DIPLOMARBEIT

„Implementierung eines selbst organisierenden drahtlosen Sensornetzwerkes“

Ausgeführt zum Zweck der Erlangung des akademischen Grades eines
Dipl.-Ing. (FH) für Computersimulation
am Fachhochschuldiplomstudiengang Computersimulation St. Pölten

Unter der Leitung von

Dipl.-Ing. Dr. Martin Brandl
Donau-Universität Krems
Zentrum für Biomedizinische Technologie

und

Dipl.-Ing. Dr. Christian Fabian
Fachhochschule St. Pölten
Communications Engineering

ausgeführt von

Christoph Mayerhofer
si0310095008

Eidesstattliche Erklärung

Ich, Christoph Mayerhofer, geboren am 03. Mai 1981 in Wien, erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Christoph Mayerhofer
28. August 2007, St. Pölten

Vorwort

Die vorliegende Diplomarbeit entstand im Rahmen meines Fachhochschulstudiums Computersimulation an der Fachhochschule St. Pölten in Zusammenarbeit mit der Elektronik-Abteilung der Donau-Universität Krems. Ich möchte mich bei der Elektronik-Abteilung der Donau-Universität für den netten Arbeitsplatz sowie das zur Verfügung Stellen aller notwendigen Geräte bedanken. Besonders bedanken möchte ich mich auch für die technischen Ratschläge meiner beiden Arbeitsplatzkollegen Ing. Thomas Posnicek und Ing. Karlheinz Kellner. Weiters gilt mein Dank meinen beiden Betreuern, Dr. Martin Brandl (Donau-Universität Krems) und Dipl. Ing. Dr. Christian Fabian (Fachhochschule St. Pölten), welche mir stets für Fragen zur Verfügung standen.

Kurzfassung

Drahtlose Sensornetzwerke (DSNe) bestehen aus einer Vielzahl von kleinen Sensorknoten, welche in der Lage sind eine gewisse Größe (z.B. Temperatur) zu messen und diese in Form eines digitalen Funksignals an ihre Nachbarn zu versenden. DSNe sind sowohl im wissenschaftlichen, medizinischen als auch im militärischen Bereich einsetzbar und ihre Anwendungsmöglichkeiten daher beinahe unbegrenzt. Diese Diplomarbeit beschäftigt sich mit der Implementierung eines DSNs. Hierzu gehören sowohl das Programmieren einer Betriebsoftware als auch die Entwicklung der Hardware. Zu den beiden Hauptaufgaben der Software zählen das Initialisieren der Hardware und das Routing von Datenpaketen. Der hierfür benutzte Routingalgorithmus wurde von meinem Studienkollegen Andreas Kos entwickelt und auf seine Funktionalität mit Hilfe von Simulationen getestet. Im ersten Hauptteil meiner Arbeit wird sowohl auf den Routingalgorithmus als auch auf dessen Implementierung eingegangen. Die theoretischen Grundlagen werden mit Grafiken (Ablaufdiagramme, etc.) und Codebeispielen verdeutlicht. Der zweite Hauptteil der Arbeit beschäftigt sich mit der Hardware der Sensorknoten. Hier werden die einzelnen Bauteile, deren Funktion sowie die Kommunikation untereinander beschrieben. Des Weiteren beinhaltet dieser Abschnitt Abbildungen von dem Schaltplan sowie dem Layout der Sensorknoten. Abschließend wird ein kurzer Überblick über die erreichten Ziele, sowie ein Ausblick auf weitere Problemstellungen gegeben.

Abstrakt

Ziel dieser Diplomarbeit war die Implementierung eines selbst organisierenden drahtlosen Sensornetzwerkes. Hierzu gehören die Entwicklung der Hardware und der Betriebssoftware für die Sensorknoten. Die Software initialisiert die Hardware der Sensorknoten und ist für das Routing der Datenpakete zuständig. Der hierfür verwendete Routingalgorithmus basiert auf dem Verhalten von Ameisen bei der Futtersuche und hat sich in Simulationen als äußerst robust erwiesen. Strukturänderungen im Netzwerk werden rasch erkannt und führen zu selbsttätigen Anpassungsreaktionen. Des Weiteren steuert die Software das Energiemanagement der Sensorknoten, da sich diese selbst in einen stromsparenden Modus versetzen, aus welchem alle Knoten im Netzwerk zeitgleich erwachen müssen. Wird ein Sensorknoten nachträglich in das Netz integriert, so synchronisiert er sich selbständig mit seinen Nachbarn. Die für das Sensornetzwerk entwickelten Knoten sind 14,08 cm^2 groß. Ihre Höhe beträgt 4,5 cm. Sie werden mit einer 3 V Batterie versorgt. Während der Routingaktivitäten beträgt die Stromaufnahme der Knoten 19,9 mA. Im stromsparenden Modus wird diese deutlich auf ca. 1 μA abgesenkt. Beim Design der Sensorknoten wurde darauf geachtet, dass ein nachträgliches Anbringen von diversen Sensoren einfach zu realisieren ist. Sowohl die Hardware, als auch das Routingprotokoll erwiesen sich in einem Versuch mit 4 Sensorknoten als voll funktionsfähig.

Abstract

The goal of this thesis was the implementation of one self-organised Wireless Sensor Network (WSN). This contains the development of both the sensor node's hardware and the operating system. The software initialises the sensor nodes' hardware and is responsible for the datagrams' routing. The algorithm used for that is based on the behaviour of ants during their forage and has proved to be very robust in simulations. Structure changes in networks are recognised rapidly and cause automatic adjustment reactions. Furthermore the software is controlling the sensor nodes' energy management as they have to be able to set themselves in power saving mode out of which all nodes in the network have to wake up concurrently. If one sensor node is integrated into the network later, it is synchronising with it's neighbours autonomously. The nodes developed for the sensor network are 14.08 cm^2 big. Their height is 4.5 cm. They are powered by a 3 Volt battery. During the routing activities the nodes' power input is 19.9 mA. In power saving mode it can be reduced significantly down to 1 μA. The simple subsequent fixing of new sensors was one more criteria that was considered during the sensor nodes' designing process. In an experiment with 4 sensor nodes the hardware as well as the routing protocol proved to be entirely functionally.

Inhaltsverzeichnis

1 Einleitung

1.1 Einführung

Drahtlose Sensornetzwerke (DSNe) bestehen aus einer Vielzahl von kleinen Sensorknoten, welche in der Lage sind gewisse Größen (z.B. Temperatur, Luftdruck, ...) zu messen und diese in Form eines digitalen Funksignals an ihre Nachbarn zu versenden. Ziel eines solchen Netzwerkes ist es zahlreiche Sensoren auf einer geographisch großen Fläche zu platzieren und all deren Messwerte bequem von einem einzigen Standort aus lesen zu können. Des Weiteren sollten die einzelnen Elemente billig produzierbar und über einen sehr langen Zeitraum funktionsfähig sein. Dies setzt voraus, dass die Knoten äußerst energiesparend betrieben werden können.
Die einfachste Art Daten an eine Zentrale zu übermitteln ist, wenn jeder Sensor die von ihm aufgenommenen Größen direkt zu einer Basisstation sendet. Dazu muss die Übertragung jedoch über eine große Entfernung erfolgen, was wiederum einen sehr hohen Stromverbrauch und damit auch eine kurze Lebensdauer des Knotens zur Folge hat. Ziel ist es daher die Funkstrecken so kurz wie möglich zu halten. Hierzu werden die Datenpakete in einem DSN von einem Knoten zu seinem Nachbarknoten weitergereicht, bis sie letztendlich bei der Datensenke eintreffen. Das bedeutet, dass die Messwerte eine große Anzahl an Elementen passieren können, bevor sie am Ziel eintreffen. Da jeder Knoten über mehrere Nachbarn verfügen kann, ist es notwendig, dass der am besten

geeignete ausgewählt wird. Daher müssen die einzelnen Knoten in der Lage sein, intelligente Entscheidungen zu treffen um einen günstigen Weg durch das Netzwerk zu finden (Routing).

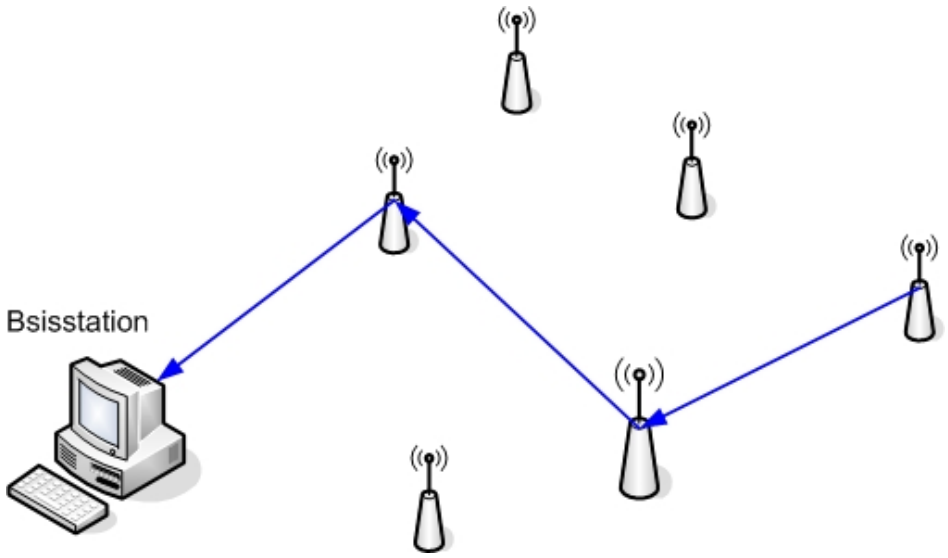


Abbildung 1.1: Routing über Nachbarknoten

1.2 Anwendungen eines Sensornetzes

Ein Beispiel für den Einsatz von Sensornetzwerken wäre zum Beispiel die Waldbrandbekämpfung. Hierzu werden in einem gefährdeten Gebiet mehrere Sensorknoten verteilt. Jedes Modul misst in gewissen Zeitabständen die Temperatur. Bei Hitzeentwicklung sendet der Sensorknoten eine Anfrage an seine Nachbarn. Werden hier ebenfalls erhöhte Temperaturen gemessen, wird ein Warnsignal, welches die Position des Brandes beinhaltet, zu einer Basisstation (Feuerwehr) übermittelt. Die Anfrage an die Nachbarn ist wichtig, da ein einzelner Sensor durch einen Defekt oder durch andere Einflüsse (z.B.: Lagerfeuer) abgelenkt werden könnte. Des Weiteren kann dadurch Information über das Ausmaß des Brandes gegeben werden.
Weitere Möglichkeiten wären:

- Warenverwaltung in Lagerhäusern (z.B.: Überprüfung von Kühlaggregaten)
- Überwachung von Naturgebieten (Auftreten von Schadstoffen bzw. Chemikalien)
- Überwachung von Kernkraftwerken (Messen der Strahlung in der Umgebung)

- Überwachung von seismischen Aktivitäten auf Brücken oder in Gebäuden

1.3 Ausgangssituation

Es gibt bereits zahlreiche Routingprotokolle für Sensornetzwerke [8]. Das Problem ist nur dass die meisten relativ komplex sind und daher eine Menge Ressourcen (Speicher, Rechenleistung) benötigen. Eine starke Auslastung des Prozessors erfordert jedoch viel Energie. Manche Protokolle legen Routingtabellen über das gesamte Netzwerk in ihrem Speicher ab. Daher müssen die Knoten mit einem relativ großen Datenspeicher bestückt werden. Dies hat einen erheblichen Anstieg der Kosten zur Folge.
Diese Diplomarbeit beschäftigt sich sowohl mit den Hardware- als auch mit den Softwarekomponenten eines drahtlosen Sensornetzwerkes. Ziel ist die Entwicklung von Sensorknoten sowie deren Betriebssoftware, um das Aufnehmen und Routing von Messdaten zu ermöglichen.
Der Routing-Algorithmus ist in der Programmiersprache C programmiert und basiert auf den theoretischen Grundlagen der Diplomarbeit „Management Of Selforganising Wireless Sensor Networks“ von Andreas Kos. [1]

1.4 Motivation

Der Grundgedanke des Routing-Algorithmus basiert auf dem Koordinationsverhalten von Ameisen bei der Futtersuche. Es handelt sich hierbei um eine äußerst einfache und doch effiziente Methode den richtigen Pfad durch ein Netzwerk zu finden. Dadurch ergeben sich nur geringe Anforderungen an Speicher und Prozessor und somit auch an die Batterie. Des Weiteren entsteht ein hoch dynamisches Netzwerk, welches bei Ausfall oder Hinzufügen eines neuen Teilnehmers in der Lage ist sich rasch und eigenständig neu zu organisieren. Vorteile der Routingstrategie:

- Einfacher Algorithmus
- Geringe Anforderungen an die Hardware
- Lange Lebensdauer der Sensorknoten
- Rasche Konvergenz des Netzes
- Schleifenfreies Routing

1.5 Funktionsweise des Routing-Algorithmus

Sobald sich der Speicher (Pufferfüllstand) eines Sensorknotens über den Schwellwert von 70% gefüllt hat, versucht dieser die gesammelten Daten zur Basis zu senden. Der Knoten beginnt mit der Suche nach einem geeigneten Nachbarn zu welchem die Daten übermittelt werden können. Zu Beginn wird ein Broadcast an alle Nachbarn gesendet. Diese antworten mit ihrer ID Nummer, sowie mit ihrem aktuellen Energie- und Pufferfüllstand (beide Werte zwischen 0 und 1). Die empfangene Feldstärke dieser Antworten wird von dem nach einem Kommunikationspartner suchenden Knoten abgespeichert. Nun wird aus den empfangenen Daten (Energiezustand, Pufferfüllstand, Signalstärke) ein Vergleichswert errechnet, welcher dazu dient den am best geeigneten Nachbarn zu eruieren. Dieser Wert wird Pheromon genannt. Die Basis, welche die Daten des gesamten Netzwerks sammelt, hat stets den höchsten vorkommenden Pheromonwert von 1. Daten dürfen nur an Nachbarknoten gesendet werden, wenn dieser einen höheren Pheromongehalt als der sendende Knoten besitzen. Andernfalls würden die Daten in die falsche Richtung laufen. Sollte kein geeigneter Nachbarknoten gefunden werden, so wird der eigene Pheromonwert schrittweise abgesenkt. Dies geschieht solange bis der Wert niedriger ist als der des Nachbars. Sollte jedoch überhaupt kein Nachbarknoten gefunden werden, so versetzt sich der anfragende Knoten in einen Stromsparmodus und versucht es nach einiger Zeit wieder. Dieser Algorithmus hat sich in Simulationen bereits als sehr robust herausgestellt. Ziel dieser Diplomarbeit ist es nun ihn auch in einem realen Prototyp zu implementieren. In Abbildung 1.2 ist das Ergebnis einer simulierten Pheromonverteilung in einem drahtlosen Sensornetzwerk ersichtlich. [1]

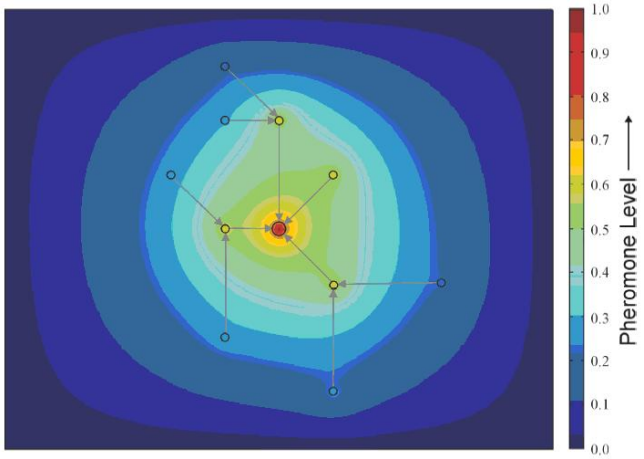


Abbildung 1.2: Pheromonverteilung in einem DSN [1]

2
Routing Protokolle für DSNe

2.1 Einleitung

Routing bezeichnet in der Telekommunikation das Festlegen von Wegen für Nachrichtenströme bei der Nachrichtenübermittlung über vermaschte Nachrichtennetze bzw. Rechnernetze. Insbesondere in paketvermittelten Datennetzen ist hierbei streng genommen zwischen den beiden verschiedenen Prozessen Routing und Forwarding zu unterscheiden: Das Routing bestimmt den gesamten Weg eines Nachrichtenstroms durch das Netzwerk; das Forwarding beschreibt hingegen den Entscheidungsprozess eines einzelnen Netzknotens, über welchen seiner Nachbarn er eine vorliegende Nachricht weiterleiten soll. Häufig werden jedoch Routing und Forwarding unter dem Begriff „Routing“ miteinander vermengt; in diesem Fall bezeichnet Routing ganz allgemein die Übermittlung von Nachrichten über vermaschte Nachrichtennetze. [18]

2.2 Netzwerkprotokolle

Die ersten Untersuchungen zu Sensornetzwerken begannen circa 1980. Die damaligen Forschungen waren auf rein militärischer Basis, wurden jedoch schon bald auf den

zivilen Bereich ausgeweitet. Bedingt durch die immer größer werdende Packungsdichte der elektronischen Bauelemente, erlebte die Forschung in den 90iger Jahren einen großen Aufschwung. In den vergangenen Jahren wurden bereits einige Routingstrategien für Sensornetzwerke entwickelt. Ein Großteil orientiert sich jedoch an Algorithmen, die für das Internet geschaffen wurden und daher nicht auf die speziellen Bedürfnisse von DSNen eingehen. Das Hauptproblem hierbei ist der hohe Stromverbrauch, bedingt durch das Senden und Empfangen von zahlreichen Steuerdaten (keine Nutzdaten), welche zur Organisation des Netzwerkes dienen. Im folgenden Abschnitt wird die Funktionsweise von zwei gängigen Routingprotokollen kurz vorgestellt, wobei hier nicht auf Details eingegangen wird.

2.2.1 Destination Sequenced Distance Vector (DSDV)

Wie aus dem Namen bereits hervorgeht handelt es sich um ein „Distance Vector“ Routing Protokoll. Grundidee ist, dass jeder Knoten den Aufbau des gesamten Netzes kennt. Um dies zu erreichen, eruiert jeder Teilnehmer seine direkten Nachbarn und ordnet ihnen, je nach dem wie günstig sie zu erreichen sind, einen Wert (Kosten) zu. Im nächsten Schritt wird die eben erstellte Tabelle an alle Nachbarn ausgeschickt. Die erhaltenen Tabellen werden nun mit der eigenen verglichen. Ergeben sich bessere Pfade zum Ziel, wird die eigene Tabelle aktualisiert und erneut ausgeschickt.

Vorteile:

- Schleifenfreies Routing
- Nach der oben beschriebenen Organisationsphase werden kaum noch Steuerdaten gesendet

Nachteile:

- Während der Organisationsphase wird das Netz mit einer sehr großen Menge an Steuerdaten konfrontiert. Dies hat eine starke Auslastung des Prozessors und des Transceivers zur Folge und erhöht dadurch die Stromaufnahme enorm
- Hoher Speicherbedarf, da alle Routen in Tabelle gespeichert werden müssen
- Die Organisationsphase muss bei jedem Ausfall oder Hinzufügen eines Sensorknotens wiederholt werden

2.2.2 Dynamic Source Routing (DSR)

Im Gegensatz zum DSDV findet hier keine Organisationsphase statt. Die einzelnen Knoten müssen nicht zwingend alle Routen kennen, sondern sie werden erst bei Bedarf gesucht (On Demand). Möchte ein Knoten Daten senden, sieht er zuerst in seinem Routecache nach, ob ein Pfad vorhanden ist. Wurde keiner gefunden, beginnt der Knoten das gesamte Netz mit einer Anfrage zu fluten. Jeder Teilnehmer, der das Paket weitersendet, fügt seine Adresse an den Header an. Erreicht die Anfrage schlussendlich sein Ziel, wird eine Bestätigung zurückgesandt, welche alle Adressen der passierten Teilnehmer enthält. Die so erhaltene Route wird nun abgespeichert und die Daten können versendet werden. Des Weiteren können die im Paket enthaltenen Adressdaten auch von anderen Netzteilnehmern benutzt werden um Routen zu bilden.

Vorteile:

- Geringerer Speicherbedarf, da nur benötigte Routen abgespeichert werden
- Keine komplexen Berechnungen zur Wahl der Route nötig, daher nur geringe Prozessorauslastung
- Weniger Organisationspakete, da die Knoten schon bei der Datenübertragung die Netzstruktur kennen lernen

Nachteile:

- Fluten des Netzes bei Pfadanfragen, belastet alle Teilnehmer
- Relativ große Anfrage- und Antwortpakete, da alle Adressinformationen gesendet werden müssen

2.3 Ameisenalgorithmus

Die Suche nach einer guten Route kann, wie bereits aus der Beschreibung von DSDV und DSR ersichtlich, zu einer komplizierten und aufwendigen Arbeit werden. Beobachtet man nun Ameisen auf Futtersuche, welche Nahrung zu ihrem Bau tragen, bemerkt man dass diese stets eine sehr guten Wegwahl treffen, obwohl eine einzelne Ameise nur über wenig Intelligenz verfügt und kaum mit ihren Kollegen kommuniziert. Vergleicht man nun eine Ameisenkolonie mit einem Netzwerk von Sensorknoten, bemerkt man, dass sich hier ein gemeinsames Ziel sowie ähnliche Voraussetzungen ergeben:

- Ziel ist das Auffinden des am besten geeigneten Weges
- Geringe Intelligenz der einzelnen Elemente (Ameisen bzw. Sensorknoten). Dies ist besonders wichtig, da der Programmieraufwand und die benötigte Prozessorleistung sonst sehr hoch werden würden
- Hohe Intelligenz im Gruppenverhalten (Schwarmintelligenz)
- Geringes Kommunikationsaufkommen

2.3.1 Grundprinzip

Ameisen scheiden bei der Futtersuche Duftstoffe (Pheromone) aus, welche eine anziehende Wirkung auf ihre Kollegen ausüben. Am Anfang ihrer Suche laufen sie zufällig und ziellos in verschiedene Richtungen. Sobald eine Futterquelle gefunden wurde, nimmt die Ameise Nahrung auf und kehrt zurück zu ihrem Nest um sie dort abzuladen. Der Weg zur Futterquelle ist nun mit einer 2-fachen Pheromonspur gekennzeichnet (Hin- und Rückweg). Machen sich nun weitere Ameisen auf die Suche, wird jener Pfad mit dem höchsten Pheromongehalt gewählt. Der Pheromonwert steigt immer weiter und schon bald laufen alle Ameisen den Weg, der am schnellsten zu einer Futterquelle führt (Ameisenstraße).
Die folgenden Abbildungen sollen anhand eines einfachen Beispiels helfen, dieses Prinzip zu verdeutlichen.

