Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Phillip Durdaut

Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Phillip Durdaut

Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Karl-Ragmar Riemschneider
Zweitgutachter : Prof. Dr.-Ing. Jürgen Vollmer

Abgegeben am 8. Februar 2013

Phillip Durdaut

Thema der Bachelorthesis

Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz

Stichworte

Batteriezellensensor, Sensornetz, Wakeup, drahtlose Kommunikation, Mikrocontroller

Kurzzusammenfassung

Diese Arbeit beschreibt die Konzipierung, Realisierung und die funktionale Erprobung eines Zellensensors für Fahrzeugbatterien mit drahtloser Kommunikation. Die Kommunikation verläuft im ISM-Band bei 434 MHz bidirektional und paketbasiert zwischen einer zentralen Batteriemanagementeinheit und den, in die Batteriezellen integrierten, Zellensensoren. Aus Gründen der Energieeffizienz verfügt der Zellensensor über einen zweiten, nahezu passiven Empfänger, über den eine Aktivierung (Wakeup) aller Zellensensoren aus einem Schlafzustand möglich ist.

Phillip Durdaut

Title of the paper

Cell sensor for automotive batteries with communication and wakeup-feature in the ISM band at 434 MHz

Keywords

battery cell sensor, sensor nodes, wakeup, wireless communication, microcontroller

Abstract

This paper describes the conception, the implementation and the field trial of a cell sensor for automotive batteries with wireless communication. The communication operates bidirectional and packet-based in ISM band at 434 MHz between the central battery management system and cell sensors which are integrated into the battery cells. For reasons of energy efficiency the cell sensor contains a secondary, almost passive receiver that is used for activating the cell sensors from sleep mode (wakeup).

Inhaltsverzeichnis

1 Einführung

1.1 Projektbeschreibung

Aus modernen Fahrzeugen sind Batterien bzw. Akkumulatoren nicht mehr wegzudenken. Ob als Starterbatterie in nahezu allen Fahrzeugen mit Verbrennungsmotor oder als Antriebs- bzw. Traktionsbatterie in Förderfahrzeugen, wie Gabelstaplern oder in modernen Elektroautos, kommen Batterien zum Einsatz. Diese Akkumulatoren sind dabei oft mehrzellig ausgeführt und elektrisch in Reihe geschaltet, um Klemmenspannungen erreichen zu können, die i.A. über der Spannung einer einzelnen elektrochemischen Zelle liegen. So bestehen beispielsweise die Bleiakkumulatoren mit einzelnen Zellenspannungen von $2\,\mathrm{V}$ in konventionellen Automobilen meistens aus 6 Zellen, um die $12\,\mathrm{V}$ des elektrischen Bordnetzes zu erreichen.

Zunehmend an Bedeutung gewinnen Systeme, die im Betrieb einer Batterie Daten wie Spannung, Strom und Temperatur aufzeichnen, um den Batteriebetrieb zu optimieren. Sogenannte Batteriemanagementsysteme (*BMS*) sind besonders dann lohnenswert, wenn die Kosten für einen Akkumulator einen erheblichen Teil der gesamten Kosten ausmachen [43], wie es bei modernen Elektroautos der Fall ist. Ein solches System hilft die verbleibende Reichweite eines Elektrofahrzeugs zu ermitteln und kann gegebenenfalls frühzeitig vor einem Ausfall des Akkumulators warnen.

Kommerzielle Systeme existieren bereits. Das *Sentinel* von *LEM Holding* nimmt Messdaten an jeder Zelle innerhalb einer Batterie auf und überträgt diese kabelgebunden an ein Steuergerät. Nachteilig ist dabei der große Verkabelungsaufwand, welcher zum einen hohe Kosten verursacht und zum anderen ein Sicherheitsrisiko birgt. Der *intelligente Batteriesensor* von *Hella KGaA Hueck & Co* nimmt die Messdaten direkt an den Hauptklemmen der Batterie auf. Dieser spart Verkabelungskosten, birgt jedoch den großen Nachteil, dass nicht jede Zelle einzeln überwacht wird. Ist nur eine einzelne Zelle stark geschwächt, wird dies in vielzelligen Batterien nicht erkannt, führt aber mittelfristig zu einem Defekt der Batterie. [42]

Dazu befasst sich das Forschungsprojekt *Drahtlose Zellensensoren für Fahrzeugbatterien* (*BATSEN*) der Hochschule für Angewandte Wissenschaften (*HAW*) Hamburg. Gefördert vom Bundesministerium für Bildung und Forschung und mehrerer Industriepartner besteht das Forschungsvorhaben darin, Messdaten an jeder Zelle einer Batterie aufzunehmen und drahtlos an das Steuergerät zu übertragen. Mit den gewonnenen Daten aus der Überwachung sollen der Ladezustand, der *State of Charge* (*SOC*) sowie der Gesundheitszustand, der *State of Health* (*SOH*) der Batterie ermittelt werden. Der Einsatz einer drahtlosen Kommunikation im Vergleich zu verdrahteten Sensoren bietet neben den geringeren Kosten noch viele weitere Vorteile, wobei der entscheidenste Vorteil die automatisch gegebene Potentialtrennung ist [46].

Die Abbildung 1.1 stellt das Prinzip zur Überwachung der Batteriezellen schematisch dar. Parallel zur jeder Zelle wird ein Zellensensor geschaltet, der entweder autonom oder auf Kommando (vgl. Abschnitt 1.2) Spannung und Temperatur misst, und diese an die Basisstation bzw. das Steuergerät sendet. Da aufgrund der Reihenschaltung der Strom durch die Last (z.B. Verbraucher wie Radio oder Licht im Auto) stets derselbe ist, wie der durch jede Batteriezelle fließende, kann dieser zentral im Steuergerät gemessen werden. Dieser wird benötigt, um mit den einzelnen Zellenspannungen die Impedanzen der Zellen berechnen zu können, welche zur Berechnung des *SOC* und des *SOH* beitragen. Weiterhin misst das Steuergerät die Spannung über die Last und ist zudem in das Bordcomputersystem des Fahrzeuges integriert, sodass es in der Lage ist, das Netzwerk aus Zellensensoren zu steuern.

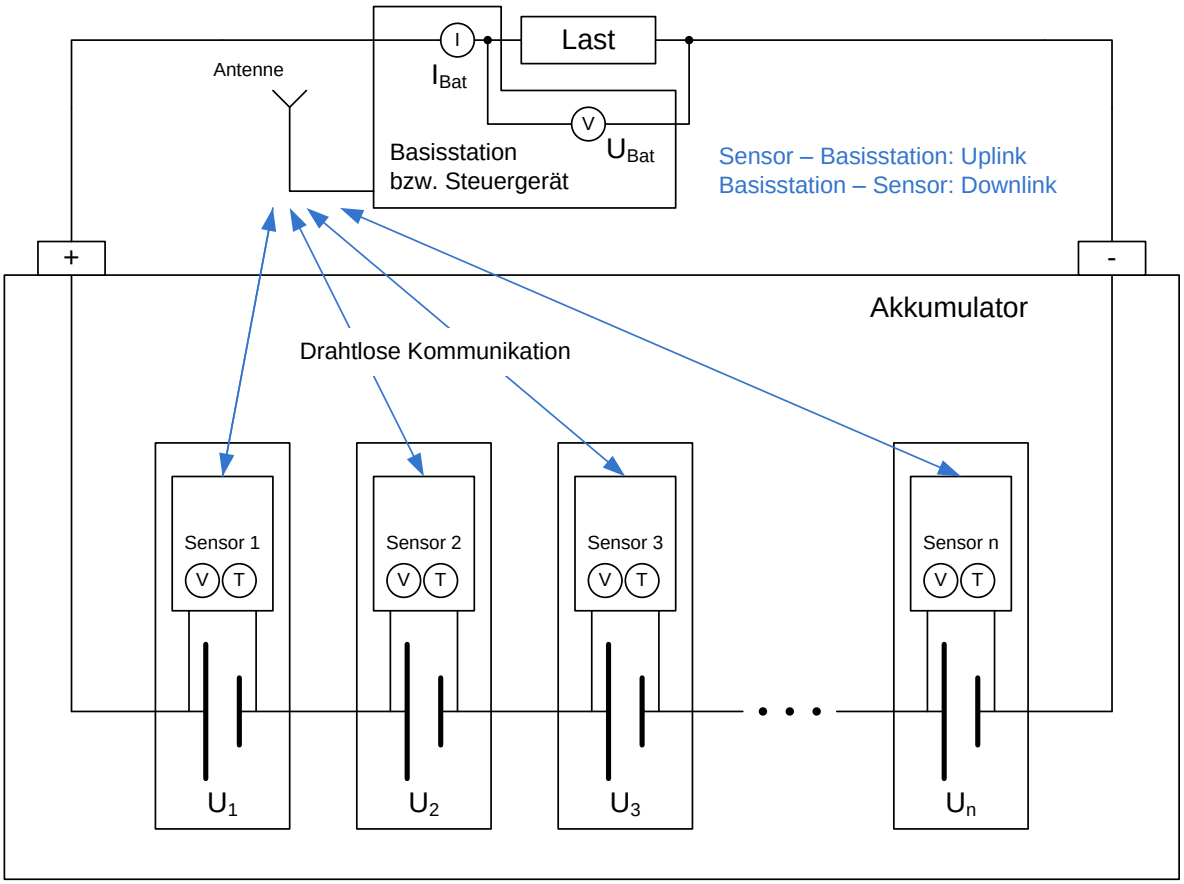


Abbildung 1.1: Prinzip der drahtlosen Zellenüberwachung nach [47]

Im Folgenden wird die Kommunikation vom Zellensensor zur Basisstation als *Uplink* und die von der Basisstation zu den Zellensensoren als *Downlink* bezeichnet.

1.2 Kategorisierung der Zellensensoren

Innerhalb des Forschungsvorhabens *BATSEN* wurden drei Klassen von Zellensensoren unterschiedlicher Komplexität definiert. Welcher Typ von Sensor sich für welchen Anwendungszweck bzw. für welchen Batterietyp eignet, wurde zum Teil bereits untersucht, ist aber immer noch aktueller Forschungsgegenstand, insbesondere, da noch keine Zellensensoren der Klasse 3 realisiert wurden. Die wichtigsten Unterscheidungsmerkmale sind in der Tabelle 1.1 zusammengefasst.

	Klasse 1	Klasse 2	Klasse 3
Übertragung zwischen Sensor und Steuergerät	Nur Uplink, kein Downlink	Uplink und Downlink mit Broadcast-Wakeup	Uplink und Downlink mit Multicast und adressierten Kommandos
Empfänger im Sensor	Kein Empfänger	Passiver Empfänger	Aktiver Empfänger
Wechsel des Sensor- und Messbetriebsmodus	Autonome Entscheidung	Teilautonome Entscheidung	Zentral kommandierte Entscheidung
Mess- und Netzorganisation	Ohne Synchronisation	Einfache zentrale Synchronisation	Komplexe paarweise Synchronisation
Balancierungseffektor in der Zelle	Dezentrale Steuerbarkeit schwierig	Bedingt möglich (dezentrale Entscheidung)	Gut möglich (zentrale Entscheidung)

Tabelle 1.1: Übersicht über die Zellensensorklassen nach [46]

Der entscheidende Unterschied liegt in der Kommunikation zwischen Sensor und Basisstation. Sensoren der Klasse 1 besitzen keinen Empfänger, können also keine Befehle von der Basisstation empfangen und entscheiden deshalb autonom, wann Messwerte übertragen werden. Im Gegensatz dazu besitzen Sensoren der Klasse 3 einen vollwertigen Downlink-Kanal, sodass eine Synchronisation des Sensornetzwerks durch die Basisstation problemlos möglich ist. Dafür benötigte aktive Empfänger besitzen allerdings einen nicht zu vernachlässigenden Strombedarf, sodass in manchen Anwendungsfällen möglichweise ein Sensor der Klasse 2 vorzuziehen ist.

Diese besitzen einen passiven Empfänger, um eine einfache zentrale Synchronisation zu ermöglichen. Abhängig von der Fähigkeit Befehle von der Basisstation empfangen zu können, sind zudem Funktionen wie das Einschalten eines Balancierungseffektors. Dabei handelt es sich um einen ein- und ausschaltbaren Lastwiderstand auf dem Zellensensor, um zusätzlich elektrische Ladung aus der Zelle zu entnehmen. Diese sogenannte *passive Ladungsbalancierung* soll dazu benutzt werden, um stärker geladene Zellen auf das Ladungsniveau schwächer geladener Zellen zu bringen, da dadurch die Lebensdauer eines mehrzelligen Akkumulators erhöht werden kann. Ein Sensor der Klasse 1 kann diese Aufgabe grundsätzlich nicht bewerkstelligen, da diesem der Zugang zu Informationen über die benachbarten Zellen fehlt.

Es wird erwartet, dass der Hardwareaufwand für Sensoren der Klasse 3 am größten sein wird, wodurch auch die Kosten für einen solchen steigen. Ein mögliches Anwendungsgebiet dieser Sensoren wären demnach nicht die relativ preisgünstigen Starterbatterien, sondern große und kostenintensive Antriebsbatterien [25].

1.3 Aktueller Stand der Zellensensorik

Bisher wurden im Projekt *BATSEN* Zellensensoren der Klasse 1 und der Klasse 2 entwickelt und erprobt.

Die Sensoren der **Klasse 1** nach Plaschke [42] und Ilgin [23] sind bewusst einfach und daher kostengünstig aufgebaut. Diese bestehen im Wesentlichen aus einem 16-Bit Low Power Mikrocontroller der *MSP430*-Familie von *Texas Instruments*, über dessen internen A/D-Umsetzer (*ADC*) und Temperatursensor die Messwerte aufgenommen werden und einem *UHF*-Transmitter, der die Daten im $433\,\mathrm{MHz}$-Band an die Basisstation überträgt. Für die Übertragung werden die Daten einer Manchestercodierung unterzogen und als *OOK*-Signal mit einer Bitrate von $10\,\mathrm{kBit/s}$ bzw. einer Symbolrate von $5\,\mathrm{kBaud/s}$ an die Basisstation übertragen. Da die Sensoren der Klasse 1 über keinen Downlink verfügen, müssen die einzelnen Zellensensoren autonom entscheiden, wann die Messdaten übertragen werden. Durch die Asynchronität kommt es vor, dass mehrere Sensoren zur selben Zeit Daten senden, sodass sich diese überlagern und für die Basisstation unbrauchbar werden. Um dieses Problem zu minimieren, wurde von Püttjer [43] ein Verfahren entwickelt, bei dem der Übertragungszeitpunkt um eine pseudozufällige Zeit variiert, um die Wahrscheinlichkeit zu maximieren, dass ein Paket fehlerfrei übertragen wird.

Sensoren der Klasse 1 wurden neben Versuchen im Labor an einer Antriebsbatterie eines Gabelstaplers der Firma *Still* [42] und an Starterbatterien verschiedener PKW in der Fahrzeughalle der *HAW Hamburg* erprobt. Auch wenn die Sensoren grundsätzlich, auch unter realitätsnahen Bedingungen, Messwerte aufnehmen und versenden, zeigten sich bei den Versuchen verschiedene Probleme. Für die Berechnung des *SOC* und des *SOH* der Starterbatterie sind Hochstromereignisse, wie der Startvorgang eines Verbrennungsmotors, von besonderer Bedeutung (vgl. Abschnitt 2.1.4). Solche Ereignisse lassen sich bei der gewählten Messrate von etwa 0.5 bis $1\,\mathrm{Hz}$ nur unvollständig erfassen. Ebenfalls von Püttjer [43] wurde ein Verfahren entwickelt, mit dem der Sensor Hochstromereignisse selbstständig erkennen kann, um die Messrate zu erhöhen

und die Daten erst nach Ende des Ereignisses zu übertragen. Dabei ist es für die Basisstation anschließend jedoch problematisch alle Messwerte den Zeitpunkten der Messwertaufnahme zuzuordnen, wobei erschwerend hinzukommt, dass aufgrund des eingesparten Quarzes für den Mikrocontroller auf dem Sensor die RC-Zeitgeber auseinanderlaufen. Zudem führt die kontinuierliche Abtastung der Spannung zur Erkennung eines Hochstromereignisses zu einem erhöhten Energiebedarf des Zellensensors.

Aktuelle Sensoren der Klasse 1 eignen sich derzeit also vor allem für die Überwachung von Batterien, bei denen die zeitliche Veränderung der Zellenspannung gering ist, diese also konstant belastet werden.

In der Masterarbeit nach Jegenhorst [25] wurde erstmals ein Zellensensor der **Klasse 2** entwickelt und erprobt. Bei diesem Sensortyp wurde erstmals ein Downlink-Kanal von der Basisstation zu den Zellensensoren realisiert. Wie bei den Sensoren der Klasse 1 besteht der Zellensensor im Kern aus einem 16-Bit Low Power Mikrocontroller der *MSP430*-Familie von *Texas Instruments*, der die Messwerte aufnimmt und an den *UHF*-Transmitter weiterleitet. Die Uplink-Kommunikation findet ebenfalls im $433\,\mathrm{MHz}$-Band, bei einer *OOK*-Modulation und einer Baudrate von $5\,\mathrm{kBaud/s}$ statt. Der Downlink-Kanal wurde über *Radio Frequency Identification* (*RFID*) bei $13.56\,\mathrm{MHz}$ realisiert. Das System ist so ausgelegt, dass zwischen dem *RFID*-Reader der Basisstation und der Downlink-Antenne des Zellensensors eine induktive Kopplung besteht, über die der Zellensensor während des Empfangs Energie aufnimmt. Diese wird dazu benutzt, den Zellensensor aus einem Ruhezustand zu aktivieren, indem der Spannungswandler eingeschaltet wird. Mit diesem Konzept kann der Ruhezustand des Zellensensors aktiviert werden, wenn dieser für längere Zeit nicht benötigt wird (z.B. Fahrzeug geparkt) und rein passiv auf den nächsten Befehl warten. Dieses System bietet den enormen Vorteil das Netzwerk aus Zellensensoren steuern zu können. Die während Hochstromereignissen gesammelten Messwerte können auf Anfrage von den Sensoren übertragen werden, sodass es nicht mehr zu Überlagerungen auf dem Funkkanal kommt. Auch das Angleichen auseinandergelaufener Zeitgeber ist in diesem System möglich. Weiterhin von Vorteil ist die Steuerbarkeit von Balancierungseffektoren auf jedem Sensor, um die Niveaus der elektrischen Ladungen in jeder Batteriezelle angleichen zu können.

Nachteilig ist allerdings der enorme Hardwareaufwand und die damit verbundenen Kosten sowie der benötigte Platzbedarf aufgrund der zwei getrennten Frequenzbänder. Sowohl die Basisstation als auch jeder Zellensensor benötigt zwei verschiedene Antennen, eine je Übertragungsrichtung. Dabei fallen Antennen im $13.56\,\mathrm{MHz}$-Band im Allgemeinen größer aus als Antennen im $433\,\mathrm{MHz}$-Band, da eine Antenne immer in einer Größenordnung der Wellenlänge liegt und diese umgekehrt proportional zur Frequenz ist. Nicht zu vernachlässigen ist zudem die hohe benötigte Sendeleistung von $750\,\mathrm{mW}$, um die Zellensensoren ansprechen zu können.

1.4 Motivation und Zielsetzung

In dieser Arbeit soll erstmalig ein Zellensensor der Klasse 3 entwickelt werden, der sich besonders durch den aktiven Empfänger von den bisherigen Sensoren unterscheidet. Die bidirektio-

nale Kommunikation soll nur noch in einem Frequenzband stattfinden, um den Platzbedarf und den Hardwareaufwand im Vergleich zu den Sensoren der Klasse 2 zu senken.

Der Schwerpunkt dieser Bachelorarbeit soll in der Entwicklung eines energieeffizienten Konzeptes und der dafür benötigten Hardware liegen. Vor der Realisierung sollen verschiedene Sensorkonzepte erarbeitet und Recherchen zu Bauteilen, insbesondere Transceiver-Bausteinen, getätigt werden. Neben Simulationen einzelner Schaltungsteile am PC sollen ebenfalls praktische Voruntersuchungen zur Machbarkeit angestellt werden. Abschließend soll ein geeigneter Nachweis der grundsätzlichen Funktion des aufgebauten Netzwerks aus Zellensensoren stattfinden (vgl. Aufgabenstellung im Anhang A).

Die Programmierung der umfangreichen Software, sowohl für die Basisstation als auch auf Seiten des Zellensensors zur Realisierung der verschiedenen Kommandostrukturen, welche den Sensor erst zu einem vollwertigen Sensor der Klasse 3 werden lassen, gehören nicht zur Aufgabe dazu. Entstehende Software soll nach Möglichkeit jedoch modular aufgebaut sein, sodass eine Wiederverwendung in späteren Arbeiten vereinfacht wird.

2 Problemanalyse und Konzeptfindung

2.1 Anforderungen an den Zellensensor

2.1.1 Platzierung und Form des Zellensensors

Grundsätzlich soll der Zellensensor für verschiedene Batterietechnologien eingesetzt werden können. Im Speziellen soll der in dieser Arbeit zu entwickelnde Zellensensor in den Akkumulatoren der Firma *ecc Repenning GmbH* eingesetzt werden können. Dabei handelt es sich um Lithium-Eisenphosphat-Batteriezellen mit einer maximalen elektrischen Ladung von $48 - 50\,\mathrm{Ah}$ [17] für den Einsatz u.a. als Traktions- und Starterbatterien [31] in Fahrzeugen.

Die Abbildung 2.1 zeigt eine solche Rundzelle im geöffneten Zustand. Sie besteht im Kern aus einem Wickel verschieden beschichteter Metallfolien, umgeben von einem zylinderförmigen Gehäuse aus Aluminium. Die Abmaßungen dieser Zelle betragen etwa $28\,\mathrm{cm}$ in der Höhe bei einem Durchmesser von $6\,\mathrm{cm}$.

Abbildung 2.1: Geöffnete Lithium-Eisenphosphat-Batteriezelle

Für die Platzierung des zu konzipierenden Zellensensors gibt es zwei Möglichkeiten. Wünschenswert ist eine Unterbringung des Zellensensors direkt innerhalb der Batteriezelle. Dies würde neben dem mechanischen Schutz des Sensors einen weiteren großen Vorteil bieten. Parallel zu dieser Bachelorarbeit wird im Projekt *BATSEN* an optischen Methoden zur Beurteilung des *SOC* und des *SOH* geforscht. Für diesen Zweck ist ein Zugang zu einer dritten Elektrode innerhalb der Batteriezelle nötig, welche somit langfristig leichter mit dem Zellensensor zu verbinden wäre. Nachteilig ist allerdings die Dämpfung des Funksignals durch die Aluminiumhülse, wodurch eine drastische Verringerung der Kommunikationsreichweite zu erwarten ist.

Alternativ könnte der Zellensensor auch außerhalb auf der Batteriezelle montiert und extern mit Elektrode und Kathode verdrahtet werden. Ob die Dämpfung durch das Gehäuse der Zelle ein Problem darstellen wird, muss im Laufe der Arbeit untersucht werden. In beiden Fällen ist es jedoch sinnvoll, den Zellensensor in runder Form mit einem Durchmesser von $6\,\mathrm{cm}$ auszulegen. So ergibt sich im Vergleich zu einer quadratischen Form eine maximale Nutzfläche für die elektrischen Komponenten und die Antenne (siehe Abschnitt 3.3).

2.1.2 Spannungsbereich

Wie in Abschnitt 2.1.1 beschrieben wird, soll der in dieser Arbeit entstehende Zellensensor in Lithium-Eisenphosphat-Batteriezellen eingesetzt werden. Diese besitzen eine Nennspannung von $3.3\,\mathrm{V}$, wobei die Ladeschlussspannung etwas höher und die Entladeschlussspannung erheblich geringer ist [31]. Der Zellensensor soll nach Möglichkeit universell gehalten werden und weitere Batterietechnologien unterstützen, die bereits von früher entstandenen Zellensensoren unterstützt wurden. In der Diplomarbeit [43] wurden häufig genutzte Akkumulatortechnologien gegenübergestellt, woraus pro Batteriezelle Nennspannungen von $1.2\,\mathrm{V}$ bis $3.8\,\mathrm{V}$ hervorgehen. Da für den Betrieb des zu entwickelnden Zellensensors eine konstante Versorgungsspannung benötigt wird, ist eine Spannungsregelung erforderlich. Die Versorgungsspannung soll $3.3\,\mathrm{V}$ betragen, wie es bei *CMOS*-Schaltkreisen verbreitet ist, sodass ein Step-up/Step-down-Wandler nötig sein wird, der eine zu hohe Eingangs- bzw. Zellenspannung ($> 3.3\,\mathrm{V}$) verkleinern- und eine zu niedrige Eingangsspannung ($< 3.3\,\mathrm{V}$) vergrößern kann.

2.1.3 Energiebedarf

Die Realisierung einer vollwertigen Downlink-Kommunikation, wie sie angestrebt wird, setzt auf jedem Zellensensor einen aktiven Empfänger voraus. Für weitere Konzeptüberlegungen muss an dieser Stelle bereits bedacht werden, dass Empfänger oftmals dauerhaft Energie verbrauchen, auch wenn kein Nutzsignal empfangen wird. Trotz der großen Kapazität von $48 - 50\,\mathrm{Ah}$, die die unter 2.1.1 beschriebenen Zellen bieten, ist es sinnvoll eine Möglichkeit zu finden, den Zellensensor möglichst energieeffizient zu gestalten. Ziel ist es, den durchschnittlichen Energiebedarf so gering zu halten, dass dieser in etwa in der Größenordnung der Selbstentladung der Batteriezelle liegt. Für $LiFePO_4$-Batterien liegt diese laut [43] unter 10 %.

2.1.4 Reaktionszeit

Die Abbildung 2.2 zeigt die Zellenspannungen der sechs Zellen einer herkömmlichen Starterbatterie während des Startvorgangs eines *Mercedes Benz Vito L* aus der Arbeit [43] (weitere Startvorgänge mit $LiFePO_4$-Starterbatterien sind in [31] zu finden). Derartige Startvorgänge sind für die spätere Bestimmung des Ladezustands (*SOC*) und des Gesundheitszustands (*SOH*) der Batterie von besonderem Interesse. Das bedeutet, dass sich der zu entwickelnde Zellensensor in diesem Zeitraum (hier etwa $0.5\,\mathrm{s}$ - $2\,\mathrm{s}$) im aktiven Zustand befinden muss, um Messwerte aufnehmen zu können.

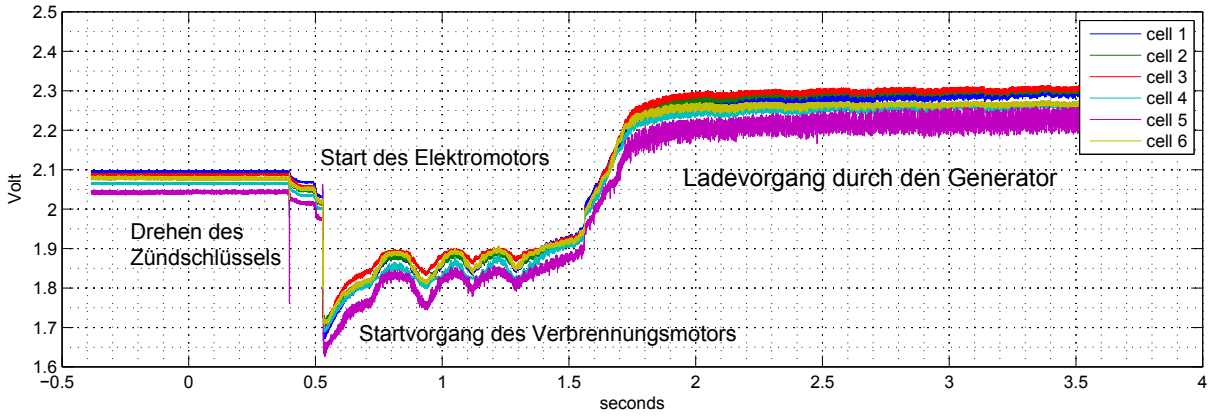


Abbildung 2.2: Zellspannungen der Starterbatterie im Startmoment eines Mercedes Benz Vito L, 4 Zylinder Diesel, $95\,\mathrm{kW}$, $2.0\,\mathrm{l}$. Entnommen und modifiziert aus [43]

Zellensensoren der Klasse 1 sind dauerhaft im aktiven Zustand und können derartige Spannungseinbrüche selbstständig erkennen (siehe Abschnitt 1.3). Zellensensoren der Klasse 3 hingegen werden dazu nicht zwangsläufig in der Lage sein. Durch die zu erwartende größere Stromaufnahme (vgl. Abschnitt 2.1.3) kann es sinnvoll sein, diese in einen Ruhezustand zu versetzen, wenn diese für längere Zeit nicht benötigt werden (z.B. Fahrzeug geparkt). In der Abbildung 2.2 ist zu erkennen, dass zwischen dem Drehen des Zündschlüssels und dem Start des Elektromotors, der den Verbrennungsmotor hochdreht und somit den interessanten Startvorgang einleitet, etwa $100\,\mathrm{ms}$ liegen. Während dieser Zeit wird über einen elektromagnetischen Schalter das Ritzel in den größeren Zahnkranz an der Schwungscheibe des Verbrennungsmotors gedrückt [11]. Auch wenn diese Zeit fahrzeugabhängig ist, werden $100\,\mathrm{ms}$ als diejenige Zeit festgelegt, in der das gesamte Netzwerk aus in dieser Arbeit zu entwickelnden Zellensensoren aus einem Ruhezustand in einen aktiven Messmodus versetzbar sein muss.

2.1.5 Messgenauigkeit und -intervall

„Für nutzbare Ladezustandsberechnungen (SOC, State of Charge) sind akkurate Messungen notwendig“[1]. Auch wenn in dem Begrif „akkurat“ keine Spezifizierung im technischen Sinne

[1] „Präzises Messen und Überwachen von Fahrzeugbatterien“, Artikel von Mike Kultgen in der *Elektronik automotive*, Ausgabe 12/2012

steckt, ist die Aussage trotzdem richtig. Für die Berechnung des *SOC* und des *SOH* werden die Impedanzen der Batteriezellen berechnet. Diese sind umso genauer, je exakter der Strom in der Basisstation und die Spannungen an den Zellen gemessen werden. Trotzdem ist es fragwürdig, ob der Einsatz eines 24-Bit Audio A/D-Umsetzers erheblich bessere Resultate als ein mikrocontrollerinterner 12-Bit A/D-Umsetzer liefern würde, da die zu messende Spannung wahrscheinlich mit Rauschen überlagert sein wird. Der Schwerpunkt dieser Arbeit liegt in der Konzeptentwicklung eines neuen Sensors mit drahtloser Kommunikation und nicht in der Messtechnik, sodass die Messgenauigkeit zweitrangig sein wird.

Da bei einer Batterie zwischen verschiedenen Zuständen, insbesondere dem Hochstrombetrieb und dem Normalbetrieb, unterschieden wird, ist eine Anpassung des Messintervalls sinnvoll. Hochstromereignisse (vgl. Abbildung 2.2) liefern aufgrund des hohen Energieumsatzes wertvolle Informationen, sodass eine hohe Abtastrate der Zellenspannung nötig ist. Sensoren der Klasse 1 sind mittlerweile in der Lage, solche Ereignisse selbstständig zu erkennen, um ihre Messrate zu erhöhen. Die zu entwickelnden Sensoren der Klasse 3 hingegen werden zentral gesteuert und können von der Basisstation über ein Broadcast-Kommando über ein solches Ereignis informiert werden (vgl. Abschnitt 2.1.4).

2.1.6 Drahtlose Kommunikation

Wie einführend bereits erwähnt, soll die Kommunikation zwischen der Basisstation und den Zellensensoren, im Gegensatz zu der des verwandten Systems der Klasse 2, in nur einem Frequenzband erfolgen. Das verringert den Hardwareaufwand auf beiden Seiten, da für den Uplink und für den Downlink jeweils dieselbe Antenne benutzt werden kann. Stattfinden soll die Kommunikation, wie bisher für jeden Uplink-Kanal, im *ISM-Band* bei $434\,\mathrm{MHz}$, da in diesem bereits viele Untersuchungen zu Antennen, zu Ausbreitungseigenschaften und zur Störfestigkeit im Projekt *BATSEN* gemacht wurden. Dieses Frequenzband wurde von der Internationalen Fernmeldeunion (*ITU*) auf den Frequenzbereichbereich von $433.05\,\mathrm{MHz}$ bis $434.79\,\mathrm{MHz}$ für die Region 1[2] festgelegt [24]. In der Bundesrepublik Deutschland ist die Nutzung dieses Frequenzbandes für Funkanwendungen geringer Reichweite (*SRD*) durch die Bundesnetzagentur geregelt. Diese legt beispielsweise fest, mit welcher maximalen Signalleistung gesendet werden darf. Im ausgewählten Frequenzband beträgt die maximale äquivalente Strahlungsleistung (ERP) $10\,\mathrm{mW}$ [13]. Dem Frequenzplan [12] der Bundesnetzagentur lässt sich entnehmen, dass dieses Frequenzband vielseitig genutzt wird, weshalb mit Störungen zu rechnen ist. Dabei handelt es sich um militärische Funkanwendungen, Amateurfunk, weiteren Funkanwendungen geringer Reichweite (*SRD*), Betriebsfunk und die Fernsteuerung von Modellen.

Die bisherigen Sensoren der Klasse 1 und Klasse 2 arbeiten im Uplink mit einer Übertragungsrate von $10\,\mathrm{kBit/s}$ bzw. $5\,\mathrm{kBaud/s}$ aufgrund der Manchestercodierung. Sensoren der Klasse 3 sollen aufgrund des insgesamt größer werdenden Datenaufkommens die Datenrate mindestens verdoppeln.

[2] Europa, Afrika, Nachfolgestaaten der UdSSR und Mongolei

In Bezug auf die Kommunikation durch das Gehäuse der Batteriezelle (vgl. Abschnitt 2.1.1) ist zu erwarten, dass die Aluminiumhülse wie ein Faradayscher Käfig wirkt und hochfrequente Wechselfelder abschirmt. Im Allgemeinen wirkt die Abschirmung auf die elektrische Komponente des elektromagnetischen Feldes stärker als auf die magnetische, sodass für den Zellensensor eine Antenne zu entwickeln ist, die stärker auf die magnetische Komponente anspricht.

2.1.7 Realisierungsaufwand

Sollte es langfristig zu einem flächendeckenden Einsatz der in diesem Forschungsprojekt entwickelten Zellensensoren kommen, wäre eine Minimierung der Herstellungskosten aus wirtschaftlicher Sicht von Interesse. Für eine solche Kosteneinsparung ist eine Integration großer Teile des Zellensensors auf einem Chip sinnvoll. In einem zweiten Forschungsprojekt *ESZ-ABS*, das ebenfalls unter der Leitung von Prof. Dr.-Ing. K.-R. Riemschneider steht, wurde erst vor kurzem erfolgreich ein Chip entwickelt und produziert, sodass bereits eine Menge Erfahrung gesammelt werden konnte und die Entwicklung eines *BATSEN-Chips* langfristig nicht unwahrscheinlich ist. Aus diesem Grund soll der zu entwickelnde Zellensensor abschließend auf eine mögliche Realisierbarkeit in dieser Hinsicht überprüft werden. Während der Entwicklung steht der funktionale Erfolg im Fokus.

2.2 Mögliche Sensorkonzepte

In diesem Abschnitt sollen verschiedene Konzepte erarbeitet werden, um einen Weg zu finden, die in 2.1 formulierten Anforderungen möglichst umfangreich umsetzen zu können.

2.2.1 Zellensensor dauerhaft im Empfangszustand

Für den Aufbau einer bidirektionalen Kommunikation im *UHF*-Band bieten verschiedene Halbleiterhersteller *Integrierte Schaltkreise*, sogenannte *Transceiver* an (vgl. Abschnitt 2.4). Die einfachste Realisierung zeigt die Abbildung 2.3. Neben dem immer nötigen Mikrocontroller, der die zentrale Steuerung des Zellensensors übernimmt, sind nur der Transceiver-Baustein, ein frequenzstabiler Quarz und eine Antenne nötig. Über digitale Steuerungsleitungen kann der Mikrocontroller den Transceiver konfigurieren, in verschiedene Zustände schalten und Empfangs- und Sendedaten austauschen. Zwischen dem Transceiver und der Antenne ist meistens eine Impedanzanpassung nötig, um die im allgemeinen komplexe Eingangsimpedanz des Transceivers auf die Eingangsimpedanz der Antenne zu transformieren, um einen maximalen Leistungsfluss zu gewährleisten.

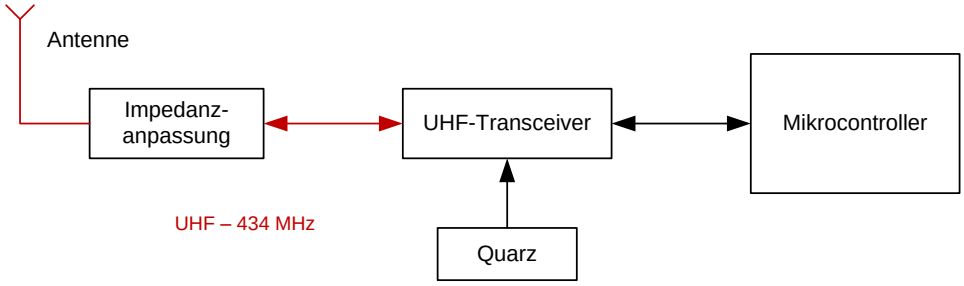


Abbildung 2.3: Konzeptidee: Zellensensor dauerhaft im Empfangszustand

Der Hauptnachteil dieses Konzepts besteht darin, dass der Transceiver nach jedem gesendeten Datensatz in den Empfangsmodus zurückkehren muss, um die nächsten Instruktionen der Basisstation empfangen zu können. Wie einführend bereits erwähnt, ist im Empfangsmodus mit einem nicht zu vernachlässigen Strombedarf zu rechnen. Um eine Abschätzung zu erhalten, in welchem Zeitraum der Zellensensor die eingangs vorgestellte Batteriezelle entladen würde, werden in diesem Abschnitt für jedes vorgestellte Konzept die Zeiträume bis zur vollständigen Entladung mit Durchschnittswerten berechnet. Dabei werden weitere Verbraucher, wie zum Beispiel ein zusätzlicher Temperatursensor, erstmal nicht mit berücksichtigt. Außerdem wird vereinfacht nur der Leerlauffall betrachtet, in dem der Sensor weder Messdaten aufnimmt noch versendet, sondern nur auf einen Befehl von der Basisstation wartet. Dieser Fall dürfte bei vielen Fahrzeugen der Normalzustand sein, wenn es zum Beispiel geparkt ist oder nachts grundsätzlich nicht benutzt wird. Während das Fahrzeug, dessen Batterie überwacht werden soll, in Bewegung ist, ist der Stromverbrauch der Zellensensoren weniger kritisch, da zumindest Starterbatterien während der Fahrt durch den Generator geladen werden. Die folgende Tabelle zeigt durchschnittliche Stromverbräuche der beiden wichtigsten Bausteine auf dem Zellensensor, die für die Energiebedarfsabschätzungen in diesem Kapitel zu Grunde gelegt werden.

Transceiver im Schlafzustand	$I_{TR,sleep}$	$1\,\mu\mathrm{A}$
Transceiver im Ruhezustand	$I_{TR,idle}$	$2\,\mathrm{mA}$
Transceiver im Empfangszustand[a]	$I_{TR,rx}$	$15\,\mathrm{mA}$
Transceiver im Sendezustand	$I_{TR,tx}$	$25\,\mathrm{mA}$
Mikrocontroller[b]	I_{MCU}	$1.5\,\mathrm{mA}$
Mikrocontroller im Schlafzustand	$I_{MCU,sleep}$	$100\,\mu\mathrm{A}$

Tabelle 2.1: Angenommene durchschnittliche Stromverbräuche der Hauptkomponenten des Zellensensors zur Abschätzung der Dauer zur vollständigen Entladung der $LiFePO_4$-Batteriezelle durch den Zellensensor

[a] vgl. Tabelle 2.3
[b] *MSP430* Ultra-Low Power 16-Bit Mikrocontroller von *Texas Instruments*

Der durchschnittliche Strombedarf I_1 des Zellensensors nach dem ersten Konzept berechnet sich zu

$$I_1 = I_{TR,rx} + I_{MCU} = 15\,\text{mA} + 1.5\,\text{mA} = 16.5\,\text{mA}.$$

Innerhalb eines Jahres benötigt ein solcher Zellensensor die folgende Energie bzw. Ladungsmenge.

$$E_1 = U_{ZS} \cdot I_{ZS} \cdot 24\,\text{h} \cdot 365 = 3.3\,\text{V} \cdot I_1 \cdot 24\,\text{h} \cdot 365 \approx 477\,\text{Wh}$$

$$Q_1 = I_1 \cdot 24\,\text{h} \cdot 365 \approx 144.5\,\text{Ah}$$

Die in Abschnitt 2.1.1 vorgestellte $50\,\text{Ah}$-Batteriezelle wäre also nach etwa 4 Monaten durch den Zellensensor entladen, wobei die Selbstentladung nicht berücksichtigt wurde.

2.2.2 Zellensensor im periodischen Wakeup-Modus

Eine Verbesserung würde sich ergeben, wenn sich der Transceiver nicht dauerhaft im Empfangsmodus befindet, sondern regelmäßig in einen Schlafzustand verfällt. Zu diesem Zweck besitzen ausgewählte Transceiver (siehe Abschnitt 2.4) eine sogenannte *WOR*-Funktion (engl. Wake On Radio). Diese erlaubt es, den Transceiver selbstständig in konfigurierbaren periodischen Abständen (Abbildung 2.4) aus dem Schlafzustand erwachen zu lassen, um in den Empfangsmodus zu wechseln.

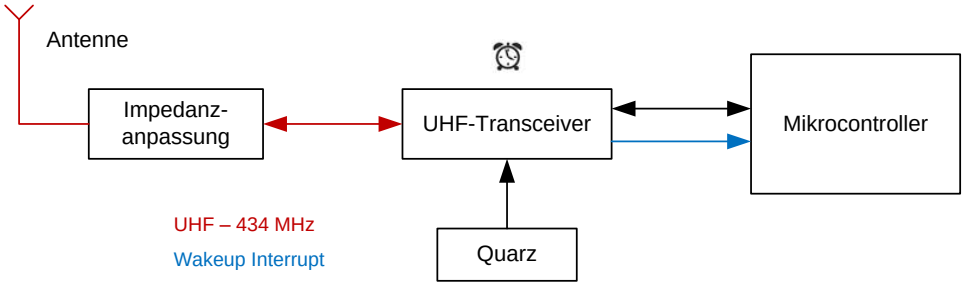


Abbildung 2.4: Konzeptidee: Zellensensor im periodischen Wakeup-Modus

Wird innerhalb einer bestimmten Zeit ein Nutzsignal erkannt, kann der Transceiver den Mikrocontroller über einen Interrupt in den aktiven Zustand versetzten, sodass der Messbetrieb des Sensors aufgenommen werden kann. Wird kein Nutzsignal empfangen, wechselt der Transceiver automatisch zurück in den Schlafzustand. Diese Möglichkeit bietet durchaus Energieeinsparungspotential, verbraucht dennoch unnötige Energie in Zeiträumen, in den denen das Fahrzeug für längere Zeit nicht bewegt wird. Wie in Abschnitt 2.1.4 gefordert, muss der Zellensensor innerhalb von $100\,\text{ms}$ dazu bereit sein Messungen aufzunehmen. Wie das Verhältnis zwischen

Schlaf- und Empfangszeiten dazu aussehen müsste, ist abhängig von dem verwendeten Transceiver, da je nach Typ individuelle Zeiten zum Beispiel zur Kalibirierung des Frequenzsynthesizers in die Rechnung einbezogen werden müssen.
Geht man von einem Duty-Cycle von 12.5 % aus, berechnet sich der durchschnittliche Strombedarf I_2 zu

$$I_2 = 0.125 \cdot I_{TR,rx} + 0.875 \cdot I_{TR,sleep} + I_{MCU,sleep}$$

bzw.

$$I_2 = 0.125 \cdot 15\,\mathrm{mA} + 0.875 \cdot 1\,\mu\mathrm{A} + 100\,\mu\mathrm{A} = 1.98\,\mathrm{mA}.$$

Innerhalb eines Jahres benötigt ein solcher Zellensensor die folgende Energie bzw. Ladungsmenge.

$$E_2 = U_{ZS} \cdot I_{ZS} \cdot 24\,\mathrm{h} \cdot 365 = 3.3\,\mathrm{V} \cdot I_2 \cdot 24\,\mathrm{h} \cdot 365 \approx 57.2\,\mathrm{Wh}$$

$$Q_2 = I_2 \cdot 24\,\mathrm{h} \cdot 365 \approx 17.3\,\mathrm{Ah}$$

Die in Abschnitt 2.1.1 vorgestellte 50 Ah-Batteriezelle wäre dann nach fast 3 Jahren durch den Zellensensor entladen (Selbstentladung nicht berücksichtigt).

2.2.3 Wakeup mit RF Power Detector

Die drei folgenden Konzeptideen sehen jeweils einen zweiten Empfangszweig auf dem Zellensensor für das Aktivieren des Transceivers und des Mikrocontrollers aus dem Schlafzustand vor. So soll der Zellensensor nur bei Bedarf in einen aktiven Zustand versetzt werden können und nur einen minimalen Energiebdarf besitzen, wenn dieser keine Messungen tätigen soll.

Für die Realisierung ist dazu stets ein sogenannter *UHF*- oder auch Antennenumschalter (engl. RF switch) nötig, der das von der Antenne empfangene Signal entsprechend entweder an den Transceiver oder an die Wakeup-Schaltung weiterleitet. Dadurch entsteht ein erweiterter Hardwareaufwand (vgl. Abbildung 2.5). Eine Parallelschaltung von Antennenausgang und den Eingängen des Transceivers und der Wakeup-Schaltung ist aufgrund der im Allgemeinen unterschiedlichen Eingangsimpedanzen nicht möglich bzw. sinnvoll. Aus diesem Grund sind auch zusätzliche Netzwerke zur Impedanzanpassung nötig.

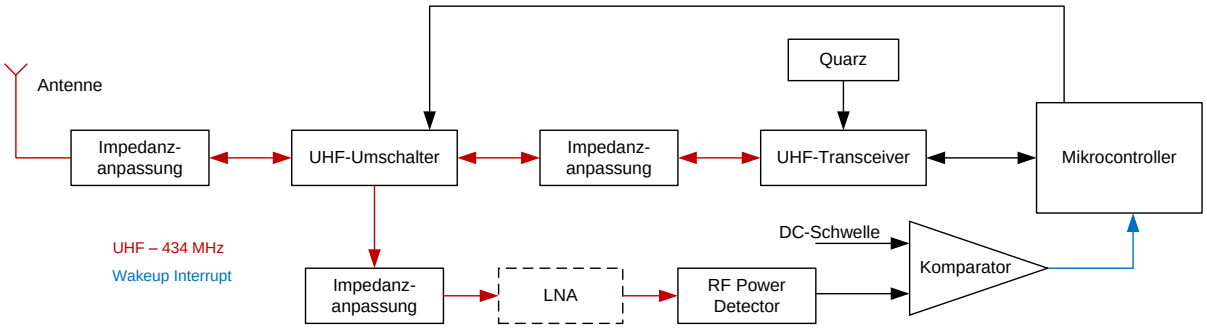


Abbildung 2.5: Konzeptidee: Wakeup mit RF Power Detector

Die Wakeup-Schaltung in Abbildung 2.5 besteht im Wesentlichen aus einem *RF Power Detector*. Als solche bezeichnete Schaltkreise detektieren das eingehende Ultra-Hochfrequenzsignal und erzeugen an ihrem Ausgang eine Spannung, die proportional zur Leistung des Eingangssignals ist. Diese Spannung könnte in Verbindung mit einem einfachen Komparator einen Interrupt generieren, der den Mikrocontroller aus dem Schlafzustand in einen aktiven Zustand versetzt, welcher wiederum den Transceiver aktiviert. Ein möglicher Baustein wäre der *LTC5507* von *Linear Technology* [29]. Dieser besitzt einen Strombedarf von $550\,\mu\mathrm{A}$ bei einer Eingangsempfindlichkeit von $-34\,\mathrm{dBm}$. Die Empfindlichkeit ist im Vergleich zu gängigen Transceivern (Abschnitt 2.4) im Bereich von $-100\,\mathrm{dBm}$ recht gering. Zur Erhöhung der Empfindlichkeit, insbesondere im Hinblick auf den Einsatz des Zellensensors innerhalb der Batteriezelle, wäre ein zusätzlicher *LNA* als Vorverstärker denkbar. Die Recherche ergab allerdings, dass diese einen erheblichen Strombedarf von etwa $25\,\mathrm{mA}$ besitzen, weshalb diese Möglichkeit nicht weiter verfolgt wurde. Der Betriebsstrom eines Komparators liegt bei vernachlässigbaren $650\,\mathrm{nA}$ (z.B. *MAX9119* von *Maxim* [32]). Bei Inkaufnahme der geringen Empfindlichkeit berechnet sich der durchschnittliche Strombedarf I_3 zu

$$I_3 = I_{LTC5507} + I_{MAX9119} + I_{TR,sleep} + I_{MCU,sleep}$$

bzw.

$$I_3 = 550\,\mu\mathrm{A} + 650\,\mathrm{nA} + 1\,\mu\mathrm{A} + 100\,\mu\mathrm{A} = 651.65\,\mu\mathrm{A}.$$

Innerhalb eines Jahres benötigt ein solcher Zellensensor die folgende Energie bzw. Ladungsmenge.

$$E_3 = U_{ZS} \cdot I_{ZS} \cdot 24\,\mathrm{h} \cdot 365 = 3.3\,\mathrm{V} \cdot I_3 \cdot 24\,\mathrm{h} \cdot 365 \approx 18.8\,\mathrm{Wh}$$

$$Q_3 = I_3 \cdot 24\,\mathrm{h} \cdot 365 \approx 5.7\,\mathrm{Ah}$$

Die in Abschnitt 2.1.1 vorgestellte 50 Ah-Batteriezelle wäre dann nach etwa 8,5 Jahren durch den Zellensensor entladen (Selbstentladung nicht berücksichtigt).

2.2.4 Wakeup mit Spannungsvervielfachung

Eine weitere Konzeptidee ist durch das *Energy Harvesting* (wörtl. übersetzt Energie-Ernten) inspiriert entstanden. Dabei handelt es sich um die Gewinnung elektrischer Energie aus der Umgebung, zum Beispiel aus elektromagnetischer Strahlung [16] [38]. Dazu werden Spannungsvervielfacherschaltungen bzw. Ladungspumpen (meistens Villard-Schaltungen) eingesetzt, die hochfrequente Signale (z.B. im 869 MHz- aber auch im 433 MHz-Band) gleichrichten, und die erzeugte Gleichspannung beispielsweise für die Versorgung eines Sensors zur Verfügung stellen. Weiterentwicklungen nutzen diese Vorgehensweise in Verbindung mit einem Komparator bereits als Wakeup-Schaltung in drahtlosen Sensornetzwerken [4] [20]. Die Abbildung 2.6 zeigt eine mehrstufige Spannungsvervielfacherschaltung, wie sie in [4] genutzt wurde, um drahtlose Sensoren aus einem Niedrigenergie-Zustand mit einem modulierten Trägersignal im 869 MHz-Band aufzuwecken. Diese ist nötig, da die Spannung an der Antenne schon durch die Freiraumdämpfung des eintreffenden Signals im Allgemeinen sehr klein ist und somit keine Pegel erreichen kann, wie sie von *TTL*- oder *CMOS*-Digitalschaltungen benötigt werden.

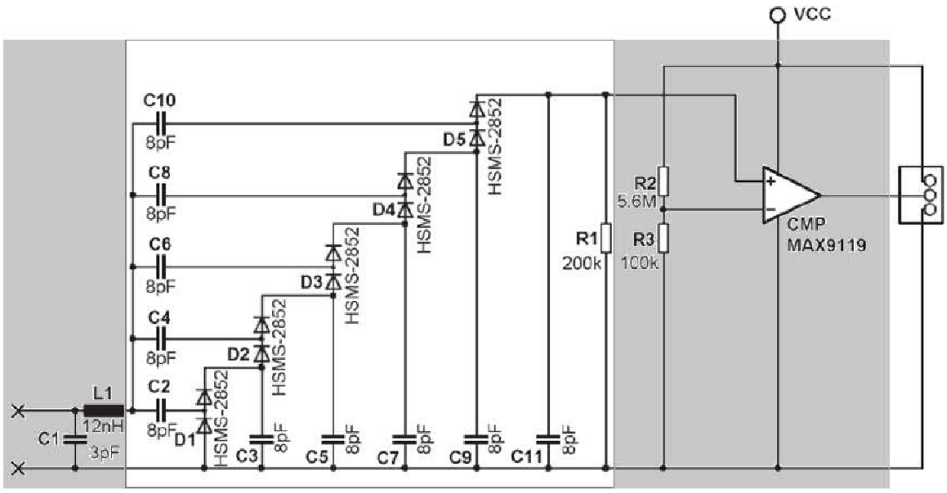


Abbildung 2.6: Fünfstufige Villard-Spannungsvervielfacherschaltung mit Komparator als Wakeup-Schaltung. Entnommen aus [4].

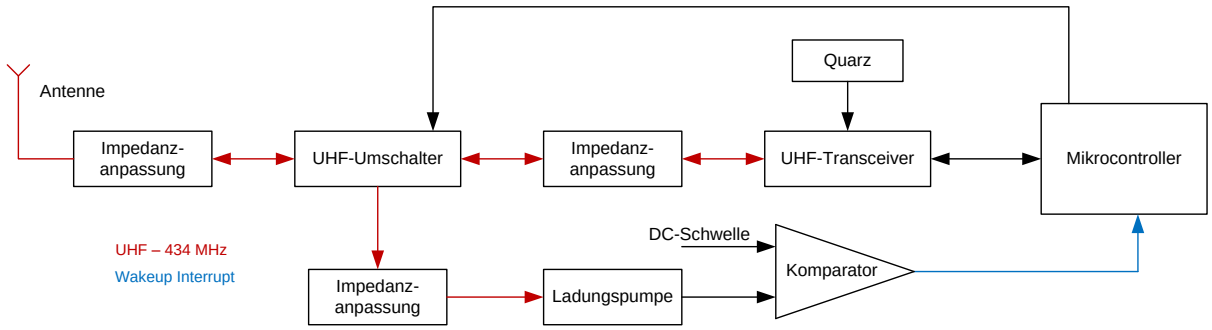


Abbildung 2.7: Konzeptidee: Wakeup mit Spannungsvervielfachung

Vorteilhaft ist, dass diese Art von Wakeup-Schaltung, bis auf den Komparator, passiv arbeitet, und somit keinerlei Energie verbraucht, während sich der Sensor im Schlafzustand befindet und auf das Wakeupsignal gewartet wird. Nachteilig hingegen ist, dass an den für die Gleichrichtung benötigten Dioden stets eine Mindestvorwärtsspannung anliegen muss, damit diese leitend werden. Deshalb wurden hier spezielle Schottky-Dioden eingesetzt, die eine gerine Flußspannung von nur etwa $200\,\mathrm{mV}$ benötigen [8]. Zudem ergeben sich Spannungsverluste aufgrund der Flußspannungen an den Dioden jeder Stufe. Somit muss vor dem Einsatz einer solchen Schaltung genauer evaluiert werden, unter welchen Bedingungen mit welcher Eingangsleistung gerechnet werden kann. Wird davon ausgegangen, dass sich der Mikrocontroller im Schlafzustand befindet und durch einen Interrupt durch den Komparator *MAX9119* [32] geweckt wird, berechnet sich der durchschnittliche Strombedarf I_4 zu

$$I_4 = I_{MAX9119} + I_{TR,sleep} + I_{MCU,sleep}$$

bzw.

$$I_4 = 650\,\mathrm{nA} + 1\,\mu\mathrm{A} + 100\,\mu\mathrm{A} = 101.65\,\mu\mathrm{A}.$$

Innerhalb eines Jahres benötigt ein solcher Zellensensor die folgende Energie bzw. Ladungsmenge.

$$E_4 = U_{ZS} \cdot I_{ZS} \cdot 24\,\mathrm{h} \cdot 365 = 3.3\,\mathrm{V} \cdot I_4 \cdot 24\,\mathrm{h} \cdot 365 \approx 2.94\,\mathrm{Wh}$$

$$Q_4 = I_4 \cdot 24\,\mathrm{h} \cdot 365 \approx 0.9\,\mathrm{Ah}$$

Die in Abschnitt 2.1.1 vorgestellte $50\,\mathrm{Ah}$-Batteriezelle wäre dann nach etwa 56 Jahren durch den Zellensensor entladen (Selbstentladung nicht berücksichtigt).

2.2.5 Wakeup mit Hüllkurvendemodulator und LF Wakeup Receiver

Eine Weiterentwicklung des vorherigen Konzepts ist in [19] beschrieben. Anstelle einer Kaskade aus Spannungsvervielfacherschaltungen wird bei dieser Lösung lediglich eine Gleichrichterschaltung in Verbindung mit einem Tiefpassfilter eingesetzt, um das Empfangssignal einer Amplitudendemodulation zu unterziehen. Nach einer anschließenden Bandpassfilterung erfolgt die weitere Auswertung des wesentlich niederfrequenteren Signals durch einen *LF Wakeup Receiver*-Baustein von *austriamicrosystems* [6].

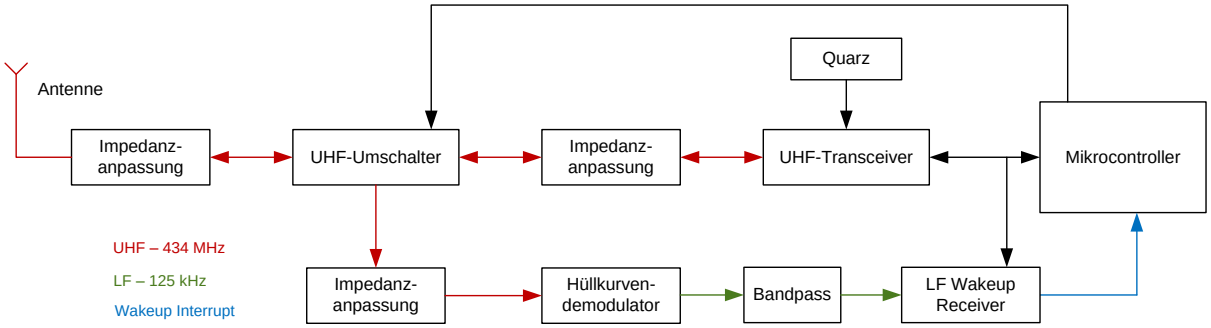


Abbildung 2.8: Konzeptidee: Wakeup mit Hüllkurvendetektor und LF Wakeup Receiver

Dieser Baustein erlaubt in dessen einfachsten Betriebsmodus die Erkennung eines *LF*-Trägersignals zwischen $110\,\mathrm{kHz}$ und $150\,\mathrm{kHz}$. Sobald ein Trägersignal erkannt wird, erzeugt der Chip ein Wakeup-Interrupt, um Microcontroller und Transceiver aus deren Schlafzuständen wecken zu können. Dabei benötigt der LF Wakeup Receiver lediglich einen geringen Betriebsstrom von $2.7\,\mu\mathrm{A}$. Der Hüllkurvendemodulator, bestehend aus Gleichrichter und Tiefpassfilter, sowie das Bandpassfilter arbeiten passiv. Weiterhin von Vorteil ist die hohe Empfindlichkeit des LF Wakeup Receivers von $-113\,\mathrm{dBm}$ sowie die Möglichkeit, ein optionales Wakeup-Protokoll implementieren zu können. Dieses könnte dazu verwendet werden, Zellensensoren während der Wakeup-Phase zu addressieren, indem Daten entsprechend des Wakeup-Protokolls in das *UHF*- bzw. das *LF*-Signal kodiert werden.

Wird davon ausgegangen, dass sich der Mikrocontroller und der Transceiver im Schlafzustand befinden, berechnet sich der durchschnittliche Strombedarf I_5 zu

$$I_5 = I_{AS3930} + I_{TR,sleep} + I_{MCU,sleep}$$

bzw.

$$I_5 = 2.7\,\mu\mathrm{A} + 100\,\mu\mathrm{A} + 1\,\mu\mathrm{A} = 103.7\,\mu\mathrm{A}.$$

Innerhalb eines Jahres benötigt ein solcher Zellensensor die folgende Energie bzw. Ladungsmenge.

$$E_5 = U_{ZS} \cdot I_{ZS} \cdot 24\,\mathrm{h} \cdot 365 = 3.3\,\mathrm{V} \cdot I_5 \cdot 24\,\mathrm{h} \cdot 365 \approx 3\,\mathrm{Wh}$$

$$Q_5 = I_5 \cdot 24\,\mathrm{h} \cdot 365 \approx 0.91\,\mathrm{Ah}$$

Die in Abschnitt 2.1.1 vorgestellte $50\,\mathrm{Ah}$-Batteriezelle wäre dann nach etwa 55 Jahren durch den Zellensensor entladen (Selbstentladung nicht berücksichtigt).

2.3 Vergleich der Konzepte

Um entscheiden zu können, welches Konzept praktisch umgesetzt und erprobt werden soll, wurde die Tabelle 2.2 erstellt. In dieser werden die fünf Konzeptvorschläge anhand verschiedener Kriterien miteinander verglichen.

Das Hauptaugenmerk liegt dabei auf der Energieeffizienz und der Reaktionsgeschwindigkeit des Zellensensors, um die Anforderungen einhalten zu können. Erstere wurde jeweils in Abschnitt 2.2 berechnet, wobei klare Unterschiede festzustellen waren. Da die meiste Energie im wartenden Betrieb des Sensors verloren geht, schneiden die Konzepte mit zusätzlicher Wakeup-Schaltung besser ab, als die beiden ersten. Dieser Vorteil muss allerdings mit einem erhöhten Hardwareaufwand erkauft werden, welcher sich jedoch im Rahmen befindet und vertretbar wäre. Eine weitere wichtige Eigenschaft ist die Empfindlichkeit des Zellensensors. Bei den Konzepten 1 und 2 existiert kein zusätzlicher Empfangszweig, sodass die hohe Empfindlichkeit eines Transceiver-Bausteins (vgl. Abschnitt 2.4) ausgenutzt werden kann. Dieses könnte von großem Vorteil im Hinblick auf die Kommunikation durch die Aluminiumhülse der Rundzelle (vgl. Abschnitt 2.1.1) sein. Aus diesem Grund scheidet das Konzept 3 mit der geringen Empfindlichkeit von nur $-34\,\mathrm{dBm}$ aus. Die Empfindlichkeit der Wakeup-Schaltungen in den Konzepten 4 und 5 sind ohne Voruntersuchungen zu diesem Zeitpunkt noch nicht mit Sicherheit bezifferbar. In beiden Fällen kommen bzw. könnten Dioden zum Einsatz kommen, mit denen sich laut Datenblatt einer verbreiteten Schottky-Diode [8] diskrete Mikrowellenempfänger mit einer Empfindlichkeit von $-57\,\mathrm{dBm}$ aufbauen lassen.

	Konzept	Empfindlichkeit	Reaktions-geschwindigkeit	Wakeup-Adressierung	Hardware-aufwand	Energie-effizienz	Ungewollte Wakeups
Zellensensor dauerhaft im Empfangszustand	1	++	++	entfällt	++	--	entfällt
Zellensensor im periodischen Wakeup-Modus	2	++	--	-	++	o	+
Wakeup mit RF Power Detector	3	--	++	-	-	+	--
Wakeup mit Spannungsvervielfachung	4	+	++	-	--	++	--
Wakeup mit Hüllkurvendemodulator und LF Wakeup Receiver	5	+	++	+	--	++	++

Tabelle 2.2: Vergleich der vorgestellten Zellensensorkonzepte hinsichtlich des Energiebedarfs und der Reaktionszeit

Somit fallen die Konzepte 4 und 5 in die engere Auswahl, wobei auch das Konzept 2, trotz des mittleren Energiebedarfs, seine Vorzüge bietet. Weil der Hardwareaufwand zwischen Konzept 4 und 5 vergleichbar ist, der Einsatz des LF Wakeup Receivers jedoch zusätzlich den großen Vorteil einer Wakeup-Adressierung bietet bzw. ungewollte Wakeups vermeiden kann, soll das Konzept 5 realisiert und erprobt werden. Außerdem ist es jedoch möglich mit einem Zellensensor nach Konzept 5 das Konzept 2 zu erproben, da dafür lediglich Softwareanpassungen nötig sind. So könnten die Konzepte 2 und 5 in späteren Versuchen direkt miteinander verglichen werden.

2.4 Recherche geeigneter Transceiver

Der Begriff Transceiver setzt sich zusammen aus den englischen Begriffen *transmit* und *receive*. Ein solcher Baustein besitzt also die Fähigkeit Daten zu senden und Daten empfangen zu können. Für den zu entwickelnden Zellensensor wird ein *UHF*-Transceiver benötigt, der Daten über eine Antenne empfangen und umgekehrt Daten über diese abstrahlen kann, um mit der Basisstation bidirektional kommunizieren zu können. Ein Transceiver enthält zu diesem Zweck alle wichtigen Bausteine eines digitalen Übertragungssystems wie Filter, Mischer und Verstärker. Für die Steuerung und den Austausch von Daten steht ein digitales Interface zur Verfügung, an das beispielsweise ein Mikrocontroller angeschlossen werden kann.

In der Tabelle 2.3 werden einige dieser Bausteine von unterschiedlichen Herstellern anhand verschiedener Kriterien verglichen. Allen Transceivern gemeinsam ist ein möglicher Betrieb bei einer Versorgungsspannung von $3.3\,\mathrm{V}$ und ein komfortables *Packet Handling* in Verbindung mit internen *FIFO*-Ringspeichern, sodass ein- und ausgehende Daten über eine serielle Schnittstelle ausgetauscht werden können. Praktisch daran ist, dass ein Mikrocontroller die aktuelle Programmausführung nicht unterbrechen muss, wenn Daten empfangen werden, sondern diese zu einem passenden Zeitpunkt anfragen kann. Bei den bisherigen Systemen bestand diese Möglichkeit nicht (siehe Abschnitt 3.1). Alle hier aufgelisteten Transceiver besitzen einen stromsparenden Ruhezustand, welcher gemäß des gewählten Konzeptes entscheidend ist. Zudem bieten alle Transceiver die Möglichkeit ein Taktsignal bereitzustellen, das direkt von dem sehr präzisen Quarz abgeleitet wird und dazu verwendet werden kann, den eher ungenauen RC-Oszillator des Mikrocontrollers zu korrigieren.

Aufgrund der aktiven Empfangseinheit besitzen alle Transceiver eine hohe Empfindlichkeit, von denen keine kritisch sein dürfte, wenn mit einer Signalleistung von etwa $10\,\mathrm{dBm}$ gesendet wird. Den für die eventuelle Realisierung des Konzeptes 2 (Abschnitt 2.2.2) notwendigen Wakeup-Timer besitzen mehrere Transceiver allerdings nicht. Aufgrund der hohen möglichen Datenrate, mit der auch das Wakeup-Signal in der Basisstation erzeugt werden könnte und der geringen Stromaufnahme während des Empfangs, fiel die Entscheidung auf den *SPIRIT1* von *ST Microelectronics*. Es stellte sich jeoch heraus, dass dieser erst 2013 lieferbar sein wird. Aufgrund des geringeren Preises als der des *CC1101* von *Texas Instruments* wurde deshalb entschieden, dass der *Si4431* von *Silicon Labs* auf dem Zellensensor verwendet werden soll. Dieser ermöglicht eine vierfach höhere Datenrate als die bisherigen Sensoren, ein automatisches *Quarz-Tuning*, und eine Ausgangsleistung von $13\,\mathrm{dBm}$ ($20\,\mathrm{mW}$). Sollte sich im Anbetracht der Kommunikation durch die Aluminiumhülse der Rundzelle (vgl. Abschnitt 2.1.1) zeigen, dass ein Empfang der Daten des Zellensensors außerhalb der Zelle nicht möglich sein sollte, wäre der *Si4431* aufgrund der Pinkompatibilität ohne Probleme gegen einen *Si4432* austauschbar, welcher eine Ausgangsleistung von bis zu $20\,\mathrm{dBm}$ bietet.

In Abschnitt 2.1.7 wurde bereits erwähnt, dass langfristig eine Integration vieler der auf dem Zellensensor enthaltenen elektronischen Komponenten auf einem Chip anzustreben ist. Quarze, die für die Erzeugung einer frequenzgenauen Trägerschwingung nötig sind, lassen sich aufgrund der internen Mikromechanik nicht integrieren und sind zudem kostenintensiv. Aus

diesem Grund wurde sowohl auf neueren Sensoren der Klasse 1 [23], als auch auf den Sensoren der Klasse 2 [25] ein Transmitter-*IC* verwendet, das aufgrund eines präzisen internen LC-Oszillators keinen externen Quarz benötigt. Diese quarzfreien Transmitter sind bislang nur von dem Hersteller *Silicon Labs* erhältlich. Quarzfreie Transceiver wiederum existieren zwar noch nicht, doch bei der Wahl des *Si4431* besteht der Hintergedanke, dass langfristig ein quarzfreier Transceiver in das Angebot des Herstellers aufgenommen wird und ein Austausch dann einfacher würde. Wahrscheinlich könnten große Teile der Software wiederverwendet werden, da sich die digitale Ansteuerung von Bausteinen einer Bauteilfamilie eines Herstellers meistens stark ähnelt.

Transceiver	Wakeup-Timer (*WOR*)	*OOK*	*FSK*	Max. Datenrate (*OOK*) in kbps	Max. Sendeleistung (*OOK*) in dBm	Empfindlichkeit (*OOK*) in dBm	Stromaufnahme beim Empfang in mA	Stromaufnahme beim Senden in mA	SAW-Filter benötigt	Packet Handling	Manuelles Senden
ATA5428	ja	ja	ja	**10**	10	-112.5	10.5	10.5 (10 dBm)	nein	ja	ja
ADF7020	**nein**	ja	ja	64	10	-106.5	19	26.8 (10 dBm)	nein	ja	**k.A.**
ADF7020-1	**nein**	ja	ja	64	13	-111.8	17.6	21 (10 dBm)	nein	ja	**k.A.**
nRF905	**nein**	**nein**	ja	50	10	-100	12.5	30 (10 dBm)	nein	ja	**k.A.**
TRC105	**nein**	ja	ja	32	13	-112	2.7	25 (10 dBm)	**ja**	ja	ja
Si4431	ja	ja	ja	40	13	-102	18.5	30 (13 dBm)	nein	ja	ja
Si4432	ja	ja	ja	40	20	-102	18.5	85 (20 dBm)	nein	ja	ja
CC1101	ja	ja	ja	250	10	-95	15	29.2 (10 dBm)	nein	ja	ja
SPIRIT1	ja	ja	ja	250	11	-87	9	19.5 (11 dBm)	nein	ja	ja

Tabelle 2.3: Vergleich einiger derzeit auf dem Markt befindlicher Transceiver für das 434 MHz-Band

2.5 Wakeup-Schaltung

In diesem Abschnitt soll das zu realisierende Konzept aus Abschnitt 2.2.5, bzw. insbesondere das benötigte Wakeup-Signal und dessen Verarbeitung auf den Zellensensoren, näher beschrieben werden. Auf der Abbildung 2.8 wurde bereits das Blockschaltbild der gesamten bidirektionalen Kommunikationseinheit gezeigt, welche sich auf jedem Zellensensor befinden wird. Im Folgenden wird lediglich der zum Transceiver parallele Empfangszweig behandelt, die Wakeup-Schaltung, welche den Mikrocontroller aus dessen Schlafzustand aufwecken kann.

2.5.1 Funktionsprinzip

Kern der Wakeup-Schaltung ist ein integrierter Schaltkreis mit der Bezeichnung „AS3930 - LF Wakeup Receiver“, welche von *austriamicrosystems* [5] entwickelt und vertrieben wird. Wie der Name bereits vermuten lässt, arbeitet dieser Baustein mit einem Eingangssignal im *LF*- bzw. Langwellen-Bereich. Laut Datenblatt liegt der Eingangsfrequenzbereich von $110\,\mathrm{kHz}$ bis $150\,\mathrm{kHz}$, sodass an dieser Stelle $125\,\mathrm{kHz}$ als Zielfrequenz festgelegt werden. Am Ausgang stellt der *AS3930* ein für den Mikrocontroller digitales Eingangssignal zu Verfügung, das von diesem als Interrupt genutzt werden soll. Der Pegel dieser Wakeup-Leitung ändert sich abhängig vom Betriebszustand des LF Wakeup Receivers.

Im einfacheren Betriebszustand wird eine reine Trägererkennung des $125\,\mathrm{kHz}$-Sinussignals durchgeführt. Sobald dieses Trägersignal länger als $550\,\mu\mathrm{s}$ [5] detektiert wird, wird auf der Wakeup-Leitung eine positive Flanke erzeugt.

Im zweiten Betriebszustand wird das Eingangssignal als digital-amplitudenmoduliertes Signal (*OOK*) mit der Trägerfrequenz $125\,\mathrm{kHz}$ und einer konfigurierbaren ($1024\,\mathrm{Bit/s}$ - $8192\,\mathrm{Bit/s}$) Bitdauer interpretiert. In diesem Modus muss das LF-Wakeup-Signal nach einem Carrier-Burst zusätzlich ein Muster enthalten, das dem im Chip konfigurierten Muster entsprechen muss, damit auf der Wakeup-Leitung ein Pegelwechsel erzeugt wird. Dieses Verfahren bietet großen Schutz vor ungewollten Wakeups. Da dieses Verfahren in Software nachrüstbar ist, soll es erst zum Einsatz kommen, falls sich zeigen sollte, dass es bei der reinen Trägererkennung häufig zu ungewollten Wakeups kommt.

Da die bidirektionale Kommunikation mit der Basisstation in einem Frequenzband stattfinden soll, damit es jeweils nur einer Antenne bedarf, muss eine Signalverarbeitung stattfinden, die das Wakeup-Signal aus dem $434\,\mathrm{MHz}$-Band so verarbeitet, dass das geforderte $125\,\mathrm{kHz}$-Signal entsteht. Beschrieben wird diese in [19] für ein Wakeup-Signal im $868\,\mathrm{MHz}$-Band. Die Abbildungen 2.9 und 2.10 sollen die Funktionsweise der in dieser Arbeit entstehende Wakeup-Schaltung prinzipiell erläutern.

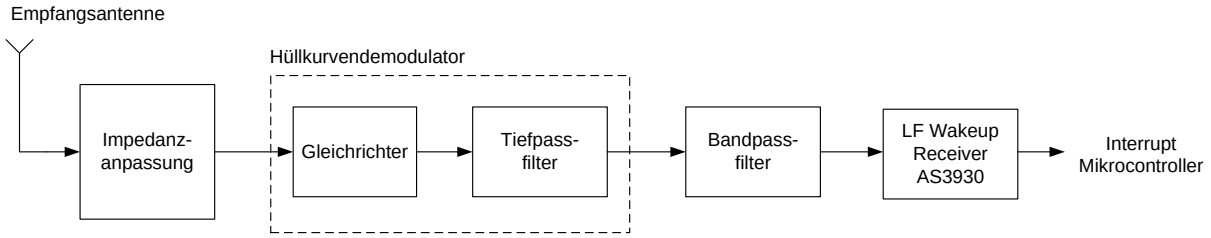


Abbildung 2.9: Blockschaltbild der Wakeup-Schaltung

Das von der Basisstation abgestrahlte Wakeup-Signal (grau) ist zweistufig digital amplitudenmoduliert (*OOK*-Signal) mit einer Bitdauer von $4\,\mu$s. Im Wechsel wird das $434\,$MHz-Trägersignal ein- und abgeschaltet. Dieses Signal wird von allen Zellensensoren des Netzwerks empfangen und auf einen Hüllkurvendemodulator geleitet. Dieser besteht aus einem Gleichrichter und einem Tiefpassfilter und verarbeitet das Empfangssignal so, dass am Ausgang die Hüllkurve des *OOK*-Signals erscheint. Dieses ideale und periodische Rechtecksignal (rot) mit einem Tastgrad von 50 % und einer Periodendauer von $8\,\mu$s wird anschließend geeignet bandpassgefiltert, um ein Sinussignal gleicher Periodendauer bzw. mit der Frequenz $125\,$kHz zu erhalten (blau). Dieses Signal kann dann anschließend von dem LF Wakeup Receiver weiterverarbeitet werden.

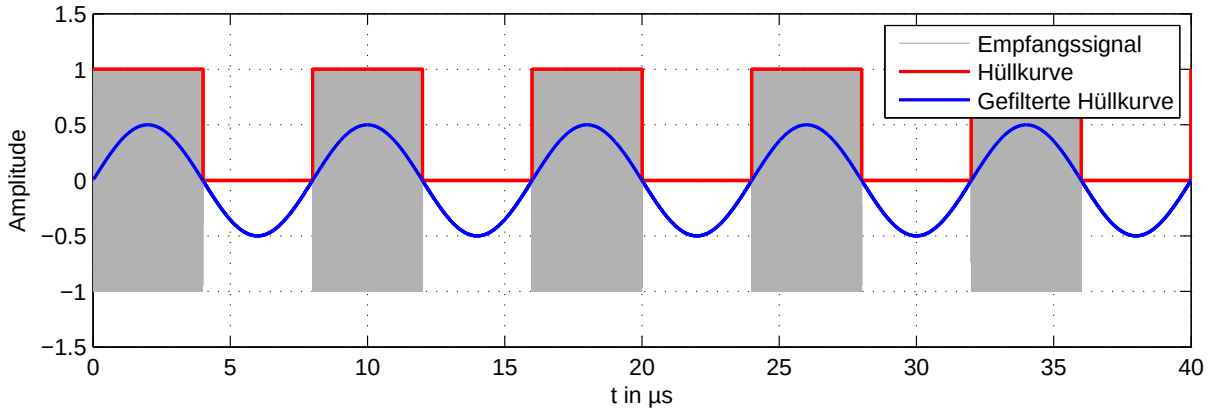


Abbildung 2.10: Signale in der Wakeupschaltung

Im nächsten Kapitel müssen Untersuchungen angestellt werden, in welcher Weise sich die Filter, der Gleichrichter und insbesondere die Impedanzanpassung in Hardware auf dem Zellensensor realisieren lassen.

