Diplomarbeit

Frank Jakubowsky

MOTEs – Evaluation und exemplarische Anwendung einer neuen Technologie

Fachbereich Elektrotechnik und Informatik
Department of Electrical Engineering and Computer Science

Frank Jakubowsky

MOTEs – Evaluation und exemplarische Anwendung einer neuen Technologie

Diplomarbeit eingereicht im Rahmen der Diplomprüfung
im Studiengang Technische Informatik

am Fachbereich Elektrotechnik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Gunter Klemke
Zweitgutachter : Prof. Dr. Kai von Luck

Abgegeben am 13. April 2005

Frank Jakubowsky

Thema der Diplomarbeit

MOTEs – Evaluation und exemplarische Anwendung einer neuen Technologie

Stichworte

MOTE, Smart Dust, Wireless Sensor Network, Tiny OS

Kurzzusammenfassung

Diese Arbeit umfasst die Ausarbeitung der Technik von drahtlosen Sensor-Netzwerken. Dabei wird die Hardware einer zur Verfügung gestellten Produkt-Familie allgemein, und an ausgesuchten Exemplaren konkret beschrieben.
Weiterhin beinhaltet sie die Beschreibung des, für diese Art von Geräten entwickelten, Betriebssystems Tiny OS und dessen integrierten Compiler NesC.

Die Einsatzmöglichkeit dieser Systeme wird anhand der Implementation einer Analyse-Applikation für den Laufsport dargestellt. Dabei werden die Sensordaten für Pulsfrequenz und Schrittfrequenz in einem tragbaren Gerät gemessen, gespeichert und per Funk zu einer, auf einem PC laufenden, Auswertesoftware gesendet, und dort grafisch dargestellt.

Frank Jakubowsky

Title of the paper

MOTEs - Evaluation and exemplary application of a new technology

Keywords

MOTE, Smart Dust, Wireless Sensor Network, Tiny OS

Abstract

This work covers the elaboration of the technology of wireless sensor networks. The hardware of a provided product family is generally, and at selected samples, concretely described.
Further it contains the description of the operating system Tiny OS and its integrated compiler NesC, which was specially developed for this kind of devices.

The possibility for practical use of these systems is represented by the implementation of an analysis application for run sports. On a portable equipment, sensor data for pulse frequency and step frequency are measured, stored and then sent by radio to an evaluation software, running on a PC. There the data is graphically shown.

Inhaltsverzeichnis

Einleitung

Die Computertechnik schreitet unaufhaltsam voran, sie ist aus dem täglichen Leben heute nicht mehr wegzudenken.
Sei es im privaten Bereich als Informationsquelle und Unterhaltungsmedium, oder im geschäftlichen Bereich, wo sie uns die Arbeit erleichtert, beschleunigt, oder zum Teil erst ermöglicht.
Dabei ist nicht nur die Sammlung persönlicher Erfahrungswerte, sondern auch die Speicherung von gemessenen Werten von großer Wichtigkeit.
Da diese jederzeit, und dank der heutigen Kommunikationstechnik, auch sofort weltweit verfügbar sind, kann darauf basierend z.B. die Statik von Gebäuden, als auch die Effizienz von Verbrennungsmotoren immer weiter verbessert werden.

Benötigt werden somit Sensoren für möglichst alle physikalischen Größen, dessen Daten als Ausgangspunkt für eine Steuerung oder Regelung, oder als Erfahrungswerte für folgende Generationen dienen können.

Da bei Verwendung von vielen einzelnen Sensoren allerdings der Aufwand an Verkabelung steigt und das System somit anfällig und unübersichtlich wird, sollten diese möglichst aktiv arbeiten, und für Feldanwendungen, eine drahtlose Verbindung erlauben. Somit sind Aufbauten, wie z.B. bei der strukturalen Überwachung von Gebäuden (siehe Abbildung 1), in Zukunft vermeidbar.
Die Datenübertragung per Funk gehört heutzutage auch schon zum Standard in der Computertechnik, sodass diese auch auf kleinere Systeme übertragbar wird.

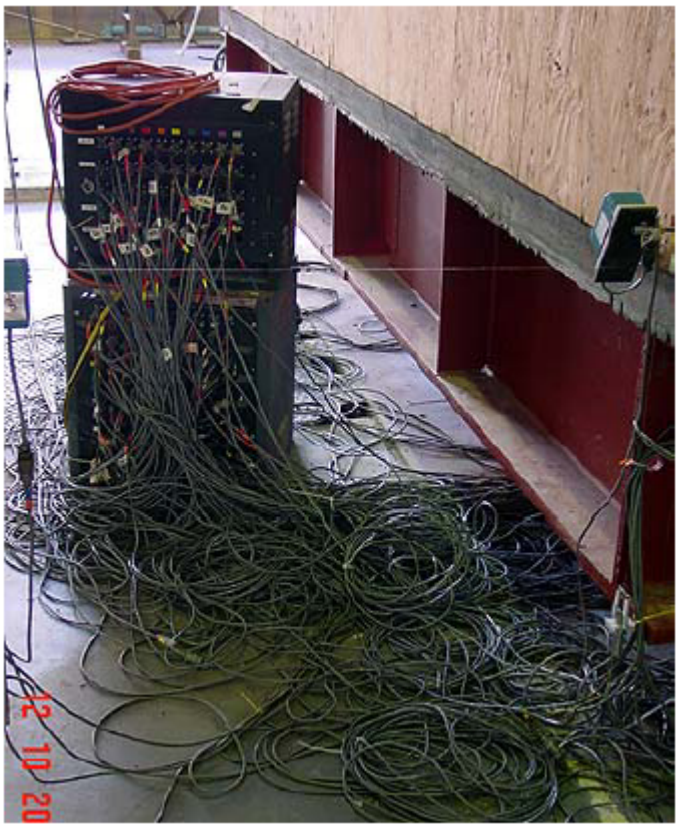


Abbildung 1 - Structural Monitoring

Entstanden sind so genannte ‚Smart Dust', welche als ‚intelligenter Staub' genau diese Aufgaben übernehmen können. Sie sind nur wenige Millimeter groß, besitzen eine Vorverarbeitung der gemessenen, meist analogen Daten, und können diese drahtlos, via Funk oder Laser zu einer Basisstation übermitteln.
Basierend auf dieser Technik wurde an der Berkeley University ein universell einsetzbares System entwickelt.

Dieses System trägt den Namen ‚MOTE', was für Remote, also entfernt steht. Gemeint sind intelligente Sensoren, welche mit einer Datenverarbeitungs- und Funkübertragungs- Einheit ausgestattet sind. Hiermit lassen sich so genannte ‚Wireless Sensor Networks' bilden.
Diese Geräte sind völlig eigenständig, einmal in einer Umgebung ausgesetzt, versorgen sie sich selbstständig über eine Batterie, sammeln Daten wie z.B. die Temperatur und Luftfeuchtigkeit, speichern diese Daten, und senden sie zu einem anderen MOTE, oder zu einer performanteren Station, wie z.B. einem PC.
Somit sind sie in der Lage, Umweltüberwachungen durchzuführen, seismische Aktivitäten an Gebäuden zu analysieren, oder durch ihre geringe Größe auch als tragbares Gerät zur Gesundheitsüberwachung einer Person zu agieren.

Ziel dieser Arbeit ist es, die Technologie der MOTEs zu beleuchten.
Nach einem kurzen Exkurs, bei der eine verwandte, aber bei weitem nicht so leistungsfähige, Technik vorgestellt wird, folgt ein Kapitel über die Hardware dieser Geräte.
Hier werden momentan verfügbare Geräte in ihre Bestandteile zerlegt und dessen Zusammenspiel erläutert.
Nachdem die Hardware bekannt ist, wird im darauf folgenden Kapitel die Software der Technologie betrachtet. Dabei werden die Konzepte des Betriebssystems, und dessen integrierten Compiler, unter dem Gesichtspunkt betrachtet, eine lauffähige Applikation erstellen zu können, und die Technik der Funk-Kommunikation zu verstehen.
Anschließend wird, mit den Kenntnissen der Hard- und Software, eine Beispiel-Applikation entwickelt, welche auch die Möglichkeit der spontanen Vernetzung dieser Geräte ausnutzt. Hier wird ein tragbares Gerät entwickelt, welches Sensordaten sammelt und, mittels der Funk-Technik, eine Verbindung zu einem PC aufbauen kann.

1. Verwandte Technologie

An dieser Stelle soll eine weitere zur Zeit existierende Technologie vorgestellt werden, da sie den MOTEs, oder auch Smart Dust, sehr ähnlich sind.
Die Rede ist von Transpondern, welche aber für andere Zwecke eingesetzt werden und somit keine Konkurrenz darstellen.

1.1. Transponder

Ein Transponder ist die Zusammensetzung von **Trans**mitter und Res**ponder**, die Geräte können also senden und empfangen.
Die Geräte sind typischerweise sehr klein und meist passiv konstruiert. Das heißt, sie besitzen keine eigene Stromversorgung, sondern werden jeweils nur für die Zeit einer Kommunikation von einem Aktiven Partner durch Induktion versorgt.
Aktive Transponder können aber auch Batterien, oder Solarzellen als Energieversorgung besitzen.

Besonders bekannt sind Transponder aus dem Bereich der Warensicherung im Kaufhaus.
Ein Transponder kann aber wesentlich mehr, als einen Alarm auszulösen.
Seit den 70er Jahren werden sie auch als zum Teil implantiertes Gerät im Bereich der Tierhaltung zur Identifikation benutzt.

Abbildung 1.1 - Transponder

Somit ist es auch möglich, Personen zu identifizieren, um z.B. bargeldlos bezahlen zu können.
Gleichzeitig kann bei einem Marathon-Lauf mit sehr vielen Teilnehmern, in Verbindung mit Computer-Technik, jeder einzelne Läufer identifiziert, sein Weg nachvollzogen und seine Zeit gemessen werden.
Mit diesen Aufgaben der Identifikation ist allerdings die Leistungsfähigkeit der Geräte nahezu ausgeschöpft.
Sie besitzen, wenn überhaupt, nur eine sehr geringe Rechenleistung und auch ihre Funk-Reichweite ist mit max. 3m sehr begrenzt.

Durch ihre niedrigen Produktionskosten sind sie für die genannten Einsatzgebiete bestens geeignet. Überall dort, wo es auf Flexibilität und Performanz ankommt, können sie den MOTEs aber nicht das Wasser reichen.

2. Hardware

Nachdem nun mögliche Anwendungen bekannt sind, soll zunächst die Hardware der dazu benötigten Komponenten beschrieben werden.
Da für diese Arbeit Geräte der Firma Crossbow (http://www.xbow.com) zur Verfügung standen, sollen diese erläutert werden.
Die Geräte sind modular aufgebaut, das bedeutet, es existieren verschiedene Verarbeitungseinheiten (Prozessor-Board) und Sensoreinheiten (Sensor-Board), welche je nach Anwendung miteinander kombiniert werden können.
Für die Entwicklung und Kommunikation mit einem PC steht eine Basis (Gateway-Board) zur Verfügung.

Dabei wird für jede Komponente (Prozessor-Board, Sensor-Board und Gateway-Board) zunächst eine Grundlage in Form einer Allgemeinen Beschreibung geschaffen.
Danach folgt jeweils eine detailierte Beschreibung ausgesuchter, und später in der eigenen Anwendung benutzten Geräte.
Weitere interessante Entwicklungen werden, der Vollständigkeit halber, nur kurz und nicht im Detail vorgestellt.
Je nach Verfügbarkeit, sind die Schaltpläne der Geräte im Anhang zu finden.

2.1. Crossbow-Kit

Crossbow bietet diese Geräte einzeln, als auch als Paket an.
Für diese Arbeit standen die Pakete „MOTE-KIT 4x0 – MICA2 Basic Kit", bestehend aus:

- 3 MICA2 Processor/Radio Boards (MPR4x0)
- 2 Multi Sensor Boards (MTS300)
- 1 Programming and Serial Interface Board (MIB510)

und
„MOTE-KIT 5x4x –MICA2/MICA2DOT Professional Kit", bestehend aus:

- 4 MICA2 Processor/Radio Boards (MPR4x0)
- 4 MICA2DOT Processor/Radio Boards (MPR5x0)
- 3 Multi Sensor Boards (MTS310)
- 2 Multi Sensor Boards (MTS510)
- 2 Prototype Boards (MDA500)
- 1 Programming and Serial Interface Board (MIB510)

zur Verfügung.
Diese Geräte werden später im Detail beschrieben.

2.2. *Processor/Radio-Board*

Die Processor-Boards bilden die Grundlage der Wireless-Sensor-Networks, da sie, wie der Name schon sagt, den Prozessor beinhalten, auf welchem später einmal die Software laufen soll.
Außerdem verfügen diese Module über einen Funk- Sender und Empfänger, welcher die drahtlose Kommunikation zwischen diesen Geräten ermöglicht.
Sie besitzen allerdings keine Sensoren, oder Aktoren, sodass für ein Wireless-Sensor-Network noch ein Kleinigkeit fehlt. Doch dazu später mehr.

2.2.1. Allgemein

Da die von der Firma Crossbow angebotenen Prozessor-Boards alle einen sehr ähnlichen Aufbau besitzen, wird dieses in den nächsten Kapiteln am speziellem Beispiel des MICA2-Boards erklärt.

Prinzipiell bestehen diese Module aus:

- einem Prozessor
- einem Funk-Transceiver
- einem Daten-Flash
- einer Silicon-Serial-Number
- und einem Interface zum Anschluss von Sensor-Boards und als Programmier-Schnittstelle

Dieses Blockdiagramm soll den Aufbau eines Processor-Boards verdeutlichen.

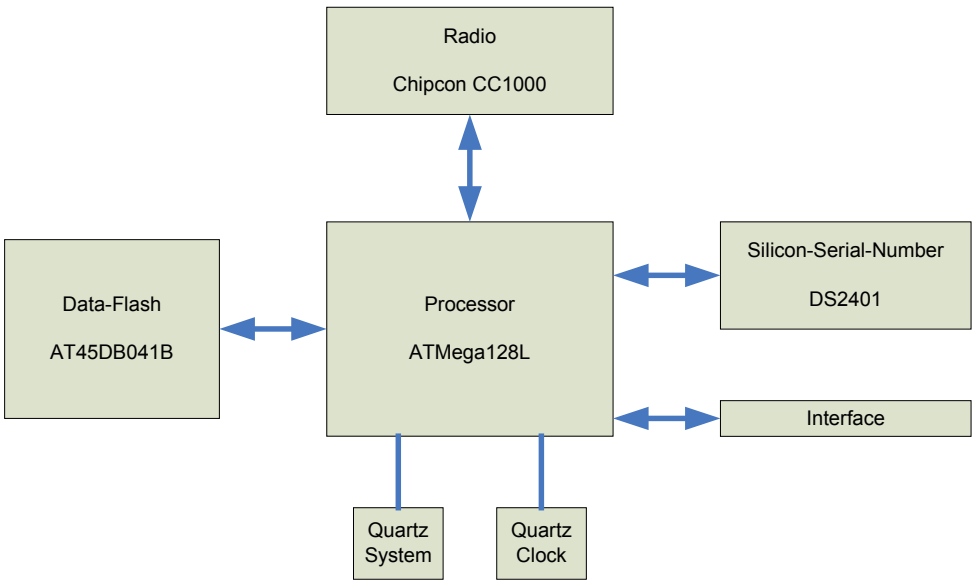


Abbildung 2.1 – Aufbau Prozessor-Board

Im Folgenden werden die einzelnen Module näher erläutert.

2.2.1.1. Prozessor

Der Prozessor bildet die Basis dieser Module und ist für die Verarbeitung der Sensor-Daten und deren Übertragung zuständig.
Bei dem Prozessor handelt es sich um einen „ATMega128L“ der Firma Atmel.
Die wichtigsten Eigenschaften sollen hier kurz erläutert werden:

- *Niedrige Versorgungsspannung*
  Der ATMega128**L** benötigt nur eine Versorgungsspannung von 2,7V – 5,5V und ermöglicht dadurch den Betrieb mit nur 2 x 1,5V Batterien.
- *Hohe Taktfrequenz*
  Durch die Taktung mit 0-8 MHz und der direkten Umsetzung in max. 8 MIPS ergibt sich eine ausreichende Performanz.
  Durch den Hardware-Multiplizierer, welcher nur 2 Zyklen benötigt, wird die Performanz weiterhin gesteigert.
- *Großer Programmspeicher*
  Der Prozessor bieten einen 128 KByte großen Flash-Programmspeicher, der dem Betriebssystem und der nötigen Applikation genügend Freiraum lässt.
  Die Wiederbeschreibbarkeit soll laut Hersteller Atmel bei 10.000 Zyklen liegen.
- *Großer Arbeitsspeicher*
  Gleichzeitig bietet der Prozessor einen für MicroController sehr großen SRam von 4 KByte, der für die meisten Applikationen, zur Datensammlung, Auswertung und Übertragung, ausreichend sein sollte.
  Weiterhin bietet der Prozessor ein
- *Externes Speicher-Interface*
  Dieses ermöglicht den Anschluss eines bis zu 64 KByte großen weiteren Speichers, ob SRam, Flash, oder andere speicherorganisierte Geräte, wie z.B. LCD-Displays.
- *EEPRom*
  Für Initial-Daten, oder andere, häufig auszulesende aber nur selten zu überschreibende Daten, bietet der Prozessor einen 4 KByte großen EEPRom.
  Die Wiederbeschreibbarkeit liegt laut Hersteller Atmel bei 100.000 Zyklen.
- *Großes Repertoire an Peripherie*
  - *SPI*
    Das SPI-Interface kann sowohl im Master- als auch im Slave-Modus arbeiten. Übertragungsraten bis zu 4Mbit/s sind möglich.
  - *Double USART*
    Der Prozessor besitzt 2 unabhängig voneinander arbeitende Universal(Synchron/Asynchron)Receiver/Transmitter. Dadurch kann er im asynchronen Modus direkt mit einem PC, im synchronen Modus mit einem weiteren SPI-Gerät kommunizieren.
    Übertragungsraten bis 1MBit/s lassen sich realisieren.
  - *I2C*
    Des I2C-Interface, aus patentrechtlichen Gründen von Atmel Two-wire Serial Interface genannt, lässt Übertragungsfrequenzen bis 400 KBit/s zu.
  - *8 Kanal 10-Bit-ADC*
    Der integrierte ADC verfügt über 8 Kanäle mit einer Auflösung bis 10 Bit. Bei maximaler Auflösung lassen sich bis zu 15 KSamples/s erreichen.

- *Mehrkanal-PWM*
  Der Prozessor bietet mehrere PWM (PulseWidthModulation) – Ausgänge mit 8-, oder 16 Bit Auflösung.
- *JTAG*
  Ein JTAG-Interface ermöglicht on-chip-debugging und die Programmierung von Programm-Speicher, EEPRom und Konfiguration
- *Digital-IO*
  Alle vorgestellten Peripherien sind optional und können je nach Anwendung aktiviert, bzw. deaktiviert werden.
  Sind die Module deaktiviert, sind die Ports als digitale IO-Ports verwendbar, und der Stromverbrauch wird minimiert.

Durch die Verwendung von 2 verschiedenen Quarzen für den Systemtakt und die „interne Uhr“, wird es dem Betriebssystem möglich, unabhängig vom Systemtakt Software-Timer mit konstanter Periode zu realisieren.
Für diesen Zweck wird ein 32,768 KHz Quarz benutzt, wie er auch in Uhren zu finden ist.

Das folgende Bild zeigt das Pinout des Prozessors. Hier wurden die wichtigsten Interfaces nochmals als Zusammenfassung gekennzeichnet.

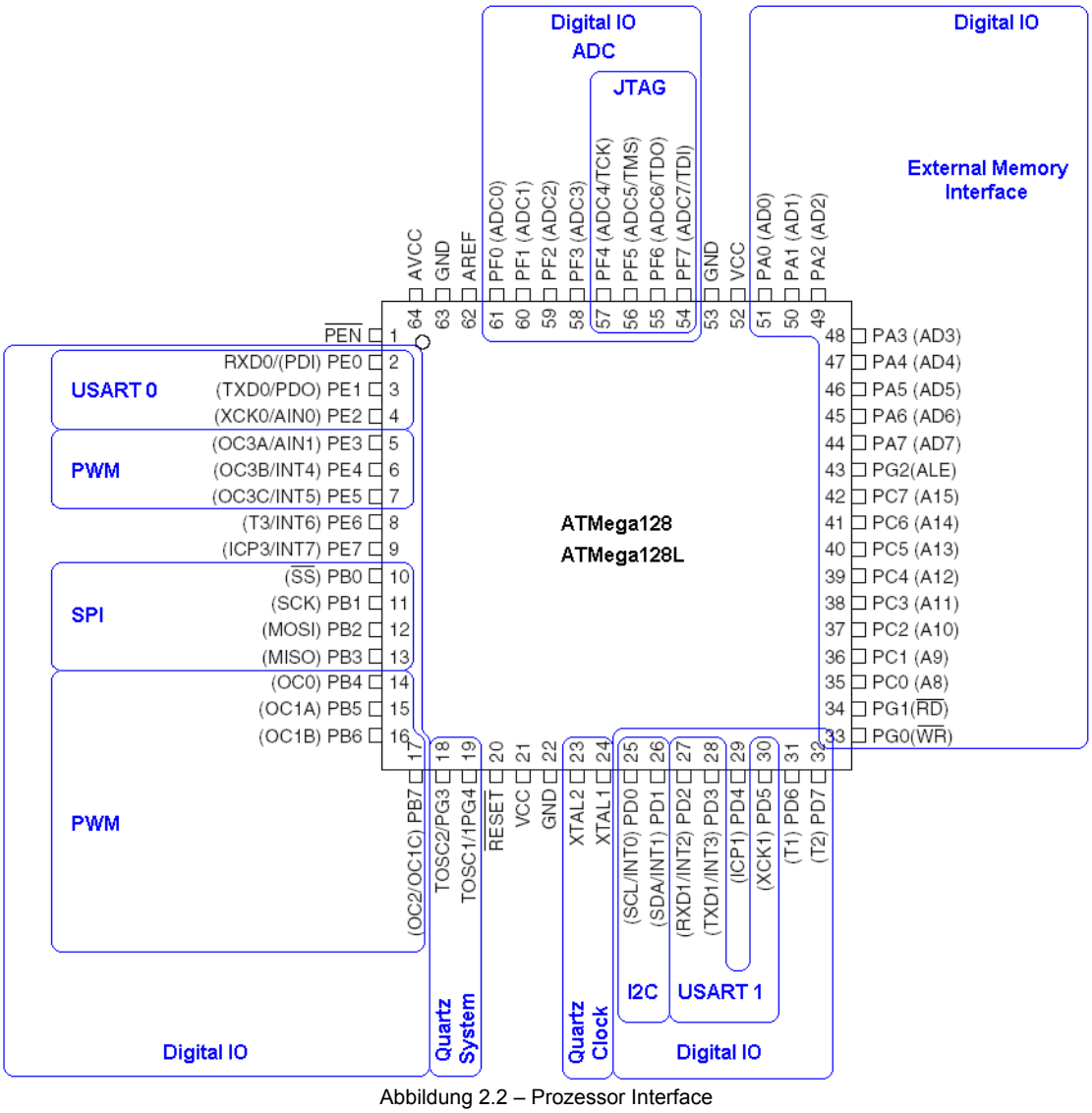


Abbildung 2.2 – Prozessor Interface

2.2.1.2. Radio

Das Radio, oder Funkmodul ist für die Datenübertragung zu anderen Geräten zuständig.
Bei dem Funk-Modul handelt es sich um einen „CC1000“ der Firma Chipcon.
Die wichtigsten Eigenschaften sollen hier kurz erläutert werden:

- *Single-Chip RF-Transceiver*
  Es sind für den Betrieb nur wenige externe Bauelemente nötig.
- *Niedrige Versorgungsspannung*
  2,1 V – 3,6 V, somit für Batteriebetrieb geeignet.
- *Programmierbare Ausgangsfrequenz*
  Geeignet für die Bänder: 315MHz, 433MHz, 868MHz, 915MHz
  Durch eine in 250Hz-Schritten einstellbare Ausgangsfrequenz kann aber der gesamte Bereich von 300 – 1000 MHz abgedeckt werden.
  Somit gleichzeitige Eignung für Frequency-Hopping (wie z.B. bei BlueTooth).
- *Programmierbare Ausgangs-Feldstärke*
  im Bereich von –20dBm bis 10dBm
- *Codierung, Datenrate*
  Es sind verschiedene Codierungen einstellbar:
  - *Synchron NRZ*
    Taktsynchrone Datenübertragung ohne Codierung mit 0,6 – 76,8 KBit/s
  - *Synchron Manchester*
    Taktsynchrone Datenübertragung mit Manchester-Codierung mit 0,3 – 38,4 KBit/s
  - *Transparent Asynchron UART*
    Datenübertragung ohne Synchronisation und Codierung mit 0,6 – 76,8 KBit/s

Das folgende Bild zeigt das Pinout des Funk-Moduls. Hier wurden die wichtigsten Interfaces gekennzeichnet.

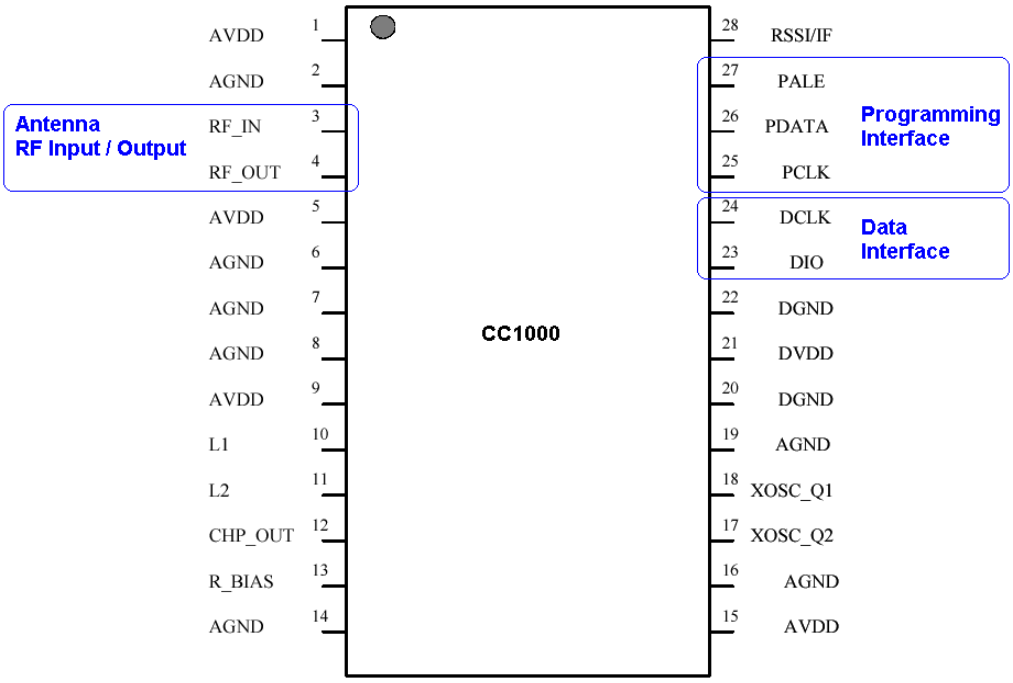


Abbildung 2.3 – Transceiver Interface

Wie im Pinout zu sehen, wird das Radio-Modul über 2 getrennte Interfaces mit dem Prozessor verbunden.

1. Für die Daten, die über das Radio-Modul versendet, bzw. empfangen werden, ist hier ein synchroner 2-Draht-Bus vorgesehen. Dieser wird beim Prozessor auf das SPI-Interface abgebildet.
2. Für die Konfiguration des Radio-Moduls ist ein synchroner 3-Draht-Bus vorgesehen, über welchen der Prozessor alle notwendigen Einstellungen vornehmen kann (z.B. Ausgangsfrequenz, Codierungsart, ...).
   Hier wird kein in den Prozessor integriertes Interface benutzt, sondern dieses I2C-ähnliche Interface per Software auf GeneralPurpose-Pins abgebildet.

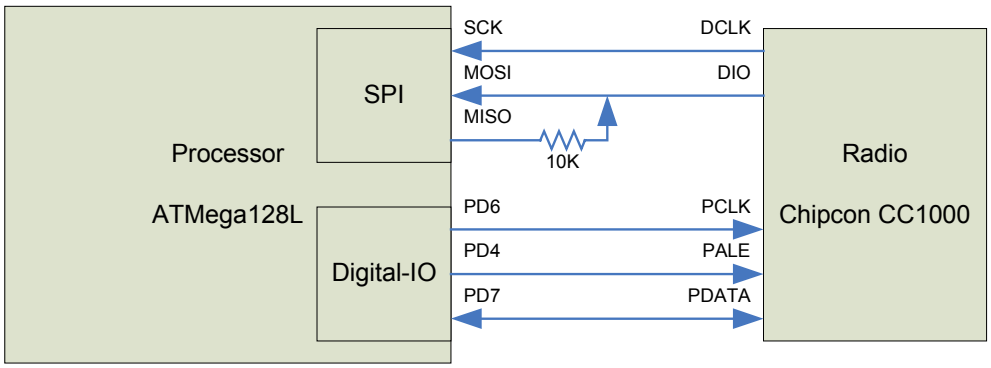


Abbildung 2.4 – Transceiver Mapping

2.2.1.3. Data-Flash

Das Data-Flash ist für den Data-Logger gedacht, welcher Daten über einen langen Zeitraum sammeln und hier nichtflüchtig abspeichern kann.
Somit ist ein garantierte Sicherung der Daten, auch nach Ausfall der Stromversorgung gegeben.
Bei dem eingesetzten Flash handelt es sich um ein „AT45DB041B“ der Firma Atmel.
Die wichtigsten Eigenschaften sollen hier kurz erläutert werden:

- *Niedrige Versorgungsspannung*
  2,5 V – 3,6 V, somit für Batteriebetrieb geeignet.
- *Niedriger Stromverbrauch*
  4mA Active-Read, 2µA Standby
- *SPI-Interface*
  Durch das SPI-Interface ist eine einfache Kopplung an den Prozessor möglich.
  Taktung bis 20MHz
- *Speichergröße, Aufbau*
  Es handelt sich um einen 4 MBit Speicher, welcher sich in
  2048 Pages zu je 264 Byte teilt.
  Er verfügt außerdem über zwei 264 Byte große SRam-Puffer, welche es ermöglichen, bei eingeleiteter Programmierung, weitere Daten zu empfangen.
  Über den Pin WP/ können die ersten 256 Pages schreibgeschützt werden.
- *Pinout*
  Den Speicher wird in verschiedenen Gehäusen angeboten, eingesetzt wird hier ein 8-Pin-SOIC Gehäuse, welches den Chip sehr klein macht.

Das folgende Bild zeigt das Pinout des Speichers im SOIC-8 Gehäuse.
Bei diesem Gehäuse kann der interne Zustand (Ready/Busy) nicht herausgeführt und somit benutzt werden. Dieser Zustand ist nur bei größeren Gehäusen (mehr Pins!) verfügbar.
Hier wurden die wichtigsten Interfaces gekennzeichnet.

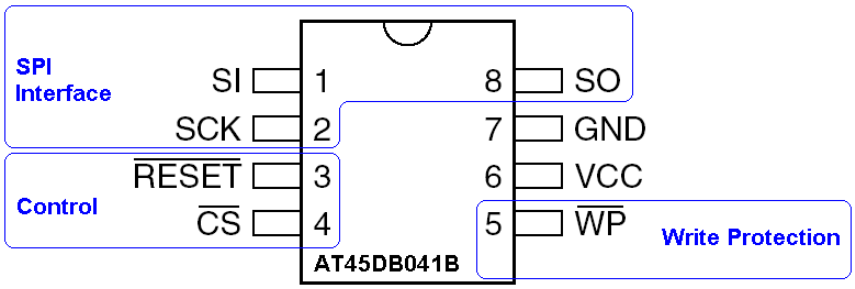


Abbildung 2.5 – Flash Interface

Da das interne SPI-Interface des Prozessors bereits mit dem Funk-Modul verbunden ist, und offensichtlich kein adressierter Bus aufgesetzt werden sollte, um die Kommunikation zu den, voneinander völlig unabhängigen, Modulen nicht zu beeinträchtigen, wurde als Verbindung zum Speicher eines der beiden USART-Interfaces verwendet.
Das heißt, dass die synchrone Variante des Interfaces als SPI-Interface benutzt wird. Die Chip-Select – Steuerung übernimmt dabei die Software über einen GeneralPurpose-Pin.
Wie in der unten dargestellten Grafik zu sehen, existiert keine Reset-Steuerung. Der Reset-Eingang des Speichers ist statisch High.
Auch die Möglichkeit des Schreibschutzes wird nicht benutzt. Der Eingang wurde offen gelassen, und ist somit durch den integrierten PullUp-Widerstand statisch High.

Ein Schreibschutz würde an dieser Stelle auch wenig Sinn machen, da dies den Speicher reduzieren würde und hier keine wirklich sensitiven Daten, wir z.B. ein Boot-Kernel, gespeichert werden.

