Masterarbeit

Entwicklung eines mit Fernzugriff konfigurierbaren drahtlosen Sensornetzwerkes mit teils selbstversorgenden Knoten für den universitären Laborunterricht

Dipl.-Ing.(FH) Michael Steinberger

Institut für Technische Informatik
Technische Universität Graz
Vorstand: Interim Univ.-Prof. Dipl.-Ing. Dr. techn. Gernot Kubin

Begutachter: Dipl.-Ing. Dr. techn. Christian Kreiner
Betreuer: Dipl.-Ing. Leander Bernd Hörmann, BSc

Graz, im März 2013

Kurzfassung

Drahtlose Sensornetzwerke bestehen aus einer Vielzahl von Sensorknoten, welche über Funk miteinander kommunizieren. Im Vergleich zu drahtgebundenen Sensornetzwerken wird der Aufwand für Herstellung, Installation und im Betrieb reduziert. Diese drahtlosen Sensorknoten verfügen typischerweise über einen einfachen Prozessor, wenig Speicher und einen limitierten Energievorrat und werden von Batterien oder mit Energiegewinnungssystemen zur Umwandlung der Umgebungsenergie versorgt. Energiemanagement und die Auswahl energieeffizienter Bauteile sind in diesem Bereich besonders wichtig, um über lange Zeiträume Sensordaten erfassen, verarbeiten und zu einer Basisstation übertragen zu können.

Diese Masterarbeit beinhaltet die Entwicklung und Realisierung einer Laborplattform, mit der ein möglichst praxisnahes Arbeiten mit drahtlosen Sensorknoten für den universitären Laborunterricht erreicht werden kann. Damit können Studenten die Funktionsweise von drahtlosen Sensornetzwerken erarbeiten, sowie im Speziellen das Verhalten von EHDs (Energy-Harvesting-Devices) unter verschiedensten Umgebungseinflüssen testen. Diese Masterarbeit ist Teil des Europäischen Projekts “Remote-labs Access in Internet-based Performance-centered Learning Environment for Curriculum Support” (RIPLECS).

Abstract

A wireless sensor network is a collection of nodes communicating with each other via radio. In comparison to wired sensor networks, the cost of manufacture, installation and operation is reduced. These wireless sensor nodes typically have a small processor, less memory and limited energy storage, and are powered by batteries or energy harvesting systems. Energy management and the selection of energy-efficient components are very important in order to acquire data over long periods as well as to process and to transmit it to a base station.

This master thesis presents the development and implementation of a remote lab for laboratory practice-based of learning energy harvesting enhanced wireless sensor networks. Students can learn the behaviour and can get practical experience of wireless sensor networks and the influence of energy harvesting under various environmental conditions. This master thesis is part of the European project "Remote-labs Access in Internet-based Performance-centered Learning Environment for Curriculum Support" (RIPLECS).

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

Graz,am ....
(Unterschrift)

Danksagung

Ich bedanke mich bei meinen Eltern, die mich nicht nur finanziell, sondern auch moralisch sehr unterstützt haben.

Ganz besonders Bedanken möchte ich mich bei meiner Freundin Daniela für ihre Unterstützung und ihre unglaublichen Geduld in den letzten 3 Jahren.

Ich bedanke mich bei allem am Institut für Technische Informatik, vor allem bei Dipl.-Ing.(FH) Engelbert Meissl für die Unterstützung beim Laboraufbau.

Ich bedanke mich vor allem bei meinem Betreuer Dipl.-Ing. Leander Bernd Hörmann, BSc., der mich während der ganzen Masterarbeit unterstützt hat und bei allen aufgetretenen Schwierigkeiten immer Zeit für mich hatte.

Zum Schluss möchte ich mich noch bei meinem Begutachter Dipl.-Ing. Dr.techn. Christian Kreiner bedanken, der mich vor allem bei der Software-Implementierung unterstützt hat.

Diese Arbeit wurde unterstützt von dem Europäischen Projekt "Remote-labs Access in Internet-based Performance-centered Learning Environment for Curriculum Support" (RIPLECS).

517836-LLP-1-2011-1-ES-ERASMUS-ESMO

Graz, im März 2013 Michael Steinberger

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

Wireless Sensor Networks (WSNs) sind Rechnernetzwerke aus Sensorknoten, die mittels Funk miteinander kommunizieren. Dabei wird häufig ein selbstkonfigurierendes Ad-hoc Netzwerk verwendet. Die Hauptaufgabe der einzelnen Knoten besteht darin Informationen von den jeweiligen Sensoren zu sammeln und an den Host über ein sogenanntes Gateway weiterzuleiten. In Abbildung 1.1 ist ein typischer Aufbau eines drahtlosen Sensornetzwerks dargestellt.

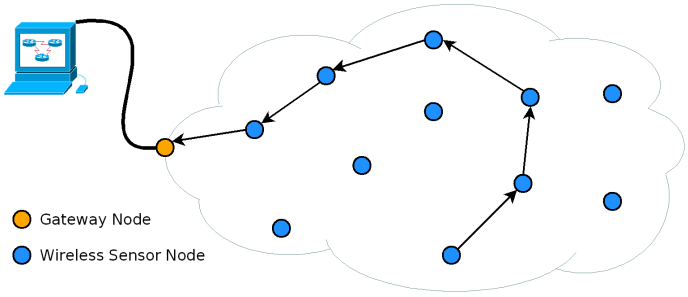


Abbildung 1.1: Typische Struktur eines drahtlosen Sensornetzwerks [1]

Aufgrund ihrer vorzugsweise geringen Baugröße sind Rechenleistung, Kommunikation, Sensorik und Energievorrat begrenzt. Zu ihren Vorteilen zählen der flexible Einsatz, geringe Herstellungskosten sowie geringer Installationsaufwand. Einsatzorientierte Strukturen und Übertragungstechnologien ermöglichen einen großen Anwendungsbereich für den drahtlosen und mobilen Einsatz. Das Hauptaugenmerk liegt bei der Energieaufnahme bzw. deren Speicherung, weil eine lange Laufzeit (Wochen bis Jahre) ohne zusätzliche Wartung, wie zum Beispiel der Batterietausch, in solchen Netzwerken vorausgesetzt wird. Dementsprechend wird viel Aufwand betrieben, um Lösungen für energieautarke Sensorknoten zu finden. Einer der erfolgversprechendsten Ansätze ist 'Energy-Harvesting'.

Erfolgsversprechende Fortschritte im Bereich 'Energy-Harvesting', das heißt Energieerzeugung durch Umgebungseinflüsse (Vibrationen, Temperaturdifferenzen, elektromagnetische Strahlung usw.), versprechen wesentliche Verbesserung in der Effizienz solcher Systeme. Mit dieser Methode kann Umgebungsenergie in elektrische Energie umgewandelt werden, um den Knoten ausreichend zu versorgen. Ein sogenanntes 'Energy-Harvesting-System' (EHS) besteht im wesentlichen aus einem 'Energy-Harvesting-Device' (EHD) zur Energiegewinnung und einem Energiespeicher. Mit dem Energiespeicher wird überschüssige Energie von den EHDs gespeichert. Diese Energie kann dann in Phasen mit zu geringer oder keiner Energiegewinnung für einen unterbrechnungsfreien Betrieb genutzt werden. Als Energiespeicher können entweder wiederaufladbare Batterien oder Ultrakondensatoren zum Einsatz kommen.

1.1 Motivation

Die Lehre der Theorie von drahtlosen Sensornetzwerken ist für Studierende meist nicht ausreichend, um die komplexe Arbeitsweise innerhalb des Systems gänzlich zu verstehen. Das praktische Arbeiten mit solchen Systemen würde den Lernprozess beschleunigen. Die Studenten könnten mit verschiedensten Einflüssen auf das Verhalten der selbstversorgenden Sensorknoten experimentieren. Vorgegebene Übungsaufgaben sollen die Studenten dazu bringen sich intensiv mit Energiemanagement, Ausfallsicherheit und Datenakquisitation zu beschäftigen.

Ein geeigneter Laboraufbau würde einige permanent versorgte Sensorknoten sowie selbstversorgende Sensorknoten mit Solarzellen und Energiespeicher beinhalten. Den Studierenden soll ein interaktiver Zugang zur Laborübung über das Internet bereitgestellt werden. Dieser Web-Zugang soll Statusinformation und Messergebnisse darstellen und die Konfiguration des Sensornetzwerkes bereitstellen.

Diese Arbeit ist Teil des Europäischen Projekts "Remote-labs Access in Internet-based Performance-centered Learning Environment for Curriculum Support" (RIPLECS). Die Motivation für das RIPLECS-Projekt wird folgend beschrieben:
Ziel des RIPLECS-Projekts ist die Definition und Entwicklung einer telematikgestützten Infrastruktur in Europa sowie die Ausarbeitung eines Lehrplans für Informations und Kommunikations Technologie (IKT), welcher eine Vielzahl an Möglichkeiten für gemeinsames Verfassen von Beiträgen und gemeinsames Lernen sowie simulationsbezogene und laborpraktische Lernmöglichkeiten bietet. Das RIPLECS-Projekt dient der Anpassung des DIPSEIL-Systems, um in einem E-Learning-Kontext im Rahmen eines Lehrplans zu Informations- und Kommunikationssystemen reale Experimente in einer „Remote"-Umgebung durchzuführen. Die Studierenden werden in der Lage sein, mit dem Experiment zu interagieren, Parameter zu ändern und in einigen Fällen auch die Experimente zu modifizieren und mitzugestalten. Mit der RIPLECS-Plattform wird die Verteilung von Ressourcen der Laborexperimente ermöglicht durch Nutzung von mehreren Web-Servern in einer einzelnen Netzwerktopologie. Lehrbeauftragte aus verschiedensten europäischen Ländern bekommen die Möglichkeit, die Vorteile eines lauffähigen Laborexperiments mit Inhalten in der jeweiligen Landessprache und ihrem eigenen Blickwinkel durchzuführen. Mit der Einbindung von Telekommunikationstechnologien und der Informatik in die virtuellen Messvorrichtungen wird eine Entwicklung von realen ferngesteuerten Laboren sowie ein Zugang in Echtzeit über das Internet ermöglicht. Den Beschränkungen traditioneller Labore, durch Platzmangel, teurer Laborausrüstung, Personalmangel und deren Verfügbarkeit nur innerhalb der Geschäftszeiten, kann damit entgegengewirkt werden. [2] [3]

1.2 Gliederung des Dokumentes

Dieses Dokument ist wie folgt aufgebaut. In **Kapitel 2** wird auf die Literaturrecherche sowie ähnliche Arbeiten und Projekte eingegangen. In **Kapitel 3** werden die Konzepte für den Laboraufbau, die Hardware-Komponenten und die dazugehörige Software beschrieben. **Kapitel 4** behandelt die Implementierungen der Konzepte. In **Kapitel 5** werden die Ergebnisse von Messungen und das Verhalten des gesamten Systems im Echtbetrieb zusammengefasst. **Kapitel 6** beinhaltet die Schlussbemerkungen sowie einen Ausblick für zukünftige Arbeiten. Im Anhang finden sich sämtliche Schaltpläne.

1.3 Aufgabenstellung

In dieser Masterarbeit soll ein Konzept erstellt sowie die Implementierung einer fernsteuerbaren Laborplattform für drahtlose Sensornetzwerke durchgeführt werden.

Der Laboraufbau soll alle Aspekte von drahtlosen Sensornetzwerken abdecken können. Das gesamte System muss möglichst flexibel aufgebaut sein, um auf eine Vielzahl von Laborübungen angepasst werden zu können. Besonderes Augenmerk soll auf Energiegewinnungsysteme für den autarken Betrieb von Sensorknoten gelegt werden. Für die Studierenden soll eine Möglichkeit geschaffen werden sich mit der Konfiguration und dem Betrieb eines drahtlosen Sensornetzwerkes auseinanderzusetzen.

Die Umsetzung der Masterarbeit besteht aus einem Hardware-Teil und aus dem Software-Teil. Der Hardware-Teil beinhaltet die Entwicklung der Schaltungen für das EHS sowie für die Steuerung des Laboraufbaus. Im Software-Teil wird auf die Entwicklung der Serverdienste sowie der Applikation für das drahtlose Sensornetzwerk eingegangen.

2 Literaturrecherche

In diesem Kapitel werden bestehende Arbeiten aus den Bereichen interaktiver Laborplattformen, drahtlose Sensornetzwerke und Energiegewinnungssystemen behandelt. Ein bereits etabliertes Projekt zur Verwaltung von interaktiven Laborplattformen für Studenten wird in Kapitel 2.1 beschrieben. Laborplattformen für unterschiedliche Einsatzzwecke zur Evaluierung von drahtlosen Sensornetzwerken werden in Kapitel 2.2 beschrieben. In Kapitel 2.3 sind verschiedene Netzwerkprotokolle und deren Einsatzbereiche erläutert. In Kapitel 2.4 wird der Einsatz von Energiegewinnungssystemen und deren Varianten beschrieben. Zum Schluss wird in Kapitel 2.5 die Literaturrecherche zusammengefasst.

2.1 iLab Projekt

Das iLab Projekt des Massachusetts Institute of Technology (MIT) [4] liegt der Idee zugrunde, Mitgliedern der Fakultät ihre Testplattformen für unterschiedlichste Experimente über das Internet für Studenten zur Verfügung zu stellen. Es ermöglicht Studierenden via ferngesteuerte Laboreinrichtungen mit echten Instrumenten und Geräten zu arbeiten. Finanzielle, räumliche und sicherheitsrelevante Gesichtspunkte können mit ferngesteuerten Online-Laboren teilweise entkräftet werden. Vor allem aber wäre der Zugang jederzeit und von jedem beliebigen Ort mit Internetzugang möglich.

Das iLab Projekt stellt eine Sammlung von Software-Werkzeugen zur Verfügung, um einen einfachen Web-Zugang für komplexe Laborexperimente zu etablieren. Die *iLab Shared Architecture* (ISA) besteht aus einen robusten, skalierbarer und offenen (Open-Source) Infrastruktur für Internetdienste, mit der ein Software-Framework für Benutzerverwaltung für eine große Vielfalt an Laborexperimenten bereitgestellt wird. Benutzerzugriffe und Online-Labore werden durch gemeinsames Anmeldeverfahren (single sign-on) und eine einfache Administrations-Schnittstelle verwaltet. In Abbildung 2.1 ist die Struktur des iLab Projekts dargestellt. Die iLab-Shared-Architecture hat folgende Ziele:

- Minimaler Entwicklungs- und Managementaufwand für Benutzer und Anbietern von ferngesteuerten Laboreinrichtungen
- Bereitstellung einer umfangreichen Sammlung von Diensten und Entwicklungswerkzeugen
- Einfach skalierbar für eine große Anzahl an Benutzern
- Ermöglicht einen gemeinsamen Zugang für Universitäten mit unterschiedlichsten Netzwerkinfrastrukturen

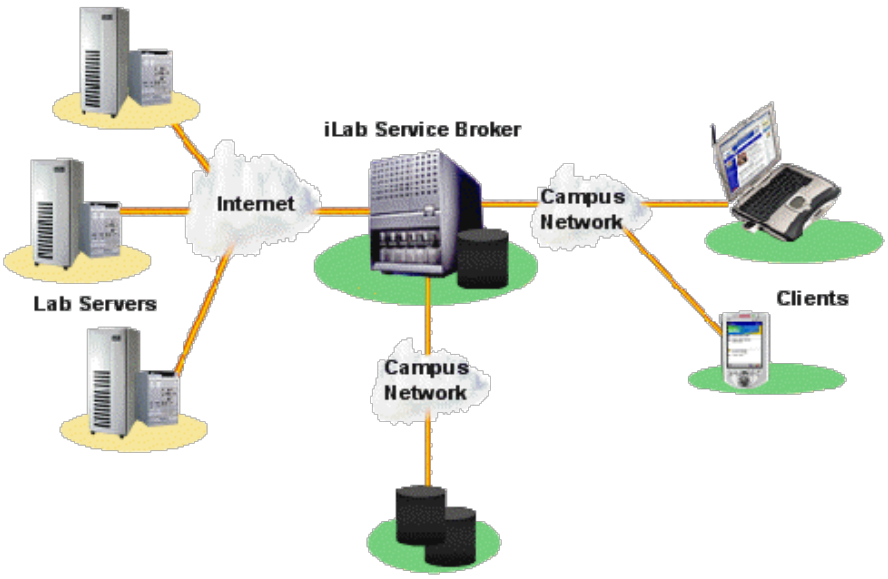


Abbildung 2.1: Struktur des iLab Projekts [4]

2.2 Testumgebungen für drahtlose Sensornetzwerke

Für die Simulation von drahtlosen Sensornetzwerken existieren bereits brauchbare Werkzeuge [5] [6]. Einen tieferen und realistischeren Einblick in solche Systeme können jedoch nur "echte" Testumgebungen vermitteln. Themen wie begrenzte Ressourcen, Verlust von Kommunikationspaketen und Energieabhängigkeiten sollen damit im Echtbetrieb evaluiert werden können. Es gibt unterschiedlichste Arten von Testumgebungen für drahtlose Sensornetzwerke. In diesem Kapitel werden drei unterschiedliche Konzepte vorgestellt.

Bei der Testplattform *MoteLab* [7] werden die MIB-600 Sensorknoten von Crossbow® verwendet, damit werden sogenannte Backchannel-Interfaces zur Verfügung gestellt. Dadurch soll eine Remote-Programmierung und eine Überwachung von Sensorknoten ermöglicht werden. Abbildung 2.2 zeigt den strukturellen Aufbau von der MoteLab-Testplattform. Ein Web-Zugang ermöglicht dem Benutzer den Inhalt und Ablauf von Experimenten zu erstellen. Generierte Daten aus den Experimenten werden in einer Datenbank gespeichert und können vom Benutzer direkt oder über den Web-Zugang wieder extrahiert werden.

Mit dem *WSNTB* [8] wurde eine Testumgebung für heterogene drahtlose Sensornetzwerke geschaffen. Die Umgebung wird verwendet, um Ergebnisse von simulierten Netzwerkprotokollen mit realen Ergebnissen zu vergleichen. Das System besteht aus einer Hardware-Infrastruktur und einem Software-Framework. Die Hardware-Infrastruktur besteht aus Servern, Gateways und Sensorknoten. Die Gateways sind direkt mit den Sensorknoten per USB oder RS232 verbunden und stellen gleichzeitig eine Verbindung mit dem Internet her. Diese Anbindung garantiert jederzeit den Zugriff auf die Sensorknoten, um diese zu rekonfigurieren und Statusinformation zu erhalten. Es können gleichzeitig mehrere drahtlose Sensornetzwerke über das Internet bedient werden. Das Software-Framework unterstützt

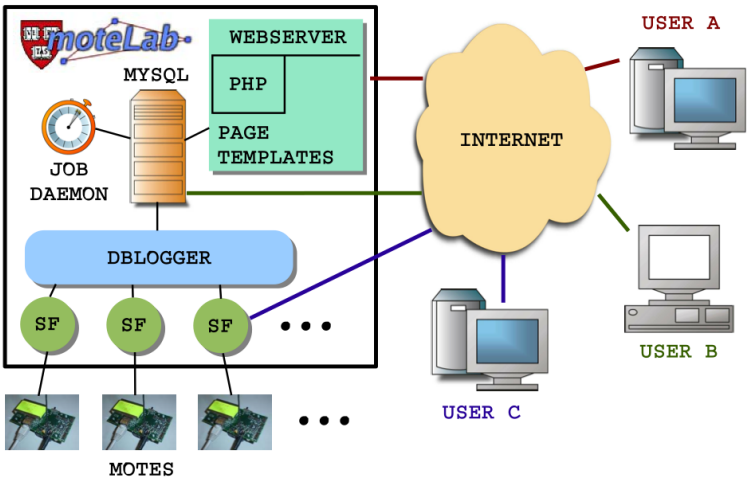


Abbildung 2.2: Struktur der MoteLab-Testplattform [7]. Die farbigen Verbindungen untereinander zeigen den direkten Datenfluss. Drei externe Benutzer (A,B und C) benutzen zum Beispiel gemeinsam die Testplattform. Benutzer A legt einen Ablauf für ein Experiment fest, der erst später gestartet werden soll. Benutzer B greift auf die Datenbank zu, um Daten von einem bereits abgeschlossenen Experiment zu sammeln. Benutzer C hat ein Experiment gestartet und kann durch die Verbindung mit den Serial-Forwarder (SF) direkt Nachrichten von dem Sensorknoten empfangen oder senden.

das verbreitete TinyOS [9] sowie das von den Autoren entwickelte LOS [10]. TinyOS und LOS sind beides Betriebssysteme für drahtlose Sensornetzwerke mit komponentenbasierter Archtitekur und ereignisbasierten Ausführungsmodellen. Mit der zur Verfügung gestellten Middleware soll dem Benutzer die Generierung von eigenen Experimenten erleichtert werden.

Das *Wireless Sensor Network Testbed for Mobile Data Communication* [11] ist speziell auf das Evaluieren von drahtlosen Sensornetzwerkprotokollen mit beweglichen Sensorknoten ausgelegt. Die Testumgebung besteht aus einer Basisstation, fünf Sensorknoten und dem sogenannten Data-Mule. Abbildung 2.3 zeigt die Struktur der Testumgebung. Durch sich bewegende Sensorknoten ergeben sich andere Anforderungen an die verwendeten Netzwerkprotokolle. Es besteht zwar die Möglichkeit vorhandene Simulatoren zu verwenden, jedoch können diese die Performance der Netzwerkprotokolle nur mangelhaft im Echtbetrieb evaluieren. Die fixen Sensorknoten haben die Aufgabe die Umgebungstemperaturen zu messen und diese lokal zu speichern. Diese gewonnenen Daten werden von den mobilen Sensorknoten (Data-Mule) gesammelt, bei einem zentralen Knoten (Basisstation) abgeliefert und schließlich auf einem Desktop-PC visualisiert. Kommt der Data-Mule in die Sende-/Empfangsreichweite eines fixen Sensorknotens, wird ein drahtloser Datentransfer zwischen diesen beiden Knoten durchgeführt. Um die Performance von dem eingesetzten Netzwerkprotokoll zu evaluieren, sind zwei fixe Sensorknoten über ein kabelgebundenes Netzwerk mit dem Desktop-PC verbunden.

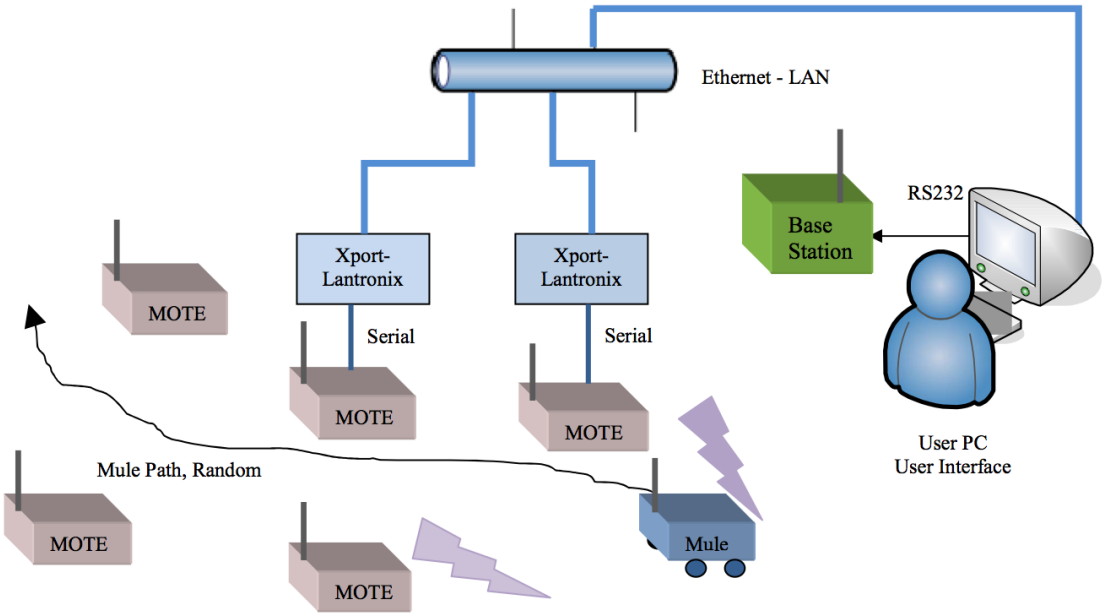


Abbildung 2.3: Struktur der Testumgebung für drahtlose Sensornetzwerke mit beweglichen Sensorknoten [11]

2.3 Software für drahtlose Sensornetzwerke

Drahtlose Sensornetzwerke sind aufgrund der möglichen Einsatzgebiete empfänglich für Störungen. Umgebungseinflüsse können negative Auswirkungen auf die Stabilität und Ausfallsicherheit des gesamten Systems zur Folge haben. Kriterien wie flexibel ein Netzwerkprotokoll reagiert wenn ein Sensorknoten im Kommunikationspfad ausfällt, oder wie Sensorknoten nach einem Ausfall wieder synchronisiert werden, sind entsprechend zu berücksichtigen.

In [12] werden Faktoren wie Zeitverzögerungen, zur Verfügung stehende Energie, Datendurchsatz, Fehlerwahrscheinlichkeit und Netzwerktopologie (Stern- oder vermaschte Topologie) herangezogen, um den Einfluss auf die Fehlertoleranz und Ausfallsicherheit zu bestimmen. Die zur Verfügung stehende Restenergie eines Sensorknotens kombiniert mit energieabhängigen (Power-Aware) Netzwerkrouting erhöht die Fehlerwahrscheinlichkeit und vermindert die Leistungsfähigkeit des Netzwerks aufgrund der steigenden Latenzzeiten.

Energieeffizienz und Netzwerk-Kapazität sind vielleicht die wichtigsten Aspekte in einem drahtlosen Sensornetzwerk. Mit Topologie-Kontrollalgorithmen können Netzwerkverbindungen so etabliert werden, dass Netzwerk-Kapazitäten steigen und gleichzeitig der Energieverbrauch sinkt. In [13] wird die Schlüsselidee der Topologie-Kontrolle so definiert, dass im Gegensatz zum Senden mit maximaler Leistung, alle Knoten in einem Multi-Hop-Netzwerk ihre Sendeleistungen ermitteln und entsprechend der Kriterien eine energieeffiziente Topologie mit optimierten Nachbarschaftsbeziehungen definiert werden kann. Die Annahme, dass baugleiche Sensorknoten immer dieselbe maximale Sendeleistung haben, trifft in der Praxis nicht immer zu. Bei heterogenen Sensornetzwerken sind die Unterschiede noch weit größer.

Die meisten Algorithmen können nicht direkt auf heterogene Netzwerke angewandt werden. Dementsprechend werden in [13] geeignete Topologie-Kontrollalgorithmen, für heterogene drahtlose Sensornetzwerke mit unterschiedlichen Sendereichweiten, vorgeschlagen.

Die Wahl von geeigneten Routing-Protokollen hängt von mehreren Faktoren ab. Nutzungsdauer, Verlustrate der übertragenen Pakete sowie die Anzahl an teilnehmenden Knoten im Netzwerk haben einen entscheidenden Einfluss auf die Auswahl. Routing-Protokolle sind prinzipiell in zwei Gruppen unterteilt, in Proaktive und Reaktive. Reaktive Protokolle verwenden Tabellen mit den Routing-Informationen, den sogenannten Distanzvektoren. Jeder Knoten generiert während der Laufzeit die entsprechenden Einträge und speichert diese lokal. Bei den proaktiven Protokollen werden die Routing-Informationen bereits vorab festgelegt und werden zur Laufzeit nicht mehr verändert. Die offensichtlichen Nachteile sind die unberücksichtigten Energiereserven einzelner Sensorknoten sowie eventuell nicht vorhandene Alternativen, neue Pfade zu etablieren wenn ein oder mehrere Knoten ausfallen. In [14] werden vier unterschiedliche Protokolle für drahtlose Sensornetzwerke miteinander verglichen. Folgende Routing-Protokolle wurden simuliert und evaluiert:

- **Flooding:** Nachrichten werden an alle benachbarten Sensorknoten versandt, zusätzliche Routing-Tabellen werden nicht verwendet.
- **TinyOS 1.x Multihop Routing:** Algorithmus für den kürzesten Pfad zu einem gemeinsamen Zielknoten
- **Low-Energy Adaptive Clustering Hierarchy (LEACH):** Gruppenbasierendes Protokoll
- **Ad hoc On-demand Distance Vector (AODV):** Reaktives Protokoll

Das Ergebnis der Studie brachte hervor, dass jedes einzelne Protokoll Vor- und Nachteile hat und deswegen selektiv auf die Bedürfnisse und Anforderungen ausgewählt werden muss.

2.4 Systeme zur Energiegewinnung aus der Umgebung

Energiegewinnungssysteme finden bei Sensorknoten Verwendung, die ohne Wartung über lange Laufzeiten verfügen sollen [15]. Auch örtliche Gegebenheiten können Wartungsarbeiten unmöglich machen und so die Notwendigkeit eines autarken Betriebs erfordern. Der Einsatz solcher Systeme ermöglicht Energie aus der Umgebung der Sensorknoten in elektrische Energie umzuwandeln. Tabelle 2.1 zeigt typische Technologien zur Energiegewinnung aus der Umgebung.

Harvesting technology	**Power densitiy**
Solar cells (outdoors at noon)	15mW/cm^2
Piezoelectric (shoe inserts)	300μW/sm^3
Vibration (small microwave oven)	116μW/sm^3
Thermoelectric (10°C gradient)	40μW/sm^3
Acoustic noise (100dB)	960nW/sm^3

Tabelle 2.1: Typische Technologien zur Energiegewinnung aus der Umgebung [16]

Für einen autarken Betrieb besteht die Notwendigkeit, dass Energiegewinnungssysteme mehr Energie liefern können als Sensorknoten im Durchschnitt verbrauchen. Unter gewissen Umständen sind zum Beispiel Solarzellen nicht in der Lage die angegebenen Leistungswerte zu generieren. Das trifft vor allem bei starker Bewölkung oder während der Nacht für im Freien installierte Solarzellen zu. Ein weiterer Störfaktor bei Solarzellen ist die Verschmutzung oder die altersbedingte Abnahme der Effizienz. Die genannten Faktoren müssen bei Dimensionierung der Solarzellen berücksichtigt werden. Periodische Schwankungen können durch einen geeigneten Energiespeicher überbrückt werden. Brauchbare Komponenten sind zum Beispiel wiederaufladbare Batterien oder Kondensatoren mit hohen Speichervermögen, sogenannte Ultrakondensatoren [16][17]. Zur Charakterisierung von Energiegewinnungssystemen in Abhängigkeit unterschiedlichster Einflussfaktoren kann das System von [18] verwendet werden.

2.5 Zusammenfassung

Aus der Literaturrecherche von unterschiedlichen Arbeiten geht hervor, dass alle mehr oder weniger Gemeinsamkeiten mit dem Thema dieser Diplomarbeit haben. Das iLab-Projekt stellt ein äußerst interessantes System für das Zusammenspiel von Laborplattformen und Benutzerverwaltung dar. Der größte Unterschied besteht jedoch, wie die Plattformen und der Web-Zugang für die Studierenden verwaltet werden. Während die Benutzer in dieser Diplomarbeit ebenfalls zentral verwaltet werden, ist die Laborplattform im lokalen Netzwerk des Instituts integriert und die Studierenden können von außen über das Internet zugreifen.

Das Motelab-Projekt beinhaltet ebenfalls verwendbare Ansätze. Die eingesetzten Sensorknoten von Crossbow® zeichnen sich wegen der flexiblen Einsatzzwecke als geeignete Komponenten aus. Die beschriebenen Netzwerk-Protokolle bilden eine geeignete Auswahl für die Evaluierung und fließen in das Konzept der Diplomarbeit ein. Dadurch können Studierende die Eigenschaften aus einer Vielzahl von Algorithmen evaluieren und ein Verständnis für die unterschiedlichen Einsatzzwecke erhalten.

Die Testumgebung im Projekt WSNTB zeigt eine verwertbare Kombination aus Software-Framework und Hardware-Infrastruktur. Der Webserver stellt den Studierenden den Zugang zu der Laborübung über das Internet zu Verfügung. Die Basisstation und der Webserver können über die MySQL-Datenbank Informationen austauschen.

3 Design

Dieses Kapitel behandelt die Ideen und Möglichkeiten zur Entwicklung eines Laboraufbaus für konfigurierbare drahtlose Sensornetzwerke. Es werden folgend auch die Hardware-Komponenten sowie die Software-Architekturen spezifiziert.

3.1 Konzept des Remote-Labs

Das Konzept für den gesamten Laboraufbau umfasst eine Vielzahl von Komponenten und wurde in [19] und [20] vorgestellt. Abbildung 3.1 zeigt das Konzept des Remote-Lab.

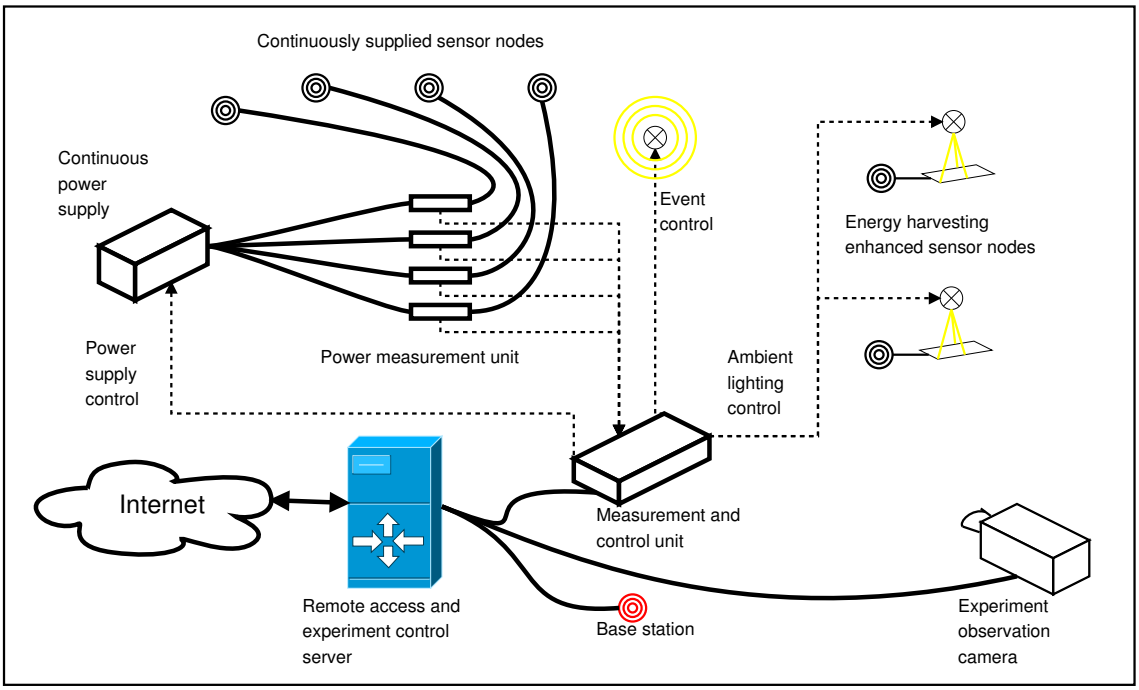


Abbildung 3.1: Konzept für das Remote-Lab [19]

Die notwendigen Eigenschaften und Aufgaben der Komponenten sowie deren Kombination zu einer Einheit werden auf den folgenden Seiten erläutert.

3.1.1 Server für Fernzugriff und Experimentensteuerung

Der Server für den Fernzugriff und die Experimentensteuerung (Remote Access and Experiment Control Server) wird benötigt, um die Laborübungen durchführen zu können.

Damit wird auch der Zugang zum Laboraufbau über das Internet ermöglicht. Folgende Funktionen muss der Server bereitstellen:

- An- und Abmelden am/vom System
- Auswahl einer Übung aus einer vorab definierten Übungssammlung
- Anzeigen der Messdaten von den Sensorknoten
- Anzeigen der Statusinformationen der Sensorknoten
- Anzeigen und Exportieren der Messergebnisse
- Konfiguration der Sensorknoten und des Netzwerkes
- Videoüberwachung in Echtzeit
- Steuerung des Hintergrundlichts und Ereignislichts
- Steuerung der Versorgung der permanent versorgten Sensorknoten

3.1.2 Basisstation

Die Basisstation (Base Station) wird benötigt, um mit den drahtlosen Sensorknoten zu kommunizieren. Es werden Daten und Statusinformationen vom Netzwerk empfangen und Konfigurationen über das Netzwerk übermittelt. Die Basisstation ist mit dem Server für den Fernzugriff und die Experimentensteuerung verbunden. Dadurch wird den Studenten ermöglicht mit dem Sensornetzwerk zu kommunizieren. Folgende Funktionen müssen von der Basisstation bereitgestellt werden:

- Empfangen von Statusinformation und Messdaten sowie die Weiterleitung an den Server
- Übermittlung von Konfigurationsnachrichten vom Server zu den einzelnen Sensorknoten

3.1.3 Permanent versorgte Sensorknoten

Mit den permanent versorgten Sensorknoten (Continuously Supplied Sensor Nodes) sollen die Basisfunktionen eines drahtlosen Sensornetzwerkes dargestellt und evaluiert werden können. Folgende Funktionen müssen zur Verfügung stehen:

- Konfiguration der Sensorknoten
- Kommunikation der Sensorknoten untereinander
- Konfiguration des Netzwerkes
- Messung von Umgebungsgrößen wie Temperatur und Helligkeit
- LEDs zur Anzeige von Statusinformationen

3.1.3.1 Netzteil für permanente Versorgung

Das Netzteil (Continuous Power Supply) wird für die permanent versorgten Sensorknoten benötigt. Für einen störungsfreien Betrieb der Sensorknoten muss eine stabile Spannungsversorgung garantiert werden. Folgende Funktionen muss das Netzteil erfüllen können:

- Konstante Spannungsversorgung
- Schaltbare Versorgungsausgänge

3.1.3.2 Steuerung der Versorgung

Die Steuerung der Versorgung (Power Supply Control) übernimmt das selektive Ein- oder Ausschalten der Sensorknoten. Damit soll die Versorgung vom Studenten ferngesteuert werden. Mit dieser Eigenschaft kann das Ausfallen von Sensorknoten im Netzwerk simuliert werden. Folgende Funktion muss gewährleistet sein:

- Steuerung der schaltbaren Versorgungsleitungen für die permanent versorgten Sensorknoten

3.1.4 Mess- und Kontrolleinheit

Die Mess- und Kontrolleinheit (Measurement and Control Unit) übernimmt drei Aufgaben. Zum einen die Messung der Leistungsaufnahme der permanent versorgten Sensorknoten, zum anderen die Steuerung des Umgebungslichts und des Ereignislichts sowie die Steuerung der schaltbaren Versorgungsleitungen.