1. Zwei Ameisen starten vom selben Startpunkt (Nest) und laufen verschiedene Wege entlang um Nahrung zu suchen.

Abbildung 2.1: Beginn der Futtersuche

2. Beide Ameisen haben Futter gefunden und kehren nun auf demselben Weg, den sie gekommen sind, zu ihrem Nest zurück.

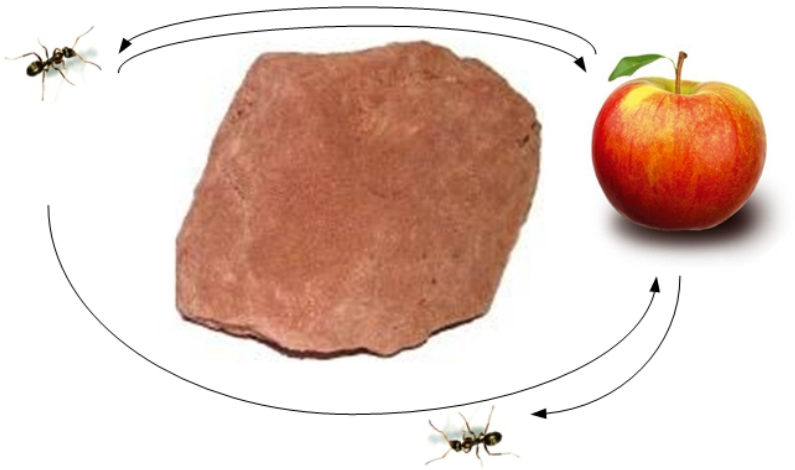


Abbildung 2.2: Doppelte Kennzeichnung des kürzeren Pfades

3. Da die obere Ameise zuerst am Nest ankommt, ist ihr Weg mit einer 2-fachen Pheromonspur gekennzeichnet, während der längere Pfad nur eine einfache Kennzeichnung aufweist. Daher wird diese Route von weiteren Futter suchenden Ameisen gewählt, wobei die Pheromonspur (durch Pfeile dargestellt) immer stärker wird.

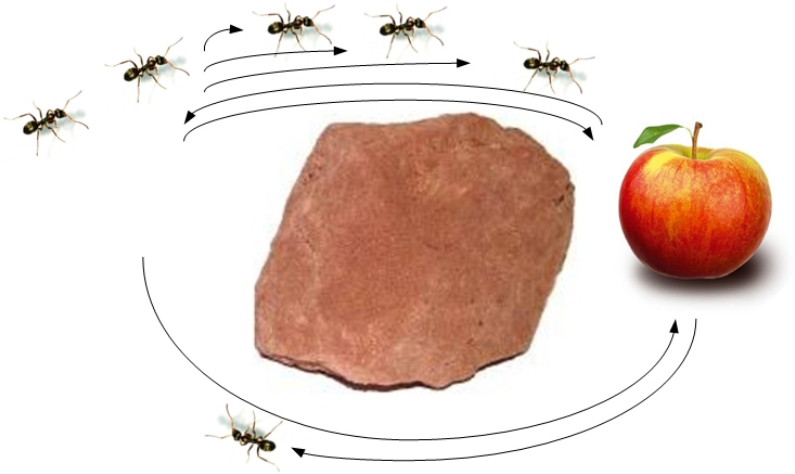


Abbildung 2.3: Pheromonkonzentration steigt weiter

3

Implementierung einer pheromonbasierten Routingstrategie in DSNen

Natürlich kann man das Verhalten von Ameisen nicht mit dem eines Sensornetzes gleichstellen, jedoch kann die Grundidee des Algorithmus für dessen Zwecke verwendet werden. Jeder Sensorknoten erhält einen „Pheromonwert“, welcher sich aus folgenden Komponenten errechnet:

- Anzahl der Knoten, die zwischen sich selbst und der Basisstation liegen
- Eigene Batteriekapazität
- Qualität der Funkverbindung zum Nachbarknoten (Empfangsfeldstärke)
- Belegter Speicher

Möchte ein Knoten nun seine gespeicherten Daten senden, schickt er eine Anfrage an seine unmittelbaren Nachbarn aus. Diese antworten mit ihrem „Pheromonwert“. Der anfragende Knoten wählt nun den Nachbarn mit der höchsten Zahl aus und sendet diesem seine Daten. Für die Berechnung des Pheromonwertes siehe Abschnitt 3.2.
Im folgenden Beispiel (Abbildung: 3.1) sind die verschiedenen Knoten mit K1 - K5

beschriftet. Jeder Knoten hat seinen Pheromonwert berechnet, welcher in den folgenden Abbildungen mit P abgekürzt wird. Die Funkreichweite jedes Knotens ist anhand des gestrichelten Kreises dargestellt.

1. Wir nehmen an, dass der Datenspeicher von Knoten K1 (Abbildung: 3.1) voll ist. Daher möchte er seine Daten zur Basisstation übermitteln. K1 sendet eine Anfrage (Broadcast) an alle benachbarten Knoten, welche sich noch innerhalb seiner Reichweite befinden.

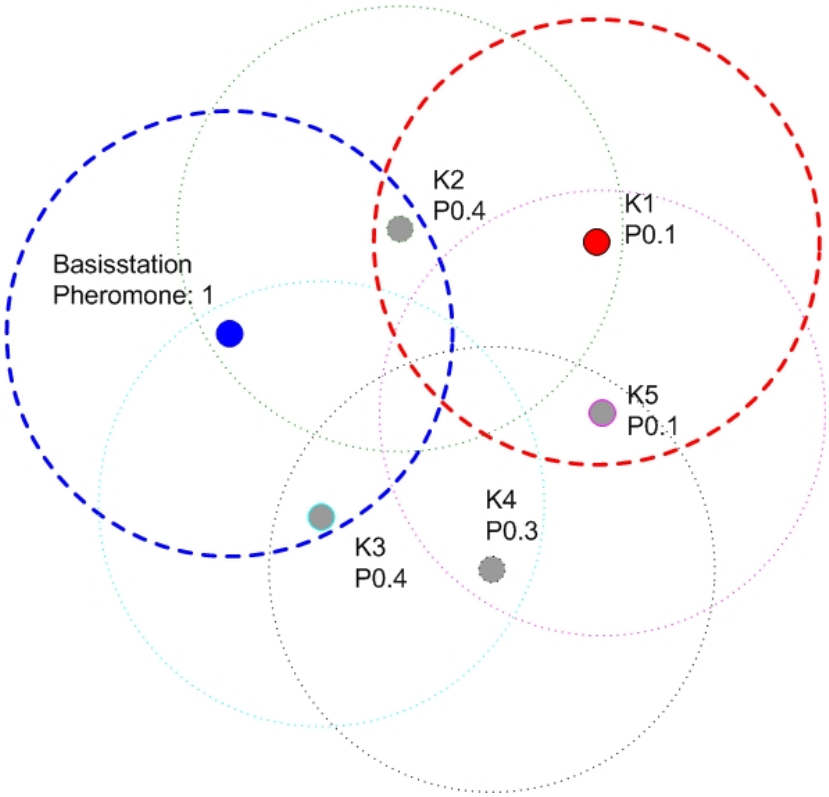


Abbildung 3.1: Netzwerk

2. Die angesprochenen Knoten senden ein Antwortpaket, welches den eigenen Pheromonwert und ihre Adresse beinhaltet (Abbildung: 3.2).

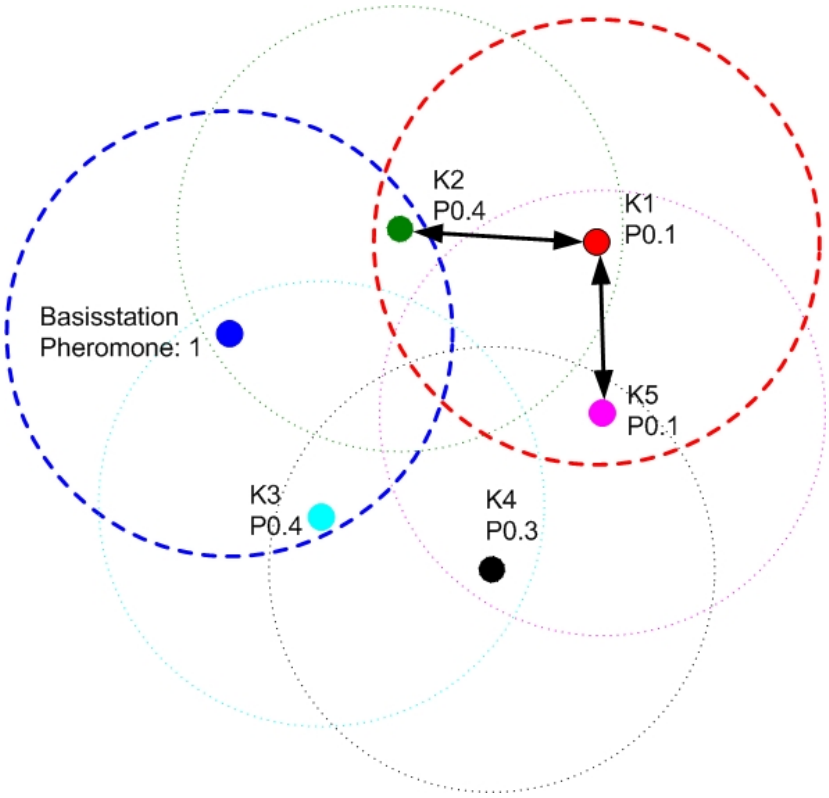


Abbildung 3.2: Anfrage und Antworten

3. Da innerhalb des Funkbereiches der Knoten K2 (Abbidlung: 3.3) den höchsten Pheromonwert (P = 0.4) aufweist, wird er als Empfänger für die Daten ausgewählt. K2 möchte die aufgenommenen Daten rasch wieder loswerden, um seinen eigenen Speicher nicht unnötig zu belasten. Er wiederholt daher den gleichen Ablauf wie K1. Die Basisstation besitzt immer den höchsten Pheromonewert (P = 1) im gesamten Netzwerk und erhält daher schlussendlich die Datenpakete aller Knoten.

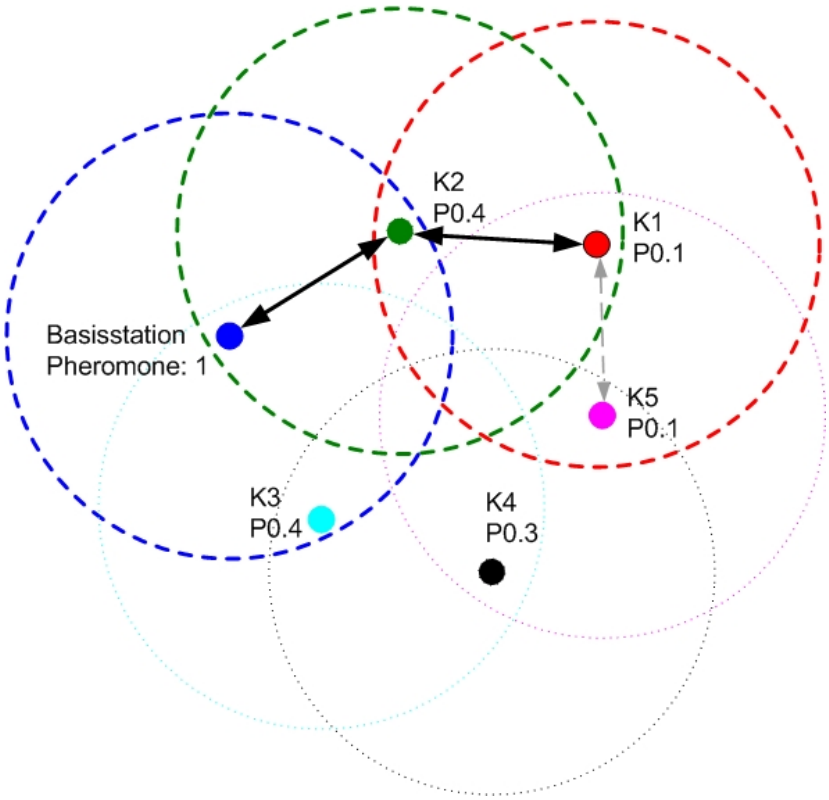


Abbildung 3.3: Auswahl getroffen; Senden der Daten

3.1 Betriebssoftware und Routingprotokoll

3.1.1 Aufgaben der Betriebssoftware

Initialisierung der Hardware

Im ersten Abschnitt der Software werden die verwendeten Bauteile initialisiert. Beim Mikrokontroller müssen beispielsweise die verwendeten Pins als Ein- oder Ausgänge definiert werden beziehungsweise für die Datenübertragung zu Peripheriebausteinen vorbereitet werden. Dies geschieht über das Setzen der entsprechenden Register, welche im zugehörigen Datenblatt ausführlich beschrieben sind [5].

Aufnahme von Messdaten

Es gibt eine Vielzahl an Sensoren am Markt, welche für die verschiedendsten Anwendungen (z.B. Temperaturmessung) konstruiert wurden. Abgesehen von ihren Zweck, können sie auch durch ihre Zugriffsart unterschieden werden:

- Analog
- Digital: SPI Schnittstelle, I-Quadrat-C Schnittstelle oder andere digitale Übertragungsprotokolle

Analoge Sensoren sind meist sehr einfach aufgebaut. In der Regel handelt es sich hierbei um ein Bauteil, dessen elektronische Eigenschaften von gewissen Größen beeinflusst werden.
Für Wärmemessungen könnte beispielsweise ein Keramik-Kondensator herangezogen werden, welcher über einen temperaturabhängigen NTC-Widerstand geladen wird. Der Mikrokontroller kann nun durch Messen der Entladezeit die Kapazität in einen digitalen Wert umwandeln. Im letzten Schritt wird das erhaltene Ergebnis anhand von Tabellen oder Kurven in einen Temperaturwert umgerechnet.
Die Regeln und Rechenvorschriften für die oben beschriebenen Abläufe erhält der Mikrokontroller aus der Betriebssoftware.
Bei digitalen Sensoren werden die Messergebnisse bitweise zum Mikrokontroller übertragen. Die Betriebsoftware definiert hier die richtige Zugriffsart (SPI, I-Quadrat-C) auf den Sensor und beinhaltet eventuell auch Regeln zur Datenübertragung (selbst entworfene Übertragungsprotokolle).

3.1.2 Datenpakete und deren Funktion

Für das in dieser Arbeit beschriebene Routingprotokoll werden 5 verschiedene Paketarten benutzt. Jede Paketart ist durch eine eindeutige Nummer (1 Byte) gekennzeichnet (siehe 3.5.5). Daher ist die Mindestlänge (inklusive Paketlänge) jedes Paketes 15 Byte (1Byte Paketlänge + 13 Byte Header + 1 Byte Paketart, siehe 3.5.1).

Hello-Paket

Hello-Pakete dienen ausschließlich der Synchronisation, welche im Abschnitt 3.4 genauer erläutert wird. Als Zusatzinformation enthalten sie den Pheromonwert (1 Byte) des Knotens und weisen daher eine Paketlänge von 16 (15 Byte Mindestlänge + 1 Byte Pheromon) auf.

Anfrage-Paket

Möchte ein Knoten seine Daten senden, so schickt er ein Anfrage-Paket an alle seine Nachbarn (Broadcast). Sobald ein Knoten dieses Paket empfängt, sendet er ein Antwort-Paket zurück. Anfrage-Pakete benötigen keine zusätzlichen Informationen und sind daher nur 15 Byte (Mindestlänge) groß.

Antwort-Paket

Wurde ein Anfrage-Paket eines Nachbarknoten empfangen, wird ein Antwort-Paket zurückgesandt, welches den Pheromonwert beinhaltet. Der anfragende Knoten kann mehrere Antwort-Pakete von den verschiedenen Nachbarn empfangen. Er entscheidet sich für jenen Knoten mit dem höchsten Pheromonwert und sendet diesem seine Daten. Der Aufbau des Antwort-Pakets stimmt mit dem des Hello-Pakets überein. Sie unterscheiden sich lediglich durch die Paketart (15tes Byte).

Nutzdaten-Paket

Hier werden die eigentlichen Nutzdaten (Messwerte) übermittelt. Die Größe kann je nach Anwendung variieren (15 Byte + gewünschte Datenmenge). Das Daten-Paket ist das einzige Paket, dessen Empfang bestätigt werden muss.

Empfangsbestätigung

Bestätigt den Empfang eines Datenpaketes. Die Besonderheit ist, dass hier die Paketnummer nicht fortlaufend ist, sondern einfach durch die Nummer des empfangenen Datenpaketes ersetzt wird, um nicht irrtümlicher Weise das falsche Paket zu bestätigen. Das Paket enthält keine Zusatzinformationen und ist daher 15 Byte lang.

3.2 Berechnung des Pheromonwerts

Jeder Knoten verfügt über einen Pheromonwert (P), welcher seinen Nachbarn bei der Auswahl des besten Weges helfen soll. Bei Empfang eines Anfrage-Paketes wird dieser Wert noch mit der Ladung (Q) und mit dem Füllstand des Speichers (S) gewichtet. Zusätzlich werden die Größen noch mit einem Gewichtungsfaktor (GF) multipliziert, welcher es ermöglicht, die Auswirkung der einzelnen Faktoren auf das Gesamtergebnis zu regeln.

$$P_{\text{gewichtet}} = P * (Q * Q_{\text{GF}}) * (S * S_{\text{GF}}) \tag{3.1}$$

Anschließend wird der soeben errechnete Wert ($P_{\text{gewichtet}}$) an den anfragenden Knoten zurückgeschickt, welcher diesen noch mit der empfangenen Feldstärke (F) gewichtet.

$$P_{\text{neu}} = P_{\text{gewichtet}} * (F * F_{\text{GF}}) \tag{3.2}$$

Jener Nachbar, dessen P_{neu} am höchsten ist, wird als Empfänger der Daten ausgewählt. Danach wird der eigene Pheromonwert (P) noch mal neu berechnet. Hierzu wird einfach

das höchste empfangene $P_{\text{gewichtet}}$ mit einem Reduktionsfaktor (R) multipliziert.

$$P = P_{\text{gewichtet}} * R \tag{3.3}$$

Da die Basisstation stets den höchstmöglichen Pheromonwert in einem Netz aussendet, haben Knoten, welche in ihrer Nähe platziert sind stets einen relativ hohen Wert. Je weiter ein Knoten von der Basis entfernt ist, desto geringer wird auch sein Pheromonwert. Eine detaillierte Beschreibung der Pheromonberechnung sowie der Routingstrategie ist der Diplomarbeit [1] von Andreas Kos zu entnehmen.

3.3 Programmablauf

In Abbildung 3.4 wird die main-Methode des Programms dargestellt. Zu allererst werden hier Funktionen aufgerufen, welche die Hardware des Moduls initialisieren. Im Anschluss erfolgt die Synchronisation des Knotens. Sobald dieser Nachbarn gefunden hat, wird das eigentliche Routing durch den Aufruf der Methode „eventhandling“, gestartet, welche in Abbildung 3.5 noch genauer dargestellt wird.