2.5.2 Theoretische Beschreibung

Ziel ist in diesem Abschnitt die Berechnung des Frequenz- bzw. des Leistungsdichtespektrums des Wakeup-Signals, um später das gemessene Leistungsdichtespektrum bewerten zu können.

Mathematisch betrachtet setzt sich das Wakeup-Signal $s(t)$ multiplikativ aus der Trägerschwingung $s_T(t)$ und einem periodischen Rechtecksignal $r(t)$ zusammen.

$$s(t) = s_T(t) \cdot r(t)$$

Es gelten die folgenden Festlegungen:

- Frequenz der Trägerschwingung $f_T = 434\,\mathrm{MHz}$
- Frequenz des Modulationssignals $f_0 = 125\,\mathrm{kHz}$
- Bitdauer $T_0 = \frac{1}{f_0} = 8\,\mu\mathrm{s}$
- Amplituden beider Signale vereinfacht zu 1 angenommen

Bei dem Trägersignal handelt es sich um eine harmonische Schwingung der Form

$$s_T(t) = sin(2\pi f_T t).$$

Das Rechtecksignal lässt sich durch eine Fourierreihe beschreiben [41].

$$r(t) = \frac{1}{2} + \frac{2}{\pi} \cdot \sum_{n=1}^{\infty} \frac{1}{2n-1} \cdot sin((2n-1) \cdot 2\pi f_0 t)$$

Zur Berechnung des Frequenzspektrums muss das Zeitsignal $s(t)$ mit Hilfe der Fourier-Transformation in den Frequenzbereich transformiert werden.

$$\underline{S}(f) = \mathfrak{F}\left\{\frac{1}{2} \cdot s_T(t) + \frac{2}{\pi} \cdot s_T(t) \cdot \sum_{n=1}^{\infty} \frac{1}{2n-1} \cdot sin((2n-1) \cdot 2\pi f_0 t)\right\}$$

Gemäß dem Linearitätssatz der Fourier-Transformation darf gliedweise transformiert werden. Zur Erhöhung der Übersichtlichkeit werden zwei neue komplexe Funktionen $\underline{A}(f)$ und $\underline{B}(f)$ eingeführt.

$$\underline{S}(f) = \underline{A}(f) + \underline{B}(f)$$

$$\underline{A}(f) = \mathfrak{F}\left\{\frac{1}{2} \cdot sin(2\pi f_T t)\right\} = \frac{j}{4}(\delta(f + f_T) - \delta(f - f_T))$$

$$\underline{B}(f) = \mathfrak{F}\left\{\frac{2}{\pi} \cdot sin(2\pi f_T t) \cdot \sum_{n=1}^{\infty} \frac{1}{2n-1} \cdot sin((2n-1) \cdot 2\pi f_0 t)\right\}$$

Die Berechnung von $\underline{A}(f)$ konnte direkt in diesem Schritt geschehen, da eine einzelne harmonische Schwingung lediglich aus zwei diskreten Spektrallinien besteht, welche durch die Diracsche Deltafunktion $\delta(f)$ beschrieben werden.

Zur Lösung von $\underline{B}(f)$ werden die Sinusschwingungen gemäß den Produktregeln für trigonometrische Funktionen [41] miteinander multipliziert, sodass sich nur noch gerade Kosinus-Aufbauschwingungen ergeben, welche leicht zu transformieren sind.

$$\underline{B}(f) = \frac{1}{\pi} \cdot \mathfrak{F} \left\{ \sum_{n=1}^{\infty} \frac{1}{2n-1} \left(cos(2\pi(f_T - (2n-1)f_0)t) - cos(2\pi(f_T + (2n-1)f_0)t) \right) \right\}$$

$$\begin{aligned} \underline{B}(f) = \frac{1}{\pi} \sum_{n=1}^{\infty} \frac{1}{4n-2} (&\delta(f + (f_T - (2n-1)f_0)) + \delta(f - (f_T - (2n-1)f_0)) \\ &-\delta(f + (f_T + (2n-1)f_0)) - \delta(f - (f_T + (2n-1)f_0))) \end{aligned}$$

Die Abbildung 2.11 zeigt das berechnete Leistungsdichtespektrum des Wakeup-Signals. Da es sich um ein periodisches Signal handelt, sind nur diskrete Spektrallinien vorhanden, die sich in einem Abstand von $250\,\mathrm{kHz}$ befinden. Die dargestellte Leistung ist auf $1\,\Omega$ normiert.

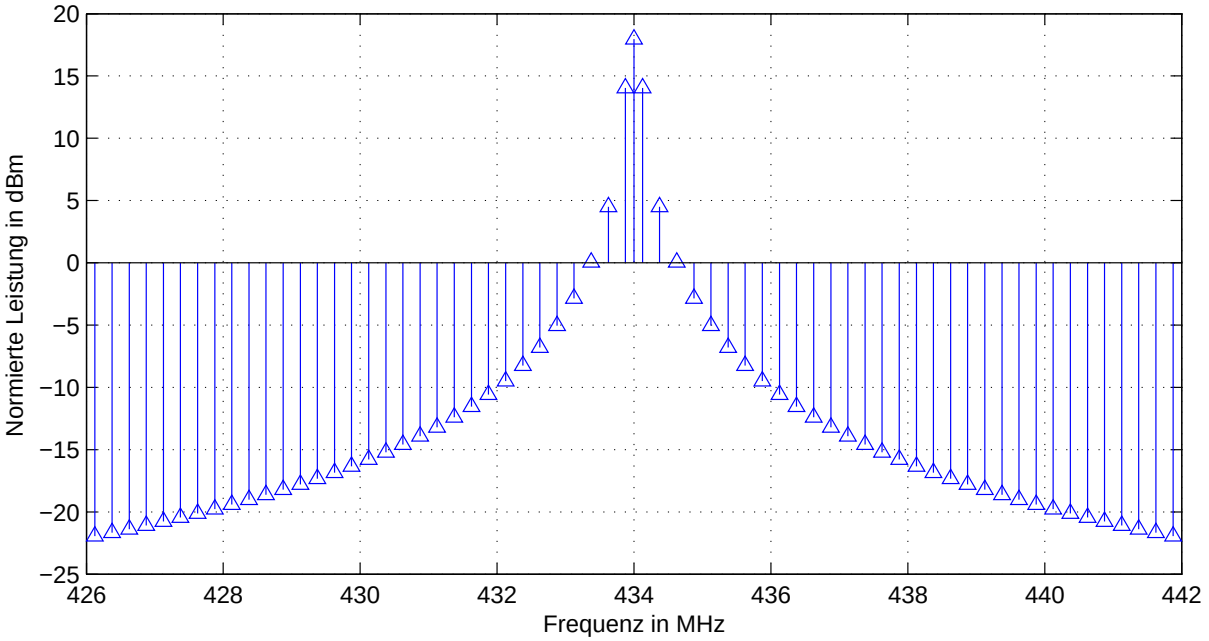


Abbildung 2.11: Theoretisch berechnetes Leistungsdichtespektrum des Wakeup-Signals

Die Nachbarkanalbeeinflussung ist aufgrund der hohen Datenrate von Interesse. Bei einem gewählten Dynamikumfang des Signals von $40\,\mathrm{dB}$ beträgt die benötigte Bandbreite $16\,\mathrm{MHz}$, geht

also weit über das *ISM-Band* hinaus (vgl. Abschnitt 2.1.6). Anders betrachtet (vgl. dazu Tabelle 2.4) ist jedoch festzustellen, dass mehr als 96.7 % der spektralen Leistung innerhalb des ISM-Bandes liegen und eine Beeinflussung anderer Funkdienste entsprechend gering ist. Sollte sich in der Praxis dennoch eine zu hohe Beeinflussung zeigen, könnte diese durch zusätzliche Abschirmungsmaßnahmen beseitigt werden, sollte das Signal durch die Karosserie des Fahrzeugs nicht ohnehin schon ausreichend nach außen hin bedämpft sein.

Bandbreite	Leistungsanteil
125 kHz	50 %
375 kHz	90.5 %
875 kHz	95 %
1.375 MHz	96.7 %
5.375 MHz	99.1 %

Tabelle 2.4: Verteilung der spektralen Leistung des Wakeup-Signals

Eine weitere Möglichkeit wäre die steilen Flanken des *OOK*-Signals gegen rampenförmige Steigungen zu ersetzen. Steile Flanken entsprechen gemäß der Fourier-Theorie immer höheren Frequenzanteilen, sodass ein trapezförmiges *OOK*-Signal diese vermeiden würde, ohne dabei den Frequenzbereich um den Träger herum entscheidend zu beeinflussen. Da in der Realität ohnehin Einschwingzeiten bzw. nichtideale Eigenschaften der senderseitigen Filter eine Rolle spielen, ist in der Praxis keine problematische Beeinflussung benachbarter Frequenzbänder zu erwarten.

3 Praktische Voruntersuchungen

3.1 Bestehende Basisstationen

Bisher wurden im Forschungsprojekt *BATSEN* zwei verschiedene Arten von Sensoren erprobt, die Sensoren der Klasse 1 und die der Klasse 2. Durch die sich unterscheidenden Sensorkonzepte wurden auch verschiedene Basisstationen nötig, welche im Folgenden kurz beschrieben werden.

Beide Basisstationen basieren auf dem Entwicklungsboard *MSP430-169STK* des Herstellers *OLIMEX Ltd*. Dieses Entwicklungsboard ist mit einem 16-Bit Mikrocontroller der *MSP430-Serie* von *Texas Instruments* (*MSP430F169*) und weiterer Hardware bestückt[40]:

- 16 x 2 LC-Display
- 64 MB nichtflüchtiger NAND-Flash-Speicher
- RS-232-Schnittstelle
- *SPI*-Schnittstelle auf Steckerleiste
- 3 Taster und 2 *LEDs*
- *JTAG*-Schnittstelle

Beide Systeme benutzen für den Datenempfang auf dem Uplink-Kanal das Receiver-Modul *DR5100* des Herstellers *RFM* [44], welches sich auf einer Adapterplatine für die Anbindung an das Entwicklungsboard befindet (BCM Rx-Modul 03/2008 [42] bzw. BCM Rx-Modul v0.2 - 05/2011 [25]).

Die Uplink-Kommunikation findet im 434 MHz-Band statt, wobei das Antennensignal digital amplitudenumgetastet ist (2-*ASK* bzw. *OOK*) und die Daten vor der Abstrahlung einer Manchester-Kodierung unterzogen wurden. Die Dekodierung des Signals wird in der Basisstation komplett von dem Mikrocontroller durchgeführt, da das Receiver-Modul *DR5100* an einem digitalen Ausgangspin lediglich das digitale Äquivalent des zu diesem Zeitpunkt empfangenen *OOK*-Signals zur Verfügung stellt.

Weiterhin wurde für die Basisstation umfangreiche Software entwickelt, die es erlaubt, die empfangenen Messdaten im Flash-Speicher abzulegen und auf Befehl über die serielle RS-232-Schnittstelle an einen Terminal-PC für die weitere Auswertung zu übertragen.

Die in Verbindung mit dem Receiver-Modul verwendete Dipolantenne ist auf der Abbildung 3.1 abgebildet.

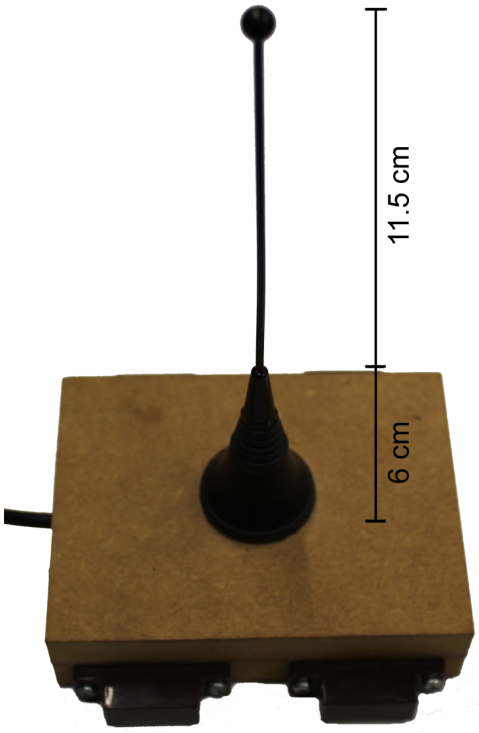


Abbildung 3.1: 434 MHz Antenne der Basisstation mit einer Eingangsimpedanz von 50 Ω

3.1.1 Weiterentwickelte Basisstation für Zellensensoren der Klasse 2

In der Masterarbeit [25] wurden neben der Entwicklung der Zellensensoren der Klasse 2 umfangreiche Hardwareerweiterungen an der Basisstation vorgenommen. Neben dem *UHF*-Empfänger für den Uplink-Kanal hat diese einen *HF*-Sender für die Frequenz 13.56 MHz, um Kommandos auf dem Downlink-Kanal versenden, bzw. Zellensensoren der Klasse 2 aus einem energiesparenden Zustand in einen aktiven Zustand (Wakeup) versetzen zu können. Dieser *RFID*-Reader ist auf einer extra Platine (BS Reader 13.56MHz v0.3 - 06/2011) untergebracht und wird zusätzlich an eine gedruckte *PCB*-Antenne angebunden (BS Antenna v0.1 - 03/2011).

3.2 Anpassung der Basisstation

Wie in Abschnitt 2.1.6 gefordert, muss die Basisstation nicht nur im *UHF*-Band bei 434 MHz empfangen, sondern auch senden können. Zudem muss die Basisstation die Fähigkeit besitzen,

das unter 2.5 beschriebene Wakeup-Signal für das Wakeup der Zellensensoren erzeugen zu können.

In einem ersten Versuch wurde der 433.92 MHz Hybrid Transceiver *TR3000* [45] des Herstellers *RFM* erprobt, da dieser im Labor vorrätig war und direkt mit der Adapterplatine „BCM Rx-Modul v0.2 - 05/2011“ nach [42] und [25] an das Entwicklungsboard angebunden werden konnte (Abbildung 3.2).

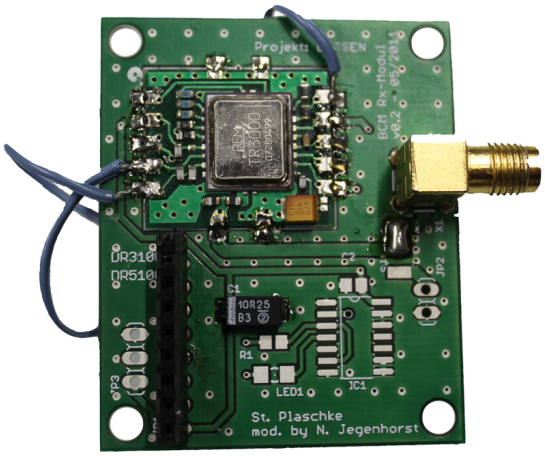


Abbildung 3.2: Adapterplatine mit Transceiver TR3000 von *RFM*

Es zeigte sich, dass das Senden eines *OOK*-Signals grundsätzlich und ohne weitere Konfigurationen problemlos möglich ist. Als kritisch erwies sich jedoch, dass der Transceiver weit außerhalb seiner Spezifikation betrieben werden musste, um das Wakeup-Signal zu erzeugen. Wie unter 2.5 dargestellt, muss die Frequenz des Modulationssignals 125 kHz betragen, was einer Datenrate von 250 kBaud entspricht. Laut Datenblatt unterstützt der *TR3000* Transmissionsraten bis zu 115.2 kBaud, sodass das Wakeup-Signal entsprechend stark verzerrt ist (vgl. Abbildung 3.4). Hinzu kommt, dass der *TR3000* nach demselben Prinzip wie auch das vorher verwendete Receiver-Modul *DR5100* [44] arbeitet. Das heisst, dass keinerlei Paketverarbeitung in Hardware unterstützt wird und die empfangenen Daten direkt vom Mikrocontroller verarbeitet werden müssen. Auch wenn die bestehende Software der Basisstation dies bereits unterstützt, ist es für zukünftige Anpassungen und Erweiterungen doch eher unkompfortabel. Auch die mögliche Ausgangsleistung von maximal 0 dBm ist vergleichsweise (vgl. Abschnitt 2.4) niedrig, wenn man bedenkt, dass das Funksignal optimalerweise die Aluminiumhülse der Batteriezelle durchdringen soll. Aus diesen Gründen wurde entschieden, ein neues und moderneres Transceiver-Modul für die Anbindung an die Basisstation zu entwickeln.

3.2.1 Transceiver-Modul

Die Wahl eines geeigneten Transceivers für die Basisstation mit Hilfe der Tabelle 2.3 fiel auf den *CC1101* von *Texas Instruments* [66], da nur dieser die benötigte Datenrate von 250 kbps unterstützt (*SPIRIT1* zu diesem Zeitpunkt noch nicht lieferbar). Dieser Transceiver unterstützt den

kontinuierlichen Sendemodus, in dem ein Trägersignal gesendet wird, sobald ein *GPIO*-Pin des Transceivers auf logisch 1 gebracht wird und das Senden unterbrochen wird, sobald dieser Pin wieder auf logisch 0 gezogen wird. In diesem Modus lässt sich das benötigte Wakeup-Signal (siehe Abschnitt 2.5) erzeugen. Außerdem unterstützt dieser Transceiver die automatische Abwicklung von ein- und ausgehenden Datenpaketen und ist laut Datenblatt zudem kompatibel zu den Datenpaketen des *Silicon Labs Si4431* [52] Transceivers, welcher auf den Zellensensoren zum Einsatz kommen soll.

Das Transceiver-Modul ist so gestaltet, dass es (wie die Transmitter-Module bisher auch) über Buchsenleisten an das *MSP430-169STK* Entwicklungsboard von OLIMEX Ltd. [40] angesteckt werden kann, das bisher als Kernmodul der Basisstation diente. Über diese Verbindungen erhält der Transceiver die $3.3\,\mathrm{V}$ Spannungsversorgung und der Mikrocontroller auf dem Entwicklungsboard Informationen über den Status des Transceivers über mehrere *GPIO*-Leitungen. Darüberhinaus wird der Transceiver über die Steckverbindung mit dem *SPI*-Bus des Mikrocontrollers verbunden. Über diese serielle Schnittstelle wird zum einen der Empfangspuffer sowie der Sendepuffer des Transceivers gelesen bzw. beschrieben und zum anderen die Register des Transceivers konfiguriert.

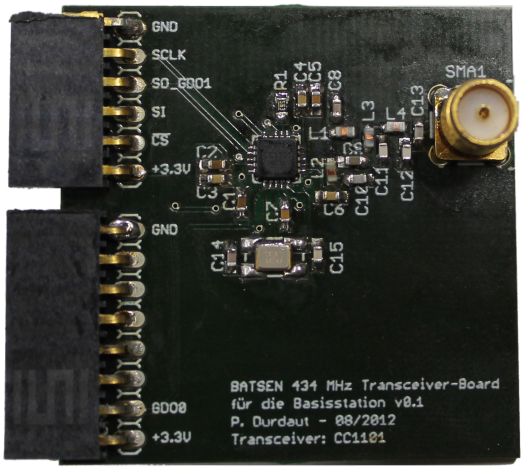


Abbildung 3.3: Transceiver-Modul mit *CC1101* für die Basisstation

Die Abbildung 3.3 zeigt den ersten funktionsfähigen Prototyp des Transceiver-Moduls. Neben dem Transceiver-*IC* und dem Quarz erkennt man lediglich einige passive Bauteile. Darunter sind Koppelkondensatoren, ein Symmetrierglied (Balun) sowie die Impedanzanpassung des Transceivers an die $50\,\Omega$-Basisstationsantenne (Abbildung 3.1), die über die zu sehende *SMA*-Buchse verbunden wird. Bei der Erstellung der Schaltung wurde sich eng an die Vorgaben des Referenzdesigns[1] und an die des Datenblatts [66] gehalten. Der Schaltplan und das Platinenlayout sind im Anhang B.4 und C.4 zu finden.

[1] http://www.ti.com/litv/zip/swrr046a

3.2.1.1 Bauelemente

Auf dem Transceiver-Modul wurden die folgenden elektrischen Bauelemente verwendet:

- *UHF*-Transceiver *CC1101* von *Texas Instruments* [66]
- 26 MHz Quarz *C3E-26.000-12-1010-X* von *Aker Technology* [1]
- *SMD*-Induktivitäten der *0603HP-Serie* von *Coilcraft* [14]
- Standard Chip-Kondensatoren und Chip-Widerstände in *SMD*-Bauform (0603)

3.2.2 Wakeup-Signal im Vergleich

3.2.2.1 Zeitbereich

Nach der ersten Inbetriebnahme des Transceiver-Moduls wurde das in Abschnitt 2.5 beschriebene Wakeup-Signal erneut erzeugt, um es mit dem des vorher bereits gemessenen Signals des 433.92 MHz Hybrid Transceiver *TR3000* von *RFM* vergleichen zu können. Das Ergebnis ist der folgenden Abbildung 3.4 zu entnehmen. Wie erhofft, gleicht das neue Sendesignal viel eher einem *OOK*-Signal. Die An- und Abstiegszeiten sind aufgrund der jetzt eingehaltenen Spezifikation sehr viel geringer.

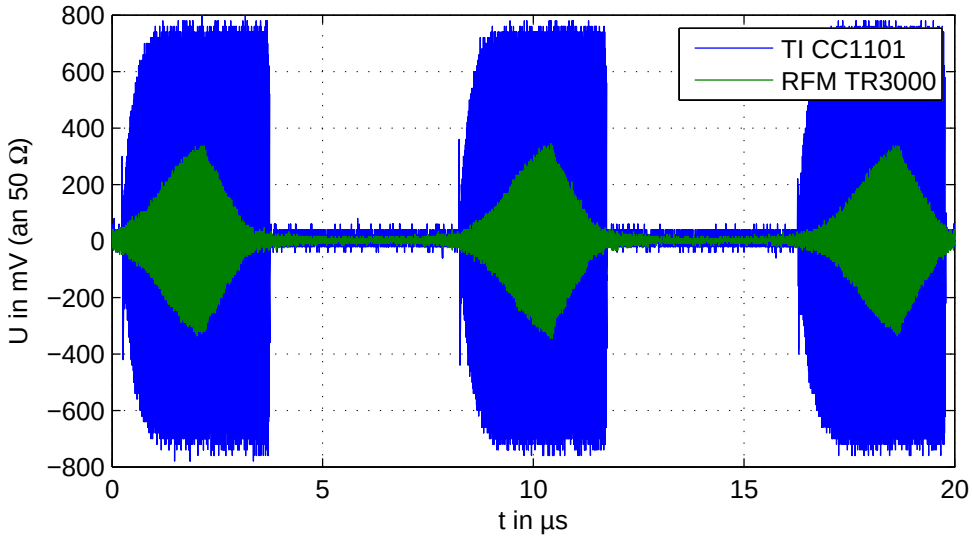


Abbildung 3.4: Vergleich der Wakeup-Signale der verschiedenen Transceiver

Vorteilhaft ist zudem die höhere Ausgangsleistung, was sich an der größeren Ausgangsspannung des Trägersignals erkennbar macht. Eine abgelesene Amplitude des Trägersignals von etwa 750 mV entspricht an einem Widerstand von 50 Ω einem Leistungspegel von

$$10 \cdot log\left(\frac{(750\,\mathrm{mV})^2}{2 \cdot 50\,\Omega} \cdot \frac{1}{1\,\mathrm{mW}}\right) = 7.5\,\mathrm{dBm}.$$

Da der Transceiver auf seine maximale Sendeleistung von $10\,\mathrm{dBm}$ konfiguriert war, und ebenfalls Leistung auf weitere spektrale Komponenten fällt, bedeutet dies in erster Schätzung, dass die Impedanzanpassung zwischen Transceiver und Antenne mit den vorgeschlagenen Bauteilwerten aus dem Datenblatt erfolgreich war.

3.2.2.2 Frequenzbereich

Um beurteilen zu können, ob das Trägersignal auf der gewünschten Frequenz $434\,\mathrm{MHz}$ schwingt, ist eine Analyse des Wakeup-Signals im Frequenzbereich notwendig. Mit Hilfe eines Spektrumanalysators wurde das Leistungsdichtespektrum gemessen und in Abbildung 3.5 dargestellt (blau). Dabei wurde diejenige Bandbreite dargestellt, in der sich 96.7 % der spektralen Leistung des Wakeup-Signals befindet (Tabelle 2.4). Für einen direkten Vergleich mit dem vorberechneten Leistungsdichtespektrum (rot) aus Abschnitt 2.5 wurde ein Offset von etwa $43\,\mathrm{dB}$ addiert. Gut zu erkennen ist, dass die gemessenen Frequenzlinien exakt auf die Vorausberechneten fallen und auch die Verhältnisse der Pegel recht gut übereinstimmen. Zudem liegt die Frequenz der Trägerschwingung genau bei den gewünschten $434\,\mathrm{MHz}$. Abweichend von der Berechnung sind allerdings zusätzliche Frequenzlinien zwischen den Solllinien. Diese haben mehrere Ursachen. Zum einen liegt das Verhältnis zwischen *on*- und *off*-Zustand des *OOK*-Signals nicht exakt bei 50 %. Zum anderen entstehen diese Frequenzlinien durch Nichtlinearitäten in der Mischerstufe des Transceivers bzw. sind Intermodulationsprodukte, die durch den Ausgangsverstärker des Transceivers erzeugt werden.

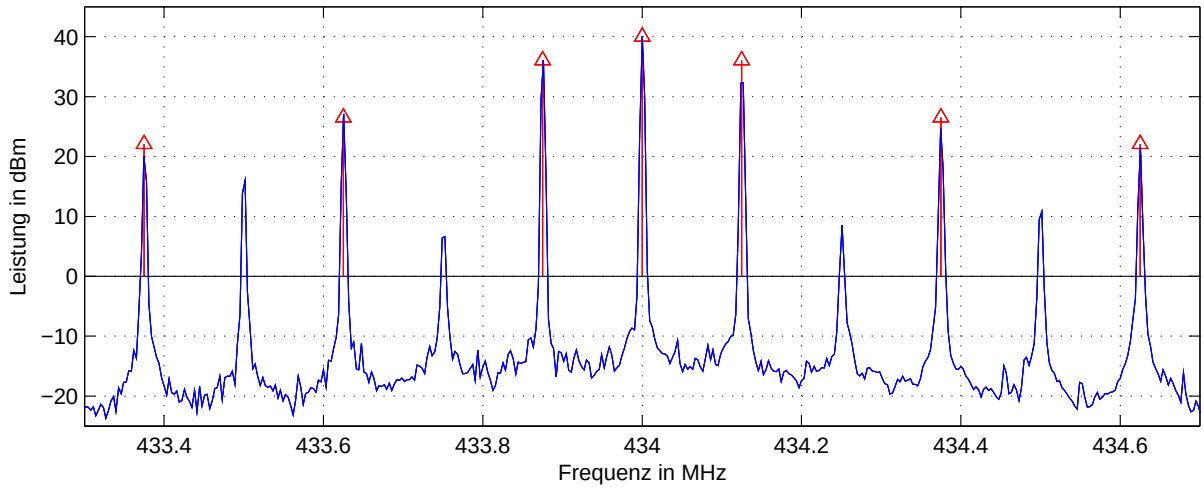


Abbildung 3.5: Vergleich des vorberechneten Leistungsdichtespektrums des Wakeup-Signals mit der realen Messung

3.3 Entwurf und Erprobung einer PCB-Antenne

Sowohl für den Empfang des Wakeup-Signals als auch für die bidirektionale Kommunikation über den Transceiver soll auf dem Zellensensor über dieselbe Antenne bei $434\,\mathrm{MHz}$ erfolgen. Als äußerst kostengünstig und leicht reproduzierbar gelten *PCB*-Antennen, welche im Wesentlichen aus Leiterbahnen auf einer Platine bestehen.

Grundsätzlich gilt, dass der Empfang einer Antenne umso besser ist, je größer die Fläche dieser ist. Deshalb soll eine Antenne maximal möglicher Fläche entworfen werden. Wie unter 2.1.1 dargestellt, wird der Zellensensor eine runde Bauform besitzen. Da auf diesem viele elektrische Bauteile unterzubringen sind, scheiden rechteckförmige Antennen, wie auf den Zellensensoren der Klasse 1, oder Antennen mit vielen Windungen bzw. eine spiralförmige Antenne aus.

Die beste Alternative bietet eine kreisrunde Schleifenantenne, die am Rand der runden Sensor-Platine verläuft. Schleifenantennen in dieser Größenordnung (Durchmesser des Zellensensors $6\,\mathrm{cm}$) mit einem Umfang von bis zu etwa $\lambda/4$ (bei $434\,\mathrm{MHz}$ beträgt die Wellenlänge $\lambda = 69\,\mathrm{cm}$) bezeichnet man als *magnetische Antennen* oder auch *kleine Schleifenantennen*. Der Name rührt daher, dass der Impedanzverlauf einer *magnetische Antenne* mit steigener Frequenz induktiv verläuft und die Antenne stärker auf die magnetische Komponente des elektromagnetischen Feldes anspricht [28]. Dieser Umstand ist in diesem Anwendungsfall von besonderer Bedeutung, da eine Funkkommunikation durch die Aluminiumhülse der Rundzelle stattfinden soll. Auf die elektrische Komponente des elektromagnetischen Feldes wirkt das Aluminium eher dämpfend, als auf die magnetische Komponente (vgl. Abschnitt 2.1.6).

3.3.1 Dimensionierung

Mit Hilfe der Applikationsanweisungen [58] und [34] wird in diesem Abschnitt eine kleine Schleifenantenne dimensioniert. Die folgende Abbildung 3.6 zeigt die zu dimensionierenden geometrischen und elektrischen Parameter der Antenne, die geeignet festzulegen sind, um eine Resonanz bei $434\,\mathrm{MHz}$ zu erhalten.

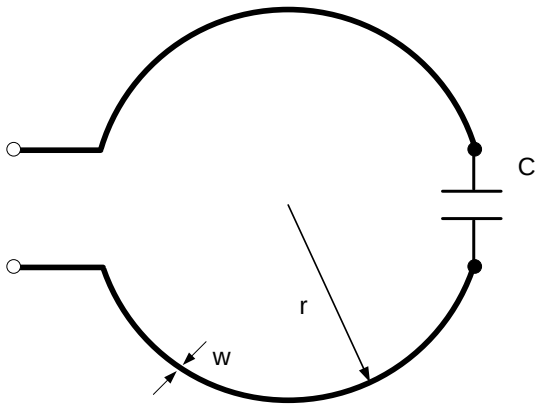


Abbildung 3.6: Geometrie einer kleinen Schleifenantenne

Dazu gehört der Radius r, die Breite der Leiterbahn w sowie die Kapazität des Kondensators C. Der Radius der Antenne wird an dieser Stelle auf $r = 2.5\,\mathrm{cm}$ festgelegt, um am Rand des Zellensensors mit dem Durchmesser $6\,\mathrm{cm}$ noch einen Ring von etwa $5\,\mathrm{mm}$ frei zu lassen, damit Wechselwirkungen mit der Aluminiumhülse minimiert werden.

Die Abbildung 3.7 zeigt das elektrische Ersatzschaltbild der Schleifenantenne. Dieses besteht aus dem Strahlungs- bzw. dem Radiationswiderstand R_r, der Schleifeninduktivität L, dem ohmschen Verlustwiderstand R_{loss} sowie der diskreten Kapazität C.

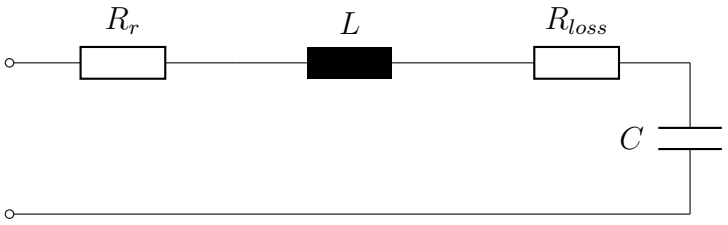


Abbildung 3.7: Ersatzschaltbild einer kleinen Schleifenantenne

Gemeinsam bilden diese Elemente einen elektrischen Serienschwingkreis. Die Resonanzfrequenz f_0 in einem solchen berechnet sich wie folgt [21].

$$f_0 = \frac{1}{2\pi\sqrt{LC}} \tag{3.1}$$

Zur Berechnung der Ersatzparameter gelten die folgenden Gleichungen.[2]

$$R_r = \frac{A^2}{\lambda^4} \cdot 320\pi^4 \cdot 1\,\Omega \tag{3.2}$$

$$L = \mu_0 \cdot r \cdot \left(ln\left(\frac{r}{0.35 \cdot d + 0.24 \cdot w} \right) + 0.079 \right) \tag{3.3}$$

$$R_{loss} = \frac{U}{2 \cdot w} \cdot \sqrt{\frac{\pi \cdot f \cdot \mu_0}{\sigma}} + R_{ESR,C} \tag{3.4}$$

Dabei bezeichnet σ die elektrische Leitfähigkeit der Leiterbahn. Für Kupfer beträgt diese $\sigma = 5.8 \cdot 10^7 \frac{1}{\Omega m}$ [49]. μ_0 ist eine Naturkonstante, die magnetische Permeabilität des Vakuums. Weiterhin gelten mit $d = 35\,\mu\mathrm{m}$ für die Höhe der Leiterbahn die folgenden geometrischen Zusammenhänge:

- Umfang der Schleifenantenne: $U = 2 \cdot \pi \cdot r = 15.7\,\mathrm{cm}$

[2] Die Gleichung für die Berechnung des Strahlungswiderstands R_r wurde um die richtige Dimension ergänzt. Eine weiterführende mathematische Herleitung ist in [9] auf den Seiten 237f. zu finden.

- Fläche der Schleifenantenne: $A = \pi \cdot r^2 = 19.63\,\mathrm{cm}^2$

Aus der Gleichung 3.1 folgt, dass das Produkt $LC = (f_0 \cdot 2\pi)^{-2} = const$ sein muss. Da die Induktivität L nur noch von der frei wählbaren Breite w der Leiterbahn abhängig ist, und somit frei gewählt werden kann, wird die Kapazität C auf $1\,\mathrm{pF}$ festgelegt, da Kondensatoren nicht beliebig erhältlich sind. Die Induktivität der kleinen Schleifenantenne berechnet sich dann zu

$$L = 134.5\,\mathrm{nH}.$$

Die Breite der Leiterbahn berechnet sich mit Hilfe der Gleichung 3.3 zu

$$w = \frac{1}{0.24} \cdot \left(\frac{r}{e^{\frac{L}{\mu_0 \cdot r} - 0.079}} - 0.35 \cdot d \right) = 1.508\,\mathrm{mm}.$$

Damit ergeben sich bei einem vernachlässigbaren Serienverlustwiderstand des Kondensators $R_{ESR,C}$ [58] die verbleibenden Ersatzparameter der kleinen Schleifenantenne nach den Gleichungen 3.2 und 3.4:

- $R_r = 530.2\,\mathrm{m\Omega}$
- $R_{loss} = 283.1\,\mathrm{m\Omega}$

Der Antennengewinnfaktor bzw. der Gewinn einer solchen kleinen Schleifenantenne beträgt $G = 1.5$ bzw. $g = 1.76\,\mathrm{dBi}$ in Bezug auf einen Isotropstrahler [28].

3.3.2 Realisierung

Zur Erprobung der Antenne wurde eine Platine gefertigt (Schaltplan und Layout sind im Anhang B.2 bzw C.2 zu finden). Diese enthält die Schleifenantenne nach der Dimensionierung im vorherigen Abschnitt und eine *SMA*-Buchse, um die Eingangsimpedanz messen zu können (siehe Abbildung 3.12). Zwischen der *SMA*-Buchse und der Schleifenantenne befinden sich Pads, um eine Impedanzanpassung der Antenne bei $434\,\mathrm{MHz}$ an $50\,\Omega$ vornehmen zu können.

In einem ersten Versuch wurden die Pads Z_3 und Z_4 mit $0\,\Omega$-Widerständen gebrückt, um die Eingangsimpedanz und den Eingangsreflexionsfaktor der Antenne für den unangepassten Fall betrachten zu können. Diese Daten werden zudem benötigt, um später eine Impedanzanpassung möglich zu machen.

Die Abbildung 3.8 zeigt die am Netzwerkanalysator aufgenommene Eingangsimpedanz der Schleifenantenne, dargestellt in einer speziellen Form der komplexen Ebene, dem Smith-Diagramm. Es zeigt sich, dass der Verlauf der Eingangsimpedanz abhängig von der Frequenz sowohl induktiv (obere Halbebene) als auch kapazitiv (untere Halbebene) verläuft. Bei $434\,\mathrm{MHz}$ nimmt die Eingangsimpedanz den komplexen und kapazitiven Wert $193.7\,\Omega - j231.6\,\Omega$ an.

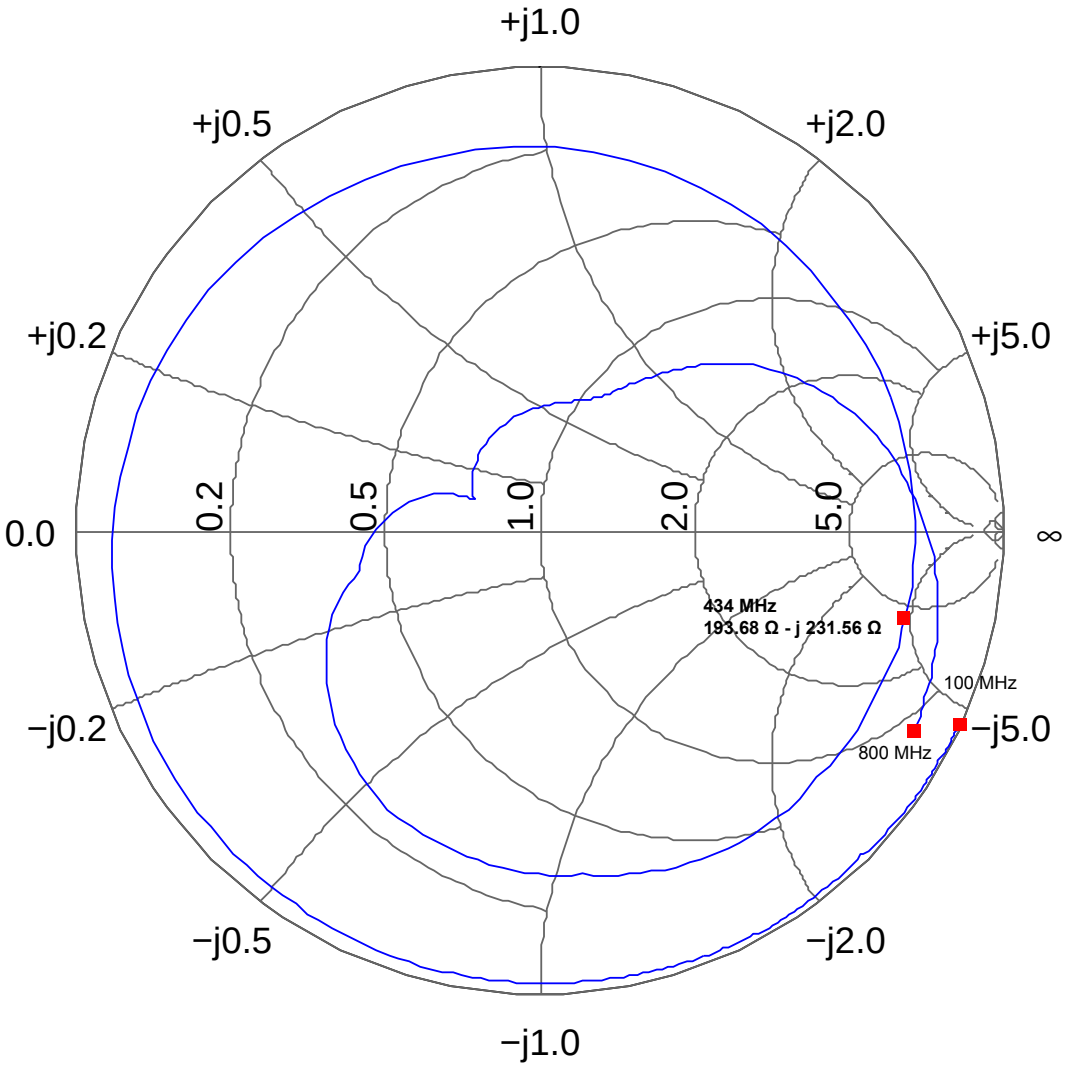


Abbildung 3.8: Smith-Diagramm der nicht angepassten Schleifenantenne

Aus der Eingangsimpedanz $\underline{Z}_e$ und der Wellenimpedanz $\underline{Z}_0 = 50\,\Omega$ lässt sich der Betrag des Eingangsreflexionsfaktors r_e berechnen.

$$r_e = \left|\frac{\underline{Z}_e - \underline{Z}_0}{\underline{Z}_e + \underline{Z}_0}\right| = \left|\frac{193.7 - j231.6 - 50}{193.7 - j231.6 + 50}\right| = 0.81$$

Dieser Wert bedeutet, dass etwa 66 % ($0.81^2 \approx 0.66$) der eingehenden Leistung wieder reflektiert wird und nicht an den Transceiver, bzw. an die Wakeup-Schaltung, weitergeleitet werden kann. Diese deutliche Reduzierung der Empfindlichkeit ist nicht akzeptabel, sodass, wie erwartet, eine Impedanzanpassung notwendig ist.

Auf der Abbildung 3.9 ist zusätzlich der logarithmische Eingangsreflexionsfaktor aufgetragen. Die Resonanz der Schleifenantenne liegt bei etwa $600\,\mathrm{MHz}$. Der Wert des Eingangsreflexionsfaktors beträgt dort $-16\,\mathrm{dB}$ bzw.

$$r_e = 10^{-\frac{16}{20}} \approx 0.16.$$

Sollte ein derartig niedriger Eingangsreflexionsfaktor bei $434\,\mathrm{MHz}$ durch die Impedanzanpassung erreicht werden, wäre das mehr als zufriedenstellend. Es wird festgelegt, dass ein Reflexionsfaktor kleiner als $-10\,\mathrm{dB}$ bzw. 0.316, entsprechend weniger als 10 % reflektierte Leistung, erreicht werden sollte.

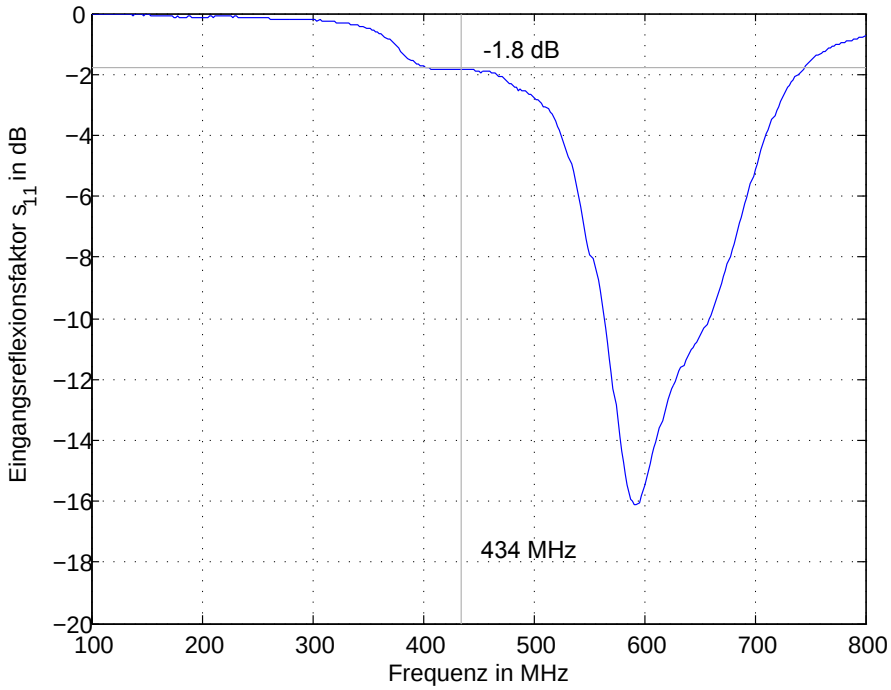


Abbildung 3.9: Eingangsreflexionsfaktor der nicht angepassten Schleifenantenne

3.3.3 Impedanzanpassung

Die Impedanzanpassung wurde mit Hilfe von *Microwave Office*, einer Hochfrequenzsimulationssoftware von AWR, durchgeführt. Es zeigte sich allerdings, dass in der Simulation ermittelte Netzwerke in der Erprobung nicht zur gewünschten Impedanzanpassung führten. Die Resonanz lag bis zu $200\,\mathrm{MHz}$ neben den gewünschten $434\,\mathrm{MHz}$. Infolgedessen wurde die Software nur dazu eingesetzt, um Tendenzen zu ermitteln, also welche Veränderungen an den Bauteilen zu einer Verschiebung der Resonanz in die gewünschte Richtung führen. So mussten einzelne reaktive Bauelemente mehrfach ausgetauscht werden, bis die optimalen Bauteilwerte gefunden waren. Die verwendete Anpassschaltung zeigt die Abbildung 3.10.

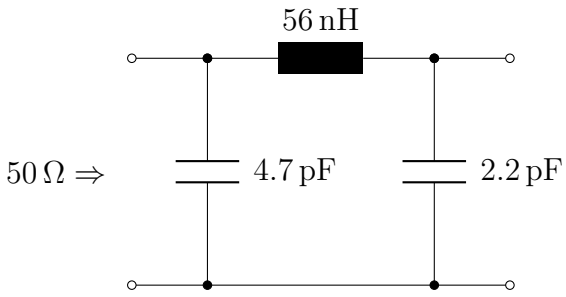


Abbildung 3.10: Impedanzanpassungsnetzwerk der Schleifenantenne

Den mit dieser bestückten Anpassschaltung gemessenen Verlauf des Eingangsreflexionsfaktors zeigt die Abbildung 3.11.

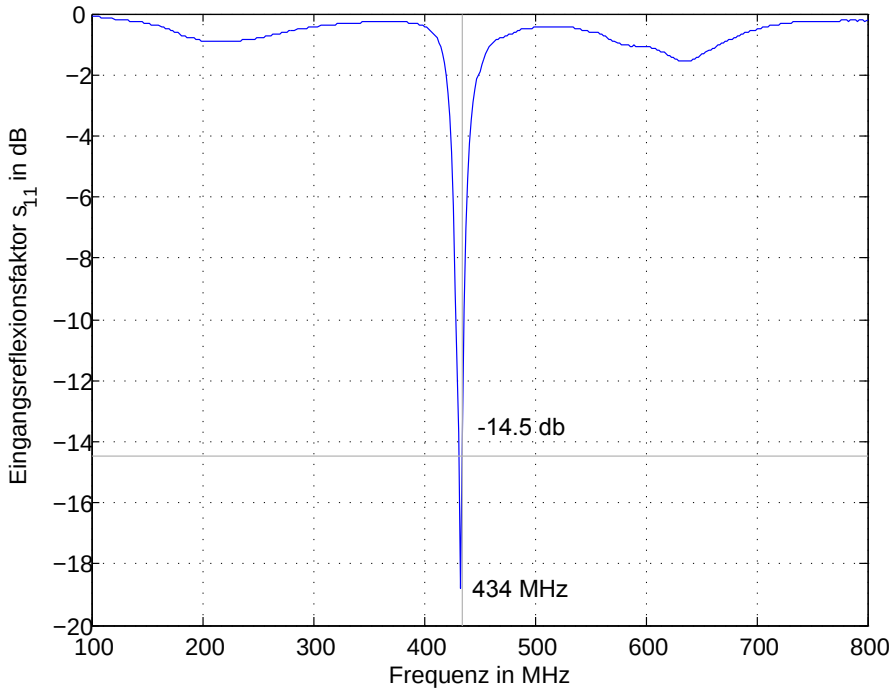


Abbildung 3.11: Eingangsreflexionsfaktor der angepassten Schleifenantenne

Zu erkennen ist eine schmalbandige Impedanzanpassung hoher Güte mit einem Eingangsreflexionsfaktor von $-14.5\,\mathrm{dB}$ bei der Frequenz $434\,\mathrm{MHz}$. Dies bedeutet, dass nur etwa 3.5 % der von der Schleifenantenne empfangenen Leistung nicht für die Weiterverarbeitung zur Verfügung steht.

Das folgende Foto zeigt die gefertigte Platine mit der runden Schleifenantenne, der *SMA*-Buchse zum Anschluss des Netzwerkanalysators und dem Impedanzanpassungsnetzwerk. Die weiteren Pads sind derzeit nicht von Interesse und sollen in den folgenden Versuchen eher dazu dienen, die Schleifenantenne realitätsnah erproben zu können. Durch das Metall innerhalb der

Antennenfläche, wie es auch bei dem späteren Zellensensor nicht zu vermeiden ist, ist mit einer Störung des elektromagnetischen Feldes zu rechnen, sodass die Schleifenantenne ggf. weniger Leistung aus dem Feld aufnehmen kann.

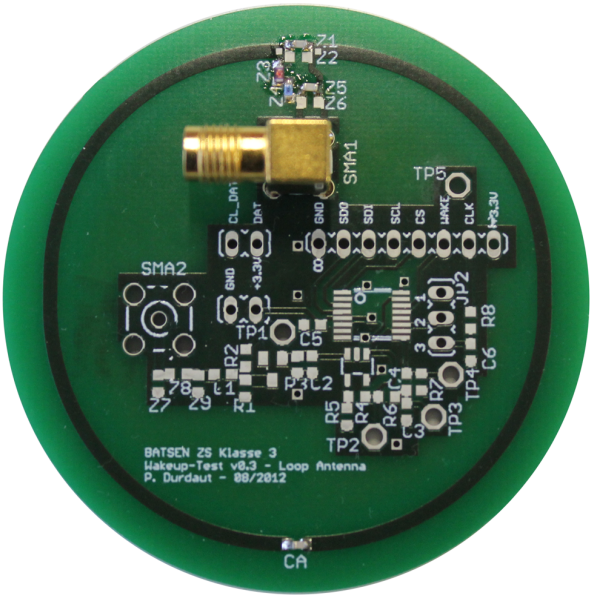


Abbildung 3.12: Angepasste Schleifenantenne auf Versuchsplatine

3.3.4 Kommerzielle „Chip-Antenne“

Alternativ zu der oben erläuterten Schleifenantenne sollte eine sogenannte *Chip-Antenne* Antenne erprobt werden. Der Hintergrund dabei ist, dass diese für eine bestimmte Frequenz bereits abgestimmt ist und ein zusätzliches Impedanzanpassungsnetzwerk ggf. eingespart werden kann. Für das $434\,\mathrm{MHz}$-Band ist das Angebot verfügbarer Antennen, aufgrund deren Größe wegen der Wellenlänge von $69\,\mathrm{cm}$ stark eingeschränkt. Chip-Antennen sind eher bei größeren Frequenzen, wie im GSM-Mobilfunk, üblich. Die einzige lieferbare Antenne ist die Chip-Antenne mit der Bezeichnung *ANT-433-SP* von *Linx Technologies* [30] gewesen. Diese bietet laut Datenblatt eine Eingangsimpedanz von $50\,\Omega$ und ein Stehwellenverhältnis ($VSWR$) < 1.9, was einer reflektierten Leistung von maximal 10 % entspricht.

Für die Erprobung dieser Antenne wurde eine Platine gefertigt (Schaltplan und Layout sind im Anhang B.3 bzw C.3 zu finden). Die Platine ist auf der folgenden Abbildung 3.13 zu sehen.

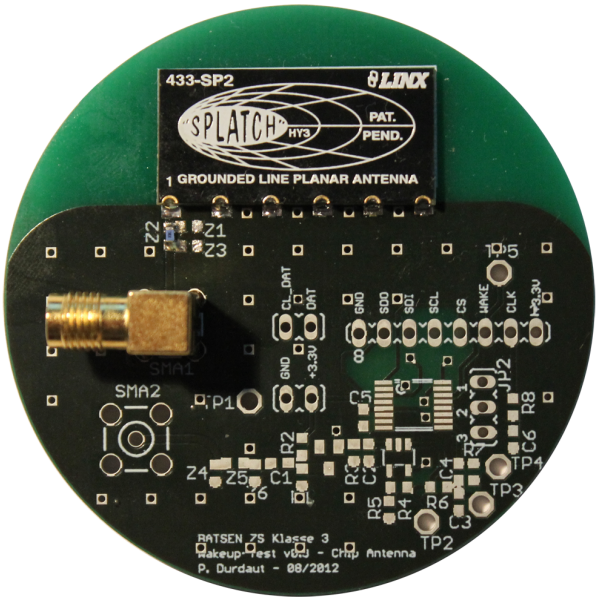


Abbildung 3.13: Versuchsplatine mit „Chip-Antenne“ und $0\,\Omega$-Brücke

Wie auf dem Foto zu sehen, wurde auch hier eines der für eine eventuelle Impedanzanpassung vorgesehenden Pads (Z_2) mit einer $0\,\Omega$-Brücke bestückt, um über die *SMA*-Buchse den Eingangsreflexionsfaktor messen zu können.

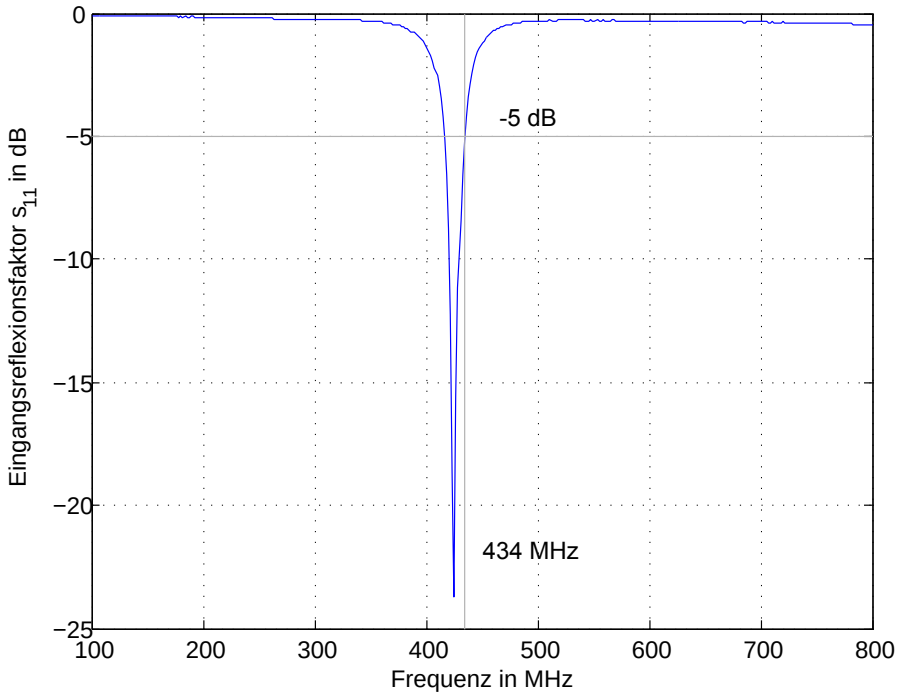


Abbildung 3.14: Eingangsreflexionsfaktor der nicht angepassten Chipantenne

Die Abbildung 3.14 zeigt das Resultat der Messung am Netzwerkanalysator, wobei eine Resonanz hoher Güte erkennbar ist. Kritisch ist jedoch die Resonanzfrequenz. Diese liegt mit $425\,\mathrm{MHz}$ genau $9\,\mathrm{MHz}$ unter der gewünschten Frequenz. Aufgrund der hohen Güte, bzw. der Schmalbandigkeit, beträgt der Eingangsreflexionsfaktor bei $434\,\mathrm{MHz}$ nur $-5\,\mathrm{dB}$, was einer Leistungsreflexion von etwa 32 % entspricht. Eine deart hohe Reflexion der Leistung ist nicht akzeptabel, sodass vor einer funktionalen Erprobung auch die Impedanz dieser Antenne an $50\,\Omega$ angepasst wird.

Das Ergebnis der Impedanzanpassung, die nach dem gleichen Vorgehen, wie bei der Schleifenantenne durchgeführt wurde, zeigen die beiden folgenden Abbildungen 3.15 und 3.16. Der Eingangsreflexionsfaktor bei $434\,\mathrm{MHz}$ konnte auf $-10\,\mathrm{dB}$ verbessert werden.

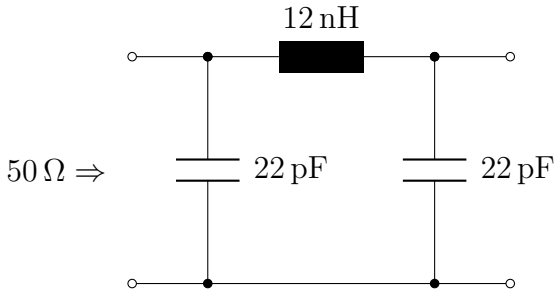


Abbildung 3.15: Anpassschaltung der Chipantenne

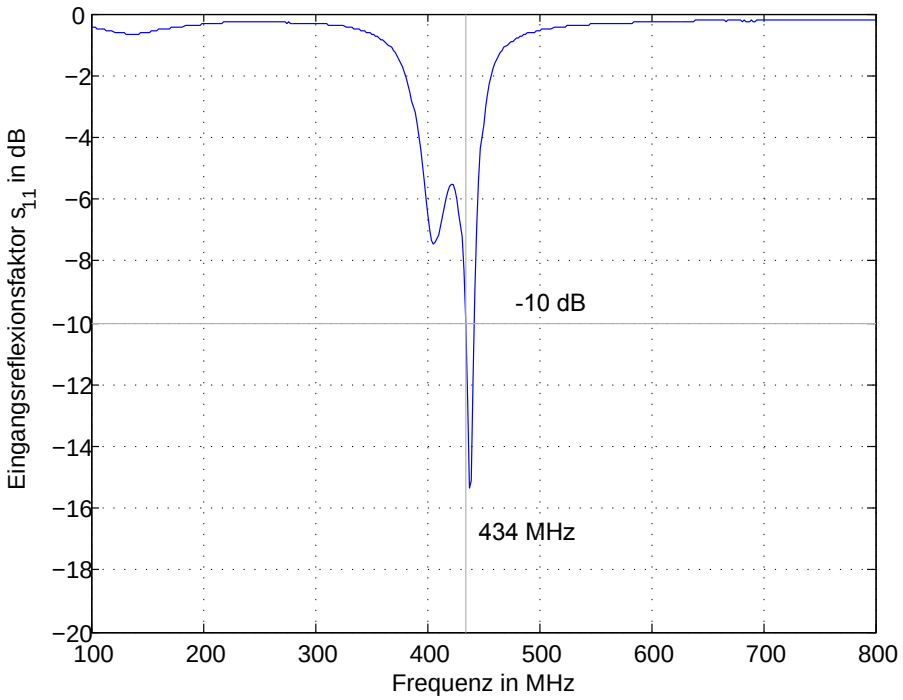


Abbildung 3.16: Eingangsreflexionsfaktor der angepassten Chipantenne

3.3.5 Erprobung und Vergleich

Zur Erprobung beider Antennen und zum anschließenden Vergleich wurde eine Versuchsreihe durchgeführt, die folgend beschrieben wird.

Mit der modifizierten Basisstation, bzw. mit dem in Abschnitt 3.2.1 vorgestellten Transceiver-Modul, wird über die Antenne aus Abbildung 3.1 ein 434 MHz-Trägersignal dauerhaft abgestrahlt. In einem veränderlichen Abstand und in verschiedenen Positionen wird dieses mit beiden Antennen empfangen und die Höhe des Empfangspegels an einem mit der Antenne verbundenen Spektrumanalysator ausgewertet.

Die Abbildung 3.17 verdeutlicht den Versuchsaufbau schematisch, wobei insbesondere die Größenverhältnisse erkennbar sein sollen. Für das bessere Verständnis existiert im Anhang F ein Foto des Versuchsaufbaus (Abbildung F.1).

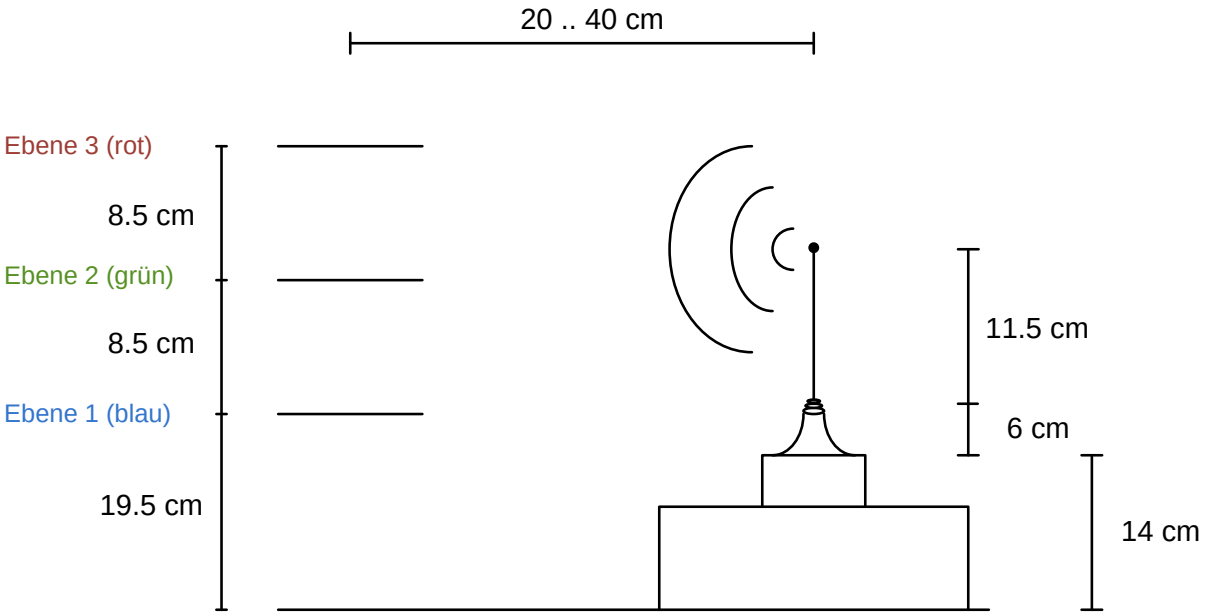


Abbildung 3.17: Versuchsaufbau zur Erprobungs- und Vergleichsmessungen an den Antennen

Die folgende Abbildung 3.18 zeigt das Leistungsdichtespektrum des gesendeten Trägersignals. Dieses liegt bei exakt 434 MHz und wird mit einer Leistung von 10 dBm gesendet.

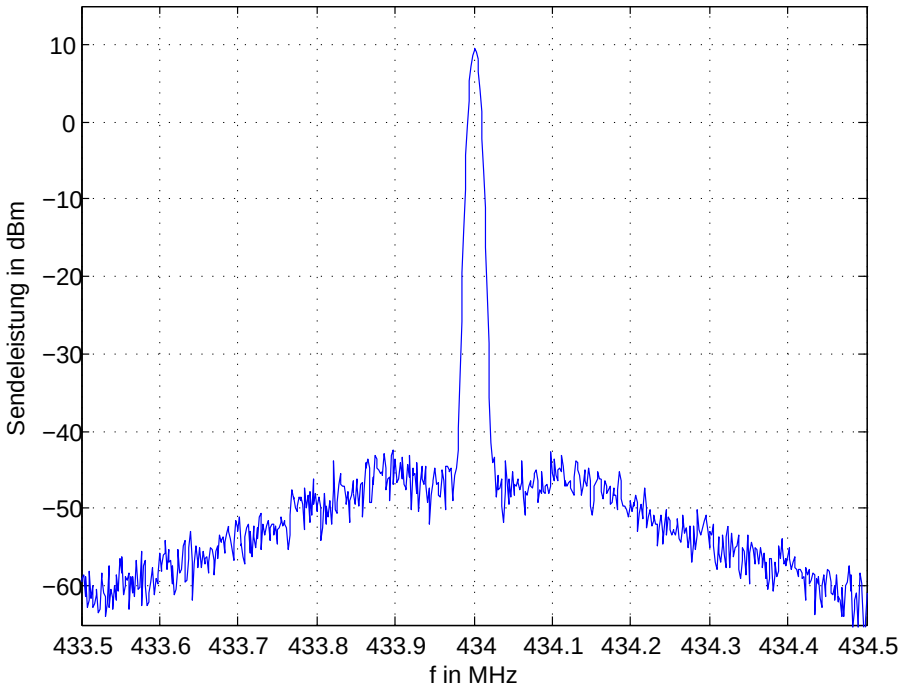


Abbildung 3.18: Leistungsdichtespektrum des gesendeten $434\,\mathrm{MHz}-$ Trägersignals

Insgesamt wurden für jede der beiden Antennen 18 Messwerte aufgenommen. Jeder Sensor befand sich sowohl liegend als auch stehend auf jeder der in Abbildung 3.17 eingezeichneten Ebenen. Zudem wurde der horizontale Abstand zur Sendeantenne in $10\,\mathrm{cm}$-Schritten von $20\,\mathrm{cm}$ bis $40\,\mathrm{cm}$ verändert.

Die Ergebnisse sind den Abbildungen 3.19 und 3.20 zu entnehmen. Die Farben rot, grün und blau stehen für die Ebene, auf der die entsprechende Messung durchgeführt wurde, wie in Abbildung 3.17 definiert. Man erkennt, dass der Empfangspegel am größten ist, wenn sich die Chip-Antenne in einem geringen horizontalen Abstand zur Sendeantenne stehend auf der Ebene 3 bzw. Ebene 2 befindet. Mit der Vergrößerung des horizontalen Abstandes verkleinert sich die empfangene Leistung. Im liegenden Zustand ist der Empfangspegel an der Chip-Antenne stets geringer als an der Schleifenantenne. Es fällt zudem auf, dass der Empfangspegel an der Schleifenantenne in einem horizontalen Abstand von $30\,\mathrm{cm}$ maximal ist. Die von der Antenne aufgenommene Leistung beträgt in diesem Abstand etwa $-30\,\mathrm{dBm}$. Bei einer Sendeleistung von $10\,\mathrm{dBm}$ beträgt der Verlust dann etwa $40\,\mathrm{dB}$. Da die ausgewählten Transceiver hohe Empfindlichkeiten (vgl. Abschnitt 2.4) bieten, stellen die $40\,\mathrm{dB}$ für die bidirektionale Kommunikation über die Transceiver kein Problem da. Aufgrund der erheblich geringeren Kosten für die Schleifenantenne und der Tatsache, dass diese innerhalb der Batteriezelle empfindlicher auf die magnetische Komponente der elektromagnetischen Strahlung reagieren wird, soll die oben dimensionierte Schleifenantenne auf den zu entwickelnden Zellensensoren Verwendung finden. Da mit einer zusätzlichen Dämpfung des Funksignals durch die Aluminiumhülse der Batterie zu rechnen ist, muss die noch zu erprobene Wakeup-Schaltung eine entsprechend höhere Empfindlichkeit als $-30\,\mathrm{dBm}$ aufweisen.

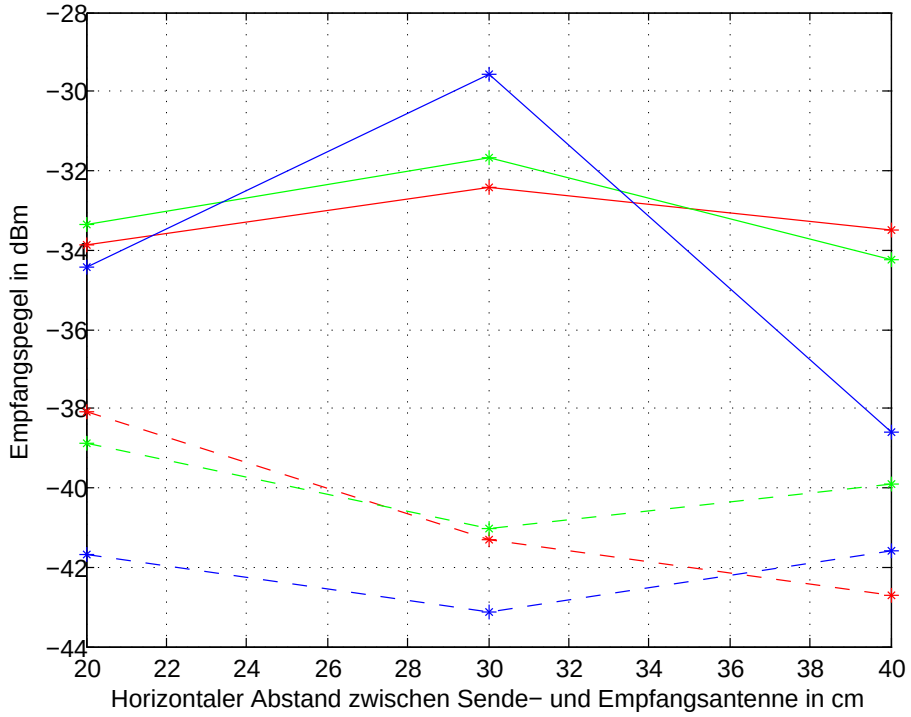


Abbildung 3.19: Empfangspegel an der Schleifenantenne (durchgezogene Linien) und der Chip-Antenne (gestrichelte Linien), jeweils im **liegenden** Zustand

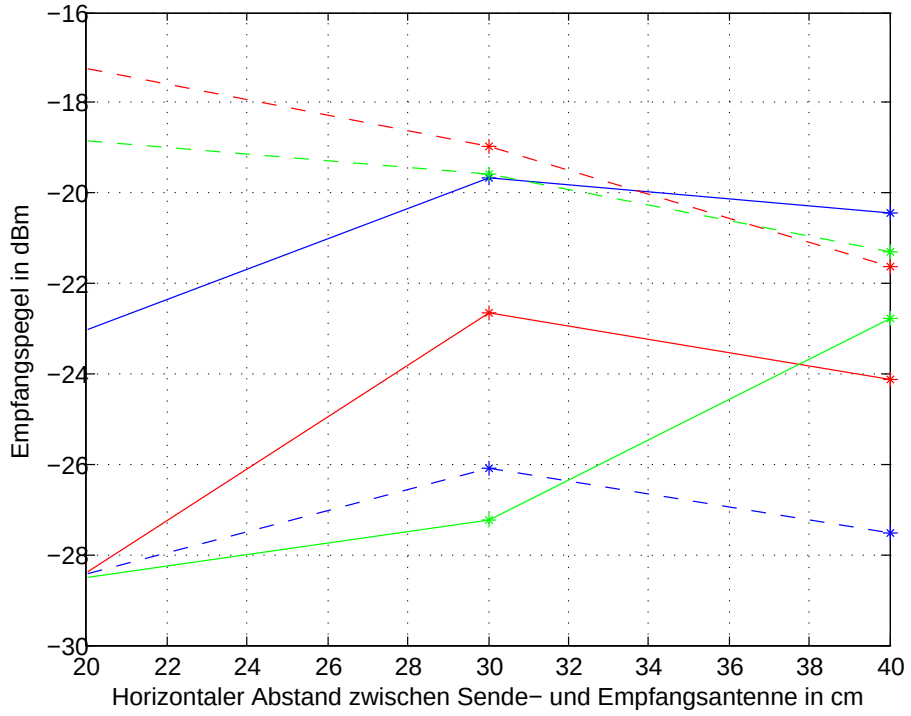


Abbildung 3.20: Empfangspegel an der Schleifenantenne (durchgezogene Linien) und der Chip-Antenne (gestrichelte Linien), jeweils im **stehenden** Zustand

3.4 Demodulatorschaltungen

3.4.1 Einführung

In den Abschnitten 2.2.5 und 2.5 wurde das grundsätzliche Prinzip der für den Zellensensor zu entwickelnden Wakeup-Schaltung bereits erläutert. Ziel ist es, aus dem empfangenen Wakeup-Signal ein *LF*-Signal der Frequenz $125\,\mathrm{kHz}$ zu erhalten, das als Eingangssignal für den LF Wakeup Receiver dient. Die dazu notwendige Signalverarbeitung durch eine geeignete elektrische Schaltung auf dem Zellensensor soll in diesem Abschnitt untersucht und praktisch erprobt werden.

Ein Vergleich zwischen der theoretischen Untersuchung des Wakeup-Signals in Abschnitt 2.5.2 und der Messung des Leistungsdichtespektrums des praktisch gesendeten Wakeup-Signals in Abschnitt 3.2.2.2 im Frequenzbereich, zeigt, dass das Wakeup-Signal durch eine Aufwärtsmischung um $434\,\mathrm{MHz}$ des periodischen und rechteckförmigen Modulationssignals entsteht.

Empfängerseitig, in der Wakeup-Schaltung, gilt es diese Aufwärtsmischung mit einer Abwärtsmischung rückgängig zu machen, um das Spektrum des Rechtecksignals wieder ins Basisband zu verschieben. Anschließend genügt eine einfache Bandpassfilterung, um aus dem Rechtecksignal eine gleichanteilfreie Sinusschwingung zu erhalten (vgl. Abbildung 2.10).

Eine Abwärtsmischung wird in modernen Nachrichtenempfängern (auch in den vorgestellten Transceivern) über die Multiplikation mit einer Trägerschwingung realisiert. Der Nachteil ist bei diesen aktiven Empfängern der dazu notwendige Betriebsstrom. Da dieser mit der grundsätzlichen Idee einer nahezu passiven Wakeup-Schaltung kollidiert, muss für diesen Zweck eine andere Möglichkeit gefunden werden.

Neue Frequenzkomponenten können im Allgemeinen nur durch nichtlineare Bauelemente entstehen, sodass der Gedanke nahe liegt, solche für die Frequenzumsetzung zu verwenden. In der Hochfrequenztechnik benutzt man dazu meist Halbleiterdioden mit einer Metall-Halbleiter-Sperrschicht, sogenannte *Schottky-Dioden*. Aufgrund geringerer Ladungsträgerpeicherung sind diese den pn-Sperrschicht-Dioden im Schaltverhalten überlegen. [70]

3.4.2 Mischung an einer Schottky-Diode

Wie es zum Effekt der Mischung an einer Halbleiterdiode kommt soll im folgenden anhand eines Beispiels erläuert werden. Gemäß der Abbildung 3.21 wird eine Schottky-Diode dazu mit zwei additiv überlagerten harmonischen Schwingungen der Frequenzen f_1 und f_2 der Form

$$u_q(t) = cos(2\pi f_1 t) + cos(2\pi f_2 t)$$

ausgesteuert.

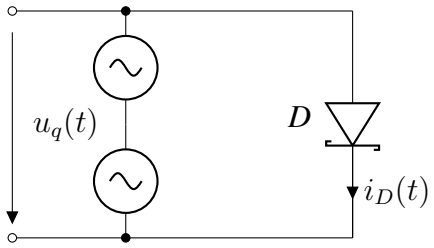


Abbildung 3.21: Aussteuerung einer Schottky-Diode mit einem Zweitonsignal

Der Zusammenhang zwischen der Spannung an der Diode $u_D(t)$ und dem Strom durch die Diode $i_D(t)$ wird durch die Shockley-Gleichung beschrieben [68].

$$i_D(t) = I_S \left(e^{\frac{u_D(t)}{n \cdot U_T}} - 1 \right)$$

- I_S: Sättigungssperrstrom der Diode
- n: Emissionskoeffizient der Diode
- U_T: Temperaturspannung $U_T = \frac{k \cdot T}{e}$
- k: Boltzmann-Konstante
- T: Temperatur
- e: Elementarladung

Entwickelt man die darin enthaltene Exponentialfunktion in eine Taylorreihe 2. Ordnung ergibt sich näherungsweise [41]

$$i_D(t) \approx I_S \left(\frac{1}{nU_T} \cdot u_D(t) + \frac{1}{2n^2 U_T^2} \cdot u_D(t)^2 \right).$$

Mit den Substitutionen $A = \frac{I_S}{nU_T}$ und $B = \frac{I_S}{2n^2 U_T^2}$ ergibt die Aussteuerung der Diodenkennlinie mit dem Zweitonsignal $u_q(t)$ den folgenden Strom durch die Diode.

$$\begin{aligned} i_D(t)\big|_{u_D(t)=u_q(t)} \approx\ & A \cdot cos(2\pi f_1 t) + A \cdot cos(2\pi f_2 t) \\ & + \frac{B}{2} \cdot (1 + cos(2\pi 2 f_1 t)) \\ & + \frac{B}{2} \cdot (1 + cos(2\pi 2 f_2 t)) \\ & + B \cdot (cos(2\pi(f_1 - f_2)t) + cos(2\pi(f_1 + f_2)t)) \end{aligned}$$

Offensichtlich entstehen durch die Diode neue Frequenzanteile. Neben einem Gleichstromanteil, den Harmonischen bei $2f_1$ und $2f_2$ sind bei der Mischung die Summen- und Differenzfrequenzen $f_1 + f_2$ und $f_1 - f_2$ von besonderem Interesse. Nimmt man für die Frequenzen $f_1 = 434.125\,\mathrm{MHz}$ und $f_2 = 434\,\mathrm{MHz}$ an, entsteht eine Frequenzlinie im Basisband bei $125\,\mathrm{kHz}$ (Abwärtsmischung). Da durch die hier nicht berücksichtigten Anteile der nichtlinearen Diodenkennlinie zudem noch weitere Frequenzanteile entstehen, ist das Signal nach der Mischung entsprechend mit einem Bandpassfilter zu filtern.