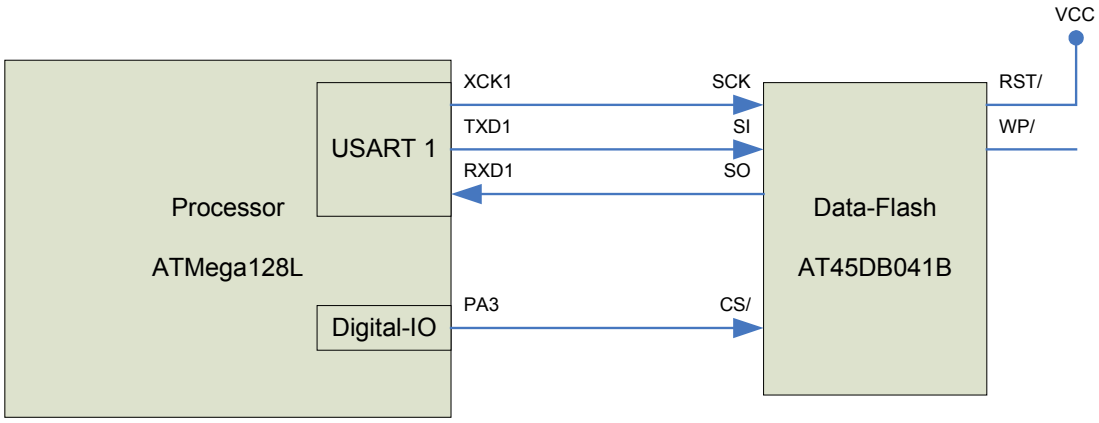


Abbildung 2.6 – Flash Mapping

2.2.1.4. Silicon-Serial-Number

Eine Möglichkeit, ein Gerät mit einer Seriennummer auszustatten ist, diese auf die Platine zu drucken. Sie kann nun zwar vom Anwender abgelesen werden, die Software weiß aber nichts über seine eigene Seriennummer.
Eine zusätzliche Möglichkeit ist, einen Silicon-Serial-Number-Chip auf die Platine zu setzen.
Er dient als digitales Label, nachdem er per Laser mit einem 64-bit Code versehen wurde. Dieser besteht aus:

- 8 Bit Family-Code
- 48 Bit Serial-Number
- 8 Bit CRC

Nun ist die, auf dem Prozessor laufende, Software in der Lage diese Seriennummer auszulesen und sie z.B. direkt als Wireless-Network - Adresse zu benutzen, da sie einmalig ist.

Benutzt wird der „DS2401“ von der Firma Dallas Semiconductor, welcher folgende weitere Eigenschaften besitzt.

- *Niedrige Versorgungsspannung*
  2,8 V – 6.0 V, somit für Batteriebetrieb geeignet.
- *Eindeutige Seriennummer*
  gewährleistet durch einen Fabrik-gelaserten und getesteten 64-Bit Code, und kann somit als
  - PCB-Identifikation
  - Netwerk Node ID
  - Geräte Registrierung

  genutzt werden.
- *1-Wire*
  Reduzierung der benötigten Pins für Control, Adresse, Daten und Stromversorgung auf 1!
- *Multidrop-Controller*
  Durch den integrierten Multidrop-Controller ist eine Kompatibilität zu anderen 1-Wire-Net Geräten sichergestellt.
- *Datenrate*
  Mit diesem Baustein lassen sich Datenraten bis 16,3 KBit/s erreichen.

In der folgenden Abbildung ist das Pinout des Chips dargestellt. Da nur 2 Pins benötigt werden, gibt es ihn auch als TO92 und SOT-223.

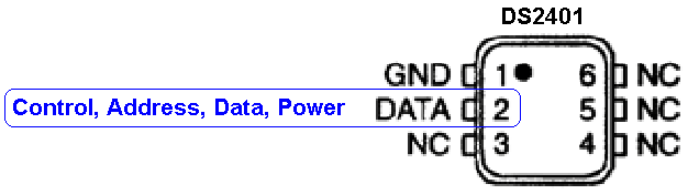


Abbildung 2.7 – Silicon Serial Number Interface

Der Anschluss am Prozessor ist somit denkbar einfach. Da nur eine einzige Leitung nötig ist, wird ein GeneralPurpose-Pin für die Kommunikation benutzt.

Die Stromversorgung des Gerätes erfolgt ebenfalls über die Datenleitung, in einem vom Master initiierten Slot, bei dem ein Kondensator aufgeladen wird, um für die nächste Kommunikation ausreichend Strom liefern zu können.

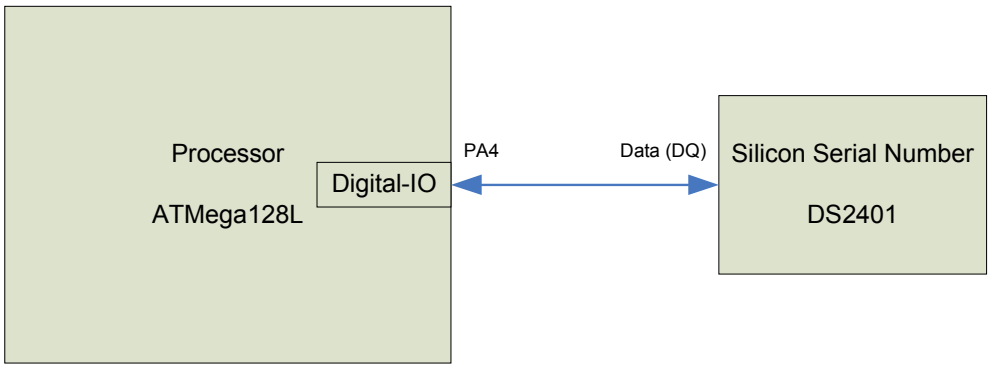


Abbildung 2.8 – Silicon Serial Number Mapping

2.2.1.5. Interface

Das Interface ist kein aktives Bauelement, sondern nur ein Erweiterungssteckplatz, an welchen ein, oder mehrere Sensor-Board(s) angeschlossen werden kann / können.
Es handelt sich hierbei um den sehr kleinen Board-Verbinder „DF9-51P-1V“ der Firma Hirose mit 51 Kontakten.
Außerdem wird über diese Schnittstelle, bei Verwendung eines Gateway-Boards (siehe Kapitel Gateway-Board), eine Verbindung zum PC möglich.
Somit kann das Board (Prozessor) programmiert werden, oder mit anderen, auf dem PC laufenden Applikationen, kommuniziert werden.
An diese Schnittstelle wurden alle Prozessor-Pins geführt, sodass jegliche Art von Erweiterungs-Board angeschlossen werden kann.

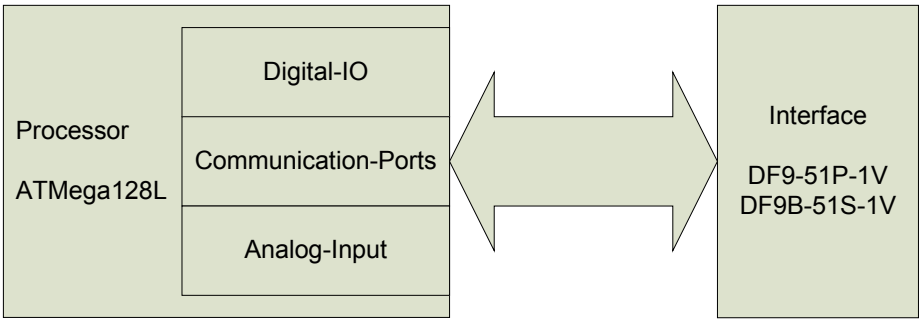


Abbildung 2.9 – Interface Mapping

Nach der allgemeinen Beschreibung eines Prozessor-Board sollen nun die konkreten Geräte beschrieben werden.
Nun werden die später benutzten Module MICA2 und MICA2DOT im Detail betrachtet und weitere zur Zeit verfügbare Geräte kurz vorgestellt.

2.2.2. MICA2 (MPR4x0)

Jetzt folgt die detailierte Beschreibung des wichtigsten Prozessor-Boards, da auf diese Basis alle weiteren aufsetzen.
Das MICA2-Board ist der Nachfolger des MICA-Boards, welches das erste kommerziell eingesetzte Gerät war, mittlerweile aber nicht mehr verfügbar ist.
Das MICA-Board kam aus der 2. Generation dieser Geräte und besaß im Prinzip nur einen schwächeren Prozessor, der durch den jetzt eingesetzten ATMega128L ersetzt wurde.

2.2.2.1. Eigenschaften

Folgend werden die wichtigsten Eigenschaften des MICA2 beschrieben:

- 7,3728 MHz Taktung
  Dadurch genaue Einhaltung des UART-Timings möglich (z.B. 115,2 KBaud)
- 8-Kanal 10 Bit-ADC
- Funk
  Der MICA2 ist mit verschiedenen Funk-Modulen verfügbar, welche sich durch das Frequenzband und die Reichweite unterscheiden:

Eigenschaft	**MPR400CB**	**MPR410CB**	**MPR420CB**
Frequenz	868/916 MHz	433 MHz	315 MHz
Übertragungsrate	Max. 38,4 KBit/s		
Reichweite	167m	330m	330m

- Stromverbrauch
  - In Betrieb ca. 8 mA
  - Standby < 15 µA
  - Funk
    - Senden ca. 25-27 mA
    - Empfangen ca. 8-10 mA
- Erweiterungs-Schnittstelle
  51 Pol
- Abmessungen
  58 x 32 x 7 mm ohne Batterie-Halter
- Gewicht
  18 g ohne Batterien

Als User-Interface stehen 3 LEDs zur Verfügung, somit kann durch den Anwender der interne Status angezeigt werden.
Eine Erweiterungs-Schnittstelle ist für den Anschluss von Sensor-Boards und Gateway-Boards vorhanden.
Versorgt wird das Gerät im Feldbetrieb von 2x1,5V Batterien.
Für die Stromversorgung im Labor ist ebenso ein Anschluss für 3 VDC vorhanden.

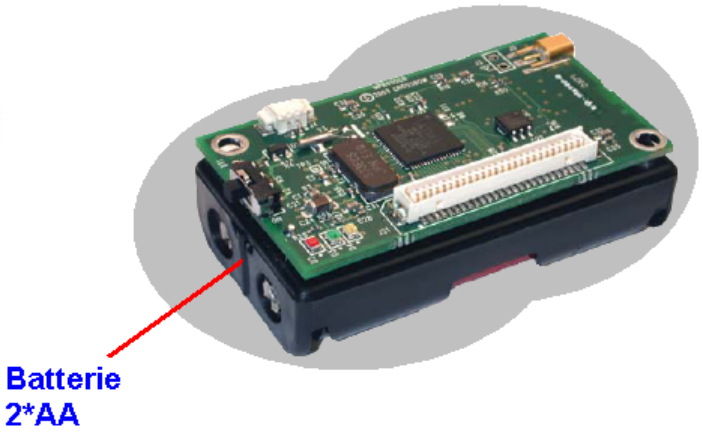


Abbildung 2.10 – Mica2

Die folgende Grafik zeigt die Oberseite des MICA2-Board im Detail.

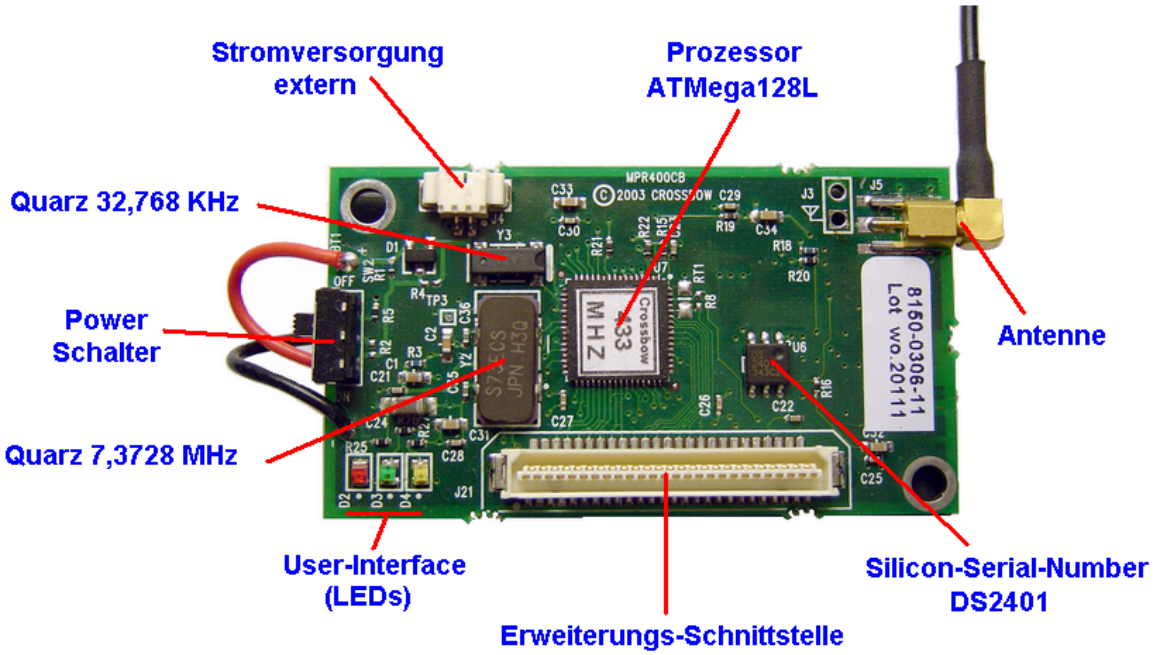


Abbildung 2.11 – Mica2 Detail (Oberseite)

Die folgende Grafik zeigt die Unterseite des MICA2-Boards im Detail (mit entferntem Batteriefach).

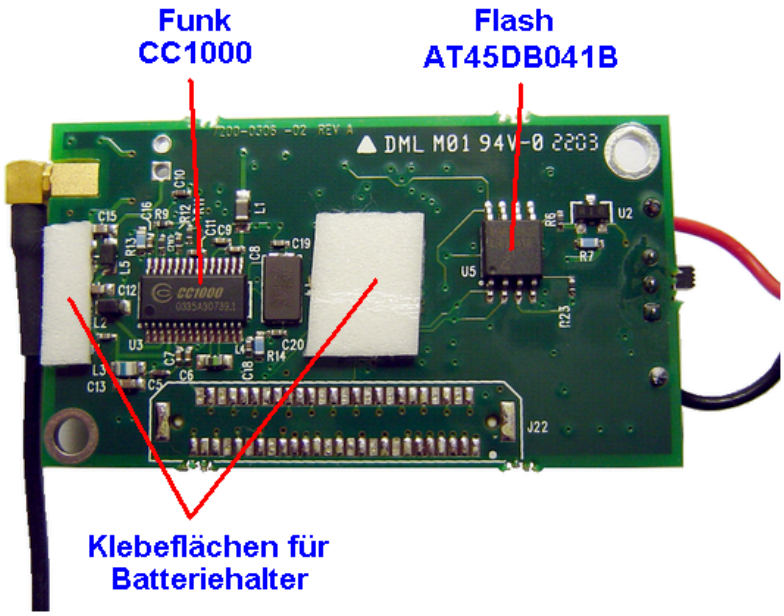


Abbildung 2.12 – Mica2 Detail (Unterseite)

2.2.2.2. Anschlussmöglichkeiten

Es stehen verschiedene Sensor-Boards zur Verfügung, die über die Erweiterungs-Schnittstelle an MICA2 angeschlossen werden können.
Hierzu zählen:

- MTS101CA
- MTS300CA
- MTS310CA

Die Beschreibung der Module folgt in Kapitel Sensor-Board.

Außerdem ist ein Gateway-Board anschließbar,

- MIB510CA

um das Gerät per PC zu programmieren, aber auch um direkt mit dem PC zu kommunizieren.
Dann kann der PC z.B. zur Visualisierung von übertragenen Statistiken benutzt werden.

2.2.3. MICA2DOT (MPR5x0)

Das MICA2DOT-Board besteht im Prinzip aus der gleichen Hardware wie ein MICA2, besitzt aber kleinere Abmessungen.
Ein paar Änderungen sollen hier aber kurz beschrieben werden.

2.2.3.1. Änderungen

Abweichungen zum unter Kapitel „Allgemein“ dargestellten Hardware:

- Einige Signalleitungen werden hier anders verwendet.
- Das Board besitzt keinen Silicon-Serial-Number – Baustein
- Die Erweiterungsschnittstelle besteht hier nicht aus dem 51-poligen Adapter der Firma Hirose, sondern aus einem 18-poligen Feld von Stiften.

2.2.3.2. Eigenschaften

Folgend werden die wichtigsten Eigenschaften des MICA2DOT beschrieben:

- 4 MHz Taktung
- 8-Kanal 10 Bit-ADC
- Funk
  Der MICA2Dot ist, wie auch der MICA2, mit verschiedenen Funk-Modulen verfügbar, welche sich durch das Frequenzband und die Reichweite unterscheiden:

Eigenschaft	**MPR500CA**	**MPR510CA**	**MPR520CA**
Frequenz	868/916 MHz	433 MHz	315 MHz
Übertragungsrate	Max. 38,4 KBit/s		
Reichweite	167m	330m	330m

- Stromverbrauch
  - In Betrieb ca. 8 mA
  - Standby < 15 µA
  - Funk
    - Senden ca. 25-27 mA
    - Empfangen ca. 8-10 mA
- Erweiterungs-Schnittstelle
  18 Pol
- Abmessungen
  25 x 6 mm ohne Batterie (Knopfzelle)
- Gewicht
  3 g ohne Batterie

Als User-Interface steht eine LED zur Verfügung, somit kann durch den Anwender der interne Status angezeigt werden.
Eine Erweiterungs-Schnittstelle ist für den Anschluss von Sensor-Boards und Gateway-Boards vorhanden.
Versorgt wird das Gerät im Feldbetrieb von 1x3V Batterien (Knopfzelle).
Der Anschluss einer externen Stromversorgung ist nicht direkt möglich.

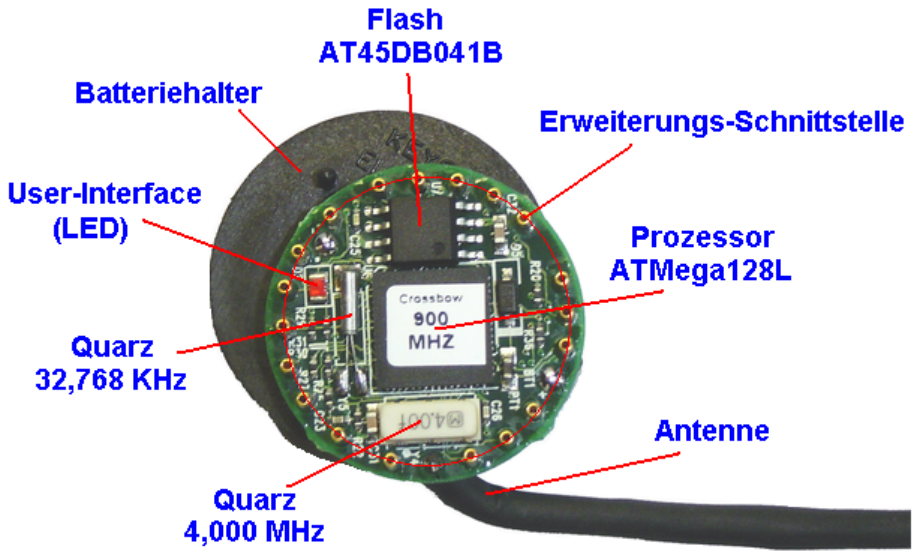


Abbildung 2.13 – Mica2Dot Detail

2.2.3.3. Anschlussmöglichkeiten

Es stehen verschiedene Sensor-Boards zur Verfügung, die über die Erweiterungs-Schnittstelle an MICA2DOT angeschlossen werden können.
Hierzu zählen:

- MTS510CA
- MDA500CA

Die Beschreibung der Module folgt in Kapitel Sensor-Board.

Außerdem ist ein Gateway-Board anschließbar,

- MIB510CA

um das Gerät per PC zu programmieren, aber auch um direkt mit dem PC zu kommunizieren.

2.2.4. MICAz (MPR2400)

Das MICAz-Board ist auf Basis des MICA2 aufgebaut und besitzt ein Funk-Interface, welches durch seine Standardisierung einen universellen Einsatz zulässt. An dieser Stelle sollen nur die Änderungen beschrieben werden.

2.2.4.1. Änderungen

Abweichungen zum unter Kapitel „Allgemein“ dargestellten Hardware:

- Bis auf das Funk-Modul ist die Hardware von MICAz prinzipiell gleich der des MICA2-Board.
- Funk
  Hier wird statt dem CC1000, der Funk-Transceiver CC2420 eingesetzt.
  Durch Verwendung dieses Funkmoduls ist das Gerät „ZigBee“-kompatibel.

2.2.4.2. Eigenschaften

Die Eigenschaften des MICAz sind mit denen des MICA2 identisch, mit Ausnahme des beschriebenen Funk-Interfaces.
Dieses ist im Bereich von 2,4 GHz bis 2,4835 GHz einstellbar und leistet eine wesentlich höhere Datenrate von 250 KBit/s, statt den 38,4 KBit/s beim MICA2.
Die Reichweite beträgt in Gebäuden bis 30m, im Freien ca. 100m.

2.2.5. MCS Cricket (MCS410)

Das MCS Cricket-Board ist eine sehr interessante Entwicklung auf Basis des MICA2-Boards und soll als „Wireless Location System“ dienen.
Es besitzt, zusätzlich zum Funk-Transceiver, einen Ultraschall-Transceiver, deren Zusammenspiel eine Laufzeitmessung des Ultraschalls, und somit eine genaue Abstandsmessung ermöglicht.
Bei Benutzung von mehreren Sendern, in Bezug zu einem Empfänger, kann ein „GPS-System“ für Gebäude realisiert werden.
Werden in einem Gebäude ‚alle’ Räume mit je mehreren dieser Sender ausgestattet, und eine Person trägt ein MCS Cricket-Board am Körper, so kann dessen genaue Position, auch auf unterschiedlichen Etagen ermittelt werden.
Dieses wäre mit einem ‚normalen’ GPS nicht möglich, da hier zwar die Längen- und Breiten-Informationen zur Verfügung steht, nicht jedoch eine Höheninformation, bzw. in welchen Stockwerk sich die Person gerade befindet.
Außerdem werden die Funk-Signale der GPS-Satelliten durch die Gebäude-Wandung gestört oder komplett reflektiert.

Das System wurde von „Crossbow“ in Zusammenarbeit mit „MIT“ entwickelt und ist auch als „Cricket V.2“ bekannt.
Ein MCS Cricket kann als Sender oder als Empfänger dienen. Wenn keine statische Raum-Struktur vorliegt, kann er auch gleichzeitig als Sender und Empfänger fungieren.
Da diese Geräte für diese Arbeit aber nicht zur Verfügung standen, soll hier keine nähere Beschreibung erfolgen.
Für weiterführende Informationen sei auf die Seite des Herstellers http://www.xbow.com verwiesen.

2.3. Sensor-Board

Die Sensor-Boards ergänzen die Prozessor-Boards durch ihre Möglichkeit, verschiedene physikalische Größen messen zu können.
In Verbindung mit den beschriebenen Prozessor-Boards steht somit einem Wireless-Sensor-Network nichts mehr im Wege.
Die Sensor-Boards sind allerdings rein passiv, sodass sie für sich alleine gesehen, keinen Nutzen haben. Sie besitzen weder Prozessoren oder Controller, noch verfügen sie über aktive Kommunikations-Schnittstellen.

2.3.1. Allgemein

Sie werden über die Erweiterungs-Schnittstelle eines Prozessor-Boards mit diesem verbunden und dienen hier nur als Slaves.

Die folgende Grafik zeigt den prinzipiellen Aufbau dieser Boards.
Die verschiedenen Boards beinhalten unterschiedliche Sensoren und eventuell auch die hier dargestellte Prototypen-Fläche.

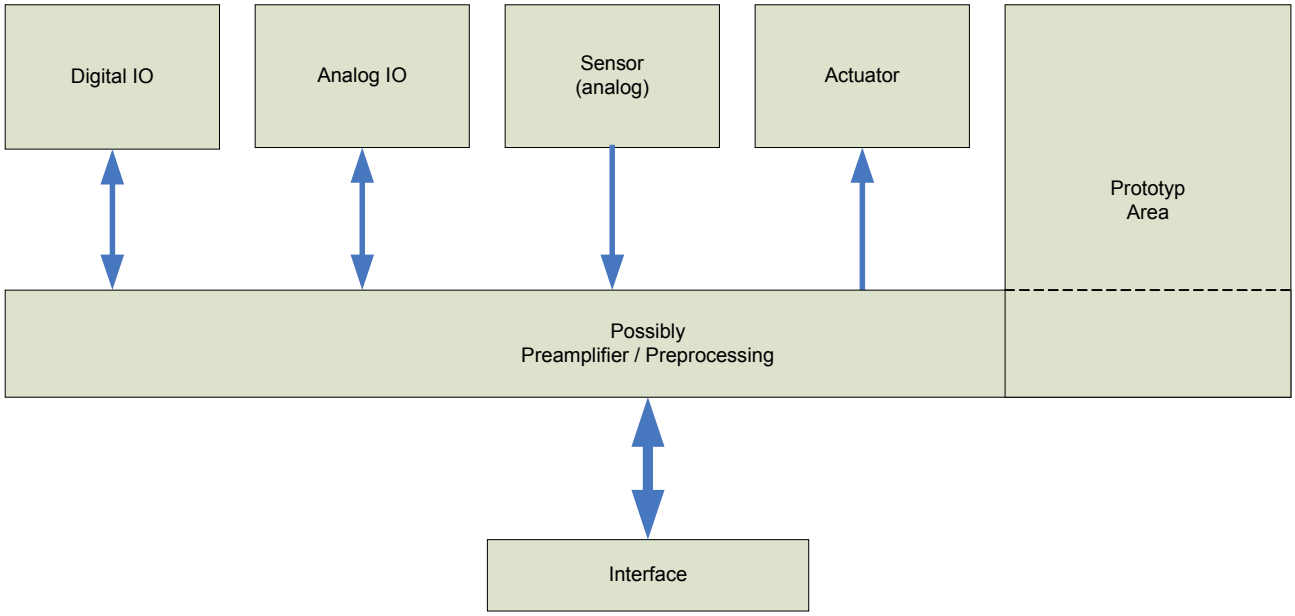


Abbildung 2.14 – Aufbau Sensor-Board

Nachfolgend werden die zwei, in der eigenen Applikation benutzten, Sensor-Boards im Detail beschrieben.
Daran anschließend folgt eine Zusammenfassung aller weiteren zur Zeit verfügbaren Geräte.

2.3.2. Multi Sensor (MTS510)

Dieses Sensor-Board zeichnet sich durch einen Beschleunigungs-Sensor aus und besitzt zusätzlich noch folgende Sensoren:

- Licht
- Mikrofon

Nach einer Beschreibung seines Aufbaus und Anschluss an ein Prozessor-Board folgt eine Erläuterung der einzelnen Sensoren.

2.3.2.1. Aufbau

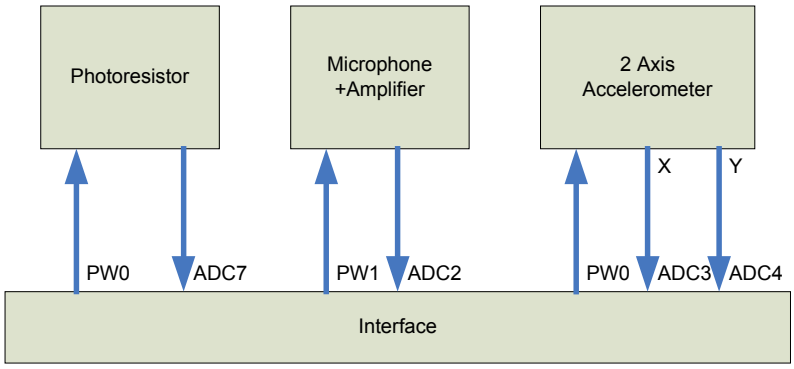


Abbildung 2.15 – Aufbau MTS510

Die Sensoren werden durch die Signalleitungen PW0 bzw. PW1 aktiviert und sind direkt an die gekennzeichneten ADC-Eingänge des Prozessors angeschlossen.
Der Beschleunigungs-Sensor besitzt sowohl einen digitalen, als auch einen analogen Ausgang. Benutzt wird hier der analoge Ausgang, da dieser einfach an den ADC des Prozessors angeschlossen werden kann.

2.3.2.2. Eigenschaften

- *Licht-Sensor*
  Der Sensor besteht aus einer CdSe-Photozelle vom Typ „Clairex CL94L“.
  Die maximale Empfindlichkeit wird bei einer Wellenlänge von 690nm (Rotlicht) erreicht.
  - Widerstand
    Licht an 2KOhm
    Licht aus 520KOhm
- *Mikrofon*
  Die Mikrofon-Kapsel „WM-62A“ von Panasonic besitzt folgende Eigenschaften:
  - Empfindlichkeit: -45dB +-2 dB
  - Impedanz: < 2,2 KOhm
  - Frequenzbereich: 20-16000 Hz

  Die Empfindlichkeit ist über ein Digital-Potentiometer via I2C-Bus einstellbar.
- *Beschleunigungsmesser*
  Das Modul beinhaltet einen 2-Achsen Beschleunigungsmesser „ADXL202E“ von Analog Devices.
  Er besitzt sowohl einen digitalen (PWM), als auch analogen Ausgang.
  - Messbereich: +-2*g*
  - Genauigkeit: bis 2m*g*
  - Bandbreite ca. 10KHz

Folgende Grafik zeigt den MTS510 mit seinen Sensoren:

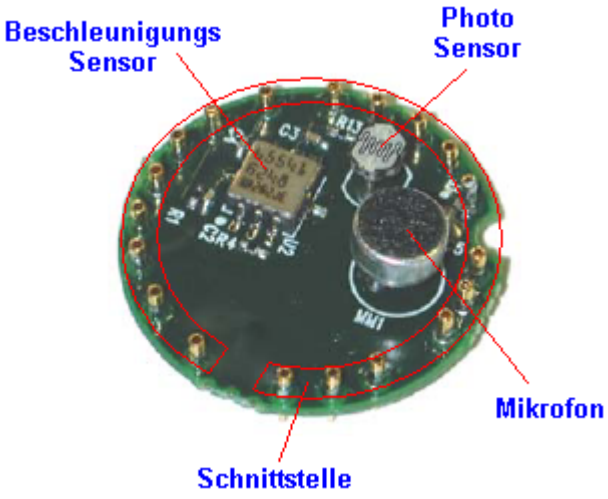


Abbildung 2.16 – MTS510 Detail

2.3.2.3. Anschlussmöglichkeiten

Das MTS510-Board kann durch die 18-polige Schnittstelle an dem Prozessor-Board

- MICA2DOT

angeschlossen werden.

2.3.3. Prototype and Data Acquisition (MDA500)

Das Prototype and Data Acquisition-Board besitzt keinerlei Sensoren, es besitzt nur eine Prototyp-Fläche, auf welcher sich eigene Sensoren realisieren lassen.

2.3.3.1. Aufbau

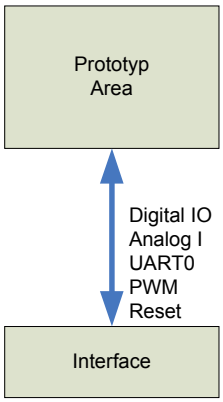


Abbildung 2.17 – Aufbau MDA500

Auf der Prototyp-Fläche sind die wichtigsten Signalleitungen des Prozessors vorhanden. Es stehen sowohl analoge, digitale, als auch ein Bus-Interface (UART) zur Verfügung.

2.3.3.2. Eigenschaften

Auf dieser Fläche hat der Anwender die Möglichkeit eigene Sensoren oder Module mit einem MICA2DOT-Board zu verbinden.
Hier steht eine Selektion der Prozessor-IOs zur Verfügung.

- 6 Digital-IO
- 6 Analog-I
- UART0 Tx, Rx
- SPI Clock
- Reset
- 1 PWM-Kanal
- Stromversorgung

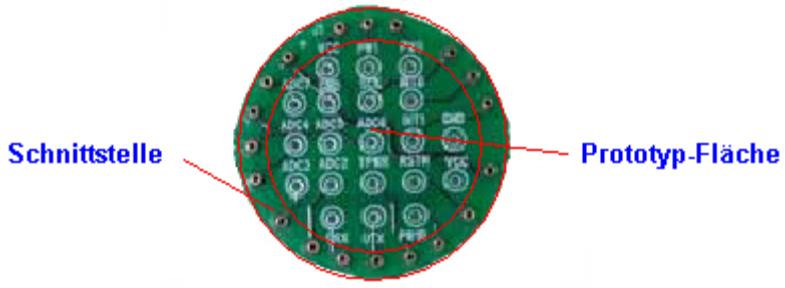


Abbildung 2.18 – MDA500 Detail

2.3.3.3. Anschlussmöglichkeiten

Das MDA500-Board kann durch die 18-polige Schnittstelle an dem Prozessor-Board

- MICA2DOT

angeschlossen werden.

2.3.4. Weitere Sensor-Boards

Der Vollständigkeit halber werden hier alle weiteren Sensor-Boards der Firma Crossbow in einer Zusammenfassung gezeigt.

