HTBLuVA Mödling

Höhere Lehranstalt für Elektronik
Ausbildungsschwerpunkt Telekommunikation

DIPLOMARBEIT

"Intelligentes drahtloses Messdatenerfassungssystem"

"Intelligent Wireless Data Acquisition System"

Ausgeführt an der Abteilung für Elektronik im Schuljahr 2007/08
An der HTBLuVA Mödling, Technikerstraße 1 – 5, A – 2340 Mödling von:

HOFER	Josef	5AHELT	2264 Jedenspeigen	Schlossplatz 2
KNÖBEL	Patrick	5AHELT	2620 Neunkirchen	Gustav - Dieselstraße 26
LENZ	Dominik	5AHELT	2345 Brunn/Geb.	Roter Kreuzbaumweg 6

Betreuer:
DI. Geza BESZEDICS

Mödling, am 26.05.2008

1. Kurzfassung

Speziell im Bereich des Motorsportes und bei schienengebundenen Fahrzeugen erweist es sich oft als sehr aufwändig, die wirkenden Kräfte auf einer Achse, beziehungsweise die Temperatur eines Lagers zu messen.
Ziel dieses Projektes ist es ein Messsystem zu entwickeln, welches auf den normierten Industriefrequenz-Kanälen von 433MHz arbeitet. Dafür soll eine eigene Empfangseinheit, welche über USB mit dem PC verbunden ist, entwickelt werden. Diese wird als Master bezeichnet. Zur Erfassung der Messdaten sollen Slaves entwickelt werden.
Die Slaves sollen nicht nur als Messstationen dienen, sondern auch bei Funk- Hindernissen in der Lage sein, die Messdaten weiterzuleiten. Der „Funk-Pfad“ soll bei Inbetriebnahme selbst gefunden werden und auch mehrere Verbindungswege ermöglichen.

Da die Sensoren auch bei rauen Umweltbedingungen funktionieren sollen, ist ein geeignetes Gehäuse zu konstruieren und eine induktive Ladung der integrierten Akkus zu entwickeln.

Zur Verarbeitung der Messdaten soll eine Software entwickelt werden.
Diese soll in der Lage sein die Messdaten grafisch anzuzeigen und zu protokollieren.
Der User soll auch die Möglichkeit haben, die Art der Anzeige zu konfigurieren.

2. Abstract

Task of this project is to develop a wireless data logging system in which environmental parameters can be transferred from a Slave device to the Master device which is connected to a recording PC via the USB port.

In this project the measured environmental parameters are humidity, gravitation in three axis, environmental temperature and the surface temperature of a device under test.

The reason for this project was the necessity to obtain the temperature of moving parts, since a wiring can be very difficult or very expensive. The system can be used in rail transportation systems, car crash analyses, mechanical stress tests of vehicles and surveillance of sensitive cargo.

The system has to work at the standard industrial frequency of 433 MHz. In the system there are up to 62 Slaves, which are able to communicate with the Master. The innovation is that all Slaves can work as a rerouter of the information path. There are up to four levels in rerouting of the transmitted data.

In the project the circuitry of the Master and the Slaves have to be designed, and the controlling program for the microcontroller has to be developed.

Furthermore the control program for the PC, which is coordinating the transmitting and receiving of sensor data, has to be written.

The energy sources for the Slaves are Lion-batteries, which are charged due to an inductive coupling of the Slave to a charging station for three devices.

3. Inhaltsverzeichnis

Seite

Seite

Seite

4. Projektplanung

4.1. Pflichtenheft

4.1.1. Ausgangspunkt

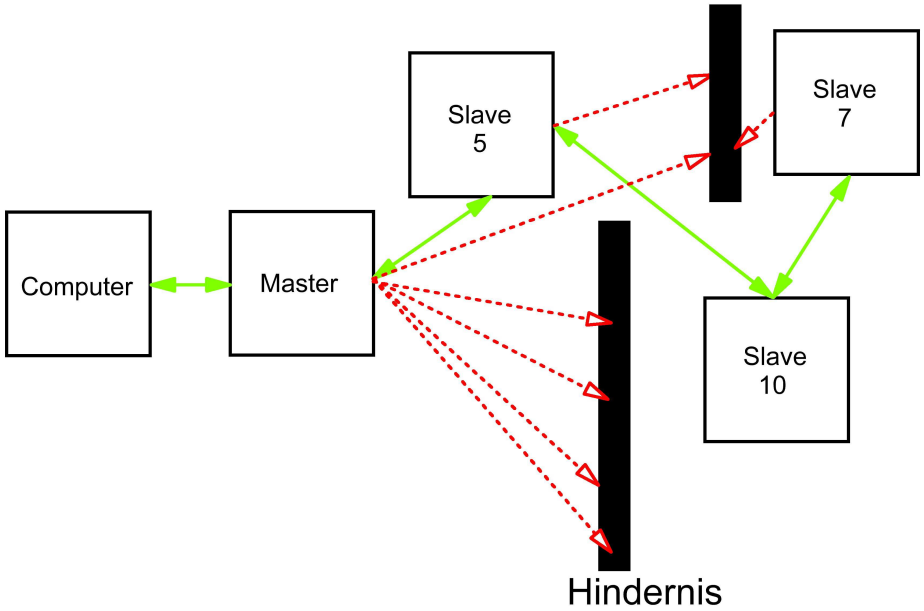
Speziell im Bereich des Motorsportes und bei schienengebundenen Fahrzeugen erweist es sich oft als sehr aufwändig, die wirkenden Kräfte auf einer Achse, beziehungsweise die Temperatur eines Lagers zu messen. Häufig ist dafür eine aufwändige Verkabelung notwendig.
Ziel dieses Projektes ist es ein Messsystem zu entwickeln, welches auf den normierten Industriefrequenz-Kanälen von 433MHz arbeitet, um den Installationsaufwand des Messsystems zu minimieren.

4.1.2. Wege

4.1.2.1. Übliche Variante

Bisherige drahtlose Messsysteme waren nur in der Lage eine direkte Kommunikation zwischen der Empfangseinheit und den Messstationen aufzubauen. Dies sorgt natürlich dafür, dass sobald eine Abschirmung zwischen der Empfangseinheit und der Messstation vorliegt der Funkverkehr unterbrochen oder stark beeinträchtigt wird.
In solchen Fällen müsste man dann auf ein drahtgebundenes Messsystem zurückgreifen.

4.1.2.2. Neue Variante



Die große Innovation des neu entwickelten Systems ist ein Funkprotokoll, welches einen Funkverkehr über mehrere Slaves ermöglicht und ein spezieller Suchalgorithmus, welcher jedem Slave ermöglicht andere Slaves in seiner Umgebung zu finden.
Dadurch ist dieses Messsystem flexibler und zuverlässiger als andere Messsysteme, da immer mehrere Verbindungen zur Wahl stehen und daraus die beste gewählt werden kann.
Sollte eine bestehende Verbindung unterbrochen werden, so wird automatisch auf die nächst beste umgeschaltet.

4.1.2.3. Zielsetzung

Es soll ein möglichst universell einsetzbares Messsystem entwickelt werden, welches über eine PC- Software zu steuern ist.
Die einzelnen Messstationen (Slave) sollen äußerst robust und einfach zu montieren sein und die Energieversorgung soll ein Lithium – Ionen – Akku übernehmen.
Die Empfangseinheit (Master) soll möglichst klein sein (USB – Stick – Größe) und die Energieversorgung soll über den USB – Port erfolgen.
Weiters soll ein eigenes Netzteil entwickelt werden, um die Akkus der Slaves über induktive Kopplung laden zu können.

4.1.3. Allgemeines

Aus der obigen Zielsetzung ergeben sich vier Hauptkomponenten. Deshalb haben wir unser Projekt in die folgenden vier, weitgehend unabhängigen Module aufgespaltet.

- PC – Software
  Sie hat die Funktion der Steuerung des gesamten Messsystems. Weiters ermöglicht die PC Software die Verarbeitung, Anzeige und Speicherung der Messdaten.

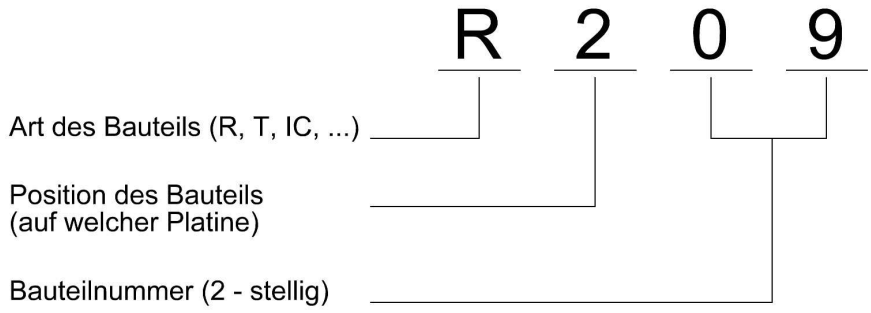
- Master
  Der Master dient als Schnittstelle zwischen PC und dem Messsystem. Außerdem ist er in der Lage eine Kanalbelastungsmessung durchzuführen. Zur Überwachung der Kommunikation mit den Slaves dient ein Mikrocontroller.

- Slave
  Die Aufgabe des Slaves besteht grundsätzlich darin die Daten der Sensoren abzuholen und diese an das Funkprotokoll anzupassen. Weiters gibt es eine Statusleuchtdiode und einen Taster zur Interaktion.

- Netzteil
  Das Netzteil dient zum Laden der Akkus von bis zu drei Slaves gleichzeitig.

Benennung der Bauelemente:

Damit an der Bauteilbezeichnung sofort erkannt werden kann welcher Bauteil welchem Modul zuzuordnen ist, haben wir uns auf folgende Nummerierung der Bauteile geeinigt:

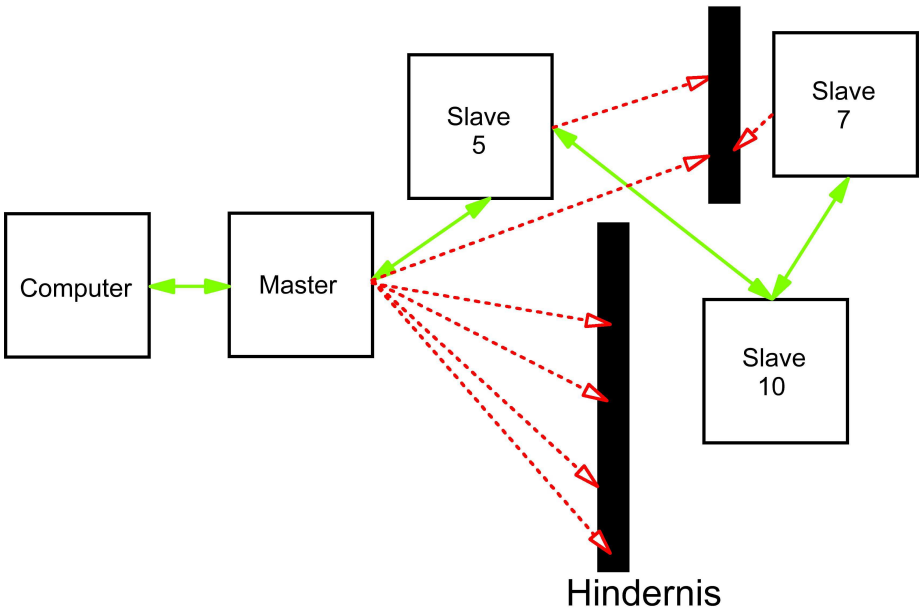


Legende der Bauteilposition:
1: Master - Platine
2: Slave - Platine
3: Temperaturabnehmer - Platine
4: Netzteil - Platine

4.1.4. Anforderungen an das Messsystem

- Einfache Bedienung
- Schnelles Aufbauen und Verwalten einer Messumgebung
- Zuverlässigkeit
- Lange Betriebsdauer der Slaves ca. 24 Stunden
- Unterstützung von bis zu 62 Slaves
- Datenrate von bis zu 3,2 kBit/s
- Grafische Anzeige der Messdaten
- Protokollierung der Messwerte
- Zu erfassende Messdaten
    - Feuchtigkeit
    - Gravitation
    - Temperatur
        - Umgebungstemperatur
        - Punktuelle Temperatur

4.1.5. Schematische Darstellung des Messsystems



4.1.6. PC – Software

4.1.6.1. Funktionelle Anforderung

- Die Computer – Software hat als Hauptaufgabe die Steuerung des Messsystems.
- Außerdem dient sie zur Auswertung der Messdaten.
- Weiters soll sie in der Lage sein die Messdaten grafisch anzuzeigen.
- Die Protokollierung der Messdaten soll im CSV- oder Benutzerdefinierten – Format möglich sein.

4.1.6.2. Dokumentation

- Die Dokumentation der PC - Software umfasst einerseits die Bedienung der Software zum Einrichten der Messumgebung und andererseits die Dokumentation des Programmcodes.
- Außerdem wird hier das Funkprotokoll genau dokumentiert.

4.1.7. Master

4.1.7.1. Funktionelle Anforderung

- Der Master dient zur Umwandlung des PC – USB – Signals in ein für das Funkmodul verständliches RS232- Signals.
- Weiters hat der Master die Aufgabe eine Kanalbelastungsmessung durchzuführen.
- Außerdem soll der Master vorab fehlerhafte empfangene Daten verwerfen.

4.1.7.2. Elektrische Anforderung

- Der Master soll ein USB – Signal in ein RS232 – Signal umwandeln können.
- Weiters benötigt der Master ein Funkmodul, dass aus einem RS232 – Signal ein FM - Signal im 433MHz Band erzeugen kann.
- Außerdem benötigt der Master einen A/D – Wandler um die Kanalbelastungsmessung durchführen zu können.
- Für die Aussortierung fehlerhafter Daten wird ein Mikrokontroller benötigt.
- Zur Sende- und Empfangsanzeige sollen zwei LED's dienen. Eine blaue LED soll den Sendestaus anzeigen und eine Rote den Empfangsstatus.
- Der Master soll direkt über den USB- Port versorgt werden.

4.1.7.3. Programmtechnische Anforderung

- Für den Mikrokontroller des Masters muss ein C – Programm geschrieben werden, welches zum Aussortieren fehlerhafter Daten dient und die Kanalbelastungsmessung veranlasst.

4.1.7.4. Mechanische Anforderung

- Der Master soll möglichst klein sein (ca. 70x30x30) und in USB- Stick- Form realisiert werden.
- Das Gehäuse soll aus PMMA bestehen und eine SMB – Buchse zum Anschließen einer Antenne besitzen.

4.1.7.5. Dokumentation

- Es müssen sämtliche elektrischen Berechnungen und alle elektrischen und mechanischen Fertigungsunterlagen (Schaltplan, Layout, Bohrplan, Bestückungsplan, Stückliste und Gehäusezeichnungen) dokumentiert werden.
- Weiters muss auch das PIC – Programm dokumentiert werden.

4.1.8. Slave

4.1.8.1. Funktionelle Anforderung

- Die Hauptaufgabe des Slaves besteht darin die vier Messsensoren (Außentemperatur, Luftfeuchtigkeit, Gravitation und punktuelle Temperatur) ständig abzufragen und die aktuellsten Messwerte zwischen zu speichern.
    - Gravitation
        - Bereich: ±6g
        - Auflösung: 12Bit
        - Minimale Auflösung: 0,00294 g/ LSB
    - Luftfeuchtigkeit
        - Bereich: 0 … 100 %RH
        - Auflösung: 12Bit
        - Minimale Auflösung: 0,03 %RH/ LSB (± 3%RH Toleranz)
    - Außentemperatur
        - Bereich: -40 … +124 °C
        - Auflösung: 14Bit
        - Minimale Auflösung: 0,01 C°/ LSB (± 0,5°C Toleranz)
    - Punktuelle Temperatur
        - Bereich: -40 … +150 °C
        - Auflösung: 12Bit
        - Minimale Auflösung: 0,03125 °C/ LSB (± 0,5°C Toleranz)
- Weiters soll der Slave das Funkprotokoll verarbeiten und die Befehle des Computers interpretieren können.
- Außerdem soll er als Relaisstation fungieren können. Dies bedeutet weiterleiten der Daten und Befehle an einen benachbarten gefundenen Slave.
- Zum Senden der Messdaten und zum Empfangen der Befehle sollen RS232 – Signale dienen.

4.1.8.2. Elektrische Anforderung

- Die Energieversorgung des Slaves soll ein Lithium – Ionen – Akku mit 3,7V/1200mAh übernehmen.
- Zum Versorgen des Mikrokontrollers, Funkmoduls, Außentemperatursensors und punktuelle Temperatursensors soll ein 5V Aufwärtswandler dienen.
- Die Versorgung des Gravitationssensors soll ein 3,3V – Spannungsregler übernehmen.
- Der Akku soll über induktive Kopplung mit einem LION – Laderegler geladen werden können. Der Laderegler wird über das Netzteil extern versorgt.
- Die Verarbeitung der Funk - Befehle, Abfrage der Sensoren und die Anpassung der Messdaten an das Funkprotokoll soll der PIC 18F1320 übernehmen.
- Zur Umwandlung des RS232 – Signals in ein FM- Signal und umgekehrt, soll ein Funkmodul im 433 MHz Band dienen.

4.1.8.3. Programmtechnische Anforderung

- Für den Mikrokontroller des Slaves muss ein C – Programm geschrieben werden, welches zum Abfragen der Messsensoren dient, die Anpassung der Messdaten an das Funkprotokoll und die Interpretation der Funk – Befehle übernimmt.

4.1.8.4. Mechanische Anforderung

- Das Gehäuse des Slaves muss äußerst robust sein, um auch bei rauen Umweltbedingungen eingesetzt werden zu können.
- Weiters müssen Öffnungen für die induktive Ladung, einen Funktionstaster, eine Signal - LED, den Feuchtigkeitssensor und die Antenne vorgesehen werden.
- Dennoch sollte es spritzwasserfest sein.
- Das Gehäuse soll so konstruiert werden, dass mehrere Aufsätze, zur Befestigung an verschiedenen Oberflächen, möglich sind.

4.1.8.5. Dokumentation

- Es müssen sämtliche elektrischen Berechnungen und alle elektrischen und mechanischen Fertigungsunterlagen (Schaltplan, Layout, Bohrplan, Bestückungsplan, Stückliste und Gehäusezeichnungen) dokumentiert werden.
- Weiters muss auch das PIC – Programm dokumentiert werden.

4.1.9. Netzteil

4.1.9.1. Funktionelle Anforderung

- Das Netzteil soll zum Laden von bis zu drei Slaves über die induktive Kopplung dienen.

4.1.9.2. Elektrische Anforderung

- Die Versorgung des Netzteils soll über ein extern zugekauftes 12V / 1,8A Steckernetzteil erfolgen.
- Jeder Ladeteil des Netzteils benötigt eine Strombegrenzung für 500mA und eine Spannungsstabilisierung von 5V.
- Die Signalisierung eines Fehlers soll eine rote LED übernehmen und die Signalisierung eines Ladevorgangs übernimmt eine grüne LED.
- Zur Erzeugung einer 100kHz Wechselspannung aus 5V Gleichspannung soll ein IC-Baustein dienen, der nach dem Push – Pull Schaltnetzteil Prinzip arbeitet. Dies wird zur Energieübertragung per induktive Kopplung benötigt.
- Die Energieübertragung soll mit Hilfe von zwei Trafohalbschalen in RM8- Bauform realisiert werden. Eine Halbschale ist in der Ladestation angebracht und die andere im Slave.

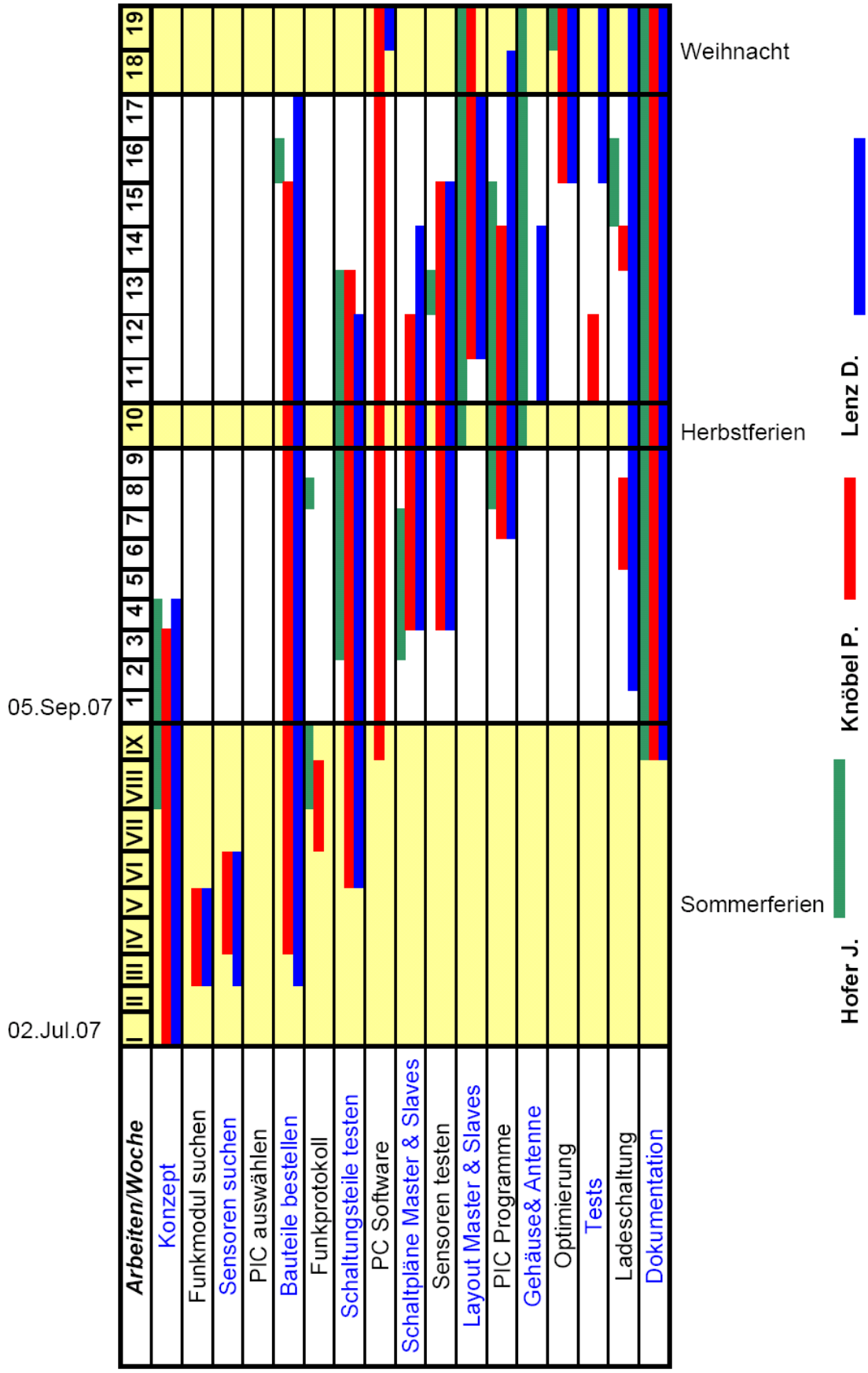
4.1.9.3. Mechanische Anforderung

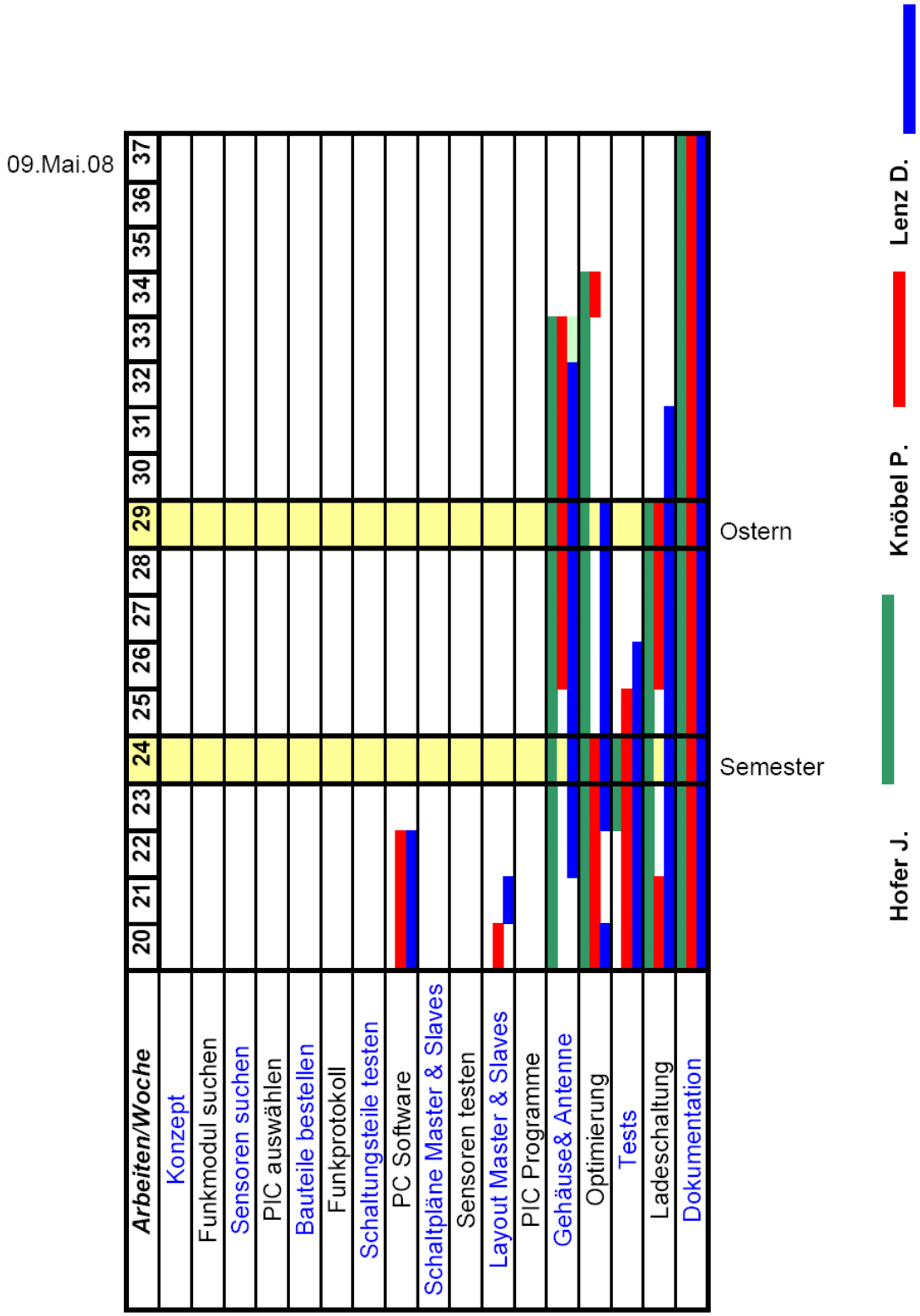
- Das Netzteil soll aus thermischen Gründen aus Aluminium gefertigt werden.
- Es sind dabei Öffnungen für das externe Steckernetzteil, die induktive Kopplung und die Signal – LED vorzusehen.

4.1.9.4. Dokumentation

- Es müssen sämtliche elektrischen Berechnungen und alle elektrischen und mechanischen Fertigungsunterlagen (Schaltplan, Layout, Bohrplan, Bestückungsplan, Stückliste und Gehäusezeichnungen) dokumentiert werden.

4.2. Zeitschiene (Milestones)



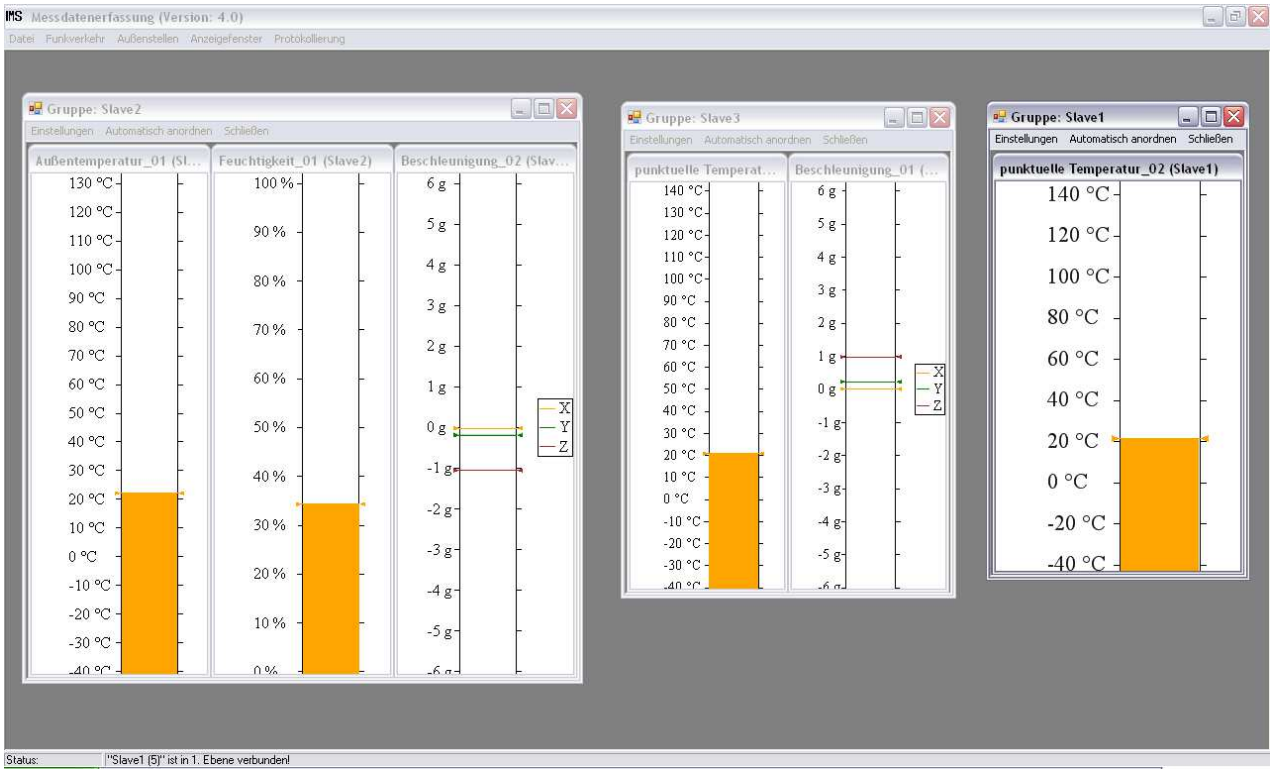

09.Mai.08
Arbeiten/Woche
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
Konzept
Funkmodul suchen
Sensoren suchen
PIC auswählen
Bauteile bestellen
Funkprotokoll
Schaltungsteile testen
PC Software
Schaltpläne Master & Slaves
Sensoren testen
Layout Master & Slaves
PIC Programme
Gehäuse& Antenne
Optimierung
Tests
Ladeschaltung
Dokumentation
Ostern
Semester
Hofer J.
Knöbel P.
Lenz D.

5. Projektdokumentation

5.1. Istzustand

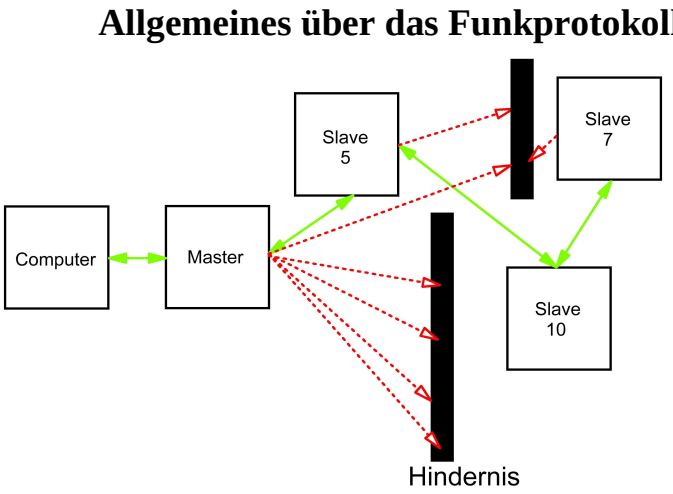
5.1.1. PC- Software

Bedienoberfläche im Messzustand:



Konkretes Beispiel zur Anwendung der PC- Software. Hier werden gerade Messwerte visualisiert.

5.1.1.1. Funkprotokoll



Da das Messsystem auch in einem stark abgeschirmten Umfeld gut arbeiten soll, ist es notwendig nicht nur direkt mit den Slaves kommunizieren zu können, sondern auch indirekt über andere Slaves. Ein Slave der zum Weiterleiten von Daten verwendet wird, wird Relaisstation genannt.

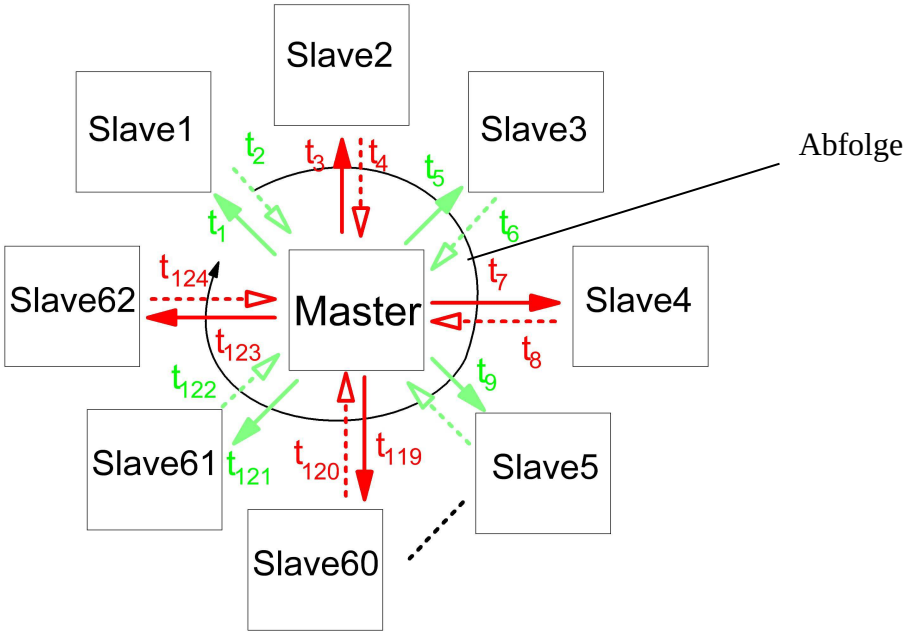
Wichte Eigenschaften des Funkprotokolls

Das Funkprotokoll weißt folgende Eigenschaften auf:

- Verwendung von bis zu 3 zwischen Relaisstationen
- Checksumme zur Fehlererkennung
- Bis zu 62 verschiedene Slaves können pro Messung verwendet werden
- Möglichkeit zum Suchen von Slaves auch über Relaisstationen
- Dynamische Größe der zur übertragenden Datenmenge (0 bis 7 Bytes)
- Das Protokoll ist für ein Master/ Slave - System entwickelt

Abfrageprinzip

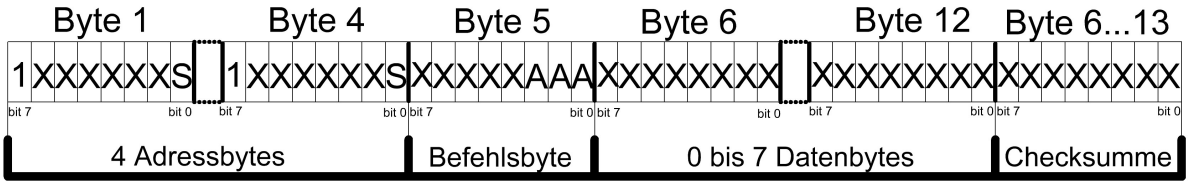
Da es sich bei dem Protokoll um ein Master/ Slave - System handelt, kann immer nur ein Slave gleichzeitig abgefragt werden:



Somit wird es bei einer großen Anzahl von Slaves immer wichtiger nur dann Messwerte abzufragen, wenn sie wirklich benötigt werden. Dazu wurde in der Software ein ausgeklügeltes Prioritätensystem eingebaut, um unnötige Abfragen und somit zu lange Wartezeiten bei der Abfrage der verschiedenen Messwerte zu verhindern.

Aufbau des Funkprotokolls

Das Funkprotokoll sieht minimal 6 und maximal 13 Bytes vor, die folgende Aufgaben haben:



X...beliebiger Wert
S...Statusbit
A...Anzahl der Folgebytes
1...konstanter Wert für Header

Adressbytes

Die ersten 4 gesendeten Bytes entsprechen den Adressbytes, wobei die einzelnen Bytes für Master- , Slave- , oder Relaisstationsadressierung zuständig sind.
Bit 7 ist in jedem Adressbyte 1 und wird benötigt, um eine schnellere Synchronisierung zu ermöglichen.
Bit 0 (S) gibt den Status an. Ist S=1 so ist die Adresse aktiv, ist S=0 so ist die Adresse inaktiv. Die Funktion dieses Bits wird weiter unten genauer erläutert.

Befehlsbyte

Byte 5 gibt den jeweiligen Befehlstyp an, wobei Bit 0 bis 2 die Länge der nachfolgenden Datenbytes angeben.

Datenbytes

Byte 6 bis Byte 12 können Datenbytes sein. Es kann aber auch der Fall sein, dass für gewisse Befehle keine Datenbytes benötigt werden. Ist dies der Fall, so werden die Datenbytes weggelassen und die Checksumme angehängt.

Checksummenbyte

Die Checksumme ist eine XOR- Verknüpfung der einzelnen Bytes und bildet immer den Abschluss. Es kann als Byte Nummer 6 kommen aber auch erst als Byte Nummer 13 je nachdem wie viele Datenbytes übertragen wurden.

Befehle und Antworten

Je nach dem ob der Master einen Befehl schickt oder der Slave antwortet, hat das Befehlsbyte andere zulässige Werte:

Befehle vom Master zum Slave

Nr.	Befehlsbyte	Anzahl der Datenbytes	Bedeutung	Antwort vom Slave
1	00000	1 (001)	Neue Adresse setzen	Befehl 14
2	00001	4 (100)	ID setzen	Befehl 14
3	00010	0 (000)	Bist du da? mit ID	Befehl 13
4	00011	0 (000)	Bist du da? ohne ID	Befehl 14
5	00100	1 (001)	Sende Messergebnis	Befehl 8,9,10,11,12
6	00101	0 (000)	Blinke LED	Befehl 14
7	00110	1 (001)	Wechsle den Kanal	Befehl 14
15	01110	0 (000)	Schalte dich aus	Befehl 14

Genauere Erklärung zu den einzelnen Befehlen:

- **Neue Adresse setzen**
  Wird verwendet, um den Slave eine neue Adresse zuzuweisen. Die Antwort des Slaves dauert um 4ms länger, da der Slave den neuen Adresswert im EEPROM speichert.
- **ID setzen**
  Wird verwendet, um den Slave eine eindeutige ID zuzuweisen. Die ID wird verwendet um einen Slave leichter wieder zu erkennen.
  Die Antwort des Slaves dauert um 16ms länger, da der Slave die neue ID im EEPROM speichert.
- **Bist du da? mit ID**
  Mit Hilfe dieses Befehls ist es möglich die ID des Slaves abzufragen.
- **Bist du da? ohne ID**
  Hier antwortet der Slave mit einer bestimmten Antwort. Dies wird verwendet um Slaves zu suchen.
- **Sende Messergebnis**
  Hier wird der Slave aufgefordert einen bestimmten Messwert aus dem Speicher zurückzugeben.
  Mit dem mitgelieferten Datenbyte kann der Messtyp bestimmt werden:

Byte 6	Bedeutung
00000000	Sende Messergebnis Beschleunigung
00000001	Sende Messergebnis Feuchtigkeit
00000010	Sende Messergebnis Temperatur1(Feucht./Temp.)
00000011	Sende Messergebnis Temperatur 2
00000100	Sende Messergebnis Batteriestand

- **Blinke LED**
  Wenn dieser Befehl an einen Slave geschickt wird, dann blinkt diese 5-mal.

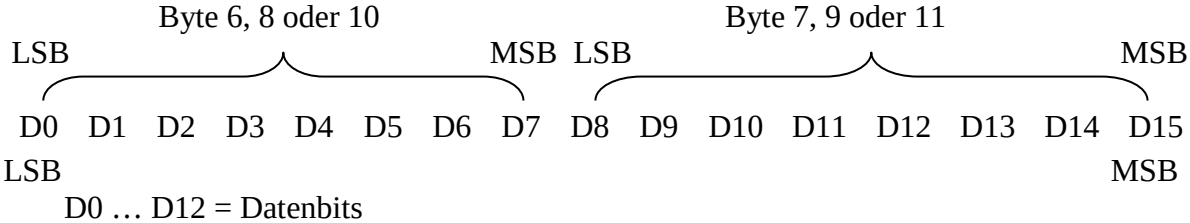
- **Wechsle den Kanal**
  Dieser Befehl startet den Kanalwechsel. Der Kanalwechsel wird aber erst nach 10 Sekunden durchgeführt. Innerhalb dieser Zeit ist der Slave noch auf dem alten Kanal erreichbar.
  Dies wird verwendet, um mehrfach nachzuprüfen, ob der Slave umgeschaltet wurde. Weiters kann dieser Slave während dieser Zeit noch als Relaisstation fungieren und zum Umzuschalten der Kanäle anderer Slaves verwendet werden.
  Die Antwort des Slaves dauert um 4ms länger, da des Slave den neuen Adresswert im EEPROM speichert.
- **Schalte dich aus**
  Dieser Befehl veranlasst den Slave sich abzuschalten. Das Abschalten wird aber erst nach 2 Sekunden durchgeführt. Dies hat den Vorteil, dass man noch nachprüfen kann ob sich der Slave auch wirklich den Befehl erhalten hat.

Antworten vom Slave zum Master

Nr.	Befehlsbyte	Anzahl der Datenbytes	Bedeutung	Antwort auf
8	00111	6 (110)	Sendet Messergebnis Beschleunigung	Befehl 5
9	01000	2 (010)	Sendet Messergebnis Feuchtigkeit	Befehl 5
10	01001	2 (010)	Sendet Messergebnis Temperatur1 (Umgebungstemperatur)	Befehl 5
11	01010	2 (010)	Sendet Messergebnis Temperatur 2 (punktuelle Temperatur)	Befehl 5
12	01011	1 (001)	Sendet Messergebnis Batteriestand	Befehl 5
13	01100	4 (100)	Bin da mit ID!	Befehl 3
14	01101	0 (000)	Bin da ohne ID!	Befehl 1,2,4,6,7,15
16	01111	1 (001)	Messfehler	Befehl 5

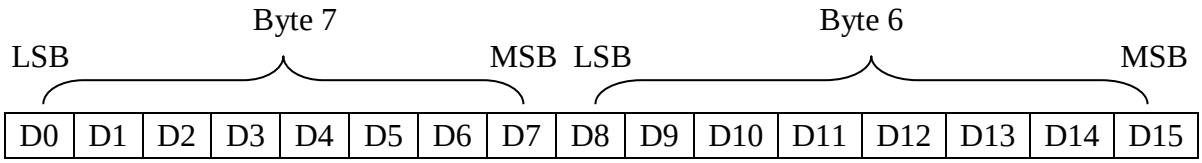
Genauere Erklärung zu den einzelnen Antworten:

- **Sendet Messergebnis Beschleunigung**
  Sendet die Messwerte der Beschleunigungsmessung zurück zum Master. Diese Werte entsprechen immer dem Spitzenwert der in der Zwischenzeit durchgeführten Messungen. Byte 6 und 7 enthalten den X- Wert, Byte 8 und 9 enthalten den Y- Wert, Byte 10 und11 enthalten den Z- Wert.
  Die Daten werden wie folgt übertragen:



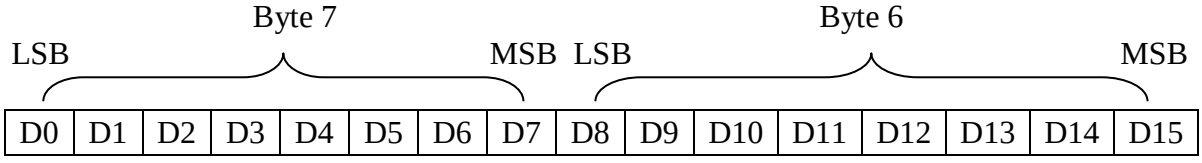
D0 … D12 = Datenbits
D13 … D15 = Vorzeichenbits

- **Sendet Messergebnis Feuchtigkeit**
  Sendet den Messwert der Feuchtigkeitsmessung zurück zum Master. Die Messdaten sind in den Bytes 6 und 7 enthalten.
  Die Daten werden wie folgt übertragen:



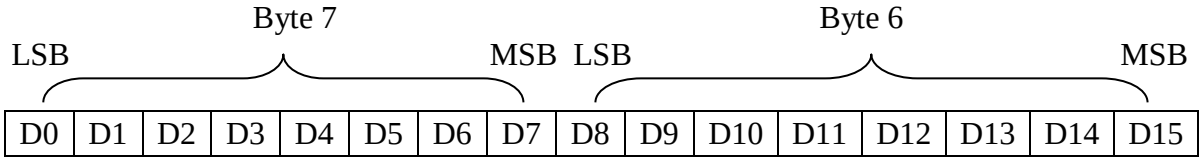
  D0 … D11 = Datenbits
  D12 … D15 = nicht in Verwendung

- **Sendet Messergebnis Temperatur1 (Umgebungstemperatur)**
  Sendet den Messwert der Umgebungstemperaturmessung zurück zum Master. Die Messdaten sind in den Bytes 6 und 7 enthalten.
  Die Daten werden wie folgt übertragen:



  D0 … D13 = Datenbits
  D14 … D15 = nicht in Verwendung

- **Sendet Messergebnis Temperatur 2 (punktuelle Temperatur)**
  Sendet den Messwert der punktuellen Temperaturmessung zurück zum Master. Die Messdaten sind in den Bytes 6 und 7 enthalten.
  Die Daten werden wie folgt übertragen:



  D0 … D12 = Datenbits
  D13 = Vorzeichenbit
  D14 … D15 = nicht in Verwendung

- **Sendet Messergebnis Batteriestand**
  Sendet den Messwert Batteriestatus zurück zum Master. Byte 6 enthält den Wert 1, wenn der Spannungswert des Akkus größer als 3V ist und 0 wenn der Spannungswert kleiner als 3V ist.
- **Bin da mit ID!**
  Sendet die ID zum Master. Byte 6 bis 9 enthalten die ID.
- **Bin da ohne ID!**
  Wird bei vielen Befehlen des Masters als Antwort zurückgegeben.
- **Messfehler!**
  Wird gesendet wenn bei irgendeiner Messungsanfrage ein Fehler vorliegt. Im mitgelieferten Datenbyte sind folgende Zustände möglich:

Byte 6	Bedeutung
00000000	Keine neuen Messdaten
00000001	Kein Sensor vorhanden

Es ist somit möglich gewisse Sensoren auf einen Slaves nicht zu bestücken. Werden keine Sensoren bestückt, dient dieser Slave nur als Relaisstation.

Antworten des Masters

Dasselbe Protokoll wird auch verwendet wenn der Master über eine Handshakeleitung aufgefordert wird Daten zu senden:

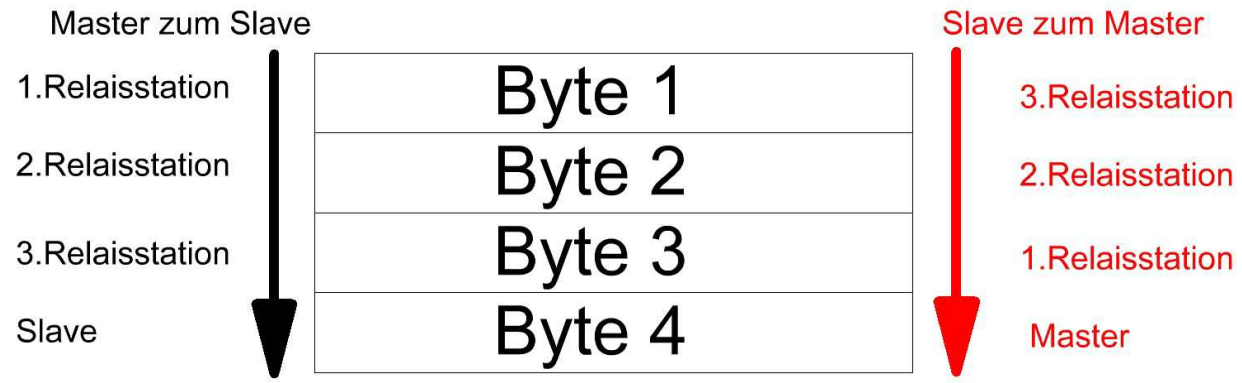
Flag	Byte 5	Anzahl der Daten	Bedeutung
DTR	01101	0 (000)	Bin da!
RTS	11111	2 (010)	Kanalbelastungswert (Byte 6 und 7)

Genauere Erklärung zu den einzelnen Antworten:

- **Bin da!**
  Beim setzen der Handshakeleitung DTR sendet der Master dem PC die Bestätigung, dass er vorhanden ist.
- **Kanalbelastungswert (Byte 6 und 7)**
  Beim Setzen der Handshakeleitung RTS wird eine Kanalbelastungsmessung des aktuellen Kanals durchgeführt. Die Messwerte werden dann in Byte 6 und 7 an den PC geschickt.

Genauere Informationen zu den Adressbytes

Die Adressen sind in 4 Ebenen unterteilt:



Die Abarbeitung der Adressen erfolgt von Byte 1 bis Byte 4 ausgehend vom Master. Das heißt wenn der Master das Paket sendet, ist das Byte 1 aktiv und die 1. Relaisstation wird angesprochen. Dann wird von der 1. Relaisstation das Byte 2 aktiviert und das Byte 1 deaktiviert. Damit wird es weiter an die 2. Relaisstation gesendet. Dieser Vorgang wird fortgesetzt bis der Slave das Paket bekommt.
Der Slave muss dann die Reihung der Adressen wie oben angeführt ändern um die Antwort auf denselben Weg wieder zurück zu senden.
Alle 4 Adressbytes werden nur dann benötigt, wenn der Funkpfad 3 Relaisstationen benötigt. Werden weniger Adressen benötigt, so werden alle nicht benötigten Adressbytes auf $10000000_{(B)}$ gesetzt. Bei direkter Kommunikation wird nur Byte 1 mit der Adresse des Slaves versehen.

Spezialadressen:

10000000.10000000.10000000.10000000
Diese Bytefolge signalisiert einem Slave im Adressiermodus, dass er angesprochen ist.
11111111
Ist die Masteradresse.

5.1.1.2. Programmaufbau

Im folgenden Kapitel wird der Istzustand der PC- Software beschreiben:

Seite

Allgemeines

Auf den folgenden Seiten wird ein grober Überblick über den Aufbau der PC- Software gegeben.
Weitere Informationen sind auf der beiliegenden DVD im Dokument „PC Software\C# Programmdokumentation_V2.4.doc“ zu finden.
Die einzelnen Klassen sind auf der DVD im Ordner „PC Software\C# Programm“ zu finden.
Die Software wurde mit Hilfe der Programmierumgebung „Microsoft Visual Studio Solution 2005 Professional Edition“ in C# programmiert.
Folgende Anforderungen wurden an die Software gestellt:

- Steuerung des Messsystems
- Auswertung der Messdaten
- Messdaten grafisch anzeigen
- Protokollierung der Messdaten im CSV- oder Benutzerdefinierten – Format

Übersicht

Hier soll ein Überblick über alle Klassen geboten werden, um den generellen Zusammenhang des Programms zu verstehen.
Achtung: Das OOP (**O**bjekt **o**rientiertes **P**rogrammierung)- Konzept wurde in vielen Klassen nicht hundertprozentig eingehalten. Die Klassen wurden eher als logische Gruppierung des Quellcodes benutzt.

Vorhandene Klassen

- **K_Hauptprogramm**
  Das Hautprogramm wird gestartet, wenn das Programm aufgerufen wird und führt für den Start des Programms wichtige Funktionen aus. Dann wird mit Hilfe des Hauptprogramms das Mainform aufgerufen.
  Sobald das Mainform geschlossen wird, wird im Hauptprogramm alles zum Schließen des Programms in die Wege geleitet.
- **K_MainForm**
  Ist das Hauptfenster des Programms von dem aus der Benutzer alle möglichen Funktionen aufrufen kann.
  Die Klasse hat Verbindungen zu jeder anderen Klasse, um die gewünschten Anfragen weiterzuleiten.
  Weiters bietet es die Möglichkeit als Gruppe für Anzeigefenster verwendet zu werden.
- **K_Einstellungen**
  Ist eine einfach aufgebaute und selbst programmierte Datenbank zur Verwaltung von allen möglichen Einstellungen.
  Die Einstellungen werden in einer editierbaren Form in einer Textdatei abgespeichert und wieder abgerufen.
- **K_Variablen**
  Hier werden wichtige Informationen temporär zwischengespeichert.
  Die gesamte Datenübergabe vom Funkverkehr zu den Anzeigen und dem Protokoll erfolgt über diese Klasse.
  Weiters werden auch die Kalibrationskurven über die Variablenklasse bearbeitet und verwaltet.
- **K_Sprachen**
  Hit Hilfe dieser Klasse ist es möglich die Sprache des Programms zu ändern. Jeder sprachabhängige Text wird von dieser Klasse aus einer Textdatei ausgelesen, somit ist man in der Lage bei Auswahl einer anderen Sprachdatei die Sprache zu wechseln

- **K_Eigenschaften**
  Mit Hilfe dieser Klasse ist es möglich dynamisch während der Laufzeit ein Eigenschaftsfenster zu erstellen. Diese Klasse wird immer dann verwendet, wenn Einstellungen vorzunehmen sind.
  Um verschiedene Abläufe der Software besser zu verstehen, ist es notwendig sich mit dieser Klasse auseinanderzusetzen.
- **K_Anzeigen**
  Wird verwendet, um alle angezeigten Anzeigefenster und Gruppenfenster zu verwalten. Weiters werden hier auch die Vorlagen für die Anzeigefenster verwaltet.
- **K_GruppenForm**
  Wird verwendet um mehrere Anzeigefenster zu gruppieren.
  Achtung: Diese Klasse wird im Programm mehrfach initialisiert.
- **K_AnzeigeFenster**
  Diese Klasse repräsentiert ein Anzeigefenster.
  Achtung: Diese Klasse wird im Programm mehrfach initialisiert.
  Mit Hilfe der Klasse K_Paint werden hier die Messwerte optisch dargestellt, und viele andere Einstellungen sind noch möglich.
- **K_Paint**
  Mit Hilfe dieser Klasse ist es möglich diverse Zeichnungen zu erstellen und dadurch können spezielle Diagramme für die Darstellung von Messwerten erstellt werden.
  Achtung: Diese Klasse wird im Programm mehrfach initialisiert.
- **K_Beep**
  Mit Hilfe dieser Klasse wird es ermöglicht Töne auszugeben. Da das gesamte Programm sehr asynchron aufgebaut ist, wird es hier ermöglicht dennoch einen periodischen Ton auszugeben.
- **K_Funkverwaltung**
  Diese Klasse besteht aus zwei Teildateien: „Funkverwaltung.cs“ und „Funkverwaltung.Funkverkehr.cs“.
  Das File „Funkverwaltung.cs“ übernimmt die Synchronisationsaufgaben und die Weitergabe der empfangenen Daten.
  Das File „Funkverwaltung.Funkverkehr.cs“ hingegen beschäftigt sich mit der Realtime-Kommunikation.
- **K_Port_List**
  Ist eine Verbindungsklasse zu einer Systemressource zum Abfragen der verfügbaren Ports. Mit Hilfe dieser Ports wird nach dem Master gesucht.
- **K_Protokoll**
  Diese Klasse beinhaltet ein sehr einfaches Protokollsystem. Damit ist es möglich die gemessenen Daten temporär zu speichern und später in einem gewünschten Format zu exportieren.

Multithreading

Das gesamte Programm arbeitet mit Hilfe von mehreren unabhängigen Threads. Dadurch entstehen viele Vorteile und auch einige Nachteile:

- **Vorteile**
  - Verarbeitung von mehreren Aufgaben nahezu gleichzeitig
  - Dynamisches aufteilen der Rechenleistung
  - Vermeiden von unnötigen Wartezeiten

- **Nachteile**
  - Threadübergreifende Variablen- Zugriffe müssen gesichert werden
  - Zugriffe auf Formulare, die nicht mit demselben Thread erstellt wurden, sind verboten!
  - Synchronisationen sind notwendig

Folgende Klassen verwenden unabhängige Threads:

- **K_Paint**
  Hier wird ein Thread verwendet, um die Zeichenfläche neu zu zeichnen.
- **K_Beep**
  Um die Tonausgabe zu koordinieren, wird ein Thread verwendet.
- **K_Funkverwaltung**
  Hier werden sogar zwei Threads verwendet, um die Daten abzufragen und weiterzugeben und ein temporärer Thread zum Ausführen von Funkbefehlen.

Verweise zu anderen Klassen

Um in einer Klasse auf andere Klassen zuzugreifen, werden beim Erstellen einer Klasse die benötigten Zeiger übergeben oder nachgeladen.
In fast jeder Klasse ist einer der folgenden Verweise zu finden:

```
private K_Sprachen Sprache;
private K_Variablen Variablen;
private K_Einstellungen Einstellungen;
private K_Funkverwaltung Funkverwaltung;
private K_Anzeigen Anzeigen;
private K_Protokolle Protokoll;
```

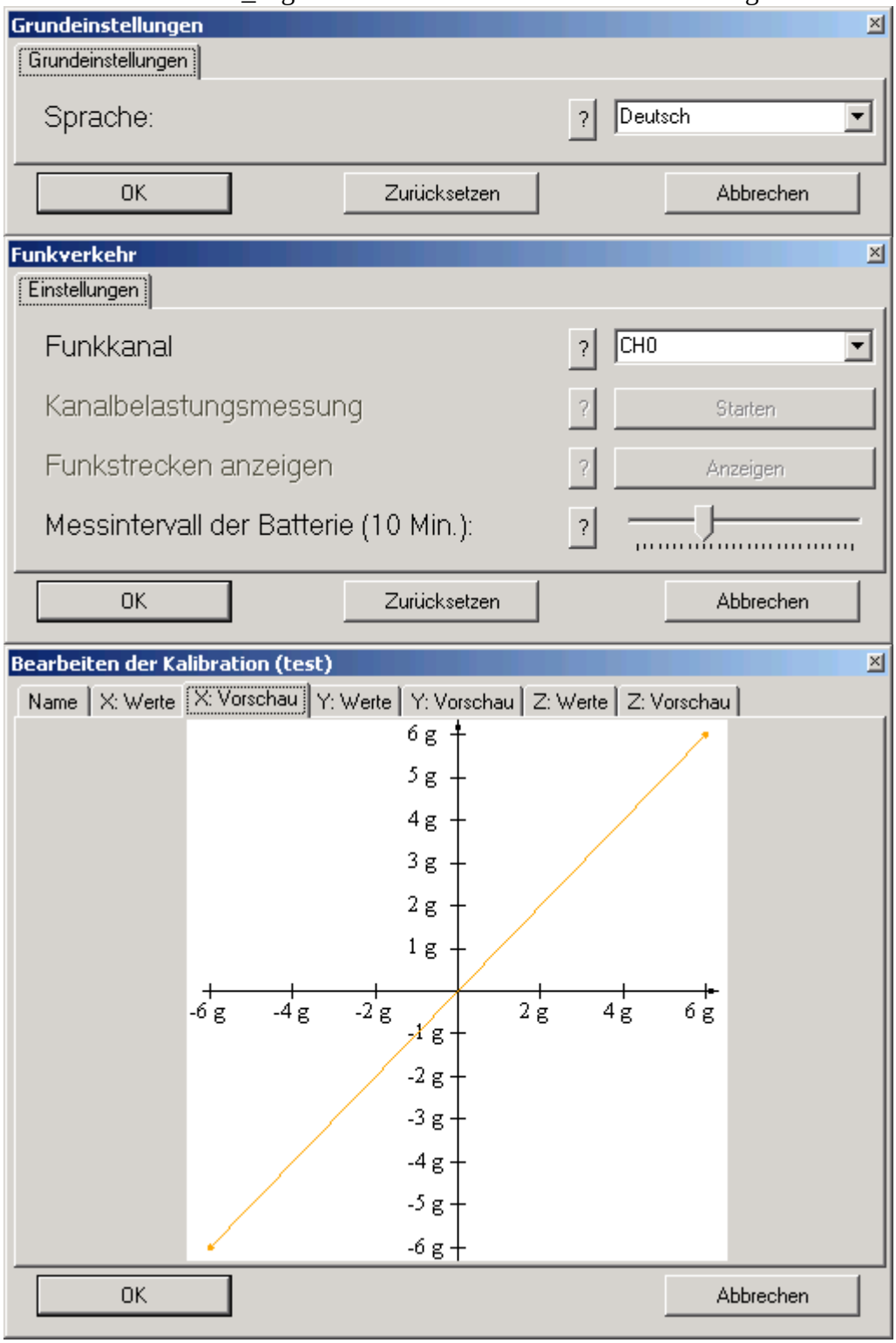
Speziell die Klassen: K_Vaiablen, K_Sprachen, K_Einstellungen werden sehr oft verwendet

K_Eigenschaften

Zum besseren Verständnis des Programms ist es sehr wichtig zuerst den Aufbau des oft verwendeten Eigenschaftsfensters zu verstehen.

In der Klasse wurde das OOP- Konzept eingehalten.

Das aus der Klasse K_Eigenschaften erstellte Fenster kann wie folgt aussehen:



Allgemeine Aufgaben

Diese Klasse kann im Grunde wie eine Messagebox verwendet werden und bietet viele weitere Möglichkeiten:

- Bietet die Möglichkeit Einstellungen vorzunehmen
- Dynamisches Erstellen des Eigenschaftsfensters während der Laufzeit
- Automatische optische Anordnung der Eigenschaften
- Sperren von Eigenschaftsgruppen
- Ausblenden von nicht benötigten Registrierkarten
- Kontrolle der eingegebenen Werte:
  - Testen, ob etwas eingegeben wurde
  - Testen, ob eine Zahl eingegeben wurde
  - Testen, ob eingegebene Zahl in einem bestimmten Bereich liegt
  - Testen, ob eine voreingestellte Eingabe ausgewählt wurde
- Buttons zum Drücken als Eingabemöglichkeit
- Anzeigen von Zeichnungen für die Kalibrationsverwaltung

Verwendungsbeispiel

Zuerst muss eine Instanz der Klasse erstellt werden:

```
K_Eigenschaften Eigenschaftsfenster = new
   K_Eigenschaften(Sprache,Einstellungen,"Überschrift");
```

Sprache und Einstellungen sind Zeiger auf K_Sprachen und K_Einstellungen. Die Überschrift wird später dann als Titel des Fensters angezeigt.

Jetzt muss eine Registrierkarte erstellt werden, wenn dies nicht passiert kann es zu einem Absturz kommen!

```
Eigenschaftsfenster.add_Tab("Tab_Name");
```

Nun kann schon eine Eigenschaft angelegt werden:

```
int Eigenschaft = Eigenschaftsfenster.add_Eigenschaft("Eigenschaftsname",
   "Beschreibung",K_Eigenschaften.E_Typen.TextBox);
```

Hier wird eine Textbox als Eigenschaft erstellt.

Nun können noch weitere Einstellungen mit Hilfe des Indexes der Eigenschaft durchgeführt werden:

```
Eigenschaftsfenster.Set_Standard_Wert(Eigenschaft, "Standardwert");
```

So kann ein Standardwert gesetzt werden. Für weitere Einstellungen sind die oben angeführten Funktionen zu verwenden.

Nun kann wieder ein neues Tab angelegt werden:

```
Eigenschaftsfenster.add_Tab("Neu_Tab_Name");
```

Weitere Eigenschaften sind auch möglich.

Achtung: Wenn auf einen Tab keine aktiven Eigenschaften angezeigt werden, dann wird dieses Tab nicht angezeigt!

Zum Anzeigen des Fensters wird folgendermaßen vorgegangen:

```
Eigenschaftsfenster.ShowDialog();
```

Diese Funktion gibt auch zurück, ob auf „Abbrechen“ gedrückt wurde.

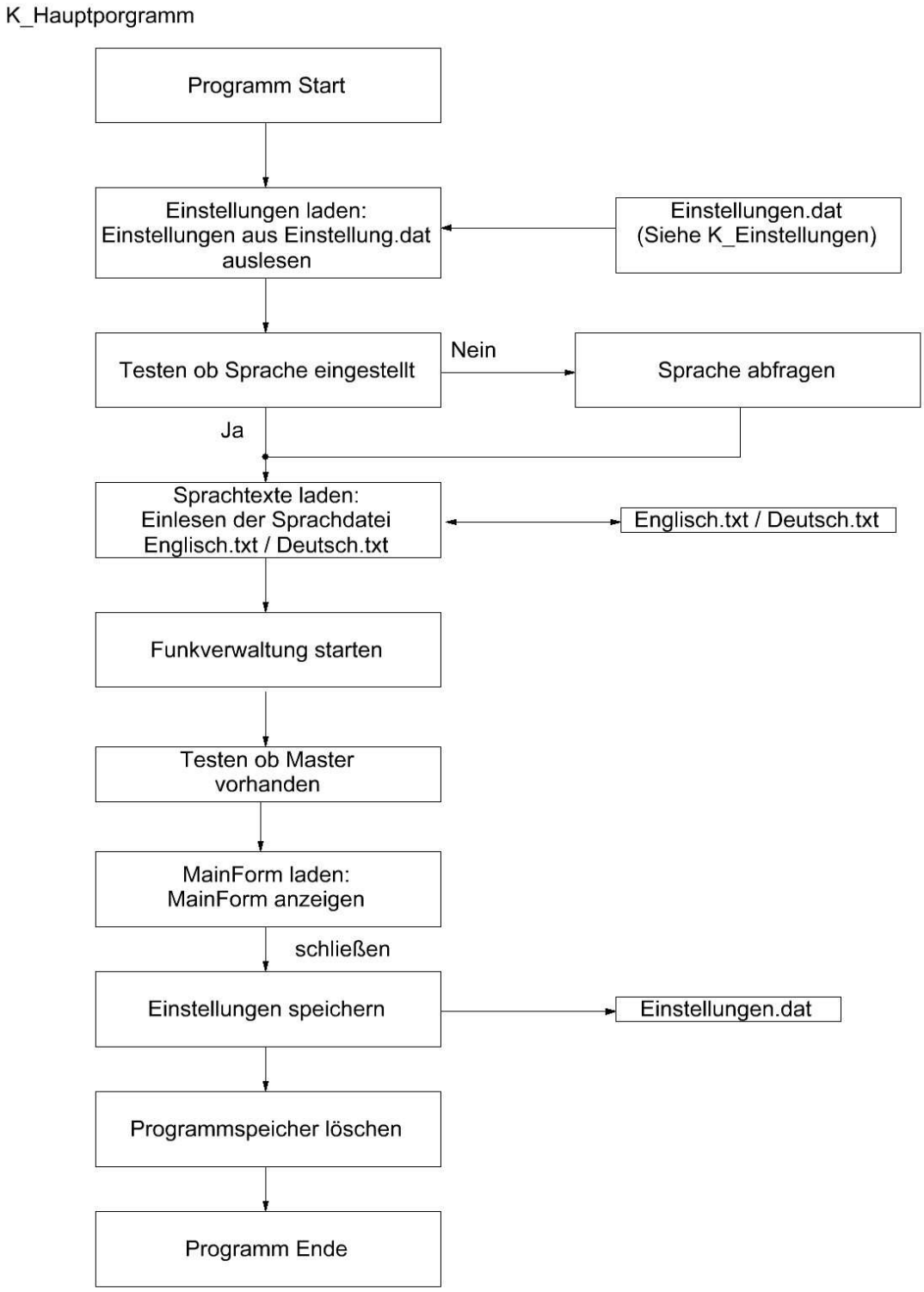
Um die Ressourcen wieder freizugeben, wird wie folgt vorgegangen:

```
Eigenschaftsfenster.Dispose();
```

K_Hauptprogramm

Das Hautprogramm wird gestartet, wenn das Programm aufgerufen wird und führt für den Start des Programms wichtige Funktionen aus. Dann wird mit Hilfe des Hauptprogramms das Mainform aufgerufen.
Sobald das Mainform geschlossen wird, wird im Hauptprogramm alles zum Schließen des Programms in die Wege geleitet.

Prinzipieller Ablauf des Hauptprogramms



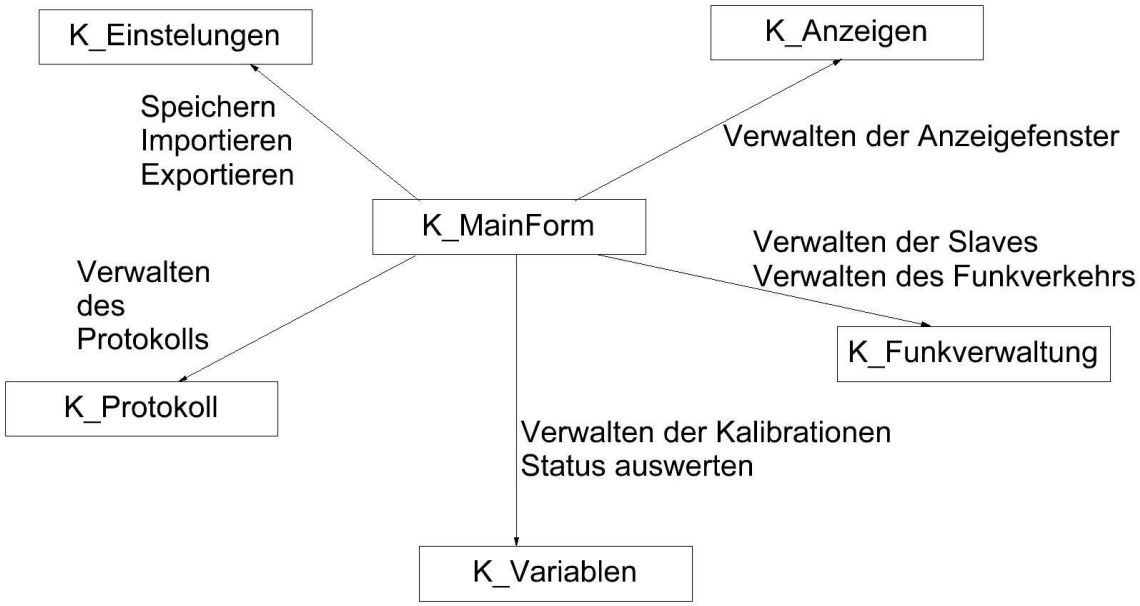
Genauer Ablauf des Hauptprogramms

- Testen, ob die Software bereits gestartet wurde
  - Wenn ja, dann die gerade geöffnete Software sofort beenden
- Programmverzeichnis abfragen und sichern für spätere Dateizugriffe
- Klasse K_Variablen initialisieren
- Klasse K_Einstellungen initialisieren
  - Somit werden automatisch die Einstellungen geladen
- Die Einstellungen für die K_Variablen nachladen
  - Daten über die Slaves
  - Daten über die Kalibrationen
- Klasse K_Sprachen initialisieren
  - Wenn noch keine Sprache eingestellt wurde, wird diese abgefragt
  - Sprachdatei einlesen
- Klasse K_Funkverwaltung initialisieren
  - Aktiviert die notwendigen Threads
- Nun wird nach dem Master gesucht
- Klasse K_Anzeige initialisieren
- Klasse K_Protokoll initialisieren
- Klasse K_MainForm initialisieren
- Nun muss für die Klasse Anzeigen das MainForm nachgeladen werden
- Einstellungen laden
  - Für die Tonausgabe
  - Für die minimale Schriftgröße
- MainForm anzeigen
- Wenn notwendig werden jetzt die Anzeigefenster und Gruppenfenster der letzten Messung mit den benötigten Einstellungen wiederhergestellt
  - Dies kann nur korrekt ablaufen, wenn das MainFrom angezeigt wird! Ist dies nicht der Fall kann für die Zeichnung im Anzeigefenster kein Verweis erstellt werden
- Abarbeiten von Ereignissen und Warten bis der Benutzer das Programm beendet
  - Das Programm wird beendet, wenn das MainForm geschlossen wird
- Speichern der aktuellen Messumgebung
  - Dies wird in der Klasse MainForm vorgenommen, da dies im Hauptprogramm nicht möglich ist, weil dann schon das Fenster geschlossen sein würde
- Speichern der Einstellungen
  - Mit Hilfe eines Events werden alle Einstellungen eingesammelt und dann gespeichert
- Zerstören aller Klassen
- Testen, ob Neustart gewollt
  - Ein Neustart ist dann erforderlich, wenn der Benutzer z.B. die Einstellungen zurücksetzt
- Durch verlassen der Funktion Main() wird das Programm beendet

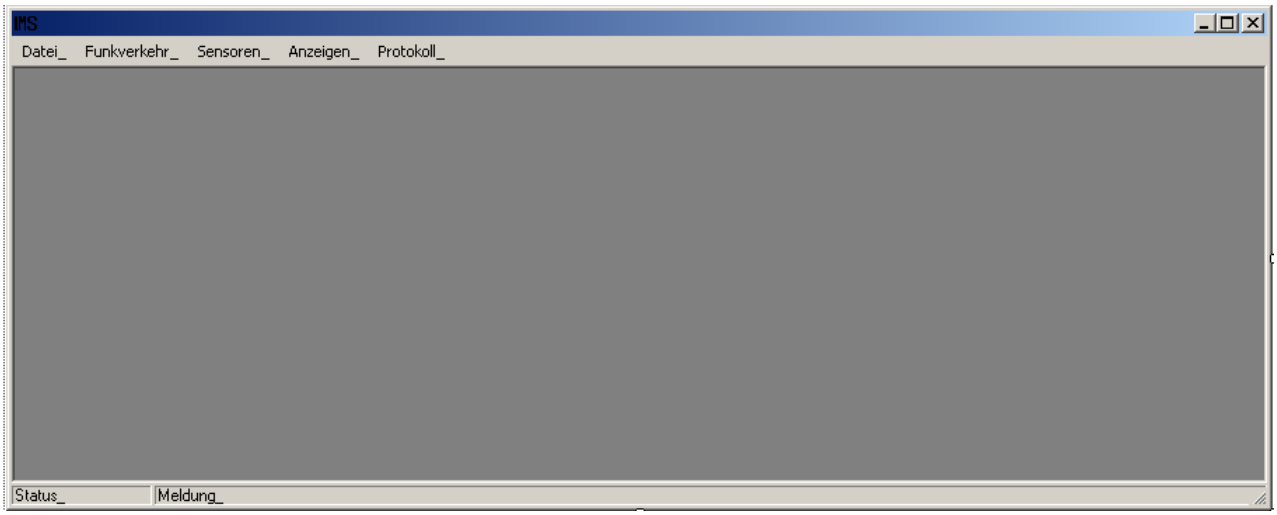
K_MainForm

Ist das Hauptfenster des Programms, von dem aus der Benutzer alle möglichen Funktionen aufgerufen kann.
Die Klasse hat Verbindungen zu jeder anderen Klasse, um die gewünschten Anfragen weiterzuleiten.
Weiters bietet es die Möglichkeit als Gruppe für Anzeigefenster verwendet zu werden.

Verbindungen zu anderen Klassen



Aussehen des Mainforms



Über das Menü werden die einzelnen Funktionen in den verschiedenen Klassen aufgerufen.
Das Menü ist wie folgt aufgebaut:

- Dateien_
  - Grundeinstellungen_ => Intern über `mIGundeinstellungen_Click()`
  - Einstellungen_
    - Speichern_ => K_Einstellungen über `mISpeichern_Click()`
    - Importieren_ => Intern über `mIImportieren_Click()`
    - Exportieren_ => Intern über `mIExportieren_Click()`
  - Beenden_ => Intern über `mIBeenden_Click()`
- Funkverkehr_
  - Starten_ => K_Funkverwaltung über `mIFStarten_Click()`
  - Stoppen_ => K_Funkverwaltung über `mIFStoppen_Click()`
  - Einstellungen_ => K_Funkverwaltung über `mIFEinstellungen_Click()`
  - Sensoren neu Suchen_ => K_Funkverwaltung über `mIFFind_Click()`
- Sensoren_
  - Hinzufügen_
    - Neu_ => K_Funkverwaltung über `mISHNeu_Click()`
    - Vorhanden_ => K_Funkverwaltung über `mISHVorhanden_Click()`
  - Entfernen_ => K_Funkverwaltung über `mISEntfernen_Click()`
  - Abschalten_ => K_Funkverwaltung über `mISAbschalten_Click()`
  - Status_ => K_Funkverwaltung über `mISStatus_Click()`
  - Kalibration verwalten_ => K_Variablen über `mIKali_verarbeiten_Click()`
- Anzeigen_
  - Hinzufügen_
    - Neu_ => K_Anzeigen über `mIAHNeu_Click()`
    - Vorlage_ => K_ Anzeigen über `mIAHVorlage_Click()`
    - Gruppen_ => K_ Anzeigen über `mIAHGruppe_Click()`
    - Mehrere Fenster_ Anzeigen => K_ über `mIAHMFenster_Click()`
  - Entfernen_ => K_Anzeigen über `mIEntfernen_Click()`
  - Vorlagen verwalten_ => K_Anzeigen über `mIVerwalten_Click()`
  - Automatisch anordnen_ => Intern über `mIAAutomatisch_anordnen_Click()`
    - Vertikal_ => Intern über `mIAAutomatisch_anordnen_Click()`
    - Horizontal_ => Intern über `mIAAutomatisch_anordnen_Click()`
- Protokoll_
  - Starten_ => K_Protokoll über `mIPStart_Click()`
  - Stoppen_ => K_Protokoll über `mIPStoppen_Click()`
  - Daten löschen_ => K_Protokoll über `mIPDaten_loeschen_Click()`
  - Daten exportieren_ => K_Protokoll über `mIPDaten_exportieren_Click()`
  - Meldungsfenster_ => K_Protokoll über `mIPMeldungsfenster_Click()`

K_Einstellungen

Ist eine einfach aufgebaute und selbst programmierte Datenbank zur Verwaltung von allen möglichen Einstellungen.
Die Einstellungen werden in einer editierbaren Form in einer Textdatei abgespeichert und wieder abgerufen.

Allgemeines

Die Einstellungen werden in der Datei „**Einstellungen.dat**“ abgespeichert. Diese Datei befindet sich im selben Verzeichnis wie die Anwendung.
Die Datei kann mit einem Texteditor wie „Notepad.exe“ editiert werden. Alle Eigenschaften werden in Klartext angezeigt.
Das Kommerzeichen wird immer als „.“ (Punkt) ausgeführt, um unterschiedliche Ländereinstellungen zu umgehen.

Aufbau der Datei

Am Anfang der Datei muss folgendes stehen:
„[Einstellungen]“

Eine Einstellung einspricht einer Zeile in der Datei.
Die Zeilen sind wie folgt aufgebaut:
Einstellungsname=Einstellungswert

Wichtige Einstellungen

Hier folgt eine Liste, der wichtigsten Einstellungsnamen und deren Bedeutung:

- Bat_Messungsintervall
  - Beschreibt wie oft der Batteriestatus abgefragt werden soll
  - Der Wert muss in Sekunden angegeben werden
  - Kommerzahlen sind nicht zulässig
  - Der Wert darf minimal 60 und maximal 1800 betragen
- Funkkanal
  - Gibt an, auf welchem Kanal das Messsystem sendet
  - Der Wert muss im Bereich [0,9] liegen
- Main_Frame_Vordergrund
  - Gibt an, ob das Messsystem immer im Vordergrund sein soll
  - „True“ und „False“ sind zulässige Eigenschaftswerte
- Paint_Min_schriftgroesse
  - Gibt die minimale Schriftgröße der Darstellung von Messwerten an
  - Kommazahlen sind nicht zulässig
  - Der Wert darf nicht kleiner als 1 und nicht größer als 30 sein
- Sprache
  - Name der Sprache
- TonEinAus
  - Gibt an, ob das Messsystem Töne ausgibt oder nicht
  - „True“ und „False“ sind zulässige Eigenschaftswerte
- Aussenstelle_“XX“_B_ID
  - Diese Eigenschaft gibt die ID eines Slavese an
  - Der Wert muss zwischen 0 und 4294967296 liegen

    - „XX“ = Adresse des Slaves
- Aussenstelle_“XX“_D_Kanal
    - Diese Eigenschaft gibt den Kanal an auf dem der Slave zuletzt erreicht wurde
    - Der Wert muss im Bereich [0,9] liegen
    - „XX“ = Adresse des Slaves

Die übrigen Einstellungen sollten nicht unbedingt manipuliert werden. Sollten die Einstellungen dennoch manipuliert werden, ist es bei einer Fehleingabe möglich, dass die Software nicht mehr ordnungsgemäß funktioniert.

K_Variablen

Hier werden wichtige Informationen temporär zwischengespeichert.
Die gesamte Datenübergabe vom Funkverkehr zu den Anzeigen und dem Protokoll erfolgt über diese Klasse.
Weiters werden auch die Kalibrationskurven über die Variablenklasse bearbeitet und verwaltet.

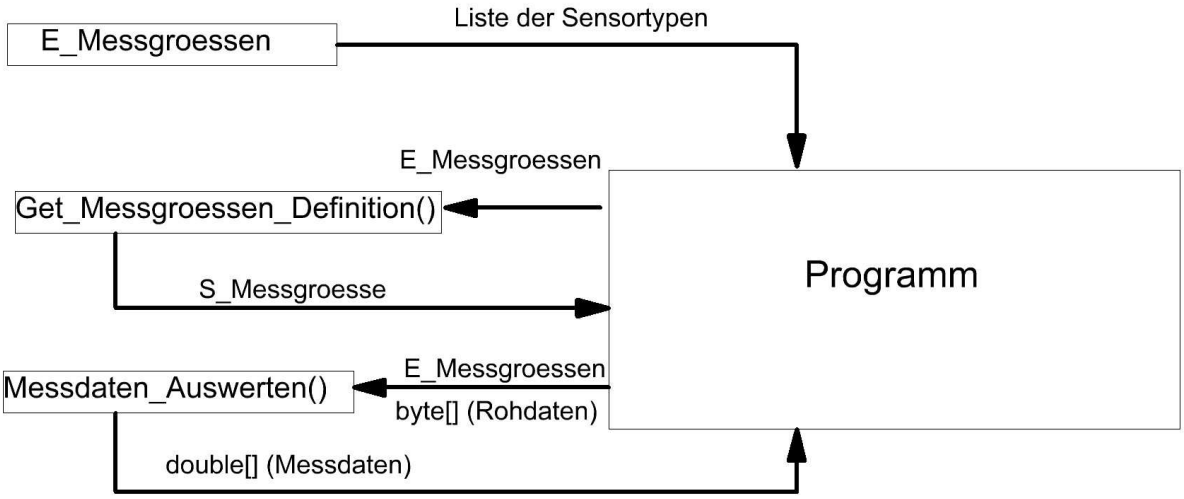
Aufgaben der Klasse

- Generelle Variablen verwalten
- Dynamische Beschreibung der Messtypen
- Mögliche Kalibrationen verwalten
- Eigenschaften der Slaves verwalten
- ID- Verwaltung
- Funktionen zum Speichern und Wiederherstellen der Messumgebung

Variablen und Funktionen

Hier folgt eine Auflistung aller wichtigen Funktionen und Variablen die in dieser Klasse vorkommen:

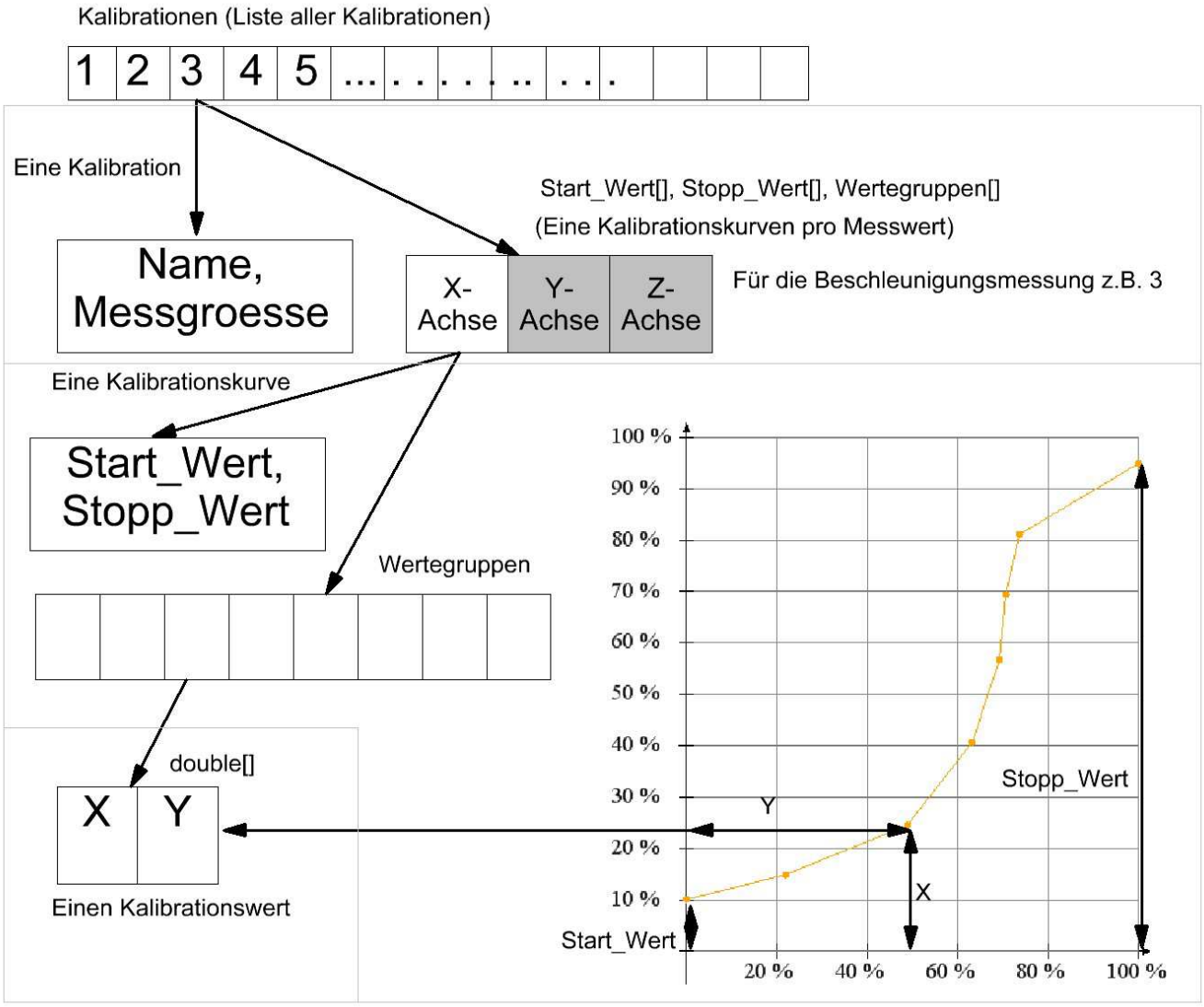
Dynamische Beschreibung der Messtypen:
Es wurde Versucht im Programm die Möglichkeit offen zu halten neue Messtypen einfach einzuführen. Daraus ergab sich folgendes System:



- `enum E_Messgroessen`
  - Liste der Sensoren, die auf den Slaves bestückt sein können
  - Die Reihung entspricht auch gleich dem Wert des Messdatenabfragebefehls. Genauere Informationen sind im Kapitel „K_Funkverwaltung“ zu finden
  - Die Liste kann im Nachhinein noch immer erweitert werden. Es wurde im gesamten Programm darauf geachtet, dass nachträglich die Sensortypen verändert werden können. **Achtung**: Bei Erweiterung der Liste müssen auch die Funktionen: „`Get_Messgroesse_Deffinition()`“ und „`Messdaten_Auswerten()`“ergänzt werden
- `int Messgroessen_Anzahl`
  - Gibt die Anzahl der eingetragenen Sensortypen zurück
  - Dieser Wert wird automatisch erstellt und während der gesamten Laufzeit nicht geändert

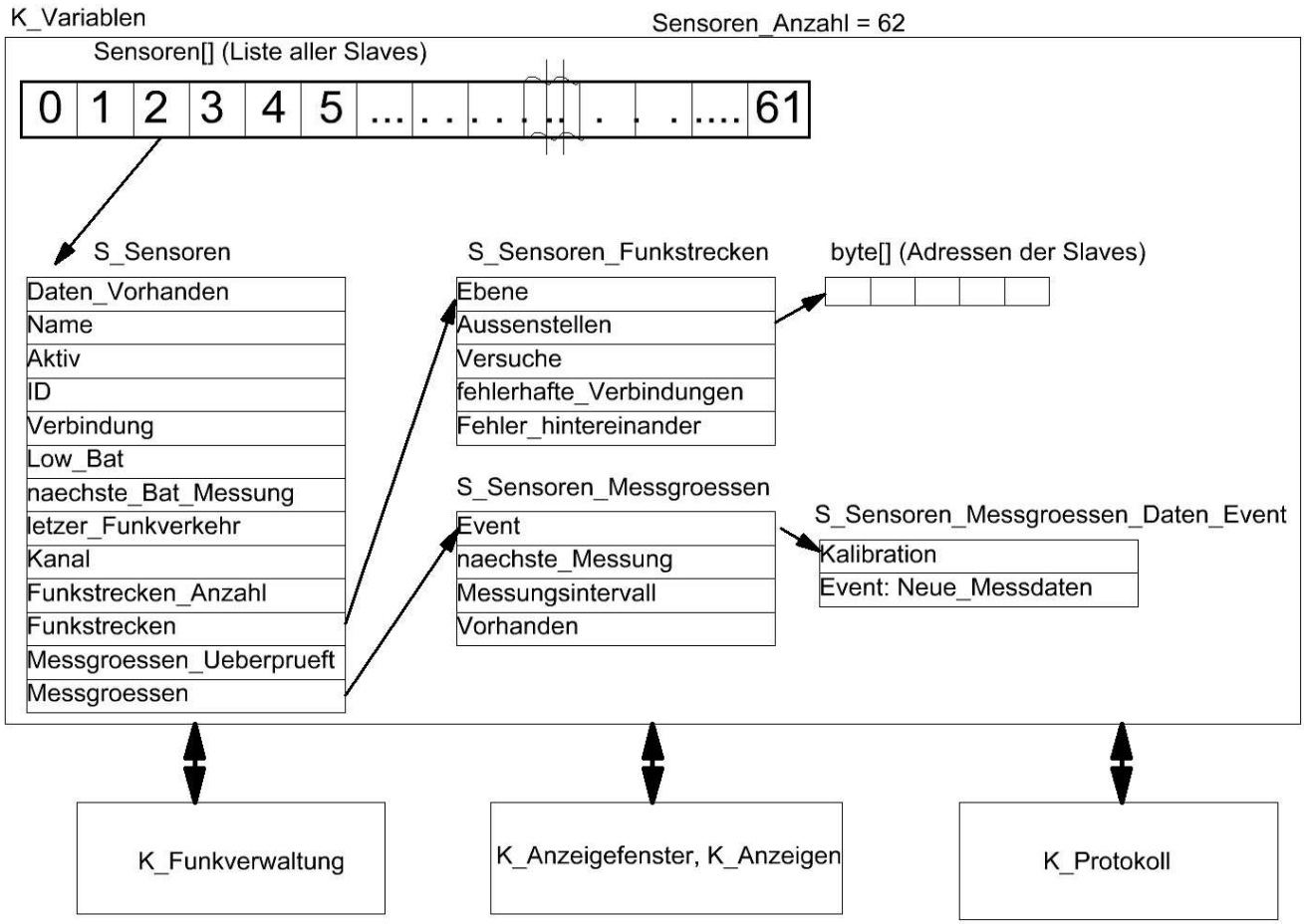
- `struct S_Messgroesse`
  - Zum Beschreiben eines Sensortyps
- Get_Messgroessen_Definition()
  - Hier wird mit Hilfe der Eigenschaft "`E_Messgroessen`„ die Eigenschaften (`S_Messgroesse`) für den betreffenden Sensortyp zurückgegeben
- Messdaten_Auswerten()
  - Wandelt die empfangenen Rohdaten nach einem Algorithmus in Messdaten um
  - Es wird ein „`double[]`" mit den Messwerten zurückgegeben

Zum Verwalten der Kalibrationen:
Hier folgt eine schematische Darstellung des Systems zum verwalten von Kalibrationen:



- `struct S_Kalibration`
  - Beschreibung einer Kalibration
- `ArrayList Kalibrationen`
  - Enthält alle eingegebenen Kalibrationen
  - Jedes Element dieser dynamischen Liste ist vom Typ „`S_Kalibration`"

Zum Verwalten der Eigenschaften der Slaves:



- `struct S_Sensoren_Messgroessen_Daten_Event`
  - Dieses Struct beschreibt einen Client der Messdaten weiterverarbeitet
- `struct S_Sensoren_Messgroessen`
  - Für jede Messgröße eines Slaves wird dieser Struct verwendet
- `struct S_Sensoren_Funkstrecken`
  - Zum Verwalten der einzelnen Funkstrecken
- `struct S_Sensoren`
  - Dieses Struct beschreibt einen Slave
- `const int Sensoren_Anzahl`
  - Anzahl der möglichen Slaves
  - Der Wert beträt derzeit 61
  - Durch eine nachträgliche Änderung des Funkprotokolls, kann hier die Anzahl der Slaves nachgebessert werden
- `S_Sensoren[] Sensoren`
  - Liste aller Slaves
  - Der Index entspricht der Adresse des Slaves

ID- Verwaltung:

- Get_Zufalls_ID()
  - Wird verwendet, um zufällig eine ID zu erzeugen
  - Diese ID ist mit einer sehr hohen Wahrscheinlichkeit einzigartig
- calc_ID()
  - Diese Funktion berechnet aus Rohdaten eine ID oder umgekehrt

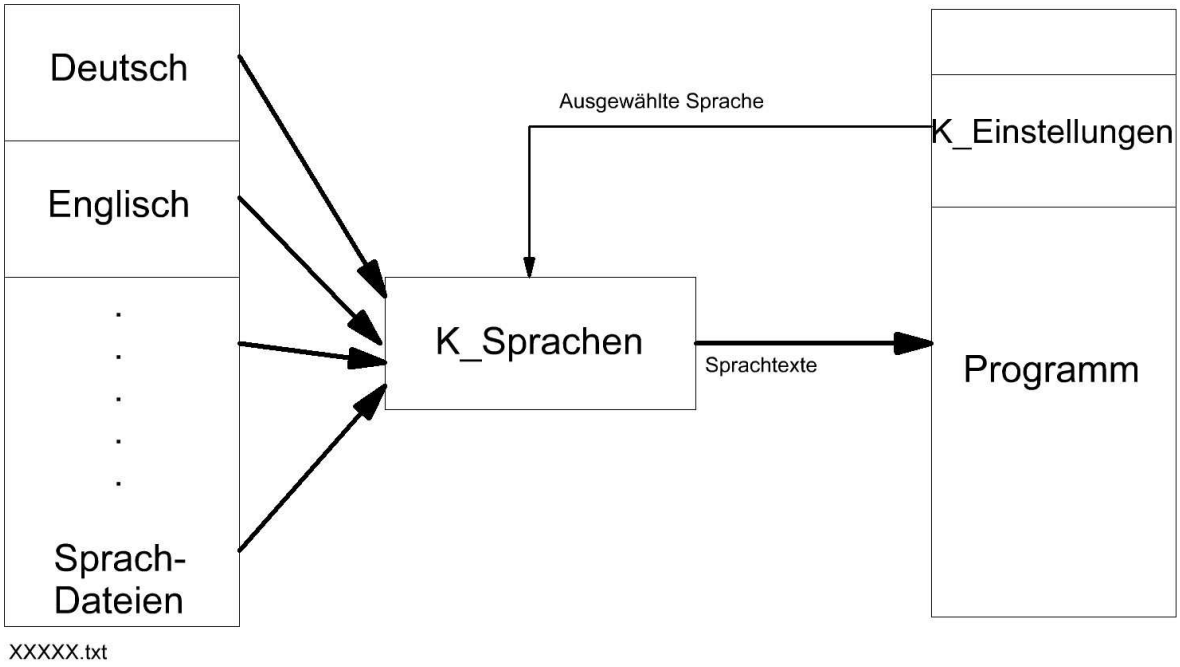
Funktionen zum Speichern und Wiederherstellen der Messumgebung:

- Messumgebung_speichern(), Messumgebung_wiederherstellen()
  - Diese Funktionen speichert die aktuell eingestellten Umgebungsvariablen, die normalerweise beim Schließen verloren gehen und stellen sie bei Nachfrage wieder her
  - Folgende Einstellungen sind betroffen:
    - Position des MainForms
    - Aktive Slaves
    - Erstellte Gruppen
    - Erstellte Anzeigefenster
    - Gruppenzugehörigkeit der Anzeigefenster
  - Die Funktion Messumgebung_speichern() wird von der Klasse K_MainForm aufgerufen bevor das Formular geschlossen wird
  - Die Funktioen Messumgebung_wiederherstellen() wird vom Hauptprogramm beim Starten der Anwendung aufgerufen

K_Sprachen

Mit Hilfe dieser Klasse ist es möglich die Sprache des Programms zu ändern. Jeder sprachabhängige Text wird von dieser Klasse aus einer Textdatei ausgelesen, somit ist man in der Lage bei Auswahl einer anderen Sprachdatei die Sprache zu wechseln.

