Entwicklung einer Empfangs-Software für Digital Radio Mondiale

Eine Studienarbeit von Burkart Lingner im Studiengang
Elektronik und Informationstechnik an der HTW Aalen

Sommersemester 2008

Betreuer: Prof. Dr. Manfred Horn

Aalen, den 31.10.2008

Inhaltsverzeichnis

1 Einleitung

1.1 Standards für digitalen terrestrischen Hörfunk

Im Zuge der allgemeinen Umstellung der Rundfunksysteme von analoger auf digitale Übertragungstechnik in den letzten Jahren wurden verschiedene Standards für einen digitalen Hörfunk erarbeitet. In Deutschland ist zurzeit Digital Audio Broadcasting (DAB) und Digital Video Broadcasting – Terrestrial (DVB-T) am Bekanntesten – letzteres aber trotz der technischen Möglichkeiten weniger für die Übertragung von Hörfunk als von digitalem Fernsehen. DAB soll auf lange Sicht den analogen UKW-Rundfunk ablösen. Mit Digital Radio Mondiale (DRM) ist ein Standard für digitalen Rundfunk über Kurz-, Mittel- und Langwelle entstanden. Weitere weniger bekannte Standards sind HD-Radio, FMeXtra und ISDB-Tsb.

1.2 Kanaleigenschaften

DRM wird auf Lang-, Mittel- und Kurzwelle eingesetzt, also im Bereich unter 30 MHz. Während auf Langwelle mithilfe der Bodenwelle übertragen wird, zeichnen sich Mittel- und Kurzwelle dadurch aus, dass sie sowohl über die Bodenwelle als auch über die Raumwelle übertragen können und dabei große Reichweiten überbrücken [STOT01]. Dies liegt insbesondere daran, dass die Raumwelle an der Ionosphäre reflektiert wird. Selbst mehrfache Reflexion an Ionosphäre und Erdoberfläche ist möglich, was zu Reichweiten von mehreren tausend Kilometern führen kann. Die Reflektionsfähigkeit der Ionosphäre hängt stark von der Sonneneinstrahlung ab, weshalb nachts die größte Reichweite erreicht wird. Auch eine Abhängigkeit vom Sonnenfleckenzyklus mit 11,6 Jahren Dauer sowie vom 27-tägigen Rhythmus der Eigenrotation der Sonne [MEYE08, 340-343].

Dadurch, dass sich die Ionosphäre auf und ab bewegt, kommt es bei Übertragungen per Raumwelle unerwünschterweise auch zu Doppler-Verschiebungen des übertragenen Frequenzspektrums.

Ein weiterer unerwünschter Effekt ist der Mehrwegeempfang. Hierbei gelangt ein ausgestrahltes Signal durch Reflexion oder Streuung auf mehreren Wegen vom Sender zum Empfänger. Dort überlagern sich dann die unterschiedlich verzögerten und abgeschwächten Signale.

Ein Spezialfall des Mehrwegeempfangs stellen Gleichwellennetze dar. Hier wird von mehreren Sendern aus zum gleichen Zeitpunkt das gleiche Signal ausgestrahlt. Wie Abschnitt 1.3.1 zeigen wird, ist das bei Digital Radio Mondiale verwendete Modulationsverfahren OFDM gut geeignet, mit Mehrwegeempfang umzugehen.

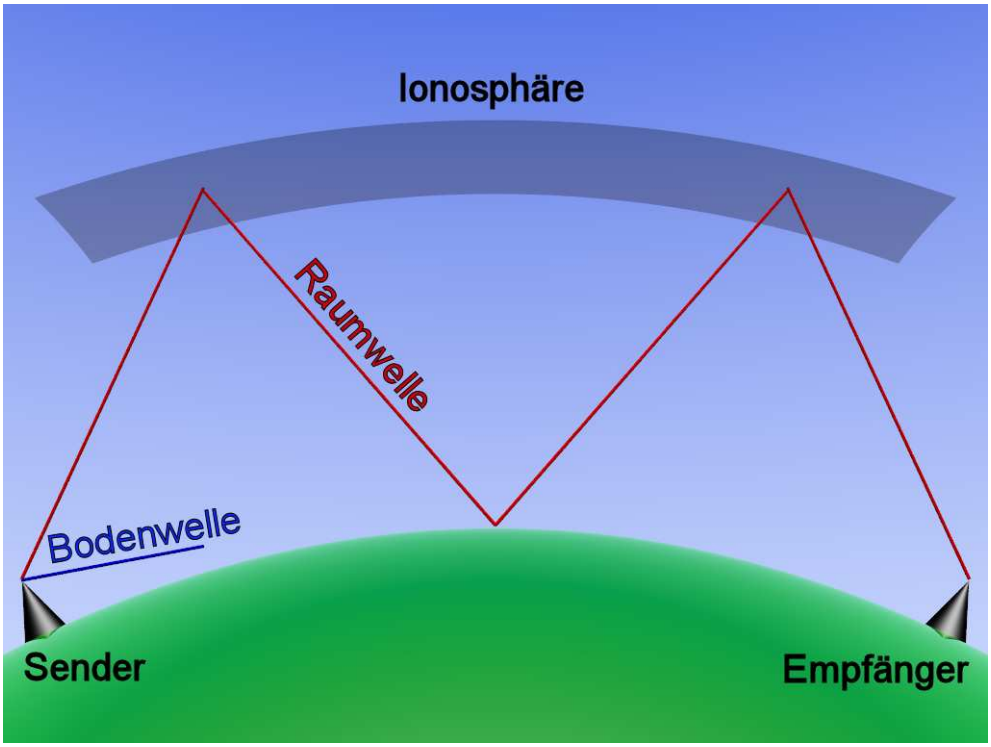

Abbildung 1: Raumwelle mit Reflexion an der Ionosphäre [WIKI07]

1.3 Orthogonal Frequency Division Multiplexing

Orthogonal Frequency Division Multiplexing (OFDM) ist ein sehr bandbreiteneffizientes Verfahren zur Datenübertragung. Die Grundidee ist, einen Datenstrom auf mehrere Kanäle mit unterschiedlicher Frequenz aufzuteilen, zu modulieren und dann zu übertragen. Werden die Frequenzen so gewählt, dass sie um

$$f_u = \frac{1}{T_u}$$

auseinander liegen, mit T_u als der Integrationszeit des Signals („*useful part*“, siehe 1.3.1), heißen sie orthogonal. Die einzelnen Subträger sind definiert als

$$\Psi_k(t) = e^{j2\pi kt/T_u}$$

mit k als ganzzahligem Subträger-Index. Mit dem komplexen, QAM-modulierten Zellwert $c_k(t)$ ergibt sicht im Basisband das Zeitsignal

$$x_{OFDM}(t) = \sum_{k=k_{\min}}^{k_{\max}} c_k(t) \cdot \Psi_k(t).$$

Das Spektrum jedes QAM-Signales hat einen sin(x)/x-Verlauf, mit dem Maximum in $k \cdot f_0$ und Nullstellen links und rechts im Abstand $n \cdot f_0$.

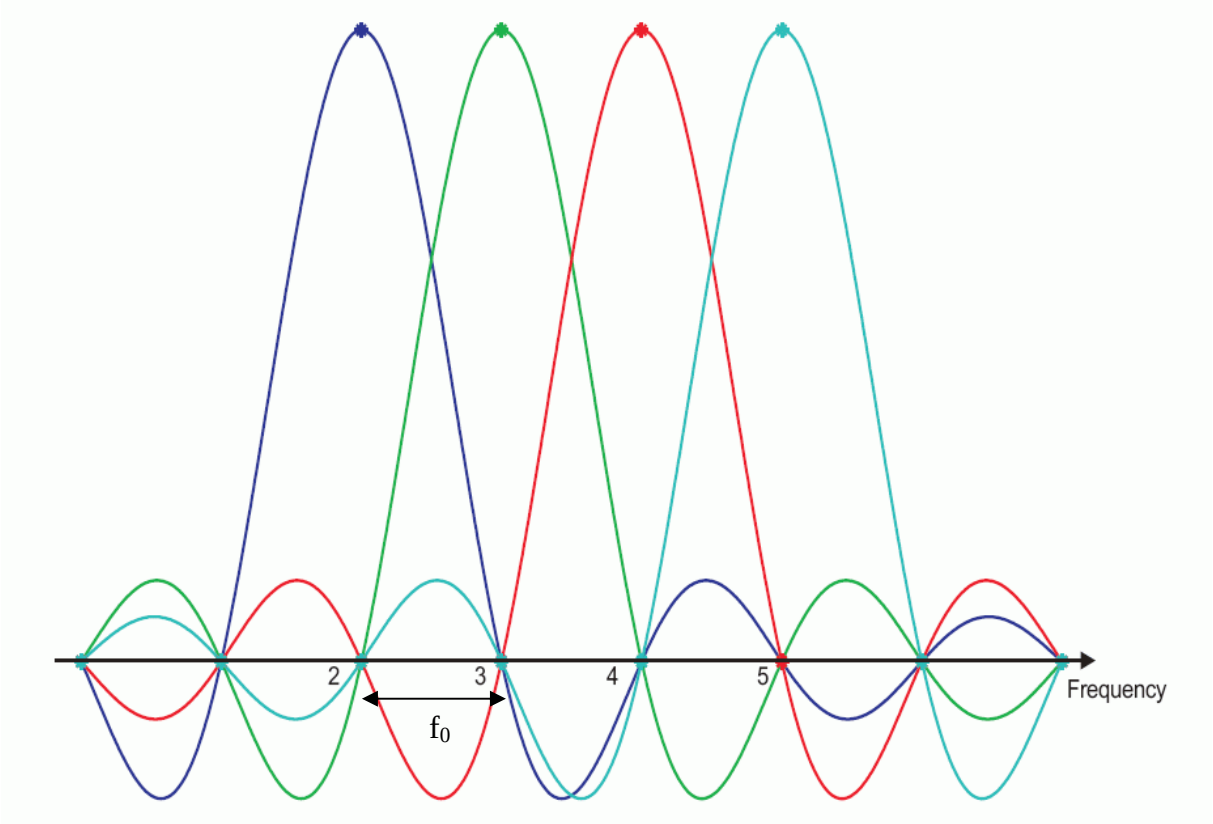

Abbildung 2: Leistungsspektren orthogonaler QAM-Signale [HAJJ07]

Da es sich um digitale Signale handelt, ist die Zeitvariable t diskret. Somit wird die Berechnung von $x_{OFDM}(t)$ identisch zur inversen diskreten Fourier-Transformation (IDFT), für die mit der schnellen Fourier-Transformation (Fast Fourier Transform, FFT) ein effizienter Berechnungs-Algorithmus zur Verfügung steht [MEYE08, 223f.].

