Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterarbeit

Michel Rottleuthner

Eine Plattform für energieautarke Sensorknoten im Feldtest

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Michel Rottleuthner

Eine Plattform für energieautarke Sensorknoten im Feldtest

Masterarbeit eingereicht im Rahmen der Masterprüfung

im Studiengang Master of Science Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Thomas Schmidt
Zweitgutachter: Prof. Dr. Franz Korf

Eingereicht am: 4. Juni 2018

Michel Rottleuthner

Thema der Arbeit

Eine Plattform für energieautarke Sensorknoten im Feldtest

Stichworte

Sensorknoten, Energy-Harvesting, Feldtest, RIOT

Kurzzusammenfassung

In dieser Arbeit wird die Entwicklung einer modularen Plattform zur Durchführung von Feldtests und stationären Versuchen mit energieautarken, kabellosen Sensorknoten gezeigt. Gestützt durch Erkenntnisse aus verwandten Arbeiten werden ausgehend von einem abstrakten Konzept alle Schritte bis zum praktischen Einsatz der Plattform behandelt. Zu diesem Zweck werden flexible Hardware-Komponenten angefertigt, die ein Photovoltaik-basiertes Energy-Harvesting System bilden. Für die verwendeten Komponenten werden Treiber und eine Firmware auf Basis des Betriebssystems RIOT implementiert. Im Rahmen eines mehrwöchigen Feldtests im Freien wird demonstriert, wie mit mehreren dieser Knoten die Messung, lokale Speicherung und Funkübertragung von Umweltdaten und Metriken zur eigenen Energiebilanz umgesetzt wird. Anhand der gewonnenen Informationen wird illustriert wie das System zur detaillierten, praxisrelevanten Analyse energiebezogener Systemparameter eingesetzt werden kann, um bei der Entwicklung generischer Energieverwaltungsmechanismen zu unterstützen.

Michel Rottleuthner

Title of the paper

A Platform for Energy Self-sufficient Sensor Nodes in Field Tests

Keywords

Sensor Nodes, Energy-Harvesting, Fieldtest, RIOT

Abstract

In this paper the development of a modular platform for field tests and stationary experiments with energy self-sufficient, wireless sensor nodes is shown. Based on findings from related work, all steps from an abstract concept up to the practical use of the platform are covered. For this purpose, flexible hardware components are built that form a photovoltaic-based energy harvesting system. Drivers and firmware based on the RIOT operating system are implemented for the components used. An outdoor field test is performed over several weeks to demonstrate how multiple of these nodes can be used to measure, locally store and transmit environmental and measurement data about their own energy balance. The information obtained is used to illustrate how the system can be used for detailed, practically relevant analysis of energy-related system parameters to support the development of generic energy management mechanisms.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

1. Einleitung

Daten und darin enthaltene Informationen werden heute zu allen erdenklichen Themen automatisch generiert, erfasst, verarbeitet und gespeichert. Diese Prozesse konzentrierten sich zu Beginn der Digitalisierung hauptsächlich auf Einrichtungen wie Rechenzentren oder Forschungsinstitute. Heute werfen auch allgegenwärtige, kleine Geräte eigenständig Informationen ab, ohne Interaktionen durch den Menschen zu erfordern. Vernetzt man diese Geräte untereinander und ermöglicht ihnen über standardisierte Protokolle Informationen auszutauschen, so erhält man ein weltumspannendes Informationssystem. Werden die Daten mit einer Semantik versehen, können damit Analysen, Entscheidungsprozesse und andere Algorithmen automatisiert oder erst ermöglicht werden. In der Literatur wird dieses Gebilde unter anderem als Internet of Things bezeichnet wird. Das Internet of Things (IoT) kann damit als Metapher für eine intelligente, vollständig vernetzte Welt verstanden werden.

Jedes der allgegenwärtigen, vernetzten Geräte muss zum Betrieb mit Energie versorgt werden und bietet somit das Potenzial, dabei auf neue, effektive und nachhaltige Wege zur Energieversorgung zurückzugreifen. Die Bausteine des IoT bestehen in der Regel aus kleinen eingebetteten Systemen mit stark eingeschränkten Ressourcen. Dies betrifft nicht nur die Rechenleistung und Speicherkapazität, sondern auch die Energieverfügbarkeit. Energy-Harvesting (EH) bietet als Alternative zu endlichen Batteriekapazitäten oder einer externen Energieversorgung eine Lösung, mit der Sensorknoten dauerhaft energieautark operieren können. Energieautarke Sensorknoten bieten gegenüber konventionell batteriebetriebenen Systemen die Vorteile eines wartungsarmen Dauereinsatzes und erhöhter Verfügbarkeit von Energieressourcen. Energy-Harvesting Prozesse hängen prinzipbedingt stark von Umgebungsfaktoren ab und verhalten sich häufig sehr dynamisch. Um diese Einflüsse realistisch zu bewerten, sind sehr detaillierte Informationen über Umgebung und Systemverhalten erforderlich. Sind diese Faktoren nicht bekannt oder nicht ausreichend präzise bestimmt, dann lassen sich Simulationen oft nur mit unzureichender Genauigkeit ausführen[3]. Die simulierten Ergebnisse verlieren damit ihre Belastbarkeit zur Vorhersage des realen Systemverhaltens. Feldversuche und Messungen helfen in diesen Fällen dabei, Systemparameter genauer zu bestimmen, diese überhaupt erst zu identifizieren oder Abweichungen zur Simulation aufzuzeigen.

Neben den Ressourcen die ein IoT-Gerät zum Betrieb benötigt, existieren Anforderungen, die bei der Entwicklung von IoT-Anwendungen relevant sind. Neben den zuvor bereits genannten standardisierten Protokollen, sind einheitliche Software-Plattformen und Abstraktionsebenen ein wichtiger Bestandteil der Anwendungsentwicklung im IoT. Die Arbeitsgruppe INET[1] der HAW-Hamburg gestaltet durch die Mitentwicklung des IoT-Betriebssystems RIOT [4] die Zukunft des IoT aktiv mit und bietet damit ein passendes Umfeld für aktuelle Forschungsarbeiten in diesem Bereich.

Vor diesem Hintergrund wird in der vorliegenden Arbeit die Entwicklung einer modularen Plattform zur Durchführung von praxisnahen Feldtests mit energieautarken Sensorknoten gezeigt. Die Plattform wird konzipiert um bei Entwicklung, Test und Analyse von generischen Lösungen zur Energieverwaltung für das Betriebssystem RIOT zu unterstützen. Der Einsatz der Plattform soll dabei Möglichkeiten aufzeigen, wie RIOT zur Verwendung auf energieautarken Sensorknoten erweitert und optimiert werden kann.

Der anschließende Teil der Arbeit ist in 8 Abschnitte untergliedert. In Kapitel 2 wird zunächst ein Überblick zu den Themengebieten Internet of Things, Energy-Harvesting und RIOT gegeben. Dabei werden Forschungsthemen aufgezeigt und anhand verwandter Arbeiten der Kontext der Arbeit dargestellt. Die Problemstellung und der Ansatz werden in Kapitel 3 genauer definiert, woraufhin in Kapitel 4 die Konzeption und Implementierung benötigter Software- und Hardware-Komponenten gezeigt wird. Die Evaluierung vorab zu testender Komponenten wird in Kapitel 5 dargelegt, wonach in Kapitel 6 die Module in ein Gesamtsystem integriert werden. Anhand der vollständigen Plattform wird in Kapitel 7 der Aufbau und die Durchführung eines Feldtests gezeigt, mit dem der Einsatz des entwickelten Sensorsystems in einem realistischen Szenario demonstriert wird. Die Ergebnisse aus dem Feldtest werden in Kapitel 8 vorgestellt und diskutiert. Zum Ende wird in Kapitel 9 ein kurzer Ausblick auf weiterführende Arbeiten mit der entwickelten Plattform gegeben und mit einem Fazit abgeschlossen.

[1] https://inet.haw-hamburg.de/

2. Grundlagen

2.1. Internet of Things

Das Themengebiet des Internet of Things kann als eine Schnittmenge von Identifikation, Sensorik, Kommunikation und Datenverarbeitung verstanden werden. Reichert man diese Bestandteile mit einer Semantik an, lassen sich neue Dienste erschaffen, um automatisiert Daten zu verarbeiten, auszuwerten und möglichst intelligent Prozesse zu steuern [5]. Als Schlüsselkomponenten für ein derartiges kontextsensitives Computing, wird Hardware in Form von Sensorik und eingebetteten Kommunikationsfunktionen, eine geeignete Middleware mit Datenspeicherungs- und Analyseverfahren in Kombination mit leicht verständlichen Darstellungs- und Interpretationstools genannt [6]. Den elementaren Grundbaustein bilden allgegenwärtige Kommunikationsnetzwerke, welche kontextuelle Informationen dorthin übermitteln sollen, wo sie benötigt werden. Verschiedene Studien nennen Smart Cities als typisches Referenzszenario für IoT-basierte Entwicklungen, da diese eine umfassende Abbildung der erwarteten Technologieeinflüsse auf Bereiche wie Transport, Umwelt-Monitoring, Energie und Gesundheit darstellen [7].

Zur Kommunikation zwischen diesen Geräten werden Technologien wie IEEE 802.15.4 oder Bluetooth Low Energy eingesetzt. Mit Protokollen wie IPv6, 6LowPAN (IPv6 over Low power Wireless Personal Area Networks), RPL (Routing Protocol for Low Power and Lossy Networks) existieren bereits etablierte Standards um kleine eingebettete Systeme global zu vernetzen. Für den Informationsaustausch auf Applikationsebene stehen mit CoAP (Constrained Application Protocol), MQTT (Message Queue Telemetry Transport), XMPP (Extensible Messaging and Presence Protocol) und weiteren Protokollen, Lösungen für diverse Einsatzszenarien bereit.

Auf logischer Ebene wird z. B. versucht, vernetzten Geräten ein möglichst detailliertes Abbild ihrer Umgebung zu vermitteln. Dieser Kontext soll dazu dienen, den Nutzern die Interpretation abzunehmen, um daraus notwendige Handlungen abzuleiten und im Anschluss ggf. deren Ausführung anzuordnen. Um Geräte und Dienste in diesem Maße zu vernetzen sind Technologien nötig, welche die Kombination von Informationen aus heterogenen Quellen ermöglichen. Dazu sind Softwarelösungen erforderlich die hochdynamisch, skalierbar, sicher

und zuverlässig sind. Durch die umfassendere Vernetzung aller Lebensbereiche und den damit verknüpften Nutzerdaten, wird besonders der weiteren Entwicklung im Bereich der Sicherheit und Privatsphäre eine erhebliche Bedeutung zugemessen [8]. Dem gegenüber steht Hardware die mit jedem Entwicklungsschritt kleiner, energiesparender und kostengünstiger werden soll, um den Anforderungen an ein ubiquitäres IoT näher zu kommen.

Während die notwendige Infrastruktur zum Aufbau von IoT-Netzwerken in bewohnten Gebieten stetig ausgebaut wird oder bereits ausreichend zur Verfügung steht, sind in größerer Entfernung zu Wohnhäusern und Straßen meist weder eine Anbindung an Kommunikationsnetze noch an ein Energienetz vorhanden. Im nachfolgenden Abschnitt wird daher ein Verfahrensmodell beschrieben, mit dem sich IoT-Geräte unabhängig von einem Energienetz realisieren lassen.

2.2. Energy-Harvesting

Nachfolgend werden für diese Arbeit relevante Teile aus dem Forschungsgebiet um Energy-Harvesting beschrieben. Zunächst wird dafür ein das Themenfeld grob umrissen um im Anschluss einzelne praktische Beispiele genauer zu behandeln. Energy-Harvesting (EH) beschreibt einen Prozess, bei dem Energie aus der Umgebung „geerntet“und angesammelt wird. Je nach Anwendung und Umfeld kann dafür auf verschiedene Energiequellen mit sehr unterschiedlicher Leistungsfähigkeit zurückgegriffen werden. Es stellt eine praktikable Lösung zur unabhängigen Energieversorgung von kleinen eingebetteten Systemen wie kabellosen Sensorknoten dar, wodurch ein wartungsarmer Dauereinsatz ohne Batteriewechsel erreichbar ist [1]. Diese Aufwandsersparnis ermöglicht Geräte und Einsatzszenarien, die andernfalls nicht wirtschaftlich sind. Durch das kontinuierliche Sammeln von Energie kann außerdem der Kompromiss zwischen Lebensdauer, Kosten, Zuverlässigkeit und der Performanz, verglichen mit „endlichen“Batteriekapazitäten, deutlich optimiert werden [9].

Abbildung 2.1 zeigt ein allgemeines Modell eines EH-Systems. Außerhalb eines eingebetteten Systems werden Energiequellen dargestellt, die in der Systemumgebung vorkommen. Die Energiequellen können von verschiedenen Energiewandlern innerhalb des Systems verwendet werden um nutzbare, elektrische Energie zu erzeugen. Die erzeugte Energie wird vom Speichersystem gesammelt, welches damit wiederum Sensoren, einen Mikrocontroller und ein Übertragungsmodul versorgt.

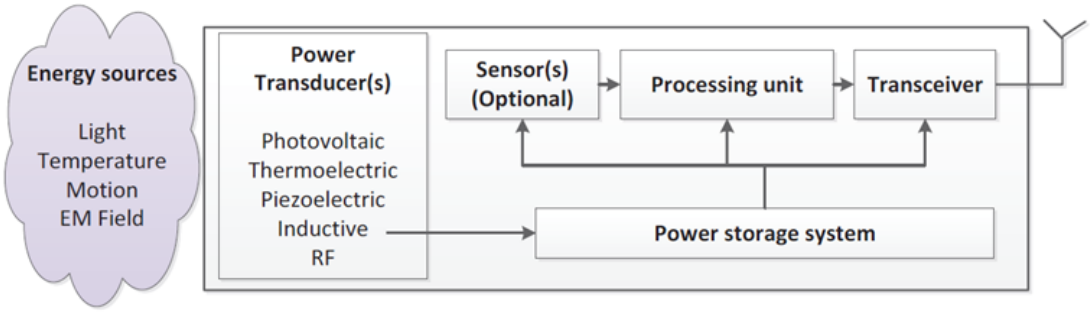


Abbildung 2.1.: Modell eines Energy-Harvesting Systems [1]

Thematisch lässt sich das Gebiet Energy-Harvesting in die Teilaspekte der Gewinnung, Speicherung und dem Management von Energie aufteilen. Ein Überblick über diese Themen wird in den folgenden Abschnitten dargestellt.

2.2.1. Energiegewinnung

Die Energiegewinnung in EH-Systemen wird mit Energiewandlern für eine bestimmte Energiequelle umgesetzt, welche entsprechend den Anforderungen des jeweiligen Anwendungsfalls ausgewählt werden. Hierzu zählen z. B. Parameter wie Standort, Größe, Platz oder Leistung. Für Energy-Harvesting lassen sich gängige Energiequellen wie beispielsweise Sonnen-, Wind- und Wasserkraft nutzen [10]. Auch alternative Ansätze wie Thermoelektrizität, menschliche Bewegungen oder Körperwärme [11, 12], Vibrationen oder Funkwellen [13] finden Verwendung. Für stationäre Systeme im Freien können dementsprechend Photovoltaik (PV) Module, Windgeneratoren, oder Wasserräder als Energiewandler verwendet werden, während am Körper getragene EH-Systeme meist auf kinetische Generatoren oder Peltier-Elemente zur Nutzung von Körperwärme zurückgreifen. Die Menge an verfügbarer Energie variiert zwischen den verschiedenen Quellen stark. Eine PV-Zelle liefert z. B. eine um den Faktor 300 größere Energiemenge verglichen mit einem thermoelektrischen Element gleicher Größe, welches die Energie aus dem Temperaturunterschied zwischen Körper und Umgebung schöpft [1]. Wird darüber hinaus auch die praktikable Größe der Elemente am Körper betrachtet, ergeben sich damit weitere Einschränkungen. In der Arbeit von Sudevalayam und Kulkarni [9] wird ein Überblick zu verschiedenen EH-Systemen und deren Konzepten gegeben. Als bevorzugte Energiequelle wird die Sonne unter Zuhilfenahme von Photovoltaik genannt, da diese einfach verfügbar, kostengünstig und problemlos skalierbar ist. PV-Systeme haben sich durch ihre problemlose Skalierbarkeit sowohl in leistungsfähigen Kraftwerken als auch in Kleinstanwendungen wie Taschenrechnern etabliert. Das Verhalten der Sonne als Energiequelle ist durch die Periodizität

zwar vorhersagbar, unterliegt durch Wetterbedingungen wie Bewölkung oder Niederschlag aber typischen Leistungsschwankungen von EH-Systemen.

2.2.2. Energiespeicherung

Eine häufige Problemstellung von EH-Systemen ist, dass die zur Verfügung stehende Energie aus dem Harvesting-Prozess zu gering für den Dauerbetrieb ist, oder nicht ohne Unterbrechungen zur Verfügung steht. Zur Speicherung der Energie in EH-Systemen bieten sich verschiedene Verfahren an, welche sich hauptsächlich im Speichermedium und dessen inhärenten Eigenschaften unterscheiden. Die in portablen Geräten häufig eingesetzten chemischen Akkus (z. B. Lithium-Ionen-Akkumulator) eignen sich durch ihre auf 500 bis 1200 begrenzten Ladezyklen nur bedingt für einen autarken Langzeiteinsatz im Bereich von mehreren Jahren [14]. Allerdings bieten nur derartige Akkus eine hohe Energiedichte, welche für die Überbrückung längerer Perioden ohne Energiezufuhr zwingend notwendig ist. Mehrere Forschungsarbeiten haben gezeigt, dass sich für sparsame Systeme auch Kondensatoren bzw. Superkondensatoren einsetzen lassen [15, 2]. Zwar ist deren Energiedichte ca. um einen Faktor von zehn niedriger, die Lebensdauer bezüglich Ladezyklen liegt allerdings mindestens um einen Faktor von 1000 höher [14]. Ein sich abzeichnender Trend ist die Verwendung hybrider Energiespeichersysteme die aus Akkus und Superkondensatoren bestehen und über eine integrierte Schaltung überwacht und gesteuert werden [2]. Abbildung 2.2 zeigt ein solches System. Die DC-DC Wandler sind dabei notwendig um die Spannung der jeweils angeschlossenen Komponente auf das vom Mikrocontroller benötigte Niveau zu regeln. Bei der Verwendung von Superkondensatoren ist eine Regelung der Spannung unumgänglich, da deren Spannungspegel durch die Entnahme von Energie stetig weiter abfällt. Dieser Effekt wirkt sich bei Akkus, die innerhalb ihrer Spezifikationen betrieben werden, wesentlich geringer aus.

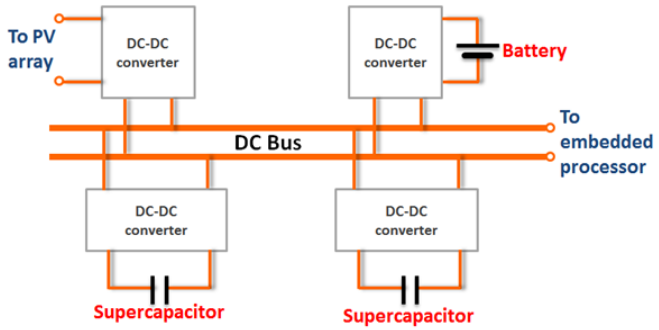


Abbildung 2.2.: Hybrides Energiespeichersystem mit zwei Superkondensatoren und einem chemischen Akku [2]

Aus einem Vergleich der gängigen Energiespeicher wie SLA (Sealed Lead Acid), NiCd (Nickel-Cadmium), NiMH (Nickel-Metallhydrid), Li-Ion (Lithium-Ionen), LiPO (Lithium-Polymer) und Superkondensatoren lässt sich zusammenfassend ableiten, dass Lithium Akkus verglichen mit anderen chemischen Akkus für den Einsatz in kabellosen Sensorknoten vorzuziehen sind. Dies geht allerdings mit einem komplexeren Ladeverfahren einher. Superkondensatoren bieten neben ihrer weitaus höheren Lebensdauer gegenüber Akkus die Vorteile eines einfachen Ladeverfahrens und einer hohen physikalischen Robustheit. Reicht die Energiedichte für den geplanten Anwendungsfall aus, so überwiegen die Vorteile des Superkondensators. Bezüglich des Ladereglers werden Lösungen in Hardware einer Softwaresteuerung bevorzugt, um im Falle einer vollständigen Entladung sicherzustellen, dass das System wieder anläuft [9].

2.2.3. Energiemanagement

Trotz erheblicher Unterschiede zwischen den Eigenschaften der Energiequellen werden auf Software-Ebene des eingebetteten Systems allgemeine Energiesparmechanismen und eine möglichst intelligente Energieverwaltung benötigt, um die Leistungsfähigkeit der Anwendung im Rahmen der Energieverfügbarkeit zu maximieren.

Dadurch entsteht ein Bedarf an Energieverwaltungsmechanismen, welche unter Anwendung von Optimierungsstrategien versuchen, die zur Verfügung stehende Energiemenge möglichst sinnvoll zu Nutzen. Ist es unter ordnungsgemäßem Betrieb des Systems möglich, höchstens die Menge an Energie zu verbrauchen, die vom System selbst gesammelt und umgewandelt werden kann, spricht man von einem Energy Neutral Operation (ENO) Zustand [16].

EH-Anwendungen benötigen häufig keinen permanenten Betrieb der Hardware und nur kurzzeitige Funkverbindungen. Dabei ist nur ein Bruchteil der Energie verglichen mit dem Dauerbetrieb notwendig und die energieerzeugenden sowie -speichernden Elemente können relativ klein und damit kostengünstig dimensioniert werden.

Um die verfügbare Energie gleichmäßig zu verbrauchen, werden häufig *Duty-Cycling*-Verfahren genutzt, in denen das System überwiegend in einem möglichst sparsamen Modus oder vollständig abgeschaltet verharrt. Zum Verrichten der notwendigen Arbeit wird das System periodisch für kurze zeit aufgeweckt. Der Vorteil liegt hierbei in der einfachen Umsetzung durch integrierte Timer-Peripherie eingebetteter Prozessoren und der flexiblen Möglichkeit zur Anpassung der Parameter für das an/aus Verhältnis und der Aufwachfrequenz. Dieser primitive Ansatz lässt sich durch *Power Gating* auf System- oder Chipebene, *Dynamic Voltage and Frequency Scaling* und durch Algorithmen gestütztes, dynamisches Scheduling und adaptiven Übertragungsstrategien weiter optimieren [1]. Der Einsatz von prädikativen Systemen kann

durch eine Vorhersage des zu erwartenden Energiebudgets weitere Vorteile mit sich bringen [17].

Da Untersuchungen an verteilten kabellosen Sensornetzen oft eine große Anzahl an Knoten benötigen, hat sich für Versuchs- und Forschungszwecke der Einsatz von Testbeds wie dem FIT IoT-LAB [18] bewährt. Ein Themengebiet das bei derartigen Testbeds bisher wenig untersucht wird, sind Energy-Harvesting Systeme [19].

2.2.4. Praktische Anwendungen

Forschungsarbeiten zu EH-Systemen beschäftigen sich z. B. mit Performanz-Adaptionen [9] und Vorhersagen zur zukünftigen Energieverfügbarkeit [10]. Adaptionen für EH-Systeme können in die Felder *Node-level Adaptions* und *Network-level Design* eingeteilt werden. *Node-level Adaptions* beeinflussen dabei z. B. Duty-Cycle, Sendeleistung, Zuverlässigkeit der Messung oder das Scheduling von Übertragungen. Zu *Network-level Design* zählen z. B. Routing, Clustering, Data Collection, Knotenredundanz oder MAC-Layer Optimierungen. Der Arbeit von Sudevalayam und Kulkarni kann entnommen werden, dass EH-Systeme unter Anwendung der genannten Optimierungen das Potenzial bieten, die beiden bei kabellosen Sensorknoten in Konflikt stehenden Designziele *Lebensdauer* und *Performanz* zu erhöhen.

Shaikh und Zeadally stellen in ihrer Arbeit [10] verschiedene EH-Konzepte gegenüber und gehen dabei detailliert auf existierende Verfahren zur Vorhersage der Energieverfügbarkeit ein. Sie kritisieren insbesondere die bisher hohe Fehleranfälligkeit der Vorhersagen. Ein Vergleich verschiedener Energiequellen zeigt auch hier die Vorzüge von Solarenergie bzw. PV-Zellen. Abgesehen von der Verwendung im Freien, verweisen die Autoren auch auf die Möglichkeiten PV-Zellen im Innenbereich zu verwenden, wenn dieser gut ausgeleuchtet ist. Als Herausforderungen für zukünftige Forschungsarbeiten nennen sie generische Harvester, Miniaturisierung, Protokoll-Adaptionen, effiziente und weniger fehleranfällige Vorhersagen, die Entwicklung von Simulationsumgebungen und die weitere Untersuchung von Energiespeichern und der Zuverlässigkeit von EH-Systemen. Als langfristiges Ziel für EH-Sensornetze sehen die Autoren einen Paradigmenwechsel von Batteriebetrieb hin zu autarken Lösungen welche ihre Energie ausschließlich aus ihrer Umwelt extrahieren.

Movers and Shakers

Im Rahmen der Arbeit von Gorlatova et al. [12] wird die potenzielle Leistungsfähigkeit von Mikrogeneratoren untersucht, welche durch Energie aus menschlichen Bewegungen angetrieben werden. Als Energiequelle dienen dabei alltägliche Aktivitäten, bei denen ein Gerät am

Körper getragen wird. Als Ausgangspunkt für Ihre Untersuchungen verwenden die Autoren einen umfangreichen Datensatz, der vorangegangenen Arbeit zur Gestenerkennung mittels Beschleunigungssensoren entstammt. Die Datensätze repräsentieren hierbei dreidimensionale Beschleunigungswerte verschiedener Bewegungen, von über 40 Testpersonen. Zusätzlich werden mit fünf weiteren Probanden Messungen über mehrere Tage durchgeführt. Statt bestimmten Gesten, stehen dabei allerdings die typischerweise auftretenden Bewegungsabläufe an regulären Tagen der Probanden im Fokus. Mit den Datensätzen wird anschließend über ein Modell eines Massenträgheitsgenerators simuliert, welche Energiemengen sich durch die Bewegungen in elektrische Energie umwandeln lassen. Die erhobenen Daten zeigen, dass sich mit einem kleinen Taschengenerator eine Energieversorgung realisieren lässt, die durchschnittlich zwischen $7\,\mu W$ und etwas unter $30\,\mu W$ zur Verfügung stellen kann. Aus den Ergebnissen lässt sich außerdem ableiten, wie sich die Parameter derartiger Generatoren anhand von Gewicht und Größe des Nutzers optimieren lassen. Die aus alltäglichen Bewegungen verfügbare Energie ist damit ausreichend um sehr sparsame eingebettete Systeme zu versorgen.

In-shoe Harvester

Eine weitere Arbeit, welche ebenfalls auf kinetische Energie aus menschlichen Bewegungen setzt, stellen Xie und Cai vor [11]. Die Arbeit zeigt die Leistungsfähigkeit eines EH-Systems, das mit Abmessungen von $80\,mm \times 47\,mm \times 22\,mm$ in einen Schuh integriert wird und die kinetische Energie beim Auftreten in elektrische Energie umwandelt. Hervorzuheben ist hierbei, dass keine piezoelektrischen Elemente, sondern eine kompakt realisierte mechanische Konstruktion verwendet wird. Dadurch kann mithilfe einer Übersetzung eine Rotation an einem Mikrogenerator ausgelöst werden, der mit $1\,W$ bei einer Laufgeschwindigkeit von $3{,}5\,km/h$ wesentlich mehr Energie erzeugt als bisherige Konstruktionen auf Piezo-Basis.

Remote Sensing of Wind-Driven Wildfire

Tan und Panda zeigen in ihrer Arbeit [15] die Entwicklung eines typischen Knotens für ein verteiltes Sensornetzwerk. Viele dieser Knoten sollen dabei in einem Überwachungssystem für Waldbrände über große Flächen, an verschiedenen Orten im Wald angebracht werden um die Windrichtung und Geschwindigkeit zu Messen. Die erhobenen Daten sollen in Echtzeit Aussagen zur voraussichtlichen Waldbrandgefahr und dessen Ausbreitung zulassen. Der Knoten verfügt über einen kleinen Windgenerator der bei einer Windgeschwindigkeit von $3{,}6\,m/s$ eine Leistung von $7{,}7\,mW$ erzeugt. Das Speichersystem basiert auf einem Superkondensator und Ladeelektronik. Die Messung, Berechnungen und die Funkübertragung werden von einem kleinen eingebetteten System inklusive Funkmodul erledigt, das im *Industrial, Scientific and*

Medical (ISM) Band operiert und für den Betrieb durchschnittlich $3{,}5\,\mathrm{mW}$ Leistung benötigt. Hervorsticht, dass hierbei der Windgenerator sowohl als Messinstrument als auch zur Energieversorgung des Systems verwendet wird.

Project Loon

Das ursprünglich von Google initiierte *Project Loon* [20] zeigt, dass sowohl ein kommerzielles Interesse an autarker Netzwerkinfrastruktur besteht, als auch die praktische Umsetzbarkeit im großen Maßstab. Ziel des Projekts ist, über autonom navigierte Wetterballons flächendeckend Internetzugänge über das 4G-Funknetz bereitstellen. Die Ballons fliegen dabei in der Stratosphäre auf einer Höhe von ca. $20\,\mathrm{km}$ und nutzen die mit der Höhe wechselnden Windrichtungen zur kontrollierten Fortbewegung. Ein Ballon deckt eine Kreisfläche mit einem Durchmesser von ca. $40\,\mathrm{km}$ auf dem Boden ab. Zur Energieversorgung werden die Ballons mit PV-Modulen ausgestattet, die eine Leistung von $100\,\mathrm{W}$ bereitstellen. Als Energiespeicher dienen Lithium-Akkus. Die Ballons sollen automatisch von Bodenstationen gestartet werden und bis zu 100 Tage in der Luft bleiben. Im Rahmen des Projekts wurden bereits über 25 Millionen km Testflüge absolviert. Die Ballons funktionieren als Relay-Stationen für Bodenstationen anderer Telekommunikationsfirmen und können pro Hop ca. eine Strecke von $100\,\mathrm{km}$ überbrücken. Auf das dadurch aufgespannte Netz kann mittels handelsüblicher LTE Endgeräte zugegriffen werden, was mit einer Bandbreite von bis zu $10\,(\mathrm{Mbit})/\mathrm{s}$ möglich sein soll.