Prinzipieller Aufbau



Aufbau einer Sprachdatei

Die Datei muss sich im selben Verzeichnis wie das Programm befinden. Die Sprachdatei ist eine einfache Textdatei im UTF-8 Format mit dem Namen „XXXXX.txt“. XXXXX steht für den Namen der Sprache.
Die Textdatei ist wie folgt aufgebaut:

Name_des_Eintrags “=“ Text_in_der_gewollten_Sprache

Es ist kein eigener Dateikopf erforderlich.
Jede Zeile entspricht einem Eintag. Zeilenumbrüche in einem Eintrag werden mit den String „\r\n“ realisiert.
Zeilen in dem kein = vorkommt und kein Text steht werden einfach ignoriert.
Zeilen mit gleichem Namen werden beim Zugriff durch eine Fehlermeldung beanstandet.
Wenn ein benötigter Name für einen Eintrag beim Abfragen nicht gefunden wird, wird der Standardtext „Fehler“ zurückgegeben.
Ein „*“ im Namen eines Eintrages wird als Leerzeichen interpretiert und kann zum Markieren von Einträgen verwendet werden.
Alle führenden und nachfolgenden Leerzeichen werden nicht berücksichtigt. Somit ist es möglich durch das Einrücken von Einträgen eine Struktur aufzubauen.
Auszug aus einer Sprachdatei:

```
MainFram_mIDatei = Datei
  MainFram_mIEinstellungen = Einstellungen
    MainMainFram_mISpeichern = Speichern
      MainFram_mISpeichern_Meldung = Die Einstellungen wurden erfolgreich
                                     gespeichert
    MainMainFram_mIImportieren = Importieren
```

```
MainMainFram_Importieren_Meldung = Durch den Import neuer Einstellungen
                                   werden die Aktuellen überschrieben \r\nDie Einstellungen
                                   werden erst beim Programmneustart wirksam!
MainMainFram_Importieren_Fehlermeldung = Importfehler!
MainMainFram_mIExportieren = Exportieren
MainFram_mIBeenden = Beenden
MainFram_mIAnzeigen = Anzeigefenster
```

Erstellen einer neuen Sprache

Man nimmt eine schon vorhandene Sprachdatei her und kopiert diese. Anschließend benennt man diese Datei nach der neuen Sprache.

Nun übersetzt man alle Einträge in der Datei. Aber dabei ist darauf zu achten, dass man nur Änderungen rechts vom „=“ Zeichen vornimmt.

Danach ist die Datei im Programmverzeichnis zu speichern.

Zuletzt muss man in der Klasse K_Variablen das Array „`SprachenVerfuegbar`“ um den Namen der neuen Sprache erweitern. Dabei ist darauf zu achten, dass die Datei denselben Namen trägt.

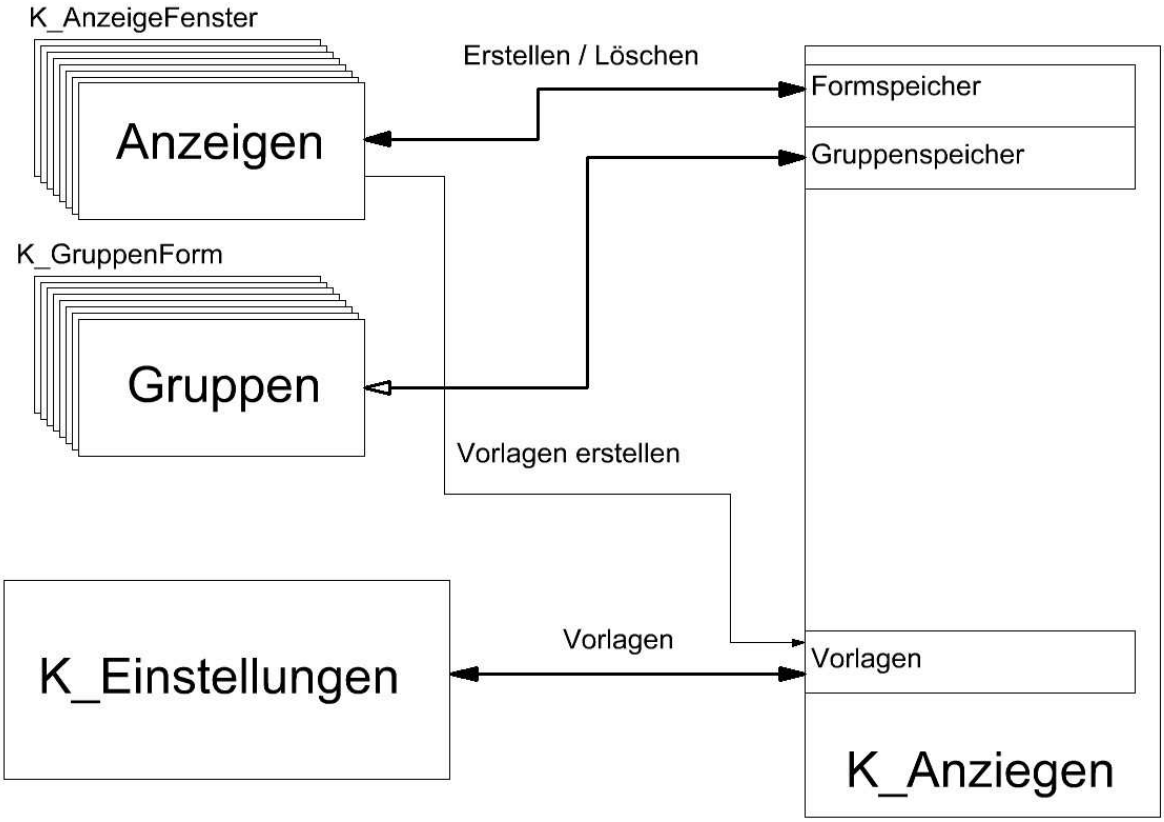
K_Anzeigen

Wird verwendet, um alle angezeigten Anzeigefenster und Gruppenfenster zu verwalten. Weiters werden hier auch die Vorlagen für die Anzeigefenster verwaltet.

Aufgaben

- Verwalten der angezeigten Anzeigefenster
- Verwalten der angezeigten Gruppenfenster zum Gruppieren
- Verwalten von Vorlagen
- Erzeugen von neuen Anzeigefenstern und Gruppenfenstern
- Sicheres Schließen von Anzeigefenstern und Gruppenfenstern

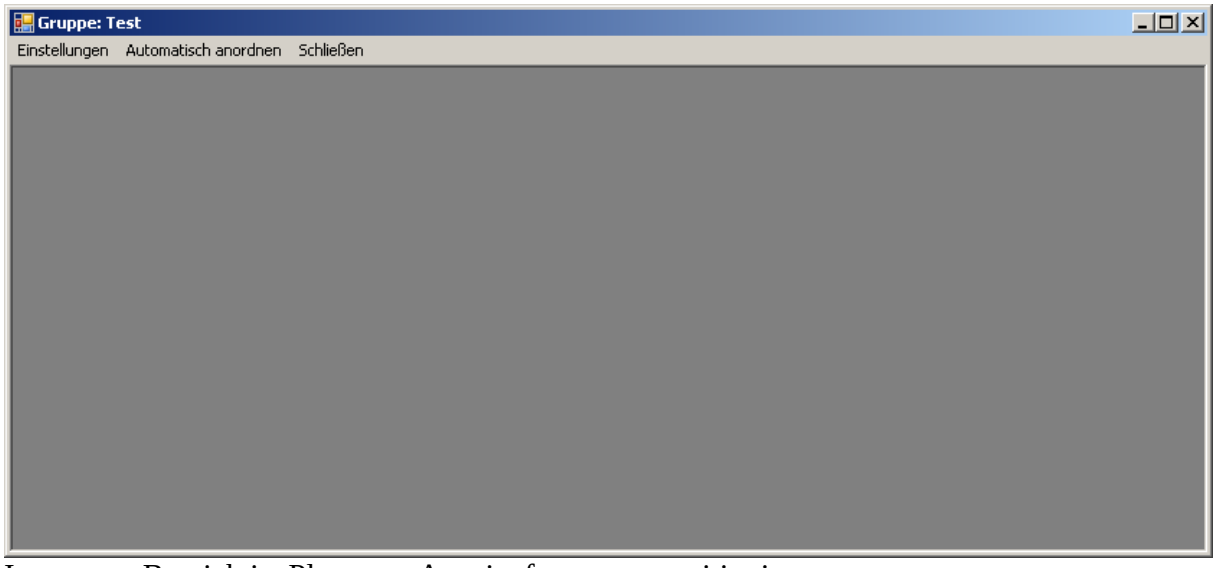
Prinzipieller Aufbau



K_GruppenForm

Diese Klasse beschreibt ein Gruppenfenster. Dieses Fenster hat nur die Aufgabe mehrere Anzeigefenster zu einer Gruppe zusammenzufügen.
Achtung: Diese Klasse wird im Programm mehrfach initialisiert.

Aussehen eines Gruppenfensters



Im grauen Bereich ist Platz, um Anzeigefenster zu positionieren.

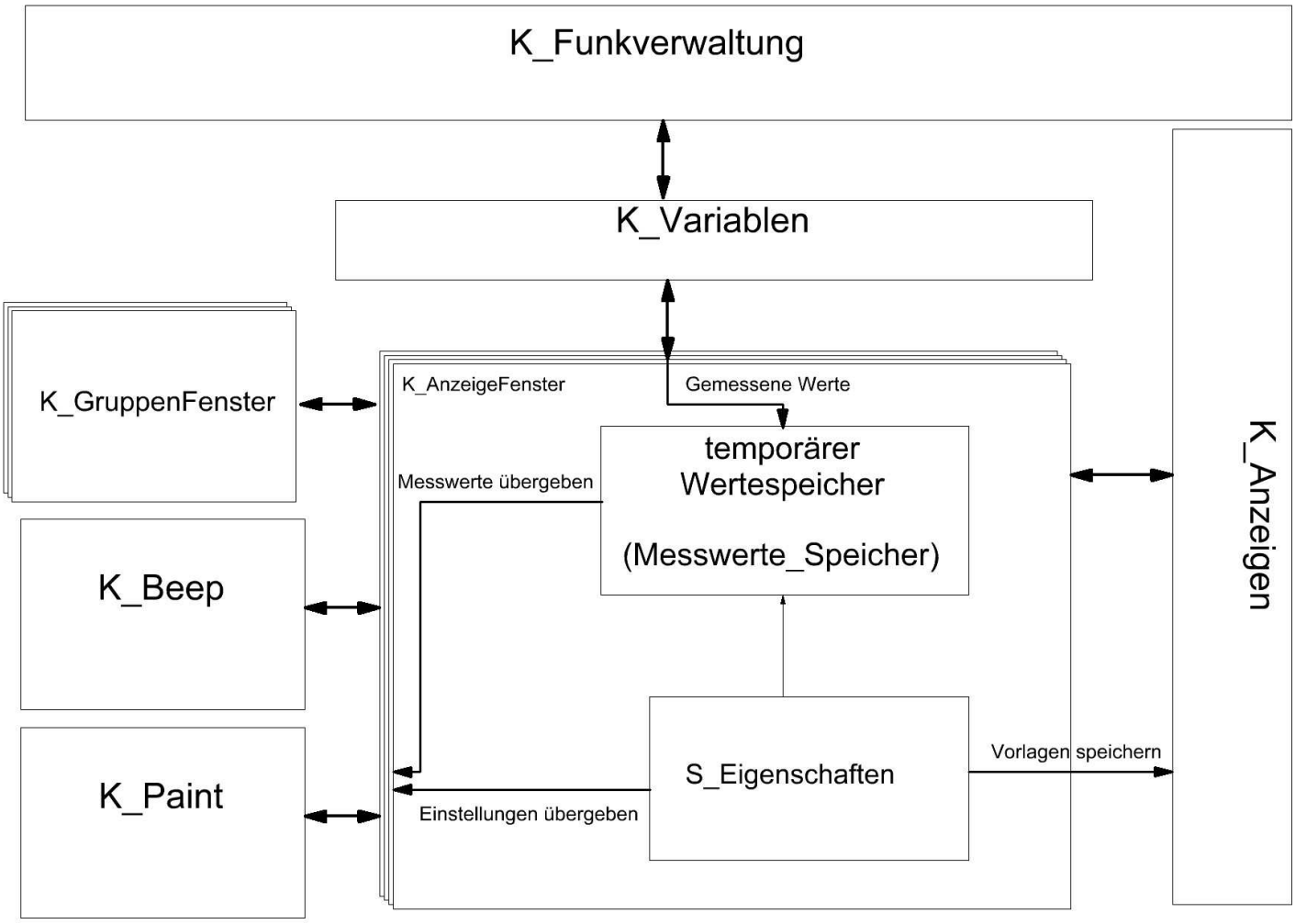
K_AnzeigeFenster

Diese Klasse repräsentiert ein Anzeigefenster.
Achtung: Diese Klasse wird im Programm mehrfach initialisiert.
Mit Hilfe der Klasse K_Paint werden hier die gemessenen Messwerte optisch dargestellt und viele andere Einstellungen sind noch möglich.

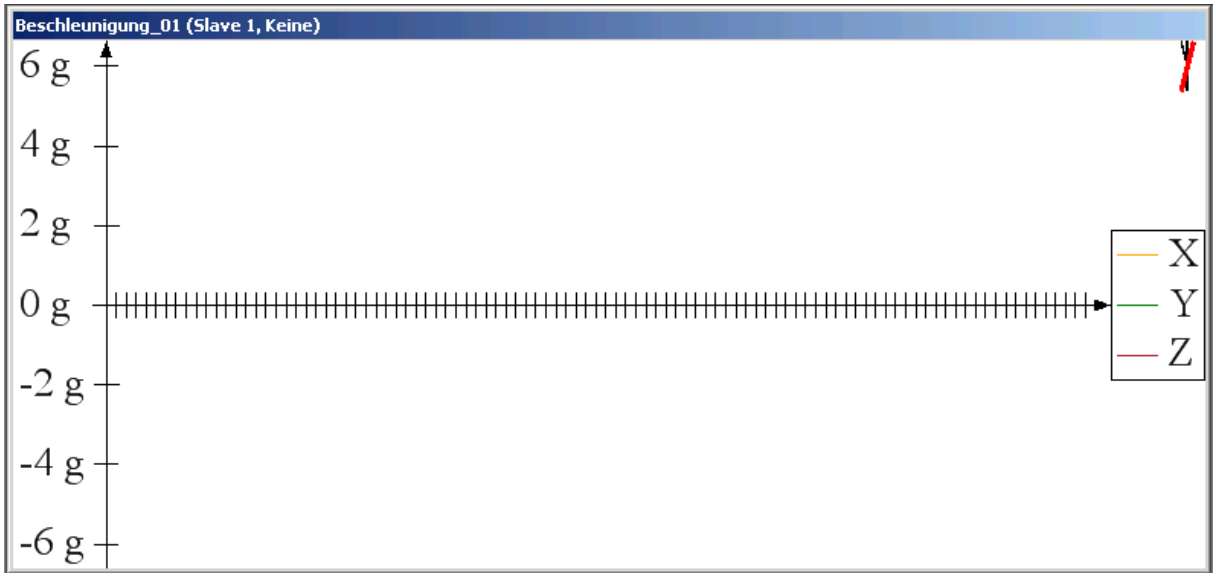
Aufgaben

- An das Variablen- System der Slaves in der Klasse K_Variablen anhängen
- Messwerte für Auswertungen temporär zwischenspeichern
- Tonausgabe über die Klasse K_Beep
- Optische Anzeige mit Hilfe der Klasse K_Paint erstellen
- Verwalten der Einstellungen und Erstellen von Vorlagen

Prinzipieller Aufbau



Aussehen



Kontextmenü

Durch einen Rechtskick auf die Zeichenfläche wird ein Menü geöffnet, das folgende Punkte enthält:

- Eigenschaften_ (`mIEigenschaften`)
  - Zum Ändern der Einstellungen
- Außenstellen_ (`mIAussenstelle`)
  - Zum Ändern der Einstellungen, nur wird hier speziell auf die Registrierkarte Slaves geschalten
- Speichern_ (`mISpeichern`)
  - Zum Erstellen einer Vorlage
- Beenden_ (`mIBeenden`)
  - Zum Schließen der Anzeige

K_Paint

Mit Hilfe dieser Klasse ist es möglich diverse Zeichnungen zu erstellen und dadurch können spezielle Diagramme für die Darstellung von Messwerten erstellt werden.
Achtung:

Diese Klasse wird im Programm mehrfach initialisiert.
Diese Klasse enthält auch einen eigenen Thread zum Aktualisieren der Zeichnung.

Diese Klasse hält das OOP- Konzept ein.

Aufgaben

- Optische Darstellung von Messdaten mit Hilfe einer Zeichenfläche
- Automatisches Aktualisieren der Zeichnung
- Anzeige für Statusmeldungen
- Optische Darstellung von Kalibrationen
- Manuelle Bearbeitung von Kalibrationen
- Optische Effekte für Über- oder Unterschreiten von Grenzwerten

K_Beep

Mit Hilfe dieser Klasse wird es ermöglicht Töne auszugeben. Da das gesamte Programm sehr asynchron aufgebaut ist, wird es hier ermöglicht dennoch einen periodischen Ton auszugeben.

Achtung: Die Klasse besitzt einen Thread und wird während der Laufzeit mehrfach initialisiert.

Statische Variablen

Um die Funktion dieser Klasse genauer zu verstehen, ist es notwenig die Funktion von statischen Variablen zu verstehen.
Statische Elemente sind ohne eine realisierte Instanz einer Klasse verfügbar. Man kann sie während der Laufzeit einfach über den Namen der Klasse ansprechen.
Folgender Quellcode zeigt ein Beispiel:

```
K_Beep.Alarm_count++;
```

Statische Variablen werden beim Start des Programms initialisiert und werden erst wieder nach dem Schließen gelöscht.

Somit ist es möglich das verschiedene Instanzen einer Klasse auf ein und dieselbe Information innerhalb der Klasse zu greifen können.
Dieser Effekt wird bei der Klasse K_Beep ausgenutzt, um die aktiven Klassen und die aktiven Alarmmeldungen global zu zählen.

K_Funkverwaltung

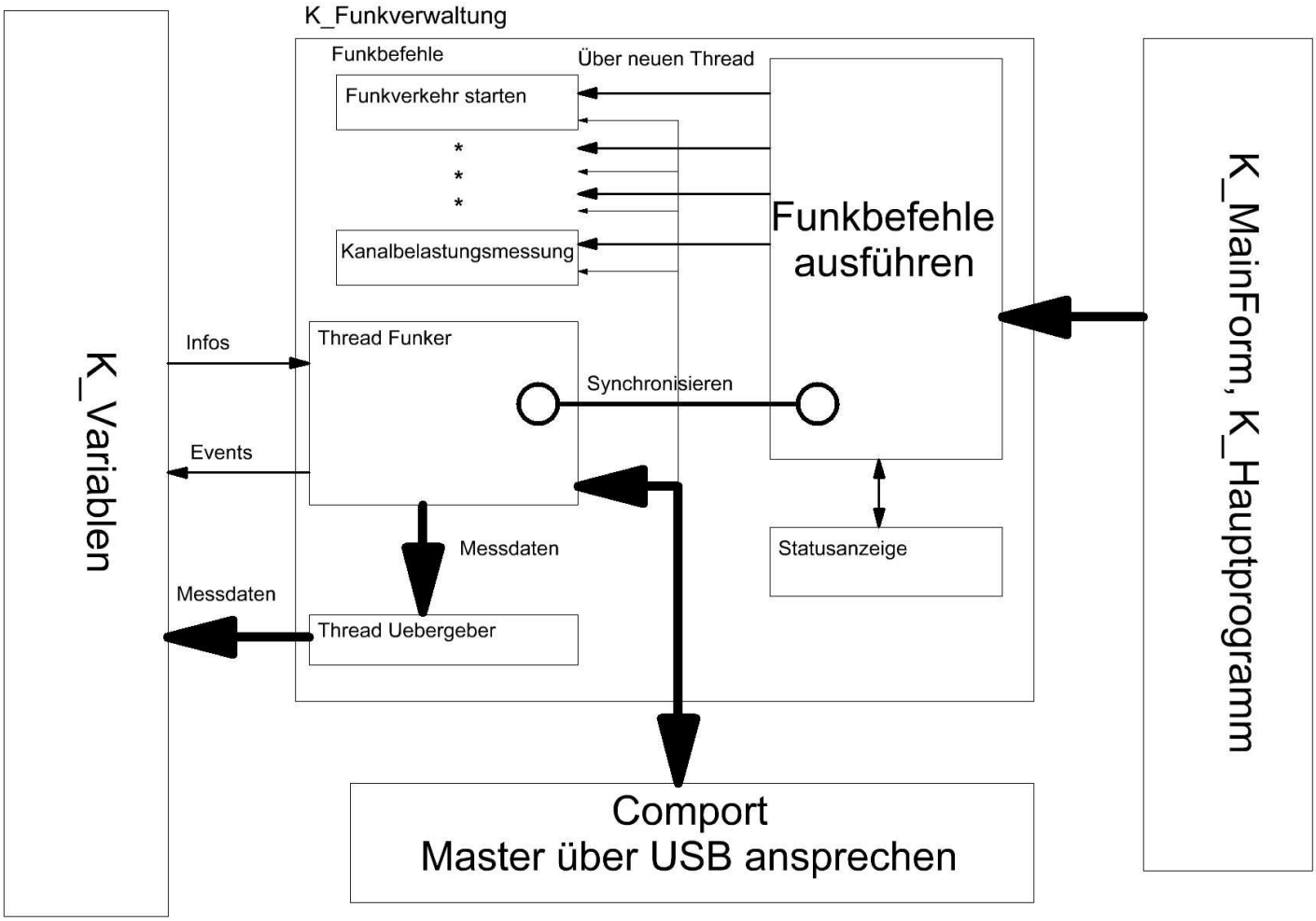
Diese Klasse besteht aus zwei Teildateien: „Funkverwaltung.cs“ und „Funkverwaltung.Funkverkehr.cs“.
Das File „Funkverwaltung.cs“ übernimmt die Synchronisationsaufgaben und die Weitergabe der empfangenen Daten.
Das File „Funkverwaltung.Funkverkehr.cs“ hingegen beschäftigt sich mit der Realtime-Kommunikation.

Aufgabenaufteilung

Hier wird genau angegeben, welche Aufgaben die einzelnen Dateien übernehmen.

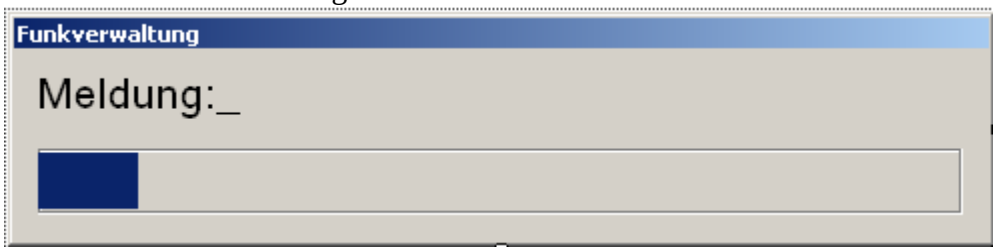
- Funkverwaltung.cs
  - Verwaltung der Statusanzeige
  - Thread zum Sammeln der Messdaten verwalten
  - Thread zum Weitergeben der Messdaten verwalten
  - Threads synchronisieren
  - Verwalten der Comport- Ressource
  - Funkbefehle einleiten
  - Funkverkehr starten und stoppen
  - Funkkanalbelastungsmessung durchführen
  - Einfache Funkalgorithmen ausführen
- Funkverwaltung.Funkverkehr.cs
  - Einen Funkbefehl durchführen
  - Messdatenabfrage durchführen
  - Algorithmus zum Suchen der Slaves
  - Algorithmus zum Kanalwechseln
  - Masterbefehle durchführen
  - Mögliche Simulation des Masters und der Slaves

Prinzipieller Aufbau



Statusanzeige

Aussehen der Statusanzeige:



In Wirklichkeit ist die Klasse K_Funkverwaltung eine Ableitung der Formklasse. Somit ist die Statusanzeige quasi das Formular der Klasse.

K_Port_List

Ist eine Verbindungsklasse zu einer Systemressource zum Abfragen der verfügbaren Ports. Mit Hilfe dieser Ports wird nach dem Master gesucht.

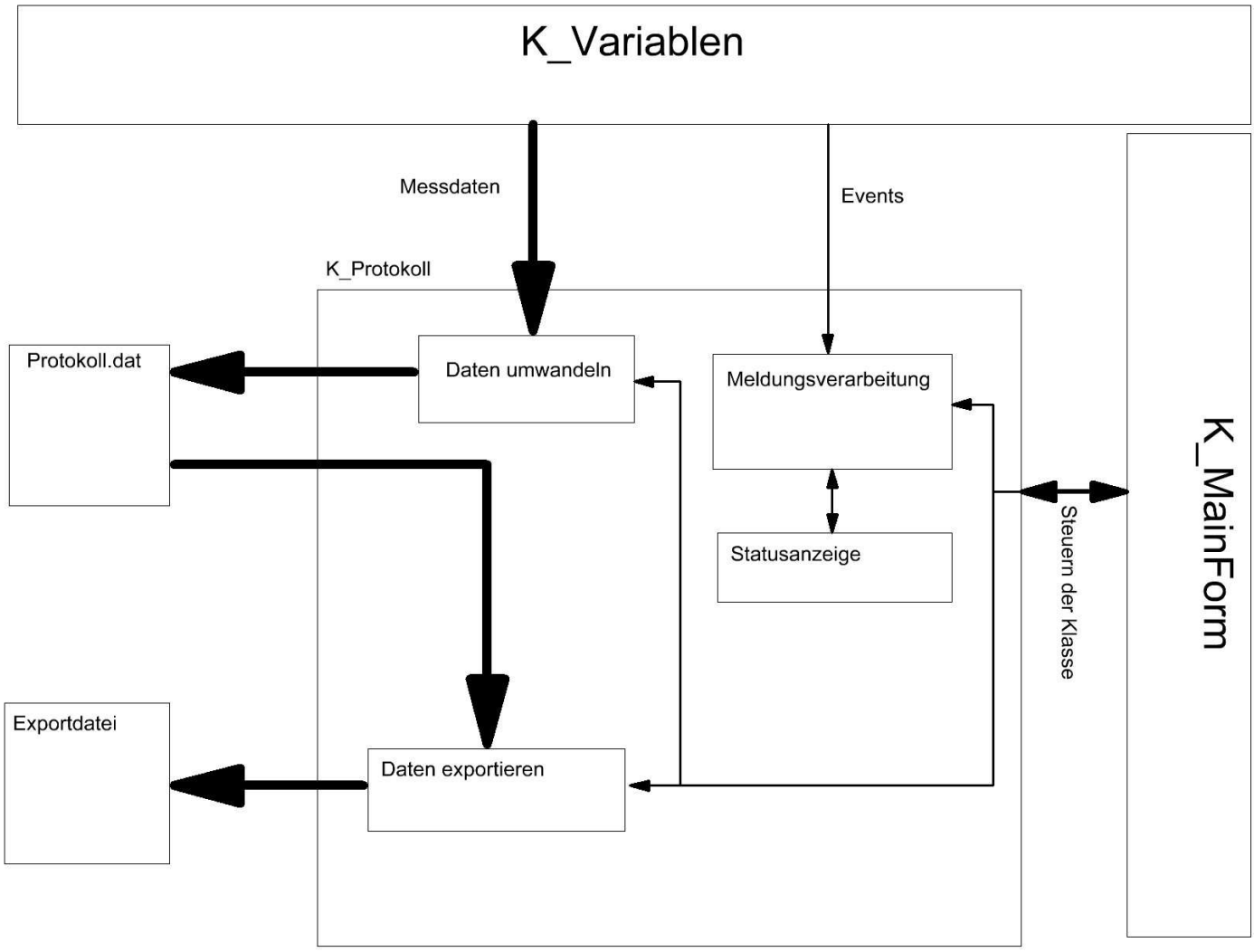
Diese Klasse besteht aus mehreren statischen Funktionen zum Erzeugen einer Liste von möglichen Comport- Ressourcen. Es wird mit Hilfe der API (application programming interface) Klasse von Microsoft eine Liste von Comports abgerufen und verarbeitet.

Die Funktion „`Get_Port_List()`“ liefert ein „`int[]`“ mit möglichen Comports zurück.

K_Protokoll

Diese Klasse beinhaltet ein sehr einfaches Protokollsystem. Damit ist es möglich die gemessenen Daten temporär zu speichern und später in einem gewünschten Format zu exportieren.

Prinzipieller Aufbau



Aussehen des Statusfensters

In dem weißen Bereich wird mit Hilfe der Klasse K_Paint der Status aller beteiligten Slaves angezeigt.

5.1.1.3. Bedienungsanleitung

Seite

Installation

Anforderungen

	Mindestens	Empfohlen
Betriebssystem:	Windows NT	Windows XP
Arbeitsspeicher:	256MB	512MB
Prozessor:	Pentium 4 mit 1GHz	Pentium 4 mit 2GHz
Festplattenspeicher:	10MB	

Zu installierende Treiber und Applikationen

Microsoft.NET Framework 2.0

Im Verzeichnis „PC Software\ Setup“ auf der DVD ist das Setup für die Microsoft.NET Framework 2.0 mit dem Namen „dotnetfx.exe“ zu finden.

FTDI- Treiber

Download: http://www.ftdichip.com/Drivers/CDM/CDM%202.02.04%20WHQL%20Certified.zip
Dieser Treiber wird während der Laufzeit beim erstmaligen Anschließen des Masters verlangt.

IMS- Software

Im Verzeichnis „PC Software\ Setup“ auf der DVD ist das Setup zu finden.

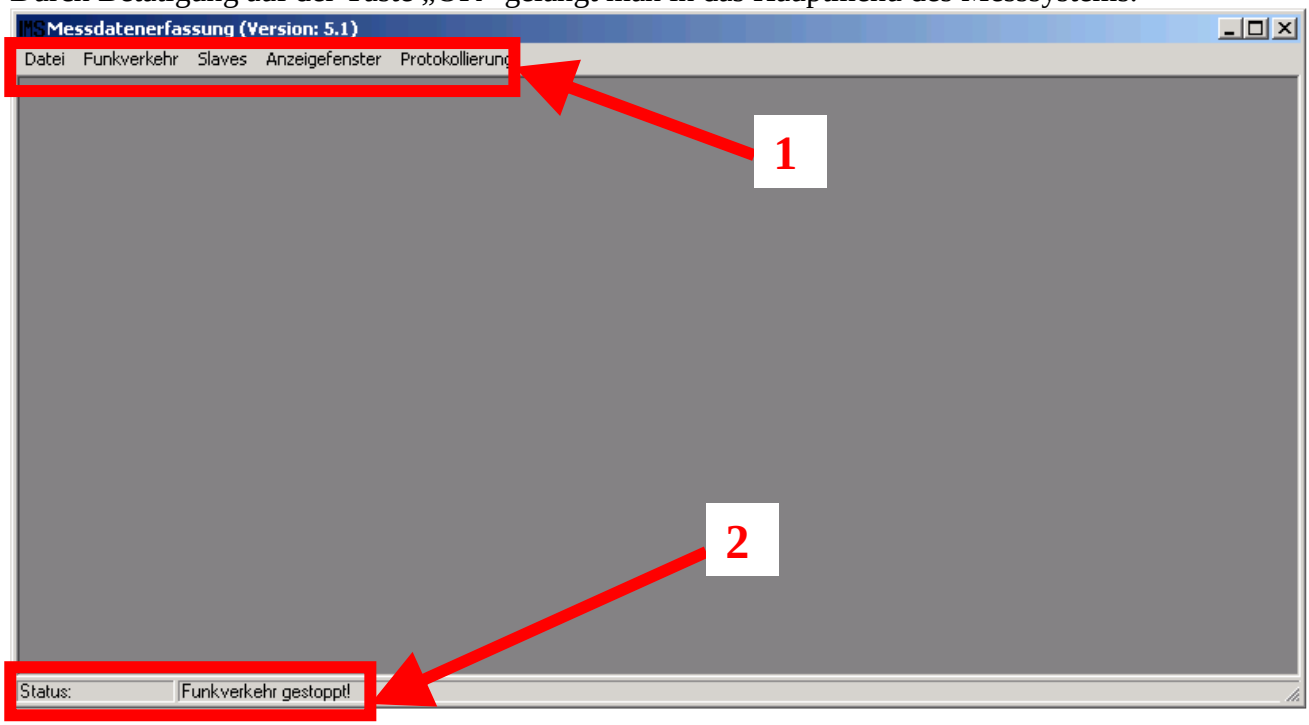
Bedienung

Bevor das Programm gestartet wird, sollte der Master an einen USB- Anschluss angeschlossen werden!

Erster Programmaufruf

Beim erstmaligen Aufrufen der Programmsoftware erscheint folgendes Fenster zum Auswählen der gewünschten Sprache:

Der rote Pfeil markiert die Position des Fensters, an der die gewünschte Sprache mittels Drop-Down- Menü ausgewählt werden kann.
Nähere Informationen zum Hinzufügen einer neuen Sprache finden Sie im Kapitel „PC-Software/ K_Sprachen“.
Durch Betätigung auf der Taste „OK“ gelangt man in das Hauptmenü des Messsystems.



Der Pfeil mit der Nummer 1 zeigt auf die verfügbaren Menüpunkte des Programms:

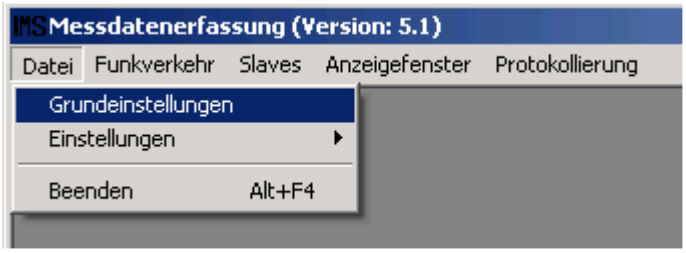
- Datei
  Generelle Einstellungen des Programms
- Funkverkehr
  Verwaltung des Funkverkehrs
- Slaves
  Verwaltung der Slaves und verwalten der Kalibrationdateien
- Anzeigefenster
  Verwaltung der Anzeigefenster
- Protokollierung
  Verwaltung des Protokollsystems

Auf die Menüpunkte wird in den nächsten Unterpunkten der Dokumentation näher eingegangen. Der Pfeil mit der Nummer 2 zeigt auf die Statusanzeige. Sie zeigt immer die aktuellsten Einstellungen und Vorgänge des Messsystems im Schnelldurchlauf. Die Anzeige wird periodisch wiederholt.

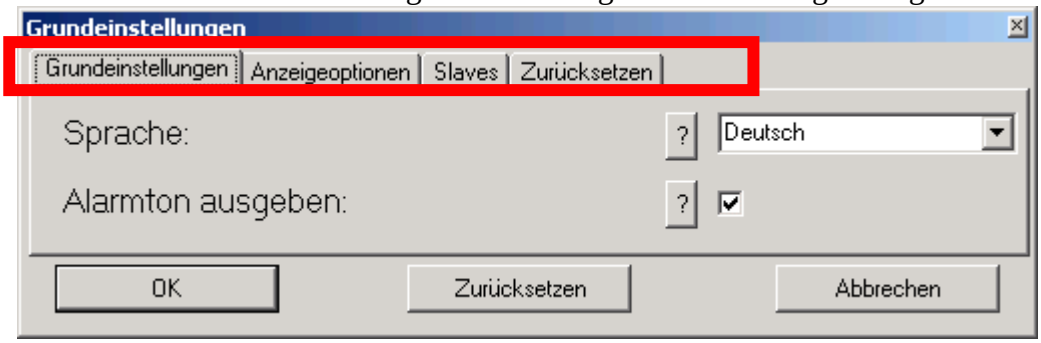
Datei

Unter dem Menüpunkt Datei können die Grundeinstellungen vorgenommen werden, Einstellungen importiert, exportiert oder gespeichert werden und unter den Punkt Beenden kann die Applikation beendet werden.

Grundeinstellungen

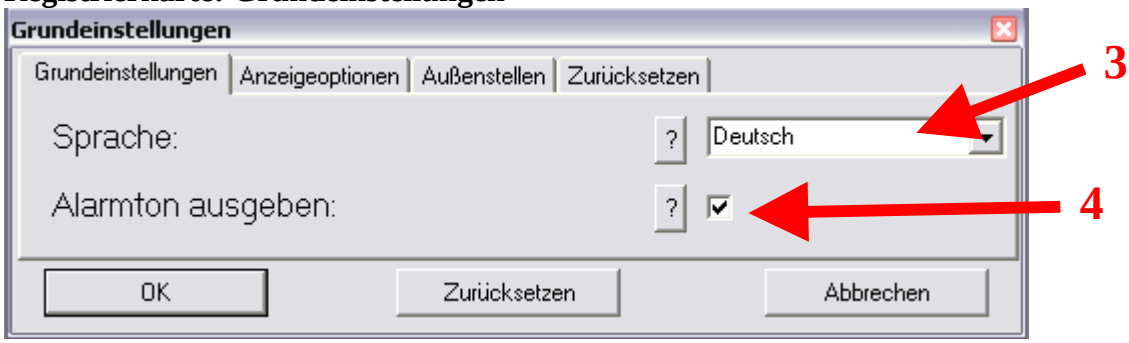


Unter Datei →Grundeinstellungen können folgende Einstellungen vorgenommen werden:



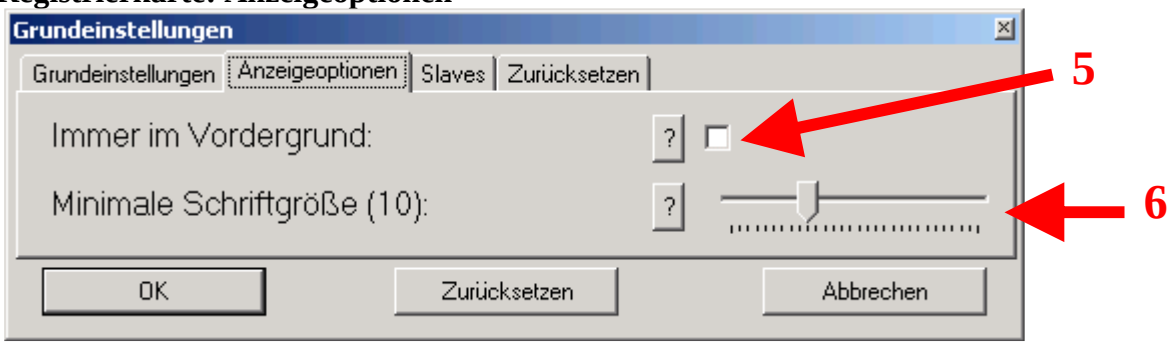
Je nach Auswahl der Registrierkarte (rot markiert) können die generellen Grundeinstellungen, Anzeigeoptionen oder Slaveseinstellungen vorgenommen werden. Durch Auswahl der Registrierkarte Zurücksetzen können sämtliche Einstellungen auf den Urzustand zurückgesetzt werden. Durch das drücken des Buttons „Zurücksetzen“ werden jedoch nur die aktuell eingegebene Werte auf Ihren vorigen Zustand zurückgesetzt.

Registrierkarte: Grundeinstellungen



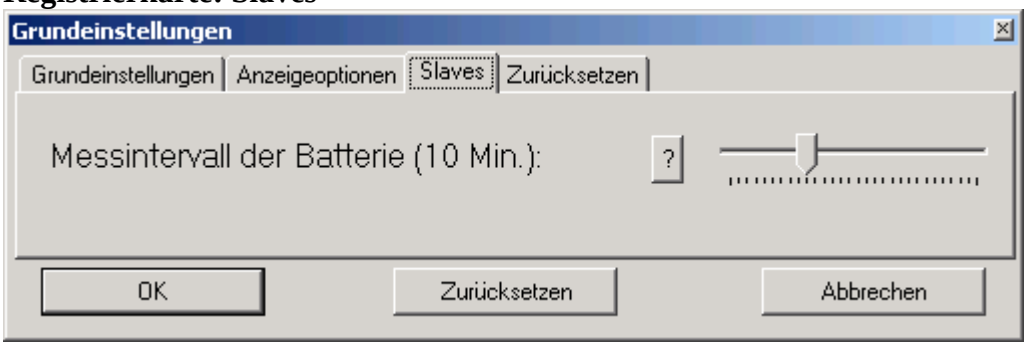
Die Registrierkarte Grundeinstellungen ermöglicht die Auswahl der gewünschten Sprache (mit dem Pfeil 3 gekennzeichnet). Weiters kann hier auch eingestellt werden, ob beim Über- bzw. Unterschreiten der definierten Werte ein Ton ausgegeben wird oder nicht (gekennzeichnet durch Pfeil 4).

Registrierkarte: Anzeigeoptionen



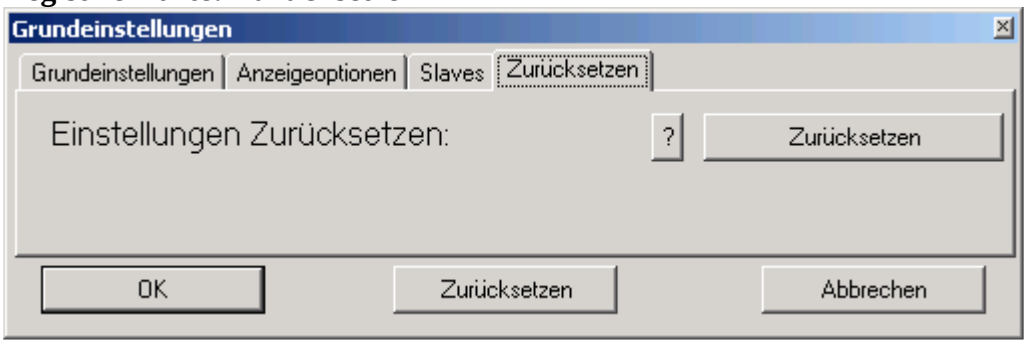
Die Registrierkarte Anzeigeoptionen ermöglicht die Einstellung einer minimalen Schriftgröße der Anzeigefenster, unabhängig von deren Größe, (mit Pfeil 6 gekennzeichnet) und die Einstellung, ob das Messprogramm sich immer im Vordergrund befinden soll (mit Pfeil 5 gekennzeichnet). Anklicken der Checkbox aktiviert diese Option.

Registrierkarte: Slaves



Die Registrierkarte Slaves ermöglicht eine generelle Einstellung des Abfrageintervalls für den Batteriemessstand der Slaves.

Registrierkarte: Zurücksetzen



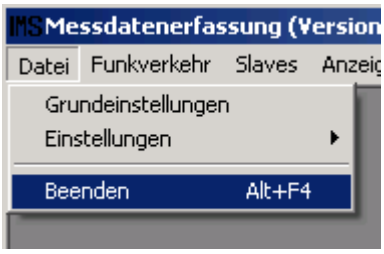
Der Punkt Zurücksetzen ermöglicht dem Benutzer das Zurücksetzen des Programms auf die Werkseinstellungen.

Einstellungen

Unter Datei →Einstellungen können die aktuellen Einstellungen gespeichert (mit Pfeil 7 gekennzeichnet) werden oder es können andere Einstellungsdateien importiert werden (mit Pfeil 8

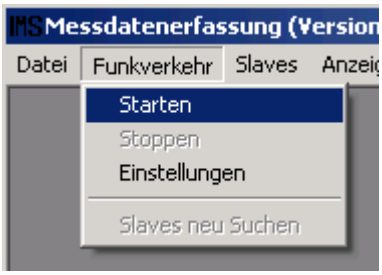
gekennzeichnet) oder die eigenen Einstellungen können in eine externe Datei exportiert werden, um sie in einem anderen Messsystem zu verwenden (mit Pfeil 9 gekennzeichnet).
Die aktuellen Einstellungen liegen im Programmordner.

Beenden



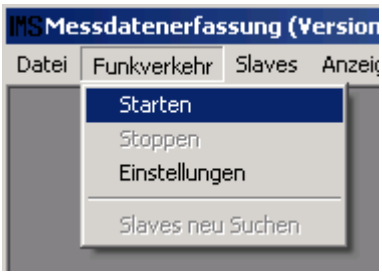
Unter Datei →Beenden kann das Messsystem beendet werden.

Funkverkehr



Der Menüpunkt Funkverkehr ermöglicht die Aktivierung bzw. Deaktivierung des Funkverkehrs. Weiters können hier diverse Einstellungen dem Funkverkehr betreffend vorgenommen werden und die Slaves können gesucht werden.

Starten

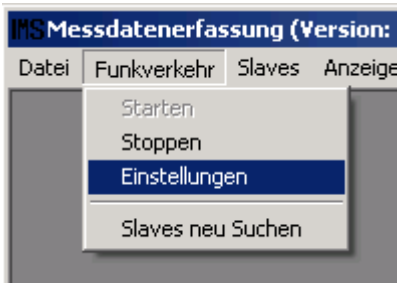


Unter Funkverkehr →Starten kann der Funkverkehr gestartet werden.
Dies ist notwendig wenn der Master während des Betriebes entfernt wurde oder er beim Starten des Programms nicht eingesteckt war. Weiters muss dieser Menüpunkt aufgerufen werden, sobald der Funkverkehr manuell gestoppt wurde und man wieder messen will.
Normalerweise wird dieser Punkt automatisch bei Starten des Programms ausgeführt.

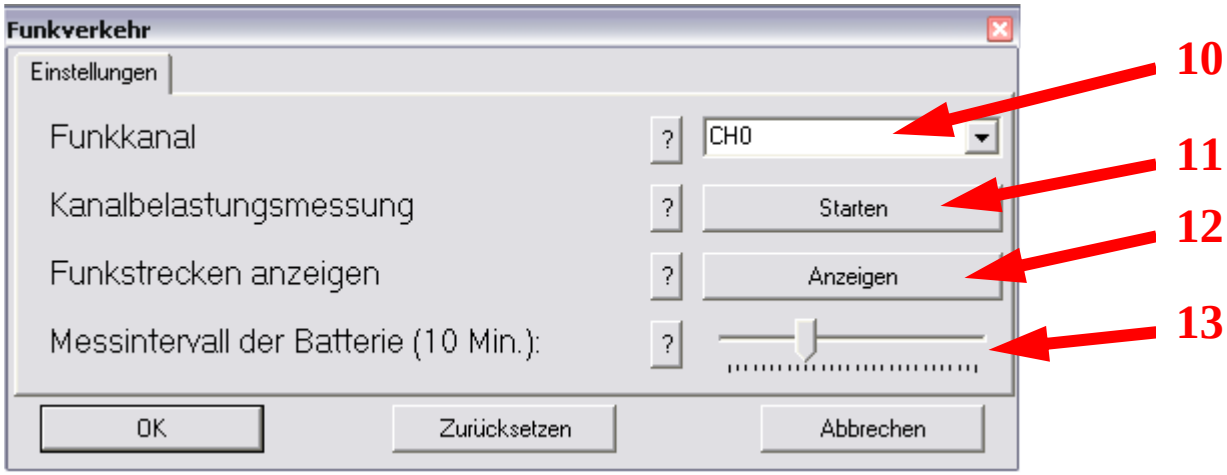
Stoppen

Unter Funkverkehr →Stoppen kann der Funkverkehr jederzeit unterbrochen werden.

Einstellungen



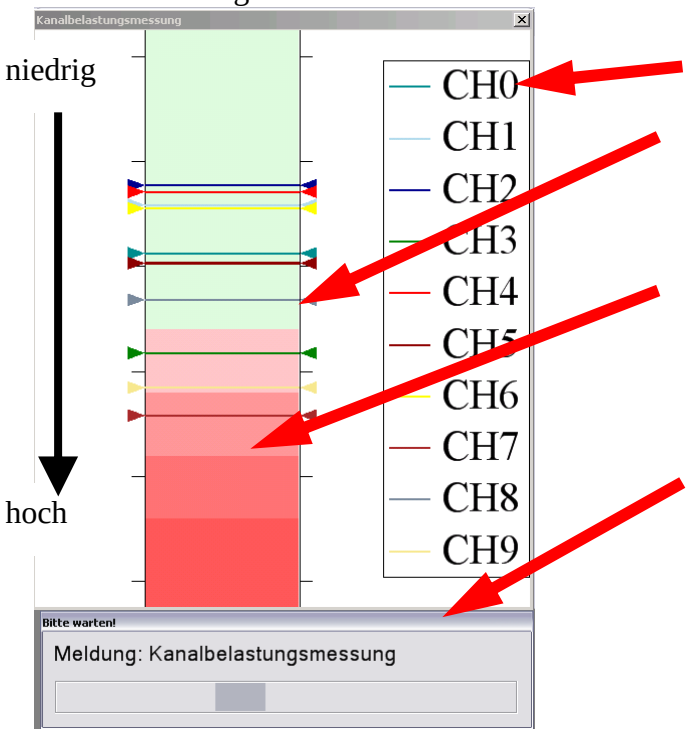
Unter Funkverkehr →Einstellungen können die Grundeinstellungen für den Funkverkehr vorgenommen werden. Weiters kann hier eine Kanalbelastungsmessung durchgeführt werden.



Pfeil 10: Funkkanal

Hier kann der gewünschte Funkkanal ausgewählt werden, auf dem das Messsystem sendet. Es sind die Kanäle von 0 bis 9 verfügbar.

Pfeil 11: Kanalbelastungsmessung
Durch betätigen des Buttons „Starten“ wird die Kanalbelastungsmessung gestartet.
Es erscheint folgendes Fenster:



Hier befindet sich die Legende der Kanäle

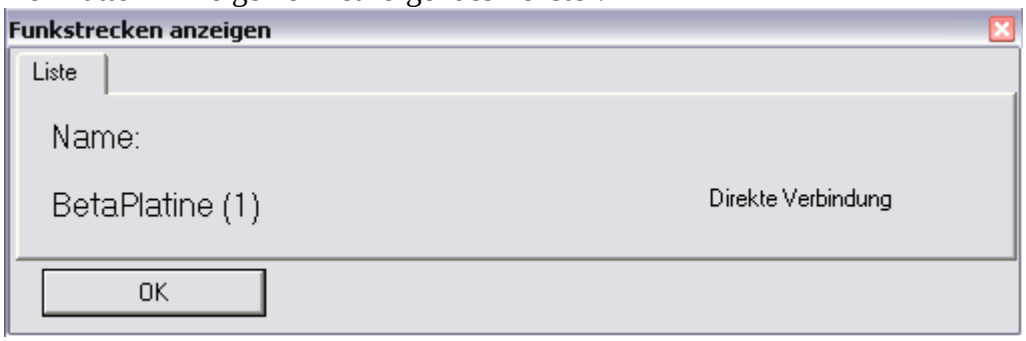
Der Belastungspegel des jeweiligen Kanals wird durch einen solchen Strich angezeigt

Der rote Bereich zeigt an, dass hier die Kanalbelastung besonders hoch ist. Diese(r) Kana(e)l(e) sollte(n) zum Senden nicht benutzt werden

Beim Aufrufen der Kanalbelastungsmessung erscheint auch dieses Fenster. Es zeigt an das immer noch eine Messung stattfindet

Die Kanalbelastungsmessung nimmt mehrere Sekunden in Anspruch bis sich die Werte eingependelt haben. Zum Beenden der Messung muss man das „X“ des Kanalbelastungsmessungsfensters betätigen.

Pfeil 12: Funkstrecken anzeigen
Der Button Anzeigen öffnet folgendes Fenster:



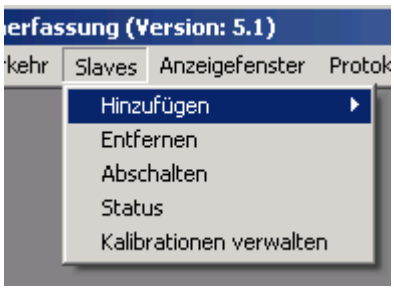
Hier wird angezeigt, wie die einzelnen Slaves mit dem Master verbunden sind.

Pfeil 13: Messintervall der Batterie
Diese Einstellung ermöglicht eine generelle Einstellung des Abfrageintervalls für den Batteriemessstand der Slaves. Dieser Menüpunkt hat denselben Effekt wie der Punkt unter Datei →Grundeinstellungen → Slaves.

Slaves neu Suchen

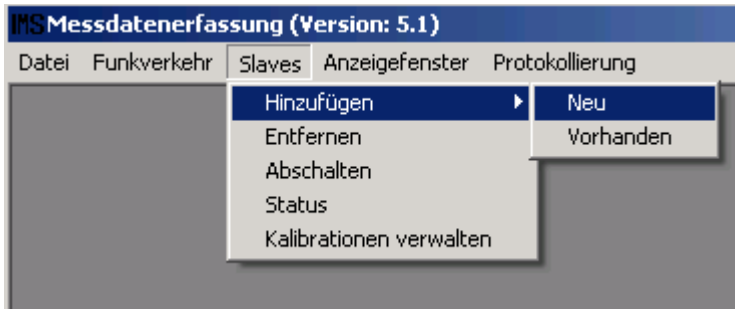
Unter Funkverkehr →Slaves neu Suchen wird versucht alle aktiven Slaves zu erreichen. Dieser Punkt wird benötigt sobald der Funkverkehr unterbrochen wurde oder neue Slaves in das Messsystem integriert werden.

Slaves



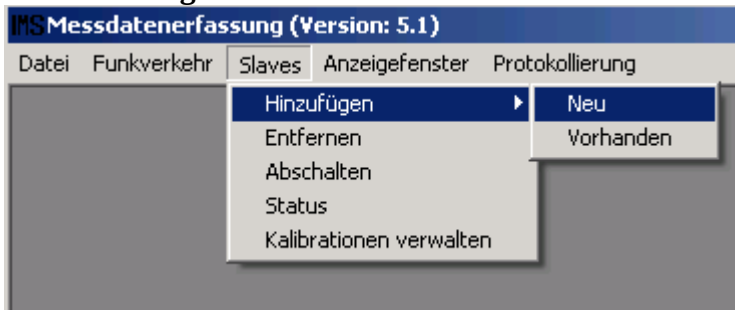
Der Menüpunkt Slaves erlaubt den Benutzer die Verwaltung der Slaves. Hier können Slaves hinzugefügt, entfernt oder per Funk abgeschaltet werden. Weiters kann hier der Status der einzelnen Slaves angesehen werden und die Verwaltung der Kalibrationen erfolgt ebenfalls hier.

Hinzufügen

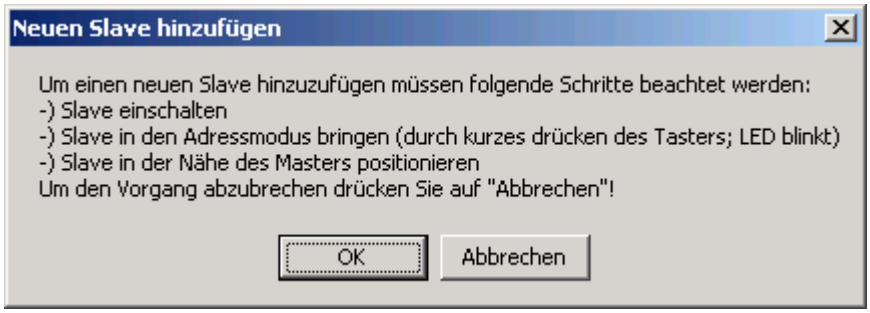


Unter Slaves →Hinzufügen können neue Slaves in das System integriert werden beziehungsweise bereits vorhandene Slaves wieder ausgewählt und für neuerliche Messungen verwendet werden.

Neu Hinzufügen



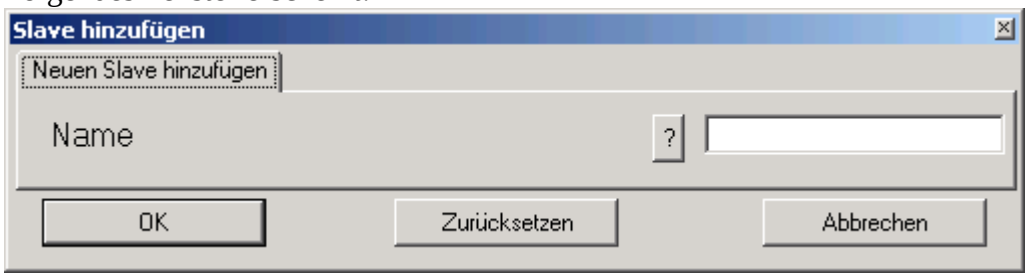
Slaves →Hinzufügen →Neu erlaubt den Benutzer noch nie verwendete Slaves in das Messsystem zu integrieren. Bei der Auswahl dieses Menüpunktes erscheint folgendes Fenster:



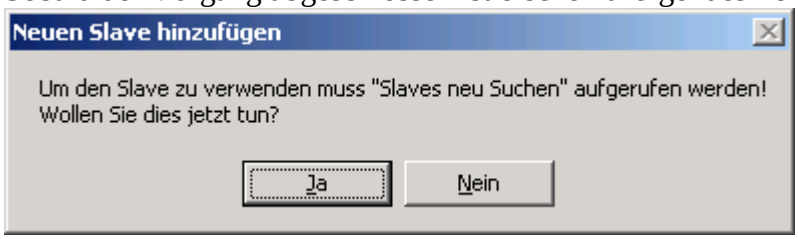
Das erste Fenster beschreibt wie man einen neuen Slave hinzufügt. Das zweite Fenster zeigt an was momentan gerade passiert.
Um eine neue Slave hinzufügen zu können müssen folgende Punkte erledigt werden:

- Den neue Slave einschalten
- Den Slave in den Adressiermodus bringen (den Taster kurz betätigen, die LED beginnt zu blinken)
- Den Slave in der unmittelbaren Umgebung des Masters positionieren

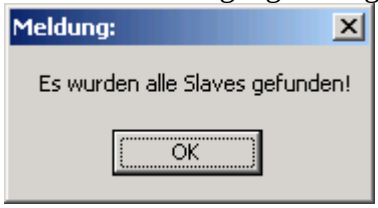
Sobald diese drei Punkte durchgeführt wurden, ist der Button „OK“ zu betätigen.
Folgendes Fenster erscheint:



Hier kann man den gewünschten Namen des Slaves in das Textfeld (grau hinterlegt) eingeben.
Der Name muss einzigartig sein.
Danach wird versucht den Slave zu erreichen dies kann einige Sekunden in Anspruch nehmen.
Sobald der Vorgang abgeschlossen ist erscheint folgendes Fenster:

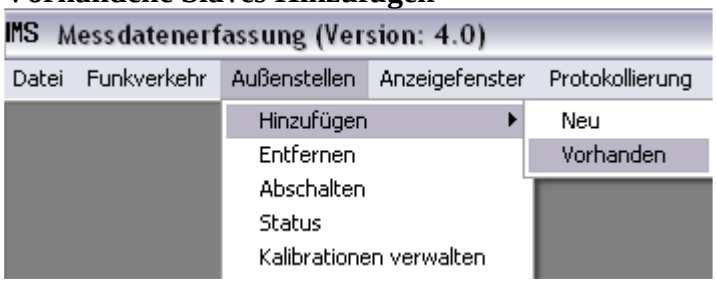


Ein Klick auf den Button „Ja“ bewirkt eine Kontrolle, ob der Slave tatsächlich erreichbar ist, wenn man auf Nein klickt, muss der Vorgang später manuell unter Funkverkehr →Slaves neu Suchen vorgenommen werden.
Wurden alle Vorgänge erfolgreich abgeschlossen erscheint folgendes Fenster:

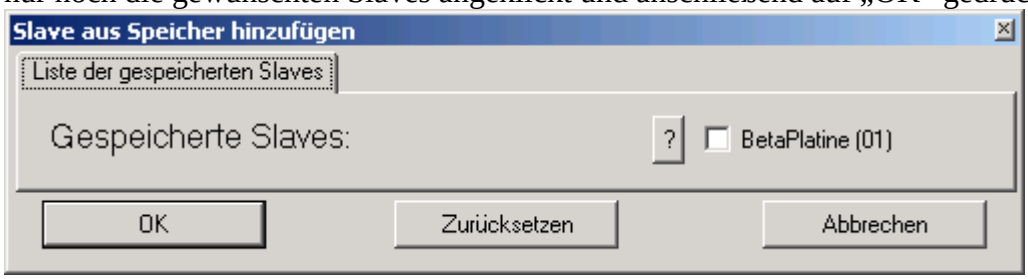


Der Vorgang wird durch ein klicken auf „OK“ abgeschlossen.

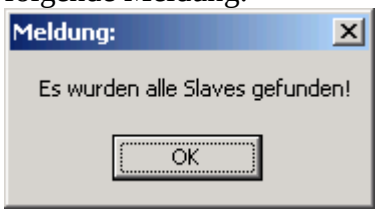
Vorhandene Slaves Hinzufügen



Slaves →Hinzufügen →Vorhanden erlaubt dem Benutzer bereits früher hinzugefügte Slaves auszuwählen. Dieser Menüpunkt erspart den Benutzer die Slaves in den Adressiermodus zu bringen und sorgt so für eine einfachere Handhabung.
Wurde dieser Menüpunkt ausgewählt erscheint eine Liste bereits eingesetzter Slaves. Es müssen nur noch die gewünschten Slaves angeklickt und anschließend auf „OK“ gedrückt werden.

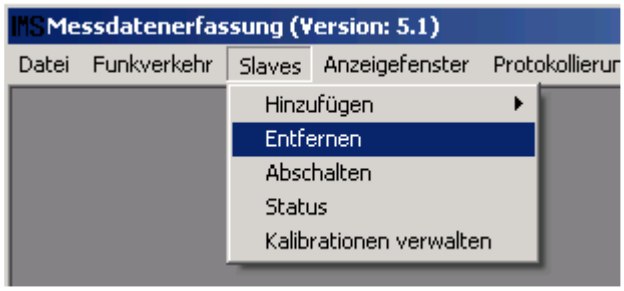


Danach wird die Funktion Slaves neu Suchen aufgerufen. Wurden alle Slaves erreicht erscheint folgende Meldung:

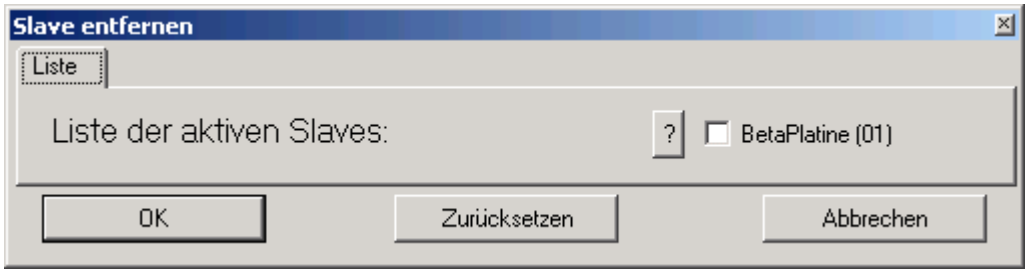


Durch betätigen der „OK“- Taste wird der Vorgang abgeschlossen.

Entfernen

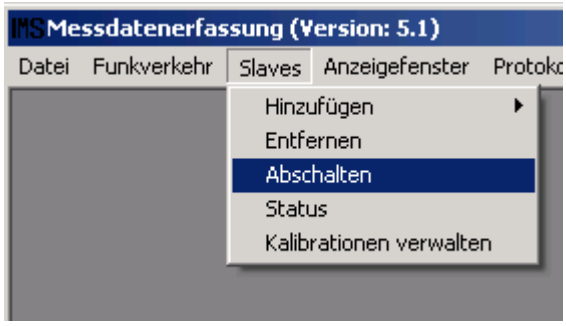


Slaves →Entfernen erlaubt den Benutzer nicht mehr benötigte Slaves aus dem Messsystem zu entfernen. Der Slave bleibt danach aber trotzdem in der Liste der gespeicherten Slaves vorhanden. Er steht bloß nicht mehr für Messungen zur Verfügung.
Achtung: Der Slave wird beim Entfernen nicht abgeschaltet und muss manuell abgeschaltet werden.
Wenn man diesen Slave später wieder benötigt muss man den Menüpunkt Slaves →Hinzufügen →Vorhanden aufrufen.
Hat man sich für das Entfernen eines Slaves entschieden, so müssen lediglich den zu entfernenden Slave angeklickt werden und zum Bestätigen die „OK“ Taste betätigt werden.

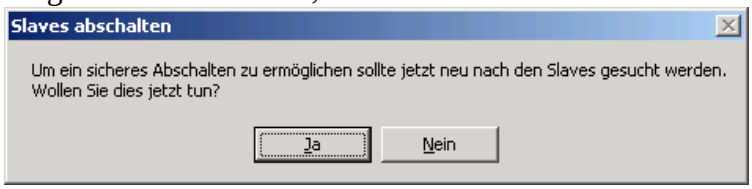


Die gewünschten Slaves werden nun entfernt.
Wurden alle aktiven Slaves entfernt erscheint folgende Meldung:

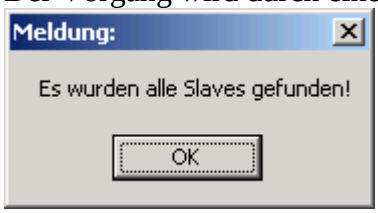
Abschalten



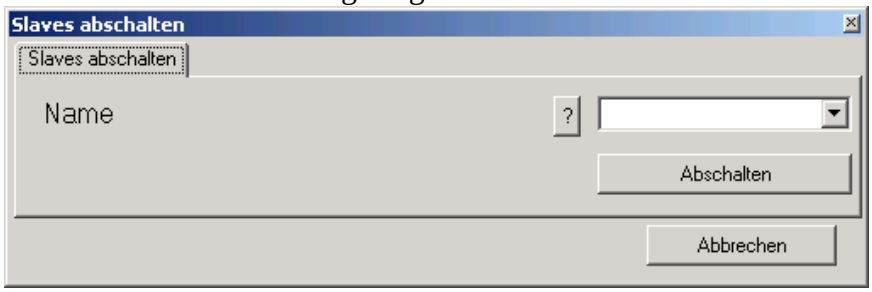
Slaves →Abschalten ermöglicht das Abschalten des Slaves per Funk. Damit diese Funktion ausgeführt werden kann, muss zunächst nach den Slaves neu gesucht werden.



Der Vorgang wird durch einen Klick auf den Button „Ja“ bestätigt.

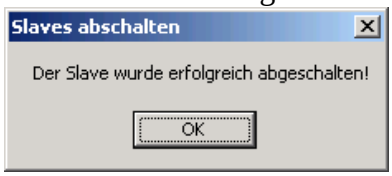


Nachdem diese Meldung erschienen ist, wird eine Liste der möglichen Slaves zum Abschalten angezeigt.

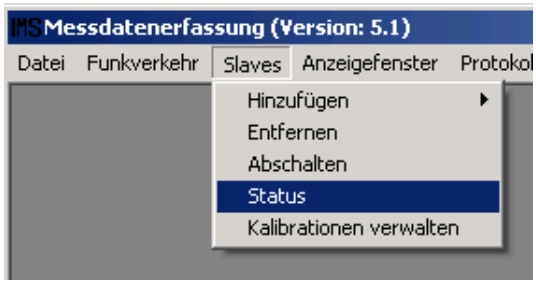


Hier kann man den gewünschten Slave wählen.
Zum Durchführen des Befehls muss danach auf den Button „Abschalten“ gedrückt werden.

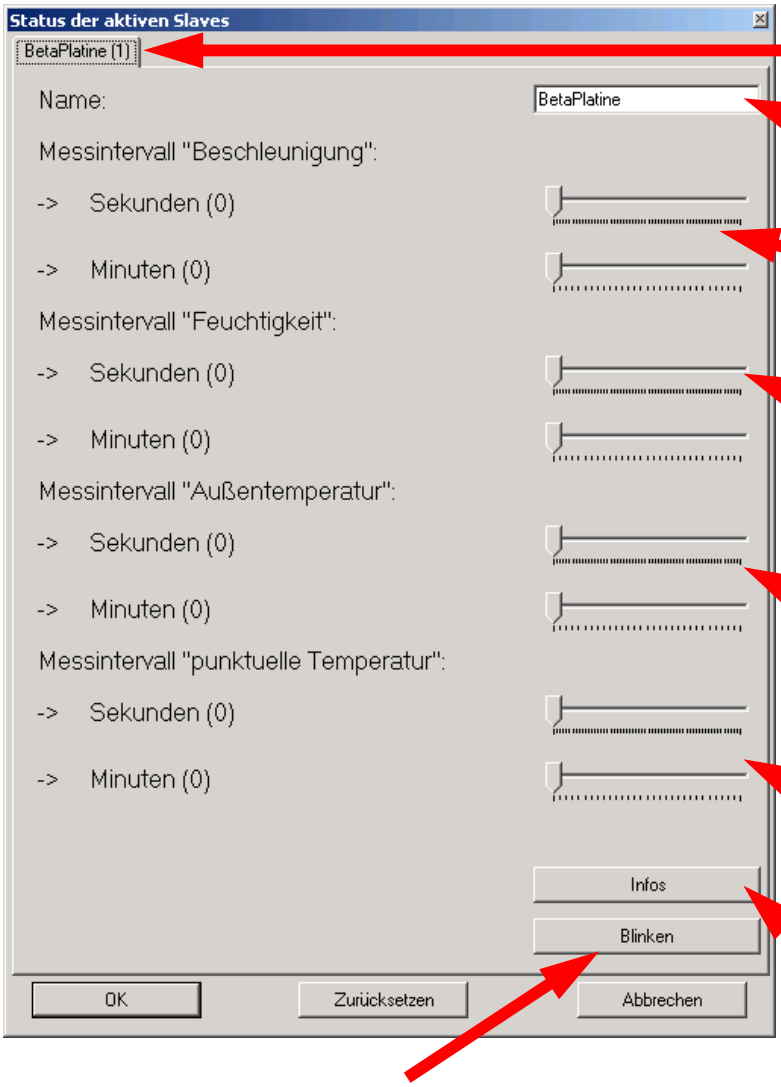
Sobald der Slave abgeschaltet wurde erscheint die folgende Meldung:



Status



Der Menüpunkt Slaves →Status dient dazu, um die Messintervalle der Sensoren der einzelnen Slaves einzustellen und nähere Informationen über den jeweiligen Slave zu erhalten.



Hier steht der Name des jeweiligen Slaves

Hier kann der Name des Slaves editiert werden

Hier kann das Messintervall des Gravitationssensors eingestellt werden.

Hier kann das Messintervall des Feuchtigkeitssensors eingestellt werden.

Hier kann das Messintervall der Außentemperatur eingestellt werden.

Hier kann das Messintervall des punktuellen Temperatursensors eingestellt werden.

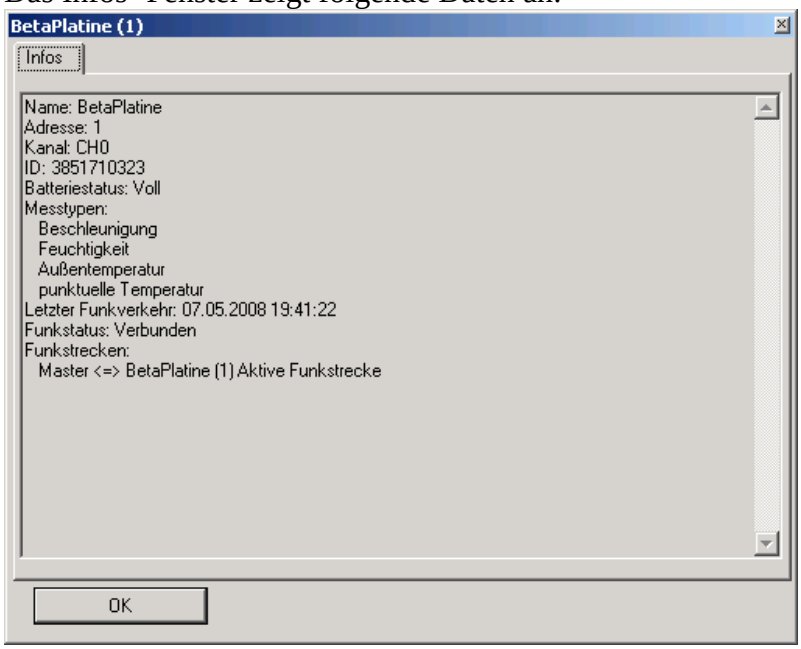
Hier erhält man nähere Informationen zum Slave

Diese Funktion lässt die LED des jeweiligen Slaves 5x blinken

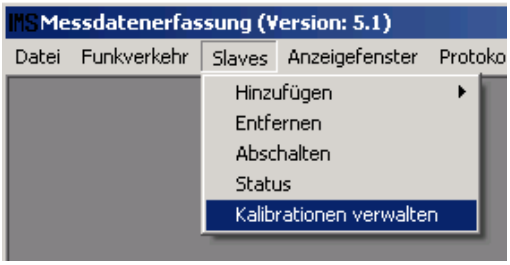
Die Einstellungen des Messintervalls hier, beeinflussen auch die Einstellungen des Messintervalls des Protokollierungssystems.

Infos

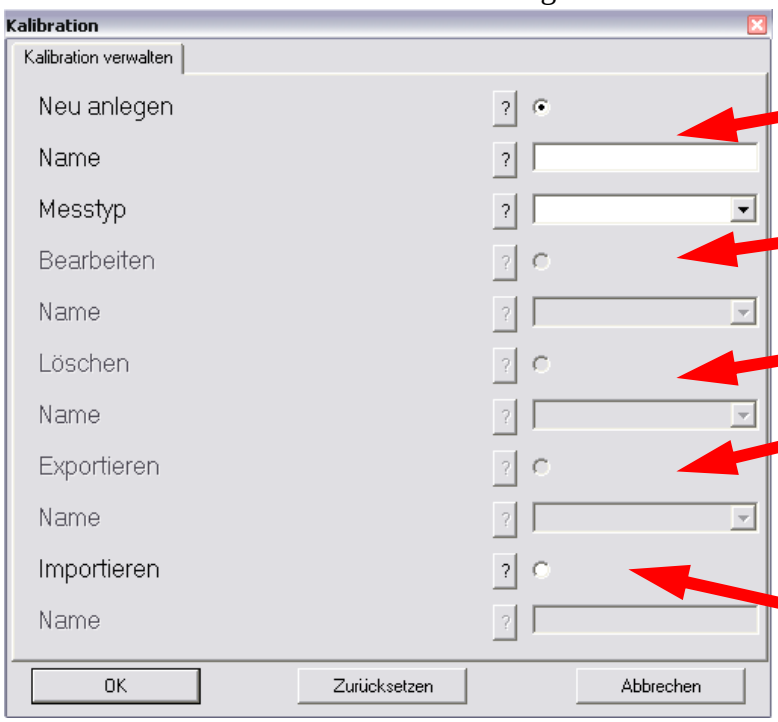
Das Infos- Fenster zeigt folgende Daten an:



Kalibrationen verwalten



Slaves →Kalibrationen verwalten erlaubt dem User das Anlegen, Bearbeiten, Importieren, Exportieren und Löschen von Kalibrationsdateien.
Diese Kalibrationen können für die Anzeigefenster oder zur Protokollierung verwendet werden.



Hier können neue Kalibrationsdateien angelegt werden

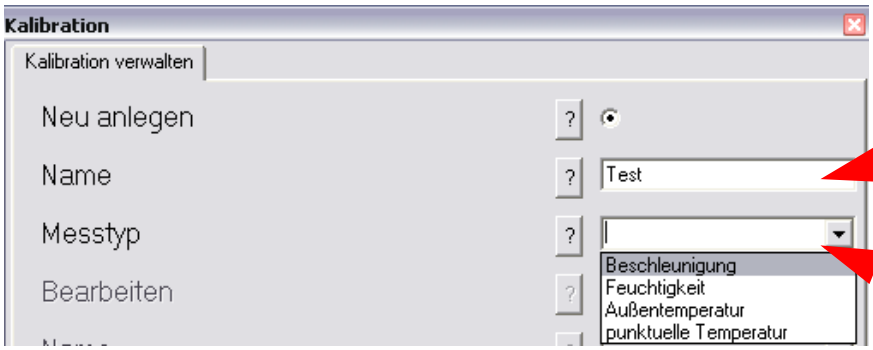
Hier können vorhandene Dateien bearbeitet werden

Hier können vorhandene Kalibrationen gelöscht werden

Hier können vorhandene Kalibrationsdateien exportiert werden

Hier können externe Kalibrationsdateien in das Messsystem integriert werden

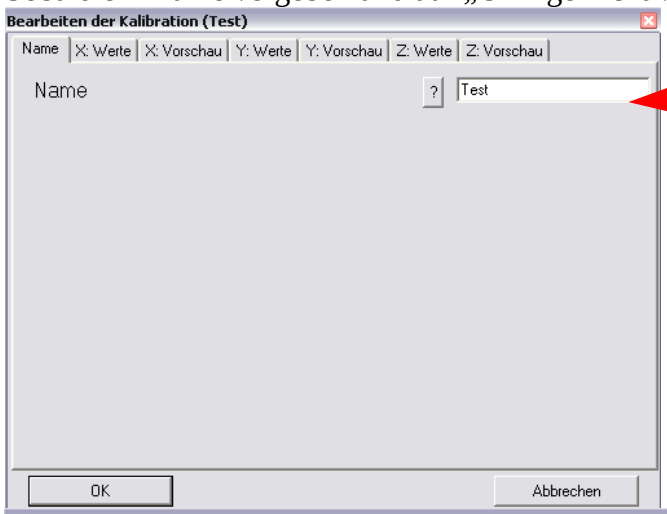
Kalibration neu anlegen



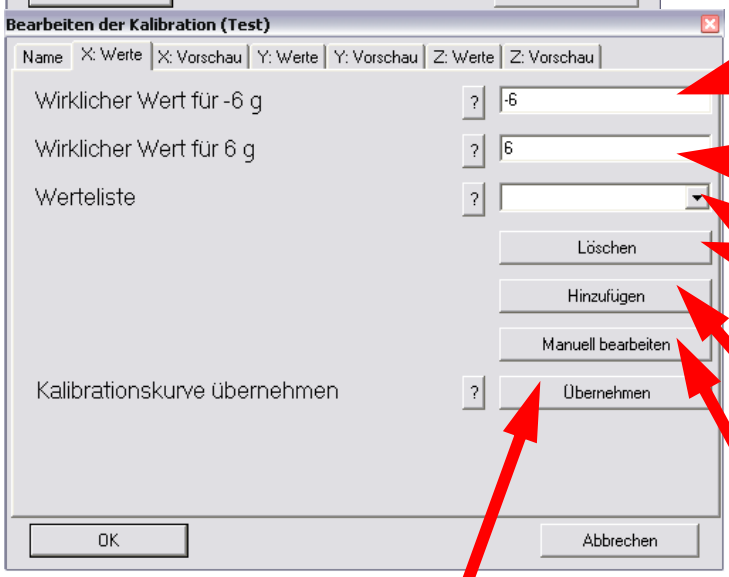
Hier kann man den Namen für die neue Kalibration vergeben

Hier kann ausgewählt werden für welchen Messtyp die Kalibration gelten soll

Sobald ein Name vergeben und auf „OK“ geklickt wurde kommt man in folgendes Menü:



Hier kann nachträglich noch der Name geändert werden



Hier kann der wahre Minimalwert angegeben werden

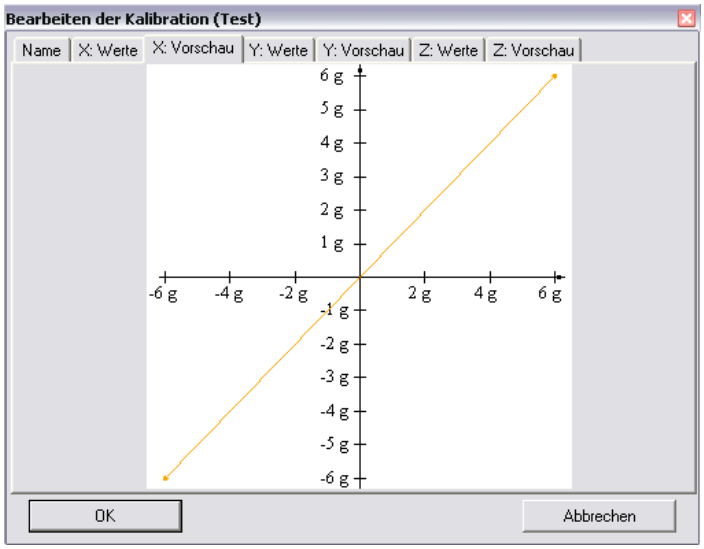
Hier kann der wahre Maximalwert angegeben werden

Hier kann man ein Wertepaar auswählen und durch betätigen des „Löschen“ Buttons entfernen

Mit Hinzufügen können neue Kalibrationswerte hinzugefügt werden

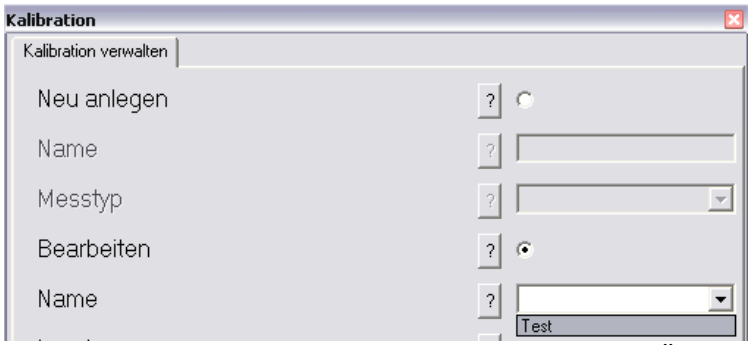
Dieser Button sorgt dafür, dass die Kalibration der X-Achse für die Y- und Z- Achse übernommen wird

Diese Funktion ermöglicht die grafische Bearbeitung der Kalibration

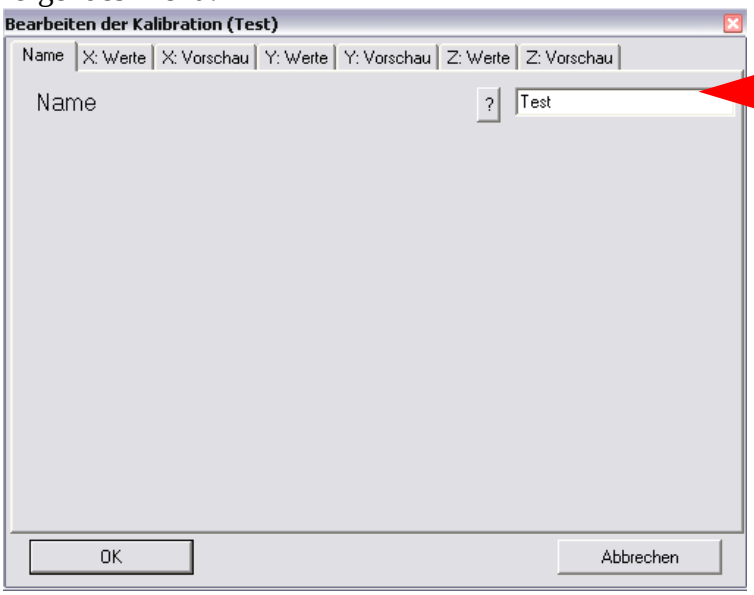


Die Vorschau ermöglicht dem Benutzer eine grafische Ansicht der Kalibration

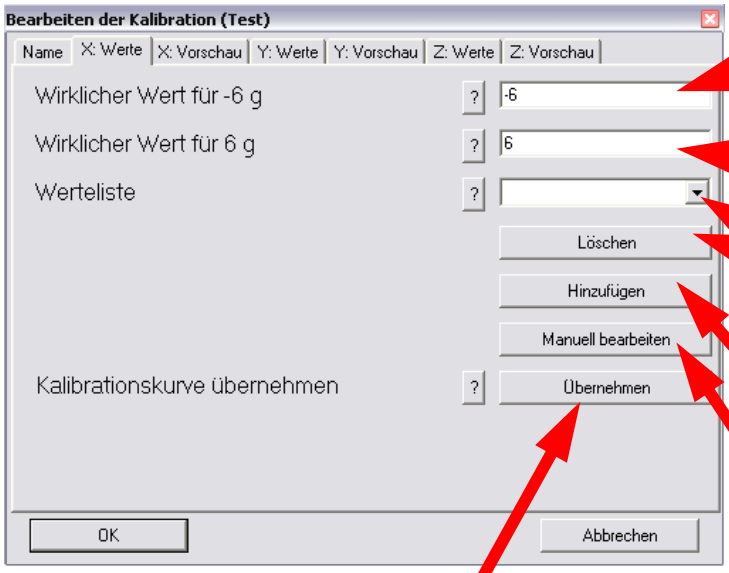
Kalibration bearbeiten



Die Funktion Bearbeiten ermöglicht nachträgliche Änderungen der Kalibrationen.
Sobald eine Kalibrationseinstellung ausgewählt und auf „OK“ geklickt wurde, kommt man in folgendes Menü:



Hier kann nachträglich noch der Name geändert werden



Hier kann der wahre Minimalwert angegeben werden

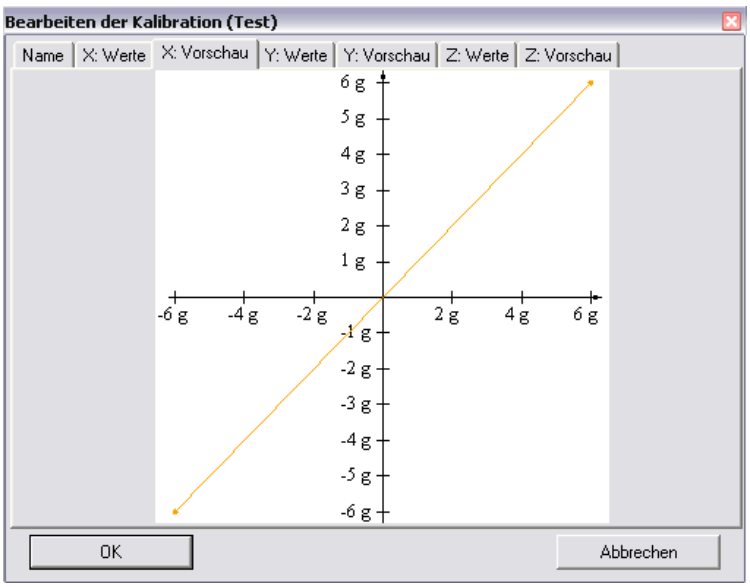
Hier kann der wahre Maximalwert angegeben werden

Hier kann man ein Wertepaar auswählen und durch betätigen des “Löschen“ Buttons entfernen

Mit „Hinzufügen“ können neue Kalibrationswerte hinzugefügt werden

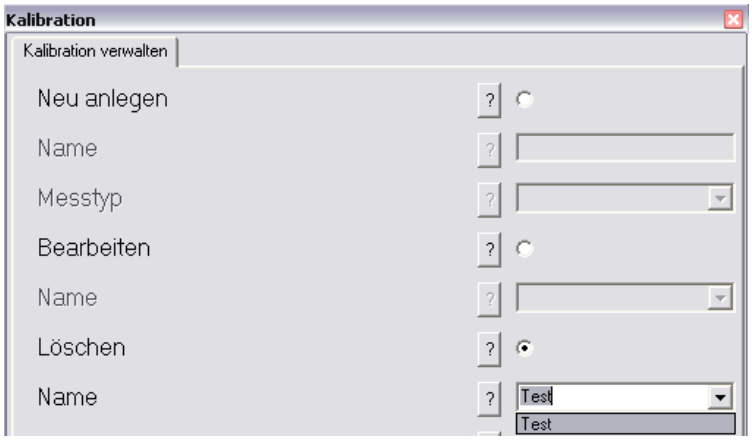
Dieser Button sorgt dafür, dass die Kalibration der X-Achse für die Y- und Z- Achse übernommen wird

Diese Funktion ermöglicht die grafische Bearbeitung der Kalibration



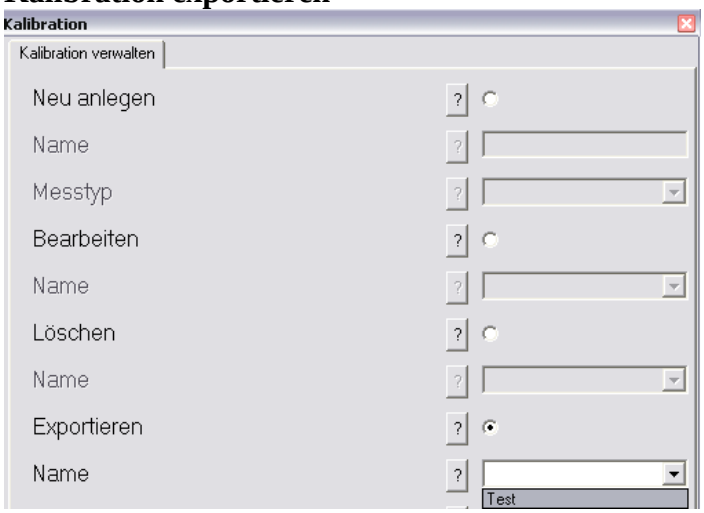
Die Vorschau ermöglicht dem Benutzer eine grafische Ansicht der Kalibration

Kalibration löschen

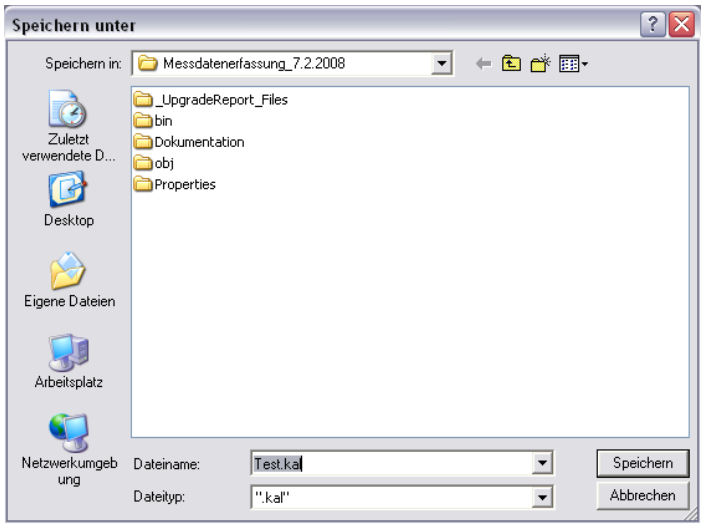


Durch Auswahl eines Namens, der zu löschenden Datei, wird mit „OK“ die Kalibration entfernt.

Kalibration exportieren

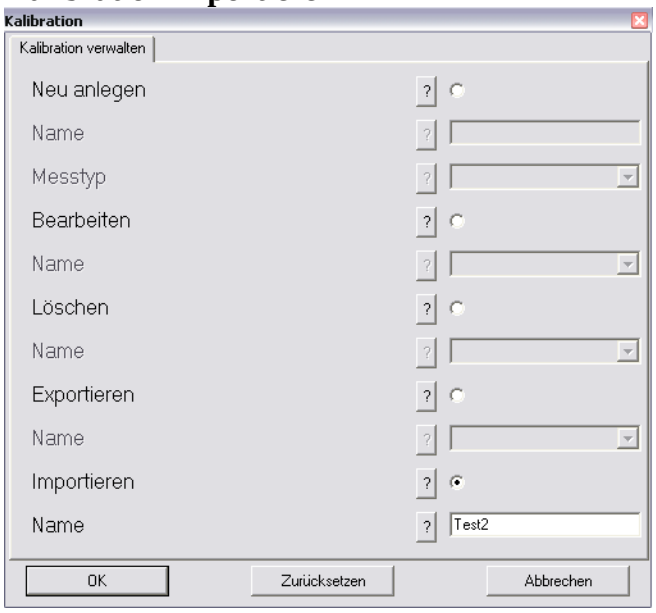


Diese Funktion erlaubt den Export von vorhandenen Kalibrationsdateien. Dies dient dazu, um eine Datei auf mehreren Rechnern zu verwenden.

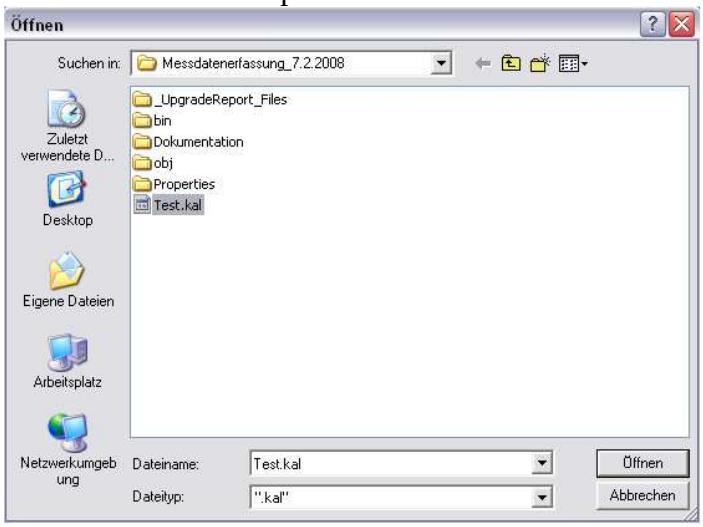


Die Datei wird mit der Endung „.kal“ abgespeichert.