3.4.3 Hüllkurvendemodulator mit einer Schottky-Diode

3.4.3.1 Funktionsprinzip

Die einfachste elektrische Schaltung zur inkohärenten Demodulation einer amplitudenmodulierten Trägerschwingung ist in Abbildung 3.22 zu sehen. Eine solche Einweg-Gleichrichterschaltung wird zusammen mit dem Tiefpassfilter, bestehend aus einem Widerstand R und einem Kondensator C, als Hüllkurvendemodulator oder auch als Hüllkurvendetektor bezeichnet.

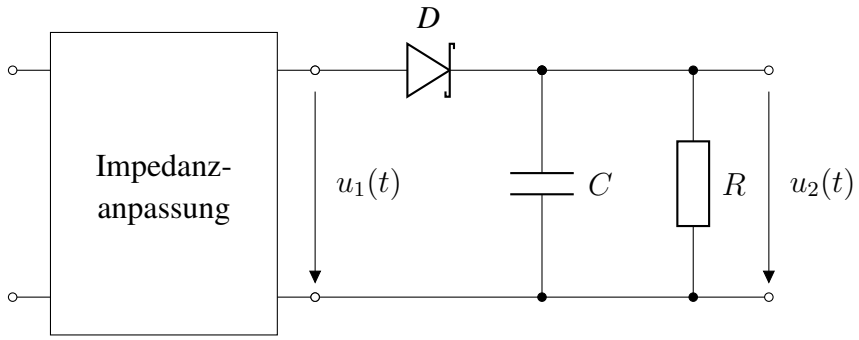


Abbildung 3.22: Prinzipschaltbild eines Hüllkurvendemodulators mit einer Schottky-Diode

Die Diode ist leitend, wenn die Eingangsspannung $u_1(t)$ größer ist als die Spannung am Kondensator. Der Kondensator wird solange aufgeladen, bis die Spannung $u_2(t)$ der Spitzenspannung $\widehat{u}_1$ abzüglich der Flussspannung der Diode entspricht. Sinkt die Eingangsspannung, entlädt sich die Kapazität C über den Widerstand R. Um näherungsweise die Hüllkurve des Eingangssignals zu erhalten, muss die Zeitkonstante $T = R \cdot C$ des Tiefpassfilters erheblich größer als die Periodendauer der Trägerschwingung T_T und kleiner als die Periodendauer des Modulationssignals T_m gewählt werden [33].

$$10 \cdot T_T < T < T_m$$

Für $f_T = 1/T_T = 434\,\mathrm{MHz}$ und $f_m = 1/T_m = 125\,\mathrm{kHz}$ liefern $R = 1\,\mathrm{k\Omega}$ und $C = 100\,\mathrm{pF}$ eine brauchbare Zeitkonstante, wie die folgende Simulation in PSpice zeigt.

Simulation Zur Verdeutlichung des Funktionsprinzips wurde der oben beschriebene Hüllkurvendemodulator nach Abbildung 3.22 in PSpice simuliert. Die Simulationsschaltung zeigt die Abbildung 3.23. Das Wakeup-Signal wird in Pspice multiplikativ aus einer $434\,\mathrm{MHz}$-Trägerschwingung und einem Rechtecksignal der Periodendauer $8\,\mu\mathrm{s}$ erzeugt. Als Gleichrichter wird eine ideale Diode mit einer Flussspannung von $0.7\,\mathrm{V}$ verwendet.

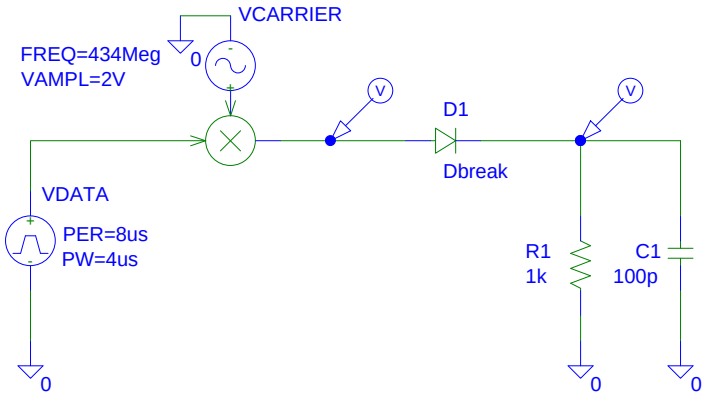


Abbildung 3.23: Schaltung der PSpice-Simulation eines Hüllkurvendemodulators mit idealer Diode

Das Ergebnis der Simulation im Zeitbereich zeigt die Abbildung 3.24. Das Ausgangssignal der Schaltung (rot) stellt die Hüllkurve der modulierten Trägerschwingung da. Wie erwartet ist die Amplitude der Hüllkurve um etwa $0.7\,\mathrm{V}$ geringer als die Spitzenspannung des *UHF*-Signals.

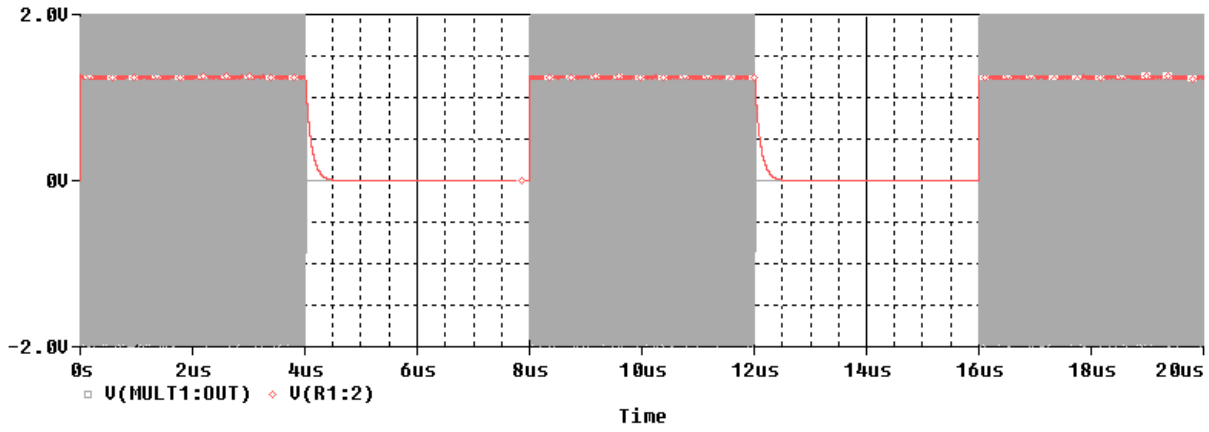


Abbildung 3.24: Ergebnis der PSpice-Simulation eines Hüllkurvendemodulators mit idealer Diode

Damit am Ausgang der Schaltung die Hüllkurve erscheint, muss die Spitzenspannung des Trägersignals größer als $0.7\,\mathrm{V}$ sein. Aufgrund der geringen Sendeleistung von $10\,\mathrm{dBm}$ und der nochmals erheblich geringeren Empfangsleistung auf dem Zellensensor (angenommen werden im Folgenden $-40\,\mathrm{dBm}$), muss jedoch davon ausgegangen werden, dass keine Spannung größer als $0.7\,\mathrm{V}$ zur Verfügung stehen wird.

3.4.3.2 Abschätzung der verfügbaren Spannung an der Diode

Die Erprobung der Schleifenantenne in Abschnitt 3.3.5 hat gezeigt, dass bei einer Sendeleistung von $10\,\mathrm{dBm}$ mit einer Empfangsleistung von etwa $-30\,\mathrm{dBm}$ zu rechnen ist. Wird vorläufig davon ausgegangen, dass das Aluminiumgehäuse der Batteriezelle die empfangene Leistung um weitere $10\,\mathrm{dB}$ bedämpft, ergibt sich eine Empfangsleistung von $-40\,\mathrm{dBm}$. Eine Leistung von $-40\,\mathrm{dBm}$ erzeugt an einem $50\,\Omega$-Widerstand eine Spannung von

$$U = \sqrt{10^{\frac{-40}{10}} \cdot 1\,\mathrm{mW} \cdot 50\,\Omega} = 2.23\,\mathrm{mV}.$$

Es ist zu erwarten, dass die Eingangsimpedanz des Hüllkurvendemodulators größer als $50\,\Omega$ sein wird, sodass sich durch eine Impedanzanpassung eine Spannungserhöhung vor der Diode ergibt. Mit welchem Faktor bei der Spannungserhöhung zu rechnen ist, kann zum jetzigen Zeitpunkt nicht angegeben werden. Wahrscheinlich wird die Spannung trotzdem im mV-Bereich liegen, sodass der Einsatz einer Schottky-Diode mit einer geringen Flussspannung unverzichtbar ist.

3.4.3.3 Dimensionierung

In verschiedenen Veröffentlichungen [4] [16] [19] [38], die bereits bei der Konzeptfindung herangezogen wurden, ist immer die Schottky-Diode *HSMS-285x* von *Avago Technologies* verwendet worden. Aufgrund der vielen verfügbaren Informationen und der guten Dokumentation durch den Hersteller, wurde entschieden, diese Diode auch in dieser Arbeit zu verwenden. Diese Schottky-Diode [8] ist für Eingangsleistungen kleiner als $-20\,\mathrm{dBm}$ und Frequenzen unter $1.5\,\mathrm{GHz}$ geeignet. Laut Datenblatt sind Detektorschaltungen mit einer Empfindlichkeit von bis zu $-57\,\mathrm{dBm}$ realisierbar.

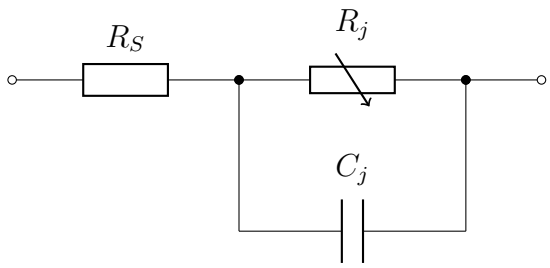


Abbildung 3.25: Lineares Ersatzschaltbild der Schottky-Diode *HSMS-285x* für kleine Eingangsleistungen

Für Leistungen kleiner als $-30\,\mathrm{dBm}$ existiert ein quadratischer Zusammenhang zwischen der Eingangsleistung und der Ausgangsspannung der Diode. Je höher die Leistung, desto größer ist die Ausgangsspannung. Der wichtigste Parameter, der in diesem Arbeitsbereich der Diode

berechenbar und zur Dimensionierung der Detektorschaltung herangezogen wird, ist die Spannungssensitivität γ (engl. voltage sensitivity). Um diese berechnen zu können, wird das lineare Ersatzschaltbild der Diode für diesen Arbeitsbereich betrachtet (Abbildung 3.25). [7]

R_j ist der Sperrschichtwiderstand der Diode, über den idealerweise die gesamte Spannung des hochfrequenten Eingangssignals abfällt. Die parasitäre Sperrschichtkapazität C_j schließt einen Teil dieser Spannung kurz. R_s ist ein parasitärer Widerstand, der die ohmschen Verluste auf dem Chip und in den Bonddrähten berücksichtigt. Die Schottky-Diode *HSMS-285x* ist durch die folgende Werte gekennzeichnet:

- Sättigungssperrstrom: $I_S = 3\,\mu\mathrm{A}$
- Idealitätsfaktor: $n = 1.06$
- Sperrschichtkapazität: $C_j = 0.18\,\mathrm{pF}$
- Serienwiderstand $R_S = 25\,\Omega$

Die Sperrschichtwiderstand der Diode berechnet sich nach[3]

$$R_j = \frac{8.33 \cdot 10^{-5} \cdot n \cdot T}{I_T} \cdot \frac{V}{K}$$

bzw. bei einer Temperatur von 25 °C zu

$$R_j = \frac{26.3\,\mathrm{mV}}{I_T}.$$

Der Strom durch die Diode I_T setzt sich additiv aus dem Sättigungssperrstrom I_S und einem optionalen Bias-Strom I_b zusammen.

$$I_T = I_S + I_b$$

Daraus lässt sich die Spannungssensitivität [7]

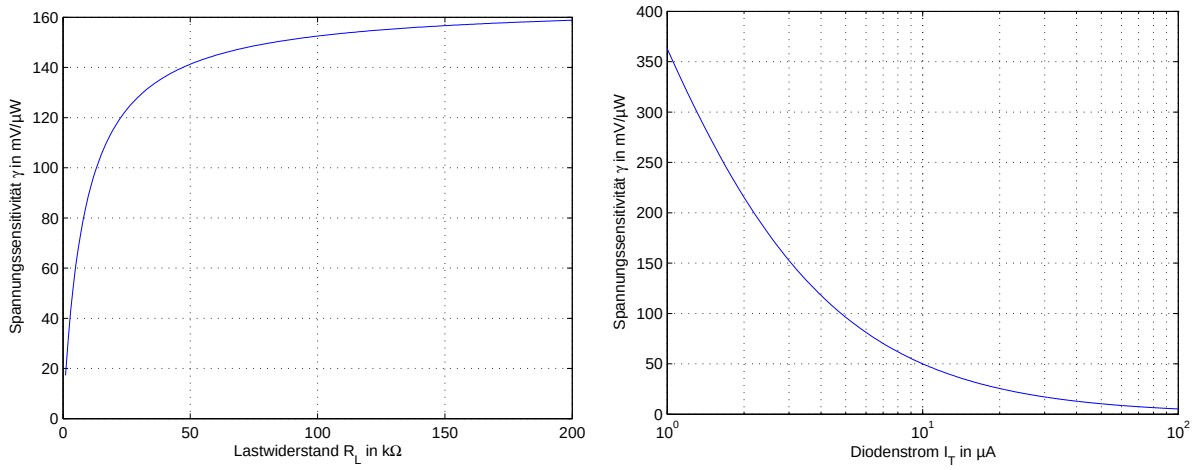
$$\gamma = \frac{0.52}{I_T \cdot (1 + \omega^2 C_j^2 R_S R_j) \cdot (1 + \frac{R_j}{R_L})} \tag{3.5}$$

berechnen, wobei R_L den externen Lastwiderstand (R in Abbildung 3.22) bezeichnet. Um die Spannungssensitivität zu maximieren, muss R_L groß im Vergleich zu R_j sein. R_j lässt sich verkleinern, indem die Diode mit einem Bias-Strom I_b vorbestromt wird. Da der Strom im Nenner der Gleichung auftritt, darf dieser jedoch auch nicht zu groß werden, da die Spannungssensitivität dadurch direkt sinkt. Zudem muss darauf geachtet werden, dass der Sperrschichtwiderstand

[3] Die Gleichung für die Berechnung des Sperrschichtwiderstands R_j wurde um die richtige Dimension ergänzt.

R_j klein genug bleibt, um eine Impedanzanpassung an $50\,\Omega$ schaltungstechnisch realisieren zu können.

Letztendlich sind die beiden Größen R_L und I_b frei wählbar. Für deren Dimensionierung wurde die Spannungssensitivität in Abhängigkeit beider Größen in Abbildung 3.26 dargestellt.



(a) Spannungssensitivität in Abhängigkeit des Lastwiderstands R_L bei $I_b = 0$

(b) Spannungssensitivität in Abhängigkeit des Diodenstroms I_T bei $R_L = 100\,\mathrm{k}\Omega$

Abbildung 3.26: Spannungssensitivität γ in Abhängigkeit des Lastwiderstands und des Diodenstroms für die Schottky-Diode *HSMS-285x*

Abbildung 3.26 (a) zeigt, dass die Spannungssensitivität mit steigendem Lastwiderstand exponentiell größer wird. Der Grenzwert liegt bei etwa $160\,\mathrm{mV}/\mu\mathrm{W}$. Bei einem, vom Hersteller empfohlenen, $R_L = 100\,\mathrm{k}\Omega$ beträgt die Spannungssensitivität bereits $152.5\,\mathrm{mv}/\mu\mathrm{W}$. Zudem ist aus der zweiten Abbildung zu erkennen, dass die Spannungssensitivität bei einem steigenden Diodenstrom sinkt. Somit ist es nicht sinnvoll die Diode vorzustromen. Damit beträgt $I_T = I_S = 3\,\mu\mathrm{A}$.

3.4.3.4 Simulation

Um alle Bauelemente der Wakeup-Schaltung dimensionieren zu können, wurden parallel zueinander Simulationen in PSpice und in Microwave Office durchgeführt. Zeitbereichssimulationen in PSpice wurden verwendet, um die zur Filterung der Hüllkurve nötigen Bauelemente dimensionieren zu können, sodass am Ausgang der Schaltung eine Trägerschwingung der Frequenz $125\,\mathrm{kHz}$ entsteht. Microwave Office wurde dazu verwendet, die Eingangsimpedanz des Hüllkurvendemodulators zu bestimmen und anschließend ein Netzwerk zur Impedanzanpassung an $50\,\Omega$ auszulegen.

Auf Abbildung 3.27 ist die Schaltung zu sehen, die der PSpice-Simulation zugrunde liegt. Das verwendete Diodenmodell stammt direkt vom Hersteller, sodass mit einem realitätsnahen Verhalten zu rechnen ist.

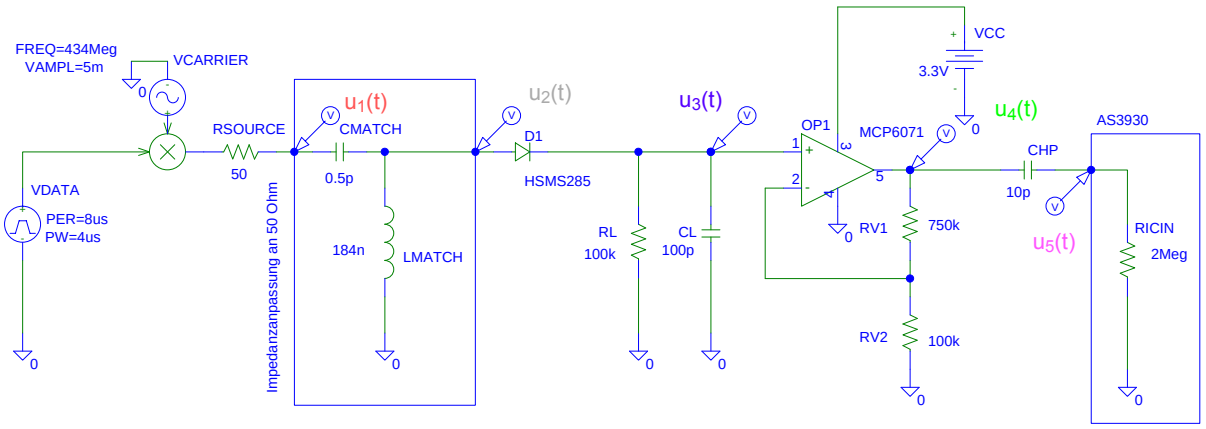


Abbildung 3.27: Schaltung der PSpice-Simulation der Wakeup-Schaltung mit der Schottky-Diode *HSMS-285x*

Vor die Diode ist ein Netzwerk aus zwei Blindelementen geschaltet, um die Eingangsimpedanz des Hüllkurvendetektors an $50\,\Omega$, die Impedanz des späteren RF-Umschalters, anzupassen. Die Dimensionierung des Anpassungsnetzwerks erfolgte nach der Simulation der Eingangsimpedanz (vgl. Abbildung 3.29) ebenfalls in Microwave Office.

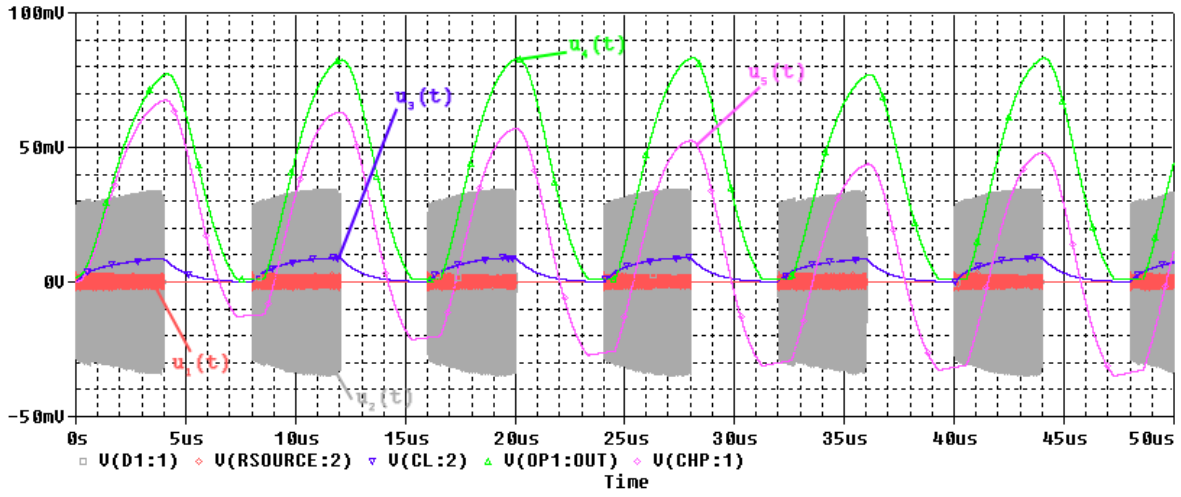


Abbildung 3.28: Ergebnis der PSpice-Simulation der Wakeup-Schaltung mit der Schottky-Diode *HSMS-285x*

Das Ergebnis der Simulation zeigt, dass die Amplitude des *UHF*-Signals hinter dem Anpassungsnetzwerk (grau) mit $30\,\mathrm{mV}$ etwa 10-fach höher als vor dem Netzwerk (etwa $3\,\mathrm{mV}$) aus Blindelementen ist (rot). Der Faktor der Spannungsüberhöhung bildet sich aus dem Quotienten des Realteils der Eingangsimpedanz von $523.9\,\Omega$ und dem Quellwiderstand von $50\,\Omega$.

Die Zeitkonstante des Tiefpassfilters wurde im Vergleich zur vorherigen Simulation in Abschnitt 3.4.3.1 um den Faktor 100 vergrößert, sodass die Form der Hüllkurve weniger rechteckförmig ist. Dadurch wird das Spektrum des Signals schmalbandiger und das Signal leichter zu

filtern. Bei einer Amplitude der „Hüllkurve“ von etwa $9\,\mathrm{mV}$ (violett) und einer Empfindlichkeit des LF Wakeup Receivers *AS3930* von $100\,\mu\mathrm{V}$ [5] wäre eine Bandpassfilterung des Signals auf passivem Wege möglich, um ein $125\,\mathrm{kHz}$-Signal zu erhalten. Im Hinblick auf spätere Messungen während der praktischen Erprobung wurde jedoch entschieden, einen Operationsverstärker als nichtinvertierenden Verstärker (auch Elektrometerverstärker genannt) einzusetzen. Ausgewählt wurde für diesen Zweck der Operationsverstärker *MCP6071* von *Microchip* mit einem geringen Strombedarf von $110\,\mu\mathrm{A}$ [35]. Auch dieser ist vom Hersteller als PSpice-Modell vorhanden. Aufgrund des geringen Verstärkungs-Bandbreite-Produkts von $1.2\,\mathrm{MHz}$ wird durch die 8.5-fache Verstärkung eine Tiefpassfilterung des Eingangssignals erreicht (grün). Im Allgemeinen ist dieser Effekt unerwünscht, in diesem Anwendungsfall jedoch nützlich. Die Entfernung des Gleichanteils geschieht über einen einzelnen Kondensator, der in Verbindung mit dem Eingangswiderstand des LF Wakeup Receiver-Chips als Hochpassfilter wirkt. Das Ausgangssignal (magenta) befindet sich nach etwa $40\,\mu\mathrm{s}$ im eingeschwungenen Zustand. Es ist zwar leicht verzerrt, kommt der geforderten $125\,\mathrm{kHz}$-Trägerschwingung jedoch sehr nahe. Sollte sich während der Erprobung zeigen, dass die Verzerrung zu stark ist, liesse sich die Verzerrung weiter vermindern, indem hinter dem Operationsverstärker beispielsweise ein entsprechender Parallelschwingkreis eingefügt wird.

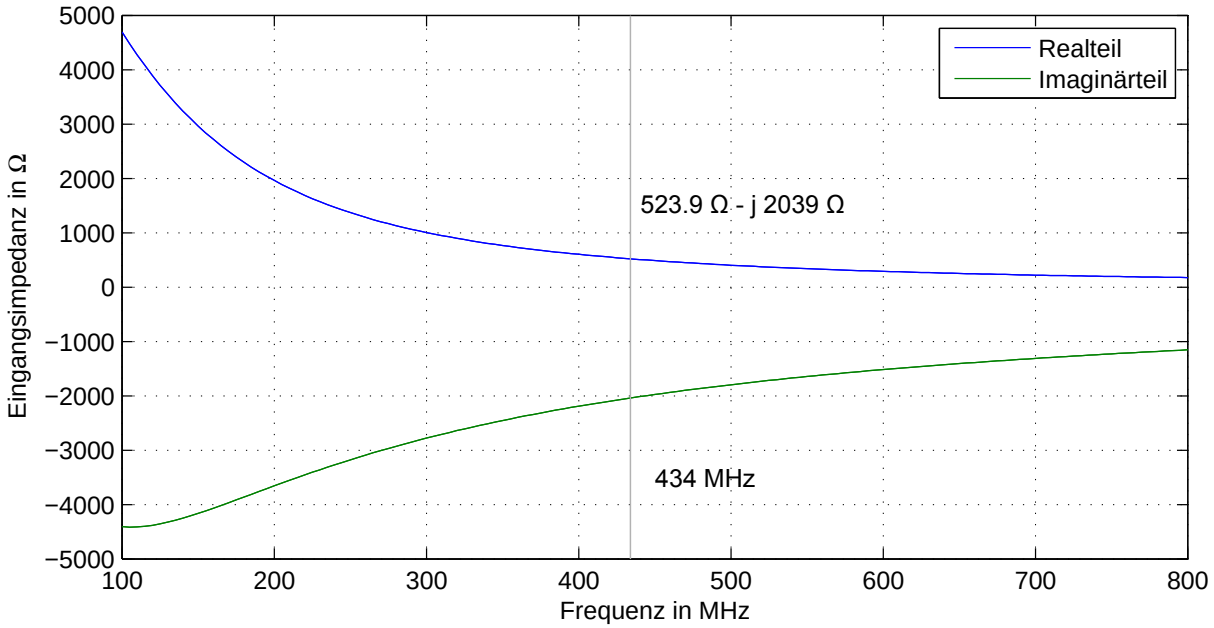


Abbildung 3.29: In Microwave Office ermittelte Eingangsimpedanz des Hüllkurvendemodulators mit der Diode *HSMS-285x*

3.4.3.5 Realisierung

Für die praktische Erprobung des Hüllkurvendemodulators wurde eine Versuchsplatine gefertigt, die in Abbildung 3.31 abgebildet ist. Die der Simulationsschaltung im Wesentlichen identische Schaltung zeigt die Abbildung 3.30.

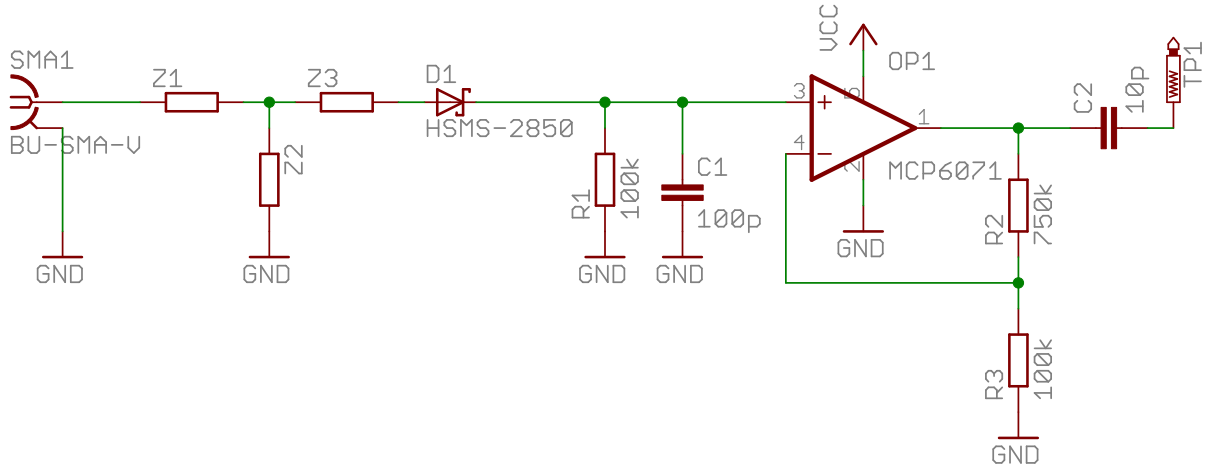


Abbildung 3.30: Schaltung des auf der Versuchsplatine realisierten Hüllkurvendemodulators mit einer Schottky-Diode

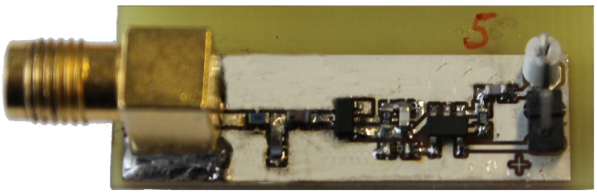


Abbildung 3.31: Versuchsplatine für den Hüllkurvendemodulator mit einer Schottky-Diode vom Typ *HSMS-285x*

Wie bei den Versuchsplatinen für die Antennen wurden auch vor der Schottky-Diode Pads vorgesehen, über die eine Impedanzanpassung erfolgen kann. Um einen maximalen Leistungsfluss in die Diode zu gewährleisten, wurden Impedanzanpassungsnetzwerke in Microwave Office berechnet und anschließend messtechnisch über die *SMA*-Buchse am Netzerkanalysator verifiziert. Dabei konnte, ähnlich wie bei den Antennen, eine Diskrepanz zwischen Simulation und Messung festgestellt werden. Um eine brauchbare Impedanzanpassung an $50\,\Omega$ zu erreichen, mussten die Werte der Bauelemente mehrmals korrigiert werden. Das Ergebnis der Impedanzanpassung zeigen die Abbildungen 3.32 und 3.33. Mit zwei *SMD*-Induktivitäten von *Coilcraft* [14] konnte bei $434\,\mathrm{MHz}$ und einer Eingangsleistung von $-20\,\mathrm{dBm}$ ein Eingangsreflexionsfaktor von $-12.9\,\mathrm{dB}$ erreicht werden. Das entspricht einem Leistungsfluss von 94.9 % der von der Antenne abgegebenen Leistung in den Hüllkurvendemodulator.

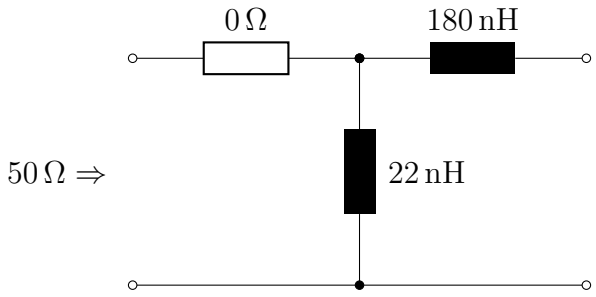


Abbildung 3.32: Impedanzanpassungsschaltung des Hüllkurvendemodulators mit einer Schottky-Diode

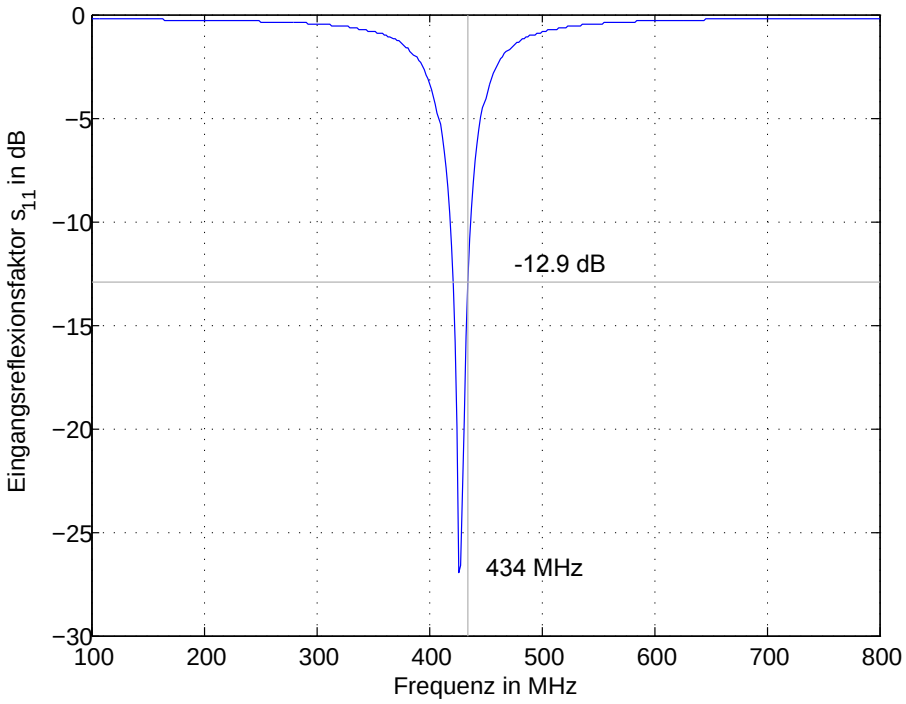


Abbildung 3.33: Eingangsreflexionsfaktor des Hüllkurvendemodulators nach der Impedanzanpassung bei einer Eingangsleistung von $-20\,\mathrm{dBm}$

Nachdem die Impedanzanpassung erfolgreich abgeschlossen war, konnte der Hüllkurvendemodulator erstmals praktisch erprobt werden. Dazu wurde das bereits in Abschnitt 3.2.2 analysierte Wakeup-Signal mit der Basisstation und dem entwickelten Transceiver-Modul erzeugt. Über verschiedene $50\,\Omega$-Dämpfungsglieder wurde die Versuchsplatine mit dem Wakeup-Signal gespeist und der Ausgang der Schaltung oszillographiert. Zu beachten ist, dass sich die Grenzfrequenz des am Ausgang befindlichen Hochpassfilters, durch den Tastkopf des Oszilloskops verändert. Die Wirkung kann jedoch vernachlässigt werden. Die Abbildung 3.34 zeigt den gemessenen Verlauf der Ausgangsspannung.

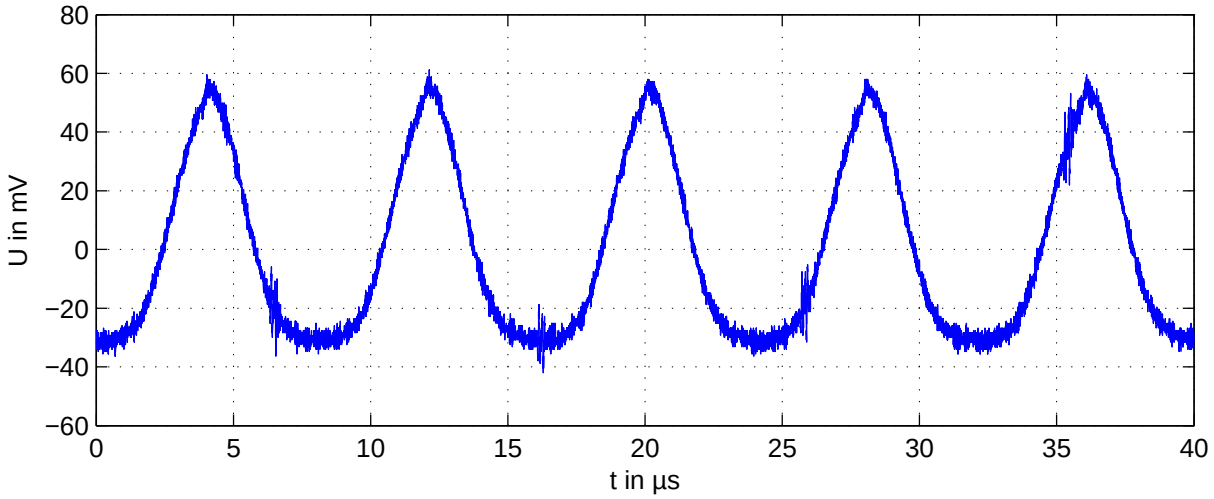


Abbildung 3.34: Ausgangsspannung des Hüllkurvendemodulators auf der Versuchsplatine bei einer Eingangsleistung von $-38.3\,\mathrm{dBm}$

Der Verlauf der Spannung zeigt, dass das Wakeup-Signal erfolgreich demoduliert werden konnte. Die Frequenz des Ausgangssignals beträgt, wie gefordert, $125\,\mathrm{kHz}$ und hat, wie in der Simulation, eine Spitze-Spitze-Spannung von knapp $100\,\mathrm{mV}$. Zu erkennen sind, wie schon in der Simulation, enorme Verzerrungen des Signals. In weiteren Versuchen muss untersucht werden, wie tolerant der LF Wakeup Receiver bei dessen Eingangssignal ist, oder ob eine zusätzliche Filterung notwendig ist.

3.4.4 Hüllkurvendemodulator mit Greinacher-Schaltung

Während der Recherche der verschiedenen Konzepte ist aufgefallen, dass die in verschiedenen Veröffentlichungen vorgestellten Schaltungen durchweg mit mehr als einer Schottky-Diode des Typs *HSMS-285x* arbeiten. Aus diesem Grund werden folgend zwei weitere Schaltungsvarianten mit jeweils zwei Dioden kurz vorgestellt und erprobt. Die erste Schaltung enthält eine Villard- bzw. Greinacher-Schaltung, wie sie als Teil einer Wakeup-Schaltung in [19] und als Teil einer Schaltung für das Energy Harvesting in [16] zum Einsatz kommt.

3.4.4.1 Funktionsprinzip

In Abbildung 3.35 ist die Schaltung eines Hüllkurvendemodulators dargestellt, die im Wesentlichen aus zwei Villard-Schaltungen besteht. Zusammen ergeben diese die Greinacher-Schaltung.

Der Grundidee der Greinacher-Schaltung besteht in der Verdopplung der Eingangsspannung. Entgegen der vorgestellten Schaltung mit einer Diode wird dazu die Spannung an der Diode D_1 abgegriffen. Diese besitzt die doppelte Spitze-Spitze-Spannung der Eingangsspannung $u_1(t)$ und lädt den Kondensator C_2. Zur Verhinderung der Entladung des Kondensators ist die Diode

D_2 in Sperrrichtung dazwischen geschaltet [10]. Die Ausgangsspannung $u_2(t)$ erreicht so nach gewisser Zeit eine Spannung von

$$\lim_{t\to\infty} u_2(t) = 2 \cdot \widehat{u_1} - 2 \cdot U_F,$$

wobei U_F die Flussspannung der Dioden bezeichnet. Um eine große Ausgangsspannung zu erhalten, muss die Flussspannung der Dioden möglichst klein sein, sodass auch in diesem Fall der Einsatz von Schottky-Dioden sinnvoll ist.

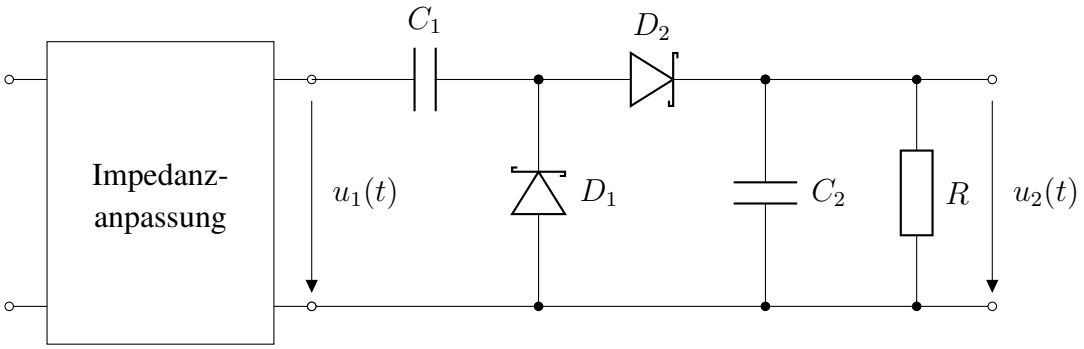


Abbildung 3.35: Prinzip eines Hüllkurvendemodulators mit der Greinacher-Spannungsverdopplerschaltung

Theoretisch denkbar ist die Serienschaltung zusätzlicher Villard-Schaltungen, bestehend aus einer Diode und einem Kondensator, wobei die Eingangsspannung einer Stufe jeweils an der Diode der vorherigen Stufe abgegriffen wird. Ziel dabei ist eine Spannungsvervielfachung, die über die Spannungsverdopplung hinausgeht. Angewandt wird eine solche Schaltung beim Energy Harvesting, wie in [16] beschrieben. Da in jeder Stufe eine Verminderung der Ausgangsspannung um die Flussspannung der Diode auftritt, ist diese Vorgehensweise jedoch nur bei ausreichend großen Eingangsspannungen sinnvoll. Wie die Simulation in Abschnitt 3.4.3.4 gezeigt hat, liegt die Flussspannung der Schottky-Diode zwar bei vergleichsweise sehr geringen $20\,\mathrm{mV}$, die Amplitude des hochfrequenten Signals vor der Diode ist mit etwa $30\,\mathrm{mV}$ jedoch auch nicht erheblich größer. Auch wenn die Eingangsimpedanz der Schaltung steigen sollte, sodass die Spannungsanhebung hinter dem Impedanzanpassungsnetzwerk größer wird, scheint die Serienschaltung zusätzlicher Villard-Schaltungen für diese Anwendung nicht geeignet zu sein.

3.4.4.2 Realisierung

Die Realisierung und Erprobung erfolgte nach demselben Schema, wie bei dem Hüllkurvendemodulator mit einer Schottky-Diode in Abschnitt 3.4.3.5. Die leicht veränderte Schaltung, die lediglich um eine Diode und einen Kondensator erweitert wurde, zeigt die Abbildung 3.36.

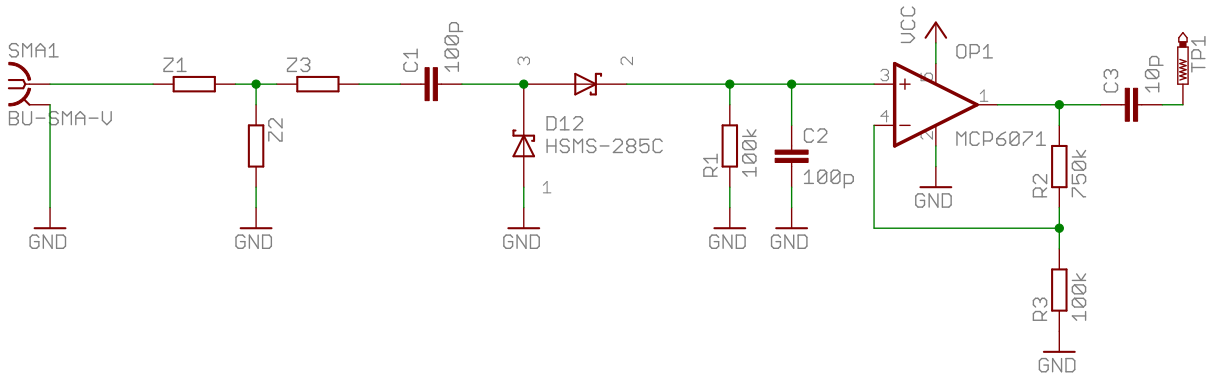


Abbildung 3.36: Schaltung des auf der Versuchsplatine realisierten Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Greinacher-Schaltung

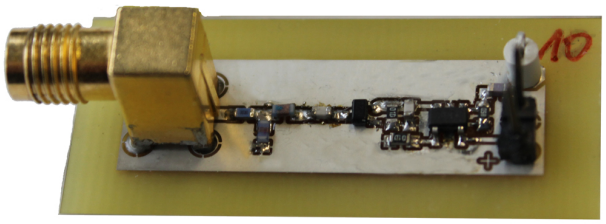


Abbildung 3.37: Versuchsplatine des Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Greinacher-Schaltung

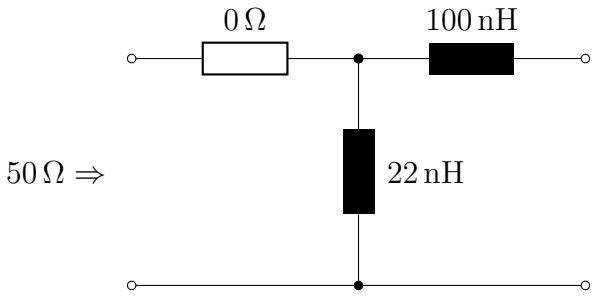


Abbildung 3.38: Impedanzanpassungsschaltung des Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Greinacher-Schaltung

Die Impedanzanpassung erfolgte wieder mit zwei Induktivitäten von *Coilcraft*, wobei die Güte gesunken ist. Die Abbildung 3.39 zeigt eine vergleichsweise erheblich breitbandigere Impedanzanpassung mit einem Eingangsreflexionsfaktor von $-11\,\mathrm{dB}$ bei $434\,\mathrm{MHz}$.

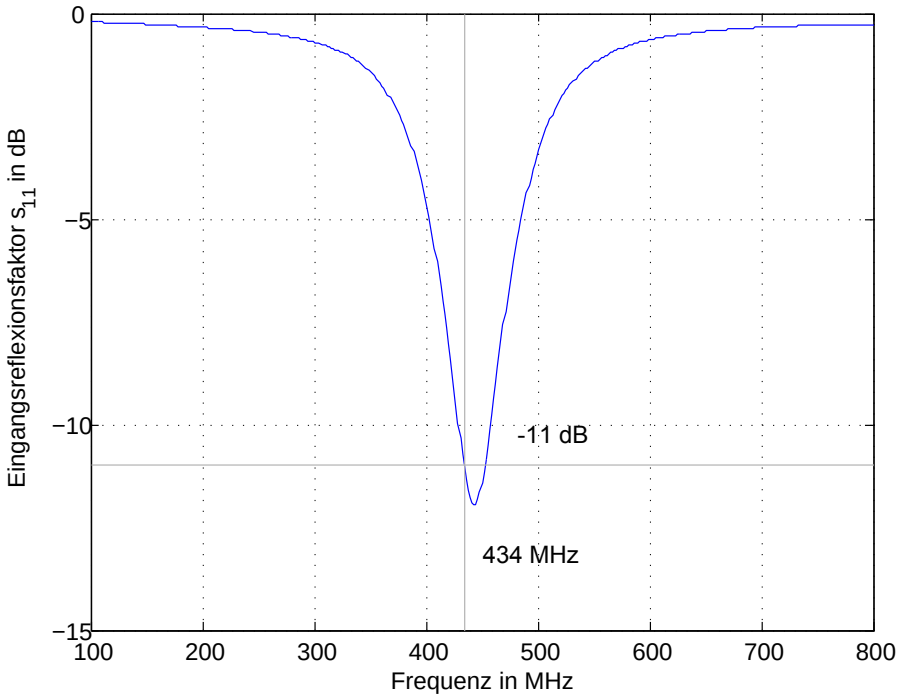


Abbildung 3.39: Eingangsreflexionsfaktor des Hüllkurvendemodulators mit Greinacher-Schaltung nach der Impedanzanpassung bei einer Eingangsleistung von $-20\,\mathrm{dBm}$

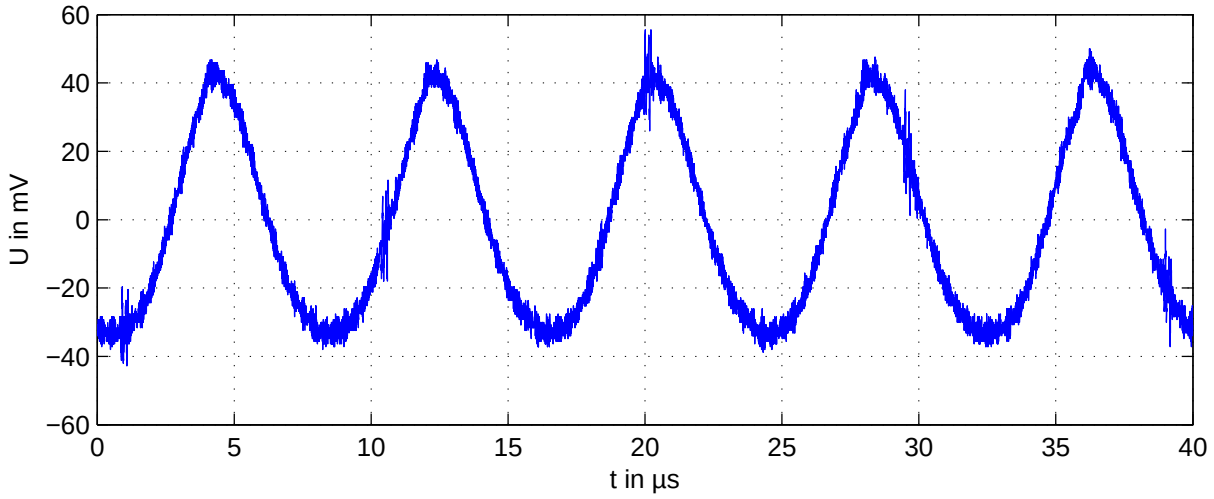


Abbildung 3.40: Ausgangsspannung des Hüllkurvendemodulators mit Greinacher-Schaltung auf der Versuchsplatine bei einer Eingangsleistung von $-38.3\,\mathrm{dBm}$

Betrachtet man das Ausgangssignal der Versuchsplatine bei gleicher Eingangsleistung wie oben, erkennt man kaum einen Unterschied. Die Höhe der Amplituden ist in etwa genauso groß wie beim Hüllkurvendemodulator mit nur einer Diode. Eine Verdopplung, oder zumindest eine maßgebliche Vergrößerung der Ausgangsspannung, ist nicht eingetreten. Die Verzerrungen

des Ausgangssignals, soweit man das im Zeitbereich erkennen kann, fallen etwas geringer aus. Für eine genauere Analyse müsste das Signal jedoch einer Frequenzbereichsanalyse unterzogen werden. Bisher ist also keine entscheidende Verbesserung durch den Einsatz einer zweiten Schottky-Diode zu erkennen. Wie sich die Schaltungen bei weiteren Eingangsleistungen verhalten, wird in Abschnitt 3.4.6 erläutert.

3.4.5 Hüllkurvendemodulator mit Delon-Schaltung

Bei der dritten Hüllkurvendemodulatorschaltung handelt es sich ebenfalls um einen Schaltung mit zwei Dioden. In der sogenannten Delon-Schaltung soll auch eine Spannungsverdopplung erreicht werden. In [38] wurde diese Schaltungsvariante benutzt, um Energie aus dem elektromagnetischen Feld zu gewinnen und eine Gleichspannung zu gewinnen (Energy Harvesting).

3.4.5.1 Funktionsprinzip

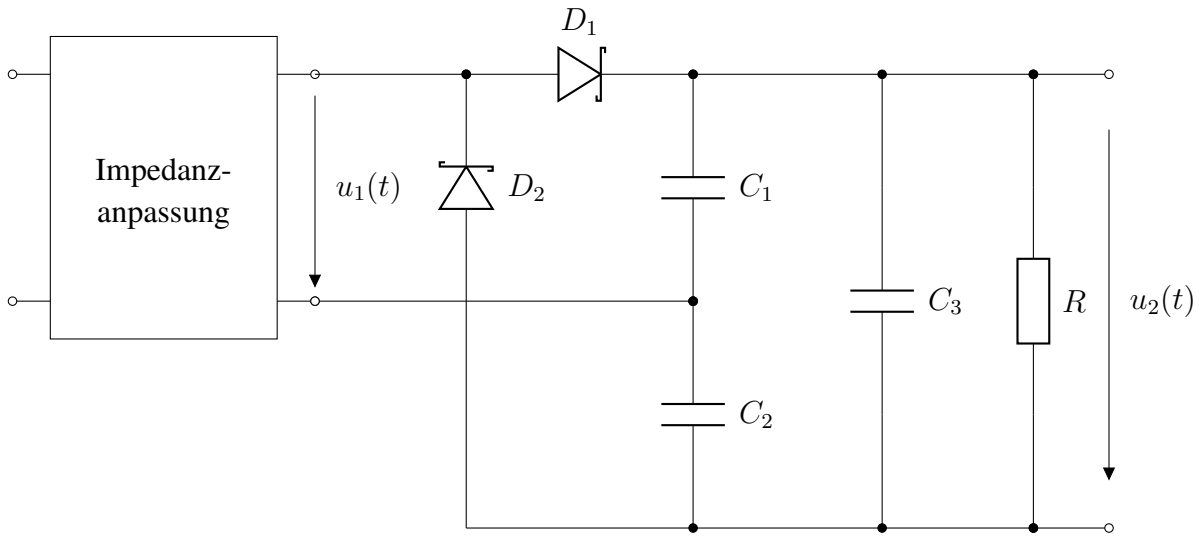


Abbildung 3.41: Prinzip eines Hüllkurvendemodulators mit der Delon-Spannungsverdopplerschaltung

Die Delon-Schaltung benötigt im Vergleich zur Greinacher-Schaltung einen zusätzlichen Kondensator und zwei getrennte Massepotentiale. Theoretisch sind die Ausgangssignale jedoch identisch. Die positive Halbwelle der Eingangsspannung $u_1(t)$ lädt den Kondensator C_1 auf die Spannung $\widehat{u_1} - U_F$ auf, wobei U_F auch hier wieder die Flussspannung der Diode bezeichnet. Aufgrund des symmetrischen Aufbaus der Schaltung, wird der Kondensator C_2 durch die negative Halbwelle der Eingangsspannung über die Diode D_2 auf die Spannung $-\widehat{u_1} + U_F$ geladen [48]. Durch den Abgriff der Ausgangsspannung zwischen der Kathode der Diode D_1 und

der Anode der Diode D_2 ergibt sich, genau wie bei der Greinacher-Schaltung, die Spannungsverdopplung

$$\lim_{t\to\infty} u_2(t) = 2 \cdot \widehat{u_1} - 2 \cdot U_F.$$

3.4.5.2 Realisierung

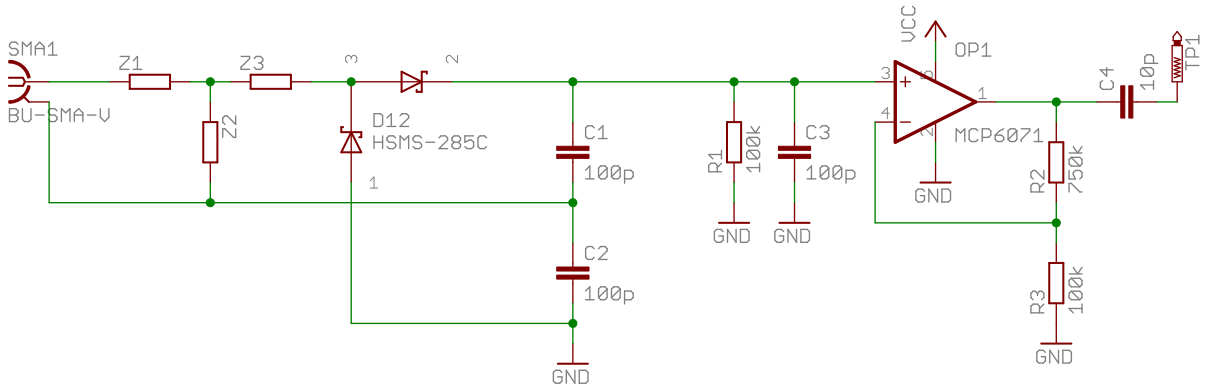


Abbildung 3.42: Schaltung des auf der Versuchsplatine realisierten Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Delon-Schaltung

Auch für den Hüllkurvendemodulator mit Delon-Schaltung erfolgte eine Realisierung und eine Erprobung, wie für die beiden vorherigen Fälle. Der einzige Unterschied, neben der veränderten Spannungsverdopplerschaltung, besteht in einer zweiten Massefläche auf der Versuchsplatine. Wie oben beschrieben und auch in der Schaltung in Abbildung 3.42 zu erkennen ist, ist das Massepotential des hinteren Schaltungsteils nicht dasselbe, wie das des Eingangssignals. Dementsprechend ist auf dem Foto eine Trennung der Masseflächen zu erkennen.

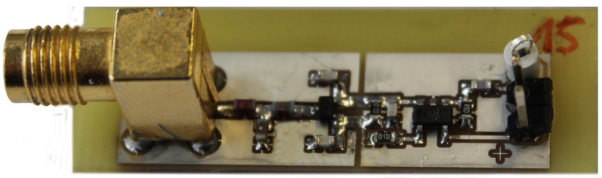


Abbildung 3.43: Versuchsplatine des Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Delon-Schaltung

Für die Impedanzanpassung an $50\,\Omega$ wurde zwar eine dritte Induktivität benötigt, mit der allerdings eine Anpassung sehr hoher Güte erreicht werden konnte. Bei $434\,\mathrm{MHz}$ beträgt der Eingangsreflexionsfaktor $-17.8\,\mathrm{dB}$, was einer reflektierten Leistung von nur 1.7 % entspricht.

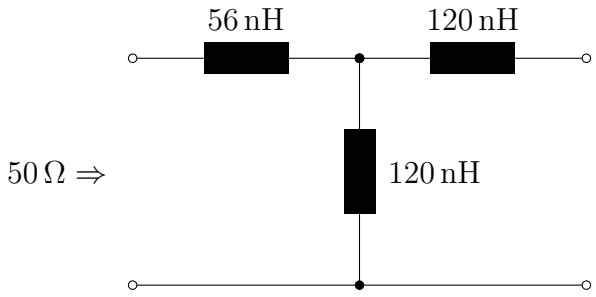


Abbildung 3.44: Impedanzanpassungsschaltung des Hüllkurvendemodulators mit zwei Schottky-Dioden in einer Delon-Schaltung

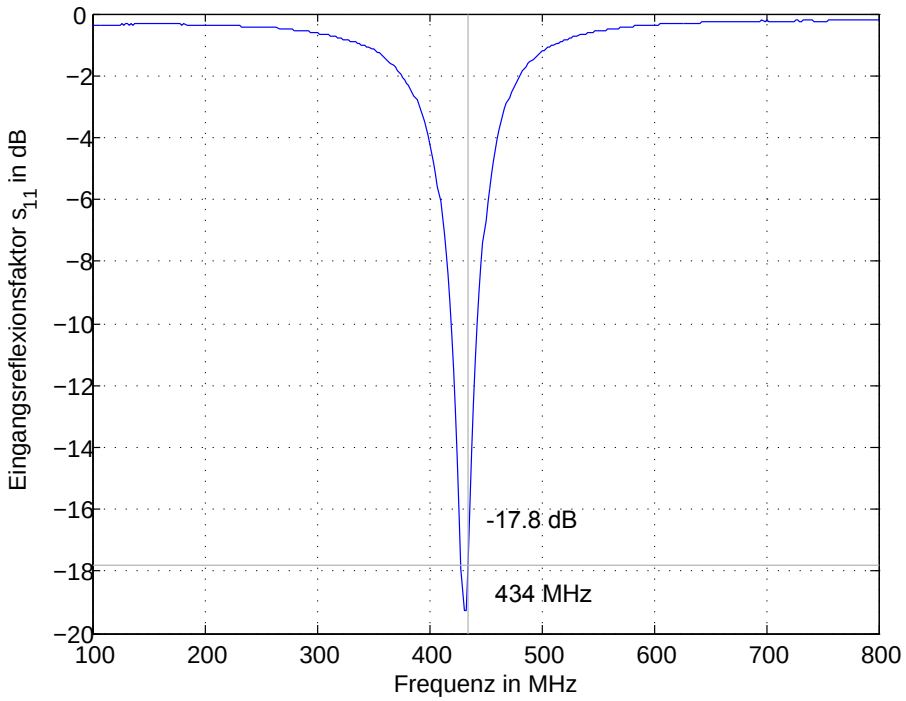


Abbildung 3.45: Eingangsreflexionsfaktor des Hüllkurvendemodulators mit Delon-Schaltung nach der Impedanzanpassung bei einer Eingangsleistung von $-20\,\mathrm{dBm}$

Trotz der besseren Impedanzanpassung zeigt die Abbildung 3.46 auch für die Delon-Schaltung keine Verdopplung der Eingangsspannung. Im Vergleich zu den Ausgangssignalen der beiden bereits untersuchten Schaltungen, sind hier zudem sehr viel stärkere Verzerrungen festzustellen. Weitere Untersuchungen haben gezeigt, dass die enormen Signalverzerrungen erst ab einer Eingangsleistung ab etwa $-25\,\mathrm{dBm}$ verschwinden. Für noch größere Eingangsleistungen werden die Verzerrungen sogar geringer als bei den vorherigen Schaltungsvarianten (vgl. Abschnitt 3.4.6).

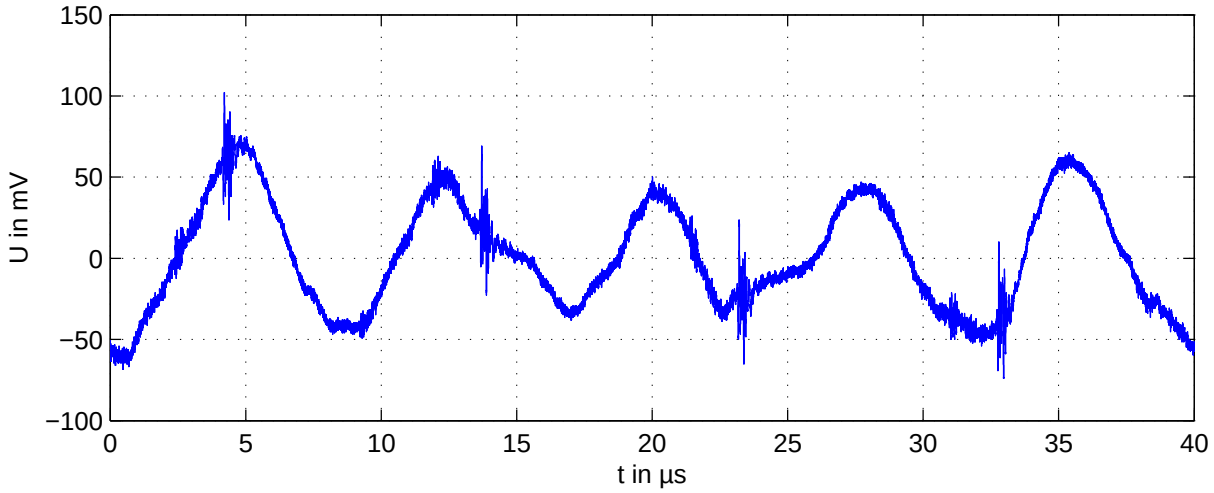


Abbildung 3.46: Ausgangsspannung des Hüllkurvendemodulators mit Delon-Schaltung auf der Versuchsplatine bei einer Eingangsleistung von $-36.3\,\mathrm{dBm}$

3.4.6 Vergleich

Um entscheiden zu können, welcher der vorgestellten Hüllkurvendemodulatoren für den Einsatz als Wakeup-Schaltung auf dem Zellensensor geeignet ist, wurde jeweils die Höhe der Ausgangsspannung in Abhängigkeit der Eingangsleistung untersucht. Aufgrund der Nichtlinearität der Schaltungen verändert sich abhängig von der Eingangsleistung die Eingangsimpedanz, sodass infolgedessen auch der Eingangsreflexionsfaktor je nach Eingangsleistung variiert.

Die Abbildung 3.47 macht deutlich, dass die Ausgangsspannung zwischen dem Hüllkurvendemodulator mit einer Diode und dem mit der Spannungsverdopplerschaltung nach Greinacher im Wesentlichen identisch ist. Die Delon-Schaltung erzeugt weitestgehend erheblich geringere Ausgangsspannungen. Betrachtet man dazu die Eingangsreflexionsfaktoren in Abbildung 3.48 ist festzustellen, dass sich dieses Verhalten trotz der erheblich besseren Impedanzanpassung der Greinacher-Schaltung zeigt. Da in erster Linie die Höhe der Ausgangsspannung entscheidend ist, kommt ein Einsatz der Delon-Schaltung nicht in Frage.

Sollte sich in weiteren Versuchen zeigen, dass der LF Wakeup Receiver tolerant gegenüber Verzerrungen des Eingangssignals ist, wird der zu Beginn vorgestellte Hüllkurvendemodulator mit einer Schottky-Diode verwendet werden. Dieser bietet, trotz des im Vergleich zur Greinacher-Schaltung niedrigeren Eingangsreflexionsfaktors, dieselbe Ausgangsspannung und ist zudem preisgünstiger zu realisieren. Die Empfindlichkeit liegt etwa zwischen $-45\,\mathrm{dBm}$ und $-50\,\mathrm{dBm}$, lässt sich durch eine Optimierung der Impedanzanpassung womöglich jedoch noch erhöhen.

Der Grund, weshalb die Spannungsverdopplerschaltungen keine größeren Ausgangsspannungen liefern, ist nach Rücksprache mit der Forschungsgruppe *IMTEK* der Universität Freiburg [19] hauptsächlich durch Sperrströme der Dioden zu begründen, die bei den hier eingesetzten Dioden des Typs *HSMS-285x* bei bis zu $175\,\mu\mathrm{A}$ liegen [8].

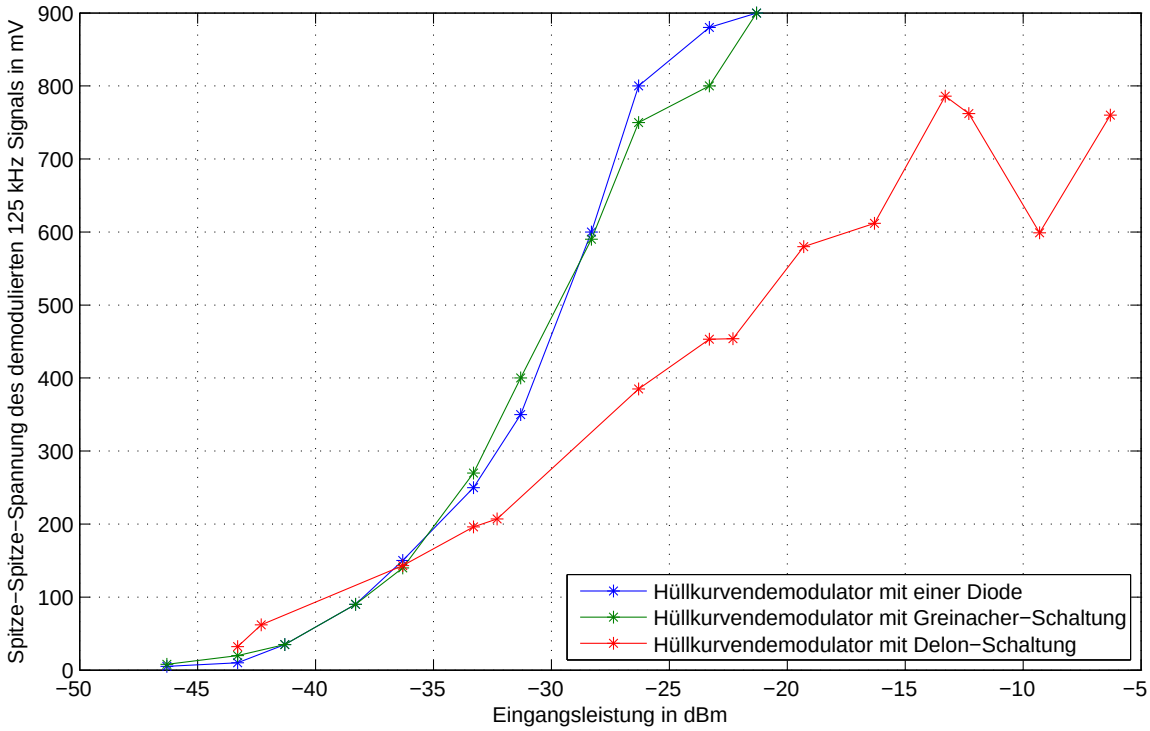


Abbildung 3.47: Ausgangsspannungen der untersuchten Hüllkurvendemodulatorschaltungen in Abhängigkeit der Eingangsleistung

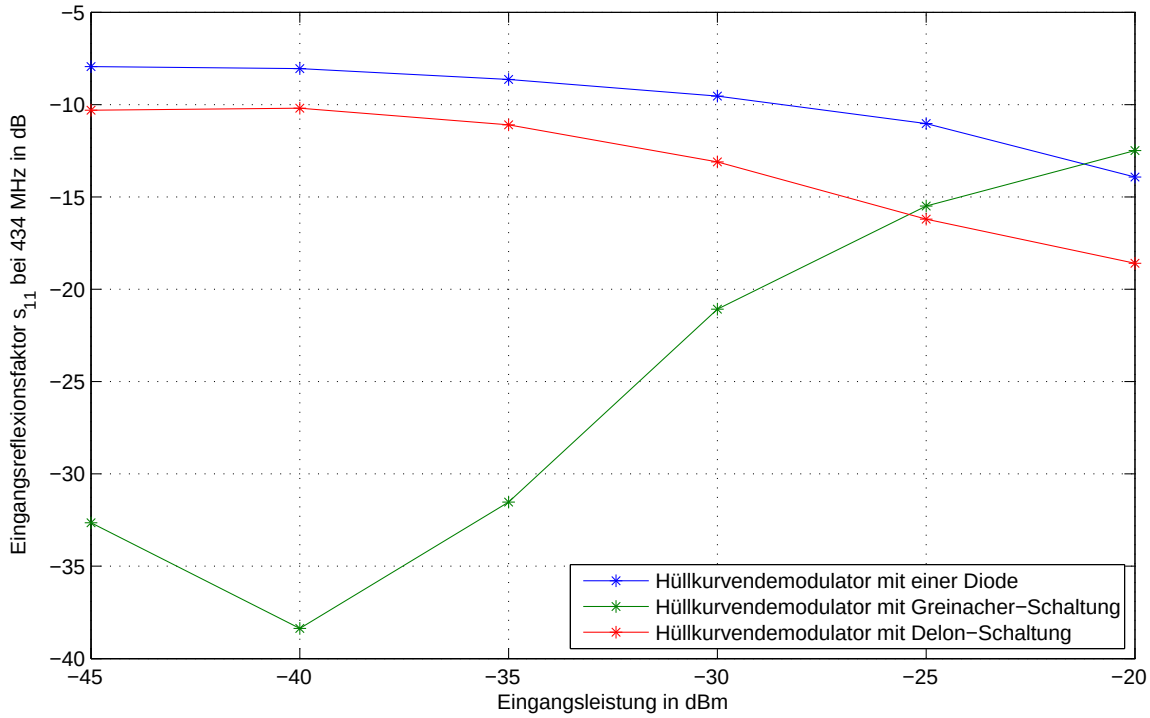


Abbildung 3.48: Eingangsreflexionsfaktoren der untersuchten Hüllkurvendemodulatorschaltungen in Abhängigkeit der Eingangsleistung

3.5 Erprobung des Wakeup

Nachdem im vorherigen Abschnitt verschiedene Hüllkurvendemodulatoren untersucht wurden, soll im Folgenden das gesamte Wakeup erprobt werden. Dabei liegt das Hauptaugenmerk auf dem LF Wakeup Receiver *AS3930* von *austriamicrosystems* [5]. Ein Foto des nachfolgend beschriebenen Versuchsaufbaus ist im Anhang in Abbildung F.3 zu sehen.

Für die Versuchsreihe wurde ein kontinuierliches Wakeup-Signal mit der Basisstation und dem entwickelten Transceiver-Modul aus Abschnitt 3.2.1 erzeugt und mit einer Gesamtleistung von $10\,\mathrm{dBm}$ über die Antenne abgestrahlt. In einem Abstand von etwa $20\,\mathrm{cm}$ wurde das Signal über die Versuchsplatine mit der Schleifenantenne aus Abschnitt 3.3 empfangen und über ein *SMA*-Kabel nacheinander in die Hüllkurvendemodulatorschaltungen eingespeist. Wie die folgende Abbildung 3.49 zeigt, wurde der LF Wakeup Receiver Chip auf der Versuchsplatine der Schleifenantenne platziert.

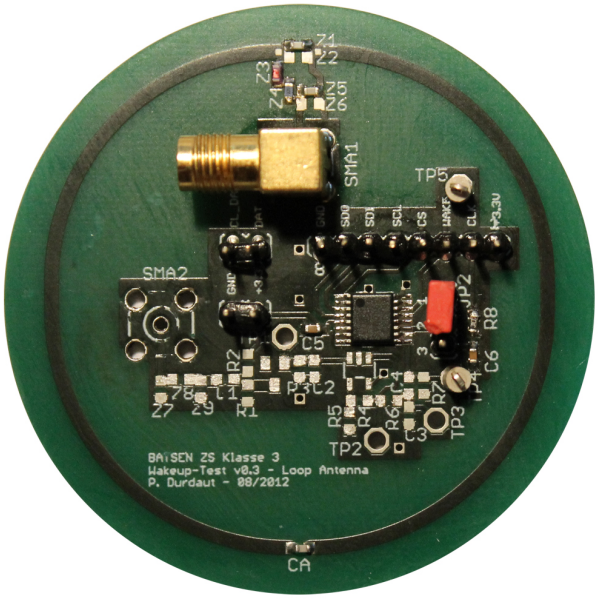
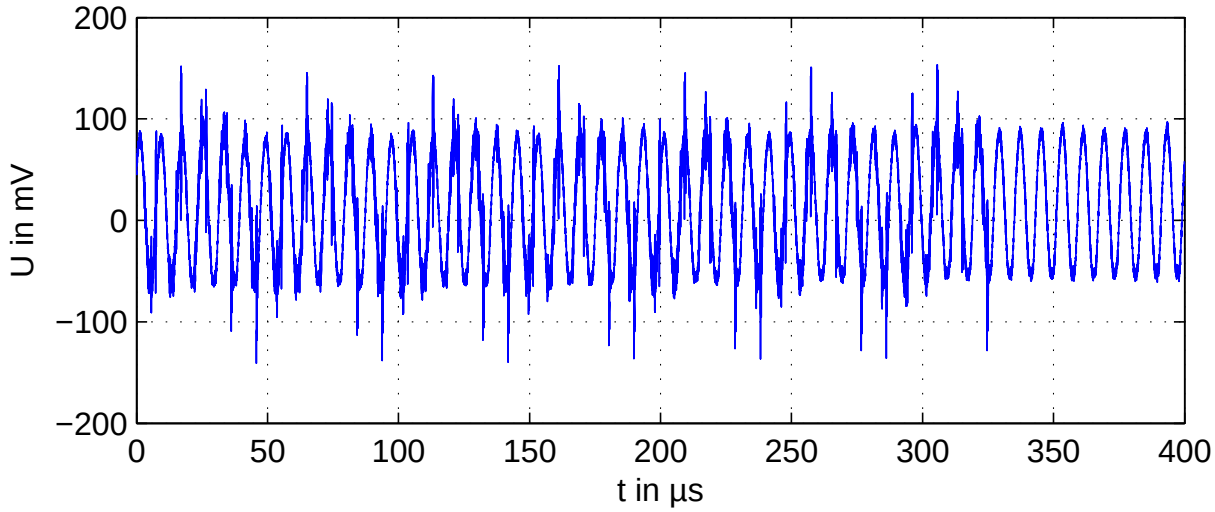


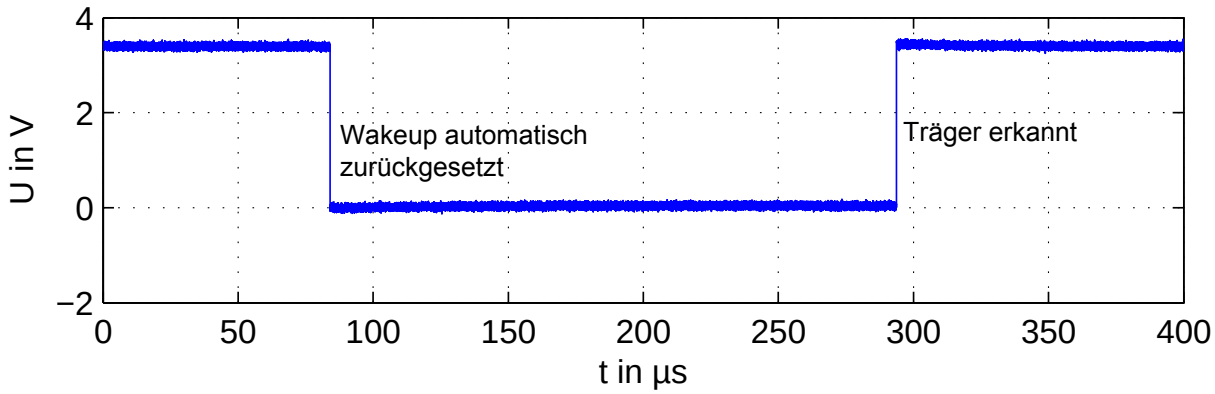
Abbildung 3.49: Versuchssensor mit Schleifenantenne und LF Wakeup Receiver *AS3930* von *austriamicrosystems*

Über ein Kabel wurde das demodulierte Trägersignal der Frequenz $125\,\mathrm{kHz}$ von den untersuchten Hüllkurvendemodulatorschaltungen an den Eingang des LF Wakeup Receiver Bausteins (TP4) übertragen. Über ein zweites *MSP430-169STK* Entwicklungsboard und einer Verbindung zwischen diesem und der Versuchsplatine wurde der *AS3930* über die *SPI*-Schnittstelle so programmiert, dass dieser eine reine *LF*-Trägererkennung durchführt. Wird ein solcher erkannt, wird dies über eine steigende Flanke am digitalen Ausgang *WAKE* signalisiert und diese nach $50\,\mathrm{ms}$ automatisch zurückgesetzt. Sowohl der digitale Ausgang als auch das demodulierte LF-Signal am Eingang des Chips wurden parallel über ein Oszilloskop aufgenommen. In

der Abbildung 3.5 sind diese beiden Signale zu sehen, wobei die Demodulation mit nur einer Schottky-Diode erfolgte.