2.3. RIOT

RIOT [4] ist ein offenes IoT-Betriebssystem für Ressourcenarme eingebettete Systeme. Es unterstützt verschiedene Architekturen wie z. B. AVR, MSP430, MIPS, ARM7 und ARM Cortex-M und bietet Entwicklern die Möglichkeit RIOT-basierten Code ohne Anpassungen auf allen unterstützen Plattformen einzusetzen. RIOT stellt dazu eine umfangreiche Hardwareabstraktionsschicht bereit und erleichtert dem Entwickler durch bekannte Programmierparadigmen und Standardschnittstellen, modulare Anwendungen zu erstellen. Bestandteil des Betriebssystems sind außerdem ein eigener Netzwerkstack mit Unterstützung von 6LowPAN, Implementierungen von UDP, TCP und diversen ioT-Protokollen auf Applikationsebene wie CoAP oder MQTT. Zur Ansteuerung von Prozessorperipherie, Sensorik und Aktorik sind zusätzliche Treiber vorhanden. Die aktuelle Version von RIOT[1] unterstützt dabei über 100 verschiedene Zielplattformen. Um auch Komponenten auf Basis von fremdem Quellcode nahtlos in den Build-Prozess von RIOT zu integrieren, existiert ein Package-Konzept. Ein Package enthält

[1] RIOT Release 2018.01 https://github.com/RIOT-OS/RIOT/tree/2018.01-branch

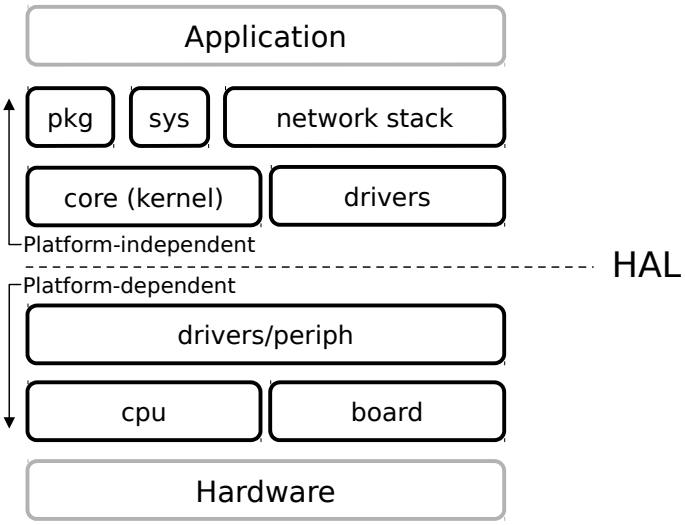


Abbildung 2.3.: RIOT Architekturübersicht

dabei alle notwendigen Ausführungsschritte und ggf. Patches, um den Quellcode in einem automatisierten Ablauf kompatibel zum RIOT Build-Prozess zu machen. In der Regel umfassen die Schritte das Beziehen des Quellcodes von seiner Originalquelle, den Code im Anschluss in einer geeigneten Struktur im Quellcodeverzeichnis von RIOT abzulegen und evtl. benötigte Patches anzuwenden. Abbildung 2.3 zeigt eine Übersicht der Komponenten von RIOT. Auf niedrigster Ebene setzt das Betriebssystem mit Treibern für CPU und Boards direkt über der Hardware an. Darüber liegen Peripherietreiber, welche den stärker abstrahierten Modulen oberhalb des HAL (Hardware Abstraction Layer) Schnittstellen bereitstellen um Peripherie wie z. B. SPI-Bausteine (Serial Peripheral Interface) anzusteuern. Der Kernel, Treiber für externe Geräte oder der Netzwerkstack greifen nur über diese Abstraktion auf die Hardware zu und stellen in Richtung der Applikation eigene Schnittstellen bereit.

3. Problemstellung und Ansatz

Das Entwickeln von energiesparenden IoT-Geräten kann durch das verwendete Betriebssystem mithilfe generischer Lösungen zur Energieverwaltung vereinfacht werden, um auch in diesem Bereich von einer stärkeren Abstraktion zur Hardware zu profitieren. Energieverwaltungsmechanismen auf Betriebssystemebene zu implementieren, welche gleichzeitig effektiv, effizient und zuverlässig sind, ist allerdings kein triviales Problem. Dieser Sachverhalt wird im folgenden genauer dargelegt und untermauert. Eine Schwierigkeit bei der Implementierung generischer Energieverwaltungsmechanismen auf Ebene des Betriebssystems besteht in den stark variierenden Konfigurationsmöglichkeiten der verschiedenen Zielplattformen. Beispiele hierfür sind verschiedene Low-Power Modi, MCU-interne Taktkonfigurationen oder Teilabschaltungen von internen und externen Peripheriebausteinen. Weiterhin entstehen zwischen diesen Ressourcen dynamische Abhängigkeiten, die sich zur Laufzeit verändern. Zusätzlich zur großen Anzahl an Plattformen entstehen durch die kontinuierliche Weiterentwicklung Herausforderungen bezüglich Testmethoden und Verifikation der Energieverwaltung. RIOT wird von einer aktiven Community entwickelt und regelmäßig um neue Funktionen erweitert. Durch regelmäßige Änderungen sind automatisierte Tests zur Qualität und Funktionalität des Quellcodes unverzichtbar. Jede kleine Änderung manuell auf allen relevanten Plattformen zu testen, wäre aufgrund der Anzahl an Plattformen und Änderungen mit moderatem Aufwand nicht realisierbar. Ein Continuous Integration System (CI) schafft hierbei Abhilfe. Fehler im Code können damit erkannt und durch den Entwickler behoben werden, bevor sie in die Code-Basis übernommen, oder gar auf einem Produktivsystem eingesetzt werden. Der im IoT-Umfeld besonders wichtige Aspekt des Energieverbrauchs wird dabei häufig nicht ausreichend abgedeckt. Das CI System von RIOT bietet derzeit keine Möglichkeit, automatisiert die Auswirkungen von Software-Änderungen auf den Energieverbrauch zu Messen, um bereits während des Entwicklungsprozesses vor starken Abweichungen zu warnen oder Energiebezogene Optimierungen zu quantifizieren. Dass sich Messungen zum Energieverbrauch als Test formulieren und in ein CI-System integrieren lassen, wurde bereits gezeigt [21]. Während der Aufbau von dedizierten Messplätzen mit automatisierbaren Messgeräten technisch möglich ist, ergeben sich dabei bereits durch die Anzahl an zu testenden Geräten erhebliche Schwierig-

keiten bezüglich Logistik und Wirtschaftlichkeit. Ein wesentlicher Faktor besteht außerdem darin, dass die Überprüfung auf korrektes Systemverhalten in Laborversuchen häufig nicht ausreicht, um eine ordnungsgemäße Funktion unter Realbedingungen sicherzustellen. Dies gilt insbesondere für verteilte und dynamische Systeme, deren Verhalten von Umgebungsfaktoren oder anderen Systemen abhängt.

Vor diesem Hintergrund soll im Rahmen dieser Arbeit eine Plattform aufgebaut werden, die bei der Entwicklung und dem Testen von energieautarken IoT-Systemen unterstützen soll. Mithilfe der Plattform sollen dafür Messreihen zu Energieverbrauchsmetriken gemessen und aufgezeichnet werden. Diese Messwerte sollen dann beispielsweise zur Optimierung, Bewertung und Vergleich von Scheduling-, Energiemanagement- oder Übertragungsverfahren und zur weiteren Verwendung im Rahmen von Simulationen dienen. Der Aufbau ist weitestgehend modular und flexibel zu halten um die Konfigurationsfreiheit und Verwendungszwecke für weitere Forschungsarbeiten wenig einzuschränken.

Ein wesentliches Ziel ist es, die Plattform so zu konzipieren, dass damit sowohl stationäre Laborversuche, als auch realistische Feldtests durchführbar sind. Dadurch soll erreicht werden, Daten aus möglichst praxisnahen Tests zu gewinnen um z. B. die Auswirkungen von Umgebungs- und Störfaktoren wie Interferenzen, Temperaturschwankungen oder Bewegungen direkt am Einsatzort zu berücksichtigen.

Die Eignung bzw. das Optimierungspotenzial von RIOT als Betriebssystem für energieautarke Sensorknoten soll in diesem Zuge ebenfalls beleuchtet werden. Schwachpunkte in vorhandenen Schnittstellen und fehlende Komponenten sollen dabei aufgezeigt und gegebenenfalls implementiert werden.

4. Konzeption und Implementierung der Einzelkomponenten

Als dynamisches Untersuchungsobjekt soll Hard- und Software für ein Energy-Harvesting System entwickelt werden, das die Realisierbarkeit von verteilten Feldtests mit der Plattform praktisch demonstriert. Messabläufe und deren Aufzeichnung sollen sowohl an dem zu untersuchenden Mikrocontroller selbst, als auch in Form eines eigenständigen Systems einsetzbar sein. Je nach Anforderungen des untersuchten Szenarios erhält der Entwickler dadurch die Wahlmöglichkeit zwischen einer integrierten Selbstüberwachung und einer weniger invasiven Fremdüberwachung. Die Monitoring Komponente soll sowohl den Verbrauch, als auch die Einspeisung von Energie messen. Weiterhin soll eine Möglichkeit zur Aufzeichnung von Leistungsgraphen während einzelner Programmabschnitte berücksichtigt werden.

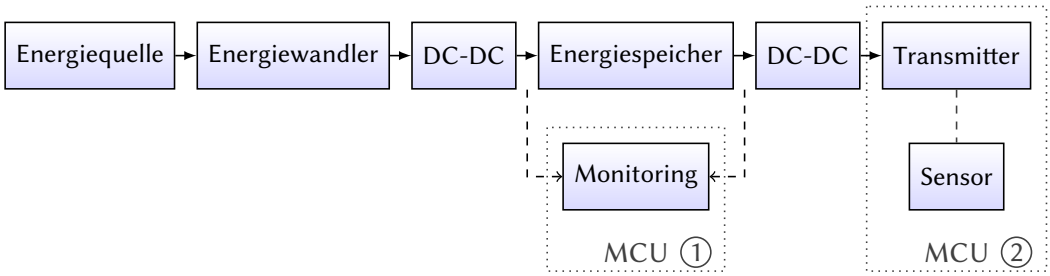


Abbildung 4.1.: Schematischer Aufbau eines EH-Systems mit Monitoring Komponente

Zur Veranschaulichung zeigt Abbildung 4.1 den schematischen Aufbau der geplanten Plattform in Form eines kabellosen Sensorknotens. Die von der Energiequelle bereitgestellte Energie wird dabei von einem Energiewandler in Elektrizität umgewandelt. Diese elektrische Energie wird mit einem Laderegler (DC-DC-Wandler (**D**irect **C**urrent)) im Energiespeicher angesammelt. Mit der gepufferten Energie kann anschließend über einen weiteren DC-DC-Wandler der Mikrocontroller mit Übertragungsmodul und Sensoren betrieben werden. Die Gleichstrom-Wandler sind hierbei nötig um das Spannungsniveau zwischen den jeweiligen Komponenten anzupassen. Einen grundlegenden unterschied zu einem einfachen EH-System stellt hier die

zusätzliche Monitoring-Komponente dar, welche zur Aufzeichnung der relevanten Systemparameter und des allgemeinen Systemverhaltens eingesetzt wird.

Der folgende Teil dieses Kapitels gliedert sich in Abschnitte, welche die Entwicklung der einzelnen Komponenten darlegen. Anhand des abstrakten Konzepts und dem zuvor erläuterten Kontext werden dazu jeweils die Anforderungen abgesteckt und Lösungen zur Implementierung ausgewählt und umgesetzt.

4.1. Energiequelle: Photovoltaik

Als Energiequelle für das EH-System wird aufgrund der in Unterabschnitt 2.2.1 und Unterabschnitt 2.2.4 dargelegten Vorzüge, die Sonne unter Einsatz eines PV-Moduls verwendet. PV-Zellen besitzen die Eigenschaft, dass die von ihnen abgegebene elektrische Leistung stark einbricht, wenn sie entweder zu stark oder zu schwach belastet werden.

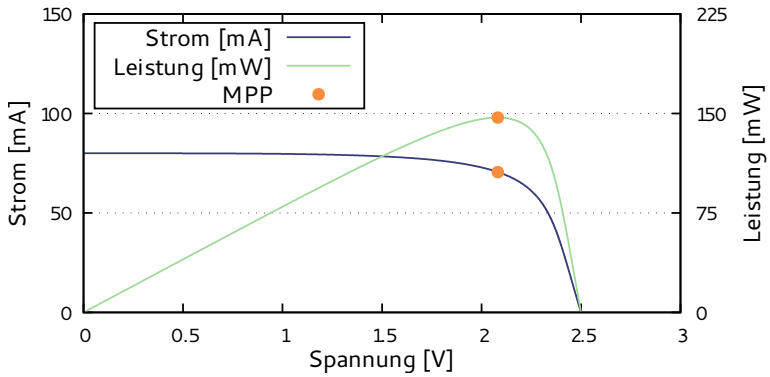


Abbildung 4.2.: Strom-/Spannungskennlinie und Leistungs-/Spannungskennlinie eines PV-Moduls mit markiertem MPP

Ein Verfahren namens Maximum Power Point Tracking (MPPT) kann dazu eingesetzt werden, die energieerzeugenden Elemente zu jeder Zeit an ihrem Leistungsmaximum zu betreiben. Abbildung 4.2 zeigt diesen Sachverhalt am Beispiel eines PV-Moduls. Die Leistungscharakteristik des PV-Moduls wird dabei durch eine Strom-/Spannungskennlinie beschrieben. Eine Kurve bildet den Strom in Abhängigkeit der Spannung ab, die andere zeigt die daraus resultierende Leistungs-/Spannungskennlinie. Diese trägt auf der Y-Achse das Produkt von Strom und Spannung, was wiederum der Leistung entspricht. Die Effizienz und damit auch direkt die Leistungsfähigkeit wird somit durch die Spannung bestimmt, bei welcher der Strom am PV-Modul entnommen wird. Durch die Veränderung der Strom/Spannungskennlinie, beispielsweise durch den Einfluss von Wolkenfeldern auf die Sonneneinstrahlung, entsteht ein sich

stetig veränderndes System. In diesem wird durch eine geeignete Regelung versucht immer den aktuellen MPP zu erreichen, der in beiden Graphen mit einem Punkt markiert ist. Für ein EH-System kann es dabei von Vorteil sein die aktuellen Parameter der Energiequelle zur Laufzeit zu überwachen, um mit den zusätzlich zur Verfügung stehenden Informationen das Systemverhalten hinsichtlich des Energieverbrauchs anzupassen [22].

Es wird ein polykristallines PV-Modul mit Abmessungen von $69\,\mathrm{mm} \times 110\,\mathrm{mm}$ ausgewählt. Dieses ist auf der Oberseite in Epoxidharz eingegossen und weist laut Herstellerangaben eine maximale Leistungsfähigkeit von $1{,}2\,\mathrm{W}$ bei einer Nennspannung von $5\,\mathrm{V}$ auf.

4.2. Energiespeicher: Superkondensator

Die in Unterabschnitt 2.2.2 behandelte Literatur zeigt, dass Superkondensatoren besonders für Langzeiteinsätze viele Vorteile gegenüber gängigen Akkus bieten, wodurch längerfristige Einsätze im Bereich von Jahrzehnten realisierbar sind [9]. Es ist jedoch sicherzustellen, dass die ebenfalls vorhandenen Nachteile für den gewählten Anwendungsfall unkritisch sind. Ein großer Nachteil von Superkondensatoren sind auftretende Leckströme, die zu einer vergleichsweise raschen Selbstentladung führen und zur genauen Bestimmung des Verhaltens komplexe Modelle erfordern [23]. Der Ladezustand lässt sich bei typischen Anwendungen von Sensorknoten allerdings auch mit einfacheren Modellen meist ausreichend genau bestimmen [24].

Für den Einsatz in der Testplattform wird ein GreenCap des Herstellers Samwha mit einer Kapazität von $100\,\mathrm{F}$ und $2{,}7\,\mathrm{V}$ Nennspannung ausgewählt. Abbildung 4.4 zeigt den Superkondensator. Nachfolgend werden die grundlegenden Eigenschaften des Superkondensators und dadurch entstehende Implikationen beschrieben.

Anhand der Kapazität lässt sich mit Gleichung 4.1 die maximale Energiemenge von bis zu $364{,}5\,\mathrm{W\,s}$ bestimmen, die der Superkondensator speichern kann. Das Datenblatt [25] des Herstellers macht keine präzisen Angaben zur Höhe des Leckstroms. Lediglich ein Maximum von $0{,}27\,\mathrm{mA}$ nach 72 Stunden ist aufgeführt. Um sicher zu stellen, dass der Leckstrom innerhalb von 24 Stunden nach dem Aufladen nicht zu hoch für den Anwendungsfall ist, wird vorab eine Messung zur Eignungsprüfung des Bauteils durchgeführt. Dazu werden zwei Kondensatoren zunächst vollständig entladen, indem die Anschlusskontakte für 48 Stunden kurzgeschlossen werden. Dies stellt sicher, dass interne Ladungsumverteilungen hinreichend abgeklungen sind [26]. Zu Beginn des Versuchs wird einer der Kondensatoren mit einer strombegrenzten Spannungsquelle aufgeladen. Die Strombegrenzung wird auf $2\,\mathrm{A}$, die Spannung auf $2{,}7\,\mathrm{V}$ eingestellt. Nach erreichen der Ladeschlussspannung bleibt der Superkondensator für 10

Stunden mit der Spannungsquelle verbunden. Für einen weiteren baugleichen Kondensator wird der Versuch in gleicher Weise durchgeführt, wobei die Ladeschlussphase auf eine Dauer von 1 Stunde reduziert wird.

Anschließend wird die Selbstentladung anhand der Spannung an den Anschlussklemmen beobachtet. Abbildung 4.3 zeigt den zeitlichen Verlauf der Spannung am Superkondensator durch Selbstentladung ohne angeschlossenen Verbraucher. Zu den Messzeitpunkten liegt eine Belastung durch das Multimeter an, diese ist aufgrund des hohen Innenwiderstands von $10\,\mathrm{M\Omega}$ und der kurzen Dauer der Messung allerdings zu vernachlässigen. Der Verlauf, bei nur für eine Stunde angelegter Ladeschlussspannung, zeigt zu Beginn einen sehr steilen Abfall, der erst nach einigen Tagen einen näherungsweise linearen Verlauf annimmt. Ausgehend von der zehn stündigen Ladeschlussphase stellt sich der annähernd lineare Verlauf deutlich früher ein. Für die generelle Eignung als Energiespeicher für das geplante System werden hierdurch keine grundlegenden Einschränkungen erwartet. Diese Folgerung ergibt sich aus der Tatsache, dass im gewählten Anwendungsszenario zu jeder Sonnenphase wieder Energie in das System eingespeist werden kann und somit nur die Nacht überbrückt werden muss. Weiterhin liegt auch im Fall der relativ kurzen Ladeschlussphase nach knapp zwei Tagen noch ausreichend Spannung an, um einen Mikrocontroller mithilfe eines DC-DC-Boost-Converters zu betreiben.

4.3. DC-DC Boost-Konverter

Das prinzipbedingte Entladeverhalten eines Superkondensators führt dazu, dass die Spannung proportional zur entnommenen Energie abfällt. Dadurch kann effektiv nur ein Teil der Kapazität genutzt werden. Die tatsächlich verwertbare Kapazität wird somit direkt durch die minimale Eingangsspannung des zweiten DC-DC-Konverters (in Abbildung 4.1 rechts) bestimmt. Gleichung 4.2 liefert die nutzbare Energiemenge in Abhängigkeit von Nennspannung U des Superkondensators und Minimalspannung U_{min}. Um möglichst wenig Energie ungenutzt im Superkondensator zu belassen sollte vorzugsweise ein Spannungswandler ausgewählt werden der auch mit geringen Eingangspannungen zuverlässig arbeitet.

Je nach Betriebsspannung und Energieverbrauch der verwendeten MCU und der Effizienz des eingesetzten Konverters kann der Wert für U_{min} stark variieren. Mit einem einfachen, handelsüblichen DC-DC-Konverter (ME2108A33) mit fester Ausgangsspannung von $3{,}3\,\mathrm{V}$ und einem großzügig dimensionierten Verbraucher lässt sich experimentell eine Untergrenze bestimmen. Bei einer durchschnittlichen Belastung von $40\,\mathrm{mA}$ liegt die Untergrenze für den erfolgreichen Betrieb des Systems bei knapp unter $1\,\mathrm{V}$. In diesem Beispiel liegt die verwertbare Energiemenge E_2 bei $314{,}5\,\mathrm{W\,s}$. Auf den Einsatz eines Konverters der geringere Eingangsspan-

nungen erlaubt, wird in dieser Arbeit verzichtet. Hierfür sind mehrere Gründe zu nennen. Bei weiter sinkender Spannung wird die Effizienz bei Entnahme und späterer Wiedereinspeisung der Energie zunehmend schlechter. Weiterhin verbleiben bei 1 V Ladespannung lediglich ca. 13,7 % der Energie im Kondensator, da die Spannung quadratisch mit der Energiemenge im Kondensator zusammenhängt.

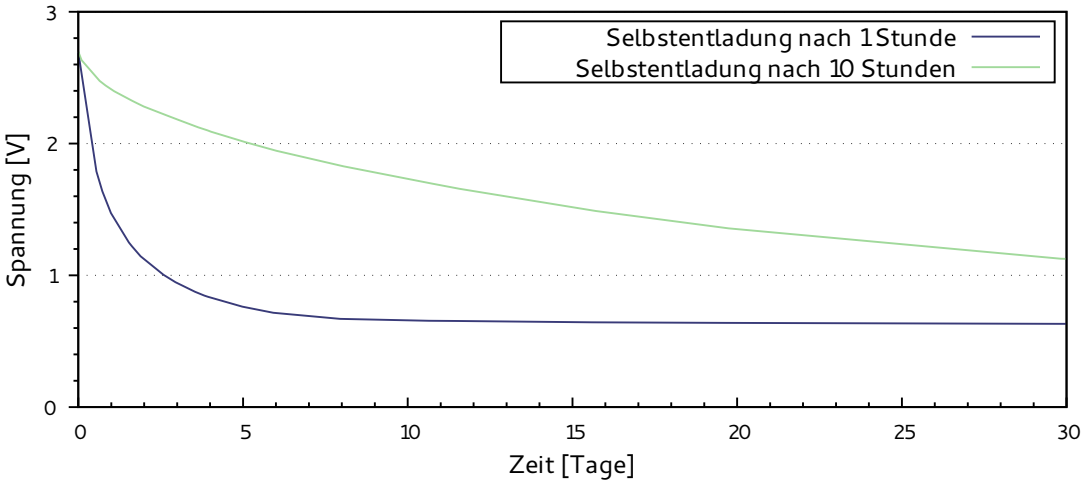


Abbildung 4.3.: Spannungsverlauf bei Selbstentladung des 100 F 2,7 V GreenCap

Abbildung 4.4.: Superkondensator

$$E_1 = \frac{1}{2} C \cdot U^2 \tag{4.1}$$

$$E_2 = \frac{1}{2} C \cdot (U^2 - {U_{min}}^2) \tag{4.2}$$

4.4. Ladeelektronik

Anhand der zuvor getroffenen Auswahl von Energiequelle und -Speicher ergeben sich für das dazwischenliegende Lademodul einige Rahmenparameter. Die Ladeschlussspannung des Superkondensators darf in keinem Fall überschritten werden, da dieser dadurch beschädigt

wird. Sie sollte dennoch flexibel einstellbar sein, um nötigenfalls andere Modelle von Superkondensatoren oder auch Lithium Akkus verwenden zu können. Weiterhin ist für eine effiziente Nutzung Rücksicht zu nehmen, auf die zuvor in Abschnitt 4.1 beschriebenen Eigenschaften von PV-Zellen. Dafür soll die Möglichkeit bereitgestellt werden, den Maximum Power Point der Solarzelle festzulegen. Um in späteren Versuchen potenziell unterschiedliche PV-Zellen einsetzen zu können, soll diese Einstellmöglichkeit variabel sein. Das Modul sollte mit einer möglichst niedrigen Spannung betrieben werden können, um auch mit relativ kleinen Solarzellen, bei schlechten Witterungsverhältnissen oder in der Dämmerung Energie aus der PV-Zelle extrahieren zu können.

Für den Aufbau der Ladeelektronik mit den zuvor genannten Kriterien eignet sich der LTC3105 von Lienar Technology. Für den Entwurf werden anhand des Datenblattes [27] die benötigten Bauteilgrößen zur Beschaltung des ICs (Integrated Circuit) berechnet. Die Ausgangsspannung des Bausteins wird über einen Spannungsteiler konfiguriert und ist nach Gleichung 4.3 zu berechnen.

$$V_{OUT} = 1{,}004\,\mathrm{V} \cdot \left(\frac{R1}{R2} + 1\right) \tag{4.3}$$

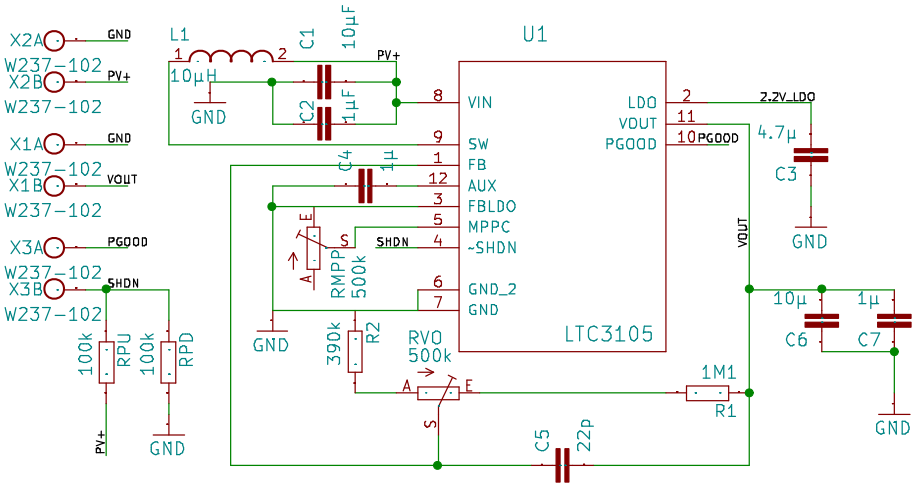


Abbildung 4.5.: Schaltplan des Ladereglers auf Basis des LTC3105

Um die Schaltung an dieser Stelle flexibel konfigurierbar zu halten, wird zusätzlich zwischen den beiden festen Widerständen R1 und R2 ein Potentiometer mit 500 kΩ (RVO) platziert.

Mit R1=1,1 MΩ und R2=390 kΩ ergibt sich ein einstellbarer Spannungsbereich zwischen 2,24 V und 5,12 V. Die MPP-Spannung wird nach Gleichung 4.4 über einen einzelnen Wider-

stand (RMPP) festgelegt, welcher ebenfalls als Potentiometer ausgeführt ist und mit einem Wert von 500 kΩ die Einstellung im Bereich bis 5 V ermöglicht.

$$V_{MPPC} = 10\,\mu A \cdot R_{MPPC} \tag{4.4}$$

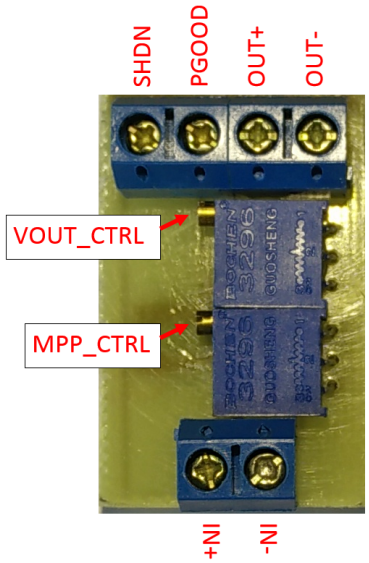
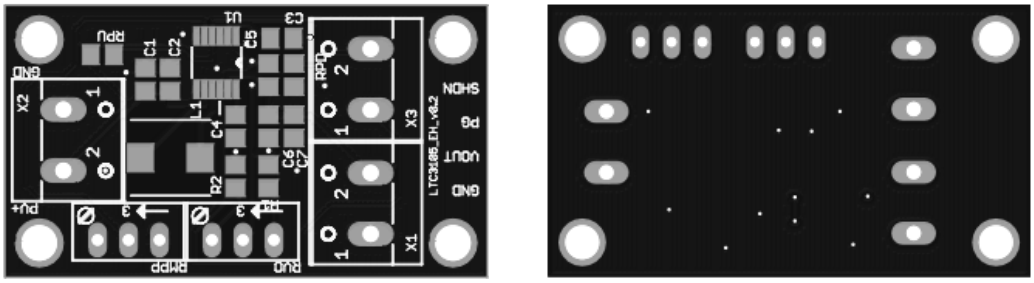


Abbildung 4.6.: Prototyp des Ladereglers

Als Spule wird eine niederohmige (44 mΩ) Speicherdrossel mit einer Induktivität von 10 µH verwendet. Die resultierende Beschaltung des Bausteins ist Abbildung 4.5 zu entnehmen, welche den entworfenen Schaltplan für die Ladeelektronik zeigt. Als Entwurfs-Software wird die Software KiCad EDA[1] verwendet. Nach der erfolgreichen Funktionsprüfung eines eigens angefertigten Prototypen der Platine, wird diese in mehrfacher Ausführung bei einem Auftragsfertiger bestellt und anschließend bestückt. Abbildung 4.6 zeigt den Prototyp mit der Seite der Anschlussklemmen und den Potentiometern. Zusätzlich zum Eingang für die PV-Zelle (IN-, IN+) und dem Ausgang für den Superkondensator (OUT-, OUT+) verfügt das Modul über einen Eingang zum Aktivieren und Deaktivieren der Schaltung (SHDN) und einen Ausgang der signalisiert, ob die Ausgangsspannung im Rahmen der Toleranz auf die Zielspannung geregelt wird.

Eine Überprüfung des Moduls durch Messungen mit einem Voltmeter zeigt, dass die Ladeelektronik die Rahmenanforderungen erfüllt. Sowohl die Einstellung der MPP-Spannung als auch der Ausgangsspannung funktioniert zuverlässig und mit ausreichender Genauigkeit

[1] http://kicad-pcb.org/



(a) Oberseite (b) Unterseite

Abbildung 4.7.: Platinenlayout des Lademoduls

($< 0{,}05$ V). Der Superkondensator wird über die PV-Zelle vollständig geladen, während die Ladeschlussspannung stets eingehalten wird. Nach der Überprüfung des Prototyps werden Platinen mit dem in Abbildung 4.7 gezeigten Layout bei einem Auftragsfertiger bestellt. Die Bauteilbestückung der leeren Platinen erfolgt händisch. Abbildung 4.8 zeigt die Module während diesem Arbeitsschritt.