3.3.1 Flussdiagramm der Methode „main“

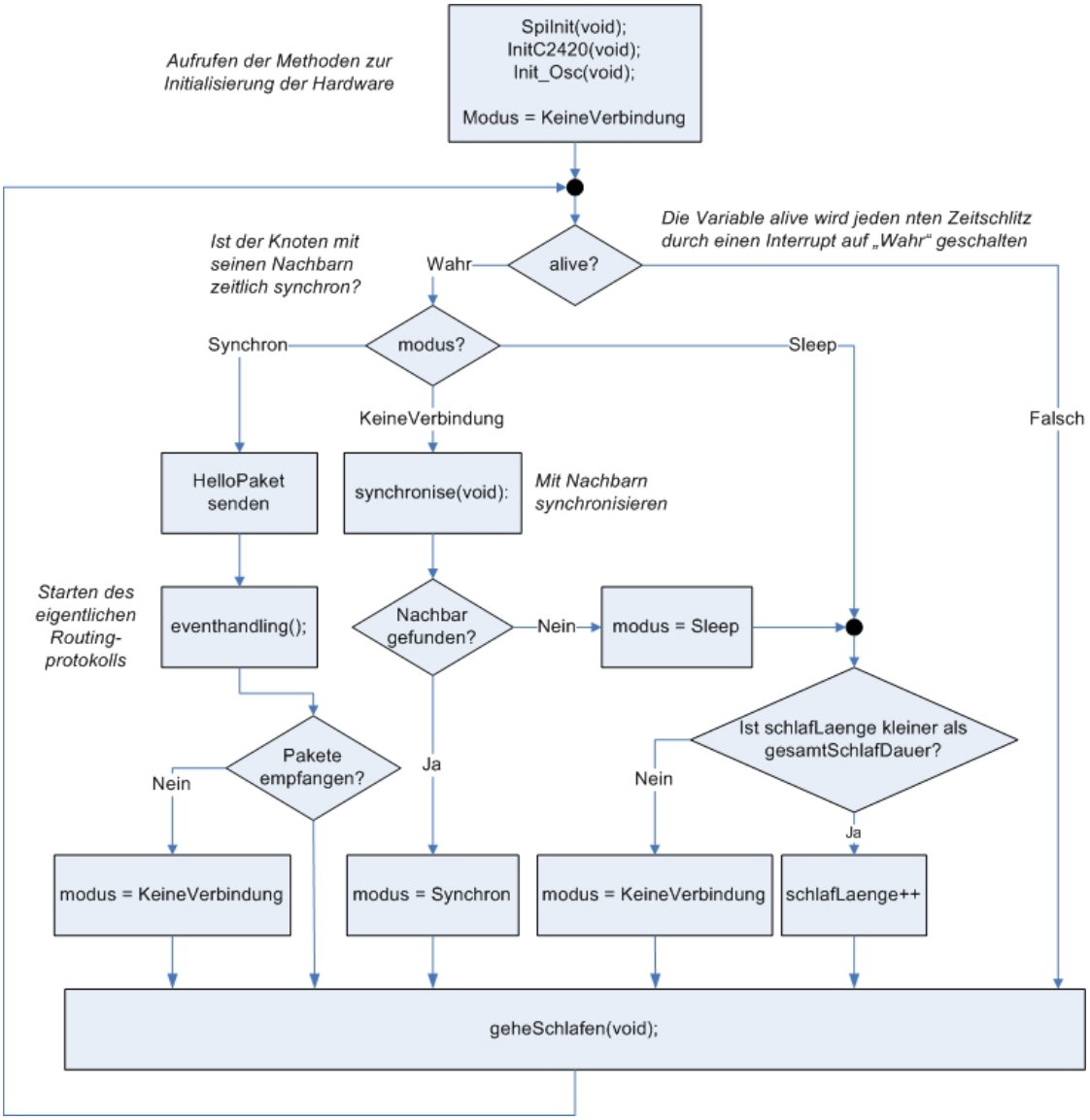


Abbildung 3.4: Flussdiagramm: main

3.3.2 Flussdiagramm der Methode „eventhandling“

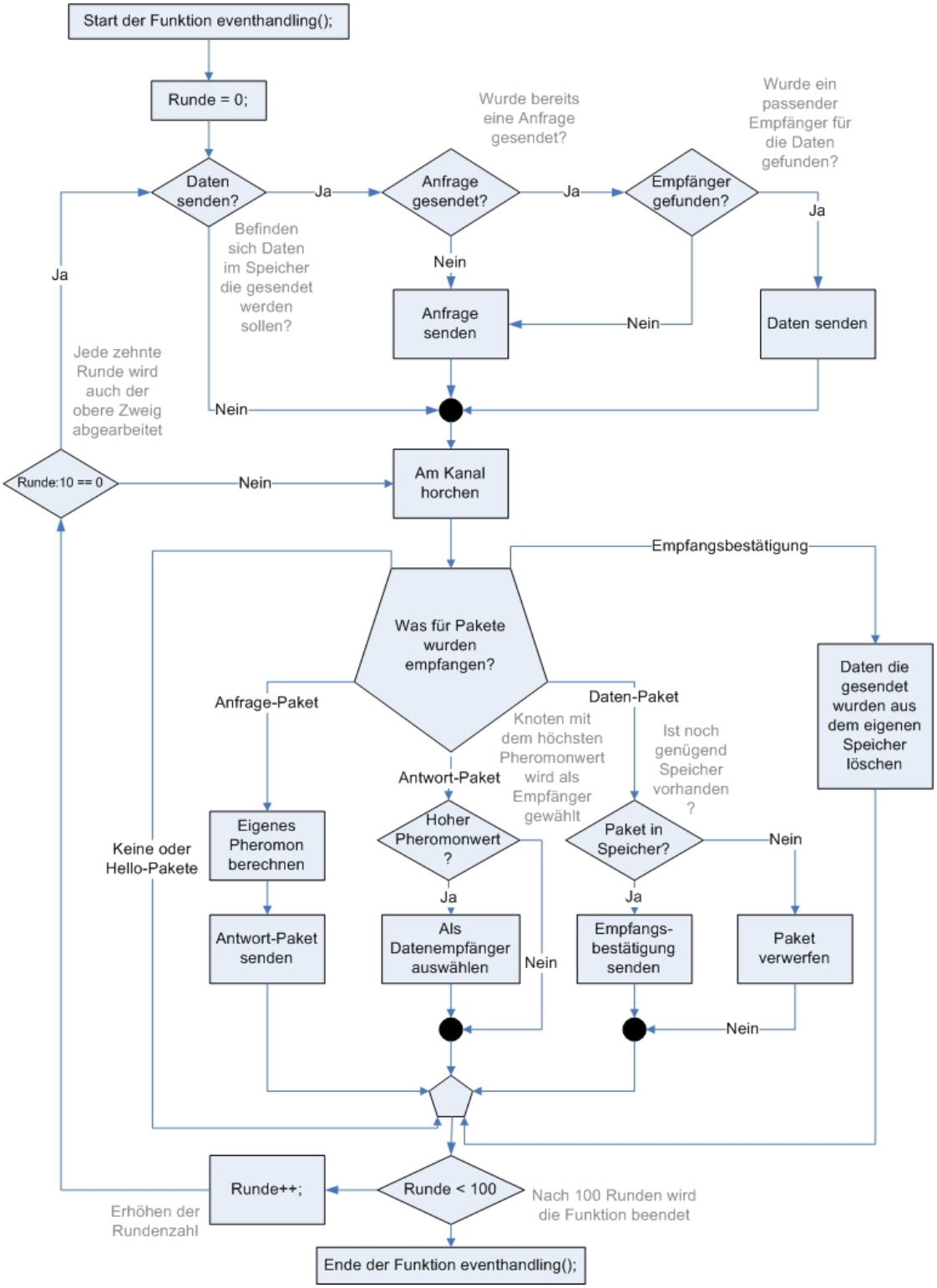


Abbildung 3.5: Flussdiagramm: eventhandling

3.4 Ablaufsteuerung und Synchronisation der Netzwerkknoten

Unter einem Interrupt versteht man in der Informatik eine kurzfristige Unterbrechung eines Programms, die meist durch ein externes Signal, wie zum Beispiel Tastatur- oder Mauseingabe ausgelöst wird. Während dieser Unterbrechung wird die ISR (englisch: Interrupt Servic Routine) abgearbeitet, welche für dieses Signal geschrieben wurde. Abgesehen von Eingabegeräten können Interrupts aber auch durch zeitliche Ereignisse ausgelöst werden. [3]
Der MSP430 verfügt über einen Zähler (Timer_A), dessen Wert bei jedem Takt des Quarzes um eins erhöht wird. Dieser Wert wird ständig mit dem Inhalt des TACCR0-Registers (englisch: Timer_A Capture Compare Register) verglichen. Bei Übereinstimmung findet ein Überlauf des Zählers statt, welcher den Inhalt zurück auf null setzt. Weiters wird durch dieses Ereignis ein Interrupt ausgelöst. Der zwei Byte große Wert des TACCR0-Registers kann selbst bestimmt beziehungsweise verändert werden. Würde man dieses Register mit seinem größten Wert (2 Byte: 65535) beschreiben so ergibt sich bei Benutzung eines Quarzes, welcher mit 32768 Hertz schwingt, eine Zeit von exakt zwei Sekunden. Um einen größeren Abstand zwischen den Unterbrechungen zu erzielen, kann der Takt des Quarzes zusätzlich noch durch bis zu sechzehn geteilt werden bevor er das Timer_A Register erhöht. Dadurch ergibt sich ein maximaler Zeitabstand von 128 Sekunden.

Um den Energieaufwand der einzelnen Module gering zu halten, kann der Mikrokontroller in einen Strom sparenden Modus (LPM3) versetzt und der Transceiver-Chip ausgeschaltet werden. Während dieser Zeit können weder Programmcode abgearbeitet, noch Pakete empfangen oder gesendet werden. Dieser Ruhezustand kann nur durch eine Unterbrechung (Interrupt), welche durch das Überlaufen des Timer-Registers ausgelöst wird, beendet werden. Um möglichst energiesparend zu sein, sollten die Knoten ihre Aufgaben in relativ kurzer Zeit erledigen, um rasch wieder in den Ruhezustand zurückkehren zu können. Die Zeitspanne, in welcher die Messdaten aufgenommen werden und der Datenaustausch statt findet, wird die Aktive Phase genannt. Addiert man zu der Zeit der Aktiven Phase noch jene des Ruhezustandes, so erhält man einen kompletten Zeitzyklus.

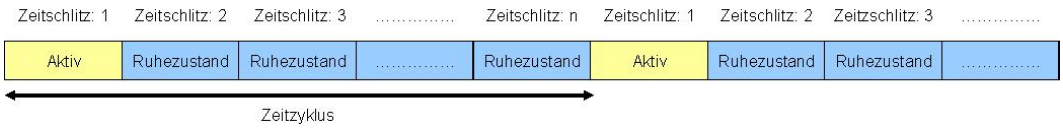


Abbildung 3.6: Zeitzyklus

Da benachbarte Module untereinander Daten austauschen, muss garantiert sein, dass sie alle zum selben Zeitpunkt in den aktiven Zustand zurückkehren. Als Vorgabe für den Beginn des Aktiven- und des Ruhezustands wird die Basisstation herangezogen. Das bedeutet, dass jeder Knoten im Netz, nach dem er sich synchronisiert hat, zeitgleich mit der Basis erwacht.

3.4.1 Erstsynchronisation

Bei Inbetriebnahme des Sensornetzes wird jeder Knoten zu einem anderen Zeitpunkt aktiviert, daher muss mit der Synchronisation gestartet werden. Jeder Knoten horcht zunächst einen kompletten Zeitzyklus lang den Kanal ab. Empfängt er ein Hello-Paket, so wird der Zeitpunkt (Zeitschlitz, Timerwert) dieses Ereignisses gespeichert. Bei Empfang mehrerer Pakete wird der Empfangszeitpunkt des Paketes mit dem höchsten Pheromonwert ausgewählt. Im nächsten Zeitzyklus wird solange gewartet, bis der gespeicherte Zeitpunkt wieder erreicht wird. Nun werden Zeitschlitz und Timerwert mit null initialisiert. Der Knoten ist nun synchron und kann mit seiner Tätigkeit beginnen. Außerdem sendet er zu Beginn jeder aktiven Phase ein Hello-Paket aus welches seinen Pheromonwert beinhaltet, um seinen Nachbarn ebenfalls die Synchronisation zu ermöglichen. Da bei Inbetriebnahme des Netzes noch kein Knoten synchron sein kann, ist die Basisstation der einzige Teilnehmer der Hello-Pakete ausschickt. Daher erfolgt die Synchronisation schichtweise von der Basis weg. Das Codebeispiel aus Abschnitt 5.2 soll helfen das Prinzip der Synchronisation zu verdeutlichen.

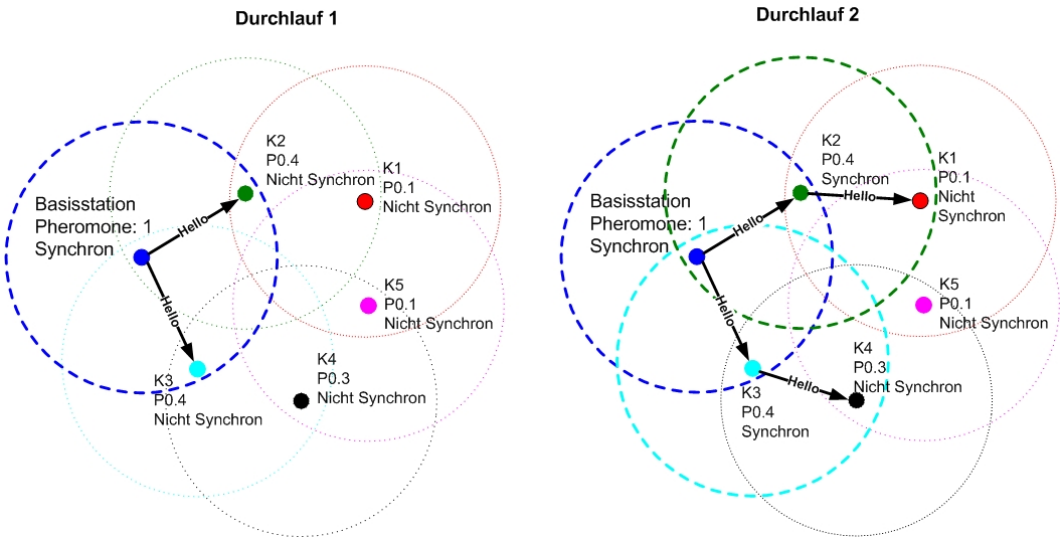


Abbildung 3.7: Synchronisation

3.4.2 Nachjustierung

Die Timer der jeweiligen Sensoren werden mit Uhrenquarzen getaktet. Die Toleranzen der einzelnen Quarze führen zu einem Auseinanderdriften der aktiven Phasen der Sensoren. Daher ist es möglich, dass ein Prozessor erwacht, während seine Nachbarn noch inaktiv sind. Sollte dies der Fall sein, ist es die Aufgabe eines Knotens sich selbst darum zu kümmern wieder aktiv zu werden. Merkt ein Knoten, dass Anfragen an seine Nachbarn nicht beantwortet werden, so beginnt er selbständig mit der in 3.4.1 beschriebenen Synchronisation.

3.5 Aufbau von Datenpaketen

Der CC2420 verwendet Datenpakete die dem IEEE 802.15.4 Standard [13] entsprechen. Das bedeutet, dass beim Senden und Empfangen von Paketen auf den richtigen Aufbau des Headers geachtet werden muss.

1 Byte	2 Byte	1Byte	2 Byte	2 Byte	2 Byte	2 Byte	Variabel	2
Paket-länge	Frame Control	Paket-Nr.	Ziel-PAN-Adresse	Ziel-Adresse	Absender-PAN-Adresse	Absender-Adresse	Daten	FCS
			Adressfelder					

Abbildung 3.8: Datenheader

3.5.1 Paketlänge

Da die Datenmenge variabel ist, muss die Paketlänge bei jedem Sendevorgang angegeben werden. Dadurch kann der Empfänger erkennen, ob ein Paket vollständig empfangen wurde. Die Empfangs- (RXFIFO) und Senderegister (TXFIFO) des CC2420 weisen jeweils eine Länge von 128 Byte auf und definieren somit die maximale Größe eines Datenpaketes. Wie aus Abbildung 3.8 hervorgeht handelt es sich bei der Paketlänge um einen 8 Bit (1 Byte) großen Wert. Das Bit mit der höchsten Wertigkeit (MSB) ist jedoch reserviert und muss immer mit 1 initialisiert werden. Daher bleiben nur noch 7 Bit (entspricht einem maximalen Wert von 127) für die Angabe der eigentlichen Paketlänge. Trotzdem ist das Versenden von 128 Byte großen Paketen möglich, da das Byte für die Paktlänge in der Längenangabe nicht mitgezählt wird. Beispiel:

Paketlänge: 24

1 Byte	2 Byte	1Byte	2 Byte	2 Byte	2 Byte	2 Byte	11 Byte	2
Paket-länge	Frame Control	Paket-Nr.	Ziel-PAN-Adresse	Ziel-Adresse	Absender-PAN-Adresse	Absender-Adresse	Daten	FCS
			Adressfelder					

Abbildung 3.9: Beispiel für Datenheader

3.5.2 Frame Control Field

Hier können diverse Einstellungen für das Versenden vorgenommen werden. Beispielsweise kann definiert werden, ob der Empfänger eine Empfangsbestätigung senden soll. Diese Bestätigung kann bei Bedarf vom CC2420 automatisch generiert werden. Für detailierte Informationen siehe [13] Abbildung 35.

3.5.3 Paketnummer

Die Paketnummer ist eine fortlaufende Zahl, welche nach Senden eines Datenpaketes erhöht wird. Dadurch lässt sich erkennen, ob Pakete verloren gegangen sind beziehungsweise mehrmals empfangen wurden.

3.5.4 Adressfelder

Jeder Knoten verfügt über eine 2 Byte große Adresse. Zusätzlich steht noch eine 2 Byte große PAN-Adresse (englisch: Personal Area Network, siehe [2]) zur Verfügung.

Da sich das in dieser Arbeit beschriebene Sensornetz in keine logischen Netzwerke (PANs) aufteilt, wird die PAN-Adresse als Adresse für die Sensorknote verwendet. Das bedeutet, dass 4 Byte (32 Bit) für die Adressierung der Sensoren zur Verfügung stehen. Dadurch würden sich theoretisch 4.294.967.292 ($2^{32} - 4$) verschiedene Knoten ansprechen lassen.

3.5.5 Daten

Hierbei handelt es sich um die eigentlichen Nutzdaten (Messwerte). Das erste Byte wird allerdings verwendet, um Aufschluss über die in diesem Protokoll benutzten Paketarten zu geben (siehe 3.1.2).

3.5.6 Frame Check Sequence (FCS)

Die FCS ist eine Prüfsequenz die gewährleisten soll, dass ein Paket fehlerfrei empfangen wurde. Dies geschieht über Checksummen, die der CC2420 automatisch generiert und überprüft (siehe [11] Seite 38). Beim Senden eines Paketes werden diese 2 Byte von der Hardware an das Paket angefügt, so dass sich der Softwareentwickler nicht weiter darum zu kümmern braucht. Da dem Empfänger die Richtigkeit der Checksumme zwar wichtig, der Wert selber jedoch nicht von Interesse ist, wird sie im Empfangsregister (RXFIFO) durch den RSSI-Wert (englisch: Receive Signal Strength Indicator) ausgetauscht, welcher Aufschluss über die empfangene Feldstärke gibt.

3.6 Senden und Empfangen von Paketen

3.6.1 Senden

Die Daten werden über die SPI-Schnittstelle vom Mikrocontroller des Sensorknotens in das 128 Byte große FIFO Register „TXFIFO“ des CC2420 übertragen. Das „TXFIFO“ wird über die Hexadezimaladresse 0x3E angesprochen. Während des Schreibvorganges in die Register muss die Chipselect-Leitung des Transceiver auf logisch „0“ gesetzt sein (ansonsten logisch „1“). Die tatsächliche Übertragung beginnt erst mit dem Absetzen des Kommandos „STXON“. Es ist wichtig, dass nicht unmittelbar nach der Datenübertragung in den Empfangsmodus gewechselt wird, da dies zur Beschädigung des Datenpaketes führen könnte. Daher muss darauf geachtet werden, dass die SFD-Leitung des CC2420, welche während des Sendevorganges auf logisch „0“ steht, zurück auf logisch „1“ schaltet und hiermit das Ende der Übertragung signalisiert. Ein Codebeispiel zum Senden von Paketen ist in Abschnitt 5.1.1 ersichtlich.

3.6.2 Empfangen

Im ersten Schritt muss in den Empfangsmodus gewechselt werden. Dies geschieht mit dem Kommando „SRXON“. Als nächstes wird die Chipselect-Leitung auf logisch „0“ gesetzt und das Empfangsregister „RXFIFO“ mit der Hexadezimaladresse 0x7F angesprochen. Nun muss auf das Eintreffen eines Datenpaketes gewartet werden. Falls ein Paket mit passender Zieladresse erkannt wird, schaltet der CC2420 die FIFOP-Leitung auf logisch „1“. Die empfangenen Daten können jetzt einfach aus dem Empfangsregister gelesen werden. Sobald mit dem Auslesen des letzten empfangenen Bytes begonnen wird, schaltet die FIFO-Leitung auf logisch „0“ um das Ende des Paketes zu signalisieren. Um den Empfang sauber zu beenden, wird die Chipselect-Leitung zurück auf logisch „1“ gesetzt. Ein Beispiel für den Empfang von Paketen ist in der Funkion synchronise im Abschnitt 5.2 zu finden.

3.7 Vermeidung von Gleitkommazahlen

Der Mikrokontroller des Sensorknotens MSP430 verfügt, wie die meisten Mikrokontroller, über keine FPU (englisch: Floating Point Unit). Daher ist die Verarbeitung von Gleitkommazahlen sehr zeit- und speicheraufwendig. Der Pheromonwert, welcher theoretisch eine Zahl zwischen 0 und 1 ist, wurde daher in natürlichen Zahlen realisiert. Um auch die Größe der Datenpakete nicht unnötig zu erhöhen, wurde der Wertebereich zwischen 0 und 255 festgelegt, welcher sich mit einem Byte (8 Bit) darstellen lässt. Bei den in 3.2 beschriebenen Berechnungen wird der Pheromonwert mit anderen Größen, welche ebenfalls zwischen 0 und 255 liegen, multipliziert. Daher könnte das Ergebnis dieser Berechnungen größer als 255 sein. Um dies zu vermeiden muss nach jeder Multiplikation das Ergebnis durch 255 dividiert werden. Da Divisionen jedoch auch relativ zeitaufwendig sind, wurden sie durch das 8malige nach rechts Verschieben des Bitmusters ersetzt. Das folgende Beispiel soll dieses Prinzip verdeutlichen:

1 ist der höchstmögliche Pheromonwert und wird daher mit 255 dargestellt. Wir nehmen an, dass dieser Wert nun mit einer anderen Größe von ebenfalls 1 (255) multipliziert wird. Das Ergebnis dieser Berechnung sollte wiederum 1 (255) ergeben.

255 * 255 = 65025
65025 entspricht einem Bitmuster von 1111 1110 0000 0001
Verschiebt man dieses Bitmuster um 8 Stellen nach rechts erhält man folgenden Wert
1111 1110 = 254

Wie in obigen Beispiel zu sehen ergibt sich ein geringer Fehler bei der Berechnung. Da dieser Fehler jedoch bei jedem Knoten entsteht und das Pheromon nur als Vergleichswert dient, ist dieser zu vernachlässigen.

3.8 Entwicklungsumgebungen

Die Betriebsoftware der Sensorknoten wurde in der Entwicklungsumgebung „IAR - Embedded Workbench“ verfasst. Eine kostenlose Testversion ist auf der Homepage http://www.iar.de/ erhältlich. Die einzige Beschränkung hierbei ist, dass der Programmcode nur eine maximale Größe von 4 Kilobyte erreichen darf, was jedoch für unsere Zwecke ausreicht.