(a) 125 kHz-Trägersignal am Eingang des LF Wakeup Receivers *AS3930*



(b) *WAKE*-Ausgangspin des LF Wakeup Receivers *AS3930*

Abbildung 3.50: Signalverläufe am Eingang (a) und am Ausgang (b) des LF Wakeup Receivers bei der Erprobung der Wakeup-Schaltung

Das Vorhandensein eines *LF*-Trägersignals führt nach ca. $200\,\mu s$ zu einer steigenden Flanke am *Wake*-Ausgangspin, die später als Interruptquelle für den Mikrocontroller dienen soll. Die vielen in Abbildung 3.50(a) zu sehenden Spannungsspitzen sind Störungen durch den provisorischen Versuchsaufbau, die trotzdem nicht zu Fehlfunktionen des Wakeup-Chips führen. Untersuchungen mit den anderen vorgestellten Hüllkurvendemodulatorschaltungen zeigten, dass sogar stärkste Verzerrungen (Delon-Schaltung) zu keinem Fehlverhalten führen. Zu beobachten war lediglich, dass die für die Trägererkennung benötigte Zeit auf mehrere hundert Mikrosekunden ansteigen kann. Es wird somit entschieden, dass auf dem Zellensensor der Hüllkurvendemodulator mit nur einer Schottky-Diode eingesetzt werden soll.

3.6 Packetbasierte Kommunikation

In Abschnitt 2.4 wurde entschieden, den Transceiver *Si4431* von *Silicon Labs* auf dem Zellensensor einzusetzen. Über diesen soll eine packetorientierte Kommunikation mit der Basisstation realisiert werden, wobei in dieser der Transceiver *CC1101* von *Texas Instruments* (vgl. Transceiver-Modul in Abschnitt 3.2.1) verwendet wird.

In der Bezeichnung *packetorientiert* sind einige wichtige Eigenschaften enthalten, die für einen Zellensensor der Klasse 3 von Bedeutung sind und nachfolgend erläutert werden. Durch das sogenannte *Packet handling*, das beide Transceiver unterstützen, wird die Programmierung insofern erleichtert, als dass ein- und ausgehende Daten lediglich aus den transceivereigenen *FIFO*-Ringspeichern ausgelesen bzw. in diese geschrieben werden müssen. Weitere Bestandteile des Paketes, wie *PREAMBLE* , *SYNC*-Bytes oder eine Prüfsumme (vgl. Datenpakete der Zellensensoren der Klasse 2 in [25]) werden beim Senden automatisch angefügt bzw. nach dem Empfang automatisch entfernt. Außerdem unterstützen die Transceiver Adressierungsmöglichkeiten, mit denen ein Transceiver beim Datenempfang selbstständig überprüfen kann, ob der Mikrocontroller über eingetroffene Daten benachrichtigt werden soll, oder die Daten verworfen werden sollen. Auch die Auswertung der *CRC*-Prüfsumme (cyclic redundancy check) kann von den Transceivern in Hardware erfolgen, sodass der Mikrocontroller direkt über fehlerhafte Daten informiert werden kann, ohne selber eine entsprechende Berechnung durchführen zu müssen. Alle diese Mechanismen erleichtern die Programmierung und entlasten den Mikrocontroller, der für die Steuerung des Transceivers zuständig ist.

3.6.1 Transceiver-Modul für *Si4431* von *Silicon Labs*

Um die angestrebte packetorientierte Kommunikation zwischen der Basisstation und einem Transceiver des Typs *Si4431* von *Silicon Labs* [52] erproben zu können wurde, nach Vorbild des für die Basisstation entwickelten Transceiver-Moduls aus Abschnitt 3.2.1, eine zweite Versuchsplatine entwickelt.

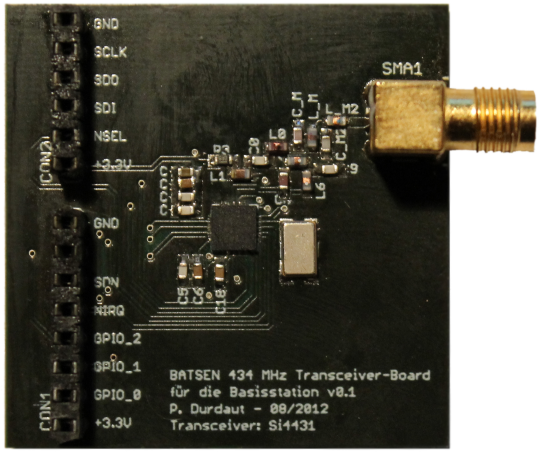


Abbildung 3.51: Transceiver-Modul mit Transceiver *Si4431* von *Silicon Labs*

Diese ist ebenfalls für den Anschluss an ein *MSP430-169STK* Entwicklungsboard von OLIMEX Ltd. vorgesehen, da die Versuchsplatine so keinen eigenen Mikrocontroller benötigt, wodurch der Aufwand sinkt. Die Entwicklung der Schaltung erfolgte gemäß des Referenzdesigns aus dem Datenblatt des Bausteins [52]. Die Bauteilwerte für Filter und die Impedanzanpassung an $50\,\Omega$ wurden der Applikationsanweisung [53] entnommen. Der Schaltplan sowie das Platinenlayout sind im Anhang unter B.5 bzw. C.5 zu finden.

3.6.1.1 Bauelemente

Auf dem Transceiver-Modul wurden die folgenden elektrischen Bauelemente verwendet:

- *UHF*-Transceiver *Si4431* von *Silicon Labs* [52]
- $30\,\mathrm{MHz}$ Quarz *7B-30.000MEEQ-T* von *TXC* [69]
- *SMD*-Induktivitäten der *0603HP-Serie* von *Coilcraft* [14]
- Standard Chip-Kondensatoren und Chip-Widerstände in *SMD*-Bauform (0603)

3.6.2 Erprobung

Für die Erprobung wurde die Schleifenantenne aus Abbildung 3.12 als Sende- und Empfangsantenne verwendet und über ein *SMA*-Kabel mit dem Transceiver-Board aus Abbildung 3.51 verbunden. Die Transceiver auf beiden Seiten wurden zunächst wie folgt konfiguriert:

- Trägerfrequenz: $434\,\mathrm{MHz}$
- Sendeleistung: $10\,\mathrm{dBm}$ (*CC1101*) bzw. $13\,\mathrm{dBm}$ (*Si4431*)
- Datenrate: $40\,\mathrm{kBit/s}$ ohne Manchester-Kodierung
- *OOK*-Modulation
- Automatisches Packet Handling
- 4 *PREAMBLE* Bytes
- 4 *SYNC* Bytes: 0x12 0x09 0x12 0x09
- 5/10 *DATA*-Bytes (Downlink/Uplink)
- Prüfsummengenerierung und Überprüfung deaktiviert

Die Software wurde so geschrieben, dass Datenpakete auf Knopfdruck gesendet und empfangene Daten auf dem LC-Display des jeweiligen Entwicklungsboards angezeigt wurden. Dabei zeigte sich, dass die Kommunikation in beiden Richtungen auf mehrere Meter innerhalb des Labors einwandfrei funktionierte. Sobald jedoch die Prüfsummengenerierung über die *DATA*-Bytes und die Überprüfung auf der Gegenseite aktiviert wurde, funktionierte die Kommunikation nicht mehr. Die weitere Untersuchung dieses Phänomens erfolgt im nächsten Abschnitt.

3.6.3 Zyklische Redundanzprüfung (CRC)

Um fehlerhafte Datenpakete von korrekt übertragenen Datenpaketen unterscheiden zu können, werden den zu übertragenen Daten zusätzlich sogenannte Prüfdaten angehängt, die im Sender aus den Rohdaten generiert werden. Im Empfänger werden nach demselben Verfahren ebenfalls Prüfdaten berechnet, die dann mit den empfangenen Prüfdaten verglichen werden. Abhängig davon, ob eine Übereinstimmung festgestellt wurde, werden die empfangenen Daten als korrekt übertragen interpretiert oder verworfen.

Ein gebräuchliches Verfahren zur Berechnung der Prüfdaten ist die zyklische Redundanzprüfung (engl. cyclic redundancy check). Diese beruht auf einer binären Polynomdivision, welche über rückgekoppelte Schieberegister und einfache XOR-Gatter besonders leicht in Hardware zu implementieren ist. Der Divisor der Polynomdivision wird als Generatorpolynom bezeichnet und muss in Sender und Empfänger übereinstimmen. Das Generatorpolynom der beiden Transceiver *CC1101* und *Si4431* wird als *IBM-CRC-16* bezeichnet [54] und lautet [63]

$$x^{16} + x^{15} + x^2 + 1.$$

Aufgrund der Übereinstimmung der Generatorpolynome müssten beide Transceiver kompatibel zueinander sein. Um den Fehler in der Kommunikation zu finden, wurden beide Transceiver so konfiguriert, dass sie die gleichen fünf *DATA*-Bytes 0x00 0x01 0x02 0x03 0x00 senden und beide Sendesignale am Oszilloskop aufgezeichnet (siehe Abbildungen 3.52 und 3.53).

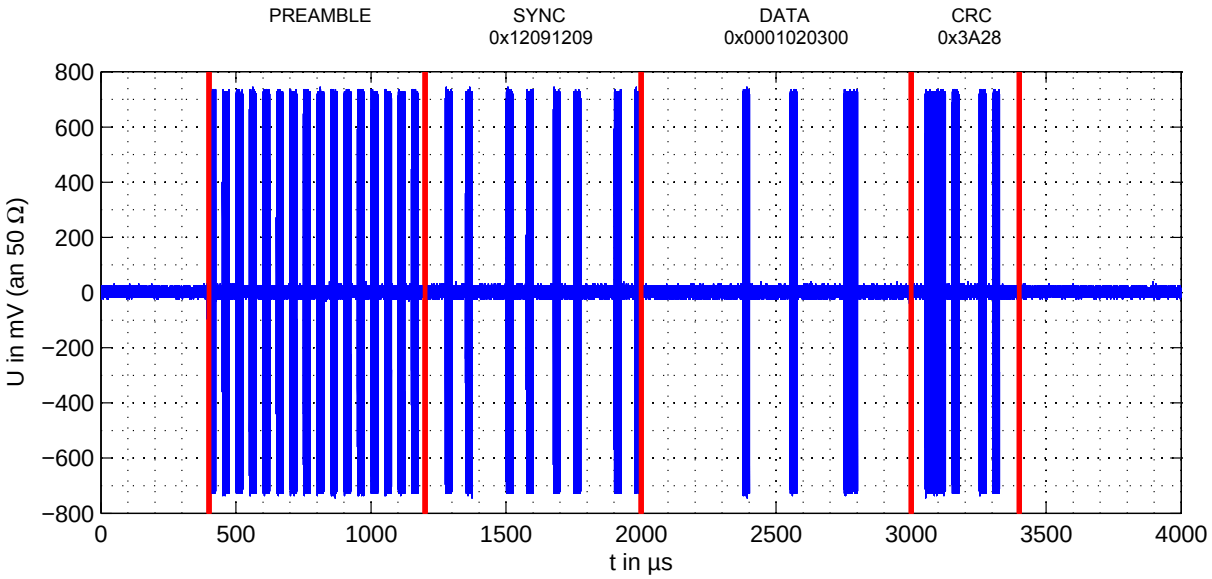


Abbildung 3.52: Gesendetes Datenpaket mit dem Transceiver *CC1101* bei aktivierter Prüfdatengenerierung

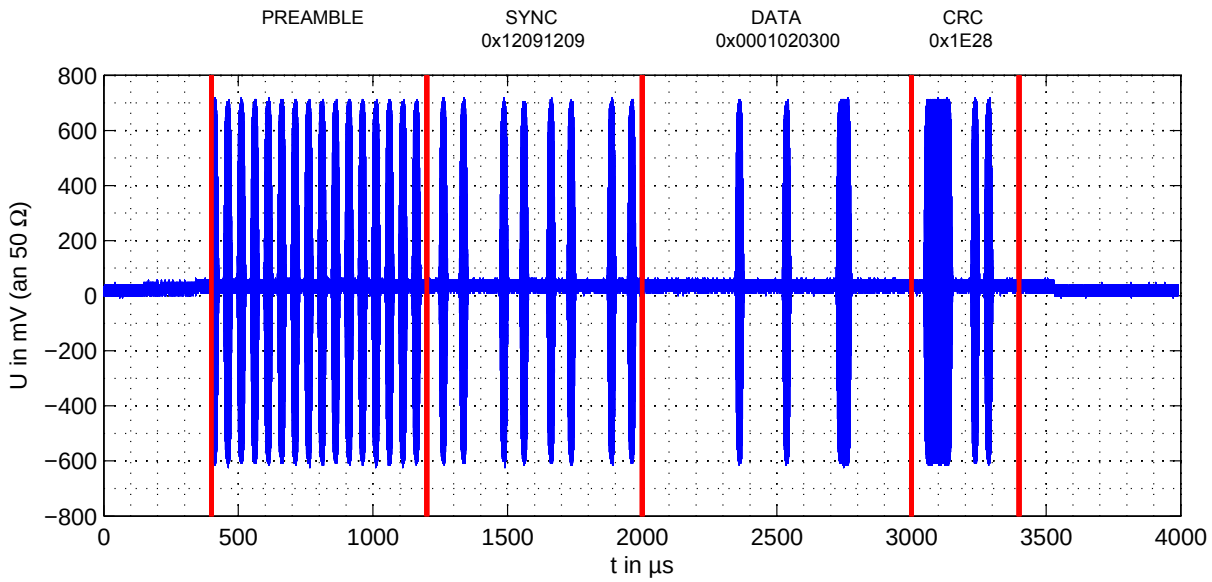


Abbildung 3.53: Gesendetes Datenpaket mit dem Transceiver *Si4431* bei aktivierter Prüfdatengenerierung

Die Messungen zeigen, dass sich, trotz des identischen Generatorpolynoms und derselben fünf *DATA*-Bytes, die generierten *CRC*-Daten unterscheiden. Um die Ursache dafür zu ermitteln, wurde in *C* ein PC-Programm geschrieben, das Prüfdaten für das oben angegebene Generatorpolynom berechnen kann (siehe Anhang D.2.1). Mit Hilfe dieses Programms wurde festgestellt, dass prinzipiell beide Transceiver die *CRC*-Berechnung korrekt durchführen. Der Unterschied kommt durch die unterschiedliche Vorinitialisierung der für die Berechnung verwendeten Schieberegister zustande. Der Transceiver *CC1101* verwendet Schieberegister, die mit Einsen vorinitialisiert sind [63], wohingegen der *Si4431* scheinbar mit Nullen vorinitialisierte Schieberegister verwendet. Diese Tatsache wurde durch den Hersteller *Silicon Labs* nicht dokumentiert.

3.6.4 Lösungsmöglichkeiten

Da sich herausgestellt hat, dass aufgrund eines kleinen Details die beiden Transceiver *CC1101* und *Si4431* nicht vollständig kompatibel zueinander sind, muss eine praktikable Alternative gefunden werden.

Eine Möglichkeit wäre die *CRC*-Prüfdaten auf beiden Seiten durch den Mikrocontroller berechnen zu lassen, und diese manuell an das Datenpaket anzuhängen. So wäre eine Überprüfung auf Datenintegrität beim Empfang trotzdem möglich. Zudem könnte die bisher entstandene Software für den *Si4431* (Anhang D.1.3) weiterverwendet werden. Eine Neuentwicklung direkt mit einer Behelfslösung zu beginnen, soll jedoch vermieden werden.

Alternativ könnte auch die Basisstation zusätzlich zu dem *CC1101* mit einem *Si4431* ausgestattet werden, sodass ersterer das Wakeup-Signal erzeugt und zweiterer die packetorientierte

Kommunikation mit den Zellensensoren übernimmt. Das würde allerdings wieder zusätzlichen Hardwareaufwand bedeuten.

Am einfachsten ist der Einsatz des *CC1101* sowohl in der Basisstation als auch auf den Zellensensoren. So ist die *CRC*-Berechnung problemlos in Hardware möglich und weitere, möglicherweise vorhandene, aber noch nicht aufgetretene Kompatibilitätsprobleme, sind damit im Vorfeld ausgeschlossen. Vorteilhaft ist, dass mit dem *CC1101* Datenraten von bis zu $250\,\mathrm{kBit/s}$ möglich sind, welche die des *Si4431* um das 6.25-fache übersteigt.

4 Schaltungsentwicklung

4.1 UHF-Transceiver

Wie sich während den praktischen Voruntersuchungen in Abschnitt 3.6 herausgestellt hat, ist der Transceiver der Basisstation (*CC1101* von *Texas Instruments*) nicht kompatibel zu dem Transceiver *Si4431* von *Silicon Labs*, welcher ursprünglich (siehe auch Abschnitt 2.4) für den Zellensensor vorgesehen wurde. Deshalb wurde entschieden den Transceiver *CC1101* (vgl. Abschnitt 3.6.4) auch auf dem Zellensensor einzusetzen.

Bei der Entwicklung des Transceiver-Moduls für die Basisstation in Abschnitt 3.2.1, hat sich bereits gezeigt, dass sich mit den Vorgaben des Referenzdesigns[1] und des Datenblatts [66] gute Ergebnisse erzielen lassen, weshalb diese Vorgaben wieder berücksichtigt werden sollen.

Die Referenzbeschaltung des Transceivers enthält neben einigen Kondensatoren zur Filterung und Stabilisierung der digitalen und analogen Versorgungsspannungen des Transceivers eine Vielzahl reaktiver Bauelemente. Insgesamt zehn Induktivitäten und Kondensatoren werden als Symmetrierglied und als Impedanzanpassungsnetzwerk benötigt. Das Symmetrierglied (Balun) wird zur Wandlung zwischen dem symmetrischen bzw. differentiellen *UHF*-Ein- und Ausgang des Transceivers und dem asymmetrischen Antennensignal benötigt. Das Impedanzanpassungsnetzwerk sorgt für einen maximalen Leistungsfluss zwischen der Antenne (bzw. dem Antennenumschalter) mit der Eingangsimpedanz $50\,\Omega$ und dem Transceiver. Der Hersteller *Johanson Technology, Inc.* bietet speziell für den Transceiver *CC1101* einen nur $2.0\,\mathrm{mm}$ x $1.25\,\mathrm{mm}$ großen Baustein [26] an, der ohne weitere externe Bauelemente das Symmetrierglied und die Impedanzanpassung integriert. Der Einsatz dieses passiven Bauelements führt zu einer nicht zu vernachlässigenden Platzersparnis. Laut [65] ergeben sich durch diesen jedoch eine geringfügig verminderte Ausgangsleistung und eine verminderte Unterdrückung von Harmonischen. Da auf dem zu entwickelnden Zellensensor mit dem Durchmesser $6\,\mathrm{cm}$ nicht mit einem erheblichen Platzmangel zu rechnen ist und zuvor performante Ergebnisse mit den diskreten Bauelementen erzielt werden konnten, wird auf den Einsatz der kostenintensiveren integrierten Schaltung verzichtet.

Aus Kostengründen soll auf dem Zellensensor nicht der Quarz *C3E-26.000-12-1010-X* [1] eingesetzt werden, der bereits auf dem Transceiver-Modul der Basisstation verwendet wurde, sondern eine preisgünstigere Variante des Typs *CTS405C11A26M00000* [15]. Dieser erfüllt mit einer Frequenztoleranz von $\pm 10\,\mathrm{ppm}$ und einer Lastkapazität von $C_L = 10\,\mathrm{pF}$ ebenfalls die Anforderungen des Transceivers [66]. Der interne Oszillator des *CC1101* ist so ausgelegt, dass

[1] http://www.ti.com/litv/zip/swrr046a

zwei zusätzliche Lastkondensatoren (C_{30} und C_{31}) benötigt werden. Die Dimensionierung dieser erfolgt gemäß der Formel aus dem Datenblatt.

$$C_L = \frac{1}{\frac{1}{C_{30}} + \frac{1}{C_{31}}} + 2.5\,\mathrm{pF}$$

Entsprechend dieser werden die Lastkondensatoren zu $C_{30} = C_{31} = 15\,\mathrm{pF}$ festgelegt.

Für den Betrieb des Transceivers wird neben einer Versorgungsspannung von $3.3\,\mathrm{V}$ ein Mikrocontroller benötigt, der einen integrierten *SPI*-Controller für den Datenaustausch mit dem Transceiver besitzt. Außerdem werden am Mikrocontroller drei weitere digitale, interruptfähige Eingange benötigt, über die die frei konfigurierbaren Ausgänge *GDO0*, *GDO1* und *GDO2* des Transceivers verbunden werden können. Über diese kann der Mikrocontroller beispielsweise über den Empfang eines Datenpaketes benachrichtigt werden. Zukünftig ist es außerdem denkbar, den internen RC-Oszillator des Mikrocontrollers mit einem abgeleiteten Takt des präzisen Transceiver-Quarzes zu kalibrieren.

Die Beschaltung des Transceivers ist im Anhang B.1 auf Seite 4 des Zellensensor-Schaltplans zu finden.

4.2 Antenne und UHF-Umschalter

Als Antenne für die bidirektionale Kommunikation mit der Basisstation und für den Empfang des Wakeup-Signals soll die *magnetische Antenne* bzw. *kleine Schleifenantenne* verwendet werden, die bereits in Abschnitt 3.3 erfolgreich erprobt wurde.

Damit diese sowohl für den Transceiver als auch für die Wakeup-Schaltung genutzt werden kann, ist ein sogenannter Antennenumschalter oder *UHF*-Umschalter (engl. RF switch) nötig. Verwendet werden soll der *UHF*-Umschalter mit der Bezeichnung *ADG918* von *Analog Devices* [2]. Dieser besitzt einen Port, der sich über einen digitalen *CMOS*-Steuerungeingang durch den Mikrocontroller mit einem von zwei weiteren Ports verbinden lässt. Die Eingangsimpedanzen betragen jeweils $50\,\Omega$ bei einer Rückflussdämpfung von mindestens $17\,\mathrm{dB}$. Äußerst gering ist mit kleiner als $1\,\mu\mathrm{A}$ der benötigte Strombedarf sowie die Einfügedämpfung (engl. insertion loss) mit etwa $0.5\,\mathrm{dB}$.

Die Verwendung des *UHF*-Umschalters bedeutet, dass insgesamt drei Impedanzanpassungsnetzwerke auf dem Zellensensor benötigt werden, um die Schleifenantenne, die Wakeup-Schaltung und den Transceiver an die $50\,\Omega$-Eingangsimpedanzen des Umschalters anzupassen.

Die Beschaltung der Schleifenantenne und des *UHF*-Umschalters ist im Anhang B.1 auf Seite 3 des Zellensensor-Schaltplans zu finden.

4.3 Wakeup-Schaltung

Die Wakeup-Schaltung besteht aus einem Hüllkurvendemodulator und dem LF Wakeup Receiver *AS3930* von *austriamicrosystems* [5]. Wie in den Abschnitten 3.4.6 und 3.5 bereits entschieden wurde, soll der Hüllkurvendemodulator mit einer Schottky-Diode des Typs *HSMS-2850* von *Avago Technologies* [8] auf dem Zellensensor verwendet werden (vgl. Abschnitt 3.4.3).

Aufgrund der hohen Empfindlichkeit des LF Wakeup Receivers wird der Operationsverstärker als nichtinvertierender Verstärker aller Voraussicht nach nicht zwingend notwendig sein. Da dieser jedoch Teil der Tiefpassfilterung ist und eine Veränderung der Schaltung nach der Erprobung zu diesem Zeitpunkt vermieden werden soll, wird der Operationsverstärker *MCP6071* von *Microchip* [35] in dieser Version des Zellensensors übernommen. Für Messungen während der Inbetriebnahme des Zellensensors kann dieser zudem hilfreich sein. Um den Energiebedarf während des Schlafzustands des Sensors weiter zu optimieren, kann auf den Operationsverstärker in zukünftigen Versionen des Zellensensors verzichtet werden.

Die Konfiguration des LF Wakeup Receivers erfolgt über *SPI*, wie auch bei dem Transceiver. Da es sich bei *SPI* um einen seriellen Datenbus handelt, kann dieser von dem Transceiver und dem LF Wakeup Receiver gemeinsam im Zeitmultiplex genutzt werden. Dies erfordert für beide Bausteine eine zusätzliche Chip-Select Leitung, über die der Mikrocontroller, als *SPI*-Master, die Kommunikation mit den Bausteinen aktiviert. Darüberhinaus muss der Mikrocontroller einen interruptfähigen Eingang besitzen, über den die *WAKE*-Leitung des LF Wakeup Receivers den Mikrocontroller aktivieren kann.

4.4 Ladungsbalancierung

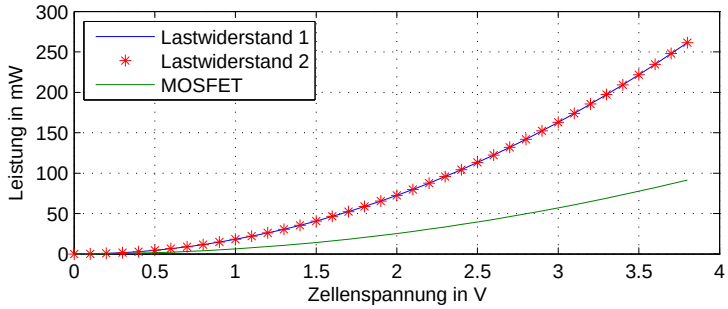
Um unterschiedliche Ladungsniveaus der einzelnen Zellen eines Akkumulators ausgleichen zu können, wird eine Ladungsbalancierung benötigt. Die einfachste Möglichkeit ist die passive Ladungsbalancierung, bei der Zellen mit höheren Ladungsmengen denen mit geringeren Ladungsmengen angepasst werden, indem der Energieverbrauch ersterer vervielfacht wird.

Da bei diesem Verfahren der Ladezustand aller Zellen bekannt sein muss, ist eine Steuerung der passiven Ladungsbalancierung nur durch die Basisstation möglich. Voraussetzung ist dementsprechend ein Downlink-Kanal von der Basisstation zu den Zellensensoren. Auf den Sensoren der Klasse 2 [25] wurde bereits eine passive Ladungsbalancierung realisiert, bei der zwei Lastwiderstände über zwei Transistoren eingeschaltet werden können. Diese Schaltung soll im Wesentlichen übernommen werden.

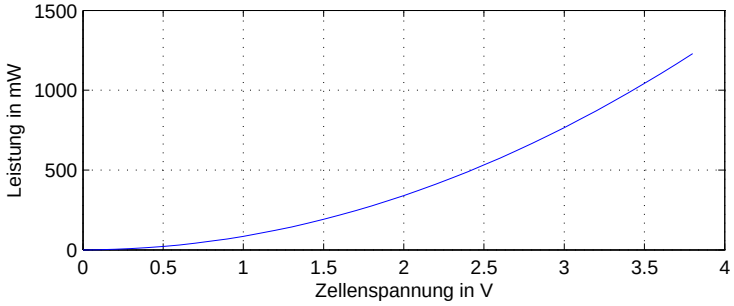
Für den Betrieb innerhalb der $LiFePO_4$-Batteriezelle (vgl. Abschnitt 2.1.1) ist es wichtig, dass der maximal zulässige Strom durch die Transistoren und die Lastwiderstände, sowie die in diesen Bauelementen umgesetzte Leistung eingehalten werden, damit die Wärmeentwicklung nicht zu groß wird. Verwendet werden *MOSFETs* des Typs *2N7002* von *NXP* [39] mit einem maximal zulässigen Drain-Source-Strom von $300\,\mathrm{mA}$ und einer maximal zulässigen Ver-

lustleitung von $830\,\mathrm{mW}$. Als Lastwiderstände[2] werden Chip-Widerstände der *SMD*-Bauform 1206[3] eingesetzt, in denen maximal $500\,\mathrm{mW}$ umgesetzt werden dürfen. Die Parallelschaltung beider Nebenstrompfade in Sourceschaltung ist im Anhang B.1 auf Seite 6 des Zellensensor-Schaltplans zu finden.

Die beiden folgenden Abbildungen zeigen die Verlustleistung der Bauelemente eines Nebenstrompfades sowie die gesamte Verlustleistung beider Nebenstrompfade in Abhängigkeit der Zellenspannung. In dem Arbeitspereich der Zellenspannungen von $1.2\,\mathrm{V}$ bis $3.8\,\mathrm{V}$ (vgl. Abschnitt 2.1.2) liegt die umgesetzte Leistung zwischen $123\,\mathrm{mW}$ und $1230\,\mathrm{mW}$.



(a) Umgesetzte Leistungen in einem Nebenstrompfad



(b) Umgesetzte Leistung in beiden Nebenstrompfaden

Abbildung 4.1: Umgesetzte Leistungen in den Nebenstrompfaden zur passiven Ladungsbalancierung in Abhängigkeit der Zellenspannung

Für das Einschalten der passiven Ladungsbalancierung bzw. für die Ansteuerung der Gates der *MOSFETs* wird ein *GPIO* des Mikrocontrollers benötigt.

4.5 Temperatursensor

Zur Messung der Temperatur auf dem Zellensensor bzw. innerhalb der Batteriezelle, die neben der Zellenspannung ebenfalls für die Bestimmung des *SOC* und des *SOH* benötigt wird, soll

[2] *CRM1206-FX-10R0ELF* von *Vishay*
[3] $3.2\,\mathrm{mm}$ x $1.6\,\mathrm{mm}$

der Temperatursensor *TMP102* von *Texas Instruments* [61] verwendet werden, der bereits auf den Zellensensoren der Klasse 2 eingesetzt wurde [25].

Der Temperatursensor besitzt eine Temperaturauflösung von $0.0625\,°\mathrm{C}$ bei einer Genauigkeit von $0.5\,°\mathrm{C}$ im Bereich von $-25\,°\mathrm{C}$ bis $+85\,°\mathrm{C}$. Die Ruhestromaufnahme beträgt lediglich $7\,\mu\mathrm{A}$ bis $10\,\mu\mathrm{A}$. Die erforderliche äußere Beschaltung des Temperatursensors ist mit zwei passiven Bauelementen minimal (vgl. Seite 6 des Zellensensor-Schaltplans im Anhang B.1).

Die gemessene Temperatur wird sensorintern über einen 12-Bit Delta-Sigma-A/D-Umsetzer gewandelt und digital über ein *Two-Wire Serial Interface* bzw. I^2C zur Verfügung gestellt. Somit sollte der verwendete Mikrocontroller in Hardware einen I^2C-Controller besitzen.

4.6 Mikrocontroller

Kern des Zellensensors soll ein Mikrocontroller sein. Dieser ist u.A. für die Erfüllung der folgenden Aufgaben notwendig:

- Steuerung des Antennenumschalters
- Konfiguration des LF Wakeup Receivers über *SPI*
- Reaktion auf Wakeup Interrupt des LF Wakeup Receivers
- Konfiguration des Transceivers über *SPI*
- Datenaustausch der empfangenen und zu sendenden Daten mit dem Transceiver über *SPI*
- Steuerung des mikrocontrollerinternen A/D-Umsetzers
- Ansteuerung des Temperatursensors über I^2C
- Zwischenspeicherung der Messwerte, insbesondere während Hochstromereignissen
- Auswertung der Downlink-Kommandos
- Ansteuerung der Transistoren für die passive Ladungsbalancierung
- Ansteuerung der *LEDs* (für Entwicklungszwecke)

Bisher wurden auf allen Sensoren des Forschungsprojekts *BATSEN* Mikrocontroller der *MSP430*-Serie von *Texas Instruments* verwendet. Diese besitzen einen optimierten Stromverbrauch sowie verschiedene Energiesparzustände, sodass diese besonders für batteriebetriebene Anwendungen geeignet sind. Aufgrund den bisher gesammelten Erfahrungen soll wieder ein *MSP430*-Mikrocontroller eingesetzt werden. Das bietet zudem den Vorteil, dass bisher entstandener Programmcode weiterhin verwendet werden kann.

Es wurde entschieden, dass der 16-Bit-Mikrocontroller mit der Bezeichnung *MSP430F235* [62], der auch auf den Sensoren der Klasse 2 [25] erfolgreich eingesetzt wurde, verwendet

werden soll. Dieser bietet Kommunikationsschnittstellen für *SPI* und I^2C, einen 12-Bit A/D-Umsetzer mit interner Referenzspannung für die Messung der Zellenspannung und 16-Bit Timer, die für die Generierung von Zeitstempeln verwendet werden können. Im Hinblick auf die spätere Umsetzung der Kommandostrukturen für die bidirektionale Kommunikation zwischen der Basisstation und der Zellensensoren wurde gezielt keine Optimierung des Mikrocontrollers vorgenommen, sondern eine leistungsfähige Variante gewählt. Mit $16\,\mathrm{kB}$ Flash-Speicher und $2\,\mathrm{kB}$ *RAM* bietet der Mikrocontroller ausreichend Programm- und Arbeitsspeicher. Ein großzügiger Arbeitsspeicher ist besonders während Hochstromereignissen für die Zwischenspeicherung der Messdaten sinnvoll, bevor diese an die Basisstation übertragen werden.

Weiterhin vorteilhaft ist der interne *DCO* (Digital Controlled Oscillator), der Taktraten zwischen $1\,\mathrm{MHz}$ und $16\,\mathrm{MHz}$ unterstützt, welche per Software einstellbar sind. So wäre es denkbar, beispielsweise während Hochstromereignissen, die Taktrate zu erhöhen und während eines langsameren Messbetriebs wieder zu verringern, um den Strombedarf zu senken. Bei einer Taktrate von $1\,\mathrm{MHz}$ beträgt der Strombedarf des Mikrocontrollers (ohne weitere Peripherie) lediglich etwa $400\,\mu\mathrm{A}$. Im tiefsten Schlafzustand (*LPM4*), in den der Mikrocontroller wechseln soll, während auf das Wakeup-Signal gewartet wird, beträgt der Strombedarf weniger als $1\,\mu\mathrm{A}$. Durch die Nutzung des *DCO* wird zudem kein externer Quarz benötigt, welcher zum einen kostenintensiv ist und sich zum anderen nicht in einen Chip integrieren lässt. Aufgrund der höheren Ungenauigkeit des internen Oszillators, als der eines Quarzes muss dieser während des Betriebs jedoch wahrscheinlich kalibriert werden, damit die Basisstation die empfangenen Messwerte den richtigen Messzeitpunkten zuordnen kann. Dies könnte mit Hilfe des Transceivers geschehen, welcher ein digitales Taktsignal, abgeleitet von dem präzisen Transceiver-Quarz, bereitstellen kann.

Die elektrische Beschaltung des Mikrocontrollers ist auf Seite 1 des Zellensensor-Schaltplans im Anhang B.1) zu finden.

4.7 Spannungsversorgung

Die aktiven Bauelemente

- Antennenumschalter
- Operationsverstärker
- LF Wakeup Receiver
- Transceiver
- Mikrocontroller
- Temperatursensor

des konzipierten Zellensensors wurden so ausgewählt, dass diese mit einer einheitlichen Versorgungsspannung von $3.3\,\mathrm{V}$ betrieben werden können. Aufgrund der Forderung aus Abschnitt 2.1.2, nach der ein Einsatz des Zellensensors in verschiedenen Akkumulatortechnologien möglich sein soll, ergeben sich mögliche Zellenspannungen U_{cell} von $1.2\,\mathrm{V}$ bis $3.8\,\mathrm{V}$. Um aus diesen eine stabilisierte Versorgungsspannung von $3.3\,\mathrm{V}$ zu erzeugen, ist ein Gleichspannungswandler notwendig, der sowohl zu geringe Eingangsspannungen ($U_{cell} < 3.3\,\mathrm{V}$) anheben und zu hohe Eingangsspannungen ($U_{cell} > 3.3\,\mathrm{V}$) verringern kann.

Ein solcher Step-Up/Step-Down-Wandler wurde bereits auf den Sensoren der Klasse 2 [25] erfolgreich eingesetzt. Der dort verwendete Spannungswandler *TPS61201* von *Texas Instruments* [60] soll weiterhin auch auf den neuen Zellensensoren verwendet werden.

Dieser kann ausgangsseitig, abhängig von der Eingangsspannung, einen ausreichenden Strom zwischen $300\,\mathrm{mA}$ und $500\,\mathrm{mA}$ liefern. Es sind Eingangsspannungen von $0.3\,\mathrm{V}$ bis $5.5\,\mathrm{V}$ möglich. Außerdem bietet der Spannungswandler eine *Power-Save*-Funktionalität, durch die der Wirkungsrad bei Ausgangsströmen von weniger als etwa $50\,\mathrm{mA}$ erheblich gesteigert wird [60]. Da nicht von einem benötigten Betriebsstrom von mehr als $50\,\mathrm{mA}$ auszugehen ist, wird die *Power-Save*-Funktionalität über die Beschaltung des Spannungswandlers dauerhaft eingeschaltet. Die gesamte Beschaltung ist auf Seite 2 des Zellensensor-Schaltplans im Anhang B.1 zu finden.

4.8 Blockschaltbild

Die folgende Abbildung 4.2 zeigt das Blockschaltbild des gesamten Zellensensors, wie dieser im folgenden Kapitel 5 realisiert werden soll.

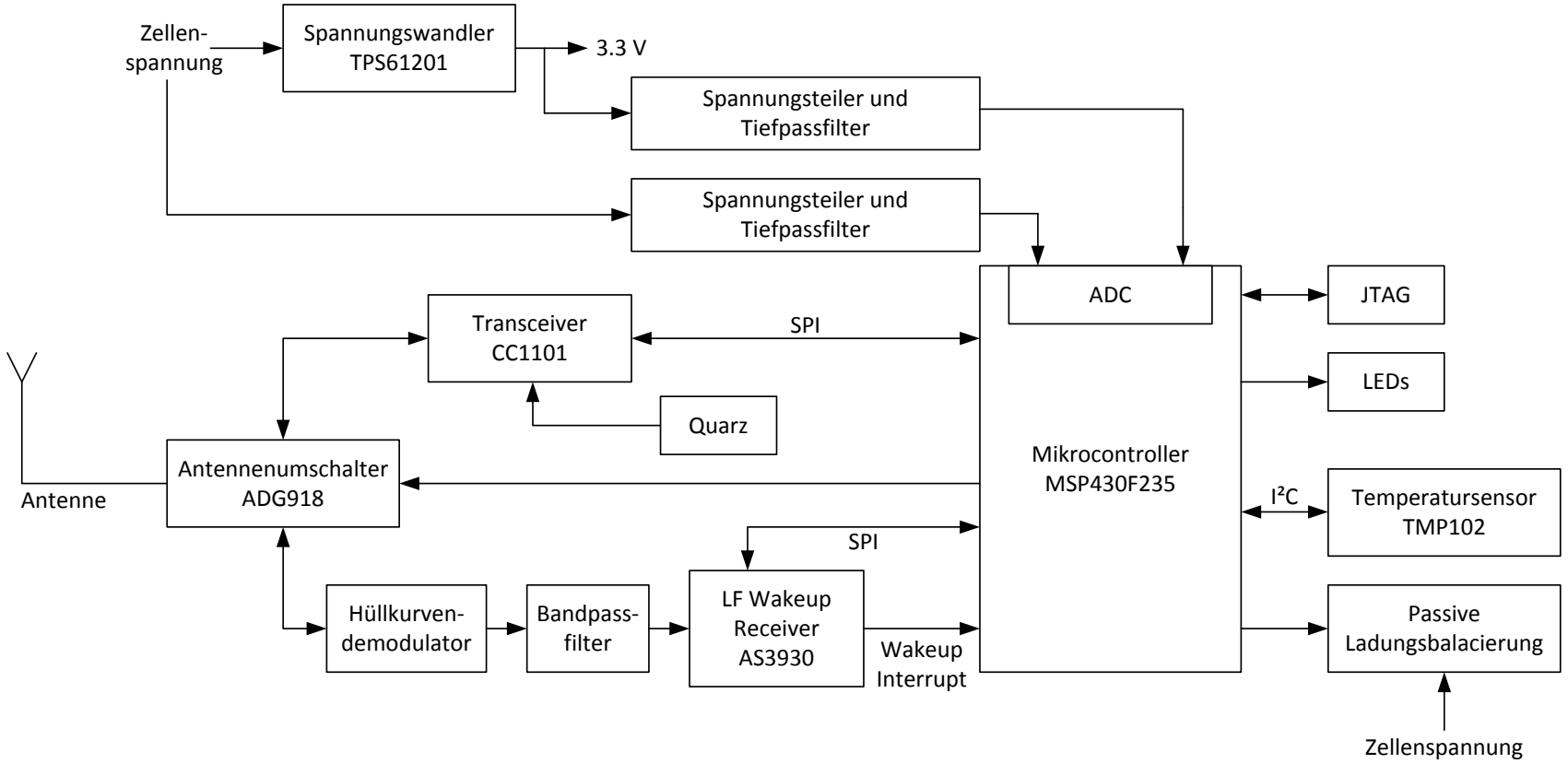


Abbildung 4.2: Blockschaltbild des Zellensensors

5 Aufbau und Erprobung

5.1 Aufbau des Zellensensors

Die Abbildung 5.1 zeigt einen, von insgesamt vier, vollständig bestückten Zellensensoren. Die Bestückung erfolgte größtenteils per Hand und aufgrund der Größe[1] der Bauteile überwiegend mit Hilfe eines Mikroskops. Lediglich einige der integrierten Schaltungen (Mikrocontroller, Transceiver, Spannungswandler) wurden in einem Reflow-Ofen [23] gelötet, da diese auf der Unterseite des Chip-Gehäuses eine Massefläche besitzen, dessen Verlötung mit einem herkömmlichen Lötkolben nicht möglich ist.

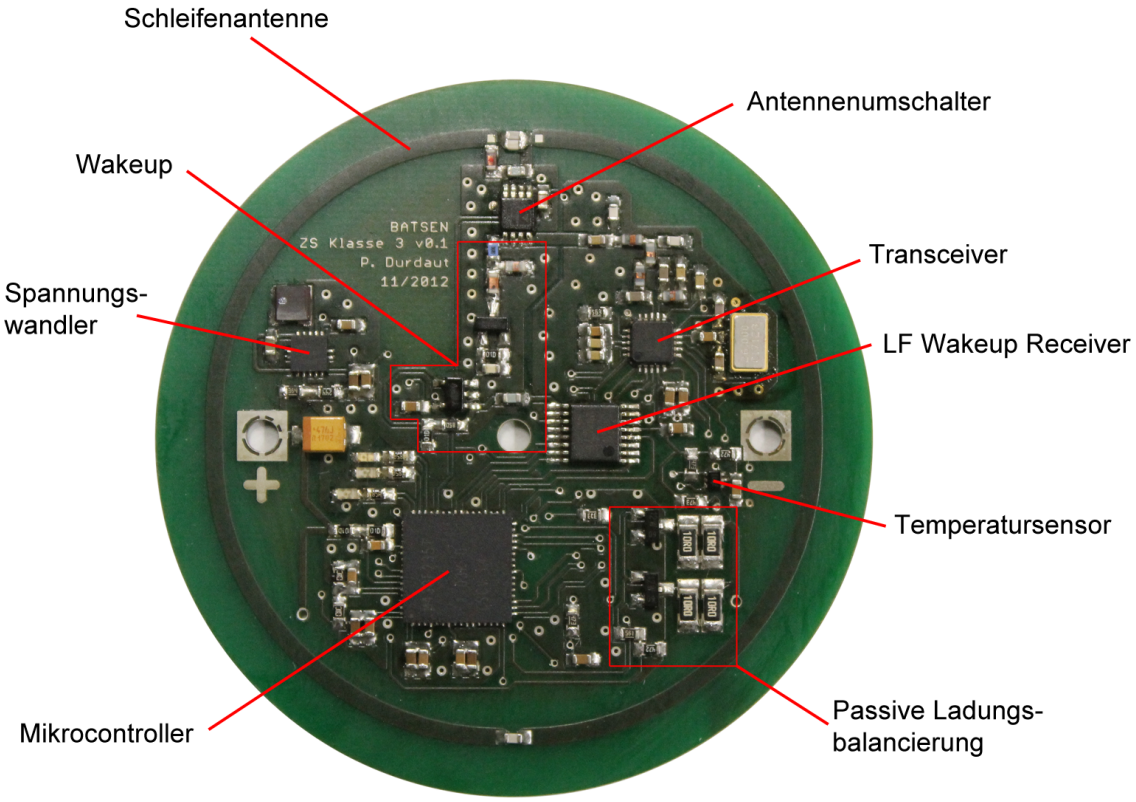


Abbildung 5.1: Vollständig bestückter Zellensensor

[1] Standardbauelemente in *SMD*-Bauform 0603 (1.6 mm x 0.8 mm)

5.1.1 Impedanzanpassung

Während den praktischen Voruntersuchungen an den Antennen und den Demodulatorschaltungen hat sich gezeigt, dass die Ergebnisse der Simulationen zu den Impedanzanpassungen in Microwave Office nicht ausreichend genau mit den realen Messungen übereinstimmen. Um geringe Reflexionsfaktoren bei $434\,\mathrm{MHz}$ zu erreichen, sind Impedanzanpassungen hoher Güte notwendig, welche deshalb zwangsläufig sehr schmalbandig sind. Es hat sich außerdem gezeigt, dass für die Impedanzanpassungen benötigte Kondensatoren Kapazitäten in der Größenordnung weniger Pikofarad (pF) benötigen und bereits Zehntel-Pikofarad zu Resonanzverschiebungen um mehrere Megahertz führen. Da parasitäre Kapazitäten in dieser Größenordnung liegen und abhängig von der geometrischen Anordnung der Bauelemente zueinander, den Materialeigenschaften der Leiterplatte und weiteren Parametern sind, ist damit zu rechnen, dass die bisher ermittelten Impedanzanpassungsnetzwerke keine optimalen Stehwellenverhältnisse ($1 < VSWR < 1.9$) auf dem Zellensensor ergeben werden.

Deshalb wurde eine zusätzliche Messplatine (Abbildung 5.2) erstellt, dessen Kupferflächen im Wesentlichen mit denen auf der eigentlichen Zellensensor-Platine übereinstimmen. Anstatt des Antennenumschalters sind auf dieser jedoch zwei *SMA*-Buchsen vorgesehen, um sowohl in die Schleifenantenne als auch in den Hüllkurvendemodulator mit dem Netzwerkanalysator hineinmessen zu können. Die mit dieser Messplatine ermittelten Impedanzanpassungsnetzwerke sollen auf den eigentlichen Zellensensoren eingesetzt werden.

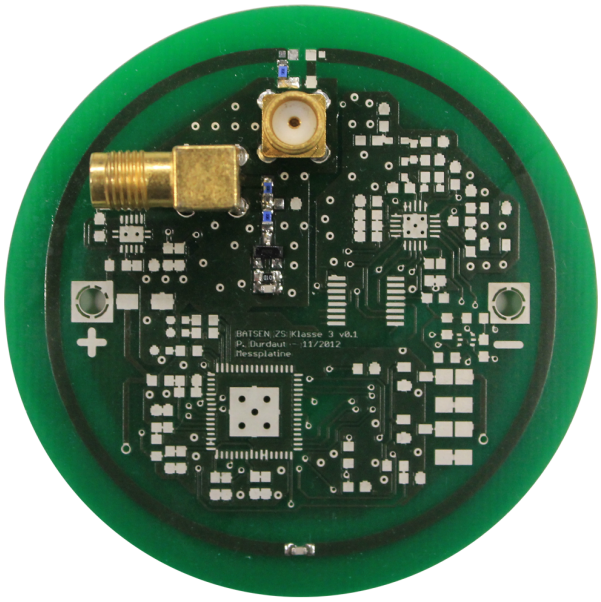


Abbildung 5.2: Zellensensor-Messplatine mit *SMA*-Buchsen zur Impedanzanpassung der Schleifenantenne und des Hüllkurvendemodulators

5.1.1.1 Schleifenantenne

Die beiden folgenden Abbildungen zeigen das Ergebnis der Impedanzanpassung der Schleifenantenne auf der Zellensensor-Messplatine. Im Vergleich zu der Impedanzanpassung der Schleifenantenne auf der Versuchsplatine in Abschnitt 3.3 mussten die Kapazitäten beider Kondensatoren leicht erhöht werden, um das Minimum des Eingangsreflexionsfaktors auf $434\,\mathrm{MHz}$ zu verschieben.

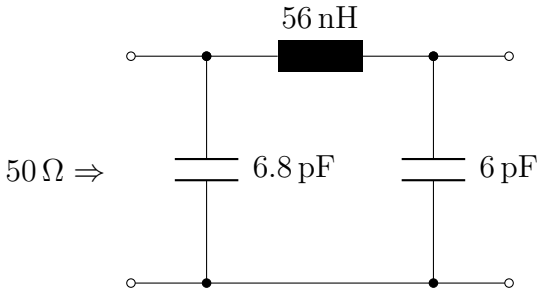


Abbildung 5.3: Netzwerk zur Impedanzanpassung der Schleifenantenne auf der Zellsensor-Messplatine an $50\,\Omega$

Der Wert des Eingangsreflexionsfaktors ist von $-14.5\,\mathrm{dB}$ auf $-11\,\mathrm{dB}$ angestiegen. Das entspricht einer akzeptablen Leistungsreflexion von etwa 8 %.

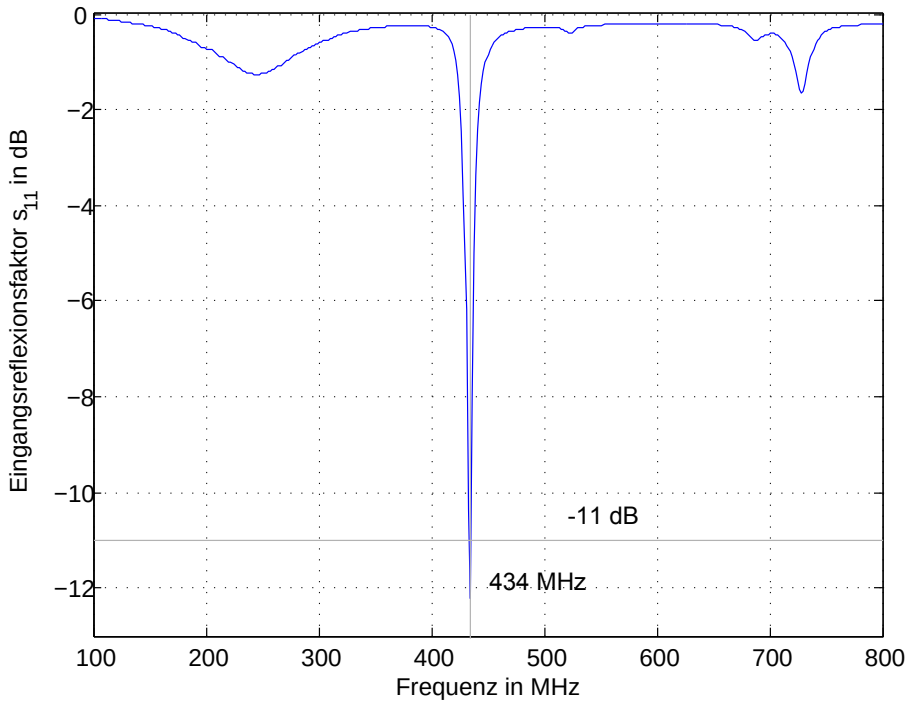


Abbildung 5.4: Angepasste Schleifenantenne auf der Zellensensor-Messplatine

5.1.1.2 Hüllkurvendemodulator

Die Impedanzanpassung des Hüllkurvendemodulators ist im Vergleich zu der, während den Voruntersuchungen entstandenen Versuchsplatine, weniger gut ausgefallen. Bei $434\,\mathrm{MHz}$ ist der Eingangsreflexionsfaktor mit $-9.1\,\mathrm{dB}$ größer als angestrebt. Die Leistungsreflexion beträgt etwa $12\,\%$. Um das Minimum, das bei etwa $424\,\mathrm{MHz}$ liegt, zu höheren Frequenzen zu verschieben, müsste die Längsinduktivität verringert werden. Die nächstkleinere Induktivität in dieser Reihe [14] liegt allerdings bei $150\,\mathrm{nH}$, welche das Minimum um fast $20\,\mathrm{MHz}$ verschiebt. Ein Wert von etwa $165\,\mathrm{nH}$ würde die Impedanzanpassung wahrscheinlich erheblich verbessern.

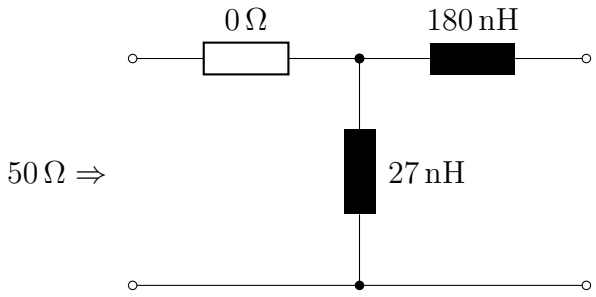


Abbildung 5.5: Netzwerk zur Impedanzanpassung des Hüllkurvendemodulators auf der Zellsensor-Messplatine an $50\,\Omega$

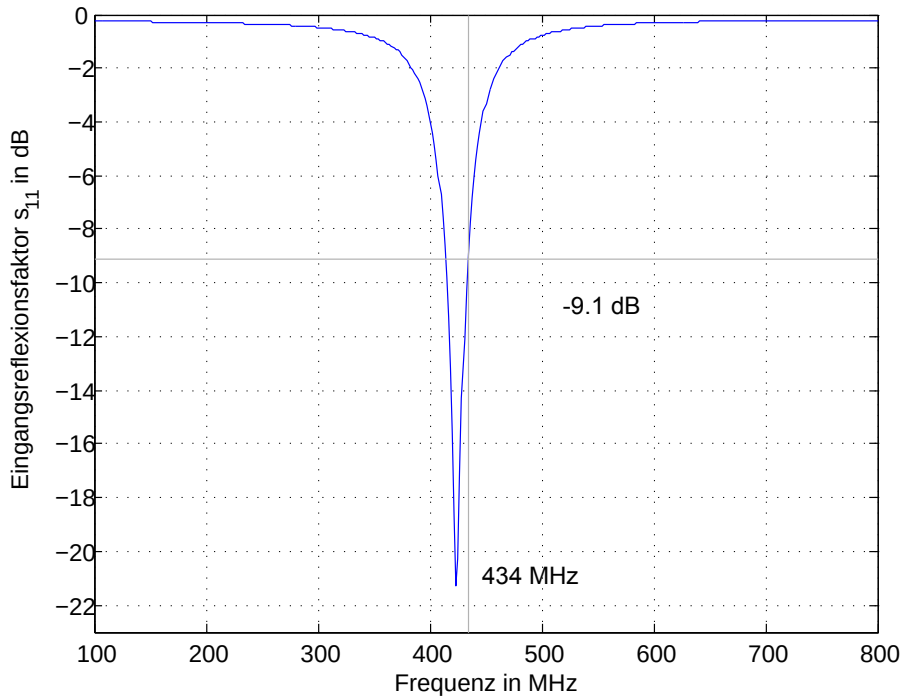


Abbildung 5.6: Angepasste Detektorschaltung auf der Zellensensor-Messplatine

5.2 Programmierung der Zellensensoren

Um auf den Zellensensoren keine zusätzlichen Steckerbuchsen vorsehen zu müssen, erfolgt die Programmierung der Sensoren über projekteigene Nadeladapter. Diese führen alle benötigten Leitungen der Programmier- und Debug-Schnittstelle (*JTAG*) auf gefederte Prüfnadeln, welche die aufsteckbaren Sensoren auf der Unterseite kontaktieren. Die Prüfnadeln wiederum sind über Drähte mit einer Buchsenleiste verbunden, sodass der Programmieradapter *MSP-FET430UIF* mit dem Nadeladapter verbunden werden kann.

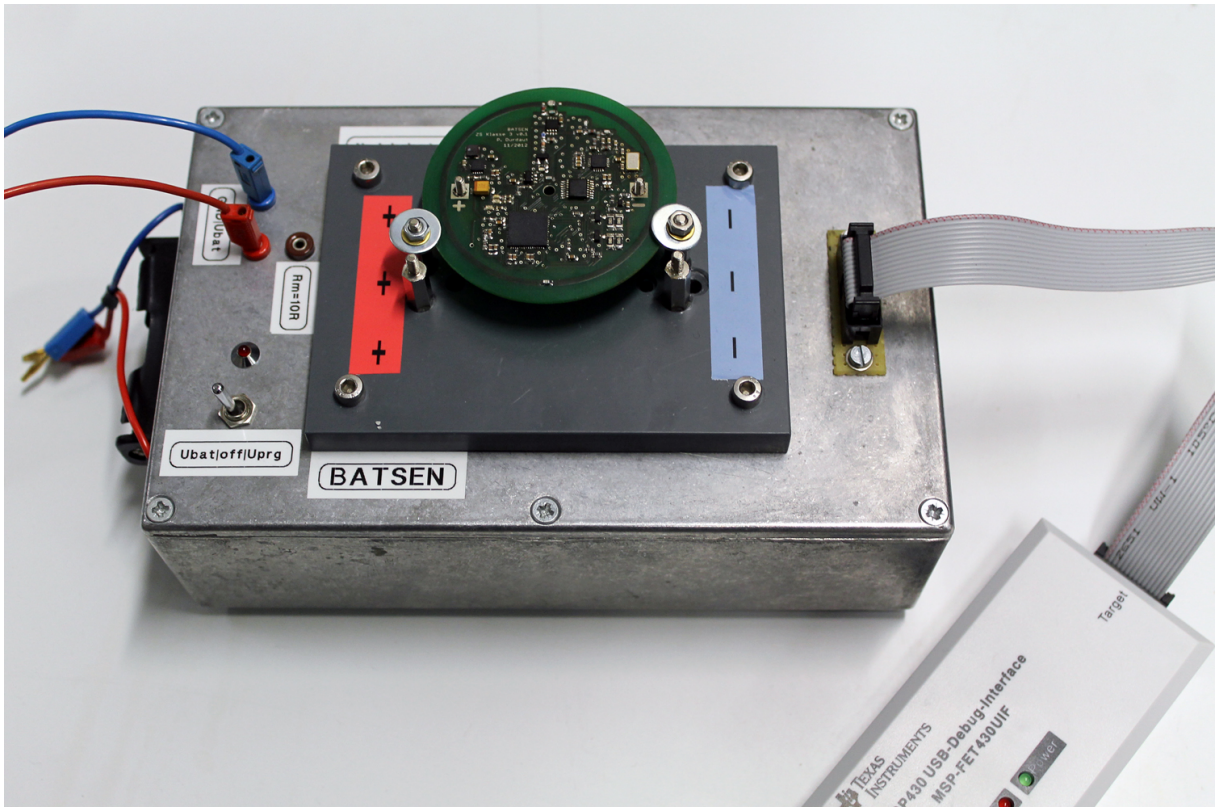


Abbildung 5.7: Nadeladapter zur Programmierung der Zellensensoren

5.3 Bestimmung der Zellendämpfung und der Übetragungsreichweiten

Nachdem sich in kleineren Probeversuchen gezeigt hat, dass sowohl das Wakeup eines Zellensensors als auch die bidirektionale Kommunikation zwischen Basisstation und Zellensensor grundsätzlich funktioniert, blieb zu untersuchen, wie sich das Verhalten ändert, wenn der Zellensensor innerhalb der Batteriezelle aus Abschnitt 2.1.1 platziert wird.

Dazu wurde in einem ersten Versuch ermittelt, wie stark ein Funksignal im Nahfeld durch die Aluminiumhülse der Rundzelle bedämpft wird. Hierfür wurde ein Zellensensor (Sensor 3) so konfiguriert, dass dieser ein dauerhaftes Trägersignal mit einer Sendeleistung von $10\,\mathrm{dBm}$ aussendet. Im Abstand von etwa $10\,\mathrm{cm}$ wurden die Empfangspegel mit der Antenne der Basisstation (Abbildung 3.1) am Spektrumanalysator bei offener und geschlossener Batteriezelle aufgenommen.

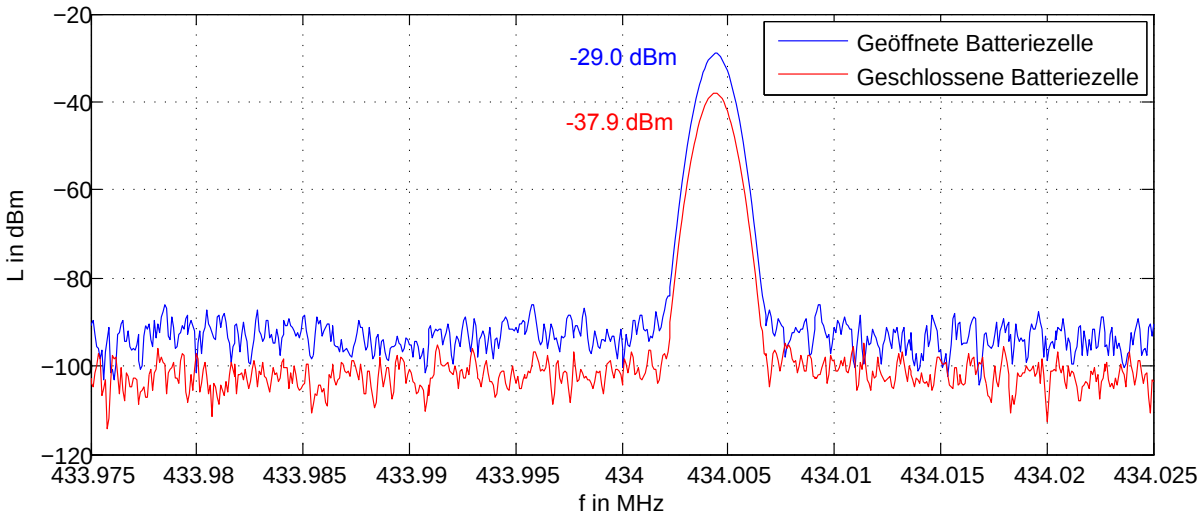


Abbildung 5.8: Empfangspegel an der Basisstation eines vom Zellensensor gesendeten Trägersignals bei offener und geschlossener Batteriezelle

Auf der Abbildung 5.8 sind die Empfangspegel für diese beiden Fälle abgebildet. Die Differenz von etwa $9\,\mathrm{dB}$ entspricht der Dämpfung der Aluminiumhülse. Die Höhe des Empfangspegels bei geöffneter Batteriezelle von $-29\,\mathrm{dBm}$ passt gut zu den Ergebnissen der Erprobung der Schleifenantenne in Abschnitt 3.3.5. Auch dort zeigte sich, dass bei einer Kommunikation zwischen der Schleifenantenne und der Antenne der Basisstation auf kürzere Distanzen ($\approx 30\,\mathrm{cm}$) Empfangspegel von etwa $40\,\mathrm{dB}$ unter den Sendepegeln zu erwarten sind. Eine zusätzliche Bedämpfung der elektromagnetischen Strahlung von etwa $10\,\mathrm{dB}$ bei geschlossener Batteriezelle führt bei einer Sendeleistung von $10\,\mathrm{dBm}$ also zu Empfangspegeln von etwa $-40\,\mathrm{dBm}$. Bei der Erprobung des Hüllkurvendemodulators wurde eine Empfindlichkeit zwischen $-45\,\mathrm{dBm}$ und $-50\,\mathrm{dBm}$ festgestellt, sodass davon ausgegangen werden kann, dass auch in die Batteriezelle integrierte Zellensensoren über das Wakeup-Signal der Basisstation aufweckbar sind.

Nach der vorbereitenden Bestimmung der Dämpfung durch das Aluminiumgehäuse, wurde das Wakeup und die bidirektionale Kommunikation über den Transceiver getestet und die maximal möglichen Reichweiten ermittelt.

Zur Erprobung des Wakeup wurde der Zellensensor so programmiert, dass dieser nach der Initialisierung in den Ruhezustand wechselt, indem der Stromverbrauch minimal ist und auf ein Wakeup durch die Basisstation gewartet wird. Wird der Sensor aufgeweckt, sendet dieser eine Bestätigung über den Transceiver an die Basisstation zurück, sodass an dieser ausgewertet

werden kann, ob das Wakeup erfolgreich war. Dabei zeigte sich, dass die Impedanzanpassung, die auf der Zellensensor-Messplatine (Abschnitt 5.1) durchgeführt wurde, erfolgreich war bzw. auch zu vernünftigen Ergebnissen auf dem eigentlichen Zellensensor führt. Dass dies der Fall sein wird, konnte vorher nicht mit Sicherheit angenommen werden. Wie die Tabelle 5.1 zeigt, ist ein Wakeup des Zellensensors im Abstand von etwa $3\,\mathrm{m}$ zur Antenne der Basisstation erfolgreich. Befindet sich der Zellensensor innerhalb der Batteriezelle sinkt die maximale Distanz auf etwa 15 bis $20\,\mathrm{cm}$. In erster Linie ist entscheidend, dass das Wakeup auch innerhalb der Batteriezelle funktioniert. Die Entfernung dabei ist zwar nicht besonders groß, für eine Anwendung in Starterbatterien jedoch wahrscheinlich sogar ausreichend. Möglichkeiten zur Erhöhung der Reichweite werden im letzten Kapitel genannt.

Kommunikationstyp	Ausbreitungsbedingung	Mögliche Entfernung
Wakeup	Freiraum	$3\,\mathrm{m}$
	Sensor in Batterie	$15\,..\,20\,\mathrm{cm}$
Transceiver	Freiraum	$> 15\,\mathrm{m}$
	Sensor in Batterie	$3\,\mathrm{m}$

Tabelle 5.1: Übersicht über die ermittelten Übertragungsreichweiten

Aufgrund der aktiven Empfangseinheiten der Transceiver und der damit verbundenen größeren Empfindlichkeit, zeigten sich bei der Erprobung der bidirektionalen Kommunikation entsprechend größere Reichweiten. Ohne die dämpfende Wirkung der Aluminiumhülse funktionierte die Kommunikation sogar in $15\,\mathrm{m}$ Entfernung und durch eine Wand hindurch noch problemlos. Im Betrieb innerhalb der Batteriezelle sind Reichweiten von etwa $3\,\mathrm{m}$ möglich, die bei der Überwachung einer mehrzelligen Batterie bei weitem ausreichend sind.

5.4 Erprobung des Gesamtsystems

Einführend wurde in Abschnitt 1.4 bereits erwähnt, dass umfangreiche Software nötig ist, um aus dem in dieser Arbeit entwickelten Zellensensor einen vollwertigen Sensor der Klasse 3 zu realisieren. Obwohl der Schwerpunkt in dieser Arbeit in der Konzept- und der Hardwareentwicklung des Zellensensors liegt, soll für den grundsätzlichen Funktionsnachweis eine Erprobung des realisierten Gesamtsystems erfolgen.

Dazu wurde Software für die Basisstation und die Zellensensoren entwickelt, die aufgrund ihrer Modularität größtenteils wiederverwendbar ist. Diese Software, die Durchführung der Erprobung und die Resultate, werden im Folgenden näher beschrieben.

5.4.1 Versuchsaufbau

Den Versuchsaufbau zur Erprobung des Gesamtsystems zeigt die folgende Abbildung 5.9. Die vier aufgebauten Zellensensoren werden mit dem Zellspannungsgenerator aus der Arbeit [55]

mit einer sich zeitlich veränderlichen Betriebsspannung versorgt (sinusförmig mit der Frequenz $100\,\mathrm{mHz}$, der Amplitude $1\,\mathrm{V}$ und einem Offset von $4\,\mathrm{V}$). Diese Spannung wird auf Kommando der Basisstation von den Zellensensoren gemessen und an die Basisstation versendet. Von dort werden die Messdaten für eine weitere Auswertung seriell über RS-232 an den PC gesendet. Für einen Vergleich der Messwerte mit den an den Zellensensoren anliegen Spannungen werden diese zusätzlich parallel mit einem Oszilloskop aufgenommen. Zur Synchronisierung wird das Oszilloskop durch die Basisstation getriggert.

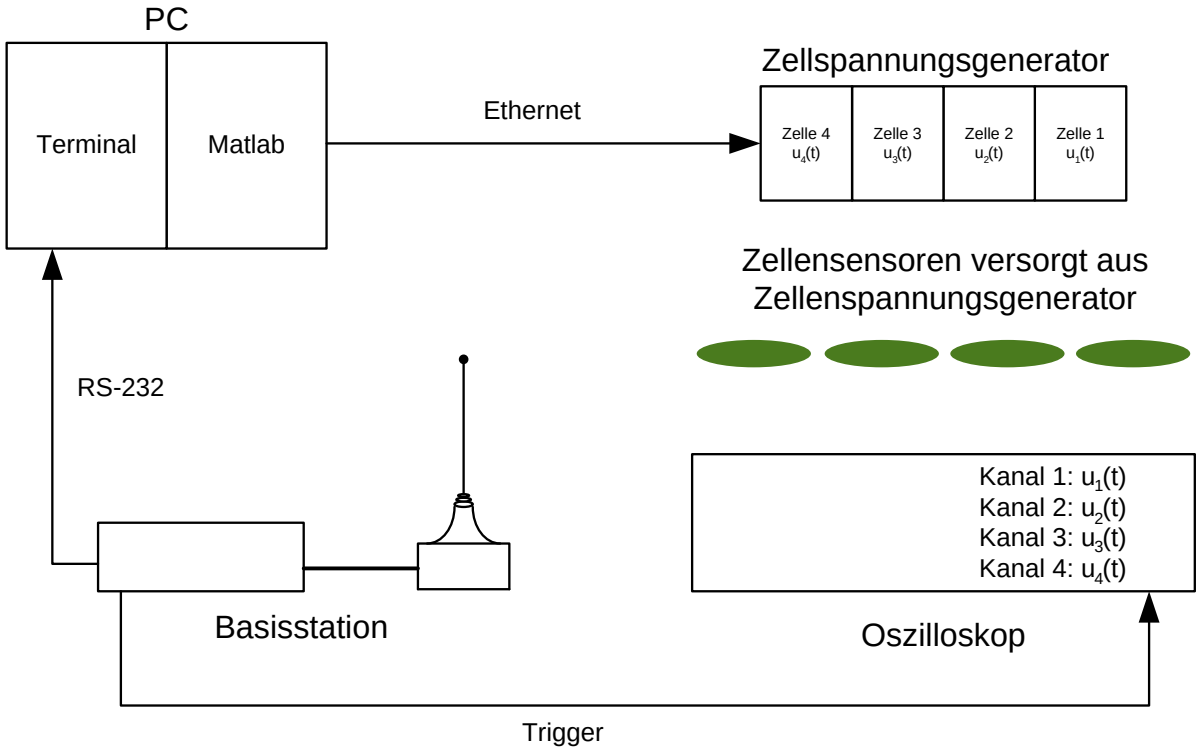


Abbildung 5.9: Messaufbau zur Erprobung des Gesamtsystems

5.4.2 Ablauf

Der Ablauf der Erprobung ist in Abbildung 5.10 schematisch dargestellt. Nach dem Einschalten der generierten Zellspannungen initialisieren sich die Zellensensoren und wechseln anschließend in den Ruhe- bzw. Schlafzustand. Aus diesem werden alle Zellensensoren zeitgleich durch das Wakeup-Signal der Basisstation aufgeweckt. Nacheinander wird jeder Zellensensor adressiert angesprochen, um den Status abzufragen. War das Wakeup erfolgreich, wird dies über eine Antwort an die Basisstation signalisiert.

Nachdem alle Zellensensoren empfangsbereit sind, wird im Sekundentakt ein Broadcast-Befehl gesendet, der alle Zellensensoren gleichzeitig einen Messwert aufnehmen lässt. Die Messwerte werden anschließend nacheinander von den Zellensensoren abgefragt und an die Basisstation übertragen.

Das Ende des Messvorgangs wird durch ein Broadcast-Kommando der Basisstation angekündigt, auf welches alle Zellensensoren zurück in den energiesparenden Schlafzustand wechseln.

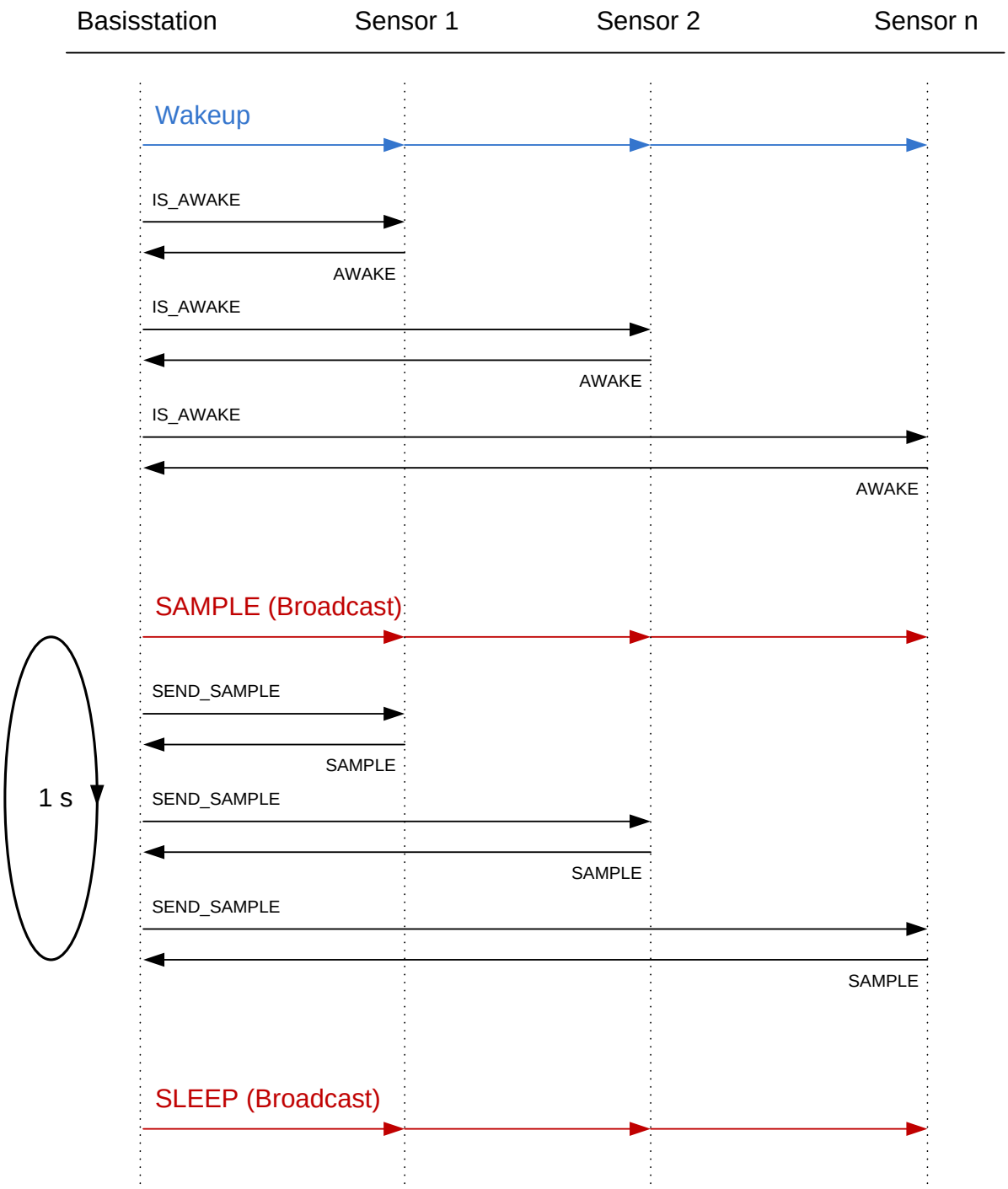


Abbildung 5.10: Ablauf der Kommunikation zwischen der Basisstation und den Zellensensoren bei der Erprobung des Gesamtsystems

5.4.3 Software

5.4.3.1 Zellensensor

Für die Erprobung des Gesamtsystems wurde ein Programm entwickelt, das sich in identischer Form[2] in dem Mikrocontroller jedes Zellensensors befindet. Die Abbildung 5.11 zeigt das Zustandsdiagramm eines Zellensensors. Sobald die Versorgungsspannung zur Verfügung steht, führt der Mikrocontroller einen *Reset* durch, der das Programm in einen definierten Zustand bringt. Nach der Initialisierung (*S_INIT*) des Mikrocontrollers und der weiteren digitalen Bausteine wechselt der Sensor in den Schlafzustand (*S_SLEEP*), in dem nur ein minimaler Betriebsstrom benötigt wird (Mikrocontroller im *LPM4*[3] und Transceiver ausgeschaltet). Sobald die Basisstation das Wakeup-Signal abstrahlt, wird der Sensor aktiviert und wechselt in den Zustand *S_RX*, in dem auf Befehle von der Basisstation gewartet wird. Für die Erprobung wurden vier Downlink-Kommandos implementiert:

- *IS_AWAKE*: Die Basisstation erwartet eine Antwort, um zu kontrollieren, ob das Wakeup erfolgreich war.
- *SAMPLE*: Die Basisstation fordert den Zellensensor auf, die aktuelle Zellenspannung zu wandeln.
- *SEND_SAMPLE*: Die Basisstation fordert die zuvor gewandelte Zellenspannung an.
- *SLEEP*: Die Basisstation fordert den Zellensensor auf, in den Schlafzustand zu wechseln.

Zu finden sind alle Programmdateien der Zellensensor-Software im Anhang in Abschnitt D.1.1. Die Software ist weitestgehend modular aufgebaut, um die Weiterentwicklung der Zellensensoren zu vereinfachen. So existieren die Programmdateien *adc12.c*, *adg918.c*, *as3930.c*, *balancing.c*, *cc1101.c* und *led.c*, die Funktionen zur Steuerung der gleichnamigen Peripherie bieten. Um diese nur in Einzelfällen bearbeiten zu müssen, existieren zu jeder dieser Programmdateien gleichnamige Header-Dateien, in welchen u.a. die *GPIO*-Konfigurationen des entsprechenden Moduls hinterlegt sind.

Die Programmdatei *main.c* enthält die Initialisierung und den Zustandsautomaten. Die Initialisierung des Mikrocontrollers besteht im Wesentlichen in der Konfiguration des mikrocontrollerinternen *DCO* (Digital Controlled Oscillator), bei dem es sich um einen RC-Oszillator handelt. Dieser ist aufgrund des geringeren Betriebsstroms auf eine Taktfrequenz von $1\,\mathrm{MHz}$ konfiguriert. Sollte in späteren Anwendungen mehr Rechenleistung benötigt werden, ist die Taktfrequenz des *MSP430F235* auf bis zu $16\,\mathrm{MHz}$ erhöhbar.