Kalibration importieren

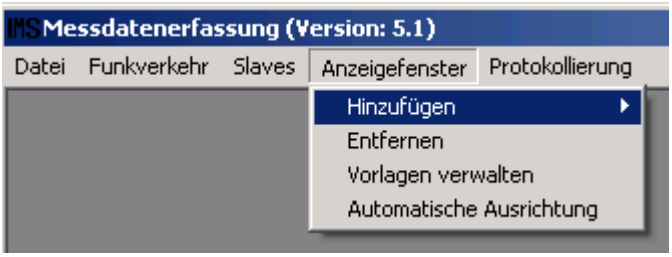


Zum Importieren einer Kalibrationsdatei muss man zu nächst einem Namen eingeben, unter den man die Kalibration später verwenden will.



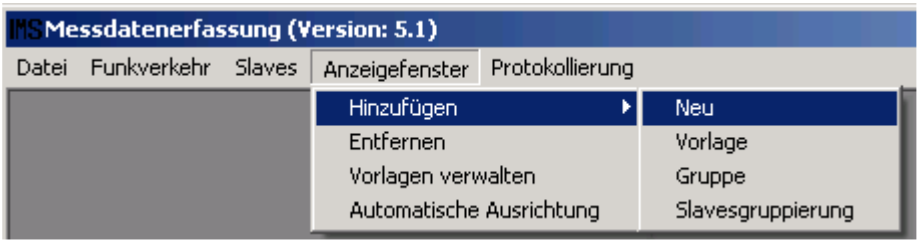
Danach muss eine geeignete Datei für den Import ausgewählt werden. Die Endung der Datei muss „.kal“ sein.

Anzeigefenster

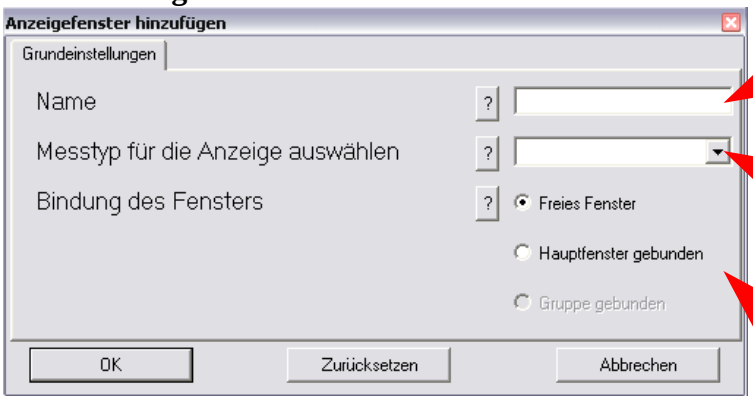


Der Menüpunkt Anzeigefenster dient im Wesentlichen dazu, um die Anzeige der Messdaten zu verwalten.

Hinzufügen



Neu hinzufügen

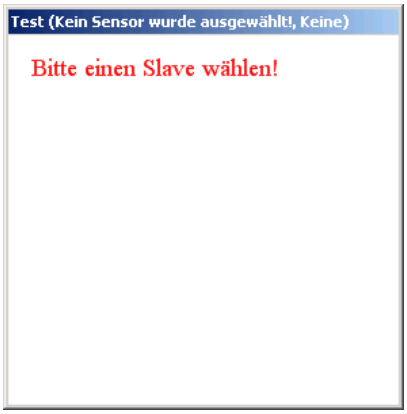


Hier kann der Name des Anzeigefensters vergeben werden

Hier muss der Messtyp festgelegt werden, für den das Fenster später benutzt wird

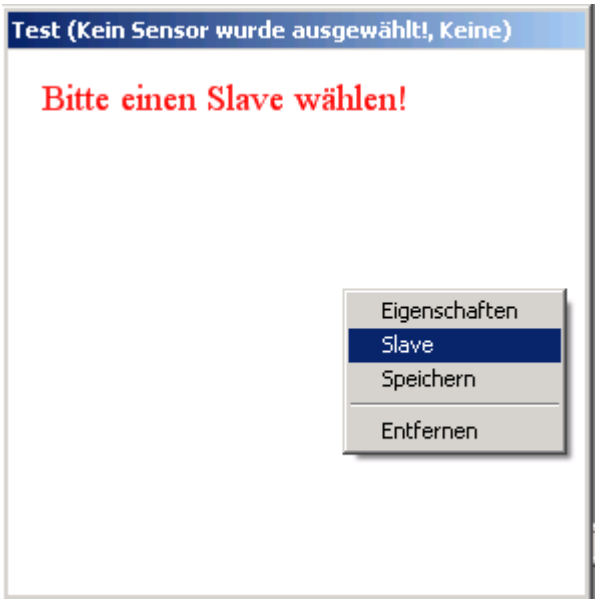
Hier kann eingestellt werden, in welchen Bereich das Fenster bewegt werden kann.

Unter Hinzufügen →Neu kann ein neues Fenster zur Anzeige der Messdaten angelegt werden. Ein neu erstelltes Fenster sieht wie folgt aus:

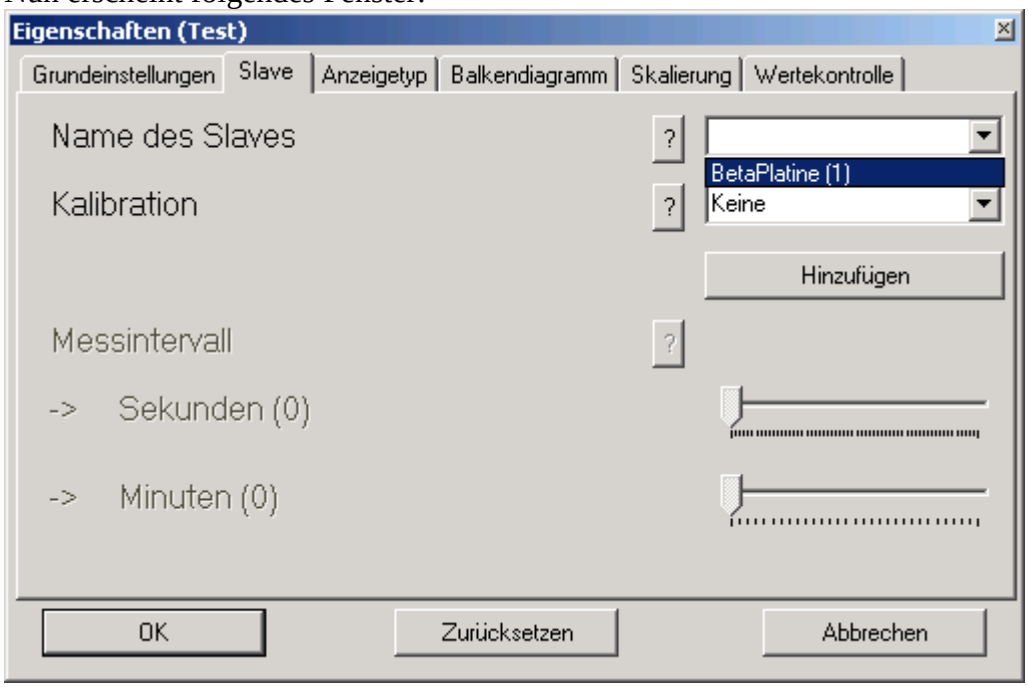


Nun muss das Anzeigefenster noch mit einem Slave verbunden werden und dies funktioniert wie folgt:

- Auswählen des gewünschten Fensters
- Betätigen der rechte Maustaste
- Auswählen des Menüpunktes Slave

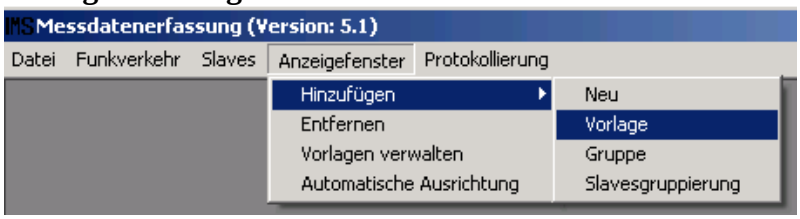


Nun erscheint folgendes Fenster:

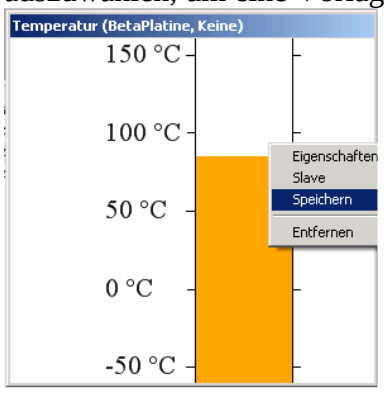


Hier kann man einen der möglichen Slaves unter Name des Slaves hinzufügen, eine Kalibration einbinden und das Messintervall einstellen.
Sobald ein Slave gewählt und auf „OK“ geklickt wurde, werden die Messdaten angezeigt.

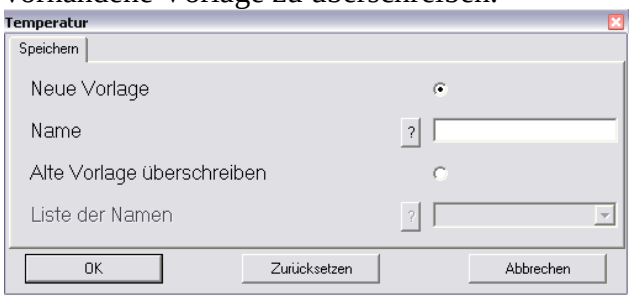
Vorlage hinzufügen



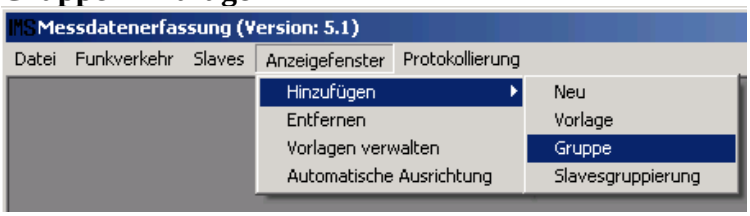
Unter Anzeigefenster → Hinzufügen→ Vorlage können gespeicherte Vorlagen zur Erstellung neuer Fenster herangezogen werden.
Damit dies möglich ist muss zunächst eine Vorlage erstellt werden.
Dies erfolgt durch Rechtsklick auf ein Anzeigefenster. Danach ist der Punkt Speichern auszuwählen, um eine Vorlage zu erstellen.



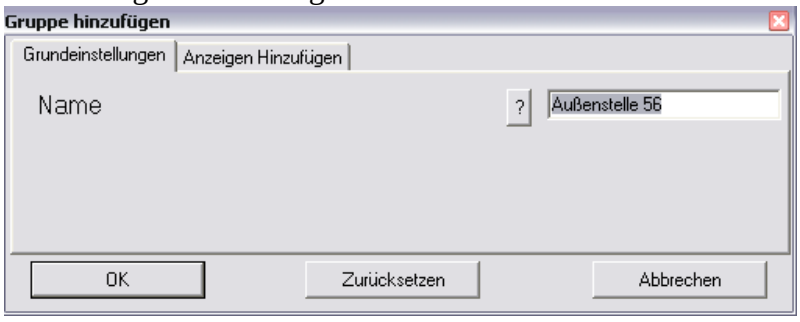
Danach erfolgt eine Aufforderung zur Eingabe eines Namens für die Vorlage oder eine bereits vorhandene Vorlage zu überschreiben.

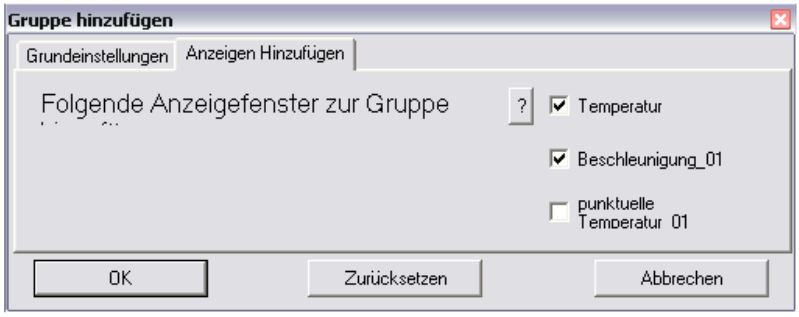


Gruppe hinzufügen

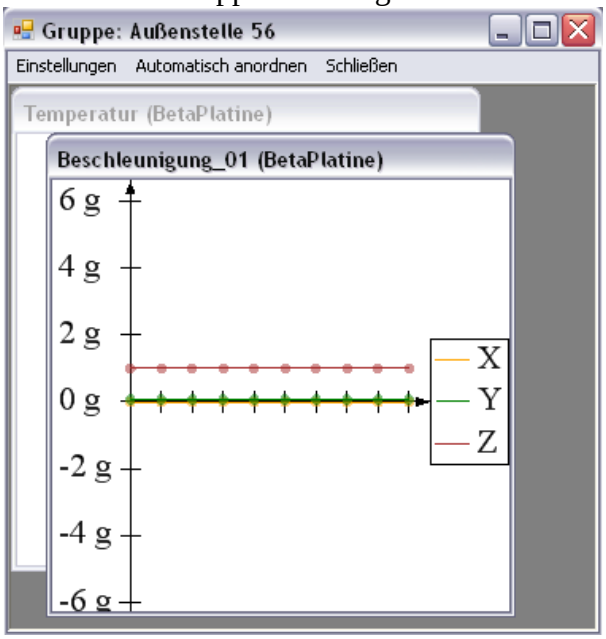


Die Funktion dient zum Gruppieren der Anzeigefenster. Dies soll für mehr Übersicht sorgen.
Unter Anzeigefenster → Hinzufügen→ Gruppe kann eine neue Gruppe erstellt werden.
Zunächst wird man aufgefordert einen Namen für die Gruppe anzugeben. Danach kann man die hinzuzufügenden Anzeigefenster auswählen.

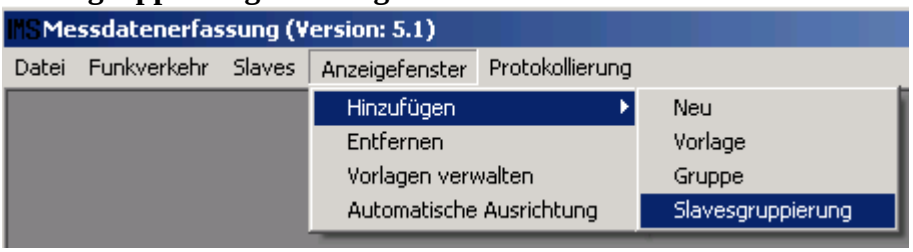




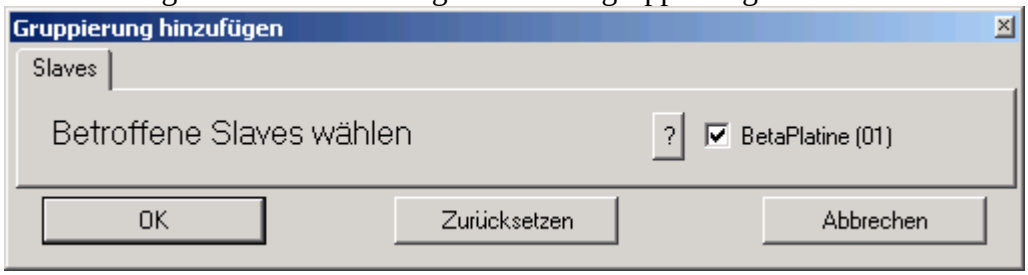
Die erstellte Gruppe sieht folgendermaßen aus:



Slavesgruppierung hinzufügen

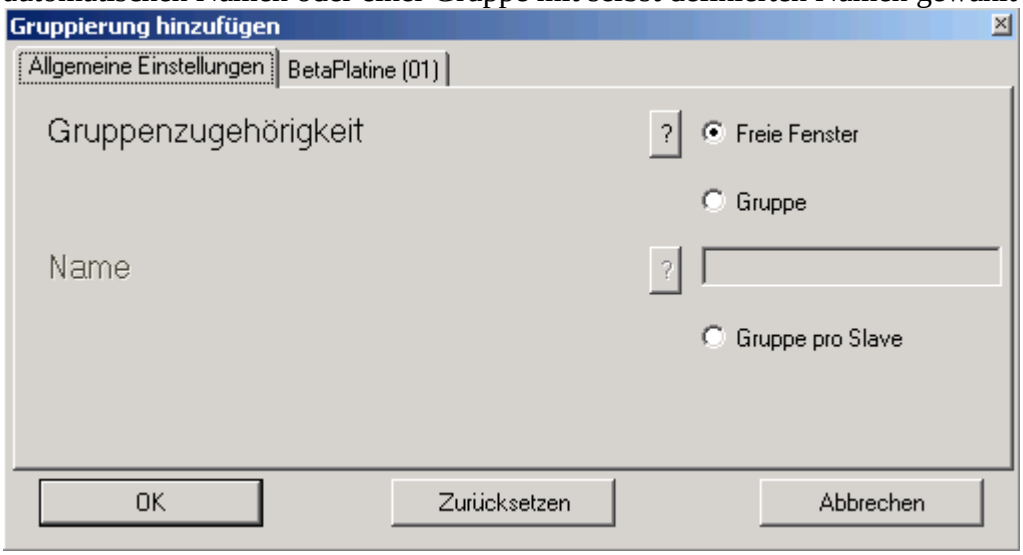


Diese Funktion ist eine Sonderform der Funktion Gruppe. Sie erstellt automatisch für alle gewünschten Sensoren der gewählten Slaves Anzeigefenster. Für diese Funktion müssen keine Anzeigefenster vorhanden sein.
Unter Anzeigefenster → Hinzufügen→ Slavesgruppierung wird diese Funktion aufgerufen.

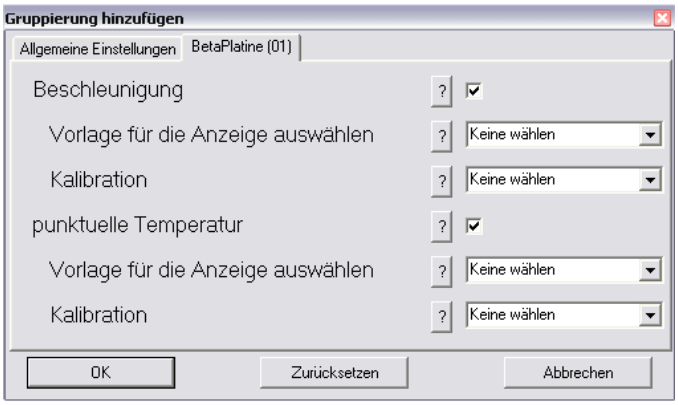


Hier kann man auswählen von welchen Slaves Anzeigefenster erstellt werden sollen.
Die gewünschten Sensoren werden im folgenden Schritt ausgewählt.

Zu nächst kann ausgewählt werden, wie die einzelnen Fenster gruppiert werden.
Hierbei kann zwischen freien Fenstern, dies entspricht keiner Gruppe, einer Gruppe pro Slave mit automatischen Namen oder einer Gruppe mit selbst definierten Namen gewählt werden.

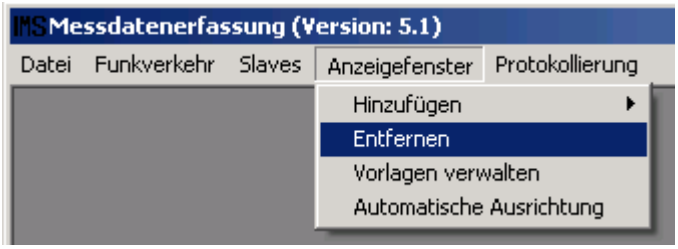


Danach kann noch eingestellt werden, welche Sensoren pro Slave angezeigt werden sollen.
Weiters kann hier eingestellt werden, ob spezielle Vorlagen oder Kalibrationen verwendet werden sollen.

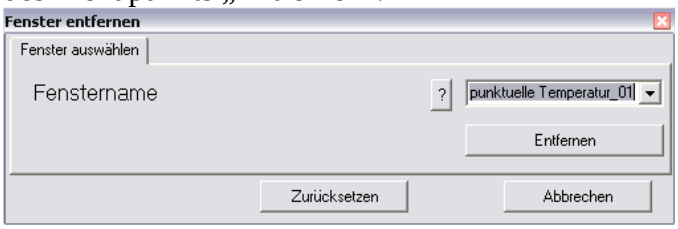


Sobald der OK- Button betätigt wird, werden die gewünschten Fenster erzeugt.

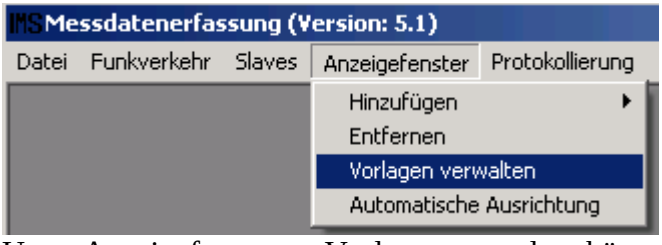
Entfernen



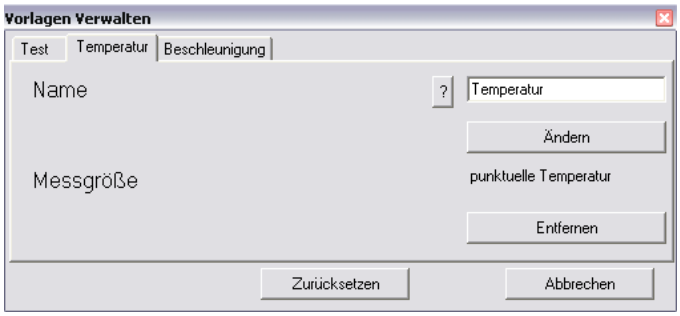
Unter Anzeigefenster → Entfernen können nicht mehr benötigte Anzeigefenster entfernt werden.
Alternativ kann diese Funktion auch durch Rechtsklick auf ein Anzeigefenster und dann Auswahl des Menüpunkts „Entfernen“.



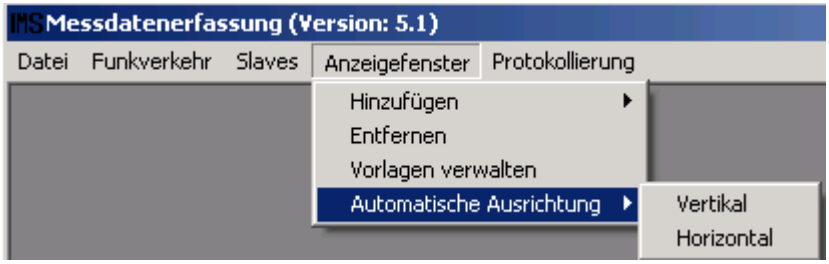
Vorlagen verwalten



Unter Anzeigefenster → Vorlagen verwalten können die angelegten Vorlagen bearbeitet und entfernt werden.



Automatisch anordnen



Die Funktion automatisch anordnen erlaubt den Benutzer die an das Hauptfenster gebundenen Anzeigefenster optimal anzuordnen. Man kann dabei zwischen vertikaler oder horizontaler Ausrichtung der Fenster wählen.

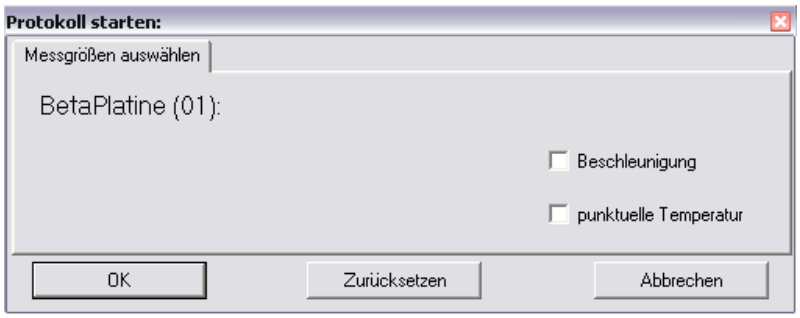
Protokollierung

Dieser Menüpunkt erlaubt dem Benutzer die Protokollierung der Messdaten.

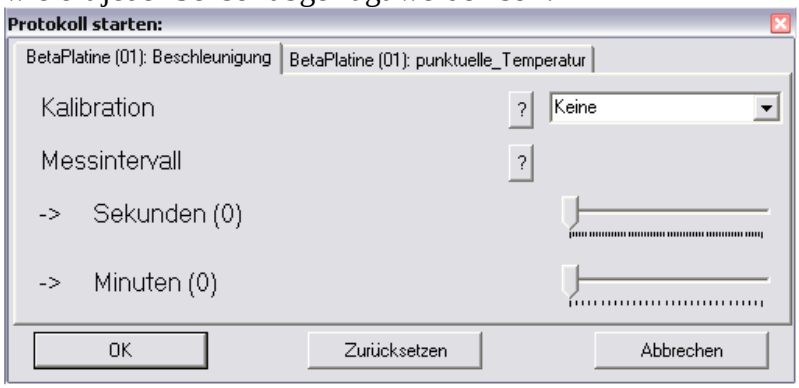
Starten

Diese Funktion startet die Protokollierung der Messdaten.

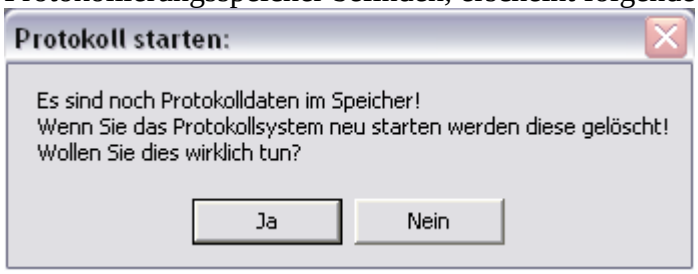
Zuerst muss ausgewählt werden, welche Sensoren der jeweiligen Slaves protokolliert werden sollen.



Danach kann man wählen ob eine Kalibration für die jeweiligen Sensoren benutzt werden soll und wie oft jeder Sensor abgefragt werden soll.



Wenn sich beim Starten des Protokollsystems noch Messdaten im temporären Protokollierungsspeicher befinden, erscheint folgende Meldung:



Durch Klick auf den Button „Ja“ wird die temporäre Zustandsspeicherungsdatei überschrieben.

Stoppen

Unter Protokollierung → Stoppen kann die Protokollierung der Messdaten beendet werden.

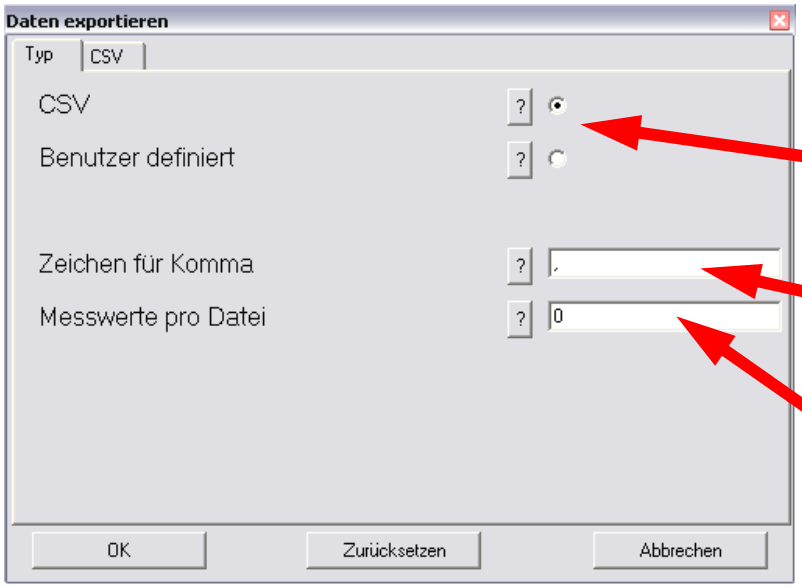
Daten löschen

Die Funktion Daten löschen dient zum Löschen der temporären Messdaten.

Daten exportieren

Unter Protokollierung → Daten exportieren können die temporär protokollierten Messdaten in eine externe Datei gespeichert werden. Für den Export sind zwei Datenformate möglich:

- CSV
- Benutzer definiert



Hier kann einer der beiden möglichen Exporttypen ausgewählt werden

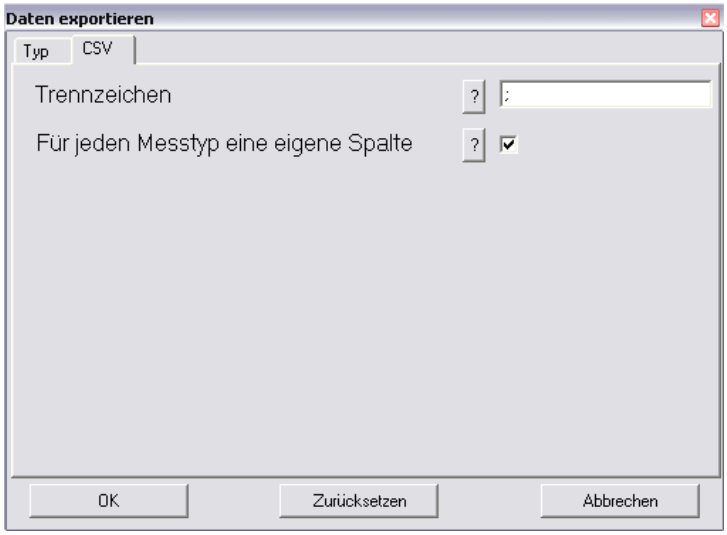
Hier kann das Kommazeichen eingestellt werden

Hier kann definiert werden, wie viele Messwerte pro Datei exportiert werden sollen. Der Wert 0 stellt einen Sonderwert dar und sorgt dafür, dass sämtliche Messwerte in eine Datei gespeichert werden.

Je nach dem, ob beim Starten der Protokollierung eine Kalibration ausgewählt wurde oder nicht, werden entweder die Kalibrationsdaten oder die Rohdaten ausgegeben.

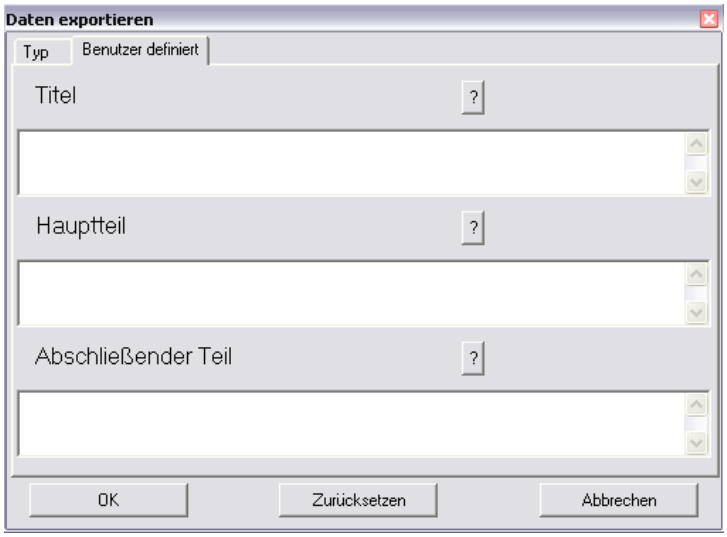
Je nach Auswahl des Exporttyps können folgende Sondereinstellungen vorgenommen werden:

CSV



Hier kann eingestellt werden welches Zeichen zur Trennung der einzelnen Messwerte benutzt werden und ob für jeden Messtyp eine eigene Spalte angelegt werden soll.

Benutzer definiert



Beim benutzerdefinierten Format kann der User selbst festlegen wie die Exportdatei aussehen soll. Für den Hauptteil gelten folgende Regeln:

- Für jeden Messwert wird der Hauptteil neu geschrieben!
  Damit immer in eine neue Zeile geschrieben wird muss am Zeilenende ein „Enter“ eingefügt werden.
- Folgende Marken zum Einsetzen von Werten stehen zur Verfügung:

<TAB> = wird immer mit dem TAB Zeichen ersetzt
[NAME] = Name des Slaves
[ID] = ID des Slaves
[TYP] = Typ der Messung
[CALIBRATION] = Name der verwendeten Kalibrierung
[Zeitstempel] = Repräsentiert die Zeit in 100- Nanosekundenschritte
[Datum] = Gibt das Datum und die Uhrzeit an

[Wert_0] = Wert für z.B. Temperaturmessung und für die X-Achse bei Gravitationsmessung

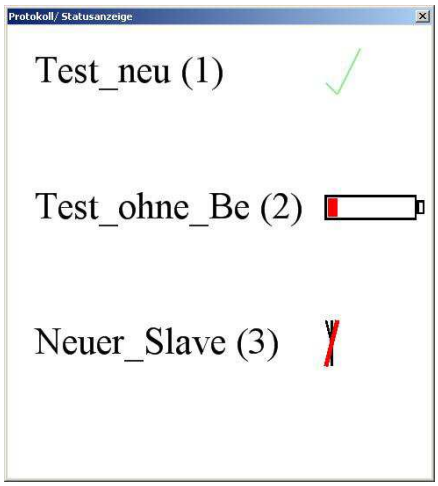
[Wert_1] = Wert für Y-Achse bei Gravitationsmessung
[Wert_2] = Wert für Z-Achse bei Gravitationsmessung
Wenn ein Wert nicht vorhanden ist wird er immer durch nichts ersetzt.

Wenn man eindeutige Marken für die Protokollierung verwenden will und nicht Wertemarken, die für mehrere Slaves benutzt werden, so ist folgende Struktur zu benutzen:
[Wert_“ID“_“M“_“S“] = „ID“ ersetzt man durch die ID des jeweiligen Slaves,
„M“ bestimmt den Messwert,
„S“ bestimmt die Anzahl der Werte zum jeweiligen Messwert (beim Gravitationssensor 0 bis 2, anderenfalls 0)
Diese Marken werden dynamisch während des Betriebs ermittelt. Aus der Hilfe des Mittelteils können sie herauskopiert werden.

Status anzeigen

Die Statusanzeige informiert den Benutzer über aktuelle Ereignisse während des Messvorgangs. Bei besonderen Vorkommnissen, wie zum Beispiel einer Funkunterbrechung oder einer nahezu leeren Batterie, erscheint das Statusfester automatisch.

Wenn keine Besonderheiten auftreten sieht der Status so aus:

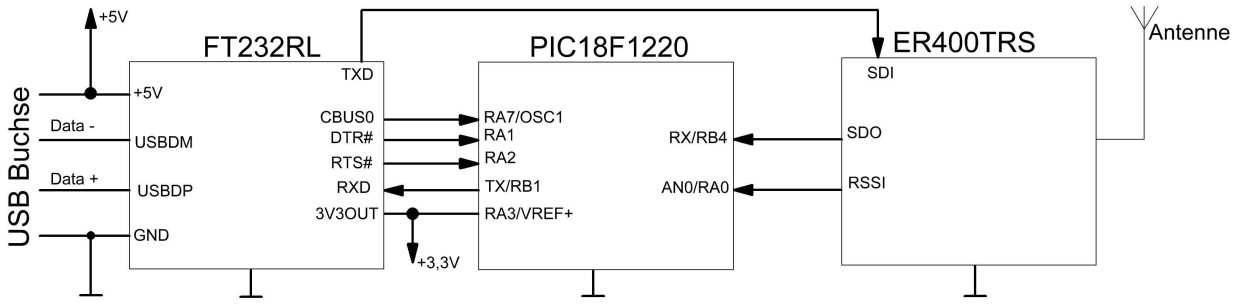


Hat ein Akku den Schwellenwert von 3V unterschritten, so erscheint folgendes Symbol:

Ist die Verbindung zu einem Slave unterbrochen worden, so erscheint dieses Symbol:

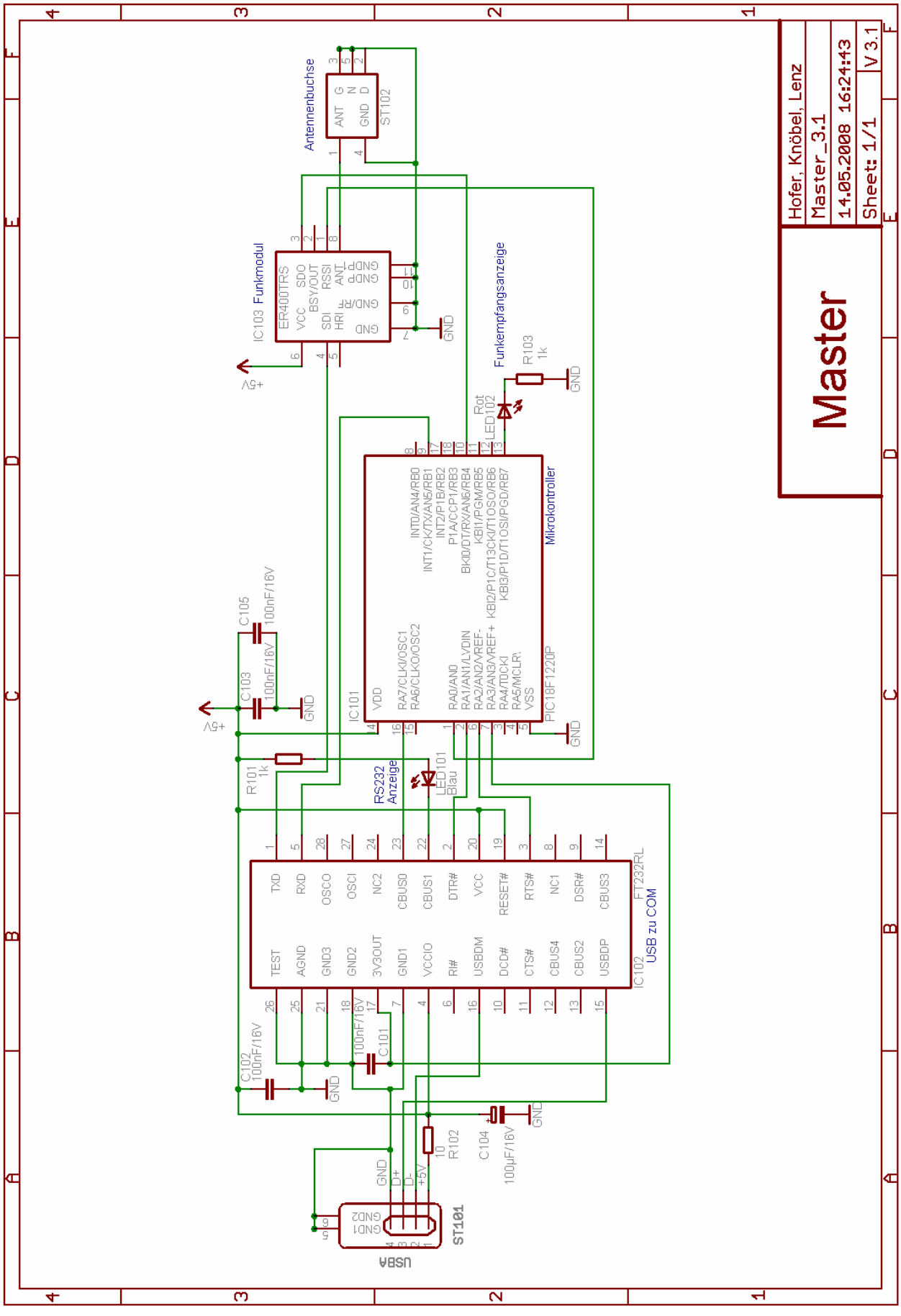
5.1.2. Master

Blockschaltbild des Masters

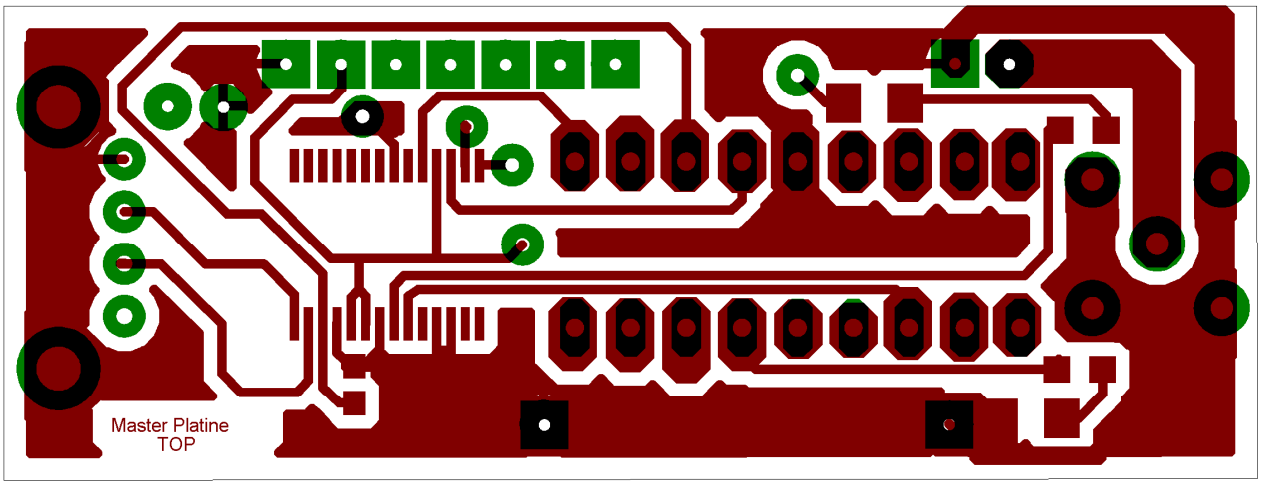


Detaillierte Informationen zum Master sind im Kapitel 5.2.2. zu finden.

5.1.2.1. Schaltplan Master

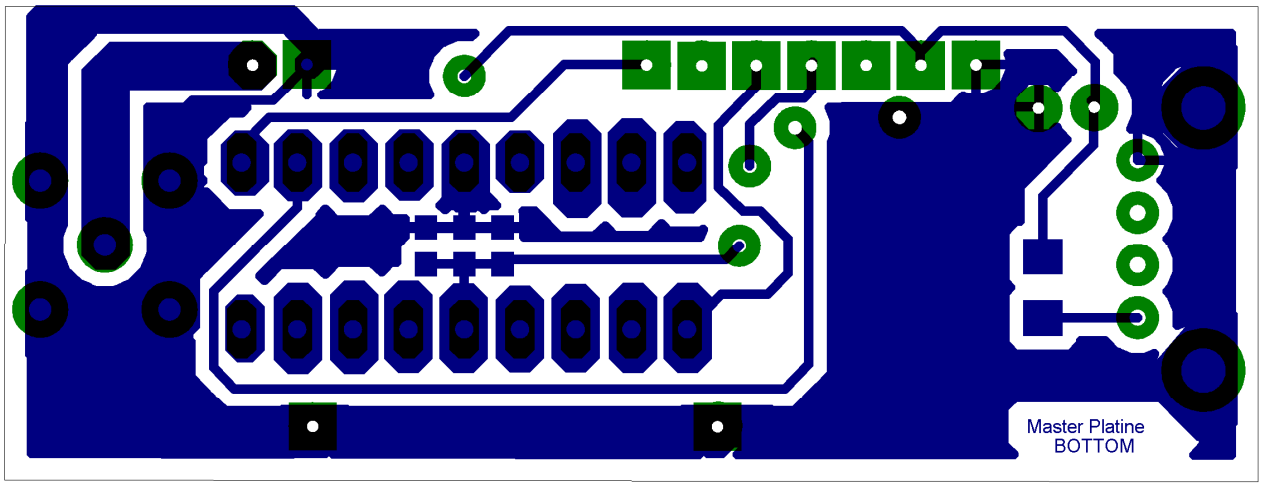


5.1.2.2. Layout Master Top Layer



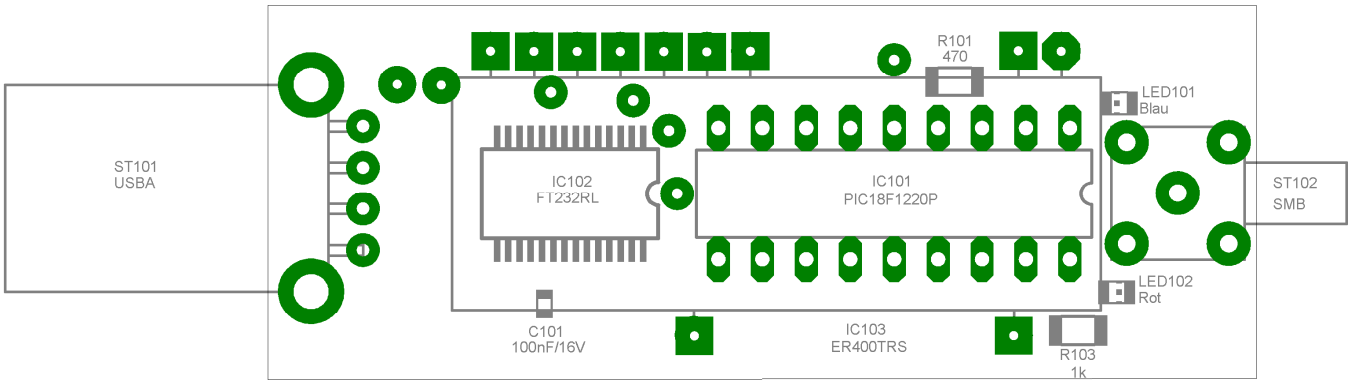
Top View
Abbildung nicht im Maßstab
Abmessungen: 57mm x 21mm

5.1.2.3. Layout Master Bottom Layer



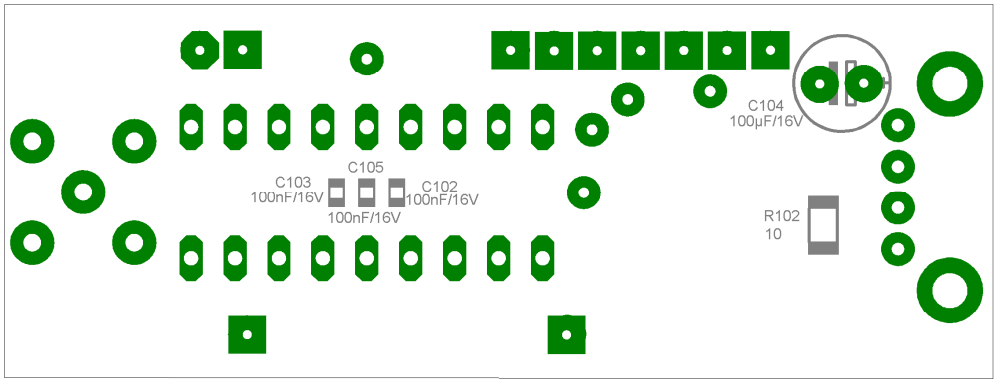
Bottom View
Abbildung nicht im Maßstab
Abmessungen: 57mm x 21mm

5.1.2.4. Bestückungsplan Master Top Place



Top View
Abbildung nicht im Maßstab
Abmessungen: 57mm x 21mm

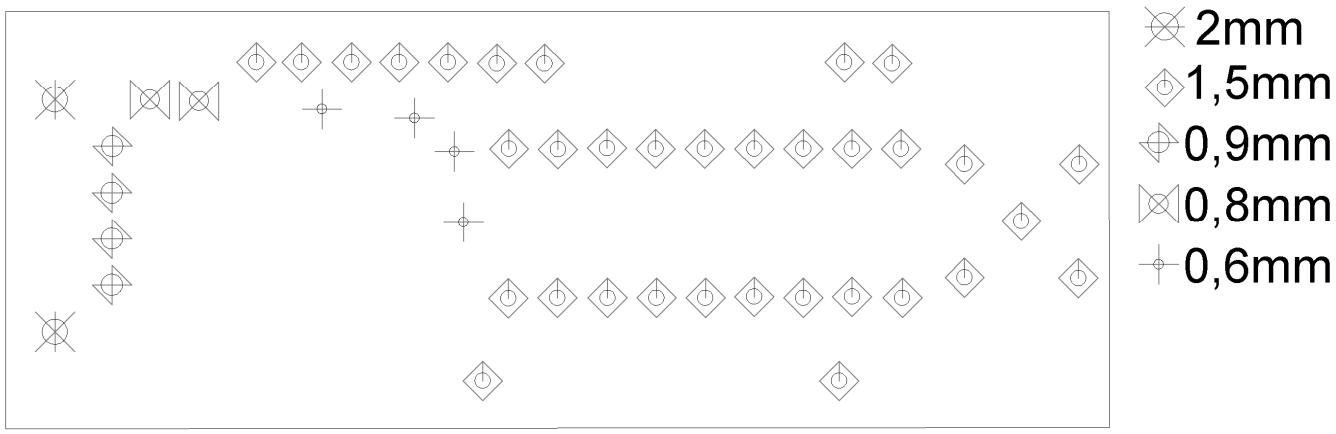
5.1.2.5. Bestückungsplan Master Bottom Place



Bottom View
Abbildung nicht im Maßstab
Abmessungen: 57mm x 21mm

Achtung: Mit der Bestückung des Funkmoduls (IC103) und des PICs (IC101) sollte bis nach der Inbetriebnahme gewartet werden.
Nähere Informationen sind dem Kapitel „5.1.2.8 Inbetriebnahme des Masters“ zu entnehmen.

5.1.2.6. Bohrplan Master



Top View
Abbildung nicht im Maßstab
Abmessungen: 57mm x 21mm

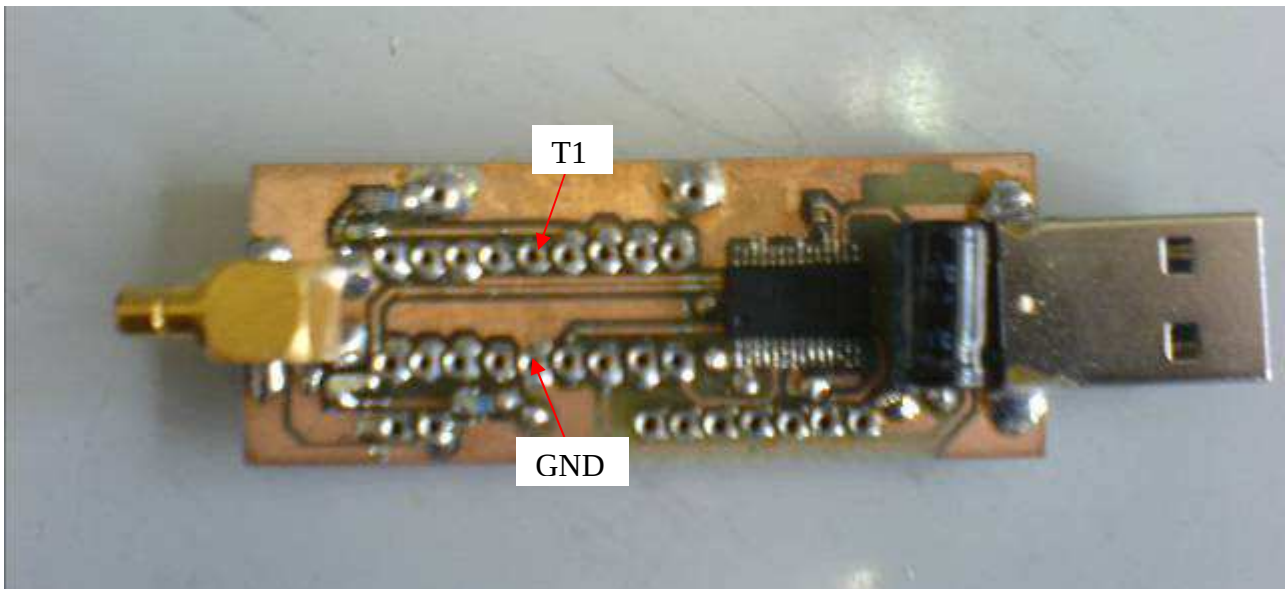
5.1.2.7. Stückliste Master

Nr.	Stk.	Bez.	Typ	Wert	Bemerkung
1	1	LED102	Leuchtdiode Rot	Stanley BR1111C	0805
2	1	LED101	Leuchtdiode Blau	HSMB- C110	0805
3	1	IC102	USB zu COM	FT232RL	28 Pin SSOP
4	1	IC103	Transceiver	ER400TRS	eigenes Package
5	1	IC101	Mikrokontroller	PIC18F1220	DIL18
6	4	C101, C102, C103, C105	Keramikkondensator	100nF ±10% / 16V	0603
7	1	C104	Elektrolytkondensator	100µF ±20% / 16V	RM2 (6,3 x 2,5) liegend
8	2	R101, R103	Widerstand	1kΩ ±1% / 1/8W	1206
9	1	R102	Widerstand	10Ω ±1% / 1/8W	1206
10	1	ST101	USBA Verbindung	USBA Stecker	PN87520
11	1	ST102	SMB Kupplung	TYCO Electronics 50Ω	SMB

5.1.2.8. Inbetriebnahme des Masters

Nach dem vollständigen bestücken der Platine muss sie noch getestet werden. Wie die Schaltung überprüft werden kann, wird in diesem Kapitel beschrieben.

Übersicht der Testpunkte:



Testen der Stromaufnahme:

Die Messung der Stromaufnahme wird ohne Funkmodul (IC103) und ohne PIC (IC101) durchgeführt.
Die Schaltung wird über den Testpunkt T1 mit 5V versorgt.
Beim Anlegen der Spannung muss kurz die blaue LED aufleuchten.
Dann sollte sich eine Stromaufnahme von 100µA einstellen.

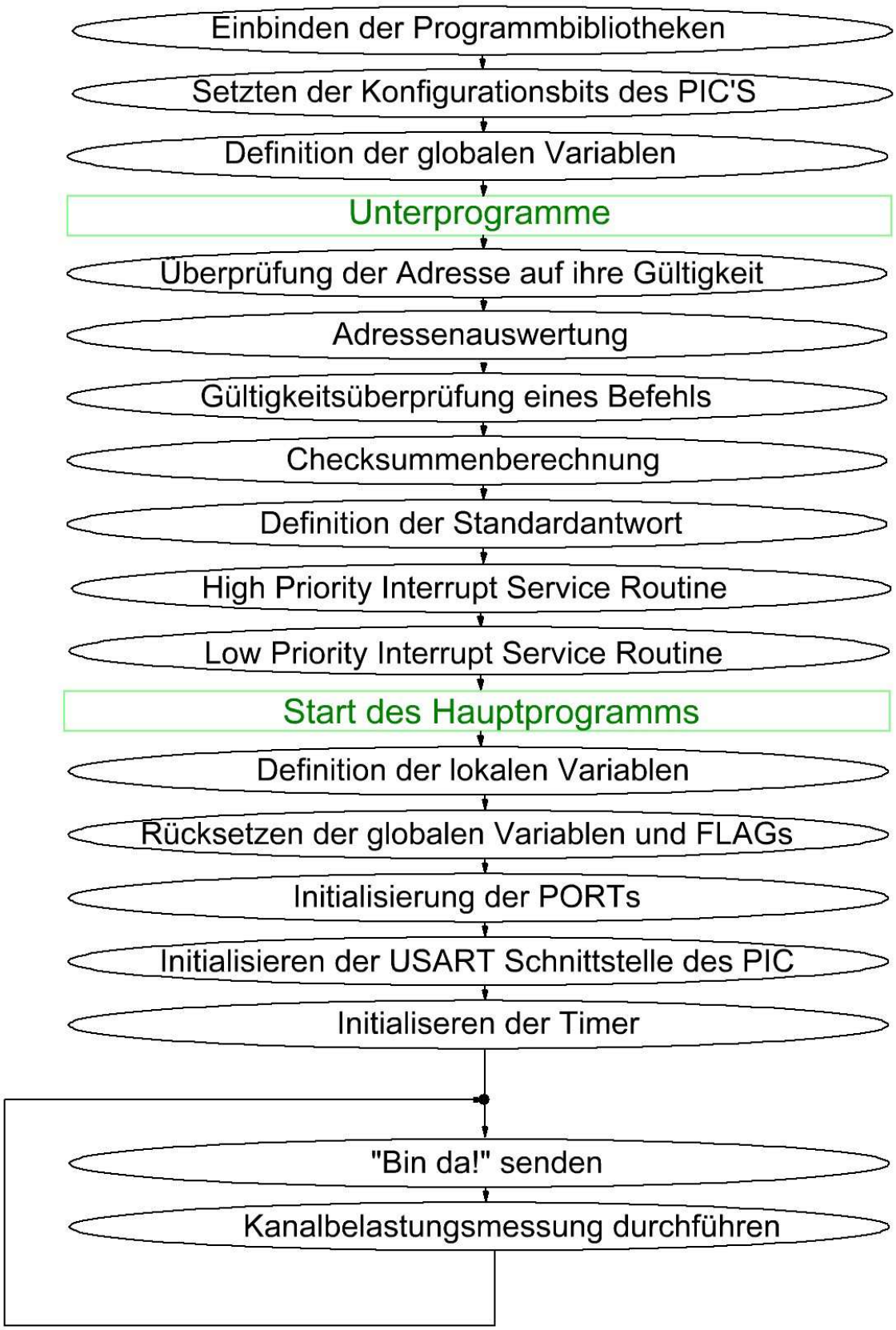
Testen der USB- Kommunikation:

Dazu muss der Master voll bestückt an einen PC angeschlossen werden.
Weiters muss für diesen Test bereits die PC – Software mit sämtlichen benötigten Komponenten, wie im Kapitel 5.1.1.3 beschrieben, installiert werden.
Dies kann über einen USB - Hub realisiert werden um den PC zu schützen.
Beim Anstecken muss der Master vom PC sofort erkannt werden.
Die vollständige Funktionalität ist garantiert wenn bei der Kanalbelastungsmessung beide LEDs zum Blinken beginnen.

5.1.2.9. PIC Software Master

Seite

Überblick über das PIC Masterprogramm



Bibliotheken, Konfigurationsbits und globale Variablen

Einbinden der Programmbibliotheken

Programmbibliotheken ersparen dem Programmierer viel Zeit, da in ihnen bereits vorgefertigte und getestete Methoden für häufig gebrauchte Anwendungen vordefiniert sind. Diese Bibliotheken müssen nur in das Programm eingebunden und die jeweilige Methode richtig initialisiert werden.

#include <p18cxxx.h>

Diese Bibliothek enthält die Definitionen der PIC Serie 18FXXX für den C- Compiler.

#include "delays.h"

Diese Bibliothek enthält Methoden für verschiedenste Zeitverzögerungen.

#include <usart.h>

Diese Bibliothek enthält Methoden für die im Chip eingebaute USART- Schnittstelle.

#include <adc.h>

Diese Bibliothek enthält Methoden für die Initialisierung und Bedienung der Analog- Digital-Konverter Hardware.

#include <stdlib.h>

Diese Bibliothek enthält Methoden für die Umwandlung von einem Datentyp in einen anderen (z.B.: Konvertierung einer Integerzahl zu einem String).

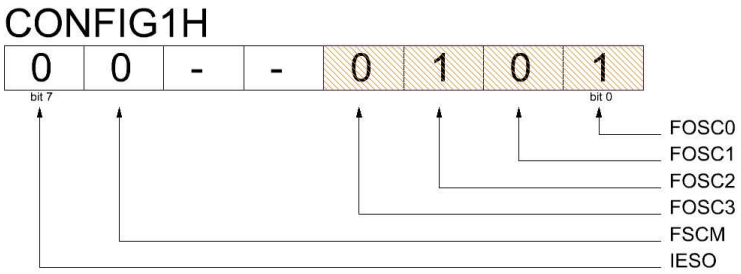
#include <timers.h>

Diese Bibliothek enthält Methoden für die Initialisierung und Bedienung der Timer- Hardware.

Setzen der Konfigurationsbits des PIC's

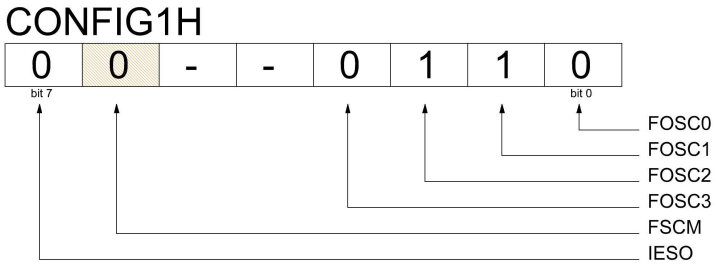
In diesem Programmteil werden die wichtigsten Bits für das Arbeiten mit dem PIC gesetzt.

#pragma config OSC = ECIO



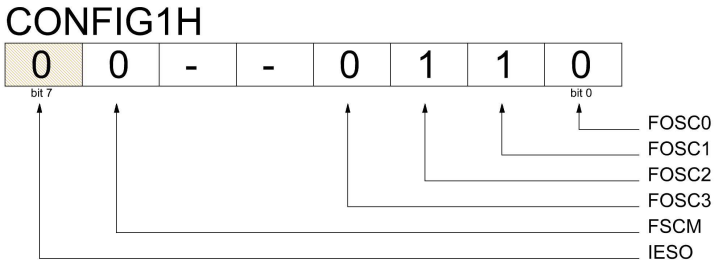
Mit diesem Befehl wird auf den High Speed Modus des PIC umgeschaltet und die interne PLL aktiviert. Die PLL sorgt für eine Vervierfachung der Oszillatorfrequenz.
Der Quarz wird an die PORTs RA6 und RA7 angeschlossen. Die zugehörigen Kondensatoren für die jeweilige Frequenz sind dem Datenblatt des PIC 18F1220/1320 zu entnehmen.

#pragma config FSCM = OFF



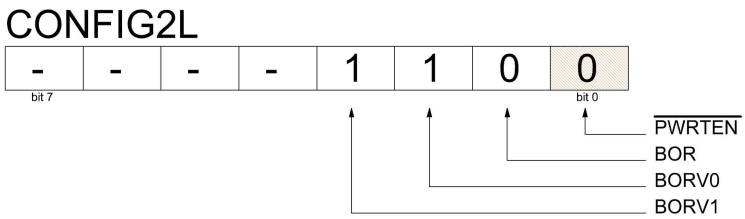
Deaktivierung des „fail safe clock monitor“. Dieser dient dazu, um beim Ausfall des externen Oszillators auf den internen Oszillator umzuschalten und dann mit diesem weiterzuarbeiten.

#pragma config IESO = OFF



Diese Konfigurationseinstellung wird dazu benötigt um ein schnelleres hochfahren des PIC zu ermöglichen. Für unser Projekt wurde dies nicht benötigt, deshalb wurde sie deaktiviert.

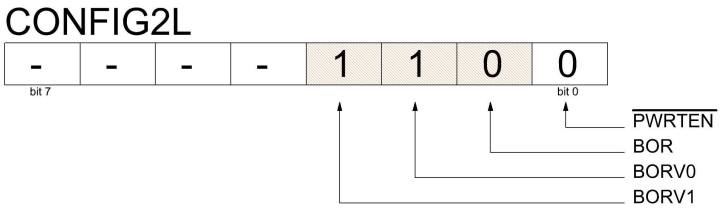
#pragma config PWRT = ON



Der Power – up - Timer dient zur Einschaltverzögerung. Er sorgt dafür, dass der PIC erst bei stabiler Oszillatorfrequenz und stabiler Betriebsspannung zu arbeiten beginnt.

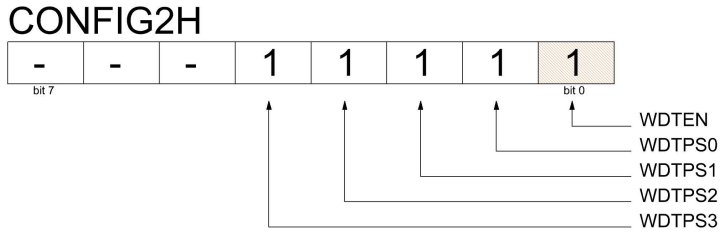
Damit der Power – up – Timer aktiviert wird(PWRT = ON), muss das Bit $\overline{PWRTEN}$ auf 0 gesetzt werden.

#pragma config BOR = OFF



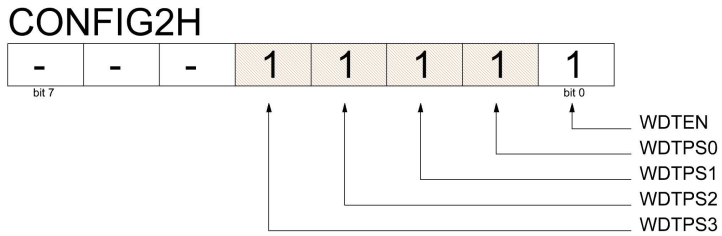
Der Brown-out Reset dient dazu, um den Mikrokontroller, sobald die Versorgungsspannung unter einem definierten Wert gefallen ist, zu reseten.

#pragma config WDT = ON



Der Watchdog Timer dient zur Überprüfung, ob der Prozessor in einer Schleife zu lange hängen geblieben ist. Wenn dieser Fall eintritt, wird der Chip resetet.
Für unser Projekt wurde der Timer aus Stabilitätsgründen aktiviert.

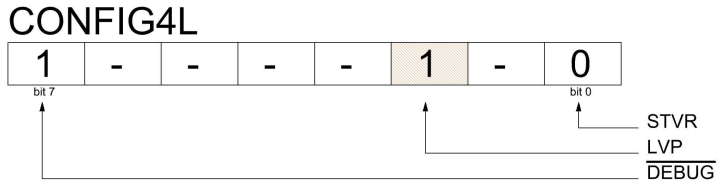
#pragma config WDTPS = 512



Mit diesem Befehl wird die Dauer des Watchdog Timers eingestellt. In unserem Fall wären es ca. 2 Sekunden, da der Timer mit dem internen RC- Oszillator betrieben wird und die minimale Zeit 4ms beträgt.
Daraus ergibt sich folgende Formel für die Dauer des WDT:

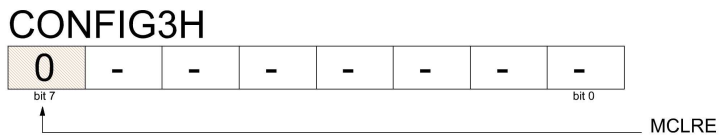
$$4ms \cdot WDTPS = Dauer$$

#pragma config LVP = OFF



Dieser Befehl ermöglicht die Incircuit- Programmierung. Für unser Projekt wurde dies nicht benötigt, da wir den PIC über den PIC Starter Plus programmiert haben.

#pragma config MCLRE = OFF



Der Main Clear Reset, dient zum rücksetzen des Mikrokontrollers über einen Port. Diese Funktion wurde für unsere Anwendung deaktiviert. Da wir diesen PIN als Eingang benötigen.

Definition der globale Variablen

FLAGbits.Is_Aussenstelle

Dieses Bit zeigt an, ob die Datenpakete für den Master bestimmt sind.
Es wird im Unterprogramm Adressencalc (Beschreibung siehe folgende Seiten) gesetzt.
Wenn FLAGbits.Is_Aussenstelle gesetzt wurde, werden zunächst alle Bytes empfangen, danach die Gültigkeit und Checksumme überprüft und anschließend werden die Daten an den PC weitergeleitet.

FLAGbits.RX_Sperren

FLAGbits.RX_Sperren zeigt an, ob der RX Interrupt gesperrt ist oder nicht.
Dieses Bit wird gesetzt, sobald eine Zeitüberschreitung beim Empfangen von zwei Bits eintritt oder ein falsches Byte empfangen wurde.
Es wird beim RX Interrupt abgefragt und beim Interrupt von Timer 2.

unsigned char rx_puffer[13]

Dieses 13 Byte große Array dient dazu, um die Empfangsbytes zwischenzuspeichern.
Das Array wird für die Unterprogramme Adressencheck, Adressencalc, Befehl_Check_Calc, Checksumme_Empfangen und die RX Interruptroutine benötigt.
Der rx_puffer wird in der RX Interrupt Service Routine mit Werten befüllt.

unsigned char byte

Diese Variable ist eine globale Zählervariable und wird bei jeden empfangenen Byte um eins erhöht. Der Zähler hat einen Standardwert von 0 und kann maximal den Wert 12 erreichen.
Diese Variable wird in den Interruptroutinen von RX und Timer2 und im Unterprogramm Befehl_Check_Calc benötigt.
Der Zähler wird zurückgesetzt, sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird, sämtliche Bytes richtig empfangen wurden oder sämtliche Bytes gesendet wurden.

unsigned char bytemax_empfangen

Diese Byte ist eine globale Statusvariable und zeigt an, wie viele Bytes maximal empfangen werden. Sie hat einen Standardwert von 12 und kann minimal den Wert 5 erreichen.
Diese Variable wird in den Interruptroutinen von RX und Timer2 und in den Unterprogrammen Befehl_Check_Calc und Checksumme_Empfangen benötigt.
Sie wird im Unterprogramm Befehl_Check_Calc gesetzt und sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird, sämtliche Bytes richtig empfangen wurden oder sämtliche Bytes gesendet wurden, wieder auf den Wert 12 zurückgesetzt.

unsigned char tx_puffer[13]

Dieses 13 Byte große Array dient dazu, um die Sendebytes zwischenzuspeichern.
Das Array wird für die Unterprogramme Adressencalc, Checksumme_Senden, Standardantwort und die RX und TX Interruptroutine benötigt.
Der tx_puffer wird in den Unterprogrammen Adressencalc und Standardantwort mit Werten befüllt.

unsigned char sendecounter

Diese Variable ist eine globale Zählervariable und wird bei jedem gesendeten Byte um eins erhöht. Der Zähler hat einen Standardwert von 0 und kann maximal den Wert 12 erreichen.
Diese Variable wird in der TX Interruptroutine benötigt.
Der Zähler wird zurückgesetzt, sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird oder sämtliche Bytes gesendet wurden.

unsigned char bytemax_senden

Diese Byte ist eine globale Statusvariable und zeigt an, wie viele Bytes maximal gesendet werden. Sie hat einen Standardwert von 12 und kann minimal den Wert 5 erreichen.
Diese Variable wird in den RX und TX Interruptroutinen und in den Unterprogrammen Checksumme_Senden und Standardantwort benötigt.
Sie wird im Unterprogramm Standardantwort und in der RX Interruptroutine gesetzt. Sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird oder sämtliche Bytes gesendet wurden wird sie wieder auf den Wert 12 zurückgesetzt.

LED:

Immer wenn in der Masterprogrammdokumentation die Bezeichnung LED verwendet wird, bezieht sich dieser Ausdruck auf die rote LED (LED102), welche zur Funkempfangsanzeige dient. Diese Leuchtdiode hängt am Port RB7 des Mikrokontrollers.

Programmcode:

Sämtlicher Programmcode wurde zwecks besserer Übersicht weggelassen.
Der vollständige Programmcode befindet sich auf der Diplomarbeit - DVD im C – File „PIC\Master\Master_V3.0.c“.
Die detaillierte Programmdokumentation mit zugehörigem Programmcode ist auf der Diplomarbeit – DVD in der Datei “ PIC\Master\PIC Masterprogrammdokumentation_V1.2.doc” zu finden.

Unterprogramme

Überprüfung der Adresse auf ihre Gültigkeit

char Adressencheck(void)

Dient zur Überprüfung, ob die vier empfangenen Adressbytes dem Muster 1XXXXXXX (B) entsprechen.
Diese Funktion liefert im Fehlerfall den Wert 0 zurück.

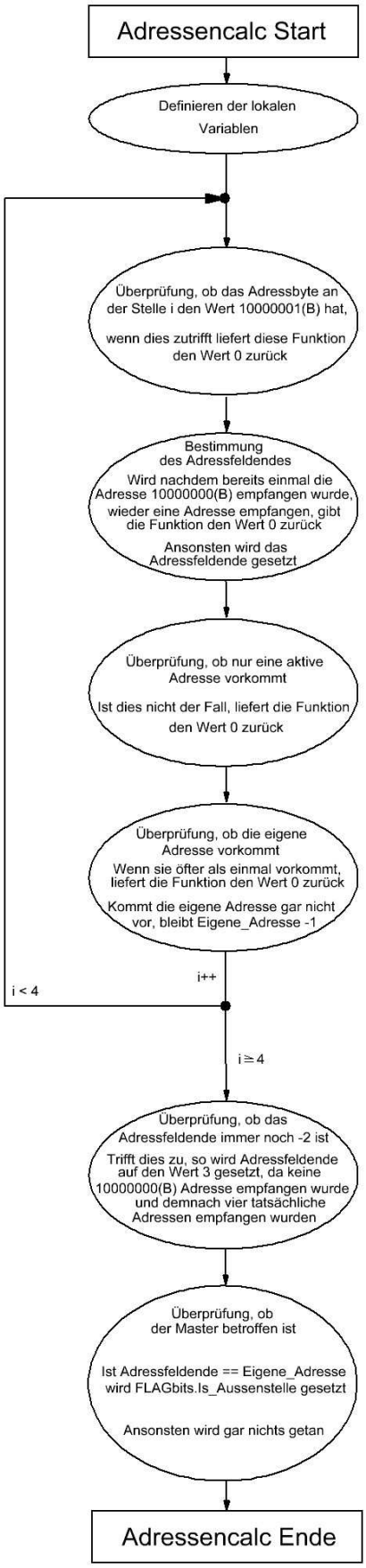
Adressenauswertung

char Adressencalc(void)

Lokale Variablen:

- char Adressfeldende
  Gibt an wie viele der vier empfangen Adressbytes tatsächliche Slaveadressen sind und dem Muster 1XXXXXXX (B) entsprechen. Die Adresse 10000000 (B) ist in diesem Fall eine Spezialadresse und wird nicht als gültige Adresse erkannt.
  Die Variable Adressfeldende wird dann auf den Wert der tatsächlichen Adressen gesetzt.
  Sie kann im Bereich von -2 bis 3 liegen und hat einen Standardwert von -2.
- char Aktive_Adresse
  Diese Variable dient zur Überprüfung, ob tatsächlich immer nur eine Adresse aktiv ist.
  Eine aktive Adresse wird an der Form 1XXXXXX1 (B) erkannt.
  Sie liegt im Bereich von -1 und 3 und hat einen Standardwert von -1.
- char Eigene_Adresse
  Diese Variable zeigt an, welches Byte der Adressbytes die eigene Adresse enthält.
  Sie liegt im Bereich von -1 bis 3 und hat einen Standardwert von -1.
- char i
  Ist eine lokale Zählvariable. Sie dient zum Berechnen der neuen Adresse und wird von 0 bis 3 hoch gezählt.
  Diese Variable wird standardmäßig mit 0 initialisiert.

Prinzipieller Aufbau:
Adressenauswertung



Beschreibung:
Im Fall eines Fehlers liefert diese Funktion den Wert 0 zurück, andernfalls den Wert 1.
Diese Funktion überprüft im Masterprogramm hauptsächlich, ob die Adressen dem Protokoll entsprechen und die Datenpakete für den Master bestimmt sind. Wenn die Daten nicht für den Master bestimmt sind, tut der Master nichts weiter als auf für ihn bestimmte Daten zu warten.

Gültigkeitsüberprüfung eines Befehls

char Befehl_Check_Calc(void)

Dient zur Überprüfung, ob das fünfte empfangene Byte (Befehlsbyte) dem Muster $0XXXXXXX_{(B)}$ entspricht. Weiters setzt das Unterprogramm die Variable bytemax_empfangen auf den Zahlenwert, der den drei niederwertigsten Bits des Befehlsbyte plus sechs entspricht.
Diese Funktion liefert im Fehlerfall den Wert 0 zurück.

Checksummenberechnung fürs Senden und Empfangen

Lokale Variablen:

- char checksumme
  Zwischenvariable zum Bilden der Checksumme. Deren Wert wird abschließend von der Funktion zurückgegeben.
- char i
  Lokale Zählvariable für die Checksummenbildung.
  Sie kann Werte von 0 bis 11 annehmen und wird standardmäßig mit 0 initialisiert.

char Checksumme_Empfangen(void)

Diese Funktion verknüpft alle empfangenen Bytes, bis auf das letzte Byte, durch eine Exklusiv-Oder Operation miteinander.
Das Endergebnis wird von dieser Funktion zurückgegeben.

char Checksumme_Senden(void)

Diese Funktion verknüpft alle zu sendenden Bytes, bis auf das letzte Byte für die Checksumme, durch eine Exklusiv- Oder Operation miteinander.
Das Endergebnis wird von dieser Funktion zurückgegeben.

Definition der Standardantwort

void Standardantwort(void)

Dieses Unterprogramm dient zum Senden der Antwort „Bin da ohne ID!“.
Es setzt das fünfte Byte des Sendearrays (tx_puffer[4]) auf den Wert $01101000_{(B)}$, was der Antwort auf den Befehl „Bist du da? ohne ID“ entspricht.
Die Variable bytemax_senden wird auf den Wert 6 gesetzt und das tx_puffer Byte für die Checksumme wird mit der zugehörigen Checksumme beschrieben.
Außerdem wird danach die Senderoutine gestartet.

High Priority Interrupt Service Routine

void high_isr (void)

Lokale Variable:

- char Empfang_OK
  Ist eine lokale FLAG die anzeigt, ob beim Empfang ein Fehler aufgetreten ist oder sämtliche Bytes korrekt empfangen wurden.
  Im Fehlerfall wird diese FLAG auf 0 gesetzt.
  Im Falle, dass alle Bytes empfangen wurden wird diese FLAG auf -1 gesetzt.

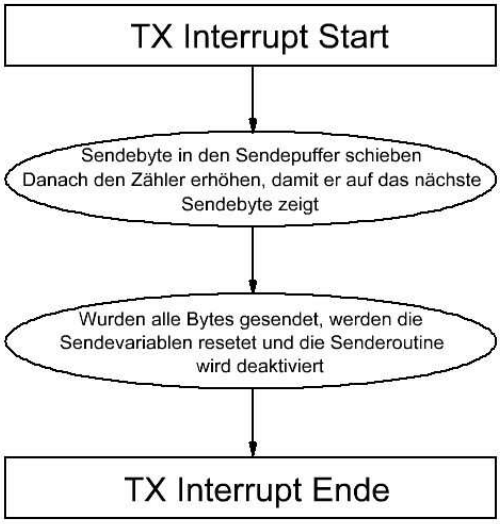
Begriffserklärung Timeout:

Ein Timeout tritt immer dann ein, wenn vom Empfang eines Bytes bis zum nächsten mehr als 2ms vergangen sind oder ein falsches Byte empfangen wurde.
Das Timeout dient zur schnelleren Synchronisation nachdem ein Fehler aufgetreten ist.

Prinzipieller Aufbau:

Verarbeitung eines Empfang(RX) Interrupts

Verarbeitung eines Sende(TX) Interrupts



Beschreibung:
Mit dieser ISR erfolgt die Koordination zwischen der Empfangs- und Sendefunktion.
Weiters erfolgt in dieser Routine die Überprüfung der empfangenen Bytes.

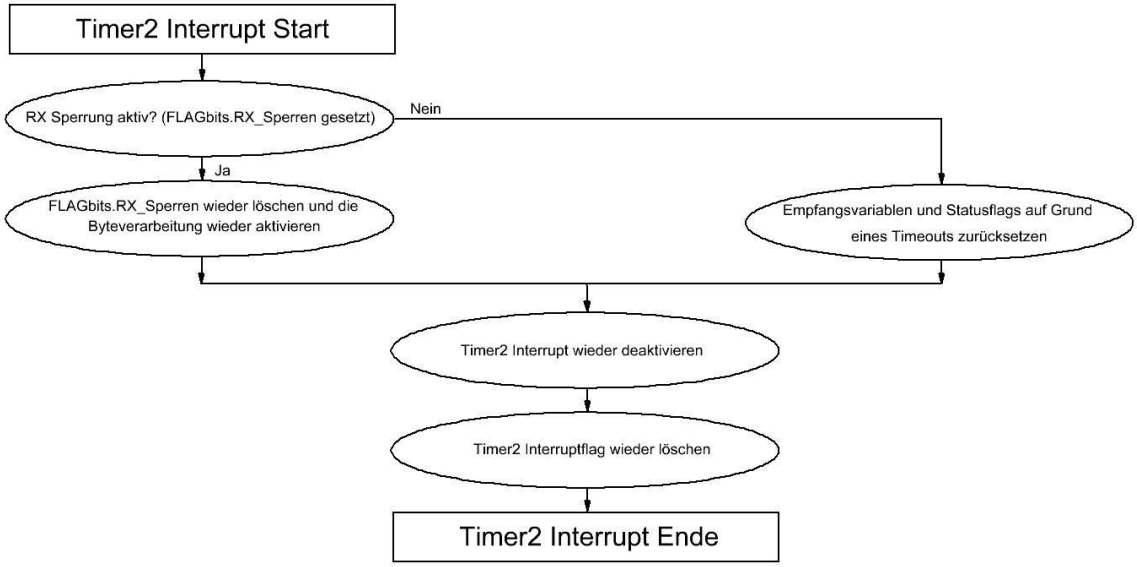
Low Priority Interrupt Service Routine

void low_isr (void)

Prinzipieller Aufbau:

Verarbeitung eines Timer 0 Interrupts
Im Timer 0 Interrupt wird die Empfangsstatus LED im 500ms Takt ein - und ausgeschaltet.

Verarbeitung eines Timer 2 Interrupts



Beschreibung:

In dieser Routine erfolgt die Verwaltung der Timer für die folgenden Aufgaben:

- Ausschalten der Status LED (Timer 0)
- Entsperren bzw. sperren der Empfangsroutine bei einem Timeout (Timer 2)

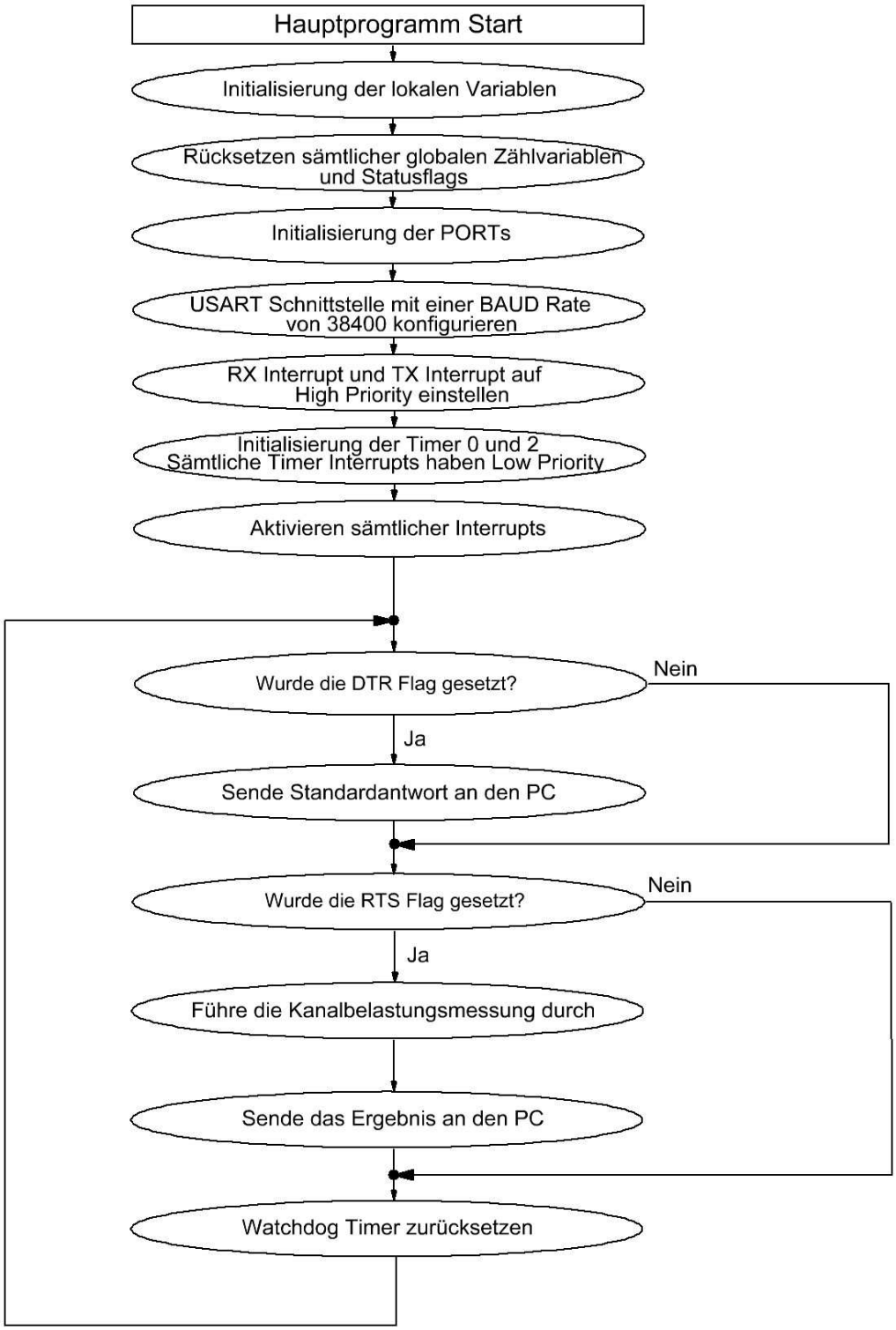
Hauptprogramm

Lokale Variable:

- int resultat

  Wird benötigt um das ADC Ergebnis zu Speichern

Prinzipieller Aufbau:

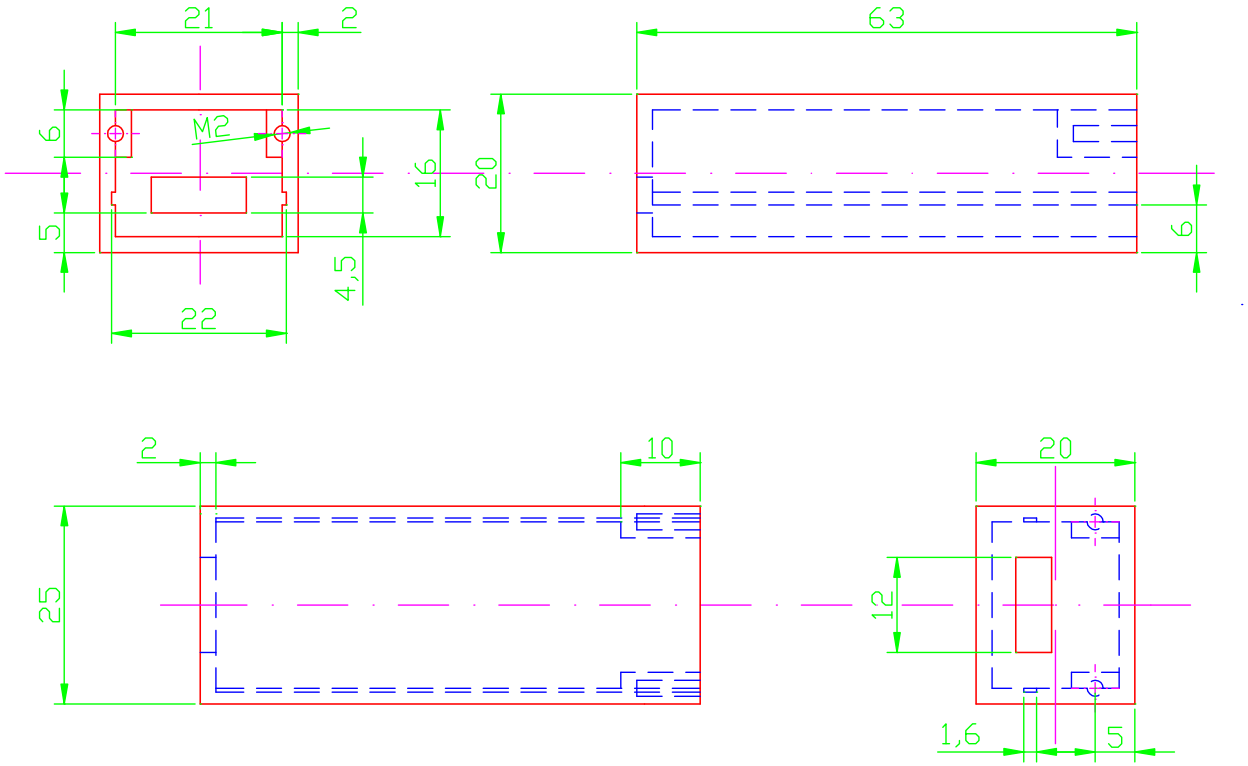


Beschreibung:
Das Hauptprogramm dient im Wesentlichen dazu, um auf direkte Befehle des PCs an den Master zu warten.
Sobald der PC die DTR Leitung kurzfristig setzt, wird dem Master signalisiert, dass er dem PC die Standardantwort übermitteln soll. Dies dient zur Überprüfung, ob der Master an den PC angeschlossen wurde.
Sobald der PC die RTS Leitung kurzfristig setzt, wird dem Master signalisiert, dass er eine Kanalbelastungsmessung durchführen soll. Das Kanalbelastungsmessungsergebnis wird nach einer erfolgreichen Messung direkt an den Computer weitergeleitet.
Das Hauptprogramm wird erst durch Verlust der Betriebsspannung beendet.

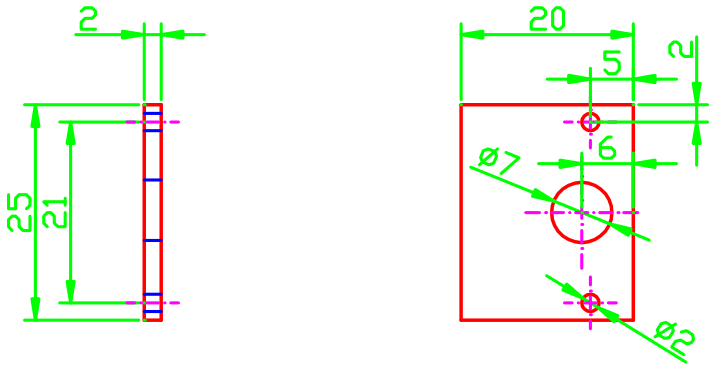
5.1.2.10. Gehäuse Master

Sämtliche Abbildungen sind nicht im Maßstab! Alle Abmessungen sind in mm!

Längsteil (Material: PMMA)



Oberer und unterer Deckel (Material: PMMA)



Die Fertigung des Deckels erfolgt für die Rückwand wie hier eingezeichnet wurde. Der Deckel zum Durchstecken des USB – Anschlusses erfolgt ohne die Löcher. Dieser zweite Teil wird mit dem Längsteil verklebt. Die Ausnehmung für den USB Anschluss sollte erst im geklebten Zustand gefertigt werden.

