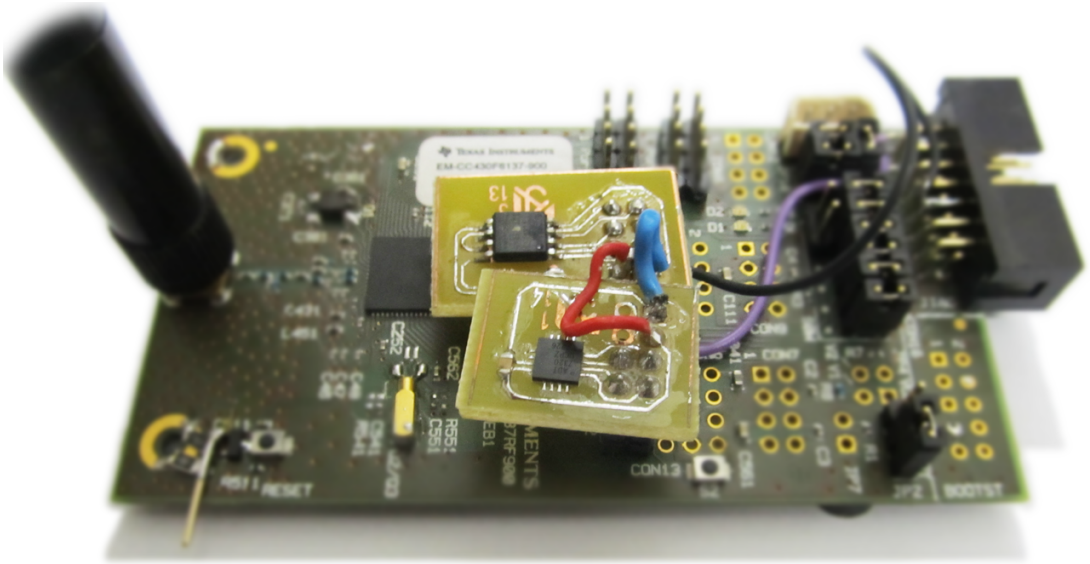
IMTEK

Masterarbeit

Drahtlose Entfernungsmessung



Thorbjörn Jörger

UNI FREIBURG

Eingereichte Masterarbeit gemäß den Bestimmungen der Prüfungsordnung der Universität Freiburg für den Masterstudiengang Mikrosystemtechnik vom 31. 10. 2010

Lehrstuhl für elektrische Mess- und Prüfverfahren
Institut für Mikrosystemtechnik (IMTEK)
Albert-Ludwigs-Universität Freiburg
Freiburg im Breisgau

Autor Thorbjörn Jörger

Bearbeitungszeit 26. Juni 2013 bis 26. Juni 2014

Gutachter
Prof. Dr. Leonhard M. Reindl,
Lehrstuhl für elektrische Mess- und Prüfverfahren
Prof. Dr. Ulrich Schwarz,
Lehrstuhl für Optoelektronik

Betreuer
Dr.-Ing. Gerd-Ulrich Gamm,
Lehrstuhl für elektrische Mess- und Prüfverfahren
Dr.-Ing. Fabian Höflinger,
Lehrstuhl für elektrische Mess- und Prüfverfahren

Titelseite Das Bild zeigt den Aufbau der Messplattform für die Entfernungsmessung mit angefertigten Hardware-Erweiterungen

Erklärung

nach der Prüfungsordnung:

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

Freiburg, den 26. Juni 2014

Thorbjörn Jörger

Kurzfassung

Die vorliegende Masterarbeit befasst sich mit dem Entwurf, dem Aufbau und der Charakterisierung eines Systems zur drahtlosen Entfernungsbestimmung durch die Messung der *round trip time* (RTT) in drahtlosen Sensorknoten.
Vor dem Hintergrund der steigenden Verbreitung von drahtlosen Sensornetzwerken wächst auch das Interesse an der Positionsbestimmung in solchen Netzwerken. Für eine Vielzahl von Anwendungen ist es notwendig, die genaue Position eines Sensorknotens in einem dreidimensionalen Bezugssystem zu kennen. Bestehende Sensorknoten verwenden zur Positionsbestimmung die integrierte Signalstärkemessung von dem Stand der Technik entsprechenden HF-Chips, wie beispielsweise des CC430 von *Texas Instruments*. Dies hat den Nachteil, dass die Signalstärkemessung stark von der räumlichen Orientierung der Knoten abhängt, da die Antennen keine perfekten Kugelstrahler darstellen. Darüber hinaus nimmt die Genauigkeit auf größeren Entfernungen aufgrund der kleiner werdenden Änderung der empfangenen Signalstärke zunehmend ab. Zudem muss die Gesamtdämpfung und die Ausgangsleistung der Knoten kalibriert werden. In geschlossenen Räumen treten zudem noch Mehrwegausbreitungs- und Interferenzeffekte störend in Erscheinung.
Eine weitere Möglichkeit zur Positionsbestimmung mittels Trilateration ist die Messung der Signallaufzeit. Dies hat den Vorteil, dass die Genauigkeit auch auf langen Strecken, auf denen signalstärkebasierte Messverfahren zunehmend schlechtere Genauigkeit bieten, recht genau bestimmt werden kann. Bestehende Systeme verwenden die RTT-Messung bereits in einer Vielzahl von Anwendungsfällen, es existiert jedoch noch kein System, dass zu einer Positionsbestimmung von *low-cost wireless sensor nodes* in der Lage ist. Der CC430 von *Texas Instruments* ist einer der am häufigsten für kostenoptimierte Sensornetzwerke verwendeten Funkchips. In dieser Arbeit wird dieser Funkchip daher auf seine Verwendbarkeit in einem System zur laufzeitbasierten Positionsbestimmung untersucht und ein Überblick über die optimalen Einstellungen für verschiedene Umgebungsparameter gegeben. Es wurden weiterhin die systematischen und statistischen Fehlerquellen der Laufzeitmessung in den verschiedenen Betriebsmodi untersucht und die besten Einstellungen für zukünftige Messungen vorgestellt.

Stichwörter: drahtlose Entfernungsmessung, drahtlose Laufzeitmessung, drahtlose Positionsmessung, Rundlaufzeit

Abstract

The thesis at hand deals with the design, the construction and the characterization of a system for wireless distance estimation by round trip time (RTT) measurements in low-power sensor nodes.
In the context of increasing use of moving wireless sensor nodes the interest in localizing these nodes in their application environment is strongly rising. For many applications, it is necessary to know the exact position of the nodes in two- or three-dimensional space. Commonly used nodes use state-of-the-art transceivers like the CC430 from *Texas Instruments* with integrated signal strength measurement for this purpose. This has the disadvantage, that the signal strength measurement is strongly dependent on the orientation of the node through the antennas inhomogeneous radiation pattern as well as it has a small accuracy on long ranges. Also, the nodes overall attenuation and output power has to be calibrated and interference and multipath effects appear in closed environments.
Another possibility to trilaterate the position of a sensor node is the time of flight measurement. This has the advantage, that the position can also be estimated on long ranges, where signal strength methods give only poor accuracy. Existing systems use round trip measurements already in a multitude of application environments, but no system is existens for low-cost wireless sensor nodes. The CC430 from *Texas Instruments* is one of the most applied RF front ends for wireless sensor networks. Therefore in this thesis an investigation of the suitability of the state-of-the-art transceiver CC430 for a system based on time of flight methods is presented and an overview of the optimal settings under various circumstances for the in-field application is given. For this investigation, the systematic and statistical errors in the time of flight measurements with the CC430 have been investigated under a multitude of parameters.

Keywords: wireless distance estimation, wireless runtime measurement, wireless position estimation, round trip time

Danksagung

Mein besonderer Dank gilt Herrn Prof. Dr. Leonhard M. Reindl, Lehrstuhl für elektrische Mess- und Prüfverfahren für die Betreuung der Arbeit und das familiäre Umfeld, dass er am Lehrstuhl geschaffen hat, welches den Lehrstuhl zu einem zweiten Zuhause für mich hat werden lassen. Herrn Prof. Dr.-Ing. Ulrich Schwarz, Lehrstuhl für Optoelektronik, möchte ich für die kurzfristige Bereitschaft als Zweitkorrektor für die Arbeit zur Verfügung zu stehen, herzlich danken.
Ganz besonders danke ich meinen Betreuer, Herrn Dr.-Ing. Ulrich Gamm und Herrn Dr.-Ing. Florian Höflinger, beide Lehrstuhl für elektrische Mess- und Prüfverfahren, für die intensive Betreuung der Arbeit, ihre stete Hilfsbereitschaft und Unterstützung während der Erstellung der Arbeit.
Unseren Technikern Hans Baumer, Uwe Burzlaff und Christoph Bohnert danke ich für ihre technische Hilfe und Unterstützung beim Entwurf und Aufbau, ihrer immer währenden Bereitschaft und Hilfe beim Brainstorming und in Lebenskrisen und Rettern in allerlei technischen wie wissenschaftlichen Notfällen. Weiterhin gilt mein Dank den Kollegen und Mitarbeitern des Lehrstuhls für die fachliche und freundschaftliche Unterstützung während der Erstellung der Arbeit. Nicht zuletzt gilt mein besonderer Dank Frau Angelina Müller, Lehrstuhl für Mikroaktorik, für ihre vielfältige fachliche Unterstützung und Freundschaft während der letzten sieben Jahre.

Inhaltsverzeichnis

Abbildungsverzeichnis

Nomenklatur

Lateinische Buchstaben

Variable	Bedeutung	Einheit
c_x	Signalgeschwindigkeit	m/s
d	Abstand	m
D	Bodenbeschaffenheit	–
f	Frequenz	Hz
F_n	n-te Fresnell-Zone	–
G_x	Antennengewinn	dBi
H	Übertragungsfunktion	–
n	Anzahl an Messungen	–
P_r	empfangene Leistung	dBm
P_s	Sendeleistung	dBm
q	Quantisierungsintervall	–
Q	Menge der diskretisierten Werte	–
r	Entfernung	m
s^2	empirische Varianz	–
s	empirische Standardabweichung	–
t	Laufzeit	s
Δt	Laufzeitdifferenz	s
$t_{latency}$	Verarbeitungszeit	s
Δt_{TOF}	korrigierte Rundlaufzeit	s
Δt_{RTT}	Rundlaufzeit	s
T	Temperatur	°C
U	Spannung	V
v_p	Verkürzungsfaktor	–
X_{in}	Eingangswertebereich	V

Griechische Buchstaben

Variable	Bedeutung	Einheit
λ	Wellenlänge	m
σ	Standardabweichung	–
ω	Kreisfrequenz	Hz

Konstanten

Variable	Bedeutung	Wert
c_0	Lichtgeschwindigkeit im Vakuum	299792458 $^{m}/_{s}$
c_{air}	Lichtgeschwindigkeit in Luft	299704644 $^{m}/_{s}$
n_{air}	Brechungsindex der Luft	1,000292

Abkürzungen

Abkürzung	Bedeutung
IMTEK	Institut für Mikrosystemtechnik
EMP	Lehrstuhl für elektrische Mess- und Prüfverfahren
SPI	Serial Peripheral Interface Bus
IC	Integrated Circuit (engl. integrierter Schaltkreis)
μC	Mikrokontroller
RSSI	Received Signal Strength Indication (engl. Anzeige der Empfangsfeldstärke)
LoS	Line of Sight (engl. Sichtlinie)
ToA	Time of arrival (engl. Messung der Ankunftszeit)
TDoA	Time difference of arrival (engl. Differenzmessung der Ankunftszeit)
RTT	Round trip time (engl. Messung der Rundenzeit)
ToF	Time of flight (engl. Messung der Laufzeit)
AoA	Angle of arrival (engl. Messung des Einfallwinkels)
EBx	Evaluation-Board Nummer x
RPi	Raspberry Pi
SoC	System on Chip (engl. Monolithisch integrierte Komponenten)
2-FSK	Frequency Shift Keying (Modulationsart)

Abkürzung	**Bedeutung**
2-GFSK	Gaussian Frequency Shift Keying (Modulationsart)
MSK	Minimum Shift Keying (Modulationsart)
OOK	On Off Keying (Modulationsart)
SRD-Band	Frequenzband für Short Range Devices
ISM-Band	Frequenzband für Industrie, Forschung und Medizin
PWM	Pulsweitenmodulation
UART	Serieller Bus
LSB	Least significant bit (engl. niederwertigste Stelle)
PLL	Phase-locked loop (engl. Phasenregelschleife)
PID	Proportionaler integrierender und differenzierender Regler
VGA	Variable Gain Amplifier (engl. einstellbarer Verstärker)
AGC	Automatic Gain Control (engl. automatische Steuerung der Verstärkung)
FRAM	Ferroelectric Random Access Memory (engl. Statischer Festspeicher)
LNA	Low-noise Amplifier (engl. rauscharmer Verstärker)
ISR	Interrupt Service Routine (engl. Verarbeitung einer Unterbrechungsanforderung)
ppm	part per million = 10^{-6}
ppb	part per billion = 10^{-9}
RF	Radio Frequency (engl. Radiowellen)

1 Zielsetzung und Motivation

Ziel dieser Arbeit ist die Analyse und Verbesserung der drahtlosen und laufzeitbasierten Entfernungsmessung. Ein spezifisches System, namentlich das *Evaluation-Board CC430F6137*[1] von *Texas Instruments*, wird dazu charakterisiert, um verschiedenste Einflussparameter zu evaluieren. Es existieren bereits viele Lösungen für drahtlose Entfernungsmessung und spezifisch für dieses Einsatzgebiet entworfene Hardware. Der Ansatz dieser Arbeit ist hingegen, eine präzise Entfernungsmessung für bereits eingesetzte und in der Praxis verwendete Hardware zu implementieren. Ein große Anzahl dieser Hardware besitzt bereits integrierte Funktransceiver zur Kommunikation. Es besteht ein großer Bedarf solche drahtlosen Sensorknoten in ihrem Einsatzgebiet zu lokalisieren. Die Anwendungsgebiete solcher Sensorknoten reichen von persönlicher Navigation für Fußgänger in Bahnhöfen, Messen, Einkaufszentren oder Flughäfen über die Lokalisation autonomer Roboter in Fertigungsanlagen bis zur Verwaltung der Position von Baumaschinen auf Großbaustellen. Für die meisten dieser Anwendungsfälle steht kein ausreichender GPS-Empfang zur Verfügung und zudem ist der Energiebedarf von GPS-Empfängern um zwei bis drei Größenordnungen zu hoch für den dauerhaften Betrieb energieautarker Sensorknoten. Die häufig eingesetzte Methode der Lokalisierung durch Messung der Empfangsfeldstärke erfordert in der Regel keinen zusätzlichen Bauteilaufwand und lässt sich problemlos für energieautarke Sensorsysteme implementieren. Die Empfangsfeldstärke variiert allerdings durch den Einfluss von Objekten innerhalb des Bereichs der Funkausbreitung und Umwelteinflüssen wie beispielsweise der Luftfeuchtigkeit. Zudem ist die Empfangsfeldstärke von der Orientierung der Antennen zueinander, der Abstrahlcharakteristik der Antennen und der Fehlanpassung durch Bauteilabweichungen auf den einzelnen Knoten abhängig. Dies macht eine Kalibrierung der Ausgangsleistung für jeden einzelnen Knoten notwendig und die Methode somit sehr aufwendig und teuer. Um daher eine kostengünstige und energieeffiziente Lokalisation zu ermöglichen, kann die Signallaufzeit des Funksignals zur Positionsbestimmung mittels Trilateration verwendet werden.
Das Hauptproblem bei Laufzeitmessungen ist die verhältnismäßig langsame Taktrate

[1] MSP430-Prozessor mit monolithisch integriertem CC1101 Sub-1-GHz-Funkchip

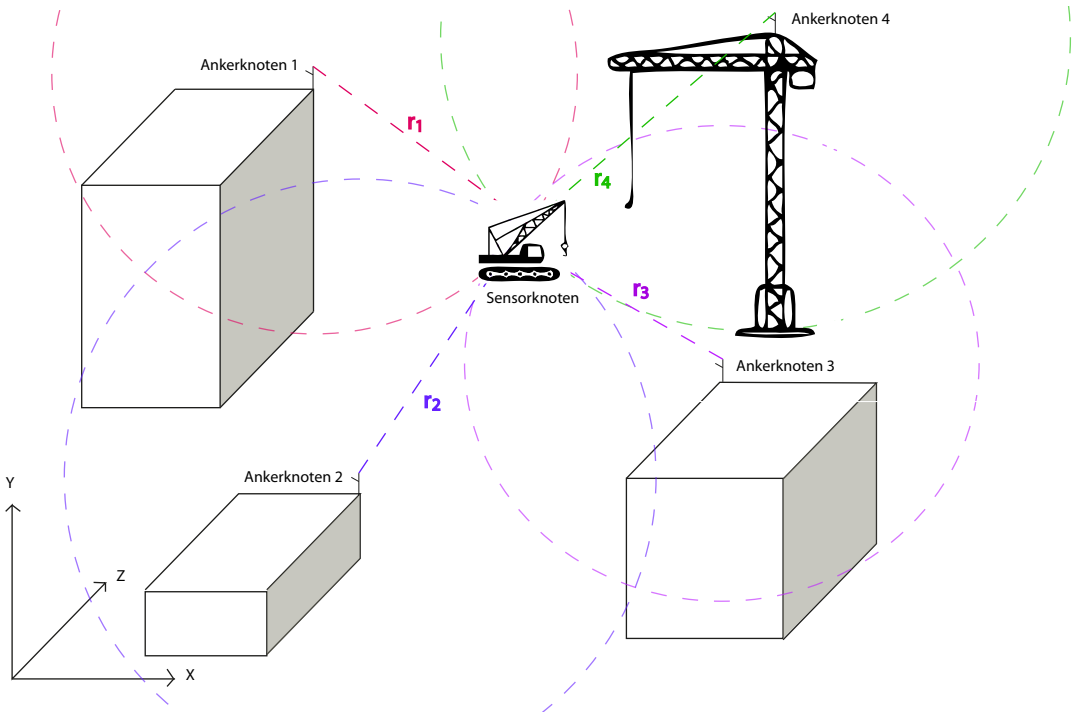


Abb. 1.1: Konzeptidee: Lokalisation auf einer Baustelle. Durch strategisch positionierte Ankerknoten mit bekannter Position kann durch Trilateration ein mobiler Sensorknoten lokalisiert werden, um beispielsweise Baumaschinen oder Großwerkzeug zu überwachen und zu steuern.

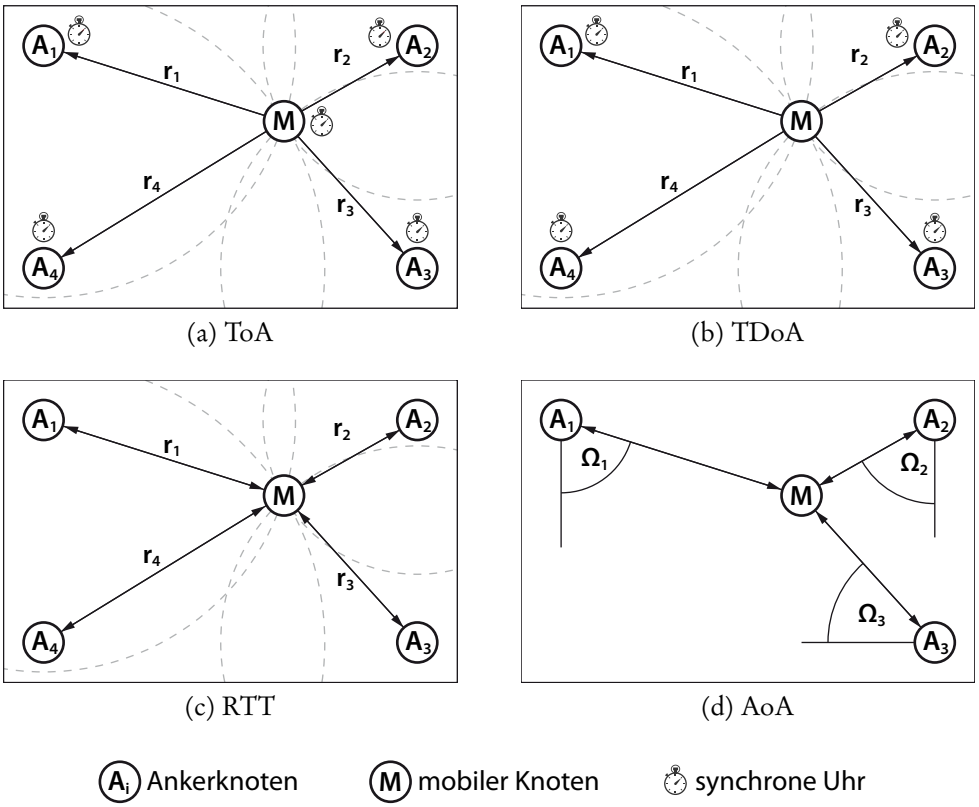
der verwendeten Mikrokontroller. Jede Messung ist unter anderem mit einem Messfehler durch den Clock-Jitter in Höhe von einem Clock-Zyklus behaftet. Dies ist aufgrund der langsamen Signallaufzeit bei Laufzeitmessungen mit Ultraschall unkritisch, führt jedoch bei der Laufzeitbestimmung über elektromagnetische Wellen die sich mit Lichtgeschwindigkeit ausbreiten, zu einem nicht mehr vernachlässigbaren Fehler. Ein einzelner Zyklus einer 26 MHz-System-Clock hat eine Länge von 38,4 ns und erzeugt somit einen Fehler in der Distanzmessung von 11,5 m. Da der erzeugte Fehler jedoch normalverteilt ist, lässt sich dieser mit häufigen Messungen herausrechnen. Ein weitere Herausforderung stellt die Hardware dar, die nicht für Messungen der Signallaufzeit ausgelegt ist. Diese erzeugt in der vorliegenden Arbeit einen Offset von mehreren hundert Zyklen zwischen dem physikalischen Empfang eines Pakets und dem logischen Signalisieren des Empfangs im Digitalteil. Dieser, durch den Analogteil des Transceivers verursachte, Offset ist zudem von der Temperatur, der Dämpfung des Signals, der verwendeten Trägerfrequenz und Modulationsart und verschiedenen weiteren Parametern abhängig. Diese Arbeit untersucht diese Einflüsse und ihre Auswirkungen auf Entfernungsmessung mittels drahtloser Sensorknoten und gibt einen Überblick über die besten Einstellungen für weitere Forschung in der Zukunft.

2 Theorie

In diesem Kapitel werden die angewendeten theoretischen Zusammenhänge dargestellt und erläutert. Die einfachen und bekannten Sachverhalte sind, soweit es notwendig war, der gängigen Fachliteratur entnommen worden. Grundlagen und Einzelheiten der Positionsbestimmung wurden von *Höflinger* beschrieben [1]. Diese wurden teilweise übernommen und für den konkreten Anwendungsfall erweitert und weiterentwickelt. Die theoretischen Ansätze zum Dithering und der Plausibilitätstest wurden in Eigenarbeit erstellt. Aus anderen Arbeiten entnommene Abbildungen wurden gekennzeichnet, und falls eine Bearbeitung erfolgte, mit einem Vermerk versehen.

2.1 Positionsbestimmung

Die Positionsbestimmung eines mobilen Objekts wird in aller Regel durch Trilateration, seltener, da wesentlich aufwendiger, durch Triangulation, durchgeführt. Die Trilateration lässt sich auf mehrere Entfernungsmessungen von Referenzpunkten mit bekannter Position zum mobilen Objekt zurückführen, wohingegen sich die Trilateration analog auf mehrere Winkelmessungen zwischen bekannten Referenzpunkten und dem mobilen Objekt zurückführen lässt. Für den zweidimensionalen Fall werden bei der Trilateration drei Entfernungsmessungen von verschiedenen Referenzpunkten benötigt, für die Triangulation im Gegensatz nur zwei. Trotzdem wird in den meisten Anwendungsfällen die Trilateration bevorzugt, da sie technisch einfacher zu realisieren ist. Im dreidimensionalen Fall sinkt zudem der verhältnismäßige Mehraufwand für die Trilateration gegenüber der Triangulation nochmals, da hier im ersten Fall vier und im zweiten Fall drei Messungen zur zweifelsfreien Zuordnung der Position zum mobilen Objekt notwendig sind und der Messaufwand bei der Triangulation durch die Notwendigkeit der Erfassung des Raumwinkels nochmals steigt. Die verschiedenen Verfahren sind in Abbildung 2.1 vergleichend dargestellt.
Die Triangulation wird in dieser Arbeit nicht verwendet und daher nicht weiter beschrieben. Zu diesem Thema bieten *Mo et al.* einen guten Überblick und zeigen unter Anderem Methoden zur Verarbeitung der Einfallswinkelmessung (*angle of arrival*) [2].



(a) ToA (b) TDoA

(c) RTT (d) AoA

Abb. 2.1: Überblick über Methoden zur Positionsbestimmung im Raum. a) Bei Ankunftszeitmessung wird aus dem Zeitstempel des Sendens und des Empfanges die Entfernung bestimmt. b) Bei der Differenzzeitmessung wird direkt aus den Differenzen in der Ankunftszeit die Position berechnet. c) Bei der Messung der Rundlaufzeit wird die Entfernung durch bidirektionale Kommunikation gemessen. d) Bei der Messung des Ankunftswinkels wird durch Triangulation die Position bestimmt.

2.2 Entfernungsmessung

Da sich die Positionsbestimmung, wie im vorhergehenden Abschnitt beschrieben, auf mehrere Entfernungsmessungen zurückführen lässt, werden in diesem Abschnitt verschiedene Verfahren zur drahtlosen Entfernungsmessung mittels Funk vorgestellt.

2.2.1 Empfangsfeldstärke

Mit Hilfe der abgestrahlten Sendeleistung P_t, der Antennengewinne auf Empfänger- und Senderseite G_r und G_t sowie der Entfernung r und der Wellenlänge λ kann an-

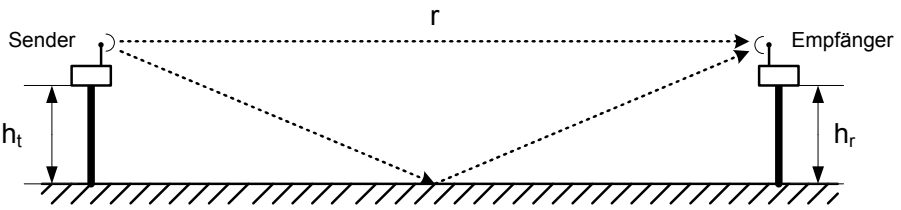


Abb. 2.2: Bei Messungen im freien Feld kommt es durch Reflexionen am Boden zu Mehrwegausbreitungen. Diese führen durch konstruktive und destruktive Interferenz zu Schwankungen in der Empfangsfeldstärke. Aus [3], bearbeitet.

hand der Friis-Gleichung die maximale theoretische Empfangsleistung P_r im Empfängerknoten mit

$$P_r = P_t \cdot G_t \cdot G_r \cdot \left(\frac{\lambda}{4\,\pi\,r}\right)^2 \tag{2.1}$$

berechnet werden. Bei bekannten Parametern kann daraus durch Messung der Empfangsfeldstärke im Empfängerknoten die entsprechende Entfernung durch

$$r = \frac{\lambda}{4\,\pi} \cdot \sqrt{\frac{P_t}{P_r} \cdot G_t \cdot G_r} \tag{2.2}$$

berechnet werden. Die Empfangsfeldstärke nimmt in diesem Modell bei logarithmisch aufgetragener Ordinate linear über der Entfernung ab. Für die tatsächliche Ausbreitung im Freifeld, insbesondere im Hinblick auf Reflexionen am Boden, wie in Abbildung 2.2 dargestellt, und die dadurch verursachte konstruktive und destruktive Interferenz, muss jedoch ein komplexeres Modell betrachtet werden. So berechnet sich durch Verwendung zusätzlicher Parameter wie der Höhe der Sensorknoten h_r und h_t und dem Einfluss der Bodenbeschaffenheit D ein Korrekturfaktor für Gleichung 2.1. Diese wird somit zu:

$$P_r = P_t \cdot G_t \cdot G_r \cdot \left(\frac{\lambda}{4\,\pi\,r}\right)^2 \cdot 2\left[1 - D\cos\frac{4\pi\,h_r\,h_t}{\lambda\,r}\right]. \tag{2.3}$$

Diese Gleichung besitzt keine algebraische Lösung für r mehr. Mit numerischen Methoden lassen sich für r mehrere Lösungen finden. Es lässt sich also für eine gegebene Empfangsfeldstärke keine eindeutige Entfernung mehr angeben. Der Sachverhalts ist in Abbildung 2.3 dargestellt. Zur Entfernungsbestimmung werden daher Schätzmethoden, wie die Maximum-Likelihood-Methode, zur Minimierung des Gesamtfehlers des Systems herangezogen. Die Genauigkeit der Entfernungsmessung durch die Feldstärkemessung hängt dabei von der Anzahl der Ankerknoten und der Umgebung ab.

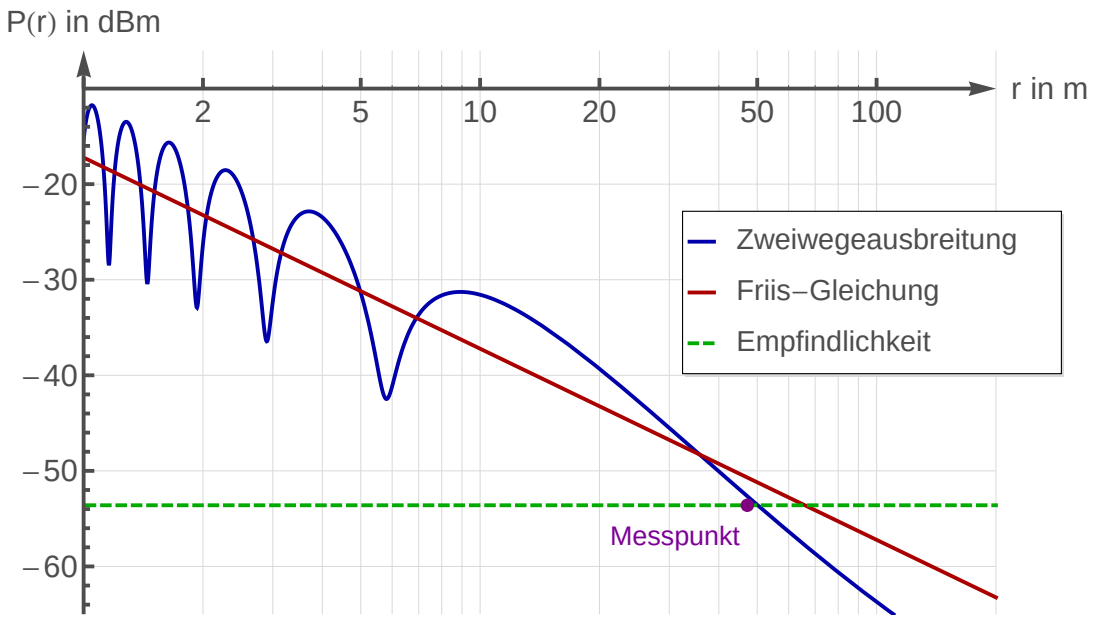


Abb. 2.3: Periodische Schwankungen der Empfangsfeldstärke durch Interferenz durch Bodenreflexionen. Eine eindeutige Zuordnung der Entfernung zur gemessenen Empfangsfeldstärke ist aufgrund der Mehrdeutigkeit nicht mehr möglich. Aus [3].

Für die Umrechnung der digitalen, im Zweierkomplement vorliegenden, Werte $RSSI_{dec}$ die der HF-Chip bereitstellt in absolute Werte $RSSI_{dBm}$ wird folgende Formel herangezogen [4]:

$$RSSI_{dBm} = \begin{cases} \frac{RSSI_{dec}-256}{2} - RSSI_{off} & \text{für } RSSI_{dec} >= 128 \\ \frac{RSSI_{dec}}{2} - RSSI_{off} & \text{sonst} \end{cases} \quad \text{mit } RSSI_{off} = 74 \tag{2.4}$$

2.2.2 Laufzeitmessungen

Laufzeitmessungen beruhen auf dem Verhältnis zwischen der Entfernung r, der Ausbreitungsgeschwindigkeit c und der Laufzeit t:

$$r = c \cdot t. \tag{2.5}$$

Die Entfernung wird hierbei durch Messung der vergangenen Laufzeit mit bekannter Ausbreitungsgeschwindigkeit $c_{air} \approx c_0$ der Funkwellen in Luft berechnet. Die genaue Abweichung der Lichtgeschwindigkeit in Luft, gegeben mit dem Brechnungsindex der

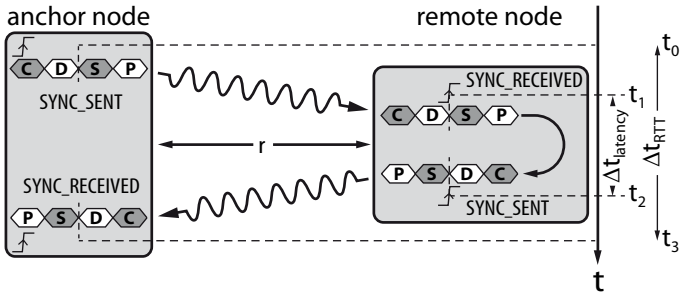


Abb. 2.4: Übersicht der Timing-Parameter bei der Kommunikation zwischen den Knoten. Die Zeitindizes kennzeichnen den genauen Zeitpunkt, zu dem das Capture-Register des Mikrokontrollers getriggert wird.

Luft n_{air} durch

$$c_{air} = \frac{c_0}{n_{air}}, \tag{2.6}$$

ist dabei zu klein, um bei den für diese Arbeit erreichten Genauigkeiten ins Gewicht zu fallen. Für kabelgebundene Messungen muss hingegen der Verkürzungsfaktor v_p des Kabel berücksichtig werden. Die Signalgeschwindigkeit im Kabel c_{cable} ergibt sich dann aus:

$$c_{cable} = c_0 \cdot v_p. \tag{2.7}$$

Die Laufzeitmessungen können in verschiedenen Modi durchgeführt werden. Bei der Ankunftszeitmessung (ToA, engl. *time of arrival*) wird der Zeitpunkt der Ankunft t_2 eines Pakets anhand einer genauen Uhr erfasst und mit dem bekannten Sendezeitstempel t_1 die Laufzeit Δt über

$$\Delta t = t_2 - t_1 \tag{2.8}$$

berechnet. Dieses Verfahren erfordert allerdings hochgenaue und synchrone Uhren und ist somit nicht für alle Einsatzgebiete geeignet. Dieses Problem lässt sich teilweise durch Differenzzeitmessungen (TDoA, engl. *time difference of arrival*) umgehen. Hier wird aus der Differenz der Ankunftszeiten an zwei bekannten Referenzstationen die Entfernung berechnet. Der Vorteil dieses Verfahrens ist, dass nur noch die Referenzstationen über synchrone und genaue Uhren verfügen müssen. Der zu lokalisierende Sender ist dagegen frei von solchen Einschränkungen.
Um sich völlig von der Notwendigkeit der synchronen Uhren zu lösen, kann die Rundlaufzeit Δt_{RTT} (RTT, engl. *round trip time*) gemessen werden. Die Genauigkeit der Messung hängt dann nur noch von der Genauigkeit der einzelnen Uhren ab, nicht mehr von ihrer Synchronität. Dazu sendet der Ankerknoten ein Paket an den mobilen Knoten und erfasst den Zeitpunkt des Sendens t_0. Der mobile Knoten erfasst zur Ent-

fernung der eigenen Verarbeitungszeit $t_{latency}$ aus der Laufzeitmessung den Moment des Empfangs des Pakets t_1. Nach vollständigem Empfang des Pakets sendet der mobile Knoten das Paket zurück an den Ankerknoten und speichert den Moment des Sendens t_2. Sobald der Ankerknoten das retournierte Paket empfängt, wird der letzte benötigte Zeitstempel t_3 erfasst. Die Rundlaufzeit lässt sich dann mit

$$\Delta t_{RTT} = t_3 - t_0 \tag{2.9}$$

berechnen. Durch Entfernung der Verarbeitungszeit $\Delta t_{latency} = t_2 - t_1$ berechnet sich dann die korrigierte Rundlaufzeit

$$\Delta t_{TOF} = \Delta t_{RTT} - \Delta t_{latency}. \tag{2.10}$$

2.3 Funkkommunikation

2.3.1 Modulation

Unter Modulation wird in der Funktechnik die Kombination eines Trägersignals mit einem zu übertragenden Nutzsignal verstanden. Dazu stehen verschiedene Modulationsarten zur Verfügung. Für diese Arbeit wurde ausschließlich das binäre Frequenzumtastungsverfahren in einer Variante ohne (2-FSK, engl. *frequency shift keying*) und einer Variante mit Gauß'schem Filter (2-GFSK, engl. *Gaussian frequency shift keying*) verwendet. Informationen werden bei diesem Verfahren dadurch übertragen, dass die Frequenz des Signals um einen gewissen Frequenzhub zur einen oder anderen Seite der Mittenfrequenz verschoben wird. Der Gaußfilter wird dazu verwendet, die Bandbreite des Signals zu verringern indem hochfrequente, durch hohe Flankensteilheit verursachte Anteile des Digitalsignals entfernt werden. Dadurch erhöht sich die spektrale Effizienz des übertragenen Signals. Die Übertragungsfunktion des Gauß-Filters

$$H(j\omega) = e^{-(\frac{\omega}{2\alpha})^2} \text{ mit } \alpha = \frac{\pi}{\sqrt{\ln(\sqrt{2})}} \tag{2.11}$$

erzeugt dabei für jeden Rechteckimpuls in Form eines Bits eine entsprechende gaußsche Glockenkurve mit kontinuierlichem Werteverlauf.