[2] Einziger Unterschied zwischen den Sensoren ist die Sensoradresse, die in *main.h* definiert ist.
[3] Tiefster Schlafzustand von *MSP430*-Mikrocontrollern

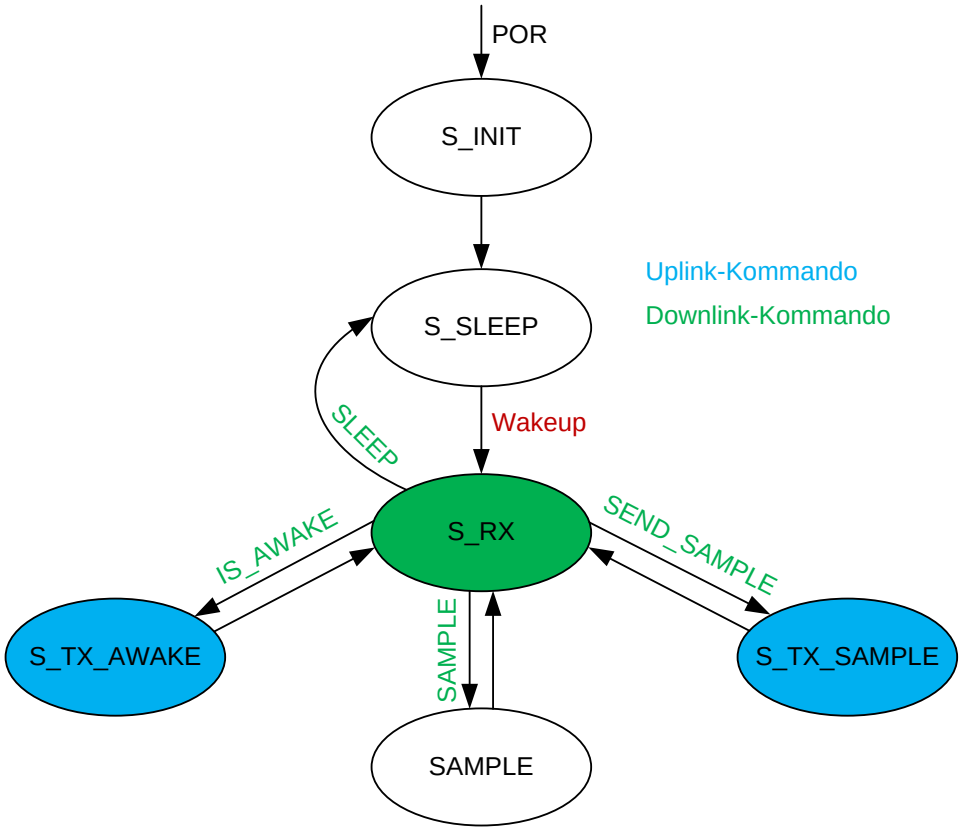


Abbildung 5.11: Zustandsdiagramm der Zellensensor-Software

5.4.3.2 Basisstation

Die Aufgabe der Basisstation, während der Erprobung des Gesamtsystems, ist die Koordinierung der Messungen durch die Zellensensoren mittels verschiedener Kommandos, die über den Transceiver abgesetzt bzw. empfangen werden. Das Zustandsdiagramm der Basisstation zeigt die Abbildung 5.12.

Nach dem Initialisierungsvorgang (*S_INIT*) wird das Wakeup-Signal für die Dauer von $10\,\mathrm{ms}$ versendet, um die Zellensensoren in den aktiven Empfangszustand zu versetzen. Im Anschluss daran wird nacheinander jedem Sensor das Kommando *IS_AWAKE* gesendet, auf das die Basisstation eine Antwort (*AWAKE*) erwartet (Timeouts sind im Zustandsdiagramm nicht berücksichtigt). Auf diese Weise kann die Basisstation abfragen, ob das Wakeup erfolgreich gewesen und der entsprechende Zellensensor bereit ist. Wie schon die Abbildung 5.10 zum Ablauf der Erprobung gezeigt hat, wird im Sekundentakt das Broadcast-Kommando *SAMPLE* an alle Sensoren zugleich verschickt. Das auf diesen Befehl hin aufgenommene digitale Äquivalent der Zellenspannung wird anschließend von der Basisstation nacheinander bei jedem Sensor angefragt (*S_TX_SEND_SAMPLE*) und im Zustand *S_RX_SAMPLE* empfangen. Dieser Vorgang wiederholt sich jede Sekunde, bis die konfigurierte Anzahl an Messwerten erreicht ist.

Bevor das Programm beendet wird, wird der Messvorgang mit dem Broadcast-Kommando *SLEEP* abgeschlossen, das alle Zellensensoren zurück in den energiesparenden Schlafzustand versetzt. Die Programmdateien für die Software der Basisstation sind im Anhang im Abschnitt D.1.2 zu finden.

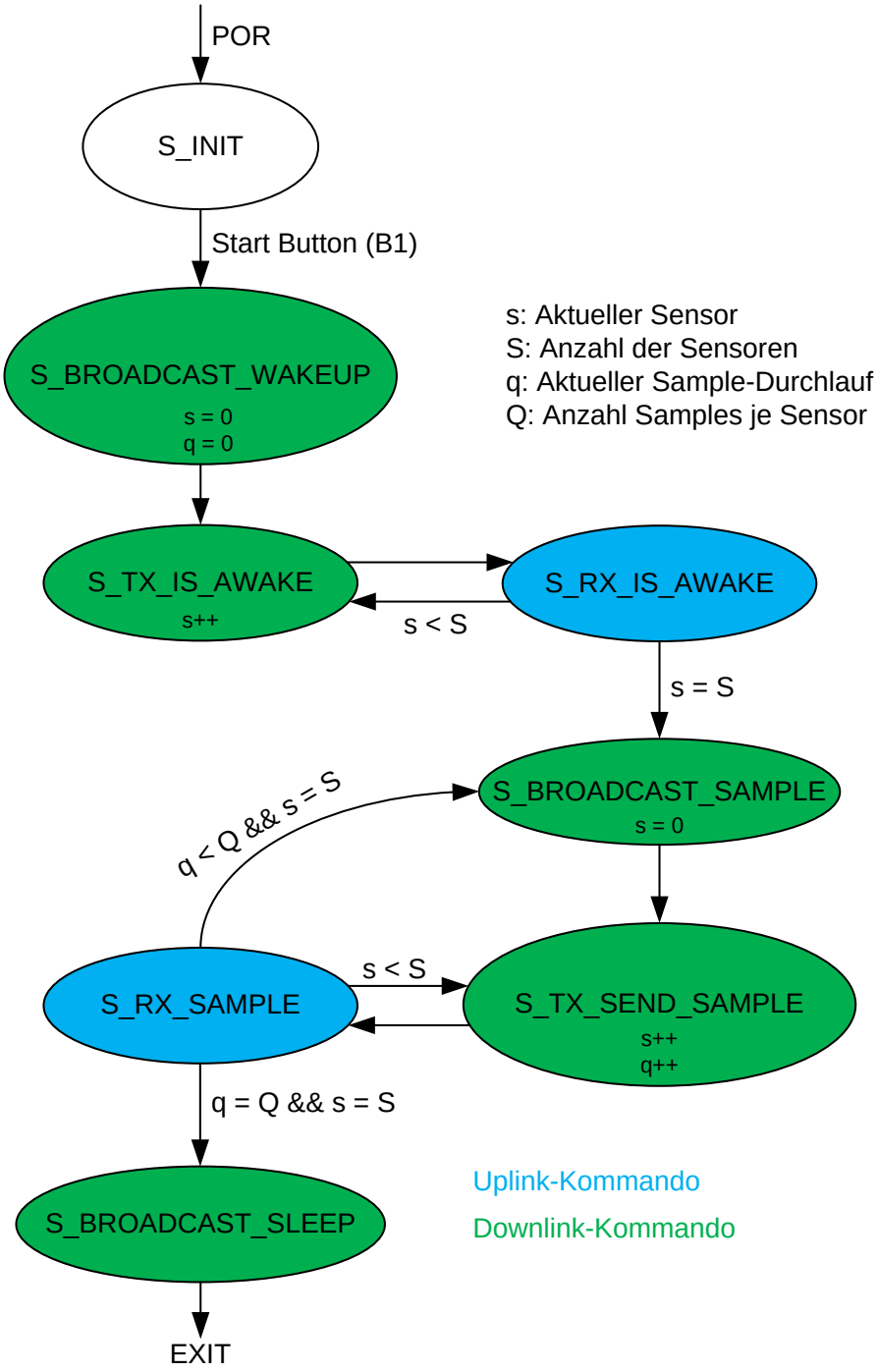


Abbildung 5.12: Zustandsdiagramm der Software der Basisstation zur Erprobung des Gesamtsystems

5.4.4 Übertragung

Die folgende Auflistung zeigt die Parameter der bidirektionalen Kommunikation zwischen der Basisstation und den Zellensensoren. Im Wesentlichen stimmen diese mit den Parametern überein, die bereits bei der Erprobung des Transceiver-Moduls mit dem Transceiver *Si4431* von *Silicon Labs* in Abschnitt 3.6.1 verwendet wurden. Ein entscheidender Unterschied ist die nun mögliche zyklische Redundanzprüfung (*CRC*), weil auch auf den Sensoren der Transceiver *CC1101* von *Texas Instruments* verwendet wird. Im Vergleich zu den Zellensensoren der Klassen 1 und 2 ist keine Manchester-Kodierung mehr notwendig, sodass die Übertragungsrate der Symbolrate entspricht. Diese wurde, noch während der Verwendung des Transceivers *Si4431*, zu den maximal möglichen $40\,\mathrm{kBit/s}$ gewählt. Mit den nun verwendeten Transceivern könnte diese auf bis zu $250\,\mathrm{kBit/s}$ erhöht werden [66], was in dieser Arbeit allerdings nicht mehr erprobt wurde.

- Trägerfrequenz: $434\,\mathrm{MHz}$
- Sendeleistung: $10\,\mathrm{dBm}$
- Datenrate: $40\,\mathrm{kBit/s}$ ohne Manchester-Kodierung
- *OOK*-Modulation
- Automatisches Packet Handling
- 4 *PREAMBLE* Bytes (1010 . . . 1010)
- 4 *SYNC* Bytes: 0x12 0x09 0x12 0x09
- 1 Adressierungsbyte
- 5/10 *DATA*-Bytes (Downlink/Uplink)
- Prüfsummengenerierung und Überprüfung (*CRC*) aktiviert (2 Bytes)

Die Abbildung 5.13 zeigt den Aufbau eines einzelnen Datenpaketes. Die Anzahl der *PREAMBLE*- und der *SYNC*-Bytes entspricht den Empfehlungen des Datenblatts [66]. Anhand dieser synchronisiert sich die Empfangseinheit und erkennt den Beginn der Nutzdaten. Das *ADDRESS*-Byte enthält die Zieladresse des Datenpaketes. 0x00 ist ein Broadcast-Kommando. Die Basisstation hat die Adresse 0xFF. Somit ist die maximal mögliche Größe des Sensornetzwerks auf 254 Sensoren beschränkt.

Für die eigentlichen Nutzdaten sind auf dem Downlink-Kanal 5 Bytes und auf dem Uplink-Kanal 10 Bytes vorgesehen. Diese Anzahl ist frei gewählt und kann über Macros im entsprechenden Header-File angepasst werden. Die während der Erprobung verwendeten Kommandos haben eine Länge von einem Byte, abgesehen von dem 12-Bit Messwert. Es sollte nur gezeigt werden, dass die Datenpakete unterschiedliche Größen haben können. Für spätere Anwendungen wäre es auch denkbar, die Anzahl der Nutzdaten während des Betriebs anzupassen. So könnten im Normalbetrieb zum Beispiel sekündlich kleine Datenpakete von den Zellensensoren

versendet werden. Während eines Hochstromereignisses macht es jedoch Sinn, die Rechenleistung des Mikrocontrollers für eine hohe Abtastrate der Zellenspannung auszunutzen, und die Messwerte anschließend in einem- oder mehreren großen Datenpaketen zu übertragen.

Die Berechnung der 16-Bit *CRC*-Daten erfolgt über das *ADDRESS*- und die *DATA*-Bytes.

Bei einer Datenrate von $40\,\mathrm{kBit/s}$ bzw. einer Symboldauer von $25\,\mu\mathrm{s}$ ergibt sich für das gesamte Datenpaket eine zeitliche Länge von $3.2\,\mathrm{ms}$ (Downlink) bzw. $4.2\,\mathrm{ms}$ (Uplink). Diese ließe sich durch eine Erhöhung der Datenrate weiter vermindern.

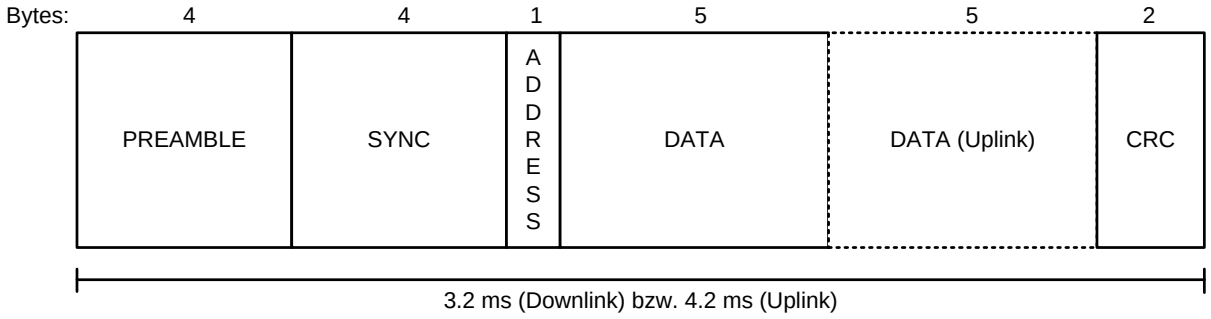
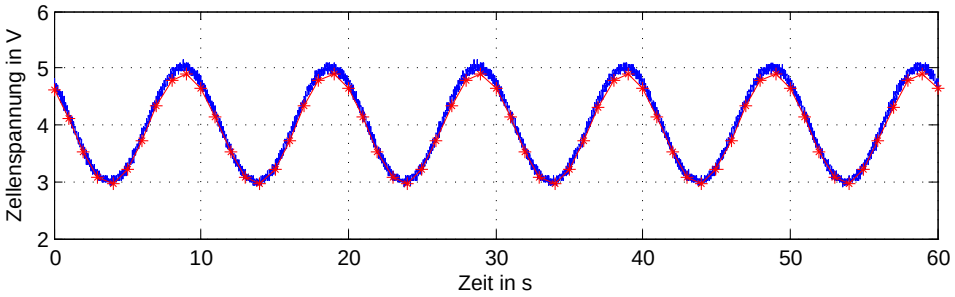


Abbildung 5.13: Aufbau eines Datenpaketes der paketorientierten Kommunikation zwischen Basisstation und den Zellensensoren während der Erprobung des Gesamtsystems

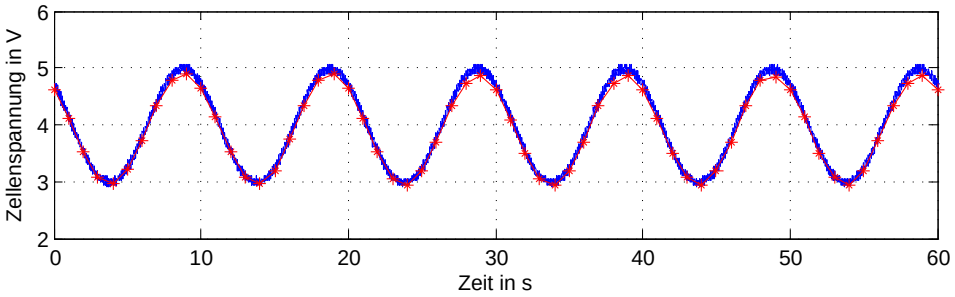
5.4.5 Ergebnis

Die Messwerte wurden von der Basissation im Sekundentakt über die serielle Schnittstelle RS-232 ($19\,200\,\mathrm{Baud}$, 8 Datenbits, keine Parität, 1 Stopp-Bit) an einen PC übertragen. An diesem erfolgte anschließend, zusammen mit den aufgenommenen Zellspannungsverläufen durch ein Oszilloskop, die Auswertung mit Hilfe eines Matlab-Skripts (siehe D.33). Die in Matlab entstandenen Grafiken zeigt die folgende Abbildung 5.14.

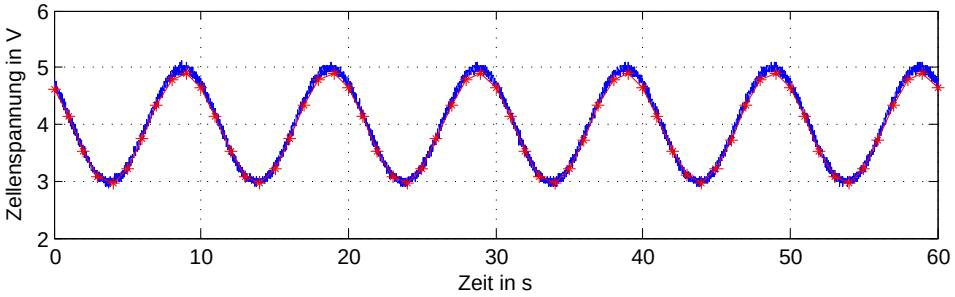
Es ist zu erkennen, dass die aufgenommenen Messwerte (rot) aller vier Zellensensoren gut zu den parallel aufgenommenen Spannungsverläufen (blau) passen. Die grundsätzliche Funktion des Gesamtsystems ist damit nachgewiesen. Dabei sei jedoch erwähnt, dass es sich um eine selektierte Messung handelt. Der Zellensensor 2 zeigt mitunter als Einziger Ausfallerscheinungen, die sich in einer stark gestörten Kommunikation zeigen. Es kam vor, dass mehrere Sekunden kein Messwert vom Zellensensor 2 empfangen werden konnte. In dieser Messreihe zeigte sich dieses Verhalten jedoch nicht. Da die drei anderen Sensoren problemlos funktionieren, scheint es sich um ein spezielles Problem des Sensors 2 zu handeln. Weil alle Sensoren dieselbe Software nutzen wird vermutet, dass bei der Bestückung des Sensors 2 versehentlich Bauelemente falscher Werte bestückt wurden. Eine weitere Untersuchung dieses Problems konnte aufgrund fehlender Zeit nicht mehr durchgeführt werden.



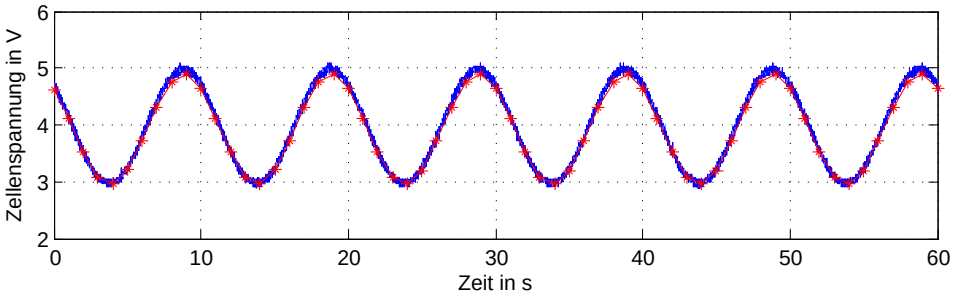
(a) Zellensensor 1



(b) Zellensensor 2



(c) Zellensensor 3



(d) Zellensensor 4

Abbildung 5.14: Vergleich der an den Zellensensoren anliegenden Zellenspannungen (blau) und der im Sekundentakt gemessenen und an die Basisstation übertragenen Spannungswerte (rot).

Bei der Betrachtung des Messfehlers lassen sich Abweichungen von etwa $-150\,\mathrm{mV}$ bis $+50\,\mathrm{mV}$ feststellen. Ursachen dafür, die im weiteren Entwicklungsverlauf durch entsprechende Kalibrierungen der Zellensensoren minimiert werden können, sind Offsetfehler und Linearitätsfehler. Letztere resultieren insbesondere durch Abweichungen der Widerstände des Spannungsteilers am Eingang des A/D-Umsetzers (vgl. Schaltplan-Seite 2 im Anhang B.1). Außerdem zeigte sich während den Messungen eine Antennenwirkung der verwendeten BNC-Kabel zwischen dem Zellenspannungsgenerator und dem Oszilloskop (vgl. Foto des Messaufbaus im Anhang in Abbildung F.4), sodass sich ein überlagertes Rauschsignal mit einer Amplitude von bis zu $200\,\mathrm{mV}$ zeigte. In Anbetracht dessen kann die Erprobung des Gesamtsystems als erfolgreich angesehen werden.

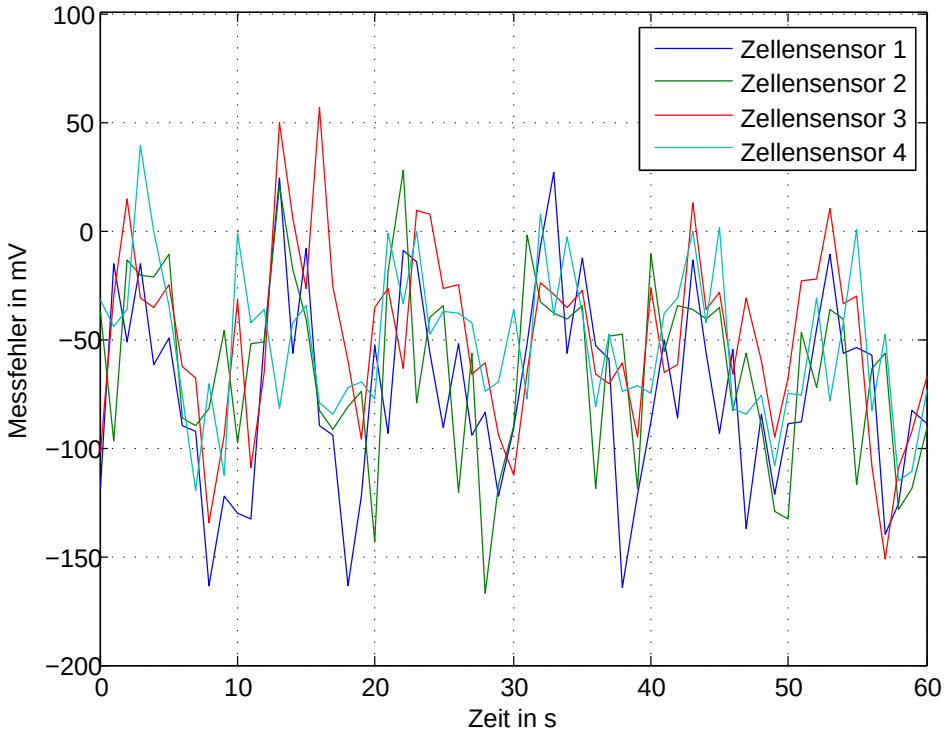


Abbildung 5.15: Darstellung der Messfehler bei der Messung der Zellenspannungen durch die Zellensensoren 1 bis 4 (vgl. Abbildung 5.14)

5.5 Stromaufnahme des Zellensensors

Um eine Aussage zur geplanten Energieeffizienz treffen zu können, wurde der benötigte Strom jedes Zellensensors in verschiedenen Betriebszuständen gemessen (vgl. Tabelle 5.2). Alle Messungen wurden bei einer Versorgungsspannung $U_{cell} = 3.3\,\mathrm{V}$ durchgeführt. Bei Messung vier, für die der Zellensensor dauerhaft ein Träger gesendet hat, betrug die Sendeleistung $+13\,\mathrm{dBm}$.

Von besonderem Interesse ist der Stromverbrauch im energiesparsamsten Zustand des Sensors, in dem sich sowohl der Transceiver als auch der Mikrocontroller im SLEEP-Zustand befinden, durch die Wakeup-Schaltung aber dennoch jederzeit aktivierbar sind. Der Stromverbrauch in diesem Zustand liegt durchschnittlich bei $736\,\mu\mathrm{A}$. Eine Vergleichsmessung an den Sensoren 3 und 4, bei denen zu diesem Zeitpunkt die Bauteile des Nebenstrompfades zur Ladungsbalancierung noch nicht bestückt waren, ergab in beiden Fällen sogar nur einen Stromverbrauch von $570\,\mu\mathrm{A}$ bzw. $590\,\mu\mathrm{A}$. Das deutet auf einen Kriechstrom von etwa $150\,\mu\mathrm{A}$ durch die beiden Nebenstrompfade der passiven Ladungsbalancierung hin (vgl. Schaltplan-Seite 6 im Anhang B.1).

Mikro-controller	Trans-ceiver	Balancing	Sensor 1	Sensor 2	Sensor 3	Sensor 4
SLEEP	SLEEP	OFF	$751\,\mu\mathrm{A}$	$806\,\mu\mathrm{A}$	$757\,\mu\mathrm{A}$	$628\,\mu\mathrm{A}$
ON	SLEEP	OFF	$1.638\,\mathrm{mA}$	$1.925\,\mathrm{mA}$	$1.758\,\mathrm{mA}$	$1.828\,\mathrm{mA}$
ON	RX	OFF	$38.96\,\mathrm{mA}$	$39.01\,\mathrm{mA}$	$25.47\,\mathrm{mA}$	$26.54\,\mathrm{mA}$
ON	TX	OFF	$35.69\,\mathrm{mA}$	$42.54\,\mathrm{mA}$	$35.07\,\mathrm{mA}$	$44.54\,\mathrm{mA}$
ON	RX	ON	$225.86\,\mathrm{mA}$	$230.47\,\mathrm{mA}$	$222.64\,\mathrm{mA}$	$236.28\,\mathrm{mA}$

Tabelle 5.2: Stromaufnahme des Zellensensors in verschiedenen Betriebszuständen

Der durchschnittliche Stromverbrauch im energiesparsamsten Zustand des Sensors setzt sich gemäß der Tabelle 5.3 zusammen. Gemäß den Angaben aus den Datenblättern ergibt sich ein theoretischer Gesamtstrom von $462.7\,\mu\mathrm{A}$. Dieser liegt etwa $270\,\mu\mathrm{A}$ unter dem durchschnittlich gemessenen Strom und ergibt sich durch Abweichungen der Angaben in den Datenblättern sowie Verlusten im Spannungswandler.

Beschreibung	Strom in μA	Vgl. Schaltplan-Seite
Mikrocontroller *MSP430F235* im *LPM4*	< 1	1
Spannungsteiler *DCDC*	5.5	2
Spannungsteiler *ADC VBAT*	16.5	2
Spannungsteiler *ADC VCC*	165	2
Antennenumschalter *ADG918*	< 1	3
Transceiver *CC1101*	< 1	4
Operationsverstärker *MCP6071*	110	5
LF Wakeup Receiver *AS3930*	2.7	5
Temperatursensor *TMP102*	10	6
Kriechströme passive Ladungsbalancierung	150	6
Summe	462.7	

Tabelle 5.3: Zusammensetzung der Stromaufnahme im Schlafzustand des Zellensensors

In einer Weiterentwicklung könnte untersucht werden, ob die beiden *MOSFETs* vollständig sperren und eine Optimierung des Stromverbrauchs vorgenommen werden, indem insbesondere die Verluste durch die Spannungsteiler minimiert werden. Ungeklärt ist bisher außerdem der erhebliche Unterschied zwischen den vier Zellensensoren in der Stromaufnahme beim Senden und Empfangen durch den Transceiver.

Neben den Messungen mit dem Amperemeter oben, wurde eine Messung des benötigten Stroms (Zellensensor 1) über einen $10\,\Omega$-Messwiderstand und ein Oszilloskop im Zeitbereich über $10\,\mathrm{s}$ aufgenommen. Der Ablauf dabei entsprach dem bei der Erprobung des Gesamtsystems in Abschnitt 5.4.2, mit dem Unterschied, dass nur ein Messwert aufgenommen und versendet wurde.

Die Abbildung 5.16 zeigt den aufgezeichneten Stromverlauf über die gesamten 10 Sekunden. Das Einschalten der Betriebsspannung über einen Kippschalter erzeugt kurzzeitig eine Stromspitze von $80\,\mathrm{mA}$, woraufhin sich der Sensor initialisiert und zur visuellen Kontrolle für 3 Sekunden die drei *LEDs* einschaltet. Dann wechselt der Zellensensor in den Schlafzustand, aus welchem dieser bei Sekunde 5, über das Wakeup-Signal der Basisstation, wieder aktiviert wird. Für 2 Sekunden befindet sich der Sensor im aktiven Empfangszustand, in dem auf Kommandos von der Basisstation reagiert werden kann. Da die Reaktionen auf diese Kommandos in dieser groben Zeitauflösung nicht zu erkennen sind, zeigt die Abbildung 5.17 die markierten Bereiche in kleinerer Zeitauflösung. Nachdem bei Sekunde 7 das Sleep-Kommando empfangen wurde, werden erneut für eine Sekunde alle *LEDs* eingeschaltet und anschließend der Energiesparmodus aktiviert.

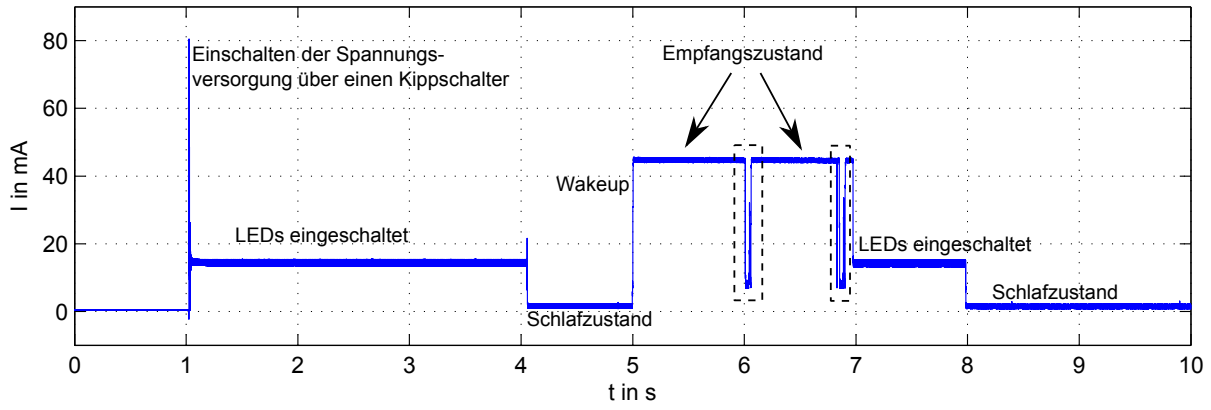


Abbildung 5.16: Stromaufnahme des Zellensensors 1 während verschiedenen Betriebszuständen über $10\,\mathrm{s}$ gemessen

Die Abbildung 5.17 zeigt, wie der Betriebsstrom sinkt, sobald ein Kommando empfangen und der Empfangszustand verlassen wird. Das Senden eines einzelnen Datenpaketes benötigt für $5\,\mathrm{ms}$ etwa $30\,\mathrm{mA}$. Die gemessene Stromaufnahme in Tabelle 5.2 ist größer, weil dort ein durchgängiges, unmoduliertes Trägersignal gesendet wurde. Die jeweils unteren Abbildungen zeigen parallel zum Strombedarf die Datenpakete der bidirektionalen Kommunikation. Diese wurden

mit der Schleifenantenne auf der Zellensensor-Messplatine als dritte Antenne parallel empfangen und ebenfalls mit dem Oszilloskop aufgezeichnet.

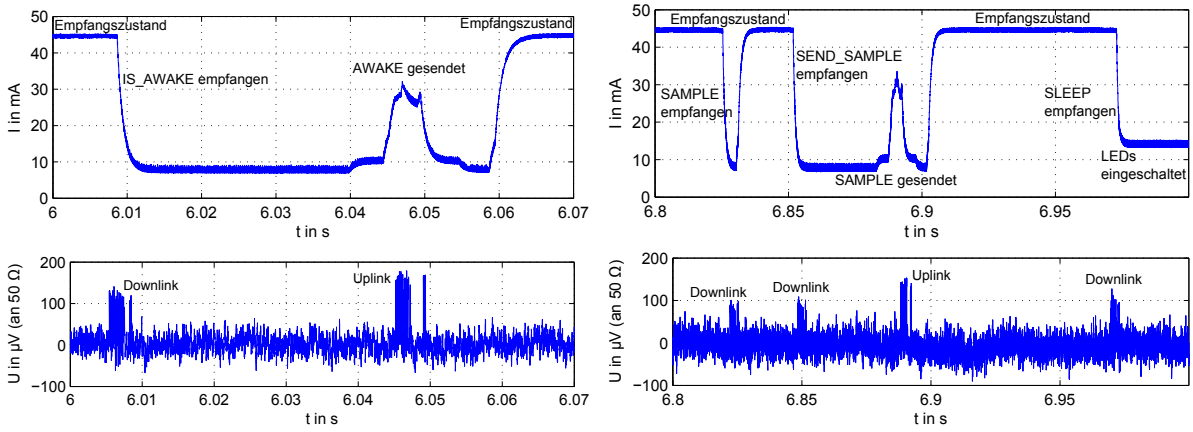


Abbildung 5.17: Stromaufnahme des Zellensensors 1 beim Übergang zwischen verschiedenen Betriebszuständen (oben) und die Datenpakete auf dem Funkkanal (unten)

Die Vergrößerung des zweiten Ausschnitts zeigt den Empfang von insgesamt drei Downlink-Kommandos und einem Uplink-Kommando. Das Kommando *SAMPLE* lässt den Zellensensor einen Messwert aufnehmen. Dieser wird über das Kommando *SEND_SAMPLE* von der Basisstation angefragt, woraufhin der Messwert gesendet wird. Der Empfang des Kommandos *SLEEP* führt nach dem Einschalten aller *LEDs* für eine Sekunde zur Aktivierung des Schlafzustands.

6 Fazit

6.1 Zusammenfassung und Bewertung

In der vorliegenden Arbeit wurde erfolgreich die Hardware eines Zellensensors für Fahrzeugbatterien der Klasse 3 entwickelt. Ein solcher ist gekennzeichnet durch eine aktive Empfangseinheit, um eine vollwertige bidirektionale Kommunikation zwischen der Basisstation und den Zellensensoren realisieren zu können.

Gemäß der Aufgabenstellung (vgl. Anhang A) wurden verschiedene Konzepte erarbeitet und Energiebedarfsabschätzungen unternommen, um die Energieeffizienz der Sensoren zu maximieren. Für die Umsetzung des gewählten Konzepts wurden geeignete Transceiver recherchiert und praktisch erprobt.

Wichtiger Bestandteil des entwickelten Sensors ist die sogenannte *Wakeup-Schaltung*, die es erlaubt, die Zellensensoren in weniger als $1\,\mathrm{ms}$ aus einem äußerst energiesparsamen Zustand in einen aktiven Messbetrieb zu versetzen. Neben theoretischen Betrachtungen des Wakeup wurden umfangreiche praktische Untersuchungen durchgeführt, wobei auch Hardware für die Basisstation entwickelt und in Betrieb genommen wurde, um das geforderte Wakeup-Signal zu erzeugen. Für die elektrische Signalverarbeitung dieses Wakeup-Signals auf dem Zellensensor wurden verschiedene Schaltungsvarianten recherchiert, simuliert sowie praktisch umgesetzt und erprobt. Dabei spielten insbesondere Impedanzanpassungen eine besondere Rolle, ohne die Reflexionen der Ultra-Hochfrequenzsignale in den Leiterbahnen zu nicht akzeptablen Leistungsverlusten geführt hätten.

Für den Empfang des Wakeup-Signals, sowie für die bidirektionale Kommunikation zwischen der Basisstation und den Zellensensoren wurde eine *PCB*-Antenne dimensioniert und erprobt. Die entstandene *kleine Schleifenantenne* besteht im Wesentlichen aus einer Leiterbahn, sodass diese besonders preisgünstig und reproduzierbar herstellbar ist. Zusätzlich wurde vergleichsweise eine kommerzielle *Chip-Antenne* erprobt.

Neben dem praktischen Aufbau von vier Zellensensoren wurde eine Erprobung des Sensornetzwerks zum grundsätzlichen Nachweis der Funktion durchgeführt. Sowohl das Wakeup als auch die bidirektionale Kommunikation zeigten dabei die geforderte Funktion. Damit wurde die Aufgabenstellung erfolgreich umgesetzt.

Besonders erwähnenswert ist der Erfolg, einen Zellensensor über das Wakeup-Signal aktivieren zu können, auch wenn sich dieser innerhalb der $LiFePO_4$-Batteriezelle befindet, welche von einem zylinderförmigen Aluminiumgehäuse umgeben ist (vgl. Abschnitt 2.1.1).

Im Vergleich zu den bisher entstandenen Zellensensoren des Forschungsprojekts, bietet der in dieser Arbeit entstandene Zellensensor mit derzeit $40\,\mathrm{kBit/s}$ 8-fach höhere Symbolraten bei der Funkübertragung. Im Vergleich zu den Zellensensoren der verwandten Klasse 2 ergeben sich zudem weitere Vorteile. Sowohl das Wakeup-Signal als auch die paketbasierte Datenkommunikation finden in einem Frequenzband statt, sodass auf dem Zellensensor und an der Basisstation nur noch jeweils eine Antenne benötigt wird. Die benötigte Sendeleistung der Basisstation, um die Zellensensoren aus dem Ruhezustand aktivieren zu können, konnte zudem um 98.7 % von $750\,\mathrm{mW}$ auf $10\,\mathrm{mW}$ verringert werden.

Mit einer durchschnittlichen Stromaufnahme von $736\,\mu\mathrm{A}$ (Abschnitt 5.5) während des Schlafzustands liegt die Entladung der $LiFePO_4$-Batteriezelle in der Größenordnung der Selbstentladung (vgl. Abschnitt 2.1.3). Lässt man die Selbstentladung unberücksichtigt, wäre die $50\,\mathrm{Ah}$-Batteriezelle (aus Abschnitt 2.1.1) erst nach etwa knapp 8 Jahren vollständig durch den Zellensensor entladen.

An dieser Stelle sei noch angemerkt, dass es sich bei dem entwickelten Sensor laut Definition (Abschnitt 1.2) nicht um einen reinen Zellensensor der Klasse 3 handelt. Die herausstellende Eigenschaft ist der aktive Empfänger im Vergleich zu einem passiven Empfänger beim Zellensensor der Klasse 2. Je nach Auslegung der Definition handelt es sich also um eine Mischung der Klassen 2 und 3. Vor dem Hintergrund, dass es jedoch möglich ist, den entstandenen Zellensensor auch ohne die Wakeup-Schaltung zu betrieben (vgl. Konzept 2 in Abschnitt 2.2.2), ist die Bezeichnung als Sensor der Klasse 3 zulässig.

6.2 Ausblick

Mit der in dieser Arbeit entwickelten Hardware für einen Zellensensor der Klasse 3 und den Anpassungen an der Basisstation ist die Entwicklung des Sensornetzwerks der Klasse 3 noch nicht abgeschlossen. Die in dieser Arbeit entstandene Software für die Erprobung umfasst größtenteils hardwarenahe Funktionen zur Konfiguration der integrierten Schaltkreise und zum Datenaustausch mit diesen. Auch wenn diese Software als Basis weiterhin benutzt werden kann, sind die Funktionsebenen der Steuersprache *BMCL* (Battery Monitoring and Control Language) noch zu implementieren. Dabei handelt es sich um eine beschreibende Sprache, die über verschiedene Batterietypen abstrahiert und die verschiedenen Schichten der Funktionalität beschreibt. [46]

In Abschnitt 2.5.1 wurden die beiden möglichen Betriebszustände des LF Wakeup Receivers *AS3930* beschrieben. In dieser Arbeit wurde ausschließlich der Modus erprobt, in dem eine reine *LF*-Trägererkennung durchgeführt wird. Durch den Empfang anderer Funkdienste sind ungewollte Wakeups in diesem Betriebsmodus möglich. Auch wenn diese unter Laborbedingungen bisher nicht aufgetreten sind, ist die Implementierung des adressierbaren Wakeup für einen realitätsnahen Einsatz der Zellensensoren in Betracht zu ziehen. Neben minimalen Änderungen in der Konfiguration des *AS3930* müsste dazu vor allem das von der Basisstation abgestrahlte Wakeup-Signal angepasst werden.

Wichtiger als die Optimierung des Wakeup ist mittelfristig die Inbetriebnahme des Temperatursensors *TMP102*. Dieser wird durch den Mikrocontroller des Zellensensors bislang nicht angesteuert. Da die Temperaturmessung auf den Sensoren der Klasse 2 mit demselben integrierten Schaltkreis erfolgte, kann die Software zur Ansteuerung des Temperatursensors über den I^2C-Bus aus der Masterarbeit [25] übernommen werden.

Um die Stromaufnahme im Schlafzustand des Zellensensors von derzeit durchschnittlich $736\,\mu\mathrm{A}$ (Abschnitt 5.5) weiter zu verringern wäre es denkbar, den Operationsverstärker als Kern des nichtinvertierenden Verstärkers aus der Wakeup-Schaltung zu entfernen. Mit einem Strombedarf von $110\,\mu\mathrm{A}$ trägt dieser maßgeblich zur gesamten Stromaufnahme bei. In diesem Zuge müsste dann allerdings die Tief- bzw. Bandpassfilterung verändert und neu dimensioniert werden (vgl. Abschnitt 4.3). Die Tabelle 5.3 zeigt außerdem, dass im Schlafzustand des Zellensensors 40 % des aufgenommenen Stromes durch Spannungsteiler gegen das Massepotential abfließt. Besonders der Spannungsteiler für die Messung der Ausgangsspannung des Spannungswandlers könnte optimiert werden. Untersucht werden muss außerdem, ob die *MOSFET*s der passiven Ladungsbalancierung richtig sperren, da ein Kriechstrom von etwa $150\,\mu\mathrm{A}$ durch diese aufgefallen ist.

In Abschnitt 5.3 hat sich gezeigt, dass ein Zellensensor, der sich innerhalb der Batteriezelle befindet, aus einer Entfernung von $15\,\mathrm{cm}$ bis $20\,\mathrm{cm}$ geweckt werden kann. Für kleinere Batterien, wie Starterbatterien, ist diese Reichweite wahrscheinlich ausreichend. Sollte langfristig eine höhere Reichweite nötig sein, sollte die Impedanzanpassung des Hüllkurvendemodulators (Abschnitt 5.1.1.2) optimiert werden. Derzeit werden bei einem Eingangsreflexionsfaktor von $-9.1\,\mathrm{dB}$ etwa 12.3 % der eingehen Leistung reflektiert. Eine andere Möglichkeit wäre die Erhöhung der Leistung, mit der das Wakeup-Signal von der Basisstation abgestrahlt wird. Derzeit wird mit der maximal möglichen Ausgangsleistung des *CC1101* von $10\,\mathrm{dBm}$ bzw. $10\,\mathrm{mW}$ gesendet. Um diese weiter zu erhöhen könnte ein zusätzlicher *UHF*-Verstärker wie zum Beispiel der *ADL5324* von *Analog Devices* [3] eingesetzt werden. Bei einem gewählten, maximal zulässigen Intermodulationsabstand 3. Ordnung von $40\,\mathrm{dB}$ könnte die Ausgangsleistung auf $20\,\mathrm{dBm}$ bzw. $100\,\mathrm{mW}$ verzehnfacht werden. Dieses Vorgehen setzt allerdings voraus, dass eine ausreichende Abschirmung des Systems durch die Karosserie des Fahrzeugs oder zusätzliche Maßnahmen gegeben ist, um eine Störung anderer Funkdienste zu vermeiden.

Derzeit existieren noch keine Transceiver, die keinen externen Quarz benötigen. Ein solcher wäre von Vorteil, um zum einen langfristig eine Integration der Komponenten des Zellensensors auf einem Chip zu erleichtern und zum anderen ein preisintensives Bauelement einsparen zu können. Da jedoch bereits quarzfreie Transmitter existieren (Sensoren der Klassen 1 und 2), kann langfristig ein Erscheinen quarzfreier Transceiver erwartet werden. Abgesehen von der Schleifenantenne ließen sich dann alle aktuell verwendete Bauelemente, die nicht sowieso bereits integriert sind, auf einem Chip platzieren.

Ebenfalls in Bezug auf eine spätere Integrierung auf einem Chip könnte eine Entwicklung des Fraunhofer Instituts für Integrierte Schaltungen (IIS) von Interesse sein. Die Forschungsgruppe aus Erlangen hat auf dem Ultra-Low-Power-Entwicklerforum 2010 in München den Ultra Low-Current WakeUp Receiver *μRX1080* vorgestellt [36], der im Wesentlichen dieselbe Funktionalität wie der verwendete LF Wakeup Receiver bietet, ohne jedoch vorher eine Abwärtsmischung

vornehmen zu müssen. Bei einem Strombedarf von lediglich $10\,\mu\mathrm{A}$ beträgt die Empfindlichkeit $-60\,\mathrm{dBm}$. Derzeit ist dieser Chip für das Frequenzband um $869\,\mathrm{MHz}$ ausgelegt, eine Portierung nach $433\,\mathrm{MHz}$ ist nach Angaben des Instituts jedoch möglich. [18]

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[1] AKER TECHNOLOGY: *C3E Series*, Juni 2006. http://www.aker-usa.com/Crystal_Specs/C3E%20General%20Specification.pdf, Abruf: 07.01.2013

[2] ANALOG DEVICES: *ADG918 - Wideband switch*. Rev. C, 2008. http://www.analog.com/static/imported-files/data_sheets/ADG918_919.pdf, Abruf: 13.12.2012

[3] ANALOG DEVICES: *ADL5324 - 400 MHz to 4000 MHz 1/2 Watt RF Driver Amplifier*. Rev. B, 2012. http://www.analog.com/static/imported-files/data_sheets/ADL5324.pdf, Abruf: 31.01.2013

[4] ANSARI, Junaid ; PANKIN, Dmitry ; MÄHÖNEN, Petri: *Radio-Triggered Wake-ups with Addressing Capabilities for Extremely Low Power Sensor Network Applications*. Department of Wireless Networks, RWTH Aachen University, 2009. http://www.inets.rwth-aachen.de/fileadmin/templates/images/PublicationPdfs/2009/RTWAC-IJWIN_Ansari_et_al._Radio_Triggered_Wakeups_with_Addressing_Capabilities_for_Extremely_Low_Power_Sensor_Network_Applications.pdf, Abruf: 27.12.2012

[5] AUSTRIAMICROSYSTEMS: *AS3930 - Single Channel Low Frequency Wakeup Receiver*. Revision 1.0, 2009. http://www.ams.com/eng/content/download/23692/414425/file/AS3930_Datasheet_v1_00.pdf, Abruf: 30.12.2012

[6] AUSTRIAMICROSYSTEMS: *AS3932 - 3D Low Frequency Wakeup Receiver*. Revision 1.4, 2010. http://www.ams.com/eng/content/download/23636/413647/file/AS3932_LF_Wakeup_Receiver_Datasheet_v1_4.pdf, Abruf: 30.12.2012

[7] AVAGO TECHNOLOGIES: *Application Note 1089 - Designing Detectors for RF/ID Tags*, 2008. http://www.avagotech.com/docs/AV02-1577EN, Abruf: 14.01.2013

[8] AVAGO TECHNOLOGIES: *HSMS-285x Series - Surface Mount Zero Bias Schottky Detector Diodes*, 2009. http://www.avagotech.com/docs/AV02-1377EN, Abruf: 27.12.2012

[9] BALANIS, Constantine A.: *Antenna Theory: Analysis and Design*. 3. Auflage. John Wiley & Sons, 2005

[10] BEUTH, Klaus ; SCHMUSCH, Wolfgang: *Elektronik / Grundschaltungen: BD 3*. 14., überarb. u. erw. Aufl. Vogel Business Media/VM, 2000

[11] BORGEEST, Kai: *Elektronik in der Fahrzeugtechnik: Hardware, Software, Systeme und Projektmanagement (ATZ/MTZ-Fachbuch)*. 2., überarb. u. erw. Aufl. 2010. Vieweg+Teubner Verlag, 2010

[12] BUNDESNETZAGENTUR: *Frequenznutzungsplan*. August 2011, 2011. http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/BNetzA/Sachgebiete/Telekommunikation/Regulierung/Frequenzordnung/Frequenznutzungsplan/Frequenznutzungsplan2011pdf.pdf?__blob=publicationFile, Abruf: 13.12.2012

[13] BUNDESNETZAGENTUR: *Allgemeinzuteilung von Frequenzen zur Nutzung durch Funkanwendungen mit geringer Reichweite für nicht näher spezifizierte Anwendungen; Non-specific Short Range Devices (SRD)*. Vfg 43 / 2012, 2012. http://www.bundesnetzagentur.de/SharedDocs/Downloads/DE/BNetzA/Sachgebiete/Telekommunikation/Regulierung/Frequenzordnung/Allgemeinzuteilung/SRDShortRangeDevicesVfg432012.pdf?__blob=publicationFile, Abruf: 13.12.2012

[14] COILCRAFT: *Chip Inductors - 0603HP Series (1608)*, Mai 2012. http://www.coilcraft.com/pdf_viewer/showpdf.cfm?f=pdf_store:0603hp.pdf, Abruf: 13.12.2012

[15] CTS: *Model 405 - Surface Mount Quartz Crystal*. Rev. G, 2012. http://www.ctscorp.com/components/Datasheets/008-0253-0.pdf, Abruf: 13.12.2012

[16] DEVI, Kavuri Kasi A. ; DIN, Norashidah M. ; CHAKRABARTY, Chandan K.: *Optimization of the Voltage Doubler Stages in an RF-DC Convertor Module for Energy Harvesting*. Department of Electrical and Electronic Engineering, INTI International University, Nilai, Malaysia and Department of Electronics and Communication Engineering, Universiti Tenega Nasional, Kajang, Malaysia

[17] ECC REPENNING GMBH: *Übersicht Rundzellen*, 2012. http://www.eccbatteries.com/6-0-Sortiment.html, Abruf: 13.12.2012

[18] FRAUNHOFER INSTITUT FÜR INTEGRIERTE SCHALTUNGEN IIS: *ULTRA LOW-CURRENT WAKEUP RECEIVER*, 2010. http://www.iis.fraunhofer.de/content/dam/iis/de/dokumente/ic/wake-up-receiver_2010.pdf, Abruf: 03.02.2013

[19] GAMM, Gerd U. ; REINDL, Leonard M.: *Smart Metering Using Distributed Wake-up Receivers*. Laboratory for Electrical Instrumentation, Department of Microsystems Engineering - IMTEK, University of Freiburg, Germany, 2012. http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=6229365, Abruf: 30.12.2012

[20] GU, Lin ; STANKOVIC, John A.: *Radio-Triggered Wake-Up for Wireless Sensor Networks*. Department of Computer Science, University of Virginia, http://www.cse.ust.hk/~lingu/academia/RadioTrigger.JRTS.10.pdf, Abruf: 27.12.2012

[21] HAGMANN, Gert: *Grundlagen der Elektrotechnik: Das bewährte Lehrbuch für Studierende der Elektrotechnik und anderer technischer Studiengänge ab 1. Semester*. 15. durchgesehene und korr. Auflage 2011. Aula, 2010

[22] HERING, Ekbert ; BRESSLER, Klaus ; GUTEKUNST, Jürgen: *Elektronik für Ingenieure und Naturwissenschaftler (Springer-Lehrbuch)*. 5., aktualisierte Aufl. Springer, 2005

[23] ILGIN, Sergej: *Drahtlose Sensoren für Batteriemodule - Konzeption, Kalibrierung, Hard- und Softwareentwicklung*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, 2011

[24] INTERNATIONAL TELECOMMUNICATION UNION: *Radio Regulations*. Edition of 2004, 2004. http://www.itu.int/dms_pub/itu-s/oth/02/02/S020200001A4501PDFE.pdf, Abruf: 15.12.2012

[25] JEGENHORST, Niels: *Entwicklung eines Zellensensors für Fahrzeugbatterien mit bidirektionaler drahtloser Kommunikation*. Hamburg, Hochschule für Angewandte Wissenschaften, Masterthesis, 2011

[26] JOHANSON TECHNOLOGY, INC.: *430/435 MHz Impedance Matched Balun/LPF Integrated Component for T.I. Chipsets*, 2012. http://www.johansontechnology.com/datasheets/chipset-specific/0433BM15A0001.pdf, Abruf: 25.01.2013

[27] KARK, Klaus: *Antennen und Strahlungsfelder: Elektromagnetische Wellen auf Leitungen, im Freiraum und ihre Abstrahlung*. 4, akt. u. erw. Aufl. 2011. Vieweg+Teubner Verlag, 2011

[28] KRISCHKE, Alois: *Rothammels Antennenbuch*. 12. DARC, 2001

[29] LINEAR TECHNOLOGY: *LTC5507* - 100 kHz *to* 1 GHz *RF Power Detector*, 2001. http://cds.linear.com/docs/Datasheet/5507f.pdf, Abruf: 21.12.2012

[30] LINX TECHNOLOGIES: *ANT-433-SP DATA SHEET*, November 2008. https://www.linxtechnologies.com/resources/data-guides/ant-433-sp.pdf, Abruf: 13.12.2012

[31] LOSCHWITZ, Rico: *Überwachung-, Zellenbalancierungs- und Leistungselektronik für eine Starterbatterie in Lithium-Eisen-Phosphat-Technologie*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, 2013

[32] MAXIM: *SC70,* 1.6 V, *Nanopower, Beyond-the-Rails Comparators With/Without Reference*, Januar 2007. http://datasheets.maximintegrated.com/en/ds/MAX9117-MAX9120.pdf, Abruf: 27.12.2012

[33] MEYER, Martin: *Kommunikationstechnik: Konzepte der modernen Nachrichtenübertragung*. 4., korr. Aufl. 2012. Vieweg+Teubner Verlag, 2011

[34] MICROCHIP: *Loop Antenna Basics and Regulatory Compliance for Short-Range Radio*, http://www.microchip.com/stellent/groups/picmicro_sg/documents/devicedoc/en020982.pdf, Abruf: 07.01.2013

[35] MICROCHIP: 110 μA, *High Precision Op Amps*, 2010. http://ww1.microchip.com/downloads/en/DeviceDoc/22142_B_MCP6071.pdf, Abruf: 14.01.2013

[36] MILOSIU, Heinrich ; EPPEL, Markus ; OEHLER, Frank: *Ultra Low-Current WakeUp Receiver with Energy Harvesting*. Vortrag auf dem Ultra-Low-Power-Entwicklerforum München, 06. Juli 2010. Fraunhofer Institut für Integrierte Schaltungen IIS, 2010. http://www.embedded-world.eu/fileadmin/user_upload/pdf/ULP_Entwicklerforum_2010/10_Milosiu.pdf, Abruf: 03.02.2013

[37] MURATA MANUFACTURING CO., LTD.: *Chip Inductor (Chip Coil) Power Inductor (Wire Wound Type) - LQH3NP_G0 Series (1212 Size)*, 2011. http://search.murata.co.jp/Ceramy/image/img/PDF/ENG/L0075S0103LQH3NP_G0.pdf, Abruf: 25.01.2013

[38] NIMO, Antwi ; GRGIC, Dario ; REINDL, Leonard M.: *Impedance optimization of wireless electromagnetic energy harvester for maximum output efficiency at μW input power*. University of Freiburg, IMTEK, Department of Microsystems Engineering, Laboratory for Electrical Instrumentation, 2012

[39] NXP: *2N7002* - 60 V, 300 mA *N-channel Trench MOSFET*. Rev. 7, 2011. http://www.nxp.com/documents/data_sheet/2N7002.pdf, Abruf: 25.01.2013

[40] OLIMEX LTD.: *MSP430-169STK*, 2004. https://www.olimex.com/Products/MSP430/Starter/MSP430-169STK/, Abruf: 11.12.2012

[41] PAPULA, Lothar: *Mathematische Formelsammlung: für Ingenieure und Naturwissenschaftler*. 10. Aufl. 2009. Vieweg+Teubner Verlag, 2009

[42] PLASCHKE, Stephan: *Experimentalsystem für drahtlose Batteriesensorik*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit, 2008

[43] PÜTTJER, Simon: *Diagnosefunktion für Automobil-Starterbatterien mit drahtlosen Zellensensoren*. Hamburg, Hochschule für Angewandte Wissenschaften, Diplomarbeit, 2011

[44] RFM: *DR5100* - 433.92 MHz *Receiver Module*, 2008. http://www.rfm.com/products/data/dr5100.pdf, Abruf: 17.12.2012

[45] RFM: *TR3000* - 433.92 MHz *Hybrid Module*, 2008. http://www.rfm.com/products/data/tr3000.pdf, Abruf: 17.12.2012

[46] RIEMSCHNEIDER, Karl-Ragmar ; SCHNEIDER, Matthias: *Drahtlose Sensoren in den Zellen von Fahrzeug-Batterien*. HAW Hamburg, 2011. http://www.haw-hamburg.de/fileadmin/user_upload/Schulung/_temp_/IWKM21_Batsen.pdf, Abruf: 16.12.2012

[47] RIEMSCHNEIDER, Karl-Ragmar ; SCHNEIDER, Matthias: *Drahtlose Sensoren in den Zellen von Fahrzeug-Batterien*. Paper und Vortrag auf der 21. Internationalen Wissenschaftlichen Konferenz Mittweida, 26. - 27. Oktober 2011. Mittweida, 2011. http://www.staff.hs-mittweida.de/~delport/dlslides.php?id=318, Abruf: 16.12.2012

[48] SIEGL, Johann: *Schaltungstechnik - Analog und gemischt analog/digital: Entwicklungsmethodik, Funktionsschaltungen, Funktionsprimitive von Schaltkreisen (Springer-Lehrbuch)*. 3., bearb. u. erg. Aufl. Springer Berlin Heidelberg, 2008

[49] SILICON LABS: *AN422 - Antenna Development Guide for the Si4020, Si4320 & Si4420 ISM Band FSK EZRadio® Chipsets*. Version 1.51, 2009. http://www.silabs.com/Support%20Documents/TechnicalDocs/an422.pdf, Abruf: 08.01.2013

[50] SILICON LABS: *AN415 - EZRADIOPRO LAYOUT DESIGN GUIDE*. Rev 0.2 12/10, 2010. http://www.silabs.com/Support%20Documents/TechnicalDocs/AN414.pdf, Abruf: 17.01.2013

[51] SILICON LABS: *AN415 - EZRADIOPRO PROGRAMMING GUIDE*. Rev 0.7 7/10, 2010. http://www.silabs.com/Support%20Documents/TechnicalDocs/AN415.pdf, Abruf: 17.01.2013

[52] SILICON LABS: *Si4430/31/32 ISM TRANSCEIVER*, 2010. http://www.silabs.com/Support%20Documents/TechnicalDocs/Si4430-31-32.pdf, Abruf: 09.01.2013

[53] SILICON LABS: *AN436 - Si4030/4031/4430/4431 PA MATCHING*. Rev 0.2 4/11, 2011. http://www.silabs.com/Support%20Documents/TechnicalDocs/AN436.pdf, Abruf: 17.01.2013

[54] SILICON LABS: *AN440 - Si4430/31/32 REGISTER DESCRIPTIONS*. Rev 0.4 11/11, 2011. http://www.silabs.com/Support%20Documents/TechnicalDocs/AN440.pdf, Abruf: 17.01.2013

[55] STEINMANN, Tobias: *Hard- und Softwareentwicklung für einen Controller-gesteuerten, vernetzten Zellspannungsgenerator*. Hamburg, Hochschule für Angewandte Wissenschaften, Bachelorthesis, 2012

[56] TEXAS INSTRUMENTS: *MSP430F15x, MSP430F16x, MSP430F161x Mixed Signal Microcontroller*. Rev. G, 2002. http://www.ti.com/lit/ds/symlink/msp430f169.pdf, Abruf: 27.01.2013

[57] TEXAS INSTRUMENTS: *MSP430x2xx Family User's Guide*. Rev. I, 2004. http://www.ti.com/lit/ug/slau144i/slau144i.pdf, Abruf: 21.01.2013

[58] TEXAS INSTRUMENTS: *ISM-Band and Short Range Device Antennas*, March 2005. http://www.ti.com/lit/an/swra046a/swra046a.pdf, Abruf: 11.12.2012

[59] TEXAS INSTRUMENTS: *MSP430x1xx Family User's Guide*. Rev. F, 2006. http://www.ti.com/lit/ug/slau049f/slau049f.pdf, Abruf: 21.01.2013

[60] TEXAS INSTRUMENTS: *Low Input Voltage Synchronous Boost Converter with* 1.3 A *Switches*. Rev. C, 2007. http://www.ti.com/lit/ds/symlink/tps61201.pdf, Abruf: 25.01.2013

[61] TEXAS INSTRUMENTS: *Low Power Digital Temperature Sensor With SMBus/Two-Wire Serial Interface in SOT563*. Rev. C, 2007. http://www.ti.com/lit/ds/symlink/tmp102.pdf, Abruf: 25.01.2013

[62] TEXAS INSTRUMENTS: *MSP430F23x, MSP430F24x(1), MSP430F2410 Mixed Signal Microcontroller*. Rev. I, 2007. http://www.ti.com/lit/ds/symlink/msp430f235.pdf, Abruf: 27.01.2013

[63] TEXAS INSTRUMENTS: *Design Note DN502 - CRC Implementation*. Rev. D, 2009. http://www.ti.com/lit/an/swra111d/swra111d.pdf, Abruf: 17.01.2013

[64] TEXAS INSTRUMENTS: *Application Report - MSP430 Interface to CC1100/2500 Code Library*. Rev. A, 2010. http://www.ti.com/lit/an/slaa325a/slaa325a.pdf, Abruf: 25.01.2013

[65] TEXAS INSTRUMENTS: *Design Note DN025 - Johanson Technology Matched Balun Filters for CC110x & CC111x*, 2011. http://www.ti.com/lit/an/swra250a/swra250a.pdf, Abruf: 25.01.2013

[66] TEXAS INSTRUMENTS: *CC1101 - Low-Power Sub-1 GHz RF Transceiver*. Rev. H, 2012. http://www.ti.com/lit/ds/symlink/cc1101.pdf, Abruf: 11.12.2012

[67] TEXAS INSTRUMENTS: *CC110x/CC111x OOK/ASK Register Settings*, 2012. http://www.ti.com/lit/an/swra215e/swra215e.pdf, Abruf: 25.01.2013

[68] TIETZE, Ulrich ; SCHENK, Christoph: *Halbleiter-Schaltungstechnik*. 13., neu bearbeitete Auflage. Springer, 2009

[69] TXC: *Quartz Crystals - 7B Series*, http://www.txccrystal.com/images/pdf/7b.pdf, Abruf: 17.01.2013

[70] ZINKE, O. ; BRUNSWIG, H.: *Hochfrequenztechnik 2: Elektronik und Signalverarbeitung (Springer-Lehrbuch)*. 5., neubearb. Aufl. Springer, 1998

Abkürzungsverzeichnis

ADC Analog-to-Digital-Converter

(2-)ASK (Binary) Amplitude-Shift Keying - (Zweistufige) Amplitudenumtastung

BATSEN Drahtlose Zellensensoren für Fahrzeugbatterien

BMCL Battery Monitoring and Control Language

BMS Batteriemanagementsystem

CMOS Complementary Metal Oxide Semiconductor

CRC Cyclic Redundancy Check - Zyklische Redundanzprüfung

DCO Digital Controlled Oscillator

FIFO First In - First Out

FSK Frequency-Shift Keying - Frequenzumtastung

GPIO General Purpose Input/Output

HAW-Hamburg Hochschule für Angewandte Wissenschaften Hamburg

HF High frequency - Kurzwellen ($3\,\mathrm{MHz}$ bis $30\,\mathrm{MHz}$)

I^2C Inter-Integrated Circuit

IC Integrated Circuit - Integrierte Schaltung

ISM-Band Industrial, Scientific and Medical Band

ITU International Telecommunication Union - Internationale Fernmeldeunion

JTAG Joint Test Action Group - Programmier-/Debug-Schnittstelle

LED Light-Emitting Diode

LF Low frequency - Langwelle ($30\,\mathrm{kHz}$ bis $300\,\mathrm{kHz}$)

$LiFePO_4$ Chemische Bezeichnung für Lithium-Eisenphosphat

LNA Low Noise Amplifier - Rauscharmer Verstärker

LPM Low Power Mode

MOSFET Metal-Oxide-Semiconductor Field-Effect Transistor

OOK On-off keying - Digitale Amplitudenmodulation

PCB Printed Circuit Board

POR Power-On Reset

RAM Random-Access Memory

RFID Radio Frequency Identification

SMD Surface-Mounted Device

SMA SubMiniature - Hochfrequenz-Steckverbinder

SOC State of charge - Ladezustand

SOH State of health - Gesundheitszustand

SPI Serial Peripheral Interface - Synchroner serieller Datenbus

SRD Short Range Devices - Kurzstreckenfunk

TTL Transistor-Transistor-Logik

UHF Ultra-High-Frequency - Ultra-Hochfrequenz

VSWR Voltage Standing Wave Ratio - Stehwellenverhältnis

WOR Wake On Radio, Low power polling

A Aufgabenstellung

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Hochschule für Angewandte Wissenschaften Hamburg
Department Informations- und Elektrotechnik
Prof. Dr.-Ing. Karl-Ragmar Riemschneider

30. Oktober 2012

Bachelorthesis Phillip Durdaut

Zellensensor für Fahrzeugbatterien mit Kommunikation und Wakeup-Funktion im ISM-Band bei 434 MHz

Motivation

Antriebs- und Traktionsbatterien werden für elektrische Fahrzeuge eingesetzt und werden aus in Reihe geschalteten Batteriezellen aufgebaut. Zunehmend werden moderne Lithium-Technologien eingesetzt. Für den optimierten Betrieb mit dem Ziel der Lebensdauersicherung dieser Batterien ist ein messtechnischer Zugang zu den Zellen und nicht nur zur Gesamtbatterie wünschenswert. Für die Betriebssicherheit und Verfügbarkeitsprognose ist dieser Zellen-Zugang in vielen Fällen notwendig. Im Rahmen des vom Bundesministerium für Bildung und Forschung geförderten Forschungsvorhabens BATSEN (drahtlose Zellensensoren für Fahrzeugbatterien) werden dafür Lösungen untersucht, bei denen Messwerte von jeder einzelnen Batteriezelle aufgenommen und drahtlos übertragen werden.
Die wechselnden Lang- und Kurzzeitbelastungen der Batterie sind dabei zu berücksichtigen. Dabei kann eine bidirektionale Auslegung des Nachrichtenkanals die Nachteile der bisherigen unkoordinierten Betriebsart vermeiden. Durch Wake-Up- und Sleep-Betriebsarten können Möglichkeiten der Senkung der Versorgungsenenerige für das Sensorsystem erschlossen werden.

Aufgabe

Herr Phillip Durdaut erhält die Aufgabe, die Konzepte für eine verbesserte Zellen-Sensorik zu untersuchen. Hierbei soll sowohl die Datenübermittlung vom zentralen Batteriemanagement zum Sensor (sog. Downlink) als auch die Übermittlung vom Sensor zur Managementeinheit (sog. Uplink) erfolgen.
Als neuer Aspekt ist dabei zu berücksichtigen, dass die Kommunkation von Up- und Downlink in einem UHF-Band erfolgen soll. Damit entfallen aufwändig getrennte Antennen. Experimentell soll geklärt werden, ob einfache stromsparende Wakeup-Empfänger in Kombination mit einem aktiven Transceiver Vorteile erbingen. Dabei soll der Transceiver nur in kurzzeitigem Betrieb genutzt werden, weil er relativ großen Stromverbrauch besitzt. Der einfache und stromsparende Wakeup-Empfänger soll hingegen in der übrigen Zeit aktiv sein. Er soll nur der Aktivierung des Sensors dienen.
Für die Abschlußarbeit sind die folgenden Arbeitspakete geplant:

1. Einführung und Analyse der Rahmenbedingungen
   - Einarbeitung in die Projektzielstellung
   - Darstellung der bereits im Projekt erarbeiteten Lösungsvarianten

   Lösungsvarianten
2. Analyse, Konzept und Lösungsvarianten
   - Erarbeitung verschiedener Sensorkonzepte zur bidirektionalen Kommunikation mit der Basisstation
   - Recherche und Erarbeitung besonders energieeffizienter Wakeup-Möglichkeiten

- Darstellung der Strombilanzen der verschiedenen Konzepte
- Spice-Simulationen
- Recherche von Bauteilen, insbesondere Transceiver-Bausteinen und Empfänger-Dioden

3. Praktische Voruntersuchungen zur Machbarkeit

   - Entwurf und Erprobung einer PCB-Schleifenantenne
   - Vergleichserprobung einer kommerziellen "Chip-Antenne"
   - Entwurf und Aufbau eines Transceiver-Moduls für die bestehende Basisstation
   - Entwurf und Erprobung passiver Detektorschaltungen
   - Microwave Office Simulationen und Vergleichsmessungen
   - Einfache Mess- und Laboraufbauten von Teilschaltungen

4. Schaltungsentwicklung und Aufbau nach Festlegung des gewählten Konzepts

   - Schaltungsentwicklung
   - Microwave Office Simulationen
   - Aufbau von Sensorplatinen

5. Praktische Erprobung der Zellensensoren

   - Erprobung des Wakeups
   - Erprobung bidirektionale Kommunikation
   - Nachweis der grundsätzlichen Funktion

6. Auswertung und Bewertung

   - Vergleich mit alternativen Lösungen aus Vorarbeiten
   - Auswertung der realen Strombilanz und weiteren Erkenntnisse aus der Realisierung
   - Diskussion von Vor- und Nachteilen, gelöste und offene Punkte, Ausblick

Dokumentation

Die Fachliteratur, die Vorarbeiten und die kommerziellen Unterlagen sind zielgerichtet zu recherchieren. Dabei sind insbesondere wichtige Grundlagen, andere aktuelle Produkte und die vorgesehene Anwendung näher zu betrachten. Die Analyse der grundsätzlichen Löungsvarianten soll helfen, deren Potential beurteilen zu können. Die gesetzten Rahmenbedingungen, gewählte Lösung und die Funktionsweise sind gut nachvollziehbar zu dokumentieren. Die Messergebnisse sind in exemplarischem Umfang zu erfassen und auszuwerten. Die realisierten Lösungen und die Ergebnisse sind kritisch einordnend zu bewerten. Ansätze für Verbesserungen und weitere Arbeiten sind zu nennen.