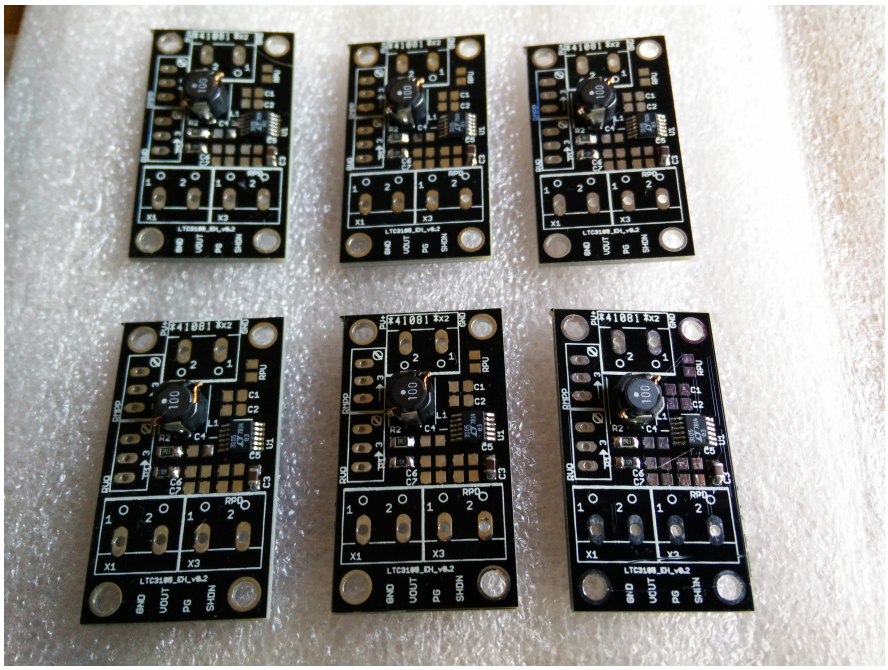


Abbildung 4.8.: Zwischenstand beim Bestücken der Platinen

4.5. Messung der Energieparameter

Der Energieverbrauch, die Einspeisung von Energie und die Spannung am Superkondensator soll gemessen werden um den Leistungsbedarf verschiedener Betriebsmodi und den verfügbaren Ladezustand zu bestimmen. Zu diesem Zweck wird eine Schaltung aufgebaut, die für ein permanentes Monitoring des EH-Systems verwendet werden kann. Die Anforderungen an das Messmodul werden nachfolgend angeführt. Zur Verwendung in stationären Messungen und in Feldtests soll der Betrieb sowohl an dem zu vermessenden System selbst, als auch an einem separaten Mikrocontroller möglich sein (vgl. MCU1 und MCU2 in Abbildung 4.1). Der Messbereich soll konfigurierbar sein, um in verschiedenen Systemkonfigurationen für die jeweilige Größenordnung der Leistungsaufnahme eine möglichst hohe Auflösung zu erreichen. Für eine hohe Genauigkeit soll es außerdem möglich sein die Schaltung über ein Referenzmessgerät zu kalibrieren. Weiterhin soll es möglich sein, das Messmodul durch eine eigene Spannungsquelle zu versorgen, um nötigenfalls den Energiebedarf des Moduls von dem überwachten System zu trennen. Für die Verwendung des Messmoduls unter dem Betriebssystems RIOT wird ein Treiber und eine Messanwendung entwickelt um die aufgezeichneten Messdaten zur weiteren Analyse an einen PC übertragen zu können.

Einen für diesen Einsatzzweck geeigneten Baustein bietet Texas Instruments in Form des INA226[2] an. Dieser ist in der Lage, gleichzeitig die Spannung und den Strom (über den Spannungsabfall an einem Shunt-Messwiderstand) an einem angeschlossenen Verbraucher zu überwachen. Der Baustein verwendet zur Messung einen $\Delta\Sigma$-ADC mit $500\,\mathrm{kHz}$ Abtastrate und lässt sich per Registerkonfiguration kalibrieren. Die Anbindung an einen Mikrocontroller erfolgt über eine Inter-Integrated Circuit (I^2C) Schnittstelle. Weiterhin bietet das Modul Einstellungen für Konvertierungszeiten zwischen $140\,\mu\mathrm{s}$ und $8{,}2\,\mathrm{ms}$ und einer automatischen Mittelwertbildung für bis zu 1024 Werte. Durch den höchsten messbaren Spannungsabfall von $81{,}92\,\mathrm{mV}$ am Messwiderstand kann der Messbereich durch Auswahl eines geeigneten Widerstands dimensioniert werden. Bei Verwendung eines $750\,\mathrm{m\Omega}$ Widerstands ergibt sich beispielsweise ein Messbereich bis ca. $109\,\mathrm{mA}$ bei einer Auflösung von rund $3{,}3\,\mu\mathrm{A}$. Für einem Messwiderstand von $10\,\Omega$ ergibt sich ein Messbereich bis ca. $8\,\mathrm{mA}$ bei einer Auflösung von $0{,}25\,\mu\mathrm{A}$. Der entsprechende Schaltplan ist in Abbildung 4.9 dargestellt.

Wie bereits bei der Ladeelektronik wird vorab ein Prototyp der Platine angefertigt, um die ordnungsgemäße Funktionalität zu überprüfen. Der Prototyp bietet die Möglichkeit, die I^2C-Adresse des Bausteins mit Jumper-Brücken einzustellen, und so mehrere baugleiche Module gleichzeitig an einem Bus zu betreiben. Der flexible Messbereich wird bei dem Prototyp

[2] http://www.ti.com/lit/ds/symlink/ina226.pdf

ebenfalls mittels Jumper-Brücken realisiert, die zum Umschalten der Messwiderstände dienen. Evaluierungsmessungen ergeben, dass das Modul im Rahmen der Bauteiltoleranzen funktioniert. Allerdings zeigt sich, dass der Übergangswiderstand an den Messbereichs-Jumpern zwischen Umschaltvorgängen starken Abweichungen unterliegt. Da sich dieser veränderliche Widerstand mit dem Messwiderstand in einer Reihenschaltung befindet werden die Messwerte dadurch direkt beeinflusst. Bei einer zweiten Revision der Schaltung (siehe Abbildung 4.10) wird daher versucht, mit parallel geschalteten DPDT-Schaltern (*double pole, double throw*, siehe [28]) diesen Effekt zu reduzieren.

Ergebnisse von Tests mit dieser Revision zeigen, dass das Problem aufgrund der physikalischen Empfindlichkeit der Schalter weiterhin besteht. Werden diese leicht angedrückt oder Vibrationen ausgesetzt, meldet das Messmodul abweichende Werte und es wird eine erneute Kalibrierung notwendig. Durch die gefederten Kontakte verändert sich zusätzlich die Abweichung über einen Zeitraum von mehreren Minuten.

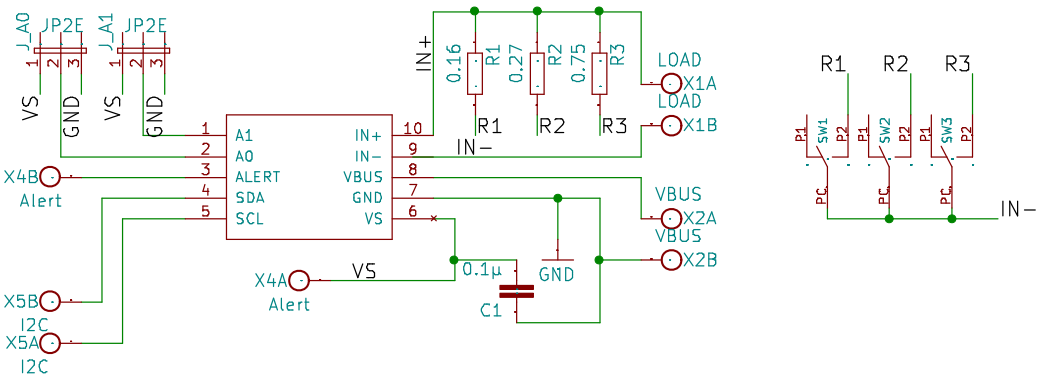


Abbildung 4.9.: Schaltplan des Messmoduls

Bei der dritten Revision wird im Unterschied zu den Prototypen auf hochwertigere Schalter mit Silberkontaktierungen gesetzt, die einen niedrigeren Übergangswiderstand garantieren. Laut Datenblatt kann dadurch der Innenwiderstand auf ein Maximum von $10\,\mathrm{m\Omega}$ begrenzt werden. Evaluierungsmessungen haben gezeigt, dass die Änderung des Innenwiderstands bei geschlossenem Zustand zwischen verschiedenen Schaltvorgängen nun wesentlich geringer ausfällt. Bei einem Wechsel der Schaltposition blieb die Abweichung bei allen Versuchen unterhalb von $1\,\mathrm{m\Omega}$. Diese Schwankung wirkt sich direkt auf den Messfehler aus, wenn

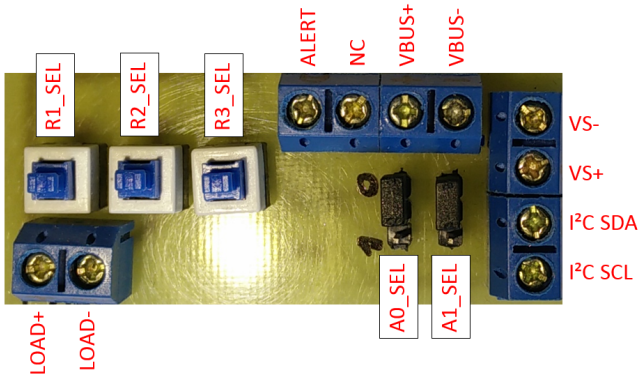
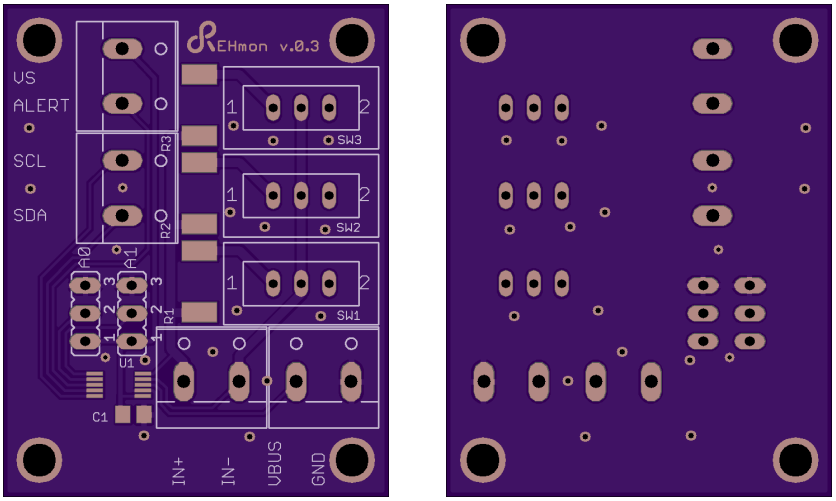


Abbildung 4.10.: Zweiter Prototyp des Messmoduls

das Modul nach dem Umschalten des Messbereichs nicht erneut kalibriert wird. Selbst im Messbereich mit dem kleinsten Messwiderstand von $160\,\mathrm{m\Omega}$ bleibt der dadurch verursachte Fehler nun unter $0{,}7\,\%$. Im Messbereich für die kleinsten Ströme ($750\,\mathrm{m\Omega}$ Messwiderstand) entspricht das einem Fehler von knapp $0{,}13\,\%$, was unter Betrachtung anderer Einflussfaktoren wie Temperatur und Leitungswiderständen vernachlässigbar wenig ist. Abbildung 4.11 stellt das finale Platinenlayout dar. Ein vollständig bestücktes Modul wird in Abbildung 4.12 gezeigt.



(a) Oberseite (b) Unterseite

Abbildung 4.11.: Platinenlayout des Messmoduls

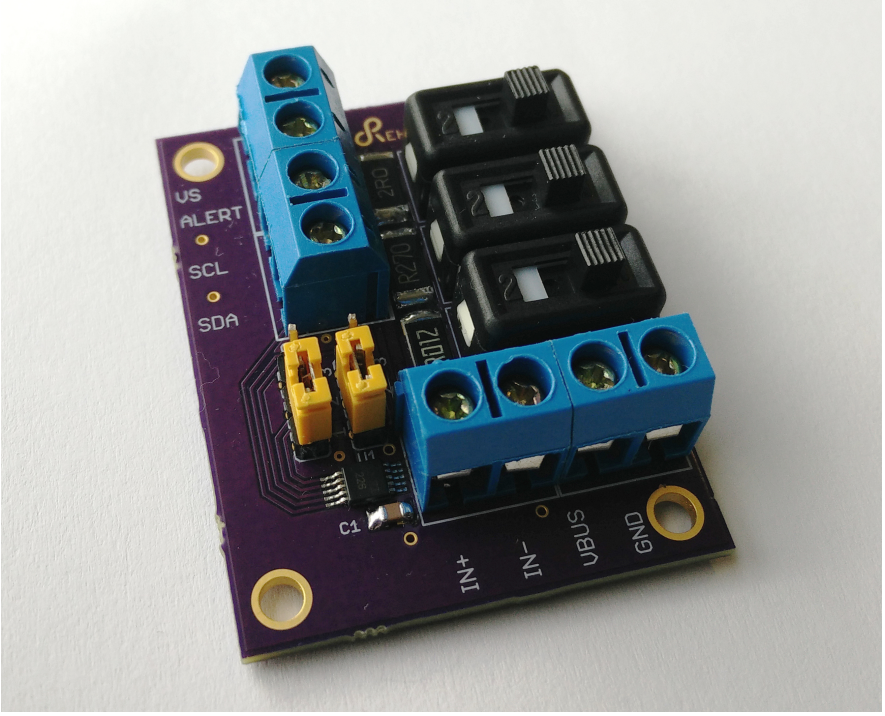


Abbildung 4.12.: Vollständig bestückte Platine des Messmoduls

4.6. Aufzeichnung der Messdaten

Ein Einsatzzweck der Testplattform ist es, unter möglichst authentischen Bedingungen Feldtests mit Sensorknoten durchzuführen. Dazu ist eine Komponente für das Protokollieren der vom Messmodul geliefert Daten erforderlich. Aufgrund des eingeschränkten Volumens des internen Speichers einer klassischen MCU, wird hierzu ein externes Speichermedium benötigt. Softwareseitig soll für das Medium ein RIOT-Treiber implementiert werden, der die Verwendung auf allen Plattformen erlaubt, welche die entsprechende Hardware-Schnittstelle zur Anbindung bieten. Um die Verarbeitung der aufgezeichneten Daten weiter zu vereinfachen, soll außerdem die Unterstützung für ein Dateisystem integriert werden. Damit wird das spätere Auslesen und die Handhabung der Messdaten erheblich erleichtert, indem die Daten beispielsweise direkt als CSV-Datei (Comma-separated Values) abgelegt werden können. Weiterhin entsteht die Möglichkeit der Firmware des Mikrocontrollers dadurch z. B. Konfigurationsdateien bereitzustellen.

4.6.1. Speichermedium

In diesem Abschnitt wird die Auswahl eines geeigneten Speichermediums erläutert. Hierzu werden die folgenden Anforderungen an das Medium betrachtet:

- hohe Kompatibilität zu diversen MCUs
- Speichervolumen für mehrtägige Messungen
- geringer Energieverbrauch
- interoperabler Datenzugriff

Ein potenziell geeignetes Medium unter diesen Gesichtspunkten stellen Speicherkarten im SD-Format dar. Um das weiter zu eruieren werden die diesbezüglichen Eigenschaften von SD-Karten genauer untersucht.

SD-Karten unterstützen neben dem nativen SD-Host Interface einen SPI-Modus, der zwar langsamer ist, jedoch ohne dedizierte SD-Peripherie des Mikrocontrollers auskommt. Dadurch besteht eine sehr hohe Kompatibilität zu diversen Mikrocontrollern. Eine Überprüfung von 104 RIOT-kompatiblen Zielplattformen anhand der momentan aktuellen RIOT Version *2018.01* zeigt, dass bereits 81 davon eine SPI-Implementierung besitzen und damit potenziell kompatibel sind. Auf den anderen Plattformen besteht die Möglichkeit, nachträglich die Kompatibilität durch einen SPI-Treiber für bisher nicht unterstützte Peripherie bereitzustellen. Alternativ kann eine Software-basierte SPI-Implementierung durch manuelles Ansteuern der Pins über die GPIO-Abstraktion bereitgestellt werden. Einzig vier Plattformen verbleiben, welche aufgrund fehlender GPIO-Treiber weder mittels Hardware- noch Software-SPI verwendet werden können.

Ein Vorteil von SD-Karten besteht in der einfachen Handhabung und der direkten Interoperabilität mit handelsüblichen Geräten. Weiterhin besitzen SD-Karten bereits in ihrem kleinsten Formfaktor *microSD* bis zu 512 GB[3] Speichervolumen und bieten damit auch für Langzeitmessungen mit erhöhtem Datenaufkommen eine ausreichende Kapazität. Aufgrund der häufigen Verwendung von SD-Karten in Mobilgeräten wie Smartphones kann angenommen werden, dass der Energieverbrauch von SD-Karten auch Feldtests ohne externe Stromversorgung zulässt. Aus der SD-Spezifikation [29] lassen sich dazu allerdings keine genauen Angaben entnehmen. Es wird explizit genannt, dass der Energieverbrauch im Energiesparmodus nicht spezifiziert ist und zwischen verschiedenen Produkten variieren kann. Es werden lediglich

[3] https://www.businesswire.com/news/home/20180122006328/en/UK's-Integral-Memor-Market-Largest-Capacity-microSD

Obergrenzen von einer Sekunde für das Zurückkehren aus dem Energiesparmodus und eine maximale Leistungsaufnahme von 0,36 W während des Schreibvorgangs im *Default Speed Mode* genannt. Inwiefern dieser Wert in der Praxis zutrifft, soll im Rahmen der Evaluierung genauer überprüft werden, um die Grenzen des Systems detailliert aufzuzeigen.

Treiber-Implementierung: sdcard_spi

Nachfolgend wird die Implementierung des Treibers sdcard_spi behandelt. Dazu werden einige grundlegende Funktionen von SD-Karten und deren Umsetzung im Treiber beschrieben. Zusätzlich zum Treiber wird eine Testapplikation implementiert, welche es Entwicklern erlaubt die ordnungsgemäße Funktion des Treibers auf verschiedenen Zielplattformen zu testen.

Wie bereits erwähnt, kommunizieren SD-Karten nativ über eine eigene SD-Host Schnittstelle. Wird eine Karte eingeschaltet, befindet sie sich standardmäßig im SD-Modus. Um mit der Karte über SPI zu kommunizieren, muss diese zunächst in den SPI-Modus versetzt werden. Dazu ist in der SD Spezifikation eine Power-Up-Sequenz vorgesehen. Einige SD-Karten lassen sich nur dann erfolgreich in den SPI-Modus versetzen, wenn der MISO-Pin mit einem Pull-Up-Widerstand versehen ist. Dieser muss mindestens ab beginn der Power-Up-Sequenz bis zum erfolgreichen Software-reset aktiviert sein. Anschließend befindet sich die Karte im SPI-Modus und arbeitet mit Push-Pull Ansteuerung, wodurch der Widerstand nicht mehr benötigt wird. Auf MCU-Boards mit fest verbautem microSD-Slot, bei denen dieser Umstand nicht im Entwurf der Hardware berücksichtigt wurde, kann im Regelfall kein externer Pull-Up-Widerstand nachgerüstet werden. Die Kompatibilität zu dahingehend empfindlichen SD-Karten auszuschließen, sollte aufgrund der in Unterabschnitt 4.6.1 genannten Anforderungen bezüglich Kompatibilität und Interoperabilität vermieden werden. Eine einfache Lösung aus Sicht der Schnittstellen bietet die Erweiterung der Konfigurierbarkeit der Pull-Ups über die SPI-Schnittstelle. Jedoch unterstützt nicht jede Plattform die unabhängige Konfiguration von alternativen Funktionen der Pins (SPI) und deren Pull-Up Einstellungen. Zur Lösung wird daher entschieden die Initialisierung über eine Software-SPI Implementierung innerhalb des sdcard_spi Treibers abzuwickeln. Diese steuert die einzelnen Pins manuell über die GPIO-API an, welche die Konfiguration von internen Pull-Ups erlaubt. Die Software Variante wird nur während den ersten Schritten der Initialisierung verwendet, anschließend wird mittels Funktions-Pointer wieder auf die performantere Variante mit dedizierter SPI-Hardware umgeschaltet. Aus dieser Lösung ergibt sich der Nachteil, dass zur Initialisierung des Treibers statt eines einzelnen SPI-Enumerators alle Pins einzeln spezifiziert werden müssen, da eine Identifikation der jeweiligen Pins anhand des SPI-Enumerators bisher nicht plattformunabhängig möglich ist.

Zum Testen wird die remote-revb[4] Plattform verwendet, welche einen microSD-Slot besitzt. Bei der vorhanden Revision des Boards sind die Pins PA6 und PA7 zur Verwendung für die microSD-Karte vorgesehen. Diese sind bei der vorliegenden Produktionscharge jedoch zur Verwendung als ADC-Eingänge beschaltet. Um den microSD-Slot funktionstüchtig zu machen, sind zwei nachträgliche Brücken notwendig.[5]

Die Kommunikation mit SD-Karten basiert stets auf Kommandos, welche an die SD-Karte übertragen werden. Diese Kommandos können beispielsweise bestimmte Sequenzen einleiten, die Übertragung eines Registerwertes anweisen oder den Status einer Operation abfragen. Das Senden eines Kommandos wird in einer Methode gekapselt, welche blockiert, bis eine Rückmeldung der Karte empfangen wurde, oder es zu einem Timeout kommt.

Die Initialisierung der SD-Karte wird mit einer Finite-State-Machine (FSM) realisiert. Das Flussdiagramm in Abbildung 4.13 stellt die relevanten Fallunterscheidungen und den Ablauf der FSM dar. Bei der Initialisierung der Karte muss die Kommunikation mit einer Taktrate zwischen 100 kHz und 400 kHz durchgeführt werden. Alle Schritte, von der Konfiguration der verwendeten Peripherie im Zustand `SD_INIT_START`, bis zur Bereitstellung der Karte für anschließende Lese- und Schreiboperationen werden dabei von der FSM übernommen. Die zuvor erwähnte Power-Up-Sequenz wird im Zustand `SD_INIT_SPI_POWER_SEQ` ausgeführt. Am Ende des Zustands `SD_INIT_SEND_CMD_0` wird die Umschaltung von Soft- auf Hardware-SPI vorgenommen. Anschließende Kommandos werden dazu verwendet, die CRC-Funktion der Karte zu aktivieren, die ordnungsgemäße Kommunikation mit der Karte zu überprüfen und ihre Metadaten abzufragen, damit diese vom Treiber anschließend richtig angesprochen werden kann. Zum Abschluss wird die SPI-Bus Geschwindigkeit im Zustand `SD_INIT_SET_MAX_SPI_SPEED` auf die angegebene Maximalfrequenz von höchstens 50 MHz angehoben.

Lese- und Schreiboperationen des Treibers arbeiten block orientiert und verwenden gegebenenfalls automatisch die performantere Mehrfachübertragung[6] für mehrere Blöcke hintereinander.

Um die Schnittstelle übersichtlich zu gestalten, werden Low- und High-Level Operationen über zwei getrennte Header-Dateien bereitgestellt. Funktionen zur Initialisierung der Karte und zum Lesen und Schreiben von Blöcken werden in der Datei `include/sdcard_spi.h` deklariert. Funktionen zum Senden von Kommandos und zugehörige Literale befinden sich dagegen in der Datei `sdcard_spi/sdcard_spi_internal.h`. Beide Dateien befinden

[4] Die in dieser Arbeit verwendeten Board-Bezeichnungen beziehen sich auf deren eindeutige Namen in RIOT, siehe: https://github.com/RIOT-OS/RIOT/tree/2018.01/boards

[5] https://github.com/Zolertia/Resources/wiki/%5BRE-Mote%5D-Enabling-uSD

[6] siehe *7.2.3 Data Read* und *7.2.4 Data Write* in [29]

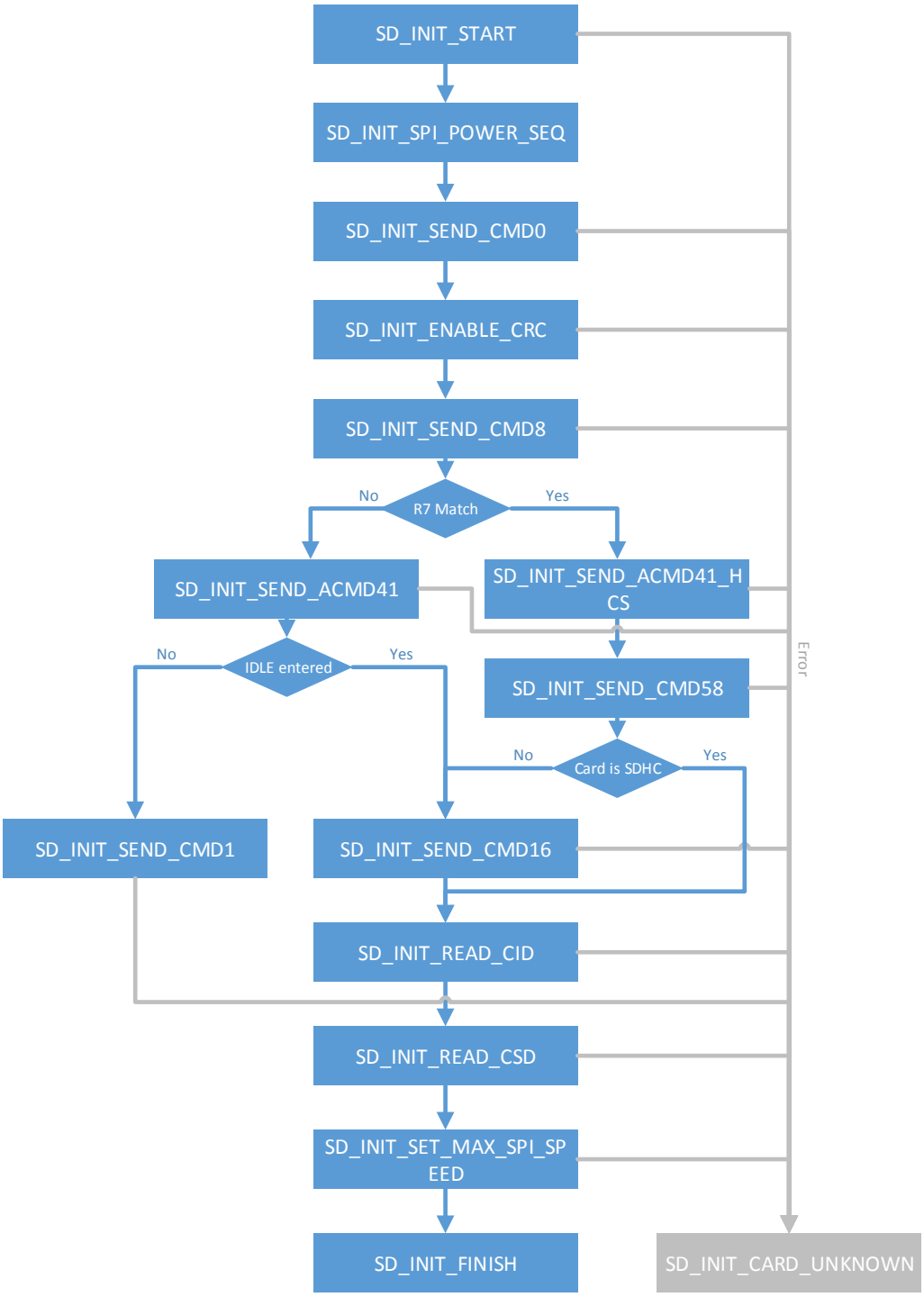


Abbildung 4.13.: Ablauf der Initialisierung der SD-Karte im sdcard_spi Treiber

sich im `drivers`-Verzeichnis von RIOT. Für einen detaillierteren Einblick sei an dieser Stelle auf den Quellcode im offiziellen git-Repository von RIOT oder den beigelegten Datenträger (siehe Anhang A) verwiesen [7].

4.6.2. Dateisystem

Um die aufgezeichneten Messdaten flexibel weiterverarbeiten zu können, soll mit einem Dateisystem eine einfache und interoperable Möglichkeit zum Datenaustausch gegeben werden. FAT ist ein Dateisystem, das auf SD-Karten eine hohe Verbreitung findet und sich auf kleinen Ressourcenarmen Mikrocontrollern einsetzen lässt. Eine Quelloffene FAT Implementierung mit GNU GPL kompatibler Lizenz ist das FatFs Generic FAT File System Module[8], welches unter anderem FAT und exFAT Unterstützung bietet. Weiterhin lässt sich das Modul abhängig von den benötigten Funktionen konfigurieren, um Beispielsweise einen geringeren Speicherbedarf zu erreichen. Nach Außen bietet das FatFs Modul dem Entwickler eine API, die sich an bekannten Linux Systemaufrufen orientiert, während Hardwareoperationen über eine vorgegebene Schnittstelle abstrahiert werden. Die darunterliegende Low-Level Implementierung ist in FatFs nicht enthalten und muss separat entwickelt werden. Dazu muss für alle Methodendeklarationen der Schnittstelle `diskio.h` eine Implementierung bereitgestellt werden. Über diese Abstraktion ist die transparente Nutzung verschiedener Speichermedien möglich.

Das Dateisystem FatFs wird in das Virtual-File-System „vfs“integriert, um die Austauschbarkeit von Dateisystemen zu ermöglichen. Abbildung 4.14 zeigt die Anordnung der verschiedenen Komponenten für eine Logging-Applikation. Mit der vfs-Integration ist für die Applikation vollständig transparent, welches Dateisystem verwendet wird. Um diese Flexibilität auch in den unteren Schichten zu erreichen, wird die FatFs-Schnittstelle *diskio* oberhalb der generischen Datenträger-Schnittstelle *mtd* von RIOT implementiert. Somit ist für FatFs transparent, welches Speichermedium verwendet wird. Implementierungen für weitere Speichermedien müssen dadurch lediglich eine mtd-Unterstützung mitliefern, damit darauf FatFs verwendet werden kann. Entsprechend ist zwischen der mtd-Schnittstelle und dem sdcard_spi-Treiber die Komponente „mtd_sdcard“zu sehen, welche die mtd-Schnittstelle für den Treiber sdcard_spi implementiert.

[7] RIOT-Release 2018.01 https://github.com/RIOT-OS/RIOT/releases/tag/2018.01
[8] http://elm-chan.org/fsw/ff/00index_e.html

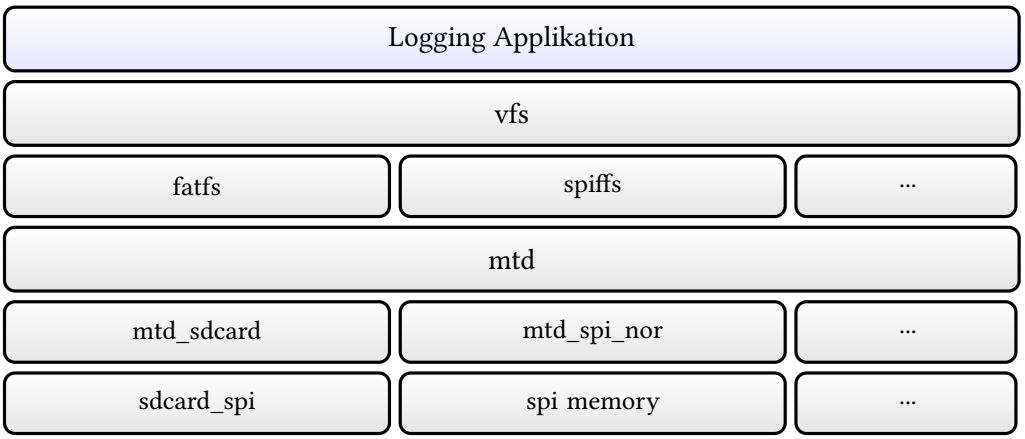


Abbildung 4.14.: Komponenten-Stack der Logging-Applikation

4.7. Mikrocontroller

Das in vorherigen Tests verwendete Board samr21-xpro bietet umfassende Unterstützung bzgl. Peripherie-Treibern wie (I^2C), SPI oder dem integrierten Funkmodul. Das Board besitzt eine Energiesparende MCU, deren Energieverbrauch komfortabel über zwei Pins gemessen werden kann. Dennoch lässt es sich ohne erhebliche Modifikationen an der Hardware nicht für Feldtests verwenden. Unter anderem verhindern ein fest verlöteter Embedded Debugger (EDBG) und dauerhaft aktivierte LEDs die reale Ausnutzung der Energiesparmechanismen der MCU. Um den EDBG-Chip zu deaktivieren, müssen 18 verschiedene Lötbrücken entfernt werden. Um den EDBG anschließend zu reaktivieren, müssen alle Lötbrücken wieder geschlossen werden. Häufig neue Firmwareversionen auf das Board zu laden, erweist sich damit als nicht praktikabel. Im Rahmen von Tests und Evaluierungsmessungen kann dies aber durchaus notwendig sein. Da alle entwickelten Komponenten der Testplattform plattformunabhängig ausgelegt sind, wird für den Feldtest die Verwendung eines anderen Evaluierungsboards angestrebt.