2.3.2 Frequenz und Mehrwegausbreitung

Mehrwegausbreitung und die dadurch verursachte Signalauslöschung durch destruktive Interferenz stellen ein bekanntes Problem in der Datenübertragung mittels Funk, insbesondere im Innenraum, dar. *Hashemi* beschreibt verschiedene Verfahren zur Modellierung des Funkkanals im Innenraum [5]. Da der Umfang der Modellierung den Rahmen dieser Arbeit sprengen würde, werden an dieser Stelle nur die wichtigsten Einflussparameter vorgestellt.
Die Trägerfrequenz f für die Entfernungsmessung mit der Wellenlänge λ hat durch die Bragg-Gleichung

$$n\lambda = 2d \sin\phi, \tag{2.12}$$

direkten Einfluss auf das Interferenzmuster. Sollte an einer Empfangsstelle durch destruktive Interferenz keine ausreichende Empfangsfeldstärke vorliegen, kann dies über einen Wechsel der Trägerfrequenz umgangen werden. Um ein Auftreten von destruktiven Interferenzen im Vorfeld zu unterbinden, müssen die von *Fresnel* beschrieben Einflüsse von Objekten in den so genannten Fresnelzonen F_n berücksichtigt werden. Die Fresnelzonen beschreiben ein Gebiet, dass sich kreisförmig orthogonal zur direkten Sichtverbindung erstreckt. Der Radius der n-ten Fresnelzone am Punkt P errechnet sich aus der Wellenlänge, dem Abstand zwischen den Antennen d und dem jeweiligen Abstand d_1 und d_2 zum entsprechenden Punkt P durch:

$$F_n = \sqrt{\frac{n\,\lambda\,d_1\,d_2}{d}}. \tag{2.13}$$

Hindernisse innerhalb der ungeraden Fresnelzonen reflektieren die einfallende Welle so, dass destruktive Interferenz im Empfänger auftritt. Um eine geringe Dämpfung in der Funkverbindung sicherzustellen, ist daher mindestens die erste Fresnelzone F_1 frei von Hindernissen zu halten. Dies gestaltet sich beim Einsatz im Innenraum häufig sehr schwierig. Objekte in den geraden Fresnelzonen erhöhen hingegen die Signalstärke im Empfänger, jedoch sind diese Signale um eine Wellenlänge phasenverschoben. Dies kann unter Umständen zu Schwierigkeiten bei der Demodulation führen.

2.4 Statistische Verfahren für die Signalverarbeitung

Zur Auswertung der Messdaten und zur Signalverarbeitung wurden verschiedene statistische Verfahren verwendet. Neben den Standardverfahren für die Berechnung des Mittelwerts und der Standardabweichung wurden noch Verfahren zur Berechnung der erforderlichen Anzahl an Messungen verwendet, sowie Dithering, eine Methode zur Verfeinerung der Messauflösung durch häufige Messungen.

2.4.1 Standardverfahren

Wenn in der Ausarbeitung von Mittelwerten die Rede ist, ist immer der arithmetische Mittelwert

$$\overline{x} = \frac{1}{n} \sum_{i=1}^{n} x_i \tag{2.14}$$

einer Stichprobe mit Werten x_i und dem Umfang n gemeint. Auf die Streuung der Werte um den Mittelwert lässt sich aus der empirischen Varianz

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2 \tag{2.15}$$

durch Berechnung der empirischen Standardabweichung

$$s = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \overline{x})^2} \tag{2.16}$$

schließen. Diese gibt dann den einfacher handhabbaren Wert an, um den die Messwerte schwanken. Für den Fall normalverteilter Messwerte lässt sich unter Verwendung der Gammafunktion $\Gamma(x)$ aus dem Stichprobenumfang n und der empirischen Standardabweichung s der erwartungstreue Schätzer

$$\widehat{\sigma} = \sqrt{\frac{N-1}{2}} \cdot \frac{\Gamma\left(\frac{N-1}{2}\right)}{\Gamma\left(\frac{N}{2}\right)} \tag{2.17}$$

für die absolute Standardabweichung

$$\sigma = s \cdot \widehat{\sigma} \tag{2.18}$$

berechnen. Dieser nähert sich für einen großen Stichprobenumfang sehr schnell an 1 an. Für einen gegebenen Mittelwert mit der Standardabweichung σ_1 über einem Stichprobenumfang der Größe n lässt sich die Standardabweichung der Mittelwerte

$$\sigma_n = \sigma_1 \cdot \sqrt{\frac{1}{n}} \tag{2.19}$$

berechnen. Damit kann die benötigte Anzahl an Messungen

$$n = \frac{\sigma_1^2}{\sigma_x^2} \tag{2.20}$$

für einen Mittelwert, der maximal um die gewünschte Unsicherheit σ_x vom Populationsmittelwert entfernt liegt, berechnet werden.

(2.21)

2.4.2 Quantisierungsfehler und Dithering

Eine grundsätzliche Problemstellung in der digitalen Messtechnik entsteht durch die Diskretisierung der Messwerte. Aus dem Eingangswertebereich X_{in} und der Bit-Tiefe des Wandlers n ergibt sich das Quantisierungsintervall

$$q = \frac{X_{in}}{2^n} \mathrel{\widehat{=}} \text{LSB} \tag{2.22}$$

der Diskretisierung. Der digitale Wert X_q kann jetzt nur Werte aus der Menge der diskretisierten Werte Q annehmen. Die Diskretisierung erzeugt deshalb einen Fehler f_q der Messung von ±0,5 LSB. Dies berechnet sich nach

$$f_q = X - X_q \text{ mit } X_q \in Q \text{ und } Q = \{0\,,\ q\,,\ \ldots\,,\ 2^n - 1 \cdot q\} \tag{2.23}$$

aus dem wahren Wert X und dem quantisierten Wert X_q. Es ist nun nicht möglich sich, selbst durch häufige Wiederholung der Messung, dem wahren Wert der Messgröße anzunähern, da bei jeder Messung nur der quantisierte Wert X_d erhalten wird.

Üblicherweise wird in diesem Fall die Auflösung der Quantisierung bis zur gewünschten Auflösung verfeinert oder das Eingangssignal analog vorverstärkt. Falls dies nicht möglich ist, bietet sich die Option zusätzliches weißes Rauschen mit einer Amplitude größer 1 LSB auf das Signal zu addieren. Alternativ ist es ebenfalls ausreichend, wenn

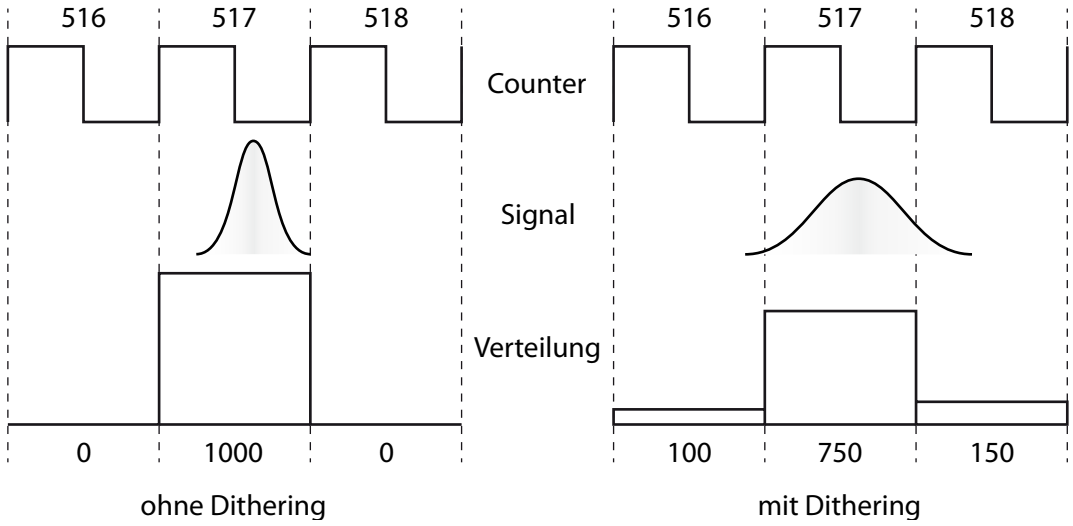


Abb. 2.5: Durch Dithering, die Addition von zusätzlichem Rauschen auf das Eingangssignal, wird die Messauflösung durch eine größere Anzahl an Messungen kleiner als das *Least Significant Bit* (LSB). Dargestellt ist der Takt des Zählers in der ersten Spalte, das Eingangssignal in der zweiten Spalte und das zugehörige Histogramm in der dritten Spalte.

das Signal selbst bereits einen Rauschanteil größer 1 LSB enthält. Nun ergeben sich bei wiederholter Messungen unterschiedliche diskrete Werte deren Häufigkeitsverteilung Informationen über den wahren Wert enthält. Dies ist möglich, da die Amplitude von weißem Rauschen im Mittel genau gleich 0 ist und somit im Mittel keine Verfälschung des wahren Werts bewirkt. Eine grafische Darstellung des Effekts findet sich in Abbildung 2.5.

2.4.3 Plausibilitätstest der Messdatenzuordnung

Die Einzelmessungen des Zeitintervalls zur differentiellen Zeitmessung werden nicht synchron übertragen, sondern im weiteren Verlauf der Messung wieder einander zugeordnet. Daher ist es notwendig die Plausibilität dieser Zuordnung zu überprüfen. Diese Überprüfung wird durch ein statistisches Verfahren durchgeführt. Die Zeitmessungen unterliegen dem in Gleichung 2.10 dargestellten Zusammenhang:

$$\Delta t_{RTT} = \Delta t_{latency} + \Delta t_{TOF}. \tag{2.24}$$

Die Verarbeitungszeit der Paketdaten $\Delta t_{latency}$ im Empfängerknoten schwankt um einen normalverteilten Fehler f_n. Diese Schwankung wird durch die Zeitmessung im Senderknoten ebenfalls miterfasst:

$$\Delta t_{RTT} + f_n = \Delta t_{latency} + f_n + \Delta t_{TOF}. \tag{2.25}$$

Werden die zueinander gehörigen Messwerte voneinander abgezogen, so fällt dieser Fehler wieder aus der Messung heraus. Werden jetzt aber zwei nicht zueinander gehörige Zeitmessungen verknüpft, bleibt aufgrund ihrer Normalverteilung die Differenz der Fehler erhalten:

$$\Delta t_{RTT,2} - \Delta t_{latency,1} = \Delta t_{TOF} + (f_{n,1} - f_{n,2}). \tag{2.26}$$

Damit lässt sich leicht überprüfen, ob die richtigen Werte zugeordnet werden. In Abbildung 2.6 ist dies beispielhaft für ein Messregime bestehend aus 2500 Einzelmessungen dargestellt. Die Normalverteilung der Messwerte ist bei richtiger Zuordnung der Messwerte deutlich schmaler.

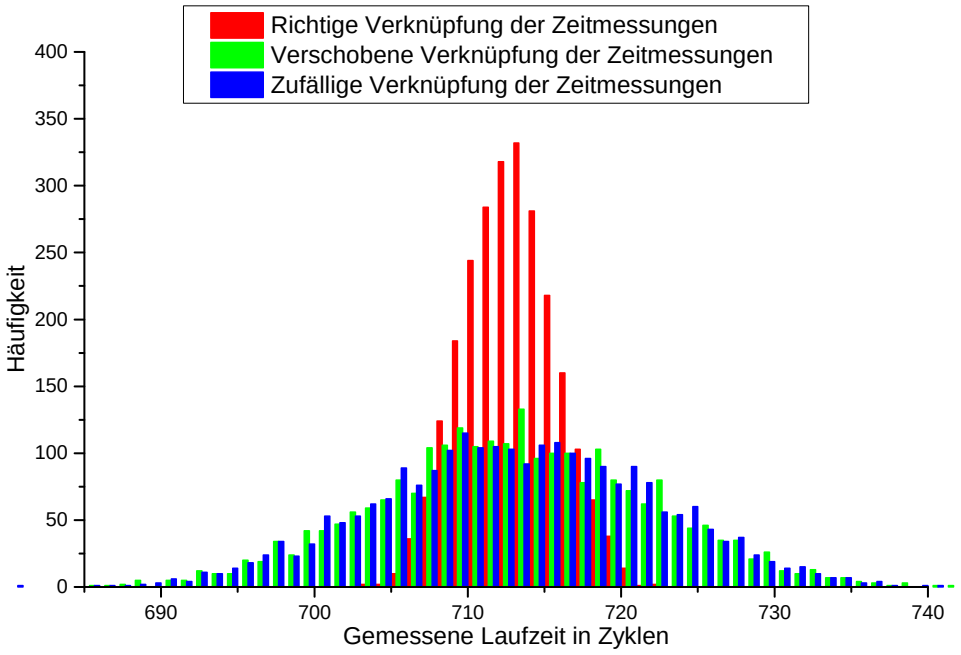


Abb. 2.6: Plausibilitätstest der Messdatenzuordnung durch ein Histogramm über 2500 Einzelmessungen. Die Normalverteilung der Messwerte ist bei richtiger Zuordnung der Messwerte sichtbar schmaler als bei Verschiebung um eins oder zufälliger Zuordnung. Damit wurde eine korrekte Zuordnung der Werte durch die Software sichergestellt.

3 Stand der Technik

Für die Lokalisation kostengünstiger drahtloser Sensorknoten wird in der Forschung und in den meisten Anwendungsgebieten der Signalstärkeindikator (RSSI) verwendet [6–8]. Er ist mittlerweile auf den meisten Funktransceivern bereits vorhanden und bietet somit eine einfache und günstige Möglichkeit zur Lokalisation. Die Ausbreitung von Funksignalen im Innenraum wurde bereits 1993 von *Hashemi* umfangreich untersucht [5]. Neuere Forschung in diesem Gebiet beschäftigt sich primär mit der Anwendung in verschiedenen Funk-Standards, der optimalen Verteilung von Ankerknoten und der effizientesten Modellierung der Positionsabschätzung. So beschreiben *Sugano et al.* eine Lokalisierungsmethode mit einer Ankerknotendichte von 0,27 $^{\text{Knoten}}/_{\text{m}^2}$ unter Verwendung des *ZigBee*-Standards und erreichen einen Fehler in der Positionsbestimmung von 2 m. Sie verwenden eine Maximum-Likelihood-Methode zur Positionsbestimmung [9]. Einen aktueller Überblick über die Literatur zur Ausbreitung und Koexistenz von Funkwellen im Innenraum und zugehörigen Messungen geben *Amzucu et al.* in [10]. Sie beschreiben unter anderem Interferenzeffekte durch Auslöschung, die bei wenigen Zentimetern betragenden Positionswechseln Änderungen in der empfangen Signalstärke von bis zu 30 dBm hervorrufen. Zudem wird der Einfluss von leistungsstarken Störquellen in Form von Mikrowellenöfen in der Nähe der Sensorknoten untersucht und Lösungsansätze unter Ausnutzung von störungsfreien Zeitfenstern vorgestellt. Die beschrieben Interferenzeffekte im Innenraum unterliegen zudem einem zeitlichen Wandel. Dies erschwert die Modellierung nochmals, da eine einmalige Modellierung des Umfeldes nicht mehr für eine genaue Entfernungsmessung ausreicht. *Viogradov et al.* beschreiben in [11] ein Szenario, dass mittels statistischer Evaluation der Auslöschungseffekte modelliert wird.
Andere Gruppen versuchen über eine Kombination von RSSI-Messungen und laufzeitbasierten Messungen eine höhere Genauigkeit der Positionsbestimmung herbeizuführen. *Murphy et al.* beschreiben ein trilaterationsbasiertes Prinzip zur Positionsbestimmung in Kohleminen und stellen durchgeführte Messungen und Berechnungen vor [12]. *Bahillo et al.* stellen ein hybrides Lokalisierungs-Schema vor, indem eine Kombination aus beiden Messverfahren verwendet wird [13]. In [14] führen sie eine Anwendungsstudie des vorgestellten Schemas durch. *Golden et al.* verwenden einen ähnlichen Ansatz, aber mit der spezifischen Kombination von WLAN mit GPS [15]. Hierbei

wird besonders auf die Versorgung von vom GPS abgeschatteten Bereichen in städtischer Umgebung mit Positionsdaten abgezielt. *Patwari* stellt ein System zur kooperativen Lokalisierung vor und untersucht verschiedene Systemkonzepte [16]. Eine Untersuchung mit Fokus auf die Hardware-Eigenschaften und den inhärenten Fehler der Messungen durch die Drift der Oszillatoren geben *Günther* und *Hoene* in [17]. *Ciurana et al.* stellen ein System vor, das nur über Software ToA-Positionierungen in WLAN-Netzen durchführt. Dies geschieht über direktes Abgreifen der MAC-Pakete von der Infrastruktur [18, 19].
Eine theoretische Betrachtung der Grenzen der Cramér-Ungleichung (CRB) für verschiedene Hybridsystem findet sich in [20]. *Catovic* und *Sahinoglu* untersuchen darin die CRB für Kombinationen aus Signalstärke- und Laufzeitmessungen, insbesondere in der Nähe von Ankerknoten. *Fritsche* und *Klein* untersuchen in [21] die CRB für ein Hybridsystem im Mobilfunkbereich und unter Einbindung von GPS in ihre Betrachtung. Sie zeigen, dass die Hybridmethode signifikante Genauigkeitssteigerung bei der Lokalisierung aufweist.
Will et al. stellen in [22] ein System vor, dass einen ausschließlich laufzeitbasierten Ansatz verwendet. Sie verwenden einen MSB-A2 Sensorknoten mit integriertem LPC2387 Mikrokontroller auf ARM7-Basis mit 72 MHz Taktrate. Als Funktransceiver komm ein CC1101 von *Texas Instruments* zum Einsatz. Aus der zugehörigen Diplomarbeit von *Pfeiffer* geht die erreichte Genauigkeit des Systems hervor [23]. Diese liegt im Außenbereich bei etwa 3 m, im Innenraum dagegen zwischen 10-20 m.
Zur Auswertung der Positionsdaten werden von verschiedenen Gruppen Algorithmen verwendet. So stellen *Wendeberg et al.* ein Verfahren zur ankerlosen Selbstlokalisation durch Zeitdifferenzmessung vor. Die Berechnung der Position eines Knotens erfolgt durch einen Feder-Masse-Algorithmus [24]. *Wigren* und *Wennervirta* beschreiben einen Algorithmus zur Positionsbestimmung von Mobiltelefonen im UMTS Netz und führen eine Feldstudie durch. Der Algorithmus führt eine Datenfusion zwischen gemessener Laufzeit und der Zellengeometrie durch [25, 26]. Einen Gesamtsystem zur Positionierung von Sensoren wird von *Salman et al.* vorgestellt, sowie eine Betrachtung der Cramér-Rao-Grenzen durchgeführt. Das System verwendet Entfernungsmessungen über die Signalstärke und einen gewichteten *linear least squares*-Algorithmus [27].
Sun et al. geben einen Überblick über alle Verfahren die dem Stand der Technik entsprechen. Sie legen besonderes Augenmerk auf die Signalverarbeitung und versuchen eine einheitliche Lösung für verschiedene Netzwerkkonfigurationen zu finden [28].
Röhrig et al. stellen das kommerziell bereits vermarkte System nanoLOC vor, dass auf Rundlaufzeitmessungen basiert. Es bietet Genauigkeiten im Bereich von 0,5 m und wird beispielsweise bereits zum Monitoring in der Tierzucht verwendet [29, 30]

4 Entwurf und Aufbau

Dieses Kapitel widmet sich der Konstruktions- und Planungsphase für das in Abbildung 4.1 dargestellte Messsystem, sowie die zugehörigen Messstände. Es wird der verwendete Transceiver und das zugehörige Evaluation-Board vorgestellt und Gründe für die Auswahl genau dieses System dargelegt. Weiterhin werden die vorgenommenen Einstellungen und Änderungen in der Sofware- und Hardware-Konfiguration vorgestellt und die grundlegende Funktionsweise erklärt. Der entworfene Entfernungsmessstand und die zugehörige Programmierung werden präsentiert und die Grundgedanken des Designs und die Funktionsweise erläutert. Zudem werden die wichtigsten verwendeten Laborgeräte und Ausrüstungsgegenstände vorgestellt und ihre wichtigsten Charakteristiken genannt. Abschließend wird noch auf die vorgenomme automatisierte Auswertung eingegangen.

4.1 Evaluation-Board EM-CC430-F6137-900

Für die Durchführung aller Messungen wurden zur besseren Vergleichbarkeit und Nachvollziehbarkeit der Arbeit kommerziell frei verfügbare Evaluation-Boards der Firma *Texas Instruments* verwendet. Tabelle 4.1 ist zu entnehmen, welchem Board welche arbeitsinterne Nummer zugeordnet wurde. Der CC430-Mikrokontroller wird bereits häufig im Bereich der drahtlosen Sensorknoten eingesetzt und wurde auch schon vom Autor in mehreren Arbeiten, Projekten und Veröffentlichungen verwendet [31–33]. Ebenso wurde er am Lehrstuhl für elektrische Mess- und Prüfverfahren bereits für verschiedenste Untersuchungen verwendet und wird von einer Ausgründung des Lehrstuhl sogar kommerziell in einer Überwachungs- und Lokalisierungsanwendung eingesetzt [34, 35]. Daher wurde er als naheliegendste Wahl für die Untersuchung von drahtloser, paketbasierter Entfernungsmessung herangezogen.

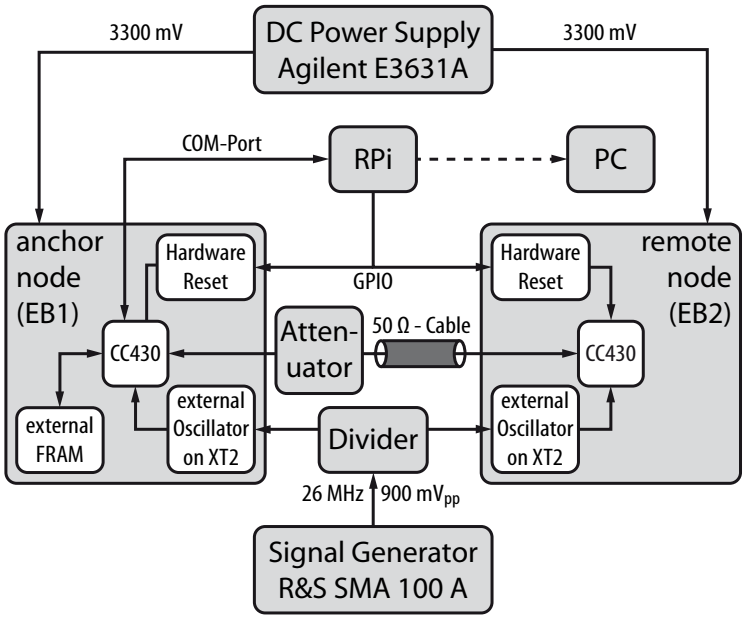


Abb. 4.1: Blockdiagramm des Messsystems. Die zwei Evaluation-Boards sind mit einem Koaxial-Kabel verbunden. Die Messung wird durch Senden eines Befehls, der das Start-Kommando und die durchzuführende Anzahl an Einzelmessungen enthält, an den Ankerknoten gestartet. Dieser initiiert dann die Kommunikation mit dem mobilen Knoten und speichert alle anfallenden Daten in seinem FRAM-Speicher ab. Nach Ende der vorgegebenen Anzahl an Messungen werden die gespeicherten Daten an die Messstandsteuerung übermittelt.

4.1.1 Mikrokontroller

Der in den Evaluation-Boards verbaute Mikrokontroller CC430F6137 bietet eine echte System-on-Chip (SoC) Integration eines MSP430-16-Bit-RISC-Mikrokontrollers und einem Derivat des CC1101-Funktransceivers auf einem einzelnen Chip. Der Mikrokontroller verfügt über 32 kB programmierbaren Flashspeicher, 4 kB SRAM, 44 GPIOs, zwei 16 Bit-Timer, DMA, 32 Bit-Hardware-Multiplikation, einen 10 Bit-ADC-Wandler und diverse Bustreiber für UART, I^2C und SPI bei einer maximalen Taktrate von

Tab. 4.1: Evaluation-Boards mit Seriennummer und arbeitsinterner Bezeichnung.

Bezeichnung innerhalb der Arbeit	Seriennummer	Revision
EB1	SN05095	3.2
EB2	SN07416	3.2
EB3	SN05726	3.2
EB4	SN05096	3.2
EB5	SN07737	3.2

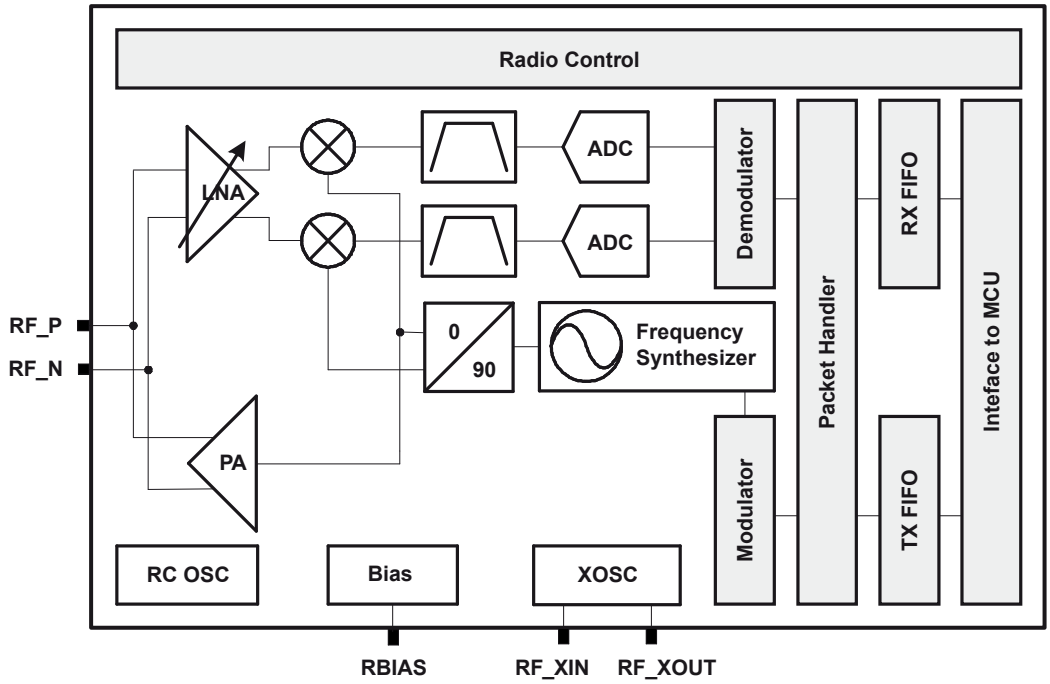


Abb. 4.2: Vereinfachtes Blockdiagram des HF-Teils des CC430 [36]. Dargestellt ist der symmetrische Eingang auf der linken Seite, der nachgeschaltete LNA, die Teilung und Abmischung des Signals auf eine Zwischenfrequenz und die anschließende AD-Wandlung. Auf der rechten Seite des Blockdigramms findet sich die digitale Schnittstelle zum Mikrokontroller.

20 MHz. Des Weiteren lässt sich der Energiebedarf des Mikrokontrollers durch Implementierung von fünf sogenannten *low power modes* (LPM) und den großen Versorgungsspannungseingangsbereich von 1,8 - 3,6 V drastisch reduzieren, was ihn besonders geeignet für den Einsatz in energieautarken drahtlosen Sensornetzwerken macht [36].
Das ebenfalls integrierte CC1101-Derivat biete eine hohe Ausgangsleistung von 10 mW im ISM- und SDR-Band bei gleichzeitig geringem Stromverbrauch von 20 mA. Die maximale Übertragungsrate von 250 kB/s im Paketbetrieb erlaubt die schnelle Übertragung von Sensormesswerten und entsprechend hohe Messfrequenzen. Durch die hohe Sensitivität des Funktransceivers ist eine Kommunikation selbst bei der höchsten Datenrate über eine Strecke von mehreren hundert Metern möglich. Als Modulationsverfahren kommen *on off keying* (OOK), *minimum shift keying* (MSK), *frequency shift keying* (FSK) und *gaussian shaped frequency shift keying* (GFSK) zum Einsatz. Genaueres zur Modulation kann Abschnitt 2.3.1 entnommen werden.
Das verbaute CC1101-Derivat unterscheidet sich nur in wenigen maßgeblichen Punkten vom Originalchip. Die meisten Unterschiede rühren von der monolithischen Integration mit einem Mikrokontroller her. So werden gewisse Clock-Signale mit dem μC geteilt, die *states* nach einem Reset unterscheiden sich und es ist ein interrupt- und

timer-gesteuerter Betrieb der Funk-Funktion möglich. Weitere geringfügige Änderungen, wie die Chip-ID und verschiedene Register-Werte nach dem Reset sind nicht weiter von Belang. Jedoch haben sich einige Werte im Datenblatt leicht verändert, was auf ein zumindest teilweises Redesign auf dem Chip-Level schließen lässt. Es wäre denkbar, dass durch die kleinere Distanz stärker auftretende hochfrequente Einstreuungen durch den Mikrokontroller zusätzliche Filtermechanismen im Analogteil notwendig werden ließen. Ob dies einen Einfluss auf die Vergleichbarkeit der Ergebnisse mit anderen Gruppen, die den orginalen CC1101 verwenden, hat, ist bisher nicht abschließend geklärt.
In Abbildung 4.2 ist ein vereinfachtes Blockdiagramm des HF-Teils dargestellt. Das Einganssignal wird in einem *low noise amplifier* (LNA) vorgefiltert, verstärkt und in zwei gleiche Teile gespalten. Diese werden dann mit einer vom Frequenzsynthetisierer erzeugten bekannten Frequenz und der um 90 ° phasenverschobenen Kopie zum Interphasen- und Quadratursignal heruntergemischt. Diese Signale mit einer wesentlich niedrigeren Zwischenfrequenz werden dann nach einer Bandpassfilterung abgetastet und anschließend digital demoduliert. Die anschließende Signalverarbeitung und -übertragung geschieht digital. Der Sendeteil des Transceivers arbeitet nach einem vergleichbaren Prinzip, das *Texas Instruments* aber nicht offenbart.
Ebenfalls nicht dargestelllt sind verschiedene Techniken zur Signalverstärkung. Die Verstärkung wird über einen *variable gain amplifier* (VGA) der anliegenden Signalstärke nachgeführt, um möglichst gleichbleibende Pegel in der Analog-Digital-Wandlung zu erhalten. Dies wird automatisch durch die sogenannte *automatic gain control* (AGC) durchgeführt. Die AGC regelt die Vorverstärkung in Analogteil über die im Digitalteil gemessenen Empfangspegel, namentlich den RSSI-Wert. Diese Nachführung ermöglicht einen größeren Dynamikbereich des Eingangssignalpegels und somit eine höhere Flexibilität des Gesamtsystems.
Eine weitere Verbesserung des Empfangsverhaltens wird durch die automatische Nachführung der Trägerfrequenz erreicht. Dies bietet insbesondere im Low-Cost-Sektor entscheidende Vorteile, da so auf den Einsatz genauer und gut kalibrierter und somit teurer Quarze verzichtet werden kann. Dabei wird während des Empfangs der Präambel die interne *phase-locked loop* (PLL) des Empfängers auf die empfangene Trägerfrequenz geschoben und dort fixiert. Dadurch können die internen Filter wesentlich schmalbandiger gestaltet werden, was das *signal to noise ratio* (SNR) erhört und damit einen besseren Empfang bei niedrigem Eingangssignalpegel ermöglicht.

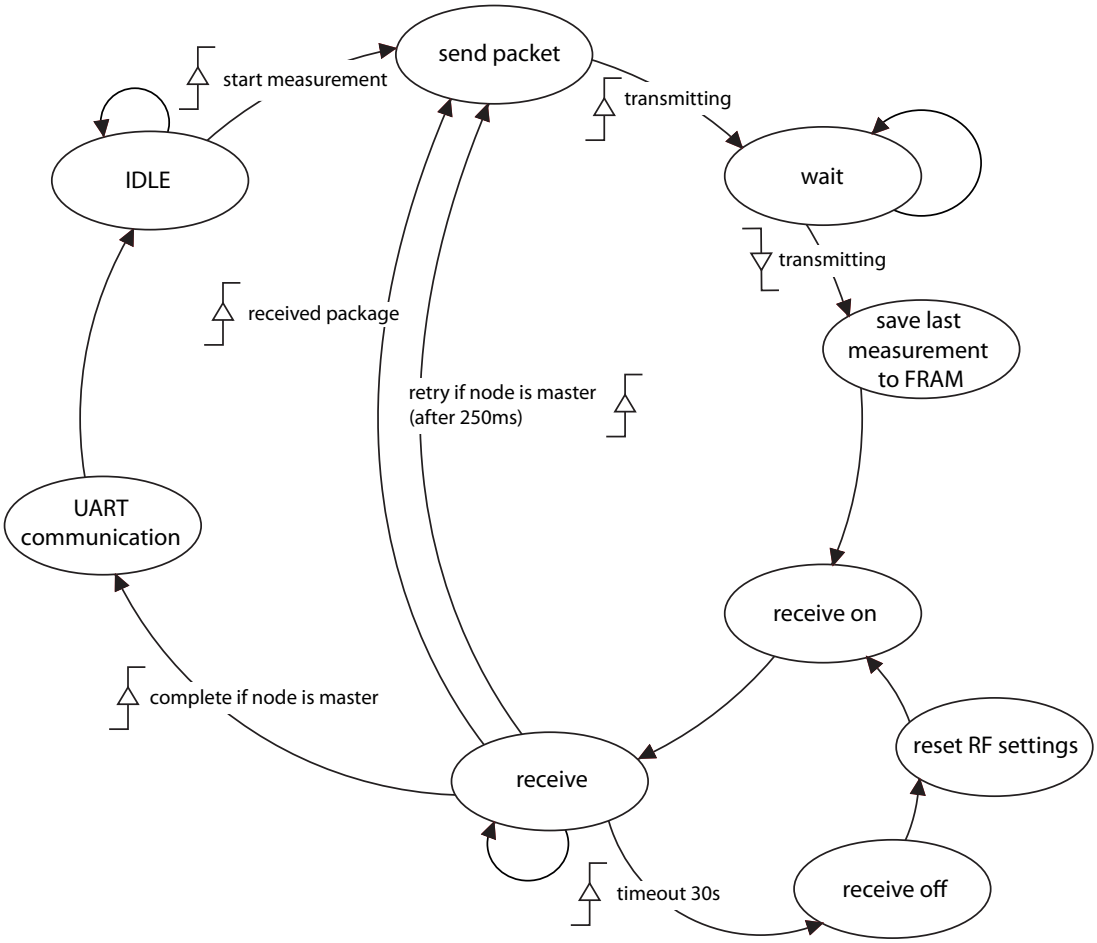


Abb. 4.3: Flußdiagramm des Messprogramms auf den Evaluation-Boards. Die Übergänge zwischen des *states* werden durch verschiedene Unterbrechungsanforderungen und Steuersignale ausgelöst. Abgesehen von zwei Pfaden, die nur vom Master-Knoten beschritten werden, ist die Programmierung der Platinen von Master und Slave identisch.

