Zürcher Hochschule
für Angewandte Wissenschaften

MSE | MASTER OF SCIENCE IN ENGINEERING

zhaw School of Engineering
ZSN Zentrum für Signalverarbeitung und Nachrichtentechnik

Masterarbeit

SW-Realisierung eines DAB-Empfängers mit GNU Radio

Student	Michael Höin, hoinmic@students.zhaw.ch
Arbeitsausgabe	7. März 2011
Arbeitsabgabe	12. August 2011
Betreuer	Prof. Dr. M. Rupf

Zusammenfassung

Das Ziel der vorliegenden Masterarbeit war, die komplette Signalverarbeitung eines Digital Audio Broadcasting Radios zu entwickeln, damit das Zentrum für Signalverarbeitung und Nachrichtentechnik (ZSN) Erfahrungen mit der Implementierung eines breitbandigen OFDM-Systems in Software-Definded Radio (SDR) Technik machen kann.
Um dies zu realisieren wurde als erstes eine Simulation in Matlab erstellt, damit die Signalverarbeitung des OFDM Signals mit dem Digital Audio Broadcasting Standard EN 300 401 schrittweise erarbeitet werden konnte. Mit dem Simulationsprogramm konnte aus einer Aufzeichnung des in der Deutschschweiz ausgesendeten digitalen Ensembles "SRG SSR D01" die Serviceinformationen der Radiostationen und einzelne Audiosequenzen wiedergegeben werden. Die Simulation funktionierte sehr gut und war hilfreich zum besseren Verständnis des Standards.
In einem zweiten Schritt wurde ein Digital Audio Broadcasting Radio auf Basis des Open Source GNU Radio Projekts hergestellt. Die zeitkritische Signalverarbeitung ist in C++ und der Programmablauf in Python programmiert worden. Entstanden ist ein Echtzeit Radio, welches auf einem herkömmlichen Computer läuft und die Audiodaten einem beliebigen Musikplayer zum Abspielen bereitstellt.
Das entwickelte Digital Audio Broadcasting Radio hat mehrere Langzeittests erfolgreich bestanden und funktioniert wie gewünscht. Dank den erarbeiteten Algorithmen kann ein Computer mit etwas Zusatzhardware in ein handelsübliches Digital Radio (Abbildung 1) umgewandelt werden.

Abbildung 1.: Digital Audio Broadcasting Radio

Abstract

The purpose of this master thesis was to develop the complete signal processing of a digital audio broadcasting radio for the Center for Signal Processing and Communications Engineering. The goal is to gain more experience in the implementation of a broadband OFDM system in software defined radio (SDR) technique.
At the beginning a Matlab simulation was created to engineer step by step the OFDM signal processing of the digital audio broadcasting standard EN 300 401. This simulation can generate service information from the radio stations and play back a selected radio station by using a record of the digital ensemble "'SRG SSR D01"'. The simulation worked excellent and helped for a better understanding of the standard.
In a second step, a digital audio broadcasting radio was programmed based on the open source GNU Radio project. Time-critical blocks were programmed in C++ and the program flow was written in Python. The output was a real time radio receiver working on a conventional computer. The produced audio stream may be played on any music player. The produced digital audio broadcasting radio passed several long term tests and works as desired. Therefore, thanks to the developed algorithms, any computer, with some extra hardware, can be turned into a commercial digital radio (figure 2).

Figure 2.: Digital Audio Broadcasting Radio

Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering

Erklärung betreffend das selbständige Verfassen einer Masterarbeit an der School of Engineering

Mit der Abgabe dieser Masterarbeit versichert der/die Studierende, dass er/sie die Arbeit selbständig und ohne fremde Hilfe verfasst hat. (Bei Gruppenarbeiten gelten die Leistungen der übrigen Gruppenmitglieder nicht als fremde Hilfe.)

Der/die unterzeichnende Studierende erklärt, dass alle zitierten Quellen (auch Internetseiten) im Text oder Anhang korrekt nachgewiesen sind, d.h. dass die Masterarbeit keine Plagiate enthält, also keine Teile, die teilweise oder vollständig aus einem fremden Text oder einer fremden Arbeit unter Vorgabe der eigenen Urheberschaft bzw. ohne Quellenangabe übernommen worden sind.

Bei Verfehlungen aller Art treten die Paragraphen 39 und 40 (Unredlichkeit und Verfahren bei Unredlichkeit) der ZHAW Prüfungsordnung sowie die Bestimmungen der Disziplinarmassnahmen der Hochschulordnung in Kraft.

Ort, Datum: Unterschrift:

Das Original dieses Formulars ist bei der ZHAW-Version aller abgegebenen Masterarbeiten zu Beginn der Dokumentation nach dem Abstract bzw. dem Management Summary mit Original-Unterschriften und -Datum (keine Kopie) einzufügen.

Inhaltsverzeichnis

Glossar 4

Literaturverzeichnis 7

Linkliste 8

1. Aufgabenstellung 10

2. Einleitung 12
2.1. Ausgangslage . . . 12

3. Software Defined Radio 13

4. Entwicklungsplattform: GNU Radio 14
4.1. Hardware . . . 15
4.2. Software . . . 16
4.3. GNU Radio Companion . . . 18
4.4. FM Radio in GNU Radio Companion . . . 18

5. OFDM 20
5.1. Grundidee . . . 21
5.2. Signalverarbeitung in einem OFDM Transmitter . . . 23

6. Analyse des DAB Standards um einen DAB Empfänger zu entwickeln 26
6.1. Frames . . . 29
6.2. Modi . . . 29
6.3. Synchronization Channel und Modulation . . . 30
6.4. QPSK symbol mapper und frequency interleaving . . . 33
6.5. Datenblöcke in den OFDM Symbolen . . . 34
6.6. Time Interleaving . . . 35
6.7. Convolution, Puncturing und Viterbidecoder . . . 37
6.8. Energy Dispersal . . . 40
6.9. Fast Information Block . . . 40
6.9.1. Cyclic Redundancy Check . . . 40
6.9.2. Fast Information Group . . . 41
6.10. Reed Solomon und Superframing (DAB+) . . . 42
6.11. Audiocodec MP2 (DAB) . . . 43
6.12. Audiocodec HE-AAC v2 (DAB+) . . . 44

7. DAB Simulation in Matlab mit realen Daten 45
7.1. Signalaufzeichnung . . . 47

7.2. Detektion des Framestarts . . . 47
7.3. Modulation . . . 48
7.4. Frequency Deinterleaving und QPSK Symbol Demapping . . . 50
7.5. Depuncturing und Viterbidecoder . . . 51
7.6. Energy Dispersal . . . 52
7.7. Cyclic Redundancy Check . . . 52
7.8. FIC Auszug und Analyse . . . 52
7.9. Reed Solomon und Superframing . . . 54
7.10. MP2 Audiofile und Netzwerkstream . . . 55
7.11. Audiocodec HE-AAC v2 . . . 55
7.12. Simulink DAB Empfänger . . . 56

8. Implementation **57**
8.1. Erste Implementierung des DAB Empfängers . . . 58
8.1.1. UHD Source und Resampling . . . 58
8.1.2. Framedetektion und Symbolerkennung . . . 60
8.1.3. FFT, differentielle Demodulation und Frequency Interleaving . . . 61
8.1.4. FIC und Subchannel aus den DAB Symbolen herauslesen . . . 62
8.1.5. Time Interleaving . . . 63
8.1.6. Depuncturing . . . 64
8.1.7. Viterbidecoder . . . 64
8.1.8. Energy Dispersal . . . 65
8.1.9. Cyclic Redundancy Check . . . 66
8.1.10. FIB Sink . . . 66
8.1.11. Prearrangement und TCP Sink . . . 66
8.2. Zweite Implementierung des DAB Empfängers . . . 66
8.2.1. Resampling . . . 68
8.2.2. Framestart Detecter und OFDM Symbol Cutter . . . 68
8.2.3. Null Symbol Resampler, Bonder und FIB Cutter . . . 69
8.2.4. FIB Sink 2 und FIB Sink 3 . . . 69

9. Projektabschluss **71**
9.1. Testaufbau . . . 71
9.2. Testprogramme . . . 71
9.3. Fazit . . . 72

10. Ausblick **73**
10.1. DAB/DAB+ . . . 73
10.2. GNU Radio . . . 74

11. Conclusion **75**
11.1. Schlusswort . . . 75
11.2. Danksagung . . . 75

Anhang **76**
A. CD-ROM Wurzelverzeichnis . . . 76
B. Zeitplan . . . 77

C. GNU Radio Installationsanleitungen (Ubuntu 10.10) 78
C.1. UHD Treiber 78
C.2. FPGA Firmware aktualisieren 78
C.3. GNU Radio 78
D. Eigenschaften der erstellten Blöcke 79
E. Hardware 85
F. Geräte 85
G. Programme 86

Glossar

AAC Advanced Audio Codec

ADC Analog Digital Converter

AFG Arbitrary Function Generator

ASCII American Standard Code for Information Interchange

ASCTy Audio Service Component Type

AU Access Unit

BER Bit Error Rate

BPSK Binary Phase Shift Keying

CEN European Committee for Standardisation

CIF Common Interleaved Frame

CORDIC COordinate Rotation DIgital Computer

CPLD Complex Programmable Logic Device

CPU Central Processing Unit

CRC Cyclic Redundancy Check

CU Capacity Units

DAB Digital Audio Broadcasting

DMT Discrete Multitone Modulation

DQPSK Differential Quaternary Phase Shift Keying

DSP Digital Signal Processor

EEP Equal Error Protection

ETSI European Telecommunications Standards Institute

FFT Fast Fourier Transform

FIB Fast Information Block

FIC Fast Information Channel

FIFO First In First Out

FIG Fast Information Group

FPGA Field Programmable Gate Array

GPL GNU General Public License

GNU	GNU's Not Unix
GUI	Graphical User Interface
HE-AAC	High Efficiency Advanced Audio Codec
ICI	Interchannel Interference
IFFT	Inverse Fast Fourier Transformation
IQ	Inphase/Quadraturphase
LAN	Local Area Network
LSB	Least Significant Bit
MCI	Multiplex Configuration Information
MSB	Most Significant Bit
MSC	Main Service Channel
MSE	Master of Science in Engineering
NCO	Numerically Controlled Oscillator
OFDM	Orthogonal Frequency Division Multiplexing
PC	Personal Computer
PI	Puncturing Index
PRBS	Pseudo Random Binary Sequence
QPSK	Quadrature Phase Shift Keying
RCPC	Rate Compatible Punctured Convolutional
RF	Radio Frequency
RSD	Radio Data System
RS	Reed Solomon
RSSI	Received Signal Strength Indication
RX	Receiver
SDR	Software Defined Radio
SMA	Sub Miniature A
SNR	Signal to Noise Ratio
SWIG	Simplified Wrapper and Interface Generator
TX	Transmitter
UEP	Unequal Error Protection
UHD	Universal Hardware Driver
USB	Universal Serial Bus
USRP	Universal Software Radio Peripheral

UTC	Coordinated Universal Time
VLC	Video LAN Client
ZHAW	Zürcher Hochschule für Angewandte Wissenschaften
ZSN	Zentrum für Signalverarbeitung und Nachrichtentechnik

Literaturverzeichnis

[1] *EN 300 401; Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, June 2006.

[2] *TS 101 756; Digital Audio Broadcasting (DAB); Registered Tables*, July 2009.

[3] *TS 102 563; Digital Audio Broadcasting (DAB); Transport of Advanced Audio Coding (AAC) audio*, May 2010.

[4] Hoeg, Wolfgang (Hrsg.); Lauterbach, Thomas (Hrsg.): *Digital Audio Broadcasting, Principles and Applications of DAB,DAB+ and DMB.* Nummer ISBN 978-0-470-51037-7. Wiley & Sons, 3 Auflage, 2009.

[5] Lüders, Prof. Dr. Christian: *Mobilfunksysteme.* Nummer ISBN 3-8023-1847-1. Vogel, 1 Auflage, 2001.

[6] Mathis, Prof. Dr. Heinz: *Signal Processing and Transmission, Part 2a of Lecture Notes.* Spring 2009.

[7] Rupf, Prof. Dr. Marcel: *NTM Praktikum 7, Teil 2, CRC-Verfahren.* 2006.

Linkliste

[A] Ettus Research LLC, Hersteller GNU Radio Hardware (17.5.2011)
http://www.ettus.com/

[B] GNU Radio Homepage (17.5.2011)
http://gnuradio.org

[C] Josh Blums Homepage, Programmierer vom GNU Radio Companion (17.5.2011)
http://www.joshknows.com

[D] Ansammlung von naturwissenschaftlichen Funktionen für Python (25.5.2011)
http://www.scipy.org/

[E] California State University Northridge, GNU Radio Tutorials (31.5.2011)
http://www.csun.edu/~skatz/katzpage/sdr_project/sdrproject.html

[F] Frequenztabelle mit allen DAB Kanälen (14.4.2011)
http://de.wikipedia.org/wiki/T-DAB-Frequenz

[G] Eintrag bei Wikipedia über DAB (14.4.2011)
http://de.wikipedia.org/wiki/Digital_Audio_Broadcasting

[H] Eintrag bei Wikipedia über OFDM (15.4.2011)
http://de.wikipedia.org/wiki/OFDM

[I] Informiert über digitales Radio in der Schweiz (20.4.2011)
http://www.digiradio.ch/

[J] Open source AAC decoder (2.5.2011)
http://www.audiocoding.com

[K] MPEG-4 HE-AAC v2 Decoder für Blackfin Processoren (2.5.2011)
http://www.analog.com/en/processors-dsp/blackfin/bf_mpeg-4_he-aac_v2_decoder/processors/product.html

[L] Eintrag bei Wikipedia über Interleaving (3.5.2011)
http://de.wikipedia.org/wiki/Interleaving

[M] Eintrag bei Wikipedia über Pseudorandom Binary Sequence (5.5.2011)
http://en.wikipedia.org/wiki/Pseudorandom_binary_sequence

[N] Eintrag bei Wikipedia über Software Defined Radio (5.5.2011)
http://de.wikipedia.org/wiki/Software_Defined_Radio

[O] Eintrag bei Wikipedia über GNU Radio (16.5.2011)
http://en.wikipedia.org/wiki/GNU_Radio

[P] Matlab Communications System Toolbox (31.5.2011)
http://www.mathworks.com/products/communications/

[Q] UHD funktioniert noch nicht in Simulink (1.6.2011)
http://lists.ettus.com/pipermail/usrp-users_lists.ettus.com/2011-May/001169.html

[R] Firma die DAB Analysetools herstellt (9.6.2011)
http://www.vadgmbh.de/

[S] Anleitung wie man ein Signalverarbeitungsblock mit GNU Radio schreibt (13.6.2011)
http://www.gnu.org/software/gnuradio/doc/howto-write-a-block.html

[T] Pythagoras Näherungsformel (15.6.2011)
http://www.radartutorial.eu/18.explanations/ex35.de.html

[U] GNU Radio Forum (4.8.2011)
https://lists.gnu.org/archive/html/discuss-gnuradio/

1. Aufgabenstellung

MSE-Masterarbeit
Zentrum für Signalverarbeitung und Nachrichtentechnik

Zürcher Hochschule für Angewandte Wissenschaften
zh aw School of Engineering

Betreuer:	Prof. Dr. M. Rupf	MSE-Student:	Michael Höin
Ausgabe der Anzahl Credits:	7.3.2011 27 bzw. 810h	Abgabetermin: Präsentation:	12.8.2011 Form und Zeitpunkt folgen

Thema: SW-Realisierung eines DAB-Empfängers mit GNU Radio

1. Einleitung und Zielsetzung

Das Zentrum für Signalverarbeitung und Nachrichtentechnik (ZSN) hat schon mehrmals SW-Defined-Radio- bzw. SDR-Systeme realisiert, bisher aber immer bei Nachrichtensystemen mit relativ kleiner Datenrate.

Das ZSN möchte in Zukunft aber auch nachrichtentechnische Untersuchungen an wireless-Systemen mit hoher Datenrate bzw. grösserer Bandbreite machen. Dabei sind vor allem die OFDM-basierten Wireless-Systeme interessant. Breitbandige Systeme können aber auch gut und relativ schnell mit der SDR-Technik realisiert werden.

Mit dem GNU Radio [1] steht ein SW-Toolkit zur Verfügung, mit dem ausgehend von bestehenden SW-Modulen SDR-Systeme aufgebaut werden können. Die Firma Ettus Research [2] bietet HW-Peripherieprodukte zu PCs an,
um SDR-Systeme einfach zu realisieren.

In dieser Masterarbeit soll mit GNU Radio ein breitbandiges OFDM-System realisiert werden, wobei die Realisierung eines DAB-Empfängers Priorität hat.

2. Aufgabenstellung

a) Machen Sie sich mit GNU Radio und der Ettus-HW vertraut. Realisieren Sie dafür einen FM-SW-Empfänger.

b) Studieren Sie existierende DAB-SW-Lösungen, z.B. [3] und [4], sowie DAB-Systeme im Allgemeinen [5].

c) Entwerfen Sie einen DAB-SW-Empfänger mit interessanten Zusatz-Funktionen (fürs Monitoring oder die Daten- auswertung) und besprechen Sie das Konzept mit dem Betreuer.

d) Implementieren Sie den DAB-Empfänger und testen und optimieren Sie ihn.

e) Versuchen Sie, wenn möglich, die Performance mit eigenen Empfangsblöcken zu steigern.

f) Halten Sie in einem Ausblick u.a. fest, wie das ZSN weitere Projektarbeiten mit GNU Radio machen soll.

3. Bericht/Präsentation

Über die MSE-Masterarbeit ist ein Bericht zu schreiben. Der Bericht ist in 2-facher Ausführung (gebundene Papierversion und elektronisch) termingerecht abzugeben.

Verteiler am Ausgabetag: 1
1 Expemplar zH. Kandidatin/Kandidat
1 Exemplar (ohne Beilagen) zH. Schulsekretariat

4. Referenzen

[1] http://en.wikipedia.org/wiki/GNU_Radio
[2] http://www.ettus.com/
[3] http://opendigitalradio.org/index.php/Gnuradio_DAB_constellation_using_gr-dab_and_USRP
[4] http://0x7.ch/text/dab-slides.pdf
[5] W. Hoeg, Th. Lauterbach, “Digital Audio Broadcasting”, Wiley & Sons, 2009.

2

2. Einleitung

Das Zentrum für Signalverarbeitung und Nachrichtentechnik (ZSN) hat seit geraumer Zeit immer mehr mit Software Defined Radios (SDR) gearbeitet. Die realisierten Systeme wiesen aber immer eine sehr geringe Datenrate auf. Deshalb wollte man mit dieser Master-Thesis einen Vorstoss in Systeme mit hohen Datenraten wagen und Wissen aneignen.
Die in dieser Arbeit realisierte Software soll nach dem Projektabschluss in diversen Praktika, Vorlesungen und Informationsveranstaltungen präsentiert werden. Zudem möchte man den Quellcode und die Simulationen auf der Zentrumshomepage veröffentlichen, damit auch andere Hochschulen und Institutionen von den Erkenntnissen profitieren können.

2.1. Ausgangslage

Im Forum von GNU Radio [U] konnte von einer Matlab Digital Audio Broadcasting (DAB) Simulation gelesen werden. Leider antwortete die betreffende Person nicht auf die E-Mailanfrage und andere Quellen durften die Simulation nicht ohne Erlaubnis des Urhebers weitergeben. Darum stand zum Zeitpunkt des Projektstarts keine Simulation zur Verfügung.
In diversen Open Source Projekten wurde versucht ein DAB Sender oder ein DAB Empfänger zu realisieren. Bei genauerer Betrachtung stellte man aber fest, dass die Projekte meist schon lange auf ihre Fertigstellung warteten oder in einer Sackgasse festgefahren sind. Bei gewissen Projekten die auf den ersten Blick viel versprechend waren, war der Sourcecode in einem sehr schlechten, unübersichtlichen oder meist kommentarlosen Zustand, womit eine Weiterführung sinnlos erschien.
Deshalb war es sinnvoll bei der Simulation und der Implementation eine Eigenentwicklung zu starten, um am Ende des Projekts einen zuverlässigen DAB Empfänger zu erhalten.

3. Software Defined Radio

Bei einem Software Defined Radio (SDR) wird die analoge Hardware auf ein Minimum reduziert und die komplette Signalverarbeitung digital bewerkstelligt [N]. Die Abbildung 3.1 zeigt die Struktur eines idealen SDR Empfängers. In der Realität findet man sehr selten ideale SDR. Das Problem ist die Geschwindigkeit der Analog Digital Converter (ADC). Möchte man ein Signal mit einer maximalen Frequenz von 1 GHz verarbeiten, müsste die Taktfrequenz des ADCs und der Hardware (nach dem Nyquist Theorem) bei mindestens 2 GHz liegen. Ein ADC mit dieser Geschwindigkeit ist heutzutage sehr teuer. Eine Möglichkeit dieses Problem zu entschärfen ist das Vorschalten eines Mischers und eines Anti-Aliasing-Filters. Anschliessend kann die Wandlung und die Verarbeitung auf einer Zwischenfrequenz stattfinden.
Der Vorteil von einem SDR ist die Flexibilität auf Erneuerungen zu reagieren. Ohne die Hardware zu modifizieren, können mit einem Firmwareupdate Erneuerungen in ein Produkt eingebunden werden. Wenn eine funktionierende Datenverbindung vorhanden ist, kann man alle Algorithmusprobleme sogar via Fernwartung lösen.
Der Nachteil liegt bei den Anschaffungskosten und beim Stromverbrauch der Signalverarbeitungshardware (FPGA, DSP, Mikrokontroller usw.). Da die Prozesserleistungen jedoch stetig zunehmen, wird dieses Problem immer kleiner.

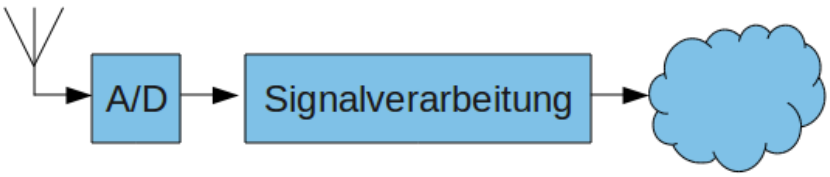


Abbildung 3.1.: Aufbau eines idealen SDR Empfängers

4. Entwicklungsplattform: GNU Radio

Inhalt

4.1. Hardware **15**
4.2. Software **16**
4.3. GNU Radio Companion **18**
4.4. FM Radio in GNU Radio Companion **18**

GNU Radio ist ein Open Source SDR Projekt in welchem versucht wird möglichst viel der Signalverarbeitung im Personal Computer (PC) zu erledigen. Gestartet hat das Projekt im Jahr 2001. Der kluge Kopf hinter dem Projekt ist Eric Blossom, er hat den Code hauptsächlich kreiert [B]. Das Ziel der GNU Radio Programmierer ist es, den softwareorientierten Personen einen einfachen Einstieg in den Bereich der elektromagnetischen Wellen zu ermöglichen. Dies wird erreicht in dem die digitale Signalverarbeitung so nahe wie möglich an die Antenne gerückt wird und sich somit Hardwareprobleme in Softwareprobleme umwandeln lassen.
GNU Radio ist Hardware unabhängig, somit kann zur Signalverarbeitung jeder beliebige Computer verwendet werden. Gängige Computer enthalten Prozessoren mit mehreren Kernen, wobei jeder Kern mit Taktraten um 3 GHz läuft. Somit ist man in der Lage mehrere Milliarden Floatingpoint- Operationen in einer Sekunde zu verarbeiten. Ein Elektro-Ingenieur stellt sich GNU Radio am besten als Mikrokontroller mit externem Funkmodul vor, wobei der Mikrokontroller den PC mit seiner immensen Rechenleistung und sehr viel Speicher darstellt.
Das Projekt wurde unter den Vertragsbedingungen der GNU's Not Unix (GNU) General Public License (GPL) veröffentlicht. [O]

4.1. Hardware

Die von GNU Radio einwandfrei unterstützte Hardware wird von Ettus [A] hergestellt. Die Ettus Hardware wird Universal Software Radio Peripheral (USRP) genannt. Ein USRP besteht aus einem Daughterboard und einer Hauptplatine. Die Abbildung 4.1 zeigt ein USRP N210 (dritte Generation) mit aufgesetztem WBX Daughterboard. Die genauen Daten der eingesetzten USRP N210 Hardware können im Anhang der Tabelle E.23 entnommen werden.

Abbildung 4.1.: Ettus USRP N210 Hardware mit WBX Daughterboard

Die Abbildung 4.2 zeigt das Funktionsprinzip des Empfängerpfads in der USRP Hardware. Das Daughterboard wird zum Herunter- oder Hinaufmischen der Trägerfrequenz gebraucht. Um dies gezielt zu bewerkstelligen enthält der Aufsatz einen Numerically Controlled Oscillator (NCO) der von der Hauptplatine gesteuert werden kann. Das zu verarbeitende Signal bezieht das Board vom Sub Miniature A (SMA) Anschluss, wenn die Hardware als Receiver (RX) arbeitet oder es wird vom FPGA ausgegeben, wenn die Hardware als Transmitter (TX) arbeitet. Es ist auch möglich beide Pfade der Hardware gleichzeitig zu gebrauchen und das USRP als Transeiver einzusetzen. Auf dem Daughterboard befinden sich zudem verschiedene Anti-Aliasing-Filter.
Auf der Hauptplatine sind ein FPGA, die Stromversorgung, die ADCs und die Schnittstellen zum Computer. Über den Gigabit Ethernet Anschluss kann über das FPGA der Signalfluss gesteuert werden. Die Überwachung des NCOs geschieht ebenfalls über das FPGA. Die gewandelten ADC Samples werden im FPGA auf die gewünschte Rate reduziert, gefiltert und an die Gigabit Ethernet Schnittstelle weitergegeben (USRPs der ersten Generation wurden über die USB Schnittstelle angeschlossen).
Der Computer kann die USRP Box über den Universal Hardware Driver (UHD) ansteuern. Dieser läuft unter allen gängigen Betriebssystemen.