Folgende Funktionen müssen von der Mess- und Kontrolleinheit erfüllt werden:

- Messung der Versorgungsspannung und Stromaufnahmen der permanent versorgten Sensorknoten
- Steuerung des Umgebungslichts und Ereignislichts
- Steuerung der schaltbaren Versorgungsleitungen für die permanent versorgten Sensorknoten

3.1.4.1 Messung der Leistungsaufnahmen

Die Messung der Leistungsaufnahmen (Power Measurement Unit) besteht aus der Messung der Versorgungsspannungen und der Stromaufnahmen. Folgende Funktionen müssen erfüllt werden:

- Spannungsmessung
- Strommessung

3.1.4.2 Ereigniskontrolle

Die Ereigniskontrolle (Event control) wird verwendet, um Ereignisse zu generieren, welche von den Sensorknoten detektiert werden sollen. Es kann zum Beispiel der Lichtkegel einer Taschenlampe simuliert werden. Ein drahtloses Alarmsystem kann dann derartig konfiguriert werden, um auf eine solche Lichtquelle zu reagieren. Folgende Funktionen müssen erfüllt werden:

- Steuerung des Ereignislichts

3.1.4.3 Umgebungslichtkontrolle

Die Umgebungslichtkontrolle (Ambient lighting control) dient dazu, realistische Umwelteinflüsse zu simulieren. Dementsprechend können damit Zustände des Tageslichts wie Sonnenaufgang, Sonnenstand sowie Nächte nachgeahmt werden. Folgende Funktionen müssen erfüllt werden:

- Steuerung des Umgebungslichts

3.1.5 Selbstversorgende Sensorknoten mit Energiegewinnungssystem

Die selbstversorgende Sensorknoten mit Energiegewinnungssystem (Energy Harvesting Enhanced Sensor Nodes) sind mit Solarzellen ausgestattet. Diese sind nicht elektrisch mit dem restlichen Laboraufbau verbunden und können nur über das drahtlose Netzwerk konfiguriert werden. Zusammen mit den permanent versorgten Sensorknoten bilden sie das kombinierte drahtlose Sensornetzwerk. Folgende Funktionen müssen bereitgestellt werden:

- Konfiguration der Sensorknoten
- Kommunikation der Sensorknoten untereinander
- Konfiguration des Netzwerkes
- Messung von Umgebungsgrößen wie Temperatur und Helligkeit
- LEDs zur Anzeige von Statusinformationen
- Energiegewinnungssystem mit Solarzellen (EHS)
- Speicherung von Energie
- Energiemanagement

3.1.6 Überwachungskamera

Die Überwachungskamera (Experiment Observation Camera) wird verwendet, um den Laboraufbau über den Web-Zugriff in Echtzeit darzustellen. Folgende Funktion muss gewährleistet werden:

- Stream der Videoüberwachung in Echtzeit

3.2 Hardwaredesign

In diesem Kapitel werden die Konzepte für das Energiegewinnungssystem der selbstversorgenden Sensorknoten sowie die Mess-, Kontroll- und Versorgungseinheit für permanent versorgte Sensorknoten und die Beleuchtungssteuerung definiert.

3.2.1 Energiegewinnungssystem

Die selbstversorgenden Sensorknoten müssen mit einem Energiegewinnungssystem ausgestattet werden [21]. In Abbildung 3.2 ist die schematische Darstellung des Energiegewinnungssystems ersichtlich [19]. Dafür soll eine Platine entworfen werden, die mit dem Erweiterungsstecker der Sensorknoten kompatibel ist. Die Speicherkondensatoren sollen einfach austauschbar sein, zum Beispiel mit Hilfe einer Steckerleiste. Die Schaltung muss folgende Funktionen beinhalten:

- **Solarzellensteuerung:** Ermöglicht das Aktivieren/Deaktivieren der Solarzelle mit einem digitalen Ausgang vom Sensorknoten.
- **Spannungsreglersteuerung:** Damit kann durch einen digitalen Ausgang vom Sensorknoten der gewünschte Spannungsregler bestimmt werden.
- **Speicherkondensatorsteuerung:** Entspricht der Vorgabe, dass sich die Kondensatoren unabhängig voneinander mit der Spannungsversorgung, ebenfalls durch einen digitalen Ausgang gesteuert, verbinden können. Überschüssige Energie, welche nicht für den Betrieb der Sensorknoten benötigt wird, soll mit den Kondensatoren gespeichert werden. **Wichtig:** Ungeladene Kondensatoren können beim Ladevorgang die Solarzelle insoweit belasten, dass die Eingangsspannung der Spannungsregler unter die minimale Eingangsspannung abfällt. Die Spannungsregler könnten somit die Versorgungsspannung für den Sensorknoten nicht mehr gewährleisten. Dies muss berücksichtigt werden, indem eine Regelung für den Ladestrom implementiert wird, die ein Absinken unter einen bestimmten Pegel verhindert.
- **Spannungsmessungen:** Kann verwendet werden, um die Eingangsspannung der Solarzelle sowie die Spannungsniveaus der beiden Speicherkondensatoren zu messen.
- **Versorgungsspannung für Sensorknoten:** Bereitstellen von stabilen Eingangsspannungen für den Sensorknoten mit Hilfe von Spannungsreglern.

3.2.2 Mess- und Kontrolleinheit für permanent versorgte Sensorknoten

Die permanent versorgten Sensorknoten können einzeln über den Web-Zugang aktiviert oder deaktiviert werden, des Weiteren wird die Stromaufnahme jedes einzelnen Sensorknotens über einen Shunt-Widerstand bestimmt.

Die Mess- und Kontrolleinheit für die Sensorknoten besteht aus zwei Teilen:

- Einer Mess- und Steuerkarte, die mit dem Host verbunden ist und von diesem gesteuert wird

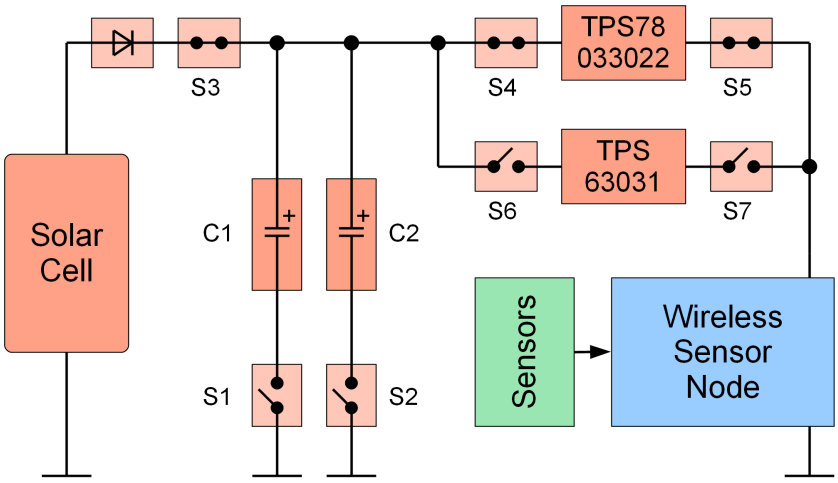


Abbildung 3.2: Schematische Darstellung des Energiegewinnungssystem [19]

- Einer Komponente mit den Aufgaben die digitalen Steuersignale für die gewünschte Aktion umzusetzen (Steuerung der Spannungsversorgungen), sowie die Messwerte der Strommessung entsprechend für die Messeingänge der Karte vorzubereiten.

Dafür soll eine Platine entworfen werden, die eine entsprechende Schaltung inklusive der notwendigen Anschlüsse bereitstellt.

3.2.3 Beleuchtungssteuerung

Die Helligkeit der Leuchtmittel für das Umgebungslicht und das Ereignislicht sind vom Web-Zugang einstellbar.

Die Beleuchtungssteuerung besteht ebenfalls aus zwei Teilen:

- Einer Mess- und Steuerkarte, die mit dem Host verbunden ist und von diesem gesteuert wird
- Einer Schaltung, welche die analogen Steuerausgänge der Mess- und Steuerkarte entsprechend einer LED-Helligkeitsregelung umsetzt

Dafür soll eine Platine entworfen werden, die eine entsprechende Schaltung inklusive der notwendigen Anschlüsse bereitstellt.

3.3 Softwaredesign

Das Konzept für die Software wird in diesem Kapitel beschrieben. Die Software kann in zwei Bereiche unterteilt werden. Einmal in die Serverapplikationen, welche dem Host die benötigten Services zur Verfügung stellen. Der andere Bereich betrifft die Applikation für das drahtlose Sensornetzwerk, welche die oben definierten Anforderungen erfüllen muss.

Es soll darauf geachtet werden, eine hohe Ausfallsicherheit mit einem möglichst stabilen Code zu gewährleisten. Die Maßnahmen dafür waren folgende:

- Benötigte Ressourcen nach Verwendung immer freigeben
- Alle möglichen Rückgabewerte von Funktionen behandeln
- Pufferlängen beachten, vor allem bei Call-By-Reference
- Vermeidung von Typenkonversionen
- Testen auf Modul- und Komponentenebene

Des Weiteren wurde auch darauf geachtet, so wenig CPU-Ressourcen wie möglich zu verwenden, deswegen wurde auf Busy-Waiting soweit als möglich verzichtet.

3.3.1 Serverapplikationen

Die notwendigen Serverapplikationen sollen als Daemons am System laufen. Zwei wesentliche Services sollen implementiert werden:

- Steuerung der Mess- und Steuerkarte mit einem Kommunikationsinterface für den Web-Server und Datenbankzugriff zum Speichern der Messdaten
- Kommunikation mit der Basisstation des drahtlosen Sensornetzwerkes, sowie einem Kommunikationsinterface für den Web-Server und Datenbankzugriff für Konfigurations- und Messdaten.

In Abbildung 3.3 ist das Konzept der serverseitigen Aufgaben dargestellt.

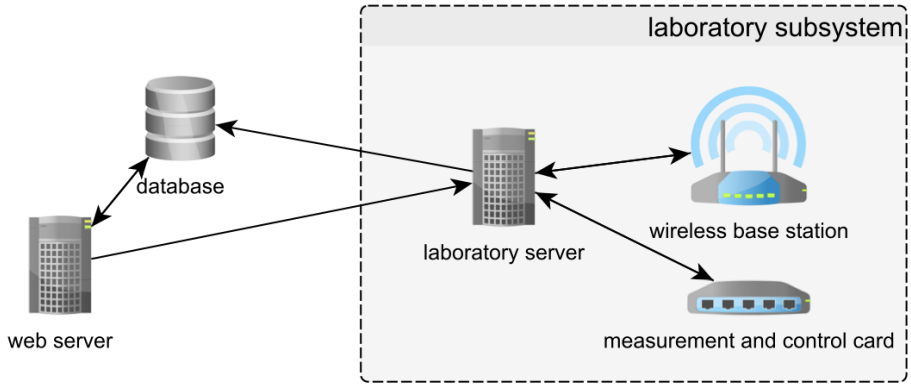


Abbildung 3.3: Konzept der serverseitigen Applikationen [22]

Die Serverapplikationen sollen unabhängig voneinander lauffähig sein. Diese Unabhängigkeit gewährleistet einen autarken Betrieb der einzelnen Applikationen. Es muss darauf geachtet werden mögliche Mehrfachzugriffe auf Ressourcen zu vermeiden beziehungsweise diese korrekt abzuhandeln.

3.3.2 Drahtlose Sensorknoten

Die Sensorknoten sind das Endglied in der Informationskette. Dort werden die Interaktionen vom Benutzer umgesetzt und Informationen werden zum Benutzer übermittelt. Folgende Aufgaben sollen für die Sensorknoten implementiert werden:

- Sensoren auswerten und Spannungsmessungen durchführen (nur EHD-Sensorknoten)
- Informationen/Konfigurationen empfangen, auswerten und Messdaten übermitteln
- Verhalten von Netzwerk-Topologien und Routing-Protokollen umsetzen

Bei der Entwicklung der Applikation muss auf die geringe Prozessorleistung sowie auf die Größe des vorhanden Daten- und Programmspeichers besondere Rücksicht genommen werden.

3.3.3 Basisstation

Die Basisstation dient als Bindeglied zwischen den Sensorknoten und dem Host. Die gesamte Kommunikation wird somit über diesen dedizierten Sensorknoten abgehandelt. Zusätzlich zur Radio-Kommunikation mit den Sensorknoten soll auch die Kommunikation mit dem Host per serieller Schnittstelle implementiert werden. Die zu implementierenden Funktionen sind folgend aufgelistet:

- Nachrichtenpakete vom Host mittels Radio-Kommunikation an Sensorknoten weiterleiten
- Nachrichtenpaket von Sensorknoten mittels serieller Kommunikation an Host weiterleiten
- Konfiguration aller Sensorknoten durch Benutzerinteraktion
- Konfiguration einzelner Sensorknoten durch dedizierte Anfrage

Bei der Entwicklung der Applikation muss auf die geringe Prozessorleistung sowie auf die Größe des vorhanden Daten- und Programmspeichers besondere Rücksicht genommen werden.

3.4 Komponentenauswahl

Dieses Kapitel beinhaltet die Auswahl der Komponenten für das gesamte System. Sorgfältiges Recherchieren der einzelnen Komponenten bezüglich der geforderten Eigenschaften und deren Kompatibilität untereinander ist vorausgesetzt.

3.4.1 Server

Der Server muss wesentliche Aufgaben wie die Verwaltung der Datenbank, den Web-Server und die beiden Daemons (Basisstation und Mess-Kontrolleinheit) übernehmen.

3.4.1.1 Hardware

Die Serverhardware wurde vom IT-Team vom Institut für Technische Informatik zur Verfügung gestellt. Die Spezifikationen entsprechen dem eines aktuellen Netzwerkservers. Durch den Einsatz von zwei gleichwertigen Festplatten soll ein RAID-1-System installiert werden, um eine höhere Ausfallsicherheit des Servers zu gewährleisten. In Tabelle 3.1 sind die verwendeten Komponenten aufgelistet.

CPU	Intel Xeon E3 1220 3,1GHz S1155 8MB Cache
Arbeitsspeicher	2x 4GB DDR3 1333MHz CL9 Kingston KVR1333D3E9SK2/8GI
Mainboard	Intel S1200BTL UC204 S1155 VGA 2xGBLan
Festplatten	Seagate HD 500GB Sata3 7200rpm 64MB ST500NM0011
Netzteil	Thermaltake 630W
Gehäuse	Chieftec MID SH-01B-B-B Schwarz

Tabelle 3.1: Auflistung der Serverkomponenten

3.4.1.2 Betriebssystem

Die Auswahl der Betriebssysteme für den Host ist überschaubar. Aufgrund der vom Treiber unterstützten Betriebssysteme der Mess- und Steuerkarte NI-USB-6211 von National Instruments® [23] [24], reduziert sich die Auswahl auf folgende Systeme:

- Microsoft Windows 2000/Vista/XP/7 sowie Windows Server 2003 und 2008
- Apple Mac OS X 10.8
- Red Hat Enterprise Linux WS 5 und 6
- Scientific Linux 5.x und 6.x
- openSUSE 11.2 und 11.3

Auszuschließen sind jene Betriebssysteme mit Lizenzgebühren. Daher werden Produkte von Microsoft® und RedHat® nicht weiter berücksichtigt. Das *Mac OS X* läuft theoretisch nur auf eigenen Geräten von Apple® und disqualifiziert sich wegen der Einschränkungen damit selbst. Die übrig gebliebenen Systeme sind beide Linux-Distributionen. Das *Scientific Linux* basiert auf der Distribution Red Hat Enterprise Linux (RHEL) und ist vor allem in wissenschaftlichen Einrichtungen wie Universitäten und Forschungsinstitute verbreitet. Mit *openSUSE* kommt eine weit verbreitete Distribution in die engere Auswahl.

Mit der Wahl von *openSUSE*, werden nicht nur die Empfehlungen des IT-Teams vom Institut für Technische Informatik (ITI) berücksichtigt, sondern es wurde auch der Support und Wartung von diesem zugesichert. Der Nachteil dieser Distribution ist das Alter der unterstützten Version von National Instruments®. Die Version 11.3 wurde bereits im Juli 2010 veröffentlicht und wird nicht mehr offiziell gewartet. Der Versuch wenigstens die Version 11.4 mit den Treibern von National Instruments® zu testen war jedoch erfolgreich und wurde somit eingesetzt. Aktuell wird diese Version vom März 2011 ebenfalls nicht mehr gewartet, daher können auch eventuelle Sicherheitslücken nicht mehr durch Aktualisierungen geschlossen werden.

3.4.2 Mess- und Kontrolleinheit

Für diese Komponente wurde ein Produkt für den industriellen Einsatz gewählt. Produkte von National Instruments® wurden auf dem Institut für Technische Informatik (ITI) bereits erfolgreich eingesetzt und getestet. Die Mess- und Steuerkarte NI-USB-6211 [23] [24] verfügt über die benötigten digitalen und analogen Ein- und Ausgänge für den Laboraufbau. In Tabelle 3.2 werden die Anforderungen (Kapitel 3.1.4) mit den Eigenschaften der Mess- und Steuerkarte gegenübergestellt.

	Anforderung		**NI-USB-6211**	
Stromversorgung und Datenanbindung	USB	V2.0	USB	V2.0
Schnittstelle für Treiber	ANSI C	OS-Abhängig	NI-DAQmx	Win, Mac OS X und Linux
Analogeingänge	4	0-5V	16	16bit, 250kS/s, ±10V
Analogausgänge	2	0-10V	2	16bit, 250kS/s, ±10V
Digitaleingänge	k.A.	k.A.	4	0-5,25V
Digitalausgänge	4	0-3,3V	4	0-3,8V

Tabelle 3.2: Gegenüberstellung der Eigenschaften und Anforderungen der Mess- und Steuerkarte

3.4.3 Drahtlose Sensorknoten

Wegen der Anforderungen aus Kapitel 3.1 muss das Sensorknotenpaket aus drei Komponenten (Programmierboard, dedizierte Basisstation und Sensorknoten) bestehen. Evaluierungskits bieten durch die Kombination aus der Hardware und der vorgefertigten Software-Umgebung wesentliche Vorteile in Bezug auf die Entwicklungszeit.

Das Evaluierungskit *Professional Kit* von Crossbow® [25] erfüllt exakt die geforderten Eigenschaften. Die mitgelieferte Software-Umgebung *Moteworks 2.0* [26] basiert auf TinyOS v1.1 [27] und beinhaltet ein große Sammlung an Beispielen und Tutorials. Nachteilig ist jedoch die eingeschränkte Unterstützung von Betriebssystemen, in dem Fall nur Windows XP mit SP3.

3.4.4 Beleuchtung

Als Beleuchtungsmittel haben sich großflächige LED-Module mit flacher Bauform als beste Lösung für den Laboraufbau herausgestellt. Sogenannte LED-Platinen kombinieren viele einzelne LEDs auf einem Modul. Damit reduziert sich nicht nur die Anzahl der Versorgungsleitungen, sondern auch die Anzahl der notwendigen Kühlkörper. Ein einzelner großer Kühlkörper vereinfacht die Montage für den Laboraufbau erheblich. Zur Auswahl standen somit zwei Produkte von *Barthelme* [28]. In Tabelle 3.3 sind beide Module gegenübergestellt. Die Entscheidung fiel zu Gunsten des LED-Moduls mit der höheren Lichtausbeute. Ein Lichtstrom von 850lm ist jedoch nicht ausreichend um genügend Leistung für die EHD-Sensorknoten durch die Solarzellen zu generieren. Der Einsatz von mindestens zwei LED-Modulen pro EHD-Sensorknoten wird demensprechend empfohlen. In Abbildung 3.4 ist das LED-Modul und der Aufbau mit dem Kühlkörper dargestellt.

Artikelnummer	61300432	61300455
Farbton	Warmweiß (3000K)	Kaltweiß (6000K)
Maximaler Strom I_F	300mA	300mA
Vorwärtsspannung U_F	33V	33V
Leistungsaufnahme	10W	10W
Leuchtstärke bei 300mA	750lm	850lm
Abstrahlwinkel	130°	130°
Abmessungen LxBxH	50x50x2.5mm	50x50x2.5mm

Tabelle 3.3: Gegenüberstellung der LED-Module

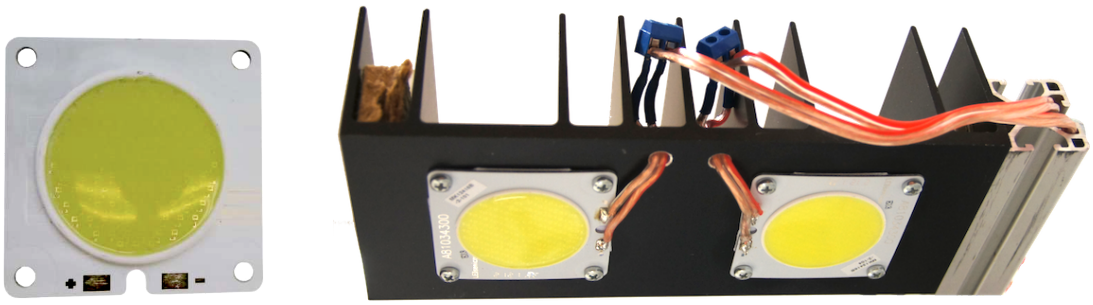


Abbildung 3.4: Darstellung des einzelnen LED-Moduls und dem Aufbau mit Kühlkörper

3.4.5 Solarzellen

Die Auswahl geeigneter Solarzellenmodule für die EHD-Sensorknoten musste evaluiert werden. Aufgrund der verwendeten LED-Leuchtmittel zur Beleuchtung der Module gab es dahingehend keine Erfahrungswerte bezüglich der notwendigen Größe für eine brauchbare U-I-Kennlinie. In Tabelle 3.4 sind die evaluierten Solarzellenmodule gegenübergestellt.

Hersteller	Conrad	Conrad	Conrad	SANYO
Typ	19 13 21	11 04 54	FPST-03	AM-591CAR-SCE
Abmessungen LxH	57x65mm	120x82mm	200x180mm	60x55mm
Wirkungsgrad	k.A.	15%	15%	k.A.
Leerlaufspannung	k.A.	6.6V	6,6V	7,7V
Kurzschlussstrom	k.A.	165mA	450mA	14,8mA
Spannung bei P_{MAX}	5V	6V	6V	5,9V
Strom bei P_{MAX}	81mA	150mA	400mA	26,6mA
Leerlaufspannung Messung	5,45V	6,8V	6,22V	7V
Kurzschlussstrom Messung	12mA	37mA	66mA	16mA

Tabelle 3.4: Gegenüberstellung der evaluierten Solarzellenmodule. Die Kurzschlussströme und Leerlaufspannungen wurden bei gleichen Abstand zu den LEDs (8cm) und im abgedunkeltem Raum aufgenommen.

Als brauchbarstes Solarzellenmodul hat sich die *Conrad FPST-03* herausgestellt. Bei allen anderen Modulen wurden die Vorgaben aufgrund der geringeren Solarzellenfläche nicht erreicht.

3.4.6 Speicherkondensatoren

Der Einsatz von Doppelschichtkondensatoren oder auch Ultra-Kondensatoren genannt, zur Pufferung der überschüssigen Energie hat sich als effizienteste Lösung beim Konzept des Energiegewinnungssystem herausgestellt. Im Gegensatz zur maximalen Ladespannung der Kondensatoren muss auf den Lade- und Entladestrom keine Rücksicht genommen werden und die Regelung der Speicherkondensatoren fällt dadurch wesentlich einfacher aus.

Die gespeicherte Energie in Kondensatoren wird wie folgt berechnet:

$$E = \frac{C \cdot U^2}{2}$$

Der Nachteil der Doppelschichtkondensatoren ist die geringe Spannungsfestigkeit. Die Kondensatoren von SAMWHA Electronic® und Nichicon® verfügen über eine maximale Spannungsfestigkeit von 2,7V [29][30]. Diese Eigenschaft erfordert eine Reihenschaltung der Kondensatoren, dadurch kann die angelegte Spannung auf mehrere Kondensatoren verteilt werden und die Spannungsfestigkeit erhöht sich entsprechend um den Faktor n. Unterschiedliche Isolationswiderstände der einzelnen Kondensatoren können jedoch eine asymmetrischen Spannungsverteilung hervorrufen. Aus diesem Grund können in Serie geschaltete Kondensatoren *symmetriert* werden. Die einfachste Methode ist, jedem Kondensator einen hochohmigen Widerstand parallel zu schalten um eine symmetrische Verteilung der Einzelspannungen zu gewährleisten. Bei der Reihenschaltung reduziert sich die Gesamtkapazität mit:

$$\frac{1}{C_{ges}} = \frac{1}{C_1} + \frac{1}{C_2} \cdots \frac{1}{C_n}$$

Die gespeicherte Energie für in Reihe geschaltete Kondensatoren mit selber Kapazität C und anliegender Einzelspannung U_c wird wie folgt berechnet:

$$E = \frac{1}{2} \cdot \frac{C}{n} \cdot (U_C \cdot n)^2 = \frac{1}{2} \cdot C \cdot U_C^2 \cdot n$$

Die Anzahl der in Serie geschalteten Kondensatoren erhöhen somit die speicherbare Energie um den Faktor n.

Die Laufzeit t der Sensorknoten hängt von der Leistungsaufnahme P_{knoten} und der gespeicherten Energie E_C in den Kondensatoren ab:

$$E_C = P_{knoten} \cdot t \Rightarrow t = \frac{E_C}{P_{knoten}}$$

Die Laborübungen sollen einen gewissen Zeitrahmen nicht überschreiten. Für die Berechnung der Kapazitäten der Speicherkondensatoren wird eine Annahme der Laufzeit von 120s getroffen. Eine konstante Stromaufnahme der Sensorknoten wird mit 28mA bei 3V angenommen [31]. Die dafür benötigte Energie E ergibt sich durch

$$E = P_{knoten} \cdot t = 84mW \cdot 120s = 10,08Ws$$

Die berechneten Kapazitätswerte sollen nur als Richtwert dienen, entsprechend sind vorhandene Verluste der Kondensatoren und der Spannungskonverter nicht weiter berücksichtigt.

Durch Verwendung von zwei Kondensatoren in Reihenschaltung ergibt sich eine Gesamtspannung U_{ges}=5,4V. Die Kapazität C der Kondensatoren wird berechnet mit:

$$C = \frac{2 \cdot E}{U_{ges}^2} = \frac{2 \cdot 10,08Ws}{(5,4V)^2} = 691mF$$

Die Spannungsregler können die Speicherkondensatoren jedoch nicht vollständig entladen. Wegen der Verwendung von Spannungsreglern mit unterschiedlicher minimaler Eingangsspannung, wird U_{min}=3V angenommen. Die reduzierte Energieentnahme wird berechnet mit:

$$C = \frac{2 \cdot E}{U_{ges}^2 - U_{min}^2} = \frac{2 \cdot 10,08Ws}{(5,4V)^2 - (3)^2} = 1F$$

Mit dem Richtwert von 1F kommen Kondensatoren in Frage die eine größere Kapazität besitzen. Aufgrund der einfachen Austauschbarkeit der Speicherkondensatoren auf dem Energiegewinnungssystem, sollte eine brauchbare Sammlung unterschiedlichster Kapazitätswerte angedacht werden. In Tabelle 3.5 sind mögliche Kondensatoren aufgelistet.

Hersteller	Nichicon®	Nichicon®	SAMWHA Electronic®	SAMWHA Electronic®
Kapazität	1F	2,2F	3F	5F
Spannungsfestigkeit	2,7V	2,7V	2,7V	2,7V

Tabelle 3.5: Auflistung der Speicherkondensatoren

3.4.7 Überwachungskamera

Es standen zwei Kameras mit unterschiedlichen Schnittstellen zur Auswahl (USB und Ethernet). In Tabelle 3.6 sind beide Produkte gegenübergestellt.

Hersteller	HP	Axis
Typ	Elite Autofocus Webcam [32]	M1011 [33]
Schnittstelle	USB 2.0	Ethernet 10/100BASE-TX
Video Formate	UVC (Universal Video Class) [34]	H.264, Motion JPEG und MPEG-4 Part 2
Video Auflösung/Bildrate	640x480/30Bilder/s 1280x720/8Bilder/s 1600x1200/5-6Bilder/s	160x120/30Bilder/s 640x480/30Bilder/s

Tabelle 3.6: Gegenüberstellung der Überwachungskameras

Bei der Evaluierung der USB-Überwachungskamera *HP Elite Autofocus Webcam* sind Seiteneffekte durch den hohen Datentransfer über die USB-Schnittstelle aufgetreten. Diese äußerten sich durch Zeitverzögerungen bei der Ansteuerung der Mess- und Steuerkarte NI-USB-6211 von National Instruments®. Diese Zeitverzögerungen können bis zu 1s betragen und bei kleinen Messintervallen zu Problemen führen. Bei der Verwendung der Ethernet-Überwachungskamera von Axis treten keine derartigen Verzögerungen auf. Dementsprechend wurde für den Laboraufbau die Ethernet-Variante *Axis M1011* eingesetzt.

3.5 Zusammenfassung

Diese Kapitel behandelten die Konzepte des Laboraufbaus, der Hardware, der Software und die Auswahl der Komponenten. Beim Laboraufbau wurde auf die Eigenschaften und Aufgaben der Komponenten sowie deren Zusammenspiel eingegangen. Die Dienste am Server übernehmen die Benutzerverwaltung, die Steuerung der Laborübungen und die Verwaltung der Daten auf der Datenbank. Die Basisstation und die Sensorknoten bilden zusammen das drahtlose Sensornetzwerk wobei die Basisstation die Verbindung zum Server bereitstellt. Die drahtlosen Sensorknoten werden noch unterteilt in permanent- und selbstversorgten Sensorknoten um eine brauchbare Anzahl an Szenarien bereitzustellen. Bei der Mess- und Kontrolleinheit handelt es sich um eine Kombination aus Messung der Leistungsaufnahmen, Steuerung des Umgebungs- und Ereignislichts sowie die Steuerung der Versorgungsleitungen der permanent versorgten Sensorknoten. Die Hardwarekonzepte umfassen das Energiegewinnungssystem, die zusätzlichen Schaltungen der Mess- und Kontrolleinheit sowie die Schaltung der Beleuchtungssteuerung. Im Softwarekonzept wurden schließlich die Dienste am Server und die Applikationen für das drahtlose Sensornetzwerk spezifiziert. Zum Schluss wurde noch auf die Auswahl der Komponenten eingegangen. Für den Server wurden die Hardware und das Betriebssystem definiert. Bei der Mess- und Kontrolleinheit wurde eine geeignete Messkarte von National Instruments® ausgesucht. Für das drahtlose Sensornetzwerk wurde auf ein Eval-Kit von Crossbow® ausgewählt. Als Leuchtmittel kommen vier LED-Module mit einer Leistung von je 10W zum Einsatz. Die Solarzellen und Speicherkondensatoren müssen im Echtbetrieb noch genauer evaluiert werden um eine geeignete Auswahl zu erhalten. Als letztes stand noch die Wahl der Überwachungskamera zur Diskussion. Beide Modelle konnten jedoch nur im Echtbetrieb bezüglich ihrer Vor- und Nachteile evaluiert werden. Probleme entstanden bei der Version mit USB-Schnittstelle, der hohe Datenverkehr hatte erhebliche Verzögerungen der USB-Messkarte zur Folge und konnte nicht eingesetzt werden.

4 Implementierung

Dieses Kapitel beschreibt die Umsetzung der Vorgaben für den Laboraufbau sowie die Vorgehensweise bei der Implementierung der Konzepte und Komponenten.

4.1 Drahtloses Sensornetzwerk

Dieser Abschnitt behandelt die Umsetzung der Konzepte für das drahtlose Netzwerk mit den dazugehörigen Komponenten. Im Speziellen thematisiert werden die räumliche Verteilung, die Kommunikation und die Konfiguration der Knoten, sowie Aufbau und Umsetzung der Netzwerk-Topologien und Netzwerk-Routing-Protokolle.

4.1.1 Aufbau

Die räumliche Anordnung der Knoten wurde aus dem Aspekt der Übersichtlichkeit gewählt. Aufgrund der Tatsache, dass die Solarzellen der selbstversorgenden Sensorknoten (EHS-Sensorknoten) in einem optimalen Abstand zu der Lichtquelle gebracht werden müssen, konnte deren Position nur am oberen Bereich des Laboraufbaus festgelegt werden. Im unteren Bereich sind die permanent Versorgten Sensorknoten positioniert. Wie in Abbildung 4.1 ersichtlich ergibt sich ein übersichtlicher Aufbau bei dem auch alle Lichtquellen (Hintergrund- und Event-Licht) mit der Überwachungskamera erfasst werden können.

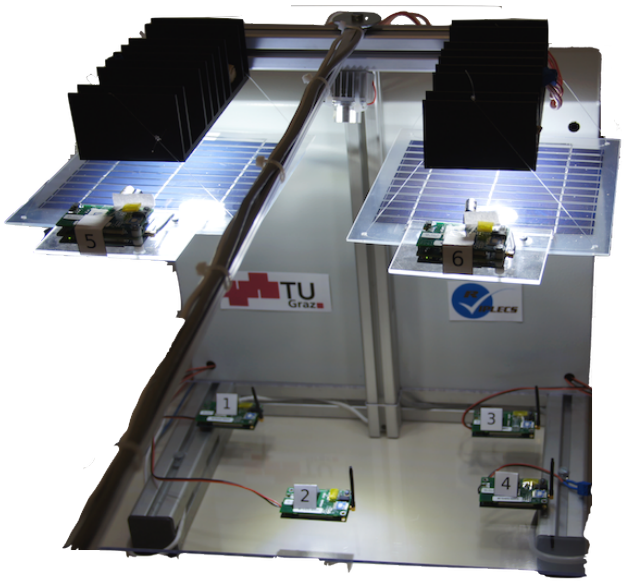


Abbildung 4.1: Anordnung der Knoten im Laboraufbau

Die für die Kommunikation mit dem Host notwendige Basisstation ist in Abbildung 4.1 nicht ersichtlich. Diese befindet sich links vom Laboraufbau in ungefähr 1m Entfernung zu den Knoten. Durch diesen Aufbau bedingten geringen Abständen zwischen den einzelnen Sensorknoten ist eine drahtlose Kommunikation zwischen jedem dieser Sensorknoten gewährleistet. Diese Eigenschaft ist notwendig um die Sensorknoten von der Basisstation aus einfach konfigurieren zu können und um simulierte Netzwerk-Topologien zu etablieren, welche für die Daten-Pakete notwendig sind.

4.1.2 Kommunikation

Nachrichten von der Basisstation sowie der von Sensorknoten werden als Broadcast-Nachrichten versendet. Damit wird gewährleistet, dass alle Sensorknoten die Nachricht erhalten. Bei der Umsetzung von Netzwerk-Topologien wird mit der sogenannten virtuellen Nachbarschaft eine Methodik implementiert, welche eine realistische Konnektivität der Knoten untereinander emuliert. Weiterer Vorteil dieser Methode ist die einfache Nachrichtenverfolgung durch die Basisstation welche in Kapitel 4.3.4.3 (Modul Routing) beschrieben wird.

Als Kommunikationsprotokoll wird das IEEE 802.15.4 verwendet. Die maximale Übertragungsgeschwindigkeit wird mit 250 kbps angegeben [31]. Für diesen Laboraufbau wird der Kanal 3 im 2,4Ghz-Band verwendet.

4.1.3 Virtuelle Nachbarschaft

Die Konnektivität der drahtlosen Kommunikation hängt normalerweise von den örtlichen Gegebenheiten, der räumlichen Verteilung der Sensorknoten und der zu übertragende Datenmenge ab. Bedingt durch die Forderung, unterschiedlichste Topologien zu simulieren sowie durch den Laboraufbau, müssen Möglichkeiten geschaffen werden, um eine Kommunikation zwischen einzelnen Knoten einfach zu etablieren oder zu unterbinden. Um solche Sortierungen von Nachrichten zu etablieren werden zusätzliche Information für jedes Nachrichtenpaket benötigt (genauer erläutert in Kapitel 4.3.5 Radiopakete). Folgende Auflistung zeigt den Teil der zusätzlichen Paketinformationen, die für eine erfolgreiche Sortierung notwendig sind:

Adresse des Absenders: Diese Information dient zur Auswertung des gesamten Pfades durch das Netzwerk.

Adresse des Zieles: Damit wird festgelegt, ob es sich um eine Broadcast-Nachricht für alle Teilnehmer oder einer Nachrichtenübermittlung über das vorab gewählte Routing-Protokoll (Kapitel 4.1.5) für einen bestimmten Teilnehmer handelt.

Adresse vom letzten Netzwerkknoten bei Weiterleitung: Mit dieser Information wird anhand der Nachbarschaftstabelle verifiziert ob es sich um einen sogenannten virtuellen Nachbarn handelt.

Die Nachbarschaftstabellen definieren das Verhalten aller Netzwerkteilnehmer untereinander. Mit der Information der virtuellen Nachbarn sowie den zusätzlichen Informationen aus den Nachrichtenpaketen filtert das Routing-Modul (in Kapitel 4.3.3.2 für Sensorknoten, in Kapitel 4.3.4.3 für die Basisstation) alle Nachrichten über das Netzwerk und gibt sie bei

erfolgreicher Überprüfung an die Applikation weiter. Dieses Modul entspricht somit einer Schicht zwischen der Applikation und dem Kommunikationsmodul. Wie derartige Tabellen aufgebaut sind und wie virtuelle Nachbarn bestimmt werden, wird im nächsten Kapitel genau erläutert.

4.1.4 Netzwerk-Topologien

Um die Knoten mit den entsprechenden Netzwerk-Topologien zu konfigurieren, werden sogenannte Nachbarschaftstabellen verwendet. Diese Tabellen werden in Konfigurationspakete (Abbildung 4.2) verpackt und an jeden Sensorknoten inklusive der Basisstation weitergeleitet.