4.1.2 Konfiguration

Die Platinen wurden zur Ansteuerung der externen Hardware und Durchführung der RTT-Messungen, wie Anhang B entnommen werden kann, programmiert. In Abbildung 4.3 ist das Flußdiagramm des Messprogramms dargestellt. Es wurde eine *state machine* zur Ablaufsteuerung implementiert. Eine der Hauptaugenmerke lag auf einer einfach nachvollziehbaren Struktur des Programms, um eine Erweiterbarkeit sicherzustellen. Die Programmierung auf Empfänger- und Senderseite unterscheidet sich nur an wenigen Stellen, so dass die Konfiguration des Quellcodes durch zwei Präprozessorbefehle ohne Weiteres möglich ist. Zudem ist eine möglichst deterministisches Ausführungsverhalten von besonderer Bedeutung, da häufige Sprünge in Interruptfunktionen große Variationen in der Verarbeitungszeit einzelner Pakete zur Folge haben. Um

diese Einflüsse zu minimieren, wird die vorgebene Wiederholungszahl an Messungen von den Platinen selbstständig und ohne Unterbrechung durch den Messstand durchgeführt. Es war daher erforderlich, sämtliche Messdaten auf den Evaluation-Boards abzuspeichern und erst nach der Durchführung der Messung zu übertragen. Aufgrund des begrenzten Arbeistspeichers des CC430, wurde ein FRAM-Speicher in das Messsystem integriert. Damit ist es möglich, ungestört circa 20.000 Einzelmessungen durchzuführen, bevor eine Kommunikation mit dem Messstand erforderlich ist. So können etwaige Einschwingprozesse durch Verwerfen der Anfangsmessungen aus der Messung herausgenommen werden.
Die Messungen der Laufzeit wird durch Abfragen des *capture counters* des CC430 bewerkstelligt. Dieser erhält seinen Takt direkt aus der 26 MHz-Referenz des Hochfrequenzteils. Der Counter wird direkt durch das vom GDO1-Port an den entsprechenden Trigger-Port weitergeleitete SYNC_WORD_RECEIVED-Flag des HF-Teils getriggert. Dies schließt Messfehler durch schwankende Sprungzeiten der Interrupt-Service-Routine (ISR) aus. Der gespeicherte Capture-Wert hingegen wird durch eine ISR abgefragt und gespeichert. Die Differenz von zwei so erfassten Werten wird dann als Verarbeitungszeit an den Messstand weitergeleitet. Eine Plausibilitätsüberprüfung wird von den Platinen selbst nicht durchgeführt. Fehler durch verlorene Pakete oder anderweitige Fehler äußern sich aber sehr deutlich, so dass sie ohne Weiteres durch das Auswertungsprogramm herausgefiltert werden können.
Mit der von *Texas Instruments* frei zur Verfügung gestellten Software *SmartRF Studio 7* wurden die empfohlenen Register-Werte für die Konfiguration des HF-Teils bestimmt. Die Software berechnet für gewünschte Rahmenparameter wie Modulation, Frequenz, Bandbreite, Datenrate und Paketstruktur, die entsprechenden funktionierenden Einstellungen der internen Vorverstärker, Mischer, Digitalisierer und Eingangsfilter. An den von der Software vorgeschlagenen Konfigurationsdaten wurden aufgrund der Erhaltung der Vergleichbarkeit und der Kompatibilität keine Änderungen vorgenommen. Die Konfigurationsdaten sind aufgrund ihrer Vielfalt sehr umfangreich und daher hier nicht dargestellt. Sie können dem Anhang C entnommen werden. Tabelle 5.1 auf Seite 32 gibt unter anderem einen Überblick über die verwendeten Rahmenparameter.
Die Paketgröße für die durchgeführten Messungen beträgt 17 Byte. Die Pakete bestehen aus einer 4 Byte langen Präambel, dem ebenfalls 4 Byte langen Sync-Word, 7 Daten-Bytes und 2 Bytes für den CRC-Wert, wie in Abbildung 4.4 dargestellt. Die Daten-Bytes enthalten den 4 Byte großen Timer-Wert $\Delta t_{latency}$, den 1 Byte großen RSSI-Wert und eine 2 Byte große Adresse zur Knotenidentifikation. Für die Temperaturmessungen wurden die Adress-Bytes zur Übermittlung des 2 Byte großen Temperaturwertes

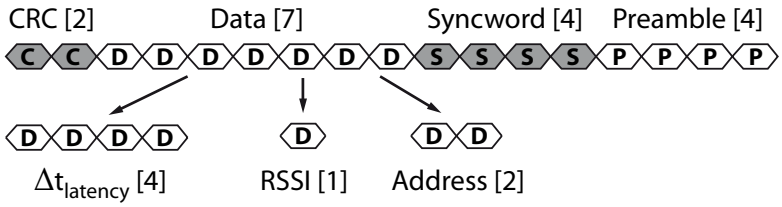


Abb. 4.4: Paketstruktur und Paketgröße, sowie Paketlänge. Die Zahlen in den Klammern geben die Größe der Variable in Byte an. Der CRC-Wert wird für jedes Paket automatisch vom Mikrokontroller berechnet. Das Sync-Word ist konstant und muss im Master- und Slave-Knoten identisch sein. Die Präambel wird automatisch vom Transceiver generiert und dient unter anderem der Trägerfrequenzkorrektur.

verwendet. Weil die Genauigkeit des Systems von einer hohen Anzahl an Messungen abhängig ist, wurde die Paketstruktur möglichst kompakt gewählt, um eine möglichst hohe Anzahl an Messungen in einem gegebenen Zeitrahmen durchzuführen. Allerdings bedürfen hohe Datenraten einer geringen Gesamtdämpfung der Funkausbreitung, was mit limitierter Ausgangsleistung und zunehmender Entfernung schwieriger wird.

4.1.3 Hardware-Erweiterungen

Für einige Messungen waren Modifikationen und Erweiterungen der Hardware notwendig. Der Arbeitsspeicher des Mikrokontrollers war mit 4 KB nicht ausreichend für die Speicherung der anfallenden Messungen. Um eine große Anzahl an Messungen am Stück durchführen zu können, wurde daher ein FRAM-Speicher in das System integriert. Zur Verwendung kam der 2-MBit-FRAM-IC FM25H20 von *Cypress*. Die Ansteuerung erfolgt über einen der SPI-Ports des CC430. Das Layout der dafür angefertigte Erweiterungsplatine findet sich im Anhang A. Die erstellte Programm-Bibliothek findet sich im Anhang B.
Die Temperatur der Platinen sollte für die Charakterisierung des Transceivers genau erfasst werden. Der bereits auf dem CC430 vorhandene Temperatursensor war allerdings nicht hinreichend genau. Daher wurde ein Präzisions-Temperatursensor ADT7320 von *Analog Devices* ins System integriert. Er bietet mit einer Auflösung von 7,3 mK und einer Genauigkeit von ±0,25 K im verwendeten Messbereich eine hinreichend genaue Temperaturreferenz. Der Sensor wurde ebenfalls über SPI angesteuert. Das Layout dieser Platine und die Programm-Bibliothek finden sich ebenfalls im Anhang.
Die Kommunikation mit dem Messstand erfolgt über den COM-Port des Mikrokontrollers mit einer Datenrate von 115,2 kbaud. Die Taktrate für die Kommunikation ist direkt vom internen RC-Oszillator des Mikrokontrollers abgeleitet. Da das eingesetzte Kommunikations-Protokoll keine Synchronisierung vorsieht und der Takt des

RC-Oszillators mit 0,1 % /°C stark temperaturabhängig ist, kam es bei der Durchführung der Temperaturzyklen immer wieder zu Übertragungsfehlern und Verbindungsabbrüchen durch voneinander abweichende Taktraten. Daher wurde ein temperaturstabiler Uhrenquarz mit einer Taktrate von 32,768 kHz, einer maximalen Abweichung von 20 ppm und einer Temperaturdrift von maximal 40 ppb/K auf die von *Texas Instruments* ohnehin vorgesehene Stelle auf dem Evaluation-Board gelötet und der Systemtakt von diesem Quarz abgeleitet.
Um die vollständige Automatisierung der Messung selbst bei schwerwiegenden Störungen in der Messung sicherzustellen, wurde der Systemreset-Knopf des Evaluation-Boards durch einen Pull-Down-MOSFET ebenfalls ansteuerbar gemacht. Die Messstandsteuerung ist damit in der Lage den Master-Knoten beliebig zurückzusetzen.

4.2 Laborgeräte und Hilfsmittel

Für die Durchführung der Messungen wurden verschiedene Geräte und Hilfsmittel verwendet. Der folgende Abschnitt gibt einen kurzen Überblick über die wichtigsten Eigenschaften der verwendeten Geräte und Hilfsmittel.

Agilent E3631A

Die Versorgung der Messplattform erfolgt durch das Labornetzgerät *Agilent E3631A*. Es liefert im Bereich zwischen 0 V und 6 V eine stabile Gleichspannung mit einer Genauigkeit von 0,5 ‰.

Rohde & Schwarz SMA 100 A

Der Signalgenerator *SMA 100 A* von *Rohde & Schwarz* wurde für einige Experimente zur synchronen Taktung des HF-Teils der Evaluation-Boards verwendet. Er ist in der Lage hochgenaue, einstellbare Frequenzen zwischen 9 kHz und 6 GHz zu liefern. Das Phasenrauschen beträgt -141 dBc/Hz bei 1 GHz.

Agilent 34401A

Die Spannungen auf der Messplattform werden mit dem *Agilent 34401A* Digital Multimeter mit einer Genauigkeit von 3,5 ‰ vom Messbereich gemessen. Das Multimeter wurde zur Überwachung der eingestellten Spannung verwendet.

Dämpfglieder hp8494B und hp8496B

Da die Signalstärke im Empfänger bei einer Verbindung über Kabel um circa 40 dB höher ist, als bei Ausbreitung über die Luft, ist eine zusätzliche Dämpfung des Signals bei den kabelgebundenen Messungen notwendig. hierzu wird ein Kombi-Dämpfglied von *hp* verwendet. Es erlaubt die Einstellung der gewünschten Dämpfung in 1 dB-Schritten im Bereich von 0-110 dB. Die Bandbreite des Dämpfglieds reicht vom Gleichspannungsbereich bis 18 GHz.

Hochfrequenzkabel

Die verwendeten Hochfrequenzkabel von *elspec* standen in Längen von 5 m, 10 m und 20 m zur Verfügung und besitzen nur eine äußerst geringe Dämpfung von 14 dB/100m. Der Frequenzgang der Kabel ist im benutzten Frequenzbereich von 868-915 MHz glatt. Zur Verbindung dieser Hochfrequenzkabel mit dem Messaufbau und dem Dämpfglied kamen RG223-Hochfrequenzkabel von 1 m Länge mit unterschiedlichen Verbindungsteilen zum Einsatz.

Klimakammer

Die Klimakammer VCL4010 von *Vötsch* mit einem Prüfraumvolumen von 100 l ist in der Lage einen Messaufbau über einen Temperaturbereich von -40 °C bis +180 °C zu temperieren. Die maximale Heizleistung liegt dabei bei 3,5 K/min, die maximale Kühlleistung bei -5 K/min und die zeitliche Temperaturabweichung bei 0,3-1,0 K.

4.3 Entfernungsmessstand

4.3.1 Komponenten

Für diese Arbeit wurde ein vollautomatischer Entfernungsmessstand, wie in Abbildung 4.5 dargestellt, konzipiert, entworfen und aufgebaut. Die Infrastruktur besteht aus DN 200 KG-Rohr in dem eine Seilbahn verläuft. Die gesamte Konstruktion ruht auf Dreifüßen aus Holz und hält damit einen Abstand zum Boden von 1,90 m. Die Dreifüße wurden mit *Mathematica* dimensioniert, wie in Abbildung 4.6 dargestellt. Im Innern des Rohres wird der in Abbildung 4.7 dargestellte Wagen durch die Seilbahn positioniert. Auf dem Wagen befindet sich der mobile Knoten, ein diffuser Reflektor als Ziel für den Laserentfernungsmesser, ein 6 V-Bleiakku und ein 3,3 V-Spannungswandler zur Energieversorgung des Knotens. Die Bahnsteuerung basiert auf einem Raspberry

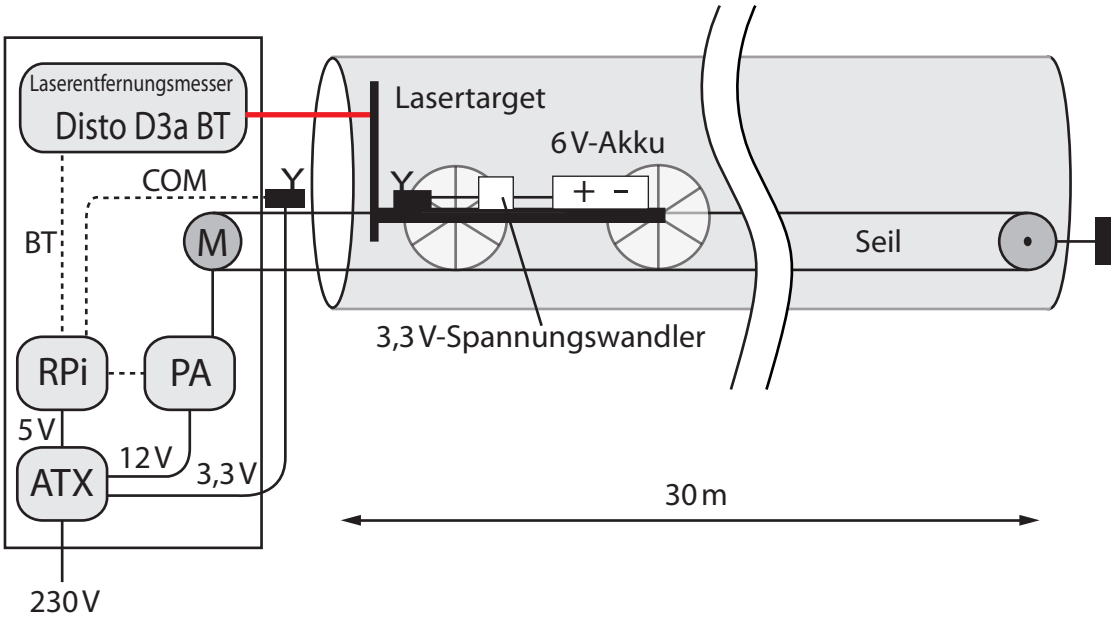


Abb. 4.5: Blockdiagram des Messstands. Im linken Teil der Grafik befindet sich die Messstandsteuerung mit dem Antrieb und der Entfernungsmessung. Im rechten Teil der Grafik ist das Rohr mit innenlaufender Seilbahn und Messbahn dargestellt. Über das Seil zieht der Motor den Messwagen auf die gewünschte Position.

Pi, der die Leistungselektronik zur Ansteuerung des Motors kontrolliert. Diese wird vom Raspberry Pi mit einem PWM-Logik-Signal angesteuert, das dann verstärkt wird und so den Motor antreibt. Die Richtungsumschaltung wird durch ein ebenfalls entsprechend gesteuertes Relais durchgeführt. Der Schaltplan und die entworfene Platine der Leistungselektronik finden sich im Anhang A.

Über einen Laserentfernungsmesser D3a BT von *Leica* ist der Raspberry Pi in der Lage, Entfernungen auf 1 mm genau zu messen, und auf 3 mm genau einzustellen. Der Laserentfernungsmesser wird mittels Bluetooth gesteuert und ausgelesen. Desweiteren steuert er den Ankerknoten über einen COM-Port und speichert die vom Ankerknoten erhaltenen Messwerte. Der komplette Messstand wird über ein ATX-Netzeil versorgt. Die Messstandsteuerung ist in Abbildung 4.8 dargestellt. Der Aufbau ist durch die Verwendung von herkömmlichem Abwasserrohr modular und ohne Weiteres auf mehr als die verwendeten 30 m erweiterbar. Ein Auschnitt des Gesamtaufbaus der alle vorgestellten Komponenten vereint, ist in Abbildung 4.9 abgebildet. Alle in dieser Arbeit vorgestellten nicht-kabelgebundenen Entfernungsmessungen wurden mit dem Entfernungsmessstand durchgeführt. Da der Raspberry Pi keinerlei zeitkritische Funktion zu erfüllen hat, sondern nur zur Steuerung des Messstandes und Speicherung der Messdaten verwendet wird, wird auf eine Charakterisierung an dieser Stelle verzichtet.

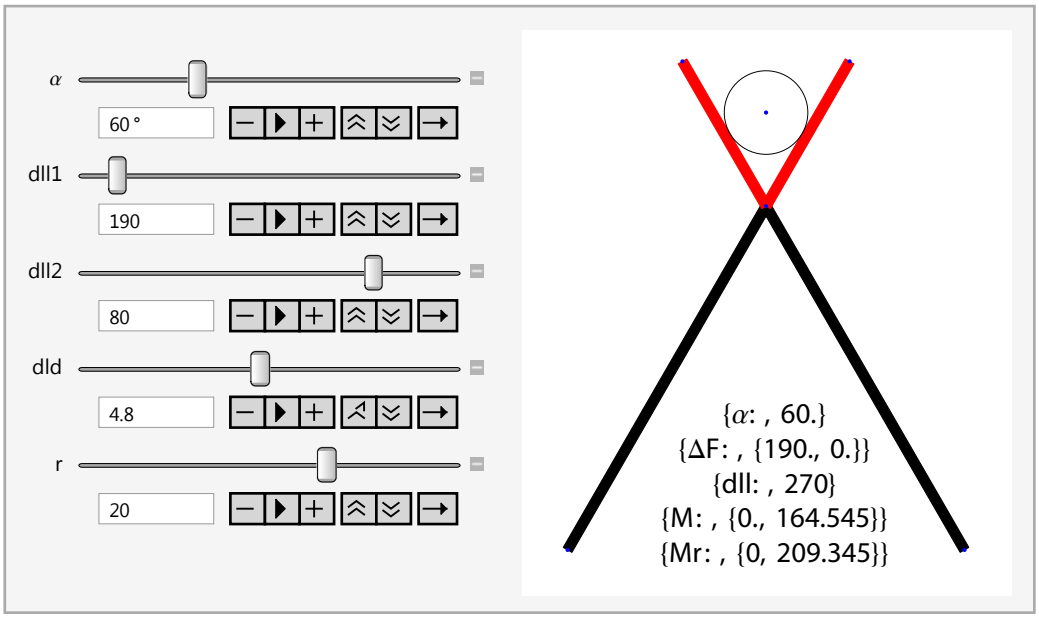


Abb. 4.6: Berechnung der Stützen mit einem erstellten Modell in *Mathematica*. Als Parameter können der Winkel α, die Längen unterhalb und oberhalb der Drehachse $dll1$ und $dll2$ sowie der Radius des Rohres r variiert werden. Die relevante Ausgangsvariable ist dann der Abstand des Rohrmittelpunkts vom Boden $M_{r,x}$.

4.3.2 Programmierung

Für den Messstand wurde Software entwickelt, die einen unbeaufsichtigten Dauermessbetrieb ermöglicht. Das Steuerungsprogramm besteht aus vier Komponenten.

- Das **Hauptprogramm** regelt den logistischen Teil der Steuerung wie das Abfragen und Verarbeiten von Nutzereingaben, die Ablaufsteuerung, die zeitgesteuerte Sicherheitsabschaltung, das Aufzeichnen, Darstellen und Speichern der Programmmeldungen und das Aufrufen der Unterprogramme. Es stellt eine Hilfefunktion zur Verfügung, anhand der sich die Konfiguration des Entfernungsmessstandes einfach durchführen lässt.

- Der **Bluetooth-Thread** steuert und konfiguriert den Laserentfernungsmesser, fragt über Bluetooth regelmäßig die Entfernung ab und stellt sie den anderen Programmteilen zur Verfügung. Bei einem Unterschreiten einer vorgebenen Messfrequenz von 1 Hz oder einer Fehlermeldung des Laserentfernungsmessers wird die Messung unterbrochen. Kann der Fehler nicht durch die automatischen Routinen behoben werden, wird die Messung abgebrochen.

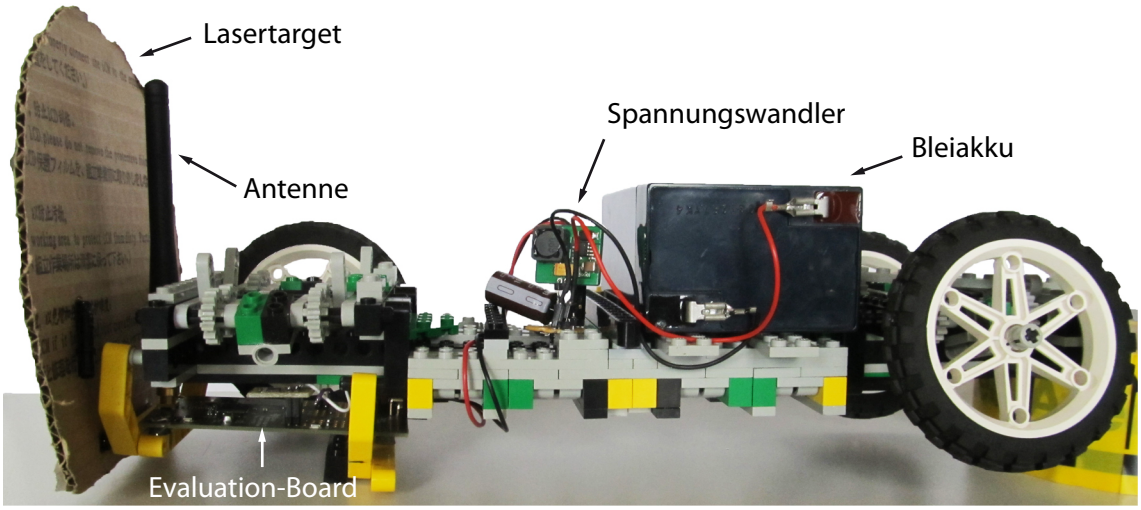
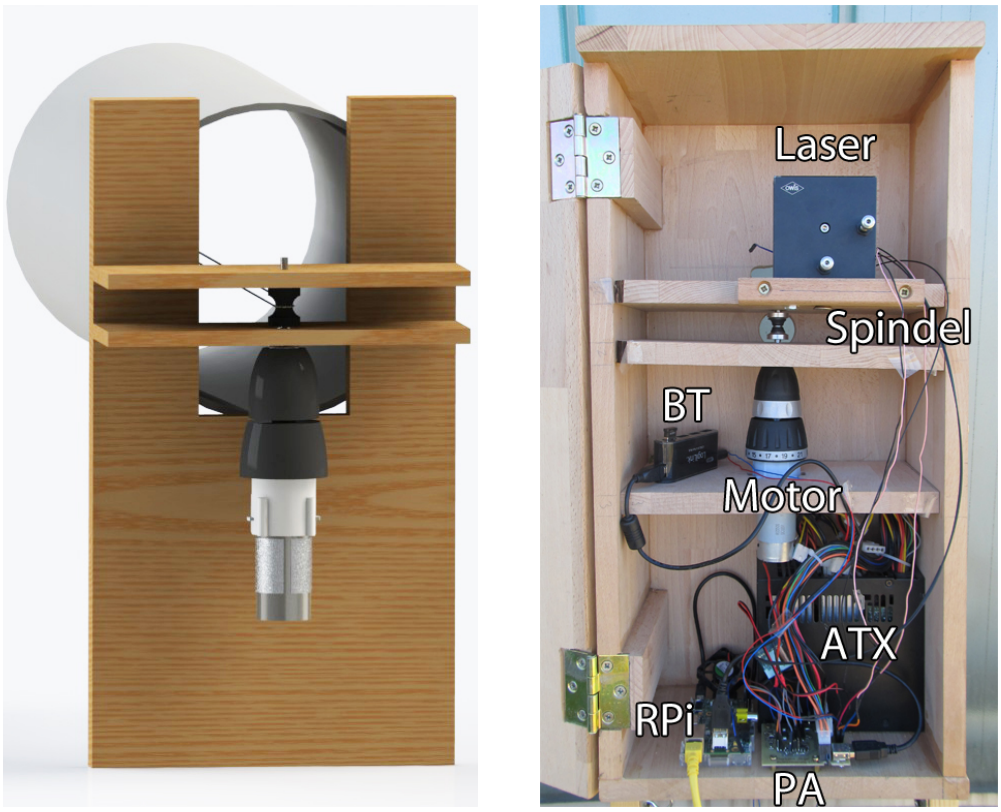


Abb. 4.7: Trägerwagen der Messbahn mit Evaluation-Board. Das vordere linke Rad wurde zur besseren Sicht auf die Konstruktion abmontiert.

- Der **PID-Regler** berechnet mit der aktuellen, vom BT-Thread zur Verfügung gestellten, Entfernung und der durch das Hauptprogramm vorgegebenen Entfernung einen Stellwert für den Regler. Dieser wird dann unter Berücksichtigung der von der Hardware limitierten Minimal- und Maximalwerte über den PWM-Port des Raspberry Pi zur Steuerung der Leistungselektronik verwendet. Ebenso regelt dieser Programmteil die technisch notwendige Anfahrhilfe für den Motor und den Ruckbetrieb für kleinste Stellweiten.

- Das **Messprogramm** sendet die Steuerbefehle an den Master-Knoten, zerlegt das empfangene Paket in die einzelnen Messdaten und sorgt für eine systematische Abspeicherung der Daten als vorformatierte datierte Textdateien in einer entsprechenden Ordnerstruktur. Es vergleicht die vom Masterknoten berechnete Prüfsumme mit der tatsächlichen Prüfsumme und fordert bei gestörter Übertragung die Daten erneut an. Bei sonstigen Fehlfunktionen führt das Messprogramm einen Neustart des Masterknotens durch und wiederholt die Messung.

Die gesamte Steuerung des Messstandes wurde in *Python* geschrieben und ist somit uneingeschränkt plattformunabhängig. Ebenso ist durch den modularen Aufbau des Messprogramms mit klar definierten Schnittstellen ein Wechsel zum Beispiel des Laserentferungsmessers oder ein Umstieg von einem normalen Elektromotor auf einen Schrittmotor problemlos möglich. Besonders einfach ist der Austausch des verwendeten Messprogramms. Der Messstand kann somit für verschiedenste Messaufgaben wiederverwendet werden.



(a) Konstruktionszeichnung (b) fertige Konstruktion

Abb. 4.8: Messstandsteuerung mit Einzelkomponenten a) Konstruktion in einem CAD-Programm zur Bestimmung der Abmessungen und Kontrolle des Konzepts b) Foto vom fertigen Aufbau mit beschrifteten Einzelkomponenten.

4.3.3 Automatisierte Auswertung

Die aufgezeichneten Messdaten wurden aufgrund der anfallenden Menge von mehreren hundert Megabyte pro Messregime durch selbst angefertigte Programme einer automatisierten statistischen Auswertung unterzogen. Der hohe Datenaufwand musste von vorne herein in die Planung miteinbezogen werden, um alle Systemteile einfach und kompatibel zu halten und die anschließende Auswertung nicht unnötig zu erschweren. Dazu wurde besonderer Wert auf eine saubere Implementierung der Schnittstellen zwischen den einzelnen Systembestandteilen gelegt.
Bei der Auswertung werden automatisch an jedem Messpunkt für alle Messparameter die absolute und die relative Standardabweichung, Median und Modus, sowie der Mittelwert berechnet. Ausreißer außerhalb eines Konfidenzintervalls von 6 σ entfernt und die Messdaten in ein, automatisch in ein Zeichenprogramm einlesbares, Format gebracht. Zudem wird bei einigen Messungen ein zusätzliches farblich kodiertes Histo-

Abb. 4.9: Aufgebaute Bahn mit Messstandsteuerung, Stützen und Wartungsstück zum Einsetzen des Wagens und Zugriff auf die mobile Plattform und den Akku.

gramm erzeugt, das die Verteilung der Messwerte anzeigt. Die Auswertung der Rohdaten wurde bei jeder Messung stichprobenhaft manuell überprüft. Für die statistische Auswertung wurde in *Python* die Statistik-Bibliothek *numpy* verwendet. Das Grundgerüst des Auswerteprogramms findet sich im Anhang B, alle verwendeten Programme befinden sich im Anhang C auf dem beigelegten Datenträger.

5 Experimentelle Durchführung

Für diese Arbeit wurden zwei verschiedene Arten von Messungen durchgeführt. Einerseits wurden kabelgebundene Messung zur Verifikation des Messprinzips, zur Kalibrierung des Offset über der Temperatur und der Dämpfung und der Bestimmung der Laufzeitabhängigkeit in einer Klimakammer ausgeführt. In einem zweiten Schritt wurde der Messaufbau als Feldversuch mit nicht-kabelgebundener Ausbreitung der Funkwellen durchgeführt. Dies diente der Überprüfung des Messprinzips unter annäherungsweise realen Bedingungen. Im Folgenden werden die Messaufbauten zur Ermittlung der Genauigkeit der Entfernungsmessung dargestellt und die durchgeführten Messungen beschrieben. Zudem werden die durchgeführten Voruntersuchungen vorgestellt, anhand derer die optimalen Parameter für die weiteren Messungen festgelegt wurden.
Da der CC430 eine sehr große Anzahl an Einstellungsmöglichkeiten bietet, musste aus der Gesamtheit der Parameter ein sinnvoller Auschnitt gewählt werden. Mit den gewählten Parametern wird das volle Spektrum der Funktionen des CC430 abgedeckt. Als Datenrate kamen die schnellste verfügbare, paketbasierte Datenrate von 250 kb/s, eine mittlere und weitverbreitete Einstellung mit 38,4 kb/s und die langsamste, auf größte Reichweiten ausgelegte Datenrate von 1,2 kb/s zum Einsatz. Von den vier verfügbaren Modulationsarten *on off keying* (OOK), *minimum shift keying* (MSK), *frequency shift keying* (FSK) und *gaussian shaped frequency shift keying* (GFSK) wurden FSK und GFSK näher untersucht, da sie am häufigsten in der Praxis eingesetzt werden. OOK wird aufgrund seiner geringen spektralen Effizienz und der hohen Bandbreitenanforderungen äußerst selten für Anwendungen eingesetzt. Der Einfluss der Frequenz wurde in zwei häufig verwendeten Frequenzbändern, dem SRD-Band bei 868 MHz und dem ISM-B-Band bei 915 MHz untersucht. Der CC430 unterstützt zwar auch Frequenzen im ISM-A-Band bei 433 MHz, jedoch waren die Evaluation-Boards auf den Frequenzbereich von 868-915 MHz angepasst. Der Tabelle 5.1 können die unterschiedlichen Einstellungen entnommen werden.

Tab. 5.1: Standardabweichung der Mittelwerte σ mit -60 ± 1 dB Dämpfung und 18 m Kabellänge.

Größe der Gruppe[1]			1	20	50	100	200	500	1000	2000	5000
σ der Mittelwerte[2,3] in m	250 kb/s GFSK	868 MHz	12.80	2.85	1.80	1.26	0.88	0.58	0.41	0.29	0.19
		915 MHz	12.83	2.87	1.82	1.28	0.91	0.55	0.37	0.24	0.15
	250 kb/s FSK	868 MHz	6.09	1.37	0.86	0.61	0.44	0.27	0.19	0.13	0.08
		915 MHz	6.26	1.40	0.88	0.62	0.44	0.28	0.21	0.14	0.09
Messdauer[5] in ms			1.4	28	70	140	280	700	1400	2800	7000
σ der Mittelwerte in m	38.4 kb/s GFSK	868 MHz	78.93	17.73	11.19	7.87	5.59	3.57	2.59	1.71	1.12
		915 MHz	78.36	17.56	11.12	7.88	5.62	3.63	2.42	1.69	1.04
	38.4 kb/s FSK	868 MHz	30.10	7.11	4.50	3.17	2.27	1.45	1.02	0.77	0.43
		915 MHz	29.85	7.04	4.55	3.20	2.25	1.41	1.03	0.77	0.49
Messdauer in ms			4.3	86	215	430	860	2150	4300	8600	21500
σ der Mittelwerte[4] in m	1.2 kb/s GFSK	868 MHz	1741.3	394.14	253.2	179.76	126.22	86.41	64.92	58.48	29.16
		915 MHz	1745.1	395.28	250.51	177.88	123.46	81.49	55.20	33.32	24.51
Messdauer in ms			110	2200	5500	11000	22000	55000	110000	220000	550000

Fehlerfarben: $2\,\text{m} < \sigma$ | $2\,\text{m} \leq \sigma < 1\,\text{m}$ | $1\,\text{m} \leq \sigma < 0.5\,\text{m}$ | $\sigma < 0.5\,\text{m}$

[1]Die Einzelmessungen wurden in Gruppen der angegebenen Größe eingeteilt und dann die Standardabweichung der Mittelwerte der Gruppen berechnet.[2]Der Fehler in der Entfernung wurde durch Multiplikation der Standardabweichung mit der Signalausbreitungsgeschwindigkeit $v_s = 9.2244\,\text{m/cycle}$ berechnet, die sich aus dem Verkürzungsfaktor aus dem Datenblatt der Kabel $v_p = 0.8$ nach Gleichung 2.7 ergibt. [3]300.000 Messungen wurden jeweils für die schnellen Datenraten aufgezeichnet. [4]50.000 Messungen wurden für die langsame Datenrate aufgezeichnet [5]Alle Werte wurden von der gemessenen Länge der Einzelmessung berechnet.

5.1 Voruntersuchungen

Zuallererst wurde untersucht, ob die Zeitmessung über das Capture-Register des CC430-Mikrokontrollers plausible Messergebnisse liefert. Dazu wurden die verwendeten Interrupts auf entsprechende Port-Pins weitergeleitet und diese Signale mit einem TDS3034B-Oszilloskop von *Tektronix* untersucht. Die Ergebnisse dieser Messung sind in Abbildung 5.1 dargestellt. Zwischen den steigenden Taktflanken des Sendens und des Empfangens liegt ein zeitlicher Offset von zweimal 13,8±0,1 μs. Dieser Offsetwert wird von den Platinen über die im vorherigen Kapitel beschriebene Zeitmessung erfasst. Diese Messung liefert einen Offset von circa 714 Zyklen was mit einer Periodendauer von 38,4 ns einem zeitlichen Offset von circa 27,4 μs entspricht. Die Zeitmessung durch das Capture-Register der Platinen ist also plausibel und auch wesentlich genauer, als die Messung mit dem Oszilloskop.

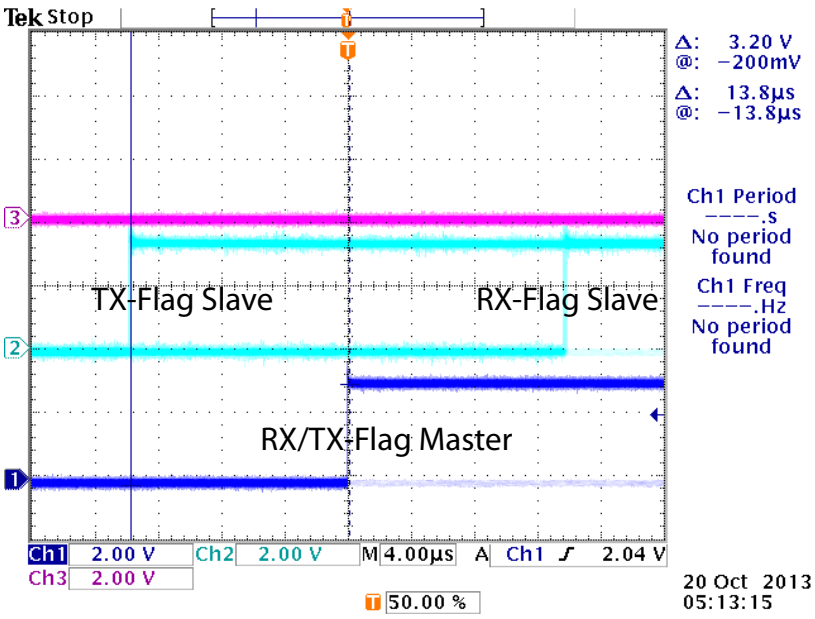


Abb. 5.1: Zeitdifferenz zwischen Empfangs- und Sende-Flanke bei der Entfernungsmessung auf den beiden Knoten. Die Zeitdifferenz zwischen dem Senden des Pakets und dem Empfangen ist sehr viel größer als das sie durch Laufzeiteffekte verursacht sein könnte. Vielmehr ist die Laufzeit durch einen zeitlichen Offset des Analogteils überlagert. Das Oszilloskop war mit den GDO1-Ausgängen der Platinen verbunden, die durch setzen des IOCFG1-Registers auf 0x06 das entsprechende Interrupt-Signal ausgeben.

