Inhaltsverzeichnis

I. Motivation

Digital Video Broadcasting – Satellite Services to Handheld Devices (DVB-SH) gehört zur *aktuellsten und modernsten* Errungenschaften der Nachrichtenübertragungstechnik. Der erste europäische Satellit mit Namen "W2A", der DVB-SH Dienste anbietet, wurde am 3. April 2009 in eine geostationäre Umlaufbahn gebracht. [1]

„Am 6. Oktober 2009 wurde in Erlangen am Lehrstuhl LIKE ein Versuchssender in Betrieb genommen und somit das erste DVB-SH-Testprogramm in Deutschland. Im Rahmen des Projekts "Mobiler Rundfunk Erlangen" werden Rundfunkstandards für den mobilen Fernsehempfang erprobt und weiterentwickelt.“ [2]

Der Auftrag zur Entwicklung eines DVB-SH Empfängers wurde von der Europäischen Weltraumbehörde (ESA) ausgeschrieben. Das Fraunhofer IIS wurde mit der Realisierung des Projektes beauftragt. Seitdem beschäftigt sich ein Entwicklerteam mit Entwurf und Implementierung eines Testempfängers. Es ist wichtig, jede Stufe im Entwicklungsprozess mit einem Test abzuschließen und zu überprüfen, ob die Vorgaben korrekt umgesetzt wurden.

Die vorliegende Arbeit beschäftigt sich mit einem Testverfahren zur Beurteilung der Qualität von digitalen Rundfunkempfängern. Die Kerngröße bei diesen Messungen ist die Bitfehlerrate (BER) im Bezug zum Verhältnis von Trägersignal-Leistung und Rauschenleistung. Wird dieser Zusammenhang in einem Diagramm dargestellt kann der sogenannten "Cliff-Effekt" nachgewiesen werden. Diese Erscheinung ist eine wichtige Kenngröße in digitalen Übertragungssystemen, da sie genau die Grenze bestimmt, an der der Empfänger das Sendesignal nicht mehr wiederherstellen kann. Die Forschergruppe des Fraunhofer Instituts kann dadurch den aktuellen Stand der Empfängerentwicklung auf mögliche Schwachstellen untersuchen und frühzeitig Verbesserungen umsetzten. Ein hoher Grad an Automatisierung des Messvorgangs erlaubt eine effiziente Nutzung der verfügbaren Arbeitszeit.

II. Aufgabenstellung

Für die Beurteilung der Qualität von digitalen Rundfunkempfängern ist die Möglichkeit, auch bei schlechten Empfangsbedingungen das Ausgangssignal zu rekonstruieren, von entscheidender Bedeutung. Zur Beurteilung der Qualität wird der digitale Rundfunkempfänger im Labor mit Signalen unterschiedlicher Modulationen und Coderaten gespeist. Diese Signale werden vor dem Empfang durch Echtzeit-Simulation von verschiedenen Kanalmodellen gestört. Eine hohe praktische Relevanz haben Kanalmodelle mobiler Empfangsszenarien, da diese den späteren Einsatzszenarien nahe kommen. Allerdings sind die Ergebnisse dieser Kanalmodelle nur durch Simulationen zu bestimmen. Statische Kanäle sind hingegen berechenbar, können aber nur eingeschränkt auf den realen Einsatz übertragen werden. Eine hohe Qualität eines Empfängers zeigt sich durch einen möglichst geringen Abstand der gemessenen Daten zu den berechneten bzw. simulierten unteren Schranken.

Es soll ein Framework aufgebaut werden, das die automatisierte Durchführung von Messreihen an einem DVB-SH Empfänger ermöglicht. Die Messreihen sollen mit verschiedenen Kanalmodellen und Signal-Rausch-Abständen (SNR) sowie unterschiedlichen Modulationen und Coderaten durchgeführt werden. Mit den ausgewerteten Messreihen soll eine Aussage über die Qualität des Empfängers getroffen werden.