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Bytes 5	Byte 6
basestation	node 1	node 2	node 3	node 4	node 5	node 6

Abbildung 4.2: Struktur des Nachbarschaftspacket

Jeder Sensorknoten erhält auf diese Weise eine eigene Nachbarschaftstabelle mit den Informationen mit welchen Sensorknoten eine direkte Kommunikation stattfinden kann. Alle Verbindungen im Netzwerk sind unidirektional. Für diese Information muss lediglich ein Byte pro Sensorknoten verwendet werden. In Abbildung 4.3 ist der Aufbau einer Nachbarschaftstabelle ersichtlich.

Bit 0/LSB	Bit 1	Bit 2	Bit 3	Bit 4	Bit 5	Bit 6	Bit 7/MSB
basestation	node 1	node 2	node 3	node 4	node 5	node 6	n.c.

Abbildung 4.3: Struktur der Nachbarschaftstabelle

Etwaige Nachbarschaften werden durch ein gesetztes Bit realisiert, Positionen mit nicht gesetztem Bit werden bei der Kommunikation ignoriert.

Die Auswahl der Netzwerktopologie geschieht durch den Benutzer über den Web-Zugang. Dort wird eine festgelegte Anzahl an möglichen Topologien zur Verfügung gestellt. Die Menge und Art der auswählbaren Topologien wird vorab durch den Administrator der Übungen definiert. Die Netzwerk-Topologien können durch einen Hex-Editor erstellt, sowie bestehende einfach abgeändert werden. Tabelle 4.1 zeigt eine Möglichkeit zum Erstellen einer Topologie mithilfe eines einfachen Hex-Editors in binärer Darstellungsart. Die Struktur entspricht den beiden oben gezeigten Abbildungen 4.2 und 4.3. Die erstellte Topologie wird dann in einer binären Datei am Host gespeichert (genaue Beschreibung in

Kapitel 4.3.2.7 Konfiguration). Das Ergebnis dieser Konfiguration ist in Abbildung 4.4 ersichtlich.

basestation/LSB	node 1	node 2	node 3	node 4	node 5	node 6/MSB
01100000	00011100	00011010	01110110	01101110	01011001	00111001

Tabelle 4.1: Erstellen einer Netzwerktopologie

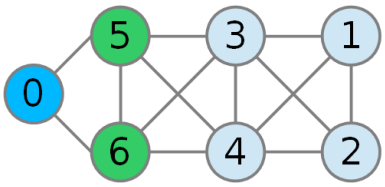


Abbildung 4.4: Beispiel einer Maschen-Topologie. Der dunkelblaue Knoten entspricht der Basisstation, die grünen der EHS-Sensorknoten und die hellblauen der permanent versorgten Sensorknoten.

4.1.5 Netzwerk-Routing

Das sogenannte *Routing* beschreibt die Art der Bewegung von Nachrichten durch ein Netzwerk. In diesem Abschnitt wird auch das sogenannte *Forwarding* behandelt, welches den Entscheidungsprozess von Netzwerkknoten beschreibt an welche Nachbarknoten Nachrichten weitergeleitet werden. Die Verbindung zwischen den Knoten ist unidirektional, damit können Nachrichten in beide Richtungen übermittelt werden. Um die Anforderungen zu erfüllen, eine möglichst umfangreiche Auswahl an Übungen zu generieren, sind drei unterschiedliche Protokolle implementiert:

Direkt zur Basisstation: Jeder Sensorknoten versucht die Nachricht direkt zur Basisstation zu senden. Eine erfolgreiche Übertragung erfolgt nur dann, wenn die Basisstation ein direkter Nachbar ist.

Fluten: Jeder Sensorknoten sendet Nachrichten an all seine direkten Nachbarn. Die Nachricht wird solange weitergeleitet bis ein Sensorknoten die Basisstation als direkten Nachbar hat oder der Sensorknoten die Nachricht bereits verbreitet hat.

Kürzester Weg: Algorithmus zur Feststellung der geringsten Anzahl an Knoten bis zur Basisstation. Für diesen Algorithmus wird ein Distanzvektor verwendet, damit wird die Anzahl an Netzwerknoten (Verbindungskosten) und der Pfad bis zum Ziel für jeden Sensorknoten separat gespeichert. Mit der Bedingung, dass das Ziel in diesem Netzwerk immer die Basisstation ist, ist der Distanzvektor eindimensional implementiert. Der Vektor hat eine Länge von 7 Bytes, für jeden Knoten ist ein Byte reserviert, inklusive der Basisstation. Dementsprechend kann ein Wert von maximal 255 pro Position gespeichert werden. Die Distanzvektoren werden dynamisch aufgebaut und können sich während der Laufzeit dementsprechend ändern. Um eine Neubildung der Vektoren zu starten muss ein sogenanntes Hallo-Paket (hello-package) von der

Basisstation generiert werden. Dieses Paket hat die Aufgabe wie eine Welle alle Pfade im Netzwerk zu erkunden. Schleifenbildungen in vermaschten Netzwerken werden mit Hilfe von Sequenzzählern entgegengewirkt. Abbildung 4.5 zeigt den Verlauf eines Hallo-Pakets durch das Netzwerk, Startpunkt ist immer die Basisstation. Erreicht das Hallo-Paket von der Basisstation den ersten Sensorknoten (Abbildung 4.5a), welcher ein Nachbar sein muss, wird ein Wert von 1 in die Position der Basisstation (Byte 0) im Distanzvektor des Sensorknotens gespeichert, alle anderen Positionen bleiben auf 0. Damit ist ersichtlich, dass nur Werte größer als 0 auf gültige Verbindungen verweisen. Im nächsten Schritt, werden die Verbindungskosten inkrementiert und zu der Adresse von der das Hallo-Paket gesendet wurde im Distanzvektor gespeichert (in Abbildung 4.5b ersichtlich). Abbildung 4.5d zeigt die Vermeidung von Schleifen, Knoten 5 hat bereits ein Hallo-Paket von Knoten 3 erhalten, deswegen wird das Paket nur an Knoten 6 weitergeleitet. Nachdem die Hallo-Paket-Welle das gesamte Netzwerk geflutet hat und die Distanzvektoren fertig aufgebaut sind, können Nachrichten von den Sensorknoten zur Basisstation übertragen werden. Möchte ein Sensorknoten eine Nachricht übertragen, wird im Modul Routing (Kapitel 4.3.3.2) der Distanzvektor nach dem kleinsten Wert, mit der Bedingung größer als 0, durchsucht. Nach einer erfolgreichen Auswertung wird die Nachricht zum Nachbarn mit den geringsten Verbindungskosten versandt. Dieser Vorgang wird solange wiederholt bis damit das Ziel, die Basisstation, erreicht wurde. Für dieses Protokoll muss ein zusätzlicher Parameter im Konfigurationspacket angegeben werden. Damit wird definiert mit welchen Zeitintervall ein Hallo-Packet generiert wird. Ein Zeitintervall kann von 1s bis 255s gewählt werden, 0 definiert eine einmalige Generierung.

4.1.6 Netzwerk-Konfiguration

In diesem Abschnitt wird die Konfiguration der Sensorknoten über das Netzwerk behandelt. Die Vorgabe war ein möglichst flexibles System zu implementieren, bei dem alle Netzwerkparameter und alle verfügbaren Einstellungen am Sensorknoten konfigurierbar sind. Implementiert wurde zwei Konfigurationsarten:

Benutzer-initiierte Konfiguration: Wird durch den Benutzer ausgelöst, alle Knoten werden gleichzeitig konfiguriert.

Knoten-initiierte Konfiguration: Durch eine Anfrage des Sensorknoten wird eine direkte Konfigurationssequenz mit der Basisstation durchgeführt.

Die *benutzer-initiierte Konfiguration* beschreibt eine Art der Konfiguration, die nur über den Web-Zugriff ausgelöst werden kann. Alle Konfigurationsparameter werden vom Benutzer definiert, in Konfigurationspakete verpackt und an die Basisstation weitergeleitet. Die Basisstation stellt diese Pakete als Broadcast-Nachrichten im Netzwerk den Sensorknoten zur Verfügung (detaillierte Beschreibung in Kapitel 4.3.4.4). Die erfolgreich empfangenen Konfigurationspakete werden dann von den Sensorknoten in der Konfigurationssequenz behandelt (detaillierte Beschreibung in Kapitel 4.3.3.5). Nachteil dieser Konfigurationsmethode ist, dass Sensorknoten mit deaktiviertem Radio-Modul (Energiesparmodus) nicht berücksichtigt werden.

Die *knoten-initiierte Konfiguration* beschreibt eine Art der Konfiguration die nur unter bestimmten Zuständen des Sensorknotens ausgelöst werden kann:

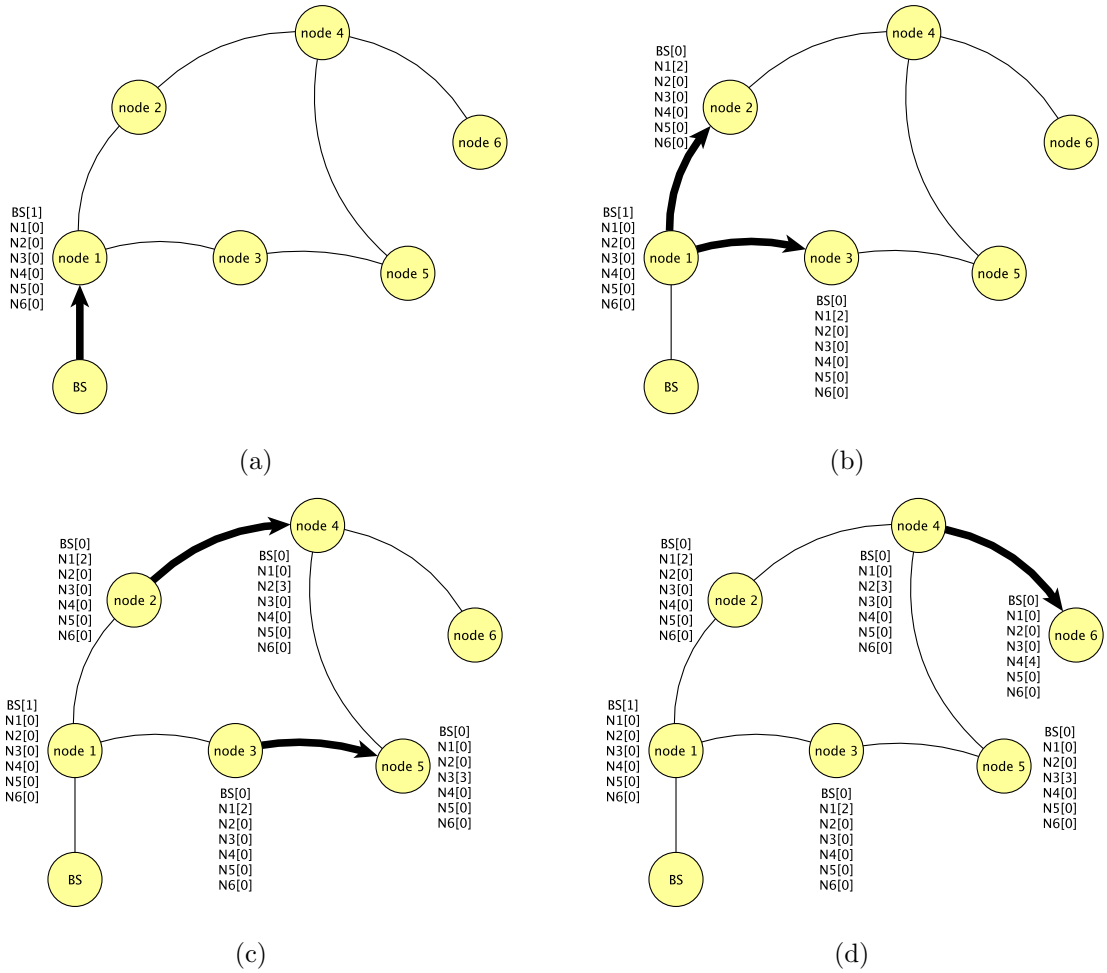


Abbildung 4.5: Ablauf einer Hallo-Packet Sequenz. BS: Basisstation Nx: Sensorknoten

- In der Initialisierungssequenz des Knoten nach dem Bootvorgang
- Im Energiesparmodus des Radio-Moduls
- Fehler in der Konfigurationssequenz im Sensorknoten

Der Sensorknoten übermittelt eine Anfrage für eine Konfiguration direkt an die Basisstation (detaillierte Beschreibung in Kapitel 4.3.3.5). Vorab definierte Netzwerk-Topologien werden dabei nicht berücksichtigt. Im Energiesparmodus des Radio-Moduls wird nach jeder Übertragung der Messdaten eine solche Anfrage gestartet. Grund dafür ist eine mögliche Änderung der Konfiguration durch den Benutzer wenn das Radio-Modul nicht im Empfangsmodus ist und Nachrichten über das Netzwerk nicht empfangen werden können. Die Basisstation übermittelt daraufhin das angeforderte Paket nur an den anfragenden Sensorknoten (detaillierte Beschreibung in Kapitel 4.3.4.4), alle anderen Teilnehmer im Netzwerk werden dadurch nicht beeinträchtigt. Die darauf folgende Konfigurationssequenz im Sensorknoten entspricht der im oben beschriebenen Teil der *knoten-initiierte Konfiguration*.

4.1.7 Messdaten

Die Zusatzplatine für die Sensorknoten von Crossbow® [35] erweitern den Knoten um einige Sensortypen. Bei diesem Laboraufbau wird nur der Helligkeitssensor verwendet und ausgewertet. An den EHD-Sensorknoten werden zusätzliche Spannungsmessungen durchgeführt, um das Verhalten der selbstversorgenden Sensorknoten zu evaluieren. Ausgewertet werden die Spannungsniveaus von der Solarzelle und den beiden Speicherkondensatoren. Die Analog-Digital-Konverter des Atmel ATmega128L [36] haben eine Auflösung von 10Bit, dementsprechend werden alle Messwerte in einem 16Bit-Integer im Nachrichtenpaket übertragen. Die Basisstation übermittelt die Daten an den Host und weiter zur Visualisierung an den Web-Zugang.

Für die Messung kann der Benutzer mehrere Parameter beeinflussen, um den Zusammenhang zwischen Energieverbrauch und Intervalle der Messungen zu evaluieren. Folgende Parameter können konfiguriert werden:

Kommunikationsintervall: Zeitintervall für das Versenden von Messdaten an die Basisstation. Helligkeitswerte in Lux für die permanent versorgten Sensorknoten und zusätzliche Spannungslevels (Solarzelle und beide Speicherkondensatoren) für die EHD-Sensorknoten.

Messungen pro Intervall: Anzahl der Messungen die innerhalb des oben definierten Zeitintervalls durchgeführt werden, die erhaltenen Messwerte werden gemittelt.

Ereignisschwelle: Bestimmt das Helligkeitsniveau bei der das Ereignis für Helligkeitsmessung ausgelöst wird. Bei der Überschreitung dieser Schwelle übermittelt der jeweilige Sensorknoten ein Nachrichtenpaket an die Basisstation, welches dann gesondert im Web-Zugriffsfenster dargestellt wird. Tendenziell langsame Helligkeitsanstiege werden ignoriert, damit kann zum Beispiel ein Lichtkegel einer Taschenlampe im Raum detektiert werden.

4.2 Hardware

In diesem Kapitel wird die Implementierung der zusätzlichen Hardware behandelt. Die Komponenten sind in drei Bereiche unterteilt:

- Das *Energiegewinnungssystem* zur Steuerung und Spannungsmessung der Solarzelle, der Speicherkondensatoren und der Spannungswandler für EHD-Sensorknoten
- Die *Mess-, Kontroll- und Versorgungseinheit* für die permanent versorgten Sensorknoten
- Die *Beleuchtungssteuerung* für das Umgebungslicht und das Ereignislicht

4.2.1 Energiegewinnungssystem

Die Abmessungen der Platine wurde durch den Aufbau der MICAz-Knoten [31] bestimmt. Die Bohrungen für die Befestigungsschrauben sowie die Position des elektrischen Verbindungssteckers mussten exakt umgesetzt werden, um mechanische Spannungen auf den

Platinen zu vermeiden. Die bestückte Platine ist in Abbildung 4.6 ersichtlich. Der Schaltplan ist unter Anhang A einsehbar. Der Aufbau am EHD-Sensorknoten ist in Abbildung 4.7 dargestellt.

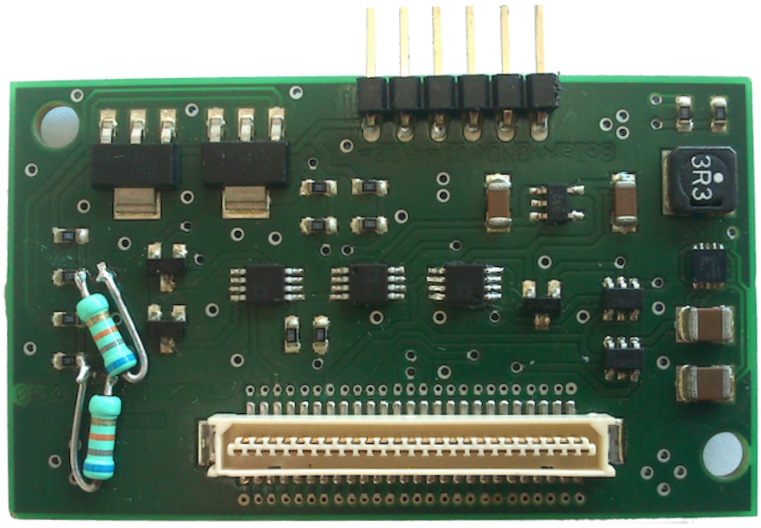


Abbildung 4.6: Darstellung der bestückten Energiegewinnungssystemplatine

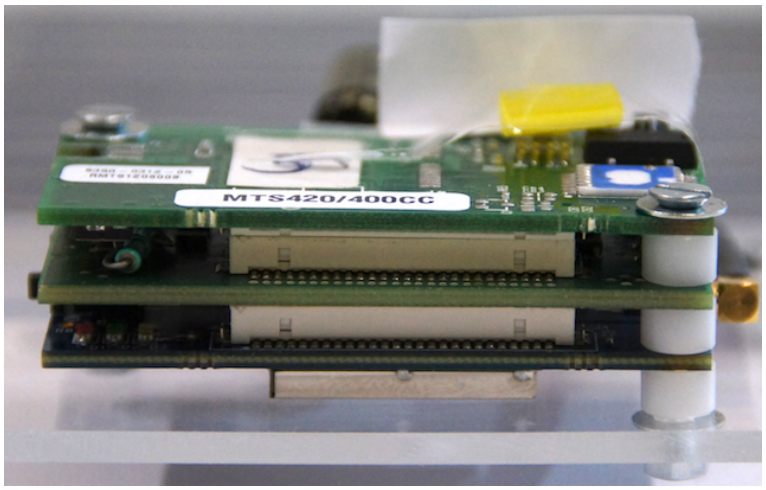


Abbildung 4.7: Die Energiegewinnungssystemplatine wird zwischen der Prozessorplatine und der Zusatzplatine montiert

4.2.1.1 Solarzellensteuerung

Die Steuerung wurde so implementiert, dass möglichst geringe ohm'sche Verluste zwischen dem Ausgang der Solarzelle und der Steuerung der Spannungswandler auftreten. Die

effizienteste Möglichkeit ergibt sich mit CMOS-Schaltern, sogenannte analoge Schalter, welche über einen digitalen Eingang (CMOS-Spannungspegel) steuerbar sind und einen geringen Durchgangswiderstand R_{ON} aufweisen. Um diese Voraussetzungen zu erfüllen wurde der *ADG802* von Analog Devices [37] eingesetzt. Folgend sind die wichtigsten Eigenschaften aufgelistet:

- Schaltertyp: Normal geschlossen (NC)
- Versorgungsspannungsbereich V_{DD}: -0,3V bis +7V
- Maximaler schaltbarer Strom I_D: 400mA (kontinuierlich)
- Widerstand R_{ON}: 0,25Ω (typisch)

4.2.1.2 Spannungsreglersteuerung

Um den Pfad für die Versorgungsspannung möglichst verlustfrei zu halten, wurden hier ebenfalls die bewährten analogen CMOS-Schalter verwendet. Durch Verwendung von nur einem digitalen Signal zu Steuerung müssen beide Schaltertypen (*normal geschlossen* und *normal geöffnet*) eingesetzt werden. Der *ADG802* wurde oben bereits beschrieben, die Eigenschaften vom *ADG801* [37] sind folgend aufgelistet:

- Schaltertyp: Normal offen (NO)
- Versorgungsspannungsbereich V_{DD}: -0,3V bis +7V
- Maximaler schaltbarer Strom I_D: 400mA (kontinuierlich)
- Widerstand R_{ON}: 0,25Ω (typisch)

4.2.1.3 Speicherkondensatorsteuerung

Die Voraussetzungen für das Zuschalten der Speicherkondensatoren sind in Kapitel 3.2.1 beschrieben. Dementsprechend darf der Ladestrom für die Kondensatoren nur so groß werden, dass die Versorgungsspannung V_{DD} nicht unter einen definierten Pegel absinken kann. Wegen dem Einsatz von sogenannten *Low-Side*-Schaltern, werden N-Kanal MOSFETs (Metall Oxyd Schicht Feld Effekt Transistor) verwendet. Da der MOSFET nicht geregelt, sondern nur zwischen Sperrverhalten und Durchgang umgeschaltet werden muss, erfolgt die Ansteuerung des MOSFETs einfacherweise mit einem digitalen Ausgang. Die Ansteuerung der MOSFETs muss so konzipiert werden, dass beim Absinken der Versorgungsspannung V_{DD} unter dem kritischen Wert, der MOSFET zu sperren (R_{DS} steigt an) beginnt und so den Ladestrom des Speicherkondensators begrenzt. Die Spannungspegel der digitalen Ausgänge des Atmel ATmega128L [36] entsprechen genau der Versorgungsspannung V_{DD}. Mit dieser Eigenschaft kann der Spannungspegel des digitalen Ausgangs als Referenzspannung herangezogen werden. Das Regelverhalten wird durch die Threshold-Spannung U_{GSth} eines MOSFETs bestimmt.

Die Auswahl an MOSFETs mit geringem Durchlasswiderstand R_{DSon} beschränkt sich auf jene mit einem hohem Drain-Strom I_D. Diese Eigenschaft wirkt sich jedoch nachteilig auf die Bauteilgröße aus. Je höher der maximale Strom I_D zwischen Drain (D) und Source (S) ist, umso größer sind die Abmessungen des MOSFETs.

Aufgrund der notwendigen Eigenschaften kommt der *PMT21EN* von NXP Semiconductors [38] zum Einsatz:

- Typ: N-Kanal MOSFET
- Maximaler Strom I_D in Durchlassrichtung: 7,4A
- Maximale Spannung U_{DS}: 30V
- Threshold-Spannung U_{GSth}: 1V bis 2,5V (1,5V typisch)
- Durchlasswiderstand R_{DSon}: 18mΩ (typisch)

Die Threshold-Spannung U_{GSth} wurde bewusst kleiner gewählt als die vorgegebene minimal zulässige Versorgungsspannung V_{DD}. Mit einem Spannungsteiler wird dann die benötigte Gate-Source-Spannung U_{GS} am MOSFET für die Einhaltung der Vorgaben angepasst. Die Schaltung ist in Abbildung 4.8 dargestellt. Die bedingte Varianz der Threshold-Spannung U_{GSth} zeigt sich im Betrieb dann als großer Störfaktor und kann das Regelverhalten des MOSFETs in einen nicht brauchbaren Bereich verschieben. Eine grobe Abstimmung des Spannungsteilers kann mit dem typischen Wert für U_{GSth}=1,5V durchgeführt werden. Die minimale Versorgungsspannung wird mit V_{DDmin}=3V angenommen:

$$\frac{U_{GSth}}{U_{DDmin}} = \frac{R_2}{R_1 + R_2} = \frac{1,5V}{3V} = 0,5$$

Mit R_2=100kΩ ergibt sich

$$R_1 = \frac{100k\Omega}{0,5} - 100k\Omega = 100k\Omega$$

Etwaige Anpassungen des Spannungsteilers müssen im Echtbetrieb vorgenommen werden, wobei hier nur der Widerstandswert für R_1 geändert werden soll. Damit wird vermieden dass bei einem zu kleinen Widerstand R_2 gegen Masse, der digitale Ausgang des ATmega128L zu stark belastet werden kann. Die Simulationsergebnisse sind in Abbildung 4.9 dargestellt und zeigen das erwartete Regelverhalten der MOSFETs. Wird der ungeladene Speicherkondensator elektrisch mit der Solarzelle verbunden, wird folgend der Ladestrom soweit begrenzt, dass die Ausgangsspannung des Spannungsreglers nicht unter einen bestimmten Wert abfallen kann. Die Solarzelle wird daher nur soweit belastet, dass der Spannungsregler noch ausreichend versorgt wird um die gewünschte Versorgungsspannung V_{DD} zu erreichen.

4.2.1.4 Spannungsmessungen

Die Spannungsmessungen müssen aufgrund der Ausgangsspannung der Solarzelle, um mit den ADCs des ATmega128L erfasst werden zu können, mit einem Spannungsteiler angepasst werden. Bei den Spannungsmessungen der Speicherkondensatoren muss wegen der Verwendung von Low-Side-Schaltern eine Referenzspannung zur Messung herangezogen werden. Bei geöffneten Schaltern würde ansonsten eine Messung gegen Masse nur die Ausgangsspannung der Solarzelle wiedergeben, jedoch nicht den gewünschten Ladezustand

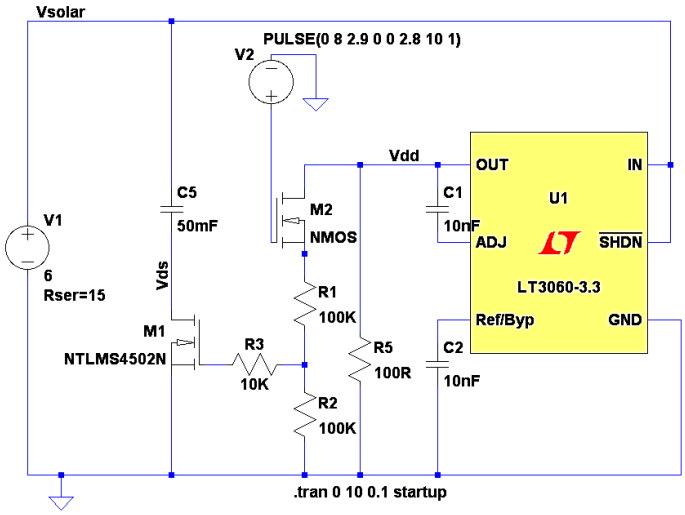


Abbildung 4.8: Schaltung für die Simulation der Kondensatorregelung. Als Spannungskonverter wurde ein LDO von Linear Technologies® eingesetzt, welcher ähnliche Eigenschaften zu dem verwendeten LDO von Texas Instruments® hat. Der MOSFET M2 wird als Schalter verwendet um das Setzen des digitalen Ausgangs im Betrieb zu simulieren. Der Widerstand R_5 wird zu Belastung des LDO mit 30mA verwendet. Die Solarzelle wird vereinfacht durch eine Spannungsquelle mit annähernd selben Innenwiderstand ersetzt.

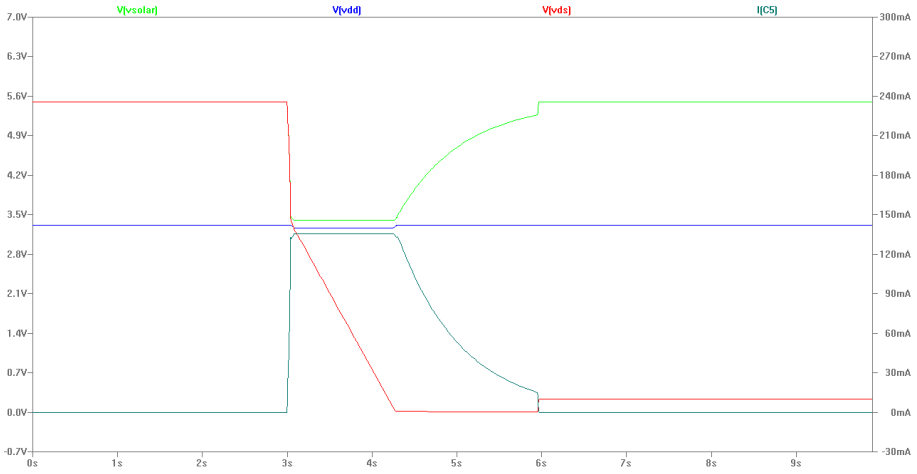


Abbildung 4.9: Simulationsergebnis für die Kondensatorregelung. Die grüne Kennlinie zeigt die Ausgangsspannung der Solarzelle. Die blaue Kennlinie entspricht der geregelten Ausgangsspannung des LDOs. Die rote Kennlinie entspricht dem Spannungsabfall am MOSFET zwischen Drain und Source. Der Ladestrom des Speicherkondensators wird durch die türkise Kennlinie dargestellt.

der Speicherkondensatoren. Die Beschaltung ist unter Anhang A im Schaltplan des EHS-Boards einsehbar. Mit der Referenzspannung U_{Cref} folgt:

$$U_C = U_{Cref} - U_{Cgemessen}$$

Die Spannungsteilerverhältnisse müssen entsprechend zur Berechnung der tatsächlichen Spannung berücksichtigt werden.

4.2.1.5 Spannungsregler

Um die Vorgabe zu erfüllen wurden zwei unterschiedliche Konzepte von Spannungsreglern implementiert, einmal einen Buck-Boost-Konverter und einen Low-Drop Längsregler (LDO). Für den Buck-Boost-Konverter viel die Auswahl auf den von Texas Instruments® hergestellten *TPS63031* [39]:

- Bis zu 96% Effizienz
- 800mA Ausgangsstrom bei 3,3V in Step-Down-Modus (VIN = 3,6V bis 5,5V)
- Bis zu 500mA Ausgangsstrom bei 3,3V im Boost-Modus (VIN > 2,4V)
- Eingangsspannungsbereich von 1,8V bis 5,5V
- Einstellbarer Ausgangsspannungsbereich von 1,2V bis 5,5V

Die Berechnung für die notwendige Induktivität wurde in der Masterarbeit von Leander Bernd Hörmann auf Seite 33 [1] bereits durchgeführt und wurde übernommen.

Für den LDO wurde der *TPS78330220* von Texas Instruments® [40] gewählt:

- Stromaufnahme I_Q: 500nA
- Maximaler Ausgangsstrom I_{OUT}: 150mA
- Dropout Spannung: 200mV bei 150mA
- Einstellbare Ausgangsspannung U_{OUT}: 2,2V oder 3,3V
- Genauigkeit Ausgangsspannung: ±3%

4.2.2 Versorgungsschaltung

Für die Steuerung und Messung wird die Mess- und Steuerkarte *NI-USB-6211* von National Instruments® [23] eingesetzt. Die Steuerung erfolgt durch digitale Ausgänge der Karte. Die Messung wird durch die integrierten ADCs der Karte durchgeführt. Folgende Funktionen wurden in der Versorgungsschaltung implementiert:

- **Steuerung der Spannungsversorgungen:** Die digitalen Signale von der NI-Karte werden zur Steuerung der Smart-Switches benutzt
- **Messung der Stromaufnahmen:** Verstärkung der Spannungsabfälle an den Shunt-Widerständen um Messungenauigkeiten zu vermeiden

Zum Schutz der empfindlichen Bauteile wurde der Verpolungs- und Überspannungsschutz bei allen Eingängen für hohe Stromstärken ausgelegt. Bei inkorrektem Anschluss des Versorgungsnetzteils kann bei zu hoch eingestellter Strombegrenzung dennoch ein Schutz der Bauteile gewährleistet werden. Im Versorgungspfad wird zusätzlich noch eine Schmelzsicherung eingesetzt. Für den Verpolungs- und Überspannungsschutz werden, auf die Eingangsspannungen abgestimmte Suppressor-Dioden verwendet. Der Schaltplan ist im Anhang A dargestellt. In Abbildung 4.10 ist die bestückte Platine ersichtlich.

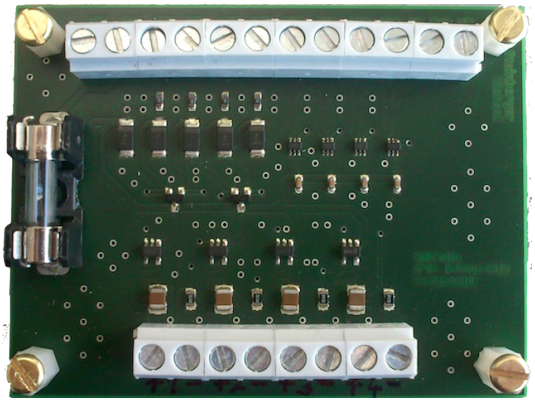


Abbildung 4.10: Darstellung der Versorgungsplatine

4.2.2.1 Steuerung der Spannungsversorgungen

Die Verwendung von Smart-Switches ermöglicht ein einfaches An- und Ausschalten stromführender Leitungen. Die Ansteuerung erfolgt direkt mit den digitalen Ausgängen der Messkarte. Eingesetzt wird der bereits bewährte *FPF2100* von Fairchild Semiconductors [41]:

- Schaltungstyp: High-Side
- Eingangsspannungsbereich von 1,8V bis 5,5V
- Max. schaltbarer Strom: 200mA
- Widerstand im EIN-Zustand: 125mΩ

4.2.2.2 Messung der Stromaufnahmen

Die abfallende Spannung am Shunt-Widerstand wird mit einem geeigneten Operationsverstärker (OP) verstärkt. Optimal dafür geeignet sind sogenannte dedizierte Current-Shunt-Monitore die einen sehr geringen Spannungsdrift und eine hohe Genauigkeit aufweisen. Dafür wurde der *INA210* von Texas Instruments® [42] ausgewählt:

- Versorgungsspannungsbereich U_S: 2,7V bis 26V
- Verstärkungsfaktor: 200
- Genauigkeit: ±1%

4.2.3 LED Treiber

Für die Steuerung wird die Mess- und Steuerkarte NI-USB-6211 von National Instruments® [23] eingesetzt. Mit den analogen Ausgängen der Karte werden die vorhandenen LED-Treiber geregelt. Der Schaltplan ist im Anhang A dargestellt.

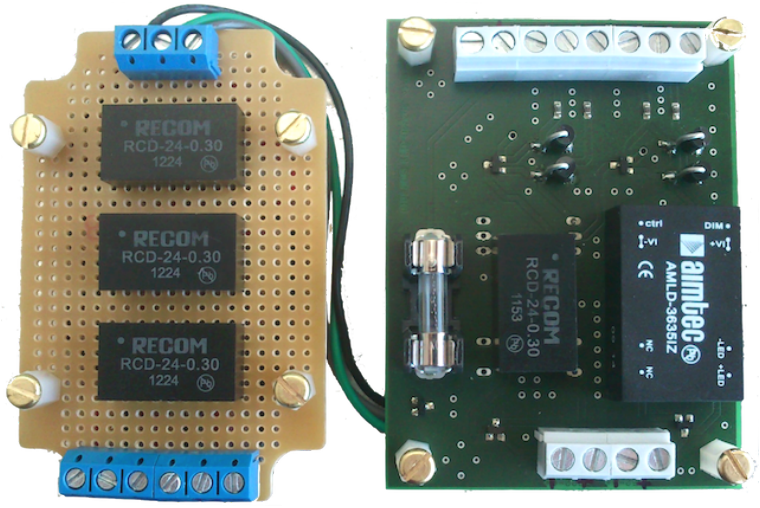


Abbildung 4.11: Darstellung der LED-Treiber-Platine und der Erweiterungsplatine

Die Platine wurde so entworfen, dass LED-Treiber mit unterschiedlichem Pin-Layout verwendet werden können. Mit der Erweiterungsplatine können zusätzliche LED-Treiber eingebunden werden. Die bestückte Platine ist in Abbildung 4.11 dargestellt. Für die verwendeten LEDs (in Kapitel 3.4.4 aufgelistet) werden Treiber mit einem maximalen Ausgangsstrom von 300mA für das Umgebungslicht beziehungsweise 350mA für das Ereignislicht benötigt. In Tabelle 4.2 sind die möglichen LED-Treiber gegenübergestellt.

	Umgebungslicht		**Ereignislicht**	
Hersteller	Recom [43]	Aimtec [44]	Recom [43]	Aimtec [44]
Typ	RCD-24-0.30	AMLD-3630IZ	RCD-24-0.35	AMLD-3635IZ
Ausgangsstrom	300mA	300mA	350mA	350mA
Ausgangsspannung	2-35V	2-32V	2-35V	2-32V
Eingangsspannung	4,5-35V	5-36V	4,5-35V	5-36V
Steuerspannung	0-4,5V	0-4V	0-4,5V	0-5V
Kurzschluss- und Überspannungsschutz	Ja	Nein	Ja	Nein

Tabelle 4.2: Übersicht LED-Treiber

4.2.4 Schaltkasten

Im Schaltkasten sind alle Komponenten inklusive der Mess- und Steuerkarte von NI übersichtlich angeordnet. Abbildung 4.12 zeigt den geöffneten Schaltkasten mit kompletter Verkabelung aller Komponenten.