Stückliste des Master Gehäuses

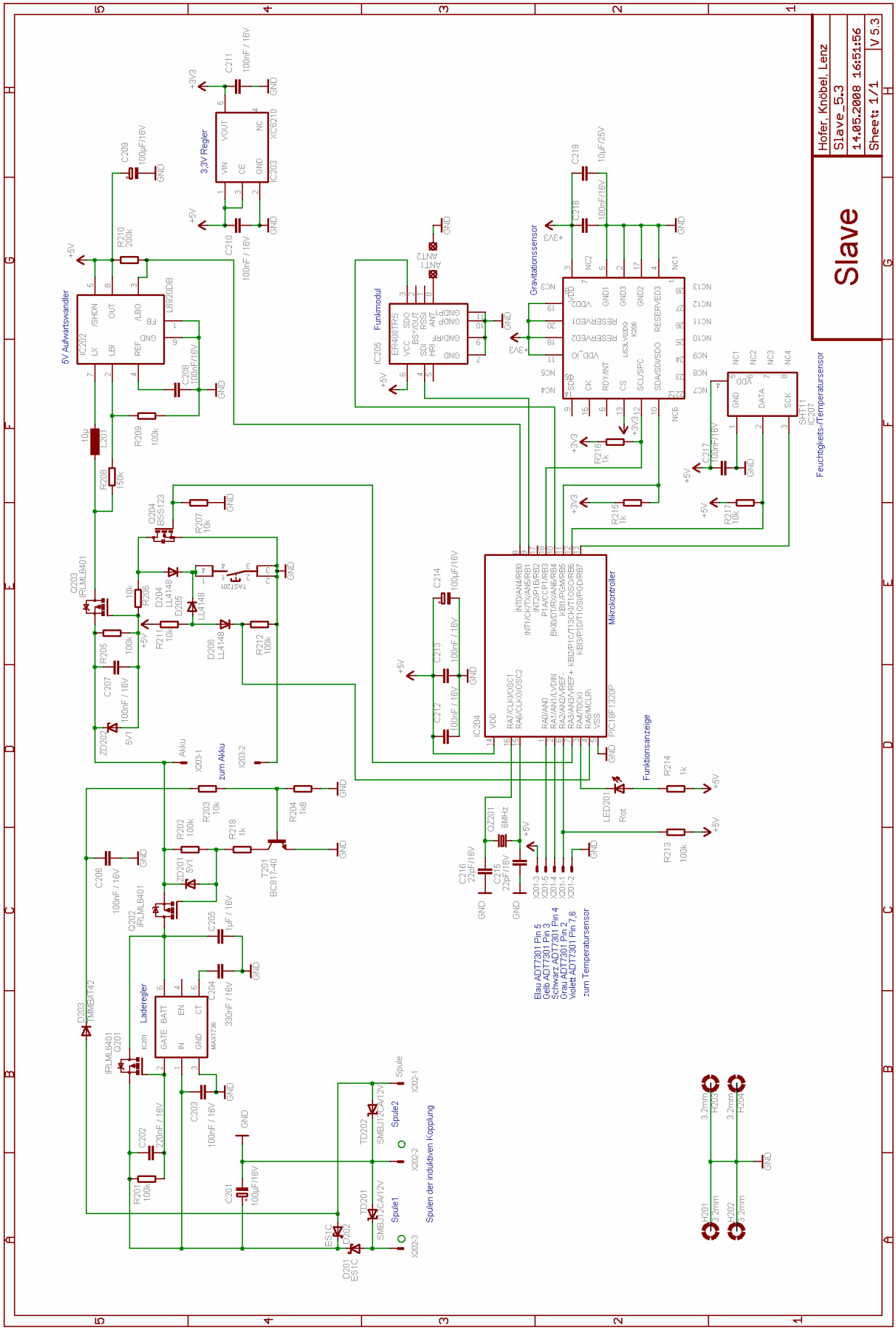
Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	2	Seitenwand	PMMA	63x25x2
2	2	Seitenwand	PMMA	63x20x2
3	2	Deckel	PMMA	20x25x2
4	1	Deckel	PMMA	16x17x2
5	2	Bohrverstärkung	PMMA	10x6x2
6	2	Senkkopfschrauben		M2x10

5.1.3. Slave

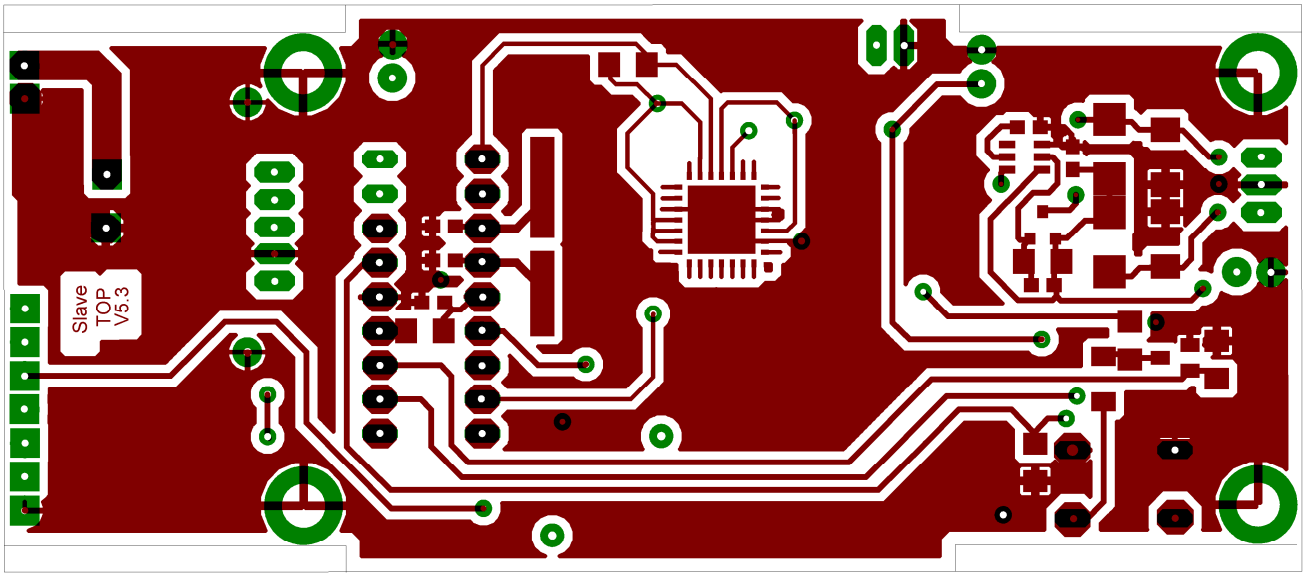
Der fertige Slave sieht von außen wie folgt aus:

Für den Slave wurde eine extra Temperaturabnehmerplatine gefertigt, um einen besseren thermischen Kontakt für die punktuelle Temperaturmessung zu ermöglichen.
Diese Vorrichtung sieht wie folgt aus:

5.1.3.1. Schaltplan Slave

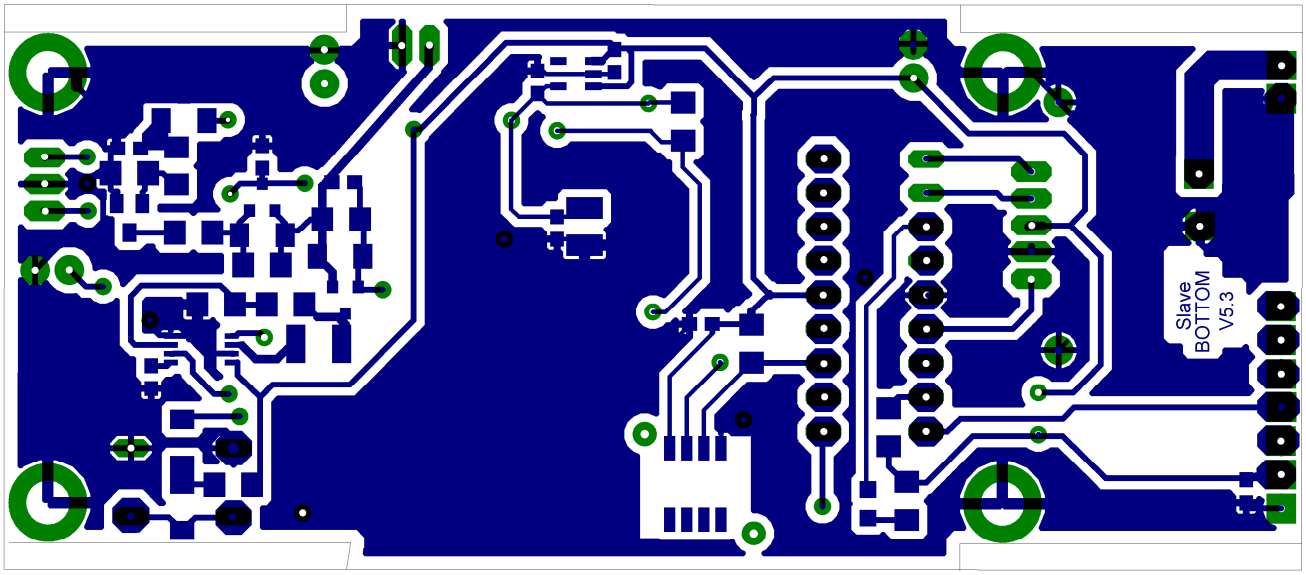


5.1.3.2. Layout Slave Top Layer



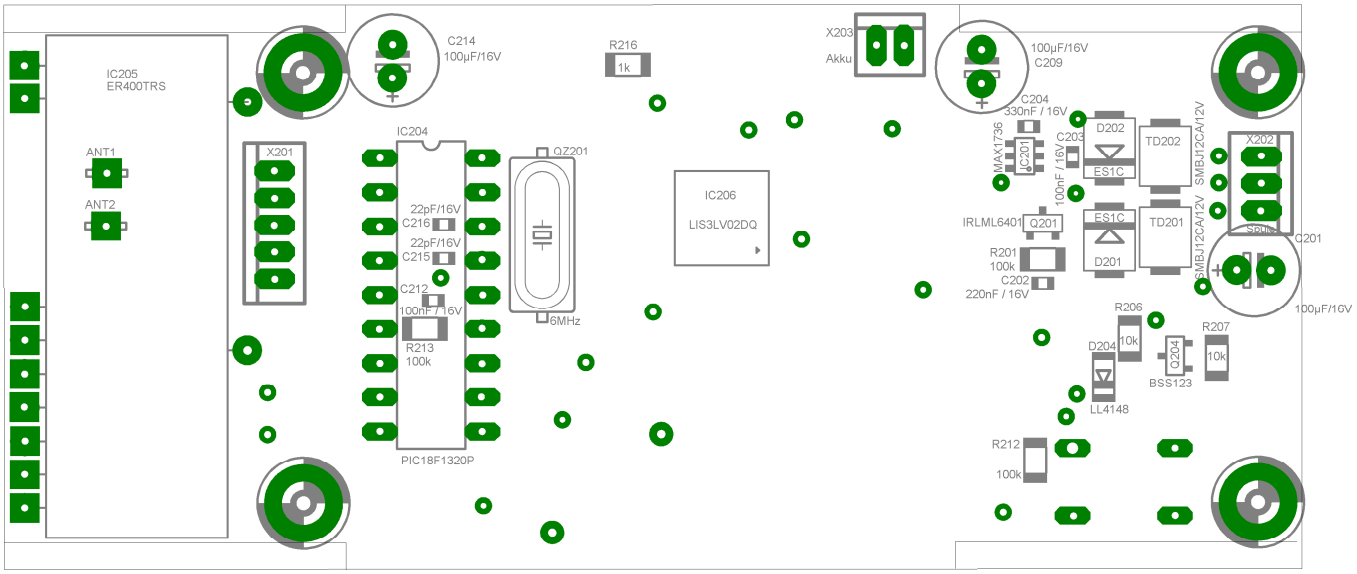
Top View
Abbildung nicht im Maßstab
Abmessungen: 96,5mm x 42mm

5.1.3.3. Layout Slave Bottom Layer



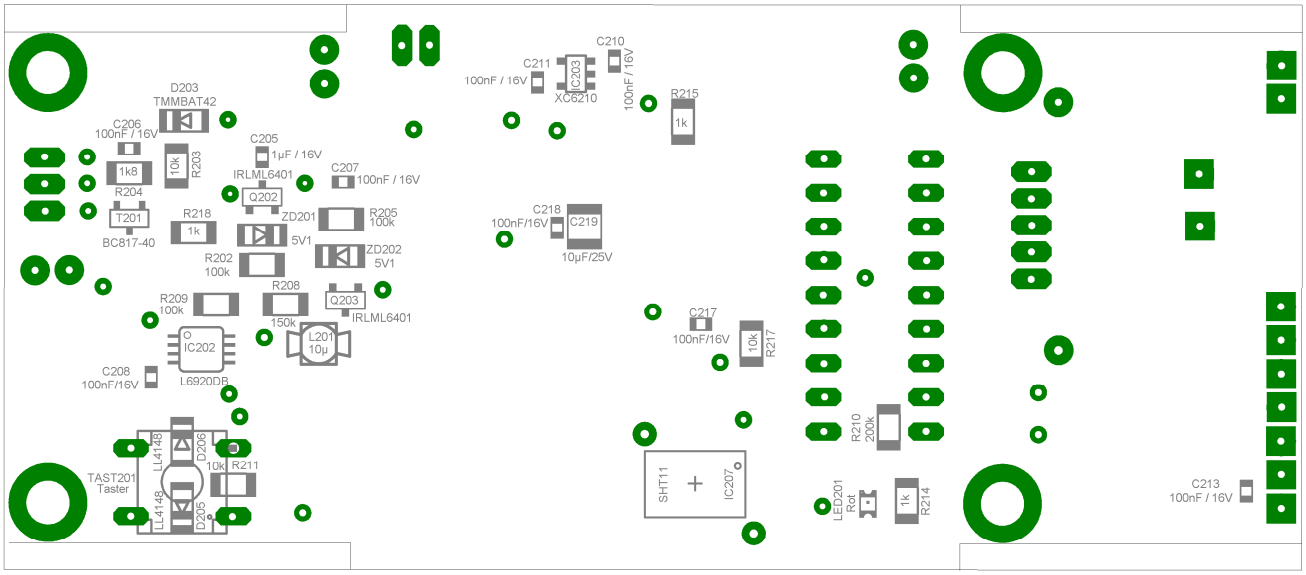
Bottom View
Abbildung nicht im Maßstab
Abmessungen: 96,5mm x 42mm

5.1.3.4. Bestückungsplan Slave Top Place



Top View
Abbildung nicht im Maßstab
Abmessungen: 96,5mm x 42mm

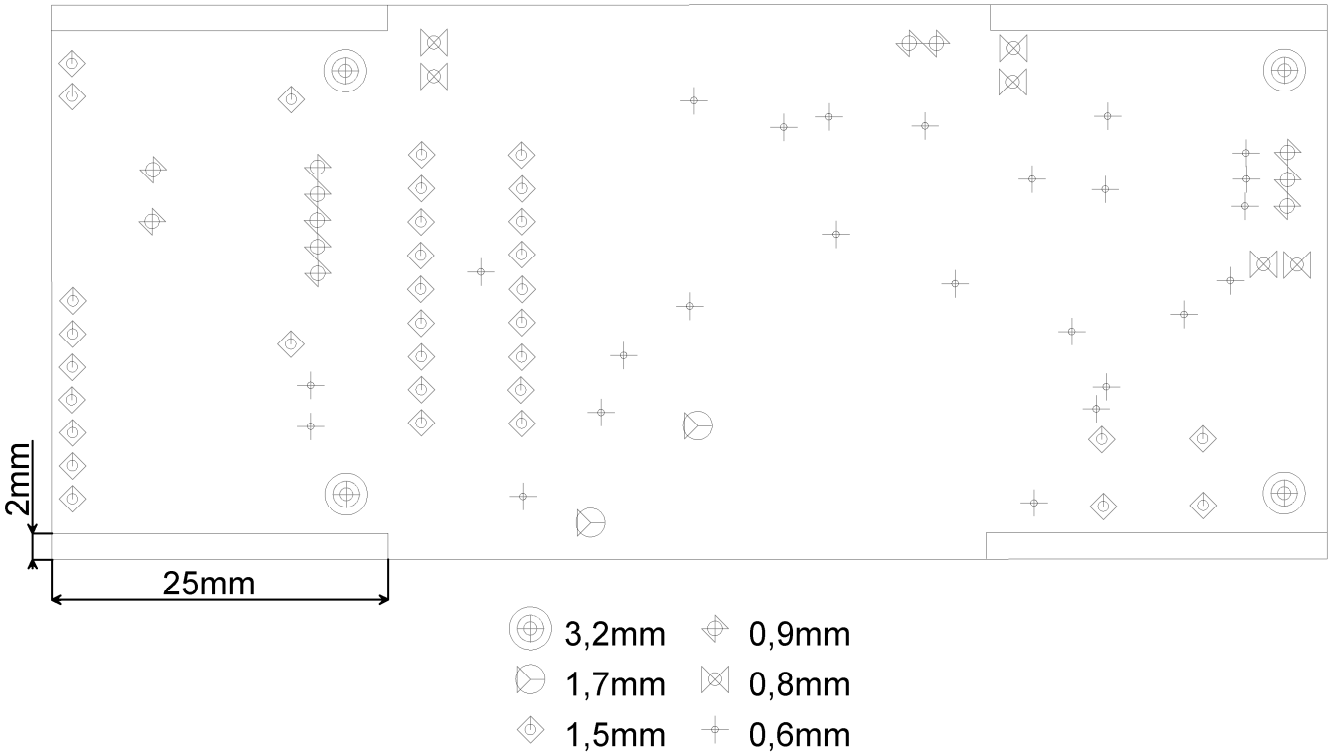
5.1.3.5. Bestückungsplan Slave Bottom Place



Bottom View
Abbildung nicht im Maßstab
Abmessungen: 96,5mm x 42mm

Achtung: Mit der Bestückung der Sensoren (IC205 und IC206), dem Funkmodul und dem PIC (IC204) sollte bis nach der Inbetriebnahme gewartet werden.
Nähere Informationen sind dem Kapitel „5.1.3.15 Inbetriebnahme des Slaves“ zu entnehmen.

5.1.3.6. Bohrplan Slave



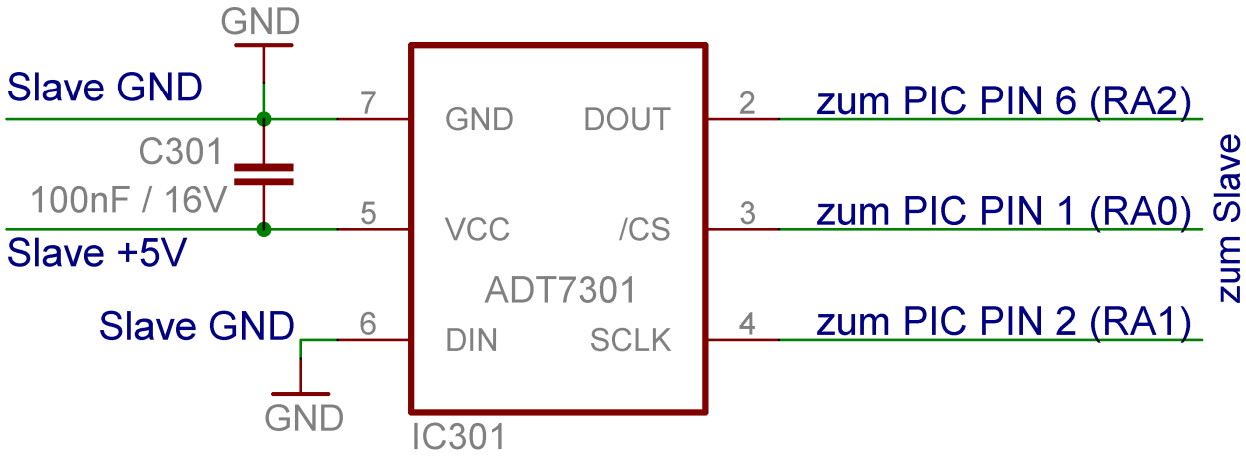
Top View
Abbildung nicht im Maßstab
Alle Ausnehmungen haben dieselben Maße
Abmessungen: 96,5mm x 42mm

5.1.3.7. Stückliste Slave

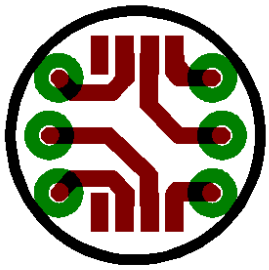
Nr.	Stk.	Bez.	Typ	Wert	Bemerkung
1	1	Q204	N-Kanal MOSFET	BSS123	SOT23
2	3	Q201, Q202, Q203	P -Kanal MOSFET	IRLML6401	SOT23
3	1	T201	NPN Bipolartransistor	BC817 - 40	SOT23
4	2	TD201, TD202	Transil Transorb Diode	SMBJ12CA / 12V	SMB
5	1	LED201	Leuchtdiode Rot	Stanley BR1111C	0805
6	2	ZD201, ZD202	Zenerdiode	BZV5V1	SOD80C
7	3	D204, D205, D206	Diode	LL4148	Minimelf
8	1	D203	Schottky Diode	TMMBAT42	Minimelf
9	2	D201, D202	Schottky Diode	ES1C	SMA
10	1	IC203	Spannungsregler 3,3V	XC6210	SOT23-5
11	1	IC202	Aufwärtswandler	L6920DB	MSOP8
12	1	IC201	Laderegler	MAX1736	SOT23-6
13	1	IC205	Transceiver	ER400TRS	eigenes Package
14	1	IC204	Mikrokontroller	PIC18F1320 I/P	DIL18

Nr.	Stk.	Bez.	Typ	Wert	Bemerkung
15	1	IC206	Beschleunigungssensor	LIS3LV02DQ	QFPN
16	1	IC207	Temperatur- / Feuchtigkeitssensor	SHT11	eigenes Package
17	1	L201	Spule	LQH32CN100K33L	1210
18	1	C219	Keramikkondensator	10µF +80% -20% / 25V	1210
19	1	C205	Keramikkondensator	1µF ±10% / 16V	0603
20	1	C204	Keramikkondensator	330nF ±10% / 16V	0603
21	1	C202	Keramikkondensator	220nF ±10% / 16V	0603
22	10	C203, C206, C207, C208, C210, C211, C212, C213, C217, C218	Keramikkondensator	100nF ±10% / 16V	0603
23	2	C215, C216	Keramikkondensator	22pF ±5% / 200V	0603
24	3	C201, C209, C214	Elektrolytkondensator	100µF ±20% / 16V	RM2(6,3 x 2,5) liegend
25	1	R210	Widerstand	200kΩ ±1% / 1/8W	1206
26	1	R208	Widerstand	150kΩ ±1% / 1/8W	1206
27	6	R201, R202, R205, R209, R212, R213	Widerstand	100kΩ ±1% / 1/8W	1206
28	5	R203, R206, R207, R211, R217	Widerstand	10kΩ ±1% / 1/8W	1206
29	1	R204	Widerstand	1,8kΩ ±1% / 1/8W	1206
30	4	R214, R215, R216, R218	Widerstand	1kΩ ±1% / 1/8W	1206
31	1	QZ201	Quarz	6MHz Rakon	SM49
32	1	X203	Stiftleiste stehend 2-polig	2-polig	RM2(5,9 x 2)
33	1	X203	Crimpleergehäuse 2-polig	2-polig	RM2(5,8 x 2)
34	1	X202	Stiftleiste stehend 3-polig	3-polig	RM2(7,9 x 2)
35	1	X202	Crimpleergehäuse 3-polig	3-polig	RM2(7,8 x 2)
36	1	X201	Stiftleiste stehend 5-polig	5-polig	RM2(11,9 x 2)
37	1	X201	Crimpleergehäuse 5-polig	5-polig	RM2(11,8 x 2)
38	10		Crimpkontakte	Kontakte	0,05mm² - 0,22mm²

5.1.3.8. Schaltplan Temperaturabnehmer

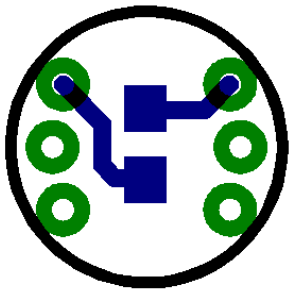


5.1.3.9. Layout Temperaturabnehmer Top Layer



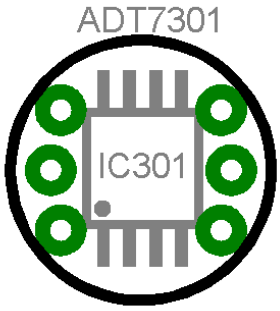
Top View
Abbildung nicht im Maßstab
Abmessungen: Ø 6,5mm

5.1.3.10. Layout Temperaturabnehmer Bottom Layer



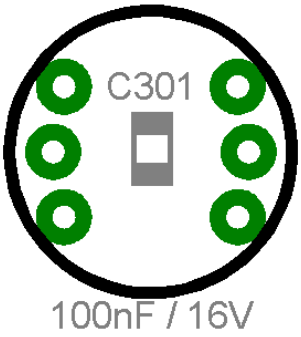
Bottom View
Abbildung nicht im Maßstab
Abmessungen: Ø 6,5mm

5.1.3.11. Bestückungsplan Temperaturabnehmer Top Place



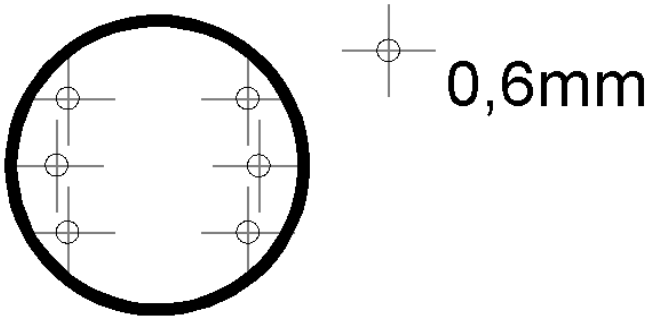
Top View
Abbildung nicht im Maßstab
Abmessungen: Ø 6,5mm

5.1.3.12. Bestückungsplan Temperaturabnehmer Bottom Place



Bottom View
Abbildung nicht im Maßstab
Abmessungen: Ø 6,5mm

5.1.3.13. Bohrplan Temperaturabnehmer



Top View
Abbildung nicht im Maßstab
Abmessungen: Ø 6,5mm

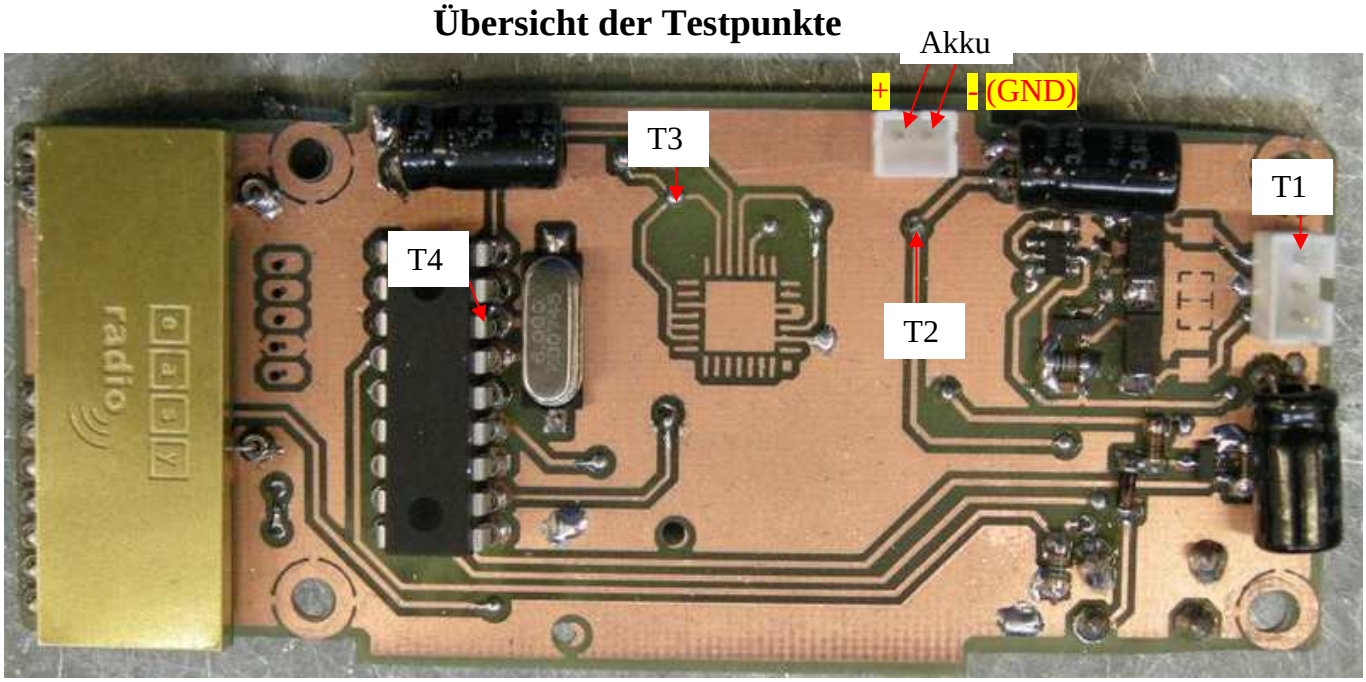
5.1.3.14. Stückliste Temperaturabnehmer

Nr.	Stk.	Bez.	Typ	Wert	Bemerkung
1	1	IC301	Temperatursensor	ADT7301	MSOP8
2	1	C301	Keramikkondensator	100nF ±10% / 16V	0603

5.1.3.15. Inbetriebnahme des Slaves

Beim Testen der Slave - Platine ist es vorteilhaft wenn die Sensoren (IC205 und IC206) erst eingelötet werden wenn die Versorgungsspannungen kontrolliert wurden.

Übersicht der Testpunkte



Testen der Versorgungsspannungen

Zuerst muss kontrolliert werden, ob die Versorgungsspannungen vorhanden sind.
Für diesen Test sollten die Sensoren (IC205 und IC206) noch nicht eingelötet sein und das Funkmodul und der PIC (IC204) nicht bestückt sein.

Die Schaltung wird im nächsten Schritt mit einem Labornetzteil versorgt.
Dieses muss auf **3.5V** und **300mA** Strombegrenzung eingestellt werden. Die Versorgungsspannung wird über die Buchse für den Akku zugeführt.
Solange der Taster nicht betätigt wird, sollte kein Strom in die Schaltung fließen.
Sobald der Taster gedrückt wird, wird eine Stromaufnahme von rund 1mA erwartet. Bei gedrücktem Taster muss weiters an dem Testpunkt T2 eine Spannung von 5V und an dem Testpunkt T3 eine Spannung von 3.3V gemessen werden.

ACHTUNG: Zur Strommessung wird ein Messbereich mit einem Innenwiderstand von $\leq 1\Omega$ empfohlen. Andernfalls kann es zu Fehlmessungen kommen!

Testen der Ladeschaltung

Im nächsten Schritt muss die Ladeschaltung des Akkus überprüft werden.

Anstelle eines echten Akkus muss für diesen Test ein Elektrolytkondensator verwendet werden. Dessen Kapazität sollte im Bereich von 330µF liegen und eine Spannungsfestigkeit von mindestens 16V aufweisen.

Der Kondensator muss mit der richtigen Polarität an den Akkustecker angeschlossen werden.
Weiters wird ein Labornetzgerät mit einer Strombegrenzung von 50mA benötigt.
Dieses wird an dem Testpunkt T1 angeschlossen und gegen GND verbunden.

Wird nun langsam die Spannung des Labornetzgerätes erhöht, sollte sich folgendes Verhalten zeigen:

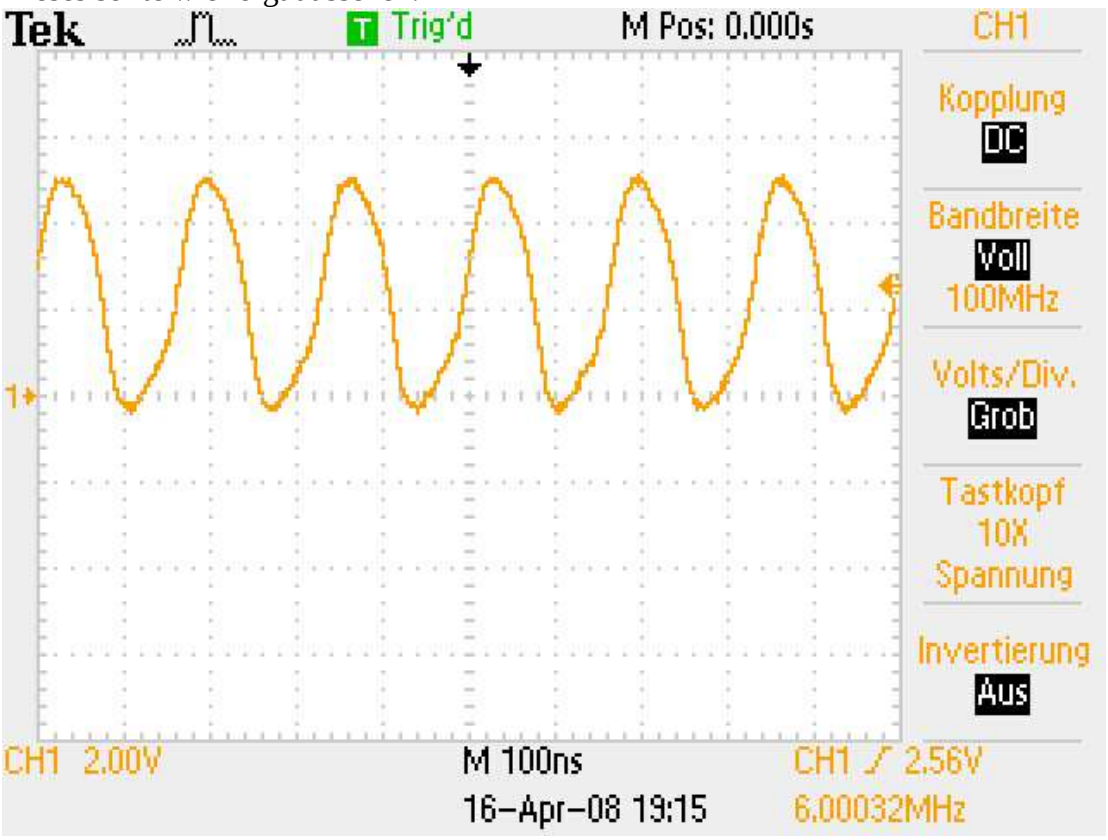
- Bei einer Spannung von 2.8V am Testpunkt T1 sollte die Spannung des Kondensators langsam ansteigen.
- Ab einer Spannung von 5.2V am Testpunkt T1 sollte die Kondensatorspannung 4.5V erreicht haben und nicht weiter ansteigen.
- Ab einer Spannung von 15V am Testpunkt T1 sollte das Labornetzgerät in Strombegrenzung gehen.

Testen der Stromaufnahme

Jetzt können die Sensoren bestückt werden.

Die Schaltung wird nun mit einem Labornetzgerät versorgt.
Dieses muss auf **3.5V** und **300mA** Strombegrenzung eingestellt werden.
Die Versorgungsspannung wird über die Buchse für den Akku zugeführt.
Die Stromaufnahme sollte bei gedrücktem Taster 2mA betragen.

Wenn noch das Funkmodul und der PIC bestückt werden, wird eine Stromaufnahme von 90mA erwartet. Mit bestücktem PIC sollte die Selbsthaltung aktiviert werden. Der Taster kann nun losgelassen werden. Weiters sollte die rote LED zum Blinken beginnen.
Am Testpunkt T4 kann das Clock – Signal von 6MHz des PICs gemessen werden.
Dieses sollte wie folgt aussehen:



Funktest

Sobald alle Tests durchgeführt wurden, kann eine Antenne angelötet werden. Mit dieser kann getestet werden, ob der Slave über Funk erreichbar ist und ob alle bestückten Sensoren erkannt werden.

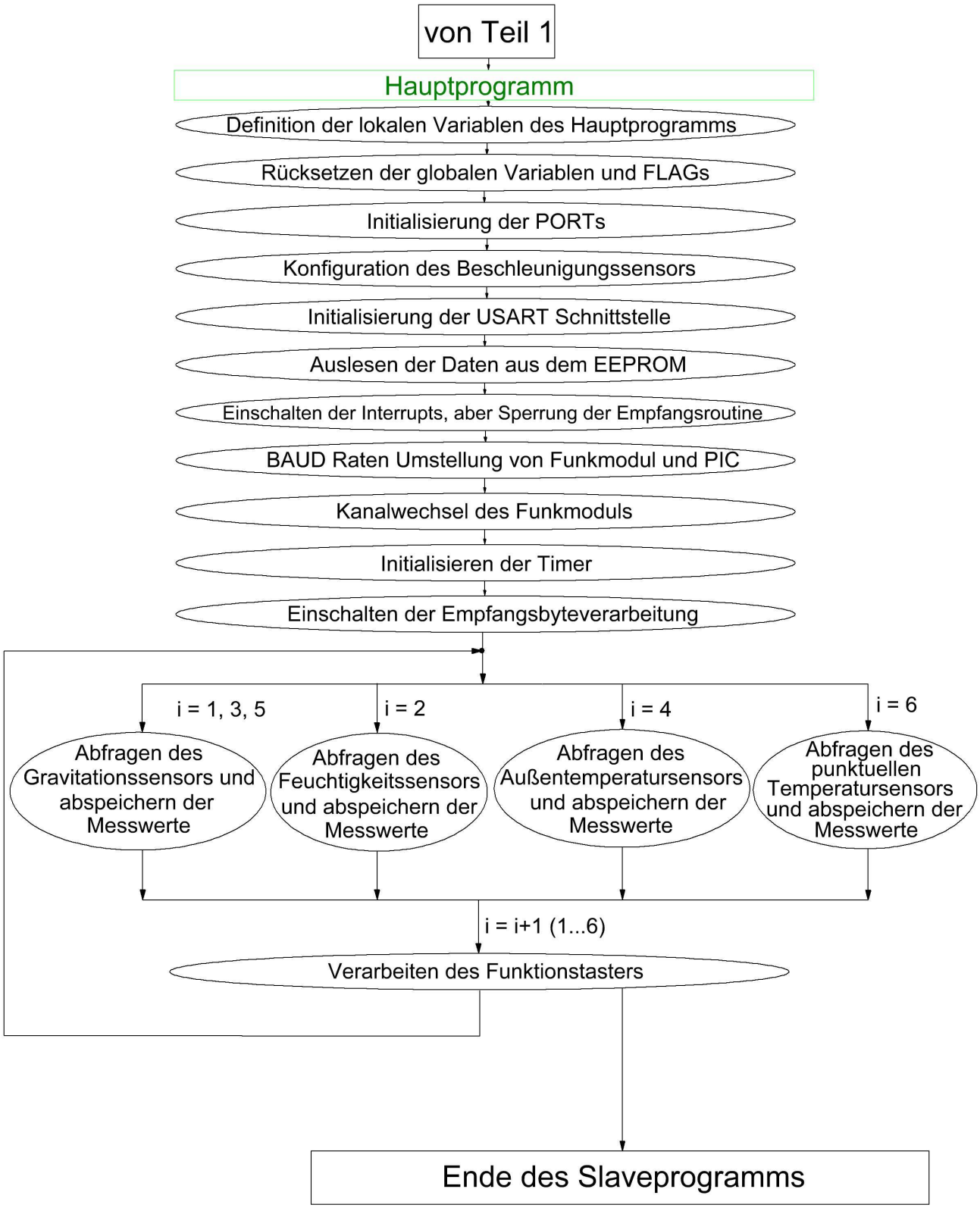
5.1.3.16. PIC Software Slave

PIC Software Slave Beschreibung

Seite

Seite

Überblick über das PIC Slaveprogramm


von Teil 1
Hauptprogramm
Definition der lokalen Variablen des Hauptprogramms
Rücksetzen der globalen Variablen und FLAGs
Initialisierung der PORTs
Konfiguration des Beschleunigungssensors
Initialisierung der USART Schnittstelle
Auslesen der Daten aus dem EEPROM
Einschalten der Interrupts, aber Sperrung der Empfangsroutine
BAUD Raten Umstellung von Funkmodul und PIC
Kanalwechsel des Funkmoduls
Initialisieren der Timer
Einschalten der Empfangsbyteverarbeitung
i = 1, 3, 5
i = 2
i = 4
i = 6
Abfragen des Gravitationssensors und abspeichern der Messwerte
Abfragen des Feuchtigkeitssensors und abspeichern der Messwerte
Abfragen des Außentemperatursensors und abspeichern der Messwerte
Abfragen des punktuellen Temperatursensors und abspeichern der Messwerte
i = i+1 (1...6)
Verarbeiten des Funktionstasters
Ende des Slaveprogramms

Bibliotheken, Konfigurationsbits und globale Variablen

Einbinden der Programmbibliotheken

Programmbibliotheken ersparen dem Programmierer viel Zeit, da in ihnen bereits vorgefertigte und getestete Methoden für häufig gebrauchte Anwendungen vordefiniert sind. Diese Bibliotheken müssen nur in das Programm eingebunden und die jeweilige Methode richtig initialisiert werden.

#include <p18cxxx.h>

Diese Bibliothek enthält die Definitionen der PIC Serie 18FXXX für den C- Compiler.

#include "delays.h"

Diese Bibliothek enthält Methoden für verschiedenste Zeitverzögerungen.

#include <usart.h>

Diese Bibliothek enthält Methoden für die im Chip eingebaute USART- Schnittstelle.

#include <adc.h>

Diese Bibliothek enthält Methoden für die Initialisierung und Bedienung der Analog- Digital-Konverter Hardware.

#include <stdlib.h>

Diese Bibliothek enthält Methoden für die Umwandlung von einem Datentyp in einen anderen (z.B.: Konvertierung einer Integerzahl zu einem String).

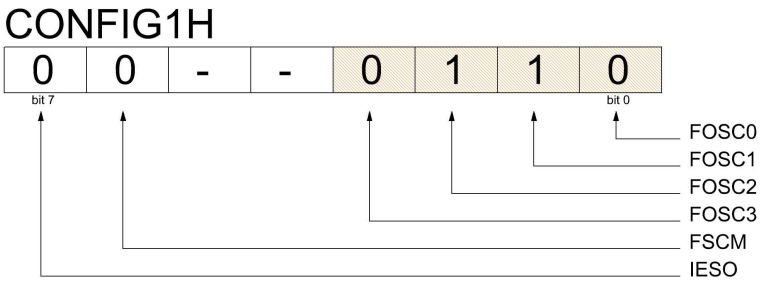
#include <timers.h>

Diese Bibliothek enthält Methoden für die Initialisierung und Bedienung der Timer- Hardware.

Setzen der Konfigurationsbits des PIC's

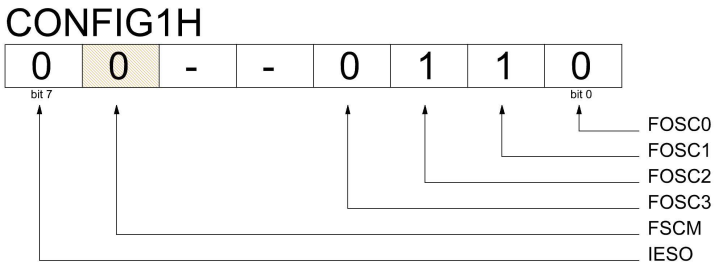
In diesem Programmteil werden die wichtigsten Bits für das Arbeiten mit dem PIC gesetzt.

#pragma config OSC = HSPLL



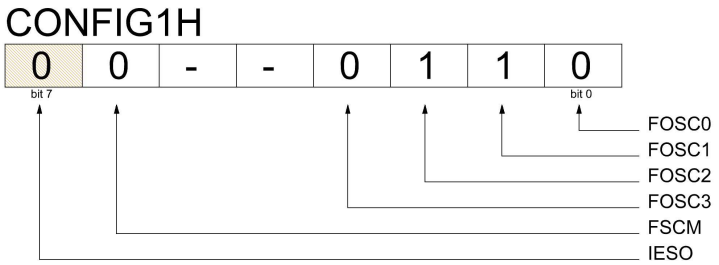
Mit diesem Befehl wird auf den High Speed Modus des PIC umgeschaltet und die interne PLL aktiviert. Die PLL sorgt für eine Vervierfachung der Oszillatorfrequenz.
Der Quarz wird an die PORTs RA6 und RA7 angeschlossen. Die zugehörigen Kondensatoren für die jeweilige Frequenz sind dem Datenblatt des PIC 18F1220/1320 zu entnehmen.

#pragma config FSCM = OFF



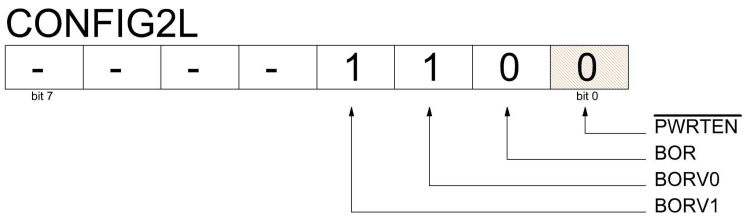
Deaktivierung des „fail safe clock monitor“. Dieser dient dazu, um beim Ausfall des externen Oszillators auf den internen Oszillator umzuschalten und dann mit diesem weiterzuarbeiten.

#pragma config IESO = OFF



Diese Konfigurationseinstellung wird dazu benötigt um ein schnelleres hochfahren des PIC zu ermöglichen. Für unser Projekt wurde dies nicht benötigt, deshalb wurde sie deaktiviert.

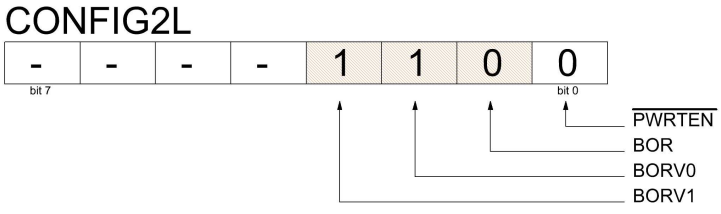
#pragma config PWRT = ON



Der Power – up - Timer dient zur Einschaltverzögerung. Er sorgt dafür, dass der PIC erst bei stabiler Oszillatorfrequenz und stabiler Betriebsspannung zu arbeiten beginnt.

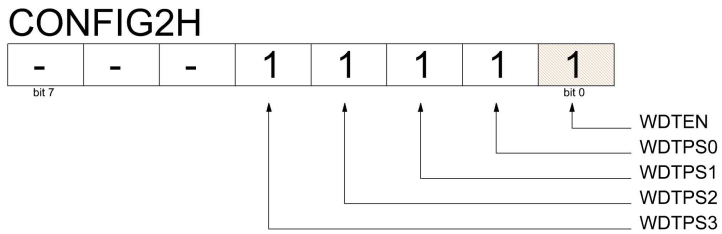
Damit der Power – up – Timer aktiviert wird(PWRT = ON), muss das Bit $\overline{PWRTEN}$ auf 0 gesetzt werden.

#pragma config BOR = OFF



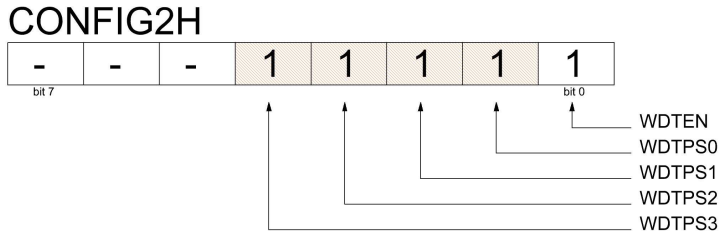
Der Brown-out Reset dient dazu, um den Mikrokontroller, sobald die Versorgungsspannung unter einem definierten Wert gefallen ist, zu reseten.

#pragma config WDT = ON



Der Watchdog Timer dient zur Überprüfung, ob der Prozessor in einer Schleife zu lange hängen geblieben ist. Wenn dieser Fall eintritt, wird der Chip resetet.
Für unser Projekt wurde der Timer aus Stabilitätsgründen aktiviert.

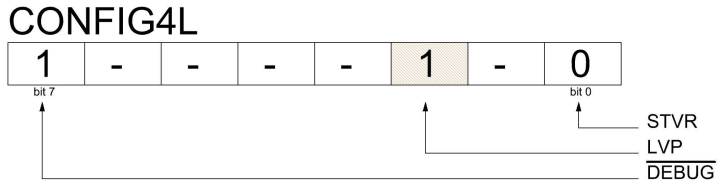
#pragma config WDTPS = 512



Mit diesem Befehl wird die Dauer des Watchdog Timers eingestellt. In unserem Fall wären es ca. 2 Sekunden, da der Timer mit dem internen RC- Oszillator betrieben wird und die minimale Zeit 4ms beträgt.
Daraus ergibt sich folgende Formel für die Dauer des WDT:

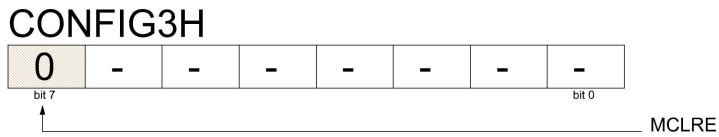
$$4ms \cdot WDTPS = Dauer$$

#pragma config LVP = OFF



Dieser Befehl ermöglicht die Incircuit- Programmierung. Für unser Projekt wurde dies nicht benötigt, da wir den PIC über den PIC Starter Plus programmiert haben.

#pragma config MCLRE = OFF



Der Main Clear Reset, dient zum rücksetzen des Mikrokontrollers über einen Port. Diese Funktion wurde für unsere Anwendung deaktiviert. Da wir diesen PIN als Eingang benötigen.

Definition der globale Variablen

FLAGbits.GetAdress

Dieses Bit dient zur Anzeige, ob sich der Slave gerade im Adressiermodus befindet.
Es wird gesetzt, sobald der Funktionstaster kurz betätigt wird oder keine gültige Adresse aus dem EEPROM ausgelesen werden konnte.

FLAGbits.Is_Relais

Dieses Bit zeigt an, ob dieser Slave als Relaisstation fungiert oder nicht.
Es wird im Unterprogramm Adressencalc (Beschreibung siehe folgende Seiten) gesetzt.
Wenn FLAGbits.Is_Relais gesetzt wurde, dient es zum durchschleifen des Signals nach den vier empfangenen Adressbytes.

FLAGbits.Is_Aussenstelle

Dieses Bit zeigt an, ob dieser Slave als Außenstelle benutzt wird und somit einen Befehl zu verarbeiten hat.
Es wird im Unterprogramm Adressencalc (Beschreibung siehe folgende Seiten) gesetzt.
Wenn FLAGbits.Is_Aussenstelle gesetzt wurde, werden zunächst alle Bytes empfangen, danach die Gültigkeit und Checksumme überprüft und der Befehl ausgeführt. Anschließend werden die geforderten Daten wieder zum Master gesendet.

FLAGbits.BlinkeLED

Dieses Bit dient zum Anzeigen, ob der Befehl Blinke LED empfangen wurde.
Es wird gesetzt sobald der Befehl Blinke LED vom jeweiligen Slave empfangen wurde.
FLAGbits.BlinkeLED wird beim Timer0 Interrupt abgefragt. Wenn der Timer0 Interrupt eingesetzt hat und das Bit gesetzt wurde blinkt die LED fünf Mal.

FLAGbits.RX_Sperren

FLAGbits.RX_Sperren zeigt an, ob der RX Interrupt gesperrt ist oder nicht.
Dieses Bit wird gesetzt, sobald eine Zeitüberschreitung beim Empfangen von zwei Bits eintritt oder ein falsches Byte empfangen wurde.
Es wird beim RX Interrupt abgefragt und beim Interrupt von Timer1 und Timer 2.

FLAGbits.Abschalten

Dieses Bit wird gesetzt, sobald der Befehl Abschalten empfangen wurde.
Es dient dazu um eine Zeitverzögerung zwischen Befehlsempfang und dem Abschalten einzubauen.
FLAGbits.Abschalten wird in der Interrupt Service Routine von Timer1 abgefragt.

SensorDaten.Beschleunigung

Dieses Bit zeigt an, ob seit der letzten Messung neue Messdaten des Beschleunigungssensors vorliegen.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt im Unterprogramm Beschleunigung (Erklärung siehe folgende Seiten).
SensorDaten.Beschleunigung wird rückgesetzt, sobald die Messdaten abgeholt wurden.
Dies erfolgt im Unterprogramm Befehl_verarbeiten.

SensorDaten.DaBeschleunigung

Dieses Bit zeigt an, ob der Beschleunigungssensor vorhanden ist.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt im Unterprogramm Beschleunigung und Beschleunigung_init(Erklärung siehe folgende Seiten).

SensorDaten.TemperaturFT

Dieses Bit zeigt an, ob seit der letzten Messung neue Temperaturmessdaten des SHT11 vorliegen.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt im Unterprogramm Temperatur_ft (Erklärung siehe folgende Seiten).
SensorDaten.TemperaturFT wird rückgesetzt, sobald die Messdaten abgeholt wurden.
Dies erfolgt im Unterprogramm Befehl_verarbeiten.

SensorDaten.FeuchteFT

Dieses Bit zeigt an, ob seit der letzten Messung neue Feuchtigkeitsmessdaten des SHT11 vorliegen.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt im Unterprogramm Feuchtigkeit_ft (Erklärung siehe folgende Seiten).
SensorDaten.FeuchteFT wird rückgesetzt, sobald die Messdaten abgeholt wurden.
Dies erfolgt im Unterprogramm Befehl_verarbeiten.

SensorDaten.DaFeuchteTemperatur

Dieses Bit zeigt an, ob der Sensor SHT11 vorhanden ist.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt in den Unterprogrammen Feuchtigkeit_ft und Temperatur_ft (Erklärung siehe folgende Seiten).

SensorDaten.Temperatur

Dieses Bit zeigt an, ob seit der letzten Messung neue Temperaturmessdaten des Temperatursensors ADT7301 vorliegen.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt im Unterprogramm Temperatur (Erklärung siehe folgende Seiten).
SensorDaten.Temperatur wird rückgesetzt, sobald die Messdaten abgeholt wurden.
Dies erfolgt im Unterprogramm Befehl_verarbeiten.

SensorDaten.DaTemperatur

Dieses Bit zeigt an, ob der Temperatursensor ADT7301 vorhanden ist.
Es wird abgefragt im Unterprogramm Befehl_verarbeiten (Erklärung siehe folgende Seiten) und wird gesetzt in den Unterprogrammen Temperatur (Erklärung siehe folgende Seiten).

unsigned char Adresse

Dieses Byte dient zum Speichern der Slaveadresse während des Betriebs.
Es wird beim Starten des Programms die Adresse aus dem EEPROM (Adresse 0) des Mikrokontrollers ausgelesen und in dieses Byte geschrieben.
Wenn im EEPROM keine Adresse vorhanden ist, wird der Adressiermodus freigegeben und die Status LED beginnt zu leuchten.
Das Byte Adresse wird benötigt für das Unterprogramm Adressencalc und Befehl_verarbeiten. Dort wird es ausgelesen und gesetzt.

unsigned char ID[4]

Diese vier Bytes dienen zum Speichern der ID während der Laufzeit des Programms.
Es werden zum Beginn die vier Bytes aus dem EEPROM (Adresse 2,3,4 und 5) des Mikrokontrollers ausgelesen und in diese Bytes geschrieben.
Die ID wird benötigt für das Unterprogramm Befehl_verarbeiten.
Dort wird sie gesetzt und ausgelesen.

char channel

Dieses Byte dient zum Speichern Sendekanals für das Funkmodul.
Es wird beim Starten des Programms der Kanal aus dem EEPROM (Adresse 1) des Mikrokontrollers ausgelesen und in dieses Byte geschrieben.
Danach wird der Kanal beim Funkmodul eingestellt.
Die Variable channel wird im Unterprogramm Befehl_verarbeiten gesetzt und in der Interrupt Service Routine des Timer1 wird der Kanal ausgelesen und zeitverzögert umgestellt.

unsigned char blinkstatus

Dieses Byte ist eine globale Zählvariable, welche anzeigt wie oft die LED seit dem Befehl BlinkeLED bereits geblinkt hat. Es hat einen Standardwert von 0.
In der Timer0 Interrupt Service Routine erfolgt eine Inkrementierung der Variable um den Wert 1, wenn das Bit FLAGbits.BlinkeLED gesetzt wurde. Diese Variable wird zurückgesetzt, sobald der Befehl BlinkeLED empfangen wurde oder blinkstatus den Wert 10 erreicht.

unsigned char timer0zaehler

Dieses Byte ist eine globale Zählvariable, welche anzeigt wie oft der Timer0 bereits übergelaufen ist. Es hat einen Standardwert von 0.
Diese Variable wird in der Interrupt Service Routine des Timer0 um jeweils eins erhöht.
Sie wird zurückgesetzt, sobald diese Variable den Wert 19 überschreitet.

unsigned char timer1overflow

Dieses Byte ist eine globale Zählvariable, welche anzeigt wie oft der Timer1 bereits übergelaufen ist. Es hat einen Standardwert von 0.
Diese Variable wird in der Interrupt Service Routine des Timer1 erhöht.
Sie wird zurückgesetzt, sobald einer der Befehl „Wechsle den Kanal“ oder „Schalte dich aus!“ kommt.
Weiters wird timer1overflow zurückgesetzt nach einem erfolgreichen Kanalwechsel während des Betriebes.

unsigned char rx_puffer[13]

Dieses 13 Byte große Array dient dazu, um die Empfangsbytes zwischenzuspeichern.
Das Array wird für die Unterprogramme Adressencheck, Adressencalc, Befehl_Check_Calc, Checksumme_Empfangen, Befehl_verarbeiten und die RX Interruptroutine benötigt.
Der rx_puffer wird in der RX Interrupt Service Routine mit Werten befüllt.

unsigned char byte

Diese Variable ist eine globale Zählervariable und wird bei jeden empfangenen Byte um eins erhöht. Der Zähler hat einen Standardwert von 0 und kann maximal den Wert 12 erreichen.
Diese Variable wird in den Interruptroutinen von RX, Timer1 und Timer2 und im Unterprogramm Befehl_Check_Calc benötigt.

Der Zähler wird zurückgesetzt, sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird, sämtliche Bytes richtig empfangen wurden, sämtliche Bytes gesendet wurden oder ein Kanalwechsel durchgeführt wurde.

unsigned char bytemax_empfangen

Diese Byte ist eine globale Statusvariable und zeigt an, wie viele Bytes maximal empfangen werden. Sie hat einen Standardwert von 12 und kann minimal den Wert 5 erreichen.
Diese Variable wird in den Interruptroutinen von RX, Timer1 und Timer2 und in den Unterprogrammen Befehl_Check_Calc, Checksumme_Empfangen benötigt.
Sie wird im Unterprogramm Befehl_Check_Calc gesetzt und sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird, sämtliche Bytes richtig empfangen wurden, sämtliche Bytes gesendet wurden oder ein Kanalwechsel durchgeführt wurde wieder auf den Wert 12 zurückgesetzt.

unsigned char tx_puffer[13]

Dieses 13 Byte große Array dient dazu, um die Sendebytes zwischenzuspeichern.
Das Array wird für die Unterprogramme Adressencalc, Checksumme_Senden, Befehl_verarbeiten, Standardantwort und die RX und TX Interruptroutine benötigt.
Der tx_puffer wird hauptsächlich in den Unterprogrammen Adressencalc, Standardantwort und Befehl_verarbeiten mit Werten befüllt.

unsigned char sendecounter

Diese Variable ist eine globale Zählervariable und wird bei jeden gesendeten Byte um eins erhöht. Der Zähler hat einen Standardwert von 0 und kann maximal den Wert 12 erreichen.
Diese Variable wird in der TX Interruptroutine benötigt.
Der Zähler wird zurückgesetzt, sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird oder sämtliche Bytes gesendet wurden.

unsigned char bytemax_senden

Diese Byte ist eine globale Statusvariable und zeigt an, wie viele Bytes maximal gesendet werden. Sie hat einen Standardwert von 12 und kann minimal den Wert 5 erreichen.
Diese Variable wird in den RX und TX Interruptroutinen und in den Unterprogrammen Checksumme_Senden, Standardantwort und Befehl_verarbeiten benötigt.
Sie wird im Unterprogramm Standardantwort, Befehl_verarbeiten und in der RX Interruptroutine gesetzt. Sobald das Programm neu gestartet wird, ein Timeout auftritt, ein falsches Byte empfangen wird oder sämtliche Bytes gesendet wurden wird sie wieder auf den Wert 12 zurückgesetzt.

unsigned char x_besch[2], y_besch[2], z_besch[2]

Drei jeweils zwei Byte große Variablen zum Zwischenspeichern des Beschleunigungsmesswertes bis er vom Master abgeholt wird.
Diese Variablen werden im Unterprogramm Beschleunigung gesetzt und im Unterprogramm Befehl_verarbeiten ausgelesen.

unsigned char feuchte_ft[2]

Eine zwei Byte große Variable zum Zwischenspeichern des Feuchtigkeitsmesswertes des Sensors SHT11 bis er vom Master abgeholt wird.
Diese Variable wird im Unterprogramm Feuchtigkeit_ft gesetzt und im Unterprogramm Befehl_verarbeiten ausgelesen.

unsigned char temp1_ft[2]

Eine zwei Byte große Variable zum Zwischenspeichern des Temperaturmesswertes des Sensors SHT11 bis er vom Master abgeholt wird.
Diese Variable wird im Unterprogramm Temperatur_ft gesetzt und im Unterprogramm Befehl_verarbeiten ausgelesen.

unsigned char temp2_t[2]

Eine zwei Byte große Variable zum Zwischenspeichern des Temperaturmesswertes des Temperatursensors ADT7301 bis er vom Master abgeholt wird.
Diese Variable wird im Unterprogramm Temperatur gesetzt und im Unterprogramm Befehl_verarbeiten ausgelesen.

LED:

Immer wenn in der Slaveprogrammdokumentation die Bezeichnung LED verwendet wird, bezieht sich dieser Ausdruck auf die rote LED (LED201), welche zur Funktionsanzeige dient. Diese Leuchtdiode hängt am Port RA4 des Mikrokontrollers.

Programmcode:

Sämtlicher Programmcode wurde zwecks besserer Übersicht weggelassen.
Der vollständige Programmcode befindet sich auf der Diplomarbeit - DVD im C – File „PIC\Slave\Slave_V3.3.c“.
Die detaillierte Programmdokumentation mit zugehörigem Programmcode ist auf der Diplomarbeit – DVD in der Datei “PIC\Slave\PIC Slaveprogrammdokumentation_V1.3.doc” zu finden.

Unterprogramme

Senden von Strings mittels USART Schnittstelle

void putrsUSART_(const rom char *data)

Dieses Unterprogramm dient zum Umstellen der Einstellungen des Funkmoduls.
Mit dem vordefinierten Unterprogramm der USART Bibliothek konnten die Einstellungen des Funkmoduls nicht geändert werden, da dieses Programm am Ende jedes Strings eine 0 gesendet hat was zu Konflikten geführt hat.
Deshalb wurde ein eigenes Unterprogramm dafür geschrieben.
Es wird in der Interrupt Service Routine des Timer1 und im Hauptprogramm benötigt.

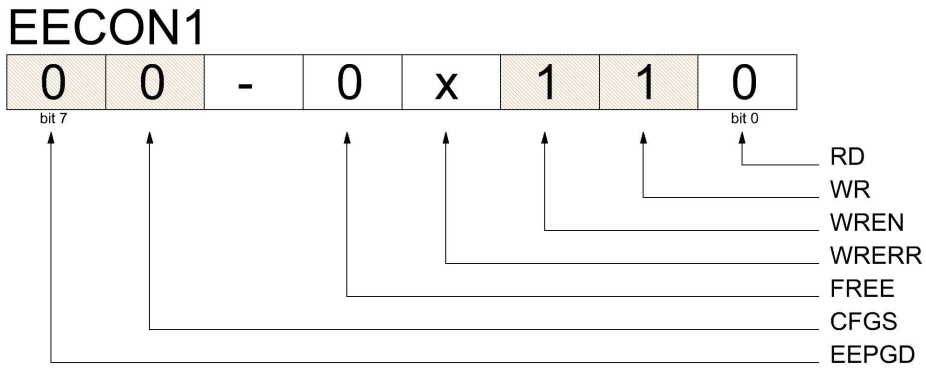
I2C Bus Treiber

Funktionsbeschreibung:
Dieses Unterprogramm wird zur Kommunikation zwischen PIC und dem Beschleunigungssensor benötigt.
Die Kommunikation erfolgt nach dem I2C Standard.
Dieser Treiber wurde nahezu vollständig aus der Compiler Library des MCC18 Compilers entnommen. Es wurde der Programmcode lediglich für unsere Applikation modifiziert.
Teile dieses Treibers werden in den Unterprogrammen Beschleunigung und Beschleunigung_init verwendet.

Beschreiben des EEPROM

void WriteEEPROM(unsigned char Adresse, unsigned char Data)

Setzen der Bits zum Schreiben der an der Stelle von Data stehenden Variable in die an der Stelle von Adresse stehenden Adresse in das EEPROM.

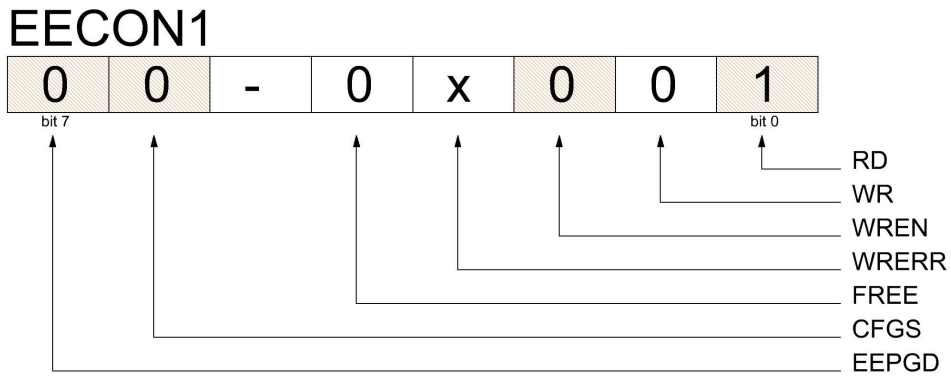


Nach dem beschreiben des EEPROM wird wieder von EEPROM Memory auf Programm Memory umgeschaltet.
Wenn diese Funktion verwendet wird ist es wichtig, dass keine Interrupts auftreten können, da es ansonsten zu Schreibfehlern kommen könnte.

Auslesen des EEPROM

unsigned char ReadEEPROM(unsigned char Adresse)

Auslesen eines Bytes an der Stelle von Adresse stehenden Adresse aus dem EEPROM.



Diese Funktion gibt als Rückgabewert das ausgelesene Byte zurück.

Überprüfung der Adresse auf ihre Gültigkeit

char Adressencheck(void)

Dient zur Überprüfung, ob die vier empfangenen Adressbytes dem Muster 1XXXXXXX (B) entsprechen.
Diese Funktion liefert im Fehlerfall den Wert 0 zurück.

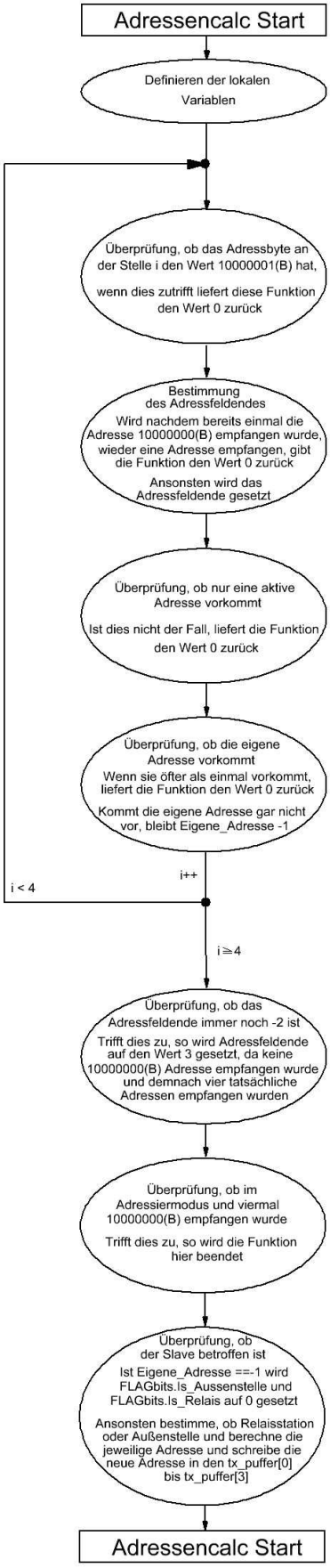
Adressensberechnung

char Adressencalc(void)

Lokale Variablen:

- char Adressfeldende
  Gibt an wie viele der vier empfangen Adressbytes tatsächliche Slaveadressen sind und dem Muster 1XXXXXXX (B) entsprechen. Die Adresse 10000000 (B) ist in diesem Fall eine Spezialadresse und wird nicht als gültige Adresse erkannt.
  Die Variable Adressfeldende wird dann auf den Wert der tatsächlichen Adressen gesetzt.
  Sie kann im Bereich von -2 bis 3 liegen und hat einen Standardwert von -2.
- char Aktive_Adresse
  Diese Variable dient zur Überprüfung, ob tatsächlich immer nur eine Adresse aktiv ist.
  Eine aktive Adresse wird an der Form 1XXXXXX1 (B) erkannt.
  Sie liegt im Bereich von -1 und 3 und hat einen Standardwert von -1.
- char Eigene_Adresse
  Diese Variable zeigt an, welches Byte der Adressbytes die eigene Adresse enthält.
  Sie liegt im Bereich von -1 bis 3 und hat einen Standardwert von -1.
- char i
  Ist eine lokale Zählvariable. Sie dient zum berechnen der neuen Adresse und wird von 0 bis 3 hoch gezählt.
  Diese Variable wird standardmäßig mit 0 initialisiert.

Prinzipieller Aufbau:
Adressensberechnung



Beschreibung:
Im Fall eines Fehlers liefert diese Funktion den Wert 0 zurück, andernfalls den Wert 1.
Fungiert der betroffene Slave als Relaisstation, so setzt sie die eigene Adresse auf inaktiv und die nächste Adresse auf aktiv.
Dient der Slave als Außenstelle so wird der empfangene Adressbereich gespiegelt. Die letzte empfangene Adresse vor der eigenen Adresse wird auf aktiv gesetzt und anstelle der eigenen Adresse wird als letzte gültige Adresse die Masteradresse eingefügt. Der Rest der nicht benötigten Adressbytes wird mit $10000000_{(B)}$ angefüllt.

Gültigkeitsüberprüfung eines Befehls

char Befehl_Check_Calc(void)

Dient zur Überprüfung, ob das fünfte empfangene Byte (Befehlsbyte) dem Muster $0XXXXXXX_{(B)}$ entspricht. Weiters setzt das Unterprogramm die Variable bytemax_empfangen auf den Zahlenwert, der den drei niederwertigsten Bits des Befehlsbyte plus sechs entspricht.
Diese Funktion liefert im Fehlerfall den Wert 0 zurück.

Checksummenberechnung fürs Senden und Empfangen

Lokale Variablen:

- char checksumme
  Zwischenvariable zum bilden der Checksumme. Deren Wert wird abschließend von der Funktion zurückgegeben.
- char i
  Lokale Zählvariable für die Checksummenbildung.
  Sie kann Werte von 0 bis 11 annehmen und wird standardmäßig mit 0 initialisiert.

char Checksumme_Empfangen(void)

Diese Funktion verknüpft alle empfangenen Bytes, bis auf das letzte Byte, durch eine Exklusiv-Oder Operation miteinander.
Das Endergebnis wird von dieser Funktion zurückgegeben.

char Checksumme_Senden(void)

Diese Funktion verknüpft alle zu sendenden Bytes, bis auf das letzte Byte für die Checksumme, durch eine Exklusiv- Oder Operation miteinander.
Das Endergebnis wird von dieser Funktion zurückgegeben.

Befehlsverarbeitung

void Standardantwort(void)

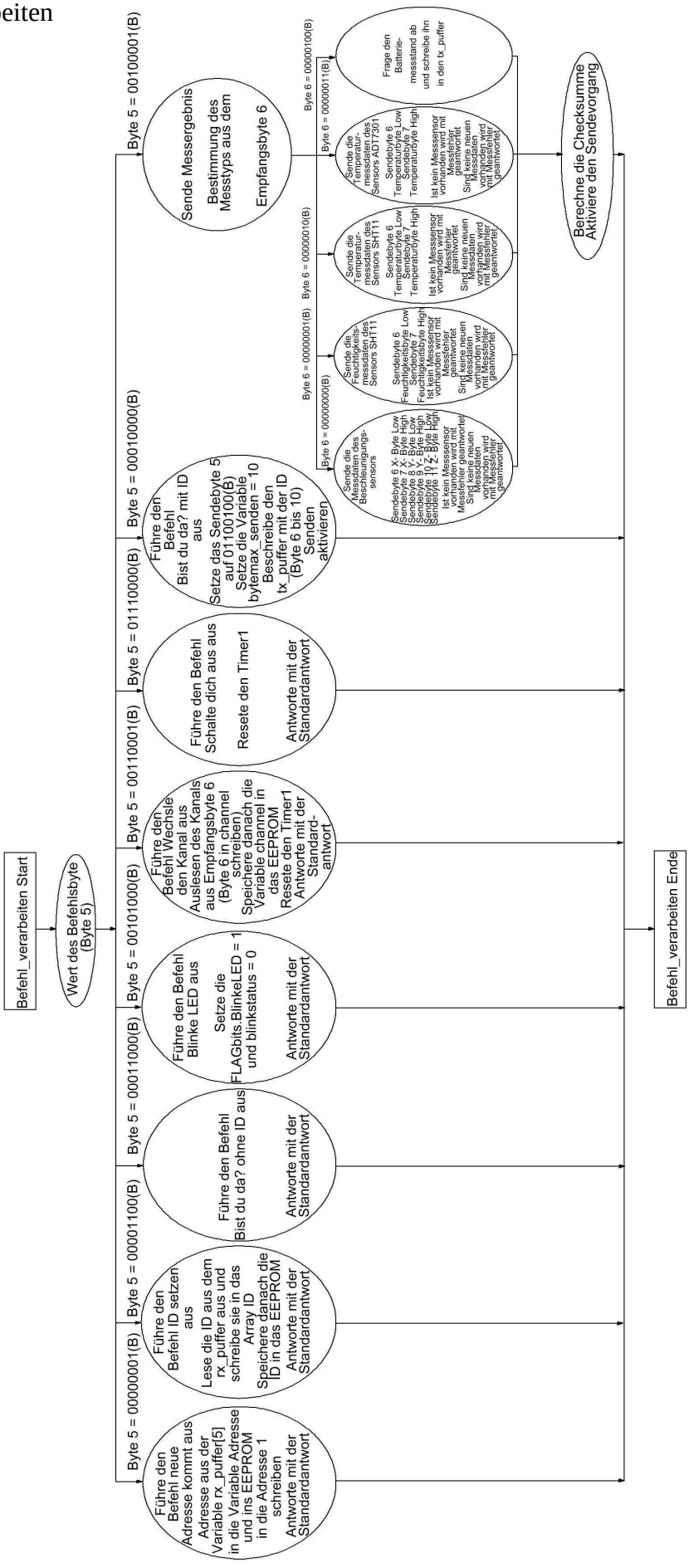
Dieses Unterprogramm dient zum Senden der Antwort „Bin da ohne ID!“.
Es setzt das fünfte Byte des Sendearrays (tx_puffer[4]) auf den Wert $01101000_{(B)}$, was der Antwort auf den Befehl „Bist du da? ohne ID“ entspricht.
Die Variable bytemax_senden wird auf den Wert 6 gesetzt und das tx_puffer Byte für die Checksumme wird mit der zugehörigen Checksumme beschrieben.
Außerdem wird danach die Senderoutine gestartet.

void Befehl_verarbeiten(void)

Lokale Variable:

- char i
  Lokale Zählvariable für die Befehlsverarbeitung.
  Sie kann Werte von 0 bis 9 annehmen und ist standardmäßig auf dem Wert 0.
  Diese Variable wird für das Speichern der ID in das EEPROM und für die Antwort auf den Befehl „Bist du da? mit ID“ benötigt.

Prinzipieller Aufbau:
Befehl_verarbeiten

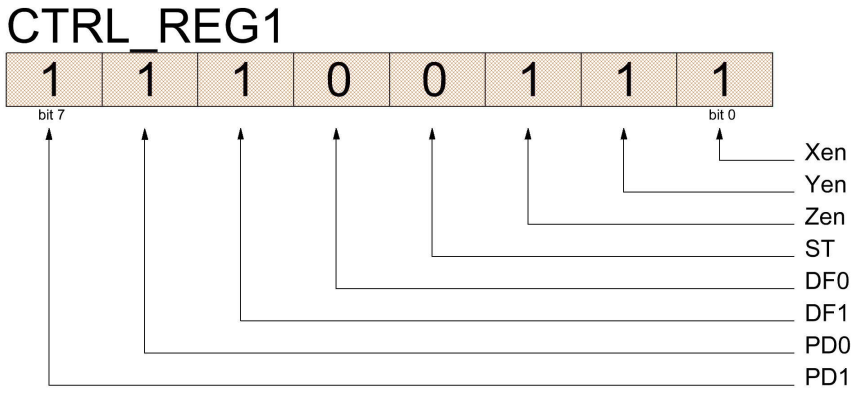


Beschreibung der Funktion Befehl_verarbeiten:
In diesem Unterprogramm werden die einzelnen Befehle des Masters entschlüsselt und durchgeführt. Je nach Befehlstyp wird entweder mit der Standardantwort „Bin da! ohne ID“ geantwortet oder mit den geforderten Daten. Weiters werden die Variablen ID, Channel und Adresse in den EEPROM geschrieben.

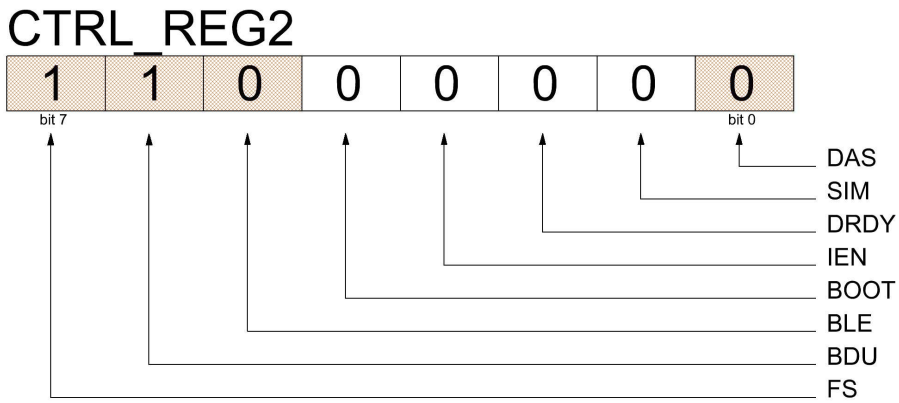
Messen der Beschleunigung

void Beschleunigung_init(void)

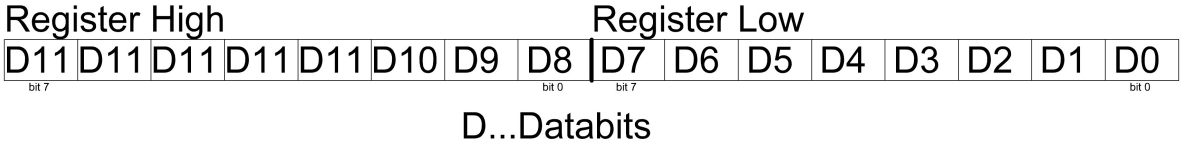
Funktionsbeschreibung:
Mit diesem Unterprogramm erfolgen die Einstellungen des Beschleunigungssensors.
Er wird auf „Device ON, 640Hz, XYZ aktiv, 6g, Block Data Update, 12 bit right justified“ eingestellt.
Für diese Einstellungen müssen folgende Register des Sensors wie folgt gesetzt werden:



Durch das Setzen dieser Bits des CTRL_REG1 – Register wird der Sensor aktiviert (PD1 = 1, PD0 = 1), die Sampling - Rate auf 640Hz eingestellt (DF1 = 1, DF0 = 0) und die X -, Y - und Z – Achse aktiviert (Bit 0 bis 2 auf 1 setzen). Der Selbsttest (ST) des Sensors bleibt standardmäßig inaktiv.



Durch das Setzen dieser Bits des CTRL_REG2 – Register wird der Full Scale (FS) Modus aktiviert (± 6g), das BDU – Bit verhindert das Aktualisieren der Ausgaberegister während die Daten abgefragt werden (BDU = 1) und das BLE – Bit dient zum Umstellen zwischen Big - Endian oder Little – Endian Modus (BLE = 0 aktiviert Little – Endian Modus). Das Bit DAS bewirkt bei der obigen Konfiguration folgende Anordnung der Messdaten:



Alle anderen Bits des CTRL_REG2 – Registers sind für die Kommunikation über I2C nicht notwendig. Deshalb werden sie auf den Werkseinstellungen belassen.

char Neuer_Maximalwert(char alter_Wert0,char alter_Wert1,char neuer_Wert0,char neuer_Wert1)

Übergabe Variablen:

- char alter_Wert0
  Dieser Platzhalter übergibt der Funktion, dass in der globalen Variable stehende Byte mit den MSB für den jeweiligen Messwert (x_besch[0], y_besch[0] oder z_besch[0])
- char alter_Wert1
  Dieser Platzhalter übergibt der Funktion, dass in der globalen Variable stehende Byte mit den LSB für den jeweiligen Messwert (x_besch[1], y_besch[1] oder z_besch[1])
- char neuer_Wert0
  Dieser Platzhalter übergibt der Funktion das aktuelle Messbyte mit den MSB für den jeweiligen Messwert (x_besch_[0], y_besch_[0] oder z_besch_[0])
- char neuer_Wert1
  Dieser Platzhalter übergibt der Funktion das aktuelle Messbyte mit den MSB für den jeweiligen Messwert (x_besch_[1], y_besch_[1] oder z_besch_[1])

Funktionsbeschreibung:
In dieser Funktion wird der aktuelle Messwert mit den Letzten, in den globalen Variablen stehenden Messbytes, verglichen.
Diese Funktion bewertet negative und positive Spitzenwerte gleichermaßen und sorgt somit dafür, dass der absolute Spitzenwert erkannt wird.
Ist der aktuelle Wert größer als der zuletzt gemessene Wert, so liefert diese Funktion 1 zurück, ist dies nicht der Fall so liefert sie 0 zurück.

void Beschleunigung(void)

Lokale Variablen:

- unsigned char x_besch_[2], y_besch_[2], z_besch_[2]
  Dies sind lokale jeweils 2 Byte große Arrays, die zum Zwischenspeichern des aktuellen X-, Y- und Z- Wertes dienen
- unsigned char x_spitze, y_spitze, z_spitze
  Diese Variablen sind Statusvariablen, die den Rückgabewert der Funktion Neuer_Maximalwert auffangen. Sie enthalten die Information, ob der letzte Messwert größer oder kleiner als der aktuelle Messwert ist.
  Ist der aktuelle Messwert größer als der Letzte so ist diese Variable 1 andernfalls ist sie 0.

Funktionsbeschreibung:
Mit diesem Unterprogramm erfolgt die Erfassung der Beschleunigungsmesswerte. Die Messung ist eine Spitzenwertmessung.
Sobald die Messdaten vom Master abgeholt wurden, wird der Spitzenwert auf den aktuellen Messwert gesetzt.
Die Übertragung der Messdaten erfolgt nach dem I2C Standard.
Weiters wird in diesem Unterprogramm festgestellt, ob der Sensor vorhanden ist oder nicht.

Messen der Feuchtigkeit

signed int Feuchtigkeit_ft(signed int ManagerFT)

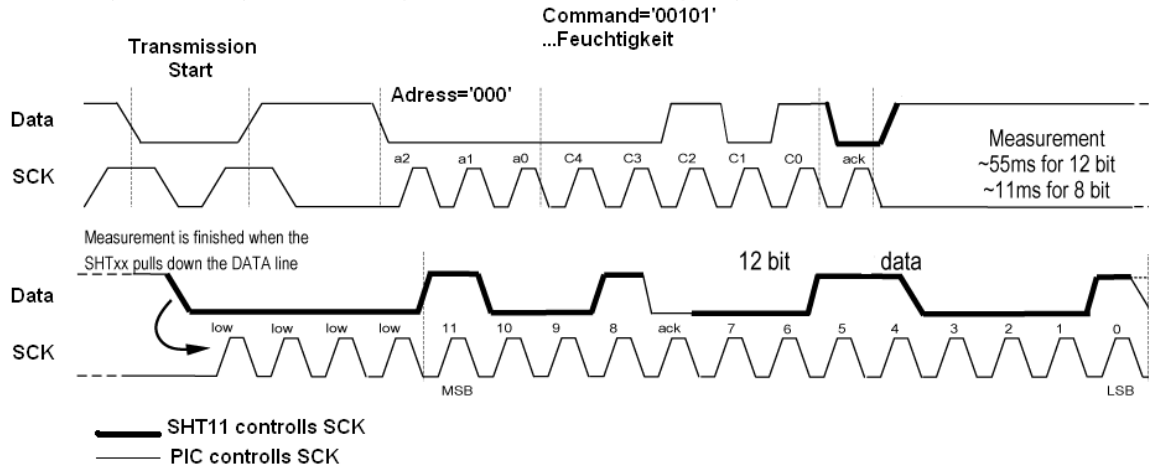
Übergabe Variable:

- signed int ManagerFT
  Diese Variable dient zum Koordinieren des Sensors SHT11.
  Sie dient dazu, um nach einer erfolgreichen Messung vom Messtyp Feuchtigkeit auf den Messtyp Temperatur umzuschalten. Die Variable sorgt weiters dafür, dass die laut Datenblatt empfohlene Wartezeit von circa einer Sekunde zwischen den Messungen eingehalten wird.

Lokale Variablen:

- unsigned char i
  Diese Variable ist eine lokale Zählvariable.
  Sie dient zum Empfangen der Messdaten des Sensors und zum Generieren des Clock-Signals für den Sensor.
  Diese Zählvariable wird mit 0 initialisiert und kann maximal den Wert 7 annehmen.
- unsigned char data
  Diese lokale Variable enthält die MSB des Feuchtigkeitsmesswertes.
- unsigned char data2
  Diese lokale Variable enthält die LSB des Feuchtigkeitsmesswertes.

Funktionsbeschreibung:
Mit diesem Unterprogramm erfolgt die Erfassung des Feuchtigkeitsmesswertes des SHT11 Sensors.
Dieses Unterprogramm erzeugt folgendes Bitmuster zur Kommunikation mit dem Sensor:



Weiters wird in diesem Unterprogramm festgestellt, ob der Sensor vorhanden ist oder nicht.
Außerdem wird die Übergabevariable im Bereich von 0 bis 1000 verändert.

Messen der Außentemperatur

signed int Temperatur_ft(signed int ManagerFT)

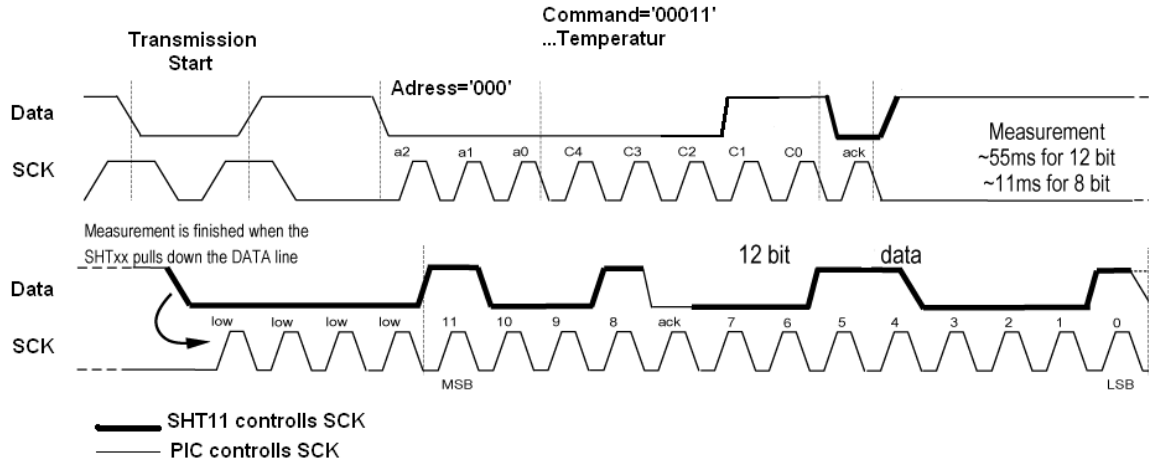
Übergabe Variable:

- signed int ManagerFT
  Diese Variable dient zum Koordinieren des Sensors SHT11.
  Sie dient dazu, um nach einer erfolgreichen Messung vom Messtyp Temperatur auf den Messtyp Feuchtigkeit umzuschalten. Die Variable sorgt weiters dafür, dass die laut Datenblatt empfohlene Wartezeit von circa einer Sekunde zwischen den Messungen eingehalten wird.

Lokale Variablen:

- unsigned char i
  Diese Variable ist eine lokale Zählvariable.
  Sie dient zum Empfangen der Messdaten des Sensors und zum Generieren des Clock-Signals für den Sensor.
  Diese Zählvariable wird mit 0 initialisiert und kann maximal den Wert 7 annehmen.
- unsigned char data
  Diese lokale Variable enthält die MSB des Temperaturmesswertes.
- unsigned char data2
  Diese lokale Variable enthält die LSB des Temperaturmesswertes.

Funktionsbeschreibung:
Mit diesem Unterprogramm erfolgt die Erfassung des Temperaturmesswertes des SHT11 Sensors. Dieses Unterprogramm erzeugt folgendes Bitmuster zur Kommunikation mit dem Sensor:



Weiters wird in diesem Unterprogramm festgestellt, ob der Sensor vorhanden ist oder nicht. Außerdem wird die Übergabevariable im Bereich von 0 bis -1000 verändert.

Messen der punktuellen Temperatur

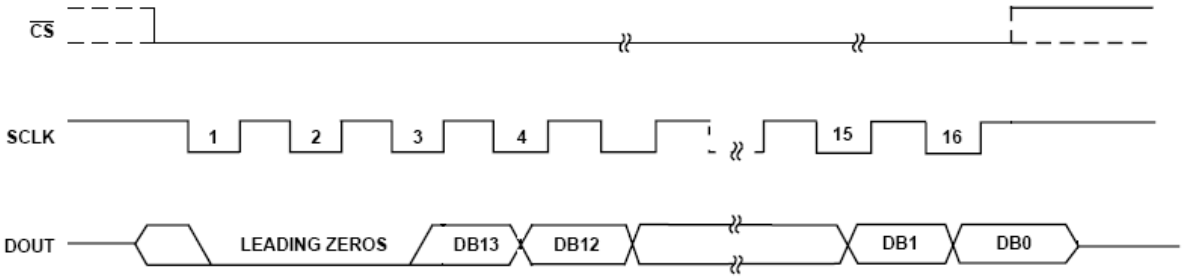
void Temperatur(void)

Lokale Variablen:

- char temp_zaehler
  Diese Variable ist eine lokale Zählvariable.
  Sie dient zum Empfangen der Messdaten des Sensors und zum Genieren des Clock-Signals für die Kommunikation zwischen PIC und Sensor.
  Diese Zählvariable wird mit 0 initialisiert und kann maximal den Wert 15 annehmen.
- unsigned int temp_wert

Diese lokale Variable enthält die aktuellen 14Bit Temperaturmessdaten.

Funktionsbeschreibung:
Mit diesem Unterprogramm erfolgt die Erfassung der Temperaturmessdaten des ADT7301 Sensors.
Dieses Unterprogramm erzeugt folgendes Bitmuster zur Kommunikation mit dem Sensor:



Weiters wird in diesem Unterprogramm festgestellt, ob der Sensor vorhanden ist oder nicht.

Definition der Interrupt Service Routinen

Beschreibung:
Der folgende Code wird zur Initialisierung der Interrupt Service Routinen vom Compiler benötigt.

High Priority Interrupt Service Routine

void high_isr (void)

Lokale Variable:

- char Empfang_OK
  Ist ein lokales FLAG das anzeigt, ob beim Empfang ein Fehler aufgetreten ist oder sämtliche Bytes empfangen wurden.
  Im Fehlerfall wird dieses FLAG auf 0 gesetzt.
  Im Falle, dass alle Bytes empfangen wurden wird diese FLAG auf -1 gesetzt.

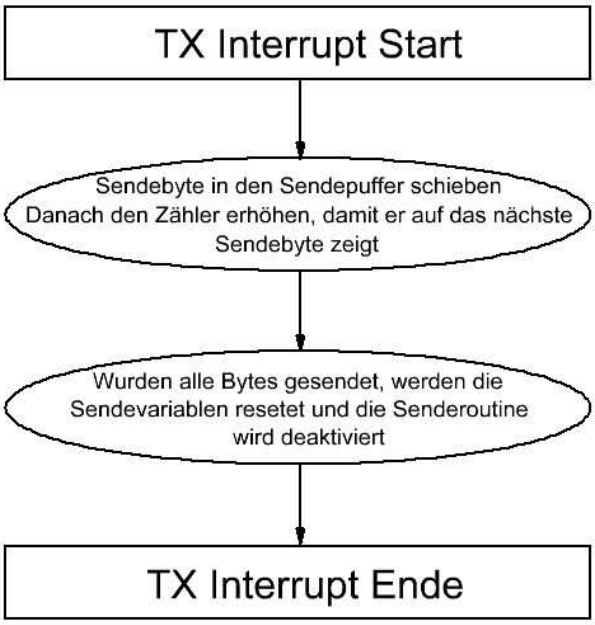
Begriffserklärung Timeout:

Ein Timeout tritt immer dann ein, wenn vom Empfang eines Bytes bis zum nächsten mehr als 2ms vergangen sind oder ein falsches Byte empfangen wurde.
Das Timeout dient zur schnelleren Synchronisation nachdem ein Fehler aufgetreten ist.

Prinzipieller Aufbau:

Verarbeitung eines Empfang(RX) Interrupts

Verarbeitung eines Sende(TX) Interrupts



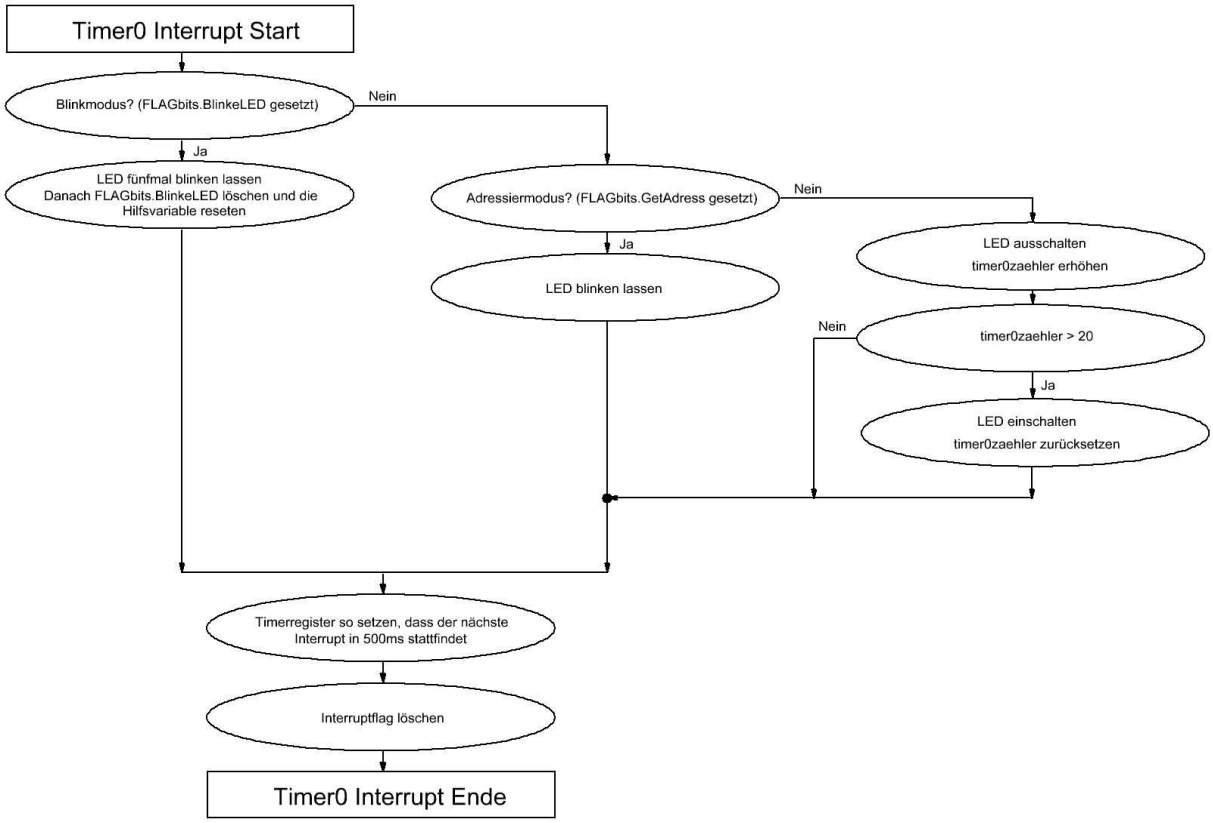
Beschreibung:
Mit diesem ISR erfolgt die Koordination zwischen der Empfangs- und Sendefunktion.
Weiters erfolgt in dieser Routine die Adressenverwaltung und die Befehlsverarbeitung.