1.3.1 Schutzintervall

Würde man mehrere auf diese Weise erzeugte OFDM-Symbole hintereinander übertragen, ergäbe sich im Empfänger zwangsläufig eine Überlagerung verschiedener Symbole aufgrund unterschiedlich verzögerter Pfade von Sender zu Empfänger. Abbildung 3 veranschaulicht

den Fall mit zwei unterschiedlich verzögerten Pfaden. Bei den Überlappungen unterschiedlicher Symbole ergibt sich Inter-Symbol-Interferenz (ISI). An der Stelle, wo sich zwei gleiche Symbole mit unterschiedlicher Verzögerung überlappen, addieren sie sich je nach Phasenlage der Subträger oder löschen sich aus. Auf Grund der Linearphasigkeit der Laufzeitwirkung kann dies aber niemals bei allen Trägern gleichzeitig der Fall sein [OHM07].

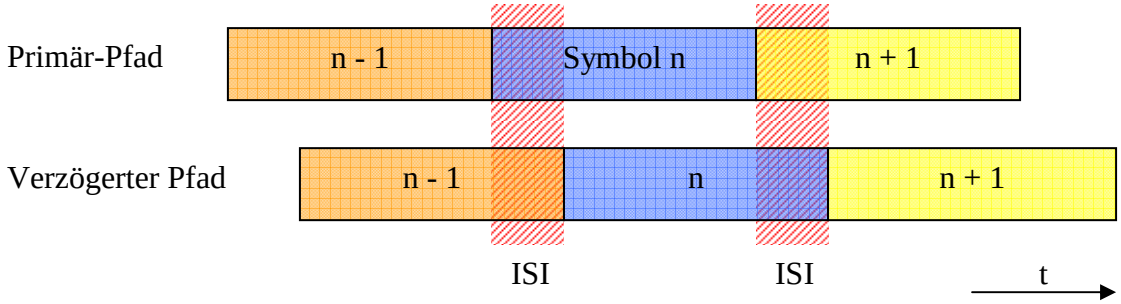

Abbildung 3: Inter-Symbol-Interferenz durch verzögerten Pfad, nach [STOT98]

Zur Vermeidung von ISI wird ein Schutzintervall verwendet (*guard interval*, auch *cyclic extension*). Das Schutzintervall wird zeitlich vor dem eigentlichen Symbol übertragen und beinhaltet die Daten vom Ende des Symbols. Es treten also zwischen Schutzintervall und Symbol keine Phasensprünge auf.

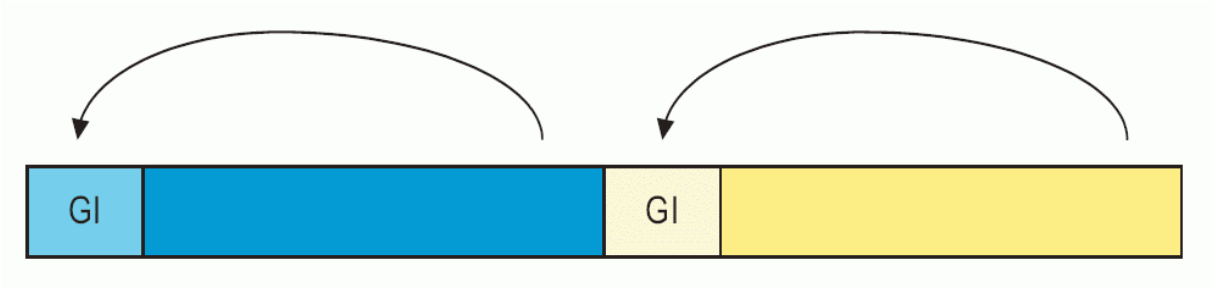

Abbildung 4: Schutzintervall (guard interval, GI) [HAJJ07]

Überlagern sich nun zwei unterschiedlich verzögerte OFDM-Symbole mit Schutzintervall, wobei der Zeitunterschied kürzer ist als die Länge des Schutzintervalls, tritt keine ISI auf. Je länger das Schutzintervall ist, umso größere Verzögerungen sind tolerierbar.

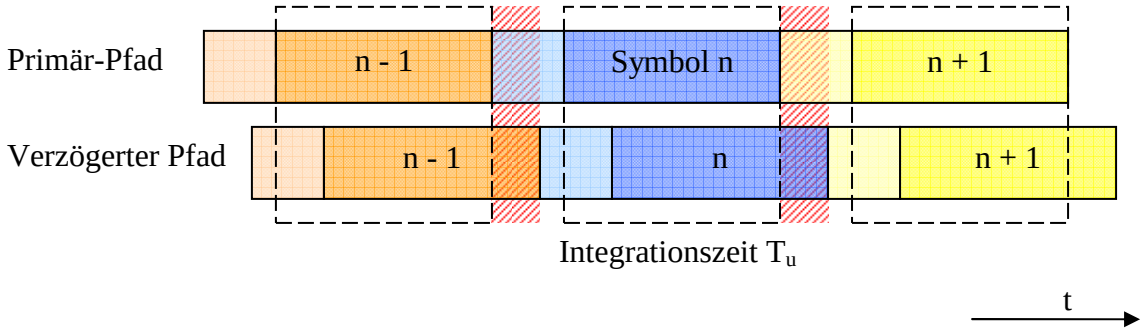

Abbildung 5: Keine ISI durch Schutzintervall

Dadurch, dass in dem für DRM verwendeten Kanal viele Verzögerungen auftreten, ist der Einsatz eines ausreichend bemessenen Schutzintervalls unabdingbar. Weiterhin ermöglicht es den Betrieb von Gleichwellennetzen, wie in Absatz 1.2 beschrieben.

1.3.2 Crest-Faktor

Ein gravierender Nachteil von OFDM ist dessen hoher Crest-Faktor. Dieser ist definiert als der Quotient des Spitzenwertes und des rms-Wertes. Für einen unmodulierten Träger lautet der Crest-Faktor 3 dB. Eine ähnliche Bewertung gelingt mit der *peak-to-average power (PAP) ratio*. Die Spitzenleistung (*peak power*) ist definiert als die Leistung einer Sinusschwingung mit einer Amplitude gleich dem Maximalwert der Einhüllenden. Ein unmodulierter Träger hat entsprechend eine *PAP ratio* von 0 dB. Für ein OFDM-Signal mit N Subträgern beträgt die maximale *PAP ratio* N. Für DRM liegt N je nach Betriebsmodus zwischen 88 und 458.

In der Praxis führt eine hohe *PAP ratio* dazu, dass der Sender für deutlich höhere als die Durchschnittsleistung ausgelegt sein muss.

In [NEE00] werden verschiedene Verfahren zur Abmilderung des „*Peak Power Problems*“ vorgeschlagen: Abschneiden (*clipping*) der Signalspitzen, spezielle Kanalcodes, die OFDM-Symbole mit hohe *PAP ratio* vermeiden sowie das gezielte Verwürfeln (*scrambling*) der Daten mit anschließender Anwendung der effektivsten Verwürfelung.

Per Digital Radio Mondiale versandte Signale werden in deterministischer Weise verwürfelt. In [ETSI08, 106f.] ist das damit begründet, dass ungewollte Regelmäßigkeiten im übertragenen Signal vermieden werden sollen. Ob hiermit das Problem der hohen *PAP ratio* gemeint ist, wird nicht konkret erwähnt. Da es sich um ein festgelegtes, in keiner Weise von den versendeten Daten abhängiges Verfahren handelt, scheint die Effektivität hinsichtlich *PAP ratio* zumindest fragwürdig.

1.4 Synchronisationsfehler

OFDM ist gegenüber Einträgersystemen hinsichtlich der Synchronisation empfindlich. Aufgrund der kohärenten Demodulation und um eine Beeinträchtigung des Signal-Rausch-Abstands durch ISI oder Inter-Carrier-Interferenz (ICI) zu verhindern, spielt die korrekte Synchronisation eine wichtige Rolle.

1.4.1 Phasensynchronisation

Eine fehlende Phasensynchronisation bezieht sich auf den Lokaloszillator zur Erzeugung einer Zwischenfrequenz bzw. der Generierung eines äquivalenten Tiefpasssignals. Fehler können dadurch auftauchen, dass die Phase von Empfänger und Sender nicht übereinstimmen oder dadurch, dass die Phase des Oszillators *Jitter* aufweist.

Laut [Nee00, 75] ergeben sich hieraus zwei Effekte. Zum Einen eine zufällige, für alle Subträger gleiche Phasendrehung, zweitens ICI. Die ICI wird dadurch hervorgerufen, dass die Subträger nicht mehr genau im f_u-Frequenzraster angeordnet sind.

1.4.2 Frequenzsynchronisation

Synchronisationsfehler führen entweder zu ISI oder zu ICI. Beides bewirkt letztlich einen verschlechterten Signal-Rausch-Abstand.

Eine schlechte Synchronisation auf die Trägerfrequenz hat zur Folge, dass die Subträger im Basisband nicht mehr auf die Frequenz $k \cdot f_u$ zu liegen kommen. Somit ist die Orthogonalität verletzt und es ergibt sich ICI.

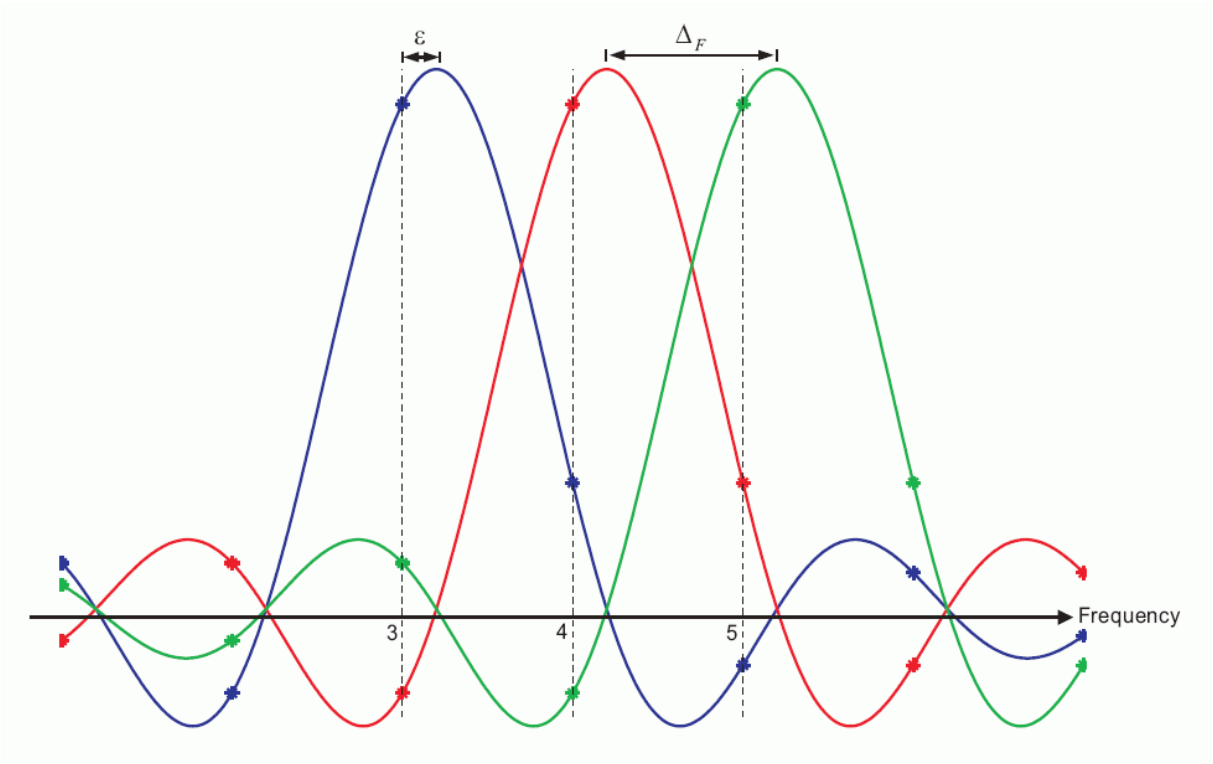

Abbildung 6: Um ε falsche Frequenzsynchronisation [HAJJ07]

In [NEE00, 77f.] wird die Verschlechterung des Signal-Rausch-Abstands mit

$$D \cong \frac{10}{3\ln 10}(\pi \varepsilon T_u)^2 \frac{E_S}{N_0}$$

abgeschätzt. Für eine vernachlässigbare Verschlechterung um 0,1 dB dürfe die Frequenzabweichung ε maximal 1% des Subträger-Abstandes f_u betragen. Für DRM mit Subträger-Abständen von 41,67 Hz für Mode A bis 107,14 Hz für Mode D entspricht das einem maximal tolerierten ε von 0,4 bis 1,1 Hz.