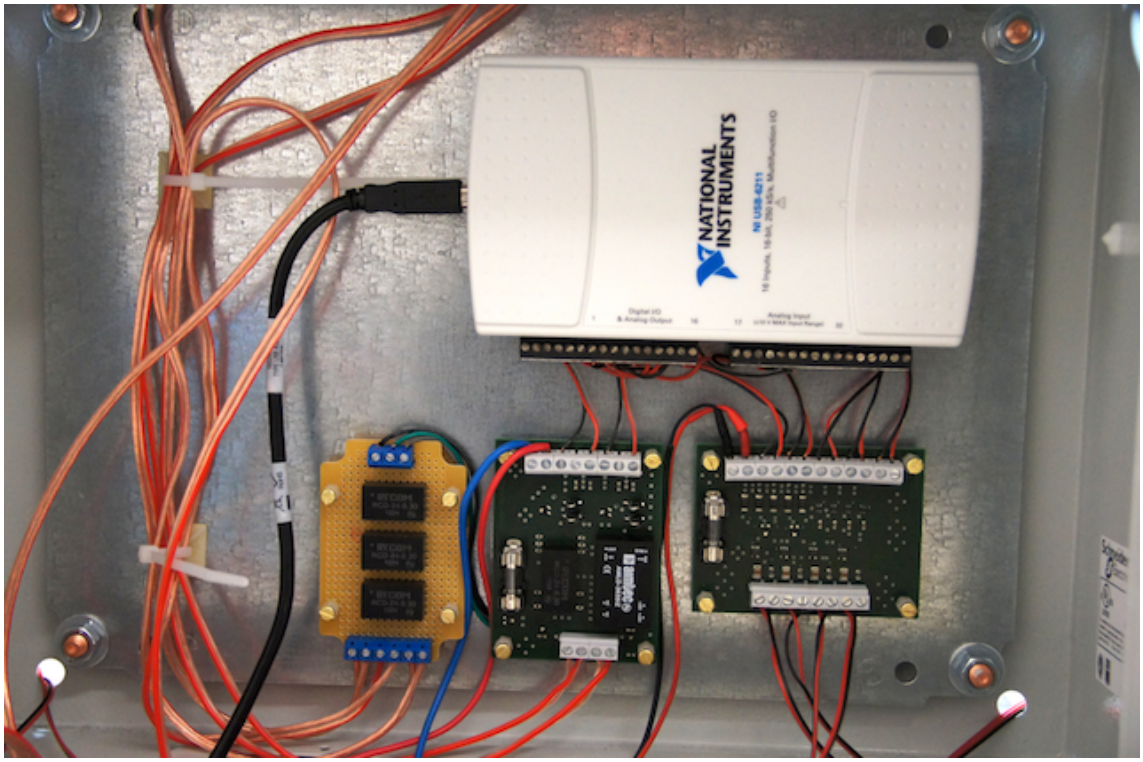


Abbildung 4.12: Darstellung des Schaltkastens inklusive aller Komponenten

4.3 Software

In diesem Abschnitt wird die Implementierung der Software für das Remote-Lab behandelt. Die Implementierung wurde in vier Komponenten unterteilt. Die beiden Daemons sind als Serverapplikationen implementiert, die beiden anderen Komponenten für den Bereich der drahtlosen Sensorknoten.

Die Daemons besitzen jeweils nur einen einzigen Prozess, dadurch muss auf Mehrfachzugriffe auf Ressourcen, wie im Kapitel 3.3.1 beschrieben, keine Rücksicht genommen werden.

4.3.1 Daemon Mess- und Kontrolleinheit

Diese Komponente hat zwei wesentliche Aufgaben, erstens die Konfiguration und Handhabung der Mess- und Kontrolleinheit von National Instruments® und zweitens die Akquisition und Verwaltung der Messdaten. Um die Anforderungen der zeitlichen exakten Messzyklen in einem einzigen Prozess zu entsprechen, wird die Restzeit (Wartezeit) für jeden Zyklus neu berechnet. Die Konfiguration des Daemons wird teilweise zum Kompilierungszeitpunkt sowie zur Laufzeit (durch Startparameter) festgelegt und ist im Kapitel 4.3.1.7 Konfiguration ersichtlich. Durch die Verwendung von Sockets für die Kommunikation zwischen Daemon und dem Web-Server wird eine stabile Kommunikation sowie ein ortsunabhängiger

Betrieb möglich, einzige Voraussetzung dafür ist eine Netzwerkverbindung. Durch diese Trennung können geforderte Sicherheitsvorschriften der IT einfacher umgesetzt werden. In Abbildung 4.13 sind die implementierten Module ersichtlich.

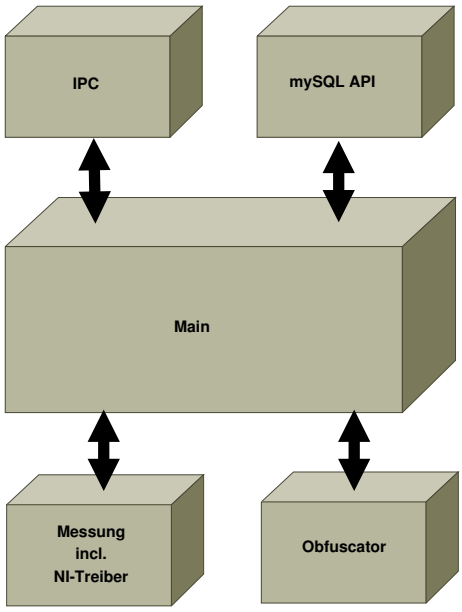


Abbildung 4.13: Blockschaltbild Daemon National Instruments® Mess- und Kontrolleinheit

4.3.1.1 Modul Main

Dieses Modul beinhaltet die Kontrollstruktur der Komponente und kann in zwei Abschnitte unterteilt werden, folgend werden auch die dazugehörigen Dienste aufgelistet:

- Konfiguration und Konnektivität
  - Auswertung der Startparameter und Konfiguration der Module
  - Etablieren der Verbindungen
- Endlosschleife zum sequenziellen Abarbeiten der Dienste
  - Messdaten akquirieren und in Datenbank speichern
  - Abarbeitung der Timeline
  - Socket-Kommunikation mit Clients und Aktionen ausführen

Mit der *Timeline* können Helligkeitsverläufe oder Ausfälle von einzelnen Sensorknoten simuliert werden. Diese Abläufe sind zeitlich koordiniert und werden in der Datenbank als eine Abfolge von Aktionen gespeichert [22].

Im Flussdiagramm in Abbildung 4.14 sind die Abfolge sowie die Dienste der einzelnen Abschnitte dargestellt.

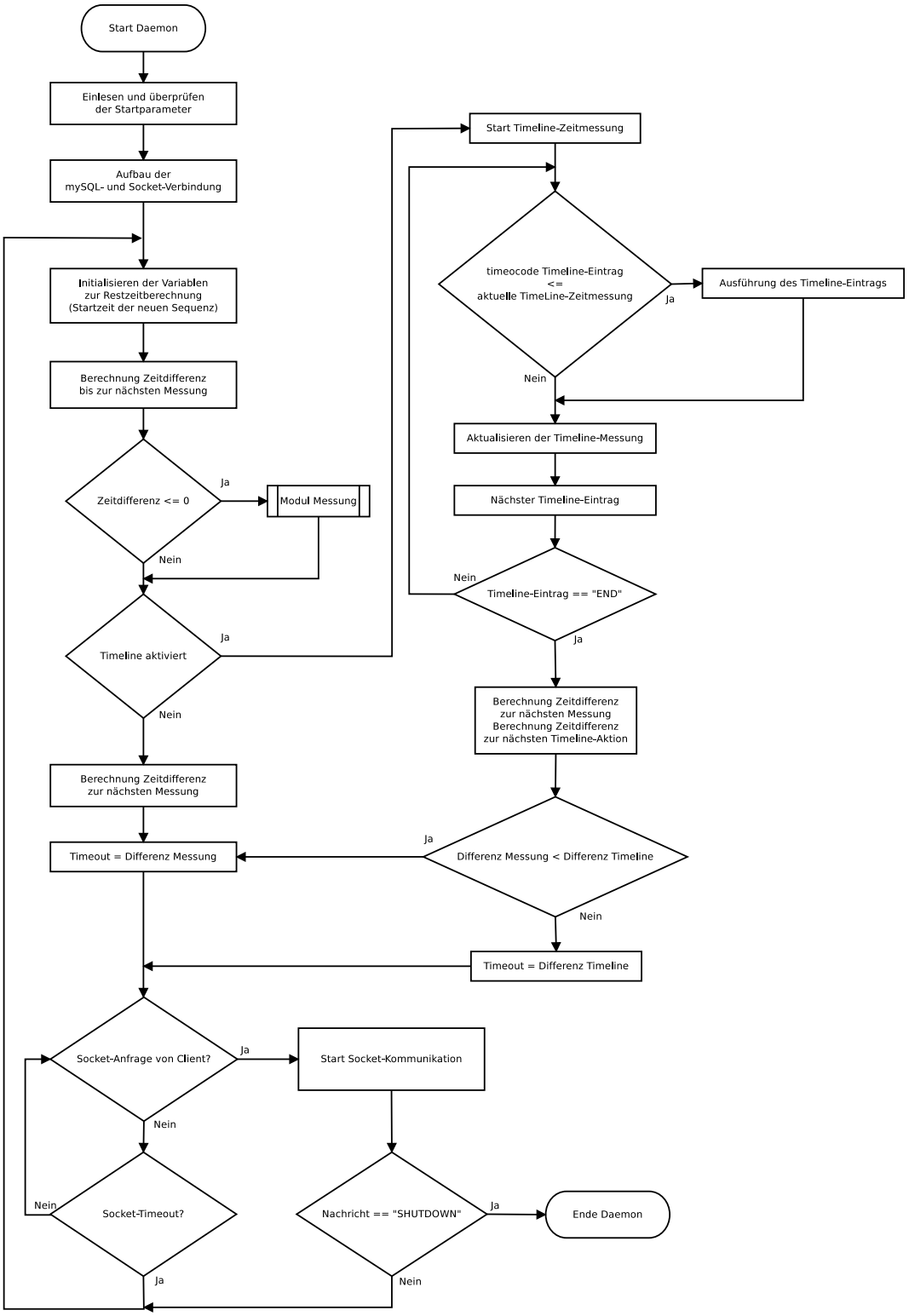


Abbildung 4.14: Flussdiagramm Mess- und Kontrolleinheit Modul Main

In Abbildung 4.15 ist die Socket-Kommunikation ausführlich dargestellt. Die verwendeten Socket-Nachrichten sind in Kapitel 4.3.7 beschrieben.

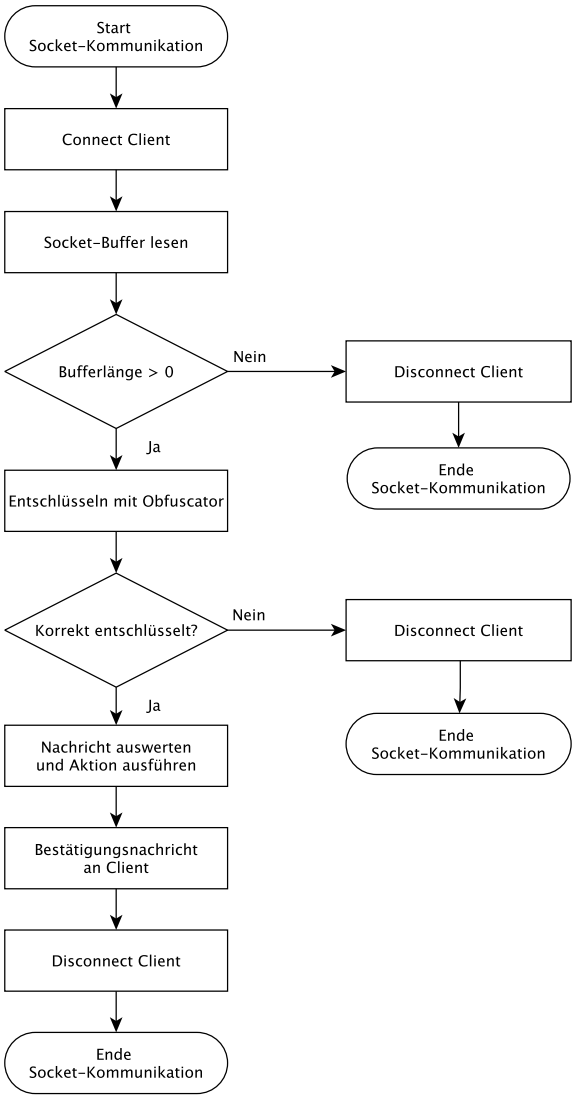


Abbildung 4.15: Flussdiagramm Mess- und Kontrolleinheit Modul Main Socket-Kommunikation

Die Struktur der *Timeout*-Berechnung ist in Abbildung 4.14 nicht ausreichend erkennbar. Aufgrund der dynamischen Berechnung der Restzeit bis zur nächsten Aktion ist in Abbildung 4.16 der chronologische Ablauf der Dienste sowie die benötigten Zeiten ersichtlich. Bei den Sequenzen *Messung* und *Timeline* wird die benötigte Zeit gemessen und die Zeitdifferenz bis zur nächsten Aktion bestimmt. Abhängig davon welche Aktion als nächstes ausgeführt werden muss, wird diese als Referenz zur Berechnung der Restzeit herangezogen. Die Restzeit setzt sich zusammen aus der benötigten Zeit der Dienste und

der Zeitdifferenz bis zur nächsten Aktion (Messung oder Timeline-Event). Die berechnete Restzeit wird in Folge dann als *Timeout*-Wert für die `select()`-Funktionalität [45] [46] im Modul Inter Process Communication (IPC), welches für die Socket-Kommunikation zuständig ist, verwendet.

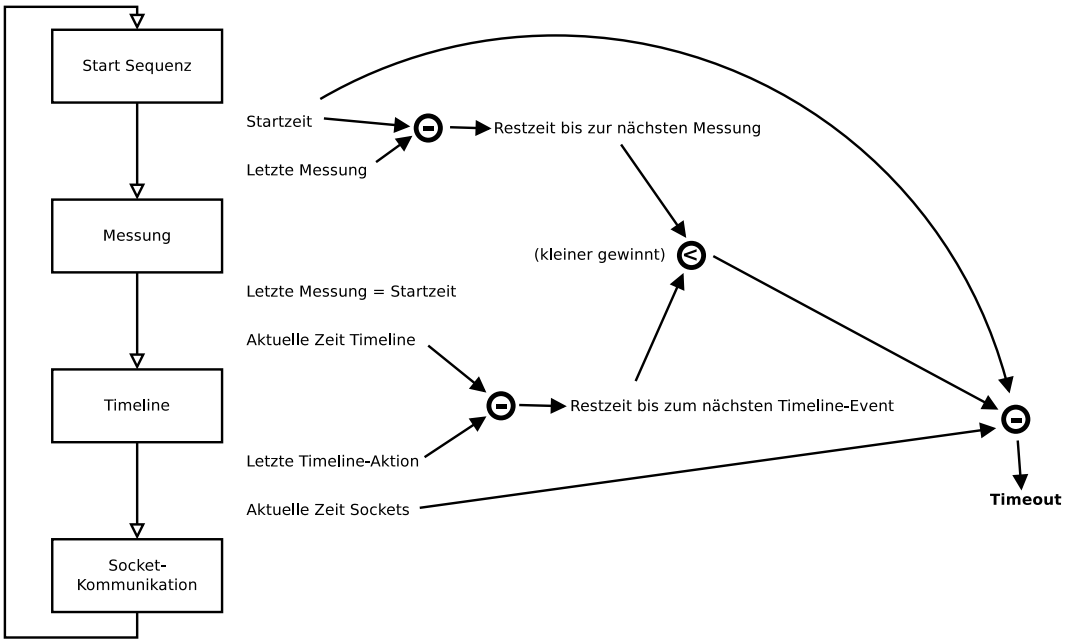


Abbildung 4.16: Flussdiagramm Mess- und Kontrolleinheit Modul Main mit Restzeitberechnung

Wird ein Socket-Ereignis detektiert, wird der Vorgang des *Timeouts* unterbrochen und die Socket-Kommunikation abgearbeitet. Tritt diese Situation ein, ist noch nicht ausreichend Zeit vergangen um die nächste *Messaktion* bzw. *Timeline*-Aktion auszuführen. Wie in Abbildung 4.16 ersichtlich kann dadurch ein kompletter Schleifendurchlauf ohne eine Aktion erfolgen und nur das *Timeout* wird neu berechnet. Dadurch wird gewährleistet, dass alle Socket-Ereignisse sowie Mess- oder Timeline-Aktionen ohne relevante Zeitverzögerungen in einem einzigen Prozess abgearbeitet werden können.

In den Tabellen 4.3 und 4.4 sind die implementierten Funktionen für das Modul Main aufgelistet.

int checkArgvList(int argc, char *argv[], char *param, void **value, uint number)	
Beschreibung	Überprüft die Kommandozeilen-Parameter (argv) mit den festgelegten Parametern für den Daemon in `daemonconf.h`. Die spezifischen Parameter sind in Kapitel 4.3.1.7 Konfiguration beschrieben.
Parameter	`argc`...von der Funktion `main()` übergebener Parameter für die Anzahl der Kommandozeilen-Parameter `argv`...von der Funktion `main()` übergebener Zeichenketten-Array mit den Kommandozeilen-Parametern `*param`...Zu überprüfende Zeichenkette `**value`...Speicherort für Zahlenwert oder Zeichenkette entsprechend des gesuchten Parameters `number`...`0` für Zeichenkette, `1` für Zahlenwert
Rückgabewert	`-1`: Fehler bei Konvertierung für Zahlenwert `0`: Gesuchter Parameter wurde nicht in Parameter-Liste (argv) gefunden `1`: Auswertung erfolgreich
int getTimeStamp(char *timeStampVal)	
Beschreibung	Konvertierung der Systemzeit in definiertes Format zur Speicherung in Datenbank
Parameter	`timeStampVal`...Pointer auf Zeichenkette für das konvertierte Zeitformat
Rückgabewert	`EXIT_FAILURE`: Fehler bei Akquisition der Systemzeit oder bei Formatkonvertierung `EXIT_SUCCESS`: Konvertierung erfolgreich
void writeMysqlEventLog(char *content, char *type)	
Beschreibung	Übermittelt die Information eines Ereignis inklusive der aktuellen Systemzeit an die Datenbank
Parameter	`content`...Zeichenkette für das Ereignis im sogenannten *localization-format* (Tabelle 4.6) `type`...Art des Ereignis (Tabelle 4.7)

Tabelle 4.3: Schnittstellen allgemein für Modul Main für Daemon Mess- und Kontrolleinheit

void enableNode(int node)	
Beschreibung	Aktiviert den gewünschten Sensorknoten und führt den entsprechenden Datenbankeintrag durch
Parameter	`node`...ID des Sensorknotens
void disableNode(int node)	
Beschreibung	Deaktiviert den gewünschten Sensorknoten und führt den entsprechenden Datenbankeintrag durch
Parameter	`node`...ID des Sensorknotens
void setAmbLightLvl(int level)	
Beschreibung	Setzt die Helligkeit für das Hintergrundlicht und führt den entsprechenden Datenbankeintrag durch
Parameter	`level`...Helligkeitswert
void setEventLightLvl(int level)	
Beschreibung	Setzt die Helligkeit für das Ereignislicht und führt den entsprechenden Datenbankeintrag durch
Parameter	`level`...Helligkeitswert
void setTimelineStart(void)	
Beschreibung	Aktiviert die Timeline-Funktionalität und führt den entsprechenden Datenbankeintrag durch
void setTimelineStop(void)	
Beschreibung	Deaktiviert die Timeline-Funktionalität und führt den entsprechenden Datenbankeintrag durch
void getMeasurementData(void)	
Beschreibung	Startet die Strommessung für die Sensorknoten. Das Ergebnis wird in der Datenbank abgelegt.
int getTimelineAndExecute(uint time)	
Beschreibung	Auslesen der Timeline-Daten zur übergebenen Zeit aus der Datenbank und Ausführung der Aktion
Parameter	`time`...Zeitpunkt des Timeline-Eintrags
Rückgabewert	`EXIT_FAILURE`: Fehler wegen leeren oder fehlerhaften Datenbankeintrag `EXIT_SUCCESS`: Timeline-Aktion erfolgreich durchgeführt

Tabelle 4.4: Schnittstellen Events für Modul Main für Daemon Mess- und Kontrolleinheit

4.3.1.2 Modul Inter Process Communication (IPC)

In diesem Modul wurde die vollständige Socket-Kommunikation unter Verwendung der POSIX Socket API [47] [46] implementiert. Folgende Dienste werden unterstützt:

- Verbindung von Server und Client etablieren und beenden
- Reagieren auf Client-Nachrichten durch `select()`-Funktionalität
- Unidirektionale Konnektivität zwischen Server und Client

Der Start der *Server*-Funktionalität für die Socket-Kommunikation (`ipcConnectServer()`) muss nur einmal durchgeführt werden und bleibt bis zur Beendigung des Daemons bestehen. Mit der Systemfunktion `socket()` aus `sys/socket.h` wird eine Stream-Kommunikation initialisiert, dafür muss der Funktion die Konstante `SOCK_STREAM` als Parameter übergeben werden. Durch das `bind()` wird dem Server eine Socket-Adresse zugewiesen und durch `listen()` der Empfang von Sockets aktiviert. Tritt während der Initialisierung ein Fehler

auf wird der Socket-Dienst mit `close()` beendet. Der Ablauf ist mit dem Flussdiagramm in Abbildung 4.17 dargestellt.

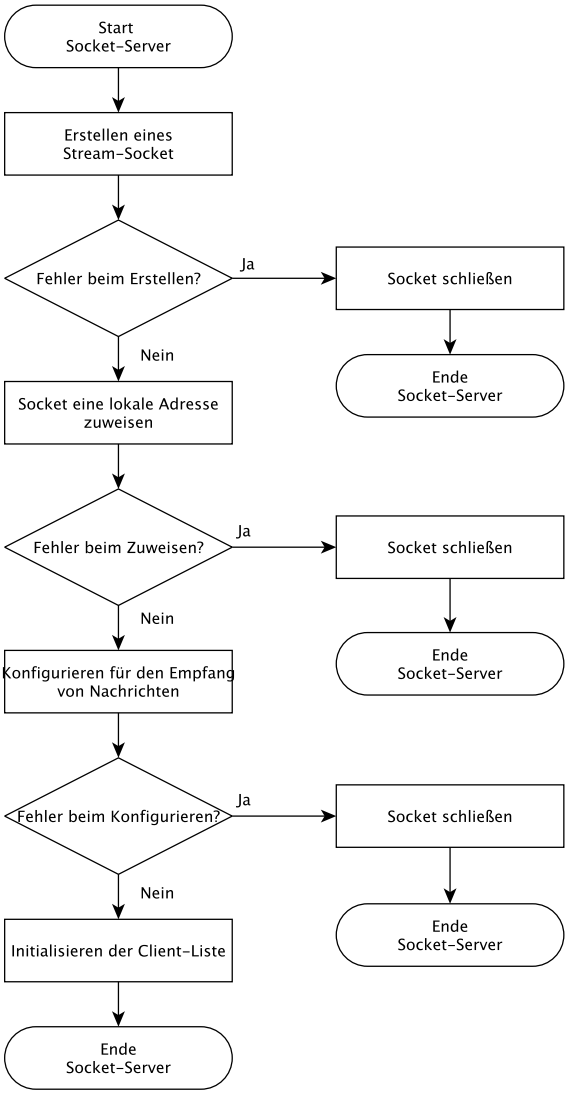


Abbildung 4.17: Flussdiagramm Mess- und Kontrolleinheit Modul IPC - Serververbindung

Das Überwachen der Socket-Dateideskriptoren und das Akzeptieren von *Client-Verbindungen* werden in der Funktion `ipcConnectClient()` zusammengefasst. Die `select()`-Funktion [45] [46] bietet sich aufgrund ihrer Eigenschaften für eine Kommunikation mit nicht-blockierenden Sockets an. Damit kann eine große Anzahl an Clients mit einer einfachen Schnittstelle organisiert werden. Die Socket-Dateideskriptoren der Clients werden überwacht ob Nachrichten eintreffen und gegebenenfalls eingelesen. Eine weitere Funktionalität von `select()` hat sich für dieses Modul angeboten, die des *Timeouts*. Dies ermöglicht, über ein vorab definiertes Zeitfenster zu warten und auf ein Signal oder

Ereignis zu reagieren, ohne unnötige Prozessorlast zu verursachen. Der Ablauf kann folgendermaßen beschrieben werden: Mit `FD_SET()` werden zuerst alle bereits akzeptierten Clients auf die Select-Überwachungsliste gesetzt, diese Liste und der übergebene Timeout-Wert werden dann der `select()`-Funktion übergeben. Wird vor dem Ablauf des Timeouts ein Ereignis detektiert, in dem Fall eine Verbindungsanfrage von einem Client, kann das mit `FD_ISSET()` überprüft werden. Ist das der Fall wird dieser zur Client-Liste hinzugefügt. Tritt jedoch kein Ereignis auf und das Timeout ist abgelaufen werden alle Einträge in der Client-Liste auf die Anzahl ihrer Timeouts überprüft (`IPC_MAX_TIMEOUTS_READ` in `daemons/daemonconf.h`) und gegebenenfalls aus der Liste entfernt. Damit können Kommunikationsprobleme mit Clients erkannt werden die über eine definierte Zeitdauer keine Nachricht übermittelt haben. Der Ablauf ist mit einem Flussdiagramm in Abbildung 4.18 dargestellt.

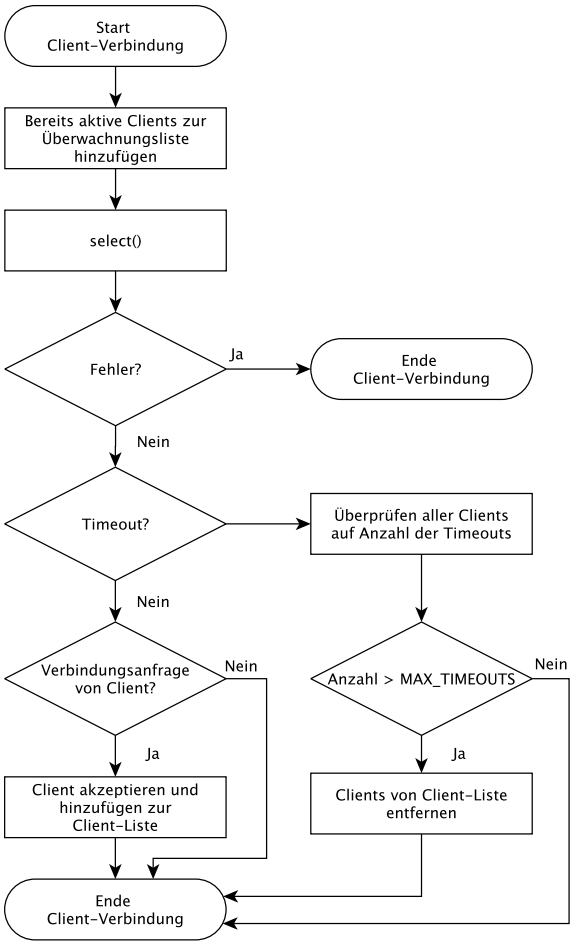


Abbildung 4.18: Flussdiagramm Mess- und Kontrolleinheit Modul IPC - Clientverbindung

Falls das Verbinden mit dem Client erfolgreich war, muss die Nachricht mit `ipcRead()` eingelesen werden. Zuerst wird mit `FD_ISSET()` die Client-Liste geprüft welcher Client eine

Nachricht übermitteln möchte. Das Einlesen der Nachricht geschieht durch ein `read()` auf den Socket-Dateideskriptor. Jede erfolgreiche Anfrage muss mit einem `ACK` oder `NACK` bei einem Fehler bestätigt werden. Das wird über `ipcWrite()` mit einem `write()` auf den Socket-Dateideskriptor durchgeführt. Nach der Bestätigungsnachricht oder nach einem Fehler in der Kommunikation wird der Client mit `ipcDisconnectClient()` aus der Client-Liste entfernt und die Verbindung mit `close()` beendet.

Die implementierten Schnittstellen für das Modul IPC sind in Tabelle 4.5 aufgelistet.

int ipcConnectServer(int port)	
Beschreibung	Startet dem Serverseitige Socket-Kommunikation
Parameter	`port`...lokaler Port für den Server, sollte höher als 5000 sein
Rückgabewert	`EXIT_FAILURE`: Fehler bei Verbindungsaufbau
	`EXIT_SUCCESS`: Erfolgreicher Start des Dienstes
void ipcDisconnectServer(void)	
Beschreibung	Beendet die serverseitige Socket-Kommunikation
int ipcConnectClient(struct timeval timeout)	
Beschreibung	Verbindet und überwacht alle aktiven Clients. Timeout-Funktionalität durch die `select()`-Funktion.
Parameter	`timeout`...Wartezeit für die `select()`-Funktion im Format `struct timeval`
Rückgabewert	`-1`: Fehler bei Verbindungsaufbau
	`0`: Kein Ereignis innerhalb der Wartezeit
	`1`: Verbindungsanfrage vom Client erfolgreich, bereit zum Einlesen der Nachricht
void ipcDisconnectClient(void)	
Beschreibung	Entfernt Client nach erfolgreichen Einlesen der Nachricht von der Client-Liste und beendet die Socket-Verbindung
int ipcRead(char *buffer)	
Beschreibung	Einlesen der Nachricht von anfragenden Client
Parameter	`buffer`...Pointer auf Buffer für die Nachricht
Rückgabewert	`-2`: Socket ist blockiert und Nachricht kann nicht empfangen werden
	`-1`: Fehler Beim Einlesen der Nachricht
	`>=0`: Länge der Nachricht in Bytes
int ipcWrite(char *buffer)	
Beschreibung	Übermittlung einer Nachricht an den aktuellen Client bei dem zuvor ein `ipcRead()` durchgeführt wurde
Parameter	`buffer`...Pointer auf Nachrichtenbuffer
Rückgabewert	`EXIT_FAILURE`: Fehler in Übertragung
	`EXIT_SUCCESS`: Nachricht erfolgreich übermittelt

Tabelle 4.5: Schnittstellen Modul IPC

4.3.1.3 Modul MySQL

Das Modul wird verwendet, um durchgeführte Aktionen, Ereignisse oder Fehlermeldungen an die Datenbank zu übermitteln. Für die Konfiguration der Sensorknoten oder für die Bearbeitung der Timeline-Funktionalität müssen Informationen von der Datenbank gelesen werden. Der Zugriff auf die MySQL-Datenbank wurde mit der MySQL C API [48] implementiert. Bei kontinuierlich auftretenden Zugriffen auf die Datenbank werden sogenannte *Prepared-Statements* verwendet. Im Vergleich zum Zugriff mit Queries kann damit die Prozessorlast auf dem Datenbank-Server wesentlich reduziert werden. Dieser

optimierte Datenbankzugriff findet beim Archivieren von Messwerten (Measurement-Log) und Ereignissen (Event-Log) Verwendung.
Das Modul beinhaltet mehrere Aufgabenbereiche:

- Verwalten der Verbindung mit der MySQL-Datenbank
- Auswerten und verwalten der Timeline-Daten
- Speichern der Messdaten in der MySQL-Datenbank
- Übermitteln aller Ereignis-Nachrichten an die MySQL-Datenbank

Dem Benutzer stehen im Web-Zugang mehrere Sprachen zur Auswahl. Um eine dynamische Lokalisierung durchführen zu können, müssen die Nachrichten des *Event-Log* (Ereignis-Nachrichten) ein spezielles Format beinhalten. Die Formatierung besitzt zwei Schlüsselwörter und wird wie folgt verwendet:

```
{@lang/Section/Key}
```

Die Definitionsklasse der Nachricht wird mit `Section` festgelegt. Mit dem Schlüsselwort `Key` werden die vorab definierten Nachrichten übergeben. Werden zusätzliche Zahlenwerte für die Nachricht benötigt, können diese einfach an die Zeichenkette mittels des Steuerzeichens `?` angehängt werden. Mit Verwendung von `&` als Seperator, können beliebig viele Zahlenwerte angefügt werden.

```
{@lang/Section/Key?param1=value1&param2=value2&...}
```

Die zur Verfügung stehenden Ereignis-Nachrichten für den Daemon Mess- und Kontrolleinheit sind in Tabelle 4.6 aufgelistet. Die Möglichkeit, Nachrichten nachträglich zu ändern oder hinzuzufügen besteht. **Hinweis:** Zu beachten ist, dass die Nachrichtenformate mit dem Host äquivalent sind.

Im Web-Zugang können die Nachrichten in verschiedensten Darstellungsarten abgebildet werden, Farben und Schriftgröße sind dabei variabel. Dementsprechend wurde vorab für jede Art von Nachricht eine eigene Darstellungsart spezifiziert. Sogenannte *Event-Log*-Typen (siehe Tabelle 4.7) werden mit den jeweiligen Ereignis-Nachrichten in der Datenbank gespeichert.

Statusmeldungen
@lang/status/enableNode?node=%d
@lang/status/disableNode?node=%d
@lang/status/setAmbLightLvl?level=%d
@lang/status/setEventLightLvl?level=%d
@lang/status/StartTimeline
@lang/status/StopTimeline
@lang/status/getMeasurementData
@lang/status/loadBasestationConfiguration?id=%d
@lang/status/setBasestationConfiguration?id=%d
@lang/status/eventLightFired?node=%d&lux=%d
@lang/status/meteringData?node=%d&vsolar=%d&vcap1=%d&vcap2=%d&lux=%d
@lang/status/nodeError?node=%d&error=%d
Fehlermeldungen
@lang/error/enableNode?node=%d
@lang/error/disableNode?node=%d
@lang/error/setAmbLightLvl?level=%d
@lang/error/setEventLightLvl?level=%d
@lang/error/StartTimeline
@lang/error/getMeasurementData
@lang/error/loadBasestationConfiguration?id=%d
@lang/error/setBasestationConfiguration?id=%d
@lang/error/unknownNodeMsg

Tabelle 4.6: Event-Log Nachrichten

Typ	Beschreibung
MYSQL_LOG_STATUS	Statusmeldung
MYSQL_LOG_ERROR	Fehlermeldung
MYSQL_LOG_MESSAGE	*message* von einem Timeline-Eintrag
MYSQL_LOG_EVENT	Ereignismeldung
MYSQL_LOG_PACKAGE_TRACE	Meldung für eine Packetverfolgung
MYSQL_LOG_CONTROL	Meldung für eine durchgeführte Aktion der Kontroll- und Messkarte

Tabelle 4.7: Event-Log Typen

Die vom Modul MySQL zur Verfügung gestellten Schnittstellen sind in den Tabellen 4.8, 4.9 und 4.10 aufgelistet.

int mysqlSetStatusNodeEnable(MYSQL *conn, int node)	
Beschreibung	Statusmeldung für aktiven Sensorknoten mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`node`...ID des aktiven Knoten
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlSetStatusNodeDisable(MYSQL *conn, int node)	
Beschreibung	Statusmeldung für inaktiven Sensorknoten mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`node`...ID des inaktiven Knoten
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlSetStatusAmbLightLvl(MYSQL *conn, int level)	
Beschreibung	Statusmeldung für aktuellen Helligkeitswert für das Umgebungslicht mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`level`...Helligkeitswert für Umgebungslicht
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlSetStatusEventLightLvl(MYSQL *conn, int level)	
Beschreibung	Statusmeldung für aktuellen Helligkeitswert für das Ereignislicht mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`level`...Helligkeitswert für Ereignislicht
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlWriteMeasurementData(MYSQL *conn, char *timeStamp, int numNodes, double *buffer)	
Beschreibung	Speicherung der Messdaten in der Datenbank mittels *Prepared-Statements*
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`timeStamp`...Pointer auf Zeitstempel
	`numNodes`...Anzahl der gemessenen Sensorknoten
	`buffer`...Array von Buffern der Messdaten für jeden Sensorknoten
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlWriteEventLogEntry(MYSQL *conn, char *timeStamp, char *msgType, char *msgContent)	
Beschreibung	Speicherung einer Ereignis-Meldung in der Datenbank mittels *Prepared-Statements*
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`timeStamp`...Pointer auf Zeitstempel
	`msgType`...Art des Ereignis. Auflistung der möglichen Typen in Tabelle 4.7.
	`msgContent`...Zeichenkette mit Inhalt der Ereignis-Meldung
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlGetConfiguration(MYSQL *conn, int id, configurationTable *conf)	
Beschreibung	Einlesen einer Konfigurations-Tabelle aus der Datenbank für eine Sensorknotenkonfiguration mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
	`id`...ID der gewünschten Konfiguration
	`conf`...Konfigurationsdaten im Format der configurationTable-Struktur
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag
	`EXIT_FAILURE` bei einem MySQL-Fehler

Tabelle 4.8: Schnittstellen Modul MySQL – Teil 1

int mysqlGetTimeline(MYSQL *conn)	
Beschreibung	Einlesen und lokale Speicherung der Timeline-Daten aus der Datenbank mittels Query
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlGetTimelineEntry(MYSQL *conn, uint time)	
Beschreibung	Suche nach dem übergebenen Zeitstempel in den lokal gespeicherten Timeline-Daten
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct `time`...Ganzzahliger Wert für den Zeitstempel
Rückgabewert	`-2`: Tabelle für Timeline-Daten ist leer `-1`: Ein MySQL-Fehler ist aufgetreten `0`: Kein Eintrag für übergebenen Zeitstempel gefunden `1`: Suche war erfolgreich, Eintrag für Zeitstempel gefunden
int mysqlCheckTimelineEntryType(void)	
Beschreibung	Überprüfen der Art des Timeline-Eintrags (`field`, `message` oder `end`)

Rückgabewert	`0`: Unbekannter Eintrag
	`1`: `field`
	`2`: `message`
	`3`: `end`

int mysqlCheckTimelineEntryField(int *var1, int *var2)	
Beschreibung	Überprüfen des Inhalts von `field` des Timeline-Eintrags
Parameter	`var1`...Variable für Helligkeitswert bei `lightning` (Hintergrundlicht), Adresse des Sensorknoten bei `node` oder Helligkeitswert bei `event` (Ereignislicht) `var2`...Aktion für Sensorknoten bei `node`. `1` für aktivieren, `0` für deaktivieren und `-1` für unbekannten Eintrag.
Rückgabewert	`0`: Unbekannter Eintrag `1`: `lightning` für Hintergrundlichtänderung `2`: `node` für Aktion mit Sensorknoten `3`: `event` für Ereignislichtänderung
char *mysqlCheckTimeLineMessageContent(void)	
Beschreibung	Einlesen des Inhalts von `content` des Timeline-Eintrags
Rückgabewert	Zeichenkette für Inhalt von `content`
void mysqlCloseTimeline(void)	
Beschreibung	Beenden der Timeline und freigeben der lokal gespeicherten Timeline-Daten
int mysqlSetExerciseStart(MYSQL *conn)	
Beschreibung	Speichert Statusmeldung für Beginn der Übung in der Datenbank
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlSetExerciseStop(MYSQL *conn)	
Beschreibung	Speichert Statusmeldung für Ende der Übung in der Datenbank
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler
int mysqlShowAllTables(MYSQL *conn)	
Beschreibung	Auslesen aller Tabellen in der Datenbank. Kann zur Fehlersuche eingesetzt werden.
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler

Tabelle 4.9: Schnittstellen Modul MySQL – Teil 2

int mysqlConnect(MYSQL *conn, char *server, char *user, char *password, char *database)	
Beschreibung	Initiieren der Verbindung zur MySQL-Datenbank
Parameter	`conn`...Variable zum Speichern der aktiven MySQL Verbindung im MYSQL-struct `server`...Zeichenkette für Serveradresse `user`...Zeichenkette für Benutzername `password`...Zeichenkette für Benutzerpasswort `database`...Zeichenkette für Name der gewünschten Datenbank
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler
void mysqlCloseConnection(MYSQL *conn)	
Beschreibung	Beenden der aktiven Verbindung zur MySQL-Datenbank
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
int mysqlHeartbeat(MYSQL *conn)	
Beschreibung	Aktiver Ping zur MySQL-Datenbank um eine Zeitüberschreitung der Verbindung zu verhindern
Parameter	`conn`...Aktive MySQL Verbindung im Format MYSQL-struct
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Datenbankeintrag `EXIT_FAILURE` bei einem MySQL-Fehler

Tabelle 4.10: Schnittstellen Modul MySQL – Teil 3

4.3.1.4 Modul Messung

Für die Steuerung der LEDs zur Beleuchtung, An- und Ausschalten sowie zur Messung der Ströme der permanent versorgten Sensorknoten wird die Messkarte NI-USB-6211 [23] von National Instruments® (NI) eingesetzt. Für die Ansteuerung der Messkarte wurde die NI-DAQmx C API für Linux, welche in der Version 3.5 von NI zur Verfügung gestellt wird, verwendet. Das Modul Messung beinhaltet die gesamte Verwaltung über die API und abstrahiert die benötigten Schnittstellen für das Modul Main.

