FLASHCRAFT Funkboard

Dokumentation – Revision 1.1.0 - 07.12.08

Florian Scherb
www.flashcraft.de

Inhaltsverzeichnis

1. Überblick

1.1 Allgemeines

Das FLASHCRAFT Funkboard basiert auf dem RFM12S Tranceiver von HOPE RF und ist mit einem ATmega32 ausgestattet. Dieser Mikrocontroller soll alle teils sehr aufwändigen Kommunikations- und Konfigurationsroutinen übernehmen und als eine Art „Blackbox“ wirken. Der Anwender kann somit eine umfangreiche und ausgereifte Funklösung sehr einfach in sein Projekt einbinden ohne sich mit komplexeren Zusammenhängen beschäftigen zu müssen.
Es wurde jedoch darauf geachtet neben der einfachen Bedienung auch eine möglichst hohe Flexibilität des Boards zu gewährleisten um es universell einsetzbar zu machen. Nahezu sämtliche internen Konfigurationsparameter lassen sich verändern. Darunter auch fast alle Parameter für den RFM12 Transceiver. So lassen sich einerseits im „Plug & Play“-Verfahren Funksysteme aufbauen und zum anderen die Funksysteme nach belieben an die eigenen Bedürfnisse anpassen.

1.2 Eckdaten : Hardware

- Funkmodul RFM12 von HOPE RF
- Frequenzbereich: 433 MHz (ISM-Band) (868MHz Modul ebenfalls verwendbar)
- Geringe Abmessungen: 31mm x 33mm
- Anschluss- und Aufsteckmöglichkeit über 2 13-Pin-Stiftleisten im 2,54mm Raster
- ATmega32 Mikrocontroller mit 32kByte Flash und externem Quarz (ATmega16 möglich)
- 3 Schnittstellen zur Kommunikation: UART TTL / echte RS-232, I2C sowie SPI
  (Achtung, momentan nur UART lauffähig, SPI und I2C vorgesehen!)
- 1 herausgeführter I/O-Pin + 1 Lötjumper zur freien Verwendung
- Very Low Drop Festspannungsregler (wahlweise 3,3V oder 5V-Version) onBoard!
- Direkte Verbindung mit PC möglich durch MAX3221 RS-232-Treiber
- Breite Auswahl an UART-Baudraten durch austauschbares Quarz
- 3 LEDs zur Zustandsanzeige
- Großer Betriebsspannungsbereich von 3,2V - 5,4V[1]
- 42mA @ 5V / 21mA @ 3.3V / 50µA im Schlafmodus und ca. 15µA im Power-Down-Modus[2]
- SMA-Buchse für einen professionellen Antennenanschluss
- Reichweite bis zu einigen hundert Metern!
- Bidirektionale Funkverbindung (halbduplex): Ein Modul kann Senden und Empfangen!
- Clock Takt für externe Schaltung abgreifbar!
- Modularer Aufbau: Peripheriebauelemente können auch nicht bestückt werden
- u.v.m.

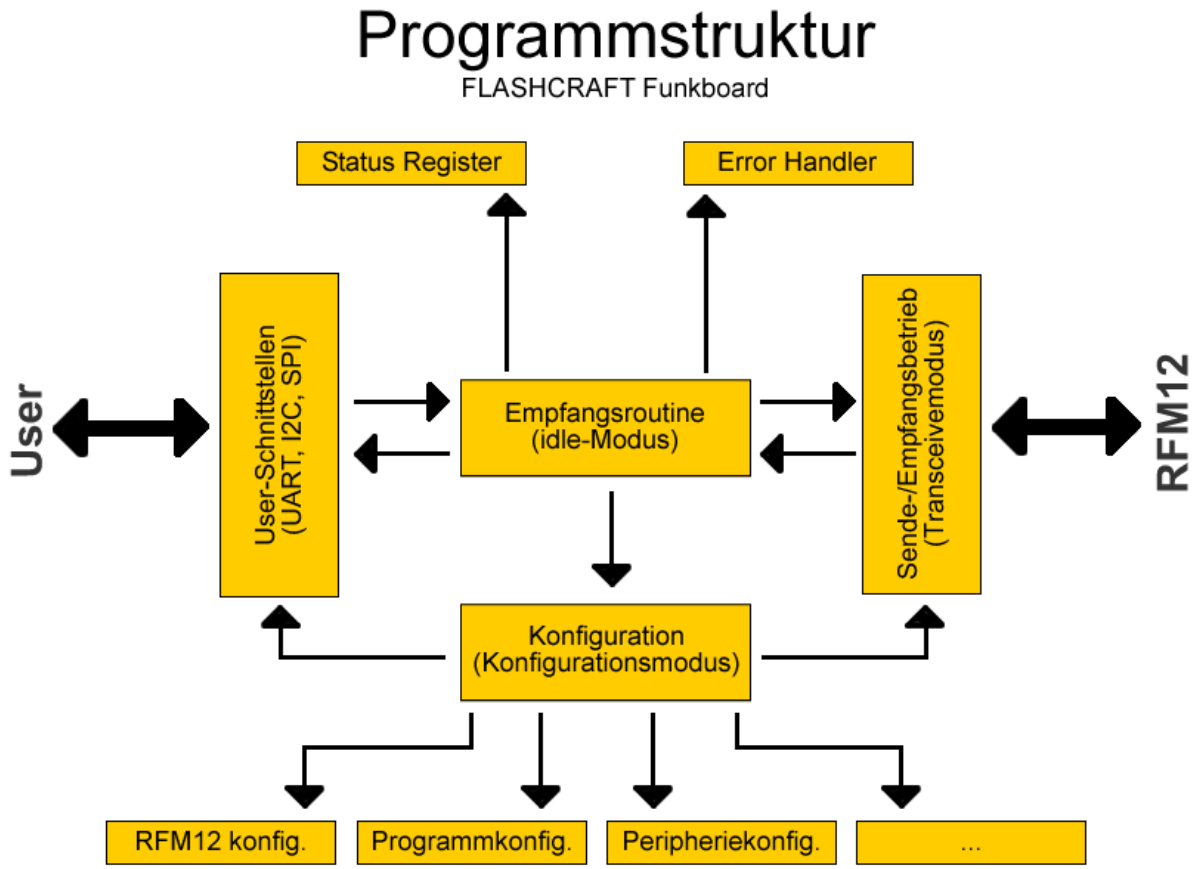
[1] Für den Betrieb bei 3,2V aufwärts ist ein ATmega32**L**-Typ notwendig!
[2] Werte gültig für ATmega32

1.3 Eckdaten : Software

- Open-Source: Der gesamte Code ist frei verfügbar
- Komplettes Programm in C und für AVRStudio mit dem GCC-Compiler ausgelegt
- Benutzerfreundlich: Ausführlich kommentierter und klar strukturierter Code
- Schnell: Zwischen Empfang und Ausgabe vergehen nur ca 1,5ms!
- Anpassungsfähig: Nahezu alle Systemparameter sind einstellbar und ohne erneutes Programmieren im laufenden Betrieb änder- und abspeicherbar
- Sicher: Startbytes, Identifikationsnummer, Längenprüfung, Timeouts, Checksummen und Acknowledgeprüfungen senken die Fehlerwahrscheinlichkeit enorm
- Funknetzwerk aus bis zu 125 Funkmodulen möglich!
- Daten können über 3 Schnittstellen plus einer weiteren I/O-Leitung in das Board übertragen oder aus diesem gelesen werden. Es kann zwischen I2C, SPI, oder UART/RS232 gewählt werden (bisher nur UART implementiert!)
- Zur Ansteuerung sind minimal nur 3 I/O-Leitungen nötig!
- Frei zuschaltbare Acknowledgementprüfung: Jeder einzelne Datenstring der gesendet wird kann auf Wunsch vom Empfänger quittiert werden
- Bis zu 70 Nutzdatenbytes auf einmal sendbar: Die Nutzdatenbytes können beliebige Konstellationen aufweisen. Die Länge der zu sendenden Nutzdatenbytes kann zwischen 1 und 70 varrieren. Der Wert eines jeden Bytes ist frei wählbar. Es gibt keine reservierten oder verbotene Konstellationen aus Nutzdatenbytes, alles ist erlaubt!
- Klare Trennung zwischen Konfigurationsmodus und Tranceive-Modus: Über den Zustand (High oder Low) einer I/O-Leitung wird dem Board signalisiert ob es sich um interne Konfigurationsdaten oder um Nutzdaten handelt, die gesendet werden sollen.
- Interne FIFO-Puffer variabler Größe: Das Programm arbeitet mit FIFO-Puffern, die vom Benutzer frei eingestellt werden können. Damit ist es möglich Daten zu einem späteren Zeitpunkt auszulesen oder mehr Nutzdaten zu einem Zeitpunkt dem Board zu übergeben als es momentan verarbeiten kann.
- Bequem: Statusabfrage, das Board verfügt über ein internes Statusregister, das die wichtigsten internen Meldungen sammelt und für den Nutzer abrufbar macht.
- Interrupt-Pin, der triggert, sobald neue Daten über Funk empfangen werden
- PC Terminalprogramm zum Testen und Konfigurieren
- u.v.m.

1.4 Modell der Softwareimplementierung

Um einen Überblick über die Software zu erlangen ist hier ein sehr einfach gehaltenes Prinzipschaltbild der Softwareimplementierung. Details wurden absichtlich ausgeblendet um das Bild übersichtlich zu halten.

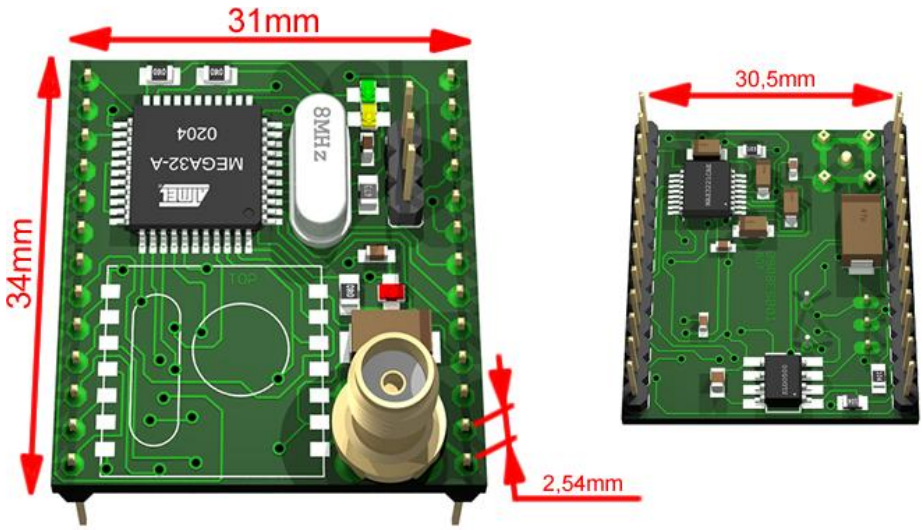


2. Detailierte Beschreibung aller Features

2.1 Echdaten: Abmessung, Grundlegendes

2.1.1 Abmessungen

Das Funkboard misst lediglich 31mm x 34mm und ist damit kleiner als eine handelsübliche Streichholzschachtel! An den Längsseiten hat das Funkboard 2 13-Pin Stiftleisten im 2,54mm Raster. Dadurch eignet es sich besonders zum Aufstecken auf andere Leiterplatten oder Steckbretter.



2.1.2 Funkmodul

Das FLASHCRAFT Funkboard basiert auf dem RFM12 von Hope RF. Es handelt sich hierbei um ein sehr preiswertes und leistungsfähiges Funkmodul mit vielen Funktionen. In diesem Funkboard werden fast alle Funktionen dieses kleinen Funkmoduls nutzbar gemacht, so dass das Modul seine volle Leistungsfähigkeit ausspielen kann. Die an vielen Stellen sehr komplexe und umfangreiche Interaktion und Handhabung des Moduls wird in diesem Funkboard **vollständig vor dem Anwender versteckt**. Es lohnt sich zwar einen Blick in das Datenblatt dieses RFM12 Moduls, **es ist jedoch nicht nötig das Modul zu verstehen um das Funkboard einsetzen zu können.**

2.1.3 Mikrocontroller

Das FLASHCRAFT Funkboard hat einen ATmega32 onBoard! Dieser leistungsfähige 8-Bit Mikrocontroller übernimmt die **komplette** Ansteuerung des RFM12 Moduls! Das hat gleich mehrere Vorteile: Der AVR kann seine vollen Ressourcen für eine möglichst effziente Ansteuerung und Handhabung des RFM12 Moduls einsetzen. Verbrauchte Interrupts, Timer und Flash haben keinerlei Auswirkungen für die externe Peripherie.
Diese externe Perihperie (zum Beispiel ein weiterer Mikrocontroller) werden in ihren Ressourcen damit außerdem enorm entlastet. So können diese Controller ihren wirklich wichtigen Aufgaben nachgehen ohne kostbare Zeit an der Funkübertragung zu vergeuden. **Selbst ein winziger ressourcenarmer ATTiny AVR kann damit Teil eines hocheffzienten Funknetzwerkes sein und dabei gleichzeitig noch anderen Aufgaben nachgehen!** Damit können kleinere Mikrocontroller zum Einsatz kommen was letztendlich wieder Geld und Platz spart.

2.1.4 Modularisierung

Das FLASHCRAFT Funkboard soll vor allem eine für Hobbyanwendungen durchaus semiprofessionelle Funklösung bereitstellen. Aus diesem Grund wurde das Funkboard mit zahlreichen ICs ausgestattet. Im Zentrum natürlich das **RFM12** Funkmodul und der **ATmega32**. Darüber hinaus ist ein **MAX3221** RS232-Driver und ein very low drop **ZLDO Festspannungsregler** vorhanden. Nutzt man die Eigenschaften all dieser ICs aus kann man auf externen Leiterkarten dazu noch Platz sparen.
Das Board ist jedoch so modular wie möglich gehalten. **Das heist, dass Baugruppen, die nicht benötigt werden auch nicht bestückt werden müssen!** Dazu zählen:

- MAX3221 inklusive dessen Kondensatoren
- ZLDO xxx Festspannungsregler
- Tantalkondensatoren zur Spannungsstabiliserung
- SMA-Buchse

2.1.5 Spannungstoleranzen

Das FLASHCRAFT Funkboard ist für den Betrieb mit 5V ausgelegt. Dies hat vor allem den Grund, dass hobbybetriebene Elektronikprojekte überwiegend mit 5V betrieben werden. Außerdem kann der ATmega32 unter dieser Spannung seine volle Leistungsfähigkeit entfalten. Es ist jedoch auch möglich das Funkboard unter 3,3V zu betreiben. Hierfür sind nur sehr wenige Änderungen oder Einschränkungen nötig.
Bis auf den ATmega32 ist die komplette Peripherie auf dem Board 100% 3,3V spannungstolerant. **Der ATmega32 kann mit 3,3V betrieben werden, wenn ein L-Typ (Atmega32L) verwendet wird.** Ist diese Bedingung erfüllt steht einem Betrieb mit 3,3V nichts mehr im Wege. Darüber hinaus lässt sich sogar der ZLDO-Spannungswandler entweder abschalten (falls es ein 5V-Typ ist) oder gegen einen 3,3V-Typ ersetzen.

2.1.6 Stromverbrauch

Es wurde darauf geachtet den Stromverbrauch des Funkboards so gering wie möglich zu halten um auch batteriebetriebenen Funklösungen ein langes Leben zu schenken.

- **Normalbetrieb** (ATmega32, kein L-Typ)

8MHz,	V_{CC} = 5V,	LEDs = An	I_{CC} = 44mA	(MAX3221 = Aus)
8MHz,	V_{CC} = 5V,	LEDs = Aus	I_{CC} = 38mA	(MAX3221 = Aus)
8MHz,	V_{CC} = 3,5V,	LEDs = An	I_{CC} = 23mA	(MAX3221 = Aus)
8MHz,	V_{CC} = 3,5V,	LEDs = Aus	I_{CC} = 20mA	(MAX3221 = Aus)

- **Schlafmodus**
  Im Schlafmodus (engl. Sleepmode) schlafen **ATmega32, RFM12 und MAX3221**. Der Stromverbrauch sinkt auf 70µA. Der Schlafmodus ist eine Art Standby. Konfigurationen und Daten gehen nicht verlohren, nach dem Aufwecken läuft das Funkboard mit den selben Einstellungen weiter die auch vor dem Einschlafen gesetzt waren.

- **Power-Down-Modus**
  Für maximale Stromersparnis gibt es einen Power-Down-Modus. Dieser Modus funktioniert jedoch nur, wenn das Funkboard über den ZLDO-Spannungswandler betrieben wird. Der Spannungswandler lässt sich durch den SC-Pin deaktivieren (SC auf high). Damit ist das komplette Funkboard spannungsfrei und es fließt ein Ruhestrom von lediglich etwa 11µA.

2.2 Funk

2.2.1 Frequenz

Die Funkfrequenz beträgt **433MHz**. Diese Frequenz im ISM-Band ist unter Einschränkungen (die das Funkmodul bereits „von Haus aus“ unterstützt) frei und ohne Lizenz nutzbar.

2.2.2 Reichweite

Die Reichweite hängt in erster Linie von den Räumlichkeiten und der Antennenform ab. Generell gilt jedoch dass Reichweiten von über 150m durchaus realisierbar sind. Mit speziellen Rundstrahl oder gar Richtantennen lässt sich diese Reichweite nochmals um ein Vielfaches vergrößern.

2.2.3 Antennenanschluss

Auf dem Funkboard gibt es einen SMA-Printanschluss für einen professionellen Anschluss einer SMA-Antenne. SMA ist neben BNC einer der am verbreitesten Antennensteckverbinder. Damit ist es kein Problem professionelle Fertig-Antennen zu montieren. SMA ist zudem eine **robuste** und damit für Hobbyanwendungen geeignete Schraubverbindung. Auch nach dutzendfachen De-/Montagen ist eine solche Buchse noch verwendbar.
Darüber hinaus besteht jedoch auch die Möglichkeit den Antennenanschluss über eine **externe Verdrahtung** zu führen. Die Antennensignalleitung ist an einem Pin der Stiftleisten abgreifbar. Damit ist es zum Beispiel möglich den Antennenanschluss über eine externe Leiterkarte auf einen anderen Steckverbinder zu führen oder gar eine PCB-Antenne zu realisieren. Das RFM12-Modul verfügt außerdem über eine automatische Antennenanpassung. So werden auch geringfügige „Baumängel“ an Antennen ausgeglichen ohne dass sich dies gleich massiv auf die Reichweite niederschlägt.

2.3 Schaltungstechnische Features

2.3.1 Festspannungsregler

Auf dem Funkboard kann ein Festspannungsregler montiert werden, der **ZLDO 500 (für 5V Betrieb)** bzw. der **ZLDO 330 (für 3,3V Betrieb)**. Der ZLDO eignet sich fabelhaft für stromsparende und batteriebetriebene Anwendungen. Sein interner Spannungsabfall beträgt lediglich ca. **8mV**! Auch die reglereigne Stromaufnahme ist mit ca. 500-700µA (je nach angelegter Spannung) sehr gering. Mit seinem Ausgang versorgt er nicht nur die Elektronik des Funkboards, ab Platinenrevision 1.1 kann man diese geregelte Ausgangsspannung auch auf externe Schaltungen nutzen.
Der Spannungsregler verfügt außerdem über einige weitere Besonderheiten:

- **Spannungsüberwachung**
  Sinkt die Eingangsspannung zu weit ab wechselt der Pegel der **LBF**-Pin auf low. Dies signalisiert, dass die Spannung zwar noch nicht zu niedrig ist, dies aber in naher Zukunft der Fall sein wird.

- **Übertemperaturüberwachung**
  Steigt die Temperatur im Regler auf über 125°C an deaktiviert er sich automatisch. Eine Überbeanspruchung endet damit nicht in einer Zerstörung!

- **Deaktivierung**
  Es ist auch möglich den Regler manuell über den SC-Pin (SC auf high) zu deaktivieren. Damit ist (bei verwendetem Regler) das komplette Funkboard spannungsfrei.

2.3.2 Debugginganschlüsse

Zur Modifikation und Weiterentwicklung des Boards ist es fast zwangshaftig nötig die intern auf dem Funkboard genutzten Signalleitungen des RFM12-Moduls mit Hilfe von Oszilloskopen oder Logic-Analyzern zu überwachen. Aus diesem Grund wurden alle wichtigen Signalleitungen auf den Stiftleisten abgreifbar gemacht (siehe Schaltplan). So ist auch ein Debugging leicht und jederzeit möglich.

2.3.3 Externe Taktleitung

Das RFM12 Modul bietet einen Anschluss an dem ein einstellbarer Takt abgegriffen werden kann. Diese Taktleitung wird auf dem Funkboard nicht genutzt und ist auf die Stiftleisten herausgeführt. **Mit diesem Takt kann beispielsweise ein externer Mikrocontroller betrieben werden**. Ein zusätzliches externes Quarz oder der meist ungenaue interne Oszillator wäre damit nicht nötig!
Die Taktrate ist über ein Konfigurationsregister änderbar. Folgende Werte sind möglich:

$$\mathbf{1}\boldsymbol{MHz}, \mathbf{1,25}\boldsymbol{MHz}, \mathbf{1,66}\boldsymbol{MHz}, \mathbf{2}\boldsymbol{MHz}, \mathbf{2,5}\boldsymbol{MHz}, \mathbf{3,33}\boldsymbol{MHz}, \mathbf{5}\boldsymbol{MHz}, \mathbf{10}\boldsymbol{MHz}$$

2.3.4 Austauschbares Quarz

Der ATmega32 ist mit einem externen Quarz getaktet. Besonders im Hinblick auf die Nutzung der UART ist dies wichtig. Die Software unterstützt die gängigsten Baudraten. Jedoch sind mit dem 8MHz Quarz mit dem der AVR standardmäßig ausgerüstet ist nicht alle Baudraten fehlerfrei erzielbar. Aus diesem Grund ist das Quarz leicht austauschbar. In der Software muss lediglich eine Zeile geändert werden um dem ATmega32 auf einen anderen Takt umzustellen.