1.4.3 Zeitliche Synchronisation

Hinsichtlich der zeitlichen Synchronisation ist OFDM ziemlich robust. Durch die Erweiterung um das Schutzintervall ist es möglich, die Symbol-Integration um die Länge des Schutzintervalls T_g zu verschieben. Wird dieser Spielraum tatsächlich ausgenutzt, geht allerdings der Nutzen des Schutzintervalls verloren – die Vermeidung von ISI bei verzögert eintreffenden Echos. Der optimale Zeitpunkt für die Symbol-Integration ist daher direkt am Ende des Schutzintervalls.

Laut [NEE00, 79] und [FAZE03, 135] besteht ein Zusammenhang zwischen der Phase φ_k des Subträgers k und der Abweichung von der perfekten zeitlichen Synchronisation τ gemäß $\varphi_k = 2\pi f_k \tau$.

Im Konstellationsdiagramm eines betroffenen Symboles wirkt sich der Timingfehler dementsprechend als Rotation um Vielfache von $2\pi f_u \tau$ aus.

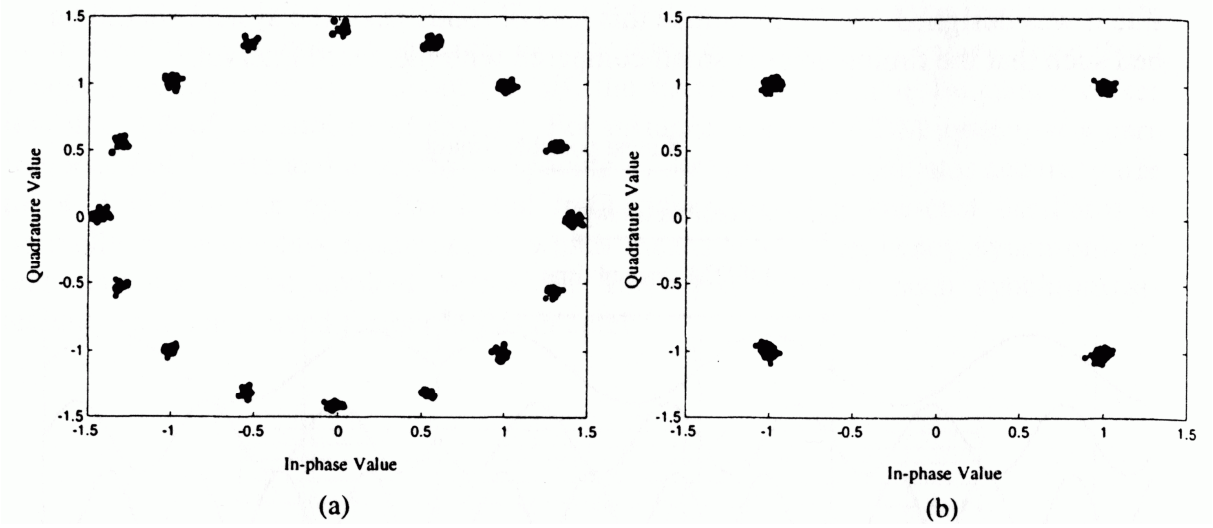

Abbildung 7: Konstellationsdiagramm mit Timingfehler (a) und ohne (b) [NEE00]

Dieser Zusammenhang lässt sich ausnutzen, um mit Hilfe von Pilotzellen, deren Soll-Phasenlage bekannt ist, auf den Fehler der zeitlichen Synchronisation zu schließen und ihn dann zu korrigieren. Der zeitliche Fehler beträgt

$$\tau = \frac{\varphi_k \cdot T_u}{2\pi \cdot k}.$$

1.5 Software Defined Radio

Gemäß [JOND02, 25] versteht man unter einem Software (Defined) Radio „einen Transmitter, dessen Funktionen so weit wie möglich als Programme auf einem Rechner ablaufen.“ Neben der Besonderheit, mit Digitalrechnern statt mit analogen Bausteinen Funkstandards umzusetzen, wird hier Wert darauf gelegt, über einfache Anpassung der Übertragungsparameter verschiedene Standards einer Familie zu unterstützen.

Neben dieser Anpassbarkeit zeichnen sich Software Defined Radios durch die gleichen Vorteile aus wie die allgemeine digitale Signalverarbeitung: Mangels toleranzbehafteter elektronischer Bauteile eine sehr gute Reproduzierbarkeit und Genauigkeit.

2 Aufbau des Digital-Radio-Mondiale-Signals

2.1 Übertragungsbandbreite

Je nach ITU-Region belegt ein Kanal auf Lang- und Mittelwelle eine Bandbreite von 9 oder 10 kHz. Auf Kurzwelle sind weltweit 10 kHz Bandbreite vorgesehen.

DRM kann sowohl mit einem Kanalraster von 9 als auch von 10 kHz betrieben werden. Weiterhin ist es möglich, jeweils die halbe, ganze oder doppelte Bandbreite zu belegen, d.h. es kann mit 4,5 kHz, 5 kHz, 9 kHz, 10 kHz, 18 kHz und 20 kHz übertragen werden [ETSI08, 131f.]. Hierdurch ist es möglich, im Simulcast-Betrieb ein DRM-Signal und parallel ein analoges Ein- oder Zweiseitenband-AM-Signal zu übertragen [ETSI08, 181-183]. Diese Übertragungsart erleichtert den Umstieg auf DRM, solange noch nicht flächendeckend moderne, digitale Empfänger zur Verfügung stehen. Um Qualitätseinbußen für die Nutzer analoger Empfangsgeräte in Grenzen zu halten, muss das DRM-Signal mit 6 bis 16 dB weniger Leistung übertragen werden als das analoge [ETSI06, 7]. Alternativ kann ein modifiziertes und abgeschwächtes DRM-Signal parallel zu einem analogen über denselben Kanal übertragen werden [ETSI06].

2.2 Übertragungsmodi (robustness modes)

Für DRM sind vier verschiedene Übertragungsmodi, genannt *robustness modes*, definiert. Mode A bietet die höchste Kapazitat, ist aber auch am Empfindlichsten für Störungen und eignet sich daher insbesondere für die lokale Versorgung. Über Mode B und C bis hin zu Mode D sinkt die Datenrate kontinuierlich, während sich das Störungsverhalten verbessert.

Dies wird dadurch erreicht, dass für die besser geschützten *robustness modes* die Nutzteil-Dauer der Symbole verkürzt wird, was automatisch den Abstand zwischen den Subträgern erhöht. Außerdem wird die Länge des Schutzintervalls verlängert und es werden anteilig mehr Pilotzellen eingestreut.

2.3 Quellcodierung

Um mit der geringen Datenrate des DRM-Signals von 4,8 kbit/s bis 72 kbit/s [ETSI08, 167] ein Audiosignal in bestmöglicher Qualität zu übertragen, wird dieses einer Quellencodierung unterzogen. Hierbei wird die Redundanz des Eingangsstroms soweit als möglich entfernt. Die Spezifikation sieht das Codierungsverfahren MPEG-4 Advanced Audio Coding (AAC) für

Audiosignale aller Art inclusive Musik vor sowie die Verfahren MPEG CELP und MPEG-4 HVXC (Harmonic Vector eXcitation Coding) für Sprachsignale. Während die letztgenannten Codecs bei reinen Sprachsendungen, wie sie auf Kurz- und Mittelwelle häufig auftreten, eine besonders gute Effizienz aufweisen, bieten sie bei Musiksendungen oder auch der Übertragung von Jingles u.ä. keinen befriedigenden Hörgenuss. Aus diesem Grund wird als Quellcoder vorwiegend AAC eingesetzt.

Eine weitere Verbesserung der Klangqualität bei niedriger Bitrate wird durch den Einsatz von *Spectral Band Replication* (SBR) erreicht. Da das menschliche Gehör für höhere Frequenzen weniger empfindlich ist als für niedrige, wird bei diesem Verfahren nur der untere, wichtigere Frequenzbereich direkt mit z.B. AAC codiert. Der Bereich höherer Frequenzen wird mit Hilfe sehr kompakter Zusatzinformationen aus den tiefen Tönen errechnet. Dies ist möglich, weil in der Regel ein Zusammenhang zwischen hohen und tiefen Tönen besteht, z.B. in Form von Oberwellen. SBR kann gemäß der DRM-Spezifikation mit allen der drei oben genannten Quellcodern verwendet werden. Zum Decodieren sind zwei Verfahren vorgesehen, die entweder mit komplexen oder mit reellwertigen Filterbänken arbeiten und dementsprechend eine hohe Klangqualität oder eine gute Recheneffizienz ermöglichen sollen.

Für AAC + SBR wird mit *Parametric Stereo (PS) coding* eine weitere Möglichkeit für bessere Klangqualität bei niedrigen Übertragungsraten zur Verfügung gestellt. Bei diesem Verfahren wird ein gewöhnlich codiertes Mono-Signal zusammen mit kompakten Stereo-Informationen übertragen. So wird eine hohe Qualität für das Mono-Signal garantiert. Das Verfahren soll daher auch insbesondere dann verwendet werden, wenn die Übertragungsrate für ein vollwertiges Stereo-Signal nicht zur Verfügung steht.

Laut DRM-Konsortium bietet DRM dicht an UKW-Radio reichende Audioqualität. Um diesen Anspruch zu untermauern, stellt das Konsortium auf seiner Website einige Hörbeispiele von Übertragungen über teils mehrere tausend Kilometer zur Verfügung [DRMC08]. Die mit 16 bis 25 kbit/s übertragenen DRM-Signale klingen subjektiv deutlich besser als per konventioneller Zweiseitenband-AM übertragene.