Für jede auszuführende Aktion mit der Messkarte muss ein eigener sogenannter `TaskHandle` verwaltet werden. So kann der Zugriff und der Status der Ressource lokal gespeichert werden. Aufgrund der Einschränkung für den USB-VISA-Treiber für Linux können keine analogen und digitalen Ressourcen gleichzeitig aktiv (gestartet) sein. Die einzige Möglichkeit besteht nun, dass für jede Aktion der Task der Ressource neu gestartet (`DAQmxBaseStartTask()`) und zum Schluss wieder beendet werden muss (`DAQmxBaseStopTask()`). Wesentlicher Nachteil dieser Methode ist die nicht unerhebliche Zeitverzögerung durch den Neustart eines Tasks von bis zu 600ms. Dementsprechend ergibt sich dadurch ein minimales konstantes Messintervall von 1s. **Hinweis:** Bei hoher Prozessorauslastung und/oder hohem USB-Datentransfer des Host-Systems können zusätzliche Zeitverzögerungen entstehen.

Tritt ein Fehler während eines aktiven Tasks auf, muss dieser durch `DAQmxBaseStopTask()` gestoppt und mit `DAQmxBaseClearTask()` zurückgesetzt werden. Dieser Task muss dann mit `DAQmxBaseCreateTask()` neu initialisiert werden, dasselbe gilt auch wenn der `TaskHandle` für den gewünschten Task noch nicht existieren sollte (erstmaliger Einsatz einer Aktion). Zusätzlich muss noch die dazugehörige Ressource konfiguriert werden. Als Beispiel für diesen Ablauf wird folgend der Task für die Strommessung der Sensorknoten herangezogen:

1. `DAQmxBaseCreateTask()`
2. `DAQmxBaseCreateAIVoltageChan()`
3. `DAQmxBaseCfgSampClkTiming()`

War die Initialisierung und Konfiguration erfolgreich, muss nur mehr der folgende Ablauf für eine Messung durchgeführt werden:

1. `DAQmxBaseStartTask()`
2. `DAQmxBaseReadAnalogF64()`
3. `DAQmxBaseStopTask()`

Die Messkarte NI-USB-6211 besitzt genau die Anzahl der benötigten digitalen sowie analogen Ausgänge [23] [24]. Für die Strommessung wird nur ein Teil der vorhandenen analogen Eingänge genutzt. Eine Übersicht der benutzten Ein- und Ausgänge ist in Tabelle 4.11 ersichtlich. Für jede Ressource müssen nicht nur die Pins zugewiesen werden, sondern auch die Parameter für die Konfiguration festgelegt werden. Vor allem der Aussteuerbereich der Ausgangsspannungen für die LED-Treiber wurden durch Versuche ermittelt, um die volle Auflösung nutzen zu können. Bei der Strommessung wird das *Multi-Sampling* genutzt, dadurch können mehrere Messungen mit einem definierbaren Zeitintervall auf demselben Eingang durchgeführt werden. Am einfachsten zu konfigurieren sind die digitalen Ausgänge, welche nur auf die Zuweisung des Pins sowie die Zeitüberschreitung beschränkt sind. In Tabelle 4.12 sind die Parameter für die Konfiguration aller benutzten Ressourcen ersichtlich.

Ressource	**Pinname**	**E/A**	**D/A**	**Beschreibung**
Messung	AI 0	Eingang	analog	node1
	AI 1	Eingang	analog	node2
	AI 2	Eingang	analog	node3
	AI 3	Eingang	analog	node4
	AI SENSE	Eingang	analog	Benötigt für NRSE-Messung (Non Referenced Single Ended) [24]
Umgebungslicht-steuerung	AO 0	Ausgang	analog	Spannungslevel für LED-Treiber
	AO GND	Ausgang	analog	Analoge Masse
Ereignislicht-steuerung	AO 1	Ausgang	analog	Spannungslevel für LED-Treiber
	AO GND	Ausgang	analog	Analoge Masse
Aktivieren/ Deaktivieren der Sensorknoten	P1.0	Ausgang	digital	node1
	P1.1	Ausgang	digital	node2
	P1.2	Ausgang	digital	node3
	P1.3	Ausgang	digital	node4
	D GND	Ausgang	digital	Digitale Masse

Tabelle 4.11: Pinbelegung der Mess- und Steuerkarte NI USB-6211

Ressource	Parameter	Wert	Beschreibung
Messung	`chan[]`	`ai0:ai3`	Analoge Eingänge AI 0 bis AI 3
	`min`	`0.0`	Minimaler Eingangspannungslevel
	`max`	`5.0`	Maximaler Eingangspannungslevel
	`clockSource[]`	`OnboardClock`	Interne Zeitmessung
	`samplesPerChan`	`10`	10 Messungen pro Kanal
	`sampleRate`	`1000.0`	Kontinuierliche Messungen mit 1kHz
	`numOfChannels`	`4`	Anzahl der zu messenden Kanäle
	`timeout`	`5.0`	
Umgebungs-licht	`chan[]`	`Dev1/ao0`	Analoger Ausgang AO 0
	`min`	`0.5`	Minimaler Ausgangsspannungslevel
	`max`	`4.8`	Maximaler Ausgangsspannungslevel
	`samplesPerChan`	`1`	konstanter Spannungslevel
	`timeout`	`5`	
Ereignis-licht	`chan[]`	`Dev1/ao1`	Analoger Ausgang AO 0
	`min`	`0.0`	Minimaler Ausgangsspannungslevel
	`max`	`5.5`	Maximaler Ausgangsspannungslevel
	`samplesPerChan`	`1`	konstanter Spannungslevel
	`timeout`	`5`	
Steuerung Sensorknoten	`chan[]`	`port1/line0:3`	Digitale Ausgänge von P1.0 bis P1.3
	`timeout`	`5`	

Tabelle 4.12: Konfigurationsparameter für die Mess- und Steuerkarte NI USB-6211

Die abstrahierten Schnittstellen für das Modul Messung sind teils mit den `typedef`-Parametern von `NIDAQmxBase.h` implementiert. Eine Übersicht der zur Verfügung gestellten Schnittstellen ist in Tabelle 4.13 ersichtlich.

4.3.1.5 Modul Obfuscator

Der Obfuscator hat die Aufgabe Nachrichten für die Socket-Kommunikation zu verschlüsseln. Dabei wird aber keine Sicherheitsrelevante Verschlüsselungsmethode (wie RSA oder ECC) angewandt. Es soll lediglich das Fälschen von Socket-Nachrichten erschwert werden. Der Host nutzt den Obfuscator zum Übermitteln von Nachrichten zu den Daemons [22]. Empfangene Nachrichten vom Host müssen vor der Auswertung entschlüsselt werden, Nachrichten von den Daemons zum Host werden unverschlüsselt übermittelt. Für eine erfolgreiche Entschlüsselung muss zuerst mit `of_convert` die Nachricht selbst entschlüsselt werden, danach folgt mit `of_checkMagic` eine Überprüfung der Nachrichtensignatur. Waren beide Funktionen erfolgreich kann die Nachricht weiterverarbeitet werden.

Die implementierte Verschlüsselungsmethode wird in diesem Dokument nicht weiter beschrieben. Die zur Verfügung stehenden Schnittstellen sind in Tabelle 4.14 aufgelistet.

4.3.1.6 Testmodule

Mit dem vorhandenen Testmodul `daemons/tests/testdnicard.c` können alle Socket-Nachrichten vom Host simuliert werden. Das Testmodul generiert dafür einen Socket-Client der sich mit dem Socket-Server (Daemon) verbindet und die ausgewählte Aktion per Socket-Nachricht übermittelt. Empfangene Nachrichten vom Daemon werden direkt auf die

int niUsb6211Shutdown(void)	
Beschreibung	Stoppt alle laufenden Tasks und setzt alle verwendeten `TaskHandle` zurück.
Rückgabewert	`EXIT_SUCCESS`
	`EXIT_FAILURE`
int niUsb6211SetAmbLight(int level)	
Beschreibung	Steuerung der Helligkeit des Hintergrundlichts
Parameter	`level`...Helligkeitswert mit einer Auflösung von 16bit
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Setzen der Ausgangsspannung
	`EXIT_FAILURE` bei einem Fehler von der Mess- und Steuerkarte
int niUsb6211SetEventLight(int level)	
Beschreibung	Steuerung der Helligkeit des Ereignislichts
Parameter	`level`...Helligkeitswert mit einer Auflösung von 16bit
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Setzen der Ausgangsspannung
	`EXIT_FAILURE` bei einem Fehler von der Mess- und Steuerkarte
int niUsb6211StartMetering(double *buffer)	
Beschreibung	Durchführung einer Strommessung aller permanent versorgten Sensorknoten. Es werden, je nach Konfiguration, alle Messwerte pro Sensorknoten gemittelt.
Parameter	`buffer`...Speicherbereich für die Messwerte aller permanent versorgten Sensorknoten mit doppelter Genauigkeit
Rückgabewert	`EXIT_SUCCESS` bei erfolgreicher Messung
	`EXIT_FAILURE` bei einem Fehler von der Mess- und Steuerkarte
int niUsb6211EnableNode(uInt8 node)	
Beschreibung	Aktiviert den gewünschten Sensorknoten
Parameter	`node`...Adresse des Sensorknoten
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Setzen des digitalen Ausgangs
	`EXIT_FAILURE` bei einem Fehler von der Mess- und Steuerkarte
int niUsb6211DisableNode(uInt8 node)	
Beschreibung	Deaktiviert den gewünschten Sensorknoten
Parameter	`node`...Adresse des Sensorknoten
Rückgabewert	`EXIT_SUCCESS` bei erfolgreichem Setzen des digitalen Ausgangs
	`EXIT_FAILURE` bei einem Fehler von der Mess- und Steuerkarte

Tabelle 4.13: Schnittstellen Modul Messung

Konsole umgeleitet. Zum Verschlüsseln der Nachrichten wird die Funktion `of_create()` vom Modul Obusfactor (Kapitel 4.3.1.5) benutzt.

4.3.1.7 Konfiguration

Der Daemon besitzt umfangreiche Konfigurationsmöglichkeiten, um den Anforderungen gerecht zu werden. Diese sind in `daemons/daemonconf.h` als Konstanten und als Startparameter des Daemons aufgeteilt.

Als *Startparameter* wurden jene implementiert, die für eine Anpassung an die Systemumgebung zuständig sind. Die Systemumgebung beinhaltet die Anbindung an die MySQL-Datenbank, die Socket-Kommunikation sowie die Konfiguration der Mess- und Steuerkarte von NI. Die vollständige Liste der Startparameter ist in Tabelle 4.15 dargestellt.

char of_convert(char *message, unsigned int length, of_message *result)	
Beschreibung	Entschlüsseln einer vom Host empfangenen Socket-Nachricht
Parameter	`message`...Verschlüsselte Socket-Nachricht vom Host `length`...Länge der Nachricht `result`...Pointer auf Speicherplatz für entschlüsselte Nachricht im Format of_message (Details in `obfuscate.h`)
Rückgabewert	`1`: OF_OK `-1`: OF_ERROR_LENGTH `-2`: OF_ERROR_HEADER `-3`: OF_ERROR_TYPE `-4`: OF_NULL_POINTER `-5`: OF_ERROR_MALFORMED_LENGTH
char of_checkMagic(of_message *message)	
Beschreibung	Überprüft die Signatur der entschlüsselten Nachricht
Parameter	`message`...Pointer auf die Zeichenkette
Rückgabewert	`1`: Gültige Signatur $\neq$`1`: Fehlerhafte Signatur
of_message* of_create(of_entry type, of_entry *args, unsigned int argc)	
Beschreibung	Verschlüsselt die übergebene Nachricht. Wird nur vom Host benutzt
Parameter	`type`...Aktionstyp der Nachricht, kann Pseudo-Wert beinhalten `args`...Zeichenkette der Nachricht `argc`...Länge der Zeichenkette
Rückgabewert	Pointer der verschlüsselten Nachricht

Tabelle 4.14: Schnittstellen Modul Obfuscator

Parameter	**Typ**	**Beschreibung**
`--measurement-intervall`	Zahlenwert	Intervall für Strommessung der permanent versorgten Sensorknoten. Ganzzahliger Wert in Sekunden, Minimum 1s
`--timeline-step`	Zahlenwert	Kleinster möglicher Zeitsprung bis zu einem neuen Timeline-Ereignis, Ganzzahliger Wert in Sekunden, Minimum 1s.
`--ipc-port`	Zahlenwert	Port des Socket-Servers
`--mysql-server`	Zeichenkette	Adresse des MySQL-Servers
`--mysql-user`	Zeichenkette	Benutzername für MySQL-Datenbank
`--mysql-password`	Zeichenkette	Benutzerpasswort für MySQL-Datenbank
`--mysql-database`	Zeichenkette	Name der MySQL-Datenbank

Tabelle 4.15: Startparameter für Daemon Mess- und Kontrolleinheit

Die Zuweisung des Zahlenwerts/Zeichenkette zu den Parameter erfolgt mit einem =-Zeichen und muss folgendermaßen aussehen:

```
./dnicard --ipc-port=5432 --timeline-step=1 --measurement-intervall=1
    --mysql-server=localhost --mysql-user=root --mysql-password=1234
    --mysql-database=riplecs
```

Hinweis: Für einen erfolgreichen Start des Daemons müssen alle Parameter angegeben werden. Die Reihenfolge kann jedoch beliebig gewählt werden.

In Tabelle 4.16 sind die wichtigsten *konstanten Konfigurationsparameter* in `daemoconf.h` für den Kompilierungszeitpunkt beschrieben:

Parameter	Standardwert	Beschreibung
`NUM_OF_NODES_CURRENT_METERING`	`4`	Anzahl der permanent versorgten Sensorknoten
`CURRENT_METERING_SHUNT_VALUE`	`0.47`	Shunt-Widerstand auf Versorgungsplatine (Kapitel 4.2.2) für Strommessung in Ω
`CURRENT_METERING_SHUNT_GAIN`	`200`	Verstärkungsfaktor für Mess-Operationsverstärker INA210 [42]

Tabelle 4.16: Konstante Parameter in daemonconf.h für Daemon Mess- und Kontrolleinheit

4.3.2 Daemon Basisstation

Der Daemon für die Basisstation läuft unabhängig vom Daemon der Mess- und Kontrolleinheit. Für die Socket-Kommunikation mit dem Web-Server wurde ein eigener Socket-Server implementiert, der einen eigenen Port verwaltet. Die Aufgaben des Daemons sind die Kommunikation mit der Basisstation sowie das Speichern aller Nachrichten von den Sensorknoten in der MySQL-Datenbank. Die Kommunikation mit der Basisstation erfolgt mittels serieller Schnittstelle. In Abbildung 4.19 sind alle implementierten Module dargestellt.

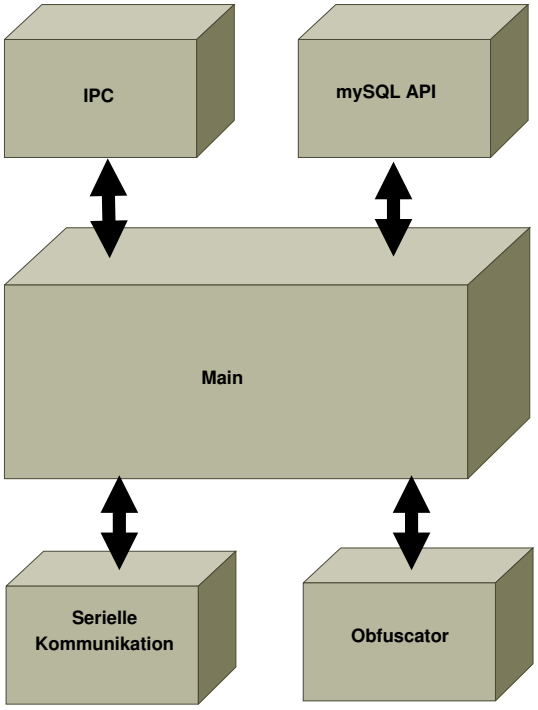


Abbildung 4.19: Blockschaltbild Daemon Basisstation

4.3.2.1 Modul Main

Dieses Modul beinhaltet die Kontrollstruktur der Komponente und kann in zwei Abschnitte unterteilt werden. Folgend werden auch die dazugehörigen Dienste aufgelistet:

- Konfiguration und Konnektivität
  - Auswertung der Startparameter und Konfiguration der Module
  - Etablieren der Verbindungen
- Endlosschleife zum sequenziellen Abarbeiten der Dienste
  - Nachrichten von der Basisstation verwalten und in Datenbank speichern
  - Socket-Kommunikation mit Clients und Aktionen ausführen

Mit dem Flussdiagramm in Abbildung 4.20 sind die Abfolge sowie die Dienste der einzelnen Abschnitte dargestellt. Der Abschnitt der Socket-Kommunikation und Auswertung der Socket-Nachrichten (in Kapitel 4.3.7) wird in Abbildung 4.15 vollständig dargestellt.

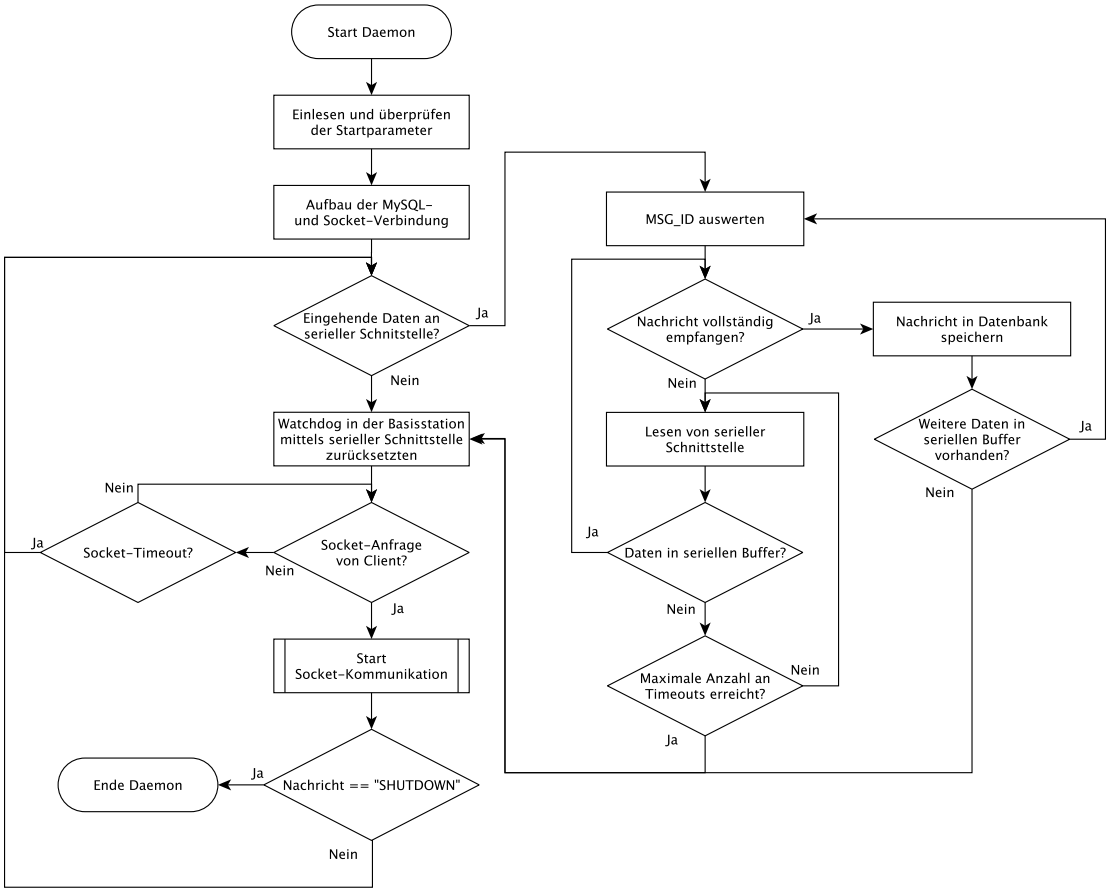


Abbildung 4.20: Flussdiagramm Daemon Basisstation Modul Main

Werden Daten auf der serielle Schnittstelle durch den Aufruf von `serialComRead()` registriert, wird die Funktion `processReadData()` verwendet, um diese Daten auszuwerten. Zuerst wird versucht die `MSG_ID` einer vorhandenen Nachrichtenstruktur (Auflistung aller Nachrichten in Kapitel 4.3.5) zuzuweisen. Nun kann die Länge der erwarteten Nachricht mit der Länge der seriellen Buffers verglichen werden. Für den Fall, dass zu wenige Daten im Buffer liegen, wird mehrfach mit `serialComRead()` überprüft, ob weitere Daten empfangen wurden. Ist eine Nachricht vollständig empfangen worden, wird diese umgehend in der Datenbank gespeichert. Als nächster Schritt wird geprüft, ob weitere Daten, abzüglich der Daten der empfangenen Nachricht, im seriellen Buffer vorhanden sind. In diesem Fall wird `processReadData()` rekursiv aufgerufen bis alle Daten im seriellen Buffer abgearbeitet wurden. Das Vorhandensein mehrerer Nachrichten im seriellen Buffer ergibt sich durch zyklisches Überprüfen der seriellen Schnittstelle auf Daten. In diesem Zusammenhang kann die serielle Schnittstelle nicht Interrupt-Basierend verwendet werden.

Der Watchdog der Basisstation muss regelmäßig mit `serialComSendHeartbeat()` zurückgesetzt werden. Tritt ein Problem mit der seriellen Schnittstelle auf, erhält die Basisstation dadurch keine Nachricht mehr zum Rücksetzen des Watchdogs. Die Basisstation wird beim Auslösen des Watchdogs die serielle Schnittstelle neu initialisieren, um wieder mit dem Daemon kommunizieren zu können.

In Tabelle 4.17 sind die implementierten Funktionen für das Modul Main aufgelistet.

4.3.2.2 Modul Inter Process Communication (IPC)

Dieses Modul entspricht dem *Modul IPC* vom Daemon Mess- und Kontrolleinheit und ist in Kapitel 4.3.1.2 ausführlich beschrieben.

4.3.2.3 Modul MySQL

Dieses Modul entspricht dem *Modul MySQL* vom Daemon Mess- und Kontrolleinheit und ist in Kapitel 4.3.1.3 ausführlich beschrieben.

4.3.2.4 Modul Obfuscater

Dieses Modul entspricht dem *Modul Obfuscator* vom Daemon Mess- und Kontrolleinheit und ist in Kapitel 4.3.1.5 ausführlich beschrieben.

4.3.2.5 Modul Serielle Kommunikation

Die Kommunikation mit der Basisstation wird durch die serielle Schnittstelle zwischen Host und Basisstation ermöglicht. Die Erweiterungsplatine der Basisstation stellt dafür einen UART-zu-USB-Konverter zur Verfügung. Diese emulierte UART-Schnittstelle kann vom Host als RS232 konfiguriert, gelesen und beschrieben werden.

Für die Konfiguration der Sensorknoten wurde `serialComSetConfiguration()` implementiert. Diese Funktion bereitet die Konfigurationsdaten von der Datenbank für die jeweiligen Konfigurationsnachrichten der Sensorknoten (in Kapitel 4.3.5.2 Konfigurationspakete aufgelistet) vor, um anschließend zur Basisstation übermittelt zu werden. Nach jedem übermittelten Packet wird 100ms gewartet bevor das nächste Packet aufgebaut und versandt wird. Damit sollen eventuelle Kollisionen der Nachrichtenpakete im Netzwerk der

int checkArgvList(int argc, char *argv[], char *param, void **value, uint number)	
Beschreibung	Überprüft die Kommandozeilen-Parameter (argv) mit den festgelegten Parametern für den Daemon in `daemonconf.h`. Die spezifischen Parameter sind in Kapitel 4.3.2.7 Konfiguration beschrieben.
Parameter	`argc`...von der Funktion `main()` übergebener Parameter für die Anzahl der Kommandozeilen-Parameter `argv`...von der Funktion `main()` übergebener Zeichenketten-Array mit den Kommandozeilen-Parametern `*param`...Zu überprüfende Zeichenkette `**value`...Speicherort für Zahlenwert oder Zeichenkette entsprechend des gesuchten Parameters `number`...`0` für Zeichenkette, `1` für Zahlenwert
Rückgabewert	`-1`: Fehler bei Konvertierung für Zahlenwert `0`: Gesuchter Parameter wurde nicht in Parameter-Liste (argv) gefunden `1`: Auswertung erfolgreich
int getTimeStamp(char *timeStampVal)	
Beschreibung	Konvertierung der Systemzeit in definiertes Format zur Speicherung in Datenbank
Parameter	`timeStampVal`...Pointer auf Zeichenkette für das konvertierte Zeitformat
Rückgabewert	`EXIT_FAILURE`: Fehler bei Akquisition der Systemzeit oder bei Formatkonvertierung `EXIT_SUCCESS`: Konvertierung erfolgreich
void writeMysqlEventLog(char *content, char *type)	
Beschreibung	Übermittelt die Information eines Ereignis inklusive der aktuellen Systemzeit an die Datenbank
Parameter	`content`...Zeichenkette für das Ereignis im sogenannten *localization-format* (Tabelle 4.6) `type`...Art des Ereignis (Tabelle 4.7)
void getConfigurationAndExecute(int configID)	
Beschreibung	Konfigurationsdaten von Datenbank holen und an die Basisstation weiterleiten
Parameter	`configID`...ID der gewünschten Konfiguration
void processReadData(char *buffer, int bufLen, int numOfBytes)	
Beschreibung	Auswerten der empfangenen Nachrichten im seriellen Buffers. Vollständig empfangene Nachrichten werden in Datenbank gespeichert. Bei unvollständig übertragenen Nachrichten wird auf fehlende Daten eine definierte Zeit gewartet.
Parameter	`buffer`...Pointer auf seriellen Buffer `bufLen`...Maximale Länge des seriellen Buffers in Bytes `numOfBytes`...Anzahl der empfangenen Bytes im seriellen Buffer

Tabelle 4.17: Schnittstellen Modul Main für Daemon Basisstation

Sensorknoten vermieden werden. Die Reihenfolge der Nachrichtenpakete wurde wie folgt implementiert:

1. `radioConfigMsg`
2. `eventLightConfigMsg`
3. `debugMsg`
4. `switchesConfigMsg`

5. `meteringConfigMsg`

6. `helloMsg`

Bis auf die Nachbarschaftstabellen (in Kapitel 4.1.4 ersichtlich) werden alle Konfigurationsdaten von der Datenbank zur Verfügung gestellt. Diese Tabellen werden in sogenannte Konfigurationspakete zusammengefasst und lokal als binäre Dateien gespeichert. Für das Konfigurationspaket `radioConfigMsg` wird die entsprechende Datei geöffnet und der Inhalt kopiert.

Der direkte Schreib- und Lesezugriff mit `write()` und `read()` wird nur geringfügig bezüglich dem *FileDescriptor* abstrahiert und kann durch `serialComWrite()` beziehungsweise `serialComRead()` bedient werden.

Für dieses Modul sind die implementierten Schnittstellen in Tabelle 4.18 aufgelistet.

4.3.2.6 Testmodule

Das Testmodul `daemons/tests/testserial.c` kann mittels Socket-Kommunikation mit dem Daemon kommunizieren, um einen Komponententest durchzuführen. Es können somit alle Socket-Nachrichten vom Host simuliert werden. Das Testmodul generiert dafür einen Socket-Client der sich mit dem Socket-Server (Daemon) verbindet und die ausgewählte Aktion per Socket-Nachricht übermittelt. Empfangene Nachrichten vom Daemon werden direkt auf die Konsole umgeleitet. Zum Verschlüsseln der Nachrichten wird die Funktion `of_create()` vom Modul Obusfactor (Kapitel 4.3.1.5) benutzt.

Als weitere Möglichkeit kann noch direkt mit der Basisstation über die serielle Schnittstelle kommuniziert werden, damit kann explizit die Übertragung der Nachrichtenpakete getestet werden.

4.3.2.7 Konfiguration

Wie beim Daemon Mess- und Kontrolleinheit wird auch hier zwischen konstanten Konfigurationsparameter in `daemons/daemonconf.h` und den Startparametern unterschieden. Zusätzlich werden noch notwendige Konfigurationsdaten für die Sensorknoten in Form von Dateien verwaltet.

Die *Startparameter* bestehen aus leicht änderbaren Informationen für die Konfiguration der Schnittstellen (Socket-Kommunikation, MySQL-Datenbank-Anbindung und serielle Schnittstelle). Die notwendigen Parameter sind in der Tabelle 4.19 zusammengefasst. Die Zuweisung des Zahlenwerts/Zeichenkette zu den Parameter erfolgt mit einem =-Zeichen und muss folgendermaßen aussehen:

```
./dbasestation --serialcom-port=/dev/ttyUSB1 --ipc-port=5433 --mysql-server=
    localhost --mysql-user=root --mysql-password=1234 --mysql-database=riplecs
```

Hinweis: Für einen erfolgreichen Start des Daemons müssen alle Parameter angegeben werden. Die Reihenfolge kann jedoch beliebig gewählt werden.

In Tabelle 4.20 sind die wichtigsten *konstanten Konfigurationsparameter* in `daemoconf.h` für den Kompilierungszeitpunkt beschrieben: Diese *zusätzlichen Konfigurationsdaten* sind als binäre Dateien lokal auf dem Host gespeichert. Der Inhalt dieser Dateien besteht aus

int serialComOpen(char *port)	
Beschreibung	Das übergebene Port wird geöffnet und konfiguriert
Parameter	`port`...Zeichenkette für die gewünschte Schnittstelle, zum Beispiel `/dev/ttyUSB1` bei einem Linux-Host.
Rückgabewert	`-1`: Fehler beim Konfigurieren der Schnittstelle `0`: Fehler beim Öffnen der Schnittstelle `>0`: Erfolgreich geöffnet und konfiguriert
void serialComClose(void)	
Beschreibung	Beenden der Verbindung mit der seriellen Schnittstelle
int serialComWrite(char *msg, int numOfBytes)	
Beschreibung	Direktes Schreiben auf serielle Schnittstelle
Parameter	`msg`...Pointer auf Nachricht `numOfBytes`...Länge der Nachricht
Rückgabewert	`<0`: Fehler beim Schreiben der Daten $\geq$`0`: Anzahl der übertragenen Bytes
int serialComRead(char *msg, int numOfBytes)	
Beschreibung	Direktes lesen von der seriellen Schnittstelle
Parameter	`msg`...Pointer auf Nachrichtenbuffer `numOfBytes`...Maximale Länge des Buffers
Rückgabewert	`<0`: Fehler beim Lesen der Daten $\geq$`0`: Anzahl der empfangenen Bytes
int serialComSetConfiguration(configurationTable *config)	
Beschreibung	Vorbereiten der Konfigurationsdaten und übermitteln der fertigen Konfigurationspakete
Parameter	`config`...Konfigurationsdaten von der Datenbank im Format `configurationTable`

Rückgabewert	`EXIT_SUCCESS`: Erfolgreiche Konfigurationssequenz
	`1`: Fehler bei `MSG_ID_RESET_BASESTATION`
	`2`: Fehler bei `MSG_ID_CONFIG_RADIO`
	`3`: Fehler bei `MSG_ID_CONFIG_EVENT_LIGHT`
	`4`: Fehler bei `MSG_ID_DEBUG_CONFIG`
	`5`: Fehler bei `MSG_ID_CONFIG_SWITCHES`
	`6`: Fehler bei `MSG_ID_CONFIG_METERING`
	`7`: Fehler bei `MSG_ID_HELLO_PACKAGE`

int serialComSendHeartbeat(void)	
Beschreibung	Übermitteln des Nachrichtenpackets `MSG_ID_HEARTBEAT` zum Rücksetzen des Watchdogs der Basisstation
Rückgabewert	`<0`: Fehler beim Schreiben der Daten $\geq$`0`: Anzahl der übermittelten Bytes

Tabelle 4.18: Schnittstellen Modul Serielle Kommunikation

Topologie-Informationen für das Netzwerk der Sensorknoten. Im Speziellen sind dort die Nachbarschaftstabellen in den sogenannten Nachbarschaftspaketen untergebracht (Kapitel 4.1.4). Der Pfad und die Dateierweiterung werden als Startparameter übergeben, der Dateiname selbst besteht aus Zahlen. So könnte zum Beispiel in der Datenbank unter dem Eintrag `Topology` die Zahl `3` eingetragen sein, damit würde das Modul *Serielle Kommunikation* nach der Datei `daemons/topologies/3.topology` suchen und deren Inhalt in die Struktur der Konfigurationsnachricht kopieren.

Parameter	Typ	Beschreibung
`--serialcom-port`	Zeichenkette	Name der gewünschten seriellen Schnittstelle
`--ipc-port`	Zahlenwert	Port des Socket-Servers
`--mysql-server`	Zeichenkette	Adresse des MySQL-Servers
`--mysql-user`	Zeichenkette	Benutzername für MySQL-Datenbank
`--mysql-password`	Zeichenkette	Benutzerpasswort für MySQL-Datenbank
`--mysql-database`	Zeichenkette	Name der MySQL-Datenbank

Tabelle 4.19: Startparameter für Daemon Basisstation

Parameter	Standardwert	Beschreibung
`MYSQL_HEARTBEAT`	`10000`	Zeitintervall in Sekunden für einen Ping an die Datenbank um ein *Timeout* zu vermeiden
`SERIALCOM_HEARTBEAT`	`10`	Anzahl der Zyklen in der Endlosschleife im Modul Main nach diesen der Watchdog der Basisstation zurückgesetzt werden muss
`CONFIG_TOPOLOGY_FILE_FOLDER`	`topologies/`	Relativer Pfad für die zusätzlichen Konfigurationsdateien der Sensorknoten
`CONFIG_TOPOLOGY_FILE_EXT`	`.topology`	Dateiendung für die zusätzlichen Konfigurationsdateien der Sensorknoten

Tabelle 4.20: Konstante Parameter in daemonconf.h für Daemon Basisstation

4.3.3 Drahtlose Sensorknoten

Folgende Funktionen wurden aufgrund der Anforderungen aus Kapitel 3.3.2 für die Sensorknoten implementiert:

- Sensoren auswerten und Spannungsmessungen durchführen (nur EHD-Sensorknoten)
- Informationen/Konfigurationen empfangen, auswerten und Messdaten übermitteln
- Verhalten von Netzwerk-Topologien und Routing-Protokollen umsetzen

Die gesamte Applikation ist ereignisbasierend in TinyOS [27] implementiert. In Abbildung 4.21 sind die verwendeten Module der Sensorknoten ersichtlich.

4.3.3.1 Modul Radio Kommunikation

Das Modul entspricht der Standardkomponente `GenericComm` von TinyOS. Die Auswahl wurde aufgrund der einfachen Schnittstelle (`SendMsg` und `ReceiveMsg`) und der Stabilität der Standardkomponente getroffen. Es kommen jedoch nur Teile der gesamten Funktionalität der Komponente zum Einsatz. So wurde nur die Radio-Kommunikation mittels Broadcast-Nachrichten verwendet. Dadurch wird die Filterung der Nachrichten an die nächste Schicht, in diesem Fall dem Modul Routing, übertragen. Dementsprechend muss die TinyOS-Komponente nicht modifiziert oder erweitert werden. Der Vorteil liegt in der Austauschbarkeit der Kommunikations-Standardkomponenten von TinyOS, vorausgesetzt wird nur dieselbe Schnittstelle. Auch ein Upgrade auf eine höhere TinyOS-Version sollte somit vereinfacht werden. In Tabelle 4.21 sind die implementierten Schnittstellen aufgelistet.