Low Priority Interrupt Service Routine

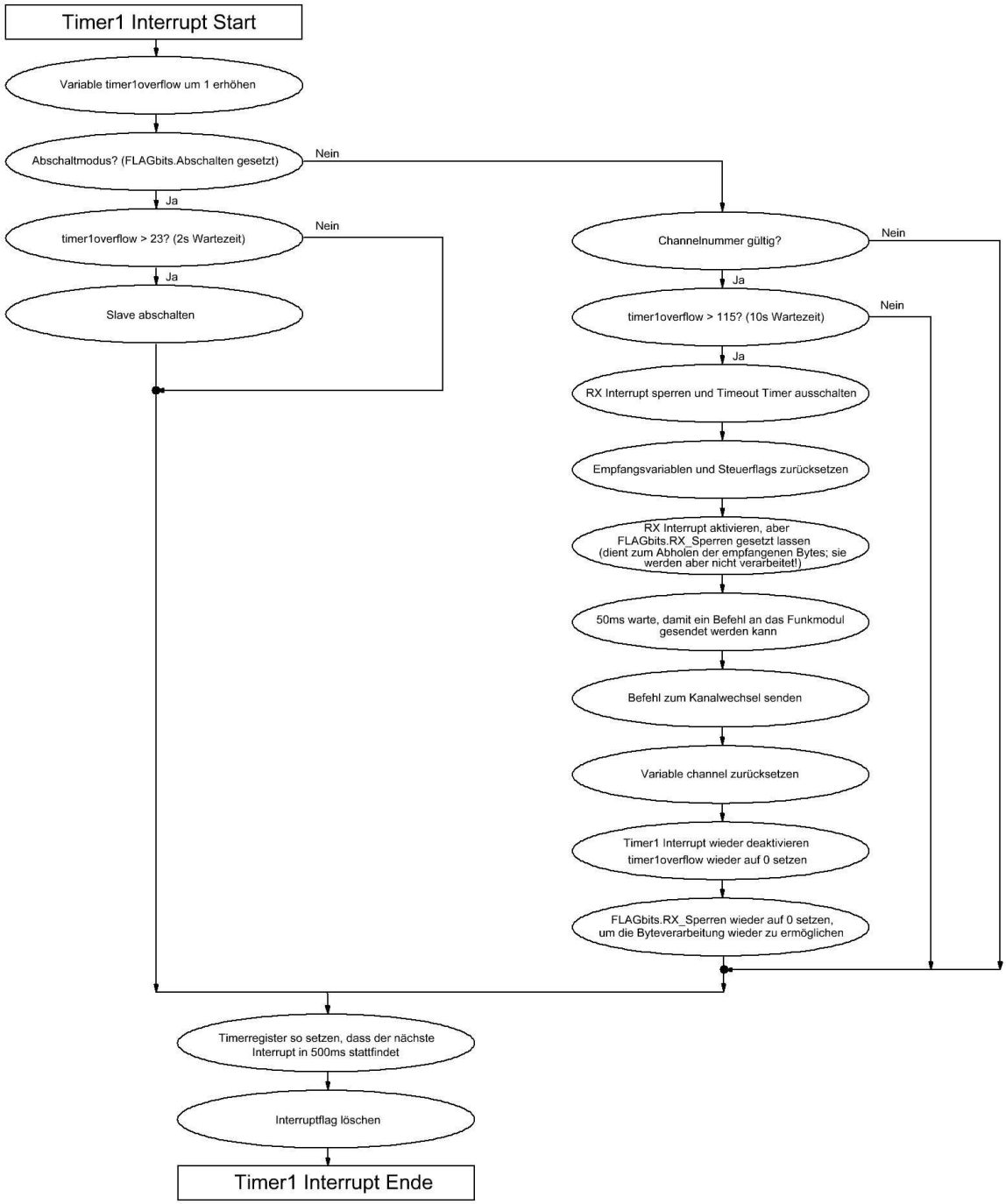
void low_isr (void)

Prinzipieller Aufbau:

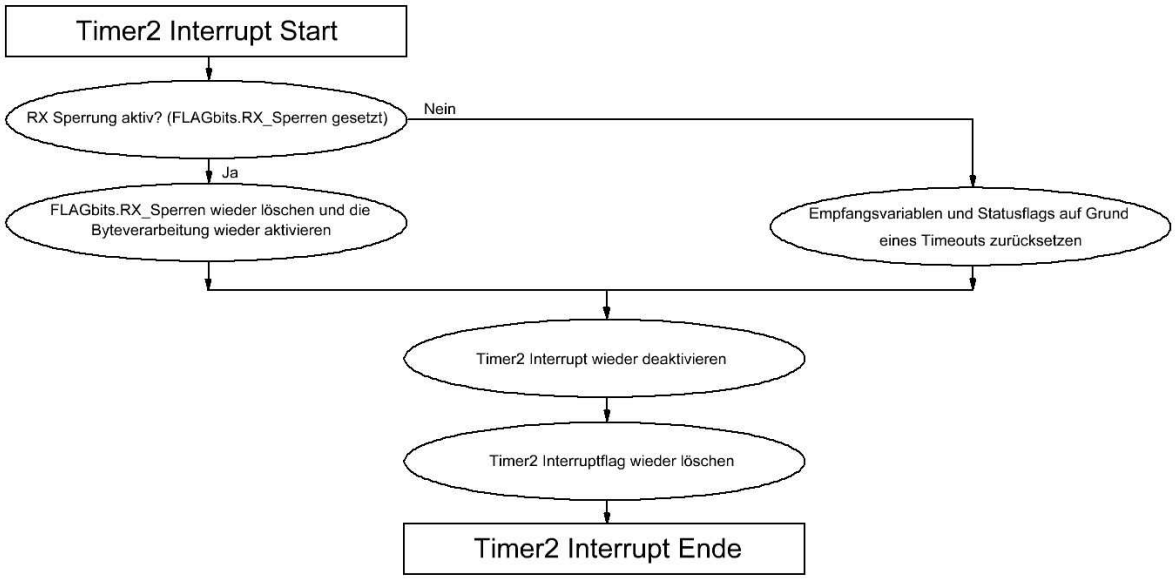
Verarbeitung eines Timer 0 Interrupts



Verarbeitung eines Timer 1 Interrupts



Verarbeitung eines Timer 2 Interrupts



Beschreibung:

In dieser Routine erfolgt die Verwaltung der Timer für die folgenden Aufgaben:

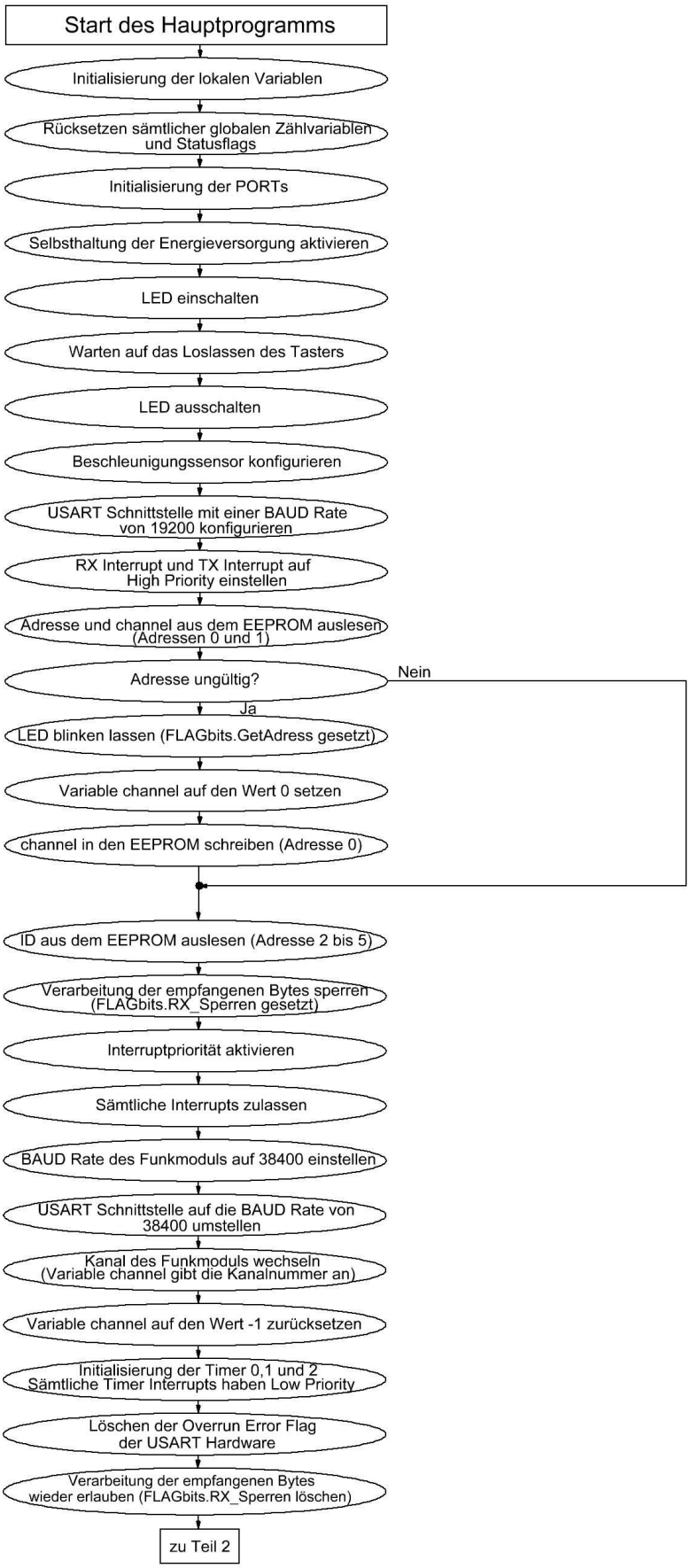
- Blinken der LED (Timer 0)
  - Im Adressiermodus im Sekundentakt
  - Bei Befehl „Blinke LED“ 5x im Sekundentakt
  - Ansonsten leuchtet die LED alle zehn Sekunden einmal für 500ms auf
- Abfragen des Funktionstasters (Timer 0)
- Zeitverzögertes Ausschalten (Timer 1)
- Zeitverzögertes Kanalwechseln (Timer 1)
- Entsperren bzw. sperren der Empfangsroutine bei einem Timeout (Timer 2)

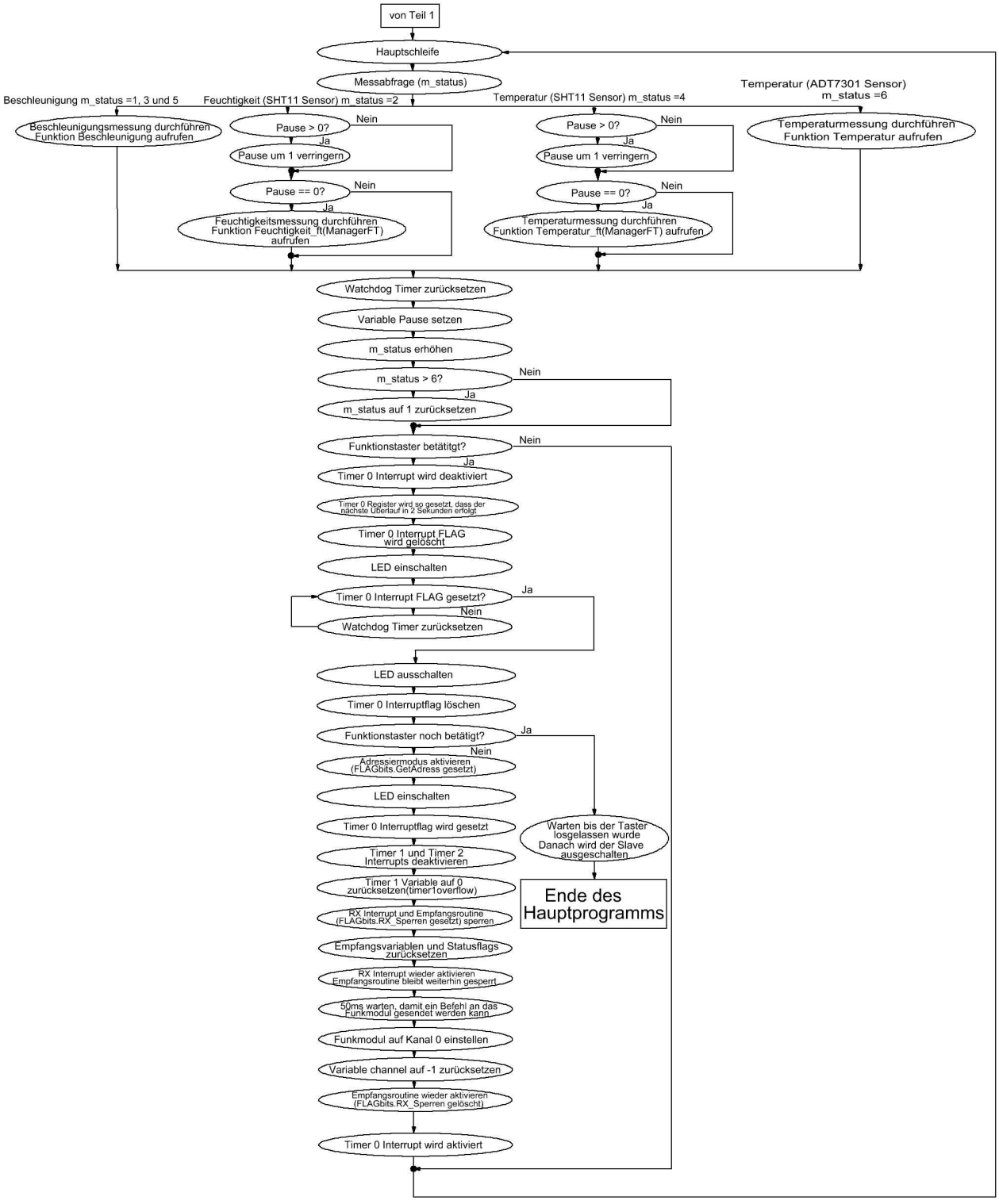
Hauptprogramm

Lokale Variablen:

- char m_status
  Diese Variable dient dazu, um die Messvorgänge zu steuern.
  Nach jeder Messung wird diese Variable um den Wert 1 erhöht und es wird dadurch der nächste Sensor abgefragt.
  Die Variable wird mit dem Wert 1 initialisiert und kann maximal den Wert 7 erreichen.
- char i
  i ist eine lokale Zählvariable, die zum Auslesen der ID aus dem EEPROM dient.
  Sie wird dem Wert 0 initialisiert und kann maximal den Wert 3 erreichen.
- signed int ManagerFT
  Diese Variable dient zur Koordination des SHT11 Sensors.
  Da dieser Sensor eine Kombination aus Feuchtigkeits- und Temperatursensor ist, wird eine Variable benötigt, die dafür sorgt, dass der Messtyp bei Bedarf gewechselt wird.
  ManagerFT wird in den Funktionen Feuchtigkeit_ft und Temperatur_ft verändert.
  Diese Variable wird mit dem Wert 0 initialisiert und kann im Bereich von -1000 bis +1000 liegen.
- unsigned char Pause
  Pause ist eine Statusvariable, die dafür sorgt, dass die nötigen Pausen für den Sensor SHT11eingehalten werden, um ein überhitzen des ICs zu vermeiden.
  Pause wird mit dem Wert 1 initialisiert und kann im Bereich von 0 bis 251 liegen.

Prinzipieller Aufbau:





Beschreibung:

Das Hauptprogramm dient zum Erfassen der Messwerte und zur Verwaltung des Funktionstasters. Weiters werden zu Beginn des Hauptprogramms sämtliche Variablen und FLAGs auf definierte Zustände gebracht und die benötigten Timer und die USART Schnittstelle konfiguriert.

PIC Software Slave Linker File

Da der vom Compiler zur Verfügung gestellte Stack nicht ausreichend war, musste der Stack-Bereich manuell im Linker File vergrößert werden.
Dies erreicht man durch folgende Änderungen im File „18f1320.lkr“:

Original Linker File		**Änderungen im Linker File**
STACK SIZE=0x40	→	STACK SIZE=0x70
RAM=gpr0	→	RAM=accessram

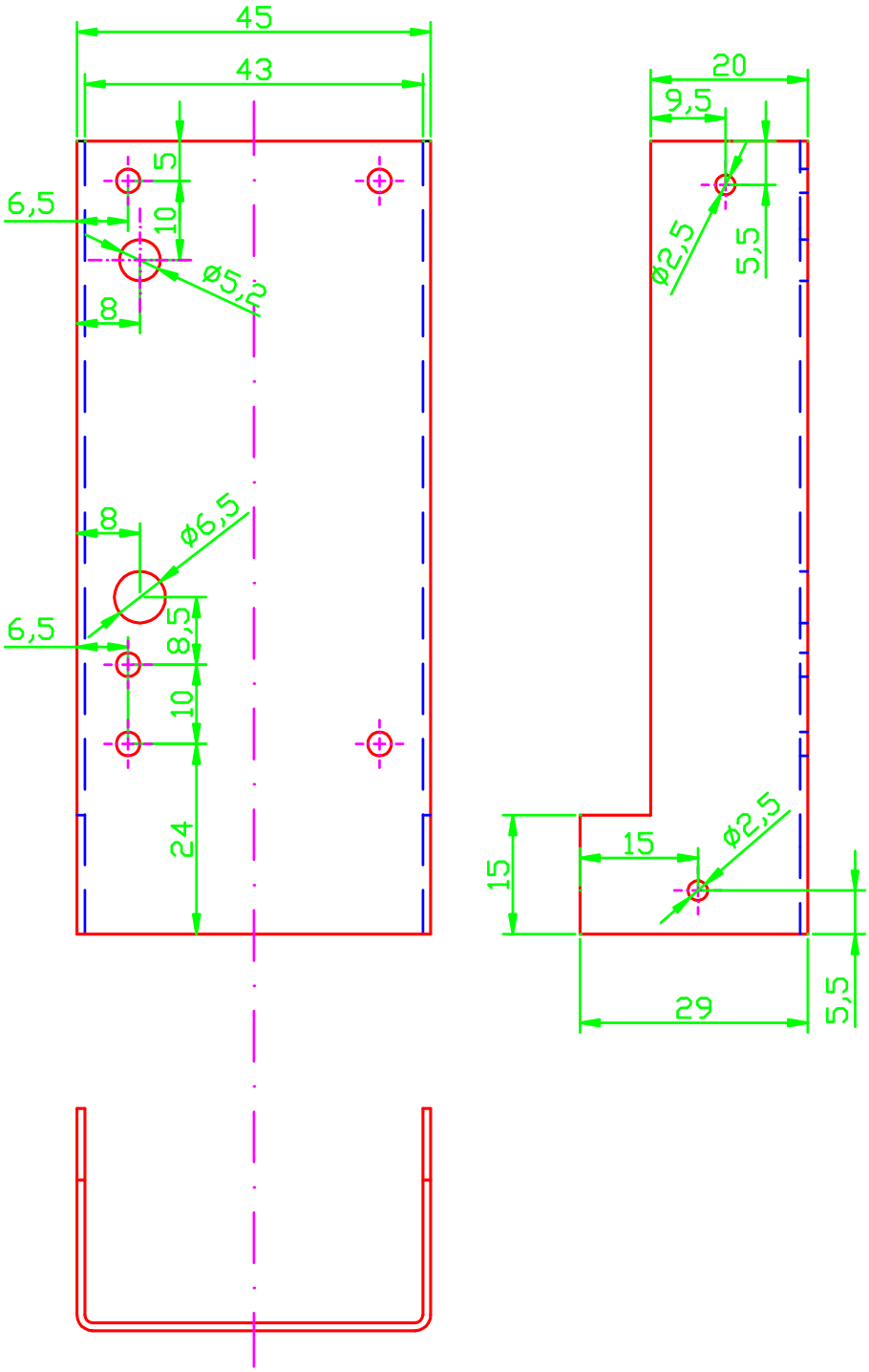
Diese Daten befinden sich in der letzten Zeile der Datei.
Diese Änderung bringt eine Stackerweiterung um 48 Byte.

5.1.3.17. Gehäuse Slave

Sämtliche Abbildungen sind nicht im Maßstab! Alle Abmessungen sind in mm!
Alle Biegungen haben einen Innenradius von 1mm und einen Außenradius von 2mm.
Alle unbemaßten Bohrlöcher haben einen Durchmesser von 3mm.

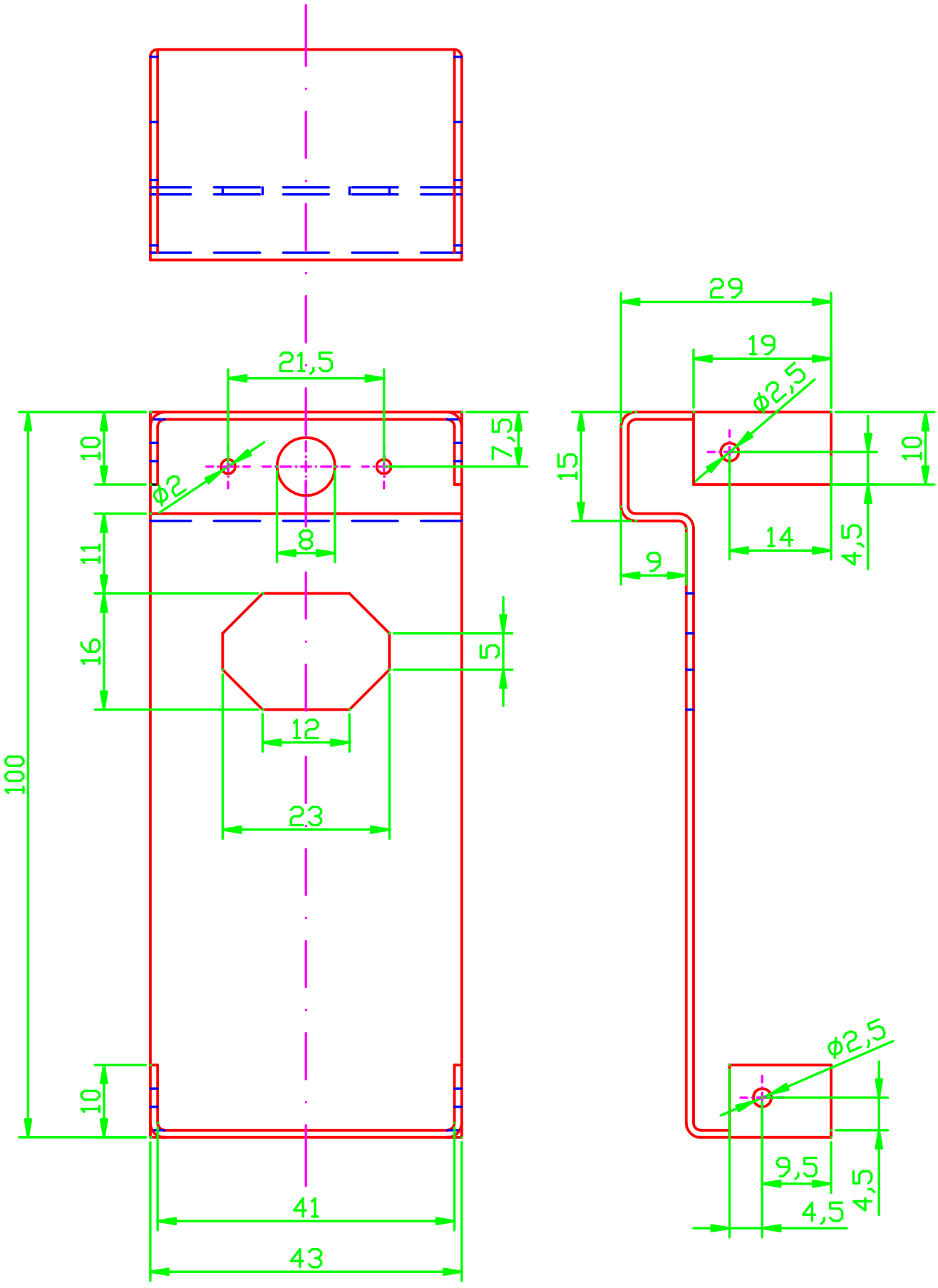
Gehäuse

Oberteil (Material: Aluminiumblech 1mm)

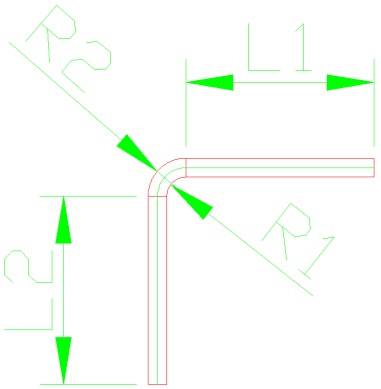


Die seitlichen Bohrlöcher dienen zur Verbindung zwischen Ober – und Unterteil. Diese Löcher gehören gemeinsam mit dem Unterteil gebohrt.

Unterteil (Material: Aluminiumblech 1mm)



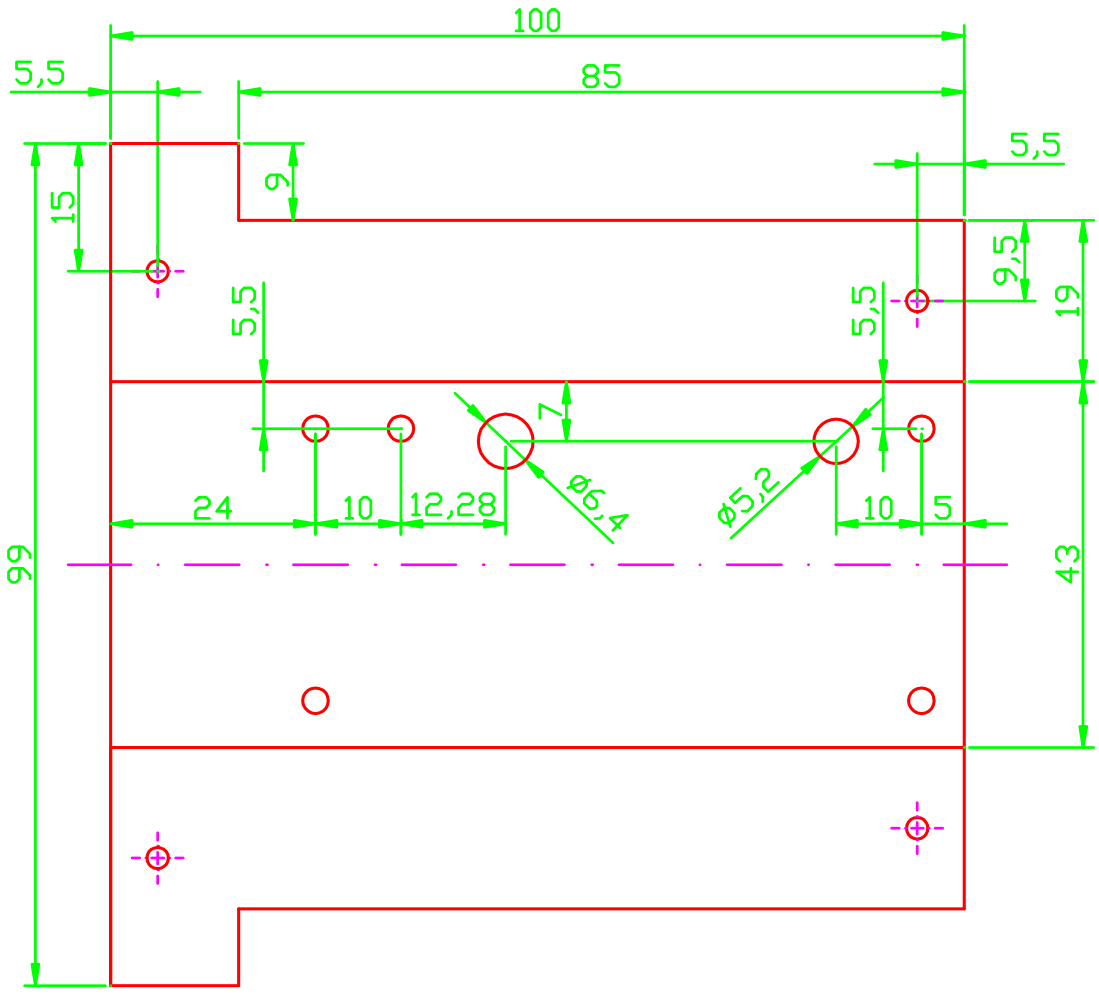
Materialberechnung fürs Biegen



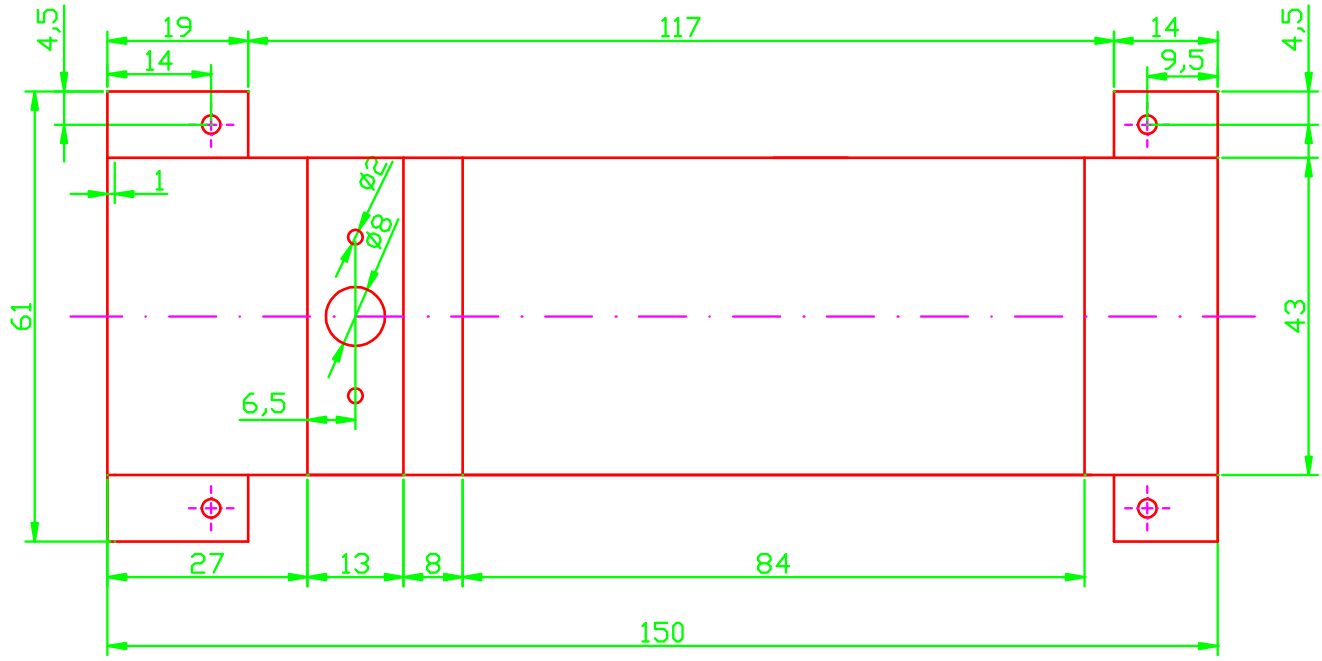
$$r = \frac{R2 + R1}{2}$$

$$wahre\ Länge = \frac{r \cdot \pi}{2} + L1 + L2$$

Auffaltungszeichnung Oberteil (Material: Aluminiumblech 1mm)



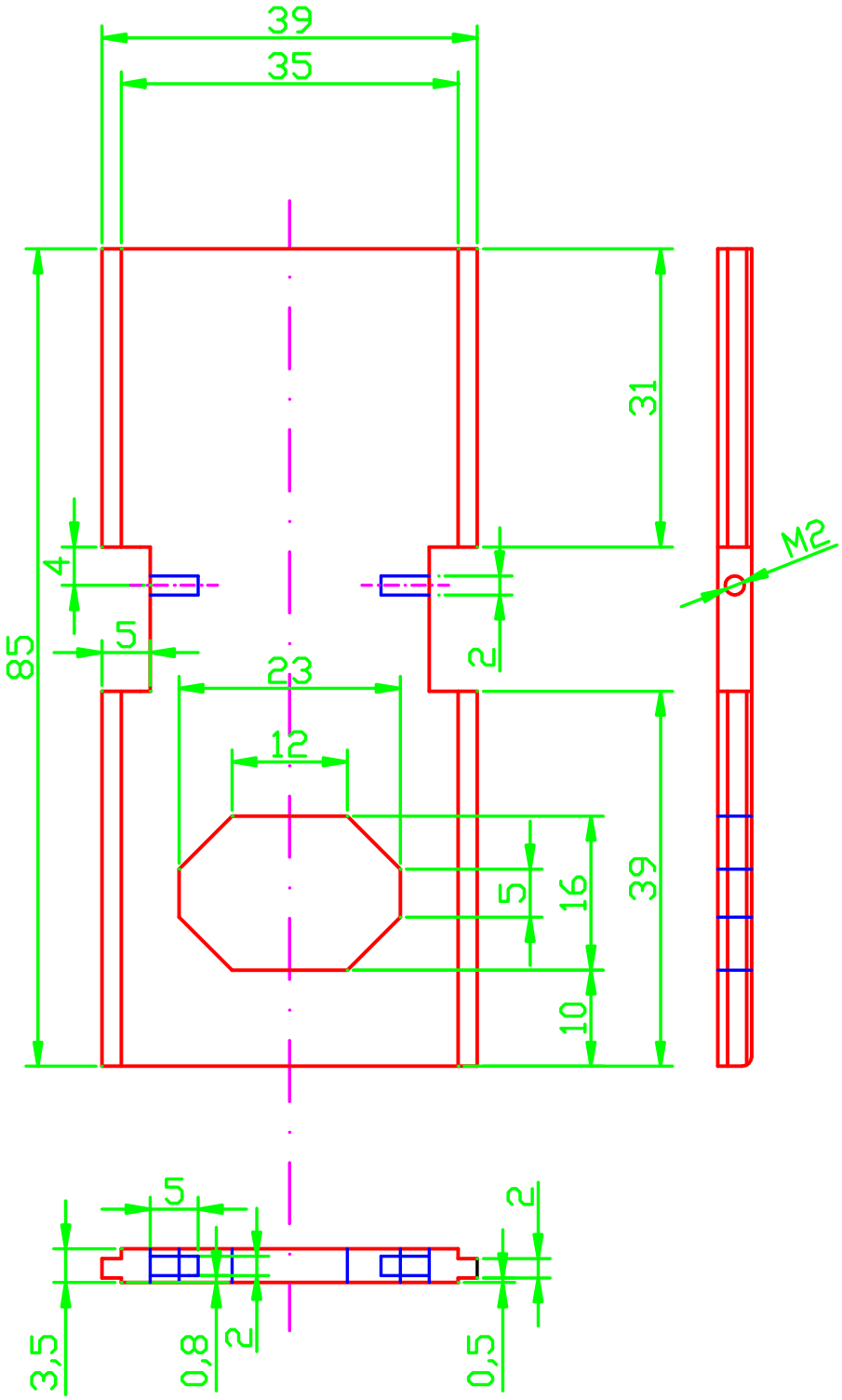
Auffaltungszeichnung Unterteil (Material: Aluminiumblech 1mm)



Stückliste des Slave Gehäuses

Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	1	Oberteil	Aluminiumblech	99x100x1
2	1	Unterteil	Aluminiumblech	61x150x1
3	6	Blechschrauben		2,9x5
4	4	Linsenkopfschrauben		M3x10
5	2	Senkkopfschrauben		M2x10
6	2	Mutter		M2
7	4	Mutter		M3

Einschubplatte (Material: Aluminium)

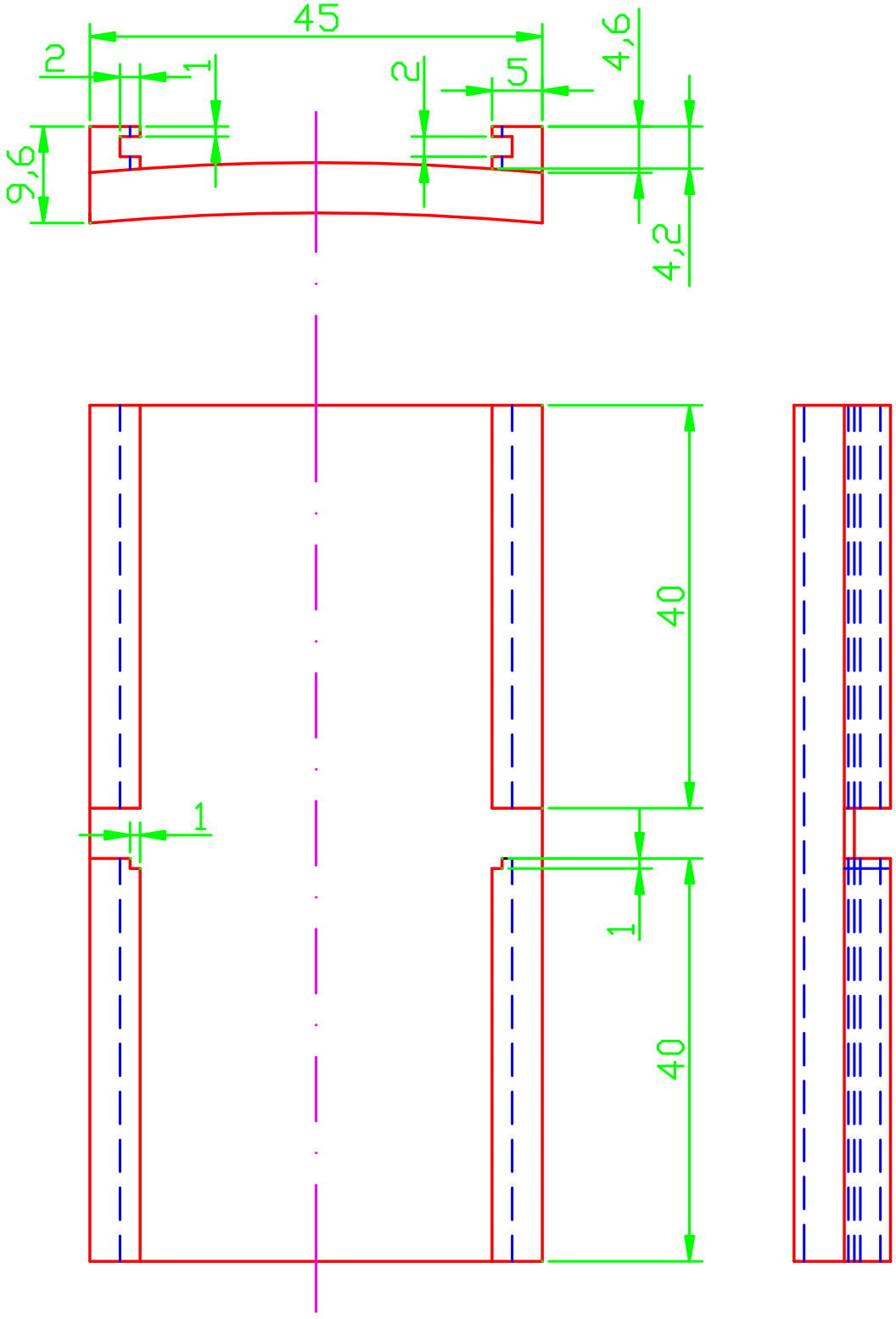


Stückliste der Einschubplatte

Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	1	Einschubplatte	Aluminium	39x85x3,5

Diese Einschubplatte wird auf den Unterteil des Slave Blechgehäuses geklebt. Dabei ist darauf zu achten, dass die Ausnehmung für den Spulenkörper erst im zusammengeklebten Zustand gefertigt wird.

Fixierleisten für den Magneten (Material: Aluminium)



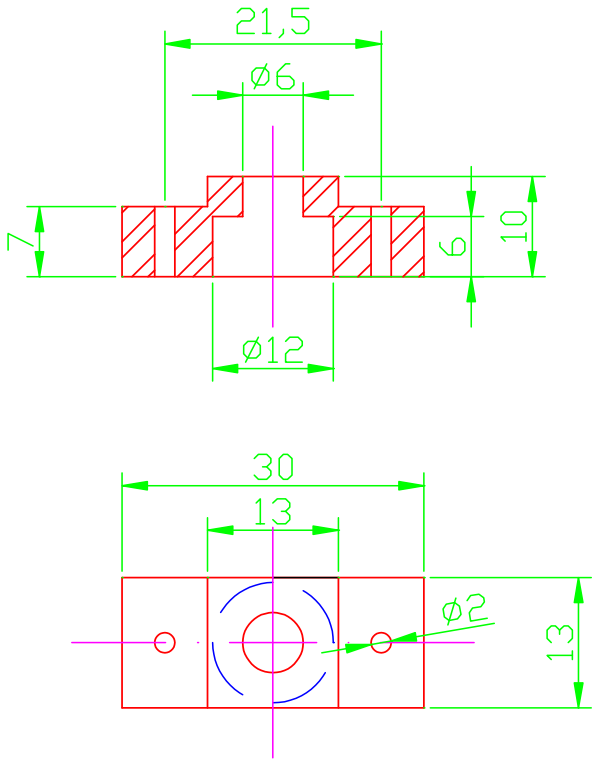
Stückliste der Fixierleisten

Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	4	Fixierleisten	Aluminium	5x40 x4,6

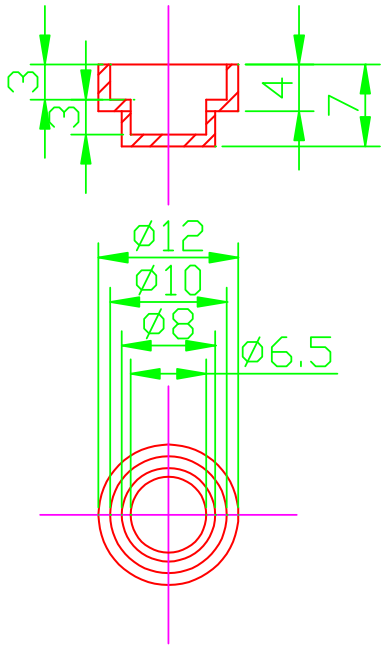
Diese Leisten sind zum Kleben auf einen Magneten gedacht und dienen als Einschubvorrichtung. Sie müssen aber nicht als Befestigung dienen sie können auch auf jede andere beliebige Oberfläche geklebt werden und somit als Halterung für den Slave dienen.

Temperaturabnehmer

Kunststoffhalterung für den Temperatursensor (Material: PVC)



Messingdrehteil zum Einkleben des Temperatursensor (Material: Messing)



Stückliste der Temperaturabnehmervorrichtung

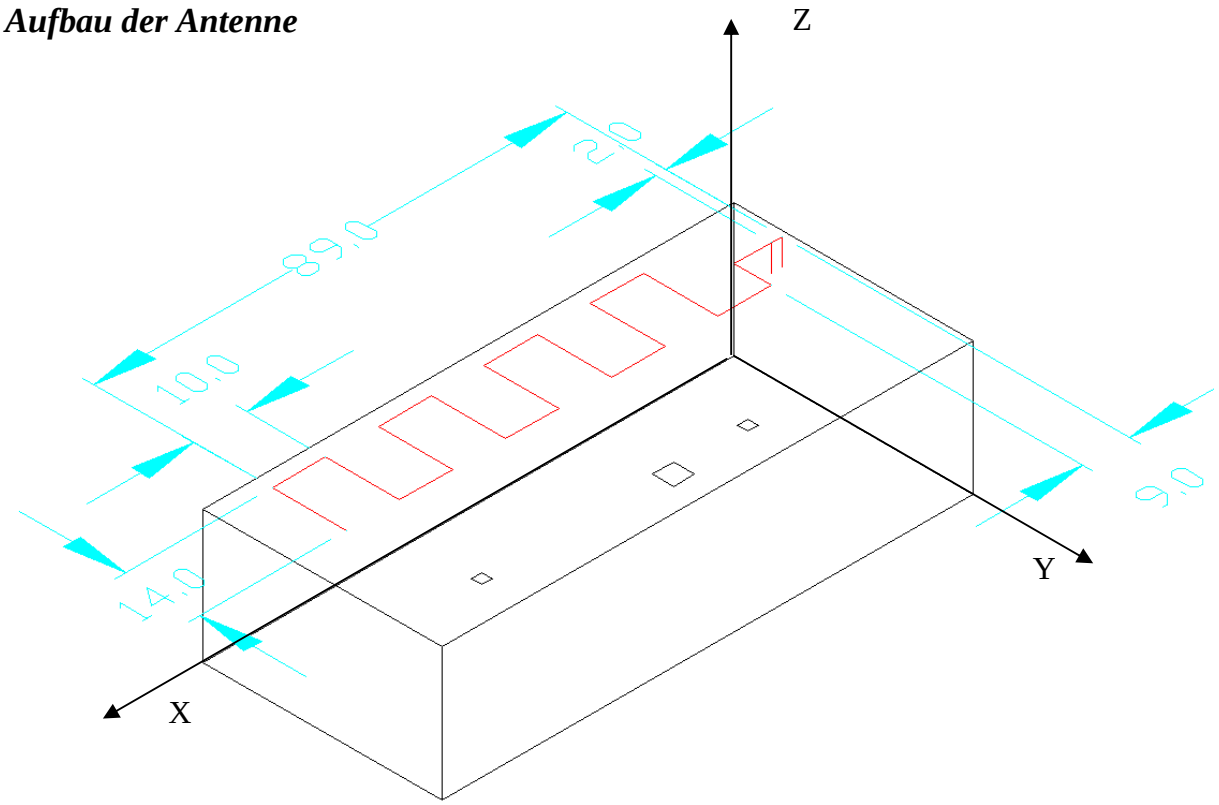
Nr.	Stk.	Typ	Material	Abmessungen
1	1	Temperaturabnehmerhalterung	PVC	30x13x10
2	1	Temperatursensorvorrichtung	Messing	ø12x7

In das Messingteil wird der Temperatursensor eingeklebt. Zwischen dem Messingteil und der Kunststoffhalterung wird eine Feder mit den Abmessungen ø10x5 positioniert.

Antenne

Für den Slave wurde eine eigene Antenne berechnet mit Hilfe der Berechnungssoftware ANTCAD. Weitere Berechnungen sind auf der DVD im Ordner „Antenne\AantcadSimulation“ zu finden.

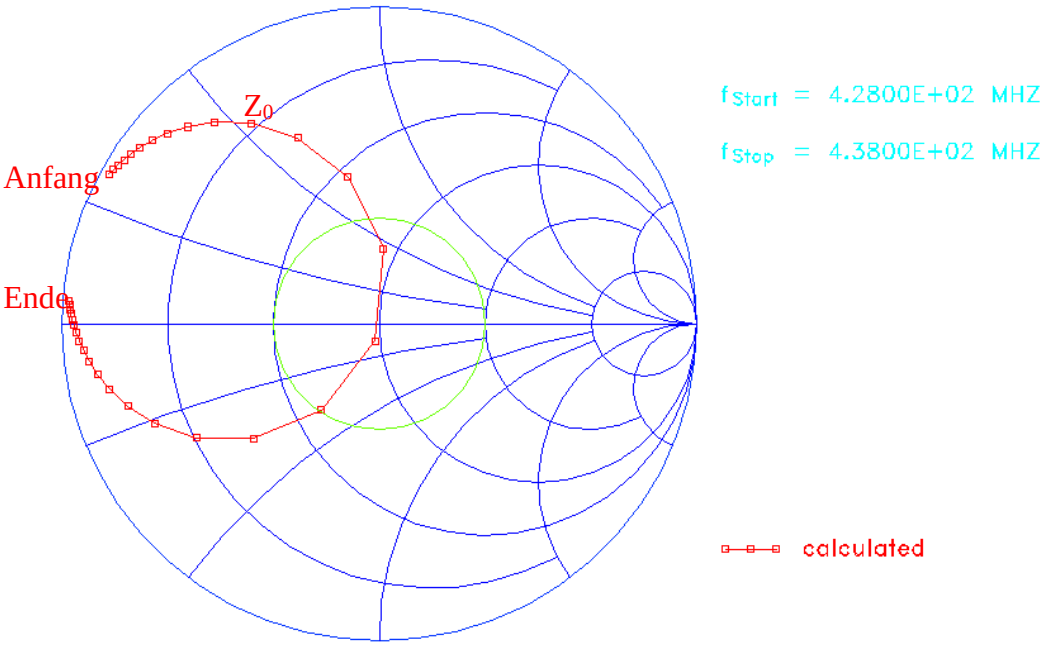
Aufbau der Antenne



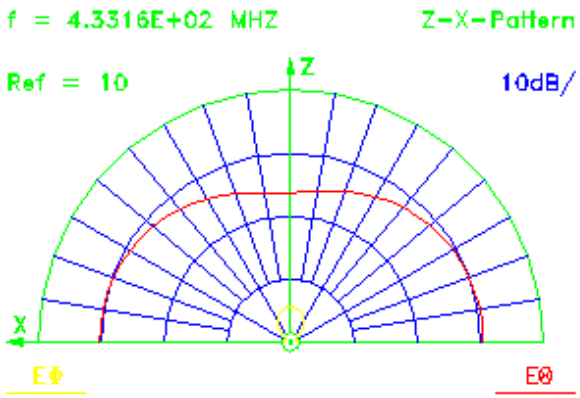
Die Antenne wird mit einem Abstand von 5mm zum Gehäuse angebracht.
Die Abbildung ist nicht im Maßstab! Sämtliche Angaben sind in mm!

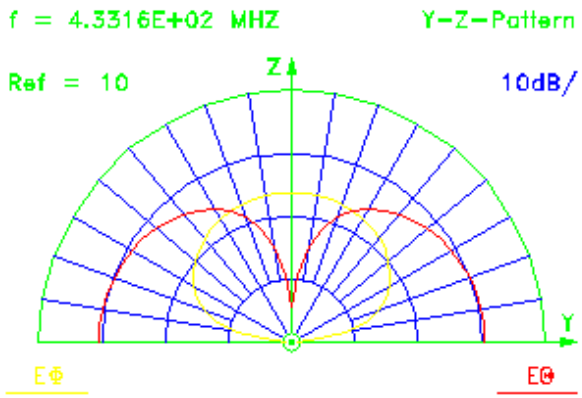
Berechnung der Antenne

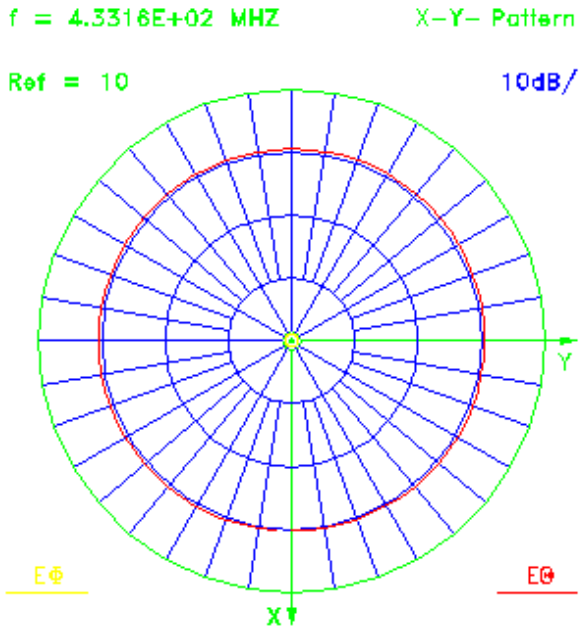
Ergebnisse der Berechnung mit ANTCAD:



Die Antenne hat bei 433MHz ihre Resonanzfrequenz und weist eine Bandbreite von 800 kHz auf. Die Richtcharakteristik der Antenne sieht bei 433MHz wie folgt aus:



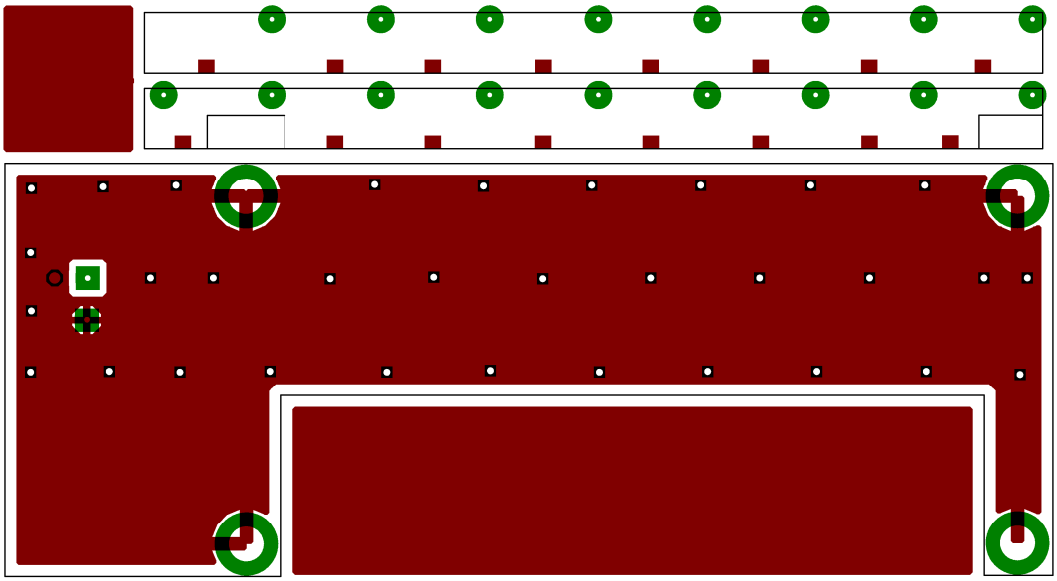




Realisierung der Antenne

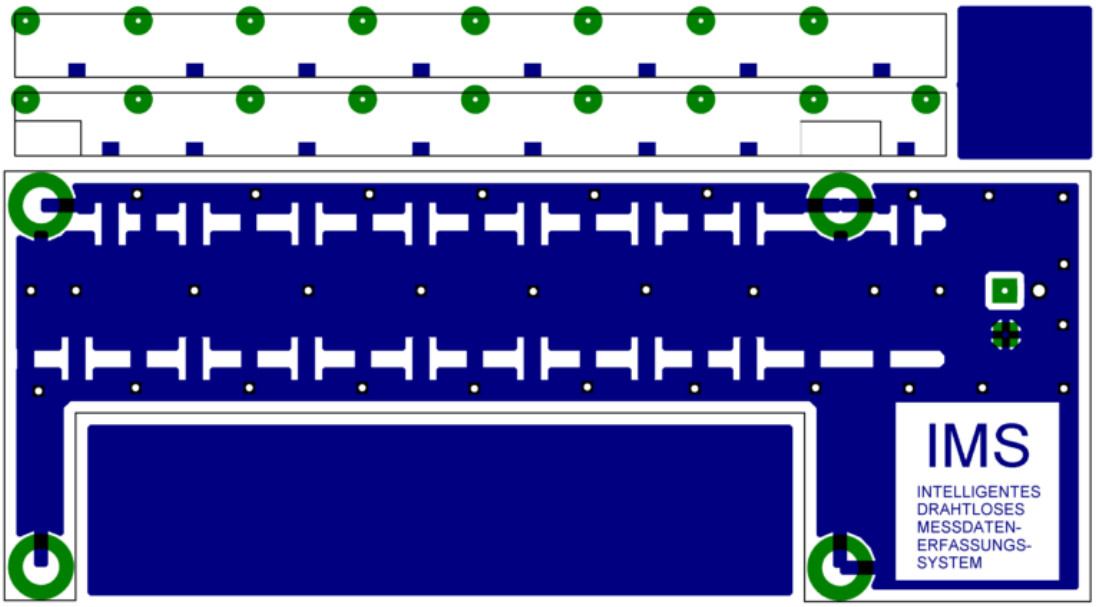
Die Antenne wurde auf ein FR4 – Platinenmaterial aufgebaut. Die Halterung aus dem Platinenmaterial sieht wie folgt aus:

Layout Antenne Top Layer



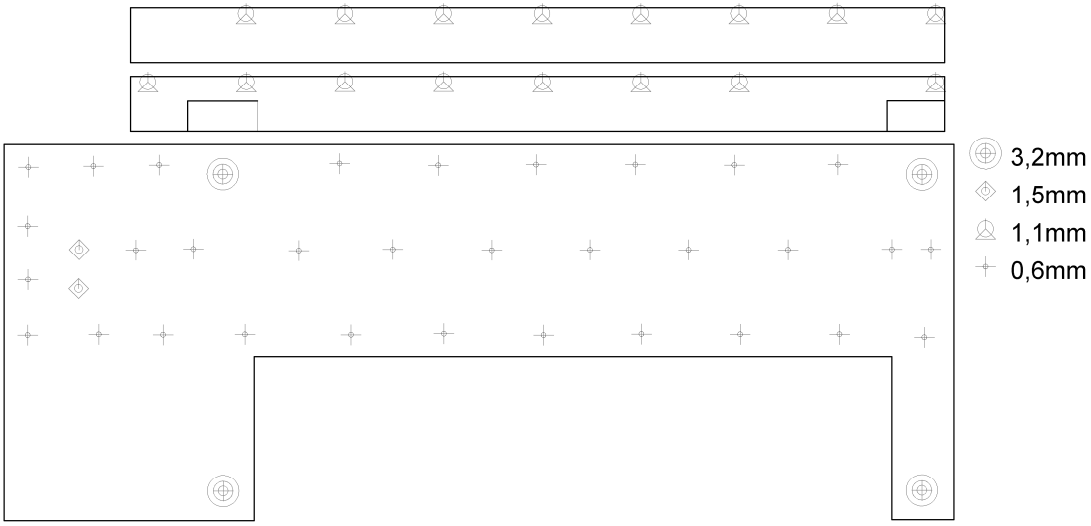
Top View
Abbildung nicht im Maßstab
Abmessungen der Platine mit den Stegen: 97mm x 53mm

Layout Antenne Bottom Layer



Bottom View
Abbildung nicht im Maßstab
Abmessungen der Platine mit den Stegen: 97mm x 53mm

Bohrplan Antenne



Top View
Abbildung nicht im Maßstab
Abmessungen der Platine mit den Stegen: 97mm x 53mm

Die Stege werden aufgestellt und auf die Bottom – Seite der Platine gelötet.
Diese Vorrichtung wird mit der Top – Seite auf das Gehäuse des Slaves aufliegend montiert.
In die Stege wird ein 1mm dicker Silberdraht nach der obigen Autocad – Zeichnung eingelötet.

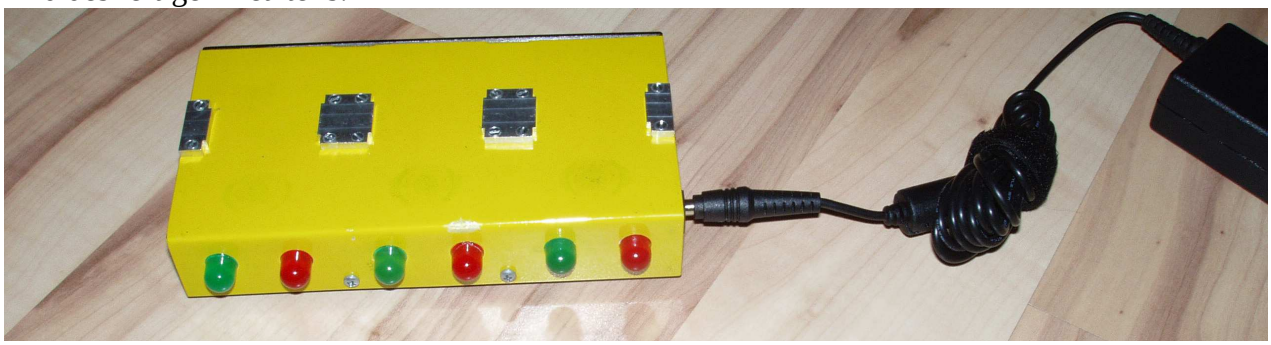
Abgleich der Antenne

Die Antenne muss abschließend noch auf 50Ω - Eingangsimpedanz bei einer Frequenz von 433MHz abgeglichen werden.
Das abgeglichene Bild sieht am Spektrumanalysator wie folgt aus:

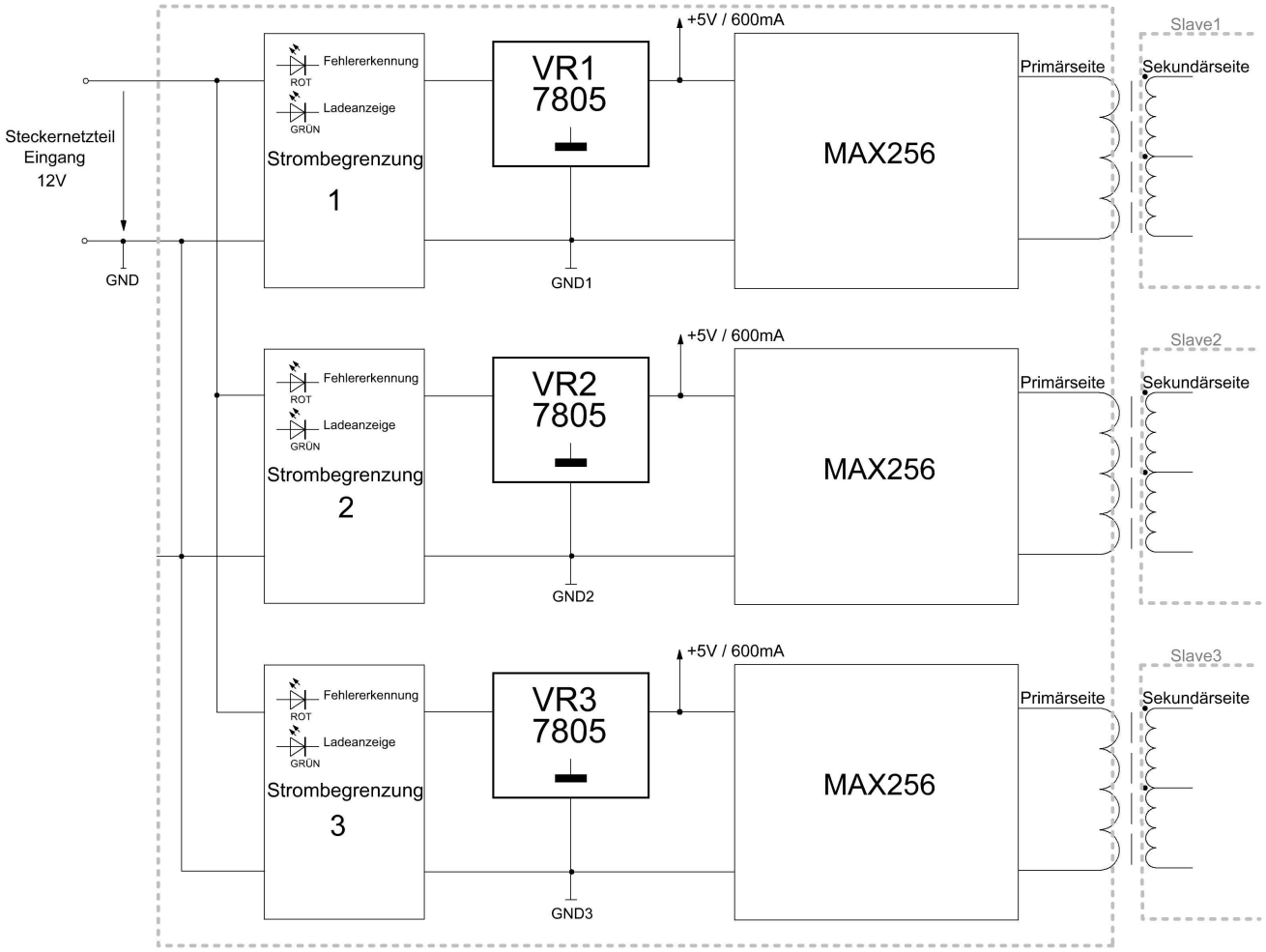
Der Abgleich erfolgt durch das Kürzen des Silberdrahtes.

5.1.4. Netzteil

Bild des fertigen Netzteils:

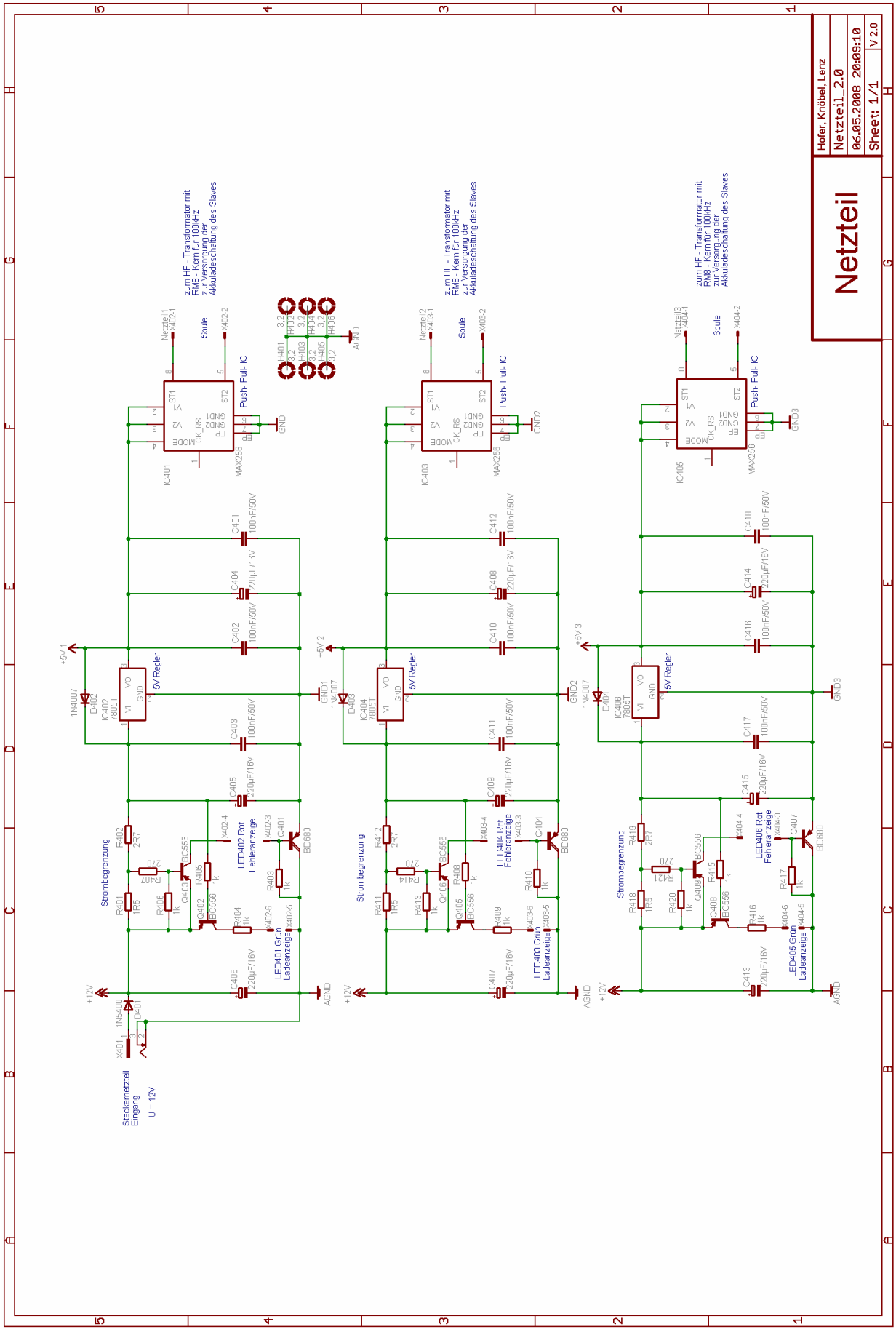


Blockschaltbild des Netzteils

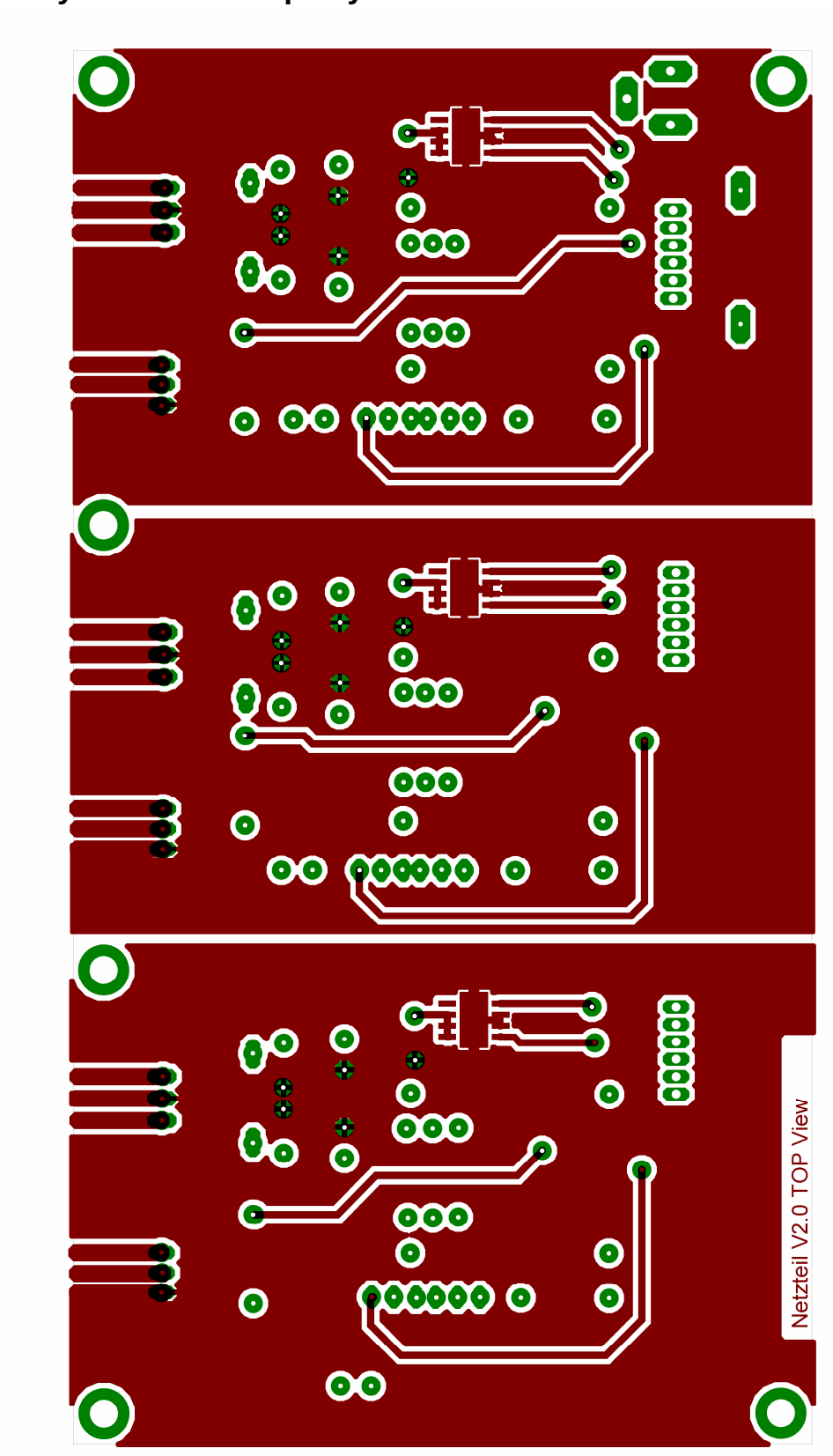


Detaillierte Informationen zum Netzteil sind im Kapitel 5.2.4. zu finden.

5.1.4.1. Schaltplan Netzteil

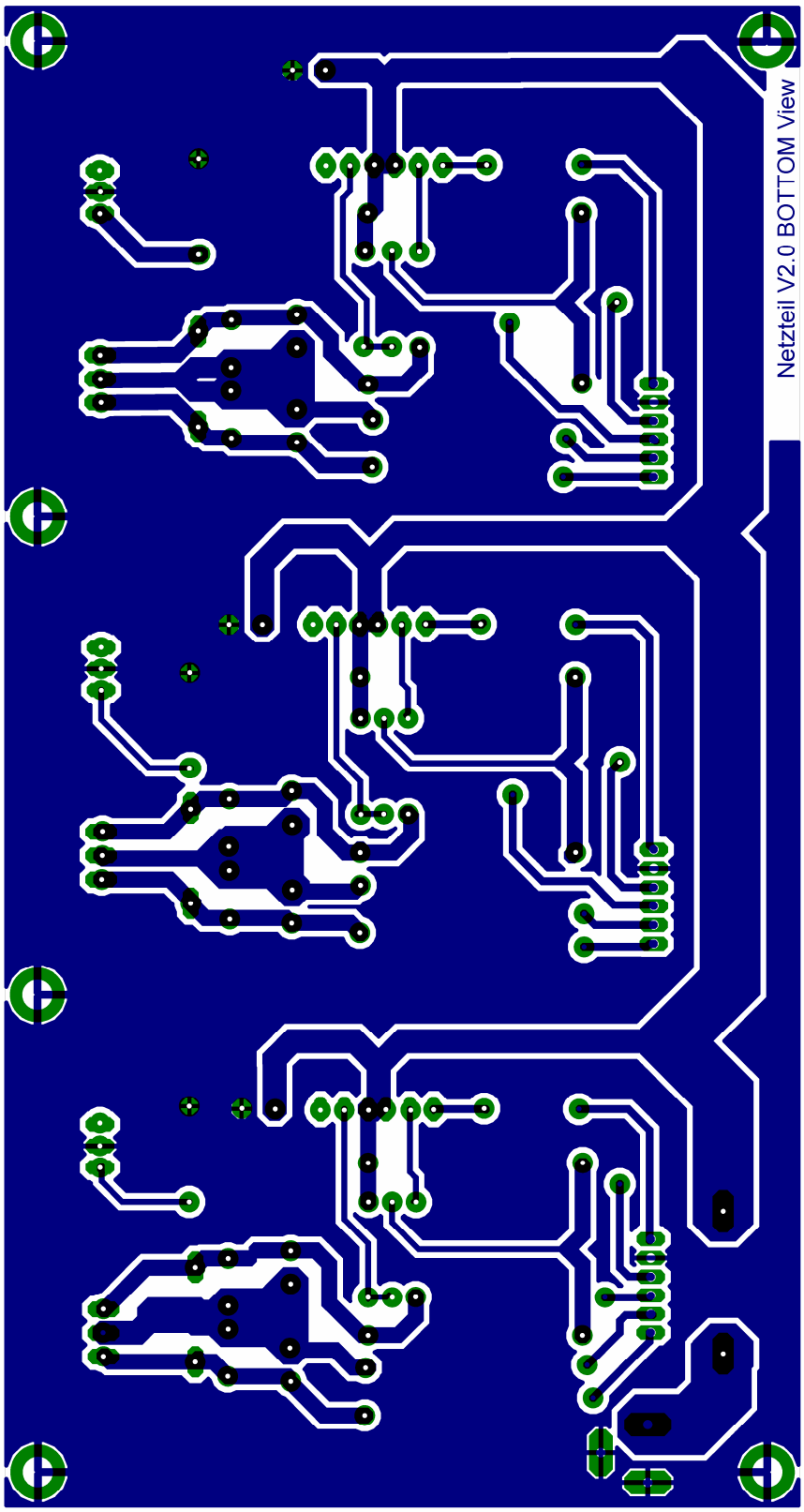


5.1.4.2. Layout Netzteil Top Layer



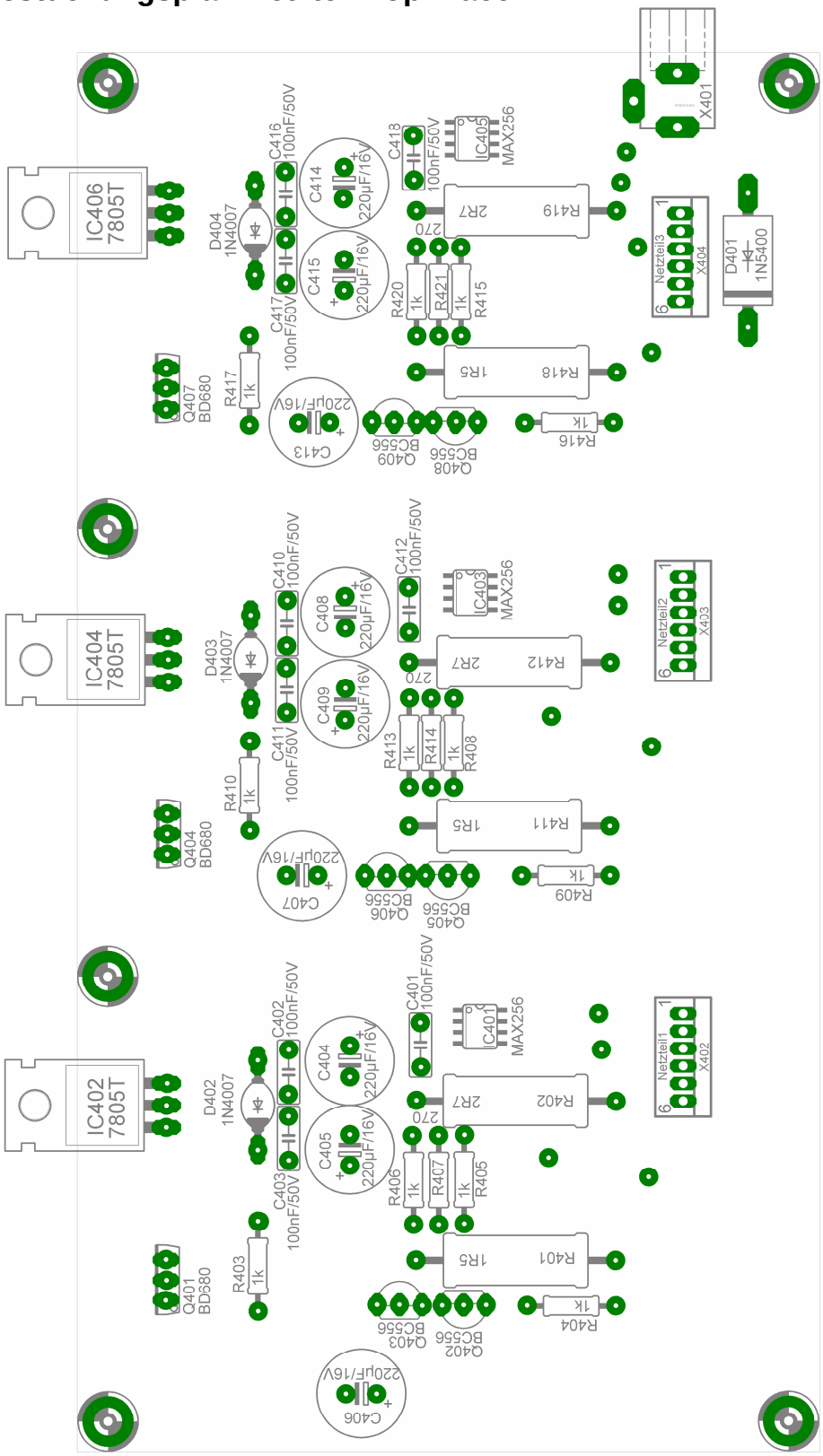
Top View
Abbildung nicht im Maßstab
Abmessungen: 160mm x 85mm

5.1.4.3. Layout Netzteil Bottom Layer



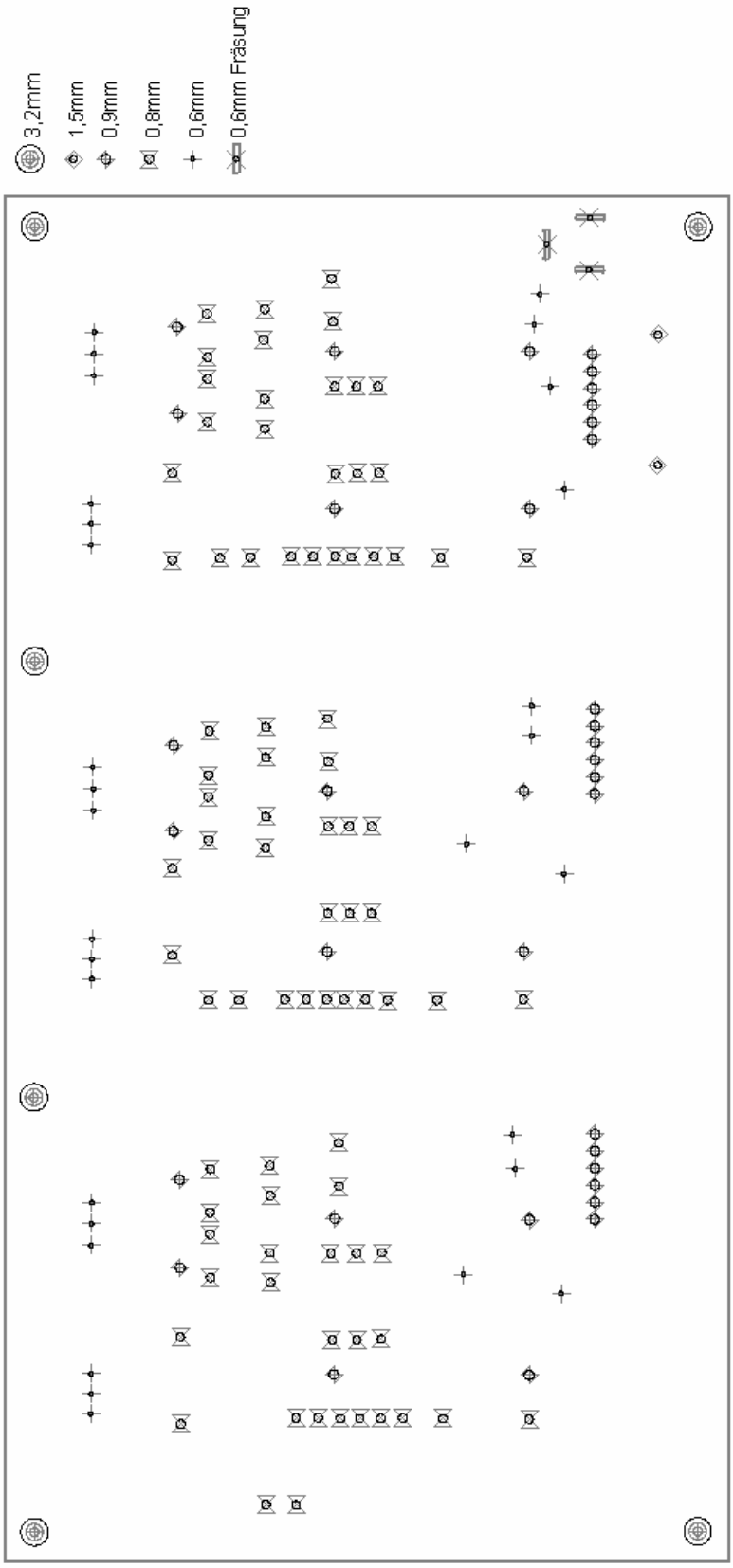
Bottom View
Abbildung nicht im Maßstab
Abmessungen: 160mm x 85mm

5.1.4.4. Bestückungsplan Netzteil Top Place



Top View
Abbildung nicht im Maßstab
Abmessungen: 160mm x 85mm

5.1.4.5. Bohrplan Netzteil



Top View
Abbildung nicht im Maßstab
Abmessungen: 160mm x 85mm

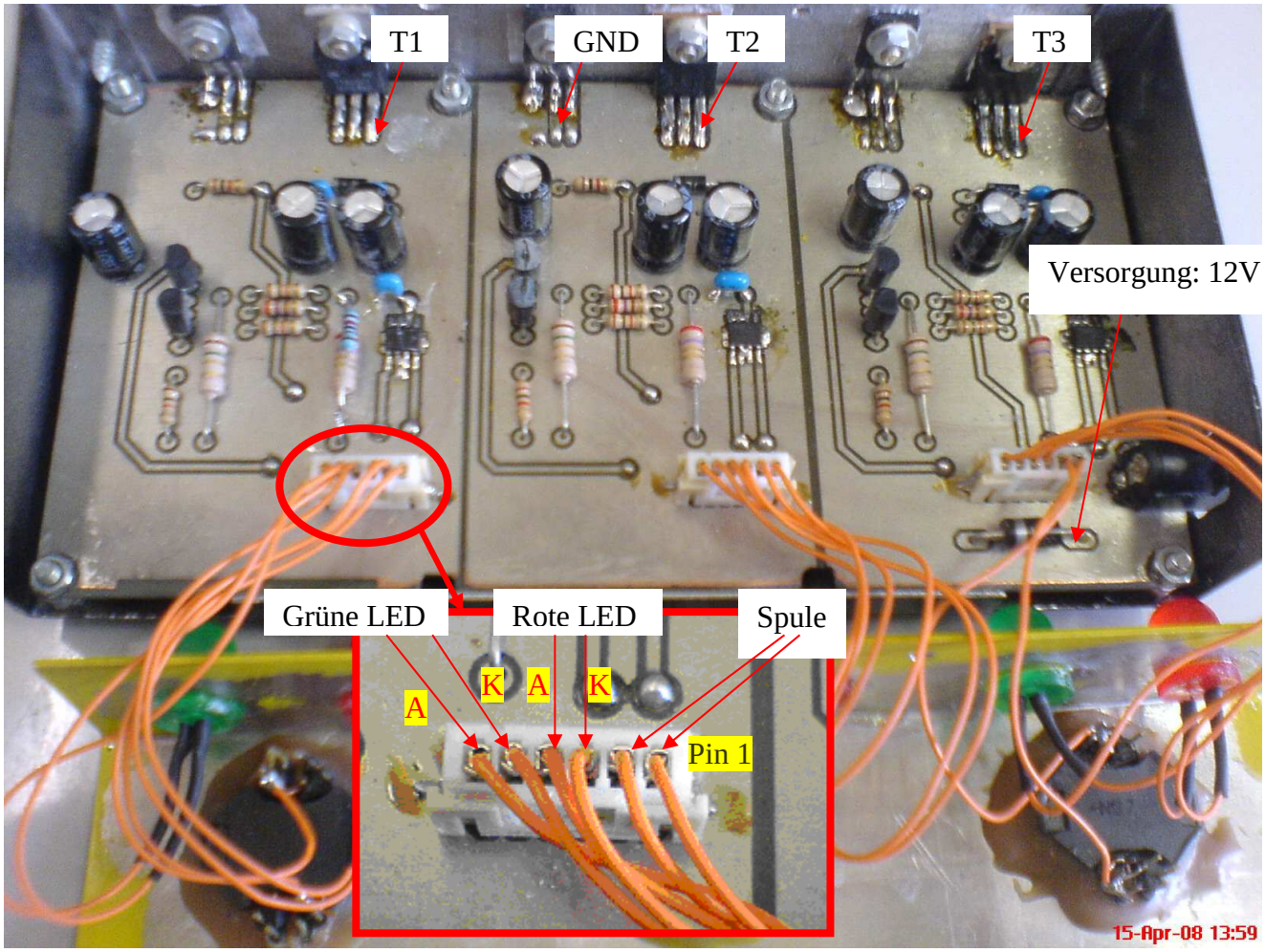
5.1.4.6. Stückliste Netzteil

Nr.	Stk.	Bez.	Typ	Wert	Bemerkung
1	3	Q401, Q404, Q407	PNP Darlingtontransistor	BD680	STM SOT-32
2	6	Q402, Q403, Q405, Q406, Q408, Q409	PNP Bipolartransistor	BC556	TO-92
3	3	LED401, LED403, LED405	Leuchtdiode Grün	Grün	RM2 (Ø11 x 2,5)
4	3	LED402, LED404, LED406	Leuchtdiode Rot	Rot	RM2 (Ø11 x 2,5)
5	1	D401	Diode	1N5400 50V / 3A	DO- 201
6	3	D402, D403, D404	Diode	1N4007 1000V/1A	DO-41
7	3	IC402, IC404, IC406	Spannungsregler	LM 7805/ 1A	TO-220
8	3	IC401, IC403, IC405	Push- Pull- IC	MAX256	SOT23-6
9	9	C401, C402, C403, C410, C411, C412, C416, C417, C418	Keramikkondensator	100nF ±20% / 50V	RM5 (5,08 x 5,08)
10	9	C404, C405, C406, C407, C408, C409, C413, C414, C415	Elektrolytkondensator	220µF ±20% / 16V	RM3 (11,5 x 3,5)
11	12	R403, R405, R406, R408, R410, R413, R415, R417, R420	Widerstand	1kΩ ±5% / 1/4W	0207
12	3	R407, R414, R421	Widerstand	270Ω ±5% / 1/4W	0207
13	3	R402, R412, R419	Widerstand	2,7Ω ±5% / 1W	0414
14	3	R401, R411, R418	Widerstand	1,5Ω ±5% / 1W	0414
15	1	X401	Niedervolt Buchse	für Steckernetzteil	733980
16	3	X402, X403, X404	Stiftleiste stehend 6-polig	6-polig STM	RM2(13,9 x 2)
17	3	X402, X403, X405	Crimpleergehäuse 6-polig	6-polig	RM2(13,8 x 2)
18	18		Crimpkontakte	Kontakte	0,05mm² - 0,22mm²

5.1.4.7. Inbetriebnahme des Netzteils

Nach dem vollständigen bestücken der Platine muss sie noch getestet werden. Wie die Schaltung überprüft werden kann, wird in diesem Kapitel beschrieben.

Übersicht der Testpunkte



Testen des Ruhestroms und der Spannungen

Zuerst muss überprüft werden ob der Ruhestrom in einem zulässigen Bereich liegt. Mit dieser Überprüfung kann festgestellt werden ob Kurzschlüsse in der Schaltung vorhanden sind.

Zum Testen des Ruhestroms werden alle LEDs und Spulen nicht angeschlossen.
Weiters wird die Schaltung mit einem Labornetzteil versorgt. Dieses muss auf **12V** und **50mA** Strombegrenzung eingestellt werden. Die Versorgungsspannung kann über die Buchse oder den oben gezeigten Testpunkten zugeführt werden.
Es wird eine **Stromaufnahme** von **20mA** erwartet.

Ist das der Fall können noch die Spannungen überprüft werden.
An den Testpunkten **T1, T2** und **T3** sollte jetzt eine Spannung von **5V** gegen GND gemessen werden.

Testen der LEDs

Zum Testen der LEDs müssen alle LEDs angeschlossen werden.

Achtung: Die rote LED muss angeschlossen sein, damit die Strombegrenzung funktioniert. Wenn die rote LED nicht angeschlossen wird, gibt es keine Strombegrenzung!

Der folgende Testvorgang muss mit jeder der drei Stufen durchgeführt werden:

An den Kontakten der Spule muss ein 50Ω - Potentiometer angeschlossen werden.
Bei diesem Test wird die Schaltung ebenfalls mit einem Labornetzteil versorgt.
Dieses muss auf **12V** und **700mA** Strombegrenzung eingestellt werden.
Die Versorgungsspannung kann über die Buchse oder den oben gezeigten Testpunkten zugeführt werden.

Der Widerstand des Potentiometers wird so lange verringert bis die grüne LED der Stufe zum Leuchten anfängt. Dabei sollte die Schaltung einen Strom im Bereich von **160mA** verbrauchen.

Wenn dies der Fall ist kann der Widerstand so weit verringert werden bis die rote LED zum Leuchten anfängt. Sobald die rote LED leuchtet, sollte die Schaltung einen Strom im Bereich von **500mA bis 580mA** verbrauchen.
Der Strom sollte bei einer weiteren Verringerung des Widerstandes nicht weiter ansteigen. Wenn dies nicht der Fall ist, ist die Strombegrenzung defekt.
Achtung: Der Test muss bei einer Stromaufnahme von 600mA abgebrochen werden, da es sonst zur Beschädigung von Bauelementen kommen kann.

Test mit angeschlossener Spule

Zuletzt werden noch die Spulen angeschlossen. Die Schaltung kann für die weiteren Tests mit dem zugekauften Steckernetzteil betrieben werden.
Wenn die Spule ohne Kern verwendet wird sollen beide LEDs der angeschlossenen Stufe leuchten.
Wenn die Spule mit Kern verwendet wird, sollte keine der beiden LEDs leuchten.

5.1.4.8. Zustandsanzeige

Die rote und die grüne LED des Netzteils zeigen dessen Zustand an.

Sobald das Netzteil an das Stromnetz angeschlossen wird leuchten kurz alle LED auf.
Damit kann überprüft werden, ob alle LED noch in Ordnung sind.

Wenn in einer Ladezelle die Strombegrenzung einsetzt, beginnen die grüne und die rote LED der jeweiligen Zelle zu leuchten.
Sobald dieser Fall eintritt, ist zu überprüfen, ob ein metallischer Gegenstand auf die Spule der induktiven Kopplung liegt, ist dies der Fall so ist der metallische Gegenstand zu entfernen, andernfalls ist das Netzteil vom Stromnetz zu trennen und der Zustand des Netzteils zu überprüfen.

Die grüne LED leuchtet immer dann wenn ein Akku eines Slaves geladen wird.
Die grüne LED kann aber auch aufleuchten wenn sich ein metallischer Gegenstand auf die Spule der induktiven Kopplung des Netzteils befindet, ist dies der Fall, so ist der Gegenstand vom Netzteil zu entfernen.

Verhalten der grünen LED wenn ein Akku geladen wird und dessen Bedeutung:

Grüne LED leuchtet
Der Akku ist nahezu entladen und muss nun wieder vollständig geladen werden.

Grüne LED blinkt schnell (mehrmals pro Sekunde)
Der Akku ist mittelmäßig entladen und wird nun wieder geladen.

Grüne LED blinkt langsam (ca. alle drei Sekunden)
Der Akku ist bald vollständig geladen und kann bald wieder verwendet werden.

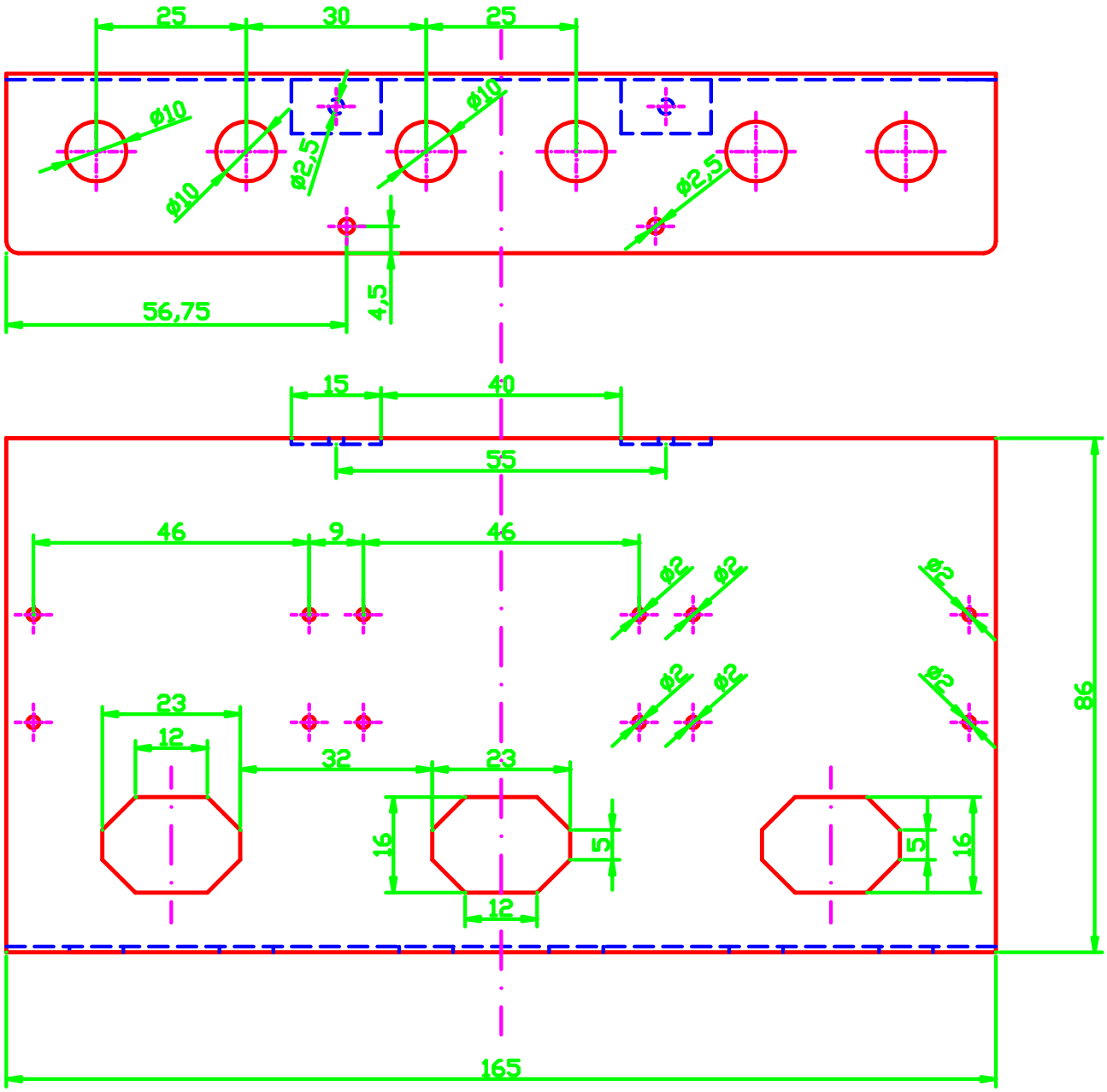
Grüne LED blinkt sehr langsam (ca. alle zehn Sekunden)
Der Akku ist vollständig geladen und der Slave kann wieder verwendet werden.

5.1.4.9. Gehäuse Netzteil

Sämtliche Abbildungen sind nicht im Maßstab! Alle Abmessungen sind in mm!
Alle Biegungen haben einen Innenradius von 1mm und einen Außenradius von 2mm.