2.4 Kanalmultiplex (FAC, SDC, MSC)

Per DRM werden Daten über drei logische Kanäle übertragen: *Fast Access Channel* (FAC), *Service Description Channel* (SDC) und *Main Service Channel* (MSC).

Der Fast Access Channel wird vom Empfänger zuerst ausgewertet. FAC-Zellen sind immer im Bereich von DC bis +4,5 kHz enthalten, es wird zum Empfang also keine Kenntnis über die verwendete Bandbreite vorausgesetzt. Die Daten im FAC sind per 4-QAM moduliert, mit einem Faltungscode der Rate R = 0,6 codiert und mit einer Prüfsumme gesichert. Alle 400 ms werden die Daten des FAC wiederholt. All dies ermöglicht auch unter schlechten Bedingungen einen schnellen und fehlerfreien Empfang.

Im FAC wird unter anderem die verwendete Bandbreite, die Position des Rahmens im *transmission super frame* (siehe 2.5) sowie die QAM-Konstellationen für SDC und MSC übertragen. Außerdem enthält der FAC auch Informationen über das gesendete Programm: Typ und Sprache der Aussendung.

2.5 Rahmenstruktur

Oberhalb der OFDM-Symbole existieren zwei Abstraktionsebenen. Je nach *robustness mode* werden 15 bis 24 Symbole zu einem *transmission frame* gruppiert. Diese *transmission frames* sind unabhängig vom *robustness mode* jeweils 400 ms lang. Drei *transmission frames* ergeben wiederum einen *transmission super frame* der Länge 1200 ms.

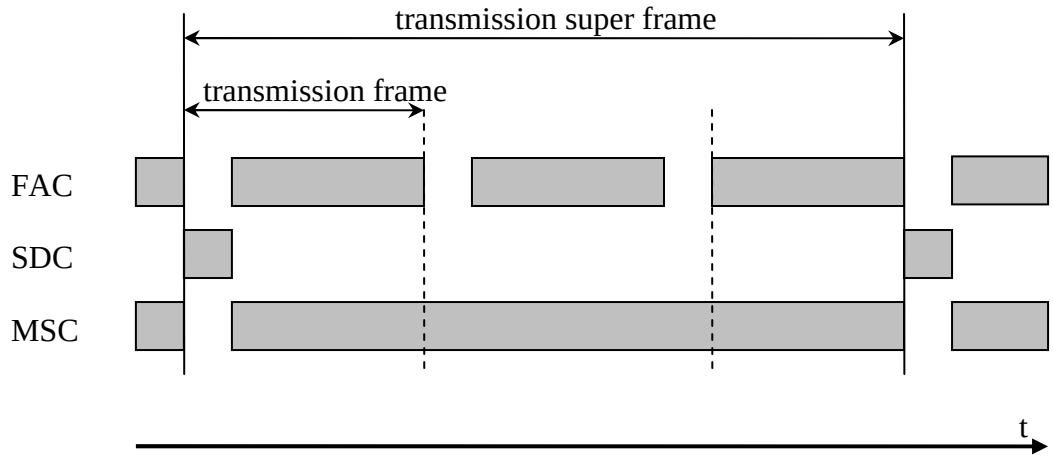

Abbildung 8: Kanalmultiplex und Rahmenstruktur

Die in Abhängigkeit vom *robustness mode* ersten zwei bis drei Symbole eines *transmission super frames* enthalten den SDC. Die Position eines Rahmens innerhalb des *transmission super frames* wird im FAC übertragen, der in jedem *transmission frame* erneut komplett gesendet wird. Die übrigen Zellen entfallen auf die Piloten und den MSC mit den eigentlichen Daten.

2.6 Kanalcodierung und Interleaving

Eine direkte Übertragung von digitalen Daten über einen Funkkanal ist nicht möglich, da unweigerlich Fehler zu gekippten Bits am Empfänger führen würden. Durch das Hinzufügen von Redundanz wird es dem Empfänger ermöglicht, bei nicht zu großer Anzahl an Fehlern wieder auf die ursprünglich gesendete Bitfolge zurückzuschließen.

Bei DRM wird ein 1/4-Faltungscode der Ordnung K = 7 verwendet. Ein Bit am Eingang erzeugt vier Bit am Ausgang, wobei sich jedes Eingangsbit auf die nachfolgenden $4 \cdot K = 28$ Ausgangsbits auswirkt. Die Generatorpolynome lauten 133_8, 171_8, 145_8 und 133_8.

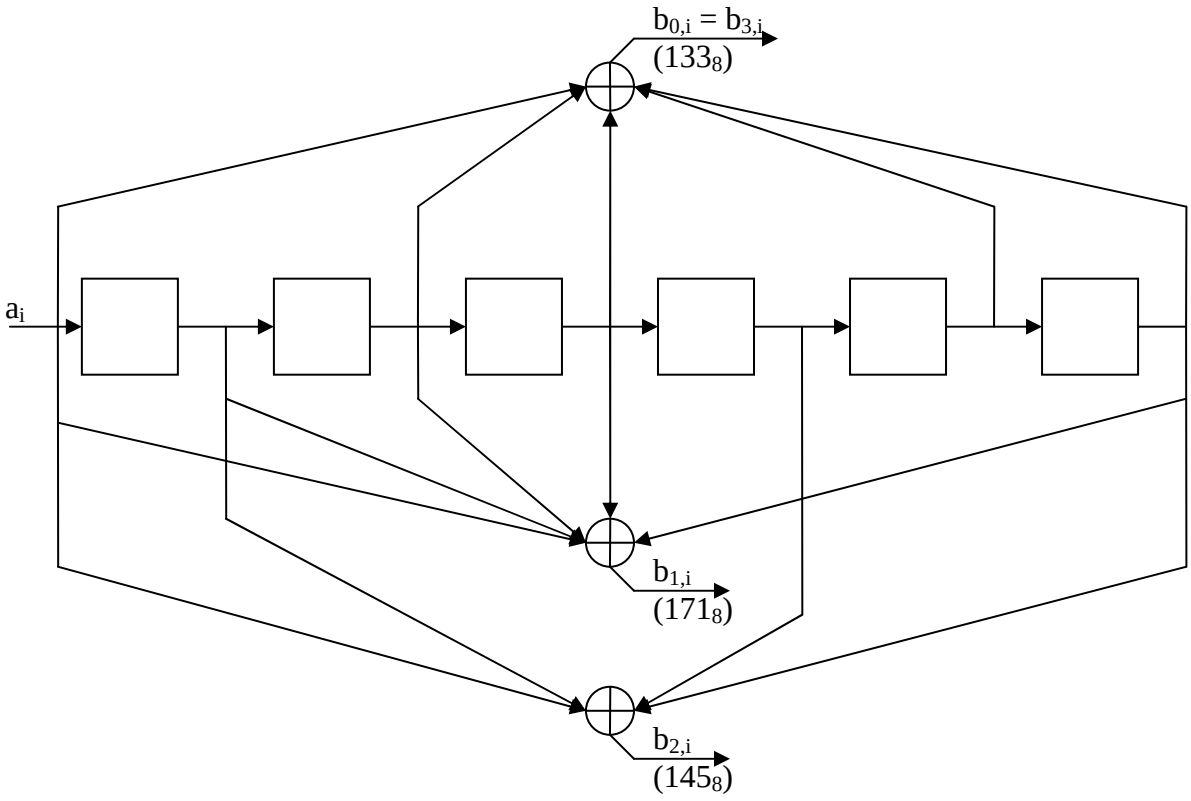

Abbildung 9: Faltungsencoder

Anhand des obigen Codes werden für unterschiedlich stark zu codierende Teile des Datenstroms verschieden punktierte Codes verwendet. Hierzu werden nach einem festgelegten Schema regelmäßig Bits am Encoderausgang übersprungen. Aufbauend auf dem beschriebenen *mother code* werden so Coderaten von 0,25 bis 0,889 erreicht.

Da Empfangsstörungen meist nicht punktuell sondern als Bündelfehler auftreten, also mehrere Bits hintereinander gestört sind, wird ihre Reihenfolge vom *Interleaver* deterministisch verwürfelt [ETSI08, 117-119]. Entstehen während der Übertragung Bündelfehler, werden sie

durch den *De-Interleaver* im Empfänger so verschoben, dass sie für den Decoder als Einzelfehler erscheinen.

2.7 QAM-Konstellation der Subträger

Die einzelnen OFDM-Zellen bzw. Subträger werden mit 4-QAM, 16-QAM oder 64-QAM moduliert und tragen dementsprechend zwei, vier oder sechs Bit. Die FAC-Zellen werden immer mit 4-QAM moduliert, um den größtmöglichen Fehlerschutz zu gewährleisten. Für den SDC wird 4- oder 16-QAM verwendet, für den MSC 16- oder 64-QAM.

2.8 Pilotzellen

Zu Synchronisationszwecken und für die Kanalschätzung sind verschiedene Pilotzellen in die OFDM-Symbole eingestreut. Die Piloten haben in der Regel eine um den Faktor $\sqrt{2}$ höhere Amplitude als die Datenzellen. Dies senkt zwar das Signal-Rausch-Verhältnis für die Datenzellen, stellt jedoch sicher, dass die wichtigen Pilotzellen gut empfangen werden. Die Piloten am Rand der Übertragungsbandbreite werden gegenüber Datenzellen um den Faktor 2 verstärkt, um dem abflachenden Spektrum entgegenzuwirken.

Bei den Frequenzen 750 Hz, 2250 Hz und 3000 Hz liegen die drei Frequenzreferenzen. Sie sind in jedem Symbol und in jedem *robustness mode* vorhanden. Sie wurden so gewählt, dass sie über alle Symbole hinweg kontinuierlich sind. Die Frequenzreferenzen dienen der Erkennung eines vorhandenen Signals und zur Bestimmung von Frequenzabweichungen. Außerdem können sie zur Kanalschätzung verwendet werden [ETSI08, 133].

Eine weitere Gruppe von Pilotzellen sind die Zeitreferenzen. Sie sind im ersten Symbol jedes *transmission frames* vorhanden und werden entsprechend zur Rahmensynchronisation verwendet. Darüber hinaus eignen sich die Zeitreferenzen zur Ermittlung von Frequenzabweichungen. Laut [ETSI08, 134] fungieren die Frequenzreferenzen gleichzeitig auch als Zeitreferenzen. Inwieweit sie die Rahmensynchronisation tatsächlich unterstützen können, wird in Kapitel 3.6 diskutiert.

Die letzte Art von Referenzzellen stellen die *gain references* dar. Sie werden zur Kanalschätzung verwendet, die für eine kohärente Demodulation erforderlich ist. Je nach *robustness mode* ist im Frequenzbereich jede zweite bis fünfte Zelle eine *gain reference*. Je enger die Piloten angeordnet sind, desto genauer kann die Kanalschätzung durchgeführt werden, desto weniger Bandbreite bleibt aber für die eigentlichen Nutzdaten.

In Abbildung 10 ist die Verteilung der Pilotzellen sowie der FAC-Zellen für *robustness mode* A und Bandbreite 4,5 kHz veranschaulicht. Frequenzreferenzen sind mit „f“, *gain references* mit „o“ und FAC-Zellen mit „x“ dargestellt.