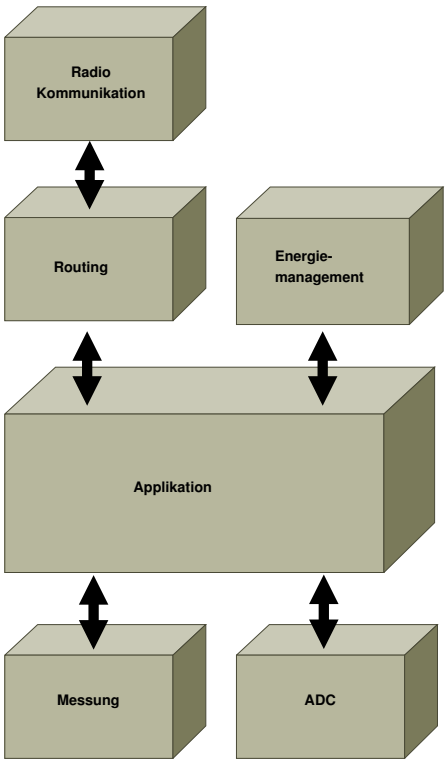


Abbildung 4.21: Blockschaltbild Sensorknotenmodule

command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg)	
Beschreibung	Standardschnittstelle zum Versenden von TinyOS v1.1 `AM`-Nachrichten
Parameter	`address`...Adresse des Empfängerknoten `length`...Länge der Nachricht `msg`...Pointer auf Buffer mit `TOS_Msg`-Struktur
Rückgabewert	Status der Übertragung durch `FAIL` oder `SUCCESS`
event result_t sendDone(TOS_MsgPtr msg, result_t success)	
Beschreibung	Benachrichtigung wenn Nachrichtenübertragung abgeschlossen ist
Parameter	`msg`...Pointer auf Buffer mit `TOS_Msg` Struktur `success`...`FAIL` oder `SUCCESS`
event TOS_MsgPtr receive(TOS_MsgPtr m)	
Beschreibung	Benachrichtigung wenn ein Nachrichtenpaket empfangen wurde
Parameter	`m`...Pointer auf Buffer mit `TOS_Msg` Struktur

Tabelle 4.21: Schnittstellen Modul Radio Kommunikation

4.3.3.2 Modul Routing

Das Routing-Modul wird als Schicht zwischen der Applikation und dem Kommunikations-Modul eingesetzt. Für die Sensorknoten und die Basisstation wurde dasselbe Modul implementiert. Damit wird bei Code-Änderungen die Konsistenz bei allen Netzwerkteil-

nehmern garantiert. Nachteil dieser Implementierung sind die unbenutzten spezifischen Funktionen. Die dedizierten Funktionen für die Basisstation werden in Kapitel 4.3.4.3 Modul Routing genauer beschrieben. Folgende Funktionen wurden für das Routing-Modul implementiert:

- **Filterung der Nachrichten** zur Unterscheidung, ob diese nur an den nächsten Knoten weitergeleitet oder an die Applikation übergeben wird.
- **Umsetzung der Netzwerk-Topologien** um festlegen zu können, mit welchen Sensorknoten im Netzwerk kommuniziert werden kann.
- **Umsetzung der Routing-Protokolle** um die Art der Bewegung der Nachrichten durch das Netzwerk zu bestimmen.
- **Verfolgung der Nachrichten durch das Netzwerk** ermöglicht den Benutzer den vollständigen Pfad der Nachricht zu ermitteln. Wird im Routing-Modul der Basisstation verwendet.

Das Routing und die Weiterleitung von Nachrichten funktioniert nur mit zusätzlichen Information, die für jedes Nachrichtenpaket generiert werden müssen. Diese Routing-Informationen (erläutert in Kapitel 4.3.5 Radiopakete) sind nur für das Modul Routing ersichtlich und werden nur dort ausgewertet und modifiziert. Das Modul besitzt im Wesentlichen drei Pfade, die für eine effiziente und nachvollziehbare Kommunikation implementiert wurden. Die Kommunikationspfade sind folgend aufgelistet:

- **Down-Stream** für Nachrichten von den Sensorknoten zur Basisstation
- **Up-Stream** für Nachrichten von der Basisstation zu den Sensorknoten
- **Raw-Stream** für direkte Kommunikation mit einzelnen Sensorknoten

Der *Down-Stream* wird verwendet um Informationen von den Sensorknoten zur Basisstation zu übermitteln. Für diese Funktionalität stellt das Routing-Modul die Schnittstelle `RadioSendMsg.send( uint16_t address, uint8_t length, TOS_MsgPtr msg )` zur Verfügung. Als Ziel muss die Adresse der Basisstation `BASE_STATION_ADDRESS` übergeben werden. Die implementierte Schnittstelle entspricht der Standardschnittstelle `SendMsg` von TinyOS. Zusätzlich wird noch `RadioSendMsg.sendDone( TOS_MsgPtr msg, result_t success )` als Funktionalität von dieser Schnittstelle zur Verfügung gestellt. Um auch den Empfang von Nachrichten zu standardisieren, wird ebenfalls die Standardschnittstelle `ReceiveMsg` von TinyOS verwendet. Der Einsatz von TinyOS-Standardschnittstellen hat den Vorteil der leichteren Austauschbarkeit der Module. In einem Testszenario könnte die Applikation zum Beispiel direkt mit dem TinyOS Kommunikations-Modul kommunizieren. Um den *Down-Stream* sinnvoll zu beschreiben, ist in Abbildung 4.22 ein Sequenzdiagramm des Pfades dargestellt. Für eine übersichtlichere Darstellung ist nur ein Knoten zur Weiterleitung der Nachricht vorhanden. Die Applikation des Sensorknoten, die eine Nachricht übermitteln möchte ruft nun die Funktion `RadioSendMsg.send()` auf. Die Funktion `SignalUpperLayer()` in Abbildung 4.22, 4.23 sowie 4.24 wird als Synonym für die Funktionalität `signal` in TinyOS verwendet. Damit wird der Applikation mittels `RadioSendDoneMsg()` mitgeteilt, dass die Übertragungssequenz abgeschlossen

ist. Als nächsten Schritt empfängt ein Nachbarknoten die Nachricht durch die Routine `RecieveMsgDownStream()` und leitet diese an seinen nächsten Nachbarn weiter, in diesem Fall an die Basisstation. Die Interpretation für Nachbarn hängt von der jeweiligen Netzwerk-Topologie und vom verwendeten Routing-Protokoll ab und ist in Kapitel 4.1.3 genauer erklärt. Bei der Weiterleitung von Nachrichten wird die Applikation nicht über das empfangene Paket informiert, die Verarbeitung wird nur im Routing-Modul behandelt. Im selben Schritt wird auch die Funktionalität `receivedPackageTrace()` ausgeführt, um die sogenannte Nachrichtenverfolgung (genaue Beschreibung in Kapitel 4.3.4.3 Modul Routing) dem Benutzer zur Verfügung zu stellen. Wird die Nachricht im letzten Schritt von der Basisstation empfangen, wird die Applikation in der Basisstation (Kapitel 4.3.4) dahingehend informiert, dass ein Paket für die Übermittlung an den Host bereit steht. Der *Up-Stream* hat

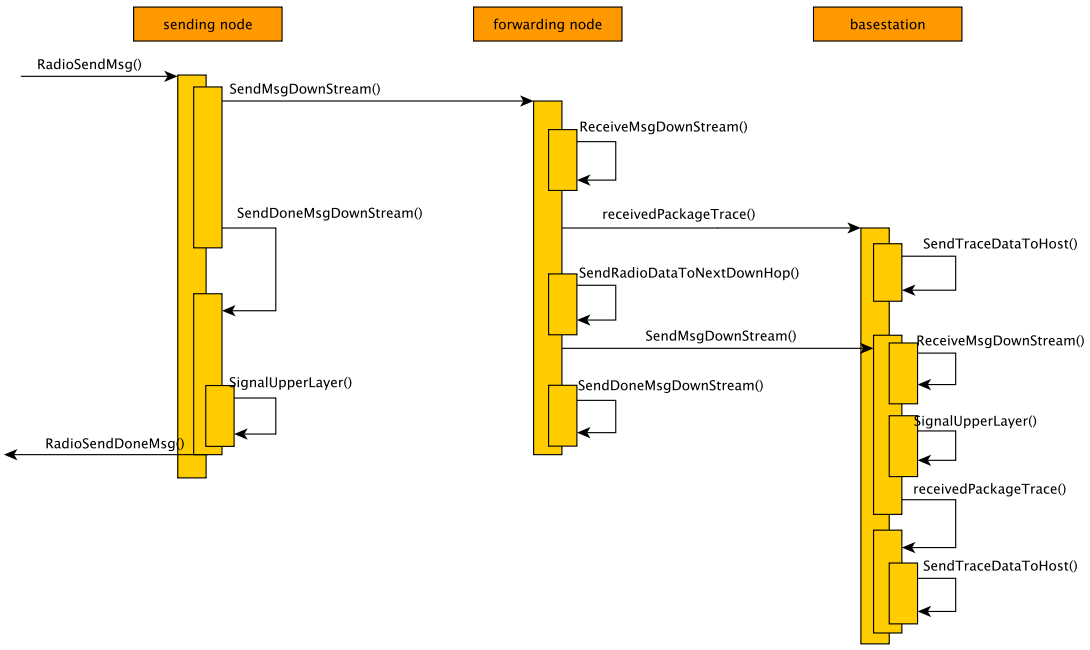


Abbildung 4.22: Sequenzdiagramm Down-Stream. Der Down-Stream wird verwendet um Informationen von den Sensorknoten zur Basisstation zu übermitteln. Dadurch können die generierten Messdaten der Sensorknoten über die Basisstation zum Host weitergeleitet werden.

die Aufgabe, Nachrichten von der Basisstation zu den Sensorknoten zu übermitteln. Dieser Pfad wird über dieselben Standardschnittstellen `SendMsg` und `ReceiveMsg` von TinyOS bedient. Die notwendige Unterscheidung, welcher Pfad genutzt werden soll, geschieht über den Parameter `address` in `RadioSendMsg.send( uint16_t address, uint8_t length, TOS_MsgPtr msg )`. Damit stehen zwei Möglichkeiten zur Übermittlung von Nachrichten für den *Up-Stream* zur Auswahl:

`TOS_BCAST_ADDR` Damit wird eine Broadcast-Nachricht generiert, welche zur Konfiguration der Sensorknoten dient. Die sogenannte benutzer-initiierte Konfiguration ist in

Kapitel 4.1.6 beschrieben. Allen aktiven Sensorknoten im Netzwerk wird mittels der Routine `RecieveMsgUpStream()` der Erhalt der Nachricht signalisiert und entsprechend an die Applikation weitergeleitet. Bei dieser Art der Kommunikation steht keine Nachrichtenverfolgung zur Verfügung.

`HELLO_PACKAGE_ADDRESS` Mit diesem Parameterwert wird eine sogenannte Hallo-Sequenz von der Basisstation initiiert. Diese Sequenz dient zur Bildung von Kommunikationspfaden im Netzwerk, welche im Kapitel 4.1.5 beschrieben ist, und berücksichtigt die jeweilige Netzwerk-Topologie sowie das verwendete Routing-Protokoll. Um den *Up-Stream* zu beschreiben ist in Abbildung 4.23 ein Sequenzdiagramm des Pfades dargestellt. Für eine übersichtlichere Darstellung ist nur ein Knoten zur Weiterleitung der Nachricht vorhanden. Die Applikation der Basisstation ruft nun die Funktion `RadioSendMsg.send()` auf und übergibt dieser das Nachrichtenpaket. Die Routine `RadioSendDoneMsg()` informiert dann die Applikation über die abgeschlossene Übertragungssequenz. Als nächsten Schritt empfängt der für die Weiterleitung verantwortliche Nachbarknoten die Nachricht durch die Routine `RecieveMsgUpStream()` und leitet diese mittels `SendMsgUpStream()` an den Empfängerknoten weiter. Wie auch beim *Down-Stream* wird bei der Weiterleitung von Nachrichten die Applikation nicht über das empfangene Paket informiert. Die Nachrichtenverfolgung steht auch dem *Up-Stream* zur Verfügung. Die Basisstation wird über die Routine `receivedPackageTrace()` über eine erfolgreiche Übermittlung der Nachricht informiert und gibt diese an den Host weiter. Die Applikation des Empfängerknotens wird beim erfolgreichen Empfang der Nachricht informiert, gleichzeitig wird die Basisstation hinsichtlich der weiteren Nachrichtenverfolgung nochmals durch das Routing-Modul informiert.

Beim *Raw-Stream* handelt es sich um eine spezielle Form der implementierten Kommunikation. Es ermöglicht eine direkte Kommunikation zwischen einzelnen Sensorknoten sowie der Basisstation. Die Verbindung untereinander ist unidirektional. Die Nachrichtenverfolgung ist in diesem Pfad nicht verfügbar. Für diese Funktionalität wurde eine möglichst einfache und reduzierte Schnittstelle implementiert. Zum Übermitteln von Nachrichten wird die Funktion `sendRawMsg( uint8_t *msg, uint8_t length, uint8_t nodeID )` und `sendDoneRawMsg()` benutzt. Für den Empfang steht `receivedRawMsg( uint8_t *msg, uint8_t length )` als Schnittstelle zur Verfügung. Der *Raw-Stream* wird in diesem Projekt für folgende Aufgaben verwendet:

Fehlerberichte: Diese Nachrichten dienen zur Fehlererkennung sowie Fehlerdiagnose und werden direkt zur Basisstation übermittelt. Der Sensorknoten mit dem generierten Fehler (Definitionen in Kapitel 4.3.6 ersichtlich) startet die Übertragungssequenz mittels `sendRawMsg()` und erhält die Bestätigung nach einer erfolgreichen Übermittlung durch `sendDoneRawMsg()`. Die Applikation der Basisstation wird durch die Routine `receivedRawMsg()` über ein empfangenes Nachrichtenpaket informiert. Der Fehlerbericht wird dann von der Applikation der Basisstation mittels serieller Schnittstelle (Kapitel 4.3.4.1 Modul Serielle Kommunikation) an den Host weitergereicht.

Konfiguration: Im Vergleich zur benutzer-initiierte Konfiguration im *Up-Stream* handelt es sich bei dieser Art der Konfiguration um die knoten-initiierte Konfiguration und ist in Kapitel 4.1.6 beschrieben. Abbildung 4.24 zeigt das Sequenzdia-

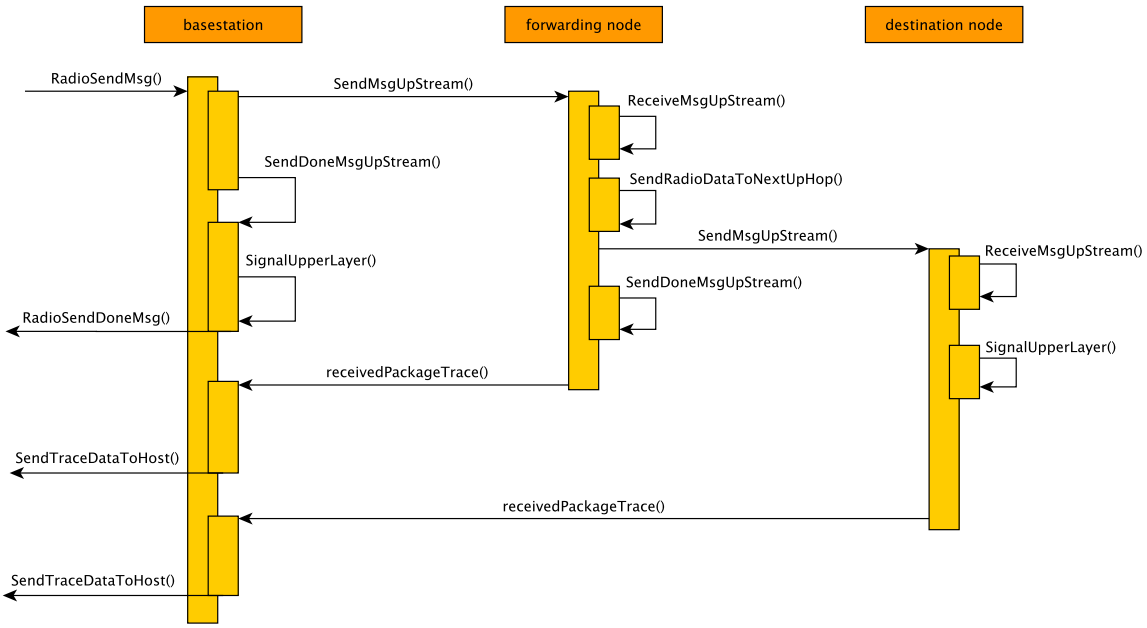


Abbildung 4.23: Sequenzdiagramm Up-Stream. Der Up-Stream wird verwendet um Informationen von der Basisstation zu den Sensorknoten zu übermitteln. Damit können Konfigurationspakete für die Sensorknoten welche vom Host über die Basisstation weitergeleitet werden.

gramm für eine solche Konfiguration. Der zu konfigurierende Sensorknoten startet eine Anfrage an die Basisstation mit der Funktion `sendRawMsg()`. Mit der Routine `sendDoneRawMsg()` ist die erfolgreiche Übermittlung abgeschlossen und die Applikation im Sensorknoten wird damit informiert. Die Basisstation erhält mit der Routine `ReceivedMsgBaseStation()` die Konfigurationsanfrage und leitet diese an die Applikation weiter. Die Basisstation-Applikation übermittelt unmittelbar darauf das Konfigurationspaket mittels `sendRawMsg()` an den Sensorknoten. Der erfolgreiche Empfang des Paketes am Sensorknoten durch `ReceivedMsgBaseStation()` löst die Konfigurationssequenz `configureNode()` (Beschreibung folgt in Kapitel 4.3.3.5) in der Applikation aus.

Die Schnittstellen für das Modul sind unterteilt in die Bereiche Kommunikation (Tabelle 4.23) und Einstellungen (Tabelle 4.22).

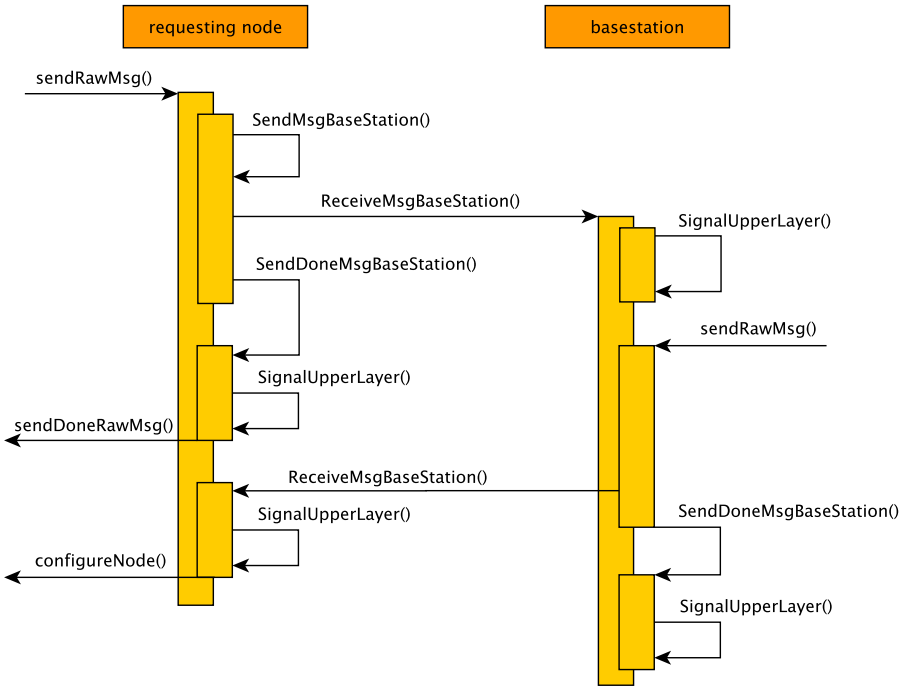


Abbildung 4.24: Sequenzdiagramm Raw-Stream. Beim Raw-Stream handelt es sich um eine spezielle Form der implementierten Kommunikation. Es ermöglicht eine direkte Kommunikation zwischen einzelnen Sensorknoten sowie der Basisstation.

command void setRoutingNeighbours(uint8_t value)	
Beschreibung	Übergibt dem Routing-Modul die Nachbarschaftstabelle
Parameter	`value`...Nachbarschaftstabelle
command void setRoutingType(uint8_t type)	
Beschreibung	Bestimmt das Routing-Protokoll für das Netzwerk
Parameter	`type`...`SHORTEST_PATH`, `DIRECT_TO_BASESTATION` oder `FLOODING`
command void setRoutingOptions(uint8_t value)	
Beschreibung	Übergibt, falls benötigt, zusätzlichen Informationen für das gewählte Routing-Protokoll
Parameter	`value`...Definiert das Zeitintervall für die Generierung eines Hallo-Pakets. Wird nur für `SHORTEST_PATH` benötigt. Der Zeitintervall kann von 1s bis 255s gewählt werden, 0 definiert eine einmalige Generierung.
command void setDebugDelay(uint8_t msec)	
Beschreibung	Zeitverzögerung zwischen Zeitpunkt vom Empfang eines Paketes bis zur Weiterleitung. Hinweis: Busy-Waiting durch Verwendung von `TOSH_uwait`
Parameter	`msec`...Zeitverzögerung in Millisekunden
command void setPackageTrace(bool enabled)	
Beschreibung	Aktivierung/Deaktivierung der Nachrichtenverfolgung
Parameter	`enable`...`TRUE` für Aktivierung, `False` für Deaktivierung

Tabelle 4.22: Schnittstellen für Einstellungen im Modul Routing für Sensorknoten

command result_t RadioSendMsg.send(uint16_t address, uint8_t length, TOS_MsgPtr msg)	
Beschreibung	Zum Versenden von Nachrichten per Up- oder Down-Stream, abhängig vom Parameter `address`

Parameter	`address`...`BASE_STATION_ADDRESS`, `TOS_BCAST_ADDR` oder `HELLO_PACKAGE_ADDRESS`
	`length`...Länge der Nachricht in Bytes, maximale Länge: `TOSH_DATA_LENGTH - ROUTING_PAYLOAD`
	`msg`...Pointer auf eine `TOS_Msg Struktur`

Rückgabewert	Status der Übertragung durch `FAIL` oder `SUCCESS`
event result_t RadioSendMsg.sendDone(TOS_MsgPtr msg, result_t success)	
Beschreibung	Benachrichtigung wenn Nachrichtenübertragung abgeschlossen ist
Parameter	`msg`...Pointer auf eine `TOS_Msg` Struktur `success`...`FAIL` oder `SUCCESS`
event TOS_MsgPtr RadioReceiveMsg.receive(TOS_MsgPtr msg)	
Beschreibung	Benachrichtigung wenn ein Nachrichtenpaket empfangen wurde
Parameter	`msg`...Pointer auf eine `TOS_Msg` Struktur
command result_t sendRawMsg(uint8_t *msg, uint8_t length, uint8_t nodeID)	
Beschreibung	Zum Versenden von Fehlerberichten oder knoten-initiierte Konfigurationsanfragen
Parameter	`msg`...Pointer auf eine Byte-Array `length`...Länge der Nachricht in Bytes, maximale Länge: `TOSH_DATA_LENGTH` `nodeID`...Adresse des Empfängerknotens
Rückgabewert	Status der Übertragung durch `FAIL` oder `SUCCESS`
event void sendDoneRawMsg()	
Beschreibung	Benachrichtigung wenn eine Raw-Nachrichtenübertragung abgeschlossen ist
event void receivedRawMsg(uint8_t *msg, uint8_t length)	
Beschreibung	Benachrichtigung wenn ein Raw-Nachrichtenpaket empfangen wurde
Parameter	`msg`...Pointer auf eine Byte-Array `length`...Länge der Nachricht in Bytes
event void receivedPackageTrace(uint16_t sourceAddr, uint16_t fromAddr, uint16_t toAddr, uint8_t packageID)	
Beschreibung	Benachrichtigung wenn ein Paket für die Nachrichtenverfolgung empfangen wurde. Hinweis: Wird nur für die Basisstation verwendet, wird zur Laufzeit über die ID des Knoten überprüft.
Parameter	`sourceAddr`...Adresse des Knoten welcher der Ursprung der Nachricht ist `fromAddr`...Adresse des Knoten welcher zuletzt die Nachricht weitergeleitet hat `toAddr`...Adresse des Empfängerknotens `packageID`...Eindeutige ID des Paketes

Tabelle 4.23: Schnittstellen für Kommunikation im Modul Routing für Sensorknoten

4.3.3.3 Modul ADC

Dieses Modul wird für die Messwertgenerierung der gewünschten Spannungsniveaus eingesetzt. Der Helligkeitssensor selbst wird nicht über das ADC-Modul bedient. Crossbow® liefert für diesen Sensortyp eine Eigenentwicklung, das Modul `TaosPhoto` wird somit über das SDK zur Verfügung gestellt.

Folgende Messwerte werden durch das ADC-Modul generiert:

- Versorgungsspannung V_{CC}
- Ausgangspannung der Solarzelle
- Referenzspannung der Speicherkondensatoren
- Spannungsniveaus der Speicherkondensatoren C1 und C2

Für die Verwaltung der integrierten ADC wird die Standardkomponente `ADCsC` von TinyOS verwendet [27]. Diese Komponente generiert asynchrone-Events (`async event` [9]), um das Ende einer Konvertierung mitzuteilen. Zur Vermeidung von Race-Conditions werden im ADC-Modul die Konvertierungen sequentiell ausgeführt. Die Reihenfolge entspricht der oben gezeigten Liste. Die *Versorgungsspannung* V_{CC} wird als Referenzspannung für die ADC-Werte herangezogen, weicht diese von den vorausgesetzten 3,3V ab, wird die interne Referenzspannungsquelle [36] verwendet. Dieses Verhalten kann auftreten, wenn der verwendete Spannungsregler nicht in der Lage ist, die geforderte Versorgungsspannung von 3,3V auszuregeln. Der Nachteil der internen Referenzspannungsquelle ist die relative Ungenauigkeit des Spannungsniveaus von annähernd ±10%. Die *Referenzspannung der Speicherkondensatoren* wird zur Berechnung der tatsächlichen Spannungsniveaus der Speicherkondensatoren verwendet. Diese Notwendigkeit ergibt sich aus der Beschaltung der Speicherkondensatoren (ersichtlich in Kapitel 4.2.1).
Die verwendeten Schnittstellen für das Modul ADC wird in Tabelle 4.24 dargestellt.

command result_t SensorADCData.getData()	
Beschreibung	Startet die Messreihe
Rückgabewert	Status der ersten AD-Konvertierung durch `FAIL` oder `SUCCESS`
event result_t dataReady(adcDataTable *data)	
Beschreibung	Benachrichtigung wenn alle Messungen durchgeführt und ohne Fehler konvertiert wurden.
Parameter	`adcDataTable`...Pointer auf Ergebnisse durch Verwendung der ADC-Messwerte Struktur (`struct adcDataTable`)

Tabelle 4.24: Schnittstellen Modul ADC

4.3.3.4 Modul Energiemanagement

Dieses Modul stellt dem Benutzer einfache Schnittstellen für das Energiemanagement zur Verfügung. Es werden Standardkomponenten von TinyOS in diesem Modul abstrahiert und verwaltet. Folgende Standardkomponenten werden verwendet:

- **HPLPowerManagementM** für Prozessor-Energiemanagement
- **CC2420RadioC** für Radio-Energiemanagement

Mit der Standardkomponente *HPLPowerManagementM* kann der Betriebszustand des Prozessors geändert werden. Im Wesentlichen entscheidet TinyOS welcher Energiesparmodus bei aktiviertem Energiemanagement für den Prozessor gewählt wird. Diese Entscheidungsstruktur hängt von mehreren Faktoren ab, wie zum Beispiel ob

Timer aktiv sind oder auf externe Ereignisse gewartet wird. Die Aktivierung erfolgt mit `HPLPowerManagementM.Enable()`, die Deaktivierung mit `HPLPowerManagementM.Disable()`. Beide `command`-Aufrufe müssen mit `PowerManagement.adjustPower()` bestätigt werden.

Die Standardkomponente *CC2420RadioC* ermöglicht dem Benutzer, den Radio-Hardwareblock in einen Schlafzustand zu versetzen. Dies wird durch die Standardschnittstelle `StdControl` mittels `stop()` ermöglicht. Um den Hardwareblock wieder zu aktivieren wird `start()` verwendet. Damit kann der Betriebszustand ohne nennenswerte Zeitverzögerung umgeschaltet werden. Des Weiteren kann mit dieser Standardkomponente die Sendeleistung während der Laufzeit geändert werden. Folgende Einstellungen sind möglich:

- `TXPOWER_M25DBM` entspricht -25dBm
- `TXPOWER_M15DBM` entspricht -15dBm
- `TXPOWER_M10DBM` entspricht -10dBm
- `TXPOWER_M5DBM` entspricht -5dBm
- `TXPOWER_M3DBM` entspricht -3dBm
- `TXPOWER_0DBM` entspricht 0dBm
- `CC2420_TXPOWER`

Mit `CC2420_TXPOWER` wird eine Sendeleistung von `TXPOWER_0DBM` bestimmt. Dieser Wert definiert den Initialisierungswert von TinyOS v1.1.
Die zur Verfügung gestellten Schnittstellen sind in Tabelle 4.25 definiert.

void enableCPUSleep()	
Beschreibung	Aktivierung des CPU-Energiemanagement. Abstraktion von `HPLPowerManagementM`-Funktionen
void disableCPUSleep()	
Beschreibung	Deaktivierung des CPU-Energiemanagement. Abstraktion von `HPLPowerManagementM`-Funktionen
command result_t RadioPM.start()	
Beschreibung	Aktivieren des Radio-Hardwareblocks zum Senden und Empfang von Nachrichten
Rückgabewert	Status der Betriebszustandsänderung durch `FAIL` oder `SUCCESS`
command result_t RadioPM.stop()	
Beschreibung	Änderung des Betriebszustand in `RadioSleep`
Rückgabewert	Status der Betriebszustandsänderung durch `FAIL` oder `SUCCESS`
command void RadioControl.SetRFPower(uint8_t power)	
Beschreibung	Änderung der Sendeleistung des Radio-Hardwareblocks
Parameter	`power`...Definierte Sendeleistungswerte durch TinyOS

Tabelle 4.25: Schnittstellen Modul Energiemanagement

4.3.3.5 Konfiguration

Die Konfiguration der Sensorknoten ist Teil der Applikation und dementsprechend ereignisgesteuert. Folgende zwei Arten der Konfiguration wurden implementiert:

- **Knoten-initiierte Konfiguration** wird durch eine Anfrage des Sensorknoten an die Basisstation mit einer direkten Konfigurationssequenz durchgeführt. Nur der anfragende Knoten wird entsprechend konfiguriert.
- **Benutzer-initiierte Konfiguration** wird durch den Benutzer ausgelöst, alle Knoten werden gleichzeitig von der Basisstation konfiguriert.

Beide Konfigurationsmethoden benutzen jedoch dieselbe Konfigurationssequenz `configureNode()`. Die vom Sensorknoten empfangenen Nachrichtenpakete werden von dieser Funktion überprüft und entsprechend ausgewertet. Die einzelnen Konfigurationspakete und deren Inhalt sind weiter unten in Kapitel 4.3.5.2 aufgelistet.

Für eine erfolgreiche Konfigurationssequenz muss eine festgelegte Reihenfolge für den Empfang der Pakete eingehalten werden. Folgende Liste zeigt die implementierte Reihenfolge anhand der Paketnamen:

1. `radioConfigMsg`
2. `eventLightConfigMsg`
3. `debugMsg`
4. `switchesConfigMsg`
5. `meteringConfigMsg`
6. `helloMsg`

Tatsächlich vorausgesetzt werden nur die Positionen für das erste Paket `radioConfigMsg` und das letzte Paket `helloMsg` um die Konfigurationssequenz erfolgreich abzuschließen. Die restlichen Konfigurationspakete können eine beliebige Reihenfolge haben. Im Konfigurationspacket `radioConfigMsg` sind alle Informationen für die Netzwerk- und Radio-Konfiguration enthalten. Eine laufende Messsequenz wird gestoppt um die Konfigurationssequenz nicht zu unterbrechen und der Watchdog-Timer für die Konfiguration wird initialisiert. Das Paket `helloMsg` wird benötigt um die Netzwerkkonfiguration abzuschließen (im Kapitel 4.1.5 genauer erläutert) und um das Radio-Energiemanagement zu aktivieren.

Eine weitere Bedingung, um die Kommunikationssequenz erfolgreich abzuschließen, ist der Empfang aller Konfigurationspakete innerhalb einer definierten Zeitspanne. Dafür wird ein sogenannter Watchdog-Timer eingesetzt. Initialisiert mit `restartConfigurationTimer()` überprüft dieser am Ende der festgelegten Zeitspanne eine interne Zählvariable welche den Zustand der Konfiguration zeigt (`statusConfig`). Für jedes erfolgreich empfangenes und überprüftes Paket wird diese Zählvariable inkrementiert. Werden durch ein Kommunikationsproblem ein oder mehrere Pakete nicht empfangen, kann damit auf eine unvollständige Konfiguration reagiert werden. Ist diese Bedingung nicht erfüllt wird der Sensorknoten mit `Reset.reset()` neu gestartet. Nach einem Neustart wird die *knoten-initiierte Konfiguration* erneut ausgeführt. Die zur Kompilierungszeit festgelegten Zeitspannen variieren

für jeden Sensorknoten. Dadurch kann gleichzeitigen Konfigurationsanfragen, welche zu Netzwerkkollisionen und folgend zu Paketverlusten führen, effektiv entgegengewirkt werden.

Für die Konfiguration sind in der Tabelle 4.26 gezeigten Schnittstellen vorhanden.

void configureNode(uint8_t *msg, uint8_t length)	
Beschreibung	Überprüfung und Ausführung der übergebenen Konfigurationspackete
Parameter	`msg`...Pointer auf Konfigurationspackete
	`length`...Länge des Konfigurationspakets in Bytes
void restartConfigurationTimer()	
Beschreibung	Neustart des Watchdog-Timers für die Konfigurationssequenz

Tabelle 4.26: Schnittstellen Konfiguration Sensorknoten

4.3.4 Basisstation

Folgende Funktionen wurden aufgrund der Anforderungen aus Kapitel 3.3.3 für die Basisstation implementiert:

- Weiterleitung der Nachrichtenpakete vom Host mittels Radio-Kommunikation zum Sensorknoten
- Weiterleitung der Nachrichtenpakete von Sensorknoten mittels serieller Kommunikation zum Host
- Konfiguration aller Sensorknoten durch Benutzerinteraktion
- Konfiguration einzelner Sensorknoten durch dedizierte Anfrage

Die gesamte Applikation ist wie bei den Sensorknoten ereignisbasierend in TinyOS [27] implementiert. In Abbildung 4.25 sind die verwendeten Module der Basisstation ersichtlich.

4.3.4.1 Modul Serielle Kommunikation

Wie oben bereits erwähnt, dient die Basisstation als Schnittstelle zwischen den Sensorknoten und dem Host. Um mit dem Host zu kommunizieren, wird die serielle Hardware-Schnittstelle (UART) des Atmel ATmega128L [36] verwendet. Der MICAz-Knoten der Basisstation ist mit dem *MIB520CB Mote Interface Board* [31] erweitert. Diese Zusatzplatine stellt eine Seriell-zu-USB-Schnittstelle zur Verfügung. Somit kann die Basisstation durch ein Standard USB-Verlängerungskabel mit dem Host verbunden werden. Die Basisstation bezieht seine Stromversorgung über die USB-Schnittstelle des Host-Rechners.

Um über die UART-Schnittstelle zu kommunizieren, wird die Standardkomponente `UART` von TinyOS v1.1 verwendet. Diese Komponente stellt eine byteweise Übertragung zur Verfügung. Die Übertragungsgeschwindigkeit wird mit 9600 Baud (Standardkonfiguration von TinyOS v1.1) zur Kompilierungszeit festgelegt. Es werden auch höhere Übertragungsgeschwindigkeiten unterstützt, diesen werden jedoch aufgrund der Anforderungen nicht benötigt und wurden auch nicht getestet.

Für eine stabile Kommunikation mit dem Host wurde das Modul für die serielle Kommunikation auf der Standardkomponente `PacketNoCRC` von TinyOS v1.1 aufgebaut. Dafür

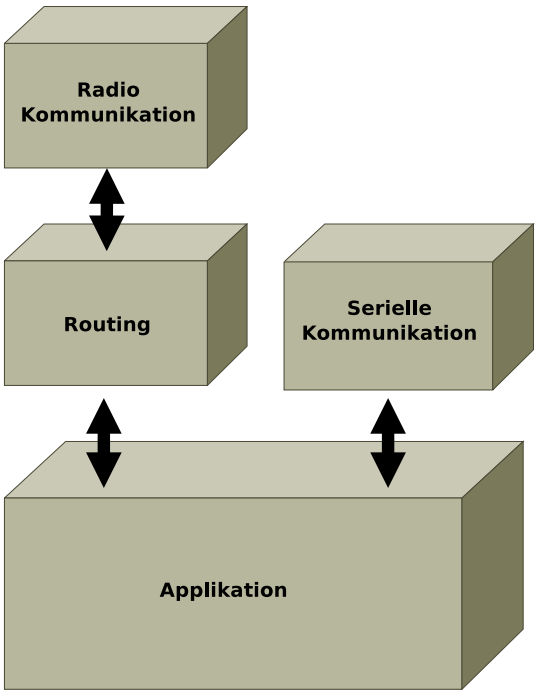


Abbildung 4.25: Blockschaltbild Basisstation

wurde die vorhandene Komponente soweit bereinigt, dass nur die benötigte Funktionen, inklusive der doppelten Bufferung für empfangene Nachrichten, erhalten blieben. In Tabelle 4.27 sind die implementierten Schnittstellen aufgelistet.

command result_t send(uint8_t *msg, uint8_t length)	
Beschreibung	Übermittelt die übergebene Nachricht an den Host
Parameter	`msg`...Pointer auf Buffer der übertragen werden soll `length`...Länge des Buffers in Bytes
Rückgabewert	Status der Übertragung durch `FAIL` oder `SUCCESS`
event result_t sendDone(result_t success)	
Beschreibung	Benachrichtigung wenn die Übertragung von einem Nachrichtenpaket zum Host abgeschlossen ist
Parameter	`success`...Status der abgeschlossenen Übertragung durch `FAIL` oder `SUCCESS`
event uint8_t *receive(uint8_t *msg, uint8_t length)	
Beschreibung	Benachrichtigung wenn ein Nachrichtenpaket vom Host empfangen wurde
Parameter	`msg`...Pointer auf den Buffer der empfangenen Nachricht `length`...Länge der Nachricht in Bytes

Tabelle 4.27: Schnittstellen Modul serielle Kommunikation

4.3.4.2 Modul Radio Kommunikation

Dieses Modul entspricht den bereits oben beschriebenen Sensorknoten-Modul Radio Kommunikation in Kapitel 4.3.3.1.