Als besonders geeignet zeigt sich das nucleo-l476 der Firma STMicroelectronics. Dieses ist werksseitig dafür ausgelegt den Flash/Debug-Chip vollständig vom Evaluierungsboard abtrennen zu können. In Abbildung 4.15 ist das Board mit eingezeichneter Trennlinie zu sehen. Über den CN4/SWD-Header kann das abgetrennte Erweiterungsboard weiterhin per Kabel angebunden werden.

Um die Handhabung hier noch weiter zu vereinfachen, werden drei weitere Jumper-Brücken angebracht, anstatt das Board abzutrennen. Mit den Jumper-Brücken wird die Funktion der Lötbrücken SB12 (siehe Abbildung 4.16) und SB2 (siehe Abbildung 4.15) einfach zugänglich

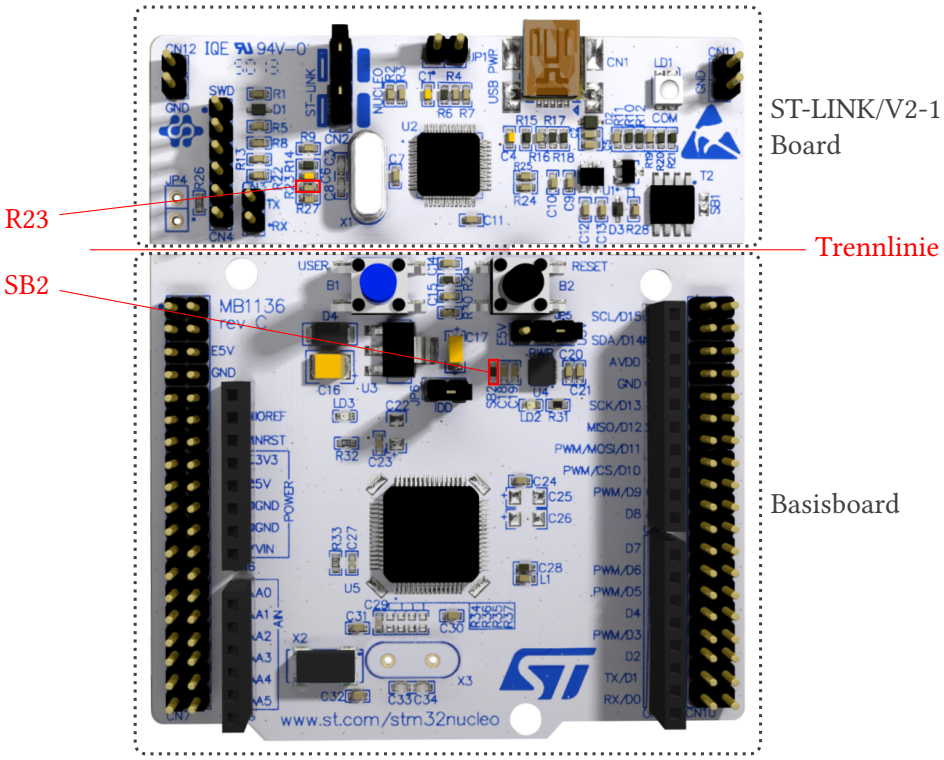


Abbildung 4.15.: STM32L476 Nucleo-64 Board Oberseite

und schnell konfigurierbar gemacht. SB12 verbindet den NRST-Pin der MCU mit dem ST-LINK-Board. SB2 verbindet die MCU mit dem 3,3 V Spannungsregler U4. Beide müssen getrennt werden, damit das Board über einen externen Spannungsregler betrieben werden kann. Laut dem Benutzerhandbuch[9] sind außer dem Auftrennen dieser beiden Lötbrücken keine weiteren Modifikationen nötig. Messungen zeigen, dass bei Verwendung einer externen Spannungsquelle durch den Widerstand R23 (47,2 kΩ, siehe Abbildung 4.15) ein Leckstrom zum ST-LINK-Chip auftritt. Die dritte Jumper-Brücke dient dazu, diese Verbindung zu trennen. Eine Kontrollmessung zeigt, dass mit diesen Anpassungen der Strombedarf mit aktiver Real-Time Clock (RTC) für das gesamte Board unterhalb von 1 µA liegt. Damit ist diese Plattform auch zur Verwendung in weiteren Feldtests geeignet. Werden die Jumper-Brücken wieder verbunden, kann das Board weiterhin wie im Auslieferungszustand neu bespielt werden.

[9] http://www.st.com/resource/en/user_manual/dm00105823.pdf

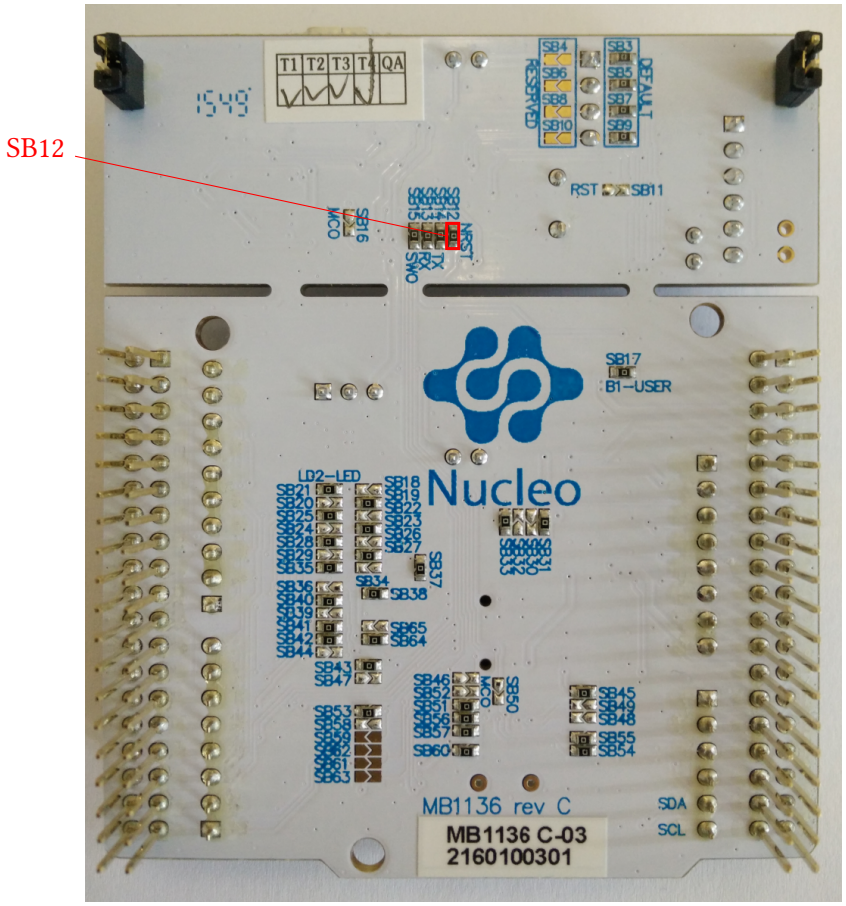


Abbildung 4.16.: STM32L476 Nucleo-64 Board Unterseite

4.7.1. Energiesparmodi für STM32L4 basierte Plattformen

Das Power-Management Modul von RIOT bietet eine Schnittstelle zum aktivieren der verfügbaren Energiesparmodi. Für STM32L4-basierte Plattformen wird anhand der Angaben im Datenblatt [30] die Implementierung der vier Energiesparmodi *SHUTDOWN*, *STANDBY*, *STOP2* und *SLEEP* in die Power Management Variante für STM32-basierte MCUs integriert. Diese befindet sich in der Datei pm.c[10].

[10] https://github.com/RIOT-OS/RIOT/blob/master/cpu/stm32_common/periph/pm.c

4.7.2. Real Time Clock Implementierung

Eine Real Time Clock (RTC) dient auf Mikrocontrollern dazu eine Referenz zur physikalischen Uhrzeit zu halten. Diese benötigen aufgrund Ihrer langsamen Taktung in der Regel besonders wenig Energie und können über eine separate Batterie betrieben werden. In Low-Power Applikationen benötigt das System zum verlassen des Energiesparmodus einen externen Trigger oder Zeit-basierte Ereignisse. Diese Aufgabe kann durch die RTC erfüllt werden. Für STM32L4-basierte Plattformen wird zu diesem Zweck die Unterstützung in den RIOT RTC-Treiber für STM32-basierte Plattformen integriert.

Tests der Implementierung zeigen, dass im generischen Teil (stm32_common) der RTC-Funktionen mehrere Fehler existieren. Diese zeigen sich darin, dass ein Aufruf der Funktion `rtc_get_time(struct tm *time)` fehlerhafte Auswirkungen auf nachfolgende Aufrufe bewirkt und dadurch inkonsistente Ergebnisse liefert. Die Ursache hierfür ist eine fehlende Synchronisation, die aufgrund der verschiedenen Frequenzen zwischen APB- und RTC-Takt notwendig ist. Der generische Test für den RTC-Treiber maskiert diesen Fehler jedoch durch einen Aufruf von `xtimer_usleep()`. Dadurch ist die fehlende Synchronisation auch bei einer ordnungsgemäßen Ausführung des Tests zunächst nicht sichtbar. Weiterhin wird im Treiber die Spezifikation zur Lesereihenfolge der RTC_TR und RTC_DR-Register verletzt. Dadurch verursacht eine MCU-interne Funktion zum atomaren Auslesen beider Register, dass der Inhalt des RTC_DR-Registers zwischen zwei Aufrufen gesperrt bleibt und beim zweiten Aufruf zusammen mit einem neuen Wert des RTC_TR-Registers kombiniert wird. Zur Behebung dieses Fehlers wird die Implementierung an die Spezifikationen im Datenblatt angepasst.

4.7.3. Taktkonfiguration

Die Taktkonfiguration befindet sich in der Datei `cpu/stm32l4/cpu.c` und kann über die Methode `cpu_clock_init()` aufgerufen werden. Diese ist durch das Schlüsselwort static ausschließlich zur Verwendung innerhalb dieser Datei vorgesehen. Nachdem die MCU den Low-Power-Mode z. B. aufgrund eines Interrupts verlässt, muss die Taktkonfiguration gegebenenfalls erneut vorgenommen werden. Dazu muss die Funktionalität der zuvor genannten Methode für andere Komponenten verfügbar gemacht werden. Der monolithische Ansatz der Methode `cpu_clock_init()` initialisiert Clock-Domains unabhängig von deren Verwendung. Das betrifft auch Clock-Domains, die von der RTC benötigt werden. Da dies bei mehrmaligem Aufwachen aus dem Energiesparmodus zu Problemen wie Jitter, einem Reset oder dem Anhalten der RTC führen kann, wird die Implementierung des neuen Interfaces zur Konfiguration von Takteinstellungen, namens stmclk bevorzugt. Dieses erlaubt eine differenziertere Einstellung

der Clock-Domains. Das Interface dient in RIOT dazu, die Taktkonfiguration der vielzähligen unterstützten STM32-Mikrocontroller zu vereinheitlichen. Die Konfiguration folgt auf diesen Plattformen meist einem ähnlichen Schema. Über das Interface können verschiedene Clock-Domains initialisiert, ein- und ausgeschaltet werden. Dazu stehen folgende Methoden bereit:

```
1 void stmclk_init_sysclk(void);
2 void stmclk_enable_hsi(void);
3 void stmclk_disable_hsi(void);
4 void stmclk_enable_lfclk(void);
5 void stmclk_disable_lfclk(void);
6 void stmclk_bdp_unlock(void);
7 void stmclk_bdp_lock(void);
```

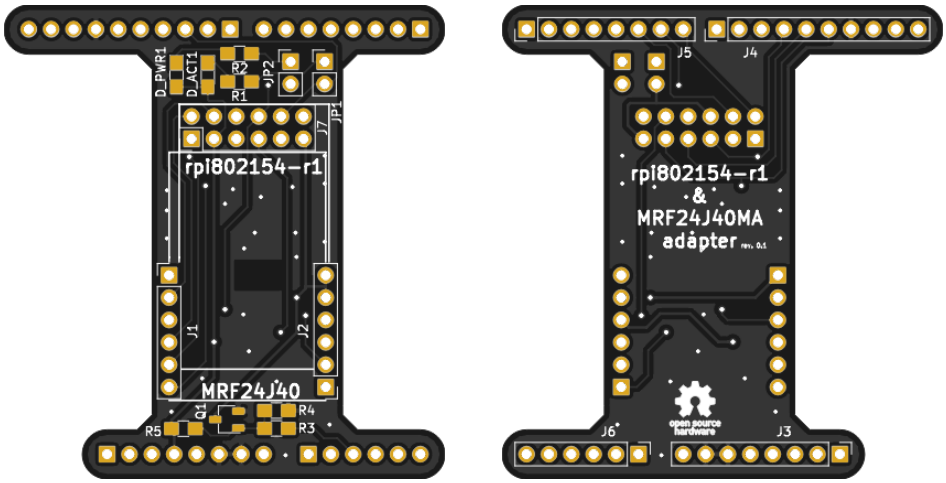
Die Methode `stmclk_init_sysclk()` konfiguriert während des Bootvorgangs die zum Betrieb der MCU notwendigen Taktgeber. Entsprechend können die Methoden `*hsi()` und `*lfclk()` aufgerufen werden, um die HSI-Clock (High Speed Internal) respektive LF-Clock (Low Frequency) unabhängig voneinander ein und auszuschalten. Mittels `*bdp_unblock()` und `*bdp_unblock()` kann die Backup Domain Protection gesteuert werden, was z. B. zur Konfiguration der geschützten RTC-Register notwendig ist. Für die MCU des verwendeten Evaluierungsboards werden diese Methoden anhand der Datenblattvorgaben umgesetzt.

4.8. Funkmodul

Das nucleo-l476 Board besitzt kein integriertes Funkmodul, weshalb auf eine externe Anbindung zurückgegriffen wird. Diese erlaubt zusätzlich eine bessere Kontrolle über die Energieversorgung. Wird das Modul nicht verwendet, kann die Versorgungsspannung des vollständig abgeschaltet werden. Weiterhin ergibt sich die Möglichkeit, Messungen mit verschiedenen Modellen durchzuführen, ohne auf eine andere MCU ausweichen zu müssen. Zur unkomplizierten und zuverlässigen Anbindung eines Funkmoduls wird dafür eine Adapterplatine entworfen. Diese soll mit dem nucleo-l476 und anderen Evaluierungsboards mit dem bekannten Arduino-Header kompatibel gehalten werden. Die Platine wird so entworfen, dass zwei unterschiedliche Modelle damit verwendet werden können, für die bereits ein RIOT-Treiber existiert. Zum einen wird das Raspberry Pi 802.15.4 Modul[11] von openlabs unterstützt, welches auf dem AT86RF233 von Atmel basiert. Zum anderen kann das Modul MRF24J40MA[12] (bzw.

[11] http://openlabs.co/store/Raspberry-Pi-802.15.4-radio
[12] http://www.microchip.com/downloads/en/devicedoc/70329b.pdf

MOD-MRF24J40[13]) basierend auf dem MRF24J40 von Microchip verwendet werden. Abbildung 4.17 zeigt das entworfene Layout. Auf der Platine wird ein FET (Feldeffekttransistor) angebracht mit dem die Spannungsversorgung des Funkmoduls ein- und ausgeschaltet werden kann. Über die Bestückung einer der beiden Widerstände R4 oder R5 kann eingestellt werden, ob das Funkmodul standardmäßig ein- oder ausgeschaltet ist. Weiterhin besitzt die Platine zwei LEDs (D_PWR1, D_ACT1), die für Demonstrationszwecke zur Visualisierung von Duty-Cycling Anwendungen verwendet werden können. Über die beiden Jumper-Brücken (JP1 und JP2) lassen sie die LEDs deaktivieren um deren Energieverbrauch im Produktiveinsatz zu eliminieren. Für detaillierte Angaben zur Bauteilauswahl und den Quelldateien zu Schaltplan und Layout sei auf das entsprechende Repository[14] verwiesen.



(a) Oberseite (b) Unterseite

Abbildung 4.17.: Platinenlayout des Funkmodul-Adapters

4.9. Persistenter Zustand im Energiesparmodus

Im sparsamsten Energiesparmodus *SHUTDOWN* (siehe Unterabschnitt 4.7.1) werden fast alle Komponenten der MCU deaktiviert. Dies gilt ebenso für die Erhaltung des Arbeitsspeichers. Wird dieser Energiesparmodus genutzt, geht zwischen verschiedenen Zyklen der gesamte Applikationszustand im RAM verloren. Um dennoch einige Zustandsinformationen zwischen verschiedenen Ausführungen zu speichern, ohne dafür auf externen Speicher zurückzugreifen,

[13] https://www.olimex.com/Products/Modules/RF/MOD-MRF24J40/resources/MOD-MRF24J40.pdf
[14] https://github.com/MichelRottleuthner/rpi-mrf-radio-shield

kann auf der verwendeten MCU ein kleiner Speicherbereich der RTC verwendet werden. Dieser stellt 32 Register mit jeweils 32 bit bereit und wird über die Backup Domäne der RTC mit Energie versorgt. Der RTC Treiber stellt hierfür allerdings keine Schnittstelle zu Verfügung. Daher wird das Modul *backup_reg* entwickelt, das den Zugriff auf diese Register in der folgenden Schnittstelle kapselt:

```
1 int backup_reg_read(uint32_t idx, uint32_t *data);
2 int backup_reg_write(uint32_t idx, uint32_t val);
```

Das Modul implementiert den Ablauf zum Sperren und Entsperren des Registerschreibschutzes und stellt sicher, dass nur auf gültige Register zugegriffen wird.

4.10. Zeitsynchronisation

Die Aufgezeichneten Messwerte sollen auf dem Sensorknoten mit Zeitstempeln versehen werden. Hierfür ist es notwendig den RTC-Baustein mit einer externen Zeitreferenz zu synchronisieren. Eine Variante um dies zu erreichen, ist die Verwendung eines externen Zeitempfängers. Hierfür eignet sich z. B. ein GPS (Global Positioning System) Modul oder ein Empfänger für das Zeitsignal des Senders DCF77 in Mainflingen. Eine andere Variante besteht in der Netzwerk-basierten Synchronisation über NTP (Network Time Protocol). Da in RIOT bereits eine Implementierung für SNTP (Simple NTP) existiert und dadurch der Einsatz zusätzlicher Hardware umgangen werden kann, wird diese Lösung ausgewählt. Die generische Schnittstelle der RTC stellt eine Funktion zum Setzen der Systemzeit bereit, diese bietet allerdings lediglich eine Sekundengenaue Auflösung. Für einige Anwendungsfälle wie dem Monitoring von Umgebungstemperaturen reicht das bereits aus. Um die Plattform später auch für Multi-Hop Szenarien und Verfahren wie TSCH (Time Slotted Channel Hopping) einsetzen zu können, ist die Auflösung von einer Sekunde jedoch deutlich zu niedrig.[15] Es wird daher ein Modul namens node_time implementiert, das die RTC-Synchronisation zusammen mit der SNTP-Implementierung kapselt. Um eine höhere Genauigkeit zu erreichen, wird die Interrupt-gesteuerte Alarmfunktion der RTC verwendet. Zur Abwicklung der Synchronisation werden folgende Methoden bereitgestellt:

```
1 int node_time_sync(void);
2 uint32_t node_time_get_seconds_since_last_sync(void);
3 int node_time_get_ntp_rtc_diff_us(int64_t *diff, int avg_cnt);
4 int node_time_adjust_rtc_us(int64_t offset_us, int32_t drift_ppb);
```

[15] Typische Dauer eines TSCH-Slots: 10ms, siehe https://tools.ietf.org/html/rfc7554

Außerdem werden diverse Hilfsfunktionen für den Zugriff auf Zeitbezogene Systemparameter wie dem Bootzeitpunkt oder der Dauer der letzten Schlafphase bereitgestellt. Für Details zur Implementierung sei auf die Dateien `node_time.c` und `node_time.h` im Applikationsverzeichnis (siehe Anhang A) verwiesen.

4.11. Duty-Cycling Anwendung

Durch die Kombination der zuvor eingeführten Komponenten lässt sich mit dem nucleo-l476 eine einfache Anwendung mit Duty-Cycling realisieren. Abbildung 4.18 zeigt die Stromaufnahme der MCU in einer Anwendung, die alle zwei Sekunden ein Paket an einen anderen Knoten überträgt und dazwischen in den tiefsten Energiesparmodus wechselt. Diesen Schlafmodus verlässt die MCU durch einen RTC-gesteuerten Interrupt. Zum Speichern von Zustandsinformationen zwischen den Aktiv-Zyklen wird die zuvor gezeigte Lösung auf Basis der Backup-Register verwendet.

Beim Sendevorgang beträgt die Stromaufnahme ca. $13\,\mathrm{mA}$ wohingegen im Schlafmodus (*SHUTDOWN*) nur $0{,}9\,\mu\mathrm{A}$ benötigt werden. Zum Messen wird ein digitales Sampling-Multimeter (Keithley DMM7510 7,5 Digits) über JP6 (IDD Header[16]) mit dem nucleo-l476 Board verbunden. Die Messwerte beziehen sich dementsprechend ausschließlich auf die MCU.

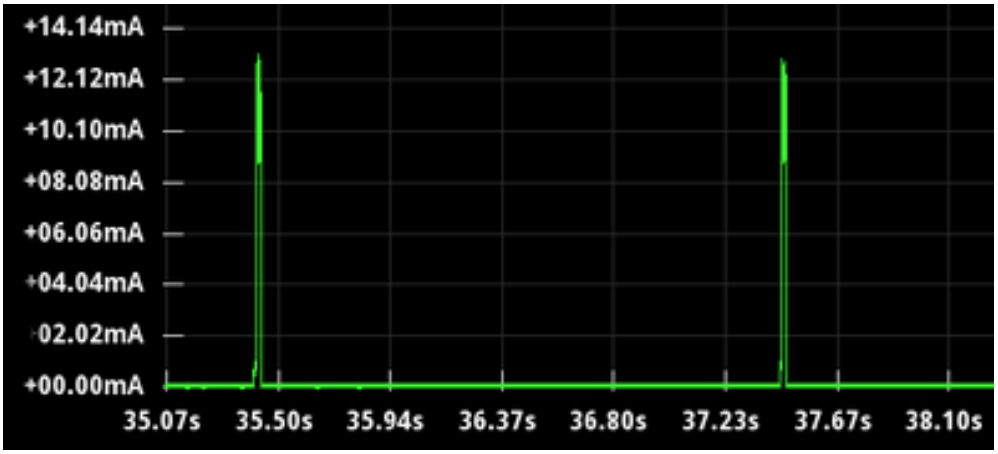


Abbildung 4.18.: Duty-Cycling mit RTC als Weakup-Trigger

[16] siehe Hardware layout and configuration: 6.6 in [31]

5. Evaluierung der Logging Komponente

Die Logging-Komponente dient dazu, die Messdaten wie z. B. den Energieverbrauch des EH-Systems auf einer SD-Karte persistent zu speichern. Dazu wird der Treiber sdcard_spi verwendet. Um Limitierungen der Speicherlösung identifizieren und deren Leistungsfähigkeit beziffern zu können, werden im folgenden Abschnitt Messungen beschrieben. Außerdem wird die weitere Entwicklung und Integration von generischen Abstraktionsschichten in die Teilmodule der Logging-Komponente erläutert. Die Abstraktion soll die Modularität erhöhen und damit Flexibilität durch das einfachere Austauschen von Einzelkomponenten ermöglichen.

5.1. Evaluierung des Treibers sdcard_spi

Zur Evaluierung des Treibers wird die ordnungsgemäße Funktion mit microSD-Karten unterschiedlicher Hersteller und unterschiedlichen Typs getestet. Tabelle 5.1 listet die Kartenmodelle mit einer eindeutigen Nummerierung. Nachfolgend wird diese Nummerierung zum Referenzieren der einzelnen Karten verwendet.

Alle Karten werden unter Verwendung der implementierten Testapplikation zunächst auf den Plattformen *samr21-xpro* und *remote-revb* eingesetzt. Zunächst wird über die Testapplikation mit den Kommandos *init*, *csd* und *cid* überprüft, ob die Initialisierung erfolgreich ausgeführt wird. Durch einen Abgleich der CID und CSD-Registerwerte mit Referenzwerten, welche über einen Kartenleser an einem PC ausgelesen werden, wird überprüft, ob diese durch den Treiber

Nr.	Hersteller	Modell	Kapazität	Typ
1	SanDisk	Ultra (SL32G)	32GB	SDHC
2	SanDisk	SU02G (A)	2GB	SDSC
3	SanDisk	SU02G (B)	2GB	SDSC
4	Intenso	UHS-I 3423470	16GB	SDHC
5	Kingston	SD-C01G	1GB	SDSC

Tabelle 5.1.: Verschiedene Modelle getesteter SD-Karten

korrekt übertragen werden. Anschließend wird mit den Kommandos write, read und copy getestet, ob Lese und Schreibvorgänge ordnungsgemäß ausgeführt werden.

Während dem Test treten teilweise Kommunikationsfehler auf. Eine Analyse des Fehlers liefert zu niedrig angesetzte Timeouts in der Karten Kommunikation als Ursache. Ein Erhöhen der Timeout-Parameter in der Treiberimplementierung behebt dieses Problem. Abschließend kann in keinem dieser Tests ein fehlerhaftes Verhalten hervorgerufen werden.

Zur Evaluierung des Treibers sdcard_spi werden anschließend verschiedene Messreihen durchgeführt. Zum einen wird untersucht, wie hoch der erreichbare Durchsatz beim Lesen und Schreiben ist. Eine weitere Metrik von Interesse, ist die Dauer des Initialisierungsvorgangs. Anhand dieser Daten soll eine Abwägung bezüglich der Dimensionierung von Puffergrößen und dem auftretenden Overhead durch die Initialisierung nach einem vollständigen Abschalten der Karte ermöglicht werden. Der zweite Aspekt der Messungen soll den Energieverbrauch in verschiedenen Zuständen der Karte beleuchten. Unterschieden wird hierbei zwischen Lesen, Schreiben, Initialisieren und Ruhezustand. Da verschiedene Karten sich erheblich in Ihren Eigenschaften unterscheiden können, werden fünf verschiedene Karten untersucht und miteinander verglichen. Die Karten unterscheiden sich unter anderem anhand des Herstellers, der Speicherkapazität und des SD-Karten Standards, um ein möglichst diversifiziertes Bild abzugeben.

5.1.1. Performance

Zum Messen der Performance wird eine Testapplikation implementiert. Für diese Messungen wird das nucleo-l476 Board eingesetzt, das auch in dem anschließenden Feldtest verwendung findet. Dieses wird mit einer Frequenz von $80\,\mathrm{MHz}$ getaktet. Zur Initialisierung der Karte wird die Standardkonfiguration des sdcard_spi-Treibers benutzt. Diese verwendet zum Aktivieren des SPI-Modus eine Taktfrequenz von $100\,\mathrm{kHz}$. Der darauf folgende Teil der Initialisierungsroutine erfolgt mit $400\,\mathrm{kHz}$. Nach der Initialisierung wird die Taktfrequenz des SPI-Bus auf $10\,\mathrm{MHz}$ angehoben. Der Durchsatz wird bei der Übertragung einer Datenmenge von $10\,\mathrm{MiB}$ gemessen. Hierbei wird in beiden Richtungen ein Puffer von 10 Blockgrößen ($5\,\mathrm{KiB}$) verwendet. Diese Größe wird aufgrund der Annahme gewählt, dass auf den Zielplattformen in der Regel ausreichend RAM dafür zur Verfügung steht. Beim Schreiben auf die Karte wird kontinuierlich aus einem vorinitialisierten statischen Speicherbereich gelesen. Dies wird so lange wiederholt, bis die Grenze von $10\,\mathrm{MiB}$ erreicht ist. Der vorinitialisierte statische Speicher enthält dabei über die gesamte Größe von $5\,\mathrm{KiB}$ alle Werte von 0 bis 255 in gleicher Häufigkeit und zufälliger Reihenfolge. Nach jedem Übertragungsvorgang wird über den Rückgabewert geprüft,

ob die Übertragung erfolgreich war, um verfälschte Ergebnisse durch übersprungene Blöcke auszuschließen. Außerdem ist die CRC-Überprüfung in beiden Richtungen aktiv.

Um die Dauer der Initialisierung zu bestimmen, muss die Karte ausgehend vom ausgeschalteten Zustand neu initialisiert werden. Um sicher zu gehen, dass der gemessene Wert keinen Ausreißer darstellt, wird die Messung über 1000 Initialisierungsvorgänge durchgeführt und daraus der Mittelwert gebildet. Damit die Karte automatisiert ausgeschaltet werden kann, wird ein MOSFET zwischen Spannungsversorgung und SD-Karte geschaltet. Das Steuersignal wird an einen Mikrocontroller-Pin angebunden, der in der Konfiguration des sdcard_spi-Treibers als Power-Pin hinterlegt wird. Nach fertiggestellter Initialisierung kann die Karte über diesen Pin abgeschaltet werden, der Treiber kümmert sich darum, die Spannungsversorgung der Karte vor der Software-seitigen Initialisierung zu aktivieren. Die Zeit wird direkt auf dem Mikrocontroller über die Differenz zwischen zwei Aufrufe der Funktion `xtimer_now()` gemessen. Damit die ausreichende Genauigkeit dieser Zeitmessung verifiziert werden kann, schaltet die Testapplikation bei Beginn und Beendigung des zu messenden Vorgangs einen GPIO-Pin des Mikrocontrollers. Dieses Signal kann mit einem externen Messgerät ausgewertet und anschließend mit der Zeitangabe der Testapplikation verglichen werden.

Tabelle 5.2 zeigt die Ergebnisse der Tests für den Lese- und Schreibdurchsatz sowie für die Dauer eines Initialisierungsvorgangs. Die Ergebnisse erlauben eine erste Einschätzung bezüglich des erreichbaren Durchsatzes. Sehr deutlich zeigt sich, dass der Durchsatz auf allen Karten sowohl beim Schreiben als auch beim Lesen in der gleichen Größenordnung limitiert wird. Beim Lesen liegt die Rate im Bereich zwischen $283\,\mathrm{KiB/s}$ und $296\,\mathrm{KiB/s}$ wohingegen das Schreiben etwas langsamer mit $231\,\mathrm{KiB/s}$ bis $264\,\mathrm{KiB/s}$ ausgeführt wird. Die Abweichung über alle Karten liegt bei der Leserate somit unterhalb von $5\,\%$ und beim Schreiben unterhalb von $15\,\%$. Wesentlich signifikantere Unterschiede zeigen sich bei der Initialisierungsdauer. Im langsamsten Fall benötigt die Karte $381{,}3\,\mathrm{ms}$, während die schnellste Karte bereits nach $7{,}5\,\mathrm{ms}$ einsatzbereit ist.