Wie im Theorieteil beschrieben, ist zur genauen Entfernungsmessung ein Rauschen des Systems nötig, um über den Mittelwert von mehreren hundert Messungen einen genauen Wert für die Laufzeit berechnen zu können. Liegt das Rauschen nicht über der notwendigen Schwelle, ergibt also jede Messung den selben diskreten Messwert, so kann beispielsweise über einen Spread-Spectrum-Quarz das Taktsignal künstlich verrauscht werden. Dies führt zwar nicht zu einer Erhöhung des Rauschens im Eingangssignal, hat aber aus signaltheoretischer Sicht den selben Effekt.
Um diese Frage zu klären, wurden statistische Untersuchungen vor den eigentlichen Messungen durchgeführt. Dafür wurden für jede, der Tabelle 5.1 entnehmbahre Einstellung, 300.000 beziehungsweise 50.000 Einzelmessungen durchgeführt, um eine ausreichend große Datenbasis zu erhalten. Um die Genauigkeit der Messungen abschätzen zu können, wurden die Einzelmessungen in Gruppen zunehmender Größe eingeteilt, der Mittelwert berechnet und die Standardabweichung der Mittelwerte betrachtet. Die Ergebnisse dieser Untersuchung können Tabelle 5.1 entnommen werden. Am deutlichsten sticht der Unterschied zwischen den verschiedenen Datenraten ins Auge. Schnellere Datenraten erlauben es, mehr Messungen in kürzerer Zeit durchzuführen. Zudem machen sich Fehlerquellen, die sich über die Zeit addieren, aufgrund der längeren Messzeit

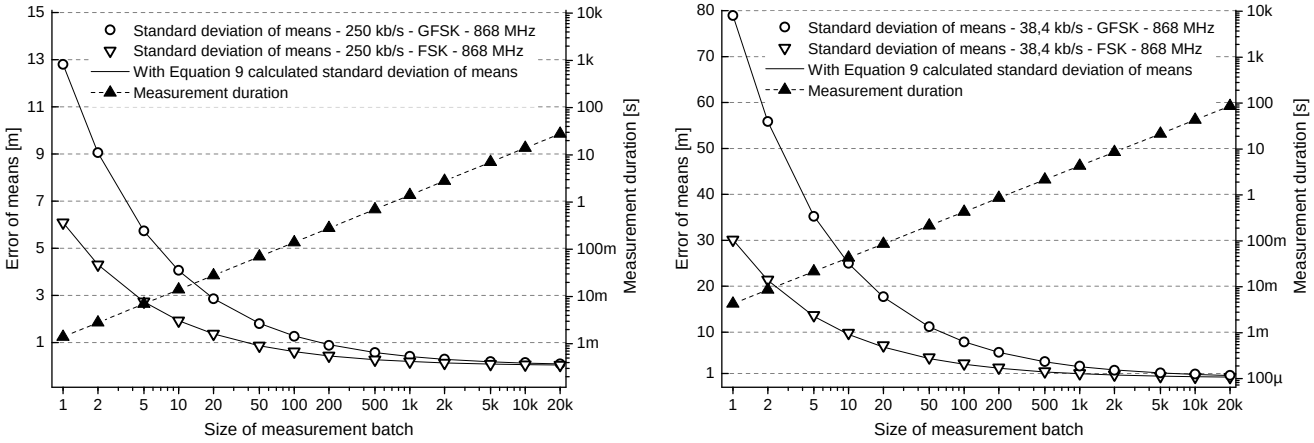


Abb. 5.2: Einfluss der Gruppengröße auf den Messfehler und die Messdauer für a) Datenrate von 250 kb/s und b) Datenrate von 38,4 kb/s.

stärker bemerkbar. Es sollte daher stets die schnellste verfügbare Datenrate für die Messungen verwendet werden, und nur wenn Übertragungsfehler aufgrund zu geringer Empfangsfeldstärke auftreten auf eine langsamere Datenrate ausgewichen werden. In Abbildung 5.2 ist das Verhältnis aus Gruppengröße, Standardabweichung der Mittelwerte und Messdauer dargestellt.

In Abbildung 5.3 sind einige der Ergebnisse der Voruntersuchung dargestellt. Die Ergebnisse der Laufzeitmessungen unterliegen statistischen Schwankungen die von der Normalverteilung abweichen. Die Verteilung entspricht einer Rayleigh-Verteilung, was auf eine Überlagerung von zwei unabhängigen Zufallsvariablen schließen lässt. Da die Verteilung allerdings nicht sehr ausgeprägt ist, wird für alle folgenden Untersuchungen eine Gleichverteilung angenommen. Der Unterschied der Mittelwerte zwischen Rayleigh- und Standardverteilung beträgt durch die schwache Ausprägung nur wenige Promille und rechtfertigt daher nicht den erheblich größeren Berechnungsaufwand.

Die verwendete Frequenz hat praktisch keinen Einfluss auf die Messgenauigkeit. Sie kann anhand der lokalen und gesetzlichen Gegebenheiten frei gewählt werden. Stark bemerkbar macht sich allerdings die Modulationsart. Das GFSK-Verfahren zeigt eine um den Faktor zwei höhere Standardabweichung der Mittelwerte, es muss also öfter gemessen werden, um die selbe Präzision zu erreichen. Anhand dieser Ergebnisse konnten die Parameter für die weiteren Messungen abgeschätzt werden. Ebenso kann eine Kosten-Nutzen-Abschätzung durchgeführt werden, da eine größere Anzahl an Messungen mehr Zeit und damit mehr Energie benötigt, welche in energieautarken Sensorsystemen einen limitierenden Faktor darstellt. Dies war allerdings keine Problemstellung der vorliegenden Arbeit.

Um die Präzision der Zeitmessung zu erhöhen und die Standardabweichung zu verringern wurde versuchsweise der 26 MHz-Quarzoszillator der Platinen entfernt und der Takt für den HF-Teil durch einen Signalgenerator eingespeist. Durch die externe synchrone Taktung hat der Freqenzdrift der Quarze keinen Einfluss mehr auf die Messung. Der Signalgenerator liefert eine äußerst präzise und stabile Frequenzreferenz mit einem sehr geringen Phasenrauschen. Das Signal mit einer Amplitude von 900 mV wurde durch einen einfachen Leistungsteiler aufgespalten, der Arbeitspunkt auf die halbe Versorgungsspannung eingestellt und dieses Signal auf den Platinen in den XT2-Eingang eingespeist. Die Kommunikation zwischen den Evaluation-Boards funktionierte mit diesen Einstellungen einwandfrei, lediglich der hohe Signalpegel der Referenzfrequenz führte zu Einstreuungen auf der ganzen Platine. Einstreuungen über das Kabel zum Rücksetzen der Platine führten in unregelmäßigen Abständen zu einer Unterbrechung der Messung. Es war daher notwendig, dieses Kabel zusätzlich abzuschirmen und mit einem Eingangsfilter zu versehen. In Abbildung 5.4 ist ein Vergleich zwischen den Messungen mit und ohne externen Takt dargestellt. Die Messungen unter Verwendung des internen Quarz-Oszillators zeigen überraschenderweise eine deutlich geringere Standardabweichung der Messwerte. Dies wird darauf zurückgeführt, das durch die starre Frequenzreferenz interne Mechanismen zur Stabilisierung der Quarze ins Leere laufen. Zudem ist denkbar, dass die Phasenregelschleife, die normalerweise auf die einfallende Trägerfrequenz einrastet, durch die fehlende Rückwirkung auf den Quarz nicht mehr in der Lage ist die Frequenz präzise nachzuführen. Dies hätte zur Folge, dass das heruntermodulierte IF-Signal gegenüber den internen, sehr schmalbandigen Filtern fehlangepasst ist und somit die Abtastung fehleranfällig wird.
Die Ergebnisse der Voruntersuchung erlaubten es, die Menge an Parametern für die folgenden Messungen zu verringern. Die langsame Datenrate von 1,2 kb/s wird nicht weiter untersucht, da die erreichbare Präzision deutlich zu gering ist. Ebenfalls wird bei der mittleren Rate nur noch das Modulationsverfahren ohne Gauß'sche Filterung verwendet. Ein Vergleich der beiden Modulationsverfahren wird nur noch für die schnelle Datenrate durchgeführt. Die externe Taktung erfüllte nach der Analyse der Untersuchungen nicht die Erwartungen an eine präzisere Messung. Die weiteren Messungen wurden daher ohne eine externe Taktung durchgeführt.

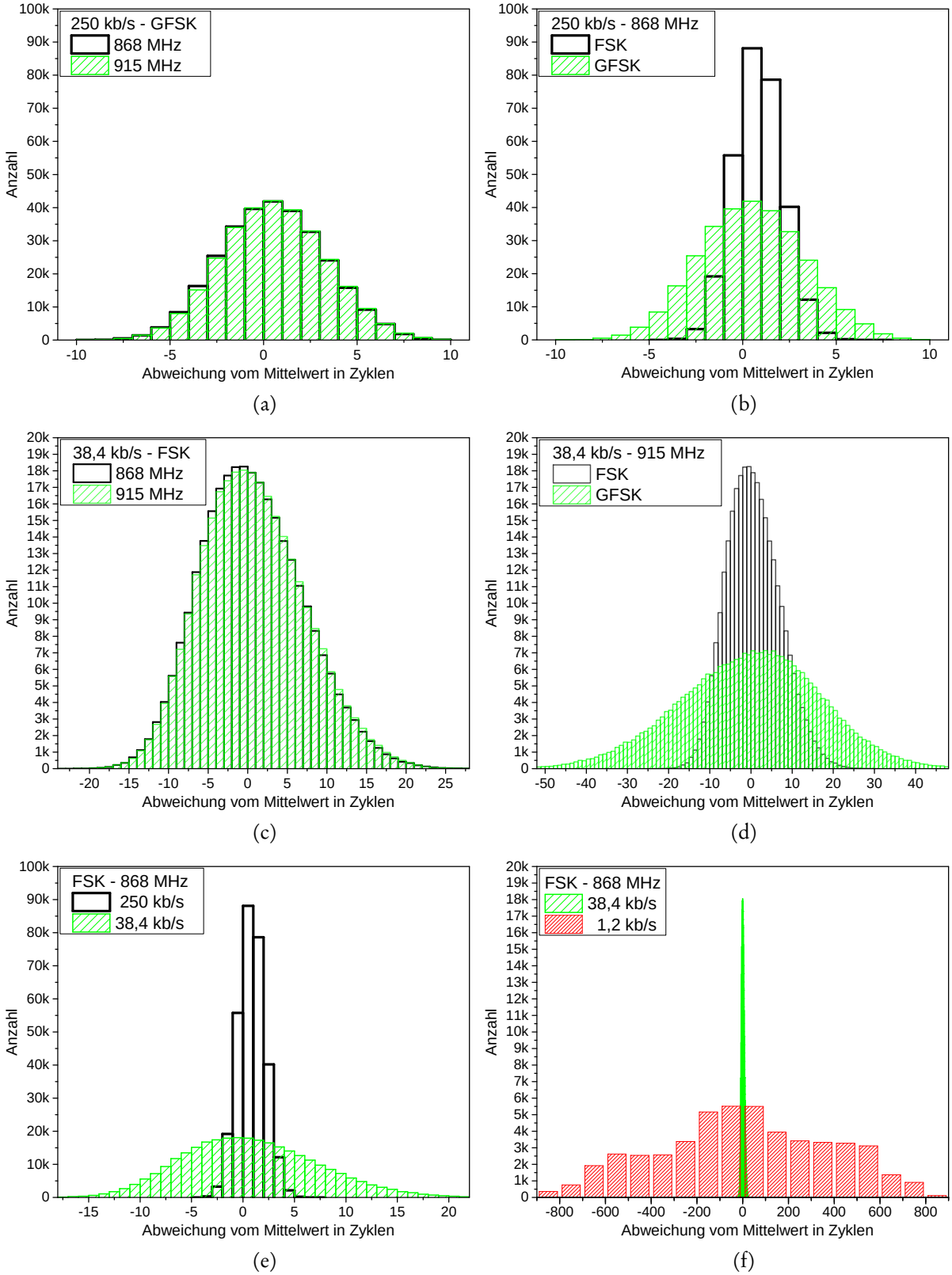


Abb. 5.3: Überblick über die Ergebnisse der statistischen Voruntersuchungen a,c) Vergleich zwischen den zwei Trägerfrequenzen 868 MHz und 915 MHz. b,d) Vergleich zwischen den beiden Modulationsverfahren FSK und GFSK. e,f) Vergleich zwischen den Datenraten.

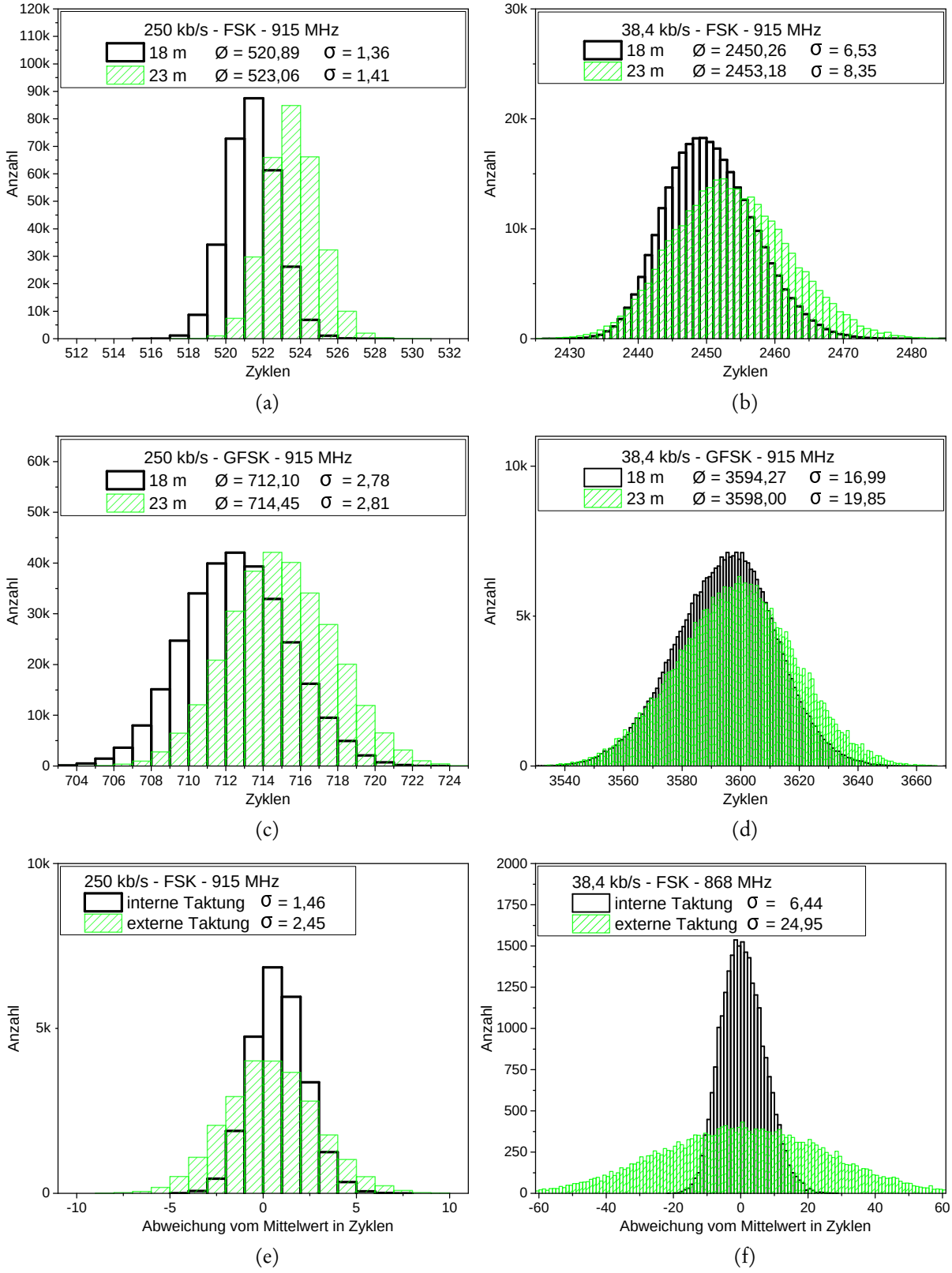


Abb. 5.4: Statistische Voruntersuchung der Entfernungsmessung für zwei verschiedene Distanzen 18 m und 23 m und beide Modulationsarten, sowie Vergleich zwischen internem und externem Taktgenerator. a,c,e) für eine Datenrate von 250 kb/s. b,d,f) für eine Datenrate von 38,4 kb/s.

5.2 Referenzmessungen in der Klimakammer

Bei den Referenzmessungen in der Klimakammer sollen verschiedene Einflussparameter auf den von *Schill et al.* beschriebenen und in dieser Arbeit ebenfalls aufgetretenen Offset im Analogteil des Transceivers untersucht werden [22]. Dazu werden die Evaluation-Boards wie in Abbildung 5.5 dargestellt in einen Messstand verbracht und verschiedene Parameter untersucht. Die Kommunikation zwischen den Platinen erfolgt kabelgebunden, um Störeinflüsse weitestgehend auszuschließen und verschiedene Parameter isoliert untersuchen zu können. Die geringe Dämpfung der direkten Kabelverbindung macht den Einsatz eines Dämpfglieds notwendig. Für die Referenzmessungen wurde soweit nicht anders vermerkt, eine Dämpfung des Eingangssignals um -60 dB vorgenommen. Die Versorgung der Platinen erfolgt mit einer stabilisierten Eingangsspannung von 3300 mV. Diese Spannung ist eine der Standardspannungen für Mikrokontroller und wurde gewählt, um eine Vergleichbarkeit zu anderen Arbeiten zu gewährleisten. Die Spannung wurde während der Messungen ständig überwacht und lag abhängig von der Raumtemperatur maximal um ±0,5 mV neben der eingestellten Spannung.

5.2.1 Temperaturzyklus

Um die Temperaturabhängigkeit des Offsets der Platinen zu untersuchen, wird ein lineares Temperaturprofil von -28 °C bis 50 °C über einen Zeitraum von 6,5 h durch den Klimaschrank gefahren, wie in Abbildung 5.6 dargestellt. Dies entspricht einem Temperaturgradienten von 0,1 $^{°C}/_{min}$. Das Messintervall lag je nach durchgeführter Anzahl an Einzelmessungen der Platinen bei 2-4 Messungen pro Minute. Damit beträgt die Temperaturänderung pro Messpunkt 0,025-0,05 °C und ist somit hinreichend fein aufgelöst. Um eventuelle Hysterese-Effekte ebenfalls zu erfassen, wird der Zyklus anschließend auch rückwärts durchlaufen. Die Platinen werden vor der Messung bei einer Raumtemperatur von 21 °C initialisiert. Die Raumtemperatur schwankt während den Messungen um circa ±0,5 °C.

5.2.2 Variation der Versorgungsspannung

Die Versorgungsspannung wurde in 20 mV-Schritten zwischen 3200 mV und 3400 mV variiert. Tiefere Versorgungsspannungen bis 1800 mV werden zwar prinzipiell vom Mi-

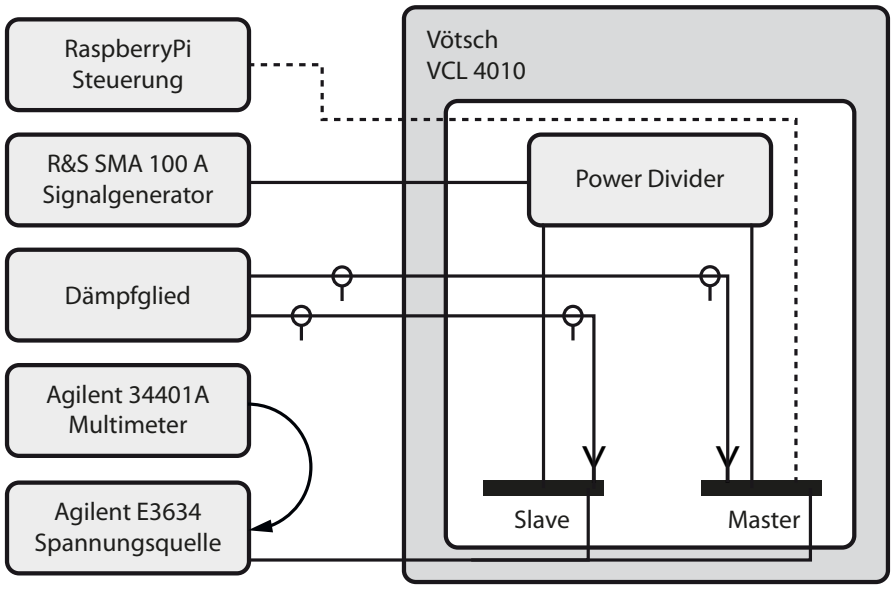


Abb. 5.5: Messaufbau für die Referenzmessung im Klimaschrank mit externer synchroner Taktung durch einen Signalgenerator.

krokontroller unterstützt, hätten aber Änderungen in der Konfiguration des internen Spannungsreglers und des HF-Teils notwendig gemacht. Um die Vergleichbarkeit mit den anderen Ergebnissen dieser Arbeit zu erhalten, wurde auf diese Rekonfiguration verzichtet. Zudem ist 3300 mV die Standardspannung die üblicherweise für Sensorknoten zur Verfügung steht und ist somit von besonderem Interesse. Für alle anderen Versuche wurden die Evaluation-Boards deshalb auch stets mit einer stabilisierten Gleichspannungsversorgung von 3300 mV betrieben.

5.2.3 Variation der Dämpfung

Um den Einfluss der Dämpfung auf die gemessenen Werte zu erfassen, wurden beide Platinen bei einer konstanten Temperatur von 21 °C im Klimaschrank aufbewahrt. Ein Dämpfglied wurde zur Variation der Dämpfung verwendet und S21 mittels eines 4345A Network Analyzers von *Agilent* vermessen. Das Übertragungsverhalten des Dämfgliedes ist sehr linear. Die zuerst zur Messung eingesetzten RG58-Kabel zeigten aber ein im Frequenzbereich periodisches und damit nicht tolerierbares Übertragungsverhalten. Zum Einsatz kamen stattdessen spezielle RG223-Hochfrequenzkabel.
Die Dämpfung wurde dann in 1 dB-Schritten zwischen -36 dB und -80 bis -110 dB variiert. Hinzu kamen 1 dB Dämpfung durch die Verbindungskabel und Steckerübergänge. Die Grenzen der Dämpfung wurden experimentell ermittelt. Dazu wurde die

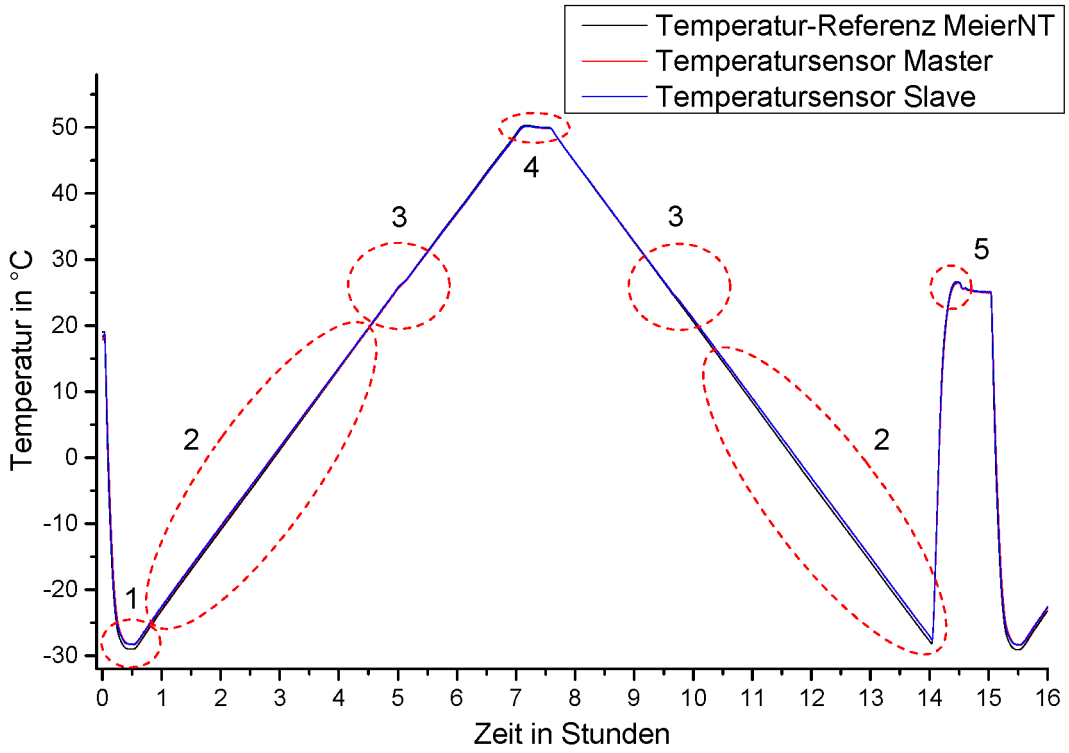


Abb. 5.6: Gefahrener Temperaturzyklus in der Klimakammer gemessen mit den Temperatursensoren auf den Evaluation-Boards und einem analogen Referenzthermometer. Nach dem Abkühlen von Raumtemperatur auf -28 °C wird die Temperatur mit 0,1 °C pro Minute auf 50 °C erhöht. Interessante Punkte: 1) Hysterese-Fehler aufgrund geringerer thermischer Masse des analogen Temperatursensors. 2) Steigerungsabweichung beziehungsweise Verstärkungsfehler zwischen Referenztemperatursensor und Temperatursensoren auf dem Evaluation-Board. 3) leichter Überschwinger des Klimaofens, vermutlich aufgrund der Zuschaltung einer zweiten Heizstufe. 4) Überschwinger nach Erreichen der Maximaltemperatur; klingt im Laufe von 60 Minuten ab. 5) Starker Überschwinger durch schnelle Regelung auf Raumtemperatur.

Dämpfung variiert, bis keine Kommunikation zwischen den Knoten mehr möglich war. Bei der unteren Grenze der Dämpfung geraten die Knoten in Sättigung, was die Demodulation zuerst stört, und anschließend unmöglich macht. Bei der oberen Grenze der Dämpfung ist die empfangene Signalstärke und damit der Signal-Rausch-Abstand (SNR) zu gering, um noch ausreichend störungsfrei verstärkt und demoduliert zu werden. Hierbei ist die Grenze um so höher, je niedriger die Datenrate der Funkkommunikation ist.

5.2.4 Variation der Kabellänge

Die Kabellänge wird variiert, um den Einfluss der Signallaufzeit auf das gewonnene Signal zu untersuchen und mit dem theoretischen Wert zu vergleichen. Dazu wird die Kabellänge in 5 m-Schritten variiert. Mit den 1 m langen Verbindungskabeln zwischen

den Platinen und dem Dämpfglied, sowie dem zusätzlichen Kabel zum Anschluss der *elspec*-Kabel, ergeben sich somit die Kabellängen 2, 8, 13, 18, 23, 28, 33, 38, 43, 48, 53, und 58 m. Die Kabeldämpfung wurde vor jeder Messung gemessen und mit dem Dämpfglied eine einheitliche Dämpfung von 60 dB ± 0,5 dB eingestellt. So konnte isoliert der Einfluss der Entfernung auf die Messungen untersucht werden.

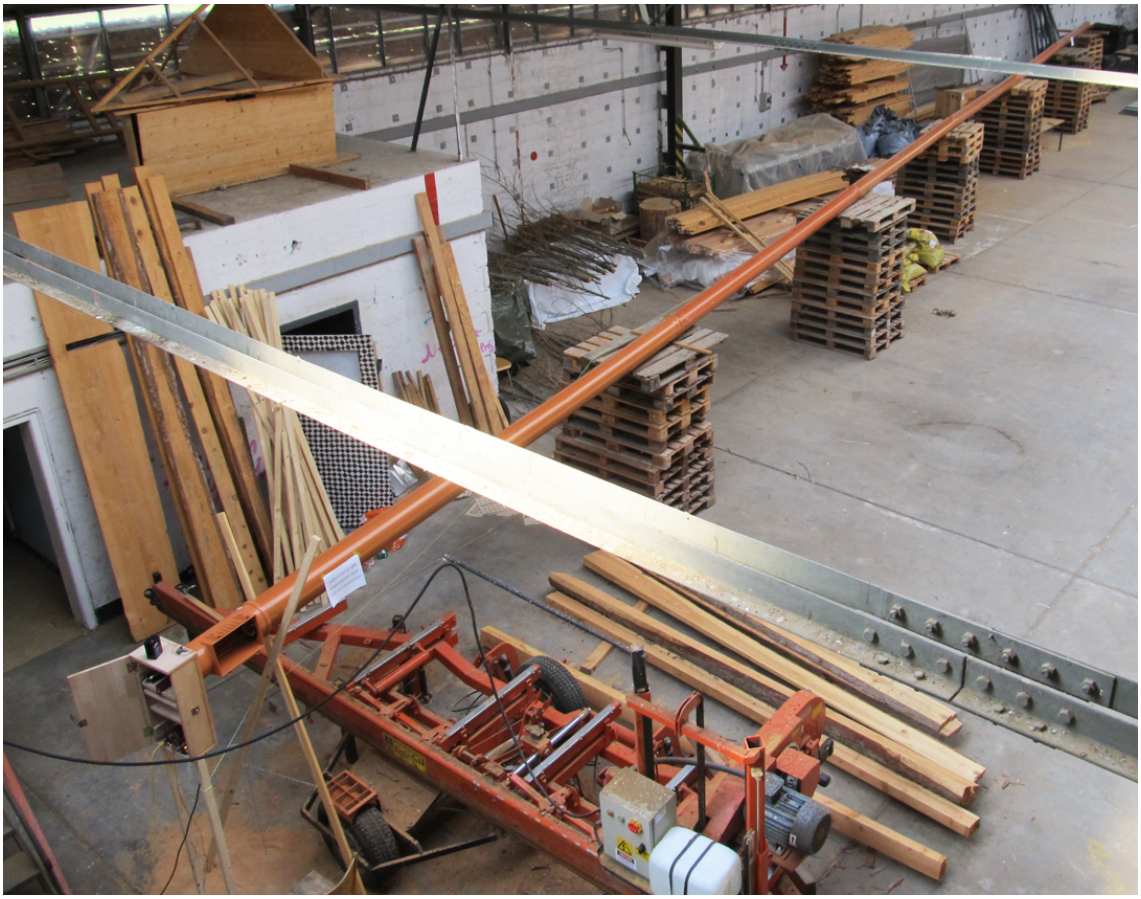


Abb. 5.7: Aufgebaute Bahn im Innenraum. Die Umgebung des Messstandes ist vergleichbar zu den Verhältnissen auf einer Baustelle. Die metallischen Hallenelemente reflektieren das HF-Signal und führen zu destruktiver Interferenz.

5.3 Feldversuche

Mit den aus den Referenzmessungen gewonnen Erkenntnissen wurden die Feldversuche konzipiert. In dieser Umgebung lassen sich die verschiedenen Parameter nicht mehr isoliert voneinander betrachten, da insbesondere die Dämpfung direkt von der Umgebung und der Entfernung beeinflusst wird. Daher wird nur noch die Entfernung variiert. Die Versorgungsspannung als einziger direkt beeinflussbarer Parameter wird konstant

bei 3300 mV gehalten. Die variablen Parameter wie Dämpfung, Laufzeit, Temperatur und Entfernung werden an jedem Messpunkt aufgezeichnet. Die Messungen werden im Messstand durchgeführt, der in Kapitel 4 vorgestellt wurde. Dieser wird zuerst im Innenraum und anschließend im Freien aufgebaut. Um den Einfluss von Bodenreflexionen zu minimieren, musste der Messstand einen Mindestabstand zum Boden aufweisen. Der größte Radius der ersten Fresnell-Zone beträgt bei 30 m Bahnlänge 1,13 m. Daher wurde der Messstand jeweils in 1,90 m Höhe über dem Boden aufgebaut.
Als Antennen kamen λ/4-SMA-Antennen von *taoglas* mit Rundstrahlcharakteristik und 0 dBi Antennengewinn zum Einsatz. Die Antennen wurden parallel zueiander und senkrecht zum Boden ausgerichtet. Der gewünschte Messpunkt wurde vom Messstand angefahren und die Position für die Dauer der Messungen gehalten. Pro Messpunkt wurden zu Beginn der Messungen 25.000 Einzelmessungen durchgeführt, um eine kleine Varianz der Mittelwerte zu erhalten.

5.3.1 Innenraum

Abb. 5.8: Aufgebaute Bahn im Freifeld zur Sicherstellung einer möglichst ungestörten Messung.

Die Innenraummessung fand in einem Wirtschaftsgebäude der Universität Freiburg statt, wie in Abbildung 5.7 dargestellt. Dieses wird vorwiegend zur Lagerung von überzähligem Baumaterial und Holzschnitten, sowie Gerätschaften der forstwissenschaftlichen Fakultät verwendet. Des Weiteren sind in der etwa 70 m langen Halle Bauzäune, eine vollautomatische Bandsäge und Hochregale vorhanden. Die Halle selbst besteht

als ehemaliger Hangar aus einer Metallgerüstkonstruktion mit eingezogenen Dachstreben, Stahl- und Steinwänden und einer großen, massiven und stählernen Hangartür. Die Umgebung ist dadurch äußerst störanfällig und gekennzeichnet von starken Mehrwegausbreitungen und Interferenzmustern. Sie kommt somit den tatsächlichen Gegebenheiten, beispielsweise auf einer Großbaustelle, sehr nahe. Die Lagerung des Messtandes erfolgte aus aufbautechnischen Gründen auf Holzpaletten, und nur zum Teil aus den angefertigten Ständern. Das zusätzliche Material innerhalb der ersten Fresnell-Zone führt zu einer stärkeren Dämpfung des Signals, wie es ebenfalls auf Baustellen vorkommen kann.
Bei den zuerst durchgeführten Messungen wurde eine Entfernungsintervall von 1 m gewählt, sowie die Anzahl an Einzelmessungen auf 25.000 festgelegt. Die Ergebnisse dieser Messung waren jedoch aufgrund starker Schwankungen durch die Mehrwegausbreitung nicht fein genug aufgelöst. Um das Interferenzmuster innerhalb der Halle daher ebenfalls zu erfassen, wurde ein Messintervall von 22 mm gewählt. Dies entspricht einem Achtel der Wellenlänge des verwendeten Trägers von 868 MHz. Damit ist das Nyquist-Kriterium für beide Trägerfrequenzen erfüllt. Aufgrund der hohen Anzahl an Messpunkten, wurde die Anzahl an Einzelmessungen pro Messpunkt auf 5000 reduziert, auch um die Länge der Gesamtmessung zu reduzieren. Diese lag in dieser Konfiguration bei circa 24 h für ein komplettes Messregime. Wie in den Voruntersuchungen gezeigt, beträgt der Fehler bei dieser Anzahl an Einzelmessungen lediglich 9 cm und war somit für eine Charakterisierung des Entfernungsverhaltens ausreichend.
Insgesamt wurden jeweils zwei Messungen für die schnellen Datenraten von 250 kb/s durchgeführt. Es wurden jeweils beide Trägerfrequenzen mit der FSK-Modulation verwendet. Bei der mittleren Datenrate von 38,4 kb/s kam nur der 868 MHz-Träger mit der FSK-Modulation zum Einsatz.