III. Einleitung

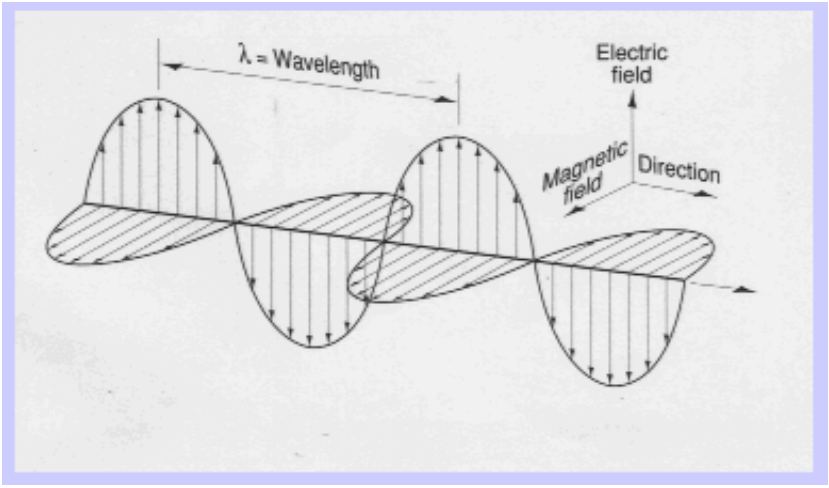
Im theoretischen Teil der Arbeit werden grundlegende Themen der digitalen Nachrichtenübertragung und der Kanalmodelle beschrieben. Dieses Hintergrund-Wissen unterstützt den Leser beim Verstehen der späteren praktischen Messungen. Im ersten Kapitel liegt der Schwerpunkt auf den AWGN- und Rice-Kanalmodellen. Diese Modelle sind zur Beschreibung einer DVB-SH Übertragungsstrecke über den Satelliten geeignet. Die Kanalmodelle zur Simulation des terrestrischen Pfades wurden in dieser Arbeit nicht betrachtet. Der während der Messung verwendete Signalgenerator stellt entweder Kanalbedingungen eines Rice- oder eines AWGN-Kanals nach, um die reale Übertragung der DVB-SH Signale zu simulieren. Zudem enthält das Kapitel Beschreibungen der Übertragungswege. Im zweiten Kapitel geht es grundlegend um den digitalen Fernsehübertragungsstandard DVB mit dem Schwerpunkt DVB-SH. Dabei werden Systemarchitektur, Empfängerfunktionalität, sowie die Qualitätsmerkmale von DVB-SH betrachtet.

Der praktische Teil erläutert die konkrete Umsetzung der theoretischen Vorgaben (Kapitel 3). Zu Beginn steht ein Überblick über den Versuchsaufbau und die beispielhafte Beschreibung eines Testszenarios. Abschließend werden die Messergebnisse dargestellt.

1. Kapitel: Grundlagen

1.1 Elektromagnetisches Feld

Funktechnik bedeutet die Methode, Signale aller Art mit Hilfe elektromagnetischer Wellen drahtlos zu übertragen. Die elektromagnetische Welle ist die Ausbreitung der Änderung elektrischer und magnetischer Zustände im Raum (Abbildung 1). Die Ausbreitungsgeschwindigkeit wird von den dielektrischen Eigenschaften des Mediums bestimmt. Die Energien der elektrischen und magnetischen Felder existieren gleichzeitig und bilden zusammen die elektromagnetische Welle. Die mathematische Beschreibung der Zusammenhänge zwischen den bekannten elektrischen und magnetischen Phänomenen ermöglichen die Maxwell-Gleichungen. Diese Differentialgleichungen können unter bestimmten Bedingungen zu den so genannten Wellengleichungen führen. [3]



$$E(d) = E_0 \cos\omega(t - \frac{d}{v}) \text{ und } H(d) = H_0 \cos\omega\left(t - \frac{d}{v}\right)$$

Abbildung 1: Elektronische (E) und Magnetische (H) Felder (Quelle: [a])

Der Ausbreitungsmechanismus von elektromagnetischen Wellen wird durch die Wellen- und Strahlungstheorie erklärt. Die Grundlage der Wellentheorie sind die Maxwell-Gleichungen. Die Strahlungstheorie verwendet die Regeln der geometrischen Optik. Eine elektromagnetische Welle hat eine Energie von $\overline{S} = \overline{E} \times \overline{H}$. Als Grundlage für die Rechnung, der durch eine Antenne ausgestrahlten elektromagnetischen Energie wird der einfache isotrope

Kugelstrahler genommen. Die Abbildung 2a stellt die Feldanteile dar. Das erzeugte elektromagnetische Feld wird in die drei Bereiche Strahlungsfeld (Fernfeld), induktives und statisches Nahfeld aufgeteilt.