Karte	**Lesen**	**Schreiben**	**Initialisieren**
1	$293\,\mathrm{KiB\,s^{-1}}$	$263\,\mathrm{KiB\,s^{-1}}$	$14{,}9\,\mathrm{ms}$
2	$288\,\mathrm{KiB\,s^{-1}}$	$231\,\mathrm{KiB\,s^{-1}}$	$23{,}7\,\mathrm{ms}$
3	$290\,\mathrm{KiB\,s^{-1}}$	$261\,\mathrm{KiB\,s^{-1}}$	$10{,}7\,\mathrm{ms}$
4	$283\,\mathrm{KiB\,s^{-1}}$	$259\,\mathrm{KiB\,s^{-1}}$	$7{,}5\,\mathrm{ms}$
5	$296\,\mathrm{KiB\,s^{-1}}$	$264\,\mathrm{KiB\,s^{-1}}$	$381{,}3\,\mathrm{ms}$

Tabelle 5.2.: Schreib- und Leserate (Mittelwert bei Bulk-Übertragung von $10\,\mathrm{MiB}$) und Initialisierungsdauer (Mittelwert über 1000 Initialisierungsvorgänge)

Als eine Ursache für die gleichmäßig limitierenden Geschwindigkeiten wird das Zusammenspiel des Treibers sdcard_spi mit den Low-Level Treibern für SPI vermutet. Der sdcard_spi-Treiber ruft zur Datenübertragung stets die Funktion `spi_transfer_byte` für jedes Byte auf, statt zur Übertragung von größeren zusammenhängenden Blöcken auf die Methode `spi_transfer_bytes` zurückzugreifen. Die Ursache hierfür ist, dass aus Kompatibilitätsgründen als sogenanntes Dummy-Byte bei der Kommunikation mit der Karte 0xFF übertragen werden muss. Der Low-Level SPI-Treiber überträgt bei einer reinen Leseoperation (out-Parameter == NULL) jedoch 0x00 als Dummy-Byte, womit einige Karten nicht funktionieren. Um dieses Problem zu umgehen, werden drei Möglichkeiten betrachtet. Es kann jedes Byte einzeln unter Angabe des Dummy-Bytes übertragen werden, was der momentanen Implementierung entspricht. Diese Variante bietet den Vorteil einer geringen Speicherauslastung, die auf Kosten einer reduzierten Performance erkauft wird.

Alternativ kann ein statischer Speicherbereich im ROM angelegt werden, der mit den Dummy-Bytes vorinitialisiert wird. Dieser kann an die Funktion `spi_transfer_bytes` übergeben werden. Damit wird ein höherer Durchsatz erreicht, allerdings steigt auch der Speicherbedarf.

Als weitere Möglichkeit könnte die Schnittstelle des Low-Level-SPI Treibers um einen Parameter für einen expliziten Dummy-Byte-Wert ergänzt werden. Diese Anpassung der Low-Level-Schnittstelle bietet aus Sicht der Performance sowohl den Vorteil eines besseren Durchsatzes, als auch eine niedrigere Speicherauslastung, da kein zusätzlicher Puffer notwendig ist. Dennoch wird von dieser Lösung abgesehen, da SD-Karten hier einen Spezialfall darstellen, der sich nicht auf die Gestaltung der Schnittstelle des SPI-Treibers auswirken sollte. Weiterhin würden sich z. B. bei der Verwendung verschiedener Geräte an einem Bus weitere Probleme ergeben, da diese nicht gezwungenermaßen die gleiche Konvention bzgl. Dummy-Bytes aufweisen. Als Lösung wird angestrebt, per Präprozessordirektive zwischen der momentanen Implementierung und einer Variante mit statischem Puffer wählen zu können. So kann je nach Anforderung entschieden werden, ob ein höherer Durchsatz oder mehr Speicher benötigt wird.

5.1.2. Stromaufnahme

Der Strombedarf wird mit einem digitalen Sampling-Multimeter (Keithley DMM7510 7,5 Digits) gemessen. Dieses wird auf den $100\,\mathrm{mA}$ Messbereich fixiert um ein automatisches Umschalten des Messbereichs und damit einhergehende Verzögerungen zu vermeiden. Es wird eine Abtastrate von 1000 Samples/s gewählt, da der Mittelwert über einen relativ langen Zeitraum von bis zu mehreren Minuten gemessen wird und mit einer niedrigeren Abtastrate eine höhere effektive Auflösung zur Verfügung steht. Damit nur der relevante Teil des Testablaufs gemessen wird,

werden die über GPIO-Pins herausgeführten Start- und Stop-Signale des Testablaufs als Trigger für das Messgerät verwendet. Dazu werden die GPIO-Pins mit der digitalen I/O-Schnittstelle des Messgerätes verbunden. Zur Steuerung des Messgeräts wird ein einfaches Triggerflow Modell erstellt, das anhand der externen Signalflanken den Messvorgang steuert. Abbildung 5.1 zeigt das Modell zum Starten und Beenden des Messvorgangs. Die Wait-Zustände (1) und (3) blockieren die Ausführung des Flows bis eine steigende, respektive fallende Flanke auftritt. Die erste Digitize-Aktion (2) startet die Messung. Die zweite Digitize-Aktion (4) beendet diese wieder.

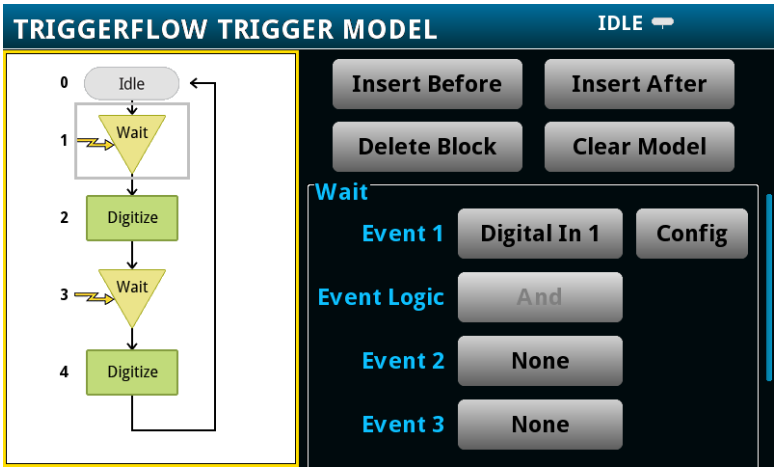


Abbildung 5.1.: Triggerflow Modell zum steuern des Messvorgangs

Tabelle 5.3 zeigt die Messergebnisse zur Leistungsaufnahme. Alle angegebenen Ströme wurden bei einer konstanten Versorgungsspannung von 3,3 V gemessen. Im Unterschied zu dem zuvor erläuterten limitierenden Schreibdurchsatz, zeichnet sich hier ein deutlich dynamischeres Bild ab. Der Strombedarf beim Initialisieren bewegt sich zwischen 11,22 mA und 37 mA. Beim Lesen dehnt sich diese Spanne auf 9,87 mA bis 39,05 mA aus, während Schreibvorgänge zwischen 12,64 mA und 40,35 mA fordern.

Vergleicht man die verschiedenen Zustände, zeigt sich, dass Schreiben meist nicht signifikant mehr Leistung benötigt als Lesen. Bei einer Mehrheit der Karten steigt der Energiebedarf beim Schreiben nur im niedrigen einstelligen Prozentbereich, bei den Karten 4 und 5 aber auch im Bereich von knapp 30 %.

Bei der Gegenüberstellung der Transferoperationen und dem Initialisieren fällt auf, dass das Initialisieren auf einigen Karten weniger, auf anderen Karten mehr Leistung benötigt. In besonderem Maße ist dies bei Karte 5 zu sehen, die beim Initialisieren knapp dreimal so viel Leistung aufnimmt.

Ein Aspekt, den die Messungen zur Leistungsaufnahme genauer beleuchten sollen, ist der auftretende Energieverbrauch im Ruhezustand. Um diesen zu Messen wird eine Testapplikation verwendet, welche die SD-Karte über die auto-init-Funktion initialisiert. Vor Beginn der Messung wird ein Block (512 B) von der Karte gelesen und überprüft, ob die Funktion ordnungsgemäß ausgeführt wird. Dies soll sicherstellen, dass die Kommunikation mit der Karte in der aktuellen Ausführungsinstanz funktioniert.

Die Messergebnisse zu diesem Versuch sind in der Spalte **Idle** zu finden und werfen zunächst Fragen auf. Der benötigte Strom steigt im Vergleich zu den Transferoperationen bei Karte 2 leicht an, bei drei der Karten wird er nur unwesentlich kleiner. Eine starke Abweichung zeigt sich bei Karte 5, welche im Ruhezustand nur noch 93 µA benötigt. Als Ursache für diesen Unterschied wird vermutet, dass der Karten-interne Controller der Karten 1 bis 4 nicht direkt nach dem letzten Lesezugriff in einen sparsameren Modus wechseln kann, da für die Ausführung von internen Operationen meist das Taktsignal der SPI-Schnittstelle benötigt wird[29]. Das Taktsignal ist bei dem sdcard_spi-Treiber allerdings nur aktiv, wenn mit der Karte kommuniziert wird, wie z. B. beim Lesen oder Schreiben von Daten. Mit einer weiteren Messung wird diese Vermutung untersucht. Der einzige Unterschied zur vorherigen Messung ist dabei, dass nach dem initialen Lesen eines Blocks 10 Dummy-Bytes an die Karte übertragen werden, wodurch auch das Taktsignal kurzzeitig weiter aktiv bleibt.

Die Ergebnisse dieser zweiten Messung zum Ruhezustand sind in der Spalte **Idle (2)** zu finden. Die Karten 1 bis 4 erreichen durch die Anpassung einen wesentlich sparsameren Betriebszustand mit einer Stromaufnahme zwischen 144 µA und 460 µA. Ebenfalls fällt auf, dass die Karte 5 nun mehr Strom aufnimmt als zuvor. Als Erklärung hierfür erscheinen zwei Möglichkeiten plausibel: Der interne Controller wechselt durch die zusätzlichen Taktzyklen in einen Zustand, in dem er Verwaltungsoperationen o. A. auf der Karte ausführt. Außerdem ist es möglich, dass die Pins, die zur Kommunikation genutzt werden, in einen anderen Konfigurationsmodus wechseln, wodurch Leckströme an den Logik-Pins entstehen. Letzteres wird als wahrscheinlicher erachtet, da sich bei mehrfacher Messdurchführung teilweise deutliche Schwankungen zeigen. Weitere Messungen könnten hier detailliertere Ergebnisse liefern. Es wird allerdings davon abgesehen, da hiermit zwar ein genaueres Verständnis der Karteneigenschaften in bestimmten Situationen nachvollzogen werden kann, sich daraus aber keine allgemeingültigen Aussagen ableiten lassen. Mehrfache Messdurchläufe zeigen bei den Karten 1 bis 4 keine signifikanten Abweichungen.

Die Karte im Ruhezustand zu belassen, wirkt sich nur dann positiv auf die Energiebilanz aus, wenn die Karte regelmäßig innerhalb eines bestimmten Zeitintervalls wiederverwendet wird. Die Obergrenze für dieses Zeitintervall wird in Tabelle 5.4 in der Spalte $\mathbf{t_{Idle}}$ gelistet. Diese

Nr.	Initialisieren	Lesen	Schreiben	Idle	Idle(2)
1	20,22 mA	22,36 mA	24,34 mA	19,03 mA	389 µA
2	30,24 mA	27,45 mA	27,16 mA	29,94 mA	460 µA
3	37 mA	39,05 mA	40,35 mA	39,76 mA	144 µA
4	11,22 mA	14,54 mA	19,38 mA	10,08 mA	291 µA
5	29,77 mA	9,87 mA	12,64 mA	93 µA	741 µA

Tabelle 5.3.: Strombedarf verschiedener Karten-Modi

Nr.	Initialisieren	$\mathbf{t_{Idle}}$
1	0,99 mW s	0,77 s
2	2,36 mW s	1,56 s
3	1,32 mW s	2,76 s
4	0,28 mW s	0,29 s
5	37,46 mW s	122,07 s

Tabelle 5.4.: Energieverbrauch beim Initialisieren und daraus resultierende Zeitintervalle für den Ruhezustand

Grenze ergibt sich aus dem Energieverbrauch der Karte beim Initialisieren und der kleinstmöglichen Leistungsaufnahme der Karte im Ruhezustand (**Idle** für Karte 5, sonst **Idle(2)**). Nur für Karte Nr. 5 liegt dieses Zeitintervall in einer Größenordnung, die für Messszenarien mit Duty-Cycle-Betreibsmodus interessant erscheint, was hauptsächlich daran liegt, dass die Karte zur Initialisierung vergleichsweise viel Energie benötigt. Sobald seltener auf die Karte geschrienen wird, als das Zeitintervall $\mathbf{t_{Idle}}$ vorgibt, ist es sparsamer die Karte vollständig abzuschalten und vor einem Schreibvorgang neu zu initialisieren. Da für die Messdatenspeicherung in Feldtests auf den verwendeten Mikrocontrollern nur ein relativ kleiner Puffer bereitsteht, ist Karte 4 eindeutig zu bevorzugen. Für diese ist das Zeitintervall $\mathbf{t_{Idle}}$ nur 0,29 s lang. Dieses Zeitintervall ist für viele Messszenarien zu klein, um eine Menge von 512 B oder gar 5 KiB anzusammeln. Es wird daher davon ausgegangen, dass das Abschalten der Karte im Regelfall energieeffizienter ist, als die Karte im Ruhezustand zu belassen. Weitere Versuche könnten zeigen, ob sich dieser Sachverhalt unter Verwendung des expliziten Energiesparmodus von SD-Karten verändert. Die SD-Spezifikation garantiert allerdings nicht, dass dieser im SPI-Modus verwendet werden kann. Daher wird vorerst davon abgesehen.

5.1.3. Optimierungen

Nachfolgend werden einige Optimierungen genannt, mit denen der Treiber in Zukunft verbessert werden kann. Die genannten Punkte sollten auch bei der Konzipierung von Anwendungen berücksichtigt werden, die den Treiber verwenden – insbesondere wenn Energieverbrauch und Performanz von erhöhter Wichtigkeit sind. Um den durchschnittlichen Energieverbrauch zu minimieren, können verschiedene Optimierungen angewandt werden. Eine Optimierung besteht darin, die Karte in den Energiesparmodus zu versetzen oder sie vollständig abzuschalten und nur für tatsächliche Schreibvorgänge zu aktivieren. Weiterhin sollte der SPI-Bus in dem schnellstmöglichen Modus verwendet werden, um die aktiven Phasen so kurz wie möglich zu halten. Die Daten sollten im RAM gepuffert werden und nur auf die Speicherkarte geschrieben werden, wenn eine bestimmte Datenmenge erreicht wurde. Um die Karte bezüglich ihrer Lebensdauer zu schonen und den Overhead für den Controller innerhalb der Karte zu minimieren, sollte diese Datenmenge einem vielfachen der Sektorgröße entsprechen, die bei einem Schreibzugriff durch interne Vorgänge der Karte gelöscht wird. Je nach Version der SD-Karte wird diese Größe durch das Feld , SECTOR_SIZE im CSD-Register oder durch AU_SIZE im SD Status Register definiert und unterscheidet sich zwischen verschiedenen Karten. In der Praxis bewegen sich SECTOR_SIZE und AU_SIZE zwischen 512 Byte und 64 MB, womit das Puffern eines vollständigen Sektors im RAM der MCU in vielen Fällen nicht praktikabel ist. Je nach Anwendung kann es daher sinnvoll sein, bestimmte Kartenmodelle für den Einsatz auszuwählen. Für den Schreibvorgang sollte bevorzugt die *Multiple Block Write* Operation verwendet werden, um auch den Overhead für die Datenübertragung gering zu halten. Die SD-Spezifikation nennt zur weiteren Optimierung des Durchsatzes beim Schreiben die Verwendung des SET_WR_BLK_ERASE_COUNT Kommandos (ACMD23) vor dem Schreiben mehrerer neuer Blöcke. Weiteres Optimierungspotential besteht bei der Festlegung der Timeouts für Kommandos. Die derzeitige Lösung verwendet als Timeout stets eine feste Anzahl an Versuchen, die unternommen werden, bevor ein Kommando als nicht erfolgreich interpretiert wird. Je nach Art des Kommandos steht dazu ein eigenes Literal zur Verfügung, das nötigenfalls angepasst werden kann. Versuche mit den in Tabelle 5.1 gelisteten Karten legen nahe, dass diese sich zwischen verschiedenen Karten erheblich unterscheiden, was zu einer langsameren Übertragungsgeschwindigkeit führen kann. Zur weiteren Optimierung sollten die Timeouts dynamisch anhand der Karteneigenschaften festgelegt werden.

5.2. Test: Dateisystem

Um die Funktion des FatFs Moduls zu testen wird, wie bereits für den Treiber sdcard_spi, eine interaktive Testapplikation implementiert. Mit dieser können beispielsweise Dateien gelistet, gelesen oder geschrieben werden. Um zu überprüfen, ob dies ordnungsgemäß funktioniert, werden die Dateiinhalte nach verschiedenen Dateioperationen mit den Erwartungswerten abgeglichen. Weiterhin wird überprüft, ob Änderungen an den Dateien (über einem PC oder RIOT) auf dem jeweils anderen System übereinstimmend angezeigt werden. Für die vfs-Integration des FatFs-Moduls wird eine Testapplikation bereitgestellt, die auf Basis des in RIOT häufig verwendeten pexpect[1] eine automatisierte Lösung zum Testen der verschiedenen Dateisystemoperationen bereitstellt. Damit kann die gesamte Logging-Komponente von vfs bis SD-Karte auf einem Mikrocontroller getestet werden. Auf einem PC wird statt der SD-Karte ein Dateisystemabbild verwendet. In einem abschließenden Versuch wird über mehrere Stunden hinweg das in Abbildung 4.5 vorgestellte Messmodul mit einem Messintervall von $0{,}6\,\mathrm{s}$ betrieben. Dabei werden alle aufgezeichneten Messwerte in eine CSV-Datei auf die SD-Karte geschrieben. Im Rahmen dieser Tests konnte ebenfalls kein fehlerhaftes Verhalten hervorgerufen werden.

[1] https://github.com/pexpect/pexpect

6. Aufbau der Testplattform

Nachfolgend wird beschreiben wie die zuvor entwickelten Einzelkomponenten in ein Gesamtsystem integriert werden. Weiterhin wird dargelegt, welche ergänzenden Komponenten zur Durchführung des Feldtests nötig sind und wie diese bereitgestellt werden.

Die Plattform wird in verschiedenen Konfigurationen mit Sensoren zum Messen von Umweltdaten bestückt. Als Sensor für Temperatur und Luftfeuchtigkeit wird der DHT22 Sensor ausgewählt. Dieser misst die Temperatur mit einer Auflösung von $0{,}1\,°\mathrm{C}$ und einer Genauigkeit von $\pm 0{,}1\,°\mathrm{C}$. Die relative Luftfeuchtigkeit wird mit $0{,}1\,\%$ Auflösung und einer Genauigkeit von $\pm 2\,\%$ angegeben[32].

Des Weiteren werden mit dem Bosch BMP280 und dem NXP MPL3115A2 zwei verschiedene Luftdrucksensoren eingesetzt, die mit einer absoluten Genauigkeit von $\pm 1\,\mathrm{hPa}$ (Auflösung bis zu $0{,}0016\,\mathrm{hPa}$)[33] respektive $\pm 0{,}4\,\mathrm{kPa}$ (Auflösung bis zu $0{,}25\,\mathrm{Pa}$)[34] spezifiziert sind.

Die zuvor genannten Sensoren verwenden alle passive Messverfahren, die mit Werten zwischen $1\,\mathrm{mA}$ und $2\,\mathrm{mA}$ bei $3\,\mathrm{V}$ eine vergleichsweise geringe Leistungsaufnahme aufweisen. Zum Testen der Plattform mit einem leistungsstärkeren Verbraucher wird ein SDS011 Feinstaubsensor von Nova eingesetzt, der mit einem aktiven Messverfahren arbeitet. Der Betrieb des integrierten Lüfters und der Lasermessvorrichtung benötigt $70\,\mathrm{mA}$ bei $5\,\mathrm{V}$. Die relative Genauigkeit des Sensors wird mit einem Maximalwert von $15\,\%$ und $\pm 10\,\mu\mathrm{g/m^3}$ angegeben[35].

Bis auf den Feinstaubsensor existiert für alle Sensoren bereits ein RIOT-Treiber. Für diesen wird daher ein Treiber implementiert, der die Messdaten über die UART Schnittstelle ausliest. Der Treiber des DHT22 Sensors wird angepasst, da die vorhandene Implementierung keinen stabilen Betrieb in einer Applikation mit mehreren Threads erlaubt. Die Ursache hierfür ist, dass der Treiber zur Flankenerkennung auf Polling setzt. Dadurch können auftretende Interrupts dazu führen, dass eine Flanke nicht gemessen wird und das System in einem Deadlock stecken bleibt. Durch die Umstellung auf eine Interrupt-gesteuerte Flankenerkennung kann dieses Problem umgangen werden.

6.1. Sensorknoten

Die Firmware des Sensorknotens basiert auf der aktuellen Entwicklungsversion von RIOT (master, 19.03.2018)[1] und beinhaltet die Softwareunterstützung für die zuvor vorgestellten Einzelkomponenten.

Zur Verbindung aller Hardwarekomponenten wird eine Hauptplatine entworfen, die eine flexible Anbindung externer Sensorik erlaubt. Auf der Platine sind bis zu sieben paarweise Schraubklemmen-Anschlüsse angebracht um die zuvor vorgestellten Module und verschiedene externe Sensoren anzuschließen. Abbildung 6.1 zeigt die Montageplatte, mit den Umrissen der einzelnen Module und dem Schaltplan der Verbindungsplatine. Die zwei doppelten Steckleisten mit jeweils 38 Pins dienen als Sockel zum Aufstecken des nucleo-l476 Boards. Die Hauptplatine bietet neben den zum nucleo-l476 durchgeschleiften Anschlüssen zwei per GPIO schaltbare Anschlüsse zur Spannungsversorgung externer Sensoren. Weiterhin ist ein Spannungsteiler angebracht um die Spannung des PV-Moduls mit dem internen ADC Messen zu können. Tabelle 6.1 führt entsprechend der Grafik die Funktionen der nummerierten Anschlüsse auf.

Nr.	**Funktion**
1	PV-Modul Spannungsmessung
2	Ladeelektronik Ein Aus
3	3.3V Ausgang
4	VIN Eingang
5	GND
6	I^2C-2 SCL (PC0)
7	I^2C-2 SDA (PC1)
8	WKUP1 (PA0)
9	WKUP5 / UART1 RX (PC5)
10	Analog 1 / GPIO (PA1)
11	GND SW
12	VCC SW
13	WKUP5 / UART1 RX (PC5)
14	UART1 TX (PC4)

Tabelle 6.1.: Nummerierung Schraubanschlussfunktionen der Basisplatine

In Tabelle 6.2 wird für jede Komponente aufgeführt, wie sie mit den anderen Komponenten verbunden ist. Tabelle 6.2f zeigt die Verbindung des Messmoduls mit Hauptplatine, Ladeelektronik und Superkondensator. Die Anschlusskonfiguration der Ladeelektronik an PV-Modul,

[1] commit: 565341c9fdbbe07a5f433cbda210bd6a805f31ea

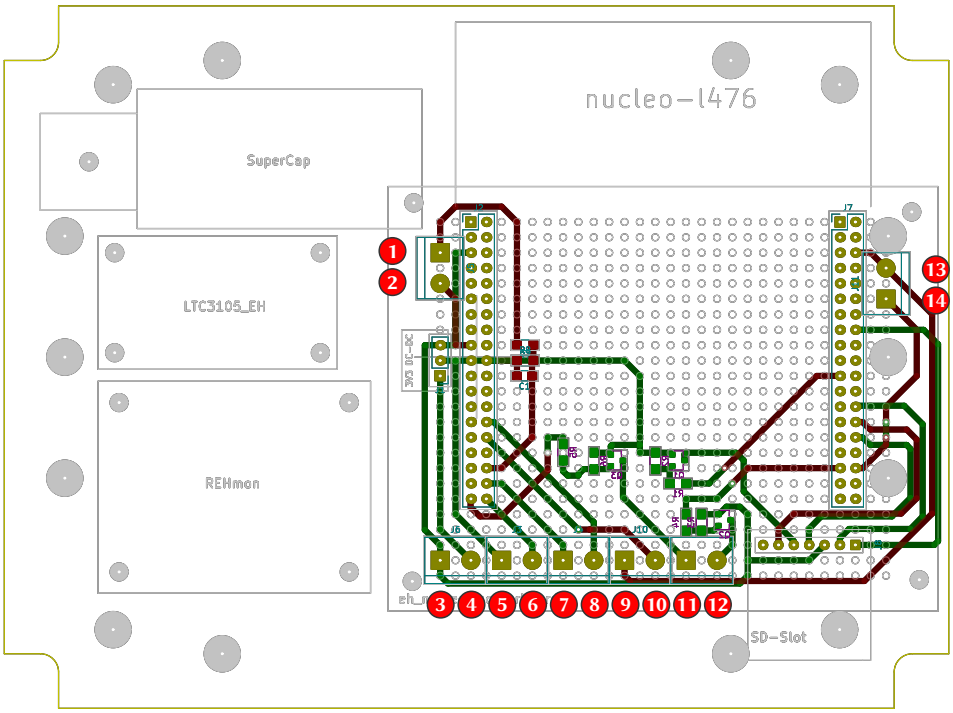


Abbildung 6.1.: Montageplatte für Einzelkomponenten mit Layout der Verbindungsplatine

Messmodul, Superkondensator und Hauptplatine wird in Tabelle 6.2g gelistet. Die Tabellen 6.2a, 6.2b, 6.2c und 6.2d beinhalten entsprechend die Anschlussbelegung für die Sensoren BMP280, MPL3115A2, DHT22 und SDS011 in genannter Reihenfolge. Bis auf den Feinstaubsensor SDS011 können alle Sensoren mit der 3,3 V Versorgungsspannung betrieben werden, die auch für den Mikrocontroller genutzt wird. Der Feinstaubsensor erfordert eine Spannung von 5 V. Dazu wird diesem entsprechend Tabelle 6.2e ein S7V8A DC-DC Wandler von Pololu[2] vorgeschaltet.

6.2. Basisstation

Das Empfangen und Speichern der per Funk übertragenen Daten erfordert den Einsatz einer Basisstation mit einem kompatiblen Funkmodul. Hierzu wird ein Raspberry Pi 3 Model B mit dem 802.15.4 Modul von openlabs verwendet, welches auch auf den Sensorknoten eingesetzt

[2] https://www.pololu.com/product/2118

BMP280	
Anschluss	**Verbunden mit**
VCC	Hauptplatine (3)
GND	Hauptplatine (11)
SCL	Hauptplatine (6)
SDA	Hauptplatine (7)

(a)

MPL3115A2	
Anschluss	**Verbunden mit**
VIN	Hauptplatine (3)
GND	Hauptplatine (11)
SCL	Hauptplatine (6)
SDA	Hauptplatine (7)

(b)

DHT22	
Anschluss	**Verbunden mit**
VCC	Hauptplatine (3)
GND	Hauptplatine (11)
DATA	Hauptplatine (10)

(c)

SDS011	
Anschluss	**Verbunden mit**
5V	S7V8A (VOUT)
GND	Hauptplatine (5)
TX	Hauptplatine (13)
RX	Hauptplatine (14)

(d)

S7V8A	
Anschluss	**Verbunden mit**
GND	Hauptplatine (5)
VIN	Hauptplatine (12)
VOUT	SDS011 (5V)

(e)

Messmodul	
Anschluss	**Verbunden mit**
PV+	PV-Modul (+), Hauptplatine (1)
GND	PV-Modul (-), Superkondensator (-)
VOUT	Messmodul (IN-)
SHDN	Hauptplatine (2)

(f)

Ladeelektronik	
Anschluss	**Verbunden mit**
PV+	PV-Modul (+), Hauptplatine (1)
GND	PV-Modul (-), Superkondensator (-)
VOUT	Messmodul (IN-)
SHDN	Hauptplatine (2)

(g)

Tabelle 6.2.: Anschlusskonfigurationen der Einzelkomponenten

wird. Auf dem Raspberry Pi wird das Betriebssystem Raspbian Stretch Lite in der Release Version vom 2017-11-29 verwendet. Dieses basiert auf dem Desktop Betriebssystem Debian Stretch, verzichtet im Gegensatz dazu aber auf eine grafische Desktop-Umgebung. Um das Funkmodul als 6LoWPAN Gerät zu konfigurieren, werden die wpan-tools[3] in Version 0.8 installiert. Die Netzwerkkonfiguration soll nach einem Neustart automatisch wiederhergestellt werden. Hierzu wird ein systemd-Service konfiguriert[4]. In der Datei /etc/default/lowpan werden die folgenden Einstellungen festgelegt:

```
1 CHN="26"
2 PAN="0x23"
3 MAC="42:94:7A:AD:4F:C1:9A:88"
4 ACKREQ=1
```

Mit einem weiteren systemd-Service wird der lowpan0 Netzwerkschnittstelle beim Systemstart eine feste IP-Adresse zugewiesen. Damit in dem bereitgestellten Netz automatisch IPv6 Adressen an die Sensorknoten vergeben werden, wird auf dem Raspberry Pi der Router Advertisement Daemon radvd[5] installiert. Dieser Dienst wird ebenfalls automatisch über einen systemd-Service gestartet.

Für die Zeitsynchronisation der Sensorknoten wird ein NTP-Dienst installiert. Dieser verwendet als Zeitreferenz den Zeitserver der HAW-Hamburg. Die IP-Adresse des Zeitservers wird der Basisstation dabei automatisch per DHCP übermittelt.

Für den Fernzugriff auf die Basisstation wird ein SSH-Server eingerichtet, der ausschließlich die Authentifizierung über ein Public-Key Verfahren erlaubt. Da die Basisstation eine global erreichbare IP-Adresse besitzt, wird mit fail2ban[6] ein einfaches Intrusion Prevention System eingerichtet, das IP-Adressen nach mehrmals fehlgeschlagenen Anmeldeversuchen blockiert.

6.2.1. Übertragung und Speicherung der Sensordaten

Zur Kommunikation zwischen Sensorknoten und Basisstation werden CoAP Nachrichten verwendet. Eine CoAP Nachricht enthält dabei stets ein Objekt im JSON (JavaScript Object Notation) Format. Jedes dieser Objekte beinhaltet einen Zeitstempel des Sensorknotens und zusätzlich Datenwerte in variabler Anzahl. Die CoAP Nachrichten werden stets als *confirmable* (CON) übertragen, wobei das Warten auf eine Antwort mit einem Timeout von 3 s festgelegt ist. Wird innerhalb dieser Zeit kein *acknowledge* (ACK) empfangen, wird die unbestätigte

[3] https://github.com/linux-wpan/wpan-tools(Zugriff 26.05.2018)
[4] https://github.com/riot-makers/wpan-raspbian(Zugriff 26.05.2018)
[5] https://github.com/reubenhwk/radvd.git
(commit 7488976216dabc4c9baf41e16884da0341e14fbd)
[6] https://www.fail2ban.org/wiki/index.php/Main_Page

Übertragung protokolliert und kein weiterer Übertragungsversuch vorgenommen. Für die Messdatenaufzeichnung wird eine Serverapplikation in Python implementiert. Diese besteht aus einem CoAP Server auf Basis der Bibliothek aiocoap[7] und einer SQLite Datenbank zur persistenten Speicherung der Daten. Beim Start des Servers werden mehrere CoAP-Ressourcen registriert, die per POST-Request eingehende Daten entgegennehmen. Für jede Ressource wird eine Tabelle in der Datenbank angelegt, falls diese nicht bereits existiert. Jede dieser Tabellen wird initial mit Spalten für Empfangszeitpunkt, Absender und Rohdaten versehen. Für jedes Datenelement innerhalb eines empfangenen JSON-Objekts wird ebenfalls bei Bedarf eine eigene Spalte angelegt. Die Speicherung der Objektbestandteile in separaten Spalten dient dazu spätere Abfragen zu vereinfachen.