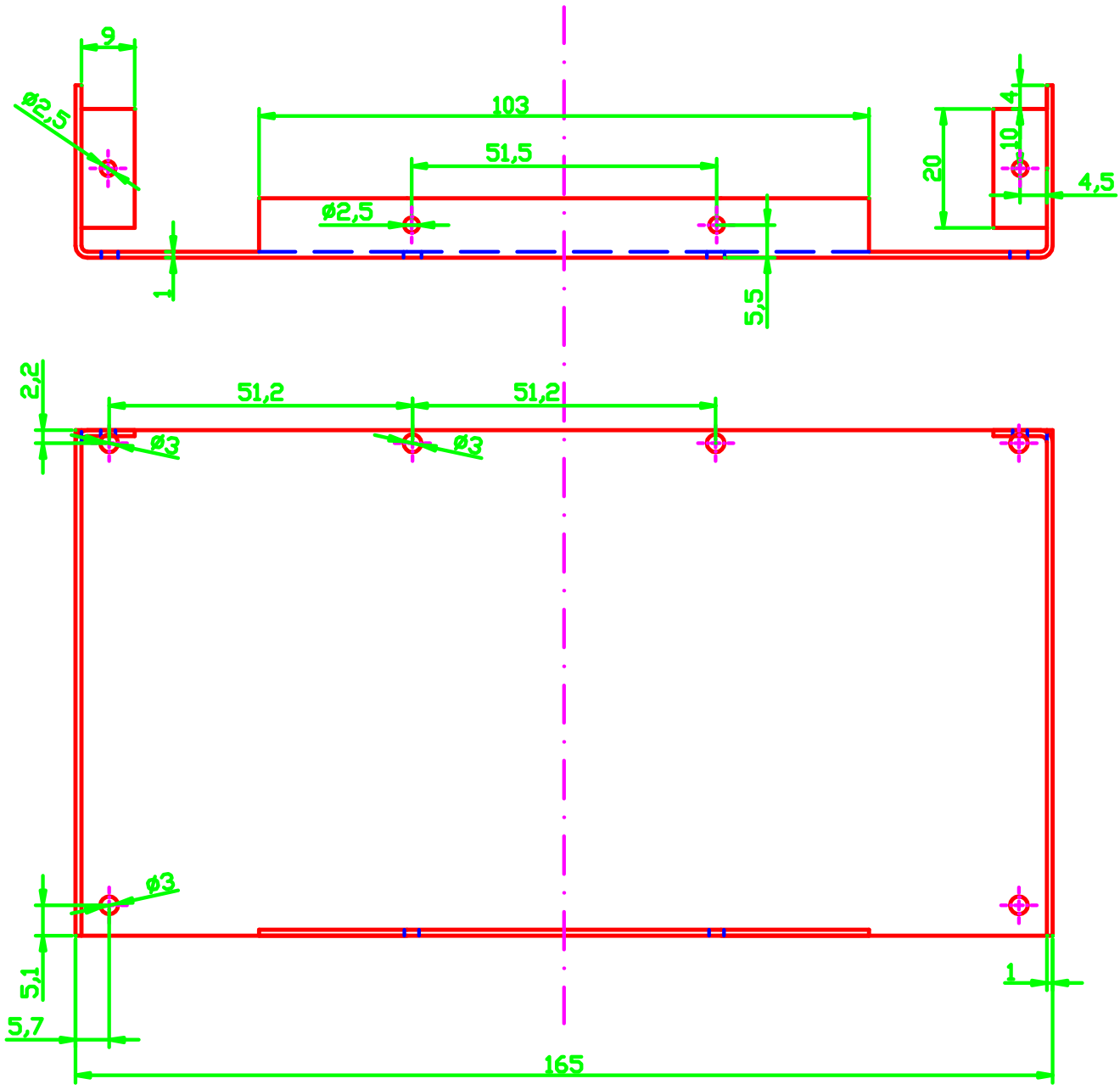
Gehäuse

Oberteil (Material: Aluminiumblech 1mm)

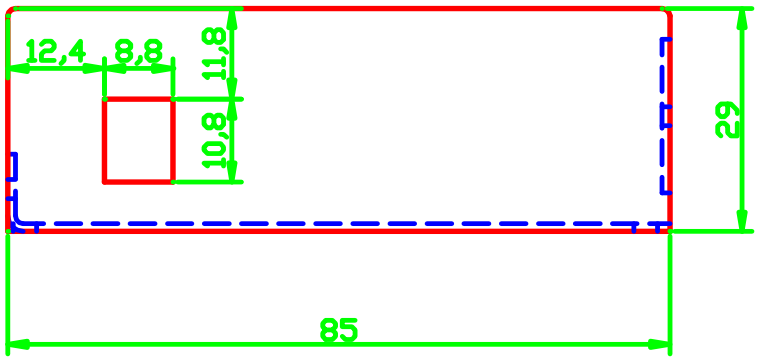


Kreuzriss

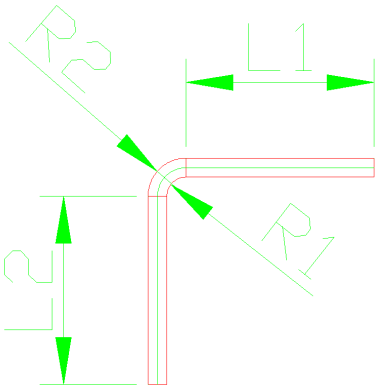
Unterteil (Material: Aluminiumblech 1mm)



Kreuzriss



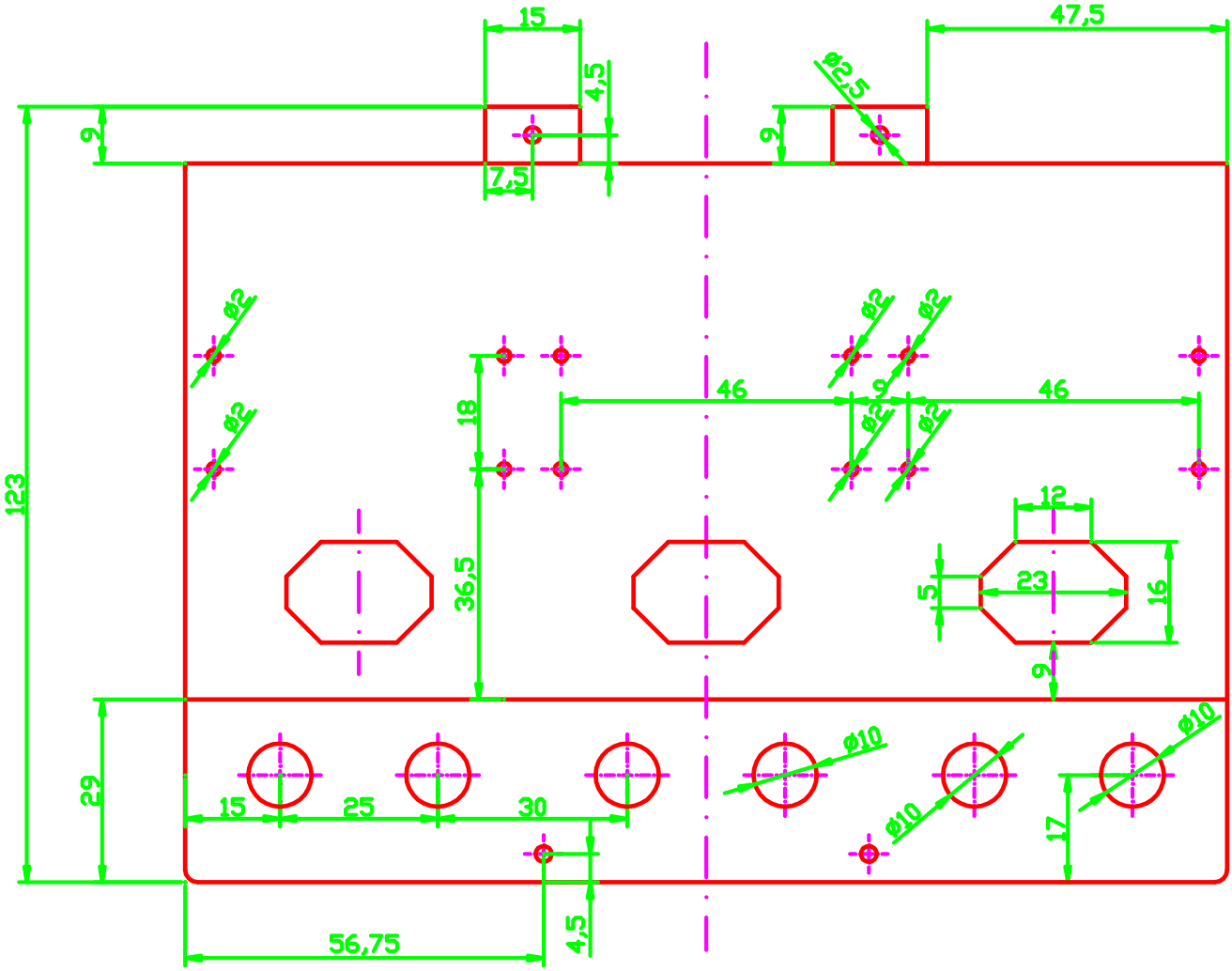
Materialberechnung fürs Biegen



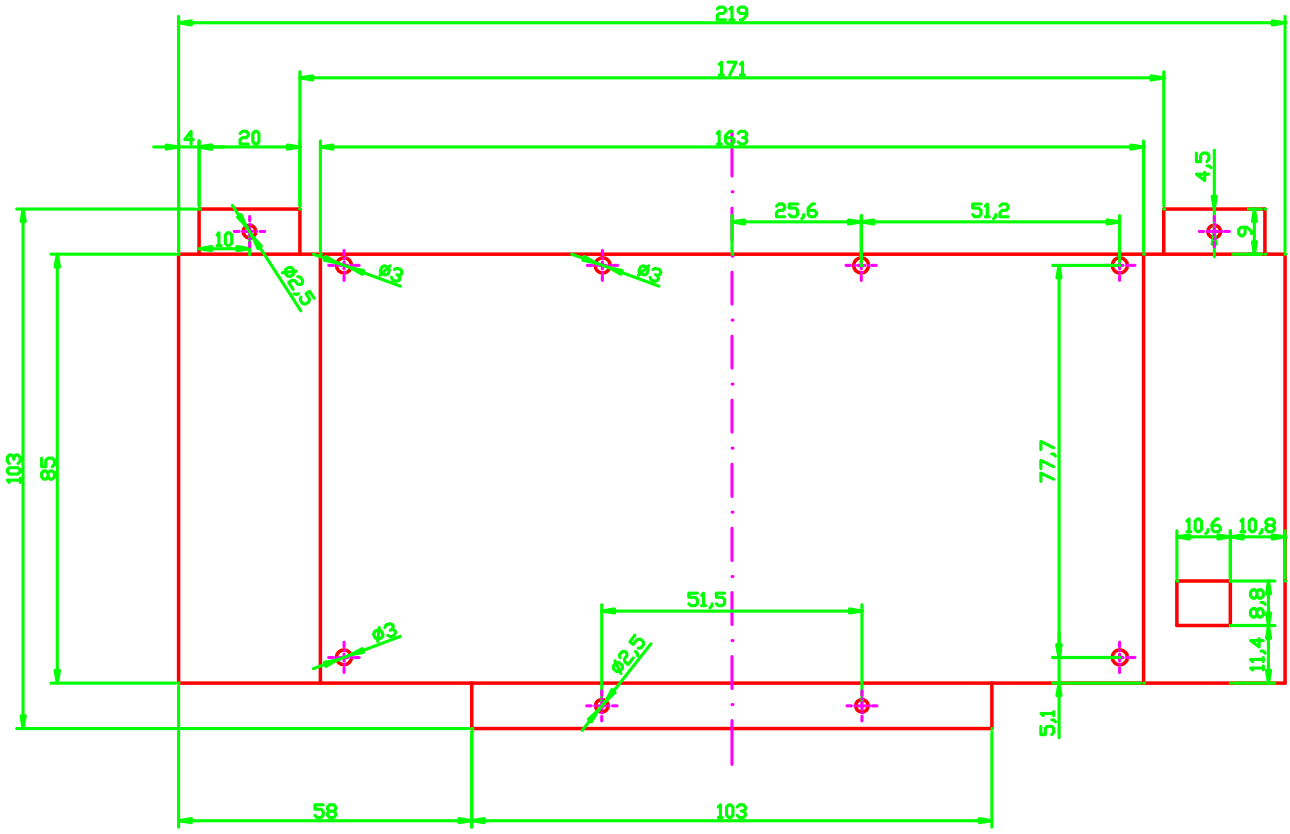
$$r = \frac{R2 + R1}{2}$$

$$wahre\ Länge = \frac{r \cdot \pi}{2} + L1 + L2$$

Auffaltungszeichnung Oberteil (Material: Aluminiumblech 1mm)



Auffaltungszeichnung Unterteil (Material: Aluminiumblech 1mm)

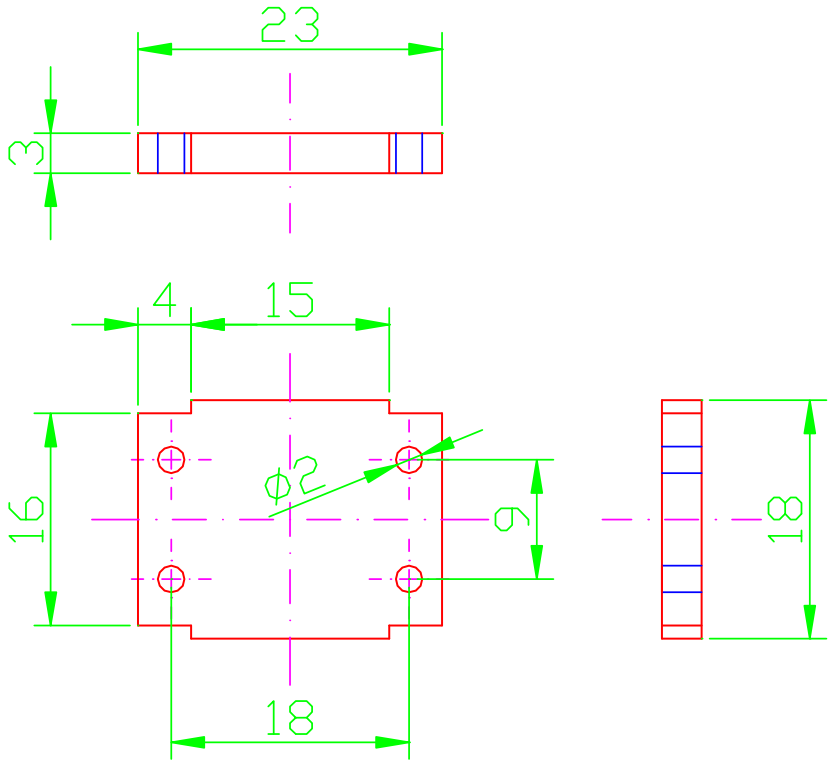


Stückliste des Netzteils

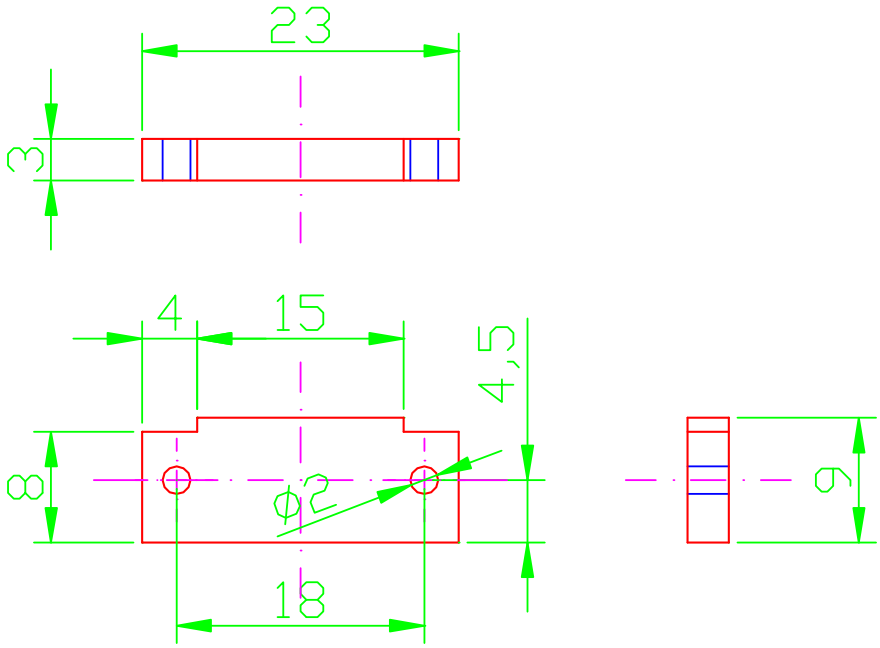
Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	1	Unterteil	Aluminiumblech	219x103x1
2	1	Oberteil	Aluminiumblech	165x123x1
3	1	Rückwand	Aluminiumblech	40x165x2
4	12	Senkkopfschrauben		M2x5
5	6	Blechschrauben		2,9x5
6	6	Linsenkopfschrauben		M3x10
7	6	Mutter		M3

Fixierungen

Mittlere Fixierung (Material: Aluminium)



Seitliche Fixierung (Material: Aluminium)



Stückliste der Fixierungen des Netzteils

Nr.	Stk.	Typ	Material	Abmessungen [mm]
1	2	Seitliche Fixierungen	Aluminium	18x23x3
2	2	Mittlere Fixierungen	Aluminium	9x23x3

5.2. Realisierung

5.2.1. PC- Software

5.2.1.1. Allgemein

Wahl der Programmiersprache

Hier wird erläutert aus welchen Gründen das Programm in C#.NET entwickelt wurde und warum die Framework 2.0 verwendet wurde.
Um eine Auswahl zu treffen, müssen zuerst die Anforderungen an die Programmiersprache erläutert werden:

Anforderungen

Hier folgt eine Übersicht der wichtigsten Eigenschaften, welche die Programmiersprache benötigt um eingesetzt werden zu können:

- Schnelles abarbeiten von Befehlen
- Viele vordefinierte Funktionen und Elemente um das Programmieren zu erleichtern
- Einfachere Objekt - und Klassenverwaltung
- Übersichtliche Quellcodes zum besseren Einlesen
- Möglichkeit zum Multithreading

Mögliche Sprachen die in Frage kamen

Hier eine Liste der bekannten Programmiersprachen die in Frage kamen. Dabei wurde darauf geachtet, dass nicht zu viel Einarbeitungszeit benötigt wird um die Programmiersprache zu verwenden:

- **Visual Basic**
  - Vorteile
    - Diese Programmiersprache hat den großen Vorteil, dass sie sehr leicht zu erlernen ist und der Quellcode nicht allzu kompliziert aufgebaut ist
    - Das VB.NET bietet auch viele Funktionen und Elemente, um den Programmierer zu unterstützen
  - Nachteile
    - Die Programmiersprache ist nicht für komplexe Programme geeignet, da sie bei längeren Quellcodes sehr unübersichtlich ist
    - Das Arbeiten mit Objekten ist nur eingeschränkt möglich. Die Syntax ist für komplexe Zeigerzugriffe nur sehr schlecht geeignet
    - Weiters ist die Performance nicht sehr hoch, da Visual Basic eine Scriptsprache ist und erst während der Laufzeit kompiliert wird
- **Java**
  - Vorteile
    - Die Programmiersprache ist plattformunabhängig und somit gut für das Projekt geeignet
    - Der Quellcode kann sehr übersichtlich gestaltet werden
    - Die Klassen - und Objektverwaltung ist sehr einfach
    - Die Performance der Programmiersprache ist im Vergleich zu anderen Programmiersprachen sehr hoch und für zeitkritische Programme gut geeignet
  - Nachteile

        - Aus fehlenden Kenntnissen über die Programmiersprache, ist eine sehr lange Einarbeitungsphase zu erwarten
- **C++**
    - Vorteile
        - Der Quellcode kann sehr übersichtlich gestaltet werden
        - Die Klassen - und Objektverwaltung ist sehr einfach
    - Nachteile
        - Aus fehlenden Kenntnissen über die Programmiersprache, ist eine sehr lange Einarbeitungsphase zu erwarten
- **C#**
    - Vorteil
        - Der Quellcode kann sehr übersichtlich gestaltet werden
        - Die Klassen - und Objektverwaltung ist sehr einfach
        - Das C#.NET bietet auch viele Funktionen und Elemente, um den Programmierer zu unterstützen
        - Möglichkeit zum Multithreading ist vorhanden
        - Für diese Programmiersprache sind tiefgehende Kenntnisse vorhanden
    - Nachteile
        - Die Programmiersprache ist im Durchschnitt zu anderen Programmiersprachen etwas langsamer, da jede Variable als Objekt deklariert wird

Auswahl der Programmiersprache

Nach längerem Überlegen hatte sich die Programmiersprache C#.NET als am optimalsten herausgestellt.
Es wurde schlussendlich die Programmiersprache mit der kürzesten Einarbeitungszeit ausgewählt. Dafür musste jedoch ein Performance Nachteil in Kauf genommen werden.

Auswahl der Entwicklungsumgebung

Weiters gab es noch zu entscheiden in welcher Programmierumgebung das Programm geschrieben werden soll:

- Microsoft Visual Studio 2003
    - Diese Programmierumgebung verwendet die Framework 1.1
    - Vorteile
        - Die Framework 1.1 wird ab Windows XP SP2 automatisch unterstützt
    - Nachteile
        - Es sind viele Bugs und einige wichtige Funktionen die im Projekt verwendet werden nicht unterstützt
    - Die Umgebung unterstützt keine Änderung der Quellcodes während der Laufzeit
- Microsoft Visual Studio 2005
    - Diese Programmierumgebung verwendet die Framework 2.0
    - Vorteil
        - Es sind nahezu alle Funktionen vorhanden die im Projekt benötigt werden
    - Nachteil
        - Die Framework 2.0 wird erst ab Windows Vista standardmäßig mitgeliefert. Sonst muss sie installiert werden
    - Es ist in dieser Umgebung auch möglich den Quellcode während der Laufzeit zu verändern
    - Weiters bietet die Entwicklungsumgebung viele verbesserte Funktionen zum erleichterten Erstellen von Quellcodes

Um das Erstellen des Quellcodes zu erleichtern wurde „Microsoft Visual Studio 2005 Entwicklungsumgebung“ als Programmierumgebung ausgewählt.

Entwicklung der Struktur

Bei größeren Projekten ist es zunächst sehr wichtig eine Struktur zu entwickeln.
Durch die Größe des Programms war es nicht möglich mit einer Klasse auszukommen. Deshalb ist es sinnvoll mehrere Klassen zu verwenden um den zusammengehörenden Code logisch zu gruppieren.

Namen

Weiters wurden auch Benennungsregeln eingeführt, um besser zu erkennen um welches Element es sich handelt:

- **Variablen** und **Objekte** werden immer nur mit dem Namen benannt
- **Klassen** => K_NAME
- **Struct** => S_NAME
- **Enum** => E_NAME
- **Delegate** => D_NAME

Struktur

So wurden Folgende Strukturen entwickelt:
Durch die benötigten Formulare wurden folgende Klassen erstellt:

- **K_MainForm**
  Repräsentiert das Hauptfenster
- **K_Eigenschaften**
  Ein dynamisches Fenster für Einstellungen
- **K_GruppenForm**
  Repräsentiert ein Gruppenfenster
- **K_AnzeigeFenster**
  Repräsentiert ein Anzeigefenster
- **K_Protokoll**
  Repräsentiert das Statusfenster des Protokollsystems und enthält das gesamte Protokollsystem

Durch die Aufteilung der Aufgaben kamen noch folgende Klassen dazu:

- **K_Hauptprogramm**
  Enthält den Code zum Starten und Stoppen des Programms
- **K_Einstellungen**
  Code zum Verwalten der Einstellungsdatei
- **K_Variablen**
  Deklaration von wichtigen Klassenübergreifenden Variablen
- **K_Sprachen**
  Code zum Laden und Verwenden der Sprachdatei
- **K_Anzeigen**
  Code zum Verwalten der Gruppen - und Anzeigefenster
- **K_Paint**
  Code zum Erzeugen von dynamischen Zeichnungen
- **K_Beep**
  Code zur Ausgabe von periodischen Tönen
- **K_Funkverwaltung**
  Diese Klasse besteht aus zwei Dateien:

- Funkverwaltung.cs
  - Code zum Synchronisieren und Weitergeben der empfangenen Daten
- Funkverwaltung.Funkverkehr.cs
  - Code für die realtime Kommunikation

Diese Klasse ist eigentlich auch ein Formular welches die Statusleiste repräsentiert.

- **K_Port_List**
  Code zum Suchen von möglichen Mastern

Multithreading

Es ist auch nötig sich zu entscheiden ob Multithreding verwendet werden soll oder nicht. Multitherading bedeutet, dass mehrere Threads gleichzeitig laufen. Dies hat mehrere Vor- und Nachteile:

- Vorteile:
  - Verarbeitung von mehreren Aufgaben nahezu gleichzeitig
  - Dynamisches aufteilen der Rechenleistung
  - Vermeiden von unnötigen Wartezeiten
- Nachteile:
  - Threadübergreifende Variablen- Zugriffe müssen gesichert werden
  - Zugriffe auf Formulare, die nicht mit demselben Thread erstellt wurden, sind verboten!
  - Synchronisationen sind notwendig

Da es in diesem Projekt notwendig ist Messdaten abzufragen und dem Benutzer gleichzeitig die Möglichkeit zu bieten Optionen einzustellen ohne den Funkverkehr zu unterbrechen, ist es unumgänglich Multithreading zu verwenden.

Bei folgenden Punkten konnten Optimierungen durch das Verwenden von Multithreading erzielt werden:

- **Abfrage und Verarbeitung der Messwerte in zwei Threads aufteilen**
  Durch die Aufteilung der zwei Aufgaben wird die Abfragerate erhöht, da unnötige Rechenzeit im Abfragethread wegfiel.
  Die Verarbeitung wird dann vom zweiten Thread durchgeführt während der Abfragethread auf neue Messwerte wartet.
  Dies sieht wie folgt aus:

Verarbeitungsthread	Gesperrt!	Werte verarbeiten	Gesperrt!
Abfragethread	Befehl senden	Auf Antwort warten	Daten weitergeben

  Daten weitergeben…Daten an den Verarbeitungsthread weitergeben

- **Thread in der Klasse K_Beep**
  Da es vorkommen kann, dass mehrere Alarme gleichzeitig auftreten können, ist mit diesem Thread dennoch möglich ein periodisches Alarmsignal zu erzeugen
- **Thread in der Klasse K_Paint**
  Da nicht bei jedem neuen Messwert eine neue Zeichnung erstellt werden soll, wird hier mit Hilfe eines Threads die Anzeige der Messwerte zirka im Sekundetakt aktualisiert.

Es wurde im gesamten Programm darauf geachtet, dass Synchronisierungen zwischen den einzelnen Threads stattfinden und dass keine Zugriffsrechte verletzt werden.

Verweise zwischen den einzelnen Klassen

Das nächste Problem ergibt sich wenn eine Klasse auf mehrere Variablen und Funktionen von anderen Klassen zugreifen muss.
Dies ist fast überall der Fall. Beispielsweise muss die Klasse K_MainForm auf nahezu alle Klassen zugreifen, um die einzelnen Funktionen auszuführen.
Diese Problemstellung wurde wie folgt gelöst:
In jeder Klasse sind Global die benötigten Zeiger deklariert. Zum Beispiel kann das wie folgt aussehen:

```
private K_Sprachen Sprache;
private K_Variablen Variablen;
private K_Einstellungen Einstellungen;
private K_Funkverwaltung Funkverwaltung;
private K_Anzeigen Anzeigen;
private K_Protokolle Protokoll;
```

Es kann auch vorkommen, dass Zeiger beim Ausführen von Funktionen übergeben werden müssen.
Beim Erstellen einer Klasse werden dann vom Hauptprogramm die benötigten Zeiger auf die Objekte der Klassen übergeben.
Wenn ein benötigtes Objekt beim Erstellen einer Klasse noch nicht zur Verfügung steht, gibt es auch eigene Funktionen zum Nachladen des Zeigers.

5.2.1.2. Lösungswege

Seite

K_Eigenschaften

Diese Klasse wurde zuerst entwickelt. Da es abzusehen war, dass der Benutzer die Möglichkeit haben muss viele verschiedene Einstellungen vorzunehmen. Deshalb wurde ein dynamisches Einstellungsfenster entwickelt.

Problemstellungen

Folgende Problemstellungen haben zur Entwicklung der Klasse geführt:

- Bei der Statusanzeige von den Slaves sollen nur aktive Slaves angezeigt werden. Diese Funktion setzt ein dynamisch erstelltes Statusfenster voraus.
- Zum Erstellen von Kalibrationskurven können pro Messwert bis zu drei Kurven notwendig sein. Darum ist es notwendig ein dynamisches Fenster zu erstellen.
- Beim Erstellen von mehreren Anzeigefenstern mit der Funktion „Slavesgruppierung“ ist auch ein dynamisches Einstellungsfenster für weitere Optionen erforderlich.

Vorteile

Folgende Vorteile ergeben sich durch die Entwicklung:

- Es muss nicht für jedes Einstellungsformular eine neue Klasse erstellt werden. Jedes Eigenschaftsfenster wird mit Hilfe dieser Klasse erstellt.
- Das Eigenschaftsfenster kann dynamisch verändert werden.
- Es wird sehr viel Entwicklungszeit durch das Einsetzten des dynamischen Fensters gespart.

Nachteile

Folgende Nachteile ergeben sich:

- Um einen Fehler auszubessern der mit einem Eigenschaftsfenster zusammenhängt sind gute Kenntnisse über diese Klasse erforderlich.
- Man findet sich im Programmcode ohne ausreichende Kenntnisse über diese Klasse nicht einfach zurecht.

Verwendet in

Die Klasse wird an vielen verschiedenen Stellen im Programm verwendet:

- K_AnzeigeFenster
  - Einstellungen_vornehmen()
- K_Anzeigen
  - Hinufuegen()
  - Hinzufuegen_Gruppe()
  - Hinzufuegen_Mehrere_Fenster()
  - Entfernen()
  - Speichern()
  - Vorlagen_Verwalten()
- K_Eigenschaften (Selbstaufruf!)
  - Beschreibung_Aufrufen()
- K_Funkverwaltung
  - Funkverkehr_Einstellungen()
  - Sensoren_Status_Anzeigen()
  - Befehl_Aussenstelle_Hinzufuegen_Neu()
  - Befehl_Aussenstelle_Hinzufuegen_Vorhanden()
  - Befehl_Aussenstelle_Entfernen()

        - Befehl_Aussenstelle_Abschalten()
        - Befehl_Aussenstelle_neu_Suchen()
        - Aussenstellen_Kanal_wechseln()
- K_GruppenForm
    - mIEinstellungen_Click()
- K_Hauptprogramm
    - Main()
- K_MainForm
    - mIGrundeinstellungen()
- K_Protokolle
    - Start()
    - Daten_exportieren()
- K_Variablen
    - Kalibrationen_verwalten()
    - Kalibration_bearbeiten()
    - Manuelle_Bearbeitung_Werte_Kalibration()

K_Einstellungen

Eine weitere Problemstellung ergab sich bei der Speicherung der Einstellungen. Hier wollte man editierbare Einstellungsdateien erzeugen. In C#.NET wird aber nur die Möglichkeit geboten Binärdateien zu erstellen.
Darum wurde eine Klasse entwickelt, die es ermöglicht editierbare Einstellungsdateien zu erstellen.

Aufgabenstellung

Für folgende Aufgaben musste die Klasse entwickelt werden:

- Erstellen von editierbaren Einstellungsdateien
- Exportieren und importieren von Einstellungen
- Einstellung auf Standardwerte zurücksetzen
- Dynamische Anzahl von Einstellungswerten
- Kommazahlen ländereinstellungsunabhängig speichern

Realisierung

Folgende Eigenschaften der Einstellungsdatei wurden gewählt um die Aufgabenstellungen zu erfüllen:

- Die Einstellungen werden in der Datei „**Einstellungen.dat**“ abgespeichert.
- Die Datei kann mit einem Texteditor wie „Notepad.exe“ editiert werden. Alle Eigenschaften werden in Klartext angezeigt.
- Das Kommerzeichen wird immer als „.“ (Punkt) ausgeführt, um unterschiedliche Ländereinstellungen zu umgehen.
- Am Anfang der Datei muss folgendes stehen:
  „[Einstellungen]“
- Eine Einstellung einspricht einer Zeile in der Datei.
  Die Zeilen sind wie folgt aufgebaut:
  Einstellungsname=Einstellungswert
- In der Variable „`string[,] Standards`“ können Standardwerte eingetragen werden. Wenn diese Werte in der Einstellungsdatei nicht vorkommen, werden die Werte aus der Variable Standards verwendet.

Vorteile

- Erleichtert das Verwalten von Einstellungen.
- Standardwerte können sehr leicht implementiert werden.
- Der Benutzer kann Einstellungen vornehmen ohne das Programm starten zu müssen.

Nachteile

- Durch unsachgemäße Manipulation der Werte können Systemfehler auftreten, da die Werte der Einstellungsdatei nicht auf die richtige Eingabe kontrolliert werden.

K_Variablen

Die Klasse K_Variablen wurde als Platzhalter für klassenübergreifende Variablen erstellt. Hier werden wichtige Informationen temporär zwischengespeichert.

Folgende Daten und Funktionen sind in dieser Klasse untergebracht:

- Allgemeine Informationen
- Dynamische Beschreibung von Messtypen
- Kalibrationen
- Eigenschaften der Slaves
- ID- Verwaltung

Nun folgen einige der wichtigsten Problemlösungen die in diese Klasse eingebaut wurden:

Dynamische Beschreibung von Messtypen

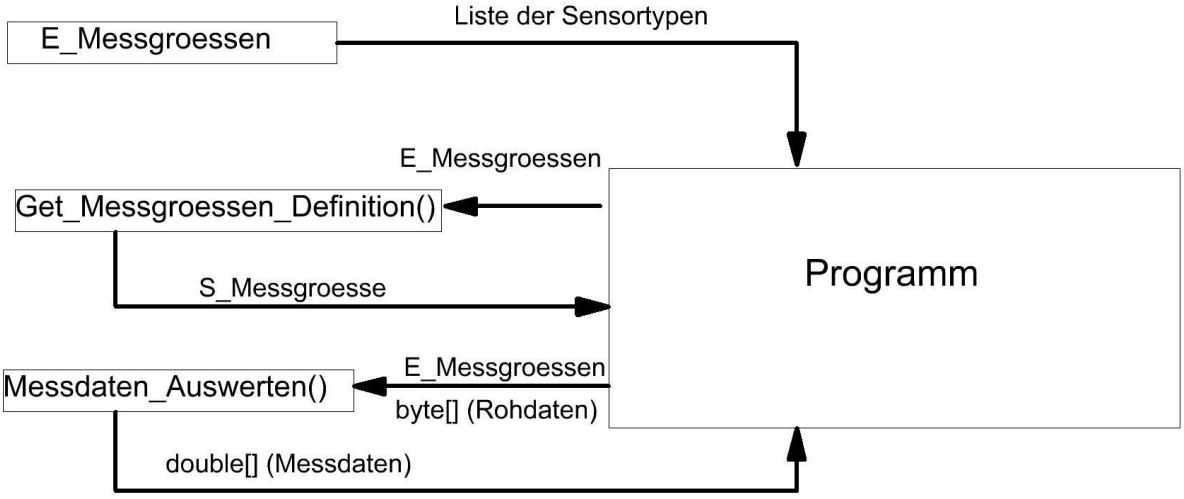
Hier wurde ein System entwickelt zum dynamischen Verwalten von Messtypen.

Problemstellung

Am Anfang der Programmentwicklung waren die Messtypen noch nicht festgelegt. Deshalb war es notwendig ein System zu entwickeln, dass im Nachhinein neue Messtypen hinzufügen oder alte Messtypen verändern kann.

Realisierung

Es wurden eine Liste mit den Messtypen und zwei Funktionen zur Verwaltung eingeführt:



- `enum E_Messgroessen`
  - Liste der Sensoren, die auf den Slaves bestückt sein können
  - Die Reihung entspricht auch gleich dem Wert des Messdatenabfragebefehls. Genauere Informationen darüber finden Sie im Kapitel „K_Funkverwaltung“
- Get_Messgroessen_Definition()
  - Hier werden mit Hilfe der Eigenschaft “`E_Messgroessen`„ die Eigenschaften (`S_Messgroesse`) des betroffenen Sensortyps zurückgegeben
- Messdaten_Auswerten()
  - Wandelt die empfangenen Rohdaten nach einem Algorithmus in Messdaten um
  - Es wird ein „`double[]`“ mit den Messwerten zurückgegeben

Vorteile

- Es ist nachträglich sehr leicht möglich neue Messtypen einzuführen und zu verwenden
- Die Änderungen der Parameter eines Messtyps müssen nur an einer Stelle vorgenommen werden

Nachteil

- Der Quellcode ist durch diese Maßnahme komplexer und unübersichtlicher geworden.

Kalibrationen

Es soll dem Benutzer die Möglichkeit geboten werden gemessene Werte zu kalibrieren:

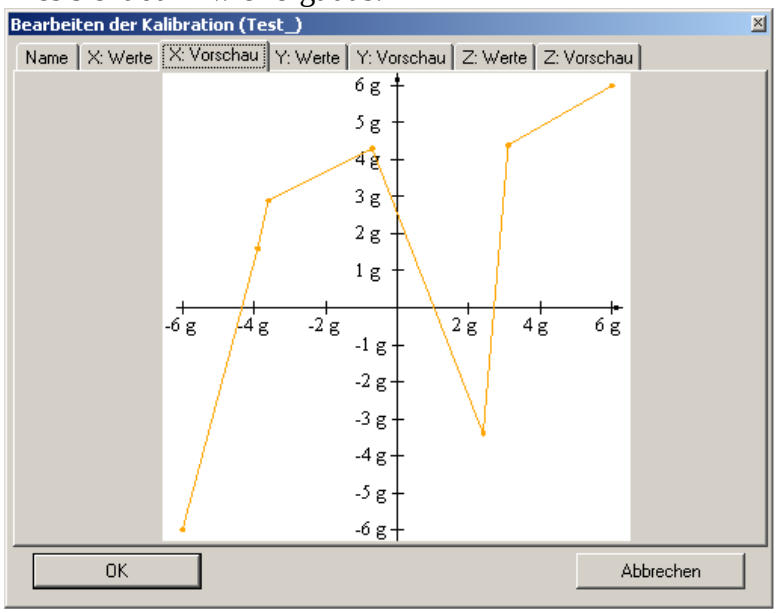
Aufgabenstellung

Folgende Aufgaben mussten erledigt werden:

- Verwaltung von allen Kalibrationen
  - Optische Darstellung der Kurven
  - Manuelle Beareitung
- Speichern der Messwerte mit Hilfe der Klasse K_Einstellungen
- Export und Import von Einstellungsdateien

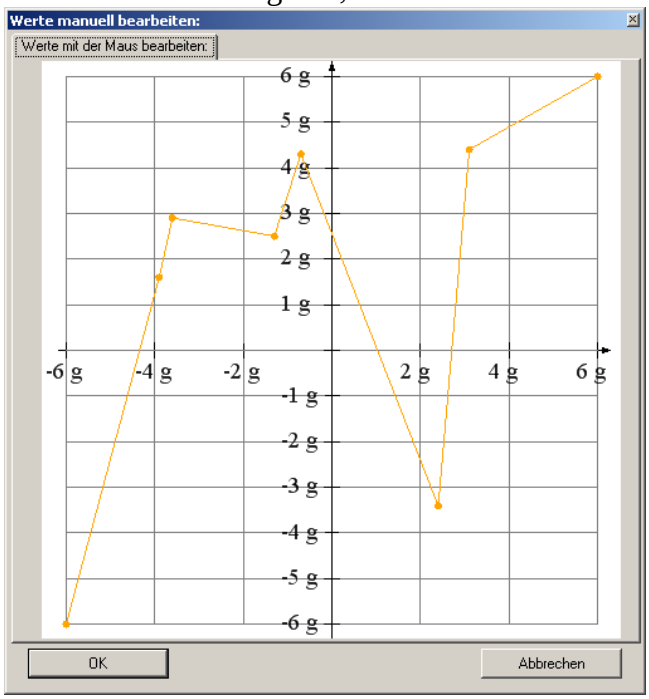
Realisierung

Zum Verwalten der Kalibrationen wurde die Klasse K_Eigenschaften verwendet und mit speziellen Funktionen erweitert. Zum optischen Darstellen wurde auch die Klasse K_Paint angepasst.
Dies sieht dann wie folgt aus:



Hier wurde in der Klasse K_Eigenschaften eine Funktion eingebaut, welche dynamisch einen Verweis auf ein Objekt der Klasse K_Paint erzeugt.

Es wurde auch ermöglicht, dass der Benutzer manuell die Kurve bearbeiten kann:



Hier wurde in die Klasse K_Eigenschaften eine Eventverwaltung eingebaut, um einen Mausklick des Benutzers zu erfassen und zu verarbeiten.

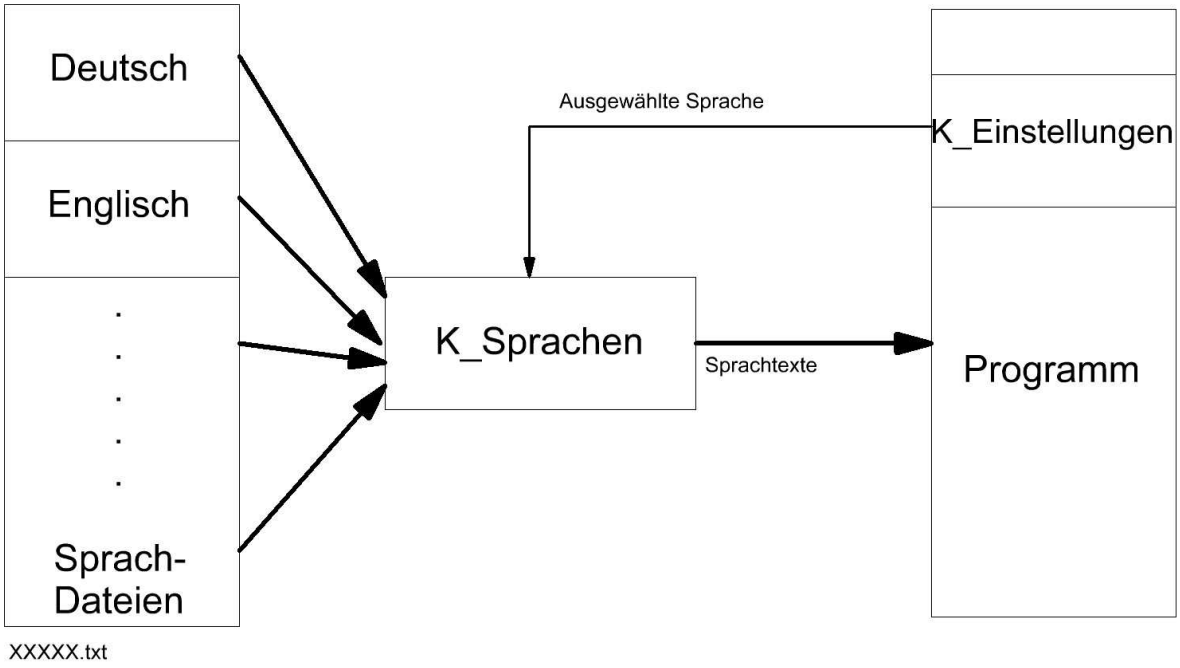
K_Sprachen

Problemstellung

Das Programm soll später auch in andere Sprachen übersetzt werden können. Der Aufwand dafür soll sehr gering sein.

Realisierung

Um diese Problemstellung zu lösen wurde die Klasse K_Sprachen entwickelt.
Die Klasse greift auf eine Sprachdatei (Textdatei) zu und liest die Informationen beim Starten des Programms aus.



Mit der Funktion „Get()“ kann dann auf die Texte zugegriffen werden.
Um eine neue Sprache zu realisieren muss nur eine neue Sprachdatei erstellt werden. Dazu muss nur eine vorhandene Sprachdatei übersetzt werden.

K_Paint

Im gesamten Programm ist es an vielen Stellen notwendig sich sehr oft ändernde Informationen anzuzeigen. Dazu wurde die Klasse K-Paint entworfen.

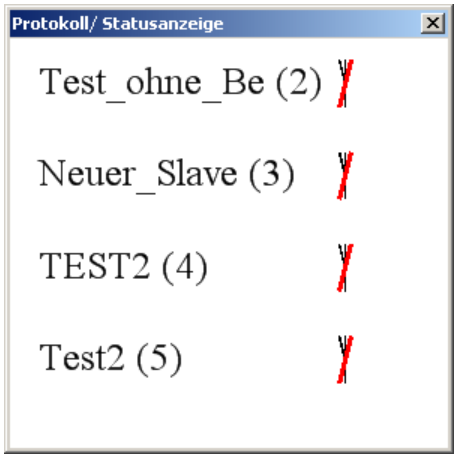
Aufgabenstellung

- Anzeigen von dynamischen, sich sehr oft ändernden Inhalten
- Verschiedene Anzeigearten
  - Numerische Darstellung
  - Balkendarstellung
  - XY- Darstellung
  - Status Anzeige
- Automatische Aktualisierung der Anzeige

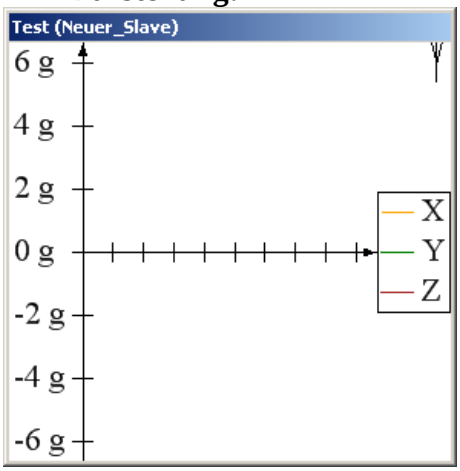
Realisierung

Da sehr viel Rechenleistung verwendet wird, um neue Elemente eines Formulars zu erstellen, wurde bei der Problemlösung auf die Zeichenfunktion der Formklasse zurückgegriffen.
Somit wurde es ermöglicht diverse Inhalte sehr schnell optisch darzustellen ohne viel Rechenleistung zu verbrauchen.
Weiters wird in der Klasse ein eigener Thread erstellt, um die Zeichnungen aktuell zu halten.
Die einzelnen Darstellungsarten sehen wie folgt aus:

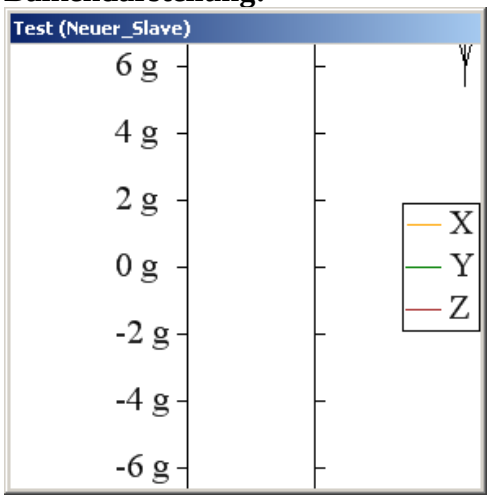
Status Anzeige:



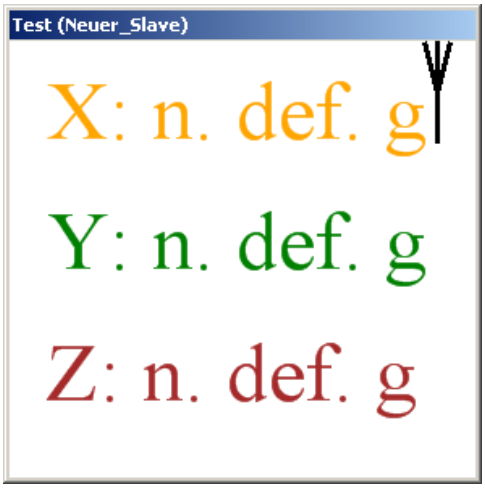
XY- Darstellung:



Balkendarstellung:



Numerische Darstellung:



Vorteile

- Schnelles Anzeigen von neuen Messwerten
- Dynamische Gestaltung der Anzeige
- Sparen von Rechenleistung
- Eine Zeichnung kann von jedem Thread aus erstellt werden, auch wenn der Thread nicht der Eigentümer des betroffenen Formulars ist

Nachteile

- Gezeichnete Zeichnungen werden nicht permanent angezeigt, darum müssen sie regelmäßig aktualisiert werden
- Die Berechnung einer Zeichnung ist sehr aufwändig

Verwendet in

Die Klasse wird von Folgenden Funktionen und Klassen verwendet:

- K_AnzeigeFenster
  - Zum Visualisieren der Messwerte
- K_Funkverwaltung
  - Befehl_Funkbelastungsmessung()
- K_Protokoll
  - Zum Visualisieren der Statusmeldung
- K_Variablen
  - Kalibration_bearbeiten()
  - Manuelle_Bearbeitung_Werte_Kalibration()

K_Beep

Problemstellung

Folgende Problemstellung hat zur Entwicklung der Klasse K_Beep geführt:
Durch die Verwendung von mehreren Threads war es nicht möglich bei mehreren Alarmen ein periodisches Alarmsignal zu erzeugen.

Realisierung

Zum Synchronisieren der Alarmsignale wurde diese Klasse entworfen.
Diese Klasse wird von jedem System, welches einen Alarm auslösen kann, initialisiert. Durch die Verwendung von statischen Variablen werden die Informationen koordiniert.
Beim ersten Erstellen der Klasse wird auch ein neuer Thread erstellt, der dann bei einem oder mehreren Alarmen periodische Alarmsignale ausgeben kann.

Weiters begründet diese Funktion auch die Verwendung der „.NET Framework 2.0“. In der „.NET Framework 1.1“ ist die Ausgabe von akustischen Signalen nicht möglich. Dies wird erst ab der Version 2.0 unterstützt.

K_Funkverwaltung

In dieser Klasse sind alle Algorithmen vorhanden, die für die Funkverwaltung benötigt werden.

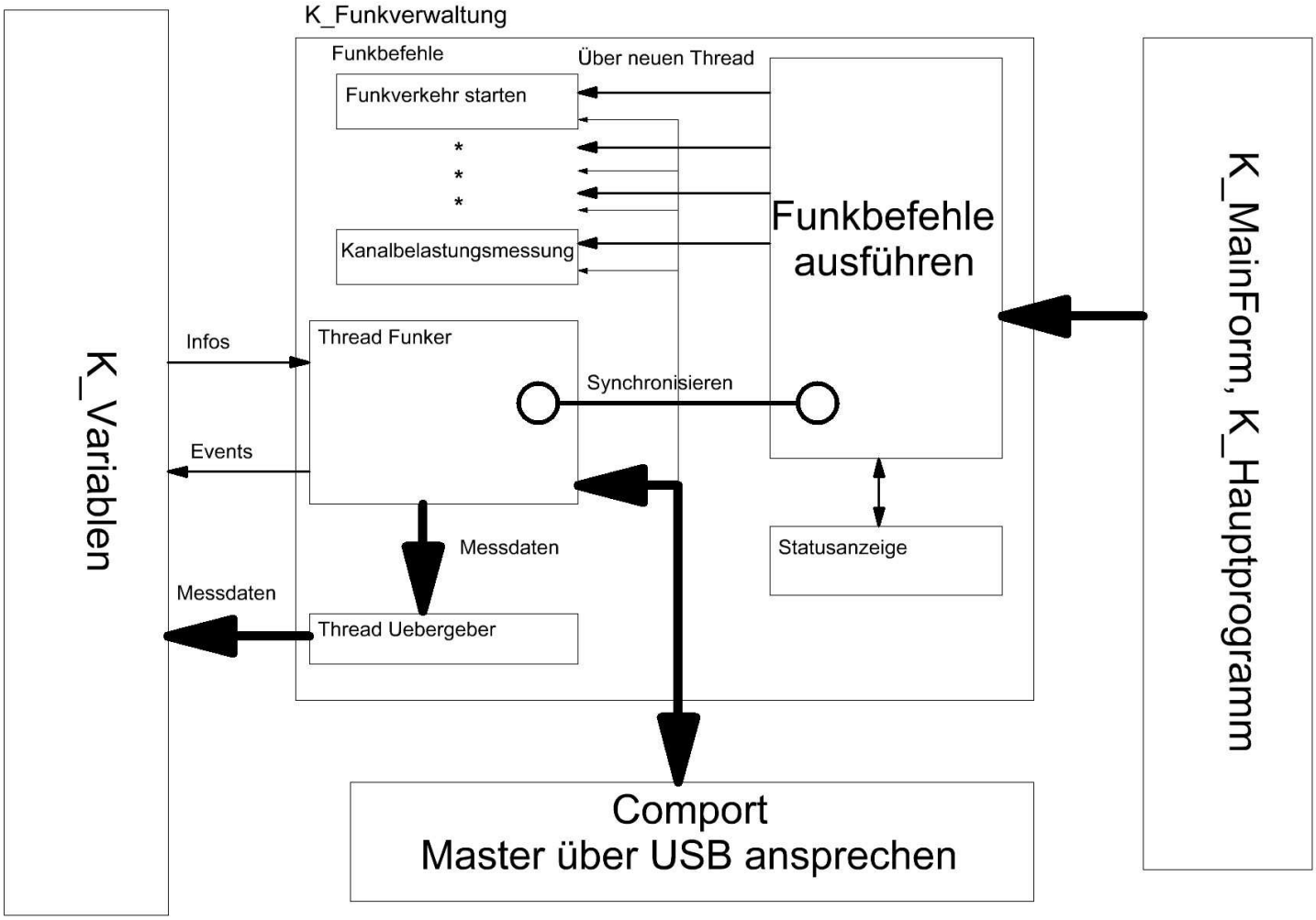
Problemstellung

Folgende Problemstellungen haben sich bei der Entwicklung der Software ergeben:

- Aufteilen der Rechenleistung zum Abfragen und Weitergeben der Messwerte auf zwei Threads
- Kommunikation zum Master über USB herstellen
- Diverse Funkbefehle verarbeiten
  - Kanalbelastungsmessung
  - Funkverkehr Starten und Stoppen
  - Suchen der Slaves
  - Kanalwechsel

Problemlösung

In der folgenden Grafik wird die Realisierung der Klasse ersichtlich:



Die Rechenleistung wurde auf den Thread Funker und den Thread Uebergeber aufgeteilt. Weiters wird das System mit dem Funkerthread synchronisiert um Funkbefehle durchzuführen.

Realisierung Slaves neu suchen

In diesem Kapitel geht es darum den Algorithmus der Funktionen `Aussenstelle_neu_Suchen()` und `Netzwerk_berechnen()` genauer zu beschreiben.
Diese Funktionen haben zur Aufgabe alle aktiven Slaves zu suchen und alle geeigneten Funkstrecken zu bestimmen.

Genaue Aufgaben:

- Kanalwechsel aller Slaves die nicht denselben Kanal haben
- Alle Slaves in der ersten Ebene suchen
- Alle Slave in weiteren Ebenen suchen
- Zusätzliche Funkstrecken berechnen
- Wenn gewollt, eine Meldung ausgeben

Kanalwechsel aller Slaves die nicht denselben Kanal haben:
Wenn der Benutzer vorhandene Slaves zum Messen verwenden will, kann es vorkommen, dass manche Slaves einen anderen Kanal, als den eingestellten Kanal, haben.
Um Messen zu können müssen alle Slaves denselben Kanal aufweisen.
Um den Kanal zu wechseln, muss der Benutzer den Slave in die erste Ebene bringen.
Vom Programm wird immer der Kanal vorgeschlagen, welchen die meisten Slaves eingestellt haben. Durch die Maßnahme müssen weniger Slaves in die erste Ebene gebracht werden.
Beim Kanalwechsel wird auch die ID überprüft, um sicherzustellen, dass es sich um den richtigen Slave handelt.
Wurde ein alternativer Kanal gewählt, wird der Benutzer gefragt, ob wieder auf den zuvor eingestellten Kanal zurückgewechselt werden soll. Um den Kanal zu wechseln, wird die Funktion nochmals aufgerufen, um temporäre Funkstrecken zu berechnen
Danach wird versucht für alle Slaves den Kanal zu wechseln. Genaueres zum Kanalwechsel finden Sie in Kapitel „Funktionsbeschreibung Kanalwechsel“.

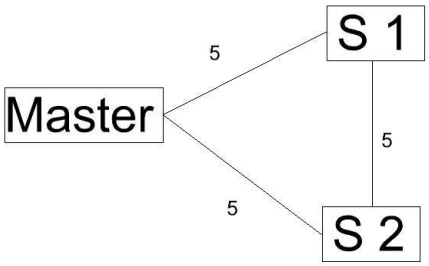
Alle Slaves in der ersten Ebene suchen (direkte Kommunikation):
Beim Suchen der Slaves werden sofort Funkstrecken mitgeschrieben. Somit kann von Anfang an eine Aussage über die Zuverlässigkeit der Funkstrecken getroffen werden.
Eine Streche wird immer fünfmal getestet. Wenn keiner dieser fünf Versuche erfolgreich war, wird angenommen, dass diese Funkverbindung nicht vorhanden ist.

Nun wird versucht alle aktiven Slaves in der ersten Ebene zu finden. Die Funkstrecken der gefundenen Slaves werden mit ihren Bewertungen gespeichert.

Alle Slaves in weiteren Ebenen suchen (indirekte Kommunikation):
Es wird nun versucht alle Slaves zu finden und deren Umgebung zu scannen.
Dabei wird aber darauf geachtet, dass nur die besten Funkstrecken weiterverwendet werden, da unsichere Funkstrecke zu noch unsichereren Suchergebnissen führen.

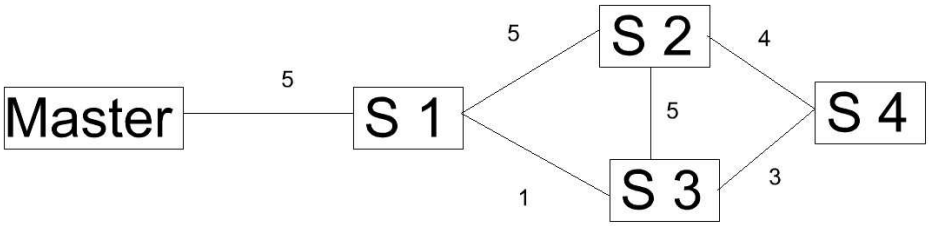
Jede Umgebung wird nur einmal gescannt und der Vorgang wird beendet, sobald jeder Slave gescannt worden ist. Beim Sannen sind auch die Slaves ausgeschlossen von denen schon ein Scann durchgeführt wurde. Dies wird gemacht, um so wenige Funkbefehle wie möglich zu benutzen.

Nun folgen Beispiele für Funkumgebungen und die Reaktion des Algorithmus:
Die Zahlen geben an, wie viele Versuche, zum Finden der Slaves, erfolgreich waren.



In der ersten Ebene werden die Slaves S1 und S2 gefunden.
Beim Scannen der Umgebung werden auch jeweils S1 und S2 gefunden.
Daraus ergeben sich folgende Funkstrecken:
Für S1:
M
M ⇔ S2
Für S2:
M
M ⇔ S1
Der Algorithmus hat ohne Probleme funktioniert.

Ein weiters Beispiel:



In der ersten Ebene wird nur der Slave S1 gefunden.
Beim Scannen in weiteren Ebenen wird also zuerst das Umfeld von S1 gescannt.
Dadurch wird S2 und S3 gefunden.
Nun wird das Umfeld von S2 gescannt, da die Verbindung am Sichersten ist.
Dadurch wird S4 gefunden und eine sicherere Verbindung zu S3.
Nun wird das Umfeld von S3 gescannt.
Dies passiert über die Funkstrecke M ⇔ S1 ⇔ S2, da die Verbindung über M ⇔ S1 zu S3 zu unsicher ist.
Es wird S4 gefunden.
Da bereits von S1, S2 und S3 das Umfeld gescannt wurde, ist es nicht notwendig das Umfeld von S4 zu bestimmen, da es errechnet werden kann.

Daraus ergeben sich folgende Funkstrecken:
Für S1:
M
Für S2:
M ⇔ S1
Für S3:
M ⇔ S1
M ⇔ S1 ⇔ S2
Für S4:
M ⇔ S1 ⇔ S2

M ⇔ S1 ⇔ S2 ⇔ S3

Auf den ersten Blick sieht das Ergebnis ganz brauchbar aus. Aber folgende mögliche Funkstrecken sind noch nicht vorhanden:
Für S2:
M ⇔ S1 ⇔ S3
M ⇔ S1 ⇔ S3 ⇔ S4
Für S3:
M ⇔ S1 ⇔ S2 ⇔ S4
Für S4:
M ⇔ S1 ⇔ S3 ⇔ S2
M ⇔ S1 ⇔ S3
Um dieses Problem zu lösen könnte man auch schon gescannte Slaves nochmals scannen.
Bei dieser einfachen Struktur würde das zum Erfolg führen, aber bei Umgebungen mit bis zu 60 Slaves würde dies zu einer nahezu unerträglichen Wartezeit führen.

Eine weitere Möglichkeit ist es weitere Funkstrecken zu berechnen und dann auszuwerten.

Zusätzliche Funkstrecken berechnen:
Hierzu wird ein Algorithmus zur Netzwerkanalyse verwendet. Mit Hilfe dieses Algorithmus ist es möglich weitere Funkstrecken für die Messumgebung zu berechnen. Beim vorigen Algorithmus wird mitgeschrieben welche Slaves sich in der Umgebung von einem Slave befinden. Mit Hilfe dieser Daten werden die Berechnungen vorgenommen.

Um von einem Slave alle möglichen Wege zum Master zu finden, wird eine Funktion für jeden Zweig von dem Slave weg aufgerufen.
Diese Funktionen rufen wieder für jeden weiteren Zweig sich selbst auf. Diese Mehrfachaufrufe führen dazu, dass alle Wege bis zum Master in weitaus komplexeren Umgebungen gefunden werden.
Das System ist so aufgebaut, dass bei fertiger Berechnung eine Liste von Funkstrecken zurückgegeben wird, wobei die Funkstrecken nach der Länge sortiert sind.
Um diesen Algorithmus zu verstehen sind fortgeschrittene Programmierkenntnisse und ein exzellentes logisches Verständnis erforderlich.

Nachdem die weiteren Funkstrecken berechnet wurden, werden sie bewertet indem wieder fünfmal versucht wird den Slave über den Weg zu finden. Aus Zeitgründen werden insgesamt nur 5 Funkstrecken bewertet. Wenn bereits beim Umgebungsscann 5 Funkstrecken gefunden wurden, wird die Netzwerkanalyse gar nicht durchgeführt.

Realisierung des Kanalwechsels

In diesem Kapitel geht es darum den Algorithmus der Funktion `Aussenstellen_Kanal_wechseln()` genauer zu beschreiben.

Die Funktion hat zur Aufgabe das gesamte Messsystem auf einen anderen Kanal umzuschalten. Somit muss von jedem Slave der Kanal gewechselt werden.

Hierbei handelt es sich um einen sehr einfachen Algorithmus. Der Kanalwechsel der gesamten Messumgebung ist nur so einfach möglich, weil jeder Slave 10 Sekunden wartet bevor sie den Kanal wechselt. Das gibt dem Messsystem die Möglichkeit rund 100 Befehle zu senden bevor der erste Slave seinen Kanal wechselt.

Bei einer guten Funkbedingung würden 60 Befehle reichen um 60 Slaves zu wechseln. Die weiteren 40 Befehle sind ein Sicherheitspuffer wenn die Umgebung stark belastet ist.

Um eine sichere Wechseln zu ermöglichen werden zuerst nochmals alle Funkstrecken neu bestimmt.
Hierbei wird eine Meldung ausgegeben, wenn ein Slave nicht gefunden wird. Der Benutzer wird zum Wiederholen des Suchvorganges aufgefordert. Wenn der Benutzer dies aber nicht will, werden die nicht gefundenen Slaves deaktiviert.

Beim Wechsel wird jedem Slave der Befehl zum Wechseln geschickt. Dieser Durchlauf wird auch fortgesetzt, wenn ein Slave nicht antwortet. Es werden alle zur Verfügung stehenden Funkstrecken durchprobiert.
Dieser Schleife wird beendet wenn sich nichts mehr ändert oder alle Slaves erfolgreich gewechselt wurden.
Nach dieser Schleife wird noch 10 Sekunden gewartet um zu garantieren, dass alle Slaves den Kanal gewechselt haben.

Funktionsbeschreibung der Messdatenerfassung

Die Messdatenerfassung ist ein sehr komplexer Vorgang in dem zahlreiche Funktionen und Variablen verwickelt sind.

Um die Abfrage der Messdaten so schnell wie nur möglich zu gestalten, wurde die Aufgabe auf zwei Threads aufgeteilt.
Der erste Thread ist der so genannte Funkerthread. Er ist für die Messdatenabfrage verantwortlich.
Der zweite Thread ist der so genannte Übergeberthread. Er ist dafür verantwortlich, dass die Messwerte an die Anzeigefenster und an das Protokollsystem übergeben werden.

Messdaten zum Abfragen bestimmen:
Zum Sammeln der Messdaten ist die Funktion „`FunkerThread()`“ zuständig. Diese Funktion wird vom Funkerthread aufgerufen und läuft asynchron im Hintergrund.
Durch die Variablen „`Funkerthreadsperren`“ und „`Funkerthreadgesperrt`“ ist es möglich mit dem Thread zu synchronisieren. Dies wird von der Funktion „`Fuehre_Befehl_aus()`“ ausgenutzt um Funkbefehle auszuführen.
Wenn der Funkverkehr gestartet ist, wird nach aktiven und verbundenen Slaves gesucht. Wenn von den Slaves Messwerte gebraucht werden, wird die Abfrage eingeleitet.
Die Funktion „`Messwert_abfragen()`“ führt dann die Messwertabfrage durch.
Wenn mehrere Messwerte abgefragt werden sollen, wird folgendes System zu Koordinierung eingesetzt:
Für jede Messgröße existiert folgende Variable: „`naechste_Messung`“ und für die Batteriemessung „`naechste_Bat_Messung`“. Diese Variablen sind vom Typ `long` und können einen Zeitstempel speichern.
Bei jeder Messung wird der Zeitstempel neu gesetzt. Wenn die Messung erfolgreich war, wird die aktuelle Zeit plus das Messintervall eingespeichert. Wenn es zu einem Funkfehler kommt wird nur die aktuelle Zeit eingetragen.
Eine Messung wird dann ausgeführt, wenn der Zeitstempel jünger als die aktuelle Zeit ist.
Somit werden auch bei vielen Funkfehlern immer noch Messungen mit sehr langen Intervallen zeitgerecht abgefragt und Abfragen mit Funkfehlern werden bevorzugt behandelt.

Sind keine Messungen durchzuführen wird eine Pause von 100ms eingelegt und dann erneut nach Messungen gesucht.

Steht die nächste Messung erst in mehr als 100ms an, wird dennoch nach 100ms nach einer neuen Messung gesucht, denn die Pause ist auf 100ms beschränkt.
Somit wird garantiert, dass bei Einstellungsänderungen zu keinen längeren Wartezeiten kommt.

Messdaten abfragen:
Hier wird die genaue Funktionsweise der Funktion „`Messwert_abfragen()`“ beschrieben. Diese Funktion wird vom Funkerthread aufgerufen um Messdaten abzufragen.

Als Erstes wird die beste Funkstrecke bestimmt. Hierbei wird folgende Bewertung benutzt:
Bei jeder Funkstrecke wird eine Statistik mitgeschrieben:
Die Variable „`Versuche`“ gibt an, wie oft die Funkstrecke benutzt wurde.
Die Variable „`fehlerhafte_Verbindungen`“ gibt an, wie viele Fehler dabei aufgetreten sind.
Die Variable „`Fehler_hintereinander`“ gibt an, wie viele Fehler hintereinander aufgetreten sind.
Wenn man die fehlerhaften Verbindungen durch die Versuche dividiert, dann erhält man eine Aussage darüber wie gut eine Funkstrecke ist.
Mit Hilfe dieses Faktors wird auch die beste Funkstrecke ausgewählt.

Nun ergibt sich noch folgendes Problem:
Wenn es zu einer Änderung der Messumgebung kommt dauert es sehr lange bis sich dies auf den Faktor auswirkt.
Somit werden auch noch die Fehler hintereinander mitgeschrieben.
Wenn 10 Fehler hintereinander ermittelt werden, kann davon ausgegangen werden, dass sich die Funkumgebung geändert hat und die Statistik der Funkstrecke wird zurückgesetzt.
Wenn es wiederum zu 10 Funkfehlern hintereinander kommt, kann davon ausgegangen werden, dass die Funkstrecke nicht mehr vorhanden ist und wird deaktiviert.
Deaktivierte Funkstrecken werden nicht mehr zum Datentransport verwendet, somit wird auf eine andere zuvor ermittelte Funkstrecke zurückgegriffen.
Wenn alle Funkstrecken inaktiv sind, wird die Variable „`Verbindung`“ zurückgesetzt und es wird ein Event ausgelöst der dem Benutzer anzeigt, dass keine Funkverbindung zum Slave mehr vorhanden ist.

Nachdem eine Funkstrecke ausgewählt wurde, wird eine Messanfrage durchgeführt.
Wenn Daten empfangen werden, werden diese mit Hilfe der Funktion „`Wert_Uebergeben()`“ an den Übergeberthread weitergegeben. Dies wird gemacht um Rechenzeit des Funkerthreads zu sparen.

Nun wird noch die Statistik der Funkstrecke und die Variable „`naechste_Messung`“ aktualisiert.

Messdaten weitergeben:
Diese Aufgabe wird vom Übergeberthread übernommen. In der Variable „`Uebergebene_Werte`“ sind alle zu übergebenden Messwerte abgespeichert. Die Variable wird vom Funkerthread gefüttert.

Der Übergeberthread hat eine niedrigere Priorität als der Funkerthread, um die Messwertabfrage nicht so zu drosseln. Wenn es jedoch zu einem Überlauf der Variable „`Uebergebene_Werte`“ kommt, dass heißt das mehr als 50 Messwerte noch zu übergeben sind, wird die Priorität des Übergeberthreads erhöht, somit wird der Funkerthread gedrosselt. Wenn nur mehr 10 Messwerte zu übergeben sind, wird die Priorität wieder zurückgesetzt.

Beim Übergeben der Messwerte wird ein Event ausgelöst, der jedem Anzeigefenster oder Protokollsystem die Messwerte übergibt. Wenn erforderlich wird auch die Kalibration berechnet.

Funktionsbeschreibung Slave neu hinzufügen

Hier wird er Algorithmus der Funktion „`Befehl_Aussenstelle_Hinzufuegen_Neu()`“ genauer beschrieben.

Um einen Slave neu hinzuzufügen muss sie in den Adressiermodus gebracht werden. Hierfür wird dem Benutzer eine Meldung angezeigt.

Danach wird versucht den Slave in der ersten Ebene auf dem Kanal CH0 zu erreichen. Wenn dies nicht möglich ist, wird nochmals die Meldung angezeigt.

Nun wird die ID des Slaves abgefragt. Dies ist notwendig um zu verhindern, dass ein Slave mit einer ID zwei Adressen zugewiesen bekommt. Wenn der Slave noch keine gültige ID hat, wird eine zufällig generierte ID gesetzt und dem Benutzer angezeigt.

Dann wird der Benutzer aufgefordert einen Namen für den Slave einzugeben. Wenn die ID schon bekannt ist, wird der Name des alten Eintrages vorgeschlagen.

Dann wird nach einer freien Adresse gesucht. Wenn keine freie Adresse vorhanden ist, wird der Benutzer aufgefordert einen Eintrag zu überschreiben.

Danach werden alle wichtigen Informationen über den Slave abgefragt wie zum Beispiel welche Sensoren bestückt sind.

Zum Schluss wird noch die Adresse neu gesetzt und der Kanal des Slaves gewechselt. Ist dies erfolgreich, wird der Benutzer noch gefragt ob die Funkstrecken neu berechnet werden sollen.

Funktionsbeschreibung Slave abschalten

Hier wird der Algorithmus der Funktion „`Befehl_Aussenstelle_Abschalten()`“ genauer beschrieben.

Zuerst wird der Benutzer gefragt, ob er die Funkstrecken neu bestimmen will um ein sicheres Abschalten zu gewährleisten.

Danach kann der Benutzer einen Slave zum Ausschalten auswählen.

Wurde einen Slave ausgewählt, wird der Vorgang zum Abschalten gestartet.
Es wird für jede Funkstrecke 10x der Befehl zum Abschalten ausgesendet. Wenn der Slave den Befehl empfängt, schaltet sie sich erst nach 2 Sekunden ab.
Wenn ein Befehl eine richtige Rückmeldung liefert, kann davon ausgegangen werden, dass der Slave sich abschaltet. Wenn keine Rückmeldung erfolgt, ist der Zustand ungewiss und der Benutzer muss den Slave manuell abschalten.

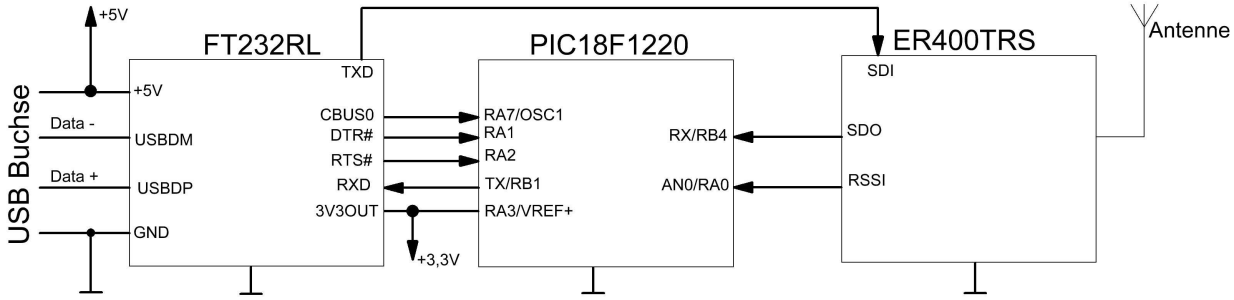
Es wird abschließend eine entsprechende Meldung ausgegeben.

5.2.2. Master

5.2.2.1. Allgemein

Die Funktion des Masters besteht grundsätzlich darin die Messdaten der Slaves zu Empfangen und an den PC weiterzuleiten. Weiters hat der Master die Aufgabe eine Kanalbelastungsmessung durchzuführen und die Daten an den PC weiterzuleiten.

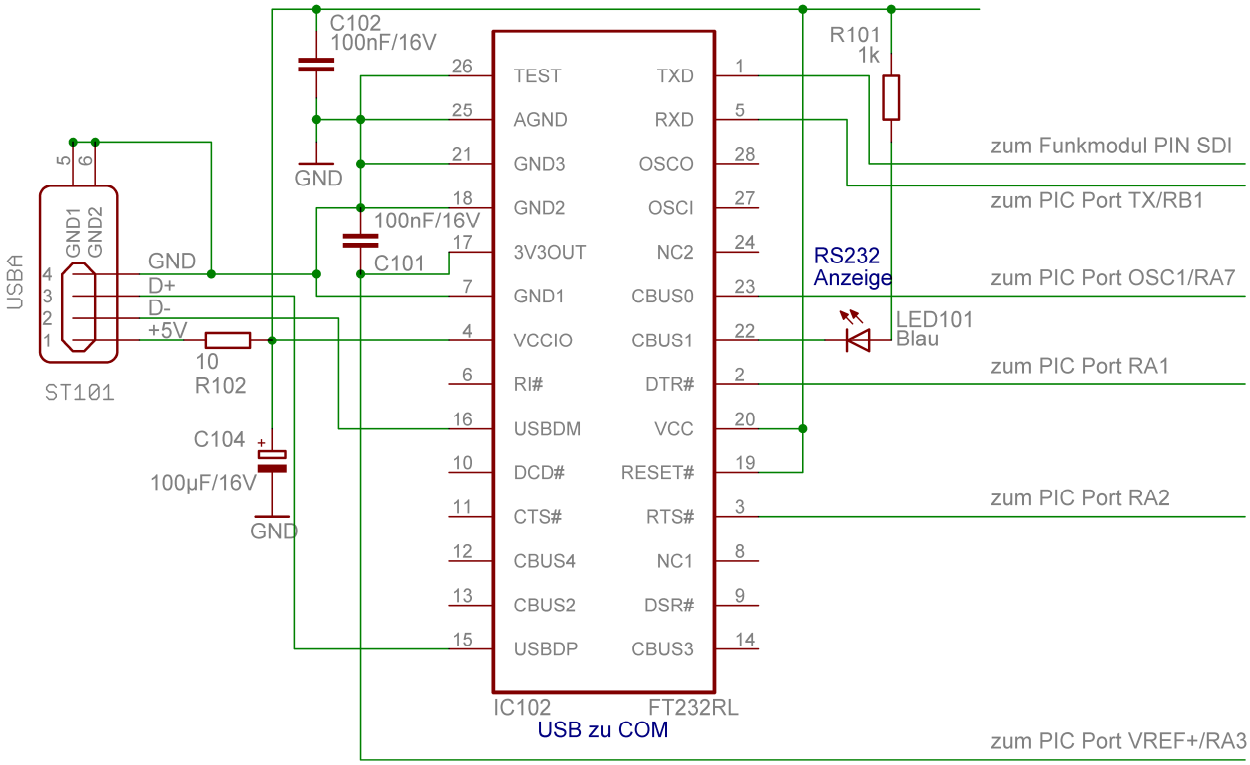
5.2.2.2. Blockschaltbild



5.2.2.3. Bauteilbeschreibungen/ Berechnungen

FT232RL

Der Chip FT232RL ist ein von der Firma FTDI Chip entwickelter USB zu USART IC. Er hat einen Clock- Ausgang der mit 6MHz, 12MHz, 24MHz oder 48MHz initialisiert werden kann. Weiters bietet der Chip einen Referenzausgang von 3,3V und 2- Handshake- Leitungen.
Es wurde dieser IC gewählt, da er preiswert ist und in der Lage ist den 24MHz- Clock für den PIC18F1220 zu liefern.



Der Chip FT232RL wandelt das USB- Signal des Computers in ein USART Signal um, welches dann direkt an das Funkmodul weitergeleitet wird.
Weiters liefert der IC die Referenzspannung für den A/D- Wandler des PIC18F1220.

Die Handshakeleitungen DTR# und RTS# dienen zur direkten Kommunikation zwischen PIC und PC.
Die LED101 dient zur Anzeige, ob eine USART Kommunikation vorliegt.
Der Ausgang CBUS0 liefert den 24MHz Clock für den PIC.

Bauteilauswahl:
R102 und C104 dienen als Eingangstiefpass und sollen hochfrequente Störungen auf der Versorgungsleitung unterdrücken.
R102 und C104 wurden experimentell ermittelt.
R102 wurde mit $10\Omega / 0{,}125W$ und C104 mit 100µF/16V gewählt.

R102 = 10Ω / 0,125W
C104 = 100µF / 16V

C102 dient zur Spannungsstabilisierung in der unmittelbaren Umgebung des FT232RL und wurde mit 100nF/16V gewählt.

C102 = 100nF / 16V

C101 wurde laut Datenblatt des FT232RL mit 100nF/16V gewählt.

C101 = 100nF / 16V

R101 dient als Vorwiderstand der LED101 und wurde wie folgt berechnet:
Vorgaben:
$U_+ = 5V \ldots Versorgungsspannung$
$U_{LED101} = 3V \ldots Flussspannung\ der\ LED101$
$I_{LED} = 2mA \ldots Diodenstrom$
Der Diodenstrom wurde hier bewusst sehr niedrig gewählt, da eine Erhöhung des Stromes die Leuchtkraft der LED kaum erhöht hat.

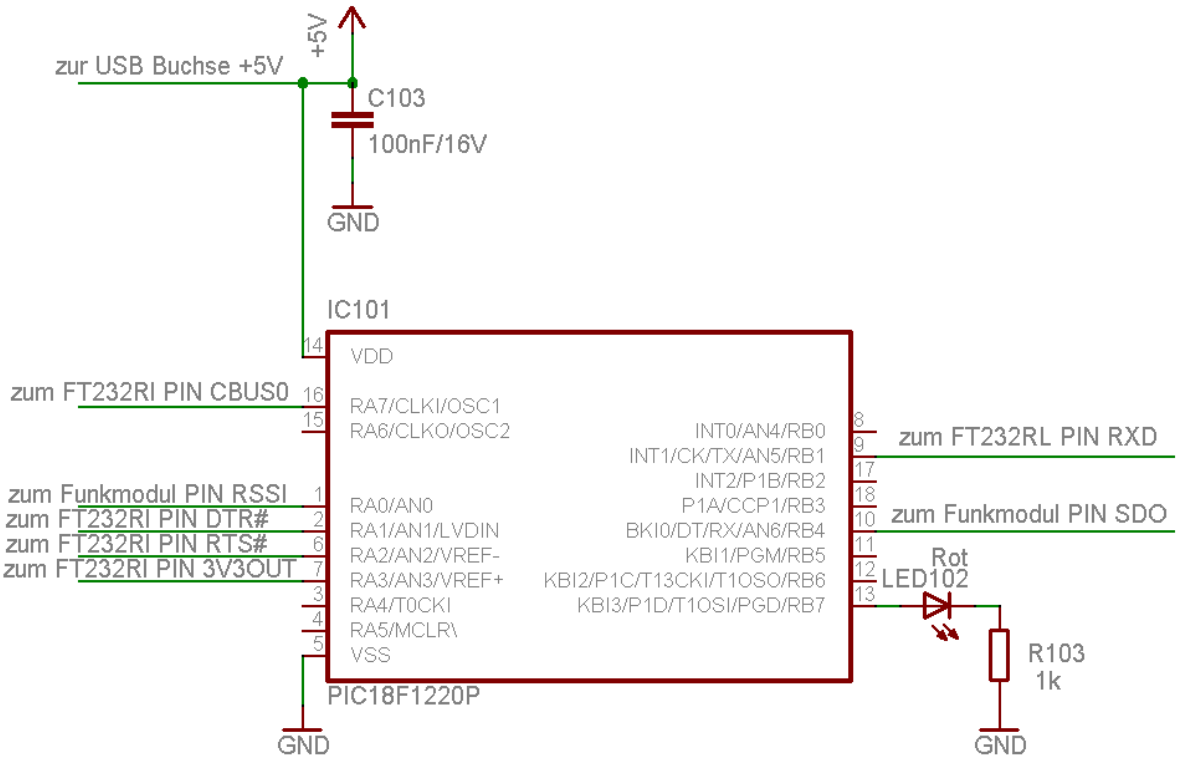
$$R101 = \frac{U_+ - U_{LED101}}{I_{LED}} = \frac{5V - 3V}{2mA} = 1k\Omega$$

gewählt:

R101 = 1kΩ / 0,125W

PIC18F1220

Der PIC18F1220 ist ein von der Firma Microchip entwickelter Mikrokontroller.
Im Gegensatz zur 18F1320 Variante hat diese Type nur den halben Programmspeicher (4kB) zur Verfügung.
Er besitzt genauso wie der PIC18F1320 eine eingebaute USART Hardware, 4 Timer, einen EEPROM und spezielle Interrupt- Prioritäten.
Anstelle des 18F1220 kann auch der 18F1320 verwendet werden. Es muss hierfür beim Programmieren des Mikrokontrollers lediglich anderes Linker File verwendet werden und die Programmierumgebung auf den PC18F1320 umgestellt werden.
Es wurde dieser Mikrokontroller gewählt, da der MCC18 - Compiler gratis zur Verfügung stand, er nicht so viele Aufgaben wie der Slave zu erledigen hat und bereits ein Vorwissen über die PIC - Familie vorhanden war.



Der PIC18F1220 dient zur Überprüfung der Empfangsdaten, ob sie dem Funkprotokoll entsprechen. Ist dies der Fall, so werden die Daten an den PC weitergeleitet. Andernfalls werden die Daten verworfen.
Außerdem hat der PIC die Aufgabe der Kanalbelastungsmessung.
Bei Bedarf hat der ADC des PIC18F1220 den analogen Kanalbelastungswert des Funkmoduls zu digitalisieren und an den PC weiter zu leiten.
Die LED102 dient zur Anzeige, ob Daten empfangen werden.

Bauteilauswahl:
C103 dient als Stabilisationskondensator und wurde mit 100nF/16V gewählt. Auf einen Kondensator im µF- Bereich in der unmittelbaren Umgebung des PIC wurde aus Platzgründen bewusst verzichtet.

C103 = 100nF / 16V

R103 dient als Vorwiderstand der LED102 und wurde wie folgt berechnet:
Vorgaben:

$U_+ = 5V \ldots Versorgungsspannung$

$U_{LED102} = 2V \ldots Flussspannung\ der\ LED102$

$I_{LED} = 3mA \ldots Diodenstrom$

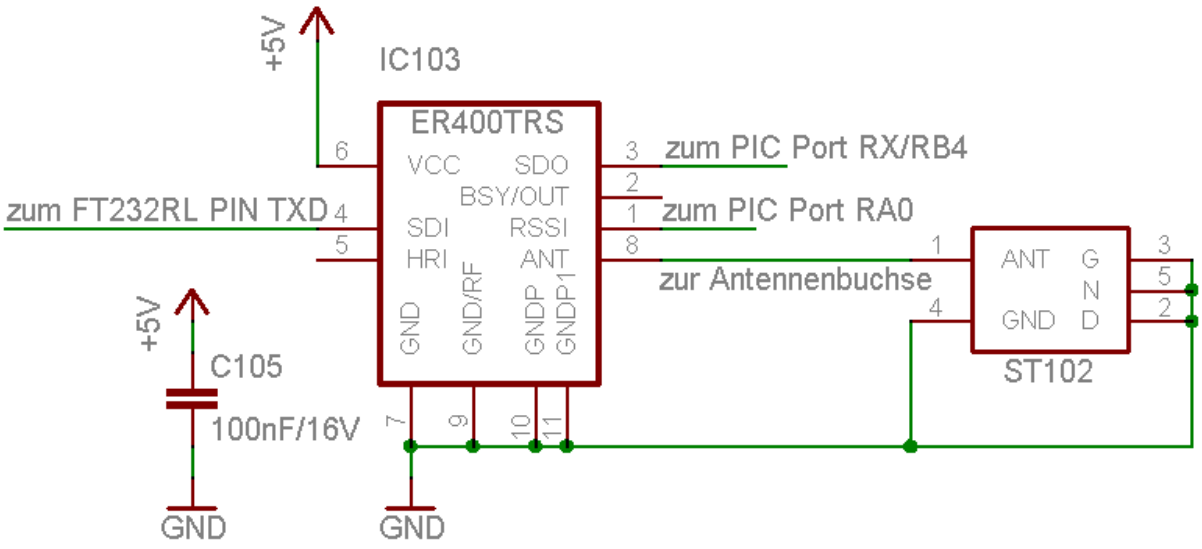
$$R103 = \frac{U_+ - U_{LED102}}{I_{LED}} = \frac{5V - 2V}{3mA} = 1k\Omega$$

gewählt:

R103 = 1kΩ / 0,125W

ER400TRS

Das Transceivermodul ER400TRS von der Firma Easy Radio, kann über RS232 angesprochen werden und hat insgesamt 10 verschiedene Funkkanäle zur Auswahl.
Das Modul sendet auf der normierten Industriefrequenz von 433MHz.
Die Sendeleistung des Funkmoduls beträgt bis zu 10mW und der Antennenanschluss hat eine Impedanz von 50Ω.
Weiters ist es mit diesem Modul möglich eine Kanalbelastungsmessung durchzuführen.
Es wurde dieses Modul gewählt, da es das preiswerteste war und nahezu alle Anforderungen erfüllt.



Das Funkmodul dient zum Übertragen der PC- Software- Befehle an alle umliegenden Slaves und zum Empfangen der Daten der Slaves.

Bauteilauswahl:
C105 dient zur Stabilisierung und wurde mit 100nF/16V gewählt.

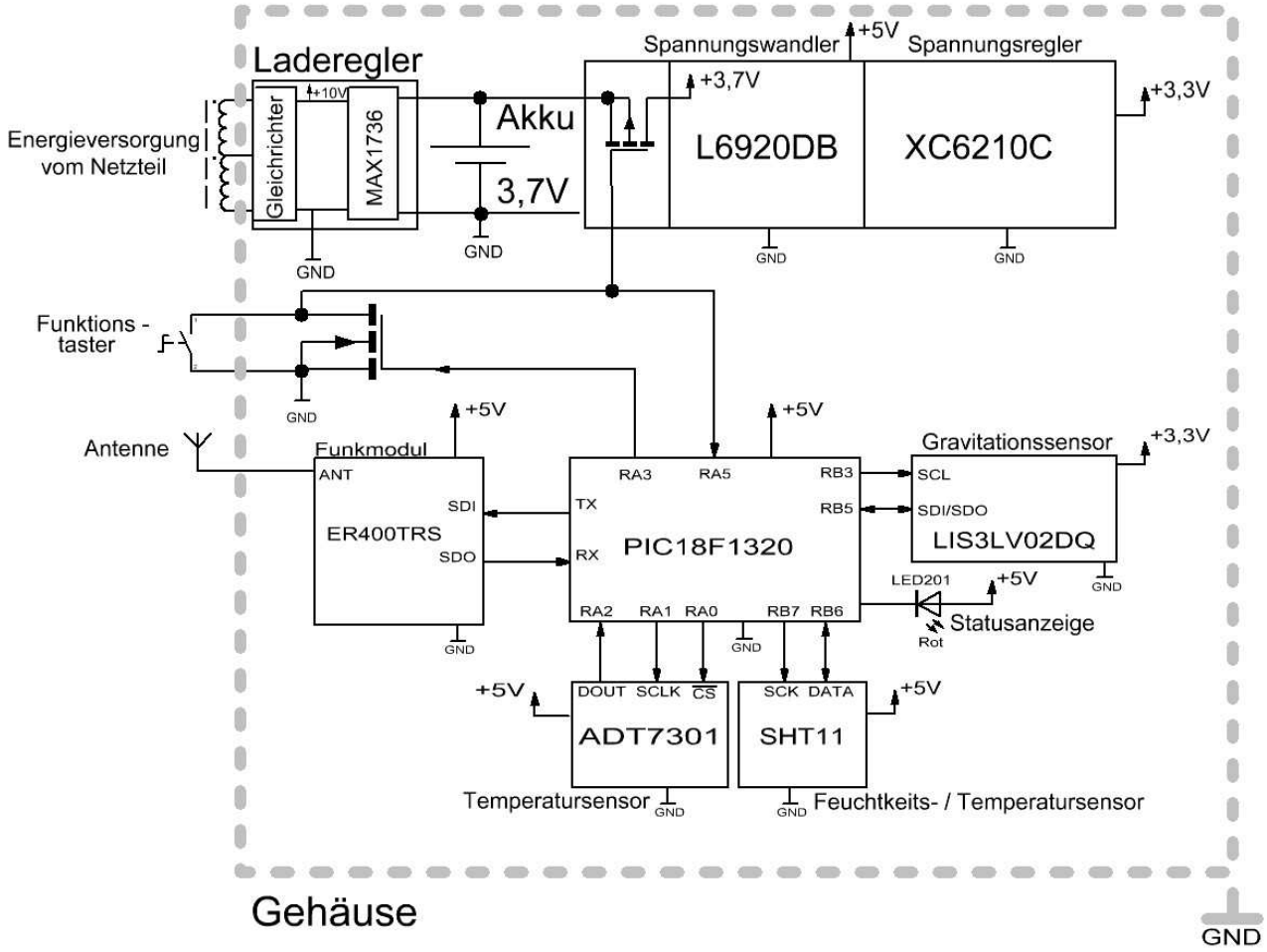
C105= 100µF / 16V

5.2.3. Slave

5.2.3.1. Allgemein

Der Slave hat als Aufgabe die Verarbeitung des Funkverkehrs und die Erfassung der Messdaten der Sensoren. Weiters soll der Slave auch als Relaisstation dienen können, um Messdaten und Befehle weiter leiten zu können.

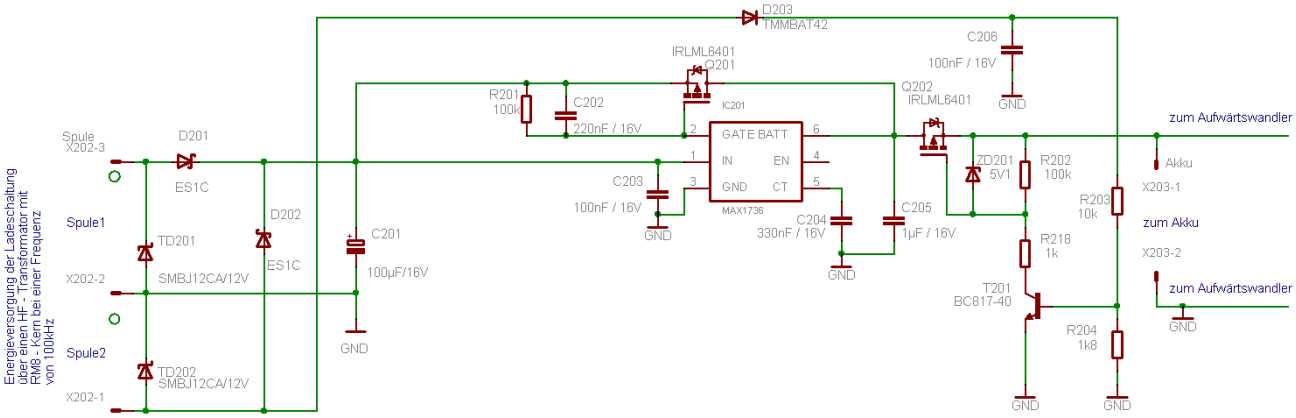
5.2.3.2. Blockschaltbild



5.2.3.3. Bauteilbeschreibungen/ Berechnungen

MAX1736

Der MAX1736 ist ein von der Firma MAXIM IC entwickelter Laderegler. Er ist speziell auf das Laden eines LION- Akku zugeschnitten. Diesen Baustein gibt es für die 3,6V und 3,7V Akku-Typen. In unserem Projekt wird der Typ für einen 3,7V Akku verwendet. Die Ladeschlussspannung dieses Bausteins beträgt 4,2V.



Die Transil- Transorb- Dioden TD201 und TD202 dienen als Überspannungsschutz und sorgen dafür, dass die Spannungsspitzen beim Schalten des Transformators gedämpft werden.
Die Dioden D2011 und D202 dienen zur 2- Weg- Gleichrichtung. Der Kondensator C201 dient zur Glättung der Spannung.
Die Kondensatoren C202, C203, C204 und C205 und der Widerstand R201 sind dem Datenblatt des MAX1756 entnommen.
Der P- Kanal- MOSFET Q201 wird vom MAX1736 angesteuert und je nach Ladungszustand des Akkus geschalten.
Der P- Kanal- MOSFET Q202 verhindert eine Rückentladung des Akkus in den Laderegler. Er wird durchgesteuert, sobald der Laderegler aktiv und der Transistor T201 durchgesteuert ist.
Mittels Diode D203 wird ein Teil der Transformatorspannung für die Ansteuerung des Transistors T201 benutzt. Der Kondensator C206 dient zur Glättung der Einweggleichrichtung. Die Widerstände R203 und R204 dienen zur Aussteuerung des Transistors.
R202 dient zur Ansteuerung des MOSFET Q202 und ZD201 dient zum Überspannungsschutz im Fehlerfall. R218 dient als Strombegrenzung beim Einschalten des MOSFET Q202.