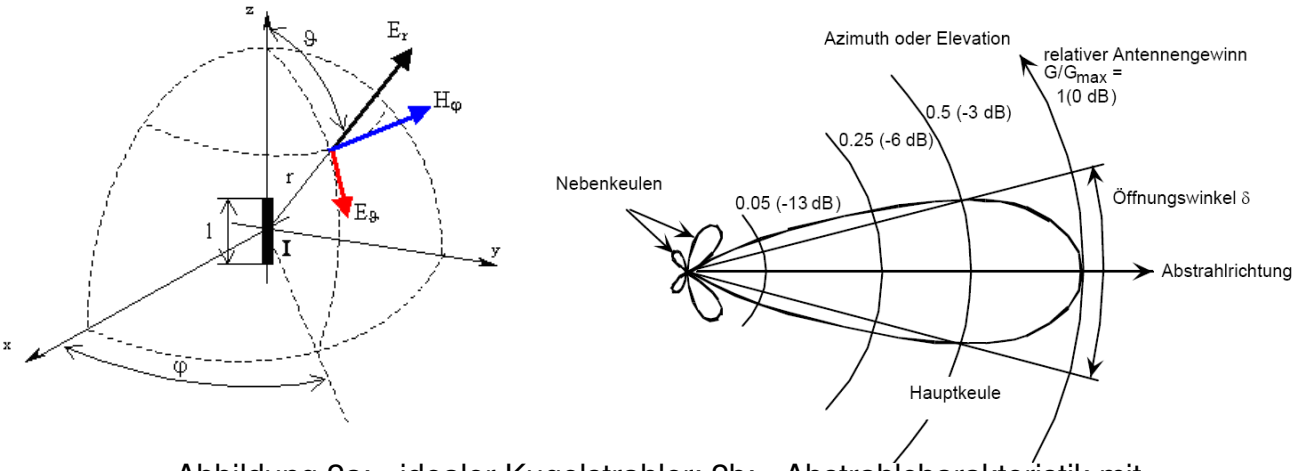


Abbildung 2a: - idealer Kugelstrahler; 2b: - Abstrahlcharakteristik mit Öffnungswinkel (Quelle: [b,c])

Das Fernfeld beginnt, wenn gilt: $r >> \frac{\lambda}{2\pi}$.

Bei Antennen, die im Vergleich zur Wellenlänge eine kurze Antennenlänge besitzen, sind nur die Anteile im Fernfeld von Bedeutung. Aus Sicht des Fernfeldes kann die Antenne als Punktquelle betrachtet werden. In diesem Fall verteilt sich die ausgestrahlte Energie gleichmäßig auf einer kugelförmigen Oberfläche mit der Größe $A = 4\pi r^2$. Die Leistungsdichte in einem bestimmten Punkt im Fernfeld lässt sich bestimmen mit (1.1.1).

$$S_0 = \frac{P_{Sender}}{A} \qquad (1.1.1)$$

Im Gegensatz zur theoretischen Betrachtung als symmetrischer Kugelstrahler besitzen Antennen in der Praxis charakteristische Abstrahlungsbereiche. Der wichtigste Parameter bei der Abstrahlcharakteristik ist der Öffnungswinkel (siehe Abbildung 2b). Innerhalb dieses Winkels konzentriert sich die Hälfte der abgestrahlten Energie, das entspricht einem Verlust von 3 dB, daher wird er oft als *„3 dB“*-Öffnungswinkel bezeichnet. [4] Je stärker eine Antenne die Strahlen bündelt, desto größer ist ihr Antennengewinn. Die Antenne kann bei höherem Gewinn über größere Entfernungen hinweg senden oder schwächere Signale

störungsfrei empfangen. Eine Antenne, mit Gewinn G_{Sender} erzeugt nach der Gleichung (1.1.2) die Leistungsdichte S.

$$S = \frac{P_{Sender} * G_{Sender}}{4\pi r^2} \qquad (1.1.2)$$

Die Empfängerantenne, mit Gewinn $G_{Empfänger,}$ wandelt diese Leistungsdichte in eine Leistung um. Diese Leistung ist von der effektiven Oberfläche der Antenne A_h abhängig (1.1.3).

$$P_{Empfänger} = S * A_h \qquad (1.1.3)$$

wo für A_h gilt:

$$A_h = G_{Empfänger} * \frac{\lambda^2}{4\pi} \qquad (1.1.4)$$

Durch Einsetzen der Gleichungen (1.1.2) und (1.1.4) in (1.1.3) ergibt sich die Gleichung (1.1.5). Mit dessen Hilfe lässt sich die Empfangsleistung in einer bestimmten Entfernung vom Sender abschätzen.

$$P_{Empfänger} = \frac{P_{Sender} * G_{Sender} * G_{Empfänger} * \lambda^2}{4\pi * r^2 * 4\pi} \qquad (1.1.5)$$

Die theoretische Empfängerleistung nach (1.1.5) fällt unter Praxisbedingungen geringer aus. Die Gründe dafür sind die diversen Phänomene der Wellenausbreitung (Kapitel 1.2) und die Eigenschaften des Übertragungskanals (Kapitel 1.3).