Sensor-Board	Sensoren/ Aktoren	Anschlussmöglichkeit
MTS101	▪ Licht ▪ Temperatur ▪ Prototyp-Fläche	▪ MICA2 ▪ MICAz ▪ MCS Cricket
MTS300 MTS310	▪ Licht ▪ Temperatur ▪ Mikrofon ▪ Ton-Detektor ▪ Sounder Das MTS310 besitzt zusätzlich: ▪ 2-Achsen Beschleunigungsmesser ▪ 2-Achsen Magnetfeldmesser	
MTS400 MTS420	▪ Licht ▪ Temperatur ▪ Luftfeuchtigkeit ▪ Luftdruck ▪ 2-Achsen Beschleunigungsmesser Das MTS420 besitzt zusätzlich: ▪ GPS-Modul	
MDA300	Das MDA300 besitzt folgende eigene Sensoren: ▪ Temperatur ▪ Luftfeuchtigkeit und bietet die Möglichkeit, eine Reihe von zusätzlichen Sensoren anzuschließen. Hierzu verfügt es über folgende Schnittstellen: ▪ Bis zu 11 ADC-Eingänge (12 Bit) ▪ 6 Digital Ein/ Ausgänge mit Interrupt-Auslösung ▪ 2 Relais-Kanäle für große Lasten Weiterhin verfügt es über ein 64KBit großes I2C-EEPRom für Benutzerdaten und führt das I2C-Interface nach außen.	

Es stehen somit eine Reihe von Modulen zur Verfügung, sodass für fast jede Anwendung etwas passendes dabei sein sollte.
Diese Boards lassen sich aufgrund ihrer Bauausführung nur im Einzelfall miteinander kombinieren, was aber in Anbetracht ihrer Vielfalt und den jeweils vorhandenen Sensoren, sowieso unnötig sein sollte.

2.4. Gateway/Network-Interface - Board

Die Gateway-Boards bilden die Grundlage für die Entwicklung einer Applikation und bieten zusätzlich eine Verbindung zu anderen Geräten.
Es folgt zunächst eine allgemeine Beschreibung, bevor das, für die eigene Applikation eingesetzte, Board im Detail erläutert wird.

2.4.1. Allgemein

Die Gateway-Boards dienen zur Verbindung der MOTEs mit einem PC oder dem Ethernet. Dadurch kann der Atmel-Prozessor auf den Prozessor-Boards via UART, oder Ethernet programmiert werden.
Gleichzeitig dienen die Gateway-Boards als Entwicklungs-Plattform, da sie eine eigene Stromversorgung (Netzteil) besitzen, und weiterhin neben dem Prozessor-Board auch ein Sensor-Board aufnehmen können.
In diesem Zusammenhang besteht außerdem die Möglichkeit, eine direkte Kommunikation zwischen einer Anwender-PC-Software und einer auf dem MOTE laufenden Software herzustellen.
Dadurch kann der PC zur Darstellung von Statistiken oder sonstigen Daten dienen. Gleichzeitig besitzt der Benutzer hiermit ein sehr viel leistungsstärkeres User-Interface, als es direkt auf den MOTEs vorhanden ist (bis zu 3 LEDs).

Somit kann ein MOTE zur Basis-Station ernannt werden, welche die Daten aus dem Wireless-Sensor-Network sammelt und anschließend zu einem PC sendet, oder diese im Ethernet verfügbar macht.

2.4.2. Serial Gateway (MIB510)

Das Serial-Gateway bietet als Kommunikations-Schnittstelle ein UART-Interface, mittels dessen ein MOTE programmiert werden kann, oder eine Kommunikation mit einer PC-Software aufgebaut werden kann.

2.4.2.1. Aufbau

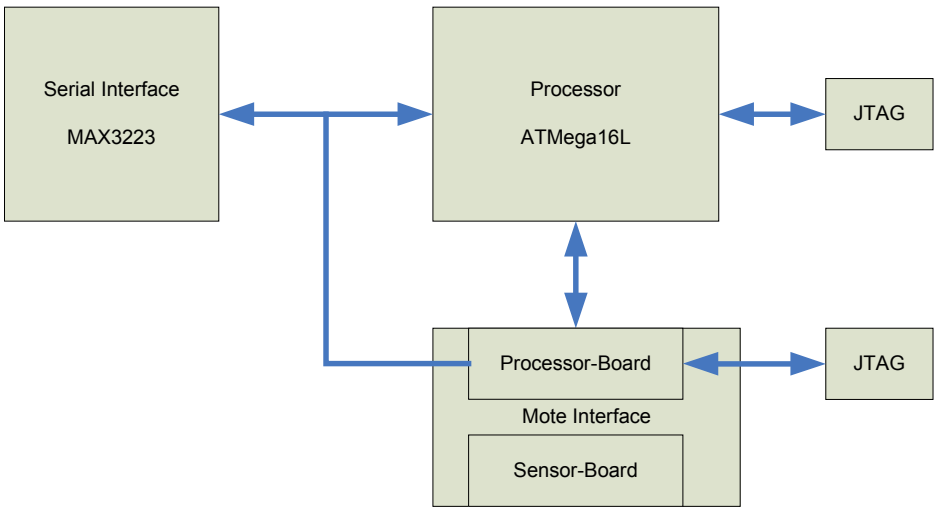


Abbildung 2.19 – Aufbau MIB510

2.4.2.2. Funktion

Da es sich um ein aktives Gerät handelt, soll hier seine Funktion und Funktionsweise erklärt werden.

- *Serial Interface*
  Die Serielle Schnittstelle RS232 besteht nur aus den Leitungen Tx und Rx, somit ist hier keine Hardware-Flusssteuerung möglich.
  - Übertragungsraten
    MOTE: typ. 57,6 kBaud (vom Benutzer einstellbar)
    Programmierung: 115,2 kBaud

  Es steht nur ein Interface für den MOTE und die Programmierung zur Verfügung.
  Während der Programmierung sind die Leitungen des MOTEs deaktiviert.
- *Prozessor*
  Der ATMega16L ist für die Programmierung des MOTEs zuständig.
  Er überwacht den Datenverkehr auf der Seriellen Schnittstelle und prüft auf ein spezielles multi-byte-pattern, das nur zu Programmierzwecken auftreten darf.
  Ist dieses der Fall, so deaktiviert er die Schnittstelle des MOTEs und übernimmt die Kontrolle der RS232-Schnittstelle, um den MOTE zu programmieren.
  Der PC sendet dabei die Rohdaten des zu programmierenden Speichers an den Prozessor, während dieser den Programmier-Algorithmus ausführt um schließlich den MOTE per SPI zu programmieren.
  Außerdem überwacht er die Stromversorgung und deaktiviert bei Unterschreitung einer festgesetzten Grenze die Programmierung.
- *MOTE Interface*
  Hier existieren 2 Schnittstellen, an die alle verfügbaren MOTEs angeschlossen werden können.
  - 51-Pol Schnittstelle für MICA, MICA2-basierierte Boards
  - 19-Pol Schnittstelle für MICA2DOT

  Gleichzeitig kann ein Sensor-Board angeschlossen werden.
- *JTAG*
  Es sind 2 JTAG-Schnittstellen vorhanden,
  - Eine für in-circuit-debugging des MOTEs
  - Die Zweite für den eingebauten Prozessor ATMega16L (nur für den Hersteller, sollte nicht benutzt werden!)

Die folgende Grafik zeigt die Oberseite des MIB510-Boards.

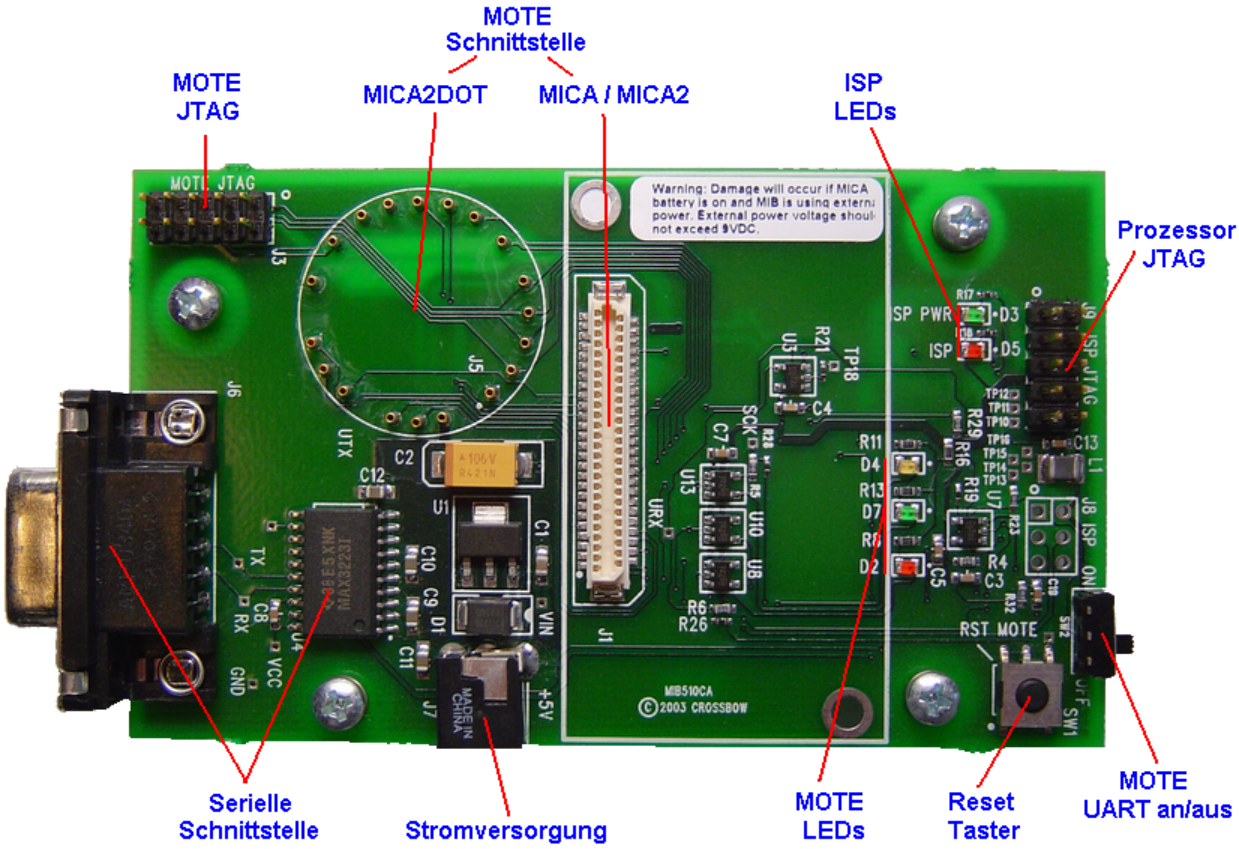


Abbildung 2.20 – MIB510 Detail (Oberseite)

Für die Stromversorgung hat der Anwender zwei Möglichkeiten:

1. Versorgung durch den externen Anschluss (5-7V), wobei gleichzeitig ein angeschlossenes MOTE mitversorgt wird, oder
2. Versorgung durch die Stromversorgung eines MOTEs (Batterien).

Dabei muss aber sichergestellt werden, dass immer nur eine Quelle aktiv ist, da es ansonsten zu Beschädigungen kommen kann!
Die ISP (InCircuitProgram) LEDs zeigen an, ob eine ausreichende Versorgungsspannung vorhanden ist, und ob gerade eine Programmierung erfolgt.
Die MOTE LEDs sind mit denen auf dem Prozessor-Board kurzgeschlossen.
Mit dem Reset-Taster kann der MOTE resettet werden.

Die folgende Grafik zeigt die Unterseite des MIB510-Boards.

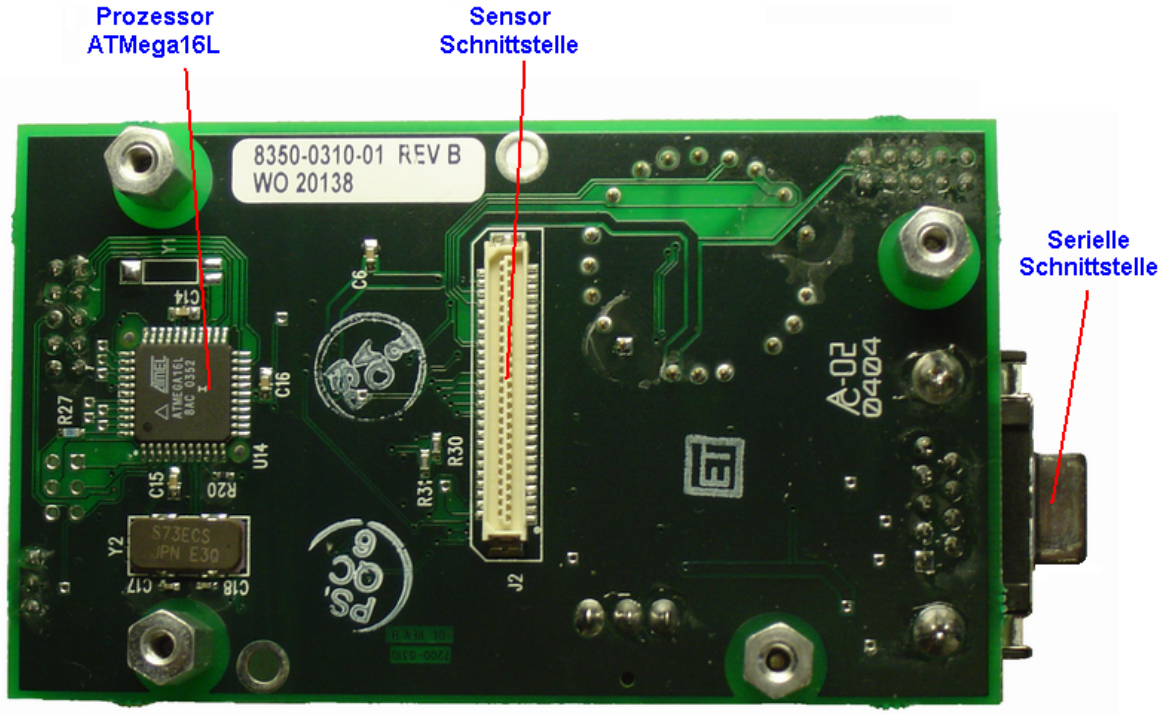


Abbildung 2.21 – MIB510 Detail (Unterseite)

Der Anschluss der MOTEs erfolgt an den jeweils dafür vorgesehenen Schnittstellen, wobei sich immer nur ein MOTE auf dem Gateway befinden darf.

2.4.2.3. Anschlussmöglichkeiten

An das Gateway-Board MIB510 können alle vorhandenen Prozessor-Boards:

- MICA
- MICA2
- MICA2DOT
- MICAz
- MCS Cricket

und Sensor-Boards:

- MTS101
- MTS3x0
- MTS4x0
- MTS510
- MDA300
- MDA500

angeschlossen werden.
An die Serielle Schnittstelle kann jedes RS232-fähige Gerät angeschlossen werden.

2.4.3. Ethernet Gateway (MIB600)

Eine weitaus leistungsfähigere Kommunikations-Schnittstelle bietet das MIB600.
Es besitzt ein Ethernet-Interface (10/100 Base-T), mittels dessen ein MOTE programmiert, und in einem LAN verfügbar gemacht werden kann.

Es beinhaltet das gesamte TCP/IP Protokoll und kann als Bridge zwischen „wired“ und „wireless“ -Segmenten eines Netzwerks dienen.
Es beinhaltet das Telnet-Protokoll, einen Serial-Server für den direkten Zugriff auf den MOTE-UART, sowie einen Web-Server und unterstützt sogar eine Passwort-Authentifizierung.
Das MIB600 unterstützt PowerOverEthernet (POE), sodass keine externe Stromversorgung nötig ist.

Alle weiteren Eigenschaften sind mit denen des MIB510 gleichzusetzen, sodass hier keine weitere Beschreibung erforderlich ist.

2.4.4. Stargate Gateway (SPB400)

Einen etwas anderen Weg geht das SPB400. Hierbei handelt es sich nicht um ein reines Gateway, sondern vielmehr um einen kompletter Single-Board-Computer (kurz SBC), welcher neben der Möglichkeit der Adaption von MOTEs auch eine Verarbeitungs-Einheit zur Verfügung stellt und somit den PC, als leistungsstarke Komponente in einem Netzwerk, ersetzen kann.
Gleichzeitig bietet es neben Embedded-typischen Schnittstellen wie UART und I2C auch PC-typische Schnittstellen, wie LAN, und USB.
Der SBC arbeitet auf Basis eines

- Intel PXA255 Prozessors,

welcher heutzutage auch in leistungsstarken PDAs zu finden ist.
Als Betriebssystem kommt ein

- Embedded Linux

zum Einsatz.
Das Gerät entstand als Resultat des
‚Intel's Ubiquitous Computing Research Program’.
Die Lizenz für die kommerzielle Produktion wurde an Crossbow übergeben, wodurch das Gerät gleichzeitig Applikationen für

- TinyOS-basierte Wireless Sensor Networks

sowie um die

- Intel's Open-Source Robotics initiative.

Unterstützt.

2.5. Fazit

Es stehen eine Reihe von leistungsfähigen Geräten zur Verfügung, mit denen sich sowohl simple, als auch professionelle Anwendungen (z.B. inklusive absoluter Positionserkennung dank GPS) realisieren lassen sollten.
Ob diese Erwartungen in Verbindung mit dem benutzten Betriebssystem erfüllt werden, bleibt jedoch abzuwarten.

3. Software

Für diese Art von Geräten, die nur sehr begrenzte Ressourcen in Bezug auf Rechenleistung, verfügbaren Speicher und Energie besitzen, wurde ein eigenes Betriebssystem entwickelt, welches auf mehrere Plattformen portiert wurde und auch weiterhin auf neue Geräte zugeschnitten wird.
Die Rede ist von „Tiny OS“, welches eine Open-Source Entwicklung mehrerer engagierter Entwickler-Teams ist.
Unter anderem wird die Entwicklung durch die „Berkeley University of California“ und „Intel’s Research Laboratory“ in Berkeley vorangetrieben.
Hinzu kommen diverse Applikationen sowohl für das Wireless Sensor Network, als auch für den PC, die dem Anwender und Entwickler den schnellen Einstieg in die Materie ermöglichen.
Gleichzeitig wird die Entwicklungsdauer neuer Produkte und Softwarelösungen weitestgehend minimiert. Hierzu leistet auch der eingesetzte Compiler „NesC“ einen großen Beitrag.
Der Entwickler erhält ein komplettes Entwicklungs-Paket, welches hier beschrieben werden soll.

3.1. Crossbow-Kit

Das Software-Paket, welches zum Zeitpunkt dieser Arbeit verfügbar war, besteht aus folgenden Komponenten:

- TinyOS 1.1.0 für Windows
  darin enthalten sind:
  - Cygwin, ein Linux-Emulator für Windows
  - TinyOS Bibliotheken für die verfügbaren Geräte
  - NesC – Compiler
  - Simulator, zur schnellen Software-Entwicklung am PC
  - Tutorial, für die ersten Schritte
  - JDK 1.4.1
  - diverse kleine Java-basierte Applikationen
  - Dokumentationen sowohl in digitaler, als auch in Papier-Form

Anschließend werden die wichtigsten Komponenten beschrieben. Angefangen mit einem kurzen Einblick in Cygwin, wird dann das TinyOS und der dazugehörende Compiler NesC mit dessen Konzepten beschrieben.
Am Ende des Kapitels folgt eine Einführung in die Kommunikation des TinyOS und dessen Protokoll-Stack.

3.1.1. Cygwin

Da die Entwicklung auf Cygwin aufbaut, soll es hier kurz beschrieben werden. Cygwin ist ein Linux-Emulator für Windows-basierte Systeme. Es bringt einige Linux-API Funktionen mit sich und verhält sich hierdurch wir ein echtes Linux-System.
Es können aber keine nativen Linux-Applikationen ausgeführt werden, diese müssten ausgehend von den Sourcen neu für Windows übersetzt werden.

Das installierte Tiny-OS Paket hat folgenden Verzeichnisbaum:

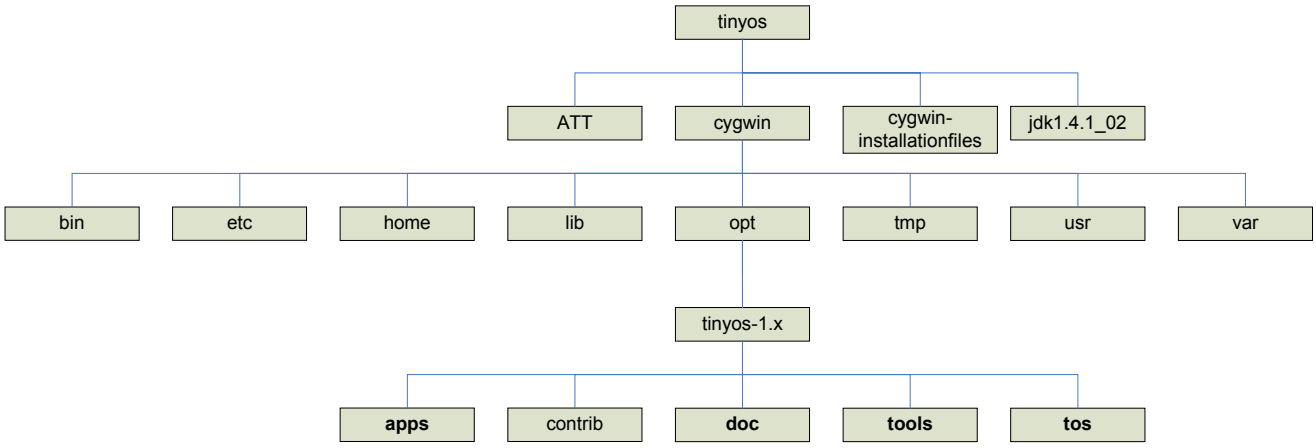


Abbildung 3.1 – Entwicklungsumgebung Verzeichnisbaum

Der unterste Ebene ist für den Entwickler die einzig wichtige und soll daher kurz erläutert werden.

- **apps**
  Hier befinden sich die Applikationen. Eine Reihe von Beispiel-Programmen, gehörend zum Tutorial, werden mitgeliefert.

- **doc**
  Hier befinden sich die Dokumentationen zu allen möglichen Themen rund um die Entwicklung.
  Außerdem ist das Tutorial hier zu finden, genauso wie die mit dem NesC-Compiler erstellbaren Dokumentationen zu eigenen Projekten.

- **tools**
  Hier angesiedelt sind diverse Applikationen (Java), die dem Entwickler helfen können, sein Projekt zu testen oder am PC zu simulieren.

- **tos**
  Hier befinden sich alle System-Komponenten von Tiny-OS und Interfaces zu den verschiedenen Soft- und Hardware-Modulen.

3.2. Tiny-OS

Da für eine erfolgreiche Entwicklung einige Grundkenntnisse über das Betriebssystem vorhanden sein sollten, werden dessen Aufbau und Eigenheiten in den nächsten Abschnitten erläutert.

Tiny-OS ist eine Open-Source Entwicklung eines Betriebssystems für „Embedded Sensor Networks“ vom „University of California Berkeley EECS Department“.
Es ist speziell auf die Hardware dieser Geräte zugeschnitten. Eine typische Umgebung ist das MICA2-Board von Crossbow, dessen Aufbau hier noch einmal gezeigt werden soll:

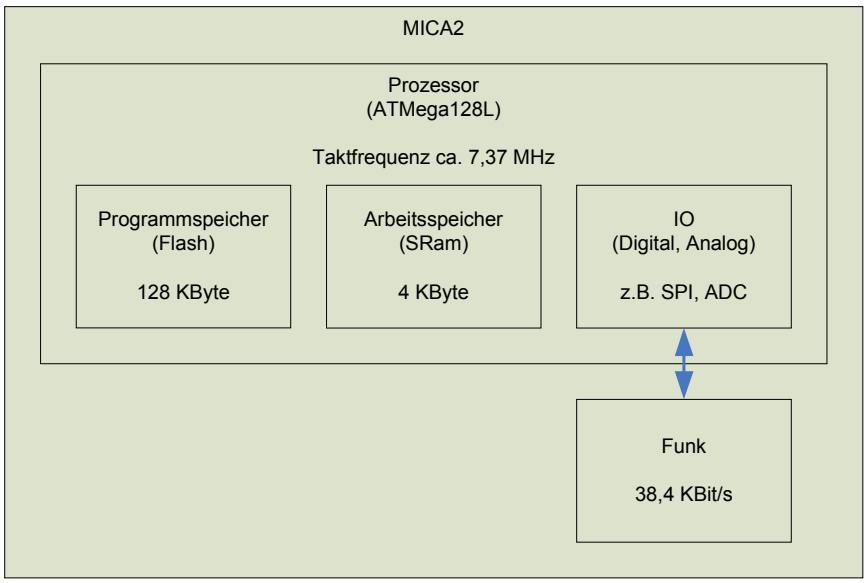


Abbildung 3.2 – TinyOS Umgebung

Es sind nur begrenzte Ressourcen im Bezug auf Programmspeicher, Arbeitsspeicher und Taktfrequenz vorhanden, daher muss das Betriebssystem sehr klein und effektiv sein.
Dieses wird durch seine Architektur erreicht:

3.2.1. Ereignis / Interrupt -gesteuert

Das Betriebssystem ist Ereignis- und Interrupts-gesteuert, wobei Interrupts die unterste Schicht darstellen, dessen Ereignisse an höhere Schichten gesendet wird. Das Anwenderprogramm bekommt eine ‚Benachrichtigung’, wenn eine Aktion erfolgreich ausgeführt wurde, oder ein externes Ereignis, wie z.B. ein Datenpaket über das Funk-Modul empfangen, eintritt. Es ist kein Polling nötig.

3.2.2. 2-Schicht FIFO Scheduler,

Tiny-OS beinhaltet einen 2-Schicht-Scheduler, welcher die Tasks und Events in ihrer Ausführung steuert.

3.2.2.1. Task

Alle Tasks liegen in einer Warteschlange und werden vom Scheduler sequenziell abgearbeitet. Sie können nicht priorisiert werden. In der Standard-Version von Tiny-OS können maximal 7 Tasks in der Warteschlange stehen.
Jeder Task wird bis zu seiner Terminierung bearbeitet, das heißt, sie können von Events unterbrochen werden, nicht jedoch von anderen Tasks!
Somit sind keine Aussagen über Reaktionszeiten möglich, sodass das Betriebssystem nicht Echtzeit-fähig ist.

3.2.2.2. Event

Events sind zeitkritische Abläufe und sollten nur eine sehr kurze Laufzeit bekommen. Sie werden von anderen Software-Modulen, oder Interrupts angestoßen, wie z.B. ein TimerOverflow-Interrupt.
Sie können Tasks und andere Events unterbrechen.

3.2.3. Speicher

Tiny-OS verfügt über ein Statisches Speicher-Model, dass heißt, es gibt keinen Heap (kein ‚malloc' möglich) und keinen Virtuellen Speicher.
Es gibt keine ‚MemoryManagementUnit' und somit auch keinen Speicherschutz.
Lokale Variablen werden auf einem gemeinsamen Stack gehalten.

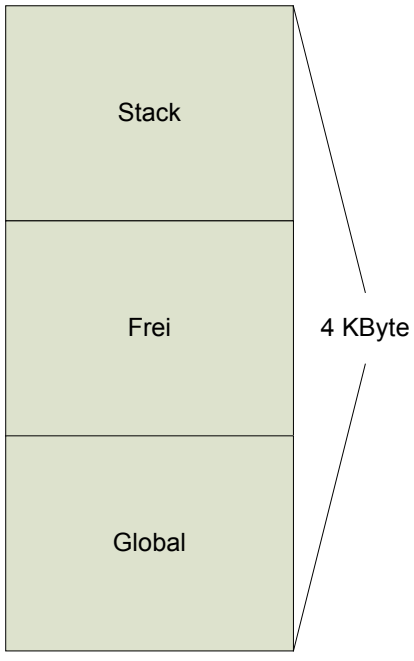


Abbildung 3.3 – TinyOS Speicher-Model

3.2.4. Kernel

Tiny-OS verfügt über keinen wirklichen Kernel, einzig der Scheduler kann als Kernel-Modul gesehen werden.
Die folgende Grafik soll dessen Aufbau und Funktionsweise erläutern.

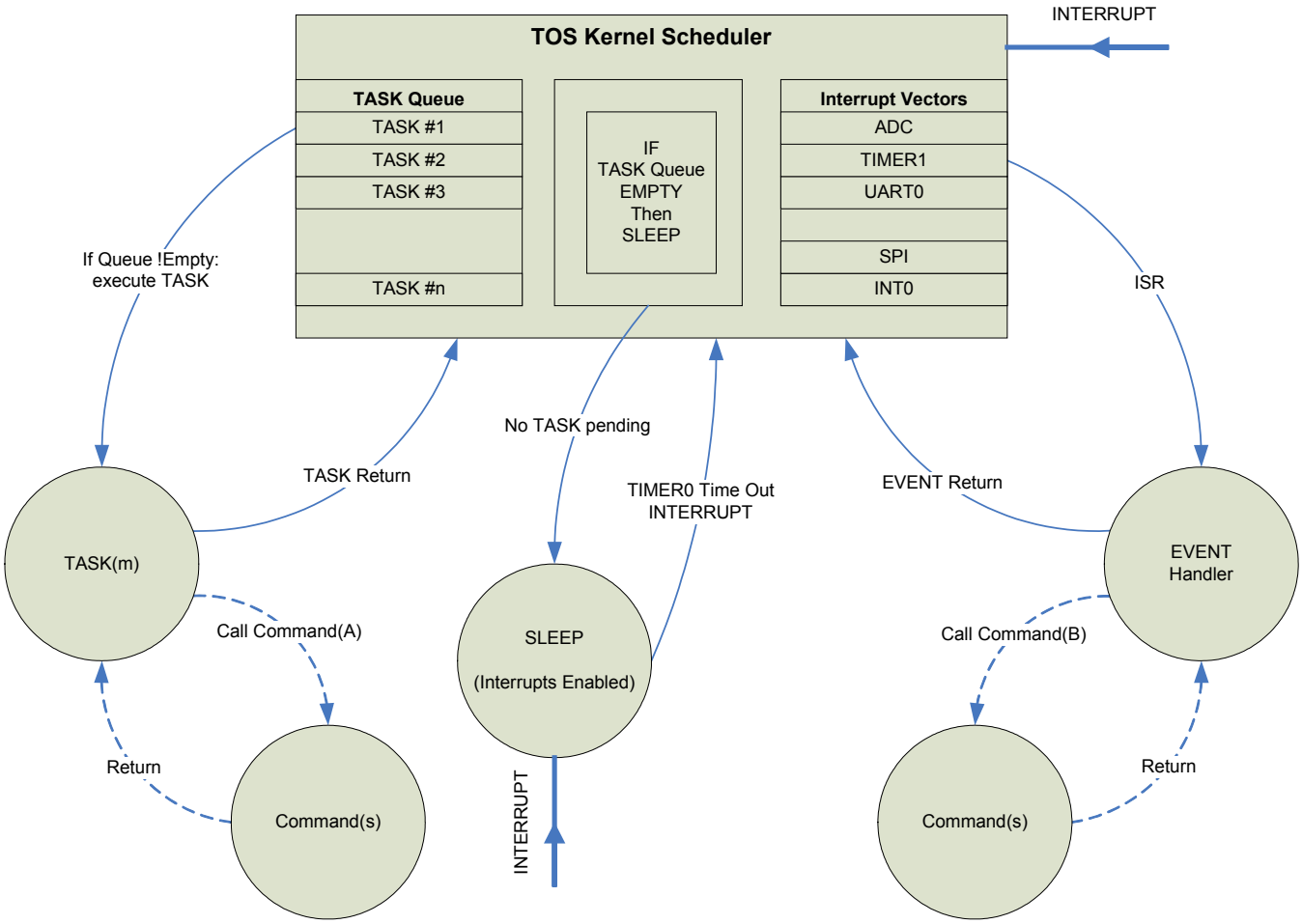


Abbildung 3.4 – TinyOS Kernel

Wie hier zu sehen, werden von einem Task oder Event so genannte Commands aufgerufen. Dessen Beschreibung folgt in einem späteren Abschnitt.
Wird kein Task oder Event bearbeitet, so wird der Prozessor in den Sleep-Modus versetzt, um Energie zu sparen.
Bei Eintreffen eines Interrupts wird dieser Zustand sofort verlassen und die entsprechende Aktion ausgeführt.

3.2.5. Aufbau

Jetzt soll der allgemeine Aufbau des Betriebssystems erläutert werden, beginnend mit einer Übersichts-Grafik.

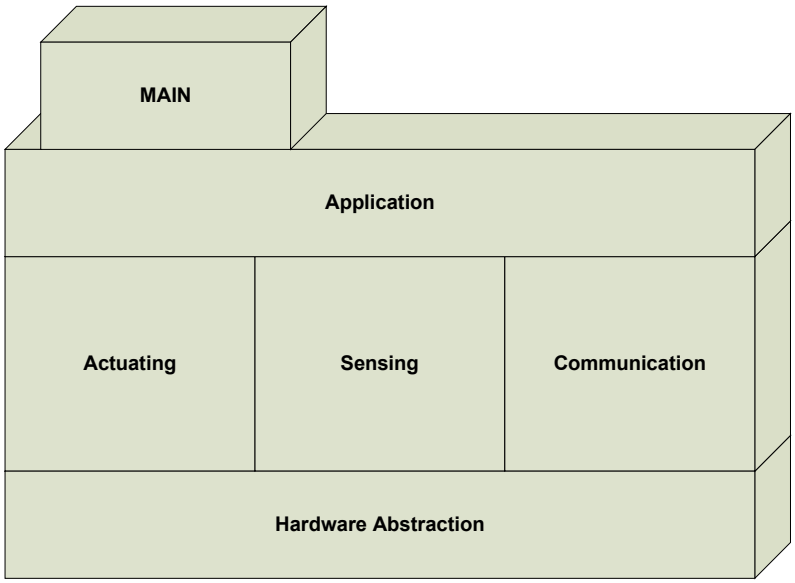


Abbildung 3.5 – TinyOS Aufbau

Folgend sollen die einzelnen Komponenten kurz beschrieben werden.