5.3.2 Freifeld

Die Freifeldmessungen fand auf dem weitläufigen Gelände der Technischen Fakultät statt. Die Umgebung des Messstandes war bis zu einem Umkreis von 30 m frei von Hindernissen. Lediglich ein kleiner Baum war nur 10 m von der Messstandsteuerung entfernt. Die Messstand wurde durch die in Kapitel 4 beschriebenen Stützen getragen und auf eine durchgehende Höhe von 1,90 m justiert. Der aufgebaute Messstand ist in Abbildung 5.8 dargestellt.
Das Entfernungsintervall betrug wie im Innenraumversuch 22 mm. Es wurde für die schnelle Datenrate eine Messung mit einer Trägerfrequenz von 915 MHz und der FSK-Modulation durchgeführt.

6 Ergebnisse und Diskussion

In diesem Kapitel werden die Ergebnisse der verschiedenen durchgeführten Messungen vorgestellt und hinsichtlich ihres Anwendungsbezuges diskutiert, sowie Empfehlungen für den Aufbau eines Systems gegeben. Zuerst werden die Ergebnisse der Referenzmessungen vorgestellt und im Einzelnen diskutiert. Anschließend werden die Ergebnisse der Feldmessungen untersucht und besprochen. Zuletzt werden die Ergebnisse sämtlicher Messungen nochmals zueinander in Relation gesetzt und eine abschließende Empfehlung für das weitere Vorgehen gegeben.

6.1 Referenzmessungen

Die im vorhergehenden Kapitel beschriebenen kabelgebundenen Referenzmessungen liefern Ergebnisse zur Charakterisierung des Verhaltens des Entfernungs-Messsystems. Von besonderer Bedeutung ist das thermische Verhalten des Schaltkreises, da es den größten Einfluss auf die gemessenen Parameter aufweist. Die Messergebnisse werden zuerst isoliert voneinander betrachtet und anschließend die gegenseitigen Abhängigkeiten erläutert.

6.1.1 Versorgungsspannung

In Abbildung 6.1 sind die Ergebnisse der Variation der Versorgungsspannung dargestellt. Die verschiedenen Funkeinstellungen liefern jeweils einen anderen Offset. Die Messergebnisse wurden daher durch Subtraktion des Mittelwerts der jeweiligen Messung normiert. Besonders auffällig ist das Verhalten des Transceivers bei 3220 mV. Der Offset ist um circa drei Zyklen gegenüber den anderen Messungen erhöht. Dies wird durch das Verhalten des internen Spannungsreglers erklärt. Dieser schaltet intern zwischen verschiedenen Eingangsspannungsbereichen, um die interne Chipspannung konstant zu halten. An bestimmten Punkten wird die interne Chipspannung erhöht, um beispielsweise den erhöhten Energiebedarf der Funkendstufe zu bedienen. Eine solche

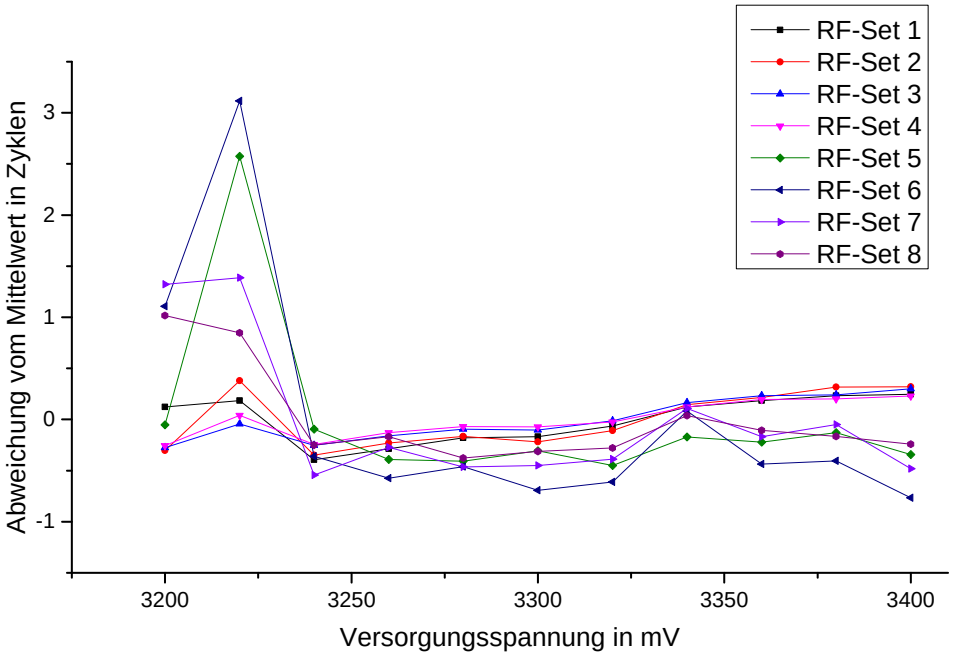


Abb. 6.1: Abhängigkeit des Offsets von der Versorgungsspannung. Dargestellt sind die normierten Abweichungen vom Mittelwert.

Schaltschwelle liegt laut Datenblatt bei 3200 mV. Unter Vernachlässigung der ersten beiden Messwerte ergibt sich damit eine versorgungsspannungsabhängige Änderung des Offsets von 4,28 Zyklen/V. Umgerechnet in Entfernung entspricht dies einem Fehler von 4,9 cm/mV. Es ist also für den Entwurf eines System notwendig, die Versorgungsspannung zeitlich stabil zu halten, da sonst Fehler in der Entfernungsmessung auftreten. Die kontinuierliche Erhöhung der Offsets kann auf eine Erhöhung der Verlustleistung des Chips zurückgeführt werden. Diese hängt durch $P = U^2 \cdot R^{-1}$ quadratisch von der Eingangsspannung ab. Dadurch haben kleine Änderungen der Eingangsspannung bereits große Auswirkungen auf die Verlustleistung und damit auf den Temperaturhaushalt des Chips. Wie die Messungen der Temperaturabhängigkeit zeigen, erhöht sich der Offset in manchen Temperaturbereichen bei Erhöhung der Temperatur. Dieses Verhalten könnte auch das Verhalten der Transceiver bei einer Erhöhung der Spannung erklären.

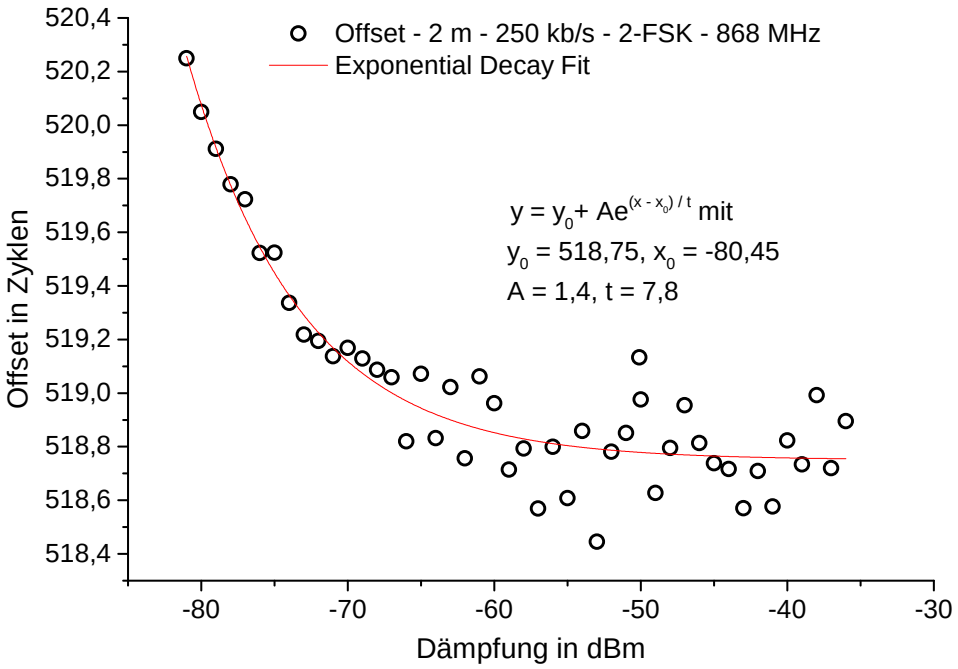


Abb. 6.2: Einfluss der Dämpfung auf die Laufzeitmessungen bei einer Datenrate von 250 kb/s und FSK-Modulation bei einer Trägerfrequenz von 868 MHz und konstanter Temperatur von 20 °C. Die Messung wurde mit einem exponentiellen Zerfall approximiert. Dessen Parameter sind der Grafik zu entnehmen.

6.1.2 Dämpfung

Die Empfangsfeldstärke sollte theoretisch keinen Einfluss auf die Laufzeit haben, jedoch haben die Messungen ergeben, das der Offset dämpfungsabhängig ist und somit eine Rückwirkung auf die Entfernungsmessung besitzt. In Abbildung 6.2 ist das Verhalten des Offsets gegenüber der Dämpfung dargestellt. Der Verlauf zeigt ein exponentiell abfallendes Verhalten mit abnehmender Dämpfung. Über den gesamten Pegelbereich des Eingangssignals in dem eine Kommunikation zwischen den Evaluation-Boards möglich ist, beträgt der Unterschied im Offset 0,4 Zyklen. Umgerechnet in Entfernung entspricht dies einem Fehler von 4,6 m. Es ist daher notwendig, den Eingangssignalpegel bei der Entfernungsmessung mitzuerfassen, um eine Korrektur des Offsets durchzuführen.

Die Ursache des Verhaltens könnte darin liegen, dass durch den geringen Eingangssignalpegel Kondensatoren innerhalb des Analogteils langsamer geladen werden und somit langsamer Schaltschwellen des Digitalteils erreicht werden. Ebenso ist denkbar, dass der Einschwingvorgang der Demodulation eine längere Zeit in Anspruch nimmt. Dafür spricht auch, dass das Verhalten bei der mittleren Datenrate nur in schwächerer

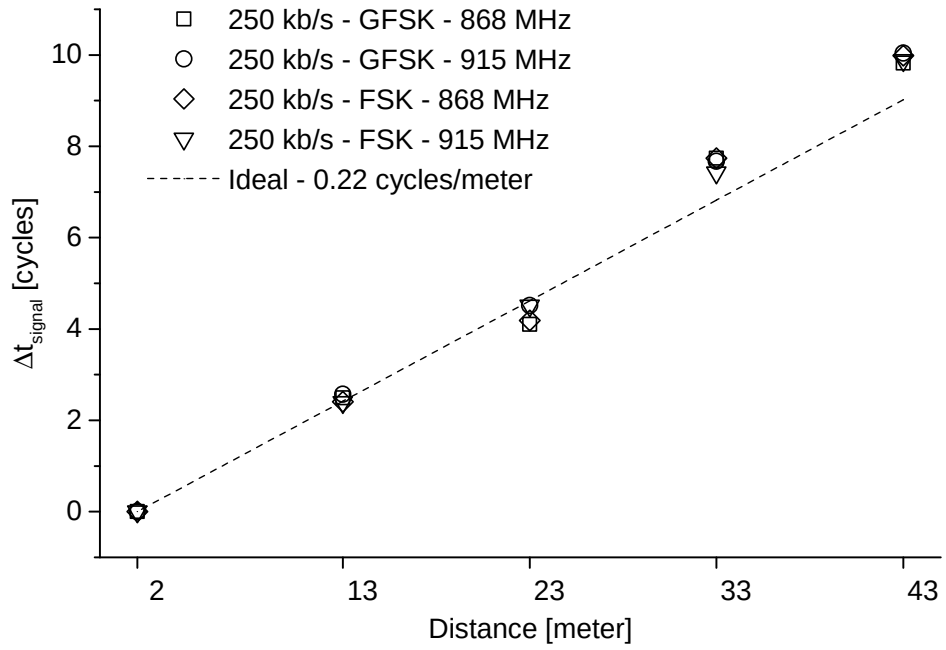


Abb. 6.3: Ergebnisse der Entfernungsmessung. Die Idealline hat eine Steigung von 0.22 Zyklen/m die sich aus dem Verkürzungsfaktor des Kabels von 0,8 ergibt. Die Standardabweichung der Punkte ist kleiner als die Symbolgröße und ist daher nicht eingezeichnet.

Form auftritt, da hier mehr Zeit für die Demodulation zu Verfügung steht und die Elektronik bei den hier untersuchten Signalpegeln nicht im Grenzbereich arbeitet.

6.1.3 Entfernung

Die Referenzmessung der Entfernung wurde verwendet, um zu prüfen, ob in dem um drei Größenordnungen größeren Signal der Laufzeitmessung überhaupt die tatsächliche Laufzeit enthalten ist. In Abbildung 6.3 sind die Ergebnisse der Entfernungsmessung dargestellt. Die Ergebnisse der verschiedenen Funkeinstellungen liegen alle sehr dicht beieinander, jedoch liegen die Abweichungen von der Ideallinie zwischen den einzelnen Entfernungsmessungen im Bereich von einem Zyklus. Dies entspricht einem Fehler der Entfernungsmessung von 11,53 m. Aufgrund der relativ langen Messdauer, es wurden nebenher auch die Werte für die Dämpfung variiert, liegen die Messpunkte der Entfernungsintervalle zeitlich circa 24 h auseinander. Es wurde daher geschlussfolgert, dass Umgebungsparameter, wie beispielsweise die schwankende Labor-Temperatur, einen direkten Einfluss auf die Messung haben. Daher wurde in einem nächsten Schritt die Temperaturabhängigkeit des Offset untersucht. Tatsächlich ergaben die ersten vorläufigen Messungen eine umgekehrt proportionale Temperaturabhängigkeit, wie sie in Ab-

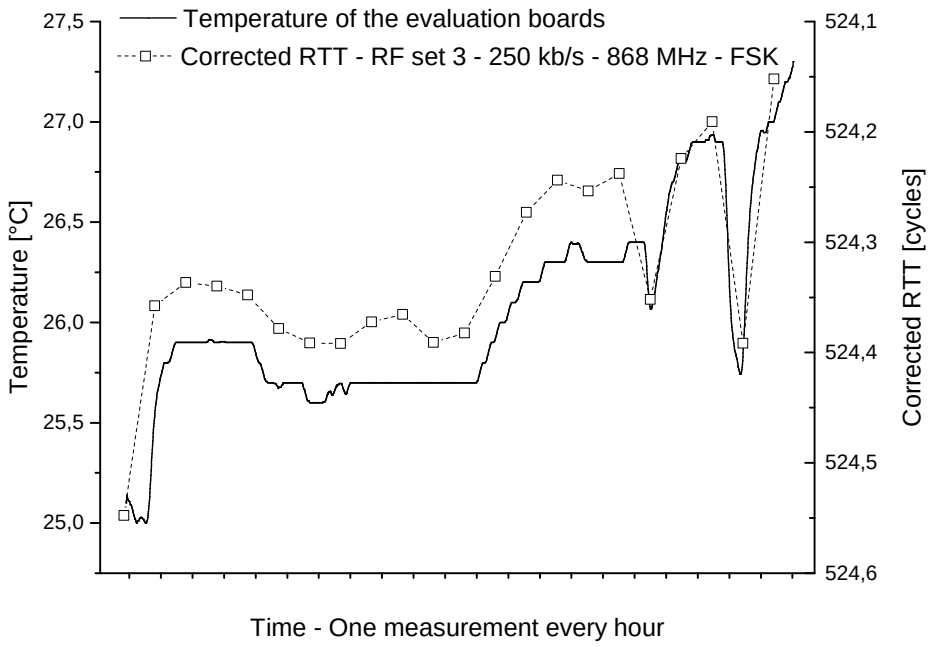


Abb. 6.4: Abhängigkeit der Entfernungsmessung von der Raumtemperatur. Um die Abweichungen der Entfernungsmessung von der Temperatur zu erklären, wurde eine 24-stündige Messung des Offsets und der Raumtemperatur durchgeführt. Zur besseren Veranschaulichung ist die rechte Skala umgekehrt aufgetragen.

bildung 6.4 dargestellt ist. Mit dieser Abhängigkeit kann die Abweichung von der Ideallinie der Entfernungsmessung durch die täglich schwankende Raumtemperatur erklärt werden. Im folgenden Abschnitt wird die Temperaturabhängigkeit daher genau quantifiziert, um den Einfluss auf die Entfernungsmessungen präzise erfassen zu können.

6.1.4 Temperatur

Die in der vorherigen Referenzmessung festgestellte Temperaturabhängigkeit wurde genauer untersucht. Aufgrund des aufgetretenen Verhaltens wurde von einem negativen temperaturunabhängigen Temperaturkoeffizienten ausgegangen. In einem ersten Schritt wurde daher im Klimaschrank die Temperatur in 5 °C-Schritten von -30 °C bis +50 °C variiert.Die Ergebnisse dieser Messung waren allerdings erratisch und schwankten scheinbar willkürlich um bis zu drei Zyklen, wie in Abbildung 6.5 dargestellt ist. Es wurden daraufhin Messungen mit kontinuierlichem Temperaturverlauf, wie im Kapitel 5.2.1 beschrieben, durchgeführt.
Der, im vorhergehenden Kapitel beschriebene, nachteilige Einfluss eines externen synchronen Taktgenerators geht aus Abbildung 6.9 besonders deutlich hervor. In der oberen Grafik ist deutlich die wesentlich breitere Standardverteilung der Messung zu erken-

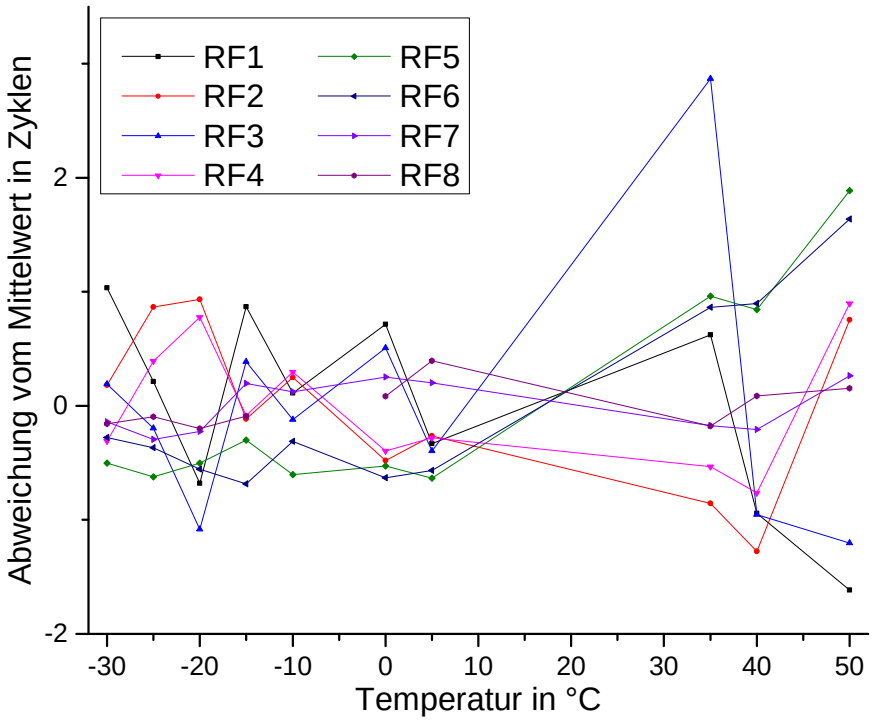


Abb. 6.5: Abhängigkeit der Entfernungsmessung von der Temperatur. Für die genauere Untersuchung der Temperaturabhängigkeit, wurde die Messung im Klimaschrank wiederholt. Aufgrund des erwarteten linearen oder exponentiellen Verhaltens wurde ein Messintervall von 5 °C gewählt.

nen. Dieses Ergebnis kann durch die starre Taktreferenz des Signalgenerators begründet werden. Es wird vermutet, dass die Regelschleife des Oszillators aufgrund des fehlendes Feedbacks instabil wird, und somit die interne Elektronik zum Schwingen anregt. In den Grafiken zum Temperaturverlauf ist jeweils die Richtung des positiven Temperaturgradienten (schwarz) und die des negativen Temperaturgradienten (weiß) dargestellt. Die sichtbaren Hysterese-Effekte beruhen auf der höheren thermischen Trägheit der Evaluation-Boards gegenüber den Temperatursensoren.

In Abbildung 6.6 ist das Ergebnis der Messung ohne externe Referenz nochmals mit einer feiner aufgelösten Farbkodierung versehen. Zudem ist der jeweilige Median (grün) und Modus (gelb) eingezeichnet. Beide Werte haben die meiste Zeit den selben diskreten Wert, jedoch ist zu erkennen, dass der Modus stets etwas früher abnimmt und etwas später zunimmt. Dies deutet darauf hin dass, wie auch schon in den Voruntersuchungen festgestellt wurde, die Messwerte nicht vollständig normalverteilt sind, sondern eine leicht rechts-schiefe Normalverteilung vorliegt.

Die Schwankungen des Offset mit der Temperatur betragen an der steilsten Stelle bis zu 1,07 Zyklen/°C. Dies entspricht einer Entfernungsabweichung von 12,4 m/°C. Ebenso

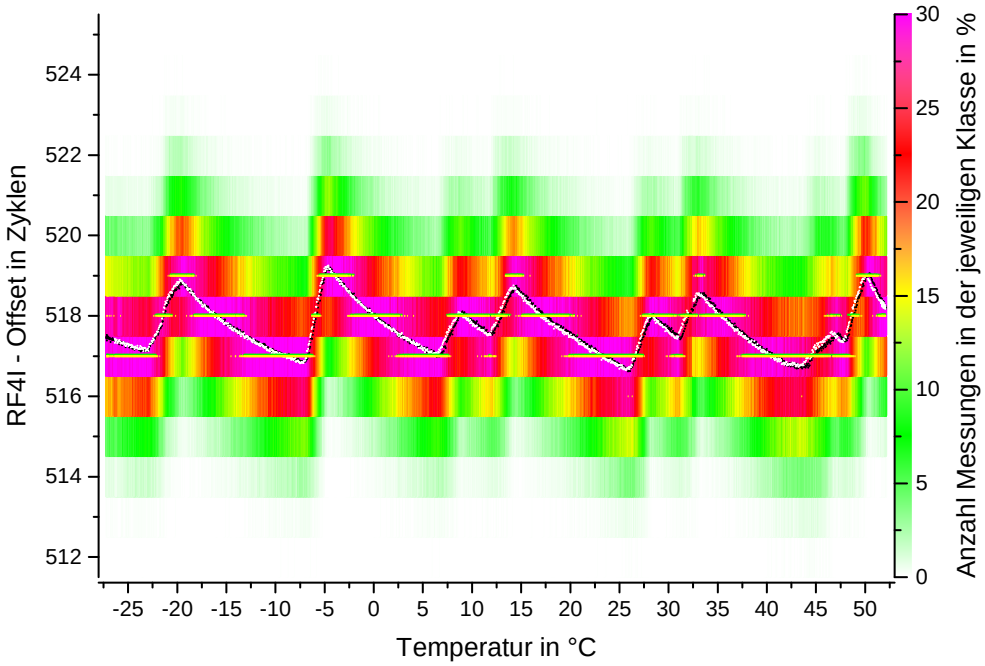


Abb. 6.6: Statistische Auswertung des Temperaturzyklus der Offsetmessung. Der Median ist grün dargestellt, der Modus gelb. Die Standardabweichung wird aus Gründen der Übersichtlichkeit in Abbildung 6.7 dargestellt.

sind die Wechsel zwischen positivem und negativem Temperaturkoeffizienten des Offsets augenfällig. Dieses Verhalten des Chips wird auf eine interne Temperaturkompensation zurückgeführt, da es reproduzierbar war und auf allen Platinen in der selben Form vorgekommen ist. Um diese These zu überprüfen, wurden verschiedene Einstellungen auf den Evaluation-Boards verändert. Abbildung 6.8 zeigt das Verhalten des Offsets bei verschiedenen Einstellungen für die Frequenzkompensation. Über den gesamten Temperaturbereich verhalten sich die Platinen mit verschiedenen Einstellungen gleich. Bei Temperaturen oberhalb von 45 °C wird allerdings der Fehler in der Frequenz bei der unkompensierten Messung zu groß und der Offset ändert sich stark. Die Messung deutet jedoch darauf hin, dass die Frequenzkompensation der Evaluation-Boards im normalen Temperaturbereich keinen Einfluss auf die Platinen hat.
Eine weitere aufschlußreiche Information ist in der Standardabweichung enthalten. Diese steigt während der Phase des positiven Temperaturkoeffizienten um 0,2-0,4 Zyklen an, wie in Abbildung 6.7 dargestellt ist. Dieses Verhalten ist ein Hinweis auf ein internes aktives Gegensteuern zur Temperaturkompensation, da ein Regelungsschaltkreis das System aus der Gleichgewichtsposition bringt und damit eine zusätzliche Störquelle darstellt.

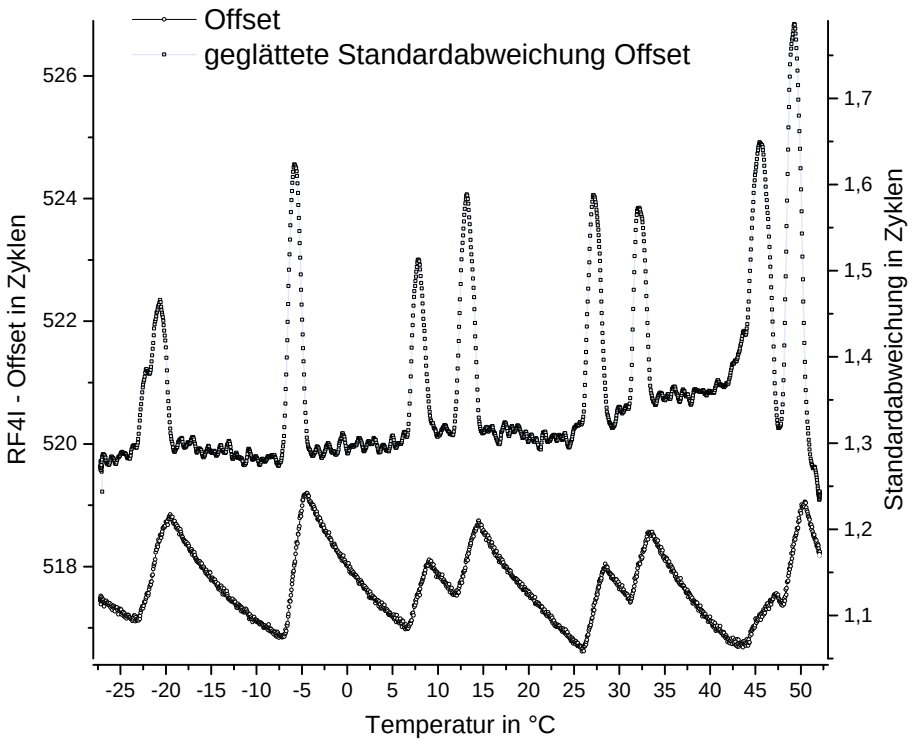


Abb. 6.7: Standardabweichung des Offset über der Zeit. Wenn der Temperaturkoeffizient der Evaluation Boards einen Vorzeichenwechsel absolviert, steigt für die Zeit der positiven Steigung die Standardabweichung um 0,2-0,4 Zyklen.

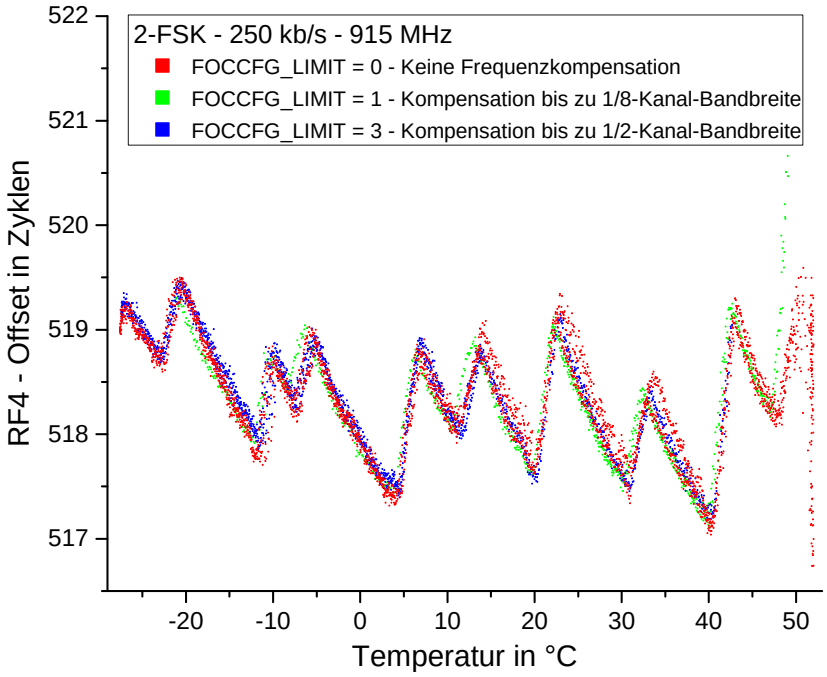
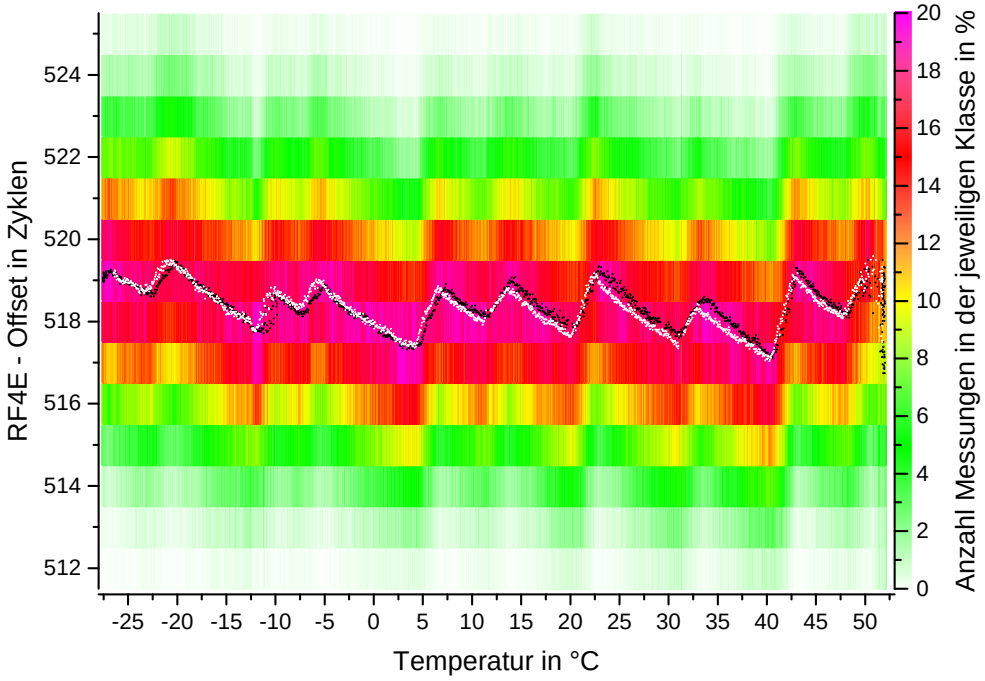
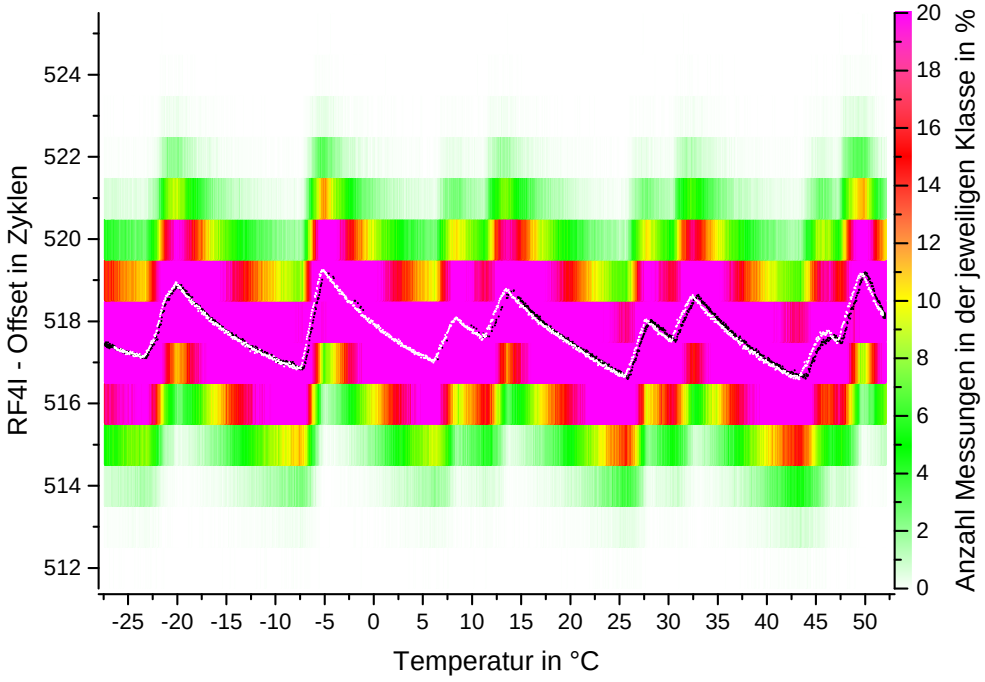


Abb. 6.8: Einfluss der automatischen Frequenzkompensation auf den Temperaturzyklus. Die Kompensation bis zu einem Achtel der Kanal-Bandbreite war der vom *SmartRF-Studio* vorgegebene Standardwert für das FOCCFG-Register. Die anderen beiden Werte stellen jeweils die maximale und minimale Einstellung des Registers dar [36].



(a)



(b)

Abb. 6.9: Vergleich zwischen a) externem synchronem und b) den asynchronen internen Taktgeneratoren. Der externe Taktgenerator verursacht eine Verbreiterung der Standardabweichung. Das thermische Verhalten hat zudem ein anderes, schlecht reproduzierbares Aussehen. Die schwarzen Punkte auf der Kurve stellen den Weg positiver Temperaturänderung dar, wohingegen die weißen Punkte den Weg negativer Temperaturänderung darstellen.

6.2 Messungen im Entfernungsmessstand

Im Entfernungsmessstand lassen sich nur die Entfernung, und einige Rahmenparameter, wie die Versorgungsspannung und die Beschaffenheit der Umgebung, kontrollieren. Bei den hier vorgestellten Messungen wurde daher die Entfernung zwischen 2 m und 30 m variiert.

6.3 Innenraum

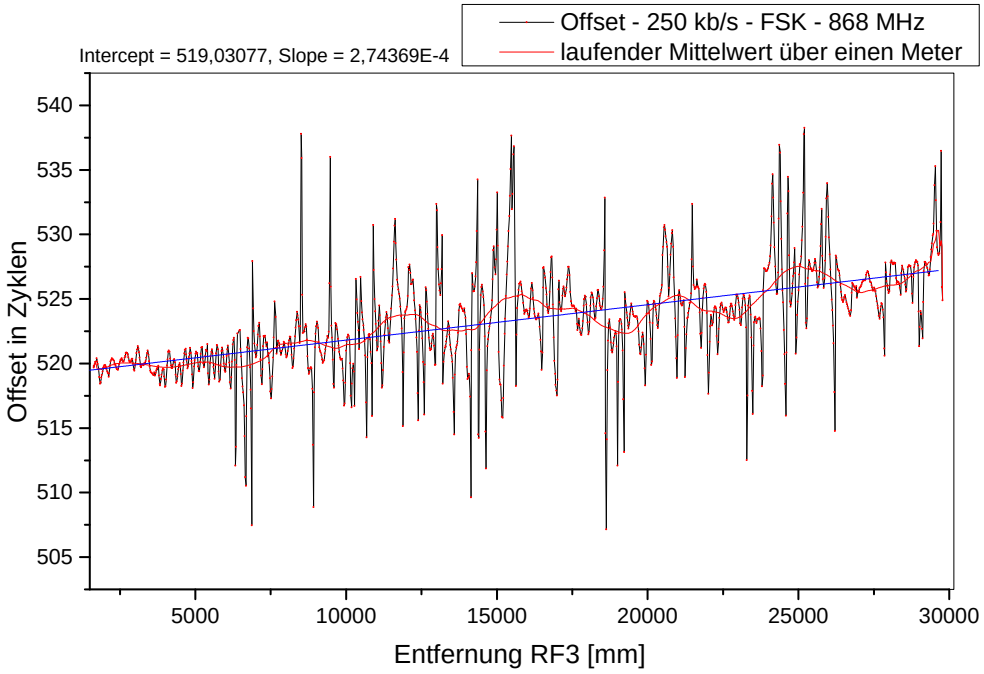
Die Ergebnisse der Entfernungsmessung im Innenraum sind in Abbildung 6.10 dargestellt. Das Signal sieht bei kleiner Vergrößerung stark verrauscht und gestört aus. Bei vergrößertem Bildauschnitt sind allerdings deutlich die einzelnen Messpunkte zu sehen. Die Offset-Messung zeichnet ein Bild des Interferenzmusters der Halle. Die Interferenzmuster sahen sich unter ähnlichen Bedingungen entsprechend ähnlich.
Die Temperatur während der Messung wurde ebenfalls erfasst. Allerdings ist die Änderung der Temperatur mit einem maximalen Hub von 9,6 °C während einer Messung von 12 h Dauer mit 0,013 °C/min deutlich geringer, als in der Temperaturzyklierung. Durch die langsame Änderung verteilt sich die Offset-Änderung sukzessive über eine große Anzahl an Messpunkten. Dieser Einfluss ist daher visuell in den Daten nicht auszumachen, kann und muss aber zur Offset-Korrektur herangezogen werden.