1.2 Wellenausbreitung

Das mit einer Antenne erregte elektromagnetische Feld breitet sich mit bestimmter Geschwindigkeit in Strahlungsrichtung aus. Mit zunehmender Entfernung nimmt die Intensität des Feldes ab. Nach der Strahlungstheorie breitet sich die Welle geradlinig aus. Zudem verhält sich die Leistungsdichte indirekt proportional zum Quadrat der Entfernung. Im Freiraum beträgt die Ausbreitungsgeschwindigkeit nahezu Lichtgeschwindigkeit. Die Anwesenheit eines Mediums beeinflusst die Ausbreitungseigenschaften. Die Charakteristik und dielektrischen Eigenschaften des Übertragungsmediums sind frequenzabhängig, daher hängt auch die Wellenausbreitung von der Frequenz ab. Anhand der Welleneigenschaften kann festgestellt werden, dass sich Funkwellen als *Bodenwellen*, *Direktwellen* oder als *reflektierte Wellen* (von Ionosphäre oder von Troposphäre) in der Luft ausbreiten. Die Wellenausbreitung wird von den Erscheinungen *Reflexion, Beugung, Streuung, Interferenz, Brechung, Absorption, Abschattung und Schwund (engl. Fading)* beeinflusst. Detaillierte Beschreibungen zu diesem Thema befinden sich in Anhang A. Aufgrund dieser Phänomene muss bei den Rundfunksystemen mit einer Mehrwegeausbreitung und mit Schwund-Erscheinungen gerechnet werden.

1.2.1 Ausbreitung im Freiraum

Das Übertragungsmedium der Funktechnik ist die freie Luft. Bei der Freiraumausbreitung wird mit der Freiraumdämpfung a_s als Störeffekt gerechnet. Grund dafür ist der spezifische Freiraumwiderstand $\eta = 120\pi$ und die Tatsache, dass die Wellenenergie sich verteilt, wie in Kapitel 1.1 erklärt wurde. Die Freiraumdämpfung gibt an, wie stark sich die Leistung elektromagnetischer Wellen auf dem Weg vom Sender zum Empfänger mit zunehmendem Abstand r durch Strahldivergenz verringert. Anders gesagt ist a_s der Quotient der Ausgangsleistung P_{Sender} von der Senderantenne und der Eingangsleistung $P_{Empfänger}$ bei der Empfängerantenne (1.2.1).

$$L_s = 10 * \lg\left(\frac{P_{Sender}}{P_{Empfänger}}\right) \qquad (1.2.1)$$

Mit (1.1.5):

$$L_s = \frac{P_{Sender}}{P_{Empfänger}} = \left(\frac{4\pi r}{\lambda}\right)^2 * \frac{1}{G_{Sender}} * \frac{1}{G_{Empfänger}} \qquad (1.2.2)$$

In der Praxis kann die Gleichung (1.2.3) auch für die Berechnung der Freiraumdämpfung verwendet werden [4]. Durch die Ersetzung der Wellenlänge λ mit dem Quotienten $\frac{c}{f}$ erhält man eine vereinfachte Gleichung.

$$\mathrm{L}_{\mathrm{s\,[dB]}} = 32.4_{\,[\mathrm{dB}]} + 20\lg \mathrm{f}_{\,[\mathrm{MHz}]} + 20\lg \mathrm{r}_{\,[\mathrm{km}]} - G_{Sender} - G_{Empfänger} \qquad (1.2.3)$$

Die Gleichung (1.2.3) zeigt, dass die Größe der Freiraumdämpfung mit der Frequenz und mit der Entfernung zusammenhängt. Aufgrund der atmosphärischen Erscheinungen, wie zum Beispiel das Wetter, ist die Signaldämpfung in Wirklichkeit größer als die Dämpfung im Freiraum. Die gesamte Empfangsleistung am Ende einer Funkstrecke errechnet sich durch (1.2.4).

$$P_{Empfänger} = P_{Sender} - L_s - L \qquad (1.2.4)$$

L - zusätzlicher Pfadverlust in dB

Wird die Gleichung (1.1.5) in die Gleichung (1.2.4) eingesetzt, ergibt sich die Grundberechnungsformel für das Link Budget. In Kapitel 2.2.3 wird das Link Budget, für die Übertragung mit terrestrischen und satellitischen Sendeanlagen, beschrieben.