- **MAIN**
  Die Main beinhaltet prinzipiell nur den Scheduler, sie ist für den Entwickler uninteressant und verborgen.

- **Application**
  Hier findet das Anwendungsprogramm seinen Platz. Es ist natürlich dem Scheduler untergeordnet und hat Zugriff auf die darunter liegende Schicht für Aktoren, Sensoren und die Kommunikation via Funk, oder bedrahteten Kommunikations-Modulen, wie z.B. UART.
  Die Application-Schicht nutzt dabei Interfaces der darunter liegenden Schicht um auf diese Komponenten zuzugreifen.
  Für die Nutzung der meist Interrupt/Event-gesteuerten Komponenten müssen die jeweiligen Event-Handler und Callback-Funktionen vom Entwickler implementiert werden, hier existiert meist keine Standard-Implementation.

- **Actuating**
  Die Funktionalität dieser Module, z.B. Buzzer, wird dem Entwickler über definierte Interfaces zur Verfügung gestellt.

- **Sensing**
  Hier sitzen die einzelnen Sensor-Module, welche meist auf dem AnalogDigitalComperator aufsetzen.
  Der typische Ablauf:
  1) Der Anwender startet eine Sensor-Abfrage per Command
  2) Das Modul startet den ADC und wartet auf dessen Interrupt
  3) Bei Eintreten des Interrupts schickt sie der Application ein Event
  4) Das Anwendungsprogramm befindet sich im entsprechenden Event-Handler und hat Zugriff auf die angeforderten Daten

- **Communication**
  Das Kommunikations-Modul benutzt die verschiedenen Kommunikations-Geräte, wie z.B. den Onboard-UART, SPI..., und das extern angeschlossene Funk.
  Es stellt dem Entwickler die Daten auf Message-Ebene zur Verfügung.
  Der Entwickler muss sich also nicht um einzelne Bits, Bytes, Pakete und die Fehlererkennung kümmern.
  Außerdem stellt das Modul verschiedene Status-Informationen wie z.B. Signalstärke (Funk) bereit.
  Eine genauere Beschreibung der Funk-Kommunikation folgt zu einem späteren Zeitpunkt.

- **Hardware Abstraction**
  Die Hardware Abstraktion bietet eine weitestgehend hardwareunabhängiges Interface für die darüber liegende Schicht an.
  Hier enthalten sind.
  - LED (übernimmt Pinbelegung und Verdrahtung)
  - CLOCK (System-Timer mit Interrupt-Ausgabe)
  - UART (Baudraten-Einstellung, Übertragung)
  - ADC (Interrupt-Behandlung)
  - Funk (Timing, Konfiguration)

3.3. NesC

Der Compiler spielt bei diesem Betriebssystem eine entscheidende Rolle, er bietet nicht nur eine von vielen Möglichkeiten der Programm-Erstellung, sondern ist fest mit Tiny-OS verknüpft. Daher soll auch dieser beschrieben werden.

NesC ist ein erweiteter C-Compiler, welcher ein Komponenten-basiertes Konzept mit sich bringt und Tiny-OS somit effektiv nutzt.
Tiny-OS selbst wurde in Verbindung von C und NesC implementiert.

3.3.1. Basis-Konzepte

Folgend werden das Konzept von NesC V1.1 vorgestellt, hier verwendete Begriffe werden im darauf folgenden Kapitel erläutert:
(Diese Aussagen stammen größtenteils aus übersetzten Zitaten der NesC-Referenz)

- Die Programme bestehen aus Komponenten („components“), welche miteinander ‚verdrahtet’ werden, um das komplette Programm zu bilden.
  Die Komponenten bestehen aus zwei Teilen, der Spezifikation und der Implementation.

- Die Spezifikation des Verhaltens und Funktionalität der Komponenten wird durch einen Satz an Interfaces dargestellt.
  Interfaces werden von einer Komponente benutzt, oder bereitgestellt.
  Dabei repräsentieren bereitgestellte Interfaces die Funktionalität der Komponente.
  Die benutzten Interfaces repräsentieren die Funktionalität, die die Komponente für ihre Ausführung benötigt.

- Interfaces sind bidirektional:
  Sie spezifizieren einerseits eine Ansammlung von Funktionen, welche vom Interface-Provider implementiert werden muss („commands“).
  Gleichzeitig können sie aber auch eine Ansammlung von Funktionen sein, die vom Interface-Benutzer implementiert werden muss („events“).

  Dadurch ist es möglich, dass ein einzelnes Interface eine komplexe Interaktion zwischen Komponenten darstellt. Zum Beispiel wird bei Eintreffen eines Ereignisses, dessen Detektierung in einer System-Komponente implementiert ist, durch eine Call-back-Funktion an den Benutzer gereicht, die von diesem implementiert werden muss.
  Dieses ist sehr wichtig, da länger dauernde Commands in Tiny-OS nicht-blockierend implementiert sind und dessen Komplettierung durch Events signalisiert werden.
  Durch die Spezifizierung des Interfaces kann eine Komponente nicht z.B. das „sendPacket“ Command aufrufen, ohne gleichzeitig eine Implementation für das „sendDone“ Event zu besitzen.

Typischerweise werden Commands von „oben nach unten“, also von der Applikation in Richtung Hardware, und Events von „unten nach oben“, also Richtung Applikation gereicht. (siehe hierzu auch TinyOS Aufbau)

- Die Komponenten sind durch ihre Interfaces statisch mit anderen verbunden. Das verbessert die Ausführgeschwindigkeit und Effizienz, führt zu einem robustem Design, und erlaubt eine bessere statische Analyse des Programms.

- NesC wurde unter der Voraussetzung entwickelt, dass jeweils das gesamte Programm durch den Compiler übersetzt wird.
  Dieses fördert eine bessere Code-Generierung und Analyse, wie z.B. die Erkennung von „data-races“ während des Kompilierens.

- Das Zusammenspiel der NesC-Komponenten basiert auf den ununterbrechbaren Tasks, welche also immer bis zu ihrer Terminierung oder eigenständiger Unterbrechung laufen, und Interrupt-Handler, welche sich gegenseitig und Tasks unterbrechen können.
  Der NesC-Compiler erkennt potentielle „data-races“, welche durch Interrupt-Handler verursacht werden können.

3.3.2. Bestandteile

Im Folgenden sollen wichtige Bestandteile der Sprache NesC erläutert werden.

3.3.2.1. Interfaces

Wie bereits beschrieben, sind Interfaces in NesC bidirektional. Das bedeutet, dass sie einerseits dem Benutzer die Funktionalität eines Moduls offenbaren, auf der anderen Seite ihm aber auch angeben, welche Funktionen er selbst implementieren muss, um das gegebene Modul benutzen zu können.
Die im Modul implementierten Funktionen sind „Commands“, die vom Benutzer aufgerufen werden können.
Die selbst zu implementierenden Funktionen sind „Events“ und werden als Call-back-Funktionen benutzt.

Ein einfaches Beispiel für ein Interface:

```
interface SendMsg {
        command result_t send(uint16_t address, uint8_t length, TOS_MsgPtr msg);
        event result_t sendDone(TOS_MsgPtr msg, result_t success);
}
```

Bei diesem Interface kann der Benutzer die Funktion ‚send’ aufrufen. Da das Senden eines kompletten Paketes längere Zeit in Anspruch nimmt, wurde diese Funktion nicht-blockierend implementiert, was bedeutet, dass sie sofort zurückkehrt.
Der Rückgabe-Wert gibt nur Auskunft darüber, ob die Funktion ausgeführt wird, oder ob z.B. gerade eine andere Übertragung läuft.
Die Funktion ‚sendDone’ muss vom Benutzer implementiert werden! Sie wird vom jeweiligen Kommunikations-Modul aufgerufen, wenn die Übertragung beendet ist und dient somit als Call-back-Funktion.

3.3.2.2. Commands, Events

Commands sind Funktionen, deren Benutzung über Interfaces geregelt wird.
Soll also eine Funktion von anderen Komponenten benutzt werden können, so muss sie mit der Direktive ‚command' versehen werden.
Für Funktionen, die nur innerhalb einer Komponente benutzt werden, reicht die Standard-Syntax von C aus.
Commands werden mit dem Schlüsselwort ‚call' aufgerufen.

Events sind Software-Interrupts, die als Call-back-Funktionen dienen und wiederum über Interfaces bereitgestellt werden. Events in einem Komponenten-Interface geben dem Entwickler bekannt, dass er diese implementieren muss.
Events werden durch die Direktive ‚event' gekennzeichnet und können mit dem Schlüsselwort ‚signal' ausgelöst werden.

Durch die völlige Freiheit der Verbindungen zwischen beliebig vielen Komponenten kann ein ‚call' oder ‚signal' auch mehrere ‚Commands' oder ‚Events' ausführen.

3.3.2.3. Task

Ein Task ist ein kleines eigenständiges Programm, dass nach seinem Start bis zum Ende ausgeführt wird. Er kann durch Hardware-Interrupts unterbrochen werden.
Durch den simplen Tiny-OS-Scheduler können sich Tasks aber nicht gegenseitig unterbrechen.
Des Entwickler muss also darauf achten, dass ein Task nur eine kurze Bearbeitungszeit benötigt, da er für diese Zeit alle anderen blockiert.
Eine Funktion, die einen eigenen Task bilden soll wird wie folgt deklariert:
task void myTask() { ... }
Ein Task darf keine Parameter besitzen und der Rückgabewert muss void sein.
Durch das Schlüsselwort ‚post' wird ein Task in die Warteschlange gestellt und wartet dort bis zu seiner Ausführung, eine sofortige Ausführung ist durch den FIFO-Scheduler nicht möglich.
Ein Task kann innerhalb eines Commands, eines Events, oder innerhalb eines anderen Tasks per ‚post' für die Ausführung vorbereitet werden.

3.3.2.4. Components

NesC verwendet zwei verschiedene Code-Teile, einen für die Deklaration (genannt Konfiguration) und die andere für die Code-Implementation (genannt Modul).
Eine NesC-Komponente besteht somit aus einem **Modul**, oder einer **Konfiguration**.

- **Spezifikation**

Eine Applikation besteht aus einer oder mehrerer Komponenten, die zu einem lauffähigen Programm zusammengefügt werden.
Eine Komponente kann ein oder mehrere Interfaces benutzen und bereitstellen.
Dieses Interfaces sind der einzige Zugang zu dieser Komponente.
Weiterhin kann sie mehrere Instanzen eines einzelnen Interfaces benutzen, oder bereitstellen.
Dabei muss sie die „Commands“ der bereitgestellten Interfaces, und die „Events“ der benutzten Interfaces implementieren.

Ein kleines Beispiel:

```
module A1 {
        uses interface X;
        uses interface Y as Y1;
        provides {
                interface V;
                interface W;
        }
}
```

Das bedeutet, dass das Modul das Interface X und Y benutzt, wobei hier im Fall von Y1 eine Instanz mit diesem Namen von Y gebildet wird. Im Fall von X wird eine Instanz mit dem selben Namen gebildet.
Die Interfaces V und W werden bereitgestellt, dessen Funktionen müssen also noch implementiert werden.

- **Implementation**

Die Implementation findet auf zwei unterschiedliche Weisen statt.
Wobei die Module den ausführbaren Code für die bereitgestellten Interfaces implementieren und die Konfigurationen Interfaces mit, von anderen Komponenten bereitgestellten Interfaces, verbindet. Dieses Verbinden der Komponenten wird „wiring“ genannt.
Jede NesC-Applikation besitzt eine top-level-Konfiguration, die die Komponenten im innern verbindet.

Ein kleines Beispiel, welches die LEDs eines MOTEs blinken lässt, soll dieses erklären.

```
configuration Blink {
}
implementation {
        components Main, BlinkM, SingleTimer, LedsC;

        Main.StdControl -> BlinkM.StdControl;
        Main.StdControl -> SingleTimer.StdControl;
        BlinkM.Timer -> SingleTimer.Timer;
        BlinkM.Leds -> LedsC;
}
```

Die Konfiguration bekommt den Namen Blink und ist an sich leer, aber hier könnten auch Interfaces bereitgestellt (provides), oder benutzt (uses) werden.
In diesem Fall folgt aber sofort die Implementation und somit die Verbindung von Modulen.
Zuerst werden die benötigten Komponenten eingebunden, dann werden die Interfaces, die von Komponenten benutzt werden mit denen verbunden, die von anderen Komponenten bereitgestellt werden.
Somit wird z.B. das von BlinkM benutzte Leds-Interface mit dem Interface der LedsC-Komponente verbunden.

Grafisch sieht diese Verbindung dann folgendermaßen aus:

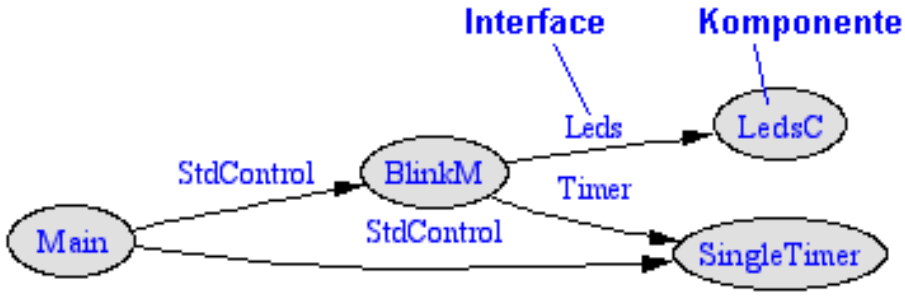


Abbildung 3.6 – Blink Wiring

Die Konfiguration und Verbindung der Komponenten ist abgeschlossen, nun muss der eigentliche Code implementiert werden.
Vorab werden im Modul die bereitgestellten und benutzen Interfaces deklariert.

```
module BlinkM {
        provides {
                interface StdControl;
        }
        uses {
                interface Timer;
                interface Leds;
        }
}
…
```

Da das Modul das Interface StdControl bereitstellt, müssen dessen Funktionen implementiert werden. Weiterhin benutzt es zwei Interfaces, dessen Commands aufgerufen werden können und dessen Events gleichzeitig implementiert werden müssen.

Code-Fortsetzung

```
...
implementation {

        command result_t StdControl.init() {
                call Leds.init();
                return SUCCESS;
        }

        command result_t StdControl.start() {
                return call Timer.start(TIMER_REPEAT, 1000) ;
        }

        command result_t StdControl.stop() {
                return call Timer.stop();
        }

        event result_t Timer.fired() {
                call Leds.redToggle();
                return SUCCESS;
        }
}
```

Wie hier zu sehen, werden die 3 Basis-Funktionen jeder Applikation vom Interface StdControl implementiert (init, start, stop). Diese werden automatisch vom System aufgerufen, bei der Initialisierung ‚init', beim Start ‚start' und beim Beenden der Applikation ‚stop'.
Außerdem wird ein Event des Interfaces Timer implementiert.
Dessen Funktion Timer.fired() ist somit die Call-back-Funktion, die von einer System-Komponente aufgerufen wird, wenn der Software-Timer überläuft.
In diesem Fall wird hier die rote LED getoggelt.
Somit ist diese kleine ‚Hello World'-Applikation lauffähig.

3.4. Kommunikation

Die Funk-Kommunikation der Geräte untereinander ist wahrscheinlich mit der wichtigste Punkt, da es sich schließlich um ein Drahtloses Sensornetzwerk handeln soll.
Aus diesem Grund soll die Funk-Schnittstelle und dessen Protokoll hier erläutert werden.

Die Datenübertragung erfolgt dabei transparent für den Entwickler, über das Betriebssystem.
Zum Einsatz kommt hier der Single-IC-Transceiver CC1000, der bereits im Kapitel Hardware vorgestellt wurde.
Dieser kann in einem beliebigen Band zwischen 300 MHz und 1000 MHz eingestellt werden (je nach Freigabe innerhalb eines Landes) und besitzt, zur sicheren Takterkennung, eine einschaltbare Manchester-Codierung.

3.4.1. Allgemeine Probleme

Jede Funk-Kommunikation muss mit verschiedenen Schwierigkeiten, wie z.B. dem hier dargestellten Multi-Path fading Effect, fertig werden.
Dabei handelt es sich um das Problem, dass zu einem Empfänger meist mehrere Funk-Strecken führen.
Dadurch kommt es zu Interferenzen und unterschiedliche Übertragungszeiten, auch bei konstantem Abstand.
Gleichzeitig kann es natürlich auch zu einem kompletten Ausfall führen.

Folgende Grafik soll das ersichtlich machen:

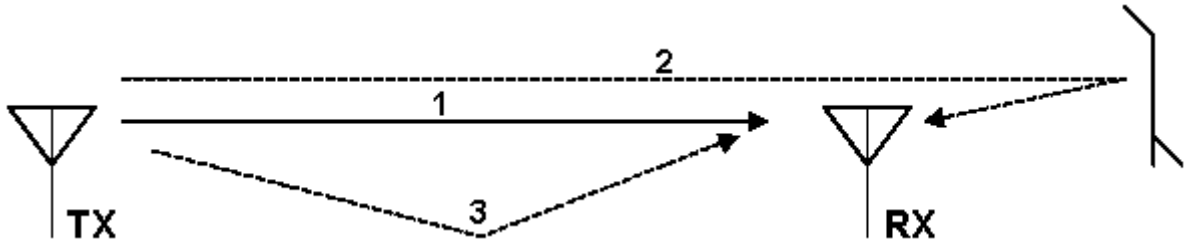


Abbildung 3.7 – Multi-Path Effect

Hier trifft ein Funk-Signal auf 1. Direktem Weg, 2.,3. über Reflektion auf den Empfänger.

Im Inneren von Gebäuden wirkt sich dieser Effekt besonders drastisch aus. Hier reduziert sich der Reichweite auf 1/3 der Maximalen, wenn sich das Gerät bewegt. Gleichzeitig ist auch die Beschaffenheit von Wänden, Boden und Fenstern von großer Bedeutung.

Weiterhin führen Kollisionen von Nachrichten, die immer stärker zunehmende Dichte von Funkquellen in den ISM-Bändern, und somit entstehende Interferenzen durch benachbarte Kanäle, zu Problemen.
Diesen Preis muss man aber zahlen, wenn ein lizenzfreies Band benutzt werden soll.

3.4.2. Protokoll

Folgend soll das Protokoll der Funkübertragung und das Datenformat von TinyOS-Nachrichten erklärt werden.

Nachrichten von diesem Typ werden über das Funk-Interface versendet:

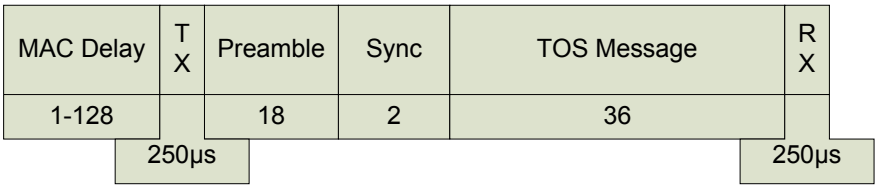


Abbildung 3.8 – TinyOS Frame

Feld	Bedeutung
MAC Delay	Der Sender wartet eine zufällige Zeit (bis zu 15 Paket-Längen) und prüft dann, ob das Medium frei ist (CSMA).
TX	Der Transmitter wird eingeschaltet.
Preamble	Er wird eine Preamble von 18 Byte der Form (10101) gesendet. Das dient der Taktsynchronisierung am Empfänger.
Sync	Mit dem Sync wird der Start der Nachricht bekannt gegeben.
TOS Message	TinyOS-Nachricht, siehe unten
RX	Der Transmitter wird abgeschaltet und der erfolgreiche Versand der Applikation signalisiert

Dabei erfolgt der Versand der gesamten Nachricht (Preamble, Sync, TOS Message) direkt vom Prozessor über das SPI-Interface an den dort angeschlossenen Funk-Transceiver. Dieser dient lediglich der (De)Codierung und (De)Modulierung.

Die TOS Message besteht dabei aus folgenden Teilen:

TOS-Header				User-Data	
Address	Active Message Type	Group ID	Payload Length	Payload	CRC
2	1	1	1	0..29	2

Abbildung 3.9 – TinyOS Message

Feld	Bedeutung
Address	Die Empfänger-Adresse des MOTEs (16 Bit), Ausnahme: - Eine reservierte Adresse für den UART vorhanden - Broadcast
Active Message Type	Nachrichten-Typ, vom Anwender wählbar
Group ID	Welcher Gruppe gehört das empfangende MOTE an. Von allen MOTEs, die nicht dieser Gruppe angehören, wird das Paket gefiltert.
Payload Length	Wie lang ist das Benutzerdatenfeld
Payload	Benutzerdaten mit maximal 29 Byte
CRC	Checksumme zur Fehlererkennung

3.4.3. Kommunikations-Stack

Im Betriebssystem verläuft die Kommunikation wie folgt:

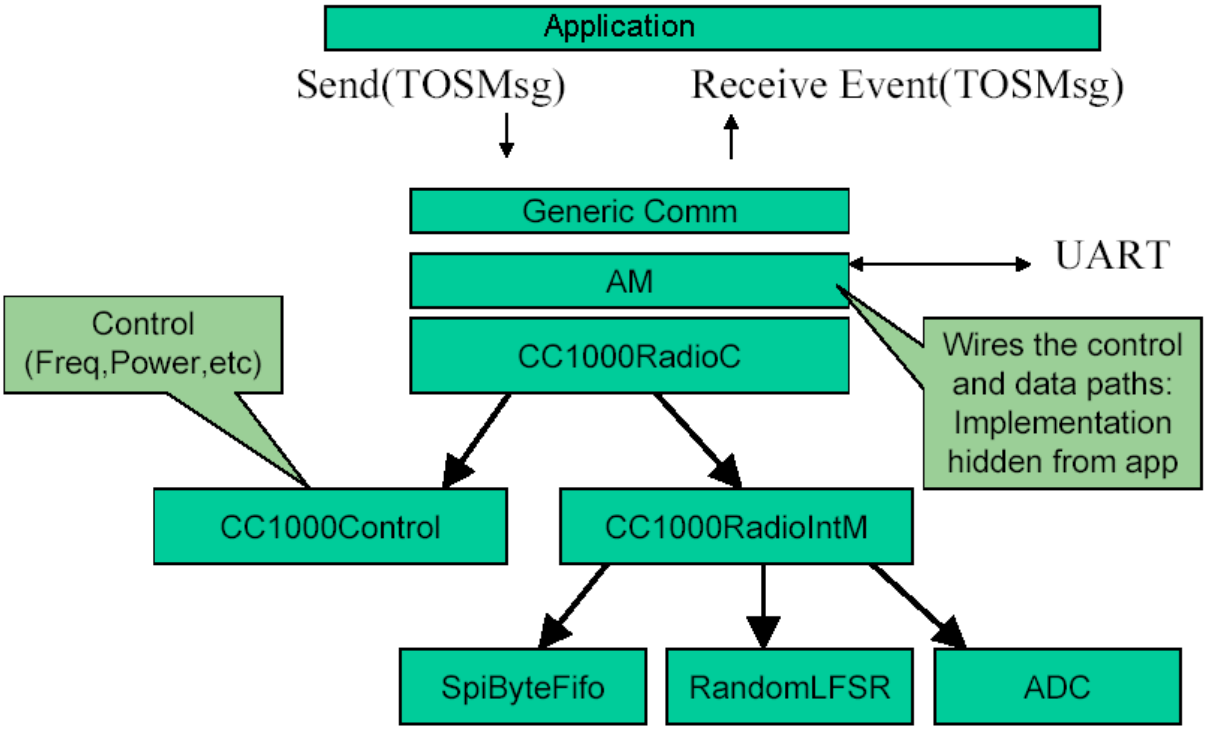


Abbildung 3.10 – TinyOS Kommunikations-Stack

Die zugrunde liegende Hardware wird durch das TinyOS-Modul „AM“ abstrahiert und kann somit durch Auswahl einer speziellen Adresse (siehe TOS Message) auch an das UART-Interface geschickt werden.
Dadurch kann sehr leicht eine andere Hardware, z.B. ein ZigBee-Transceiver, eingesetzt werden, ohne dass Änderungen in der Applikation nötig sind, hier reicht eine veränderte ‚Verdrahtung’ der Module.

Der Entwickler kann durch Aufruf des Commands ‚Send’ eine Nachricht verschicken und wird bei Empfang einer Nachricht durch das Event ‚Receive’ benachrichtigt.

Über das Modul ‚CC1000Control’ wird der Transceiver konfiguriert, also Codierung-Art, Frequenzband, Kanal und die Feldstärke, eingestellt.
Über das Modul ‚SpiByteFiFo’ erfolgt der eigentliche Versand zum Transceiver, die zu wartende zufällige Zeit vor dem Versand erfolgt mittels ‚RandomLFSR’.
Bei Empfang einer Nachricht erfolgt eine Feldstärken-Messung mittels ‚ADC’.

3.4.4. Fehlererkennung

Für den Entwickler ist es wichtig zu wissen, welche Fehler vom System erkannt und eventuell automatisch behoben werden.
TinyOS besitzt diesbezüglich eine Fehlererkennung mittels einer 16-Bit CRC.
Wird ein Paket korrekt empfangen, so wird automatisch ein Ack an den Absender geschickt.
Wird ein Fehler erkannt, so wird das entsprechende Paket verworfen und gilt als verloren.
Empfängt der Absender kein Ack, so weiß er, dass ein Fehler aufgetreten ist und das Paket wiederholt werden muss.

Fehlertyp	**Erkennung durch**	**Folge**
Bit-Fehler	TinyOS Link-Layer	Paket wird verworfen
Paket-Verlust	Applikation	Paket wird wiederholt

3.5. Fazit

Mit den frei verfügbaren und mitgelieferten Software-Komponenten stellt sich der Erfolg bei der Softwareentwicklung relativ schnell ein.
Dies wird besonders durch die existierenden Beispiel-Programme in Verbindung mit dem vorhandenem Tutorial erreicht.

Das Betriebssystem ist sehr schlank und daher schnell zu verstehen, das Konzept der Modularität und Ereignissteuerung wirkt gut durchdacht und fördert wiederum eine schnelle Entwicklung.

4. Applikation

Im Folgenden soll eine praktische Anwendung der vorgestellten Technologie realisiert werden.
Da nur begrenzt Geräte zur Verfügung standen, sowohl was die Prozessor-Boards, als auch die Einsatzmöglichkeiten vorhandener Sensor-Boards angeht, konnte keine praktisch einsetzbare Feldanwendung wie z.B. ein Umwelt-Monitoring, zu der z.B. Feuchtigkeits-Sensoren nötig wären, entstehen.
Durch das Vorhandensein eines Beschleunigungs-Sensors kam hier die Idee, ein Analyse-System für den Laufsport zu erstellen. Durch den möglichen günstigen Erwerb einer Pulsuhr wurde diese Idee verstärkt und sollte schließlich in die Tat umgesetzt werden.
Daher wurde das Standard-Einsatzgebiet, bei welchem sich die Sensoren statisch in einer Umgebung befinden, ausgeweitet, und in den Bereich der ‚Wearable Computer' überführt.

Es sollte also ein Analyse-System entstehen, mit dem ein Sportler seine Puls- und Schrittfrequenz, die während eines Laufes aufgezeichnet werden, grafisch analysieren kann.
Gleichzeitig kann auch ein privater Jogger sein Laufverhalten grafisch revue passieren lassen und entsprechend der gewünschten Zielzone (ein Pulsbereich für ein effektives Training) anpassen.
Dabei soll der Proband ein Gerät am Körper tragen (Wearable), welches die entsprechenden Daten zyklisch aufzeichnet und während des Laufs, oder danach, diese Daten zu einem PC überträgt.
Hier werden die Daten visualisiert und können durch einen Trainer schon während des Laufs analysiert, bzw. später vom Läufer selbst in Augenschein genommen werden.
Da ein PC kein Funk-Interface besitzt, und somit nicht direkt mit einem MOTE kommunizieren kann, muss ein Gateway dazwischengeschaltet werden.

Folgendes Bild soll das veranschaulichen.

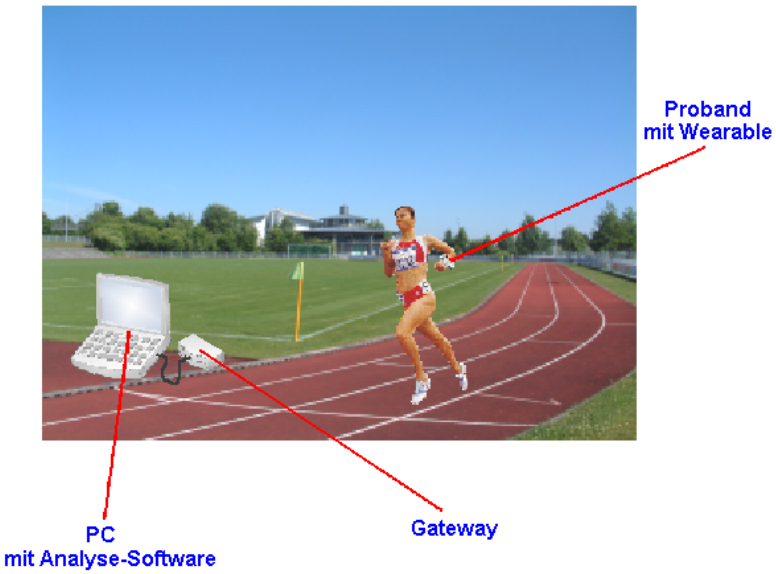


Abbildung 4.1 - Applikation

4.1. Analyse

In diesem Abschnitt soll das System analysiert und abstrakt beschrieben werden, um mit diesen Vorgaben einen konkreten Entwurf zu ermöglichen.

4.1.1. Zielsetzung

Das System soll auf Basis von MOTEs mit möglichst wenig externer Hardware realisiert werden. Es soll die Daten für die Herzfrequenz und Schrittfrequenz zyklisch aufzeichnen und persistent speichern, dass heißt, die Daten sollen auch nach einem Lauf und nach Entzug der Stromversorgung erhalten bleiben.
Das Gerät soll eine Abgrenzung der Datenaufzeichnung besitzen, es muss also möglich sein, zu einem gewünschten Zeitpunkt die Aufzeichnung zu starten und zu stoppen.
Es soll die Möglichkeit bestehen, mehrere Läufe aufzuzeichnen um diese später miteinander vergleichen zu können.
Da es sich bei der verwendeten Technologie um ein drahtloses Sensornetzwerk handelt, sollen auch mehrere Läufer gleichzeitig überwacht werden können.
Es wird im Rahmen dieser Arbeit nur die Entwicklung eines Prototyps angestrebt, die Erweiterung auf mehrere Geräte sollte aber ohne weiteres möglich sein.
Die PC-Software zur Analyse soll die Daten grafisch darstellen und die Möglichkeit der Speicherung besitzen.

4.1.2. Systemüberblick

Das System besteht aus einem tragbaren Gerät zur Datenaufzeichnung, einem Gateway zur Anbindung eines PCs an das Sensor-Netzwerk und der PC-Software zur Visualisierung.

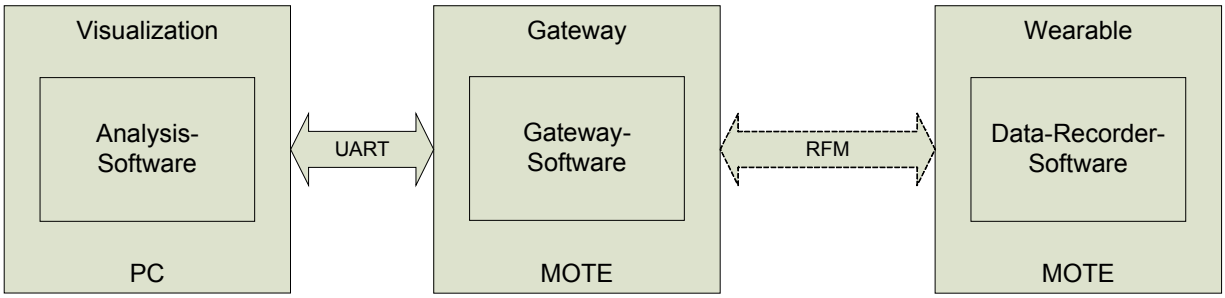


Abbildung 4.2 – Applikation Systemüberblick

4.1.3. Externe Schnittstellen

Die einzelnen Komponenten besitzen folgende Schnittstellen:

- **PC**
  - UART zur Kommunikation mit dem Gateway
  - GUI zur Visualisierung der Daten und Benutzereingaben

- **Gateway**
  - UART zur Kommunikation mit dem PC
  - RFM, Funk-Interface zur Kommunikation mit dem (den) Wearable(s)

- **Wearable**
  - RFM, Funk-Interface zur Kommunikation mit dem Gateway
  - Benutzerinterface zur Status-Anzeige und Starten/Stoppen der Datenaufzeichnung

4.1.4. Funktionale Anforderungen

Die Komponenten müssen folgende Anforderungen erfüllen:

- **PC**
  - Kommunikation
    mit dem Gateway über ein UART-Interface
    - Bei einer grob definierten Entfernung soll, bei laufender Datenaufzeichnung im Wearable, ein automatischer Datenempfang ermöglicht werden
  - Visualisierung
    Herzfrequenz und Schrittfrequenz
  - Online / Offline – Betrieb
    Anzeige von aktuellen Aufzeichnungs-Teilen, als auch zuvor übertragenen kompletten Aufzeichnungen

- **Gateway**
  - Passiv
    Es soll kein Datenverkehr initiiert werden.
  - Kommunikation
    - Umsetzung von Funk-Paketen auf UART
    - Umsetzung von UART-Paketen auf Funk
  - Visualisierung
    Der Datenverkehr über Funk und UART soll ersichtlich sein.