4.3.4.3 Modul Routing

Da es sich bei dem Routing-Modul um dieselbe Implementierung für die Sensorknoten (in Kapitel 4.3.3.2 Modul Routing beschrieben) und der Basisstation handelt, werden in diesem Teil nur die dedizierten Funktionen für die Basisstation beschrieben. Diese Funktionen sind folgend aufgelistet:

- **Hallo-Paket** Generierung, um Kommunikationspfade im Netzwerk zu etablieren
- **Nachrichtenverfolgung** ermöglicht dem Benutzer die Kommunikationspfade im Netzwerk zu evaluieren

Die Eigenschaften und der Einsatz des *Hallo-Paketes* ist weiter oben in Kapitel 4.1.5 ausführlich beschrieben. Das Hallo-Paket kommt in Abhängigkeit vom Routing-Protokoll (`routingType`) zum Einsatz um das Netzwerk zu erkunden. Implementiert wurde die Möglichkeit eine einmalige oder kontinuierliche Generierung zu etablieren. Die kontinuierliche Generierung wird über einen konfigurierbaren Timer realisiert. Mit dem zusätzlichen Parameter für Routing-Protokolle (`routingOptions`) kann das Zeitintervall in Sekunden angegeben werden. Wird der Schnittstelle `RadioSendMsg.send()` der Parameter `address` mit `HELLO_PACKAGE_ADDRESS` übergeben, wird der Inhalt der Nachricht lokal gespeichert und das Timer-Setup durchgeführt. Anschließend wird das erste Hallo-Paket an alle Nachbarn übermittelt. Soll eine kontinuierliche Generierung stattfinden und der Timer-Interrupt steht an, wird mit dem zuvor lokal gespeicherten Inhalt `RadioSendMsg.send()` erneut aufgerufen. Dieser Ablauf kann nur mit einer Neukonfiguration des Sensorknotens unterbrochen werden.

Die Möglichkeit der *Nachrichtenverfolgung* findet nur im Routing-Modul der Basisstation Verwendung. Aufgrund des verwendeten Broadcasting aller Teilnehmer im Netzwerk kann somit die Basisstation alle Nachrichtenpakete empfangen und entsprechend auswerten. Wird ein Paket mittels `ReceiveMsgUpStream.receive` oder `ReceiveMsgDownStream.receive` empfangen, erfolgt eine Benachrichtigung an die Applikation mit der in Tabelle 4.28 gezeigten Schnittstelle. Die notwendigen Informationen für die Schnittstelle sind im Paket-Kopf der abgefangenen Nachricht enthalten. In der Applikation werden die Informationen in das Nachrichtenpaket `packageTracerMsg` kopiert und an den Host weitergeleitet.

event void receivedPackageTrace(uint16_t sourceAddr, uint16_t fromAddr, uint16_t toAddr, uint8_t packageID)	
Beschreibung	Benachrichtigung wenn ein Paket für die Nachrichtenverfolgung empfangen wurde. Hinweis: Wird nur für die Basisstation verwendet, wird zur Laufzeit über die ID des Knoten überprüft.
Parameter	`sourceAddr`...Adresse des Knoten welcher der Ursprung der Nachricht ist `fromAddr`...Adresse des Knoten welcher zuletzt die Nachricht weitergeleitet hat `toAddr`...Adresse des Empfängerknotens `packageID`...Eindeutige ID des Paketes

Tabelle 4.28: Schnittstellen Modul Routing für Basisstation

4.3.4.4 Konfiguration

Dieser Abschnitt besteht inhaltlich aus zwei Teilen. Erstens, wie die Konfiguration der Sensorknoten implementiert ist. Zweitens, wie die Basisstation selbst konfiguriert wird.

Wie bereits in Kapitel 4.3.3.5 beschrieben, sind zwei Möglichkeiten für eine Konfiguration der Sensorknoten implementiert.

Knoten-initiierte Konfiguration: Wird eine Konfigurationsanfrage von einem Sensorknoten durch `RoutingConfigI.receivedRawMsg()` über den Raw-Stream empfangen, dann löst dies den Aufruf von `requestConfiguration()` aus. Dadurch wird eine Sequenz initiiert, die alle benötigten Konfigurationspakete an den Sensorknoten übermittelt. In Abbildung 4.24 wird dies zwecks Übersichtlichkeit vereinfacht dargestellt. Tatsächlich wird `sendRawMsg()` für jedes einzelne Konfigurationspaket sequenziell ausgeführt. Der Zustand der State-Machine wird durch die lokale Variable `statusConfig` gespeichert. Für jedes erfolgreich versandte Konfigurationspaket, Benachrichtigung durch `sendDoneRawMsg()`, wird der Zustand aktualisiert und eine Zeitverzögerung von 1ms mit `DelayTimer.start()` bis zum Versand des nächsten Paketes verwendet. Mit der Zeitverzögerung wird garantiert, dass der Sensorknoten das Konfigurationspaket fertig ausgewertet hat und für den Empfang des nächsten Paketes zur Verfügung steht. Folgende Übertragungsreihenfolge der Konfigurationspakete wurde implementiert:

1. `radioConfigMsg`
2. `eventLightConfigMsg`
3. `debugMsg`
4. `switchesConfigMsg`
5. `meteringConfigMsg`
6. `helloMsg`

Das `helloMsg`-Paket wird jedoch gesondert behandelt und mit `RadioSendHelloPackage()` via Up-Stream übermittelt. Damit wird der finale Zustand der State-Machine erreicht und die Konfigurationssequenz wird beendet. **Anmerkung:** Die *knoten-initiierte Konfiguration* kann nur dann benutzt werden, wenn zuvor eine `benutzer-initiierte Konfiguration` durchgeführt wurde. Die Konfigurationspakete werden lokal gespeichert wenn der Benutzer eine Konfiguration durchführt, diese werden dann für eine *knoten-initiierte Konfiguration* herangezogen.

Benutzer-initiierte Konfiguration: Konfigurationspakete, die vom Host mittels `UART-ReceiveMsg.receive()` empfangen werden, werden durch `RadioSendDataPrepare()` überprüft und für die Übermittlung an die Sensorknoten vorbereitet und lokal gespeichert. Die angepassten Pakete werden durch `RadioSendMsg.send()` an das Routing-Modul übergeben und mittels Broadcasting an die Sensorknoten übermittelt.

Die Konfiguration der Basisstation geschieht bei jeder *benutzer-initiierte Konfiguration*. Für jedes Konfigurationspaket wird in `RadioSendDataPrepare()` die jeweilige Konfiguration durchgeführt, vergleichbar mit der Konfigurationssequenz der Sensorknoten. Die benötigten Funktionen für die Konfigurationen sind in Tabelle 4.29 ersichtlich.

void task requestConfiguration()	
Beschreibung	Konfigurationssequenz für eine Konfigurationsanfrage von Sensorknoten. Alle Konfigurationspakete werden per *Raw-Stream* an den Sensorknoten übermittelt.
void task RadioSendHelloPackage()	
Beschreibung	Startet eine Hallo-Sequenz. Kommt bei der *knoten-initiierte Konfiguration* zur Anwendung.
result_t RadioSendDataPrepare()	
Beschreibung	Konfigurationspaket vom Host wird für Radio-Kommunikation vorbereitet. Weiters wird die Basisstation konfiguriert und die Pakete lokal gespeichert.
Rückgabewert	Status Auswertung der Konfigurationspakete durch `FAIL` oder `SUCCESS`

Tabelle 4.29: Schnittstellen Konfiguration Basisstation

4.3.5 Radiopakete

In diesem Teilabschnitt werden die verwendeten Nachrichten bezüglich ihrer Struktur und Art der Verwendung spezifiziert. Des Weiteren wird auch die Struktur der Routing-Zusatzinformationen im Paket-Kopf erläutert.

Durch die Verwendung der TinyOS v1.1 Standardkomponente `GenericComm` für die Kommunikation, darf die maximale Länge der Nachricht von 29 Bytes (`TOSH_DATA_LENGTH`) nicht überschritten werden. Aufgrund der Notwendigkeit von Zusatzinformationen für das Routing-Modul (in Kapitel 4.3.3.2 für Sensorknoten, in Kapitel 4.3.4.3 für die Basisstation) reduziert sich die maximale Länge der Nachrichten noch um den Wert `ROUTING_PAYLOAD`. Damit ergibt sich in der momentan implementierten Konfiguration eine maximale Nachrichtenlänge von 17 Bytes. Aufgrund dieser Tatsache wurden inhaltlich zusammengehörige Pakete in kleinere Paketeinheiten aufgeteilt, dies wurde vor allem bei den Konfigurationspaketen durchgeführt. Jedes Nachrichtenpaket besitzt eine eigene Identifikationsnummer (`msgID`). Dieses Byte muss an erster Stelle im Array übermittelt werden, um die Nachricht korrekt zu identifizieren. Ein Nachrichtenpaket muss mindestens aus diesem Byte bestehen, mögliche nachfolgende Daten unterliegen keiner vorgegebenen Struktur. Ein Beispiel eines Nachrichtenpaketes ist in Abbildung 4.26 dargestellt:

Byte 0	Byte 1	Byte 2	Byte 3	Byte 4		Byte n
msgID	data	data	data	data	...	data

Abbildung 4.26: Beispiel eines Nachrichtenpaketes. `n` wird durch die maximal zulässige Länge der Nachricht definiert.

Das Nachrichtenpaket wird im Modul Routing noch um einen sogenannten Packet-Header erweitert. Folgende Zusatzinformationen werden dem Paket angefügt:

- **addrFrom:** Adresse des Knoten welcher zuletzt die Nachricht weitergeleitet hat
- **addrDest:** Adresse des Empfängerknotens

- **addrSource:** Adresse des Knoten welcher der Ursprung der Nachricht ist
- **packageID:** Eindeutige ID des Paketes, wird durch Zufallsgenerator festgelegt
- **seqID:** Sequenznummerierung, dient zur Vermeidung von Mehrfachzustellungen derselben Nachricht
- **linkCosts:** Verbindungskosten der Nachricht durch das Netzwerk

Die Informationen werden vor der ursprünglichen Nachricht im Byte-Array eingefügt. Für eine fixe Strukturierung des Gesamtpaketes wird ein `struct` verwendet, welches in Tabelle 4.30 dargestellt ist.

struct routingMsg	
`uint16_t addrFrom`	Adresse des Knoten welcher zuletzt die Nachricht weitergeleitet hat
`uint16_t addrDest`	Adresse des Empfängerknotens
`uint16_t addrSource`	Adresse des Knoten welcher der Ursprung der Nachricht ist
`uint8_t packageID`	Eindeutige ID des Paketes
`uint8_t seqID`	Sequenznummerierung
`uint8_t linkCosts`	Verbindungskosten
`uint8_t data[]`	Nachrichtenpaket

Tabelle 4.30: Radiopaket eventLightMsg

Diese Zusatzinformationen werden durch die Funktion `removeRoutingHeader( TOS_MsgPtr msg )` wieder entfernt wenn die Nachricht ihr Ziel erreicht hat und an die Applikation weitergereicht wird.

4.3.5.1 Vom Sensorknoten generierte Nachrichten

Bereitgestellte Messdaten vom ADC-Modul werden mittels `meteringMsg` (Tabelle 4.31) von den Sensorknoten zur Basisstation übermittelt.

struct meteringMsg	
`uint8_t msgID`	6
`uint8_t nodeID`	Adresse des Sensorknotens
`uint16_t VSolar`	Solarzellenspannung in mV
`uint16_t VCap1`	Ladespannung Speicherkondensator C1 in mV
`uint16_t VCap2`	Ladespannung Speicherkondensator C2 in mV
`uint16_t lux`	Helligkeitswert in lux

Tabelle 4.31: Radiopaket meteringMsg

Bei der Überschreitung einer definierten Ereignisschwelle für den Helligkeitswert (Kapitel 4.1.7) übermittelt der jeweilige Sensorknoten ein Nachrichtenpaket (Tabelle 4.32) an die Basisstation.

struct eventLightMsg	
uint8_t msgID	3
uint8_t nodeID	Adresse des Sensorknotens
uint16_t lux	Helligkeitswert in lux

Tabelle 4.32: Radiopaket eventLightMsg

Soll eine Fehlermeldung gemeldet werden, wird eine Nachricht erstellt, die die Adresse des Knoten sowie den Fehlercode liefert (Tabelle 4.33).

struct errorMsg	
uint8_t msgID	238
uint8_t nodeID	Adresse des Sensorknotens
uint16_t err	Fehlercode (Auflistung in Kapitel 4.3.6)

Tabelle 4.33: Radiopaket errorMsg

Bei der `requestConfigMsg` in Tabelle 4.34 handelt es sich um eine Anfrage für eine knoten-initiierte Konfiguration, diese wird ausführlich in Kapitel 4.1.6 beschrieben.

struct requestConfigMsg	
uint8_t msgID	9
uint8_t nodeID	Adresse des anfragenden Sensorknotens

Tabelle 4.34: Radiopaket requestConfigMsg

4.3.5.2 Konfigurationspakete

Die folgenden Nachrichten beinhalten die benötigten Informationen für die Konfiguration der Sensorknoten und werden von der Basisstation generiert (Tabelle 4.35 bis Tabelle 4.40).

struct radioConfigMsg	
uint8_t msgID	1
uint8_t neighbours[]	Nachbarschaftstabelle, dient zur Definition der Netzwerktopologie (Struktur in Kapitel 4.1.4)
uint8_t radioTxPower	Sendeleistung des Radio-Hardwareblock (Kapitel 4.3.3.4 Modul Energiemanagement)
uint8_t radioSleep	Aktivierung/Deaktivierung Energiesparmodus des Radio-Hardwareblock (Kapitel 4.3.3.4 Modul Energiemanagement)
uint8_t routingType	Auswahl des Routing-Protokolls (Kapitel 4.1.5)
uint8_t routingOptions	Parameter für gewähltes Routing-Protokoll (Kapitel 4.1.5)

Tabelle 4.35: Radiopaket radioConfigMsg

struct eventLightConfigMsg	
uint8_t msgID	2
uint8_t padding	Platzhalter für Address-Alignment
uint16_t threshold	Ereignisschwelle für Helligkeitswert in lux (Kapitel 4.1.7)

Tabelle 4.36: Radiopaket eventLightConfigMsg

struct helloMsg	
uint8_t msgID	4

Tabelle 4.37: Radiopaket helloMsg für Hallo-Sequenz

struct meteringConfigMsg	
uint8_t msgID	5
uint8_t measureInterval	Kommunikationsintervall für Messergebnisse (Kapitel 4.1.7)
uint8_t numOfMeasurePerInterval	Anzahl der Messungen pro Intervall (Kapitel 4.1.7)
uint8_t measureAvgFactor	Faktor für Durchschnittswertberechnung

Tabelle 4.38: Radiopaket meteringConfigMsg

struct debugMsg	
uint8_t msgID	7
uint8_t delay	Zeitverzögerung zwischen Zeitpunkt vom Empfang eines Paketes bis zur Weiterleitung
uint8_t trace	Aktivierung/Deaktivierung der Nachrichtenverfolgung
uint8_t led	Aktivierung/Deaktivierung der MICAz-LEDs

Tabelle 4.39: Radiopaket debugMsg

struct switchesConfigMsg	
uint8_t msgID	8
uint8_t enableSolarcell	Aktivierung/Deaktivierung der Solarzelle auf dem Sensorboard
uint8_t enableCap1	Aktivierung/Deaktivierung des Speicherkondensators C1 auf dem Sensorboard
uint8_t enableCap2	Aktivierung/Deaktivierung des Speicherkondensators C2 auf dem Sensorboard
uint8_t selectVReg	Auswahl für den Spannungsregler auf dem Sensorboard

Tabelle 4.40: Radiopaket switchesConfigMsg

4.3.5.3 Nachrichten von der Basisstation zum Host

Die Nachrichtenverfolgung erfolgt nur über das Routing-Modul der Basistation. Die gesammelten Informationen werden mittels `packageTracerMsg` (Tabelle 4.41) durch die serieller Kommunikation an den Host weitergeleitet.

struct packageTracerMsg	
uint8_t msgID	10
uint8_t sourceAddr	Adresse des Knoten welcher der Ursprung der Nachricht ist
uint8_t fromAddr	Adresse des Knoten welcher zuletzt die Nachricht weitergeleitet hat
uint8_t toAddr	Adresse des Empfängerknotens
uint8_t packageID	Eindeutige ID des Paketes

Tabelle 4.41: Nachrichtenpaket packageTracerMsg

4.3.5.4 Nachrichten vom Host zur Basisstation

Diese Nachrichten werden vom Host über die serielle Kommunikation zur Basisstation übermittelt.

Vor jeder benutzer-initiierten Konfiguration wird die Basisstation mittels der Funktion `resetBasestationMsg` (Tabelle 4.42) zurückgesetzt. Damit wird die Stabilität des Gesamtsystems erhöht. Die Sensorknoten werden dadurch nicht beeinflusst.

struct resetBasestationMsg	
`uint8_t msgID`	11

Tabelle 4.42: Nachrichtenpaket resetBasestationMsg

Für eine zusätzliche Stabilisierung des Gesamtsystems wird noch `heartbeatMsg` (Tabelle 4.43) eingesetzt. Damit wird in festgelegten Zeitabständen der Watchdog-Timer der Basisstation zurückgesetzt. Eventuell auftretende Probleme mit der seriellen Kommunikation können damit verhindert werden.

struct heartbeatMsg	
`uint8_t msgID`	12

Tabelle 4.43: Nachrichtenpaket heartbeatMsg

4.3.6 Fehlercodes

Fehlerberichte werden durch die Sensorknoten generiert und mittels `raw`-Nachrichten (in Kapitel 4.3.3.2 Modul Routing) an die Basisstation übermittelt. Der Fehlerbericht wird dann von der Basisstation mittels serieller Schnittstelle (Kapitel 4.3.4.1 Modul Serielle Kommunikation) an den Host weitergereicht und ausgewertet. Die möglichen Fehlercodes sind in Tabelle 4.44 aufgelistet.

4.3.7 Socket-Pakete

Alle Socket-Nachrichten vom Web-Server zu den Daemons müssen mit dem *Modul Obfuscator* (Kapitel 4.3.1.5) entschlüsselt werden. Die fertig entschlüsselte Nachricht wird mit der Datenstruktur `of_message` vom Obfuscator bereitgestellt. Die Implementierung von *of_message* ist im Listing 4.1 dargestellt.

```
typedef struct {
  of_entry header;                // fixed header
  of_entry length;                // calced length bytes
  of_entry type;                  // action type
  of_entry param[OF_MAX_PARAM];   // parameters
  of_entry magic;                 // magic signature
} of_message;
```

Listing 4.1: Struktur von of_message

Fehler	Code	Beschreibung
ERROR_MSG_NEIGHBOURS	0xE0	Fehler bei Konfigurationsnachricht `radioConfigMsg`
ERROR_MSG_EVENT_LIGHT	0xE1	Fehler bei Konfigurationsnachricht `eventLightConfigMsg`
ERROR_MSG_EVENT_LIGHT_FIRED	0xE2	Fehler bei Helligkeitsereignis
ERROR_MSG_HELLO_PACKET	0xE3	Fehler während der Hallo-Sequenz
ERROR_MSG_CONFIG_METERING	0xE4	Fehler bei Konfigurationsnachricht `meteringConfigMsg`
ERROR_MSG_DEBUG_CONFIG	0xE5	Fehler bei Konfigurationsnachricht `debugMsg`
ERROR_MSG_CONFIG_SWITCHES	0xE6	Fehler bei Konfigurationsnachricht `switchesConfigMsg`
ERROR_MSG_UNKNOWN	0xE7	Kommunikationsfehler durch unbekannte `msgID`
ERROR_MSG_SEQUENCE	0xE8	Fehler in der Kommunikationssequenz
ERROR_MSG_BUFFER	0xE9	Maximale Bufferlänge für serielle Kommunikation überschritten
ERROR_COM_UART	0xEA	Problem bei der seriellen Kommunikation, unvollständige Nachricht
ERROR_CONFIGURATION_REQUEST	0xEB	Fehler während einer knoten-initiierte Konfiguration
ERROR_COM_RADIO	0xEC	Überschreitung der Länge eines Radiopaketes mit maximal MSG_LEN_MAX Bytes

Tabelle 4.44: Fehlercodes

Als `header` wird bei jeder Nachricht `0x53505921` verwendet. Die Möglichkeiten für `type` definieren, um welche Art Nachricht es sich handelt, in `param` sind die dazugehörigen Parameter abgebildet. In Tabelle 4.45 sind alle Socket-Nachrichten aufgelistet.

Art der Nachricht	type	param
activate node	0xDCDC0201	1 Byte Knoten ID
deactivate node	0xDCDC5301	1 Byte Knoten ID
set ambient light level	0xACAC7701	2 Bytes für Level
set event light level	0xACACB901	2 Bytes für Level
start timeline	0xEC574800	
stop timeline	0xEC570B00	
dnicard shutdown	0xDEADDEAD	
load configuration from db	0x3E5C3C01	4 Bytes Konfigurations ID
dbasestation shutdown	0xDEADDEAD	

Tabelle 4.45: Socket-Packete

5 Ergebnisse

Diese Kapitel zeigt Messergebnisse von den permanent versorgten und den selbstversorgenden Sensorknoten mit Energiegewinnungssystemen. Alle Messungen wurden über den Web-Zugang durchgeführt, die exportierten Messdaten wurden mit Octave (kompatibel zu Matlab, jedoch Open-Source) visualisiert.

Die durchschnittliche Leistungsaufnahme der Sensorknote kann berechnet werden durch:

$$P_{average} = \frac{1}{t_{interval}} \cdot \sum_{i=1}^{n} P_i \cdot t_i$$

Mit $t_{interval}$ ist die gesamte Intervallzeit gemeint, welche der Summe von t_1 bis t_n entspricht. P_i entspricht der Leistungsaufnahme eines Intervalls t_i.

5.1 Permanent versorgte Sensorknoten

Bei den permanent versorgten Sensorknoten besteht die Möglichkeit die Stromaufnahmen der einzelnen Knoten zu messen. Diese Messungen eignen sich zur Evaluierung von unterschiedlichen Energiemanagementeinstellungen. Folgende Einstellungen wurden über den Web-Zugang konfiguriert:

- Kommunikationsintervall: *5s*
- Anzahl der Messungen pro Intervall: *1*
- Topologie: *Stern*
- Routing-Protokoll: *kürzester Pfad*
- Status LEDs: *Aus*

Abbildung 5.1 zeigt die Stromaufnahme von einem Sensorknoten ohne jegliche Optimierung des Energiemanagements. Einfach erkennbar ist das Kommunikationsintervall von 5s durch die teilweise erhöhte Stromaufnahme. Die Messdaten werden erst kurz vor dem Versand akquiriert und verarbeitet, das erzeugt einen Anstieg der Stromaufnahme um fast 1mA. Die kurzzeitig erhöhte Stromaufnahme des Radio-Moduls beim Senden der Nachricht ist wegen der geringen Sampling-Rate leider nicht möglich.

Im Vergleich zur oben beschriebenen Messung besteht der Unterschied der Messung in Abbildung 5.2 darin, dass das Radio-Modul nur zum Versenden der Messdaten aktiviert wird. Beim selben Kommunikationsintervall von 5s ist die wesentlich reduzierte durchschnittliche Stromaufnahme leicht zu erkennen. Erkennbar ist ebenfalls die geringe Stromaufnahme (<1mA) des Prozessors im Idle-Zustand.

Der durchschnittliche Stromverbrauch und die berechnete durchschnittlich aufgenommene Leistung ist in Tabelle 5.1 ersichtlich.

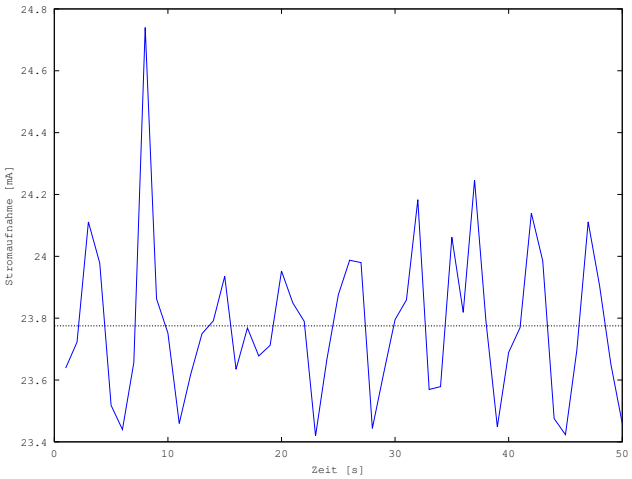


Abbildung 5.1: Messung der Stromaufnahme eines Sensorknoten ohne jegliche Optimierung in Bezug auf den Energieverbrauch. Die gepunktete Linie zeigt die durchschnittliche Stromaufnahme der Messreihe.

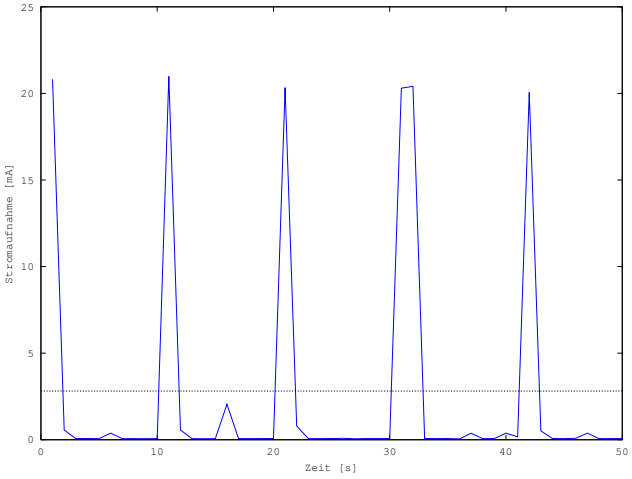


Abbildung 5.2: Messung der Stromaufnahme eines Sensorknoten mit deaktiviertem Radio-Modul während der Idle-Phase. Die gepunktete Linie zeigt die durchschnittliche Stromaufnahme der Messreihe.

	Stromaufnahme	Leistungsaufnahme
ohne Optimierung	23,78mA	78,47mW
mit Optimierung (sleep beim Radio-Modul)	2,4mA	7,92mW

Tabelle 5.1: Durchschnittliche Stromaufnahme und berechnete Leistungsaufnahme von permanent versorgten Sensorknoten bei einer Versorgungsspannung von Vcc=3,3V

5.2 Energiegewinnende Sensorknoten

Bei den selbstversorgenden Sensorknoten durch Solarzellen können mit Hilfe des Energiegewinnungssystems Spannungsmessungen der Solarzelle und der beiden Speicherkondensatoren durchgeführt werden. Die Konfigurationsmöglichkeiten bei den Spannungsreglern sowie die Anzahl der benutzten Speicherkondensatoren ergeben eine große Auswahl an Szenarien.

In den folgenden Abschnitten werden zwei Szenarien beschrieben und entsprechende Messungen durchgeführt. Bei der Messung wurde die maximale Helligkeit (255) über den Web-Zugang eingestellt. Die Speicherkondensatoren unterscheiden sich durch deren Kapazität, wobei C1 einen Kapazitätswert von 0,5F und C2 von 1,5F aufweist. Bei den Messungen wurde keine Optimierung der Stromaufnahme verwendet, um einen konstanten Lade- und Entladestrom zu erzeugen. Folgende Einstellungen wurden über den Web-Zugang konfiguriert:

- Kommunikationsintervall: *1s*
- Anzahl der Messungen pro Intervall: *1*
- Topologie: *Stern*
- Routing-Protokoll: *kürzester Pfad*
- Status LEDs: *Aus*

Kennlinien der Speicherkondensatoren C1 und C2 mit Linearregler

Die Abbildungen 5.3 und 5.4 zeigen die Verläufe der gemessenen Lade- und Entladekennlinien von C1 und C2 mit dem Längsregler TPS78033022. Der MICAz-Knoten [31] besitzt eine minimale Versorgungsspannung von V_{CCmin}=2,8V. Der Längsregler hat laut Datenblatt [40] eine Dropout-Spannung von V_{DO}=200mV. Dadurch sollte sich für den Betrieb der Sensorknoten eine minimale Kondensatorspannung von 3V ergeben.

In der Tabelle 5.2 sind die Ergebnisse der beiden Sensorknoten für die jeweiligen Kondensatoren C1 sowie C2 gegenübergestellt. Die minimale Kondensatorspannung entspricht dem letzten übertragenen Messwert der Sensorknoten vor deren Abschaltung.

Die Unterschiede bei den maximalen Spannungen der Solarzellen und der Kondensatoren ergeben sich aus den Toleranzen der verwendeten Komponenten. Dasselbe gilt auch für die unterschiedlichen Laufzeiten der Sensorknoten, bedingt durch die Toleranzen bei den Kapazitätswerten.

Kennlinien der Speicherkondensatoren C1 und C2 mit DCDC-Konverter

Die Abbildungen 5.5 und 5.6 zeigen die Verläufe der gemessenen Lade- und Entladekennlinien von C1 und C2 mit dem Buck-Boost-Regler TPS63031. Wie bereits oben erwähnt, besitzt der MICAz-Knoten [31] eine minimale Versorgungsspannung von V_{CCmin}=2,8V. Der Buck-Boost-Regler hat laut Datenblatt [39] eine minimale Eingangsspannung von U_{Ityp}=1,8V. Dadurch sollte sich für den Betrieb der Sensorknoten eine minimale Kondensatorspannung von 1,8V ergeben.

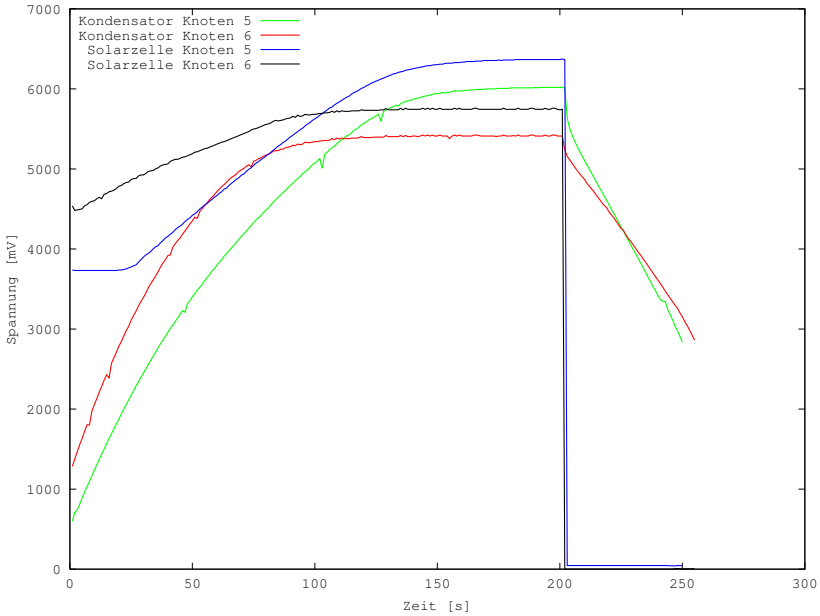


Abbildung 5.3: Spannungsverläufe von Solarzelle und Speicherkondensator C1 mit Längsregler ohne Optimierung in Bezug auf den Energieverbrauch

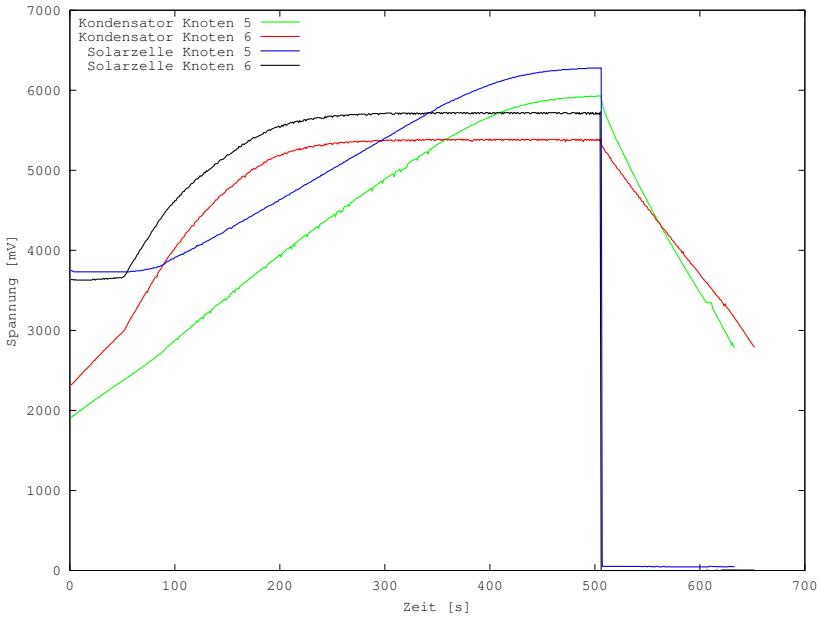


Abbildung 5.4: Spannungsverläufe von Solarzelle und Speicherkondensator C2 mit Längsregler ohne Optimierung in Bezug auf den Energieverbrauch

	Kondensator C1		Kondensator C2	
	Knoten 5	Knoten 6	Knoten 5	Knoten 6
max. Solarzellenspannung	6,36V	5,74V	6,28V	5,69V
max. Kondensatorspannung	6,02V	5,41V	5,92V	5,37V
min. Kondensatorspannung	2,84V	2,89V	2,78V	2,79V
Entladezeit	47s	52s	127s	145s

Tabelle 5.2: Ergebnisse der Lade- und Entladekennlinien von beiden Sensorknoten mit dem Längsregler

In der Tabelle 5.3 sind die Ergebnisse der beiden Sensorknoten für die jeweiligen Kondensatoren C1 sowie C2 gegenübergestellt. Die Kennlinien sind im Vergleich zur Messung mit dem Längsregler mit erhöhten Messungenauigkeiten versehen. Für diese Auswirkungen könnten die nicht vorhandenen EMV-Schutzmaßnahmen in Kombination mit dem oszillierenden Buck-Boost-Konzept auf der Energiegewinnungssystem-Platine verantwortlich sein. Die Ergebnisse wurden aufgrund der starken Varianz gemittelt. Die minimale Kondensatorspannung entspricht dem letzten übertragenen Messwert der Sensorknoten vor deren Abschaltung.

Die Unterschiede bei den maximalen Spannungen der Solarzellen und der Kondensatoren ergeben sich aus den Toleranzen der verwendeten Komponenten. Dasselbe gilt auch für die unterschiedlichen Laufzeiten der Sensorknoten, bedingt durch die Toleranzen bei den Kapazitätswerten.

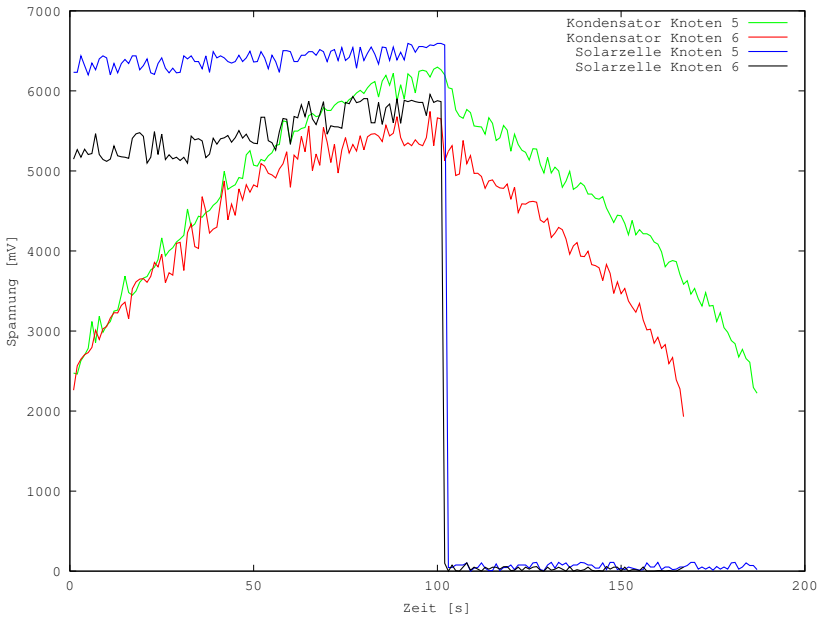


Abbildung 5.5: Spannungsverläufe von Solarzelle und Speicherkondensator C1 mit Buck-Boost-Regler ohne Optimierung in Bezug auf den Energieverbrauch

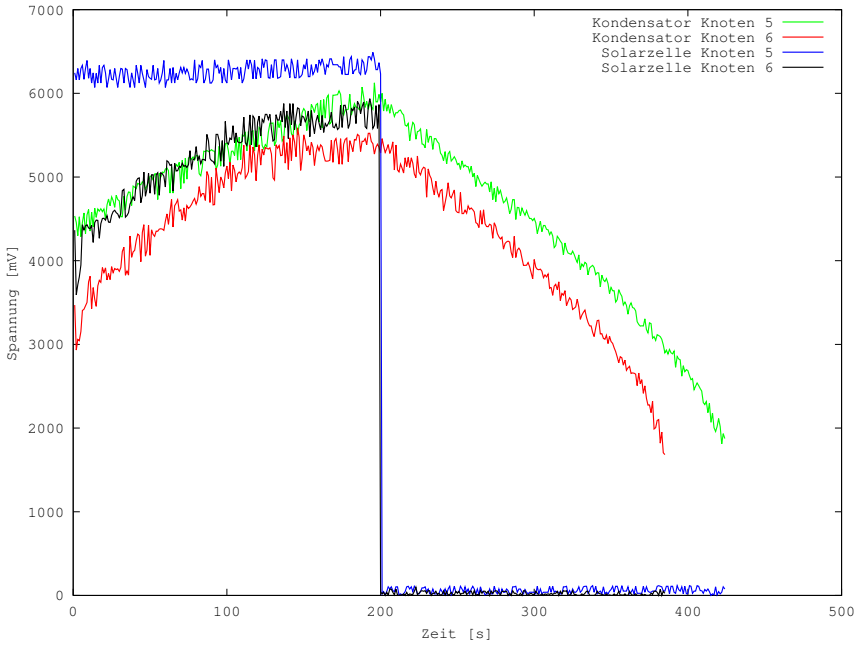


Abbildung 5.6: Spannungsverläufe von Solarzelle und Speicherkondensator C2 mit Buck-Boost-Regler ohne Optimierung in Bezug auf den Energieverbrauch

	Kondensator C1		Kondensator C2	
	Knoten 5	Knoten 6	Knoten 5	Knoten 6
max. Solarzellenspannung	6,63V	5,81V	6,37V	5,76V
max. Kondensatorspannung	6,22V	5,54V	5,97V	5,41V
min. Kondensatorspannung	2,24V	1,89V	1,83V	1,70V
Entladezeit	84s	65s	221s	185s

Tabelle 5.3: Ergebnisse der Lade- und Entladekennlinien von beiden Sensorknoten mit dem Buck-Boost-Regler

5.3 Messabweichung der Messkarte NI-USB-6211

In diesem Abschnitt werden die Messdaten der Messkarte mit Messungen mit einem Multimeter verglichen und die Messfehler bewertet. Die relative Messabweichung f wird berechnet durch:

$$f = \frac{x_{mk} - x_{mu}}{x_{mu}} \cdot 100 = \left(\frac{x_{mk}}{x_{mu}} - 1\right) \cdot 100$$

Mit x_{mk} als ausgegebener Messwert von der Messkarte und x_{mu} als tatsächlicher Messwert vom Multimeter.