1.2.2 Mehrwegeausbreitung

Zwischen Sender und Empfänger existiert natürlich nicht nur der Freiraum, sondern verschiedene Hindernisse, die wiederum die Funkübertragung beeinflussen. Das Signal kann aufgrund von Reflexionen, Brechungen, Streuungen über verschiedene Wege beim Empfänger ankommen. Die Empfangssignale der unterschiedlichen Übertragungswege werden beim Empfänger addiert und verstärken oder schwächen einander. Im ungünstigsten Fall könnten sie sich ganz auslöschen. Die Feldstärke beim Empfänger $E_{Empfänger}$ ist die vektorielle Summe der direkten und der reflektierten Wellen. Das Verhältnis

zwischen der einfallenden E_i und einer reflektierten E_r Welle ist der Reflexionsfaktor Γ. Siehe Gleichung (1.2.5).
Abbildung 3 zeigt den Fall der Zweiwegeausbreitung. Dabei wird mit dem Reflexionsfaktor der Erde (Γ = -1) gerechnet. [5] Die Empfangsfeldstärke errechnet sich aus (1.2.6).

$$\Gamma = \frac{E_r}{E_i} \tag{1.2.5}$$

$$E_{Empfänger} = E_{Direkt} + E_{\mathrm{Re}\,flektiert} \cong E_{Direkt} + E_{Direkt} * \Gamma e^{-j\beta\Delta R} \tag{1.2.6}$$

Die Mehrwegeausbreitung beeinflusst, in welcher Umgebung der Empfänger funktioniert. Auf geradem Gelände werden die Funkwellen nur von der Erdoberfläche reflektiert. In einer Innenstadt können beispielsweise Gebäude, Verkehrsmittel und Straßenschilder Reflektionen verursachen. Für die Beschreibung dieser Effekte werden komplexe Kanalmodelle eingesetzt. Erläuterungen dazu befinden sich in Kapitel 1.3.

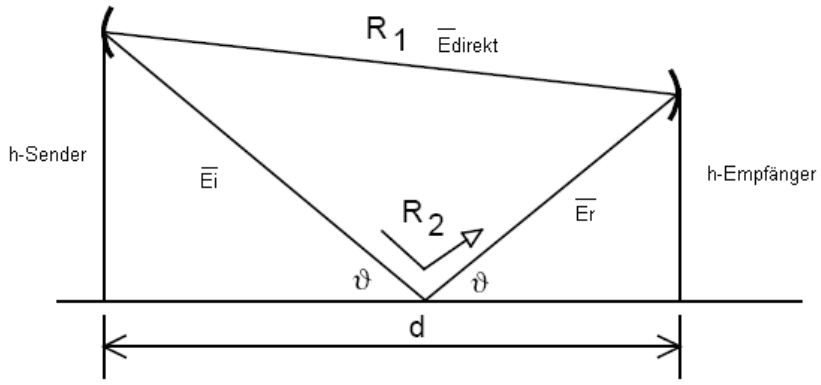


Abbildung 3: Zweiwegeausbreitung (Quelle [d])

1.2.3 Die Bewegung des mobilen Empfangsgeräts

Die moderne Funkübertragung bietet mobile Dienste an. Autos, Züge und Flugzeuge werden während der Fahrt mit Medienprogrammen versorgt. Dabei treten neue Erscheinungen auf, die neue systemtechnische Lösungen benötigen.

1.2.3.1 Doppler- Effekt

Wenn das Empfangsgerät sich an den Sender annähert, erhöht sich die Empfangsfrequenz, wenn der Empfänger sich entfernt, wird diese Frequenz niedriger sein, als die Sendefrequenz. Der Unterschied zwischen der Sende- und

Empfangsfrequenz heißt Doppler-Frequenz und lässt sich mit der Gleichung (1.2.7) berechnen.

$$f_D = \frac{\omega_D}{2\pi} = \frac{2 f_{Sender} * \upsilon_D}{c} \qquad (1.2.7)$$

für w_D gilt:

$$\omega_D = \frac{2\omega\upsilon_D}{c} \qquad (1.2.8)$$

f_D - Dopplerfrequenz

ω_D - Phasenänderung

υ_D - Empfängergeschwindigkeit

f_{Sender} – Senderfrequenz

Wenn der Sender mit der Mittenfrequenz f sendet, dann erhält der Empfänger ein Signal mit $f \pm f_D$ Mittenfrequenz. Diese Frequenzverschiebung muss bei Berechnungen in der Mobil- und Satellitenkommunikation berücksichtigt werden. In Quelle [6] wird der Einfluss des Doppler-Effektes detailliert beschrieben.