- **Wearable**
    - Datenerfassung
      Erfassung und Berechnung der Herz- / Schrittfrequenz pro Minute
    - Datenaufzeichnung
      Soll vom Benutzer gestartet / gestoppt werden können.
      Datenspeicherung im Sekundentakt.
      Persistente Speicherung von mehreren Datensätzen.
    - Kommunikation
      mit dem Gateway über ein Funk-Interface
        - Bei laufender Datenaufzeichnung soll ein automatischer Datenversand initiiert werden
    - Visualisierung
      Anzeige einer Datenaufzeichnung

4.1.5. Leistungsanforderungen

Die Komponenten sollen möglichst effektiv arbeiten, dabei sind auch die Ressourcen der verwendeten Hardware zu beachten.
Für die nachhaltige Nutzung der erfassten Daten sollten diese nachvollziehbar und möglichst präzise sein.
Die Kommunikation zwischen dem Wearable und dem PC muss sichergestellt werden. Es dürfen keine defekte Pakete am Empfänger akzeptiert werden, noch darf es zu einem Paketverlust kommen, da in diesen Fällen die sinnvolle Nutzung der Daten nicht mehr gegeben wäre.

4.2. Entwurf

Im Entwurf soll nun ausgehend von der nötigen Hardware das System entwickelt werden.
Dabei soll der Entwurf für die einzelnen Komponenten des System (Wearable, Gateway, PC) getrennt beschrieben werden.

4.2.1. Hardware

Da die Software für eine spezielle Hardware entwickelt werden muss, soll zunächst die Hardware der einzelnen Komponenten beschrieben werden.

4.2.1.1. Wearable

Als Wearable soll ein Mica2Dot-Board zum Einsatz kommen, da es, von seinen Abmessungen her, für diesen Einsatz sehr gut geeignet ist. Gleichzeitig steht ein Sensor-Board mit einem 2-Achsen-Beschleunigungsmesser zur Verfügung (MTS510), mit dessen Hilfe sich ein Schrittmesser realisieren lässt.
Ein Sensor zur Herzfrequenz-Messung steht nicht direkt zur Verfügung, sodass eine handelsübliche Pulsuhr benutzt werden muss, welche hierfür einen Brustgurt besitzt. Die Übertragung des Herzschlags erfolgt dabei frequenzmoduliert, direkt ohne Protokoll, zur Uhr.
Durch Messungen an dieser wurde ein Punkt hinter dem Filter und Verstärker ermittelt, welcher den Herzschlag digital ausgibt.
Über einen digitalen Port kann also die Uhr mit dem MOTE verbunden werden.
Hierzu kommt ein weiteres Sensor-Board (MDA500) zum Einsatz, da dieses über ein Prototyp-Feld verfügt, auf welchem eigene Sensoren realisiert und an ein Mica2Dot-Board angeschlossen werden können.
Somit ermöglicht dieses Prototyp-Feld auch eine Erweiterung des Mica2Dot User-Interfaces (1 LED) durch weitere LEDs und Taster.
Die Sensor-Boards lassen sich miteinander kombinieren, sodass das Wearable per Sandwich-Bauweise realisiert werden kann.

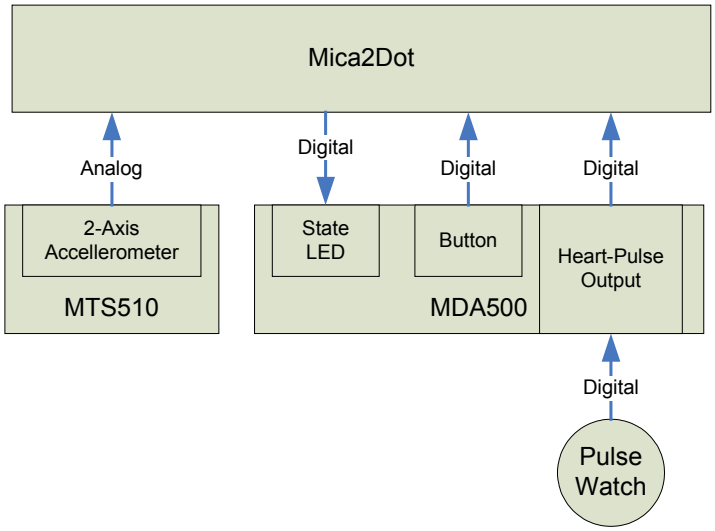


Abbildung 4.3 – Entwurf Wearable Hardware

4.2.1.2. Gateway

Als Gateway soll ein Mica2-Board zum Einsatz kommen, da die serielle Datenübertragungsgeschwindigkeit hier 57,6 KBit/s beträgt, statt 19,2 KBit/s beim Mica2Dot.
Da ein Mica2-Board nicht direkt an einen PC angeschlossen werden kann (kein RS232-Transceiver und COM-Anschluss vorhanden), wird dieses an ein Serielles Gateway-Board (MIB510) angeschlossen.
Dieses Board besitzt den notwendigen RS232-Transceiver zur Pegelwandlung und einen COM-Anschluss, sodass jetzt per Seriellen Kabel eine Verbindung zum PC hergestellt werden kann.

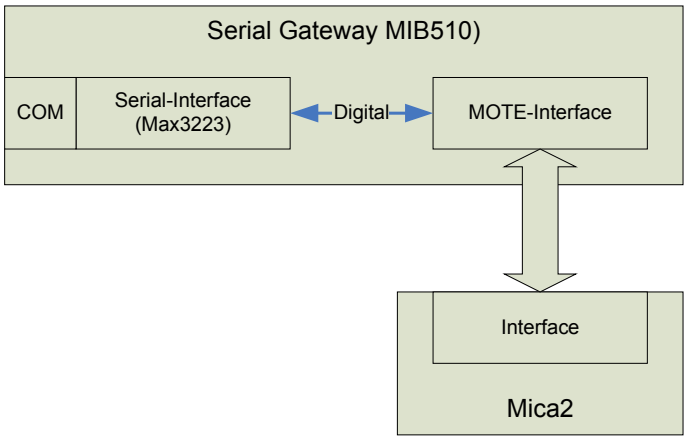


Abbildung 4.4 – Entwurf Gateway Hardware

4.2.1.3. PC

Die Hardware des PCs bedarf keiner näheren Betrachtung, bis auf die Voraussetzung, dass ein Standard-UART, also Serieller COM-Anschluss zur Verfügung stehen muss.

4.2.2. Software

Da jetzt die benötigte Hardware definiert ist, sollen nun die einzelnen Komponenten mit ihren benötigten Modulen beschrieben werden.

4.2.2.1. Wearable

Das Software-Paket für den Wearable basiert auf dem Betriebssystem TinyOS und besitzt eine ganze Reihe von verwendeten Modulen.
In diesem Rahmen sollen nur die selbst entwickelten Module, und von diesen direkt benutzte, dargestellt und erläutert werden, da die Hierarchisierung der Module in TinyOS sehr komplex ist und die vervollständigte Darstellung sehr unübersichtlich werden würde.
An dieser Stelle soll schließlich die Software und nicht das Betriebssystem erläutert werden.

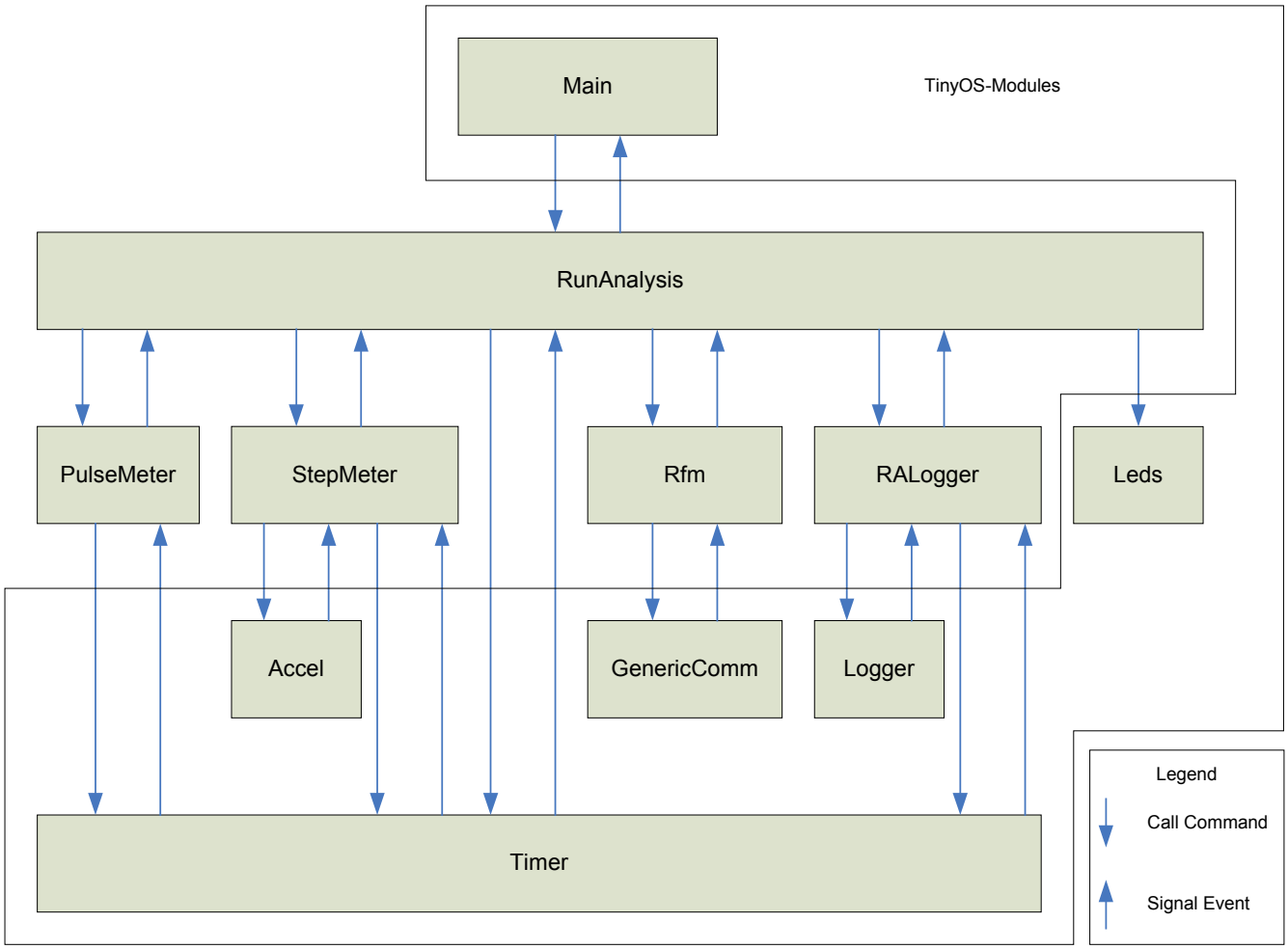


Abbildung 4.5 – Entwurf Wearable Software

- **Systemkomponenten**

Im Folgenden sollen kurz die direkt benutzten System-Module von TinyOS beschrieben werden.

Main

Das Main-Modul ist die Basis jeder Applikation für TinyOS. Hier enthalten ist der Scheduler.

Timer

Dieses Modul stellt dem Entwickler, abgeleitet von einem Hardware-Timer, bis zu 256 Software-Timer zur Verfügung, die zyklisch oder einmalig gestartet und gestoppt werden können. Bei Ablauf signalisieren sie ein Event.
Das Modul spielt bei dieser Applikation eine zentrale Rolle, es wird von fast allen eigenen Modulen benutzt. Somit lassen sich zyklische, oder zeitlich aufeinander folgende Abläufe sehr leicht realisieren.

Accel

Dieses Modul abstrahiert einen 2-Achsen Beschleunigungssensor, welcher sich auf dem verwendetem Sensor-Board MTS510 befindet. Im Prinzip verbindet es nur logisch die analogen Ausgänge des Sensors mit dem AnalogDigitalConverter des Prozessors.
Einmal initialisiert, liefert es auf Anfrage ein Event pro Kanal mit dem vom ADC gemessenen Wert.

GenericComm

Dieses Modul abstrahiert das Funk-Modul. Es bietet im Prinzip nur eine Funktion zum Senden eines Paketes an. Über je ein Event wird signalisiert, wenn ein Paket versendet wurde, bzw. ein Paket empfangen wurde.

Logger

Dieses Modul abstrahiert ein Daten-Flash, das beschrieben und gelesen werden kann. Das Modul bietet Schreib- und Lese-Funktionen, dessen Abschluss mit je einem Event signalisiert wird.
Es bietet sowohl sequenzielles als auch wahlfreies Schreiben und Lesen an, jedoch kann jeweils nur auf eine komplette Zeile, bestehend aus 16 Bytes, zugegriffen werden.
Es biete Zugriff auf insgesamt 32752 Zeilen und somit auf 512 KByte –256 Byte. Diese 256 Byte am Anfang des Speichers sind für spezielle Funktionen reserviert.

Leds

Dieses Modul abstrahiert den Zugriff auf 3 LEDs (Mica2), oder 1 LED (Mica2Dot). Durch einfache Commands können die Leds ein-, ausgeschaltet, oder getoggelt werden.
Der Entwickler kann hierdurch (sehr begrenzt) den internen Zustand des Systems anzeigen.

- **Eigene Komponenten**

Nachfolgend sollen die eigenen, zu erstellenden, Module spezifiziert werden:

PulseMeter

Dieses Modul dient zur Messung der Herzfrequenz.
Das Modul bedient sich nur einer weiteren Komponente, dem TinyOS-Modul Timer.
Das abgebildete Use-Case-Diagramm soll die Funktionen und Abhängigkeiten darstellen.

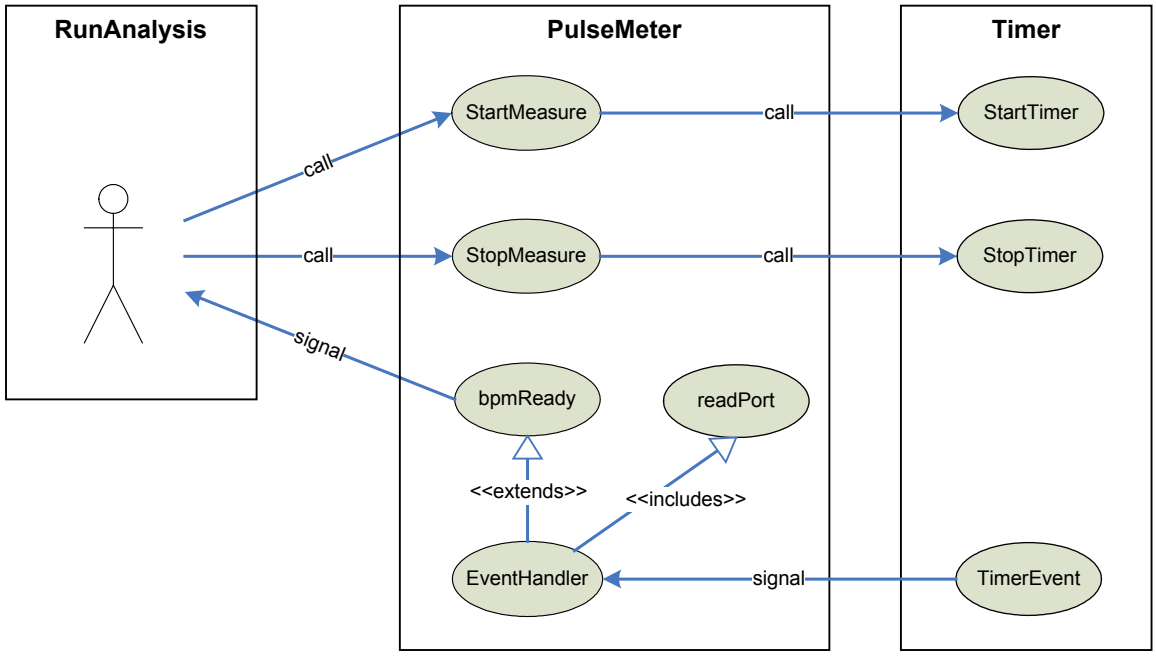


Abbildung 4.6 – Entwurf Wearable-Modul PulseMeter

- Mit der Funktion ‚StartMeasure' wird das Modul aktiviert. Dabei wird ein zyklischer Software-Timer gestartet.
  Läuft der Timer über, wird ein Event ausgelöst, und der digitale Messeingang abgefragt. Dabei wird die Zeit zwischen zwei Herzschlägen ermittelt und nach Durchlauf eines Filters, zur präziseren Messung, die Pulsfrequenz / Minute berechnet.
- Der hierbei ermittelte plausible Wert wird per Event der nächst höheren Schicht, hier RunAnalysis, signalisiert.
- Mit der Funktion ‚StopMeasure' wird das Modul angehalten und verbraucht keine Prozessorzeit mehr.
- Da TinyOS kein dynamisches Speichermanagement besitzt, bleibt die RAM-Nutzung konstant.
- Das Modul arbeitet aktiv, das heißt, nach der Aktivierung benötigt es keine weiteren Eingaben und liefert selbstständig Ereignisse.

StepMeter

Dieses Modul dient zur Messung der Schrittfrequenz.
Das Modul bedient sich zwei weiteren Komponenten, dem TinyOS-Modul Timer und Accel für den 2-Achsen Beschleunigungssensor.
Das abgebildete Use-Case-Diagramm soll die Funktionen und Abhängigkeiten darstellen.

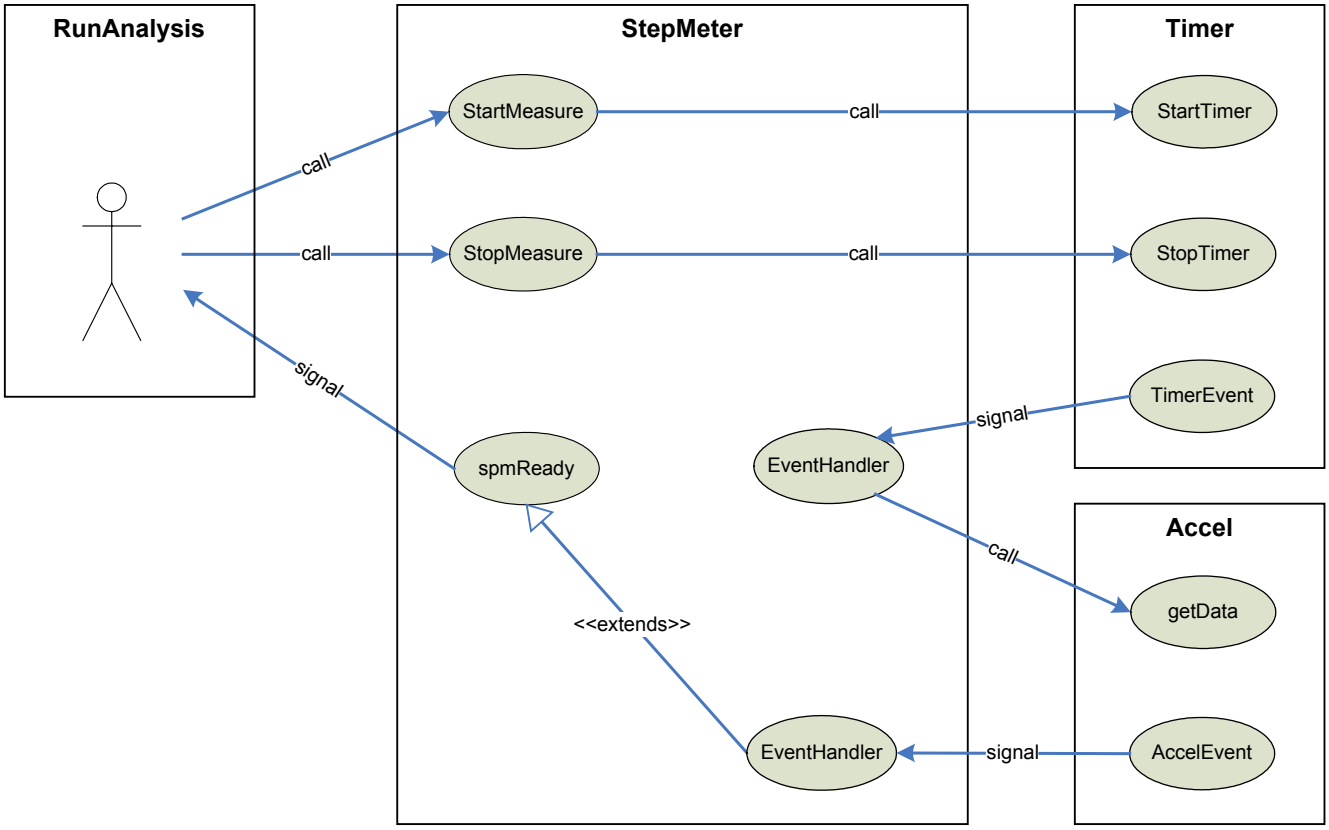


Abbildung 4.7 – Entwurf Wearable-Modul StepMeter

- Mit der Funktion ‚StartMeasure' wird das Modul aktiviert. Dabei wird ein zyklischer Software-Timer gestartet.
  Läuft der Timer über, wird ein Event ausgelöst, und der Beschleunigungssensor per ADC abgefragt.
  Mit dessen Hilfe wird eine Schrittdetektion durchgeführt und gleichzeitig die Zeit zwischen zwei Schritten gemessen.
- Nach Durchlauf eines Filters, zur präziseren Messung, wird die Schrittfrequenz / Minute berechnet und bei Plausibilität per Event der nächst höheren Schicht, hier RunAnalysis, signalisiert.
- Mit der Funktion ‚StopMeasure' wird das Modul angehalten und verbraucht keine Prozessorzeit mehr.
- Da TinyOS kein dynamisches Speichermanagement besitzt, bleibt die RAM-Nutzung konstant.
- Das Modul arbeitet aktiv, das heißt, nach der Aktivierung benötigt es keine weiteren Eingaben und liefert selbstständig Ereignisse.

Rfm

Dieses Modul ist eine Erweiterung zum verwendeten TinyOS-Modul GenericComm.
Es dient zum Senden und Empfangen von Nachrichten über das Funk-Modul.
Das abgebildete Use-Case-Diagramm soll die Funktionen und Abhängigkeiten darstellen.

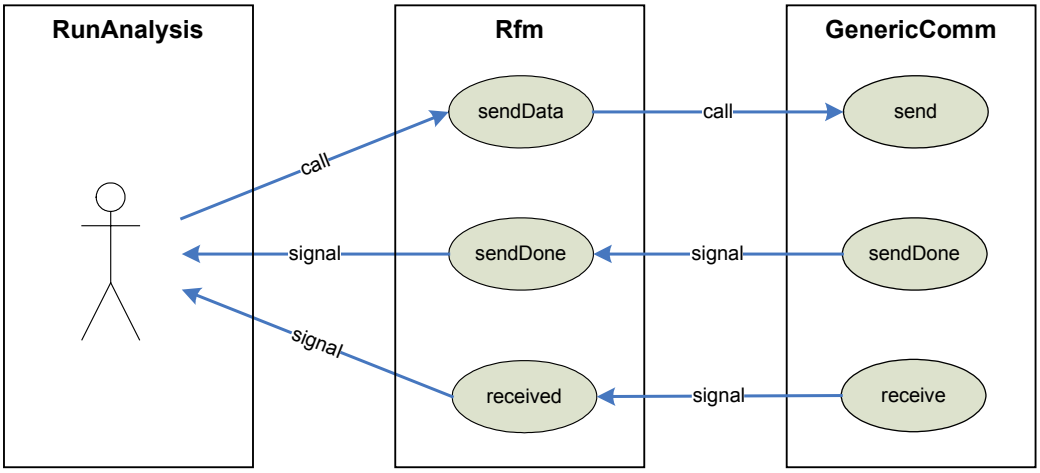


Abbildung 4.8 – Entwurf Wearable-Modul Rfm

- Die Funktion ‚sendData' sendet eine Nachricht durch den Aufruf der entsprechenden GenericComm-Funktion. Hierbei wird nach den unterschiedlichen Nachrichten-Typen unterschieden. Hierzu später mehr. Wurde diese Nachricht erfolgreich an das Betriebssystem übergeben, so wird von GenericComm ein Event ausgelöst. Das Rfm-Modul unterstützt zu jeder Zeit nur eine zu sendende Nachricht.
  Erst wenn diese Nachricht erfolgreich versendet und das entsprechende Event ausgelöst wurde, darf eine weitere Nachricht versendet werden.
- Aus diesem Grund wird der erfolgreiche Versand auch per Event an die nächst höhere Schicht, hier RunAnalysis, signalisiert.
- Wird eine Nachricht empfangen, so wird dieses ebenfalls durch ein Event von GenericComm signalisiert. Diese Nachricht wird dann direkt per Event an RunAnalysis weitergereicht.
- Das Modul arbeitet passiv, das heißt, es wartet auf externe Ereignisse, entweder ein Kommando zum Senden einer Nachricht, oder den Empfang einer Nachricht, bevor es selbst Ereignisse liefert.

RALogger

Dieses Modul bietet eine Erweiterung zum TinyOS-Modul Logger. Es dient dem zeilenweisen (16 Byte) Lesen und Schreiben von Datensätzen in einen nichtflüchtigen Speicher (Flash).
Dabei übernimmt es die Organisation von kompletten Aufzeichnungssätzen (Logs genannt), das heißt die Zusammenfassung aller Daten, die zwischen dem Starten und Stoppen einer Aufzeichnung gespeichert wurden.

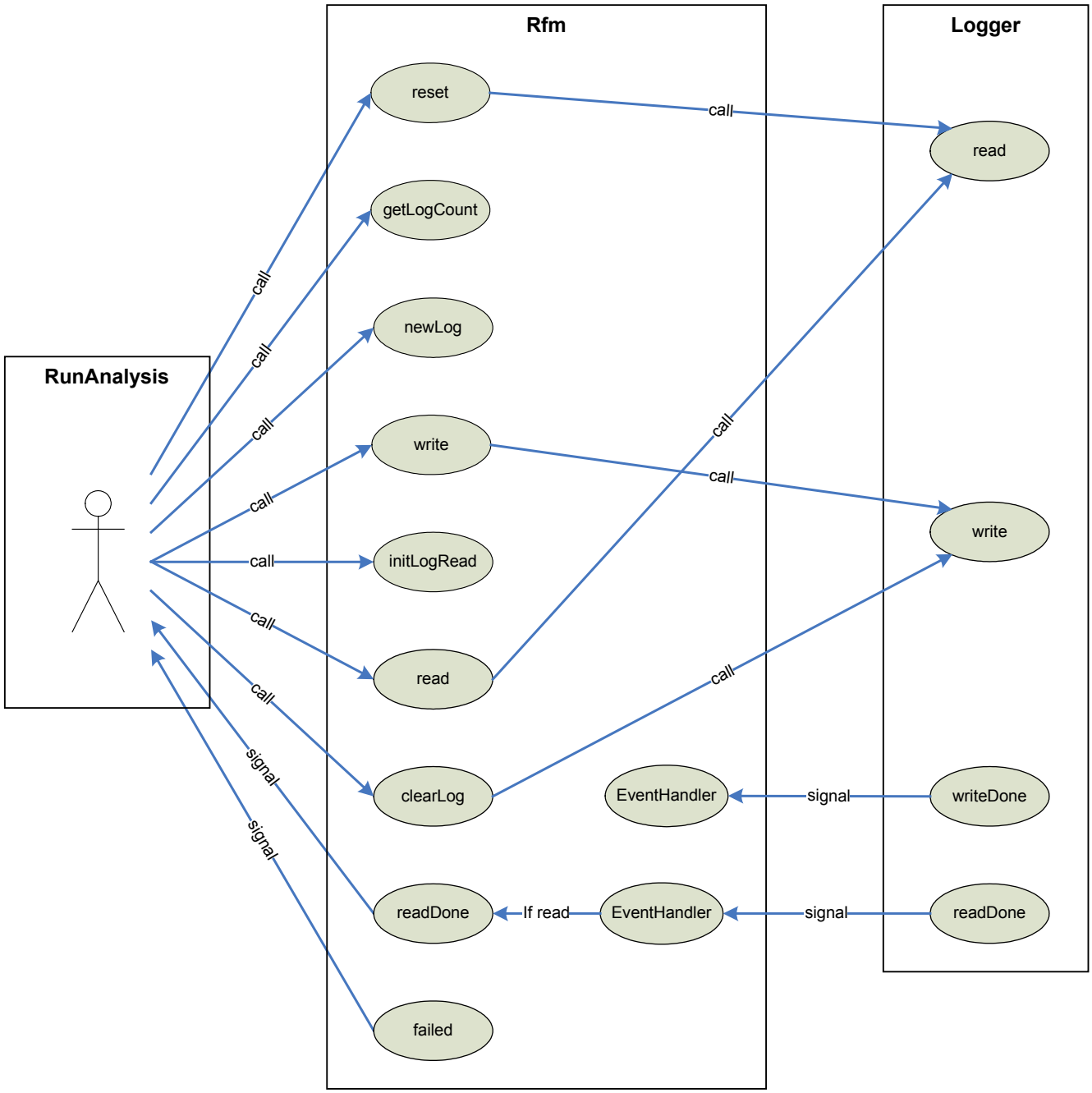


Abbildung 4.9 – Entwurf Wearable-Modul RALogger

- Die Funktion ‚reset' dient zur Erkennung der gespeicherten Logs nach dem Einschalten.
- ‚GetLogCount' liefert die Anzahl der gespeicherten Aufzeichnungssätze.
- ‚NewLog' initialisiert einen neuen Aufzeichnungssatz.

- ‚write' schreibt einen Datensatz (1 Zeile) in das Flash
- ‚initLogRead' initialisiert das Modul zum Lesen eines bestimmten Logs.
- ‚read' liest einen Datensatz (1 Zeile) aus dem Flash.
- ‚clearLog' dient zum Löschen aller zuvor gespeicherten Aufzeichnungsdaten.
- Bei Abschluss einer Leseoperation wird vom Modul Logger ein Event signalisiert.
  Wurde dieser Datensatz aufgrund des Commands ‚read' gelesen, so wird der nächst höheren Schicht, hier RunAnalysis, dieses Datum per Event signalisiert.
- Tritt während einer Lese- oder Schreiboperation ein Fehler auf, so wird dieses der Applikation per Event signalisiert.
- Das Modul arbeitet passiv, das heißt, es bietet eine reine Befehlsverarbeitung, ohne eigenständige Ereignisse.

RunAnalysis

Dieses Modul bildet die Basis der Applikation. Es bedient sich der vorgestellten Module, und verbindet dessen Fähigkeiten miteinander.
Somit können die Sensoren (Pulsmesser, Schrittmesser) ausgelesen, dessen Daten gespeichert, wieder ausgelesen und per Funk übertragen werden.
Das Modul muss eine sichere Datenübertragung gewährleisten, dass heißt, eine Neuaufsetzung bei Paketverlust.
Gleichzeitig muss es das Applikations-Protokoll implementieren (siehe Kapitel Kommunikation) und bei laufender Datenaufzeichnung eine Kommunikation per Ping initiieren und anschließend alle bis dahin verfügbaren Daten versenden.
Außerdem muss es für eine sichere und eindeutige Speicherung der Analysedaten sorgen (siehe Kapitel Analyse-Daten).

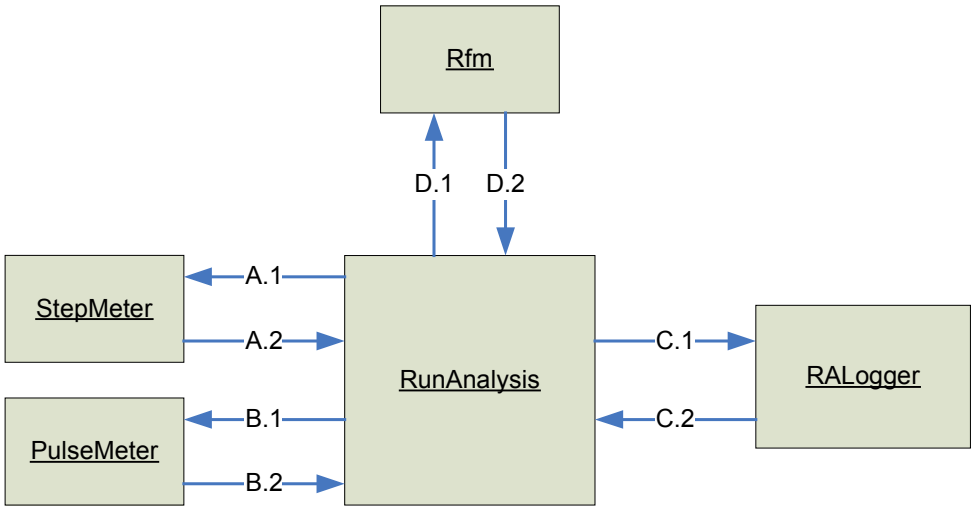


Abbildung 4.10 – Entwurf Wearable-Modul RunAnalysis

- A.1 Befehl, das Modul zu starten, bzw. stoppen
- A.2 Event, Datum der berechneten Schrittfrequenz
- B.1 Befehl, das Modul zu starten, bzw. stoppen
- B.2 Event, Datum der berechneten Pulsfrequenz
- C.1 Befehl, Datensatz schreiben, neuen Log anlegen, ...
- C.2 Event, Result, gelesener Datensatz, Anzahl der Logs
- D.1 Daten senden
- D.2 Befehl empfangen

4.2.2.2. Gateway

Das Software-Paket für das Gateway basiert ebenfalls auf dem Betriebssystem TinyOS und verwendet nur wenige Module.
In diesem Rahmen sollen nur die selbst entwickelten Module, und von diesen direkt benutzte, dargestellt und erläutert werden, da die Hierarchisierung der Module in TinyOS sehr komplex ist und die vervollständigte Darstellung sehr unübersichtlich werden würde.