6.4 Freifeld

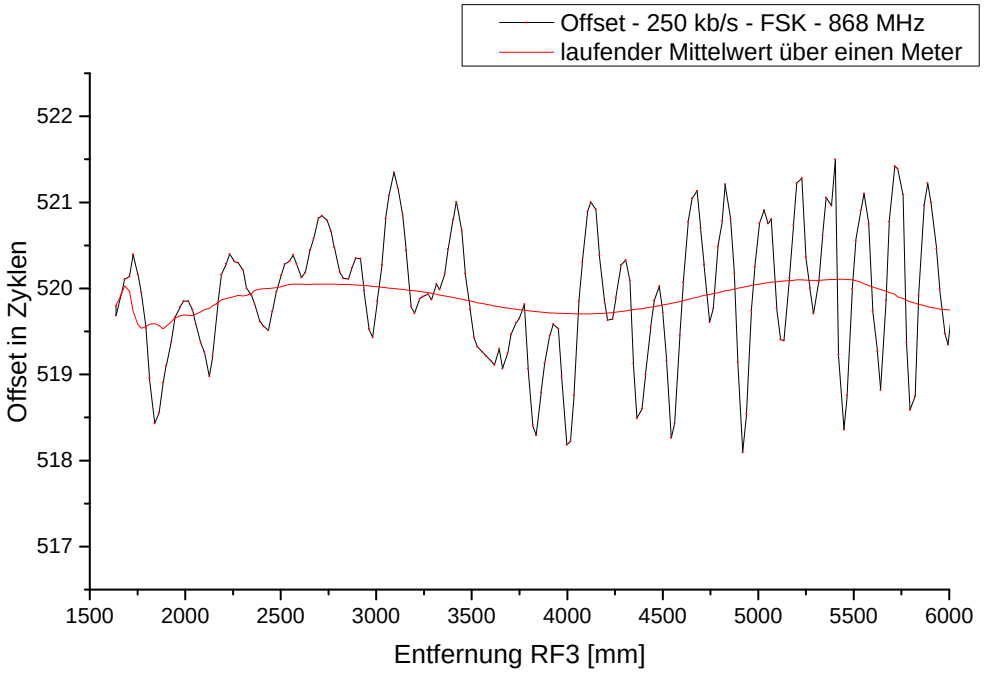
Die Messungen im Freifeld sollten deutlich störungsärmere Ergebnisse liefern. Jedoch treten auch bei den Outdoor-Messungen Interferenz-Effekte durch Bodenreflexion auf, die sich genau wie in der Indoor-Messung in den Messergebnissen wiederfinden. In Abbildung 6.11 sind die Ergebnisse der Freifeldmessungen dargestellt. Die Ergebnisse weisen, verglichen mit den Indoor-Messungen, deutlich geringere Spitzen in der empfangenen Signalstärke auf. Dennoch beträgt die Standardabweichung der Entfernungsmessung 14,30 m. Durch Glättung über einen laufenden Mittelwert, kann ein annäherungsweise lineares Verhalten der Distanzmessung erreicht werden. Die geglättete Kurve hat eine Standardabweichung vom idealen Wert von 3,11 m.

6.5 Schlußfolgerungen

Alle Messungen mit dem Entfernungsmessstand haben gemeinsam, dass die Messergebnisse für den Offset ein starkes Auftreten von Interferenzmustern zeigen. Dies scheint durch das Verhalten des Transceivers bei Empfang über die Luft und eine Antenne ein mindestens teilweise gewolltes Verhalten zu sein. Die starken Schwankungen der Offset-Werte machen eine Entfernungsbestimmung außerordentlich schwierig und schwer zu kalibrieren. Trotzdem bleibt anzumerken, dass die Datenübertragung, die eigentliche Anwendung des CC430-Sensorknotens, selbst unter widrigen Umständen mit Mehrwegausbreitungen und Bodenreflexionen tadellos funktioniert. In den zugrunde liegenden Design-Prinzipien dieses Transceivers scheint der Zuverlässigkeit der Datenübertragung eine wesentlich höhere Priorität zugeordnet zu sein, als der Determiniertheit der Sendeabläufe. Zudem kann sich in der Umgebung des Messtandes durch die statische Messung ein gleichbleibendes Feld ausbilden. Bei den Wiederholungsmessungen traten jeweils ähnliche bis identische Interferenzmuster auf, die sich durch Umstellen von Gerätschaften verändern ließen.
In der Theorie ist die Laufzeitmessung sehr robust gegen klassische Störeffekte, wie Mehrwegausbreitungen und Signalauslöschungen, sowie unabhängig von unpräzisen Parametern, wie der Eingangssignalstärke. In der Praxis ist das Gegenteil der Fall. Die Empfangsfeldstärke beeinflusst über den zeitlichen Offset des Analogteils direkt die Entfernungsmessung. Umwelteinflüsse, wie Temperatur, Versorgungsspannung und Luftfeuchte wirken direkt auf die Ergebnisse der Laufzeitmessung ein. Zudem übersteigen diese Umwelteinflüsse betragsmäßig den Einfluss der Laufzeit auf die Entfernungsmessung deutlich.

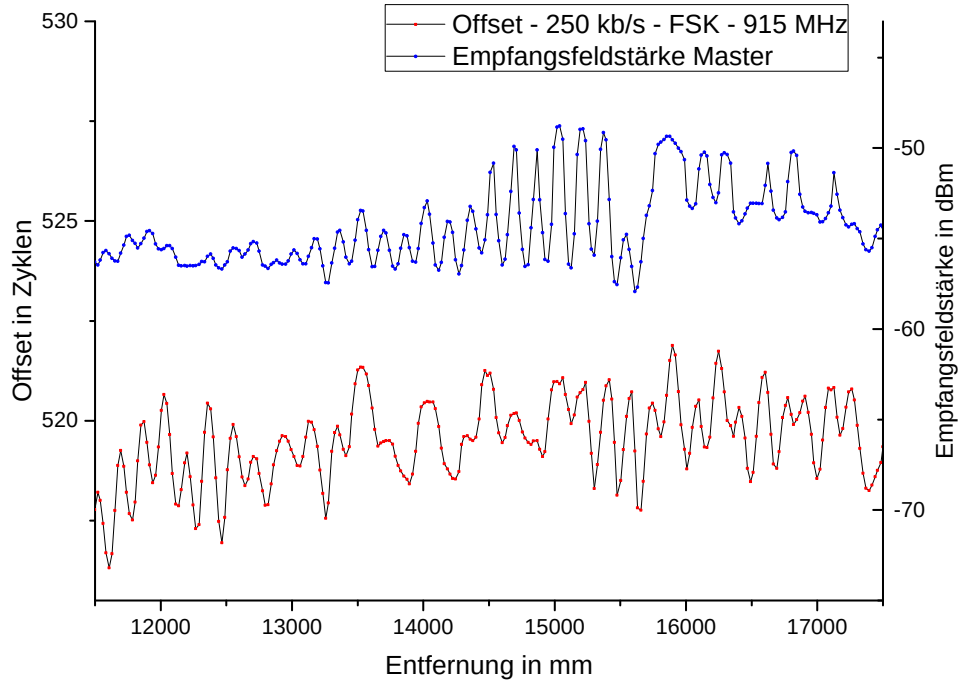


(a)

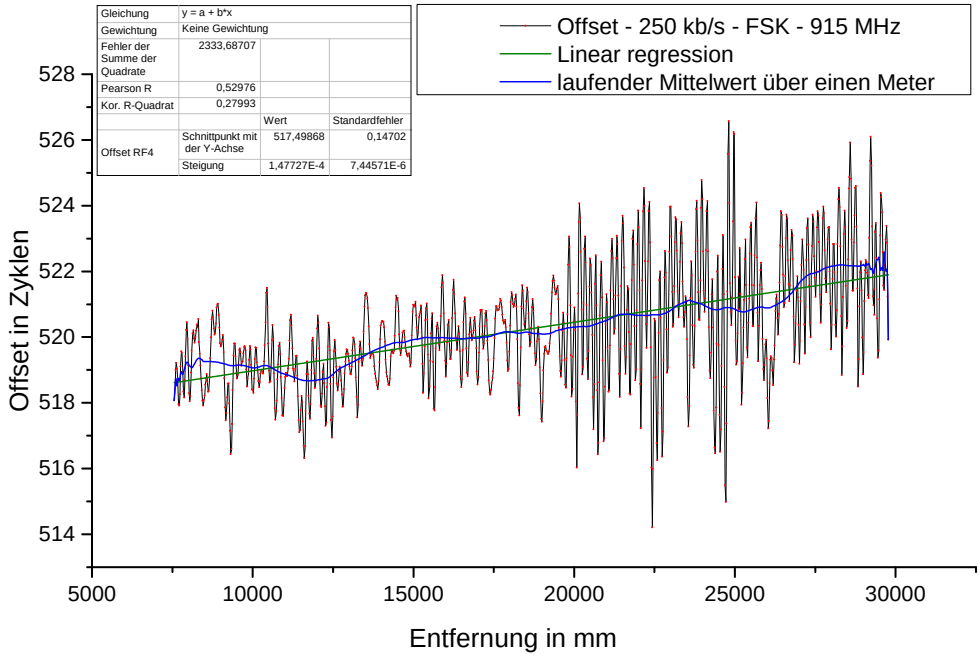


(b)

Abb. 6.10: Ergebnisse der Entfernungsmessung im Innenraum mit einer Datenrate von 250 kb/s bei 868 MHz und FSK-Modulation. a) Messwerte über die gesamte Länge des Messstands. Der gleitende Mittelwert soll eine Bewegung des Messobjektes durch das Intereferenzmuster imitieren, da diese Bewegung auch die Messwerte glättet. b) Vergrößerung der ersten 6 m. Die Verläufe des Offsets sind kontinuierlich, es ist das Interferenzmuster der Halle abgebildet.



(a)



(b)

Abb. 6.11: Ergebnisse der Entfernungsmessung im Freifeld mit einer Datenrate von 250 kb/s bei 915 MHz und FSK-Modulation. a) Stark vergrößerter Bereich und Vergleich mit Empfangsfeldstärkemessung. Die Werte sind kontinuierlich. b) Ergebnis der gesamten Messung. Es treten insbesondere weniger Störeffekte durch konstruktive und destruktive Interferenz auf.

7 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde ein Messsystem zur rundlaufzeitbasierten Entfernungsmessung (RTT) mit *low-cost*-Hardware implementiert, untersucht und das Konzept experimentell bestätigt. Es wurde ein kurzer Überblick über die theoretischen Grundlagen gegeben und der aktuelle Stand der Technik erörtert. Für die Charakterisierung des Systems wurde zudem ein Entfernungsmesstand konzipiert, entworfen und aufgebaut. Mit diesem ist es möglich, einen mobilen Knoten auf einer Entfernung von 30 m mit 3 mm Genauigkeit zu positionieren und mit 1 mm Genauigkeit zu lokalisieren. Des Weiteren ist es mit dem Messstand möglich, zeitaufwändige Präzisionsmessungen, wie sie für die Charakterisierung des in dieser Arbeit untersuchten Messsystems notwendig waren, vollständig automatisiert durchzuführen.
Das für diese Arbeit entworfene Messsystem wurde insbesondere auf sein Verhalten gegenüber diversen Umwelteinflüssen, wie der Temperatur, der Versorgungsspannung, Mehrwegausbreitungs- und Interferenzeffekten untersucht. Dabei ergab sich ein sehr starker Einfluss der Umweltparameter auf die RTT-Messung, die vom Standpunkt der Literatur betrachtet, als sehr robust gegenüber solchen Störeffekten gilt. Im Einzelnen wurden die Abweichungen in der Entfernungsmessung mit dem CC430-Transceiver bestimmt. Die Temperatur hat mit 12,4 m/°C bei weitem den größten Einfluss. Daran schließt sich die Signaldämpfung an, mit circa 1,38 m/dB ab einer Eingangssignalstärke von -60 dBm. Eine Schwankung der Versorgungsspannung verursacht eine Abweichung von 4,9 cm/V. Der Einfluss der Entfernung auf die Laufzeitmessung liegt, je nach angesetzter Ausbreitungsgeschwindigkeit, bei circa 0,2 Zyklen/m und damit bei 0,39 ‰/m des Messsignals, weit unterhalb des Einflusses der anderen Parameter. Diese Ergebnisse wurden auf der Mechatronics Conference 2014 in Karlstad, Schweden, veröffentlicht [37].
Für eine kommerzielle und funktionale Anwendung der Rundlaufzeitmessung zur Entfernungsbestimmung müssen die in dieser Arbeit vorgestellten Einflussgrößen zwingend berücksichtigt werden. Weiterhin sollte in den wenig kostensensitiven Ankerknoten eine hoher Aufwand betrieben werden, den Transceiver von Umwelteinflüssen

abzuschotten. Insbesondere erscheint es, in Anbetracht der Ergebnisse dieser Arbeit, geboten und notwendig den Einfluss der Temperatur auf den Sensorknoten durch Temperierung zu eliminieren. Dazu sollte eine Temperatur auf der Temperaturkurve des Transceivers gewählt werden, die eine möglichst geringe Steigung zu beiden Seiten aufweist. Zudem ist ein, nur in eine Richtung regelndes, Kühlsystem stabiler und einfacher umzusetzen.

Neben der Temperierung ist noch eine weitere Methode nach vorläufigen Messergebnissen sehr viel versprechend. Die Ankerknoten werden mit einer festgelegter Winkelgeschwindigkeit in einem Kreis von circa 1 m bewegt. Während eines Umlaufes werden die Einzelmessung für eine einzelne Positionsbestimmung durchgeführt. Durch dieses Vorgehen erfolgt eine räumliche Mittelung über das Interferenzmuster, welche die Genauigkeit der Entfernungsmessung deutlich erhöht. Durch Anwendung dieses Verfahrens ließ sich die Standardabweichung der Entfernungsmessung in der vorliegenden Arbeit von 14,30 m auf 3,11 m senken.

Literaturverzeichnis

[1] F. Höflinger, "Lokalisierungssysteme für die Positionsbestimmung von Personen und Objekten im Innenraum," Dissertation, Albert-Ludwigs-Universität Freiburg, 2014.

[2] M. Li, G. Calinescu, P.-J. Wan, and Y. Wang, "Localized Delaunay triangulation with application in ad hoc wireless networks," *Parallel and Distributed Systems, IEEE Transactions on*, vol. 14, no. 10, pp. 1035–1047, Oct 2003.

[3] U. Gamm, "Sensorknoten mit Ultra Low Power Aufweckempfängern," Dissertation, Albert-Ludwigs-Universität Freiburg, 2013.

[4] Johnsrund and Aaberge, *Design Note DN505 - RSSI Interpretation and Timing*, Texas Instruments. [Online]. Available: www.ti.com

[5] H. Hashemi, "The indoor radio propagation channel," *Proceedings of the IEEE*, vol. 81, no. 7, pp. 943–968, 1993.

[6] A. Bildea, O. Alphand, and A. Duda, "Link Quality Metrics in Large Scale Indoor Wireless Sensor Networks," in *IEEE 24th International Symposium on Personal Indoor and Mobile Radio Communications*, Pornic, France, 8 - 11 September 2013, pp. 1888–1892.

[7] J. Xu, W. Liu, F. Lang, Y. Zhang, and C. Wang, "Distance Measurement Model Based on RSSI in WSN," *Wireless Sensor Network*, vol. 2, no. 8, pp. 606–611, 2010.

[8] A. Awad, T. Frunzke, and F. Dressler, "Adaptive Distance Estimation and Localization in WSN using RSSI Measures," in *Digital System Design Architectures, Methods and Tools, 2007. DSD 2007. 10th Euromicro Conference on*, August 2007, pp. 471–478.

[9] M. Sugano, T. Kawazoe, Y. Ohta, and M. Murata, "Indoor localization system using RSSI measurement of wireless sensor network based on ZigBee standard," in *Wireless and Optical Communication MultiConference 2006*, 3 - 5 July 2006.

[10] D. M. Amzucu, H. Li, and E. Fledderus, "Indoor Radio Propagation and Interference in 2.4 GHz Wireless Sensor Networks: Measurements and Analysis," *Wireless Personal Communications*, vol. 76, no. 2, pp. 245–269, May 2014. [Online]. Available: http://dx.doi.org/10.1007/s11277-014-1694-2

[11] E. Vinogradov, W. Joseph, C. Oestges *et al.*, "Modeling and simulation of fast fading channels in indoor peer-to-peer scenarios," in *8th European Conference on Antennas and Propagation*, 2014.

[12] W. Murphy and W. Hereman, "Determination of a position in three dimensions using trilateration and approximate distances," *Department of Mathematical and Computer Sciences, Colorado Sch. of Mines, MCS-95-07*, vol. 19, 1999.

[13] A. Bahillo, S. Mazuelas, J. Prieto, P. Fernandez, R. Lorenzo, and E. Abril, "Hybrid RSS-RTT localization scheme for wireless networks," in *International Conference on Indoor Positioning and Indoor Navigation*. Zürich, Switzerland, 15 - 17 September 2010.

[14] A. Bahillo, J. Prieto, S. Mazuelas, R. Lorenzo, J. Blas, and P. Fernandez, "IEEE 802.11 Distance Estimation Based on RTS/CTS Two-Frame Exchange Mechanism," in *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, 2009, pp. 1–5.

[15] S. A. Golden and S. S. Bateman, "Sensor Measurements for Wi-Fi Location with Emphasis on Time-of-Arrival Ranging," *Mobile Computing, IEEE Transactions on*, vol. 6, no. 10, pp. 1185–1198, 2007.

[16] N. Patwari, J. Ash, S. Kyperountas, A. Hero, R. Moses, and N. Correal, "Locating the nodes: cooperative localization in wireless sensor networks," *Signal Processing Magazine, IEEE*, vol. 22, no. 4, pp. 54–69, 2005.

[17] A. Günther and C. Hoene, "Measuring round trip times to determine the distance between WLAN nodes," in *NETWORKING 2005. Networking Technologies, Services, and Protocols; Performance of Computer and Communication Networks; Mobile and Wireless Communications Systems*. Springer, 2005, pp. 768–779.

[18] M. Ciurana, D. López, and F. Barceló-Arroyo, "SofTOA: Software ranging for TOA-based positioning of WLAN terminals," in *Location and Context Awareness*. Springer, 2009, pp. 207–221.

[19] M. Ciurana, F. Barcelo-Arroyo, and F. Izquierdo, "A ranging system with IEEE 802.11 data frames," in *Radio and Wireless Symp., 2007 IEEE*, 2007, pp. 133–136.

[20] A. Catovic and Z. Sahinoglu, "The Cramér-Rao bounds of hybrid TOA/RSS and TDOA/RSS location estimation schemes," *Communications Letters, IEEE*, vol. 8, no. 10, pp. 626–628, 2004.

[21] C. Fritsche and A. Klein, "Cramér-Rao Lower Bounds for hybrid localization of mobile terminals," in *Positioning, Navigation and Communication, 2008. WPNC 2008. 5th Workshop on*, 2008, pp. 157–164.

[22] H. Will, S. Pfeiffer, S. Adler, T. Hillebrandt, and J. Schiller, "Distance measurement in wireless sensor networks with low cost components," in *International Conference on Indoor Positioning and Indoor Navigation*. Guimarães, Portugal, 21 - 23 September 2011.

[23] S. Pfeiffer, "Abstandsmessung durch Laufzeitmessung in drahtlosen Sensornetzwerken," Diplomarbeit, Freie Universität Berlin, Dezember 2010.

[24] J. Wendeberg, F. Höflinger, C. Schindelhauer, and L. Reindl, "Calibration-free TDOA self-localisation," *J. of Loc. Based Services*, vol. 7, no. 2, pp. 121–144, 2013.

[25] T. Wigren and J. Wennervirta, "RTT Positioning in WCDMA," in *Wireless and Mobile Comm., 2009. ICWMC '09. 5th Int. Conf. on*, Aug 2009, pp. 303–308.

[26] J. Wennervirta and T. Wigren, "RTT Positioning Field Performance," *Vehicular Technology, IEEE Transactions on*, vol. 59, no. 7, pp. 3656–3661, Sept 2010.

[27] N. Salman, M. Ghogho, and A. Kemp, "Optimized low complexity sensor node positioning in wireless sensor networks," *Sensors Journal, IEEE*, vol. 14, no. 1, pp. 39–46, Jan 2014.

[28] G. Sun, J. Chen, W. Guo, and K. Liu, "Signal processing techniques in network-aided positioning: a survey of state-of-the-art positioning designs," *Signal Processing Magazine, IEEE*, vol. 22, no. 4, pp. 12–23, July 2005.

[29] C. Röhrig, "Lokalisierungsverfahren für drahtlose Sensornetzwerke," *Herzog MH (Hrsg.): Wireless Communication and Information–Radio Engineering and Multimedia Applications. Boizenburg, Deutschland: Verlag Werner Hülsbusch*, pp. 81–97, 2009.

[30] C. Röhrig and M. Müller, "Localization of Sensor Nodes in a Wireless Sensor Network Using the nanoLOC TRX Transceiver," in *Vehicular Technology Conference, 2009. VTC Spring 2009. IEEE 69th*, April 2009, pp. 1–5.

[31] T. Fellner, J. Wilde, R. Zeiser, T. Jörger, M. Törk, and L. Reindl, "Neuartige hochohmige Dünnschicht-Folien-DMS für Wireless Condition Monitoring," *MikroSystemTechnik*, 2011.

[32] R. Zeiser, T. Jörger, T. Fellner, and J. Wilde, "Entwicklung von kapazitiven Dehnungsmessstreifen in Dünnschichttechnologie für drahtlose, intelligente Systeme," *MikroSystemTechnik*, 2011.

[33] T. Jörger, "Drahtlose Sensorsysteme mit energieoptimierter Dehnungsmessung," Bachelorarbeit, Albert-Ludwigs-Universität Freiburg, Oktober 2010.

[34] *Produktübersicht*, SmartExergy WMS GmbH.

[35] T. Ungan, G. Gamm, L. Reindl, T. Ostertag, M. Sippel, and T. Wendt, "Electronic device comprising an operating mode switching unit," US Patent 20 130 130 636 A1, May 23, 2013.

[36] *Datasheet CC430F613x*, Texas Instruments. [Online]. Available: www.ti.com

[37] T. Jörger, U. Gamm, F. Höflinger, and L. M. Reindl, "Wireless distance estimation with low-power standard components in wireless sensor nodes," in *14th Mechatronics Forum International Conference, Karlstad, Schweden*, 16-18 Juni 2014.

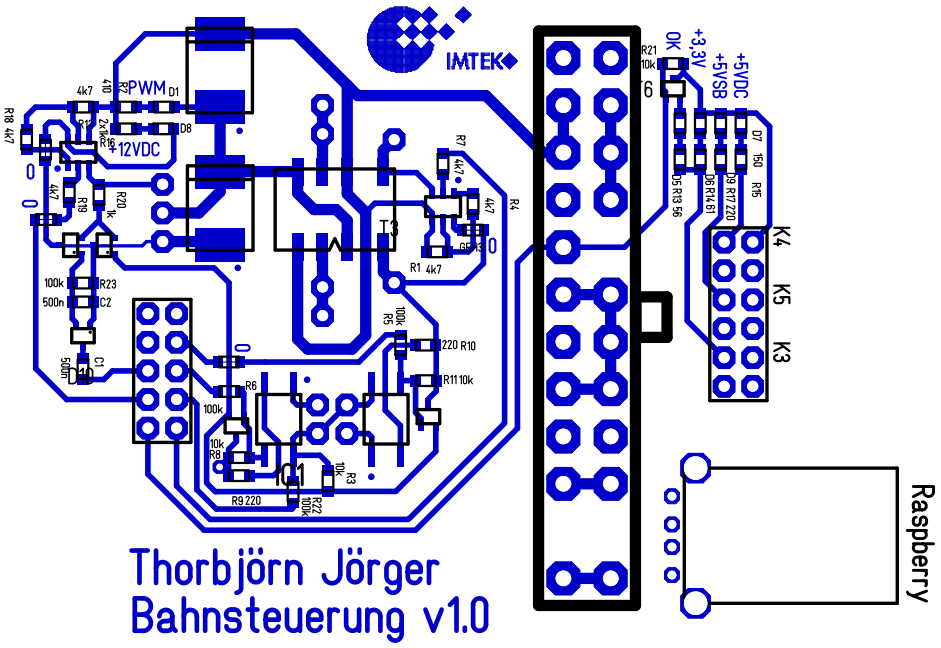
A Hardware

Messstand

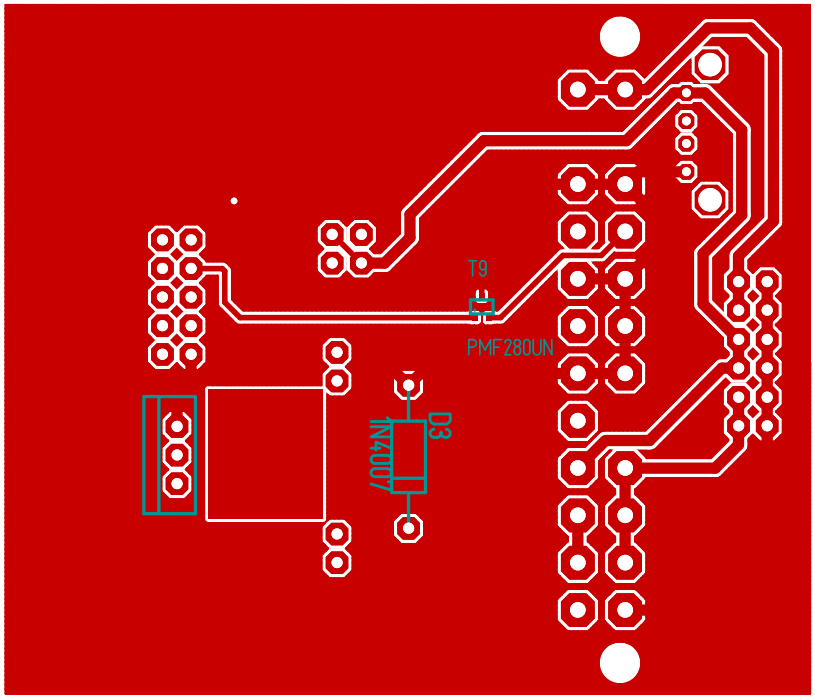
Das Platinenlayout für die Leistungelektronik des Platinenlayouts findet sich auf Seite 66, der dazugehörige Schaltplan auf Seite 69.

Erweiterungen Evaluation-Board

Das Layout der für die Evaluation-Boards entwickelten Hardwareerweiterungen findet sich auf Seite 67.

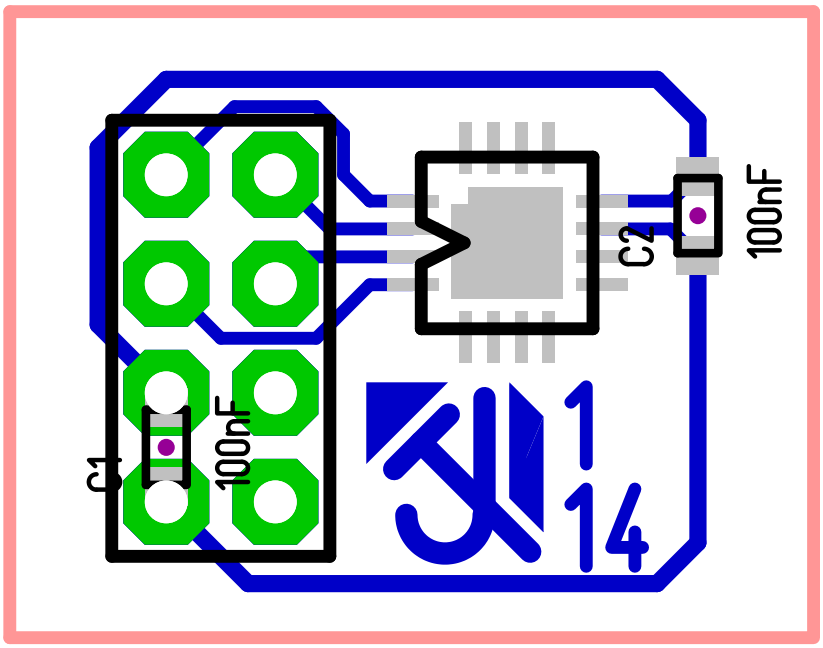


(a) Oberseite mit Bestückung

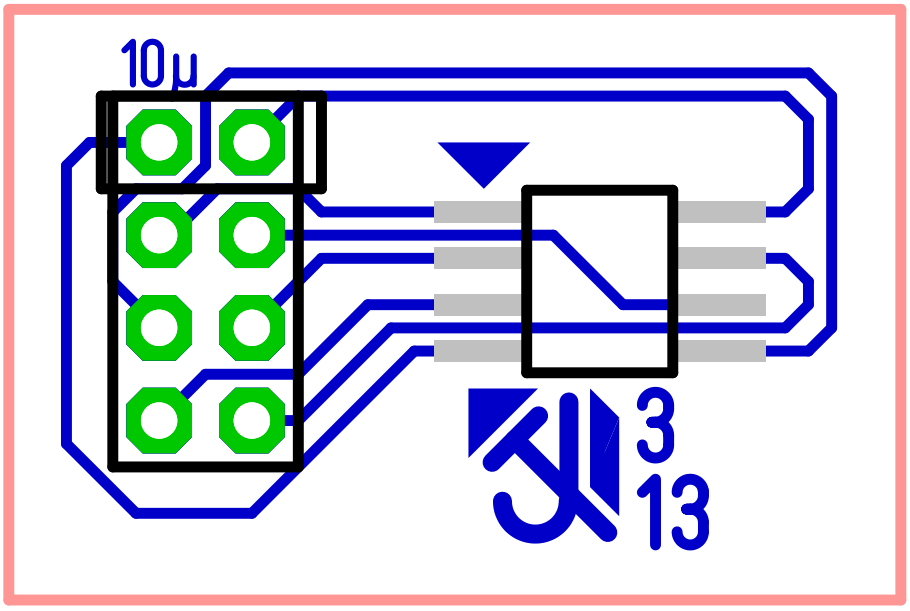


(b) Unterseite mit Bestückung

Abb. A.1: Platinenlayout der Leistungselektronik

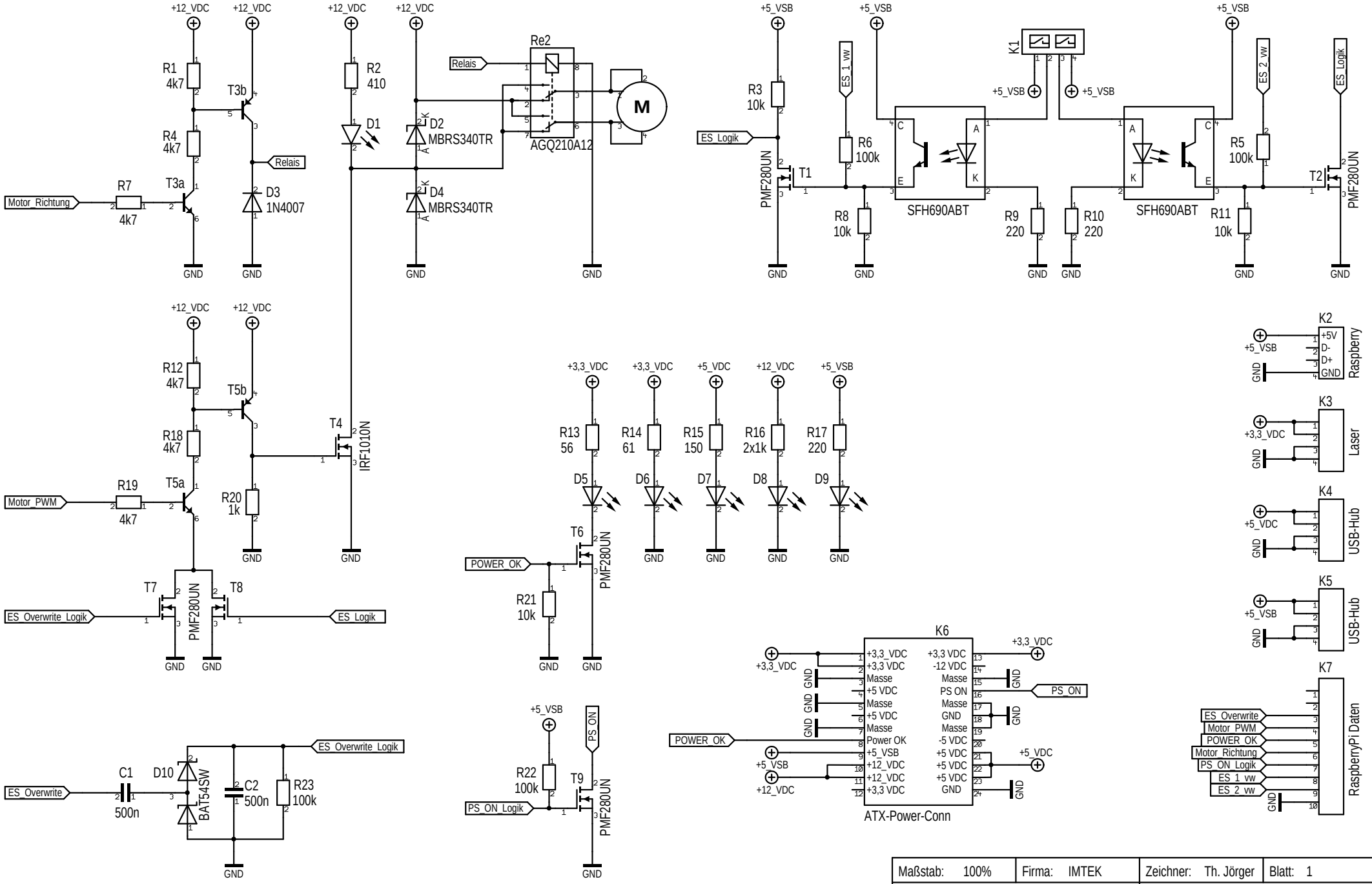


(a) Erweiterungsplatine für Temperatursensor ADT7320



(b) Erweiterungsplatine für FRAM FM25H20

Abb. A.2: Platinenlayout der Erweiterungsplatinen



X = A oder NICHT (B oder C)

Maßstab: 100%	Firma: IMTEK	Zeichner: Th. Jörger	Blatt: 1
Änderung: 08.06.14 15:28		Titel: Bahnsteuerung	
Ausgabe: 08.06.14 15:29			
Datei: Bahnsteuerungs.T3001		Projekt: Masterarbeit	

B Programmierung

Messtand

Die Programme für die Steuerung des Messtandes finden sich ab Seite 70. Das Programm des Laserentfernungsmessers ist auf Seite 76

Evaluation-Boards

Die Programmierung der Evaluation-Boards findet sich am Seite 84. Direkt zu Beginn wird das Hauptprogramm für die Entfernungsmessung über die Funkkommunikation präsentiert. Auf Seite 100 findet sich die Bibliothek zur Ansteuerung des FRAM-Speichers. Ab Seite 103 findet sich die Bibliothek für die Ansteuerung des Temperatursensors.