1.2.3.2 Doppler- Spektrum

Bei einer mobilen Verbindung besteht das Empfangsignal aus mehreren Signalen, die aus verschiedenen Richtungen einfallen und unterschiedliche Amplituden, Frequenzen und Phasen haben. Dieses Signal wird mit der spektralen Funktion der Leistungsdichte $S(\omega)$ (1.2.9) beschrieben. Zum Beispiel in einem *„Single Frequency Network“* (SFN) sind mehrere Sender zu berücksichtigen und darüber hinaus fallen am Empfänger Signale aus mehreren Richtungen ein. So entsteht nicht nur ein, durch eine einzige f_D, zu beschreibender Dopplereffekt, sondern ein Dopplerspektrum.

$$S(\omega) = \begin{cases} \dfrac{E_0}{2 f_m} \dfrac{1}{\sqrt{1 - \dfrac{f}{f_m}}}; & |f| \le f_m; f_m = \dfrac{f_D v}{c} \\ 0; & |f| > f_m \end{cases} \qquad (1.2.9)$$

f_m ist die maximale Doppler-Frequenz.

Die Maximal- und Minimalwerte der Frequenzschwankungen kennzeichnen die Schranken des Dopplerspektrums [7]. Zwischen diesen Grenzwerten konzentriert sich die Leistungsdichte, wie in Abbildung 4 dargestellt.

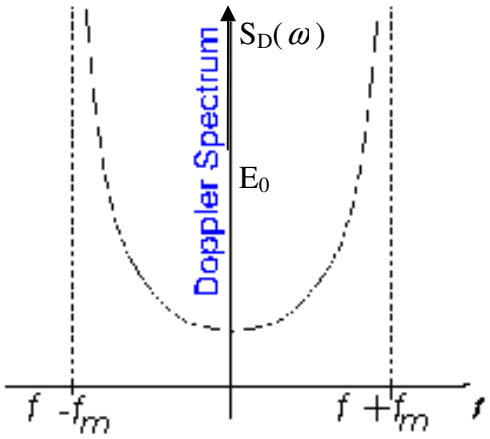


Abbildung 4: Leistungsdichte des Dopplerspektrums („Jakes“ Spectrum) (Quelle: [e])

1.2.4. Aufteilung der Umwelt in der Funktechnik

Die Welt der Wellenausbreitung wird in zwei Gruppen aufgeteilt. Man unterscheidet zwischen natürlichen und unnatürlichen Gebieten. Eine weitere Unterteilung ist beispielsweise auf folgende Weise möglich [8]:

- Natur:
  - frei (z.B. Feld)
  - Flachland
  - hügelig
  - gebirgig
  - bewaldet
- nicht Natur:
  - ländlich /rural/: (Abbildung 5 grün) Die Signalausbreitung in diesen Gebieten hängt hauptsächlich von der natürlichen Vegetation ab.
  - vorstädtisch /suburban/: (Abbildung 5 gelb) Eine mittlere Bebauung mit meist niedrigen Gebäuden.
  - städtisch /urban/: (Abbildung 5 orange) Charakteristisch hierfür sind eine dichte Bebauung, schmale tiefe Häuserschluchten und eine hohe Anzahl von Nutzern auf kleiner Fläche.
  - Im Gebäude