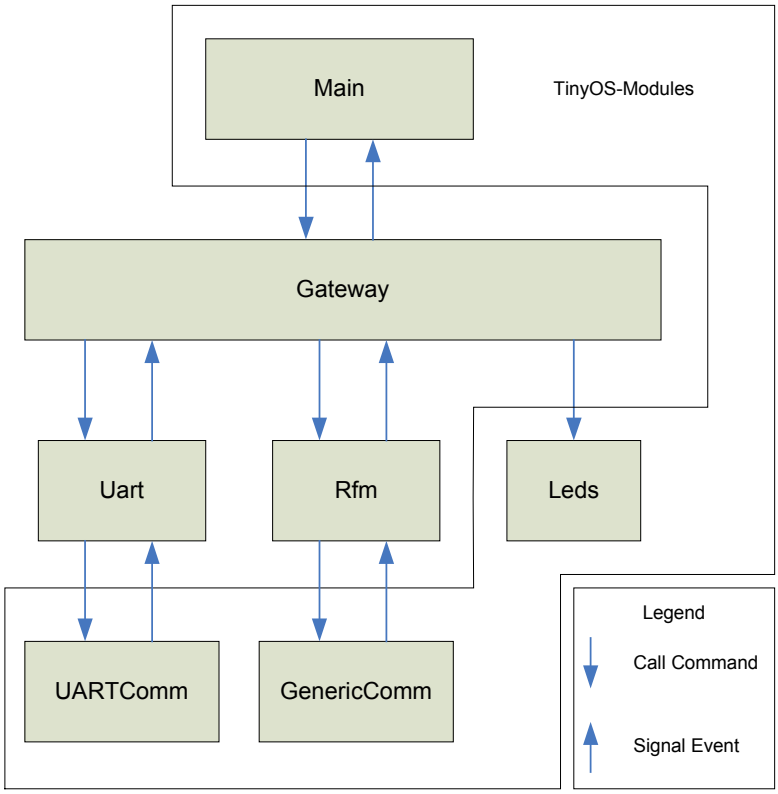


Abbildung 4.11 – Entwurf Gateway Software

- **Systemkomponenten**

Im Folgenden sollen kurz die direkt benutzten System-Module von TinyOS beschrieben werden.

Main
Das Main-Modul ist die Basis jeder Applikation für TinyOS. Hier enthalten ist der Scheduler.

UartComm
Dieses Modul abstrahiert das Uart-Modul. Es bietet im Prinzip nur eine Funktion zum Senden eines Paketes an. Über je ein Event wird signalisiert, wenn ein Paket versendet wurde, bzw. ein Paket empfangen wurde.

GenericComm
Dieses Modul abstrahiert das Funk-Modul. Es bietet im Prinzip nur eine Funktion zum Senden eines Paketes an. Über je ein Event wird signalisiert, wenn ein Paket versendet wurde, bzw. ein Paket empfangen wurde.

Leds

Dieses Modul abstrahiert den Zugriff auf 3 LEDs (Mica2), oder 1 LED (Mica2Dot). Durch einfache Commands können die Leds ein-, ausgeschaltet, oder getoggelt werden.
Der Entwickler kann hierdurch (sehr begrenzt) den internen Zustand des Systems anzeigen.

- **Eigene Komponenten**

Nachfolgend sollen die eigenen, zu erstellenden, Module spezifiziert werden:

Uart

Dieses Modul ist eine Erweiterung zum verwendeten TinyOS-Modul UARTComm. Es dient zum Senden und Empfangen von Nachrichten über eine Uart-Schnittstelle.
Das abgebildete Use-Case-Diagramm soll die Funktionen und Abhängigkeiten darstellen.

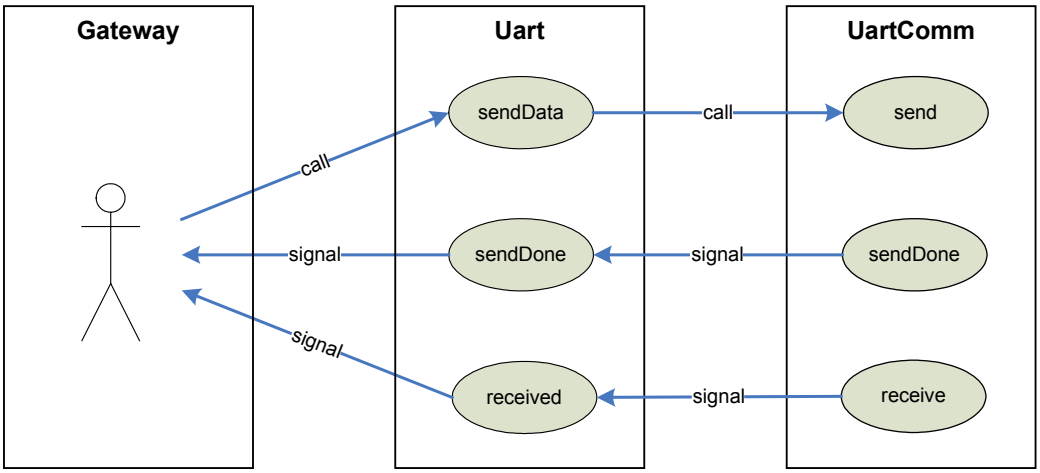


Abbildung 4.12 – Entwurf Gateway-Modul Uart

- Die Funktion ‚sendData' sendet eine Nachricht durch den Aufruf der entsprechenden UartComm-Funktion.
  Wurde diese Nachricht erfolgreich an das Betriebssystem übergeben, so wird von UartComm ein Event ausgelöst. Das Uart-Modul unterstützt zu jeder Zeit nur eine zu sendende Nachricht.
  Erst wenn diese Nachricht erfolgreich versendet und das entsprechende Event ausgelöst wurde, darf eine weitere Nachricht versendet werden.
- Aus diesem Grund wird der erfolgreiche Versand auch per Event an die nächst höhere Schicht, hier Gateway, signalisiert.
- Wird eine Nachricht empfangen, so wird dieses ebenfalls durch ein Event von UartComm signalisiert. Diese Nachricht wird dann direkt per Event an Gateway weitergereicht.
- Das Modul arbeitet passiv, das heißt, es wartet auf externe Ereignisse, entweder ein Kommando zum Senden einer Nachricht, oder den Empfang einer Nachricht, bevor es selbst Ereignisse liefert.

Rfm

Dieses Modul ist eine Erweiterung zum verwendeten TinyOS-Modul GenericComm.
Es dient zum Senden und Empfangen von Nachrichten über das Funk-Modul.
Das abgebildete Use-Case-Diagramm soll die Funktionen und Abhängigkeiten darstellen.

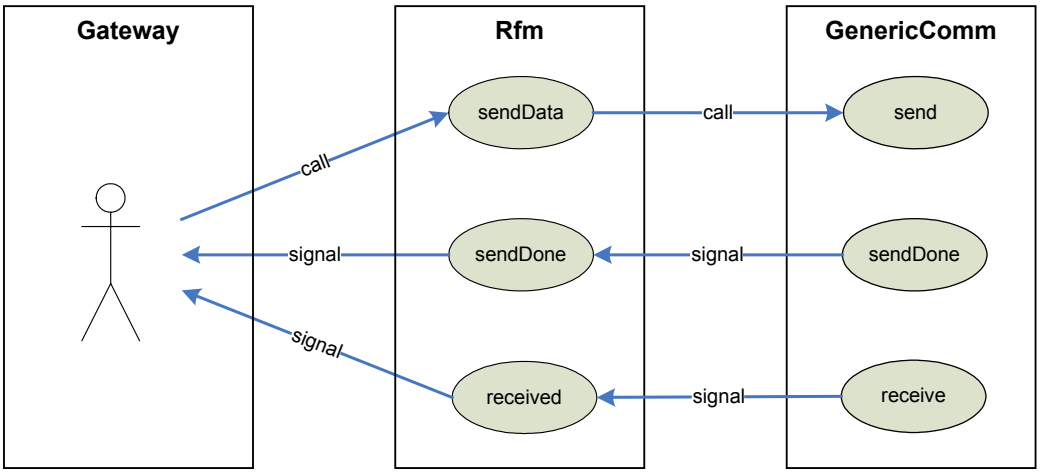


Abbildung 4.13 – Entwurf Gateway-Modul Rfm

- Die Funktion ‚sendData' sendet eine Nachricht durch den Aufruf der entsprechenden GenericComm-Funktion.
  Wurde diese Nachricht erfolgreich an das Betriebssystem übergeben, so wird von GenericComm ein Event ausgelöst. Das Rfm-Modul unterstützt zu jeder Zeit nur eine zu sendende Nachricht.
  Erst wenn diese Nachricht erfolgreich versendet und das entsprechende Event ausgelöst wurde, darf eine weitere Nachricht versendet werden.
- Aus diesem Grund wird der erfolgreiche Versand auch per Event an die nächst höhere Schicht, hier Gateway, signalisiert.
- Wird eine Nachricht empfangen, so wird dieses ebenfalls durch ein Event von GenericComm signalisiert. Diese Nachricht wird dann direkt per Event an Gateway weitergereicht.
- Das Modul arbeitet passiv, das heißt, es wartet auf externe Ereignisse, entweder ein Kommando zum Senden einer Nachricht, oder den Empfang einer Nachricht, bevor es selbst Ereignisse liefert.

Gateway

Dieses Modul bildet die Basis der Applikation. Es bedient sich der vorgestellten Module, und verbindet dessen Fähigkeiten miteinander.
Dadurch können Pakete von der Funk-Schnittstelle zur Uart-Schnittstelle, und umgekehrt geleitet werden.
Eine Datensicherung ist hier nicht notwendig, diese Komponente wird als Passiv implementiert und dient lediglich der Weiterleitung.

Allerdings muss eine Nachrichtenerweiterung implementiert werden, da mit jedem empfangenem Paket über das Funk-Interface dessen Feldstärke als Indiz für die Entfernung des Gateways benutzt werden kann. Diese muss dem PC also übermittelt werden.
Bewegt sich ein Läufer auf einer Kreisbahn und das Wearable sendet fortlaufend Funk-Nachrichten, kann permanent dessen Entfernung ermittelt werden.
Bei Annährung kann somit vom PC aus eine automatische Datenübertragung gestartet, und die bis dahin erfassten Daten übermittelt werden.

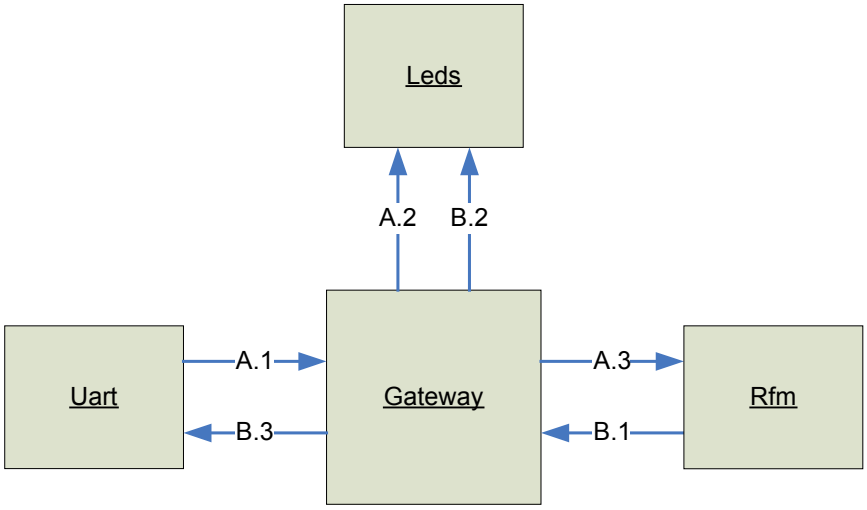


Abbildung 4.14 – Entwurf Gateway-Modul Gateway

- A.1 Paket vom PC auf der Uart-Schnittstelle empfangen
- A.2 Empfang signalisieren
- A.3 Paket an Wearable über die Funk-Schnittstelle senden
- B.1 Paket von Wearable auf der Funk-Schnittstelle empfangen
- B.2 Empfang signalisieren
- B.3 Paket an PC über die Uart-Schnittstelle senden

4.2.2.3. PC

An dieser Stelle soll die Software für den PC nur grob erläutert werden, da sie letztendlich nur zur Visualisierung dient.
Sie muss für eine sichere Datenübertragung sorgen und das Kommunikations-Protokoll implementieren (siehe Kapitel Kommunikation).
Gleichzeitig sollen die empfangenen Daten für Schritt- und Herz-Frequenz angezeigt werden und auch auf Festplatte gespeichert werden können.
Auf die Darstellung und Erläuterung der Betriebssystem-Komponenten soll hier verzichtet werden.

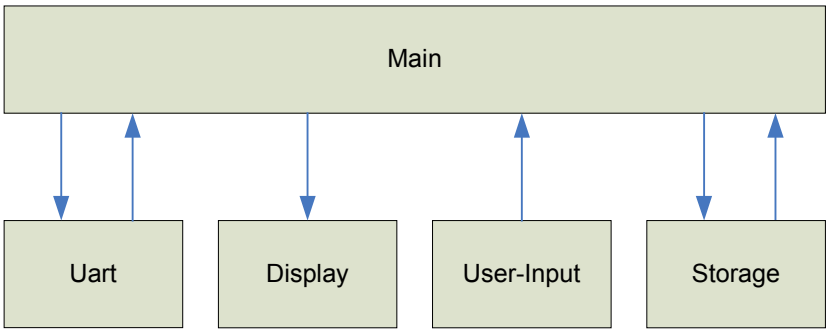


Abbildung 4.15 – Entwurf PC Software

- **Komponenten**

Nachfolgend sollen die einzelnen Module kurz beschrieben werden:

Uart
Dieses Modul dient zum Senden und Empfangen von Nachrichten über eine UART-Schnittstelle an ein TinyOS-kompatibles Gerät. Hierzu ist das TinyOS-Protokoll zu implementieren (siehe Kapitel Kommunikation).
Das Modul muss eine sichere Datenübertragung gewährleisten, dass heißt, eine Neuaufsetzung bei Paketverlust realisieren.

Display
Dieses Modul stellt dem Anwender eine grafische Oberfläche zu Verfügung, auf welcher empfangene, oder von der Festplatte geladene Analyse-Daten als Diagramm dargestellt werden.
Während einer Kommunikation mit einem Wearable ist die Funk-Feldstärke (grobe Entfernung) anzuzeigen.

User-Input
Der Benutzer hat über die grafische Oberfläche die Möglichkeit, dem Wearable Befehle zu schicken (siehe Kapitel Kommunikation), zwischen der Anzeige von verschiedenen Aufzeichnungs-Sätzen zu wählen und diese zu laden oder zu speichern.

Storage
Dieses Modul stellt die Funktionen zum Laden und Speichern von Aufzeichnungs-Sätzen von/auf Festplatte zur Verfügung.

Main

Dieses Modul bildet die Basis der Applikation und verbindet die vorgestellten Module miteinander. Somit kann die Applikation mit einem Wearable kommunizieren, dessen Daten darstellen und speichern.
Gleichzeitig muss es das RunAnalysis-Protokoll implementieren (siehe Kapitel Kommunikation) und bei einer grob gemessenen Entfernung des Wearables, eine Kommunikation und somit den Empfang von Analysedaten starten.

Bevor nun die Implementation starten kann, sollen zunächst die benötigten Daten-Formate beschrieben werden.
Das bezieht sich sowohl auf die zu speichernden (Analyse-Datenblock), als auch die zu übertragenden Daten (Kommunikation).

4.2.2.4. Analyse-Datenblock

Ein Datenblock besteht aus dem Zeitpunkt der Aufzeichnung und der entsprechenden Daten der Pulsfrequenz und Schrittfrequenz. In einem Datenblock werden jeweils 3 Messwerte, und somit 3 Sekunden der Aufzeichnung zusammengefasst.
Ein Datenblock wird jeweils als eine Zeile im Daten-Flash (1 Zeile ≙ 16 Byte) gespeichert. Das erleichtert das Lesen und Schreiben und bietet trotzdem genügend Zeitraum für die Aufzeichnung.

Ein Datenblock sieht dann folgendermaßen aus:

	Analysis-Data					
Time	BPM(0)	SPM(0)	BPM(1)	SPM(1)	BPM(2)	SPM(2)
2	2	2	2	2	2	2

Abbildung 4.16 – Analyse Datenblock

Folgende Informationen sind dabei enthalten:

Feld	**Bedeutung**
Time	Zeitpunkt des Datenblocks in dieser Aufzeichnung. Jede Aufzeichnung beginnt bei Zeitpunkt 0 (Sekunden). Da ein Datenblock aus einem Zeitraum von 3 Sekunden besteht, ist ∆Time zweier aufeinander folgender Datenblöcke == 3.
BPM(x)	Pulsfrequenz zum Zeitpunkt (Time + x) der Aufzeichnung
SPM(x)	Schrittfrequenz zum Zeitpunkt (Time + x) der Aufzeichnung

Wie zu sehen, wird die Nummer der aktuellen Aufzeichnung nicht mit gespeichert, diese ergibt sich aus dem Auslesen beim Start des Systems.
Jede Time = 0 steht für eine neue Aufzeichnung.

Demzufolge sieht der Flash bei mehreren Aufzeichnungen wie folgt aus:

Time = 0	Analysis-Data
Time = 3	Analysis-Data
⋮	⋮
Time = 0	Analysis-Data
⋮	⋮

Abbildung 4.17 – Analyse Datenblöcke

Durch eine Flash-Größe von 512 KByte, von denen 524032 Byte für eigene Daten zur Verfügung stehen, ergibt sich eine Anzahl von 32752 Zeilen á 16 Byte.
Es können also maximal 32752 x 3 = 98256 Sekunden (ca. 27,3 Stunden) aufgezeichnet werden.

4.2.2.5. Kommunikation

Die Kommunikation zwischen den Endgeräten Wearable und PC wird als direkte Verbindung angesehen, sodass auf die automatischen Bestätigungs-Mechanismen des Betriebssystems verzichtet wird.
Hier wäre es zwar möglich, eine Bestätigung zwischen PC und Gateway, und zwischen Wearable und Gateway zu realisieren, es soll aber eine Art Token-Übergabe zwischen den beiden Endgeräten implementiert werden.
Der Vorteil ist die hohe Datensicherung und die einfache Implementierung des Gateway, da hier keine Paketpuffer für einen eventuellen einseitigen Engpass nötig sind.
Der natürlich große Nachteil der zeitlichen Verzögerung wird dabei in Kauf genommen, hier geht es nicht um Performanz.

- **TinyOS**

Im Folgenden sollen die von TinyOS standardisierten Frame-Formate erläutert werden, um danach die eigenen Formate auf diese aufzusetzen.

RFM

Das TinyOS-Frame-Format für die Funk-Kommunikation wurde bereits im Kapitel TinyOS vorgestellt, hier soll es nur zur Erinnerung noch einmal kurz dargestellt werden.

TOS-Header				User-Data	
Address	Active Message Type	Group ID	Payload Length	Payload	CRC
2	1	1	1	0..29	2

Abbildung 4.18 – TinyOS RFM-Frame

UART

Für die Kommunikation über das Uart-Interface wurde dieser Daten-Frame erweitert.
Um ein Synchronisation zwischen den beiden Endgeräten sicherzustellen, wurde ein Sync-Byte eingefügt, das den Start eines neuen Frames angibt.
Da das Sync-Byte natürlich auch in den darauf folgenden Daten vorkommen kann, muss es hier mit einem speziellen Escape-Zeichen versehen werden. Gleichzeitig wird der Wert, des mit einem Escape versehenen Zeichens, reversibel modifiziert.
Folgende Tabelle soll dies erklären.

Zeichen	**Wert**
Sync	0x7E
Escape	0x7D
Escapetes Zeichen	Neuer Wert = Wert XOR 0x20

Die 16-Bit CRC beinhaltet dabei alle Zeichen, mit Ausnahme des Sync-Bytes. Die eventuellen Escape-Zeichen werden erst nach der CRC-Berechnung eingefügt.

Weiterhin wurde ein Paket-Typ eingefügt, welcher folgende Bedeutung hat:

Paket-Typ	**Bedeutung**	**Wert**
Ack	Das gesendete Paket ist eine Bestätigung	0x40
Packet_Ack	Das gesendete Paket verlangt eine Bestätigung	0x41
Packet_No_Ack	Das gesendete Paket benötigt keine Bestätigung	0x42
Unknown	Der Typ ist unbekannt	0xFF

Daraus resultieren diese Datenframes:

Packet_No_Ack

Wird ein Paket versendet, das nicht bestätigt werden soll und somit bei Übertragungsfehlern verloren ist:

Packet-Header		TOS-Header				User-Data	
Sync	Packet Type	Address	Active Message Type	Group ID	Payload Length	Payload	CRC
1	1	2	1	1	1	0..29	2

Abbildung 4.19 – TinyOS Packet_No_Ack-Frame

Packet_Ack

Muss der einwandfreie Empfang sichergestellt werden, so wird folgender Frame benutzt.

Packet-Header			TOS-Header				User-Data	
Sync	Packet Type	Seq	Address	Active Message Type	Group ID	Payload Length	Payload	CRC
1	1	1	2	1	1	1	0..29	2

Abbildung 4.20 – TinyOS Packet_Ack-Frame

Das empfangene Gerät muss daraufhin ein Ack mit der gleichen Sequenznummer (Seq) senden.
Wird kein Ack gesendet, so wird nach einem Timeout das Paket wiederholt.
Die Sequenznummer wird einfach nach jedem Paket erhöht, oder nach Belieben des Anwenders modifiziert.

Ack

Wurde ein Paket vom Typ Packet_Ack einwandfrei empfangen, so muss ein Ack als Bestätigung gesendet werden. Die Sequenznummer (Seq) muss dabei identisch mit der des zu bestätigenden Paketes sein.

Packet-Header			
Sync	Packet Type	Seq	CRC
1	1	1	2

Abbildung 4.21 – TinyOS Ack-Frame

- **RunAnalysis**

Für die Kommunikation sind verschiedene Nachrichten-Typen und somit auch verschiedene Pakete notwendig.
Folgende Tabelle beschreibt die nötigen Nachrichten und deren Richtung:

Nachrichten-Typ	**Bedeutung**	**Richtung**
Ack	Die Nachricht ist eine Bestätigung auf eine von der Gegenstelle gesendeten Nachricht.	Wearable -> PC PC -> Wearable
Ping	Bei laufender Aufzeichnung pingt der Wearable den PC an, bis dieser ein Kommando sendet. Dient zur Aufnahme einer automatischen Kommunikation.	Wearable -> PC
Log_Count	Nach dem Kommando Cmd_Send_Log_Count sendet der Wearable die Anzahl der im Flash gespeicherten Aufzeichnungen.	Wearable -> PC
Cmd_Send_Log	Aufforderung, die gespeicherten Informationen einer Aufzeichnung zu senden. Nach diesem Befehl wird nach dem Ack eine Nachricht vom Typ Log_Data gesendet.	PC -> Wearable
Cmd_Clear_Log	Aufforderung, alle vorhandenen Aufzeichnungen zu löschen. Das Flash wird unwiderruflich gelöscht!	PC -> Wearable
Cmd_Send_Log_Count	Aufforderung, die Anzahl der gespeicherten Aufzeichnungen zu senden. Auf diese Anforderung folgt statt einem Ack, sofort eine Nachricht vom Typ Log_Count.	PC -> Wearable
Log_Data	Es wird ein Datenblock einer Aufzeichnung gesendet.	Wearable -> PC

RFM

Aus den Anforderungen entsteht folgender Basis-Frame, welcher anschließend in den TinyOS-Frame eingebettet (folgt später) und über das Funk-Interface versendet wird:

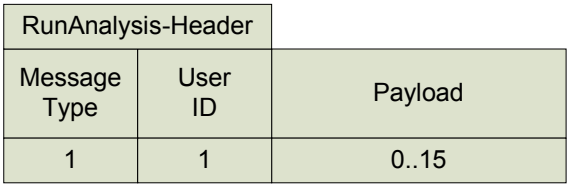


Abbildung 4.22 – RunAnalysis RFM-Frame

Feld	**Bedeutung**
Message Type	Der Nachrichtentyp ist entsprechend der oberen Tabelle zu wählen
User ID	Das Feld gibt an, welcher Benutzer oder welches Wearable gerade Daten sendet. Im Rahmen dieser Arbeit allerdings unwichtig.
Payload	Der Payload ist abhängig vom verwendeten Nachrichten-Typ und wird für diese getrennt beschrieben

Dieser Basis-Frame wird nun für den jeweiligen Nachrichtentyp wie folgt benutzt.

Ack, Ping, Command_Clear_Log, Command_Send_Log_Count

Für diese Nachrichten-Typen muss lediglich der Header übertragen werden. Hier ist keine weitere Erklärung notwendig, da nur diese beiden, bereits beschriebenen, Felder gefüllt werden müssen.

RunAnalysis-Header	
Message Type	User ID
1	1

Abbildung 4.23 – RunAnalysis Header-Frame

Log_Count

Bei diesem Nachrichtentyp wird als Payload die Anzahl der im Flash gespeicherten Aufzeichnungen, inklusive der eventuell momentan laufenden, eingetragen.

RunAnalysis-Header		Payload
Message Type	User ID	Log Count
1	1	1

Abbildung 4.24 – RunAnalysis Log_Count-Frame

Cmd_Send_Log

Bei diesem Nachrichtentyp wird als Payload die Nummer, der vom Wearable zu sendenden Aufzeichnung, eingetragen.
Hierbei ist folgende Tabelle zu beachten:

Wert	**Bedeutung**
0x00	Es soll die momentan laufende Aufzeichnung, bzw. falls gerade keine läuft, die zuletzt abgeschlossene Aufzeichnung übertragen werden. Der Speicher wird dabei sequenziell ausgelesen.
0x01 ... 0xFE	Es soll die angegebene Aufzeichnung übertragen werden, wobei die Nummer 0x01 die älteste Aufzeichnung meint. Der Speicher wird dabei sequenziell ausgelesen.
0xFF	Bei diesem Wert wird der Lese-Zeiger resettet, das heißt, dass beim nächsten Befehl mit einem Wert < 0xFF die entsprechende Aufzeichnung vom Anfang (Zeit = 0, siehe Log_Data) gelesen wird.

RunAnalysis-Header		Payload
Message Type	User ID	Log Number
1	1	1

Abbildung 4.25 – RunAnalysis Cmd_Send_Log-Frame

Log_Data

Bei diesem Nachrichtentyp wird, nach Aufforderung, ein Datenblock mit Analyse-Daten vom Wearable an den PC gesendet.
Hier sind folgende Informationen enthalten:

Feld	**Bedeutung**
Log Number	Nummer der Aufzeichnung, die gesendet wird, entspricht der Log Number des Kommandos Cmd_Send_Log
Time	Zeitpunkt des Datenblocks
BPM(x)	Pulsfrequenz zum Zeitpunkt (Time + x)
SPM(x)	Schrittfrequenz zum Zeitpunkt (Time + x)

RunAnalysis-Header		Payload							
Message Type	User ID	Log Number	Analysis-Data						
			Time	BPM(0)	SPM(0)	BPM(1)	SPM(1)	BPM(2)	SPM(2)
1	1	1	2	2	2	2	2	2	2

Abbildung 4.26 – RunAnalysis Log_Data-Frame

RunAnalysis-RFM ⇒ TinyOS-RFM

Diese vorgestellten Daten-Frames müssen nun in den TinyOS-Frame eingefügt werden, um sie per Funk übertragen zu können.
Dabei wird der RunAnalysis-Frame als Payload in den TinyOS-Frame eingesetzt.

Eine über Funk versendete Nachricht sieht dann folgendermaßen aus:

TOS-Header				TOS-Payload			
Address	Active Message Type	Group ID	Payload Length	RunAnalysis-Header		Payload	CRC
				Message Type	User ID		
				1	1	0..15	
2	1	1	1	2..17			2

Abbildung 4.27 – RunAnalysis RFM -> TinyOS RFM

UART

Der Uart-Frame wird vom Rfm-Frame abgeleitet und um die Entfernungsmessung erweitert.
Da TinyOS für jede über Funk empfangene Nachricht dessen Feldstärke bereitstellt, kann diese zur groben Entfernungsmessung benutzt werden, und muss vom empfangenden Gateway zum PC übermittelt werden.

Der resultierende Uart-Frame sieht folgendermaßen aus:

RunAnalysis-Header			Uart-Ext.
Message Type	User ID	Payload	RFM Strength
1	1	0..15	2

Abbildung 4.28 – RunAnalysis UART-Frame

Bis auf die Erweiterung der Funk-Feldstärke, ist dieser Frame mit allen Rfm-Nachrichtentypen identisch.

RunAnalysis-UART ⇒ TinyOS-UART

Diese vorgestellten Daten-Frames müssen nun in den TinyOS-Frame eingefügt werden, um sie über den UART versenden zu können.
Dabei wird der RunAnalysis-Frame als Payload in den TinyOS-Frame eingesetzt.
Als Paket-Typ wird ausschließlich Packet_No_Ack, also eine Paketübertragung ohne automatisches Ack benutzt, da das Gateway kein Endgerät ist und somit ein Ack zwischen PC und Gateway keine Bedeutung für den sicheren Datentransfer zwischen PC und Wearable hat.

Eine über UART versendete Nachricht sieht dann folgendermaßen aus:

Packet-Header		TOS-Header				TOS-Payload				
						RunAnalysis-Header			Uart-Ext.	
Sync	Packet Type	Address	Active Message Type	Group ID	Payload Length	Message Type	User ID	Payload	RFM Strength	CRC
						1	1	0..15	2	
1	1	2	1	1	1	4..19				2

Abbildung 4.29 – RunAnalysis UART -> TinyOS UART

4.3. Implementation

Da jetzt die Funktionalität des gesamten Systems, und auch der einzelnen Module, definiert ist, soll jetzt die Funktionsweise der einzelnen Module und dessen Zusammenspiel erklärt werden.
Dieses wird auch für die drei Bestandteile des Systems (Wearable, Gateway, PC) getrennt erläutert.

4.3.1. Hardware

Auch die Implementation soll mit der Hardware begonnen werden.

4.3.1.1. Wearable

Das Wearable besteht, wie bereits beschrieben, aus einem Mica2Dot-Board, einem MTS510-Sensor-Board und einem MDA500-Prototyp-Board.
Die Beschreibung dieser Komponenten erfolgte bereits im Hauptkapitel Hardware.
An dieser Stelle soll nun die Zusammenschaltung dieser Komponenten in Verbindung mit der Pulsmessuhr, der Status-LED und einem Start/Stopp-Taster erfolgen.

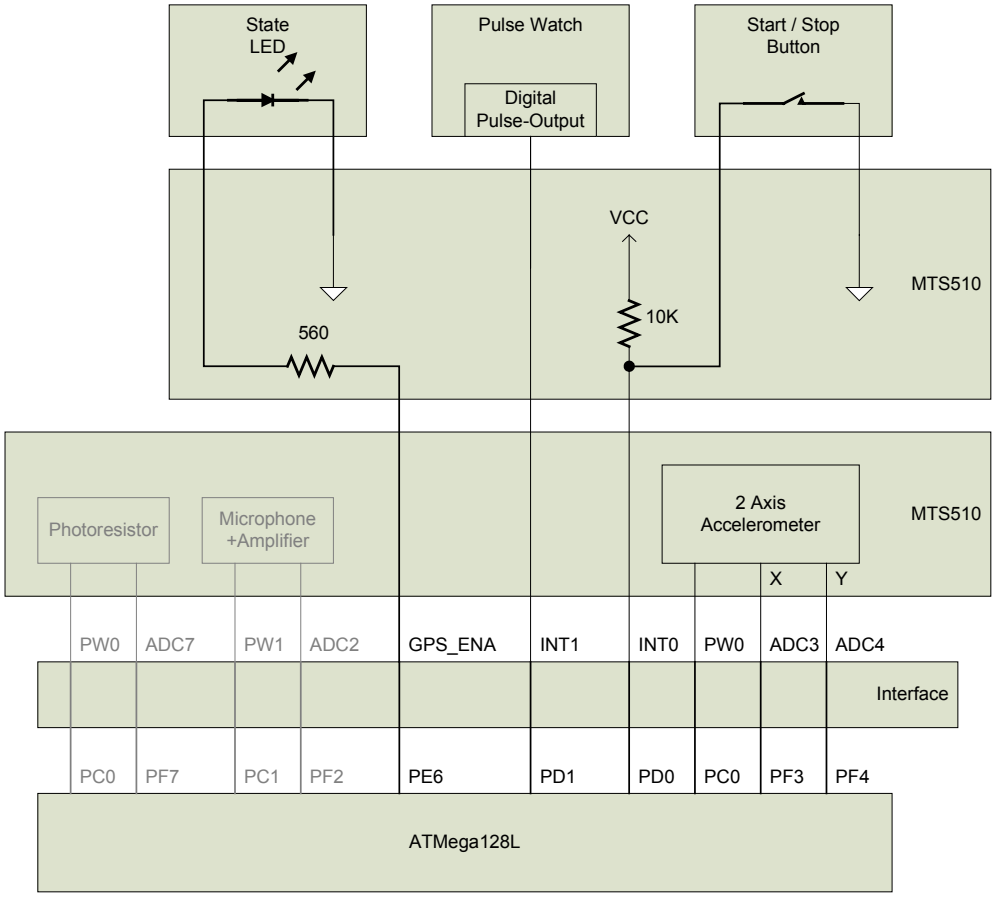


Abbildung 4.30 – Implementation Wearable Hardware

Das Prototyp-Board enthält für alle verfügbaren Signale Lötpunkte, sodass die externen Komponenten hier an unbenutzte Signalleitungen angeschlossen werden können. Der Photo-Sensor und das Mikrofon sind zwar vorhanden, werden aber nicht benutzt.