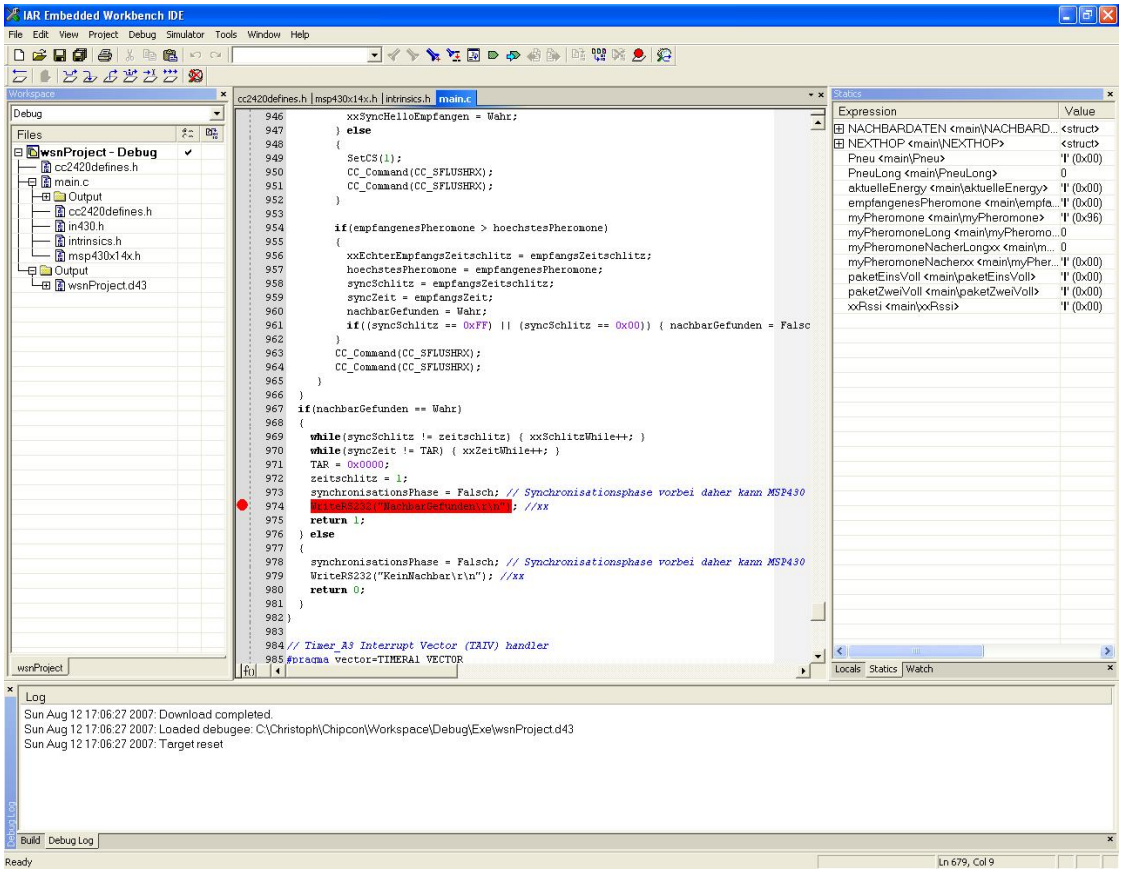


Abbildung 3.10: IAR Embedded Workbench

Die Oberfläche der IAR-Entwicklungsumgebung (Abbildung: 3.10) ist in 4 Einheiten aufgeteilt:

- Workspace: (Links) Gibt eine Übersicht über alle Dateien die dem aktuellen Projekt angehören
- Hauptfenster: (Mitte) Editor zur Gestaltung des eigentlichen Programmcodes
- Debugfenster: (Rechts) Bietet die Möglichkeit den Inhalt von Variablen, Registern und Adressen im Speicher zu überprüfen
- Compilerfenster: (Unten) Hier werden Warnungen beziehungsweise Fehlermeldungen des Compilers angezeigt

Die Tatsache, dass die einzelnen Sensorknoten über keinerlei Ausgabemöglichkeiten wie beispielsweise ein Display oder Leuchtdioden verfügen, erschwert die Suche nach Fehlern in der Software erheblich. Eine Möglichkeit, diese Schwierigkeit zu umgehen, ist das Setzen von Breakpoints (Maximal 2). Hierzu wird eine gewünschte Codezeile im Programm markiert. Sobald die besagte Zeile abgearbeitet wird stoppt das Programm. Nun ist es möglich die Werte der benutzten Variablen anzusehen und somit Informationen zum Programmablauf zu erhalten. Das Problem hierbei ist, dass die Tätigkeit des Knoten gestoppt wird und er daher nicht mehr mit anderen Netzteilnehmern kommunizieren kann. Des Weiteren ist er durch die Unterbrechung nicht mehr im richtigen Zeitintervall und muss sich nach Beendigung der Unterbrechung erst neu synchronisieren. Daher ist es beinahe unmöglich auf diese Weise brauchbare Informationen über die Kommunikation mehrerer Knoten zu erhalten.
Die zweite Möglichkeit ist der Zugriff auf die Serielle Schnittstelle. Diese ermöglicht es auch bei laufenden Programm Daten auf ein Hyper-Terminal auszugeben. Dennoch ist es schwierig den genauen zeitlichen Verlauf der Kommunikation mehrerer Knoten zu ermitteln.
Das gemeinsame Problem der beiden Methoden ist, dass man zur Überprüfung der Programmabläufe für jeden Sensorknoten auch einen eigenen Computer benötigt.
In Abschnitt 5.5 befinden sich Codebeispiele zur Initialisierung sowie Benutzung der Seriellen Schnittstelle.

4

Entwicklung eines Sensorknotens

4.1 Anforderungen an die Hardware

Sensornetzwerke bestehen meist aus einer Vielzahl von Knoten, welche sich auch häufig in freier Natur befinden können. Daher ergeben sich spezielle Anforderungen an die Hardware:

- Wasserdichtes Gehäuse, um vor Umwelteinflüssen geschützt zu sein
- Leicht und klein, um einfaches Anbringen der Sensoren zu gewährleisten
- Billig produzierbar
- Energiesparend, um eine lange Lebensdauer zu gewährleisten

Da es etliche Anwendungen für Sensornetzwerke gibt, sollten auch eine Vielzahl von Anschlüssen für die verschiedendsten Sensoren leicht zugänglich gemacht werden, um auf unterschiedliche Anforderungen eingehen zu können.

4.2 Aufbau eines Sensorknotens

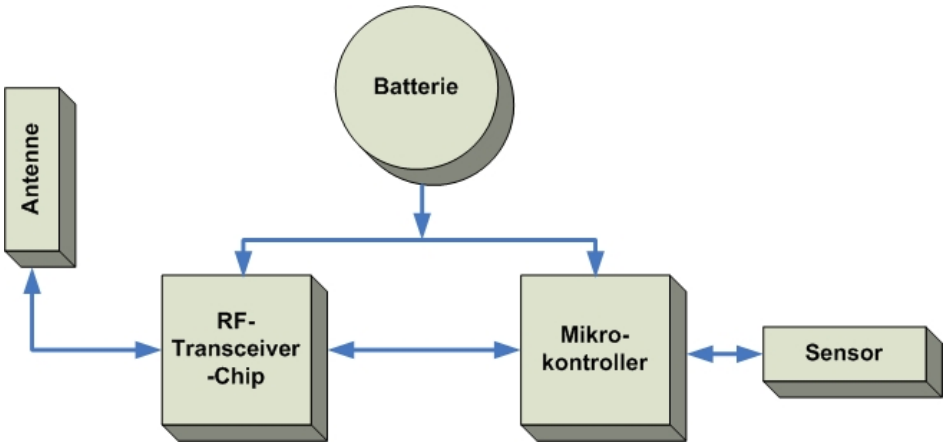


Abbildung 4.1: Blockschaltbild eines Sensorknotens

Üblicherweise werden die in Abbildung 4.1 ersichtlichen Komponenten auf eine Leiterplatte mit den entsprechenden Leiterbahnen gelötet. Da jedoch der für dieses Projekt gewählte Transceiver Chip und die dafür vorgesehene Antenne sich bereits auf einer eigenen Leiterplatte (Evaluation Module, Abbildung 4.2) befinden, wurde eine zweite Platine von gleicher Größe entwickelt, welche das fertige Transceiver Modul huckepack aufnehmen kann.

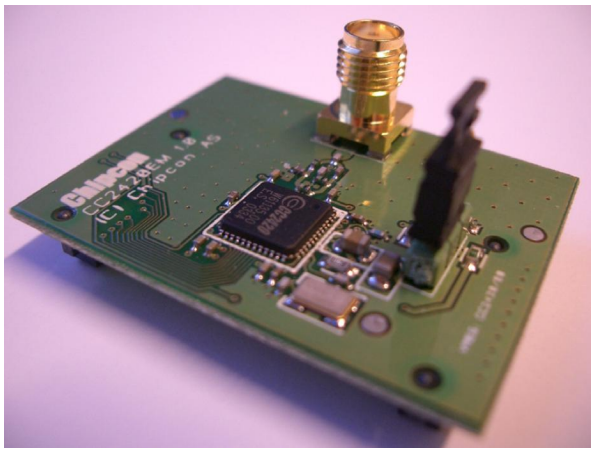


Abbildung 4.2: CC2420 Evaluation Module [10]

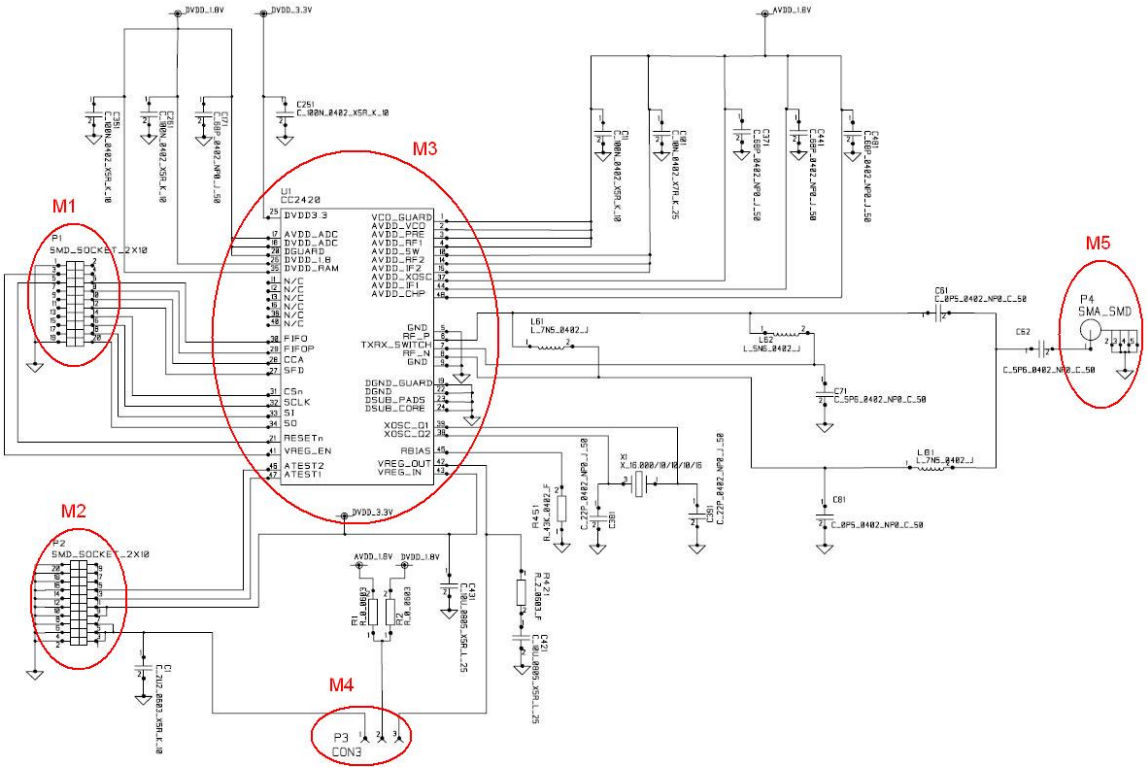


Abbildung 4.3: Schaltplan: CC2420 Evaluation Module [10]

Abbildung 4.3 zeigt den Schaltplan des Chipcon CC2420 Evaluation Module, auf welchem der RF-Transceiver Chip CC2420 (M3) sowie die SMA-Steckverbindung (M5) für die Antenne montiert sind. Der CC2420 verfügt über einen eingebauten Spannungsregler, welcher eine konstante Versorgungsspannung von 1,8 Volt liefert. Dieser kann jedoch mit Hilfe von Jumpern (M4) deaktiviert und durch einen externen Spannungsregler ersetzt werden. Die Verbindung des Evaluation Moduls mit der Leiterplatte erfolgt über die beiden SMD-Stecker (M1, M2), wobei M2 hauptsächlich der Spannungsversorgung dient. Die Kommunikation zwischen Transceiver-Chip und Mikrokontroller erfolgt über M1 und wird im Abschnitt 4.5 noch genauer beschrieben. Für weitere Informationen zum CC2420EM siehe [10].

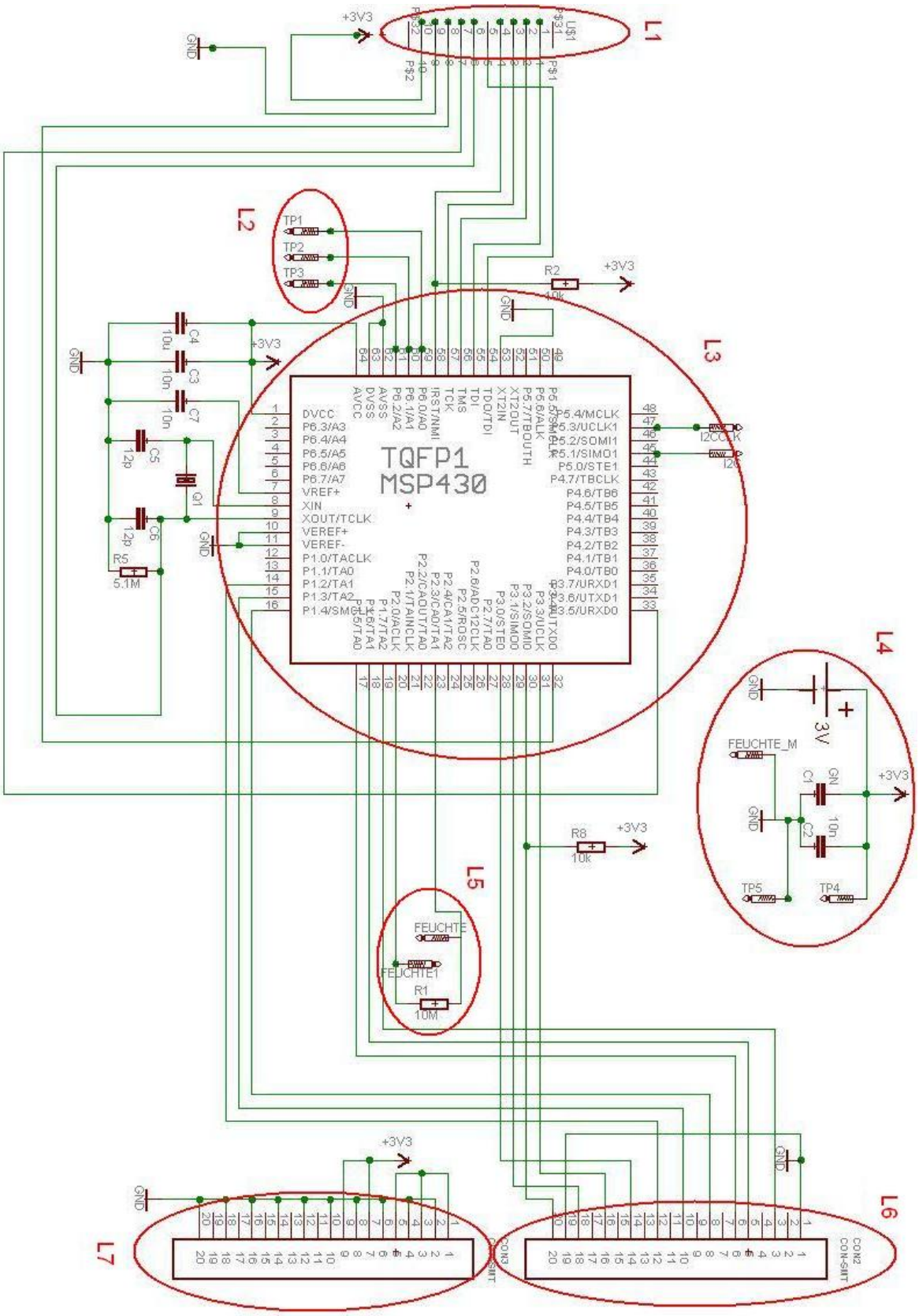


Abbildung 4.4: Schaltplan

Abbildung 4.4 zeigt den Schaltplan der Leiterplatte, welche das CC2420EM huckepack aufnimmt. Die einzige Verbindung zwischen den beiden Modulen ist durch die beiden SMD-Stecker gegeben, wobei M2 (Abbildung: 4.3) mit L7 (Abbildung: 4.4) und M1 mit L6 verbunden ist (siehe Abschnitt 4.5). Über eine weitere Steckverbindung (L1) wird das Programmieren des Mikrokontrollers (L3) sowie der Zugriff auf die serielle Schnittstelle (RS232) des PCs ermöglicht. L4 stellt die Spannungsversorgung von 3V dar, welche durch eine Lithium-Knopfzelle (CR2450) realisiert wurde.

Der hier beschriebene Sensorknoten sollte für die verschiedendsten Anwendungsgebiete benutzbar sein. Zu diesem Zweck wurden mehrere analoge (L2) und digitale (L5) Eingänge mit Lötaugen, welche sich leicht zugänglich am Rand der Leiterplatte befinden, verbunden.

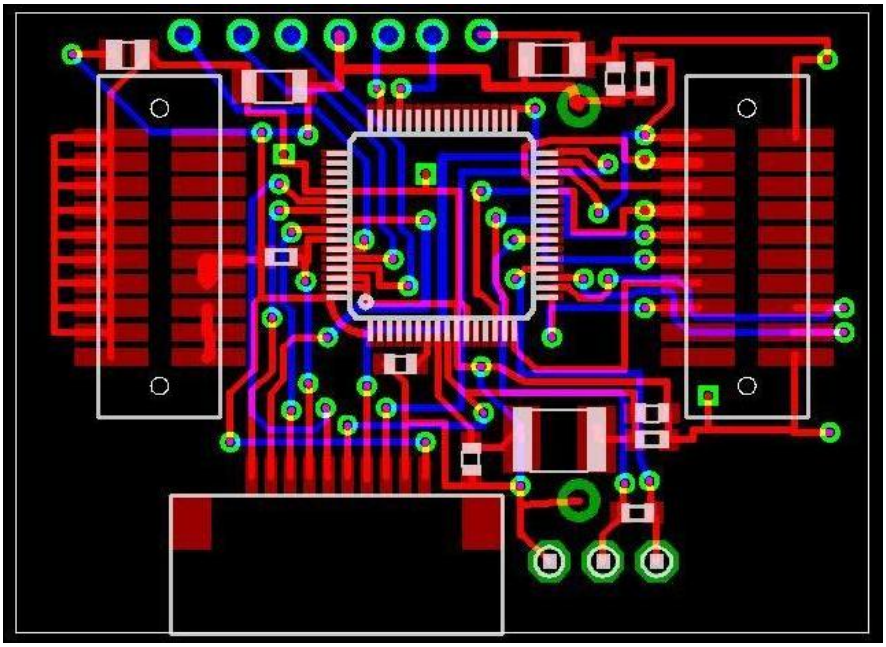


Abbildung 4.5: Layout eines Sensorknotens

Das Layout der Platine (Abbildung 4.5) besteht aus 2 Schichten. Elemente die sich auf der Oberseite der Leiterplatte befinden sind rot, jene auf der Unterseite blau dargestellt. Die mit grün dargestellten Kreise sind Durchkontaktierungen welche die beiden Schichten verbinden.

4.3 Der Mikrokontroller

Ein Mikrokontroller (MCU) dient meist der Steuerung von elektrischen Geräten. Es handelt sich hierbei um einen programmierbaren Integrierten Schaltkreis (IC). Ein MCU besteht üblicherweise aus:

- CPU
- Programm- und Datenspeicher
- Arbeitsspeicher
- Ein- und Ausgabekomponenten

Weiters verfügen moderne Mikrokontroller über Taktgeneratoren, Analog-Digital-Wandler, LCD-Controller und diverse Bussysteme (z.B.: SPI).

4.3.1 Aufgaben eines Mikrokontrollers

Der Mikrokontroller ist für die komplette Logik eines Sensorknotens verantwortlich. Alle Entscheidungen über das Senden und Empfangen von Paketen, die Aufnahme von Messdaten, sowie die Auswahl des am besten geeigneten Empfängers werden von ihm getroffen. Weiters ist er für die Taktgenerierung und die Konfiguration von externen Geräten (z.B.: Transceiver-Chip) verantwortlich. Um den Stromverbrauch gering zu halten sollte er auch seine eigenen Peripheriekomponenten sowie den Prozessor ausschalten, wenn diese gerade nicht benötigt werden.

4.3.2 Kriterien zur Auswahl des Mikrokontrollers

Für den Aufbau der Sensorknoten wurde der MSP430 F149 von Texas Instruments gewählt. Es handelt sich um einen sechzehn Bit Mikrokontroller, welchem eine Von-Neumann-Architektur zu Grunde liegt. Der Hauptgrund für die Wahl des MSP430 ist, dass er sich durch seinen extrem geringen Stromverbrauch auszeichnet. Die Stromaufnahme im aktiven Modus beträgt nur zweihundert Mikroampere. Zusätzlich ist es möglich, den MSP430 in fünf unterschiedliche stromsparende Zustände zu versetzen, bei welchen verschiedene Peripheriekomponenten ausgeschaltet werden. Dies ermöglicht es den ohnehin geringen Stromverbrauch auf 0,8 μA zu senken. Wie aus dem Namen MSP (englisch: Mixed Signal Processor) bereits hervorgeht, eignet er sich auch hervorragend für die Verarbeitung von analogen und digitalen Signalen. Dies ist von besonderem Vorteil, da eine Vielzahl von Sensoren nur analoge Signale liefern, welche von dem

Mikrokontroller erst in digitale Form gebracht werden müssen, um eine entsprechende Verwertung sowie die Übertragung zu ermöglichen. Des Weiteren ist der MSP430 in einem Plastic Quad Flat Pack Gehäuse (PQFP- 64) untergebracht und trägt somit zur geringen Größe des Sensorknotens bei.