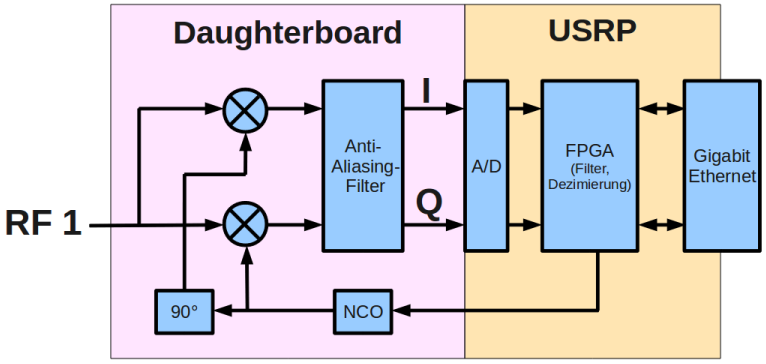


Abbildung 4.2.: Funktionsprinzip der USRP GNU Radio Hardware (RX Pfad)

4.2. Software

Ein GNU Radio Programm besteht aus so genannten Blöcken. Diese Blöcke können die Eigenschaften einer Quelle, eines Abflusses oder einer Verarbeitungseinheit annehmen. Um ein Programm zu erstellen werden mehrere Blöcke zu einem Flow Graph zusammengeführt und miteinander verbunden. Die einzelnen Blöcke können eine unterschiedliche Anzahl Ein- und Ausgänge besitzen.
Quellen und Abflüsse dienen zur Kommunikation mit der Welt ausserhalb des Programms. Blöcke die nur Ausgänge und keine Eingänge aufweisen werden als Quellen bezeichnet. Sie holen ihre Daten ausserhalb des Programms (USRP, Netzwerkverbindung, lesen eines Files auf der Festplatte usw.) und stellen die Daten über ihre Ausgänge anderen Blöcken bereit. Abflüsse haben dagegen nur Eingänge und keine Ausgänge. Sie stellen das Ende eines Signalzweiges dar. Über sie können Daten für den Benutzer sichtbar, gespeichert oder weitergeleitet werden. Das eigentliche Verarbeiten der Daten findet in den zwischengeschalteten Signalverarbeitungseinheiten statt.
Alle Blöcke werden in C++ geschrieben, da die Verarbeitung oft zeitkritisch ist. Das Bilden und Ausführen eines Flow Graphs geschieht hingegen in Python. Um die beiden Sprachen miteinander in einem Flow Graph vereinen zu können, wird der Simplified Wrapper and Interface Generator (SWIG) eingesetzt.
Das schöne an GNU Radio ist, dass mehrere hundert Blöcke bereits existieren und eingesetzt werden können. Der Umfang der Quellen Blöcke beginnt beispielsweise mit einfachen Signalquellen wie einem Sinus-Generator oder einer Rauschquelle und geht hin bis zu komplexen Signalgeneratoren. Bei den Abflüssen ist es ähnlich, beispielsweise ist ein Block zum abspielen der Daten auf einer Soundkarte vorhanden. Auch für die Signalverarbeitungsblöcke steht eine Vielzahl von Blöcken aus verschiedenen Bereichen der Mathematik und Physik bereit.
Um die Einfachheit von GNU Radio zu zeigen, ist im Listing 4.1 ein lauffähiges GNU Radio Programm (Programmiersprache Python) abgebildet. Das Programm besteht aus einem Sinusgenerator, einer Multipliziereinheit und der Soundkarte als Abfluss. Zuoberst werden die Bibliotheken in das Programm eingebunden. Die ersten beiden Importe sind Standardbibliotheken, welche von GNU Radio zur Verfügung gestellt werden. Die Bibliothek “howto“ wurde selbst geschrieben und stellt nun Signalverarbeitungsblöcke für den Flow

Graph bereit. Im Beispiel wird aus dieser Bibliothek ein Block zur Multiplikation eingesetzt. Der Flow Graph kann grob in drei Bereiche aufgeteilt werden. Im ersten werden die Variabeln definiert, im zweiten die Signalverarbeitungsblöcke mit den gewünschten Parametern instanziiert und im dritten Teil die Ports der Blöcke miteinander verknüpft. Der eigentliche Programmstart dieses einfachen Beispiels findet durch das Instanziieren des Flow Graphs (In diesem Beispiel die Klasse my_top_block) und durch aufrufen der Run-Funktion auf Zeile 32 statt. Das GNU Radio Programm bietet den Vorteil, dass problemlos Python Bibliotheken eingebunden werden können. Ein bekanntes Beispiel sind die Bibliotheken SciPy und Numpy [D], sie enthalten eine Ansammlung von naturwissenschaftlichen Funktionen. Mit dem einbinden dieser Bibliotheken hat man in der Pythonumgebung ähnliche Funktionen zur Verfügung wie in Matlab. Es gibt unzählige solcher Python-Bibliotheken die frei verfügbar sind und mit denen der Funktionsumfang erweitert werden kann.

```
1  #!/usr/bin/env python
2
3  from gnuradio import gr
4  from gnuradio import audio
5  import howto
6
7  class my_top_block(gr.top_block):
8      def __init__(self):
9          gr.top_block.__init__(self)
10
11         ######################
12         # Variablen
13         ######################
14         rate = 44000
15         amplitude = 5
16         factor = 0.005
17
18         ######################
19         # Objekte (Bloecke) instanziieren
20         ######################
21         src = gr.sig_source_f (rate, gr.GR_SIN_WAVE, 350, amplitude)
22         modif = howto.multiply (factor)
23         dst = audio.sink (sample_rate, "")
24
25         ######################
26         # Bloecke verbinden
27         ######################
28         self.connect ((src,0), (modif,0), (dst,0))
29
30 if __name__ == '__main__':
31     try:
32         my_top_block().run()
33     except KeyboardInterrupt:
34         pass
```

Listing 4.1: Python Beispielcode

4.3. GNU Radio Companion

GNU Radio Companion ist eine Möglichkeit den Flow Graph graphisch zusammenzuklicken und anschliessend in ein ausführbares Python File umzuwandeln. Geschrieben wurde das Programm von Josh Blum [C]. GNU Radio Companion eignet sich ausgezeichnet für den Laborbetrieb an einer Hochschule, da mit diesem Programm eine neue Dimension der Übersichtlichkeit von Signalverarbeitungseinheiten erreicht wird. Zudem können visuelle Blöcke wie beispielsweise ein FFT- Abfluss zur Demonstration schnell hinzugefügt oder entfernt werden.
Der Nachteil von GNU Radio Companion ist, dass nicht alle Eigenschaften der Blöcke zugänglich sind, die man bei einem komplexen Programm benötigt. Zudem sind viele Blöcke für den Gebrauch in GNU Radio Companion noch nicht angepasst worden. Ein weiterer Nachteil ist, dass ein Flow Graph auf eine einzige Zeichnungsoberfläche begrenzt ist und somit nicht mehrere Files (wie in einem Printlayoutprogramm) durch Ports miteinander verknüpft werden können.
Zusammengefasst heisst das, dass bei einem grösseren Projekt wegen den genannten Punkten nicht mit GNU Radio Companion gearbeitet werden kann. Für kleine Versuchsprogramme oder für eine Messaufzeichnung ist dieses Programm sehr gut geeignet, da man mit ihm Ziele schnell erreicht.

4.4. FM Radio in GNU Radio Companion

Mit ein paar Klicks ist es möglich ein FM Radio in GNU Radio Companion zu erstellen. Die Abbildung 4.3 zeigt die graphische Anordnung der Blöcke und die Abbildung 4.4 zeigt das Graphical User Interface (GUI) während des Betriebs.
In diesem Beispiel wird mit einer USRP N210 Hardware eine Bandbreite von 4 MHz eingelesen. Die Zenterfrequenz der Abtastung kann über ein Slider (WX GNU Slider) während des Betriebs zwischen 90 MHz und 110 MHz eingestellt werden. Damit man graphisch erkennt auf welchen Frequenzen sich Radiosender befinden, wurde am Ausgang der USRP Box ein FFT-Plot und ein FFT-Wasserfall-Plot angehängt.
Die Bandbreite der USRP Box wurde extra gross gewählt, damit man bei den FFT-Plots einen grossen Spektrumsausschnitt sieht. Um einen FM Radiosender zu empfangen reichen 250 kHz Bandbreite aus (Dies konnte in wenigen Minuten mit einem Regler in GNU Radio Companion ermittelt werden). Deshalb hat man einen Filter mit Ratendezimierung nachgeschaltet (Frequency Xlating FIR Filter). Die FM Demodulation bewerkstelligt eine FM Demodulator Box. Damit die Soundkarte nicht Signale mit zu grosser Amplitude erhält und die Lautstärke während des Betriebs geregelt werden kann, wurde noch ein Lautstärkeregler in Form eines einfachen Multiplikators in die Demodulationskette eingebaut und ebenfalls mit einem WX GNU Slider verbunden. Der demodulierte Datenstrom endet nach dem Multiplikator in der Soundkarte.

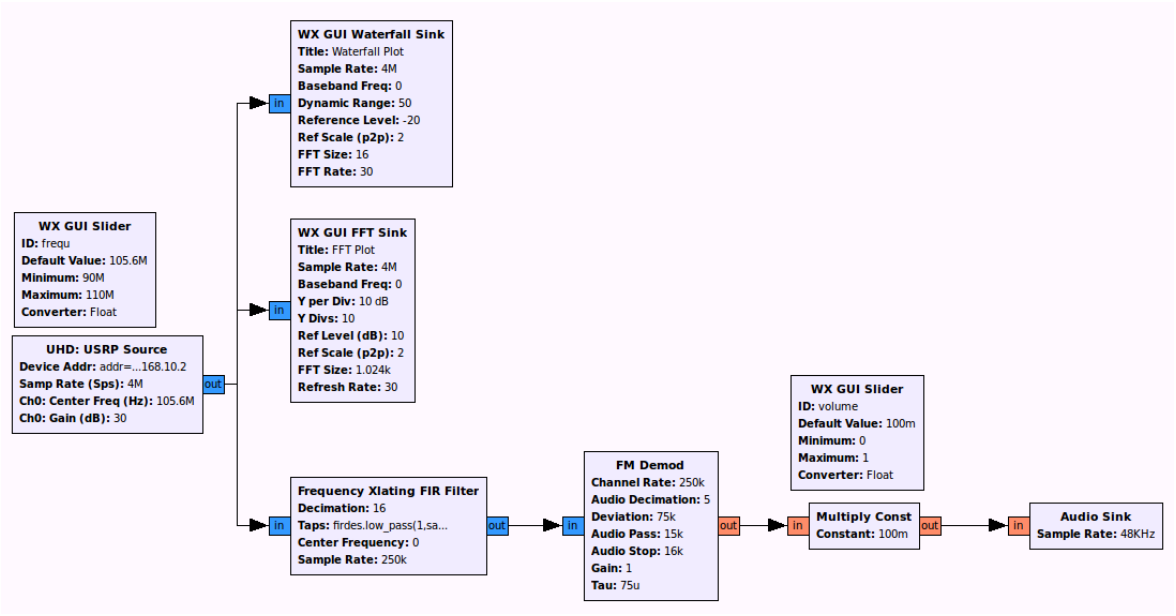


Abbildung 4.3.: Graphisches Erstellen des Flow Graphs

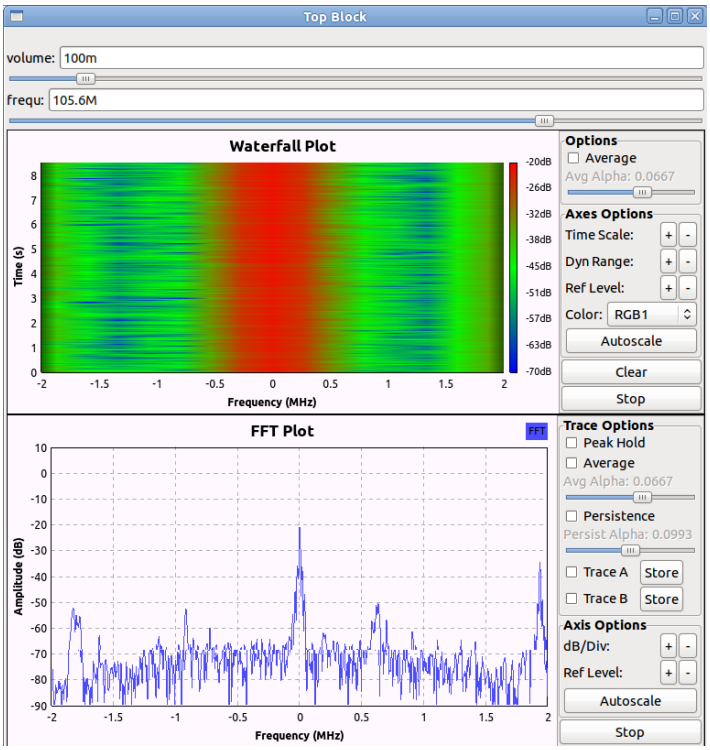


Abbildung 4.4.: GUI des Flow Graphs

5. OFDM

Inhalt

5.1. Grundidee 21
5.2. Signalverarbeitung in einem OFDM Transmitter 23

Beim Orthogonal Frequency Division Multiplexing (OFDM) wird das zur Verfügung stehende Spektrum in viele kleine Spektren aufgeteilt. In jedem einzelnen dieser Subchannels werden auf den Trägern gleichzeitig Datenbits übermittelt und es geht kein Teil des Spektrums verloren. Diese Methode wird auch Discrete Multitone Modulation (DMT) genannt. Der Abstand zwischen zwei Subcarrier ist beim OFDM Verfahren direkt von der Symboldauer in einem Subchannel abhängig. Es wird stets darauf geachtet, dass die Träger orthogonal zueinander sind. Mit dieser Orthogonalität wird ein Übersprechen in einen anderen Subchannel (interchannel interference (ICI)) verhindert. Um ein Übersprechen in den Nachbarkanal bei einem Symbolwechsel zu verhindern, wird an das Symbol ein Guard Intervall angehängt. [6, Seite 32]
Ein Vorteil von OFDM ist, wenn eine Störung bei der Übertragung in einem Subchannel auftritt, kann dieser bei der nächsten Übertragung ausgelassen werden und es findet nur eine geringe Reduktion der Datenrate statt. Hätte man wie bei anderen Verfahren nur einen Träger, könnte eine schmalbandige Störung die komplette Übertragung unbrauchbar machen. [H]

5.1. Grundidee

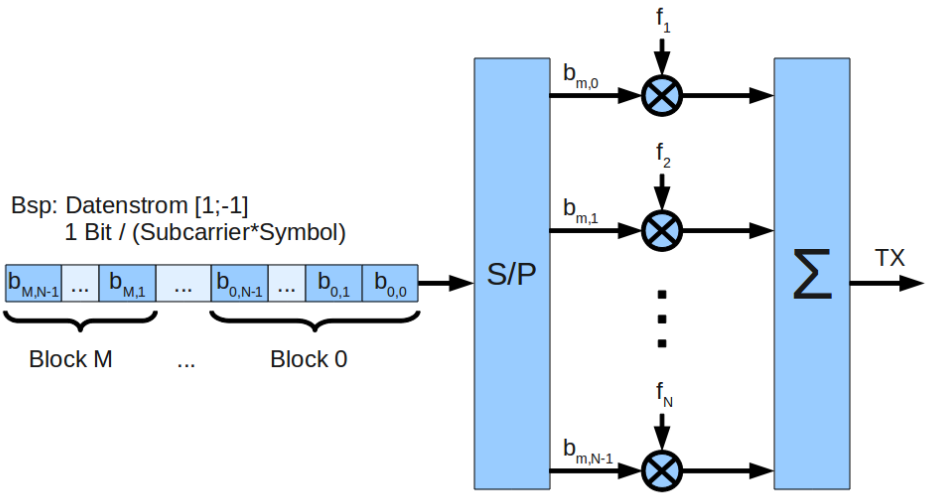


Abbildung 5.1.: Generierung eines OFDM Symbols

Die Abbildung 5.1 zeigt die Struktur um ein OFDM Signal zu erzeugen. Dabei wird ein Datenstrom in Blöcke aufgeteilt. Wird jeder Subcarrier Binary Phase Shift Keying (BPSK) moduliert, weisen die Blöcke jeweils die gleiche Anzahl Bits auf wie die Anzahl Subcarrier die zur Verfügung stehen, da pro Symbol ein Bit übertragen wird. Wird jedoch eine Quadrature Phase Shift Keying (QPSK) Modulation verwendet, werden auf jedem Subcarrier zwei Bit pro Symbol übertragen. Somit müssen auch die Blöcke doppelt so gross gewählt werden. Dieses Verfahren kann auf jede beliebige Modulation angewendet werden.

$$f_n = \frac{n}{T_s} \rightarrow \text{für } n = \mathbb{Z} \text{ ohne } 0 \tag{5.1}$$

$$\begin{aligned} f_n &= \text{Frequenz Subcarrier n} \\ n &= \text{Subcarrier Index} \\ T_s &= \text{Symboldauer} \end{aligned}$$

Bestimmen der Frequenzen der Subcarrier

Damit sich die Subcarrier im Spektrum nicht gegenseitig beeinflussen, muss jede Subcarrierfrequenz ein Vielfaches des Kehrwertes der Symboldauer aufweisen (Gleichung 5.1). Für n sind sowohl positive als auch negative Zahlen möglich. $n = 0$ wird weggelassen, da ein DC Wert entstehen würde. Mit dieser Methode zur Frequenzbestimmung kommt immer eine Nullstelle des modulierten Subcarriers auf den Träger des Nachbarkanals zu liegen. Die Abbildung 5.2 verdeutlicht dies. Ist diese Bedingung erfüllt, wird von orthogonalen Signalen gesprochen und die Gleichung 5.2 ist erfüllt.

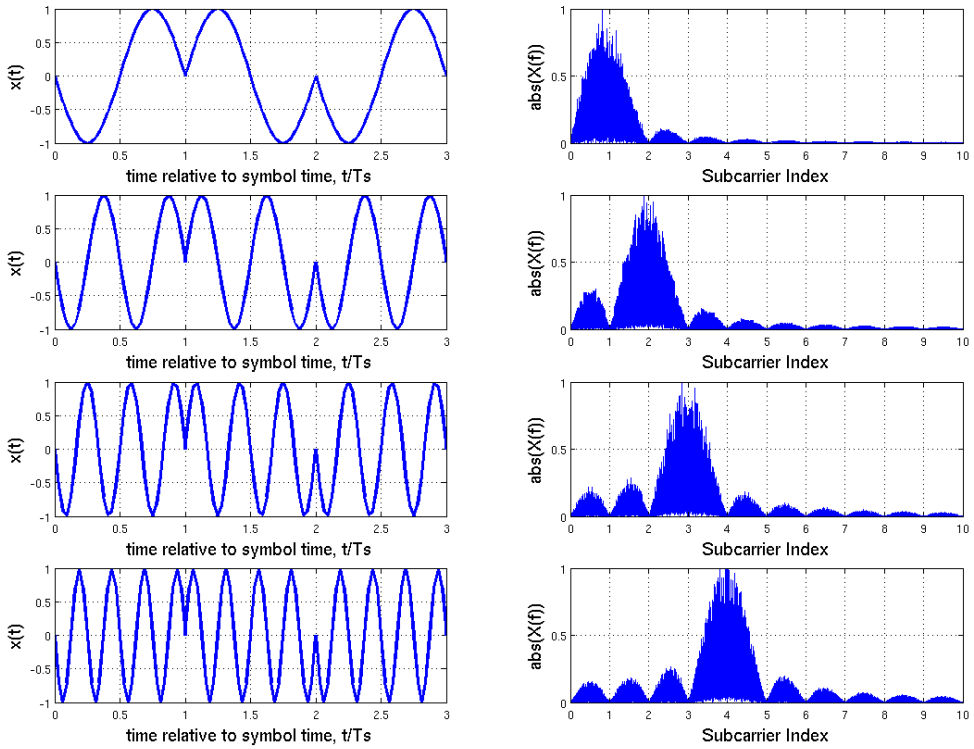


Abbildung 5.2.: Zufällige BPSK Sequenzen auf verschiedenen Subcarrier

$$\frac{1}{T_s} * \int_0^{Ts} \mathrm{e}^{j*2*\pi*f_i*t} * e^{-j*2*\pi*f_j*t}\,\mathrm{d}t = \begin{cases} 0 = \text{orthogonal}, & \mathrm{i} \neq \mathrm{j} \\ 1 = \text{nicht orthogonal}, & \mathrm{i} = \mathrm{j} \end{cases} \tag{5.2}$$

$$\begin{aligned} t &= \text{Zeit} \\ f_i &= \text{Frequenz Subcarrier i} \\ f_j &= \text{Frequenz Subcarrier j} \\ T_s &= \text{Symboldauer} \end{aligned}$$

Orthogonalitätsprüfung durch bilden des Skalarprodukts von zwei Subcarrier [6, Seite 33]

Meist wird der Kanal in mehrere hundert Subchannels aufgeteilt. Zum Beispiel bei DAB wird der Kanal für die Übermittlung der Daten in bis zu 1536 Subchannels unterteilt. Durch diese Unterteilung und die lange Dauer der Symbole auf den jeweiligen Trägern, kann mit einem Spektrumanalyser ein OFDM Spekturm wie in Abbildung 5.3 gemessen werden. Gegenüber anderen Modulationen weist ein OFDM Spektrum extreme Steilheiten an den Kanalgrenzen (In der Abbildung bei ± 768 MHz) auf.

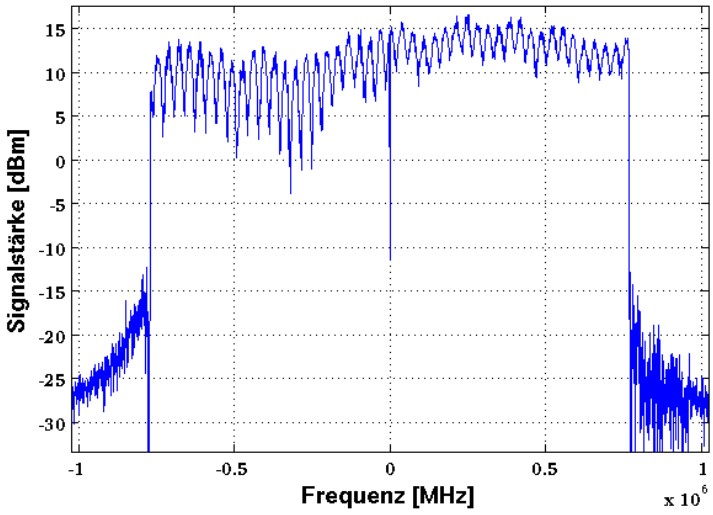


Abbildung 5.3.: DAB OFDM Symbol mit 1536 Subcarrier, zentriert auf 227.360 MHz

5.2. Signalverarbeitung in einem OFDM Transmitter

Die Struktur aus der Abbildung 5.1 kann in der Inversen Fast Fourier Transformation (IFFT) zusammengefasst werden. Dazu betrachtet man die Daten eines Datenblocks als Koeffizienten für die IFFT und bildet daraus das zu übermittelnde Signal. Zur Demodulation benötigt der Empfänger nur einen Fast Fourier Block (FFT) und kann aus den Phasen der berechneten Fourier Koeffizienten die Daten bestimmen.

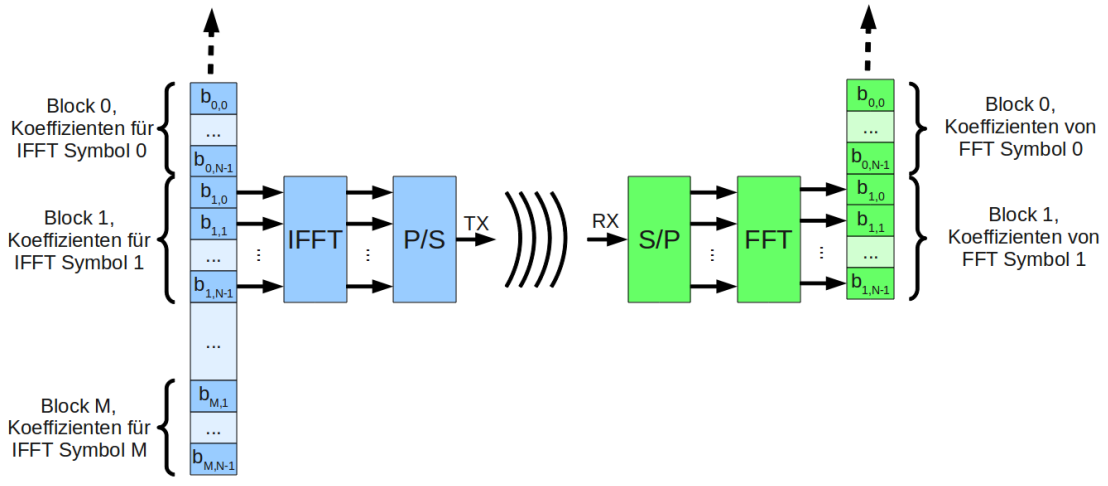


Abbildung 5.4.: Aufbau eines OFDM Senders und eines Empfängers

Da der Sender und der Empfänger nicht phasenstarr sind, wird bei einer Übertragung zu Beginn normalerweise ein Referenzsymbol auf jeden Subcarrier geschickt (Abbildung 5.6)

und die nachfolgenden Daten werden differenziell zum vorangehenden Symbol übertragen. Der Empfänger muss bei dieser Übertragungstechnik die Phasenveränderung zwischen zwei Symbolen auswerten. Der Vorteil ist nun, dass die absolute Phase auf den einzelnen Subcarrier keine Rolle mehr spielt und somit die Distanz zwischen Sender und Empfänger (welche für den Phasenunterschied verantwortlich ist) egal ist.

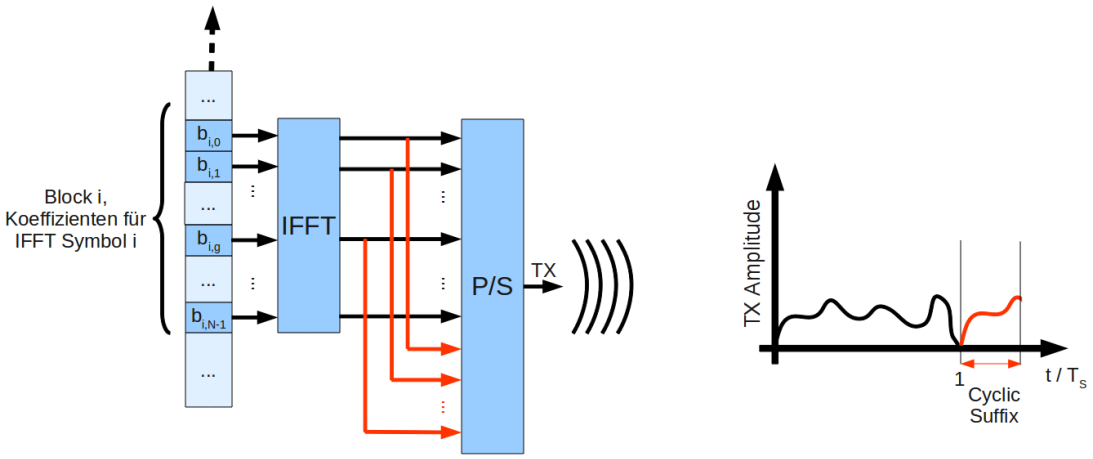


Abbildung 5.5.: Anhängen des Cyclic Suffix

Oftmals wird bei einer OFDM Übertragung die Übertragungszeit der Symbole etwas länger als T_s gewählt (Abbildung 5.5). Dies nennt man das Anhängen eines Ciclic Suffix oder manchmal auch Guard Intervall. Im Grunde wird auf jedem Subcarrier das aktuelle Symbol weiter gehalten. Für dieses Suffix wird kein grösseres Spekturm benötigt, da die Symbole bezogen auf die Symboldauer zyklisch sind und daher kein Phasensprung stattfindet.
Mit diesem Anhängsel bietet man dem Empfänger die Möglichkeit mit einer gewissen Symbol-Synchronisationstoleranz zu arbeiten (Abbildungen 5.6), der FFT Bereich des Empfängers kann irgendwo innerhalb des Symbols liegen. Zudem benötigt der Sender zwischen den Symbolen immer eine gewisse Zeit bis das neue Symbol stabil anliegt. Mit dem Ciclic Suffix kann nun ebenfalls garantiert werden, dass mindestens für die Dauer einer Symbolperiode ein Symbol stabil anliegt.
Beim festlegen der Symboldauer und dem Guard Intervall müssen gewisse Grenzen beachtet werden, damit kein Fading auftritt. Einerseits möchte man die Anzahl Subcarrier sehr gross wählen damit man eine lange Symboldauer erreicht, kein frequenzselektiver Schwund stattfindet und eine grosse coherence Bandbreite entsteht. Andererseits darf die Symboldauer nicht zu gross gewählt werden, damit die Zeitinvarianz des Kanals über das ganze Symbol konstant ist. Optimal wäre es die Parameter immer den umliegenden Gegebenheiten anzupassen, beispielsweise über das Signal to Noise Ratio (SNR) [6, Seite 43]. In den meisten standardisierten Systemen werden jedoch die Parameter festgelegt damit sie in mehreren Gebieten funktionieren. Mit dieser Methode befindet man sich unterhalb des Optimums, aber es vereinfacht den Aufbau eines Empfängers wesentlich.

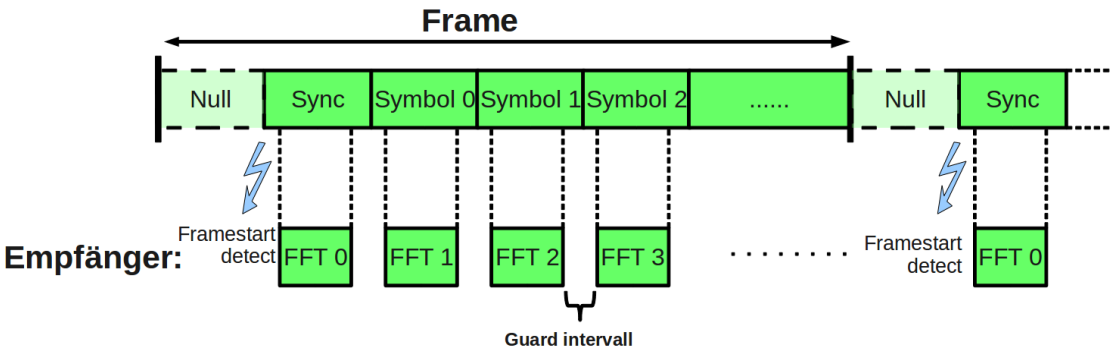


Abbildung 5.6.: Aufbau eines Übertragungsframes

Damit der Empfänger auf die einzelnen Blöcke synchronisieren kann, wird als erstes Symbol ein so genantes Null-Symbol übertragen. Während dieser Zeit wird keinerlei Energie übermittelt. Mit der Information wann das Frame startet, kann der Empfänger die Zeitbereiche festlegen über welche er eine FFT bildet 5.6.

6. Analyse des DAB Standards um einen DAB Empfänger zu entwickeln

Inhalt

6.1. Frames **29**
6.2. Modi **29**
6.3. Synchronization Channel und Modulation **30**
6.4. QPSK symbol mapper und frequency interleaving **33**
6.5. Datenblöcke in den OFDM Symbolen **34**
6.6. Time Interleaving **35**
6.7. Convolution, Puncturing und Viterbidecoder **37**
6.8. Energy Dispersal **40**
6.9. Fast Information Block **40**
6.9.1. Cyclic Redundancy Check 40
6.9.2. Fast Information Group 41
6.10. Reed Solomon und Superframing (DAB+) **42**
6.11. Audiocodec MP2 (DAB) **43**
6.12. Audiocodec HE-AAC v2 (DAB+) **44**

Der Digital Audio Broadcasting (DAB) Standard [1] wurde für die Übertragung von qualitativ guten digitalen Audiosignalen und zum Übermitteln von verschiedenen Datenservices erstellt. Mit diesen Voraussetzungen können neben den normalen Sprechdaten viele zusätzliche Informationen wie alternative AM oder FM Frequenzen oder ganze Homepages zur Visualisierung einer Information dem Empfänger bereit gestellt werden [1, Seite 10]. Zur Verbreitung von komprimierten Verkehrsinformationen wurde die Radio Data System (RDS) Kodierung der FM Sender übernommen und in einen digitalen Traffic Message Channel (TMC) implementiert [1, Seite 127]. Es können aber auch reine digitale broadcasting Daten ohne Audioinhalt der Öffentlichkeit zugänglich gemacht werden. Der Frequenzbereich der Orthogonal Frequency Division Multiplexing (OFDM) Übertragung liegt zwischen 30 MHz und 3 GHz und ist somit für den Terristrischen-, den Kabel- und den Satellitenempfang ausgelegt [F].
Audio DAB und Audio DAB+ weisen bei der Signalübertragung geringe Unterschiede auf, welche im eingesetzten Fehlerschutz (DAB -> UEP, DAB+ -> EEP, Reed Solomon und Superframing) und im Audiocodec des Main Service Channels liegen (DAB -> MP2, DAB+ -> HE AAC v2). Beim neueren DAB+ Standard hört der Hörer bei gleicher Übertragungsrate eine bessere Audioqualität. Diese Qualitätssteigerung geht zu Lasten der Rechenleistung. Dies wird bewusst in Kauf genommen, da sich diese zwischen dem Beginn

der Entwicklungen des DAB Standards (1987) und dem DAB+ Release (2001) deutlich gesteigert hat [G].
Eine Herausforderung bei der Entwicklung des DAB Standards war eine grosse Robustheit gegenüber physikalischen Effekten zu erlangen, damit das System in allen Regionen auf der Erde unter verschiedensten Bedingungen einwandfrei funktioniert. Einer dieser Effekte ist die Mehrwegausbreitung, welche durch das Reflektieren der elektromagnetischen Wellen von umliegenden Objekten verursacht wird. Ein Anderer ist der Dopplereffekt, der durch einen Geschwindigkeitsunterschied zwischen einem Sender und einem Empfänger entsteht [G, Seite 29-32]. All diesen Effekten wird mit entsprechender Wahl der Übertragungsparameter Rechnung getragen. Das System ist weltweit einsetzbar.
Die Abbildung 6.1 zeigt die Struktur mit allen wichtigen Komponenten eines DAB Empfängers. Auf jeden einzelnen Block wird im Folgenden eingegangen.

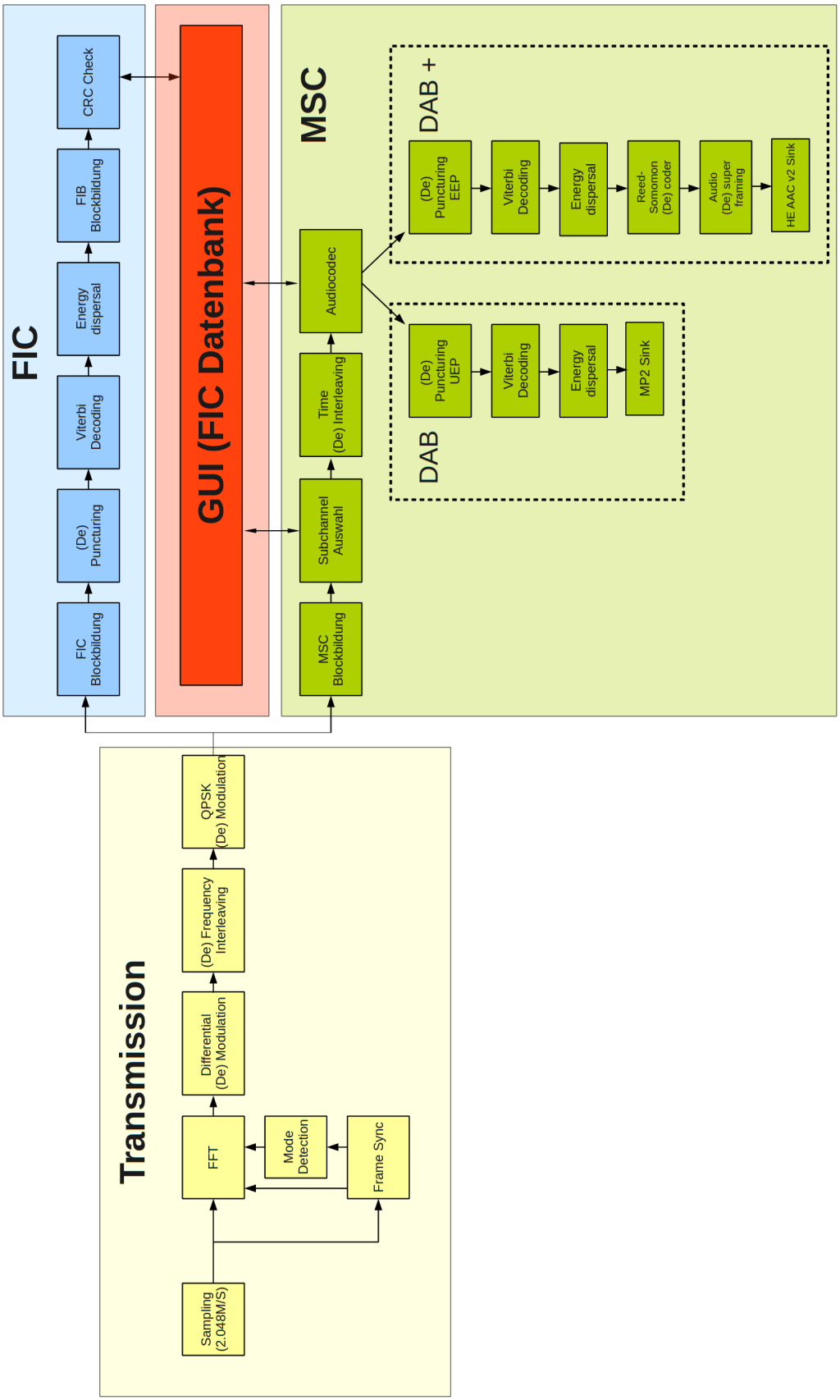


Abbildung 6.1.: Empfängerübersicht

6.1. Frames

Bei einer DAB Übertragung werden Frames nacheinander ohne Unterbruch durch das Medium geschickt. Die Abbildung 6.2 zeigt den Aufbau eines Frames. Jedes Frame besteht einstellungsunabhängig immer aus drei Bereichen (Synchronization-, Fast Information- und Main Service Channel). Je nach Übertragungsart werden unterschiedlich viele OFDM Symbole und Subcarrier in einem Frame übermittelt und somit unterscheidet sich die mögliche gesamt Datenrate einer Verbindung [1, Seite 144].

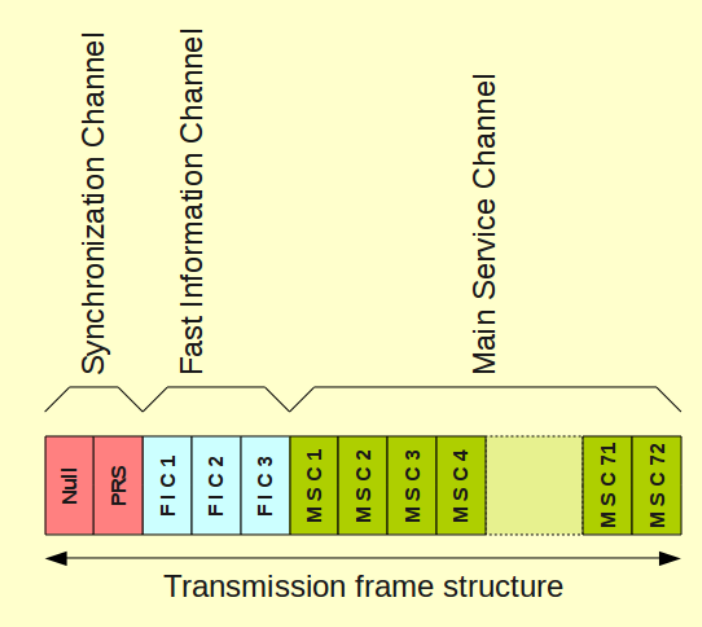


Abbildung 6.2.: Aufbau eines DAB Frames, Mode 1

6.2. Modi

Für DAB Übertragungen wurden vier verschiedene Betriebsmodi definiert, damit das System gut auf die topografischen Eigenschaften einer Region angepasst werden kann (in der Schweiz wird ausschliesslich der Modus 1 ausgesendet [I]).
Die Grafik 6.3 visualisiert die Parameter eines Frames. Die Abbildung 6.4 liefert die zugehörigen Parameter für die im Standard definierten Betriebsmodi. Als Zeitbasis wird T mit 1/2'048'000 Sekunden definiert [1, Seite 145]. Mit Hilfe dieser Angabe kann die Aussage, dass eine Samplingfrequenz von 2.048 MHz (oder ein Vielfaches davon) optimal wäre, getroffen werden.
Die von einer Übertragung insgesamt benötigte Bandbreite (1.536 MHz) ist in allen Modi gleich. Die OFDM Symboldauer hingegen variiert zwischen den Modi. Ist die Dauer kürzer, werden weniger Subcarrier parallel übermittelt und umgekehrt.

Interessant ist die jeweilige Grösse der Symboldauer (T_U). Wird die Samplingfrequenz wie erwähnt 2.048 MHz gewählt, ist die besagte Dauer immer eine Zweierpotenz der Zeitbasis

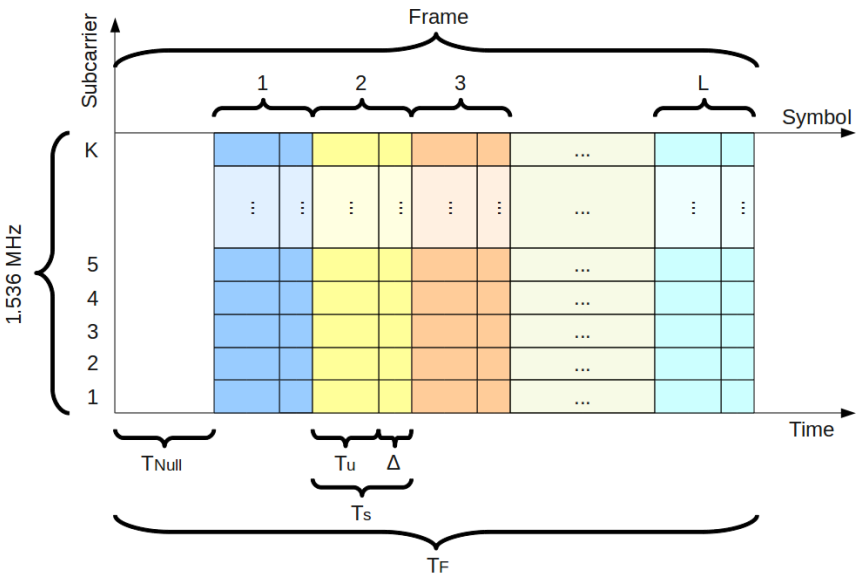


Abbildung 6.3.: Visualisierung der Frameparameter

Parameter	Mode 1	Mode 2	Mode 3	Mode 4
K	1536	384	192	768
T_F	196608 T	49152 T	49152 T	98304 T
T_{NULL}	2656 T	664 T	345 T	1328 T
T_S	2552 T	638 T	319 T	1276 T
T_U	2048 T	512 T	256 T	1024 T
Δ	504 T	126 T	63 T	252 T

Abbildung 6.4.: Parameter der verschiedenen Modi

(Beispielsweise Mode 1: $T_U = 2^{11}T = 2048$T). Über die Samples eines Symbols lässt sich daher sehr effektiv eine FFT bilden.
Ein Empfangssystem muss vor dem Verarbeiten der Daten den Übertragungsmodi erkennen um überhaupt richtig arbeiten zu können. Dies kann elegant durch das Überwachen der Zeitabstände der Nullsymbole (T_F) und der Nullsymboldauer (T_{NULL}) erreicht werden.

6.3. Synchronization Channel und Modulation

Im Synchronization Channel werden keine Daten übermittelt. Die zwei beinhalteten Symbole dienen einzig zum Synchronisieren des Empfängers.
Mit dem Null Symbol wird dem Empfänger eine klare Referenz für den Start eines Frames übermittelt. Während der Dauer dieses Symbols (etwas länger als ein datenbehaftetes

Symbol) wird keine Energie übertragen. Die Abbildung 6.5 zeigt schön den Received Signal Strength Indication (RSSI) Einbruch beim Empfänger.

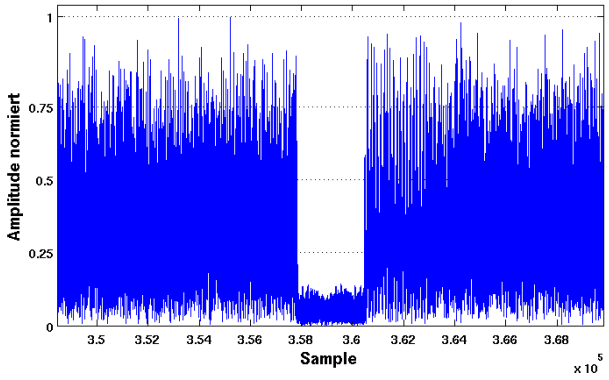


Abbildung 6.5.: Null Symbol einer DAB Aufzeichnung

Mit dem Phasenreferenzsymbol wird dem Empfänger die Referenz für die differenzielle Modulation geliefert. Dabei sind die Phasen der einzelnen Subcarrier dieses Symbols im Standard [1, Seite 149] vorgegeben. Mit dieser Massnahme wird verhindert, dass viele Subcarrier gleichzeitig die gleiche Phase aufweisen, was nach der Addition zu einem grossen Peak führen würde [1, Seite 147].

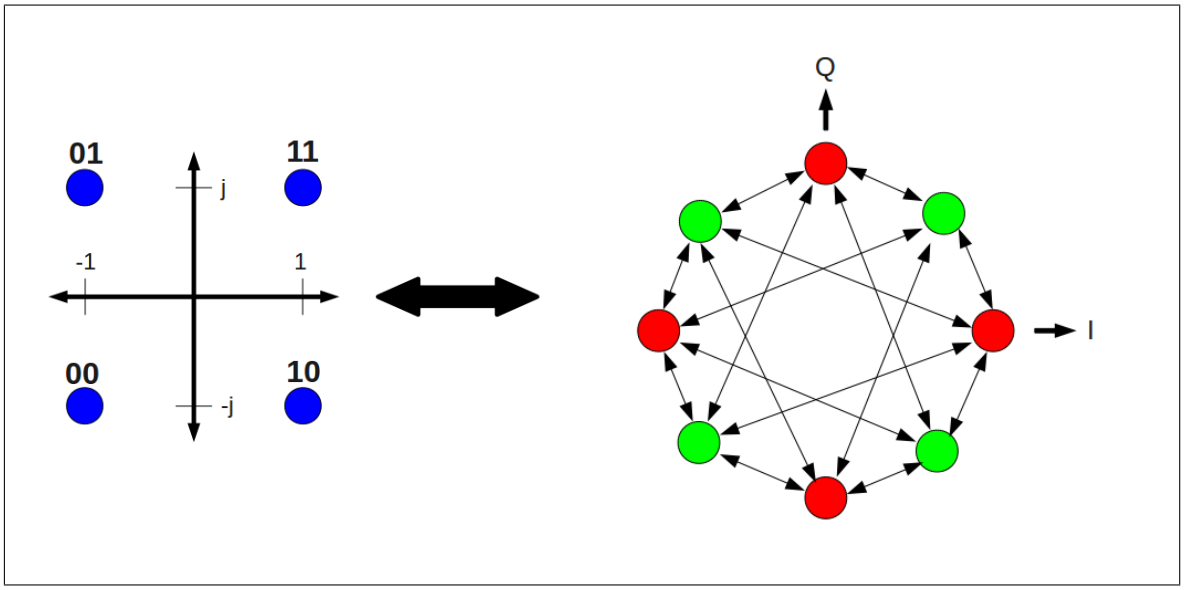


Abbildung 6.6.: Links ein QPSK Signal (vier Phasenzustände im Zustandsraum), Rechts ein DQPSK Signal (acht Phasenzustände im Zustandsraum) mit allen möglichen Phasenübergängen

Als Modulation wird Differentiell Quadrature Phase Shift Keying (DQPSK) verwendet. Bei dieser Modulation wird jeweils die Phase des neu anzuliegenden Quadrature Phase Shift

$$Z_{l,k}\angle a_{l,k} = Z_{l-1,k}\angle a_{l-1,k} * Y_{l,k}\angle b_{l,k} \tag{6.1}$$
$$a_{l,k} = a_{l-1,k} + b_{l,k} \tag{6.2}$$

Z = Amplitude eines DQPSK Symbols
Y = Amplitude eines QPSK Symbols
a = Winkel eines DQPSK Symbols
b = Winkel eines QPSK Symbols
l = Symbol Nummer
k = Subcarrier Index

Keying (QPSK) Symbols zur aktuellen Phase eines Subcarriers dazu addiert und übertragen (Gleichung 6.1). Die Abbildung 6.6 verdeutlicht alle möglichen Phasen die bei der DQPSK Modulation erreicht werden können. Durch jedes neue Symbol dreht sich die Phase mindestens um $\Pi/4$, deshalb folgt auf einen roten Zustand immer ein grüner und umgekehrt [1, Seite 161]. Mit dieser Modulation wird zudem vermieden, dass sich die Phase um 180 Grad ändert und die Anteile auf dem I und dem Q Kanal gleichzeitig durch den Nullpunkt führen.

$$Z_{l,k}\angle a_{l,k} * Z_{l-1,k}\angle a_{l-1,k}{}^{*} = Y'_{l,k}\angle b_{l,k} \tag{6.3}$$
$$a_{l,k} - a_{l-1,k} = b_{l,k} \tag{6.4}$$

Z = Amplitude eines DQPSK Symbols
Y' = Amplitude eines demodulierten QPSK Symbols
a = Winkel eines DQPSK Symbols
b = Winkel eines QPSK Symbols
l = Symbol Nummer
k = Subcarrier Index

Bei dieser Modulation steckt die Information in der Phase. Für die Demodulation eines differenziellen Signals muss die Gleichung 6.1 zu Gleichung 6.3 umgebaut werden. Dabei wird eine komplexe Multiplikation mit dem komplex konjugierten Vorgängersymbol ausgeführt. Mit dem komplexkonjugierten Symbol wird aus der Winkeladdition eine Winkelsubtraktion. Somit gewinnt man mit dieser Rechnung wieder die Phase des ursprünglichen QPSK Signals und kann daraus die Information herauslesen [4, Seite 336].

6.4. QPSK symbol mapper und frequency interleaving

Da für die DQPSK Modulation als Quelle ein QPSK Symbol dient, werden pro Subcarrier zwei Bit übertragen. Die total übertragene Anzahl Bits pro OFDM Symbol beträgt daher zwei mal die Anzahl Subcarrier. Die Vorschrift zur Symbolbildung ist in der Gleichung 6.5 zu finden. Betrachtet man die Gleichung genauer fällt auf, dass die zu übermittelnden Datenbits in zwei Gruppen aufgeteilt werden. Die führenden K Bits bilden den Realanteil des Symbols, die folgenden K Bits bilden den Imaginäranteil [1, Seite 157]. Mit dieser Massnahme können die Bits des Datenstroms über das OFDM Symbol verstreut werden. Im Falle einer Störung sind weniger Bits aus derselben Region betroffen als wenn ein Bit den Realanteil und das darauf folgende den Imaginärteil bestimmt.
Um dieses Mapping in einem Empfänger rückgängig zu machen, können die ursprünglichen Bits eines QPSK Symbols mit der Auswertung des Real- und Imaginäranteils wieder bestimmt werden. Im Buch über digitales Audio Broadcasting [4, Seite 337] wird erwähnt, dass es sich lohnt die Anteile der empfangenen Daten mit einer Auflösung von 10 Bit pro Datenbit weiterzuführen und keine Harddecision anzuwenden. Mit dieser Massnahme funktioniert der nachgeschaltete Viterbidecoder mit grösserer Zuverlässigkeit, da stark empfangene Symbole eine klarere Bitentscheidung hervorrufen als schwache.

$$q_{l,n} = \frac{1}{\sqrt{2}} * [(1 - 2 * p_{l,n}) + j * (1 - 2 * p_{l,n+K})] \text{ für n=0,1,2,...,} K - 1 \tag{6.5}$$

q = QPSK Symbol
l = OFDM Symbol Nummer
n = QPSK Symbol Nummer
p = Bit im Datenstrom
K = Anzahl Subcarrier

Mappen der QPSK Symbole

Um eine Übertragung noch robuster gegen schmalbandige Störungen zu machen, wird bei der Zuweisung eines QPSK Symbols zu einem Subcarrier ein so genanntes Frequency Interleaving angewendet. Dabei wird im Standard [1, Seite 157-161] nach einer modiabhängigen Regel die Zuweisung (Gleichung 6.6) definiert. Somit erreicht man eine Symboldurchmischung bei der sich auf zwei benachbarten Subcarrier nie zwei benachbarte Datenbits befinden.
In einem Empfänger muss das angewendete Interleaving in umgekehrter Richtung vollzogen werden, damit sich die Symbole wieder in der ursprünglichen Reihenfolge befinden und verarbeitet werden können.

$$y_{l,k} = q_{l,n} \tag{6.6}$$

q = QPSK Symbol
y = QPSK Symbol für einen Subcarrier
l = OFDM Symbol Nummer
n = QPSK Symbol Nummer
k = Subcarrier

Frequency Interleaving

6.5. Datenblöcke in den OFDM Symbolen

Im Fast Information Channel (FIC) werden modiabhängig pro DAB Frame zwischen einem und vier faltungcodierte Fast Information Blöcke (FIB) übermittelt. Für den in der Schweiz ausgestrahlten Mode 1 bedeutet dies eine Übertragung von vier Blöcken pro Frame (Abbildung 6.7) [1, Seite 151]. In einem Block befinden sich 2304 Bits (Mode 1,2,4) oder 3072 Bits (Mode 3), was bei vier Blöcken im Mode 1 insgesamt 9216 Bits entspricht. Im FIC werden Informationen zu den enthaltenen Services (Radiosender, Datenströme usw.), der Aufbau des Multiplexverfahrens sowie regionsabhängige Information übermittelt [4, Seite 35]. Da diese Bits zur Steuerung des Empfängers dienen, sind sie sehr wichtig und werden deshalb gut geschützt.

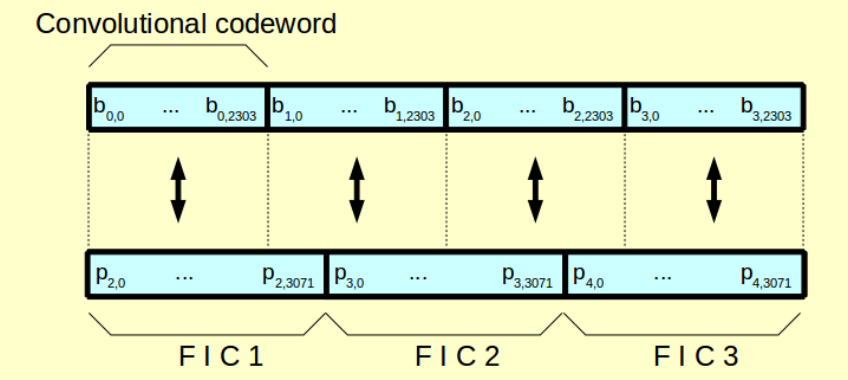


Abbildung 6.7.: Vier Blöcke des Fast Information Channels in drei OFDM Symbolen (Mode 1)

Ein Empfänger sollte die Umschachtelung der empfangenen Datenbits zurück zu den Convolutional Codewords wie in den Abbildungen 6.7 und 6.8 vollziehen, da die Codewords die Grundlage für die noch folgenden Signalverarbeitungselemente bilden und mit der Umschachtelung wird die richtige Länge der Wörter sichergestellt.

Im Main Service Channel (MSC) werden die Common Interleaved Frames (CIF) übertragen. Sie enthalten die Daten der Services. In einem DAB Frame werden bis zu vier CIFs übermittelt (Abbildung 6.8). Jedes CIF enthält 24ms Audiodaten jedes Radiosenders oder Daten der Datenservices. Ein CIF besteht modiunabhängig aus 55296 Datenbits. Bei der Zuweisung der Datenbits zwischen den Daten der CIFs und dem QPSK Symbol Mapping findet keinerlei Interleaving statt [1, Seite 154].

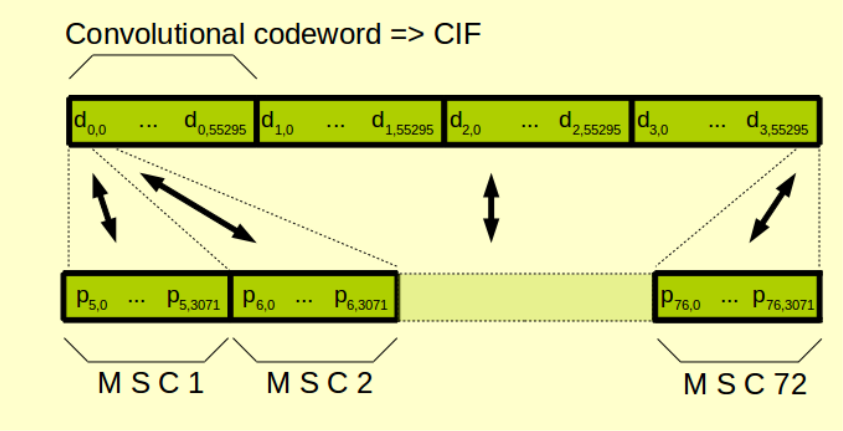


Abbildung 6.8.: Drei Blöcke des Main Service Channels verteilt auf 72 OFDM Symbole (Mode 1)

Die Bits in einem CIF werden in 64 Bits grosse Capacity Units (CU) aufgeteilt (Abbildung 6.9). Im FIC Datenstrom wird in gewissen Abständen die Position der einzelnen Services innerhalb der CIFs und die Anzahl CUs eines Services vom DAB Provider festgelegt und übertragen. Die Aufteilung eines CIFs in Subchannels ist in der Abbildung eingezeichnet [1, Seite 143]. Mit diesen Informationen ist man in der Lage gewünschte Services weiterzuverarbeiten und die Übrigen von der Signalverarbeitung auszuschliessen, somit wird Rechenleistung gespart. Wie viele CU für einen Service verbraucht werden, kann vom Ensemblebetreiber definiert werden. Theoretisch könnte laut Standard für einen einzigen Service die ganze Kapazität verbraucht werden. Normalerweise befinden sich jedoch einige Services im gleichen CIF. Bei einer DAB Audioübertragung widerspiegelt die Datenrate gleichzeitig die Audioqualität. Für DAB+ Übertragungen werden bei gleicher Audioqualität wesentlich weniger CUs für einen Audioservice benötigt als für DAB.

6.6. Time Interleaving

Die einzelnen Subchannels des MSC werden mit einem Time Interleaving versehen um die Datenübertragung resistenter gegen Burstfehler zu machen. Burst stören normalerweise nur für kurze Zeit die Kommunikation. Damit nicht mehrere aufeinander folgende Bits von einem Burst betroffen sind wird das Time Interleaving (Verzögertes Übertragen von

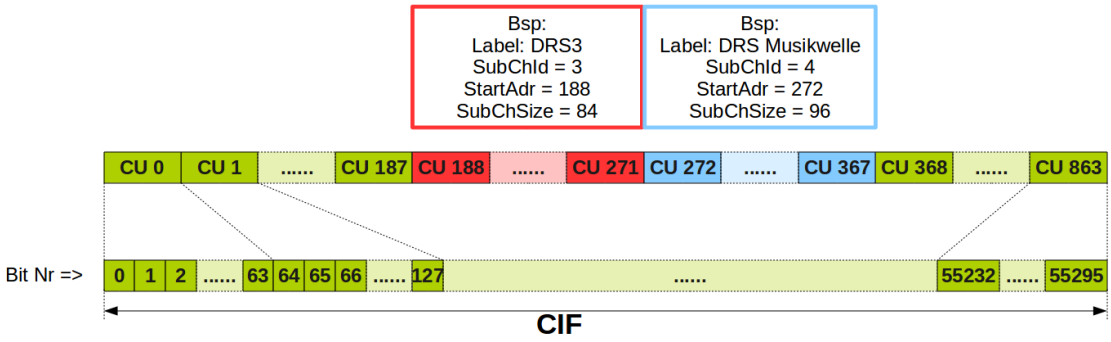


Abbildung 6.9.: Common Interleaved Frame

gewissen Bits) angewendet. Im Empfänger werden dank dieses Schrittes aus Burstfehler automatisch mehrere verteilte Einzelfehler, sofern der Burst nicht zu lange war. Einzelfehler kann ein System viel besser korrigieren als mit Burstfehler, da bei einer Übertragung meistens zusätzliche Fehlerkorrekturbits übermittelt werden [L].

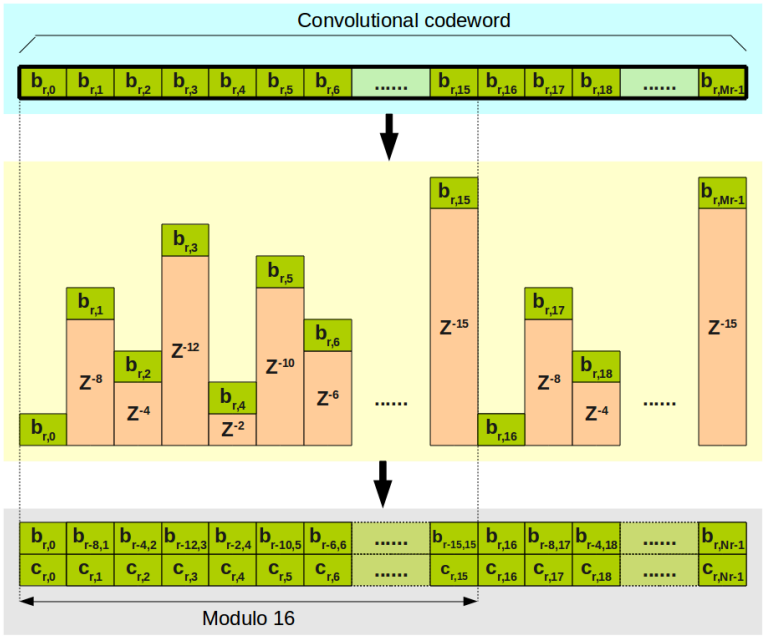


Abbildung 6.10.: Time Interleaving angewendet auf ein Datenstrom

Bei DAB wird der Datenstrom eines Subchannels für das Time Interleaving in 16 Einheiten aufgeteilt (Bitnummer modulo 16 = Einheit), somit sind nie zwei aufeinander folgende Bits in der selben Einheit (Abbildung 6.10). Jede Einheit erfährt nun eine andere Verzögerung bis zur Übertragung. Eine Verzögerung der Grösse 1 bedeutet, dass die Einheit um ein CIF verzögert wird. Zur Bestimmung der Verzögerung wird die Nummer der Einheit mit den so

genannten Bit Reverse Law verrechnet (Bsp: Einheit 8 = '1000' Binär -> '0001' Binär = 1 Verzögerung) [4, Seite 42].
Auf Seiten des Empfängers muss das Time Interleaving zwischen den Einheiten wieder ausgeglichen werden (Abbildung 6.11). Dazu muss jede Einheit verzögert werden, bis ihr gesamthaft ein Delay von 15 CIF widerfahren ist (Time Interleaving des Senders eingerechnet).
Betrachtet man eine Datenübertragung, fällt einem eine generelle Verzögerung der Daten auf. Die Verzögerung beträgt insgesamt 15*24ms = 360ms. Da es sich bei DAB um eine Einwegkommunikation handelt und der Sender keine Antwort erwartet, merkt man diese Verzögerung nicht.

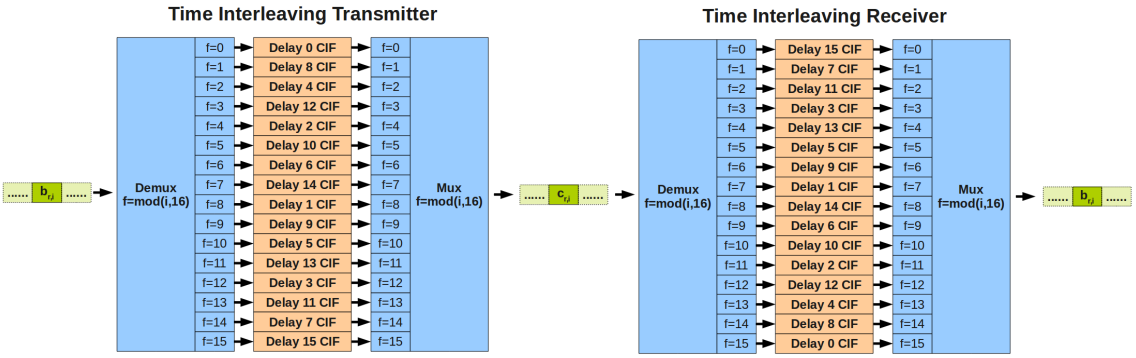


Abbildung 6.11.: Links: Time Interleaving Transmitter, Rechts: Time Interleaving Receiver

6.7. Convolution, Puncturing und Viterbidecoder

DAB enthält verschiedene Fehlerschutzcodierungen. Dies bedeutet eine grosse Flexibilität bei der Zuweisung einer Schutzcodierung für einen Subchannel. Um diese grosse Flexibilität realisieren zu können, wird Rate Compatible Punctured Convolutional (RCPC) eingesetzt. Es funktioniert ratenunabhängig immer mit der gleichen Hardware, nur die Redundanz wird verändert.
Ein RCPC System besteht aus zwei Elementen, dem Faltungscodierer (Convolutional Coder) zum realisieren einer Vorwärtsfehlerkorrektur und einer nachgeschalteten Punktiereinheit um eine bestimmte Rate zu realisieren [4, Seite 36].
Der Faltungscodierer ist in der Abbildung 6.12 gezeichnet. Er besteht aus einem Schieberegister, XOR Verknüpfungen und Addierer [5]. Für jedes Bit am Eingang generiert er vier Bits am Ausgang, was einer Rate von 8/32 = 1/4 entspricht. Diese Bitfolge am Ausgang wird als Mother Code bezeichnet. Das Generatorpolynom lautet in oktaler Schreibweise: 133, 171, 145 und 133 [1, Seite 130]. Der Codierer wird Blockweise benutzt (Gleichung 6.7). Zu Beginn einer Faltung enthalten alle Speicherelemente eine Null (Null-Zustand). Am Ende eines Blocks wird eine bestimmte Anzahl Nullen in den Decoder gelassen damit sicher wieder der Null-Zustand erreicht wird. Diese Bits des Ausklingens werden als Tail bezeichnet.

$$(a_i)_{i=0}^{I-1} \rightarrow (x_{0,i}, x_{1,i}, x_{2,i}, x_{3,i})_{i=0}^{I+5} \tag{6.7}$$

I = Inputbitvektor
X = Codewort inklusive Tail

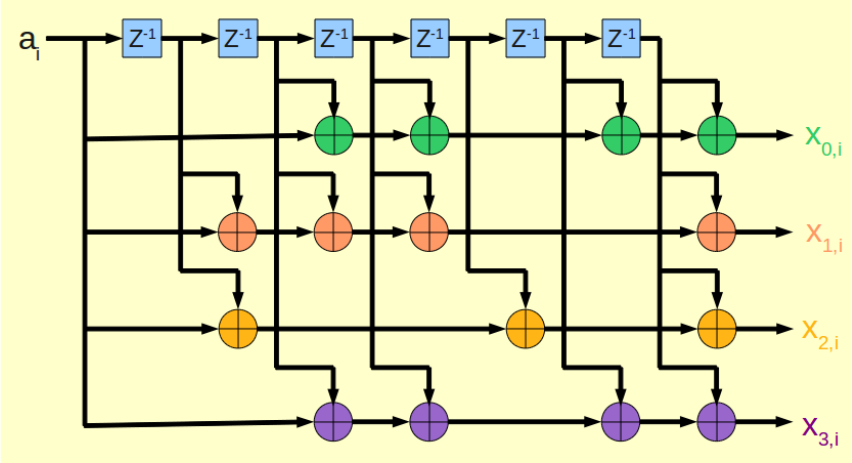


Abbildung 6.12.: Faltung

Nach dem Faltungscodierer folgt das Punktierelement (Puncturingelement). Je nach gewünschter Rate lässt dieses Element Bits des Mother Codewords weg. Es sind Raten zwischen 8/9 und 8/32 möglich, wobei es sich bei einer Rate von 8/32 um das ganze Mother Codeword ohne Punktierung handelt.
Für die Punktierung wird mit 32 Bits grossen Blöcken des Faltungscodeworts gearbeitet. Die Bits werden nach einer Tabelle im Standard [1, Seite 131] punktiert. Zur Wahl der Punktierung aus der Tabelle dient ein Puncturing Index (PI). Die Abbildung 6.13 verdeutlicht das Vorgehen des Punktierens für zwei unterschiedliche Raten. Die schwarz eingekreisten Bits werden an das Time Interleaving weitergegeben.

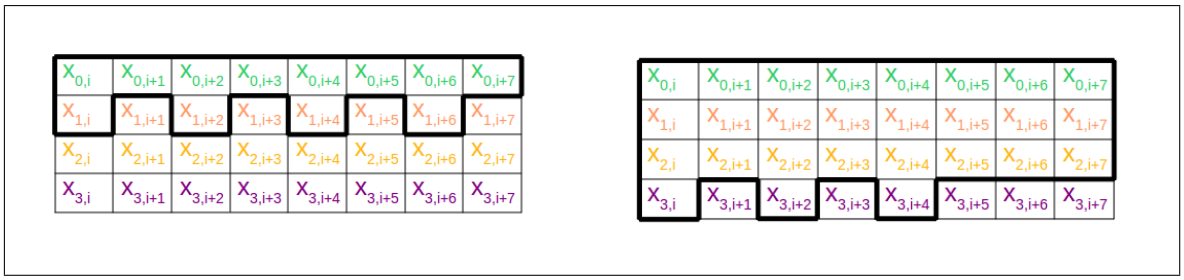


Abbildung 6.13.: Links ein Puncturing mit einer Rate von 8/12=2/3, Rechts ein Puncturing mit einer Rate von 8/27

Im FIC arbeitet man für den Faltungscodierer mit 768 Bits grossen Inputvektoren [1, Seite 132], das ergibt am Ausgang ein 3072 langes Codewort und 24 Tailbits. Für das Punktieren wird der Vektor in drei verschiedene Elemente aufgeteilt. Der erste Teil wird mit PI = 16 geschützt, der zweite mit PI = 15 und der Tail mit PI = 8. Somit weist der vorderste Bereich des Codeworts die beste Fehlerschutzcodierung auf und der Tail die Schlechteste. Aus den anfangs 767 Bits werden 2304 Bits generiert und übermittelt. Die Abbildung 6.14 zeigt die Abhandlung im FIC.

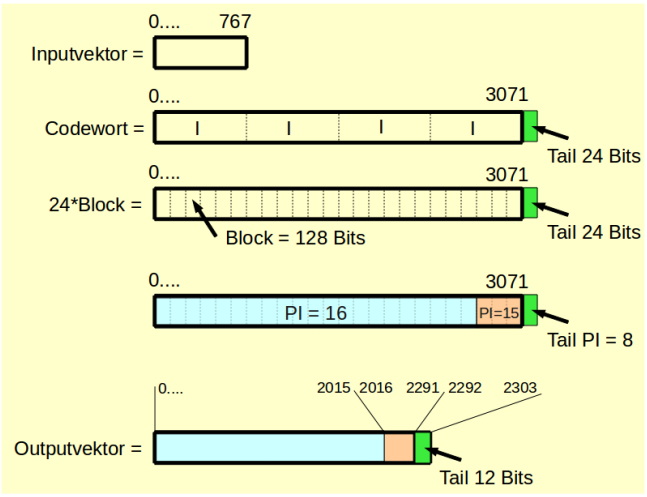


Abbildung 6.14.: RCPC Prozedur im FIC

Da die Kanalkapazität bei einer Übertragung begrenzt ist und vom Provider definiert wird, geht jeder Anbieter eines Services einen Kompromiss zwischen dem Erreichen einer hohen Datenrate (um beispielsweise beim Empfänger eine gute Audioqualität zu erreichen) und der Wahl einer guten Schutzstufe (um auftretende Fehler korrigieren zu können), ein. Im MSC wird zwischen zwei Schutzmöglichkeiten unterschieden: Unequal Error Protection (UEP) und Equal Error Protection (EEP).
UEP bietet die Möglichkeit gewisse Datenbereiche stark zu schützen und andere nur sehr wenig. Dies passt zu MP2 Audioframes, wie sie in DAB eingesetzt werden. In einem Frame befinden sich vorne der Header und alle wichtigen Audioinformationen. Hinten sind akustisch eher unwichtige Daten, welche bei Fehlübertragungen nur einen geringen Einfluss auf die Audioqualität haben. Insgesamt wird ein DAB Audioframe in vier unterschiedlich geschützte PI Bereiche aufgeteilt. Der Standard definiert datenratenabhängig bis zu fünf Schutzprofile mit individuellen PIs für die einzelnen Frameabschnitte.
Bei EEP wird ein Frame in zwei Teile mit annähernd gleichem Fehlerschutz aufgeteilt. Somit werden alle Bits in einem Frame gleich redundant übermittelt und sind deshalb gleich geschützt. Dieser Schutz wird für Datendienste und für die Audioframes bei DAB+ verwendet. Der Standard definiert insgesamt acht verschiedene Schutzmöglichkeiten mit unterschiedlichen PIs. Die Frameunterteilung für die Punktierung wird durch die Datenrate definiert.
Um in einem Empfänger das Punktieren und Wegkürzen von Daten sowie die Faltung rückgängig zu machen, werden an den ursprünglich punktierten Stellen im Datenstrom

Nullen eingebaut um anschliessend mit einem Viterbidecoder die Ursprungsdaten bestmöglich herausarbeiten zu können. Die Nullen dienen als neutrale Platzhalter und bewirken bei einem bipolaren Datenstrom in einem Decoder keine Veränderung. Die Positionen der Punktierungen sind dem Empfänger bekannt, da der eingesetzte Fehlerschutz eines Services im FIC bekannt gegeben wird. Der nachgeschaltete Viterbidecoder bestimmt die am wahrscheinlichsten gesendete Ursprungssequenz.

6.8. Energy Dispersal

In einem Datenblock treten oft ganze Ketten von Nullen oder Einsen auf. In einigen Fällen bedingt durch die Framestruktur und in anderen hervorgerufen durch das Auffüllen von ungebrauchten Bits in einem Datenstrom. Damit diese Datenfolgen keinen Einfluss auf die Performance von Signalverarbeitungselementen (zum Beispiel der Faltungsencoder) haben, werden die Daten vor der Weiterverarbeitung über ein XOR Element mit einer Pseudo Random Binary Sequence (PRBS) verknüpft [1, Seite 128]. Eine PRBS weist das Spektrum des weissen Rauschens auf [M]. Der Generator für die Zufallssequenz besteht aus einem linear rückgekoppelten Schieberegister (Abbildung 6.15).
Gut an einer XOR Verknüpfung ist, dass durch das erneute Anwenden der PRBS in einem Empfänger wieder die Ursprungsdaten entstehen und keine empfängerspezifische Anpassung gemacht werden muss.

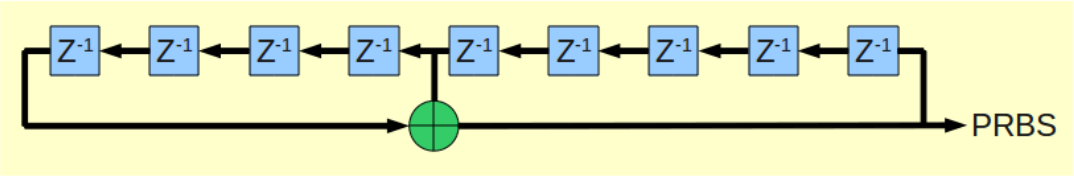


Abbildung 6.15.: Generierung der Pseudo Random Binary Sequence

6.9. Fast Information Block

Dieser Abschnitt erklärt welche Daten die FIBs enthalten und wie sie geschützt sind. Zudem wird aufgezeigt wie die einzelnen Informationen in einer Datenbank zusammengeführt werden können, um ein reibungsloses Aufschalten eines Services aus dem MSC zu ermöglichen.

6.9.1. Cyclic Redundancy Check

Die FIBs aus dem FIC enthalten alle Informationen um den Datenstrom der einzelnen Services im MSC bedienen zu können. Da fehlerhafte Daten in den FIBs fatale Folgen für eine Übertragung haben, enthält jeder Block in den letzten Bits ein Cyclic Redundancy

Check (CRC) Wort zur Verifizierung der vorhergehenden Daten [1, Seite 29]. Die Abbildung 6.16 verdeutlicht die Struktur zur Bildung des CRC Wortes, speziell am Wort ist die Komplementbildung vor dem Einbinden in den FIB.
Um die Gültigkeit eines FIBs in einem Empfänger zu überprüfen, kann die gleiche Struktur verwendet werden wie für die CRC Bildung. Im Empfänger wird das gesamte FIB inklusive des Komplements des empfangenen CRC Wortes in den CRC Block gegeben. Weist nun das generierte CRC Wort im Empfänger alles Nullen auf, verlief die Überprüfung positiv und die Daten können als gültig angesehen werden [7, Seite 2].

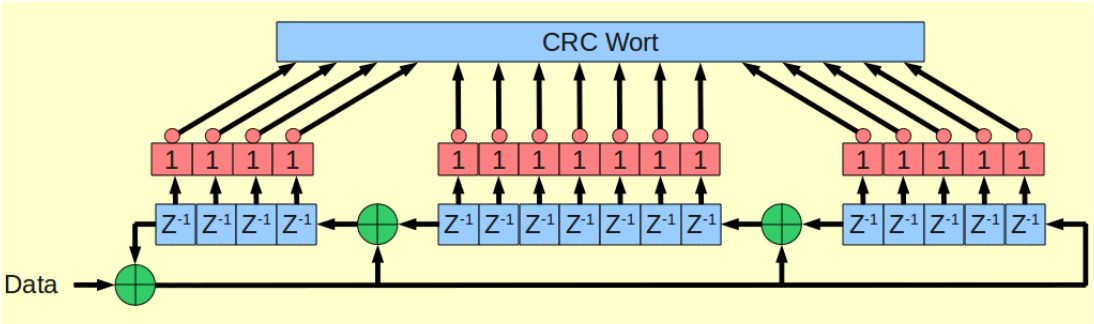


Abbildung 6.16.: Generierung des Cyclic Redundancy Check Wortes

6.9.2. Fast Information Group

Ein FIB kann in Fast Information Groups (FIGs) aufgespaltet werden (Abbildung 6.17). Jede FIG ist zwischen 2 und 30 Bytes gross (Inklusive Header) und die grobe Zuordnung der Daten wird über das Typenfeld gemacht [1, Seite 29]. Die für einen Audioempfänger wichtigen Informationen werden in den FIG Typen 0 und 1 übertragen.

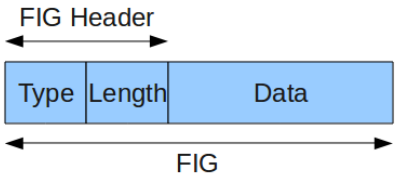


Abbildung 6.17.: FIG Struktur

Eine Type 0 Gruppe gibt Auskunft über einen Teil der Multiplex Configuration Information (MCI), welche die Struktur innerhalb der CIFs beschreibt. Ausserdem enthalten sie Informationen wie das Datum, die Coordinated Universal Time (UTC), den regionsabhängigen Offset zur UTC, das Land in dem sich der Sender befindet, die Sprachen der Services und noch vieles mehr.

Type 1 wird für die Übermittlung der Labels der einzelnen Ensembles und Services genutzt, welche für die Hörer an den Displays der Radioempfänger angezeigt werden. Pro DAB Kanal (1,536 MHz Bandbreite) kann ein Ensemble übermittelt werden, welches bis zu 64 verschiedene Services enthält.
Ein Empfänger muss bei der Inbetriebnahme die FIGs einige Zeit überwachen und mit den Daten eine Datenbank aufbauen (mögliche Struktur der Datenbank in Abbildung 6.18). Während des Betriebs muss zum rechtzeitigen Erkennen von Veränderungen der Servicesanordnung nur der FIC auf Modifikationen überwacht werden und gegebenenfalls die Datenbank mit den Daten der einzelnen Services nachgeführt werden. Auf diese Weise kann sichergestellt werden, dass jederzeit der Service ohne Verzögerung gewechselt werden kann, zudem haben Modifikationen im Multiplexing keinen Einfluss auf laufende Übertragungen.

	Label	ID	Kanal	Subchan ID	Language	Start Adr	Size	Protection	ASCTy
Ensemble	SRG SSR D01	16385	12C						
Services	.....			.....					
	DRS 3	17331		3	German	188	84	UEP 4	0
				-	-	-	-	-	-
	DRS Musikwelle	17332		4	German	272	96	UEP 3	0
				-	-	-	-	-	-
	.....			.....					
Ensemble	SMC_D02	16386	7D						
Services	.....			.....					
	Energy Zuerich+	20243		6	German	240	48	EEP 3A	63
				-	-	-	-	-	-
	.....			.....					

Abbildung 6.18.: Datenbank mit Ensemble und Service Informationen

6.10. Reed Solomon und Superframing (DAB+)

Bei DAB+ [3] werden mehrere Access Units (AU) der Audiodaten in ein Superframe verpackt (Abbildung 6.19). Abhängig von der Übertragungskapazität des Subchannels enthält das Superframe ein Vielfaches von 110 Bits. Im Header des Superframes ist ein Firecode enthalten. Er ist in der Lage bis sechs Bitfehler im Header zu erkennen und zu korrigieren (abgesehen von ein paar Spezialkonstellationen die nur detektiert und nicht korrigiert werden können). Mit den anschliessenden Flags werden die Audioeigenschaften signalisiert. Im letzten Teil des Headers befinden sich die Startadressen der AUs. Die Grösse einer AU wird über die Differenzbildung von zwei aufeinander folgenden Startadressen bestimmt. Die AUs enthalten die codierten Audiodaten, wobei jede AU mit einem zusätzlichen CRC versehen ist. Der CRC wird gleich wie für den FIC (Abschnitt 6.9.1) gebildet.

Bei DAB+ bildet der Faltungscodierer den inneren Encoder (für die Korrektur von einzelnen Bitfehlern) und ein Reed Solomon (RS) Code den äusseren Encoder (für die Korrektur von Burstfehlern).

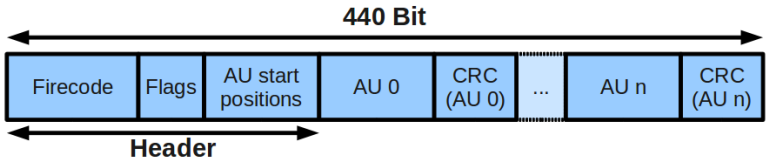


Abbildung 6.19.: Aufbau des Audio Superframes (Subchannel 32kbps)

Um den RS Code zu berechnen werden die Superframes wie in Abbildung 6.20 gezeigt nach einem sägezahnförmigen Muster in eine Matrix abgelegt [3, Seite 15]. Die Höhe der Matrix wird durch die zugeordnete Datenrate des Subchannels definiert. Für jeweils 110 horizontal angeordnete Bits aus der Matrix, werden 10 RS Bits angehängt. Mit diesem RS Code können 5 Bitfehler auf einer Zeile korrigiert werden. Durch das sägezahnförmige Auffüllen entsteht ein virtuelles Interleaving. Tritt ein Burstfehler auf, werden die Daten durch das Interleaving auf mehrere RS Codes verteilt und es können daher mehr aufeinander folgende Fehlerbits korrigiert werden. Weist beispielsweise die Matrix eine Höhe von vier Bits auf, kann ein Burstfehler von 20 Bits (4*5 Bits) korrigiert werden.
Für die Weitergabe an die nächste Einheit werden die Daten nach dem gleichen Muster ausgelesen wie sie in die Matrix hineingeschrieben wurden. Der RS Code wird ebenfalls nach diesem Muster ausgelesen und an die Daten angehängt.
Da nicht alle Bits eines Superframes im gleichen DAB Frame übermittelt werden, muss ein Empfänger entweder über den RS Code, den Firecode oder den CRC der AUs die Synchronisation suchen.

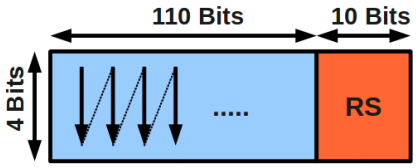


Abbildung 6.20.: Error geschütztes Packet (Subchannel 32kbps)

6.11. Audiocodec MP2 (DAB)

Bei DAB wird der MPEG-1 Audio Layer 2 (MP2) als Audiocodec verwendet. Wie aus einem DAB Audio Frame ein akustisches Signal erzeugt werden kann wird nicht im DAB Standard beschrieben sondern im ISO/IEC 13818-3 Standard (Information technology - Generic coding of moving pictures and associated audio information - Part 3: Audio). Dieser Audiostandard ist stark verbreitet und hat sich bei DAB und bei DVB etabliert. Der Standard nutzt die Schwächen des menschlichen Gehörs aus, in dem akustisch nicht wahrnehmbare Anteile im Quellsignal von der Komprimierung ausgeschlossen werden.

6.12. Audiocodec HE-AAC v2 (DAB+)

Im Standard [3, Seite 7] wird darauf vermerkt, dass bei DAB+ ein hoch effizienter MPEG 4 Code für das Audiosignal verwendet wird. Dieser High Efficiency Advanced Audio Codec (HE-AAC) v2 bietet bei einer kleinen Datenrate eine gute Audioqualität. In DAB Systemen ist er viel effizienter als MP2 und mit seiner Implementierung können etwa vier Mal mehr Sender mit besserer Audioqualität übermittelt werden als mit MP2.
Der Advanced Audio Codec (AAC) basiert in seiner Grundstruktur auf der MPEG-4 Basis. Jedoch wurde der HE-AAC v2 um zwei wesentliche Elemente erweitert. Zum einen ist dies die Spektral-Replikation (SBR) Funktion. Diese ist vor allem verantwortlich für eine gute Klangqualität bei einer kleinen Datenrate. Zum anderen wurde die Parametric Stereo (PS) Funktion implementiert. Diese kann Stereosignale sehr effizient komprimieren.
Speziell an dem in DAB+ eingesetzten Audiocodec ist die 960 Punkt-Transformation für die FFT. In allen anderen Anwendungsbereichen welche den HE-AAC v2 einsetzen, wird für Audio eine 1024 Punkt-Transformation verwendet. Dies könnte auf die geforderte Audio Superframelänge in DAB+ zurückzuführen sein.
Wie aus einem Audio Super Frame ein akustisches Signal erzeugt werden kann wird nicht im DAB Standard beschrieben sondern im ISO/IEC 14496-3 Standard (Information technology - Coding of audio - visual objects - Part 3: Audio).

7. DAB Simulation in Matlab mit realen Daten

Inhalt

7.1. Signalaufzeichnung 47
7.2. Detektion des Framestarts 47
7.3. Modulation 48
7.4. Frequency Deinterleaving und QPSK Symbol Demapping 50
7.5. Depuncturing und Viterbidecoder 51
7.6. Energy Dispersal 52
7.7. Cyclic Redundancy Check 52
7.8. FIC Auszug und Analyse 52
7.9. Reed Solomon und Superframing 54
7.10. MP2 Audiofile und Netzwerkstream 55
7.11. Audiocodec HE-AAC v2 55
7.12. Simulink DAB Empfänger 56

Die Simulation des DAB Empfängers wurde in Matlab realisiert. Als Quelle dienen mit GNU Radio Companion aufgezeichnete DAB Kanäle (in Kombination mit der USRP N210 Box). Das grundlegende Ziel der Simulation war das Erkennen der Projektgrenzen. Die Simulation besteht aus zwei Hauptfiles.

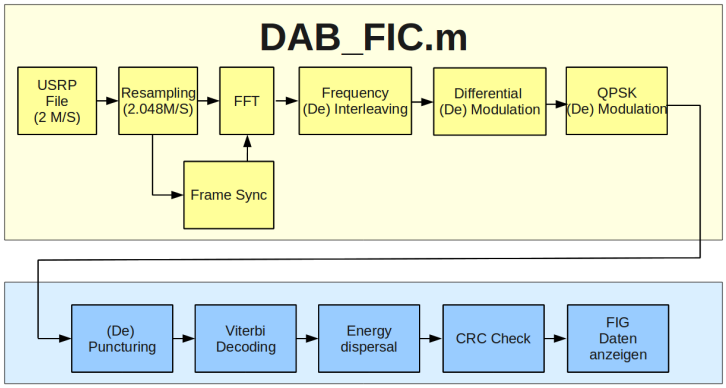


Abbildung 7.1.: Aufbau des Simulationsfiles DAB_FIC.m

Das eine Hauptfile (DAB_FIC.m) wird zum Decodieren des FICs gebraucht (Struktur des Hauptfiles ist in Abbildung 7.1 gezeigt). Es kann mit Daten aller DAB Modi arbeiten und schreibt als Output die Daten der FIGs auf das Matlab Command Window. Dieses File wurde in erster Linie nicht ausgelegt um in Echtzeit (mit hoher Geschwindigkeit) die Daten verarbeiten zu können. Im Mittelpunkt stand vielmehr dass alle Funktionen als Vorlage für eine spätere Implementierung in C++ dienen können. Deshalb arbeitet dieses Hauptfile unter anderem mit einem von Hand ausprogrammierten Viterbidecoder. Ein weiterer Vorteil dieser Implementation ist, dass nur Standardbibliotheken von der Matlab Basisversion ohne zusätzlich installierte Mathworks Toolboxen benötigt werden und dieses File auf jedem Computer mit installiertem Matlab läuft.

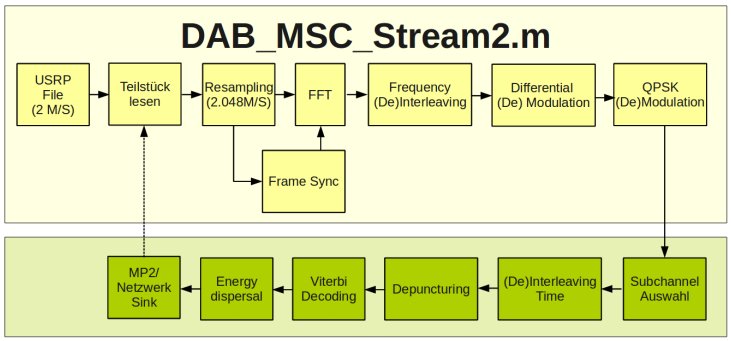


Abbildung 7.2.: Aufbau des Simulationsfiles DAB_MSC_Stream2.m

Das zweite Hauptfile (DAB_MSC_Stream2.m) wird zum herstellen eines Audiofiles (MP2 File) oder eines Audio Netzwerkstreams zum wiedergeben eines Radiosenders gebraucht (Struktur des Hauptfiles ist in Abbildung 7.2 gezeigt). Bei diesem File stand die Geschwindigkeit im Vordergrund, deshalb wurden wenn möglich von Matlab zur Verfügung gestellte geschwindigkeitsoptimierte Funktionen verwendet. Das File wurde so aufgebaut, dass es mit einem Stream umgehen kann. Nacheinander wird ein kleines Stück des abgespeicherten Files gelesen und verarbeitet. Das Hauptfile sollte ebenfalls als Grundlage eines DAB Simulink Empfängers zu gebrauchen sein.
Der generierte Netzwerkstream kann in einem beliebigen Programm wie beispielsweise dem VideoLAN Client (VLC) wiedergegeben werden.

7.1. Signalaufzeichnung

Über eine Zusammenschaltung von zwei Blöcken in GNU Radio Companion kann ein Frequenzbereich aufgezeichnet werden. Wie im Abschnitt 6.2 gezeigt, strebt man eine Bandbreite von 2.048 MHz an. Da die USRP Box aber nur ganzzahlige Teiler vom intern verwendeten 100 MHz USRP Clock als Samplingfrequenz verwenden kann, wird ein Signal mit einer Bandbreite von 2 MHz aufgezeichnet. Dieses kommt am nächsten an die gewünschten 2.048 MHz heran. Die Daten werden in ein File mit DAT Suffix abgespeichert (Abbildung 7.3). Zehn Sekunden Daten ergeben etwa 150 MB Speicherverbrauch. Damit die USRP N210 Box auf den gewünschten DAB Frequenzbereich bei 200 MHz zugreifen kann, wurde das Daughterboard WBX aufgesetzt.
Um das Datenfile mit reellen oder komplexen Samples im Matlab einlesen zu können, hat die California State University Northridge ein Matlabfile auf ihrer Homepage veröffentlicht [E]. Nach dem Einlesen wird ein Resampler angewendet, welcher aus der 2 MHz Rate der Aufzeichnung die gewünschten 2.048 MHz erzeugt.
In der Deutschschweiz wurden während der Dauer dieser Arbeit nur Mode 1 DAB Signale übermittelt, womit automatisch jede Aufzeichnung den Mode 1 einhält. Deshalb beziehen sich die folgenden Bilder und Auswertungen ausschliesslich auf den Mode 1.

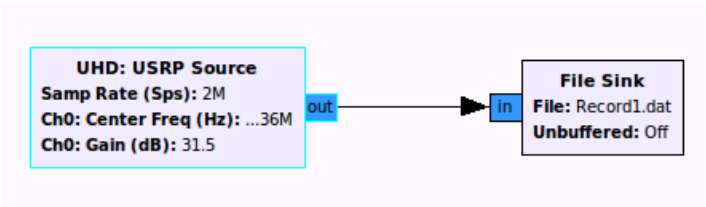


Abbildung 7.3.: Aufzeichnung der Sequenzen für die Simulation

7.2. Detektion des Framestarts

Die Abbildung 7.4 zeigt den RSSI Wert des Inputsignals, gebildet über ein Moving Average Filter. Überschreiten die Inputsamples den RSSI Wert, wird der Beginn des neuen Symbols detektiert. In der Abbildung ist dieser Moment mit einem roten Punkt markiert.

Damit nicht wegen Störungen und einmaligen Ausreissern eine Detektion stattfindet, wurde der Vergessensfaktor des Filters auf 2^3 gesetzt. Es kristallisierte sich heraus, dass während mehreren Null-Symbolen ein wenig Leistung übertragen wird. Deshalb kann der Vergessenfaktor nicht grösser gewählt werden, sonst könnte das Ende eines solchen Null-Symbols wegen zu grossem RSSI nicht detektiert werden.
Damit das Null-Symbol Ende und nicht der Start detektiert wird, wurde der berechnete RSSI Wert vor der Detektionsabfrage um 2^6 Samples verzögert. Somit lässt sich das Ende zuverlässig erkennen. Damit keine Mehrfachdetektion stattfindet, wird nach einer Detektion eine Totzeit von mindestens 2^{13} Samples zugeschaltet bis der Detektor wieder aktiviert wird. Diese Totzeit kann bis auf zirka 194500 Samples (beinahe ein ganzes Frame) ausgeweitet werden, damit die Detektion erst wieder im Null-Symbol des Folgeframes aktiv ist.

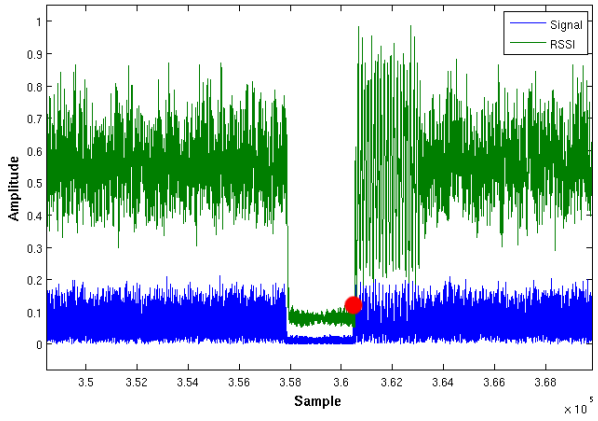


Abbildung 7.4.: Null Symbol und RSSI einer DAB Aufzeichnung, der rote Punkt zeigt die Detektion des Endes des Null-Symbols an

Um die Funktionstüchtigkeit des Resamplers zu testen wurde in einer zehn Sekunden Aufzeichnung überprüft wie weit sich die Framestarts auseinander befinden (Abbildung 7.5). Die Abbildung zeigt, dass sich die Starts nie weiter als sechs Samples vom Optimum entfernt befinden. Wäre diese Abweichung zu gross, würde man Gefahr laufen, dass die FFT zwischen zwei Symbole gerät und somit ein unbrauchbares Resultat liefert. Da der Guard Intervall aber 504 Samples beträgt, ist man sehr weit von diesem Problem entfernt.

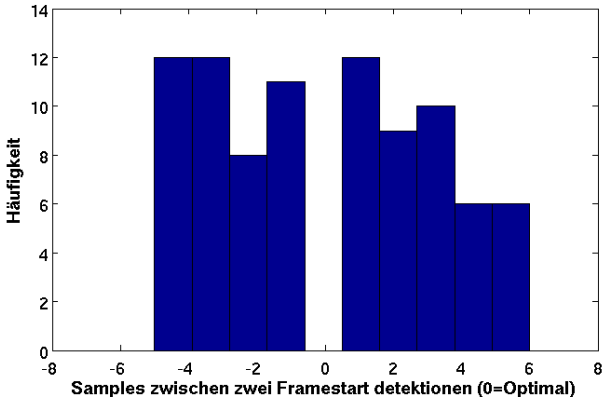


Abbildung 7.5.: Aufzeichnung der Dauer zwischen den Framestarts

7.3. Modulation

Um zu überprüfen ob die Frequenz der USRP Box genügend genau an die Zenterfrequenz des DAB Kanals herankommt und zusätzlich keinen grossen Jitter aufweist, musste das durch die komplexe FFT demodulierte Signal analysiert werden.
Bei einer differentiellen Übertragung mit einer QPSK Modulation dürfen aufeinander folgende Symbole eines Subcarriers maximal eine Phasenabweichung von $\pm(\pi/4)$ aufweisen um den Eindeutigkeitsbereich nicht zu verlassen.

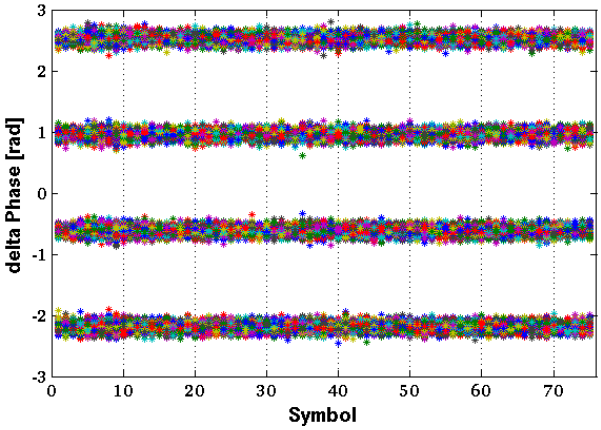


Abbildung 7.6.: Phasensprünge zwischen den OFDM Symbolen für jeden Subcarrier eines DAB Frames

Das Ergebnis der differentiellen Demodulation zeigt die Abbildung 7.6. Optimalerweise wären die Punktanordnungen bei $\pm(\pi/4)$ oder $\pm(\pi * 3/4)$, in der Abbildung beobachtet man eine kleine Abweichung. Da die Punkte keine Steigung aufweisen und nur einen Offset zeigen, kann von einem Frequenzoffset zwischen der Zenterfrequenz des DAB Kanals und der Frequenz der USRP Box gesprochen werden. Über die Gleichung 7.1 wurde ein Offset von 27.45 Hz berechnet. Aus dem Abschnitt 6.4 weiss man, dass für das Demapping der Real- und der Imaginäranteil der Modulationspunkte entscheidend sind. Deshalb kann bei einem Frequenzoffset immer von einer Verschlechterung des SNR ausgegangen werden, da einer der beiden Anteile kleiner wird. Theoretisch könnte dieser Phasenoffset mit einem nachgeschalteten Coordinate Rotation Digital Computer (CORDIC) korrigiert werden. Da die Phase mit dem USRP Empfänger nur gering abweicht und zu jedem Zeitpunkt eine Demodulation möglich war, verzichtet man auf den CORDIC.

$$\frac{\Delta\varphi}{T_s * 2 * \pi} = \Delta f = \frac{1 - \pi/4}{1.246s * 2 * \pi} = 27.45Hz \tag{7.1}$$

$\Delta\varphi$ = Phase zwischen optimalen und demodulierten Subcarrier Symbol
T_s = Symboldauer
Δf = Frequenzoffset

Bestimmen des Frequenzoffsets zwischen der Frequenz der USRP Box und der Zenterfrequenz des DAB Kanals (Mode 1)

Die Dicke der vier Punktestreifen in der Abbildung 7.6 widerspiegelt den Jitter der beiden Clocks. Da die Streifen nicht ineinander verlaufen und den Eindeutigkeitsbereich nicht verlassen, kann mit diesem System zuverlässig gearbeitet werden und es kann mit wenig Bitfehlern gerechnet werden.

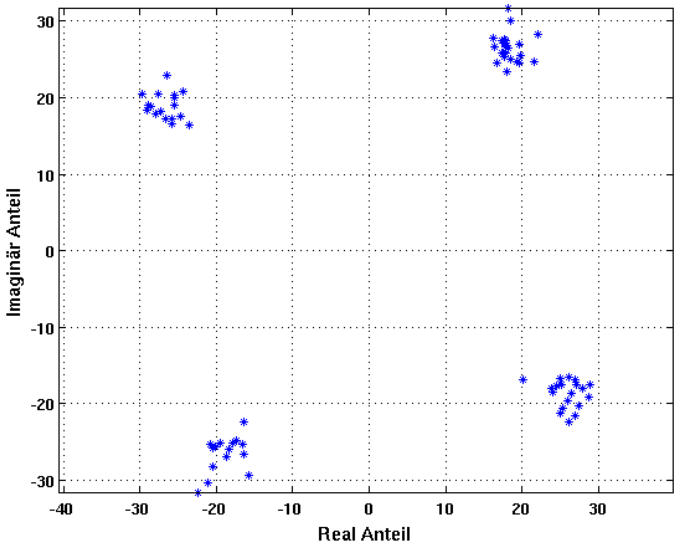


Abbildung 7.7.: QAM Modulation auf einem OFDM Subcarrier

Eine andere Möglichkeit die Zuverlässigkeit der Übertragung zu testen, kann mit Hilfe der Abbildung 7.7 bewerkstelligt werden. Je grösser die Frequenzabweichung zwischen Empfänger und Sender ist, desto mehr drehen sich die Modulationspunkte um den Nullpunkt. Solange jedoch die Modulationspunkte eines Subcarriers in ihrem Quadranten bleiben, befindet sich das Symbol im Eindeutigkeitsbereich und es entstehen keine Fehlübertragungen. Wie die Abbildung zeigt, ist die Eindeutigkeit bei diesem System gegeben.

7.4. Frequency Deinterleaving und QPSK Symbol Demapping

Damit das Frequency Interleaving rückgängig gemacht werden kann, wird beim Programmstart die Tabelle des Standards [1, Seite 157-161] (Tabellenwahl wird durch Übertragungsmodi bestimmt) gebildet. Die Regel zum Erstellen der Tabellen ist für jeden Modus gleich, daher kann immer der gleiche Programmcode mit unterschiedlichen Parametern gebraucht werden.
Für das eigentliche Deinterleaving werden die Spalten n und k aus der Tabelle benötigt. k zeigt auf eine Position des Ausgangsvektors der komplexen FFT und n zeigt auf die deinterleavte Position in einem neuen Vektor. Wird dies auf den ganzen Ausgangsvektor angewendet, ist das Deinterleaving vollzogen. Am Ausgang des Deinterleavers erhält man ein komplexes Symbol für jeden Subcarrier.
Wie in Abschnitt 6.4 beschrieben, bilden immer zwei Bit zusammen ein QPSK Symbol. Bei einem DAB Empfänger wird in einem ersten Schritt von allen Symbolen am Ausgang des Deinterleavers der Realanteil genommen und in einen Vektor geschrieben. Im darauf folgenden Schritt wird von den deinterleavten Symbolen der Imaginäranteil genommen, in einen Realwert umgewandelt (Beispiel $5.37j = 5.37$) und an den Vektor angehängt (Abbildung 7.8). Der neu gebildete Vektor hat die Länge von der doppelten Subcarrieranzahl

(2*K), was dem in der Schweiz eingesetzten Übertragungsmode 3072 Bits entspricht.

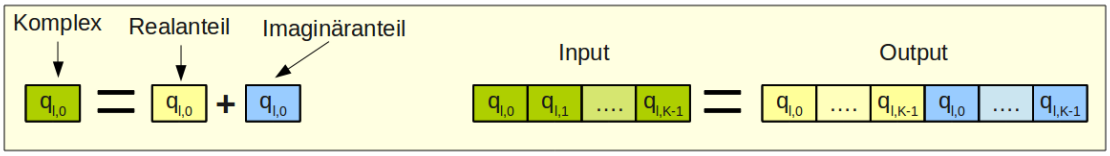


Abbildung 7.8.: Demapping der QPSK Symbole

7.5. Depuncturing und Viterbidecoder

Wie in Abschnitt 6.7 beschrieben, muss in einem Empfänger vor dem Durchlaufen des Viterbidecoders ein Depuncturing angewendet werden. Für dieses Vorhaben wurde eine Matlabfunktion generiert, welcher ein Datenstück und der zugehörige PI übergeben wird. Aus dem übergebenen PI wird als erstes der Puncturing Vector ermittelt. Dies geschieht über die Tabelle [1, Seite 131] des Standards, darin sind alle möglichen Puncturing Vectoren aufgelistet. Nach diesem Schritt wird der Vektor mit den depunktierten Daten generiert. Die Abbildung 7.9 verdeutlicht das Vorgehen. Jedes Inputbit nimmt den Platz einer Eins des Puncturing Vectors ein. Um dies zu bewerkstelligen wird der Puncturing Vector wiederholt, bis für jedes Inputbit ein Platz gefunden wurde. Der generierte Vektor stellt nun den Input des Viterbidecoders dar.

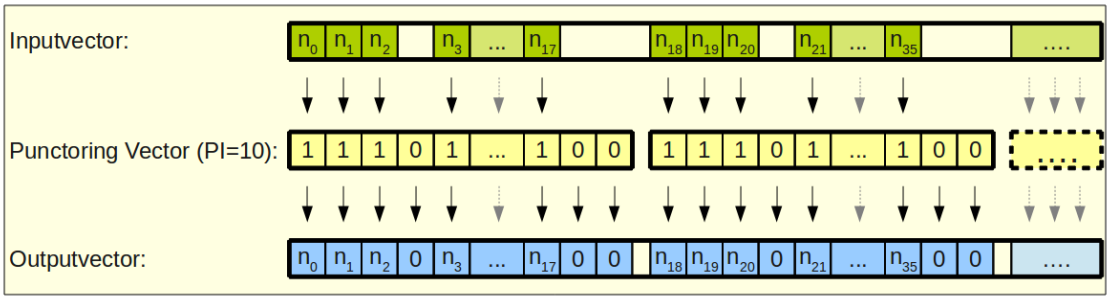


Abbildung 7.9.: Funktionsweise des Depuncturings

Der selbstgeschriebene Viterbidecoder für den FIC arbeitet mit jedem beliebigen Generatorpolynom und kann somit in weiteren Projekten eingesetzt werden. Um eine zuverlässige Bitentscheidung zu treffen wurde die Speichertiefe für einen Symbolentscheid auf die fünffache Generatorpolynombreite festgelegt. Da die aufgezeichneten Files für die Matlabsimulation (Abschnitt 7.1) eine sehr gute Signalqualität aufweisen und kein einziger Biterror detektiert wurde, könnte die Speichertiefe für die Bitentscheidung theoretisch auf ein Minimum (Tiefe 1) gekürzt werden. Da der Empfänger aber mit grosser

Wahrscheinlichkeit auch an anderen Standorten zum Einsatz kommt, lohnt sich die Wahl einer grösseren Tiefe.
Für den MSC wurde der schnelle Viterbidecoder aus der Communication System Toolbox von Matlab eingesetzt. In Abhängigkeit von der Speichertiefe müssen noch Nullen (Speichertiefe * 4) an den Datenstrom angehängt werden, damit auch die letzten Datenbits bestimmt und ausgegeben werden. Die Matlabfunktion gibt zudem vor dem gewünschten Outputvektor Nullen (Speichertiefe * 1) aus. Diese werden anschliessend abgeschnitten.

7.6. Energy Dispersal

Um das Energy Dispersal effizient zu implementieren und keine Rechenleistung bei jedem Durchlauf zu verschwenden, werden beim Programmstart einmalig die ersten 100000 Bits der PRBS berechnet. Bei der Anwendung der Sequenz auf einen Datenblock, wird immer mit dem ersten Bit der Sequenz begonnen. Mit dieser Implementation können dadurch beliebig viele Datenblöcke (bis zu einer Blocklänge von 100000 Bits) mit der generierten Sequenz XOR verknüpft werden und man erhält die gewünschten Ursprungsdaten.
Die PRBS wurde in Matlab über ein rückgekoppeltes Schieberegister mit mehreren XOR Verknüpfungen generiert, als Vorlage diente die Abbildung 6.15 des vorangehenden Kapitels. Die Funktionsüberprüfung fand mit Hilfe des Standards [1, Seite 128] statt, dort sind die ersten 16 Bits der Sequenz abgebildet.

7.7. Cyclic Redundancy Check

Diese Funktion wurde für Datenblöcke bestehend aus einer beliebigen Anzahl Bits entwickelt. Ihr werden die Daten inklusive angehängtem CRC übergeben. Als erster Schritt wird das Zweierkomplement des angehängten CRCs gebildet, damit die Checksumme des Datenblocks (inklusive Zweierkomplement des CRC) auf eine Bitfolge mit lauter Nullen überprüft werden kann. Sollte ein Datenblock eine ungültige Checksumme aufweisen, wird eine Fehlermeldung ausgegeben.
Die CRC Funktion wurde ausgiebig getestet, da die Aufzeichnungen für die Matlabsimulation aber eine sehr gute Qualität aufweisen, konnte kein Bitfehler im Originalstream detektiert werden. Von Hand vorgenommene Manipulationen des Datenstroms wurden einwandfrei detektiert.

7.8. FIC Auszug und Analyse

Der endgültige Beweis einwandfreie Daten zu empfangen liefert die Datenanalyse des FICs. In einem ersten Schritt werden die FIGs aus den FIBs herausgelöst. In einem FIB sind die FIGs aneinander gereiht und die Längeninformation im FIG Headers gibt Auskunft aus wie vielen Bits das FIB besteht. Mit dieser Information kann der Offset bis zum nachfolgenden FIG berechnet werden. Enthält ein FIB Header alles Einer, wurde der End Marker gefunden und der Rest des FIBs enthält keine Informationen mehr.
In einem zweiten Schritt werden die Daten der FIGs in einer Funktion aufbereitet und im Matlab Command Window angezeigt. Da die DAB Aufzeichnungen aus dem Raum Zürich

nur Audioservices und keine Datendienste enthalten, wird im Folgenden nur auf Audiorelevante FIG Typen eingegangen.
Die Tabelle 7.1 zeigt einen Auszug des FICs des DAB Ensembles bei 227.36 MHz im Raum Zürich. Die wichtigsten Informationen wurden mit Farbe hervorgehoben.
Die FIGs des Type 1 / Extension 0 und 1 zeigen die Labels des Ensembles und der Services. Aus dem Auszug des FIC erkennt man, dass das Ensemble das Label “SRG SSR D01“ trägt und Services wie “DRS 1“ oder “CH-POP“ enthält. Ergänzend geben die FIGs des Type 0 / Extension 17 noch die Sprache der Services an.
Die FIGs des Type 0 / Extension 2 zeigen dem Empfänger welcher Service mit welchen Subchannel zusammenarbeitet. Ein Service kann mit zwei Subchannels (einem Primären und einem Sekundären) arbeiten. Welcher Subchannel primär oder sekundär ist, wird über die Flags signalisiert. In diesem FIG Type wird ebenfalls mitgeteilt welcher Audio Service Component Type (ASCTy) zur Decodierung des Audiostreams angewendet werden muss. ASCTy 0 steht beispielsweise für “MP2“ und die Daten im MSC enthalten Vordergrundklang. ASCTy 63 steht für “HE-AAC v2“ und somit für ein DAB+ Service.
Die FIGs des Type 0 / Extension 1 weisen einem Subchannel einen Bereich im CIF zu. Zudem zeigen sie welcher Errorschutz der Subchannel aufweist und wie gross die Bitrate ist.
Neben all diesen Informationen wird mit dem FIGs des Type 0 / Extension 9 und 10 die aktuelle Zeit übermittelt.
Mit Hilfe all dieser Informationen kann die Funktionstüchtigkeit des geschriebenen Matlab Empfängers im Zusammenspiel mit der Ettus Empfängerhardware aufgezeigt werden. Ein Versucht hat gezeigt, dass die Subchannelanordnung des FICs auf dem Ensemble “SRG SSR D01“ selten ändert. Über drei Monate hinweg konnte keine Veränderung nachgewiesen werden.

Type: 0 Extension: 0 C/N: 0 OE: 0 P/D: 0 EId: 16385 ChangeFlag: 0 ALFlag: 0 CIFCount: 75 OccurenceChange: 176
Type: 0 Extension: 2 C/N: 0 OE: 0 P/D: 0 SId: 17385 MSC stream audio ASCTy: 63 SubChId: 10 P/S: 1 CAFlag: 0
SId: 17332 MSC stream audio ASCTy: 0 SubChId: 4 P/S: 1 CAFlag: 0
SId: 17373 MSC stream audio ASCTy: 63 SubChId: 13 P/S: 1 CAFlag: 0
SId: 17329 MSC stream audio ASCTy: 0 SubChId: 1 P/S: 1 CAFlag: 0
SId: 17334 MSC stream audio ASCTy: 0 SubChId: 12 P/S: 1 CAFlag: 0
Type: 1 Extension: 1 OE: 0 Charset: 0 SId: 17329 Service Label: DRS 1
Type: 1 Extension: 1 OE: 0 Charset: 0 SId: 17137 Service Label: CH-POP
Type: 0 Extension: 10 C/N: 0 OE: 0 P/D: 0 Date: 2011/03/28 Time: 12:53 UTC
Type: 0 Extension: 1 C/N: 0 OE: 0 P/D: 0 SubChID: 2 StartAdr: 84 UEP ProtLevel: 4 SubChSize: 104 BitRate: 160
SubChID: 8 StartAdr: 632 UEP ProtLevel: 4 SubChSize: 70 BitRate: 112
SubChID: 7 StartAdr: 548 UEP ProtLevel: 4 SubChSize: 84 BitRate: 128
SubChID: 9 StartAdr: 702 EEP ProtLevel: 3-A SubChSize: 24
SubChID: 12 StartAdr: 785 UEP ProtLevel: 3 SubChSize: 42 BitRate: 56
SubChID: 1 StartAdr: 0 UEP ProtLevel: 4 SubChSize: 84 BitRate: 128
Type: 0 Extension: 17 C/N: 0 OE: 0 P/D: 0 SId: 17369 S/D: 0 P/S: 0 CCFlag: 0 IntCode: 3 French
SId: 17332 S/D: 0 P/S: 0 CCFlag: 0 IntCode: 26 German
SId: 17137 S/D: 0 P/S: 0 CCFlag: 0 IntCode: 10 German
SId: 17138 S/D: 0 P/S: 0 CCFlag: 0 IntCode: 13 German
Type: 1 Extension: 0 OE: 0 Charset: 0 EId: 16385 Ensemble Label: SRG SSR D01
Type: 0 Extension: 9 C/N: 0 OE: 0 P/D: 0 LTO: +2.0h ECC: 225 InterTabId: 1
Type: 0 Extension: 5 C/N: 0 OE: 0 P/D: 0 MSC Stream SubChId: 10 Italian
MSC Stream SubChId: 6 German
MSC Stream SubChId: 4 German
MSC Stream SubChId: 11 Romansh
MSC Stream SubChId: 8 German
MSC Stream SubChId: 13 English

Tabelle 7.1.: FIC Auszug

7.9. Reed Solomon und Superframing

Bedingt durch die geringe Bit Error Rate (BER) der aufgezeichneten Files für die Matlabsimulation, entschied man sich nach einigen Schwierigkeiten den RS Code vorerst nicht zu implementieren und die Datenbytes unkorrigiert weiter zu geben. Zu einem späteren Zeitpunkt kann dieser Teil mit geringem Aufwand implementiert werden.
Da ein Superframe aus den Daten von mehreren CIFs besteht, muss beim Start eines Empfängers die Synchronisation gesucht werden. Um dies mit einem sehr kleinen Aufwand zu bewerkstelligen, suchte man die Synchronisation über die AU CRCs (Theoretisch könnte die Synchronisation auch über den RS Code oder über den Firecode des Superframes gesucht werden). Zur CRC Prüfung konnte die Einheit aus Abschnitt 7.7 wieder verwendet werden. Dies brachte den Vorteil, dass die Funktionstüchtigkeit der CRC Einheit bereits im FIC nachgewiesen werden konnte.

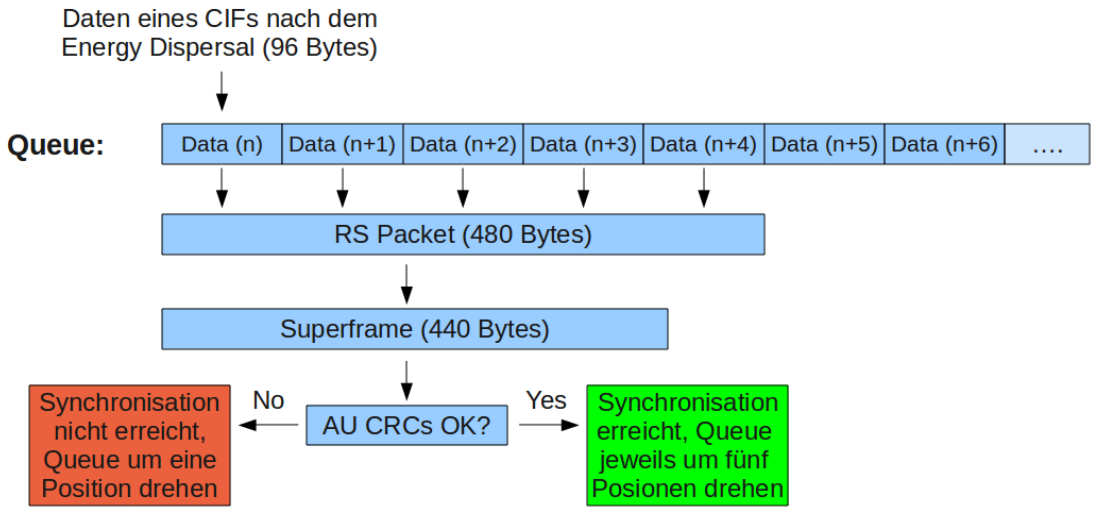


Abbildung 7.10.: Blockhandling bei DAB+ und einem 32kbps Subchannel

Die Abbildung 7.10 zeigt, wie die Synchronisationssuche abläuft. Zu Beginn der Synchronisation werden mehrere Blöcke (Datenratenabhängig) nach der Energy Dispersal Einheit aneinander gereiht. Anschliessend werden von den Daten die vermuteten RS Code Bytes abgeschnitten. Übrig bleiben die Datenbytes des Superframes. Über den Header des Superframes können nun die einzelnen AUs herausgelöst werden. Der letzte und entscheidende Schritt ist das Überprüfen der CRCs der einzelnen AUs. Stimmen die CRCs, wurde die Synchronisation gefunden und es können in Zukunft die richtigen Blöcke zu einem Superframe zusammengehängt werden. Versagt der CRC Check, muss um ein Datenblock geschoben, das Superframe erneut herausgebildet und die Prüfung wiederholt werden. Dies geschieht solange bis eine Synchronisation nachgewiesen werden kann. Das Produkt dieser Einheit sind die AUs. Sie müssen an einen Audiocodec weitergegeben werden.
Beim in der Schweiz ausgesendeten Mode 1 befinden sich vier CIF Datenblöcke in einem DAB Frame. Bei einer Audiodatenrate von 32 kbps werden fünf Datenblöcke für ein

Superframe benötigt. Somit kann nach zwei DAB Frames die Synchronisationssuche das erste Mal gestartet werden. Damit die Suche einfach vonstatten geht, wurde eine Datenblock Queue eingerichtet. Über diese Queue sieht der Synchronisator ob genügend Daten für einen Synchronisationsversuch vorhanden sind, andernfalls wird auf mehr Datenblöcke gewartet.

7.10. MP2 Audiofile und Netzwerkstream

Bei einem DAB Stream werden nach dem Energy Dispersal Block jeweils acht Bits zu einem Byte zusammengefasst und in ein File mit MP2 Suffix abgespeichert. Das erstellte MP2 File kann mit den meisten aktuellen Audioprogrammen wie beispielsweise dem VLC Player wiedergegeben werden. Das Abspeichern der Daten wurde mit einem Hexeditor überprüft. Eine andere implementierte Möglichkeit die Daten wiederzugeben bietet ein Netzwerkstream. Wird die Variable "lan" in der Matlabumgebung auf 1 gesetzt, werden die Audiodaten auf dem Port 9999 für einen anderen Computer bereitgestellt. Der fremde Computer kann nun über die Adresse "tcp://IP:9999" auf die Audiodaten zugreifen und sie abspielen. Beim Netzwerkstream ist zu beachten, dass der Computer mit der Matlabsimulation eine genügend grosse Rechenleistung für eine Live-Signalverarbeitung besitzt. Andernfalls wird auf dem fremden Computer der Ton mit vielen Unterbrüchen abgespielt. In dieser Projektarbeit arbeitete man mit dem VLC Player um den Netzwerkstream wiederzugeben.

7.11. Audiocodec HE-AAC v2

Bei einem DAB+ Stream werden acht Bits aus den AUs zu einem Byte zusammengefasst und in ein File mit MP4 Suffix gespeichert oder über ein Netzwerkstream anderen Computern bereitgestellt.
Da der Audiocodec für DAB+ mit der 960 Punkt Transformation sehr speziell ist und ausserhalb von DAB Radio keine mir bekannte Implementation mit gleichen Parametern stattgefunden hat, konnte kein Codec zum Abspielen von DAB+ gefunden werden.
Auf der Internetseite [J] wird versprochen, dass der DAB+ Standard unterstützt wird, wie sich aber im Nachhinein herausstellte ist dies nicht der Fall. Das Programm erkannte den DAB+ Datenstrom nicht.
Mit dem VideoLAN Client (VLC) Media Player konnte erreicht werden, dass der Player den AAC Datenstrom erkannte. Da er aber mit einer 1024 Punkt Transformation arbeitet, konnte er nach dem Erkennen des Framestarts die Synchronisation nicht aufrecht halten und eine Decodierung war unmöglich.
Eine mögliche Hardwarelösung wird von Analog Devices angeboten. Sie verkaufen den Audiodecoder für ihre Blackfin Processoren [K]. Da in dieser Projektarbeit eine Softwarelösung ohne zusätzlichen Prozessor gesucht wird, wurde dieser Ansatz verworfen. Der Komponentenhersteller für DAB Radios Frontier Silicon meinte, man solle am besten direkt mit dem Frauenhofer Institut Kontakt aufnehmen, da sie den Audiostandard entwickelt haben und somit alle Rechte besitzen. Nach mehreren E-Mails zeigte sich, dass das Frauenhofer Institut an dieser Arbeit Interesse bekundet, jedoch stehen zur Zeit der Masterthesis keine Ressourcen zum Unterzeichnen der Verträge und zum sicheren Ausliefern des Decoders zur Verfügung. Nach diesem Entscheid wurde die Decodierung von DAB+ zum erzeugen eines akustischen Signals in dieser Masterthesis beendet und nicht weiter verfolgt.

7.12. Simulink DAB Empfänger

Da diese Masterthesis auch als Grundlage für ein DAB-Praktikum in der Nachrichtentechnik dienen soll, versuchte man die geschriebenen Matlabfiles in Simulink zu übernehmen. Seit geraumer Zeit stellt Mathwork einen USRP Treiber für die Simulink Umgebung zur Verfügung (Communications System Toolbox [P]), damit Daten direkt von der Box übernommen und verarbeitet werden können.
Wie sich aber herausstellte funktioniert der aktuelle Treiber nur mit der USRP2 Box [Q]. Die für diese Arbeit gekaufte neuste Box (USRP N210) kann nur über den UHD Treiber angesprochen werden. Da es für diese Box aber noch keinen Simulink Treiber gibt, musste dieses Vorhaben abgebrochen werden.

8. Implementation

Inhalt

8.1. Erste Implementierung des DAB Empfängers 58
8.1.1. UHD Source und Resampling 58
8.1.2. Framedetektion und Symbolerkennung 60
8.1.3. FFT, differentielle Demodulation und Frequency Interleaving 61
8.1.4. FIC und Subchannel aus den DAB Symbolen herauslesen 62
8.1.5. Time Interleaving 63
8.1.6. Depuncturing 64
8.1.7. Viterbidecoder 64
8.1.8. Energy Dispersal 65
8.1.9. Cyclic Redundancy Check 66
8.1.10. FIB Sink 66
8.1.11. Prearrangement und TCP Sink 66
8.2. Zweite Implementierung des DAB Empfängers 66
8.2.1. Resampling 68
8.2.2. Framestart Detecter und OFDM Symbol Cutter 68
8.2.3. Null Symbol Resampler, Bonder und FIB Cutter 69
8.2.4. FIB Sink 2 und FIB Sink 3 69

Die Implementation konnte strukturmässig in sehr vielen Punkten von der Matlabsimulation übernommen werden. Der grosse Unterschied ist, dass bei GNU Radio viele Parameter und Vektorlängen bei der Objekterzeugung angegeben werden müssen. In Matlab konnte man zu einem beliebigen Zeitpunkt nach dem Programmstart die Blockgrössen wählen (Beispielsweise bei der Senderwahl).
Für die einzelnen Verarbeitungsblöcke wurden Testbenches zur Funktionsüberprüfung generiert, damit die Grundfunktionen in einem ersten Schritt getestet werden konnten. Dabei dienten Sequenzen der DAB Matlabsimulation als Inputdaten. In einem zweiten Schritt verwendete man die Aufzeichnungen eines DAB Kanals (.dat Files) welche schon für die Matlabsimulation verwendet wurden. Als letzter Schritt wurden Livedaten von der UHD Hardware mit angeschlossener Antenne in das System eingespiesen.
Als Entwicklungsbasis diente der Tarball "gr-howto-write-a-block". Er stellt die Grundstruktur zum Entwickeln neuer Blöcke bereit, damit man nur wenige Parameter selber einstellen muss und sich nachher auf die Funktion der Signalverarbeitungsblöcke konzentrieren kann. Eine Eigenentwicklung dieses Tarballs von Grund auf würde sehr grossen Aufwand bedeuten, da man sich hauptsächlich mit SWIG (Einbindung des C++ Codes in eine Python Umgebung) herumschlagen müsste.

Die Anleitung zum schreiben eines Signalverarbeitungsblocks ist auf der GNU Radio Homepage [S] zu finden. An dieser Stelle sollte erwähnt werden, dass der Zeitaufwand eigene Signalverarbeitungsblöcke zu schreiben am Anfang sehr gross ist, da implementationsabhängig Unterschiedliches beachtet und erweitert werden muss. Mit zunehmender Routine nimmt der Aufwand aber schnell ab und die GNU Radio Umgebung entwickelt sich zu einem mächtigen und effizienten Entwicklungssystem.
Damit die Blöcke später von anderen GNU Radio Nutzer eingesetzt werden können, hielt man sich beim Erstellen von neuen Signalverarbeitungsblöcken an die Namenskonvention, die von GNU Radio vorgeschlagen wurde. Diese fordert, dass das Suffix des Blocknamens den Type des Inputs (zweitletzter Buchstabe) und des Outputs (letzter Buchstabe) widerspiegelt. "c" steht für ein komplexes Signal, "f" für ein floating-point Signal und "b" für ein Byte grosses Signal. Wenn ein Block mehrere Werte gleichzeitig verarbeitet, wird ein "v" für Vektor am Anfang des Suffix geschrieben. Ist ein Block beispielsweise mit der Endung "vcf" beschriftet, fordert der Block im Betrieb einen Vektor von komplexen Werten am Eingang. Am Ausgang werden floating-point Werte den Block verlassen.

8.1. Erste Implementierung des DAB Empfängers

Die Abbildung 8.1 zeigt die Struktur der ersten Implementation des GNU Radio DAB Empfängers. Die blau gezeichneten Blöcke konnten aus den von GNU Radio zur Verfügung gestellten Bibliotheken genutzt werden und mussten nur über die Parameter richtig eingestellt werden. Die grün gezeichneten Blöcke wurden selbst geschrieben.
In den folgenden Abschnitten wird auf jeden Block einzeln eingegangen.

8.1.1. UHD Source und Resampling

Wie schon aus dem Blocknamen zu erkennen ist, wurde der UHD Treiber verwendet. Da sich im Ensemble "SRG SSR D01" bei 2.2736 MHz DAB und DAB+ Sender befinden, konnten alle möglichen Szenarien auf einer Frequenz getestet werden. Deshalb arbeitete man bei der Implementation ausschliesslich auf dieser Zenterfrequenz. Bei der eingesetzten WBX Hardware stellte man den digital einstellbaren Verstärker auf 31.5 dB Gain und die Samplingrate auf 2 MHz. Dies sind die gleichen Werte wie bei den Aufzeichnungen für die Matlabsimulation. Mit der Verstärkung von 31.5 dB traten sehr selten Bitfehler auf. Die UHD Netzwerkadresse liess man unverändert bei "192.168.10.2". Die Tabelle D.1 im Anhang der Dokumentation zeigt alle eingesetzten Parameter der UHD Quelle.
Bei einigen Testläufen mit der UHD Quelle zeigten sich Schwankungen der Signalstärke von 2-4 dB. Die Quelle dieser Inkonsistenz konnte nicht genau eruiert werden, theoretisch können sie von der Umwelt stammen oder durch die Verstärker im Empfänger verursacht werden. Möchte man zu einem späteren Zeitpunkt die Empfindlichkeit des Empfängers genau ermitteln, sollte die Verstärkung des Empfängers auf gleichem Niveau gehalten werden und mit extern zugeschalteten Attenuatoren die gewünschten Pegeländerungen vollzogen werden. Als Signalquelle sollte für die Messung ein externer Arbitrary Function Generator (AFG) dienen.
Beim Resampling möchte man von einer Samplingfrequenz von 2000 kHz auf 2048 kHz kommen. Eine Zerlegung der Angestrebten- und der UHD-Samplingfrequenz in Primzahlen

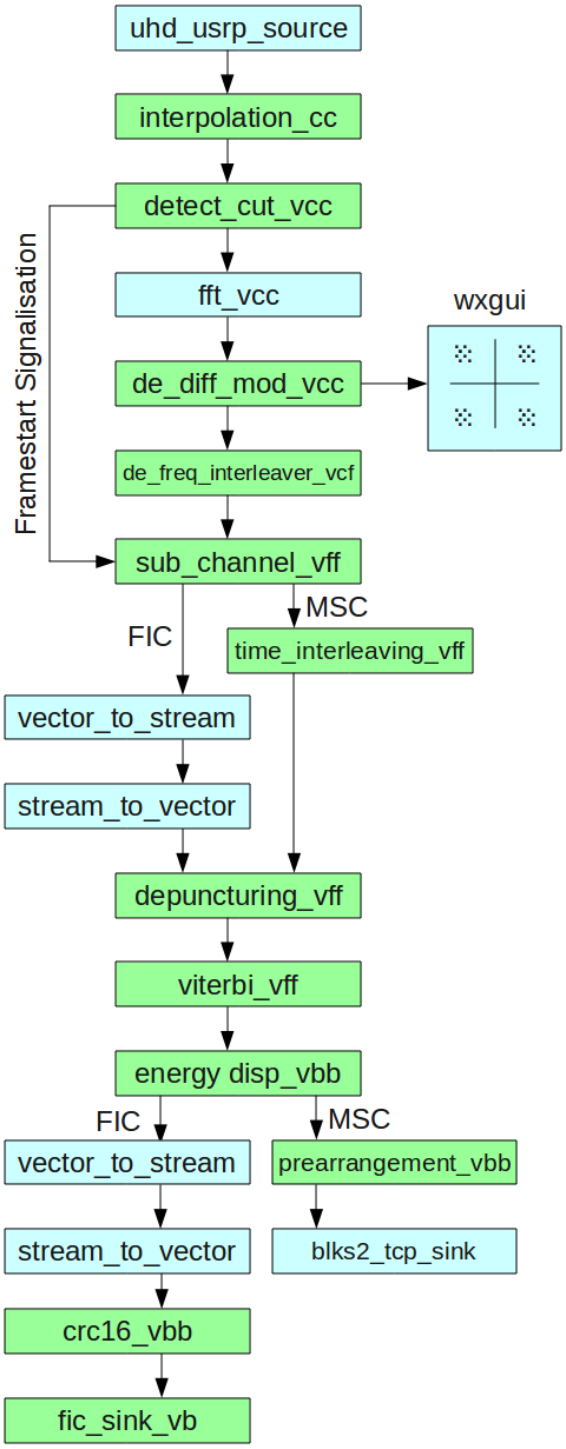


Abbildung 8.1.: Struktur der ersten Implementation des GNU Radio DAB Empfängers

zeigte, dass das Resampling eine Interpolation von 128 und eine Dezimation von 125 aufweisen muss (2000 MHz * 128 / 125 = 2048 MHz). Um dies zu realisieren schrieb man

einen Block zur linearen Interpolation. Dabei werden zwei aufeinander folgende Samples über eine Gerade verbunden und die gewünschte Anzahl Interpolationspunkte auf der Geraden berechet. Die Abbildung 8.2 verdeutlicht das Vorgehen an einem kleinen Beispiel. Die Parameter und die Blockeigenschaften des Interpolationsblocks sind im Anhang in der Tabelle D.2 abgebildet.

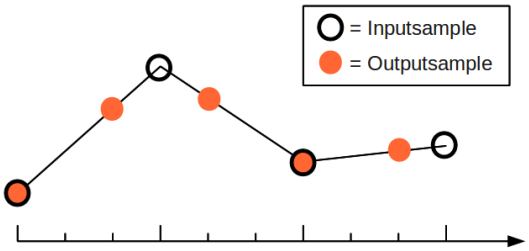


Abbildung 8.2.: Funktionsweise des Blocks zur Linearen Interpolation (Interpolation = 3 und Dezimation = 2)

8.1.2. Framedetektion und Symbolerkennung

Diesem Objekt werden die Modieigenschaften aus dem Standard [1, Seite 145] als Parameter übergeben. Es wurde keine automatische Modierkennung eingebaut. Die Framestartdetektion sollte jedoch für alle Modi funktionieren (durch entsprechende Parameterwahl bei der Objekterzeugung), ausgiebig getestet wurde sie jedoch nur mit den Mode 1 Parametern. Die Tabelle D.3 im Anhang der Dokumentation zeigt die einstellbaren Parameter des Framestartdetektionsblocks bei der Objekterzeugung.
Die Detektion des Übergangs vom Nullsymbol zum PRS wird wie in der Matlabsimulation über ein Moving Average Filter zur RSSI Bildung realisiert. Für die Bildung benötigt man die Amplitudeninformation des komplexen Signals am Eingang des Blocks, welche optimalerweise mit dem Satz von Pythagoras realisiert wird (Gleichung 8.1).

$$c = \sqrt{a^2 + b^2} \tag{8.1}$$

Satz von Pythagoras

Die Wurzelfunktion zu implementieren ist sehr rechenaufwändig. Damit die Amplitudenberechnung rechenarm ausfällt, wurde für die Pythagorasberechnung eine Näherungsformel gebraucht, welche in der Radartechnik eingesetzt wird [T]. Ist die Ankathete länger als die Gegenkathete, wird die Hypothenuse aus der Addition der Ankathete und der halben Gegenkathete gebildet. Ist die Ankathete kürzer als die Gegenkathete, wird die Hypothenuse aus der Addition der Gegenkathete und der halben Ankathete gebildet. Die Abbidlung 8.3 zeigt den Fehler dieser Näherung in Abhängigkeit zum Winkel des komplexen Signals (bei polarer Schreibweise). Um diese Näherung zu

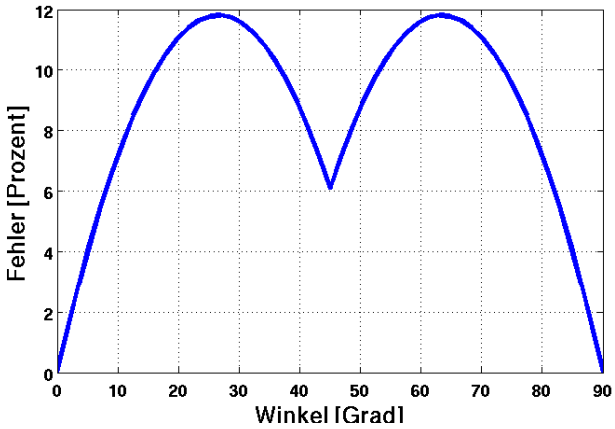


Abbildung 8.3.: Fehler der Pythagoras Näherungsformel gegenüber dem Pythagoras Satz

implementieren, benötigt man nur eine Addition und ein Rightshift (/2), was im Vergleich zum Quadrieren und Wurzelziehen der Ursprungsformel verschwindend gering ist.
Nachdem der Beginn des PRS detektiert wurde, wird eine kurze Zeit gewartet. In der Abbildung 8.4 ist diese Zeit als Offset eingezeichnet. Dieser Offset bewirkt, dass die Bereiche für die FFT in der Mitte eines Symbols zu liegen kommen. Wie in der Matlabsimulation erkannt, ist ein Offset von 100 Samples angemessen. Nach dem Offset beginnt ein Wechselspiel zwischen dem Herausschneiden von DAB Symbolen und dem Warten auf das nächste DAB Symbol.
Der ganze Block wurde als State Machine mit vier Zuständen aufgebaut. Beim Testen mit realen Daten konnte gezeigt werden, dass der Anfang der PRS mit grosser Zuverlässigkeit erkannt wird. Als Widerspiegelung der Zuverlässigkeit diente dabei die Distanz zwischen zwei Detektionen.

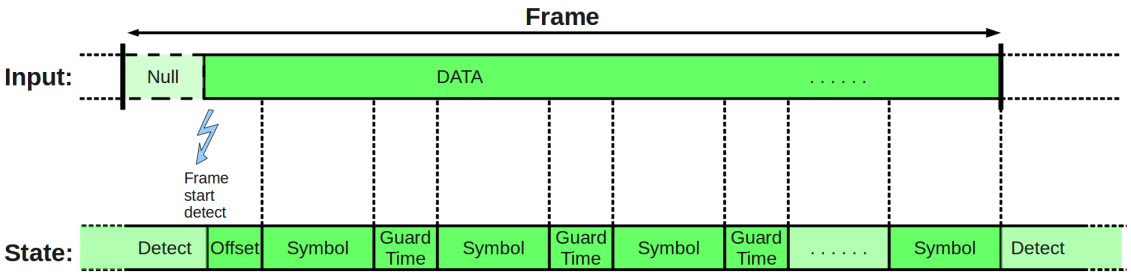


Abbildung 8.4.: Funktionsweise des Detektionsobjekts

8.1.3. FFT, differentielle Demodulation und Frequency Interleaving

Nach dem die einzelnen Symbole durch das Detektionsobjekt bestimmt wurden, wird nun wie im Kapitel 5 beschrieben die FFT aus den Symboldaten vollzogen. Dank dem Resampling kann die FFT über eine Zweierpotenzlänge (im Mode 1 sind das 2048 Samples) berechnet werden, was die Geschwindigkeit massiv erhöht. Zudem kann dank der

Zweierpotenzlänge ein vorgefertigter FFT Block verwendet werden. Da der Block in allen Simulationen zuverlässig funktionierte und in vielen Internetprojekten eingesetzt wird, verwendete man den Block ohne weitere Tests.
Die Tabelle D.4 im Anhang zeigt die einstellbaren Parameter des FFT Objekts bei der Generierung. Da es sich um ein komplexes OFDM Signal handelt wird nach der FFT ein Shift ausgeführt, damit alle Frequenzbins der Subcarrier aneinander gereiht im berechneten Spektrum liegen.
Die Tabelle D.5 im Anhang zeigt die Eigenschaften des Blocks für die differentielle Demodulation. Da dieser Block nur wenig Rechenleistung verbraucht, wird einfachheitshalber die Demodulation auf das ganze FFT-Ergebnis angewendet. Theoretisch würde die Demodulation bei den Subcarrierbins ausreichen.
Der erste Ausgang des Blocks gibt das demodulierte Signal als Vektor aus. Der Zweite gibt das demodulierte Signal eines Subcarriers aus (Gewünschter Subcarrier ist über ein Parameter bei der Objekterzeugung wählbar). Dieser zweite Ausgang kann beispielsweise auf ein Scope geleitet werden und zur Visualisierung der Modulation dienen.
Er wurde als optional implementiert. Das heisst, dass man beim generieren des Flow Graphs entscheiden kann ob dieser Ausgang genutzt wird oder nicht. Wenn man ihn an keinen weiteren Block anschliesst, wird er als inaktiv aufgefasst und es entstehen keine Fehler.
Die Eigenschaften des Frequenz Deinterleaving Blocks sind in der Tabelle D.6 im Anhang aufgezeigt. Beim Erzeugen dieses Objekts wird die Tabelle des Standards [1, Seite 158] generiert.
Im Betrieb macht der Block aus den demodulierten FFT Daten durch Neuanordnung mit Hilfe der erzeugten Tabelle die frequenzdemodulierten Daten. Die unbenutzten Bins der FFT werden abgeschnitten. Da es die Blockstruktur zulässt, wurde im gleichen Block das Demapping implementiert (aufspalten und neu anordnen des Real- und Imaginärteils des komplexen Signals). Nach diesem Block arbeitet der Flow Graph nur noch mit Floating-Point Daten.

8.1.4. FIC und Subchannel aus den DAB Symbolen herauslesen

Im Block werden aus den Daten des FICs die FIBs und aus den CIFs die Daten für den gewünschten Subchannel extrahiert. Der FIB Ausgang wurde als optional implementiert, dass heisst die FIBs können implementationsabhängig nach Wunsch beachtet oder ignoriert werden. Alle einstellbaren Parameter dieses Blocks zeigt die Tabelle D.7 im Anhang.
Als grosses Problem dieses Signalverarbeitungsblocks stellte sich heraus, dass bei GNU Radio alle Ausgänge die gleiche Rate an produzierten Datenblöcke aufweisen müssen. Um diesem Problem auszuweichen nutzte man die Tatsache, dass sich in einem DAB Frame Mode 1 vier FIBs und vier CIFs befinden. Die FIBs werden beim Erkennen zwischengespeichert und erst später beim Erkennen eines CIFs an den Ausgang geschrieben. Mit diesem Trick stehen an jedem Ausgang des Signalverarbeitungsblocks immer die gleiche Anzahl produzierter Datenblöcke bereit. Der Nachteil der Implementation mit dem Zwischenspeicher ist, dass dieser Block nur für den Mode 1 eingesetzt werden kann. Bei allen andern Modi muss der Code abgeändert und gegebenenfalls von Grund auf neu aufgebaut werden.
Die Signalverarbeitung wird durch eine State Machine gesteuert. Durch das Erkennen eines Framestarts wird der Symbolzähler zurückgesetzt und man beginnt mit dem Abspeichern der FIBs aus den FIC Symbolen. Nachdem diese Phase abgeschlossen ist, wird in

Abhängigkeit der eingestellten CU Startadresse und der CU Grösse der Subchannel ausgelesen. Diese Daten werden ebenfalls in einem Zwischenspeicher abgelegt und erst mit dem vollständigen Empfang des gewünschten Subchannels an den Ausgang geschrieben. Nachdem jedes DAB Symbol eines Frames verarbeitet wurde, beginnt die State Machine wieder von vorne und es wird auf den nächsten Framestart gewartet.
Um zu verhindern, dass die State Machine bei einer Störung nicht mehr den Betrieb aufnehmen kann, werden laufend die DAB Symbole gezählt. Wird zur Laufzeit ein Zählerwert grösser als 76 festgestellt, kehrt die State Machine in den Zustand der Framestartsuche zurück.
Damit GNU Radio erkennt wie viele Inputblöcke für einen Block am Ausgang gebraucht werden, wird das Verhältnis zwischen jedem Ein- und Ausgang mit Hilfe der "Forecast" Funktion definiert. Wie sich jedoch zeigt, dürfen für einen Ausgangsblock die Inputblöcke maximal aus einer Gesamtlänge von 8192 Floating-Point Werten bestehen. Um diesem Problem entgegen zu wirken, teilte man jeden Block am Ausgang in acht Teilblöcke auf. Dies hat den Vorteil, dass GNU Radio meint man würde acht Mal mehr Datenblöcke am Ausgang generieren, womit sich das Verhältnis zwischen benötigter Inputblöcke und produzierten Outputblöcken veränderte. Die acht Teilblöcke werden gleichzeitig produziert, dies spielt aber keine Rolle. Mit dieser Implementation generiert man im Mode 1 aus 76 DAB Symbolen 32 Datenblöcke (4 CIF aufgespaltet in jeweils acht Teilblöcke).

8.1.5. Time Interleaving

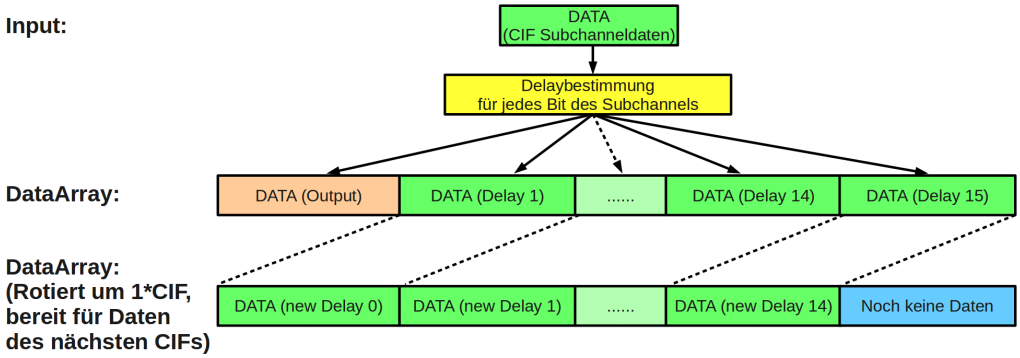


Abbildung 8.5.: Funktionsweise des implementierten Time Interleaving

In diesem Block findet das Interleaving statt und die Daten der Teilblöcke eines Subchannels werden zu einem Subchannel zusammengefügt.
Für das Time Interleaving wird mit einem grossen Floating-Point Array gearbeitet. Dieses hat die sechzehnfache Subchannelgrösse. Nach der Verarbeitung der neuen Subchanneldaten wird das Array jeweils gedreht. Wie im Abschnitt 6.6 erklärt, wird in Abhängigkeit der Position eines Zeichens im Datenstrom dessen Delay bestimmt. Deshalb werden die entsprechenden Floating-Point Werte an die entsprechende Position im Array geschrieben und erhalten dadurch die gewünschte Verzögerung. Die Daten mit der Verzögerung Null, werden nach jedem Durchgang an den Ausgang geschrieben. Es ist zu beachten, dass dieser Block erst nach 16 Durchgängen die ersten vollständig gültigen Daten am Ausgang aufweist.

Dies gilt ebenfalls nach einer Störung. Das Vorgehen ist in der Abbildung 8.5 grafisch dargestellt.
Um eine grosse Flexibilität zu erreichen wurden die Delays für die einzelnen Positionen im Datenstrom (insgesamt gibt es 16 Möglichkeiten) in einem Array abgelegt und können sehr einfach für spätere Anwendungen angepasst werden.
Die Tabelle D.8 des Anhangs zeigt die Parameter bei der Erzeugung des Time Interleaving Blocks.

8.1.6. Depuncturing

Bei der Implementierung des Puncturings wurde nur die UEP umgesetzt. Die Funktionsweise des Objekts ist gleich wie in der Matlabsimulation (Abschnitt 7.5).
Durch die Parameterübergabe (Alle PI und die Länge der Puncturingvektoren) bei der Objekterzeugung wird als erstes berechnet aus wie vielen Datenbits die Teilvektoren bestehen, zudem wird die Punktierungstabelle des Standards [1, Seite 131] als konstantes Array eingebettet. Mit diesen Informationen werden im Betrieb die Bereiche punktiert. An den Stellen wo sich eine Null im Punktierungsarray befindet, wird wie in der Matlabsimulation eine Null in den Ausgabevektor geschrieben und bei einer Eins werden die Daten übernommen. Die letzten 12 Bits des Vektors (Tail) werden immer mit dem PI 8 verarbeitet.
Die Länge des Ausgabevektors ist abhängig von der CU Anzahl des gewählten Subchannels und der Wahl der Punktierung.
Die Tabelle D.9 des Anhangs zeigt die Parameter des Depuncturing Blocks.

8.1.7. Viterbidecoder

Zu Beginn der Implementation ging man davon aus, dass der Viterbidecoder nicht selbst geschrieben werden muss. Wie sich jedoch bei kleineren Tests herauskristallisierte, produzierte der Viterbidecoder immer wieder fehlerhafte Bursts, zudem fügte er öfters unerklärbare Bits in den Datenstrom ein. Die mit der Bibliothek zur Verfügung gestellten Beispielprogramme zeigten, dass das Hauptaugenmerk der Bibliothek nicht auf dem Viterbidecoder sondern dem Faltungscodierer lag. Dies lässt vermuten, dass der Viterbidecoder noch nie in grösseren Projekten eingesetzt wurde.
Da das Verhalten des Decoders nach längerem Testen nicht unter Kontrolle gebracht werden konnte, entschied man sich den Viterbidecoder selbst auszuprogrammieren. Als Grundlage diente der für den FIC geschriebene Algorithmus aus der Matlabsimulation. Bei der Implementierung in C++ wollte man aber einen ressourcenschonenderen Ablauf als in der Simulation erreichen.
Der wichtigste Schritt zu einem effizienten Aufbau des Decoders ist, dass bei der Objekterzeugung so viele Daten wie möglich aus dem Generatorpolynoms berechnet werden und während der Laufzeit zur Verfügung stehen. Die Abbildung 8.6 verdeutlicht den Inhalt zwei wichtiger Arrays die bei der Objekterzeugung generiert werden. Über das Array “Pfad“ kann währen der Laufzeit aus dem aktuell zu berechnenden State die beiden Vorgängerstates bestimmt werden. Mit dem zweiten Array “Ref“ kann die generierte Sequenz des Faltungsencoders für einen bestimmten Übergang ermittelt werden.

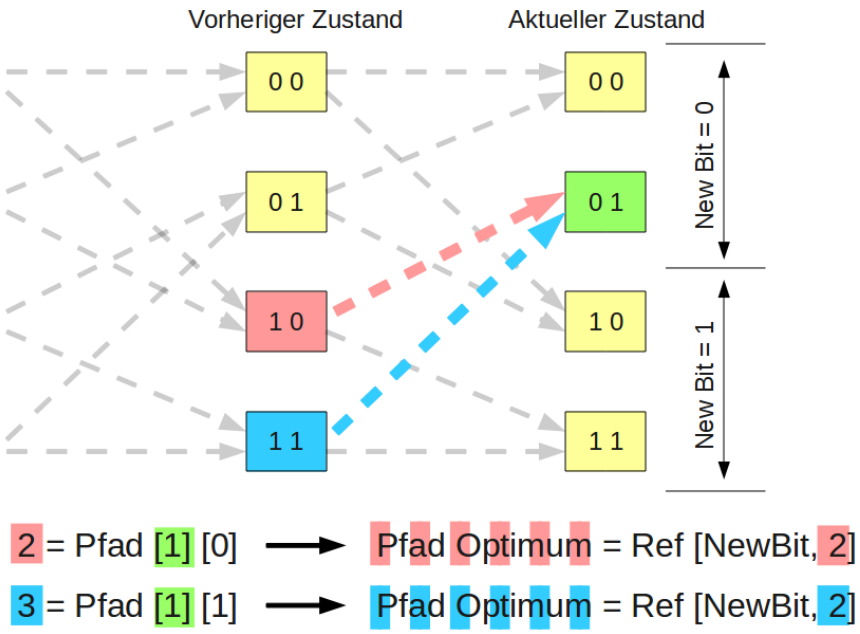


Abbildung 8.6.: Bei der Objekterzeugung berechnete Arrays des Viterbidecoders am Beispiel des binären Zustands 01

Während der Laufzeit wird eine Metrik über den letzten und den aktuellen Zustand nachgeführt damit die jeweiligen Siegerpfade bestimmt werden können. Um den Start im Zustand Null zu forcieren wird der Metrik Wert des Zustands Null zu Beginn mit einem grösseren Wert als die anderen Metrik Zustände initialisiert. Am Ende des gesamten Datenblocks wird der optimale Pfad rückwärts (Ausgehend vom Null Zustand) bis zum ersten Zustand ermittelt. Somit hat der erstelle Viterbidecoder eine Speichertiefe gleich der zu ermittelnden Anzahl Bits.
Die Blockeigenschaften des Objekts sind in der Tabelle D.10 im Anhang aufgelistet.

8.1.8. Energy Dispersal

Dieser Block hat die Aufgabe bei der Objekterzeugung die PRBS Sequenz zu generieren und in einem Array abzuspeichern. Die Sequenz wird durch ein Schieberegister auf die geforderte Vektorlänge des Eingangs vorausberechnet, somit kann der Block über Anpassung der Parameter für den FIC und den MSC gebraucht werden.
Die Verarbeitung findet blockweise statt. Als Input dienen die Ausgangsdaten des Viterbiblocks inklusive Tail. Bei der Verarbeitung wird eine XOR Verknüpfung auf die Daten angewendet und die Tail abgeschnitten.
Die Objekteigenschaften des Energy Dispersal Blocks sind in der Tabelle D.11 im Anhang abgebildet.

8.1.9. Cyclic Redundancy Check

Der CRC Block erwartet, dass das Zweierkomplement des CRCs an den Datenvektor angehängt ist. Die Länge des Datenvektors kann durch die Parameter bei der Objekterzeugung definiert werden. Durch diese Flexibilität kann dieser Block zur CRC Überprüfung bei den FIBs und bei einer allfälligen DAB+ Implementierung im MSC eingesetzt werden.
Der Block ist dazu ausgelegt die Daten inklusive CRC Anhang durch das Schieberegister zu lassen und anschliessend zu überprüfen ob die Summe aus den Bits im CRC Register 0 ergibt. Damit mehrere Blöcke nacheinander verarbeitet werden können, wird nach der Verarbeitung jedes Blocks das Schieberegister und der Summenspeicher zurückgesetzt.
Die Objekteigenschaften des CRC Blocks sind in der Tabelle D.12 im Anhang abgebildet.

8.1.10. FIB Sink

Der FIB Sink Block wird als Abschluss der FIB Kette gebraucht. Er durchsucht FIBs nach FIGs des Typs 1 mit der Extension 1. Diese Gruppen enthalten die Servicenamen welche das Ensemble aussendet. Wird ein Sendername gefunden, wird er auf der Konsole angezeigt. Es hat sich gezeigt, dass etwa jede Sekunde alle Sendernamen im FIC übertragen werden. Deshalb eignet sich dieser Block bestens für eine Präsentation, da fortlaufend Radiosendernamen angezeigt werden. Alle anderen FIGs werden nicht analysiert und werden ohne Verwendung verworfen.
Die Objekteigenschaften dieses Blocks sind in der Tabelle D.13 im Anhang abgebildet.

8.1.11. Prearrangement und TCP Sink

Der Prearrangement Block (Eigenschaften in der Tabelle D.14 im Anhang dargestellt) hat einzig die Aufgabe acht Bits zu einem Byte zusammenzufassen und an den TCP Block weiterzugeben. Bis zu diesem Block enthält jedes Char Element einzig den Wert 0 oder 1. Die restlichen sieben Bits des Elements enthalten konstant Nullen. Dies wurde gemacht, damit die Algorithmen der vorangehenden Blöcke übersichtlicher sind.
Die TCP Sink (Eigenschaften in der Tabelle D.15 im Anhang dargestellt) hat die Aufgabe die errechneten MP2 Daten eines Services an das Audioprogramm weiterzugeben. Der Block wird dazu als Server initialisiert. Bei der Objekterzeugung wartet der Block bis ein Programm wie beispielsweise der VLC Player (Verbindungsadresse tcp://IP:9999) auf den gewählten Port verbindet. Danach werden alle Daten an den Player gesendet.

8.2. Zweite Implementierung des DAB Empfängers

Die Abbildung 8.7 zeigt die Struktur der zweiten Implementation des GNU Radio DAB Empfängers. Wieder sind die blau gefärbten Blöcke aus den GNU Radio Bibliotheken und die grün gefärbten Blöcke selbst geschrieben. Die markierten Bereiche stellen die Veränderungen gegenüber der ersten Implementation dar.
Die Implementationsveränderungen wurden nötig, um einen dauerhaft zuverlässigen Betrieb des Empfängers garantieren zu können. Zudem wurden während der ersten Implementation

sehr viele Erkenntnisse über Programmiermöglichkeiten von GNU Radio gewonnen. Diese konnten in der zweiten Implementation eingesetzt werden. Dadurch wird die Struktur des Empfängers einfacher.

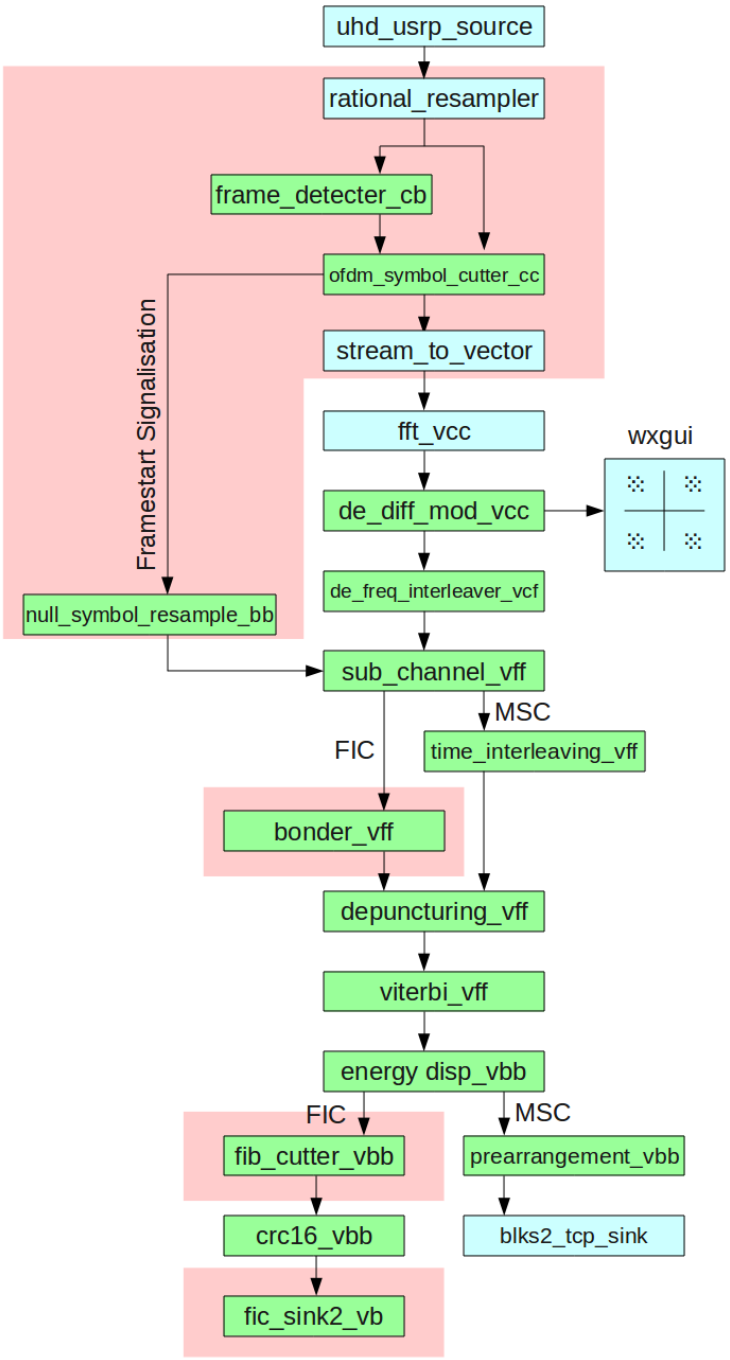


Abbildung 8.7.: Struktur der zweiten Implementation des GNU Radio DAB Empfängers

8.2.1. Resampling

Durch eine Simulation mit einem Referenzfile als Input konnte man zeigen, dass es sich lohnt für das Resampling den vorgefertigten GNU Radio Block zu verwenden. Lässt man das Programm mit dem selbst geschriebenen Interpolationsblock laufen, weist die Visualisierung der Modulation grosse Streuungen auf und es können vermehrt CRC Fehler im FIC festgestellt werden. Vermutet wird, dass der lineare Interpolator das Spektrum des Signals stark verzerrt.
Wird hingegen mit dem Resampler von GNU Radio gearbeitet ist bei der Visualisierung der Modulation die QAM Modulation mit nur geringen Schwankungen zu erkennen (Abbildung 8.8). Zudem konnten fast keine CRC Fehler mehr festgestellt werden. Wie der GNU Radio Resampling Block genau aufgebaut ist konnte auf die Schnelle nicht herausgefunden werden, dazu müsste der Quellcode analysiert werden.
Die Tabelle D.16 im Anhang zeigt die eingestellten Parameter für den Resampling Block.
Ein Überwachen der Central Processing Unit (CPU) Auslastung zeigte nach dem einsetzen des Resampling Blocks einen Anstieg von etwa 20 Prozent. Dies deutet darauf, dass dies ein sehr ressourcenintensiver Block ist.

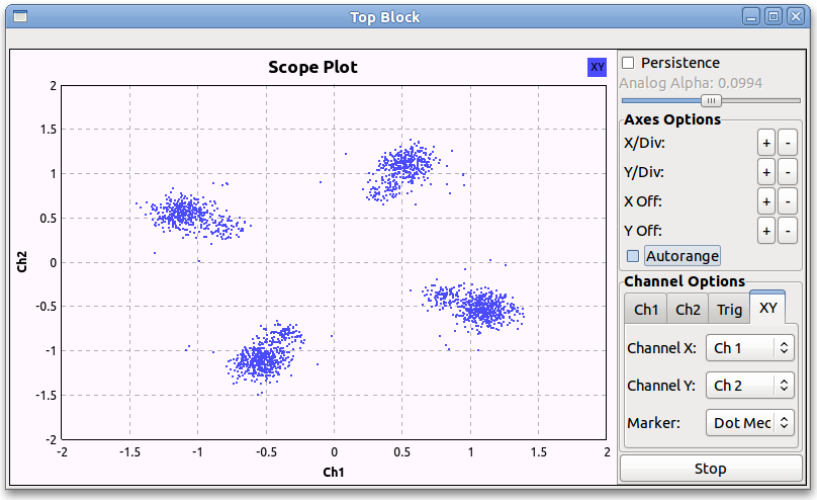


Abbildung 8.8.: Mit GNU Radio aufgenommene QAM Modulation eines Subcarriers bei 227.36 MHz

8.2.2. Framestart Detecter und OFDM Symbol Cutter

Der Block “detect_cut_vcc“ stellte einem vor das Problem, dass er zu stark verschachtelt aufgebaut ist um seine Funktionstüchtigkeit einwandfrei überprüfen zu können. Deshalb entschied man sich den Block in zwei kleinere Blöcke aufzuteilen.
Einerseits entstand daraus der Detekterblock. Dieser Block signalisiert mit einer 1 am Ausgang, wenn er das Ende des Nullsymbols erkannt hat. Dazu wurde aus dem alten Block die Absolutwertbildung und das Moving Average Filter übernommen. Über die Parameter kann der Block berechnen wie viele Samples ein DAB Frame zwischen zwei Nullsymbolen enthält. Mit diesem Wissen verhindert er eine Doppeldetektion eines Frames.

Durch die Information einer Detektion und den Daten des Resamplers kann der Symbolcutter (der zweite neu entstandene Block) arbeiten. Der Symbolcutter zählt nach einer Detektion die Samples und schneidet die DAB Symbole aus dem Datenstrom. Neu kann zusätzlich zu den alten Parametern die Zeit zwischen der Framestartdetektion und dem Ausschneiden des ersten Symbols bei der Objekterzeugung festgelegt werden. Wie vorher arbeitet er mit einer State Machine, aber im Gegensatz zum alten Block verarbeitet er nicht mehr ganze Datenblöcke sondern nur noch einzelne Samples.
Der Nachteil der Aufsplittung ist, dass der Detektor nicht mehr von der State Machine ein- und ausgeschaltet wird. Da der Detektor neu ein eigener Block ist, ist er immer aktiv. Daraus kann man ableiten, dass durch das Aufsplitten in zwei Blöcke mehr CPU Ressourcen verbraucht werden.
Beide Blöcke konnten einzeln durch eine Testbench auf ihre Funktionstüchtigkeit überprüft werden. Die Tabelle D.17 und die Tabelle D.18 im Anhang zeigen die Eigenschaften der neu entstandenen Blöcke.

8.2.3. Null Symbol Resampler, Bonder und FIB Cutter

In diese drei Blöcke wurde nur wenig Signalverarbeitung implementiert. In erster Linie dienen sie dazu einen störungsarmen Betrieb zu ermöglichen oder gewisse Anpassungen für den Programmablauf zu erledigen.
Da an allen Blockausgängen die gleiche Rate erzeugt werden muss, musste zwischen dem OFDM Symbol Cutter Block (produziert am Ausgang Samples) und dem Herausarbeiten des Subchannels (Verlangt am Eingang nur ein Sample pro DAB Symbol) eine Anpassung im Pfad “Framestart Signalisation“ eingebaut werden. Um dies zu erreichen werden die Daten auf dieser Steuerleitung reduziert. Es wird immer nur der erste Sample eines DAB Symbols an die nächste Entity weitergeleitet. Alle verworfenen Samples enthalten Nullen und werden nur wegen der Ratengleichheit am Ausgang des OFDM Symbol Cutters produziert.
Damit die FIC Daten nach dem Block zum Herausarbeiten des Subchannels nicht serialisiert und nachher wieder zu einem neuen Vektor zusammengeführt werden müssen, erstellte man den Bonder Block. Dieser vereint beide Funktionen in einem Block und trägt somit zur Übersichtlichkeit bei. Ein positiver Nebeneffekt des Bonder Blocks ist, dass das System an Stelle von zwei Blöcken nur noch einen aufrufen muss. Somit wird der Scheduler entlastet.
Der FIB Cutter Block hat einzig die Aufgabe einen grossen Datenvektor in mehrere kleine Vektoren einer gewünschten Grösse zu unterteilen. Gebraucht wird diese Funktion im DAB Empfänger nach dem Energy Dispersal Block um die drei aneinander gehängten FIBs aufzutrennen.
Die Eigenschaften und die Parameter der drei beschriebenen Blöcke können in den Tabellen D.19, D.20 und D.21 im Anhang nachgesehen werden.

8.2.4. FIB Sink 2 und FIB Sink 3

Der FIB Sink 2 Block ist eine Erweiterung des ursprünglichen FIB Sink Blocks. Gegenüber der ersten Version wertet er zusätzlich die Type 0 Extensions 1,2,9 und 10 aus. Aus diesen Informationen generiert er eine Datenbank wie im Abschnitt 6.9.2 beschrieben. Immer nach der Verarbeitung der neusten Daten des FICs werden alle empfangenen Informationen auf die Konsole geschrieben. Die Abbildung 8.9 zeigt, wie die Konsole während der Laufzeit

aussieht.

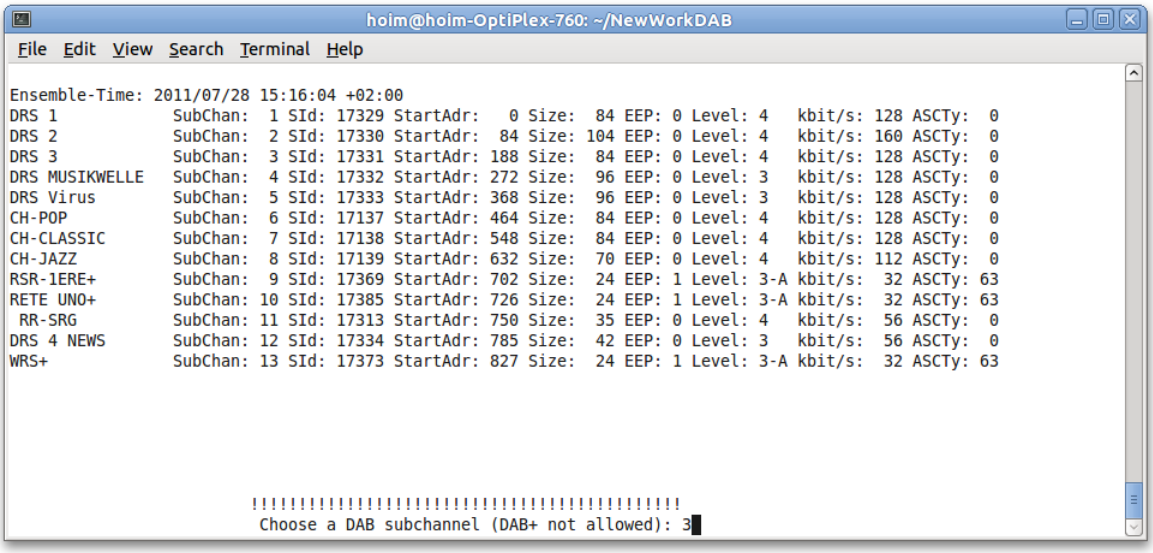


Abbildung 8.9.: Ausgewertete FIC Informationen auf der Konsole dargestellt

Neben den Senderinformationen welche für das Decodieren des MSC benötigt werden, wird die Uhrzeit und das Datum des Ensembles angezeigt. Das Datum wird im Modified Julian Date Format übermittelt, daher wird es vor dem Anzeigen auf der Konsole in Jahre, Monate und Tage umgerechnet. Die Millisekunden wurden absichtlich nicht angezeigt da man der Ansicht ist, dass diese Information in der gegenwärtigen Implementation für den Betrachter überflüssig ist.
Die FIB Sink 3 enthält vom implementierten Funktionsumfang die gleichen Elemente wie die FIB Sink 2. Das Spezielle des Abfluss 3 ist, dass er die erzeugte Datenbank weitergibt und der MSC auf die benötigten Daten zugreifen kann.

9. Projektabschluss

9.1. Testaufbau

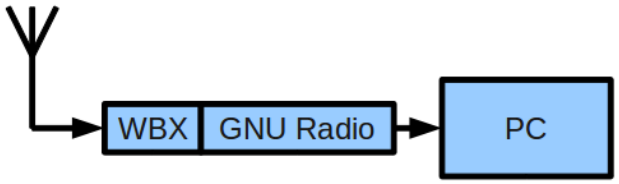


Abbildung 9.1.: Aufbau des DAB Empfängers

Um die Hardware zu testen wurde ein Aufbau wie in der Abbildung 9.1 gewählt. Als Antenne diente eine Autoradioantenne welche auf 50 Ohm angepasst wurde. Diese Antenne ist für den FM Radio Frequenzbereich gedacht, da aber für den Aufbau keine auf den DAB Frequenzbereich optimierte Antenne gefunden werden konnte und die Zeit nicht reichte eine eigene zu entwerfen, entschied man sich für diese FM Autoradioantenne.
Das GNU Radio hat man mit einem Gigabit-Ethernetkabel direkt auf eine Netzwerkkarte des Versuchscomputers verbunden.