[7] https://github.com/chrysn/aiocoap

7. Feldtest

Im folgenden Teil werden die Rahmenbedingungen und der Versuchsaufbau des angesetzten Feldtests beschrieben. Ziel des Feldtests ist die Demonstration der Funktionstüchtigkeit des Gesamtsystems. Es soll gezeigt werden, inwiefern das Energy-Harvesting System erfolgreich eingesetzt werden kann, um Messwerte von Umweltsensoren und zum eigenen Energieverbrauch aufzuzeichnen und per Funk zu übertragen. Außerdem soll der Einsatz unter Realbedingungen eventuelle Probleme aufdecken und zeigen wo weitere Optimierungen erforderlich sind. Nachfolgend wird dazu der praktische Aufbau des Feldtests beschrieben und anschließend der Ablauf des Feldtests vorgestellt.

Zunächst werden die Sensorknoten in fünffacher Ausführung nach dem in Abschnitt 6.1 geschilderten Aufbau zusammengebaut. Abbildung 7.1 zeigt ein einzelnes Exemplar vor dem Einlegen der SD-Karte und dem Anschließen von Solarzelle sowie Sensoren. Die fünf Sensorknoten werden anschließend in drei verschiedenen Konfigurationen mit Sensoren ausgestattet. Die unterschiedlichen Konfigurationen sollen zum einen dazu dienen verschiedene Umweltparameter aufzuzeichnen. Zum anderen soll durch die verschiedenen Varianten vermieden werden, dass bisher unentdeckte Fehler in einem Sensortreiber, ähnlich dem in Kapitel 6 beschriebenen Deadlock-Problem im DHT22 Treiber, zu einem vollständigen Ausfall auf allen Knoten führen. Tabelle 7.1 listet die Sensorbestückungen für alle Boxen.

Box-ID	**Sensoren**
1	BMP280
2	BMP280
3	DHT22
4	DHT22
5	SDS011, MPL3115A2

Tabelle 7.1.: Sensorbestückung der verschiedenen Knoten

Für den Einsatz im Freien werden die Sensorknoten zum Schutz vor Witterung in ein wasserdichtes (IP65) Gehäuse[1] mit Abmessungen von knapp $165\,\mathrm{mm} \times 125\,\mathrm{mm} \times 75\,\mathrm{mm}$ (LxBxH)

[1] Hammond RP1245 (http://www.hammondmfg.com/pdf/RP1245.pdf)

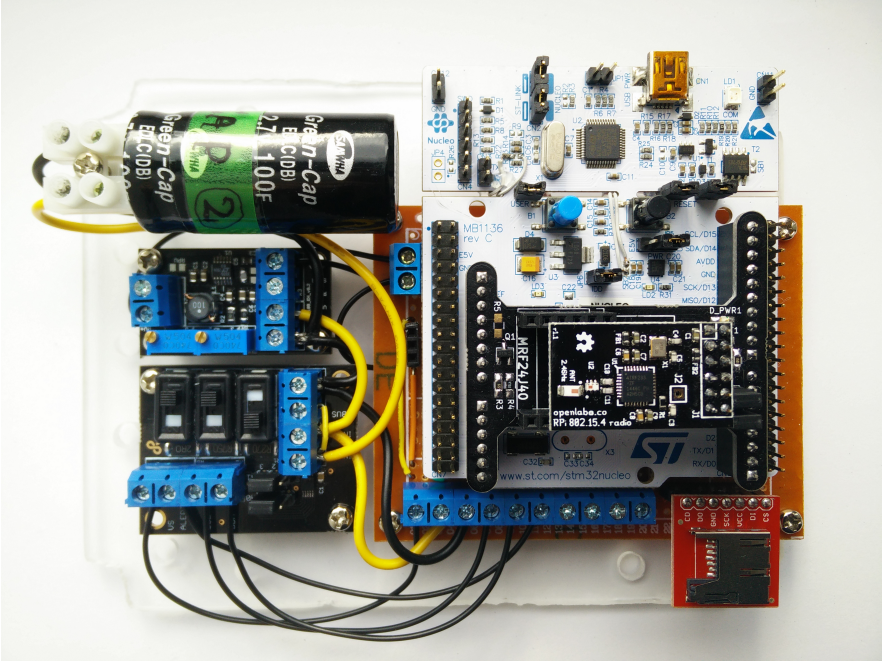


Abbildung 7.1.: Sensorknoten auf Montageplatte

eingebaut. Die Montageplatte der Sensorknoten mit allen Komponenten lässt sich als Einheit einsetzen und mit Schrauben am Boden befestigen. Das PV-Modul wird mit Silikon vor Wasser geschützt an dem Deckel angebracht. Die Sensoren werden auf der Gehäuseaußenseite befestigt und durch ein abgewinkeltes Kunststoffrohr[2] vor Regen geschützt. Der Feinstaubsensor wird in zwei miteinander verbundenen Rohrbögen[3] untergebracht.

Abbildung 7.2 zeigt alle Sensorboxen bevor der Deckel montiert wird.

Die Sensorboxen werden für den Feldtest auf dem Dach des HAW-Gebäudes am Berliner Tor 7 aufgestellt. Abbildung 7.3 zeigt eine schematische Draufsicht der Versuchsanordnung. Oben links ist die Basisstation (RPI) zu sehen, die Sensorboxen sind mit ihrer eindeutigen Nummerierung eingezeichnet. Positioniert werden die Sensorboxen in unterschiedlichen Entfernungen und Ausrichtungen zur Basisstation. Das jeder Sensorknoten die Basistation direkt erreichen kann, entspricht die Netzwerkkonfiguration einer Sterntopologie. Multi-Hop Übertragungen werden in diesem Szenario nicht behandelt, da unter Betrachtung der deutlich höheren Komplexität kein signifikanter Mehrwert hinsichtlich der Funktionsprüfung der einzelnen Sen-

[2] HT-Bogen DN 40, 87° z. B.: https://www.bauhaus.info/ht-rohre/marley-ht-bogen/p/13625004 (Zugriff 22.05.2018)
[3] HT-Bogen DN 75, 87° z. B.: https://www.bauhaus.info/ht-rohre/marley-ht-bogen/p/13625028 (Zugriff 22.05.2018)

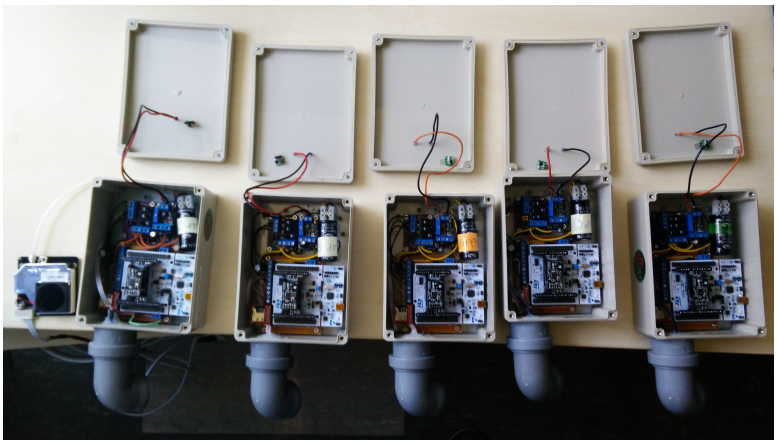


Abbildung 7.2.: Sensorboxen vor Montage des Deckels

sorknoten zu erwarten ist. Weiterhin werden dadurch negative Auswirkungen durch einzelne fehlerhafte Knoten auf den Gesamtaufbau reduziert.

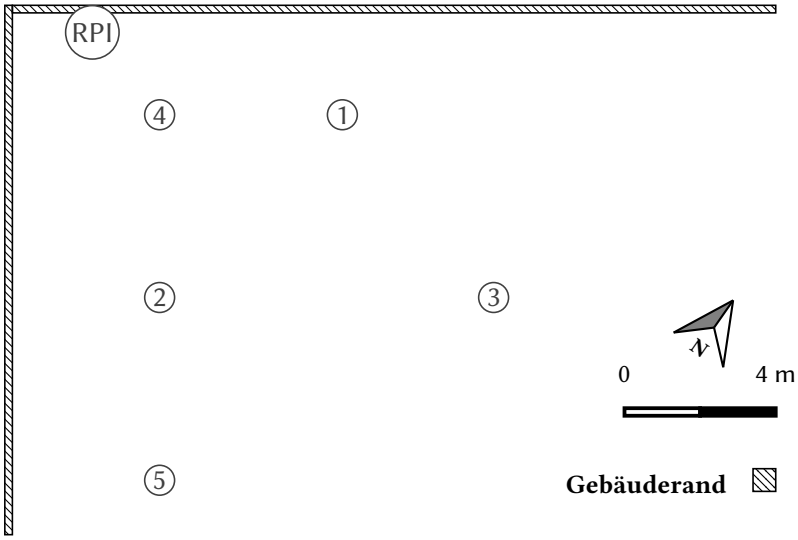


Abbildung 7.3.: Aufstellung von Basisstation und Sensorboxen (Draufsicht)

Die Basisstation wird in ein wetterfestes, beheiztes Gehäuse eingebaut das Steckdosen sowie einen Netzwerk-Switch beherbergt. Abbildung 7.4 zeigt das Gehäuse am Rand des Dachs an einer Stange montiert.

Abbildung 7.4.: Wetterfestes Gehäuse zur Unterbringung der Basisstation

Abbildung 7.5.: Box-5 mit Rohrbogen zur Unterbringung des Feinstaubsensors

7.1. Ablauf

Alle Sensorboxen werden mit einer Firmware bespielt, die den gleichen Ablauf implementiert. Unterschiede existieren lediglich in dem Code zum Auslesen der jeweiligen Sensoren. Der Ablauf ist dabei strikt zyklisch organisiert. Nach dem initialen Start des Systems wird zunächst versucht die interne Zeitreferenz per SNTP zu synchronisieren. Ist der Zeitserver nicht erreichbar, wird nach einem konfigurierbaren Timeout ein erneuter Versuch gestartet. Die Superkondensatoren werden vor dem Aufstellen aufgeladen, damit die Sensorknoten sich möglichst schnell mit der Basisstation synchronisieren können. Die Bestimmung des Duty-Cycles folgt einem sehr einfachen und konservativen Ansatz um nach Möglichkeit zu vermeiden, dass der Knoten die vordefinierte Untergrenze für die verfügbare Energiemenge erreicht. Zur Berechnung des Duty-Cycles wird die zur Verfügung stehende Energiemenge stets auf einen Zeitraum von 18 Stunden allokiert. Dieser Wert basiert auf der Annahme, dass an jedem Tag zumindest für 6 Stunden ausreichend Sonnenlicht zum Aufladen des Superkondensators bereitstehen. In jedem Fall wird nach dem Aufwachen des Sensorknotens aus einer Schlafphase überprüft, ob die Energiereserve höher als ein vordefinierter Mindestwert ist. Damit wird sichergestellt, dass die vorhandene Energiemenge ausreicht, um einen Wachzyklus inklusive aller benötigten Schritte auszuführen und anschließend ordnungsgemäß in den Energiesparmodus zu wechseln. Ist dieser Schwellwert nicht erreicht, wird umgehend wieder in den Energiesparmodus gewechselt.

Nach einer vordefinierten Anzahl an Zyklen wird eine Messung des Ladestroms initiiert. Die Messung findet statt, während der Sensorknoten sich im Schlafmodus befindet. Nach Fertigstellung der Messung wird der Sensorknoten per asynchronem Interrupt von dem Modul aufgeweckt. Daraufhin werden die Daten aus dem Modul ausgelesen und in einen gleitenden Mittelwert eingerechnet, der persistent in einem Backup-Register abgelegt wird. Ebenso wird zyklisch der Energieverbrauch während des Betriebs gemessen. Die Ergebnisse dieser Messung fließen ebenfalls in einen gleitenden Mittelwert ein und werden zusammen mit den anderen Daten in regelmäßigen Abständen an die Basisstation übertragen. Zusätzlich werden regelmäßig vollständige Stromverbrauchskurven mit variierenden Abtastraten aufgezeichnet und lokal auf der SD-Karte abgespeichert. Die unterschiedlichen Abtastkonfigurationen sollen später bezüglich der Eignung für Detailanalysen einzelner Zyklen untersucht werden. Aufgrund der Dateigröße wird bei diesen Daten eine Übertragung per Funk vermieden, um die Auslastung des Funkmediums zu reduzieren und Energie zu sparen. Durch die lokale Speicherung soll außerdem die allgemeine Praxistauglichkeit der Speicherkomponente für den Einsatz im Feld gezeigt werden.

7.2. Auslesen und Aufbereitung der Daten

Nach Beendigung des Feldtests werden die Sensorboxen vom Dach entfernt und die Daten der einzelnen SD-Karten auf einen PC kopiert. Im Anschluss wird gleichermaßen eine Kopie der Datenbankdatei der Basisstation erstellt. Zur einfacheren Auswertung der Datenbank wird eine Tabelle zur Zuordnung von IP-Adressen und Nummerierung der Sensorboxen hinzugefügt.

8. Ergebnisse

Nachstehend werden die Ergebnisse aus dem vorangegangenen Feldtest beschrieben. Dazu wird vorerst erläutert welche Daten für die Auswertung zur Verfügung stehen und im Anschluss wird der grobe Überblick ausgeweitet auf eine Auseinandersetzung mit den jeweils gemessenen Größen. Untergliedert wird die Behandlung der Daten in die Abschnitte Umweltdaten, Clock-Drift und Paketverluste und Energiedaten. Letzterer beinhaltet dabei einen gesonderten Unterpunkt in dem die Leitungs-Traces vorgestellt und diskutiert werden. Die Interpretation und Analyse von Auffälligkeiten einzelner Aufzeichnungen wird direkt in den jeweiligen Abschnitten dargelegt.

Die Sensorboxen zeigen nach dem Feldtest keinerlei Beschädigungen, abgesehen von Schmutzablagerungen durch Staub und Regenwasser. Die in den Boxen untergebrachten Feuchtigkeitsindikatoren zeigen keine Verfärbung in dem Messfeld für eine Luftfeuchtigkeit von über 60 %. Dadurch kann angenommen werden, dass eine ausreichende Wasserdichtigkeit zu jedem Zeitpunkt gegeben war.

8.1. Datensätze

Eine erste Bestandsaufnahme der Messdaten zeigt, dass je Sensorbox teilweise sehr unterschiedliche Datenmengen vorliegen. Abbildung 8.1 zeigt für jede Box die zur Basisstation übertragenen Datensätze pro Tag. Ein einzelner Eintrag korrespondiert jeweils mit einem per CoAP übertragenen Datenpaket, welches wiederum mehrere in JSON-kodierte Werte beinhaltet. Insgesamt enthält die Datenbank eine Menge von knapp 160 MB Messdaten. Die Werte von Box-1 und Box-5 weisen verglichen mit den anderen Boxen erhebliche Unregelmäßigkeiten auf. Die allgemein geringere Anzahl von Box-5 ist aufgrund der deutlich höheren Leistungsaufnahme des verbauten Feinstaubsensors nachvollziehbar, die vollständigen Ausfälle deuten aber darauf hin, dass das System nicht wie vorgesehen gearbeitet hat. Während von Box-1 nur an einem Tag kein Datensatz vorliegt, trifft dies bei Box-5 für 16 Tage zu. Über alle Boxen hinweg zeigt sich eine deutliche Abweichung am siebten Versuchstag (1. Mai 2018). Die Paketanzahl weist hier einen sehr stark reduzierten Wert auf. Plausible Ursachen hierfür wären Fehler in der Firmware oder äußere Störeinflüsse, die sich auf alle Boxen auswirken. Ein Beispiel hierfür

ist eine langfristige Störung des Funkkanals oder ein Ausfall der Basisstation. Aufgrund der Dauer wird eine Störung des Funkkanals als unwahrscheinlich eingestuft. Eine Überprüfung der Basisstation lässt weder durch eine Unterbrechung in der uptime-Statistik, noch im Service-Log des CoAP Servers eine Betriebsunterbrechung des Empfängers erkennen. Eine Störung des Empfangsmoduls auf Layer-1 oder Layer-2 kann damit allerdings nicht ausgeschlossen werden. Die bei allen Boxen reduzierte Paketanzahl am ersten und letzten Versuchstag lässt sich mit dem Auf- bzw. Abbau zur Mittagszeit begründen, da die Boxen dadurch nur jeweils für die Hälfte des Tages im Einsatz waren. Bei Betrachtung der Werte von Box-2 bis Box-4 lässt sich aus den relativ niedrigen Schwankungen die Vermutung ableiten, dass die Aufwachzeit tagsüber häufig dem einprogrammierten Mindestintervall von 10 s entspricht. Dadurch bewegt sich die Anzahl an Übertragungen häufig an einem oberen Limit. Entsprechend der höheren Energieverfügbarkeit fällt der Großteil an Daten tagsüber an, was in Kombination mit der Limitierung dazu führt, dass identische Auswirkungen von variierender Witterung und Sonnenintensität nur stark abgeschwächt erkennbar sind, wie z. B. die Anhebung am 13.05 und die leichte Senkung am 18.05 zeigen.

Zusätzlich zu der Datenbankdatei stehen rund 250 MB Daten von den SD-Karten der Sensorknoten in Form von CSV-Dateien zur Verfügung. Diese beinhalten Zusammenfassungen der Energiemessungen in zweistündiger Taktung und Traces der Leistungsaufnahme in unterschiedlichen zeitlichen Auflösungen, die detaillierte Analysen des Energieverbrauchs ermöglichen. Abbildung 8.2 zeigt für die beiden Datensätze wieder eine tageweise Aufstellung der Anzahl an Elementen. Auch die Trace-Aufzeichnung unterliegt größtenteils den zuvor beschriebenen Mustern, wobei die Schwankungen bei Box-1 weniger stark sind und sich ein stetigeres Verhalten zeigt. Das lässt die Vermutung zu, dass die in Abbildung 8.1 gezeigte reduzierte Anzahl auf Paketverluste im Übertragungsweg zurückzuführen ist. Eine plausible Erklärung für die geringere Menge an Traces könnte dementsprechend in Verzögerungen durch mehrmalige Übertragungsversuche auf Layer-2 bzw. der durch Paketverluste entstehenden längeren Wartezeit auf ACKs der CoAP Pakete zu finden sein.

Die Zahlen zu Box-5 in Abbildung 8.2 zeigen im Vergleich mit den Werten zur Funkübertragung in Abbildung 8.1 ebenso eine höhere Aktivität. Die auch hier vorhandenen vollständigen Ausfälle deuten aber auf eine Fehlfunktion hin.

Weiterhin bestehen bleibt auch die gemeinsame Lücke am 1. Mai, was untermauert, dass die Ursache dafür kein Fehler in der Funkübertragung ist.

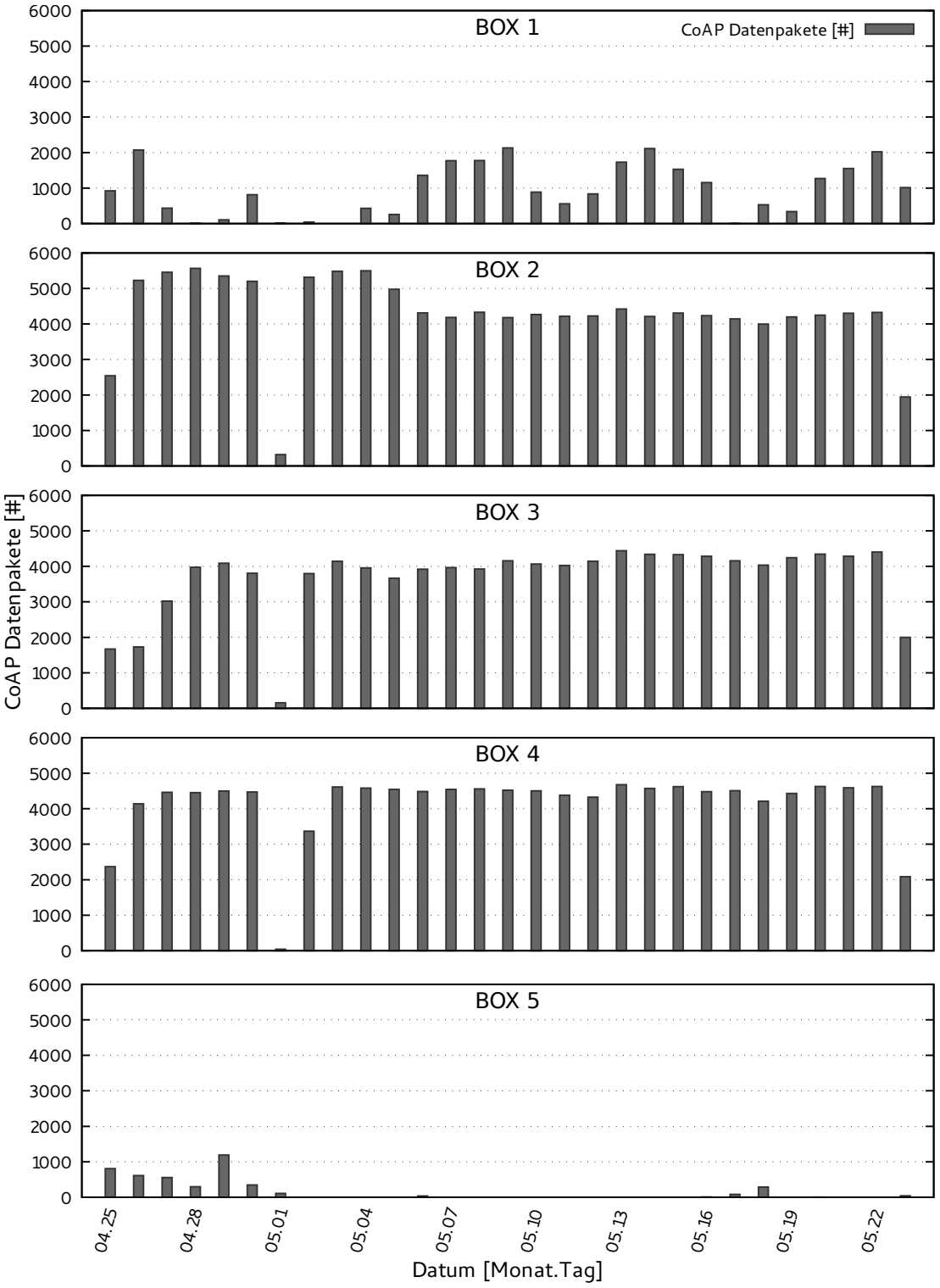


Abbildung 8.1.: Anzahl per Funkübertragung empfangener Datensätze pro Tag

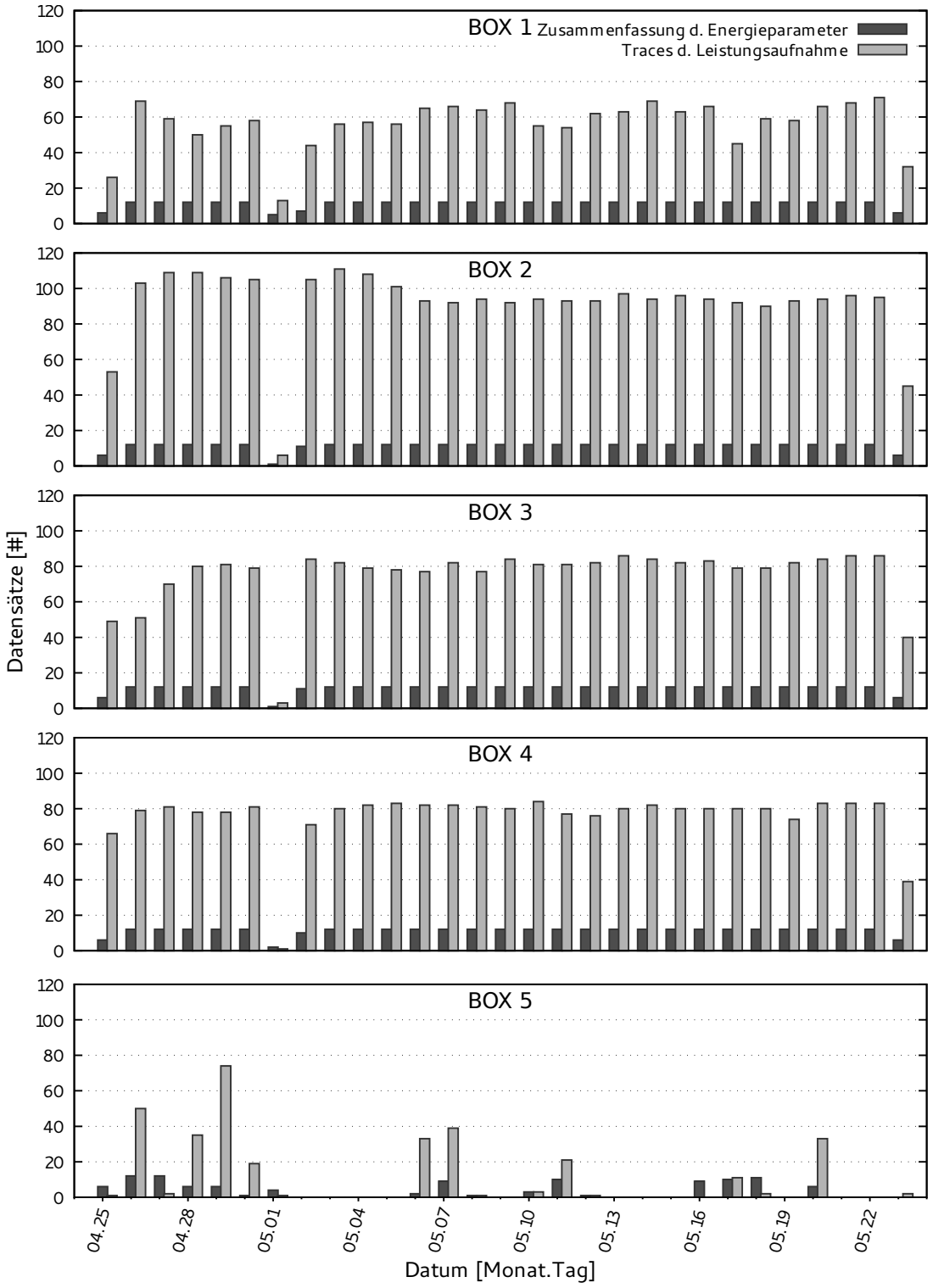


Abbildung 8.2.: Anzahl auf der SD-Karte gespeicherter Datensätze

8.2. Umweltdaten

Nach dem Überblick zum Umfang der gesammelten Datensätze werden im Folgenden einzelne gemessene Größen betrachtet. Abbildung 8.3 zeigt den Temperaturverlauf über den gesamten Versuchszeitraum. Die Werte entspringen unterschiedlichen Sensoren, entsprechend den in Kapitel 7 (Tabelle 7.1) gelisteten Sensorboxkonfigurationen. Die Messwerte der Sensorboxen zeichnen einen charakteristischen Temperaturverlauf des Tag-Nacht-Zyklus und bilden dabei ein größtenteils deckungsgleiches Bild. Lediglich die Werte von Box-5 weisen bei niedrigeren Temperaturen einen klar sichtbaren negativen Versatz auf (siehe. z. B. bei 05.18 in Abbildung 8.3). Der Grund hierfür könnte sowohl im abweichenden Standort als auch in der Genauigkeit des Sensors bestehen. Letzteres wird aufgrund der Deckungsgleichheit der anderen Werte (trotz unterschiedlichem Aufstellungsort) und der relativ geringen Unterschiede in der Positionierung, als wahrscheinlicher angesehen.

Durch die Anbringung der Sensoren direkt an der Sensorbox und deren Aufstellung in möglichst direkter Sonneneinstrahlung liegen die Tageshöchstwerte bei bis zu $40{,}1\,°C$ (BOX-4, gemessen am 09.05.2018 um 13:14 Uhr (UTC)). Die niedrigste gemessene Temperatur stammt von Box-3 und liegt bei einem Wert von $1{,}5\,°C$ (gemessen am 02.05.2018 um 03:50 Uhr (UTC)). Die signifikanteste Schwankung innerhalb eines Tages ist mit einer Differenz von $32{,}52\,°C$ am 04.05.2018 zu finden (Box-2).

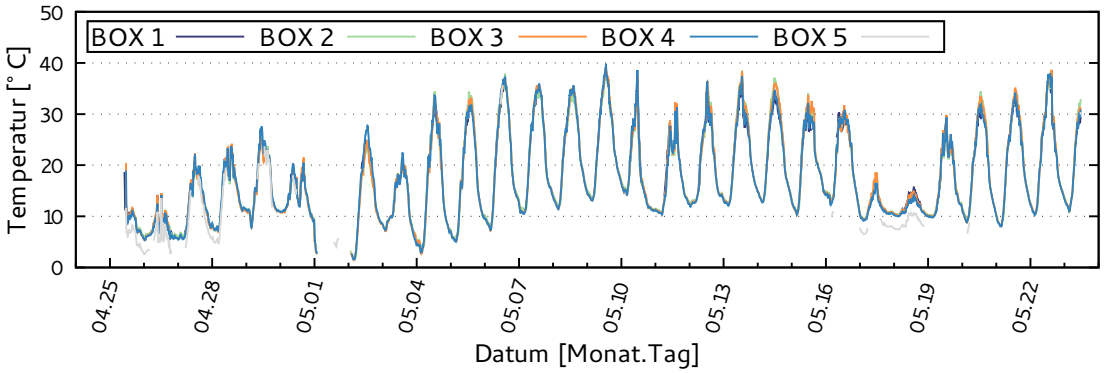


Abbildung 8.3.: Verlauf der Temperatur über den gesamten Versuchszeitraum

Eine weitere gemessene Größe ist die relative Luftfeuchtigkeit. Nur Box-3 und Box-4 sind mit einem Feuchtigkeitssensor ausgestattet. Die in Abbildung 8.4 dargestellten Werte zeigen wie bereits die Temperaturmessungen einen größtenteils deckungsgleichen Verlauf, der ebenfalls klar den Tag-Nacht-Zyklus erkennen lässt. Die relative Feuchte verläuft dabei komplementär zur Temperatur wodurch sich der Zusammenhang zwischen Sättigungskonzentration und

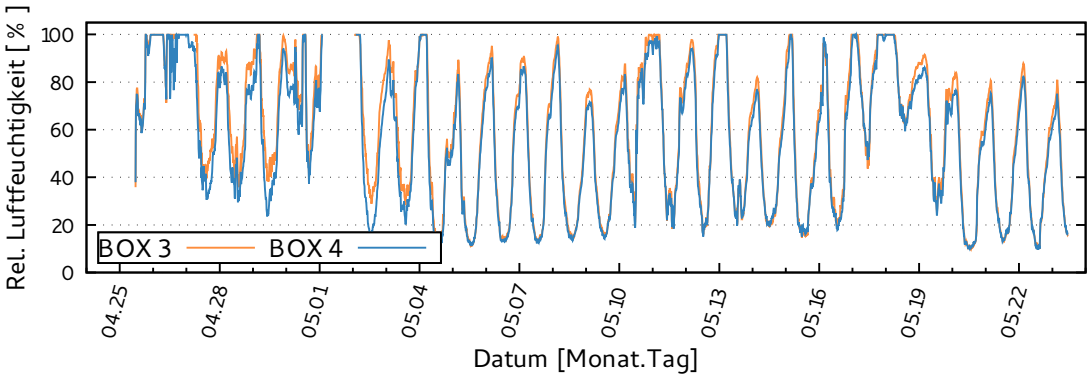


Abbildung 8.4.: Verlauf der relativen Luftfeuchtigkeit über gesamten Versuchszeitraum

Temperatur darstellen lässt – warme Luft kann mehr Feuchtigkeit aufnehmen als kalte Luft. Abgesehen von dem meist deckungsgleichen Verlauf sind z. B. am 02.05 oder am 18.05 Abweichungen erkennbar, die sich beispielsweise durch unterschiedliche Verdunstungsraten im direkten Umfeld der Sensorknoten erklären lassen. Abbildung 8.5 zeigt Box-4 (links) in einer entsprechenden Situation, bei der stehendes Wasser um den Sensorknoten die Verdunstungsrate lokal beeinflussen kann.