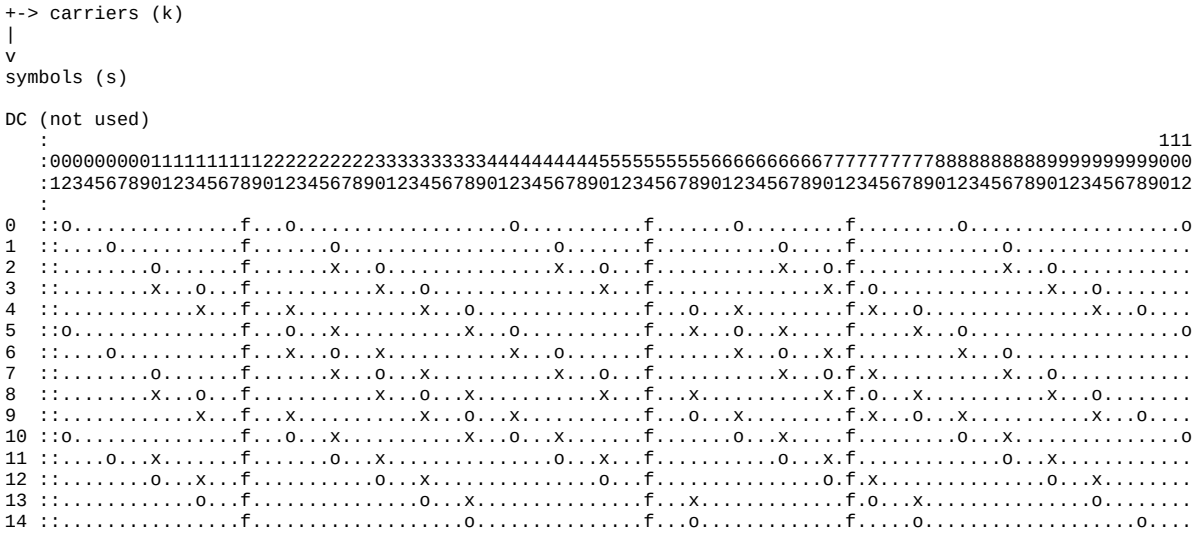

Abbildung 10: Pilotzellen im robustness mode A [ETSI08, 186]

3 Die Empfangssoftware

Im Rahmen dieser Studienarbeit ist eine Software entstanden, die ein DRM-Signal einliest und verarbeitet. Der ursprüngliche Plan, das Programm direkt auf einem DSP-Board ablaufen zu lassen, wurde zugunsten einer im Entwurfsstadium einfacher zu handhabenden PC-Software aufgegeben. Der Quellcode kann mit der kostenlos erhältlichen Microsoft Visual C++ 2008 Express Edition compiliert werden. Da es sich um Standard-C handelt, sollte eine Überführung auf einen DSP ohne größere Probleme zu bewerkstelligen sein.

Die Software führt eine Quadraturmischung des Eingangssignals aus, durchläuft verschiedene Synchronisationsstufen, um am Ende die Daten des FAC auszulesen. Sobald aus der ablaufenden Synchronisation die nötigen Informationen zur Verfügung stehen, wird in diesem Zuge eine Seriell-Parallel-Wandlung mittels DFT sowie eine Kanalschätzung vorgenommen. Die FAC-Daten werden nach dem Demapping einem Hard Decision Viterbi-Decoder zugeführt.

Im weiteren Verlauf müsste der SDC und dann der MSC mithilfe der aus dem FAC gewonnenen Parameter ausgewertet werden. Diese weiterführenden Schritte sind wegen zeitlicher Beschränkungen nicht in dieser Studienarbeit durchgeführt worden.

3.1 Das Eingangssignal

Zum Testen des Empfangs liest die Software eine WAV-Datei ein, die ein DRM-Signal enthält. Diese WAV-Datei wurde vorab mit der Software Spark 1.7.1 von Michael Feilen generiert. Es handelt sich um ein Monosignal mit 48 kHz Samplerate, das auf einer Zwischenfrequenz von 12 kHz ein DRM-Signal im *robustness mode* A enthält.

Das Programm sucht automatisch nach einer Datei namens low_q.wav. Soll eine andere Datei eingelesen werden, muss der Dateiname beim Programmaufruf als Parameter übergeben werden.

3.2 Quadraturmischung

Für die weitere Verarbeitung muss das Signal als Basisbandsignal vorliegen. Hierzu wird das äquivalente Tiefpasssignal gebildet.

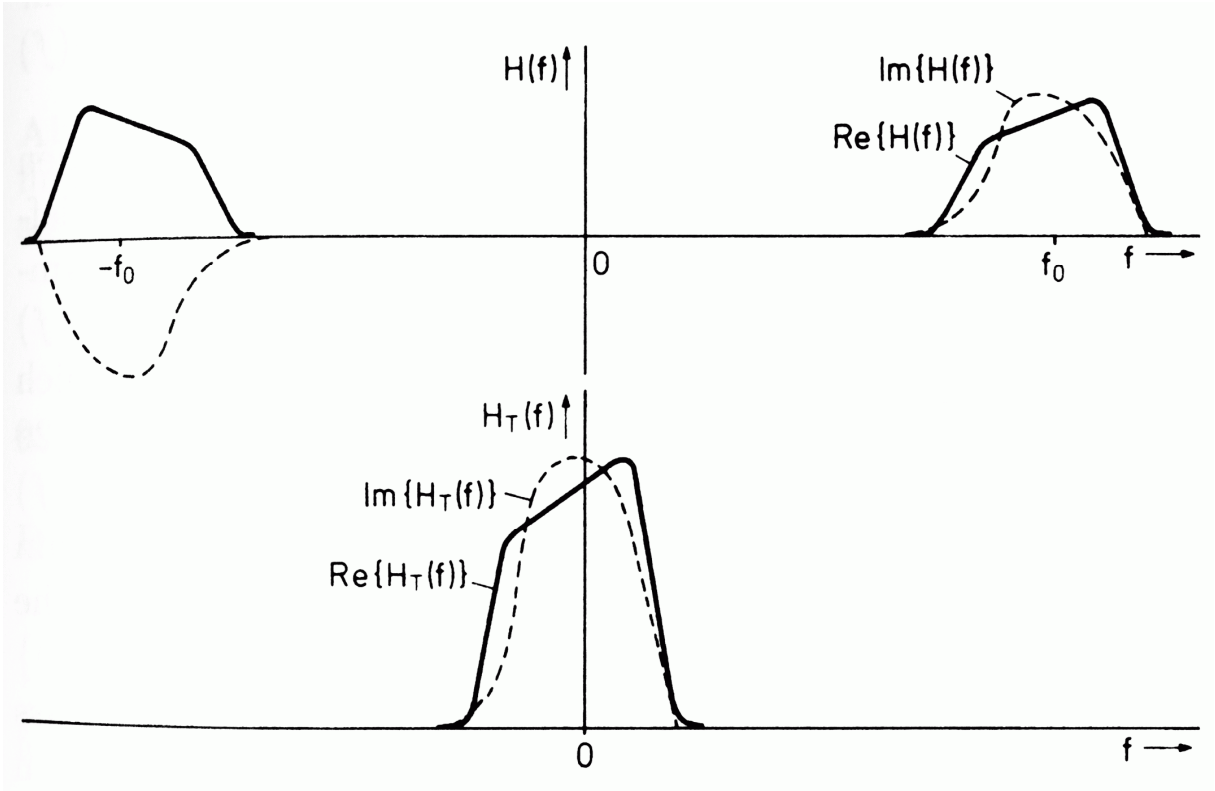

Abbildung 11: Real- und Imaginärteil eines Bandpasssignals H(f) und seines äquivalenten Tiefpasssignals $H_T(f)$ [OHM07]

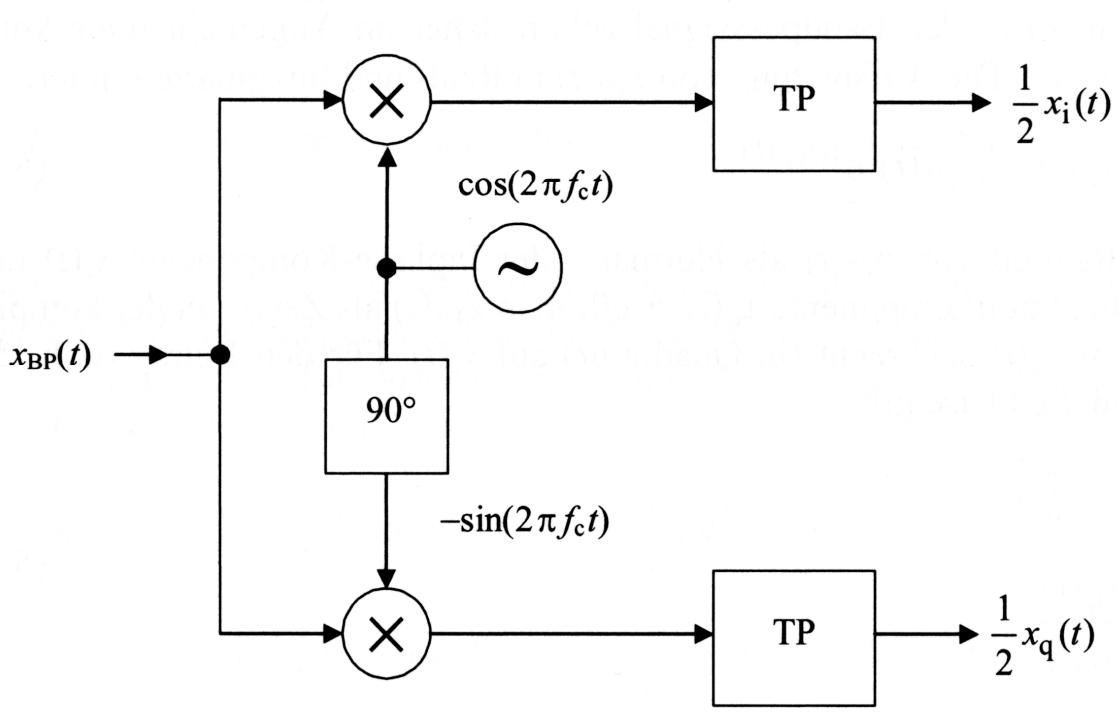

Abbildung 12: Erzeugung des äquivalenten Tiefpasssignals aus dem Bandpasssignal [ROPP06]

```
sample_c.re = (sample/32768.0) * cos(2*PI*f_zf/f_s * n);
sample_c.im = (sample/32768.0) * (-sin(2*PI*f_zf/f_s * n));
      // Quadratur-Mischer von ZF auf DC

samples_c[n % elements(samples_c)] = sample_c;
      // Store in circular buffer
```

Quellcode 1: Quadraturmischer (drm_rx.cpp)

Auf die Tiefpässe wurde vorübergehend verzichtet, weil das per Spark generierte Testsignal sehr gut bandbegrenzt ist und die Spiegelfrequenzen nicht weiter stören.