9.2. Testprogramme

Für den Versuchsbetrieb des DAB Radios wählte man die DAB Frequenz 227.36 MHz des "SRG SSR D01" Ensembles und erstellte die folgenden drei Python-Testprogramme:

- **LabelRealtime.py** Mit diesem Programm werden dem Betrachter in Realtime die empfangenen Servicelabels angezeigt. Dieses Programm eignet sich gut für eine Ausstellung wie beispielsweise die Nacht der Technik, da sich fortlaufend etwas auf dem Bildschirm verändert und keine speziellen Fachkenntnisse zum Verstehen der dargestellten Daten erforderlich sind.
  In den Flow Graph wurde nur die FIC Kette eingebunden, da nichts aus dem MSC gebraucht wird. Das Programm erwies sich in einem Zweistundentest als zuverlässig und verbrauchte während des Betriebs immer etwa gleichviel Arbeitsspeicher.
- **ServiceDB.py** Dieses Programm erstellt eine Datenbank aus den Serviceinformationen des FICs und zeigt die Daten auf dem Display an. Das Füllen der Datenbank dauerte bei den getesteten Ensembles etwa 1.5 Sekunden, nachher sind alle Informationen eingetroffen und die Datenbank ist gefüllt. Eine Vorführung dieses Programms lohnt

sich beispielsweise bei einer Fachmesse, wenn davon ausgegangen werden kann, dass der Betrachter Grundkenntnisse der Signalverarbeitung besitzt.

- **Combine.py** In diesem Programm wurde die FIC Kette und die MSC Kette implementiert. Während der ersten drei Sekunden nach dem Programmstart füllt das Programm die FIC Datenbank mit allen Serviceinformationen. Danach wird der Benutzer aufgefordert einen DAB Sender auszuwählen und einen Musikplayer mit dem Programm zu verbinden. Sind diese beiden Schritte vollbracht, kann der Betrachter den gewünschten Musiksender live hören.
  Dieses Programm testete man zwei Stunden am Stück. Abgesehen von wenigen sehr kurzen Unterbrüchen bei der Musikwiedergabe hat das Programm einwandfrei funktioniert.

9.3. Fazit

Die Auslastung des Versuchscomputers (Anhang F) belief sich bei allen Testprogrammen auf Durchschnittlich 65 Prozent, wobei Schwankungen von ±5 Prozent wahrgenommen werden konnten.
Bei einem Testlauf stellte man fest, dass der Ton des DAB Musikservice “RR-SRG“ nicht wiedergegeben werden konnte. Die anderen Musikservices funktionierten einwandfrei. Vermutlich ist beim Übernehmen der Puncturingtabelle aus dem Standard ein Fehler unterlaufen. Für eine genauere Analyse der Ursache fehlte aber die Zeit.
Interessant ist die Verzögerung des Tones zwischen dem erstellten DAB Musik Radio und dem FM Musik Radio. Sie beträgt etwa drei Sekunden. Ein kleiner Anteil kann auf das Time Interleaving zurückgeführt werden, da im MSC alle Datenbits eine Verzögerung von 16 CIF erfahren. Die verbleibende Verzögerung muss vom DAB Transmitter und vom GNU Radio DAB Empfänger stammen.
Beim vorliegenden Projekt kann man von einem funktionierenden DAB Empfänger sprechen.

10. Ausblick

10.1. DAB/DAB+

- Für das ZSN Schaufenster und für Ausstellungen könnte mit mässigem Aufwand ein DAB Demonstrationsobjekt erstellt werden. Dabei könnte dem Betrachter zusätzlich zur Grafik mit der Modulation eines Subcarriers, das Spektrum eingeblendet oder die aktuelle Bitfehlerrate gezeigt werden.
  Vielleicht wäre es auch interessant das Empfangen der FIC Daten verlangsamt und animiert darzustellen, damit der Betrachter nicht in zu kurzer Zeit mit zu vielen Informationen überschwemmt wird und den Aufbau der Datenbank mitverfolgen kann.
  Wenn das Schreiben von neuen Signalverarbeitungsblöcken in Betracht gezogen wird, könnte es interessant sein die Liedernamen und die Trafficinformationen der Services herauszuarbeiten. Spannend wäre in diesem Zusammenhang, dem Betrachter die Zusatzinformationen aller Services des Ensembles gleichzeitig anzuzeigen. Zudem könnte mit den Trafficinformationen bewiesen werden, dass Livedaten verarbeitet werden.
  Eine andere Möglichkeit wäre, dass man alle Services des MSC gleichzeitig im Internet broadcasten würde. Bei diesem Unterfangen müsste sehr wahrscheinlich ein leistungsstärkerer Computer die Signalverarbeitung übernehmen.
- Eine Erweiterung des Empfängers wäre beispielsweise das automatische Durchsuchen aller möglichen DAB Kanäle auf ein DAB Signal. Am Ende des Suchlaufs würden dann alle Ensembles und die dazugehörigen Services angezeigt. Diese Arbeit würde sich im Gegensatz zur jetzigen Implementation vermehrt in der Python Umgebung abspielen, da auf alle Signalverarbeitungsblöcke dieser Arbeit zurückgegriffen werden könnte.
  Gleichzeitig könnte versucht werden die Pythonfiles zu modifizieren um beim laufenden Programm den Sender wechseln zu können.
- Bei der Implementierung wurde die ganze Empfangskette des Empfängers auf den DAB Modus 1 ausgelegt. In einer Projektarbeit könnten die Signalverarbeitungsblöcke so erweitert werden, dass sie mit den Modi 2-4 arbeiten können. Man müsste sich im Vorfeld aber Gedanken machen, wie die einzelnen Modi getestet werden könnten.
- Zurzeit sind im DAB Empfänger nur die nötigsten Elemente um Musik abzuspielen implementiert. Vielleicht wäre es spannend die Datendienste ebenfalls zu implementieren, da im Ausland bestimmt auch Daten übertragen werden.
- Einer der ressourcenintensivsten Blöcke stellt das Resampling dar. Eine Implementierung eines einfacheren Verfahrens könnte den Ressourcenverbrauch stark reduzieren.
- Mit all dem erarbeiteten Wissen und den bereits geschriebenen Elementen wäre es spannend ein komplettes DAB-Analysetool zu schreiben. Eine kurze Suche im Internet zeigte, dass nur eine Firma in Deutschland [R] ein solches Gerät herstellt und vertreibt.
- Mit grossem Aufwand könnte man aus den Erstellten und mit einer Vielzahl neuer

Signalverarbeitungsblöcke einen eigenen DAB Sender implementieren. Dies wäre ein sehr spannendes Unterfangen. Mit dem erarbeiteten Wissen über DAB und GNU Radio müsste bis zu den ersten Funktionskontrollen mit einem Zeitaufwand von zirka zwei Mannmonaten gerechnet werden.

- Es ist anzunehmen, dass in nächster Zeit ein Simulink Treiber für die Anbindung eines UHD GNU Radios auf den Markt kommt. Mit diesem Treiber bestünde die Möglichkeit das ganze DAB Radio auch in Simulink zu realisieren. Da Simulink stark verbreitet ist könnte man bei einer Veröffentlichung auf reges Interesse stossen.
- Vielleicht würde es sich nach dem erfolgreichen Abschluss dieses Projekts lohnen nochmals mit dem Frauenhoferinstitut Kontakt aufzunehmen. Da immer mehr DAB+ Services ausgestrahlt werden, wäre es schön den Audiocodec für DAB+ einsetzen zu dürfen.

10.2. GNU Radio

- GNU Radio hat sich in dieser Projektarbeit als sehr mächtiges und zuverlässiges Hilfsmittel herauskristallisiert. Es könnte theoretisch in sehr vielen Signalverarbeitungsprojekten des ZSN einen grossen Nutzen erbringen. Einerseits hat es sich bewahrheitet, dass man mit GNU Radio auf einfachste Weise ein Signal aufzeichnen und im Nachhinein in einer Matlabsimulation verarbeiten und analysieren kann. Andererseits hat man mit GNU Radio und einem Computer die Möglichkeit in Echtzeit unglaubliche Datenmengen zu verarbeiten.
- Da GNU Radio sehr zuverlässig läuft, könnte es in zukünftigen Anwendungen durchaus dauerhaft in einem Demonstrationsobjekt oder einem Prototypen verbaut werden.

11. Conclusion

11.1. Schlusswort

Diese Arbeit empfand ich als sehr interessant und lehrreich. Vor allem die Vielseitigkeit mit den stetigen Erweiterungsmöglichkeiten sprach mich an. Daher war ich täglich motiviert und bin sehr zufrieden mit dem Resultat.
Ich würde mich freuen, wenn viele Personen durch dieses Open Source Projekt einen erleichterten Einstieg in die spannende DAB Welt erhalten. Zudem wäre es schön, wenn noch Erweiterungen für dieses Projekt geschrieben würden und dadurch die Effektivität weiter gesteigert und der Funktionsumfang des Gesamtsystems vergrössert werden könnte.

Abbildung 11.1.: M. Höin

11.2. Danksagung

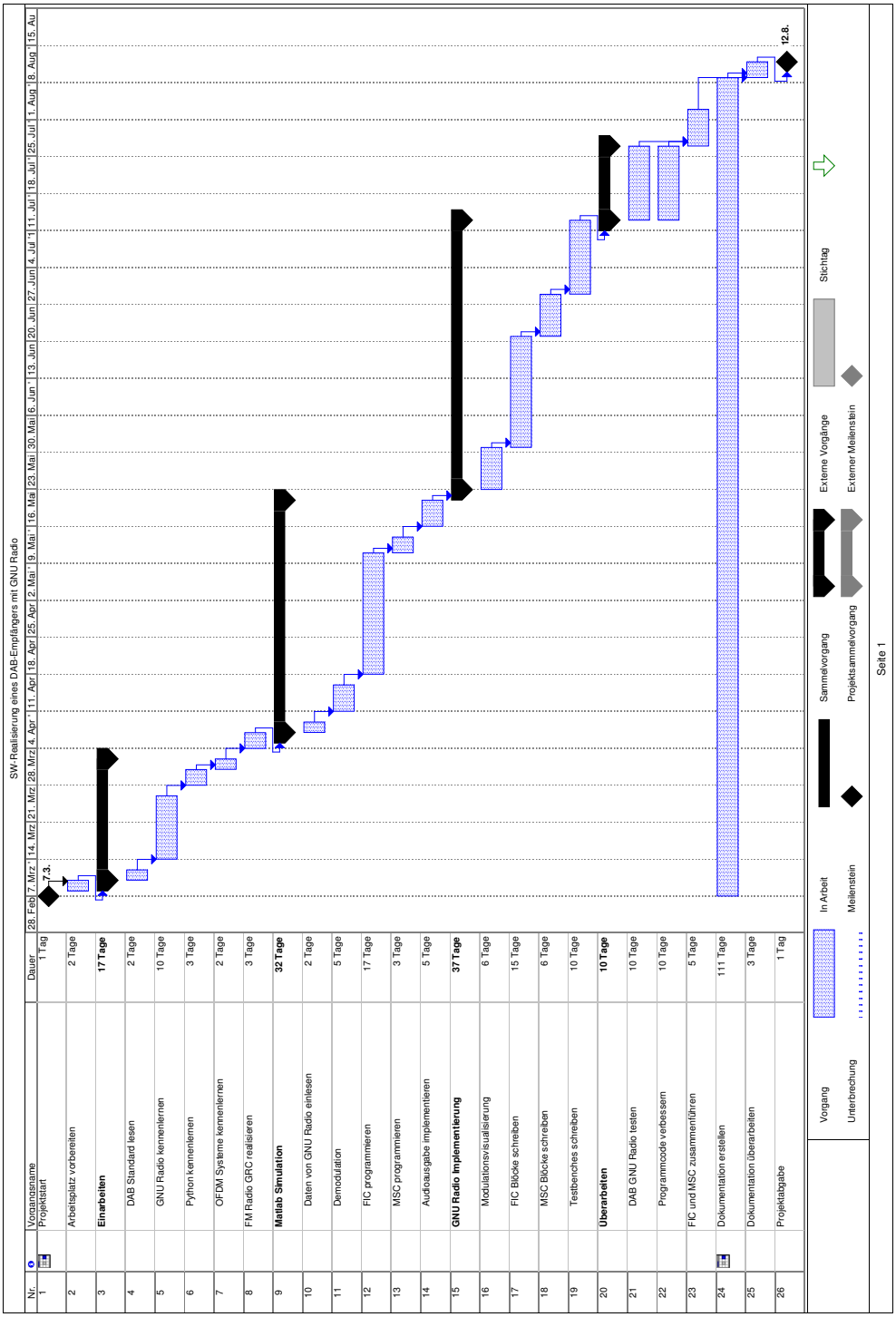
Ich danke herzlich meinem Betreuer Herr Prof. Dr. M. Rupf. Mit seinem riesigen Fachwissen konnte er mich immer mit guten Ideen unterstützen.
Bedanken möchte ich mich auch bei Markus Meier. Sein grosses Wissen über den Aufbau der Linux Umgebung hat mir mehrmals geholfen Probleme gezielt anzugehen.
Zudem möchte ich noch Michael Jäger danken, er stand mir immer mit Rat und viel Engagement zur Seite.
Vielen Dank.

Anhang

A. CD-ROM Wurzelverzeichnis

- Aufzeichnungen
- Dokumentation
- Matlabsimulation
- Python_Flow_Graph
- Repository
- Standards

B. Zeitplan


SW-Realisierung eines DAB-Empfängers mit GNU Radio
Nr.
Vorgangsname
Dauer
28. Feb
7. Mrz
14. Mrz
21. Mrz
28. Mrz
4. Apr
11. Apr
18. Apr
25. Apr
2. Mai
9. Mai
16. Mai
23. Mai
30. Mai
6. Jun
13. Jun
20. Jun
27. Jun
4. Jul
11. Jul
18. Jul
25. Jul
1. Aug
8. Aug
15. Au
1 Projektstart 1 Tag
2 Arbeitsplatz vorbereiten 2 Tage
3 Einarbeiten 17 Tage
4 DAB Standard lesen 2 Tage
5 GNU Radio kennenlernen 10 Tage
6 Python kennenlernen 3 Tage
7 OFDM Systeme kennenlernen 2 Tage
8 FM Radio GRC realisieren 3 Tage
9 Matlab Simulation 32 Tage
10 Daten von GNU Radio einlesen 2 Tage
11 Demodulation 5 Tage
12 FIC programmieren 17 Tage
13 MSC programmieren 3 Tage
14 Audioausgabe implementieren 5 Tage
15 GNU Radio Implementierung 37 Tage
16 Modulationsvisualisierung 6 Tage
17 FIC Blöcke schreiben 15 Tage
18 MSC Blöcke schreiben 6 Tage
19 Testbenches schreiben 10 Tage
20 Überarbeiten 10 Tage
21 DAB GNU Radio testen 10 Tage
22 Programmcode verbessern 10 Tage
23 FIC und MSC zusammenführen 5 Tage
24 Dokumentation erstellen 111 Tage
25 Dokumentation überarbeiten 3 Tage
26 Projektabgabe 1 Tag
7.3.
12.8.
Vorgang
Unterbrechung
In Arbeit
Meilenstein
Sammelvorgang
Projektsammelvorgang
Externe Vorgänge
Externer Meilenstein
Stichtag
Seite 1

C. GNU Radio Installationsanleitungen (Ubuntu 10.10)

C.1. UHD Treiber

Anleitungen und Hinweise zur aktuellen Version können auf der Homepage UHD Start nachgelesen werden.

1. Tarball von Ettus UHD Download herunterladen
2. Tarball in gewünschtes Verzeichnis entpacken
3. dot-File durch Doppelklick installieren

C.2. FPGA Firmware aktualisieren

Eine ausführliche Anleitung der Firmwareaktualisierungsschritte bei Schwierigkeiten kann auf der Homepage Raullen nachgelesen werden.

1. Neuste Firmware für FPGA von Ettus Firmware Download herunterladen
2. File installieren
3. Firmware in Konsole mit folgenden Befehlen aktualisieren:
   ./usrp_n2xx_net_burner.py –ip=192.168.10.2 –fpga /home/raullen/Desktop/UHD-images-002.20110122035832.cd5631f-Linux/share/uhd/images/usrp_n210_fpga.bin
   ./usrp_n2xx_net_burner.py –ip=192.168.10.2 –fw /home/raullen/Desktop/UHD-images-002.20110122035832.cd5631f-Linux/share/uhd/images/usrp_n2xx_fw.bin

C.3. GNU Radio

Anleitungen und Hinweise zur aktuellen Version können auf der Homepage von GNU Radio nachgelesen werden.

1. Alle benötigten Packete mit folgendem Befehl herunterladen:
   sudo apt-get -y install libfontconfig1-dev libxrender-dev libpulse-dev swig g++
   automake autoconf libtool python-dev libfftw3-dev \ libcppunit-dev libboost-all-dev
   libusb-dev fort77 sdcc sdcc-libraries \
   libsdl1.2-dev python-wxgtk2.8 git-core guile-1.8-dev \
   libqt4-dev python-numpy ccache python-opengl libgsl0-dev \
   python-cheetah python-lxml doxygen qt4-dev-tools \
   libqwt5-qt4-dev libqwtplot3d-qt4-dev pyqt4-dev-tools python-qwt5-qt4
2. Aktuelle Files mit folgendem Befehl in der Konsole auf den Computer kopieren:
   git clone http://gnuradio.org/git/gnuradio.git
3. In “gnuradio“ Ordner wechseln
4. Befehl “./bootstrap“ ausführen
5. Befehl “./configure“ ausführen
6. Befehl “make“ ausführen

7. Befehl “make check“ ausführen um zu überprüfen ob das Projekt einwandfrei erstellt wurde
8. Befehl “sudo make install“ um das erstellte Projekt zu installieren
9. Mit Befehl “gnuradio-companion“ GNU Radio Companion starten und überprüfen ob die UHD Blöcke zur Verfügung stehen.

D. Eigenschaften der erstellten Blöcke

uhd_usrp_source			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	-	-	-
Output	1	Komplex	-
Parameter	31.5	dB	Gain
	2.2736	MHz	Zenterfrequenz
	192.168.10.2	-	Device Addresse
	2	MHz	Sampling Rate

Tabelle D.1.: Parameter uhd_usrp_source Block

interpolation_cc			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Komplex	Original Sequenz
Output	1	Komplex	Linear Interpolierte Sequenz
Parameter	128	-	Interpolation
	125	-	Dezimation

Tabelle D.2.: Parameter interpolation_cc Block

detect_cut_vcc			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Komplex	Sequenz nach dem Interpolierer
Output	FFT Grösse	Komplex	DAB Symbole
	1	Char	1 = DAB Symbol Nummer 0
Parameter	196608	Unsigned Integer	Gesamte Framelänge
	2048	Unsigned Integer	FFT Grösse
	504	Unsigned Integer	Guard Time
	76	Unsigned Integer	Anzahl DAB Symbole in einem DAB Frame

Tabelle D.3.: Parameter detect_cut_vcc Block für den Empfang eines DAB Mode 1 Signals

fft_vcc			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	FFT Grösse	Komplex	DAB Symbol
Output	FFT Grösse	Komplex	FFT des DAB Symbols
Parameter	2048	Unsigned Integer	FFT Grösse
	True	Boolean	Forward FFT
	-	-	Window
	True	Boolean	FFT Shift

Tabelle D.4.: Parameter fft_vcc Block für den Empfang eines DAB Mode 1 Signals

de_diff_mod_vcc			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	FFT Grösse	Komplex	Ergebnis der komplexen FFT
Output	FFT Grösse	Komplex	Demoduliertes DAB Symbol
	1	Komplex	Demoduliertes Signal eines Subcarriers
Parameter	2048	Unsigned Integer	FFT Grösse
	256	Unsigned Integer	Subcarrierwahl für den Plot (FFT Bereich 0-2047)

Tabelle D.5.: Parameter de_diff_mod_vcc Block für den Empfang eines DAB Mode 1 Signals

de_freq_interleaver_vcf			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	FFT Grösse	Komplex	Demoduliertes DAB Symbol
Output	(Anzahl Subcarrier)*2	Float	Frequenz deinterleavte und demappte Daten
Parameter	2048	Unsigned Integer	FFT Grösse
	1536	Unsigned Integer	Anzahl Subcarrier

Tabelle D.6.: Parameter de_freq_interleaver_vcf Block für den Empfang eines DAB Mode 1 Signals

sub_channel_vff			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	(Anzahl Subcarrier)*2	Float	Frequenz deinterleavte und demappte Daten
	1	Char	Signalisierung DAB Symbol 0
Output	(Subchannel Grösse)*8	Float	Subchannel
	288	Float	FIC
Parameter	1536	Unsigned Integer	Anzahl Subcarrier
	188	Unsigned Integer	CU Start Adresse
	84	Unsigned Integer	Subchannel Grösse (in CUs)

Tabelle D.7.: Parameter sub_channel_vff Block für den Empfang von DRS3

time_interleaving_vff			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Subchannel Grösse*8	Float	Subchannel
Output	Subchannel Grösse*64	Float	Subchannel Time Interleaved
Parameter	84	Unsigned Integer	Subchannel Grösse (in CUs)

Tabelle D.8.: Parameter time_interleaving_vff Block für den Empfang von DRS3

depuncturing_vff			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Subchannel Grösse*64	Float	Punktierter Vektor
Output	Alle Teilvektoren + Tail	Float	Wort des Faltungsencoders
Parameter	84	Unsigned Integer	Subchannel Grösse (in CUs)
	11	Unsigned Integer	Puncturing Index Teilvektor 1
	11	Unsigned Integer	Länge des Teilvektors 1
	6	Unsigned Integer	Puncturing Index Teilvektor 2
	21	Unsigned Integer	Länge des Teilvektors 2
	5	Unsigned Integer	Puncturing Index Teilvektor 3
	61	Unsigned Integer	Länge des Teilvektors 3
	7	Unsigned Integer	Puncturing Index Teilvektor 4
	3	Unsigned Integer	Länge des Teilvektors 4

Tabelle D.9.: Parameter sub_channel_vff Block für den Empfang von DRS3

viterbi_vfb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Sequenzlänge	Float	Inputsequenz für Viterbi
Output	Sequenzlänge / 4	Char	Errechnete Sequenz
Parameter	3096	-	Input Sequenzlänge

Tabelle D.10.: Parameter viterbi_vfb Block für die Verarbeitung des FICs

energy_disp_vbb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Vektorlänge	Char	Durchmischte Sequenz
Output	Vektorlänge - 6	Char	Original Sequenz
Parameter	774	Unsigned Integer	Vektorlänge in Bit (inklusive Tail)

Tabelle D.11.: Parameter energy_disp_vbb Block für die Verarbeitung des FICs

crc16_vbb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Vektorlänge	Char	Vektor mit CRC
Output	Vektorlänge - 16	Char	Vektor ohne CRC
Parameter	256	Unsigned Integer	Vektorlänge in Bit (inklusive CRC)

Tabelle D.12.: Parameter crc16_vbb Block für die Überprüfung des FIB CRCs

fib_sink_vb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	240	Char	FIBs

Tabelle D.13.: Parameter fib_sink_vb Block

prearrangement_vbb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	Vektorlänge	Char	Bitvektor
Output	1	Char	Byte Werte
Parameter	(11+21+61+3)*32	Unsigned Integer	Vektorlänge in Bit (Subchannelabhängig)

Tabelle D.14.: Parameter prearrangement_vbb Block für den Empfang von DRS3

blks2_tcp_sink			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Char	-
Parameter	127.0.0.1	String	Adresse
	9999	Unsigned Integer	Port
	True	Boolean	Servermodus

Tabelle D.15.: Parameter blks2_tcp_sink Block für die Übertragung der Daten an einen lokalen Musikplayer

rational_resampler			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Komplex	-
Output	1	Komplex	-
Parameter	128	-	Interpolation
	125	-	Dezimation

Tabelle D.16.: Parameter rational_resampler Block

frame_detecter_cb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Komplex	Sequenz nach dem Resampler
Output	1	Char	1 = Detektion, 0 = keine Detektion
Parameter	2048	Unsigned Integer	FFT Grösse
	504	Unsigned Integer	Guard Time
	76	Unsigned Integer	Anzahl DAB Symbole

Tabelle D.17.: Parameter frame_detecter_cb Block für den Empfang eines DAB Mode 1 Signals

ofdm_symbol_cutter_cc			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Komplex	Datenstrom
	1	Char	Framestart Detektionsbit
Output	1	Komplex	Ausgeschnittene Symbole
	1	Char	Symbol 0 Sample 0 = 1, Else = 0
Parameter	196608	Unsigned Integer	Gesamte Framelänge
	2048	Unsigned Integer	FFT Grösse
	504	Unsigned Integer	Guard Time
	76	Unsigned Integer	Anzahl DAB Symbole
	150	Unsigned Integer	Zeit zwischen Framestartdetektion und Ausschneiden des ersten Symbols

Tabelle D.18.: Parameter ofdm_symbol_cutter_cc Block für den Empfang eines DAB Mode 1 Signals

null_symbol_resample_bb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Char	1 Bit für jeden Sample eines DAB Symbols
Output	1	Char	1 Bit für jedes DAB Symbol
Parameter	2048	Unsigned Integer	FFT Grösse

Tabelle D.19.: Parameter null_symbol_resample_bb Block für den Empfang eines DAB Mode 1 Signals

bonder_vff			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Floar	Achtel des FICs
Output	1	Float	Zusammengesetzter FIC
Parameter	2304/8	Unsigned Integer	Input Vektorlänge
	2304	Unsigned Integer	Output Vektorlänge

Tabelle D.20.: Parameter bonder_vff Block für den Empfang eines DAB Mode 1 Signals

cutter_vbb			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	1	Char	Mehrere FIBs
Output	1	Char	Einzelne FIBs
Parameter	768	Unsigned Integer	Input Vektorlänge
	256	Unsigned Integer	Output Vektorlänge

Tabelle D.21.: Parameter cutter_vbb Block zum Ausschneiden der FIBs

fib_sink2_vb und fib_sink3_vbf			
Art	Wert/Grösse	Datentyp	Eigenschaft
Input	240	Char	FIBs
Output	1	Float	Nur bei fib_sink3_vbf vorhanden

Tabelle D.22.: Parameter des fib_sink2_vb und fib_sink3_vbf Blocks

E. Hardware

USRP N210–Übersicht	
Merkmal	Wert
ADC 100 MS/s 14 Bit	2 Stück
Programmierbare Dezimierungsrate	1
Spurious Free Dynamic Range (Input)	88 dB
DAC 400 MS/s 16 Bit	2 Stück
Programmierbare Interpolationsrate	1
Spurious Free Dynamic Range (Output)	80+ dB
FPGA	Xilinx Spartan 3A-DSP3400
Gigabit Ethernet Interface	1 Anschluss
Steckplatz für RF Daughterboard	1 Steckplatz
Anschluss für Multi-System Betrieb	1 MIMO
TCXO Frequency Reference Anschluss	1
SRAM	1 MB

Daughterboard WBX–Übersicht	
Merkmal	Wert
Frequenzbereich	50 MHz bis 2.2 GHz
PLL lock time	200 us
Unterschiedliche TX und RX Frequenzen wählbar	-
RX Anschluss an USRP Gehäuse	RF1
TX Anschluss an USRP Gehäuse	RF2

Tabelle E.23.: Merkmale der USRP N210 Hardware und des WBX Daughterboards

F. Geräte

Geräte–Übersicht			
Gerät	Hersteller	Modell	Inventar Nummer
AFG3000 Arbitrary Function Generator	Tektronix	AFG3102	NT10.16
Versuchscomputer	Dell	Optiplex 760	DSK-T-5508
Zusatznetzwerkkarte	Intel	EXPI9301CTBLK	-
Autoradio UKW Antenne	-	-	-

Tabelle F.24.: Angaben zu den verwendeten Geräten

G. Programme

Programm–Übersicht		
Programm	Hersteller	Version
Ubuntu	-	10.10 Maverick
Matlab	MathWorks	7.11.0.584 (R2010b)
Simulink	MathWorks	7.6 (R2010b)
GNU Radio Companion	Eric Blossom	3.4.0git-1-g4d1426b8
UHD	Ettus	20110122035832.cd5631f-Linux.tar.gz
gr-howto-write-a-block	Eric Blossom	3.3.0
SciPhy	-	0.9.0
NumPy	-	1.5.1
Geany	-	0.19.1-1
ArbExpress	Tektronix	2.5.2009.10

Tabelle G.25.: Angaben zu den verwendeten Programmen