Grundsätzlich besteht die MSP430 Mikrokontrollerfamilie aus mehreren verschiedenen Controllerderivaten, welche alle eine gemeinsame Grundstruktur aufweisen. Sie unterscheiden sich jedoch anhand ihrer Ausstattung mit Speicherbausteinen und Peripherieblöcken. Für weitere Informationen zum MSP430 F149 siehe [5].

4.3.3 Komponenten und technische Eigenschaften

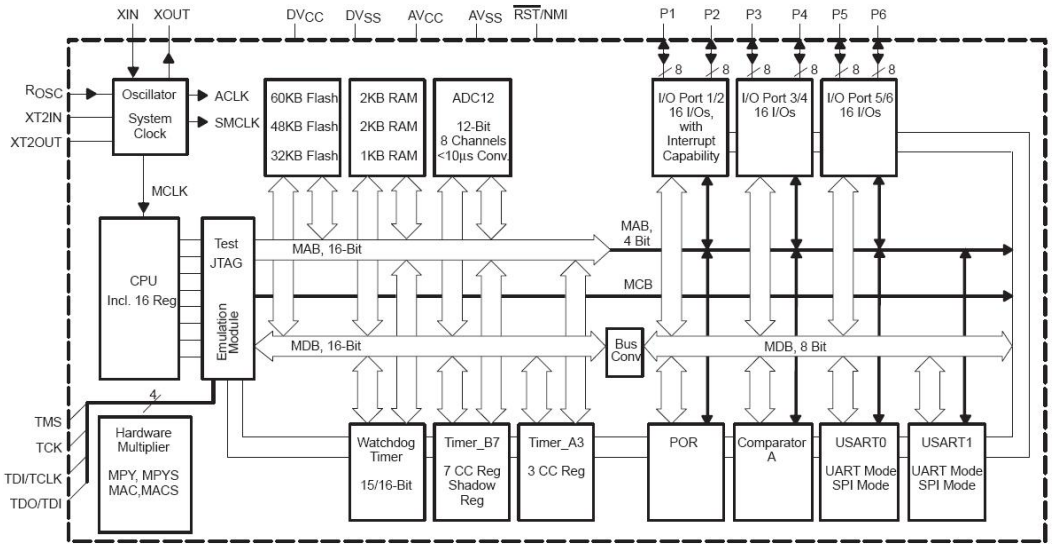


Abbildung 4.6: Blockschaltbild: MSP430 F149 [6]

Nähere Informationen zu allen in Abbildung 4.6 ersichtlichen Modulen des MSP430F149 sind dem Datenblatt [6] zu entnehmen. Im Anschluss werden kurz die wichtigsten Eigenschaften des Mikrokontrollers dargestellt:

- Geringe Versorgungsspannung: 1,8V bis 3,6V
- Geringer Stromverbrauch:
  - Standby Modus: 1,6 μA
  - RAM retention mode: 0,1 μA

- Geringer aktiver Stromverbrauch:
  - 2,5 μA bei 4 kHz, 2,2V
  - 280 μA bei 1 MHz, 2,2 V
- 5 stromsparende Modi
- Erwachen von Standby-Modus in 6 μs
- 16-Bit RISC Architektur (englisch: Reduced Instruction Set Computing)
- 12-Bit A/D-Wandler mit interner Referenzspannung, Sample and Hold
- 16-Bit Timer mit 7 Capture/Compare - Registern, Timer_B
- 16-Bit Timer mit 3 Capture/Compare - Registern, Timer_A
- On-Chip Comparator (Zum vergleichen 2er Spannungen)
- 60KB + 256B Flash Speicher
- 2KB RAM
- Erhältlich in einem 64-Pin Quad Flat Pack (QFP) Gehäuse
- Kosten: ca. 8 Euro/Stk. ab 1000 Stk.

Analog-Digital-Wandler

Sollten für eine spezielle Anwendung des Sensornetzes analoge Messwerte verarbeitet werden müssen, so geschieht dies über den Analog-Digital-Wandler ADC12 (Abbildung: 4.6 Vierter Block von links). Um die Stromaufnahme des Mikrokontrollers nicht unnötig zu erhöhen kann das ADC12-Modul deaktiviert werden wenn es nicht benötigt wird.

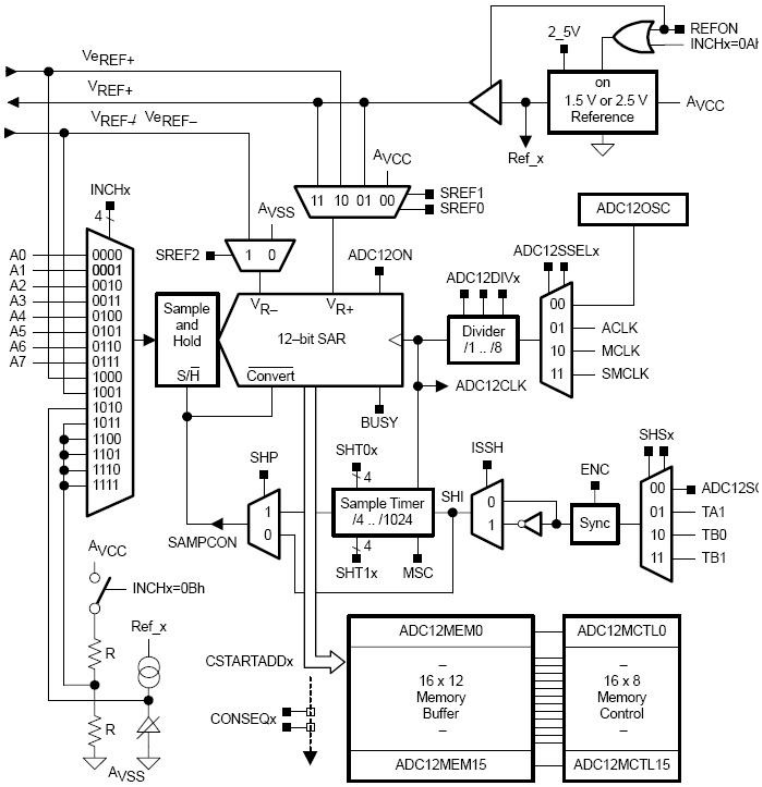


Abbildung 4.7: Blockschaltbild: ADC12 [5]

Eigenschaften des 12-Bit-Analog-Digital-Wandlers ADC12 [4]:

- Monoton über den gesamten Wandlungsbereich 0...FFFh
- Acht (zehn) externe analoge Eingänge und zwei interne Eingänge (Vcc/2 und Temperaturdiode)
- Relative (ratiometrische) oder absolute Messung möglich
- Zwei interne Spannungsreferenzen: 1,5 V und 2,5 V
- Sample-and-Hold-Funktion mit definierten Erfassungszeiten
- Mit und ohne Interrupt verwendbar
- Geringe Stromaufnahme und Abschaltmöglichkeit
- Automatische Messfolgen ohne CPU möglich
- 12-Bit-Auflösung
- Schnelle Wandelzeit ($< 10\ \mu$s)

- Vier ADC Clocks möglich: SMCLK, ACLK, MCLK, interner Oszillator
- Interne und externe Referenz möglich
- Großer Spannungsbereich

4.4 Der Transceiver Chip

Für die Datenübertragung wurde der Tranceiver Chip CC2420 von der Firma Texas Instruments gewählt. Es handelt sich hierbei um einen kostengünstigen IC, welcher eine drahtlose Datenübertragung im 2.4 GHz ISM Band (englisch: Industrial Scientific and Medical Band) ermöglicht. Als Übertragungsverfahren wird das „Direct Sequence Spread Spectrum“ (DSSS) Verfahren angewandt, welches die Übertragung relativ unempfindlich gegenüber schmalbandigen Frequenzstörungen macht [14].

Abbildung 4.8: Transceiver-Chip: CC2420 [15]

Der CC2420 übernimmt bei der Datenübertragung zahlreiche Aufgaben:

- Erzeugung einer Startsequenz (Synchronisierung)
- Adressierung
- Generieren einer Kontrollsumme, um eventuelle Übertragungsfehler zu erkennen

- Gepuffertes Empfangen von Paketen
- Bei Bedarf kann eine Empfangsbestätigung für gesendete Pakete angefordert werden

Weiters kann der RSSI-Wert (englisch: Receive Signal Strength Indicator), welcher Aufschluss über die Empfangsfeldstärke und somit über die Signalstärke des Absenders gibt, einfach ausgelesen werden. Der RSSI-Wert wird bei dem für dieses Projekt benutzten Ameisenalgorithmus als Entscheidungshilfe für die Auswahl des „besten“ Nachbarn benötigt. Ein weiterer Vorteil ist die geringe Stromaufnahme und die Tatsache, dass der CC2420 mit einer minimalen Spannungsversorgung von 2,1 Volt auskommt. Für weitere Informationen siehe [11].

4.4.1 Blockschaltbild und technische Eigenschaften

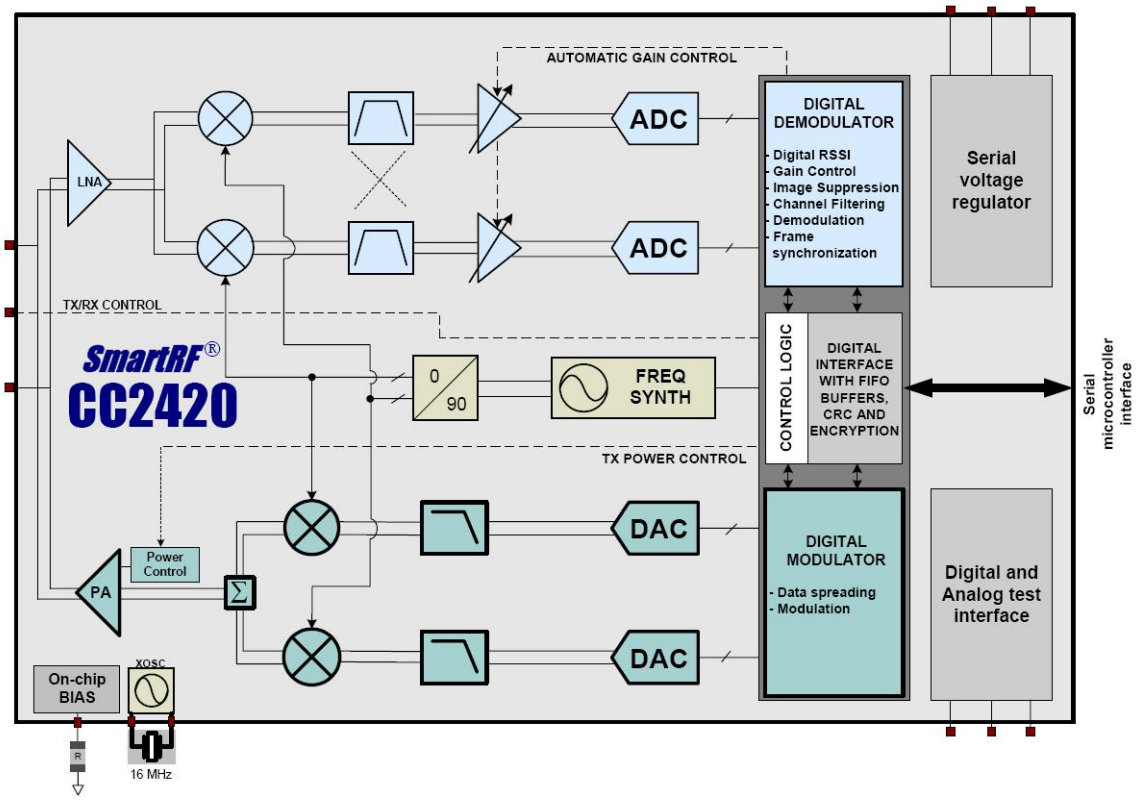


Abbildung 4.9: Blockschaltbild: CC2420 [11]

Die wichtigsten Eigenschaften des CC2420 im Überblick:

- 2,4 GHz IEEE 802.15.4 kompatibler RF-Transceiver-Chip mit Basisband-Modem; unterstützt MAC
- DSSS Basisband-Modem mit 2 MChipss und 250 kbps effektiver Datenrate
- Eignet sich sowohl für Reduced Function Device (RFD)- als auch für Full Function Device (FFD)- Anwendungen
- Geringer Stromverbrauch (RX: 18,8 mA, TX: 17,4 mA)
- Geringe Versorgungsspannung (2,1 - 3,6 V) mit internem Spannungsregler
- Geringe Versorgungsspannung (1,6 - 2,0 V) mit externem Spannungsregler
- Sehr wenige externe Komponenten
- 128(RX) + 128(TX) Byte Datenpufferung
- Digitaler RSSI / Link Quality Indication
- MAC Verschlüsselung (AES-128) durch Hardware
- Überwachung der Batteriespannung
- QLP-48 Gehäuse, 7x7 mm
- Leistungsfähige und flexible Entwicklungswerkzeuge sind erhältlich

4.5 Kommunikation zwischen MSP430 und CC2420

Der Mikrokontroller ist über zehn Leitungen mit dem Transceiver-Chip verbunden.

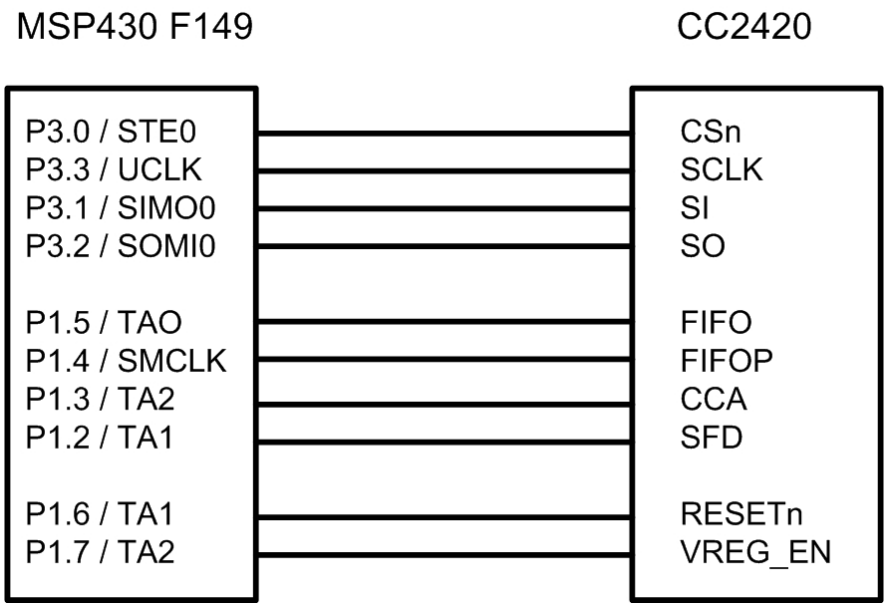


Abbildung 4.10: Verbindung zwischen MSP430 und CC2420

Wie in Abbildung 4.10 ersichtlich sind die zehn Verbindungsleitungen in drei logische Gruppen aufgeteilt.

4.5.1 SPI-Schnittstelle

Die obersten vier Leitungen aus Abbildung 4.10 bilden eine SPI-Schnittstelle (englisch: Serial Peripheral Interface). Sie werden für die Konfiguration und eigentliche Kommunikation zwischen den beiden Bauteilen benötigt.
SPI ist ein einfacher Standard, welcher es digitalen Komponenten ermöglicht nach dem Master-Slave Prinzip miteinander zu kommunizieren. In der Praxis wird diese Methode meist angewandt um einen Mikrokontroller mit Peripheriegeräten zu verbinden. Das bedeutet, dass eine SPI-Schnittstelle zwei oder mehrere Komponenten, nämlich einen Master und mindestens einen Slave, miteinander verbindet. Bei dem Master handelt es sich um jene Komponente (meist Mikrokontroller), welche den Takt angibt. Außerdem muss er entscheiden mit welchem Slave er kommunizieren möchte. Dies geschieht anhand der Chipselect-Leitung (CSn).
Die eigentliche Kommunikation geschieht über die restlichen drei Leitungen:

- SCLK: (englisch: Serial Clock) Gibt den Übertragungstakt an
- SI: (englisch: Serial In) Dateneingang der Masterkomponente
- SO: (englisch: Serial Out) Datenausgang der Masterkomponente

Aufgaben der SPI Schnittstelle bei einem Sensorknoten:

- Konfigurieren des Transceiver-Chips (Setzen und überprüfen der Konfigurationsregister)
- Übergabe der zu sendenden Daten an den Transceiver
- Auslesen der empfangenen Daten

4.5.2 Status-Leitungen

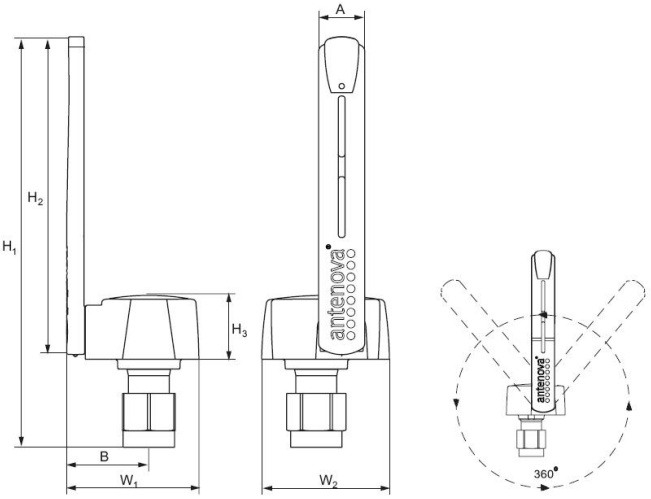
Über die Leitungen fünf bis acht können zusätzliche Informationen über den Status des Transceivers in Erfahrung gebracht werden. Wenn der Mikrokontroller sich in einem Energiesparmodus (englisch: Low Power Mode) befindet, kann er durch Pegelwechsel an einer Status-Leitung wieder in den aktiven Zustand wechseln. Dies ist zum Beispiel erforderlich, wenn auf das Eintreffen eines Paketes gewartet wird. Der Transceiver-Chip muss sich hierzu im Empfangsmodus befinden, während der Mikrokontroller, um den Stromverbrauch so gering wie möglich zu halten, sich in einen Ruhezustand versetzt. Bei Eintreffen eines richtig adressierten Paketes schaltet der Transceiver die FIFOP-Leitung auf logisch „1“. Dies löst beim Mikrokontroller eine Unterbrechung (englisch: Interrupt) aus. Er erwacht nun aus seinem Ruhezustand und kann die empfangenen Daten verarbeiten. Die FIFO, FIFOP und SFD Leitungen geben Auskunft über Ereignisse beim Senden und Empfangen von Datenpaketen. Die CCA-Leitung (englisch: Clear Channel Assessment) gibt hingegen Aufschluss darüber, ob der Übertragungskanal gerade frei oder besetzt ist. Sie wird daher benötigt um Kollisionen von Datenpaketen zu vermeiden. Für genauere Informationen siehe [11].

4.5.3 Versorgungsspannungs- und Resetleitung

Die VREG_EN (englisch: Voltage regulator enable) Leitung dient, wie der Name bereits vermuten lässt, dem Ein- und Ausschalten der Versorgungsspannung des Transceivers. Mit Hilfe der RESETn Leitung kann der CC2420 neu gestartet werden. Alle Register werden hierbei auf ihre Standardwerte zurückgesetzt.

4.6 Antenne

In Abschnitt 4.2 wurde bereits erwähnt, dass das CC2420-Evaluation-Module mit einer SMD-Steckverbindung, zur Montage einer Antenne ausgestattet ist. Es handelt sich hierbei um eine 2,4 GHz Antenne der Firma Antenova.



A	B	H1	H2	H3	W1	W2
		Height	Height	Height	Width	Width
7±0.2	12.5±0.5	62.5 ±0.5	48.3 ±0.5	9.5±0.5	20±0.3	19.5±0.3

Dimensions in millimeters

Abbildung 4.11: Antenne [7]

Die Antenne ist äußerst flexibel und kann wie aus Abbildung 4.11 ersichtlich um 360 Grad gedreht werden. Für weitere Informationen zur Antenne siehe: [7].
Eigenschaften:

Name	Titanis 2.4 GHz
Frequenz	2.4-2.5 GHz
Polarisation	Linear
Betriebstemperatur	-40 to + 85 °C
Impedanz	50 Ohm
Gewicht	7.4 g
Type	Swivel

4.7 Feuchte- und Temperatursensor

Je nach Anwendung des Sensornetzes können die Platinen mit den verschiedendsten Sensoren bestückt werden. Häufig verwendet werden DSNe für die Messung von:

- Druck
- Helligkeit

- Temperatur
- Beschleunigung
- Windstärke
- Luftfeuchte
- Schall

Da dieses Projekt noch für keine bestimmte Anwendung ausgelegt ist, wurden entsprechende Anschlüsse für verschiedene Sensoren an den Rand der Leiterplatte auf Lötaugen zugänglich gemacht.
Zu Testzwecken wurde der Feuchte- und Temperatursensor SHT71 der Firma SENSIRION montiert. Die wichtigsten Kriterien für die Wahl dieses Produktes waren die geringe Stromaufnahme und die Fähigkeit, Messungen auch bei niedriger Versorgungsspannung durchführen zu können. Ein weiterer wesentlicher Vorteil besteht darin, dass die Ergebnisse bereits in digitaler Form zur Verfügung gestellt werden.

Kriterien zur Auswahl des SHT71:

- Vollständig kalibrierter und digitaler Ausgangswert
- Benötigt keine externen Komponenten
- Geringer Stromverbrauch
- Geringe Versorgungsspannung: 2,4 - 5,5 V
- Geringe Größe

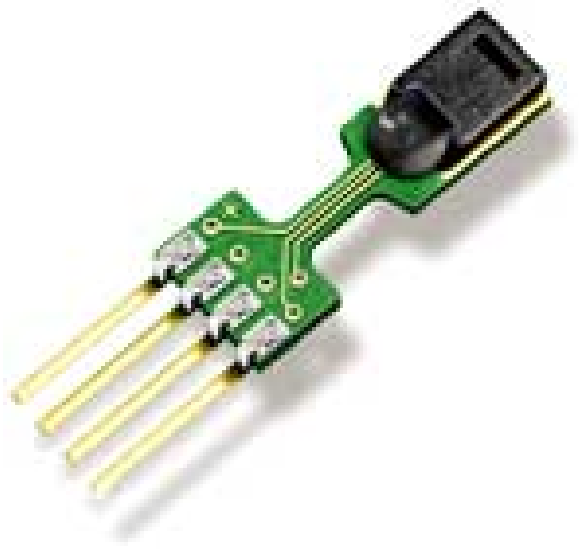


Abbildung 4.12: Feuchte- und Temperatursensor SHT71 [12]

Die Kommunikation mit dem SHT71 geschieht leider nicht über eine SPI-Schnittstelle oder einen anderen gebräuchlichen Standard. Daher musste ein eigenes kurzes „Protokoll“ für die Datenaufnahme entwickelt werden. Für eine genauere Beschreibung des „Protokolls“ sowie Informationen zu Aufbau und Pinbelegung siehe: [12]. In Abbildung 4.13 ist das Blockschaltbild des SHT71 dargestellt.

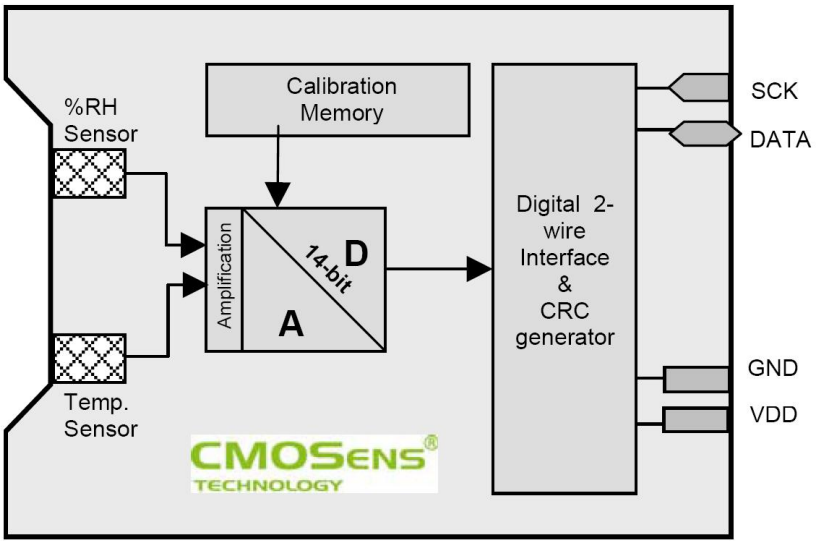


Abbildung 4.13: Blockschaltbild: Feuchtesensor SHT71 [12]

4.8 Leiterplatte

Leiterplatten dienen zur elektronischen Verbindung sowie zur mechanischen Befestigung von einzelnen Bauteilen. Sie bestehen aus einer mit Kupfer beschichteten Kunststoffplatte, auf welcher die Leiterbahnen durch Ätzen oder Fräsen aufgetragen werden. Diese Verbindungen verlaufen sowohl auf der Ober- als auch auf der Unterseite der Platine. Dies dient einerseits um die Packungsdichte der Bauteile gering zu halten und andererseits um ein Kreuzen der Leiterbahnen zu ermöglichen. Die Durchkontaktierungen, welche die Leitungen auf Ober- und Unterseite verbinden, bestehen aus innen metallisierten Bohrungen.
Während für die Architektur eines Sensorknotens eine zweischichtige Leiterplatte völlig ausreicht, werden bei komplexen elektronischen Schaltungen so genannte „Multilayer Platinen“ benutzt, welche aus bis zu achtundvierzig Schichten bestehen können.

4.8.1 Erzeugung der Leiterplatte

Um eine Leiterplatte erzeugen zu können muss zu allererst ein Schaltplan erstellt werden, auf welchem die einzelnen Bauteile logisch miteinander verbunden sind. Hierbei wird nur auf die Logik der Schaltung und nicht auf die tatsächliche Anordnung von Bauteilen und Leiterbahnen geachtet (siehe Abbildung 4.4). Im zweiten Schritt wird das eigentliche Layout der Platine erzeugt. Da die Elemente bereits durch den Schaltplan richtig verbunden sind, muss jetzt nur mehr auf eine vorteilhafte Platzierung der Bauteile, sowie auf einen Platz sparenden Verlauf der Leiterbahnen geachtet werden (siehe Abbildung 4.5). Das Layout und der Schaltplan für die Sensorknoten wurden mit dem Layout Editor EAGLE (siehe 4.8.2) konstruiert. Die daraus resultierenden Leiterplatten (Abbildung 4.14) wurden aus FR4-Material, welches eine Dicke von 1,6mm aufweist, von der Firma Conrad gefertigt. FR4 besteht aus Glasfasermatten, welche in Epoxidharz getränkt werden. Im Gegensatz zu Hartpapier zeichnet sich FR4 durch eine bessere Kriechstromfestigkeit, bessere Hochfrequenzeigenschaften sowie eine geringere Wasseraufnahme aus.

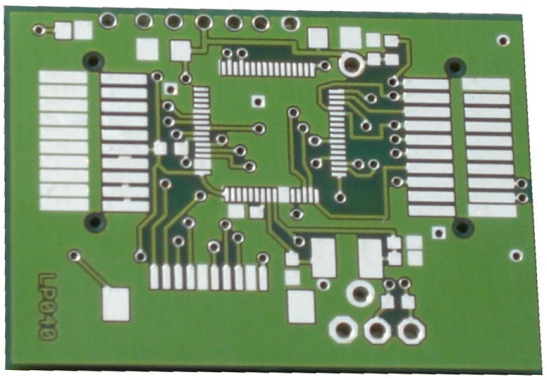


Abbildung 4.14: Oberseite der Leiterplatte

4.8.2 Der Layout Editor EAGLE

Der Schaltplan und die Architektur der Leiterplatte wurden mit dem CAD-Programm EAGLE (englisch: Easy Applicable Graphical Layout Editor) entwickelt. Die Vollversion dieser Software ist kostenpflichtig. Für die Herstellung eines Sensorknotens reicht jedoch die „light edition“ des Programms vollkommen aus. Dies kann kostenlos von http://www.cadsoft.de/ herunter geladen werden kann. Hierbei ist auf Folgendes zu achten:

- Die nutzbare Platinenfläche ist auf 100 x 80 mm beschränkt
- Die Platine darf aus maximal zwei Schichten bestehen
- Die Benutzung ist auf nicht-kommerzielle Anwendungen oder Evaluierung beschränkt

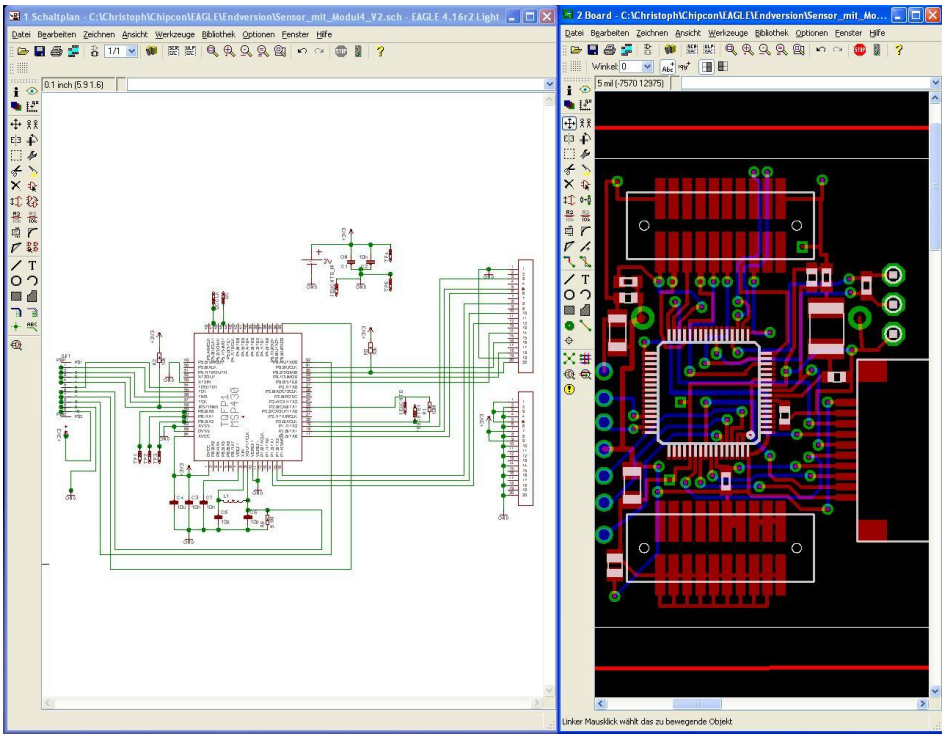


Abbildung 4.15: EAGLE

Wie aus Abbildung 4.15 hervorgeht, ist für die Erzeugung des Schaltplans und des Layouts jeweils eine eigene Umgebung verfügbar. Bauelemente die im Schaltplan hinzugefügt und verbunden wurden, sind automatisch auch im Layout ersichtlich. Die Bauteile müssen dann nur noch richtig positioniert und der Verlauf der Verbindungsleitungen vorteilhaft gewählt werden.

4.9 Anschlüsse

Die Platine wurde mit einer äußerst Platz sparenden, 10-poligen Steckverbindung ausgestattet, welche das Programmieren des Mikrokontrollers und das Auslesen des Speichers ermöglicht. Da die standardmäßige Steckverbindung des JTAG Adapters, welcher zur Verbindung von PC und Mikrokontroller dient, circa dreimal größer ist, wurde ein weiterer Adapter geschaffen, um die Platine nicht damit zu belasten.

4.10 Der JTAG- und RS232 Adapter

Um mit dem Mikrokontroller MSP430 kommunizieren zu können benötigt man diverse Steuerleitungen, welche leicht zugänglich sein sollten:

- 7 Programmierleitungen, um das Programm in den Flash Speicher zu schreiben und den Programmablauf überprüfen zu können (Debugfunktion)
- 4 Leitungen, um über die serielle Schnittstelle (RS232) des PC auf den Mikrokontroller zugreifen zu können

Der MSP430 wird üblicherweise mit Hilfe des JTAG Adapters programmiert, welcher die Verbindung zwischen PC und dem Mikrokontroller realisiert. Das Problem hierbei ist, dass die Steckverbindung des JTAG Adapters ziemlich groß ist und daher zuviel Platz auf der Platine benötigen würde.

Abbildung 4.16: JTAG Adapter [16]

Ein weiteres Problem ergibt sich beim Implementieren einer seriellen Schnittstelle da die Platine mit einem zusätzlichen Integrierten Schaltkreis, dem MAX3232 bestückt werden müsste. Der MAX3232 wird benötigt, um die Spannungspegel der seriellen Schnittstelle zu invertieren und an jene des Mikrokontrollers anzupassen. Würde man jeden Sensorknoten mit diesem Bauteil bestücken, hätte dies eine Erhöhung der Kosten und eine größere Dimensionierung der Leiterplatte zur Folge.
Um diese beiden Probleme zu vermeiden wurde von mir ein Adapter geschaffen, welcher sowohl das Programmieren als auch den Zugang über die serielle Schnittstelle ermöglicht.

4.10.1 Realisierung

Die sieben Programmierleitungen und die vier Anschlüsse der seriellen Schnittstelle können auf zehn Leitungen zusammengefasst werden, da die Verbindung mit dem

Massepotential für beide Aufgaben benötigt wird. Diese zehn Leitungen werden dem Adapter über eine Platz sparende Steckverbindung, welche sich am Rand des Sensorknotens befindet, zugänglich gemacht.
Am Adapter werden diese zehn Leitungen wieder auf zwei Stecker aufgeteilt. Außerdem enthält er den MAX3232, um die Kommunikation über die serielle Schnittstelle zu ermöglichen. Daraus ergeben sich folgende Vorteile:

- Kostenersparnis, da der MAX3232 lediglich auf dem Adapter aufgebracht werden muss und nicht auf jedem einzelnen Knoten im Netzwerk
- Deutlich kleinere Leiterplatte, da die zusätzlichen Bauteile viel Platz gebraucht hätten

Das Layout und der Schaltplan des Adapters wurden, wie auch die Leiterplatte (Abschnitt: 4.8), mit dem CAD-Programm EAGLE gestaltet und sind in den folgenden Abbildungen zu sehen. Das daraus resultierende Modul ist in Abbildung 4.19 ersichtlich.

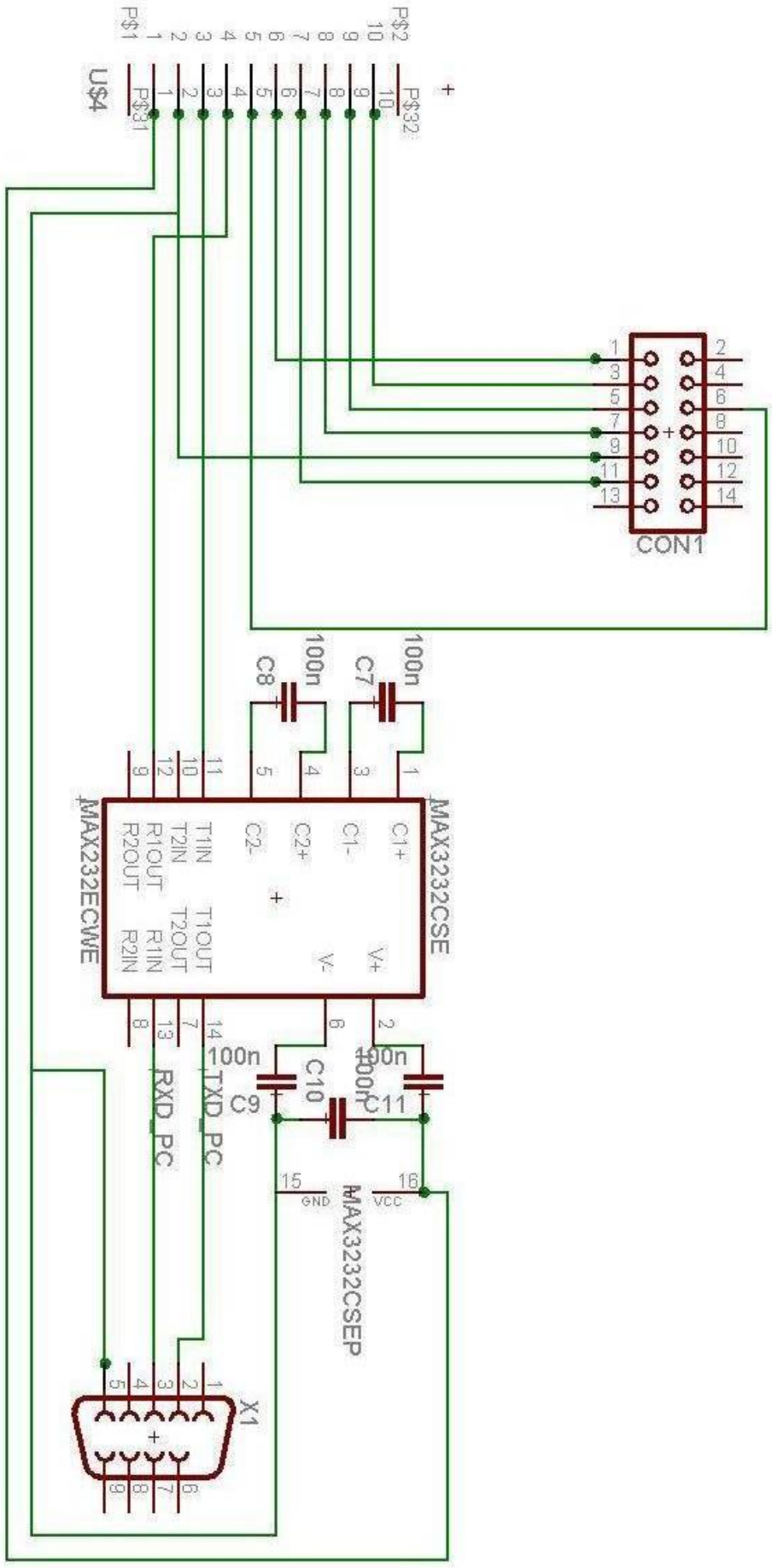


Abbildung 4.17: Schaltplan des Adapters

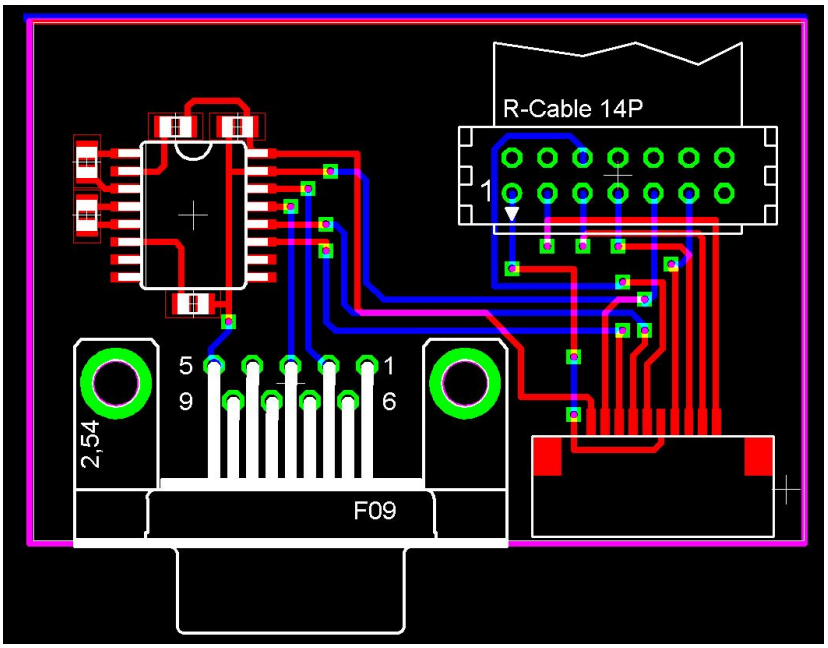


Abbildung 4.18: Layout des Adapters

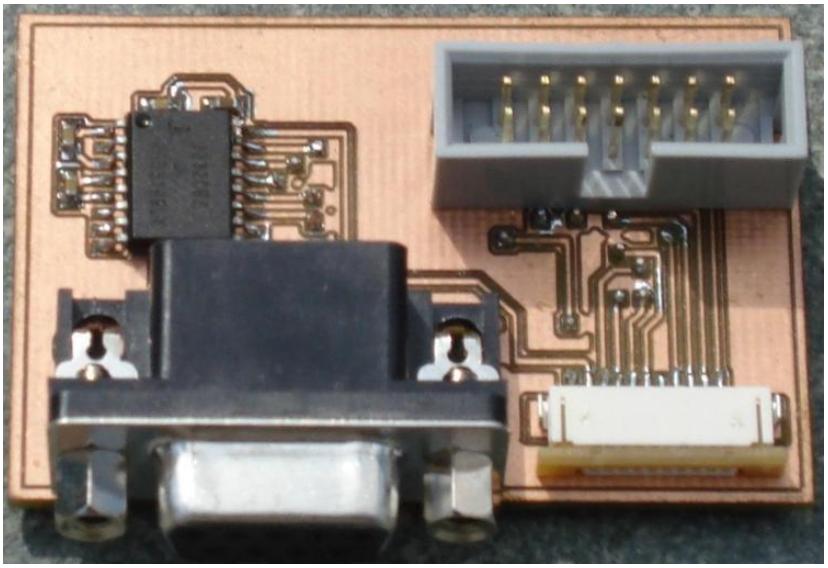


Abbildung 4.19: Adapter

5
Quellcode

In diesem Kapitel ist nicht der gesamte Quellcode des Projektes enthalten. Es sind lediglich Codebeispiele zu bereits behandelten Themen ersichtlich. Diese sollen dabei helfen das Verständnis der theoretischen Ausführungen zu festigen.

5.1 Datenübertragung

5.1.1 Daten senden

Die folgende Methode zeigt das Senden eines einfachen Paketes (z.B.: Hello-Paket). Zum starten der Übertragung wird eine weitere Funktion „CCA_Transmit“ aufgerufen, welche in Abschnitt 5.1.2 erklärt wird. Abbildung 5.1 zeigt den Aufbau eines Hello-Paketes.