2.3.5 Hardwarereset

Der **RESET des ATmega32** ist auf einen Pin der Stiftleisten herausgeführt. Damit ist es möglich den AVR hardwareseitig zu resetten. Zwar gibt es auch die Möglichkeit eines Softwareresets, wer jedoch die volle Kontrolle über das Funkboard haben möchte kann diese Möglichkeit nutzen. Ein Hardware-reset hat die selbe Wirkung wie ein kurzzeitiger Spannungsausfall über die gesamte Schaltung.

2.3.6 MAX3221 RS232-Driver

Das Funkboard hat einen RS232-Pegelwandler direkt onBoard! Damit kann es zum Beispiel direkt an einen Computer angeschlossen werden. Die externen Kondensatoren des MAX3221 sind so gewählt, dass diese sowohl für den Betrieb bei 3,3V als auch bei 5V geeignet sind!
Der MAX3221 bietet jedoch noch weitere Möglichkeiten:

- **Autoshutdown** (standardmäßig aktiviert)
  Findet über einen Zeitraum von 30s keine Aktivität über der RS232-UART statt deaktiviert sich der MAX3221 selbstständig um Strom zu sparen

- **Automatisches Deaktivieren** (standardmäßig aktiviert)
  Der AVR überprüft nach einem Reset automatisch ob die RS232-UART verwendet wird. Dies ist der Fall wenn RS232-valide Pegel am MAX3221 anliegen (z.B. wenn das Board mit dem PC verbunden ist). Wird die RS232-UART nicht verwendet deaktiviert der AVR automatisch den MAX3221. Hier muss nichts weiter vom Anwender berücksichtigt werden.

- **Manuelles deaktivieren**
  Zur vollen Kontrolle kann der MAX3221 auch manuell de-/aktiviert werden.

2.4 Schaltungstechnische Features

2.4.1 3 unterstützte Schnittstellen

Fast alle Funkmodule und Funkboards lassen sich über lediglich eine Schnittstelle ansteuern. Das FLASHCRAFT Funkboard ist anders - Hier stehen gleich 3 Schnittstellen zur Verfügung. Es lässt sich wahlweise über SPI, I2C oder UART ansteuern. Damit ist es nicht mehr nötig die externe Peripherie und ihre noch freien Schnittstellen auf das Funkboard anzupassen, vielmehr kann das Funkboard nun der externen Peripherie angepasst werden - Genau so wie man es sich auch wünscht. Alle 3 Schnittstellen sind fast vollkommen gleichberechtigt. Unterschiede gibt es nur dort, wo sie sich auf Grund des unterschiedlichen Aufbaus der Schnittstellen nicht vermeiden lassen. Durch die Master-Slave Typologie des SPI und I2C Busses ergeben sich gegenüber der UART-Schnittstelle kleine Einschränkungen. **Achtung: Bisher nur UART implementiert!**

2.4.2 Zwei getrennte Betriebsmodis

Der Umgang mit der Software des Funkboards ist grob in 2 Betriebsmodis einzuteilen. Beide Betriebsmodis sind vollkommen unabhängig vom jeweiligen anderen.

- **Transceivemodus**
  Der Transceivemodus (Kunstwort aus Transmit und Receive, also zu deutsch etwa **Sende-Empfangs-Modus**) ist der Modus in dem das Funkboard eingehende Daten über die Schnittstellen (I2C, SPI oder UART) per Funk zu anderen Funkboards überträgt. Die Übertragung verläuft äußerst zeit- und ressourceneffizient. Keine langen „Totzeiten“ zwischen Sende- und Empfangsbetrieb, interne FIFO Puffer, uvw. Alle detaillierteren Informationen inklusive einer kurzen Zusammenfassung sind dem ***Transceivemodus*** zu entnehmen.

- **Konfigurationsmodus**
  Im Konfigurationsmodus kann das Funkboard den eigenen Bedürfnissen angepasst werden. Es gibt dutzende Konfigurationsmöglichkeiten um das Funkboard nach belieben zu „personalisieren“. **Nahezu alle wichtigen Größen lassen sich hier ändern**. Von funkmodul-internen Einstellungen wie Kanalnummer, Bandbreite und RSSI über programminterne Größen wie FIFO Zwischenspeicher bis hin zu Schlafmodis, Softwareresets, Datenspeicherung etc. Alle detaillierteren Informationen inklusive einer kurzen Zusammenfassung sind dem ***Konfigurationsmodus*** zu entnehmen.

Der Vorteil dieser Methode ist, dass es keine reservierten Befehle, gesperrte Bytefolgengen etc. gibt. Im Transceivemodus kann jede Bytekonstellation über Funk übertragen werden. Gleichzeitig ist es bei einem korrekten Umgang praktisch nicht möglich unabsichtlich zwischen beiden Modis hin- und herzuwechseln. **Konfigurations- und Funkdaten sind damit immer strikt getrennt.**

Es ist auch möglich geänderte Konfigurationen dauerhaft zu speichern. Diese werden dann bei allen zukünftigen Resets und Neustarts **automatisch** geladen. Damit ist es im Normalfall ausreichend ein Funkboard **einmal** umzukonfigurieren.

2.4.3 Effizientes Sende- Empfangshandling

Viele andere Funkmodule weisen gerade im Empfangs- und Sendebetrieb teils große Einschnitte auf. Es gibt große Totzeiten, sehr kleine oder überhaupt keine internen Zwischenspeicher, langwierige Wechsel zwischen Senden und Empfangen, etc.. Damit ist bei diesem Funkboard Schluss. Zwar ist auch das FLASHCRAFT Funkboard wie fast alle anderen Funkmodule auch nur halbduplexfähig, das heist es kann immer nur gesendet oder empfangen werden, der Algorithmus dahinter ist jedoch effektiver als bei vielen anderen kommerziellen Produkten. Das FLASHCRAFT Funkboard spielt gerade hier seine volle Leistungsfähigkeit aus.
Während Daten über Funk empfangen werden können **gleichzeitig** Daten in das Funkmodul geschrieben werden. Zwischen dem Umschalten vom Empfangs- in den Sendebetrieb und umgekehrt vergehen keine 2 Mikrosekunden. Diese und noch viele weitere Eigenschaften sind die wahren Vorteile des Funkboards.

2.4.4 Sicherheitsfeatures

Die Datenübertragung wird durch zahlreiche Sicherheitsfeatures zuverlässiger gemacht. Sind alle diese Sicherheitsfeatures aktiv sinkt die Wahrscheinlichkeit eines Datenverlusts oder verfälschten Daten auf praktisch **0%**. **Alle diese Sicherheitsbarrieren sind vor dem Anwender versteckt**, er muss sich also keinerlei Gedanken um deren Funktion machen. Alle Sicherheitsmerkmale sind im ***Datenübertragung und Datensicherheit*** ausführlich erklärt.

2.4.5 Status Report

Das Funkboard führt ein internes Statusregister, das auf Wunsch vom Anwender abgerufen werden kann. Das Statusregister zeichnet die wichtigsten Zustände der Software auf. So lassen sich unter anderem der Zustand der FIFO Zwischenspeicher und des letzten aufgetretenen Programmfehlers überwachen. Der Status wird außerdem rein programmintern generiert und kann zu jeder Zeit abgerufen werden. Für die Generierung des Status muss keine kostbare Bandbreite zwischen dem AVR und dem RFM12 verschenkt werden.

2.4.6 Error Handler

Die Software des Funkboards zeichnet im laufenden Betrieb viele „bekannte“ Fehler auf und speichert den **letzten** vorgefallenen Fehler ab. Dieser Fehler kann dann im Status Register abgerufen werden. Wird der Status abgerufen wird auch danach der letzte vorgefallene Fehler **gelöscht**. So lässt sich ohne aufwendigeres Debugging auch feststellen in welchen Situationen die Software des Funkboards nicht korrekt arbeitet.
Diese Fehler könnten zum Beispiel leicht fehlerhafte Baudraten sein (Takt nicht ganz konsistent).

2.4.7 Funknetzwerk

Das FLASHCRAFT Funkboard unterstützt schon programmintern das Aufstellen eines ganzen Funknetzwerkes aus **bis zu 125 Funkboards**! So können ganze Gruppen aus Funkboards untereinander sehr effizient kommunizieren. Möglich macht dies eine Identifikationsnummer (ID), die jedes Funkboard hat und die vom Anwender frei wählbar ist. Jedes Datenpaket das gefunkt wird hat die ID des Zielfunkboards sowie des Senderfunkboards inne. Empfängt ein Funkboard ein solches Datenpaket und stimmen seine ID und die im Datenpaket überein nimmt es das Datenpaket an und weis zugleich von welchem Funkboard es gesendet wurde. Stimmen die IDs nicht überein bricht das Zielfunkboard den Datenempfang sofort ab ohne das Paket komplett abzurufen.

2.4.8 FIFO Puffer

Das Board verfügt über frei konfigurierbare Empfangs- und Sendepuffer die Daten temporär zwischenspeichern können. So ist es zum Beispiel möglich in „Stoßzeiten“ Daten schneller in das Funkmodul zu schreiben als es diese senden kann. Das Programm wird die zu sendenden Daten erst intern zwischenspeichern und dann Stück für Stück versenden. Umgekehrt werden alle empfangenen Daten ebenfalls zuerst in einem FIFO Puffer gespeichert. Dieser Puffer kann dann Stück für Stück ausgelesen werden.

2.4.9 Frei nutzbare I/O-Pins

Die Revision 1.0 der Platine unterstützt zwei, die Revision 1.1 einen frei konfigurierbaren I/O-Pin des ATmega32. Diese Pins werden vom Standardprogramm nicht verwendet und können vom Anwender selbst nach belieben konfiguriert werden.

2.4.10 Interrupt-Unterstützung

Das Funkboard bietet die Möglichkeit einen Ausgangspin des ATmega32 an einen Interruptpin eines externen Mikrocontrollers anzuschließen. Dieser Ausgangspin, der sogenannte **RECC-Pin** (für **Rec**eive-**C**omplete-Pin) gibt einen ..µs breiten positiven Spike aus sobald valide Daten über Funk empfangen werden. Ein externer Mikrocontroller kann auf dieses Signal triggern und sofort reagieren wenn neue Daten verfügbar sind.

2.4.11 In Circuit Programming (ISP) Programmiermodus

Der ATmega32 lässt sich direkt in der Schaltung durch den ISP-Programmiermodus programmieren. Alle hierfür notwendigen Pins wurden (direkt nebeneinander) auf die Stiftleisten abgreifbar gemacht.

2.4.12 Funkverzögerung

Das FLASHCRAFT Funkboard zeichnet sich außerdem durch eine sehr kleine zeitliche Verzögerung zwischen gesendeten- und empfangenen Daten aus. Während andere Funkmodule oft eine zeitliche Verzögerung von bis zu 20ms und mehr haben liegt diese beim FLASHCRAFT Funkboard bei nur wenigen einzelnen Millisekunden. Werden also Bytes zur Funkübertragung in ein Board hineingeschrieben vergeht nach dem Empfang des letzten Bytes beim Sender und dem Empfang des ersten Bytes beim Empfänger typischerweise nur etwa 1,8-8ms Zeit.
Diese sogenannte Funkverzögerungszeit ist nicht konstant. **Sie hängt von der Datenrate ab mit der einzelne Bytes in den Sender geschrieben werden**. Der Grund hierfür liegt darin, dass der Sender erst eine Übertragungsdauer eines einzelnen Bytes abwartet bevor er die bis dato bei ihm eingegangenen Bytes über Funk versendet. Umso höher die Datenrate (zum Beispiel der UART) umso kürzer ist die Übertragungsdauer eines Bytes und umso kürzer die daraus resultierende Funkverzögerungszeit. Generell kann die Funkverzögerungszeit aus folgender Formel berechnet werden:

$$\boldsymbol{Funkverzögerungszeit} = \boldsymbol{Übertragungsdauer\ eines\ Bytes} + \mathbf{1,7}\boldsymbol{ms}$$

Beispiel:
Wird die UART mit einer Baudrate von 38400 Baud verwendet ist die Übertragungsdauer eines Bytes 260µs lang. Die Funkverzögerungszeit beträgt damit 1700µs + 260µs = 1960µs, also ca. 2ms.

Alles weitere hierzu ist im ***Transceivemodus*** erläutert.

2.5 Quellcode

2.5.1 Allgemeines

Der komplette Code für den ATmega32 ist in C geschrieben und auf dem Compiler AVRStudio entwickelt worden. Er ist komplett frei verfügbar und modifizierbar (unter den gegebenen Lizenzbedingungen, siehe ***Lizenz und Weitergabe***). Der Quellcode und die fertig compilierten Hexfiles können auf meiner Webseite www.flashcraft.de heruntergeladen werden.

2.5.2 Dokumentation

Der Quellcode ist selbstdokumentierend und sehr ausführlich in englischer Sprache kommentiert. Eine zusätzliche Dokumentation gibt es daher nicht.

2.5.3 Vereinbarungen

Der Quellcode wurde so modular wie möglich aufgebaut. Es wurde darauf geachtet den Code leicht erweiterbar und leicht modifizierbar zu machen. Alle wichtigen Parameter wie Taktfrequenz, Pinbelegung, etc. sind durch Makros umeditierbar. Will man beispielsweise seine eigene Platine mit abgeänderter Pinbelegung erstellen so ist die Pinbelegung im Programm durch Makros sehr leicht änderbar. Auch die Taktfrequenz mit all ihren Folgewirkungen kann man in nur einem Makro umändern.
Um eine zu tiefe Verschachtelung von Funktionen zu umgehen wurde reger Gebrauch von globalen Variablen gemacht. Dies widerspricht zwar der gängigen Programmierpraxis macht den Code jedoch um ein vielfaches verständlicher und übersichtlicher. Durch den gesamten Quellcode zieht sich außerdem eine einheitliche Namensgebung von Funktionen, Variablen und Makros.
Es wird trotz allem unumgänglich sein sich bei einer Umeditierung oder Weiterentwicklung der Software vorher intensiv in diese hineinzuarbeiten.

2.5.4 Verwendete Bibliotheken

Es wurden 2 externe Bibliotheken in diesen Code mit eingebunden. Zum einen ist das die **UART-library von Peter Fleury**, zum anderen die **TWISLAVE-library von Uwe Große- Wortmann**. Für diese Bibliotheken gelten ausdrücklich die Rechte und Vorschriften der jeweiligen Autoren.

2.5.5 AVRStudio Entwicklung

Der gesamte C-Code wurde mit AVRStudio Version 4.13 Built 528 von Atmel entwickelt. Es wurde dabei reger Gebrauch von AVRStudio-eigenen Bibliotheken gemacht. Es kann daher zu Problemen führen wenn der Quellcode unter einer anderen Entwicklungsumgebung weiterentwickelt wird.

2.5.6 Hex- und Binärcode

Auf meiner Webseite www.flashcraft.de kann man die aktuellsten Hex- und Binärdateien für den ATmega32 des FLASHCRAFT Funkboards downloaden. Diese Dateien sind komplett eigenständig und lassen sich sofort in den ATmega32 flashen (zum Beispiel mit USBprog). Der AVR ist wie gewohnt über ISP programmierbar.

2.6 Sonstiges

2.6.1 LEDs

Das FLASHCRAFT Funkboard verfügt über 3 LEDs:

- **Power-Good-LED:**
  Diese rote LED leuchtet sobald das Funkboard seinen normalen Zustand nach dem booten eingenommen hat. Während der Initialisierung nach einem Reset oder Einschalten sowie nach dem Aufwecken aus dem Schlafmodus blinkt die LED einige Male schnell hintereinander.

- **Receive-Complete-LED:**
  Diese gelbe LED blinkt kurz sobald ein valides Datenpaket über Funk empfangen wurde.

- **Transmitting-LED:**
  Diese grüne LED leuchtet solange ein Datenpaket über Funk gesendet wird. Da das Senden von Daten maximal nur einige Millisekunden dauert leuchtet auch diese LED nicht länger auf. In der Praxis blitzt die LED dadurch immer nur kurz (aber sichtbar!) auf.

Die LEDs sind außerdem softwaremäßig abschaltbar. Werden sie also nicht genutzt können sie abgeschaltet werden um Strom zu sparen.

2.6.2 Abblock- und Stabilisierungskondensatoren

Das Funkboard ist großzügig mit Abblock- und Spannungsstabilisierungskondensatoren ausgestattet. Auch hier können externe Bauelemente eingespart werden, da keine großen Elektrolyt-kondensatoren zur Spannungsstabilisierung benötigt werden. Auf dem Funkboard ist Platz für 3 kleine Keramik-Abblockkondensatoren vorgesehen, die **in jedem Fall bestückt werden sollten**. Außerdem ist Platz für 2 große SMD-Tantalkondensatoren vorhanden. Diese Kondensatoren können, müssen jedoch nicht bestückt werden.

2.7 PC Terminalprogramm

2.7.1 Allgemeines

Zum Projekt gehört außerdem ein PC Terminalprogramm, das auf Rechnern mit Windows XP und höher lauffähig ist. Wird das Funkboard über die RS232-UART mit dem PC verbunden kann dieses kleine Programm genutzt werden, um das Funkboard zu testen und zu konfigurieren – Ganz bequem am PC, ohne sich extra eine Mikrocontrollersteuerung oder ein eigenes Programm hierfür schreiben zu müssen. Notwendig ist nur eine kleine Adapterplatine. Siehe dazu auch ***PC Terminalprogramm.***

3.Schnittstellen

3.1 Allgemeines

Das Funkboard soll sich am Ende über 3 Schnittstellen ansteuern lassen: SPI, I2C und UART. Im derzeitigen Entwicklungsstadium ist **nur die UART** (RS232 und TLL) implementiert. Der folgende Text bezieht sich, wenn nicht weiter beschrieben auf die fertige Umsetzung aller 3 Schnittstellen.

Alle 3 Schnittstellen sind bis auf wenige Ausnahmen absolut gleichberechtigt, das heist mit allen Schnittstellen lässt sich eine Funklösung auf (fast) dieselbe Weise umsetzen. Die Unterschiede werden in einem extra Unterabschnitt behandelt.
Die grundsätzliche Ansteuerung des Funkboards ist bei allen 3 Schnittstellen identisch. Allein die Art wie die Informationen in das Funkboard geschrieben und aus diesem gelesen werden unterscheidet sich. **Ein Funkboard lässt sich jedoch immer nur mit einer Schnittstelle ansteuern.** Es ist nicht möglich im Betrieb die Schnittstelle zu wechseln oder das Funkboard mit mehreren Schnittstellen gleichzeitig anzusteuern. Dies soll in erster Linie helfen Kollisionen zwischen 2 oder mehr Schnittstellen zu verhindern. Außerdem wäre es ein enormer zusätzlicher Zeit- und Ressourcenaufwand wenn der AVR alle 3 Schnittstellen zeitgleich handhaben müsste.

Genrell gilt: Nach einem Reset oder dem Anschalten der Spannungsversorgung überwacht das Funkboard alle Schnittstellen. Die Schnittstelle, durch die zuerst valide Daten in das Funkmodul geschrieben werden wird für den Rest der Laufzeit (das heist bis zum nächsten Reset oder Abschalten des Boards) zur verwendeten Schnittstelle erklärt. Die beiden anderen Schnittstellen werden daraufhin deaktiviert. Dummybytes sind nicht notwendig! Es können sofort Nutzdaten gesendet werden.
Ein Beispiel: Das Funkboard wird mit Spannung versorgt. Nach einer Sekunde wird über die UART ein zwei Byte langer Konfigurationsbefehl in das Board geschrieben. Dieses empfängt die 2 Bytes, erklärt die UART fortan zur verwendeten Schnittstelle und deaktiviert I2C und SPI. Danach führt die Software den Konfigurationsbefehl aus. Würde man nun versuchen Daten über I2C oder SPI in das Board zu schreiben wäre dies wirkungslos. Als erste Bytes könnte man aber genausogut Bytes für den Transceivemodus senden.

3.2 Die UART-Schnittstelle

3.2.1 Allgemeines

Die UART-Schnittstelle ist die favorisierte Schnittstelle des Funkboards. Sie bietet den sehr großen Vorteil gegenüber SPI und I2C, dass das Funkboard selbstständig in der Lage ist ohne Aufforderung durch einen externen Mikrocontroller oder PC empfangene Funkdaten über die UART an diese xterne Peripherie zu senden. Aufgrund der Master-Slave-Definitionen von I2C und SPI ist dies bei diesen Schnittstellen **nicht möglich**.
Die UART-Schnittstelle kann sowohl mit **TTL-/CMOS-Pegeln**[3] als auch mit **„echten RS232“-**Pegeln (nötig beim direkten Anschluss an einen PC) betrieben werden.
Die beiden Arten über die die UART betrieben werden kann unterscheiden sich ausschließlich nur durch die verwendeten Spannungspegel und sind sonst absolut identisch.
Es kann jedoch zu einer Zeit immer nur der Betrieb mit einem Pegel erfolgen. Das heist der Betrieb mit TTL-/CMOS- und RS232-Pegeln zur selben Zeit ist nicht möglich!

3.2.2 Automatische Pegelerkennung

Die Software des Funkboards ist in der Lage nach jedem Reset bzw. Neustart (z.B. nach Power-Down) selbstständig zu erkennen, welchen Betrieb der Nutzer fokusiert, das heist ob TTL-/CMOS-Pegel oder RS232-Pegel verwendet werden und konfiguriert sich automatisch.
Der Anwender muss den MAX3221 in der Regel also nicht selbst zu- oder abschalten um damit den ewünschten Pegel zu aktivieren.
Hierfür wird einmalig bei der Start-Initialisierung des AVRs der Status des MAX3221 ausgelesen und dieser dementsprechend aktiviert oder deaktiviert. Daher ist es sehr wichtig, dass ca. 0,5 Sekunden nach dem Reset in der Initialisierungsphase des Funkboards RS232-konforme Spannungspegel an dem Pin RX232 anliegen falls die UART über RS232-Pegel
genutzt wird. Ansonsten deaktivert die Software den MAX3221 automatisch und die TTL-/CMOS-Pins RX und TX sind aktiv.

3.2.3 Betrieb mit TTL-/CMOS-Pegeln

Wird die UART mit TTL-/CMOS-Pegeln betrieben werden (in der Minimalbeschaltung) neben dem MDSEL-Pin nur noch die **Pins RX und TX** des Moduls benötigt. Über RX werden Daten in das Modul gesendet, über TX aus dem Modul gelesen.
Wichtig: Beim Betrieb mit TTL-/CMOS-Pegeln muss der RS232-Treiberbaustein MAX3221 deaktiviert werden. Dies geschieht in der Regel automatisch (siehe auch ***Automatische Pegelerkennung***).