3.3 Grobe Zeitbereichs-Synchronisation

Die grobe Zeitbereichs-Synchronisation macht sich die Redundanz des Schutzintervalls zunutze, indem es nach der Stelle sucht, an der das Schutzintervall endet und T_u später der Nutzteil des Symbols ebenfalls endet. Hierfür wird ständig eine Autokorrelation der empfangenen Samples über die Länge des Schutzintervalls mit den Samples von $t = -T_u$ durchgeführt.

```
static long long ak = 0;                    // running auto-correlation value
static unsigned long long ak_0 = 0;         // running auto-correlation value
                                            // at t=0(+Ng)
static unsigned long long ak_nu = 0;        // running auto-correlation value
                                            // at t=Nu(+Ng)
static signed short samples [Nu + Ng + 10];    // buffer of old samples
static int init_done = 0;
static unsigned int buf_ptr = 0;

double autokorr (signed short sample)
{
      unsigned int ptr1, ptr2, ptr3, ptr4;

      ptr1 = buf_ptr;
      ptr2 = (buf_ptr + Nu) % (Nu + Ng + 10);
      ptr3 = (buf_ptr + Ng) % (Nu + Ng + 10);
      ptr4 = (buf_ptr + Nu + Ng) % (Nu + Ng + 10);

      ak = ak - (long)samples[ptr1]*samples[ptr2] +
             (long)samples[ptr3]*sample;
            // (15 bit + sign)*(15 bit + sign) = 30 bit + sign
            // ak is of type long long = 63 bit + sign
            // thus it can hold the addition of 2^(63-30) = 8.6*10^9
            // product terms
            // ak can be interpreted as fixed point representation with
            // sign + 33 bit integer part + 30 bit fractional part
      ak_0 = ak_0 - (long)samples[ptr1]*samples[ptr1] +
             (long)samples[ptr3]*samples[ptr3];
      ak_nu = ak_nu - (long)samples[ptr2]*samples[ptr2] +
             (long)sample*sample;

      samples[ptr4] = sample;

      buf_ptr = (buf_ptr + 1) % (Nu + Ng + 10);
```

```
	if (ak_0 >= ak_nu)
	{
		if (ak_0 == 0) return 0.0;
		else return (double)ak / (double)ak_0;
	}
	else
	{
		if (ak_nu == 0) return 0.0;
		else return (double)ak / (double)ak_nu;
	}
}
```

Quellcode 2: Autokorrelation mit dem Schutzintervall (autokorr.cpp)

Immer am Ende des Symbols weist diese Autokorrelation eine Spitze auf. Wird diese gefunden, war die grobe Zeitbereichs-Synchronisation erfolgreich.

```
static double last_max = 0.0;       // correlation value of last maximum
static unsigned int last_pos = -1;  // position of last correlation maximum
                                    // 0=this sample, 1=last sample and so on
static double new_max = 0.0;        // correlation value of new maximum
static unsigned int new_pos = -1;   // position of new correlation maximum

// peak detector of guard time auto correlation
unsigned int peak_detect (signed short sample)
{
	double ak_now;
	unsigned int return_pos;

	last_pos++;             // time moved on 1 sample time
	new_pos++;              // maximums' positions 1 s.t. more in the past

	return_pos = last_pos;

	ak_now = autokorr(sample);    // calculate autocorrelation for this
	                              // sample
	if (ak_now >= new_max)        // if new maximum found
	{
		new_max = ak_now;         // save its value and position
		new_pos = 0;
	}

	if (last_pos == 2*Nu + Ng)    // if almost 2 symbols passed since
	                              // last maximum
	{
		last_pos = new_pos;       // promote new maximum to last maximum
		last_max = new_max;
		new_pos = 0;              // restart search for new maximum at
		                          // current sample
		new_max = ak_now;
	}

	return return_pos;            // return the (old) last maximum's
}                                 // position
```

Quellcode 3: Suche nach der Korrelationsspitze (peakdtct.cpp)

3.4 Parallel-Seriell-Wandlung

Sobald die grobe Zeitbereichs-Synchronisation abgeschlossen ist und sich die Samples des letzten Symbols im Pufferspeicher befinden, wird die Parallel-Seriell-Wandlung durchgeführt:

```
int k;
for (k=kmin; k<=kmax; k++)
{
      complex c;
      c = c_dft(k, Nu, symbol_c);
      cell[(n_sym + 1) % elements(cell)][k+abs(kmin)] = c;
}
```

Quellcode 4: Parallel-Seriell-Wandlung (drm_rx.cpp)

Hierzu wird für die verwendeten Subträger k die Diskrete Fourier-Transformation (DFT) nach der Vorschrift

$$X[k] = \frac{1}{N}\sum_{i=0}^{N-1} x[i]\left(\cos\left(2\pi ki / N\right) - j\sin\left(2\pi ki / N\right)\right)$$

[SMIT99] berechnet:

```
// calculate complex DFT for one frequency component
// k=0...N/2   -> positive frequencies
// k=N/2...N-1 -> negative frequencies
// N=number of input values
// data=pointer to N complex input values
complex c_dft (int k, unsigned int N, const complex *data)
{
      unsigned int i;
      complex result;
      complex factor;

      result.re = 0;
      result.im = 0;

      for (i=0; i<=(N-1); i++)
      {
            factor.re = cos(2*PI*k*i/N);
            factor.im = -sin(2*PI*k*i/N);
            result.re += data[i].re*factor.re - data[i].im*factor.im;
            result.im += data[i].re*factor.im + data[i].im*factor.re;
      }
      result.re /= N;
      result.im /= N;

      return result;
}
```

Quellcode 5: Berechnung der DFT (dft.cpp)

3.5 Feine Zeitbereichs-Synchronisation

Mit Hilfe der Formel aus Abschnitt 1.4.3 wird τ für die drei Frequenzreferenzen errechnet und der Durchschnitt aus den drei Werten gebildet:

```
double avg_tau;
int k;

avg_tau = 0;

for (k=0; k<=kmax; k++)
{
      if (celltype(0, k) == CELL_FREQ_REF)
      {
            // Found one of the three frequency references that are
            // present in every symbol and every robustness mode at 750,
            // 2250 and 3000 Hz.
            // Calculate phase shift at these cells and derive timing
            // offset
            // tau [s] = phi_k / (2*PI*f_k).
            // tau [samples] = phi_k*Nu / (2*PI*k)
            complex c;
            double nominal, actual;
            double phi, tau;

            c = cell[n_sym % elements(cell)][k+abs(kmin)];
            nominal = theta1024_2_pha(cellphase(0, k));
            if (nominal < 0) nominal += 2*PI;
            actual = compl_pha(c);
            if (actual < 0) actual += 2*PI;
            phi = actual - nominal;
            tau = phi*Nu / (2*PI*k);
            avg_tau += tau;
      }
} // for k

avg_tau /= 3;
```

Quellcode 6: Ermittlung des durchschnittlichen Zeitsynchronisations-Fehlers (drm_rx.cpp)

Wenn das Durchschnittliche τ kleiner als eine halbe Sampleperiode ist, ist die feine Zeitbereichs-Synchronisation erreicht. Ist sie größer, wird der Symbolstart entsprechend angepasst:

```
if ((avg_tau*avg_tau > 0.5*0.5) && (avg_tau*avg_tau < Ng*0.5*Ng*0.5))
{
      if (avg_tau >= 0.0) symbol_start += (int)(avg_tau + 0.5);
      else symbol_start += (int)(avg_tau - 0.5);
      // If the average tau is greater than one sample period,
      // adjust the symbol start accordingly. Adding tau
      // actually means putting the symbol start further in the past
      // WARNING: Adjusting the symbol start based on only three
      //          sub-carriers is risky. If the channel imposes a
      //          significant phase shift on only one of them, the
      //          time synchronization can be destroyed rather than
      //          improved. tau is limited to Ng/2 because of that.
      //          Maybe this should just be handled by the channel
```

```
      //          estimator?
}
else if (avg_tau*avg_tau <= 0.5*0.5)      // Fine time synchronisation is
{                                         // achieved if tau is less than
      sync = SYNC_TIME_FINE;              // half the sample interval.
}
```

Quellcode 7: Anpassen der Zeitsynchronisation (drm_rx.cpp)

Weil die Berechnung auf nur drei Pilotzellen basiert, besteht ein gewisses Risiko, dass die Synchronisation verschlechtert und nicht verbessert wird. Das kann passieren, wenn z.B. durch eine ungünstige Kanal-Übertragungsfunktion einer oder mehrere der drei Piloten eine Phasendrehung erfährt. Es ist daher zu untersuchen, ob die Feinzeitsynchronisation nicht mit höherer Genauigkeit etwas später in der Synchronisationskette anhand der *channel state information* des Kanalschätzers erreicht werden kann. Hier könnten zusätzlich zu den Frequenzreferenzen die *gain references* in die Berechnung miteinbezogen werden.

3.6 Rahmensynchronisation

Für die Rahmensynchronisation werden die Zeitreferenzen eingesetzt, die nur im ersten Symbol eines Rahmens enthalten sind. Um herauszufinden, ob das zuletzt empfangene Symbol das erste des Rahmens war, werden die empfangenen Zellen mit den Sollwerten für die Zeitreferenzen kreuzkorreliert. Parallel wird die Autokorrelationsfunktion der Referenzen und der empfangenen Symbole berechnet. Am Ende wird der ermittelte Kreuzkorrelationswert durch die größere der beiden Autokorrelationswerte geteilt. Dadurch wird erreicht, dass der Kreuzkorrelationswert kleiner als eins bleibt.

```
static double correlations[24] =
            {DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX,
             DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX,
             DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX,
             DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX, DBL_MAX};
      // Correlation values of the last Ns symbols. Ns ranges from
      // 15 to 24.
      // Array sized for worst case at mode D (Ns = 24)
      // 'static' qualifier makes sure the data is retained even after
      // leaving variable scope.
      // Initialized to maximum value that can be held within a double
      // to make sure Ns symbols have to be received before first
      // synchronisation.

int k;
complex correlation;
double corr_max_ref;
double corr_max_signal;
correlation.re = 0.0; correlation.im = 0.0;
corr_max_ref = 0.0;
corr_max_signal = 0.0;
```

```
for (k=0; k<=kmax; k++) // time references are only in the 4.5 kHz spectrum
{                       // (thus no negative cell indices)
      if (celltype(0, k) == CELL_TIME_REF)
      {                 // assume this is 1st symbol in transmiss. frame
            complex c;
            complex_e ref;
            complex_e mult;

            c = cell[n_sym % elements(cell)][k+abs(kmin)];
            ref.mag = cellgain(0, k);
            ref.pha = theta1024_2_pha(cellphase(0, k));

            mult.mag = compl_mag(c)*ref.mag;
            mult.pha = compl_pha(c)-ref.pha;
            correlation.re += compl_re(mult);
            correlation.im += compl_im(mult);

            corr_max_ref += ref.mag*ref.mag;
            // imaginary part is automatically zero
            corr_max_signal += compl_mag(c)*compl_mag(c);  // (pha-pha = 0)
      } // CELL_TIME_REF
} // for k

if (corr_max_ref >= corr_max_signal)
{
      if (corr_max_ref == 0.0) correlation.re = 0.0;
      else correlation.re /= corr_max_ref;
}
else
{
      if (corr_max_signal == 0.0) correlation.re = 0.0;
      else correlation.re /= corr_max_signal;
}
```

Quellcode 8: Korrelation mit den Zeitreferenzen (drm_rx.cpp)

Für die Kreuzkorrelation werden zurzeit nur Zeitreferenzen herangezogen. Laut DRM-Spezifikation dienen auch die Frequenzreferenzen gleichzeitig als Zeitreferenzen. Da sie jedoch in jedem Symbol den gleichen Wert haben, können sie die Korrelation nicht verbessern. Da für die Zeitreferenzen angenommen wird, es handle sich um das erste Symbol eines *transmission frames*, könnten die *gain references* für s = 0 mit zur Kreuzkorrelation herangezogen werden. In der Spezifikation wird diese Möglichkeit nicht erwähnt, sollte es aber einmal Probleme mit der Rahmensynchronisation geben, wäre es lohnenswert, diese Erweiterung zu untersuchen.