4.3.1.2. Gateway, PC

Für diese beiden Komponenten sind keine speziellen Implementierungen nötig. Hier werden Standard-Komponenten, bzw. fertige Geräte benutzt.
Für weitere Details siehe Kapitel Entwurf.

4.3.2. Software

Aufsetzend auf der definierten Hardware kann nun die Implementation der Software erfolgen.

4.3.2.1. Wearable

Die im Entwurf vorgestellten Module sollen jetzt durch ihre jeweilige Funktion komplettiert werden.

PulseMeter

Wie im Kapitel Entwurf beschrieben, basiert das Modul auf einem Timer, der im Millisekundentakt ein Event liefert. Die durch ihn ausgeführte Event-Service-Routine ist der Hauptbestandteil des Moduls und soll durch folgendes Zustands-Diagramm beschrieben werden:

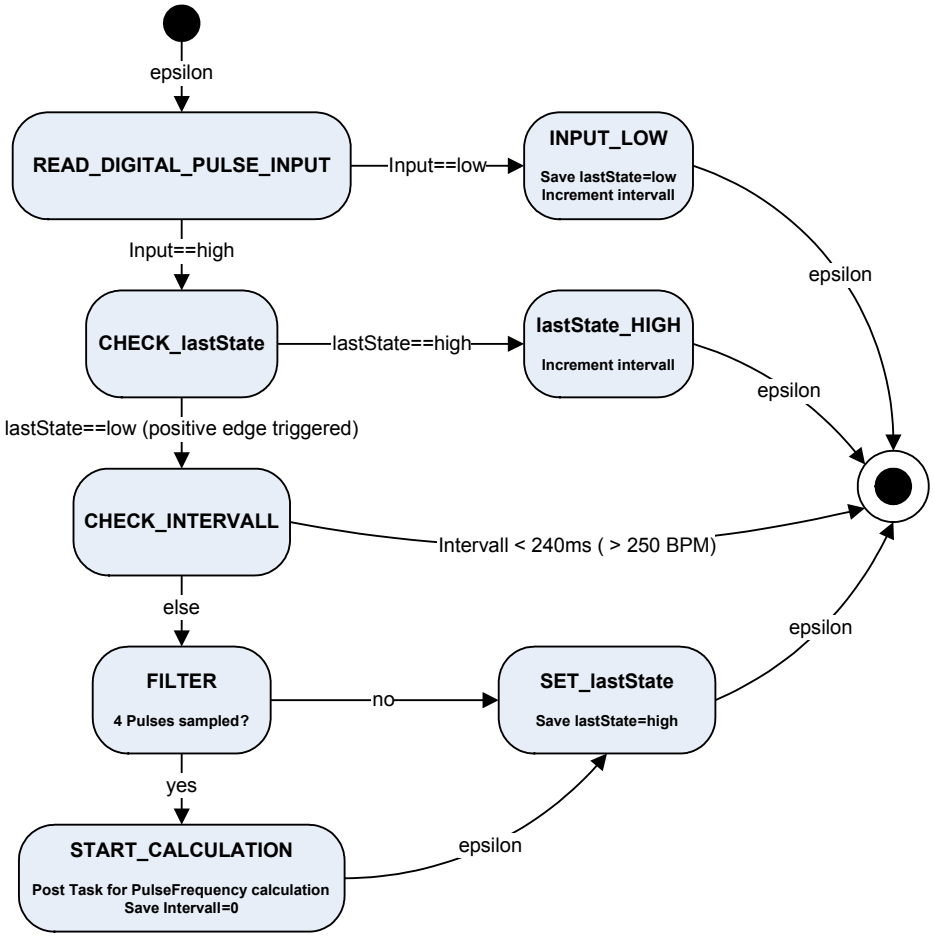


Abbildung 4.31 – Implementation Wearable-Modul PulseMeter

Der Zustand ‚FILTER' sorgt dafür, dass 4 Herzschläge zusammengefasst werden. Der Grund hierfür ist eine möglichst hohe Präzision, da durch vorangegangene Test doch sehr große Abweichungen bei eigentlich konstanter Herzfrequenz gemessen wurden.
Der Zustand ‚START_CALCULATION' startet einen Task, welcher nach folgender Formel die Herzfrequenz / Minute berechnet.
Dabei zu beachten ist, dass hier von einem gemessenem Intervall von 4 Herzschlägen ausgegangen wird. Gleichzeitig wurde ein Low-Pass-Filter implementiert, welcher wiederum zur Präzision beiträgt, indem jeder neue Wert nur zu 25% in die Neuberechnung des Pulsfrequenz eingeht.

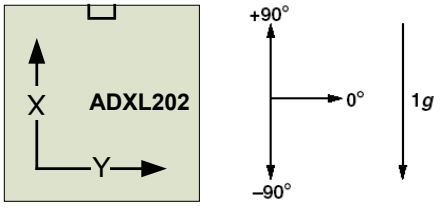
$$Pulsfrequenz = \frac{3}{4} Pulsfrequenz + \frac{1}{4}\left(\frac{60000}{\frac{Intervall}{4}}\right)$$

Nach dieser Berechnung wird das Ergebnis per Event an das Haupt-Modul RunAnalysis weitergereicht.

StepMeter

Dieses Modul basiert, wie beschrieben, auf einem 2-Achsen Beschleunigungs-Sensor. Dieser Sensor soll benutzt werden, um mit den, beim Laufen auftretenden Schwingungen, eine Schritt-Detektion zu ermöglichen.
Da die beiden Achsen 90° zueinander versetzt auf einer Ebene sitzen, ergibt sich die abgebildete Resonanz auf einwirkende Beschleunigungen.

Ausgangspunkt ist der abgebildete Sensor mit seinen zwei Achsen und nebenstehender Beeinflussung durch die Erd-Gravitation.



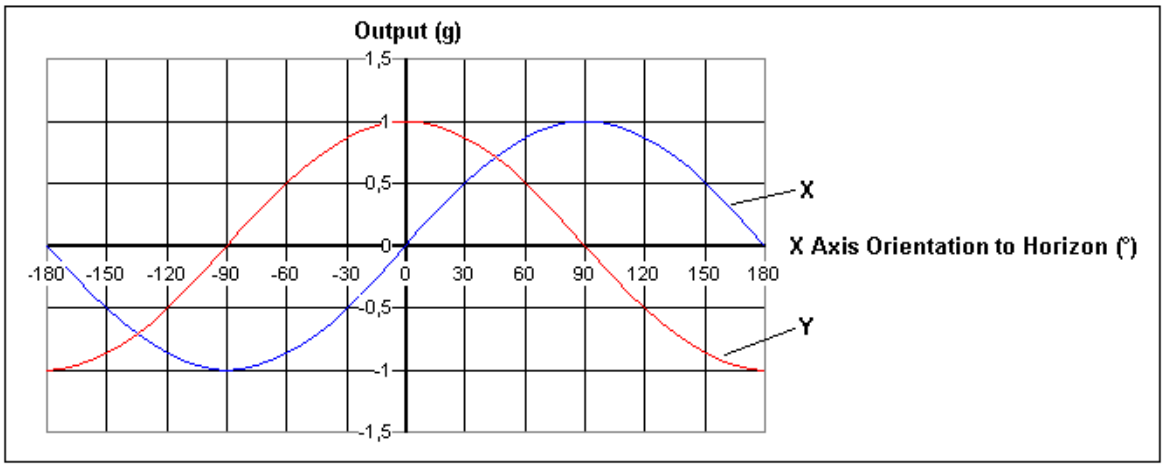


Abbildung 4.32 – Resonanz Beschleunigungssensor

Dadurch wird ersichtlich, dass bei bestimmten Verdrehungen sowohl die Erd-Gravitation, also auch andere Beschleunigungen, keinen Einfluss auf eine Sensor-Achse haben kann (Ausgang zeigt 0g bei Einwirkung von 1g).
Da die Lage des Sensors am Wearable nicht fest definiert werden kann, müssen somit beide Sensor-Werte miteinander kombiniert werden. Dann ist sichergestellt, dass jede Beschleunigung (außer die Beschleunigungsachse steht senkrecht auf der Sensor-Ebene) einen messbaren Ausgangswert erzeugt.

Hauptbestandteil dieses Moduls ist auch ein Timer, welcher alle 10ms ein Event erzeugt.
Weil nicht beide Achsen des Beschleunigungs-Sensors gleichzeitig ausgelesen werden können, wird beim Timer-Event die X-Achse und bei dessen Fertigstellung, die Y-Achse gelesen.
Wenn der Wert für die Y-Achse vorliegt, werden folgende Berechnungen angestellt:

$$X\_Amplitude = \left(Absolut\left(X\_Durchschnitt - X\_Wert\right)\right)^2$$
$$Y\_Amplitude = \left(Absolut\left(Y\_Durchschnitt - Y\_Wert\right)\right)^2$$
$$XY\_Amplitude = X\_Amplitude + Y\_Amplitude$$

Der Durchschnitt für beide Achsen wird einmal pro Sekunde, also über die letzten 100 Werte (1 Wert / 10ms), gebildet.
Dieses ist notwendig, um eine Verdrehung der Achsen, und der daraus resultierende ungewollte Einfluss der Erd-Gravitation, zu kompensieren.
Das Quadrieren der Werte wurde gewählt, um einen möglichst großen Rauschabstand zu erhalten.
Die daraus resultierenden Daten sehen wie folgt aus:

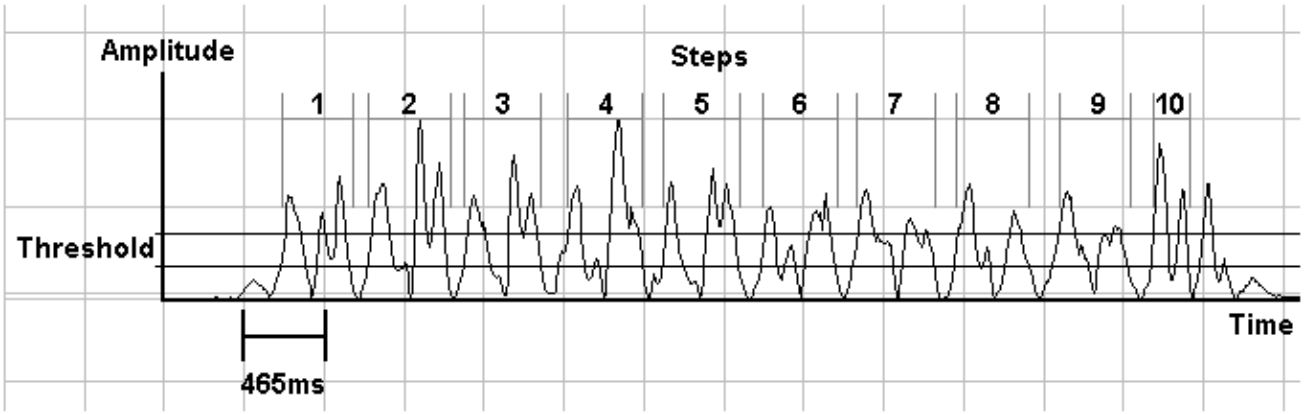


Abbildung 4.33 – Messung Beschleunigungssensor

Bei einem Schritt entstehen somit zwei Schwingungen (siehe Bild), die relativ unabhängig von der Lage des Sensors sind. Jetzt wird anhand von zwei Schwellen ein Schritt detektiert und wie folgt verfahren:

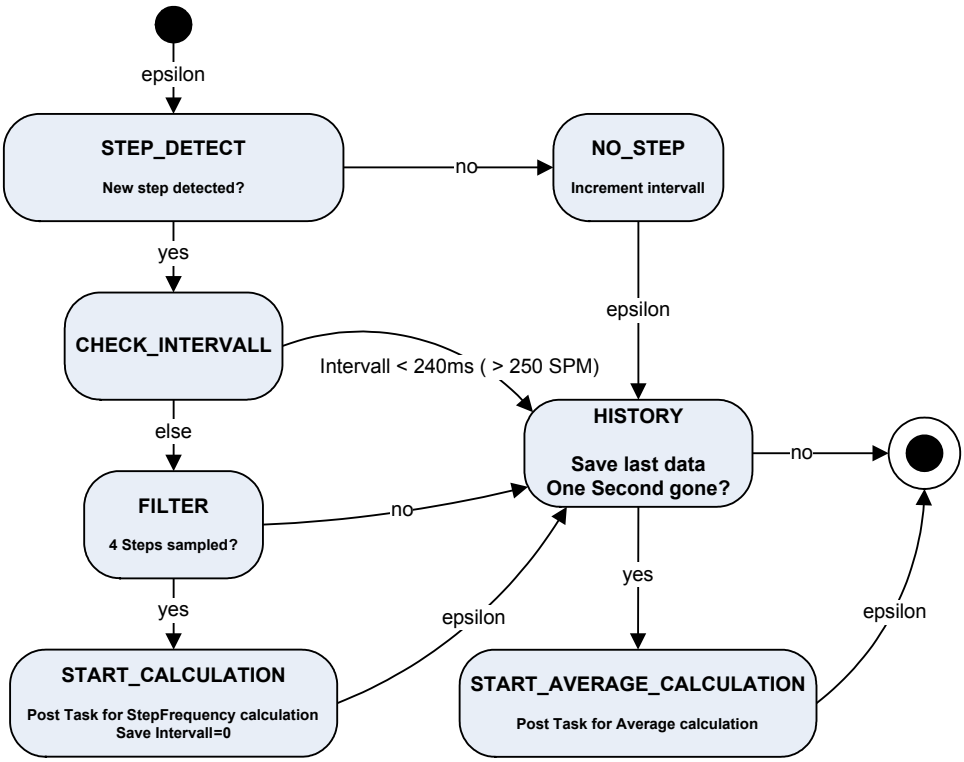


Abbildung 4.34 – Implementation Wearable-Modul StepMeter

Der Zustand ‚FILTER' sorgt dafür, dass 4 Schritte zusammengefasst werden. Der Grund hierfür ist eine möglichst hohe Präzision, da durch vorangegangene Test doch sehr große Abweichungen messen wurden.
Der Zustand ‚START_CALCULATION' startet einen Task, welcher nach folgender Formel die Schrittfrequenz / Minute berechnet.
Dabei zu beachten ist, dass hier von einem gemessenem Intervall von 4 Schritten ausgegangen wird. Gleichzeitig wurde ein Low-Pass-Filter implementiert, welcher wiederum zur Präzision beiträgt, indem jeder neue Wert nur zu 50% in die Neuberechnung des Schrittfrequenz eingeht.

$$Schrittfrequenz = \frac{1}{2} Schrittfequenz + \frac{1}{2}\left(\frac{6000 \times 4}{Intervall}\right)$$

Nach dieser Berechnung wird das Ergebnis per Event an das Haupt-Modul RunAnalysis weitergereicht.

Der Zustand ‚START_AVERAGE_CALCULATION' startet einen Task, welcher den Mittelwert der letzten 100 Sensor-Werte für beide Achsen berechnet.

$$(X/Y)\_Average = \frac{\sum_{i=0}^{99} lastData(X/Y)(i)}{100}$$

Die Schritt-Detektion ist wahrscheinlich nicht die Beste, aber sie funktioniert relativ zuverlässig.
Einziger Schwachpunkt ist der Erste und Letzte Schritt, da hier die Schwingungen nicht identisch mit denen aufeinander folgender Schritte sind. Diese eventuelle Ungenauigkeit ist aber vernachlässigbar.

RunAnalysis

Das Modul erhält die Daten der beiden Sensoren (Pulsfrequenz, Schrittfrequenz) über je ein Event. Diese Daten werden zuerst im Ram und alle drei Sekunden ins Flash gespeichert.

Es enthält zwei aktive Teile, die hier beschrieben werden sollen.

1. Einen Sekunden-Timer, der alle internen Aufgaben steuert

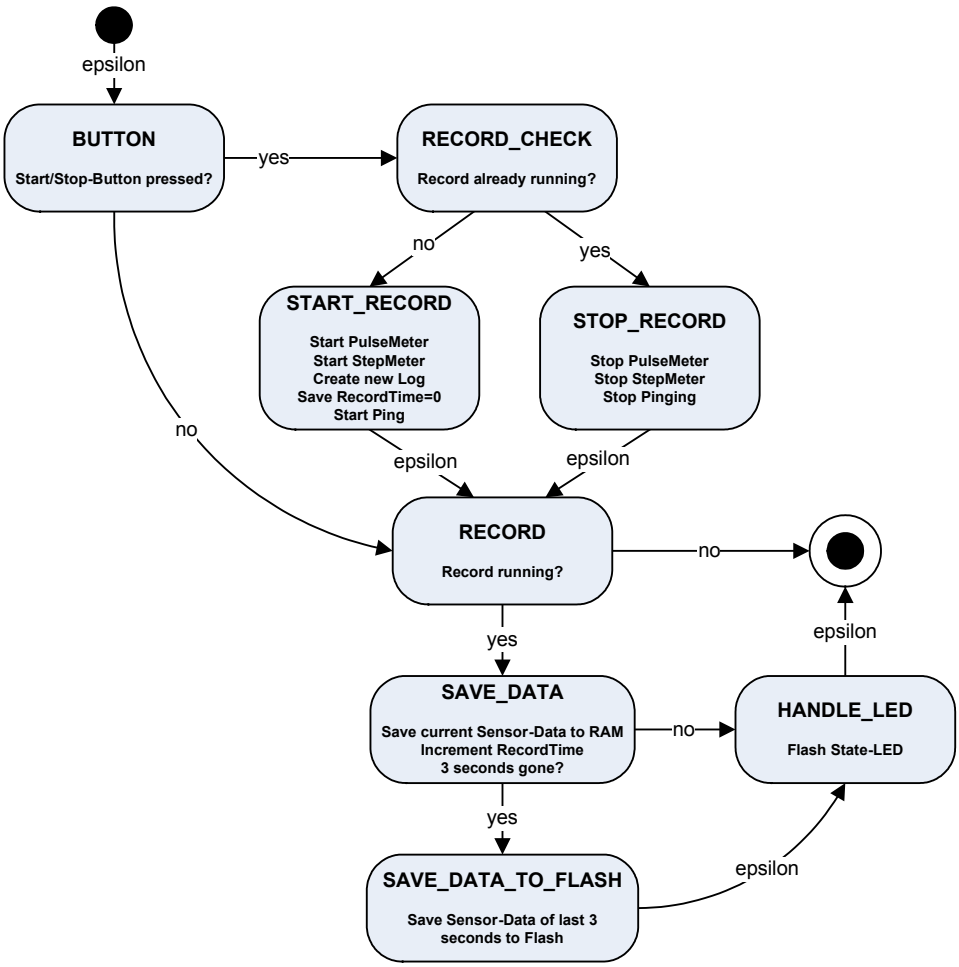


Abbildung 4.35 – Implementation Wearable-Modul RunAnalysis 1/2

Beim Start einer neuen Aufzeichnung (Zustand ‚START_RECORD') werden die Sensor-Module gestartet und ein Ping initiiert. Das heißt, nach Ablauf von 10 Sekunden beginnt das Gerät im 2Hz-Takt ein Ping auszusenden.
Dadurch kann die PC-Software die Feldstärke und somit die grobe Entfernung messen. Wenn der PC nun mit einem Befehl antwortet, wird dieses durch den 2. aktiven Teil bearbeitet.

2. Eine Befehlsverarbeitung für das Funk-Interface

Wird ein Paket empfangen, wird wie folgt verfahren:

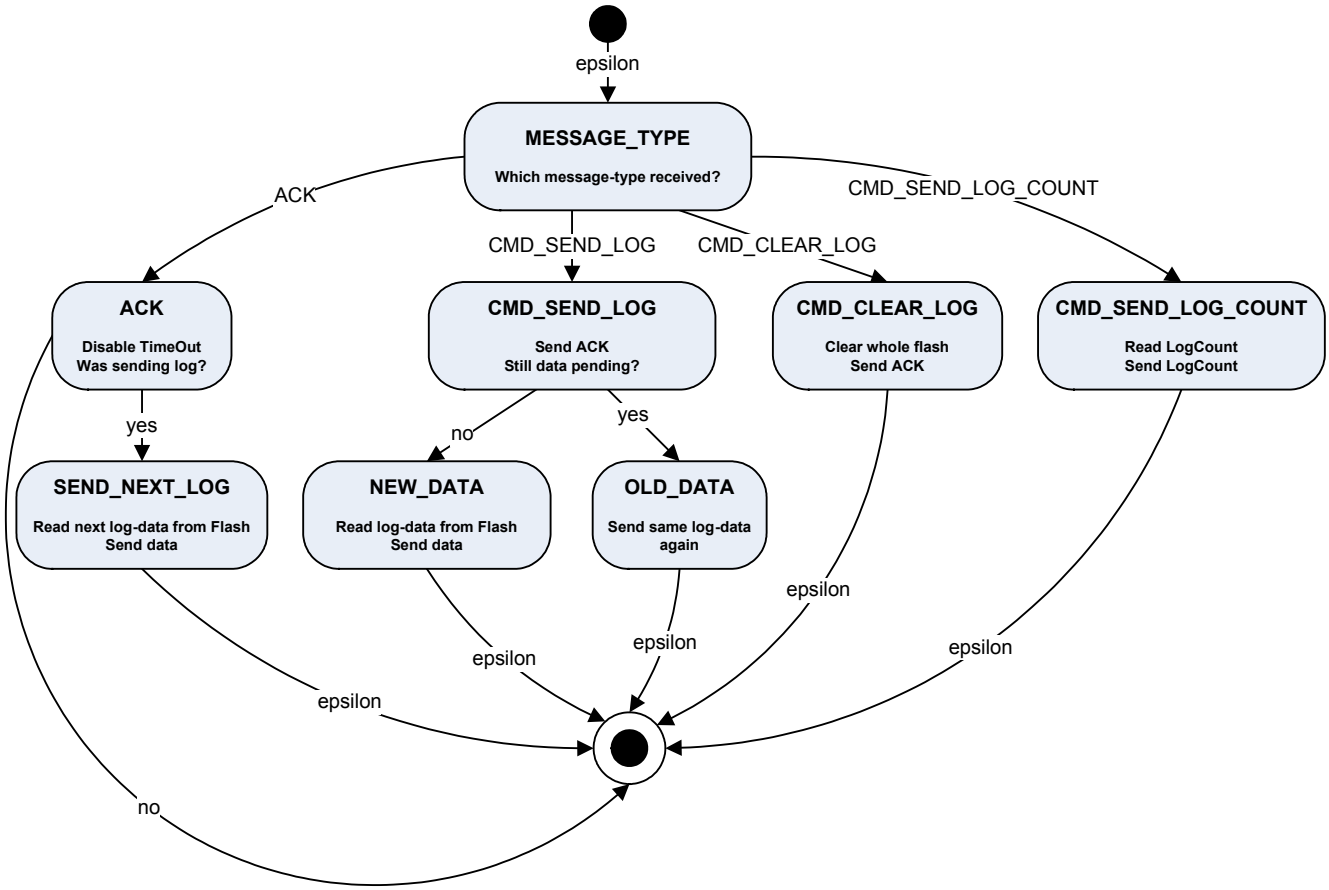


Abbildung 4.36 – Implementation Wearable-Modul RunAnalysis 2/2

Mit einem empfangenem ACK wird ein TimeOut-Timer gestoppt. Läuft dieser aber über (nach 300ms), so wird das zuletzt gesendete Paket wiederholt. Wurde ein Paket 6 Mal gesendet und kein ACK empfangen, so gilt es als verloren.
War dieses ACK eine Bestätigung auf ein Log-Daten-Paket, so wird ein weiteres aus dem Flash gelesen und versendet. Somit funktioniert der Versand automatisch, bis die Kommunikation aus Gründen der Feldstärke beendet wird, oder keine weiteren Daten vorliegen.
Wird ein CMD_SEND_LOG empfangen und ein zuvor gesendetes Log-Daten-Paket gilt als verloren, so wird dieses jetzt wiederholt (Zustand OLD_DATA).
Wurde mit diesem Befehl ein RESET_LOG geschickt, so wird das Modul RALogger auf das Lesen eines neuen Datensatzes vorbereitet, ansonsten wird sequentiell weiter gelesen (Zustand NEW_DATA).

Weitere Komponenten

Die anderen Module sind passiv, und sollen hier nicht näher erläutert werden.

4.3.2.2. Gateway

Dieses Gerät besitzt nur ein aktives Modul, sodass dessen Erläuterung hier ausreichend ist.

Gateway

Dieses Modul agiert nur bei Empfang einer Nachricht auf der Funk- oder UART-Schnittstelle.
Somit laufen hier zwei Automaten parallel.

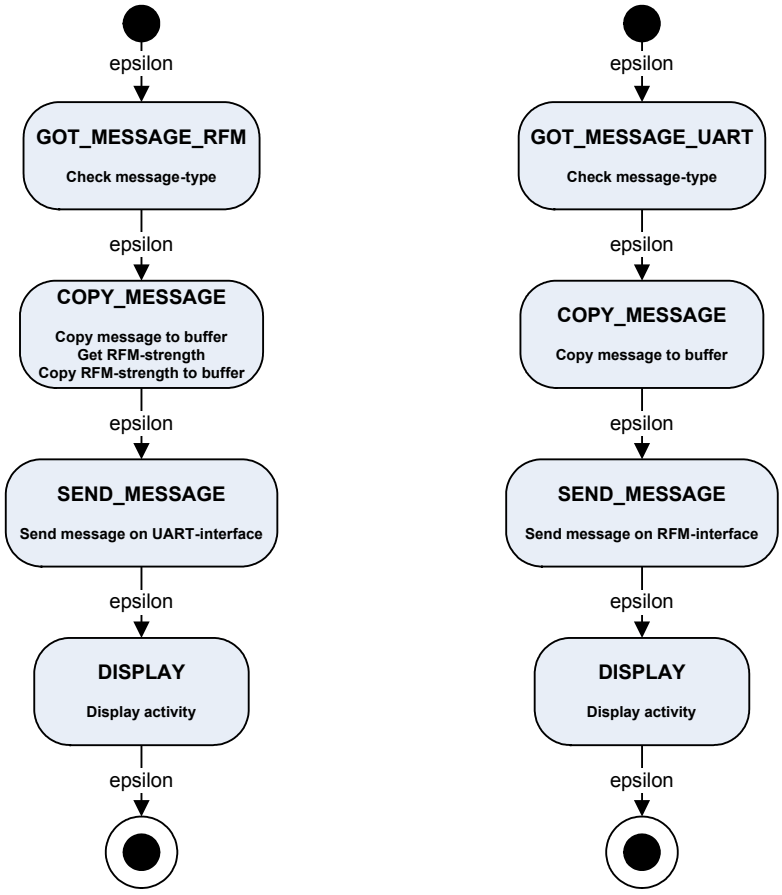


Abbildung 4.37 – Implementation Gateway-Modul Gateway

Wird auf der Funk-Schnittstelle eine Nachricht empfangen, so wird sie kopiert, die Feld-Stärke ausgelesen, dieser der zu sendenden Nachricht angehängt und schließlich auf der UART-Schnittstelle versendet.
Wird auf der UART-Schnittstelle eine Nachricht empfangen, so wird sie kopiert und versendet.
Um eine Aktivität bei der Kommunikation ersichtlich zu machen, werden die Status-LEDs des Mica2-Boards, abhängig von der Datenrichtung, getoggelt.

Weitere Komponenten

Die anderen Module sind passiv, und sollen hier nicht näher erläutert werden.

4.3.2.3. PC

Diese Software wird mit dem Entwicklungstool Delphi 7.0 entwickelt.
Die PC-Software besteht aus zwei aktiven Teilen, zum einen dem UART-Modul und zum anderen dem User-Input-Modul.
Für die Beschreibung der Software soll die Erläuterung des Moduls UART genügen, da dieses die Hauptarbeit leistet und noch zur MOTE-Technologie gehört.
Die anderen Module bestehen im Prinzip nur aus Systemfunktions-Aufrufen und sind mit einer Visuellen Entwicklungsumgebung wie Delphi schnell einsatzbereit.

UART

Dieses Modul empfängt Nachrichten über die UART-Schnittstelle des PCs und verarbeitet diese bei korrekter Checksumme entsprechend ihrer Funktion.
Es muss also eine Implementation des TinyOS-Protokolls für die UART-Schnittstelle erfolgen.

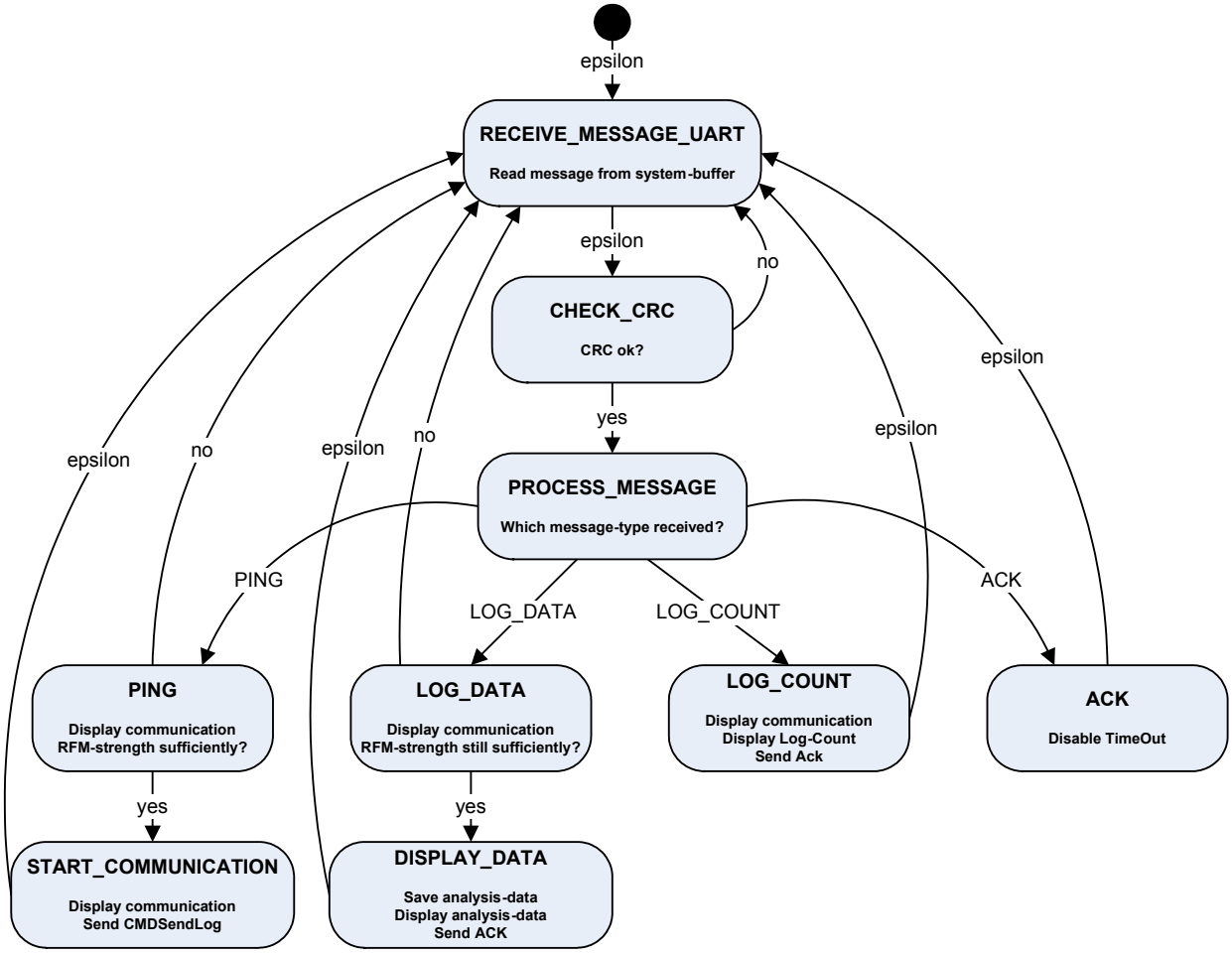


Abbildung 4.38 – Implementation PC-Modul UART 1/2

Wird eine PING-Nachricht empfangen und die damit übermittelte Feldstärke ist groß genug, dann wird mit einem Kommando, zum Senden von Analyse-Daten, geantwortet.
Wird eine LOG-DATA-Nachricht empfangen, wird überprüft, ob die Feldstärke noch groß genug ist. Ist dies der Fall, wird ein Ack gesendet und die Daten auf dem Display ausgegeben.

Bei Empfang eines ACKs wird der TimeOut-Timer gestoppt. Läuft dieser aber über, so wird die zuletzt versendete, aber nicht bestätigte, Nachricht erneut versendet.
Dieses Modul wird als Thread implementiert und wird erst durch das Beenden des Programms terminiert.
Gleichzeitig besitzt das Modul Funktionen zum Senden einer Nachricht.