3.2.4 Betrieb mit RS232-Pegeln

Wird die UART mit „echten RS232“-Pegeln betrieben werden (in der Minimalbeschaltung) neben dem MDSEL-Pin nur noch die **Pins RX232 und TX232** des Moduls benötigt. Über RX232 werden Daten in das Modul gesendet, über TX232 aus dem Modul gelesen.
Wichtig: Beim Betrieb mit „echten RS232“-Pegeln muss der MAX3221 aktiviert sein. Dies geschieht in der Regel automatisch (siehe auch nachfolgenden ***Automatische Pegelerkennung***)

[3] Der Spannungspegel hängt davon ab mit welcher Spannung das Funkboard betrieben wird. Im 5V-Betrieb arbeitet das Modul mit TTL-Pegeln, im 3,3V-Betrieb mit CMOS-Pegeln (3,3V-Pegeln)

3.2.5 Kodierung

Die Kodierung der UART-Schnittstelle ist fest eingestellt auf die typische 8N1 Kodierung, das heist: 1 Startbit, 8 Datenbits, 1 Stoppbit, keine Parität.

3.2.6 Baudraten

Die verwendete Baudrate hängt maßgeblich vom verwendeten Quarz ab. Das heist das Quarz mit dem der ATmega32 getaktet wird gibt vor welche Baudraten verwendet werden können. Umgekehrt gibt eine gewünschte Baudrate vor welche Quarze verwendbar sind.
Im ATmega32 können alle gängigen und weniger gängigen Baudraten für die UART-Schnittstelle über den Konfigurationsmodus im laufenden Betrieb geändert werden. Dabei gibt es jedoch keine interne Kontrolle ob sich das verwendete Quarz überhaupt für die gewünschte Baudrate eignet. Die Kontrolle hierfür muss der Anwender tragen!

Die möglichen Baudraten sind:

- 1200 Baud (Bitdauer 833,0µs)
- 4800 Baud (Bitdauer 208,0µs)
- 9600 Baud (Bitdauer 104,0µs)
- 14400 Baud (Bitdauer 69,4µs)
- 19200 Baud (Bitdauer 52,1µs)
- 28800 Baud (Bitdauer 34,7µs)
- 38400 Baud (Bitdauer 26,0µs)
- 57600 Baud (Bitdauer 17,4µs)
- 76800 Baud (Bitdauer 13,0µs)
- 115200 Baud (Bitdauer 8,7µs)
- 230400 Baud (Bitdauer 4,3µs)
- 250000 Baud (Bitdauer 4,0µs)

Die verwendete Baudrate wird über das Konfigurationsregister festgelegt.

3.3 Die I2C-Schnittstelle

Diese Schnittstelle ist in der aktuellen Version noch nicht implementiert, eine spätere Implementierung ist aber vorgesehen!

3.4 Die SPI-Schnittstelle

Diese Schnittstelle ist in der aktuellen Version noch nicht implementiert, eine spätere Implementierung ist aber vorgesehen!

4. Transceivemodus

4.1 Allgemeines

Im Transceivemodus werden Daten, die über eine der verfügbaren Schnittstellen eingehen gefunkt und über Funk eingehende Daten empfangen. Die Bedienung im Transceivemodus ist sehr simpel. Prinzipiell gilt für alle Schnittstellen nahezu das selbe:

1. Um im Transceivemodus zu sein muss der MDSEL-Pin auf **low** (logisch 0) gezogen werden.

2. Die Länge der Nutzdatenbytes, also der Bytes die benutzerrelevante Informationen tragen muss zwischen 1 und 50 liegen.

3. Die Übertragung ist als 1:1 Übertragung realisiert. Das heist ein Datenpaket welches über ein Funkboard gesendet wird, wird in einem anderen Funkboard genau so auch wieder ausgegeben.

4. Es gibt keine fixe Paketlänge, es können Nutzdaten unterschiedlicher Länge versandt werden.

5. Alle Datenpakete die einem Funkboard zur Funkübertragung über eine Schnittstelle übermittelt werden müssen den folgenden Aufbau haben (niedrigere Bytenummer bedeutet zeitlich vorher übertragen):

Bytenummer	**0**	**1**	**2**	**3**	...	n+2
Bezeichnung	ID Empfänger	Acknowledge verlangen?	Datenbyte 0	Datenbyte 1	...	Datenbyte n
Beispiel	0x02	0x0F	'H'	'u'	...	'm'

   a. **Byte 0** muss immer die **Empfänger-ID** tragen an die der String gesendet wird. Siehe auch ***Datenübertragung und Datensicherheit***
   b. **Byte 1** trägt die Information ob ein **Acknowledge** verlangt wird oder nicht.
   c. **Byte 2 bis Byte n+2** enthalten die **Daten**, die der Benutzer versenden will.

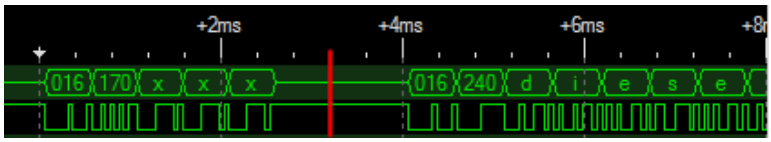
6. Ein eingehender Datenstring wird im Funkboard als solches erkannt wenn alle einzelnen Bytes **unmittelbar** aufeinander folgen. Bei einer Pause von **minimal einer Bytelänge** oder höher (Zeit hängt ab von Übertragungsrate/Baudrate) wird aus den bis dahin eingehenden Bytes ein Paket gebildet, welches dann über Funk übertragen wird. Alle Bytes die **nach** dieser Pause eingehen bilden einen neuen String der bei einer weiteren Pause ein neues Paket ergibt usw.

7. Im Transceivemodus stehen ein **Inputbuffer** sowie ein **Outputbuffer** variabler Größe zur Verfügung, die Daten temporär zwischenspeichern können. Siehe dazu auch ***Datenpufferung / FIFO-Puffer*** für genauere Informationen zur Funktionsweise.

8. Die Daten werden intern mit zahlreichen Sicherheitsmerkmalen geschützt. Alle Sicherheitsfeatures funktionieren komplett intern. Der Anwender muss sich darüber keinerlei Gedanken machen. Siehe auch ***Datenübertragung und Datensicherheit***

Nachtrag zu Punkt 6:

Ein Beispiel:
Bei der UART mit **19200 Baud** beträgt die Bitrate 52µs. Bei einer 8N1-Codierung sind für ein Datenbyte 10 Bits notwendig. Damit ist ein Byte 520µs lang. Sollen nacheinander die ASCII-Strings „xxx“ und „dieser“ (inklusive Target-ID und Acknowledge) als **2 separate Pakete** übertragen werden sieht das Zeitdiagramm für die UART folgendermaßen aus:



Die vertikale rote Linie markiert den Zeitpunkt bis zu dem weitere eingehende Zeichen/Bytes noch zum aktuellen Paket (hier zu dem String „xxx“) hinzugefügt werden. Alle Bytes die danach eingehen (also später als 520µs nachdem das letzte Byte vollständig eingegangen ist) werden in ein neues Paket gespeichert.

4.2 Funktionsweise

Der Transceivemodus läuft für alle Schnittstellen einheitlich immer gleich ab. **Ist der Transceive-modus aktiv wartet das Funkboard in einer prinzipiellen Endlosschleife auf eingehende Funkdaten.** Dieser Wartezustand wird nur (für vernachlässigbar kurze) Zeit unterbrochen, wenn neue Daten in das Funkboard geschrieben werden (entweder im Transceive- oder Konfigurationsmodus) oder ein Datenpaket über Funk versendet wird.
Abgesehen von dieser Unterbrechung ist das Board also ständig bereit eingehende Daten über Funk zu empfangen.
Der Wechsel zwischen Empfang (also Warten in der Endlosschleife) und Senden (also Annahme von Daten über I2C, SPI oder UART und anschließendem Funken) geht sehr schnell und dauert nur wenige Mikrosekunden.

Der prinzipielle Vorgang läuft folgendermaßen ab:

1. Das Funkboard befindet sich im „Leerlauf“ ständig im Empfangsmodus und geht auch immer wieder selbstständig in diesen Modus zurück.

2. Geht ein Byte über I2C, SPI oder UART ein löst erst das **vollständig empfangene Byte** einen Interrupt aus. Beispiel: Wird die UART mit 19.200 Baud verwendet beträgt die Bitrate 52µs. Wird nun ein Zeichen übertragen (bestehend aus 10 Bits, da 8N1-Codierung) springt das Programm erst nach den vollen 520µs in den Interrupt. Die Zeit in der bereits Bits über die UART empfangen werden, das Byte aber noch unvollständig ist, verbringt das Programm im Empfangsmodus.

3. Ist ein Byte vollständig über eine der Schnittstellen empfangen worden springt das Programm kurz (für wenige hundert Nanosekunden oder wenige Mikrosekunden, genaue Zeit varriert mit verwendetem Quarz) aus dem Empfangsmodus in den Interrupt und kopiert das Byte dort in den Output-Buffer. Danach springt das Programm sofort wieder in den „Leerlauf“ bzw. in den Empfangsmodus. In diesem Prozess kann es zu keinem Datenverlust bei eingehenden Funkdaten führen. **Die Zeitspanne die das Programm außerhalb des Empfangsmodus verbringt ist hierfür in jedem Fall zu kurz.**
4. Der Prozess unter Punkt 3 wiederholt sich so lange bis es zu einer Pause kommt, die im vorigen Abschnitt beschrieben ist.

5. Ist diese Pausenzeit überschritten und kein Byte bis dato eingegangen geht das Programm davon aus, dass ein vollständiges Datenpaket zur Funkübertragung vorliegt. Nun springt es erneut aus dem Empfangsmodus und initialisiert den Sendemodus. Die Datenbytes werden aufbereitet und gesendet. Dieser Vorgang kann einige Millisekunden dauern und varriert mit der Anzahl der Datenbytes, der Taktrate des AVRs und der Funk-Baudrate des RFM12-Moduls. Wichtig: **Dies ist der einzige Punkt, bei dem eingehende Funkdaten nicht empfangen werden können.** Während dieser Zeitspanne kann es also passieren, dass es zu Datenverlust kommt.

6. Anschließend schaltet das Modul sofort wieder in den Empfangsmodus und wartet erneut auf eingehende Funkdaten, entweder ein Acknowledge, falls verlangt oder ein Datenpaket.

7. Werden Daten nun über Funk empfangen werden diese in Echtzeit, das heist direkt nach dem Empfang auf Gültigkeit überprüft (siehe dazu auch ***Empfangsbetrieb***) Wichtig: Während diesem Prozess ist das Funkboard stets bereit eingehende Daten über eine der Schnittstellen aufzunehmen. Ein vollständig eingegangenes Paket wird jedoch erst versendet oder ausgewertet wenn keine Datenpakete mehr über Funk empfangen werden.

8. Ist ein Paket vollständig über Funk empfangen worden hängt alles Weitere von der aktiven Schnittstelle ab. Dies wird im Folgenden beschrieben.

Hier erkennt man die wahre Performance dieser Funklösung:

- Es gibt keine langen “Totzeiten“ zwischen dem Wechsel von Sende- und Empfangsbetrieb.
- Daten können zu jeder Zeit in das Funkboard geschrieben werden.
- Das Programm ist, bis auf die Zeitspanne in der Daten über Funk gesendet werden, immer bereit Daten über Funk zu empfangen.

4.2.1 Besonderheiten der UART

Die Besonderheit der UART (und zugleich der größte Vorteil gegenüber I2C und SPI) liegt darin, dass das Funkboard selbstständig empfangene Funkdaten an die externe Peripherie senden kann ohne einen expliziten Befehl von dieser erhalten zu müssen. Damit ist es nicht nötig, das Modul regelmäßig auf seinen Status abzufragen. Dieser Modus ist standardmäßig vorkonfiguriert.
Ist dies jedoch explizit erwünscht kann das Funkboard auch in den sogenannten „Pollmodus“ übergehen. **Dieser ist bei Verwendung der SPI und I2C Standard**, bei der UART ist er als optionale Konfigurationsmöglichkeit vorhanden. In diesem Modus muss die externe Perihperie einen Konfigurationsbefehl senden, damit das Funkboard ein (genauer das zuerst eingegangene Paket, bzw. das, welches ganz oben auf dem FIFO-Puffer liegt) Datenpaket an diese sendet.

4.2.2 Besonderheiten der SPI

Diese Schnittstelle ist in der aktuellen Version noch nicht implementiert, eine spätere Implementierung ist aber vorgesehen!

4.2.3 Besonderheiten des I2C

Diese Schnittstelle ist in der aktuellen Version noch nicht implementiert, eine spätere Implementierung ist aber vorgesehen!

4.3 Datenpufferung / FIFO-Puffer

4.3.1 Allgemeines

Die Software des Funkboards arbeitet mit FIFO (First in first out) Zwischenspeichern sowohl für ausgehende als auch für eingehende Funkdaten. Dies hat den unschätzbaren Vorteil, dass Daten temporär, zum Beispiel in Zeiten mit einem sehr hohen Datenaufkommen, gespeichert werden können und somit ein Datenverlust umgangen werden kann.
Die Größe beider Puffer ist variabel und kann im Konfigurationsmodus angepasst werden.

4.3.2 Funktionsprinzip

Beide Puffer arbeiten nach dem exakt selben Prinzip gemäß untenstehender Skizzen.

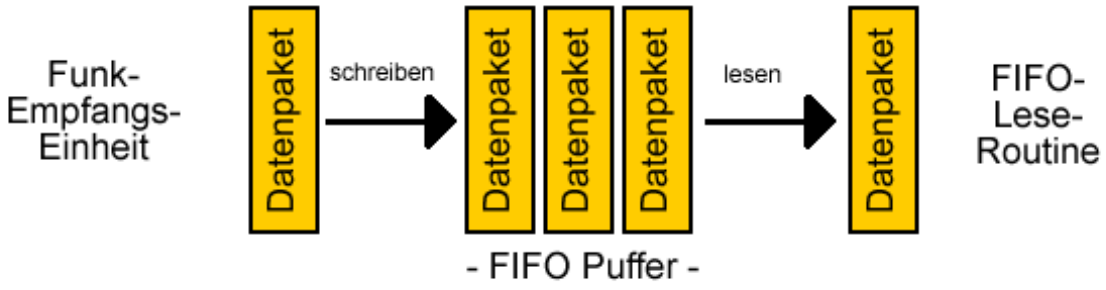


Abbildung 1 : Schematisierte Darstellung des Eingangspuffers

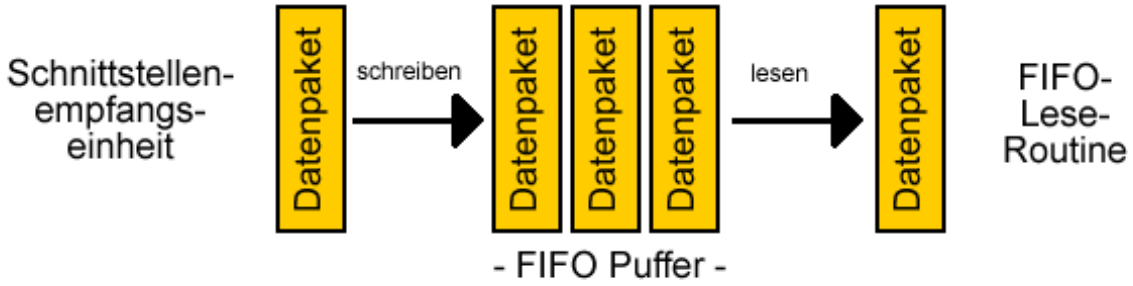


Abbildung 2 : Schematisierte Darstellung des Ausgangspuffers

Sobald die Funkempfangseinheit bzw. Schnittstellenempfangseinheit ein Datenpaket vollständig empfangen und überprüft hat speichert sie es im jeweiligen FIFO Puffer ganz unten ab. Die FIFO-Leseroutine (zum Beispiel durch den Pollbefehl angestoßen oder automatisch laufend) liest das oberste Paket im Puffer aus und entfernt es anschließend aus dem Puffer, so dass das nachfolgende Paket nachrückt.

Achtung: Die FIFO-Puffer arbeiten mit ganzen Datenpaketen, nicht mit einzelnen Bytes. Ein FIFO-Puffer der Größe 10 kann daher 10 Datenpakete zwischenspeichern. Diese Datenpakete wiederum können zwischen 1 und 50 Bytes groß sein. Die Größe (in Bytes) der einzelnen Datenpakete spielt dabei für die FIFO-Puffer jedoch keine Rolle.

4.3.3 Pufferüberlauf

Die Größe der FIFO-Puffer ist begrenzt und natürlich nicht unendlich tief. Mit jedem empfangenen Datenpaket wird der FIFO-Puffer ein Stück voller. Wird nun beispielsweise beim Empfang von Daten die Datenpakete langsamer aus dem Funkboard gelesen als dieses Daten über Funk empfängt kommt es irgendwann zu einem **Pufferüberlauf**.
Dabei stößt die Schreibroutine über die Obergrenze des Puffers hinaus – Ein Datenverlust folgt. Die Schreibroutine überschreibt nun das älteste im FIFO-Puffer verbleibende Datenpaket. Mit einem erneuten Schreibzugriff danach das zweitälteste usw.
Dieses Phänomen kann auch bei ausgehenden Daten auftreten, ist dort jedoch sehr viel schwerer zu erzielen.

Werden eingehende Daten zu schnell (das heist in einer zu hohen Datenrate) in das Funkboard geschrieben hat das Board nicht mehr genügend Zeit um diese Daten zu versenden und den Puffer zu leeren. Über kurz oder lang kommt es auch hier zu einem Pufferüberlauf.
Um einen Pufferüberlauf zu erkennen kann man den Inhalt des Status-Registers auslesen. Es enthält Bits, das den jeweiligen Zustand der beiden Puffer anzeigt. Mehr dazu im ***Status Register***.

4.3.4 FIFO Inputbuffer

Der FIFO Inputbuffer bzw. Empfangspuffer speichert eingehende Funkdaten paketweise in einen Zwischenspeicher ab, der nach und nach ausgelesen wird. Wird SPI, I2C oder die UART mit aktivem Pollmodus verwendet werden die Daten in diesem Zwischenspeicher nur nach einem Pollbefehl Paket für Paket ausgegeben (siehe auch: ***Transceivemodus***). Wird die UART mit inaktivem Pollmodus verwendet wird der FIFO Inputbuffer **automatisch** geleert.

4.3.5 FIFO Outputbuffer

Der FIFO Outputbuffer bzw. Sendepuffer speichert eingehende Daten, die über eine der Schnittstellen (SPI, I2C oder UART) eingegangen sind paketweise in einem Zwischenspeicher ab. Der Inhalt wird dann nach und nach per Funk gesendet.

4.4 Pollmodus

Der Pollmodus ist bei SPI- und I2C-Schnittstelle der einzige Betriebsmodus, bei der UART-Schnittstelle kann dieser Modus über den Konfigurationsmodus erzwungen werden. Im Pollmodus sendet das Funkboard nicht automatisch über Funk empfangene Datenpakete. Stattdessen speichert es die Datenpakete im FIFO Inputbuffer. In diesem Modus muss der externe Mikrocontroller oder Computer einen bestimmten Konfigurationsbefehl senden, damit das Funkboard ein (genauer das zuerst eingegangene Paket, bzw. das, welches ganz oben auf dem FIFO-Puffer liegt) Datenpaket an diesen sendet. Alle weiteren Informationen sind dem Konfigurationsregister zu entnehmen.

Es empfiehlt sich bei Verwendung der UART den Pollmodus nicht zu verwenden. Die beste Performance lässt sich erzielen wenn man an dieser Stelle das Funkboard diese Arbeit übernehmen lässt.

4.5 Verwendung des MDSEL-Pins im Transceivemodus

Achtung: Es ist unbedingt auf die richtige Handhabung des MDSEL-Pins zu achten! Eingehende Bytes werden dann, **und nur dann**, als Daten zur Funkübertragung für den Transceivemodus erkannt, wenn der MDSEL-Pin spätestens mit dem Empfangen des letzten Bits eines jeden einzelnen Bytes auf low geht. Besonders bei Verwendung der UART ist hier Vorsicht geboten!

Ein Beispiel:
Wir wollen Nutzdaten zur Funkübertragung mittels der UART in das Funkboard schreiben. Zum Beispiel den Nutzdatenstring „Hallo!“ (im folgenden nur noch String genannt).
Danach wollen wir das Board in den Schlaf-Modus mittels eines Konfigurationsbefehls bringen.

Nun schreiben wir mit einen externen Microcontrollers zum Beispiel folgendes:

```
putc(0x10);       // Empfänger ID
putc(0xF0);       // Acknowledge vom Empfänger nicht anfordern
printf(„Hallo!“); // Über UART den String „Hallo!“ senden
set_high(MDSEL);  // MDSEL auf high um gleich im Anschluss...
putc(0xAA);       // den Konfigurationsbefehl für den Schlafmodus zu senden
putc(0x00);
```

Was passiert: Der externe Mikrocontroller oder Computer schreibt die beiden Bytes *0x10, 0xF0* und den String „Hallo!“ in den Ausgangs-FIFO seiner UART und setzt **gleich danach** den MDSEL-Pin auf high.
Während also die UART noch ihren Ausgangs-FIFO leert ist der MDSEL-Pin bereits auf high und die Zeichenfolge 0x10, 0xF0, „Hallo!“ wird als Konfigurationsstring erkannt. Da jeder Konfigurationsbefehl aus 2 Bytes besteht wird unserer 0x10, 0xF0, „Hallo!“ in mehrere Konfigurationsbefehle umgewandelt:

$$0x10, 0xF0, \text{"Ha"}, \text{"ll"}, \text{"o!"}$$

Bzw. vollkommen in hexadezimal:

$$\underbrace{0x10, 0xF0}_{1.Konfigbefehl} \ , \ \underbrace{0x48, 0x61}_{2.Konfigbefehl} \ , \ \underbrace{0x6C, 0x6C}_{3.Konfigbefehl} \ , \ \underbrace{0x6F, 0x21}_{4.Konfigbefehl}$$

Also 4 Konfigurationsbefehle, die mit etwas Pech sogar noch Einstellungen ändern können!
Es ist also wichtig nach dem letzten gesendeten Nutzdatenstring eine Pause zu machen **bevor** der MDSEL-Pin auf high gezogen wird. Die Pause ist abhängig von der Übertragungsgeschwindigkeit der Schnittstelle und der Länge des Strings. Umgekehrt, also vom Konfigurations- zum Tranceivemodus gilt natürlich das Selbe. Wird direkt nach dem vermeindlichen „Senden“ eines Konfigurationsbefehls der MDSEL-Pin auf low gezogen werden die 2 Bytes, die als Konfigurationsbefehle vorgesehen sind im Transceivemodus als Funkdaten interpretiert. Da 2 Bytes jedoch erst Acknowledge und ID abdecken werden die 2 Bytes als fehlerhaft eingestuft und gelöscht. Der Schaden ist hier also weitaus geringer.