Nachdem der aktuelle Kreuzkorrelationswert berechnet wurde, wird er mit den letzten N_S - 2 Ergebnissen verglichen. Ist der aktuelle Wert der größte, handelt es sich um das erste Symbol des Rahmens.

```
unsigned int i;
int frame_sync;
frame_sync = 1;
```

```
for (i=Ns-1; i>0; i--)
{
      correlations[i] = correlations[i-1];
      if (correlations[i] > correlation.re) frame_sync = 0;
}
correlations[0] = correlation.re;

if (frame_sync)
{
      sync = SYNC_FRAME;
      s = 0;
      s_eq = Ns - (pilot_y - 1) - 1;
}
```

Quellcode 9: Suche nach dem Korrelationsmaximum der Zeitreferenzen (drm_rx.cpp)

Wenn die Rahmensynchronisation erreicht ist, können auch die *gain references* und damit der Kanalschätzer verwendet werden. Der Symbolzähler für den Ausgang des Kanalschätzers s_eq wird nicht wie s auf Null gesetzt, weil der Kanalschätzer durch eine Interpolation in Zeitrichtung dem zuletzt empfangenen Symbol etwas hinterherhinkt.

3.7 Kanalschätzer

Ein realer Kanal einer OFDM-Übertragung ist zeit- und frequenzvariant. Das bedeutet, dass die Übertragungsfunktion für die einzelnen Subträger unterschiedlich ist und sich über die Zeit ändert. Manche Subträger werden durch Interferenzen verstärkt, andere abgeschwächt oder ausgelöscht, zusätzlich kann sich die Phase ändern. DRM setzt für das nachfolgende Demapping einen kohärenten Empfang voraus, das heißt, dass Veränderungen durch den Kanal an Amplitude und Phase der einzelnen Zellen möglichst gut wieder herausgerechnet werden müssen.

Zu diesem Zweck werden in die Symbole Pilotzellen, insbesondere *gain references* eingestreut. Betrag und Phase dieser Referenzzellen ist bekannt, durch den Vergleich zwischen empfangenem und erwartetem Wert lässt sich also die Kanalantwort für diese Zellen bestimmen. Die Piloten sind so verteilt, dass sich durch zweidimensionale Interpolation mit ausreichender Genauigkeit bestimmen lässt, wie die Kanalantwort für die dazwischen liegenden Datenzellen lautet. Dividiert man nun den komplexen Wert einer Zelle durch den zugehörigen komplexen Wert der Kanalantwort, genannt *channel state information*, erhält man unter der Voraussetzung perfekter Synchronisation und Kanalschätzung wieder den ausgesendeten Wert zuzüglich des auf der Übertragungsstrecke aufgefangenen Rauschens.

Der Betrag der *channel state information* kann außerdem im Viterbi-Decoder zum *soft-decision decoding* verwendet werden und so einen zusätzlichen Codiergewinn erreichen.

```
void channel_est (const complex cell[][461], unsigned int n_sym, unsigned
int n_mod,
                          unsigned char s, complex *csi)
{
      char x, y;
      int k;

      for (k=kmin; k<=kmax; k++)
      {
            complex accumulator;
            double weight_acc;

            accumulator.re = 0.0;
            accumulator.im = 0.0;
            weight_acc = 0.0;

            for (x=(-(pilot_x-1)); x<=(pilot_x-1); x++)
            {
                  if ((k+x >= kmin) && (k+x <= kmax))
                  {           // don't look beyond the used cells
                        for (y=(-(pilot_y-1)); y<=(pilot_y-1); y++)
                        {
                              if (celltype(modulo(s+y, Ns), k+x) ==
                                    CELL_GAIN_REF) || (
                                 (celltype(modulo(s+y, Ns), k+x) ==
                                    CELL_FREQ_REF)
                              {
                                    complex received;
                                    complex expected;
                                    complex_e expected_e;
                                    double weight;

                              // conveniently store received cell value
                                    received.re = cell[modulo(n_sym+y,
                                                n_mod)][k+x+abs(kmin)].re;
                                    received.im = cell[modulo(n_sym+y,
                                                n_mod)][k+x+abs(kmin)].im;

                                    // get expected pilot cell value in
                                    // cartesian notation:
                                    expected_e.mag = cellgain(modulo(s+y,
                                                      Ns), k+x);
                                    expected_e.pha = theta1024_2_pha(
                                          cellphase(modulo(s+y, Ns), k+x));
                                    expected.re = compl_re(expected_e);
                                    expected.im = compl_im(expected_e);

            // interpolation weight is higher (max. 1) for pilots close
            // to the cell at s, k and smaller for pilots that are further
            // away
                                    weight = (double)((pilot_x - abs(x)) *
                               (pilot_y - abs(y))) / (pilot_x * pilot_y);
                                    weight_acc += weight;

      // received = csi[][]*expected <=> csi[][] = received / expected

      // add csi[][] term with proper interpolation weight to accumulator
```

```
                                    if (expected.re != 0.0) accumulator.re
                                     += (received.re/expected.re) * weight;
                                    if (expected.im != 0.0) accumulator.im
                                     += (received.im/expected.im) * weight;
                                } // CELL_GAIN_REF
                            } // for y
                        }
                    } // for x
                    accumulator.re /= weight_acc;
                    accumulator.im /= weight_acc;
                        // correct accumulator by accumulated weight which is
                        // 1.0 most of the time but higher near freq. refs
                    csi[k+abs(kmin)].re = accumulator.re;
                    csi[k+abs(kmin)].im = accumulator.im;
                        // store accumulated value in
                        // channel state information output array
            } // for k
}
```

Quellcode 10: Kanalschätzung (chanest.cpp)

Diese Funktion sucht für jede Zelle im aktuellen Symbol die Pilotzellen in der unmittelbaren Umgebung und addiert deren gewichtete Werte auf. Die Gewichtung hat dabei folgendes Muster (exemplarisch für Mode A):

1/20	2/20	3/20	1/5	3/20	2/20	1/20
2/20	4/20	6/20	2/5	6/20	4/20	2/20
3/20	6/20	9/20	3/5	9/20	6/20	3/20
4/20	8/20	12/20	4/5	12/20	8/20	4/20
1/4	2/4	3/4	1	3/4	2/4	1/4
4/20	8/20	12/20	4/5	12/20	8/20	4/20
3/20	6/20	9/20	3/5	9/20	6/20	3/20
2/20	4/20	6/20	2/5	6/20	4/20	2/20
1/20	2/20	3/20	1/5	3/20	2/20	1/20

Abbildung 13: Interpolationsmuster des Kanalschätzers für Mode A

Dieses Muster stellt sicher, dass die Pilotzellen, die dichter an der zu interpolierenden Zelle liegen, stärker in die Berechnung eingehen. Für den Fall, dass Frequenzreferenzen im Bereich des Interpolationsmusters liegen, ergibt sich eine Summe der Gewichte, die größer als 1 ist. In diesem Fall muss das Interpolations-Ergebnis anschließend noch durch die Summe der Gewichte geteilt werden.

Zurück im Hauptprogramm werden die Zellen dann normalisiert, indem ihre Werte durch die der *channel state information* geteilt werden:

```
int k;
for (k=kmin; k<=kmax; k++)
{
      // cell_eq[] = cell[][] / csi[]
      complex_e c;

      if (compl_mag(csi[k+abs(kmin)]) != 0.0)
            c.mag = compl_mag(cell[modulo(n_sym-(pilot_y-1),
                              elements(cell))][k+abs(kmin)])
                  / compl_mag(csi[k+abs(kmin)]);
      c.pha = compl_pha(cell[modulo(n_sym-(pilot_y-1),
                        elements(cell))][k+abs(kmin)]);
      c.pha = c.pha - compl_pha(csi[k+abs(kmin)]);
#ifdef _CHEAT_ON_CSI_PHASE
      // For unknown reasons the CSI phase of all sub-carriers
      // is about PI/4. In order to counteract the constellation
      // rotation introduced by that, the phase is back-rotated
      // by exactly PI/4.
      // This works but it should be investigated where the
      // CSI phase shift stems from.
      c.pha = c.pha + PI/4;
#endif

      cell_eq[k+abs(kmin)] = exp2rect(c);
}
```

Quellcode 11: Normalisierung der Zellen (drm_rx)

Es hat sich herausgestellt, dass für das Testsignal die Kanalantwort eine konstante Phasenschiebung von ca. π/4 über alle Zellen ist. Tatsächlich war die Phase aber in Ordnung. Der Grund für diesen Offset ist bis dato nicht geklärt, weshalb über das Makro _CHEAT_ON_CSI_PHASE temporär die Phasenschiebung wieder rückgängig gemacht werden kann.