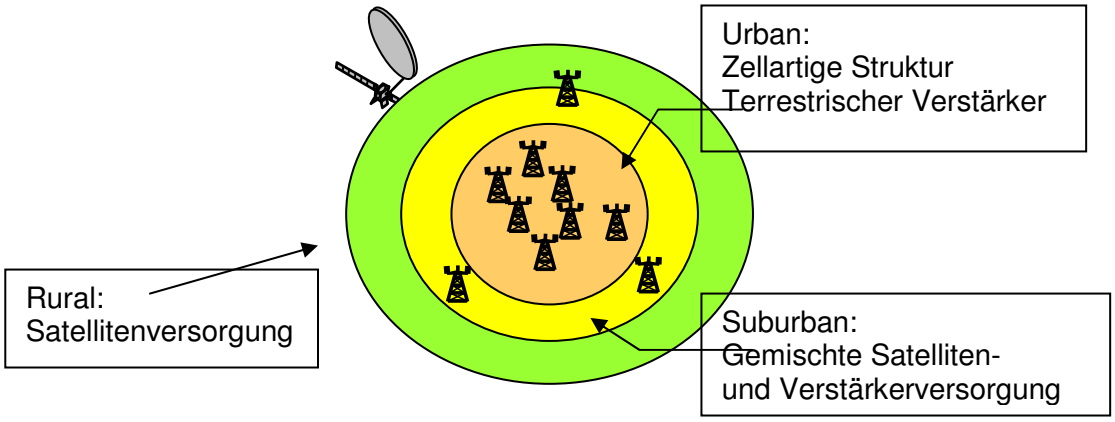


Abbildung 5: Versorgungsgebiete in DVB-SH (Quelle: [f])

Die Umgebung beeinflusst die Änderung der Feldstärke und bestimmt das passende Kanalmodell. Abbildung 5 zeigt den Zusammenhang zwischen Art der Umgebung und der benötigten Dichte von Sendeanlagen. Am Beispiel des DVB-SH-Systems lässt sich erkennen, dass im ländlichen Gelände einzig eine Satellitenversorgung nötig ist. Mit steigender Bebauungsdichte steigt die Anzahl der terrestrischen Sendeanlagen, um die gewünschte nahtlose Abdeckung zu gewährleisten.

1.3 Kanalmodelle

Die Signale, welche die Informationen enthalten, können mit Hilfe von unterschiedlichen Methoden und über verschiedene Übertragungswege verbreitet werden. Datenströme werden über Kabelnetzwerke, terrestrische Sendeanlagen oder via Satelliten weitergeleitet. Diese Übertragungsmedien werden durch verschiedene Kanalmodelle beschrieben.

Die Empfangsantenne empfängt ein Signal aus dem Funkkanal, welches sich aufgrund von Rauschen und Störungen vom gesendeten Signal unterscheidet. Der Empfänger versucht aus diesem Signal das ursprüngliche Signal herzustellen. Dieser Prozess kann nur dann erfolgreich sein, wenn das Empfangssignal einen ausreichend großen Anteil des ursprünglichen Signals enthält. In diesem Fall bedeutet „ausreichend“ eine vordefinierte minimale Sendeleistung, bei der der Empfänger das Signal erfolgreich detektieren kann. Ein anderer Aspekt der erfolgreichen Übertragung ist die Gefahr, dass der Empfänger einige Datenbits aufgrund von Rauschen im Funkkanal falsch detektiert. Um diese Bitfehler erkennen und korrigieren zu können, muss dem gesendeten Signal eine gewisse

Redundanz hinzugefügt werden. Damit reduziert sich die Anzahl der real nutzbaren Datenbits verbunden mit einer Verringerung der Datenrate.

Wenn die Kanaleigenschaften bekannt sind, können die schädlichen Wirkungen vermindert werden. Aus diesem Grund ist die Kanalmodellierung sehr wichtig. Ein passendes Modell, das den Übertragungsweg beschreibt, kann die Zahl der Basisstationen sowie Sendeleistung und Anzahl der Empfangsstationen bestimmen. Ein solches Modell ist auch bei der Systemparameterwahl für Kodierung, Fehlerkorrektur und Modulation hilfreich. Kanalmodelle charakterisieren typische Übertragungswege, auf denen die in Kapitel 1.2 beschriebenen Erscheinungen regelmäßig auftreten.

1.3.1 AWGN Kanal

Der *„Additive White Gaussian Noise“* (AWGN)-Kanal ist ein einfaches Kanalmodell. Hier wird zum Sendesignal ein weißes Rauschen mit Gauß'scher Amplitudenverteilung hinzugefügt. Abbildung 6 zeigt, dass sich der größte Teil der Signalenergie zwischen zwei Grenzwerten U_{eff} und $-U_{eff}$ konzentriert.