Es empfiehlt sich immer zwischen dem Wechsel von Konfigurations- und Transceivemodus eine kleine Pause zu machen um Fehler im Betrieb zu vermeiden.

Das Funkboard wurde dafür ausgelegt, dass einmal gesetzte Konfigurationen nicht mehr im laufenden Betrieb geändert werden. Der Zeitverlust, der beim Wechsel zwischen dem Transceive- und Konfigurationsmodus auftreten kann sollte also in der Praxis nicht von Relevanz sein.

5. Konfigurationsmodus

5.1 Allgemeines

Wird der MDSEL-Pin (ModeSelect-Pin) auf high (logisch 1) gesetzt ist das Funkboard im Konfigurationsmodus. Im Konfigurationsmodus werden alle wichtigen Einstellungen am Funkboard und seiner Software vorgenommen, editiert und geändert.
Keine, der an das Board übertragenen Daten in diesem Modus wird per Funk gesendet! Alle Daten dienen nur der internen Konfiguration des jeweiligen Funkboards.

Alle Konfigurationsdatenpakete bestehen aus 2 Bytes, es gibt kein Timeout. Das Programm wartet solange bis 2 Bytes eingegangen sind, erst dann werden die Daten verarbeitet. Es wird zwischen fixen Datenpaketen, Paketen mit einem variablen Parameter und Bitfeld-Paketen unterschieden.

- **Fixe Datenpakete** bestehen aus 2 Bytes, die in dieser Konstellation immer ein Ereignis auslösen. Zum Beispiel bewirkt das 2 Byte große Datenpaket `0xAA00` immer, dass das Funkboard in den Schlafmodus übergeht.
- **Pakete mit einem variablen Parameter** werden verwendet um interne Parameter oder Werte zu verändern. Das obere Byte ist dabei immer fix, das untere Byte ändert sich. Zum Beispiel wird mit `0xE0`**`yy`** die Identifikationsnummer des Moduls geändert. `yy` repräsentiert die jeweilige Nummer. Um dem Funkboard die Identifikationsnummer `5` zu geben würde man also `0xE005` senden. Für die ID mit der Nummer `11` folglich `0xE00B` usw.
- **Bitfeld-Pakete** enthalten hauptsächlich Daten, die dem Modul nur einmal zur Initialisierung übergeben werden und dann nicht mehr verändert werden. Bitfeld-Pakete sind natürlich auch 2 Bytes, also 16 Bit lang. Im Gegensatz zu den zuvor genannten Paketen wird hier jedoch auf Bit-Ebene gearbeitet.

Es ist neben dem setzen von Konfigurationseinstellungen auch möglich diese zu lesen. Dies soll es ermöglichen ein Funkboard zum Beispiel im Labor- oder Testbetrieb auch nach längeren Betriebspausen schnell und ohne Neuprogrammierung wieder einzusetzen. Wie dies genau geht ist im entsprechenden Abschnitt erwähnt.

5.2 Speichern und Automatisches Laden von Einstellungen

Das Funkboard lädt beim Initialisieren seine Konfigurationen aus dem internen EEPROM. Beim ersten Betrieb sind dies die Standardkonfigurationen.

Alle geänderten Konfigurationen lassen sich durch einen zusätzlichen Befehl (siehe nachfolgenden Abschnitt) im EEPROM speichern und werden dann bei allen folgenden Restarts automatisch geladen. Damit ist es in der Regel ausreichend das Funkboard einmalig umzukonfigurieren und dieseWerte dann dauerhaft abzuspeichern. Mit dem PC Terminalprogramm (siehe auch ***PC Terminalprogramm***) kann man diese Arbeit auch bequem am PC erledigen.

5.3 Konfigurations-Tabelle 1: Fixe Pakete

BEZEICHNUNG	HEX	BESCHREIBUNG
Sende Status Register	0x0000	Funkboard sendet Inhalt des Status-Registers
Schlafmodus	0xAA00	Funkboard in Schlafmodus
Standardinitialisierung	0xBA01	Initialisiert RFM12 Modul (nur Initialisierung)
Schnelle Konfiguration	0xBA02	Konfiguriert RMF12 Modul mit Standardparametern
Std.-einstellungen herstellen	0xCF01	Stellt Standardeinstellungen des gesamten Programms wieder her
Reset Funkboard	0xEE00	Softwarereset des Atmega32
Aktuelle Einstellungen speichern	0xDF01	Speichert aktuelle Einstellungen in EEPROM
FIFO Inputbuffer löschen	0xF001	Buffer für empfangene Funkdaten löschen
FIFO Outputbuffer löschen	0xF002	Buffer für zu sendende Funkdaten löschen
Versionsnummer anfordern	0xCE00	Funkboard sendet Versionsnummer und Compilierdatum der Software des AVRs
Empfangene Daten pollen	0xB500	Wenn Pollmodus aktiv sendet Funkboard oberstes Byte im FIFO Inputbuffer

5.3.1 Sende Status Register

Dieser Konfigurationsbefehl bewirkt, dass das Funkmodul den derzeitigen Inhalt seines Status-registers über die aktive Schnittstelle sendet. Der Inhalt dieses Registers ist in Abschnitt 7 beschrieben. Der Inhalt des Status-Registers kann zu jeder Zeit abgefragt werden. Wird der Status abgefragt ermittelt der ATmega32 nur interne Zustände, er liest weder den MAX3221 noch das RFM12 Modul aus. Dies hat zur Folge, dass die Peripherie parallel dazu weiterarbeitet. Zur Berechnung des Status Registers benötigt der AVR je nach Taktung nur wenige Mikrosekunden und fährt danach mit der zuvor begonnenen Aktion fort.

5.3.2 Schlafmodus

Wird der Wert für den Eintritt in den Schlafmodus gesendet werden alle LEDs ausgeschaltet, der MAX3221 und das RFM12 deaktiviert und in einen tiefen Standbymodus versetzt. Der AVR wird in den Power-Down-Modus versetzt. In diesem Zustand ist nur noch der Festspannungsregler (falls verwendet) aktiv. Der AVR speicherte bevor er in den Power-Down-Modus versetzt wurde **seinen Zustand**. Im Schlafmodus liegt der Stromverbrauch bei ca. 50µA. **Das Modul bleibt so lange im Schlafmodus wie der Zustand der MDSEL-Leitung auf high bleibt**. Sobald von außen das MDSEL-Signal auf low gezogen wird wacht das Modul auf. Es aktiviert alle LEDs und weckt das RFM12 und den MAX3221 wieder auf.

Wichtig ist, dass das Board nach seiner Wakeup-Phase, die ca. 15ms benötigt um seine Einstellungen zu laden. Anschließend arbeitet das Programm dort weiter wo es vor dem Sleepmode beschäftigt war! Es ist keine erneute Initialisierung nötig.
Ein Beispiel: Es werden dem Funkboard Daten zum Funktransfer übermittelt (im Tranceivemodus). Der FIFO Ausgangspuffer ist halb voll. Anschließend wird der Befehl zum Eintritt in den Schlafmodus gesendet und das Funkboard schläft. Nachdem es wieder aufgeweckt wurde macht es sofort damit weiter seinen FIFO Ausgangspuffer weiter zu leeren.

5.3.3 Standardinitialisierung

Die Standardinitialisierung bewirkt, dass das RFM12 Modul initialisiert wird. Das RFM12 Modul wird bereits nach jedem Reset und jedem Programmstart **automatisch** initialisiert, damit der ATmega32 mit dem RFM12 arbeiten kann. Auf Wunsch des Anwenders kann diese Initialisierung aber auch manuell gesendet werden.

Die Initialisierung des Moduls sieht dabei folgendermaßen aus (die Hexwerte werden an das RFM12 gesendet):

```
0x80D7   //Configuration Settings : EL=1, EF=1, 433MHz Band, 12.0 pF
0x8208   //Power Management: ER=0, EBB=0, ET=0, ES=0, EX=1, EB=0, EW=0, DC=0
0xA640   //Frequency Setting: A140=434MHz
0xC647   //Data Rate: 4,8kbps
0x94A0   //RX Control: VDI en, Fast resp., 134KHz Bandw., LNA 0dbm, RSSI -103dBm
0xC2AC   //Data Filter: CR=1 fast mode, S=0 digital filter, DQD Threshold = 4
0xCA81   //FIFO/RST mode: FIFO 13, SYNC, FF=0, DR=0 no sensitive RESET mode
0xC4E7   //AFC Command: Autom. Op=11, Frequ. Offset=10, ST=0, FI=1, OE=1, EN=1
0x9850   //TX Control: MP=0,9810=30kHz, Maximale Ausgangsleistung
0xE000   //Wakeuptimer deaktivieren
0xC800   //Low duty cycle deaktivieren
0xC000   //Low Battery detector und CLK-Prescaler deaktivieren
```

Für eine genauere Analyse bitte das Datenblatt zum RFM12 Modul verwenden.
Zusammenfassend: Die Standardinitialisierung wird automatisch bei Bedarf vom ATmega32 gesendet. Ein manuelles Initialisieren vor jeder Benutzung ist damit **nicht erforderlich**! Der Befehl wurde vielmehr aus Gründen der Flexibilität bereitgestellt und falls sich das RFM12-Modul (aus welchen Gründen auch immer) einmal aufhängen sollte. Der Befehl sollte außerdem nie alleine gesendet werden. Nach der Standardinitialisierung sollten weitere Konfigurationsbefehle folgen (jedoch nicht dem Standard-initialisierungsbefehl vorangehen).

5.3.4 Schnelle Konfiguration

Die schnelle Konfiguration initialisiert und konfiguriert das RFM12 Modul so, **wie es auch nach jedem Einschalten und jedem Reset erstkonfiguriert wird**. Dieser Befehl muss im Normalfall daher nicht manuell gesendet werden. Der Befehl wurde vielmehr aus Gründen der Flexibilität bereitgestellt und falls sich das RFM12-Modul (aus welchen Gründen auch immer) einmal aufhängen sollte. Der Befehl sollte außerdem nie alleine gesendet werden. Nach der Standardinitialisierung sollten weitere Konfigurationsbefehle folgen (jedoch nicht dem Standard-initialisierungsbefehl vorangehen).

5.3.5 Standardeinstellungen wiederherstellen

Dieser Befehl setzt die Standardeinstellungen des **kompletten Funkboards** wieder zurück. Der Befehl kann verwendet werden, sollten irrtümlicherweise alte (und womöglich fehlerhafte) Einstellungen aus dem EEPROM geladen worden sein oder sollte das Funkboard nach ändern gewisser Einstellungen nicht mehr reagieren.
Es werden sowohl die Standardeinstellungen des RFM12 Moduls hergestellt (die selben wie bei der Erstkonfiguration) als auch die des gesamten Programms. Diese Einstellungen werden außerdem im EEPROM abgelegt, so dass sie auch nach einem Reset wieder geladen werden. Das vorgehen entspricht dem unter ***Aktuelle Einstellungen speichern***.
Es folgt eine Liste aller Änderungen. In der mittleren Spalte ist der Wert angegeben, den der jeweilige Konfigurationsbefehl annehmen würde. Eine „5“ in der Spalte Kanalnummer würde bedeuten, dass es die selben Auswirkungen auf das Programm hätte wie wenn der Konfigurationsbefehl 0xE105 gesendet worden wäre, bzw. dass dieser Wert programmintern gesetzt wird. Bei Bitfeld-Konfigurationspaketen gibt der Wert den dezimalen Wert an, den die entsprechenden Bits repräsentieren.

BEZEICHNUNG	WERT DES PARAMETERS (DEZIMAL)	BESCHREIBUNG
Kanalnummer	5	Funkkanalnummer 5
Funk Baudrate	5	Funkinterne Baudrate 38400 Baud
LNA Gain	0	LNA Gain 0 dBm
RSSI	0	RSSI Threshold -103dBm
Bandbreite	4	Bandbreite 134kHz
DQD Report	3	DQD good signal report Stufe 3
Antenna Power	7	Maximale Sendestärke 7
Startzeichen / Startbyte	,#`, 0x23	Startzeichen / Startbyte (ASCII, Hex)
UART Baudrate	3	UART Baudrate 9600 Baud
Pollmodus	0	Pollmodus für UART deaktiviert
FIFO Eingangspuffer	8	Größe auf 8 Datenpakete setzen
FIFO Ausgangspuffer	8	Größe auf 8 Datenpakete setzen
LED Status	0	Alle LEDs aktiv
Clock Output Frequenz	0	10MHz (maximale) Taktfrequenz

Bitte beachten: Die Device ID des Funkboards bleibt davon **unbetroffen**!

5.3.6 Reset Funkboard

Das Funkboard kann nicht nur über einen Hardware- sondern auch über einen Softwarereset neu gestartet werden. Wird dieser Konfigurationsbefehl gesendet startet der ATmega32 seinen Watchdogtimer ohne ihn zu resetten. Dies hat zur Folge, dass der ATmega32 nach ca. 15ms einen Softwarereset erfährt und damit auch den MAX3221 und das RFM12 neu initialisiert und startet. **Der Softwarereset hat die gleichen Auswirkungen wie ein Hardwarereset.**

Logischerweise kann der Softwarereset jedoch nur durchgeführt werden, wenn sich der ATmega32 nicht „aufgehängt“ oder nicht „eingefrohren“ ist. An diesen Stellen hilft nur ein richtiger Hardwarereset.

5.3.7 Aktuelle Einstellungen speichern

Die Software bietet die Möglichkeit alle relevanten Konfigurationen im internen EEPROM des ATmega32 abzuspeichern. Damit ist es möglich einmalig das Funkboard an die Bedürfnisse umzukonfigurieren und diese Konfigurationen dauerhaft zu speichern. Der ATmega32 lädt dann **automatisch ohne weiteres Zutun mit jedem Reset bzw. Neustart die zuvor gespeicherten Werte und konfiguriert sich komplett selbstständig so um, wie zum Zeitpunkt der Speicherung.**
Mehrfachspeicherungen sind nicht möglich! Jedesmal wenn dieser Befehl gesendet wird werden die aktuellen unten genannten Konfigurationen gespeichert und alle vorherigen Werte überschrieben! Es werden außerdem immer **alle** unten genannten Einstellungen abgespeichert!
Achtung: Die Device-ID fällt nicht in diese Speicherung! Diese ID wird sofort nach deren erhalt im EEPROM abgelegt.

Unter der Speicherung betroffene Parameter

BEZEICHNUNG	BEZEICHNUNG
Kanalnummer	LNA Gain
Startzeichen / Startbyte	Antenna Power
Größe Inputbuffer	Funk Baudrate
Größe Outputbuffer	Clock Output Frequency
RSSI	Pollmodus
Bandbreite	UART Baudrate
DQD	Status der LEDs

5.3.8 FIFO Inputbuffer löschen

Mit diesem Befehl kann der Eingangspuffer bzw. Inputbuffer vollständig geleert werden. Im normalen Betrieb ist dieser Befehl nicht notwendig. Jedoch bietet er sich an, wenn veraltete oder fehlerhafte Daten im Puffer liegen und die externe Peripherie diese Daten nicht abrufen möchte.

5.3.9 FIFO Outputbuffer löschen

Mit diesem Befehl kann der Ausgangspuffer bzw. Outputbuffer vollständig geleert werden. Im normalen Betrieb ist dieser Befehl nicht notwendig. Jedoch bietet er sich als zusätzliche Sicherheitsmaßnahme an, wenn das Funkboard (aus welchen Gründen auch immer) diesen Puffer nicht richtig auslesen kann.

5.3.10 Versionsnummer anfordern

Mit dem ensprechenden Befehl ist es möglich die Versionsnummer der Software sowie das Compilierdatum der Software anzufordern. Achtung: Nachdem das Programm den entsprechenden Konfigurationsbefehl erhalten hat wird es die Versionsnummer sowie das Compilierdatum in einen String zusammenfügen und diesen String über die UART senden. Es ist nicht möglich diesen String über I2C oder SPI anzufordern!

5.3.11 Empfangene Daten pollen

Wird I2C, SPI oder UART mit aktivem Pollmodus verwendet muss dieser Befehl gesendet werden, damit das Funkboard ein Datenpaket über die genutzte Schnittstelle sendet. Genau gesagt wird das Paket ausgegeben, welches auch von dem im FIFO-Puffer enthaltenen Paketen zuerst empfangen wurde, d.h. zeitlich gesehen am ältesten ist.

5.4 Konfigurations-Tabelle 2: Pakete mit variablem Parameter

BEZEICHNUNG	HEX	BESCHREIBUNG
Identifikationsnummer	0xE0**yy**	Identifikationsnummer des Moduls Im Bereich zwischen 0 und 127 (dezimal)
Kanalnummer	0xE1**yy**	Funkkanal festlegen Im Bereich zwischen 0 und 38 (dezimal)
Startzeichen / Startbyte	0xE2**yy**	Startzeichen / Startbyte Im Bereich zwischen 0 und 254 (dezimal)
- Reserviert -	0xE3**yy**	Reserviert, nicht beschreiben
FIFO Inputbuffer Größe	0xE4**yy**	Größe des FIFO Eingangspuffers festlegen Im Bereich zwischen 0 und 10 Datenpaketen
FIFO Outputbuffer Größe	0xE5**yy**	Größe des FIFO Ausgangspuffers festlegen Im Bereich zwischen 0 und 10 Datenpaketen

5.4.1 Identifikationsnummer

Die Identifikationsnummer (ID) sollte idealerweise eine Nummer für ein Funkboard sein, die ansonsten kein anderes Funkboard im selben System / Netz besitzt. Die ID ist standardmäßig bei der Erstinbetriebnahme auf den Wert „0“ gesetzt. Sie kann jedoch zu jedem beliebigen Zeitpunkt geändert werden.
Wird die ID geändert, so wird ihr Wert im EEPROM des ATmega32 abgespeichert und bei einem Reset bzw. Start des Funkboards automatisch wieder geladen. Es ist also nicht nötig die ID eines Funkboards bei jedem Start oder Reset neu zu übertragen.
Der Wert für die ID kann zwischen 0 und 127 (dezimal) beziehungsweise 00 und 7F (hex.) liegen. Es wird jedoch dringend empfohlen die IDs mit den Werten „0“ und „0x55“ nicht zu verwenden da dies zu Kollisionen führen kann. Siehe auch ***Identifikationsnummer.***

5.4.2 Kanalnummer

Jedes Funkboard verfügt über 39 eigenständige Funkkanäle, die frei gewählt werden können. Das System funktioniert ähnlich zu RC-Cars oder RC-Flugzeugen, die meist zwischen 3 und 6 Funkkanäle bereitstellen. Damit zwei oder mehr Funkboards Daten austauschen können ist es zwingend erforderlich, dass sie exakt auf den selben Kanal eingestellt sind! Standardmäßig ist der Kanal 5 aktiv. Es kann jedoch auch jeder andere Kanal verwendet werden. Der Kanal kann selbstverständlich im laufenden Betrieb geändert werden.

5.4.3 Startzeichen / Startbyte

Das Startzeichen / Startbyte ist die unterste Sicherheitsabfrage für über Funk empfangene Daten. Jeder über Funk empfangene String muss mit einem bestimmten Byte beginnen (bzw. mit einem bestimmten ASCII-Zeichen, daher der Begriff Startzeichen). Entspricht das erste Zeichen eines über Funk empfangenen Strings nicht diesem Startzeichen wird der gesamte String als ungültig erklärt. Es ist damit unbedingt nötig, dass alle Funkboards die miteinander in Kontakt stehen über das selbe Startzeichen verfügen. Umgekehrt kann man 2 Funksysteme die auf dem selben Kanal senden voneinander durch das Startzeichen trennen. Der Umgang mit dem Startzeichen wird sowohl beim Sender als auch beim Empfänger wie bei allen Sicherheitsmerkmalen programmintern übernommen. Der Benutzer muss sich keine Gedanken darüber machen. Es ist jedoch möglich dieses Startzeichen (auch im laufenden Betrieb) zu ändern.
Standardmäßig ist das Startzeichen auf „#“ bzw. 0x23 festgelegt. Alles weitere zu den Sicherheitsmerkmalen im ***Datenübertragung und Datensicherheit***.

5.4.4 FIFO Inputbuffer Größe

Hier kann die Größe des FIFO Eingangspuffer (englisch Inputbuffer) konfiguriert werden. Wie der Puffer genau funktioniert ist unter ***Datenpufferung / FIFO-Puffer*** beschrieben. Die Größe des Buffers kann zwischen 1 und 10 Nutzdatenpacketen varrieren. Ist der FIFO Eingangspuffer beispielsweise auf „5“ gesetzt kann der FIFO Puffer 5 Nutzdatenpackete speichern, die dann nach und nach von einem externen Mikrocontroller oder PC abgerufen werden können. Ist die Größe des Eingangspuffers auf „1“ festgelegt kann nur ein Nutzdatenpaket gespeichert werden bevor es von einem neuen Nutzdatenpacket überschrieben wird. Für diesen Spezialfall ist also keine Pufferung der Daten vorhanden.