<u>Bauteilauswahl:</u>
TD201 und TD202 wurden so gewählt, dass alle Spannungspitzen über 12V verhindert werden, damit der MOSFET Q201 nicht zerstört wird.
Dafür wurde die SMBJ12CA gewählt, eine bidirektionale Transil- Transorb- Diode.

TD201= TD202= SMBJ12CA 12V

D201 und D202 müssen so gewählt werden, dass sie bei einer maximalen Eingangsleistung von 3W und einer Maximalspannung von 12V nicht zerstört werden. Außerdem muss die Diode besonders schnell schalten können.
Wir haben uns für die Type ES1C entschieden, da sie 105V/ 1A kann, sehr schnell schaltet und lagernd war.

D201= D202= ES1C 105V/ 1A

D203 wurde nach einem I_F von 100mA und einer hohen Schaltfrequenz gewählt. Diese Diode muss keine hohen Sperrspannungen aushalten können. Wir haben uns für die Schottky- Diode TMMBAT42 auf Grund der oben angeführten Gründe entschieden.

D203= Schottky TMMBAT42 30V/200mA

C201 dient zur Glättung der Eingangsspannung des Ladeteils. Da die Glättung für den Laderegler nicht besonders gut sein musste, wurde dieser Kondensator mit 100µF gewählt.
Die Spannungsfestigkeit muss größer als 14V sein. Wir haben uns für die 16V Type entschieden.

C201= 100µF / 16V

C202 und C203 wurde mit C202= 220nF und C203=100nF laut Datenblatt gewählt und wie C201 mit einer Spannungsfestigkeit von 16V.

C202= 220nF / 16V
C203= 100nF / 16V

C204 und C205 wurden laut Datenblatt des MAX1736 mit C204 = 330nF und C205 = 1µF gewählt. Die Spannungsfestigkeit von C204 und C205 muss mindestens 5V betragen.

C204 = 330nF / 16V
C205 = 1µF /16V

R201 wurde laut Datenblatt des MAX1736 mit $R201 = 100k\Omega / 0{,}125W$ gewählt.

R201= 100kΩ / 0,125W

C206 dient zur Glättung und wurde mit 100nF gewählt, da er keine hohen Ströme liefern muss. Die Spannungsfestigkeit von 16V muss gewährleistet sein.

C206= 100nF / 16V

Als P - Kanal- MOSFET Q201 und Q202 wurde der Typ IRLML6401 gewählt, da er einen geringen R_{DSON} besitzt und für Ströme bis zu 4,3A ausgelegt ist. Die Spannungsfestigkeit von bis zu 12V ist für unsere Anwendung ausreichend.

Q201= Q202= IRLML6401 12V / 4,3A

ZD201 dient dazu um die U_{GS} auf maximal 5,1V zu begrenzen, um eine Zerstörung des MOSFET Q202 zu verhindern.

ZD201= 5V1 / 0,5W

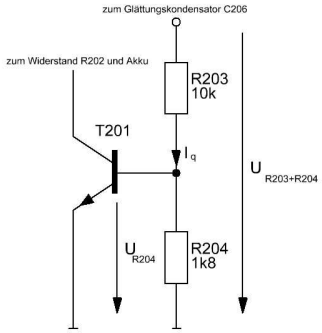
R202 dient zur Entladung der Gate – Kapazität des Q202 bei gesperrtem T201. Damit der Entladestrom des Gates nicht zu groß wird, wurde $R202 = 100k\Omega / 0{,}125W$ gewählt.

R202= 100kΩ / 0,125W

Die Auswahl des NPN - Transistors T201 unterlag keinen besonderen Aspekten. Es wurde der günstige Kleinsignaltransistor BC817- 40 gewählt.

T201= BC817- 40 45V / 500mA

Die Widerstände R203 und R204 wurden so gewählt, dass der Transistor T201 ab einer Eingangsspannung von 4V beginnt durchzuschalten.
Außerdem soll der Querstrom maximal 1mA betragen.
Berechnung der Widerstände R203 und R204:
Vorgaben:



$U_{R204} = 0{,}6V \ldots aus\ dem\ Datenblatt\ des\ BC817-40$

$U_{R203+R204\,min} = 4V \ldots \min imale\ Eingangsspannung; gewählt$

$U_{R203+R204\,max} = 10V \ldots \max imale\ Eingangsspannung; gewählt$

$I_{q\,max} = 1mA \ldots gewählt$

$$R203 = \frac{U_{R203+R204\,max} - U_{R204}}{I_{q\,max}} = \frac{10V - 0{,}6V}{1mA} = \frac{9{,}4V}{1mA} = 9400\Omega$$

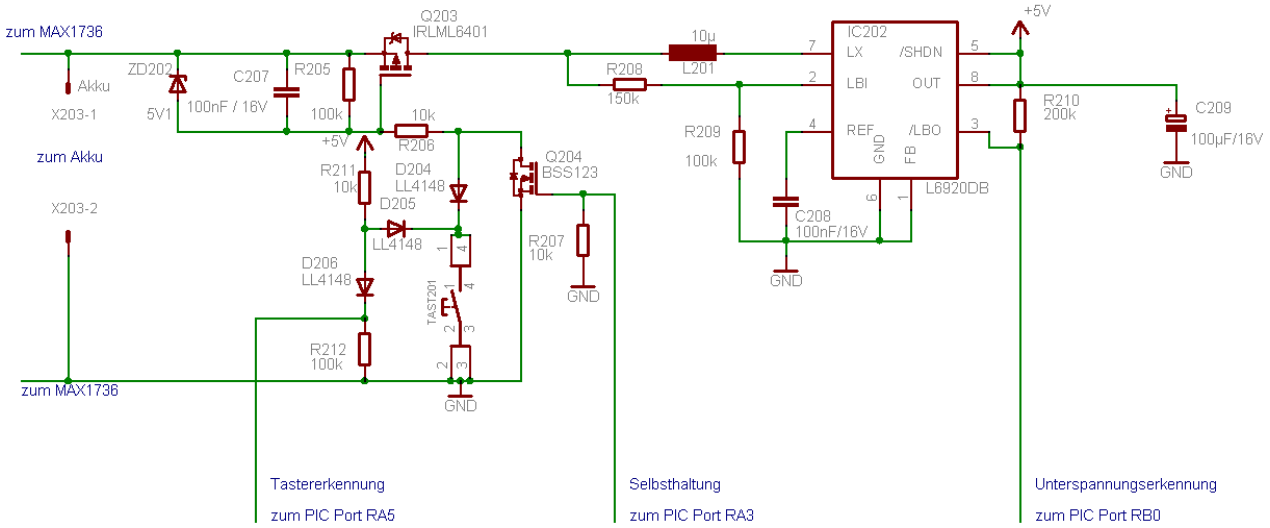
gewählt:
R203= 10kΩ / 0,125W

$$R204 = \frac{U_{R204} \cdot R203}{U_{R203+R204\,min} - U_{R204}} = \frac{0{,}6V \cdot 10k\Omega}{4V - 0{,}6V} = 1{,}67k\Omega$$

gewählt:
R204= 1.8kΩ / 0,125W

L6920DB

Der L6920DB ist ein Spannungswandler der Firma STMicroelectronics. Er ist speziell auf die Ausgangsspannungen 3,3V oder 5V ausgelegt. Der IC kann aber durch eine andere Beschaltung Ausgangsspannungen von 1,8V bis 5V erzeugen.
Für das Projekt wurde der Spannungswandler für eine Ausgangsspannung von 5V benutzt.



Die MOSFET's Q203 und Q204 dienen dazu, um den Slave auf- bzw. abzudrehen, wird der Taster TAST201 betätigt, so wird der P- Kanal- MOSFET Q203 durchgesteuert und der Spannungswandler mit dem Akku verbunden. Der PIC bekommt dadurch seine Versorgungsspannung und beginnt zu arbeiten. Der Mikrokontroller schaltet dann den Port PIN RA3 auf High und schaltet dadurch Q204 durch.
Der Taster kann nun losgelassen werden, da die Schaltung in Selbsthaltung geht.
Die Dioden D204, D205 und D206 werden dazu benötigt, damit der Taster auch während des Betriebes als Multifunktionstaster benutzt werden kann.
Wird der Taster während des Betriebes betätigt, so ist D204 in Sperrrichtung, da die Diode über Q204 gegen Masse hängt. Die Diode D205 wird durchlässig und zieht das Potential auf Masse. Dadurch sperrt die Diode D206 und R212 zieht das Potential auf Masse. Der Prozessor erkennt dadurch eine logisch 0. Wenn der Taster wieder losgelassen wird, erkennt der Prozessor wieder eine logisch 1.
Die Widerstände R208 und R209 dienen zum Einstellen des Low- Battery- Schwellenwertes.
Sobald die Spannung an R209 unter 1,23V fällt wird der PIN LBO auf 0 gesetzt.
L201, R210 und C208 wurden laut Datenblatt des L6920DB gewählt.
C209 dient zur Stabilisierung der Ausgangsspannung.
ZD202, R205 und C207 dienen zum Schutz und zur Ansteuerung des MOSFET Q203.
R206 dient als Strombegrenzung des MOSFET Q204.

Bauteilauswahl:
L201=10µH, R210=200kΩ und C208=100nF/16V wurden laut Datenblatt des L6920DB gewählt.
C209 wurde zur Glättung der Ausgangsspannung mit 100µF/16V gewählt. Für C209 muss die Spannungsfestigkeit größer als 5V sein.

L201= 10µH
R210= 200kΩ / 0,125W
C208= 100nF / 16V
C209= 100µF / 16V

Berechnung der Widerstände R208 und R209:
Vorgaben:

$U_{R209\min} = 1{,}23V \ldots laut\ Datenblatt\ des\ L6920DB$

$U_{R208+R209\min} = 3V \ldots \min imale\ Akkuspannung$

$R209 = 100k\Omega / 0{,}125W \ldots gewählt$

$$R208 = \frac{R209 \cdot U_{R208+R209\min} \cdot \left(1 - \frac{U_{R209\min}}{U_{R208+R209\min}}\right)}{U_{R209\min}} = \frac{100k\Omega \cdot 3V \cdot \left(1 - \frac{1{,}23V}{3V}\right)}{1{,}23V} = 143{,}9k\Omega$$

gewählt:
R208= 150kΩ / 0,125W
R209= 100kΩ / 0,125W

Für Q203 wurde der P- Kanal- MOSFET IRLML6401 gewählt, da er einen geringen R_{DSON} besitzt und für Ströme bis zu 4,3A ausgelegt ist. Die Spannungsfestigkeit von bis zu 12V ist für unsere Anwendung mehr als ausreichend.

Q203= IRLML6401 12V / 4,3A

ZD202 begrenzt die U_{GS} von Q3 auf 5,1V und verhindert somit eine Zerstörung des MOSFET.

ZD202= 5V1 / 0,5W

C207 schützt das Gate des MOSFET's Q203 vor Spannungsspitzen und wurde mit 100nF gewählt.

C207= 100nF / 16V

R205 dient als Entladewiderstand für das Gate des MOSFET Q203. Er wurde mit 100kΩ gewählt.

R205= 100kΩ / 0,125W

Für den N – Kanal - MOSFET Q204 wurde der Typ BSS123 gewählt, da er am preiswertesten war und für Ströme bis zu 100mA ausgelegt ist.

Q204= BSS123 100V / 150mA

R207 dient als Pull - Down Widerstand von Q204. Dies sorgt dafür, dass bei nicht angeschlossenem Mikrokontroller der MOSFET Q204 gesperrt ist.
Dieser Widerstand wurde mit $10k\Omega / 0{,}125W$ gewählt.

R207= 10kΩ / 0,125W

R206 dient zur Ansteuerung des MOSFET Q204 und wurde mit $10k\Omega / 0{,}125W$ gewählt.

R206= 10kΩ / 0,125W

Die Dioden D204, D205 und D206 unterlagen keinen besonderen Auswahlkriterien. Es wurde die preiswerte Diode LL4148 gewählt.

D204= D205= D206= LL4148 75V / 300mA

R211 dient als Pull - Up Widerstand und wurde mit $10k\Omega / 0{,}125W$ gewählt.

R211= 10kΩ / 0,125W

R212 wurde so gewählt, dass der PIC18F1320 eine logisch 1 erkennt, sobald der MOSFET Q204 durchgeschaltet und der Taster nicht betätigt wurde.

Berechnung von R212:
Vorgaben:

$U_{+} = 5V \ldots Versorgungsspannung$

$U_{R212\min} = 4V \ldots aus\ den\ Datenblatt\ des\ PIC18F1320$

$U_{D206} = 0{,}6V \ldots aus\ den\ Datenblatt\ der\ Diode\ LL4148$

$R11 = 10k\Omega \ldots gewählt$

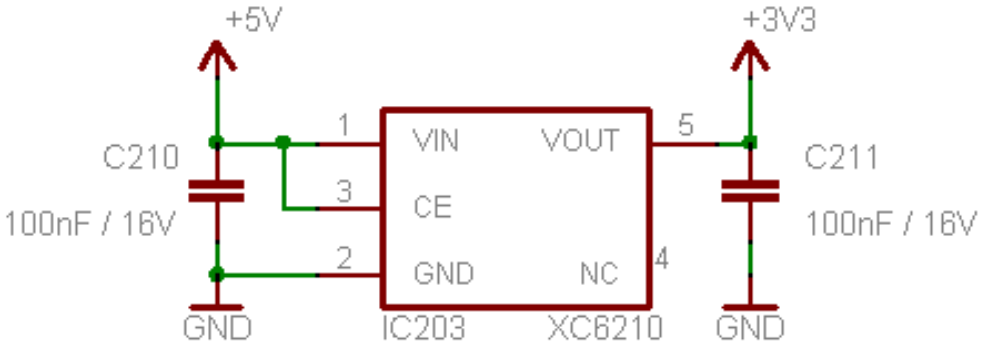
$$R212 = \frac{U_{R212\min} \cdot R211}{(U_{+} - U_{D206}) - U_{R212\min}} = \frac{4V \cdot 10k\Omega}{4{,}4V - 4V} = 100k\Omega$$

gewählt:

R212= 100kΩ / 0,125W

XC6210C

Der XC6210C Spannungswandler ist von der Firma TOREX speziell für Batterie betriebene Geräte entwickelt worden. Er zeichnet sich durch einen sehr niedrigen Stromverbrauch aus und ist in sehr kleinen Gehäusetypen verfügbar.
Dieser Chip ist für Ausgangsspannungen von 0,8V bis 5V erhältlich.
Da der Gravitationssensor eine Versorgungsspannung von 3,3V benötigt wurde die 3,3V Variante gewählt.



Der Spannungswandler dient zur Versorgung des Gravitationssensors LIS3LV02DQ.
Der Kondensator C210 wurde mit 100nF / 16V zur Stabilisierung der Eingangsspannung gewählt.

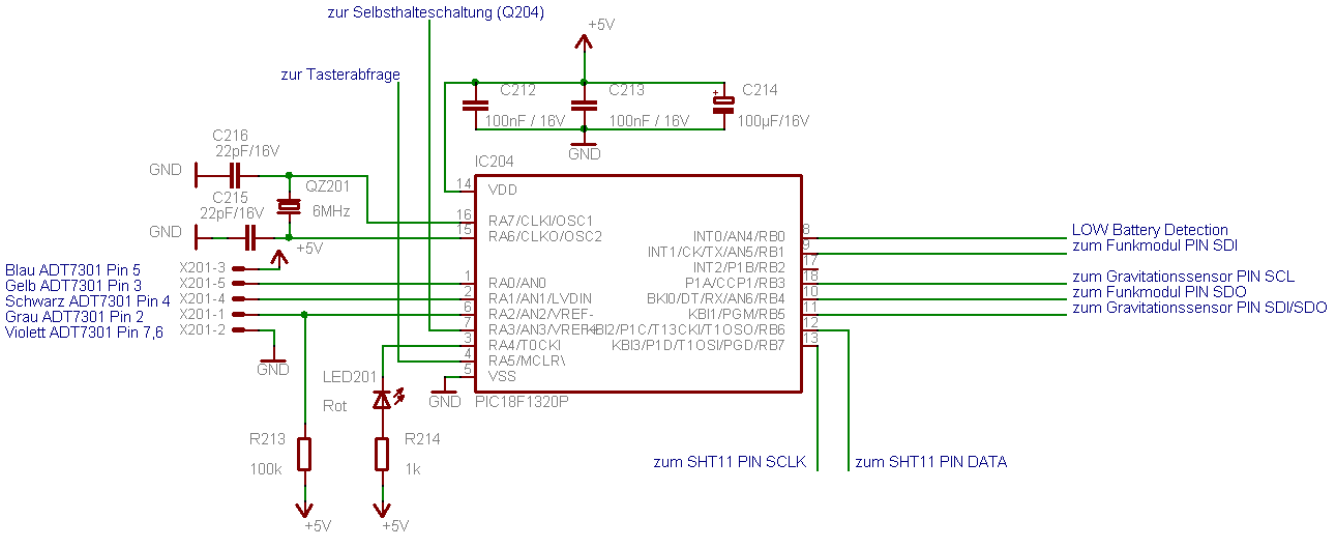
C210= 100nF / 16V

C211 wurde mit 100nF / 16V gewählt. Er dient zur Stabilisierung der Ausgangsspannung.

C211= 100nF / 16V

PIC18F1320

Der PIC18F1320 ist ein von der Firma Microchip entwickelter Mikrokontroller.
Er besitzt eine eingebaute USART Hardware, 4 Timer, einen EEPROM und spezielle Interrupt-Prioritäten. Außerdem besitzt er einen 8kB großen Programmspeicher.
Es wurde dieser Mikrokontroller gewählt, da der MCC18 - Compiler gratis zur Verfügung stand und bereits ein Vorwissen über die PIC – Familie vorhanden war.



Der PIC dient zur Erfassung der Messdaten, verarbeitet das Funkprotokoll und übernimmt die Verwaltung des Funktionstasters.

Der Quartz (QZ201) wurde mit 6MHz gewählt und dient zur Erzeugung der Oszillatorfrequenz.
Die Frequenz wird vom PIC durch die interne PLL vervierfacht.

QZ201= 6MHz

C215 und C216 wurden mit 22pF / 16V laut Datenblatt des PIC18F1320 gewählt.

C215= C216= 22pF / 16V

C213 und C214 dienen zur Stabilisierung der Versorgungsspannung des PIC.
C213 wurde mit 100nF / 16V gewählt und C214 mit 100µF / 16V.

C213= 100nF / 16V
C214= 100µF / 16V

R213 dient als Pull- Up Widerstand der DOUT- Leitung des ADT7301 und wurde mit $100k\Omega / 0{,}125W$ gewählt. Er dient hauptsächlich zum Feststellen, ob der Sensor vorhanden ist oder nicht.

R213= 100kΩ / 0,125W

R214 dient als Vorwiderstand der LED201. Die LED wird zum Ausführen des Befehls „Blinke LED“ und zum Anzeigen des Adressiermodus benötigt.

Berechnung des Vorwiderstandes R214:
Vorgaben:

$U_{+} = 5V \ldots Versorgungsspannung$

$U_{LED201} = 2V \ldots Flussspannung\ der\ roten\ LED$

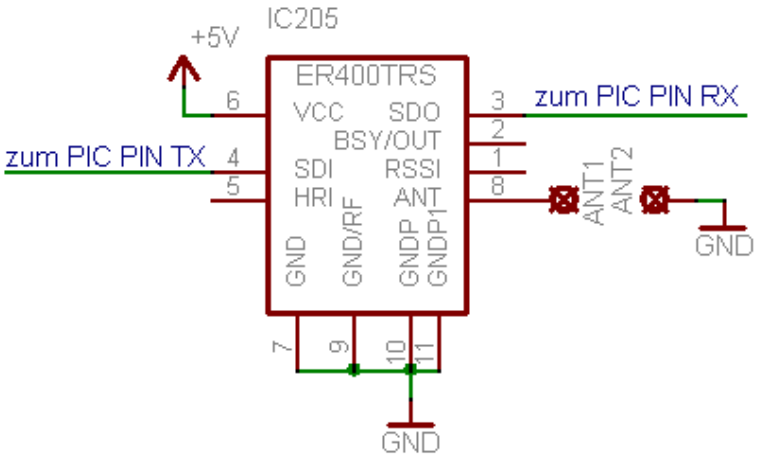
$I_{LED201} = 3mA \ldots Strom\ durch\ die\ LED$

$$R214 = \frac{U_{+} - U_{LED201}}{I_{LED201}} = \frac{5V - 2V}{3mA} = 1k\Omega$$

gewählt:
R214= 1kΩ / 0,125W

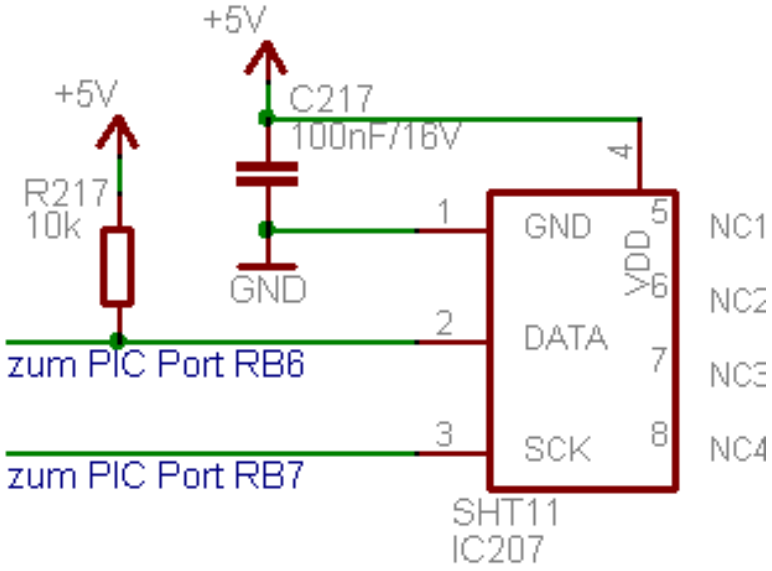
ER400TRS

Das Transceivermodul ER400TRS von der Firma Easy Radio, kann über RS232 angesprochen werden und hat insgesamt 10 verschiedene Funkkanäle zur Auswahl.
Das Modul sendet auf der normierten Industriefrequenz von 433MHz.
Die Sendeleistung des Funkmoduls beträgt bis zu 10mW und der Antennenanschluss hat eine Impedanz von 50Ω.
Weiters ist es mit diesem Modul möglich eine Kanalbelastungsmessung durchzuführen.
Es wurde dieses Modul gewählt, da es das preiswerteste war und nahezu alle Anforderungen erfüllt.



SHT11

Der SHT11 ist ein von der Firma Sensirion entwickelter Kombinationssensor. Er ist in der Lage Feuchtigkeit und Temperatur zu messen.
Er besitzt eine Auflösung von bis zu 14bit für Temperaturmessungen und bis zu 12bit für Feuchtigkeitsmessungen.
Der Einsatzbereich der Feuchtigkeitsmessung reicht von 0% bis 100% Luftfeuchtigkeit.
Temperaturmessungen können im Bereich von -40°C bis +123,8°C durchgeführt werden.



C217 dient zur Spannungsstabilisierung und wurde mit 100nF / 16V gewählt.

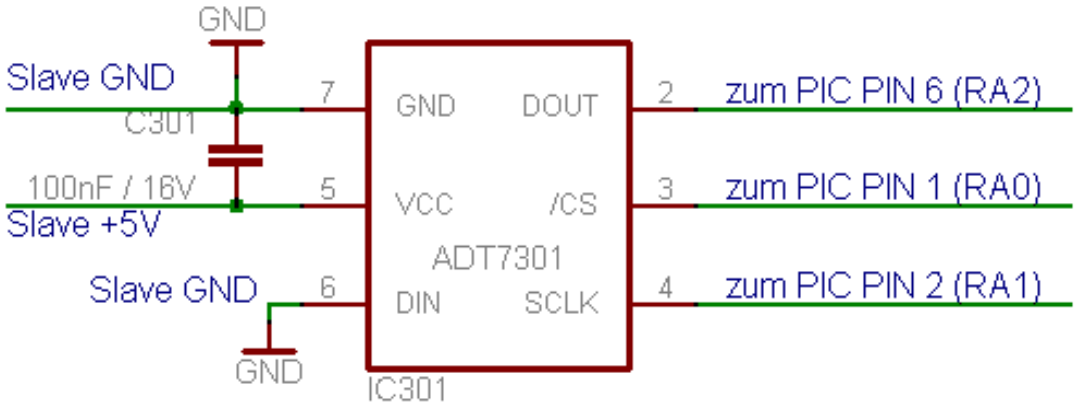
C217= 100nF / 16V

Der Widerstand R217 dient als Pull- Up Widerstand und wurde mit 10kΩ laut Datenblatt gewählt.

R217= 10kΩ / 0,125W

ADT7301

Der ADT7301 ist ein 13bit Temperatursensor der Firma Analog Devices.
Er kann für Messungen im Bereich von -40°C bis +150°C eingesetzt werden.

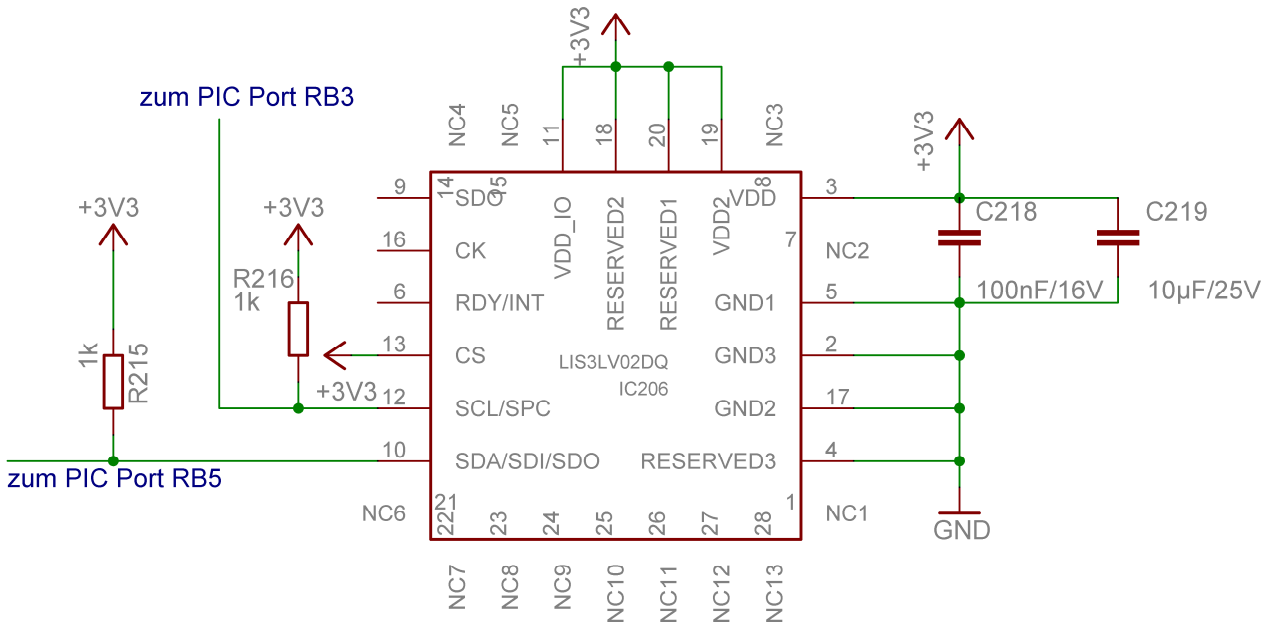


Der Kondensator C301 dient zur Spannungsstabilisierung und wurde mit 100nF / 16V gewählt.

C301= 100nF / 16V

LIS3LV02DQ

Der LIS3LV02DQ ist ein 16bit Gravitationssensor der Firma STMicroelectronics.
Er unterstützt eine Auflösung von bis zu ±6g.
Es wurde dieser Sensor gewählt, da er den Anwendungsbereich des Messsystems deutlich erhöht hat.



C218 und C219 sind Stützkondensatoren des IC206.
C218 wurde mit 100nF/16V und C219 mit 10µF/25V laut Datenblatt gewählt.

C218= 100nF / 16V
C219= 10µF / 25V

In unterschiedlichsten Tests wurde festgestellt, dass zur Pegelanpassung von 5V des PIC auf 3,3V des Gravitationssensors ein einfacher Pull- Up Widerstand ausreichend ist.
R215 und R216 dienen als Pull- Up Widerstand und wurden jeweils mit $1k\Omega / 0{,}125W$ gewählt

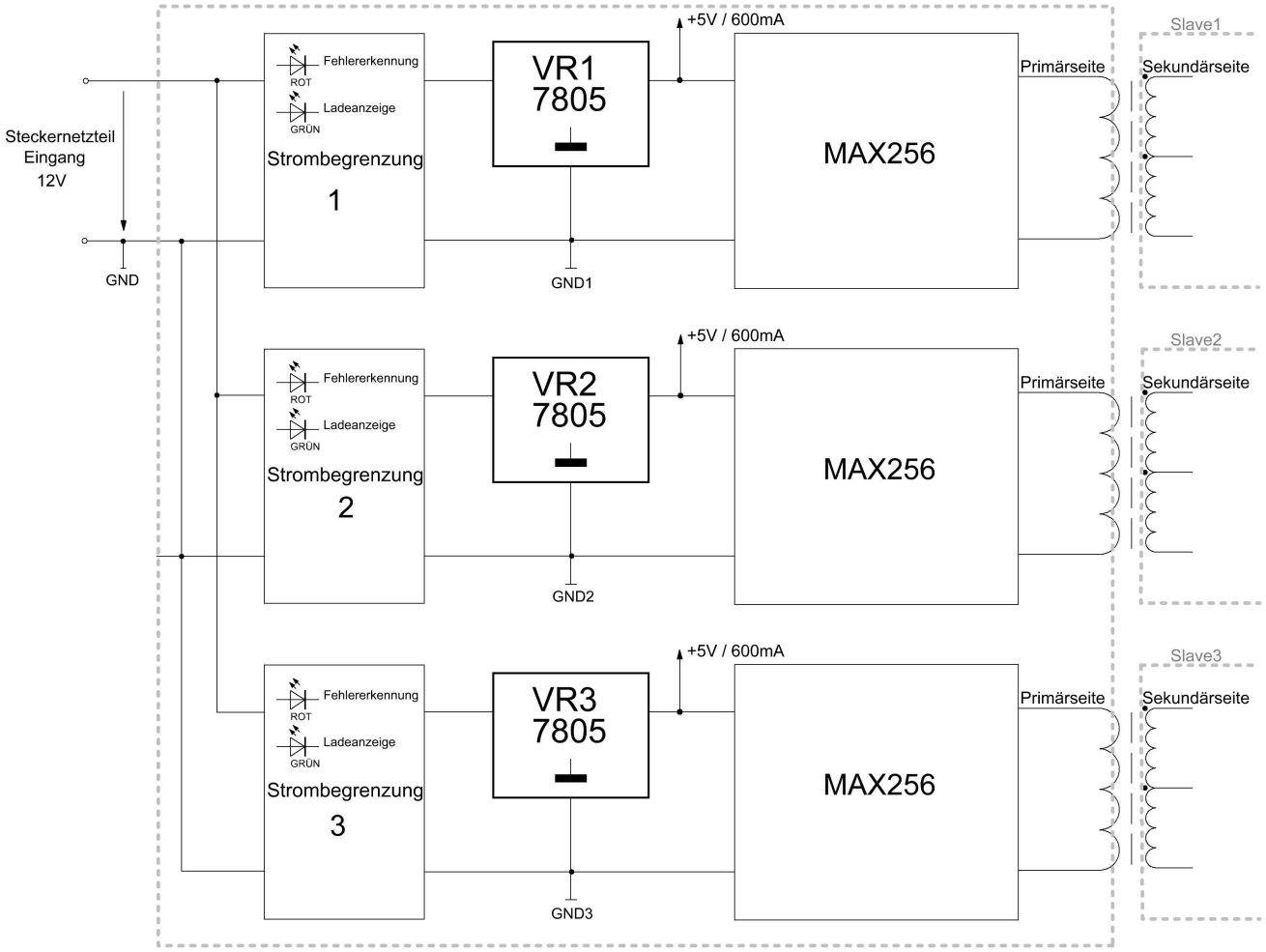
R215= 1kΩ / 0,125W
R216= 1kΩ / 0,125W

5.2.4. Netzteil

5.2.4.1. Allgemein

Das Netzteil dient dazu die Akkus von bis zu drei Slaves über induktive Kopplung zu laden.

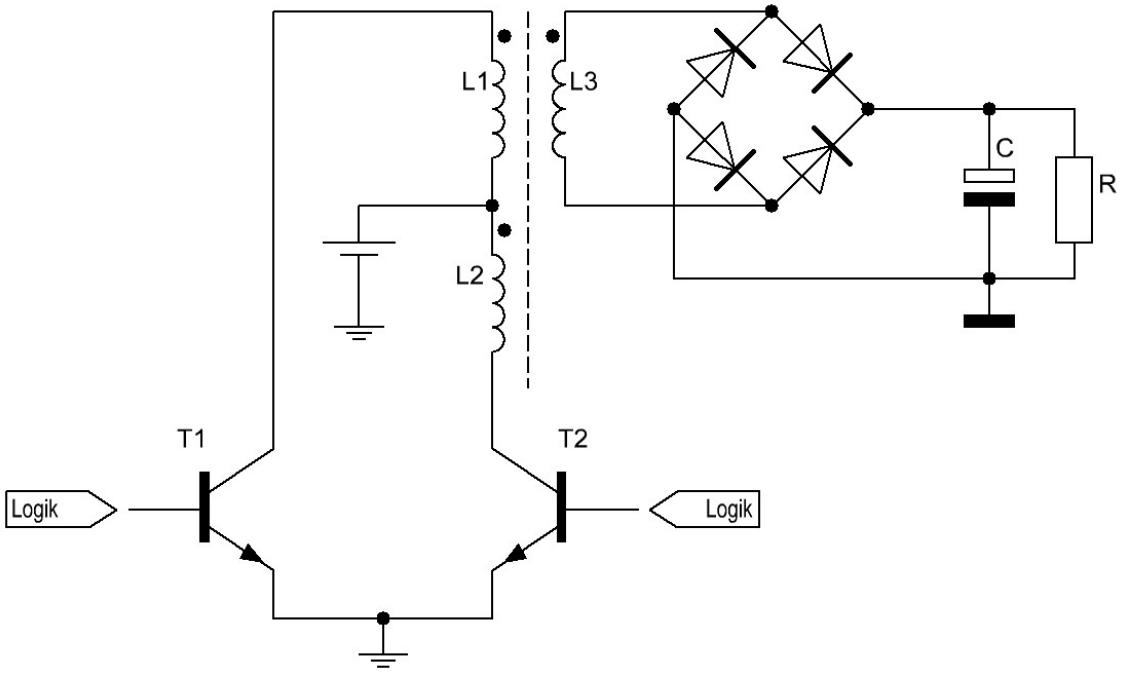
5.2.4.2. Blockschaltbild



5.2.4.3. Funktionsbeschreibung

Um die Funktion der Schaltung besser verstehen zu können wird zu nächst einmal das Prinzip eines Gegentaktdurchflusswandlers kurz erklärt.

Prinzipschaltung

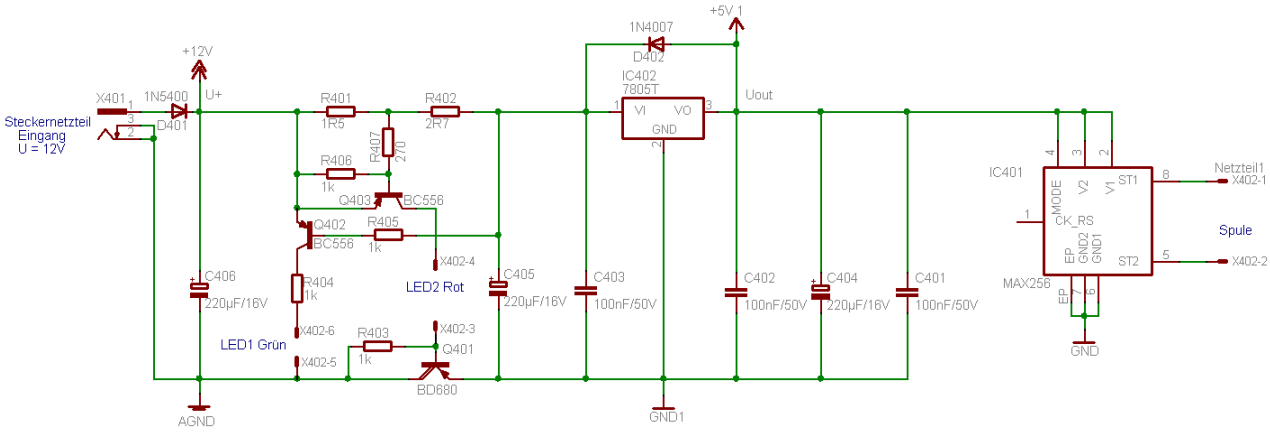


Die beiden Transistoren, welche als Schalter dienen, werden von einer Logik gegengleich geschalten. Dabei baut sich abwechselnd in der Spule L1 und einmal in der Spule L2 ein magnetisches Feld auf. Durch den ständigen Auf- und Abbau eines magnetischen Feldes wird wegen der magnetischen Kopplung in L3 von L1 und L2 eine Spannung induziert. Diese Spannung wird mittels Brückengleichrichter gleichgerichtet und mit einem geeigneten Kondensator geglättet.
Das Prinzip ist das des Gegentaktdurchflusswandlers.
Für die Diplomarbeit wird dieses Prinzip folgendermaßen ausgenutzt:
Die Spulen L1 und L2 befinden sich in einer Ladestation und dienen als Primärseite des Transformators. Die Spule L3, welche als Sekundärseite dient, wird in den Slave mit dem Akku integriert.
Will man den Akku nun laden so braucht man den Slave nur auf die Ladestation stellen und es kann eine drahtlose Energieübertragung erfolgen. Dies erspart eine zusätzliche Öffnung für eine Ladebuchse vorzusehen.
Die magnetische Kopplung ist aber, auf Grund dessen dass Primär- und Sekundärseite durch einen Luftspalt voneinander getrennt sind, schlechter.

In unserem Netzteil übernimmt das ständige Auf- und Abbauen des magnetischen Feldes, in einer Spule, der Push- Pull- IC MAX256 der Firma MAXIM IC.
Dieser funktioniert aber nach dem oben erläuterten Prinzip. Die beiden Transistoren zum Schalten sind bereits im IC integriert.
Er muss lediglich mit einer Versorgungsspannung von 5V versorgt werden und mit einer Spule beschalten werden.
Für die Sekundärseite wurde eine 2- Weg- Gleichrichtung gewählt, um eine Diodenflussspannung einzusparen.

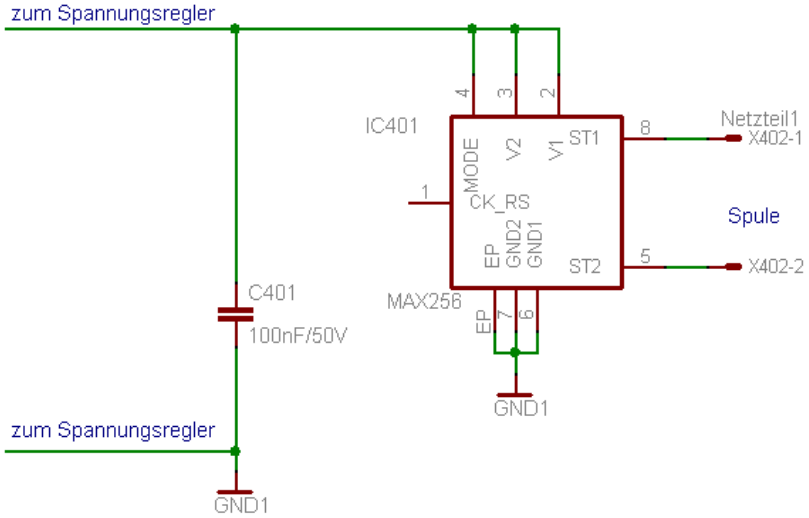
5.2.4.4. Bauteilbeschreibungen/ Berechnungen

Sämtliche Bauteilberechnungen werden für die unten angeführte Ladestufe durchgeführt.
Die Bauteilwerte der anderen beiden Stufen sind, wegen der Symmetrie, äquivalent.



MAX256

Der MAX256 ist ein von der Firma MAXIM IC entwickelter Push – Pull - IC. Er ist für Leistungen bis zu 3W ausgelegt und kann mit 3,3V oder 5V versorgt werden. In unserem Projekt wird der Baustein mit einem extern zugekauften 12V/3,33A Steckernetzteil versorgt.
Die Schaltfrequenz ist von 100kHz bis zu 1MHz einstellbar.



Es gab beim Erstentwurf eines Schaltnetzteils Probleme mit der Stromkommutierung und der Abschaltung bei Überstrom. Deshalb wurde ein Push – Pull - IC verwendet.
Für die Spulen wird ein 1:2 Verhältnis der Windungen gewählt, da auf der Sekundärseite eine Ausgangsspannung von circa 10V erreicht werden soll.
Die Sekundärspule wird, zwecks 2- Weg- Gleichrichtung, mit 2 Spulen ausgeführt.

C401 dient als Stabilisierungskondensator des MAX256 und wurde mit 100nF / 50V gewählt.
Das Datenblatt empfiehlt zur Stabilisierung einen 4,7µF und einen 470nF Kondensator.
Da aber zur Ausgangsstabilisierung des Spannungsreglers bereits ein 220µF und ein 100nF Kondensator benutzt wird, wurde beschlossen, bloß mit einem weiteren 100nF Kondensator zu stabilisieren.

C401= 100nF / 50V

wegen der Schaltungssymmetrie ergibt sich für die Kondensatoren C412 und C418 der anderen Stufen:

C412= C418= 100nF / 50V

Berechnung der Spulen

Zur Berechnung der Spulen wurden folgende Annahmen getroffen:
Vorgaben:

$I_{MAX} = 1A$…maximaler Spitzenstrom durch die Spule
$f = 100kHz$ …Schaltfrequenz des Reglers
$V_{DD} = 5V$ …Versorgungsspannung der Schaltung

Daraus ergibt sich dann folgendes:

$$\Delta\Phi = L \cdot \Delta I = V_{DD} \cdot \Delta t$$

$$\Delta t = \frac{T}{2} = \frac{10\mu s}{2} = 5\mu s$$ …Wahl von Δt um ein 50:50 Tastverhältnis zu erhalten

$$\rightarrow L = \frac{V_{DD} \cdot \Delta t}{\Delta I} = \frac{5V \cdot 5\mu s}{1A} = 25\mu H$$

Im nächsten Schritt wird die Windungszahl der Spule berechnet.

Berechnung der benötigten Windungszahl:

$$\Theta = R_m \cdot \Phi$$

für mehr als eine Windung gilt

$$\frac{R_m}{N} \cdot \Delta\Phi = N \cdot \Delta I$$

setzt man für $\frac{\Delta\Phi}{\Delta I} = L$ und $A_L = \frac{1}{R_m}$ so ergibt sich für die Windungszahl N

$$N = \sqrt{\frac{L}{A_L}}$$

Um den Wert A_L bestimmen zu können, musste eine Messung durchgeführt werden. Dafür wurde der Transformator mit den drei Spulen mit je 10 Windungen bewickelt. Im Kern des Trafos wurde auch ein Luftspalt vorgesehen (~1mm), welcher später zwangsweise durch das Gehäuse des Slaves gegeben ist.

Gemessener Wert für 10 Windungen bei einer Schaltfrequenz von 1kHz:

$L_{gem} = 21{,}24\mu H$

Berechnung der benötigten Windungszahl:

$$A_L = \frac{L_{gem}}{N^2} = \frac{21{,}24\mu H}{10^2} = 212{,}4nH$$

Berechnung der endgültigen Windungszahl um ein $L = 25\mu H$ zu erhalten

$$N = \sqrt{\frac{L}{A_L}} = \sqrt{\frac{25\mu H}{212{,}4nH}} = 10{,}8$$ gewählt: N = 10 Windungen

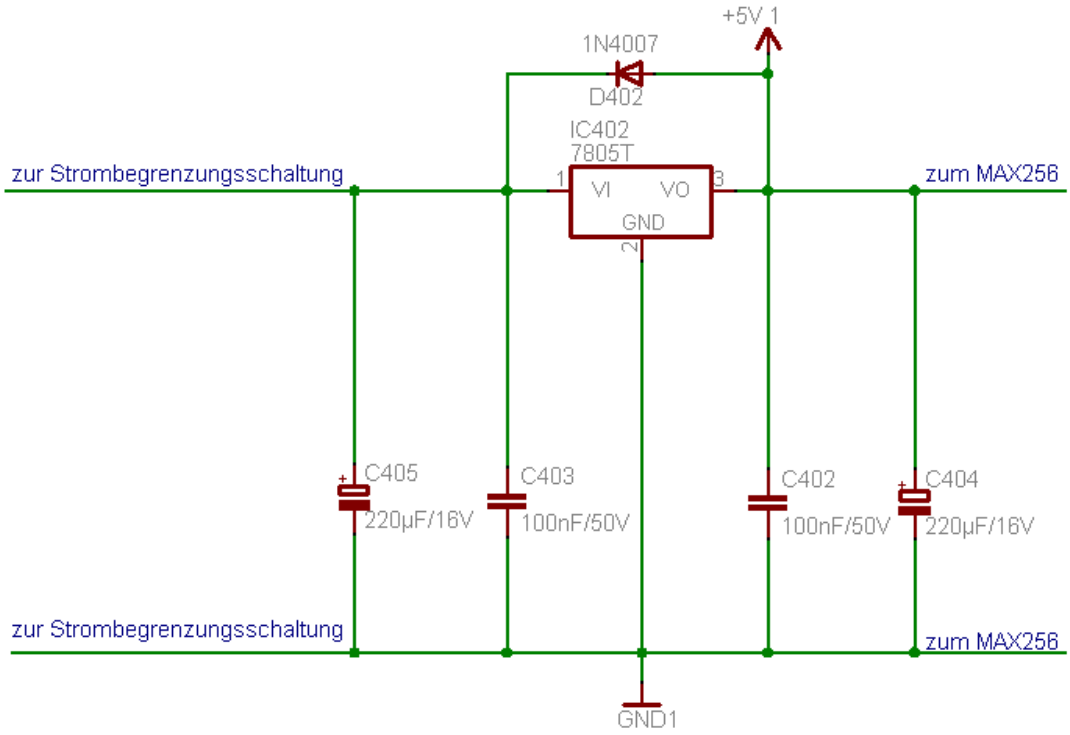
Dies sind nun die Windungen der Primärseite, um ein Spannungsverhältnis von 1:2 zu erhalten beträgt die Windungszahl der Sekundärseite pro Spule 20Windungen.

Auf Grund der Gehäuseform des Netzteils und des Slaves wurde festgestellt, dass die Energieübertragung schlechter als erwartet war.
Deshalb wurde experimentell die optimale Windungszahl ermittelt. Dabei erwiesen sich 14 Windungen für die Primärseite und 20 Windungen die Sekundärseite als optimal.

$N_{Primär}$ = 14 Windungen
$N_{Sekundär}$ = 20 Windungen

LM7805

Der LM7805 ist ein preiswerter 5V Spannungsregler.



Der LM7805 dient zum Erzeugen der 5V Versorgungsspannung des MAX256.
Die Kondensatoren C402 bis C405 dienen zur Stabilisierung und die Diode D402 dient als Schutzdiode des LM7805.

<u>Dimensionierung des Kühlkörpers für den LM7805</u>

Vorgaben:

$I_{MAX} = 500mA \ldots \max imaler\ Strom$

$U_{+} = 12V \ldots Versorgungsspannung$

$U_{OUT} = 5V \ldots Ausgangsspannung\ des\ LM\,7805$

$U_{CESATQ\,401} = 0{,}9V \ldots aus\ dem\ Datenblatt\ des\ BD680\ bei\ I = 500mA$

$T_j = 100°C \ldots gewählt$

$T_u = 25°C \ldots gewählt$

$R_{THJC} = 5°/W \ldots aus\ dem\ Datenblatt\ des\ LM\,7805$

$R_{THCH} \approx 0{,}5°/W \ldots Annahme$

Berechnung:

$$P_V = \left(U_{+} - U_{OUT} - U_{CESATQ\,401} - I_{MAX} \cdot \left(R401 + R402\right)\right) \cdot I_{MAX}$$

$$P_V = \left(12V - 5V - 0{,}9V - 500mA \cdot \left(1{,}5\Omega + 2{,}7\Omega\right)\right) \cdot 500mA = 2W$$

$$R_{THges} = \frac{T_j - T_u}{P_V} = \frac{100° - 25°}{2W} = 37{,}5°/W$$

$$R_{THHA} = R_{THges} - R_{THJC} - R_{THCH} = 37{,}5°/W - 5°/W - 0{,}5°/W = 32°/W$$

Der gesamte Kühlkörper muss demnach 10,6°/W für die drei Spannungsregler haben.

$R_{THHA3SRges}$ = 10°/W

Die Kondensatoren C403 und C405 dienen zur Eingangsspannungsstabilisierung des LM7805. C403 wurde mit 100nF / 50V gewählt und C405 wurde mit 220µF / 16V gewählt.

wegen der Schaltungssymmetrie ergibt sich für die Kondensatoren C403, C411 und C417:

C403= C411= C417= 100nF / 50V

wegen der Schaltungssymmetrie ergibt sich für die Kondensatoren C405, C409 und C415:

C405= C409= C415= 220µF / 16V
Die Kondensatoren C403 und C405 dienen zur Ausgangsspannungsstabilisierung des LM7805. C402 wurde mit 100nF / 50V gewählt und C404 wurde mit 220µF / 16V gewählt.

wegen der Schaltungssymmetrie ergibt sich für die Kondensatoren C402, C410 und C416:

C402= C410= C416= 100nF / 50V

wegen der Schaltungssymmetrie ergibt sich für die Kondensatoren C404, C408 und C414:

C404= C408= C414= 220µF / 16V

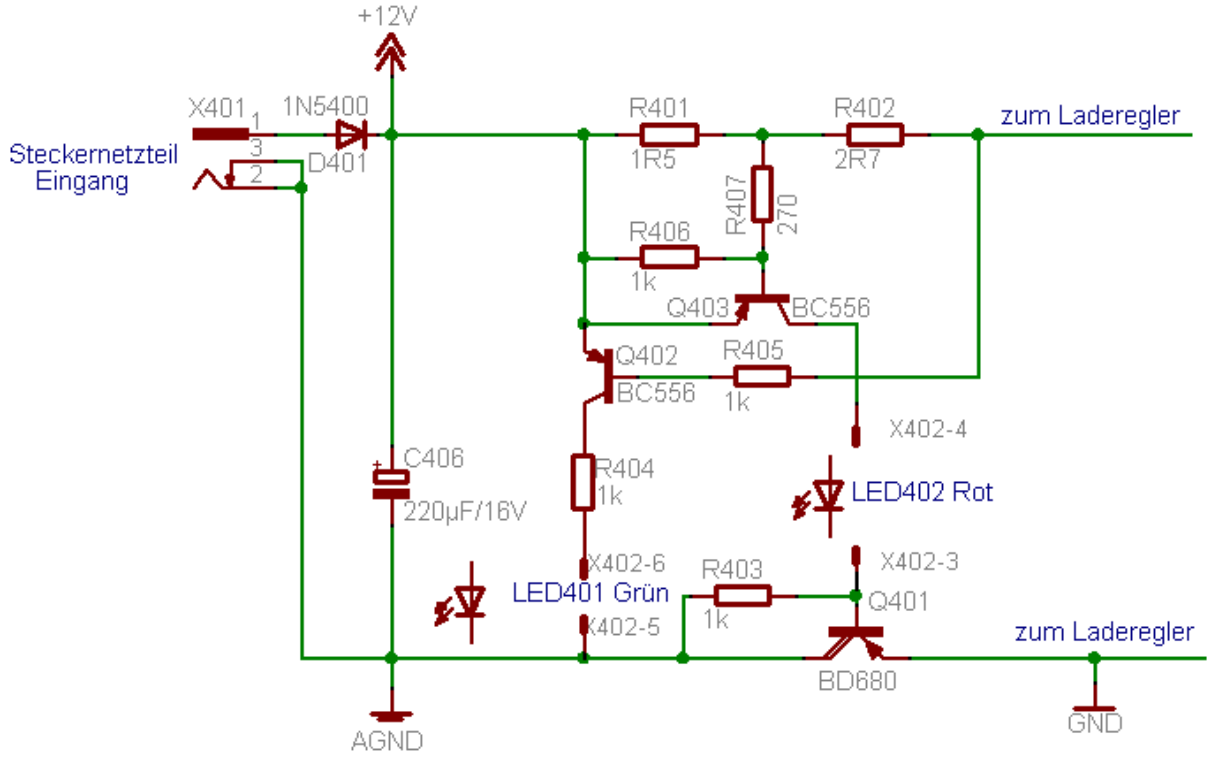
Die Diode D402 dient zum Schutz des LM7805. Sie sorgt dafür, dass beim plötzlichen Abschalten der Eingangsspannung die geladenen Stabilisierungskondensatoren nicht dafür sorgen, dass die Ausgangsspannung nicht viel positiver als die Eingangsspannung werden kann.
Als Schutzdiode wurde die 1N4007 gewählt.

wegen der Schaltungssymmetrie ergibt sich für die Schutzdioden D402, D403 und D404:

D402= D403= D404= 1N4007 1000V / 1A

Strombegrenzung

Die Strombegrenzung dient zum Schutz des MAX256, zur Anzeige ob ein Akku geladen wird und zur Fehleranzeige.
Sie sorgt dafür, dass jede beliebige 12V Spannungsquelle angelegt werden kann ohne, dass der MAX256 zerstört wird.



Sobald ein Strom von 150mA fließt, wird der Transistor Q402 durchgesteuert und die grüne LED401 beginnt zu leuchten. Dies signalisiert, dass ein Akku geladen wird.
Ab einem Stromverbrauch von 500mA liegt ein Fehler vor und der Transistor Q403 wird durchgesteuert. Dies sorgt dafür, dass die rote LED402 zu leuchten beginnt. Dadurch wird der Transistor Q401 gesperrt und die Strombegrenzung tritt ein.
Die Diode D401 dient zum Verpolungsschutz der Eingangsspannung.

<u>Wahl von R401, R411 und R418</u>
R401 wird so gewählt, dass an R401 eine Spannung von 0,75V bei 500mA anliegt, damit der Transistor Q403 vollständig durchgesteuert werden kann und der Transistor Q401 gesperrt wird.

Vorgaben:

$U_{R401MAX} = 0{,}75V \ldots gewählt$

$I_{MAX} = 500mA \ldots Strombegrenzung; gewählt$

$$R401 = \frac{U_{R401MAX}}{I_{MAX}} = \frac{0{,}75V}{500mA} = 1{,}5\Omega$$

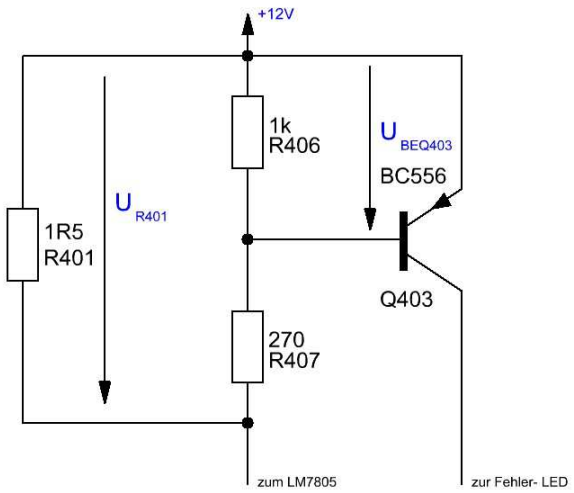
$$P_{R401MAX} = I_{MAX}{}^2 \cdot R401 = (0{,}5A)^2 \cdot 1{,}5\Omega = 375mW$$

gewählt:
R401= 1,5Ω / 1W

Da es sich um drei identische Ladestationen handelt gilt:

R401= R411= R418= 1,5Ω / 1W

Berechnung von R406 und R407 bzw. R413 und R414 bzw. R420 und R421
R406 und R407 dienen zum sanfteren Einsatz der Strombegrenzung.
Sobald an R401 eine Spannung von 0,75V anliegt, soll die Spannung an R406 0,6V betragen, damit der Transistor Q403 vollständig durchgesteuert und somit der Transistor Q401 gesperrt wird.



Vorgaben:

$U_{BEQ403} = 0{,}6V \ldots\ldots aus\ dem\ Datenblatt\ des\ BC556$

$U_{R401MAX} = 0{,}75V \ldots gewählt$

$R406 = 1k\Omega \ldots gewählt$

$$R407 = \left(\frac{U_{R401MAX}}{U_{BEQ403}} - 1 \right) \cdot R406 = \left(\frac{0{,}75V}{0{,}6V} - 1 \right) \cdot 1k\Omega = 250\Omega$$

gewählt:
R406= 1kΩ / 0,25W
R407= 270Ω / 0,25W

Da es sich um drei identische Ladestationen handelt gilt:

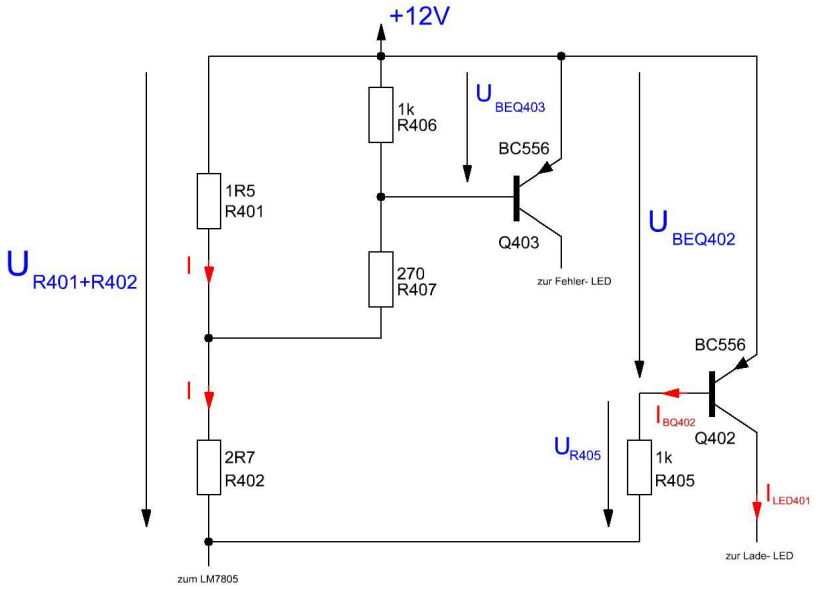
R406= R413= R420= 1kΩ / 0,25W
R407= R414= R421= 270Ω / 0,25W

Berechnung von R402 und R405 bzw. R412 und R408 bzw. R419 und R415

R402 wird so gewählt, dass an R401 und R402 eine Spannung von 0,6V ab 150mA anliegt. Dies sorgt dafür, dass Q402 vollständig durchgesteuert wird und damit die LED401 zu leuchten beginnt.

Die grüne LED401 dient zum Signalisieren, dass der Akku noch geladen wird. Sobald der Akku geladen ist, erlischt die LED401.

Der Widerstand R405 dient als Basisvorwiderstand und sorgt dafür, dass der Transistor Q402 nicht zerstört wird.



Berechnung von R402, R412 und R419

Vorgaben:

$U_{BEQ402} = 0{,}6V \ldots aus\ dem\ Datenblatt\ des\ BC556$

$U_{R405} \approx 0V \ldots wird\ \ vernachlässigt$

$I_{MIN} = 150mA \ldots gewählt$

$I_{MAX} = 500mA \ldots gewählt$

$R401 = 1{,}5\Omega$

$$R401 + R402 = \frac{U_{BEQ402}}{I_{MIN}}$$

$$R402 = \frac{U_{BEQ402}}{I_{MIN}} - R401 = \frac{0{,}6V}{150mA} - 1{,}5\Omega = 2{,}5\Omega$$

$$P_{R402MAX} = I_{MAX}{}^2 \cdot R402 = (0{,}5A)^2 \cdot 2{,}5\Omega = 625mW$$

gewählt:

R402= 2,7Ω / 1W

Da es sich um drei identische Ladestationen handelt gilt:

R402= R412= R419= 2,7Ω / 1W

Berechnung von R405, R408 und R415

Vorgaben:

$I_{MIN} = 150mA \ldots gewählt$

$R402 = 2{,}7\Omega$

$U_{BEQ402} = 0{,}6V \ldots aus\ dem\ Datenblatt\ des\ BC556$

$\beta_{Q402} = 100 \ldots aus\ dem\ Datenblatt\ des\ BC556$

$I_{LED401MIN} = 3mA \ldots gewählt$

$$I_{BQ402} = \frac{I_{LED401MIN}}{\beta_{Q402}} = \frac{3mA}{100} = 30\mu A$$

$$U_{R401+R402} = I_{MIN} \cdot (R401 + R402) = 150mA \cdot (1{,}5\Omega + 2{,}7\Omega) = 0{,}63V$$

$$U_{R405} = U_{R401+R402} - U_{BEQ402} = 0{,}63V - 0{,}6V = 30mV$$

$$R405 = \frac{U_{R405}}{I_{BQ402}} = \frac{30mV}{30\mu A} = 1k\Omega$$

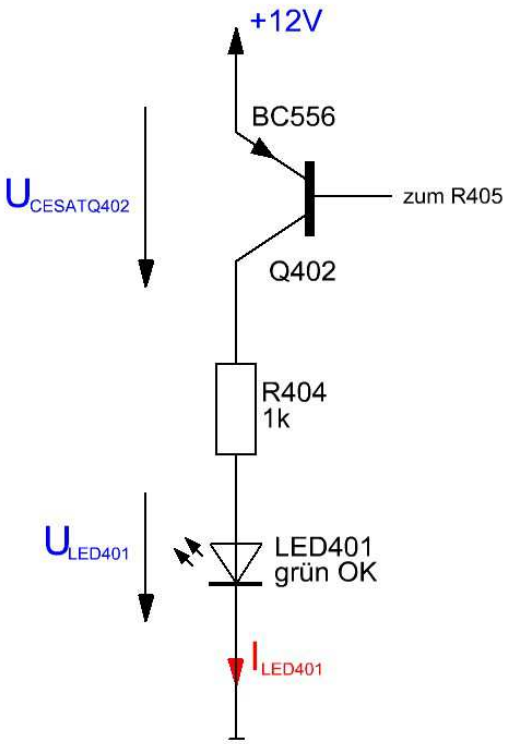
gewählt:

R405= 1kΩ / 0,25W

Da es sich um drei identische Ladestationen handelt gilt:

R405= R408= R415= 1kΩ / 0,25W

Berechnung von R404, R409 bzw. R416
R404 dient als Vorwiderstand der LED401.



Vorgaben:

$U_{CESATQ402} = 0{,}3V \ldots aus\ dem\ Datenblatt\ des\ BC556; Sättigungsspannung\ von\ Q2$

$U_{+} = 12 \ldots Spannung\ des\ Stec\,\text{ker}\ netzteils$

$U_{LED401} = 2V \ldots Flussspannung\ der\ LED$

$I_{LED401} = 10mA \ldots \max imaler\ Strom\ durch\ die\ LED; gewählt$

$$R404 = \frac{U_{+} - U_{LED401} - U_{CESATQ402}}{I_{LED401}} = \frac{12V - 2V - 0{,}3V}{10mA} = 970\Omega$$

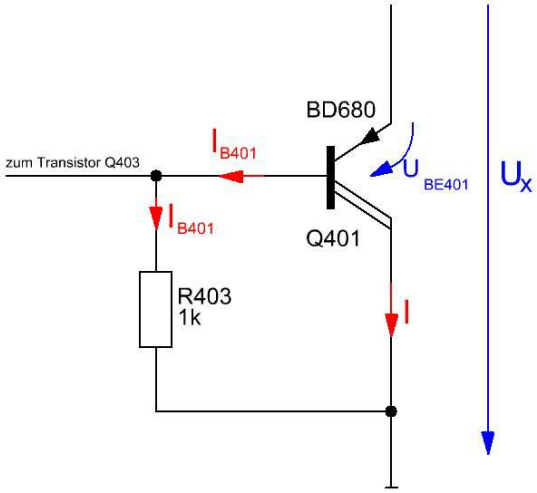
gewählt:
R405= 1kΩ / 0,25W

Da es sich um drei identische Ladestationen handelt gilt:

R404= R409= R416= 1kΩ / 0,25W

Berechnung von R403, R410 bzw. R417

R403 dient zur Aussteuerung von Q401 und soll bei maximalem Strom die maximale Spannung haben.



Vorgaben:

$U_{BE\,401} = 1{,}5V \ldots aus\ dem\ Datenblatt\ des\ BD680\ bei\ 500mA$

$B401 = 750 \ldots aus\ dem\ Datenblatt\ des\ BD680$

$I_{MAX} = 500mA \ldots \max imaler\ Strom$

$U_{+} = 12V \ldots Spannung\ des\ Stec\ker\ netzteils$

$U_{IC\,402} = 7{,}7V \ldots \min imale\ Eingangsspannung\ des\ LM\,7805;\ gewählt$

$U_x \ldots \mathrm{Re}\, stspannung\ der\ Schaltung$

$$U_x = U_{+} - U_{IC\,402} - I_{MAX} \cdot (R401 + R402) = 12V - 7{,}7V - 500mA \cdot (1{,}5\Omega + 2{,}7\Omega) = 2{,}2V$$

$$I_{B401MAX} = \frac{I_{MAX}}{B} = \frac{500mA}{750} = 666{,}7\mu A$$

$$R403 = \frac{U_x - U_{BE\,401}}{I_{B401MAX}} = \frac{2{,}2V - 1{,}5V}{666{,}7\mu A} = 1{,}05k\Omega$$

gewählt:

R403= 1kΩ / 0,25W

Da es sich um drei identische Ladestationen handelt gilt:

R403= R410= R417= 1kΩ / 0,25W

Wahl der Transistoren Q402 und Q403 bzw. Q405 und Q406 bzw. Q408 und Q409

Diese Transistoren unterlagen bei der Auswahl keinen besonderen Kriterien. Es wurde der preiswerte Kleinsignal PNP- Transistor BC556 gewählt.

Wahl der Transistoren Q401 bzw. Q404 bzw. Q407

Diese PNP - Transistoren müssen eine hohe Stromverstärkung haben, um eine bessere Regelgenauigkeit zu erreichen. Weiters müssen diese Transistoren einen Strom von 600mA treiben können.

Deshalb wurde der PNP - Darlingtontransistor BD680 gewählt.

Dimensionierung des Kühlkörpers für den Q401

Vorgaben:

$I_{MAX} = 500mA \ldots \max imaler\ Strom$

$U_{+} = 12V \ldots Spannung\ des\ Stec\ker netzteils$

$\Delta U_{LM\,7805MIN} = 1{,}5V \ldots \min imale\ Differenzspannung\ des\ LM\,7805\ aus\ Datenblatt$

$T_j = 125°C \ldots gewählt$

$T_u = 25°C \ldots gewählt$

$R_{THJC} = 3{,}5°/W \ldots aus\ dem\ Datenblatt\ des\ BD680$

$R_{THCH} \approx 0{,}5°/W \ldots Annahme$

Berechnung:

$P_V = \left[U_{+} - \Delta U_{LM\,7805MIN} - \left(R401 + R402\right) \cdot I_{MAX}\right] \cdot I_{MAX}$

$P_V = \left[12V - 1{,}5V - \left(1{,}5\Omega + 2{,}7\Omega\right) \cdot 500mA\right] \cdot 500mA = 4{,}2W$

$$R_{THges} = \frac{T_j - T_u}{P_V} = \frac{125° - 25°}{4{,}2W} = 23{,}8°/W$$

$$R_{THHA} = R_{THges} - R_{THJC} - R_{THCH} = 23{,}8°/W - 3{,}5°/W - 0{,}5°/W = 19{,}8°/W$$

Der gesamte Kühlkörper muss demnach 6,6°/W für die drei Transistoren haben.

$R_{THHA3Tges}$ = 6°/W

Da die Transistoren und Spannungsregler auf ein gemeinsames Kühlblech montiert werden, ergibt sich für das gesamte Kühlblech:

Vorgaben:

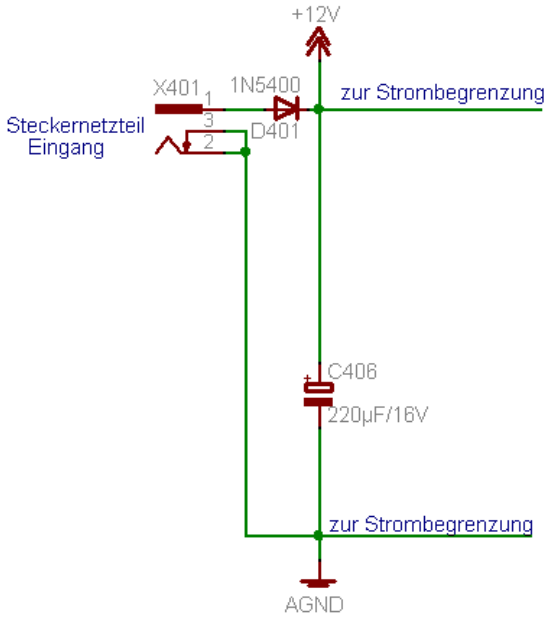
$R_{THHA3SRges} = 10°/W \ldots benötigtes\ Kühlblech\ der\ Spannungsregler$

$R_{THHA3Tges} = 6°/W \ldots benötigtes\ Kühlblech\ der\ Transistoren$

$$R_{THHAges} = \frac{R_{THHA3SRges} \cdot R_{THHA3Tges}}{R_{THHA3SRges} + R_{THHA3Tges}} = \frac{10°/W \cdot 6°/W}{10°/W + 6°/W} = 3{,}75°/W$$

$R_{THHAges}$ = 3,75°/W

Eingangsbeschaltung



X401 ist eine Niedervoltbuchse zur Verbindung mit einem Standard 12V Steckernetzteil.
Die Diode D401 dient zum Verpolungsschutz. Es wurde die Type 1N5400 gewählt, da sie einen Strom von mindestens 1,8A standhält.

D401= 1N5400 50V / 1A

Die Kondensatoren C406, C407 und C413 dienen zur Stabilisierung der Eingangsspannung und wurden mit 220µF / 16V gewählt.

C406= C407= C413= 220µF / 16V

5.2.5. Externe Komponenten

5.2.5.1. Steckernetzteil

Es wurde beschlossen ein externes Schaltnetzteil zur Versorgung des Netzteils zuzukaufen. Die Wahl fiel dabei auf folgendes Modell:

Eingang: 100..240 V~
Ausgang:12 V-/3,33 A.
Eingangsseitig Kaltgerätestecker,
Ausgangsseitiges Kabel 1,5 m mit
Hohlstecker 5,5/2,5 mm. Pluspol innen

Lieferant: **Neuhold Elektronik**
Bestellnummer: **N0130**

Preis: **4,95€**

6. Hintergründe (theoretische Grundlagen)

6.1. RS232

6.1.1. Allgemeines

RS232 ist eine serielle Schnittstelle, welche auch als EIA-232 bezeichnet wird. Sie wird verwendet um serielle Daten über längere Strecken zu übertragen.

Zum Übertragen der Daten wird ein asynchrones serielles Verfahren genutzt.
Das heißt in der einfachsten Ausführung kann eine Kommunikation über zwei Leitungen realisiert werden.

Der RS232 - Standard sieht zur Kommunikation zwei Pegel vor:

- -12V => Logisch 1
- +12V => Logisch 0

Diese Spannungspegel sind aber nicht zu den heute verwendeten Digitalbausteinen kompatibel.
Um dieses Problem zu lösen gibt es zum Beispiel den Pegelwandler IC MAX232.
Dieser liefert dann folgende Pegel:

- 0V => Logisch 1
- 5V => Logisch 0

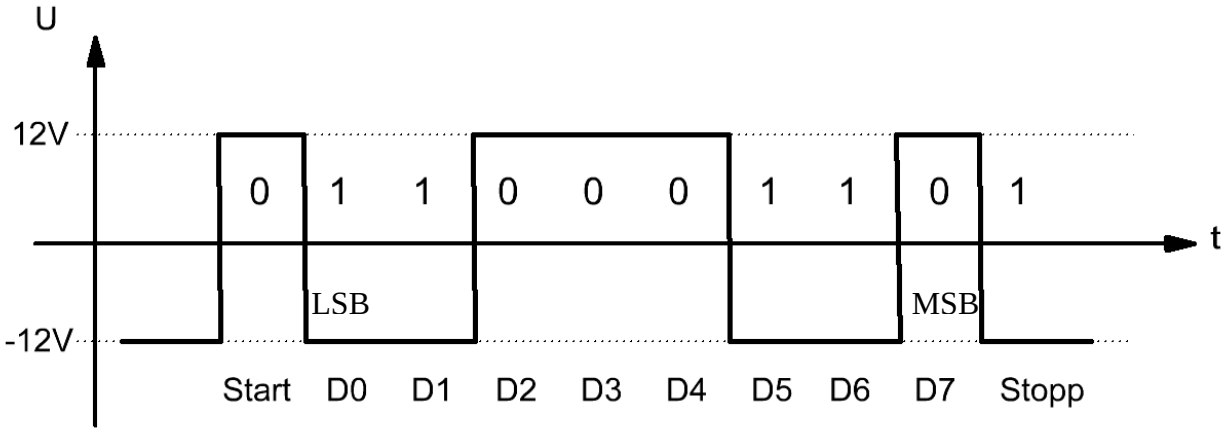
6.1.2. Datenübertragung

Um Daten zu übertragen muss zuerst eine Datenrate (Baud) festgelegt werden.
Diese muss für den Empfänger und den Sender gleich sein, damit eine Kommunikation stattfinden kann.
Folgende Baudraten sind genormt:

Baudrate	Bitlänge	max. Kabellänge
50	20ms	>900m
300	3,33ms	>900m
2400	417ms	900m
9600	52,08µs	152m
19200	52,08µs	15m
38400	26,04µs	<15m
57600	17,36µs	5m
115200	8,68µs	<2m
230400	4,34µs	<2m
460800	2,17µs	<2m

Es ist sehr wichtig diese Zeiten genau einzuhalten, da es sonst zu Kommunikationsfehlern kommen kann.

Die eigentliche Kommunikation sieht wie folgt aus:



Die Bits habe folgende Bedeutung:

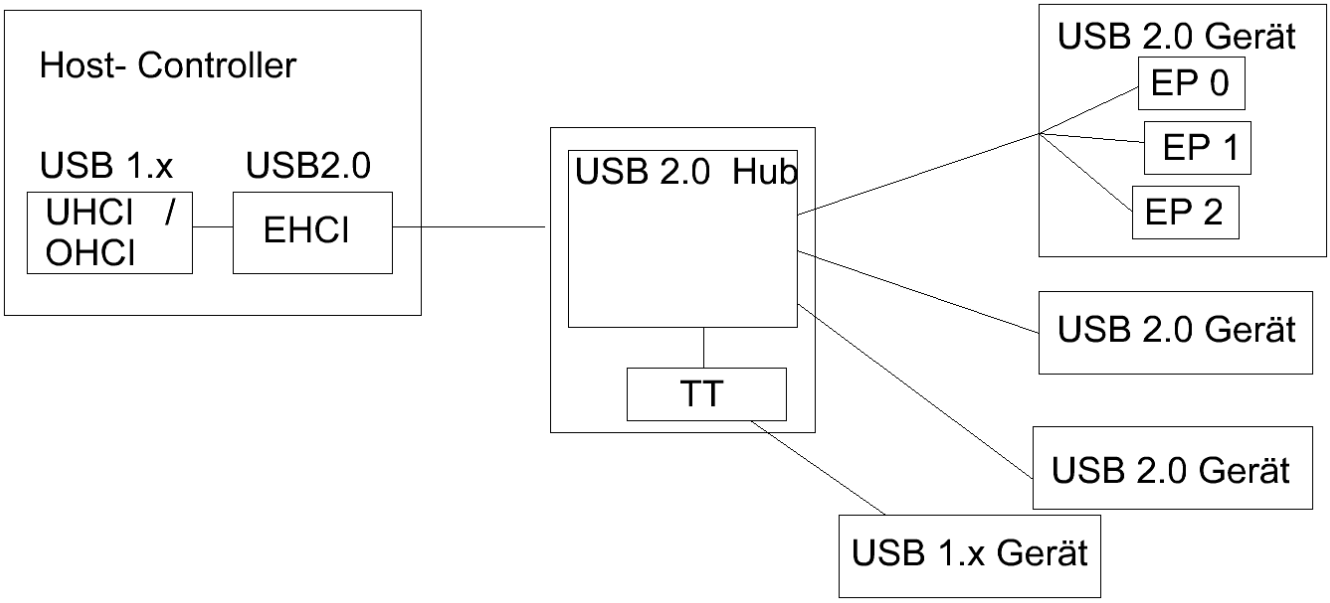
- **Startbit**
  Das Startbit dient zur Synchronisation und teilt dem Empfänger mit, dass jetzt Daten gesendet werden.
- **Bit D0 bis D7**
  Diese Bits beinhalten die Daten. D0 ist das LSB und D7 das MSB.
- **Stoppbit**
  Dient als Wartezeit bis das nächste Bit gesendet wird. Es sind auch zwei Stoppbits möglich.

6.2. USB

6.2.1. Allgemeines

Der Universal Serial Bus (USB) ist ein serielles Bussystem zur Verbindung eines Hosts mit einem Gerät. Eigentlich ist der Begriff Bus nicht richtig, da sich mit USB nur Punkt zu Punkt Verbindung aufbauen lassen.

Prinzipieller Aufbau eines USB- Netzwerks:



6.2.2. Host- Controller

In jedem USB Netzwerk muss ein Host- Controller existieren, ausgenommen davon sind USB On-the-go Geräte. Je nachdem ob der Host ein USB 1.X oder USB 2.0 Controller ist, sind folgende Komponenten enthalten:

- Ein Universal Host Controller Interface (UHCI) oder ein Open Host Controller Interface je nachdem welcher Hersteller den Controller gebaut.
- Bei USB 2.0 ist noch zusätzlich ein Enhanced Host Controller Interface (EHCI) eingebaut, welches die Hi- Speed Kommunikation übernimmt.

6.2.3. USB Hubs

Mit Hilfe von Hubs ist es möglich mehrere Geräte an einen Bus anzuschließen.
Wenn an einem USB 2.0 Hub ein USB 1.X Gerät angeschlossen wird, muss jedoch ein Transaction Translators (TT) Controller verwendet werden um die USB 1.X Pakete auf USB 2.0 Pakete umzuwandeln.

Es können maximal 5 Hubs hintereinander geschalten werden, ohne dass sich die Laufzeitenverzögerung auswirkt. Im gesamten Netzwerk dürfen nur maximal 127 Geräte verwendet werden.

6.2.4. USB Geräte

Jedes USB 2.0 Gerät besitzt wieder maximal 31 Endpunkte. USB 1.X Geräte haben nur maximal 3 Endpunkte. Die Endpunkte sind von 0 weg durchnummeriert.

In jedem USB-Gerät muss ein Endpunkt mit Adresse 0 vorhanden sein, über den die Erkennung und Konfiguration des Gerätes läuft, darüber hinaus kann er auch noch weitere Funktionen übernehmen.

Jeder dieser Endpunkte, mit ein paar Ausnahmen, kann eine der folgenden Übertragungsmodi verwenden:

- **Control-Transfer**
  Der Control-Transfer ist eine besondere Art der Datenübertragung. Diese wird auch vom Endpunkt mit der Nummer 0 verwendet.
  Die Kommunikation ist in beide Richtungen möglich. Dies ist bei den anderen Modi nicht der Fall. Dies ist sehr wichtig zum Austausch der ersten Kommunikation.
- **Isochroner Transfer**
  Dieser Transfermodus wird verwendet, wenn fixe Datenraten übertragen werden. Zum Beispiel bei Audio Karten oder ähnlichem.
  Der Transfer ist aber nicht gesichert und es kann zu Paketausfällen kommen.
- **Interrupt-Transfer**
  Bei diesem Transfermodus fragt der Host- Controller zyklisch ab ob neue Daten vorhanden sind. Dies wird zum Beispiel bei der USB Maus verwendet.
- **Bulk-Transfer**
  Dieser Modus ist für große nicht zeitkritische Datenmengen geeignet. Er wird dann ausgeführt wenn alle anderen Transfers abgeschlossen sind. Dies wird zum Beispiel bei externen Speichermedien verwendet.

6.3. Prinzipien A/D – Wandler

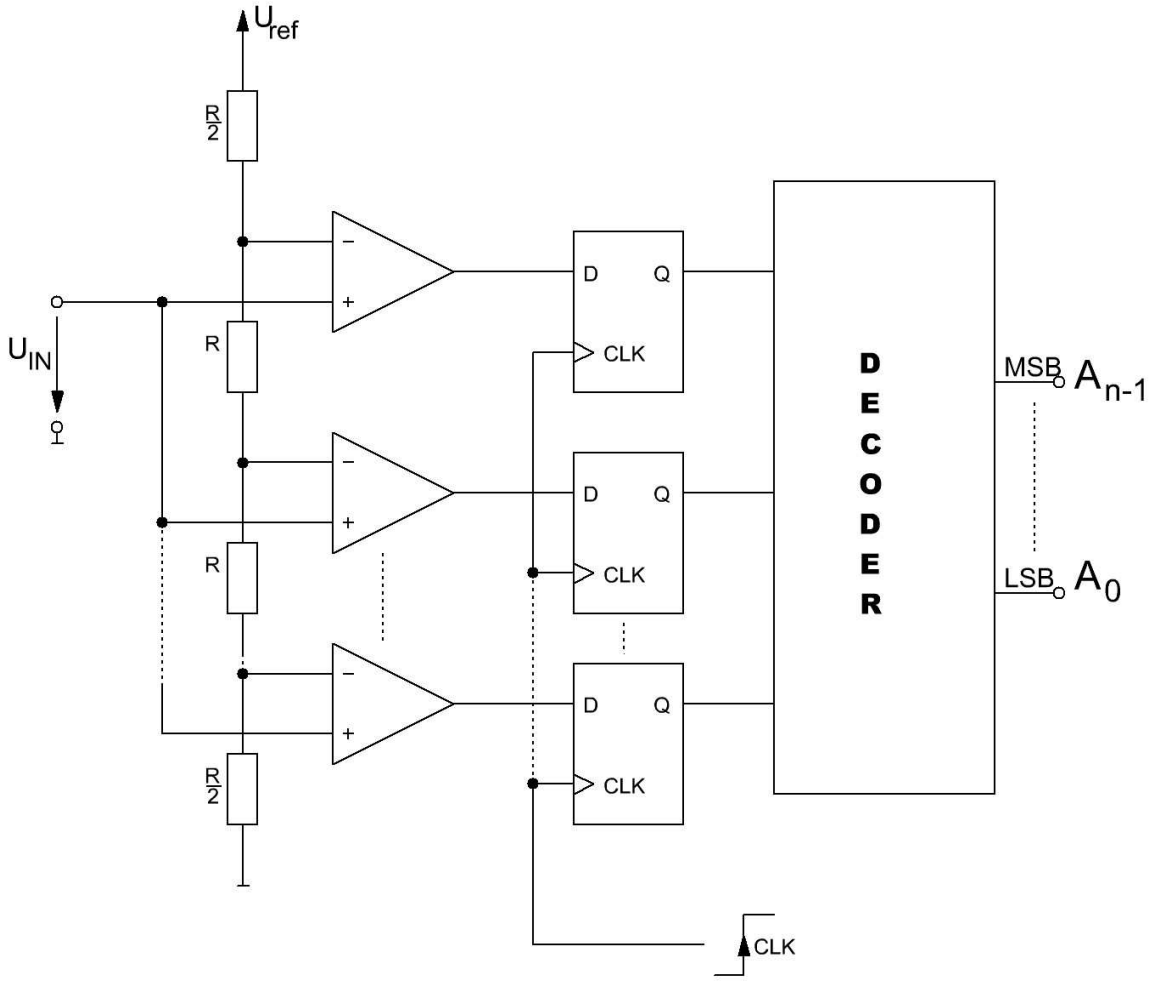
6.3.1. Allgemeines

Ein A/D – Wandler kommt immer dort zum Einsatz, wo physikalische Größen in Form einer analogen Spannung (z.B. Sensor- oder Audiosignale) vorliegen und digital weiterverarbeitet werden sollen.

6.3.2. Parallelverfahren

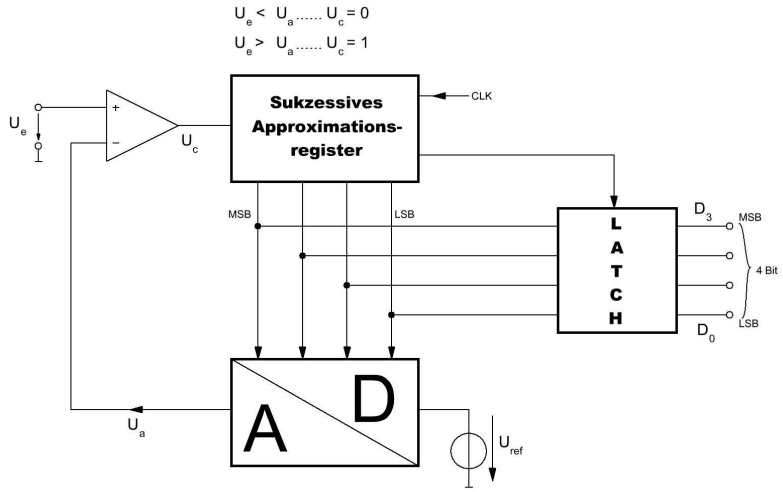
Das Parallelverfahren ist das schnellste Verfahren. Die Wandlung des Analogwertes erfolgt blitzartig. Deshalb werden diese Wandler auch als „Flash Converter“ bezeichnet.
Dieses Verfahren ist sehr aufwändig zu realisieren, da N-1 Komponenten (Komparatoren, D – FlipFlop und Referenzspannungen) notwendig sind. Deshalb sind damit nur kleine Wortbreiten von bis zu 10bit möglich.

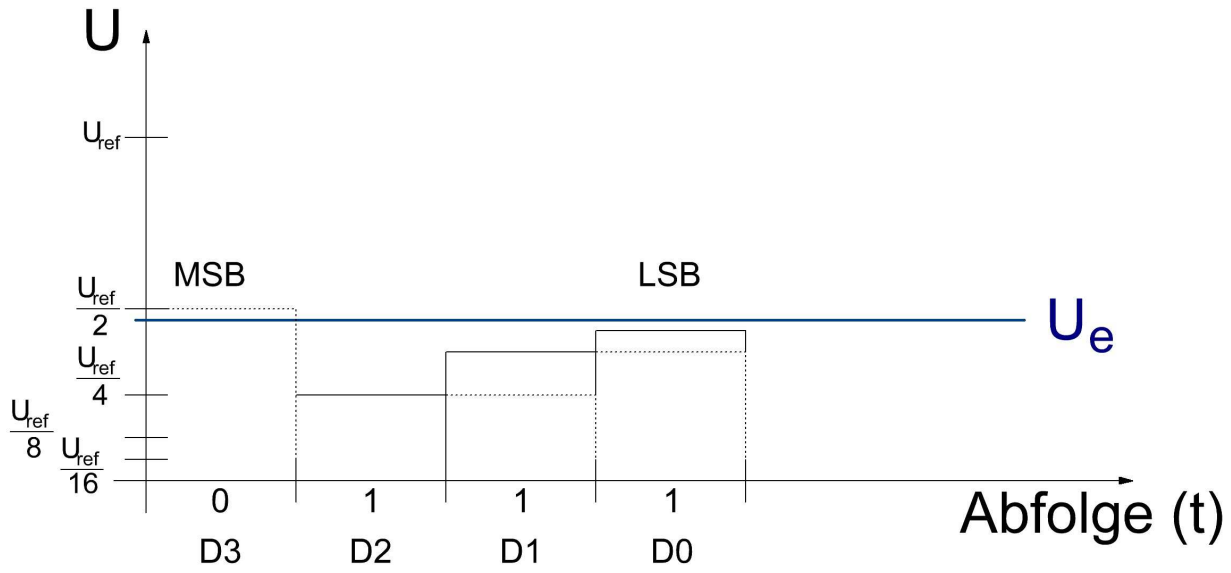
6.3.2.1. Flash Converter



6.3.3. Wägeverfahren

Das Wägeverfahren ermöglicht Auflösungen von 8 bis 18 Bit und arbeitet im Frequenzbereich von wenigen kHz bis zu 1MHz.
Beim Wägeverfahren wird immer ein Bit nach dem anderen mit der Eingangsspannung verglichen. Zunächst wird das MSB auf 1 geschaltet und mit der Eingangsspannung verglichen. Wenn die Eingangsspannung größer ist als das MSB bleibt das Bit auf 1 gesetzt, ist sie kleiner wird das Bit wieder auf 0 gesetzt. Dasselbe Verfahren wird auch bei den nachfolgenden Bits angewendet bis zum LSB.
Deshalb werden für die Wandlung W – Schritte benötigt.
Sobald die Wandlung abgeschlossen ist, wird das Ergebnis des sukzessiven Approximationsregisters an das Latch übergeben und steht an dessen Ausgang dann zur Verfügung.
Während der Wandlung des Eingangssignals ist es wichtig, dass das Eingangssignal konstant gehalten wird. Hierfür wird eine Sample & Hold – Schaltung am Eingang benötigt.



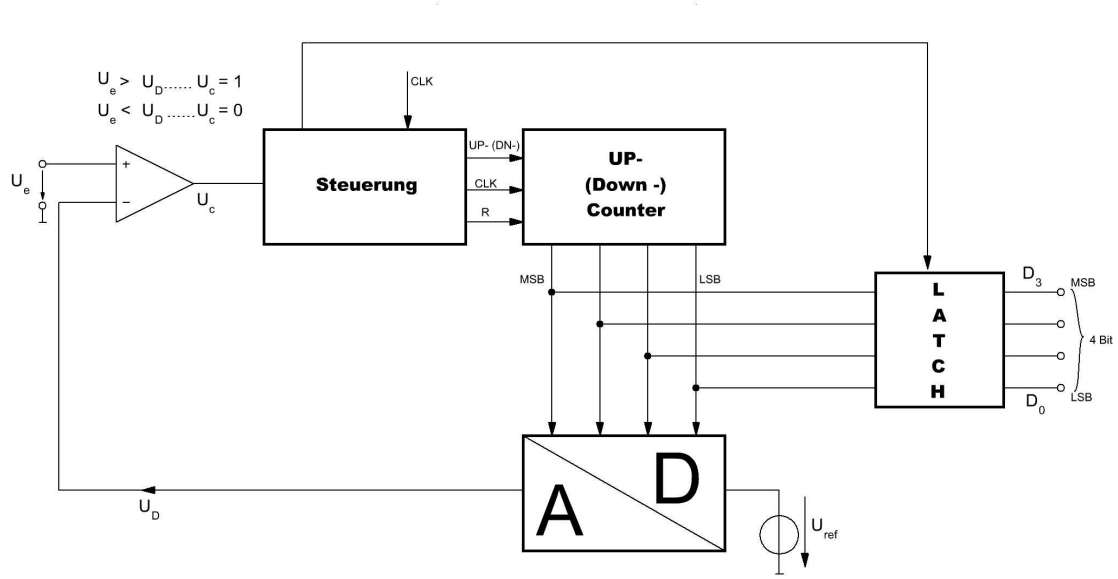


6.3.4. Zählverfahren

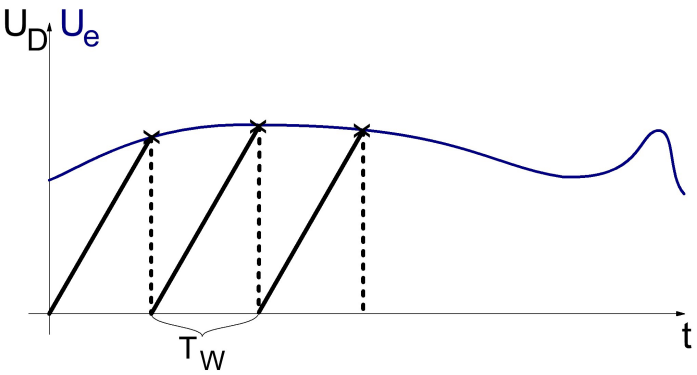
Das Zählverfahren ermöglicht Auflösungen von 10 bis 24 Bit und arbeitet im Frequenzbereich von wenigen Hz bis zu wenigen kHz.
Beim Zählverfahren gibt es zwei unterschiedliche Methoden.
Es gibt einerseits das Kompensationsverfahren und andererseits das Rampenverfahren.

6.3.4.1. Kompensationsverfahren

Beim Kompensationsvefahren wird der digitale Wert eines Counters mittels D/A – Wandler auf eine analoge Spannung umgesetzt. Dieser Wert wird dann mit der Eingangsspannung verglichen.



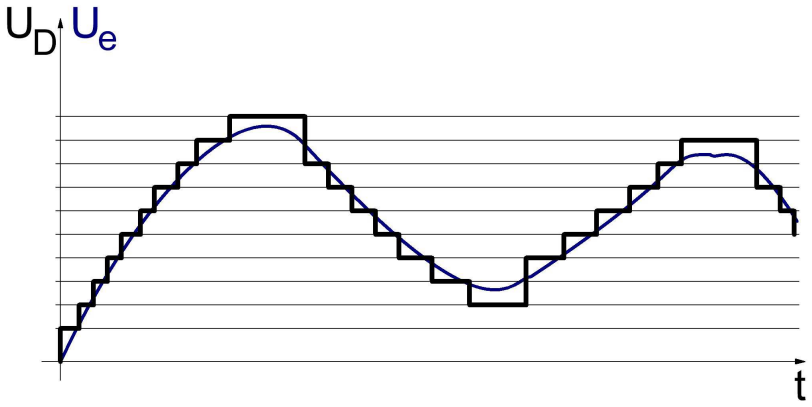
Beim Kompensationsverfahren gibt es zwei Strategien.
Es gibt die Strategie in der nur mit einem Up – Counter gearbeitet wird. Dieser wird nach jeder erfolgreichen Konvertierung wieder auf 0 gesetzt. Danach beginnt der Counter wieder hinaufzuzählen bis er den neuen Wert gewandelt hat. Dies hat bei Signalen die sich nur sehr langsam ändern den Nachteil, dass der Counter immer wieder von 0 zu zählen beginnt und somit unnötig lange für die Konvertierung braucht.