<table>
<tr><td>1 Byte</td><td>2 Byte</td><td>1Byte</td><td>2 Byte</td><td>2 Byte</td><td>2 Byte</td><td>2 Byte</td><td>1 Byte</td><td>1 Byte</td><td>2</td></tr>
<tr><td rowspan="2">Paket-länge</td><td rowspan="2">Frame Control</td><td rowspan="2">Paket-Nr.</td><td>Ziel-PAN-Adresse</td><td>Ziel-Adresse</td><td>Absender-PAN-Adresse</td><td>Absender-Adresse</td><td rowspan="2">Paket-Art: Hello-Paket</td><td rowspan="2">Pheromon</td><td rowspan="2">FCS</td></tr>
<tr><td colspan="4">Adressfelder</td></tr>
</table>

Abbildung 5.1: Aufbau eines Hello-Paketes

```
1 void sendePaket(char paketArt, char adresse1, char adresse2, char panAdresse1, char panAdresse2)
2 {
3   SetCS(0);                    // Chipselct-Leitung auf 0
4   SpiTransmit(0x3E);           // TXFIFO addressieren
5
6   SpiTransmit(0x0E);           // Packet length = 14
7
8   SpiTransmit(0x88);           // FCF MSB
9   SpiTransmit(0x01);           // FCF LSB
10
```

```
11    SpiTransmit(paketNummer);          // TX Sequence Number
12
13    SpiTransmit(adresse1);             // PAN ID (Empfaenger)
14    SpiTransmit(adresse2);             // PAN ID      (Empfaenger)
15
16    SpiTransmit(panAdresse1);          // Adresse (Empfaenger)
17    SpiTransmit(panAdresse2);          // Adresse (Empfaenger)
18
19    SpiTransmit(meinePan1);            // PAN-Adresse (Absender)
20    SpiTransmit(meinePan2);            // PAN-Adresse (Absender)
21
22    SpiTransmit(meineAdresse1);        // Adresse (Absender)
23    SpiTransmit(meineAdresse2);        // Adresse (Absender)
24
25    SpiTransmit(paketArt);             // Hello-Paket
26    SpiTransmit(pheromon);             // Pheromonwert
27
28    SetCS(1);                          // Chipselct-Leitung auf 1
29    Delay(5);
30
31          // Wenn der Kanal frei ist: Absenden des Datenpaketes
32    CCA_Transmit();
33 }
```

5.1.2 Starten der Übertragung

Wenn zwei Knoten zur gleichen Zeit Daten senden, kann dies zu einer Kollision der Signale führen. Beide Pakete wären dadurch beschädigt und könnten nicht mehr korrekt empfangen werden. Der CC2420 bietet hierzu CCA-Funktionalität (englisch: Clear Channel Assessment). Das bedeutet, dass vor einem Sendevorgang am Kanal gelauscht wird, um festzustellen, ob bereits eine Datenübertragung im Gange ist. Am einfachsten ist diese Funktionalität zu realisieren, indem man das CC2420-Kommando „STXONCCA“ verwendet. Bei Absetzen dieses Kommandos überprüft der Transceiver-Chip ob der Kanal frei ist. Ist dies der Fall sendet er alle im TXFIFO befindlichen Daten ab. Andernfalls geschieht nichts. Die Information, ob ein Paket nun gesendet wurde oder nicht erhält man aus dem Status-Byte, welches beim Absetzen des SNOP-Kommandos zurückgegeben wird.

```
1  void CCA_Transmit(void)
2  {
3    char statusCCA;
4
5    CC_Command(CC_SRXON);              // SRXON Receiver einschalten
6    do
7    {
8      CC_Command(CC_SRXON);            // SRXON Receiver einschalten
9      CC_Command(CC_STXONCCA);
10     statusCCA = CC_Command(CC_SNOP);
11   }  while(!(statusCCA & BIT3));
12   while(P1IN & BIT2);                // Warten bis Daten uebermittelt
13 }
```

5.2 Beispiel zu Synchronisation

Im folgenden Flussdiagramm ist zu beachten, dass die Variable „zeitschlitz“ alle n Millisekunden um 1 erhöht wird. Ist die maximale Anzahl an Zeitschlitzen erreicht, so ist ein Zeitzyklus zu Ende und ein Neuer beginnt. Das Vorgehen bei der Synchronisation ist in Abschnitt 3.4 beschrieben.

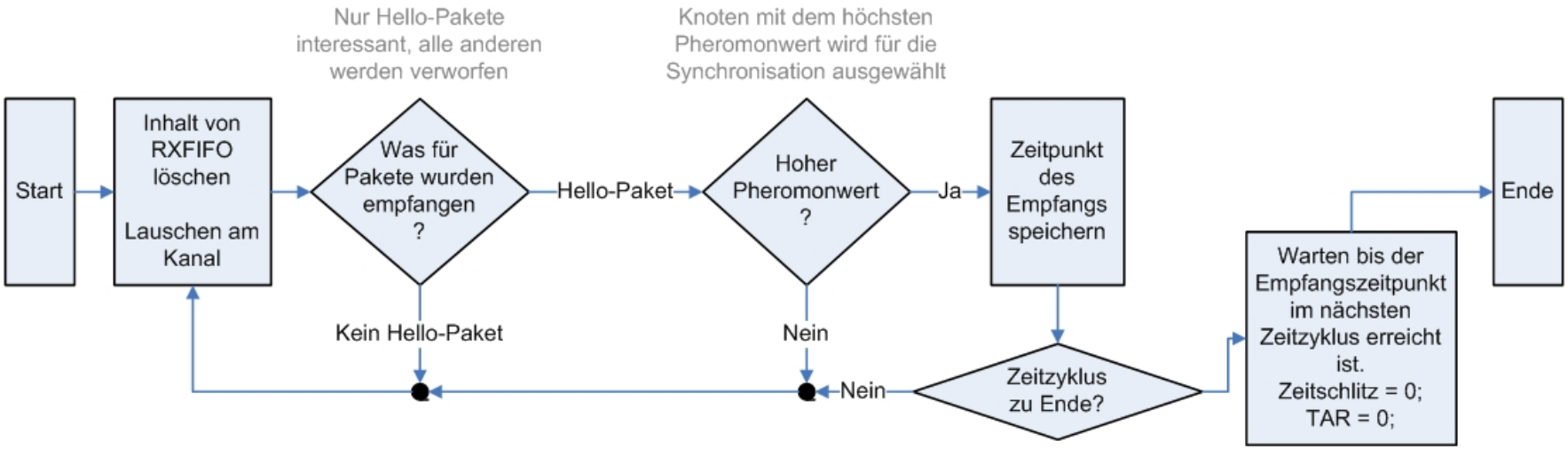


Abbildung 5.2: Flussdiagramm zur Synchronisation

```
1 char synchronise(void)
2 {
3   char nachbarGefunden = Falsch, syncSchlitz = 0, empfangsZeitschlitz;
4   char gesamt, type, paketEmpfangen = Falsch, hoechstesPheromone = 0;
5   unsigned int timer, syncZeit, empfangsZeit;
6
7   // Waehrend der Synchronisationsphase wird MSP NICHT in Ruhezustand versetzt
8   synchronisationsPhase = Wahr; // Synchronisationsphase startet
9   empfangenesPheromone = 0;
10
11  CC_Command(CC_SFLUSHRX);        // Inhalt von RXFIFO loeschen
12  CC_Command(CC_SFLUSHRX);        // Inhalt von RXFIFO loeschen
13
14  CC_Command(CC_SRXON);          // SRXON Receiver einschalten
15
16  gesamt = MaximaleAnzahlZeitschlitze + 1;  // Synchronisationsphase dauert einen Zyklus
17  syncSchlitzInterrupt = 0;
18  while(syncSchlitzInterrupt < gesamt)
19  {
20      paketEmpfangen = Falsch;
21      timer = 5000;
22      // Warten bis FIFOP auf High oder timer abgelaufen ist
23      while((((!(P1IN & BIT4)) || (!(P1IN & BIT5))) && timer) { timer--; }
24
```

```
25      if(timer > 0)
26      {
27        empfangsZeit = TAR;                    // Empfangszeit aus Timer_A auslesen
28        paketEmpfangen = Wahr;
29        empfangsZeitschlitz = zeitschlitz; // Empfangszeitschlitz speichern
30        SetCS(0);
31        SpiTransmit(0x7F);                     // RXFifo adressieren
32      }
33
34      while(paketEmpfangen && (P1IN & BIT5))  // Paket aus RXFIFO auslesen
35      {
36          SpiTransmit(0xFF);          // length
37          SpiTransmit(0xFF);          // Frame Control Field
38          SpiTransmit(0xFF);          // Frame Control Field
39          SpiTransmit(0xFF);          // Paket Nummer
40          SpiTransmit(0xFF);          // PanId
41          SpiTransmit(0xFF);          // PanId
42          SpiTransmit(0xFF);          // Adresse
43          SpiTransmit(0xFF);          // Adresse
44          SpiTransmit(0xFF);          // PanId Absender
45          SpiTransmit(0xFF);          // PanId Absender
46          SpiTransmit(0xFF);          // Adresse Absender
47          SpiTransmit(0xFF);          // Adresse Absender
48          type = SpiTransmit(0xFF); // Paket Type
```

```
49
50          if(type == HelloPaket)              // Wenn es sich um ein Hello Paket handelt
51          {
52            empfangenesPheromone = SpiTransmit(0xFF);             // Pheromone
53            SpiTransmit(0xFF);                                    // RSSI
54            SetCS(1);
55          } else
56          {
57            SetCS(1);
58            CC_Command(CC_SFLUSHRX);
59            CC_Command(CC_SFLUSHRX);
60          }
61
62          if(empfangenesPheromone > hoechstesPheromone) // Wenn hoechstes Pheromon
63          {
64            hoechstesPheromone = empfangenesPheromone;
65            syncSchlitz = empfangsZeitschlitz;
66            syncZeit = empfangsZeit;
67            nachbarGefunden = Wahr;
68            if((syncSchlitz == 0xFF) || (syncSchlitz == 0x00)) { nachbarGefunden = Falsch; }
69          }
70          CC_Command(CC_SFLUSHRX);
71          CC_Command(CC_SFLUSHRX);
72      }
```

```
73    }
74    if(nachbarGefunden == Wahr)
75    {
76      while(syncSchlitz != zeitschlitz);
77      while(syncZeit != TAR);
  // Auf richtigen Timerwert warten
78      TAR = 0x0000;                          // Timer mit 0 initialisieren
79      zeitschlitz = 1;                       // Zeitschlitz auf 1 setzen
80      synchronisationsPhase = Falsch;        // Synchronisationsphase vorbei
81      return 1;
82    } else
83    {
84      synchronisationsPhase = Falsch;        // Synchronisationsphase vorbei
85      return 0;
86    }
87 }
```

5.3 Initialisierung der Hardware

5.3.1 Initialisierung der SPI-Schnittstelle

Die Kommunikation zwischen Mikrokontroller und Transceiver-Chip findet über die SPI-Schnittstelle (englisch: Serial Peripheral Interface) statt. Hierbei handelt es sich um ein serielles Bussystem welches von der Firma Motorola entwickelt wurde. Für genauere Informationen zum Thema SPI siehe [17]. Die im folgenden Codebeispiel benutzten Register sowie deren Funktionen werden in [5] genauer beschrieben.

```
1  void SpiInit(void)
2  {
3    U0CTL  =  CHAR + SYNC + MM + SWRST; // 8-bit Transfer, SPI-Modus, Adressbit, SW-Reset
4    U0TCTL |= CKPH + SSEL1 + STC;       // Transfer bei steigender Flanke, ACLK
5    U0BR0  = 0x02;                      // SPICLK Setzen der Baudrate
6    U0BR1  = 0;                         // Baudraten-Controll-Register2 wird nicht benoetigt
7    U0MCTL = 0;                         // Es wrid keine Modulation der Baudrate benoetigt
8    P3SEL |= BIT1+BIT2+BIT3;            // Auswahl der peripheren Funktionalitaet (SPI)
9    P3DIR |= BIT0+BIT1+BIT3+BIT4;       // Als Ausgang fefinieren (SIMO,CLK,CSn, UTX)
10   P3DIR &= ~BIT2;                     // Als Eingang definierent (SOMI)
11   P1DIR &= ~BIT4;                     // FIFOP-Pin als Eingang definieren
12   P1DIR &= ~BIT5;                     // FIFO-Pin als Eingang definieren
13   P1DIR &= ~BIT2;                     // SFD-Pin als Eingang definieren
14   ME1   |= USPIE0;                    // Module aktivieren
15   U0CTL &= ~SWRST;                    // Ende RESET
16 }
```

5.3.2 Initalisierung des CC2420

```
1  void InitC2420(void)
2  {
3    P1DIR  = BIT7 + BIT6;    // Pin P1.6 und P1.7 sind Ausgaenge (Voltage Regulator und Reset)
4    P1OUT |= BIT7 + BIT6;    // Gesetzt:  Voltage Regulator ein und Reset
5    P1OUT &= ~BIT6;          // Reset auf Low
6    P1OUT |=  BIT6;          // Reset auf High
7
8    CC_Command(CC_SXOSCON);       // Einschalten des Oscillators
9    wait_for_Osc();               // Warten bis sich Oscillator stabilisiert hat
10
11   CC_RAM_Write_2B(0x16A, myAddress1, myAddress2);          // Adresse festlegen
12   CC_RAM_Write_2B(0x168, myPANAddress1, myPANAddress2);    // Pan-Adresse festlegen
13
14   CC_Reg_Write(CC_SECCTRL0, 0x01, 0xC4);    // Security Control deaktivieren
15   CC_Reg_Write(CC_MDMCTRL0, 0x0A, 0xE2);    // Adressaufloesung einschalten
16
17   // Standardeinstellung des Modem Control Registers herstellen
18   CC_Reg_Write(CC_MDMCTRL1, 0x05, 0x00);
19
20   // Standardeinstellung des Input / Output Control Registers herstellen
21   CC_Reg_Write(CC_IOCFG0, 0x00, 0x40);
22 }
```

5.3.3 Aktivierung von Interrupts

```
1  void Init_Osc()
2  {
3    // Timer_A Kontrollregister (TACTL): ACLK, continuous mode, interrupts einschalten
4    // Takt durch 8 dividieren (ID0 + ID1)
5    TACTL = TASSEL_1 + MC_1 + TAIE + ID0 + ID1;
6    _BIS_SR(GIE);  // Interrupts erlauben
7    __enable_interrupt();
8    // Wenn nachfolgende Zeile benutzt wird kann der Takt (ACLK) nochmals durch 8 dividiert werden
9    // BCSCTL1 += DIVA0 + DIVA1;
10
11   // Bei Erreichen dieses Wertes erfolgt der Ueberlauf des Timers
12   TACCR0 = 0x1FFF;
13 }
```

5.3.4 Funktion des Interrupts

```
1 // Timer_A3 Interrupt Vector (TAIV)
2 #pragma vector=TIMERA1_VECTOR
3 __interrupt void Timer_A(void)
4 {
5   switch( TAIV )
6   {
7     case  2: break;
8     case  4: break;
9     case 10: _BIC_SR_IRQ(LPM3_bits);       // Beenden des Ruhezustandes (LPM3)
10
11       // Wenn ein kompletter Zeitzyklus um ist
12       // wird die Variable zeitschlitz wieder zurueck auf 0 gesetzt
13       if(zeitschlitz == MaximaleAnzahlZeitschlitze) { zeitschlitz = 0; }
14
15       if(zeitschlitz == 0)
16       {
17           alive = Wahr;
18           InitC2420();     // Initialisieren des CC2420
19           zeitschlitz++;  // Erhoehung der Variable zeitschlitz
20       }
21       else
22       {
23           zeitschlitz++;  // Erhoehung der Variable zeitschlitz
```

```
24          }
25          break;
26      }
27  }
```

5.4 Benutzung der SPI-Schnittstelle

Die folgende Funktion ermöglicht die Kommunikation zwischen Mikrokontroller (MSP430) und Transceiver-Chip (CC2420). Der Datenaustausch kann in beide Richtungen stattfinden, wobei zu Beginn jeder Kommunikation immer ein Adressbyte, welches das zu lesende oder zu beschreibende Register auswählt, in den CC2420 geschrieben werden muss. Beim Schreibvorgang sollten der Funktion als Parameter die gewünschten Daten übergeben werden. Beim Lesevorgang wird der Standardwert 0xFF übergeben, welcher vom CC2420 ignoriert wird. Vor Beginn und nach Beendigung jedes Datenaustausches muss die Chipselct-Leitung aktiviert beziehungsweise deaktiviert werden.

```
1 char SpiTransmit(char a)          //Sendet und Empfaengt 1 Byte ueber SPI
2 {
3   while (!UTXIFG0 & IFG1);        //Warten bis der Sendepuffer frei ist (UTXIFG0.IFG1 = 1)
4   U0TXBUF = a;                    //Byte a ueber SPI senden
5   Delay(50);
6   return U0RXBUF;                 //Gelesenes Byte zurueckgeben
7 }
```

5.5 Serielle Schnittstelle

5.5.1 Initialisierung

```
1  void InitRS232(void)
2  {
3    P3SEL |= BIT4 + BIT5; // P3.4 und P3.5 als USART0 TXD/RXD
4    ME1 |= UTXE0 + URXE0; // TX- und RX-modul anschalten
5    UCTL0 = CHAR;         // 8 Datenbits, 1 Stopbit, Kein Paritaetsbit (8N1)
6    UTCTL0 = SSEL0;       // ACLK als UCLK festlegen
7    UBR00 = 0x03;         // 9600 baud aus 32.768 kHz erzeugen
8    UBR10 = 0x00;         // Keine weitere Teilung erforderlich
9    UMCTL0 = 0x4A;        // Korrektur der Division
10   UCTL0 &= ~SWRST;      // USART freigeben
11 }
```

5.5.2 Ausgabe

Die folgende Funktion ermöglicht die Ausgabe einer beliebigen Zeichenkette auf einem Hyper-Terminal. Der auszugebende Text ist der Funktion als Parameter zu übergeben.

```
1  void WriteRS232(char string[])
2  {
3    char i=0;
4    InitRS232();
5    while(string[i] != '\0')
6    {
7       TXBUF0 = string[i];
8       Delay(200);
9       i++;
10   }
11   SpiInit();
12 }
```

6

Zusammenfassung und Ausblick

Ziel dieser Diplomarbeit war die Entwicklung der Hardware eines Sensorknotens sowie die Erstellung eines Routingprotokolls. Dieses Ziel wurde erreicht da ein funktionierender Schaltplan, das zugehörige Layout sowie die Betriebssoftware von mir entwickelt wurden. Die aus dem Layout resultierenden Leiterplatten wurden mit den entsprechenden Bauteilen bestückt und mit der Software in Betrieb genommen. Die Kommunikation zwischen den einzelnen Sensorknoten (Abbildung 6.1), sowie die selbständige Synchronisation funktionieren. Bei einem Test mit vier Modulen wurden alle Pakete korrekt an der Basisstation empfangen. Zusätzlich zu den Temperaturmessungen mittels des Mikrokontrollers wurde ein Modul auch mit einem digitalen Feuchte-Sensor versehen und eine Methode zum Auslesen der Messdaten entwickelt.

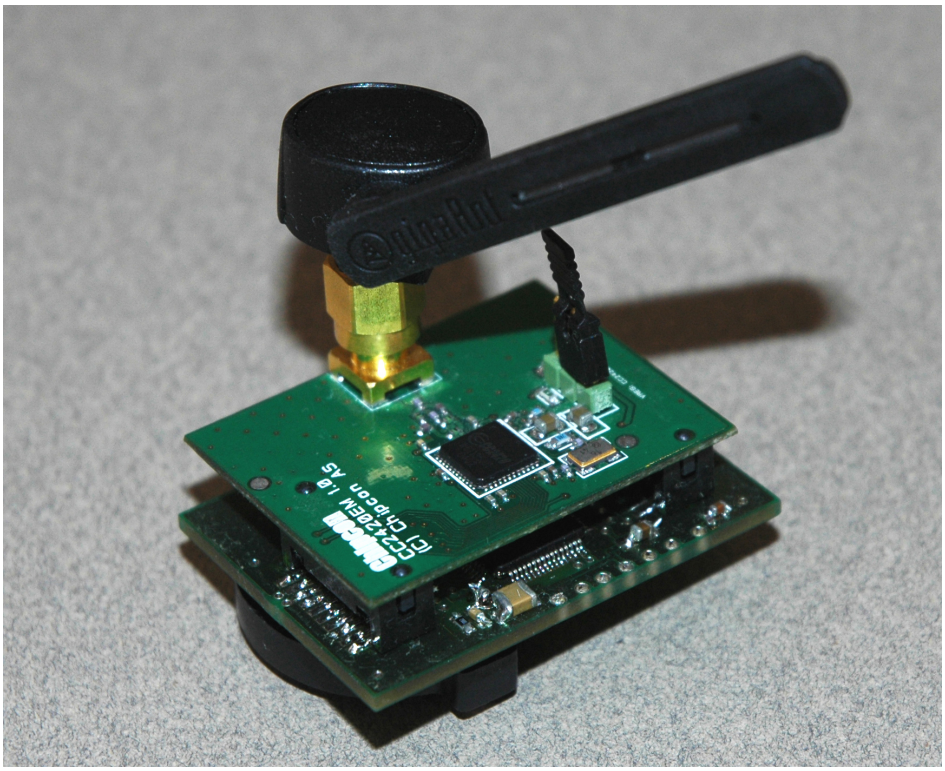


Abbildung 6.1: Sensorknoten

6.1 Ausblick

6.1.1 Kommunikation in zwei Richtungen

Der in [1] theoretisch erforschte Routingalgorithmus wurde geschaffen um Daten von einem Knoten zu einer Basis zu senden. Bei manchen Anwendungen wäre es jedoch durchaus sinnvoll in die umgekehrte Richtung zu kommunizieren. Beispielsweise wenn Messdaten nur auf Anfrage benötigt werden. Dies hätte eine sehr große Entlastung der einzelnen Knoten zu Folge, da diese nur mehr Daten senden müssten wenn diese auch wirklich von jemand benötigt werden. Des Weiteren könnte man Einstellungen an einzelnen Knoten aus der Ferne vornehmen. Ein Lösungsansatz für dieses Problem wäre, dass jeder Knoten der ein Paket zur Basis weitersendet, seine Adresse im Header vermerkt. Die Basisstation könnte sich anhand dieser Informationen eine Routingtabelle anlegen um so auch Daten an einen bestimmten Knoten im Netz zu senden. Der Nachteil in diesem Ansatz besteht darin, dass die Größe der Pakete durch die zusätzlichen Adressinformationen wachsen würde.

6.1.2 Basisstation

Bisher wurde die Basisstation nur durch Hilfe eines einfachen Sensorknotens realisiert. Die Basis sollte jedoch über einen äußerst großen Datenspeicher verfügen. Des Weiteren benötigt man eine Software, die die empfangenen Daten ausliest und in einer praktischen und übersichtlichen Form darstellt.

6.1.3 Gehäuse

Sensoren, die in freier Natur verwendet werden, sollten mit einem wasserdichten Gehäuse versehen werden. Je nach Anwendung des Netzes können auch die Anforderungen an das Gehäuse variieren.

6.1.4 Miniaturisierung

Solange sich der Transceiver-Chip und die Antenne auf dem CC2420 Evaluation Module befinden, können die Sensorknoten nicht merklich verkleinert werden. Daher sollten diese beiden Komponenten bei dem nächsten Prototypen direkt auf der Leiterplatte integriert werden.

6.1.5 Technische Daten eines Sensorknotens

Die in dieser Arbeit beschriebenen Sensorknoten weisen eine Größe von 14,08 cm^2 auf. Die Höhe beträgt 4,5 cm (inklusive SMA-Buchse). Die Spannungsversorgung von 3V erfolgt durch eine Lithium-Knopfzelle (CR2450) welche einen Durchmesser von 2,25 cm, eine Höhe von 5 mm und eine Kapazität von 560 mAh aufweist.