5.4.5 FIFO Outputbuffer Größe

Hier kann die Größe des FIFO Ausgangspu_ers (englisch Outputbu_er) konfiguriert werden. Wie der Puffer genau funktioniert ist unter ***Datenpufferung / FIFO-Puffer*** beschrieben. Die Größe des Buffers kann zwischen 1 und 10 Nutzdatenpacketen varrieren. Ist der FIFO Ausgangspuffer beispielsweise auf „5“ gesetzt kann der FIFO Puffer 5 Nutzdatenpakete speichern, die dann nach und nach über Funk versendet werden. Ist die Größe des Eingangspuffers auf „1“ festgelegt kann nur ein Nutzdatenpaket gespeichert werden bevor es von einem neuen Nutzdatenpacket überschrieben wird. Für diesen Spezialfall ist also keine Pufferung der Daten vorhanden.

5.5 Konfigurations-Tabelle 3: Erweitertes Bitfeld 1

Bezeichnung	Registerschemata															
Bitposition	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Erweitertes Bitfeld 1	0	0	1	ld	i2	i1	i0	b2	b1	b0	r2	r1	r0	d2	d1	d0

	d0-d2	**DQD Good quality report** 0…7 Dezimalwert *Standardwert*: 3
	r0-r2	**RSSI Threshold Wert** 0=-103dBm, 1=-97dBm, 2=-91dBm, 3=-85dBm 4=-79dBm, 5=-73dBm, 6=-67dBm, 7=-61dBm *Standardwert*: 0
	b0-b2	**Bandbreite** 0=400kHz, 1=340kHz, 2=270kHz, 3=200kHz 4=134kHz, 5=67kHz *Standardwert*: 4
	i0-i2	**Funk-Baudrate** 0=1200, 1=2400, 2=4800, 3=9600, 4=19200 5=38400, 6=57600, 7=115200 *Standardwert*: 5
	ld	**LEDs de-/aktivieren** 0 = Alle LEDs aktiv, 1 = Alle LEDs ausgeschaltet *Standardwert*: 0

5.5.1 DQD Good quality report

Der DQD quality report gibt an ab welcher Signalstärke RFM12-Modul von einem ausreichend guten Signal spricht. Für weitere Informationen über den DQD bitte das Datenblatt des RF12 hinzuziehen.

5.5.2 RSSI Threshold

Der RSSI (Received Signal Strength Indication) gibt einen Indikator für die Empfangsfeldstärke der Funkverbindung an. Je niedriger der RSSI-Wert ist, desto besser ist die Signalstärke. -103dBm ist daher der für die allermeisten Anwendungen beste RSSI-Wert.

5.5.3 Bandbreite

Die Bandbreite gibt an wie groß das genutzte Frequenzspektrum um den eingestellten Kanal, d.h. um die eingestellte Mittenfrequenz ist.

5.5.4 Funk-Baudrate

Die Funkbaudrate gibt die Datenrate an mit der Informationen über Funk übertragen werden. Umso höher die Baudrate umso geringer die Reichweite. Allerdings sinkt damit Bandbreite.

5.5.5 LEDs de-/aktivieren

Alle 3 LEDs (Power-Good-LED, Receive-Complete-LED, Transmit-LED) können auf Wunsch komplett abgeschaltet werden um zusätzlich Strom zu sparen.

5.6 Konfigurations-Tabelle 4: Erweitertes Bitfeld 2

Bezeichnung	Registerschemata															
Bitposition	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Erweitertes Bitfeld 2	0	1	0	pm	c2	c1	c0	e3	e2	e1	e0	p2	p1	p0	l1	l0

	l0-l1	**LNA Gain** 0=0dBm, 1=-6dBm, 2=-14dBm, 3=-20dBm *Standardwert*: 0
	p0-p2	**Antenna Power** 0...7 Dezimalwert *Standardwert*: 7
	e0-e3	**UART Baudrate** 0=1200 Baud, 1=2400 Baud, 2=4800 Baud, 3=9600 Baud 4=14400 Baud, 5=19200 Baud, 6=28800 Baud, 7=38400 Baud 8=57600 Baud, 9=76800 Baud, 10=115200 Baud 12=250000, 11=230400 Baud *Standardwert: 3*
	c0-c2	**Clock Output Frequenz** 0=10MHz, 1=5MHz, 2=3,33MHz, 3=2,5MHz 4=2MHz, 5=1,66MHz, 6=1,25MHz, 7=1MHz *Standardwert*: 0
	pm	**Pollmodus** 0 = Pollmodus aktiv, 1 = Pollmodus ausgeschaltet *Standardwert*: 0

5.6.1 LNA Gain

Der LNA Gain gibt die Verstärkung relativ zum Maximum in dB an. 0dB ist dabei eine Verstärkung/ Abschwächung von 0. Mit fallenden Dezibelwerten sinkt damit die Verstärkung. Für die meisten Anwendungen ist daher die Abschwächung von 0dB optimal.

5.6.2 Antenna Power

Sendestärke des Antennenausgangs. 0 = Minimum, 7 = Maximum. Es ist fast immer empfehlenswert den Antennenausgang mit voller Stärke zu betreiben um eine maximale Reichweite zu erzielen.

5.6.3 UART Baudrate

Die UART Baudrate ist die Baudrate für die UART-Schnittstelle, nicht für die Funkübertragung.

5.6.4 Clock Output Frequenz

Die Clock Output Frequenz gibt die Frequenz an, die am Pin CLK des RFM12 Moduls anliegt. Dieser CLK-Pin ist über einen Pin der Stiftleisten abgreifbar (ebenfalls als CLK-Pin bezeichnet). Die zuletzt festgelegte Frequenz wird intern gespeichert und nach jedem Reset / Anschalten erneut geladen.

5.6.5 Pollmodus

Aktiviert bzw. Deaktiviert den Pollmodus. Siehe auch ***Pollmodus***

5.7 Konfigurations-Tabelle 5: Erweitertes Bitfeld 3

BEZEICHNUNG	REGISTERSCHEMATA															
BITPOSITION	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ERWEITERTES BITFELD 3	0	1	1	0	0	0	0	0	0	0	0	0	0	0	m1	m2
	m0-m1			**MAX3221 Betriebsart** 0=Ein + AutoShutdown 1=Ein + Normal Operation (kein AutoShutdown) 2=Aus/Deaktiviert *Standardwert*: 0												

5.7.1 MAX3221 Betriebsart

Dieses Register legt die Betriebsart des MAX3221 fest. Im Normalfall regelt der AVR automatisch die Verwendung des MAX3221. (Siehe auch ***Abschnitt 3. Transceivemodus***). Es besteht jedoch auch die Möglichkeit den MAX3221 manuell umzukonfigurieren. **Bitte beachten, dass dieses Setting nicht gespeichert werden kann!**

5.7.2 Erweiterungsmöglichkeit

Dieses Register ist nicht voll ausgenutzt. Es ist für zukünftige Erweiterungen reserviert und vorbehalten. Es kann jedoch auch für eigene Modifikationen verwendet werden. **Wichtig ist hierbei jedoch, dass die Bits 13, 14 und 15 nicht geändert werden!** Ansonsten geht die Eindeutigkeit dieses Registers verlohren und Kollisionen mit anderen Konfigurationsbefehlen können entstehen.

5.8 Lesen von Konfigurationen

5.8.1 Allgemeines

Neben dem setzten von Konfigurationen ist es auch möglich diese auszulesen. Dies soll es ermöglichen ein Funkboard zum Beispiel im Labor- oder Testbetrieb auch nach längeren Betriebspausen schnell und ohne Neuprogrammierung wieder einzusetzen. Außerdem wäre ein PC-Terminalprogramm ohne diese Option gar nicht erst möglich.

Alle Konfigurationen mit variablem Parameter sowie alle Bitfelder sind auslesbar.

5.8.2 Auslesevorgang

Die Konfigurationsdaten lassen sich durch sehr einfache Weise auslesen.

- **Konfigurationen mit variablem Parameter**
  Das führende hexadezimale „E“ des kostant-bleibenden Bytes wird durch ein hexadezimales „D“ ersetzt. Das nachfolgende (variable) Byte wird mit hex. „00“ beschrieben.
  Ein Beispiel: Um die Identifikationsnummer zu lesen wird hex. `0xD000` gesendet. Für die Kanalnummer wird `0xD100` gesendet usw.

- **Bitfelder**
  Um Bitfeld 1 auszulesen wird hex. `0xD600` gesendet.
  Um Bitfeld 2 auszulesen wird hex. `0xD700` gesendet.
  Um Bitfeld 3 auszulesen wird hex. `0xD800` gesendet.

5.8.3 Antwort

Das Funkboard sendet nach dem Empfang eines solchen Konfigurationsbefehls den Inhalt des gewünschten Konfigurationsregisters. Dabei sendet das Funkboard, unabhängig vom Typ der gewünschten Konfiguration (variabler Parameter oder Bitfeld) **immer 2 Bytes bzw. 16 Bit!**

Bei Konfigurationen mit einem variablen Parameter entspricht das oberste Byte immer dem obersten Byte des eingegangenen Befehls. Sendet man beispielsweise `0xD100` um den aktuellen Kanal auszugeben antwortet das Funkboard beispielsweise mit `0xD104` wenn es im Moment auf Kanal 4 eingestellt ist.
Bei Konfigurationen mit Bitfeldern wird immer **das komplette Bitfeld gesendet**, da dieses gerade immer 16 Bit groß ist.

5.9 Verwendung des MDSEL-Pins im Konfigurationsmodus

Achtung: Es ist unbedingt auf die richtige Handhabung des MDSEL-Pins zu achten! Eingehende Bytes werden dann, **und nur dann**, als Konfigurationsdaten für den Konfigurationsmodus erkannt, wenn der MDSEL-Pin spätestens mit dem Empfangen des letzten Bits eines jeden einzelnen Bytes auf high geht. Besonders bei Verwendung der UART ist hier Vorsicht geboten!

Ein Beispiel:
Wir wollen Nutzdaten zur Funkübertragung mittels der UART in das Funkboard schreiben. Zum Beispiel den Nutzdatenstring „Hallo!“ (im folgenden nur noch String genannt).
Danach wollen wir das Board in den Schlaf-Modus mittels eines Konfigurationsbefehls bringen.

Nun schreiben wir mit einen externen Microcontrollers zum Beispiel folgendes:

```
putc(0x10);       // Empfänger ID
putc(0xF0);       // Acknowledge vom Empfänger nicht anfordern
printf(„Hallo!“); // Über UART den String „Hallo!“ senden
set_high(MDSEL);  // MDSEL auf high um gleich im Anschluss...
putc(0xAA);       // den Konfigurationsbefehl für den Schlafmodus zu senden
putc(0x00);
```

Was passiert: Der externe Mikrocontroller oder Computer schreibt die beiden Bytes *0x10, 0xF0* und den String „Hallo!“ in den Ausgangs-FIFO seiner UART und setzt **gleich danach** den MDSEL-Pin auf high.
Während also die UART noch ihren Ausgangs-FIFO leert ist der MDSEL-Pin bereits auf high und die Zeichenfolge 0x10, 0xF0, „Hallo!“ wird als Konfigurationsstring erkannt. Da jeder Konfigurations-befehl aus 2 Bytes besteht wird unserer 0x10, 0xF0, „Hallo!“ in mehrere Konfigurationsbefehle umgewandelt:

$$0x10, 0xF0, \text{"Ha"}, \text{"ll"}, \text{"o!"}$$

Bzw. vollkommen in hexadezimal:

$$\underbrace{0x10, 0xF0}_{1.Konfigbefehl} \ , \ \underbrace{0x48,0x61}_{2.Konfigbefehl} \ , \ \underbrace{0x6C, 0x6C}_{3.Konfigbefehl} \ , \ \underbrace{0x6F, 0x21}_{4.Konfigbefehl}$$

Also 4 Konfigurationsbefehle, die mit etwas Pech sogar noch Einstellungen ändern können!
Es ist also wichtig nach dem letzten gesendeten Nutzdatenstring eine Pause zu machen **bevor** der MDSEL-Pin auf high gezogen wird. Die Pause ist abhängig von der Übertragungsgeschwindigkeit der Schnittstelle und der Länge des Strings. Umgekehrt, also vom Konfigurations- zum Tranceivemodus gilt natürlich das Selbe. Wird direkt nach dem vermeindlichen „Senden“ eines Konfigurationsbefehls der MDSEL-Pin auf low gezogen werden die 2 Bytes, die als Konfigurationsbefehle vorgesehen sind im Transceivemodus als Funkdaten interpretiert. Da 2 Bytes jedoch erst Acknowledge und ID abdecken werden die 2 Bytes als fehlerhaft eingestuft und gelöscht. Der Schaden ist hier also weitaus geringer.

Es empfiehlt sich immer zwischen dem Wechsel von Konfigurations- und Transceivemodus eine kleine Pause zu machen um Fehler im Betrieb zu vermeiden.

Das Funkboard wurde dafür ausgelegt, dass einmal gesetzte Konfigurationen nicht mehr im laufenden Betrieb geändert werden. Der Zeitverlust, der beim Wechsel zwischen dem Transceive- und Konfigurationsmodus auftreten kann sollte also in der Praxis nicht von Relevanz sein.

6. Datenübertragung und Datensicherheit

6.1 Datenübertragung

Die Datenübertragung per Funk funktioniert nach einem immer gleichbleibenden Schema. Um die Übertragung so sicher wie möglich zu machen wurden zahlreiche Sicherheitsmerkmale implementiert, die ebenfalls im Folgenden beschrieben werden. Die gesamte Aufbereitung der Daten wird jedoch so weit wie möglich vor dem Benutzer versteckt.

Die Datenübertragung läuft - egal ob Empfangs- oder Sendebetrieb - in 3 Phasen ab:

1. **Initialisierung des Empfangs- oder Sendebetriebs**
   Bevor Daten empfangen oder gesendet werden können muss das RFM12 Modul für den jeweiligen Betrieb initialisiert werden. Die Initialisierung für den Sendebetrieb dauert ca. 922µs, die für den Empfangsbetrieb etwa 194µs. Die Initialisierungsroutine für den jeweiligen Betrieb muss jedesmals aufs Neue erfogen. Werden also 2 Nutzdatenpakete gesendet muss auch 2 mal auf Sendebetrieb initialisiert werden, ebenso wenn 2 Pakete über Funk empfangen werden.
   Das Programm befindet sich standardmäßig immer im Empfangsmodus (siehe auch ***Abschnitt 3. Transceivemodus***). Empfangene Daten können daher in der Regel sofort abgerufen werden, da die Initialisierung bereits zuvor schon durchgeführt wurde.

2. **Senden und Empfangen von Daten**
   Nach der Initialisierung können Daten gesendet oder (sofern welche vorhanden) auch empfangen werden. Diese Vorgänge gehen so lange, wie Datenbytes zum Senden bereitliegen bzw. so lange wie Datenbytes empfangen werden.

3. **Schließen des Sende- oder Empfangsbetriebs**
   Sind keine Daten mehr zu senden oder es gibt keine empfangenen Daten mehr (oder die empfangenen Daten sind fehlerhaft) wird der aktuelle Betrieb geschlossen. Die Daten werden nun verarbeitet. Für den weiteren Verlauf gibt es 2 Möglichkeiten: Das Funkboard soll Daten übertragen, dann wird gleich darauf der Sendebetrieb initialisiert und Punkt 1 fortgeführt. Liegen keine Daten zum Senden vor springt das Programm immer automatisch in den Empfangsbetrieb und initialisiert diesen.

6.1.1 Sendebetrieb

Werden Daten über Funk gesendet die zuvor über eine der Schnittstellen ins Funkboard geschrieben wurden, werden diese nicht einfach 1:1 an andere Funkboards gesendet. Vielmehr werden alle Datenstrings mit zahlreichen Sicherheitsmerkmalen „beaufschlagt“. Dazu zählen Startzeichen / Startbytes, Identifikationsnummern, Checksummen, Acknowledge-Requests und Füllbytes. Dieser komplexe String, der tatsächlich über Funk gesendet wird setzt sich aus folgenden Bytes zusammen:

Bytenummer	0	1	2	3	4	5	6	...	n-4	n-3	n-2	n-1	n
Bezeichnung	Startbyte	ID Empfänger	ID Sender	Datenlänge	8 Bit CRC	Data[0]	Data[1]	...	Data[m]	16 Bit CRC[0]	16 Bit CRC[1]	Rohrputzer	Rohrputzer
Beispiel	0x23	0x10	0x05	0x0A	0xA4	H'	a'	...	o'	0x56	0xA3	0xAA	0xAA

- **Byte 0: Startzeichen/Startbyte**
  Siehe ***Startzeichen / Startbyte*** Für eine detaillierte Beschreibung dieses Sicherheitsmerkmals

- **Byte 1: Identifikationsnummer des Empfängers**
  Die Identifikationsnummer (ID) des Funkboards, an die die gefunkten Daten gerichtet sind.

- **Byte 2: Identifikationsnummer des Senders + Acknowledge-Bestätigung**
  Die Identifikationsnummer des Boards, das den String sendet. Sie ist nötig, damit der Empfänger weis von welchem Funkboard aus der String gesendet wurde. Dieses Byte trägt außerdem noch den Status der Acknowledge-Bestätigung.
  Ist das MSB auf 1 wird eine Acknowledge-Bestätigung durch den Sender verlangt, ist das MSB auf 0 gibt es keine Acknowledge-Bestätigung. Siehe auch ***Acknowledgement***

- **Byte 3: Datenlänge**
  Dieses Byte gibt die Länge der reinen Nutzdaten an, die das Paket enthält. Dieses Byte ist notwendig, damit der Empfänger weis wieviel Bytes erwartet werden.

- **Byte 4: 8 Bit Checksumme**
  Byte 0 bis Byte 3 sind durch eine 8 Bit Checksumme gesichert um die Datensicherheit zu erhöhen. Siehe ***Checksummen*** für eine detailierte Beschreibung der Checksummen.

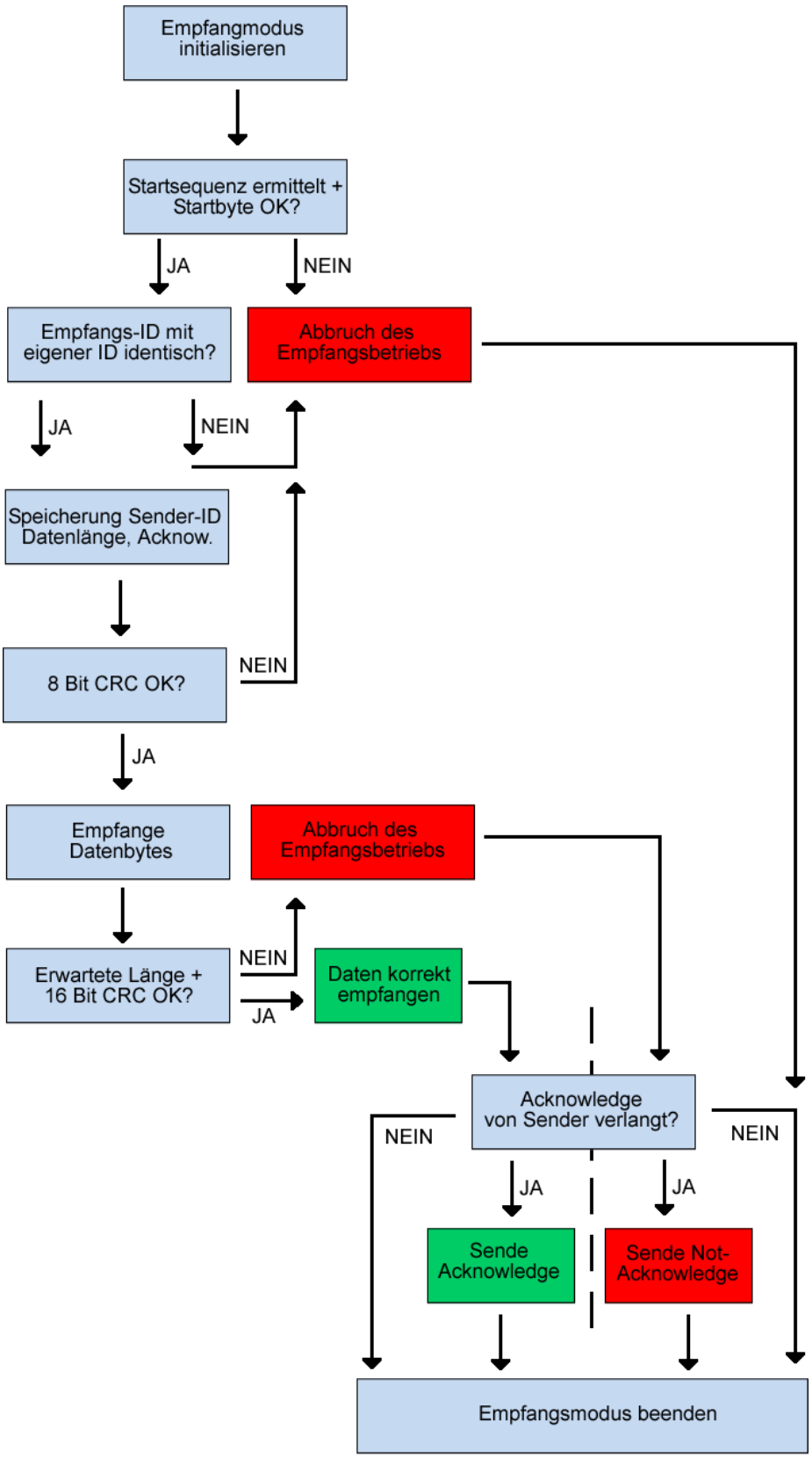
- **Byte 5 – Byte n-4: Nutzdaten**
  Nun folgen die Nutzdaten des Anwenders die übertragen werden sollen. Die Nutzdaten werden an einem Stück übertragen.

- **Byte n-3 – n-2: 16 Bit Checksumme, gesplittet auf 2 Bytes**
  Die Nutzdaten werden mit einer 16 Bit Checksumme, verteilt auf 2 Byte, gesichert. Byte n-3 enthält das MSB, Byte n-2 das LSB des 16 Bit Halfwords.

- **Byte n-1 – Byte n: Rohrputzer**
  Um den TX-FIFO des RFM12 Moduls wieder auf Standardwerte zu füllen werden 2 sogenannte „Rohrputzer“ in den TX-FIFO geschrieben. Diese werden vom Modul ebenfalls gesendet, das letzte Byte jedoch kann in seinem Wert varrieren, ein gesendetes 0xAA muss also nicht als 0xAA beim Empfänger ankommen. Aus diesem Grund werden die beiden letzten Bytes nicht auf ihren Wert überprüft sondern einfach nur abgeholt.

6.1.2 Empfangsbetrieb

Bei Empfang der Daten wird das unter dem ***Sendebetrieb*** schematisierte Datenpaket ausgewertet. Diese Prozedur läuft in mehreren Schritten ab und ist in dieser Grafik veranschaulicht:



Werden die Daten korrekt empfangen können diese abgerufen und weiterverarbeitet werden. Tritt jedoch in der Kette ein Fehler auf wird sofort abgebrochen und alle bisher empfangenen Daten verworfen. Außerdem wird dem Errorhandler eine Fehlermeldung übergeben.

6.2 Datensicherheit

6.2.1 Allgemeines

Die Sicherheit der Funkübertragung ist einer der Kernaspekte der Software. Mit zahlreichen Sicherheitsfeatures wird die Wahrscheinlichkeit eines fehlerhaft übertragenen Strings, der beim Empfänger dennoch als valide erkannt drastisch gesenkt. Alle Sicherheitsfeatures (bis auf das Acknowledgement) laufen völlig im Hintergrund und brauchen vom Benutzer nicht berücksichtigt werden. Es wurde außerdem darauf geachtet eine möglichst gute Ausgewogenheit zwischen benötigter Anzahl an Bytes und der erhaltenen Sicherheitsstufe zu realisieren. Jedes zu übermittelnde Byte braucht seine Zeit. Zeit, die sich bei einigen Stunden, Tagen oder gar Wochen Laufzeit und bei hohen Bandbreiten in der Leistungsfähigkeit der Software niederschlägt. Die hier verwendeten Sicherheitsabfragen stellen einen guten Ausgleich zwischen sehr hoher Sicherheit und benötigter Anzahl an Bytes dar.

6.2.2 Startsequenz

Die RFM12 Module bieten die Möglichkeit eine Startsequenz zu realisieren auf die das Empfangsmodul triggert. Dies wird in der Software für das Funkboard ausgenutzt. Der Empfänger reagiert dabei nur auf eingehende Daten, wenn die erwartete Startsequenz mit der tatsächlich über Funk empfangenen Sequenz überein stimmt. Die Startsequenz ist immer die selbe und nicht veränderbar. Sie wird bei jedem Sendevorgang automatisch gesendet und der Empfänger triggert immer auf diese Sequenz.

6.2.3 Startzeichen / Startbyte

Das Startzeichen bzw. Startbyte ist die unterste Sicherheitsabfrage für über Funk empfangene Daten. Jeder über Funk empfangene String muss mit einem bestimmten Byte beginnen (bzw. mit einem bestimmten ASCII-Zeichen, daher der Begriff Startzeichen). Entspricht das erste Zeichen eines über Funk empfangenen Strings nicht diesem Startzeichen wird der gesamte String als ungültig erklärt. Es ist damit unbedingt nötig, dass alle Funkboards die miteinander in Kontakt stehen über das selbe Startzeichen bzw. Startbyte verfügen.
Umgekehrt kann man 2 Funksysteme die auf dem selben Kanal senden voneinander durch das Startzeichen trennen. Der Umgang mit dem Startzeichen wird sowohl beim Sender als auch beim Empfänger wie bei allen Sicherheitsmerkmalen programmintern übernommen. Der Benutzer muss sich keine Gedanken darüber machen. Es ist jedoch möglich dieses Startzeichen (auch im laufenden Betrieb) zu ändern.

Das Startzeichen hat hauptsächlich eine Funktion: Es soll es sicherstellen, dass der Empfänger bei einem fehlerhaften String nicht einfach in der Mitte beginnt diesen String einzulesen. Die Wahrscheinlichkeit, dass das „erste“ Zeichen dieses fehlerhaften/abgehackten Strings mit dem internen Startzeichen übereinstimmt ist mit 1/254 sehr klein. Zwar würden höhere Sicherheitsstufen (wie z.B. die Checksummen) diese Fehler sehr wahrscheinlich auch aufdecken, jedoch steigert man dadurch die Sicherheit und Verifizierbarkeit der Übertragung.

Werden die Datenströme mit Messgeräten untersucht lässt sich anhand dieses Startzeichens leicht ein „Einstiegspunkt“ finden, was sich beim Debuggen als enorm nützlich erwiesen hat.

6.2.4 Längenangabe der Nutzdaten

Damit der Empfänger weis wieviel Bytes an Nutzdaten erwartet werden wird zusätzlich ein Byte bei jedem Sendevorgang mitgesendet, das die Anzahl an Nutzdatenbytes angibt, die versendet werden. So kann der Empfänger auch kontrollieren ob das empfangene Paket die richtige Länge aufweist und die Anzahl an Nutzdatenbytes mit dem zuvor empfangenen Längen-Byte übereinstimmen.

6.2.5 Identifikationsnummer

Die Identifikationsnummer (kurz ID) sollte idealerweise eine Nummer für ein Funkboard sein, die ansonsten kein anderes Funkboard im selben System / Netz besitzt. Die ID ist standardmäßig bei der Erstinbetriebnahme auf den Wert `0x00` gesetzt. Sie kann zu jedem beliebigen Zeitpunkt geändert werden.
Wird die ID geändert, so wird ihr Wert im EEPROM des ATmega32 abgespeichert und bei einem Reset bzw. Start des Funkboards **automatisch wieder geladen**. Es ist also nicht nötig die ID eines Funkboards bei jedem Start oder Reset neu zu übertragen.
Der Wert für die ID kann zwischen 0 und 127 (dezimal) beziehungsweise 00 und 7F (hex.) liegen. Es wird jedoch dringend empfohlen die IDs mit dem Werten `0x00` und `0x55` nicht zu verwenden da dies zu Kollisionen führen kann, da der Wert `0x00` für neu-programmierte Funkboards vorgesehen ist und `0x55` die globale ID ist (siehe unten).

Die ID kann als „Name“ eines Funkboards angesehen werden mit dem es angesprochen wird. Hat jedes Funkboard seine eigene ID lässt sich ein ganzes Funknetzwerk aus dutzenden Funkboards bilden, die sich untereinander durch ihre ID ansprechen. Empfängt ein Funkboard ein Datenpaket indem eine Empfänger-ID angegeben ist, die nicht mit der eigenen übereinstimmt wird das Paket abgelehnt und für ungültig erklärt.

Achtung: Es gibt dabei eine Ausnahme - Die globale ID!
Angenommen es existiert ein Funknetzwerk mit mehreren Slaves und einem Master. Der Master kann alle Slaves anhand ihrer ID Datenpakete senden, die nur für den jeweiligen Slave bestimmt sind. Will der Master jedoch Datenpakete an alle Slaves senden kann das zu einem horrenden Datenaufkommen führen, wenn der Master jeden einzelnen Slave bedienen darf.
Um das zu vermeiden gibt es eine globale ID mit dem Wert `0x55` (dezimal 85). Sendet ein Funkboard ein Paket indem die Empfänger ID gleich der globalen ID ist kann jedes andere Funkboard dieses Paket empfangen. Obwohl möglich wird in jedem Fall empfohlen Pakete mit globaler ID nicht mit einer Acknowledgement-Bestätigung zu versehen. **Ansonsten würden alle Funkboards ein Acknowledge oder Not-Acknowledge an den Master zurücksenden und diesen überlasten.**

6.2.6 Checksummen

Checksummen dienen dazu Bytegruppen innerhalb eines Datenpakets zu sichern. In der Software wird mit 2 unterschiedlichen Checksummen (bzw. CRCs) gearbeitet. Zum einen eine 8 Bit Checksumme um die ersten 4 empfangenen Bytes (Startbyte bis Datenlängenbyte) zu sichern, zum anderen eine 16 Bit Checksumme um damit die Nutzdatenbytes zu sichern. Das Verfahren ist für beide Checksummen prinzipiell gleich, jedoch nicht absolut identisch!
Der Sender summiert die Bytes, die zur jeweiligen Checksumme gehören zunächst auf und ermittelt dann aus dieser Summe eine CRC. Diese CRC wird dann über Funk zu einem Empfänger übertragen.
Der Empfänger errechnet aus den eingegangenen Bytes selbst fortlaufend die Quersumme und ab einer bestimmten Anzahl an eigegangenen Bytes errechnet er dazu nach dem selben Verfahren wie der Sender eine CRC und vergleicht sein Ergebnis mit dem Wert des Bytes wovon der Empfänger ausgeht, dass es die Checksumme repräsentiert. Stimmt gesendete und errechnete Checksumme überein wird der Datentransfer aufrecht erhalten, ansonsten sofort abgebrochen.

6.2.7 Acknowledgement

Das Acknowledgement ist die oberste Sicherheitsstufe. Alle anderen Sicherheitsmerkmale sind rein empfängerseitig. Empfängt der Empfänger ein fehlerhaftes Packet kann dieser das Paket zwar aussortieren, der Sender weis davon jedoch nichts. **Genau hier greift das Acknowledgement ein. Es fungiert als Handschlag zwischen Sender und Empfänger.** Fordert der Sender den Empfänger zu einem Acknowledge auf gibt es genau 3 Möglichkeiten:

1. **Daten korrekt empfangen: Acknowledge**

   Der Empfänger empfängt ein Paket und stuft es als korrekt ein. Nun sendet der Empfänger ein kleines Datenpaket, welches den Sender darüber informiert, dass die Datenübertragung erfolgreich war und dass der Sender das nächste Datenpaket übertragen kann. Er sendet also ein **Acknowledge**. Dieses kleine Datenpaket des Empfängers zur Bestätigung hat den folgenden Aufbau:

Bytenummer	0	1	2	3
Bezeichnung	Startzeichen	ID Empfänger	ID Sender	Acknowledge
Beispiel	'#'	0x02	0x03	0xAA

2. **Daten nicht korrekt empfangen: Not Acknowledge**

   Der Empfänger empfängt ein Paket, stuft es jedoch als fehlerbehaftet ein und verwirft es. Er sendet nun dem Sender ein kleines Datenpaket das ihn darüber informiert, dass die Übertragung fehlerhaft war, ein **Not-Acknowledge**. So kann der Sender die Daten noch einmal senden. Dieses kleine Datenpaket des Empfängers hat den folgenden Aufbau:

Bytenummer	0	1	2	3
Bezeichnung	Startzeichen	ID Empfänger	ID Sender	Not-Acknowledge
Beispiel	'#'	0x02	0x03	0x0F

3. **Daten erreichen Empfänger nicht**

   Der Sender sendet ein Paket welches beim Emfpänger nie ankommt. Während der Sender also auf ein Acknowledge/Not-Acknowledge wartet, merkt dieser nach einiger Zeit, dass nichts von beidem ankommt. Es muss also ein (schwerwiegender) Fehler in der Funkverbindung aufgetreten sein. Der Sender versucht nun noch weitere 4 Male das Packet zu senden. Schlagen auch diese Versuche fehl bricht er die Aktion mit einer Fehlermeldung ab.

Der Vorteil des Acknowledgements ist, dass fehlerhafte Daten erneut gesendet werden können. Dies ist besonders bei kritischen Daten wie Konfigurationseinstellungen wichtig. Einfach überall dort wo Daten unbedingt eintreffen müssen.

Das Acknowledgement hat jedoch auch einen Nachteil - Es braucht viel Zeit. Der Sender schaltet für etwa 2-3ms in den Empfang um die Antwort vom Empfänger aufzunehmen. Der Empfänger benötigt etwa 1,3ms um das Acknowledge oder Not-Acknowledge zu senden. Bei jedem gesendeten Paket müssen also ca. 3ms hinzugerechnet werden, was sich bei einem hohen Datenaufkommen schnell bemerkbar macht und selbst wieder eine Fehlerquelle ist (Bufferoverflow des Outputbuffers).

Desshalb sollte man das Acknowledge nur aktivieren wenn es nötig ist. Bei sehr zeitkritischen oder redundanten Daten macht ein Acknowledge kaum Sinn. Die Acknowledgement-Prüfung lässt sich für jedes einzelne Paket separat aktivieren und deaktivieren.

Das Acknowledge ist grundsätzlich das zweite Byte, welches einem Funkboard beim Transceive-modus übergeben wird. Es steht direkt nach der Empfänger-ID. Wird ein Acknowledgement verlangt ist dieses Byte 0xAA, wird keines verlangt ist es 0xF0. Entsprechend wird beim gefunkten Datenpaket bei der Sender-ID das MSB entweder auf 1 (Acknowledge verlangt) oder auf 0 (kein Acknowledge) gesetzt.

Beispiel: Der Nutzdatenstring „Hilfe!“ soll, mit einer Acknowledge-Prüfung versehen, gefunkt werden. Also wird dem Funkboard folgende Daten übermittelt (z.B. über die UART, MDSEL-Pin muss natürlich low sein):

$$\underbrace{0x10}_{Empfänger-ID} \quad , \quad \underbrace{0xAA}_{\text{Acknowledge!}} \quad , \underbrace{0x48,0x69,0x6C,0x66,0x65,0x21}_{\text{Nutzdaten,ASCII-String "Hilfe!"}}$$

Umgekehrt soll nun der Datenstring „12345“ gesendet werden, jedoch ohne Acknowledge-Prüfung:

$$\underbrace{0x10}_{Empfänger-ID} \quad , \quad \underbrace{0xF0}_{\text{kein Acknowledge!}} \quad , \underbrace{0x31,0x32,0x33,0x34,0x35}_{\text{Nutzdaten,ASCII-String "12345"}}$$

6.3 Tipps zur Anpassung: Bandbreite vs. Reichweite

6.3.1 Allgemeines

An dieser Stelle einige Dinge, auf die man achten sollte, **bevor** man das Funkboard für eine Funkübertragung einsetzen will und das Übertragungsprotokoll und die Konfigurationen fix einstellt.

Generell kann man sagen, dass sich die Bandbreite und Reichweite reziprok verhalten, das heist, dass man die Bandbreite auf Kosten der Reichweite erhöhen kann und anders herum. Es ist praktisch nicht möglich sowohl maximale Bandbreite als auch maximale Reichweite zu erzielen, es muss immer ein Kompromiss stattfinden. Wichtig ist jedoch diesen Kompromiss so auszulegen, dass er sich nicht massiv auf die Übertragungsqualität auswirkt. Im nachfolgenden werden nun die beiden Extreme maximale Bandbreite und maximale Reichweite erläutert. In der Praxis liegt die Idealeinstellung meist irgendwo dazwischen. Da dieses „irgendwo“ jedoch von Anwendung zu Anwendung verschieden ist muss hier der Anwender selbst „herumexperimentieren“. Einen Idealplan für alle Anwendungen gibt es nicht.

6.3.2 Maximale Reichweite

Eine maximale Reichweite lässt sich dadurch erzielen, dass man zum einen eine optimale Antenne verwendet (je nach Anwendung Rundstrahl- oder Richtantenne) und zum Anderen die Bandbreite (Funk Baudrate) auf ein Minimum herunterfährt. Umso kleiner diese Baudrate umso sicherer wird die Übertragung, besonders bei hohen Reichweiten. Außerdem sollte man unbedingt mit Acknowledge-Prüfungen arbeiten, da hier der Fall, dass Pakete „im Raum verschwinden“ immer wahrscheinlicher wird. Lange Datenpakete sollten ebenfalls vermieden werden. Umso mehr Bytes ein Paket enthält umso anfälliger wird es für Störungen

6.3.3 Maximale Bandbreite

Eine maximale Reichweite lässt sich dadurch erzielen, dass man zum einen eine optimale Antenne verwendet (je nach Anwendung Rundstrahl- oder Richtantenne) und zum anderen den Abstand zwischen den Modulen minimiert. Eine Sichtverbindung wäre hier optimal. Die Funk Baudrate kann dann so hoch gewählt werden, wie noch ein Empfang sichergestellt wird. Bei hohen Bandbreiten ist es unter Umständen von Vorteil die Acknowledge-Prüfung zu deaktivieren, da dies zusätzliche Performance bringt. Außerdem sollte die maximale Nutzdatenlänge immer möglichst ausgeschöpft werden, da dann die RX/TX-Wechsel sowie Initialisierungen, CRCs etc. immer weniger ins Gewicht fallen.

7.Status Register

7.1 Allgemeines

Das Status Register ist ein 16 Bit großes Register, das wichtige Zustände des Programms dort anzeigt/zwischenspeichert. Dieses Status Register kann mit dem Konfigurationsbefehl 0x00 0x00 abgerufen werden. Nachdem dieser Konfigurationsbefehl an das Funkboard gesendet wurde, werden diese 16 Bit über die aktive Schnittstelle sofort gesendet.

Mit dem Status Register ist es damit möglich schnell und einfach zum Beispiel die Zustände der FIFO Puffer auszulesen. Neben dem RECC-Pin (Receive-Complete-Pin) ist dies also eine gute Möglichkeit neu eingegangene Daten (zum Beispiel für den Pollmodus) wahrzunehmen.

Der Inhalt des Status Registers beruht nur auf programminternen Variablen des Programms im AVR. Es ist nicht nötig für das Status Register Befehle an das RFM12 zu senden. Dies hat sehr große Vorteile: Zum einen ist damit das Status Register binnen weniger Mikrosekunden generiert, zum anderen bekommt das RFM12 von alle dem nichts mit, es arbeitet also einfach ungestört weiter.

Es besteht beim Lesen des Status Register also keine Gefahr eingehende Funkdaten
zu versäumen.

Das Status Register ist wie angesprochen ein Konfigurationsbefehl, wenn auch ein besonderer. Es gelten jedoch die selben Regeln wie für andere Konfigurationsbefehle:
Beim Senden des Status-Register-Konfigurationsbefehls muss der MDSEL-Pin auf high sein. Das Funkboard reagiert erst auf den Befehl wenn alle erwarteten 6 Bit eingegangen sind. Der Konfigurationsbefehl zum Erlangen des Status-Registers kann immer gesendet werden, der Ablauf des Programms wird dadurch nicht gestört. Auf den folgenden Seiten werden die Inhalte des Status Registers genauer erläutert.

7.2 Error Handler

Die Software des Funkboards zeichnet im laufenden Betrieb viele bekannte Fehler auf und speichert den letzten vorgefallenen Fehler ab. Dieser Fehler kann dann im Status Register abgerufen werden. Wird das Status Register abgerufen wird auch anschließend (nach dem Abrufen!) **der letzte vorgefallene Fehler gelöscht**. So lässt sich ohne aufwendigeres Debugging auch feststellen in welchen Situationen die Software des Funkboards nicht korrekt arbeitet. Zuzüglich gibt es ein Bit, das ausschließlich signalisiert ob überhaupt ein Fehler vorgefallen ist oder nicht. Die externe Peripherie kann damit zuerst dieses Bit (er) auslesen. Zeigt es keinen Fehler an muss auch die 4 Bit lange Fehlermeldung nicht ausgelesen werden. Zeigt es jedoch einen Fehler an so kann mit der Fehlermeldung die Art des Fehlers bestimmt werden.

Selbstverständlich ist der Error Handler kein Pauschalrezept für eine fehlerfreie Funklösung. Vielmehr soll er Fehler, die öfters auftreten können katalogisieren.

Die 4 Bits e0 - e3 ergeben binär kodiert bzw. dezimal umgerechnet eine Nummer. Jede dieser Nummern (von 0 bis 15) steht für einen bestimmten Fehler:

0. **Kein Fehler**
   Kein Fehler seit dem letzten Register-abruf aufgezeichnet.

1. **Frame Fehler**
   Schnittstellenabhängiger Fehler:
   → UART: Fehlendes Stop Bit

2. **Überlauf Fehler**
   Schnittstellenabhängiger Fehler:
   → UART: Diverse Empfangsfehler

3. **Pufferüberlauf Fehler**
   Schnittstellenabhängiger Fehler:
   → UART: Überlauf des UART-Hardware-Puffers des ATmega32

4. **Empfangene Funkdaten nicht valide**
   Beim Empfang von Funkdaten kam es zu einem nicht spezifizierten Fehler

5. **Eingangspuffer Überlauf**
   Der FIFO Empfangsbuffer für das Empfangen von Funkdaten ist übergelaufen, ein Datenverlust ist vor-gefallen

6. **Acknowledge-Prüfung: Timeout Fehler**
   Ein Acknowledge wurde verlangt, der Empfänger hat in der vorgegebenen Zeitspanne jedoch nicht geantwortet

7. **Acknowledge-Prüfung: Empfangsfehler**
   Daten konnten auch nach mehreren Versuchen nicht an Empfänger übermittelt werden, Datenverust

8. **Acknowledge-Prüfung: NACK empfangen**
   Ein Acknowledgement wurde verlangt, der Empfänger hat mit einem NACK (Not-Acknowledge) geantwortet

9. **Acknowledge-Prüfung: Mehrf. Senden**
   Es wurde mehr als ein Versuch benötigt ein mit einem Acknowledgement ver-sehenes Datenpaket an einen Empfänger zu senden, Übertragung war jedoch erfolgreich

10. **Messages 10-15 noch nicht implementiert**
    Die Messagenummern 10 bis 15 sind in der momentanen Version noch keinem Fehler zugewiesen.

7.3 Registerinhalt Status Register

BEZEICHNUNG	REGISTERSCHEMATA															
BITPOSITION	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
STATUS REGISTER	0	0	0	0	lb	e3	e2	e1	e0	er	fv	fb	fr	fo	fe	fd

fd	**FIFO Input Buffer - Daten verfügbar:** 0=Leer, keine Daten verfügbar 1=Daten im Puffer verfügbar
fe	**FIFO Input Buffer - Puffer leer:** 0=Leer, keine Daten im Puffer verfügbar 1=Puffer nicht leer
fo	**FIFO Input Buffer - Overflow:** 0=Kein Bufferoverflow seit letzten Abruf vorgefallen 1=Bufferoverflow vorgefallen *Dieses Bit wird nach jedem Statusregister-Lesebefehl auf den Wert 0 zurückgesetzt*
fr	**FIFO Output Buffer - Bereit:** 0=Puffer ist bereit um neue Pakete einzulesen 1=Puffer ist nicht bereit, später erneut versuchen
fb	**FIFO Output Buffer - Puffer leer:** 0=Leer, keine Daten im Puffer verfügbar 1=Puffer nicht leer
fv	**FIFO Output Buffer - Overflow:** 0=Kein Bufferoverflow seit letzten Abruf vorgefallen 1=Bufferoverflow vorgefallen *Dieses Bit wird nach jedem Statusregister-Lesebefehl auf den Wert 0 zurückgesetzt*
er	**Error Handler - Fehler vorgefallen:** 0=Kein Fehler seit letzten Abruf vorgefallen 1=Fehler seit letzten Abruf vorgefallen *Dieses Bit wird nach jedem Statusregister-Lesebefehl auf den Wert 0 zurückgesetzt*
e0-e3	**Error Handler - Fehlermeldung:** Hält den Binärwert des letzten aufgetretenen Fehlers fest Siehe Tabelle unter „Error Handler“ *Diese Bits wird nach jedem Statusregister-Lesebefehl auf den Wert 0000 zurückgesetzt*
lb	**Low Battery Flag:** 0=Spannung ausreichend 1=Niedrige Spannung *Wird der ZLDO-Spannungswandler verwendet ist dieses Bit aussagekräftig, ansonsten nicht!*

8. PC Terminalprogramm

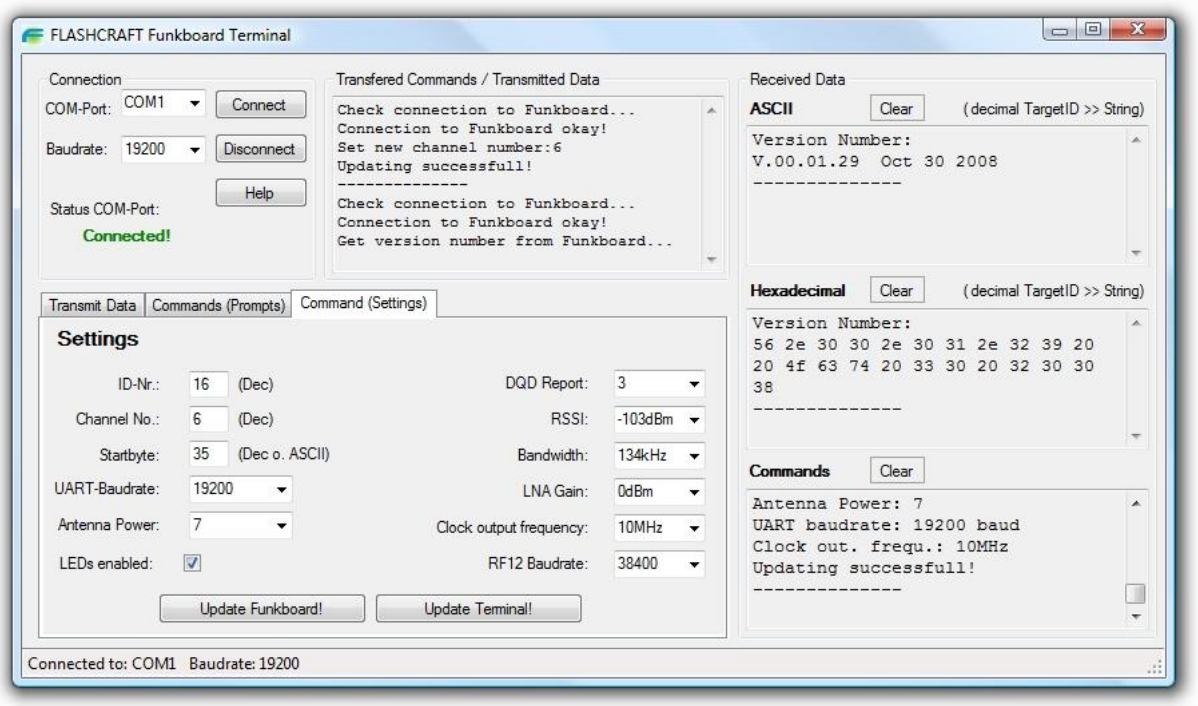
8.1 Allgemeines

Das Projekt enthält neben der Leiterkarte auch ein PC Terminalprogramm. Das Programm ist eine grafische Windows-Anwendung. Über den COM-Port eines PCs und die RS232-UART des Funkboards kann eine Verbindung zwischen beiden Einheiten hergestellt werden. Mit dem Terminalprogramm kann dann das angeschlossene Funkboard getestet und konfiguriert werden.

Dies eignet sich sehr gut um zum Beispiel vor dem Einbau in eine Schaltung ein Funkboard am PC zu testen, zu konfigurieren und dann „fertig“ in die Zielschaltung einzugliedern.

8.2 Details zur Software

Die Software kann unter www.flashcraft.de heruntergeladen werden. Sie ist als ausführbare EXE-Datei sowie als Sourcecode erhältlich. Sie ist in C# geschrieben und nur unter Windows PCs lauffähig. Zum Ausführen wird außerdem das .NET-Framework 3.5 (oder höher) benötigt. Die neueste Version kann kostenlos im MICROSOFT Download Center heruntergeladen werden. Zum Betrieb wird die serielle Schnittstelle (COM-Port) des Computers benötigt. Ein USB zu Seriell Adapter ist jedoch auch möglich.



PC-Terminalprogramm des FLASHCRAFT Funkboard Projekts

8.3 Hardware und externe Beschaltung

Die externe Zusatzbeschaltung beschränkt sich auf ein Minimum. Auf folgendes ist jedoch zu achten:

1. Der **RX-Pin** des Computers wird mit dem **Pin TX232** auf dem Funkboard verbunden

2. Der **TX-Pin** des Computers wird mit dem **Pin RX232** auf dem Funkboard verbunden (sollten RX und TX versehentlich vertauscht worden sein hat das keine Auswirkungen oder Schäden an der Hardware zur Folge)

3. Die **DTR-Leitung** des Computers muss mit dem **MDSEL-Pin** des Funkboards verbunden werden. Die Software steuert über diesen Anschluss den Betrieb im Konfigurations- oder Transceivemodus. **Achtung: Die DTR-Leitung auf gar keinen Fall direkt mit dem MDSEL-Pin verbinden!** Hier ist eine Zusatzbeschaltung notwendig! Die ±12V der DTR-Leitung müssen auf 0V/5V für den MDSEL-Pin gewandelt werden. Hier bietet sich am Besten ein RS232-driver wie der MAX232 an. Dieser kann auch über das Funkboard mit Spannung versorgt werden.

4. Spannungsversorgung für das Funkboard (z.B. über eine Batterie)

5. Der GND-Anschluss des COM-Ports wird mit GND des Funkboards verbunden

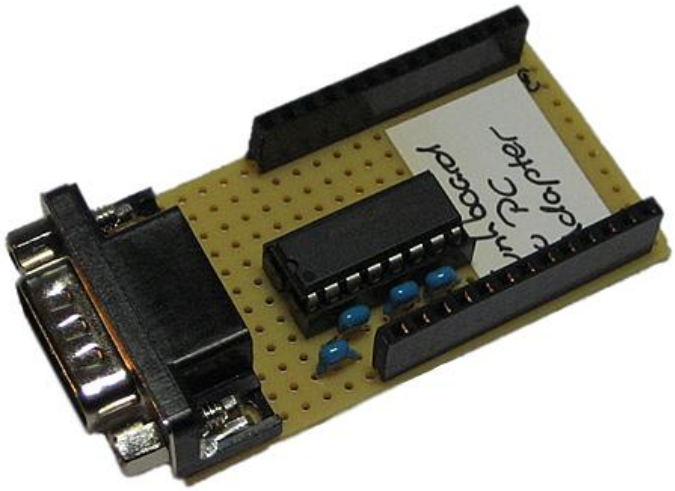
Fertig!

Dies sind alle notwendigen Maßnahmen um das Funkboard betreiben zu können. Es bietet sich an eine kleine Lochrasterplatine (mit dem externen RS232-driver) zu bauen auf die man das Funkboard dann nur noch einstecken muss. Dies birgt die geringsten Risiken einer Fehlverschaltung.

8.4 Beispielschaltung für eine Adapterplatine

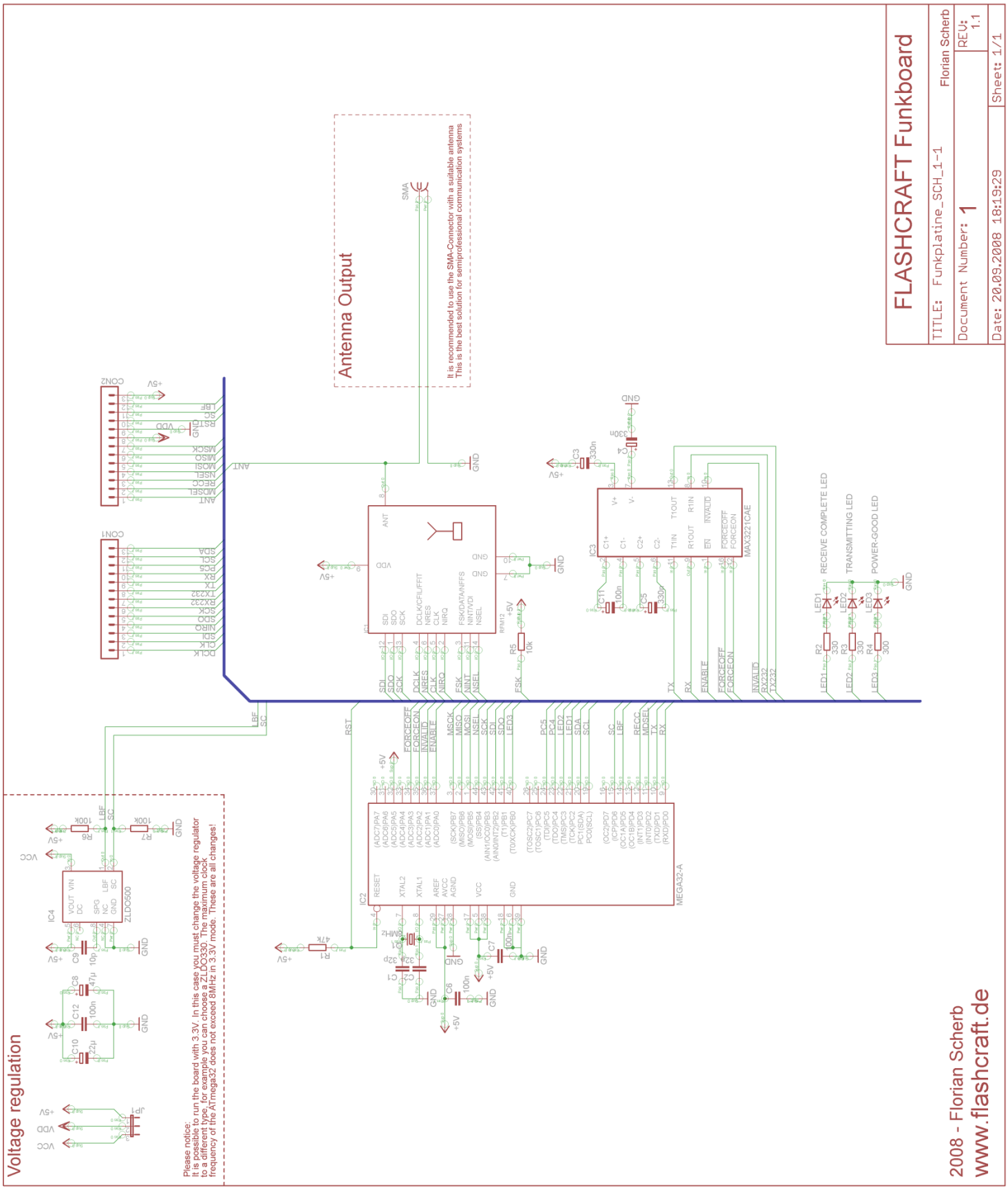
Es bietet sich an eine kleine Lochraster-Adapterplatine zu bauen, die mit einem RS232-Anschluss und einem MAX232-Pegelwandler bestückt ist. Das Funkboard muss dann nur noch auf diesen Adapter gesteckt werden. Ein Schaltungsvorschlag für eine solche Platine ist im ***Adapterplatine PC to Funkboard*** abgebildet.

Hier ein Beispielbild eines solchen Adapters (Spannungsversorgung über Batterie nicht sichtbar)

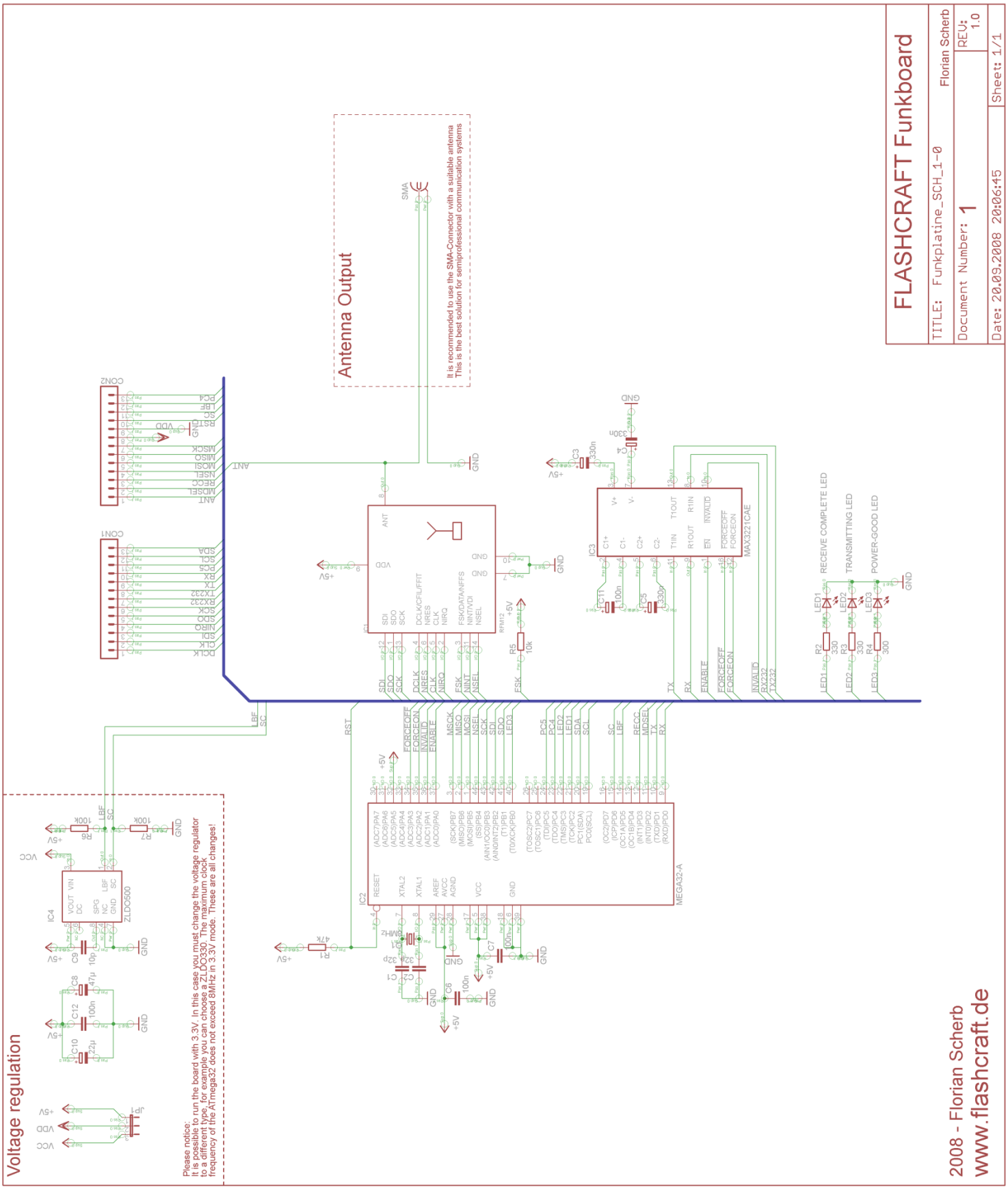


9.Schaltpläne

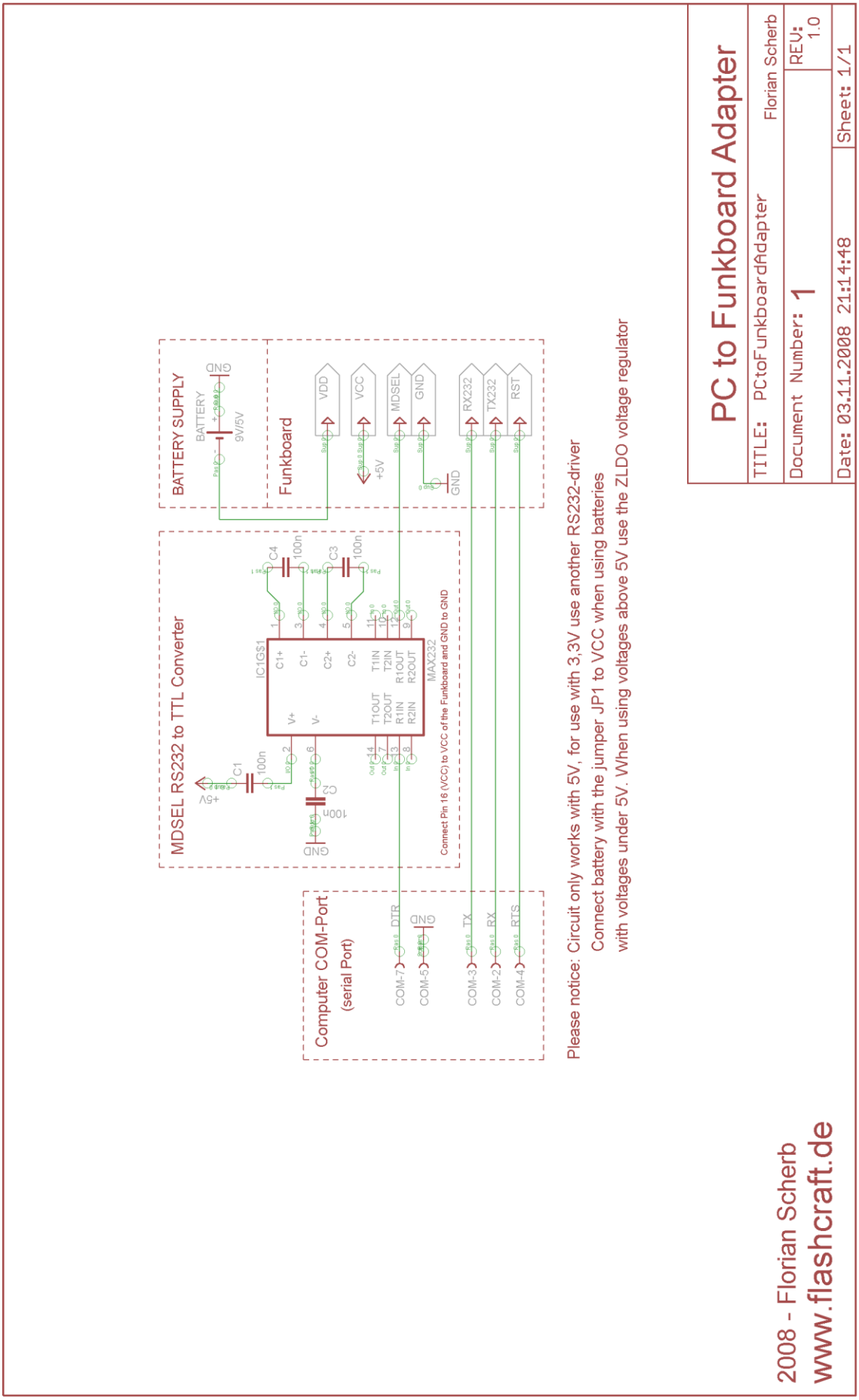
9.1 Funkboard Revision 1.1



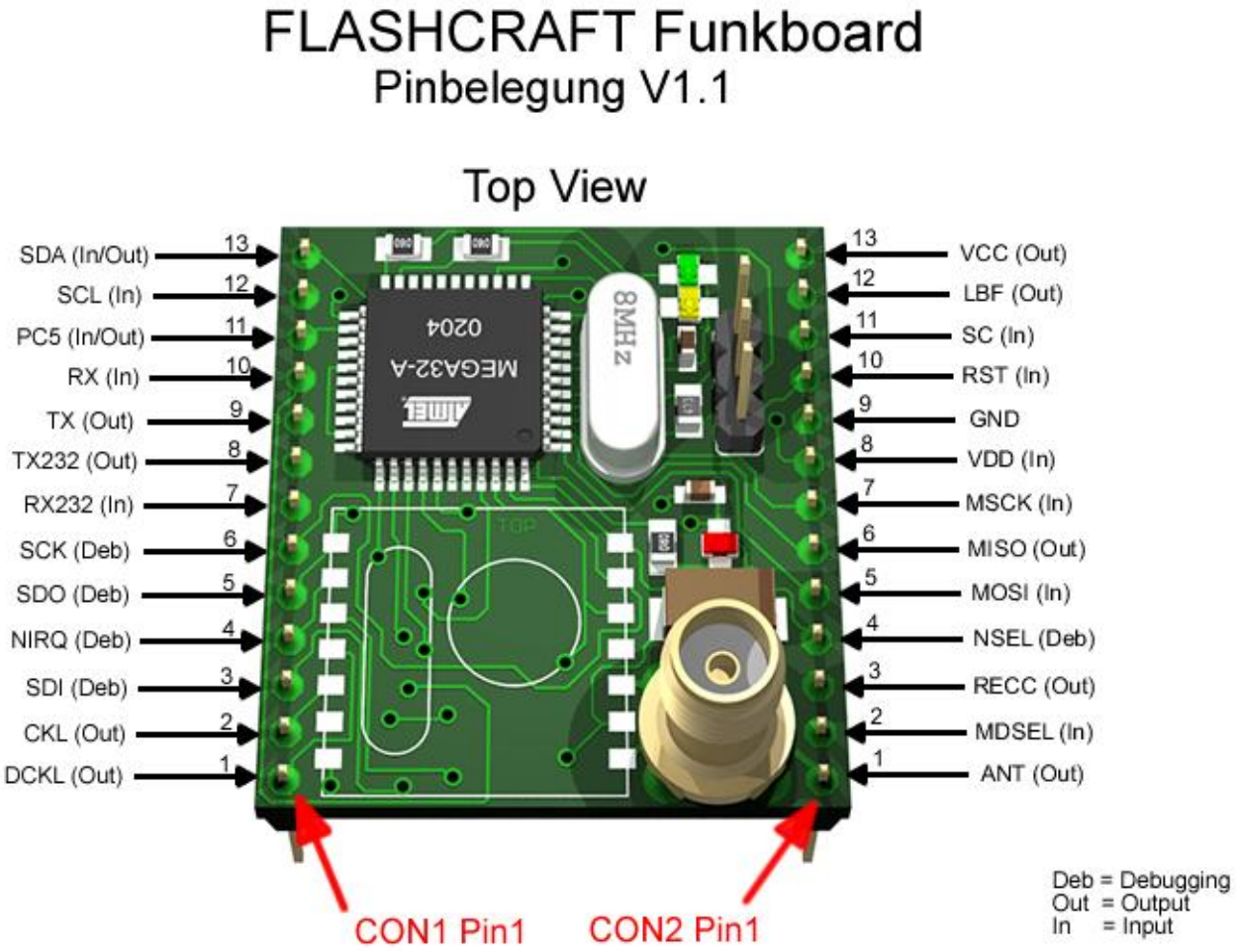
9.2 Funkboard Revision 1.0



9.3 Adapterplatine PC to Funkboard



10. Pinbelegung



10.1 Pinbelegung nach Anschlussnummern

CON1:Pin1 – DCLK (Out) Pin ist direkt mit DCLK des RFM12-Moduls verbunden. Ist auf dem Funkboard unbeschaltet und unbenutzt. Dient dem Debugging.