Auswertung

Im folgenden ist eines der Programme zur statistischen Auswertung aufgeführt. Tatsächlich war für jede der Durchgeführten Messungen eine geringfügig andere Auswertung, insbesondere hinsichtlich der gewünschten Ausgangsanordnung zur Darstellung mit Origin, erforderlich. Sämtliche Auswerteprogramme sind im Anhang C zu finden.

Hauptprogramm Entfernungsmessstand

```
#!/usr/bin/python3
#------------------------------------------------------------------------#
#                                                                        #
# Entfernungsmessstand-Steuerungsprogramm v0.4 (c) Th. JÃ¶rger 2012-2014 #
#                                                                        #
# Albert-Ludwigs-UniversitÃ¤t Freiburg - Lehrstuhl EMP                    #
#                                                                        #
#------------------------------------------------------------------------#

#---------- Imports ---------------------------------------------------#

import argparse
import globals
import sys
import signal
import time
import os
from globals import verbosity_output
from BTDistThread import BTDistThread
from DistPIDThread import DistPIDThread
from subprocess import call

#---------- Argument Parsing ------------------------------------------#
#                                                                        #
# Global variables created:                                              #
#        stepsize - Size of each measurement step in cm                  #
#        steps - Number of steps of the whole measurement                #
#        measurements - Number of measurements, -1 is infinite          #
#                                                                        #
#------------------------------------------------------------------------#

def signal_handler(signal, frame):
   print("\r\n")
   verbosity_output(0, "warning", origin, "Close program\r\n")
   PID.stop()
   BT.stop()
   sys.exit()

mainpid = os.getpid()

parser = argparse.ArgumentParser(description="Entfernungsmesstand-Steuerungsprogramm v0.3 (c) Th.
JÃ¶rger 2012-2014", \
epilog="\nYou can do this and that, I don't care", formatter_class=argparse.RawTextHelpFormatter)

parser.add_argument("-n", "--number", type=int, dest="N",       help="Number of measurements, -1 is
\u221E\t   (default: 1)", default = 1)
parser.add_argument("-b", "--both",                             help="Measure in both directions\t
  (default: no)", action="store_true")
parser.add_argument("-l", "--length", type=int, dest="L",       help="Length of track\t\t\t   (
default: 1000 mm)", default = 1000)
parser.add_argument("-e", "--error", type=int, dest="E",        help="Allowed error for PID
controller   (default: 5 mm)", default = 5)
```



```
parser.add_argument("-o", "--offset", type=int, dest="O",       help="Distance offset for zero point
\t   (default: 600 mm)", default = 600)
parser.add_argument("-P", "--ppart", type=float, dest="P",      help="P value for PID controller\t
  (default: 0.8)", default = 0.8)
parser.add_argument("-I", "--ipart", type=float, dest="I",      help="I value for PID controller\t
  (default: 0.1)", default = 0.1)
parser.add_argument("-D", "--dpart", type=float, dest="D",      help="D value for PID controller\t
  (default: 0.04)", default = 0.04)
parser.add_argument("-d", "--derivator", type=float, dest="d",  help="Derivator for PID controller\t
   (default: 0.0)", default = 0.0)
parser.add_argument("-i", "--integrator", type=float, dest="i", help="Integrator for PID controller\
t   (default: 0.0)", default = 0.0)
parser.add_argument("-m", "--imax", type=int, dest="imax",      help="Maximum Integrator value\t   (
default: 250)", default = 250)
parser.add_argument("-r", "--output",  dest="r", type=int,      help="Output range of PID controller
\t   (default: 1023)", default = 1023)
parser.add_argument("-f", "--file", dest="f", type=str,         help="File name for measurement\t",
default = '')
parser.add_argument("-z", "--z-axis", dest="z", type=int,       help="Set z-Position without ANY
checks\t", default = 0)
parser.add_argument("-q", "--continue", dest="q", type=int,     help="Continue from measurement step
\t", default = 0)
stepping = parser.add_mutually_exclusive_group(required=True)
stepping.add_argument("-s", "--stepsize", type=int, dest="S",   help="Size of single measurement
step")
stepping.add_argument("-p", "--steps", type=int, dest="p",      help="Parts track is divided to")
parser.add_argument("-v", "--verbose", help="Increase output verbosity", action="count", default =
0)
parser.add_argument("-x", "--logging", help="Increase logging verbosity", action="count", default =
1)
parser.add_argument("-c", "--color", help="Switch colored output off", action="store_false")
parser.add_argument("-lc", "--logcolor", help="Switch colored log on", action="store_true")
parser.add_argument("-fn", "--filename", help="Filename for data", default='default')
#parser.add_argument("--version", action="version", version="%(prog)s v0.3")
parser.add_argument("--version", action="version", version="Messstand v0.3")

args = parser.parse_args()

if args.O > args.L:
   verbosity_output(0, "error", origin, "Start offset is bigger than track length. Terminate program
   \r\n")
   sys.exit()

if args.S:
   steps = int((args.L - args.O)/args.S) + 1
   stepsize = args.S
else:
   stepsize = int((args.L - args.O)/args.p)
   steps = args.p

if args.both:
   prsteps = 2 * steps
else:
   prsteps = steps

measurements = args.N

filepath = "./data/" + time.strftime('%Y%m%d_%H%M%S_') + args.filename + "/"
os.mkdir(filepath)

globals.init()
globals.verbosity = args.verbose
```

```
globals.logging = args.logging
globals.color = args.color
globals.logfile = open(filepath + "messages.log", 'w+')
globals.isPrinting = False
globals.logcolor = args.logcolor

signal.signal(signal.SIGINT, signal_handler)

origin = "MainThread"

BT = BTDistThread(1, "BT-Thread")
PID = DistPIDThread(1, "PID-Thread", BT, args.P, args.I, args.D, args.d, args.i, args.imax, args.r,
args.E)
BT.start()
PID.setPoint(args.0)
PID.start()

if not(args.z == 0): #Goto specific position and end program
   verbosity_output(0, "boring", origin, "Going to position " + str(abs(args.z)) + " without ANY
   checks. Be careful!\r\n")
   PID.setPoint(abs(args.z))

   while not PID.setPointstable == True:
      time.sleep(5)

   verbosity_output(1, "success", origin, "Position " + str(abs(args.z)) + " stable\r\n")
   verbosity_output(0, "success", origin, "Set position finished. Exit program.")
   os.kill(mainpid, signal.SIGINT)
   time.sleep(1)

setPoint = args.0

verbosity_output(0, "boring", origin, "Number of measurements: " + str(measurements) + "\tStepsize:
" + str(stepsize) + "\tNumber of Steps: " + str(steps) + "\tBoth directions: " + str(args.both) + "\
r\n")

m = 0
s = 0
prs = 0
if not (args.q == 0):
   if args.q > steps:
      s = (2* steps) - args.q
      strDirection = "_Back"
      countingBackward = True
      prs = args.q
   else:
      s = args.q
      prs = args.q
      countingBackward = False
      strDirection = "_Forth"


while m != measurements:

   verbosity_output(1, "data", origin, "Measurement: " + str(m + 1) + " in progress\r\n")
   verbosity_output(3, "data", origin, "\tm = " + str(m) + "\ts = " + str(s) + "\r\n")

   while s < steps and s >= 0:
      lasttime = time.time()
      acttimestr = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time()))
      verbosity_output(1, "boring", origin, "Step " + str(prs + 1) + "/" + str(prsteps) + "\t" +
      acttimestr + "\r\n")
```



```
   verbosity_output(1, "boring", origin, "Going to position " + str(args.0 + (s * stepsize)) + "\
   r\n")
   verbosity_output(3, "data", origin, "\tm = " + str(m) + "\ts = " + str(s) + "\r\n")

   PID.setPoint(args.0 + (s * stepsize))
   temptime = time.time()
   while not PID.setPointstable == True:
      if time.time() - temptime > 90:
         verbosity_output(0, "error", origin, "No stable position achieved. Safety exit\r\n")
         os.kill(mainpid, signal.SIGINT)
      time.sleep(0.1)

   verbosity_output(1, "success", origin, "Position " + str(args.0 + (s * stepsize)) + " stable\r
   \n")

   verbosity_output(1, "boring", origin, "Measure at position " + str(BT.getDistance()) + "\r\n")
   verbosity_output(3, "data", origin, "\tm = " + str(m) + "\ts = " + str(s) + "\r\n")
   with open("dbm", "w") as f:
      dbm = str(-18) + '\n'
      meter = BT.getDistance() + 'mm\n'
      setpoint = str(args.0 + (s * stepsize)) + 'mm\n'
      measname = str(args.f) + strDirection + '\n'
      strwrite = dbm + meter + setpoint + measname
      f.write(strwrite)
      verbosity_output(1, "boring", origin, "File written (dbm):\t " + strwrite)
   time.sleep(0.2)

   call(["./autosertest.py"])
   measlength = time.time() - lasttime
   timestr = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(time.time() + (prsteps - prs) * (
   measlength)))
   verbosity_output(1, "boring", origin, "Measurement at position " + str(args.0 + (s * stepsize)
   ) + " finished\r\n")
   verbosity_output(1, "boring", origin, "Measurement took " + str(measlength) + " seconds to
   completion\r\n")
   verbosity_output(0, "boring", origin, "Estimated ETA: " + timestr + "\r\n")

   verbosity_output(1, "boring", origin, "Step " + str(prs + 1) + "/" + str(prsteps) + "
   completed, next Step\r\n")
   verbosity_output(3, "data", origin, "\tm = " + str(m) + "\ts = " + str(s) + "\r\n")

   if not countingBackward:
      s = s + 1
      strDirection = "_Forth"
   elif args.both:
      s = s - 1
      strDirection = "_Back"
   if args.both and s == steps:
      s = s - 1 #here -2 if double measurement at turn point is not wanted. prsteps is than 2 *
      steps - 1
      countingBackward = True
   prs = prs + 1

verbosity_output(1, "boring", origin, "Measurement " + str(m + 1) + " completed, next measurement
\r\n")
verbosity_output(3, "data", origin, "\tm = " + str(m) + "\ts = " + str(s) + "\r\n")

countingBackward = False
m = m + 1
s = 0
prs = 0
```

```
verbosity_output(0, "success", origin, "Measurement progress finished. Exit program")
os.kill(mainpid, signal.SIGINT)
time.sleep(1)
```

Ansteuerung Laserentfernungsmessgerät Disto D3a BT

```
#!/usr/bin/python3

import threading
import time
import socket
import globals
from globals import verbosity_output

class BTDistThread (threading.Thread):

   def __init__(self, threadID, name):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.Distance = -1
      self.BTAddr = "00:13:43:05:f7:1a"
      self.BTSock = socket.socket(socket.AF_BLUETOOTH, socket.SOCK_STREAM, socket.BTPROTO_RFCOMM)
      self.cmdSingle = (0x47, 0x0d, 0x0a)  #G\r\n
      self.cmdCont = (0x48, 0x0d, 0x0a)    #H\r\n
      self.cmdClear = (0x63, 0x0d, 0x0a)   #c\r\n
      self.cmdBTAlwaysOn = (0x73, 0x6f, 0x20, 0x33, 0x0d, 0x0a)  #so 3\r\n
      self.CRLF = '\r\n'
      self.receiveBufferData = ''
      self.receiveBuffer = ''
      self.lastTimeIndex = time.time()
      self.actualMeasurementFrequency = -1
      self.lastCommand = (0x0d, 0x0a)
      self.origin = "\t BT-Thread"
      self.running = True
      self.newMeasurement = False

   def getDistance(self):
      self.newMeasurement = False
      return str(self.Distance)

   def sendCommand(self, cmd):
      for cmdData in cmd:
         self.BTSock.send(bytes([cmdData]))
         time.sleep(0.1)
      self.lastCommand = cmd

   def receive(self):
      while True:
         try:
            self.receiveBuffer = self.BTSock.recv(1)
         except socket.error as e:
            verbosity_output(0, "error", self.origin, "Socket error \"" + str(e) + "\" occured.
            Terminate thread ...\r\n")
            self.running = False
            return

         self.receiveBufferData = self.receiveBufferData + self.receiveBuffer.decode("utf-8")
         if self.CRLF in self.receiveBufferData:
            verbosity_output(3, "success", self.origin, "CRLF detected\r\n")
```



```
            verbosity_output(3, "data", self.origin, "Receive buffer content:\r\n\t               "
            + str(self.receiveBufferData))
            break
      if "@" in self.receiveBufferData[0:5]:
         verbosity_output(1, "error", self.origin, "Error received:\r\n\t               " + str(self
         .receiveBufferData) + "\r\n")
         if "@E203" in self.receiveBufferData[0:5]:
            verbosity_output(1, "warning", self.origin, "Command " + ''.join(chr(i) for i in self.
            lastCommand).replace('\r\n', '') + " not accepted\r\n")
         else:
            verbosity_output(2, "warning", self.origin, "Restart distance measurement\r\n")
            self.sendCommand(self.cmdCont)

      else:
         if "31..00+" in self.receiveBufferData[0:7]:
            self.Distance = int(self.receiveBufferData[7:15])
            self.actualMeasurementFrequency = (1/(time.time() - self.lastTimeIndex))
            self.lastTimeIndex = time.time()
            self.newMeasurement = True
            verbosity_output(2, "data", self.origin, "Measurement frequency:\r\n\t               " +
             str(self.actualMeasurementFrequency) + '\r\n')
         else:
            if "?" in self.receiveBufferData[0:1]:
               verbosity_output(2, "success", self.origin, "Command accepted - " + ''.join(chr(i)
               for i in self.lastCommand))
            else:
               verbosity_output(2, "warning", self.origin, "Damaged transmission received:\r\n\t
                           " + str(self.receiveBufferData))

      self.receiveBufferData = ''

   def clearSocket(self):
      self.sendCommand(self.cmdClear)
      time.sleep(0.5)
      try:
         self.BTSock.settimeout(0.2)
         dump = self.BTSock.recv(8192)
         verbosity_output(3, "data", self.origin, "Clear socket dump content: " + '\r\n\t
                        ' + dump.decode("utf-8").replace('\r\n', '\r\n\t               '))
      except socket.error:
         #Do nothing
         verbosity_output(3, "warning", self.origin, "Socket timed out during socket cleaning\r\n")
      self.BTSock.settimeout(5)

   def stop(self):
      self.running = False

   def run(self):
      verbosity_output(1, "success", self.origin, "Starting " + self.name + "\r\n")
      verbosity_output(1, "warning", self.origin, "Try to connect Bluetooth socket with address " +
      self.BTAddr + "\r\n")
      try:
         self.BTSock.connect((self.BTAddr,1))
      except socket.error:
         verbosity_output(0, "error", self.origin, "Bluetooth connection failed, check if device is
         available\r\n")
         verbosity_output(0, "error", self.origin, "Terminating " + self.name + "\r\n")
         return
      verbosity_output(1, "success", self.origin, "Successfully connected with Bluetooth address " +
       self.BTAddr + "\r\n")

      time.sleep(0.1)
```

```
        self.clearSocket()
        self.sendCommand(self.cmdClear)
        self.receive()

        self.sendCommand(self.cmdBTAlwaysOn)
        self.receive()

        self.sendCommand(self.cmdCont)

        self.lastTimeIndex = time.time()

        verbosity_output(3, "boring", self.origin, "Enter data receiving while loop\r\n")
        while True:
           self.receive()
           if self.running == False:
              break

        verbosity_output(3, "boring", self.origin, "Exit data receiving while loop and sending exit
        commands\r\n")

        self.clearSocket()

        self.sendCommand(self.cmdClear)
        self.receive()

        self.sendCommand(self.cmdBTAlwaysOn)
        self.receive()

        self.BTSock.close()
        verbosity_output(1, "warning", self.origin, "Socket closed and Thread " + self.name + "
        terminated\r\n")
        return
```

Ansteuerung Motor

```
#!/usr/bin/python3

import threading
import time
import globals
from globals import verbosity_output
import wiringpi2 as wp

class DistPIDThread (threading.Thread):

   def __init__(self, threadID, name, distThread, P, I, D, Derivator, Integrator, IntegratorRange,
   OutputRange, allowableError):
      threading.Thread.__init__(self)
      self.threadID = threadID
      self.name = name
      self.distThread = distThread
      self.LastTime = time.time()
      self.Distance = 0
      self.LastDistance = 0
      self.origin = "\tPID-Thread"
      self.running = True
      self.active = False

      self.Kp = P
      self.Ki = I
      self.Kd = D
      self.Derivator = Derivator
      self.Integrator = Integrator
      self.maxIntegrator = IntegratorRange
      self.minIntegrator = -IntegratorRange
      self.PIDRange = OutputRange
      self.PIDsetPoint = 0.0
      self.Kerror = 0.0
      self.maxError = allowableError

      self.newPWMvalue = 0
      self.oldPWMvalue = 0
      self.relaisDirection = 0
      self.minPWMvalue = 100#55
      self.maxStartUpvalue = 100
      self.PWMStartUpHelp = False
      self.PWMStartUpvalue = 0

      self.setPointreached = 0
      self.setPointstable = False

      wp.wiringPiSetup()
      wp.pinMode(0,1)
      wp.digitalWrite(0,self.relaisDirection)
      wp.pinMode(1,2)
      wp.pwmSetMode(0)
      wp.pwmSetRange(1023)
      wp.pwmSetClock(64)
      wp.pwmWrite(1,0)

   def pwmWrite(self, pwmValue):

      if pwmValue <= 0:
         wp.pwmWrite(1,0)
         return
      ramp = abs(int(self.oldPWMvalue))

      while not ramp == pwmValue:
         if ramp < pwmValue:
            ramp = ramp + 1
         if ramp > pwmValue:
            ramp = ramp - 1
         wp.pwmWrite(1,ramp)
         wp.digitalWrite(0, self.relaisDirection)
         time.sleep(0.002)

   def writePWM(self):

      if self.newPWMvalue > self.PIDRange:
         verbosity_output(3, "warning", self.origin, "PWM value of " + str(self.newPWMvalue) + " out
          of range, set to " + str(self.PIDRange))
         self.newPWMvalue = self.PIDRange
      elif self.newPWMvalue < -self.PIDRange:
         verbosity_output(3, "warning", self.origin, "PWM value of " + str(self.newPWMvalue) + " out
          of range, set to " + str(-self.PIDRange))
         self.newPWMvalue = -self.PIDRange

      if abs(int(self.newPWMvalue)) < self.maxStartUpvalue and not self.newPWMvalue == 0:
         self.PWMStartUpHelp = True
         self.PWMStartUpvalue = abs(int(self.newPWMvalue))
      if abs(int(self.newPWMvalue)) < self.minPWMvalue:
         if self.newPWMvalue > 0:
            self.newPWMvalue = self.minPWMvalue
         elif self.newPWMvalue < 0:
```

```
            self.newPWMvalue = -self.minPWMvalue

        verbosity_output(2, "success", self.origin, "Written PWM-Value: " + str(abs(int(self.
        newPWMvalue))) + "\r\n")

        if (self.newPWMvalue < 1 and self.oldPWMvalue < 1) or (self.newPWMvalue > 1 and self.
        oldPWMvalue > 1):
            if self.PWMStartUpHelp:
                verbosity_output(3, "warning", self.origin, "StartupHelp\r\n")
                self.PWMStartUpHelp = False
                wp.digitalWrite(0, self.relaisDirection)
                wp.pwmWrite(1,200)
                time.sleep(0.04)
                wp.pwmWrite(1,abs(int(self.newPWMvalue)))
                wp.digitalWrite(0,self.relaisDirection)
                time.sleep(0.4 * (self.PWMStartUpvalue / self.minPWMvalue))
                wp.pwmWrite(1,0)
                return

            wp.pwmWrite(1,abs(int(self.newPWMvalue)))
            wp.digitalWrite(0,self.relaisDirection)

        else:
            verbosity_output(3, "data" , self.origin, "PWM set to zero for relais direction change to "
             + str(self.relaisDirection) + "\r\n")
            wp.pwmWrite(1,0)
            time.sleep(0.2)
            wp.digitalWrite(0,self.relaisDirection)
            time.sleep(0.2)

            if self.PWMStartUpHelp:
                verbosity_output(3, "warning", self.origin, "StartupHelp\r\n")
                self.PWMStartUpHelp = False
                wp.digitalWrite(0, self.relaisDirection)
                print("Here")
                wp.pwmWrite(1,200)
                time.sleep(0.04)

            wp.pwmWrite(1,abs(int(self.newPWMvalue)))

    def PIDUpdate(self):
        self.Distance = int(self.distThread.getDistance())
        thistime = time.time()
        DiffDistance = self.Distance - self.LastDistance
        velocity = DiffDistance / (thistime - self.LastTime)
        verbosity_output(3, "data", self.origin, "Velocity: " + str(velocity) + " mm/s\r\n")
        self.LastDistance = self.Distance
        self.LastTime = thistime
        self.Kerror = self.PIDsetPoint - int(self.distThread.getDistance())
        verbosity_output(2, "data", self.origin, "Kerror: " + str(self.Kerror) + "\r\n")
        if abs(self.Kerror) < self.maxError:
            self.oldPWMvalue = 0
            self.newPWMvalue = 0
            self.writePWM()
            verbosity_output(2, "data", self.origin, "Set point " + str(self.PIDsetPoint) + " reached\r
            \n")
            self.setPointreached = self.setPointreached + 1
            if self.setPointreached > 3:
                if not self.setPointstable:
                    verbosity_output(1, "success", self.origin, "Set point " + str(self.PIDsetPoint) + "
                    stable\r\n")
                self.setPointstable = True
```



```
                return
            else:
                self.setPointreached = 0
                self.setPointstable = False

            self.Op = self.Kp * self.Kerror
            self.Od = self.Kd * (self.Kerror - self.Derivator)
            self.Derivator = self.Kerror

            self.Integrator = self.Integrator + self.Kerror
            prIntegrator = self.Integrator

            if self.Integrator > self.maxIntegrator:
                self.Integrator = self.maxIntegrator
            elif self.Integrator < self.minIntegrator:
                self.Integrator = self.minIntegrator

            self.Oi = self.Integrator * self.Ki

            self.oldPWMvalue = self.newPWMvalue
            self.newPWMvalue = self.Op + self.Oi + self.Od

            if self.newPWMvalue > 0:
                self.relaisDirection = 0
            elif self.newPWMvalue < 0:
                self.relaisDirection = 1

            tab = "\r\n\t                "
            verbosity_output(3, "data", self.origin, \
            "PID calculation values"           + tab + "setP:    " + str(self.PIDsetPoint) + tab + \
            "Kerror: " + str(self.Kerror)      + tab + "Op:      " + str(self.Op) + tab + \
            "Od:     " + str(self.Od)          + tab + "Oi:      " + str(self.Oi) + tab +\
            "D:      " + str(self.Derivator)   + tab + "I:       " + str(self.Integrator) + tab + \
            "cI:     " + str(prIntegrator)     + tab + "old:     " + str(self.oldPWMvalue)+ tab + \
            "new:    " + str(self.newPWMvalue) + tab + "direc:   " + str(self.relaisDirection) + "\r\n")
            verbosity_output(2, "data", self.origin, "Calculated PWM value: " + str(self.newPWMvalue) + "\
            r\n")

            self.writePWM()

            return

        def setPoint(self, PIDsetPoint):
            self.setPointstable = False
            self.PIDsetPoint = PIDsetPoint
            verbosity_output(2, "data", self.origin, "New set point received: " + str(PIDsetPoint) + "\r\n
            ")

        def stop(self):
            self.running = False
            wp.pwmWrite(1,0)
            wp.pinMode(1,0)
            wp.pinMode(0,0)
            verbosity_output(2, "success", self.origin, "PWM system on chip deinitalized properly\r\n")

        def run(self):
            verbosity_output(1, "success", self.origin, "DistPIDThread " + self.name + " started\r\n")
            while self.running:
                if time.time() - self.distThread.lastTimeIndex < 1:
                    if self.distThread.newMeasurement:
                        if not self.active:
                            verbosity_output(1, "success", self.origin, "PWM started\r\n")
```

```
                verbosity_output(2, "data", self.origin, "PID-Magic with distance " + self.distThread
                .getDistance() + " \r\n")
                self.active = True
                self.PIDUpdate()
            else:
                if self.active:
                self.newPWMvalue = 0
                self.writePWM()
                verbosity_output(1, "warning", self.origin, "PWM stopped due to slow measurement " +
                \
                str(time.time() - self.distThread.lastTimeIndex) + " !< 1\r\n")
                self.active = False
            time.sleep(0.05)
        verbosity_output(1, "warning", self.origin, "DistPIDThread " + self.name + " terminated\r\n")
        return
```

Auswerteprogramm Entfernungsmessung mit internem Temperatursensor

```
import os, time, sys, signal, datetime, math, numpy

def lookupvalue(valuetofind):
    if(valuetofind < lookup[0][0] or valuetofind > lookup[-1][0]):
        raise IndexError("Value not in Lookup-Table")
    offset = valuetofind - lookup[0][0]
    modoffset = offset % 20
    if (modoffset == 0):
        return lookup[int(offset/20)][1]
    else:
        indexbelow = offset - modoffset
        indexabove = indexbelow + 20
        approx = (lookup[int(indexbelow/20)][1] * (20 - modoffset) + lookup[int(indexabove/20)][1] *
         modoffset)/20
        return approx

def printlog(string):
    print(string)
    fLog.write(string + '\n')

def sigmam(listin):
    listmean = numpy.mean(listin)
    liststdev = numpy.std(listin)
    listlen = len(listin)
    listmedian = numpy.median(listin)
    retstr = str(listmean) + ";" + str(liststdev) + ";" + str(liststdev/math.sqrt(listlen)) + ";"  +
     str(listmedian) + ";"
    return retstr

def sigma(listin):
    listmean = numpy.mean(listin)
    liststdev = numpy.std(listin)
    listlen = len(listin)
    listmedian = numpy.median(listin)
    listmode = numpy.bincount(listin).argmax()
    retstr = str(listmean) + ";" + str(liststdev) + ";" + str(liststdev/math.sqrt(listlen)) + ";"  +
     str(listmedian) + ";" + str(listmode) + ";"
    return retstr

def sigma1(listin):
    listmean = numpy.mean(listin)
    liststdev = numpy.std(listin)
    listlen = len(listin)
    listmedian = numpy.median(listin)
```



```
    listbincount = numpy.bincount(listin)
    listmode = listbincount.argmax()

    retstr = str(len(listin)) + ";" + str(listmean) + ";" + str(liststdev) + ";" + str(liststdev/
    math.sqrt(listlen)) + ";" + str(9.2244*liststdev/math.sqrt(listlen)) + ";" + str(listmedian) +
    ";" + str(listmode) + ";"


    listbincount = numpy.append(listbincount,[0,0,0,0,0,0,0,0,0,0,0,0])

    binstr = ''
    for bin in range(508,532):

        colorval = int(listbincount[bin] / (listlen*0.3) * 1020)
        colorblue = 0
        colorred = 0
        colorgreen = 0
        if colorval > 1020:
            colorval = 1020
        if colorval > 765:
            colorred = 255
            colorblue = colorval - 765
        else:
            if colorval > 510:
                colorred = 255
                colorgreen = 765 - colorval
            else:
                if colorval > 255:
                    colorgreen = 255
                    colorred = colorval - 255
                else:
                    colorgreen = 255
                    colorred = 255 - colorval
                    colorblue = colorred
        color = colorred + colorgreen * 256 + colorblue * 256 * 256
        binstr += str(bin) + ";" + str(color) + ";"
    print(binstr)
    print(listbincount[listmode])
    print("%s %%" % (str(100*listbincount[listmode]/listlen)))

    return (retstr,binstr)


lookup = list()
folderpath = "D:/Studium/EMP/Masterarbeit/Daten/20140606_Differenztemperaturcycle"
filenamelog =  time.strftime('%Y%m%d_%H%M%S_' + 'Log.txt')
fLog = open(folderpath + "/" + filenamelog, 'w')

count = 0
folderpath += "/data/RF4/Tempcycle_Sender/"
fOutput = open(folderpath + time.strftime('%Y%m%d_%H%M%S_' + 'Output30.txt'), 'w')
strHeader = "Linux Time;Date;Temperatur Master;Temperatur Slave;#;Offset;Offset Standardabweichung;
Offset Standardabweichung normiert; Offset; Offset Median; Offset Modus;Verarbeitungszeit Master;
Verarbeitungszeit Master Standardabweichung; Verarbeitungszeit Master Standardabweichung normiert;
Verarbeitungszeit Master Median; Verarbeitungszeit Master Modus; Verarbeitungszeit Slave;
Verarbeitungszeit Slave Standardabweichung;Verarbeitungszeit Slave Standardabweichung normiert;
Verarbeitungszeit Slave Median; Verarbeitungszeit Slave Modus;# RSSI;RSSI Master; RSSI Master
Standardabweichung; RSSI Master Standardabweichung normiert; RSSI Master Median; RSSI Slave; RSSI
Slave Standardabweichung; RSSI Slave Standardabweichung normiert;RSSI Slave Median
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;\ns;;Â°C;Â°C;;Zyklen;Zyklen;Zyklen;m;Zyklen;Zyklen;Zyklen;Zyklen;
Zyklen;Zyklen;Zyklen;Zyklen;Zyklen;Zyklen;Zyklen;Zyklen;;dBm;dBm;dBm;dBm;dBm;dBm;dBm;\n"
fOutput.write(strHeader)
```

```
print(folderpath)

for filename in os.listdir(folderpath):
    print(filename)
    if not ("_mean_" in filename or "_time_0_" in filename or "Output" in filename or ".exe" in
    filename):
        filedate = int(time.mktime(time.strptime(filename[:15], '%Y%m%d_%H%M%S')))
        list0 = list()
        list1 = list()
        list2 = list()
        list3 = list()
        list4 = list()
        listint = list()
        listext = list()
        with open(folderpath + filename, 'r') as measurementfile:
            discard = 0
            measurementarr = measurementfile.readline().split()
            listint.append(float(measurementarr[1]))
            listext.append(float(measurementarr[3]))
            for line in measurementfile:
                parts = line.split(";")
                list0.append(int(parts[0]))
                list1.append(int(parts[1]))
                list2.append(int(parts[2]))
                if not (discard < 60):
                    list3.append(float(parts[3]))
                    list4.append(float(parts[4]))
                discard = discard + 1
        dellist = []
        for i in range(len(list0)):
            if (list1[i] == 0 or list2[i] == 0 or list0[i] < 500 or list0[i] > 540):
                dellist.append(i)
        for item in dellist[::-1]:
            del list0[item]
            del list1[item]
            del list2[item]

        dellist = []

        if(len(list3) > 100 and len(list0) > 0):

            listmean = numpy.mean(list3)

            for i in range(len(list3)):
                if(list3[i] > (listmean + 3) or list3[i] < (listmean - 3)):
                    dellist.append(i)
            printlog("LÃ¤nge Dellist:" + str(len(dellist)))
            for item in dellist[::-1]:
                printlog(str(list3[item]) + ' ' + str(item))
                del list3[item]
            if(len(dellist) > 100):
                printlog("Warning: Possible measurement spoil in " + folderpath + filename + " ,
                Number of deleted RSSI measurements: %s" % len(dellist))

            dellist = []
            listmean = numpy.mean(list4)
            for i in range(len(list4)):
                if(list4[i] > (listmean + 3) or list4[i] < (listmean - 3)):
                    dellist.append(i)
            printlog("LÃ¤nge Dellist:" + str(len(dellist)))
            for item in dellist[::-1]:
                printlog(str(list4[item]) + ' ' + str(item))
```



```
                del list4[item]
            if(len(dellist) > 100):
                printlog("Warning: Possible measurement spoil in " + folderpath + filename + " ,
                Number of deleted RSSI measurements: %s" % len(dellist))

            list0strarr = sigma1(list0)

            filedateformat = time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(filedate))
            datastr = str(filedate) + ";" + filedateformat + ";" + str(numpy.mean(listint)) + ";" +
            str(numpy.mean(listext)) + ";" + list0strarr[0] + sigma(list1) + sigma(list2) + str(len(
            list3)) + ";" + sigmam(list3) + sigmam(list4) + list0strarr[1]
            print(datastr)
            fOutput.write(datastr + "\n")
            count += 1
            #if count > 200:
            #    sys.exit()
sys.exit()
dirlist = os.listdir(folderpath)

fOutputclean = open(folderpath + time.strftime('%Y%m%d_%H%M%S_' + 'Output_clean.txt'), 'w')

for filename in dirlist:

    if ("Output" in filename):
        vInput = list()
        vOutput = list()
        fOutputclean = open(folderpath + time.strftime('%Y%m%d_%H%M%S_' + 'Output_clean.txt'), 'w')
        with open(folderpath + filename, "r") as fInput:
            vOutput.append(fInput.readline())
            vOutput.append(fInput.readline())
            for line in fInput:
                vInput.append(line)

        offsetlist = list()
        for line in vInput:
            offsetlist.append(float(line.split(";")[4]))

        offsetmean = numpy.mean(offsetlist)
        offsetstd = numpy.std(offsetlist)
        print(offsetmean)
        print(offsetstd)

        for line in vInput:
            if(float(line.split(";")[4]) > (offsetmean + offsetstd) or float(line.split(";")[4]) < (
            offsetmean - offsetstd)):
                print(line.split(";")[4])
            else:
                vOutput.append(line)
        for line in vOutput:
            fOutputclean.write(line)
        fOutputclean.close()

dirlist = os.listdir(folderpath)

fOutputclean = open(folderpath + time.strftime('%Y%m%d_%H%M%S_' + 'Output_clean.txt'), 'w')

for filename in dirlist:

    if ("Output" in filename):
        vInput = list()
        vOutput = list()
        fOutputclean = open(folderpath + time.strftime('%Y%m%d_%H%M%S_' + 'Output_clean.txt'), 'w')
```

```
        with open(folderpath + filename, "r") as fInput:
            vOutput.append(fInput.readline())
            vOutput.append(fInput.readline())
            for line in fInput:
                vInput.append(line)

        offsetlist = list()
        for line in vInput:
            offsetlist.append(float(line.split(";")[4]))

        offsetmean = numpy.mean(offsetlist)
        offsetstd = numpy.std(offsetlist)
        print(offsetmean)
        print(offsetstd)

        for line in vInput:
            if(float(line.split(";")[4]) > (offsetmean + offsetstd) or float(line.split(";")[4]) < (
            offsetmean - offsetstd)):
                print(line.split(";")[4])
            else:
                vOutput.append(line)
        for line in vOutput:
            fOutputclean.write(line)
        fOutputclean.close()
sys.exit()
```