Drei der Sensorboxen sind mit einem Luftdrucksensor ausgestattet. Abbildung 8.6 zeigt deren Messwerte. Im Gegensatz zu Temperatur und Luftfeuchtigkeit hängt dieser Wert nicht direkt von der Sonneneinstrahlung ab und zeigt daher keinen zyklischen Verlauf. Die vorhandenen Messwerte von Box-1 und Box-2 sind durch die Verwendung des gleichen Sensormodells (BMP280) auch hier weitestgehend deckungsgleich, während die wenigen vorhandenen Daten von Box-5 wieder einen negativen Versatz aufweisen.

Abbildung 8.5.: Stehendes Wasser bei Box-4 nach einem Niederschlag

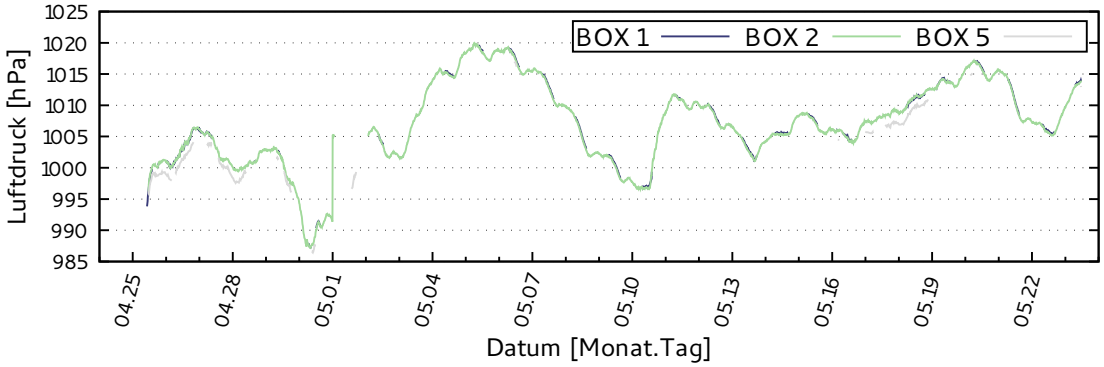


Abbildung 8.6.: Verlauf des Luftdrucks über gesamten Versuchszeitraum

8.3. Clock-Drift und Paketverluste

Die durchgeführten Messungen beleuchten neben Umweltdaten andere Metriken, die Relevanz für weitere Deployment-Szenarien besitzen. Die Implementierung der RTC-Zeitsynchronisation, die in Abschnitt 4.10 beschrieben wird, besitzt einen Mechanismus zur Driftkorrektur, um Abweichungen in der Frequenz des Taktgebers zu korrigieren. Bei Vorabtests hat sich gezeigt, dass die aktuelle RTC-Implementierung starken Abweichungen unterliegt, die unter anderem durch Anwendungsverhalten beeinflusst werden.[1] Dieses Verhalten äußert sich darin, dass die Zeit auf einmalig synchronisierten Geräten z. B. in Abhängigkeit von der Aufwachhäufigkeit stark driftet. Um diesen Aspekt weiter zu untersuchen wird der Verlauf der Dirft-Korrekturwerte ausgewertet. Abbildung 8.7 zeigt die Entwicklung des Drifts über die Zeit. Die Werte entsprechen den jeweils im RTC-Peripheriebaustein eingestellten Kalibrierungswerten. Entsprechend ist an den diskreten Schritten die Auflösung der Kalibrierungsfunktion sichtbar. Bis auf eine Ausnahme bewegen sich die gemessenen Werte im Rahmen der Hardwaregenauigkeit, die mit einem Basiswert von $20\,\mathrm{PPM}$ zuzüglich eines Frequenzkoeffizienten von $0{,}04\,\mathrm{PPM}/°\mathrm{C}^2$ und einem Alterungswert von $\pm 3\,\mathrm{PPM}/\mathrm{Jahr}$ ($\pm 5\,\mathrm{PPM}$ im ersten Jahr) angegeben ist.

Vergleicht man das Gesamtbild, sind Ähnlichkeiten zwischen den einzelnen Verläufen erkennbar. Da auch hier wiederkehrende Muster im Rhythmus der Tageszeit auftreten, kann davon ausgegangen werden, dass die Temperatur den wesentlichen Faktor für die gleichförmigen Änderungen im Drift darstellt.

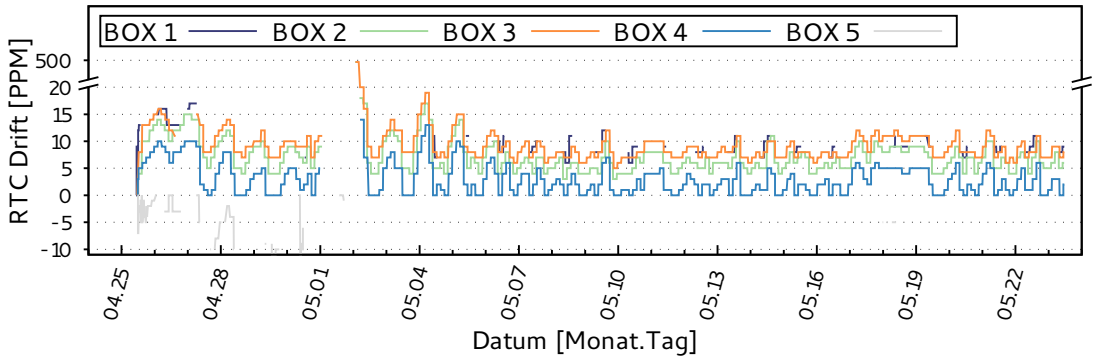


Abbildung 8.7.: RTC Driftwerte

In dem Feldtest wurden zum Übertragen der Daten ausschließlich *Confirmable* (CON) CoAP-Nachrichten verwendet. Wird nach einem Timeout von $3\,\mathrm{s}$ kein *Acknowledgement* (ACK) empfangen, wird dieses Ereignis mitgezählt und keine erneute Übertragung versucht. Ab-

[1] https://github.com/RIOT-OS/RIOT/issues/8746

bildung 8.8 listet die Anzahl nicht bestätigter CoAP Pakete für jeden Versuchstag auf einer logarithmischen Skala. Die Ursache für die zuvor in Abschnitt 8.1 dargelegte, reduzierte Anzahl an empfangenen Datensätzen von Box-1, lässt sich hiermit in Form erhöhter Übertragungsfehler identifizieren. Anzumerken ist, dass die Werte lediglich in Form eines fortlaufenden Zählers erfasst wurden. Für Tage an denen keine Daten hierzu empfangen wurden, ist der Wert nicht eindeutig festzulegen und wird dem ersten Tag zugerechnet, an dem wieder ein Paket empfangen wurde. Bei Box-1 ist dies an den Werten vom 29.04, 04.05 und 18.05 zu erkennen. Es kann angenommen werden, dass die Werte in diesen Fällen etwa zur Hälfte dem Vortag zuzurechnen sind, legt man den restlichen Verlauf zugrunde. Vergleicht man die Positionierung der Knoten (Abbildung 7.3) mit den Paketverlusten, lässt sich kein eindeutiger Zusammenhang mit der Entfernung zwischen den Boxen und der Basisstation erkennen. Box-1 befindet sich z. B. deutlich näher an der Basisstation als Box-3, weißt aber einen deutlich höheren Paketverlust auf. Ebenso betragen die Paketverluste von Box-3 und Box-4 relativ ähnliche Werte, physikalisch sind die Entfernungen zur Basisstation aber sehr unterschiedlich.

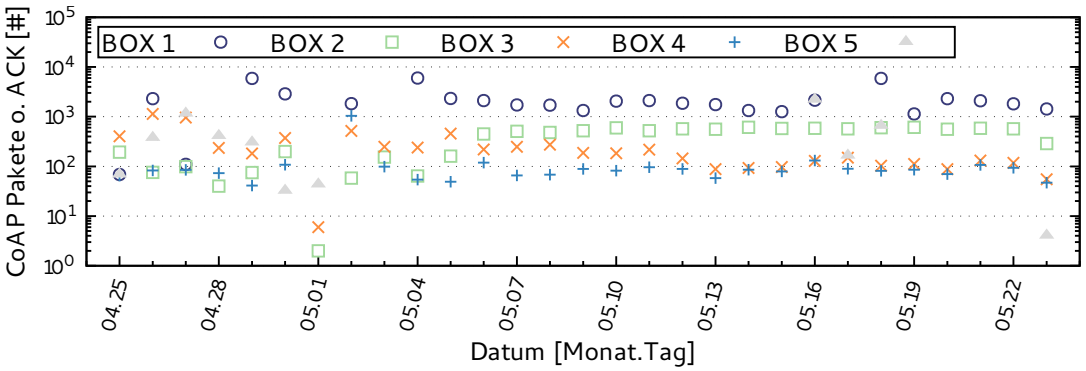


Abbildung 8.8.: Anzahl an CoAP Übertragungsfehlern

8.4. Energiedaten

Ein wesentliches Ziel dieser Arbeit ist, zu demonstrieren wie in einem realistischen Szenario mit der energieautarken Plattform Messdaten zum eigenen Energieverbrauch aufgezeichnet werden können. Abbildung 8.9 zeigt den Spannungsverlauf an Solarzelle und Superkondensator und die gespeicherte Energiemenge. Die Messwerte von Box-1 und Box-5 sind, wie bereits aus der vorangegangenen Betrachtung der Paketanzahl ersichtlich, sehr lückenhaft. Die Messwerte von Box-2 bis Box-4 geben Einblick in die Energiebilanz zu verschiedenen Tageszeiten. So lässt sich anhand der Leerlaufspannung des PV-Moduls (Upv) der Intensitätsverlauf der Sonnen-

einstrahlung erkennen. Aus dem Spannungsverlauf am Superkondensator ist zu erkennen, dass der Superkondensator tagsüber stets vollständig aufgeladen werden kann und keine Überschreitung der Ladeschlussspannung auftritt. Dementsprechend arbeitet die Ladeelektronik wie vorgesehen. Die in Abschnitt 8.1 aufgestellte Vermutung, dass die Sensorknoten tagsüber häufig mit dem kürzesten Schlafintervall betrieben werden, kann hiermit bestätigt werden. Diese Schlussfolgerung ergibt sich aus der Feststellung, dass die Spannung zu Sonnenphasen fast ausschließlich an der Obergrenze liegt und der Tatsache, dass das kürzeste Intervall bereits ab einer Schwellenspannung von $50\,\mathrm{mV}$ unterhalb der Maximalspannung ($2{,}65\,\mathrm{V}$) angewendet wird. Hieraus lässt sich weiterhin ableiten, dass das System z. B. auch mit einem kleineren PV-Modul eine vergleichbare Performanz erreichen kann, oder mit dem verbauten Modul eine höhere Systemauslastung realisierbar ist.

Der Verlauf der verfügbaren Energiemenge (Ea) zeigt, dass der Energiespeicher von Box-3 durchschnittlich tiefer entladen wird als bei Box-2 und Box-4. Ursachen hierfür sind eine höhere Selbstentladung oder ein Eigenverbrauch, der höher ist, als der Sensorknoten gemessen hat. Möglich ist auch, dass die Sonneneinstrahlung auf dem Sensorknoten erst zu einem späteren Zeitpunkt eine ausreichend hohe Intensität aufweist, um den Ladevorgang erneut zu starten.

Eine Auffälligkeit in den Daten weist Box-3 auf. Die Leerlaufspannung des PV-Moduls bricht meist kurz vor dem Zeitpunkt rasch ein, zu dem sich der Wert, verglichen mit den anderen Messreihen dem Höchstwert nähern sollte. Gegen Ende der Sonnen-intensiven Phase steigt der Wert anschließend wieder ebenso schlagartig an. Als Ursache hierfür wird ein Defekt des PV-Moduls vermutet, der bei einer erhöhten Temperatur zu einem Kurzschluss einer einzelnen Zelle innerhalb des Moduls führt. Eine weitere Möglichkeit besteht darin, dass die hochohmig durchgeführte ADC-Messung z. B. durch andere Signale gestört wird. Das deterministische Auftreten des Fehlers spricht jedoch dagegen.

Abbildung 8.10 zeigt die Abweichungen von Box-3 am 27.05.2018. Der Spannungsverlauf weist einen steilen Spannungseinbruch auf, dem ein geradliniger Verlauf und anschließend wieder ein steiler Anstieg folgt. Zum Vergleich stellt Abbildung 8.11 die Messwerte desselben Tages von Box-2 dar.

Ein weiteres Detail, das sich in beiden Graphen erkennen lässt, ist der zeitlich relativ grob aufgelöste Anstieg der PV-Spannung. Diese ist an dem kantigeren Verlauf zu Beginn des Tages zu sehen, welcher in den zunächst langen Schlafphasen begründet ist. Mit dem Anstieg der verfügbaren Energie (Ea) wird dieses Intervall wieder erkennbar kleiner.

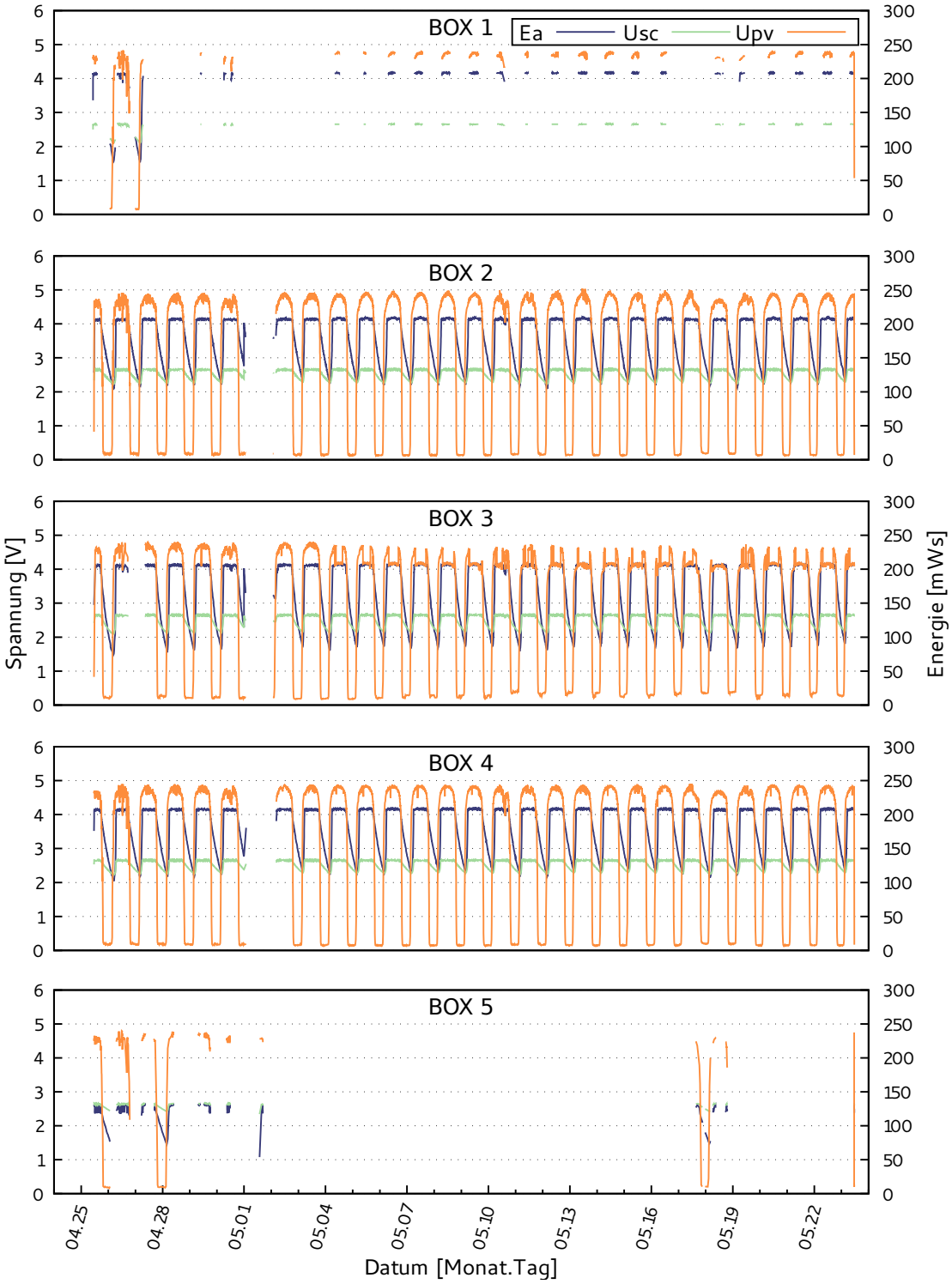


Abbildung 8.9.: Spannungsverlauf an Solarzelle und Superkondensator und Verfügbare Energiemenge

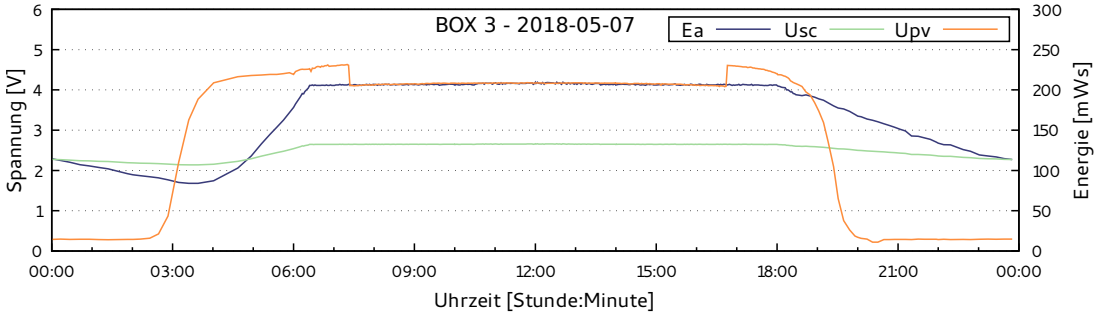


Abbildung 8.10.: Tagesverlauf der verfügbaren Energiemenge und Spannung an Superkondensator und Solarzellen

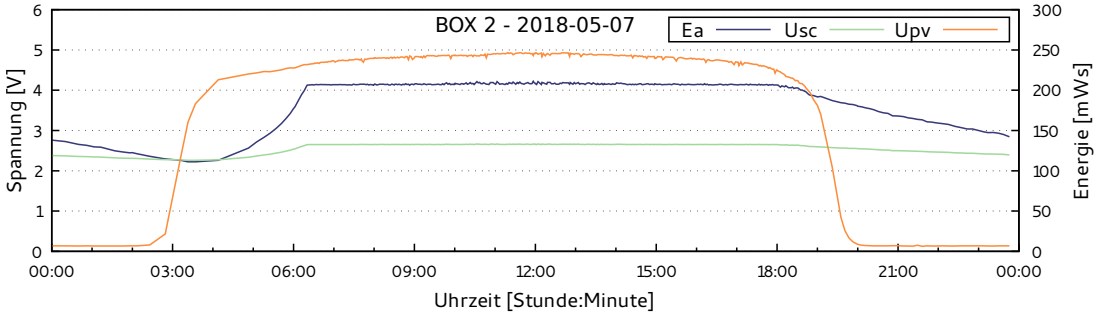


Abbildung 8.11.: Tagesverlauf der verfügbaren Energiemenge und Spannung an Superkondensator und Solarzellen

Damit das System zur Laufzeit seinen Duty-Cycle möglichst genau anpassen kann, ist die Bestimmung des eigenen Energieverbrauchs erforderlich. Dieser wird von dem Messmodul zur Laufzeit aufgezeichnet und hängt hauptsächlich vom Anwendungsverhalten, der Dauer von Übertragungsvorgängen und der momentanen Effizienz einzelner Komponenten ab.

Abbildung 8.12 zeigt die durchschnittliche Leistungsaufnahme der Sensorknoten im Zeitverlauf über den gesamten Versuchszeitraum. Die Werte beziehen sich dabei ausschließlich auf den Anteil, in dem der Sensorknoten aktiv ist. Der Ladevorgang wird dabei zum Messen unterbrochen damit der Eigenverbrauch unabhängig von der Energiegewinnung betrachtet werden kann. Die Leistungsaufnahme bewegt sich größtenteils im Bereich zwischen $75\,\mathrm{mW}$ und $90\,\mathrm{mW}$. Zu Beachten ist, dass Box-3 und Box-4 mit dem gleichen Sensormodell ausgestattet sind, während an Box-2 ein anderer Sensor angeschlossen ist. Dieser Sachverhalt ist auch an den Verbrauchswerten abzulesen, die bei Box-3 und Box-4 deutlich näher beieinander liegen. Der Verlauf zeigt abermals Tageszeit-bedingte Muster, die sich mit hoher Wahrscheinlichkeit durch den jeweiligen Ladezustand des Superkondensators und der damit verbundenen Effizienz des Spannungswandlers begründen lassen.

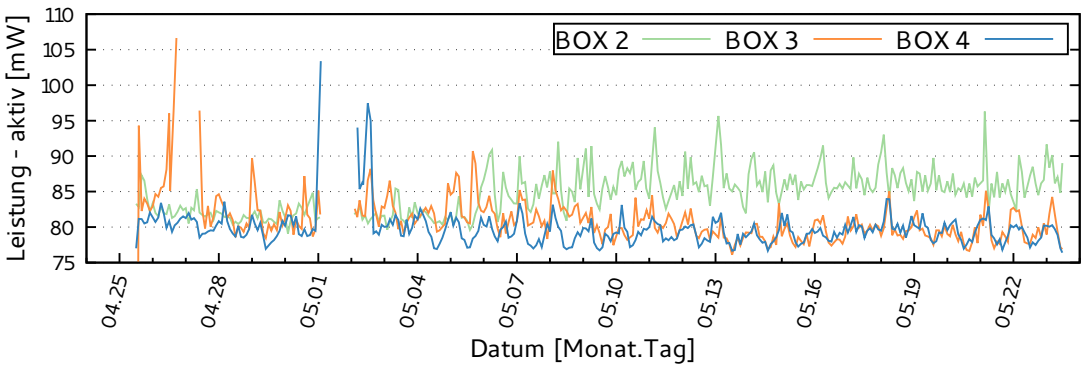


Abbildung 8.12.: Durchschnittliche Leistungsaufnahme, Sensorknoten im Betrieb

Die bidirektionale Funktionalität des Messmoduls ermöglicht neben der Bestimmung des Verbrauchs auch die Messung von negativen Werten, um den Ladestrom quantifizieren zu können.

Analog zu Abbildung 8.12 zeigt Abbildung 8.13 die durchschnittliche Leistungsaufnahme des Sensorknotens, wenn dieser nicht aktiv ist. Ein negativer Wert entspricht demnach der Leistung, mit der der Superkondensator aufgeladen wird. Zu beachten ist hierbei, dass die Werte jeweils den Gesamtverbrauch des Systems widerspiegeln und damit auch den Eigenverbrauch des Messmoduls miteinbeziehen. Messungen mit einem Multimeter (Keithley DMM7510) ergeben für das Messmodul einen Eigenverbrauch von $1{,}3\,\mathrm{mW}$. Findet gerade keine Messung statt,

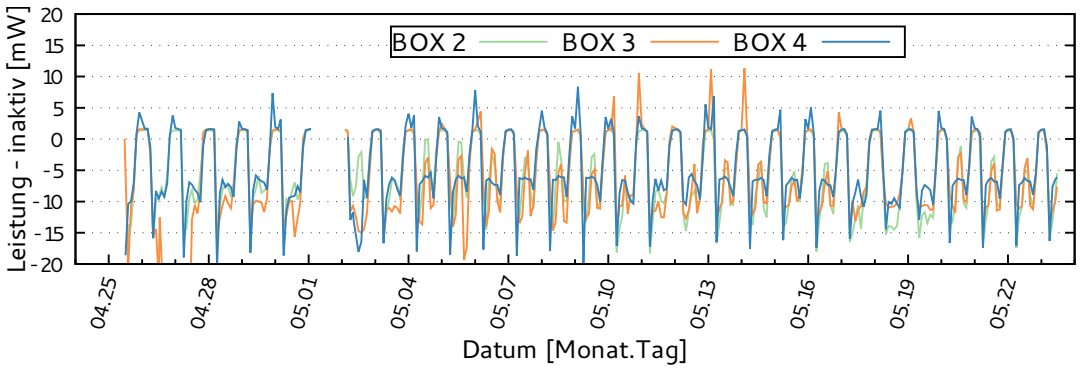


Abbildung 8.13.: Durchschnittliche Leistungsaufnahme, Sensorknoten inaktiv

fällt dieser Verbrauchsanteil weg und kann ebenso in den Superkondensator geladen werden. Die nach unten gerichteten Spitzen zum Tagesbeginn lassen in Abbildung 8.13 erkennen, dass der Ladevorgang zu Beginn mit der höchsten Rate zwischen 15 mW und 20 mW lädt und dann gegen die Mittagszeit auf ein mittleres Niveau zwischen 5 mW und 10 mW abfällt. Anschließend erhöht sich die Ladeleistung teilweise wieder, um gegen Ende des Tages in einen Zustand zu wechseln, in dem keine Energie mehr gewonnen werden kann.

Dieser Verlauf widerspricht klar der intuitiven Erwartung, dass die Ladeleistung vom Sonnenaufgang bis zur Mittagszeit ansteigt und anschließend in einem gleichförmigen aber umgekehrten Verlauf bis zum Sonnenuntergang abnimmt. Um diesen Unterschied zu erklären muss genauer betrachtet werden, durch welche Faktoren die Messung beeinflusst wird.

Die maximale Ladeleistung hängt im vorliegenden Fall unter anderem vom Spannungsunterschied zwischen der am Superkondensator anliegenden Spannung und der eingestellten Ladeschlussspannung ab. Ebenso wird die Effizienz der Ladeelektronik durch die Eingangs- und Ausgangsspannung beeinflusst und fällt ab, wenn die Eingangsspannung oberhalb oder unterhalb des Optimums liegt. Bei einer Ausgangsspannung von 3 V nennt das Datenblatt des LTC3105 eine Effizienz von ca. 90 %, wenn die Eingangspannung ca. 2,5 V beträgt. Die Effizienz kann aber bis unter 50 % fallen, wenn die Eingangsspannung bei 5 V liegt[27]. Weiterhin wirkt sich auf den Verlauf aus, wenn mehr Energie bereitsteht als verbraucht wird und der Superkondensator dadurch regelmäßig vollständig aufgeladen wird. In diesem Fall wird im Anschluss an eine kurze Verbrauchsphase nur solange ein Ladestrom gemessen, bis der Energiespeicher wieder voll ist. Für die restliche Zeit der Messdauer ist dann aufgrund der automatischen Beendigung des Ladevorgangs durch die Ladeelektronik kein Stromfluss mehr messbar. Dieser Sachverhalt zeigt sich an dem Verlauf in Form des Plateaus, das sich nach dem initial höheren Ladestrom einstellt.

Damit lässt sich schlussfolgern, dass die Höhe des aktuellen Ladestroms im Betrieb gemessen werden kann, dieser aber nicht zwingend dem theoretisch verfügbaren maximalen Ladestrom entspricht, da dieser deutlich höher liegen kann. Um mit dem Messaufbau zu jeder Zeit bestimmen zu können, wie hoch der aktuell maximal verfügbare Ladestrom ist, muss dafür gesorgt werden, stets genug Ladekapazität im Superkondensator vorzuhalten.

8.4.1. Traces

Die Möglichkeit den aktuellen Energieverbrauch eines Sensorknotens im Feld zu untersuchen, kann für viele Anwendungen ausreichend sein, um Energiequelle und -Speicher zu dimensionieren, oder eine Feinabstimmung von System- und Anwendungsparametern vorzunehmen. Je nach Anwendungsfall kann eine Detailanalyse einzelner Abläufe hilfreich sein, um Ursachen für Abweichungen im Energieverbrauch zu identifizieren oder Auswirkungen von Parameteranpassungen sichtbar zu machen. Nachfolgend werden zu diesem Thema Trace-Aufzeichnungen der Leistungsaufnahme betrachtet, die eine Untersuchung einzelner Sensor-Zyklen erlauben.

Das entwickelte Messmodul unterstützt eine flexible Einstellmöglichkeit für Abtastrate und anschließende Mittelwertbildung. Die hier vorgestellten Traces verwenden verschiedene Kombinationen dieser Parameter. Tabelle 8.1 listet die jeweiligen Konfigurationen und die daraus resultierende effektive Abtastrate.

Nr.	Messdauer Strom	Messdauer Spannung	Mittelwerte	eff. Abtastrate
1	1,1 ms	1,1 ms	64	7 Hz
2	1,1 ms	588 µs	16	37 Hz
3	332 µs	332 µs	16	94 Hz
4	588 µs	588 µs	4	213 Hz
5	588 µs	332 µs	1	1087 Hz
6	204 µs	204 µs	1	2451 Hz

Tabelle 8.1.: Verwendete Abtastkonfigurationen des Messmoduls

Nachfolgend werden exemplarisch einige der 8957 aufgezeichneten Traces genauer beschrieben, um zu zeigen, welche Konfigurationen für welchen Anwendungsfall geeignet sind und welche Informationen daraus gewonnen werden können. Zudem sollen die Beispiele illustrieren, wann Probleme mit der Selbstvermessung auftreten können.

Abbildung 8.14 zeigt einen Trace von Box-1 mit der zeitlich relativ groben Konfiguration 1. Der Trace wurde aufgezeichnet am 04.05.2018. Zu sehen sind die Spannung und die Leistungsaufnahme über den Zeitraum einer Aktivphase (22450. Messvorgang) des Sensorknotens.

Vertikale Marker zeigen die Zeitpunkte verschiedener Ereignisse, wie Start und Ende von Sensormessung und Funkübertragung. Hervorstechende Merkmale sind die sehr konstant verlaufende Spannung und die steil ansteigende Leistungsaufnahme zu Beginn des Sendevorgangs. Der Sendevorgang erzeugt einen Leistungsverlauf in Form eines Sägezahns. Der Nachteil der niedrigen Abtastrate ist zu Beginn der Sensormessung zu erkennen. Durch die Dauer der einzelnen Abtastschritte fließt der Verbrauch der Sensormessung nur in Form einer einzelnen Abtastung ein, was am zeitlichen Versatz zwischen Start der Sensormessung und dem ersten erhaltenen Leistungswert sichtbar ist.