CON1:Pin2 – CLK (Out) Pin ist direkt mit CLK des RFM12-Moduls verbunden. CLK ist Clock output des RFM12 Moduls. Clock läuft standardmäßig auf 10MHz. Der Takt am CLK-Pin kann im Konfigurationsmodus geändert werden.

CON1:Pin3 – SDI (Deb) Pin ist direkt mit SDI des RFM12-Moduls verbunden. Über SDI werden Daten vom ATmega32 in das RFM12 Modul geschrieben. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON1:Pin4 – NIRQ (Out) Pin ist direkt mit NIRQ des RFM12-Moduls verbunden. NIRQ ist der Interrupt request output Pin des RFM12 Moduls. Er wird nicht verwendet und kann in einer externen Schaltung ausgelesen werden.

CON1:Pin5 – SDO (Deb) Pin ist direkt mit SDO des RFM12-Moduls verbunden. Über SDO werden Daten vom RFM12 Modul zum ATmega32 gesendet. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON1:Pin6 – SCK (Deb) Pin ist direkt mit SCK des RFM12-Moduls verbunden. SCK trägt das Clocksignal für die Kommunikation zwischen ATmega32 und RFM12. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON1:Pin7 – RX232 (In) Eingang für serielle RS232 Datenübertragung. Dieser Pin ist zusammen mit dem TX232 Teil der RS232-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 aktiviert sein.

CON1:Pin8 – TX232 (Out) Ausgang für serielle RS232 Datenübertragung. Dieser Pin ist zusammen mit dem RX232 Teil der RS232-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 aktiviert sein.

CON1:Pin9 – TX (Out) Ausgang für serielle UART Datenübertragung mit TTL-/CMOS-Pegel. Dieser Pin ist zusammen mit dem RX-Pin Teil der UART-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 deaktiviert sein.

CON1:Pin10 – RX (In) Eingang für serielle UART Datenübertragung mit TTL-/CMOS-Pegel. Dieser Pin ist zusammen mit dem TX-Pin Teil der UART-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 deaktiviert sein.

CON1:Pin11 – PC5 (In/Out) Dieser Pin ist ein frei wähl- und programmierbarer I/O-Pin für zukünftige oder benutzereigene Erweiterungen. In Standardsoftware funktionslos.

CON1:Pin12 – SCL (In) Clock des I2C-Busses. Der Takt auf der Clock-Leitung muss von einer externen Schaltung bereitgestellt werden. Achtung: Sowohl SCL als auch SDA Signalleitungen haben keine Pullup-Widerstände.

CON1:Pin13 – SDA (In/Out) Signalleitung des I2C-Busses. Achtung: Sowohl SCL als auch SDA Signalleitungen haben keine Pullup-Widerstände.

CON2:Pin1 – ANT (Out) Antennenanschluss des RFM12-Moduls. Die Antenne kann wahlweise über diesen Pin nach außen geführt werden. Es wird empfohlen onBoard-SMA-Buchse zu verwenden!

CON2:Pin2 – MDSEL (In) Mode-Select-Pin. Pin ist zur Steuerung unbedingt notwendig, unabhängig von der Schnittstelle. Spannungspegel an diesem Pin signalisiert Programm in welchem Modus (Transceive-/Konfigurationsmodus) es sich befindet

CON2:Pin3 – RECC (Out) Receive-Complete-Pin. Spannungspegel an diesem Pin signalisiert ob neue valide Daten über Funk empfangen wurden. Sind neue Daten empfangen worden und im FIFO Eingangspuffer gespeichert worden ist Pin für ca. 10µs auf high (Spike)

CON2:Pin4 – NSEL (Deb) Pin ist direkt mit nSEL des RFM12-Moduls verbunden. NSEL ist der Chipselect-Pin des RFM12 Moduls (low active. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON2:Pin5 – MOSI (In) MOSI (Master out slave in)
Teil des ISP-Programmierinterfaces (MOSI).

CON2:Pin6 – MISO (Out) MISO (Master in slave out)
Teil des ISP-Programmierinterfaces (MISO).

CON2:Pin7 – MSCK (In) Master Clock
Teil des ISP-Programmierinterfaces (SCK).

CON2:Pin8 – VDD (In) Eingangsversorgungspannung. Der Spannungspegel der an diesem Pin angelegt wird geht wahlweise (je nach Stellung von Jumper JP1) über einen Spannungswandler oder liegt direkt an der Schaltung an.

CON2:Pin9 – GND (In) Bezugspotential.

CON2:Pin10 – RST (In) Hardware-Resetpin des ATmega32. Pin wird über einen Pullup auf high gezogen. In diesem Zustand läuft der ATmega32 im normalen Zustand. Wird dieser Pin (kurzzeitig) auf low gezogen kann damit der ATmega32 hardwareseitig resettet werden.

CON2:Pin11 – SC (In) Shutdown-Control-Pin des ZLDO-Spannungswandlers. Pin ist über einen Pullup auf low gezogen. In diesem Zustand ist Spannungswandler aktiv. Wird dieser Pin auf high gezogen wird der Spannungswandler ausgeschaltet. Wird die Spannung, die am VDD-Pin anliegt direkt an die Schaltung angelegt, so muss ZLDO deaktiviert und damit der SC-Pin auf high gesetzt werden.

CON2:Pin12 – LBF (Out) Low-Battery-Flag-Pin des ZLDO-Spannungswandlers. Pin kann als Versorgungsspannungsindikator dienen. Nur relevant wenn Spannungswandler verwendet und aktiv ist. Solange Spannung an diesem Pin gleich der Betriebsspannung ist, ist die angelegte Spannung am VDD-Pin ausreichend um das Modul zu versorgen. Fällt die Spannung auf GND ist die Spannung in naher Zukunft nicht mehr ausreichend.

CON2:Pin13 – VCC (Out) Funkboard Revision 1.1: An diesem Pin liegt die interne Versorgungsspannung des Funkboards an. Somit kann der ZLDO-Spannungswandler genutzt werden um externe Peripherie mit einer Festspannung zu versorgen.

10.2 Pinbelegung nach Gruppen

Spannungsversorgung

CON2:Pin8 – VDD (In) Eingangsversorgungspannung. Der Spannungspegel der an diesem Pin angelegt wird geht wahlweise (je nach Stellung von Jumper JP1) über einen Spannungswandler oder liegt direkt an der Schaltung an.

CON2:Pin9 – GND (In) Bezugspotential.

CON2:Pin13 – VCC (Out) Funkboard Revision 1.1: An diesem Pin liegt die interne Versorgungsspannung des Funkboards an. Somit kann der ZLDO-Spannungswandler genutzt werden um externe Peripherie mit einer Festspannung zu versorgen.

Debugging

CON1:Pin3 – SDI (Deb) Pin ist direkt mit SDI des RFM12-Moduls verbunden. Über SDI werden Daten vom ATmega32 in das RFM12 Modul geschrieben. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON1:Pin5 – SDO (Deb) Pin ist direkt mit SDO des RFM12-Moduls verbunden. Über SDO werden Daten vom RFM12 Modul zum ATmega32 gesendet. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON1:Pin6 – SCK (Deb) Pin ist direkt mit SCK des RFM12-Moduls verbunden. SCK trägt das Clocksignal für die Kommunikation zwischen ATmega32 und RFM12. Pin dient zum Debugging und sollte nicht beschaltet werden!

CON2:Pin4 – NSEL (Deb) Pin ist direkt mit nSEL des RFM12-Moduls verbunden. NSEL ist der Chipselect-Pin des RFM12 Moduls (low active. Pin dient zum Debugging und sollte nicht beschaltet werden!

Schnittstellen

CON1:Pin7 – RX232 (In) Eingang für serielle RS232 Datenübertragung. Dieser Pin ist zusammen mit dem TX232 Teil der RS232-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 aktiviert sein.

CON1:Pin8 – TX232 (Out) Ausgang für serielle RS232 Datenübertragung. Dieser Pin ist zusammen mit dem RX232 Teil der RS232-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 aktiviert sein.

CON1:Pin9 – TX (Out) Ausgang für serielle UART Datenübertragung mit TTL-/CMOS-Pegel. Dieser Pin ist zusammen mit dem RX-Pin Teil der UART-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 deaktiviert sein.

CON1:Pin10 – RX (In) Eingang für serielle UART Datenübertragung mit TTL-/CMOS-Pegel. Dieser Pin ist zusammen mit dem TX-Pin Teil der UART-Schnittstelle. Zur Nutzung dieser Pins muss der MAX3221 deaktiviert sein.

CON2:Pin5 – MOSI (In) MOSI (Master out slave in)
Teil des ISP-Programmierinterfaces (MOSI).

CON2:Pin6 – MISO (Out) MISO (Master in slave out)
Teil des ISP-Programmierinterfaces (MISO).

CON2:Pin7 – MSCK (In) Master Clock
Teil des ISP-Programmierinterfaces (SCK).

CON1:Pin12 – SCL (In) Clock des I2C-Busses. Der Takt auf der Clock-Leitung muss von einer externen Schaltung bereitgestellt werden. Achtung: Sowohl SCL als auch SDA Signalleitungen haben keine Pullup-Widerstände.

CON1:Pin13 – SDA (In/Out) Signalleitung des I2C-Busses. Achtung: Sowohl SCL als auch SDA Signalleitungen haben keine Pullup-Widerstände.

CON2:Pin2 – MDSEL (In) Mode-Select-Pin. Pin ist zur Steuerung unbedingt notwendig, unabhängig von der Schnittstelle. Spannungspegel an diesem Pin signalisiert Programm in welchem Modus (Transceive-/Konfigurationsmodus) es sich befindet

CON2:Pin3 – RECC (Out) Receive-Complete-Pin. Spannungspegel an diesem Pin signalisiert ob neue valide Daten über Funk empfangen wurden. Sind neue Daten empfangen worden und im FIFO Eingangspuffer gespeichert worden ist Pin für ca. 10µs auf high (Spike)

Hardwareseitiges

CON1:Pin2 – CLK (Out) Pin ist direkt mit CLK des RFM12-Moduls verbunden. CLK ist Clock output des RFM12 Moduls. Clock läuft standardmäßig auf 10MHz. Der Takt am CLK-Pin kann im Konfigurationsmodus geändert werden.

CON2:Pin10 – RST (In) Hardware-Resetpin des ATmega32. Pin wird über einen Pullup auf high gezogen. In diesem Zustand läuft der ATmega32 im normalen Zustand. Wird dieser Pin (kurzzeitig) auf low gezogen kann damit der ATmega32 hardwareseitig resettet werden.

CON2:Pin11 – SC (In) Shutdown-Control-Pin des ZLDO-Spannungswandlers. Pin ist über einen Pullup auf low gezogen. In diesem Zustand ist Spannungswandler aktiv. Wird dieser Pin auf high gezogen wird der Spannungswandler ausgeschaltet. Wird die Spannung, die am VDD-Pin anliegt direkt an die Schaltung angelegt, so muss ZLDO deaktiviert und damit der SC-Pin auf high gesetzt werden.

CON2:Pin12 – LBF (Out) Low-Battery-Flag-Pin des ZLDO-Spannungswandlers. Pin kann als Versorgungsspannungsindikator dienen. Nur relevant wenn Spannungswandler verwendet und aktiv ist. Solange Spannung an diesem Pin gleich der Betriebsspannung ist, ist die angelegte Spannung am VDD-Pin ausreichend um das Modul zu versorgen. Fällt die Spannung auf GND ist die Spannung in naher Zukunft nicht mehr ausreichend.

Sonstiges

CON1:Pin1 – DCLK (Out) Pin ist direkt mit DCLK des RFM12-Moduls verbunden. Ist auf dem Funkboard unbeschaltet und unbenutzt. Dient dem Debugging.

CON1:Pin4 – NIRQ (Out) Pin ist direkt mit NIRQ des RFM12-Moduls verbunden. NIRQ ist der Interrupt request output Pin des RFM12 Moduls. Er wird nicht verwendet und kann in einer externen Schaltung ausgelesen werden.

CON2:Pin1 – ANT (Out) Antennenanschluss des RFM12-Moduls. Die Antenne kann wahlweise über diesen Pin nach außen geführt werden. Es wird empfohlen onBoard-SMA-Buchse zu verwenden!

CON1:Pin11 – PC5 (In/Out) Dieser Pin ist ein frei wähl- und programmierbarer I/O-Pin für zukünftige oder benutzereigene Erweiterungen. In Standardsoftware funktionslos.

11. Revisionshistory

An dieser Stelle sind alle fortlaufenden Überarbeitungen von Hardware, Software und Dokumentation aufgelistet. Überarbeitungen in Hardware, Software und Dokumentation werden getrennt behandelt und veröffentlicht. Zwischen den Revisionsangaben dieser Teile besteht also kein direkter Bezug!

11.1 Funkboard Hardware

REVISION	DATUM	MODIFIKATIONEN GEGENÜBER VORGÄNGERREVERSION
Rev. 1.0	12.09.08	Ursprungsversion
Rev. 1.1	21.09.08	Stiftleiste CON2 geändert. Pin13 ist nun mit interner Betriebsspannung VCC anstatt mit PC4 des AVRs verbunden

11.2 AVR Software

REVISION	DATUM	MODIFIKATIONEN GEGENÜBER VORGÄNGERREVERSION
Rev. 1.0.0	12.09.08	Ursprungsversion
Rev. 1.0.1	03.11.08	Überarbeitete und zur Veröffentlichung freigegebene Version
Rev. 1.0.2	23.11.08	Bugfixes im Empfangsbetrieb
Rev. 1.1.0	07.12.08	Kleinere Bugfixes, Änderungen der Checksummen im Transceivemodus

11.3 Terminalprogramm

REVISION	DATUM	MODIFIKATIONEN GEGENÜBER VORGÄNGERREVERSION
Rev. 0.0.8	03.11.08	In Entwicklung, erste Betatests
Rev. 1.0.1	07.11.08	Erste zur Veröffentlichung freigegebene Version
Rev. 1.0.2	23.11.08	Bugfixes am DTR-Pin

11.4 Dokumentation

REVISION	DATUM	MODIFIKATIONEN GEGENÜBER VORGÄNGERREVERSION
Rev. 1.0.0	12.09.08	Ursprungsversion
Rev. 1.0.1	03.11.08	Überarbeitete und zur Veröffentlichung freigegebene Version
Rev. 1.0.2	08.11.08	Abschnitt „Terminalprogramm“ ergänzt
Rev. 1.0.3	09.11.08	LED Vorwiderstände in Schaltplan und Stückliste geändert
Rev. 1.1.0	07.12.08	Völlig überarbeitete und neu aufgelegte Dokumentation

12. TODO-Liste

An dieser Stelle findet sich eine Liste aller Features, die zum aktuellen Erstellungsdatum dieses Skripts und der aktuellen Softwareversion noch nicht lauffähig oder implementiert sind, deren Einbau jedoch noch in naher Zukunft vorgesehen ist.

I2C- und SPI Schnittstelle (Teilweise bereits implementiert, jedoch noch nicht lauffähig)
Error-Handler-Messages (Messages 10 bis 15 noch belegen)
Verschlüsselung von Funkdaten (Noch nicht implementiert)

13. Bekannte Bugs

An dieser Stelle sollen alle Bugs aufgelistet sein, die während dem Betrieb der neuesten Hard- und Software aufgetreten sind, für die jedoch noch kein Update verfügbar ist.

Du hast einen Bug gefunden, der hier nicht aufgelistet ist? Bitte hilf in diesem Fall mit den Code zu verbessern. Nutze dazu einfach das Kontaktformular auf www.flashcraft.de/index.php/kontakt und sende deinen Bug-Report mit einer kurzen Beschreibung und wenn möglich wie sich der Fehler gezielt erzwingen lässt.

13.1 PC Terminalprogramm

- Terminalprogramm sendet zusammengehörigen seriellen Datenstrom hin- und wieder zeitlich sehr „auseinandergezogen“. Dabei kann es passieren, dass ein Datenpaket nicht mehr als zusammenhängend erkannt wird.
- Problem beim Empfang von längeren Datenpaketen über Funk

13.2 AVR C-Code

- Keine Fehler bekannt

14. Stückliste

Alle Bauelemente mit Ausnahme des RFM12S-Moduls können zum Beispiel über Reichelt bezogen werden. Das RFM12S-Modul kann bei Pollin günstig bestellt werden.

Die Platine des FLASHCRAFT Funkboards gibt es (noch) nicht direkt zu kaufen. Auf www.flashcraft.de gibt es jedoch das EAGLE-File zur privaten Verwendung.

SCHALTPLANBEZ.	BAUELEMENTBEZ.	WERT	ANMERKUNG
IC1	RFM12S	-	433MHz oder 868MHz
IC2	Atmega32	TQFP-44	-
IC3[4]	MAX3221	SSOP-16	Andere Derivate mgl.
IC4[5]	ZLDO500	SM-8	ZLDO330 möglich
Q1	Quarz	8MHz	Andere Werte möglich
LED1	Chip-LED rot	0805	-
LED2	Chip-LED gelb	0805	-
LED3	Chip-LED grün	0805	-
JP1	Stiftleiste 3*2,54mm	-	Jumperbrücke
CON1,CON2	Stiftleiste 13*2,54mm	-	Evtl. Buchsenleiste notw.
SMA	Printbuchse o. –Stecker	-	-
JUMPER	2,54mm Jumper	-	-
R1	Widerstand	0805, 47kΩ	-
R2,R3,R4[6]	Widerstand	0805, 470Ω	LED-Vorwiderstände
R5	Widerstand	0805, 10kΩ	-
R6,R7	Widerstand	0805, 100kΩ	-
C1,C2	Keramikkondensatoren	0603, 32pF	Lastkapazitäten für Quarz
C9[5]	Keramikkondensatoren	0803, 10pF	Von ZLDO benötigt
C6,C7,C12	Keramikkondensatoren	0603, 100nF	Abblockkondensatoren (wichtig!)
C3,C4,C5[4]	Tantalkondensatoren	Typ A, 470nF	Von MAX3221 benötigt
C11[4]	Tantalkondensatoren	Typ A, 100nF	Von MAX3221 benötigt
C8[7]	Tantalkondensatoren	Typ D, 47µF	Stabilisierungskondensatoren
C10[7]	Tantalkondensatoren	Typ C, 22µF	Stabilisierungskondensatoren

Müssen alle Bauelemente verwendet werden?

Nein! Es sei hier nochmals erwähnt: Das Funkboard ist modular aufgebaut. Das heist, dass Bauteile die nicht benötigt werden auch einfach nicht bestückt werden. Welche Bauteile dies sind ist in den Fußnoten zusätzlich zum Schaltplan noch einmal erläutert.

[4] Bestücken wenn RS232-UART bzw. MAX3221 verwendet wird
[5] Bestücken wenn Festspannungsregler ZLDOxxx verwendet wird
[6] Vorwiderstände bei V_{CC}=5V, für V_{CC}=3,3V z.B. 330Ω verwenden
[7] Bestücken bei zusätzlicher Stabilisierung der Versorgungsspannung

15. Kontakt, Impressum und Rechtliches

15.1 Kontakt

Diese Dokumentation wurde von Florian Scherb verfasst und publiziert. Kontaktaufnahmen sind jederzeit sehr gerne möglich. Bitte hierfür das Kontaktformular auf meiner Webseite www.flashcraft.de nutzen.

15.2 Haftungsausschluss und Rechtliches

Ungeachtet der Sorgfalt, die auf die Erstellung von Text, Abbildungen und Schaltungen verwendet wurde, kann der Autor für mögliche Fehler und deren Folgen keine juristische Verantwortung oder irgendeine Haftung übernehmen. Der Autor distanziert sich ausdrücklich von jeglichen Schäden, die durch den Gebrauch dieses Dokuments entstehen, seien sie materieller, physischer oder psychischer Art. Die in diesem Dokument wiedergegebenen Gebrauchsnamen, Handelsnamen, Warenbezeichnungen usw. können auch ohne besondere Kennzeichnung Marken sein und als solche den gesetzlichen Bestimmungen unterliegen.

15.3 Lizenz und Weitergabe

Alle Teile des Projekts „FLASHCRAFT Funkboard“, seien es Dokumente, Programme oder andere explizit zugehörige Elemente dürfen zu privaten Zwecken frei verwendet, vervielfältigt und modifiziert werden. Ein Vermerk des Namens des Autors muss dabei jedoch immer gut sichtbar erfolgen.

Es ist ausdrücklich nicht gestatet jegliche Inhalte des Projekts „FLASHCRAFT Funkboard“ zu kommerziellen Zwecken zu nutzen, außer es liegt eine schriftlichte Erlaubnis des Autors vor.