Die Spannungen der Kondensatoren C1 beider Sensorknoten wurden bei unterschiedlichen Ladezuständen ermittelt. Gleichzeitig wurden auch die Ausgangsspannungen der Solarzellen bei unterschiedlichen Beleuchtungsstärken aufgenommen. Die Ergebnisse sind in Tabelle 5.4 dargestellt.

		Solarzellenspannung		Kondensatorspannung	
	Beleuchtung	160	255	160	255
Knoten 5	*Messkarte*	3,70V	6,19V	3,37V	5,83V
	Multimeter	3,64V	6,18V	3,35V	5,80V
	Messfehler	1,64%	0,16%	0,6%	0,52%
Knoten 6	*Messkarte*	4,90V	5,58V	4,56V	5,23V
	Multimeter	5,02V	5,68V	4,67V	5,36V
	Messfehler	-2,39%	-1,76%	-2,36%	-2,43%

Tabelle 5.4: Messabweichung der Messkarte

Die Messabweichungen beider Sensorknoten bewegen sich in einem vertretbaren Bereich im Rahmen der Anforderungen. Beim Knoten 6 sind im Vergleich zu Knoten 5 tendenziell größere Messabweichungen zu erwarten, können aber im Zusammenhang mit den hohen Fertigungstoleranzen der elektronischen Komponenten vernachlässigt werden.

5.4 Durchgeführte Laborübungen

In diesem Abschnitt werden die bereits durchgeführten Laborübungen (zwei Übungen von unterschiedlichen Vorlesungen) inhaltlich kurz vorgestellt. Der Inhalt der Übungen wird ausführlich in den Task-Spezifikationen für IKT-Rechnerarchitekturen beschrieben [49].

Mobile Computing Labor WS2012
Diese Laborübung wurde mit 8 Studierenden durchgeführt und hatte folgende Inhalte:

Task 1 - Generic Structure of a Sensor Node: In diesem Task sollen sich die Studenten mit dem Aufbau und Funktionsweisen eines drahtlosen Sensornetzwerkes beschäftigen.

Task 2 - WSN Communication: Die Studenten sollen sich mit der Kommunikation in drahtlosen Sensornetzwerken befassen, verschiedene Topologien konfigurieren und deren Eigenschaften diskutieren.

IKT-Rechenarchitekturen UE WS2012
Diese Laborübung wurde mit 16 Studenten durchgeführt und hatte folgende Inhalte:

Task 1 und Task 2: Entsprechen den bereits oben beschriebenen Tasks

Task 3 - Power consumption of Nodes and Networks: In diesem Task sollen sich die Studenten mit dem Energieverbrauch der Sensorknoten in unterschiedlichen Konfigurationen beschäftigen und deren Verhalten diskutieren.

Task 4 - Energy Harvesting Enhanced WSNs: Dieser Task beinhaltet den Einsatz und die Konfiguration von selbstversorgenden Sensorknoten mit Energiegewinnungssystemen mittels Solarzellen.

Task 5 - Wireless Alarm System: Die Studenten sollen eine Art Alarmsystem zur Überwachung von Räumen entwerfen. Mit dem Einsatz von selbstversorgenden Sensorknoten soll eine kontinuierliche Überwachung von sich schnell ändernden Helligkeitswerten in den Nachtzyklen konfiguriert werden.

6 Zusammenfassung und Ausblick

Diese Masterarbeit zeigt die Entwicklung und Umsetzung einer fernsteuerbaren Laborplattform für den universitären Laborunterricht. Damit soll für Studierende ein praxisnahes Arbeiten mit drahtlosen Sensornetzwerken ermöglicht werden. Die Sensorknoten werden zum Teil permanent versorgt, um Netzwerkprotokolle und Stromaufnahme zu evaluieren. Die selbstversorgenden Sensorknoten sind nur per Funk mit dem Rest des Systems verbunden und mit Solarzellen ausgestattet, welche zur Evaluierung von Energiegewinnungssystemen eingesetzt werden. Die Benutzerkonten, die Datenbank und der Web-Zugang werden über einen zentralen Server im lokalen Netzwerk des Instituts für Technische Informatik (ITI) verwaltet. Über den Web-Zugang können das Sensornetzwerk konfiguriert und einzelne Sensorknoten ein- oder ausgeschalten werden sowie Messergebnisse und Statusinformationen für den Benutzer visualisiert werden. Das Sensornetzwerk ist deswegen über ein Gateway mit dem Server verbunden. Für die Strommessungen und steuerbaren Versorgungsleitungen wird eine externe Steuer- und Messkarte verwendet. Zur Nachbildung von Tag- und Nachtzyklen können LEDs als Leuchtmittel manuell oder automatisiert angesteuert werden.

Das System wurde bereits im Umfang von mehreren Laborübungen von Studierenden getestet und hat ein stabiles Verhalten im Echtbetrieb bewiesen.

6.1 Drahtloses Sensornetzwerk

In den folgenden Abschnitten wird kurz auf die eingesetzten Komponenten im Sensornetzwerk und deren Verhalten im Echtbetrieb eingegangen.

6.1.1 Basisstation

Die Basisstation dient als Schnittstelle zwischen dem drahtlosen Sensornetzwerk und dem Server. Dies erfordert die gleichzeitige Verwaltung von zwei Kommunikationsschnittstellen, zum einen die Radio-Kommunikation mit dem Sensorknoten und zum anderen die serielle Kommunikation mit dem Server über die USB-Schnittstelle. Durch das eingesetzte TinyOS v1.x ergaben sich jedoch Probleme mit den interrupt-gesteuerten Ereignissen (Events). Die Radio-Kommunikation kann durch *synchrone Events* verwaltet werden, die serielle Kommunikation hingegen mittels *asynchronen Events*. Wegen der Definition, dass asynchrone Events die synchronen Events zu jedem Zeitpunkt unterbrechen dürfen, muss hier ein besonderes Augenmerk auf die Verwaltung der beiden Schnittstellen gelegt werden. Im schlechtesten Fall handelt es sich um Datenverluste bei den übertragenen Nachrichtenpaketen. Als Konsequenz musste eine umfangreiche Fehlerbehandlung implementiert werden.

6.1.2 Sensorknoten

Auf den drahtlosen Sensorknoten sowie auf der Basisstation wird als Zwischenschicht in der Radio-Kommunikation dasselbe Routing-Modul eingesetzt, um eventuelle Wartungsarbeiten zu reduzieren. Der Nachteil von höherem Speicherverbrauch durch nicht benötigten Code für die jeweilige Applikation wurde bewusst in Kauf genommen.

Die permanent versorgten und selbstversorgenden Sensorknoten laufen mit demselben Image und unterscheiden sich nur durch die jeweiligen Adressen (IDs).

Auf die Energiezustände der Prozessoren und der Peripherie kann nur geringfügig Einfluss genommen werden, TinyOS v1.x übernimmt hier den Großteil des Energiemanagements.

6.1.3 Energiegewinnungssystem

Das Energiegewinnungssystem überzeugt durch seine Einfachheit und funktioniert wie erwartet. Einzig die Regelung der Speicherkondensatoren musste im Echtbetrieb angepasst werden. Durch die herstellungsbedingten Toleranzen haben die Threshold-Spannungen für baugleiche MOSFETs selten denselben Schwellwert. Die Spannungsteiler für die Threshold-Spannungen müssen solange angepasst werden, damit die Ausgangsspannung der Solarzelle nicht unter einen Pegel von 3V fällt, wenn ungeladene Speicherkondensatoren zugeschaltet werden.

6.2 Peripherie

Unter den Begriff der Peripherie fällt die Messkarte von National Instruments® sowie die notwendigen Schaltungen zum Umsetzten der Signale von und zu der Messkarte.

6.2.1 Mess- und Steuerkarte

Die Mess- und Steuerkarte NI-USB-6211 von National Instruments® hat sich für den industriellen Einsatz als optimale Lösung für dieses Projekt herausgestellt. Der einzig nennenswerte Nachteil ist die teils unzureichende Unterstützung von Linux-Distributionen. Viele brauchbare Funktionen im Treiber für Windows® wurden nicht für Linux portiert. Beachtet werden müssen auch die teilweise großen Zeitverzögerungen bei hoher USB-Schnittstellenauslastung.

6.2.2 Versorgungsschaltung

Die Versorgungsplatine dient als Messverstärker für die Strommessungen über Shunt-Widerstände und zeichnet sich durch genaue Messungen aus. Die relative Messabweichung liegt unterhalb von 2,5%.

6.2.3 LED-Treiber

Die eingesetzten Led-Treiber-Module haben durchgehend eine unzureichende interne Schutzbeschaltung gegen Verpolung oder Überspannung. Entsprechende Schutzmaßnahmen mussten auf der Platine mit geeigneten Dioden umgesetzt werden.

6.3 Server

Die implementierten Daemons der Messkarte und der Basisstation funktionieren im Echtbetrieb wie erwartet und besitzen eine hohe Toleranz bei auftretenden Fehlern.

Das eingesetzte Betriebssystem openSUSE 11.4 erhält aufgrund seines Alters keine weiteren Sicherheitsupdates mehr und stellt damit auf Dauer ein Sicherheitsrisiko dar.

6.4 Ausblick

Die folgende Auflistung zeigt mögliche Verbesserungen des Systems:

TinyOS: Der Einsatz von TinyOS v1.x zeigt einige Mängel in der Stabilität und Verwaltung von Schnittstellen auf. Mit einer Portierung auf TinyOS v2.x kann auf eine weiterentwickelte und verbesserte Version zugegriffen werden. Es müssten lediglich die HAL-Schnittstellen angepasst werden.

Messkartentreiber NI-DAQmxBase von National Instruments®: Kurz vor dem Ende der Masterarbeit wurde eine neue Version (NI-DAQmxBase v3.6) von National Instruments® zur Verfügung gestellt. Mit diesem Update wird das Betriebssystem openSUSE 12.1 unterstützt, welches wegen der bereits geäußerten Sicherheitsbedenken sobald als möglich eingesetzt werden sollte.

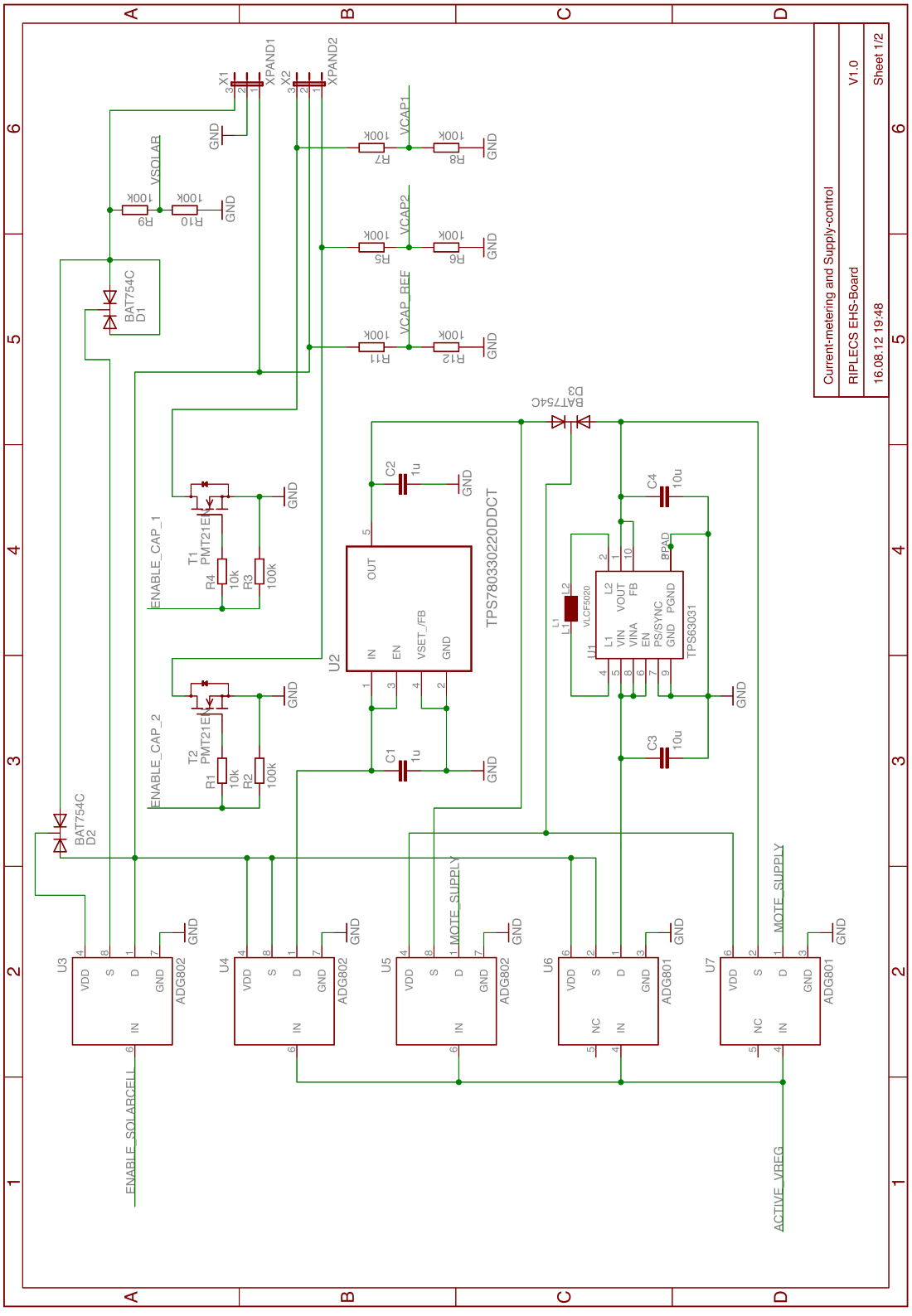
Software-Framework von Crossbow®: Durch den Einsatz von Over-The-Air-Programming (OTAP) können Images für die Sensorknoten drahtlos übertragen werden. Durch den aufwendigen Aufbau der Laborplattform hätte diese Möglichkeit ihre Berechtigung, um die Sensorknoten nicht über die Programmierplatine flashen zu müssen.

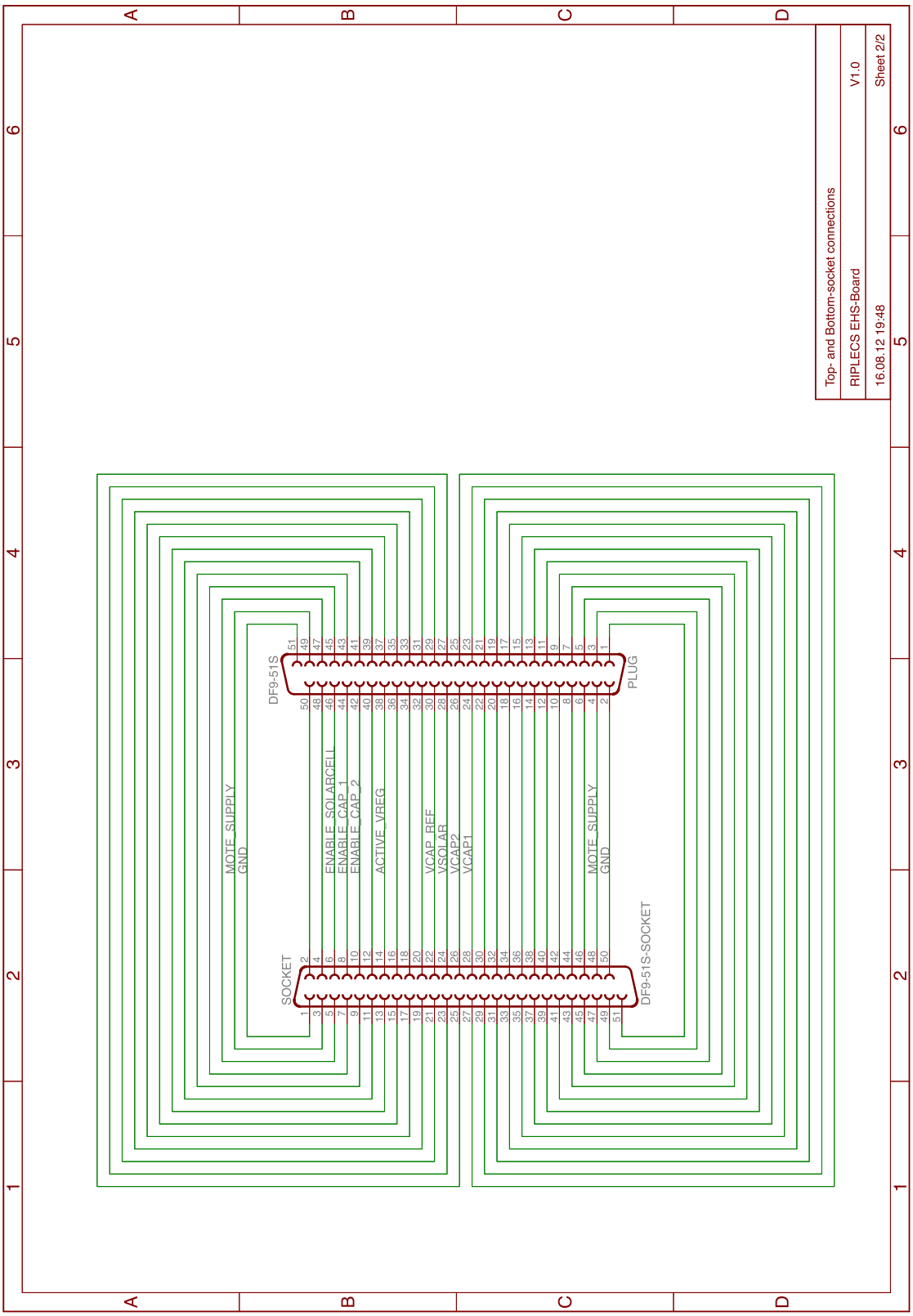
A Schaltpläne

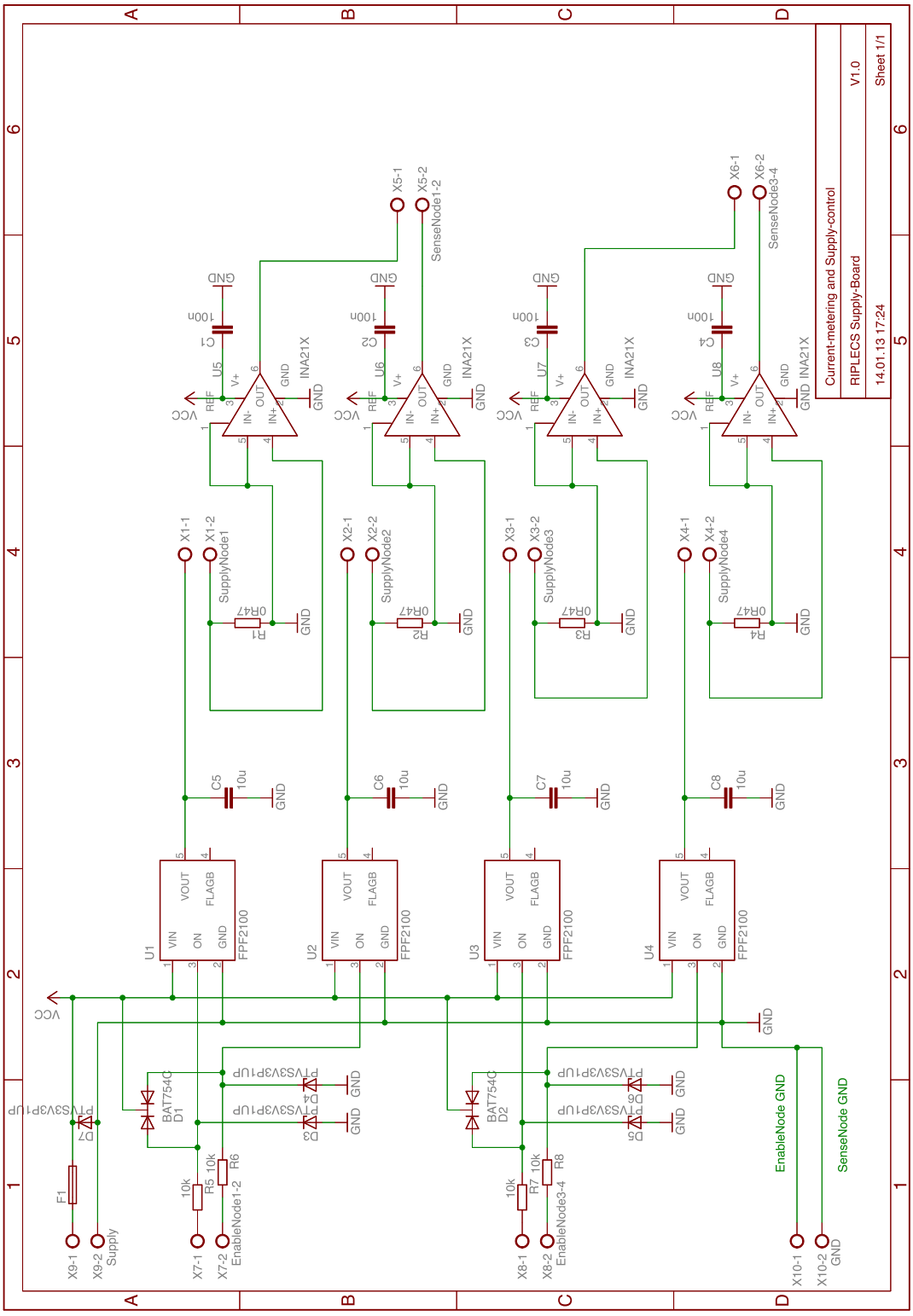
Der gesamte Schaltplan des Energiegewinnungssystems für die selbstversorgenden Sensorknoten ist auf den Seiten 95 und 96 dargestellt. Der erste Teil auf Seite 95 zeigt die Steuerung der Spannungsregler, der Kondensatoren und der Solarzelle sowie die Spannungsmessungen. Der zweite Teil auf Seite 96 zeigt die beiden Verbindungsstecker der Platine und welche Verbindungen davon genutzt werden.

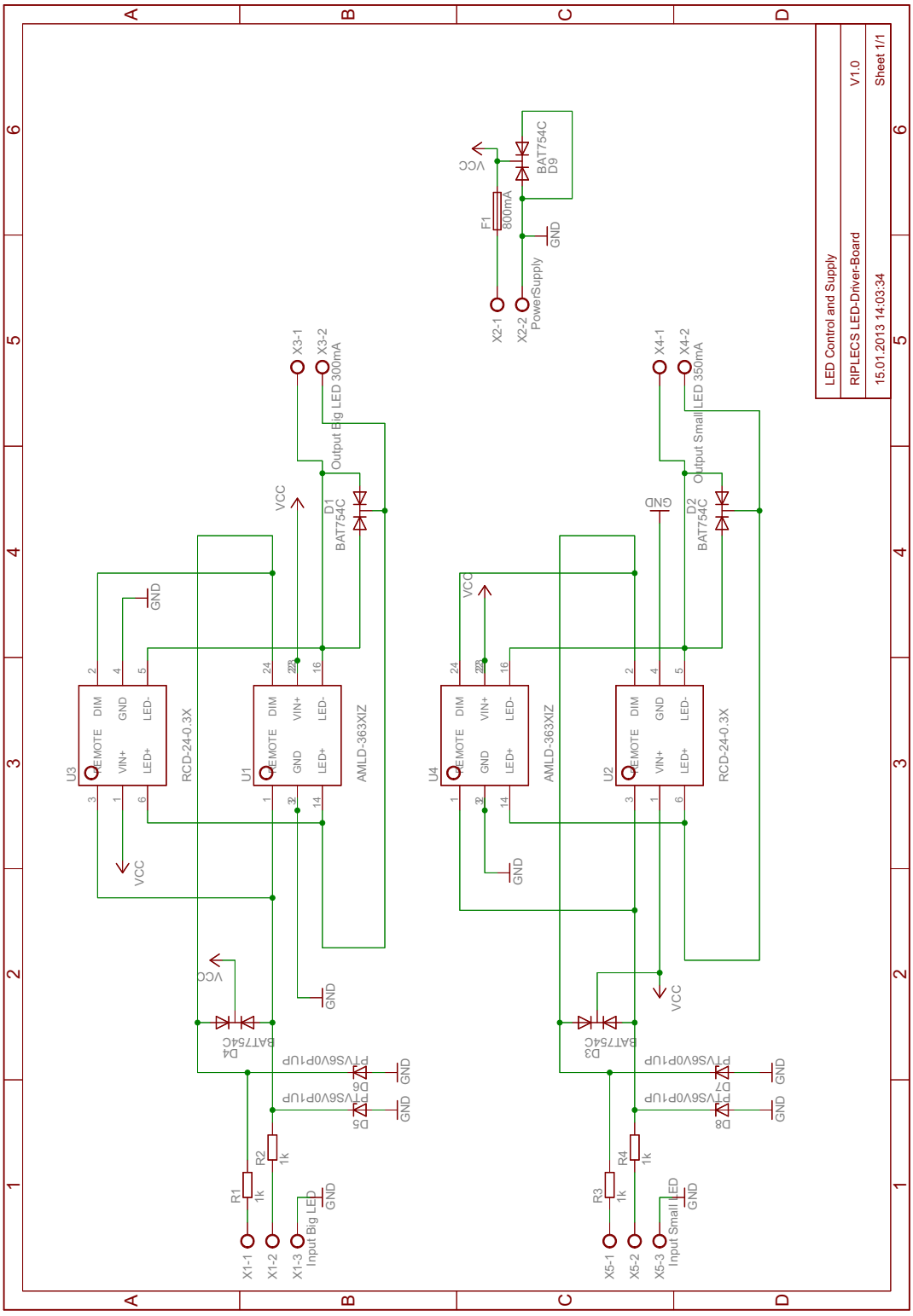
Der Schaltplan für die Versorgungsschaltung ist auf Seite 97 dargestellt. Er beinhaltet die Strommessungen über Shunt-Widerstände und Messverstärkern, die schaltbaren Versorgungsleitungen mit Smart-Switches sowie den Überspannungs- und Verpolungsschutz der Eingänge.

Der Schaltplan für die LED-Treiber ist auf Seite 98 dargestellt und umfasst die unterschiedlichen Treiber-Module sowie deren Schutzbeschaltung gegen Überspannung und Verpolung.


Current-metering and Supply-control
RIPLECS EHS-Board
V1.0
16.08.12 19:48
Sheet 1/2
TPS78033022ODDCT
TPS63031
ADG802
ADG801
BAT754C
PMT21EN
VLCF5020
ENABLE_CAP_1
ENABLE_CAP_2
ENABLE_SOLARCELL
MOTE_SUPPLY
ACTIVE_VREG
VCAP1
VCAP2
VCAP_REF
VSOLAR
XPAND1
XPAND2
GND


Top- and Bottom-socket connections
RIPLECS EHS-Board
V1.0
16.08.12 19:48
Sheet 2/2
SOCKET
DF9-51S
PLUG
DF9-51S-SOCKET
MOTE_SUPPLY
GND
ENABLE_SOLARCELL
ENABLE_CAP_1
ENABLE_CAP_2
ACTIVE_VREG
VCAP_REF
VSOLAR
VCAP2
VCAP1


Current-metering and Supply-control
RIPLECS Supply-Board
V1.0
14.01.13 17:24
Sheet 1/1
SupplyNode1
SupplyNode2
SupplyNode3
SupplyNode4
SenseNode1-2
SenseNode3-4
EnableNode1-2
EnableNode3-4
EnableNode GND
SenseNode GND
Supply
FPF2100
INA21X
BAT754C
TVS3V3P1UP
0R47
10k
10u
100n
VCC
GND


LED Control and Supply
RIPLECS LED-Driver-Board
V1.0
15.01.2013 14:03:34
Sheet 1/1
X2-1
X2-2
PowerSupply
F1
800mA
BAT754C
D9
VCC
GND
X3-1
X3-2
Output Big LED 300mA
D1
BAT754C
U3
U1
RCD-24-0.3X
AMLD-363XIZ
REMOTE
DIM
GND
LED-
VIN+
LED+
X4-1
X4-2
Output Small LED 350mA
D2
U4
U2
D3
D4
D5
D6
D7
D8
PTVS6V0P1UP
R1
R2
R3
R4
1k
X1-1
X1-2
X1-3
Input Big LED
X5-1
X5-2
X5-3
Input Small LED

Literaturverzeichnis

[1] L. B. Hörmann, "Design and implementation of a wireless sensor platform for river monitoring based on energy harvesting," Master's thesis, Institute for Technical Informatics, Graz University of Technology, März 2010.

[2] RIPLECS Project Consortium, "Riplecs project - remote-labs access in internet-based performance-centred learning environment for curriculum support," http://riplecs.dipseil.net/, 2013.

[3] E. A. . C. E. Agency, "Education, audiovisual and culture executive agency - lifelong learning programme," http://eacea.ec.europa.eu/llp/, 2012.

[4] Center for Educational Computing Initiatives, Massachusetts Institute of Technology, "ilab project at mit," http://ilab.mit.edu/wiki, 2013.

[5] P. Levis, N. Lee, M. Welsh, and D. Culler, "Tossim: accurate and scalable simulation of entire tinyos applications," in *Proceedings of the 1st international conference on Embedded networked sensor systems*, ser. SenSys '03. New York, NY, USA: ACM, 2003, pp. 126–137. [Online]. Available: http://doi.acm.org/10.1145/958491.958506

[6] V. Shnayder, M. Hempstead, B.-r. Chen, G. W. Allen, and M. Welsh, "Simulating the power consumption of large-scale sensor network applications," in *Proceedings of the 2nd international conference on Embedded networked sensor systems*, ser. SenSys '04. New York, NY, USA: ACM, 2004, pp. 188–200. [Online]. Available: http://doi.acm.org/10.1145/1031495.1031518

[7] G. Werner-Allen, P. Swieskowski, and M. Welsh, "Motelab: a wireless sensor network testbed," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, april 2005, pp. 483 – 488.

[8] J.-P. Sheu, C.-J. Chang, C.-Y. Sun, and W.-K. Hu, "Wsntb: A testbed for heterogeneous wireless sensor networks," in *Ubi-Media Computing, 2008 First IEEE International Conference on*, 31 2008-aug. 1 2008, pp. 338 –343.

[9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler, "The nesc language: A holistic approach to networked embedded systems," in *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation*, ser. PLDI '03. New York, NY, USA: ACM, 2003, pp. 1–11. [Online]. Available: http://doi.acm.org/10.1145/781131.781133

[10] J.-P. Sheu, B.-K. Hsu, P.-C. Lin, and C.-J. Chang, "Design and implementation of a lightweight operating system for wireless sensor networks," in *Proceeding of International Computer Symposium, Taipei, Taiwan*, December 2009.

[11] D. Jayasingha, N. Jayawardhane, P. Karunanayake, G. Karunarathne, and D. Dias, "Wireless sensor network testbed for mobile data communication," in *Information and Automation for Sustainability, 2008. ICIAFS 2008. 4th International Conference on*, dec. 2008, pp. 97 –103.

[12] U. Bilstrup, K. Sjoberg, B. Svensson, and P.-A. Wiberg, "Capacity limitations in wireless sensor networks," in *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, vol. 1, sept. 2003, pp. 529 – 536 vol.1.

[13] N. Li and J. Hou, "Topology control in heterogeneous wireless networks: problems and solutions," in *INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies*, vol. 1, march 2004, pp. 4 vol. (xxxv+2866).

[14] G. Niezen, G. Hancke, I. Rudas, and L. Horvath, "Comparing wireless sensor network routing protocols," in *AFRICON 2007*, sept. 2007, pp. 1 –7.

[15] T. J. Kazmierski and S. Beeby, *Energy Harvesting Systems - Principles, Modeling and Applications*. Springer, 2011.

[16] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava, "Design considerations for solar energy harvesting wireless embedded systems," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, april 2005, pp. 457 – 462.

[17] X. Jiang, J. Polastre, and D. Culler, "Perpetual environmentally powered sensor networks," in *Information Processing in Sensor Networks, 2005. IPSN 2005. Fourth International Symposium on*, april 2005, pp. 463 – 468.

[18] L. B. Hörmann, P. M. Glatz, K. B. Hein, M. Steinberger, C. Steger, and R. Weiss, "Towards an on-site characterization of energy harvesting devices for wireless sensor networks," in *Pervasive Computing and Communications Workshops (PERCOM Workshops), 2012 IEEE International Conference on*, march 2012, pp. 415 –418.

[19] L. B. Hörmann, M. Steinberger, M. Kalcher, and C. Kreiner, "Educational remote lab concept for energy harvesting enhanced wireless sensor networks," in *IEEE EDUCON - Engineering Education 2013*, 2012.

[20] L. B. Hörmann and C. Kreiner, "Needs Analysis Report - ICT Computer Architectures," Institute for Technical Informatics, Graz University of Technology, Jänner 2012.

[21] S. Priya and D. J. Inman, *Energy Harvesting Technologies*. Springer, 2009.

[22] M. Kalcher, *Remote Lab Web Access for Energy Harvesting Enhanced Wireless Sensor Networks*, Institute for Technical Informatics, Graz University of Technology, Oktober 2012.

[23] *NI USB-621x Specifications*, National Instruments Corporation, April 2009.

[24] *NI USB-621x User Manual*, National Instruments Corporation, April 2009.

[25] *Professional Kit*, 6020th ed., Crossbow Technology, Inc., San Jose, California 95134-2109, www.xbow.com.

[26] *Moteworks - Software Platform*, 6030th ed., Crossbow Technology, Inc., San Jose, California 95134-2109, www.xbow.com.

[27] TinyOSCommunityForum, "Tinyos, an open source operating system," www.tinyos.net, November 2012.

[28] *Datenblatt COB LED quadratisch 10W*, Rev. 1.4 ed., Barthelme, Josef Barthelme GmbH & Co. KG Oedenberger Strasse 149 D-90491 Nuernberg, www.barthelme.de, November 2012.

[29] *Green-Cap Electric Double Layer Capacitors*, SAMWHA ELECTRONIC CO., LTD.

[30] *ELECTRIC DOUBLE LAYER CAPACITORS EVerCAP*, Cat.8100y ed., nichicon.

[31] *Datasheet MICAz*, 6020th ed., Crossbow Technology, Inc., 4145 North First Street San Jose, California 95134-2109, www.xbow.com.

[32] *Data Sheet HP Elite Autofocus Webcam*, 1215th ed., Hewlett-Packard Development Company, L.P, www.hp.com, 2010.

[33] *Datenblatt AXIS M10 Netzwerk-Kamera-Serie*, 40706th ed., Axis Communications, www.axis.com, 2010.

[34] *Universal Serial Bus Device Class Definition for Video Devices*, Rev. 1.5 ed., USB Implementers Forum, Inc., www.usb.org, August 2012.

[35] *MTS/MDA Sensor Board Users Manual*, Rev. a ed., Crossbow Technology, Inc., 4145 North First Street San Jose, California 95134-2109, www.xbow.com, June 2007.

[36] *ATmega128L*, Rev. 2467x–avr–06/11 ed., Atmel Corporation, 2325 Orchard Parkway San Jose, CA 95131 USA.

[37] *Data Sheet ADG801/ADG802*, Rev. b ed., Analog Devices, Inc., One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., 2012.

[38] *Data Sheet PMT21EN*, Rev. 1 ed., NXP Semiconductors, Mikron-Weg 1, 8101 Gratkorn, Ausgust 2011.

[39] *Data Sheet TPS63031 - high ecient single inductor buck-boost converter with 1.8-a switches.*, Slvs696b ed., Texas Instruments Incorporated, 12500 TI Boulevard 12500 TI Boulevard Dallas, Texas 75243 USA, March 2012.

[40] *Data Sheet TPS780 Series - 150mA, Low-Dropout Regulator, Ultralow-Power, IQ500nA withPin-Selectable, Dual-Level Output Voltage*, Sbvs083d ed., Texas Instruments Incorporated, 12500 TI Boulevard 12500 TI Boulevard Dallas, Texas 75243 USA, September 2012.

[41] *Data Sheet FPF2100-FPF2107 IntelliMAXTM Advanced Load Management Products*, Rev. h ed., Fairchild Semiconductor Corporation, www.fairchildsemi.com, August 2008.

[42] *Data Sheet Voltage Output, High or Low Side Measurement, Bi-Directional Zerø-Drift Series CURRENT-SHUNT MONITOR*, Sbos437d ed., Texas Instruments Incorporated, 12500 TI Boulevard 12500 TI Boulevard Dallas, Texas 75243 USA, November 2012.

[43] *Data Sheet RCD-24*, Rev. 1 ed., Recom, www.recom-international.com, 2012.

[44] *Data Sheet Series AMLD-IZ*, F 051e r12.o ed., Aimtec, www.aimtec.com.

[45] The IEEE and The Open Group, "The open group base specifications issue 6," http://pubs.opengroup.org/onlinepubs/009695399/functions/select.html, 2004.

[46] N. Matthew and R. Stones, *Beginning Linux Programming*, 4th ed. Wiley Publishing, Inc., 2008, ch. Sockets, pp. 635–638.

[47] The Open Group, "The single unix ® specification, version 2," http://pubs.opengroup.org/onlinepubs/7908799/xns/syssocket.h.html, 1997.

[48] Oracle, "Mysql - the world's most popular open source database," http://dev.mysql.com/doc/refman/5.0/en/c.html, 2012.

[49] L. B. Hörmann, *Task Specification - ICT Computer Architectures Wireless Sensor Networks*, ws2012/13 ed., Institute for Technical Informatics, Graz University of Technology, 2013.