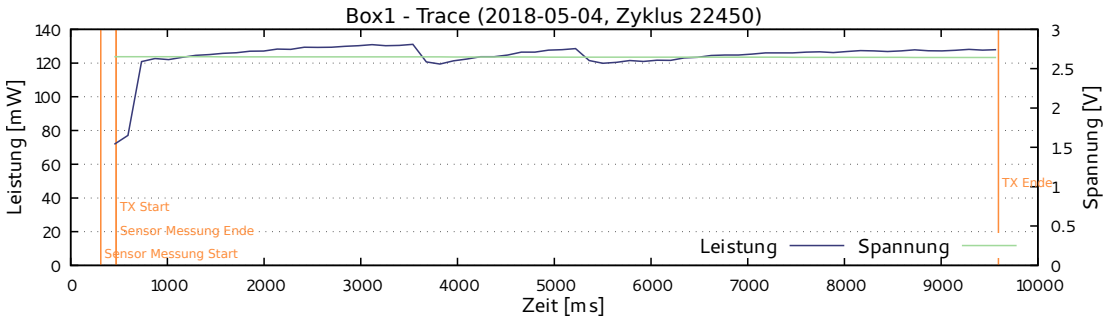


Abbildung 8.14.: Trace 22450 des Energieverbrauchs von Box-1 mit Konfiguration 1

Verglichen damit zeigt Abbildung 8.15 eine Messung mit Konfiguration 2 auf dem selben Sensorknoten. Diese Messung zeigt bereits deutlich mehr Details und stellt schnelle Änderungen in der Leistungsaufnahme bereits mit wesentlich steileren Flanken dar. Die Sensormessung wird hier mit mehreren Messpunkten erfasst, wodurch Änderungen während der Sensormessung sichtbar gemacht werden können.

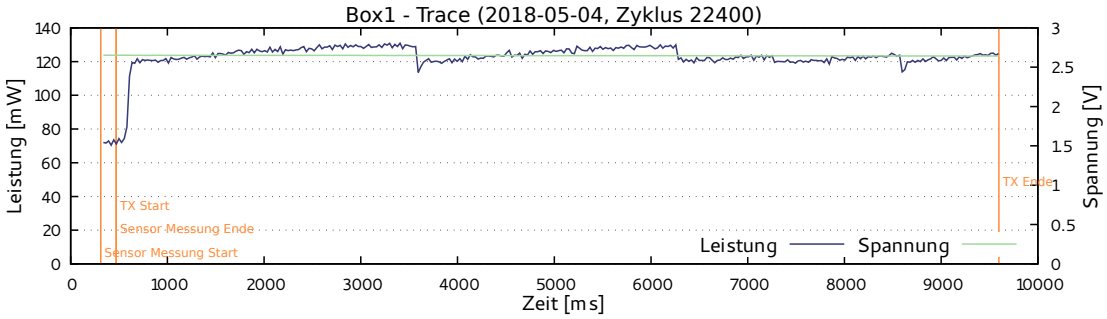


Abbildung 8.15.: Trace 22400 des Energieverbrauchs von Box-1 mit Konfiguration 2

In Abbildung 8.16 besitzt der Verlauf durch die verwendete Konfiguration 3 ein deutlich höheres rauschen, ohne dabei erkennbar mehr Details zum Verlauf zu bieten. Dahingegen ist der Leistungsabfall kurz nach 3,5 s in Abbildung 8.17 deutlicher auszumachen.

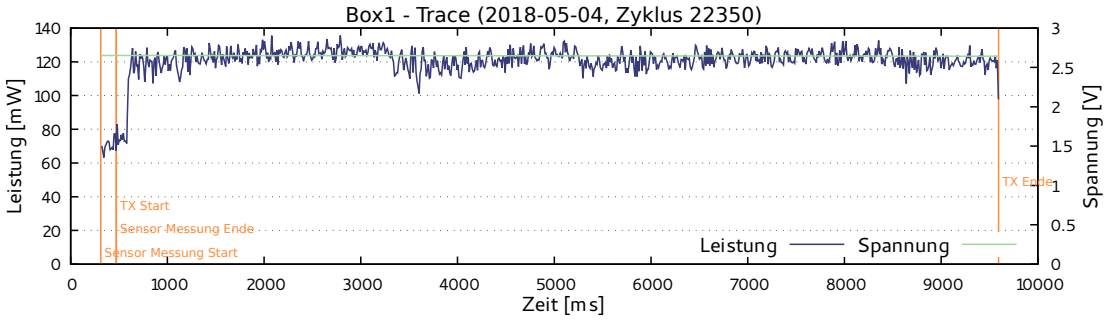


Abbildung 8.16.: Trace 22350 des Energieverbrauchs von Box-1 mit Konfiguration 3

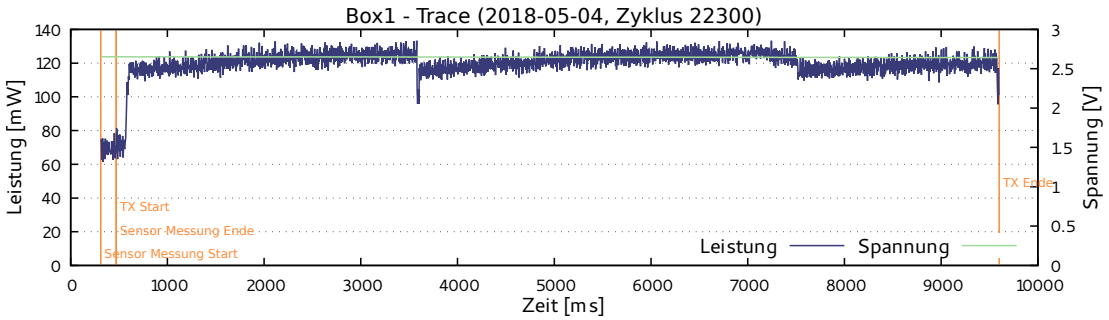


Abbildung 8.17.: Trace 22300 des Energieverbrauchs von Box-1 mit Konfiguration 4

Die beiden letzten gezeigten Traces dieser Serie decken aufgrund der statisch begrenzten Puffergröße nicht den vollständigen Zyklus ab. Abbildung 8.18 bietet mit Konfiguration 5 tiefere Einblicke in die Verteilung kurz auftretender Leistungsspitzen während die effektive Leistungsaufnahme erkennbar bleibt. Im Gegensatz dazu ist die Interpretation des Effektivwertes anhand von Abbildung 8.19 (Konfiguration 6) nicht mehr ohne analoge Filterung oder digitaler Nachverarbeitung möglich. Noch stärker tritt dieser Effekt bei Verwendung eines kleineren Messwiderstandes auf, wie in Abbildung 8.20 dargestellt ist. Der kleinere Widerstandswert erlaubt zwar einen größeren Messbereich, führt jedoch zu einer geringeren Auflösung.

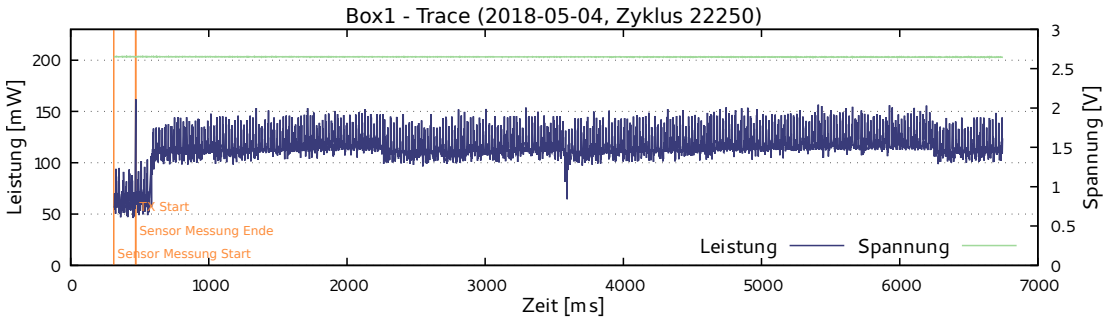


Abbildung 8.18.: Trace 22250 des Energieverbrauchs von Box-1 mit Konfiguration 5

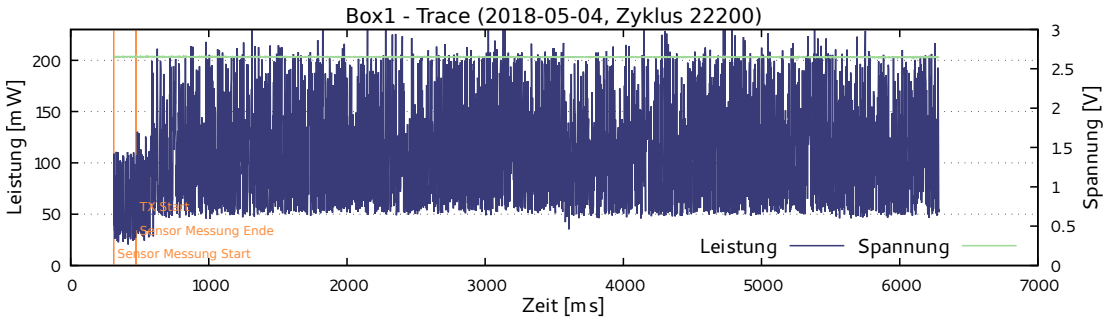


Abbildung 8.19.: Trace 22200 des Energieverbrauchs von Box-1 mit Konfiguration 6

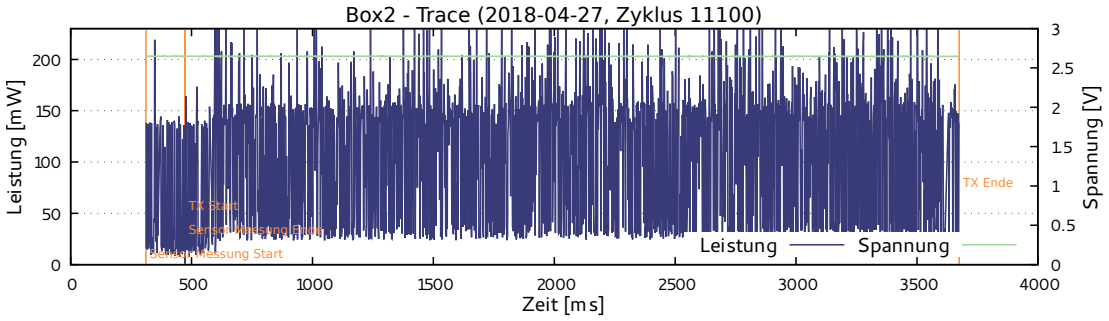


Abbildung 8.20.: Trace 11100 des Energieverbrauchs von Box-2 mit Konfiguration 6

In Abbildung 8.21 ist abweichend vom normalen Messen-Senden-Ablauf zu sehen, wie nach dem Start des Sensors zunächst die Zeitsynchronisation abgehandelt wird. Eindeutig erkennbar sind hier die fünf separaten SNTP-Abgleiche anhand der kurzen Einbrüche der Leistungsaufnahme, welche zwischen den SNTP-Aufrufen auftreten.

Eine weitere Variante des Programmablaufs wird in Abbildung 8.21 gezeigt. Hier wird zwischen Zeitsynchronisation und Funkübertragung ein Logeintrag auf der SD-Karte abgespeichert. Hier zeigt sich ein Nachteil der Eigenmessung. Wird der Programmablauf nicht häufig genug an den Thread abgegeben, der die Messwerte vom Messmodul einließt, werden keine Messwerte für diesen Zeitraum aufgezeichnet. Im vorliegenden Fall sorgt der blockierend implementierte SD-Karten Treiber dafür, dass für die Dauer des Schreibvorgangs keine Messwerte aufgezeichnet werden. Um das zu Umgehen, könnte der Thread zum Auslesen der Messwerte höher priorisiert werden, wodurch die Messung invasiver wird. Ist beides nicht mit den Anforderungen vereinbar, dann bleibt die Variante, für das Auslesen Direct Memory Access (DMA) zu verwenden oder die Messung von einem anderen Mikrocontroller durchführen zu lassen .

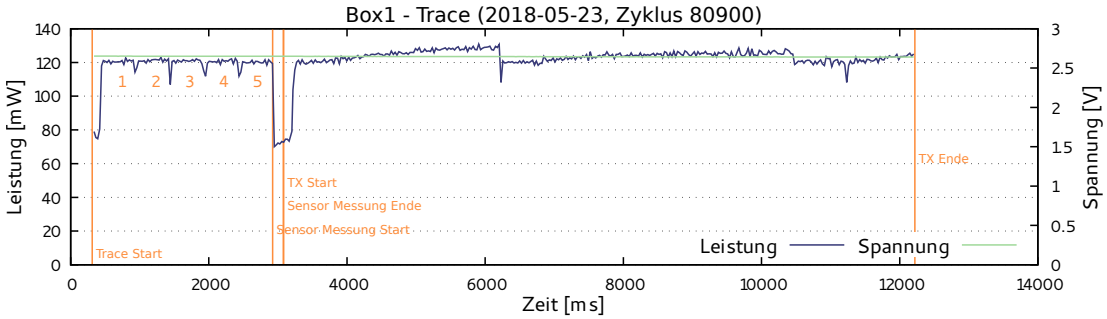


Abbildung 8.21.: Trace 80900 des Energieverbrauchs von Box-1 mit Konfiguration 2

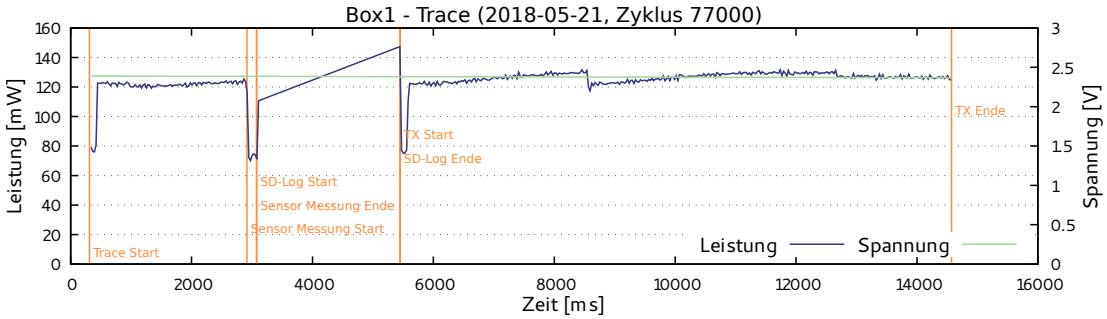


Abbildung 8.22.: Trace 77000 des Energieverbrauchs von Box-1 mit Konfiguration 2

Im Unterschied zu den anderen Sensorboxen besitzt Box-5 einen Feinstaubsensor mit aktivem Messverfahren. Abbildung 8.23 zeigt einen Trace der wesentlich höheren Leistungsaufnahme dieses Sensors. Zu Beginn des Traces wird die Zeitsynchronisation ausgeführt, wonach die Sensormessung mit dem initialisieren und Auslesen des MPL3115A2 stattfindet. Anschließend wird der Feinstaubsensor aktiviert, was an dem starken Anstieg kurz nach vier Sekunden zu erkennen ist. Der Verlauf des Graphen zeigt anfangs eine deutlich höhere Leistungsaufnahme, die dann langsam gegen einen stationären Wert von ca. $570\,\mathrm{mW}$ konvergiert. Dieser Verlauf ergibt sich aus dem Anlaufstrom des Lüfters, welcher sich langsam seiner Nenndrehzahl nähert.

Abbildung 8.24 zeigt einen weiteren Trace von Box-5, ohne Zeitsynchronisation am Anfang und mit einer deutlich schnelleren Übertragungszeit. Eine weitere Information, die sich aus dem Vergleich mit Abbildung 8.23 gewinnen lässt, ist die höhere Leistungsaufnahme bei der Feinstaubmessung. Diese ist mit hoher Wahrscheinlichkeit auf die schlechtere Effizienz des Spannungswandlers, aufgrund der niedrigeren Spannung am Superkondensator zurückzuführen.

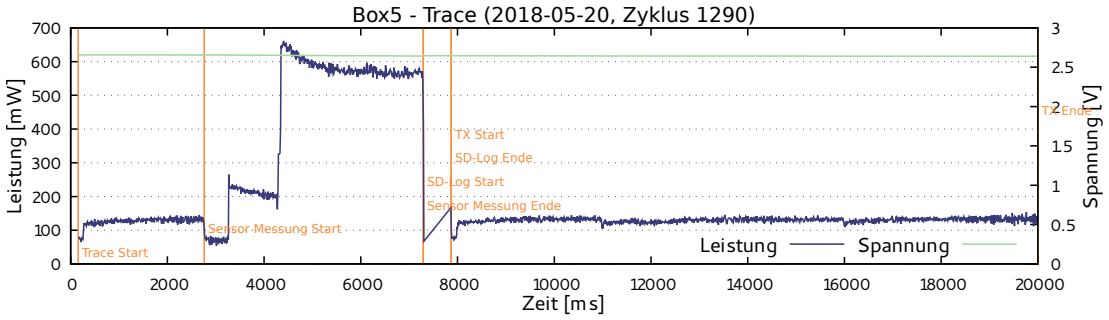


Abbildung 8.23.: Trace 1290 des Energieverbrauchs von Box-5 mit Konfiguration 3

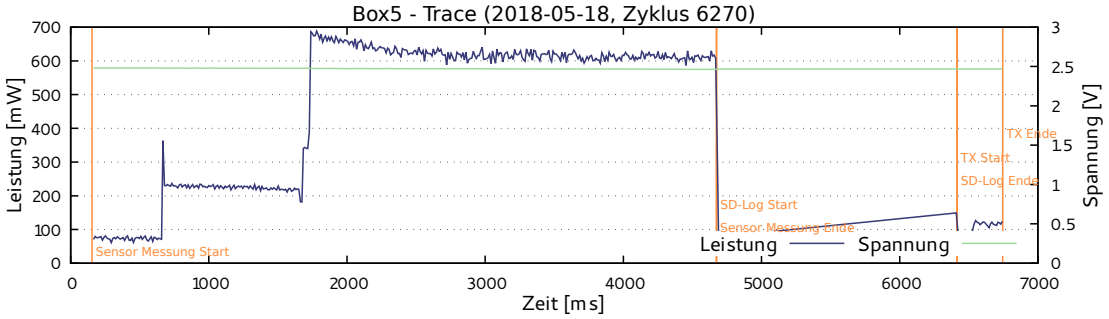


Abbildung 8.24.: Trace 6270 des Energieverbrauchs von Box-5 mit Konfiguration 3

Die Ergebnisse zeigen, dass Energiespeicher, Ladeelektronik, Messmodul und Logging-Komponente ordnungsgemäß funktionieren. Sie konnten erfolgreich zur Messdatengewinnung eingesetzt werden. Der Aufbau hat praktisch gezeigt, dass mit der lokalen Speicherung der Datensätze auch gänzlich autarke Versuche, unabhängig von umliegender Netzwerk-Infrastruktur, durchgeführt werden können. Für weitere Versuche lässt sich festhalten, dass ein detaillierteres lokales Logging hilfreich sein kann, um Übertragungsausfälle genauer zu untersuchen. Die Netzwerk-basierte Zeitsynchronisation mittels SNTP lässt sich über die energiesparende 802.15.4-Verbindung praktikabel einsetzen und kann mit der implementierten Lösung zur Korrektur des Taktgeber-Drifts verwendet werden. Es konnte aufgezeigt werden in welchen Fällen das gezeigte Konzept zur Eigenvermessung problematisch ist und welche Anpassungen für aussagekräftigere Ergebnisse geeignet sind. Weiterhin konnte demonstriert werden, wie sich anhand der Leistungs-Traces der Energieverbrauch detailliert analysieren lässt.

9. Ausblick und Fazit

In dieser Arbeit wurde die Konzeption und Implementierung einer modularen Plattform zur Untersuchung von energieautarken Sensorknoten gezeigt. Dazu wurden flexible Hardware- und Software-Komponenten für ein solarbetriebenes Energy-Harvesting System entwickelt, die es ermöglichen detaillierte Messungen zu Energieverbrauch und Systemverhalten in realistischen Szenarien zu erfassen. Die entworfenen Hardwarekomponenten wurden vom Entwurf über prototypische Testmodelle zu einsatzbereiten Modulen entwickelt. Das entworfene Lademodul bietet mit der flexiblen Einstellmöglichkeit von Ausgangsspannung und Maximum Power Point die Möglichkeit das System sowohl mit unterschiedlichen PV-Zellen und Superkondensatoren zu verwenden. Für das entwickelte Messmodul und den SD-Karten Treiber wurde eine Softwareunterstützung für RIOT implementiert. In Kombination mit dem FatFs Paket und dessen Einbindung in die virtuelle Dateisystem-Schicht von RIOT, wurde außerdem eine einfache und interoperable Möglichkeit zur persistenten Speicherung von Daten auf dem Sensorknoten geschaffen.

Durch die autarke Energieversorgung und die Möglichkeit Daten in großen Mengen Persistent zu speichern, bieten sich weitere Einsatzzwecke wie Langzeitmessungen in Gebieten mit gestörter oder fehlender Netzwerkinfrastruktur, Szenarien mit mobilen Knoten und allgemein dem Themenbereich Delay Tolerant Networking an.

Im Rahmen eines Feldtests konnte die Funktionalität des Gesamtsystems demonstriert werden. Dabei wurden erfolgreich Daten zu Umwelt, Energieverbrauch und Systemverhalten des Sensorknotens aufgezeichnet und ausgewertet. Die aufgezeichneten Leistungskurven erlauben einen detaillierten Einblick in die Energiebilanz von Sensorknoten in einem realistischen Szenario. Der vollständige Aufbau vereinfacht somit die Durchführung praxisnaher Versuche und die Gewinnung von aussagekräftigen Messdaten. Der Feldtest hat außerdem praktisch demonstriert, dass derartige Versuche hilfreich sein können, um die Praxistauglichkeit von Systemen zu überprüfen.

A. Firmware Quellcode

Der entwickelte Quellcode ist auf dem beigelegten Datenträger enthalten. Im Verzeichnis *CoAP2SQLiteLogger* befindet sich die Implementierung der Serverapplikation. *RIOT* enthält den vollständigen Firmware-Code. Die Sensor Anwendung ist im Unterverzeichnis *app* zu finden.

```
CoAP2SQLiteLogger
└── server.py
RIOT
├── ...
└── app
    └── eh_sensor
        ├── backup_reg.c
        ├── coap.c
        ├── gpio_wakeup.c
        ├── include
        │   ├── backup_reg_alloc.h
        │   ├── backup_reg.h
        │   ├── coap.h
        │   ├── gpio_wakeup.h
        │   ├── local_logger.h
        │   ├── node_time.h
        │   ├── radio.h
        │   ├── rehmon.h
        │   └── sensors.h
        ├── local_logger.c
        ├── main.c
        ├── Makefile
        ├── Makefile.include
        ├── node_time.c
        ├── radio.c
        ├── rehmon.c
        └── sensors.c
```

B. Sensordaten

Die aufgezeichneten Sensordaten befinden sich in der nachfolgenden Ordnerstruktur auf dem beigelegten Datenträger. Die Datenbankdatei ist im Ordner *data* abgelegt. Im Ordner *sd-cards* existiert ein Verzeichnis für jede Sensorbox, in dem sich die Trace-Aufzeichnungen und die Zusammenfassungen der Energieparameter befinden.

```
data
├── rpi_db_2018-05-23.db
└── sd-cards
    ├── box1
    │   ├── <e_info_*.log>
    │   ├── <trace_*.log>
    │   └── ...
    ├── box2
    ├── box3
    ├── box4
    └── box5
```

Literaturverzeichnis

[1] Maria Gregori Casas. *Transmission strategies for wireless energy harvesting nodes*. PhD thesis, Universitat Politècnica de Catalunya, Departament de Teoria del Senyal i Comunicacions, June 2014. URL http://upcommons.upc.edu/handle/2117/95379.

[2] Yi Xiang and Sudeep Pasricha. Harvesting-aware Energy Management for Multicore Platforms with Hybrid Energy Storage. In *Proc. of the 23rd ACM International Conference on Great Lakes Symposium on VLSI*, New York, NY, USA, 2013. ACM.

[3] Muhammad Imran, Abas Md Said, and Halabi Hasbullah. A survey of simulators, emulators and testbeds for wireless sensor networks. In *2010 International Symposium on Information Technology*, June 2010.

[4] Emmanuel Baccelli, Oliver Hahm, Mesut Günes, Matthias Wählisch, and Thomas C. Schmidt. RIOT OS: Towards an OS for the Internet of Things. In *Proc. of the 32nd IEEE INFOCOM. Poster*, Piscataway, NJ, USA, 2013. IEEE Press.

[5] Ala Al-Fuqaha, Mohsen Guizani, Mehdi Mohammadi, Mohammed Aledhari, and Moussa Ayyash. Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications. *IEEE Communications Surveys Tutorials*, 17(4):2347–2376, 2015.

[6] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future Gener. Comput. Syst.*, September 2013.

[7] Pedro Maia, Everton Cavalcante, Porfírio Gomes, Thais Batista, Flavia C. Delicato, and Paulo F. Pires. On the Development of Systems-of-Systems Based on the Internet of Things: A Systematic Mapping. In *ECSAW '14: Proc. of the 2014 European Conference on Software Architecture Workshops*, pages 1–8, New York, NY, USA, August 2014. ACM.

[8] Yong Ho Hwang. IoT Security & Privacy: Threats and Challenges. In *Proc. of the 1st ACM Workshop on IoT Privacy, Trust, and Security*, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3449-5.

[9] Sujesha Sudevalayam and Purushottam Kulkarni. Energy harvesting sensor nodes: Survey and implications. 13:443–461, 2011.

[10] Faisal Karim Shaikh and Sherali Zeadally. Energy harvesting in wireless sensor networks: A comprehensive review. *Renewable and Sustainable Energy Reviews*, 55:1041–1054, 2016.

[11] L. Xie and M. Cai. An In-shoe Harvester with Motion Magnification for Scavenging Energy from Human Foot Strike. 20:3264–3268, December 2015.

[12] Maria Gorlatova, John Sarik, Mina Cong, Ioannis Kymissis, and Gil Zussman. Movers and Shakers: Kinetic Energy Harvesting for the Internet of Things. 33:1624–1639, January 2015.

[13] P. Chambe, B. Canova, A. Balabanian, M. Pele, and N. Coeur. Optimization of Energy Harvesting Systems for RFID Applications. 8(7):1147–1150, 2014.

[14] Qianao Ju and Ying Zhang. Reducing Charge Redistribution Loss for Supercapacitor-operated Energy Harvesting Wireless Sensor Nodes. In *Proc. of the 2Nd International Workshop on Energy Neutral Sensing Systems*, New York, NY, USA, 2014. ACM.

[15] Yen Kheng Tan and Sanjib Kumar Panda. Self-Autonomous Wireless Sensor Nodes With Wind Energy Harvesting for Remote Sensing of Wind-Driven Wildfire Spread. *IEEE Transactions on Instrumentation and Measurement*, 2011.

[16] Trong Nhan Le, Alain Pegatoquet, Olivier Berder, and Olivier Sentieys. A Power Manager with Balanced Quality of Service for Energy-Harvesting Wireless Sensor Nodes. In *ENSsys'14: Proceedings of the 2Nd International Workshop on Energy Neutral Sensing Systems*, pages 19–24, New York, NY, USA, November 2014. ACM.

[17] Jason Hsu, Sadaf Zahedi, Aman Kansal, Mani Srivastava, and Vijay Raghunathan. Adaptive Duty Cycling for Energy Harvesting Systems. In *Proc. of the 2006 International Symposium on Low Power Electronics and Design, 2006.*, 2006.

[18] Cedric Adjih, Emnanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frederic Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. FIT IoT-LAB: A large scale open experimental IoT testbed. In *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*, Dec 2015.

[19] Anne-Sophie Tonneau, Nathalie Mitton, and Julien Vandaele. A Survey on (mobile) wireless sensor network experimentation testbeds. In *DCOSS - IEEE International Conference on Distributed Computing in Sensor Systems*, May 2014.

[20] X Development LLC. *Project Loon*, Letzter Zugriff am 14.04.2018. URL https://x.company/loon/.

[21] Matthias Woehrle et al. Power monitoring and testing in wireless sensor network development. In *WEWSN*, 2008.

[22] Xue Lin, Yanzhi Wang, Massoud Pedram, Jaemin Kim, and Naehyuck Chang. Event-driven and Sensorless Photovoltaic System Reconfiguration for Electric Vehicles. In *Proceedings of the 2015 Design, Automation & Test in Europe Conference & Exhibition*, pages 19–24, San Jose, CA, USA, 2015. EDA Consortium.

[23] Hengzhao Yang and Ying Zhang. A study of supercapacitor charge redistribution for applications in environmentally powered wireless sensor nodes. *Journal of Power Sources*, 2015.

[24] Bernd-C. Renner and Volker Turau. State-of-charge assessment for supercap-powered sensor nodes: Keep it simple stupid! In *International Conference on Networked Sensing (INSS)*, pages 1–6, June 2012.

[25] Samwha. Samwha Electric Catalogue, 2013. URL http://www.samwha.co.kr/sw{_}catalogue/catimage/23/catalogue{_}samwha{_}electric.pdf.

[26] Luis Zubieta and Richard Bonert. Characterization of double-layer capacitors for power electronics applications. *IEEE Transactions on Industry Applications*, 2000.

[27] Linear Technology. *LTC3105 Datasheet*, 2015. URL http://cds.linear.com/docs/en/datasheet/3105fb.pdf.

[28] National Association of Relay Manufacturers. *Engineers' Relay Handbook*. Hayden Book Company, 1966.

[29] SD Card Association. *SD Specifications Part 1 Physical Layer Simplified Specification Version 5.00* , 2015. URL https://www.sdcard.org/downloads/pls/pdf/part1_500.pdf.

[30] STMicroelectronics. *RM0351 Reference manual for STM32L4x5 and STM32L4x6 advanced ArmÂ®-based 32-bit MCUs*, 2018. URL http://www.st.com/resource/en/reference_manual/dm00083560.pdf. Rev. 6.

[31] STMicroelectronics. *UM1724 User manual for STM32 Nucleo-64 boards*, 2017. Rev. 12.

[32] Aosong Electronics. *Digital-output relative humidity & temperature sensor*, 2010. URL https://www.sparkfun.com/datasheets/Sensors/Temperature/DHT22.pdf. accessed April 20, 2018.

[33] Bosch Sensortec. *BMP280 Digital Pressure Sensor*, 2018. URL https://ae-bst.resource.bosch.com/media/_tech/media/datasheets/BST-BMP280-DS001-19.pdf. Rev. 1.19.

[34] NXP Semiconductors. *MPL3115A2 I2C precision pressure sensor with altimetry*, 2018. URL https://www.nxp.com/docs/en/data-sheet/MPL3115A2.pdf. Rev. 7.

[35] Nova Fitness. *Laser PM2.5 Sensor specification v1.3*, 2015. URL https://www.watterott.com/media/files_public/umnuvjtblnv/SDS011.pdf. accessed Jun 1, 2018.

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Hamburg, 4. Juni 2018 Michel Rottleuthner