Man sieht, dass bei dieser Variante die Wandlungszeit sehr lange dauert.
Maximale Wandlungszeit:

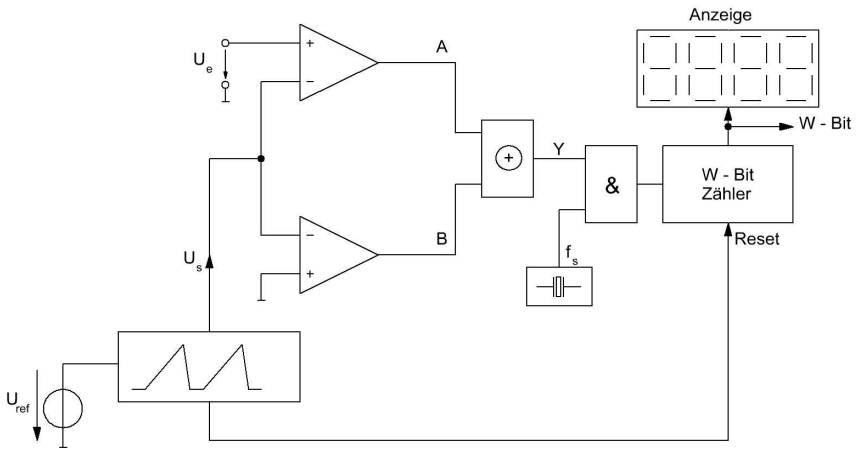
$$T_W = \left(2^n - 1\right) \cdot \frac{1}{f_{CLK}}$$

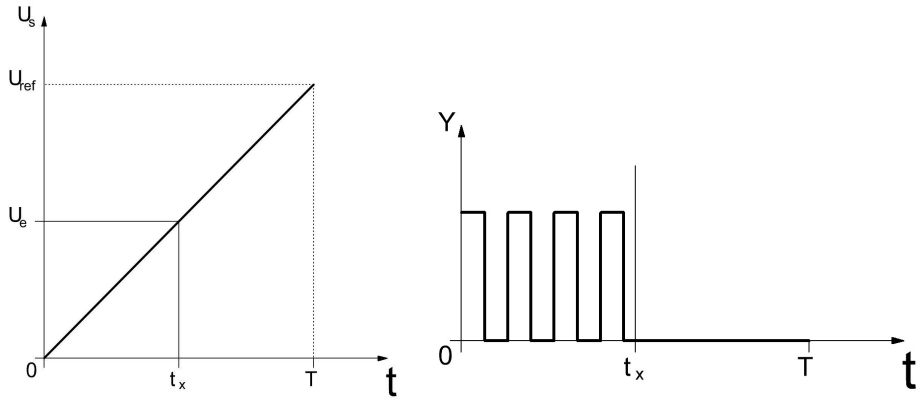
Die Variante mit dem Up - und Down – Counter ist bei sich langsam verändernden Signalen wesentlich effizienter, da der Zähler nach der letzten Wandlung entweder hinauf oder hinunter zu zählen beginnt. Dadurch wird die Wandelzeit bei sich langsam veränderbaren Signalen wesentlich verringert.
Dieser Wandler wird auch Nachlaufwandler bezeichnet.



6.3.4.2. Einzelrampenumsetzer

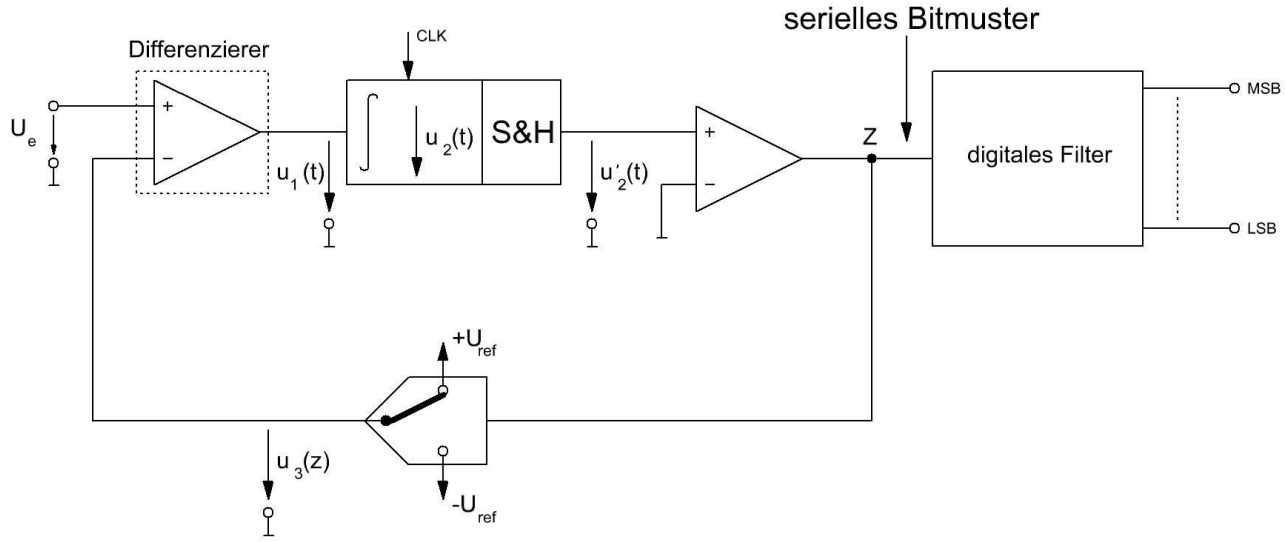
Der Einzelrampenumsetzer gehört zur zweiten Gruppe der Zählverfahren. Es ist ein indirektes Verfahren, weil das Eingangssignal zunächst in eine andere Größe (Frequenz, Impulse) umgewandelt. Dies geschieht folgendermaßen:
Zunächst wird die Eingangsspannung mit einer Sägezahnspannung verglichen. Dies ergibt am Komparatorausgang solange logisch 1 bis die Eingangsspannung gleich der Sägezahnspannung ist.
Sobald dies eintrifft, wird der Komparatorausgang logisch 0. Das nachfolgende XOR – Gatter sorgt dafür, dass der Komparatorausgang nur dann durchgeschaltet wird, wenn eine Sägezahnspannung aktiv ist.
Das Ausgangssignal das XOR – Gatter wird dann an ein getaktetes UND – Gatter geliefert was dafür sorgt, dass solange das Ausgangssignal des Komparators logisch 1 und eine Sägezahnspannung aktiv ist eine Impulsfolge entsteht. Diese Impulse werden dann in einem Bit – Zähler gezählt. Aus dem Verhältnis Anzahl der Impulse zur Periodendauer lässt sich die Eingangsspannung bestimmen. Das Ergebnis des Bit – Zählers ist gleich dem digitalen Ausgangswert.
Nachteile dieses Verfahrens sind die lange Umsetzzeit und die Abhängigkeit der Genauigkeit vom Taktsignal f_s und dem Integrator.

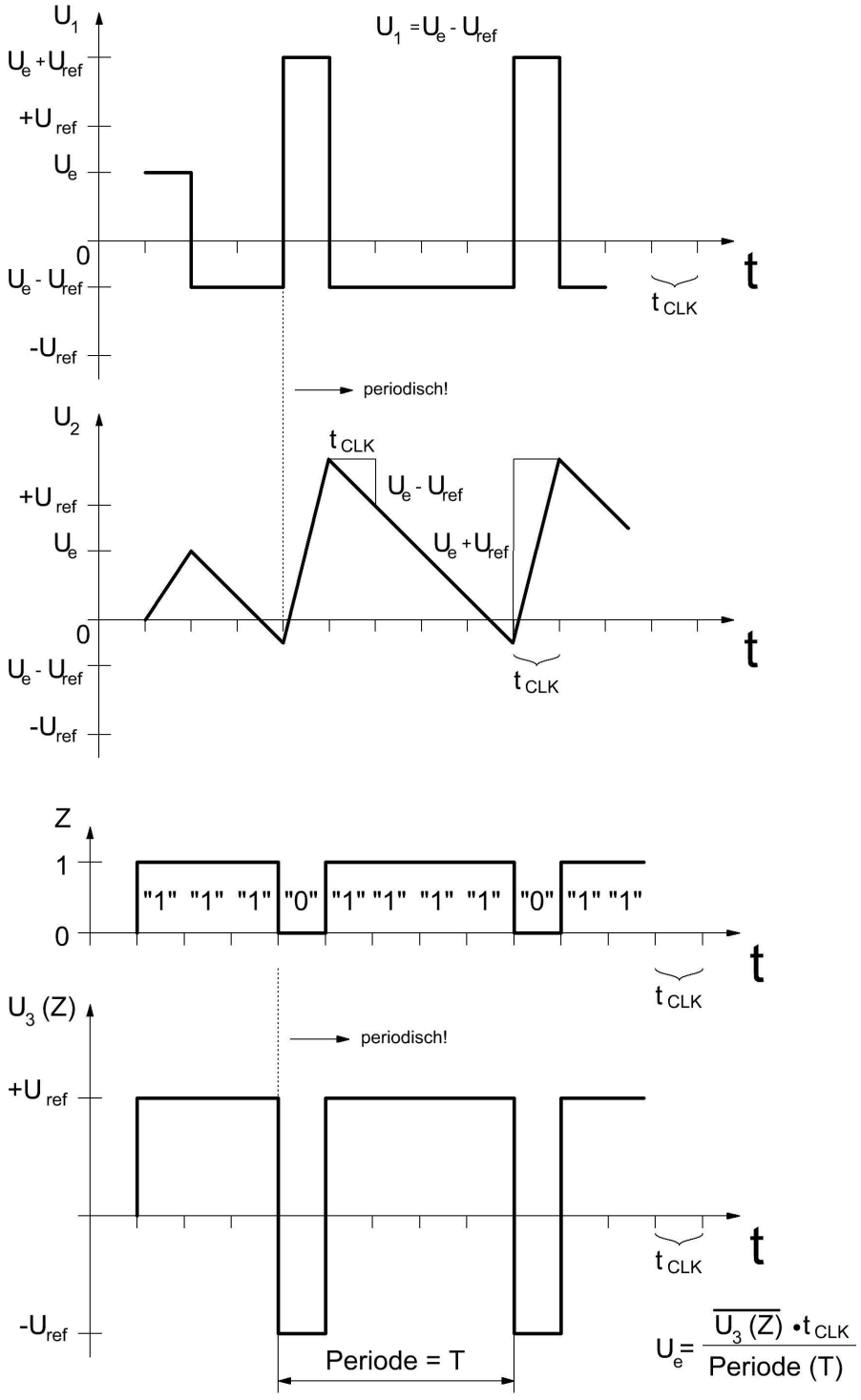




6.3.5. Delta – Sigma

Das Delta – Sigma – Prinzip ist das modernste Prinzip und ist auf Grund des enormen schaltungstechnischen Aufwandes nur in hochintegrierten Schaltungen realisierbar.
Die großen Vorteile dieses Verfahrens bestehen darin, dass kein Sample & Hold benötigt wird, eine hohe Auflösung von 12 – 22 Bit möglich ist und Abtastfrequenzen von bis zu 1MHz realisierbar sind.
Das Prinzip beruht darauf, dass das Eingangssignal mit sehr hohen Taktraten abgetastet wird. Dadurch entsteht ein Oversampling. Das dadurch entstehende serielle Bitmuster wird auf Periodizität untersucht. Aus den periodischen Bitmustern kann dann auf die Eingangsspannung geschlossen werden.





Der Differenzierer am Eingang liefert die Zeitkonstante des Integrators.

Die Anstiegszeitkonstante des Integrators ist im eingeschwungenen Zustand $\tau_{an} = \frac{U_{ref} + U_e}{t_{CLK}}$ und die Abfallskonstante ist $\tau_{ab} = \frac{U_e - U_{ref}}{t_{CLK}}$.

Im eingeschwungen Zustand entsteht somit ein periodische Signal aus dem die Eingangsspannung berechnet werden kann.

6.4. Schaltnetzteil Typen

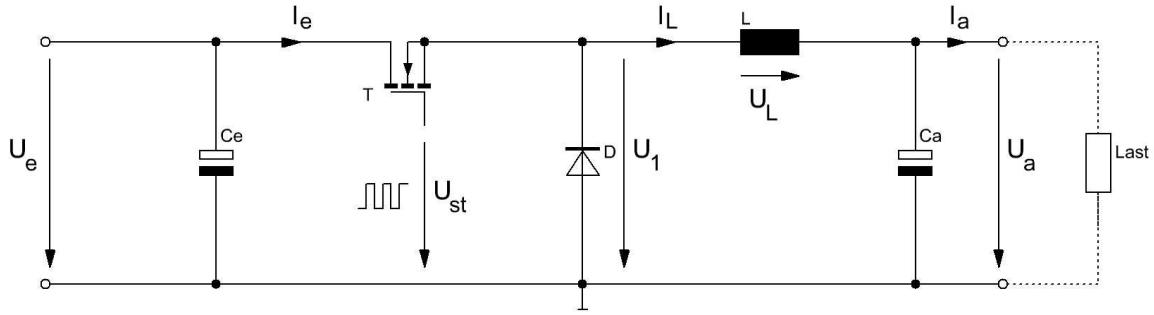
Es wurde für sämtliche Erklärungen der einzelnen Schaltnetzteil Typen der Spannungsabfall der Dioden und Transistoren nicht berücksichtigt.
Weiters gelten sämtliche Formeln nur für den kontinuierlichen Betrieb.

6.4.1. Abwärtswandler

6.4.1.1. Allgemeines

Der Abwärtswandler dient zum Wandeln einer Eingangsspannung in eine niedrigere Ausgangsspannung. Eine andere Bezeichnung für den Abwärtswandler ist auch Tiefsetzsteller. Ein Transistor wird als Schalter verwendet. Dieser „zerhackt" die Eingangsspannung um somit eine niedrigere Ausgangsspannung zu erzeugen.

6.4.1.2. Schaltung



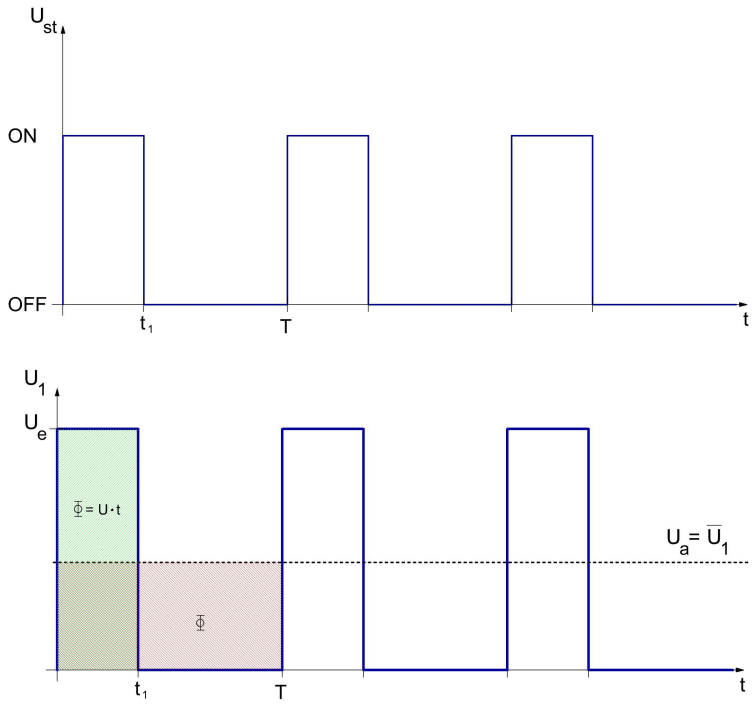
6.4.1.3. Beschreibung

Der als Schalter arbeitende Transistor T wird mit einer pulsweitenmodulierten Steuerspannung U_{st} angesteuert. Dadurch wird die Eingangsspannung U_e zerhackt.

Die Ausgangsspannung U_a ist maßgeblich durch das Tastverhältnis $\frac{t_1}{T}\left(=\frac{Abschaltzeitpunkt}{Periodendauer}\right)$ der Steuerspannung bestimmt.
Der Tiefpass am Ausgang der Schaltung, bestehend aus dem Kondensator C_a und der Spule L, dient zur Glättung der Spannung U_1. Somit gilt $U_a = \overline{U_1}$.

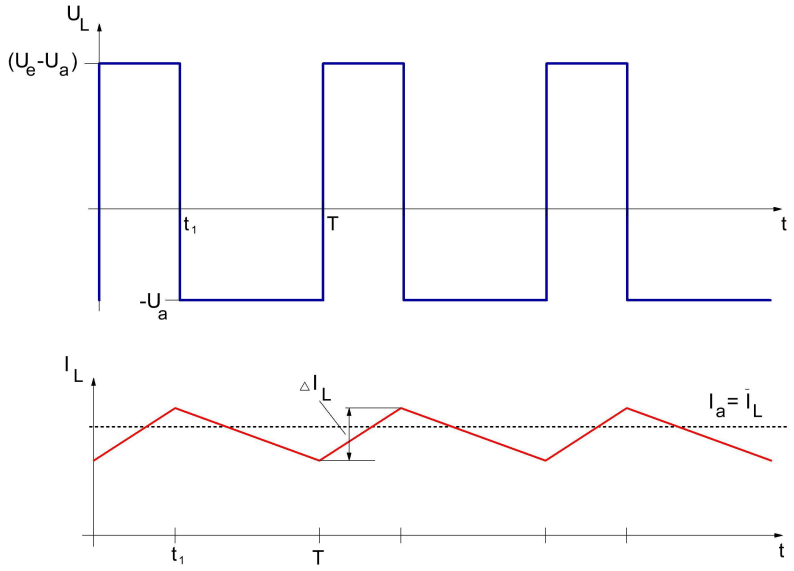
6.4.1.4. Strom - und Spannungsverläufe

Wenn der Transistor T voll durchgesteuert wird (Schalter geschlossen) liegt die Eingangsspannung U_e direkt and der Spule L und dem Kondensator C_a an (Spannung U_1).
Sobald der Transistor T beginnt zu sperren (Schalter geschlossen) versucht die Spule den Strom weiter zu treiben. Die Spule spielt nun Quelle. An der Spule liegt zu diesem Zeitpunkt die negative Ausgangsspannung an.



$$U_a = U_e \cdot \frac{t_1}{T}$$

Man sieht, dass die Ausgangsspannung im kontinuierlichen Betrieb nur vom Tastverhältnis und der Eingangsspannung abhängig ist. Die Last hat auf die Ausgangsspannung keinen Einfluss.



Der Stromfluss der Spule hat einen dreieckförmigen Verlauf, da die Spule bei durchgeschaltetem Transistor den Strom integriert und bei gesperrtem Transistor die Energie wieder abgibt. Der Mittelwert des Stromes wird maßgeblich von der Last bestimmt.
Die Welligkeit des Stromes ΔI_L wird von der Größe der Spule bestimmt.

$$u = L \cdot \frac{di}{dt} \rightarrow \Delta i = \frac{1}{L} \cdot u \cdot \Delta t \rightarrow \Delta I_L = \frac{1}{L} \cdot U_a \cdot (T - t_1) = \frac{1}{L} \cdot (U_e - U_a) \cdot t_1$$

setze für $t_1 = \frac{U_a}{U_e} \cdot \frac{1}{f}$ ein:

$$\Delta I_L = \frac{1}{L} \cdot (U_e - U_a) \cdot \frac{U_a}{U_e} \cdot \frac{1}{f}$$

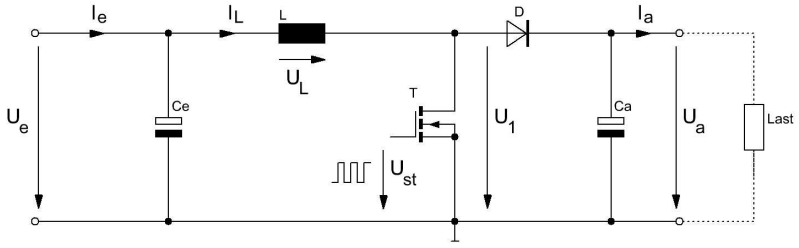
Vom lückenden Betrieb oder diskontinuierlichen Betrieb spricht man sobald der Laststrom $I_a \leq \frac{\Delta I_L}{2}$ geworden ist. In diesem Fall wird der Spulenstrom zu jeder Periode zu Null.

6.4.2. Aufwärtswandler

6.4.2.1. Allgemeines

Der Aufwärtswandler setzt eine Eingangsspannung in eine höhere Ausgangsspannung um. Dieser Wandlertyp wird auch Hochsetzsteller genannt.
Er hat ebenso wie der Abwärtswandler einen Transistor T der als Schalter fungiert.

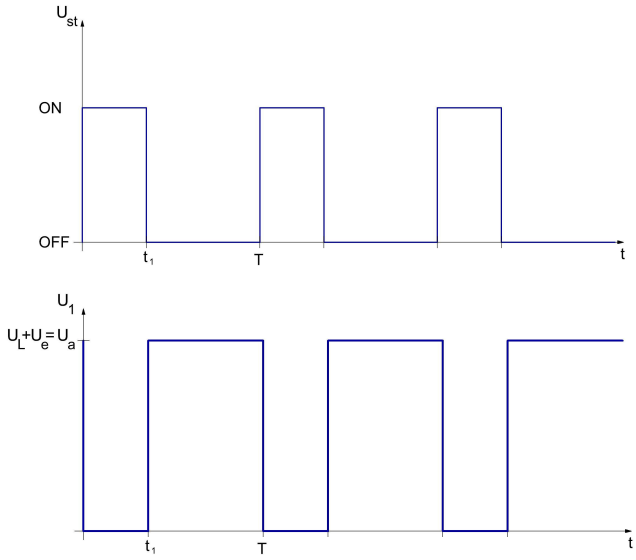
6.4.2.2. Schaltung

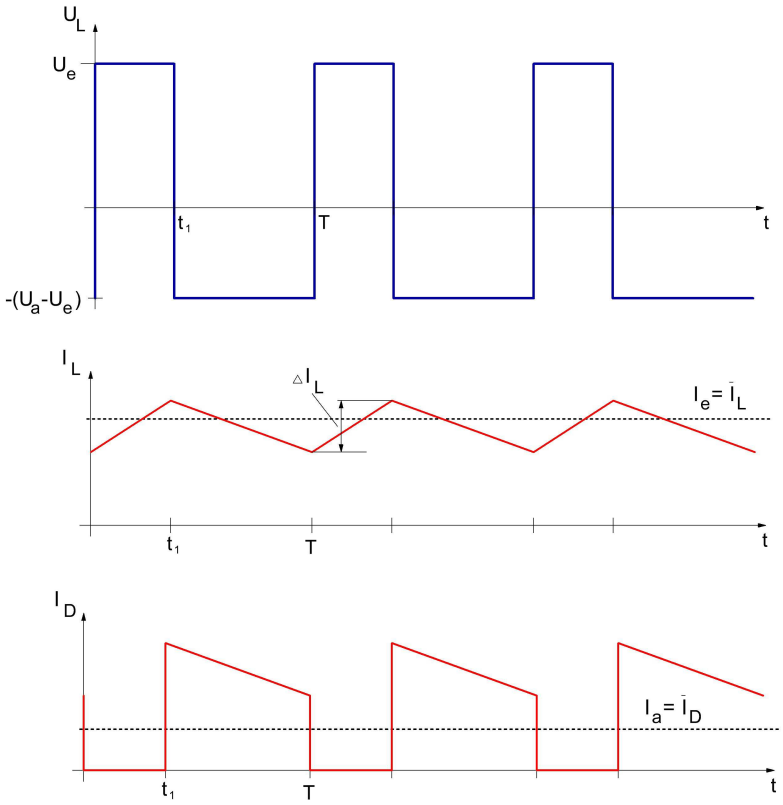


6.4.2.3. Beschreibung

Solange der Transistor T durchgesteuert ist, liegt an der Spule L die gesamte Eingangsspannung U_e an. Sobald der Transistor T wieder sperrt, treibt die Spule den Strom I_L über die Diode D weiter. Dadurch wird der Ausgangskondensator geladen und die Ausgangsspannung steigt an. Wenn der Transistor ständig sperren würde, wäre die Ausgangsspannung gleich der Eingangsspannung.
Man sieht hier auch, dass diese Schaltung nicht kurzschlusssicher ist, da kein strombegrenzendes Bauelement vorhanden ist. Der Kurzschlusspfad wäre über L und D gegeben.
Weiters ist diese Schaltung im nicht geregelten Betrieb auch nicht leerlauffest, da die Ausgangsspannung des Kondensators immer weiter ansteigen würde bis Bauelemente zerstört werden würden.

6.4.2.4. Strom - und Spannungsverläufe





Durch das Induktionsgesetz ergibt sich aus der Welligkeit des Stromes für die Ausgangsspannung:

$$u = L \cdot \frac{di}{dt} \rightarrow \Delta i = \frac{1}{L} \cdot u \cdot \Delta t \rightarrow \Delta I_L = \frac{1}{L} \cdot U_e \cdot t_1 = \frac{1}{L} \cdot (U_a - U_e) \cdot (T - t_1)$$

$$\frac{1}{L} \cdot U_e \cdot t_1 = \frac{1}{L} \cdot (U_a - U_e) \cdot (T - t_1)$$

$$\frac{U_a - U_e}{U_e} = \frac{t_1}{T - t_1}$$

$$\Rightarrow U_a = U_e \cdot \frac{T}{T - t_1}$$

Man sieht, dass die Ausgangsspannung hier nur vom Tastverhältnis und der Eingangsspannung abhängig ist.

Der Bauteilwert der Spule lässt sich durch Einsetzen der obigen Beziehung in die Formel $\Delta I_L = \frac{1}{L} \cdot (U_a - U_e) \cdot (T - t_1)$ wie folgt berechnen:

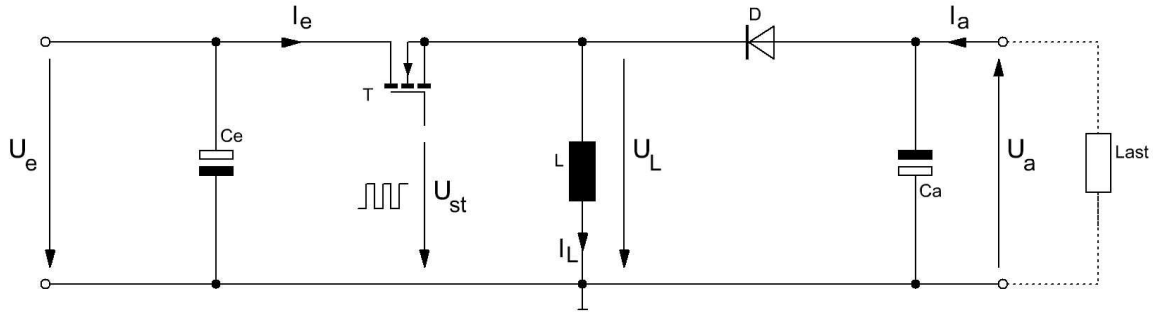
$$L = \frac{1}{\Delta I_L} \cdot (U_a - U_e) \cdot \frac{U_e}{U_a} \cdot \frac{1}{f}$$

6.4.3. Invertierender Wandler

6.4.3.1. Allgemeines

Der invertierende Wandler dient zum Umsetzen einer positiven Spannung in eine negative Spannung.
Dieser Wandler wird auch Hoch – Tiefsetzsteller genannt.

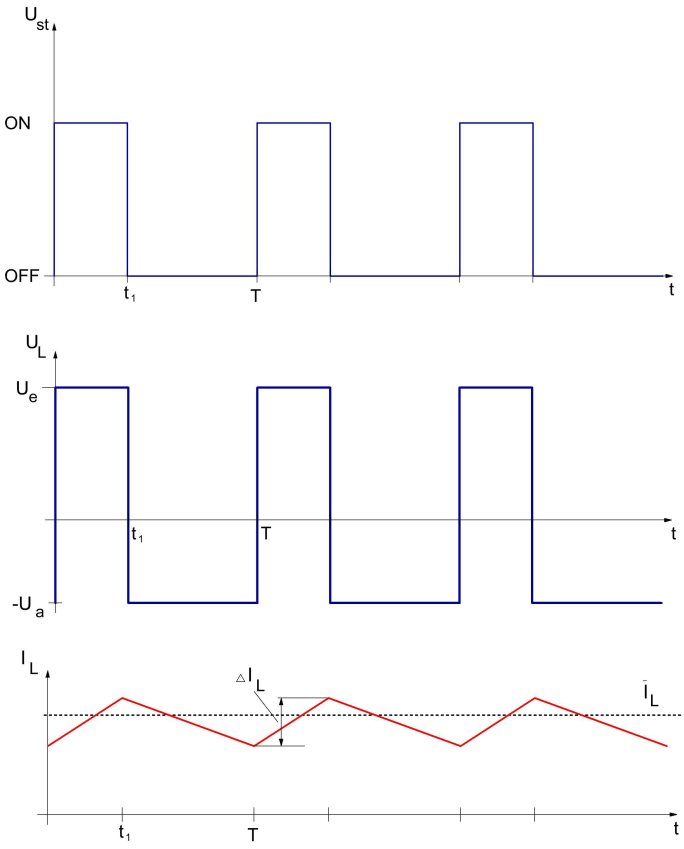
6.4.3.2. Schaltung

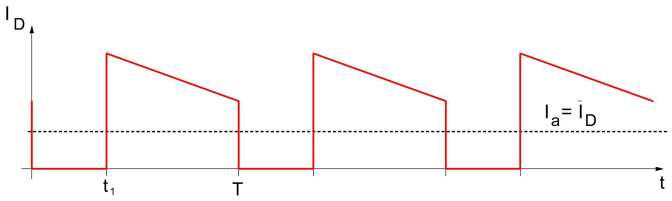


6.4.3.3. Beschreibung

Solange der als Schalter arbeitende Transistor T durchgesteuert wird, liegt die Eingangsspannung an der Spule L an. Sobald der Transistor wieder sperrt, will die Spule den Strom weiter treiben. Der Stromfluss erfolgt über den Kondensator C_a und die Diode D. Dadurch wird der Ausgangskondensator geladen und eine negative Spannung entsteht.

6.4.3.4. Strom - und Spannungsverläufe





Für die Ausgangsspannung ergibt sich:

$$U_a = U_e \cdot \frac{t_1}{T - t_1}$$

Für den Induktivitätsstrom und dessen Welligkeit gilt:

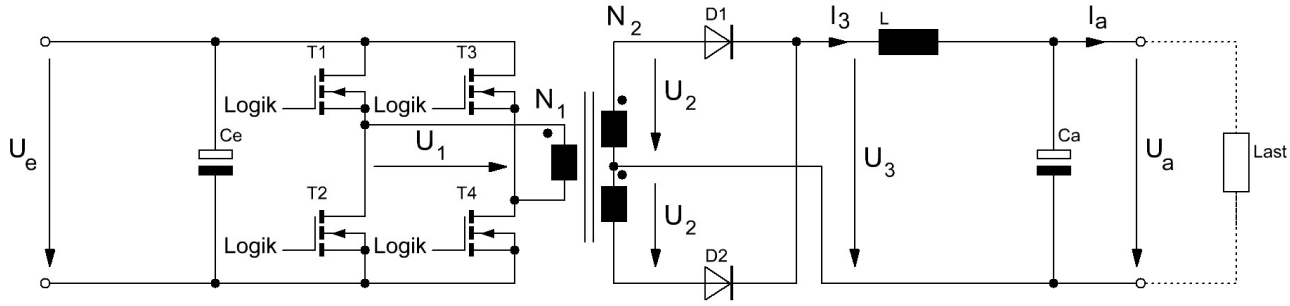
$$\overline{I_L} = I_a \cdot \frac{T}{T - t_1} \quad und \quad \Delta I_L = \frac{1}{L} \cdot U_e \cdot t_1 = \frac{1}{L} \cdot \frac{U_e \cdot U_a}{U_e + U_a} \cdot \frac{1}{f}$$

6.4.4. Prinzip Vollbrücken – Gegentaktwandler

6.4.4.1. Allgemeines

Das Prinzip des Vollbrücken – Gegentaktwandlers findet bei dem Push – Pull – IC MAX256 des Netzteiles Anwendung.
Bei diesem Prinzip kann an einem Transformator einmal die positive und einmal die negative Eingangsspannung angelegt werden. Dadurch entsteht aus einer Gleichspannung eine Wechselspannung mit der ein Transformator versorgt werden kann.

6.4.4.2. Schaltung



6.4.4.3. Eigenschaften

- Galvanisch getrennte, regelbare Ausgangsspannung
- Nur ein kleiner Kern ohne Luftspalt ist zur Energieübertragung notwendig
- Es wird keine besonders gute magnetische Kopplung benötigt

Diese drei Eigenschaften führten dazu, dass wir uns für diesen Wandlertyp entschieden haben.

7. Zusammenfassung

In diesem Projekt wurden drei vollständige Slaves mit Gehäuse und Antenne entwickelt und gebaut.
Ein Slave wurde mit sämtlichen Sensoren bestückt, ein weiterer nur mit dem Gravitationssensor und dem punktuellen Temperatursensor. Der dritte Slave wurde nur mit einem punktuellen Temperatursensor bestückt.
Weiters wurden noch zwei Slave Platinen ohne Sensoren bestückt. Für diese Slaves wurde auch kein Gehäuse gefertigt, da sie nur als Relaisstationen dienen sollen.

Für dieses Projekt wurde außerdem eine voll funktionstüchtige Ladestation zum Laden von bis zu drei Slaves entwickelt und gefertigt.

Weiters wurde die PC Software zur Verwaltung des Messsystems und der Master, welcher als Schnittstelle zwischen Funksystem und PC dient, entwickelt und komplett fertig gestellt.

Das Funksystem und dessen Komponenten wurden erfolgreich getestet.

Das Funksystem blieb von der Datenrate hinter den Erwartungen zurück. Als Grund konnte das Funkmodul lokalisiert werden. Mit einem anderen Funkmodul hätte man die Datenrate deutlich erhöhen können. Jedoch wurde der Fehler erst nach dem Kauf der Funkmodule bemerkt.

8. Conclusion

In this project three complete Slaves have been developed and constructed.
One of the Slaves includes all three sensors.
The second Slave has only two sensors, for surface temperature and gravitation.
The third Slave has only the surface temperature sensor.
For each Slave a aluminium housing was build.

Also a charging station for the Slaves was developed and constructed.

The PC software to manage the measurement system, and the Master unit to connect the PC with the slaves, was designed and programmed.

The whole system was tested successfully.

The data rate of the system wasn’t as high as expected. The reason for the low rate is the transceiver module.
The rate could have been boosted with an other transceiver module, but the error was noticed after the acquisition of the transceiver modules. So it was too late to purchase another one.

9. Literaturliste

[01] Mitschrift aus EDT (Elektronik und Digitaltechnik) Schuljahr 2005/2006 3. Klasse

[02] Mitschrift aus EDT (Elektronik und Digitaltechnik) Schuljahr 2006/2007 4. Klasse

[03] Mitschrift aus EDT (Elektronik und Digitaltechnik) Schuljahr 2007/2008 5. Klasse

[04] Mitschrift aus TKTE (Telekommunikationstechnik) Schuljahr 2006/2007 4. Klasse

[05] Mitschrift aus TKTE (Telekommunikationstechnik) Schuljahr 2007/2008 5. Klasse

[06] Mitschrift aus HF (Hochfrequenztechnik) Schuljahr 2006/2007 4. Klasse

[07] Mitschrift aus HF (Hochfrequenztechnik) Schuljahr 2007/2008 5. Klasse

[08] Taschenbuch der Elektrotechnik (Kories Schmidt – Walter/Harri Deutsch Verlag)

[09] Cadsoft Eagle v4.16r2 für Schaltung und Layoutdesign

[10] AutoDESK AutoCAD 2004 Zeichenprogramm für Konstruktionszeichnungen

[11] sPlan 6.0 Zeichenprogramm für diverse Skizzen

[12] MicroSim PSpice 9.0 für diverse Schaltungsteile

[13] AntCAD zu Antennensimulation

[14] Microsoft Visual Studio.NET 2005

[15] Microchip MPLAB IDE v 7.60

[16] Microchip MCC18 Compiler

[17] MProg 3.0a

Anhang

Anhang A Ausblick

Dieses Projekt könnte um einen weiteren Messtyp (zum Beispiel Helligkeit) erweitert werden.
Weiters könnte eine bessere Antenne für den Slave entwickelt und gebaut werden, um die Reichweite des Systems zu erhöhen.
Außerdem könnte ein anderes Funkmodul verwendet werden um die Geschwindigkeit des Messsystems drastisch zu steigern.
Es könnte in jedem Slave noch ein Speicherbaustein integriert werden, um die Messdaten zwischen zu speichern und im nach hinein bei Bedarf abzuholen.
Weiters könnten noch zahlreiche energietechnische Maßnahmen getroffen werden, um den Energieverbrauch der Slaves zu drosseln und somit die Einsatzzeit zu erhöhen.
Vor allem wenn Slaves nur sehr selten abgefragt werden könnte die Energieaufnahme sehr stark verringert werden.

Anhang B Stundennachweis für Diplomarbeit von Hofer Josef

Datum	Tätigkeit	Stunden
25.08.2007	Projektkoordination und Protokollentwurf	8
05.09.2007	Erstellung eines Zeitplanes	2
12.09.2007	Pflichtenheft und Präsentation	1
19.09.2007	Pflichtenheft und Präsentation überarbeiten, Schaltung für Master	3
24.09.2007	Layoutentwicklung Masterplatine	2
26.09.2007	Testen von Schaltungsteilen, Entwurf der Masterplatinen	2
26.09.2007	Einarbeiten in den PIC18F1320	4
29.09.2007	Einarbeiten in den PIC18F1320	5
30.09.2007	Einarbeiten in den PIC18F1320	5
08.10.2007	Testplatine belichten und ätzen	1,5
10.10.2007	Testplatine bohren und bestücken, PIC Programm starten, Netzteil testen mit MOSFET als Schalter, Testen der induktiven Kopplung	2
17.10.2007	Testplatine bestücken, PIC Programm starten, Netzteil testen mit MOSFET als Schalter	2,5
24.10.2007	Testen des A/D- Wandlers, Testen des Ladereglers und des Aufwärtswandlers, Besprechung des PIC Programmdesigns	1
24.10.2007	Besprechung der Befehle für Master und Slave	1,5
07.11.2007	Fertigstellung des Adressierteils des PIC Slave Programms, Überarbeitung der Befehlscodes für das PIC Programm, Beginn mit der Befehlsauswertungsroutine, Entwurf erster Gehäusekonzepte	2
13.11.2007	PIC Masterprogramm (Kanalbelastungsmessung, Befehlsverarbeitung)	7
14.11.2007	Überarbeitung des Slave und Master Schaltplanes, Layouten der Master und Slave Platinen, Überarbeiten der Stückliste	2
15.11.2007	Layoutentwicklung Slaveplatine	4
15.11.2007	Präsentation der Diplomarbeit für BOSCH	2
26.11.2007	Layoutüberarbeitung des Masters und mechanische Zeichnungen des Master anpassen	4
27.11.2007	Einschubvorrichtung des Slave zeichnen	3
28.11.2007	Ansprechen des Beschleunigungssensors (hat nicht geklappt, da einige Kontakte fehlerhaft waren!), Überarbeiten des Masterlayouts, Beginn der Zeichnungen des mechanischen Gehäuses für Master und Slave, Layouten des Netzteiles und der Ladeschaltung für den Akku	2
03.12.2007	Belichten und ätzen der Testplatine (Akkulader, Master Endversion und Netzteil)	1,5
05.12.2007	Ansprechen des Beschleunigungssensors und Übertragung des ersten Messstreams, Zuschneiden und bestücken der Akkulader- und Netzteiltestplatine, Überarbeitung des Masterlayoutes für die endgültige Version, Zeichnen der mechanischen Pläne für den Master und Slave	1

Datum	Tätigkeit	Stunden
11.12.2007	Punktuelle Temperatursensor Messvorrichtung konstruieren	3
12.12.2007	Zeichnen der mechanischen Pläne für den Master und Slave, Testen des Akkuladers und Netzteils, Überarbeitung des PIC Master und Slave Programms, Performance Tests der Übertragungsrate	1,5
19.12.2007	Zeichnen der mechanischen Pläne für den Slave, Überarbeiten des Slave Eaglelayouts, erstellen einer Stückliste für Master und Slave, suchen eines geeigneten Akkus, Testen der induktiven Kopplung (Netzteil macht Probleme beim treiben des Stromes)	1
03.01.2008	Zeichnen der mechanischen Pläne für den Slaveeinschaub (Seitenansicht), Slave Gehäuse obere Schale	3
04.01.2008	Zeichnen der mechanischen Pläne für den Slaveeinschaub (Seitenansicht), Slave Gehäuse obere Schale	2
05.01.2008	Dokumentation des Master PIC Programms	8
06.01.2008	Dokumentation des Master PIC Programms	3
08.01.2008	Korrektur der Master PIC Dokumentation und Korrektur der mechanischen Pläne des Slave	4,5
09.01.2008	Überarbeiten der mechanischen Pläne für den Slave, Programmentwicklung C# (Protokollierung), Testschaltung für das Schaltnetzteil entwickeln, Testplatine für den Master, Slave und das Netzteil belichten, ätzen und bohren	2
23.01.2008	Heute wurden diverse Tests des Ladereglers durchgeführt, die Spulen dafür gewickelt und mit der Fertigung des Mastergehäuses begonnen	4
30.01.2008	Fertigung des Mastergehäuses, Bohren der Slave-Platine, beginnen zum Bestücken der Slave- Platine, Testen des Slave- Energieverbrauch, Überarbeiten des Netzteilentwurfs (Strombegrenzung und Status LED)	2
04.01.2008	Überarbeiten des Slavegehäuses	5
06.02.2008	Testen der Strombegrenzung für das Netzteil, Dauertest der Ladeschaltung, Fertigstellung des Master- Gehäuses, Bugfxing der PC Software	2
08.02.2008	Auffaltungszeichnung des Slaves zeichnen	4
20.02.2008	Fertigen des Slave- Gehäuses, bestücken und testen des Netzteils und Dauertest des Messsystems	2
21.02.2008	Fertigung der Magneteinschubleisten	5
25.02.2008	Tests zum Biegen des Slave- Gehäuses	2
27.02.2008	Fertigung des Slavegehäuses (Deckel, Bodenplatte und Temperaturabnehmerteil), Layoutüberarbeitung und Test der induktiven Kopplung mit einem Metallring	2
28.02.2008	Bohren der Löcher des Slave- Gehäuses für Taster, LED, Feuchtigkeitskappe und Platinenmontagebohrungen; belichten, ätzen und bohren der neuen Slave- Platine	1

Datum	Tätigkeit	Stunden
28.02.2008	Nochmaliges fertigen der Magneteinschubleisten da bei der ersten Fertigung grobe Fehler die leisten unbrauchbar machten	5
05.03.2008	Fertigen der Slave- Gehäuse	1
06.03.2008	Kleben des Magneten, schreiben des Jugend Innovativ- Berichtes und Fertigbestückung der Slave- Platine	1
12.03.2008	Bestücken und Testen der Netzteil- Platine, kleben der Einschubplatte an das Slave- Gehäuse, fertig stellen des Slave- Gehäuses ausgenommen der Spulenausnehmung, umlöten eines Gravitationssensors und beginn des Bestückens von zwei neuer Slave- Platinen	2
16.03.2008	Netzteilkonstruktionspläne zeichnen	6
17.03.2008	Zeichnen der Netzteil Auffalt- und Konstruktionspläne	8
18.03.2008	Zeichnen der Fixierplatten des Netzteiles	2
20.03.2008	Masterdokumentation überarbeiten	5
21.03.2008	Verbesserungen der Konstruktionen in Pläne übertragen	7
25.03.2008	Verbesserungen der Konstruktionen in Pläne übertragen	8
26.03.2008	Fertigung des Netzteilgehäuses (Deckel und Boden)	2
27.03.2008	Fertigung von zwei Einschubplatten und der Fixierplatten des Netzteils	2
28.03.2008	Einschubplatten bearbeiten (Bohren, Gewindeschneiden, Kantenfeilen)	2
09.04.2008	Fertigen der Leisten für zwei weitere Magnete, kleben der LED's in das Netzteilgehäuse, fertigen der Stecker für das Netzteil, fertigen von drei Antennen, zusammenbauen des Slaves	6,5
16.04.2008	Kleben der Leisten an einen Magneten, kleben von zwei Einschubplatten, Dokumentation der Inbetriebnahme von Master, Slave und Netzteil, Fertigstellung eines neuen Slave - Deckels	2
19.04.2008	Fehlende Bemassungen in Pläne einzeichnen, Pläne aus Autocad extrahieren	7
20.04.2008	Nochmaliges extrahieren der Autocadpläne da die vorherigen für Dokumentation unbrauchbar waren	4
20.04.2008	Dokumentation bearbeiten	7
23.04.2008	Endmontage des Netzteils und der drei Slaves	1
	Anzahl der Stunden Gesamt:	**204,5**

Anhang C Stundennachweis für Diplomarbeit von Knöbel Patrick

Datum	Tätigkeit	Stunden
05.07.2007	Informationen zur induktiven Kopplung suchen	2
15.07.2007	Bauteile suchen und bestellen	8
10.08.2007	Bauteile anlöten	5
12.08.2007	Testen von Schaltungsteilen	8
16.08.2007	Funktests und USB Interface	4
25.08.2007	Projektkoordination und Protokollentwurf	8
05.09.2007	Erstellung eines Zeitplanes	2
07.09.2007	Programmentwicklung C# (Grundgerüst, Sprachdatei)	3
08.09.2007	PC Programm aufbauen	2
09.09.2007	Grundzüge des PC Programms festlegen	10
12.09.2007	Pflichtenheft und Präsentation	1
16.09.2007	PC Programmfehler suchen, Dokumentation	7
17.09.2007	PC Programmentwicklung C# (Beep Klasse)	1
18.09.2007	PC Programmdokumentation und Präsentation	1,5
19.09.2007	Pflichtenheft und Präsentation überarbeiten, Schaltung für den Master entwickeln	3
22.09.2007	PC Programmentwicklung C# (Paint Klasse)	2
23.09.2007	PC Programmentwicklung C# (Paint Klasse)	5,5
23.09.2007	Einarbeiten in den PIC18F1320	1,5
24.09.2007	Einarbeiten in den PIC18F1320, Programmentwicklung C# (Paint Klasse), Programmdokumentation	3
25.09.2007	Programmentwicklung C# (Paint Klasse, Einstellung Klasse), Programmkonzept erweitert, Programmdokumentation	3
26.09.2007	Testen von Schaltungsteilen, Entwurf der Masterplatine	2
26.09.2007	PC Programmentwicklung C# (Variablen Klasse)	1
29.09.2007	PC Programmentwicklung C# (Variablen Klasse)	4
04.10.2007	PC Programmentwicklung C# (Variablen Klasse), Bug in Klasse Einstellungen ausgebessert	0,5
05.10.2007	Protokollierung C# (Variablen Klasse), MainForm erweitert	2
06.10.2007	PC Programmentwicklung C# Funkverwaltung (Status), Variablen Klasse	4
09.10.2007	Softwareprogrammbesprechung, Organisation für den Projekttag	1
09.10.2007	PC Programmentwicklung C# Anzeigefenster und Anzeigen	3,5
10.10.2007	Testplatine bohren und bestücken, PIC Programm starten, Netzteil testen mit MOSFET als Schalter, Testen der induktiven Kopplung	2
10.10.2007	PC Programmentwicklung C# Anzeigefenster und Anzeigen	4
12.10.2007	PC Programmentwicklung C# Anzeigefenster	4,5
13.10.2007	PC Programmentwicklung C# Anzeigefenster	7
14.10.2007	PC Programmentwicklung C# Anzeigefenster, verwalten der Vorlagen in Anzeigen	3
15.10.2007	PC Programmentwicklung C# Paint Klasse auf x-y Darstellung erweitert	4

Datum	Tätigkeit	Stunden
16.10.2007	PC Programmentwicklung C# Variablen Klasse (Kalibration)	3
17.10.2007	Testplatine bestücken, PIC Programm beginnen, Netzteil testen mit MOSFET als Schalter	2,5
19.10.2007	PC Programmentwicklung C# Anzeigefenster Möglichkeit zur Einstellung	0,5
24.10.2007	Testen des A/D- Wandlers, testen des Ladereglers und des Aufwärtswandlers, Besprechung des PIC Programmdesigns	1
24.10.2007	Besprechung der Befehle für Master und Slave	1,5
28.10.2007	Besprechung des Slave PIC Programms	1
29.10.2007	Initialisierung von Timern, Programmdesign Slave	3,5
30.10.2007	PC Programmentwicklung C# Funkverwaltung (Automatische Erkennung des Masters)	6
02.11.2007	PC Programmentwicklung C# Funkverwaltung (Threads zur Funkverarbeitung initialisiert)	7
04.11.2007	Besprechung des Slave PIC Programms und Bugfixing	4
05.11.2007	Schreiben des PIC Programms Adressierteilüberarbeitung	3
06.11.2007	Besprechung der Sensorenansteuerung, Bugfixing des Adressierteils im PIC Programm, testen der Steuerleitung des FT232	4
07.11.2007	Fertigstellung des Adressierteils des PIC Slave Programms, Überarbeitung der Befehlscodes für das PIC Programm, Beginn mit der Befehlsauswertungsroutine, Entwurf erster Gehäusekonzepte	2
09.11.2007	PC Programmentwicklung C# erweitern der Funkklasse	4
10.11.2007	Programmentwicklung PIC (Befehlsverarbeitung fertig gestellt, RealTime Tests, Timeout und Sperrtimer hinzugefügt, zeitverzögertes Abschalten und Kanalwechsel, Bugfixing, Status LED initialisiert, Kommunikationsprotokoll für den SHT11 umgesetzt, erste Messungen vorgenommen)	15,5
12.11.2007	PC Programmentwicklung C# Funkklasse erweitert, Bug im Anzeigesystem ausgebessert	5
13.11.2007	PIC Masterprogramm (Kanalbelastungsmessung, Befehlsverarbeitung)	7
14.11.2007	Überarbeitung des Slave und Master Schaltplanes, layouten der Master und Slave Platinen, überarbeiten der Stückliste	2
14.11.2007	PC Programmentwicklung C# Funkklasse erweitert	2
15.11.2007	Präsentation der Diplomarbeit für BOSCH	2
15.11.2007	PC Programmentwicklung C# Funkklasse erweitert	3
16.11.2007	Testplatine für Master und Slave belichten, ätzen, bohren und bestücken	9
16.11.2007	Testen der Testplatine V2 für Master und Slave	4
17.11.2007	Testen und aufbauen der Slaveplatine	4
21.11.2007	Bugfixing des Slaveprogrammes (Kanalwechsel und Kanalbelastungsmessung), Funkmodul untersuchen für den Kanalwechsel ("ACK"- Befehl untersuchen!)	2

Datum	Tätigkeit	Stunden
22.11.2007	PC Programmentwicklung C# Anzeigeklasse erweitert (Gruppen hinzufügen)	3
25.11.2007	PC Programmentwicklung C# Anzeigeklasse erweitert (Gruppen hinzufügen)	2
28.11.2007	Ansprechen des Beschleunigungssensors (hat nicht geklappt, da einige Kontakte fehlerhaft waren!), überarbeiten des Masterlayouts, Beginn der Zeichnungen des mechanischen Gehäuses für Master und Slave,layouten des Netzteiles und der Ladeschaltung für den Akku	2
25.11.2007	PC Programmentwicklung C# Anzeigeklasse erweitert (Gruppen hinzufügen, testen und fertig stellen)	3
05.12.2007	Ansprechen des Beschleunigungssensors und Übertragung des ersten Messstreams, zuschneiden und bestücken der Akkulader- und Netzteiltestplatine, Überarbeitung des Masterlayoutes für die endgültige Version, zeichnen der mechanischen Pläne für den Master und Slave	2
07.12.2007	Optimieren des PIC Slave Programms (Temperatursensor ADT7301 ansprechen)	6
12.12.2007	Zeichnen der mechanischen Pläne für den Master und Slave, testen des Akkuladers und Netzteils, Überarbeitung des PIC Master und Slave Programms, Performance Tests der Übertragungsrate	1,5
19.12.2007	Zeichnen der mechanischen Pläne für den Slave, überarbeiten des Slave Eaglelayouts, erstellen einer Stückliste für Master und Slave, suchen eines geeigneten Akkus, Testen der induktiven Kopplung (Netzteil macht Probleme beim treiben des Stromes)	2
31.12.2007	Analysieren der Funkstrecke (Laufzeitmessung)	1
02.01.2008	PIC Slave Programm Bugfixing (Kanalwechsel und Abschalten) und Spitzenwertberechung für den Beschleunigungssensor	3
03.01.2008	PIC Slave Programm Bugfixing (Kanalwechsel und Abschalten), PC Programmentwicklung C# (Entwicklung der Funkalgorithmen), Dokumentationsbesprechung	6
03.01.2008	PC Programmentwicklung C# Funkklasse erweitert	3
04.01.2008	PC Programmentwicklung C# Funkklasse erweitert	10
05.01.2008	PC Programmentwicklung C# Funkklasse erweitert	6
06.01.2008	PC Programmentwicklung C# Funkklasse erweitert	12
08.01.2008	PC Programmentwicklung C# Variablen Klasse (Kalibrationsverwaltung)	3
09.01.2008	Überarbeiten der mechanischen Pläne für den Slave, Programmentwicklung C# (Protokollierung), Testschaltung für das Schaltnetzteil entwickeln, Testplatine für den Master, Slave und das Netzteil belichten, ätzen und bohren	2
16.01.2008	Bestückung der Master und Slave Platine, Netzteil Tests, Test der Master Platine	2
23.01.2008	Testen des Ladereglers, wickeln der Spulen und Beginn der Fertigung des Mastergehäuses	4

Datum	Tätigkeit	Stunden
28.01.2008	Ätzen einer neuen Slave Platine, Sprachdateibesprechung	2
30.01.2008	Fertigung des Mastergehäuses, bohren der Slave-Platine, beginnen zum Bestücken der Slave- Platine, testen des Slave- Energieverbrauch, überarbeiten des Netzteilentwurfs (Strombegrenzung und Status LED)	2
04.02.2008	Fertigstellen und testen der Slave- Platine, Bugsuche wegen erkennen des SHT11 trotz nicht Bestückung	3,5
06.02.2008	Testen der Strombegrenzung für das Netzteil, Dauertest der Ladeschaltung, Fertigstellung des Master- Gehäuses, Bugfxing der PC Software	2
14.02.2008	Testplatine des Netzteils ätzen und PC- Software-Dokumentation korrigieren	7
20.02.2008	Fertigen des Slave- Gehäuses, bestücken und testen des Netzteils und Dauertest des Messsystems	2
25.02.2008	Tests zum Biegen des Slave- Gehäuses	2
27.02.2008	Fertigung des Slavegehäuses (Deckel, Bodenplatte und Temperaturabnehmerteil), Layoutüberarbeitung und Test der induktiven Kopplung mit einem Metallring	2
28.02.2008	Bohren der Löcher des Slave- Gehäuses für Taster, LED, Feuchtigkeitskappe und Platinenmontagebohrungen; belichten, ätzen und bohren der neuen Slave- Platine	1
05.03.2008	Fertigen der Slave- Gehäuse	2
06.03.2008	Kleben des Magneten, schreiben des Jugend Innovativ- Berichtes und Fertigbestückung der Slave- Platine	1
12.03.2008	Bestücken und Testen der Netzteil- Platine, kleben der Einschubplatte an das Slave- Gehäuse, fertig stellen des Slave- Gehäuses ausgenommen der Spulenausnehmung, umlöten eines Gravitationssensors und beginn des Bestückens von zwei neuen Slave- Platinen	2
17.03.2008	Simulation der Antenne für den Slave	4
24.03.2008	Simulation der Antenne für den Slave und Fertigstellung der Fertigungsunterlagen für die Antenne	3
02.04.2008	Zusammenbauen des Netzteilgehäuses, Fertigung einer Testantenne und Wickeln der Spulen für die induktive Kopplung	2
03.04.2008	Fertigung des Netzteils, wickeln der Spulen, testen der Slave- Antenne und testen der induktiven Kopplung unter realen Bedingungen	1
07.04.2008	Bestückung zwei weiterer Slaves, testen der Slaves, ätzen der Antennenplatinen	8
09.04.2008	Fertigen der Leisten für zwei weitere Magnete, kleben der LED's in das Netzteilgehäuse, fertigen der Stecker für das Netzteil, fertigen von drei Antennen, zusammenbauen des Slaves	6,5

16.04.2008	Kleben der Leisten an einen Magneten, kleben von zwei Einschubplatten, Dokumentation der Inbetriebnahme von Master, Slave und Netzteil, Fertigstellung eines neuen Slave - Deckels	2
23.04.2008	Endmontage des Netzteils und der drei Slaves	2
	Anzahl der Stunden Gesamt:	**352**

Anhang D Stundennachweis für Diplomarbeit von Lenz Dominik

Datum	Tätigkeit	Stunden
02.07.2007	Informationen zur induktive Kopplung suchen	0,5
03.07.2007	Informationen zur induktive Kopplung suchen	2
04.07.2007	Informationen zur induktive Kopplung suchen	0,5
05.07.2007	Informationen zur induktive Kopplung suchen	2
06.07.2007	Informationen zur induktive Kopplung suchen	0,5
09.07.2007	Informationen zur induktive Kopplung suchen	0,5
10.07.2007	Informationen zur induktive Kopplung suchen	1
11.07.2007	Informationen zur induktive Kopplung suchen	0,5
12.07.2007	Informationen zur induktive Kopplung suchen	1
13.07.2007	Informationen zur induktive Kopplung suchen	0,5
15.07.2007	Bauteile suchen und bestellen	8
03.08.2007	1. Version des Pflichtenheftes	2
12.08.2007	Testen von Schaltungsteilen	8
25.08.2007	Projektkoordination und Protokollentwurf	8
05.09.2007	Erstellung eines Zeitplanes	2
08.09.2007	PSpice Simulation des Schaltnetzteiles	2
12.09.2007	Pflichtenheft und Präsentation	1
13.09.2007	Präsentation vorbereiten	1
16.09.2007	PSpice Simulation Übertragungsverhalten, Rohentwurf Abstract, Suche nach Programmierumgebung für PIC	3
18.09.2007	Präsentation vorbereiten, Ansteuerung der Sensoren	2,5
19.09.2007	Pflichtenheft und Präsentation überarbeiten, Schaltung für Master	3
22.09.2007	Einarbeiten in den PIC18F1320 bis Seite 15	0,5
23.09.2007	Einarbeiten in den PIC18F1320	1,5
24.09.2007	Präsentation vorbereiten, einarbeiten in den PIC 18F1320	2
25.09.2007	Einarbeiten in den PIC18F1320 (EUSART)	0,5
26.09.2007	Testen von Schaltungsteilen, Entwurf der Masterplatine	2
26.09.2007	Präsentation vorbereiten, Überlegungungen zur Konstruktion des Gehäuses	1
27.09.2007	Schaltplan Slave und Netzteil erstellen	2
28.09.2007	Schaltpläne für Netzteil und Slave in Eagle zeichnen	1,5
29.09.2007	Layoutentwicklung Slave und Ladeteil Akku	4
02.10.2007	Layoutentwicklung Slave und Ladeteil Akku	0,5
03.10.2007	Testlayout fertig stellen und drucken	0,5
08.10.2007	Testplatine belichten und ätzen	1,5
09.10.2007	Softwareprogrammbesprechung, Organisation für den Projekttag	1

Datum	Tätigkeit	Stunden
10.10.2007	Testplatine bohren und bestücken, PIC Programm beginnen, Netzteil testen mit MOSFET als Schalter, testen der induktiven Kopplung	2
10.10.2007	Dokumentation Berechnung des Netzteils	2
13.10.2007	Dokumentation Berechnung des Netzteils	1
16.10.2007	Dokumentation Berechnung des Netzteils	2,5
17.10.2007	Testplatine bestücken, PIC Programm beginnen, Netzteil testen mit MOSFET als Schalter	2,5
18.10.2007	Versuch den PIC mit einem externen Oszillator anzusteuern	2,5
19.10.2007	Ansteuern des PIC mit einem externen Oszillator, Konfiguration der USART Schnittstelle	3,5
20.10.2007	Konfiguration der USART Schnittstelle	5
21.10.2007	Konfiguration der USART Schnittstelle	1
23.10.2007	Konfiguration des A/D- Converter und Stückliste schreiben	2
24.10.2007	Testen des A/D- Wandlers, testen des Ladereglers und des Aufwärtswandlers, Besprechung des PIC Programmdesigns	1
24.10.2007	Besprechung der Befehle für Master und Slave	1,5
28.10.2007	Struktogramm für das Slave PIC Programm und Besprechung, Initialisierung von Timern	3,5
29.10.2007	Initialisierung von Timern, Programmdesign Slave	5
30.10.2007	Zuweisung von Befehlen und Start des Slave Programms	4
31.10.2007	Schreiben des PIC Slave Programms (Adressierteil)	1
01.11.2007	Schreiben des PIC Slave Programms (Adressierteil)	2,5
02.11.2007	Schreiben des PIC Slave Programms (Sende und Empfangsroutine)	3,5
04.11.2007	Besprechung des Slave PIC Programms und Bugfixing, Testen diverser Programmteile	6
05.11.2007	Schreiben des PIC Programms Adressierteilüberarbeitung und testen	2,5
06.11.2007	Besprechung der Sensorenansteuerung, Bugfixing des Adressierteils im PIC Programm	2
07.11.2007	Fertigstellung des Adressierteils des PIC Slave Programms, Überarbeitung der Befehlscodes für das PIC Programm, Beginn mit der Befehlsauswertungsroutine, Entwurf erster Gehäusekonzepte	2
07.11.2007	Projektdokumentation (PIC Programm und mechanische Erkenntnisse)	2,5
10.11.2007	Initialisierung des EEPROM's vom PIC (schreiben und lesen)	1,5
10.11.2007	Programmentwicklung PIC (Befehlsverarbeitung fertig gestellt, RealTime Tests, Timeout und Sperrtimer hinzugefügt, zeitverzögertes Abschalten und Kanalwechsel, Bugfixing, Status LED initialisiert, Kommunikationsprotokoll für den SHT11 umgesetzt, erste Messungen vorgenommen)	15,5
13.11.2007	PIC Slave Programmdokumentation (Einbinden der Bibliotheken, Konfigurationsbits des PIC und Definition der globalen Variablen)	4

Datum	Tätigkeit	Stunden
14.11.2007	Überarbeitung des Slave und Master Schaltplanes, layouten der Master und Slave Platinen, überarbeiten der Stückliste	2
14.11.2007	Schreiben des Wochenberichtes, Dokumentation des Slave PIC Programms	2
15.11.2007	Dokumentation des Slave PIC Programms, Layoutentwicklung Slaveplatine	6
15.11.2007	Präsentation der Diplomarbeit für BOSCH	2
16.11.2007	Testplatine für Master und Slave belichten, ätzen, bohren und bestücken	9
21.11.2007	Bugfixing des Slaveprogrammes (Kanalwechsel und Kanalbelastungsmessung), Funkmodul untersuchen für den Kanalwechsel ("ACK"- Befehl untersuchen!)	2
26.11.2007	PIC Slaveprogramm (Befehlsverarbeitung für den Beschleunigungs-, Temperatursensor und den Batteriestand schreiben)	0,5
28.11.2007	Ansprechen des Beschleunigungssensors (hat nicht geklappt, da einige Kontakte fehlerhaft waren!), Überarbeiten des Masterlayouts, Beginn der Zeichnungen des mechanischen Gehäuses für Master und Slave, Layouten des Netzteiles und der Ladeschaltung für den Akku	2
30.11.2007	Layouten für das Netzteil und den Akkuladeteil	2
03.12.2007	Belichten und ätzen der Testplatine (Akkulader, Master Endversion und Netzteil)	1,5
05.12.2007	Ansprechen des Beschleunigungssensors und Übertragung des ersten Messstreams, Zuschneiden und bestücken der Akkulader- und Netzteiltestplatine, Überarbeitung des Masterlayoutes für die endgültige Version, zeichnen der mechanischen Pläne für den Master und Slave	2
06.12.2007	Layout für die Slave Endversion fertigen	3
12.12.2007	Zeichnen der mechanischen Pläne für den Master und Slave, testen des Akkuladers und Netzteils, Überarbeitung des PIC Master und Slave Programms, Performance Tests der Übertragungsrate	1,5
18.12.2007	Überarbeitung des Slaveschaltplans für die Betaversion, erstellen der Stückliste für den Slave	4
19.12.2007	Zeichnen der mechanischen Pläne für den Slave, überarbeiten des Slave Eaglelayouts, erstellen einer Stückliste für Master und Slave, suchen eines geeigneten Akkus, testen der induktiven Kopplung (Netzteil macht Probleme beim Treiben des Stromes)	2
22.12.2007	Layout für die Slave Endversion fertigen	4
23.12.2007	Layout für die Slave Endversion fertigen	1,5
26.12.2007	Dokumentation des Slave PIC Programms	3,5
27.12.2007	Dokumentation des Slave PIC Programms	2,5
28.12.2007	Dokumentation des Slave PIC Programms, Konzept für die Aufteilung der Diplomarbeit Dokumentation erstellen	5
01.01.2008	Dokumentation Pflichtenheft überarbeiten, Funktionsprinzip	1

Datum	Tätigkeit	Stunden
02.01.2008	Dokumentation Funktionsprinzip und PIC Slave Programm Bugfixing	1
03.01.2008	PIC Slave Programm Bugfixing (Kanalwechsel und Abschalten), Programmentwicklung C# (Entwicklung der Funkalgorithmen), Dokumentationsbesprechung	6
03.01.2008	Überarbeitung der PIC Slave Programmdokumentation	2
09.01.2008	Überarbeiten der mechanischen Pläne für den Slave, Programmentwicklung C# (Protokollierung), Testschaltung für das Schaltnetzteil entwickeln, Testplatine für den Master, Slave und das Netzteil belichten, ätzen und bohren	2
09.01.2008	Layoutüberarbeitung des Masters	1,5
16.01.2008	Bestückung der Master und Slave Platine, Netzteil Tests, Test der Master Platine	2
16.01.2008	Schreiben des Zwischenberichts	2
19.01.2008	Dokumentation der Bauteile des Slaves	6,5
20.01.2008	Dokumentation der Bauteile des Masters	3
21.01.2008	Dokumentation der Netzteils	2
22.01.2008	Dokumentationsbesprechung des Master und Slave, Dokumentation des Funkprotokolls	5,5
23.01.2008	Diverse Tests des Ladereglers, wickeln der Spulen und Fertigung des Mastergehäuses	4
28.01.2008	Ätzen einer neuen Slave Platine, überarbeiten der Sprachdatei, Sprachdateibesprechung	4
30.01.2008	Fertigung des Mastergehäuses, bohren der Slave-Platine, beginnen zum Bestücken der Slave- Platine, testen des Slave- Energieverbrauch, überarbeiten des Netzteilentwurfs (Strombegrenzung und Status LED)	2
04.02.2008	Fertigstellen und testen der Slave- Platine, Bugsuche wegen erkennen des SHT11 trotz nicht Bestückung	3,5
06.02.2008	Testen der Strombegrenzung für das Netzteil, Dauertest der Ladeschaltung, Fertigstellung des Master- Gehäuses, Bugfxing der PC Software	2
07.02.2008	Dokumentation des Netzteils	3
09.02.2008	Dokumentation des Netzteils	2,5
12.02.2008	PC- Software testen und verfassen einer Bedienungsanleitung	5,5
13.02.2008	Verfassen der PC Software Bedienungsanleitung	5
14.02.2008	Testplatine des Netzteils ätzen und PC- Software-Dokumentation korrigieren	7
20.02.2008	Fertigen des Slave- Gehäuses, bestücken und testen des Netzteils und Dauertest des Messsystems	2
25.02.2008	Tests zum Biegen des Slave- Gehäuses	2
27.02.2008	Fertigung des Slavegehäuses (Deckel, Bodenplatte und Temperaturabnehmerteil), Layoutüberarbeitung und Test der induktiven Kopplung mit einem Metallring	2
28.02.2008	Bohren der Löcher des Slave- Gehäuses für Taster, LED, Feuchtigkeitskappe und Platinenmontagebohrungen; belichten, ätzen und bohren der neuen Slave- Platine	1

Datum	Tätigkeit	Stunden
06.03.2008	Kleben des Magneten, schreiben des Jugend Innovativ- Berichtes und Fertigbestückung der Slave- Platine	1
12.03.2008	Bestücken und Testen der Netzteil- Platine, kleben der Einschubplatte an das Slave- Gehäuse, fertig stellen des Slave- Gehäuses ausgenommen der Spulenausnehmung, umlöten eines Gravitationssensors und beginn des Bestückens von zwei neuen Slave- Platinen	2
15.03.2008	Dokumentation des Slave - Programms für die Diplomarbeit	9,5
16.03.2008	Dokumentation des Slave - Schaltplans für die Diplomarbeit	10,5
17.03.2008	Dokumentation des Netzteils für die Diplomarbeit	9
26.03.2008	Fertigung des Netzteilgehäuses (Deckel und Boden)	2
27.03.2008	Fertigung von zwei Einschubplatten und der Fixierplatten des Netzteils	2
02.04.2008	Zusammenbauen des Netzteilgehäuses, Fertigung einer Testantenne und wickeln der Spulen für die induktive Kopplung	2
03.04.2008	Fertigung des Netzteils, wickeln der Spulen, testen der Slave- Antenne und testen der induktiven Kopplung unter realen Bedinungungen	1
07.04.2008	Bestückung zwei weiterer Slaves, testen der Slaves, ätzen der Antennenplatinen	8
09.04.2008	Fertigen der Leisten für zwei weitere Magnete, kleben der LED's in das Netzteilgehäuse, fertigen der Stecker für das Netzteil, fertigen von drei Antennen, zusammenbauen des Slaves	6,5
16.04.2008	Kleben der Leisten an einen Magneten, kleben von zwei Einschubplatten, Dokumentation der Inbetriebnahme von Master, Slave und Netzteil, Fertigstellung eines neuen Slave - Deckels	2
23.04.2008	Endmontage des Netzteils und der drei Slaves	2
	Anzahl der Stunden Gesamt:	**345,5**

Anhang E Projekttagebuch

5.9.2007 bis 26.9.2007

Laufende Arbeiten werden schwarz eingefärbt.
Erledigte Arbeiten werden grün eingefärbt.
Arbeiten die Probleme machen werden rot eingefärbt.

- 1. Entwurf Pflichtenheft
- Präsentation
- Diplomarbeitsantrag
- Programmentwicklung C#
  - Programmkonzept
  - Klassen:
    - Hauptprogramm
    - Variablen
    - Einstellungen
    - Sprachen
    - MainForm (Hauptfenster)
    - Eigenschaften
    - Anzeigefenster
    - Paint
    - Beep
- Einarbeiten in PIC 18F1320/ 18F1220
- Einarbeiten in diverse Sensoren
- Platinen- und Schaltungsentwicklung:
  - Testplatine Master
  - Testplatine Slave
  - Testaufbau Ladestation
- Gehäuseentwicklung:
  - Erste Ideensammlungen
    - Material
    - Prinzipielles Design
  - Probleme mit dem Magneten

Nächste Woche kümmern wir uns um das Problem mit der Ladestation und das des Gehäuses für den Magneten.
Weiters haben wir vor mit den PIC Programmen zu beginnen.

27.9.2007 bis 3.10.2007

Im Verlauf dieser Woche wurde sich in das Datenblatt des PIC18F1320 eingelesen. Weiters wurde für die Ladestation die Schaltung etwas abgeändert aufgebaut und getestet. Außerdem wurde die Testplatine fertig gestellt.
Da wir aber leider keine bedruckbaren Folien hatten, konnte die Platine nicht geätzt werden.
Am Montag haben wir vor unsere Testplatine zu ätzen, sodass wir am Mittwoch bereits mit dem PIC- Programm und den Tests der Ladeschaltung für unseren Akku, als auch den Aufwärtswandlertests beginnen können.

4.10.2007 bis 10.10.2007

Diese Woche wurde die Testplatine gebohrt und verzinnt. Danach wurde damit begonnen die Testplatine zu bestücken.
Weiters wurde der Testaufbau der Ladestation um die MOSFET, welche als Schalter dienen, erweitert.
Dabei trat aber ein Problem auf, da der MOSFET nicht so durchgeschaltet wurde wie erwartet.
Deshalb wird nächste Woche versucht mit 3 parallel geschalteten Gattern den MOSFET anzusteuern, um somit einen höheren Ansteuerstrom zu erzielen.
Außerdem wurde die Frage bezüglich ID für die Slaves oder nicht beantwortet.
Im Weiteren wurden erste Testprogramme für den PIC 18F1220 geschrieben und getestet.

Nächste Woche wird die Testplatine fertig bestückt.
Außerdem wird sich um das Problem mit der MOSFET Ansteuerung gekümmert.
Der Ladeteil der Testplatine wird getestet werden und es werden weitere PIC Testprogramme geschrieben werden.
Diese Testprogramme werden sich Hauptsächlich mit dem Ansprechen des Transceiver- Moduls beschäftigen.

10.10.2007 bis 17.10.2007

Die Platine wurde nahezu vollständig bestückt.
Weiters wurde die Master Platine getestet und das Problem mit der Ansteuerung der MOSFET des Netzteils der Ladestation behoben.
Beim Testen der Slave Platine wurde festgestellt, dass der Laderegler MAX1736 nicht wie erwartet funktioniert.
Bei der Fehlersuche stellte sich heraus, dass ein falscher MOSFET verwendet wurde und 2 Anschlüsse vertauscht wurden.
Das Problem mit den Anschlüssen wird nächste Woche behoben und danach wird der Regler nochmals getestet.
Bei der PC Software wurden folgende Teile hinzugefügt:

Nächste Woche wird der Akku- Ladeschaltung und den PIC Programmen gewidmet.

17.10.2007 bis 24.10.2007

Es wurde die RS232 Kommunikation zwischen PIC und Hyperterminal erfolgreich getestet.
Danach wurde der A/D- Wandler des PIC erfolgreich getestet.
Anschließend wurde der Laderegler MAX1736 erfolgreich getestet.
Danach wurde die Schaltung zum Ab- bzw. Aufdrehen des Aufwärtswandlers (L6920DB), sowie der Aufwärtswandler selbst getestet. Beides hat wie erwartet funktioniert.

Die nächste Zeit wird für das PIC- Programm des Masters bzw. Slaves verwendet.
Dabei wird sich hauptsächlich um die Kommunikation zwischen PIC und Sensoren, als auch um die Kommunikation zwischen PIC – PC bzw. PIC- PIC gekümmert.

24.10.2007 bis 7.11.2007

In den vergangenen zwei Wochen wurde die Adressenverarbeitung des PIC Programms fertig gestellt und getestet. Es wurden die Befehlscodes zugewiesen und mit der Befehlsverarbeitung des PIC Programms begonnen. Weiters wurden erste Konzepte für das Gehäuse angefertigt, die Platzaufteilung der einzelnen Elemente im Gehäuse festgelegt, eine ungefähre Antenneauswahl getroffen und die Spule für das Laden über induktive Kopplung gewickelt.

In nächster Zeit wird die Befehlsverarbeitung des PIC fertig gestellt und die Kommunikation mit den Sensoren hergestellt. Sobald wie möglich wird dann versucht Daten über die Funkstrecke zu schicken.

7.11.2007 bis 14.11.2007

In dieser Woche wurde das PIC Slaveprogramm soweit fertig gestellt, dass eine Kommunikation mit dem Sensor SHT11 aufgebaut werden konnte und erste Messdaten an den PC gesendet werden konnte. Weiters wurde die Befehlsverarbeitung und Adressenberechnung fertig gestellt. Das PIC Masterprogramm wurde diese Woche ebenfalls fertig gestellt.

In der kommenden Woche werden die neuen Master- und Slaveplatinen geätzt, bestückt und gelötet. Danach wird sich wieder mit dem Gehäuse und dem Netzteil befasst.

14.11.2007 bis 21.11.2007

Diese Woche wurde die Master- und Slaveplatine geätzt und fertig bestückt. Weiters wurden die Platinen auf ihre Funktionalität geprüft.
Es wurden erste Temperatur- und Feuchtigkeitsmessstreams erfolgreich an den PC übertragen.
Außerdem wurde diese Woche ein Problem bei der Kanalbelastungsmessung und beim Kanalwechsel festgestellt. Das Funkmodul wurde daraufhin noch mal auf seine Funktionalität getestet und dabei wurde festgestellt, dass das Funkmodul den für die Ausführung eines Befehls notwendigen „ACK“ Befehl nicht richtig erkennt. Deshalb funktioniert der Kanalwechsel noch nicht richtig.
Woran konnte bisher noch nicht herausgefunden werden.

Im Lauf der nächsten Woche soll der Kanalwechselfehler genau untersucht und behoben werden. Außerdem soll sofern genügend Zeit ist die Ladestation weiter getestet werden.

21.11.2007 bis 28.11.2007

Diese Woche konnte der Fehler beim Kanalwechseln des Funkmoduls behoben werden.
Das PIC Slaveprogramm wurde so erweitert, dass der Beschleunigungssensor angesprochen werden kann. Das Programm konnte leider nicht getestet werden, da der Sensor keinen vernünftigen Kontakt mit den Leiterbahnen hatte.
Weiters wurde mit dem Testlayout für das Netzteil begonnen und das Masterlayout überarbeitet.
Außerdem wurden erste Konstruktionspläne für das mechanische Gehäuse des Master und Slave angefertigt.

Nächste Woche wird die Testplatine für das Netzteil geätzt, bestückt und gebohrt. Sofern danach noch genügend Zeit ist, wird die Platine getestet.
Außerdem wird versucht den Beschleunigungssensor neu zu kontaktieren und nochmals anzusprechen.
Weiters wird an den mechanischen Pläne für den Master und Slave weitergearbeitet.

28.11.2007 bis 5.12.2007

Diese Woche wurde die Testplatine für das Netzteil und den Akkulader belichtet, geätzt und gebohrt. Mit der Bestückung des Netzteils wurde bereits begonnen jedoch konnte sie nicht ganz fertig gestellt werden.
Weiters wurde der Beschleunigungssensor erfolgreich angesprochen und erste Messdaten per Funk übertragen. Jedoch passte die Anzeige der Messdaten im Softwareprogramm noch nicht.
Außerdem wurde heute das Layout des Masters überarbeitet, sodass die heutige Version vermutlich die endgültige ist.
Die mechanischen Pläne für den Master sind nahezu fertig und die Grundzüge des Slave Gehäuses wurden auch festgelegt.

Im Verlauf der nächsten Woche wird ein neues Slavelayout erstellt, der Akkulader und das Netzteil fertig bestückt und getestet und mit der Fertigung des Masters begonnen.

5.12.2007 bis 12.12.2007

Diese Woche wurde die Testplatine für das Netzteil und den Akkulader fertig bestückt und auf ihre Funktion getestet. Die induktive Übertragung konnte jedoch nicht mehr getestet werden.
Weiters wurde die Anzeige für den Beschleunigungssensor angepasst und der letzte Sensor erfolgreich angesprochen. Somit sind alle vier Messsensoren betriebsbereit.
Außerdem wurde der Watchdog Timer in das Master und Slave PIC Programm eingebaut.
Die mechanischen Pläne für des Slave Gehäuses wurden nahe zu fertig gestellt.
Bei einem ersten Performance Test wurde festgestellt, dass maximal 17 Messdaten/s übertragen werden können.

Im Verlauf der nächsten Woche wird ein die Induktive Übertragung getestet und versucht das Messsystem schneller zu machen.

12.12.2007 bis 20.12.2007

Diese Woche wurde die induktive Übertragung getestet.
Es wurde festgestellt, dass die Übertragung nicht wie erwartet funktioniert hat, deshalb wurde beschlossen einen fertigen PushPull IC für das Netzteil zu verwenden.
Weiters wurden die Konzepte für das mechanische Gehäuse des Slaves besprochen und mit der Erstellung der Konstruktionspläne begonnen.
Außerdem wurden die Stücklisten für den Master und Slave fertig gestellt.
Es wurde weiters auch noch die Kanalbelastungsmessung erfolgreich geprüft.

Im Zuge der Weichnachtsferien wird mit der Programmdokumentation für das PIC Master und Slave Programm, das Softwareprogramm begonnen. Außerdem wird das Softwareprogramm fertig gestellt und versucht das Messsystem zu beschleunigen.
Weiters werden die mechanischen Pläne für Master und Slave fertig gestellt.

20.12.2007 bis 10.1.2008

Während den Weihnachtsferien wurden noch ein paar Bugs des PIC Programms ausgebessert. Anschließend wurden die PIC Software für den Master und Slave fertig dokumentiert.
Es wurde auch nach Möglichkeiten gesucht das Messsystem zu beschleunigen, dabei wurde festgestellt, dass die meiste Zeit in den Funkmodulen verschwendet wird. Da aber keine neuen Funkmodule verfügbar sind, welche unseren Anforderungen entsprechen muss darauf verzichtet werden.
Weiters wurde an der PC Software um den Menüpunkt Funkverkehr erweitert.
In der Woche nach den Ferien wurde der PushPull IC MAX256 für das Netzteil getestet und die mechanischen Pläne für den Master und Slave nahezu fertig gestellt.
Außerden wurden weitere Testplatinen für den Slave und die engültige Platine für den Maste gefertigt.
Mit der Bestückung der Platinen wurde bereits begonnen.

Nächste Woche wird die Master- und Slave- Platine fertig bestückt und mit der Fertigung der Gehäuse für den Master und Slave begonnen.
Außerdem werden weitere Tests mit dem Netzteil IC durchgeführt werden.

10.1.2008 bis 17.1.2008

Diese Woche wurde die neue Masterplatine fertig bestückt und erfolgreich getestet.
Die Slave Platine wurde fertig mit allen Bauteilen bestückt. Von den Sensoren wurde bislang nur der Gravitationssensor bestückt. Bei der Funktionskontrolle wurde festgestellt, dass alle benötigten Spannungen vorhanden sind und der PIC erfolgreich anschwingt, jedoch konnten keine Messdaten des Gravitationssensors abgeholt werden. Der Fehler wurde noch nicht gefunden.
Weiters wurde der Netzteil IC anstelle einer Schaltfrequenz von 1MHz auf 100kHz getestet.
Dabei wurde festgestellt, dass immer noch schöne Rechtecksignale übertragen werden und wenn der Kern nicht geschlossen ist kommt es zu Spannungsspitzen beim Schalten.

Nächste Woche wird das Netzteil weiter getestet und der Fehler beim Ansprechen des Gravitationssensors gesucht. Außerdem wird mit der Gehäusefertigung begonnen.

17.1.2008 bis 24.1.2008

Diese Woche wurden die Spulen für die induktive Kopplung gewickelt und einige Tests damit durchgeführt. Dabei wurde festgestellt, dass die Energieübertragung bei einer Schaltfrequenz von 100kHz besser ist. Außerdem wurde mit unterschiedlichen Wicklungszahlen experimentiert. Dabei wurde festgestellt, dass 10Windungen auf der Primärseite und 20 Windungen auf der Sekundärseite optimal sind. Beim Vergrößern des Luftspalts wurde festgestellt, dass die Kopplung ab ca. 1mm Luftspalt nicht mehr gegeben ist, deshalb muss bei der Fertigung darauf geachtet werden, dass der Spalt nicht mehr als ca. 0,1 bis 0,2mm beträgt.
Weiters wurde die Software getestet und diverse Bugs behoben.
Außerdem wurde der mechanische Plan des Masters überarbeitet.
An den Layouts des Master und Slave wurden kleine Fehle behoben und anschließend eine neue Platine geätzt.
Die mechanische Fertigung des Gehäuses für Master und Slave verzögert sich leider um eine weitere Woche, da keine Lehrer der Werkstätte anwesend waren.

Im Laufe der nächsten Woche werden die neuen Master- und Slave- Platinen gebohrt, bestückt und getestet.
Weiters wird versucht endlich das Gehäuse zu fertigen.

24.1.2008 bis 30.1.2008

Diese Woche wurde der endgültige Master gefertigt.
Das Gehäuse für den Master wurde bis auf den Deckel bereits fertig geklebt und muss nur noch ein wenig gekürzt werden.
Die neue Slave- Platine wurde fertig geätzt und gebohrt. Mit der Bestückung der Platine wurde bereits begonnen.
Weiters wurde der Stromverbrauch des Slave getestet. Dabei wurde festgestellt, dass der Verbrauch bei 3,7V 100mA beträgt.
Es wurde versucht den Stromverbrauch zu reduzieren, indem anstelle des Quarzes ein Oszillator eingebaut wurde. Dabei erhöhte sich jedoch nur der Verbrauch um 20mA.
Weiters wurde der Verbrauch des Funkmoduls und des PIC gemessen. Nähere Untersuchungen ergaben, dass keine Einsparungen beim Funkmodul vorgenommen werden können.
Beim Messen des PIC wurde festgestellt, dass ein deaktivieren der Messsensorenabfrage, wenn sie nicht benötigt werden kaum Einsparungen bringt. Deshalb wurde beschlossen auf diesen Sparmodus zu verzichten.
Weiters wurde das Netzteil um eine Strombegrenzungsschaltung und eine Ladestatus LED erweitert.

Nächste Woche wird das Slave- Gehäuse gefertigt und die Slave- Platine weiter bestückt und getestet und das Netzteil weiterentwickelt.

30.1.2008 bis 7.2.2008

Diese Woche wurde der Master komplett fertig gestellt.
Weiters wurde die neue Slave- Platine fertig bestückt und erfolgreich getestet.
Die Schaltung für das Netzteil wurde um eine Strombegrenzung erweitert. Die Schaltung wurde für erste Funktionstests provisorisch aufgebaut und erfolgreich getestet.
Es wurde erstmals ein Akku mit dem provisorischen Netzteil und dem Laderegler des Slave über die induktive Kopplung erfolgreich geladen.
Diese Woche wurde mit der Entwicklung des Gehäuse und des Layout für das Netzteil begonnen. Dabei wurde beschlossen, dass eine Ladestation zum Laden von bis zu drei Slaves gleichzeitig entwickelt wird.
Außerdem wurde diese Woche ein Bug der PC Software behoben, der zum Programmabsturz führte, sobald der Master während des Betriebes entfernt wurde.
Es wurde auch ein PIC Software Bug des Slave behoben der dafür sorgte, dass ein nicht bestückter Sensortyp als bestückt erkannt wurde.

In den Semesterferien wird an der Dokumentation der PC Software, des Netzteils und der mechanischen Pläne weitergearbeitet. Außerdem wird der Bericht für Jugend Innovativ und den BOSCH Preis verfasst.
Weiters wird die Bedienungsanleitung für die PC Software geschrieben.

7.2.2008 bis 21.2.2008

In den Semesterferien wurde die Dokumentation der PC Software fertig gestellt und das Layout für das Netzteil entwickelt. Außerdem wurde der Bericht für Jugend verfasst.
und die Bedienungsanleitung für die PC Software geschrieben.
Weiters wurde die Platine für das Netzteil geätzt, bestückt und erfolgreich getestet. Es wurde festgestellt, dass die Stecker des Netzteils zu ändern sind.
Außerdem wurden Dauertests mit dem Messsystem durchgeführt. Dabei wurde ein Bug entdeckt, den vermutlich ein Überlauf des Mikrokontrollers verursacht. In weiteren Tests wird versucht diesen Bug zu rekonstruieren und mit einem Bugfix zu beheben.
Diese Woche wurden für den Slave die Einschubplatte und die Schiebeführungen gefertigt.

In der nächsten Woche werden die richtigen Stecker für das Netzteil und den Slave ins Layout eingezeichnet und danach eine neue Netzteilplatine geätzt. Weiters wird versucht das Slavegehäuse komplett fertig zu stellen und die Antennenberechnungssoftware wird versucht zu installieren.

21.2.2008 bis 28.2.2008

Diese Woche wurde die Slave- Platine überarbeitet und die neue Version geätzt und gebohrt.
Weiters wurde für das Gehäuse des Slaves der Deckel, die Bodenplatte und der Temperaturabnehmervorichtung gefertigt. Außerdem wurde der Klick- Mechanismus erfolgreich getestet.
Die Montagebohrungen des Slave- Gehäuses für die Platine wurden ebenfalls gebohrt.
Weiters wurde die induktive Kopplung mit einen Kupferring um den Kern getestet. Dabei wurde festgestellt, dass dies keine Auswirkungen zeigt.

Im Laufe dieser Woche wird die neue Slave- Platine fertig bestückt.
Außerdem wird die Ausnehmung für die Spule gefertigt und danach versucht das Slave- Gehäuse fertig zusammen zu setzen.
Weiters wird überlegt, ob die Distanzbolzen für die Montage der Platine selbst gefertigt werden oder zugekaufte auf die richtige Länge gekürzt werden und ob die Feder selbst gebaut wird.

28.2.2008 bis 6.3.2008

Diese Woche wurde die neue Slave- Platine fertig bestückt und erfolgreich getestet.
Außerdem wurden diese Woche zwei weitere Blechgehäuse mit sämtlichen Montagebohrungen fertigt. Weiters wurden zwei weitere Temperaturabnehmerteile aus Messing gefertigt und die Federn wurden ebenfalls angefertigt.
Zum Abschluss wurden noch die Führungsschienen an den Magneten geklebt.

Im Laufe der nächsten Woche wird die Einschubplatte an das Blechgehäuse geklebt und danach die Ausnehmung für die Spule angefertigt.
Außerdem wird mit dem Bau des Netzteilgehäuses begonnen und eine neue Netzteilplatine geätzt.

6.3.2008 bis 12.3.2008

Diese Woche wurde die Einschubplatte an das Slave- Gehäuse geklebt und die Vorrichtung für die punktuelle Temperaturmessung wurde komplett fertig gestellt.
Weiters wurde eine neue Netzteil- Platine geätzt, gebohrt, bestückt und erfolgreich getestet.
Außerdem wurde ein Gravitationssensor von einer früheren Testplatine entfernt und auf die aktuelle Platine gelötet. Weiters wurde damit begonnen zwei weitere Slave- Platinen zu bestücken.
Mit dem Bau des Netzteil- Gehäuses konnte auf Grund klebetechnischer Probleme, welche sehr viel Zeit in Anspruch nahmen, leider nicht begonnen worden.

In den Ferien wird mit dem Verfassen der Diplomarbeit begonnen und es werden diverse Langzeittests mit dem Messsystem durchgeführt werden.
Nach den Ferien wird sofort mit dem Bau des Netzteil- Gehäuses begonnen und die Spulenausnehmung für die induktive Kopplung im Slave fertig gestellt.

12.3.2008 bis 27.3.2008

In den Ferien wurde ein Großteil der Diplomarbeit verfasst.
Diese Woche wurde mit dem Bau des Netzteilgehäuses begonnen. Der Deckel, Boden und Kühlkörper wurden bereits fertig gestellt. Weiters wurden die Fixerhalterungen gefertigt. Jedoch blieb keine Zeit mehr für die Bohrung der Löcher.
Außerdem wurde das Schaltnetzteil passend umgebaut.
Weiters wurden zwei weitere Einschubplatten gefertigt.

Nächste Woche werden die Leisten für zwei weitere Magnete gefertigt und das Netzteilgehäuse fertig zusammengebaut.

27.3.2008 bis 3.4.2008

Diese Woche wurde eine erste Testantenne gefertigt und erfolgreich getestet.
Weiters wurde die Ladestation fertig zusammengesetzt und erfolgreich mit dem umgebauten Netzteil getestet.
Außerdem wurden die Spulen für die induktive Kopplung für drei Slaves gewickelt.
Anschließend wurde die induktive Kopplung zwischen der Ladestation und den Slaves getestet.
Dabei wurde festgestellt, dass die induktive Kopplung mit 14 Windungen auf der Primärseite besser funktioniert als mit den berechneten 10 Windungen.

Im Laufe der nächsten Woche werden drei endgültige Antennen für die Slaves gefertigt. Außerdem wird versucht die Einschubleisten für zwei weitere Magnete zu fertigen und die Spule in das Slave - Gehäuse einzugießen. Wenn möglich wird versucht die übrigen zwei Slave – Gehäuse fertig zu kleben und zwei weitere Slave Platinen zu bestücken.
Weiters wird versucht das Netzteil in der nächsten oder übernächsten Woche komplett fertig zu stellen.

3.4.2008 bis 10.4.2008

Diese Woche wurden die Einschubleisten für zwei weitere Magneten gefertigt.
Außerdem wurden die Spulen für die induktive Kopplung und die Signal LED's in das Netzteilgehäuse geklebt. Das Netzteilgehäuse wurde auch lackiert und fertig zusammengesetzt.
Weiters wurde bereits in ein Slave – Gehäuse die Spule für die induktive Kopplung geklebt und danach wurde das Gehäuse lackiert und fertig zusammengesetzt.
Abschließend wurde die Energieübertragung zwischen Slave und Netzteil erfolgreich mit einem Ladestrom von 200mA getestet.
Weiters wurden drei Antennen für die Slaves gefertigt. Beim Testen der Antennen wurde festgestellt, dass die Reichweite nicht sonderlich zufrieden stellend war.
Diese Woche wurden außerdem noch zwei weitere Slave – Platinen bestückt und erfolgreich getestet.
Außerdem wurden die Spulen für die induktive Kopplung für zwei weitere Slaves gefertigt und es wurden zwei weitere Sensoren in die Temperaturabnehmervorrichtung eingegossen.
Weiters wurde festgestellt, dass die vorhandenen Deckel für die zwei übrigen Slave – Gehäuse zu kurz sind. Deshalb wurde mit der Fertigung von zwei neuen Deckeln begonnen.

Nächste Woche werden zwei weitere Magnete und die Einschubplatten an zwei weitere Gehäuseunterteile geklebt.
Außerdem wird versucht die zwei neuen Slave – Deckel an das Gehäuse anzupassen.
Falls noch Zeit bleibt wird versucht die Antennen zu optimieren.

10.4.2008 bis 17.4.2008

Diese Woche wurden die Einschubleisten an einen weiteren Magneten geklebt.
Weiters wurden die Einschubplatten an zwei weitere Gehäuseunterteile geklebt.
Anschließend wurden die Ausnehmungen für die Spulen der beiden Slaves fertig gestellt und danach wurden die Spulen eingeklebt.

Im Laufe der nächsten Woche wird versucht die zwei fehlenden Slave – Gehäuse und den Deckel des Netzteils zu lackieren.

17.4.2008 bis 24.4.2008

Diese Woche wurden das Netzteil und die drei Slaves fertig zusammen gebaut.
Leider ging bei der Endmontage ein fertig geklebter Magnet zu bruch.
Deshalb wurde nach einem weiteren Magneten gesucht.

Nächste Woche wird ein weiterer Magnet geklebt.

Danksagung

Abschließend möchten wir uns an dieser Stelle noch für die tolle Zusammenarbeit mit den Professoren und Werkstättenlehrern bedanken ohne deren Hilfe wir diese Diplomarbeit nicht hätten fertig stellen können.
Vor allem möchten wir uns bei unserem Projektbetreuer DI Geza Beszedics bedanken ohne dessen Hilfe und vertrauen wir diese Diplomarbeit gar nicht machen hätten können.
Er hat uns oft seine Freizeit geopfert um uns bei der Diplomarbeit beizustehen.

Wir möchten uns aber auch für die tolle Zusammenarbeit mit den Werkstättenlehrern der Abteilung Elektronik bedanken, die uns stets ihre Maschinen und Werkzeuge zur Verfügung gestellt haben.

Außerdem wollen wir uns bei den Theorielehrern der Abteilung Elektronik bedanken, denn sie haben uns oft ihr spezifisches Fachwissen zur Verfügung gestellt und entschuldigten des Öfteren unser Fernbleiben vom Unterricht damit wir an der Diplomarbeit weiterarbeiten konnten.

Unser Dank gilt auch der Firma Farnell die uns mit zahlreichen Bauelementen gesponsert hat und es somit ermöglichte mehrere funktionstüchtige Slaves zu bauen.