B Schaltpläne

B.1 „BATSEN ZS Klasse 3 v0.1“

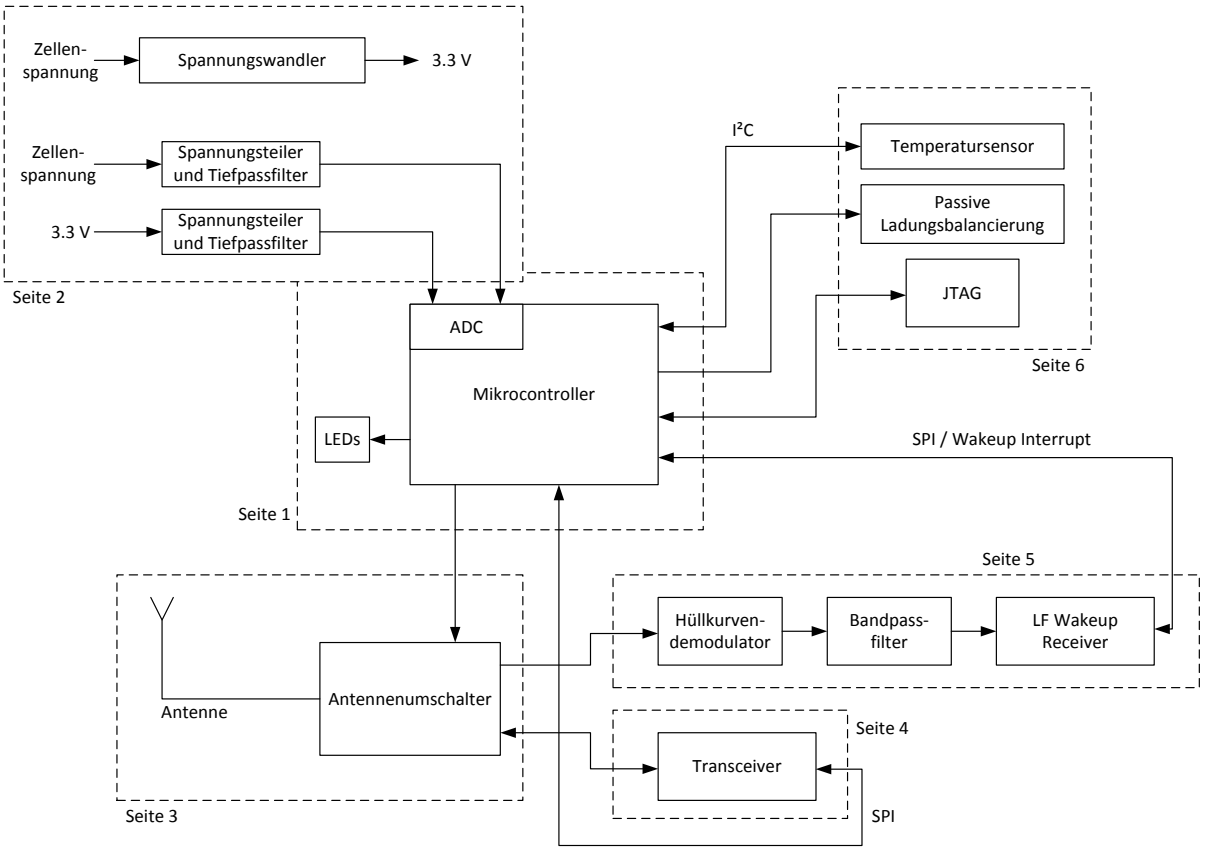
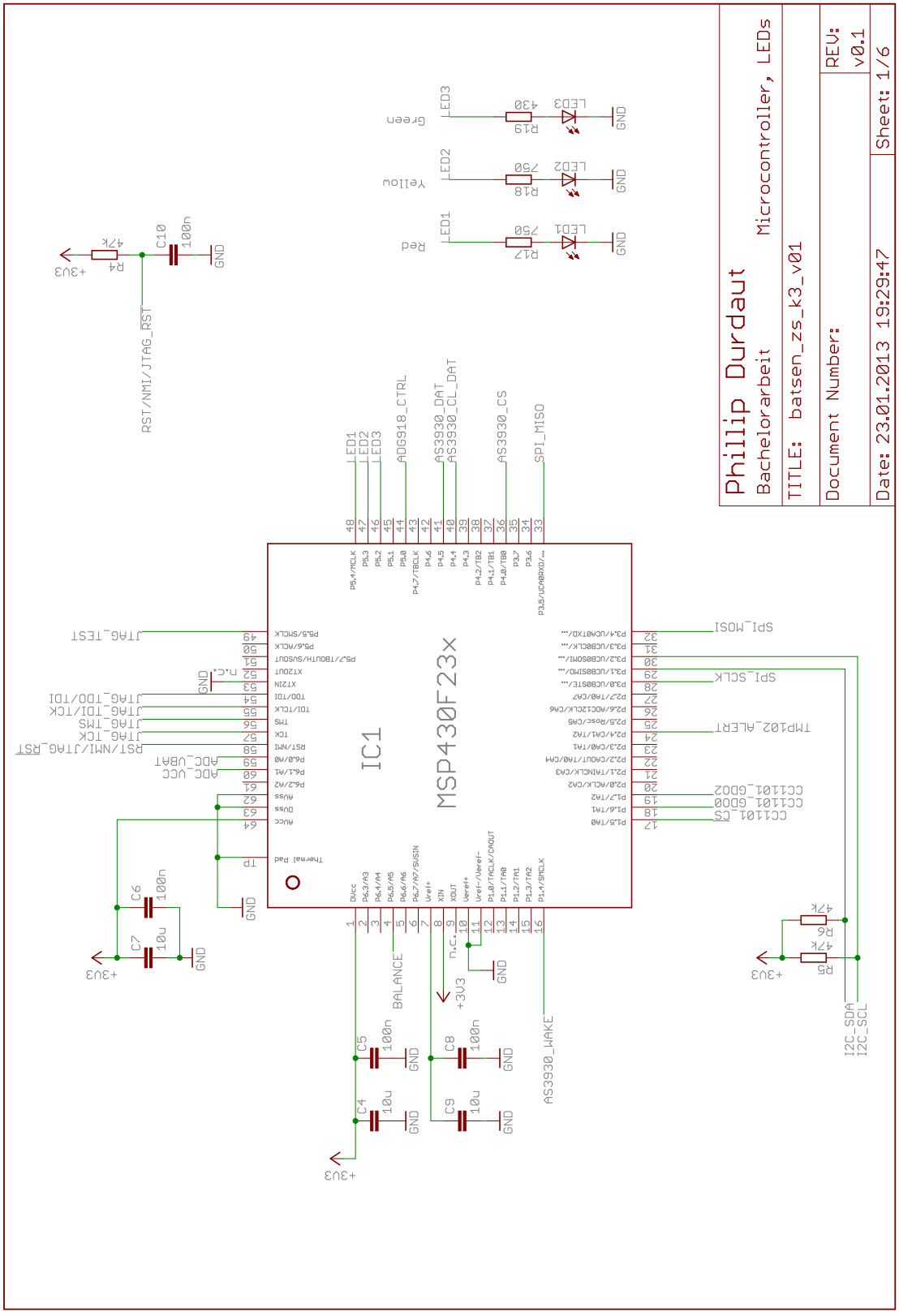
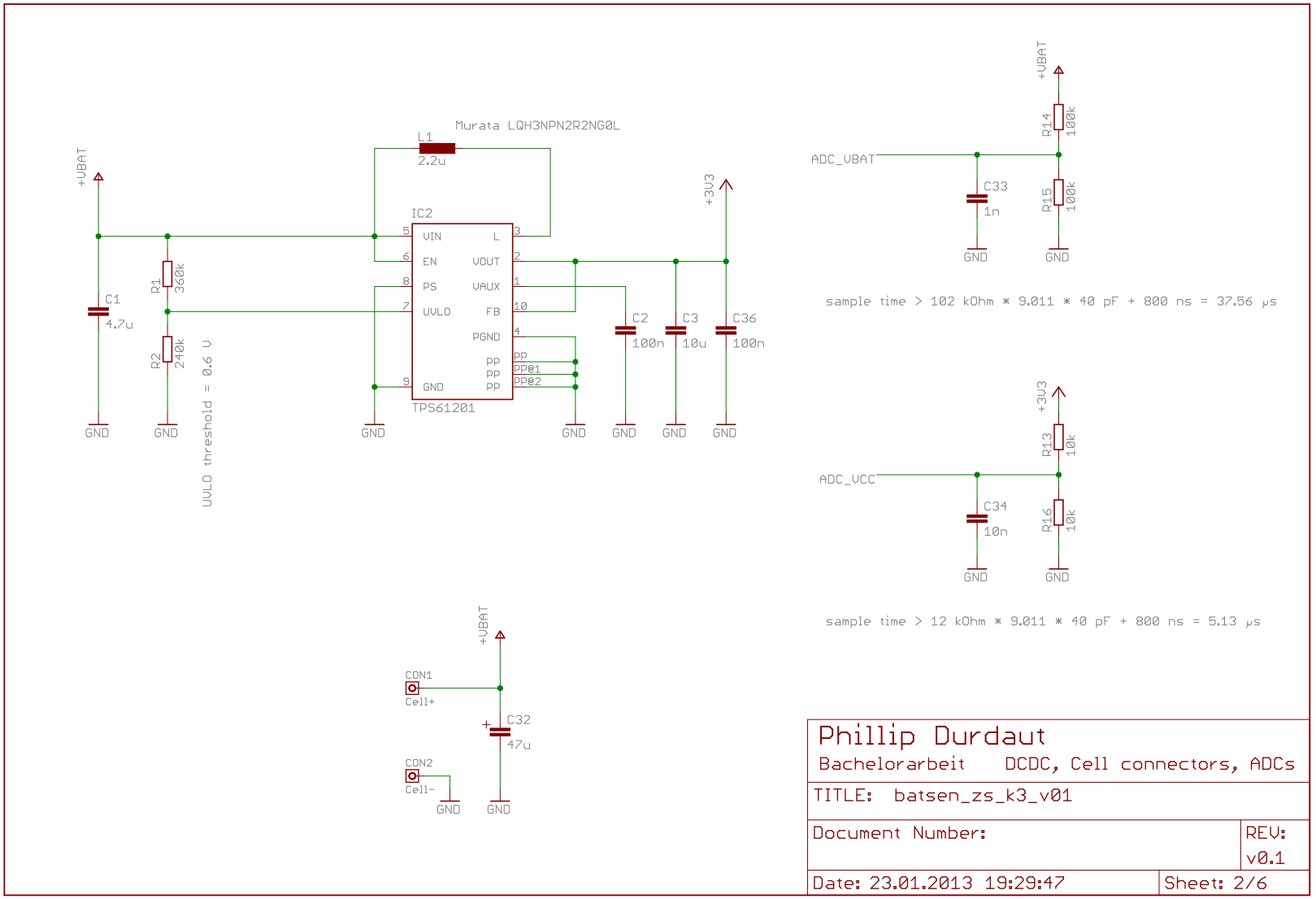
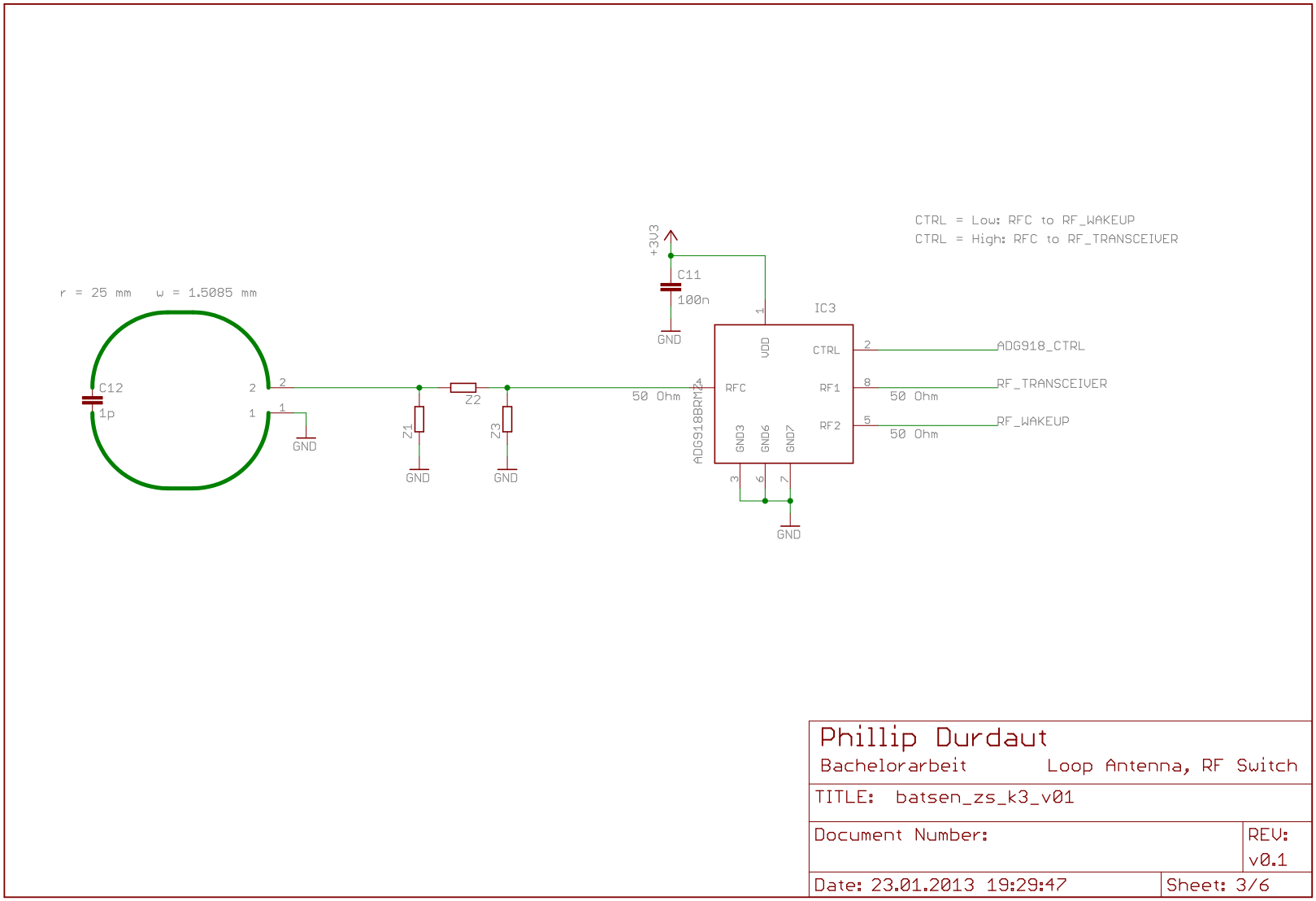


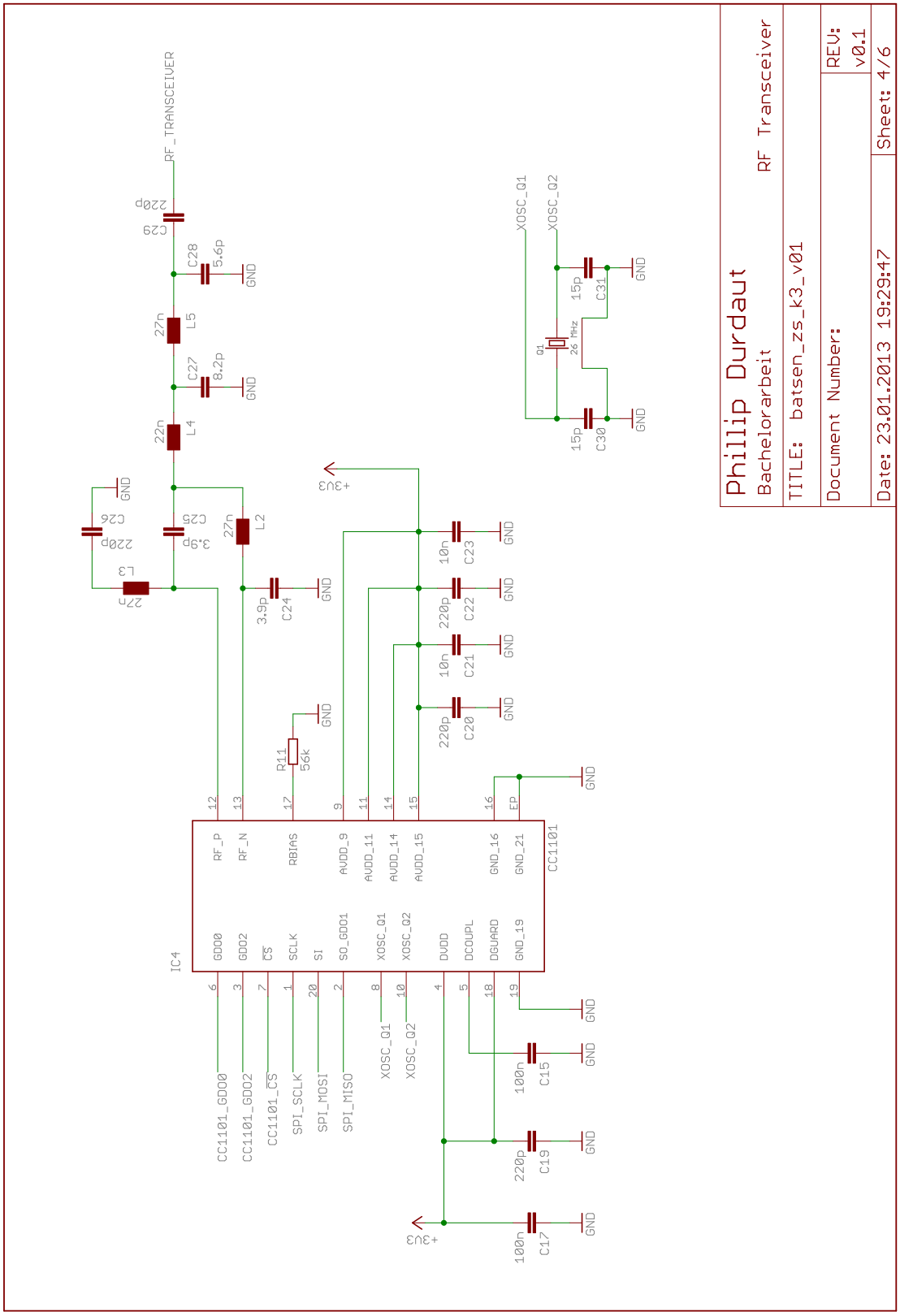
Abbildung B.1: Blockschaltbild zum entwickelten Zellensensor. Die angegebenen Seitenzahlen beziehen sich auf den folgenden Schaltplan des Zellensensors.

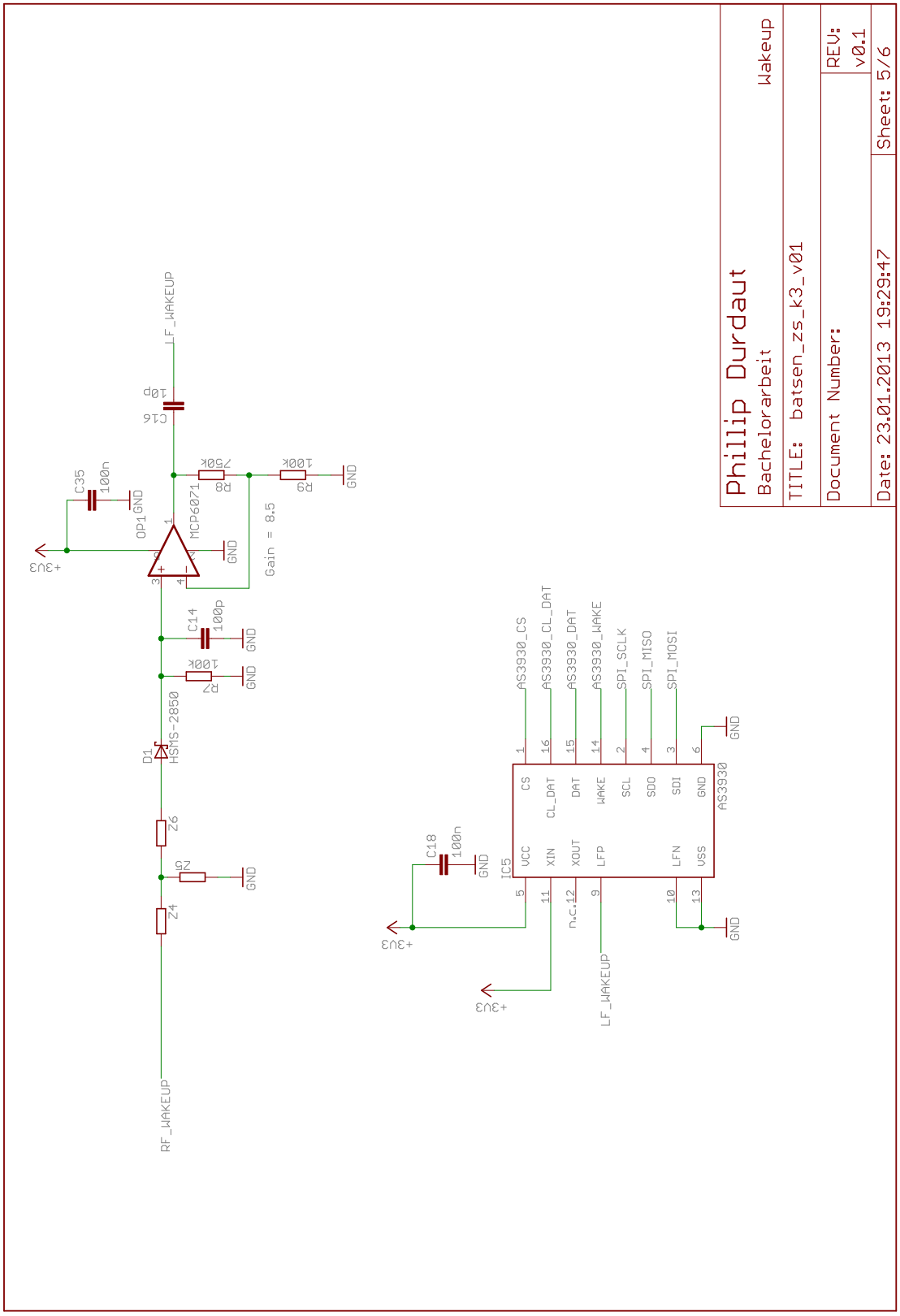

Phillip Durdaut
Bachelorarbeit
Microcontroller, LEDs
TITLE: batsen_zs_k3_v01
Document Number:
REV: v0.1
Date: 23.01.2013 19:29:47
Sheet: 1/6
IC1
MSP430F23x
LED1
LED2
LED3
ADS918_CTRL
AS3930_DAT
AS3930_CL_DAT
AS3930_CS
SPI_MISO
SPI_MOSI
SPI_SCLK
TMP102_ALERT
CC1101_CS
CC1101_GDO0
CC1101_GDO2
I2C_SDA
I2C_SCL
AS3930_WAKE
BALANCE
RST/NMI/JTAG_RST
JTAG_TEST
JTAG_TCK
JTAG_TMS
JTAG_TDI/TCK
JTAG_TDO/TDI
ADC_UBAT
ADC_VCC
Red
Yellow
Green
R17
R18
R19
R4
R5
C4
C5
C6
C7
C8
C9
C10
+3V3
GND

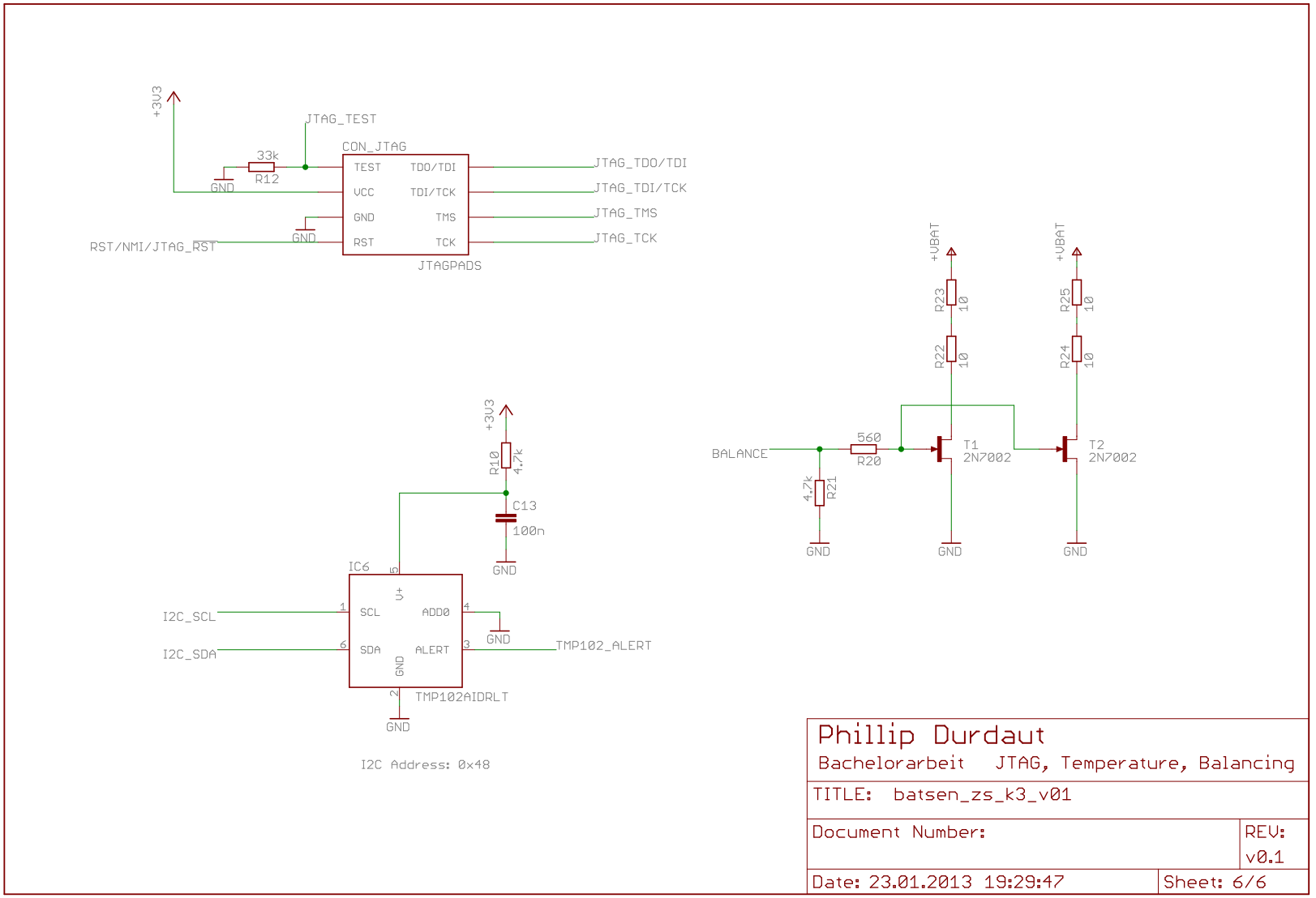

Murata LQH3NPN2R2NG0L
L1
2.2u
+VBAT
IC2
VIN
EN
PS
UVLO
L
VOUT
VAUX
FB
PGND
PP
PP
PP
GND
TPS61201
R1
360k
R2
240k
C1
4.7u
C2
100n
C3
10u
C36
100n
+3V3
GND
UVLO threshold = 0.6 V
ADC_VBAT
R14
100k
R15
100k
C33
1n
sample time > 102 kOhm * 9.011 * 40 pF + 800 ns = 37.56 µs
ADC_VCC
R13
10k
R16
10k
C34
10n
sample time > 12 kOhm * 9.011 * 40 pF + 800 ns = 5.13 µs
CON1
Cell+
CON2
Cell-
C32
47u
Phillip Durdaut
Bachelorarbeit DCDC, Cell connectors, ADCs
TITLE: batsen_zs_k3_v01
Document Number:
REV: v0.1
Date: 23.01.2013 19:29:47
Sheet: 2/6



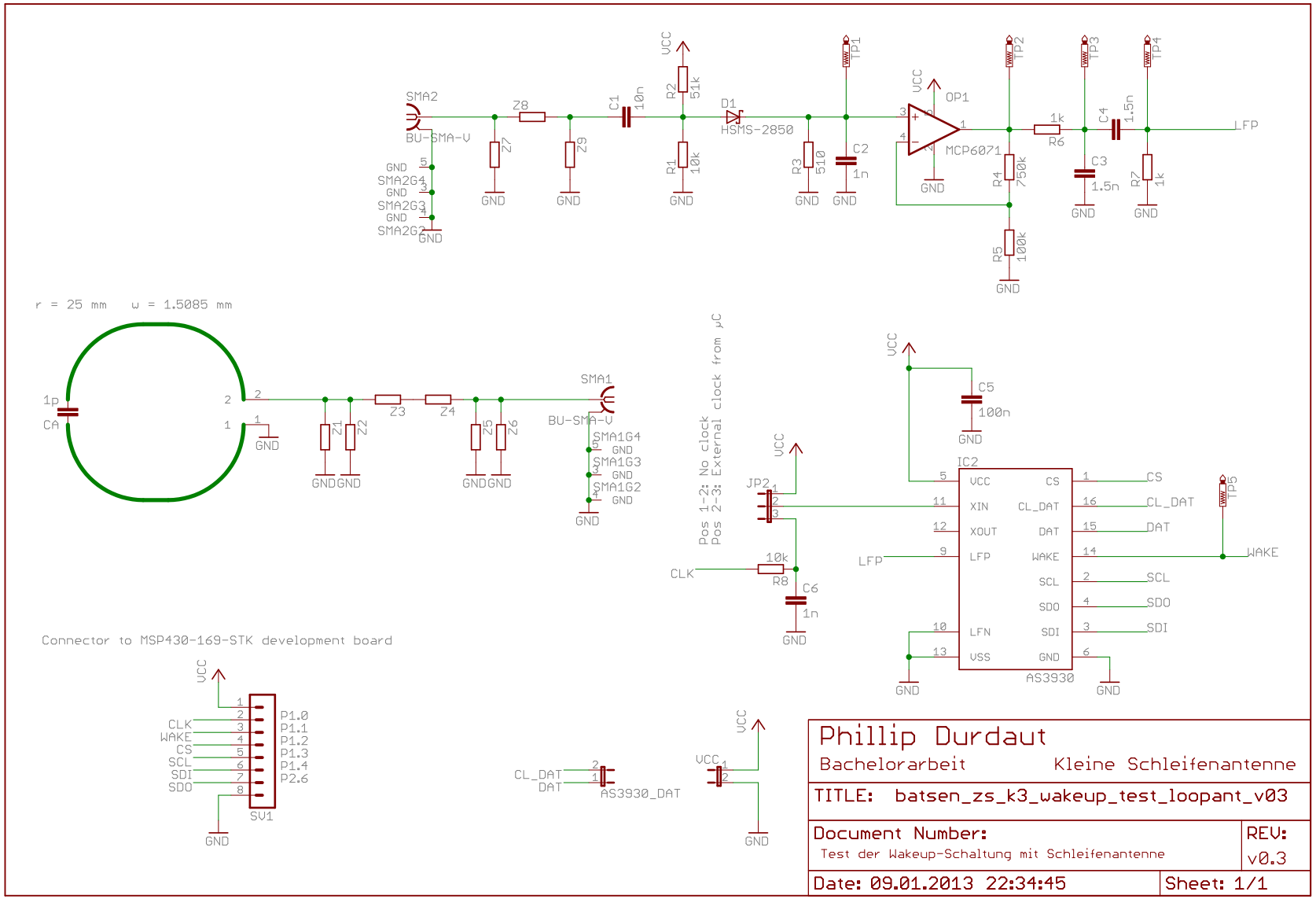
Phillip Durdaut	
Bachelorarbeit — Loop Antenna, RF Switch	
TITLE: batsen_zs_k3_v01	
Document Number:	REV: v0.1
Date: 23.01.2013 19:29:47	Sheet: 3/6


RF_TRANSCEIVER
C29 220p
C28 5.6p
L5 27n
C27 8.2p
L4 22n
C26 220p
C25 3.9p
L2 27n
L3 27n
C24 3.9p
R11 56k
C23 10n
C22 220p
C21 10n
C20 220p
C19 220p
C15 100n
C17 100n
C31 15p
C30 15p
Q1 26 MHz
XOSC_Q1
XOSC_Q2
+3V3
GND
IC4
CC1101
RF_P
RF_N
RBIAS
AVDD_9
AVDD_11
AVDD_14
AVDD_15
GND_16
GND_21
EP
GDO0
GDO2
CS
SCLK
SI
SO_GDO1
XOSC_Q1
XOSC_Q2
DVDD
DCOUPL
DGUARD
GND_19
CC1101_GDO0
CC1101_GDO2
CC1101_CS
SPI_SCLK
SPI_MOSI
SPI_MISO
Philip Durdaut
Bachelorarbeit
RF Transceiver
TITLE: batsen_zs_k3_v01
Document Number:
REV: v0.1
Date: 23.01.2013 19:29:47
Sheet: 4/6


Phillip Durdaut
Bachelorarbeit
Wakeup
TITLE: batsen_zs_k3_v01
Document Number:
REV: v0.1
Date: 23.01.2013 19:29:47
Sheet: 5/6
LF_WAKEUP
C16 10p
C35 100n
OP1
MCP6071
R9 750K
R10 100K
GND
+3V3
Gain = 8.5
C14 100p
R7 100K
D1 HSMS-2850
Z6
Z5
Z4
RF_WAKEUP
C18 100n
IC5
AS3930
AS3930_CS
AS3930_CL_DAT
AS3930_DAT
AS3930_WAKE
SPI_SCLK
SPI_MISO
SPI_MOSI
CS
CL_DAT
DAT
WAKE
SCL
SDO
SDI
GND
VCC
XIN
XOUT
LFP
LFN
VSS
n.c.12


+3V3
JTAG_TEST
33k
R12
GND
CON_JTAG
TEST
VCC
GND
RST
TDO/TDI
TDI/TCK
TMS
TCK
JTAG_TDO/TDI
JTAG_TDI/TCK
JTAG_TMS
JTAG_TCK
RST/NMI/JTAG_RST
GND
JTAGPADS
+3V3
R10
4.7k
C13
100n
GND
IC6
V+
SCL
SDA
ADD0
ALERT
GND
I2C_SCL
I2C_SDA
TMP102_ALERT
TMP102AIDRLT
GND
I2C Address: 0x48
+VBAT
R23
10
R22
10
+VBAT
R25
10
R24
10
BALANCE
560
R20
4.7k
R21
T1
2N7002
T2
2N7002
GND
GND
GND
Phillip Durdaut
Bachelorarbeit JTAG, Temperature, Balancing
TITLE: batsen_zs_k3_v01
Document Number:
REV:
v0.1
Date: 23.01.2013 19:29:47
Sheet: 6/6

B.2 „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Loop Antenna“



Phillip Durdaut

Bachelorarbeit — Kleine Schleifenantenne

TITLE: batsen_zs_k3_wakeup_test_loopant_v03

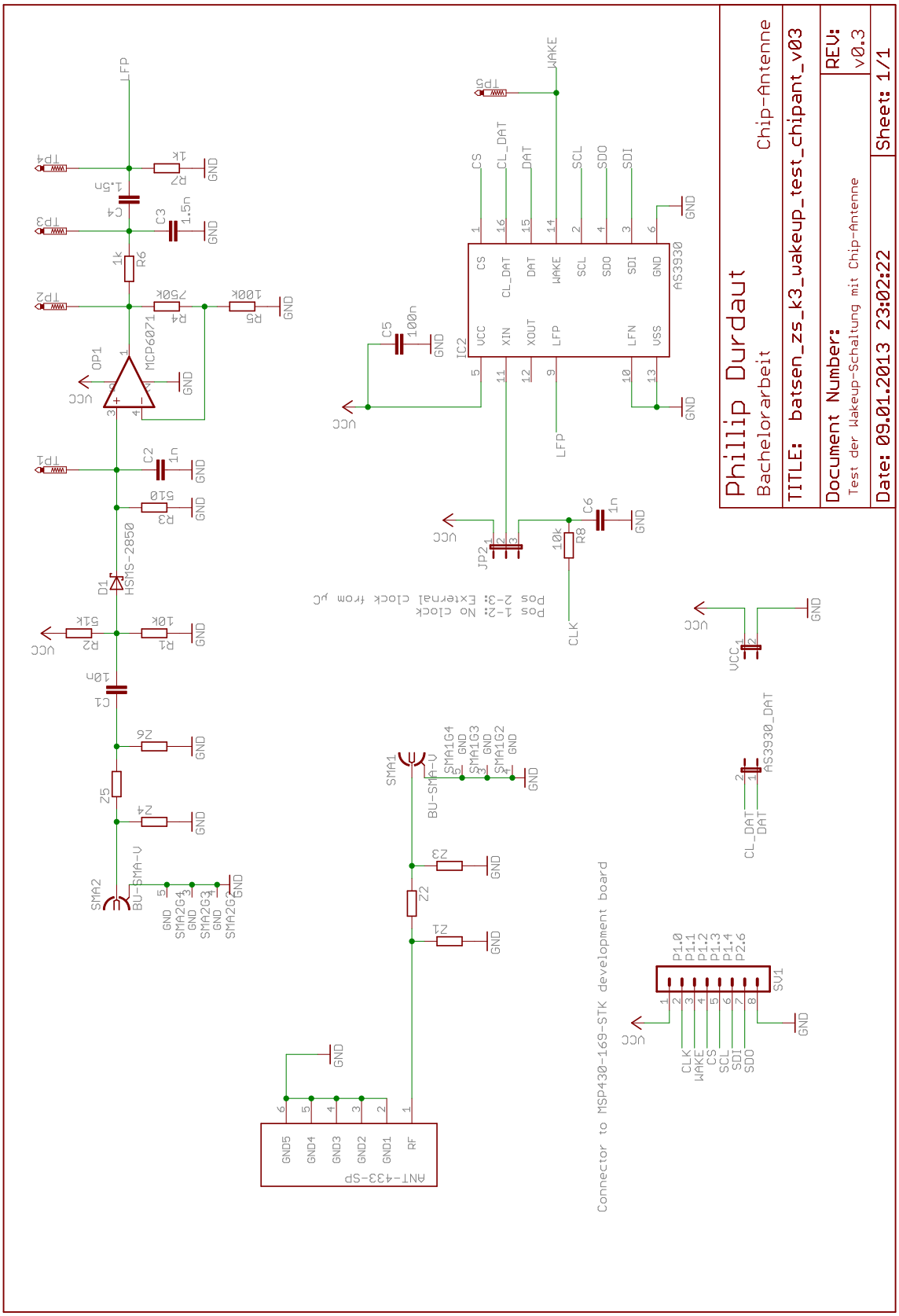
Document Number: Test der Wakeup-Schaltung mit Schleifenantenne

REV: v0.3

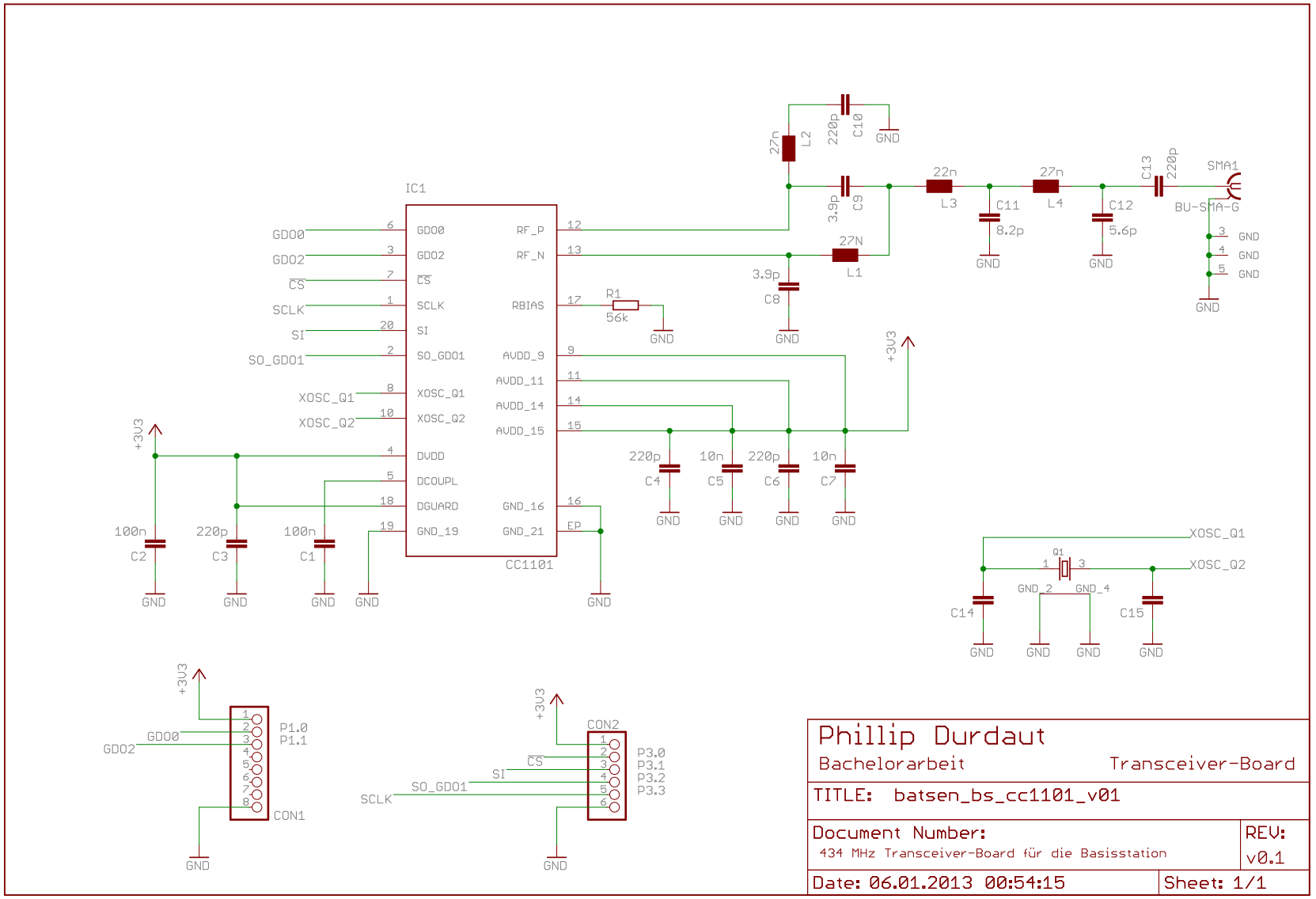
Date: 09.01.2013 22:34:45

Sheet: 1/1

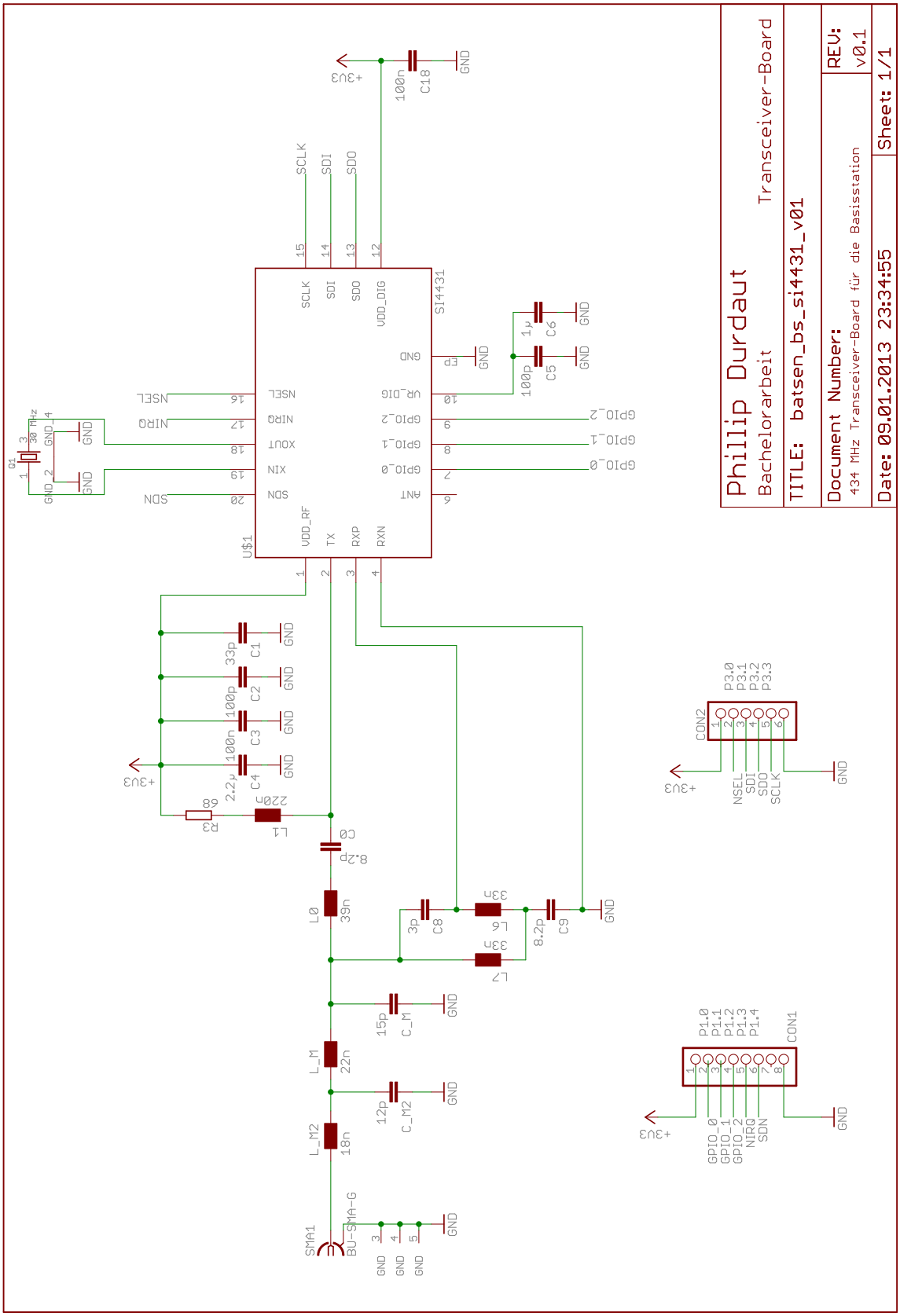
B.3 „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Chip Antenna“


Phillip Durdaut
Bachelorarbeit
Chip-Antenne
TITLE: batsen_zs_k3_wakeup_test_chipant_v03
Document Number:
Test der Wakeup-Schaltung mit Chip-Antenne
REV: v0.3
Date: 09.01.2013 23:02:22
Sheet: 1/1
Connector to MSP430-169-STK development board
AS3930
ANT-433-SP
MCP6071
HSMS-2850
BU-SMA-V

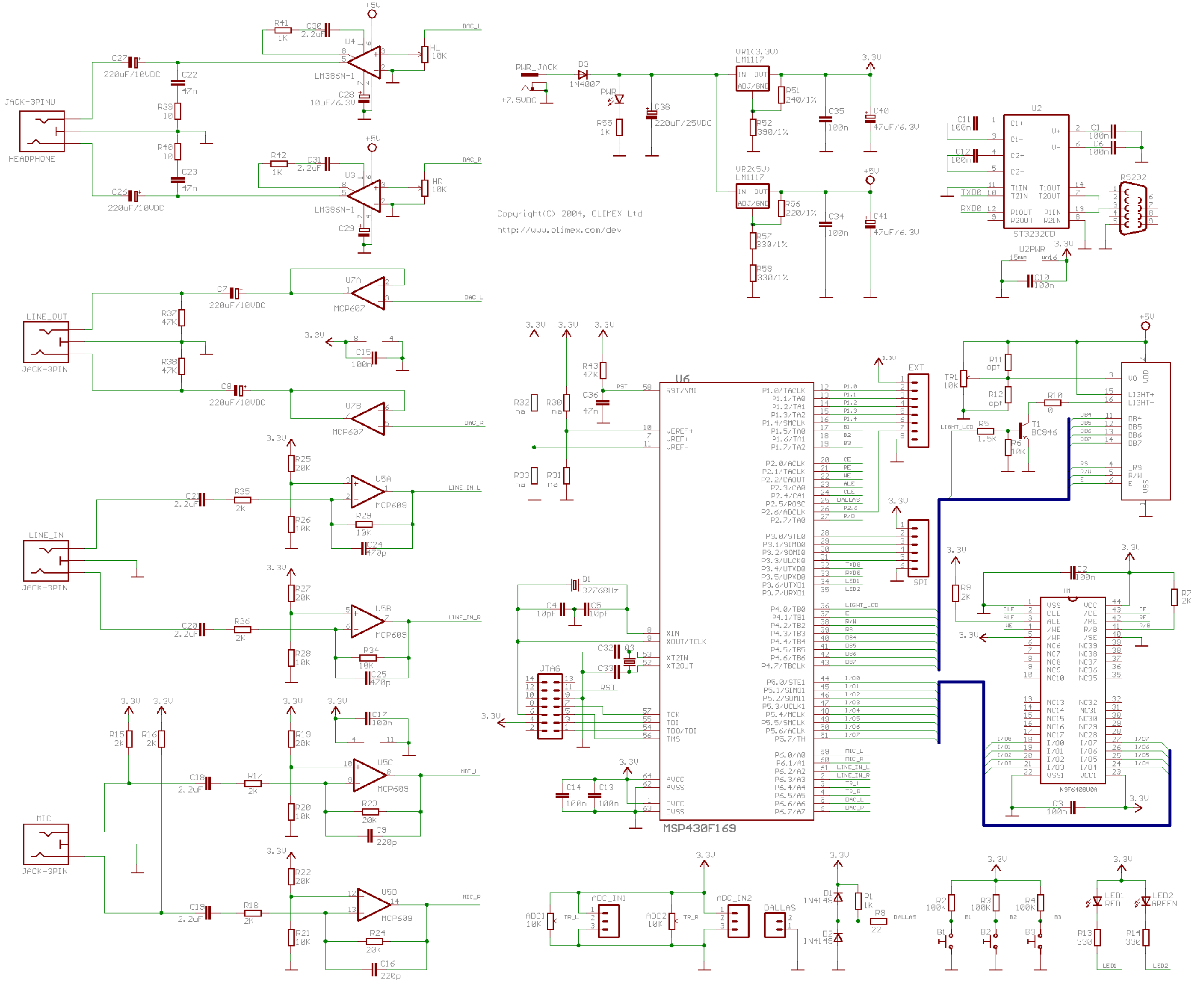
B.4 „BATSEN 434 MHz Transceiver-Board für die Basisstation v0.1 - Transceiver: CC1101“


IC1
CC1101
GDO0
GDO2
CS
SCLK
SI
SO_GDO1
XOSC_Q1
XOSC_Q2
RF_P
RF_N
RBIAS
AVDD_9
AVDD_11
AVDD_14
AVDD_15
DVDD
DCOUPL
DGUARD
GND_16
GND_19
GND_21
EP
+3V3
GND
R1
56k
C1
100n
C2
100n
C3
220p
C4
220p
C5
10n
C6
220p
C7
10n
C8
3.9p
C9
3.9p
C10
220p
C11
8.2p
C12
5.6p
C13
220p
C14
C15
L1
27N
L2
27n
L3
22n
L4
27n
SMA1
BU-SMA-G
Q1
GND_2
GND_4
CON1
P1.0
P1.1
CON2
P3.0
P3.1
P3.2
P3.3
Phillip Durdaut
Bachelorarbeit
Transceiver-Board
TITLE: batsen_bs_cc1101_v01
Document Number:
434 MHz Transceiver-Board für die Basisstation
REV:
v0.1
Date: 06.01.2013 00:54:15
Sheet: 1/1

B.5 „BATSEN 434 MHz Transceiver-Board für die Basisstation v0.1 - Transceiver: Si4431“

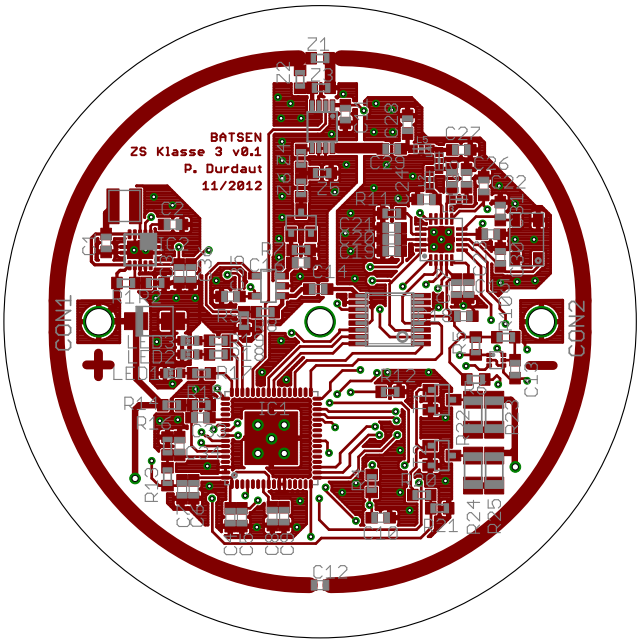

Phillip Durdaut
Bachelorarbeit
Transceiver-Board
TITLE: batsen_bs_si4431_v01
Document Number:
434 MHz Transceiver-Board für die Basisstation
REV: v0.1
Date: 09.01.2013 23:34:55
Sheet: 1/1
U$1
SI4431
SCLK
SDI
SDO
VDD_DIG
GND
VR_DIG
GPIO_2
GPIO_1
GPIO_0
ANT
VDD_RF
TX
RXP
RXN
NSEL
NIRQ
XOUT
XIN
SDN
Q1
30 MHz
C1
C2
C3
C4
C5
C6
C8
C9
C18
L0
L1
L6
L7
L_M
L_M2
C_M
C_M2
R3
SMA1
CON1
CON2
P1.0
P1.1
P1.2
P1.3
P1.4
P3.0
P3.1
P3.2
P3.3
GPIO_0
GPIO_1
GPIO_2
NIRQ
SDN
NSEL
SDI
SDO
SCLK
+3V3
GND

B.6 Entwicklungsboard MSP430-169STK

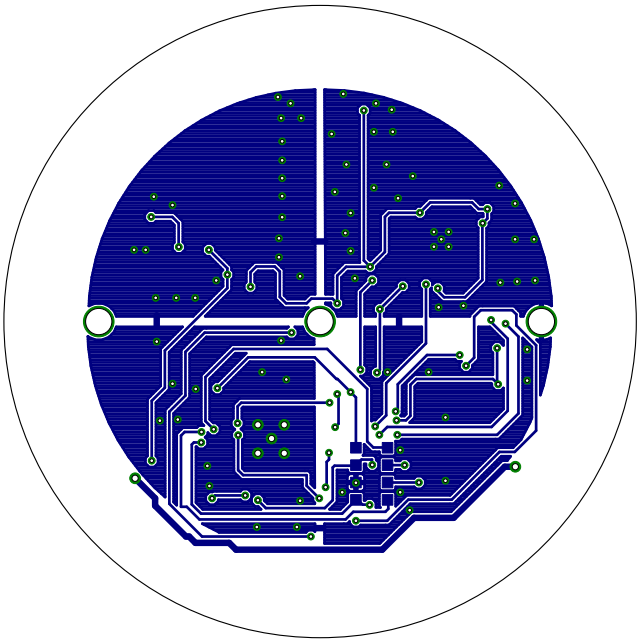

Copyright(C) 2004, OLIMEX Ltd
http://www.olimex.com/dev
JACK-3PINV
HEADPHONE
LM386N-1
LINE_OUT
MCP607
LINE_IN
MCP609
MIC
JACK-3PIN
PWR_JACK
+7.5VDC
1N4007
VR1(3.3V)
LM1117
VR2(5V)
LM1117
ST3232CD
RS232
U6
MSP430F169
JTAG
EXT
SPI
Q1
32768Hz
K9F6408U0A
ADC_IN1
ADC_IN2
DALLAS
LED1
RED
LED2
GREEN

C Platinenlayouts

C.1 „BATSEN ZS Klasse 3 v0.1“



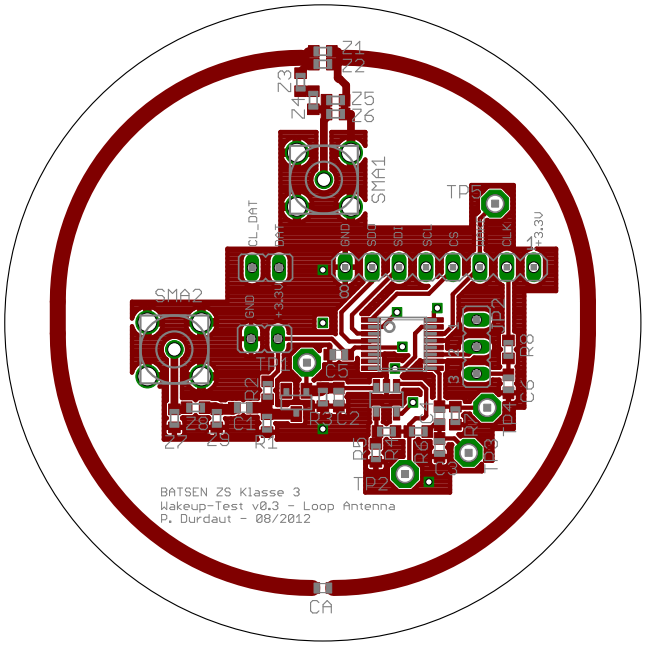
(a) Oberseite



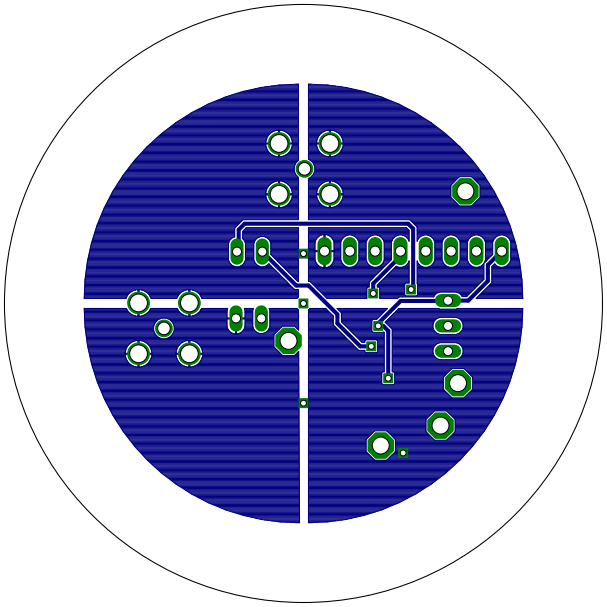
(b) Unterseite

Abbildung C.1: Platinenlayout „BATSEN ZS Klasse 3 v0.1“

C.2 „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Loop Antenna“



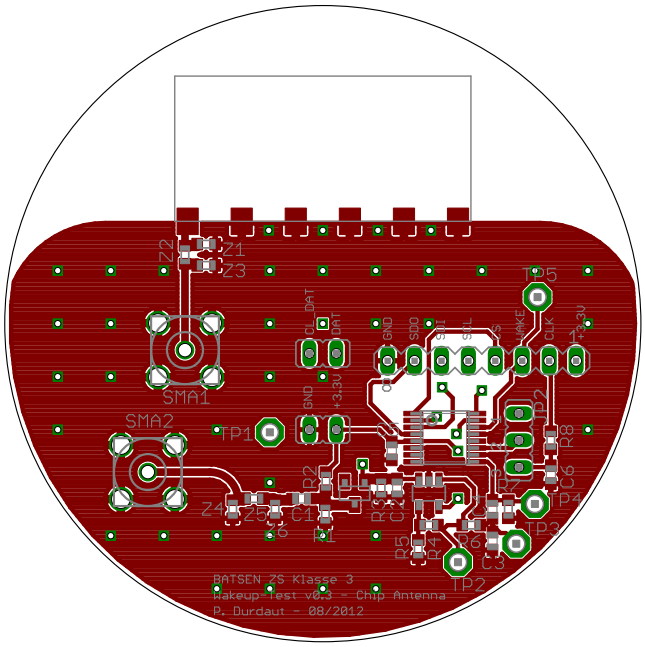
(a) Oberseite



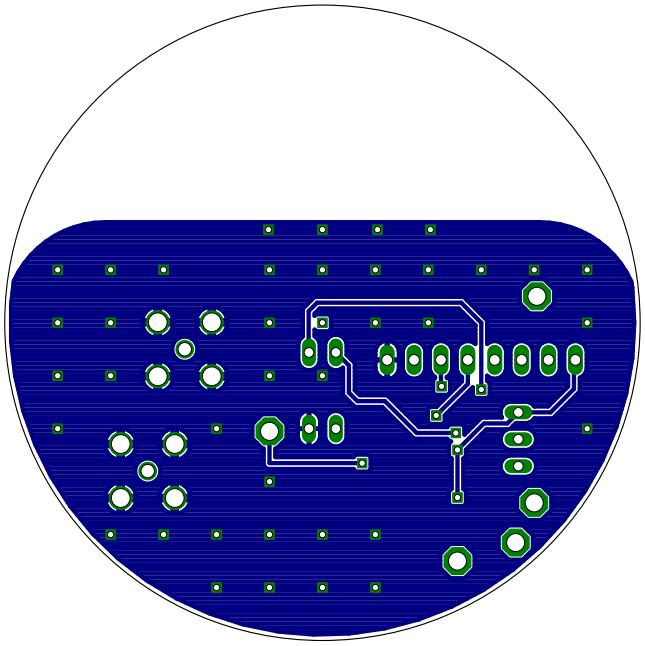
(b) Unterseite

Abbildung C.2: Platinenlayout „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Loop Antenna“

C.3 „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Chip Antenna“



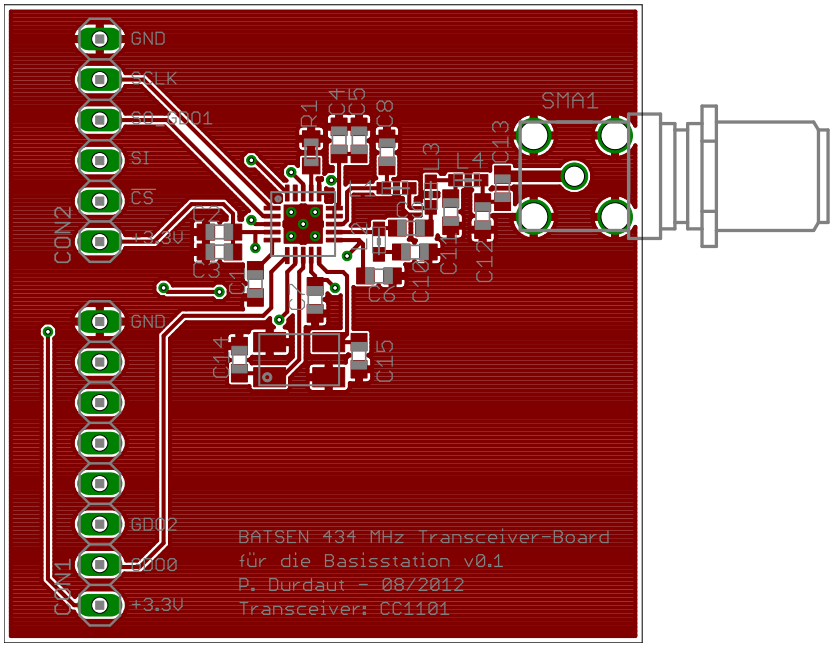
(a) Oberseite



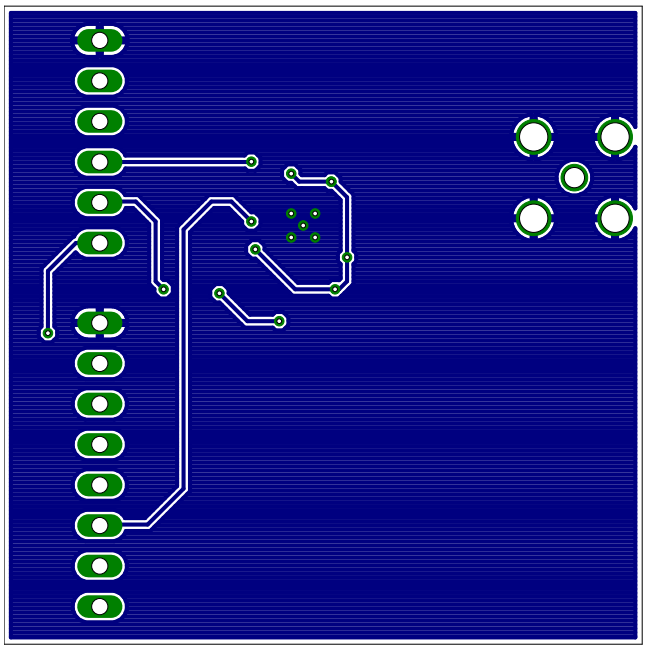
(b) Unterseite

Abbildung C.3: Platinenlayout „BATSEN ZS Klasse 3 Wakeup-Test v0.3 - Chip Antenna“

C.4 „BATSEN 434 MHz Transceiver-Board für die Basisstation v0.1 - Transceiver: CC1101“



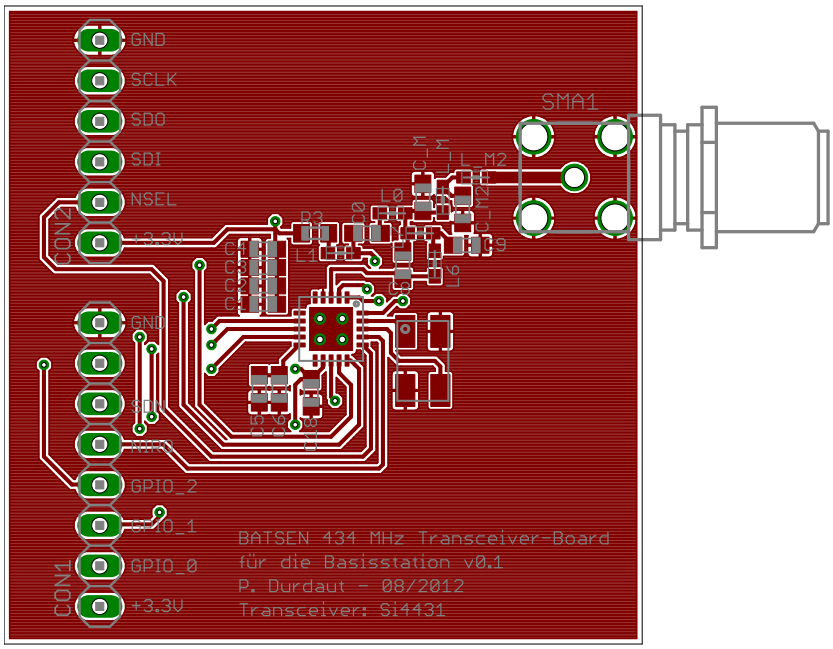
(a) Oberseite



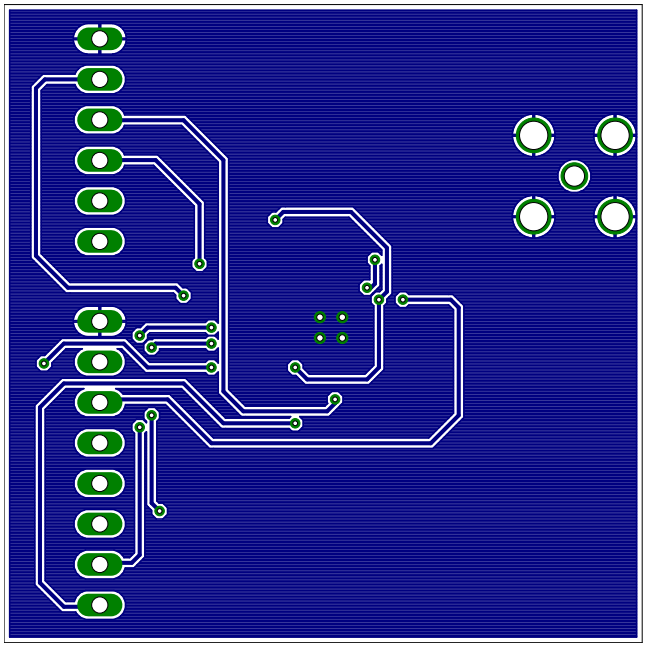
(b) Unterseite

Abbildung C.4: Platinenlayout „BATSEN BS CC1101 v0.1“

C.5 „BATSEN 434 MHz Transceiver-Board für die Basisstation v0.1 - Transceiver: Si4431“



(a) Oberseite



(b) Unterseite

Abbildung C.5: Platinenlayout „BATSEN BS Si4431 v0.1“

D Programmcode

D.1 Mikrocontroller

D.1.1 Zellensensor

Listing D.1: main.c

```
/*-------------------------------------------------------------------------------
   Description: Main program for the class 3 cell sensor used during
                trial measurements.
   Hardware:    BATSEN ZS Klasse 3 v0.1 - P. Durdaut - 11/2012
   Date:        11/22/2012
   Last Update: 11/28/2012
   Author:      Phillip Durdaut
-------------------------------------------------------------------------------*/

#include "main.h"

/*-------------------------------------------------------------------------------
 Global variables
-------------------------------------------------------------------------------*/

volatile state_t g_state = S_INIT; // The state of the sensor
u16_t g_sample = 0;          // The latest cell voltage sample

/*-------------------------------------------------------------------------------
 Prototypes of the private functions
-------------------------------------------------------------------------------*/

void init_and_sleep(void);
void rx(void);
void tx_carrier(void);
void tx_single_packet(u8_t * txbuf);
void tx_continuous_packets(u8_t * txbuf);

/*-------------------------------------------------------------------------------
 Main function
-------------------------------------------------------------------------------*/

int main(void)
{
   //*************************************************************
   // DCO - Digital Controlled Oscillator
   //*************************************************************

   // Calibrate DCO with calibration data for 1 MHz
   DCOCTL = CALDCO_1MHZ;

   // Turn off crystal oscillator XT2
   BCSCTL1 |= XT2OFF;

   // Load BCSCTL1 calibration Data for 1 MHz
   BCSCTL1 |= CALBC1_1MHZ;

   // MCLK = DCOCLK / 1
   BCSCTL2 |= BIT6;
   BCSCTL2 &= ~(BIT5 | BIT4);

   // SMCLK = DCOCLK / 1
   BCSCTL2 &= ~BIT3;
   BCSCTL2 &= ~(BIT2 | BIT1);

   // No external DCO resistor
   BCSCTL2 &= ~BIT0;

   // SMCLK output on pin P5.5
   //P5SEL |= BIT5;
   //P5DIR |= BIT5;

   //*************************************************************
   // Initialization
   //*************************************************************
```

```
66
67      // LEDs
68      led_init();
69
70      // All LEDs on for 3 seconds on startup
71      led_on(LED_ALL);
72      delay_ms(3000);
73      led_off(LED_ALL);
74
75      // Initialize periphery for entering sleep mode
76      init_and_sleep();
77
78      //*************************************************************
79      // Endless loop
80      //*************************************************************
81      while (1);
82
83      return 0;
84  }
85
86  /*-------------------------------------------------------------------------------------
87    Interrupt service handler
88  -------------------------------------------------------------------------------------*/
89
90  interrupt (PORT1_VECTOR) isr_port1(void)
91  {
92      if (AS3930_WAKE_IRQ_PENDING) { // Wake interrupt occured
93
94          // Exit Low Power Mode 4
95          EXIT_LPM4;
96
97          // Disable and clear the wake interrupt as the sensor is awake now
98          AS3930_WAKE_IRQ_DISABLE;
99          AS3930_WAKE_CLEAR_IRQ;
100
101         // Turn on the red LED
102         led_on(LED_AWAKE);
103
104         // Change to RX state
105         rx();
106     }
107
108     if (CC1101_GDO2_IRQ_PENDING) { // Sync word detected
109
110         // The received command
111         u8_t received_command = COMMAND_DOWNLINK_UNKWOWN;
112
113         // Disable and clear the sync word detected interrupt
114         CC1101_GDO2_IRQ_DISABLE;
115         CC1101_GDO2_CLEAR_IRQ;
116
117         // Wait until packet completely received
118         while(CC1101_GDO2_IN);
119
120         // Check whether received data is valid
121         if (CC1101_GDO0_IN == 1 && cc1101_get_rxbytes() == DOWNLINK_DATA_BYTES) {
122
123             // 6 bytes of data expected
124             u8_t rxbuf[] = { 0, 0, 0, 0, 0, 0 };
125             cc1101_read_rx_fifo(rxbuf, DOWNLINK_DATA_BYTES);
126
127             // Data packet for this sensor?
128             if (rxbuf[0] == BROADCAST || rxbuf[0] == ADDRESS_THIS_SENSOR) {
129
130                 led_on(LED_RX);
131
132                 // The first byte is the destination address (this sensor)
```

```
133             // The second byte is the command
134             // The last four bytes are currently unused
135             received_command = rxbuf[1];
136 
137             led_off(LED_RX);
138          }
139       }
140 
141       // Determine what the base station wants this sensor to do now
142       switch(received_command) {
143 
144          case COMMAND_DOWNLINK_UNKWOWN: {
145 
146             // Sync word detected, but no valid packet received
147             // Change to RX state
148             rx();
149          }
150          break;
151 
152          case COMMAND_DOWNLINK_IS_AWAKE: { // Send AWAKE command
153 
154             delay_ms(30);
155 
156             // 1 byte target address + 1 byte sensor address + 9 bytes zero padding
157             u8_t txisawakebuf[] = { ADDRESS_BASE_STATION, ADDRESS_THIS_SENSOR, 0, 0, 0, 0, 0,
                    0, 0, 0, 0 };
158             tx_single_packet(txisawakebuf);
159 
160             // Change to RX state
161             rx();
162          }
163          break;
164 
165          case COMMAND_DOWNLINK_SAMPLE: { // Sample the cell voltage
166 
167             // Get the current cell voltage
168             g_sample = adc12_sample();
169 
170             // Change to RX state
171             rx();
172          }
173          break;
174 
175          case COMMAND_DOWNLINK_SEND_SAMPLE: { // Send the cell voltage
176 
177             delay_ms(30);
178 
179             u8_t sample_msb = (u8_t)((g_sample >> 8) & 0x0F);
180             u8_t sample_lsb = (u8_t)((g_sample >> 0) & 0xFF);
181 
182             // 1 byte target address + 1 byte sensor address + 2 bytes cell voltage + 7 bytes
                    zero padding
183             u8_t txsendsamplebuf[] = { ADDRESS_BASE_STATION, ADDRESS_THIS_SENSOR, sample_msb,
                    sample_lsb, 0, 0, 0, 0, 0, 0, 0 };
184             tx_single_packet(txsendsamplebuf);
185 
186             // Change to RX state
187             rx();
188          }
189          break;
190 
191          case COMMAND_DOWNLINK_SLEEP: { // Go to sleep
192 
193             // Switch all LEDs on for one second
194             led_on(LED_ALL);
195             delay_ms(1000);
196             led_off(LED_ALL);
```

```
197
198             // Initialize periphery for entering sleep mode
199             init_and_sleep();
200          }
201          break;
202       }
203    }
204 }
205
206 /*-------------------------------------------------------------------------------------
207  Private functions
208 -------------------------------------------------------------------------------------*/
209
210 void init_and_sleep(void)
211 {
212    // RF-switch
213    adg918_init();
214    adg918_wakeup(); // Connect the loop antenna with the wakeup circuit
215
216    // Balancing unit
217    balancing_init();
218    balancing_off(); // Balancing unit off
219
220    // CC1101 transceiver
221    // tx_carrier();
222    cc1101_init();
223    cc1101_power_up_reset();
224    cc1101_sleep();      // Power down state
225
226    // Configure packet received interrupt
227    CC1101_GDO2_CLEAR_IRQ;
228    CC1101_GDO2_IRQ_DISABLE;
229
230    // LF wakeup receiver
231    as3930_init();
232    as3930_preset_default(); // Reset
233    as3930_config_no_pattern(); // Wakeup upon LF carrier detection
234    as3930_clear_wakeup(); // Clear wakeup
235
236    // Configure Wake interrupt
237    AS3930_WAKE_CLEAR_IRQ;
238    AS3930_WAKE_IRQ_ENABLE;
239
240    // Sensor goes to sleep state now
241    g_state = S_SLEEP;
242
243    // Global interrupt enable
244    _EINT();
245
246    // Enter Low Power Mode 4 (LPM4) with all clocks disabled
247    // (wait for wakeup)
248    ENTER_LPM4;
249 }
250
251 void rx(void)
252 {
253    g_state = S_RX;        // Sensor in RX state
254    adg918_transceiver(); // Connect the loop antenne with the transceiver
255    cc1101_init();
256    cc1101_reset();      // Reset transceiver and go to idle state
257    cc1101_config_packet(); // Configure transceiver for packet reception
258    CC1101_GDO2_CLEAR_IRQ; // Clear the sync word detected interrupt
259    CC1101_GDO2_IRQ_ENABLE; // Enable the sync word detected interrupt
260    cc1101_rx();         // Transceiver in RX state
261 }
262
263 void tx_carrier(void)
```

```
264 {
265    led_on(LED_TX);
266 
267    adg918_transceiver(); // Connect the loop antenna with the transceiver
268    cc1101_init();
269    cc1101_reset();   // Reset chip and go to idle state
270    cc1101_config_no_packet();
271    cc1101_tx_carrier();
272    P1DIR |= BIT6;
273    P1OUT |= 1;
274    while(1);
275 }
276 
277 void tx_single_packet(u8_t * txbuf)
278 {
279    // Send a single packet
280 
281    // Disable and clear the sync word detected interrupt
282    CC1101_GDO2_IRQ_DISABLE;
283    CC1101_GDO2_CLEAR_IRQ;
284 
285    led_on(LED_TX);
286 
287    adg918_transceiver(); // Connect the loop antenna with the transceiver
288    cc1101_init();
289    cc1101_reset();   // Reset chip and go to idle state
290    cc1101_config_packet(); // Configure the transceiver for sending packets
291 
292    cc1101_fill_tx_fifo(txbuf, UPLINK_DATA_BYTES);
293    cc1101_tx();
294    cc1101_idle();
295 
296    led_off(LED_TX);
297 }
298 
299 void tx_continuous_packets(u8_t * txbuf)
300 {
301    // Send a packet every 5 seconds
302 
303    // Disable and clear the sync word detected interrupt
304    CC1101_GDO2_IRQ_DISABLE;
305    CC1101_GDO2_CLEAR_IRQ;
306 
307    adg918_transceiver(); // Connect the loop antenna with the transceiver
308    cc1101_init();
309    cc1101_reset();   // Reset chip and go to idle state
310    cc1101_config_packet(); // Configure the transceiver for sending packets
311 
312    while (1) {
313       led_on(LED_TX);
314       cc1101_fill_tx_fifo(txbuf, UPLINK_DATA_BYTES);
315       cc1101_tx();
316       cc1101_idle();
317       delay_ms(1000);
318       led_off(LED_TX);
319       delay_ms(4000);
320    }
321 }
```

Listing D.2: main.h

```
1 /*-------------------------------------------------------------------------------
2    Description: Generic defines and macros.
3    Date:       11/22/2012
4    Last Update: 11/28/2012
5    Author:     Phillip Durdaut
```

```
6   ------------------------------------------------------------------------------*/
7
8   #ifndef MAIN_H_
9   #define MAIN_H_
10
11  #include <msp430x23x.h>
12  #include <signal.h>
13  #include <stdio.h>
14
15  #include "types.h"
16  #include "adc12.h"
17  #include "adg918.h"
18  #include "as3930.h"
19  #include "balancing.h"
20  #include "cc1101.h"
21  #include "delay.h"
22  #include "led.h"
23
24  /*------------------------------------------------------------------------------
25   Fix error in msp430x23x.h
26  ------------------------------------------------------------------------------*/
27
28  #define __MSP430_HAS_ADC12__
29  #include <msp430/adc12.h>
30
31  /*------------------------------------------------------------------------------
32   Defines
33  ------------------------------------------------------------------------------*/
34
35  #define ENABLE_LEDS
36
37  #define DOWNLINK_DATA_BYTES   6      // Incl. address byte
38  #define UPLINK_DATA_BYTES     11     // Incl. address byte
39
40  // Frequency offset added to the base frequency (in units of 1.59 kHz - 1.65 kHz)
41  #define FREQUENCY_OFFSET      6
42
43  // Address definitions
44  #define BROADCAST             0x00
45  #define ADDRESS_BASE_STATION  0xFF
46  #define ADDRESS_THIS_SENSOR   0x01
47
48  // Downlink commands
49  #define COMMAND_DOWNLINK_IS_AWAKE 0x01
50  #define COMMAND_DOWNLINK_SAMPLE 0x02
51  #define COMMAND_DOWNLINK_SEND_SAMPLE 0x03
52  #define COMMAND_DOWNLINK_SLEEP 0x04
53  #define COMMAND_DOWNLINK_UNKWOWN 0xFF
54
55  // Clock frequencies
56  #define DCOCLK                1000000 // DCO Clock frequency
57  #define MCLK                  1000000 // Main System Clock frequency
58  #define SMCLK                 1000000 // Sub System Clock frequency
59
60  /*------------------------------------------------------------------------------
61   Types
62  ------------------------------------------------------------------------------*/
63
64  typedef enum { S_INIT,
65             S_SLEEP,
66             S_RX,
67             S_TX_AWAKE,
68             S_SAMPLE,
69             S_TX_SAMPLE
70           } state_t ;
71
72  /*------------------------------------------------------------------------------
```

```
 Macros
------------------------------------------------------------------------------*/

#define ENTER_LPM4            __bis_SR_register(LPM4_bits + GIE);
#define EXIT_LPM4             __bic_SR_register_on_exit(LPM4_bits);

#endif /* MAIN_H_ */
```

Listing D.3: adc12.c

```
/*------------------------------------------------------------------------------
   Description: ADC (12 bit) driver.
   Date:        12/02/2012
   Last Update: 01/12/2013
   Author:      Phillip Durdaut
------------------------------------------------------------------------------*/

#include "main.h"

/*------------------------------------------------------------------------------
 Prototypes of the private functions
------------------------------------------------------------------------------*/

void adc12_init(void);
void adc12_disable(void);

/*------------------------------------------------------------------------------
 Public functions
------------------------------------------------------------------------------*/

u16_t adc12_sample(void)
{
   u16_t adc_value = 0;

   adc12_init();  // Initialize ADC

   ADC12CTL0 |= (ENC | ADC12SC); // Start conversion with sampling
   while ((ADC12IFG & 0x0001) != 0x0001); // Wait while conversion is active
   adc_value = (ADC12MEM0 & 0x0FFF);

   adc12_disable(); // Disable ADC again for saving energy

   return adc_value;
}

/*------------------------------------------------------------------------------
 Private functions
------------------------------------------------------------------------------*/

void adc12_init(void)
{
   ADC12_VBAT_PxDIR &= ~ADC12_VBAT_PIN; // ADC pin is input
   ADC12_VBAT_PxSEL |= ADC12_VBAT_PIN; // ADC functionality for pin

   ADC12CTL0 |= SHT02; // Sample and Hold 64 ADC12CLK cyles (64 us > 37.56 us)
   ADC12CTL0 |= REF2_5V; // 2.5 V reference voltage
   ADC12CTL0 |= REFON; // Enable reference voltage
   ADC12CTL0 |= ADC12ON; // Enable ADC12

   // Save conversion result to ADC12MEM0
   ADC12CTL1 &= ~(CSTARTADD3 | CSTARTADD2 | CSTARTADD1 | CSTARTADD0);

   // ADC12CLK = SMCLK / 1
   ADC12CTL1 &= ~(ADC12DIV2 | ADC12DIV1 | ADC12DIV0);
   ADC12CTL1 |= (ADC12SSEL1 | ADC12SSEL1);

```

```
57      ADC12CTL1 &= ~(CONSEQ1 | CONSEQ0); // Single-channel, single-conversion
58      ADC12CTL1 |= SHP; // SAMPCON sourced from sampling timer
59
60      // Input channel A0, VR+ = VREF+, VR- = AVss
61      ADC12MCTL0 = SREF_1;
62  }
63
64  void adc12_disable(void)
65  {
66      ADC12CTL0 &= ~ENC; // Disable conversion
67      ADC12CTL0 &= ~REFON; // Disable reference voltage
68      ADC12CTL0 &= ~ADC12ON; // Disable ADC12
69  }
```

Listing D.4: adc12.h

```
1   /*-------------------------------------------------------------------------------
2      Description: ADC (12 bit) driver.
3      Date:        12/02/2012
4      Last Update: 21/01/2013
5      Author:      Phillip Durdaut
6   -------------------------------------------------------------------------------*/
7
8   #ifndef ADC12_H_
9   #define ADC12_H_
10
11  #include "main.h"
12
13  /*-------------------------------------------------------------------------------
14    Defines -> Need to be changed depending on hardware
15   -------------------------------------------------------------------------------*/
16
17  #define ADC12_VBAT_PxSEL    P6SEL
18  #define ADC12_VBAT_PxDIR    P6DIR
19  #define ADC12_VBAT_PIN      BIT0
20
21  /*-------------------------------------------------------------------------------
22    Public functions
23  -------------------------------------------------------------------------------*/
24
25  u16_t adc12_sample(void);
26
27  #endif /* ADC12_H_ */
```