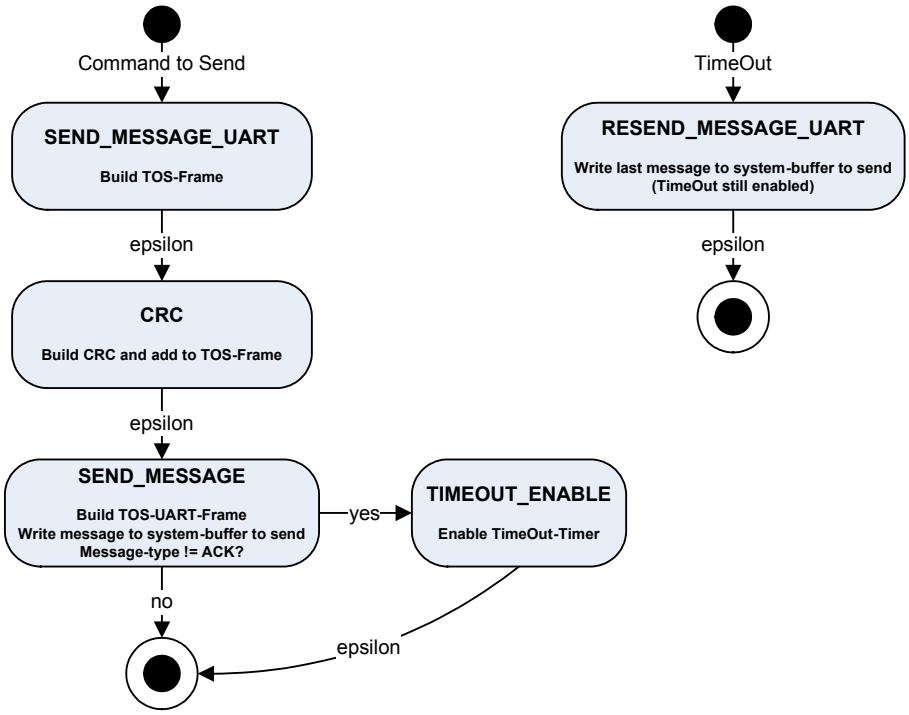


Abbildung 4.39 – Implementation PC-Modul UART 2/2

Wird ein Kommando zum Senden einer Nachricht gegeben, sei es durch den Benutzer, oder das Programm selbst, so wird zuerst ein TOS-Frame gebildet, welcher dem TinyOS-RFM-Frame entspricht.
Nach der Generierung der Checksumme wird der komplette Frame dann in das TinyOS-UART-Frame Format gebracht und anschließend zum Senden an das System übergeben.
Ist die gesendete Nachricht kein ACK, so wird ein TimeOut-Timer gestartet.
Bei dessen Überlauf wird die zuletzt gesendete Nachricht wiederholt.

Weitere Komponenten

Die anderen Module bestehen zum größten Teil aus Systemfunktions-Aufrufen und tragen nicht weiter zum Verständnis der Software bei.
Deshalb soll an dieser Stelle auf dessen Erläuterung verzichtet werden.

4.3.2.4. Gesamt-System

Hier soll an einem exemplarischen Beispiel die Kommunikation zwischen dem Wearable und dem PC gezeigt werden.
Hierzu wird vom Benutzer eine Aufforderung zum Senden von Analyse-Daten an das Wearable geschickt, welches mit einem Datenpaket antwortet.

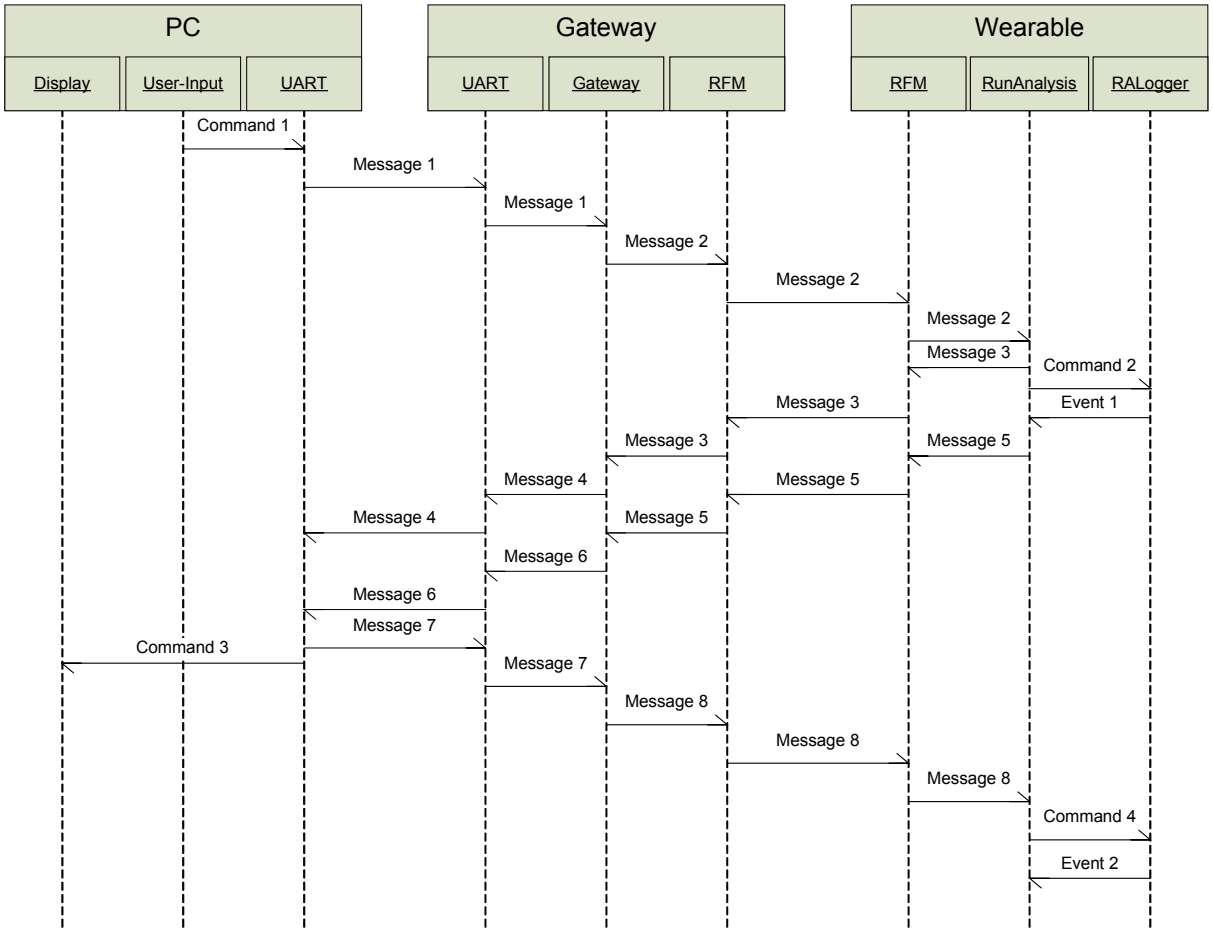


Abbildung 4.40 – System Kommunikations-Aufnahme

Erläuterung:

- Command 1: Der Benutzer sendet eine Nachricht vom Typ CMD_Send_Log
- Message 1: Die Nachricht wird über UART zum Gateway gesendet
- Message 2: Die Nachricht wird über Funk zum Wearable gesendet
- Message 3: Das Wearable bestätigt mit einem ACK über Funk
- Message 4: Das ACK wird vom Gateway über UART zum PC geschickt
- Command 2: Es wird ein Datenpaket aus dem Flash gelesen
- Event 1: Das Datenpaket liegt jetzt vor
- Message 5/6: Das Wearable versendet eine Nachricht vom Typ Log_Data
- Message 7/8: Der PC antwortet mit einem ACK
- Command 3: Die empfangenen Daten werden angezeigt
- Command 4: Nach Empfang des ACKs wird ein weiteres Datenpaket aus dem Flash gelesen und anschließend versendet

Mit jedem empfangenem ACK sendet das Wearable nun weitere Analyse-Daten, bis keine weiteren vorliegen. Der Befehl ‚CMD_Send_Log' muss also nur einmal am Anfang, und nach einem Kommunikations-Abbruch, vom PC gesendet werden.

4.4. Prototyp

Das System wurde entsprechend den Vorgaben aufgebaut und besteht aus den beschriebenen Bestandteilen.
Das Wearable besteht aus einer Pulsuhr der Firma „EKA“, welche einen Brustgurt zur Pulserfassung besitzt, dem Mica2Dot-Board (zur Datenerfassung, Berechnung, Speicherung und Übermittlung per Funk), und den Sensor-Boads MTS510 (zur Messung der Schrittfrequenz) und MDA500 (Anschlussmöglichkeit für die Pulsuhr).
Das Gateway besteht aus dem Seriellen Gateway-Board MIB510 (besitzt einen UART-Transceiver) und einem Mica2-Board.

Das folgende Bild zeigt die aufgebauten und einsatzbereiten Komponenten:

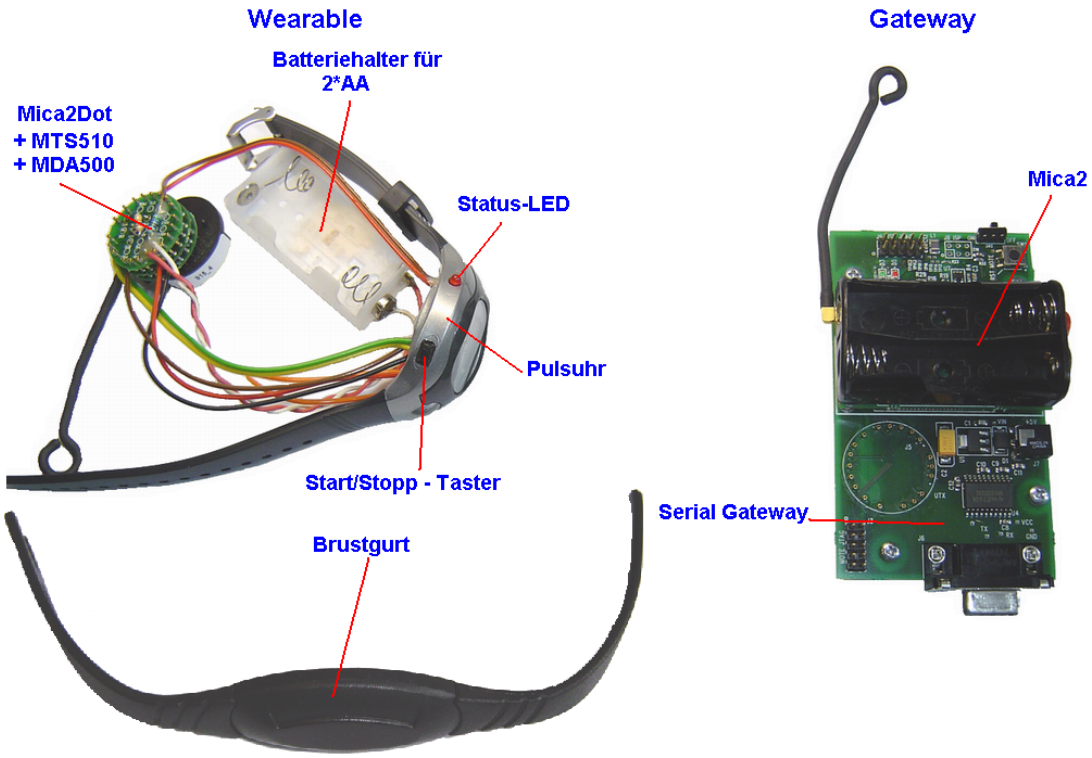


Abbildung 4.41 – Prototyp Hardware-Komponenten

Für die Stromversorgung des Wearables wurde ein Batteriehalter für 2 AA-Zellen angebracht, da diese Batterien eine höhere Kapazität als Knopfzellen besitzen und somit längere Laufzeiten ermöglichen.
Durch die Form des Mica2Dot-Boards lässt es sich direkt unter der Uhr platzieren, sodass das Gerät wie eine normale Uhr getragen werden kann. Nur der externe Batteriehalter muss gesondert befestigt werden.

Abbildung 4.42 – Prototyp Hardware Wearable

Im Einsatz mit dem Brustgurt können nun die gewünschten Daten erfasst, und durch die entwickelte PC-Software analysiert werden.

Folgendes Bild zeigt einen Screenshot der PC-Software mit den Grafiken für die Pulsfrequenz und Schrittfrequenz.

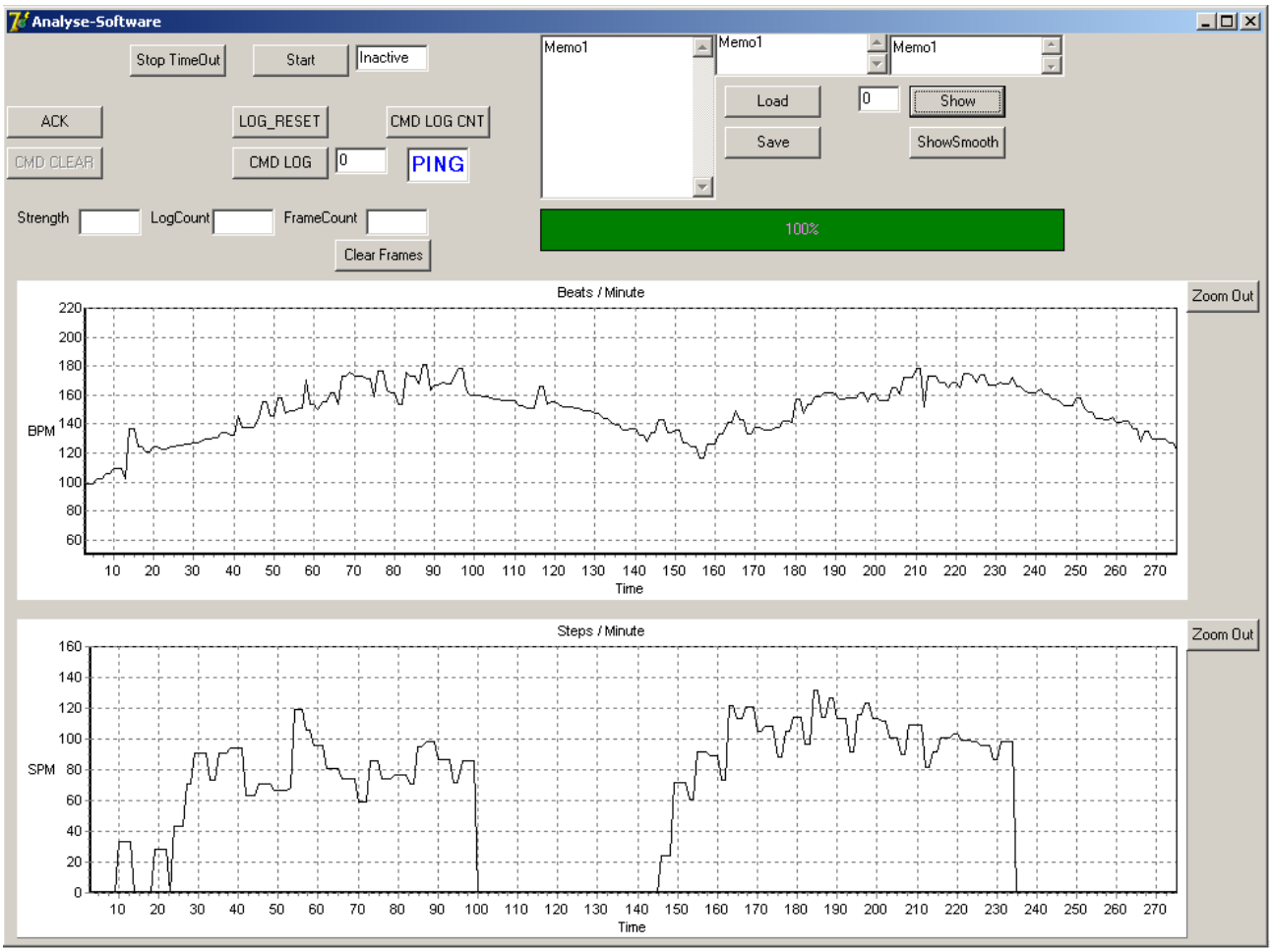


Abbildung 4.43 – Prototyp PC-Software

Im oberen Teil kann der Anwender dem Wearable verschiedene Befehle schicken, Analyse-Daten von der Festplatte laden, oder auf diese speichern.
Weiterhin hat er hier die Möglichkeit einen beliebigen Datensatz anzuzeigen.
Der, hier grün dargestellte, Balken dient bei Kommunikation zur Anzeige der Funk-Feldstärke.
Diese Prototyp-Version besitzt noch eine Reihe weiterer Felder, die nur zu Debug-Zwecken benötigt werden.
Im unteren Teil werden die Analyse-Daten grafisch dargestellt, oben die Pulsfrequenz, unten die Schrittfrequenz.
Das Programm bietet außerdem die Möglichkeit, an einer beliebigen Stelle in die Grafiken zu zoomen, um Teile genauer zu betrachten.

4.4.1. Analyse-Daten

Bei den aufgezeichneten Daten (siehe vorige Seite) beginnt der Lauf, nach einer nicht aufgezeichneten Aufwärmphase, mit einer Pulsfrequenz von ca. 100 und steigt stetig an.
Nach unterschiedlich gelaufenen Geschwindigkeiten, diese sind leider nur sehr grob erkennbar, folgt eine Pause, in welcher sich die Herzfrequenz wieder senkt.
Anschließend wurde ein weiterer kurzer Lauf mit ähnlichen Situationen unternommen.
Die in dieser Grafik dargestellten Daten sind direkt aus dem Speicher des MOTEs gelesen, wurden also mit den vorgestellten Algorithmen und Filtern generiert.

Da die Werte immer noch relativ große Abweichungen zeigen, kann mittels der PC-Software ein weiterer Filter eingeschaltet werden.
Dieses wurde in der folgenden Grafik getan:

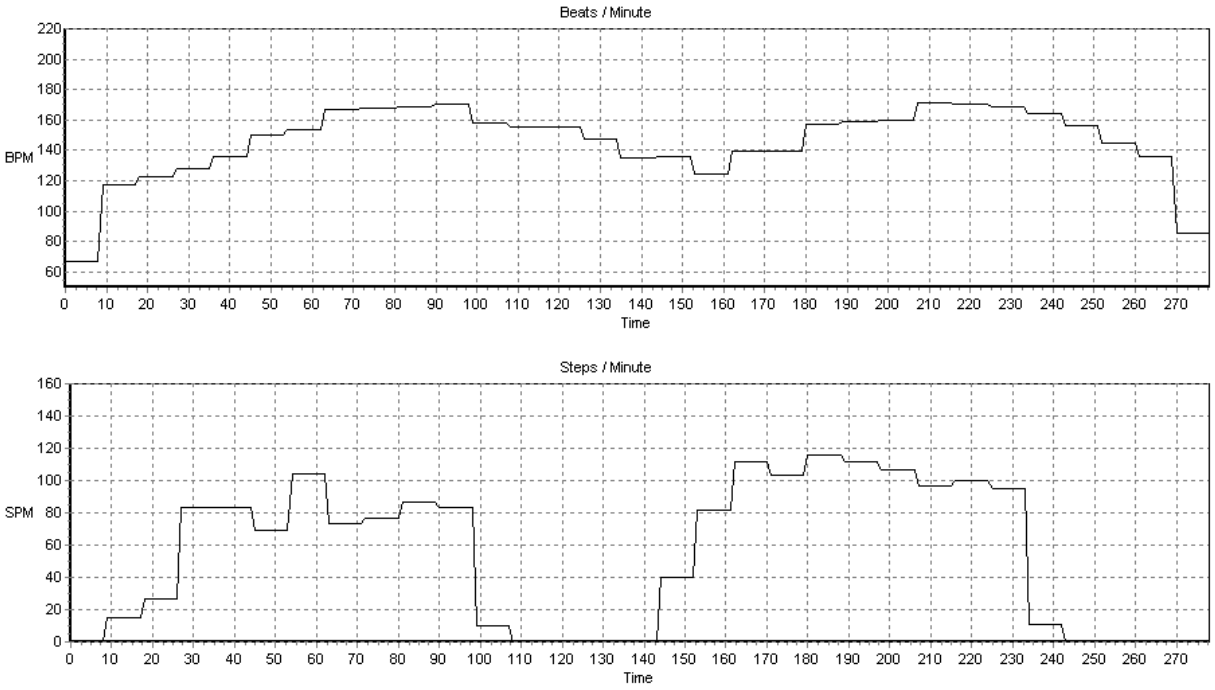


Abbildung 4.44 – Analyse-Daten

Dieser Filter fasst die Daten einiger Sekunden zusammen und bildet über diesen Zeitraum den Mittelwert. Aus diesem Grund sind die Start- und End-Werte entsprechend verzerrt.
Jetzt ist die Präzision auf der Zeitachse zwar verloren, der Verlauf, besonders der Schrittfrequenz, ist aber wesentlich besser zu erkennen.

4.5. Fazit

Diese Anwendung wurde mit den zur Verfügung stehenden Mitteln und dem begrenztem Zeitrahmen dieser Arbeit zu einem funktionierendem Prototyp gebracht. Die gewünschten Daten können aufgezeichnet und am PC analysiert werden. Ob die Analyse einer Schrittfrequenz wirklich Sinn macht, sei einem Sport-Wissenschaftler überlassen, zumal dessen Präzision zu Wünschen übrig lässt.

Die Datenübertragung funktioniert dank des eingesetztem Bestätigungs-Mechanismus einwandfrei.
Es wurde hier kein großer Wert auf Performanz gelegt, was sich allerdings an der Datenrate auch sehr deutlich zeigt:
Es werden im Mittel nur etwa 7 Datenpakete/ Sekunde übertragen, somit also die Daten von ca. 21 Sekunden. Das liegt allerdings mehr an der Performanz von TinyOS, als an den langen Übertragungswegen und Teilstrecken.

Darüber hinaus wurde ersichtlich, dass die MOTE-Technologie in Verbindung mit TinyOS wirklich nur für Sensor-Netzwerke sinnvoll ist, da das System nicht sehr leistungsfähig ist.
Die Software des Wearables scheint die Grenzen des Machbaren (zumindest auf Mica2Dot-Basis) erreicht zu haben.
Die Präzision der gemessenen Daten lässt sehr zu Wünschen übrig, besonders im Falle der Schrittfrequenz.
Das wird mit Sicherheit auch an dem eingesetztem, relativ einfachen, Algorithmus für die Schritt-Detektion liegen. Für einen reellen Einsatz des Systems, besteht hier also Nachholbedarf.

Gleichzeitig brachte die relativ geringe Stromaufnahme von ca. 18mA mehr Probleme mit sich, als sich vermuten lässt.
Der Hersteller gibt für diese Geräte und einer Knopfzelle, die direkt an einem Mica2Dot-Board angeschlossen werden kann, eine Laufzeit von mehreren Wochen oder Monaten an.
Da ein MOTE in einem Sensor-Netzwerk mehr Zeit im Sleep-Modus verbringt, als aktiv tätig zu sein, sind diese Laufzeiten wahrscheinlich auch möglich.
Bei dieser Applikation, mit einem hohen Anteil an Prozessor-Zeit, sind diese Werte bei Weitem nicht erreicht worden.
Schon nach wenigen Minuten ist die Spannung der Knopfzelle so weit abgesunken, dass weder die Datenspeicherung im Flash, noch die Funkübertragung weiterhin einwandfrei funktionieren.
Durch den Einsatz von Standard-AA Zellen konnte dieses Problem aber behoben werden, da diese einen wesentlich kleineren Innenwiderstand, als Knopfzellen, haben, und somit diesen Strom problemlos lange Zeit liefern können.

Für eine zweite Version wäre auch die Kommunikation der Geräte untereinander interessant, sodass sich große Netzwerke bilden können, und die Kommunikationsreichweite ,durch ein Routing über andere Geräte, vergrößert wird.

Fazit & Aussichten

Das für diese Geräte bestimmte Einsatzgebiet, die Überwachung von beliebigen Situation, decken sie bestens ab. Sie dienen der Datenkollektion und Vorverarbeitung, und können mit anderen Einheiten kommunizieren.

Für Anwendungen, die darüber hinaus gehen, also komplexere Applikationen, bei denen Performanz und zeitliche Präzision gefragt sind, eignen sie sich, wie in der im Rahmen dieser Arbeit gezeigten Beispiel-Applikation, nur bedingt.
Auch die Erwartungen an die Datenübertragung dürfen nicht allzu groß sein. Das liegt aber nur zum Teil an der verwendeten Hardware, viel mehr Einfluss hat hier das Betriebssystem TinyOS, welches in diesem Bereich offensichtlich noch nicht genügend Leistung bietet. Die Entwickler des Betriebssystems sind aber um eine ständige Verbesserung bemüht.
Wie das bei dem neusten ZigBee-kompatiblem Gerät MICAz aussieht, konnte hier leider nicht getestet werden. Bei diesem ist die Datenübertragung durch die Hardware wesentlich beschleunigt worden, ob TinyOS diesen Vorteil aber auch ohne große Verluste umsetzen kann, bleibt zu untersuchen.
Gleichzeitig muss berücksichtigt werden, dass bei steigender Performanz natürlich auch der Stromverbrauch steigt, was bei batteriebetriebenen Geräten wiederum keinen Vorteil mit sich bringt.

Zur Zeit stehen eine Reihe von Sensor-Modulen zur Verfügung, sodass durchaus kommerzielle Anwendungen realisiert werden können.
Die Entwicklung geht, soweit nur Standard-Komponenten benutzt werden, relativ schnell vonstatten, da ein großes Repertoire an Beispiel-Applikationen mitgeliefert, bzw. aus dem Internet geladen werden können. Auf dessen Basis lässt sich schnell die eigene Applikation generieren.
Möchte man allerdings tiefer in das System einsteigen, so verlieren sich schnell die Dokumentationen.
Das Gleiche muss leider über die Hardware gesagt werden, je neuer ein Produkt, desto schlechter ist dessen Dokumentation.
Anfragen an den Hersteller Crossbow werden zwar prompt beantwortet, als Support können diese aber meist nicht gewertet werden, da zum Teil auf Schaltpläne in Dokumenten verwiesen wird, die aufgrund extrem schlechter Auflösung, unlesbar sind.

Diese Technologie besitzt ein großes Potential und wird sich in Zukunft seinen festen Platz sichern.
Eine mögliche zukünftige Anwendung ist z.B. die Hausautomatisierung, bei der, dank der Funk-Kommunikation, der Verkabelungsaufwand minimiert wird.
Die Standardisierung des Funk-Protokolls (ZigBee) leistet hier mit Sicherheit seinen Beitrag.

Durch die anhaltende Miniaturisierung werden die Geräte in Zukunft Richtung ‚Disappearing Computer' gehen, also unsichtbar in der Umgebung, autonom agieren, Daten sammeln und miteinander kommunizieren.
Dieses wird auch in der März-Ausgabe 2005 von ‚Communications of the ACM' diskutiert.

Durch eine einfache Möglichkeit der Redundanzbildung, da die Komponenten relativ günstig sind, werden die Gesamtsysteme fehlertolerant.
Das heißt, bei Ausfall einzelner Komponenten, sei es durch Versagen der Stromversorgung, oder durch eine zu große Entfernung für die Funk-Kommunikation, wird zwar die Leistung des Systems reduziert, es bleibt aber trotzdem lauffähig.
Dabei hilft vor allem die Fähigkeit der Spontanen Vernetzung, welche es ermöglicht, ohne großen Administrationsaufwand, weitere Komponenten in ein Netzwerk zu integrieren.

Bis diese Geräte allerdings im großen Rahmen eingesetzt werden können, wird noch einige Zeit vergehen, da die Stückpreise momentan noch zu hoch sind, als dass sie auch für den privaten Einsatz interessant sind.
Bei allmählicher Verbreitung wird sich das aber relativieren.

Abbildungsverzeichnis

Literaturverzeichnis

1 Crossbow (URL: http://www.xbow.com)

1.1 URL: http://www.xbow.com/Support/Support_pdf_files/MPR-MIB_Series_User_Manual_7430-0021-06_A.pdf (Zugriff: 10.04.05)

1.2 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2_Datasheet.pdf (Zugriff: 10.04.05)

1.3 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICA2DOT_Datasheet.pdf (Zugriff: 10.04.05)

1.4 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf (Zugriff: 10.04.05)

1.5 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Cricket_Datasheet.pdf (Zugriff: 10.04.05)

1.6 URL: http://www.xbow.com/Support/Support_pdf_files/MIT_Cricket_Manual_7430-0335-01_A.pdf (Zugriff: 10.04.05)

1.7 URL: http://www.xbow.com/Support/Support_pdf_files/MTSMDA_Series_User_Manual_7430-0020-03_A.pdf (Zugriff: 10.04.05)

1.8 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS_Datasheet.pdf (Zugriff: 10.04.05)

1.9 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MDA300CA_Datasheet.pdf (Zugriff: 10.04.05)

1.10 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS400-420_Datasheet.pdf (Zugriff: 10.04.05)

1.11 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MTS510CA_Datasheet.pdf (Zugriff: 10.04.05)

1.12 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB600CA_Datasheet.pdf (Zugriff: 10.04.05)

1.13 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MIB510CA_Datasheet.pdf (Zugriff: 10.04.05)

1.14 URL: http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Stargate_Datasheet.pdf (Zugriff: 10.04.05)

1.15 URL: http://www.xbow.com/Support/Support_pdf_files/Stargate_Manual.pdf (Zugriff: 10.04.05)

1.16 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/IntroSmartDust.pdf (Zugriff: 10.04.05)

1.17 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/Hardware.pdf (Zugriff: 10.04.05)

1.18 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/TinyOS_Installation.pdf (Zugriff: 10.04.05)

1.19 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/ProgrammingI.pdf (Zugriff: 10.04.05)

1.20 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/ProgrammingII.pdf (Zugriff: 10.04.05)

1.21 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/Wireless.pdf (Zugriff: 10.04.05)

1.22 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/TinyDB-TASK.pdf (Zugriff: 10.04.05)

1.23 URL:http://www.xbow.com/support/Support_pdf_files/Motetraining/Stargate_1.pdf (Zugriff: 10.04.05)

1.24 URL: http://www.xbow.com/support/Support_pdf_files/Motetraining/Crossbow-CricketNew.pdf (Zugriff: 10.04.05)

1.25 URL: http://www.xbow.com/Support/Support_pdf_files/TinyOS_Getting_Started_Guide_7430-0022-03_A.pdf (Zugriff: 10.04.05)

2 Atmel (URL: http://www.atmel.com)
Prozessor ATMega128L
2.1 URL: http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf (Zugriff: 10.04.05)

Flash AT45DB041B
2.2 URL: http://www.atmel.com/dyn/resources/prod_documents/doc3443.pdf (Zugriff: 10.04.05)

3 Chipcon (URL: http://www.chipcon.com)
Transceiver CC1000
3.1 URL: http://www.chipcon.com/files/CC1000_Data_Sheet_2_2.pdf (Zugriff: 10.04.05)

Transceiver CC2420
3.2 URL: http://www.chipcon.com/files/CC2420_Data_Sheet_1_2.pdf (Zugriff: 10.04.05)

4 Maxim (URL: http://www.maxim-ic.com)
Silicon Serial Number DS2401
4.1 URL: http://pdfserv.maxim-ic.com/en/ds/DS2401.pdf (Zugriff: 10.04.05)

5 Panasonic (URL: http://www.panasonic.com)
Mikrofon WM-62A
5.1 URL: http://www.panasonic.com/industrial/components/pdf/em05_wm62_a_c_cc_k_b_dne.pdf (Zugriff: 10.04.05)

6 Analog Devices (URL: http://www.analog.com)
Beschleunigungssensor ADXL202
6.1 URL: http://www.analog.com/UploadedFiles/Data_Sheets/53728567227477ADXL202E_a.pdf (Zugriff: 10.04.05)

7 TinyOS Programming
7.1 URL: http://www.princeton.edu/~msimsir/perv/project/TinyOS.ppt (Zugriff: 10.04.05)

7.2 URL: http://www.cis.upenn.edu/~lee/04cis640/slides1/TinyOS.pdf (Zugriff: 10.04.05)

7.3 URL: http://www.princeton.edu/~wolf/EECS579/imotes/tos_tutorial.pdf (Zugriff: 10.04.05)

7.4 HAW TIS-Vortrag „Tiny OS“ von Christoph Kutzera im Sommersemester 2004

8 NesC

8.1 URL: http://nescc.sourceforge.net/papers/nesc-ref.pdf (V1.1) (Zugriff: 10.04.05)

9 Allgemein

9.1 URL: http://www.tinyos.net/ (Zugriff: 10.04.05)

9.2 HAW TIS-Vortrag „Transpondertechnik“ von Martin Rymsza im Sommersemester 2004

9.3 Communications of the ACM URL: http://www.acm.org/pubs/cacm/ (Zugriff 10.04.05)

A. Sourcecode

Die Quelldateien für das entwickelte System sind auf der beigefügten CD, im Verzeichnis „SourceCodes“ zu finden.

Hier befinden sich die Sourcen für die drei Systemteile:

Verzeichnis	„\Wearable“	für das Wearable
	„\Gateway“	für das Gateway
	„\PC“	für die PC-Software

Enthalten sind dabei nur die Quelldateien. Für eine Übersetzung ist die entsprechende Entwicklungsumgebung erforderlich.

B. Fremddaten

Im Verzeichnis „Schaltpläne“ der CD befindet sich eine Sammlung von Schaltplänen. Diese sind aus Fremd-Dokumenten entnommen und stammen nicht aus eigener Entwicklung.
Daher übernehme ich keine Verantwortung, über dessen Vollständigkeit und Korrektheit.

Versicherung über die Selbständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift