HTL anichstraße
Elektronik · Elektrotechnik · Maschinenbau · Wirtschaftsingenieure

htl
bildung mit zukunft

DIPLOMARBEIT

Amateurfunkpeilsender

mit RFID Teilnehmerregistrierung

Höhere Technische Bundeslehr- und Versuchsanstalt Anichstraße

Abteilung

Elektronik & Technische Informatik

Ausgeführt im Schuljahr 2015/16 von:

Josef Heel 5AHEL
Simon Kaufmann 5AHEL

Betreuer:

DI Ludwig Stonig

Projektpartner: Österreichischer Versuchssenderverband, Tirol

Innsbruck, am 8. April 2016

Abgabevermerk:
Datum:

Betreuer:

Erklärung der Eigenständigkeit der Arbeit

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Ort, Datum Verfasser

Danksagung

An dieser Stelle möchten wir uns bedanken bei allen Personen, die an der Durchführung des Projektes beteiligt waren. Besonderer Dank gilt unserem Betreuer

Dipl.-Ing. Ludwig Stonig (OE7LSH)

der uns stets mit Rat und Tat zur Seite stand, uns seine private Elektronikausstattung nutzen ließ und uns weit über das im Rahmen einer Diplomarbeit übliche Maß hinaus unterstützte. Ebenfalls bedanken möchten wir uns bei unserem Projektpartner und Auftraggeber, dem Landesverband Tirol des Österreichischen Versuchssenderverbandes unter der Leitung von

Ing. Manfred Mauler (OE7AAI)

für die Finanzierung des Projektes, für die unkomplizierte Abwicklung der Bauteilbestellungen und für das in uns gesetzte Vertrauen.

Unser Dank gilt weiters Birgit Stonig (OE7BSI) und Dr. Dietmar Müller für ihre Hilfe beim Korrekturlesen der schriftlichen Arbeit. Schließlich bedanken wir uns bei unseren Familien für die Unterstützung.

Kurzreferat

Ziel dieses Projektes ist die Entwicklung eines universellen, praxistauglichen Fuchsjagdsenders für den Landesverband Tirol des ÖVSVs (Österreichischer Versuchssenderverband). Der Entwurf erfolgt aufbauend auf der Diplomarbeit „ARDF Duoband Transmitter“ von Simon Außerlechner, Clemens Moroder und Gerald Renner an der HTL Anichstraße aus dem Jahre 2006, wobei das Design überarbeitet und teils mit zusätzlichen Funktionen ausgestattet wird.

Der Fuchsjagdsender verfügt über die Möglichkeit zum automatisierten Sendebetrieb, sowohl im 80m- als auch im 2m-Amateurfunkband. Zusätzlich ist eine Einrichtung zur Kontrolle der Antennenanpassung (SWR-Messgerät) beim Auslegen und Abstimmen der Füchse im Gelände vorhanden. Die Registrierung der Teilnehmer während des Wettbewerbs erfolgt mithilfe von RFID-Tags an einem integrierten Lesegerät. Die Konfiguration der Sender vor der Fuchsjagd sowie die Auswertung der gesammelten Teilnehmerdaten danach wird digital über eine PC-Oberfläche durchgeführt.

Das in Hard- und Software ausgearbeitete Design wird an einem Prototyp getestet, wobei sich zeigt, dass große Teile der gestellten Anforderungen erfüllt werden. Stellenweise zeigt sich noch Möglichkeit zur zukünftigen Verbesserung und Weiterentwicklung.

Abstract

The aim of this project is the development of a universal, practicable ARDF transmitter for the Austrian Amateur Radio Association (ÖVSV). The concept is based on the diploma project „ARDF Duoband Transmitter“ by Simon Außerlechner, Clemens Moroder and Gerald Renner at HTL Anichstraße from 2006, with the design revised and partly extended with new functionalities.

The ARDF transmitter is able to transmit automatically both in the 80m- and in the 2m HAM frequency band. Aditionally, a SWR meter makes the process of tuning the antennas easier when placing and installing the transmitters in the terrain. Registration of the participants during the competition is accomplished via RFID-tags in conjunction with an integrated RFID-reader. The configuration of the transmitter prior to the event as well as the analysis of the competitor's data afterwards is carried out digitally via a PC interface.

The elaborated hard- and software design is tested on a prototype, where it is found that most of the requirements are met. In some places, possibilities for future improvements and further development are shown.

Inhaltsverzeichnis

1. Einleitung **17**
1.1. Funktionalität 17
1.2. Platinenaufteilung 18
1.3. Aufgabenteilung im Team 19

I. Hardware **21**

2. Mikrocontroller-Platine **23**
2.1. Übersicht 23
2.2. Mikrocontroller 24
2.3. USB-Interface 27
2.4. Real-Time Clock (RTC) 28
2.5. EEPROM 29
2.6. Spannungsversorgung 30
2.6.1. 3.3V Schaltregler 31
2.6.2. 5V Linearregler 32
2.7. Diverse Schaltungsteile 33
2.8. Schnittstellen 34

3. DDS-HF-Platine **37**
3.1. Übersicht 37
3.2. DDS-Baustein 38
3.2.1. Prinzip der Direct Digital Synthesis (DDS) 38
3.2.2. Anforderungen 39
3.2.3. Auswahl 40
3.2.4. Implementierung 41
3.2.5. DDS-Spannungsversorgung 42
3.2.6. Simulation 42
3.3. Bandumschaltung 80m/2m 44
3.3.1. Anforderungen 44
3.3.2. Konzept und Bauteilauswahl 44
3.3.3. Implementierung 45

3.4. 80m-Leistungsverstärker 46
3.4.1. Anforderungen 46
3.4.2. Konzept und Bauteilauswahl 48
3.4.3. Vorverstärker 49
3.4.4. Endstufe 53
3.4.5. Oberwellenfilter 56
3.4.6. Messungen 62
3.5. 2m-Leistungsverstärker 64
3.5.1. Anforderungen 64
3.5.2. Konzept und Bauteilauswahl 66
3.5.3. Eingangsfilter 67
3.5.4. Vorverstärker 69
3.5.5. Endstufe 73
3.5.6. Oberwellenfilter 74
3.5.7. Messungen 78
3.6. Kühlkörper 80

4. Richtkoppler-Platine **83**
4.1. Übersicht 83
4.2. 80m-Richtkoppler 84
4.3. 2m-Richtkoppler 86
4.4. Messungen 87

5. RFID-Platine **89**
5.1. Übersicht 89
5.2. RFID-Modul 89
5.3. Abstimmeinrichtung 91

6. Weiterführende Überlegungen **93**
6.1. Gehäuse 93
6.2. Akkukriterien 94
6.2.1. Energieverbrauch 94
6.2.2. Akkutypen 95
6.3. Probleme und Verbesserungsmöglichkeiten 96
6.3.1. Nebenaussendungen durch Schaltregler 96
6.3.2. DDS Fehlfunktionen 97
6.3.3. HF-Power-Modul für 2m 98
6.3.4. Abstimmvorgang des 2m-Teils 98
6.3.5. Oberwellenunterdrückung auf 2m 98
6.3.6. Platinenlayout und Abschirmung 99
6.3.7. Störung der SPI-Übertragung zum DDS 99

II. Software 101

7. Auswahl Mikrocontroller 103
7.1. Build-Umgebung 104

8. Übersicht Mikrocontrollersoftware 107
8.1. Bausteinbibliotheken 107
8.2. Höhere Softwareschicht 108
8.3. Konventionen für Softwaremodule 109
8.4. Moduldokumentation 110

9. Softwaremodul DDS 111
9.1. Anforderungen 111
9.2. Überblick DDS-Baustein AD9859 111
9.2.1. Schnittstelle 112
9.2.2. Leitungen 112
9.2.3. Register 113
9.3. Softwareschnittstelle 115
9.4. Implementierung 116
9.4.1. dds_init: 117
9.4.2. dds_write_register und dds_read_register: 118
9.4.3. Funktionen zum Setzen der Einstellungen 120
9.4.4. dds_write_output_ftw 121
9.4.5. dds_write_amplitude 121
9.4.6. Amplitudenmodulation 121

10. Softwarebibliothek TWI/I²C 125
10.1. Anforderungen 125
10.2. Übersicht I²C 125
10.2.1. TWI beim AVR 126
10.3. Software 127
10.3.1. Softwareschnittstelle 127
10.3.2. Implementierung 128

11. Softwaremodul Real-Time Clock 131
11.1. Anforderungen 131
11.2. Überblick Real-Time Clock MCP79410 131
11.2.1. I²C-Adresse 132
11.2.2. Schreibvorgang 132
11.2.3. Lesevorgang 132
11.2.4. Register 132

11.3. Software . . . 133
11.3.1. Softwareschnittstelle . . . 133
11.3.2. Implementierung . . . 135

12. Softwarebibliothek EEPROM 137
12.1. Anforderungen . . . 137
12.2. Überblick EEPROM-Baustein 24AA512 . . . 137
12.2.1. Schreibvorgang . . . 138
12.2.2. Lesevorgang . . . 139
12.3. Software . . . 139
12.3.1. Softwareschnittstelle . . . 139
12.3.2. Implementierung . . . 140

13. Softwaremodul UART 145
13.1. Anforderungen . . . 145
13.2. UART beim AVR . . . 146
13.3. Software . . . 147
13.3.1. Softwareschnittstelle . . . 147
13.3.2. Implementierung . . . 149

14. Softwaremodul Commands 151
14.1. Anforderungen . . . 151
14.1.1. Kommandos zur Konfiguration . . . 151
14.2. Softwareschnittstelle . . . 152
14.3. Implementierung . . . 152
14.3.1. commands_execute: . . . 152

15. Softwarebibliothek Utils 155
15.1. Zweck . . . 155
15.2. Aufgaben . . . 155
15.2.1. String zu Integer Konvertierung . . . 155
15.2.2. Integer zu String Konvertierung . . . 156
15.2.3. Vergleichen von Strings . . . 156
15.2.4. Umwandlung von Kleinbuchstaben in Großbuchstaben . . . 157

16. Softwaremodul Startup 159
16.1. Anforderungen . . . 159
16.2. Softwareschnittstelle . . . 159
16.3. Implementierung . . . 160
16.3.1. Interner EEPROM-Speicher . . . 160
16.3.2. Implementierung der Funktionen . . . 161

17. Softwaremodul Morsen **165**
17.1. Anforderungen . . . 165
17.2. Überblick Morsezeichen . . . 165
17.2.1. Wörter pro Minute - WPM . . . 166
17.3. Softwareschnittstelle . . . 166
17.4. Implementierung . . . 168
17.4.1. Zeitbasis Timer - Interruptroutine . . . 168
17.4.2. morse_start_time / morse_stop_time: . . . 169
17.4.3. morse_start_minute / morse_stop_minute: . . . 169
17.4.4. update_start: . . . 169
17.4.5. is_on_minute / is_off_minute: . . . 170

18. Softwaremodul RFID **171**
18.1. Einführung . . . 171
18.1.1. Anforderungen an das Softwaremodul . . . 171
18.2. Mifare Classic 1K Transponder . . . 172
18.3. Modifikation der Arduino-RFID-Bibliothek . . . 172
18.3.1. Umbau C++ zu C . . . 173
18.3.2. Nachbau der Arduino-Funktionen . . . 174
18.3.3. Reduktion auf benötigte Elemente . . . 175
18.4. Softwareschnittstelle . . . 175
18.5. Implementierung . . . 176
18.5.1. rfid_loop . . . 176
18.5.2. rfid_read_block . . . 177
18.5.3. rfid_write_block . . . 177
18.5.4. rfid_close . . . 177

19. Softwaremodul User **179**
19.1. Anforderungen . . . 179
19.2. Aufteilung des Speichers . . . 179
19.2.1. Transponder . . . 179
19.2.2. EEPROM des Senders . . . 180
19.3. Softwareschnittstelle . . . 180
19.4. Implementierung . . . 181
19.4.1. user_new_tag . . . 181
19.4.2. user_set_id . . . 182
19.4.3. user_write_id . . . 182
19.4.4. user_add_to_history . . . 183
19.4.5. user_write_history . . . 183
19.4.6. user_read_tag . . . 184

20. Softwaremodul Main **185**
20.1. Aufgabe . . . 185
20.2. Weitere Funktionen . . . 185
20.2.1. main_reload . . . 185
20.2.2. main_set_blinking . . . 186

21. PC-Software **187**
21.1. Anforderungen . . . 187
21.2. Wahl der Programmierumgebung . . . 187
21.2.1. Auswahl: Skriptsprache Python mit Toolkit wxWidgets . . . 188
21.3. Programmstruktur . . . 188
21.3.1. Hauptfenster . . . 189
21.3.2. Panel Fox / Programming . . . 191
21.3.3. Panel User . . . 192
21.3.4. Panel Result . . . 193
21.3.5. Dialoge . . . 193
21.3.6. Verbindung zur COM-Schnittstelle . . . 194
21.3.7. Programmvariablen . . . 194
21.4. Python Installation - Test der PC-Software . . . 195
21.4.1. Auslieferung der PC-Software . . . 196

22. Weiterführende Überlegungen und Arbeiten **199**
22.1. Amplitudenmodulation . . . 199
22.2. Energiesparmodus . . . 199
22.3. Fuchshistorie . . . 200

III. Benutzerdokumentation **201**

23. Konfigurationssoftware **203**
23.1. Installationsanleitung / Inbetriebnahme . . . 203
23.1.1. Treiberinstallation . . . 203
23.1.2. PC-Programm starten . . . 203
23.2. Setzen der Fuchsnummer . . . 205
23.3. Konfiguration der Einstellungen für den Wettbewerb . . . 206
23.3.1. Übertragen der Einstellungen . . . 206
23.3.2. Kalibrierung der Quarzfrequenz . . . 207
23.3.3. Abspeichern der Einstellungen am PC . . . 208
23.4. Vorbereiten der RFID-Transponder . . . 208
23.5. Auswerten der Ergebnisse . . . 208

24. Inbetriebnahme und Abgleich **211**
24.1. Schaltregler 211
24.2. Abgleich 80m-Leistungsverstärker 211
24.3. Abgleich 2m-Leistungsverstärker 212

IV. Anhang **225**

A. Verwendete Messgeräte und Entwicklungswerkzeuge **227**
A.1. Messgeräte 227
A.2. Computersoftware 228
A.3. Sonstiges 229

B. Lastenheft **231**

C. Zeitplanung **233**

D. Kostenübersicht **235**

E. Fertigungsdokumentation **237**
E.1. Gesamtschaltpläne 237
E.2. Platinenlayouts 242
E.3. Prototyp 246
E.4. Stückliste 247
E.4.1. Mikrocontroller-Platine 247
E.4.2. DDS-HF-Platine 248
E.4.3. Richtkoppler-Platine 250
E.4.4. RFID-Platine 250
E.4.5. Diverses 251

F. Arbeitsnachweis **253**
F.1. Projektmitglied Josef Heel 253
F.2. Projektmitglied Simon Kaufmann 254

1. Einleitung

Amateur Radio Direction Finding, zu Deutsch Amateurfunkpeilen oder umgangssprachlich auch als Fuchsjagd bezeichnet, ist eine sportliche Aktivität im Amateurfunk, die vom Prinzip mit einem Orientierungslauf zu vergleichen ist. Dazu werden in einem festgelegten Gebiet, häufig in einem Wald, fünf Sender („Füchse“) ausgelegt. Mithilfe von tragbaren Peilantennen versuchen die Teilnehmer, diese Sender schnellstmöglich aufzufinden.

Vom Landesverband Tirol des Österreichischen Versuchssenderverbandes (ÖVSV) besteht Bedarf an einem Satz (fünf Stück) solcher Fuchsjagdsender zur Ausrichtung von Fuchsjagden in Tirol. Im Jahre 2006 wurde an der HTL Anichstraße im Rahmen einer Diplomarbeit von Simon Außerlechner, Clemens Moroder und Gerald Renner bereits ein solcher ARDF-Transmitter entwickelt. Aufbauend auf diesem Projekt soll der Sender überarbeitet und betriebssicherer gemacht werden, um ihn im Feld einsetzen zu können.

1.1. Funktionalität

Sendebetrieb auf 80m und 2m: Fuchsjagden werden üblicherweise auf einem dieser beiden Amateurfunkbänder durchgeführt, der Sender soll für beide einsetzbar sein. Die Signalerzeugung wird mit einem modernen DDS-Baustein der Firma Analog Devices durchgeführt, um eine frequenzstabile Aussendung mit vergleichsweise geringem Aufwand zu erreichen. Das DDS-Signal muss anschließend auf entsprechende Ausgangsleistung verstärkt werden, um es per Antenne abstrahlen zu können. Zur Abstimmung dieser Antenne wird mithilfe eines Richtkopplers (jeweils für 80m und 2m) das Stehwellenverhältnis auf der Sendeleitung erfasst und bei Bedarf angezeigt.

Zeitnehmung: Die zu einer Fuchsjagd ausgelegten (fünf) Sender müssen jeweils abwechselnd zu genau festgelegten Zeiten senden, ohne sich dabei zu überlappen. Um die Gefahr des Auseinanderdriftens der einzelnen Zeitnehmungen zu vermeiden, verfügt jeder Fuchs über eine batteriegepufferte Echtzeituhr. So ist

sichergestellt, dass alle Sender über die genaue Tageszeit verfügen, welche auch für die Registrierung der Teilnehmer benutzt wird.

Anwesenheitsbestätigung: Findet ein Teilnehmer einen Peilsender, so muss er dies später beweisen können. Traditionell werden dafür Prägezangen verwendet, mit welchen der Teilnehmer eine Markierung auf seiner Kontrollkarte anbringt. In diesem Fall soll die Prägezange durch RFID-Chips abgelöst werden, welche die Teilnehmer in die Nähe des im Fuchssender integrierten Lesegeräts führen. Dadurch kann die genaue Zeit des Eintreffens festgehalten werden. Die Speicherung der Benutzerdaten erfolgt auf einem nichtflüchtigen Speicherbaustein (EEPROM).

Konfiguration: Die Einstellungen über das gewählte Frequenzband, Start- und Endzeitpunkt der Fuchsjagd, die ID des jeweiligen Gerätes usw. werden vom Veranstalter, beispielsweise am Vortag, getätigt. Dies erfolgt über eine Konfigurationsumgebung am PC, mit der der Fuchssender über eine USB-Schnittstelle programmiert wird.

1.2. Platinenaufteilung

Grundsätzlich wird entschieden, die einzelnen Komponenten des Fuchssenders auf vier Platinen mit der Größe von jeweils einer halben Europakarte (100 mm x 80 mm) aufzuteilen, je nach Aufgabe bzw. Funktion. Diese einzelnen Karten sind über Flachbandkabel miteinander verbunden. Auf diese Weise wird ein wesentlich kompakterer Aufbau erreicht, als es bei einer einzelnen Platine der Fall wäre.

Mikrocontroller-Platine:	Mikrocontroller, Real-Time Clock, EEPROM, USB-Schnittstelle, Spannungsregler
DDS-HF-Platine:	DDS-Baustein, 80m-Leistungsverstärker, 2m-Leistungsverstärker
Richtkoppler-Platine:	Messung der Ausgangsanpassung mit 80m- und 2m-Richtkoppler
RFID-Platine:	RFID-Modul zur Benutzerregistrierung, SWR-Anzeige mit Test-Sende-Taster

Tabelle 1.1.: Platinenaufteilung

Ein Blockschaltbild des gesamten Fuchssenders ist in Abbildung 1.1 auf der nächsten Seite dargestellt.

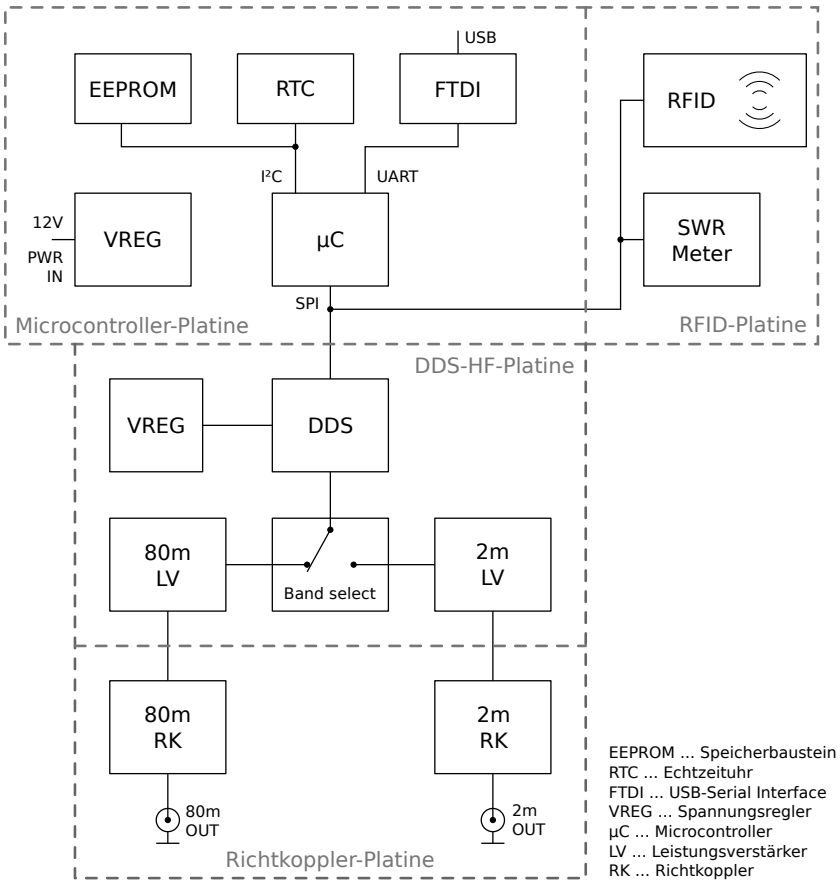


Abbildung 1.1.: Gesamt-Blockschaltbild des Fuchssenders

1.3. Aufgabenteilung im Team

Josef Heel ist hauptverantwortlich für die Hardware des Senders. Dies umfasst den Schaltungsentwurf sowie die Erstellung der Platinenlayouts, den Entwurf der HF-Leistungsverstärker und der Richtkoppler, weiters die Durchführung der dazugehörigen Messungen, die Auslegung der Spannungsversorgung, sowie die Auswahl der Bauteile und Komponenten (letzteres zum Teil mit Simon Kaufmann).

Simon Kaufmann ist hauptverantwortlich für die Software der Diplomarbeit. Dies umfasst die Programmierung der Mikrocontroller-Firmware, bestehend aus der Ansteuerung des DDS-Bausteins, der RFID-Teilnehmerregistrierung, der Programmierschnittstelle vom PC aus, sowie den Funktionen in Verbindung mit der Echtzeituhr. Die Konfigurationsoberfläche am PC ist ebenfalls Teil seines Aufgabenbereiches.

Teil I.

Hardware

2. Mikrocontroller-Platine

2.1. Übersicht

Auf dieser Platine befinden sich neben dem Mikrocontroller (ATmega644) selbst das USB-Interface zur Konfiguration über einen PC, eine Real-Time Clock zur Zeitnehmung, sowie ein nichtflüchtiger Speicherbaustein (EEPROM) für Konfigurations- und RFID-Benutzerdaten. Auch die Spannungsversorgung für die digitalen Komponenten (3.3 V und 5 V) ist auf dieser Platine untergebracht.

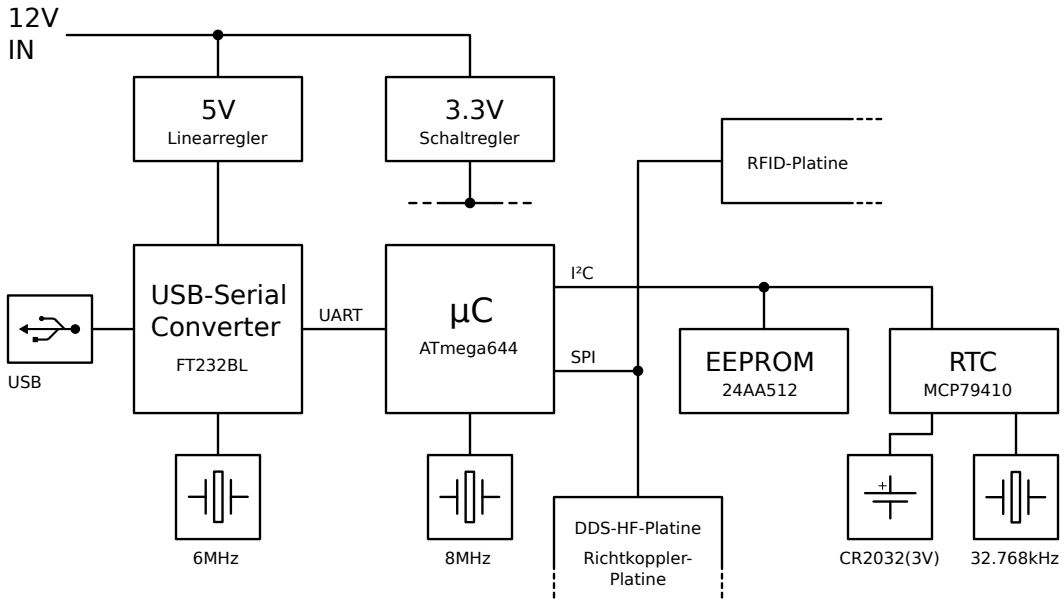


Abbildung 2.1.: Blockschaltbild Mikrocontroller-Platine

2.2. Mikrocontroller

Die zentrale Steuerung des Fuchssenders erfolgt durch einen ATmega644-Mikrocontroller der Firma ATMEL (vgl. *ATmega 644 Microcontroller Datasheet* 2012). Da die I/O-Leitungen des DDS-Bausteins und des RFID-Moduls nur mit 3.3 V-Pegeln betrieben werden können, soll zur Vermeidung von Pegelwandlern der Mikrocontroller ebenfalls mit dieser Spannung versorgt werden. Die Beschaltung des Mikrocontrollers ist in Abbildung 2.2 dargestellt.

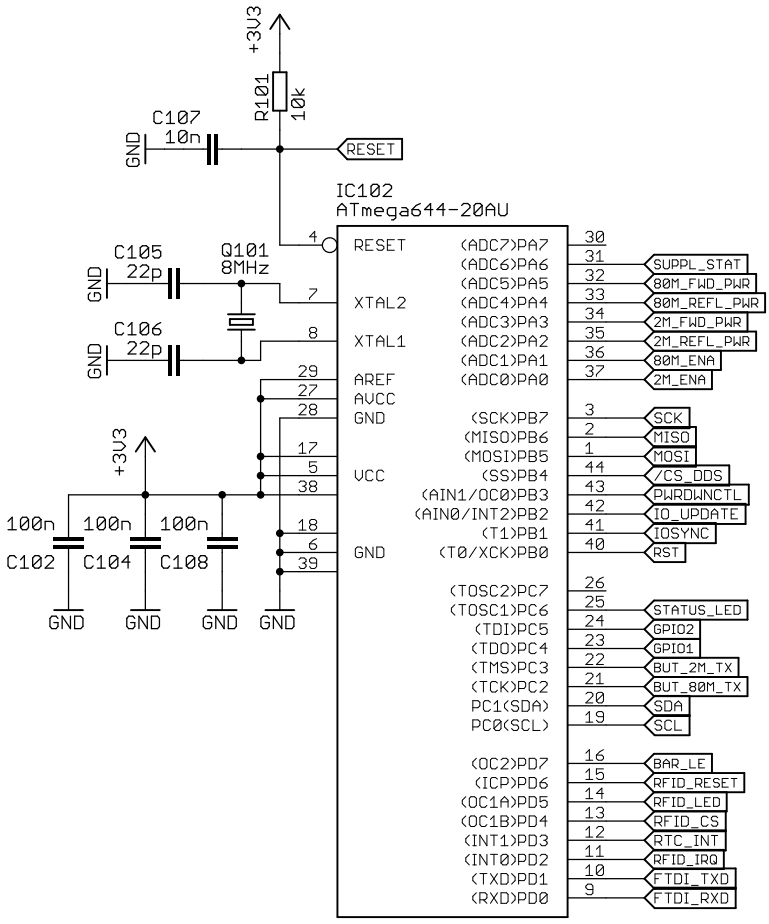


Abbildung 2.2.: Beschaltung Mikrocontroller

Zur Taktversorgung des ATmega644 wird ein externer 8 MHz-Quarz benutzt, da ein für diese Controller sonst typischer 16 MHz- oder 20 MHz-Quarz nur bei 5 V Versorgungsspannung möglich ist. Die Kondensatoren $C105$ und $C106$ sind der

Empfehlung des Datenblattes entnommen (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 30).

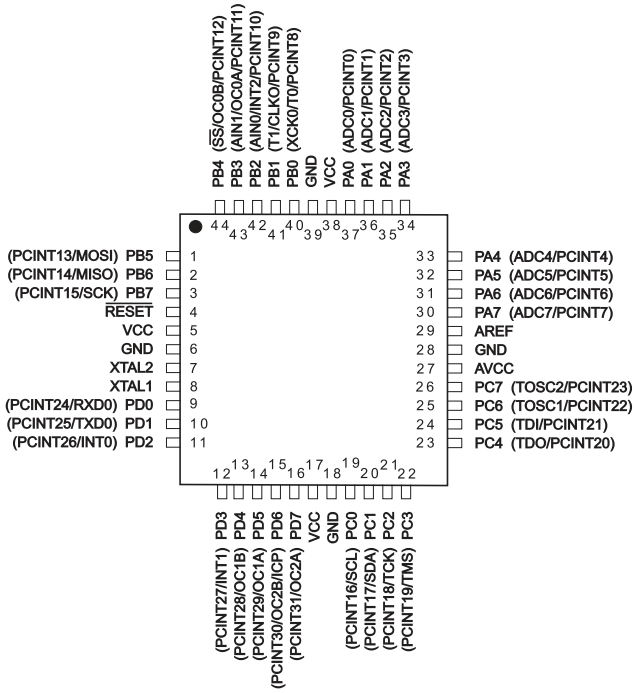


Abbildung 2.3.: Pinbelegung des TQFP-44 Gehäuses, aus dem Datenblatt

Pin			Label	Beschreibung
PA0	Out	D	`2M_ENA`	Aktivierung der Spannungsversorgung des 2m-Leistungsverstärkers + Umschaltung der PIN-Diode
PA1	Out	D	`80M_ENA`	Aktivierung der Spannungsversorgung des 80m-Leistungsverstärkers + Umschaltung der PIN-Diode
PA2	In	A	`2M_REFL_PWR`	Rücklaufende Leistung im 2m-Band
PA3	In	A	`2M_FWD_PWR`	Vorlaufende Leistung im 2m-Band
PA4	In	A	`80M_REFL_PWR`	Rücklaufende Leistung im 80m-Band
PA5	In	A	`80M_FWD_PWR`	Vorlaufende Leistung im 80m-Band
PA6	In	A	`SUPPLY_STAT`	Aktuelle Akkuspannung (geteilt)
PA7	-	-	-	(unbeschaltet)
PB0	Out	D	`RST`	DDS-Reset (zur Initialisierung)
PB1	Out	D	`IOSYNC`	Neuinitialisierung der seriellen Verbindung zum DDS

PB2	Out	D	IO_UPDATE	Neue SPI-Kommandos werden vom DDS übernommen
PB3	Out	D	PWRDWNCTL	Versetzt den DDS in einen Stromsparmodus
PB4	Out	D	$\overline{\text{CS\_DDS}}$	Chip Select des DDS (SPI-Bus)
PB5	Out	D	MOSI	Master → Slave Datenleitung des SPI-Busses
PB6	In	D	MISO	Slave → Master Datenleitung des SPI-Busses
PB7	Out	D	SCK	Taktleitung des SPI-Busses
PC0	Out	D	SCL	Taktleitung des I^2C-Busses
PC1	I/O	D	SDA	Datenleitung des I^2C-Busses
PC2	In	D	BUT_80M_TX	Taster zur Aktivierung des Abstimm-Modus auf 80m
PC3	In	D	BUT_2M_TX	Taster zur Aktivierung des Abstimm-Modus auf 2m
PC4	-	D	GPIO1	Für spätere Erweiterungen auf RFID-Platine geführt
PC5	-	D	GPIO2	(ebenso)
PC6	Out	D	STATUS_LED	LED zu Debugzwecken
PC7	-	-	-	(unbeschaltet)
PD0	In	D	FTDI_RXD	UART-Empfangsleitung vom USB-Serial Converter
PD1	Out	D	FTDI_TXD	UART-Sendeleitung zum USB-Serial Converter
PD2	In	D	RFID_IRQ	Interrupt vom RFID-Modul (aktuell unbenutzt)
PD3	In	D	RTC_INT	Interrupt durch die Real-Time Clock
PD4	Out	D	RFID_CS	Chip Select des RFID-Moduls (SPI-Bus)
PD5	Out	D	RFID_LED	Anzeige, dass RFID-Tag korrekt gelesen wurde
PD6	Out	D	RFID_RESET	Initialisierung des RFID-Moduls
PD7	Out	D	BAR_LE	Latch-Enable Eingang des Schieberegisters für die SWR-Anzeige

Tabelle 2.1.: Pinbelegung Mikrocontroller

Zur Programmierung des ATmega644 ist ein 10-poliger ISP-Header vorhanden (siehe Abbildung 2.4 auf der nächsten Seite), belegt nach Empfehlung von Atmel (vgl. *AVR Hardware Design Considerations* 2015, Seite 6-7). Diese Schnittstelle ist zum Flashen der AVR-Mikrocontroller sehr geläufig, verschiedene Hersteller bieten Programmieradapter mit USB-Anschluss an. In diesem Fall muss besonders darauf geachtet werden, dass der verwendete Adapter einen 3.3 V-Modus besitzt, da der Mikrocontroller auf dieser Spannung betrieben wird.

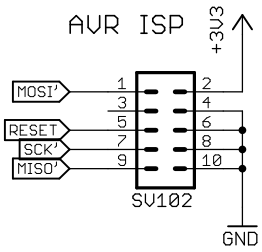


Abbildung 2.4.: ISP-Programmierschnittstelle des μC

2.3. USB-Interface

Zur Konfiguration des Fuchses (Start- und Stoppzeit, Bandauswahl, Sendefrequenz etc.) und zur Auswertung der im EEPROM gespeicherten Daten (siehe 2.7) nach der Fuchsjagd wird eine Verbindung zum PC benötigt. Da eine RS232-Schnittstelle auf modernen Computern immer seltener zu finden ist, soll diese Verbindung über USB hergestellt werden.

Eine Möglichkeit wäre die Verwendung eines Mikrocontrollers mit hardwaremäßig implementierter USB-Schnittstelle (z.B. ATmega32U2). Dabei könnte sich der Fuchssender beispielsweise als HID (Human Interface Device) enummerieren und so Daten austauschen.

In diesem Projekt fällt die Entscheidung allerdings auf einen seperaten USB-Serial-Converter, den FT232BL von FTDI. Dieser wird am PC als virtueller COM-Port erkannt und kann so einfach von verschiedenen Programmiersprachen angesprochen werden. Dadurch entfällt praktisch der Entwicklungsaufwand des USB-Interfaces sowohl PC-seitig (Treiber) als auch mikrocontrollerseitig. Der benötigte Treiber wird auf der Website des Herstellers angeboten (vgl. *Virtual COM Port Drivers - FTDI Ltd.* 2016). Aus Sicht des Mikrocontrollers handelt es sich dabei lediglich um eine UART-Schnittstelle, welche in Hardware unterstützt wird. Die Beschaltung des Bausteins ist in Abbildung 2.5 auf der nächsten Seite dargestellt.

Der FT232BL wird im „Self Powered Modus with 3.3 V logic supply“ betrieben. Die dazu vorgeschlagene Beschaltung wird im Wesentlichen übernommen. Der FT232BL benötigt zum Betrieb einen 6 MHz Quarz (Q102) mit entsprechenden Belastungskapazitäten ($C116$ und $C117$) (vgl. *FT232BL USB-Serial Converter Datasheet* 2011, Seite 20, 22).

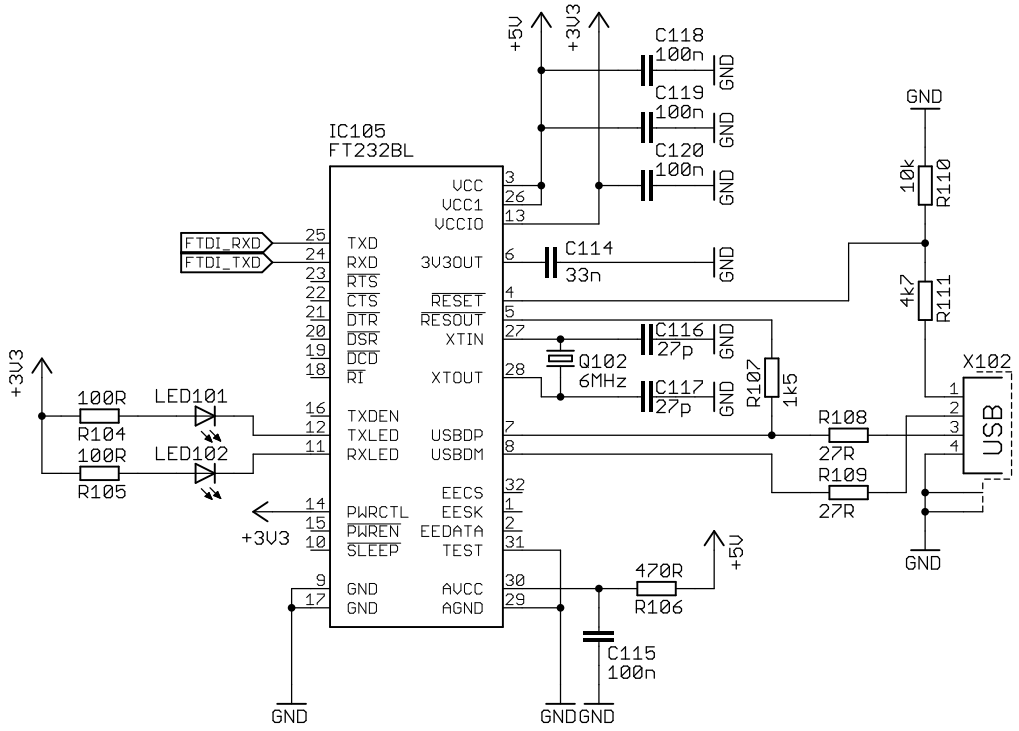


Abbildung 2.5.: Beschaltung FT232 USB-Serial Converter

LED101 und LED102 signalisieren eine aktuell stattfindende Datenübertragung. Bei einer Betriebsspannung von 3.3 V werden diese mit ca. 15 mA betrieben (laut „Absolute Maximum Ratings“ sind bis zu 24 mA möglich).

2.4. Real-Time Clock (RTC)

Zum Ausrichten einer Fuchsjagd werden üblicherweise fünf Fuchssender ausgelegt, welche gleichzeitig zu einer vorher eingestellten Zeit ihren Betrieb aufnehmen müssen. Die Zeitspanne zwischen Programmierung der Sender und Start der Fuchsjagd kann mitunter einen Tag betragen. Während der aktiven Zeit wechseln sich die Füchse im Minutentakt ab. Zusätzlich muss jeder Fuchs jeweils die Uhrzeit aufzeichnen, zu der sich ein Teilnehmer registriert.

Dazu wird eine Real-Time Clock von Microchip benutzt (MCP79410). Der Baustein besitzt einen internen Kalender, stellt also einen Zeitpunkt in Sekunden,

Minuten, Stunden, Tagen, Monaten und Jahren dar. Zusätzlich verfügt die besagte Real-Time Clock über die Möglichkeit, an programmierbaren Zeitpunkten einen Alarmausgang (MFP) zu schalten und so beim Mikrocontroller einen Interrupt auszulösen (vgl. *MCP79410 Battery-Backed Real-Time-Clock Datasheet* 2014). Dadurch kann ein ständiges Abfragen bis zum Startzeitpunkt der Fuchsjagd vermieden werden. Eine Pufferbatterie ermöglicht das Weiterlaufen der Uhrzeit auch bei getrennter Stromversorgung.

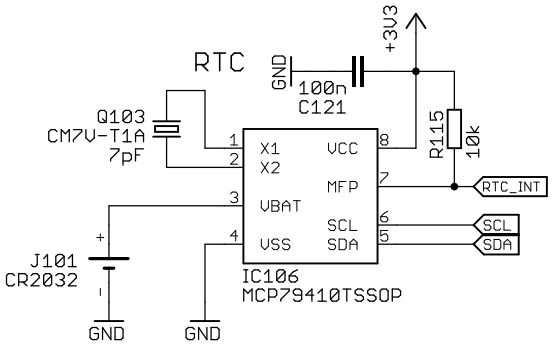


Abbildung 2.6.: Beschaltung Real-Time Clock

Die Beschaltung der Real-Time Clock ist in Abbildung 2.6 dargestellt. Der Uhrenquarz (32.768 kHz) soll nach Empfehlung im Datenblatt eine Belastungskapazität von 6 pF bis 9 pF aufweisen (vgl. *MCP79410 Battery-Backed Real-Time-Clock Datasheet* 2014, Seite 14), deshalb wird hier ein CM7V-T1A 7 pF eingesetzt (vgl. *CM7V-T1A Tuning Fork Crystal Datasheet* 2010).

Als Backupbatterie wird eine Lithium-Knopfzelle (CR2032) benutzt, welche den unabhängigen Betrieb der Uhr für theoretisch ca. 25 Jahre sicherstellt (1 µA Stromverbrauch bei 225 mAh Batteriekapazität). Praktisch wird diese Zeitspanne aufgrund der Selbstentladung der Batterie vermutlich nicht erreicht werden.

2.5. EEPROM

Während einer Fuchsjagd fallen mit jedem Teilnehmer, der sich per RFID-Tag registriert, Daten an (ID, Timestamp, etc.). Diese müssen auf jedem Fuchs gespeichert werden, und dürfen auch bei Ausfall der Stromversorgung nicht verloren gehen. Hierfür wird ein serielles EEPROM (24AA512, siehe Abbildung 2.7 auf der nächsten Seite) eingesetzt (vgl. *24AA512 Serial EEPROM Datasheet* 2010), welches über den I^2C-Bus mit dem Mikrocontroller verbunden ist. Dabei wird der

größte Baustein der Serie benutzt (512kBit), da weniger Speicher preislich kaum einen Vorteil bringt und so ausreichend Platz, auch für spätere Erweiterungen, zur Verfügung steht.

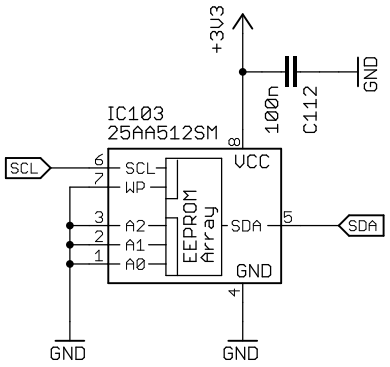


Abbildung 2.7.: Beschaltung EEPROM

Durch die drei Eingänge `A0`, `A1` und `A2` können die letzten Bits der I^2C-Adresse eingestellt werden, um mehrere EEPROMs auf dem gleichen Bus betreiben zu können. Da in diesem Projekt nur ein EEPROM benutzt wird, sind alle drei Adressleitungen mit Masse verbunden. Ebenso wird mit der `WP` (Write Protect) Leitung vorgegangen, da sowohl Lese- als auch Schreibzugriffe notwendig sind.

2.6. Spannungsversorgung

Bereits durch die Aufgabenstellung steht fest, dass die Betriebsspannung für den gesamten Fuchs 12 V Gleichspannung beträgt. Dadurch stehen eine Vielzahl von Spannungsquellen zur Verfügung, um den Sender zu betreiben (Bleigelakku, NiMH-Akkupacks, 3-Zellen-LiPo[1]). Auch lassen sich die HF-Endstufen so ohne zusätzlichen Spannungswandler betreiben, was sich positiv auf den Wirkungsgrad auswirkt, da hier die meiste Leistung benötigt wird.

Der Mikrocontroller und die Digitalelektronik sollen mit 3.3 V betrieben werden, da der ausgewählte DDS-Baustein (I/O-Leitungen) und das RFID-Modul beide mit 3.3 V betrieben werden können (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 4) (vgl. *Neuftech Mifare RC522 RFID Module für Arduino* 2016, Produktbeschreibung). Der FT232 USB-Serial-Wandler muss hingegen

[1] 11.1 V nominal

mit 5 V versorgt werden, bietet aber die Möglichkeit, dass die I/O-Pins 3.3 V-kompatibel arbeiten (vgl. *FT232BL USB-Serial Converter Datasheet* 2011, Seite 15). Es bietet sich an, die 1.8 V für die Versorgung des DDS-Bausteins mittels Linearregler aus der 3.3 V-Schiene zu gewinnen, siehe 3.2.4 auf Seite 41.

Der Stromverbauch auf 3.3 V wird in etwa abgeschätzt:

Komponente	**Stromverbrauch**
Mikrocontroller	9 mA (bei 8 MHz)
DDS Analog + Digital	100 mA (171 mW bei 1.8 V)
RFID-Modul	26 mA (max.)
EEPROM	5 mA (beim Schreiben)
RTC	3 mA (beim Schreiben)
USB-Serial Converter	0.2 mA (auf IO-Supply-Pin)
Gesamt	143.2 mA (max.)

Tabelle 2.2.: Abschätzung Stromverbrauch

2.6.1. 3.3V Schaltregler

Aufgrund des höheren Stromverbrauchs ($\approx$ 150 mA) auf der 3.3 V-Schiene und dem größeren Spannungsunterschied zwischen $U_\mathrm{B} = 12\,\mathrm{V}$ und 3.3 V wird hier ein Schaltwandler eingesetzt. Die Wahl fällt auf die fixe 3.3 V-Version des LM2674 von Texas Instruments. Dieser erlaubt einen Ausgangsstrom von max. $I_\mathrm{L} = 500\,\mathrm{mA}$ bei einem Wirkungsgrad von ca. $\eta \approx 87\%$ (vgl. *LM2674 500mA Step-Down Voltage Converter Datasheet* 2014, Seite 6, Fig. 5).

Der Eigenstrombedarf (bei $I_\mathrm{L} = 0$) beträgt typ. $I_\mathrm{Q} \approx 2.5\,\mathrm{mA}$ (vgl. *LM2674 500mA Step-Down Voltage Converter Datasheet* 2014, Seite 5). Dies mag auf den ersten Blick viel erscheinen, ist es allerdings bei genauerer Betrachtung nicht. Die Zeit zwischen Auslegen der Füchse im Gelände (Einschalten der Stromversorgung) und Beginn der Fuchsjagd wird nicht länger als ein Tag sein, und in dieser Zeit entläd der Ruhestrom des Schaltreglers den Akku um ca. 60 mAh, was bei einer Akkukapazität von mehreren Amperestunden nicht ins Gewicht fällt.

Die Beschaltung des LM2674 ist in Abbildung 2.8 auf der nächsten Seite dargestellt. Bei der Auswahl der Induktivität $L101$, der Schaltdiode $D102$, sowie Ein- und Ausgangskondensator wird nach den Empfehlungen im Datenblatt vorgegangen (vgl. *LM2674 500mA Step-Down Voltage Converter Datasheet* 2014, Seite 11-17).

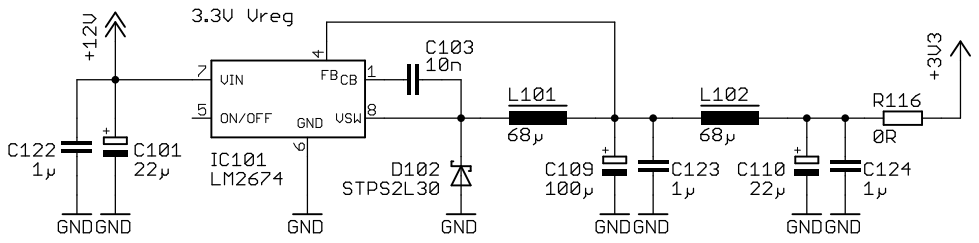


Abbildung 2.8.: 3.3 V Schaltregler

Zur zusätzlichen Glättung der Ausgangsspannung des Schaltwandlers wird ein LC-Tiefpass ($L102$ und $C110$ parallel $C124$) nachgeschaltet, dessen Grenzfrequenz

$$f_{\mathrm{g}} = \frac{1}{2\pi\sqrt{L102 \cdot C110||C124}} \approx 4.02\,\mathrm{kHz} \tag{2.1}$$

weit unter der Schaltfrequenz des LM2674 (225 kHz … 275 kHz) liegt und diese somit aus der Ausgangsspannung filtert (vgl. *LM2674 500mA Step-Down Voltage Converter Datasheet* 2014, Seite 5)

$R116$ dient lediglich dazu, den Schaltregler von der 3.3 V-Schiene trennen zu können, um ihn bei der Inbetriebnahme unabhängig testen zu können.

2.6.2. 5V Linearregler

Eine 5 V-Versorgung wird lediglich für den FT232BL USB-Serial Converter benötigt. Da dieser nur bei verbundenem USB-Kabel aktiv ist (nur kurz bei der Konfiguration vom PC aus), wird ein Linearregler mit geringem Ruhestrom ausgewählt. Der MCP1755S-5002 (5V-Version) besitzt einen Ruhestrom von I_{Q} = typ. 68 µA und reicht für den Betriebsstrom des FT232BL von typ. 25 mA leicht aus (vgl. *MCP1755 300mA High-Performance LDO Datasheet* 2012, Seite 5). Die Beschaltung des Bausteins ist in Abbildung 2.9 auf der nächsten Seite dargestellt.

Die Stützkondensatoren $C111$ und $C113$ sind aus der Standardbeschaltung entnommen (vgl. *MCP1755 300mA High-Performance LDO Datasheet* 2012, Seite 4).

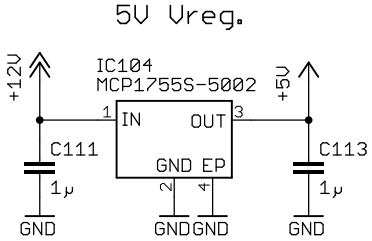


Abbildung 2.9.: 5V Linearregler

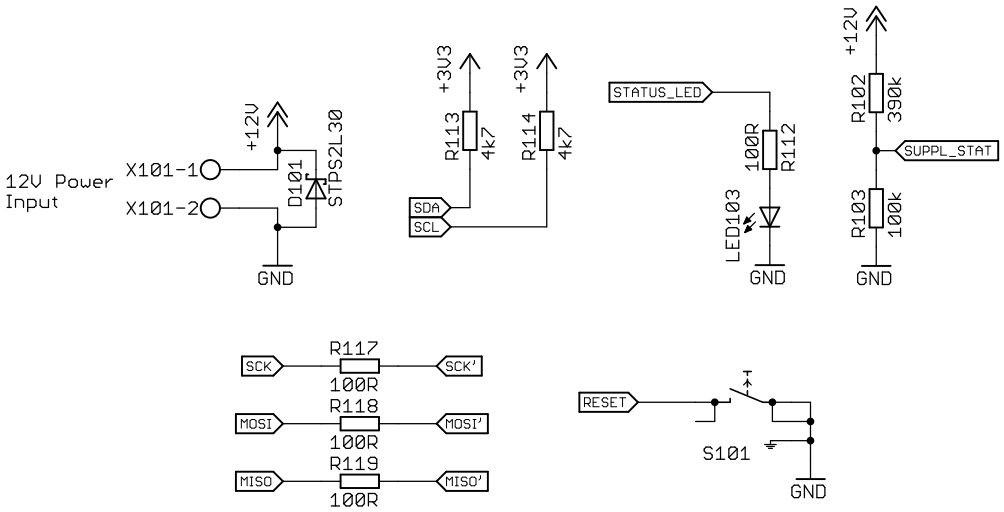


Abbildung 2.10.: Diverse Schaltungsteile

2.7. Diverse Schaltungsteile

Versorgungsklemme: Über diese wird der gesamte Fuchssender mit 12 V versorgt. Die Schottky-Diode $D101$ dient als Verpolungsschutz. Mit einer Strombelastbarkeit von 2 A bietet diese allerdings nur Schutz bei einem Netzgerät mit Strombegrenzung, nicht jedoch bei einem Akku (vgl. *STPS2L30 Schottky Diode Datasheet* 2007, Seite 1).

I²C-Pullups: $R113$ und $R114$ sind für den Betrieb des I²C-Busses notwendig, da die Busteilnehmer die Clock- und Datenleitung jeweils nur auf Masse schalten (Open-Collector Ausgang).

Betriebsspannungsteiler: Mithilfe von $R102$ und $R103$ wird die Betriebsspannung herabgeteilt auf

$$U_{\mathrm{R103}} = U_{\mathrm{B}} \frac{R102}{R102 + R103} \approx 2.44\,\mathrm{V} \tag{2.2}$$

Dadurch kann U_{B} über einen Analog-Digital-Eingang des Mikrocontrollers gemessen werden, sodass dieser abhängig vom Ladezustand des Akkus reagieren kann. Der Spannungsteiler verursacht einen Ruhestrom von nur

$$I_{\mathrm{Q}} = \frac{U_{\mathrm{B}}}{R102 + R103} \approx 24.5\,\mu\mathrm{A} \tag{2.3}$$

was bei einem Akku mit einer Kapazität von mehreren Amperestunden praktisch zu vernachlässigen ist (theoretisch ca. 23 Jahre bis zur vollständingen Entladung eines 5 Ah-Akkus, Selbstentladung abgesehen).

SPI-Trennwiderstände: Die drei SPI-Leitungen `SCK`, `MOSI` und `MISO` werden von den Mikrocontroller-Pins direkt zum DDS verbunden. In der „Abzweigung“ zum RFID-Verbindungsstecker und zur ISP-Programmierschnittstelle des AVR sind hingegen 100 Ω-Widerstände ($R117$, $R118$ und $R119$) eingeschleift.

Störungen, welche durch Einkopplung der HF-Leistung in ein langes Programmierkabel oder in die RFID-Platine entstehen, werden so daran gehindert, die Übertragung zum DDS zu beeinträchtigen. Ohne die Widerstände kam es immer wieder zu Fehlern in der Datenübertragung bei gleichzeitigem Sendebetrieb, besonders im 2m-Band.

Reset-Button und Status-LED: Diese werden im späteren Betrieb des Fuchses nicht mehr verwendet, sind jedoch bei der Softwareentwicklung eine nützliche Einrichtung.

2.8. Schnittstellen

Die Verbindung zwischen Mikrocontroller-Platine und DDS-HF- sowie Richtkoppler-Platine erfolgt über ein 20-poliges Flachbandkabel. Für die RFID-Platine wird ein separates 16-poliges Flachbandkabel benutzt, da diese je nach Gehäuse mitunter weg vom restlichen Fuchssender montiert wird. So wird ein unnötiges Verlängern von nicht auf der RFID-Platine benötigten Datenleitungen vermieden.

Die nachfolgende Beschreibung der einzelnen Pins deckt sich größtenteils mit der Pinbelegung des Mikrocontrollers (siehe 2.2 auf Seite 24), wird aber der Übersichtlichkeit halber hier nochmals ausgeführt.

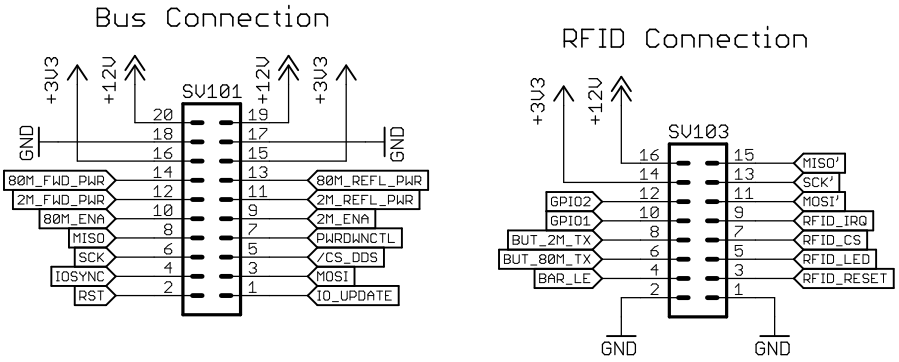


Abbildung 2.11.: Flachbandkabelverbindungen

Verbindung DDS-HF- und Richtkoppler-Platine

Nr.	Label	Beschreibung
1	IO_UPDATE	Neue SPI-Kommandos werden vom DDS übernommen
2	RST	DDS-Reset (zur Initialisierung)
3	MOSI	Master → Slave Datenleitung des SPI-Busses
4	IOSYNC	Neuinitialisierung der seriellen Verbindung zum DDS
5	$\overline{\text{CS\_DDS}}$	Chip Select des DDS (SPI-Bus)
6	SCK	Taktleitung des SPI-Busses
7	PWRDWNCTL	Versetzt den DDS in einen Stromsparmodus
8	MISO	Slave → Master Datenleitung des SPI-Busses
9	2M_ENA	Aktivierung der Spannungsversorgung des 2m-Leistungsverstärkers + Umschaltung der PIN-Diode
10	80M_ENA	Aktivierung der Spannungsversorgung des 80m-Leistungsverstärkers + Umschaltung der PIN-Diode
11	2M_REFL_PWR	Rücklaufende Leistung im 2m-Band
12	2M_FWD_PWR	Vorlaufende Leistung im 2m-Band
13	80M_REFL_PWR	Rücklaufende Leistung im 80m-Band
14	80M_FWD_PWR	Vorlaufende Leistung im 80m-Band
15	+3V3	3.3 V Versorgungsspannung

16	+3V3	(ebenso)
17	GND	Masse
18	GND	(ebenso)
19	+12V	12 V Versorgungsspannung
10	+12V	(ebenso)

Tabelle 2.3.: Flachbandkabel zur DDS-HF- und Richtkoppler-Platine

Verbindung RFID-Platine

Nr.	Label	Beschreibung
1	GND	Masse
2	GND	(ebenso)
3	RFID_RESET	Initialisierung des RFID-Moduls
4	BAR_LE	Latch-Enable Eingang des Schieberegisters f. SWR-Anzeige
5	RFID_LED	Anzeige, dass RFID-Tag korrekt gelesen wurde
6	BUT_80M_TX	Taster zur Aktivierung des Abstimm-Modus auf 80m
7	RFID_CS	Chip Select des RFID-Moduls (SPI-Bus)
8	BUT_2M_TX	taster zur Aktivierung des Abstimm-Modus auf 2m
9	RFID_IRQ	Interrupt ausgelöst vom RFID-Modul (derzeit unverwendet)
10	GPIO1	Für spätere Erweiterungen auf RFID-Platine geführt
11	MOSI’	Master → Slave Datenleitung des SPI-Busses
12	GPIO2	Für spätere Erweiterungen auf RFID-Platine geführt
13	SCK’	Taktleitung des SPI-Busses
14	+3V3	3.3 V Versorgungsspannung
15	MISO’	Slave → Master Datenleitung des SPI-Busses
16	+12V	12 V Versorgungsspannung

Tabelle 2.4.: Flachbandkabel zur RFID-Platine

3. DDS-HF-Platine

3.1. Übersicht

Auf der DDS-HF-Platine befinden sich alle Komponenten zur Hochfrequenzerzeugung und -verstärkung. Dies beginnt beim DDS-Baustein, welcher mikrocontrollergesteuert das Sendesignal im gewünschten Band erzeugt. Dieses Signal wird dann mithilfe einer Umschalteinrichtung an den jeweiligen HF-Leistungsverstärker weitergegeben. Ein solcher ist für das 80m-Band und für das 2m-Band vorhanden, in denen üblicherweise Fuchsjagden ausgerichtet werden. Diese Verstärker bestehen ihrerseits wieder aus einem Vorverstärker, einer Endstufe und einem Oberwellenfilter zur Unterdrückung unerwünschter Aussendungen. Schließlich wird das „fertige“ Sendesignal über Koaxialkabel an die Richtkoppler-Platine weitergegeben.

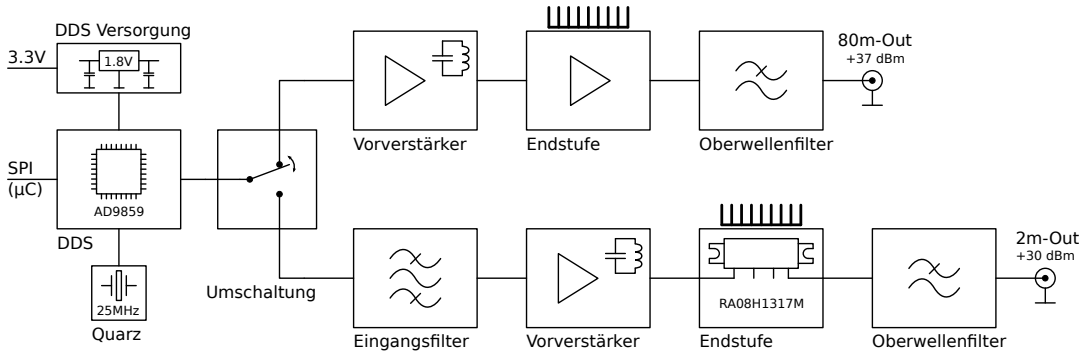


Abbildung 3.1.: Blockschaltbild der DDS-HF-Platine

Die Verbindung zur Mikrocontroller-Platine ist über ein 20-poliges Flachbandkabel ausgeführt. Über dieses findet auch die Energieversorgung der DDS-HF-Platine statt. Zur Wärmeabfuhr der beiden HF-Leistungsverstärker ist ein Kühlkörper vorhanden.

3.2. DDS-Baustein

3.2.1. Prinzip der Direct Digital Synthesis (DDS)

Die Direct Digital Synthesis, zu Deutsch direkte digitale Synthese, bringt gegenüber analogen Verfahren der Signalerzeugung wesentliche Vorteile mit sich. So kann eine beliebige Frequenz von praktisch Gleichstrom bis zu einer Maximalfrequenz mit sehr hoher Frequenzauflösung (z.B. max. 200 MHz bei 32-Bit Auflösung enspricht ca. 46 mHz-Schritten) erzeugt werden. Die Eingabe des gewünschten Frequenzwertes erfolgt digital, was die Steuerung per Mikrocontroller erheblich vereinfacht (gegenüber Kapazitätsdioden etc.). Die Frequenzstabilität des erzeugten Signals wird dabei nur durch die Taktquelle des Systems bedingt, wofür häufig ein Quarz eingesetzt wird.

Abbildung 3.2 (in Anlehnung an *Direct Digital Synthesis* 2014) zeigt die wesentlichen Bestandteile eines DDS-Generators, wobei die Bitbreiten der einzelnen Busse aus dem Datenblatt des verwendeten Bausteins (siehe 3.2.3 auf Seite 40) entnommen sind.

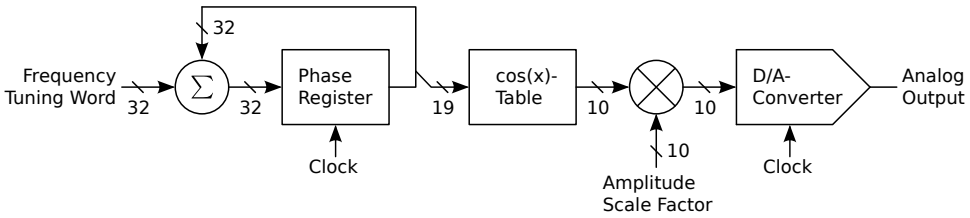


Abbildung 3.2.: DDS-Prinzip Blockschaltbild, modifiziert nach Wikipedia „Direct Digital Synthesis“

Kernbestandteil ist der sogenannte „Phasenakkumulator“, bestehend aus einem Phasenregister (Phase Register) und einem Addierer ($\sum$). Bei jedem Taktzyklus wird zum aktuellen Wert im Phasenregister das „Frequency Tuning Word“ dazuaddiert. Mit dem Überlauf des Phasenregisters beginnt eine neue Periode des Ausgangssignals. Dies bedeutet, je größer das „Frequency Tuning Word“, desto schneller wird das Phasenregister hinaufgezählt sowie zum Überlauf gebracht, und desto höher ist die Frequenz des Ausgangssignals.

Die Werte des Phasenregisters werden auf eine Cosinustabelle (cos(x)-Table) gegeben, welche die „Rampe“ des hinaufzählenden Registers in eine cos-Funktion (Schwingung) umsetzt. Der gesamte Wertebereich entspricht einer Schwingungsperiode von 0 bis 2π.

Dabei werden nur die höchstwertigen 19 Bit des Phasenregisters benutzt, da aufgrund der begrenzten Anzahl von Amplitudenwerten (10 Bit) eine noch feinere Aufteilung der Phasenschritte nicht sinnvoll wäre, es wird schließlich in einem gewissen Bereich immer der gleiche Amplitudenwert ausgegeben. Wird das Phasenregister hingegen sehr schnell inkrementiert (z.B. innerhalb weniger Taktzyklen zum Überlauf gebracht), so werden in der Cosinustabelle bei jedem Phasenschritt einige Amplitudenwerte übersprungen. Dadurch weicht das Ausgangssignal vom idealen Cosinusverlauf ab, es entstehen Ober- und Nebenschwingungen (Spurious), welche herausgefiltert werden müssen.

Diese Zahlenwerte werden nun mit einem „Amplitude Scale Factor“ multipliziert ($\otimes$), um die Ausgangsamplitude einstellen zu können. Schließlich wird der Wertestrom in einem Digital-Analog-Wandler (D/A-Converter) in ein analoges Signal umgewandelt und ausgegeben.

3.2.2. Anforderungen

Die Anforderungen in diesem Projekt an den DDS sind primär, dass beide Frequenzbänder

80m-Band:	3.5 MHz . . . 3.8 MHz
2m-Band:	144 MHz . . . 146 MHz

direkt, d.h. ohne zusätzliche Frequenzvervielfacher erzeugt werden können. Dadurch soll der Aufbau vereinfacht werden, sowie der Abgleichvorgang von Vervielfacherstufen entfallen.

Weiters muss es in irgendeiner Weise möglich sein, das Ausgangssignal in der Amplitude zu modulieren, da laut offiziellen IARU ARDF Rules im 2m-Band der Träger mit einem Ton amplitudenmoduliert sein muss (vgl. *Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1* 2015, Seite 10).

Als Schnittstelle zum Mikrocontroller wird eine serielle gegenüber einer parallelen Verbindung bevorzugt, da sich Mikrocontroller und DDS auf zwei getrennten Platinen befinden, und dadurch weniger Datenleitungen über Verbindungsstecker geführt werden müssen. Ein möglichst einfaches Konzept im Bezug auf benötigte Versorgungsspannungen sowie Systemtakterzeugung ist natürlich von Vorteil.

Zuletzt sollte der DDS-Baustein in einem Gehäusetyp verfügbar sein, der sich noch per Hand verarbeiten (löten) lässt (z.B. SOIC, TSSOP, TQFP). Ein rein für Automatenbestückung geeignetes Gehäuse (z.B. BGA) würde den Vorgang

der Prototypenherstellung sowie der Fertigung weiterer Füchse wesentlich komplizierter und teurer gestalten.

3.2.3. Auswahl

Unter obigen Anforderungen kommen zwei Bausteine der Firma Analog Devices in die nähere Auswahl, Tabelle 3.1 zeigt einen kurzen Vergleich.

Merkmal	AD9859	AD9958
Kanäle:	1	2
Samplerate max.:	400 MSPS	500 MSPS
DAC Auflösung:	10 Bit	10 Bit
μC-Interface:	SPI	SPI
Modulation:	Shaped on/off	16-stufige AM, FM, PM
Taktversorgung:	Crystal Oscillator + PLL	Crystal Oscillator + PLL
Versorgung:	1.8 V core, 3.3 V I/O	1.8 V core, 3.3 V I/O
Gehäuse:	48-TQFP	56-LFCSP (ähnl. QFN)
Preis:	ca. € 20	ca. € 32

Tabelle 3.1.: Vergleich DDS-Bausteine

(vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009) und (vgl. *AD9958 Direct Digital Synthesizer Datasheet* 2013)

In diesem Projekt fällt die Entscheidung auf den AD9859, hauptsächlich wegen dem geringeren Preis und dem bedrahteten Gehäuse (TQFP), welches einfacher per Hand zu verarbeiten ist als das LFCSP des anderen DDS. Dazu kommt, dass der AD9859 bereits älter ist als der AD9958, und somit mehr Informationsmaterial im Internet dazu vorhanden ist, was mitunter nützlich sein könnte.

Nachteilig ist allerdings, dass der ausgewählte Chip nur über eine „Shaped on/off“ Modulationseinrichtung verfügt. Dies bedeutet, dass lediglich „weich“ (Rampe) zwischen „Signal an“ und „Signal aus“ umgeschaltet werden kann, was für die Anforerungen in diesem Projekt nicht ausreicht. Folglich muss die Modulation in Software, d.h. durch andauerndes Neuschreiben des Amplitudenwertes durchgeführt werden.

Der Unterschied zwischen den Sampleraten ist hingegen unwesentlich, da der geforderte Frequenzbereich von beiden Bausteinen abgedeckt wird. Es besitzen beide einen integrierten Oszillator, sodass zur Taktversorgung lediglich ein Quarz

angeschlossen werden muss. Für die Spannungsversorgung wird neben der 3.3 V-Schiene, welche auch den Mikrocontroller versorgt, nur ein separater 1.8 V-Regler benötigt.

3.2.4. Implementierung

Die Beschaltung des AD9859 wird großteils vom Evaluations-Board übernommen (siehe Abbildung 3.3), welches Analog Devices zu diesem Baustein anbietet (vgl. *AD9859 Evaluation Board Schematics* 2004).

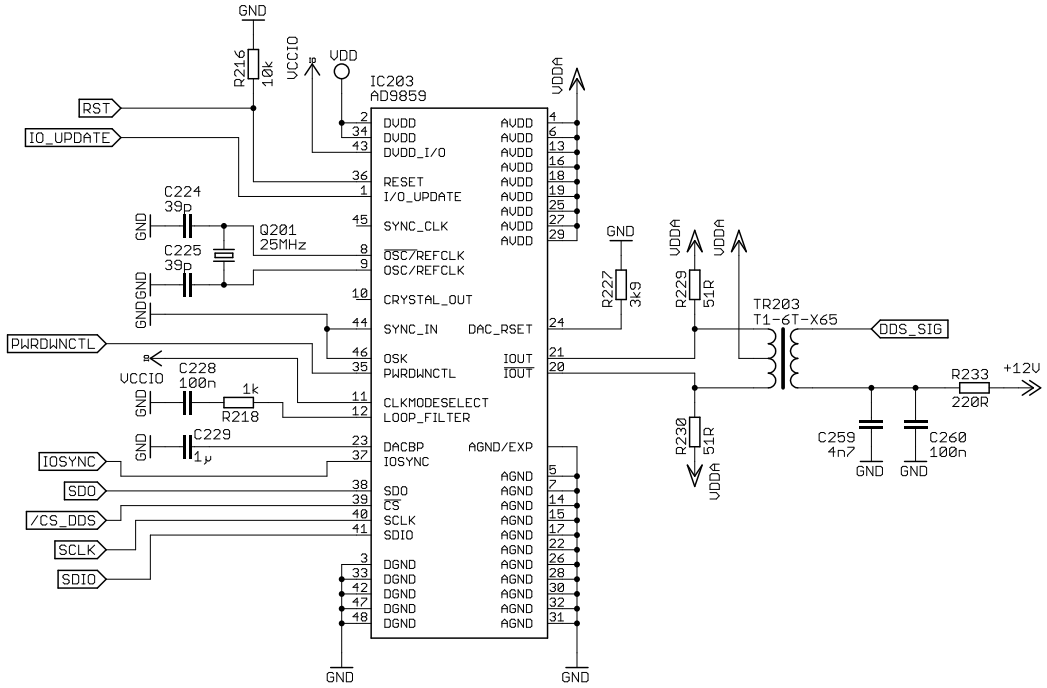


Abbildung 3.3.: Beschaltung des DDS-Bausteins

Anstelle der externen Taktversorgung auf dem Evaluations-Board wird hier ein 25 MHz Quarz eingesetzt, wodurch in Verbindung mit dem internen Phase-Locked-Loop des DDS die maximale Taktfrequenz von 400 MHz erzeugt wird.

Als Ausgangsübertrager wird ein T1-6T-X65 von Mini Circuits benutzt, welcher in 50 Ω-Systemen für Frequenzen bis 300 MHz eingesetzt werden kann (vgl. *T1-6T-X65 RF Transformer Datasheet* 2016). Dem Ausgangssignal wird ein Gleichspannungsoffset von 12 V überlagert (Sekundärwicklung), welcher für die Signalumschaltung über PIN-Dioden benötigt wird, siehe Abschnitt 3.3.3 auf Seite 45.

Damit auf diesem Weg keine Störungen in den Signalpfad gelangen können, wird die 12 V-Versorgung mit einem Tiefpass, gebildet aus $R233$, $C259$ und $C260$, abgeblockt. Dieser verfügt über eine Grenzfrequenz von

$$f_{\mathrm{g}} = \frac{1}{2\pi RC} = \frac{1}{2\pi \cdot R233 \cdot (C259 + C260)} \approx 6.9\,\mathrm{kHz}, \tag{3.1}$$

die weit unter den beiden Betriebsfrequenzen (80m-Band und 2m-Band) liegt. Es muss auch bedacht werden, dass der Strom I_{D}, welcher für die PIN-Dioden-Umschaltung benötigt wird, über $R233$ fließen muss. Da dies $I_{\mathrm{D}} \approx 5\,\mathrm{mA}$ sind (siehe 3.3.3 auf Seite 45), ergibt sich folgender Spannungsabfall:

$$U_{\mathrm{R233}} = R233 \cdot I_{\mathrm{D}} = 220\,\Omega \cdot 5\,\mathrm{mA} \approx 1.1\,\mathrm{V} \tag{3.2}$$

Für den PIN-Diodenstrom fällt dies nicht ins Gewicht.

Im Schaltplan 3.3 auf der vorherigen Seite sind der Übersichtlichkeit halber einige Abblockkondensatoren (4.7 nF und 100 nF) nicht dargestellt, dazu siehe Gesamtschaltpläne E.1 auf Seite 237.

3.2.5. DDS-Spannungsversorgung

Die Betriebsspannung des AD9859 ist 1.8 V, lediglich die I/O-Pins können aus Kompatibilitätsgründen mit 3.3 V versorgt werden (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 6). Diese 1.8 V werden mithilfe eines MCP1755-1802 Low-Dropout-Linearreglers der Firma Microchip bereitgestellt (siehe Abbildung 3.4 auf der nächsten Seite). Mit einem Maximalstrom von 300 mA ist dieser leicht ausreichend für den Stromverbrauch des DDS (Analog + Digital: ca. 100 mA, entspricht 171 mW bei 1.8 V).

$C203$ und $C207$ sind aus der Standardbeschaltung des Datenblattes übernommen (vgl. *MCP1755 300mA High-Performance LDO Datasheet* 2012, Seite 4). Zusätzlich wird mithilfe von zwei LC-Tiefpässen analoge und digitale Versorgung getrennt (`VDDA` und `VDD`), um Störungen vom Digitalteil (Schaltflanken etc.) in den Analogteil und somit in das Ausgangssignal des DDS-Bausteins zu minimieren.

3.2.6. Simulation

Analog Devices bietet auf seiner Website ein Simulationstool zu seinen DDS-Bausteinen, so auch zum AD9859, an (siehe *Design Tools: ADIsimDDS* 2016).

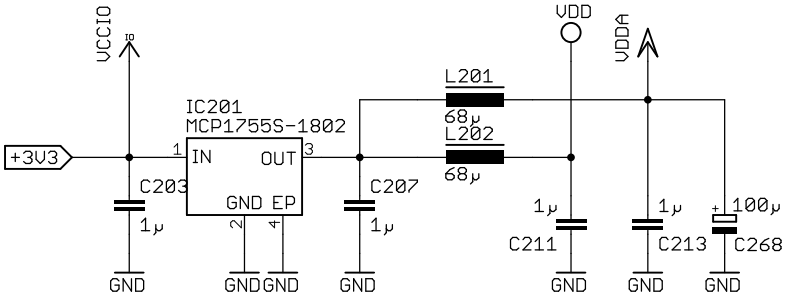


Abbildung 3.4.: 1.8 V Spannungsregler

Mithilfe dieses Tools kann das Spektrum des DDS-Ausgangssignals bei gegebener Taktfrequenz und gewünschter Ausgangsfrequenz berechnet bzw. simuliert werden.

80m-Band:	
DDS:	AD9859
Ref. Clock Frequency:	25 MHz
Desired Output Frequency:	3.55 MHz
Ref. Clock Multiplier:	16x

2m-Band:	
DDS:	AD9859
Ref. Clock Frequency:	25 MHz
Desired Output Frequency:	144.7 MHz
Ref. Clock Multiplier:	16x

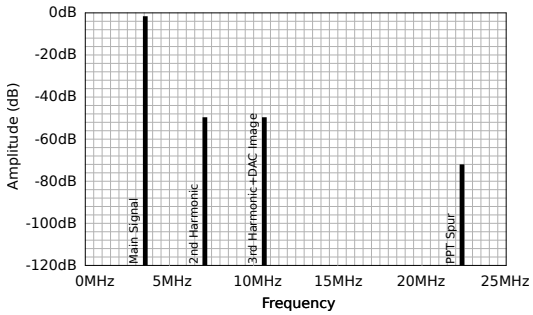


Abbildung 3.5.: DDS-Spektrum 80m

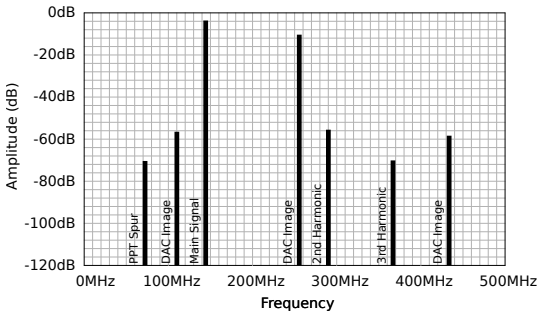


Abbildung 3.6.: DDS-Spektrum 2m

Die Simulation des 80m-Teils zeigt, dass nur Spektralanteile höher als das gewünschte Signal im Ausgangssignal enthalten sind. Auch deren Mischprodukte (an nichtlinearen Kennlinien) fallen nicht unter die Grundschwingung. Dadurch sind im 80m-Teil die diversen Nebenprodukte unkritisch, da diese spätestens vom Oberwellenfilter unterdrückt werden.

Problematischer ist hier der 2m-Teil. Verschiedene Nebenprodukte liegen unter der gewünschten Ausgangsfrequenz, zusätzlich können Mischprodukte auch unter diese fallen (z.B. „3rd Harmonic “ mit dem großen „DAC Image“). Diese werden vom Oberwellenfilter nicht unterdrückt, da dieses Tiefpasscharakteristik besitzt und somit nur über der Grundschwingung liegende Anteile unterdrücken kann. Aus diesem Grund wird vor dem 2m-Leistungsverstärker ein Eingangsfilter notwendig, siehe Abschnitt 3.5.3 auf Seite 67.

3.3. Bandumschaltung 80m/2m

3.3.1. Anforderungen

Da für das 80m- und das 2m- Band zwei getrennte Leistungsverstärker vorhanden sind, eine Fuchsjagd jedoch nicht in zwei Bändern gleichzeitig stattfindet, muss zwischen den beiden Leistungsverstärkern umgeschaltet werden. Dies beinhaltet erstens das Trennen der Versorgung des nicht benötigten Verstärkers, sowie das Durchschalten des DDS-Ausgangssignales auf den aktiven Verstärker.

Das schaltende Element in der Versorgungsspannung sollte einen geringen Spannungsabfall verursachen, um daran bei Betriebsstrom möglichst wenig Leistung in Wärme umzuwandeln. Analog dazu sollte auch die Signalumschaltung auf beiden Frequenzbändern eine möglichst geringe Dämpfung verursachen.

3.3.2. Konzept und Bauteilauswahl

Um eine positive Versorgungsspannung mit sehr geringem Spannungsabfall elektronisch (nicht mechanisch per Relais) schalten zu können, bietet sich ein P-Kanal MOSFET an. Im Gegensatz zum N-Kanal MOSFET benötigt dieser keine Schaltspannung über der Betriebsspannung, was die Ansteuerung wesentlich vereinfacht. Die Auswahl fällt auf den FDT434P der Firma Fairchild. Dieser besitzt folgende, hier relevante Eingeschaften (vgl. *P-Channel PowerTrench MOSFET Datasheet* 2011, Seite 1-3):

Übergangswiderstand (eingeschaltet):	$R_{\mathrm{DS(ON)}} < 50\,\mathrm{m\Omega}$
Maximaler Drainstrom:	$I_{\mathrm{D(MAX)}} = 5.5\,\mathrm{A}$
Gatespannung zum Durchschalten:	$U_{\mathrm{GS(ON)}} \approx 2.5\,\mathrm{V}$

Für die beiden Leistungsverstärker kann ein Stromverbrauch in der Größenordnung von 1 A angenommen werden (siehe auch 3.4.4 auf Seite 53) , was einen vernachlässigbaren Spannungsabfall von $U_{\mathrm{DS}} = R_{\mathrm{DS(ON)}} \cdot 1\,\mathrm{A} = 50\,\mathrm{mV}$ bewirkt.

Zur Umschaltung des DDS-Signals werden PIN-Dioden eingesetzt. Dem zugrunde liegt folgende Idee: Fließt ein Gleichstrom in Durchlassrichtung durch eine Diode, so besitzt diese zwischen beiden Anschlüssen einen vergleichsweise geringen differentiellen Widerstand (Tangente an die „steile“ Diodenkennlinie). Befindet sich die Diode hingegen in Sperrrichtung, ist ihr differentieller Widerstand praktisch unendlich (abgesehen vom Sperrstrom). Allerdings ist bei hohen Frequenzen (bes. im 2m-Band) die Sperrschichtkapazität unter Umständen nicht zu vernachlässigen. Aus diesem Grunde werden hier häufig PIN-Dioden eingesetzt, welche zur Senkung der Sperrschichtkapazität zwischen P- und N-Schicht eine zusätzliche I-Schicht besitzen.

Hier wird die BAP64-06 Doppel-PIN-Diode ausgewählt, welche eine Sperrkapazität in der Größenordnung von $0.2\,\mathrm{pF} \ldots 0.5\,\mathrm{pF}$ aufweist. Der differentielle Widerstand in Durchlassrichtung beträgt bei einem Strom von $I_{\mathrm{D}} = 5\,\mathrm{mA}$ ungefähr $R_{\mathrm{d}} \approx 2\,\Omega$ (vgl. *Silicon PIN diode Datasheet* 2015, Seite 2,3).

3.3.3. Implementierung

Die schaltungstechnische Realisierung der Bandumschaltung ist in Abbildung 3.7 auf der nächsten Seite dargestellt.

Da jeweils die Source-Anschlüsse der P-Kanal MOSFETs mit der 12 V-Versorgungsschiene verbunden sind, bewirkt eine entsprechende Gatespannung unter ca. 9 V ein Durchschalten der Transistoren. $R207$ und $R208$ „ziehen“ die Gates im abgeschalteten Fall der Bipolartransistoren auf Sourcepotential, sodass die FETs sicher sperren. Schaltet beispielsweise der Bipolartransistor T203 durch, so ergibt sich über den Spannungsteiler $R205/R207$ eine Gate-Spannung von

$$U_{\mathrm{GS}} = \frac{R207}{R205 + R207} \cdot (-12\,\mathrm{V}) \approx -3.8\,\mathrm{V}\ \text{(bezogen auf 12 V)} \tag{3.3}$$

Dies schaltet T203 sicher durch. U_{CE} der Bipolartransistoren wird hier vernachlässigt.

Die gleichen Bipolartransistoren T201 und T202 schalten auch den Strom durch die PIN-Dioden. Als Kompromiss zwischen möglichst geringem differentiellen Widerstand und Stromverbrauch wird ein Diodenstrom von $I_{\mathrm{D}} = 5\,\mathrm{mA}$ festgesetzt (vgl. *Silicon PIN diode Datasheet* 2015, Seite 3, Fig. 3). Auf der Leitung `DDS_SIG`

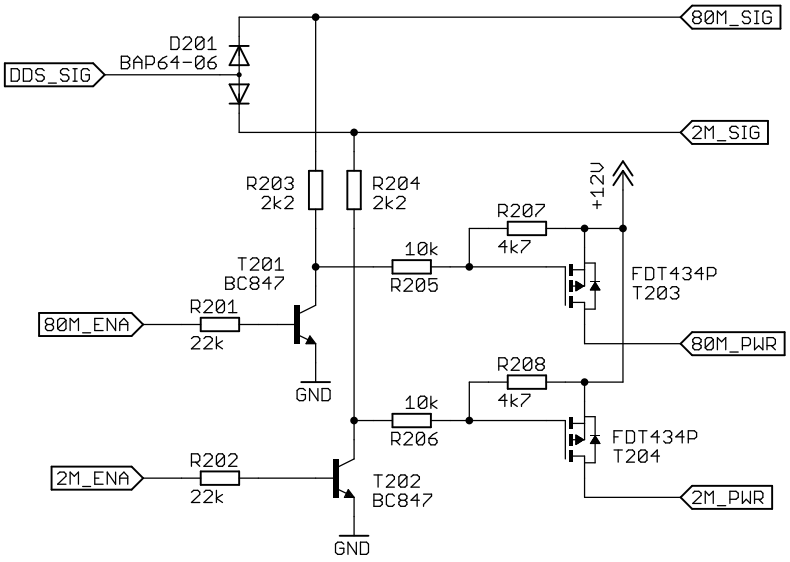


Abbildung 3.7.: Schaltplan Bandumschaltung

befindet sich aufgrund des Übertragers bereits ein 12 V-Offset (siehe 3.2.4 auf Seite 41), somit ergibt sich abzüglich der Flussspannung $U_\mathrm{F} \approx 0.95\,\mathrm{V}$ der PIN-Diode (vgl. *Silicon PIN diode Datasheet* 2015, Seite 2) unter Vernachlässigung von U_CE der Bipolartransistoren:

$$R203 = R204 = \frac{12\,\mathrm{V} - U_\mathrm{F}}{5\,\mathrm{mA}} = 2.21\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \tag{3.4}$$

3.4. 80m-Leistungsverstärker

3.4.1. Anforderungen

Aufgabe des 80m-Leistungsverstärkers ist es, das Ausgangssignal des DDS im Frequenzbereich des 80m-Bandes so zu verstärken, dass es mit gewünschter Leistung über eine Antenne abgestrahlt werden kann. Zusätzlich muss das Ausgangssignal im Bezug auf spektrale Reinheit (unerwünschte Nebenaussendungen durch Verzerrungen etc.) den Anforderungen des Amateurfunkgesetzes genügen. Darin enthalten sind die Anforderungen an Amateurfunkstellen im Frequenzbereich von 1.7 MHz bis 35 MHz im Bezug auf unerwünschte Aussendungen (vgl. *Änderungen der Vorordnung zur Durchführung des Amateurfunkgesetzes* 2008, Seite 2).

Absenkung gegen Grundwelle:	40 dBc
Maximale Leistung:	−36 dBm

Hierbei gilt immer der weniger strenge Wert.

Weitere Anforderungen an den Sender enthalten die technischen Spezifikationen im Anhang 1 der ARDF-Rules der IARU (vgl. *Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1* 2015, Seite 10). Für den 80m-Leistungsverstärker sind hier von Bedeutung:

Trägerfrequenz:	3510 kHz ... 3600 kHz
HF-Ausgangsleistung:	1 W ... 5 W

Für die folgenden Berechnungen wird ein representativer Wert von $f = 3.55\,\mathrm{MHz}$ verwendet, welcher ungefähr in der Mitte des geforderten Frequenzbereiches liegt.

Laut Datenblatt des DDS beträgt der Full-Scale Output Current der differentiellen DAC-Ausgänge, welche als Stromsenke (Open-Drain) ausgeführt sind (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 3):

$$I_{\mathrm{FSC}} = 5\,\mathrm{mA(min)} \ldots 10\,\mathrm{mA(typ)} \tag{3.5}$$

Dies bedeutet bei Standardbeschaltung, wie sie auf dem Evaluationsboard des Herstellers vorliegt (vgl. *AD9859 Evaluation Board Schematics* 2004, Seite 2), eine Ausgangsleistung in $50\,\Omega$ von:

$$\begin{aligned} P_{\mathrm{DDS}} &= \left(\frac{I_{\mathrm{FSC}}}{2 \cdot \sqrt{2}}\right)^2 \cdot 50\,\Omega \approx 156\,\mu\mathrm{W(min)} \ldots 625\,\mu\mathrm{W(typ)} \\ &\approx -8\,\mathrm{dBm(min)} \ldots -2\,\mathrm{dBm(typ)} \end{aligned} \tag{3.6}$$

Da ein Verringern der Ausgangsleistung einfacher möglich ist als ein nachträgliches Erhöhen die laut IARU Rules erlaubte Leistung

$$P_{\mathrm{OUT}} = 5\,\mathrm{W} = 37\,\mathrm{dBm} \tag{3.7}$$

angestrebt. Somit steht fest, welchen Gesamtverstärkungsfaktor der 80m-Leistungsverstärker für den typischen Fall besitzen muss:

$$G_{\mathrm{TOT}} = \frac{P_{\mathrm{OUT}}}{P_{\mathrm{DDS}}} = 37\,\mathrm{dBm} - (-2\,\mathrm{dBm}) = 39\,\mathrm{dB} \tag{3.8}$$

Die Ausgangsimpedanz des 80m-Leistungsverstärkers soll, typisch für HF-Schaltungen

$$Z_{\mathrm{OUT}} = 50\,\Omega \tag{3.9}$$

betragen.

Weiters soll die Spannungsversorgung des gesamten Verstärkers schaltbar sein, sodass, wenn nicht benötigt, dieser zur Reduzierung des Stromverbrauches von der Versorgung getrennt werden kann. Dies wird realisiert mit der Bandumschaltung, siehe 3.3 auf Seite 44.

3.4.2. Konzept und Bauteilauswahl

Grundsätzlich soll der 80m-Leistungsverstärker diskret, d.h. aus Transistoren und passiven Bauelementen aufgebaut werden, wie es bei Verstärkern in diesem Leistungs- und Frequenzbereich häufig Praxis ist. Aus obiger Berechnung der notwendigen Gesamtverstärkung wird ersichtlich, dass diese bei geforderter Ausgangsleistung nicht mit einer einzelnen Transistorstufe erzielt werden kann. Typische Verstärkungsfaktoren von vergleichbaren Endstufentransistoren der benötigten Leistungsklasse liegen bei 10 dB für Bipolartransistoren (vgl. *2SC1971 NPN RF Power Transistor Datasheet* 1997, Seite 1) sowie 16 dB für Feldeffekttransistoren (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 1).

Aufgrund der höheren Verstärkung bei vergleichbarem Preis fällt die Wahl auf einen passenden Feldeffekttransistor (RD06HHF1).

Der restliche Verstärkungsfaktor (ca. 23 dB) muss mithilfe einer Vorverstärkerstufe erzielt werden. Diese wird als Resonanzverstärker ausgeführt, um gleichzeitig unerwünschte Nebenprodukte des DDS zu dämpfen, zusätzlich zum Oberwellenfilter am Ausgang. Da die zu verarbeitenden Frequenzen vergleichsweise niedrig sind (3.55 MHz), wird in der Vorverstärkerstufe der Standardtransistor BC847 eingesetzt, welcher gut verfügbar ist und hier leicht ausreicht.

Abbildung 3.8 zeigt das Blockschaltbild des 80m-Leistungsverstärkers.

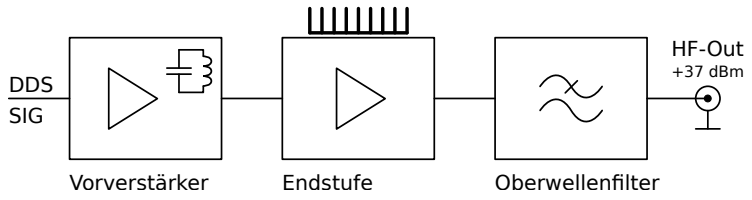


Abbildung 3.8.: Blockschaltbild 80m-Leistungsverstärker

3.4.3. Vorverstärker

Grundsätzlich soll der Vorverstärker als ein Resonanzverstärker in Emittergrundschaltung im A-Betrieb realisiert werden. Dabei wird die Arbeitspunktstabilisierung klassisch mithilfe eines Basisspannungsteilers in Kombination mit einem Emitterwiderstand ($R217$) vorgenommen. Damit der Emitterwiderstand für Nutzsignale keine Gegenkopplung darstellt und dadurch die Verstärkung verringert, wird er wechselspannungsmäßig durch $C234$ kurzgeschlossen.

Der Schaltplan des Vorverstärkers ist in Abbildung 3.9 dargestellt. Mithilfe der Zenerdiode $D202$ wird eine von der Betriebsspannung unabhängige Referenzspannung erzeugt, welche zur Einstellung des Arbeitspunkts des Vorverstärkers und der Endstufe (Abschnitt 3.4.4 auf Seite 53) benutzt wird. Dadurch wird erreicht, dass eine Änderung der Betriebsspannung (z.B. durch Entladung des Akkus) nicht zu einer Verschiebung der Arbeitspunkte führt. Für $D202$ wird 5.6 V ausgewählt, da dies größer ist als die beiden benötigten Arbeitspunktspannungen, aber immer noch wesentlich kleiner als die Betriebsspannung. Darüber hinaus sind Zenerdioden dieser Spannung sehr gängig.

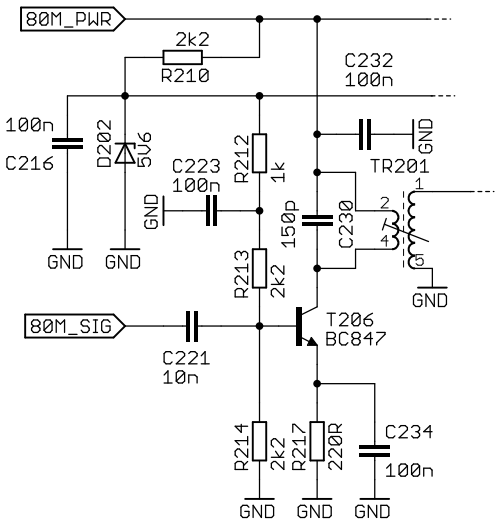


Abbildung 3.9.: Schaltplan 80m-Vorverstärker

Für die beiden Spannungsteiler, welche dann die endgültigen Arbeitspunktspannungen für Vorverstärker und Endstufe erzeugen, wird der Einfachheit halber ein Gesamtwiderstand von ca. 5.6 kΩ festgelegt. Daraus ergibt sich der Belastungs-

strom der „Konstantspannungsquelle“ (unter Vernachlässignung des Basisstromes von $T206$)

$$I_{\mathrm{LAST}} = 2 \cdot \frac{5.6\,\mathrm{V}}{5.6\,\mathrm{k\Omega}} = 2\,\mathrm{mA} \tag{3.10}$$

Nimmt man als Strom durch die Zenerdiode einen Wert von $I_{\mathrm{D202}} = 1\,\mathrm{mA}$ an, so ergibt sich für $R210$

$$R210 = \frac{U_{\mathrm{B}} - U_{\mathrm{D202}}}{I_{\mathrm{LAST}} + I_{\mathrm{D202}}} = \frac{12\,\mathrm{V} - 5.6\,\mathrm{V}}{2\,\mathrm{mA} + 1\,\mathrm{mA}} \approx 2.13\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \tag{3.11}$$

$C216$ bildet mit $R210$ zusätzlich einen Tiefpass ($f_{\mathrm{g}} \approx 720\,\mathrm{Hz}$) und verhindert, dass hochfrequente Störungen über die Betriebsspannung an die Basis von $T206$ gelangen und von diesem verstärkt werden. Der Eingang der Vorverstärkerstufe ist besonders kritisch in dieser Hinsicht, da hier vorhandene Störungen mit der (hohen) Gesamtverstärkung beider Stufen verstärkt werden.

Die Spannung der Zenerdiode $D202$ wird nun mithilfe der Widerstände $R212$, $R213$ und $R214$ geteilt und zur Arbeitspunkteinstellung von $T206$ verwendet. $R212$ bildet mit $C223$ nochmals einen Tiefpass ($f_{\mathrm{g}} \approx 1.6\,\mathrm{kHz}$), aus demselben Grund wie in vorigem Absatz.

Eine Faustformel zur Arbeitspunkteinstellung von Emitterschaltungen besagt, dass am Emitterwiderstand (in diesem Fall $R217$) mehr als ein Zehntel der Betriebsspannung der Schaltung anliegen sollte.

$$U_{\mathrm{R217}} = \frac{U_{\mathrm{B}}}{10} = 1.2\,\mathrm{V} \tag{3.12}$$

Die Basis-Emitter-Spannung von $T206$ beträgt (vgl. *BC847 NPN general-purpose transistor Datasheet* 2014, Seite 5) $U_{\mathrm{BE}} \approx 0.66\,\mathrm{V}$. Daraus folgt die Spannung U_{R214} und in weiterer Folge, unter der Annahme, dass der Gesamtwiderstand des Spannungsteilers $R_{\mathrm{ges}} \approx 5.6\,\mathrm{k\Omega}$ betragen sollte, auch $R214$:

$$R214 = (U_{\mathrm{R217}} + U_{\mathrm{BE}}) \cdot \frac{R_{\mathrm{ges}}}{U_{\mathrm{D202}}} = 1.9\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \tag{3.13}$$

Aufgerundet wird aus dem Grund, dass eine Spannung am Emitterwiderstand größer als $U_{\mathrm{B}}/10$ günstiger ist als eine kleinere, nach der erwähnten Faustformel.

Somit verbleiben für $R212$ und $R213$ noch $R_{\mathrm{ges}} - R_{\mathrm{R214}} = 3.4\,\mathrm{k\Omega}$, welche auf die beiden Widerstände aufgeteilt werden:

$$R212 = 2.2\,\mathrm{k\Omega} \qquad R213 = 1\,\mathrm{k\Omega} \tag{3.14}$$

Mithilfe von $R217$ kann nun der Arbeitspunktstrom festgelegt werden, welcher die Steilheit des Transistors im Arbeitspunkt und somit die Verstärkung der Stufe festlegt. Versuche ergaben, dass ein Arbeitspunktstrom von $I_\mathrm{C} \approx 7\,\mathrm{mA}$ zu einer ausreichenden Verstärkung führt. Dadurch ergibt sich für den Emitterwiderstand

$$R217 = \frac{U_\mathrm{R217}}{I_\mathrm{C}} = \frac{1.62\,\mathrm{V}}{7\,\mathrm{mA}} \approx 231\,\Omega \approx 220\,\Omega \tag{3.15}$$

$C234$ wird bestückt mit 100 nF, weil dessen Blindwiderstand bei Betriebsfrequenz dann

$$X_\mathrm{C234} = \frac{1}{\omega \cdot C234} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz} \cdot 100\,\mathrm{nF}} \approx 0.44\,\Omega \tag{3.16}$$

beträgt, was wesentlich kleiner ist als die Impedanz des Emitters (im Arbeitspunkt). Diese berechnet sich nach dem Kleinsignalmodell unter der Annahme, dass $U_\mathrm{T} = 26\,\mathrm{mV}$:

$$r_\mathrm{E} = \frac{U_\mathrm{T}}{I_\mathrm{C}} = \frac{26\,\mathrm{mV}}{7\,\mathrm{mA}} \approx 3.7\,\Omega \tag{3.17}$$

Somit kann der Emitter für Signale der Betriebsfrequenz praktisch als mit Masse verbunden angesehen werden, wodurch für diese Signale keine Gegenkopplung besteht und demnach die Verstärkung maximal ist.

Das Ausgangssignal des DDS wird über den Koppelkondensator $C221$ an die Basis von $T206$ geführt. Die Eingangsimpedanz des Transistorverstärkers beträgt ca.

$$Z_\mathrm{IN} = R212||R213||r_\mathrm{BE} = \frac{1}{\frac{1}{R212} + \frac{1}{R213} + \frac{I_\mathrm{C}}{U_\mathrm{T} \cdot \beta}} \approx 533\,\Omega \tag{3.18}$$

Hierbei wird die Temperaturspannung $U_\mathrm{T} = 26\,\mathrm{mV}$ und die Stromverstärkung $\beta = 280$ angenommen (vgl. B-Typ *BC847 NPN general-purpose transistor Datasheet* 2014, Seite 5). Wird $C221$ mit 10 nF bestückt, so ist sein Blindwiderstand bei Betriebsfrequenz, wie bei Koppelkondensatoren gewünscht, wesentlich kleiner als die Eingangsimpedanz des nachfolgenden Verstärkers.

$$X_\mathrm{C221} = \frac{1}{\omega \cdot C221} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz} \cdot 10\,\mathrm{nF}} \approx 4.4\,\Omega \tag{3.19}$$

Die Kopplung zwischen Vorverstärker und Endstufe erfolgt durch den abstimmbaren Übertrager TR201, welcher gleichzeitig mit $C230$ den Kollektorschwingkreis bildet. Mithilfe des Übertragungsverhältnisses kann der Vorverstärker optimal an die nachfolgende Stufe, den Leistungs-MOSFET, angepasst werden.

Für TR201 soll der abgeschirmte Filterbausatz „7F1S“ benutzt werden. Dessen Kern besitzt einen magnetischen Leitwert von

$$A_\mathrm{L} = 12\,\mathrm{nH} \tag{3.20}$$

Nun muss einerseits die optimale Güte, bestimmt durch Verhältnis von L und C bei gegebener Belastung, sowie das Übertragungsverhältnis von TR201 bestimmt werden. Rechnerisch ist dies nur möglich, wenn Daten über die Eingangsimpedanz des Leistungs-MOSFETs vorliegen. Im Datenblatt sind jedoch keine S-Parameter des FETs zur Betriebsfrequenz angegeben (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 7).

Auch eine Messung (z.B. mittels eines Netzwerkanalysators) ist nicht ohne weiteres möglich, da das Gate des FET im „realen“ Betrieb über einen großen Bereich ($\approx \pm 12\,\mathrm{V}$) ausgesteuert werden muss, um die gewünschte Ausgangsleistung zu erreichen. Bei einem solchen Großsignalbetrieb ist jedoch die Eingangskennlinie stark nichtlinear, sodass die Angabe eines eindeutigen Eingangswiderstandes schwierig ist. Ein Netzwerkanalysator ist im Allgemeinen nur für die Messung von Kleinsignalparametern geeignet, da er nur minimal um den Arbeitspunkt aussteuert.

Stattdessen wird in einigen Versuchen ermittelt, welche Konstellation aus Güte (in weiterer Folge Primärinduktivität des Übertragers) und Übertragungsverhältnis die gewünschte Bandbreite bei entsprechender Ausgangsleistung bewirkt:

$$\ddot{U} = N_\mathrm{SEK} : N_\mathrm{PRI} = 2 \qquad L_\mathrm{PRI} = 8\,\mu\mathrm{H} \tag{3.21}$$

Für den gegebenen Kern bedeutet dies:

$$\begin{aligned} N_\mathrm{PRI} &= \sqrt{\frac{L_\mathrm{PRI}}{A_\mathrm{L}}} = 25.8\,\mathrm{Wdg.} \approx 26\,\mathrm{Wdg.} \\ N_\mathrm{SEK} &= N_\mathrm{PRI} \cdot \ddot{U} = 52\,\mathrm{Wdg.} \end{aligned} \tag{3.22}$$

Nun gilt es, den Schwingkreis auf die gewünschte Resonanzfrequenz abzugleichen. Bei bekannter Induktivität und Betriebsfrequenz ergibt sich daraus die Schwingkreiskapazität C_SK

$$\omega_\mathrm{res} = \frac{1}{\sqrt{L_\mathrm{PRI} \cdot C_\mathrm{SK}}} \quad \rightarrow \quad C_\mathrm{SK} = \frac{1}{4\pi^2 f_\mathrm{res}^2 L_\mathrm{PRI}} = 251.24\,\mathrm{pF} \tag{3.23}$$

Dieser Wert darf jedoch noch nicht als $C230$ eingesetzt werden, da zusätzlich die Gatekapazität des Leistungs-MOSFETs entsprechend dem Übertragungsverhält-

nis von TR201 in den Schwingkreis transformiert wird, d.h. dort als C_{TR} aufscheint und sich zu $C230$ dazuaddiert. Die Gatekapazität beträgt $C_{\mathrm{GS}} \approx 25\,\mathrm{pF}$ (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 3). Dies ergibt im Kreis

$$C_{\mathrm{TR}} = \ddot{U}^2 \cdot C_{\mathrm{GS}} \approx 100\,\mathrm{pF} \tag{3.24}$$

Daraus folgt

$$C230 = C_{\mathrm{SK}} - C_{\mathrm{TR}} = 151.24\,\mathrm{pF} \approx 150\,\mathrm{pF} \tag{3.25}$$

3.4.4. Endstufe

Als Endstufentransistor wird, wie bereits in Abschnitt 3.4.2 auf Seite 48 erwähnt, der HF-Power-MOSFET RD06HHF1 von Mitsubishi ausgewählt. Die mögliche HF-Ausgangsleistung von 6 W (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 1) genügt für die Anforderungen nach Abschnitt 3.4.1 auf Seite 46, bezogen auf die Frequenz ist der Transistor leicht ausreichend (bis 30 MHz).

Der FET wird im Wesentlichen so beschaltet (siehe Abbildung 3.10), wie die Testschaltung im Datenblatt es vorschlägt (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 5).

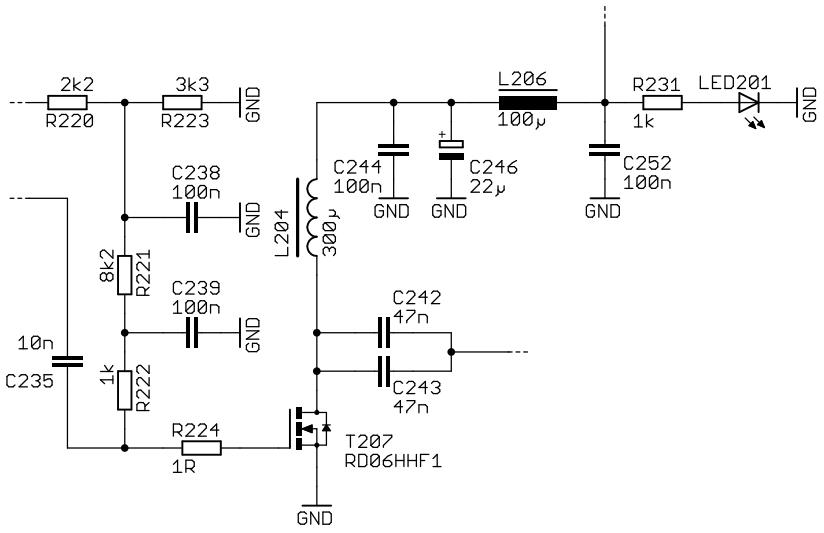


Abbildung 3.10.: Schaltplan 80m Endstufe

Mithilfe des Spannungsteilers $R220$ und $R223$ wird eine Gate-Vorspannung eingestellt. Diese wird mit $U_{\mathrm{GS}} \approx 3.5\,\mathrm{V}$ so gewählt, dass gerade noch kein Drainstrom

fließt, d.h. die Stufe im B-Betrieb arbeitet (vgl. Diagramm „Vgs-Ids Characteristics“ *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 3). Der B-Betrieb bedeutet gegenüber dem A-Betrieb neben einem erhöhten Wirkungsgrad auch größere Verzerrungen (Oberwellen) aufgrund der Nichtlinearität der Übertragungskennlinie um den Nullpunkt. Für einen batteriebetriebenen Sender überwiegt jedoch der Vorteil des höheren Wirkungsgrades gegenüber dem erhöhten Aufwand beim nachfolgenden Tiefpassfilter zur Unterdrückung der Oberwellen.

$R220$ und $R223$ teilen die Spannung der 5.6 V-Zenerdiode herab auf die gewünschte U_{GS}. Dieser Spannungsteiler sollte entsprechend der Berechnung des Zenerdiodenstromes in Abschnitt 3.4.3 auf Seite 49 einen Gesamtwiderstand von ca. 5.6 kΩ aufweisen, woraus sich unmittelbar die beiden Widerstände ergeben:

$$R223 = 3.5\,\mathrm{k\Omega} \approx 3.3\,\mathrm{k\Omega} \qquad R220 = 2.1\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \tag{3.26}$$

Zur Kontrolle wird nochmals U_{GS} berechnet:

$$U_{\mathrm{GS}} = 5.6\,\mathrm{V} \cdot \frac{R223}{R220 + R223} = 3.36\,\mathrm{V} \tag{3.27}$$

Die Abweichung von ca. 0.14 V ist hier keinesfalls problematisch, es geht lediglich darum, dass der FET nicht zu leiten beginnt, was bei einer kleineren U_{GS} nicht passieren wird.

Die Gatevorspannung wird über einen zweistufigen RC-Tiefpass auf das Gate des FET geleitet, welcher verhindert, dass Störungen auf diesem Wege an das Gate gelangen und mitverstärkt werden bzw. den Verstärker zum Schwingen bringen könnten.

$R224$ soll ebenfalls die Schwingneigung der Schaltung reduzieren und wird aus der Testschaltung im Datenblatt übernommen.

$C235$ überlagert als Eingangs-Koppelkondensator das Ausgangssignal des Vorverstärkers über die Gate-Vorspannung. Dieser wird mit $C235 = 10\,\mathrm{nF}$ bestückt, da bei der Betriebsfrequenz dessen Blindwiderstand

$$X_{C235} = \frac{1}{\omega \cdot C235} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz} \cdot 10\,\mathrm{nF}} \approx 4.4\,\Omega \tag{3.28}$$

wesentlich kleiner ist als Z_{IN} des FETs (experimentell festgestellt, da Z_{IN} nicht genauer bekannt ist).

Die Draindrossel $L204$ muss bei Betriebsfrequenz einen hohen Blindwiderstand aufweisen, um das Ausgangssignal nicht zu dämpfen, sowie den Maximalstrom aushalten. Hierfür wird der Ringkern „RIK 10“ ausgewählt, welcher mit $N =$

15 Wdg. Kupferlackdraht (ca. 0.4 mm^2) bewickelt wird. Im Datenblatt (vgl. *RIK 10 Ferroxcube Core Datasheet* 2013, Seite 1) des Ringkerns wird für den magnetischen Leitwert angegeben

$$A_\mathrm{L} = 1320\,\mathrm{nH} \ldots 1760\,\mathrm{nH} \tag{3.29}$$

Eine Messung ergab

$$A_\mathrm{L} \approx 1410\,\mathrm{nH} \tag{3.30}$$

Daraus ergibt sich mit $N = 15\,\mathrm{Wdg.}$

$$L204 = A_\mathrm{L} \cdot N^2 \approx 300\,\mu\mathrm{H} \tag{3.31}$$

Somit ist der Blindwiderstand bei Betriebsfrequenz

$$X_{L204} = \omega \cdot L204 = 2\pi \cdot 3.55\,\mathrm{MHz} \cdot 300\,\mu\mathrm{H} \approx 6.7\,\mathrm{k\Omega} \tag{3.32}$$

Dies ist wesentlich größer als die Eingangsimpedanz des nachfolgenden Tiefpassfilters, welche später auf 8 Ω festgelegt werden wird (siehe Abschnitt Oberwellenfilter 3.4.5 auf der nächsten Seite).

Der Ausgangskoppelkondensator, bestehend aus $C242$ und $C243$, wird zweigeteilt ausgeführt, da an diesem Punkt der Schaltung aufgrund der niedrigen Z_IN des Tiefpassfilters verhältnismäßig große Ströme fließen und sich ein einzelner Koppelkondensator bei Versuchen bereits erwärmte. Zusammen besitzen $C242$ und $C243$ einen Blindwiderstand von

$$X_\mathrm{C} = \frac{1}{\omega \cdot (C242||C243)} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz} \cdot 94\,\mathrm{nF}} \approx 0.48\,\Omega \tag{3.33}$$

was wesentlich kleiner ist als Z_IN des Tiefpassfilters, sodass die Ausgangsleistung nicht maßgeblich gesenkt wird.

Um zu verhindern, dass durch die hohen Ströme, die in der Endstufe mitunter fließen, Störungen auf die Betriebsspannung gelangen, wird diese mithilfe eines LC-Tiefpassfilters , bestehend aus $L206$, $C244$ und $C246$, abgeblockt. Die Dimensionierung des Tiefpasses richtet sich eher nach verfügbaren Bauteilen, die für die Anforderungen geeignet sind.

Der Kondensator des Filters wird zweiteilig ausgeführt, um eine niedrige Impedanz auch bei höheren Frequenzen (Oberwellen) sicherzustellen. So wird für $C246$ ein Low-ESR-Tantalkondensator ausgewält, welcher zumindest auf der Grundfrequenz (3.55 MHz) noch eine niedrige Impedanz aufweist (vgl. *T491 Series Tantalum SMD Capacitors Datasheet* 2014, Seite 4). Da die Impedanz des Tantalkondensators ab ca. 5 MHz wieder ansteigt, „übernimmt“ ab dort sozusagen $C244$.

Bei der Spule $L206$ ist zu beachten, dass die Grundfrequenz unterhalb der Eigenresonanzfrequenz der Spule liegen muss, da oberhalb nur mehr kapazitives Verhalten vorliegt, und somit die Tiefpasswirkung schwindet. Zusätzlich darf der Betriebs(gleich)strom die Spule nicht in Sättigung bringen.

Der Betriebsstrom kann in etwa über die gewünschte Ausgangsleistung und den Wirkungsgrad abgeschätzt werden. Dieser beträgt für die vorliegende Betriebsart des FET typischerweise $\eta \approx 50 \ldots 70\%$. Bei einer gewünschten Ausgangsleistung ergibt sich für den Stromverbrauch

$$I_\mathrm{V} = \frac{P_\mathrm{OUT} \frac{100\%}{\eta}}{U_\mathrm{B}} \approx 0.6\,\mathrm{A} \ldots 0.83\,\mathrm{A} \tag{3.34}$$

Für $L206$ wird eine Drossel der Firma Würth-Elektronik ausgewählt, die alle diese Anforderungen erfüllt. Diese besitzt bei eine Eigenresonanzfrequenz von 6 MHz sowie einer Strombelastbarkeit von 1.2 A eine Induktivität von $L206 = 100\,\mu\mathrm{H}$ (vgl. *Würth Power Choke Datasheet* 2008, Seite 1).

3.4.5. Oberwellenfilter

Grundsätzlich soll das Ausgangsfilter als LC-Tiefpass in π - Schaltung, auch genannt Collinsfilter, realisiert werden, wie es bei Oberwellenfiltern häufig Praxis ist. Die π - Schaltung bietet neben der Tiefpasswirkung auch die Möglichkeit, eine Impedanzanpassung zwischen Ein- und Ausgang vorzunehmen. Dies ist notwendig, da die Betriebsspannung von $U_\mathrm{B} = 12V$ bei einer Ausgangsimpedanz von $50\,\Omega$ die Ausgangsleistung bei

$$P_\mathrm{MAX} = \frac{U^2}{R} = \frac{\left(\frac{U_\mathrm{B}}{\sqrt{2}}\right)^2}{50\,\Omega} \approx 1.4\,\mathrm{W} \tag{3.35}$$

begrenzt, allerdings $P_\mathrm{OUT} = 5\,\mathrm{W}$ benötigt werden. Bei obiger Berechnung gilt die Annahme, dass die Ausgangsspannung aufgrund eines (einfachen) Ausgangsschwingkreises auf die doppelte Betriebsspannung ansteigen, aufgrund des FETs jedoch nicht unter Masse absinken kann. Dadurch ergibt sich eine Spitze-Spitze-Spannung, die doppelt so hoch ist wie die Betriebsspannung.

Mithilfe der Impedanzanpassung kann der Endstufentransistor genügend Leistung an die entsprechend geringe Eingangsimpedanz des Filters abgeben, ohne dass dabei die Spannung über die Betriebsspannungsgrenzen ansteigt.

Die Ordnung des Filters muss so hoch sein, dass die Anforderungen an die spektrale Reinheit (siehe Abschnitt 3.4.1 auf Seite 46) erfüllt werden. Um hier eine Rechnung durchführen zu können, müsste zuerst bekannt sein, welchen Signalpegel die Oberwellen besitzen (insbesondere die zweite Oberwelle, die durch Verzerrung an der quadratischen FET-Kennlinie entsteht). Dies analytisch zu berechnen ist aufgrund der nichtlinearen FET-Kennlinie mit erheblichem Aufwand verbunden.

Grundsätzlich besitzt ein π - Filter dritte Ordnung, was dadurch veranschaulicht werden kann, dass sich in ihm drei frequenzabhängige Bauteile (C-L-C) befinden. Dies bedeutet einen Abfall von $-18\,\mathrm{dB}$/Oktave. Die zweite Oberwelle wird somit ungefähr (abhängig von der Güte) um diese $-18\,\mathrm{dB}$ gedämpft. In einem Versuch wird ermittelt, dass dies *nicht* ausreicht.

Ein zweistufiges π-Filter besitzt die doppelte Ordnung, dämpft also die zweite Oberwelle mit ca. $-36\,\mathrm{dB}$. Am Aufbau wird experimentell festgestellt, dass dies ausreichend ist, um die Anforderungen zu erfüllen.

Zur Berechnung eines einstufigen π-Filters existieren verschiedene Ansätze bzw. Vorgehensweisen. Im Folgenden wird das Filter als zwei hintereinandergeschaltete Resonanztransformationsglieder betrachtet, welche wiederum separat dimensioniert werden.

Resonanztransformation

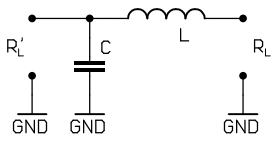


Abbildung 3.11.: Einfaches Resonanztransformationsglied

In Abbildung 3.11 bezeichnet R_L den (realen) Lastwiderstand, während R'_L für den auf den Eingang transformierten Lastwiderstand steht. Das dargestellte Beispiel transformiert einen kleinen R_L in einen größeren, scheinbaren R'_L. Baut man das Glied spiegelverkehrt auf, so transformiert es in die umgekehrte Richtung, d.h. von groß nach klein.

Bei der Resonanztransformation gelten, bezogen auf die Schaltung in Abbildung 3.11 auf der vorherigen Seite grundsätzlich folgende Annahmen:

$$R_\mathrm{L} \cdot R'_\mathrm{L} = Z_0^2 \qquad Z_0 = \sqrt{\frac{L}{C}} \tag{3.36}$$

Durch Einsetzen der Gleichungen 3.36 ineinander ergibt sich:

$$\begin{aligned} R_\mathrm{L} \cdot R'_\mathrm{L} = \frac{L}{C} \quad &\rightarrow \quad L = R_\mathrm{L} \cdot R'_\mathrm{L} \cdot C \\ &\rightarrow \quad C = \frac{L}{R_\mathrm{L} \cdot R'_\mathrm{L}} \end{aligned} \tag{3.37}$$

Die Resonanzfrequenz und somit der Punkt, an dem die Eingangsimpedanz der Schaltung rein reell ist, weicht aufgrund der Belastung (Dämpfung) im Schwingkreis von der idealen Resonanzfrequenz ab. Um diese sogenannte Resonanzverschiebung miteinzuberechnen, wird die Eingangsimpedanz R'_L der Schaltung 3.11 auf der vorherigen Seite angesetzt (nach komplexer Wechselstromrechnung):

$$R'_\mathrm{L}(j\omega) = \frac{\frac{1}{j\omega C} \cdot (j\omega L + R_\mathrm{L})}{\frac{1}{j\omega C} + j\omega L + R_\mathrm{L}} \tag{3.38}$$

Setzt man nun die Eingangsimpedanz als rein reell an (Resonanzbedingung)

$$\mathrm{Im}\{R'_\mathrm{L}(j\omega)\} \stackrel{!}{=} 0 \tag{3.39}$$

so erhält man für die „tatsächliche“ Resonanzfrequenz

$$\omega_\mathrm{res} = \sqrt{\frac{L - R_\mathrm{L}^2 C}{L^2 C}} = \frac{1}{\sqrt{LC}} \cdot \sqrt{1 - \frac{R_\mathrm{L}^2}{Z_0}} \tag{3.40}$$

Setzt man nun für das jeweils andere Bauteil (L oder C) die Ausdrücke aus 3.37 ein, so erhält man einen Ausdruck, der nur mehr von ω_res, R_L, R'_L und L bzw. C

abhängig ist. Aus diesem wiederum lässt sich für eine gewünschte Resonanzfrequenz L und C berechnen:

$$\begin{aligned} \omega_{\mathrm{res}} &= \sqrt{\frac{L - R_{\mathrm{L}}^2 \dfrac{L}{R_{\mathrm{L}} R'_{\mathrm{L}}}}{L^2 \cdot \dfrac{L}{R_{\mathrm{L}} R'_{\mathrm{L}}}}} \qquad \rightarrow \qquad L = \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{R_{\mathrm{L}} R'_{\mathrm{L}} - R_{\mathrm{L}}^2} \\ \omega_{\mathrm{res}} &= \sqrt{\frac{R_{\mathrm{L}} R'_{\mathrm{L}} C - R_{\mathrm{L}}^2 C}{R_{\mathrm{L}}^2 R'^2_{\mathrm{L}} C^3}} \qquad \rightarrow \qquad C = \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{\frac{R'_{\mathrm{L}} - R_{\mathrm{L}}}{R_{\mathrm{L}} \cdot R'^2_{\mathrm{L}}}} \end{aligned} \tag{3.41}$$

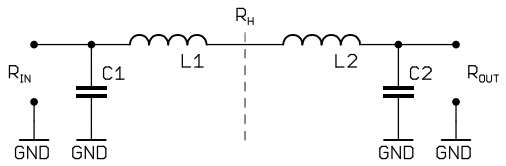


Abbildung 3.12.: π-Filter zusammengesetzt

Soll nun ein π-Glied aufgebaut werden, so werden zwei Resonanztransformationsglieder aneinandergereiht, siehe Abbildung 3.12. Der Lastwiderstand R_{OUT} wird dadurch zuerst auf einen niedrigen Zwischenwiderstand R_{H} herabtransformiert, welcher dann wieder auf den gewünschten R_{IN} hinauftransformiert wird.

Erstes π-Glied

Das erste π-Glied des zweistufigen Filters soll die niedrige Ausgangsimpedanz des Leistungs-MOSFETs auf die HF-typischen $50\,\Omega$ hochtransformieren. Der Zwischenwiderstand R_{H} muss je nach gewünschter Güte festgelegt werden. Die Güte des gesamten Oberwellenfilters sollte im Allgemeinen nicht so groß sein, dass die dadurch kleinere Bandbreite ein erneutes Abstimmen des Filters bei Änderung der Sendefrequenz im 80m-Band notwendig macht. Auch sollte das Filter so breitbandig sein, dass bei „gewissen“ Bauteilabweichungen kein Nachstimmen notwendig ist, um den Aufbau zu vereinfachen. Da diese Anforderungen unpräzise sind, wird in einigen Versuchen (Simulation) ein optimaler Zwischenwiderstand ermittelt.

$$R_{\mathrm{IN}} = 8\,\Omega \qquad R_{\mathrm{H}} = 4\,\Omega \qquad R_{\mathrm{OUT}} = 50\,\Omega \tag{3.42}$$

Daraus ergibt sich nach den Formeln 3.41 auf der vorherigen Seite ($R_\mathrm{L} = R_\mathrm{H}$ und $R'_\mathrm{L} = R_\mathrm{IN}$) folgendes für den ersten Teil des π-Glieds:

$$\begin{aligned} L1_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{R_\mathrm{H} R_\mathrm{IN} - R_\mathrm{H}^2} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz}} \cdot \sqrt{4\,\Omega \cdot 8\,\Omega - (4\,\Omega)^2} \approx 0.179\,\mu\mathrm{H} \\ C1_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{\frac{R_\mathrm{IN} - R_\mathrm{H}}{R_\mathrm{H} \cdot R_\mathrm{IN}^2}} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz}} \cdot \sqrt{\frac{8\,\Omega - 4\,\Omega}{4\,\Omega \cdot (8\,\Omega)^2}} \approx 5.604\,\mathrm{nF} \end{aligned} \tag{3.43}$$

Für den zweiten Teil des ersten π-Glieds müssen jeweils R_L und R'_L miteinander vertauscht werden, da das Glied spiegelverkehrt vorliegt. Somit wird eingesetzt:

$$\begin{aligned} R_\mathrm{L} &= R_\mathrm{H} = 4\,\Omega \\ R'_\mathrm{L} &= R_\mathrm{OUT} = 50\,\Omega \end{aligned} \tag{3.44}$$

Daraus können nun $L2$ und $C2$ des ersten π-Glieds berechnet werden:

$$\begin{aligned} L2_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{R_\mathrm{H} R_\mathrm{OUT} - R_\mathrm{H}^2} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz}} \cdot \sqrt{4\,\Omega \cdot 50\,\Omega - (4\,\Omega)^2} \approx 0.608\,\mu\mathrm{H} \\ C2_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{\frac{R_\mathrm{OUT} - R_\mathrm{H}}{R_\mathrm{H} \cdot R_\mathrm{OUT}^2}} = \frac{1}{2\pi \cdot 3.55\,\mathrm{MHz}} \cdot \sqrt{\frac{50\,\Omega - 4\,\Omega}{4\,\Omega \cdot (50\,\Omega)^2}} \approx 3.04\,\mathrm{nF} \end{aligned} \tag{3.45}$$

Zweites π-Glied

Das zweite π-Glied ändert das Impedanzverhältnis zwischen Ein- und Ausgang nicht mehr. Die Güte des Filters wird, gleich wie beim ersten π-Glied, per Simulation so getrimmt, dass der Frequenzgang zufriedenstellend ist. Somit wird für die Dimensionierung des zweiten π-Glieds ausgegangen von:

$$R_\mathrm{IN} = 50\,\Omega \qquad R_\mathrm{H} = 20\,\Omega \qquad R_\mathrm{OUT} = 50\,\Omega \tag{3.46}$$

Die einzelnen Dimensionierungsschritte sind exakt gleich wie beim ersten π-Glied, sodass hier nur mehr die Endergebnisse aufgelistet sind. Da das Glied symmetrisch ist, braucht weiters nur ein Teil berechnet werden, der andere ist spiegelgleich:

$$\begin{aligned} L1_2 &= L2_2 = 1.09\,\mu\mathrm{H} \\ C1_2 &= C2_2 = 1.09\,\mathrm{nF} \end{aligned} \tag{3.47}$$

Gesamtes Filter

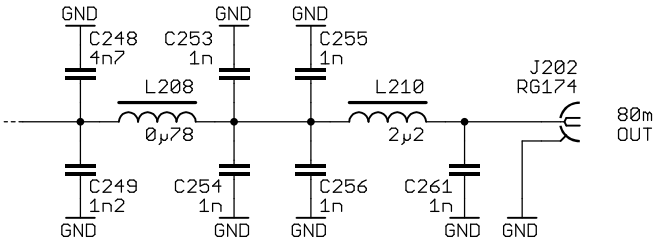


Abbildung 3.13.: Zweistufiges Oberwellenfilter

Nun müssen lediglich die einzeln berechneten Werte dort zusammengezählt werden, wo sie auf ein Bauteil zusammenfallen (siehe Abbildung 3.13).

$$\begin{aligned} L208 &= L1_1 + L2_1 = 0.179\,\mu\text{H} + 0.608\,\mu\text{H} \approx 0.78\,\mu\text{H} \\ L210 &= L1_2 + L2_2 = 1.09\,\mu\text{H} + 1.09\,\mu\text{H} \approx 2.2\,\mu\text{H} \end{aligned} \tag{3.48}$$

Diese beiden Induktivitäten sollen als Ringkernspule auf Basis des T68-2 Kerns der Firma Amidon realisiert werden. Dieser kann in einem Frequenzbereich von 2 MHz bis 30 MHz eingesetzt werden und besitzt einen magnetischen Leitwert von (vgl. *Amidon Iron Powder Cores Family Datasheet* 2007, Seite 2)

$$A_\text{L} = 57\,\mu\text{H}/(100\,\text{Wdg.})^2 = 5.7\,\text{nH}/\text{Wdg.}^2 \tag{3.49}$$

Somit ergibt sich für die beiden Induktivitäten:

$$\begin{aligned} N_{\text{L}208} &= \sqrt{\frac{L208}{A_\text{L}}} = 11.69\,\text{Wdg.} \approx 12\,\text{Wdg.} \\ N_{\text{L}210} &= \sqrt{\frac{L210}{A_\text{L}}} = 19.64\,\text{Wdg.} \approx 20\,\text{Wdg.} \end{aligned} \tag{3.50}$$

Die Kapazitäten werden teils auf zwei, teils auf vier (reale) Kondensatoren aufgeteilt, da sich einzelne Kondensatoren aufgrund der großen Ströme im Schwingkreis

bereits erwärmten. Auch lassen sich dadurch einige Werte genauer realisieren als mit einem einzelnen Kondensator aus der E-Reihe:

$$\begin{aligned} C1_1 = 5.604\,\mathrm{nF} \approx 4.7\,\mathrm{nF} + 1.2\,\mathrm{nF} \quad &\rightarrow \quad C248 = 4.7\,\mathrm{nF} \quad C249 = 1.2\,\mathrm{nF} \\ C2_1 + C1_2 = 4.13\,\mathrm{nF} \approx 4 \cdot 1\,\mathrm{nF} \quad &\rightarrow \quad C253 \ldots C256 = 1\,\mathrm{nF} \\ C2_2 = 1.09\,\mathrm{nF} \approx 1\,\mathrm{nF} \quad &\rightarrow \quad C261 = 1\,\mathrm{nF} \end{aligned} \tag{3.51}$$

Der Frequenzgang des berechneten Filters mithilfe von LTSpice wird simuliert (siehe Abbildung 3.14). Die leichte Überhöhung bei Betriebsfrequenz über die 0 dB-Grenze ist durch das Impedanzanpassverhältnis zwischen Ein- und Ausgang zu erklären, da in der Simulation Spannungen und keine Leistungen dargestellt sind.

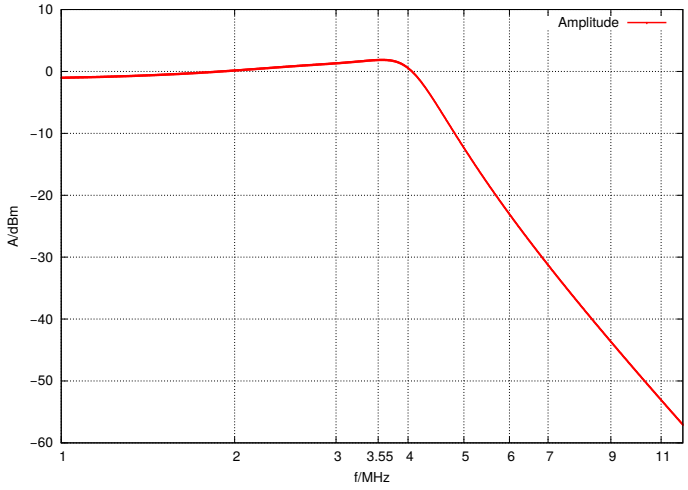


Abbildung 3.14.: Frequenzgang des Oberwellenfilters, Simulation

3.4.6. Messungen

Wirkungsgrad

Bei nominaler Ausgangsleistung von $P_{\mathrm{OUT}} = 5\,\mathrm{W}$ verbraucht der 80m-Leistungsverstärker bei $U_{\mathrm{B}} = 12V$ einen Strom von

$$I_{80\mathrm{m}} \approx 0.7\,\mathrm{A} \tag{3.52}$$

Dies bedeutet einen Wirkungsgrad von ca.

$$\eta = \frac{P_{\mathrm{OUT}}}{P_{\mathrm{IN}}} = \frac{5\,\mathrm{W}}{12\,\mathrm{V} \cdot 0.7\,\mathrm{A}} \approx 59.5\% \tag{3.53}$$

Bandbreite

Die Frequenz des Sendesignals wird durchgestimmt, um die $-3\,\mathrm{dB}$-Punkte zu ermitteln. Diese liegen bei:

$$f_{\mathrm{U}} \approx 3.1\,\mathrm{MHz} \qquad f_{\mathrm{O}} \approx 3.9\,\mathrm{MHz} \qquad \rightarrow \qquad \Delta f \approx 0.8\,\mathrm{MHz} \tag{3.54}$$

Das 80m-Amateurfunkband von 3.5 MHz bis 3.8 MHz ist somit vollständig abgedeckt.

Spektrale Reinheit

Mithilfe eines Spektrumanalysators (Start: 0 Hz, Stop: 15 MHz, RBW: 50 kHz) wird die Reinheit des Ausgangssignals überprüft (siehe Abbildung 3.15). Dazu wird ein 20 dB-Messabschwächer benutzt, welcher gleichzeitig das 50 Ω-Dummyload darstellt. In nachfolgender Grafik wurden die zusätzlichen 20 dB Abschwächung bereits korrigiert.

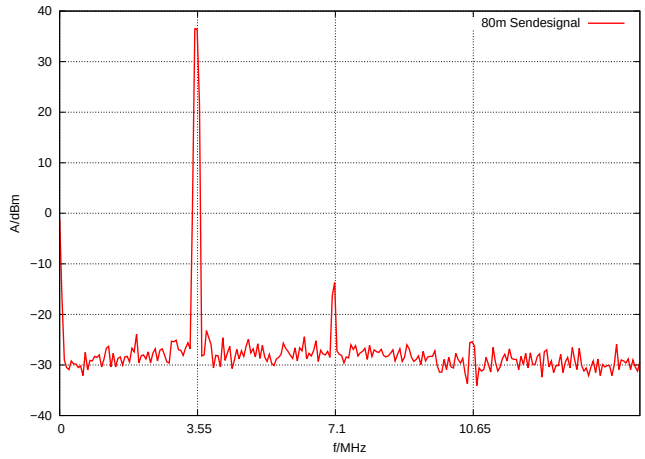


Abbildung 3.15.: Spektrum des 80m-Ausgangssignals

Der Träger bei 3.55 MHz besitzt eine Amplitude von ca. 37 dBm, was der geforderten Ausgangsleistung von 5 W entspricht. Weiters kann abgelesen werden, dass die zweite Oberwelle (bei 7.1 MHz) auf ca. −13 dBm und somit −50 dBc unter dem Träger liegt, die dritte Oberwelle noch weiter darunter (gerade noch zu erkennen). Laut Anforderungen 3.4.1 auf Seite 46 wären hier lediglich −40 dBc gefordert, sodass das erzielte Ergebnis mehr als ausreichend ist.

3.5. 2m-Leistungsverstärker

3.5.1. Anforderungen

Zweck des 2m-Leistungsverstärkers ist es, den Signalpegel des DDS im Bereich des 2m-Bandes entsprechend anzuheben, sodass mit gewünschter Leistung auf einer Antenne abgestrahlt werden kann. Weiters müssen, analog zum 80m-Leistungsverstärker (Abschnitt Anforderungen 3.4.1 auf Seite 46) die gesetzlichen Anforderungen im Bezug auf unerwünschte Nebenaussendungen eingehalten werden. Diese sind (vgl. *Änderungen der Vorordnung zur Durchführung des Amateurfunkgesetzes* 2008, Seite 2) für den Frequenzbereich von 50 MHz bis 1000 MHz:

Absenkung gegen Grundwelle:	60 dBc
Maximale Leistung:	−36 dBm

Es gilt immer der weniger strenge Wert.

Andere relevante Spezifikationen an den 2m-Leistungsverstärker werden, wie beim 80m-Leistungsverstärker, aus Anhang 1 der ARDF-Rules (vgl. *Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1* 2015, Seite 10) entnommen.

Trägerfrequenz:	144.50 MHz ... 144.90 MHz
HF-Ausgangsleistung:	0.25 W ... 1 W
Modus:	A2A
Modulationsgrad:	70% ... 80%

Für die Berechnungen der einzelnen Baugruppen wird ein representativer Wert von $f = 144.7$ MHz verwendet, welcher in der Mitte des geforderten Bereiches liegt.

Wie aus obiger Tabelle ersichtlich (A2A), muss hier, anders als beim 80m-Sendeteil, das Ausgangssignal amplitudenmoduliert werden. Dies bedeutet, dass bei gegebenem maximalen DDS-Ausgangsstrom die Ausgangsleistung des DDS im Vergleich zum 80m-Teil geringer sein wird. Die momentane Effektivwertspannung eines amplitudenmodulierten Signales kann wie folgt dargestellt werden, wobei m den Modulationsgrad bezeichnet (in Prozent).

$$U_{\text{eff}}(\phi) = U_{\text{eff}_{\text{MAX}}} \cdot \left(1 - \frac{1 + \sin(\phi)}{2} \cdot \frac{m}{100\%}\right) \tag{3.55}$$

Zur Berechnung der mittleren Leistung $\bar{P}$ wird unter Berücksichtigung, dass $P = U^2/R$, über eine „Modulationsperiode“ integriert.

$$\bar{P} = \frac{1}{2\pi} \int_0^{2\pi} \frac{U_{\text{eff}}(\phi)^2}{50\,\Omega} \mathrm{d}\phi \tag{3.56}$$

Für den konkreten Fall gilt, berücksichtigt man die unter 3.6 auf Seite 47 berechnete maximale DDS-Ausgangsleistung von $P_{\text{DDS}_{\text{MAX}}} = 156\,\mu\text{W(min)} \ldots 625\,\mu\text{W(typ)}$:

$$m = 70\% \qquad U_{\text{eff}_{\text{MAX}}} = \sqrt{P_{\text{DDS}_{\text{MAX}}} \cdot 50\,\Omega} = 88.3\,\text{mV(min)} \ldots 176.7\,\text{mV(typ)} \tag{3.57}$$

Das bestimmte Integral 3.56 wird mithilfe des freien CAS-Tools wxMaxima unter obigen Angaben berechnet.

$$\begin{aligned} \bar{P}_{\text{DDS}} &= 75.4\,\mu\text{W(min)} \ldots 302.1\,\mu\text{W(typ)} \\ &\approx -11\,\text{dBm(min)} \ldots -5\,\text{dBm(typ)} \end{aligned} \tag{3.58}$$

Auch beim 2m-Leistungsverstärker wird die maximal erlaubte Sendeleistung angestrebt, da ein nachträgliches Verringern mit weniger Aufwand verbunden ist als ein nachträgliches Erhöhen.

$$P_{\text{OUT}} = 1\,\text{W} = 30\,\text{dBm} \tag{3.59}$$

Daraus ergibt sich die notwendige Gesamtverstärkung des 2m-Sendeteils, für den typischen Fall:

$$G_{\text{TOT}} = \frac{P_{\text{OUT}}}{\bar{P}_{\text{DDS}}} = 30\,\text{dBm} - (-5\,\text{dBm}) = 35\,\text{dB} \tag{3.60}$$

Wenn der 2m-Leistungsverstärker nicht benötigt wird, soll dieser, zur Einsparung von Energie, von der Versorgung getrennt werden können. Dies wird realisiert durch die Bandumschaltung, siehe 3.3 auf Seite 44. Die Ausgangsimpedanz soll, gleich wie beim 80m-Leistungsverstärker die HF-typischen

$$Z_{\mathrm{OUT}} = 50\,\Omega \tag{3.61}$$

betragen.

3.5.2. Konzept und Bauteilauswahl

Beim 2m-Leistungsverstärker soll, im Gegensatz zum 80m-Leistungsverstärker, ein integriertes Power-Modul verwendet werden. Dadurch entfällt der Entwicklungsaufwand einer diskreten Endstufe in diesem Frequenzbereich, welcher nicht zu unterschätzen ist. Die Wahl fällt auf das RF Power Module RA08H1317M der Firma Mitsubishi. Dieses verfügt über folgende Eigenschaften (vgl. *RA08H1317M RF Power Module Datasheet* 2006, Seite 1, 3):

Betriebsspannung:	12 V mögl.
Frequenzbereich:	135 MHz ... 175 MHz
Ausgangsleistung max.:	8 W

Laut Hersteller sollte das HF-Power-Modul bei einer Eingangsleistung $P_{\mathrm{E}} < 0\,\mathrm{dBm}$ eine Leistungsverstärkung von ca.

$$G_{\mathrm{P}} \approx 37\,\mathrm{dB}\ \text{(theoretisch)} \tag{3.62}$$

aufweisen. Dieser Wert fiel im Versuchsaufbau jedoch kleiner aus. Hier konnten lediglich

$$G_{\mathrm{P}} \approx 30\,\mathrm{dB}\ \text{(real)} \tag{3.63}$$

erreicht werden. Als Gründe dafür können eine nicht perfekte Ein- und Ausgangsanpassung, eine etwas geringere Arbeitspunktspannung (siehe auch Abschnitt 3.5.5 auf Seite 73), sowie eine gewisse (nicht gänzlich vermeidbare) Induktivität in der Masseverbindung des HF-Power-Moduls und eine daraus resultierende Gegenkopplung angesehen werden.

Dadurch wird eine Vorverstärkerstufe notwendig. Diese soll als Resonanzverstärker ausgeführt werden, um unerwünschte Spektralanteile im DDS-Signal zu unterdrücken. Als aktives Element bietet sich der Kleinsignal-HF-Transistor BFR92 an, welcher mit einer Transitfrequenz von $f_{\mathrm{T}} = 5\,\mathrm{GHz}$ mehr als ausreichend ist (vgl. *BFR92 NPN Planar RF Transistor Datasheet* 2000, Seite 2).

Zuvor müssen die im DDS-Signal enthaltenen Ober- und Nebenwellen (Spurious) ausreichend unterdrückt werden, sodass diese nicht erst an den Vorverstärker gelangen und dort unerwünschte Mischprodukte entstehen. Dies geschieht noch vor dem Vorverstärker mithilfe eines Eingangsfilters.

Abbildung 3.16 zeigt das Blockschaltbild des 2m-Leistungsverstärkers.

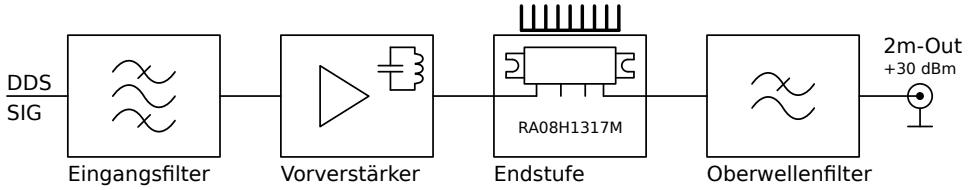


Abbildung 3.16.: Blockschaltbild 2m-Leistungsverstärker

3.5.3. Eingangsfilter

Bei der DDS-Signalerzeugung entstehen unvermeidbar diverse Nebenprodukte im Spektrum des DDS-Signals (siehe Abschnitt 3.2.6 auf Seite 42). Diese werden erstens direkt mitverstärkt, bzw. vom Schwingkreis des Vorverstärkers allein nicht ausreichend unterdrückt. Zweitens entstehen an der nichtlinearen Kennlinie des Vorverstärker-Transistors verschiedene Mischprodukte, welche teils nahe am Nutzsignal liegen, und somit ebenfalls den Schwingkreis des Vorverstärkers passieren können. Dadurch gelangen die Nebenprodukte in das Sendesignal, und die Anforderungen im Bezug auf spektrale Reinheit (siehe 3.5.1 auf Seite 64) könnten nicht eingehalten werden.

Um dies zu verhindern, wird direkt nach der PIN-Dioden Umschaltung ein schmalbandiges Bandpassfilter eingesetzt (siehe Abbildung 3.17).

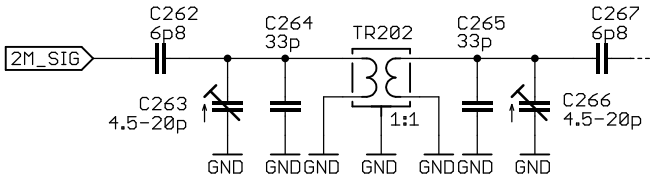


Abbildung 3.17.: Schaltplan 2m-Eingangsfilter

Das Filter besteht aus zwei gekoppelten Schwingkreisen, gebildet jeweils durch Primär- und Sekundärwicklung von TR202 mit einem fixen Kondensator ($C264$

und $C265$) parallel einem Trimmkondensator ($C263$ und $C266$). Die Kopplung erfolgt induktiv zwischen den beiden Schwingkreisspulen. TR202 befindet sich in einem Blech-Schirmgehäuse, um Einkopplungen von anderen, offenen Induktivitäten (besonders vom 2m-Oberwellenfilter) zu unterbinden. Sonst neigt die Schaltung, bedingt durch die hohe Gesamtverstärkung, zur Oszillation.

Durch $C262$ und $C267$ erfolgt (in Verbindung mit den beiden Schwingkreisinduktivitäten) eine Resonanztransformation des 50 Ω Ein- und Ausgangswiderstandes auf einen höheren Zwischenwiderstand. Dadurch werden die beiden Schwingkreise entdämpft und höhere Güten erreicht.

Der Trafo besteht aus einem HF-Filterbausatz mit entferntem Kern und jeweils einer Windung primär und sekundär. Da die Parameter bei Betriebsfrequenz nicht genauer bekannt sind, erfolgt die Dimensionierung des Filters basierend auf Messwerten. Mithilfe eines Netzwerkanalysators werden die Induktivitäten ermittelt:

$$L_{\mathrm{PRI}} \approx L_{\mathrm{SEC}} \approx 21\,\mathrm{nH} \tag{3.64}$$

Bei einer gewünschten Resonanzfrequenz von 144.7 MHz würde dies eine Schwingkreiskapazität von ca. $L_{\mathrm{SK}} \approx 57\,\mathrm{pF}$ bedeuten. Werden stattdessen $C_{\mathrm{SK}} = 47\,\mathrm{pF}$ eingesetzt, so muss die Induktivität ca. $L_{\mathrm{SK}} \approx 25.7\,\mathrm{nH}$ betragen. Die vorhandene Induktivität des Trafos kann als Parallelschaltung einer 25.7 nH-Spule mit einer zweiten Spule

$$L_{\mathrm{RT}} = \frac{1}{\frac{1}{L_{\mathrm{SK}}} - \frac{1}{L_{\mathrm{PRI}}}} \approx 159\,\mathrm{nH} \tag{3.65}$$

angesehen werden, welche zur Resonanztransformation benutzt wird. Der dafür benötigte Kondensator ($C262$ bzw. $C267$) berechnet sich ideal betrachtet:

$$C_{\mathrm{RT}} = C262 = C267 = \frac{1}{\omega_{\mathrm{res}}^2 \cdot L_{\mathrm{RT}}} = \frac{1}{4\pi^2 \cdot (144.7\,\mathrm{MHz})^2 \cdot 159\,\mathrm{nH}} \approx 7.6\,\mathrm{pF} \approx 6.8\,\mathrm{pF} \tag{3.66}$$

Versuche haben gezeigt, dass ein 6.8 pF-Kondensator die gewünschten Ergebnisse erzielt.

Theoretisch transformiert diese Dimensionierung die 50 Ω ein- und ausgangsseitig (R_{L}) hinauf auf folgenden Dämpfungswiderstand im Schwingkreis (R'_{L}):

$$R_{\mathrm{L}} \cdot R'_{\mathrm{L}} = Z_0^2 \qquad \rightarrow \qquad R'_{\mathrm{L}} = \frac{L_{\mathrm{RT}}}{C_{\mathrm{RT}} \cdot R_{\mathrm{L}}} \approx 476\,\Omega \tag{3.67}$$

Der Kopplungsfaktor kann durch den Abstand der beiden 1-Windung-Spulen auf dem Spulenkörper des Trafos eingestellt werden. Das Optimum wird empirisch unter Zuhilfenahme eines Netzwerkanalysators ermittelt.

Die Schwingkreiskapazität $C_{\mathrm{SK}} = 47\,\mathrm{pF}$ wird sowohl primär- als auch sekundärseitig abstimmbar ausgeführt, um die Resonanzfrequenz optimal einstellen zu können.

$$C264 = C265 = 33\,\mathrm{pF} \qquad C263 = C266 = 4.5\,\mathrm{pF} \ldots 20\,\mathrm{pF} \tag{3.68}$$

Mithilfe des Netzwerkanalysators wird der Amplitudengang des Filters aufgenommen, siehe Abbildung 3.18.

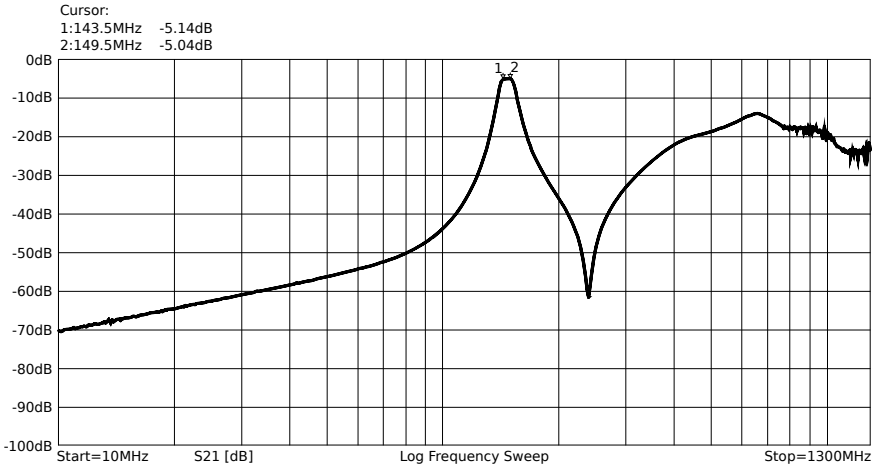


Abbildung 3.18.: Amplitudengang des Eingangsfilters, Messung

3.5.4. Vorverstärker

Der Vorverstärker soll grundsätzlich als Resonanzverstärker in Emitterschaltung im A-Betrieb aufgebaut werden. Ursprünglich wurde hier die Verwendung einer Basisschaltung in Erwägung gezogen, da diese als nichtinvertierende Verstärkerschaltung nicht von der Bandbreitenbegrenzung aufgrund des Millereffekts betroffen ist. Allerdings geht damit einher eine niedrige Eingangsimpedanz (hier im Ω-Bereich), welche eine zusätzliche Eingangsanpassung notwendig gemacht hätte. Diese Notwendigkeit besteht bei der Emitterschaltung aufgrund der dort höheren Eingangsimpedanz nicht. Abbildung 3.19 auf der nächsten Seite zeigt den Schaltplan des 2m-Vorverstärkers.

Zur Arbeitspunkteinstellung bzw. Stabilisierung wird, gleich wie beim 80m-Vorverstärker (siehe Abschnitt 3.4.3 auf Seite 49) mithilfe einer 5.6 V-Zenerdiode eine versorgungsunabhängige Referenzspannung erzeugt, welche auch für die Arbeitspunkteinstellung des HF-Power-Modules (Endstufe) dient. Aus diesem Grund befindet sich die Zenerdiode ($D203$) auf dem Schaltplan der Endstufe, siehe Abbildung 3.20 auf Seite 73.

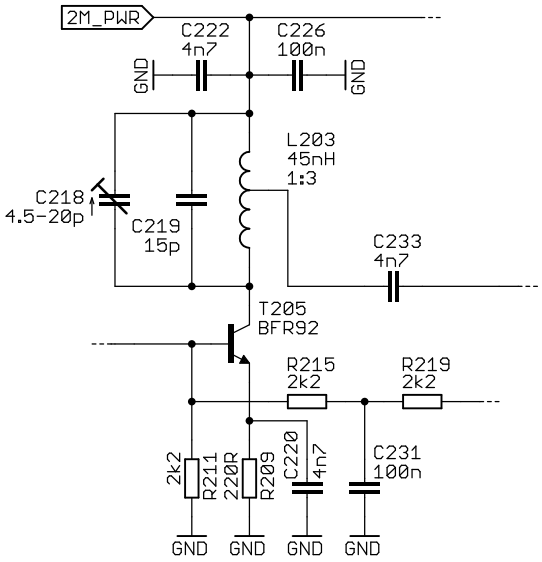


Abbildung 3.19.: Schaltplan 2m-Vorverstärker

Aus der Referenzspannung wird mittels Spannungsteiler ($R211$, $R215$ und $R219$) in Verbindung mit einem Emitterwiderstand ($R209$) der Arbeitspunktstrom eingestellt. Eine gängige Faustformel besagt hier, dass für einen stabilen Arbeitspunkt die Spannung am Emitterwiderstand

$$U_{\mathrm{R209}} > \frac{U_{\mathrm{B}}}{10} = 1.2\,\mathrm{V} \tag{3.69}$$

betragen soll.

Versuche haben gezeigt, dass ein Arbeitspunktstrom von $I_{\mathrm{C}} \approx 5\,\mathrm{mA}$ zur gewünschten Verstärkung führt. Daraus kann der Emitterwiderstand berechnet werden:

$$R209 = \frac{U_{\mathrm{R209}}}{I_{\mathrm{C}}} = \frac{1.2\,\mathrm{V}}{5\,\mathrm{mA}} = 240\,\Omega \approx 220\,\Omega \tag{3.70}$$

Wird der Einfachheit halber von einem Querstrom I_Q durch den Spannungsteiler in der Größenordnung von 1 mA ausgegangen, so ergibt sich mit einer Basis-Emitter-Spannung von T205 $U_\mathrm{BE} \approx 0.7\,\mathrm{V}$ daraus

$$R211 = \frac{U_\mathrm{R209} + U_\mathrm{BE}}{1\,\mathrm{mA}} = \frac{1.9\,\mathrm{V}}{1\,\mathrm{mA}} = 1.9\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \tag{3.71}$$

Bei dieser Berechnung wird der Basisstrom vernachlässigt, was jedoch gerechtfertigt ist, da der Transistor über eine Stromverstärkung von typisch $\beta \approx h_\mathrm{FE} \approx 50$ verfügt, und somit der Basisstrom

$$I_\mathrm{B} = \frac{I_\mathrm{C}}{h_\mathrm{FE}} \approx 50\,\mu\mathrm{A} \tag{3.72}$$

beträgt.

Damit verbleiben noch 3.7 V für $R215$ und $R219$, wodurch sich anbietet:

$$\begin{aligned} & R215 = 2.2\,\mathrm{k\Omega} \\ & R219 = 2.2\,\mathrm{k\Omega} \\ \rightarrow \quad & U_\mathrm{R211} = U_\mathrm{D203}\frac{R211}{R211 + R215 + R219} = 5.6\,\mathrm{V} \cdot \frac{2.2\,\mathrm{k\Omega}}{6.6\,\mathrm{k\Omega}} \approx 1.86\,\mathrm{V} \\ \rightarrow \quad & I_\mathrm{Q} = \frac{U_\mathrm{D203}}{R211 + R215 + R219} = \frac{5.6\,\mathrm{V}}{6.6\,\mathrm{k\Omega}} = 0.85\,\mathrm{mA} \end{aligned} \tag{3.73}$$

Zusätzlich bilden $R219$ und $C231$ einen Tiefpass ($f_\mathrm{g} \approx 723\,\mathrm{Hz}$), um hochfrequente Störungen, unter anderem auch das Rauschen der Zenerdiode, abzublocken, sodass diese nicht von T205 mitverstärkt werden.

Besondere Bedeutung kommt $C220$ zu, welcher den Emitter für Nutzsignale wechselspannungsmäßig auf Masse kurzschließt. Dafür ist eine niedrige Impedanz erforderlich. Bei der Betriebsfrequenz von $f = 144.7\,\mathrm{MHz}$ müssen neben dem kapazitiven Blindwiderstand auch die parasitären Eigenschaften des verwendeten Kondensators beachtet werden. So besitzt dieser immer auch parasitäre Induktivitäten (Anschlüsse etc.), welche eine Serienresonanz verursachen. Diese liegt für Mehrschicht-Keramikkondensatoren (Material NP0) im einstelligen Nanofarad-Bereich um 100 MHz (vgl. *Impedance curves of ceramic capacitors* 2012). Mit Überschreiten des Resonanzpunktes nimmt die Impedanz des Kondensators wieder zu. Für diese Anwendung fällt die Entscheidung auf

$$C220 = 4.7\,\mathrm{nF} \tag{3.74}$$

Zumindest theoretisch ergibt dies einen Blindwiderstand bei Betriebsfrequenz von

$$X_{\mathrm{C220}} = \frac{1}{2\pi \cdot f \cdot C220} = 0.23\,\Omega \tag{3.75}$$

Derselbe Wert wird auch für den Ausgangs-Koppelkondensator verwendet:

$$C233 = 4.7\,\mathrm{nF} \tag{3.76}$$

Die Abblockkondensatoren für die Versorgungsspannung bestehen aus 100 nF parallel 4.7 nF, sodass eine niedrige Impedanz auch bei Betriebsfrequenz gewährleistet ist.

Der Resonanzkreis besteht aus der Luftspule $L203$ sowie $C219$ parallel mit einem Trimmkondensator, $C218$. Der Lastwiderstand der Schaltung ist der Eingangswiderstand des nachfolgenden HF-Power-Moduls, welcher $R_{\mathrm{L}} = 50\,\Omega$ beträgt. Zur Erhöhung der Güte im Schwingkreis wird die Spule mit einer Anzapfung versehen, sodass der Lastwiderstand im Kreis hochtransformiert erscheint. Hier wird angesetzt:

$$\ddot{U} = 1:3 \text{ (Anzapfung : Gesamtwindungszahl)} \tag{3.77}$$

Dies bewirkt eine Widerstandstransformation von

$$R'_{\mathrm{L}} = R_{\mathrm{L}} \cdot \ddot{U}^2 = 450\,\Omega \tag{3.78}$$

Nun stellt sich zur Dimensionierung die Frage nach dem Verhältnis zwischen L und C im Schwingkreis. Mithilfe eines Netzwerkanalysators wird ermittelt, dass eine Luftspule aus 1 mm-Silberdraht mit Durchmesser $d = 7\,\mathrm{mm}$ (6 mm Schaft), Länge $l \approx 5\,\mathrm{mm}$ und drei Windungen eine Induktivität von ca.:

$$L203 \approx 45\,\mathrm{nH} \tag{3.79}$$

besitzt. Drei Windungen sind mindestens notwending, um ein Transformationsverhältnis von 1 : 3 zu realisieren.

Daraus ergibt sich bei Betriebsfrequenz $f_{\mathrm{res}} = 144.7\,\mathrm{MHz}$ für Kondensator und Scheinwiderstand im Schwingkreis:

$$\begin{aligned} \omega_{\mathrm{res}} &= \frac{1}{\sqrt{LC}} \quad \rightarrow \quad C = \frac{1}{4\pi^2 \cdot f_{\mathrm{res}}^2 \cdot L203} \approx 27\,\mathrm{pF} \\ Z_0 &= \sqrt{\frac{L}{C}} \approx 41\,\Omega \end{aligned} \tag{3.80}$$

Die berechnete Kapazität im Schwingkreis wird zweigeteilt ausgeführt, da Trimmkondensatoren ($C218$) für HF-Anwendungen mit Abstimmbereich von ca. 5 pF bis 20 pF sehr gängig sind. Der zweite Kondensator $C219$ wird so gewählt, dass die gewünschte Gesamtkapazität in etwa in Mittelstellung des Trimmkondensators erreicht wird, sozusagen auf beiden Seiten noch Abgleichspielraum besteht.

$$C218 = 4.5\,\text{pF} \ldots 20\,\text{pF} \qquad C219 = 15\,\text{pF} \tag{3.81}$$

3.5.5. Endstufe

Das HF-Power-Modul RA08H1317M von Mitsubishi muss im Wesentlichen nur mit Betriebsspannung versorgt, sowie am Pin „VGG“ mit einer Arbeitspunktspannung beaufschlagt werden, dazu siehe Abbildung 3.20.

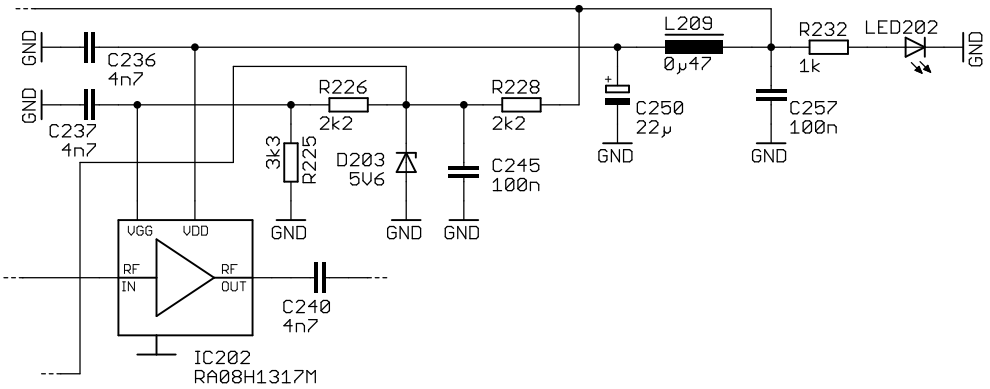


Abbildung 3.20.: Schaltplan 2m-Endstufe

Die Arbeitspunkteinstellung erfolgt mithilfe der Zenerdiode $D203$, welche eine versorgungsunabhängige Spannung für das HF-Power-Modul und für den Vorverstärker zur Verfügung stellt. Die Wahl fällt hier, gleich wie beim 80-Leistungsverstärker auf $U_{\text{D203}} = 5.6\,\text{V}$. Diese wird jeweils mit Spannungsteilern auf die gewünschten Arbeitspunktspannungen heruntergeteilt. Für jeden Spannungsteiler wird ein Laststrom von ca. $I_{\text{L}} \approx 1\,\text{mA}$ angenommen. Soll der Zenerdiodenstrom ebenfalls $I_{\text{D203}} = 1\,\text{mA}$ betragen, so ergibt sich:

$$R228 = \frac{U_{\text{B}} - U_{\text{D203}}}{2 \cdot I_{\text{L}} + I_{\text{D203}}} = \frac{6.4\,\text{V}}{3\,\text{mA}} = 2.13\,\text{k}\Omega \approx 2.2\,\text{k}\Omega \tag{3.82}$$

$R225$ und $R226$ teilen die Zenerdiodenspannung herab auf die benötigte Arbeitspunktspannung $V_{\text{GG}} = U_{\text{GG}}$. Hier wird im Datenblatt ein Wert zwischen 3 V und

3.5 V angegeben (vgl. *RA08H1317M RF Power Module Datasheet* 2006, Seite 1).

$$U_{\mathrm{GG}} = 3.25\,\mathrm{V} \tag{3.83}$$

Mit $I_{\mathrm{Q}} = 1\,\mathrm{mA}$ ergibt sich unter Vernachlässigung des Stromes in das Modul:

$$\begin{aligned} R225 &= \frac{U_{\mathrm{GG}}}{I_{\mathrm{Q}}} = \frac{3.25\,\mathrm{V}}{1\,\mathrm{mA}} = 3.25\,\mathrm{k\Omega} \approx 3.3\,\mathrm{k\Omega} \\ R226 &= \frac{U_{\mathrm{D203}} - U_{\mathrm{GG}}}{I_{\mathrm{Q}}} = \frac{2.35\,\mathrm{V}}{1\,\mathrm{mA}} = 2.35\,\mathrm{k\Omega} \approx 2.2\,\mathrm{k\Omega} \end{aligned} \tag{3.84}$$

Um zu verhindern, dass die „hochfrequente“ Stromaufnahme des Moduls Störungen in anderen Teilen der Schaltung verursacht, wird die Versorgung mit einem LC-Tiefpass abgeblockt. $L209$ wurde von Würth Elektronik so ausgewählt, dass deren Eigenresonanzfrequenz von 270 MHz höher als die Betriebsfrequenz der Schaltung liegt. Mit einer Induktivität von 0.47 µH ergibt dies bei Betriebsfrequenz immer noch einen Blindwiderstand von

$$X_{\mathrm{L209}} = 2\pi \cdot f \cdot L209 \approx 427\,\Omega \tag{3.85}$$

Die nachfolgenden Abblockkondensatoren werden aus der Testschaltung des Datenblattes übernommen (vgl. *RA08H1317M RF Power Module Datasheet* 2006, Seite 6):

$$C236 = 4.7\,\mathrm{nF} \qquad C250 = 22\,\mathrm{\mu F} \tag{3.86}$$

3.5.6. Oberwellenfilter

Das Oberwellenfiter des 2m-Leistungsverstärkers soll, gleich wie beim 80m-Teil, als zweistufiges π-Tiefpassfilter aufgebaut werden. Die Formeln zur Berechnung wurden bereits in diesem Abschnitt hergeleitet (siehe 3.4.5 auf Seite 56) und werden hier nochmals verwendet.

Im Datenblatt des HF-Power-Moduls ist für die zweite Oberwelle maximal $-25\,\mathrm{dBc}$ angegeben (vgl. *RA08H1317M RF Power Module Datasheet* 2006, Seite 2). Entsprechend dem gesetzlichen Maximalwert für unerwünschte Nebenaussendungen in diesem Frequenzbereich (siehe Abschnitt 3.5.1 auf Seite 64) ergibt sich die geforderte Dämpfung A_{2nd} für die zweite Oberwelle, welche das Filter theoretisch „leisten“ muss:

$$A_{\mathrm{2nd}} = 60\,\mathrm{dB} - 25\,\mathrm{dBc} = 35\,\mathrm{dB} \tag{3.87}$$

Um diese Dämpfung zu erreichen, ist mindestens ein zweistufiges π-Filter notwenig (6. Ordnung mit $-6\,\mathrm{dB/Oktave/Ordnung} = -36\,\mathrm{dB/Oktave}$).

Dimensionierung

Da sowohl der Ausgang des HF-Power-Moduls als auch der gewünschte Ausgangswiderstand $50\,\Omega$ betragen, muss vom Filter keine Impedanzanpassung zwischen Ein- und Ausgang stattfinden. Aus diesem Grund kann das Filter symmetrisch aufgebaut werden. Da zur Realisierung des 2m-Filters Luftspulen und Kondensatoren im pF-Bereich verwendet werden müssen, wird sich aufgrund der Toleranzen, Streukapazitäten etc. ein Abstimmen des Filters nicht umgehen lassen. Dadurch muss das Filter nicht mehr so breitbandig sein wie beim 80m-Teil. In einigen Versuchen wird per Simulation eine optimale Impedanzabfolge im zweistufigen π-Filter ermittelt.

$$R_{\mathrm{IN}} = 50\,\Omega \qquad R_{\mathrm{H1}} = 10\,\Omega \qquad R_{\mathrm{M}} = 1000\,\Omega \qquad R_{\mathrm{H2}} = 10\,\Omega \qquad R_{\mathrm{OUT}} = 50\,\Omega \tag{3.88}$$

R_{H1} und R_{H2} stellen hier die „virtuellen“ Zwischenwiderstände in den beiden π-Filtern dar, R_{M} den „Mittenwiderstand“, d.h. den Übergangswiderstand zwischen erstem und zweitem π-Filter.

Die Formeln zur Berechnung eines π-Filters anhand zweier Resonanztransformationsglieder wurde bereits im Abschnitt 3.4.5 auf Seite 56 hergeleitet, diese werden hier nochmals verwendet:

$$\begin{aligned} L &= \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{R_{\mathrm{L}} R'_{\mathrm{L}} - R_{\mathrm{L}}^2} \\ C &= \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{\frac{R'_{\mathrm{L}} - R_{\mathrm{L}}}{R_{\mathrm{L}} \cdot R'^2_{\mathrm{L}}}} \end{aligned} \tag{3.89}$$

Diese Formeln gelten für die „Aufwärtstransformation“, d.h. ein kleiner R_{L} wird auf einen größeren $R_{\mathrm{L}'}$ transformiert. Soll in die andere Richtung transformiert werden, müssen jeweils R_{L} und R'_{L} miteinander vertauscht werden.

Für den ersten Teil des ersten π-Gliedes ergibt dies, eingesetzt mit $R_{\mathrm{L}} = R_{\mathrm{H1}}$ und $R'_{\mathrm{L}} = R_{\mathrm{IN}}$ (Bezeichnungen nach Abbildung 3.21 auf der nächsten Seite):

$$\begin{aligned} L1_1 &= \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{R_{\mathrm{H1}} R_{\mathrm{IN}} - R_{\mathrm{H1}}^2} = \frac{1}{2\pi \cdot 144.7\,\mathrm{MHz}} \cdot \sqrt{10\,\Omega \cdot 50\,\Omega - (10\,\Omega)^2} \approx 22\,\mathrm{nH} \\ C1_1 &= \frac{1}{\omega_{\mathrm{res}}} \cdot \sqrt{\frac{R_{\mathrm{IN}} - R_{\mathrm{H1}}}{R_{\mathrm{H1}} \cdot R_{\mathrm{IN}}^2}} = \frac{1}{2\pi \cdot 144.7\,\mathrm{MHz}} \cdot \sqrt{\frac{50\,\Omega - 10\,\Omega}{10\,\Omega \cdot (50\,\Omega)^2}} \approx 44\,\mathrm{pF} \end{aligned} \tag{3.90}$$

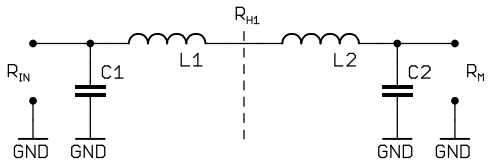


Abbildung 3.21.: Erstes π-Glied

Für den zweiten Teil des ersten π-Gliedes wird nun eingesetzt $R_\mathrm{L} = R_\mathrm{H1}$ und $R'_\mathrm{L} = R_\mathrm{M}$:

$$\begin{aligned} L2_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{R_\mathrm{H1} R_\mathrm{M} - R_\mathrm{H1}^2} = \frac{1}{2\pi \cdot 144.7\,\mathrm{MHz}} \cdot \sqrt{10\,\Omega \cdot 1000\,\Omega - (10\,\Omega)^2} \approx 109\,\mathrm{nH} \\ C2_1 &= \frac{1}{\omega_\mathrm{res}} \cdot \sqrt{\frac{R_\mathrm{M} - R_\mathrm{H1}}{R_\mathrm{H1} \cdot R_\mathrm{M}^2}} = \frac{1}{2\pi \cdot 144.7\,\mathrm{MHz}} \cdot \sqrt{\frac{1000\,\Omega - 10\,\Omega}{10\,\Omega \cdot (1000\,\Omega)^2}} \approx 11\,\mathrm{pF} \end{aligned} \tag{3.91}$$

Für das zweite π-Filter müssen die Werte nicht separat berechnet werden, da wie bereits oben erwähnt, das Filter symmetrisch aufgebaut wird.

$$\begin{aligned} L1_2 &= 109\,\mathrm{nH} & L2_2 &= 22\,\mathrm{nH} \\ C1_2 &= 11\,\mathrm{pF} & C2_2 &= 44\,\mathrm{pF} \end{aligned} \tag{3.92}$$

Gesamtes Filter

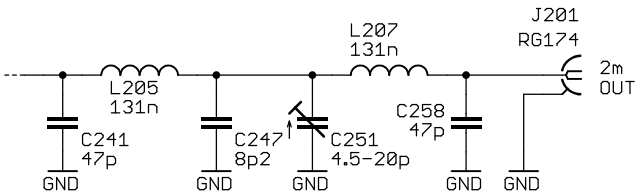


Abbildung 3.22.: Zweistufiges Oberwellenfilter

Nun müssen die separat berechneten Werte dort zusammengezählt werden, wo sie auf ein Bauteil zusammenfallen (siehe Abbildung 3.22 auf der vorherigen Seite).

$$\begin{aligned} L205 &= L1_1 + L2_1 = 22\,\mathrm{nH} + 109\,\mathrm{nH} = 131\,\mathrm{nH} \\ L207 &= 131\,\mathrm{nH}\ \text{(Symmetrie)} \end{aligned} \tag{3.93}$$

Die beiden Induktivitäten sollen als Luftspule aus Silberdraht (1 mm Durchmesser) realisiert werden. Allgemein gilt für eine „lange“ Spule:

$$L = \frac{\mu_0 \cdot \mu_\mathrm{r} \cdot A \cdot N^2}{l} \tag{3.94}$$

Fasst man die Konstanten zusammen, nimmt einen runden Querschnitt mit dem Durchmesser d an, und rechnet die Abmessungen in Millimeter um, so erhält man:

$$L \approx 0.98\,\mathrm{nH} \cdot \frac{d^2_{(\mathrm{mm})} \cdot N^2}{l_{(\mathrm{mm})}} \tag{3.95}$$

Wird zum Wickeln der Spule ein 5 mm-Schaft vorausgesetzt (z.B. Bohrerschaft), so ergibt dies bei einer Drahtdicke von 1 mm einen mittleren Spulendurchmesser von 6 mm. Geht man weiters von 10 mm Länge aus, was auf der Platine noch günstig zu handhaben ist, so erhält man für die gewünschte $L = 131\,\mathrm{nH}$ die notwendige Windungszahl

$$\begin{aligned} & d = 6\,\mathrm{mm} \qquad l = 10\,\mathrm{mm} \\ \rightarrow \quad & N = \sqrt{\frac{L \cdot l_{(\mathrm{mm})}}{0.98\,\mathrm{nH} \cdot d_{(\mathrm{mm})}}} = 6.09\,\mathrm{Wdg.} \approx 6\,\mathrm{Wdg.} \end{aligned} \tag{3.96}$$

Es bleiben noch die Kondensatoren:

$$\begin{aligned} C241 &= C1_1 = 44\,\mathrm{pF} \approx 47\,\mathrm{pF} \\ C258 &= C2_2 \approx 47\,\mathrm{pF}\ \text{(Symmetrie)} \end{aligned} \tag{3.97}$$

Die „mittlere“ Kapazität ist geteilt auf den Trimmkondensator $C251$ und den fixen Kondensator $C247$. Da der verwendete abstimmbare Kondensator einen Bereich von ca. 4.5 pF bis 20 pF besitzt, in Mittelstellung somit etwa $C251_\mathrm{MID} \approx 13\,\mathrm{pF}$, ergibt sich daraus

$$C247 = C1_2 + C2_1 - C251_\mathrm{MID} = 2 \cdot 11\,\mathrm{pF} - 13\,\mathrm{pF} = 9\,\mathrm{pF} \approx 8.2\,\mathrm{pF} \tag{3.98}$$

Der Frequenzgang des dimensionierten Filters wird im Vorhinein mithilfe von LTSpice simuliert, siehe Abbildung 3.23. Bei Resonanz weist das Filter eine Abschwächung um $-6\,\mathrm{dB}$ auf, welche durch die Leistungsanpassung zwischen Ein- und Ausgang zustande kommt. Im Diagramm sind Spannungspegel und keine Leistungen aufgetragen.

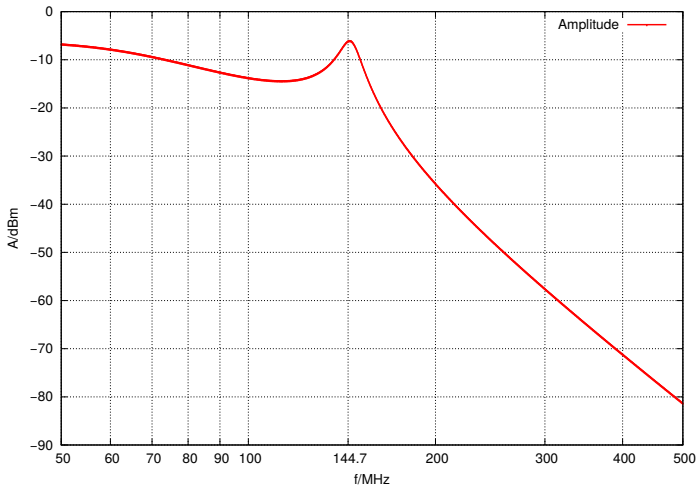


Abbildung 3.23.: Frequenzgang des Filters, Simulation

3.5.7. Messungen

Wirkungsgrad

Der 2m-Leistungsverstärker nimmt bei geforderter Ausgangsleistung von $P_{\mathrm{OUT}} = 1\,\mathrm{W}$ einen Strom von

$$I_{2\mathrm{m}} \approx 0.5\,\mathrm{A} \tag{3.99}$$

auf. Dies bedeutet bei einer Betriebsspannung von $12\,\mathrm{V}$ einen Wirkungsgrad von

$$\eta = \frac{P_{\mathrm{OUT}}}{P_{\mathrm{OUT}}} = \frac{1\,\mathrm{W}}{12\,\mathrm{V} \cdot 0.5\,\mathrm{A}} \approx 16.6\% \tag{3.100}$$

Dieser schlechte Wirkungsgrad kann unter anderem darauf zurückgeführt werden, dass das HF-Power-Modul für eine Nennleistung von $8\,\mathrm{W}$ ausgelegt ist, hier jedoch nur mit einem Bruchteil davon betrieben wird. Allerdings ist der Gesamt-Energieverbrauch des 2m-Leistungsverstärker trotz des schlechten Wirkungsgrades immer noch geringer als der des 80m-Leistungsverstärkers ($I_{80\mathrm{m}} \approx 0.7\,\mathrm{A}$).

Bei einer gewissen angestrebten Akkulaufzeit muss deshalb nur letzterer berücksichtigt werden, weshalb keine weiteren Maßnahmen unternommen wurden, den Wirkungsgrad des 2m-Leistungsverstärkers zu verbessern.

Bandbreite

Die beiden $-3\,\mathrm{dB}$-Punkte des 2m-Leistungsverstärkers liegen bei optimaler Abstimmung des Eingangsfilters, des Vorverstärkers und des Oberwellenfilters bei

$$f_\mathrm{U} \approx 140.5\,\mathrm{MHz} \qquad f_\mathrm{O} \approx 150.5\,\mathrm{MHz} \qquad \rightarrow \qquad \Delta f \approx 10\,\mathrm{MHz} \tag{3.101}$$

Somit ist das gesamte 2m-Amateurfunkband (144 MHz bis 146 MHz) abgedeckt, es besteht darüber hinaus eine gewisse Toleranz bezüglich Fehleinstellung der einzelnen Komponenten beim Abstimmvorgang.

Spektrale Reinheit

Mithilfe eines Spektrumanalysators (Start: 0 Hz, Stop: 500 MHz, RBW: 250 kHz) wird das Ausgangssignal (reiner Träger) untersucht. Dazu wird wie beim 80m-Teil ein 20 dB-Abschwächer benutzt, welcher in Abbildung 3.24 jedoch nicht aufscheint.

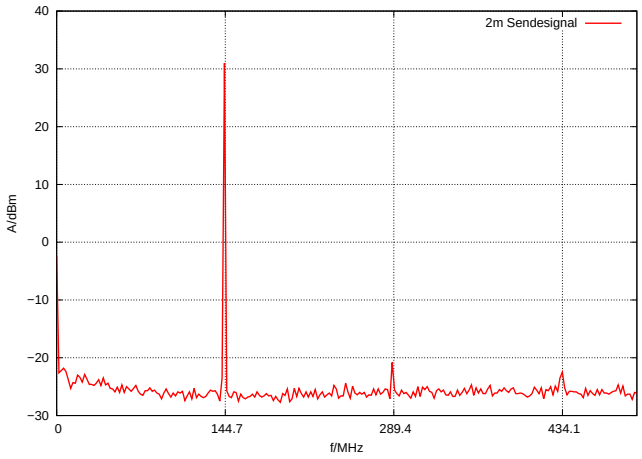


Abbildung 3.24.: Spektrum des 2m-Ausgangssignals

Der Träger bei 144.7 MHz besitzt eine Amplitude von etwas über 30 dBm, entsprechend etwas mehr als 1 W. Die zweite und dritte Oberwelle liegen, gerade noch erkennbar, bei ca. −53 dBc und −55 dBc. Hier konnte der gesetzlich geforderte Grenzwert von −60 dBc (siehe 3.5.1 auf Seite 64) nicht gänzlich eingehalten werden. Allerdings wirkt die Sendeantenne als Bandpassfilter auf ihrer Resonanzfrequenz, wodurch eine zusätzliche Oberwellenunterdrückung zumindest auf der 2. Oberwelle stattfindet.

3.6. Kühlkörper

Auf der DDS-HF-Platine befinden sich zwei Bauteile, die größere Verlustleistungen freisetzen (80m-Endstufentransistor und 2m-HF-Power-Modul), sodass diese auf einem Kühlkörper montiert werden müssen. Damit der eventuell später auch außerhalb eines Gehäuses angebracht werden kann (zur besseren Durchlüftung), soll ein gemeinsamer Kühlkörper am Rand der Platine benutzt werden.

Die Wirkungsgrade der beiden Endstufen sind von vornherein nicht genau bekannt, deshalb werden die abzuführenden Verlustleistungen empirisch ermittelt. Dazu wird die Stromaufnahme beider Endstufen bei $U_\mathrm{B} = 12\,\mathrm{V}$ Betriebsspannung und nominaler Ausgangsleistung gemessen:

	Stromaufnahme	**Ausgangsleistung**
80m-Endstufe	$I_\mathrm{80m} \approx 0.7\,\mathrm{A}$	$P_{\mathrm{OUT}_\mathrm{80m}} = 5\,\mathrm{W}$
2m-Endstufe	$I_\mathrm{2m} \approx 0.5\,\mathrm{A}$	$P_{\mathrm{OUT}_\mathrm{2m}} = 1\,\mathrm{W}$

Tabelle 3.2.: Stromverbrauch der Endstufen

Daraus ergibt sich deren Verlustleistung:

$$\begin{aligned} P_\mathrm{V\text{-}80m} &= U_\mathrm{B} \cdot I_\mathrm{80m} - P_\mathrm{OUT\text{-}80m} = 3.4\,\mathrm{W} \\ P_\mathrm{V\text{-}2m} &= U_\mathrm{B} \cdot I_\mathrm{2m} - P_\mathrm{OUT\text{-}2m} = 5\,\mathrm{W} \end{aligned} \tag{3.102}$$

In folgender Auslegung wird von einer maximalen Umgebungstemperatur

$$T_\mathrm{U} = 60\,^\circ\mathrm{C} \tag{3.103}$$

ausgegangen.

80m-Teil

Der 80m-Endstufentransistor besitzt folgende Eigenschaften (vgl. *RD06HHF1 RF Power Transistor Datasheet* 2011, Seite 2):

$$\begin{aligned} R_{\text{TH-JC}} &= 4.5\,\text{K/W} \quad \text{(Junction-Case Übergangswiderstand)} \\ T_{\text{J-max}} &= 150\,^{\circ}\text{C} \quad \text{(maximale Junction-Temperature)} \end{aligned}$$

Damit noch etwas Puffer besteht, wird davon ausgegangen, dass die Junction-Temperature $T_{\text{J-max}} = 120\,^{\circ}\text{C}$ nicht übersteigen soll. Dazu ist folgender Wärmewiderstand notwendig:

$$R_{\text{TH-ges}} = \frac{T_{\text{J-max}} - T_{\text{U}}}{P_{\text{V-80m}}} = \frac{60\,^{\circ}\text{C}}{3.4\,\text{W}} = 17.6\,\text{K/W} \tag{3.104}$$

Davon muss der Wärmewiderstand des Gehäuses abgezogen werden, um den maximal erlaubten Wärmewiderstand des Kühlkörpers zu erhalten:

$$R_{\text{TH-KK}} = R_{\text{TH-ges}} - R_{\text{TH-JC}} = 17.6\,\text{K/W} - 4.5\,\text{K/W} = 13.1\,\text{K/W} \tag{3.105}$$

2m-Teil

Für Langzeitverlässlichkeit wird hier empfohlen, dass die Gehäusetemperatur folgenden Wert nicht übersteigt (vgl. *RA08H1317M RF Power Module Datasheet* 2006, Seite 7):

$$T_{\text{C-max}} = 90\,^{\circ}\text{C} \tag{3.106}$$

Somit ergibt sich für den maximalen Wärmewiderstand des Kühlkörpers:

$$R_{\text{TH-KK}} = \frac{T_{\text{C-max}}}{P_{\text{V-2m}}} = \frac{30\,^{\circ}\text{C}}{5\,\text{W}} = 6\,\text{K/W} \tag{3.107}$$

Kühlkörperauswahl

Sowohl die 80m- als auch die 2m-Endstufe benützen zwar den gleichen Kühlkörper, es werden allerdings nie beide gleichzeitig aktiv sein. Somit muss der Kühlkörper lediglich den geforderten Wärmewiderstand des 2m-Teils einhalten, der 80m-Teil ist, wie oben berechnet, weniger kritisch.

Die Wahl fällt auf einen Aluminium-Strangguss-Kühlkörper des Typs SK574. Dieser eignet sich gut für die mechanische Montage auf der Platine und besitzt bei

84 mm Länge einen thermischen Widerstand von ca. 4 K/W (vgl. *SK574 Extruded Heatsink Datasheet* 2016, Seite 3), was gegenüber den maximal erlaubten 6 K/W einen zusätzlichen Puffer bedeutet.

4. Richtkoppler-Platine

4.1. Übersicht

Zur Abstimmung von Antennen wird üblicherweise ein Stehwellenmessgerät benutzt, welches zwischen Sender und Antenne bzw. Antennentuner eingeschleift wird. Da dies beim Auslegen der Füchse im Gelände mitunter umständlich ist, soll eine Messmöglichkeit für das Stehwellenverhältnis im Fuchssender auf der Richtkoppler-Platine integriert werden.

Da die beiden zu überwachenden Frequenzen weit auseinanderliegen (ca. Faktor 40), wird ein Richtkoppler nur für 80m sowie ein separater nur für 2m benutzt. Diese geben eine jeweils zu vor- und rücklaufender Leistung proportionale Spannung aus. Daraus kann der Mikrocontroller das vorherrschende Stehwellenverhältnis berechnen. Ein Blockschaltbild der Richtkoppler-Platine ist in Abbildung 4.1 dargestellt.

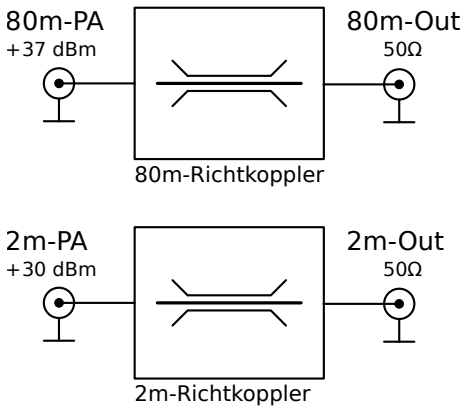


Abbildung 4.1.: Blockschaltbild Richtkoppler-Platine

Das HF-Signal wird mithilfe von zwei RG174-Koaxialkabeln von der DDS-HF-Platine zur Richtkoppler-Platine geführt. Die HF-Ausgänge der Richtkoppler-Platine können dann mit der (jeweils für das Band geeigneten) Antenne bzw. dem Antennentuner verbunden werden.

4.2. 80m-Richtkoppler

Die grundsätzliche Idee eines Richtkopplers ist, dass eine Spannung proportional zur Spannung des zu messenden Signals sowie ein Strom proportional zum Strom des zu messenden Signals ausgekoppelt wird (kapazitive und induktive Kopplung). Addiert man diese beiden ausgekoppelten Signale in zwei Lastwiderständen einmal phasenrichtig und einmal um 180° phasenverschoben, so überlagern sich diese einmal konstruktiv und einmal destruktiv. Besteht im zu messenden Signal das richtige Verhältnis zwischen Strom und Spannung (bei korrektem Abschluss mit z.B. 50 Ω) so löschen sich Strom- und Spannungsanteil im entsprechend dimensionierten Lastwiderstand gänzlich aus, es wird keine rücklaufende Leistung angezeigt.

Im hier verwenden Richtkoppler (siehe Abbildung 4.2) wird der proportionale Strom in zwei Lastwiderständen ($R304$ und $R306$) umgesetzt in eine Spannung, welche zur proportionalen Spannung addiert wird. Diese Spannung wird dann gleichgerichtet und über einen mehrstufigen Tiefpass geglättet, um sie vom Mikrocontroller messen zu können.

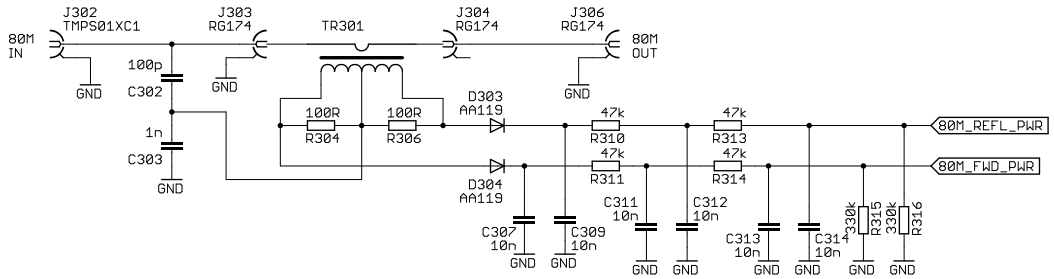


Abbildung 4.2.: Schaltplan 80m-Richtkoppler

Der Spannungsanteil wird beim 80m-Richtkoppler mit einem kapazitiven Spannungsteiler erzeugt ($C302$ und $C303$). Als Kopplungsverhältnis wird 1:11 ausgewählt, da sich dies als Kapazitätsverhältnis 1:10 realisieren lässt und bei 5 W in 50 Ω eine Spitzen-Spannung von

$$\hat{U}_{\text{U-K}} = \frac{\sqrt{5\,\text{W} \cdot 50\,\Omega} \cdot \sqrt{2}}{11} = 2.03\,\text{V} \tag{4.1}$$

ergibt. Dazu kommt (im Falle korrekten Abschlusses) dieselbe Spitzen-Spannung aus der Stromkopplung ($\hat{U}_{\text{I-K}}$), sodass insgesamt sich im Vorwärts-Ausgang die

doppelte Spannung ergibt (Addition), während sich diese im Rückwärts-Ausgang genau auslöschen.

$$U_{\mathrm{FWD}} \approx \hat{U}_{\mathrm{U\text{-}K}} + \hat{U}_{\mathrm{I\text{-}K}} \approx 4\,\mathrm{V} \tag{4.2}$$

Dies ist etwas größer als der Messbereich des Mikrocontroller-ADCs (0 V bis 3.3 V), deshalb wird der Spannungswert mit einem hochohmigen Spannungsteiler später auf einen passenden Wert heruntergeteilt.

Mit $C302 = 100\,\mathrm{pF}$ und $C303 = 1\,\mathrm{nF}$ ergibt sich für den gesamten kapazitiven Blindwiderstand bei Betriebsfrequenz, welchen der Richtkoppler (unerwünscht) verursacht:

$$X_{\mathrm{C}} = \frac{1}{2\pi f \cdot C302||C303} \approx 493\,\Omega \tag{4.3}$$

Dies wird auf einer 50 Ω-Leitung als vernachlässigbar angesehen.

Die Stromkopplung wird über TR301 realisiert. Dieser besitzt nur eine Primärwindung und ist auf Basis des Ringkernes RIK10 aufgebaut (vgl. *RIK 10 Ferroxcube Core Datasheet* 2013). Das Windungsverhältnis wird mit $\ddot{U} = 22 : 1$ gewählt, da sich diese erstmalig durch zwei teilen lassen. Bei 11 Windungen wäre eine exakte Mittelanzapfung umständlicher zu realisieren.

Da der Stromkopplungsfaktor (1/22) nun halb so groß ist wie der Spannungskopplungsfaktor (1/11), muss der Lastwiderstand entsprechend doppelt so groß sein wie der Leitungswiderstand der Sendeleitung, 50 Ω. Daraus ergibt sich:

$$R304 = 100\,\Omega \qquad R306 = 100\,\Omega \tag{4.4}$$

Bei der Auswahl des Kernes ist es wichtig, dass dieser primärseitig genügend Induktivität und somit einen ausreichend großen Blindwiderstand besitzt, um gegenüber dem transformierten Lastwiderstand nicht ins Gewicht zu fallen. Der verwendete RIK10 besitzt einen magnetischen Leitwert von ca. $A_{\mathrm{L}} = 1410\,\mathrm{nH}$ (siehe auch 3.4.4 auf Seite 53), was bei einer primärseitigen Windung zu einem induktiven Blindwiderstand von

$$X_{\mathrm{L}} = 2\pi \cdot f \cdot A_{\mathrm{L}} = 2\pi \cdot 3.55\,\mathrm{MHz} \cdot 1410\,\mathrm{nH} \approx 31.4\,\Omega \tag{4.5}$$

führt. Der Lastwiderstand transformiert sich entsprechend dem Windungsverhältnis zum Quadrat in einen Serienwiderstand $R_{\mathrm{L}'}$, welcher in der Sendeleitung aufscheint:

$$R'_{\mathrm{L}} = \frac{R304 + R306}{N^2} = \frac{200\,\Omega}{22^2} \approx 0.41\,\Omega \tag{4.6}$$

Dies ist wesentlich kleiner als X_L, sodass letzteres vernachlässigt werden kann (Parallelschaltung).

Zur Gleichrichtung des Signals werden Germaniumdioden benutzt, um die Verfälschung durch die Flussspannung der Dioden auf ein Minimum zu reduzieren. $C307$ und $C309$ stellen die Glättungskondensatoren dar. Nachfolgender mehrstufiger RC-Tiefpass wird empirisch dimensioniert, um eine möglichst HF-freie Ausgangsspannung zu erhalten. Mithilfe der Widerstände wird auch gleichzeitig eine Spannungsteilung durchgeführt. Das Teilerverhältnis wird ebenfalls empirisch bei nominaler Ausgangsleistung von 5 W ermittelt, sodass die Ausgangsspannung den Messbereich des Mikrocontrollers (0 V . . . 3.3 V) nicht überschreitet.

Besondere Bedeutung kommt $R317$ zu: Da sowohl der kapazitive Spannungsteiler als auch der Stromwandler galvanisch getrennt sind, könnte ohne diesen Widerstand kein Gleichstrom zum Messen entnommen werden. $R317$ hält sozusagen den Mittelpunkt des kapazitiven Spannungsteilers auf Masse. Ein Wert von 1 kΩ bietet sich hier an, da dies einerseits wesentlich kleiner als der Belastungswiderstand durch die mehrstufigen Tiefpässe (insgesamt ≈ 212 kΩ) ist, andererseits aber wesentlich größer als der Innenwiderstand des kapazitiven Spannungsteilers (≈ 40 Ω).

4.3. 2m-Richtkoppler

Im Frequenzbereich des 2m-Bandes werden häufig sogenannte PCB-Koppler verwendet. Dabei wird eine Messleiterbahn neben der Leistung führenden Leiterbahn (Sendeleitung) benutzt. Durch den geringen Abstand ergibt sich sowohl eine kapazitive als auch eine induktive Kopplung zwischen diesen. In einem Lastwiderstand wird wiederum Spannungs- und Stromanteil so überlagert, dass sich daraus eine Spannung proportional zur vorlaufenden und rücklaufenden Leistung ergibt. Das Prinzip ist in Abbildung 4.3 dargestellt.

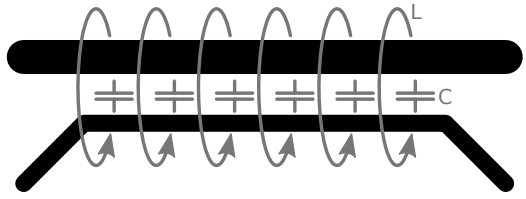


Abbildung 4.3.: Prinzip PCB-Koppler

Für vor- und rücklaufende Leistung werden zwei separate Leiterbahnen benutzt. Dadurch kann eine Seite ohne Abschlusswiderstand direkt an den Diodengleichrichter angeschlossen werden. Da eine analytische Berechnung des Richtkopplers mit hohem mathematischen Aufwand verbunden ist, wird bei der Dimensionierung empirisch vorgegangen. Die in Abbildung 4.4 dargestellten Abmessungen liefern im 2m-Band zufriedenstellende Ergebnisse.

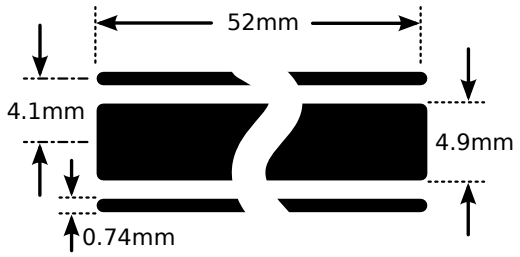


Abbildung 4.4.: Empirisch ermittelte Abmessungen

Um den Richtkoppler (siehe Abbildung 4.5 auf der nächsten Seite) auf 50 Ω abzugleichen, werden die Lastwiderstände so lange variiert, bis beim Senden in ein 50 Ω-Dummyload die Spannung am Rückwärts-Ausgang minimal ist. So werden $R301$ und $R307$ bestimmt:

$$R301 = 220\,\Omega \qquad R307 = 220\,\Omega \tag{4.7}$$

Wiederum werden Germaniumdioden mit nachfolgendem Glättungskondensator und mehrstufigem Tiefpass benutzt, um die ausgekoppelte HF-Spannung in eine vom Mikrocontroller messbare Gleichspannung umzuwandeln. Da der Kopplungsfaktor des PCB-Kopplers nicht näher bekannt ist, erfolgt die Dimensionierung empirisch, sodass bei einer Nennleistung von 1 W die Ausgangsspannung in den Messbereich des ADCs fällt.

4.4. Messungen

80m-Richtkoppler

Bei nominaler Ausgangsleistung von $P_{\mathrm{OUT}} = 5\,\mathrm{W}$ in ein 50 Ω-Dummyload liefert der Richtkoppler folgende Gleichspannungswerte:

$$\begin{aligned} U_{\mathrm{FWD}} &\approx 2.55\,\mathrm{V} \\ U_{\mathrm{REV}} &\approx 0.076\,\mathrm{V} \end{aligned} \tag{4.8}$$

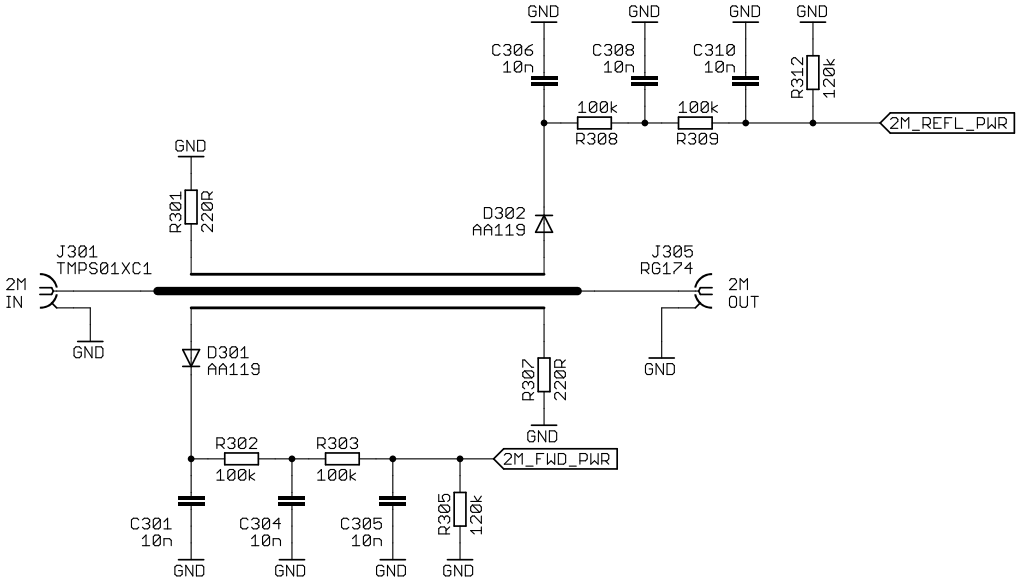


Abbildung 4.5.: Schaltplan 2m-Richtkoppler

Dies ergibt eine Richtschärfe von

$$\mathrm{RS} = 20\log\left(\frac{U_{\mathrm{FWD}}}{U_{\mathrm{REV}}}\right) \approx 30.5\,\mathrm{dB} \tag{4.9}$$

2m-Richtkoppler

Der PCB-Koppler liefert bei Nennleistung $P_{\mathrm{OUT}} = 1\,\mathrm{W}$ in einen $50\,\Omega$-Abschlusswiderstand am Ausgang des Tiefpasses folgende Gleichspannung:

$$\begin{aligned} U_{\mathrm{FWD}} &\approx 1.94\,\mathrm{V} \\ U_{\mathrm{REV}} &\approx 0.1\,\mathrm{V} \end{aligned} \tag{4.10}$$

Die Richtschärfe beträgt demnach

$$\mathrm{RS} = 20\log\left(\frac{U_{\mathrm{FWD}}}{U_{\mathrm{REV}}}\right) \approx 25.7\,\mathrm{dB} \tag{4.11}$$

5. RFID-Platine

5.1. Übersicht

Die RFID-Platine ist jene Platine, die für Benutzer (Teilnehmer der Fuchsjagd und Ausleger der Füchse) zugänglich sein muss. Auf ihr befindet sich erstens das RFID-Modul zur Teilnehmerregistrierung mit zusätzlicher Status-LED, welche die erfolgreiche Registrierung signalisiert. Dazu kommt eine LED-Anzeige für das Stehwellenverhältnis auf der Sendeleitung mit Test-Sende-Tastern (Dauerträger) zur Abstimmung der Antenne. Abbildung 5.1 zeigt das Blockschaltbild der RFID-Platine.

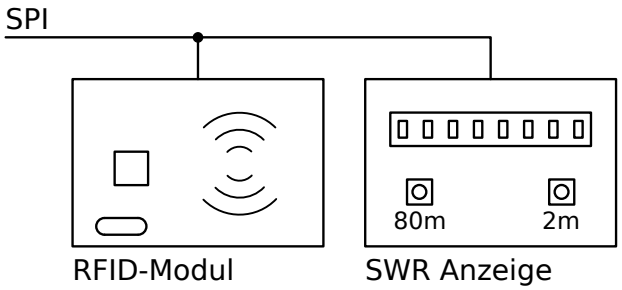


Abbildung 5.1.: Blockschaltbild RFID-Platine

Die RFID-Platine ist über eine separate Flachbandkabelverbindung (16-polig) direkt mit der Mikrocontroller-Platine verbunden. Auf diese Weise kann die Platine auch vom restlichen Fuchssender entfernt angebracht werden, sollte dies aus gehäusetechnischen Gründen (Zugänglichkeit des RFID-Moduls) notwendig sein.

5.2. RFID-Modul

Grundsätzlich sollen zur Registrierung der Teilnehmer Mifare Classic Tags benutzt werden, da diese weit verbreitet sind und sozusagen als Standard-Lösung

angesehen werden können. Zur Kommunikation mit den RFID-Tags der Teilnehmer wird ein fertiges RFID-Modul benutzt, da aufgrund der leichten Verfügbarkeit und des geringen Preises dieser Module der Aufwand einer Eigenentwicklung nicht gerechtfertigt erscheint.

Im Speziellen fällt die Entscheidung auf das „Neuftech Mifare RC522 RFID Module“, welches unter anderem auf Amazon erhältlich ist (Stand 3.3.16 *Neuftech Mifare RC522 RFID Module für Arduino* 2016). Dieses Modul ist praktisch ein Breakout-Board für den MFRC522 Chip von NXP, welcher sich mit 3.3 V betreiben und über eine SPI-Verbindung ansprechen lässt. Besonders vorteilhaft ist, dass zu diesem Baustein bereits eine Arduino-Library existiert, was die Ansteuerung erheblich erleichtert (vgl. *Arduino Mifare MFRC522 RFID Reader/Writer* 2016). Sollte das Modul in Zukunft nicht mehr erhältlich sein, so könnte der MFRC522 eventuell auch direkt auf der RFID-Platine implementiert werden.

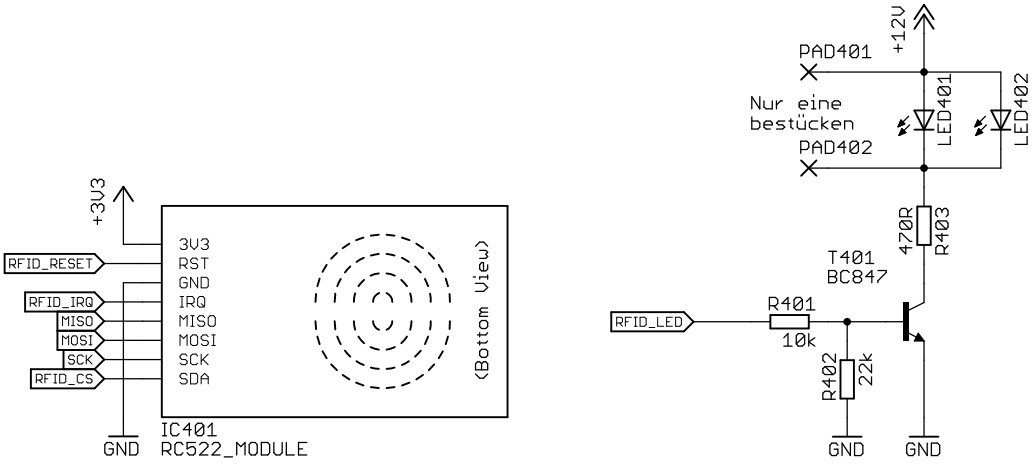


Abbildung 5.2.: Beschaltung RFID-Modul und RFID-LED

Das RFID-Modul muss lediglich durch entsprechende Datenleitungen mit dem Mikrocontroller verbunden werden, eine weitere Beschaltung ist nicht notwendig (siehe Abbildung 5.2).

Die Signalisierung eines erfolgreichen Lese-/Schreibvorgangs erfolgt nur optisch durch eine LED. Ein Summer würde unter Umständen aus größerer Entfernung hörbar sein und so die Position des Fuchses verraten. Um im Bezug auf den Betriebsstrom der LED unabhängig zu sein (um diese auch bei Sonneneinstrahlung erkennen zu können), wird diese nicht direkt am Mikrocontroller, sondern über einen Transistor von 12 V aus betrieben.

Mit einem Vorwiderstand $R403 = 470\,\Omega$ beträgt der LED-Strom ca. 20 mA, zum Betrieb einer superhellen 5 mm-LED. Soll eine noch hellere LED eingesetzt werden, muss ggf. der Wert des Vorwiderstandes angepasst werden, bis zu 100 mA sind vom Transistor aus möglich.

Im Schaltplan sind zwei LEDs mit verschiedenen Footprints eingezeichnet (SMD und Durchsteckmontage), wovon jedoch nur eine bestückt wird. Zusätzlich sind zwei Pads zum Anschluss eines Verlängerungskabels vorhanden, sollte die LED abseits der Platine im Gehäuse montiert werden.

5.3. Abstimmeinrichtung

Das durch den Richtkoppler gemessene Stehwellenverhältnis wird mithilfe einer Zeile aus 8 LEDs angezeigt, welche durch ein Schieberegister über den SPI-Bus angesteuert wird, siehe Abbildung 5.3. Zusätzlich sind zwei Taster zur Aussendung eines reinen Trägers auf 80m und auf 2m vorhanden, um die Abstimmung durchzuführen.

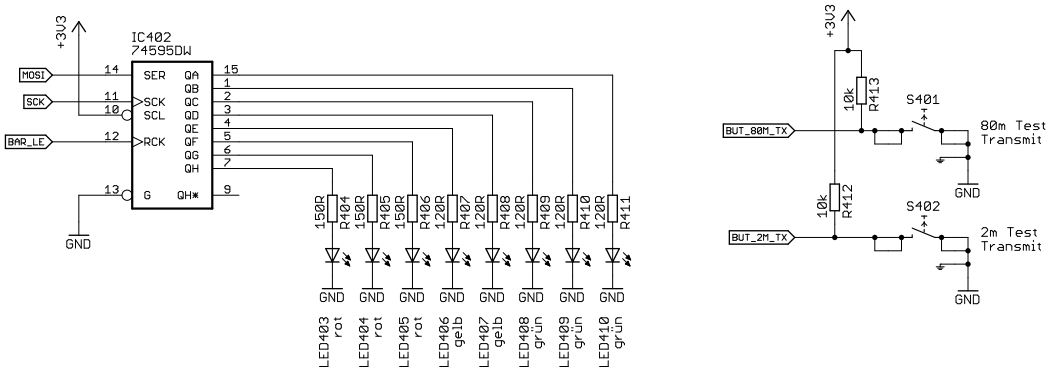


Abbildung 5.3.: Schaltplan LED-Anzeige und Test-Sende-Taster

Der Baustein 74HC595 verfügt über ein 8-Bit Schieberegister, welches direkt mit dem SPI-Bus verbunden ist (`MOSI` und `SCK`). Alle über SPI gesendeten Bytes passieren somit das Schieberegister. Mithilfe der `BAR_LE`-Leitung (`RCK`-Eingang am Schieberegister) kann das aktuell im Schieberegister befindliche Datenwort auf den Latch und somit auf die Ausgänge übertragen werden. Soll also ein neues Byte auf der LED-Zeile angezeigt werden, so wird dieses Byte über SPI gesendet und anschließend eine steigende Flanke auf `BAR_LE` auf die LEDs übertragen. (vgl. *74HC595 8-bit Shift Register Datasheet* 2004)

Für die Berechnung der Vorwiderstände wird für gelbe und grüne LEDs von einer Flussspannung $U_\mathrm{F} = 2.2\,\mathrm{V}$ ausgegangen, für die roten LEDs von $U_\mathrm{F} = 1.8\,\mathrm{V}$ (vgl. *HSMx-A10x-xxxxx SM LED Indicator Datasheet* 2015, Seite 6). Soll der LED-Strom $I_\mathrm{F} = 10\,\mathrm{mA}$ betragen, so ergibt sich bei einer Betriebsspannung von $U_\mathrm{B} = 3.3\,\mathrm{V}$ für die beiden Vorwiderstände:

$$\begin{aligned} R_\mathrm{Ye,Gr} &= \frac{U_\mathrm{B} - U_\mathrm{F}}{I_\mathrm{F}} = 110\,\Omega \approx 120\,\Omega \\ R_\mathrm{Re} &= 150\,\Omega \end{aligned} \tag{5.1}$$

6. Weiterführende Überlegungen

6.1. Gehäuse

Der Entwurf eines passenden Gehäuses für den Fuchssender ist laut Aufgabenstellung nicht Teil dieser Arbeit, die Anforderungen daran werden hier jedoch trotzdem kurz festgehalten.

Wasserschutz: Da die Füchse oft bereits am Vortag der Veranstaltung ausgelegt werden, besteht potentiell die Möglichkeit, dass bis zum Start der Fuchsjagd Niederschlag fällt. Aus diesem Grunde sollte das Gehäuse den Sender gegen Regen schützen.

Abschirmung: Um die Peilempfänger der Teilnehmer im Nahbereich um den Sender nicht zu übersteuern, sowie die EMV-Emissionen zu begrenzen, sollte das Gehäuse einen geschlossenen Käfig um den eigentlichen Sender beinhalten.

Zugang zum RFID-Modul: Damit sich die Teilnehmer während der Fuchsjagd am RFID-Modul registrieren können, muss dieses frei zugänglich, d.h. nicht wie der Rest des Senders abgeschirmt sein. Weiters muss die RFID-Status-LED von außen sichtbar sein. Diese beiden Punkte stellen im Bezug auf den Wasserschutz eine gewisse Herausforderung dar.

Manipulationsschutz: Um die Manipulation bzw. die Störung des Sendebetriebs durch Passanten oder auch Teilnehmer der Fuchsjagd zu unterbinden, sollte das Gehäuse per Schlüssel verschließbar bzw. zumindest nicht ohne Werkzeuge zu öffnen sein.

Wärmeabfuhr: Um einen Hitzestau während des Sendebetriebs zu verhindern, sollte das Gehäuse die im Inneren erzeugte Abwärme dauerhaft an die Umgebungsluft abführen können. Dies umfasst mitunter einen außerhalb des Gehäuses angebrachten Kühlkörper.

Kompaktheit: Zur Vorbereitung der Fuchsjagd muss mitunter eine Person alle fünf Füchse mit sich tragen, deshalb sollte das Gehäuse so kompakt und leicht wie möglich sein.

6.2. Akkukriterien

Die Auswahl eines Akkus zur Energieversorgung des Fuchssenders ist nach Absprache mit dem Auftraggeber nicht Teil der Arbeit. Deshalb werden hier nur die grundsätzlichen Anforderungen an den Akku behandelt. Primär muss dieser eine Spannung von ca. 12 V besitzen, um damit den Fuchs betreiben zu können.

6.2.1. Energieverbrauch

Zur Abschätzung der benötigten Akkukapazität müssen die verschiedenen Betriebsphasen betrachtet werden:

Standby: Zwischen dem Auslegen der Füchse (und somit dem Verbinden der Stromversorgung) und dem Beginn der Fuchsjagd müssen die Sender in einer Art Bereitschaftsmodus (Standby) verweilen. Dieser Zeitraum kann laut Aufgabenstellung bis zu einem Tag betragen. Für diesen Zeitraum wird großzügig ein Stromverbrauch von $I_{\mathrm{SB}} = 5\,\mathrm{mA}$ angenommen, welcher sich aus den verschiedenen Standby-Strömen der einzelnen Bauteile (hauptsächlich Schaltregler) zusammensetzt. Dabei wird folgende Akkukapazität verbraucht:

$$Q_{\mathrm{SB}} = I_{\mathrm{SB}} \cdot 24\,\mathrm{h} = 0.12\,\mathrm{Ah} \tag{6.1}$$

Sendeenergie: Der größte Teil der Energie wird für die Erzeugung des Sendesignals (80m oder 2m) benötigt. Die genaue Berechnung ist abhängig von der Dauer der Fuchsjagd, der Anzahl der Füchse (u.U. auch weniger als 5) und des An-Aus-Verhältnisses des Morsesignals (ca. $M \approx 0.6$). In den offiziellen IARU Rules wird gefordert, dass die Sender 8 Stunden betriebsfähig sein müssen (vgl. *Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1* 2015, Seite 12). Der Stromverbrauch der 80m-Endstufe beträgt $I_{\mathrm{TX}} = 0.7\,\mathrm{A}$ bei Dauerträger, mehr als die 2m-Endstufe verbraucht, deshalb wird hier nur der 80m-Teil betrachtet. Im ungünstigsten Fall mit nur zwei aktiven Füchsen ergibt dies folgende Akkukapazität:

$$Q_{\mathrm{TX}} = I_{\mathrm{TX}} \cdot 8\,\mathrm{h} \cdot M \cdot \frac{1}{2} \approx 1.68\,\mathrm{Ah} \tag{6.2}$$

Mit fünf aktiven Füchsen wird nur folgende Kapazität benötigt:

$$Q_{\mathrm{TX}} = I_{\mathrm{TX}} \cdot 8\,\mathrm{h} \cdot M \cdot \frac{1}{5} \approx 0.67\,\mathrm{Ah} \tag{6.3}$$

Betriebsstromverbrauch : Während der Fuchsjagd benötigt der Fuchs zum Betrieb, abgesehen von der Sendeenergie, ca. $I_{\mathrm{BE}} \approx 70\,\mathrm{mA}$ für Mikrocontroller, RFID-Modul, DDS, usw.

$$Q_{\mathrm{BE}} = I_{\mathrm{BE}} \cdot 8\,\mathrm{h} \approx 0.56\,\mathrm{Ah} \tag{6.4}$$

Insgesamt bedeutet dies bei einer typischen Fuchsjagd eine Mindest-Akkukapazität von

$$Q_{\mathrm{ges}} = Q_{\mathrm{SB}} + Q_{\mathrm{TX}} + Q_{\mathrm{BE}} \approx 1.35\,\mathrm{Ah} \tag{6.5}$$

Im ungünstigsten Fall mit nur zwei aktiven Füchsen steigt der Bedarf auf $Q_{\mathrm{ges}} \approx 2.36\,\mathrm{Ah}$. Allerdings handelt es sich hierbei lediglich um eine theoretische Abschätzung. Die tatsächliche Wahl der Akkukapazität sollte auf jeden Fall einen entsprechenden Puffer aufweisen.

6.2.2. Akkutypen

Zum Betrieb des Fuchssenders können folgende Akkutypen benutzt werden:

Bleigel-Akkus: Diese stellen die einfachste Variante der Spannungsversorgung dar, allerdings auch die gewichtsintensivste. Es bietet sich die Standardgröße mit 12 V (bis 13.8 V und 3.4 Ah an (ca. 1.2 kg). Eventuell, wenn beabsichtigt wird, zwei Fuchsjagden hintereinander auszurichten, auch die größere Version mit 7.2 Ah (ca. 2.5 kg). Vorteilhaft ist hier der einfache Ladevorgang, es können alle Akkus parallel (gleichzeitig) an ein entsprechendes Netzteil mit der Ladeschlussspannung 13.8 V und Strombegrenzung angeschlossen werden.

LiPo-Akkus: Drei LiPo-Zellen besitzen zwar nur eine Gesamtspannung von 11.1 V, allerdings ist auch damit ein problemloser Betrieb möglich (ggf. mit etwas geringerer Ausgangsleistung). Sie bringen gegenüber Bleigelakkus eine erhebliche Gewichtsreduktion (z.B. ca. 270 g bei 3 Ah). Allerdings ist der Ladevorgang aufwendiger (spezielles Balancer-Ladegerät notwendig).

NiMH-Akkus: Solche 12 V-Akkupacks aus dem RC-Bereich (Modellautos, -flugzeuge) sind günstig zu erwerben und bringen gegenüber Bleigelakkus ebenfalls eine große Gewichtsreduktion (z.B. ca. 300 g bei 2.9 Ah). Die Tatsache, dass NiMH-Akkus durch Tiefentladung nicht beschädigt werden, kann ebenfalls als Pluspunkt angesehen werden.

Die Entscheidung, welcher Akkutyp benutzt werden soll, kann individuell je nach Einsatzzweck, -häufigkeit und -ort getroffen werden.

6.3. Probleme und Verbesserungsmöglichkeiten

Im Zuge der Entwicklung des Fuchssenders traten an manchen Stellen Probleme auf, sodass teils noch Möglichkeit zur Verbesserung besteht, sollte das Projekt weitergeführt und überarbeitet werden.

6.3.1. Nebenaussendungen durch Schaltregler

Der 3.3 V-Schaltregler besitzt eine Betriebsfrequenz von ca. 260 kHz, welche über verschiedene Wege in das HF-Ausgangssignal gelangt. Bereits im Signal des DDS-Bausteins sind Mischprodukte dieser Schaltfrequenz enthalten (260 kHz über und unter dem Träger, sowie Vielfache davon, siehe Abbildung 6.1).

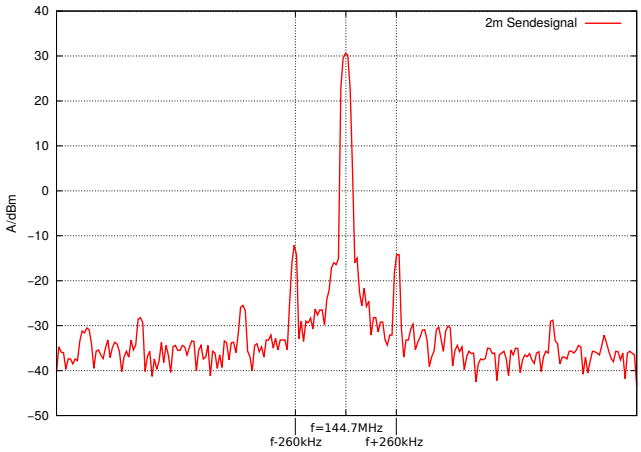


Abbildung 6.1.: Störungen durch den Schaltregler im Ausgangssignal

Eine große Verbesserung war hier die Verwendung von magnetisch geschlossenen Spulen (in den Filter-Tiefpässen und als Schaltregler-Drossel), sowie weiteren größeren Glättungskondensatoren (Tantal) auf den Versorgungsschienen.

Die Nebenaussendungen liegen mit ca. −13 dBm dennoch über dem gesetzlichen Grenzwert (in diesem Fall −30 dBm). Dementsprechend müssen diese Störungen behoben werden, um den Fuchs konform mit den Gesetzen betreiben zu können. Im Platinenlayout sind vermutlich noch Verbesserungen möglich, um die Störungen durch den Schaltregler weiter zu verringern.

6.3.2. DDS Fehlfunktionen

Der auf dem Prototyp verbaute DDS-Baustein (gemeint ist dieser „physikalische“ Chip) neigt bei längerem Betrieb und stärkerer Erwärmung dazu, dass neben der eigentlichen (programmierten) Frequenz eine Reihe von (unerwünschten) Spektralanteilen auftreten, die einen korrekten Sendebetrieb unmöglich machen. Jedoch lässt sich der Baustein beispielsweise mit einem Alu-Kühlkörper soweit kühlen, dass die Störungen verschwinden.

Dieses Verhalten steht in starkem Widerspruch mit dem Datenblatt, wo als Arbeitstemperaturbereich bis 105 °C angegeben ist, welcher bei weitem nicht überschritten wird (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 5). Darüber hinaus wird in verschiedenen Designs im Internet, welche diesen Baustein beinhalten, nirgends ein Kühlkörper verwendet. Bemerkenswert ist, dass die Fehlfunktionen auf einem Demo-Board, welches zu Beginn des Projektes zur Softwareentwicklung benutzt wurde, nicht auftreten. Auf dem Gesamt-Prototyp tritt das Problem hingegen auch nach Austauschen des DDS-Chips noch auf.

Als mögliche Ursache könnte einerseits die Beschädigung des (empfindlichen) Bausteins durch das manuelle Löten angesehen werden. Dagegen spricht hingegen, dass diese Beschädigung am Demo-Board offensichtlich nicht, am Prototyp dafür gleich zwei Mal hintereinander geschehen sein müsste, obwohl beim Löten dreimal praktisch gleich vorgegangen wurde. Eine andere Ursache könnten unzureichende Masseverbindungen sein. Bemerkenswert dabei ist allerdings, dass die DDS-Störungen vollkommen unabhängig davon auftreten, ob die HF-Leistungsverstärker aktiv sind oder nicht, obwohl diese die Hauptverursacher von HF-Strömen auf der Massefläche darstellen.

Abhilfe sollte hier ebenfalls ein verbessertes Platinenlayout (Durchkontaktierungen etc.) sowie ein Reflow-Löten mit Lötpaste schaffen.

6.3.3. HF-Power-Modul für 2m

Das HF-Power-Modul stellt zwar eine wesentliche Erleichterung für den Aufbau des 2m-Leistungsverstärkers dar, allerdings ist das RA08H1317M mitunter noch nicht die optimale Lösung. Die Nennleistung des Moduls von 8 W wird hier bei weitem nicht ausgereizt, weshalb der Wirkungsgrad wesentlich schlechter ausfällt (ca. 16% statt der im Datenblatt angegebenen $> 40\%$). Die höhere Sendeleistung des 80m-Teils bewirkt zwar, dass dies bei der Dimensionierung der Akku-Kapazität nicht ins Gewicht fällt, dennoch ist es auf einem akkubetriebenen System sinnvoll, den (unnötigen) Energieverbrauch auf ein Minimum zu beschränken.

Ein geeigneteres HF-Power-Modul oder sogar doch ein vollständig diskreter Aufbau des 2m-Leistungsverstärkers sollte hier Verbesserungen bringen.

6.3.4. Abstimmvorgang des 2m-Teils

Insgesamt sind zur Abstimmung des 2m-Teils vier Trimmkondensatoren vorhanden (zwei im Eingangsfilter, einer im Vorverstärker und einer im Oberwellenfilter). Sind bei der ersten Inbetriebnahme alle verstellt, so kann die Signalamplitude u.U. nur mehr mit einem (empfindlichen) Spektrumanalysator gemessen werden, um die Abstimmung durchzuführen. Interessant wäre hier die Verminderung der benötigten Trimmkondensatoren, um die Abstimmung mit einem einfachen Leistungsmessgerät (z.B. SWR-Messbrücke) und etwas „Probieren“ durchführen zu können.

Dieses Ziel konnte jedoch nicht umgesetzt werden, da für die Unterdrückung der Nebenprodukte des DDS mehrere Filterstufen mit entsprechend hoher Güte benötigt werden. Durch die Schmalbandigkeit wird eine Abstimmmöglichkeit notwendig, da Bauteiltoleranzen einen „fixen“ Aufbau nicht mehr zulassen.

6.3.5. Oberwellenunterdrückung auf 2m

Wie im Abschnitt 3.5.7 auf Seite 78 ersichtlich, konnten am Prototypen die gesetzlich geforderten -60 dBc nicht gänzlich eingehalten werden. Die zweite Oberwelle ist von geringerer Bedeutung, da die Sendeantenne dort nicht resonant ist und somit zusätzlich dämpft. Jedoch liegt auch die dritte Oberwelle um ca. 5 dB über dem gesetzlichen Grenzwert, welche von der Antenne nicht gedämpft wird.

Eventuell könnte die Verwendung eines dickeren (= verlustärmeren) Silberdrahtes zum Wickeln der Induktivitäten im Oberwellenfilter bereits eine ausreichende

Verbesserung bringen. Ansonsten müsste eine Neudimensionierung des Oberwellenfilters mit höherer Güte erfolgen.

6.3.6. Platinenlayout und Abschirmung

Die Tatsache, dass Spannungsversorgung, Digitalelektronik, Signalerzeugung, Leistungsverstärker und Richtkoppler auf „offenen“ Platinen realisiert sind, welche zwar über ein entsprechendes Gehäuse nach außen, allerdings nicht gegeneinander abgeschirmt sind, bringt Probleme mit sich. Vorteilhaft ist sicherlich die Trennung in die einzelnen Platinen, jedoch würde eine zusätzliche Abschirmung einzelner Komponenten (bes. Schaltregler und HF-Leistungsverstärker) vermutlich weitere Verbesserungen bringen, besonders in Bezug auf die weiter oben genannten Störungen durch den Schaltregler.

Bei der hohen notwendigen Gesamtverstärkung ist die Schwingneigung immer auch ein kritisches Thema. So reicht beispielsweise die magnetische Kopplung der Spulen des Ausgangsfilters in das Massekabel eines Oszilloskop-Tastkopfes in Verbindung mit dessen Eingangskapazität aus, den 2m-Leistungsverstärker zum Schwingen zu bringen, wenn man die Spitze auf den Eingang des Vorverstärkers hält. Zwar funktioniert der 2m-Verstärker im Normalbetrieb ohne zu schwingen, eine zusätzliche Schirmung (bes. des Vorverstärkers) würde das Risiko jedoch weiter senken.

Auch in Bezug auf die Leiterbahnführung, die Aufteilung und Anordnung der Baugruppen, der Masseführung sowie die Entkopplung der Versorgungsspannungen sind vermutlich noch Verbesserungen möglich.

6.3.7. Störung der SPI-Übertragung zum DDS

Bei aktivierter Amplitudenmodulation im 2m-Band muss über den SPI-Bus 6000 Mal pro Sekude das Amplitudenregister neu geschrieben werden (10 Mal pro Periode bei 600 Hz Modulationsfrequenz). Dabei traten in einigen Versuchen bei aktivierter 2m-Endstufe nach einigen Sekunden bis Minuten Übertragungsfehler auf, welche häufig zum kompletten Aussetzen des DDS führten. Vermutlich sind diese Störungen auf HF-Einkopplungen vom 2m-Leistungsverstärker in die ISP-Programmierleitung oder in die SPI-Verbindung zur RFID-Platine zurückzuführen.

Aus diesem Grund sind in die SPI-Datenleitungen zur RFID-Platine und zum ISP-Stecker 100 Ω-Widerstände eingeschleift. Auf diese Weise konnte das Problem nach aktuellem Erkenntnisstand behoben werden.

Teil II.

Software

7. Auswahl Mikrocontroller

Der Mikrocontroller im Projekt muss die verschiedenen vorhandenen Komponenten wie den DDS-Baustein zur Signalgenerierung, das RFID-Modul zum Erkennen der Teilnehmerchipkarten usw. ansteuern und konfigurieren. Dazu muss der Controller verschiedene Schnittstellen und Bussysteme beherrschen:

- I^2C-Bus für Real-Time Clock und EEPROM
- SPI-Bus für RFID-Modul und DDS-Baustein
- UART-Schnittstelle für Kommunikation mit dem PC

Im Folgenden werden verschiedene für das Projekt interessante Mikrocontrollerarchitekturen kurz vorgestellt:

AVR-Mikrocontroller: Eine einfache Mikrocontrollerarchitektur, die sich in der Hobbybastlerszene großer Beliebtheit erfreut, ist die AVR-Architektur der Firma Atmel. Zur Architektur gehören die ATtiny-Serie mit eher kleineren und leistungsschwächeren Controllern und die ATmega-Serie mit größeren, leistungsfähigeren Prozessoren. Beide Serien bestehen aus 8-bit Mikrocontrollern. Auch die meisten Entwicklungsboards des beliebten Elektronik-Entwicklungssystems Arduino basieren auf ATmega-Mikrocontrollern.

Die Controller verfügen über eine hardwareseitige Unterstützung aller oben genannten Schnittstellen. Die Programmierung erfolgt entweder direkt in der Assemblersprache oder in höheren Programmiersprachen wie beispielsweise C oder Basic. Für die Programmiersprache C existiert der freie GCC-Compiler (GNU Compiler Collection), der auch Programme für die AVR-Architektur übersetzen kann.

PIC-Mikrocontroller: Die von der Firma Mikrochip angebotenen 8-bit PIC-Mikrocontroller bieten ähnliche Funktionalität wie die Mikrocontroller der Firma Atmel und zeichnen sich besonders durch einen sehr schlanken, schnell zu erlernenden Befehlssatz aus. Mikrochip bietet einen proprietären C-Compiler an, der unter bestimmten Voraussetzungen auch kostenlos genutzt werden kann. Eine Portierung des freien GCC-Compilers existiert derzeit nicht.

PSoC-Mikrocontroller: Die PSoC-Mikrocontroller der Firma Cypress basieren auf einem ARM-Kern. Es handelt sich um eine Familie von 32-bit Mikrocontrollern. Die Controller können flexibel konfiguriert werden. So können beispielsweise über die mitgelieferte Entwicklungsumgebung Module für die benötigten Schnittstellen flexibel konfiguriert werden. Die Programmierung erfolgt ebenfalls in der Programmiersprache C. Die Entwicklungsumgebung greift dazu auf die ARM-Version des GCC-Compilers zurück.

Auswahl ATmega 644: Die Wahl bei diesem Projekt fällt auf den ATmega 644-Mikrocontroller der Firma Atmel. Der Controller verfügt über einen Programmspeicher mit einer Größe von 64 kB, einem Arbeitsspeicher mit 4 kB und einem internen EEPROM mit 2 kB Speicherplatz.

Ausschlaggebend für die Wahl war die Verfügbarkeit des freien C-Compilers gcc, die Vorkenntnisse der Autoren im Bereich der AVR-Programmierung und die Verfügbarkeit einer Bibliothek für das verwendete RFID-Modul.

7.1. Build-Umgebung

Das Programm für den Mikrocontroller muss in den Flashspeicher geschrieben werden, der als Programmspeicher dient. Eine Möglichkeit zur Übertragung des Programms ist die Verwendung des sogenannten ISP-Interfaces (In System Programming), das im Idealfall die Programmierung des in die fertige Schaltung eingebauten Controllers ermöglicht.

Die Firma Atmel bietet Entwicklungskits an, mit deren Hilfe der Mikrocontroller vom PC aus über die ISP-Schnittstelle programmiert werden kann. Diese Kits werden gewöhnlich über die USB-Schnittstelle mit dem PC verbunden. Als Alternative zu den relativ teuren offiziellen Kits vom Hersteller bieten diverse andere Firmen eigene, billigere Programmiergeräte an.

Alle gängigen Programmiergeräte liefern entweder eine eigene Software mit oder bieten eine Schnittstelle, die zu bereits erhältlicher Software kompatibel ist, um das im Intel-Hex-Format vorliegende Programm (*Hex-File*) auf den Controller zu übertragen.

Als Programmiersprache wird in diesem Projekt die Sprache C verwendet. Die Programmentwicklung in C ist im Vergleich zur Entwicklung in der Assemblersprache schneller und einfacher möglich. Trotzdem bietet die Sprache C die Möglichkeit eines sehr hardwarenahen Zugriffs auf die verfügbare Peripherie des Controllers durch Zugriffsmöglichkeit auf die einzelnen Register.

Ziel der Build-Werkzeuge ist es somit, den C-Programmcode in Maschinencode zu übersetzen und diesen in das Intel-Hex-Format überzuführen.

Zur Bewältigung dieser Aufgabe kann unter Windows entweder auf die offizielle Entwicklungsumgebung Atmel-Studio zurückgegriffen werden (vgl. *Atmel Studio 7* 2016) oder es wird das WinAVR-Softwarepaket verwendet, das eine wesentlich schlankere Installation bietet und einen Editor, den GCC-Compiler und alle weiteren benötigten Programme zur Erstellung des Hex-Files aus dem C-Programmcode mitliefert (vgl. *Homepage WinAVR* 2016). Das Programm kann von der Website direkt als Installer-Datei heruntergeladen werden.

WinAVR liefert das Programm *Programmer's Notepad* mit, das bereits alle benötigten Makros zur Übersetzung des Programms besitzt. Es reicht, eine beliebige Datei aus dem Ordner der Mikrocontrollersoftware mit Programmer's Notepad zu öffnen und in der Menüleiste unter *Tools* die Auswahl *Make all* anzuklicken. Im Ordner wird eine Datei namens main.hex erzeugt, die nun beispielsweise mithilfe der Software des Programmieradapters zum Mikrocontroller übertragen werden kann.

Unter Linux existieren bei den meisten Distributionen vorgefertigte Pakete unter dem Namen *avr-gcc* bzw. *gcc-avr*, welche die benötigte Software beinhalten. Sollte das Paket *avr-libc* nicht automatisch mitinstalliert werden, muss es manuell installiert werden. Zur Übersetzung des Projekts kann in der Kommandozeile in den Ordner mit den Quelldateien gewechselt und dort der Befehl *make all* ausgeführt werden. Der Befehl *make clean* löscht alle erzeugten Dateien, sodass nur mehr die Quelldateien übrigbleiben und *make program* überträgt das Programm zum Mikrocontroller. Zur Übertragung ruft *make* die Software *avrdude* auf, die eine Vielzahl verschiedener Programmiergeräte unterstützt. Um das verwendete Programmiergerät einzustellen, müssen in der Datei *Makefile* im Ordner mit den Quelldateien die Optionen `AVRDUDE_PROGRAMMER` und `AVRDUDE_PORT` abgeändert werden. Die Dokumentation des Programms avrdude liefert weitere Informationen über die unterstützten Programmer (vgl. *AVRDUDE Documentation* 2016).

Achtung Der Mikrocontroller im Projekt wird mit einer Betriebsspannung von 3 V anstatt mit 5 V versorgt. Viele Programmer gehen jedoch davon aus, dass der Mikrocontroller mit 5 V betrieben wird. Vor dem Programmiervorgang sollte der Programmer auf 3 V-Betrieb umgestellt werden. Die genaue Vorgehensweise dazu muss der jeweiligen Dokumentation des Programmiergeräts entnommen werden.

Fusebits Einige Grundfunktionen des Mikrocontrollers können über die sogenannten Fusebits eingestellt werden. Diese Bits werden, anders als die Konfigu-

rationsbits in gewöhnlichen Registern, nicht durch die Software am Controller, sondern direkt mithilfe des Programmieradapters eingestellt (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 285 ff.). Für dieses Projekt müssen die Fusebits zur Einstellung der Taktquelle (CKSEL3..0) auf den Wert 0b1101 gestellt werden. Diese Einstellung steht für einen *Low power crystal oscillator* mit einer Frequenz zwischen 3 und 8 MHz. Außerdem ist es wichtig, das JTAG-Interface mithilfe des Bits JTAGEN (→ auf 0 setzen) zu deaktivieren, da die entsprechenden Pins des JTAG-Interfaces, das zum Debuggen verwendet werden kann, für andere Aufgaben benötigt werden. Möchte man die Mikrocontrollersoftware testen, bietet es sich an, das Bit EESAVE auf 0 zu setzen. Dadurch wird der Inhalt des EEPROMs nicht bei jedem Programmiervorgang gelöscht. So gehen die Einstellungen der Software nicht jedes Mal verloren, wenn ein Fehler in der Software ausgebessert wird.

Zur Programmierung der Fusebits kann meist die entsprechende Software des Programmieradapters verwendet werden. Das Atmel-Studio und avrdude bieten die Funktionalität ebenfalls. Für avrdude bietet sich als komfortableres Frontend beispielsweise das Programm *avr8-burn-o-mat* an, das für alle gängigen Betriebssysteme auf der Projektwebsite heruntergeladen werden kann (vgl. *AVR8 Burn-O-Mat - a GUI for avrdude* 2016).

Wichtig Atmel bezeichnet Fusebits als *programmed*, wenn sie auf 0 gesetzt und als *unprogrammed*, wenn sie auf 1 gesetzt sind. Viele Programme zum Einstellen der Fusebits bieten eine Checkbox für die Einstellung an, die angehakt werden muss, um das Bit auf 0 zu setzen. Das kann verwirrend sein. In diesem Fall ist Vorsicht geboten. Die Dokumentation der entsprechenden Software sollte konsultiert werden, da eine falsche Einstellung der Taktquelle dazu führen kann, dass der Controller nicht mehr reagiert und zur „Wiederbelebung“ ein Taktsignal mittels Funktionsgenerator eingespeist werden muss.

8. Übersicht Mikrocontrollersoftware

Die Programmiersprache C erlaubt eine Unterteilung des Programms in unterschiedliche Module. Der Quellcode zu jedem Modul befindet sich in einer Datei mit der Endung .c. Die öffentlichen Variablen und Funktionen eines Moduls sind jeweils in einer Header-Datei mit der Endung .h aufgelistet, die üblicherweise (bis auf die Endung) den gleichen Dateinamen wie das entsprechende Modul trägt. Diese Headerdatei kann in anderen Modulen eingebunden werden, die auf die Funktionalität der aufgelisteten Funktionen angewiesen sind.

Diese Unterteilung in verschiedene Softwaremodule erlaubt eine Strukturierung der komplexen Software in einfachere Teilmodule. So können beispielsweise die Details zur Ansteuerung und Konfiguration eines Bausteins in einem Modul verpackt werden. Der Teil der Software, der den Baustein nutzen möchte, braucht dadurch nicht zu wissen, wie die Kommunikation mit dem Baustein im Detail aussieht, sondern kann auf die Funktionen des entsprechenden Softwaremoduls zurückgreifen.

8.1. Bausteinbibliotheken

Für die verschiedenen im Projekt vorkommenden Bausteine wird jeweils ein eigenes Softwaremodul erstellt:

DDS: Das DDS-Softwaremodul kümmert sich um die Kommunikation mit dem DDS-Signalgeneratorbaustein und greift dazu auf die SPI-Schnittstelle zu.
→ dds.c und dds.h

TWI: Das TWI-Modul liefert Funktionen zur Verwendung der I^2C-Schnittstelle am Mikrocontroller.
→ twi.c und twi.h

RTC: Dieses Softwaremodul übernimmt die Verwaltung der Real-Time Clock, die über den I^2C-Bus mit dem Mikrocontroller verbunden ist und bedient sich dazu der vom TWI-Modul bereitgestellten Funktionen.
→ rtc.c und rtc.h

EEPROM: Der externe EEPROM-Baustein ist ebenfalls über den I^2C-Bus verbunden. Das zugehörige Softwaremodul greift daher wiederum auf die TWI-Funktionen zurück.
→ ext_eeprom.c und ext_eeprom.h

RFID: Zur Ansteuerung des RFID-Moduls, das über die SPI-Schnittstelle angeschlossen ist, wird eine bereits vorhandene, für das Arduino-System konzipierte Bibliothek angepasst und in das Projekt eingebunden. Die Dateien rfid.c und rfid.h sind für dieses Projekt erstellt und greifen auf die vorgefertigte Bibliothek in den Dateien MFRC522.c und MFRC522.h zu.
→ Arduino-Bibliothek: MFRC522.c und MFRC522.h
→ Arduino-Kompatibilitätsschicht: SPI.c, SPI.h, Arduino.c und Arduino.h
→ Schnittstelle zum Projekt: rfid.c und rfid.h

UART: Das UART-Softwaremodul implementiert einen Sende- und Empfangsbuffer für die UART-Schnittstelle, die für die Kommunikation des Senders mit dem PC verwendet wird.
→ uart.c und uart.h

8.2. Höhere Softwareschicht

Die Bibliotheken selbst liefern nur Funktionen zur Ansteuerung verschiedener Bausteine, verfügen jedoch selbst nicht über Informationen, in welchen Situationen sich der Sender wie verhalten soll. Die eigentliche Steuerung des Senders erfolgt in folgenden Modulen:

Commands: Das Commands-Softwaremodul übernimmt die Kommunikation mit dem PC zur Konfiguration der Sendereinstellungen. Hier sind alle Konfigurationskommandos definiert, die vom Sender ausgewertet werden können. Das Modul greift auf die Funktionen des UART-Moduls zurück.
→ commands.c und commands.h

Startup: Das Startup-Modul übernimmt die Verwaltung der Start- und Stoppzeitpunkte des Wettbewerbs. Dazu greift das Modul auf die RTC-Bibliothek zurück, um die aktuelle Uhrzeit abzufragen und um den Timer der Real-Time Clock zu stellen, der beim Erreichen der Start- oder Stoppzeit einen Interrupt auslöst.
→ startup.c und startup.h

Morse: Das Morse-Modul übernimmt die Generierung von den Morsesignalen, die ausgesendet werden. Das Ausgangssignal des DDS-Bausteins wird im richtigen Rhythmus aktiviert und wieder deaktiviert, sodass die korrekten Morsesignale abgestrahlt werden.
→ morse.c und morse.h

User: Das User-Modul übernimmt die Verwaltung der RFID-Transponder. Jeder Teilnehmer erhält einen Transponder, über den er sich beim Sender registrieren kann, um zu bestätigen, dass der Sender aufgespürt worden ist. Das User-Modul übernimmt dabei, aufbauend auf der RFID-Bibliothek und der Bibliothek für den externen EEPROM, die Verwaltung der Teilnehmereinträge.
→ user.c und user.h

Utils: In diesem Softwaremodul befinden sich einige häufig gebrauchte Funktionen für den Umgang mit Zeichenketten, die Konvertierung von Integer-Datentypen in Text usw.
→ utils.c und utils.h

Main: Das Hauptmodul enthält den Eintrittspunkt des Programms. Von hier aus werden die anderen Module initialisiert und das Programm gestartet. In diesem Modul ist auch die Hauptschleife des Programms enthalten, die während der gesamten Programmausführungsdauer ständig durchlaufen wird und verschiedene Tätigkeiten wie beispielsweise das Abfragen neuer RFID-Transponder erfüllt. Alle Aufgaben, die nicht durch die diversen Interrupts gestartet werden, müssen in der Hauptschleife eingebaut sein.
→ main.c und main.h

8.3. Konventionen für Softwaremodule

Um eine einheitliche Gestaltung der Software zu erreichen werden verschiedene Regeln festgelegt.

Funktionsnamen: Namen von Funktionen, die in anderen Modulen verwendet werden, tragen als Prefix immer den Namen des Softwaremoduls. So beginnen beispielsweise alle öffentlichen Funktionen des Morse-Softwaremoduls mit `morse_`. Interne Funktionen, die außerhalb eines Moduls nicht verwendet werden, dürfen von der Konvention abweichen.

Coding-Style: C erlaubt eine sehr freie Gestaltung des Codes hinsichtlich Zeilenumbrüchen, Position der Schweifklammern und Leerzeichen. Im Internet existiert eine Vielzahl verschiedener Konventionen mit verschiedenen Richtlinien. In diesem Projekt wird versucht, den Coding-Style des Linux-Kernels weitgehend einzuhalten (vgl. *Linux kernel coding style* 2016).

Modulinitialisierung: Viele Module benötigen beim Programmstart eine Initialisierung, um einen definierten Zustand zu erreichen und die Funktionalität korrekt bereitstellen zu können. Für diese Aufgabe werden pro Modul drei Funktionen erstellt. Die Funktionen sind nach dem Schema `modulname_init`, `modulname_show_configuration` und `modulname_load_configuration` benannt. Sollten eine oder mehrere dieser Funktionen im Modul nicht benötigt werden, können diese auch entfallen.

Die Funktion `show_configuration` gibt die getroffenen Einstellungen und den Status des Moduls über die UART-Schnittstelle aus.

Die `load_configuration`-Funktion dient zum Einlesen der im EEPROM gespeicherten Konfiguration in das RAM des Mikrocontrollers. Da durch die Konfigurationskommandos im Commands-Modul jeweils nur die EEPROM-Konfiguration verändert wird, muss zur sofortigen Übernahme der Einstellung die entsprechende `load_configuration`-Funktion des betroffenen Moduls ausgeführt werden.

Die Init-Funktion kümmert sich um die von der EEPROM-Konfiguration unabhängigen Initialisierung des Moduls und ruft zusätzlich die `load_configuration`-Funktion des Moduls auf.

8.4. Moduldokumentation

Im Folgenden werden Besonderheiten und Details der einzelnen Module beschrieben. Zu beachten ist jedoch, dass aufgrund des Codeumfangs auf ein Abdrucken weiter Teile des Quellcodes verzichtet worden ist. Beim Lesen der Dokumentation ist es für das Verständnis hilfreich, die entsprechenden Stellen im Quellcode auf der beigelegten CD parallel mitzulesen.

9. Softwaremodul DDS

9.1. Anforderungen

Die höheren Softwareschichten sollen durch das Softwaremodul für den DDS die Möglichkeit bekommen, die Funktionalität des DDS-Chips über eine Softwareschnittstelle ansprechen zu können, ohne genau darüber Bescheid wissen zu müssen, über welche Protokolle und Schnittstellen der Chip konfiguriert wird.

Das Modul muss die Möglichkeit bieten, Frequenz und Amplitude des vom DDS ausgegebenen Signals zu beeinflussen und das Signal ein- und auszuschalten. Zudem soll das Softwaremodul auch die Amplitudenmodulation erzeugen, die beim Senden auf dem 2m-Band bei Fuchsjagden üblich ist.

Der DDS-Baustein besitzt einen Quarz, der zur Erzeugung einer Referenzfrequenz dient. Zur Einstellung der Frequenz des DDS-Signals muss immer Bezug auf die Frequenz des Referenzquarzes genommen werden. Eine möglichst exakte Einstellung der Frequenz des DDS-Ausgangssignals ist somit nur möglich, wenn auch die exakte Frequenz des Quarzes bekannt ist. Das Softwaremodul soll es erlauben, die vorher exakt bestimmte Frequenz des Referenzquarzes einzustellen, sodass Abweichungen der Frequenz des Ausgangssignals vom eingestellten Wert minimiert werden können.

9.2. Überblick DDS-Baustein AD9859

In diesem Projekt kommt der DDS-Chip AD9859 der Firma Analog Devices zum Einsatz. Mit dem Baustein können Signale mit einer Frequenz von bis zu 200 MHz erzeugt werden, da er die digital-analog gewandelten Signale mit einer Samplerate von 400 Megasamples/s ausgeben kann. Der Digital-Analog-Konverter wandelt Digitalsignale mit einer Auflösung von 10-bit (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 1)

9.2.1. Schnittstelle

Der Chip wird zur Konfiguration über SPI mit dem Mikrocontroller verbunden. SPI (= Serial Peripheral Interface) ist ein serielles Bussystem bestehend aus 4 Leitungen bei zwei Busteilnehmern. Für jeden weiteren Teilnehmer wird eine zusätzliche Leitung benötigt. Ein Teilnehmer muss die Funktion des Busmasters übernehmen, alle anderen arbeiten als Slave.

Alle Teilnehmer sind an die gleichen zwei Datenleitungen und an die Taktleitung angeschlossen. Eine der Datenleitungen dient zur Übertragung der Daten vom Master zum Slave (MOSI - Master Out Slave In), auf der anderen werden die Daten von den Slaves zum Master übertragen (MISO - Master In Slave Out). Damit nicht alle Slaves gleichzeitig senden, gibt es vom Master zu jedem Slave eine zusätzliche Chip-Select-Leitung, über die der Master den gewünschten Slave auswählt.

Die Daten werden immer gleichzeitig in beide Richtungen übertragen, sobald der Master beginnt, Pulse auf der Taktleitung zu erzeugen. Nach 8 Taktimpulsen ist somit die Übertragung eines Bytes vom Master zum Slave und umgekehrt abgeschlossen. Sollte nur die Übertragung in eine Richtung erforderlich sein, so werden in der anderen Richtung Pseudo-Daten ohne Bedeutung übertragen (z.B. lauter binäre Nullen).

9.2.2. Leitungen

Zwischen Mikrocontroller und DDS werden verschiedene Leitungen zur Steuerung benötigt. Einerseits müssen vier Leitungen verbunden werden, die für den SPI-Bus erforderlich sind. Die beiden Datenleitungen werden im Datenblatt des DDS-Chips statt MOSI als SDIO (Serial Data In/Out) und statt MISO als SDO (Serial Data Out) bezeichnet. Das In und Out ist hierbei aus Sicht des DDS-Chips, also des Slaves, zu sehen. SCLK und CS sind wie üblich bezeichnet.

Daneben besitzt der DDS-Chip noch weitere Leitungen, die mitunter (je nach Anforderungen des Projekts) zur Verbindung gebraucht werden.

I/O UPDATE: Werden die Register über die serielle Schnittstelle mit Daten beschrieben, so wird die Konfigurationsänderung erst aktiv, sobald eine steigende Flanke an der I/O UPDATE-Leitung erzeugt wird.

PWRDWNCTL: Durch Setzen dieser Leitung auf high kann der Chip in einen Stromsparmodus versetzt werden.

OSK: Mit diesem Pin können Rampen der Amplitude des Ausgangssignals erzeugt werden. Z.B. kann mit diesem Pin eine einfache Amplitudenmodulation des Ausgangssignals realisiert werden. Eine andere Möglichkeit zur Generierung von AM-Signalen ist es, im Takt des modulierenden Signals jeweils die Amplitude des Signals über das entsprechende Amplituden-Register zu verändern.

9.2.3. Register

Die gesamte Konfiguration des AD9859 erfolgt durch das Schreiben in verschiedene Konfigurationsregister (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 13). Es gibt beispielsweise Register, in denen Frequenz und Amplitude des Ausgangssignals eingetragen werden. Jedem Register ist eine Adresse zugeordnet. Die Register sind teilweise unterschiedlich groß und umfassen zwischen 16 und 32 Bits.

Zum Schreiben oder Lesen eines Registers muss über den SPI-Bus zuerst ein Befehlsbyte übertragen werden, das festlegt, ob es sich um einen Schreibe- oder Lesezugriff handelt und welches Register angefordert wird (Tabelle 9.1).

D7	D6	D5	D4	D3	D2	D1	D0
R/$\overline{\text{W}}$	x	x	A4	A3	A2	A1	A0

Tabelle 9.1.: Aufbau des Befehlsbytes des AD9859

R/$\overline{\text{W}}$	...	0 → Schreiben, 1 → Lesen
x	...	don't care
A4-A0	...	Adresse des Registers

Nach dem Senden des Befehlsbytes müssen genau so viele Bytes gelesen bzw. geschrieben werden, wie das entsprechende Register enthält. Erst dann gilt die Übertragung für den DDS-Baustein als abgeschlossen und ein neues Befehlsbyte kann gesendet werden.

Amplitude Scale Factor - ASF: In diesem 16-bit breiten Register repräsentieren die niederwertigsten 14 Bits einen Wert, der proportional zur Amplitude des Ausgangssignals ist. Mit den höchstwertigen 2 Bits kann die Anstiegsrate bei Verwendung der Rampen durch den OSK-Pin eingestellt werden.

Frequency Tuning Word - FTW: In diesem Register wird ein 32-bit Wert abgespeichert, der für die Frequenz des Ausgangssignals steht. Die Frequenz des Ausgangssignals kann durch folgende Formel berechnet werden:

$$f_{\mathrm{a}} = f_{\mathrm{sysclk}} \cdot \frac{\mathrm{FTW}}{2^{32}} \tag{9.1}$$

f_{a} ist die Frequenz des Ausgangssignals, FTW ist der im Register eingestellte Zahlenwert und f_{sysclk} ist die durch die PLL um einen einstellbaren Faktor vervielfachte Frequenz des Referenzquarzes (siehe CFR2-Register). Die Formel gilt nur, wenn FTW kleiner als 2^{31} ist, ansonsten nimmt die Frequenz wieder ab und kann nach dieser zweiten Formel berechnet werden:

$$f_{\mathrm{a}} = f_{\mathrm{sysclk}} \cdot \left(1 - \frac{\mathrm{FTW}}{2^{32}}\right) \tag{9.2}$$

Control Function Register 1 - CFR1: In diesem Register können verschiedene allgemeine Einstellungen für den DDS-Chip getroffen werden. Beispielsweise ist es wichtig für den Anschluss an das SPI-Interface des AVRs, das Bit *SDIO Input Only* zu setzen, weil ansonsten nur die SDIO-Leitung als Datenleitung für beide Richtungen benützt wird. Für den AVR werden jedoch zwei Datenleitungen benötigt. Mit den beiden Bits *OSK Enable* und *Auto OSK Keying* kann das Amplitudenverhalten des Chips beeinflusst werden (Tabelle 9.2).

OSK Enable	Auto OSK Keying	
0	x	Amplitude unabhängig vom Register ASF
1	0	Manual OSK Mode: Jeder Ausgangswert wird vor der Ausgabe mit den niederwertigsten 10-bit des ASF-Registers multipliziert → Möglichkeit zur Amplitudeneinstellung
1	1	Auto OSK Mode: Die Amplitudensteuerung erfolgt zusätzlich über den OSK-Pin. Mit Bit 13 bis 0 im ASF-Register wird eingestellt, welche maximale Amplitude erreicht werden kann, Bit 15 und 14 dienen zur Konfiguration der Anstiegs- bzw. Abfallrate.

Tabelle 9.2.: Amplitudenverhalten abhängig von den OSK-Bits

9.3. Softwareschnittstelle

Module in der Programmiersprache C bestehen jeweils aus einer C-Datei (hier dds.c) und einer Header-Datei (hier dds.h). Die C-Datei wird zu einer Objektdatei kompiliert, die als Programmbibliothek dient und am Ende mit dem Hauptprogramm verlinkt wird.

In der Header-Datei stehen alle Funktionen, Variablen und Definitionen, die verwendet werden können, um das Modul anzusteuern.

```
1  /*
2   * function declarations
3   */
4
5  void dds_init(void);
6  void dds_show_configuration(void);
7  void dds_load_configuration(void);
8
9  uint8_t dds_set_frequency(uint32_t frequency);
10 uint32_t dds_get_frequency(void);
11
12 uint8_t dds_set_crystal_frequency(uint32_t frequency);
13 uint32_t dds_get_crystal_frequency(void);
14
15 void dds_set_modulation(uint8_t mod);
16 uint8_t dds_get_modulation(void);
17
18 uint8_t dds_set_amplitude_percentage(uint8_t amp);
19 uint8_t dds_get_amplitude_percentage(void);
20
21 uint8_t dds_on(void);
22 uint8_t dds_off(void);
23
24 void dds_execute_command_buffer(void);
25
26 void dds_disable_continuous_carrier(void);
27 void dds_enable_continuous_carrier_2m(void);
28 void dds_enable_continuous_carrier_80m(void);
29
30 void dds_powerdown(void);
31 void dds_powerup(void);
```

Listing 9.1: Softwareschnittstelle zum DDS-Modul aus dds.h

Mit den Funktionen `dds_set_frequency`, `dds_set_crystal` und `dds_set_amplitude_percentage` können jeweils die Werte für die Frequenz des Ausgangssignals, die Frequenz des Referenzquarzes und der maximalen Amplitude des Ausgangssignals, die im EEPROM gespeichert werden, eingestellt werden.

Analog dazu dienen `dds_get_frequency`, `dds_get_amplitude_percentage` und `dds_get_crystal`, um die Werte aus dem EEPROM auslesen zu können.

Mit `dds_set_modulation` und `dds_get_modulation` kann die Amplitudenmodulation des Ausgangssignals aktiviert oder deaktiviert werden. Die Konfiguration wird ebenfalls im EEPROM gespeichert bzw. aus dem EEPROM gelesen.

Das alleinige Ändern der Werte im EEPROM führt noch nicht zu einer Änderung des tatsächlichen DDS-Zustandes. Die API ist gemäß der Konvention (siehe Abschnitt 8.3 auf Seite 109) so gestaltet, dass die geänderte Konfiguration am Ende mit `dds_load_configuration` neu aus dem EEPROM geladen und zum DDS übertragen wird, sodass ein kontrolliertes Neu-Laden der Einstellungen zu einem definierten Zeitpunkt möglich ist.

`dds_init` muss am Beginn aufgerufen werden, um die SPI-Schnittstelle am Mikrocontroller zu initialisieren und um eine Standardkonfiguration in den DDS zu schreiben, die für den regulären Betrieb benötigt wird. Außerdem lädt die Funktion auch die aktuelle Konfiguration aus dem EEPROM und führt sie aus (indem sie `dds_load_configuration` aufruft).

Mit `dds_show_configuration` kann die aktuelle im EEPROM gespeicherte Konfiguration zu Debugzwecken auf der seriellen Schnittstelle ausgegeben werden.

`dds_on` und `dds_off` beeinflussen den aktuellen Zustand des DDS-Ausgangssignals und schalten das Signal ein bzw. aus. Wenn `dds_on` oder `dds_off` fehlschlägt, weil die SPI-Schnittstelle momentan besetzt und keine Übertragung möglich ist, wird der zuletzt ausgeführte Befehl in einem Buffer abgespeichert und dann ausgeführt, wenn das nächste Mal `dds_execute_command_buffer` ausgeführt wird.

`dds_enable_continuous_carrier_80m` und `dds_enable_continous_carrier_2m` aktivieren eine Testaussendung im 2m- bzw. 80m-Band zum Abstimmen der Antenne. Dabei wird ein dauerhafter, unmodulierter Träger ausgesendet, der weder durch Morsen noch durch die eventuell aktivierte Amplitudenmodulation oder die Start- bzw. Stoppzeitpunkte beeinflusst wird. Mit `dds_disable_continuous_carrier` oder durch Aufruf von `dds_load_configuration` kann der Modus wieder deaktiviert werden.

`dds_powerdown` versetzt den DDS-Baustein in einen Stromsparmodus, während `dds_powerup` den Baustein wieder aktiviert.

9.4. Implementierung

Der Code zu den in der Header-Datei deklarierten Funktionen, steht in der entsprechenden C-Datei dds.c.

9.4.1. dds_init:

Zu Beginn werden die Pins, die mit dem DDS-Chip verbunden sind, initialisiert.

```
1 DDS_PORT &= ~((1 << DDS_PWRDWNCTL) | (DDS_IOSYNC) | (1 << DDS_IO_UPDATE));
2 DDS_PORT |= (1 << DDS_CS);
3 DDS_DDR  |= (1 << DDS_CS) | (1 << DDS_PWRDWNCTL) | (1 << DDS_IO_UPDATE) |
4             (1 << DDS_IOSYNC) | (1 << DDS_MOSI) | (1 << DDS_SCK);
```

Listing 9.2: Initialisierung der Pins für DDS

Pins beim AVR-Mikrocontroller sind nach Ports organisiert. Jeder Port besitzt 8 Pins und zu jedem Port existiert ein PORT-Register, über das die Ausgangszustände der Pins eingestellt werden können. Dazu müssen die Pins jedoch erst als Ausgang konfiguriert werden, was über die entsprechenden DDR-Register (Data Direction Register) erfolgt. Jedes der 8 Bits im DDR-Register steht für einen Pin. Ist dieses Bit auf 1 gesetzt, so ist der Pin als Ausgang konfiguriert, ist das Bit auf 0 gesetzt, so bleibt der Pin ein Eingang. (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 66 ff.).

Welche Pins vom Mikrocontroller mit den Pins vom DDS verbunden sind, hängt vom entsprechenden Platinenlayout ab. Da sich die Pins im Laufe des Entwicklungsprozesses ändern können, wird eine Abstraktion eingeführt, indem die entsprechenden Register über einen allgemeinen Namen (z.B. `DDS_PORT`) angesprochen werden. Die Zuordnung von beispielsweise `DDS_PORT` auf das entsprechende Register des Mikrocontrollers erfolgt mittels einer Präprozessor-#define-Direktive in der Header-Datei pins.h.

Die einzelnen Pins werden durch die entsprechenden #define-Konstanten wie beispielsweise `DDS_CS` für die Chip-Select-Leitung angesprochen, welche auf den Wert der Pinnummer im jeweiligen gewählten Port `DDS_PORT` gesetzt sind. Somit können beim Layout der Platine der Port und die jeweiligen Pinnummern frei gewählt werden, weil sie nur an einer Stelle im Programm in der Datei pins.h eingestellt werden müssen. Einzige Bedingung, um umfassendere Änderungen im Quellcode des Programms zu vermeiden ist, dass alle den DDS betreffenden Pins an den gleichen Port angeschlossen sind.

Alle Pins werden über die Port-Register auf den neutralen, nicht aktiven Zustand gesetzt. Bei der Chip-Select-Leitung `DDS_CS` bedeutet dies, dass die Leitung auf high gesetzt wird, bei den anderen Leitungen, dass sie auf low gesetzt werden. Bei den Leitungen `DDS_MOSI` und `DDS_MISO` ist der Zustand des Portregisters unerheblich, weil die Leitungen von der in Hardware implementierten SPI-Einheit des Mikrocontrollers verwaltet werden.

Zur Konfiguration des SPI-Interfaces dienen die Register SPCR (SPI Control Register) und SPSR (SPI Status Register). Der Mikrocontroller wird als SPI-Master mit der schnellsten Taktgeschwindigkeit für die Taktleitung $\frac{1}{2} f_{\text{Quarz}}$ konfiguriert. Eingestellt werden muss auch, ob bei einem Byte über SPI das niederwertigste oder das höchstwertigste Bit zuerst übertragen wird und ob die an den Datenleitungen angelegten Bits bei der fallenden oder steigenden Flanke der Taktleitung geändert werden und bei der jeweils anderen Flanke eingelesen werden. Die Standardeinstellungen des Mikrocontrollers *höchstwertiges Bit zuerst* und *Einlesen bei steigender Flanke*, *Ändern bei fallender Flanke* stimmen mit den für den DDS-Chip erforderlichen Einstellungen überein, wie im Zeitdiagramm der SPI-Übertragung im Datenblatt des DDS-Bausteins ersichtlich ist (vgl. *AD9859 Direct Digital Synthesizer Datasheet* 2009, Seite 20).

```
1 SPCR |= (1 << SPE) | (1 << MSTR);
2 SPSR |= (1 << SPI2X);
```

Listing 9.3: Konfiguration SPI Interface

Der Mikrocontroller bietet auch Interrupts an, die bei Beendigung einer SPI-Übertragung aufgerufen werden. Das Programm im vorliegenden Fall greift nicht auf diese Interrupts zurück, weil die Software problemlos bis zur Beendigung der Übertragung warten kann.

9.4.2. dds_write_register und dds_read_register:

Die beiden nur intern im Modul verfügbaren Funktionen `dds_write_register` und `dds_read_register` stellen eine einheitliche Schnittstelle zum Schreiben in die DDS-Register zur Verfügung, sodass nicht jedes Mal erneut der Code zur Erzeugung des Befehlsbytes und zum Senden per SPI eingefügt werden muss.

Da auch das RFID-Modul auf den SPI-Bus zugreifen muss, wird zuerst mithilfe der Funktion `SPI_is_free` kontrolliert, ob der SPI-Bus derzeit benützt wird. Alle Teile im Programm, die SPI benützen, warten ab, bis die SPI-Übertragung abgeschlossen ist, bevor das Programm fortsetzt. Grundsätzlich kann daher keine gleichzeitige Benützung der SPI-Schnittstelle im Hauptprogramm auftreten. Da aber die Amplitudenmodulation des DDS-Bausteins über einen Timerinterrupt gesteuert wird, der das Hauptprogramm unterbricht und der auch die Funktion `dds_write_register` aufruft, muss die Funktion überprüfen, ob das Hauptprogramm nicht gerade während einer SPI-Übertragung vom Interrupt unterbrochen worden ist.

Ist der SPI-Bus frei, so wird die Chip-Select-Leitung auf low gesetzt, was die SPI-Schnittstelle beim DDS-Baustein aktiviert, und das Befehlsbyte in das Register

SPDR (SPI Data Register) geschrieben. Anschließend werden die Registerdaten, die über den Pointer `data` als Parameter übergeben worden sind, zum DDS übertragen. Die Länge der Daten muss über einen weiteren Parameter `length` festgelegt werden. Die Funktion selbst hat keine Kenntnis darüber, wie groß die einzelnen Register sind. Am Ende der Funktion wird die Chip-Select-Leitung wieder freigegeben und auf high gesetzt.

Die Funktion `wait_spi`, die innerhalb der Schreibfunktion aufgerufen wird, liest aus dem SPI-Statusregister SPSR solange das Interruptflag SPIF aus, bis dieses signalisiert, dass die Übertragung abgeschlossen ist.

```
1  /**
2   * dds_write_register - write dds register
3   * @register_address:   address of the dds register to write into
4   * @data:               pointer to data to send in SRAM
5   * @length:             length of the register (and therefore the array that
6   *                      data points to) in bytes
7   *
8   *      Returns TRUE if spi was free and dds could be written, returns
9   *      FALSE if spi was not free (rfid was sending at the moment)
10  */
11 static uint8_t dds_write_register(int8_t register_address, const int8_t *data, ↩
       int8_t length)
12 {
13     if (SPI_is_free() == FALSE) {
14         return FALSE;
15     }
16
17     DDS_PORT &= ~(1 << DDS_CS);
18     SPDR = register_address;
19     wait_spi();
20     uint8_t i;
21     for (i = 0; i < length; i++)    {
22         SPDR = data[i];
23         wait_spi();
24     }
25     DDS_PORT |= (1 << DDS_CS);
26
27     DDS_PORT &= ~(1 << DDS_IOSYNC);
28     DDS_PORT |= (1 << DDS_IOSYNC);
29     DDS_PORT &= ~(1 << DDS_IOSYNC);
30
31     return TRUE;
32 }
```

Listing 9.4: Funktion `dds_write_register`

Analog dazu ist die Funktion `dds_read_register` zum Lesen aus einem DDS-Register aufgebaut. Die beiden Funktionen unterscheiden sich nur im Befehlsbyte, wo das R/$\overline{\text{W}}$-Bit gesetzt wird. Schreiben und Lesen erfolgt beim SPI-Bus immer gleichzeitig. Um nun aus dem Register zu lesen, müssen beliebige Pseudo-Daten (beispielsweise ein Byte mit dem Wert 0) übertragen werden. Der DDS-Baustein wird gleichzeitig die zu empfangenden Daten des Registers senden. Diese Daten

liegen dann im Register SPDR vor und werden von der Funktion in den als Parameter übergebenen Lesebuffer `data` geschrieben.

```
/**
 * dds_read_register - read a dds register
 * @register_address:   address of the dds register to read
 * @data:               shall point to an array where the read data will
 *                      get stored
 * @length:             length of the register in bytes, buffer pointed to
 *                      by data must have at least the size of length-bytes
 *
 *      Returns TRUE if spi was free and data could be read, returns
 *      FALSE if spi was not free (rfid was sending at the moment)
 */
static uint8_t dds_read_register(int8_t register_address, int8_t *data, int8_t ↩
    length)
{
    if (SPI_is_free() == FALSE) {
        return FALSE;
    }

    DDS_PORT &= ~(1 << DDS_CS);
    SPDR = register_address | (1 << 7);
    wait_spi();
    uint8_t i;
    for (i = 0; i < length; i++)    {
        SPDR = 0x00;
        wait_spi();
        data[i] = SPDR;
    }
    DDS_PORT |= (1 << DDS_CS);

    DDS_PORT &= ~(1 << DDS_IOSYNC);
    DDS_PORT |= (1 << DDS_IOSYNC);
    DDS_PORT &= ~(1 << DDS_IOSYNC);

    return TRUE;
}
```

Listing 9.5: Funktion `dds_read_register`

9.4.3. Funktionen zum Setzen der Einstellungen

Die verschiedenen den DDS betreffenden Einstellungen wie die gewünschte Frequenz des Ausgangssignals, die kalibrierte Frequenz des Schwingquarzes und den Zustand der Amplitudenmodulation werden ebenfalls durch das DDS-Softwaremodul verwaltet.

Da die Einstellungen auch ohne Betriebsspannung gespeichert bleiben sollen, werden sie im EEPROM des Mikrocontrollers abgelegt.

Das Modul bietet Funktionen wie `dds_set_frequency` und `dds_get_frequency` an, die jeweils zum Schreiben bzw. Lesen der entsprechenden EEPROM-Variablen verwendet werden können.

Nach Aufruf der Set-Funktionen werden die Einstellungen nur ins EEPROM geschrieben. Die aktuell gültigen Einstellungen befinden sich jedoch im RAM des Mikrocontrollers. Um die Frequenz des Ausgangssignals im laufenden Betrieb zu ändern, muss daher nach Aufruf der Set-Funktionen die Funktion `dds_load_configuration` aufgerufen werden, um alle Einstellungen aus dem EEPROM ins RAM zu laden.

9.4.4. dds_write_output_ftw

Die Frequenzeinstellung im EEPROM erfolgt über die Funktion `dds_set_frequency`. Die Funktion schreibt durch Aufruf der `dds_write_register`-Funktion in das korrekte Register im DDS-Chip, um die Frequenz zu setzen.

Als Parameter wird das vom DDS-Baustein benötigte Frequency Tuning Word erwartet. Die Frequenz in Hz kann mithilfe des Makros `FREQ_TO_FTW` in das Frequency Tuning Word umgerechnet werden.

9.4.5. dds_write_amplitude

Diese Funktion dient zum Übertragen der Ausgangssignalamplitude zum DDS-Baustein. Für die Übertragung wird wiederum die `dds_write_register`-Funktion aufgerufen.

9.4.6. Amplitudenmodulation

Im 2m-Frequenzband ist es bei Fuchsjagden üblich, das Morsesignal amplitudenmoduliert auszustrahlen. Die Erzeugung der Amplitudenmodulation ist ebenfalls Aufgabe des DDS-Softwaremoduls.

Zur Erzeugung wird ein Timer benützt, der Interrupts mit einem ganzzahligen Vielfachen der Modulationsfrequenz erzeugt und bei jedem Überlauf den nächsten Amplitudenwert aus einer Sinustabelle ausliest und in das Amplitudenregister des DDS-Bausteins schreibt.

Mehrfachverwendung der SPI-Schnittstelle: Eine Herausforderung bei der Implementierung der Modulation ist es, neben dem DDS-Baustein auch dem RFID-Modul die SPI-Schnittstelle zur Kommunikation zur Verfügung zu stellen. Die Datenpakete, die für die Modulation übertragen werden müssen, sind zeitkritisch, da bei falschen Timings das Ausgangssignal verzerrt wird. Das Modulationssignal soll eine Frequenz von etwa 600 Hz besitzen. Um eine brauchbare Annäherung an einen sinusförmigen Verlauf zu erreichen, ist es sinnvoll, 10 oder mehr Amplitudenwerte pro Periode auszugeben. Bei 10 Werten pro Periode ergibt sich somit eine Frequenz von 6 kHz, mit der Datenpakte zum DDS übertragen werden müssen.

Das RFID-Modul dient zur Registrierung der Teilnehmer, die jeweils einen RFID-Transponder besitzen. Es ist nicht unwahrscheinlich, dass während der Sendezeit eines Senders ein Teilnehmer zum Sender kommt und sich dort registrieren möchte. Aus diesem Grund muss das RFID-Modul auch während der Aussendungen aktiv sein. Die RFID-Kommunikation ist relativ zeitunkritisch. Im ungünstigsten Fall dauert die Übertragung der Transponderdaten einige Sekunden länger, was kein Problem darstellt. Ist die Schnittstelle bereits mit der Übertragung eines Datenbytes beschäftigt, so kann kein weiteres Byte gleichzeitig übertragen werden. Aus diesem Grund ist es wichtig, dass das RFID-Modul während der Zeitpunkte, zu denen das DDS-Sofwaremodul ein Datenpaket senden möchte, nicht sendet. Ansonsten würde sich die für die Amplitudenmodulation notwendige Übertragung verzögern.

Eine SPI-Übertragung beginnt jeweils mit der Aktivierung der Chip-Select-Leitung zum entsprechenden Baustein und endet mit der Freigabe der Chip-Select-Leitung. Da die verwendete RFID-Bibliothek ursprünglich für das Arduino-System erstellt worden ist, verwendet sie zum Setzen des Ausgangsspins nicht direkt die Mikrocontrollerregister, sondern eine vom Arduino-System zur Verfügung gestellte Funktion `digitalWrite`. Diese Funktion wird im Projekt nun so gestaltet, dass sie die Aktivierung der Chip-Select-Leitung nur zulässt, wenn der SPI-Bus frei und nicht reserviert ist. Einige Zeit bevor das DDS-Softwaremodul ein Datenbyte übertragen möchte, setzt es eine Variable, die signalisiert, dass der Bus bald benötigt wird. Von diesem Zeitpunkt an lässt die `digitalWrite`-Funktion keine Aktivierung der Chip-Select-Leitung zum RFID-Modul zu, sondern blockiert die RFID-Bibliothek so lange, bis die DDS-Übertragung für die Modulation abgeschlossen ist. Die RFID-Kommunikation erfolgt über Code aus der Hauptschleife. Da die Modulation über einen Interrupt gesteuert wird, ist die Blockierung des Codes in der Hauptschleife unkritisch. Der Controller hängt sich nicht auf, sondern arbeitet normal weiter, sobald der nächste Interrupt aufgetreten ist.

Entscheidend ist die Wahl der Zeitspanne, für die der Bus reserviert wird, bevor er vom DDS-Modul benötigt wird. Prinzipiell könnte die RFID-Bibliothek eine

Busübertragung einmal starten und den Bus andauernd belegt halten, sodass die DDS-Kommunikation nicht mehr möglich ist. Die RFID-Bibliothek ist allerdings so gestaltet, dass die Kommunikation in Teilstücke unterteilt ist, zwischen denen die Chip-Select-Leitung immer wieder freigegeben wird. Die Zeitspanne, für die der Bus vorher reserviert wird, muss also größer sein als das längste Teilstück in der RFID-Kommunikation, damit garantiert werden kann, dass der Bus bereits frei ist, wenn er vom DDS-Modul benötigt wird. Eine zu lange Zeitspanne ist allerdings ebenfalls ungünstig, da dadurch der Prozentsatz der Zeit, der für die Kommunikation mit dem RFID-Modul zur Verfügung steht, kleiner wird und die RFID-Kommunikationen dadurch mehr Zeit in Anspruch nehmen.

Die Zeitspanne kann über die beiden Konstanten `TIMER0_OVERFLOW_NUMBER` und `TIMER0_OVERFLOW_SET_SPI_OFF` eingestellt werden. `TIMER0_OVERFLOW_NUMBER` ist die Anzahl der Zeiteinheiten, in die die Zeitspanne zwischen zwei Übertragungen zum DDS-Baustein eingeteilt wird. `TIMER0_OVERFLOW_SET_SPI_OFF` stellt die Anzahl der Zeiteinheiten dar, die nach einer DDS-Kommunikation vergehen, bis der SPI-Bus reserviert wird. Über das Verhältnis der beiden Werte kann die Zeitspanne für die Reservierung der Schnittstelle eingestellt werden. Je kleiner `TIMER0_OVERFLOW_SET_SPI_OFF` im Vergleich zu `TIMER0_OVERFLOW_NUMBER` ist, desto größer ist die Zeitspanne zwischen Reservierung und tatsächlichem Gebrauch der SPI-Schnittstelle durch das DDS-Modul und desto kleiner ist die resultierende Zeitspanne für die Übertragung zum RFID-Modul.

Sinustabelle: Der grundsätzliche Verlauf der Sinustabelle ist fest im Flashspeicher in der Variable `sin_table` eingetragen. Die konkreten Amplitudenwerte zu einer bestimmten eingestellten Maximalamplitude müssen jedoch während der Programmausführung berechnet werden. Eine Berechnung innerhalb der Interruptroutine des Timers ist nicht sinnvoll, da die Berechnungen ohne Änderung der Maximalamplitude bei jeder Periode gleich sind. Aus diesem Grund wird die Berechnung einmal bei Aufruf der `dds_load_configuration`-Funktion ausgeführt und die Ergebnisse werden in den beiden RAM-Variablen `sin_table_byte0` und `sin_table_byte1` abgespeichert.

10. Softwarebibliothek TWI/I²C

10.1. Anforderungen

Der Mikrocontroller ATmega 644 verfügt bereits über eine hardwareunterstützte I²C-Schnittstelle, die im Datenblatt aus lizenzrechtlichen Gründen jedoch als TWI (Two Wire Interface) bezeichnet wird.

Aufgrund des komplexeren Protokolls als beispielsweise bei UART und SPI ist auch die Verwendung des TWI-Interfaces mit einem etwas größeren Aufwand verbunden. Damit sich die Softwaremodule, die I²C-Bausteine ansteuern sollen, nicht mit den Einzelheiten der AVR-Statuscodes und Register für TWI auseinandersetzen müssen, soll diese Softwarebibliothek eine Schnittstelle bieten, durch die Bytes über I²C gesendet und empfangen werden können.

Der Mikrocontroller arbeitet als I²C-Master, da alle verwendeten Bausteine, die über I²C verbunden sind, die Funktion von Slaves einnehmen. Die Softwarebibliothek muss somit nur die Übertragungen aus Sicht des Masters abdecken.

10.2. Übersicht I²C

I²C ist ein Bussystem, bei dem alle Teilnehmer über zwei Leitungen, der Datenleitung SDA (Serial Data) und der Taktleitung SCL (Serial Clock) miteinander verbunden sind.

Übertragungen können grundsätzlich nur vom Master initiiert werden. Im Ruhezustand liegen beide Leitungen auf high. Möchte der Master eine Übertragung beginnen, so sendet er eine Startbedingung, bei der die Taktleitung auf high bleibt, während die Datenleitung auf low wechselt.

Nach der Startbedingung wird ein Byte vom Master zu den Slaves übertragen, das eine 7-bit Adresse enthält und über das 8. Bit festlegt, ob der Master Daten nachfolgend lesen oder schreiben möchte. Anschließend werden Bytes gelesen bzw. geschrieben. Jeweils der Empfänger des Bytes (beim Lesen der Master, beim Schreiben der Slave) bestätigt den Erhalt des Bytes über ein Acknowledge-Bit,

das den Wert 0 besitzt. Nach dem Lesen des letzten Bytes soll der Master signalisieren, dass nun keine weiteren Bytes gelesen werden, indem er die Übertragung mit einer 1 als Acknowledge-Bit (NACK) abschließt.

Nach jeder Lese- bzw. Schreiboperation kann der Bus entweder durch eine Stoppbedingung wieder freigegeben werden oder durch eine neue Startbedingung (repeated start condition) eine neue Übertragung mit weiterer Adressierung starten. Dadurch kann beispielsweise ein Lesevorgang direkt nach einem Schreibvorgang folgen, ohne dass der Bus wieder freigegeben werden muss. (vgl. *I2C-Bus Specification and User Manual* 2014)

10.2.1. TWI beim AVR

Wie alle Schnittstellen und Subsysteme bei AVR-Mikrocontrollern wird auch die TWI-Schnittstelle über Register gesteuert.

Im vorliegenden Fall, dass der Mikrocontroller als Master betrieben wird, existieren vier wichtige Register:

TWI Control Register - TWCR: Mithilfe der Bits in diesem Register wird der Mikrocontroller angewiesen, am Bus verschiedene Aktionen auszuführen. Beispielsweise kann dem Controller die Anweisung gegeben werden, sobald der Bus frei ist, eine Startbedingung zu senden. Beim Lesen von Bytes kann mittels eines Bits eingestellt werden, ob der Mikrocontroller das gelesene Byte mit ACK oder NACK bestätigen soll. Das Bit TWINT (TWI Interrupt Flag) wird immer dann gesetzt, wenn die letzte Operation erfolgreich oder fehlerhaft beendet worden ist und nun wieder eine weitere Steuerung durch die Software erforderlich ist. Immer wenn dieses Bit auf high gesetzt wird, dann kann im Statusregister der aktuelle Status über den Bus ausgelesen werden, um zu entscheiden, was als Nächstes geschehen soll.

TWI Status Register - TWSR: 5 Bits in diesem Register stehen für eine Dualzahl, die den Status des TWI-Moduls repräsentiert und immer dann ausgelesen wird, wenn das Bit TWINT im Register TWCR gesetzt ist. Bei aktiviertem TWI-Interrupt wird dann ein Interrupt ausgelöst und der Status kann innerhalb der Interrupt Service Routine abgerufen werden. Die Statuscodes sind detailliert im Datenblatt aufgeführt (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 213 und 216).

TWI Data Register - TWDR: In diesem Register werden empfangene Daten abgelegt bzw. Sendedaten vor der entsprechenden Operation hineingeschrieben. Wann hier gültige Daten liegen, kann den Statuscodes entnommen werden.

TWI Bit Rate Register - TWBR: Der Wert in diesem Register stellt einen Frequenzteiler dar, durch den die Taktfrequenz der SCL-Leitung gemeinsam mit drei Prescalerbits im TWSR bestimmt wird. Die Taktfrequenz der Taktleitung errechnet sich nach folgender Formel (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 207):

$$f_{\mathrm{SCL}} = \frac{f_{\mathrm{CPU}}}{16 + 2\,(\mathrm{TWBR}) \cdot 4^{\mathrm{TWPS}}} \tag{10.1}$$

f_{SCL}	...	Frequenz der Taktleitung
f_{CPU}	...	Taktfrequenz des Mikrocontrollers
TWBR	...	Wert des TWBR Registers
TWPS	...	Wert der Prescalerbits im TWSR Register

10.3. Software

10.3.1. Softwareschnittstelle

In der Datei twi.h ist die Schnittstelle zur Verwendung der TWI-Bibliothek deklariert:

```
1 void twi_init(void);
2
3 uint8_t twi_start(void);
4 void twi_stop(void);
5
6 uint8_t twi_send_slave_address(uint8_t rw, uint8_t address);
7 uint8_t twi_send_byte(uint8_t byte);
8 uint8_t twi_read_byte(uint8_t ack, uint8_t* byte);
```

Listing 10.1: Public Functions für die TWI-Bibliothek

Um TWI am AVR zu konfigurieren, muss die Initialisierungsfunktion `twi_init` aufgerufen werden.

Start- und Stoppbedingungen können mit `twi_start` bzw. `twi_stop` ausgeführt werden. `twi_start` erzeugt entweder eine Startbedingung, wenn die letzte Operation eine Stoppbedingung gewesen ist oder eine erneute Startbedingung, wenn gerade Daten am Bus übertragen worden sind.

Nach der Startbedingung wird mit `twi_send_slave_address` der angesprochene Slave adressiert. Zusätzlich muss bei der Funktion angegeben werden, ob gelesen oder geschrieben werden soll.

Das Lesen und Schreiben selbst erfolgt nach der Adressierung mithilfe der entsprechenden Funktionen `twi_write_byte` und `twi_read_byte`. Beim Lesen wird zusätzlich angegeben, ob das gelesene Byte mit ACK oder mit NACK bestätigt werden soll, um zu unterscheiden, ob weitergelesen werden soll oder nicht.

10.3.2. Implementierung

In der Bibliothek kann auf die Header-Datei util/twi.h aus der C-Library für AVR-Mikrocontroller *AVR Libc* zugegriffen werden, in der bereits symbolische Namen für die einzelnen Statuscodes vergeben worden sind. (vgl. *Dokumentation twi.h AVR Libc* 2016).

twi_init

```
1 /**
2  * twi_init - initialize twi interface
3  */
4 void twi_init()
5 {
6     TWBR = ((F_CPU / 100000) - 16) / 2; /* set the clock rate of TWI to 100 kHz */
7     TWSR = 0x00; /* set prescaler TWI */
8     twcr_init = (1 << TWEN);
9 }
```

Listing 10.2: Funktion `twi_init`

Die Prescalerbits werden auf 0 gesetzt. Das TWBR-Register wird entsprechend der auf TWBR umgeformten Formel aus Gleichung 10.1 auf der vorherigen Seite auf 100 kHz gesetzt, was der Taktfrequenz des I²C-Standardmodes entspricht.

In der Variable `twcr_init` werden alle Flags gesetzt, die im TWCR-Register dauerhaft auf 1 bleiben sollen. Diese werden bei allen weiteren Operationen immer über die ODER-Funktion mit den Bits, die für die aktuelle Operation benötigt werden, verknüpft. In diesem Fall wird das Bit TWEN zum Aktivieren des TWI-Moduls ständig auf 1 belassen.

twi_send_byte

Alle übrigen Funktionen laufen nach dem gleichen Schema ab. Zu Beginn werden jeweils, so weit notwendig, Bits im Datenregister TWDR und im Control-Register TWCR gesetzt, um die Operation zu starten.

Anschließend wird gewartet, bis das TWINT-Flag im Register TWCR vom Mikrocontroller gesetzt wird, was kennzeichnet, dass die Operation abgeschlossen ist.

Am Ende wird jeweils der Statuscode abgeglichen und mit dem erwarteten Ergebnis verglichen. War die Operation erfolgreich, so geben alle Funktionen `TWI_OK` an den aufrufenden Codeabschnitt zurück. War die Operation nicht erfolgreich, weil z.B. der Slave das an ihn gesendete Byte nicht mit einem Acknowledge-Bit bestätigt hat, so wird `TWI_ERR` zurückgegeben.

Beispielhaft sei die Vorgangsweise anhand der Funktion `twi_send_byte` dargestellt:

```
1  /**
2   * twi_send_byte - send a byte via twi interface
3   * @byte:   byte to send
4   *
5   *      Execute a twi_start() and twi_send_slave_address() with TWI_WRITE
6   *      before using this function
7   *
8   *      Returns TWI_OK on success, TWI_ERR on failure
9   */
10 uint8_t twi_send_byte(uint8_t byte)
11 {
12     TWDR = byte;
13     TWCR = (1 << TWINT) | twcr_init;
14     while (!(TWCR & (1 << TWINT)));
15     if ((TWSR & 0xF8) != TW_MT_DATA_ACK)    {
16         return TWI_ERR;
17     }
18     return TWI_OK;
19 }
```

Listing 10.3: Funktion `twi_send_byte`

Das zu sendende Byte wird in das TWDR-Register geschrieben. Die TWI-Operation wird fortgesetzt, indem in das Bit TWINT eine 1 geschrieben wird[1].

Die `while`-Schleife wartet, bis die Operation abgeschlossen ist. Anschließend liest der Controller den Wert aus dem Statusregister aus und überprüft ihn. Die in der Datei util/twi.h (AVR Libc) definierte Konstante `TW_MT_DATA_ACK` steht für den Statuscode 0x18, der kennzeichnet, dass der Master ein Datenbyte geschrieben hat

[1] Das Bit befindet sich zwar schon auf logisch 1, jedoch muss zum Löschen des Bits, eine 1 geschrieben werden (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 226)

und dass der Slave den Empfang mittels Acknowledge bestätigt hat. Statuscodes entsprechen jeweils dem Wert des Registers TWSR unter der Bedingung, dass alle Prescalerbits auf 0 gesetzt sind.

Sollte ein anderer Statuscode aufgetreten sein, so wird `TWI_ERR`, andernfalls `TWI_OK` zurückgegeben.

11. Softwaremodul Real-Time Clock

11.1. Anforderungen

Mithilfe der RTC (Real-Time Clock) ist es möglich, mehrere Sender untereinander synchron zu halten. Dies ist wichtig, damit ein nahtloser Übergang zwischen den Sendezeiten aller Sender möglich ist und keine Überlappungen mit zwei gleichzeitig ausstrahlenden Sendern auftreten.

Die verschiedenen RTC-Module werden jeweils möglichst exakt mit der aktuellen Uhrzeit abgeglichen und dienen so zur Synchronisation der Sender untereinander. Das RTC-Modul wird zusätzlich mit einer Batterie versorgt, sodass die Uhrzeit auch weitergezählt werden kann, wenn keine andere Betriebsspannung vorhanden ist.

Dieses Softwaremodul soll Funktionen bieten mit deren Hilfe die einzelnen Einstellungen der RTC gewählt werden und die Uhrzeit abgefragt werden können. Die RTC verfügt außerdem über zwei einstellbare Uhrzeiten, bei denen ein Alarm ausgelöst wird, der über eine Interruptleitung signalisiert wird. Das Softwaremodul soll die nötige Interrupt-Service-Routine und die Verwaltung des Interrupts beinhalten und die entsprechenden Funktion aus anderen Softwaremodulen aufrufen, die vom jeweiligen Alarm informiert werden möchten.

11.2. Überblick Real-Time Clock MCP79410

Im Sender ist ein Real-Time Clock Modul MCP79410 der Firma Microchip Technology verbaut. Das Modul wird über den I^2C-Bus mit dem Mikrocontroller verbunden.

Zur Konfiguration des Bausteins müssen Daten in verschiedenen Registern eingetragen werden. Alle Register besitzen hier die konstante Länge von 1 Byte und jeweils eine Adresse mit der Länge von 1 Byte.

11.2.1. I²C-Adresse

Um den Baustein anzusprechen, muss er mit seiner I²C-Adresse adressiert werden. Jeder Baustein des Typs MCP79410 besitzt zwei Adressen, wovon eine für das Schreiben und Lesen der RTC/SRAM-Register und die andere für das Schreiben und Lesen des in der Real-Time Clock verbauten EEPROMs nötig sind.

Die letzten 3 Bits der beiden Adressen sind durch 3 Chip-Select-Leitungen des Bausteins bestimmt, die in diesem Projekt alle auf high gesetzt sind. Die ersten 4 Bits der SRAM-Adresse besitzen immer den Wert 0b1101, die ersten 4 Bits der EEPROM-Adresse haben den Wert 0b1010.

In diesem Projekt wird nur die RTC-Funktion, nicht jedoch die EEPROM-Funktion benötigt. Die verwendete Slave-Adresse des Bausteins lautet somit 0b1101111.

11.2.2. Schreibvorgang

Nach der I²C-Startbedingung folgt die Slave-Adresse mit R/$\overline{\text{W}}$-Bit auf 0. Das nächste übertragene Byte stellt die Adresse des ersten zu schreibenden Registers dar. Die weiteren gesendeten Bytes werden der Reihe nach in die Register, beginnend mit dem Register mit der vorher übertragenen Adresse, geschrieben. Abgeschlossen wird die Übertragung durch eine reguläre Stoppbedingung.

11.2.3. Lesevorgang

Zum Lesen muss zuerst genau wie beim Schreibvorgang die Adresse des angesprochenen Registers übertragen werden. Anstatt dann jedoch weitere Nutzdaten zu senden, folgt eine Repeated-Start-Condition mit anschließender Slave-Adresse und R/$\overline{\text{W}}$-Bit auf 1. Nun kann der Mikrocontroller ein oder mehrere Bytes aus dem adressierten und den nachfolgenden Registern lesen. Wie durch das I²C-Protokoll festgelegt, soll der Mikrocontroller das letzte Byte vor der gesendeten Stopbedingung nicht mehr mit einem Acknowledge-Bit bestätigen.

11.2.4. Register

Zur Einstellung der Zeit besitzt der Baustein jeweils Register für Sekunde, Minute, Stunde, Datum, Monat, Jahr und optional den Wochentag. In diesen Registern kann jeweils im BCD-Format die aktuelle Uhrzeit eingestellt werden.

Die Real-Time Clock verfügt über eine Alarm-Funktion, die das Erreichen eines bestimmten Zeitpunktes feststellen und dem Mikrocontroller über eine Interrupt-Leitung signalisieren kann. Dazu sind noch zwei Registersätze mit Sekunden- bis Monatsregister (kein Jahresregister) vorhanden, über welche die zwei Zeitpunkte eingestellt werden können.

Zusätzlich können einige Einstellungen der Real-Time-Clock über verschiedene Konfigurationsbits eingestellt werden, die teilweise in einem eigenen Control-Register und teilweise in freien Bits der Zeit-Register liegen. So können beispielsweise die Alarme und der Batteriebetrieb deaktiviert oder aktiviert und die Uhr zwischen 12- und 24-Stundenmodus umgeschaltet werden.

(vgl. *MCP79410 Battery-Backed Real-Time-Clock Datasheet* 2014)

11.3. Software

11.3.1. Softwareschnittstelle

Die Deklaration der entsprechenden Funktionen erfolgt in der Datei rtc.h:

```
1  void rtc_init(void);
2  void rtc_show_configuration(void);
3  void rtc_load_configuration(void);
4
5  uint8_t rtc_set_time(uint8_t type, uint8_t val);
6  uint8_t rtc_get_time(uint8_t type, uint8_t *val);
7
8  uint8_t rtc_set_alarm0_time(uint8_t type, uint8_t val);
9  uint8_t rtc_get_alarm0_time(uint8_t type, uint8_t *val);
10
11 uint8_t rtc_set_alarm1_time(uint8_t type, uint8_t val);
12 uint8_t rtc_get_alarm1_time(uint8_t type, uint8_t *val);
13
14 uint8_t rtc_start_oscillator(void);
15 uint8_t rtc_stop_oscillator(void);
16
17 uint8_t rtc_enable_24h(void);
18 uint8_t rtc_disable_24h(void);
19
20 uint8_t rtc_enable_battery(void);
21 uint8_t rtc_disable_battery(void);
22
23 uint8_t rtc_get_max(uint8_t type);
24
25 uint8_t rtc_set_alarm0_interrupt_flag(void);
26 uint8_t rtc_clear_alarm0_interrupt_flag(void);
27 uint8_t rtc_get_alarm0_interrupt_flag(uint8_t *flag);
28
29 uint8_t rtc_set_alarm1_interrupt_flag(void);
30 uint8_t rtc_clear_alarm1_interrupt_flag(void);
31 uint8_t rtc_get_alarm1_interrupt_flag(uint8_t *flag);
```

```
32
33 uint8_t rtc_set_alarm_polarity(uint8_t polarity);
34
35 uint8_t rtc_set_alarm0_mask(uint8_t alarm_mask);
36 uint8_t rtc_get_alarm0_mask(uint8_t *alarm_mask);
37
38 uint8_t rtc_set_alarm1_mask(uint8_t alarm_mask);
39 uint8_t rtc_get_alarm1_mask(uint8_t *alarm_mask);
40
41 uint8_t rtc_enable_alarm0(void);
42 uint8_t rtc_disable_alarm0(void);
43
44 uint8_t rtc_disable_alarm1(void);
45 uint8_t rtc_enable_alarm1(void);
46
47 void rtc_enable_avr_interrupt(void);
48 void rtc_disable_avr_interrupt(void);
```

Listing 11.1: Public Functions des RTC-Softwaremoduls

Für das Setzen der aktuellen Zeit und der Alarmzeitpunkte dienen die Funktionen `rtc_set_time` und `rtc_get_time` bzw. `rtc_set_alarm0_time` usw. Als Parameter wird jeweils eine symbolische Konstante wie `RTC_SECOND`, `RTC_MINUTE` usw. für das Setzen des Sekunden- bzw. Minutenregisters angegeben und der Wert auf den die Größe gesetzt werden soll.

Mit der Funktion `rtc_get_max` kann jeweils der Maximalwert für eine bestimmte Zeiteinheit abgerufen werden. Die Funktion liefert beispielsweise für `RTC_SECOND` und `RTC_MINUTE` jeweils 59 und für `RTC_HOUR` 23.

`rtc_enable_avr_interrupt` und `rtc_disable_avr_interrupt` dienen zur Aktivierung bzw. Deaktivierung des Interrupts, der von der Real-Time Clock ausgelöst wird. Der Interrupt muss während I^2C-Übertragungen deaktiviert werden, da er selbst wieder eine I^2C-Übertragung auslöst, die nicht stattfinden kann, wenn der Interrupt die Hauptschleife gerade während einer Übertragung unterbrochen hat. Der Controller würde sich in diesem Fall in der Interruptroutine aufhängen und nicht mehr weiterarbeiten. Sollte eine Interruptanforderung auftreten, während der Interrupt deaktiviet ist, wird diese vom Mikrocontroller gespeichert und sofort ausgeführt, sobald der Interrupt wieder aktiviert wird.

Die restlichen Funktionen dienen zur Einstellung der verschiedenen RTC-Konfigurationen wie das Umschalten zwischen 12- und 24-Stundenmodus, das Löschen und Setzen der Interruptflag der RTC etc.

11.3.2. Implementierung

set_time / get_time - Funktionen:

Die Funktionen zum Setzen der Zeitregister für die aktuelle Uhrzeit und für die Alarmzeitpunkte bauen intern auf der Funktion `rtc_set_time_internal` und `rtc_get_time_internal` auf. Die Funktion hat die Aufgabe die jeweiligen Bits in den Zeitregistern, die nicht zur Zeiteinstellung, sondern für andere Einstellungen zuständig sind, auszusparen und unverändert zu lassen und außerdem den Binärwert für das Register in den BCD-Code umzuwandeln.

Damit die Funktion die Adressen der richtigen Register (für Alarm 0, Alarm 1 oder die Echtzeit) herausfinden kann, wird der Funktion jeweils die Adresse des Sekundenregisters übergeben. Da die Reihenfolge Sekunde, Minute, Stunde usw. bei allen drei Registersätzen gleich ist, kann die Funktion damit die entsprechenden Adressen berechnen.

Zum eigentlichen Schreiben und Lesen der Register werden zwei weitere interne Funktionen `rtc_write_register` und `rtc_read_register` aufgerufen, die die entsprechende I^2C-Kommunikation übernehmen.

rtc_write_register / rtc_read_register:

Aufbauend auf dem TWI-Softwaremodul wird die in den Abschnitten 11.2.2 auf Seite 132 und 11.2.3 auf Seite 132 beschriebene Kommunikation mit der Real Time Clock durchgeführt.

```
1  /**
2   * rtc_write_register() - write a register of real time clock
3   * @address:    address of the real time clock register
4   * @byte:       byte to write into the register
5   *
6   * Returns TWI_OK on success, TWI_ERR on error
7   **/
8  uint8_t rtc_write_register(uint8_t address, uint8_t byte)
9  {
10     uint8_t ret = 0;
11     ret |= twi_start();
12     if (ret != TWI_OK)
13         return ret;
14     ret |= twi_send_slave_address(TWI_WRITE, RTC_ADDRESS);
15     if (ret != TWI_OK)
16         return ret;
17     ret |= twi_send_byte(address);
18     if (ret != TWI_OK)
19         return ret;
20     ret |= twi_send_byte(byte);
21     if (ret != TWI_OK)
22         return ret;
```

```
23      twi_stop();
24      return ret;
25  }
```

Listing 11.2: Beispiel: Funktion `rtc_write_register`

Interrupt Service Routine:

Der RTC-Interrupt-Pin (im Datenblatt der Real Time Clock bezeichnet als MFP - Multi Functional Pin) ist mit dem Interruptpin INT1 des Mikrocontrollers verbunden.

Der erste Alarm wird benützt, um Start- und Stoppzeit der Fuchsjagd abzufangen, der zweite Alarm löst (während der Fuchsjagd) jede Minute einen Interrupt aus, sodass der Fuchs überprüfen kann, ob er in dieser Minute senden soll.

Die entsprechenden zuständigen Softwaremodule startup.h und morse.h benützen dabei jeweils die RTC-Funktionen, um den benötigten Alarm zu stellen. Die ISR selbst befindet sich innerhalb des RTC-Moduls. Dort wird über die Interruptflags der RTC abgefragt, um welchen Alarm es sich gehandelt hat und anschließend die Funktion `startup_interrupt` des Startup-Moduls oder die Funktion `morse_interrupt` des Morse-Moduls aufgerufen.

12. Softwarebibliothek EEPROM

12.1. Anforderungen

Im Projekt kommt ein RFID-Modul zum Einsatz, über das sich Teilnehmer der Fuchsjagd beim Sender registrieren lassen können. So können Bewerber am Ende des Wettbewerbs nachweisen, dass sie alle Sender gefunden haben.

Jeder Teilnehmer bekommt einen RFID-Transponder mit einer eindeutigen Teilnehmernummer. Der Fuchs speichert einerseits direkt auf den Transponder einen Nachweis für die Ankunft beim Fuchs und andererseits führt der Sender auch selbst eine Datenbank, in der alle ankommenden Transponder eingetragen werden. Die Datenbank des Fuchses dient als Referenz bei Problemen wie dem Verlust eines Transponders, einem technischen Defekt oder dem Verdacht auf Manipulation.

Die Informationen müssen auf einem nichtflüchtigen Speicher abgelegt werden, sodass sie auch nach Unterbrechung der Stromversorgung erhalten bleiben. Der Mikrocontroller besitzt bereits einen eingebauten EEPROM, der jedoch größenmäßig auf wenige Kilobytes beschränkt ist und in dem bereits die Konfiguration des Senders abgelegt wird. Für die Benutzerdatenbank ist daher ein 64 kB großer EEPROM des Typs 24AA512 verbaut.

Der Baustein wird über die I^2C-Schnittstelle mit dem Mikrocontroller verbunden. Die Softwarebibliothek soll eine Abstraktion der notwendigen Übertragungen bieten und als Schnittstelle ähnliche Funktionen (z.B. `eeprom_write_byte`, `eeprom_read_byte`) bieten wie die C-Library *AVR Libc* für das interne EEPROM bereitstellt (vgl. *Dokumentation eeprom.h AVR Libc* 2016).

12.2. Überblick EEPROM-Baustein 24AA512

Der Baustein besitzt eine Speicherkapazität von 512 kbit. Somit können insgesamt 64 kB Daten im EEPROM abgelegt werden.

Gelesen und beschrieben werden kann der Chip über die I²C-Schnittstelle. Wie jeder I²C-Slave besitzt auch das EEPROM eine eigene 7-bit Adresse. Die ersten 4 Bits dieser Adresse sind konstant 0b1010, die letzten 3 Bits der Adresse können über drei Pins des EEPROM-Chips eingestellt werden. Alle drei Pins sind auf der Platine mit Masse verbunden, was jeweils zu einer digitalen 0 in der Adresse führt. Die Adresse des Chips lautet somit 0b1010000.

12.2.1. Schreibvorgang

Um ein oder mehrere Bytes in das EEPROM zu schreiben, muss am Beginn die Übertragung durch eine Startbedingung am I²C-Bus eingeleitet werden. Das Byte, welches als Nächstes übertragen wird, enthält in Bit 7 - 1 die 7-bit Adresse und in Bit 0 den Wert 0, um einen Schreibvorgang zu kennzeichnen.

Die nächsten zwei Bytes stellen die Adresse der ersten Speicherstelle im EEPROM dar, in die geschrieben werden soll. Das höherwertige Byte der 16-bit Adresse wird zuerst übertragen. 64 kB entspricht genau 2^{16} Bytes. 16 Bits für die Adresse sind somit ausreichend um jedes einzelne Byte adressieren zu können.

Die nächsten Bytes, die nach der Adresse übertragen werden, stellen die Daten dar, die ins EEPROM an die vorher angegebene Adresse und in die nachfolgenden Speicherstellen geschrieben werden. Möchte man keine weiteren Bytes mehr senden, kann der Schreibvorgang durch eine Stoppbedingung beendet werden.

In einem Schreibvorgang können maximal 128 Bytes geschrieben werden. Wichtig ist auch, dass alle Bytes innerhalb einer der 128 Byte großen *Physical Pages* liegen, da nur jeweils eine Page gleichzeitig geschrieben werden kann (vgl. *24AA512 Serial EEPROM Datasheet* 2010, Seite 10).

Pages beginnen jeweils bei Adressen der Form (n·128) und enden bei (n·128+127), wobei n eine Zahl aus der Menge der natürlichen Zahlen inklusive 0 darstellt. In der Software muss somit entweder darauf geachtet werden, dass nie über das Ende einer Page hinausgeschrieben wird oder es werden überhaupt immer nur einzelne Bytes geschrieben, sodass das Problem nicht auftreten kann.

Nach der Stoppbedingung beginnt der Chip mit dem Schreibvorgang, der bis zu 5 ms in Anspruch nehmen kann. In dieser Zeit antwortet der Baustein nicht mit einem Acknowledge-Bit, wenn man versucht zu lesen oder zu schreiben. Das Ende des Schreibvorgangs kann somit durch häufiges Adressieren des Bausteins herausgefunden werden. Sobald der Chip wieder beginnt, seine Slaveadresse mittels Acknowledge zu bestätigen, ist er bereit, weitere Lese- und Schreibanfragen zu bearbeiten (vgl. *24AA512 Serial EEPROM Datasheet* 2010, Seite 10 ff.).

12.2.2. Lesevorgang

Um ein oder mehrere Bytes vom EEPROM zu lesen, muss zuerst mithilfe eines Schreibzugriffs der Adresscounter auf die richtige Adresse gesetzt werden. Damit keine Daten geschrieben werden, wird der Schreibvorgang nach der Übertragung der Adresse abgebrochen.

Gleich anschließend kann nach einer wiederholten Startbedingung die Slaveadresse des Bausteins mit gesetztem R/$\overline{\text{W}}$-Bit übertragen werden. Nun werden so viele Bytes wie gewünscht gelesen. Der Adresscounter wird vom EEPROM-Baustein automatisch nach jedem gelesenen Byte inkrementiert, sodass die nächste Speicheradresse ausgelesen wird. Nach dem letzten zu lesenden Byte soll der Master keine Acknowledge mehr senden und die Übertragung mit einer Stoppbedingung beenden (vgl. *24AA512 Serial EEPROM Datasheet* 2010, Seite 13 f.).

12.3. Software

12.3.1. Softwareschnittstelle

Die Funktionen, über die der EEPROM-Chip angesteuert werden kann, sind in der Datei ext_eeprom.h deklariert.

```
1  uint8_t ext_eeprom_write_byte(uint16_t addr, uint8_t byte);
2  uint8_t ext_eeprom_write_word(uint16_t addr, uint16_t word);
3  uint8_t ext_eeprom_write_dword(uint16_t addr, uint32_t dword);
4  uint8_t ext_eeprom_write_block(const uint8_t *data, uint16_t addr, uint16_t ↩
       number);
5
6  uint8_t ext_eeprom_read_byte(uint16_t addr, uint8_t *byte);
7  uint8_t ext_eeprom_read_word(uint16_t addr, uint16_t *word);
8  uint8_t ext_eeprom_read_dword(uint16_t addr, uint32_t *dword);
9  uint8_t ext_eeprom_read_block(uint8_t *data, uint16_t addr, uint16_t number);
10
11 uint8_t ext_eeprom_is_ready();
```

Listing 12.1: Public Functions der EEPROM-Softwarebibliothek

Die Funktionen dienen jeweils zum Schreiben bzw. Lesen eines oder mehrerer Bytes. Jede Funktion wartet am Beginn, bis der EEPROM-Baustein bereit ist, falls er sich gerade in einem Schreibvorgang befindet und daher die I^2C-Anfragen nicht beantwortet. Die Funktionen zum Schreiben berücksichtigen auch die EEPROM-Pages und teilen den Schreibvorgang bei Bedarf in mehrere einzelne Zugriffe auf, wenn die zu schreibenden Daten sich über mehrere Pages erstrecken. Da die Funktion auf alle Schreibzugriffe außer auf den letzten warten muss, dauert das Schreiben von Daten, die sich über mehrere Pages erstrecken einige Millisekunden.

Die Schreib- und Lesefunktionen blockieren, wenn das EEPROM sich noch im letzten Schreibvorgang befindet, und warten so lange, bis das EEPROM wieder reagiert. Um ein Blockieren zu verhindern, kann vor Aufruf einer Schreib- oder Lesefunktion mit der Funktion `ext_eeprom_is_ready` abgefragt werden, ob das EEPROM schon wieder bereit ist, eine Anfrage zu erhalten. So können während des Schreibvorgangs des EEPROMs andere Aufgaben erledigt werden.

12.3.2. Implementierung

Zwei interne Funktionen `address_write_prefix` und `address_read_prefix` in der Datei ext_eeprom.c kümmern sich um das korrekte Setzen des Adresspointers für Lese- bzw. Schreibzugriff, sodass die weiteren Funktionen sich anschließend nur mehr um das Senden bzw. Lesen der Nutzdaten kümmern müssen.

Alle öffentlichen Funktionen zum Schreiben und Lesen bauen auf den beiden Funktionen `ext_eeprom_write_block` und `ext_eeprom_read_block` auf.

ext_eeprom_write_block

Die Funktion muss kontrollieren, ob die zu schreibenden Daten in verschiedenen Pages des EEPROMs liegen, weil für jede Page ein eigener Schreibzugriff erforderlich ist. Der eigentliche Schreibvorgang für eine Page erfolgt in der Funktion `ext_eeprom_write_page`.

```
1  /**
2   * ext_eeprom_write_block - write n bytes to eeprom
3   * @data:   pointer to data buffer with data to be written
4   * @addr:   address of first byte written in eeprom
5   * @number: number of bytes to be written
6   *
7   *      If number + addr is bigger than 64kByte ->
8   *      functions starts to write at beginning of eeprom
9   *
10  *      Returns TWI_OK on success and TWI_ERR on failure
11  */
12 uint8_t ext_eeprom_write_block(const uint8_t *data, uint16_t addr, uint16_t ↩
       number)
13 {
14     uint16_t current_addr = addr;
15     uint16_t current_number = 0;
16     uint8_t to_write;
17     uint8_t ret;
18     while (current_number < number) {
19         to_write = EEPROM_PAGE_SIZE - (addr % EEPROM_PAGE_SIZE);
20         if (((uint32_t)current_addr + to_write) > ((uint32_t)addr + number))↩
                {
21             to_write = ((uint32_t)addr + number) - current_addr;
22             /* if not a whole page is to write any more -> make to_write ↩
                   smaller */
```

```
23          }
24          ret = ext_eeprom_write_page(data + current_number, current_addr, ↩
                to_write);
25          if (ret != TWI_OK)  {
26              return ret;
27          }
28          current_number += to_write;
29          current_addr = current_addr + to_write;
30      }
31      return ret;
32  }
```

Listing 12.2: Funktion `ext_eeprom_write_block`

ext_eeprom_write_page

Zu Beginn der Funktion (siehe Listing 12.3 auf der nächsten Seite) wird gewartet bis das EEPROM wieder einsatzbereit ist. Bei der Funktion `ext_eeprom_wait` ist ein Timeout eingestellt, nach dessen Ablauf die Funktion einen Fehler zurückgibt, weil das EEPROM offenbar keinen normalen Schreibvorgang ausführt, sondern ein anderer Defekt vorliegt. So kann verhindert werden, dass die Funktion bei defektem oder fehlendem EEPROM den gesamten Controller blockiert. Diese Funktion ist nur innerhalb der Bibliothek verfügbar und nach außen hin unsichtbar.

Die Real-Time Clock besitzt eine Interruptleitung, die bei bestimmten, eingestellten Zeitpunkten (z.B. zum Starten der Fuchsjagd) einen Interrupt auslöst. In der Interrupt-Service-Routine wird dann begonnen, die Real-Time Clock über I^2C auszulesen. Da es zu Fehlern führen würde, wenn während einer Übertragung zum EEPROM die ISR eine Übertragung zur Real-Time Clock startet, wird der Interrupt deaktiviert. Sollte während der Übertragung tatsächlich ein Interrupt auftreten, so wird die Interruptanforderung im Mikrocontroller zwischengespeichert und der Interrupt erst am Ende der Übetragung zum EEPROM ausgeführt, sobald der Interrupt von der EEPROM-Funktion wieder aktiviert wird.

Es ist nicht notwendig, vor Aufruf der Funktion `ext_eeprom_wait` den Interrupt zu deaktivieren, weil dies nach modulinterner Konvention jeweils die Aufgabe der letzten Funktion ist, die die entsprechende I^2C-Übertragung durchführt. Dadurch wird auch erreicht, dass in der Pause zwischen Ende der Funktion `ext_eeprom_wait` und Fortsetzung der aufrufenden Funktion `ext_eeprom_write_page` der Interrupt kurzzeitig wieder aktiviert ist und dadurch ein gegebenenfalls aufgetretener Real-Time-Clock-Interrupt sofort ausgeführt werden kann, ohne dass eine zu große Verzögerung des Interrupts auftritt.

Nach der Deaktivierung des Interrupts, setzt die Funktion den Adresscounter des EEPROM-Chips mithilfe der Funktion `address_write_prefix`, die auch eine

Startbedingung und die Slaveadresse über I^2C überträgt, auf den richtigen Wert. Anschließend werden der Reihe nach die Bytes dieser Page übertragen.

Die Übertragung endet mit einer Stoppbedingung. Anschließend gibt die Funktion die Erfolgsmeldung `TWI_OK` oder den Fehlerstatus `TWI_ERR` zurück.

Funktionen wie `address_write_prefix` liefern bereits einen Rückgabewert mit `TWI_OK` oder `TWI_ERR` zurück. Dieser Wert wird in der Variablen `ret` gespeichert. Das Makro `RETURN_IF_ERROR` lässt die Funktion mit einer Fehlermeldung beenden, wenn der Rückgabewert der letzten aufgerufenen Funktion nicht `TWI_OK` war und kümmert sich gleichzeitig auch darum, dass nicht vergessen wird, den Real-Time-Clock-Interrupt wieder zu aktivieren. Es ist wichtig, dass der Interrupt auch aktiviert wird, wenn die EEPROM-Funktion fehlschlägt, da ansonsten der Interrupt über längere Zeit deaktiviert bleiben könnte.

Das Macro `RETURN_MACRO` gibt als Rückgabewert für die Funktion den letzten Wert in der Variable `ret` zurück und aktiviert wiederum den Real-Time-Clock-Interrupt.

```
1  /**
2   * ext_eeprom_write_page - write maximum of one page to eeprom
3   * @data:   pointer to data buffer with data to be written
4   * @addr:   address of first byte written in eeprom
5   * @number: number of bytes to be written
6   *
7   *       The addresses addr to (addr + number - 1) should be inside
8   *       a physical page of the eeprom. Otherwise beginning of the page
9   *       will be overwritten!
10  *
11  *       Returns TWI_OK on success, TWI_ERR on failure
12  */
13 static inline uint8_t ext_eeprom_write_page(const uint8_t *data, uint16_t addr,↩
        uint8_t number)
14 {
15     if (ext_eeprom_wait() != TWI_OK)    {
16         return TWI_ERR;
17     }
18
19     rtc_disable_avr_interrupt();
20     uint8_t ret;
21     ret = address_write_prefix(addr);
22     RETURN_IF_ERROR(ret);
23     uint8_t i;
24     for (i = 0; i < number; i++)    {
25         ret = twi_send_byte(data[i]);
26         RETURN_IF_ERROR(ret);
27     }
28     twi_stop();
29     RETURN_MACRO(ret);
30 }
```

Listing 12.3: Funktion `ext_eeprom_write_page`

ext_eeprom_read_block

Bei dieser Funktion ist es nicht erforderlich, dass die Daten in die einzelnen Pages zerlegt werden, weil ein Lesen über den gesamten Bereich hinaus möglich ist.

In den Grundzügen entspricht der Aufbau dem der Funktion `eeprom_write_block`. Zu Beginn wird gewartet, bis das EEPROM einsatzbereit ist. Anschließend wird der Interrupt der Real-Time Clock deaktiviert. Mithilfe der Funktion `address_read_prefix` wird der Adresscounter des EEPROM-Chips gesetzt und die I^2C-Startbedingung gesendet.

In einer Schleife werden im nächsten Schritt alle Bytes bis auf das letzte eingelesen. Beim letzten Byte, das außerhalb der Schleife empfangen wird, wird kein Acknowledge-Bit gesetzt. Der EEPROM-Baustein erkennt daran, dass keine weiteren Bytes mehr erwünscht sind.

Mit der Stoppbedingung wird die Übertragung abgeschlossen.

```
1  /**
2   * ext_eeprom_read_block - read n bytes from eeprom
3   * @data:   pointer to data buffer where the bytes will be stored
4   * @addr:   address of first byte to read
5   * @number: number of bytes to read
6   *
7   *       If number + addr is bigger than 64kByte ->
8   *       Functions starts to read at beginning of eeprom
9   *
10  *       Returns TWI_OK on success and TWI_ERR on failuer
11  */
12 uint8_t ext_eeprom_read_block(const uint8_t *data, uint16_t addr, uint16_t ↩
       number)
13 {
14     if (ext_eeprom_wait() != TWI_OK)    {
15         return TWI_ERR;
16     }
17
18     rtc_disable_avr_interrupt();
19     uint8_t ret;
20     ret = address_read_prefix(addr);
21     RETURN_IF_ERROR(ret);
22     uint8_t i;
23     for (i = 0; i < (number - 1); i++)  {
24         ret = twi_read_byte(TWI_ACK, &data[i]);
25         RETURN_IF_ERROR(ret);
26     }
27     ret = twi_read_byte(TWI_NACK, &data[number - 1]);
28     RETURN_IF_ERROR(ret);
29     twi_stop();
30     RETURN_MACRO(ret);
31 }
```

Listing 12.4: Funktion `ext_eeprom_read_block`

ext_eeprom_is_ready

Die Funktion `ext_eeprom_is_ready` dient zur Abfrage, ob der EEPROM-Baustein bereits einsatzbereit ist oder ob der Baustein noch mit dem letzten Schreibvorgang beschäftigt ist.

Die Funktion versucht, ein Byte aus dem EEPROM zu lesen. Wenn der EEPROM sich noch im Schreibzugriff befindet, wird die aufgerufene Funktion `address_read_prefix` scheitern, weil der Baustein seine Slaveadresse nicht mit einem Acknowledge-Bit bestätigt. Die Funktion gibt anschließend `TRUE` bzw. `FALSE` zurück, je nachdem, ob der EEPROM-Baustein reagiert hat oder nicht.

```
1  /**
2   * ext_eeprom_is_ready - checks if eeprom is ready to send or receive new data
3   *
4   *       During write progress eeprom does not respond to request. This
5   *       functions returns whether eeprom is responding or not.
6   *
7   *       Returns TRUE if eeprom is ready, FALSE otherwise
8   */
9  uint8_t ext_eeprom_is_ready()
10 {
11     uint8_t temp;
12     uint8_t ret;
13
14     rtc_disable_avr_interrupt();
15
16     ret = address_read_prefix(0x00);
17     RETURN_IF_ERROR(ret);
18
19     ret = twi_read_byte(TWI_NACK, &temp);
20     RETURN_IF_ERROR(ret);
21
22     twi_stop();
23
24     RETURN_MACRO(ret);
25
26 }
```

Listing 12.5: Funktion `ext_eeprom_is_ready`

Alle weiteren Funktionen wie `ext_eeprom_write_byte`, `ext_eeprom_read_byte` etc. bauen auf den beiden Funktionen für das Lesen oder Schreiben eines ganzen Blockes auf, indem diese Funktionen mit den entsprechenden Parametern aufgerufen werden.

13. Softwaremodul UART

13.1. Anforderungen

Der Sender soll vom PC aus konfiguriert werden. Diese Kommunikation erfolgt über die USB-Schnittstelle. Da der Mikrocontroller ATmega644 keine direkte USB-Kommunikation ermöglicht, kommt der USB-UART-Wandler FT232 der Firma FTDI zum Einsatz.

Auf der PC-Seite erscheint das USB-Gerät als COM-Schnittstelle, die von allen Programmen angesprochen werden kann, die auf die RS232-Schnittstelle zugreifen können.

Der Mikrocontroller bekommt die Daten direkt über die UART-Schnittstelle geliefert und weiß nicht, dass die Kommunikation zwischendurch über die USB-Schnittstelle geleitet wird.

Die Konfiguration des Senders soll über ein textbasiertes Command-Line-Interface möglich sein, sodass ein simples Terminalprogramm am PC ausreicht und nicht zwingend eine zusätzliche PC-Software erforderlich ist.

Das UART-Softwaremodul muss somit Möglichkeiten bieten, Text über die UART-Schnittstelle zu senden und zu empfangen. Da der Mikrocontroller empfangene Daten nicht immer unmittelbar abholen kann, muss das Softwaremodul einen Empfangsbuffer bieten, indem Daten zwischengespeichert werden. Beim Senden hat es Vorteile, wenn ebenfalls ein Buffer verwendet wird, da der aufrufende Programmteil dann nicht bis zum Abschluss des Sendevorgangs blockiert werden muss.

Optimalerweise soll das Modul auch die Möglichkeit bieten, Text aus den verschiedenen Speichermöglichkeiten des Mikrocontrollers wie RAM, FLASH oder EEPROM zu senden.

13.2. UART beim AVR

Zur Konfiguration der UART-Schnittstelle am AVR sind vor allem 5 Register von Bedeutung:

- UCSR0A - USART Control and Status Register A
- UCSR0B - USART Control and Status Register B
- UBRR0L - USART Baud Rate Register LSB
- UBRR0H - USART Baud Rate Register MSB
- UDR0 - USART I/O Data Register

UCSR0A-Register

U2X0: Über das U2X0-Bit kann die UART-Geschwindigkeit verdoppelt werden. Dadurch ist es in einigen Fällen möglich, die Abweichung zwischen der erreichbaren und der gewünschten Baudrate zu verkleinern.

UCSR0B-Register

TXEN0: Das Transmitter-Enable-Bit muss gesetzt werden, um die Sendeeinheit der UART-Schnittstelle zu aktivieren.

RXEN0: Das Receiver-Enable-Bit muss gesetzt werden, um die Empfangseinheit der UART-Schnittstelle zu aktivieren.

TXCIE0: Mit diesem Bit kann der Transmit-Complete-Interrupt aktiviert werden, der ausgelöst wird, sobald ein Byte über die UART-Schnittstelle fertig gesendet worden ist. Für das UART-Softwaremodul ist dieser Interrupt wichtig, da immer wenn ein Byte übertragen worden ist, das Softwaremodul das nächste Byte aus dem Sendebuffer lesen und übertragen soll.

RXCIE0: Mit diesem Bit kann der Empfangsinterrupt aktiviert werden, der nach jedem Empfang eines Bytes über UART ausgelöst wird. In diesem Projekt wird der Interrupt benützt, um das empfangene Byte auszulesen und in den Empfangsbuffer zu schreiben. Der unmittelbare Interrupt verhindert, dass das empfangene Byte bereits durch das nächste angekommene Byte überschrieben wird, bevor es in den Empfangsbuffer geschrieben worden ist.

UBBR0 - Register

Die beiden Register UBRR0L und UBRR0H stellen zusammen eine 16-bit Zahl dar (mit UBRR0L als niederwertigeres Byte), von der Bit 11 bis 0 zur Einstellung der Baudrate verwendet werden.

Der Wert für das Baudrate-Register kann nach folgender Formel berechnet werden (vgl. *ATmega 644 Microcontroller Datasheet* 2012, Seite 167):

$$\text{UBRR0} = \frac{f_{\text{CPU}}}{8 \cdot \text{BAUD}} - 1 \tag{13.1}$$

UBRR0	...	Wert zusammengesetzt aus UBRR0L + UBRR0H $\cdot 2^8$
BAUD	...	Die UART-Baudrate in Symbole/s
f_{CPU}	...	Taktfrequenz des Mikrocontrollers (hier 8 MHz)

UDR0 - Register

Das UART-Datenregister wird zum Senden und Empfangen der Datenbytes benötigt. Ein Sendevorgang wird gestartet, indem ein Byte in das UDR0-Register geschrieben wird.

Nach dem erfolgreichen Empfang eines Bytes kann das Byte aus dem UDR0-Register gelesen werden.

13.3. Software

13.3.1. Softwareschnittstelle

Die Funktionen sind in der Datei uart.h deklariert:

```
1  void uart_init(void);
2
3  void uart_send_text_sram(const char *text);
4  void uart_send_binary_sram(uint8_t number, uint8_t *data);
5
6  void uart_send_text_flash(uint16_t text);
7  void uart_send_binary_flash(uint8_t number, uint16_t data);
8
9  void uart_send_text_eeprom(const char *text);
10 void uart_send_binary_eeprom(uint8_t number, uint8_t *data);
11
12 void uart_send_text_buffer(const char *text);
```

```
13 void uart_send_binary_buffer(uint8_t number, uint8_t *data);
14
15 uint8_t uart_receive_buffer_text(char **text);
16
17 uint8_t uart_send_int(uint32_t val);
18 uint8_t uart_send_int_hex(uint32_t val);
```

Listing 13.1: Public Functions des UART-Softwaremoduls

Die Funktionen zum Senden von ASCII-Text (jeweils beginnend mit `uart_send_text`) übernehmen als Parameter jeweils die Adresse des erstens Zeichens des Texts. Nachfolgende Zeichen werden unabhängig vom Wert so weit ausgegeben, bis die erste abschließende binäre 0 auftritt.

Zusätzlich werden Funktionen zum Senden von Binärdaten (benannt mit `uart_send_binary`) bereitgestellt, die als Parameter die Adresse des ersten Datenbytes und die Anzahl an Bytes erhält, die übertragen werden sollen.

Je nach dem, ob die zu sendenden Daten im RAM, im Programmspeicher oder im EEPROM liegen, müssen unterschiedliche Funktionen verwendet werden, die auf `_sram`, `_flash` bzw. `_eeprom` enden.

Ein Problem der SRAM-Funktionen ist, dass die Daten bis zum Senden noch an dieser Speicherstelle vorhanden sein müssen, da bei Aufruf der entsprechenden Funktion nur der Pointer und nicht die eigentlichen Daten in den Sendebuffer geschrieben werden, um Speicherplatz zu sparen. Befinden sich die Daten jedoch in lokalen Variablen einer Funktion, so kann es vorkommen, dass diese lokalen Variablen bereits zerstört oder durch neue ersetzt worden sind, bis die Daten tatsächlich übertragen werden.

Als Abhilfe dienen die Funktionen `uart_send_text_buffer` und `uart_send_binary_buffer`, welche die gesamten Daten in einen internen Buffer des UART-Softwaremoduls kopieren, wodurch die Länge der Binärdaten bzw. des Texts jedoch auf die Länge des einzelnen Buffers begrenzt ist (siehe Konstante `UART_SEND_BUFFER_LENGTH` in der Datei uart.c)

Somit können mithilfe der SRAM-Funktionen größere Datenmengen aus dem RAM übertragen werden, die jedoch konstant sein müssen. Kleinere Datenmengen, die noch dazu in lokalen Variablen liegen oder sich öfter ändern, werden stattdessen über die Buffer-Funktionen übertragen.

Die Funktionen `uart_send_int` bzw. `uart_send_int_hex` ermöglichen die Umwandlung und Übertragung eines Integer-Werts als Text. Hierbei wird intern die Umwandlung der Zahl in Text vorgenommen. Der umgewandelte Text wird wiederum im internen Buffer abgelegt.

Zum Auslesen der Daten dient die Funktion `uart_receive_buffer_text`. Die Funktion liefert einen Rückgabewert, der signalisiert, ob neue Daten vorhanden sind. Wenn neue Daten vorhanden sind, wird in den Zeiger, dessen Adresse als Parameter übergeben worden ist (Übergabe als Zeiger auf einen Zeiger), die Adresse der empfangenen Daten eingetragen.

Daten werden über die UART-Schnittstelle zeilenweise eingelesen. Die Funktion `uart_receive_buffer_text` liefert erst neue Daten zurück, wenn bereits ein Zeilenende (entweder `CR '\r'`, `LF '\n'` oder `CRLF '\r\n'`) empfangen worden ist. Dies bedeutet, dass es nicht möglich ist, Binärdaten über die Schnittstelle zu empfangen, außer es folgen regelmäßig Zeilenumbrüche, um die Übertragung abzuschließen. In diesem Projekt basiert die Konfiguration auf einer ASCII-Kommandozeile, daher müssen keine Binärdaten empfangen werden.

13.3.2. Implementierung

Senden von Daten

Das Modul verfügt als Sendebuffer über ein Datenarray, in dem eine festgelegte Anzahl von Einträgen Platz findet. Jeder Eintrag im Buffer enthält den Typ der Daten (binär oder Text), den Speicherort (SRAM, Flash, EEPROM) und die Adresse auf den Anfang der Daten im jeweiligen Medium (bzw. die Daten selbst, wenn die `uart_send_text_buffer-` oder `uart_send_binary_buffer-` Funktion aufgerufen worden ist).

Beim Aufruf einer Sendefunktion wird jeweils der richtige Eintrag im Sendebuffer erstellt und der Sendevorgang begonnen, außer der Controller befindet sich bereits im Sendevorgang.

Der Sendevorgang kann durch Aufruf der internen Funktion `uart_send_buffer_-cont` (*cont* für continue) aufgerufen werden. Diese Funktion kann anhand der Einträge im Sendebuffer das nächste zu sendende Byte feststellen, es aus dem SRAM, dem EEPROM, dem Flashspeicher oder dem internen Buffer auslesen und über UART senden.

Sobald der Sendevorgang einmal begonnen worden ist, wird der Buffer im Laufe der Zeit automatisch abgearbeitet, da jeweils nach dem Senden eines Bytes der UART-Transmit-Complete-Interrupt aufgerufen wird, der wiederum die Funktion `uart_send_buffer_cont` aufruft und somit das Senden des nächsten Bytes einleitet.

Empfangen von Daten

Jedes empfangene Byte über die UART-Schnittstelle löst den UART-Receive-Complete-Interrupt aus. In der entsprechenden Interruptroutine wird das Byte aus dem UDR0-Register ausgelesen und in den nächsten freien Speicherplatz im Empfangsbuffer geschrieben.

Der Empfangsbuffer besteht aus einem zweidimensionalen Char-Array. Die erste Array-Ebene dient zum Speichern der Zeichen einer Zeile hintereinander, die zweite Ebene liefert mehrere Speicherplätze für je eine einzelne Zeile.

Sobald ein Zeilenende erkannt wird, wird auf den nächsten Zeilenbuffer umgeschaltet (also der Arrayindex in der zweiten Ebene inkrementiert). Der gerade geschriebene Zeilenbuffer wird zurückgegeben, wenn das nächste Mal eine Abfrage mit `uart_receive_buffer_text` erfolgt.

14. Softwaremodul Commands

14.1. Anforderungen

Zur Konfiguration des Senders über die UART-Schnittstelle soll eine textbasierte Kommandozeilenschnittstelle implementiert werden. Das Softwaremodul Commands wird die Interaktion mit dem Benutzer übernehmen. Dazu sollen beispielsweise ein Begrüßungstext am Anfang und jeweils ein Kommandozeilenprompt nach jedem Befehl ausgegeben werden.

Das Softwaremodul hat die Aufgabe, die empfangenen Daten vom UART Softwaremodul einzulesen und die eingegebenen Befehle zu interpretieren und auszuführen.

14.1.1. Kommandos zur Konfiguration

Jedes Kommando folgt in einer eigenen Zeile. Ein Kommando beginnt mit dem Kommandonamen selbst, gefolgt von einem oder mehreren Leerzeichen und je nach Kommando einem verpflichtenden oder optionalen Parameter. Die gesamte Zeile muss mit dem Zeilenende CR ('\r'), LF ('\n') oder CRLF ('\r\n') abgeschlossen werden.

Ein Beispiel ist das `set frequency`-Kommando zum Einstellen der Ausgangssignalfrequenz. Als Parameter muss die Frequenz in Hertz angegeben werden. Um die Frequenz auf 3.5 MHz zu setzen, muss über UART somit folgende Zeile übertragen werden:

```
set frequency 3500000
```

Eine Liste aller Kommandos befindet sich in der Datei commands.c. Dort kann auch eine detaillierte Beschreibung zu jedem Befehl abgerufen werden.

14.2. Softwareschnittstelle

Die nach außen hin verfügbaren Funktionen des Moduls sind in der Datei commands.h deklariert:

```
1 uint8_t commands_execute(void);
2 void commands_print_welcome_message(void);
3 void commands_print_prompt(void);
```

Listing 14.1: Public Functions des Commands-Softwaremoduls

Die Hauptschleife ruft regelmäßig die Funktion `commands_execute` auf. Die Funktion überprüft anschließend, ob neue Daten über die UART-Schnittstelle angekommen sind, und holt diese gegebenenfalls zur Verarbeitung ab.

Durch Aufruf der Funktion `commands_print_welcome_message` wird über die UART-Schnittstelle der Begrüßungstext ausgegeben, der beim Starten des Geräts Informationen über das Gerät, die Firmware, die Lizenz und einen Link zu einer Website mit weiterführenden Informationen liefert.

14.3. Implementierung

14.3.1. commands_execute:

Einlesen der Kommandos

Zentral ist die Funktion `commands_execute`, die regelmäßig von der Hauptschleife aus aufgerufen wird. Zu Beginn fragt die Funktion vom UART-Softwaremodul durch Aufruf von `uart_receive_buffer_text` die nächste empfangene Zeile über UART ab.

Ist eine neue Zeile vorhanden, so wird verglichen, ob der Anfang der eingelesenen Zeile mit einem Kommando aus der im Flash abgelegten Kommandoliste übereinstimmt. Sobald ein Kommando erkannt worden ist, wird die jeweilige interne Funktion zum Setzen der entsprechenden Einstellungen aufgerufen. Diese Funktionen sind nach dem Schema `execute_kommandoname` benannt (z.B. `execute_set_frequency` für das Kommando `set frequency`).

Extrahieren der Parameter

Da einige Funktionen einen Parameter erwarten, wird mithilfe der internen Funktion `command_get_parameter` ein Pointer auf den Anfang des Parameters in der über UART empfangenen Zeile gesetzt. Besitzt die Zeile keinen Parameter, so wird der Pointer an die Stelle mit der abschließenden binären 0 gesetzt und signalisiert somit das Fehlen eines Parameters.

Dieser Pointer auf den Parameter wird jeder `execute`-Funktion unabhängig davon übergeben, ob das entsprechende Kommando tatsächlich einen Parameter erwartet oder nicht.

15. Softwarebibliothek Utils

15.1. Zweck

In der Firmware ist es an verschiedenen Stellen notwendig, Aufgaben wie das Vergleichen von Text, die Umwandlung von Zahlen in Text bzw. von Text in Zahlen auszuführen. Einige dieser Aufgaben könnten auch mit Funktionen erledigt werden, die von der C-Standardbibliothek mitgeliefert werden.

Da jedoch einige Funktionen wie `snprintf`, die zum Umwandeln von Zahlen in Text verwendet werden können, am ATmega-Mikrocontroller relativ viele Ressourcen in Anspruch nehmen, wird bei diesem Projekt auf die Verwendung der Standardbibliothek verzichtet. Die benötigten Funktionen werden stattdessen eigenhändig implementiert und befinden sich in den Dateien utils.c bzw. utils.h.

15.2. Aufgaben

15.2.1. String zu Integer Konvertierung

Die Funktion `str_to_int` übernimmt als Parameter den Zeiger auf ein Char-Array mit dem umzuwandelnden String und einen Pointer auf eine Integer Variable, in die der umgewandelte Wert geschrieben wird.

Der String darf nur Ziffern enthalten und die umzuwandelnde Zahl muss positiv sein. Als Rückgabewert liefert die Funktion `TRUE`, wenn die Konvertierung erfolgreich ist und `FALSE`, wenn andere Zeichen als Ziffern im String gefunden werden oder wenn die Zahl im String zu groß für die Integer-Variable ist.

15.2.2. Integer zu String Konvertierung

int_to_string:

Der Funktion `int_to_string` muss ein Pointer auf ein Char-Array gemeinsam mit der maximalen Länge des Char-Arrays und dem umzuwandelnden Integer-Wert übergeben werden.

Die Zahl wird als Dezimalzahl in das Char-Array geschrieben. Sollte das übergebene Char-Array zu klein für die umgewandelte Zahl sein, wird `FALSE` zurückgegeben. Andernfalls erfolgt die Bestätigung der erfolgreichen Umwandlung durch Rückgabe von `TRUE`.

Der Text wird am Ende mit einer binären 0 abgeschlossen, die zusätzlich im Char-Array Platz haben muss.

int_to_string_hex:

Die Funktion arbeitet gleich wie `int_to_string`, jedoch wird die Zahl als Hexadezimalzahl in das Array geschrieben. Im Char-Array stehen nur die hexadezimalen Ziffern, das übliche Prefix `0x` muss bei Bedarf selbst hinzugefügt werden.

int_to_string_fixed_length:

Die Parameter sind identisch zur Funktion `int_to_string`. Diese Funktion füllt allerdings den vefügbaren Platz im Char-Array völlig aus, indem die führenden Stellen gegebenenfalls mit ASCII-Nullen ausgefüllt werden.

Soll beispielsweise eine Zahl mit konstant zwei ASCII-Stellen ausgegeben werden, muss als Länge des Char-Arrays 3 angegeben werden. Die dritte Stelle wird benötigt, um die abschließende binäre 0 zu speichern.

15.2.3. Vergleichen von Strings

Zum Einlesen der Kommandos über die UART-Schnittstelle ist es notwendig, den Text mit den Befehlen, die im Flashspeicher abgelegt sind, zu vergleichen.

str_compare_progmem:

Die Funktion übernimmt als ersten Parameter einen Pointer auf ein Char-Array im RAM und als zweiten die Adresse eines Char-Arrays im Flashspeicher. Die beiden Strings werden Buchstabe für Buchstabe verglichen.

Zurückgegeben wird `UTILS_STR_EQUAL`, falls die beiden Strings gleich lang und identisch sind, `UTILS_STR_CONTAINS`, falls der String im RAM länger als der String im Flashspeicher, jedoch bis zum Ende des Strings mit diesem ident ist. Sind die beiden Strings unterschiedlich, wird `UTILS_STR_FALSE` zurückgegeben.

15.2.4. Umwandlung von Kleinbuchstaben in Großbuchstaben

Mithilfe der Funktion `to_upper_case` können im übergebenen Char-Array alle Kleinbuchstaben in Großbuchstaben umgewandelt werden. Dadurch kann erreicht werden, dass beispielsweise das Rufzeichen über die UART-Schnittstelle unabhängig von der Klein- oder Großschreibung erkannt wird.

16. Softwaremodul Startup

16.1. Anforderungen

Die Sender werden üblicherweise im Einsatzgebiet bereits einige Zeit vor Beginn eines Wettbewerbs verteilt. In der Zeit, bevor der Wettbewerb beginnt, soll der Fuchs sich in einem möglichst stromsparenden Modus befinden. Senderendstufe, RFID-Modul und alle während des Wettbewerbs benötigten Bausteine sollen erst beim Erreichen der Startzeit aktiviert werden.

Das Startup-Softwaremodul soll eine Schnittstelle bieten, über die Start- und Stoppzeit des Wettbewerbs eingestellt werden können. Sobald die Start- bzw. Stoppzeit erreicht ist, sollen verschiedene Funktionen in anderen Modulen aufgerufen werden, die die Aktivierung bzw. Deaktivierung der entsprechenden Komponenten übernehmen.

16.2. Softwareschnittstelle

Die verfügbaren Funktionen sind in der Datei startup.h deklariert.

```
1  void startup_init(void);
2  void startup_show_configuration(void);
3  void startup_load_configuration(void);
4
5  uint8_t startup_set_start_time(uint8_t type, uint8_t time);
6  uint8_t startup_set_stop_time(uint8_t type, uint8_t time);
7  uint8_t startup_get_start_time(uint8_t type);
8  uint8_t startup_get_stop_time(uint8_t type);
9
10 uint8_t startup_interrupt(void);
```

Listing 16.1: Public Functions des Startup-Moduls

Die Funktionen zum Setzen der Zeit hier ähneln den im RTC-Modul verwendeten Funktionen für diese Aufgabe. Als erster Parameter wird wiederum der entsprechende Typ entweder als `STARTUP_SECOND`, `STARTUP_MINUTE` etc. oder als

RTC_SECOND, RTC_MINUTE etc. übergeben. Als zweiter Parameter folgt der zu setzende Wert.

Die Funktionen zum Auslesen der Start- oder Stoppzeit liefern im Gegensatz zu den entsprechenden RTC-Funktionen den Wert als Rückgabewert der Funktion zurück. Bei den RTC-Funktionen ist der Rückgabewert ein Indikator über die erfolgreiche Kommunikation mit dem RTC-Baustein. Die Rückgabe des eigentlichen Werts erfolgt dort mithilfe eines als Parameter übergebenen Pointers.

Um das Erreichen der Start- oder Stoppzeit festzustellen, wird auf den von der Real-Time Clock angebotenen Alarm-Interrupt zurückgegriffen. Beim Initialisieren des Startup-Softwaremoduls (durch startup_init) wird der Alarm der Real-Time Clock durch Zuhilfenahme der Funktionen des RTC-Softwaremoduls auf die richtige Uhrzeit gesetzt. Sobald die eingestellte Uhrzeit erreicht ist, wird beim Mikrocontroller ein Interrupt ausgelöst. Die entsprechende Interruptroutine befindet sich im Softwaremodul der Real-Time Clock, da noch ein weiterer Alarm über den gleichen Interrupt verwaltet werden muss. Das RTC-Modul ruft nach dem Auftreten des Interrupts die Funktion startup_interrupt auf, die anschließend das Aktivieren oder Deaktivieren der Fuchskomponenten initiiert.

16.3. Implementierung

16.3.1. Interner EEPROM-Speicher

Start- und Stoppzeit werden während der Konfiguration des Fuchses über den PC eingestellt und müssen auch nach Unterbrechung der Versorgungsspannung gespeichert bleiben. Aus diesem Grund werden die Daten im internen EEPROM des Mikrocontrollers abgelegt.

Die C-Library des AVRs bietet bereits Funktionen an, mit denen der interne EEPROM gelesen und beschrieben werden kann. Diesen Funktionen muss jeweils die Speicheradresse des Speicherplatzes im EEPROM übergeben werden.

Da mehrere Module der Firmware auf das interne EEPROM zugreifen und dort Daten ablegen, müssten alle Module sich über die jeweils verwendeten Adressen einig werden, damit keine Speicheradresse doppelte Verwendung findet.

Das AVR-GCC-System bietet hierfür eine komfortable Lösung. Variablen können mit dem Schlüsselwort EEMEM versehen werden, das den Compiler anweist, im EEPROM einen Speicherplatz für diese Variable zu reservieren. Der Linker

kümmert sich schlussendlich darum, dass jede EEPROM-Variable auch eine einzigartige Speicheradresse bekommt, die von keiner anderen EEPROM-Variable verwendet wird. Die endgültige Adresse steht somit erst nach dem Linken fest.

Die Variablen für die Start- und Stoppzeit im EEPROM werden daher als Array wie folgt definiert:

```
uint8_t EEMEM start_time[STARTUP_TIME_MAX + 1];
uint8_t EEMEM stop_time[STARTUP_TIME_MAX + 1];
```

Die Konstante `STARTUP_TIME_MAX` enthält den größten Index, über den auf das Array zugegriffen wird, und hilft somit bei der Feststellung der Arraylänge. Da in der Programmiersprache C der Arrayindex bei 0 beginnt, ist die Länge des Arrays um 1 größer als der maximale Index. Bei der Arraydeklaration ist die Größe des Arrays und nicht der maximale Index anzugeben. Somit muss zum maximalen Index `STARTUP_TIME_MAX` noch 1 addiert werden.

Zugriff auf EEPROM-Variablen

Auf diese EEPROM-Variablen kann nicht einfach auf die gleiche Weise zugegriffen werden wie auf Variablen, die im RAM liegen. Folgende Zeile würde dazu führen, dass die Adresse des C-Symbols `start_time`, die für einen EEPROM-Speicherplatz gültig ist, als Adresse im RAM interpretiert werden und somit das Byte 0x12 an irgendeine beliebige Adresse im RAM geschrieben werden würde:

```
start_time[0] = 0x12;
```

Die korrkete Vorgangsweise, um in das EEPROM zu schreiben, enthält einen Aufruf der Funktion `eeprom_write_byte`:

```
eeprom_write_byte(&start_time[0], 0x12);
```

Der Funktion `eeprom_write_byte` muss als erster Parameter eine Adresse übergeben werden. Um die Adresse des Speicherplatzes `start_time[0]` zu erhalten, kann der von gewöhnlichen RAM-Variablen bekannte Adressoperator `&` verwendet werden. Als zweiter Parameter wird jener Wert übergeben, der in die entsprechende Speicherstelle im EEPROM geschrieben werden soll.

16.3.2. Implementierung der Funktionen

startup_set_start_time / startup_set_stop_time:

Die beiden Funktionen dienen nur dazu, den übergebenen Wert auf Zulässigkeit zu prüfen und in die entsprechende EEPROM-Variable zu schreiben.

```
/**
 * startup_set_start_time - save time of competition start in eeprom
 * @type: defines the type of the time element (STARTUP_SECOND, etc.)
 * @time: the value to save
 *      Return: TRUE if time value was in correct range, FALSE otherwise
 */
uint8_t startup_set_start_time(uint8_t type, uint8_t time)
{
    if (type > STARTUP_TIME_MAX)    {
        return FALSE;
    }
    if (time > type_max[type])  {
        return FALSE;
    }
    if (type == STARTUP_DATE || type == STARTUP_MONTH)  {
        if (time == 0)  {
            return FALSE;
        }
    }
    eeprom_write_byte(&start_time[type], time);
    return TRUE;
}
```

Listing 16.2: Funktion `startup_set_start_time`

startup_get_start_time / startup_get_stop_time:

Die Funktionen zum Auslesen der Zeit überprüfen neben dem Lesen aus dem EEPROM auch, ob der gelesene Wert gültig sein kann. Beispielsweise kann der Wert 13 für `STARTUP_MONTH` nicht sinnvoll sein und wird deshalb auf den Höchstwert 12 begrenzt. Die Höchstwerte sind im Quellcode im Array `type_max` enthalten. Da nicht alle Monate gleich viele Tage lang sind, wird allgemein auf den höchstmöglichen Wert 31 begrenzt.

```
/**
 * startup_get_start_time - read time when competition ends
 * @type: type of time information like STARTUP_SECOND etc.
 *      Returns value of start time defined by type (second, hour, date etc.)
 */
uint8_t startup_get_start_time(uint8_t type)
{
    uint8_t byte = eeprom_read_byte(&start_time[type]);
    if (byte > type_max[type])  {
        byte = type_max[type];
    }
    if (type == STARTUP_MONTH)  {
        if (byte == 0){
            byte = 1;
        }
    }
    return byte;
}
```

Listing 16.3: Beispiel Funktion `startup_get_start_time`

is_time_between_start_and_stop:

Beim Aufruf der Funktion `startup_interrupt` durch das RTC-Modul und beim Aufruf der Funktion `startup_init` wird mithilfe der internen Funktion `is_time_between_start_and_stop` durch Auslesen der Zeit aus der Real Time Clock und durch Vergleich mit der gespeicherten Start- und Stoppzeit festgestellt, ob der jetzige Zeitpunkt zwischen Start- und Stoppzeit liegt oder nicht. Anhand dieser Aussage können schließlich die zusätzlichen Komponenten aktiviert oder deaktiviert werden.

Dadurch werden Fehler verhindert, falls der RTC-Interrupt aus irgendeinem Grund zu einem falschen Zeitpunkt auftritt. Außerdem kann durch einen Aufruf der Funktion sichergestellt werden, dass der Sender zu senden beginnt, wenn er innerhalb der Startzeit eingeschaltet wird.

startup_interrupt:

Diese Funktion wird vom Softwaremodul der Real-Time Clock aufgerufen, sobald der vom Startup-Modul eingestellte Alarm aufgetreten ist, der signalisiert, dass nun die Start- bzw. Stoppzeit erreicht ist.

Die Funktion ruft die interne Funktion `check_start_time` auf, in der noch einmal überprüft wird, ob sich der Sender zwischen Start- und Stoppzeit befindet oder nicht. Anschließend werden das Morse-, das User- und das Main-Modul vom Anfang bzw. Ende des Wettbewerbs durch Aufruf der Funktion `morse_start_time` bzw. `morse_stop_time` etc. informiert, damit sie die entsprechenden Schritte wie beispielsweise die Aktivierung bzw. Deaktivierung des RFID-Moduls, des DDS-Bausteins usw. einleiten.

17. Softwaremodul Morsen

17.1. Anforderungen

Das Ziel einer Fuchsjagd ist es, alle verteilten Sender mithilfe eines Peilers aufzuspüren. Gewöhnlich werden Wettbewerbe mit fünf Sendern ausgetragen, wobei der erste Sender in der ersten Minute sendet, der zweite in der zweiten usw. bis sich der Vorgang nach fünf Minuten wiederholt.

Jeder Fuchs bekommt ein eigenes Rufzeichen, das er als Morsesignal aussendet. Der *erste* Fuchs erhält dabei gewöhnlich das Rufzeichen MOE (zweimal lang, dreimal lang, *einmal* kurz), der *zweite* Fuchs das Rufzeichen MOI (zweimal lang, dreimal lang, *zweimal* kurz), der dritte MOS, der vierte MOH und der *fünfte* MO5 (zweimal lang, dreimal lang, *fünfmal* kurz).

Das Morse-Softwaremodul ist somit einerseits verantwortlich, in der richtigen Minute mit dem Senden zu beginnen und eine Minute später wieder aufzuhören, um die Aussendung anderer Sender nicht zu stören, und andererseits muss es die Morsezeichen des Rufzeichens generieren. Dazu muss der DDS im richtigen Takt aktiviert und deaktiviert werden.

17.2. Überblick Morsezeichen

Mithilfe des Morsecodes können Buchstaben und Zahlen durch ein Signal, das nur zwei Zustände benötigt, übertragen werden. Beim Sender werden diese beiden Zustände durch *Sender an* und *Sender aus* realisiert.

Jedes Zeichen setzt sich aus einer Kombination von ein oder mehreren kurzen oder langen Ein-Phasen zusammen. Die kurzen Phasen bezeichnet man als Punkt oder gesprochen *dit*, die langen als Strich oder gesprochen *dah*.

Dabei besteht folgendes Verhältnis zwischen der Zeitdauer der einzelnen Elemente:

- Ein Strich ist dreimal so lang wie ein Punkt.

- Die Pause zwischen Strichen und Punkten innerhalb eines Zeichens ist gleich lang wie ein Punkt.
- Die Pause zwischen dem Ende des letzten Punkts oder Strichs eines Zeichens und dem Anfang des nächsten Zeichens ist gleich lang wie ein Strich.
- Die Pause zwischen zwei Zeichen eines Wortes ist gleich lang wie 7 Punkte.

(vgl. Wikipedia 2016b)

17.2.1. Wörter pro Minute - WPM

Die Angabe der Geschwindigkeit, in der gemorst wird, erfolgt in Wörtern pro Minute. Da nicht alle Wörter gleich lang sind, wird für die Einheit Wörter pro Minute mit fünf Buchstaben pro Wort gerechnet. Da jedoch auch nicht jeder Buchstabe gleich lange dauert, ist es gängig, als Referenzwort *PARIS* zu verwenden.

Das Wort Paris im Morsecode ist insgesamt so lange wie 50 Dits. Durch Multiplikation der WPM mit 50 erhält man somit die Anzahl der Dits pro Minute. Durch Kehrwertbildung kann berechnet werden, wie lange ein Dit dauert. In der Software wird die Länge der Dits (als größter gemeinsamer Teiler aller benötigten Längen) als Basiseinheit verwendet.

17.3. Softwareschnittstelle

In der Datei morse.h sind die Funktionen zur öffentlichen Verwendung deklariert:

```
1  void morse_init(void);
2  void morse_show_configuration(void);
3  void morse_load_configuration(void);
4
5  uint16_t morse_convert_wpm_morse_unit(uint16_t wpm);
6  uint16_t morse_convert_morse_unit_wpm(uint16_t morse_unit);
7
8  void morse_set_morse_unit(uint16_t morse_unit_value);
9  uint16_t morse_get_morse_unit(void);
10
11 uint8_t morse_set_fox_number(uint8_t fox_number);
12 uint8_t morse_get_fox_number(void);
13
14 uint8_t morse_get_fox_max(void);
15 uint8_t morse_set_fox_max(uint8_t max);
16
17 uint8_t morse_set_transmit_minute(uint8_t transmit_minute);
18 uint8_t morse_get_transmit_minute(void);
19
20 void morse_set_call_sign(uint8_t call);
21 uint8_t morse_get_call_sign(void);
22
```

```
23 void morse_set_morse_mode(uint8_t mode);
24 uint8_t morse_get_morse_mode(void);
25
26 void morse_enable_continuous_carrier(void);
27 void morse_disable_continuous_carrier(void);
28
29 void morse_start_time(void);
30 void morse_stop_time(void);
31
32 void morse_interrupt(void);
```

Listing 17.1: Public Functions des Morse-Softwaremoduls

Das Morse-Modul speichert die eigenen Konfigurationsinformationen im EEPROM ab und bietet Funktionen, über den die Konfiguration gesetzt und gelesen werden kann.

Die Funktionen `morse_set_morse_unit` und `morse_get_morse_unit` dienen zum Einstellen der Morsegeschwindigkeit. Anzugeben ist hierbei die intern verwendete Zeitbasis. Mit der Funktion `morse_convert_wpm_morse_unit` kann die Geschwindigkeitsangabe in der Einheit WPM in die interne Zeitbasis umgerechnet werden.

Mithilfe der Funktion `morse_set_transmit_minute` kann angegeben werden, in welcher Minute der Fuchs senden soll. Der erste Fuchs muss Minute 0, der zweite Minute 1 usw. eingestellt haben.

Die Funktion `morse_set_fox_max` dient zur Einstellung der Gesamtzahl an Füchsen, üblicherweise 5 oder weniger. Anhand dieser Einstellung wird bestimmt, wie oft sich das Senden wiederholt. Bei 5 Füchsen insgesamt sendet jeder Fuchs alle 5 Minuten einmal, bei 4 sendet jeder alle 4 Minuten usw.

Mit der Funktion `morse_set_fox_number` kann am Fuchs eine Nummer zwischen 0 und 5 eingestellt werden, wodurch der Fuchs von der PC-Software identifiziert werden kann. Die Nummer 0 signalisiert dabei, dass es sich um den Demo-Sender handelt, der beim Startpunkt des Wettbewerbs steht und als Bakensender dauerhaft sendet.

Die Funktion `morse_set_call_sign` dient zur Einstellung des Rufzeichens. Als Parameter muss dabei eine, der in der Headerdatei morse.h definierten Konstanten, `CALL_MOE`, `CALL_MOI` usw. übergeben werden.

Über die Funktion `morse_set_morse_mode` kann das Morsen aktiviert oder deaktiviert werden. Ist das Morsen deaktiviert, wird stattdessen unterbrechungsfrei der Träger in den Minuten, in denen der Fuchs sonst morsen würde, ausgesendet. Diese Einstellung wird im EEPROM gespeichert und dient zur Festlegung der gesamten Betriebsart, nicht für das kurzzeitige Aussenden eines dauerhaften Trägers zum Abstimmen der Antenne.

Zum Abstimmen der Antenne dient die Funktion `morse_enable_continuous_carrier`. Drückt der Nutzer einen der beiden Taster zum Abstimmen der Antenne, so wird im DDS-Softwaremodul die Funktion `dds_enable_continuous_carrier` aufgerufen, die den DDS-Baustein auf die richtige Frequenz setzt und anschließend die hier beschriebene Funktion des Morsemoduls aufruft. Diese Einstellung wird nicht im EEPROM gespeichert, sondern bleibt nur solange bestehen, bis der Fuchs entweder neu gestartet oder bis die Funktion `morse_disable_continuous_carrier` aufgerufen wird.

Durch Aufruf der Funktionen `morse_start_time` und `morse_stop_time` wird dem Morse-Softwaremodul vom Startup-Softwaremodul mitgeteilt, dass nun die Start- bzw. Stoppzeit erreicht ist. Das Morsemodul beginnt oder beendet anschließend das Morsen.

Jede Minute muss das Morsemodul überprüfen, ob der Fuchs in der nächsten Minute senden soll oder nicht. Damit das Modul jede Minute eine Benachrichtigung erhält, wird ein Alarm der Real-Time Clock benützt. Die Interruptroutine zu diesem Alarm, befindet sich im Softwaremodul zur Real-Time Clock. Diese Interruptroutine ruft anschließend `morse_interrupt` auf und gibt dem Morsemodul damit die Möglichkeit, das Morsen entsprechend der Konfiguration für die nächste Minute zu starten oder einzustellen.

17.4. Implementierung

17.4.1. Zeitbasis Timer - Interruptroutine

Das DDS-Ausgangssignal muss zum Morsen regelmäßig ein- und ausgeschaltet werden. Um die richtigen Umschaltzeitpunkte festzustellen, braucht der Mikrocontroller eine Zeitbasis, die hier mithilfe eines Timers realisiert wird. Der Timer löst regelmäßig eine Interruptroutine aus, in welcher der nächste Teil des Morsezeichens ausgegeben werden kann.

Der Timer wird auf eine Intervallrate eingestellt, mit der die Interrupts ausgelöst werden. Einstellbar ist die Anzahl der Interrupts, die nötig sind, bis die Zeit eines Punktes abgelaufen ist. Die Morse-Unit, die bei der Funktion `morse_set_morse_unit` eingestellt wird, stellt die Anzahl an Interrupts dar, die auftreten, bis die Zeit für einen Punkt abgelaufen ist.

Die Zeit zwischen zwei Interrupts ist auf 5 ms gesetzt. Diese Zeit ist die geringste Auflösung, die bei einer Veränderung der Morsegeschwindigkeit konfigurierbar ist. Das bedeutet, dass insbesondere bei schnellen Morsegeschwindigkeiten keine genaue Einstellung mehr möglich ist und die tatsächliche Morsegeschwindigkeit

dann von der gewünschten abweicht. Die gewählte Intervallzeit ist ein Kompromiss zwischen zu vielen, den Controller auslastenden, Interruptroutinen und einer zu geringen Auflösung bei der Einstellung der Morsegeschwindigkeit. Für die bei Fuchsjagden üblichen Morsegeschwindigkeiten zwischen 8 und 15 WPM (vgl. *Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1* 2015, Seite 10) sind die 5 ms ausreichend.

Die Morsezeichen sind in Variablen kodiert gespeichert, die von der Interruptroutine eingelesen und der Reihe nach abgearbeitet werden. Immer 2 Bits bilden gemeinsam ein Morseelement (entweder Strich, Punkt, Pause-Wort, Ende-Wort). Die Pause zwischen zwei einzelnen Morseelementen mit der Länge eines Punktes wird automatisch eingefügt.

17.4.2. morse_start_time / morse_stop_time:

Diese beiden Funktionen beeinflussen eine Variable, die kennzeichnet, ob man sich zwischen Start- und Stoppzeit befindet. Die Funktion `morse_start_time` setzt den Timer der Real Time Clock zusätzlich auf die nächste Minute zum Senden bzw. auf das Ende der aktuellen Sendeminute.

Das eigentliche Starten bzw. Stoppen des Sendens wird hingegen durch die Funktionen `morse_start_minute` und `morse_stop_minute` eingeleitet.

17.4.3. morse_start_minute / morse_stop_minute:

Diese beiden internen Funktionen werden von der `morse_interrupt`-Funktion aufgerufen, um das Senden zu starten bzw. zu stoppen. Die beiden Funktionen setzen eine Variable im RAM, die kennzeichnet, dass in dieser Minute gesendet bzw. nicht gesendet werden soll.

Schließlich wird die Funktion `update_start` aufgerufen, die den Timer abhängig von den durch `morse_start_minute` und `morse_start_time` (bzw. den entsprechenden `stop`-Funktionen) gesetzten Einstellungen startet oder stoppt.

17.4.4. update_start:

Die interne Funktion `update_start` wird immer nach einer Veränderung durch die Funktionen `morse_start_minute`, `morse_stop_minute`, `morse_enable_continuous_carrier` und `morse_disable_continuous_carrier` aufgerufen. Der Timer wird je nach aktuellem Zustand gestartet oder gestoppt.

17.4.5. is_on_minute / is_off_minute:

Diese beiden internen Hilfsfunktionen bestimmen, ob in einer bestimmten Minute gesendet werden soll oder nicht. Für diese Überprüfung müssen die durch `morse_set_transmit_minute` festgelegte Sendeminute, die maximale Anzahl der im Wettbewerb verwendeten Füchse und die Minuteneinstellung der Startzeit berücksichtigt werden.

Der Minutenwert der Startzeit wird benötigt, da zu Beginn der Fuchsjagd der erste Fuchs senden soll. Gewöhnlich wird die Startzeit auf einen „runden“ Wert wie 0 oder 30 gesetzt sein. Sollte die Minute jedoch auf einen Wert wie 1, 2, 3 usw. gesetzt werden, so wird dadurch sichergestellt, dass auch in diesem Fall der erste Fuchs am Beginn der Fuchsjagd sendet.

Da 60 Minuten pro Stunde durch alle einstellbaren Fuchanszahlen (5, 4, 3, 2, 1) ohne Rest teilbar sind, bleiben Sendeminuten unabhängig von der Stunde. Der Stundenwert der Startzeit und die bisher seit Beginn des Wettbewerbs vergangenen Stunden müssen daher von diesen beiden Funktionen nicht berücksichtigt werden.

Die Funktionen finden beispielsweise in `morse_interrupt` Verwendung, um zu bestimmen, ob es sich bei der aktuellen Minute tatsächlich um eine Minute zum Senden bzw. Nicht-Senden handelt. So werden Fehler durch eventuell falsch ausgelöste Interrupts vermieden.

18. Softwaremodul RFID

18.1. Einführung

RFID ist eine Technologie zur drahtlosen Übertragung von Daten über kleine Entfernungen im Zentimeterbereich. Die Übertragung erfolgt zwischen einem Lesegerät und einem Transponder, der beispielsweise in Form von einer Chipkarte oder einem kleinen Plastikchip ausgeführt sein kann. Die Technologie ermöglicht die Konstruktion passiver Transponder, die keine extra Stromversorgung benötigen, sondern die benötigte Energie durch die vom Lesegerät abgestrahlte Feldenergie erhalten. (vgl. Wikipedia 2016d)

Von verschiedenen Herstellern werden ICs angeboten, die das Schreiben und Lesen von RFID-Transpondern ermöglichen. In diesem Projekt kommt ein Modul zum Einsatz, das auf dem IC MFRC522 der Firma NXP basiert (vgl. *MFRC522 Standard 3V MIFARE Reader Solution Datasheet* 2014).

Zur Ansteuerung dieses Moduls existiert eine Softwarebibliothek für das Arduino-System (vgl. *Arduino Mifare MFRC522 RFID Reader/Writer* 2016), die frei ohne Lizenzeinschränkungen verwendet werden darf. Beim Arduino-System werden ebenfalls AVR-Mikrocontroller verwendet. Die Bibliothek kann somit relativ leicht für die Verwendung in diesem Projekt umgebaut werden.

18.1.1. Anforderungen an das Softwaremodul

Aufgabe des Softwaremoduls RFID ist es, Funktionen für andere Module bereitzustellen, mit deren Hilfe neue Transponder erkannt, gelesen und beschrieben werden können.

Das Modul setzt sich zusammen aus den Dateien der RFID-Arduino-Bibliothek MFRC522.c und MFRC522.h, den Dateien Arduino.c, Arduino.h, SPI.c und SPI.h, welche die von der RFID-Bibliothek benötigten arduino-kompatiblen Funktionen anbieten, und aus den Dateien rfid.c und rfid.h, die einige Funktionen der Bibliothek zusammenfassen und die Schnittstelle zu den übrigen Teilen der Software bilden.

18.2. Mifare Classic 1K Transponder

Die MFRC522-Bausteine unterstützen verschiedene Standards unterschiedlicher RFID-Transponder wie MIFARE Mini, MIFARE 1K, MIFARE 4K, MIFARE Ultralight etc. (vgl. *MFRC522 Standard 3V MIFARE Reader Solution Datasheet* 2014). Die Standards unterscheiden sich in Eigenschaften wie dem verfügbaren Speicherplatz, der Verfügbarkeit von Verschlüsselung usw.

Dieses Projekt unterstützt die Transponder MIFARE Classic 1K, die insgesamt 1 kB Speicherplatz bieten (vgl. *MIFARE Classic 1K Product Datasheet* 2011, Seite 8). Der Speicherplatz ist in 16 Sektoren unterteilt, die jeweils vier 16-Byte große Blöcke beinhalten. Jeder einzelne Sektor kann über einen eigenen Schlüssel lese- und schreibgeschützt werden. Der Schlüssel und die Zugriffseinstellungen eines Sektors werden jeweils im 4. Block eines Sektors abgelegt. Der erste Block im ersten Sektor ist bereits durch den Hersteller vorprogrammiert und schreibgeschützt, wodurch er für die Speicherung eigener Daten nicht zur Verfügung steht (vgl. *MIFARE Classic 1K Product Datasheet* 2011, Seite 9).

Die bei Mifare-Classic-Transpondern verwendete Verschlüsselung wird seit Jahren als unsicher erachtet. Henryk Plötz hat in seiner Diplomarbeit aus dem Jahr 2008 verschiedene Angriffsmöglichkeiten aufgezeigt, die die Verschlüsselung aushebeln können (vgl. Plötz 2008).

Aus diesem Grund wird in diesem Projekt vollständig auf die Verwendung der angebotenen Verschlüsselung verzichtet. Der Schlüssel der verwendeten Transponder muss dem Schlüssel im Auslieferungszustand (6 Bytes mit jeweils 0xFF) entsprechen und wird durch die Software nicht verändert. Eine gewisse Fälschungssicherheit wird dadurch erreicht, dass jeder Sender eine den Teilnehmern unbekannte Geheimzahl und die Nummern der letzten Teilnehmer, die den Sender aufgespürt haben, auf dem Transponder ablegt.

18.3. Modifikation der Arduino-RFID-Bibliothek

Beim Arduino-System ist es üblich, viele Aufgaben wie beispielsweise das Einstellen des Ausgangszustandes von IO-Pins oder die SPI-Kommunikation nicht über die Register des Mikrocontrollers, sondern über entsprechende, durch die Arduino-Umgebung bereitgestellte Funktionen zu erledigen (vgl. Wikipedia 2016a). In diesem Projekt wird auf die Verwendung der Arduino-Umgebung verzichtet, daher müssen die von der RFID-Bibliothek aufgerufenen Arduino-Funktionen nachgebildet werden.

Zudem liegt dem Arduino-System die Programmiersprache C/C++ zugrunde (vgl. Wikipedia 2016a). Da die Software für den ARDF-Sender ausschließlich in C verfasst ist und auf die Verwendung von Programmiersprachenelementen der Sprache C++ verzichtet werden soll, sind einige Anpassungen erforderlich.

18.3.1. Umbau C++ zu C

C++-Klasse

Die Arduino-RFID-Bibliothek in den Dateien MFRC522.cpp und MFRC522.h definiert für die RFID-Funktionen eine Klasse mit dem Namen MFRC522. In C können keine Klassen verwendet werden. Der Vorteil einer Klasse ist, dass mehrere gleichartige Objekte basierend auf der in der Klasse beschriebenen Vorlage erstellt werden können. Bei Verwendung der Arduino-Bibliothek würde jedoch nur ein Objekt der Klasse erstellt werden, weil auch nur ein RFID-Modul verwendet wird. Da mehrere Objekte nicht notwendig sind, ist es einfach, die Klasse in ein C-Softwaremodul umzubauen.

In der Headerdatei müssen die Schlüsselwörter `class`, `private` und `public` entfernt werden, da diese in C nicht definiert sind. Alle als `private` deklarierten Variablen und Datenstrukturen werden aus der Headerdatei in die zugehörige C-Datei verschoben, sodass sie außerhalb des Moduls nicht sichtbar sind.

Neben den `private`-Daten werden auch die außerhalb der Klasse definierten Datenstrukturen (beispielsweise Arrays im Flashspeicher) aus der Headerdatei in die C-Datei verschoben. Ansonsten würde die Datenstruktur in jedes C-File kompiliert, das die entsprechende Headerdatei inkludiert. Dadurch wären die gleichen Variablennamen in mehreren Objektdateien vorhanden, was zu einem Fehler beim Linken führen würde.

Die als `public` definierten Variablen der Klasse werden in der Headerdatei als `extern` deklariert, wodurch dem Compiler mitgeteilt wird, dass es eine Variable mit dem entsprechenden Namen gibt, ohne dass an dieser Stelle Speicherplatz für die Variable reserviert wird. Die Definition der öffentlichen Variablen, die eine Reservierung des Speicherplatzes zufolge hat, wird in der C-Datei vorgenommen.

Weiters muss in der C-Datei vor allen Funktionen der `MFRC522::`-Prefix entfernt werden, der kennzeichnet, dass diese Funktion einer Klasse angehört. Die Funktionen sind somit alle im globalen C-Namensraum verfügbar. Es muss beachtet werden, dass nicht bereits andere Funktionen mit demselben Namen im Projekt verwendet werden.

Abschließend wird der Konstruktor `MFRC522::MFRC522` in eine Funktion namens `MFRC522_init` umbenannt.

Standardwerte für Funktionsparameter

In C++ ist es möglich, Parametern einer Funktion bei der Definition einen Standardwert zuzuweisen. Werden beim Aufruf der entsprechenden Funktion ein oder mehrere Parameter am Schluss nicht angegeben, so wird für diese Parameter der entsprechende Standardwert verwendet.

Solche Standardwerte für Parameter sind in C nicht erlaubt. Daher wird jede Funktion mit Standardwerten in mehrere Funktionsprototypen aufgeteilt, an deren Namen jeweils ein Suffix angefügt wird, das die Anzahl der zu übergebenden Parameter kennzeichnet (z.B. `PCD_CommunicateWithPICC_four` für eine Funktion bei der nur 4 Parameter angegeben und für alle anderen die Standardwerte verwendet werden sollen). Diese Funktionen mit dem entsprechenden Suffix rufen dann die ursprüngliche Funktion auf und fügen beim Aufruf die entsprechenden Standardwerte für die zusätzlichen Parameter ein.

18.3.2. Nachbau der Arduino-Funktionen

Die RFID-Bibliothek greift auf Arduino-Funktionen zur Ausgabe auf der UART-Schnittstelle, zur Kommunikation über SPI und zum Konfigurieren der digitalen IO-Pins zu. Um die Bibliothek in diesem Projekt zu verwenden, müssen Funktionen bereitgestellt werden, die die gleiche Funktionalität bieten.

Arduino.c / Arduino.h

In dieser Datei werden die bei Arduino verwendeten und in C nicht enthaltenen Datentypen `byte`, `bool` und `word` definiert. Symbolische Konstanten des Arduino-Systems wie `OUTPUT` und `INPUT` bzw. `HIGH` und `LOW` müssen ebenfalls einem Wert zugeordnet werden.

Im Arduino-System kann das Objekt `Serial` verwendet werden, um Daten auf der UART-Schnittstelle auszugeben. Für C werden die benötigten Funktionen jeweils ohne Klasse und Objekt deklariert. Aus dem Aufruf der Methode `Serial.print()` wird der Aufruf der Funktion `Serial_print()`. Die Methode `print` im Arduino-System besitzt mehrere Überladungen, wodurch eine Vielzahl verschiedener Datentypen der gleichen Funktion übergeben werden können. Überladungen werden

in C nicht unterstützt, daher muss für jeden Datentyp eine eigene Funktion angeboten werden, die nach dem Schema `Serial_print_typ` (z.B. `Serial_print_byte`, `Serial_print_string`) benannt werden. Die Funktionsaufrufe innerhalb der Bibliothek müssen dem Datentyp entsprechend abgeändert werden.

SPI.c / SPI.h

Die Dateien SPI.c und SPI.h bilden die SPI-Funktionen des Arduino-Systems nach. Wiederum wird hier das Objekt `SPI` durch den Funktionsprefix `SPI` ersetzt. Dadurch wird beispielsweise aus der Methode `SPI.transfer()` der Funktionsaufruf `SPI_transfer()`

Gewöhnlich führt die RFID-Bibliothek bei der Initialisierung die Konfiguration der SPI-Schnittstelle durch. Da die SPI-Schnittstelle in diesem Projekt jedoch bereits durch das DDS-Softwaremodul verwaltet wird, werden die entsprechenden Initialisierungsfunktionen des Arduino-Systems wie `SPI_begin` durch leere Funktionen ersetzt.

18.3.3. Reduktion auf benötigte Elemente

Die RFID-Bibliothek enthält verschiedene im Projekt ungenutzte Funktionen wie beispielsweise die Ausgabe der Transponder-Informationen über UART. Um Speicherplatz zu sparen, wird nach dem Einfügen aller benötigten Funktionsaufrufe in der höheren Softwareschicht der gesamte Code in der Datei MFRC522.c auskommentiert. Schrittweise wird die Auskommentierung bei allen benötigten Funktionen wieder rückgängig gemacht, bis der Compiler beim Übersetzungsvorgang keine Fehlermeldungen mehr liefert. Dadurch wird erreicht, dass alle nicht erforderlichen Funktionen auch nicht in das Programm einfließen und dort Speicherplatz in Anspruch nehmen.

18.4. Softwareschnittstelle

Die übrigen Programmteile des Projekts sollen nicht direkt auf Funktionen der Arduino-RFID-Bibliothek zugreifen. Stattdessen passen die Dateien rfid.c und rfid.h die Funktionsnamen der RFID-Bibliothek an das im Softwareprojekt verwendete Funktionsnamenschema an. Teilweise werden mehrere Funktionen der RFID-Bibliothek zusammengefasst und eine einfache Schnittstelle zum Zugriff auf die RFID-Hardware geboten.

```
1 void rfid_init(void);
2
3 void rfid_loop(void);
4
5 uint8_t rfid_read_block(uint8_t block, uint8_t *buff, uint8_t *length);
6 uint8_t rfid_write_block(uint8_t block, uint8_t *buff, uint8_t length);
7
8 void rfid_close_tag(void);
```

Listing 18.1: Public Functions der Datei rfid.h

Die `rfid_loop`-Funktion wird regelmäßig von der Programmhauptschleife aufgerufen und prüft, ob ein neuer Transponder erkannt worden ist.

Mithilfe der Funktionen `rfid_write_block` und `rfid_read_block` kann ein 16-Byte-Block geschrieben oder gelesen werden.

Der Funktion `rfid_read_block` muss dazu ein Pointer auf ein insgesamt 18-Bytes großes Datenarray, die Nummer des zu lesenden Blocks und ein Pointer auf eine Byte-Variable übergeben werden. Diese Byte-Variable muss zu Beginn die Größe des Arrays enthalten. Nach Funktionsaufruf kann aus dieser Variable die Anzahl der in das Array geschriebenen Bytes herausgelesen werden. Das Array muss deshalb 18-Bytes und nicht nur 16-Bytes groß sein, da die letzten zwei Bytes von der Lesefunktion mit einer Prüfsumme beschrieben werden.

Die Funktion `rfid_write_block` erwartet einen Pointer auf ein Datenarray mit den zu schreibenden Daten, eine Byte-Variable, die die Länge des Datenarrays enthält und die Nummer des zu schreibenden Blocks.

Die übergebene Blocknummer setzt sich jeweils zusammen aus:

$$\text{Blocknummer} = 4 \cdot \text{Sektornummer} + \text{Blocknummer innerhalb des Sektors} \quad (18.1)$$

Nach der Kommunikation mit einem Tag muss die Funktion `rfid_close_tag` aufgerufen werden. Ansonsten kann mit einem neuen Tag keine Verbindung mehr aufgebaut werden.

18.5. Implementierung

18.5.1. rfid_loop

Die RFID-Bibliothek bietet die Funktion `PICC_IsNewCardPresent`, um zu überprüfen, ob sich ein neuer Transponder im Einflussbereich des Schreib-/Lesegeräts befindet.

Wenn eine neuer Transponder gefunden worden ist und auch der Aufruf der Funktion `PICC_ReadCardSerial` zum Auswählen des ersten Transponders im Einflussbereich eine Erfolgsmeldung zurückgibt, wird im Softwaremodul User die Funktion `user_new_tag` aufgerufen, um dem Softwaremodul mitzuteilen, dass ein Transponder erkannt worden ist. Die mittels `PICC_ReadCardSerial` ausgelesene ID wird im weiteren Verlauf nicht mehr vom Programm verwendet, da zur Identifizierung der Transponder eine eigene Identifikationsnummer in den Speicherplatz des Transponders geschrieben wird.

18.5.2. rfid_read_block

Die Funktion zum Auslesen eines Blocks versucht zuerst, sich am Transponder mithilfe der Funktion `PCD_Authenticate` zu authentifizieren. Zur Authentifizierung wird der bei Auslieferung eingestellte Schlüssel, bestehend aus 6 aufeinanderfolgenden Bytes mit dem Wert 0xFF, verwendet (vgl. *MIFARE Classic 1K Product Datasheet* 2011, Seite 10). Auch eine mehrmalige Authentifizierung am gleichen Transponder durch mehrmaligen Aufruf der Funktion `rfid_read_block` ist problemlos möglich. Anschließend wird mithilfe der Funktion `MIFARE_Read` der angeforderte Block gelesen.

18.5.3. rfid_write_block

Die Schreibefunktion ruft wiederum die `PCD_Authenticate`-Funktion zur Authentifizierung auf. Anschließend werden die Daten durch die `MIFARE_Write`-Funktion geschrieben.

18.5.4. rfid_close

Die `rfid_close`-Funktion ruft die Funktion `PCD_StopCryptol` auf, wodurch das Schreib- und Lesegerät dazu angewiesen wird, die Authentifizierung zu beenden, damit neue Tags erkannt werden können.

19. Softwaremodul User

19.1. Anforderungen

Beim Wettbewerb sollen alle Teilnehmer mit einem RFID-Transponder ausgestattet werden, der zu Beginn jeweils mit einer einzigartigen Kennung beschrieben wird. Jeder Sender verfügt über ein RFID-Schreib-/Lesegerät, das die Kennung des Transponders ausliest, sobald der Teilnehmer den Tag in die unmittelbare Nähe des Lesegeräts bringt. Der Sender soll die Transponderkennung in sein EEPROM gemeinsam mit einem Zeitstempel eintragen. Außerdem werden auf dem Transponder selbst die Geheimzahl des Senders und die Liste mit den letzten angekommenen Teilnehmern (Teilnehmerhistorie) abgelegt, um Manipulationen zu erschweren. Diese Aufgaben sollen vom Softwaremodul User, aufbauend auf der RFID-Bibliothek, erledigt werden.

Weiters muss das Softwaremodul auch eine Möglichkeit bieten, die Transponderinformationen und die Informationen aus dem EEPROM des Senders am Ende des Wettbewerbs wieder auszulesen und zum PC zu übertragen.

19.2. Aufteilung des Speichers

19.2.1. Transponder

Am Transponder müssen folgende Informationen abgespeichert werden:

- ID des Teilnehmers
- 5x Geheimzahl der einzelnen Sender
- 5x Teilnehmerhistorie der einzelnen Sender

Insgesamt verfügt ein Transponder des Typs MIFARE Classic 1K über 16 Sektoren. Ein Sektor wird für die Geheimzahlen der einzelnen Sender und für die ID des Teilnehmers reserviert. Die 15 übrigen Sektoren ergeben aufgeteilt auf die fünf Sender drei Sektoren pro Sender zum Speichern der Historie.

19.2.2. EEPROM des Senders

Die Einträge der Historie werden der Reihe nach in den EEPROM geschrieben. Die ersten zehn Speicherplätze sind reserviert für andere Informationen, die bei eventuellen Erweiterungen im EEPROM abgelegt werden müssen. Die ersten zwei Bytes enthalten die Adresse des jeweils nächsten freien Speicherplatzes im EEPROM. Zum Löschen der gesamten Historie muss so nur der Zeiger auf den Beginn des EEPROMs (also auf Adresse 10, um die reservierten Adressen freizuhalten) gesetzt werden.

19.3. Softwareschnittstelle

In der Headerdatei user.h sind folgende Funktionen deklariert:

```
1  void user_show_configuration(void);
2
3  void user_start_time(void);
4  void user_stop_time(void);
5
6  uint8_t user_new_tag(void);
7
8  void user_time_tick(void);
9
10 uint8_t user_set_id(uint16_t tag_id);
11 void user_cancel_set_id(void);
12 uint8_t user_check_set_id(void);
13
14 void user_set_secret(uint16_t secret);
15 uint16_t user_get_secret(void);
16
17 void user_clear_history(void);
18
19 void user_print_fox_history(void);
```

Listing 19.1: Public Functions des User-Softwaremoduls

Die Funktionen `user_start_time` und `user_stop_time` werden vom Startup-Softwaremodul aufgerufen, um mitzuteilen, wann der Wettbewerb begonnen bzw. geendet hat. Das User-Modul muss über diese Information verfügen, da vor dem Start des Wettbewerbs keine Transponder in die Historie eingetragen werden sollen.

`user_new_tag` wird vom RFID-Softwaremodul aufgerufen, sobald ein neuer Transponder erkannt wird, damit das User-Modul die nächsten Schritte auswählen kann.

`user_time_tick` wird regelmäßig von einem Timer in der Hauptschleife des Main-Moduls aufgerufen, damit das User-Modul eine Zeitbasis bekommt, über die beispielsweise die Wartezeit, bis der gleiche Transponder erneut auszulesen ist, realisiert wird.

Zum Setzen der Teilnehmer-ID und zum Initialisieren eines Transponders kann die Funktion `user_set_id` aufgerufen werden. Das User-Modul speichert die Anfrage solange, bis der nächste Transponder erkannt wird, der dann mit der gewählten ID beschrieben wird. Die Funktion löscht ebenfalls alle gespeicherten Informationen vom letzten Wettbewerb.

Mithilfe der Funktion `user_cancel_set_id` kann der vorher gestartete Set-ID-Vorgang abgebrochen werden, solange kein Transponder in die Nähe des RFID-Moduls gebracht worden ist.

`user_check_set_id_state` liefert die Information zurück, ob das User-Modul derzeit einen Set-ID-Befehl zwischengespeichert hat oder nicht.

Durch `user_set_secret` und `user_get_secret` kann die Geheimzahl des Senders eingestellt und ausgelesen werden, die schließlich auf die Transponder als Bestätigung für die Ankunft beim Sender geschrieben werden.

`user_clear_history` löscht die gespeicherten Daten über die zuletzt vom Sender registrierten Teilnehmer.

Der Aufruf von `user_print_fox_history` bewirkt die Ausgabe der registrierten Teilnehmer über die UART-Schnittstelle.

Für die Ausgabe der auf dem Transponder gespeicherten Informationen ist keine öffentliche Funktion nötig. Sobald ein Transponder erkannt wird, liest ihn die Software aus und sendet die Informationen selbstständig über die UART-Schnittstelle.

19.4. Implementierung

19.4.1. user_new_tag

Die Funktion `user_new_tag` wird aufgerufen, sobald ein neuer Transponder erkannt wird. Hat die Software eine Set-ID-Anfrage zwischengespeichert, so wird die ID am erkannten Transponder durch Aufruf der internen Funktion `user_write_tag_id` gesetzt. Alle übrigen Daten am Tag werden gelöscht.

Andernfalls werden die Transponderdaten ausgelesen und über die UART-Schnittstelle gesendet. Falls es sich beim Sender nicht um den Demo-Sender handelt und

der Wettbewerb bereits begonnen hat, werden Sendergeheimzahl und Teilnehmerhistorie zusätzlich auf den Transponder geschrieben.

Sobald die Übertragung beendet ist, wird das RFID-System den Transponder wieder als neuen Transponder erkennen und erneut die Funktion `user_new_tag` aufrufen. Beim Schreiben der Teilnehmerhistorie wird deshalb darauf geachtet, ob die ID des Transponders der ID des zuletzt geschriebenen entspricht. Ist dies der Fall erfolgt kein erneuter Schreibvorgang und auch kein weiterer Eintrag des Transponders in den EEPROM des Senders.

Der Auslesevorgang muss ebenfalls begrenzt werden. Ansonsten würden die Informationen ununterbrochen über die UART-Schnittstelle gesendet werden, sobald der Transponder sich in der Nähe befindet. Durch Aufruf der Funktion `user_set_read_timeout` wird eine Zeitsperre von einigen Sekunden gesetzt, die ablaufen muss, bevor der Transponder erneut ausgelesen wird.

Dabei wird der Timer aus der Datei main.c benützt. Die ISR des Timers ruft die Funktion `user_time_tick` auf, die die Anzahl der Aufrufe zählt und nach Ablauf der Zeitspanne die Sperre zurücksetzt. Der Timer wird außerdem benützt, um eine Status-LED blinken zu lassen, die dem Benutzer signalisiert, dass der Transponder erkannt worden ist.

19.4.2. user_set_id

Die Funktion selbst setzt nur die Variable `write_id` und speichert die zu schreibende Transponder-ID. Der eigentliche Schreibvorgang wird erst ausgelöst, sobald der nächste Transponder erkannt und dadurch die Funktion `user_new_tag` aufgerufen wird.

19.4.3. user_write_id

Die interne Funktion `user_write_id` dient zum Schreiben auf den Transponder und wird von `user_new_tag` aufgerufen.

Zu Beginn der Funktion werden die Speicherstellen, die später die Teilnehmerhistorie aufnehmen, mit Nullen überschrieben, um vorhandene Daten zu löschen.

Im Block auf dem Transponder, welcher die Geheimzahlen der Sender und die Transponder-ID enthält, werden alle Geheimzahlen gelöscht und die zu setzende Transponder-ID an der entsprechenden Stelle eingetragen.

19.4.4. user_add_to_history

Die Funktion fügt den erkannten Transponder in die Teilnehmerhistorie des Senders ein. An einer festgelegten Speicherstelle im EEPROM wird die Adresse des nächsten freien Speicherplatzes im EEPROM als Zeiger eingetragen. Zum Löschen der Teilnehmerhistorie muss somit nicht der gesamte Speicher überschrieben werden, sondern es reicht, den Zeiger auf den Anfang des EEPROMs zeigen zu lassen.

Zusätzlich wird der Transponder in einen Ringbuffer im RAM geschrieben. Beim Schreiben der letzten Teilnehmer auf den Transponder werden die Daten aus diesem Ringbuffer gelesen, wodurch auf das Lesen des EEPROMs verzichtet werden kann.

Ein Eintrag besteht dabei jeweils aus vier Bytes. Die ersten beiden Bytes enthalten die 16-bit Transponder-ID, das dritte und das vierte Byte stellen einen Zeitstempel dar, der die Sekunden seit Beginn des Wettbewerbs bis zum Zeitpunkt der Ankunft beim Sender kennzeichnet. Beide Datenwörter werden im Little-Endian-Format abgelegt, sodass jeweils das niederwertigere Byte an der Speicherstelle mit der kleineren Adresse liegt.

Der 16-bit Zeitstempel kann Zeitspannen bis zu 65535 ($= 2^{16} - 1$) Sekunden abdecken. Dies entspricht etwas mehr als 18 Stunden. Die Dauer eines Wettbewerbs ist gewöhnlich bei weitem geringer.

19.4.5. user_write_history

Die Funktion bestimmt durch Aufruf der internen Funktion `get_history_block_physical` die Nummern der Blöcke, in die der Sender die Teilnehmerhistorie auf dem Transponder ablegen soll. Abhängig von der Sendernummer, die mit `morse_get_fox_number` abgefragt werden kann, und von einer logischen Blockadresse zwischen 0 und 2, kann so die reale Blocknummer auf dem Transponder ermittelt werden.

Aus dem Ringbuffer im RAM werden die letzten erkannten Transponder und die zugehörigen Zeitstempel ausgelesen und auf dem Transponder abgespeichert. Zusätzlich legt die Software die Geheimzahl des Senders in den dafür vorgesehen Speicherplatz im Block mit der Transponder-ID ab.

19.4.6. user_read_tag

Die Funktion liest die ID, alle gespeicherten Geheimzahlen und die Liste mit den zuletzt registrierten Teilnehmern der verschiedenen Sender aus und überträgt die Informationen über die UART-Schnittstelle.

Aufgerufen wird die Funktion durch `user_new_tag`.

20. Softwaremodul Main

20.1. Aufgabe

Im Main-Modul befindet sich die Funktion `main` als Eintrittspunkt des Programms. Zu Beginn werden alle anderen Softwaremodule initialisiert. Der Begrüßungstext wird auf der UART-Schnittstelle ausgegeben.

Anschließend tritt das Programm in eine Dauerschleife ein, die während der gesamten Programmausführung durchlaufen wird. Innerhalb der Dauerschleife wird durch Aufruf von `commands_execute` regelmäßig überprüft, ob neue Befehle über die UART-Schnittstelle empfangen worden sind. Durch `rfid_loop` wird überprüft, ob sich ein neuer Tag in der Nähe des RFID-Moduls befindet. Im letzten Teil werden die Taster zum Abstimmen der Antenne abgefragt.

Wenn ein Taster aktiviert wird, schaltet das Programm einen dauerhaften Träger ein und beginnt damit, die vom Richtkoppler zur Verfügung gestellten analogen Spannungswerte auszulesen, das Stehwellenverhältnis zu berechnen und auf der LED-Bar auszugeben.

20.2. Weitere Funktionen

20.2.1. main_reload

Nach Aufruf dieser Funktion weist das Main-Modul alle anderen Softwaremodule an, ihre Einstellungen aus dem EEPROM neu zu laden. Die Funktion wird gewöhnlich vom Commands-Modul nach jedem eingegebenen Befehl aufgerufen, um die Einstellungen sofort zu übernehmen.

20.2.2. main_set_blinking

Das Main-Modul verwaltet einen Timer, durch den nach Aufruf dieser Funktion die Status-LED zu blinken beginnt. Die Funktion wird von der PC-Software verwendet, um dem Benutzer zu signalisieren, welcher Sender gerade ausgewählt ist.

21. PC-Software

21.1. Anforderungen

Die Firmware des Senders ist so gestaltet, dass sie über eine textbasierte Kommandozeilenschnittstelle konfiguriert werden kann. Somit reicht zur Konfiguration grundsätzlich ein beliebiges Terminalprogramm aus, das Text über die serielle Schnittstelle empfangen und senden kann.

Zur komfortableren Konfiguration des Senders soll zusätzlich eine grafische Oberfläche zur Verfügung stehen, in der die nötigen Parameter eingestellt, gespeichert und anschließend zu den Sendern übertragen werden können.

Neben den Einstellungen der Sender wie Frequenz, Amplitude, Rufzeichen etc. soll die Software auch die Möglichkeit bieten, eine Teilnehmerliste für den Wettbewerb anzulegen und anhand dieser Liste die RFID-Transponder für die Teilnehmer einzurichten.

Nach dem Wettbewerb sollen die auf den Transpondern abgelegten Daten und die im EEPROM der Sender gespeicherte Information über die registrierten Teilnehmer wieder ausgelesen und in das CSV-Format exportiert werden. Über die exportierte CSV-Datei können die Daten mit anderen Programmen weiterverarbeitet werden.

21.2. Wahl der Programmierumgebung

Zur Erstellung einer Software mit grafischer Oberfläche existieren verschiedene Möglichkeiten. Wichtig ist in diesem Fall vor allem, dass die PC-Software auf dem Betriebssystem Windows lauffähig ist.

Mithilfe der Programmiersprachen C bzw. C++ können durch Verwendung der vom Betriebssystem bereitgestellten Windows API oder mit plattformunabhängige Toolkits wie Qt, Gtk oder wxWidgets Programme mit grafischer Oberfläche

erstellt werden. Da C/C++ auf einer niedrigen Ebene arbeiten, ist die Entwicklung relativ aufwändig. Der Programmierer muss sich auch selbst um die Speicherverwaltung kümmern, was leicht zu Fehlern führen kann.

Um die Anwendungsentwicklung zu vereinfachen, wird von Microsoft das eigens entwickelte .NET-Framework angeboten. Mit verschiedenen Programmiersprachen wie Visual Basic und C# können damit sehr einfach grafische Oberflächen erstellt werden. Die Programme sind gewöhnlich weniger ressourcenschonend, da der Quellcode nicht mehr in Maschinencode des Prozessors, sondern in einen Bytecode für den Prozessor einer virtuellen Maschine übersetzt wird (vgl. Wikipedia 2016c).

Eine dritte Möglichkeit bieten Skriptsprachen wie Perl, Python und Php, bei denen der Quellcode zur Laufzeit direkt abgearbeitet wird, ohne dass vorher eine Kompilierung erforderlich ist. Skriptsprachen erlauben eine schnelle Softwareentwicklung auf einer sehr rechnerunabhängigen Ebene. Durch die Interpretierung zur Laufzeit ist die Ausführungsgeschwindigkeit gewöhnlich noch geringer als beim .NET-Framework.

21.2.1. Auswahl: Skriptsprache Python mit Toolkit wxWidgets

Die zu entwickelnde PC-Software stellt keine großen Anforderungen an die Ausführungsgeschwindigkeit, da keine komplexen Rechenaufgaben etc. erledigt werden müssen. Somit ist es sinnvoll, nicht auf C/C++ zurückzugreifen.

In der Auswahl verbleiben das .NET-Framework und verschiedene Skriptsprachen. Skriptsprachen wie Python bieten zusätzlich den Vorteil, dass die erstellte Software auch auf anderen Betriebssystemen als auf Windows lauffähig ist. Durch das Softwareprojekt Mono existiert allerdings auch für die meisten .NET-Programme die Möglichkeit, auf anderen Betriebssystemen ausgeführt zu werden. Die Auswahl fiel schließlich auf die Programmiersprache Python, wobei die persönliche Neigung des Autors, etwas Neues zu lernen, durchaus ausschlaggebend war.

Für Python existieren verschiedene Anbindungen an Toolkits zur Erstellung grafischer Oberflächen. In diesem Projekt soll auf das Toolkit wxWidgets bzw. dessen Python-Schnittstelle wxPython zurückgegriffen werden.

21.3. Programmstruktur

Der Programmeintrittspunkt befindet sich in der Datei main.py. Zu Beginn werden die Zeilen nach der folgenden if-Abfrage ausgeführt.

```
if __name__ == '__main__':
```

Die Abfrage bewirkt, dass der nachfolgende Code nur ausgeführt wird, wenn die Datei main.py als Hauptprogramm gestartet wird. Sollte main.py stattdessen als Modul geladen werden, so wird der Code nicht ausgeführt, da `__name__` in diesem Fall einen anderen Wert besitzt (vgl. *The Python Tutorial - Modules* 2016).

Im Hauptprogramm wird anschließend eine Instanz der Hauptklasse für die wxPython-Anwendung erzeugt und die Hauptschleife dieser Instanz aufgerufen. Von nun an reagiert das Programm nur auf Ereignisse (Events).

Die Programmbibliothek wxPython liefert eine Vielzahl an verschiedenen Steuer- und Strukturierungselementen, die in der grafischen Oberfläche verwendet werden können. Für jedes Steuerelement existiert eine eigene Klasse, die jeweils die Eigenschaften und Methoden, die auf das Steuerelement angewendet werden können, zusammenfassen. In wxPython ist es üblich, für Steuerelemente mit vielen Einstellungen und Unterelementen (wie beispielsweise einem Fenster oder einem Panel), die von wxPython zur Verfügung gestellte Klasse zu beerben.

Anstatt eine Instanz einer allgemeinen Klasse zu erzeugen, dessen Eigenschaften und Unterelemente anschließend gesetzt werden, beerbt man die Klasse und erstellt eine neue Kinderklasse, die alle Eigenschaften der Elternklasse übernimmt. Der Konstruktor dieser Klasse übernimmt die Erzeugung der Unterelemente und das Setzen der individuellen Eigenschaften. So erhält man mit dieser speziellen Klasse quasi einen Bauplan für beliebig viele Fenster, die ebenfalls die gewünschten Eigenschaften, also beispielsweise einen bestimmten Fensteraufbau und eine bestimmte Anordnung von Unterelementen, aufweisen.

21.3.1. Hauptfenster

In diesem Programm wird nach dem Programmstart eine Instanz der Klasse MainApp erzeugt, die eine durch Vererbung erweiterte Klasse der wx.App-Klasse darstellt. Jede wxPython-Anwendung benötigt eine Instanz dieser oder einer von wx.App abgeleiteten Klasse (vgl. *wxPython API Documentation - wx.App* 2016).

Die Initialisierungsfunktion der MainApp-Klasse erzeugt schließlich eine Instanz der MainFrame-Klasse. Die MainFrame-Klasse ist ein Bauplan für das Hauptfenster der Anwendung und basiert auf der Klasse wx.Frame, der Basisklasse für Programmfenster.

Das Hauptfenster selbst besteht aus einer Menüleiste oben, welche die Möglichkeit zum Abspeichern und Laden von Dateien etc. bietet, und aus einer Tableiste,

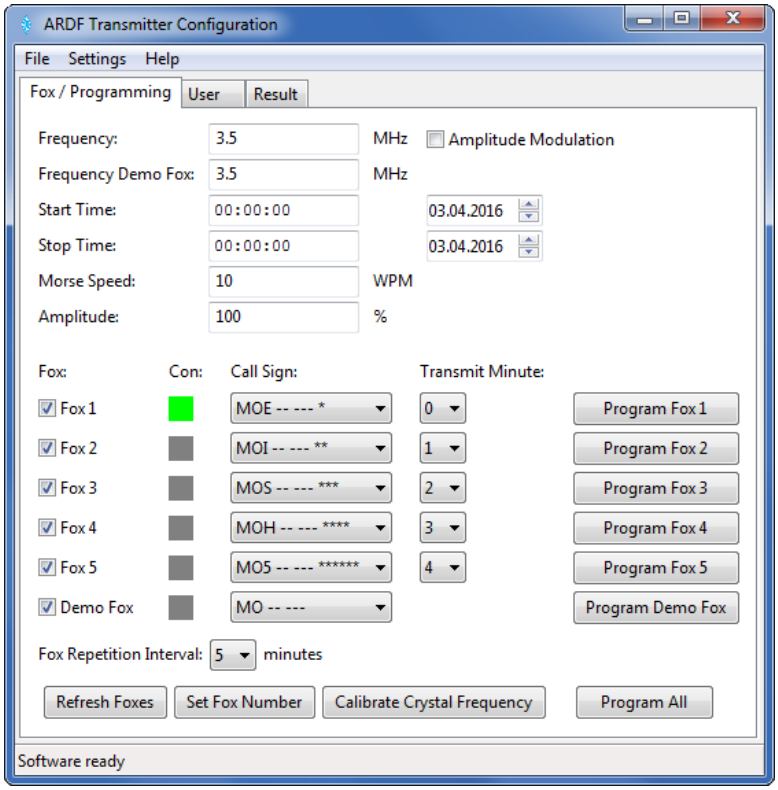


Abbildung 21.1.: Hauptfenster der PC-Software

welche die Umschaltung zwischen mehreren Unterfenstern ermöglicht (siehe Abbildung 21.1).

Die Tableiste wird im wxPython-System als Notebook bezeichnet (Klasse wx.Notebook). MainNotebook ist die angepasste Klasse, die von wx.Notebook erbt und in diesem Programm die Tabansicht generiert.

Die Tableiste besteht aus drei Unterfenstern mit den Titeln *Fox / Programming*, *User* und *Result*. Diese Unterfenster sind als Kind-Klassen der Klasse wx.Panel ausgeführt und heißen PanelFox, PanelUser und PanelResult. Ein Panel ist ein Steuerelement, in das weitere Steuerelemente eingefügt werden können (vgl. *wxPython API Documentation - wx.Panel* 2016).

Aus Übersichtsgründen sind die Klassen für die drei Ansichten in verschiedene Dateien panel_fox.py, panel_user.py und panel_result.py ausgelagert worden.

21.3.2. Panel Fox / Programming

Das Unterfenster *Fox / Programming* bietet Steuerelemente zur Konfiguration der Sendereinstellungen. Dieses Unterfenster ist standardmäßig aktiviert und somit der Teil des Programms, den der Benutzer beim Start des Programms sieht.

Im oberen Teil des Fensters werden mehrere Textboxen angelegt, die zur Frequenz-, Startzeit- und Stoppzeiteinstellung etc. dienen. Diese Einstellungen sind im Wesentlichen für alle Sender gleich und müssen daher nur einmal pro Wettbewerb eingestellt werden.

Im unteren Teil des Fensters befinden sich die Einstellmöglichkeiten wie beispielsweise Rufzeichen und Sendeminute, die für jeden Sender des Wettbewerbs unterschiedlich sind. Die mit *Program* beschrifteten Schaltflächen zur Übertragung der Einstellungen an die Sender sind ebenfalls in der unteren Hälfte des Fensters angebracht.

Signale

Steuerelemente senden in wxPython beim Auftreten von verschiedenen Ereignissen, wie z.B. dem Betätigen einer Schaltfläche, Signale aus. Diesen Signalen sind Nummern bzw. symbolische Namen wie beispielsweise `wx.EVT_BUTTON` zugeordnet.

Mithilfe der Methode `Bind` kann das Signal eines Steuerelements einer Funktion zugeordnet werden, die zukünftig aufgerufen wird, sobald das Signal auftritt. Eine solche Zuordnung innerhalb der Klasse PanelFox sieht beispielsweise wie folgt aus:

```
self.Bind(wx.EVT_BUTTON, self.TestFoxes, ↩
    button_search_foxes)
```

Die obige Zeile im Konstruktor der PanelFox-Klasse weist das Signal `wx.EVT_BUTTON` des Buttons `button_search_foxes` der Methode `TestFoxes` zu, sodass die Methode immer aufgerufen wird, wenn das Signal ausgelöst wird. Der Ausdruck `self` ist hierbei jeweils eine Referenz auf das eigene Objekt und dient dazu, Methoden des eigenen Objekts aufzurufen. In Python wird diese Referenz jeder Methode als erster Parameter übergeben und kann daher grundsätzlich auch anders benannt werden, die Bezeichnung `self` ist jedoch üblich (vgl. *The Python Tutorial - Classes* 2016).

Speichern und Laden der Einstellungen

Die Einstellungen sollen vom Benutzer auch abgespeichert werden können. Der Speichervorgang wird durch Klicken auf den entsprechenden Eintrag im *File*-Menü gestartet. Das Hauptprogramm ruft anschließend von den Objekten der drei Tabansichten jeweils die Methoden `WriteXml` bzw. `ReadXml` beim Einlesen der Datei auf.

Als Dateiformat zur Abspeicherung bietet sich das XML-Format an, da für alle gängigen Programmiersprachen Bibliotheken zum Umgang mit XML-Dateien existieren. Als textbasiertes Format kann es zudem von jedem gewöhnlichen Editor geöffnet und bearbeitet werden. In Python kann beispielsweise auf die lxml-Bibliothek zurückgegriffen werden (vgl. *Homepage lxml - XML and HTML with Python* 2016).

Die `WriteXml`-Methode liest alle vom Benutzer gewählten Einstellungen aus den Textboxen und Auswahlfeldern der Software aus und schreibt diese in entsprechende XML-Tags.

Die `ReadXml`-Methode untersucht den XML-Baum, der von der XML-Bibliothek zur Verfügung gestellt wird, und sucht nach den entsprechenden Tags, um die Einstellungen aus der Datei zu lesen und wieder in die Steuerelemente zurückzuschreiben. Sollte ein benötigter Tag nicht vorhanden sein, weil die XML-Datei bearbeitet worden ist, so wird eine Fehlermeldung angezeigt und der Ladevorgang abgebrochen.

21.3.3. Panel User

Das zweite Unterfenster der Tabansicht im Hauptfenster dient zum Eintragen der Teilnehmer einer Fuchsjagd. Die entsprechende Klasse des Unterfensters heißt PanelUser und befindet sich in der Datei panel_user.py.

Das Unterfenster enthält eine große Liste, in der die Teilnehmer mit Namen und ID eingetragen werden können. Die Schaltflächen *Add*, *Edit* und *Delete* dienen zur Verwaltung der Einträge. In der Liste wird zusätzlich neben jedem Eintrag eine Schaltfläche angezeigt, die zum Beschreiben des RFID-Transponders für den entsprechenden Teilnehmer gedacht ist. Nach dem Anklicken dieser Schaltfläche wird ein Kommando zum Sender, der im Auswahlfeld auf der rechten Seite eingestellt ist, übertragen. Der Sender schreibt die gewünschte ID auf jenen RFID-Transponder, der als Nächstes in die Nähe des RFID-Schreib-/Lesemoduls gebracht wird.

Auch die Klasse PanelUser besitzt wieder die Methoden `WriteXml` und `ReadXml`, die vom Hauptfenster aufgerufen werden, um die Teilnehmerliste in die XML-Datei zu schreiben bzw. um eine XML-Datei einzulesen.

21.3.4. Panel Result

Das dritte Unterfenster hilft beim Auswerten des Ergebnisses. Durch Drücken der Schaltfläche *Listen* beginnt das Programm, die serielle Schnittstelle des eingestellten Fuchses abzuhören und zu beobachten, ob Transponderdaten empfangen werden.

Da neben dieser Aufgabe die grafische Oberfläche weiterhin einsatzfähig sein soll, wird ein Parallelthread gestartet, welcher die Kommunikation mit der Schnittstelle im Hintergrund übernimmt. Der Code zur Threaderstellung ist dem Beispiel der Website wxPyWiki entnommen (vgl. *Non-Blocking Gui wxPyWiki* 2016).

Die Aussendung des Mikrocontrollers bei einem neu erkannten Tag beginnt mit der Zeile `--- NEW TAG 0x55005500 ---` als Erkennungszeichen und endet mit `--- END TAG 0x55005500 ---`. Dazwischen werden mehrere Zeilen mit den Informationen über die Geheimzahlen der Sender und über die Teilnehmerhistorie gesendet. Das PC-Programm empfängt diese Daten und trägt den Transponder in die dafür vorgesehene Liste ein.

Die Geheimzahlen der einzelnen Sender werden durch einen Zufallsgenerator des PC-Programms erzeugt und vor der Fuchsjagd zu den Sendern übertragen. Beim Auswerten der Transponderdaten vergleicht die Software, ob die jeweiligen Geheimzahlen mit den ursprünglich erzeugten übereinstimmen. Die letzte Spalte in der Liste mit dem Titel *Secrets correct* enthält fünf Buchstaben, die für die fünf möglichen Sender stehen. Ein *y* bedeutet, dass die Geheimzahl übereinstimmt. Ist sie falsch, steht stattdessen *n* an der entsprechenden Stelle.

Am Transponder selbst sind nur die Zeitpunkte der Ankunft bei den Sendern gespeichert. Die Ankunftszeit im Ziel muss manuell festgestellt werden. Sie kann über die Schaltfläche *Set Finish Time* in die Liste eingetragen werden.

Nach dem Einlesen aller Transponder bietet die Schaltfläche *Export CSV* eine Exportmöglichkeit.

21.3.5. Dialoge

Das PC-Programm verwendet verschiedene Dialoge z.B. zum Einstellen der Nummer eines Senders, zum Kalibrieren der Quarzfrequenz oder zum Anlegen eines

neuen Teilnehmers für die Teilnehmerliste. Sie sind als eigene Klassen ausgeführt, die jeweils wieder von der Basisklasse wx.Dialog erben. Alle Dialoge sind in der Datei fox_dialogs.py zu finden.

Um einen Dialog anzuzeigen, muss ein Objekt der entsprechenden Klasse erzeugt und die Methode `ShowModal` dieses Objekts aufgerufen werden.

21.3.6. Verbindung zur COM-Schnittstelle

Der Sender wird über die USB-Schnittstelle mit dem Computer verbunden. Der USB-Baustein FT232 am Sender wird vom PC als virtueller COM-Port erkannt und kann aus Sicht der Software wie eine gewöhnliche serielle Schnittstelle verwendet werden.

In der Standardausstattung besitzt Python keine plattformunabhängige Möglichkeit, auf die serielle Schnittstelle zuzugreifen. Deshalb muss eine Bibliothek namens PySerial nachinstalliert werden (vgl. *PySerial Documentation* 2016).

In der Datei fox_serial.py sind Funktionen zum Öffnen und Schließen der seriellen Schnittstelle definiert, die wiederum auf die Funktionen der PySerial-Bibliothek zugreifen, aber zusätzlich bei Fehlern die korrekte Fehlermeldung ausgeben und teilweise die Funktion der Bibliothek erweitern. Zum Beispiel liest die Funktion `ReadToPrompt` solange Zeilen über die serielle Schnittstelle ein, bis der Prompt des Senders empfangen wird oder bis eine eingestellte Zeit ohne Empfang einer neuen Zeile abgelaufen ist.

Die Datei fox.py definiert aufbauend auf der fox_serial-Bibliothek weitere Funktionen für häufig benötigte Aufgaben wie z.B. das Auslesen der Sendernummer.

21.3.7. Programmvariablen

Python kennt anders als beispielsweise C kein Konzept von globalen Variablen, deren Gültigkeitsbereich sich über mehrere Module hinweg erstreckt. Stattdessen befinden sich alle global benötigten Konstanten in einem eigenen Modul settings.py. In anderen Programmteilen kann über den Ausdruck `settings.KONSTANTENNAME` auf die Konstanten zugegriffen werden.

21.4. Python Installation - Test der PC-Software

Das Programm liegt vorerst als Quellcode vor. Um den Code auszuführen und zu testen, muss der Python-Interpreter gemeinsam mit den benötigten Bibliotheken wxPython, lxml und PySerial auf dem PC installiert werden.

Python kann als MSI-Installer-Datei von der Python Website heruntergeladen werden (vgl. *Python Releases for Windows* 2016). Der Test dieser Software erfolgte bei der Ausarbeitung mit Python Version 2.7. Um spätere Probleme zu vermeiden, ist es sinnvoll, als Installationsort einen Pfad auszuwählen, der keine Leerzeichen enthält. Soll später aus dem Python-Quellcode eine Exe-Datei erzeugt werden (vgl. Abschnitt 21.4.1 auf der nächsten Seite), so sollte die 32-bit Version von Python auch auf einem 64-bit Windows-System gewählt werden, da das dafür verwendete Programm PyInstaller die 64-bit Version nicht unterstützt (Stand: März 2016).

WxPython bietet ebenfalls Installationsdateien auf der Website an (vgl. *wxPython Windows Binaries* 2016). Hierbei sollte die passende Version für Python 2.7 ausgewählt werden.

Nach der Python-Installation kann in der Windows-Kommandozeile der Python Paketmanager *pip* aufgerufen werden. Mithilfe des Befehls *pip install pyserial* kann die PySerial-Bibliothek installiert werden (vgl. *PySerial Installation* 2016). Eventuell muss die Kommandozeile als Administrator ausgeführt werden, damit die Installation erfolgreich absolviert werden kann.

Die Installation der lxm-Bibliothek über den pip-Paketmanager hat beim Test durch die Autoren nicht funktioniert. Auf der Python Website (vgl. *LXML 3.3.5* 2016) finden sich ersatzweise ausführbare Installer-Dateien für die entsprechenden Python Versionen (auch hier sollte wieder die zu Python 2.7 passende Version gewählt werden).

Ist bei der Python-Installation ausgewählt worden, dass .py-Dateien mit dem Python-Interpreter verknüpft werden sollen, so lässt sich die PC-Software durch einen Doppelklick auf die Datei main.py starten. Andernfalls muss in einer Kommandozeile mithilfe des Befehls *cd* in den Ordner navigiert werden, in dem sich die main.py-Datei befindet und dort der Befehl *python main.py* ausgeführt werden.

Linux: Unter den meisten Linux-Distributionen werden die benötigten Bibliotheken über die Paketverwaltung angeboten. Die Pakete heißen beispielsweise python-serial unter Debian für die PySerial Bibliothek. Nach Installation der Pakete für PySerial, wxPython, lxml und Python selbst über die Paketverwaltung

kann das Programm mit dem Befehl *python main.py* in der Kommandozeile gestartet werden.

21.4.1. Auslieferung der PC-Software

Für den Endbenutzer ist es unpraktisch, wenn er erst umständlich den Python-Interpreter und die erforderlichen Bibliothken aus dem Internet herunterladen und installieren muss, um die PC-Software ausführen zu können. Einfacher ist es, wenn eine Exe-Datei bereitsteht, die wie gewohnt über einen Doppelklick ausgeführt werden kann. Die Tatsache, dass intern die Programmiersprache Python verwendet wird, ist für den Benutzer irrelevant.

Um aus dem Python-Quellcode eine ausführbare Exe-Datei zu erzeugen, existieren verschiedene Programme im Internet. In diesem Projekt kommt das Programm *pyinstaller* zum Einsatz (vgl. *PyInstaller Official Website* 2016).

Zur Installation des Programms PyInstaller kann wieder der Python Paketmanager verwendet werden. Dazu muss die Kommandozeile mit Administratorrechten geöffnet und der Befehl *pip install pyinstaller* ausgeführt werden, der alle benötigten Pakete herunterlädt und installiert. Anschließend kann in der Kommandozeile mithilfe des Befehls *pyinstaller --onefile --windowed --icon=oevsv_logo_med.ico main.py* die Exe-Datei erzeugt werden.

Der Schalter *--onefile* stellt ein, dass eine einzelne Exe-Datei erzeugt wird, die alle benötigten dynamischen Bibliotheken etc. bereits enthält und ohne Installation ausgeführt werden kann. Alternativ kann auch ein ganzer Ordner erstellt werden, in dem sich alle benötigten Dateien befinden. Dieser Ordner kann anschließend als Ausgangspunkt für die Erstellung einer Setup-Routine verwendet werden, die alle Dateien dieses Ordners in den entsprechenden Programmordner des Windows-Systems kopiert (vgl. *PyInstaller Manual* 2016).

Mit *--windowed* wird eingestellt, dass es sich um eine grafische Anwendung handelt, die kein Konsolenfenster benötigt. Fehlt der Schalter, so erscheint beim Öffnen der Anwendung jeweils ein Kommandozeilenfenster im Hintergrund.

Mit *--icon=oevsv_logo_med.ico* wird das Icon der Exe-Datei, das in der Taskleiste angezeigt wird, eingestellt.

Im ersten Durchgang erstellt das Programm pyinstaller eine Datei *main.spec*, die nun bearbeitet werden kann, um zwei weitere Dateien zur Exe-Datei hinzuzufügen. Die Zeile

```
datas=None,
```

soll durch die folgende Zeile ersetzt werden:

```
datas=[('user_manual.pdf', '.'), ('oevsv_logo_med.ico', ↩
    '.')],
```

Die PDF-Datei dient als Benutzerdokumentation für das Programm und kann innerhalb der Software über das Hilfe-Menü aufgerufen werden. Die Icon-Datei wird als Symbolbild in der Taskleiste und im Fenster verwendet. Beide Dateien müssen im gleichen Ordner wie die main.spec-Datei liegen. Anschließend wird ein zweiter Durchlauf mit dem Befehl *pyinstaller --onefile --windowed --icon=oevsv_logo_-med.ico main.spec* gestartet.

Die so erstellte Exe-Datei sollte auf jedem Windows-PC ohne die vorherige Installation eines Python Interpreters oder einer der verwendeten Bibliotheken lauffähig sein. Da in der Exe-Datei ein gesamter Python-Interpreter und alle benötigten Bibliotheken enthalten sind, besitzt die ausführbare Datei insgesamt eine Größe von einigen Megabytes.

22. Weiterführende Überlegungen und Arbeiten

22.1. Amplitudenmodulation

Eine Herausforderung war die Implementierung der Amplitudenmodulation, da diese und das RFID-System über den gleichen SPI-Bus abgewickelt werden. Das RFID-Modul muss während einer Fuchsjagd gleichzeitig mit der Modulation aktiviert sein. Für die Kommunikation mit dem RFID-Modul bleiben nur schmale Zeitschlitze zwischen der für die Amplitudenmodulation erforderlichen Übertragungen.

Mit dem in diesem Projekt realisierten System können Amplitudenmodulation und RFID-Modul parallel verwendet werden. Eine aktive RFID-Kommunikation führt jedoch zu leichten Verzerrungen der Amplitudenmodulation, die beim Empfang akustisch wahrnehmbar sind. So ist es bei aktivierter Amplitudenmodulation möglich, einen Teilnehmer zu „hören“, während er sich mit seinem RFID-Transponder am Sender registriert (falls der Fuchs sich derzeit in einer Aussendungsphase befindet).

Rückblickend wäre ein zusätzlicher Controller, der die Amplitudenmodulation über einen eigenen SPI-Bus abwickelt, die einfachere und weniger Probleme verursachende Option gewesen.

22.2. Energiesparmodus

Der Sender wird gewöhnlich einige Zeit vor Beginn des Wettbewerbs im Gebiet ausgelegt. Vor dem Wettbewerb soll der Sender möglichst wenig Strom aufnehmen, um den Akku nicht zu stark zu entladen. Dazu sollen möglichst viele Bausteine auf dem Sender, soweit unterstützt, in einen stromsparenden Modus versetzt werden. Aus Zeitgründen konnte diese Funktion bis zum Abschluss der Arbeit nicht umfassend implementiert werden.

22.3. Fuchshistorie

Jeder Sender speichert die IDs der RFID-Transponder der ankommenden Teilnehmer gemeinsam mit dem Zeitpunkt der Ankunft. Diese Informationen können auch über den entsprechenden Kommandozeilenbefehl vom Sender abgerufen werden. Bisher ist jedoch keine Funktion zur Auswertung der Senderhistorie in der PC-Software eingebaut.

Teil III.

Benutzerdokumentation

23. Konfigurationssoftware

23.1. Installationsanleitung / Inbetriebnahme

23.1.1. Treiberinstallation

Die folgende Beschreibung richtet sich nach der Vorgehensweise in den Betriebssystemen Windows 7 und Windows 10.

Der Sender verfügt zur Konfiguration über den Computer über eine USB-Schnittstelle. Nach Anschluss der Stromversorgung an den Sender und nach dem Herstellen der Verbindung über die USB-Schnittstelle, versucht Windows automatisch den richtigen Treiber herunterzuladen und zu installieren.

Sobald der Treiber korrekt installiert ist, sollte das Gerät als neuer COM-Port im Gerätemanager aufscheinen (vgl. Abbildung 23.1 auf der nächsten Seite)

Sollte die Treiberinstallation nicht automatisch starten, kann alternativ der Herstellertreiber für den USB-Baustein von der Website der Firma FTDI heruntergeladen und manuell installiert werden (vgl. *Virtual COM Port Drivers - FTDI Ltd.* 2016).

23.1.2. PC-Programm starten

Das PC-Programm befindet sich auf der beiliegenden CD als ausführbare Exe-Datei. Die Software muss nicht installiert werden. Es reicht, die Exe-Datei auf den Desktop oder in einen beliebigen anderen Ordner zu kopieren und das Programm mit Doppelklick zu starten.

Daraufhin erscheint das Hauptfenster der PC-Software (vgl. Abbildung 23.2 auf der nächsten Seite)

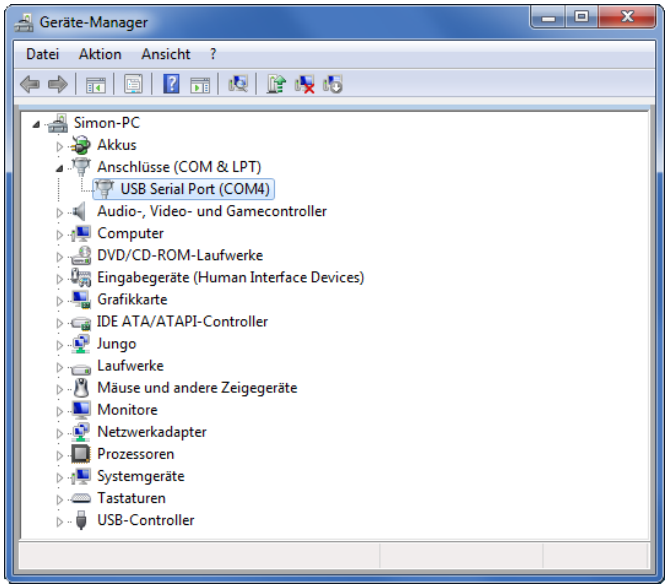


Abbildung 23.1.: Virtueller COM-Port im Gerätemanager

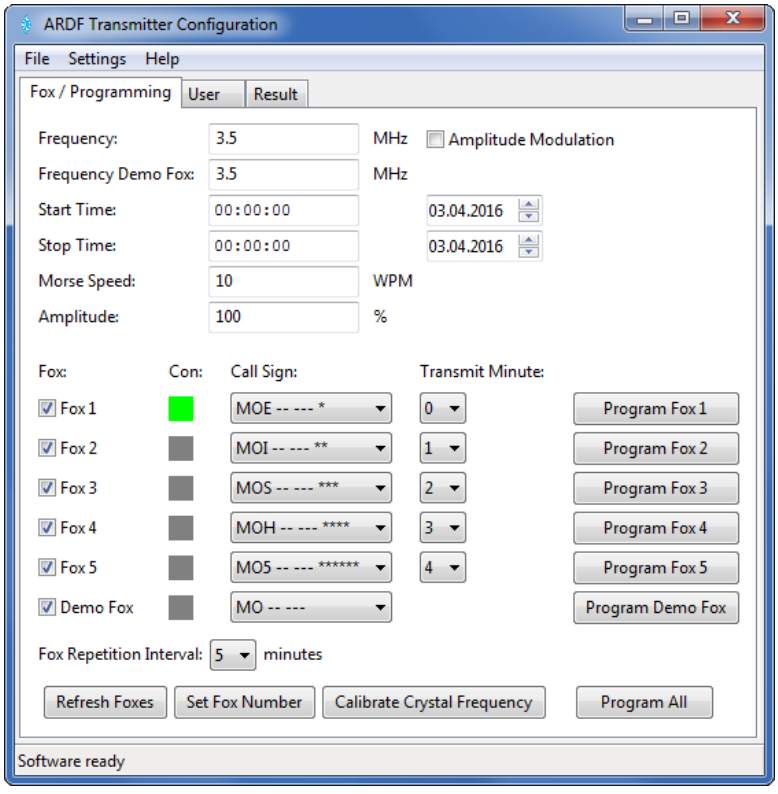


Abbildung 23.2.: Hauptfenster der PC-Software

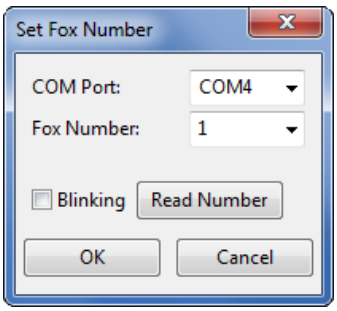


Abbildung 23.3.: Dialog zum Setzen der Fuchsnummer

23.2. Setzen der Fuchsnummer

Die verschiedenen Sender eines Wettbewerbs werden vom PC-Programm durch eine Nummer unterschieden. Die Nummer erlaubt es, die Sender zu identifizieren und somit mehrere Sender gemeinsam anzuschließen und zu konfigurieren.

Zum Einstellen der Nummer eines Senders ist es empfehlenswert, nur den einzustellenden Sender mit dem PC zu verbinden, da so der COM-Port vom Benutzer leichter identifiziert werden kann.

Durch Anklicken des Buttons *Set Fox Number* kann der entsprechende Dialog aufgerufen werden (vgl. Abbildung 23.3). Im Auswahlfeld *COM Port* muss die entsprechende COM-Schnittstelle, die dem Sender zugeordnet ist, eingestellt werden. Um zu überprüfen, ob der richtige COM-Port gewählt worden ist, kann die Checkbox *Blinking* angeklickt werden, wodurch die RFID-LED des ausgewählten Senders zu blinken beginnt.

Im unteren Auswahlfeld *Fox Number* muss nun eine Nummer zwischen 1 und 5 oder die Auswahl *Demo* gewählt werden. Die Sender 1 bis 5 sind gleichwertig. Die Auswahl *Demo* ist für den Sender gedacht, der am Startort des Wettbewerbs aufgestellt wird und dort als Bakensender arbeitet. Es ist zweckmäßig, aber nicht erforderlich, die Nummer entsprechend dem später verwendeten Rufzeichen zu vergeben, (Nummer 1 für MOE, Nummer 2 für MOI usw.)

Durch Bestätigung des Dialogs mit dem *OK*-Button wird die Nummer zum Sender übertragen und dort abgespeichert.

23.3. Konfiguration der Einstellungen für den Wettbewerb

Nachdem alle für den Wettbewerb benötigten Sender mit einer Nummer versehen worden sind, können im Programm die Wettbewerbseinstellungen getroffen werden.

Im oberen Programmteil befinden sich alle Einstellungen, die für mehrere Sender gemeinsam gelten, wie Frequenz, Start- und Endzeitpunkt des Wettbewerbs, Geschwindigkeit beim Morsen und Amplitude des Ausgangssignals.

Im unteren Teil können die individuellen Einstellungen für jeden Sender getroffen werden. Die Ziffern bei Fuchs 1, Fuchs 2 etc. beziehen sich auf die oben eingestellte Nummer der jeweiligen Sender. Jedem Sender kann ein eigenes Rufzeichen und eine Minute zum Senden (*Transmit Minute*) zugewiesen werden. Beim Demo-Sender fehlt die Auswahl der Sendeminute, da dieser Sender unterbrechungsfrei sendet.

Der Wert 0 für die Sendeminute bedeutet, dass der Sender gleich zu Beginn der Fuchsjagd sendet, beim Wert 1 beginnt der Sender erst eine Minute nach Beginn der Fuchsjagd usw. Über das Auswahlfeld *Fox Repetition Interval* kann eingestellt werden, nach wie vielen Minuten sich der Durchlauf wiederholt. Ist hier beispielsweise der Wert 5 eingestellt, so sendet jeder Fuchs alle 5 Minuten für jeweils eine Minute sein Rufzeichen aus.

Die freie Zuordnung von Fuchsnummer zu Rufzeichen und Sendeminute erlaubt eine größerer Flexibilität. Soll eine Fuchsjagd mit beispielsweise nur drei Sendern ausgerichtet werden, so können die Checkboxen vor zwei beliebigen Sendern deaktiviert, das Wiederholungsintervall auf 3 Minuten gestellt und die gewünschten Rufzeichen und Sendeminuten den einzelnen verwendeten Füchsen zugeteilt werden.

23.3.1. Übertragen der Einstellungen

Nachdem die Einstellungen getroffen sind, können alle Sender mit dem PC verbunden werden. Durch Klicken des Buttons *Refresh Foxes* sucht der PC auf allen erkannten COM-Ports nach angeschlossenen Sendern. Alle erkannten Sender werden durch ein grünes Quadrat in der Software gekennzeichnet.

Gewöhnlich sucht das Programm auch beim Start der Software nach Sendern, die mit dem PC verbunden sind. Durch das Entfernen des Hakens *Search Connected Foxes at Start* im Menü *Settings* kann dieses Verhalten deaktiviert werden, sodass

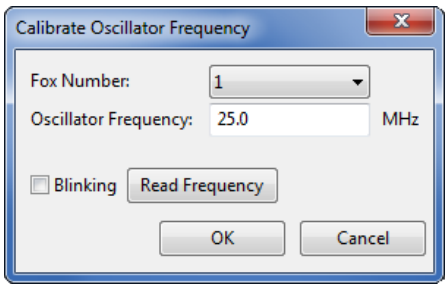


Abbildung 23.4.: Dialog zur Kalibrierung der Quarzfrequenz

nach dem Starten manuell die Schaltfläche *Refresh Foxes* angeklickt werden muss, um angeschlossene Sender zu identifizieren.

Die Einstellungen können nun entweder für jeden Sender einzeln durch Anklicken des entsprechenden *Program*-Buttons oder gemeinsam durch Klicken auf *Program All* übertragen werden.

Bei einem Klick auf *Program All* werden nur jene Füchse beschrieben, deren Checkbox aktiviert ist.

23.3.2. Kalibrierung der Quarzfrequenz

Der Signalgeneratorbaustein bezieht seine Referenzfrequenz von einem Schwingquarz. Die Genauigkeit des generierten Hochfrequenzsignals hängt somit von der Genauigkeit der Quarzfrequenz ab. Um geringfügige Abweichungen der tatsächlichen Quarzfrequenz von der am Quarz angegebenen Frequenz (25 MHz) korrigieren zu können, kann am Sender ein Kalibrierfaktor eingestellt werden.

Dazu muss im Hauptfenster die Schaltfläche *Calibrate Crystal Frequency* ausgewählt werden. Es erscheint ein Dialog zur Einstellung der Kalibrierung (siehe Abbildung 23.4). Im Dialog ist einerseits die Nummer des zu kalibrierenden Senders und andererseits die tatsächliche Frequenz des Schwingquarzes einzustellen. Durch Klicken auf *OK* wird die gewählte Einstellung zum Sender übertragen.

Zum Ermitteln der tatsächlichen Frequenz des Schwingquarzes werden im ersten Schritt 25 MHz als Quarzfrequenz konfiguriert und ein HF-Signal ausgegeben. Die Frequenz des vom Sender erzeugten HF-Signals muss nun möglichst genau gemessen werden. Mithilfe der prozentuellen Abweichung der Frequenz des Ausgangssignals von der eingestellten Ausgangssignalfrequenz kann die tatsächliche Schwingquarzfrequenz ermittelt werden, die schließlich zum Sender übertragen wird.

23.3.3. Abspeichern der Einstellungen am PC

Zur späteren Wiederverwendung können die gewählten Einstellungen am PC über das *File*-Menü als Datei abgespeichert werden.

Die Software verwendet zur Speicherung das XML-Format, sodass die Einstellungen aus der Datei mit einem gewöhnlichen Editor ausgelesen und auch manuell geändert werden können.

23.4. Vorbereiten der RFID-Transponder

Im Tab *User* können die Teilnehmer der Fuchsjagd eingetragen werden. Jeder Teilnehmer bekommt eine einzigartige ID zugewiesen, die auf den RFID-Transponder geschrieben wird, mit der sich der Teilnehmer bei jedem gefundenen Sender registriert.

Um nach dem Wettbewerb eine Zuordnung von den ID-Nummern zu den Namen der Teilnehmer zu erhalten, ist es wichtig, dass die Einstellungen nach dem Eintragen aller Namen über das *File*-Menü abgespeichert werden.

Mittels *Add*, *Edit* und *Delete* können die Einträge bearbeitet werden (siehe 23.5 auf der nächsten Seite)

Benötigt werden RFID-Transponder vom Typ MIFARE Classic 1K. Zum Einrichten eines RFID-Transponders muss der Button *Program Tag* neben dem entsprechenden Namen ausgewählt werden. Der im Auswahlfeld *Fox* ausgewählte Sender wartet anschließend so lange, bis der nächste RFID-Transponder in der Reichweite erkannt wird und beschreibt diesen mit der eingestellten ID. Das Programm setzt anschließend automatisch die Checkbox *Programmed*, um zu kennzeichnen, dass bereits ein Tag für diesen Teilnehmer erstellt worden ist.

23.5. Auswerten der Ergebnisse

Nach der Fuchsjagd können die RFID-Transponder der Teilnehmer ausgelesen werden, um festzustellen, ob und zu welchem Zeitpunkt die Füchse aufgespürt worden sind. Ein beliebiger Fuchs muss dazu über die USB-Schnittstelle mit dem PC verbunden und der Button *Refresh Foxes* im Tab *Fox / Programming* angeklickt werden. Anschließend wird im Tab *Result* der vorher angeschlossene Fuchs ausgewählt und die Schaltfläche *Listen* angeklickt. Nun können die Transponder der Reihe nach in die Nähe des RFID-Moduls vom Fuchs gebracht werden. Das

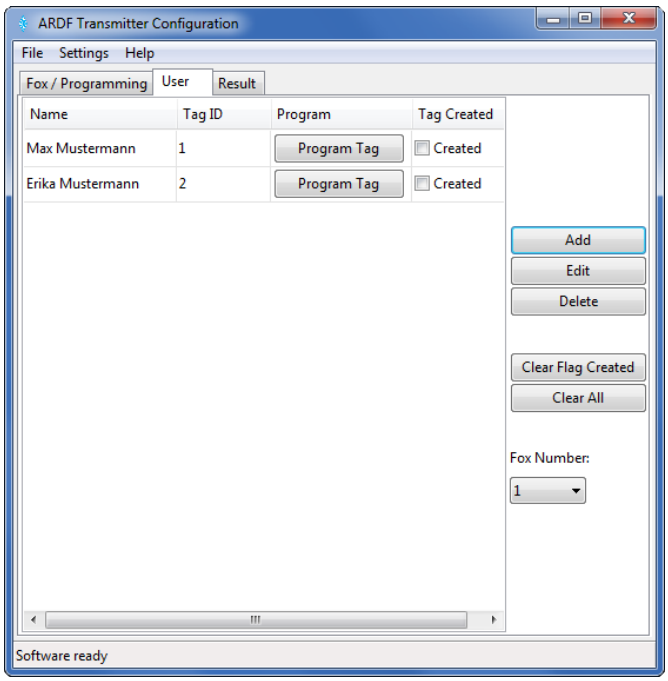


Abbildung 23.5.: Teilnehmeransicht der PC-Software

Blinken der Status-LED des Fuchses signalisiert den Abschluss des Lesevorgangs. Kurze Zeit nach dem Blinken der LED erscheint ein neuer Eintrag in der Liste im Tab *Result*.

Die ausgelesene Tag-ID wird von der Software mit der Namensliste aus dem Tab *User* verknüpft, sodass der Name des Teilnehmers angezeigt werden kann. Die Spalten *Fuchs 1* bis *Fuchs 5* enthalten die Zeitpunkte, zu denen sich der Teilnehmer bei den entsprechenden Sendern registriert hat. Die Nummer entspricht der mittels *Set Fox Number* eingestellen Fuchsnummer. Die Einlaufzeit ins Ziel muss manuell ermittelt und mithilfe des Buttons *Set Finish Time* eingetragen werden (siehe 23.6 auf der nächsten Seite).

Hinweis: Damit die Namen und Zeitpunkte in der Liste korrekt angezeigt werden können, muss die Datei geladen werden, die zur Konfiguration der Transponder und der Sender erstellt worden ist.

Die letzte Spalte der Liste enthält fünf Buchstaben, die durch y für yes bzw. n für no kennzeichnen, ob die entsprechende Geheimzahl am Transponder mit der am Fuchs eingestellten Geheimzahl übereinstimmt. Die Geheimzahl dient zur Überprüfung, ob der Teilnehmer den Sender tatsächlich aufgespürt hat. Wiederum ist es hier wichtig, dass die gleiche Datei geladen ist, die bei der Konfiguartion

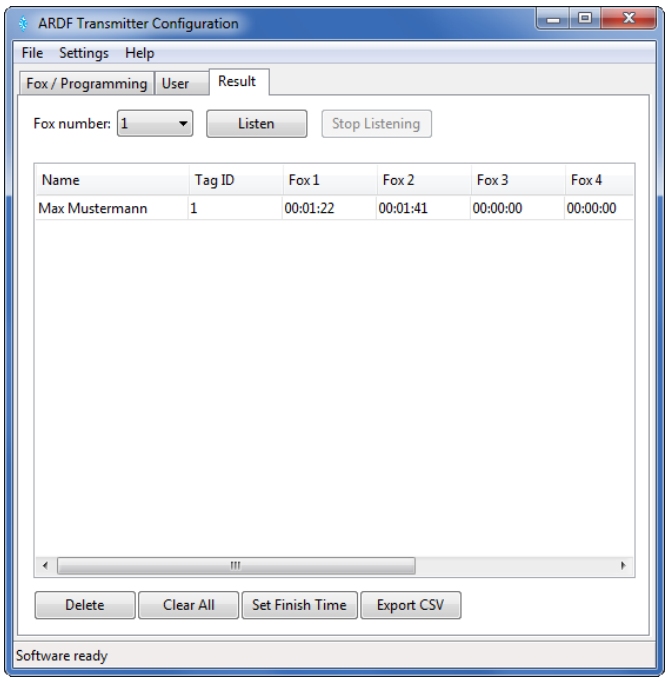


Abbildung 23.6.: Result-Ansicht der PC-Software

der Sender erstellt worden ist, da diese die korrekten Geheimzahlen enthält. Ein n an der Stelle eines Senders, der nicht in der Fuchsjagd verwendet worden ist, ist unerheblich. Ein n an einer anderen Stelle kann mitunter auf einen Manipulationsversuch hinweisen. Ausschlaggebend für die Position des Buchstabens ist nicht das Rufzeichen, sondern die mit *Set Fox Number* im Tab *Fox / Programming* eingestellte Fuchsnummer.

24. Inbetriebnahme und Abgleich

Vor der ersten Inbetriebnahme müssen beide Leistungsverstärker abgestimmt werden. Weiters sollte zuerst die korrekte Funktionsweise des Schaltreglers überprüft werden, um eine Beschädigung des Senders im Fehlerfall zu vermeiden.

24.1. Schaltregler

Der 3.3 V-Schaltregler ist insofern ein kritisches Bauteil, als dass die gesamte Digitalelektronik von diesem aus versorgt wird. Sollte er aufgrund eines Fehlers im Aufbau nicht funktionieren und die 12 V-Eingangsspannung direkt mit der 3.3 V-Schiene verbinden, so hätte dies die Zerstörung sämtlicher digitaler Bausteine auf allen Platinen zur Folge.

Aus diesem Grund wird der 0 Ω-Widerstand $R116$ anfangs noch nicht bestückt, und somit der Schaltregler vom Rest der Schaltung getrennt. Anschließend kann am Glättungskondensator $C110$ die Spannung mit einem Multimeter gemessen werden. Diese sollte bei korrekter Funktion in etwa

$$U_{\mathrm{C110}} \approx 3.3\,\mathrm{V} \tag{24.1}$$

betragen.

Ist dies der Fall, kann für $R116$ ein 0 Ω-Widerstand oder eine Drahtbrücke eingelötet werden.

24.2. Abgleich 80m-Leistungsverstärker

Da im 80m-Leistungsverstärker nur ein abzustimmendes Element, der Übertrager TR201, vorhanden ist, reicht für den Abstimmvorgang ein einfaches Leistungsmessgerät aus. Der Fuchs wird mit 12 V versorgt, sowie die Gesamtstromaufnahme mithilfe eines Amperemeters kontrolliert. Der Aufbau ist in Abbildung 24.1 auf der nächsten Seite dargestellt.

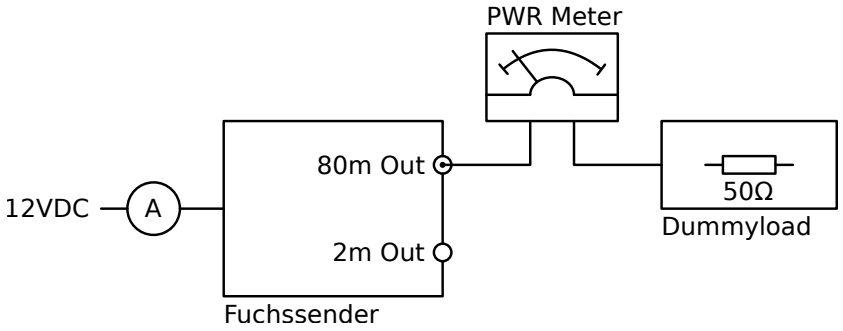


Abbildung 24.1.: Aufbau zum Abgleich des 80m-Teils

Zur Aussendung eines Trägers wird der 80m-Test-Sende Taster auf der RFID-Platine kurz gedrückt, der DDS generiert jetzt ein reines Trägersignal mit einer Frequenz von 3.55 MHz bei maximaler Amplitude. Nun wird der Kern des Übertragers TR201 mithilfe eines geeigneten, nichtmagnetischen Schraubendrehers auf maximale Leistung am PWR-Meter abgeglichen (siehe Abbildung 24.3 auf Seite 214). Dies sollten in optimaler Stellung ca. 5 W sein.

Bei dieser maximalen Leistung sollte der Stromverbrauch des Fuchses am Amperemeter in der Größenordnung von

$$I_{\mathrm{B}} = 600\,\mathrm{mA} \ldots 900\,\mathrm{mA} \tag{24.2}$$

liegen.

Weitere Schritte sind zum Abgleich des 80m-Leistungsverstärkers nicht notwendig.

24.3. Abgleich 2m-Leistungsverstärker

Beim 2m-Leistungsverstärker ist der Abgleich etwas aufwendiger, da drei Baugruppen mit insgesamt vier Trimmern (siehe Abbildung 24.3 auf Seite 214) abzustimmen sind:

- Eingangsfilter ($C263$, $C266$)
- Vorverstärker ($C218$)
- Oberwellenfilter ($C251$)

Da bei der ersten Inbetriebnahme mitunter alle Trimmer verstellt sind, kann das Ausgangssignal zu Beginn u. U. nur mit einem Spektrumanalysator (bzw. eventuell Oszilloskop mit 50 Ω-Abschluss) erfasst werden. Der Aufbau ist in Abbildung 24.2 dargestellt.

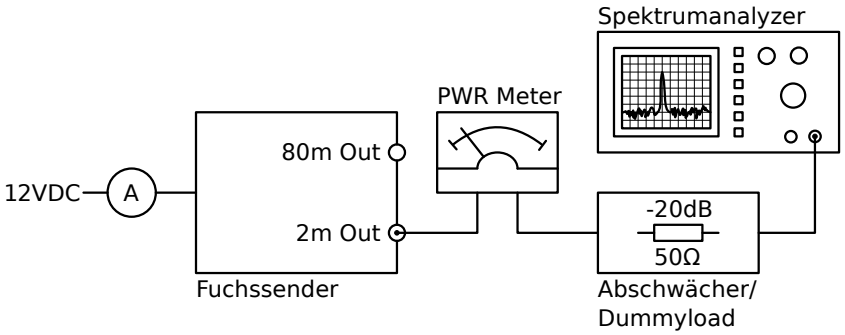


Abbildung 24.2.: Aufbau zum Abgleich des 2m-Teils

Zur Aussendung eines Trägers auf 144.7 MHz wird der 2m-Test-Sende Taster auf der RFID-Platine betätigt. Anschließend müssen alle vier Trimmer hintereinander mit Blick auf den Spektrumanalysator (maximale Signalamplitude) abgeglichen werden, siehe auch Abbildung 24.3 auf der nächsten Seite.

Am PWR-Meter sollte eine Ausgangsleistung von ca. 1 W angezeigt werden. Der Gesamtstromverbrauch sollte wieder in der Größenordnung von ca.

$$I_{\mathrm{B}} = 600\,\mathrm{mA} \ldots 900\,\mathrm{mA} \tag{24.3}$$

liegen.

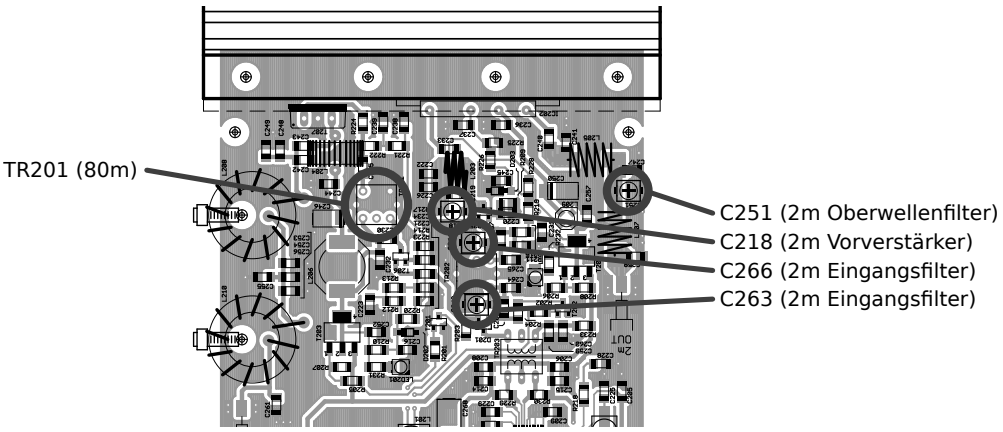


Abbildung 24.3.: Lage des Übertragers und der Trimmer auf der Platine

Literatur

24AA512 Serial EEPROM Datasheet (2010). Microchip Technology. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/21754M.pdf (besucht am 02. 01. 2016).

2SC1971 NPN RF Power Transistor Datasheet (1997). Mitsubishi Electric Semiconductor. URL: http://pdf1.alldatasheet.com/datasheet-pdf/view/12/MITSUBISHI/2SC1971.html (besucht am 02. 01. 2016).

74HC595 8-bit Shift Register Datasheet (2004). NXP. URL: https://www.sparkfun.com/datasheets/IC/SN74HC595.pdf (besucht am 03. 03. 2016).

AD9859 Direct Digital Synthesizer Datasheet (2009). Rev. A. Analog Devices, Inc. URL: www.analog.com/media/en/technical-documentation/data-sheets/AD9859.pdf (besucht am 31. 12. 2015).

AD9859 Evaluation Board Schematics (2004). Analog Devices. URL: http://www.analog.com/media/en/technical-documentation/evaluation-documentation/46543170023221AD9954_RevD_schem.pdf (besucht am 09. 02. 2016).

AD9958 Direct Digital Synthesizer Datasheet (2013). Analog Devices. URL: http://www.analog.com/media/en/technical-documentation/data-sheets/AD9958.pdf (besucht am 12. 02. 2016).

Amidon Iron Powder Cores Family Datasheet (2007). Amidon Associtates, Inc. URL: http://cdn-reichelt.de/documents/datenblatt/B400/Amidon%5C%23AMI.pdf (besucht am 06. 01. 2016).

Änderungen der Vorordnung zur Durchführung des Amateurfunkgesetzes (2008). Republik Österreich. URL: http://www.bmvit.gv.at/telekommunikation/recht/aut/verordnungen/downloads/afv/2008390.pdf (besucht am 02. 01. 2016).

Arduino Mifare MFRC522 RFID Reader/Writer (2016). URL: http://playground.arduino.cc/Learning/MFRC522 (besucht am 24. 02. 2016).

ATmega 644 Microcontroller Datasheet (2012). Atmel Corporation. URL: http://www.atmel.com/Images/doc2593.pdf (besucht am 01. 01. 2016).

Atmel Studio 7 (2016). URL: http://www.atmel.com/Microsite/atmel-studio/ (besucht am 02. 03. 2016).

AVR Hardware Design Considerations (2015). Atmel. URL: http://www.atmel.com/images/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf (besucht am 14. 02. 2016).

AVR8 Burn-O-Mat - a GUI for avrdude (2016). URL: http://avr8-burn-o-mat.aaabbb.de/avr8_burn_o_mat_avrdude_gui_en.php (besucht am 02. 04. 2016).

AVRDUDE Documentation (2016). URL: http://www.nongnu.org/avrdude/user-manual/avrdude.html (besucht am 03. 03. 2016).

BC847 NPN general-purpose transistor Datasheet (2014). NXP. URL: http://www.nxp.com/documents/data_sheet/BC847_SER.pdf (besucht am 05. 01. 2016).

BFR92 NPN Planar RF Transistor Datasheet (2000). Vishay Semiconductors. URL: http://www.impedanca.si/media/BFR92.pdf (besucht am 10. 02. 2016).

CM7V-T1A Tuning Fork Crystal Datasheet (2010). Micro Crystal Switzerland. URL: http://docs-europe.electrocomponents.com/webdocs/0f59/0900766b80f59f50.pdf (besucht am 14. 02. 2016).

Design Tools: ADIsimDDS (2016). Analog Devices. URL: http://www.analog.com/designtools/en/simdds/dtddsmain.aspx (besucht am 26. 02. 2016).

Direct Digital Synthesis (2014). Wikipedia. URL: https://de.wikipedia.org/wiki/Direct_Digital_Synthesis (besucht am 12. 02. 2016).

Dokumentation eeprom.h AVR Libc (2016). URL: http://www.nongnu.org/avr-libc/user-manual/group__avr__eeprom.html (besucht am 02. 01. 2016).

Dokumentation twi.h AVR Libc (2016). URL: http://www.nongnu.org/avr-libc/user-manual/util_2twi_8h.html (besucht am 04. 01. 2016).

FT232BL USB-Serial Converter Datasheet (2011). Future Technology Devices International (FTDI). URL: http://www.ftdichip.com/Support/Documents/DataSheets/ICs/DS_FT232BL_BQ.pdf (besucht am 14. 02. 2016).

Homepage lxml - XML and HTML with Python (2016). URL: http://lxml.de/ (besucht am 02. 03. 2016).

Homepage WinAVR (2016). URL: http://winavr.sourceforge.net/ (besucht am 02. 03. 2016).

HSMx-A10x-xxxxx SM LED Indicator Datasheet (2015). AVAGO Technologies. URL: http://docs-europe.electrocomponents.com/webdocs/0e9a/0900766b80e9af6b.pdf (besucht am 03. 03. 2016).

I2C-Bus Specification and User Manual (2014). Rev. 6. NXP Semiconductors. URL: http://www.nxp.com/documents/user_manual/UM10204.pdf (besucht am 03. 01. 2016).

Impedance curves of ceramic capacitors (2012). Wikimedia Commons. URL: https://en.wikipedia.org/wiki/File:MLCC-Imp-versus-Freqenz.engl.png (besucht am 10. 02. 2016).

Linux kernel coding style (2016). URL: https://www.kernel.org/doc/Documentation/CodingStyle (besucht am 03. 03. 2016).

LM2674 500mA Step-Down Voltage Converter Datasheet (2014). Texas Instruments. URL: http://www.ti.com/lit/ds/symlink/lm2674.pdf (besucht am 14. 02. 2016).

LXML 3.3.5 (2016). URL: https://pypi.python.org/pypi/lxml/3.3.5#downloads (besucht am 02. 03. 2016).

MCP1755 300mA High-Performance LDO Datasheet (2012). Microchip Technology. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/25160A.pdf (besucht am 14. 02. 2016).

MCP79410 Battery-Backed Real-Time-Clock Datasheet (2014). Microchip Technology. URL: http://ww1.microchip.com/downloads/en/DeviceDoc/20002266F.pdf (besucht am 07. 01. 2016).

MFRC522 Standard 3V MIFARE Reader Solution Datasheet (2014). NXP Semiconductors N.V. URL: http://www.nxp.com/documents/data_sheet/MFRC522.pdf (besucht am 24. 02. 2016).

MIFARE Classic 1K Product Datasheet (2011). NXP Semiconductors N.V. URL: http://www.nxp.com/documents/data_sheet/MF1S50YYX.pdf (besucht am 25. 02. 2016).

Neuftech Mifare RC522 RFID Module für Arduino (2016). Amazon. URL: http://www.amazon.de/Neuftech-Mifare-Module-KeyCard-Arduino/dp/B00QFDRPZY/ref=sr_1_29?ie=UTF8&qid=1439976462&sr=8-29&keywords=13,56+MHz+rfid#productDetails (besucht am 14. 02. 2016).

Non-Blocking Gui wxPyWiki (2016). URL: http://wiki.wxpython.org/Non-Blocking%20Gui (besucht am 22. 03. 2016).

P-Channel PowerTrench MOSFET Datasheet (2011). Fairchild Semiconductor. URL: https://www.fairchildsemi.com/datasheets/FD/FDT434P.pdf (besucht am 11. 02. 2016).

Plötz, Henryk (2008). "Mifare Classic - Eine Analyse der Implementierung". URL: http://sar.informatik.hu-berlin.de/research/publications/SAR-PR-2008-21/SAR-PR-2008-21_.pdf (besucht am 25. 02. 2016).

PyInstaller Manual (2016). URL: http://pythonhosted.org/PyInstaller/ (besucht am 04. 04. 2016).

PyInstaller Official Website (2016). URL: http://www.pyinstaller.org/ (besucht am 02. 03. 2016).

PySerial Documentation (2016). URL: https://pythonhosted.org/pyserial/ (besucht am 02. 03. 2016).

PySerial Installation (2016). URL: https://pythonhosted.org/pyserial/pyserial.html#installation (besucht am 02. 03. 2016).

Python Releases for Windows (2016). URL: https://www.python.org/downloads/windows/ (besucht am 02. 03. 2016).

RA08H1317M RF Power Module Datasheet (2006). Mitsubishi Electric Semiconductor. URL: http://www.box73.de/download/bauelemente/RA08H1317M.pdf (besucht am 26. 01. 2016).

RD06HHF1 RF Power Transistor Datasheet (2011). Mitsubishi Electric Semiconductor. URL: http://www.mitsubishielectric.com/semiconductors/content/product/hf/sirfpowermosfet/siliconrfdiscrete/siliconrfdiscrete_lv4/rd06hhf1.pdf (besucht am 02. 01. 2016).

RIK 10 Ferroxcube Core Datasheet (2013). Vielstedter Elektrotechnik. URL: http://cdn-reichelt.de/documents/datenblatt/B400/RIK10.pdf (besucht am 03. 01. 2016).

Rules for Championships in Amateur Radio Direction Finding Part B, IARU Region 1 (2015). International Amateur Radio Union. URL: http://www.ardf-r1.org/files/Rules_2015_B.pdf (besucht am 22. 03. 2016).

Silicon PIN diode Datasheet (2015). NXP Semiconductors. URL: http://www.nxp.com/documents/data_sheet/BAP64-06.pdf (besucht am 11. 02. 2016).

SK574 Extruded Heatsink Datasheet (2016). Fischer Elektronik. URL: http://docs-europe.electrocomponents.com/webdocs/0f3d/0900766b80f3d603.pdf (besucht am 12. 02. 2016).

STPS2L30 Schottky Diode Datasheet (2007). STMicroelectronics. URL: http://docs-europe.electrocomponents.com/webdocs/0979/0900766b8097959e.pdf (besucht am 14. 02. 2016).

T1-6T-X65 RF Transformer Datasheet (2016). Mini Circuits. URL: https://www.minicircuits.com/pdfs/T1-6T-X65.pdf (besucht am 12. 02. 2016).

T491 Series Tantalum SMD Capacitors Datasheet (2014). Kermet. URL: http://docs-europe.electrocomponents.com/webdocs/0718/0900766b80718482.pdf (besucht am 06. 01. 2016).

The Python Tutorial - Classes (2016). URL: https://docs.python.org/2/tutorial/classes.html (besucht am 02. 03. 2016).

The Python Tutorial - Modules (2016). URL: https://docs.python.org/2/tutorial/modules.html (besucht am 22. 03. 2016).

Virtual COM Port Drivers - FTDI Ltd. (2016). URL: http://www.ftdichip.com/Drivers/VCP.htm (besucht am 22. 03. 2016).

Wikipedia (2016a). *Arduino (Plattform) — Wikipedia, Die freie Enzyklopädie*. URL: https://de.wikipedia.org/w/index.php?title=Arduino_(Plattform)&oldid=151929441 (besucht am 22. 03. 2016).

– (2016b). *Morsezeichen — Wikipedia, Die freie Enzyklopädie*. URL: https://de.wikipedia.org/w/index.php?title=Morsezeichen&oldid=152486762 (besucht am 21. 03. 2016).

– (2016c). *.NET Framework — Wikipedia, Die freie Enzyklopädie*. URL: https://de.wikipedia.org/w/index.php?title=.NET_Framework&oldid=150746744 (besucht am 22. 03. 2016).

Wikipedia (2016d). *RFID — Wikipedia, Die freie Enzyklopädie*. URL: https://de.wikipedia.org/w/index.php?title=RFID&oldid=152524409 (besucht am 21. 03. 2016).

Würth Power Choke Datasheet (2008). Würth Elektronik. URL: http://docs-europe.electrocomponents.com/webdocs/0d87/0900766b80d87de7.pdf (besucht am 06. 01. 2016).

wxPython API Documentation - wx.App (2016). URL: http://wxpython.org/Phoenix/docs/html/App.html (besucht am 02. 03. 2016).

wxPython API Documentation - wx.Panel (2016). URL: http://wxpython.org/Phoenix/docs/html/Panel.html (besucht am 02. 03. 2016).

wxPython Windows Binaries (2016). URL: http://www.wxpython.org/download.php#msw (besucht am 02. 03. 2016).

Abbildungsverzeichnis

1.1. Gesamt-Blockschaltbild des Fuchssenders 19

2.1. Blockschaltbild Mikrocontroller-Platine 23
2.2. Beschaltung Mikrocontroller 24
2.3. Pinbelegung des TQFP-44 Gehäuses, aus dem Datenblatt 25
2.4. ISP-Programmierschnittstelle des μC 27
2.5. Beschaltung FT232 USB-Serial Converter 28
2.6. Beschaltung Real-Time Clock 29
2.7. Beschaltung EEPROM 30
2.8. 3.3 V Schaltregler 32
2.9. 5V Linearregler 33
2.10. Diverse Schaltungsteile 33
2.11. Flachbandkabelverbindungen 35

3.1. Blockschaltbild der DDS-HF-Platine 37
3.2. DDS-Prinzip Blockschaltbild, modifiziert nach Wikipedia „Direct Digital Synthesis“ 38
3.3. Beschaltung des DDS-Bausteins 41
3.4. 1.8 V Spannungsregler 43
3.5. DDS-Spektrum 80m 43
3.6. DDS-Spektrum 2m 43
3.7. Schaltplan Bandumschaltung 46
3.8. Blockschaltbild 80m-Leistungsverstärker 48
3.9. Schaltplan 80m-Vorverstärker 49
3.10. Schaltplan 80m Endstufe 53
3.11. Einfaches Resonanztransformationsglied 57
3.12. π-Filter zusammengesetzt 59
3.13. Zweistufiges Oberwellenfilter 61
3.14. Frequenzgang des Oberwellenfilters, Simulation 62
3.15. Spektrum des 80m-Ausgangssignals 63
3.16. Blockschaltbild 2m-Leistungsverstärker 67
3.17. Schaltplan 2m-Eingangsfilter 67
3.18. Amplitudengang des Eingangsfilters, Messung 69
3.19. Schaltplan 2m-Vorverstärker 70

3.20. Schaltplan 2m-Endstufe 73
3.21. Erstes π-Glied 76
3.22. Zweistufiges Oberwellenfilter 76
3.23. Frequenzgang des Filters, Simulation 78
3.24. Spektrum des 2m-Ausgangssignals 79

4.1. Blockschaltbild Richtkoppler-Platine 83
4.2. Schaltplan 80m-Richtkoppler 84
4.3. Prinzip PCB-Koppler 86
4.4. Empirisch ermittelte Abmessungen 87
4.5. Schaltplan 2m-Richtkoppler 88

5.1. Blockschaltbild RFID-Platine 89
5.2. Beschaltung RFID-Modul und RFID-LED 90
5.3. Schaltplan LED-Anzeige und Test-Sende-Taster 91

6.1. Störungen durch den Schaltregler im Ausgangssignal 96

21.1. Hauptfenster der PC-Software 190

23.1. Virtueller COM-Port im Gerätemanager 204
23.2. Hauptfenster der PC-Software 204
23.3. Dialog zum Setzen der Fuchsnummer 205
23.4. Dialog zur Kalibrierung der Quarzfrequenz 207
23.5. Teilnehmeransicht der PC-Software 209
23.6. Result-Ansicht der PC-Software 210

24.1. Aufbau zum Abgleich des 80m-Teils 212
24.2. Aufbau zum Abgleich des 2m-Teils 213
24.3. Lage des Übertragers und der Trimmer auf der Platine 214

A.1. Tastkopf ohne Massekabel 228

E.1. Mikrocontroller-Platine 242
E.2. DDS-HF-Platine 243
E.3. Richtkoppler-Platine 244
E.4. RFID-Platine 245
E.5. Mikrocontroller-Platine 246
E.6. DDS-HF-Platine 246
E.7. RFID-Platine 246
E.8. Richtkoppler-Platine 246

Tabellenverzeichnis

1.1. Platinenaufteilung 18

2.1. Pinbelegung Mikrocontroller 26
2.2. Abschätzung Stromverbrauch 31
2.3. Flachbandkabel zur DDS-HF- und Richtkoppler-Platine 36
2.4. Flachbandkabel zur RFID-Platine 36

3.1. Vergleich DDS-Bausteine 40
3.2. Stromverbrauch der Endstufen 80

9.1. Aufbau des Befehlsbytes des AD9859 113
9.2. Amplitudenverhalten abhängig von den OSK-Bits 114

D.1. Kostenübersicht 235

E.1. Bauteile Mikrocontroller-Platine 248
E.2. Bauteile DDS-HF-Platine 249
E.3. Bauteile Richtkoppler-Platine 250
E.4. Bauteile RFID-Platine 251
E.5. Diverse Bauteile 251

Teil IV.

Anhang

A. Verwendete Messgeräte und Entwicklungswerkzeuge

An dieser Stelle ein paar Worte über die verwendeten Werkzeuge, Messgeräte und Programme, die bei der Umsetzung dieser Arbeit benutzt wurden.

A.1. Messgeräte

Eines der wichtigsten Messmittel bei der Umsetzung der Diplomarbeit war das digitale Speicheroszilloskop Rigol DS1052E (50 MHz). Dazu kommt für die Entwicklung des 80m-Leistungsverstärkers der Function/Arbitrary Waveform Generator SDG1025 der Firma Siglent (bis 25 MHz), welcher auch eine Frequenzzähler-Funktion besitzt. Als Multimeter wurde das digitale MS8217 von MASTECH benutzt.

Besonders bei der Entwicklung des 2m-Leistungsverstärkers war das DS1052E mitunter nicht mehr ausreichend, sodass gelegentlich auf ein 4-Kanal LeCroy 9314L mit 300 MHz analoger Bandbreite und entsprechende Tastköpfe zurückgegriffen werden musste.

Hier sei auch erwähnt, dass der Vorgang des Messens im 2m-Band grundsätzlich schwieriger ist als in niedrigeren Frequenzbereichen. So muss ggf. am Tastkopf auf ein Massekabel (Krokodilklemme) verzichtet werden. Stattdessen wird eine kurze Drahtverbindung direkt vom Schirm der Spitze zur nächstgelegenen Massefläche auf der Platine benutzt (siehe A.1 auf der nächsten Seite). Nur so kann ein hochfrequenztauglicher Massebezug hergestellt werden.

Zur Kontrolle des Ausgangsspektrums beider Leistungsverstärker stand den Autoren ein Tektronix 7623A Oszilloskop mit Tektronix 7L13 Spektralanalysator-Erweiterung zur Verfügung (1 kHz bis 2 GHz). Dabei dient ein Sierra Model 661A-20 20 dB-Abschwächer als 50 Ω-Abschlusswiderstand für die Leistungsverstärker sowie als Messabschwächer für den Spektralanalysator. Die Ausgangsleistung und Anpassung wurde mit jeweils einem YAESU YS-60 SWR & Power Meter (für

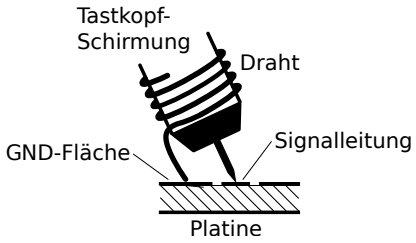


Abbildung A.1.: Tastkopf ohne Massekabel

1.6 MHz bis 60 MHz) und einem YAESU YS-500 (für 140 MHz bis 525 MHz) erfasst.

Für die endgültige Messung der Ausgangsspektren, wie sie in der schriftlichen Arbeit abgebildet sind, wurde der digitale Rhode & Schwarz FS300 benutzt, welcher im Labor der HTL Anichstraße zur Verfügung steht, da sich der Bildschirm des analogen Tektronix Spektrumanalysators nur umständlich aufzeichnen (fotografieren) lässt.

Weiters konnten bei der Dimensionierung des 2m-Leistungsverstärkers einige Messungen mit einem DG8SAQ VNWA3 Vektor-Netzwerkanalysator von SDR-Kits durchgeführt werden. Ohne diese Messmöglichkeit wäre die Realisierung bzw. Dimensionierung der Resonanzkreise (insbesondere der Spulen) wesentlich umständlicher geworden.

A.2. Computersoftware

Die schriftliche Arbeit ist in LaTeX verfasst. An dieser Stelle sei unserem Mathematikprofessor, großem Vorbild und LaTeX-Mentor Prof. Mag. Salvador Robert gedankt, welcher uns für dieses wunderbare Werkzeug begeistern konnte.

Schaltpläne und Platinenlayouts sind mit der kostenlosen Version von Cadsoft EAGLE 6.5.0 erstellt.

Grafiken wurden großteils mit dem freien Vektorzeichenprogramm Inkscape erstellt. Die Schaltpläne wurden aus Eagle in eine Postscript-Datei exportiert, um sie dann mit Inkscape weiterzubearbeiten und schließlich als PDF in LaTeX einzubinden. Gnuplot wurde benutzt, um Datensätze (z.B. Messwerte des Spektralanalysators) grafisch in einem Diagramm darzustellen.

Diverse Umformungen und Berechnungen wurden mit dem freien CAS-Mathematikpaket wxMaxima durchgeführt bzw. kontrolliert. Auch für die schnelle Umrechnung von Einheiten, s-Parametern und Parallel/Serien-Ersatzschaltbildern war dieses Tool eine große Hilfe.

Für Simulationen der Filterschaltungen und auch anderen Schaltungsteilen wurde LTSpice benutzt.

A.3. Sonstiges

Zur schnellen Erstellung von Prototypen und kleineren Teil-Testplatinen war die private Ausstattung der Autoren zum Ätzen von Platinen sehr hilfreich. Ohne diese Möglichkeit hätte die Entwicklung vermutlich um einiges mehr Zeit beansprucht.

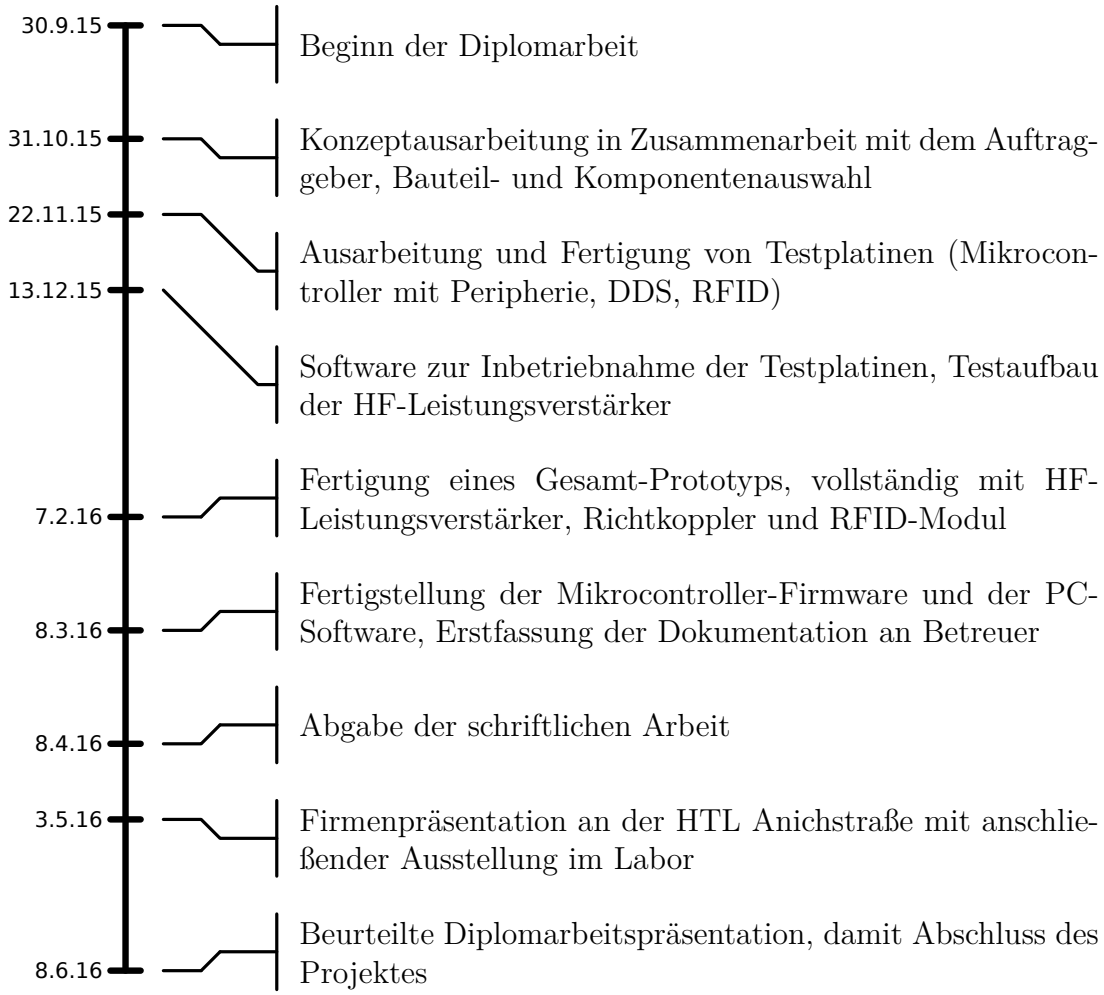
B. Lastenheft

Im Rahmen dieser Arbeit gilt es, einen Fuchsjagdsender zu entwickeln und einen Prototyp davon zu fertigen, welcher folgende, vom Projektpartner gestellte Anforderungen erfüllt:

- Sendebetrieb im 80m- und im 2m-Amateurfunkband (nicht gleichzeitig)
- Konform mit den offiziellen IARU-ARDF-Rules (Morsekennung, Modulation, Frequenzbereich, Sendeleistung, Betriebsdauer, etc.)
- Erfüllung der rechtlichen Bestimmungen in Österreich (Nebenaussendungen etc.)
- Teilnehmerregistrierung am Fuchs über RFID-Tags
- Manipulationsschutz der RFID-Tags
- Aufzeichnen der registrierten Teilnehmer mit Zeitstempel am Fuchs
- Weiterlaufen der Systemzeit auch bei entferntem Akku ($\rightarrow$ Pufferbatterie)
- Richtkoppler zur Messung der Antennenanpassung mit Anzeige
- Konfiguration des Senders über die USB-Schnittstelle vom PC aus
- Betrieb mit einem 12 V-Akku
- Umfassende Dokumentation in Form der schriftlichen Arbeit

Nicht Teil der Arbeit ist eine Möglichkeit zum Laden des Akkus, der Entwurf und die Fertigung eines passenden Gehäuses, sowie die Fertigung von 5 Sendern (ein Satz) für den Projektpartner.

C. Zeitplanung



D. Kostenübersicht

Im Folgenden wird eine grobe Übersicht über die teuersten Bauteile eines Fuchses gegeben. Die Preise sind gerundet Stand März 2016 von jenen Anbietern, welche zu den einzelnen Komponenten aufgeführt sind.

Bezeichnung	**Anbieter**	**Preis**
ATmega644 Mikrocontroller	Reichelt	ca. € 8
FT232BL USB Serial Converter	RS-Components	ca. € 5
3.3 V-Schaltregler	RS-Components	ca. € 4
AD9859 DDS Baustein	Farnell	ca. € 21
HF-Übertrager T1-6T-X65	Funkamateur	ca. € 9
RD06HHF1 80m-Endstufentransistor	Funkamateuer	ca. € 5
RA08H1317M 2m-HF-Power-Modul	Funkamateur	ca. € 27
RC522 RFID Modul	Amazon	ca. € 10
Fertigung der vier Platinen (*1)	PCP-Pool	ca. € 80
Gesamt		ca. € 169

Tabelle D.1.: Kostenübersicht

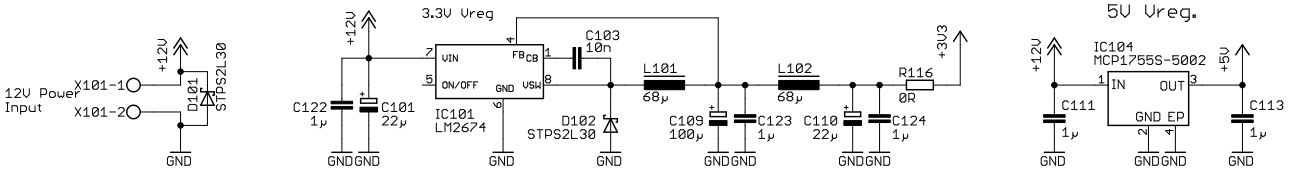
Beim Bau des Prototypen konnte inklusive aller Kleinteile ein Gesamtpreis von ca. € 200 bis € 220 pro Fuchssender erzielt werden. Dies hängt jedoch stark von den einzelnen Bezugsquellen ab. Noch nicht berücksichtigt sind die Preise für entsprechende Antennen, Akkus zur Stromversorgung sowie für den Bau geeigneter Gehäuse.

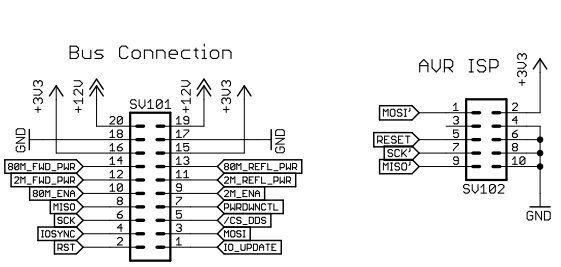
(*1) Preis bei Abnahme von jeweils 10 Stk., 2-lagig, 100 mm x 80 mm, Material FR4 35 µm CU, Lötstoplack, Bestückungsdruck Top, mind. 6 mil Leiterbahnstärke

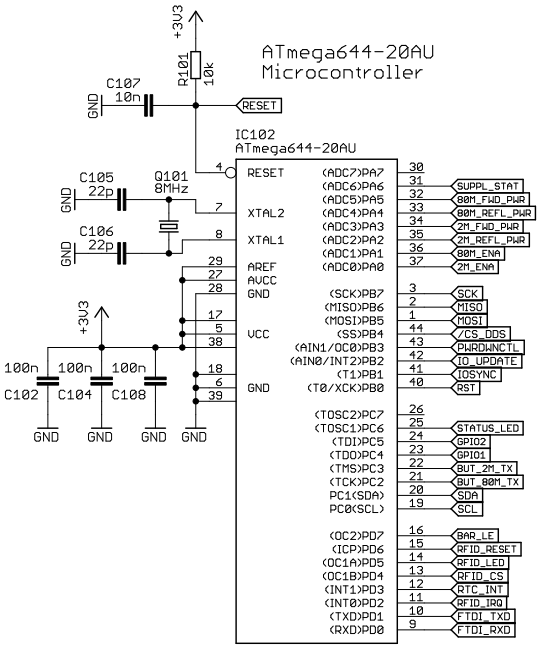
Anmerkung: Im Zuge der Entwicklung wurden Komponenten im Wert von ca. € 70 durch falsch verbundene Stecker, abrutschen mit dem Tastkopf etc. vernichtet. Das soll auch einmal erwähnt sein. Keine größere Entwicklung ohne Rauch...

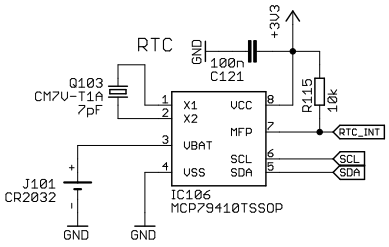
E. Fertigungsdokumentation

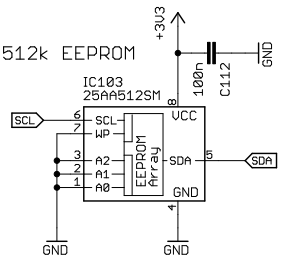
E.1. Gesamtschaltpläne

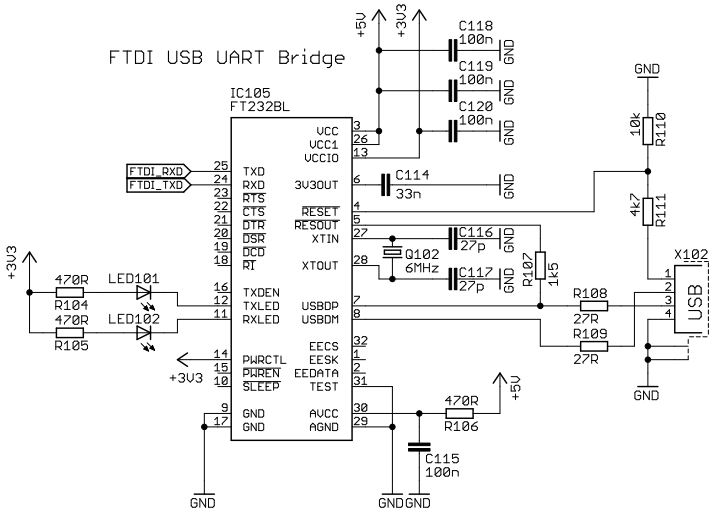


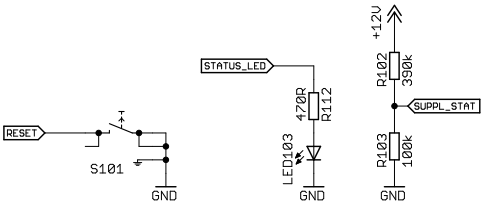


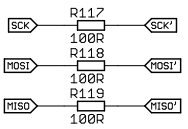


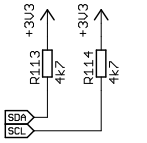




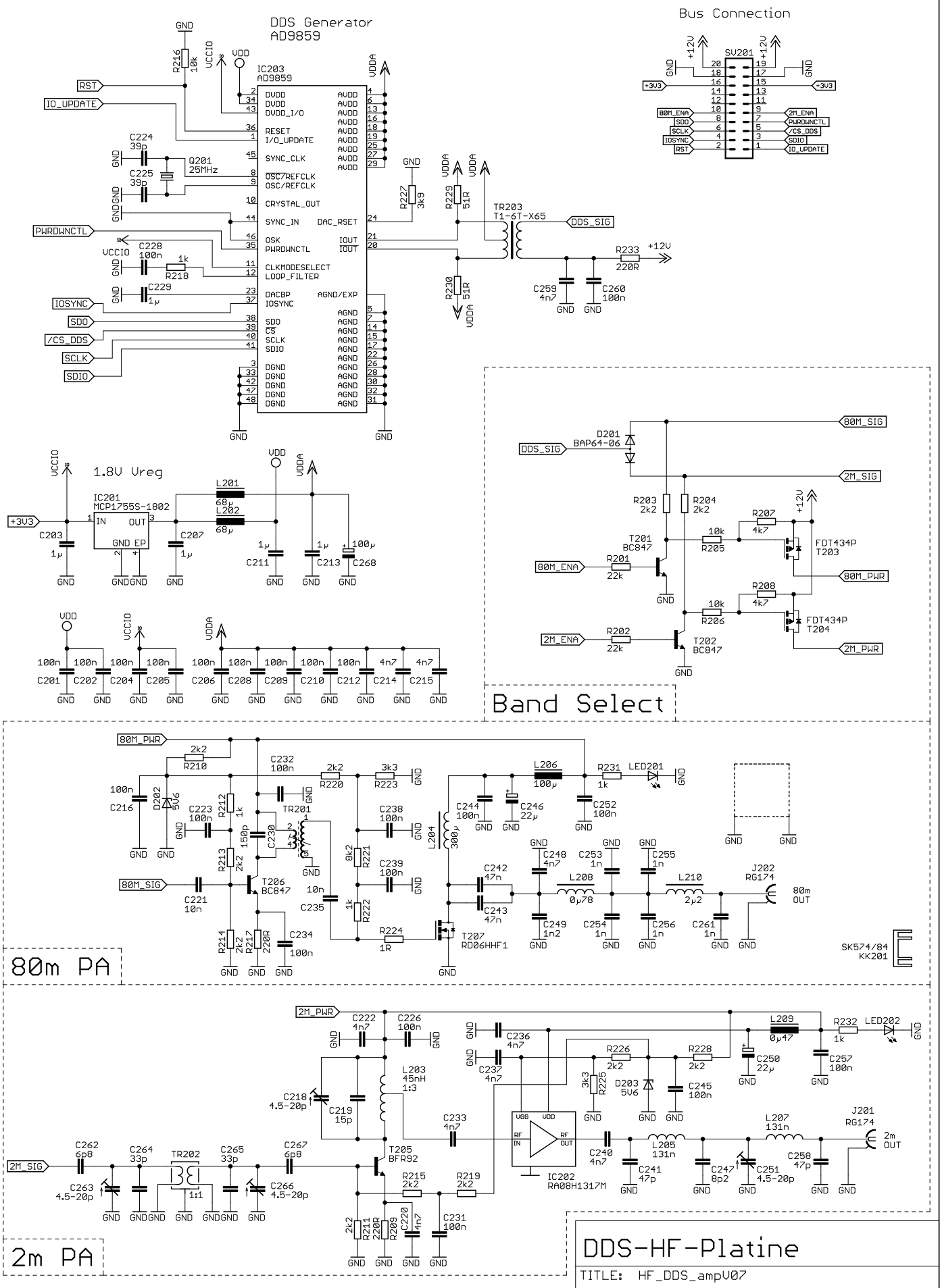




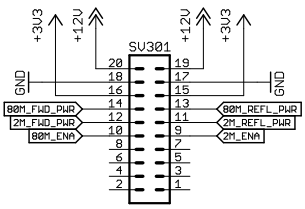




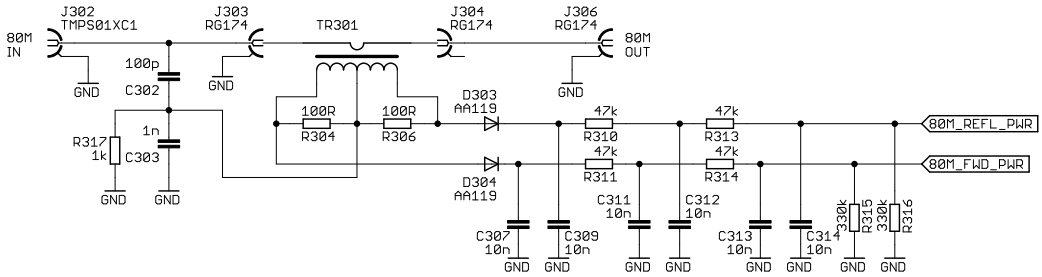
Microcontroller-Platine	
TITLE: uC_board_V05	
Document Number:	REV:
Date: nicht gespeichert!	Sheet: 1/1

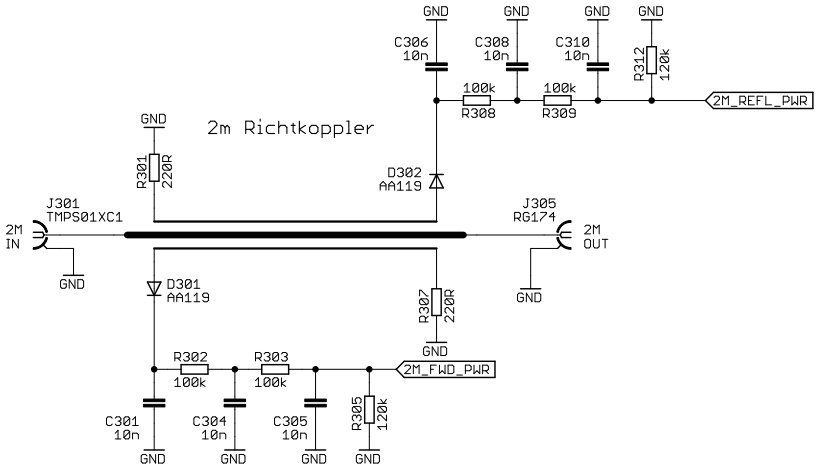

DDS Generator
AD9859
Bus Connection
1.8V Vreg
Band Select
80m PA
2m PA
DDS-HF-Platine
TITLE: HF_DDS_ampV07
Document Number:
REV:
Date: 19.03.16 13:08
Sheet: 1/1

Bus Connection

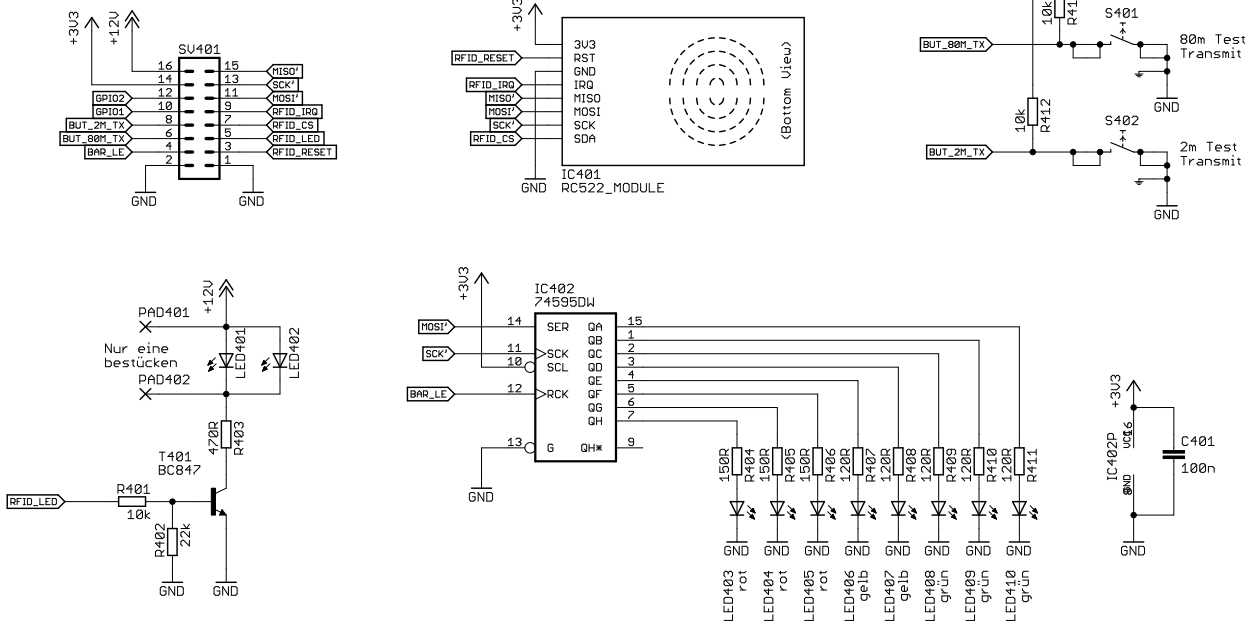


80m Richtkoppler





Richtkoppler-Platine		
TITLE: HF_meter_boardV06		
Document Number:		REV:
Date: 19.03.16 13:26	Sheet: 1/1	



RFID-Platine		
TITLE: RFID_user_interface_board_V4		
Document Number:		REV:
Date: 11.03.16 21:39	Sheet: 1/1	

E.2. Platinenlayouts

Die Platinenlayouts sind im Maßstab 1:1.5 abgedruckt.

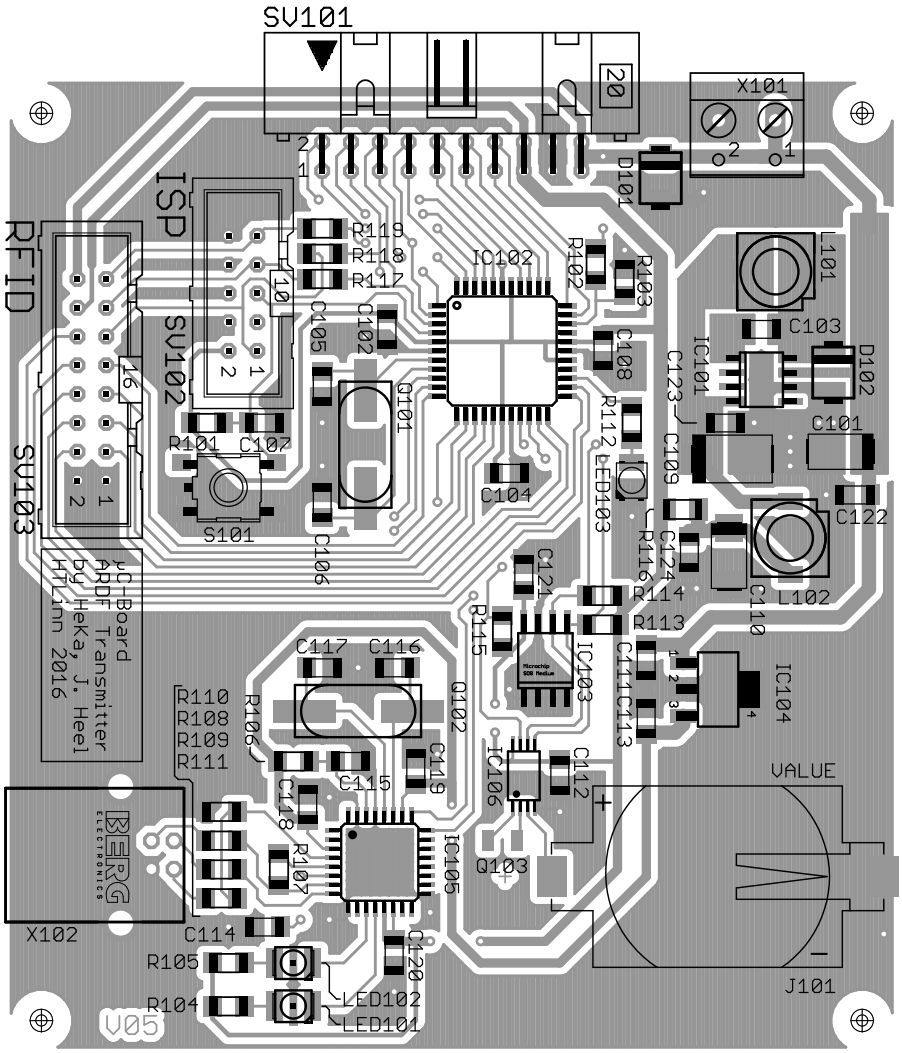


Abbildung E.1.: Mikrocontroller-Platine

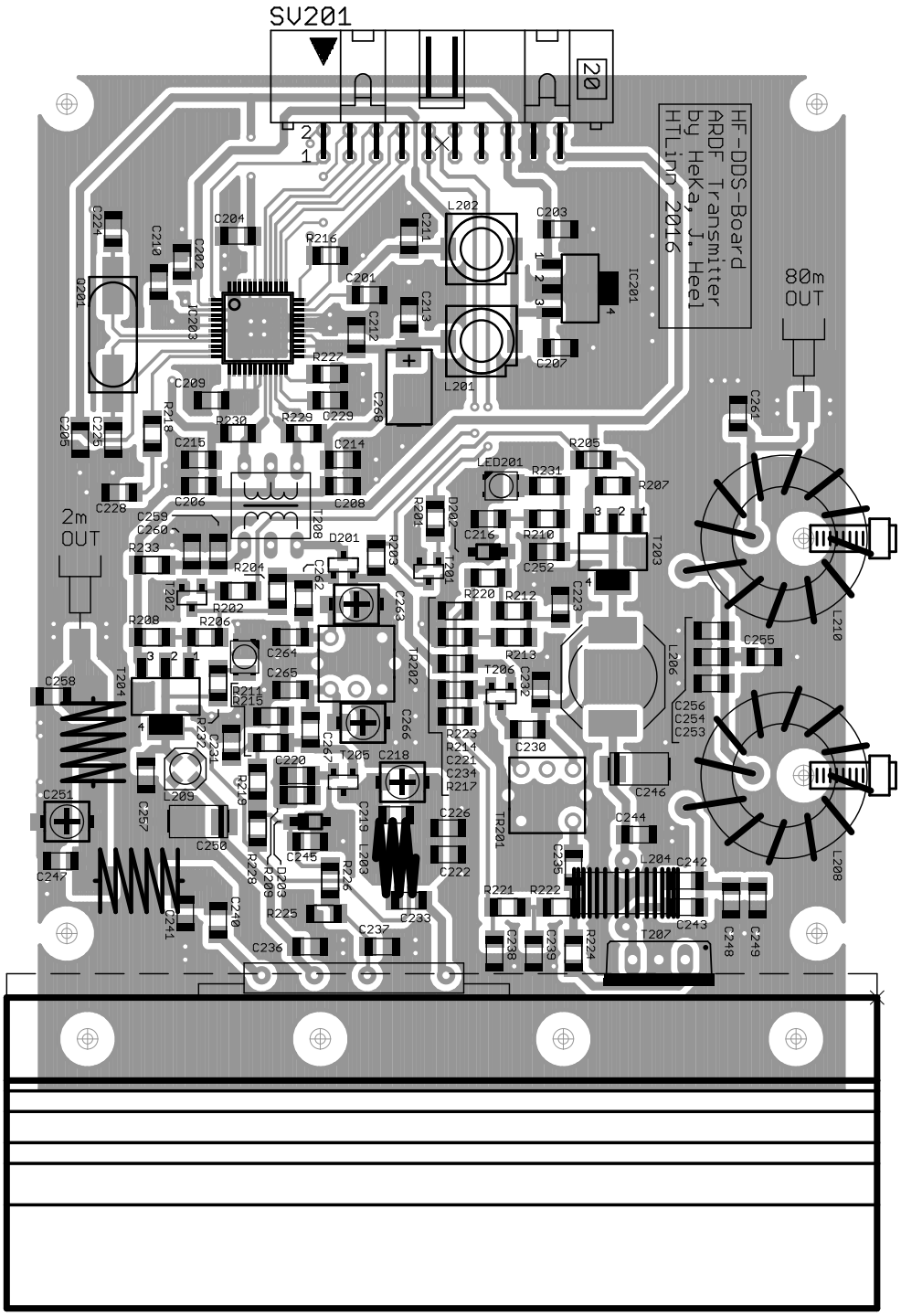


Abbildung E.2.: DDS-HF-Platine

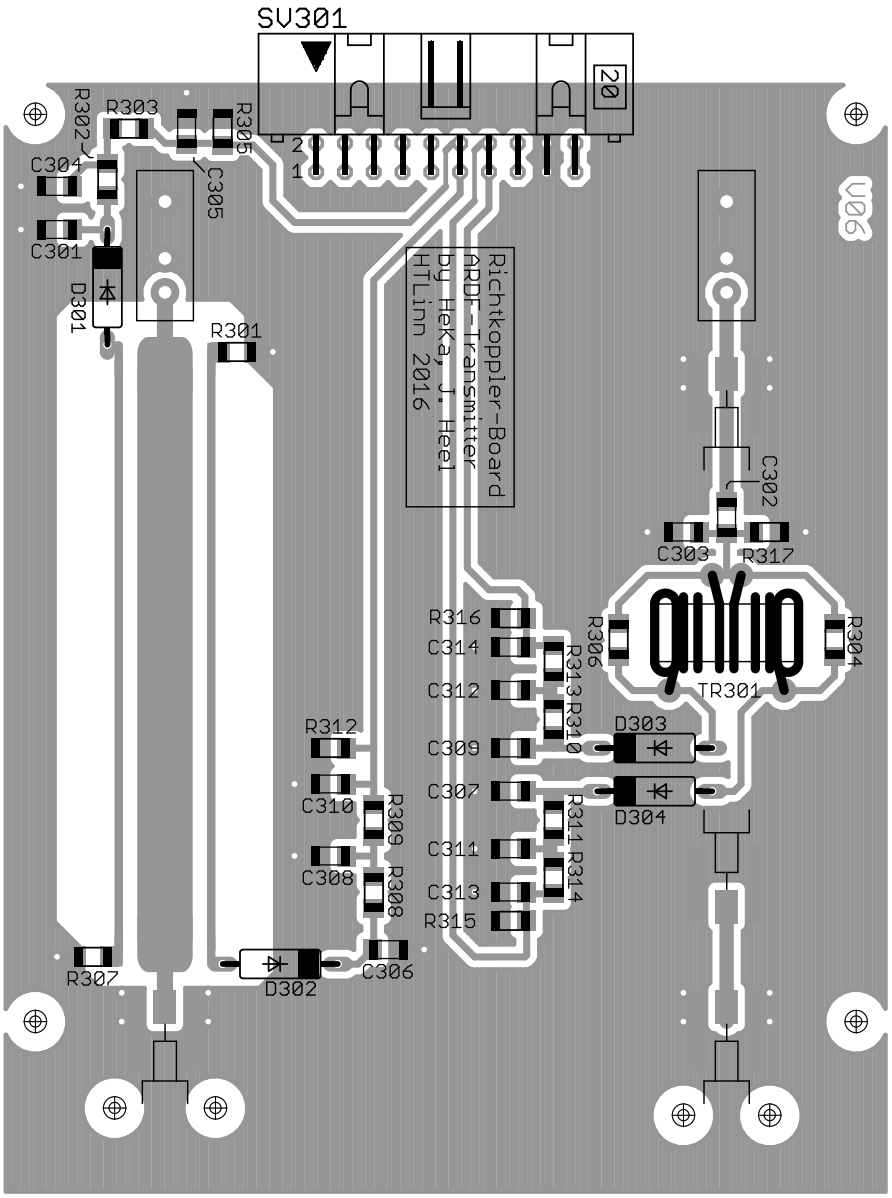


Abbildung E.3.: Richtkoppler-Platine

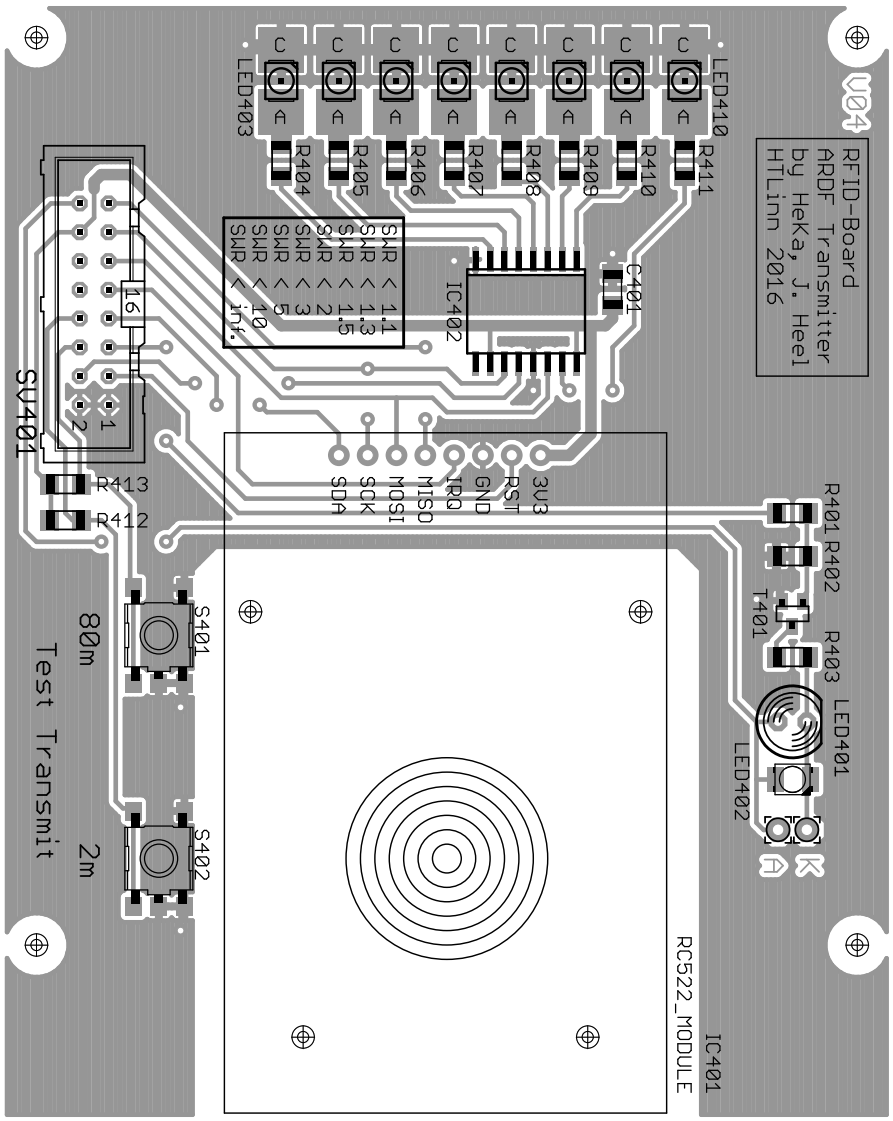


Abbildung E.4.: RFID-Platine

E.3. Prototyp

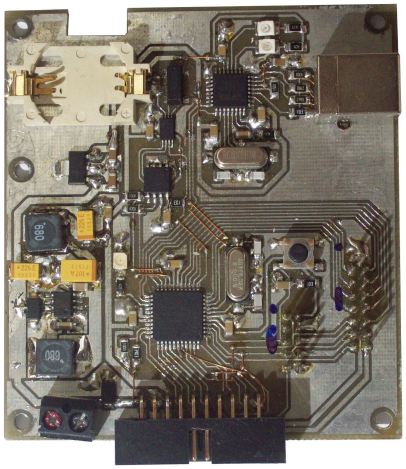


Abbildung E.5.: Mikrocontroller-Platine

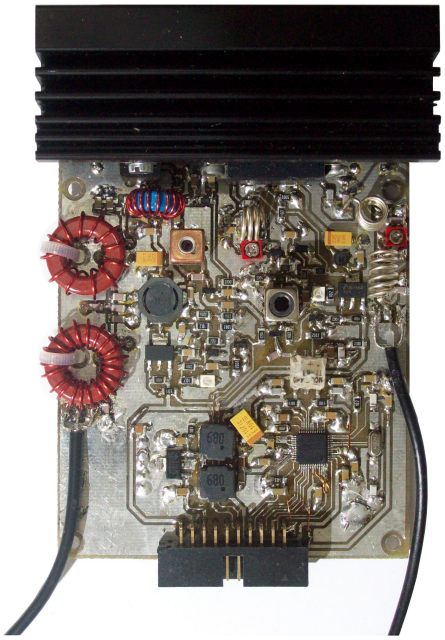


Abbildung E.6.: DDS-HF-Platine

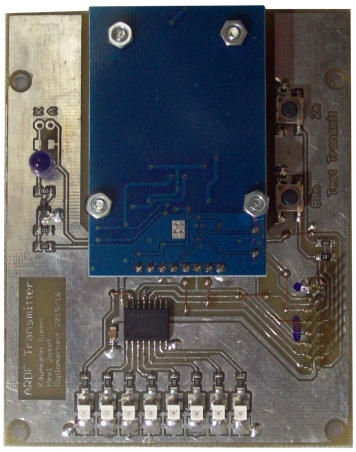


Abbildung E.7.: RFID-Platine

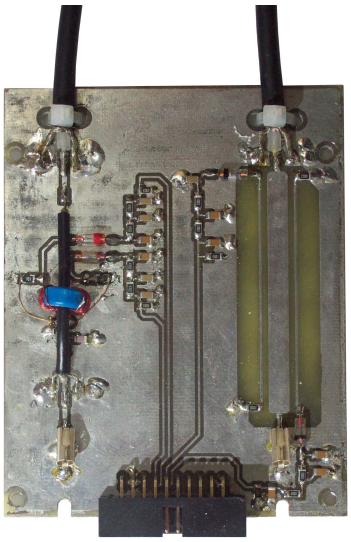


Abbildung E.8.: Richtkoppler-Platine

E.4. Stückliste

Folgende Stückliste enthält alle benötigten Bauteile und Komponenten zum Aufbau eines Fuchssenders. Für den Betrieb sind weiters ein Akku, passende Antennen und ein Gehäuse notwendig, welche hier nicht aufgelistet sind.

E.4.1. Mikrocontroller-Platine

Qty.	Part	Value	Package	Description
1	IC101	LM2674	SO-8	500mA Step-Down Voltage Regulator
1	IC104	MCP1755S-5002	SOT223	Low Dropout 5 V Linear Voltage Regulator
1	IC102	ATmega644-20AU	44-TQFP	8-bit Atmel Microcontroller with 64K Flash
1	IC105	FT232BL	LQFP	USB to Serial UART Converter
1	IC103	24AA512	SO-8M	Serial I^2C 512 kBit EEPROM
1	IC106	MCP79410	TSSOP8	Microchip I^2C Real-time Clock
2	D101, D102	STPS2L30	SMA	2 A 30 V Power Schottky Diode
1	LED101	red	PLCC-2	SMD LED (HSMS-A100-L00J1)
1	LED102	green	PLCC-2	SMD LED (HSME-A100-L01J1)
1	LED103	yellow	PLCC-2	SMD LED (HSMY-A100-L00J1)
1	Q101	8 MHz	XT49M	Low Profile SMD Crystal
1	Q102	6 MHz	XT49M	Low Profile SMD Crystal
1	Q103	32.768 kHz	(SMD)	CM7V-T1A-7pF 7 pF ±20 ppm Tuning Fork Crystal
1	R116	0 Ω	1206	SMD Jumper in Resistor-Package
2	R108, R109	27 Ω	1206	SMD Thick Film Resistor 0.25 W
3	R117, R118, R119	100 Ω	1206	SMD Thick Film Resistor 0.25 W
4	R104, R105, R106, R112	470 Ω	1206	SMD Thick Film Resistor 0.25 W
1	R107	1.5 kΩ	1206	SMD Thick Film Resistor 0.25 W
3	R111, R113, R114	4.7 kΩ	1206	SMD Thick Film Resistor 0.25 W
3	R101, R110, R115	10 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R103	100 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R102	390 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	C105, C106	22 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C116, C117	27 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C103, C107	10 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	C114	33 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
9	C102, C104, C108, C112, C115, C118, C119, C120, C121	100 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
5	C111, C113, C122, C123, C124	1 µF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C101, C110	22 µF	6032-28 (C)	AVX TPS SMD Tantal Elko Low-ESR 25 V
1	C109	100 µF	7343-31 (D)	AVX TPS SMD Tantal Elko Low-ESR 10 V
2	L101, L102	68 µH	(SMD)	TDK SMD Inductor 770 mA (SLF7045)
1	J101	CR2032	(SMD)	CR2032 Battery Holder (KZH 20SMD-2)

1	S101	(SMD)	Surface-Mount 6.2 mm x 6.5 mm TACT Switch
1	SV101	(THT)	10x2 IDC Connector
1	SV102	(THT)	Harting 5x2 IDC Connector
1	SV103	(THT)	Harting 8x2 IDC Connector
1	X101	(THT)	5.08 mm 2-pin PCB screw terminal
1	X102	(THT)	Shielded USB-B Connector

Tabelle E.1.: Bauteile Mikrocontroller-Platine

E.4.2. DDS-HF-Platine

Qty.	Part	Value	Package	Description
1	IC201	MCP1755S-1802	SOT223	Low Dropout 1.8 V Linear Voltage Regulator
1	IC203	AD9859	48-TQFP/EP	CMOS Direct Digital Synthesizer 400 MSPS, 10-Bit
1	IC202	RA08H1317M	(THT)	Mitsubishi RF MOSFET Power Module 8 W
1	T207	RD06HHF1	TO-220	MOSFET Power Transistor 30 MHz 6 W
1	T205	BFR92	SOT23	Silicon NPN Planar RF Transistor 5 GHz
3	T201, T202, TR06	BC847B	SOT23	NPN general-purpose Transistor
2	T203, T204	FDT434P	SOT223	P-Ch MOSFET Transistor 20 V 6 A
1	D201	SOT23	BAP64-06	RF Silicon Dual PIN-Diode
2	D202, D203	5.6 V	SOD-123	SMD Zener Diode 0.5 W
2	LED201, LED202	yellow	PLCC-2	SMD LED (HSMY-A100-L00J1)
1	Q201	25 MHz	XT49M	Low Profile SMD Crystal
1	R224	1 Ω	1206	SMD Thick Film Resistor 0.25 W
2	R229, R230	51 Ω	1206	SMD Thick Film Resistor 0.25 W
3	R209, R217, R223	220 Ω	1206	SMD Thick Film Resistor 0.25 W
5	R212, R218, R222, R231, R232	1 kΩ	1206	SMD Thick Film Resistor 0.25 W
11	R203, R204, R210, R211, R213, R214, R215, R219, R220, R226, R228	2.2 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	R223, R225	3.3 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R227	3.9 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	R207, R208	4.7 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R221	8.2 kΩ	1206	SMD Thick Film Resistor 0.25 W
3	R205, R206, R216	10 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	R201, R202	22 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	C262, C267	6.8 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	C247	8.2 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	C219	15 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
4	C218, C251, C263, C266	4.5 . . . 20 pF	(SMD)	Ceramic Trimm-Capacitor (TZB4R200AB10R00)
2	C264, C265	33 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C224, C225	39 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C241, C258	47 pF	1206	SMD Multilayer Ceramic Capacitor 50 V

1	C230	150 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
5	C253, C254, C255, C256, C261	1 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	C249	1.2 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
9	C214, C215, C220, C222, C233, C236, C237, C240, C248, C259	4.7 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C221, C235	10 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C242, C243	47 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
23	C201, C202, C204, C205, C206, C208, C209, C210, C212, C216, C223, C226, C228, C231, C232, C234, C238, C239, C244, C245, C252, C257, C260	100 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
5	C203, C207, C211, C213, C229	1 µF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	C246, C250	22 µF	6032-28 (C)	AVX TPS SMD Tantal Elko Low-ESR 25 V
1	C269	100 µF	7343-31 (D)	AVX TPS SMD Tantal Elko Low-ESR 10 V
1	L203	45 nH		Air Coil d=7 mm (6 mm shaft), l=4 mm, 3 turns, d=1 mm silver-plated wire
2	L205, L207	131 nH		Air Coil d=6 mm (5 mm shaft), l=10 mm, 6 turns, d=1 mm silver-plated wire
1	L209	0.47 µH	(SMD)	SMD Ferrite Choke 1.45 A $\mathrm{SRF}_{\mathrm{min}} = 270$ MHz (Würth 7440300047)
1	L208	0.78 µH	(THT)	Amidion T68-2 core (57 µH/100 turns), 12 tuns
1	L210	2.2 µH	(THT)	Amidion T68-2 core (57 µH/100 turns), 20 tuns
2	L201, L202	68 µH	(SMD)	TDK SMD Inductor 770 mA (SLF7045)
1	L206	100 µH	(SMD)	SMD Ferrite Choke 1.2 A $\mathrm{SRF}_{\mathrm{min}} = 6$ MHz (Würth 744066101)
1	L204	300 µH	(THT)	RIK10 Core (Material N30, ≈ 1.4 µH/turn), 15 turns
1	TR201	1:2	(THT)	Shielded Filter Kit $A_{\mathrm{L}} = 12$ nH, 26:52 turns (7F1S)
1	TR202	1:1	(THT)	Shielded Filter Kit (with magnetic core removed), 1:1 turn
1	TR203	T1-6T-X65	DIL-6	Mini-Circuits RF Wideband Transformer (0.015 . . . 300 MHz, 50 Ω)
1	SV201		(THT)	Harting 10x2 IDC Connector
1	KK201			SK574 Aluminium Profile, l=84 mm

Tabelle E.2.: Bauteile DDS-HF-Platine

E.4.3. Richtkoppler-Platine

Qty.	Part	Value	Package	Description
4	D301, D302, D303, D304	AA119	(THT)	Germanium Diode
2	R304, R206	100 Ω	1206	SMD Thick Film Resistor 0.25 W
2	R301, R307	220 Ω	1206	SMD Thick Film Resistor 0.25 W
1	R317	1 kΩ	1206	SMD Thick Film Resistor 0.25 W
4	R310, R311, R313, R314	47 kΩ	1206	SMD Thick Film Resistor 0.25 W
4	R302, R303, R308, R309	100 kΩ	1206	SMD Thick Film Resistor 0.25 W
2	R305, R312	120 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R315, R316	330 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	C302	100 pF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	C303	1 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
12	C301, C304, C305, C306, C307, C308, C309, C310, C311, C312, C313, C314	10 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
1	TR301		(THT)	RIK10 Core (Material N30, ≈ 1.4 µH/turn), 2 x 11 turns
1	S301		(THT)	Harting 10x2 IDC Connector
2	J301, J302		(THT)	TMPS01XC1 PCB Coax Socket

Tabelle E.3.: Bauteile Richtkoppler-Platine

E.4.4. RFID-Platine

Qty.	Part	Value	Package	Description
1	IC401		(THT)	Arduino compatible 13.56 MHz RFID Module with RC522
1	IC402	74HC595W	SO16DW	CMOS 8-bit Serial → Parallel Shift Register
1	TR401	BC847B	SOT23	NPN general-purpose Transistor
1	LED401		5 mm Std.	Super-bright 20 mA LED
0	LED402		PLCC-2	Surface-mount super-bright 20 mA LED (instead of LED401)
3	LED403, LED404, LED405	red	PLCC-2	SMD LED (HSMS-A100-L00J1)
2	LED406, LED407	yellow	PLCC-2	SMD LED (HSMY-A100-L00J1)
3	LED408, LED409, LED410	green	PLCC-2	SMD LED (HSME-A100-L01J1)
5	R407, R408, R409, R410, R411	120 Ω	1206	SMD Thick Film Resistor 0.25 W
3	R404, R405, R406	150 Ω	1206	SMD Thick Film Resistor 0.25 W
1	R403	470 Ω	1206	SMD Thick Film Resistor 0.25 W

3	R401, R412, R413	10 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	R402	22 kΩ	1206	SMD Thick Film Resistor 0.25 W
1	C401	100 nF	1206	SMD Multilayer Ceramic Capacitor 50 V
2	S401, S402		(SMD)	Surface-Mount 6.2 mm x 6.5 mm TACT Switch
1	SV401		(THT)	Harting 8x2 IDC Connector

Tabelle E.4.: Bauteile RFID-Platine

E.4.5. Diverses

Qty.	Part	Value	Package	Description
1		20-pin		RM 1.27 mm Ribbon-Cable (ca. 5 cm)
1		16-pin		RM 1.27 mm Ribbon-Cable (ca. 3 cm)
3		10x2-pin		Harting IDC bump polarized socket
2		8x2-pin		Harting IDC bump polarized socket
2	J201, J202			RG174 Coaxial Cable (to Richtkoppler-Board, ca. 7 cm)
1	J303, (J304)			RG174 Coaxial Cable (through TR301)
2	J305, J306			RG174 Coaxial Cable (RF out)
2				RG174 Coaxial Crimp Connector (TMPK01XA1)
2				N-Connector for RF Output (mounted in the Case)
4				M3 Threaded Standoff 12 mm
4				M3 Threaded Standoff 16 mm
4				M3 Threaded Standoff 25 mm
10				M3 Flat Head Screw 6 mm
4				M3 Nut

Tabelle E.5.: Diverse Bauteile

F. Arbeitsnachweis

F.1. Projektmitglied Josef Heel

Datum	Stunden	Tätigkeit
23.9.2015	1.5 h	Ausarbeiten Diplomarbeitsantrag
30.9.2015	1 h	Konzeptüberlegungen, Lesen der Vorgängerarbeit
1.10.2015	1 h	Überlegungen (HF-Endstufen, Akku)
2.10.2015	3.75 h	Recherche DDS-Baustein, Überlegungen Spannungsversorgung
3.10.2015	2.75 h	Überlegungen zum Aufbau der Testplatinen
6.10.2015	1.25 h	Komponentenauswahl RTC, EEPROM etc.
14.10.2015	1.5 h	Beginn Schaltplanausarbeitung für Testplatinen
18.10.2015	6 h	Stückliste für Bauteilbestellungen, Testplatinendesign
24.10.2015	2 h	Schaltplan der Testplatinen erweitern
26.10.2015	5.75 h	Layoutarbeiten an den Testplatinen (DDS)
2.11.2015	4 h	Layoutarbeiten Testplatinen (DDS und μC)
5.11.2015	3 h	Einarbeiten von Änderungen in die Testplatinen, Fertigung
6.11.2015	2.5 h	Bestücken DDS-Testplatine
14.11.2015	3.75 h	Bestücken DDS- und μC-Testplatine
28.12.2015	2.75 h	Erste Inbetriebnahme der Testplatinen
30.12.2015	5 h	Überlegungen und Versuche zum 80m-LV
31.12.2015	2 h	Testaufbau 80m-LV, Überlegungen zum Oberwellenfilter
1.1.2016	4 h	Schaltpläne 80m-LV, Dimensionierung 2m-LV
2.1.2016	6.5 h	Schaltplanausarbeitung 2m-LV, Überlegungen Richtkoppler
3.1.2016	3.5 h	Beginn Layoutarbeiten am Prototyp
4.1.2016	5.75 h	Layoutarbeiten am Prototyp, Dokumentation
5.1.2016	2 h	Layoutarbeiten (Richtkoppler-Platine)
6.1.2016	8 h	Fertigung der Gesamt-Prototyp-Platinen, Beginn Bestückung
16.1.2016	1 h	Bestückung Prototyp (DDS-HF-Platine)
26.1.2016	2.75 h	Bestückung Prototyp (diverse), Dokumentation
30.1.2016	5 h	Bestückung Prototyp (diverse)
31.1.2016	5.25 h	Erste schrittweise Inbetriebnahme, Testen des 80m-LV
7.2.2016	4.5 h	Testen des 2m-Leistungsverstärkers, sowie der Richtkoppler
8.2.2016	3.25 h	Überlegungen 2m-LV (Nebenaussendungen), Dokumentation
9.2.2016	7.75 h	Einarbeiten eines 2m-Eingangsfilters, Dokumentation

10.2.2016	8 h	Dokumentation 2m-Leistungsverstärker
12.2.2016	9.5 h	Messungen, DDS-Problemsuche, Dokumentation
14.2.2016	7.75 h	Dokumentation Mikrocontroller-Platine
15.2.2016	1 h	Austausch Mikrocontroller (da zu wenig Speicher)
16.2.2016	6 h	Dokumentation 80m-LV und Richtkoppler
24.2.2016	4 h	Einbau eines Eingangsfilters im 2m-LV (Prototyp)
25.2.2016	6.25 h	Zusammenbau Prototyp, Messungen, Dokumentation
28.2.2016	1.5 h	Dokumentation 2m-LV mit Eingangsfilter
29.2.2016	1.25 h	Diverse Arbeiten an der Dokumentation (Zusammenfügen etc.)
1.3.2016	5.25 h	Einleitung, Weiterführende Überlegungen, etc.
2.3.2016	1.5 h	Dokumentation Kühlkörperberechnung, Übersicht etc.
4.3.2016	2.5 h	Dokumentation DDS-Baustein
6.3.2016	6.25 h	Dokumentation DDS, Zusammenfügen bisheriger Dokumentation
8.3.2016	2 h	Zusammenstellen Fertigungsdokum. (Platinenlayouts etc.)
9.3.2016	1 h	Diverse Änderungen an der Dokumentation
10.3.2016	1.5 h	Inbetriebnahmeanweisungen
11.3.2016	3.25 h	Dokumentation Fehler und Verbesserungsmöglichkeiten
17.3.2016	3 h	Dokumentation allgemeiner Punkte (Anhang etc.)
18.3.2016	2.5 h	Zusammenstellen Stückliste
19.3.2016	10.5 h	Dokumentation allgemeiner Punkte
gesamt	**188.25h**	

F.2. Projektmitglied Simon Kaufmann

Datum	Stunden	Tätigkeit
20.9.2015	2 h	Planungen Software, Bauteilauswahl RFID
16.10.2015	1 h	Recherche RFID-Softwarebibliothek, Ansteuerung Bausteine
17.10.2015	4 h	Beginn Softwareprogrammierung Mikrocontroller
23.10.2015	2.5 h	Softwareprogrammierung DDS-Modul
24.10.2015	4.25 h	Softwareprogrammierung DDS-Modul
2.11.2015	4 h	Ansteuerung EEPROM und RTC, Schreiben von Bibliotheken
5.11.2015	2.5 h	RFID-Programmierung: Portierung der Arduino-Bibliothek
14.11.2015	2 h	RFID-Programmierung
28.11.2015	3.5 h	Programmierung der Amplitudenmodulation mittels ASF-Register
27.12.2015	4 h	Programmierung UART-Buffer und Kommandoparser
29.12.2015	3 h	Test der Software mittels Testplatinen
30.12.2015	2.5 h	Fehlerbehebung und Test der Software
31.12.2015	5 h	Programmierung der Morse-Unterstützung
1.1.2016	5 h	Morsen und Modulation kombiniert, Test gemeinsam mit RFID
2.1.2015	2 h	Fehlerbehebung Stacküberlauf

3.1.2016	1 h	Programmierung Teilnehmerregistrierung
4.1.2016	4 h	Programmierung Teilnehmerregistrierung
5.1.2016	3 h	Programmierung Teilnehmerregistrierung, diverses Refactoring
7.1.2016	1.5 h	Schreiben über DDS und EEPROM
14.1.2016	3 h	Schreiben über RTC und DDS
17.1.2016	3.5 h	PC-Software begonnen
23.1.2016	13.5 h	PC-Software verbunden mit Testplatine, diverse Erweiterungen Mikrocontroller-Software
24.1.2016	7.5 h	Erweiterung Mikrocontrollersoftware
31.1.2016	5 h	Bessere UART-Kommunikation für PC-Software
1.2.2016	1 h	Erweiterung PC-Software
6.2.2016	9 h	Dateilese- und -schreibfunktionen für PC-Software
8.2.2016	6 h	Fehlerbehebung PC-Software
10.2.2016	5 h	Schreiben über UART-Kommunikation, Verbesserung RTC-Dokumentation
11.2.2016	6 h	Schreiben über Startup-Modul
12.2.2016	2 h	Schreiben über Morsemodul
14.2.2016	2.75 h	Schreiben über Commands-Modul
16.2.2016	1 h	Programmierung UserDialog für PC-Software
17.2.2016	6 h	Abschluss Programmierung UserDialog für PC-Software
21.2.2016	3 h	Schreiben über RFID
25.2.2016	2.25 h	Schreiben über RFID
27.2.2016	7 h	Schreiben über RFID und User-Modul
28.2.2016	4 h	Fehlerbehebung PC-Software
3.3.2016	4.75 h	Überarbeitung Dokumentation DDS-Software
4.3.2016	2.5 h	Erweiterung RFID-Programmierung
5.3.2016	5.5 h	PC-Software um neue RFID-Funktionen erweitert
6.3.2016	7 h	Zusammenfügen bisheriger Dokumentation
12.3.2016	7.5 h	Erweiterung Software für Richtkoppler, Export für PC-Software
13.3.2016	5 h	PC-Software fertiggestellt
21.3.2016	9 h	Überarbeitung Dokumentation
gesamt	**185.5h**	