Listing D.5: adg918.c

```
1   /*-------------------------------------------------------------------------------
2      Description: ADG918 RF-switch driver.
3      Date:        11/22/2012
4      Last Update: 11/22/2012
5      Author:      Phillip Durdaut
6   -------------------------------------------------------------------------------*/
7
8   #include "main.h"
9
10  /*-------------------------------------------------------------------------------
11    Public functions
12  -------------------------------------------------------------------------------*/
13
14  void adg918_init(void)
15  {
16      ADG918_CTRL_PxDIR |= ADG918_CTRL_PIN;
17  }
18
19  void adg918_wakeup(void)
20  {
```

```
21     ADG918_CTRL_PxOUT &= ~ADG918_CTRL_PIN;
22  }
23
24  void adg918_transceiver(void)
25  {
26     ADG918_CTRL_PxOUT |= ADG918_CTRL_PIN;
27  }
```

Listing D.6: adg918.h

```
1   /*---------------------------------------------------------------------------
2      Description: ADG918 RF-switch driver.
3      Date:        11/22/2012
4      Last Update: 11/22/2012
5      Author:      Phillip Durdaut
6   ---------------------------------------------------------------------------*/
7
8   #ifndef ADG918_H_
9   #define ADG918_H_
10
11  #include "main.h"
12
13  /*---------------------------------------------------------------------------
14    Defines
15  ---------------------------------------------------------------------------*/
16
17  #define ADG918_CTRL_PxDIR (P5DIR)
18  #define ADG918_CTRL_PxOUT (P5OUT)
19  #define ADG918_CTRL_PIN (BIT0)
20
21  /*---------------------------------------------------------------------------
22    Public functions
23  ---------------------------------------------------------------------------*/
24
25  void adg918_init(void);
26  void adg918_wakeup(void);
27  void adg918_transceiver(void);
28
29  #endif /* ADG918_H_ */
```

Listing D.7: as3930.c

```
1   /*---------------------------------------------------------------------------
2      Description: Driver for austriamicrosystems AS3930 Single Channel
3                   Low Frequency Wakeup Receiver.
4      Date:        10/02/2012
5      Last Update: 10/03/2012
6      Author:      Phillip Durdaut
7   ---------------------------------------------------------------------------*/
8
9   #include "main.h"
10
11  /*---------------------------------------------------------------------------
12    Defines
13   ---------------------------------------------------------------------------*/
14
15  /* Configuration Modes (bits 15-14) */
16
17  #define AS3930_WRITE       0x00
18  #define AS3930_READ        0x01
19  #define AS3930_COMMAND     0x03
20
21  /* Configuration Registers (bits 13-8) */
22
23  #define AS3930_R00         0x00
24  #define AS3930_R01         0x01
```

```
25 #define AS3930_R02          0x02
26 #define AS3930_R03          0x03
27 #define AS3930_R04          0x04
28 #define AS3930_R05          0x05
29 #define AS3930_R06          0x06
30 #define AS3930_R07          0x07
31 #define AS3930_R08          0x08
32 #define AS3930_R09          0x09
33 #define AS3930_R10          0x0a
34 #define AS3930_R11          0x0b
35 #define AS3930_R12          0x0c
36 #define AS3930_R13          0x0d
37
38 /* Commands (bits 7-0) */
39
40 #define AS3930_CLEAR_WAKE   0x00
41 #define AS3930_RESET_RSSI   0x01
42 #define AS3930_TRIM_OSC     0x02
43 #define AS3930_CLEAR_FALSE 0x03
44 #define AS3930_PRESET_DEFAULT 0x04
45
46 /*-------------------------------------------------------------------------------
47  Prototypes of the private functions
48 -------------------------------------------------------------------------------*/
49
50 void as3930_spi_setup(void);
51 void as3930_spi_write_register(u8_t address, u8_t value);
52 char as3930_spi_read_register(u8_t address);
53 void as3930_spi_command(u8_t command);
54
55 /*-------------------------------------------------------------------------------
56  Public functions
57 -------------------------------------------------------------------------------*/
58
59 void as3930_init(void)
60 {
61    as3930_spi_setup();
62
63    AS3930_WAKE_DIR_IN;
64    AS3930_WAKE_IRQ_RISING_EDGE;
65    AS3930_WAKE_IRQ_DISABLE;
66    AS3930_WAKE_CLEAR_IRQ;
67 }
68
69 void as3930_config_no_pattern(void)
70 {
71    as3930_spi_write_register(AS3930_R01, BIT5); // Data correlation disable, Crystal
       oscillator disable
72    //as3930_spi_write_register(AS3930_R07, BIT5); // Automatic time-out after 50 ms
73 }
74
75 void as3930_preset_default(void)
76 {
77    as3930_spi_command(AS3930_PRESET_DEFAULT);
78 }
79
80 void as3930_clear_wakeup(void)
81 {
82    as3930_spi_command(AS3930_CLEAR_WAKE);
83 }
84
85 u8_t as3930_get_rssi(void)
86 {
87    return (as3930_spi_read_register(AS3930_R10) & 0x1F);
88 }
89
90 /*-------------------------------------------------------------------------------
```

```
91   Private functions
92  ------------------------------------------------------------------------------*/
93
94  void as3930_spi_setup(void)
95  {
96     AS3930_CS_PxDIR |= AS3930_CS_PIN; // CS is output
97     AS3930_CS_PxOUT &= ~AS3930_CS_PIN; // Chip disable
98
99     UCA0CTL1 |= UCSWRST;       // Hold state machine in reset
100
101    UCA0CTL0 &= ~UCCKPH;          // Data is changed on rising edge
102    UCA0CTL0 &= ~UCCKPL;          // Inactive clock is low
103    UCA0CTL0 |= (UCMST | UCMSB | UCSYNC); // MSB first, Master mode, Synchronous mode
104    UCA0CTL0 &= ~(UCMODE1 | UCMODE0); // 3-pin SPI
105    UCA0MCTL = 0;                 // No modulation
106
107    // SMCLK / 2
108    UCA0CTL1 |= (UCSSEL1 | UCSSEL0);
109    UCA0BR0 = 2;
110    UCA0BR1 = 0;
111
112    // SPI functionality for pins
113    AS3930_SPI_PxSEL |= (AS3930_SPI_MOSI_PIN | AS3930_SPI_MISO_PIN | AS3930_SPI_CLK_PIN);
114    AS3930_SPI_PxDIR |= (AS3930_SPI_MOSI_PIN | AS3930_SPI_CLK_PIN); // MOSI and CLK are outputs
115    AS3930_SPI_PxDIR &= ~AS3930_SPI_MISO_PIN; // MISO is input
116
117    UCA0CTL1 &= ~UCSWRST; // Initialize USART state machine
118 }
119
120 void as3930_spi_write_register(u8_t address, u8_t value)
121 {
122    AS3930_CS_PxOUT |= AS3930_CS_PIN; // Chip enable
123    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
124    UCA0TXBUF = address | (AS3930_WRITE << 6); // Send configuration mode and register
125    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
126    UCA0TXBUF = value;               // Send value
127    while (UCA0STAT & UCBUSY);       // Wait for TX complete
128    AS3930_CS_PxOUT &= ~AS3930_CS_PIN; // Chip disable
129 }
130
131 char as3930_spi_read_register(u8_t address)
132 {
133    u8_t value;
134
135    AS3930_CS_PxOUT |= AS3930_CS_PIN; // Chip enable
136    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
137    UCA0TXBUF = address | (AS3930_READ << 6); // Send configuration mode and register
138    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
139    UCA0TXBUF = 0;                   // Dummy write so we can read data
140    while (UCA0STAT & UCBUSY);       // Wait for TX complete
141    value = UCA0RXBUF;               // Read data
142    AS3930_CS_PxOUT &= ~AS3930_CS_PIN; // Chip disable
143
144    return value;
145 }
146
147 void as3930_spi_command(u8_t command)
148 {
149    AS3930_CS_PxOUT |= AS3930_CS_PIN;   // Chip enable
150    while (UCA0STAT & UCBUSY);          // Wait for TX to finish
151    UCA0TXBUF = command | (AS3930_COMMAND << 6); // Send configuration mode and register
152    while (UCA0STAT & UCBUSY);          // Wait for TX to finish
153    AS3930_CS_PxOUT &= ~AS3930_CS_PIN;  // Chip disable
154 }
```

Listing D.8: as3930.h

```
/*-------------------------------------------------------------------------------
   Description: Driver for austriamicrosystems AS3930 Single Channel
                Low Frequency Wakeup Receiver.
   Date:        10/09/2012
   Last Update: 10/09/2012
   Author:      Phillip Durdaut
-------------------------------------------------------------------------------*/

#ifndef AS3930_H_
#define AS3930_H_

#include "main.h"

/*-------------------------------------------------------------------------------
 Defines -> Need to be changed depending on hardware
 -------------------------------------------------------------------------------*/

#define AS3930_CS_PxDIR    P4DIR
#define AS3930_CS_PxOUT    P4OUT
#define AS3930_CS_PIN      BIT0

#define AS3930_SPI_PxSEL   P3SEL
#define AS3930_SPI_PxDIR   P3DIR
#define AS3930_SPI_PxIN    P3IN
#define AS3930_SPI_MOSI_PIN BIT4
#define AS3930_SPI_MISO_PIN BIT5
#define AS3930_SPI_CLK_PIN BIT0

#define AS3930_WAKE_PxDIR  P1DIR
#define AS3930_WAKE_PxIES  P1IES
#define AS3930_WAKE_PxIE   P1IE
#define AS3930_WAKE_PxIFG  P1IFG
#define AS3930_WAKE_PIN    BIT4

/*-------------------------------------------------------------------------------
 Macros
-------------------------------------------------------------------------------*/

#define AS3930_WAKE_DIR_IN    (AS3930_WAKE_PxDIR &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_RISING_EDGE (AS3930_WAKE_PxIES &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_FALLING_EDGE (AS3930_WAKE_PxIES |= AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_ENABLE (AS3930_WAKE_PxIE |= AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_DISABLE (AS3930_WAKE_PxIE &= ~AS3930_WAKE_PIN)
#define AS3930_WAKE_IRQ_PENDING ((AS3930_WAKE_PxIFG & AS3930_WAKE_PIN) == AS3930_WAKE_PIN)
#define AS3930_WAKE_CLEAR_IRQ (AS3930_WAKE_PxIFG &= ~(AS3930_WAKE_PIN))

/*-------------------------------------------------------------------------------
 Public functions
-------------------------------------------------------------------------------*/

void as3930_init(void);
void as3930_config_no_pattern(void);
void as3930_preset_default(void);
void as3930_clear_wakeup(void);
u8_t as3930_get_rssi(void);

#endif /* AS3930_H_ */
```

Listing D.9: balancing.c

```
/*-------------------------------------------------------------------------------
   Description: Balancing unit driver.
   Date:        11/22/2012
   Last Update: 11/22/2012
   Author:      Phillip Durdaut
```

```
6   ------------------------------------------------------------------------------*/
7
8   #include "main.h"
9
10  /*------------------------------------------------------------------------------
11   Public functions
12  ------------------------------------------------------------------------------*/
13
14  void balancing_init(void)
15  {
16     BALANCING_BALANCE_PxDIR |= BALANCING_BALANCE_PIN;
17  }
18
19  void balancing_on(void)
20  {
21     BALANCING_BALANCE_PxOUT |= BALANCING_BALANCE_PIN;
22  }
23
24  void balancing_off(void)
25  {
26     BALANCING_BALANCE_PxOUT &= ~BALANCING_BALANCE_PIN;
27  }
```

Listing D.10: balancing.h

```
1   /*------------------------------------------------------------------------------
2      Description: Balancing unit driver.
3      Date:        11/22/2012
4      Last Update: 11/22/2012
5      Author:      Phillip Durdaut
6   ------------------------------------------------------------------------------*/
7
8   #ifndef BALANCING_H_
9   #define BALANCING_H_
10
11  #include "main.h"
12
13  /*------------------------------------------------------------------------------
14   Defines
15  ------------------------------------------------------------------------------*/
16
17  #define BALANCING_BALANCE_PxDIR (P6DIR)
18  #define BALANCING_BALANCE_PxOUT (P6OUT)
19  #define BALANCING_BALANCE_PIN (BIT5)
20
21  /*------------------------------------------------------------------------------
22   Public functions
23  ------------------------------------------------------------------------------*/
24
25  void balancing_init(void);
26  void balancing_on(void);
27  void balancing_off(void);
28
29  #endif /* BALANCING_H_ */
```

Listing D.11: cc1101.c

```
1   /*------------------------------------------------------------------------------
2      Description: Driver for Texas Instruments CC1101 Low-Power Sub-1 GHz
3                   RF Transceiver..
4      Date:        10/02/2012
5      Last Update: 10/03/2012
6      Author:      Phillip Durdaut
7   ------------------------------------------------------------------------------*/
8
9   #include "main.h"
```

```
10
11  /*---------------------------------------------------------------------------
12   Variables
13   ---------------------------------------------------------------------------*/
14
15  /* Output power table (433 MHz): -30, 10 (dBm) */
16  u8_t cc1101_patable[] = { 0x12, 0xc0 };
17  u8_t cc1101_patablelen = 2;
18
19  /*---------------------------------------------------------------------------
20   Defines
21   ---------------------------------------------------------------------------*/
22
23  /* Command Strobes (Table 42 in datasheet) */
24
25  #define CC1101_CS_SRES     0x30
26  #define CC1101_CS_SFSTXON  0x31
27  #define CC1101_CS_SXOFF    0x32
28  #define CC1101_CS_SCAL     0x33
29  #define CC1101_CS_SRX      0x34
30  #define CC1101_CS_STX      0x35
31  #define CC1101_CS_SIDLE    0x36
32  #define CC1101_CS_SAFC     0x37
33  #define CC1101_CS_SWOR     0x38
34  #define CC1101_CS_SPWD     0x39
35  #define CC1101_CS_SFRX     0x3A
36  #define CC1101_CS_SFTX     0x3B
37  #define CC1101_CS_SWORRST  0x3C
38  #define CC1101_CS_SNOP     0x3D
39
40  /* Configuration Registers */
41
42  #define CC1101_CR_IOCFG2   0x00   /* GDO2 output pin configuration */
43  #define CC1101_CR_IOCFG1   0x01   /* GDO1 output pin configuration */
44  #define CC1101_CR_IOCFG0   0x02   /* GDO0 output pin configuration */
45  #define CC1101_CR_FIFOTHR  0x03   /* RX FIFO and TX FIFO thresholds */
46  #define CC1101_CR_SYNC1    0x04   /* Sync word, high byte */
47  #define CC1101_CR_SYNC0    0x05   /* Sync word, low byte */
48  #define CC1101_CR_PKTLEN   0x06   /* Packet length */
49  #define CC1101_CR_PKTCTRL1 0x07   /* Packet automation control */
50  #define CC1101_CR_PKTCTRL0 0x08   /* Packet automation control */
51  #define CC1101_CR_ADDR     0x09   /* Device address */
52  #define CC1101_CR_CHANNR   0x0A   /* Channel number */
53  #define CC1101_CR_FSCTRL1  0x0B   /* Frequency synthesizer control */
54  #define CC1101_CR_FSCTRL0  0x0C   /* Frequency synthesizer control */
55  #define CC1101_CR_FREQ2    0x0D   /* Frequency control word, high byte */
56  #define CC1101_CR_FREQ1    0x0E   /* Frequency control word, middle byte */
57  #define CC1101_CR_FREQ0    0x0F   /* Frequency control word, low byte */
58  #define CC1101_CR_MDMCFG4  0x10   /* Modem configuration */
59  #define CC1101_CR_MDMCFG3  0x11   /* Modem configuration */
60  #define CC1101_CR_MDMCFG2  0x12   /* Modem configuration */
61  #define CC1101_CR_MDMCFG1  0x13   /* Modem configuration */
62  #define CC1101_CR_MDMCFG0  0x14   /* Modem configuration */
63  #define CC1101_CR_DEVIATN  0x15   /* Modem deviation setting */
64  #define CC1101_CR_MCSM2    0x16   /* Main Radio Cntrl State Machine config */
65  #define CC1101_CR_MCSM1    0x17   /* Main Radio Cntrl State Machine config */
66  #define CC1101_CR_MCSM0    0x18   /* Main Radio Cntrl State Machine config */
67  #define CC1101_CR_FOCCFG   0x19   /* Frequency Offset Compensation config */
68  #define CC1101_CR_BSCFG    0x1A   /* Bit Synchronization configuration */
69  #define CC1101_CR_AGCCTRL2 0x1B   /* AGC control */
70  #define CC1101_CR_AGCCTRL1 0x1C   /* AGC control */
71  #define CC1101_CR_AGCCTRL0 0x1D   /* AGC control */
72  #define CC1101_CR_WOREVT1  0x1E   /* High byte Event 0 timeout */
73  #define CC1101_CR_WOREVT0  0x1F   /* Low byte Event 0 timeout */
74  #define CC1101_CR_WORCTRL  0x20   /* Wake On Radio control */
75  #define CC1101_CR_FREND1   0x21   /* Front end RX configuration */
76  #define CC1101_CR_FREND0   0x22   /* Front end TX configuration */
```

```
77  #define CC1101_CR_FSCAL3   0x23   /* Frequency synthesizer calibration */
78  #define CC1101_CR_FSCAL2   0x24   /* Frequency synthesizer calibration */
79  #define CC1101_CR_FSCAL1   0x25   /* Frequency synthesizer calibration */
80  #define CC1101_CR_FSCAL0   0x26   /* Frequency synthesizer calibration */
81  #define CC1101_CR_RCCTRL1  0x27   /* RC oscillator configuration */
82  #define CC1101_CR_RCCTRL0  0x28   /* RC oscillator configuration */
83  #define CC1101_CR_FSTEST   0x29   /* Frequency synthesizer cal control */
84  #define CC1101_CR_PTEST    0x2A   /* Production test */
85  #define CC1101_CR_AGCTEST  0x2B   /* AGC test */
86  #define CC1101_CR_TEST2    0x2C   /* Various test settings */
87  #define CC1101_CR_TEST1    0x2D   /* Various test settings */
88  #define CC1101_CR_TEST0    0x2E   /* Various test settings */
89
90  /* Status Registers */
91
92  #define CC1101_SR_PARTNUM  0x30   /* Part number */
93  #define CC1101_SR_VERSION  0x31   /* Current version number */
94  #define CC1101_SR_FREQEST  0x32   /* Frequency offset estimate */
95  #define CC1101_SR_LQI      0x33   /* Demodulator estimate for link quality */
96  #define CC1101_SR_RSSI     0x34   /* Received signal strength indication */
97  #define CC1101_SR_MARCSTATE 0x35  /* Control state machine state */
98  #define CC1101_SR_WORTIME1 0x36   /* High byte of WOR timer */
99  #define CC1101_SR_WORTIME0 0x37   /* Low byte of WOR timer */
100 #define CC1101_SR_PKTSTATUS 0x38  /* Current GDOx status and packet status */
101 #define CC1101_SR_VCO_VC_DAC 0x39 /* Current setting from PLL cal module */
102 #define CC1101_SR_TXBYTES  0x3A   /* Underflow and # of bytes in TXFIFO */
103 #define CC1101_SR_RXBYTES  0x3B   /* Overflow and # of bytes in RXFIFO */
104 #define CC1101_SR_RCCTRL1_STATUS 0x3C /* Last RC oscilator calibration results */
105 #define CC1101_SR_RCCTRL0_STATUS 0x3D /* Last RC oscilator calibration results */
106
107 /* Single / Burst access */
108
109 #define CC1101_WRITE_BURST 0x40
110 #define CC1101_READ_SINGLE 0x80
111 #define CC1101_READ_BURST  0xC0
112
113 /* Memory locations */
114
115 #define CC1101_ML_PATABLE  0x3E
116 #define CC1101_ML_TXFIFO   0x3F
117 #define CC1101_ML_RXFIFO   0x3F
118
119 /*---------------------------------------------------------------------------
120  Prototypes of the private functions
121 ---------------------------------------------------------------------------*/
122
123 void cc1101_spi_setup(void);
124 void cc1101_spi_write_register(u8_t address, u8_t value);
125 void cc1101_spi_write_register_burst(u8_t address, u8_t * buffer, u8_t count);
126 char cc1101_spi_read_register(u8_t address);
127 void cc1101_spi_read_register_burst(u8_t address, u8_t *buffer, u8_t count);
128 u8_t cc1101_spi_read_status(u8_t address);
129 void cc1101_spi_command_strobe(u8_t strobe);
130
131 /*---------------------------------------------------------------------------
132  Public functions
133 ---------------------------------------------------------------------------*/
134
135 void cc1101_init(void)
136 {
137    cc1101_spi_setup();
138
139    CC1101_GDO2_DIR_IN;
140    CC1101_GDO2_IRQ_RISING_EDGE;
141    CC1101_GDO2_IRQ_DISABLE;
142    CC1101_GDO2_CLEAR_IRQ;
143 }
```

```
144
145 void cc1101_power_up_reset(void)
146 {
147    // Manual power-on reset
148    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
149    delay_100us(1);
150    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
151    delay_100us(1);
152    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
153    delay_100us(5);
154    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
155    delay_100us(5);
156
157    cc1101_reset();
158 }
159
160 void cc1101_reset(void)
161 {
162    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
163
164    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
165    UCA0TXBUF = CC1101_CS_SRES;      // Send reset strobe
166    while (UCA0STAT & UCBUSY);       // Wait for TX to finish
167
168    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
169
170    delay_ms(1);
171 }
172
173 void cc1101_config_no_packet(void)
174 {
175    // GDO0 at high impedance
176    cc1101_spi_write_register(CC1101_CR_IOCFG0, 0x2E);
177
178    // GDO2 at high impedance
179    cc1101_spi_write_register(CC1101_CR_IOCFG2, 0x2E);
180
181    // Asynchronous serial mode, Infinite packet length mode
182    cc1101_spi_write_register(CC1101_CR_PKTCTRL0, 0x32);
183
184    // Channel 0
185    cc1101_spi_write_register(CC1101_CR_CHANNR, 0x00);
186
187    // Carrier frequency
188    cc1101_spi_write_register(CC1101_CR_FSCTRL0, FREQUENCY_OFFSET);
189
190    // Carrier frequency: 433.999969 MHz
191    cc1101_spi_write_register(CC1101_CR_FREQ2, 0x10);
192    cc1101_spi_write_register(CC1101_CR_FREQ1, 0xB1);
193    cc1101_spi_write_register(CC1101_CR_FREQ0, 0x3B);
194
195    cc1101_spi_write_register(CC1101_CR_MDMCFG4, 0xFD);
196
197    // 250 kBaud, OOK
198    cc1101_spi_write_register(CC1101_CR_MDMCFG3, 0x3B);
199    cc1101_spi_write_register(CC1101_CR_MDMCFG2, 0x30);
200
201    cc1101_spi_write_register(CC1101_CR_MDMCFG1, 0x00);
202
203    cc1101_spi_write_register(CC1101_CR_DEVIATN, 0x15);
204    cc1101_spi_write_register(CC1101_CR_MCSM0, 0x18);
205
206    // FCL gain: 3000, Saturation point for the frequency offset compensation algorithm ->
           SmartRF Studio
207    cc1101_spi_write_register(CC1101_CR_FOCCFG, 0x16);
208
209    // 10 dBm output power
```

```
210     cc1101_spi_write_register_burst(CC1101_ML_PATABLE, cc1101_patable, cc1101_patablelen);
211     cc1101_spi_write_register(CC1101_CR_FREND0, 0x11);
212  }
213
214  void cc1101_config_packet(void)
215  {
216     // Rising edge on GDO0 when packet received and CRC check OK
217     // (Deasserted when first byte is read from RX FIFO)
218     cc1101_spi_write_register(CC1101_CR_IOCFG0, 0x07);
219
220     // Rising edge on GDO2 when sync word has been received
221     cc1101_spi_write_register(CC1101_CR_IOCFG2, 0x06);
222
223     // 4 SYNC bytes: 0x12 0x09 (repeated once)
224     cc1101_spi_write_register(CC1101_CR_SYNC1, 0x12);
225     cc1101_spi_write_register(CC1101_CR_SYNC0, 0x09);
226
227     // Flush RX packets when CRC is not OK, Address check and 0x00 broadcast
228     cc1101_spi_write_register(CC1101_CR_PKTCTRL1, 0x0A);
229
230     // No whitening, FIFO mode, CRC disable, Fixed packet length
231     cc1101_spi_write_register(CC1101_CR_PKTCTRL0, 0x00);
232
233     // Device Address
234     cc1101_spi_write_register(CC1101_CR_ADDR, ADDRESS_THIS_SENSOR);
235
236     // Channel 0
237     cc1101_spi_write_register(CC1101_CR_CHANNR, 0x00);
238
239     // IF frequency: 152.34375 kHz
240     cc1101_spi_write_register(CC1101_CR_FSCTRL1, 0x06);
241
242     // Carrier frequency
243     cc1101_spi_write_register(CC1101_CR_FSCTRL0, FREQUENCY_OFFSET);
244
245     // Carrier frequency: 433.999969 MHz
246     cc1101_spi_write_register(CC1101_CR_FREQ2, 0x10);
247     cc1101_spi_write_register(CC1101_CR_FREQ1, 0xB1);
248     cc1101_spi_write_register(CC1101_CR_FREQ0, 0x3B);
249
250     // 40 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
251     // No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
252     cc1101_spi_write_register(CC1101_CR_MDMCFG4, 0x8A);
253     cc1101_spi_write_register(CC1101_CR_MDMCFG3, 0x93);
254     cc1101_spi_write_register(CC1101_CR_MDMCFG2, 0x33);
255     cc1101_spi_write_register(CC1101_CR_MDMCFG1, 0x22);
256     cc1101_spi_write_register(CC1101_CR_MDMCFG0, 0x7A);
257
258     // Calibrate when going from IDLE to RX or TX, Crystal off when in SLEEP state
259     cc1101_spi_write_register(CC1101_CR_MCSM0, 0x10);
260
261     // FCL gain: 3000, Saturation point for the frequency offset compensation algorithm ->
            SmartRF Studio
262     cc1101_spi_write_register(CC1101_CR_FOCCFG, 0x16);
263
264     //cc1101_spi_write_register(CC1101_CR_FREND1, 0x00);
265     cc1101_spi_write_register_burst(CC1101_ML_PATABLE, cc1101_patable, cc1101_patablelen);
266     cc1101_spi_write_register(CC1101_CR_FREND0, 0x11); // 10 dBm output power
267  }
268
269  void cc1101_fill_tx_fifo(u8_t * buffer, u8_t length)
270  {
271     // Clear TX FIFO
272     cc1101_clear_tx_fifo();
273
274     // Transfer the bytes via SPI to the TX fifo of the transceiver
275     cc1101_spi_write_register_burst(CC1101_ML_TXFIFO, buffer, length);
```

```
276 }
277
278 void cc1101_read_rx_fifo(u8_t * buffer, u8_t length)
279 {
280    // Disable the receiver
281    cc1101_idle();
282
283    // Transfer the bytes via SPI from the RX fifo
284    cc1101_spi_read_register_burst(CC1101_ML_RXFIFO, buffer, length);
285 }
286
287 void cc1101_enable_crc(void)
288 {
289    u8_t current = cc1101_spi_read_register(CC1101_CR_PKTCTRL0);
290    cc1101_spi_write_register(CC1101_CR_PKTCTRL0, current | BIT2);
291 }
292
293 void cc1101_disable_crc(void)
294 {
295    u8_t current = cc1101_spi_read_register(CC1101_CR_PKTCTRL0);
296    cc1101_spi_write_register(CC1101_CR_PKTCTRL0, current & ~BIT2);
297 }
298
299 void cc1101_clear_tx_fifo(void)
300 {
301    cc1101_spi_command_strobe(CC1101_CS_SFTX);
302 }
303
304 void cc1101_clear_rx_fifo(void)
305 {
306    cc1101_spi_command_strobe(CC1101_CS_SFRX);
307 }
308
309 u8_t cc1101_get_rxbytes(void)
310 {
311    return (cc1101_spi_read_status(CC1101_SR_RXBYTES) & 0x7F);
312 }
313
314 void cc1101_sleep(void)
315 {
316    cc1101_spi_command_strobe(CC1101_CS_SPWD);
317 }
318
319 void cc1101_idle(void)
320 {
321    cc1101_spi_command_strobe(CC1101_CS_SIDLE);
322 }
323
324 void cc1101_tx_carrier(void)
325 {
326    cc1101_spi_command_strobe(CC1101_CS_STX);
327 }
328
329 void cc1101_tx(void)
330 {
331    // DATA packet length
332    cc1101_spi_write_register(CC1101_CR_PKTLEN, UPLINK_DATA_BYTES);
333
334    // Enable CRC calculation when transmitting data
335    cc1101_enable_crc();
336
337    // TX state
338    cc1101_spi_command_strobe(CC1101_CS_STX);
339
340    // Wait 10 ms for TX finished
341    delay_ms(10);
342 }
```

```
343
344 void cc1101_rx(void)
345 {
346    // DATA packet length
347    cc1101_spi_write_register(CC1101_CR_PKTLEN, DOWNLINK_DATA_BYTES);
348
349    // Enable CRC check when receiving data
350    cc1101_enable_crc();
351
352    // Clear RX FIFO
353    cc1101_clear_rx_fifo();
354
355    // RX state
356    cc1101_spi_command_strobe(CC1101_CS_SRX);
357 }
358
359 u8_t cc1101_get_partnum(void)
360 {
361    return cc1101_spi_read_status(CC1101_SR_PARTNUM);
362 }
363
364 u8_t cc1101_get_version(void)
365 {
366    return cc1101_spi_read_status(CC1101_SR_VERSION);
367 }
368
369 u8_t cc1101_get_marcstate(void)
370 {
371    return (cc1101_spi_read_status(CC1101_SR_MARCSTATE) & 0x1F);
372 }
373
374 /*-------------------------------------------------------------------------------
375  Private functions
376 -------------------------------------------------------------------------------*/
377
378 void cc1101_spi_setup(void)
379 {
380    CC1101_CSn_PxDIR |= CC1101_CSn_PIN; // nCS is output
381    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Unselect CC1101 chip
382
383    UCA0CTL1 |= UCSWRST;          // Hold state machine in reset
384
385    UCA0CTL0 |= UCCKPH;           // Data is sampled on rising edge
386    UCA0CTL0 &= ~UCCKPL;          // Inactive clock is low
387    UCA0CTL0 |= (UCMST | UCMSB | UCSYNC); // MSB first, Master mode, Synchronous mode
388    UCA0CTL0 &= ~(UCMODE1 | UCMODE0); // 3-pin SPI
389    UCA0MCTL = 0;                 // No modulation
390
391    // SMCLK / 2
392    UCA0CTL1 |= (UCSSEL1 | UCSSEL0);
393    UCA0BR0 = 5;
394    UCA0BR1 = 0;
395
396    // SPI functionality for pins
397    CC1101_SPI_PxSEL |= (CC1101_SPI_SI_PIN | CC1101_SPI_SO_GDO1_PIN | CC1101_SPI_SCLK_PIN);
398    CC1101_SPI_PxDIR |= (CC1101_SPI_SI_PIN | CC1101_SPI_SCLK_PIN); // MOSI and CLK are outputs
399    CC1101_SPI_PxDIR &= ~CC1101_SPI_SO_GDO1_PIN; // MISO is input
400
401    UCA0CTL1 &= ~UCSWRST; // Initialize USART state machine
402 }
403
404 void cc1101_spi_write_register(u8_t address, u8_t value)
405 {
406    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
407    while (UCA0STAT & UCBUSY);      // Wait for TX to finish
408    UCA0TXBUF = address;            // Send address
409    while (UCA0STAT & UCBUSY);      // Wait for TX to finish
```

```
410     UCA0TXBUF = value;              // Send value
411     while(UCA0STAT & UCBUSY);       // Wait for TX complete
412     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
413  }
414
415  void cc1101_spi_write_register_burst(u8_t address, u8_t * buffer, u8_t count)
416  {
417     u8_t i;
418
419     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
420     while (UCA0STAT & UCBUSY);      // Wait for TX to finish
421     UCA0TXBUF = address | CC1101_WRITE_BURST; // Send address
422
423     for (i = 0; i < count; i++) {
424        while (UCA0STAT & UCBUSY);   // Wait for TX to finish
425        UCA0TXBUF = buffer[i];       // Send data
426     }
427
428     while(UCA0STAT & UCBUSY);       // Wait for TX complete
429     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
430  }
431
432  char cc1101_spi_read_register(u8_t address)
433  {
434     u8_t x;
435
436     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
437     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
438     UCA0TXBUF = (address | CC1101_READ_SINGLE); // Send address
439     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
440     UCA0TXBUF = 0;                    // Dummy write so we can read data
441     while(UCA0STAT & UCBUSY);         // Wait for TX complete
442     x = UCA0RXBUF;                    // Read data
443     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
444
445     return x;
446  }
447
448  void cc1101_spi_read_register_burst(u8_t address, u8_t * buffer, u8_t count)
449  {
450     u16_t i;
451
452     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
453     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
454     UCA0TXBUF = (address | CC1101_READ_BURST); // Send address
455     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
456
457     for (i = 0; i < count; i++) {
458        UCA0TXBUF = 0;                // Initiate next data RX, meanwhile
459        while (UCA0STAT & UCBUSY);    // Wait for TX to finish
460        buffer[i] = UCA0RXBUF;        // Store data from last data RX
461     }
462
463     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
464  }
465
466  u8_t cc1101_spi_read_status(u8_t address)
467  {
468     u8_t status;
469
470     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
471     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
472     UCA0TXBUF = (address | CC1101_READ_BURST); // Send address
473     while (UCA0STAT & UCBUSY);        // Wait for TX to finish
474     UCA0TXBUF = 0;                    // Dummy write so we can read data
475     while (UCA0STAT & UCBUSY);        // Wait for TX complete
476     status = UCA0RXBUF;               // Read data
```

```
477     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
478
479     return status;
480 }
481
482 void cc1101_spi_command_strobe(u8_t strobe)
483 {
484     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
485     while (UCA0STAT & UCBUSY);       // Wait for TX to finish
486     UCA0TXBUF = strobe;              // Send strobe
487     while (UCA0STAT & UCBUSY);       // Wait for TX complete
488     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
489 }
```

Listing D.12: cc1101.h

```
1   /*-------------------------------------------------------------------------------------
2     Description: Driver for Texas Instruments CC1101 Low-Power Sub-1 GHz
3                  RF Transceiver.
4     Date:        10/02/2012
5     Last Update: 10/03/2012
6     Author:      Phillip Durdaut
7   -------------------------------------------------------------------------------------*/
8
9   #ifndef CC1101_H_
10  #define CC1101_H_
11
12  #include "main.h"
13
14  /*-------------------------------------------------------------------------------------
15    Defines -> Need to be changed depending on hardware
16   -------------------------------------------------------------------------------------*/
17
18  #define CC1101_CSn_PxDIR   P1DIR
19  #define CC1101_CSn_PxOUT   P1OUT
20  #define CC1101_CSn_PIN     BIT5
21
22  #define CC1101_SPI_PxSEL   P3SEL
23  #define CC1101_SPI_PxDIR   P3DIR
24  #define CC1101_SPI_PxIN    P3IN
25  #define CC1101_SPI_SI_PIN  BIT4
26  #define CC1101_SPI_SO_GDO1_PIN BIT5
27  #define CC1101_SPI_SCLK_PIN BIT0
28
29  #define CC1101_GDO0_PxDIR  P1DIR
30  #define CC1101_GDO0_PxIN   P1IN
31  #define CC1101_GDO0_PIN    BIT6
32  #define CC1101_GDO0_POS    6
33
34  #define CC1101_GDO2_PxDIR  P1DIR
35  #define CC1101_GDO2_PxIN   P1IN
36  #define CC1101_GDO2_PxIES  P1IES
37  #define CC1101_GDO2_PxIE   P1IE
38  #define CC1101_GDO2_PxIFG  P1IFG
39  #define CC1101_GDO2_PIN    BIT7
40  #define CC1101_GDO2_POS    7
41
42  /*-------------------------------------------------------------------------------------
43    Macros
44  -------------------------------------------------------------------------------------*/
45
46  // CRC check done
47  #define CC1101_GDO0_DIR_IN    (CC1101_GDO0_PxDIR &= ~CC1101_GDO0_PIN)
48  #define CC1101_GDO0_IN        ((CC1101_GDO0_PxIN & CC1101_GDO0_PIN) >> CC1101_GDO0_POS)
49
50  // Sync word detected
```

```
51 #define CC1101_GDO2_DIR_IN    (CC1101_GDO2_PxDIR &= ~CC1101_GDO2_PIN)
52 #define CC1101_GDO2_IN        ((CC1101_GDO2_PxIN & CC1101_GDO2_PIN) >> CC1101_GDO2_POS)
53 #define CC1101_GDO2_IRQ_RISING_EDGE (CC1101_GDO2_PxIES &= ~CC1101_GDO2_PIN)
54 #define CC1101_GDO2_IRQ_ENABLE (CC1101_GDO2_PxIE |= CC1101_GDO2_PIN)
55 #define CC1101_GDO2_IRQ_DISABLE (CC1101_GDO2_PxIE &= ~CC1101_GDO2_PIN)
56 #define CC1101_GDO2_IRQ_PENDING ((CC1101_GDO2_PxIFG & CC1101_GDO2_PIN) == CC1101_GDO2_PIN)
57 #define CC1101_GDO2_CLEAR_IRQ (CC1101_GDO2_PxIFG &= ~(CC1101_GDO2_PIN))
58
59 /*---------------------------------------------------------------------------
60  Public functions
61 ---------------------------------------------------------------------------*/
62
63 void cc1101_init(void);
64 void cc1101_power_up_reset(void);
65 void cc1101_reset(void);
66 void cc1101_config_no_packet(void);
67 void cc1101_config_packet(void);
68 void cc1101_fill_tx_fifo(u8_t * buffer, u8_t length);
69 void cc1101_read_rx_fifo(u8_t * buffer, u8_t length);
70 void cc1101_enable_crc(void);
71 void cc1101_disable_crc(void);
72 void cc1101_clear_tx_fifo(void);
73 void cc1101_clear_rx_fifo(void);
74 u8_t cc1101_get_rxbytes(void);
75 void cc1101_sleep(void);
76 void cc1101_idle(void);
77 void cc1101_tx_carrier(void);
78 void cc1101_tx(void);
79 void cc1101_rx(void);
80 u8_t cc1101_get_partnum(void);
81 u8_t cc1101_get_version(void);
82 u8_t cc1101_get_marcstate(void);
83
84 #endif /* CC1101_H_ */
```

Listing D.13: delay.c

```
1  /*---------------------------------------------------------------------------
2     Description: Delay functions.
3     Date:        10/25/2012
4     Last Update: 10/25/2012
5     Author:      Phillip Durdaut
6  ---------------------------------------------------------------------------*/
7
8  #include "main.h"
9
10 /*---------------------------------------------------------------------------
11  Prototypes of the private functions
12 ---------------------------------------------------------------------------*/
13
14 void delay(u16_t i);
15
16 /*---------------------------------------------------------------------------
17  Public functions
18 ---------------------------------------------------------------------------*/
19
20 void delay_ms(u16_t delay_ms)
21 {
22    while (delay_ms > 0) {
23       delay(DELAY_CYCLES_PER_MS);
24       delay_ms--;
25    }
26 }
27
28 void delay_100us(u16_t delay_100us)
29 {
```

```
30      while (delay_100us > 0) {
31          delay(DELAY_CYCLES_PER_100US);
32          delay_100us--;
33      }
34  }
35
36  /*-------------------------------------------------------------------------------
37   Private functions
38  -------------------------------------------------------------------------------*/
39
40  void delay(u16_t i)
41  {
42      while (i > 0) {
43          _NOP();
44          i--;
45      }
46  }
```

Listing D.14: delay.h

```
1   /*-------------------------------------------------------------------------------
2      Description: Delay functions.
3      Date:        10/25/2012
4      Last Update: 10/25/2012
5      Author:      Phillip Durdaut
6   -------------------------------------------------------------------------------*/
7
8   #ifndef DELAY_H_
9   #define DELAY_H_
10
11  #include "main.h"
12
13  /*-------------------------------------------------------------------------------
14   Defines
15  -------------------------------------------------------------------------------*/
16
17  #define DELAY_CYCLES_PER_MS 106
18  #define DELAY_CYCLES_PER_100US 10
19
20  /*-------------------------------------------------------------------------------
21   Public functions
22  -------------------------------------------------------------------------------*/
23
24  void delay_ms(u16_t delay_ms);
25  void delay_100us(u16_t delay_100us);
26
27  #endif /* DELAY_H_ */
```

Listing D.15: led.c

```
1   /*-------------------------------------------------------------------------------
2      Description: LED driver.
3      Date:        11/22/2012
4      Last Update: 11/22/2012
5      Author:      Phillip Durdaut
6   -------------------------------------------------------------------------------*/
7
8   #include "main.h"
9
10  /*-------------------------------------------------------------------------------
11   Public functions
12  -------------------------------------------------------------------------------*/
13
14  void led_init(void)
15  {
16      LED_1_PxDIR |= LED_1_PIN;
```

```
17      LED_2_PxDIR |= LED_2_PIN;
18      LED_3_PxDIR |= LED_3_PIN;
19
20      led_off(LED_ALL);
21  }
22
23  void led_on(LED_t led)
24  {
25  #ifdef ENABLE_LEDS
26      switch(led) {
27          case LED_AWAKE: LED_1_PxOUT |= LED_1_PIN; break;
28          case LED_TX: LED_2_PxOUT |= LED_2_PIN; break;
29          case LED_RX: LED_3_PxOUT |= LED_3_PIN; break;
30          case LED_ALL: led_on(LED_AWAKE);
31                        led_on(LED_TX);
32                        led_on(LED_RX);
33                        break;
34          default: break;
35      }
36  #endif /* ENABLE_LEDS */
37  }
38
39  void led_off(LED_t led)
40  {
41      switch(led) {
42          case LED_AWAKE: LED_1_PxOUT &= ~LED_1_PIN; break;
43          case LED_TX: LED_2_PxOUT &= ~LED_2_PIN; break;
44          case LED_RX: LED_3_PxOUT &= ~LED_3_PIN; break;
45          case LED_ALL: led_off(LED_AWAKE);
46                        led_off(LED_TX);
47                        led_off(LED_RX);
48                        break;
49          default: break;
50      }
51  }
52
53  void led_toggle(LED_t led)
54  {
55  #ifdef ENABLE_LEDS
56      switch(led) {
57          case LED_AWAKE: LED_1_PxOUT ^= LED_1_PIN; break;
58          case LED_TX: LED_2_PxOUT ^= LED_2_PIN; break;
59          case LED_RX: LED_3_PxOUT ^= LED_3_PIN; break;
60          case LED_ALL: led_toggle(LED_AWAKE);
61                        led_toggle(LED_TX);
62                        led_toggle(LED_RX);
63                        break;
64          default: break;
65      }
66  #endif /* ENABLE_LEDS */
67  }
```

Listing D.16: led.h

```
1   /*-------------------------------------------------------------------------------
2      Description: LED driver.
3      Date:       11/22/2012
4      Last Update: 11/22/2012
5      Author:     Phillip Durdaut
6   -------------------------------------------------------------------------------*/
7
8   #ifndef LED_H_
9   #define LED_H_
10
11  #include "main.h"
12
```

```
13 /*---------------------------------------------------------------------------
14   Defines
15 ---------------------------------------------------------------------------*/
16
17 #define LED_1_PxDIR (P5DIR)
18 #define LED_1_PxOUT (P5OUT)
19 #define LED_1_PIN (BIT4)
20
21 #define LED_2_PxDIR (P5DIR)
22 #define LED_2_PxOUT (P5OUT)
23 #define LED_2_PIN (BIT3)
24
25 #define LED_3_PxDIR (P5DIR)
26 #define LED_3_PxOUT (P5OUT)
27 #define LED_3_PIN (BIT2)
28
29 /*---------------------------------------------------------------------------
30   Types
31 ---------------------------------------------------------------------------*/
32
33 typedef enum
34 {
35   LED_AWAKE = 0,
36   LED_TX,
37   LED_RX,
38   LED_ALL
39 } LED_t;
40
41 /*---------------------------------------------------------------------------
42   Public functions
43 ---------------------------------------------------------------------------*/
44
45 void led_init(void);
46 void led_on(LED_t led);
47 void led_off(LED_t led);
48 void led_toggle(LED_t led);
49
50 #endif /* LED_H_ */
```

Listing D.17: types.h

```
1  /*---------------------------------------------------------------------------
2     Description: MSP430 (mspgcc) data types.
3     Date:        12/02/2012
4     Last Update: 12/02/2012
5     Author:      Phillip Durdaut
6  ---------------------------------------------------------------------------*/
7
8  #ifndef TYPES_H_
9  #define TYPES_H_
10
11 /*---------------------------------------------------------------------------
12   Types
13 ---------------------------------------------------------------------------*/
14
15 typedef unsigned char u8_t;
16 typedef signed char  s8_t;
17 typedef unsigned int u16_t;
18 typedef signed int   s16_t;
19 typedef unsigned long u32_t;
20 typedef signed long  s32_t;
21 typedef unsigned long long u64_t;
22 typedef signed long long s64_t;
23
24 #endif /* TYPES_H_ */
```

D.1.2 Basisstation

Listing D.18: main.c

```
/*-------------------------------------------------------------------------------
   Description: Main program for the base station used during
                trial measurements.
   Hardware:    Olimex MSP430-169STK
                BATSEN 434 MHz Transceiver-Board üfr die
                Basisstation v0.1 - Transceiver: CC1101
   Date:        10/02/2012
   Last Update: 10/03/2012
   Author:      Phillip Durdaut
-------------------------------------------------------------------------------*/

#include "main.h"

// P1.0 <- GDO0 <-| soldered bridge
// P1.1 <- GDO2  |
// P1.2 -> GDO0 (TX) -|
// P1.5 <- Button 1
// P2.5 <- RTC
// P3.6 -> LED1
// P3.7 -> LED2
// P4.x -> LCD

/*-------------------------------------------------------------------------------
 Global variables
-------------------------------------------------------------------------------*/

const u8_t g_sensor_address[] = { ADDRESS_SENSOR_1, ADDRESS_SENSOR_2,
                          ADDRESS_SENSOR_3, ADDRESS_SENSOR_4 };

volatile state_t g_state = S_INIT;
volatile u8_t g_rxbuf[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
volatile u8_t g_data_received_flag = 0;
volatile u8_t g_awake_sensors[NUMBER_OF_SENSORS];
volatile u16_t g_samplebuf[NUMBER_OF_SENSORS];
volatile u8_t g_current_sensor = 0;
volatile u16_t g_sample_cnt = 0;

/*-------------------------------------------------------------------------------
 Prototypes of the private functions
-------------------------------------------------------------------------------*/

void rx(void);
void tx_wakeup(void);
void tx_single_packet(u8_t * txbuf);
void osc_configuration(void);

/*-------------------------------------------------------------------------------
 Main function
-------------------------------------------------------------------------------*/

//*************************************************************
int main (void)
//*************************************************************
{
   //*************************************************************
   // OSC configuration & startup (8 MHz)
   //*************************************************************
   osc_configuration();

   //*************************************************************
   // Button 1 configuration
   //*************************************************************
   BUTTON1_DIR_IN;
   BUTTON1_FALLING_EDGE;
   BUTTON1_CLEAR_IRQ;
```

```
66      BUTTON1_IRQ_ENABLE;
67
68      //*************************************************************
69      // RTC interrupt configuration
70      //*************************************************************
71      RTC_DIR_IN;
72      RTC_FALLING_EDGE;
73      RTC_CLEAR_IRQ;
74      RTC_IRQ_DISABLE;
75
76      //*************************************************************
77      // Port configuration
78      //*************************************************************
79      LEDx_DIR_OUT;
80      LED1_OFF;
81      LED2_OFF;
82      LCD_DIR_OUT;
83      LCD_OUT_0x00;
84
85      // TX pin should be input at first because the transceiver sends
86      // a clock signal on this pin on startup
87      TXPIN_MAN;
88      TXPIN_DIR_IN;
89
90      //*************************************************************
91      // LCD configuration
92      //*************************************************************
93      LCD_init();
94      LCD_send_cmd(LCD_CLR);
95      LCD_send_cmd(LCD_LINE1);
96      LCD_send_text("Press Button 1");
97      LCD_send_cmd(LCD_LINE2);
98      LCD_send_text("to start wakeup");
99
100     //*************************************************************
101     // CC1101 configuration
102     //*************************************************************
103     cc1101_init();
104     cc1101_power_up_reset();
105     cc1101_config_packet();
106     cc1101_rx();
107
108     CC1101_GDO2_CLEAR_IRQ;
109     CC1101_GDO2_IRQ_ENABLE;
110
111     //*************************************************************
112     // Timer A configuration
113     //*************************************************************
114     TIMERA_RESET;
115     TIMERA_OFF;
116
117     // Global interrupt enable
118     _EINT();
119
120     //*************************************************************
121     // Endless loop
122     //*************************************************************
123     while(1) {
124
125        switch(g_state) {
126
127           case S_TX_IS_AWAKE: {
128
129              g_current_sensor++;
130
131              // Send IS_AWAKE command to sensor "current_sensor"
132              // 1 byte target address + 1 byte command + 4 bytes zero padding
```

```
133             u8_t txisawakebuf[] = { g_sensor_address[g_current_sensor-1],
                    COMMAND_DOWNLINK_IS_AWAKE, 0, 0, 0, 0 };
134             tx_single_packet(txisawakebuf);
135
136             // Change to next state for waiting for the sensors response
137             g_state = S_RX_IS_AWAKE;
138          }
139          break;
140
141          case S_RX_IS_AWAKE: {
142
143             // Change to RX
144             g_data_received_flag = 0; // Reset flag
145             rx();
146
147             // Wait for answer of sensor "current_sensor"
148             delay_ms(100);
149
150             // Data received?
151             if (g_data_received_flag) {
152
153                // Answer from the correct sensor?
154                if (g_rxbuf[1] == g_sensor_address[g_current_sensor-1])
155                   g_awake_sensors[g_current_sensor-1] = 1;
156             }
157
158             // Count number of awake sensors
159             u8_t i;
160             u8_t awake_sensors_cnt = 0;
161             for (i = 0; i < NUMBER_OF_SENSORS; i++) {
162                if (g_awake_sensors[i])
163                   awake_sensors_cnt++;
164             }
165
166             if (NUMBER_OF_SENSORS == 4) {
167
168                unsigned char line1[16];
169                unsigned char line2[16];
170                sprintf(line1, "Awake sensors: %d", awake_sensors_cnt);
171                sprintf(line2, "%d %d %d %d ",
172                         (g_awake_sensors[0] == 1),
173                         (g_awake_sensors[1] == 1),
174                         (g_awake_sensors[2] == 1),
175                         (g_awake_sensors[3] == 1));
176
177                // Display number of awake sensors
178                LCD_send_cmd(LCD_CLR);
179                LCD_send_cmd(LCD_LINE1);
180                LCD_send_text(line1);
181                LCD_send_cmd(LCD_LINE2);
182                LCD_send_text(line2);
183             }
184             else {
185
186                // Display number of awake sensors
187                LCD_send_cmd(LCD_CLR);
188                LCD_send_cmd(LCD_LINE1);
189                LCD_send_text("Awake sensors:");
190                LCD_send_cmd(LCD_LINE2);
191                LCD_send_unsigned_int(awake_sensors_cnt);
192             }
193
194             // Ask the next sensor for its attendance
195             if (g_current_sensor < NUMBER_OF_SENSORS)
196                g_state = S_TX_IS_AWAKE;
197
198             // Change to the next state where the first
```

```
199             // SAMPLE command will be sent
200             if (g_current_sensor == NUMBER_OF_SENSORS) {
201                g_current_sensor = 0;
202                g_state = S_BROADCAST_SAMPLE;
203
204                // Wait 2 seconds before refreshing the LCD
205                // delay_ms(2000);
206
207                LCD_send_cmd(LCD_CLR);
208                LCD_send_cmd(LCD_LINE1);
209                LCD_send_text("Sampling will be");
210                LCD_send_cmd(LCD_LINE2);
211                LCD_send_text("started now ... ");
212             }
213          }
214          break;
215
216          case S_BROADCAST_SAMPLE: {
217
218             // Number of samples to take reached?
219             if (g_sample_cnt == NUMBER_OF_SAMPLES_PER_SENSOR) {
220
221                // Send the sensors back to sleep state
222                g_state = S_BROADCAST_SLEEP;
223             }
224             else {
225
226                // Enable the RTC interrupt that occurs each second and is
227                // used to send the broadcast SAMPLE command to all sensors
228                RTC_CLEAR_IRQ;
229                RTC_IRQ_ENABLE;
230             }
231          }
232          break;
233
234          case S_TX_SEND_SAMPLE: {
235
236             if (g_current_sensor == 0)
237                delay_ms(20);
238
239             g_current_sensor++;
240
241             // Send SEND_SAMPLE command to sensor "current_sensor"
242             // 1 byte target address + 1 byte command + 4 bytes zero padding
243             u8_t txsendsamplebuf[] = { g_sensor_address[g_current_sensor-1],
                    COMMAND_DOWNLINK_SEND_SAMPLE, 0, 0, 0, 0 };
244             tx_single_packet(txsendsamplebuf);
245
246             // Change to next state for waiting for the sensors sample
247             g_state = S_RX_SAMPLE;
248          }
249          break;
250
251          case S_RX_SAMPLE: {
252
253             // Change to RX
254             g_data_received_flag = 0; // Reset flag
255             rx();
256
257             // Wait for answer of sensor "current_sensor"
258             delay_ms(100);
259
260             // Sample is zero in case there was no answer
261             g_samplebuf[g_current_sensor-1] = 0;
262
263             // Data received?
264             if (g_data_received_flag) {
```

```
265
266                // Answer from the correct sensor?
267                if (g_rxbuf[1] == g_sensor_address[g_current_sensor-1]) {
268
269                   u16_t sample_msb = (((u16_t)(g_rxbuf[2] & 0x0F)) << 8) & 0x0F00;
270                   u16_t sample_lsb = (((u16_t)(g_rxbuf[3] & 0xFF)) << 0) & 0x00FF;
271
272                   // Save sample
273                   g_samplebuf[g_current_sensor-1] = sample_msb | sample_lsb;
274                }
275             }
276
277             // Ask the next sensor for its sample
278             if (g_current_sensor < NUMBER_OF_SENSORS)
279                g_state = S_TX_SEND_SAMPLE;
280
281             // Got all samples, send them to the PC and
282             // take new samples after that
283             if (g_current_sensor == NUMBER_OF_SENSORS) {
284
285                // Sample number (hex) ADC value 1 (hex) ADC value 2 (hex) ... <LF><CR>
286                // 000 000 000 ... <LF><CR>
287                // 001 000 000 ... <LF><CR>
288
289                u8_t uartbuf[16];
290
291                uart_init();
292
293                // Sample number (hex)
294                sprintf(uartbuf, "%03X ", g_sample_cnt);
295                uart_tx(uartbuf, 4);
296
297                // ADC values
298                u8_t i;
299                for (i = 0; i < NUMBER_OF_SENSORS; i++) {
300                   sprintf(uartbuf, "%03X ", g_samplebuf[i]);
301                   uart_tx(uartbuf, 4);
302                }
303
304                // <LF><CR>
305                uartbuf[0] = 10;
306                uartbuf[1] = 13;
307                uart_tx(uartbuf, 2);
308
309                g_sample_cnt++;
310                g_current_sensor = 0;
311                g_state = S_BROADCAST_SAMPLE;
312             }
313          }
314          break;
315
316          case S_BROADCAST_SLEEP: {
317
318             // Send SLEEP command to all sensors
319             // 1 byte target address + 1 byte command + 4 bytes zero padding
320             u8_t txsendsamplebuf[] = { BROADCAST, COMMAND_DOWNLINK_SLEEP, 0, 0, 0, 0 };
321             tx_single_packet(txsendsamplebuf);
322
323             // Refresh LCD
324             LCD_send_cmd(LCD_CLR);
325             LCD_send_cmd(LCD_LINE1);
326             LCD_send_text("End of");
327             LCD_send_cmd(LCD_LINE2);
328             LCD_send_text("Measurement");
329
330             // End of measurement
331             while (1);
```

```
332            }
333            break;
334
335            default: break;
336         }
337      }
338
339      return 0;
340   }
341
342   /*---------------------------------------------------------------------------------
343     Interrupt service handler
344   ---------------------------------------------------------------------------------*/
345
346   interrupt (PORT1_VECTOR) isr_port1(void)
347   {
348      if (BUTTON1_IRQ_PENDING) { // Start button pressed
349
350         // Disable and clear Button 1 interrupt
351         BUTTON1_IRQ_DISABLE;
352         BUTTON1_CLEAR_IRQ;
353
354         // Refresh LCD
355         LCD_send_cmd(LCD_CLR);
356         LCD_send_cmd(LCD_LINE1);
357         LCD_send_text("Starting ...");
358
359         // Suppose all sensors are asleep
360         u8_t i;
361         for (i = 0; i < NUMBER_OF_SENSORS; i++) {
362            g_awake_sensors[i] = 0;
363         }
364
365         // Send the wakeup signal
366         g_state = S_BROADCAST_WAKEUP;
367         tx_wakeup();
368
369         // Wait 1 second before starting (LCD message)
370         delay_ms(1000);
371
372         // Change to the next state where every sensor is
373         // checked whether the wakeup was successful
374         g_current_sensor = 0;
375         g_state = S_TX_IS_AWAKE;
376      }
377
378      if (CC1101_GDO2_IRQ_PENDING) { // Sync word detected
379
380         // Disable and clear the sync word detected interrupt
381         CC1101_GDO2_IRQ_DISABLE;
382         CC1101_GDO2_CLEAR_IRQ;
383
384         // Wait until packet completely received
385         while(CC1101_GDO2_IN);
386
387         // Check whether received data is valid
388         if (CC1101_GDO0_IN == 1 && cc1101_get_rxbytes() == UPLINK_DATA_BYTES) {
389
390            // 11 bytes of data expected
391            u8_t rxbuf[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };
392            cc1101_read_rx_fifo(rxbuf, UPLINK_DATA_BYTES);
393
394            // Data packet for this sensor?
395            if (rxbuf[0] == ADDRESS_BASE_STATION) {
396
397               // The received data needs to be interpreted depending on the current state
398               // Therefore the data is copied to a global buffer
```

```
399             u8_t i;
400             for (i = 0; i < UPLINK_DATA_BYTES; i++) {
401                g_rxbuf[i] = rxbuf[i];
402             }
403
404             g_data_received_flag = 1;
405          }
406       }
407       else {
408          // Received corrupted packet
409          // Error handling has to be done here later
410       }
411
412       cc1101_idle(); // Transceiver back to IDLE state
413    }
414 }
415
416 interrupt (PORT2_VECTOR) isr_port2(void)
417 {
418    if (RTC_IRQ_PENDING) { // RTC interrupt (each second)
419
420       RTC_IRQ_DISABLE; // Disable RTC interrupt
421
422       if (g_state == S_BROADCAST_SAMPLE) {
423
424          LED1_TOGGLE; // Used as trigger for the scope
425          LED2_TOGGLE;
426
427          // Send SAMPLE command to all sensors
428          // Address + 1 byte command + 4 bytes zero padding
429          u8_t txbuf[] = { BROADCAST, COMMAND_DOWNLINK_SAMPLE, 0, 0, 0, 0 };
430          tx_single_packet(txbuf);
431
432          // Change to next state where each sensor is
433          // asked for its taken sample
434          g_state = S_TX_SEND_SAMPLE;
435       }
436
437       // Clear RTC interrupt
438       RTC_CLEAR_IRQ;
439    }
440 }
441
442 /*---------------------------------------------------------------------------
443  Private functions
444 ---------------------------------------------------------------------------*/
445
446 void rx(void)
447 {
448    cc1101_init();
449    cc1101_reset();      // Reset transceiver and go to idle state
450    cc1101_config_packet(); // Configure transceiver for packet reception
451    CC1101_GDO2_CLEAR_IRQ; // Clear the sync word detected interrupt
452    CC1101_GDO2_IRQ_ENABLE; // Enable the sync word detected interrupt
453    cc1101_rx();         // Transceiver in RX state
454 }
455
456 void tx_wakeup(void)
457 {
458    // Disable and clear the sync word detected interrupt
459    CC1101_GDO2_IRQ_DISABLE;
460    CC1101_GDO2_CLEAR_IRQ;
461
462    TXPIN_DIR_IN;  // TX pin is input until CC1101 is in TX state
463    TIMERA_OFF;    // Make sure timer for TX pin toggling is not running
464    TIMERA_RESET;  // Reset timer for TX pin toggling
465    TIMERA_CONFIG_125; // Configure the timer for toggling the TX pin with 125 kHz
```

```
466
467     cc1101_reset();         // Reset all registers to their default values and go to idle state
468     cc1101_config_no_packet(); // Configure the transceiver for sending the wakeup signal
469     cc1101_tx_asynchronous_mode(); // Change to TX state
470
471     TXPIN_DIR_OUT; // TX pin is output
472     TXPIN_TIMER;   // TX pin driven by the timer
473     TIMERA_UP;     // Start the timer that is toggling the TX pin
474
475     delay_ms(WAKEUP_DURATION_MS); // Transmit the wakeup signal
476
477     TIMERA_OFF;    // Stop the timer again
478     TXPIN_MAN;     // TX pin controlled manually
479     TXPIN_DIR_IN;  // TX pin is input again
480
481     cc1101_idle(); // Transceiver back to IDLE state
482 }
483
484 void tx_single_packet(u8_t * txbuf)
485 {
486     // Disable and clear the sync word detected interrupt
487     CC1101_GDO2_IRQ_DISABLE;
488     CC1101_GDO2_CLEAR_IRQ;
489
490     TXPIN_MAN;        // TX pin controlled manually
491     TXPIN_DIR_IN;     // TX pin is input
492
493     cc1101_init();
494     cc1101_reset();   // Reset chip and go to idle state
495     cc1101_config_packet(); // Configure the transceiver for sending packets
496
497     cc1101_fill_tx_fifo(txbuf, DOWNLINK_DATA_BYTES);
498     cc1101_tx();
499     cc1101_idle();
500 }
501
502 // Taken (modified) from "Diagnosefunktion üfr Automobil-
503 // Starterbatterien mit drahtlosen Zellensensoren" by
504 // Simon Puettjer p. 167
505 void osc_configuration(void)
506 {
507     u16_t max_tries = 1000;
508     u8_t i;
509
510     _BIC_SR(OSCOFF);           // Switch LFXT on
511     BCSCTL1 &= ~XT2OFF;        // XT2on
512     do {
513         IFG1 &= ~OFIFG;        // Clear OSCFault flag
514         for (i = 0xFF; i > 0; i--); // Time for flag to set
515         if (!(--max_tries)) {  // Stop after 1k tries, error with ext osc!
516         }
517     }
518     while ((IFG1 & OFIFG));    // OSCFault flag still set?
519     IFG1 &= ~OFIFG;            // Clear OSCFault flag
520
521     BCSCTL2 |= SELM_2 | SELS;  // MCLK = SMCLK = XT2 (safe)
522
523     // MCLK = 8 MHz
524     BCSCTL2 &= ~(DIVS_3 | DIVM_3); // SMCLK = MCLK = XT2 / 1
525 }
```

Listing D.19: main.h

```
1  /*---------------------------------------------------------------------------
2     Description: Generic defines and macros.
3     Date:       11/22/2012
```

```
4       Last Update: 11/28/2012
5       Author:     Phillip Durdaut
6    -------------------------------------------------------------------------------*/
7
8    #ifndef MAIN_H_
9    #define MAIN_H_
10
11   #include <msp430x16x.h>
12   #include <signal.h>
13   #include <stdio.h>
14   #include <stdlib.h>
15
16   #include "types.h"
17   #include "delay.h"
18   #include "cc1101.h"
19   #include "uart.h"
20   #include "lcd16x2.h"
21
22   /*-------------------------------------------------------------------------------
23    Defines
24   -------------------------------------------------------------------------------*/
25
26   #define NUMBER_OF_SENSORS     4
27   #define NUMBER_OF_SAMPLES_PER_SENSOR 61
28
29   #define WAKEUP_DURATION_MS    10
30
31   #define DOWNLINK_DATA_BYTES   6      // Incl. address byte
32   #define UPLINK_DATA_BYTES     11     // Incl. address byte
33
34   // Address definitions
35   #define BROADCAST             0x00
36   #define ADDRESS_BASE_STATION  0xFF
37   #define ADDRESS_SENSOR_1      0x01
38   #define ADDRESS_SENSOR_2      0x02
39   #define ADDRESS_SENSOR_3      0x03
40   #define ADDRESS_SENSOR_4      0x04
41
42   // Downlink commands
43   #define COMMAND_DOWNLINK_IS_AWAKE 0x01
44   #define COMMAND_DOWNLINK_SAMPLE 0x02
45   #define COMMAND_DOWNLINK_SEND_SAMPLE 0x03
46   #define COMMAND_DOWNLINK_SLEEP 0x04
47
48   /*-------------------------------------------------------------------------------
49    Types
50   -------------------------------------------------------------------------------*/
51
52   typedef enum { S_INIT,
53              S_BROADCAST_WAKEUP,
54              S_TX_IS_AWAKE,
55              S_RX_IS_AWAKE,
56              S_BROADCAST_SAMPLE,
57              S_TX_SEND_SAMPLE,
58              S_RX_SAMPLE,
59              S_BROADCAST_SLEEP
60            } state_t ;
61
62   /*-------------------------------------------------------------------------------
63    Macros
64   -------------------------------------------------------------------------------*/
65
66   // Button 1 (Start)
67   #define BUTTON1_DIR_IN  (P1DIR &= ~BIT5)
68   #define BUTTON1_FALLING_EDGE (P1IES |= BIT5)
69   #define BUTTON1_IRQ_ENABLE (P1IE |= BIT5)
70   #define BUTTON1_IRQ_DISABLE (P1IE &= ~BIT5)
```

```
71  #define BUTTON1_IRQ_PENDING ((P1IFG & BIT5) == BIT5)
72  #define BUTTON1_CLEAR_IRQ (P1IFG &= ~(BIT5))
73
74  // RTC
75  #define RTC_DIR_IN       (P2DIR &= ~BIT5)
76  #define RTC_FALLING_EDGE (P2IES |= BIT5)
77  #define RTC_IRQ_ENABLE   (P2IE |= BIT5)
78  #define RTC_IRQ_DISABLE  (P2IE &= ~BIT5)
79  #define RTC_IRQ_PENDING ((P2IFG & BIT5) == BIT5)
80  #define RTC_CLEAR_IRQ    (P2IFG &= ~(BIT5))
81
82  // LEDs
83  #define LEDx_DIR_OUT     (P3DIR |= (BIT7 | BIT6))
84  #define LED1_ON          (P3OUT &= ~BIT6)
85  #define LED1_OFF         (P3OUT |= BIT6)
86  #define LED1_TOGGLE      (P3OUT ^= BIT6)
87  #define LED2_ON          (P3OUT &= ~BIT7)
88  #define LED2_OFF         (P3OUT |= BIT7)
89  #define LED2_TOGGLE      (P3OUT ^= BIT7)
90
91  // LCD
92  #define LCD_DIR_OUT      (P4DIR = 0xFF)
93  #define LCD_OUT_0x00     (P4OUT = 0x00)
94
95  // CC1101 asynchronous TX
96  #define TXPIN_TIMER      (P1SEL |= BIT2)
97  #define TXPIN_MAN        (P1SEL &= ~BIT2)
98  #define TXPIN_DIR_IN     (P1DIR &= ~BIT2)
99  #define TXPIN_DIR_OUT    (P1DIR |= BIT2)
100 #define TXPIN_LOW        (P1OUT &= ~BIT2)
101 #define TXPIN_HIGH       (P1OUT |= BIT2)
102
103 // Timer A
104 #define TIMERA_OFF       (TACTL &= ~(MC0 | MC1))
105 #define TIMERA_RESET     (TACTL |= TACLR)
106
107 // Clock source is SMCLK / 1
108 // PWM period 8 us
109 #define TIMERA_CONFIG_125 (TACTL |= TASSEL1); \
110                           (TACCR0 = 64 - 1); \
111                           (TACCR1 = 32 - 1); \
112                           (TACCTL1 |= (OUTMOD2 | OUTMOD1 | OUTMOD0))
113 #define TIMERA_UP        (TACTL |= MC0)
114
115 #endif /* MAIN_H_ */
```