Hauptprogramm Entfernungsmessung

```
#define BPressDemo
#define MTSender
//#define noRPi
//#define MTResponder
#define TimerAXT2
#include "main.h"
#include "RF1A.h"
#include "SPIBitBang.h"
#include "cc430f6137.h"
#include "inc/hw_memmap.h"
#include "cc430.h"
#include "fm25h20.h"
#include "driverlib/5xx_6xx/timera.h"


extern RF_SETTINGS rfSettings2;

unsigned char packetReceived;
unsigned char packetTransmit;

unsigned char RxBuffer[64];
unsigned char RxBufferLength = 0;
unsigned char TxBuffer[7];
unsigned char buttonPressed = 0;
unsigned char goforit = 0;
unsigned char operating = 0;
unsigned int i = 0;

unsigned char transmitting = 0;
unsigned char receiving = 0;

unsigned char state;
uint32 statecounter;
uint16 cycles;
```



```
uint16 maxcycles;
uint32 retry;
uint16 bigretry;
uint16 packetloss;
uint16 mnum;
uint8 rssi_int;
uint8 rssi_ext;
uint16 crcerr;
uint32 idle;
uint8 radio_setting;
uint8 radio_settings_changed;
uint8 tmp;
//uint32 clkint[MAX_CYCLES];
//uint32 clkext[MAX_CYCLES];
uint8 membuf[10];
uint8 membuf_last[10];

uint8 package[100];
uint16 package_length = 0;

uint32 timeSyncWord_temp = 0;
uint32 timeSyncWord_received = 0;
uint32 timeSyncWord_sent = 0;
uint32 timeSyncWord_overflow = 0;
uint32 timeSyncWord = 0;
uint8 *timeSync = 0;
uint16 temp_ext = 0;
uint16 temp_temp = 0;
uint8 temp_ext_H = 0;
uint8 temp_ext_L = 0;
uint8 temp_ext_HL = 0;
uint8 temp_toggle = 0;

uint32 baseaddr, fram_size, max_cycles, membuf_length, measurement_size, fram_page, last_addr;


#ifdef BPressDemo

void main (void)
{


        SystemBoot();

        while(0){

                temp_ext = ADT7320Temp();
                temp_ext_H = (temp_ext >> 8);
                temp_ext_H &= ~BIT0;
                temp_ext_L = (temp_ext >> 1);
                temp_ext_L |= BIT0;

                temp_ext = temp_ext * 2;

                //if(temp_ext_L & BIT0){

                        temp_ext &= ~0x1ff;
                        if(temp_ext_L & BIT7)
                                temp_ext |= 0x100;
                        temp_ext_L &= ~BIT0;
                        temp_temp = temp_ext_L;
                        temp_ext |= (temp_temp << 1);
```

```
                //}
                //else{
                        temp_ext &= ~0xfe00;
                        temp_temp = temp_ext_H;
                        temp_ext |= (temp_temp << 8);
                //}

        }

        /* Testcode für Clocks etc. fLED = 1,428 MHz @ 20 MHz Mainclock
    P1OUT ^= BIT0;
    P1DS |= BIT0;
    P1DIR |= BIT0;
    P1REN |= BIT0;
        while(1){
                _delay_cycles(10000000);
                P1OUT ^= BIT0;
        }// */

        #ifdef MTSender
        state = sIDLE;
        #endif
        #ifdef MTResponder
        state = sRECEIVEON;
        #endif
        statecounter = 0;

        /* for(i = 0; i < (MAX_CYCLES); i++){
                clkint[i] = 0;
                clkext[i] = 0;
         }*/
         for(i = 0; i < (MEMBUF_LENGTH); i++){
                 membuf[i] = 0;
                 membuf_last[i] = 0;
         }
         for(i = 0; i < (sizeof(RxBuffer)); i++){
                 RxBuffer[i] = 0xff;
         }

         ReceiveOn();
         receiving = 1;
         package_length = 0;

         TxBuffer[0] = PACKET_LEN;
         TxBuffer[1] = 0xAA;
         TxBuffer[2] = 0xBB;
         TxBuffer[3] = 0xCC;
         TxBuffer[4] = 0xEE;
         TxBuffer[5] = 0xFE;

        cycles = 0;
    mnum = 0;
    goforit = 0;
    operating = 0;
#ifdef noRPi
    operating = 1;
#endif
    retry = cRETRY;
    bigretry = cBIGRETRY;
    packetloss = 0;
    rssi_int = 0;
```



```
rssi_ext = 0;
crcerr = 0;
idle = 0;
radio_setting = 4;//0
radio_settings_changed = 0;
maxcycles = (uint16)MAX_CYCLES;
fram_size = (uint32)FRAM_SIZE;
max_cycles = (uint32)MAX_CYCLES;
membuf_length = (uint32)MEMBUF_LENGTH32;
measurement_size = max_cycles * membuf_length;
fram_page = fram_size / measurement_size;


timeSyncWord_overflow = 0;

__bis_SR_register(GIE);

TA0CCTL3 = CM_1 + CCIS_1 + CAP + CCIE;       //Capture on rising edge, capture on CCI0B (GD01),
Asynchronous capture, Capture mode
__delay_cycles(600000);
    while(1){


           switch(__even_in_range(state,18)){

                   case sSTOP:
                           RF1AIE &= ~(BIT9 + BIT1);                        // Disable RX
                           interrupts
                           RF1AIFG &= ~(BIT9 + BIT1);                       // Clear pending
                           IFG
                       Strobe( RF_SXOFF );
                           state = sIDLE;
                           break;
                   case sIDLE:
                           #ifdef MTResponder
                   P1IE |= BIT7;                           // Re-enable button press
                           #endif
                   if(UCA0IFG&UCRXIFG){

                   }
                           if(operating){
                                   baseaddr = (mnum % fram_page) * measurement_size;
                                   last_addr = baseaddr;
                                   maxcycles = (uint16)max_cycles;
                                   cycles = 0;
                                   state = sSEND;
                           }
                           break;
                   case sSEND:
                           #ifdef MTSender
                           ToggleLED1();
                           #endif
                           #ifdef MTResponder
                           ToggleLED2();
                           #endif
                           receiving = 0;
                   RF1AIE &= ~(BIT9 + BIT1 + BIT2);                         // Disable RX
                   interrupts
                   RF1AIFG &= ~(BIT9 + BIT1 + BIT2);                        // Clear pending IFG
                   RF1AIES |= (BIT9);                                               //IFG9 is
                   inverted
                   RF1AIFG &= ~(BIT9 + BIT1 + BIT2);          // Clear pending interrupts
```

```
    RF1AIE |= BIT9 + BIT1 + BIT2;              // Enable TX end-of-packet and
    Sync Word interrupt
    WriteBurstReg(RF_TXFIFOWR, (unsigned char*)TxBuffer, 0x07);
    #ifdef MTSender
            #ifdef TimerAXT2
    timeSyncWord_overflow = 0;
TA0CTL = TASSEL__ACLK + ID_0 + MC_2  + TAIE; //Clocksource Select A_CLOCK (XT2),
 Input Divider = 1, Count cont upward
            #else
timeSyncWord_overflow = 0;
TA0CTL = TASSEL__SMCLK + ID_0 + MC_2 + TAIE; //Clocksource Select SM_CLOCK,
Input Divider = 1, Count cont upward
            #endif
            #endif
Strobe( RF_STX );                         // Strobe STX
transmitting = 1;
            state = sWAIT;
            break;
    case sWAIT:
            if (!transmitting)
                    if(cycles)
                            state = sMEM;
                    else
                            state = sRECEIVEON;
            break;
    case sRECEIVEON:
        RF1AIES |= BIT9;                                        // Falling edge
        of RFIFG9
        RF1AIFG &= ~(BIT9 + BIT1 + BIT2);                       // Clear a
        pending interrupt
        RF1AIE  |= BIT9 + BIT1 + BIT2;                          // Enable the
        interrupt
            #ifdef MTResponder
            #ifdef TimerAXT2
        timeSyncWord_overflow = 0;
        TA0CTL = TASSEL__ACLK + ID_0 + MC_2 + TAIE; //Clocksource Select A_CLOCK
         (XT2), Input Divider = 1, Count cont upward
            #else
        timeSyncWord_overflow = 0;
        TA0CTL = TASSEL__SMCLK + ID_0 + MC_2 + TAIE; //Clocksource Select
        SM_CLOCK, Input Divider = 1, Count cont upward
            #endif
            #endif
        Strobe( RF_SRX );
        receiving = 1;
retry = cRETRY;                     //Reset Retry counter
            state = sRECEIVE;
            break;
    case sRECEIVE:
            if(goforit){
                    #ifdef MTSender
                    cycles++;
                    #endif
                    if(cycles < maxcycles - 1){
                            //__delay_cycles(5000000);
                            state = sSEND;
                            goforit = 0;
                    }
                    else{
                            state = sCOMM;
                            goforit = 0;
                    }
```



```
                    }
                    if(retry == 0){
#ifdef MTSender
                            state = sSEND;
#endif
                        #ifdef MTResponder
                        state = sRECEIVEOFF;
#endif
                    }
                    if(bigretry == 0){
                        state = sRESET;
                    }
                    retry--;
                    break;
        case sRECEIVEOFF:
            RF1AIE &= ~(BIT9 + BIT1);                       // Disable RX
            interrupts
            RF1AIFG &= ~(BIT9 + BIT1);                      // Clear pending IFG
            Strobe( RF_SIDLE );
            Strobe( RF_SFRX  );
            retry = cRETRY;
                        //Reset retry counter
            bigretry--;
                #ifdef MTResponder
            temp_ext = ADT7320Temp();
            temp_ext_H = (temp_ext >> 8);
            temp_ext_H &= ~BIT0;
            temp_ext_L = (temp_ext >> 1);
            temp_ext_L |= BIT0;
                #endif
                state = sRECEIVEON;
                break;
            case sCOMM:
//                  SaveMeasurement(membuf, mnum, cycles-1);
                    last_addr = SaveMeasurement(membuf, last_addr);
                    membuf[0] = membuf_last[0];
                    membuf[1] = membuf_last[1];
                    membuf[2] = membuf_last[2];
                    membuf[3] = membuf_last[3];
                    membuf[8] = membuf_last[8];
                    membuf[4] = 0;
                    membuf[5] = 0;
                    membuf[6] = 0;
                    membuf[7] = 0;
                    membuf[9] = 0;
//                  SaveMeasurement(membuf, mnum, cycles);
                    last_addr = SaveMeasurement(membuf, last_addr);

                    //package_length = addUint16toPackage(package, ADT7320Temp(),
                    TEMP_INT, package_length);
                    //package_length = addUint16toPackage(package, 0x0A, TEMP_EXT,
                    package_length);
                    //package_length = SendUart(package, package_length);

//                  FM25_ReadFromAddress(RxBuffer,63, 0x0);
/*                  package_length = addUint16toPackage(package, mnum, MNUM,
package_length);
 *              package_length = addUint16toPackage(package, mnum, MNUM, package_length);
                package_length = SendUart(package, package_length);
                    package_length = addUint8toPackage(package, rssi_int, RSSI_INT,
                    package_length);
```

```
                        package_length = addUint8toPackage(package, rssi_ext, RSSI_INT,
                        package_length);
                        package_length = addUint16toPackage(package, crcerr, CRC_ERR,
                        package_length);
                        package_length = addUint32toPackage(package, idle, IDLE_COUNT,
                        package_length);
                        package_length = addUint16toPackage(package, packetloss, PKT_LOSS,
                        package_length);
                        package_length = SendUart(package, package_length);*/
//                      SendUartData(clkint, clkext, MAX_CYCLES);
#ifndef noRPi
                        SendMeasurement(baseaddr, maxcycles, ADT7320Temp(), temp_ext);
#endif
                        goforit = 0;
                        packetloss = 0;
                        crcerr = 0;
                        mnum++;
                        operating = 0;
//                      __delay_cycles(20000000);
                        if(radio_settings_changed)
                                state = sRESET;
                        else
                                state = sIDLE;
                        break;
                case sRESET:
                        Strobe(RF_SRES);                        // Reset the Radio Core
                        Strobe(RF_SNOP);                        // Reset Radio Pointer
                    InitRadio(radio_setting);
                    radio_settings_changed = 0;
                    __delay_cycles(600000);
                        TxBuffer[1] = 0xAA;                                             //
                        Reset TxBuffer after transmitting new radio settings
                        TxBuffer[2] = 0xBB;
                        TxBuffer[3] = 0xCC;
                        TxBuffer[4] = 0xEE;
                        #ifdef MTSender
                    state = sIDLE;
                        #endif
                        #ifdef MTResponder
                    state = sRECEIVEON;
                        #endif
                    bigretry = cBIGRETRY;                                       //Reset
                    Bigretry counter
                        break;
                case sMEM:
                        if(cycles == 1){                                                //
                         erste empfangene Messung verwerfen
                                membuf[0] = 0;
                                membuf[1] = 0;
                                membuf[2] = 0;
                                membuf[3] = 0;
                                membuf[8] = 0;
                        }
//                      SaveMeasurement(membuf, mnum, cycles-1);  // Messung letzte Runde
speichern
                        last_addr = SaveMeasurement(membuf, last_addr);
                        membuf[0] = membuf_last[0];                              // Messung
                         dieser Runde für Speicherung nächste runde vorbereiten
                        membuf[1] = membuf_last[1];
                        membuf[2] = membuf_last[2];
                        membuf[3] = membuf_last[3];
                        membuf[8] = membuf_last[8];
```



```
                                state = sRECEIVEON;
                }

//              statecounter++;


        }

}

#endif

#pragma vector=TIMER0_A1_VECTOR
__interrupt void TIMER0_A1_ISR(void)
{
  switch(__even_in_range(TA0IV,14))
  {
    case  0: break;                          // No interrupt
    case  2: break;                          // CCR1 not used
    case  4: break;                          // CCR2 not used
    case  6:
        TA0CTL &= ~(TAIFG);
        timeSyncWord = (timeSyncWord_overflow << 16) + TA0CCR3;
        break;                          // reserved
    case  8: break;                          // reserved
    case 10: break;                          // reserved
    case 12: break;                          // reserved
    case 14:
        TA0CTL &= ~(TAIFG);
        timeSyncWord_overflow++;        // overflow
            break;
    default: break;
  }
}

#ifdef BPressDemo
#ifdef MTSender

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
        switch(__even_in_range(UCA0IV,4))
        {
        case 0: break;
        case 2:
                __bic_SR_register(GIE);
                UCA0IV &= ~BIT1;
                if(UCA0RXBUF != 0xd3){
                        __bis_SR_register(GIE);
                        break;
                }
                mnum = 0;
                tmp = 0;
                max_cycles = 0;
                while(!(UCA0IFG&UCRXIFG));
                operating = UCA0RXBUF;
                while(!(UCA0IFG&UCRXIFG));
                tmp = UCA0RXBUF;
                if(tmp != radio_setting){
                        radio_setting = tmp;
                        TxBuffer[1] = radio_setting;
                        TxBuffer[2] = radio_setting;
```

```
                    TxBuffer[3] = radio_setting;
                    TxBuffer[4] = radio_setting;
                    radio_settings_changed = 1;
                }
                while(!(UCA0IFG&UCRXIFG));
                mnum += (UCA0RXBUF << 8);
                while(!(UCA0IFG&UCRXIFG));
                mnum += UCA0RXBUF;
                while(!(UCA0IFG&UCRXIFG));
                max_cycles += (UCA0RXBUF << 8);
                while(!(UCA0IFG&UCRXIFG));
                max_cycles += UCA0RXBUF;

                measurement_size = max_cycles * membuf_length;
            fram_page = fram_size / measurement_size;
                __no_operation();
                //transmitting = 0;
                //state = sIDLE;
                __bis_SR_register(GIE);
                break;
        case 4: break;
        default: break;
        }
}

#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
  switch(__even_in_range(P1IV, 16))
  {
    case  0: break;
    case  2: break;                         // P1.0 IFG
    case  4: break;                         // P1.1 IFG
    case  6: break;                         // P1.2 IFG
    case  8: break;                         // P1.3 IFG
    case 10: break;                         // P1.4 IFG
    case 12: break;                         // P1.5 IFG
    case 14: break;                         // P1.6 IFG
    case 16:                                // P1.7 IFG
      P1IE = 0;                             // Debounce by disabling buttons
      if(operating)
          operating = 0;
      else
          operating = 1;
      //__bic_SR_register_on_exit(LPM3_bits); // Exit active
      break;
  }
}

#pragma vector=CC1101_VECTOR
__interrupt void CC1101_ISR(void)
{
  switch(__even_in_range(RF1AIV,32))        // Prioritizing Radio Core Interrupt
  {
    case  0: break;                         // No RF core interrupt pending
    case  2: break;                         // RFIFG0
    case  4: break;
    case  6: break;                         // RFIFG2
    case  8: break;                         // RFIFG3
    case 10: break;                         // RFIFG4
    case 12: break;                         // RFIFG5
    case 14: break;                         // RFIFG6
```



```
    case 16: break;                         // RFIFG7
    case 18: break;                         // RFIFG8
    case 20:                                // RFIFG9
      RF1AIFG &= ~BIT9;

      if(receiving)                         // RX end of packet
      {

        timeSyncWord_received = timeSyncWord;

        TA0CTL  &= ~(MC_3 + TAIE);
        TA0CTL |= TACLR;

        timeSyncWord_temp = timeSyncWord_received - timeSyncWord_sent;

        timeSync = (uint8*) &timeSyncWord_temp;

        membuf_last[3] = *timeSync++;
        membuf_last[2] = *timeSync++;
        membuf_last[1] = *timeSync++;
        membuf_last[0] = *timeSync;

//              clkint[cycles+1] = timeSyncWord_temp;


        //P3OUT != BIT4;
        // Read the length byte from the FIFO
        RxBufferLength = ReadSingleReg( RXBYTES );
        ReadBurstReg(RF_RXFIFORD, RxBuffer, RxBufferLength);
        rssi_int = ReadSingleReg(RSSI);
        membuf_last[8] = rssi_int;

        timeSync = (uint8*) &timeSyncWord_temp;

        *timeSync++ = RxBuffer[4];
        membuf[7] = RxBuffer[4];
        *timeSync++ = RxBuffer[3];
        membuf[6] = RxBuffer[3];
        *timeSync++ = RxBuffer[2];
        membuf[5] = RxBuffer[2];
        *timeSync = RxBuffer[1];
        membuf[4] = RxBuffer[1];

//        clkext[cycles] = timeSyncWord_temp;
        rssi_ext = RxBuffer[5];
        membuf[9] = RxBuffer[5];

        temp_ext_HL = RxBuffer[6];

                if(temp_ext_HL & BIT0){
                        temp_ext &= ~0x1ff;
                        if(temp_ext_HL & BIT7)
                                temp_ext |= 0x100;
                        temp_ext_HL &= ~BIT0;
                        temp_temp = temp_ext_HL;
                        temp_ext |= (temp_temp << 1);
                }
                else{
                        temp_ext &= ~0xfe00;
                        temp_temp = temp_ext_HL;
                        temp_ext |= (temp_temp << 8);
                }
```

```
        // Stop here to see contents of RxBuffer
        __no_operation();

        // Check the CRC results
        if(RxBuffer[CRC_LQI_IDX] & CRC_OK){
          crcerr++;
        }

        //Bounce packet when correct packet received - Thor
        if (RxBuffer[0] == PACKET_LEN)
                goforit = 1;
      }
      else if(transmitting)                 // TX end of packet
      {
        timeSyncWord_sent = timeSyncWord;

        //P3OUT &= ~BIT4;
        RF1AIE &= ~BIT9;                    // Disable TX end-of-packet interrupt
        //P3OUT &= ~BIT6;                     // Turn off LED after Transmit
        transmitting = 0;
      }
      else while(1);                        // trap
      break;
    case 22: break;                         // RFIFG10
    case 24: break;                         // RFIFG11
    case 26: break;                         // RFIFG12
    case 28: break;                         // RFIFG13
    case 30: break;                         // RFIFG14
    case 32: break;                         // RFIFG15
  }
  //__bic_SR_register_on_exit(LPM3_bits);
}
#endif
#ifdef MTResponder

#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
{
        __no_operation();
}

#pragma vector=PORT1_VECTOR
__interrupt void PORT1_ISR(void)
{
  switch(__even_in_range(P1IV, 16))
  {
    case  0: break;
    case  2: break;                         // P1.0 IFG
    case  4: break;                         // P1.1 IFG
    case  6: break;                         // P1.2 IFG
    case  8: break;                         // P1.3 IFG
    case 10: break;                         // P1.4 IFG
    case 12: break;                         // P1.5 IFG
    case 14: break;                         // P1.6 IFG
    case 16:                                // P1.7 IFG
      P1IE = 0;                             // Debounce by disabling buttons
      if(operating)
          operating = 0;
      else
          operating = 1;
      //__bic_SR_register_on_exit(LPM3_bits); // Exit active
      break;
  }
}

#pragma vector=CC1101_VECTOR
__interrupt void CC1101_ISR(void)
{
  switch(__even_in_range(RF1AIV,32))        // Prioritizing Radio Core Interrupt
  {
    case  0: break;                         // No RF core interrupt pending
    case  2: break;                         // RFIFG0
    case  4: break;
    case  6: break;                         // RFIFG2
    case  8: break;                         // RFIFG3
    case 10: break;                         // RFIFG4
    case 12: break;                         // RFIFG5
    case 14: break;                         // RFIFG6
    case 16: break;                         // RFIFG7
    case 18: break;                         // RFIFG8
    case 20:                                // RFIFG9
      RF1AIFG &= ~BIT1;
      if(receiving)                         // RX end of packet
      {
        timeSyncWord_received = timeSyncWord;

        // Read the length byte from the FIFO
        RxBufferLength = ReadSingleReg( RXBYTES );
        ReadBurstReg(RF_RXFIFORD, RxBuffer, RxBufferLength);
        rssi_ext = ReadSingleReg(RSSI);

        // Stop here to see contents of RxBuffer
        __no_operation();

        // Check the CRC results
        //if(RxBuffer[CRC_LQI_IDX] & CRC_OK)
        //  P1OUT ^= BIT0;                    // Toggle LED1

        //Bounce packet when correct packet received - Thor
        if (RxBuffer[0] == PACKET_LEN)
                goforit = 1;
        if (RxBuffer[1] == RxBuffer [2] && RxBuffer[2] == RxBuffer[3] && RxBuffer[3] == RxBuffer[4])
                radio_setting = RxBuffer[1];
      }
      else if(transmitting)                 // TX end of packet
      {
       timeSyncWord_sent = timeSyncWord;

       TA0CTL  &= ~(MC_3 + TAIE);
       TA0CTL |= TACLR;

       timeSyncWord_temp = timeSyncWord_sent - timeSyncWord_received;

           TxBuffer[1] = (timeSyncWord_temp >> 24);
           TxBuffer[2] = (timeSyncWord_temp >> 16);
           TxBuffer[3] = (timeSyncWord_temp >> 8);
           TxBuffer[4] = timeSyncWord_temp;
           TxBuffer[5] = rssi_ext;
           if(temp_toggle){
                   TxBuffer[6] = temp_ext_H;
                   temp_toggle = 0;
```

```
            }
            else{
                    TxBuffer[6] = temp_ext_L;
                    temp_toggle = 1;
            }



       RF1AIE &= ~BIT9;                    // Disable TX end-of-packet interrupt
       //P3OUT &= ~BIT6;                     // Turn off LED after Transmit
       transmitting = 0;
      }
      else while(1);                        // trap
      break;
    case 22: break;                         // RFIFG10
    case 24: break;                         // RFIFG11
    case 26: break;                         // RFIFG12
    case 28: break;                         // RFIFG13
    case 30: break;                         // RFIFG14
    case 32: break;                         // RFIFG15
  }
  //__bic_SR_register_on_exit(LPM3_bits);
}
#endif
#endif

void toTrapornottoTrap(){
//      while(1);
        __no_operation();
}

#pragma vector=AES_VECTOR
__interrupt void AES_ISR(void)
{toTrapornottoTrap();}

#pragma vector=RTC_VECTOR
__interrupt void RTC_ISR(void)
{toTrapornottoTrap();}

#pragma vector=LCD_B_VECTOR
__interrupt void LCD_ISR(void)
{toTrapornottoTrap();}

#pragma vector=PORT2_VECTOR
__interrupt void PORT2_ISR(void)
{toTrapornottoTrap();}

#pragma vector=TIMER1_A1_VECTOR
__interrupt void TIMER1_A1_ISR(void)
{toTrapornottoTrap();}

#pragma vector=TIMER1_A0_VECTOR
__interrupt void TIMER1_A0_ISR(void)
{toTrapornottoTrap();}

#pragma vector=DMA_VECTOR
__interrupt void DMA_ISR(void)
{toTrapornottoTrap();}

#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR(void)
{toTrapornottoTrap();}
```



```
#pragma vector=ADC12_VECTOR
__interrupt void ADC12_ISR(void)
{toTrapornottoTrap();}

#pragma vector=USCI_B0_VECTOR
__interrupt void USCI_B0_ISR(void)
{toTrapornottoTrap();}

#pragma vector=WDT_VECTOR
__interrupt void WDT_ISR(void)
{toTrapornottoTrap();}

#pragma vector=COMP_B_VECTOR
__interrupt void COMP_B_ISR(void)
{toTrapornottoTrap();}

#pragma vector=UNMI_VECTOR
__interrupt void UNMI_ISR(void)
{toTrapornottoTrap();}

#pragma vector=SYSNMI_VECTOR
__interrupt void SYSNMI_ISR(void)
{__no_operation();}

#pragma vector=RESET_VECTOR
__interrupt void RESET_ISR(void)
{__no_operation();}

#ifdef backup
void main (void)
{
        SystemBoot();

        ReceiveOn();
        receiving = 1;

        package_length = 0;

        TxBuffer[0] = PACKET_LEN;
        TxBuffer[1] = 0xAA;
        TxBuffer[2] = 0xBB;
        TxBuffer[3] = 0xCC;
        TxBuffer[4] = 0xEE;
        TxBuffer[5] = 0xFF;

        package_length = addStringtoPackage(package, "Hier gehts los: ", ORANGE, package_length);
        package_length = SendUart(package, package_length);

/*      package_length = addStringtoPackage(package, "Size of TxBuffer: ", ORANGE, package_length);
        package_length = addUint8toPackage(package, sizeof TxBuffer, PI_RFSTATUS, package_length);
        package_length = SendUart(package, package_length);
*/
    cycles = 0;
    maxcycles = MAX_CYCLES;

    //WriteBurstReg(RF_TXFIFOWR, (unsigned char*)TxBuffer, sizeof TxBuffer);
    __bis_SR_register(GIE);
        while (1)
          {
            // __bis_SR_register( LPM3_bits + GIE );
            //__no_operation();
```

```
            if (buttonPressed)                      // Process a button press->transmit
            {
              if (cycles <= maxcycles){
                P3OUT |= BIT6;                       // Pulse LED during Transmit
                buttonPressed = 0;
                P1IFG = 0;
                ReceiveOff();
                receiving = 0;
                Transmit( (unsigned char*)TxBuffer, sizeof TxBuffer);
                //RF1AIES |= BIT9;
               // RF1AIFG &= ~(BIT9 + BIT1);               // Clear pending interrupts
               // RF1AIE |= BIT9 + BIT1;                   // Enable TX end-of-packet and Sync
               Word interrupt
               // Strobe( RF_STX );                       // Strobe STX
                transmitting = 1;
                cycles++;
              }

                P1IE |= BIT7;                        // Re-enable button press
                ToggleLED2();
            }

            else if(!transmitting && !receiving)
            {
              ReceiveOn();
              receiving = 1;
            }
            if (cycles == maxcycles){
                for(i = 0; i <= maxcycles; i++){
                package_length = addUint32toPackage(package, timer[i], CLK_INT, package_length);
                }
                package_length = addStringtoPackage(package, "Hier gehts los: ", ORANGE,
                package_length);
                package_length = SendUart(package, package_length);
                cycles = 0;
                buttonPressed = 0;
            }
          }
#endif
```

Bibliothek für FRAM FM25H20

```
//#+-*§----------------------------------------------------------------------§*-+#
//#+-*  Library for FM25H20 Serial F-RAM Memory                                *-+#
//#+-   Copyright © 2013 Thorbjörn Jörger                                      -+#
//#+                                                                            +#
//#+-   Lehrstuhl für elektrische Mess- und Prüfverfahren - IMTEK              -+#
//#+-*  Albert-Ludwigs-Universität Freiburg im Breisgau                        *-+#
//#+-*§----------------------------------------------------------------------§*-+#
//#+-*+       o     +              o            o     +           o     +     *-+#
//#+-     +             o     +       +     +               o                -+#
//#+  o          +                      o            +          +     o   o +#
//#        o  +           +        +             +                            #
//#o  +        o     o       +       o     o          o       +               #
//# -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_,------,      o      o    +#
//# -__-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-|    /\_/\                #
//# -_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_--|__( ^ .^)  +     +     o  #
//# -__-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-_-""  ""                    #
//#   +      o         o   +       o      +       o   +      o     +      +    #
//#o      +         +                                                           #
//#+  o        o         o      o     +  o      o         +      o     o      +#
```



```
//#+-     o           +                                o                  o ~+#
//#+-*+      +     o        o      +         +    o            +    o       *~+#
//#+-*§----------------------------------------------------------------------§*-+#

#include "main.h"
#include "fm25h20.h"
#include "cc430f6137.h"


void SPISetup(){

    UCB0CTL1 |= UCSWRST;
    UCB0CTL1 |= UCSSEL_2;
    UCB0CTL0 = UCMST + UCMODE_0 + UCMSB + UCSYNC + UCCKPH; //Master and select 4-pin SPI with STE
    active low
    UCB0BR0 = 1;
    UCB0BR1 = 0;
    UCB0STAT = 0;
    P1SEL |= BIT2 + BIT3 + BIT4 ;
        P1DIR |= BIT0 + BIT1 + BIT3 + BIT4 + BIT5 + BIT6 + BIT7;        //Set BIT6 to Vdd, and BIT7
        to GND, BIT0 is HOLD,
        P1OUT |= BIT0 + BIT1 + BIT5 + BIT6;
        P1OUT &= ~BIT7;
        P1DS |= BIT6 + BIT7;
    UCB0CTL1 &= ~UCSWRST;

    return;

}

void FM25_SendOpCode(uint8 byte){

        P1OUT &= ~BIT5;
    while (!(UCB0IFG&UCTXIFG));
    UCB0TXBUF = byte;
    while(UCB0STAT & UCBUSY);
    P1OUT |= BIT5;
    return;

}

uint8 FM25_ReadReg(uint8 byte){

        uint8 tmp = 0;

        P1OUT &= ~BIT5;
    while (!(UCB0IFG&UCTXIFG));
    UCB0TXBUF = byte;
        __delay_cycles(1);
    UCB0TXBUF = 0xaa;

    while(UCB0STAT & UCBUSY);

    tmp = UCB0RXBUF;
    P1OUT |= BIT5;
    return tmp;

}

void FM25_WriteToAddress(uint8 *data, uint16 length, uint32 addr){

        P1OUT &= ~BIT5;
```

```
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = FM_WRITE;
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = (addr >> 16);
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = (addr >> 8);
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = addr;
        while(length--){
                while (!(UCB0IFG&UCTXIFG));
                UCB0TXBUF = *data++;
        }
        while (!(UCB0IFG&UCTXIFG));
        P1OUT |= BIT5;
        return;
}

void FM25_ReadFromAddress(uint8 *data, uint16 length, uint32 addr){

        P1OUT &= ~BIT5;
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = FM_READ;
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = (addr >> 16);
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = (addr >> 8);
        while (!(UCB0IFG&UCTXIFG));
        UCB0TXBUF = addr;
        while(length--){
                while (!(UCB0IFG&UCTXIFG));
                UCB0TXBUF = 0x0;
                while(UCB0STAT & UCBUSY);
                *data++ = UCB0RXBUF;
        }
        P1OUT |= BIT5;
        return;
}
```

Bibliothek für Temperatursensor ADT7320

```
//#+-*§-------------------------------------------------------------------§*-+#
//#+-*  SPI Interface for ADT7320                                          *-+#
//#+-   Copyright © 2011 - 2014 Thorbjörn Jörger                          -+#
//#+                                                                       +#
//#+-   Lehrstuhl für elektrische Mess- und Prüfverfahren - IMTEK          -+#
//#+-*  Albert-Ludwigs-Universität Freiburg im Breisgau                   *-+#
//#+-*§-------------------------------------------------------------------§*-+#
//#+-*                          ‘. ___                                   *-+#
//#+-                         __,’ __‘.                 _..----....____   -+#
//#+              __...--.’’‘;.   ,.   ;‘‘--..__     .’    ,-._    _.-’   +#
//#        _..-’’--------’   ‘’   ‘’   ‘’     0 ‘‘-’’._   (,;’) _,’        #
//#       ,’-----------------                          \‘-._‘-’,’           #
//#        ‘._              ‘‘‘‘‘‘‘‘‘‘‘------...___   ’-..-’-:            #
//#           ‘‘‘--..._      ,.                     ‘‘‘‘--...__\-.          #
//#                  ‘.--. ‘-‘                        ____    |  |‘          #
//#                    ‘. ‘.                       ,’‘‘‘‘‘.  ;  ;‘          #
//#                      ‘._‘.        __________   ‘.      \’__/‘          #
//#                         ‘-:._____/______/___/____‘.     \  ‘           #
//#+                                |       ‘._    ‘.    \               +#
//#+-                               ‘.________‘-.   ‘.   ‘.___           -+#
//#+-*                                          SSt  ‘------’‘         *-+#
//#+-*§-------------------------------------------------------------------§*-+#
```



```
#include "SPIBitBang.h"

void CC430SPISetup()
{

  SPIBitBangPxDIR |= (SPIBitBangSync | SPIBitBangClock | SPIBitBangSIMO);
  //SPIBitBangPxDS |= (SPIBitBangSync | SPIBitBangClock | SPIBitBangSIMO | SPIBitBangSOMI);
  //SPIBitBangPxREN |= (SPIBitBangSync | SPIBitBangClock | SPIBitBangSIMO | SPIBitBangSOMI);
  SPIBitBangPxOUT |= SPIBitBangSync;
  SPIBitBangPxOUT |= SPIBitBangClock;

  return;

}

void ADT7320Setup()
{
        CC430SPIReset();
        while(CC430SPIReadRegister(ADT_REG_ID) != 0xc3)
                CC430SPIReset();
        while(CC430SPIReadRegister(ADT_REG_CONFIG) != 0x00)
                CC430SPIReset();
        while(CC430SPIReadRegister(ADT_REG_TCRIT) != 0x4980)
                CC430SPIReset();
        while(CC430SPIReadRegister(ADT_REG_THIGH) != 0x2000)
                CC430SPIReset();
        while(CC430SPIReadRegister(ADT_REG_TLOW) != 0x0500)
                CC430SPIReset();
        CC430SPIWriteRegister(ADT_REG_CONFIG, (ADT_OP_MODE_CONT + ADT_RESOLUTION_16));

        return;
}

unsigned int ADT7320Temp(){

        return CC430SPIReadRegister(ADT_REG_TEMP);

}

void CC430SPIReset()
{
        unsigned char x = 0;

        SPIBitBangPxOUT &= ~SPIBitBangSync;
        SPIBitBangPxOUT |= SPIBitBangSIMO;

        for(x=48; x>0; x--){

                SPIBitBangPxOUT &= ~SPIBitBangClock;
                __delay_cycles(6);

            SPIBitBangPxOUT |= SPIBitBangClock;
            __delay_cycles(6);

        }

        SPIBitBangPxOUT |= SPIBitBangSync;
        SPIBitBangPxOUT &= ~SPIBitBangSIMO;
}

void CC430SPIWriteByte(unsigned char value)
```

```
{

  unsigned char x;

  //SPIBitBangPxOUT &= ~SPIBitBangSync;

  for(x=8; x>0; x--){

        SPIBitBangPxOUT &= ~SPIBitBangClock;
       __delay_cycles(6);

    if(value & 0x80){
      SPIBitBangPxOUT |= SPIBitBangSIMO;
    }
    else{
      SPIBitBangPxOUT &= ~SPIBitBangSIMO;
    }

    value = value << 1;

    SPIBitBangPxOUT |= SPIBitBangClock;
    __delay_cycles(6);
  }

  //SPIBitBangPxOUT |= SPIBitBangSync;

  return;

}

unsigned char CC430SPIReadByte(void)
{

  unsigned char x = 0;
  unsigned char value = 0;

  //SPIBitBangPxOUT &= ~SPIBitBangSync;

  for(x=8; x>0; x--){

        SPIBitBangPxOUT &= ~SPIBitBangClock;
        __delay_cycles(6);

        value = value << 1;

    if(SPIBitBangPxIN & SPIBitBangSOMI){
      value |= 0x1;
    }
    else{
      value |= 0x0;
    }


    SPIBitBangPxOUT |= SPIBitBangClock;
    __delay_cycles(6);

  }

  //SPIBitBangPxOUT |= SPIBitBangSync;

  return value;
```



```
}

void CC430SPIWriteRegister(unsigned char reg, unsigned int data){

        unsigned char twobytes = 0;

        if(reg & 0x1)
                twobytes = 1;
            reg = reg & 0xfe;

        SPIBitBangPxOUT &= ~SPIBitBangSync;

        CC430SPIWriteByte(SPI_WREG | reg);

        if(twobytes){
                CC430SPIWriteByte(data >> 8);
        }

        CC430SPIWriteByte(data);

        SPIBitBangPxOUT |= SPIBitBangSync;

        return;
}

unsigned int CC430SPIReadRegister(unsigned char reg){

        unsigned int data = 0;
        unsigned char twobytes = 0;

        if(reg & 0x1)
                twobytes = 1;
        reg = reg & 0xfe;

        SPIBitBangPxOUT &= ~SPIBitBangSync;

        CC430SPIWriteByte(SPI_RREG | reg);
        //CC430SPIWriteByte(0x0);
        data = CC430SPIReadByte();

        if(twobytes){
                data = data << 8;
                data |= CC430SPIReadByte();
        }

        SPIBitBangPxOUT |= SPIBitBangSync;

        return data;
}
```

C Datenträger