Bei einem Versuch wurde die Stromaufnahme eines Sensorknotens während des Aktiven Zustandes sowie in der Ruhephase gemessen.

- Ruhephase: 0.001 mA
- Aktive Phase: 19.900 mA

Um ein einfaches Anbringen der verschiedendsten Sensoren (Temperatur, Feuchte, ...) zu ermöglichen, wurden verschiedene Sensoranschlüsse (Analog und Digital) an den Rand der Leiterplatte geführt (Abbildung: 6.3).

Beispiel

Im folgenden Beispiel wird die Lebensdauer eines Sensorknotens errechnet. Wir nehmen an, dass der aktive Zustand 2 Sekunden und die Ruhephase 15 Minuten dauern (Abbildung 6.2). Diese Zeiten können je nach Anwendung des Sensornetzes variieren.

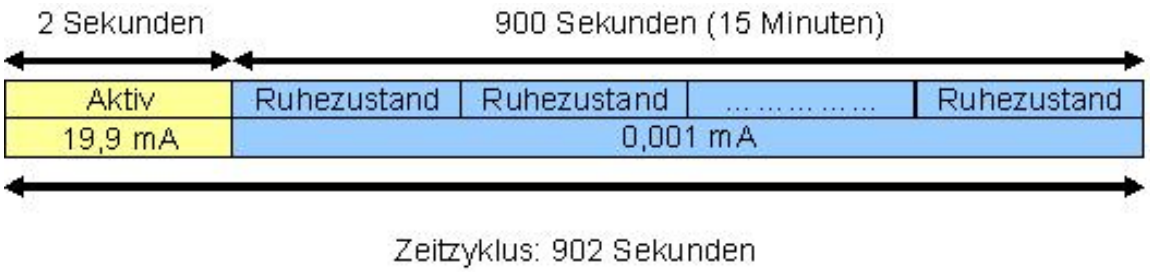


Abbildung 6.2: Beispiel: Zeitzyklus

Die benötigte Kapazität Q_{b} für einen Zeitzyklus errechnet sich wie folgt:

$$Q_{\mathrm{b}} = 2s * 19,9mA + 900s * 0,001mA = 40,7mAs$$

Die Kapazität der Batterie beträgt 560 mAh. Es wird angenommen, dass die Batterie nach 70%iger Entladung nicht mehr genügend Versorgungsspannung liefern kann. Die Nutzkapazität (Q_{n}) ergibt sich aus:

$$Q_{\mathrm{n}} = 0,7 * 560mAh = 392mAh = 1.411.200mAs$$

Dividiert man nun Q_{n} durch Q_{b} so erhält man die Anzahl (A) der Zeitzyklen, die mit der zur Verfügung stehenden Kapazität ausführbar sind:

$$A = Q_{\mathrm{n}}/Q_{\mathrm{b}} = 1.411.200mAs/40,7mAs \approx 34.673,22$$

Multipliziert man die A mit der Dauer eines Zeitzykluses, erhält man die Lebensdauer (L) eines Sensorknotens:

$$L = 34.673,22 * 902s \approx 31.275.243,24s \approx 8.687,57h \approx 361,98Tage$$

Die Zeitspanne zwischen der Aktivierung des Sensorknotens und dem ersten Batteriewechsel beträgt also ca. 362 Tage.

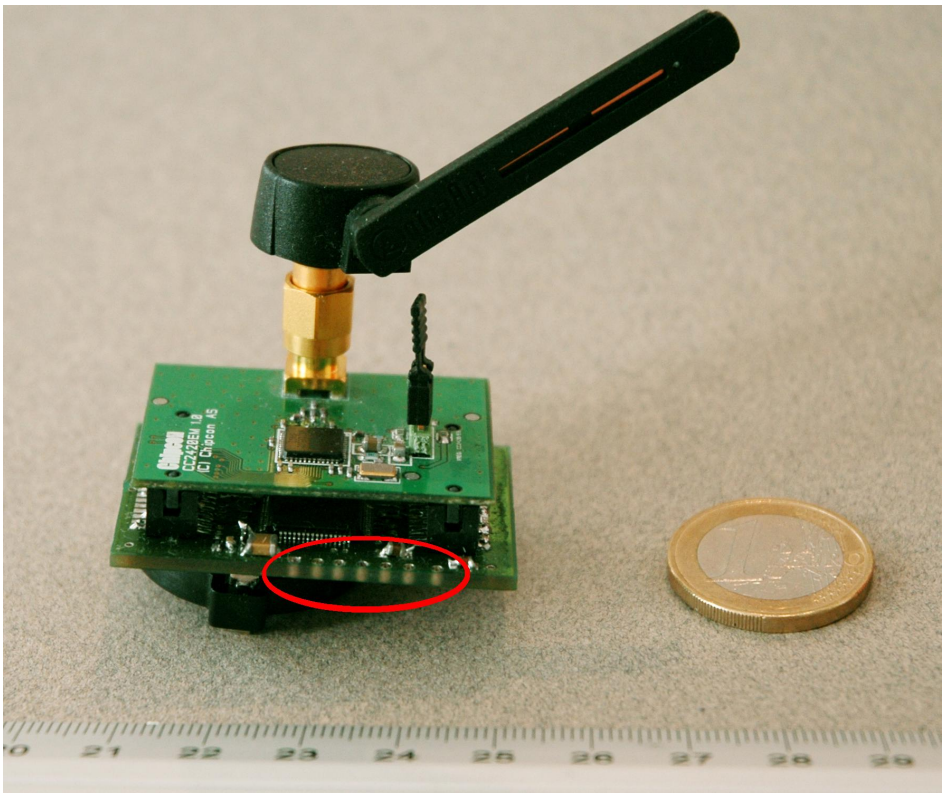


Abbildung 6.3: Anschlüsse eines Sensorknoten

Abbildungsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-to-Digital-Converter
CAD	Computer Aided Design
CCA	Clear Channel Assessment
CLK	Clock Signal
CPU	Central Processing Unit
CS	Chip Select
DSDV	Destination Sequenced Distance Vector
DSN	Drahtloses Sensornetzwerk
DSR	Dynamic Source Routing
DSSS	Direct Sequence Spread Spectrum
FCF	Frame Control Field
FCS	Frame Check Sequence
FFD	Full Function Device
FIFO	First In First Out
FPU	Floating Point Unit
GHz	Gigahertz
IC	Integrated Circuit
ID	Identifikationsnummer
IEEE	Institute of Electrical and Electronics Engineers
ISM	Industrial, Scientific and Medical Band
ISR	Interrupt Servic Routine
JTAG	Joint Test Action Group
LPM	Low Power Mode
MCU	Mikrocontroller Unit
MSB	Most Significant Bit
MSP	Mixed Signal Processor
NTC	Negative Temperature Coefficient
PAN	Personal Area Network

PC ..... Personal Computer
PQFP ..... Plastic Quad Flat Pack
RAM ..... Random Access Memory
RFD ..... Reduced Function Device
RISC ..... Reduced Instruction Set Computing
RS232 ..... Recommended Standard 232
RSSI ..... Receive Signal Strength Indicator
RX ..... Receive
SCLK ..... Serial Clock
SFD ..... Start of Frame Delimiter
SI ..... Serial In
SMA ..... Sub-Miniature-A
SMD ..... Surface Mounted Device
SO ..... Serial Out
Spi ..... Serial Peripheral Interface
TACCR ..... Timer-A Capture Compare Register
TX ..... Transmit
VREG_EN ... Voltage Regulator Enable

Literaturverzeichnis

[1] Andreas Kos, Management Of Selforganising Wireless Sensor Networks, Diplomarbeit

[2] DIGITAL COMMUNICATIONS, Ian A. Glover, Peter M. Grant, PEARSON Education, Seite 946

[3] Matthias Sturm, Mikrocontrollertechnik, Fachbuchverlag Leipzig im Carl Hanser Verlag, 2006, Seite 86

[4] Lutz Bierl, Das große MSP430 Praxisbuch, Franzis Verlag GmbH, 2004, Seite 154

[5] MSP430x1xx Family User's Guide, http://focus.ti.com/lit/ug/slau049f/slau049f.pdf 30.Juli 2007

[6] MSP430 Data Sheet, http://focus.ti.com/lit/ds/symlink/msp430f149.pdf, 30.Juli 2007

[7] Produkt Spezifikation der Titanis 2.4 GHz Swivel SMA Antenne, www.antenova.com/?id=753&pv=1&lang=1, 30.Juli 2007

[8] Introduction To Ad Hoc Networks and Routing Protocolls, Karthik Samudram Jayaraman, http://www.cs.wmich.edu/wsn/cs691_sp03/adhocrouting.ppt, 12.August 2007

[9] http://www.chemgapedia.de/vsengine/popup/vsc/de/glossar-/s/sc/schwingquarz.glos.html, 17.Juni.2007

[10] User Manual CC2420DK Development Kit, http://focus.ti.com/lit/ug/swru045/swru045.pdf, 30.Juli 2007

[11] 2.4 GHz IEEE 802.15.4 / ZigBee-Ready RF Transceiver, http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 30.Juli 2007

[12] Datasheet Humidity and Temperature Sensor,
http://www.sensirion.com/en/pdf/product_information/Data_Sheet-_humidity_sensor_SHT1x_SHT7x_E.pdf, 30.Juli 2007

[13] IEEE 802.15.4 Standard for Information technology,
http://standards.ieee.org/getieee802/download/802.15.4-2003.pdf, 30.Juli 2007

[14] Direct Sequence Spread Spectrum,
http://de.wikipedia.org/wiki/DSSS, 12.August 2007

[15] http://www.hometoys.com/htinews/aug04/articles/chipcon/zigbee.htm,
28.August 2007

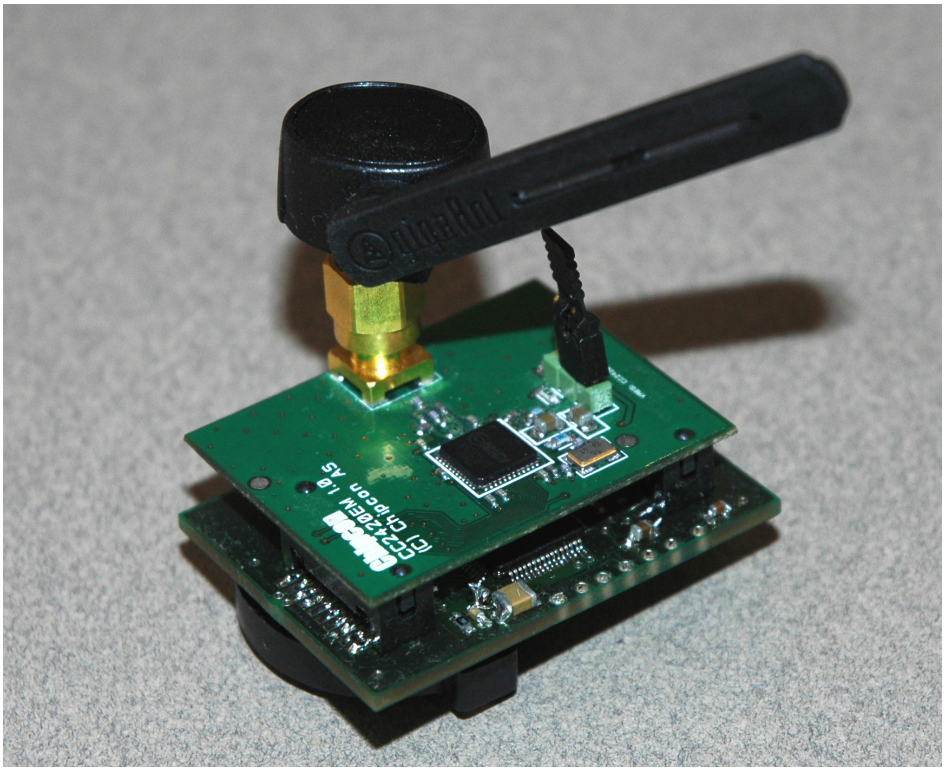
[16] http://elmicro.com/de/armjtag.html, 28.August 2007

[17] http://www.mct.de/faq/spi.html, 31.Juli 2007

[18] http://de.wikipedia.org/wiki/Routing, 18.Juni 2007

Anhang

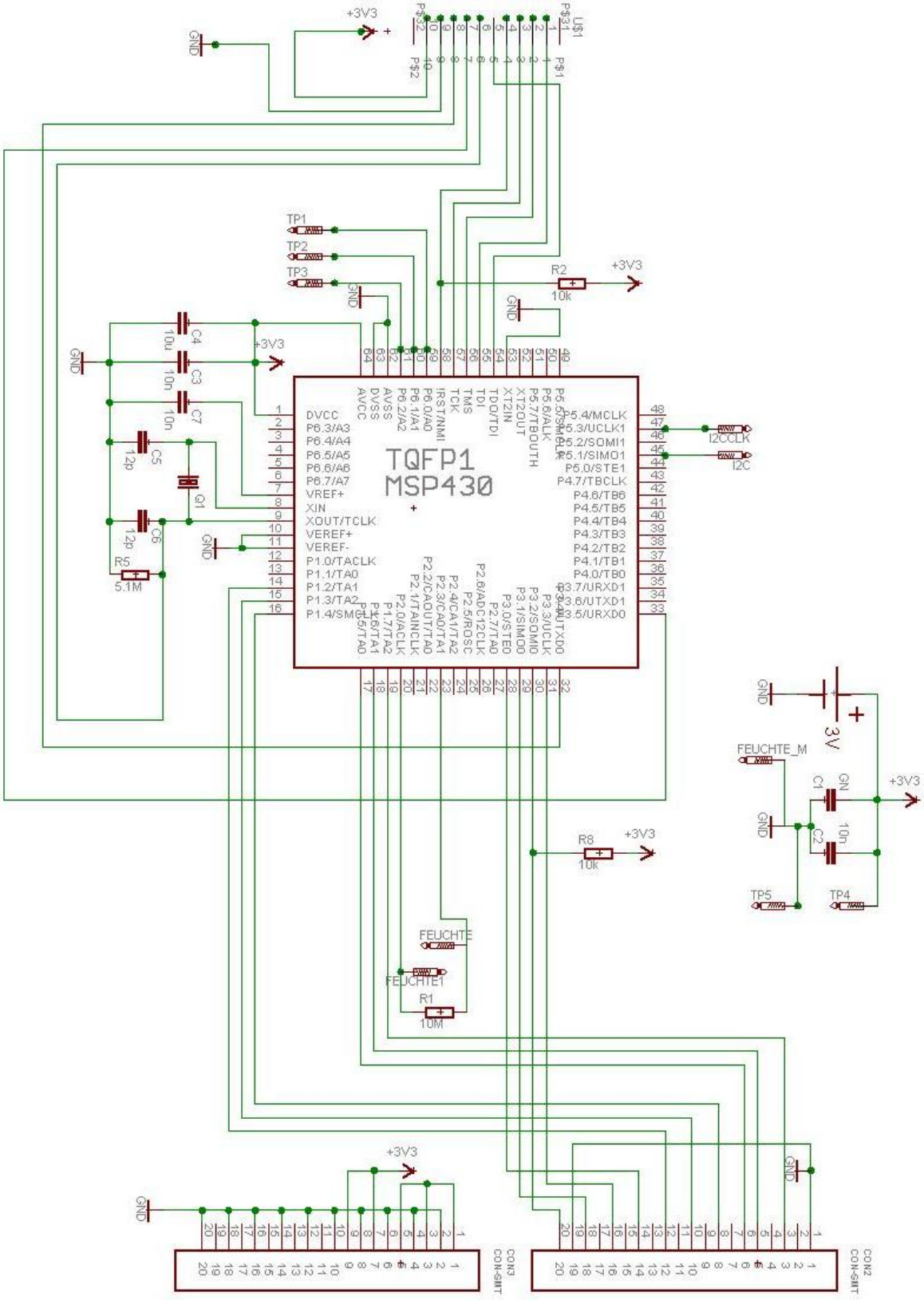
A Foto: Sensorknoten



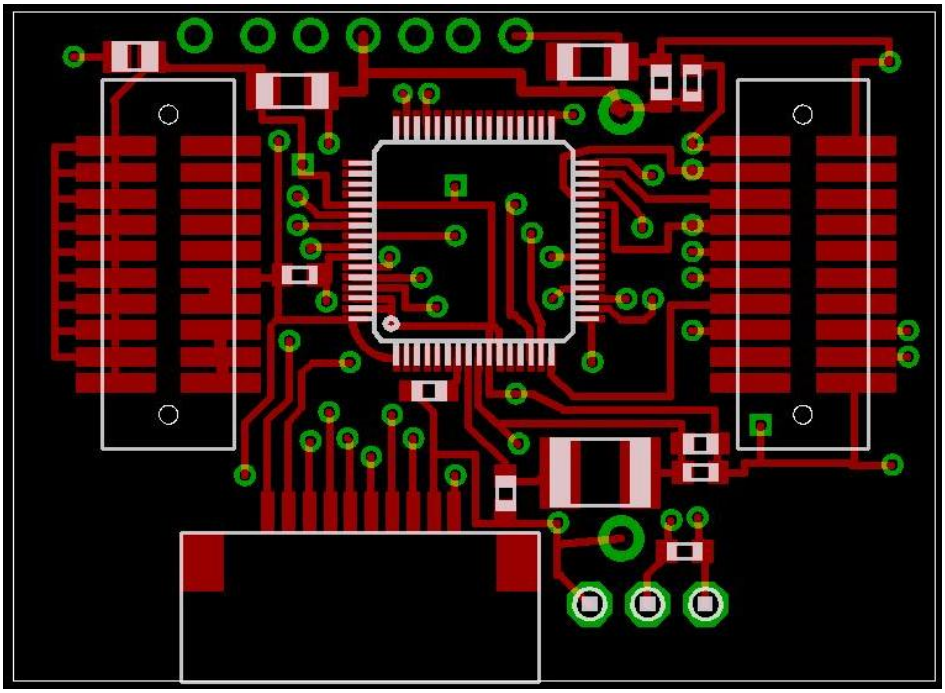
B Stückliste

Bezeichnung	**Anz.**	**Type**	**Firma**	**Best. Nr.**
Mikrokontroller	1	MSP430F149	Farnell	9853197
Transceiver-Modul	1	C2420EMK	Farnell	1248474
Uhrenquarz	1	32.768kHz	Farnell	1216227
Stiftleiste 1,27 mm	2	20 Pol	Farnell	1099539
Steckverbinder	1	Molex-Socket	Farnell	978-6120
Knopfzelle	1	CR2450	Farnell	1293463
Batteriehalterung	1	Batteriehalterung	RS-Components	430-681

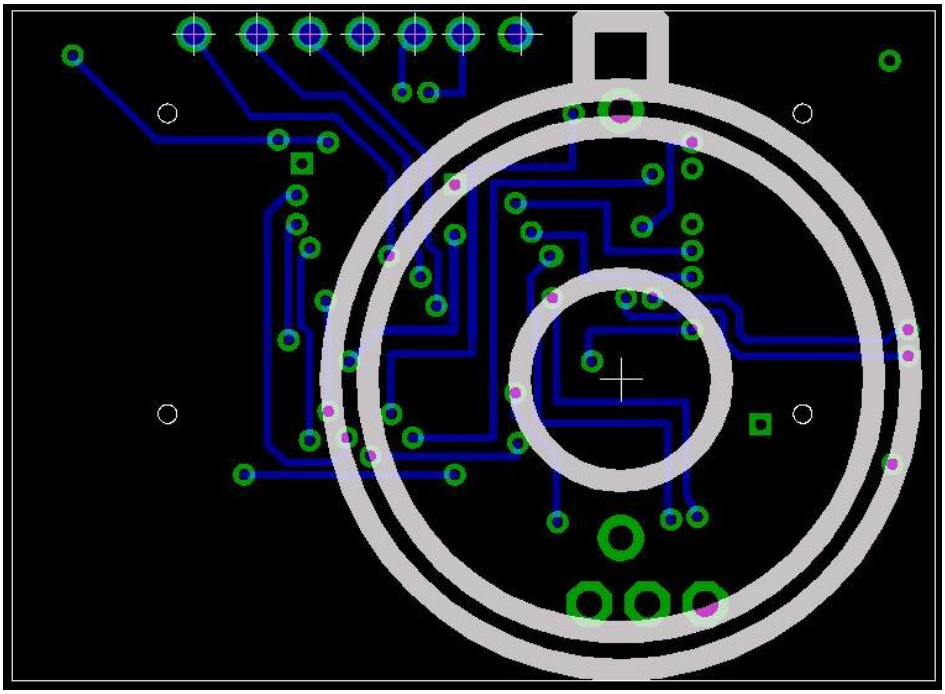
C Schaltplan eines Sensorknotens



D Layout eines Sensorknotens (Oberseite)



E Layout eines Sensorknotens (Unterseite)



F Lebenslauf

Persönliche Daten

Name	Christoph Mayerhofer
Adresse	Pfeilgasse 47-49/3/15; 1080 Wien,
Telefon Nr.	0650 94 04 624
E-Mail	chri.stop@gmx.at
Geboren am	03.05.1981
Familienstand	ledig
Staatsbürgerschaft	Österreich

Schulische Ausbildung

1987-1991	Volksschule Gilgegasse 12, 1090 Wien
1991-1995	BRG Feldgasse 6-8, 1080 Wien
1995-1996	BORG Hegelgasse 14, 1014 Wien
1996-2000	Berufsschule für Optiker, Apollogasse 1, 1070 Wien
2001-2003	Vorbereitungslehrgang für Berufsreifeprüfung Volkshochschule Floridsdorf, Pitkagasse 3, 1210 Wien
2003-2007	FH Studiengang für Computersimulation Fachhochschule St. Pölten, Herzogenburger Straße 68, 3100 St. Pölten

Berufsausbildung

02.09.1996-02.03.2000	Optikerlehre bei Optiker Prior GmbH, Auhofstraße 142, 1130 Wien
02.03.2001	Lehrabschlussprüfung Optiker

Präsenzdienst

03.04.2000-01.12.2000	Bundesheer: Maria-Theresien-Kaserne, Am Fasangarten 2, 1130 Wien