Listing D.20: cc1101.c

```
1   /*-------------------------------------------------------------------------------
2      Description: Driver for Texas Instruments CC1101 Low-Power Sub-1 GHz
3                   RF Transceiver.
4      Date:        10/02/2012
5      Last Update: 10/03/2012
6      Author:      Phillip Durdaut
7   -------------------------------------------------------------------------------*/
8
9   #include "main.h"
10
11  /*-------------------------------------------------------------------------------
12   Variables
13   -------------------------------------------------------------------------------*/
14
15  /* Output power table (433 MHz): -30, 10 (dBm) */
16  u8_t cc1101_patable[] = { 0x12, 0xc0 };
17  u8_t cc1101_patablelen = 2;
18
```

```
19  /*---------------------------------------------------------------------------
20   Defines
21   ---------------------------------------------------------------------------*/
22
23  /* Command Strobes (Table 42 in datasheet) */
24
25  #define CC1101_CS_SRES     0x30
26  #define CC1101_CS_SFSTXON  0x31
27  #define CC1101_CS_SXOFF    0x32
28  #define CC1101_CS_SCAL     0x33
29  #define CC1101_CS_SRX      0x34
30  #define CC1101_CS_STX      0x35
31  #define CC1101_CS_SIDLE    0x36
32  #define CC1101_CS_SAFC     0x37
33  #define CC1101_CS_SWOR     0x38
34  #define CC1101_CS_SPWD     0x39
35  #define CC1101_CS_SFRX     0x3A
36  #define CC1101_CS_SFTX     0x3B
37  #define CC1101_CS_SWORRST  0x3C
38  #define CC1101_CS_SNOP     0x3D
39
40  /* Configuration Registers */
41
42  #define CC1101_CR_IOCFG2   0x00   /* GDO2 output pin configuration */
43  #define CC1101_CR_IOCFG1   0x01   /* GDO1 output pin configuration */
44  #define CC1101_CR_IOCFG0   0x02   /* GDO0 output pin configuration */
45  #define CC1101_CR_FIFOTHR  0x03   /* RX FIFO and TX FIFO thresholds */
46  #define CC1101_CR_SYNC1    0x04   /* Sync word, high byte */
47  #define CC1101_CR_SYNC0    0x05   /* Sync word, low byte */
48  #define CC1101_CR_PKTLEN   0x06   /* Packet length */
49  #define CC1101_CR_PKTCTRL1 0x07   /* Packet automation control */
50  #define CC1101_CR_PKTCTRL0 0x08   /* Packet automation control */
51  #define CC1101_CR_ADDR     0x09   /* Device address */
52  #define CC1101_CR_CHANNR   0x0A   /* Channel number */
53  #define CC1101_CR_FSCTRL1  0x0B   /* Frequency synthesizer control */
54  #define CC1101_CR_FSCTRL0  0x0C   /* Frequency synthesizer control */
55  #define CC1101_CR_FREQ2    0x0D   /* Frequency control word, high byte */
56  #define CC1101_CR_FREQ1    0x0E   /* Frequency control word, middle byte */
57  #define CC1101_CR_FREQ0    0x0F   /* Frequency control word, low byte */
58  #define CC1101_CR_MDMCFG4  0x10   /* Modem configuration */
59  #define CC1101_CR_MDMCFG3  0x11   /* Modem configuration */
60  #define CC1101_CR_MDMCFG2  0x12   /* Modem configuration */
61  #define CC1101_CR_MDMCFG1  0x13   /* Modem configuration */
62  #define CC1101_CR_MDMCFG0  0x14   /* Modem configuration */
63  #define CC1101_CR_DEVIATN  0x15   /* Modem deviation setting */
64  #define CC1101_CR_MCSM2    0x16   /* Main Radio Cntrl State Machine config */
65  #define CC1101_CR_MCSM1    0x17   /* Main Radio Cntrl State Machine config */
66  #define CC1101_CR_MCSM0    0x18   /* Main Radio Cntrl State Machine config */
67  #define CC1101_CR_FOCCFG   0x19   /* Frequency Offset Compensation config */
68  #define CC1101_CR_BSCFG    0x1A   /* Bit Synchronization configuration */
69  #define CC1101_CR_AGCCTRL2 0x1B   /* AGC control */
70  #define CC1101_CR_AGCCTRL1 0x1C   /* AGC control */
71  #define CC1101_CR_AGCCTRL0 0x1D   /* AGC control */
72  #define CC1101_CR_WOREVT1  0x1E   /* High byte Event 0 timeout */
73  #define CC1101_CR_WOREVT0  0x1F   /* Low byte Event 0 timeout */
74  #define CC1101_CR_WORCTRL  0x20   /* Wake On Radio control */
75  #define CC1101_CR_FREND1   0x21   /* Front end RX configuration */
76  #define CC1101_CR_FREND0   0x22   /* Front end TX configuration */
77  #define CC1101_CR_FSCAL3   0x23   /* Frequency synthesizer calibration */
78  #define CC1101_CR_FSCAL2   0x24   /* Frequency synthesizer calibration */
79  #define CC1101_CR_FSCAL1   0x25   /* Frequency synthesizer calibration */
80  #define CC1101_CR_FSCAL0   0x26   /* Frequency synthesizer calibration */
81  #define CC1101_CR_RCCTRL1  0x27   /* RC oscillator configuration */
82  #define CC1101_CR_RCCTRL0  0x28   /* RC oscillator configuration */
83  #define CC1101_CR_FSTEST   0x29   /* Frequency synthesizer cal control */
84  #define CC1101_CR_PTEST    0x2A   /* Production test */
85  #define CC1101_CR_AGCTEST  0x2B   /* AGC test */
```

```
86  #define CC1101_CR_TEST2    0x2C   /* Various test settings */
87  #define CC1101_CR_TEST1    0x2D   /* Various test settings */
88  #define CC1101_CR_TEST0    0x2E   /* Various test settings */
89
90  /* Status Registers */
91
92  #define CC1101_SR_PARTNUM  0x30   /* Part number */
93  #define CC1101_SR_VERSION  0x31   /* Current version number */
94  #define CC1101_SR_FREQEST  0x32   /* Frequency offset estimate */
95  #define CC1101_SR_LQI      0x33   /* Demodulator estimate for link quality */
96  #define CC1101_SR_RSSI     0x34   /* Received signal strength indication */
97  #define CC1101_SR_MARCSTATE 0x35  /* Control state machine state */
98  #define CC1101_SR_WORTIME1 0x36   /* High byte of WOR timer */
99  #define CC1101_SR_WORTIME0 0x37   /* Low byte of WOR timer */
100 #define CC1101_SR_PKTSTATUS 0x38  /* Current GDOx status and packet status */
101 #define CC1101_SR_VCO_VC_DAC 0x39 /* Current setting from PLL cal module */
102 #define CC1101_SR_TXBYTES  0x3A   /* Underflow and # of bytes in TXFIFO */
103 #define CC1101_SR_RXBYTES  0x3B   /* Overflow and # of bytes in RXFIFO */
104 #define CC1101_SR_RCCTRL1_STATUS 0x3C /* Last RC oscilator calibration results */
105 #define CC1101_SR_RCCTRL0_STATUS 0x3D /* Last RC oscilator calibration results */
106
107 /* Single / Burst access */
108
109 #define CC1101_WRITE_BURST 0x40
110 #define CC1101_READ_SINGLE 0x80
111 #define CC1101_READ_BURST  0xC0
112
113 /* Memory locations */
114
115 #define CC1101_ML_PATABLE  0x3E
116 #define CC1101_ML_TXFIFO   0x3F
117 #define CC1101_ML_RXFIFO   0x3F
118
119 /*---------------------------------------------------------------------------
120  Prototypes of the private functions
121 ---------------------------------------------------------------------------*/
122
123 void cc1101_spi_setup(void);
124 void cc1101_spi_write_register(u8_t address, u8_t value);
125 void cc1101_spi_write_register_burst(u8_t address, u8_t * buffer, u8_t count);
126 char cc1101_spi_read_register(u8_t address);
127 void cc1101_spi_read_register_burst(u8_t address, u8_t *buffer, u8_t count);
128 u8_t cc1101_spi_read_status(u8_t address);
129 void cc1101_spi_command_strobe(u8_t strobe);
130
131 /*---------------------------------------------------------------------------
132  Public functions
133 ---------------------------------------------------------------------------*/
134
135 void cc1101_init(void)
136 {
137    cc1101_spi_setup();
138
139    CC1101_GDO2_DIR_IN;
140    CC1101_GDO2_IRQ_RISING_EDGE;
141    CC1101_GDO2_IRQ_DISABLE;
142    CC1101_GDO2_CLEAR_IRQ;
143 }
144
145 void cc1101_power_up_reset(void)
146 {
147    // Manual power-on reset
148    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
149    delay_100us(1);
150    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
151    delay_100us(1);
152    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
```

```
153     delay_100us(5);
154     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
155     delay_100us(5);
156
157     cc1101_reset();
158  }
159
160  void cc1101_reset(void)
161  {
162     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
163
164     while (!(IFG1 & UTXIFG0));        // Wait for TX to finish
165     U0TXBUF = CC1101_CS_SRES;         // Send reset strobe
166     while(!(UTCTL0 & TXEPT));         // Wait for TX to finish
167
168     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
169
170     delay_ms(1);
171  }
172
173  void cc1101_config_no_packet(void)
174  {
175     // GDO0 at high impedance
176     cc1101_spi_write_register(CC1101_CR_IOCFG0, 0x2E);
177
178     // GDO2 at high impedance
179     cc1101_spi_write_register(CC1101_CR_IOCFG2, 0x2E);
180
181     // Asynchronous serial mode, Infinite packet length mode
182     cc1101_spi_write_register(CC1101_CR_PKTCTRL0, 0x32);
183
184     // Channel 0
185     cc1101_spi_write_register(CC1101_CR_CHANNR, 0x00);
186
187     // Carrier frequency: 433.999969 MHz
188     cc1101_spi_write_register(CC1101_CR_FREQ2, 0x10);
189     cc1101_spi_write_register(CC1101_CR_FREQ1, 0xB1);
190     cc1101_spi_write_register(CC1101_CR_FREQ0, 0x3B);
191
192     cc1101_spi_write_register(CC1101_CR_MDMCFG4, 0xFD);
193
194     // 250 kBaud, OOK
195     cc1101_spi_write_register(CC1101_CR_MDMCFG3, 0x3B);
196     cc1101_spi_write_register(CC1101_CR_MDMCFG2, 0x30);
197
198     cc1101_spi_write_register(CC1101_CR_MDMCFG1, 0x00);
199
200     cc1101_spi_write_register(CC1101_CR_DEVIATN, 0x15);
201     cc1101_spi_write_register(CC1101_CR_MCSM0, 0x18);
202
203     // FCL gain: 3000, Saturation point for the frequency offset compensation algorithm ->
            SmartRF Studio
204     cc1101_spi_write_register(CC1101_CR_FOCCFG, 0x16);
205
206     // 10 dBm output power
207     cc1101_spi_write_register_burst(CC1101_ML_PATABLE, cc1101_patable, cc1101_patablelen);
208     cc1101_spi_write_register(CC1101_CR_FREND0, 0x11);
209  }
210
211  void cc1101_config_packet(void)
212  {
213     // Rising edge on GDO0 when packet received and CRC check OK
214     // (Deasserted when first byte is read from RX FIFO)
215     cc1101_spi_write_register(CC1101_CR_IOCFG0, 0x07);
216
217     // Rising edge on GDO2 when sync word has been received
218     cc1101_spi_write_register(CC1101_CR_IOCFG2, 0x06);
```

```
219
220     // The 4 SYNC bytes: 0x12 0x09 (repeated once)
221     cc1101_spi_write_register(CC1101_CR_SYNC1, 0x12);
222     cc1101_spi_write_register(CC1101_CR_SYNC0, 0x09);
223
224     // Flush RX packets when CRC is not OK, Address check and 0x00 broadcast
225     cc1101_spi_write_register(CC1101_CR_PKTCTRL1, 0x0A);
226
227     // No whitening, FIFO mode, CRC disable, Fixed packet length
228     cc1101_spi_write_register(CC1101_CR_PKTCTRL0, 0x00);
229
230     // Device Address
231     cc1101_spi_write_register(CC1101_CR_ADDR, ADDRESS_BASE_STATION);
232
233     // Channel 0
234     cc1101_spi_write_register(CC1101_CR_CHANNR, 0x00);
235
236     // IF frequency: 152.34375 kHz
237     cc1101_spi_write_register(CC1101_CR_FSCTRL1, 0x06);
238
239     // Carrier frequency: 433.999969 MHz
240     cc1101_spi_write_register(CC1101_CR_FREQ2, 0x10);
241     cc1101_spi_write_register(CC1101_CR_FREQ1, 0xB1);
242     cc1101_spi_write_register(CC1101_CR_FREQ0, 0x3B);
243
244     // 40 kBaud, OOK, Channel spacing: 149.963379 kHz, RX filter bandwidth: 203.125000 kHz
245     // No Manchester en/decoding, 30/32 SYNC bits detected, No FEC, 4 Preamble bytes
246     cc1101_spi_write_register(CC1101_CR_MDMCFG4, 0x8A);
247     cc1101_spi_write_register(CC1101_CR_MDMCFG3, 0x93);
248     cc1101_spi_write_register(CC1101_CR_MDMCFG2, 0x33);
249     cc1101_spi_write_register(CC1101_CR_MDMCFG1, 0x22);
250     cc1101_spi_write_register(CC1101_CR_MDMCFG0, 0x7A);
251
252     // Calibrate when going from IDLE to RX or TX, Crystal off when in SLEEP state
253     cc1101_spi_write_register(CC1101_CR_MCSM0, 0x10);
254
255     // FCL gain: 3000, Saturation point for the frequency offset compensation algorithm ->
            SmartRF Studio
256     cc1101_spi_write_register(CC1101_CR_FOCCFG, 0x16);
257
258     //cc1101_spi_write_register(CC1101_CR_FREND1, 0x00);
259     cc1101_spi_write_register_burst(CC1101_ML_PATABLE, cc1101_patable, cc1101_patablelen);
260     cc1101_spi_write_register(CC1101_CR_FREND0, 0x11); // 10 dBm output power
261 }
262
263 void cc1101_fill_tx_fifo(u8_t * buffer, u8_t length)
264 {
265     // Clear TX FIFO
266     cc1101_clear_tx_fifo();
267
268     // Transfer the bytes via SPI to the TX fifo of the transceiver
269     cc1101_spi_write_register_burst(CC1101_ML_TXFIFO, buffer, length);
270 }
271
272 void cc1101_read_rx_fifo(u8_t * buffer, u8_t length)
273 {
274     // Disable the receiver
275     cc1101_idle();
276
277     // Transfer the bytes via SPI from the RX fifo
278     cc1101_spi_read_register_burst(CC1101_ML_RXFIFO, buffer, length);
279 }
280
281 void cc1101_enable_crc(void)
282 {
283     u8_t current = cc1101_spi_read_register(CC1101_CR_PKTCTRL0);
284     cc1101_spi_write_register(CC1101_CR_PKTCTRL0, current | BIT2);
```

```
285 }
286
287 void cc1101_disable_crc(void)
288 {
289    u8_t current = cc1101_spi_read_register(CC1101_CR_PKTCTRL0);
290    cc1101_spi_write_register(CC1101_CR_PKTCTRL0, current & ~BIT2);
291 }
292
293 void cc1101_clear_tx_fifo(void)
294 {
295    cc1101_spi_command_strobe(CC1101_CS_SFTX);
296 }
297
298 void cc1101_clear_rx_fifo(void)
299 {
300    cc1101_spi_command_strobe(CC1101_CS_SFRX);
301 }
302
303 u8_t cc1101_get_rxbytes(void)
304 {
305    return (cc1101_spi_read_status(CC1101_SR_RXBYTES) & 0x7F);
306 }
307
308 void cc1101_idle(void)
309 {
310    cc1101_spi_command_strobe(CC1101_CS_SIDLE);
311 }
312
313 void cc1101_tx_asynchronous_mode(void)
314 {
315    cc1101_spi_command_strobe(CC1101_CS_STX);
316 }
317
318 void cc1101_tx(void)
319 {
320    // DATA packet length
321    cc1101_spi_write_register(CC1101_CR_PKTLEN, DOWNLINK_DATA_BYTES);
322
323    // Enable CRC calculation when transmitting data
324    cc1101_enable_crc();
325
326    // TX state
327    cc1101_spi_command_strobe(CC1101_CS_STX);
328
329    // Wait 5 ms for TX finished
330    delay_ms(5);
331 }
332
333 void cc1101_rx(void)
334 {
335    // DATA packet length
336    cc1101_spi_write_register(CC1101_CR_PKTLEN, UPLINK_DATA_BYTES);
337
338    // Enable CRC check when receiving data
339    cc1101_enable_crc();
340
341    // Clear RX FIFO
342    cc1101_clear_rx_fifo();
343
344    // RX state
345    cc1101_spi_command_strobe(CC1101_CS_SRX);
346 }
347
348 u8_t cc1101_get_partnum(void)
349 {
350    return cc1101_spi_read_status(CC1101_SR_PARTNUM);
351 }
```

```
352
353 u8_t cc1101_get_version(void)
354 {
355    return cc1101_spi_read_status(CC1101_SR_VERSION);
356 }
357
358 u8_t cc1101_get_marcstate(void)
359 {
360    return (cc1101_spi_read_status(CC1101_SR_MARCSTATE) & 0x1F);
361 }
362
363 /*---------------------------------------------------------------------------
364  Private functions
365 ---------------------------------------------------------------------------*/
366
367 void cc1101_spi_setup(void)
368 {
369    CC1101_CSn_PxDIR |= CC1101_CSn_PIN; // nCS is output
370    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Unselect CC1101 chip
371
372    ME1 |= USPIE0;                    // Enable USART0 SPI
373    U0CTL |= SWRST;                   // USART logic held in reset state
374    U0CTL |= (CHAR | SYNC | MM);      // 8-bit, SPI, Master
375    U0TCTL |= (CKPH | SSEL1 | SSEL0 | STC); // UCLK delayed, SMCLK, 3-pin SPI mode
376
377    U0BR0 = 2; // BRCLK/2
378    U0BR1 = 0;
379    U0MCTL = 0; // Always 0 in SPI mode
380
381    // SPI functionality for pins
382    CC1101_SPI_PxSEL |= (CC1101_SPI_SI_PIN | CC1101_SPI_SO_GDO1_PIN | CC1101_SPI_SCLK_PIN);
383    CC1101_SPI_PxDIR |= (CC1101_SPI_SI_PIN | CC1101_SPI_SCLK_PIN); // SI and SCLK are outputs
384
385    UCTL0 &= ~SWRST; // Initialize USART state machine
386 }
387
388 void cc1101_spi_write_register(u8_t address, u8_t value)
389 {
390    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
391    while (!(IFG1&UTXIFG0));          // Wait for TX to finish
392    U0TXBUF = address;                // Send address
393    while (!(IFG1&UTXIFG0));          // Wait for TX to finish
394    U0TXBUF = value;                  // Send value
395    while(!(UTCTL0&TXEPT));           // Wait for TX complete
396    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
397 }
398
399 void cc1101_spi_write_register_burst(u8_t address, u8_t * buffer, u8_t count)
400 {
401    u8_t i;
402
403    CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
404    while (!(IFG1 & UTXIFG0));        // Wait for TX to finish
405    U0TXBUF = address | CC1101_WRITE_BURST; // Send address
406
407    for (i = 0; i < count; i++) {
408       while (!(IFG1 & UTXIFG0));    // Wait for TX to finish
409       U0TXBUF = buffer[i];          // Send data
410    }
411
412    while(!(UTCTL0 & TXEPT));
413    CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
414 }
415
416 char cc1101_spi_read_register(u8_t address)
417 {
418    u8_t x;
```

```
419
420     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
421     while (!(IFG1 & UTXIFG0));        // Wait for TX to finish
422     U0TXBUF = (address | CC1101_READ_SINGLE); // Send address
423     while (!(IFG1 & UTXIFG0));        // Wait for TX to finish
424     U0TXBUF = 0;                      // Dummy write so we can read data
425     while(!(UTCTL0 & TXEPT));         // Wait for TX complete
426     x = U0RXBUF;                      // Read data
427     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
428
429     return x;
430  }
431
432  void cc1101_spi_read_register_burst(u8_t address, u8_t * buffer, u8_t count)
433  {
434     u16_t i;
435
436     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
437     while (!(IFG1 & UTXIFG0));      // Wait for TXBUF ready
438     U0TXBUF = (address | CC1101_READ_BURST); // Send address
439     while(!(UTCTL0 & TXEPT));       // Wait for TX complete
440
441     U0TXBUF = 0;
442     while (!(IFG1&URXIFG0));
443
444     for (i = 0; i < count; i++) {
445        U0TXBUF = 0;                 // Initiate next data RX, meanwhile
446        while (!(IFG1&URXIFG0));     // Wait for end of TX data byte
447        buffer[i] = U0RXBUF;         // Store data from last data RX
448     }
449
450     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
451  }
452
453  u8_t cc1101_spi_read_status(u8_t address)
454  {
455     u8_t status;
456
457     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
458     while (!(IFG1 & UTXIFG0));      // Wait for TX to finish
459     U0TXBUF = (address | CC1101_READ_BURST); // Send address
460     while (!(IFG1 & UTXIFG0));      // Wait for TX to finish
461     U0TXBUF = 0;                    // Dummy write so we can read data
462     while(!(UTCTL0 & TXEPT));       // Wait for TX complete
463     status = U0RXBUF;               // Read data
464     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
465
466     return status;
467  }
468
469  void cc1101_spi_command_strobe(u8_t strobe)
470  {
471     CC1101_CSn_PxOUT &= ~CC1101_CSn_PIN; // Chip enable
472     while (!(IFG1 & UTXIFG0));      // Wait for TX to finish
473     U0TXBUF = strobe;               // Send strobe
474     IFG1 &= ~URXIFG0;               // Clear flag
475     while(!(UTCTL0 & TXEPT));       // Wait for TX complete
476     CC1101_CSn_PxOUT |= CC1101_CSn_PIN; // Chip disable
477  }
```

Listing D.21: cc1101.h

```
1  /*-------------------------------------------------------------------------------
2     Description: Driver for Texas Instruments CC1101 Low-Power Sub-1 GHz
3                  RF Transceiver.
4     Date:        10/02/2012
```

```
5       Last Update: 10/03/2012
6       Author:     Phillip Durdaut
7    ------------------------------------------------------------------------------*/
8
9    #ifndef CC1101_H_
10   #define CC1101_H_
11
12   #include "main.h"
13
14   /*---------------------------------------------------------------------------------
15    Defines -> Need to be changed depending on hardware
16    ---------------------------------------------------------------------------------*/
17
18   #define CC1101_CSn_PxDIR   P3DIR
19   #define CC1101_CSn_PxOUT   P3OUT
20   #define CC1101_CSn_PIN     BIT0
21
22   #define CC1101_SPI_PxSEL   P3SEL
23   #define CC1101_SPI_PxDIR   P3DIR
24   #define CC1101_SPI_PxIN    P3IN
25   #define CC1101_SPI_SI_PIN  BIT1
26   #define CC1101_SPI_SO_GDO1_PIN BIT2
27   #define CC1101_SPI_SCLK_PIN BIT3
28
29   #define CC1101_GDO0_PxDIR  P1DIR
30   #define CC1101_GDO0_PxIN   P1IN
31   #define CC1101_GDO0_PIN    BIT0
32   #define CC1101_GDO0_POS    0
33
34   #define CC1101_GDO2_PxDIR  P1DIR
35   #define CC1101_GDO2_PxIN   P1IN
36   #define CC1101_GDO2_PxIES  P1IES
37   #define CC1101_GDO2_PxIE   P1IE
38   #define CC1101_GDO2_PxIFG  P1IFG
39   #define CC1101_GDO2_PIN    BIT1
40   #define CC1101_GDO2_POS    1
41
42   /*---------------------------------------------------------------------------------
43    Macros
44   ---------------------------------------------------------------------------------*/
45
46   #define CC1101_GDO0_DIR_IN    (CC1101_GDO0_PxDIR &= ~CC1101_GDO0_PIN)
47   #define CC1101_GDO0_IN        ((CC1101_GDO0_PxIN & CC1101_GDO0_PIN) >> CC1101_GDO0_POS)
48
49   #define CC1101_GDO2_DIR_IN    (CC1101_GDO2_PxDIR &= ~CC1101_GDO2_PIN)
50   #define CC1101_GDO2_IN        ((CC1101_GDO2_PxIN & CC1101_GDO2_PIN) >> CC1101_GDO2_POS)
51   #define CC1101_GDO2_IRQ_RISING_EDGE (CC1101_GDO2_PxIES &= ~CC1101_GDO2_PIN)
52   #define CC1101_GDO2_IRQ_FALLING_EDGE (CC1101_GDO2_PxIES |= CC1101_GDO2_PIN)
53   #define CC1101_GDO2_IRQ_ENABLE (CC1101_GDO2_PxIE |= CC1101_GDO2_PIN)
54   #define CC1101_GDO2_IRQ_DISABLE (CC1101_GDO2_PxIE &= ~CC1101_GDO2_PIN)
55   #define CC1101_GDO2_IRQ_PENDING ((CC1101_GDO2_PxIFG & CC1101_GDO2_PIN) == CC1101_GDO2_PIN)
56   #define CC1101_GDO2_CLEAR_IRQ (CC1101_GDO2_PxIFG &= ~(CC1101_GDO2_PIN))
57
58   /*---------------------------------------------------------------------------------
59    Public functions
60   ---------------------------------------------------------------------------------*/
61
62   void cc1101_init(void);
63   void cc1101_power_up_reset(void);
64   void cc1101_reset(void);
65   void cc1101_config_no_packet(void);
66   void cc1101_config_packet(void);
67   void cc1101_fill_tx_fifo(u8_t * buffer, u8_t length);
68   void cc1101_read_rx_fifo(u8_t * buffer, u8_t length);
69   void cc1101_enable_crc(void);
70   void cc1101_disable_crc(void);
71   void cc1101_clear_tx_fifo(void);
```

```
72 void cc1101_clear_rx_fifo(void);
73 u8_t cc1101_get_rxbytes(void);
74 void cc1101_idle(void);
75 void cc1101_tx_asynchronous_mode(void);
76 void cc1101_tx(void);
77 void cc1101_rx(void);
78 u8_t cc1101_get_partnum(void);
79 u8_t cc1101_get_version(void);
80 u8_t cc1101_get_marcstate(void);
81
82 #endif /* CC1101_H_ */
```

Listing D.22: delay.c

```
1  /*-------------------------------------------------------------------------------
2     Description: Delay functions.
3
4     Date:       10/25/2012
5     Last Update: 10/25/2012
6     Author:     Phillip Durdaut
7  -------------------------------------------------------------------------------*/
8
9  #include "main.h"
10
11 /*-------------------------------------------------------------------------------
12  Prototypes of the private functions
13 -------------------------------------------------------------------------------*/
14
15 void delay(u16_t i);
16
17 /*-------------------------------------------------------------------------------
18  Public functions
19 -------------------------------------------------------------------------------*/
20
21 void delay_ms(u16_t delay_ms)
22 {
23    while (delay_ms > 0) {
24       delay(DELAY_CYCLES_PER_MS);
25       delay_ms--;
26    }
27 }
28
29 void delay_100us(u16_t delay_100us)
30 {
31    while (delay_100us > 0) {
32       delay(DELAY_CYCLES_PER_100US);
33       delay_100us--;
34    }
35 }
36
37 /*-------------------------------------------------------------------------------
38  Private functions
39 -------------------------------------------------------------------------------*/
40
41 void delay(u16_t i)
42 {
43    while (i > 0) {
44       _NOP();
45       i--;
46    }
47 }
```

Listing D.23: delay.h

```
1  /*-------------------------------------------------------------------------------
2     Description: Delay functions.
```

```
3
4     Date:       10/25/2012
5     Last Update: 10/25/2012
6     Author:     Phillip Durdaut
7  ----------------------------------------------------------------------------*/
8
9  #ifndef DELAY_H_
10 #define DELAY_H_
11
12 #include "main.h"
13
14 /*----------------------------------------------------------------------------
15   Defines
16 ----------------------------------------------------------------------------*/
17
18 #define DELAY_CYCLES_PER_MS 610
19 #define DELAY_CYCLES_PER_100US 57
20
21 /*----------------------------------------------------------------------------
22   Public functions
23 ----------------------------------------------------------------------------*/
24
25 void delay_ms(u16_t delay_ms);
26 void delay_100us(u16_t delay_100us);
27
28 #endif /* DELAY_H_ */
```

Listing D.24: types.h

```
1  /*----------------------------------------------------------------------------
2     Description: MSP430 (mspgcc) data types.
3     Date:       12/02/2012
4     Last Update: 12/02/2012
5     Author:     Phillip Durdaut
6  ----------------------------------------------------------------------------*/
7
8  #ifndef TYPES_H_
9  #define TYPES_H_
10
11 /*----------------------------------------------------------------------------
12   Types
13 ----------------------------------------------------------------------------*/
14
15 typedef unsigned char u8_t;
16 typedef signed char  s8_t;
17 typedef unsigned int u16_t;
18 typedef signed int   s16_t;
19 typedef unsigned long u32_t;
20 typedef signed long  s32_t;
21 typedef unsigned long long u64_t;
22 typedef signed long long s64_t;
23
24 #endif /* TYPES_H_ */
```

Listing D.25: uart.c

```
1  /*----------------------------------------------------------------------------
2     Description: UART driver (only TX implemented).
3     Date:       11/30/2012
4     Last Update: 11/30/2012
5     Author:     Phillip Durdaut
6  ----------------------------------------------------------------------------*/
7
8  #include "main.h"
9
10 /*----------------------------------------------------------------------------
```

```
11   Public functions
12  ------------------------------------------------------------------------------*/
13
14  void uart_init(void)
15  {
16     UART_TX_PxDIR |= UART_TX_PIN; // TX pin is output
17     UART_TX_PxSEL |= UART_TX_PIN; // SPI functionality for TX pin
18
19     U0CTL |= SWRST; // USART logic held in reset state
20
21     U0CTL &= ~PENA; // No parity
22     U0CTL &= ~SPB; // 1 stop bit
23     U0CTL |= CHAR; // 8 data bits
24     U0CTL &= ~LISTEN; // No loopback mode
25     U0CTL &= ~SYNC; // UART mode
26
27     U0TCTL &= ~CKPL;         // UCLKI = UCLK
28     U0TCTL |= (SSEL1 | SSEL1); // BRCLK = SMCLK
29
30     // 19200 Baud
31     U0BR0 = 0xA0;
32     U0BR1 = 0x01;
33     U0MCTL = 0x6D;
34
35     ME1 |= UTXE0;  // Enable TX
36     UCTL0 &= ~SWRST; // Initialize USART state machine
37  }
38
39  void uart_tx(u8_t * buffer, u8_t length)
40  {
41     u8_t i;
42
43     for (i = 0; i < length; i++) {
44        while (!(IFG1 & UTXIFG0)); // Wait for TX to finish
45        TXBUF0 = buffer[i];  // Send data
46        delay_100us(10);
47     }
48  }
```

Listing D.26: uart.h

```
1   /*------------------------------------------------------------------------------
2      Description: UART driver (only TX implemented).
3      Date:       11/30/2012
4      Last Update: 11/30/2012
5      Author:     Phillip Durdaut
6   ------------------------------------------------------------------------------*/
7
8   #ifndef UART_H_
9   #define UART_H_
10
11  #include "main.h"
12
13  /*------------------------------------------------------------------------------
14   Defines -> Need to be changed depending on hardware
15   ------------------------------------------------------------------------------*/
16
17  #define UART_TX_PxDIR      P3DIR
18  #define UART_TX_PxSEL      P3SEL
19  #define UART_TX_PIN        BIT4
20
21  /*------------------------------------------------------------------------------
22   Public functions
23  ------------------------------------------------------------------------------*/
24
25  void uart_init(void);
```

```
26 void uart_tx(u8_t * buffer, u8_t length);
27
28 #endif /* UART_H_ */
```

D.1.3 Testsoftware ISM Transceiver Si4431

Listing D.27: main.c

```
/*-------------------------------------------------------------------------------
   Description: Main program for testing the Silicon Labs Si4431 ISM
                transceiver.
   Hardware:    Olimex MSP430-169STK
                BATSEN 434 MHz Transceiver-Board üfr die
                Basisstation v0.1 - Transceiver: Si4431
   Date:        10/02/2012
   Last Update: 11/02/2012
   Author:      Phillip Durdaut
-------------------------------------------------------------------------------*/

#include <msp430x16x.h>
#include <signal.h>
#include <stdio.h>
#include "si4431.h"
#include "lcd16x2.h"

// P1.7 <- Button 3 (Interrupt on falling edge)
// P3.6 -> LED1
// P3.7 -> LED2
// P4.x -> LCD

// Button 3 (Send data packet)
#define BUTTON3_DIR_IN  (P1DIR &= ~BIT7)
#define BUTTON3_FALLING_EDGE (P1IES |= BIT7)
#define BUTTON3_IRQ_ENABLE (P1IE |= BIT7)
#define BUTTON3_IRQ_DISABLE (P1IE &= ~BIT7)
#define BUTTON3_IRQ_PENDING ((P1IFG & BIT7) == BIT7)
#define BUTTON3_CLEAR_IRQ (P1IFG &= ~(BIT7))

// LEDs
#define PORT3_DIR (BIT7 | BIT6)
#define LED1_ON   (P3OUT &= ~BIT6)
#define LED1_OFF  (P3OUT |= BIT6)
#define LED1_TOGGLE (P3OUT ^= BIT6)
#define LED2_ON   (P3OUT &= ~BIT7)
#define LED2_OFF  (P3OUT |= BIT7)
#define LED2_TOGGLE (P3OUT ^= BIT7)

// LCD
#define PORT4_DIR (0xFF)
#define PORT4_OUT (0x00)

void osc_configuration(void);

//*********************************************************
interrupt (PORT1_VECTOR) isr_port1(void)
//*********************************************************
{
   // Si4431 interrupt
   if (SI4431_nIRQ_IRQ_PENDING) {

      // Read Si4431 interrupt status registers
      unsigned char status1 = 0;
      unsigned char status2 = 0;
      si4431_get_irq_status(&status1, &status2);

      // Packet received
      if ((status1 & SI4431_IRQ_PACKET_RECEIVED) == SI4431_IRQ_PACKET_RECEIVED) {

         // 5 bytes of data expected
         unsigned char buffer[5];
         unsigned char length = 5;

         si4431_read_rx_fifo(buffer, length);
```

```
66
67            unsigned char str[16];
68            sprintf(str, "received: %d%d%d%d%d ", buffer[0], buffer[1], buffer[2], buffer[3],
                  buffer[4]);
69
70            LCD_send_cmd(LCD_CLR);
71            LCD_send_cmd(LCD_LINE1);
72            LCD_send_text(str);
73            delay_ms(1500);
74            LCD_send_cmd(LCD_CLR);
75         }
76         else {
77            LCD_send_cmd(LCD_CLR);
78            LCD_send_cmd(LCD_LINE1);
79            LCD_send_unsigned_int((unsigned int)status1);
80            LCD_send_cmd(LCD_LINE2);
81            LCD_send_unsigned_int((unsigned int)status2);
82         }
83
84         si4431_rx_state();
85         SI4431_nIRQ_CLEAR_IRQ;
86      }
87
88      // Button 3 interrupt
89      if (BUTTON3_IRQ_PENDING) {
90
91         // Send 5 data bytes + 2 CRC bytes
92         unsigned char buffer[] = { 0x00, 0x01, 0x02, 0x03, 0x00, 0x3a, 0x28 };
93         unsigned char length = 7;
94
95         si4431_fill_tx_fifo(buffer, length);
96         si4431_tx_state();
97         si4431_idle_state_ready_mode();
98
99         LCD_send_cmd(LCD_CLR);
100        LCD_send_cmd(LCD_LINE1);
101        LCD_send_text("sent: 01230 ");
102        delay_ms(1500);
103        LCD_send_cmd(LCD_CLR);
104
105        si4431_rx_state();
106
107        BUTTON3_CLEAR_IRQ;
108     }
109  }
110  //************************************************************
111
112
113  //************************************************************
114  int main (void)
115  //************************************************************
116  {
117     //************************************************************
118     // OSC configuration & startup (8 MHz)
119     //************************************************************
120     osc_configuration();
121
122     //************************************************************
123     // Button configuration
124     //************************************************************
125     BUTTON3_DIR_IN;
126     BUTTON3_FALLING_EDGE;
127     BUTTON3_IRQ_ENABLE;
128     BUTTON3_CLEAR_IRQ;
129
130     //************************************************************
131     // Port configuration
```

```
132     //*************************************************************
133     P3DIR = PORT3_DIR;
134     P4DIR = PORT4_DIR;
135     P4OUT = PORT4_OUT;
136
137     LED1_OFF;
138     LED2_OFF;
139
140     //*************************************************************
141     // LCD configuration
142     //*************************************************************
143     LCD_init();
144     LCD_send_cmd(LCD_CLR);
145
146     //*************************************************************
147     // SI4431 configuration
148     //*************************************************************
149     si4431_init();
150     si4431_config();
151     si4431_rx_state();
152
153     //*************************************************************
154     // Global interrupt enable
155     //*************************************************************
156     _EINT();
157
158     //*************************************************************
159     // Endless loop
160     //*************************************************************
161     while(1);
162
163     return 0;
164 }
165 //*************************************************************
166
167 // Taken (modified) from "Diagnosefunktion üfr Automobil-
168 // Starterbatterien mit drahtlosen Zellensensoren" by
169 // Simon Puettjer p. 167
170 //*************************************************************
171 void osc_configuration(void)
172 //*************************************************************
173 {
174     unsigned int max_tries = 1000;
175     unsigned char i;
176
177     _BIC_SR(OSCOFF);            // Switch LFXT on
178     BCSCTL1 &= ~XT2OFF;         // XT2on
179     do {
180         IFG1 &= ~OFIFG;         // Clear OSCFault flag
181         for (i = 0xFF; i > 0; i--); // Time for flag to set
182         if (!(--max_tries)) {   // Stop after 1k tries, error with ext osc!
183         }
184     }
185     while ((IFG1 & OFIFG));     // OSCFault flag still set?
186     IFG1 &= ~OFIFG;             // Clear OSCFault flag
187
188     BCSCTL2 |= SELM_2 | SELS;  // MCLK = SMCLK = XT2 (safe)
189
190     // MCLK = 8 MHz
191     BCSCTL2 &= ~(DIVS_3 | DIVM_3); // SMCLK = MCLK = XT2 / 1
192 }
193 //*************************************************************
```

Listing D.28: delay.c

```
1 /*-------------------------------------------------------------------------------
```

```
2     Description: Delay functions.
3     Date:        10/25/2012
4     Last Update: 10/25/2012
5     Author:      Phillip Durdaut
6  ------------------------------------------------------------------------------*/
7
8  #include <msp430x16x.h>
9  #include "delay.h"
10
11 /*-----------------------------------------------------------------------------
12   Prototypes of the private functions
13 ------------------------------------------------------------------------------*/
14
15 void delay(unsigned int i);
16
17 /*-----------------------------------------------------------------------------
18   Public functions
19 ------------------------------------------------------------------------------*/
20
21 void delay_ms(unsigned int delay_ms)
22 {
23    while (delay_ms > 0) {
24       delay(DELAY_CYCLES_PER_MS);
25       delay_ms--;
26    }
27 }
28
29 void delay_100us(unsigned int delay_100us)
30 {
31    while (delay_100us > 0) {
32       delay(DELAY_CYCLES_PER_100US);
33       delay_100us--;
34    }
35 }
36
37 /*-----------------------------------------------------------------------------
38   Private functions
39 ------------------------------------------------------------------------------*/
40
41 void delay(unsigned int i)
42 {
43    while (i > 0) {
44       _NOP();
45       i--;
46    }
47 }
```

Listing D.29: delay.h

```
1  /*-----------------------------------------------------------------------------
2     Description: Delay functions.
3     Hardware:    Olimex MSP430-169STK
4     Date:        10/25/2012
5     Last Update: 10/25/2012
6     Author:      Phillip Durdaut
7  ------------------------------------------------------------------------------*/
8
9  #ifndef DELAY_H_
10 #define DELAY_H_
11
12 /*-----------------------------------------------------------------------------
13   Defines
14 ------------------------------------------------------------------------------*/
15
16 #define DELAY_CYCLES_PER_MS 610
17 #define DELAY_CYCLES_PER_100US 57
```

```
18
19 /*---------------------------------------------------------------------------
20  Public functions
21 ---------------------------------------------------------------------------*/
22
23 void delay_ms(unsigned int delay_ms);
24 void delay_100us(unsigned int delay_100us);
25
26 #endif /* DELAY_H_ */
```

Listing D.30: si4431.c

```
1  /*---------------------------------------------------------------------------
2     Description: Driver for Silicon Labs Si4431 ISM transceiver.
3     Date:       10/18/2012
4     Last Update: 10/22/2012
5     Author:     Phillip Durdaut
6  ---------------------------------------------------------------------------*/
7
8  #include "si4431.h"
9
10 /*---------------------------------------------------------------------------
11  Defines
12  ---------------------------------------------------------------------------*/
13
14 /* Configuration Modes (bit 7) */
15
16 #define SI4431_READ        0x00
17 #define SI4431_WRITE       0x80
18
19 /* Registers (bits 6-0) */
20
21 #define SI4431_DEVICETYPE                 0x00
22 #define SI4431_DEVICEVERSION              0x01
23 #define SI4431_DEVICESTATUS               0x02
24 #define SI4431_INTERRUPTSTATUS1           0x03
25 #define SI4431_INTERRUPTSTATUS2           0x04
26 #define SI4431_INTERRUPTENABLE1           0x05
27 #define SI4431_INTERRUPTENABLE2           0x06
28 #define SI4431_OPERATINGANDFUNCTIONCONTROL1 0x07
29 #define SI4431_OPERATINGANDFUNCTIONCONTROL2 0x08
30 #define SI4431_CRYSTALOSCILLATORLOADCAPACITANCE 0x09
31 #define SI4431_MICROCONTROLLEROUTPUTCLOCK 0x0A
32 #define SI4431_GPIO0CONFIGURATION         0x0B
33 #define SI4431_GPIO1CONFIGURATION         0x0C
34 #define SI4431_GPIO2CONFIGURATION         0x0D
35 #define SI4431_IOPORTCONFIGURATION        0x0E
36 #define SI4431_ADCCONFIGURATION           0x0F
37 #define SI4431_ADCSENSORAMPLIFIEROFFSET   0x10
38 #define SI4431_ADCVALUE                   0x11
39 #define SI4431_TEMPERATURESENSORCONTROL   0x12
40 #define SI4431_TEMPERATUREVALUEOFFSET     0x13
41 #define SI4431_WAKEUPTIMERPERIOD1         0x14
42 #define SI4431_WAKEUPTIMERPERIOD2         0x15
43 #define SI4431_WAKEUPTIMERPERIOD3         0x16
44 #define SI4431_WAKEUPTIMERVALUE1          0x17
45 #define SI4431_WAKEUPTIMERVALUE2          0x18
46 #define SI4431_LOWDUTYCYCLEMODEDURATION   0x19
47 #define SI4431_LOWBATTERYDETECTORTHRESHOLD 0x1A
48 #define SI4431_BATTERYVOLTAGELEVEL        0x1B
49 #define SI4431_IFFILTERBANDWIDTH          0x1C
50 #define SI4431_AFCLOOPGEARSHIFTOVERRIDE   0x1D
51 #define SI4431_AFCTIMINGCONTROL           0x1E
52 #define SI4431_CLOCKRECOVERYGEARSHIFTOVERRIDE 0x1F
53 #define SI4431_CLOCKRECOVERYOVERSAMPLINGRATIO 0x20
54 #define SI4431_CLOCKRECOVERYOFFSET2       0x21
```

```
55  #define SI4431_CLOCKRECOVERYOFFSET1       0x22
56  #define SI4431_CLOCKRECOVERYOFFSET0       0x23
57  #define SI4431_CLOCKRECOVERYTIMINGLOOPGAIN1 0x24
58  #define SI4431_CLOCKRECOVERYTIMINGLOOPGAIN0 0x25
59  #define SI4431_RECEIVEDSIGNALSTRENGTHINDICATOR 0x26
60  #define SI4431_RSSITHRESHOLDFORCLEARCHANNELINDICATOR 0x27
61  #define SI4431_ANTENNADIVERSITYREGISTER1  0x28
62  #define SI4431_ANTENNADIVERSITYREGISTER2  0x29
63  #define SI4431_AFCLIMITER                 0x2A
64  #define SI4431_AFCCORRECTIONREAD          0x2B
65  #define SI4431_OOKCOUNTERVALUE1           0x2C
66  #define SI4431_OOKCOUNTERVALUE2           0x2D
67  #define SI4431_SLICERPEAKHOLD             0x2E
68  #define SI4431_DATAACCESSCONTROL          0x30
69  #define SI4431_EZMACSTATUS                0x31
70  #define SI4431_HEADERCONTROL1             0x32
71  #define SI4431_HEADERCONTROL2             0x33
72  #define SI4431_PREAMBLELENGTH             0x34
73  #define SI4431_PREAMBLEDETECTIONCONTROL   0x35
74  #define SI4431_SYNCWORD3                  0x36
75  #define SI4431_SYNCWORD2                  0x37
76  #define SI4431_SYNCWORD1                  0x38
77  #define SI4431_SYNCWORD0                  0x39
78  #define SI4431_TRANSMITHEADER3            0x3A
79  #define SI4431_TRANSMITHEADER2            0x3B
80  #define SI4431_TRANSMITHEADER1            0x3C
81  #define SI4431_TRANSMITHEADER0            0x3D
82  #define SI4431_PACKETLENGTH               0x3E
83  #define SI4431_CHECKHEADER3               0x3F
84  #define SI4431_CHECKHEADER2               0x40
85  #define SI4431_CHECKHEADER1               0x41
86  #define SI4431_CHECKHEADER0               0x42
87  #define SI4431_HEADERENABLE3              0x43
88  #define SI4431_HEADERENABLE2              0x44
89  #define SI4431_HEADERENABLE1              0x45
90  #define SI4431_HEADERENABLE0              0x46
91  #define SI4431_RECEIVEDHEADER3            0x47
92  #define SI4431_RECEIVEDHEADER2            0x48
93  #define SI4431_RECEIVEDHEADER1            0x49
94  #define SI4431_RECEIVEDHEADER0            0x4A
95  #define SI4431_RECEIVEDPACKETLENGTH       0x4B
96  #define SI4431_ADC8CONTROL                0x4F
97  #define SI4431_CHANNELFILTERCOEFFICIENTADDRESS 0x60
98  #define SI4431_CRYSTALOSCILLATORCONTROLTEST 0x62
99  #define SI4431_AGCOVERRIDE1               0x69
100 #define SI4431_TXPOWER                    0x6D
101 #define SI4431_TXDATARATE1                0x6E
102 #define SI4431_TXDATARATE0                0x6F
103 #define SI4431_MODULATIONMODECONTROL1     0x70
104 #define SI4431_MODULATIONMODECONTROL2     0x71
105 #define SI4431_FREQUENCYDEVIATION         0x72
106 #define SI4431_FREQUENCYOFFSET1           0x73
107 #define SI4431_FREQUENCYOFFSET2           0x74
108 #define SI4431_FREQUENCYBANDSELECT        0x75
109 #define SI4431_NOMINALCARRIERFREQUENCY1   0x76
110 #define SI4431_NOMINALCARRIERFREQUENCY0   0x77
111 #define SI4431_FREQUENCYHOPPINGCHANNELSELECT 0x79
112 #define SI4431_FREQUENCYHOPPINGSTEPSIZE   0x7A
113 #define SI4431_TXFIFOCONTROL1             0x7C
114 #define SI4431_TXFIFOCONTROL2             0x7D
115 #define SI4431_RXFIFOCONTROL              0x7E
116 #define SI4431_FIFOACCESS                 0x7F
117
118 /*---------------------------------------------------------------------------
119   Prototypes of the private functions
120 ---------------------------------------------------------------------------*/
121
```

```
122 void si4431_spi_setup(void);
123 void si4431_spi_write_register(unsigned char address, unsigned char value);
124 unsigned char si4431_spi_read_register(unsigned char address);
125
126 /*---------------------------------------------------------------------------
127  Public functions
128 ---------------------------------------------------------------------------*/
129
130 void si4431_init(void)
131 {
132    SI4431_SDN_PxDIR |= SI4431_SDN_PIN; // SDN is output
133    SI4431_nIRQ_PxDIR &= ~SI4431_nIRQ_PIN; // nIRQ is input
134    SI4431_nIRQ_PxIES |= SI4431_nIRQ_PIN; // Interrupt on falling edge
135    SI4431_nIRQ_PxIE &= ~SI4431_nIRQ_PIN; // Interrupt disable
136
137    si4431_power_off();
138    si4431_spi_setup();
139    si4431_power_on();
140
141    delay_ms(25); // Wait at least 15 ms after POR
142
143    // Clear the interrupt flags and release nIRQ pin
144    si4431_clear_all_irqs();
145
146    // Crystal Oscillator Load Capacitance: 9.972 pF
147    si4431_spi_write_register(SI4431_CRYSTALOSCILLATORLOADCAPACITANCE, 0x61);
148 }
149
150 void si4431_power_on(void)
151 {
152    SI4431_SDN_PxOUT &= ~SI4431_SDN_PIN;
153 }
154
155 void si4431_power_off(void)
156 {
157    SI4431_SDN_PxOUT |= SI4431_SDN_PIN;
158 }
159
160 void si4431_reset(void)
161 {
162    si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x80);
163
164    // Wait for chip ready
165    while((SI4431_nIRQ_PxIN & SI4431_nIRQ_PIN) == SI4431_nIRQ_PxIN);
166
167    // Clear the interrupt flags and release nIRQ pin
168    si4431_clear_all_irqs();
169 }
170
171 unsigned char si4431_get_device_type(void)
172 {
173    return si4431_spi_read_register(SI4431_DEVICETYPE);
174 }
175
176 unsigned char si4431_get_device_version(void)
177 {
178    return si4431_spi_read_register(SI4431_DEVICEVERSION);
179 }
180
181 unsigned char si4431_get_device_status(void)
182 {
183    return si4431_spi_read_register(SI4431_DEVICESTATUS);
184 }
185
186 void si4431_clear_all_irqs(void)
187 {
188    unsigned char status1;
```

```
189     unsigned char status2;
190     si4431_get_irq_status(&status1, &status2);
191  }
192
193  void si4431_get_irq_status(unsigned char * status1, unsigned char * status2)
194  {
195     *status1 = si4431_spi_read_register(SI4431_INTERRUPTSTATUS1);
196     *status2 = si4431_spi_read_register(SI4431_INTERRUPTSTATUS2);
197  }
198
199  void si4431_set_irq_enable(unsigned char enable1, unsigned char enable2)
200  {
201     si4431_spi_write_register(SI4431_INTERRUPTENABLE1, enable1);
202     si4431_spi_write_register(SI4431_INTERRUPTENABLE2, enable2);
203     SI4431_nIRQ_PxIE |= SI4431_nIRQ_PIN; // Interrupt enable
204  }
205
206  void si4431_set_irq_disable(void)
207  {
208     si4431_spi_write_register(SI4431_INTERRUPTENABLE1, SI4431_IRQ_DISABLE_ALL);
209     si4431_spi_write_register(SI4431_INTERRUPTENABLE2, SI4431_IRQ_DISABLE_ALL);
210     SI4431_nIRQ_PxIE &= ~SI4431_nIRQ_PIN; // Interrupt disable
211  }
212
213  void si4431_idle_state_standby_mode(void)
214  {
215     // Clear and disable all interupts
216     si4431_clear_all_irqs();
217     si4431_set_irq_disable();
218
219     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x00);
220  }
221
222  void si4431_idle_state_ready_mode(void)
223  {
224     // Clear and disable all interupts
225     si4431_clear_all_irqs();
226     si4431_set_irq_disable();
227
228     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x01);
229  }
230
231  void si4431_idle_state_tune_mode(void)
232  {
233     // Disable and clear all interupts
234     si4431_set_irq_disable();
235     si4431_clear_all_irqs();
236
237     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x03);
238  }
239
240  void si4431_tx_state(void)
241  {
242     // 5 data bytes + 2 CRC bytes (later 10 + 2)
243     si4431_spi_write_register(SI4431_PACKETLENGTH, 0x07);
244
245     // Disable CRC calculation when transmitting data as this was done manually
246     si4431_disable_crc();
247
248     // Disable all interrupts and clear current interrupts
249     si4431_set_irq_enable(SI4431_IRQ_DISABLE_ALL, SI4431_IRQ_DISABLE_ALL);
250     si4431_clear_all_irqs();
251
252     // TX state
253     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x09);
254
255     // Wait 5ms for TX finished
```

```
256     delay_ms(5);
257  }
258
259  void si4431_rx_state(void)
260  {
261     // 5 data bytes
262     si4431_spi_write_register(SI4431_PACKETLENGTH, 0x05);
263
264     // Enable CRC check when receiving data
265     si4431_enable_crc();
266
267     // Enable and clear interrupts
268     si4431_clear_all_irqs();
269     si4431_set_irq_enable(SI4431_IRQ_PACKET_RECEIVED | SI4431_IRQ_CRC_ERROR,
            SI4431_IRQ_DISABLE_ALL);
270
271     // Clear RX FIFO
272     si4431_clear_rx_fifo();
273
274     // RX state
275     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL1, 0x04);
276  }
277
278  void si4431_config(void)
279  {
280     /*************** Frequency Control ******************
281
282           Operating Frequency: 434,000 MHz
283      ******************************************************/
284
285     // No frequency offset
286     si4431_spi_write_register(SI4431_FREQUENCYOFFSET1, 0x00);
287     si4431_spi_write_register(SI4431_FREQUENCYOFFSET2, 0x00);
288
289     // 434 MHz
290     si4431_spi_write_register(SI4431_FREQUENCYBANDSELECT, 0x53);
291     si4431_spi_write_register(SI4431_NOMINALCARRIERFREQUENCY1, 0x64);
292     si4431_spi_write_register(SI4431_NOMINALCARRIERFREQUENCY0, 0x00);
293
294     // No frequency hopping
295     si4431_spi_write_register(SI4431_FREQUENCYHOPPINGSTEPSIZE, 0x00);
296     si4431_spi_write_register(SI4431_FREQUENCYHOPPINGCHANNELSELECT, 0x00);
297
298     /*************** TX Modulation Options ***************/
299
300     // Output power: +13 dBm
301     si4431_spi_write_register(SI4431_TXPOWER, 0x1F);
302
303     // Data rate: 39.993 kbit/s
304     si4431_spi_write_register(SI4431_TXDATARATE1, 0x0A);
305     si4431_spi_write_register(SI4431_TXDATARATE0, 0x3D);
306
307     // No Manchester Coding
308     si4431_spi_write_register(SI4431_MODULATIONMODECONTROL1, 0x00);
309
310     // No TX data clock, FIFO Mode, OOK
311     si4431_spi_write_register(SI4431_MODULATIONMODECONTROL2, 0x21);
312
313     /*************** RX Modem settings ******************/
314
315     // IF Filter bandwidth: 37.7 kHz
316     si4431_spi_write_register(SI4431_IFFILTERBANDWIDTH, 0x11);
317
318     // Settings generated with Silicon Labs Wireless Development Suite
319     si4431_spi_write_register(SI4431_AFCLOOPGEARSHIFTOVERRIDE, 0x3C);
320     si4431_spi_write_register(SI4431_AFCTIMINGCONTROL, 0x02);
321     si4431_spi_write_register(SI4431_CLOCKRECOVERYGEARSHIFTOVERRIDE, 0x03);
```

```
322     si4431_spi_write_register(SI4431_CLOCKRECOVERYOVERSAMPLINGRATIO, 0x32);
323     si4431_spi_write_register(SI4431_CLOCKRECOVERYOFFSET2, 0x02);
324     si4431_spi_write_register(SI4431_CLOCKRECOVERYOFFSET1, 0x8F);
325     si4431_spi_write_register(SI4431_CLOCKRECOVERYOFFSET0, 0x3F);
326     si4431_spi_write_register(SI4431_CLOCKRECOVERYTIMINGLOOPGAIN1, 0x02);
327     si4431_spi_write_register(SI4431_CLOCKRECOVERYTIMINGLOOPGAIN0, 0x91);
328     si4431_spi_write_register(SI4431_OOKCOUNTERVALUE1, 0x28);
329     si4431_spi_write_register(SI4431_OOKCOUNTERVALUE2, 0x1F);
330     si4431_spi_write_register(SI4431_SLICERPEAKHOLD, 0x27);
331     si4431_spi_write_register(SI4431_AGCOVERRIDE1, 0x60);
332
333     /***************** Packet handling ********************/
334
335     // Enable RX, TX packet handling, MSB first, CRC over DATA, CRC disable, CRC-16 (IBM)
336     si4431_spi_write_register(SI4431_DATAACCESSCONTROL, 0xA9);
337
338     // No broadcast address enable, No received header check
339     si4431_spi_write_register(SI4431_HEADERCONTROL1, 0x00);
340
341     // No TX/RX header, No packet length included, 4 Sync bytes (3, 2, 1, 0)
342     si4431_spi_write_register(SI4431_HEADERCONTROL2, 0x0E);
343
344     // 8 Preamble nibbles = 32 bits = 4 bytes
345     si4431_spi_write_register(SI4431_PREAMBLELENGTH, 0x08);
346
347     // Preamble detection threshold: 6 nibbles, no RSSI offset
348     si4431_spi_write_register(SI4431_PREAMBLEDETECTIONCONTROL, 0x30);
349
350     // Sync word: 0x12 0x09 0x12 0x09
351     si4431_spi_write_register(SI4431_SYNCWORD3, 0x12);
352     si4431_spi_write_register(SI4431_SYNCWORD2, 0x09);
353     si4431_spi_write_register(SI4431_SYNCWORD1, 0x12);
354     si4431_spi_write_register(SI4431_SYNCWORD0, 0x09);
355 }
356
357 void si4431_enable_crc(void)
358 {
359     unsigned char current = si4431_spi_read_register(SI4431_DATAACCESSCONTROL);
360     si4431_spi_write_register(SI4431_DATAACCESSCONTROL, current | BIT2);
361 }
362
363 void si4431_disable_crc(void)
364 {
365     unsigned char current = si4431_spi_read_register(SI4431_DATAACCESSCONTROL);
366     si4431_spi_write_register(SI4431_DATAACCESSCONTROL, current & ~BIT2);
367 }
368
369 void si4431_clear_tx_fifo(void)
370 {
371     unsigned char current = si4431_spi_read_register(SI4431_OPERATINGANDFUNCTIONCONTROL2);
372     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL2, current | BIT0);
373     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL2, current & ~BIT0);
374 }
375
376 void si4431_clear_rx_fifo(void)
377 {
378     unsigned char current = si4431_spi_read_register(SI4431_OPERATINGANDFUNCTIONCONTROL2);
379     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL2, current | BIT1);
380     si4431_spi_write_register(SI4431_OPERATINGANDFUNCTIONCONTROL2, current & ~BIT1);
381 }
382
383 void si4431_fill_tx_fifo(unsigned char * buffer, unsigned char length)
384 {
385     unsigned char i;
386
387     si4431_clear_tx_fifo(); // Clear TX FIFO
388
```

```
389     for (i = 0; i < length; i++) {
390
391        // Transfer the bytes via SPI to the TX FIFO of the transceiver
392        si4431_spi_write_register(SI4431_FIFOACCESS, buffer[i]);
393     }
394  }
395
396  void si4431_read_rx_fifo(unsigned char * buffer, unsigned char length)
397  {
398     unsigned char i;
399
400     si4431_idle_state_ready_mode(); // Disable the receiver
401
402     for (i = 0; i < length; i++) {
403
404        // Transfer the bytes via SPI from the RX FIFO of the transceiver
405        buffer[i] = si4431_spi_read_register(SI4431_FIFOACCESS);
406     }
407  }
408
409  unsigned char si4431_get_rssi()
410  {
411     return si4431_spi_read_register(SI4431_RECEIVEDSIGNALSTRENGTHINDICATOR);
412  }
413
414  /*---------------------------------------------------------------------------
415    Private functions
416  ---------------------------------------------------------------------------*/
417
418  void si4431_spi_setup(void)
419  {
420     SI4431_nSEL_PxDIR |= SI4431_nSEL_PIN; // nSEL is output
421     SI4431_nSEL_PxOUT |= SI4431_nSEL_PIN; // Chip disable
422
423     ME1 |= USPIE0;                    // Enable USART0 SPI
424     U0CTL = SWRST;                    // USART logic held in reset state
425     U0CTL |= (CHAR | SYNC | MM);      // 8-bit, SPI, Master
426     U0TCTL |= (CKPH | SSEL1 | SSEL0 | STC); // CLK delayed, 3-pin SPI mode
427
428     U0BR0 = 2; // BRCLK/2
429     U0BR1 = 0;
430     U0MCTL = 0; // Always 0 in SPI mode
431
432     // SPI functionality for pins
433     SI4431_SPI_PxSEL |= (SI4431_SPI_SDI_PIN | SI4431_SPI_SDO_PIN | SI4431_SPI_SCLK_PIN);
434     SI4431_SPI_PxDIR |= (SI4431_SPI_SDI_PIN | SI4431_SPI_SCLK_PIN); // MOSI and CLK are outputs
435
436     UCTL0 &= ~SWRST; // Initialize USART state machine
437  }
438
439  void si4431_spi_write_register(unsigned char address, unsigned char value)
440  {
441     SI4431_nSEL_PxOUT &= ~SI4431_nSEL_PIN; // Chip enable
442     while (!(IFG1&UTXIFG0));          // Wait for TX to finish
443     U0TXBUF = SI4431_WRITE | address; // Send configuration mode and register
444     while (!(IFG1&UTXIFG0));          // Wait for TX to finish
445     U0TXBUF = value;                  // Send value
446     while(!(UTCTL0&TXEPT));           // Wait for TX complete
447     SI4431_nSEL_PxOUT |= SI4431_nSEL_PIN; // Chip disable
448  }
449
450  unsigned char si4431_spi_read_register(unsigned char address)
451  {
452     unsigned char value;
453
454     SI4431_nSEL_PxOUT &= ~SI4431_nSEL_PIN; // Chip enable
455     while (!(IFG1 & UTXIFG0));        // Wait for TX to finish
```

```
456     U0TXBUF = SI4431_READ | address; // Send configuration mode and register
457     while (!(IFG1 & UTXIFG0));      // Wait for TX to finish
458     U0TXBUF = 0;                    // Dummy write so we can read data
459     while(!(UTCTL0 & TXEPT));       // Wait for TX complete
460     value = U0RXBUF;                // Read data
461     SI4431_nSEL_PxOUT |= SI4431_nSEL_PIN; // Chip disable
462
463     return value;
464 }
```

Listing D.31: si4431.h

```
1  /*-------------------------------------------------------------------------------
2     Description: Driver for Silicon Labs Si4431 ISM transceiver.
3     Date:       10/18/2012
4     Last Update: 10/22/2012
5     Author:     Phillip Durdaut
6  -------------------------------------------------------------------------------*/
7
8  #ifndef SI4431_H_
9  #define SI4431_H_
10
11 #include <msp430x16x.h>
12 #include "delay.h"
13
14 /*-------------------------------------------------------------------------------
15   Defines -> Need to be changed depending on hardware
16  -------------------------------------------------------------------------------*/
17
18 #define SI4431_nSEL_PxDIR  P3DIR
19 #define SI4431_nSEL_PxOUT  P3OUT
20 #define SI4431_nSEL_PIN    BIT0
21
22 #define SI4431_SPI_PxSEL   P3SEL
23 #define SI4431_SPI_PxDIR   P3DIR
24 #define SI4431_SPI_PxIN    P3IN
25 #define SI4431_SPI_SDI_PIN BIT1
26 #define SI4431_SPI_SDO_PIN BIT2
27 #define SI4431_SPI_SCLK_PIN BIT3
28
29 #define SI4431_nIRQ_PxDIR  P1DIR
30 #define SI4431_nIRQ_PxIN   P1IN
31 #define SI4431_nIRQ_PIN    BIT3
32 #define SI4431_nIRQ_PxIES  P1IES
33 #define SI4431_nIRQ_PxIE   P1IE
34 #define SI4431_nIRQ_IRQ_PENDING ((P1IFG & BIT3) == BIT3)
35 #define SI4431_nIRQ_CLEAR_IRQ (P1IFG &= ~(BIT3))
36
37 #define SI4431_SDN_PxDIR   P1DIR
38 #define SI4431_SDN_PxOUT   P1OUT
39 #define SI4431_SDN_PIN     BIT4
40
41 /*-------------------------------------------------------------------------------
42   Defines
43  -------------------------------------------------------------------------------*/
44
45 #define SI4431_IRQ_DISABLE_ALL 0x00
46 #define SI4431_IRQ_CRC_ERROR 0x01
47 #define SI4431_IRQ_PACKET_RECEIVED 0x02
48 #define SI4431_IRQ_PACKET_SENT 0x04
49
50 /*-------------------------------------------------------------------------------
51   Public functions
52 -------------------------------------------------------------------------------*/
53
54 void si4431_init(void);
```

```
55 void si4431_power_on(void);
56 void si4431_power_off(void);
57 void si4431_reset(void);
58 unsigned char si4431_get_device_type(void);
59 unsigned char si4431_get_device_version(void);
60 unsigned char si4431_get_device_status(void);
61 void si4431_clear_all_irqs(void);
62 void si4431_get_irq_status(unsigned char * status1, unsigned char * status2);
63 void si4431_set_irq_enable(unsigned char enable1, unsigned char enable2);
64 void si4431_set_irq_disable(void);
65 void si4431_idle_state_standby_mode(void);
66 void si4431_idle_state_ready_mode(void);
67 void si4431_idle_state_tune_mode(void);
68 void si4431_tx_state(void);
69 void si4431_rx_state(void);
70 void si4431_config(void);
71 void si4431_enable_crc(void);
72 void si4431_disable_crc(void);
73 void si4431_clear_tx_fifo(void);
74 void si4431_clear_rx_fifo(void);
75 void si4431_fill_tx_fifo(unsigned char * buffer, unsigned char length);
76 void si4431_read_rx_fifo(unsigned char * buffer, unsigned char length);
77 unsigned char si4431_get_rssi();
78
79 #endif /* SI4431_H_ */
```

D.2 PC

D.2.1 IBM-CRC-16 Berechnung

Listing D.32: C-Programmcode zur Berechnung einer IBM-CRC-16 Checksumme.

```
/*---------------------------------------------------------------------------
   Description:   CRC-16 (IBM) checksum calculation.
   Platform:      x86, x64
   Date:          10/18/2012
   Last Update:   10/22/2012
   Author:        Phillip Durdaut
---------------------------------------------------------------------------*/
#include <stdio.h>
#include "types.h"

#define CRC16IBM 0x8005
#define LFSR_INIT 0xFFFF

u16 calc_crc(u8 * data, u8 length, u16 reg);

int main()
{
   u8 data[] = { 0, 0x01, 0x02, 0x03, 0 };
   u8 length = 5;

   u16 checksum = calc_crc(data, length, LFSR_INIT);
   printf("0x%04x\n", checksum & 0xFFFF);

   /* wait with exiting the program */
   getchar();

   return 0;
}

u16 calc_crc(u8 * data, u8 length, u16 reg) {

   u8 i, j;

   for (i = 0; i < length; i++) {

      for (j = 0; j < 8; j++) {

         if (((reg & 0x8000) >> 8) ^ (data[i] & 0x80))
            reg = (reg << 1) ^ CRC16IBM;
         else
            reg = (reg << 1);

         data[i] <<= 1;
      }
   }

   return reg;
}
```

D.2.2 Auswertung der Zellensensor-Messdaten

Listing D.33: Matlab-Programmcode zur Auswertung der Zellensensor-Messdaten

```
%--------------------------------------------------------------------------
%   Description:   Analysis of the base stations measurement data
%                  (4 cell nodes fixed in this version).
%   Platform:      x86, x64
%   Date:          12/05/2012
%   Last Update:   12/13/2012
%   Author:        Phillip Durdaut
%--------------------------------------------------------------------------
clear all;
clc;

% Konstanten
SAMPLE_RATE_SCOPE = 1000; % Samplerate Oszilloskop in Samples/s

% Daten des Oszilloskops einlesen
SCOPE = csvread('scope.csv');
SCOPE_X = SCOPE(:,1);
SCOPE_CELL_1 = SCOPE(:,2);
SCOPE_CELL_2 = SCOPE(:,3);
SCOPE_CELL_3 = SCOPE(:,4);
SCOPE_CELL_4 = SCOPE(:,5);

% ADC-Daten der Basisstation einlesen
ADC_HEX = textread('log.txt','%3c');
ADC_HEX = hex2dec(char(ADC_HEX));
ADC_HEX = reshape(ADC_HEX, 5, [])';

% Umrechnung in Spannung (2: Spannungsteiler)
ADC_V = ADC_HEX / 4095 * 2.5 * 2;
ADC_V(:,1) = ADC_HEX(:,1);

ADC_X = ADC_V(:,1);
ADC_X = ADC_X';
ADC_CELL_1 = ADC_V(:,2);
ADC_CELL_2 = ADC_V(:,3);
ADC_CELL_3 = ADC_V(:,4);
ADC_CELL_4 = ADC_V(:,5);

% Die 4 Zellspannungen aus dem Generator mit überlagerten Messwerten
figure(1);
subplot(4,1,1);
plot(SCOPE_X, SCOPE_CELL_1, 'b');
hold on;
plot(ADC_X, ADC_CELL_1, 'r-*');
grid on;
xlim([0 length(ADC_X)-1]);
xlabel('Zeit in s');
ylabel('Zellenspannung in V');

subplot(4,1,2);
plot(SCOPE_X, SCOPE_CELL_2, 'b');
hold on;
plot(ADC_X, ADC_CELL_2, 'r-*');
grid on;
xlim([0 length(ADC_X)-1]);
xlabel('Zeit in s');
ylabel('Zellenspannung in V');

subplot(4,1,3);
plot(SCOPE_X, SCOPE_CELL_3, 'b');
hold on;
plot(ADC_X, ADC_CELL_3, 'r-*');
grid on;
xlim([0 length(ADC_X)-1]);
xlabel('Zeit in s');
```

```
66  ylabel('Zellenspannung in V');
67
68  subplot(4,1,4);
69  plot(SCOPE_X, SCOPE_CELL_4, 'b');
70  hold on;
71  plot(ADC_X, ADC_CELL_4, 'r-*');
72  grid on;
73  xlim([0 length(ADC_X)-1]);
74  xlabel('Zeit in s');
75  ylabel('Zellenspannung in V');
76
77  % Messfehler berechnen
78  SCOPE_SEC = zeros(61,4);
79  DIFF_SEC_MV = zeros(61,4);
80
81  for i = 1 : 1 : length(ADC_X)
82     SCOPE_SEC(i,1) = SCOPE_CELL_1(((i-1)*SAMPLE_RATE_SCOPE)+1,1);
83     SCOPE_SEC(i,2) = SCOPE_CELL_2(((i-1)*SAMPLE_RATE_SCOPE)+1,1);
84     SCOPE_SEC(i,3) = SCOPE_CELL_3(((i-1)*SAMPLE_RATE_SCOPE)+1,1);
85     SCOPE_SEC(i,4) = SCOPE_CELL_4(((i-1)*SAMPLE_RATE_SCOPE)+1,1);
86
87     DIFF_SEC_MV(i,1) = (ADC_CELL_1(i,1) - SCOPE_SEC(i,1)) .* 1000;
88     DIFF_SEC_MV(i,2) = (ADC_CELL_2(i,1) - SCOPE_SEC(i,2)) .* 1000;
89     DIFF_SEC_MV(i,3) = (ADC_CELL_3(i,1) - SCOPE_SEC(i,3)) .* 1000;
90     DIFF_SEC_MV(i,4) = (ADC_CELL_4(i,1) - SCOPE_SEC(i,4)) .* 1000;
91  end
92
93  % Messfehler plotten
94  figure(2);
95  plot(ADC_X, DIFF_SEC_MV(:,1), ADC_X, DIFF_SEC_MV(:,2), ADC_X, DIFF_SEC_MV(:,3), ADC_X,
        DIFF_SEC_MV(:,4));
96  legend('Zellensensor 1', 'Zellensensor 2', 'Zellensensor 3', 'Zellensensor 4');
97  grid on;
98  xlim([0 length(ADC_X)-1]);
99  xlabel('Zeit in s');
100 ylabel('Messfehler in mV');
```

E Verwendete Messgeräte

- Rohde & Schwarz FSC3 Spectrum Analyzer $9\,\mathrm{kHz} \ldots 3\,\mathrm{GHz}$
- Tektronix MSO 3034 Mixed Signal Oscilloscope
- Rohde & Schwarz ZVR Vector Network Analyzer
- Rohde & Schwarz DC Power Supply NGT 20
- Fluke 45 Dual Display Multimeter

F Fotos von Messaufbauten

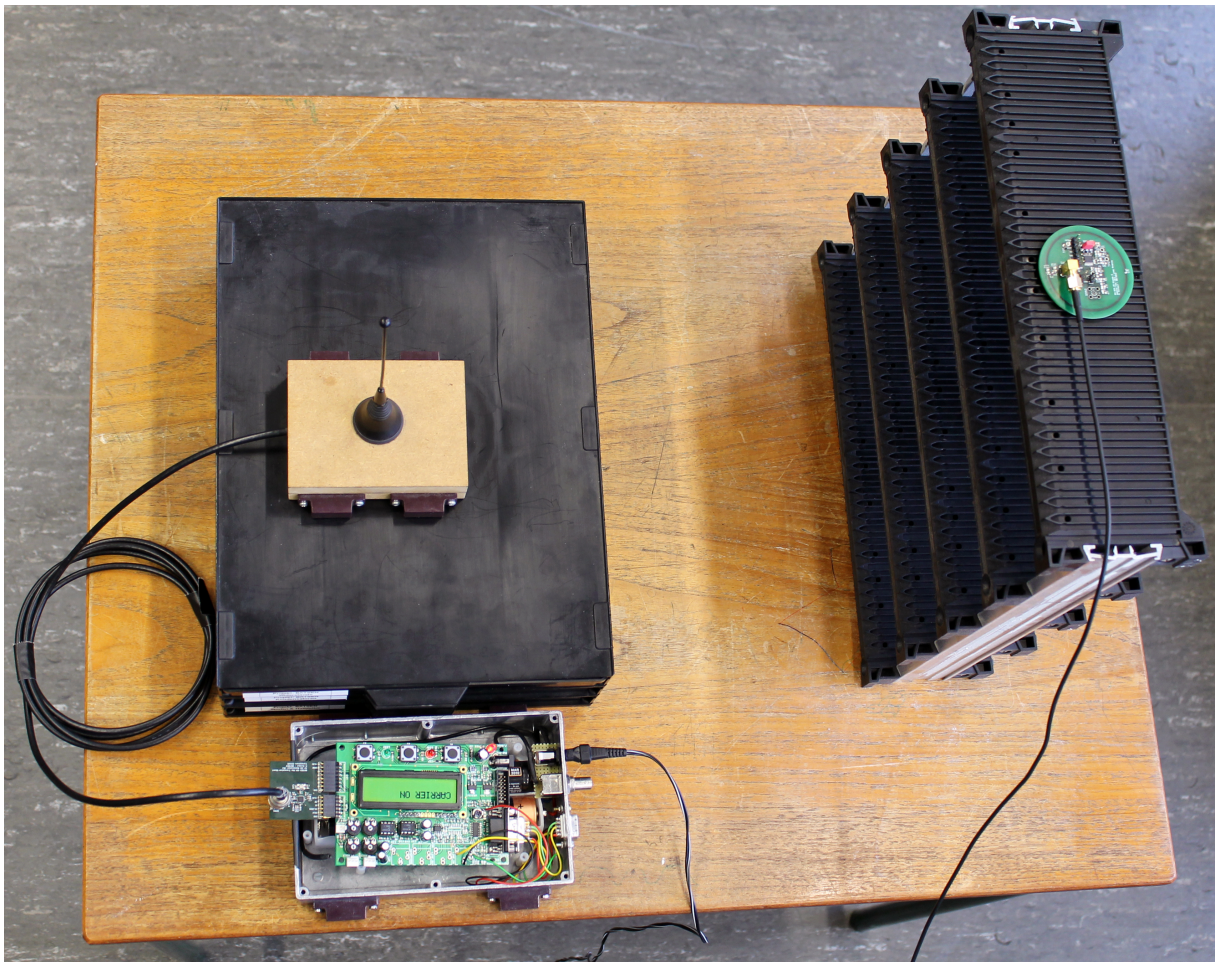


Abbildung F.1: Foto zum Versuchsaufbau zur Erprobungs- und Vergleichsmessungen an den Antennen

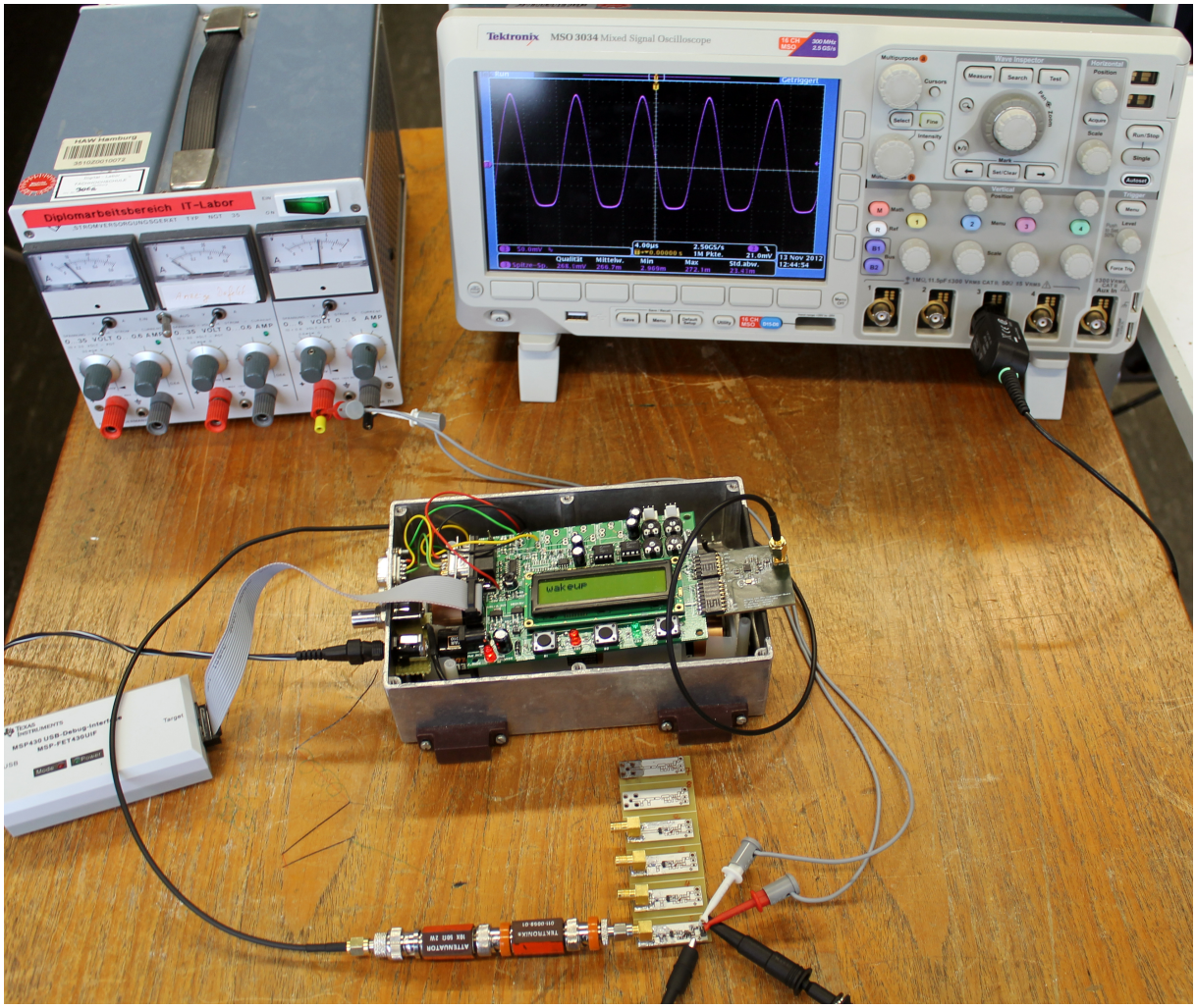


Abbildung F.2: Foto zum Versuchsaufbau zur Erprobung der Hüllkurvendemodulator-schaltungen

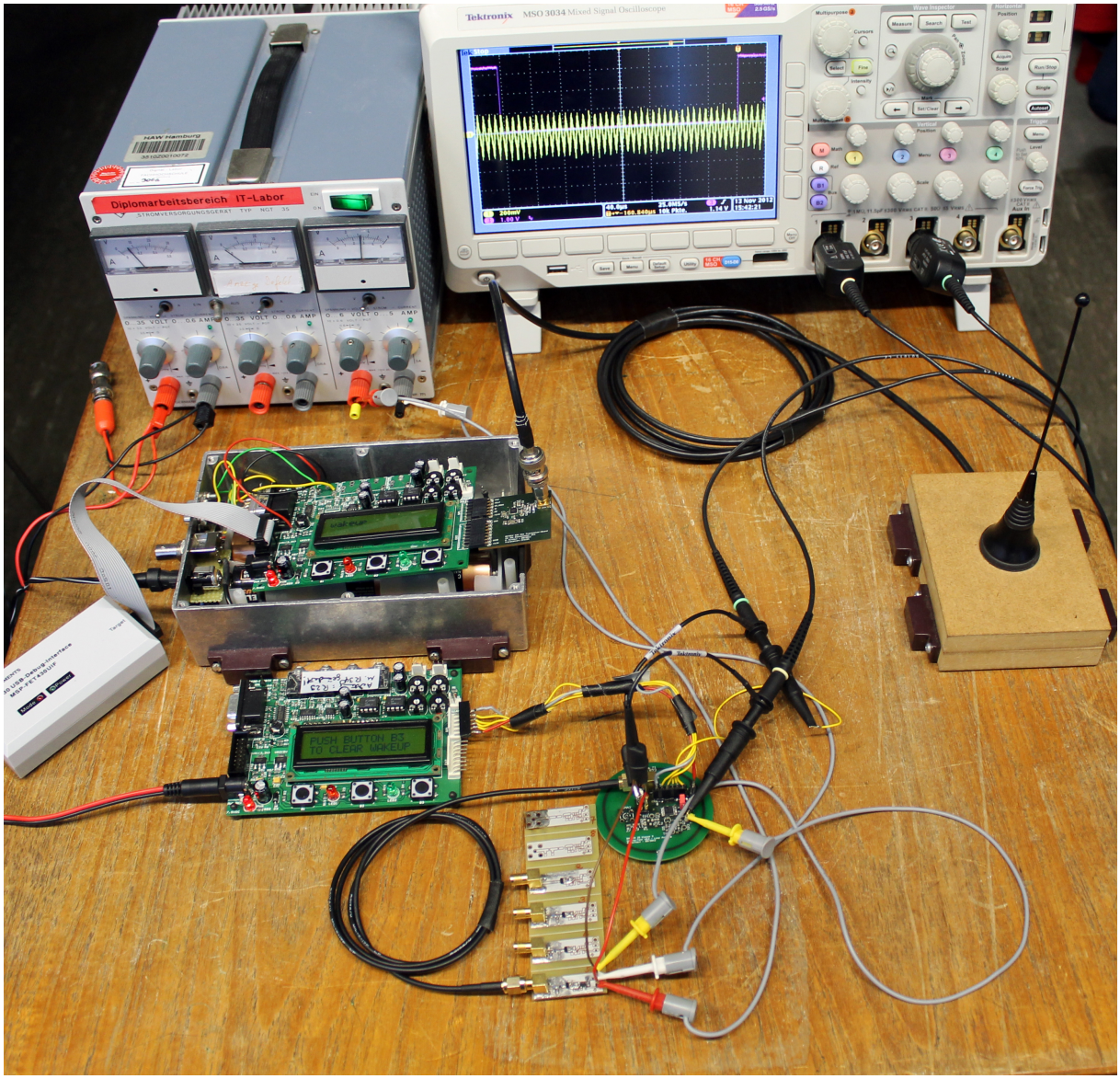


Abbildung F.3: Foto zum Versuchsaufbau zur Erprobung des Wakeups

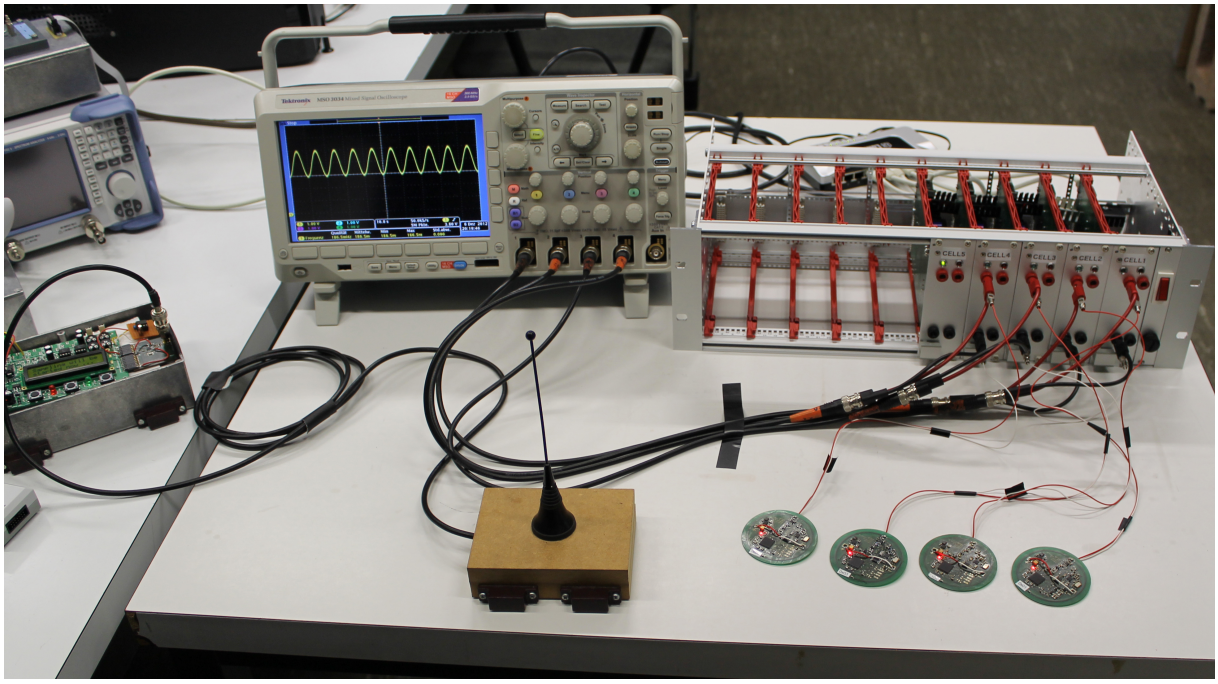


Abbildung F.4: Foto zum Versuchsaufbau zur Erprobung des Gesamtsystems bzw. zum Nachweis der grundsätzlichen Funktionsfähigkeit der Zellensensoren

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

Hamburg, 8. Februar 2013

Ort, Datum Unterschrift