3.8 Demapping der Subträger

Da bisher nur mit 4-QAM modulierte FAC-Zellen ausgewertet werden, ist das Demapping sehr einfach: Ist der Realteil kleiner oder gleich Null, ist das höherwertige Bit eine Eins. Andernfalls ist es eine Null. Analog für das niederwertige Bit und den Imaginärteil:

```
// demapping:
y[0][n_fac*2]     = cell[k+abs(kmin)].re <= 0.0 ? 1 : 0;
y[0][n_fac*2 + 1] = cell[k+abs(kmin)].im <= 0.0 ? 1 : 0;
```

Quellcode 12: Demapping von 4-QAM (fac.cpp)

3.9 Deinterleaving

Zu Beginn des Programms muss einmal eine Zahlenreihe permutiert werden, anhand derer später das Deinterleaving stattfindet:

```
q = s/4 - 1;

perm[0] = 0;
unsigned int i;
for (i=1; i<xin; i++)
{
      perm[i] = modulo((t[p]*perm[i-1] + q), s);
      while (perm[i] >= xin)
      {
            perm[i] = modulo((t[p]*perm[i] + q), s);
      }
}
```

Quellcode 13: Ermitteln der Permutation (intrleav.cpp)

Sind alle Daten eines FAC-Blocks eingetroffen, wird mit Hilfe der vorher angelegten Tabelle das Deinterleaving durchgeführt:

```
unsigned int i;
for (i=0; i<2*Nfac; i++)
{
      v[0][perm[i]] = y[0][i];
}
```

Quellcode 14: Deinterleaving (fac.cpp)

3.10 Viterbi-Decoder

Der Viterbi-Decoder muss zuerst initialisiert werden. Anhand der an viterbi_init() übergebenen Parameter wird das zugehörige *puncturing pattern* ausgewählt. Für DRM werden punktierte Codes verwendet, diese Punktierung im Sender muss also erst wieder rückgängig gemacht werden:

```
// Depuncture received bitstream
// "i" is the index (starting with 0) of the bit
// "in" is the next received bit
// If this ("ith") input bit is punctured the flag CHAR_MAX is returned
// and the value of "in" otherwise.
// Notice that "in" need not be 0 or 1 only but can be anything except
// for CHAR_MAX. This enables soft-decision Viterbi decoding based on
// a finer differentiation between 0 and 1, i.e. -4 to 3 (3 bit)
char depuncture (char in, unsigned int i)
{
      // The puncturing pattern repeats every n*RX bits, its dimension is
B[n][RX]
//    if (B[(i % (n*RX)) % n][(i % (n*RX)) / n] == 1) return in;
      if (B[(i % (effective_n*RX)) % effective_n][(i % (effective_n*RX)) /
effective_n] == 1) return in;
```

```
      else return CHAR_MAX;
} // depuncture()
```

Quellcode 15: Routine zum De-Punktieren (viterbi.cpp)

```
N_depnct = 0;
for (i=0; i<N; i++)
{
      do                      // Store special depunctured bit symbols until
      {                       // a non-punctured bit is copied
            depunctured[N_depnct] = depuncture(v[i], N_depnct);
            N_depnct++;       // Keep track of the total number of bits
                              // (punctured and un-punctured)
      } while (depunctured[N_depnct-1] == CHAR_MAX);
}
while ((N_depnct % effective_n) != 0)
{
      depunctured[N_depnct] = CHAR_MAX;
      N_depnct++;
}
```

Quellcode 16: De-Punktierung (viterbi.cpp)

Es folgt der Aufbau der *accumulated error matrix*:

```
// Build the accumulated error matrix:
for (symbol=0; symbol<(N_depnct/effective_n); symbol++)
{
      unsigned char state;
      for (state=0; state<STATES; state++)     // go through all possible
      {                                       // old states
            for (a=0; a<=1; a++)              // go through all possible
            {                                 // encoder input bits
                  if ((symbol >= (N_depnct/effective_n - m))
                        && (a != 0)) break;
                  // After the N_depnct data bits follow m tailbits
                  // which are specified to be all zeroes

                  // state: state = memory contents at ENcoder side before
                  // input bit a arrived
                  // bit: received bit index, starting at 0
                  // a: ENcoder input bit
                  unsigned char mem, enc_out;

                  mem = state;
                  enc_out = b(a, &mem);
                  // mem: next state / memory contents at ENcoder side
                  // after input bit a arrived
                  // enc_out: Convolutional ENcoder output (not punctured)

                  char enc_out_punc[n];   // De-puncture enc_out and store
                                          // it in char array
                  unsigned int i;
                  for (i=0; i<effective_n; i++)
                  {
                        enc_out_punc[i] = depuncture(enc_out & 1,
                                          symbol*effective_n + i);
                        enc_out >>= 1;
                  }
```

```
                  unsigned int distance;  // Calculate Hamming distance
                  distance = hamming_distance(enc_out_punc,
                              &depunctured[symbol*effective_n]);

                  if (symbol >= (N_depnct/effective_n - m)) distance = 0;
                        // The tailbits are not transmitted, thus there's
                        // nothing to compare against which means zero
                        // distance

                  // add-compare-select operation start
                  distance += err_acc[state][symbol]; // add accumulated
                                                      // error of predecessor state
                  if (distance > UCHAR_MAX) distance = UCHAR_MAX;
                        // limit to maximum value

                  if (distance <= err_acc[mem][symbol + 1])
                  {
                        err_acc[mem][symbol + 1] = distance;
                              // store new accumulated error metric
                        predec[mem][symbol] = state;
                              // store predecessor state
                  }
                  // add-compare-select operation done
            } // for a
      } // for state
} // for symbol
```

Quellcode 17: Aufbau der accumulated error matrix (viterbi.cpp)

Der Decoder spielt alle Möglichkeiten durch, die der Encoder ausgehend vom bisherigen Status erreichen kann. Anschließend berechnet er die Hamming-Distanz zwischen dem tatsächlich empfangenen und dem in dieser Variante durchgespielten Ausgangsdatum. Dieses wird dann auf dasjenige des Vorgängerstatus' aufaddiert. Im Idealfall, wenn keinerlei Empfangsfehler vorliegen, gibt es einen Pfad durch den Trellis, dessen akkumulierter Fehler stets Null ist. Andernfalls steigen die Fehlerwerte an, es gibt aber trotzdem noch einen Pfad, der den geringsten Fehler aufweist. Die Daten, die zu diesem Pfad gehören, sind mit größter Wahrscheinlichkeit diejenigen, die gesendet wurden. Um diesen Pfad zu ermitteln, wird der Trellis jetzt rückwärts abgesucht:

```
// Traceback:
state = 0;              // Because of the all-zeroes tailbits
                        // the traceback always starts at state 0

for (symbol=(N_depnct/effective_n - 1); (int)symbol>=0; symbol--)
{
      unsigned char next_state;
      unsigned char pred_state;

      if (symbol < L)   // make sure the tailbits are not written to x[]
      {
            next_state = state;
            pred_state = predec[state][symbol];

            b(0, &pred_state);
```

```
            // pred_state is now the shift register memory content
            // when a_i = 0 has been the input. Thus if the new
            // pred_state is equal to next_state, next_state is
            // reached from (the old) pred_state by having a_i be 0.

            if (pred_state == next_state) x[symbol] = 0;
            else x[symbol] = 1;
      }

      state = predec[state][symbol];
} // for symbol
```

Quellcode 18: Traceback durch den Trellis (viterbi.cpp)

3.11 Transmission-super-frame-Synchronisation

Die Position des momentan übertragenen *transmission frame* im *transmission super frame* wird im FAC direkt gesendet. Sobald der FAC dekodiert ist, wird die Transmission-super-frame-Synchronisation erreicht durch:

```
frame_eq = fac.identity % 3;  // set transmission super frame position
```

Quellcode 19: Transmission-super-frame-Synchronisation (drm_rx.cpp)

4 Fazit

Es war eine schwierige, aber auch sehr interessante Aufgabe, ein so komplexes Signal wie das von Digital Radio Mondiale zu decodieren. Leider war die Aufgabe zu komplex, um im Rahmen meiner Studienarbeit tatsächlich einen voll funktionsfähigen Empfänger zu bauen. Nichtsdestoweniger habe ich bei der Bearbeitung dieses Projektes viel über moderne Datenübertragung gelernt.

Ich hoffe, dass meine Arbeit – von mir selbst oder anderen – fortgeführt wird. Neben der Unterstützung für die Multiplex-Kanäle SDC und MSC gibt es vor Allem noch Detailverbesserungen durchzuführen. Ein Viterbi-Decoder mit soft decision könnte die Störfestigkeit um bis zu 3 dB verbessern, eine kontinuierlich im Hintergrund ablaufende Synchronisation würde Sprünge in den Übertragungsparametern ausgleichen können und eine Frequenzsynchronisation, wie sie für „echte“ Signale benötigt würde, fehlt noch komplett.

Überhaupt ist die Software noch sehr wenig auf Belastung getestet worden. In der DRM-Spezifikation sind verschiedene Kanalmodelle angegeben, die es sich lohnen würde, auf die Software loszulassen.

Aber auch wenn diese Liste der nötigen und wünschenswerten Verbesserungen lang ist, ein Etappenziel ist erreicht und die Software dient vielleicht auch als Anschauungsobjekt dafür, worauf es beim Empfang von OFDM ankommt.

5 Literaturverzeichnis

[DRMC08]: http://www.drm.org/for-listeners/watch-hear-drm/listen-to-drm-audio-samples-drm-field-tests/

[ETSI06]: European Telecommunications Standards Institute: ETSI TS 102 509 V1.1.1 (2006-05): Digital Radio Mondiale (DRM); Single Channel Sumulcast

[ETSI08]: European Telecommunications Standards Institute: ETSI ES 201 980 V2.3.1 (2008-02): Digital Radio Mondiale (DRM); System Specification

[FAZE03]: Fazel, Khaled; Kaiser, S.: Multi-Carrier and Spread Spectrum Systems. 1. Auflage. Chichester: John Wiley & Sons Ltd. 2003

[HAJJ07]: El Hajjar, Charbel: Synchronization Algorithms for OFDM Systems (IEEE802.11a, DVB-T): Analysis, Simulation, Optimization and Implementation Aspects. Erlangen. Diss. 2007

[JOND02]: Jondral, Friedrich; Machauer, Ralf; Wiesler, Anne: Software Radio: Adaptivität durch Parametrisierung. Weil der Stadt: J. Schlembach Fachverlag. 2002

[MEYE08]: Meyer, Martin: Kommunikationstechnik: Konzepte der modernen Nachrichtenübertragung. 3. Auflage. Wiesbaden: Vieweg + Teubner. 2008

[NEE00]: van Nee, Richard; Prasad, Ramjee: OFDM for wireless multimedia communications: Artech House Publisher. Boston; London. 2000

[OHM07]: Ohm, Jens-Rainer; Lüke, Hans Dieter: Signalübertragung: Grundlagen der digitalen und analogen Nachrichtenübertragungssysteme. 10. Auflage. Berlin: Springer-Verlag. 2007

[ROPP06]: Roppel, Carsten: Grundlagen der digitalen Kommunikationstechnik: Übertragungstechnik – Signalverarbeitung – Netze. München: Fachbuchverlag Leipzig. 2006

[SMIT99]: Smith, Steven W.: The Scientist and Engineer’s Guido to Digital Signal Processing. Second Edition. San Diego: California Technical Publishing. 1999

[STOT98]: Stott, Jonathan H.: The how and why of COFDM, EBU Technical Review, Winter 1998

[STOT01]: Stott, Jonathan H.: DRM – key technical features, EBU Technical Review, March 2001

[WIKI07]: http://de.wikipedia.org/w/index.php?title=Bild:Ionospheric_reflection_german.png&filetimestamp=20070221125823

6 Anhang A: Inhalt der Begleit-CD

- Digitale Fassung dieser Arbeit
- Angegebene Literatur, soweit digital vorhanden
- Quellcode der Empfänger-Software
- DRM-Test-Files
- Spark 1.7.1 zur Erzeugung von DRM-WAV-Files
- Dream-software
- Zusätzliche, nicht direkt in der Arbeit zitierte digitale Literatur

Erklärung zur Urheberschaft

Hiermit erkläre ich, dass ich diese Arbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel verwendet habe.

Aalen, den 31.10.2008 Burkart Lingner