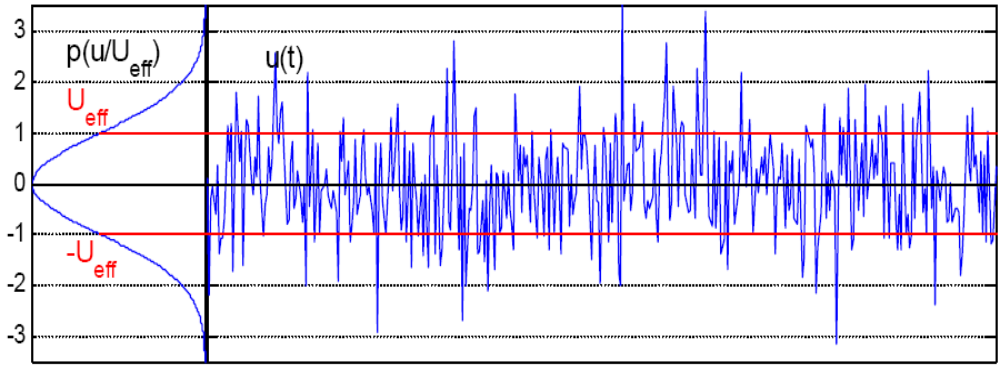


Abbildung 6: Gauss-sche Amplitudenverteilung; Normalverteilung
(Quelle: [g])

Das Empfangssignal y[m] wird berechnet nach (1.3.1) aus der Addition des Sendesignals x[m] und dem zeitabhängigen Rauschen w[m] .

$$y[m] = x[m] + w[m] \qquad (1.3.1)$$

Die Kapazität C des AWGN-Kanals wird mit (1.3.2) bestimmt.

$$C = \frac{1}{2}\log_2\left(1+\frac{S}{N}\right) \qquad (1.3.2)$$

Von einem AWGN Funkkanal kann man beispielsweise ausgehen, wenn es eine direkte Sichtverbindung zwischen Sender und Empfänger gibt und sich weder Sender noch Empfänger bewegen. Mit guter Nährung existiert ein solcher Kanal z.B. für Satellitensysteme, Richtfunkstrecken und WLAN Hot Spots in der Nähe des Senders. Für die Simulation der OFDM- und TDM-Satellitenübertragung wurde das AWGN-Kanalmodell gewählt.

1.3.2 Fading Kanal

Fading sind Schwankungen der Empfangsfeldstärke bei Funkübertragungen. Verursacht wird dieses Fading durch alles, was das Übertragungssignal negativ beeinflusst. Dazu zählen unter anderem die Dämpfung (Kapitel 1.2.1), die in Anhang A aufgezählten Erscheinungen und die Mehrwegeausbreitung (Kapitel 1.2.2).

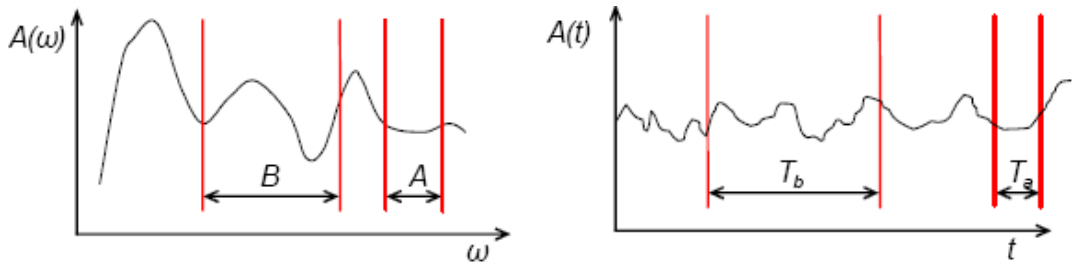


Abbildung 7a: - Frequenzselektive (B)- und frequenzunabhängige (A) Fading;
7b: Fast (T_b)- und Slow (T_a) Fading (Quelle: [h,i])

Man unterscheidet zwischen Fast- und Slow-Fading (Abbildung 7b rechts). Diese Einteilung hängt davon ab, wie schnell sich die Dämpfung im zeitlichen Verlauf ändert. Des Weiteren unterscheidet man zwischen frequenzselektiven und frequenzunabhängigen Fading (Abbildung 7a links). Bei breitbandigen terrestrischen Signalen tritt das freuquenzselektive Fading auf. Differenzierter ist es bei den breitbandigen Signalen vom Satelliten. Bei diesen kommt es meist zum nicht-frequenzselektiven Fading, je nach Bandbreite können die Fading-Erscheinungen auch freuquenzselektiv sein. Dabei wird das Fading dann als

frequenzunabhängig (flach) betrachtet, wenn das Spektrum des Fadings innerhalb der Übertragungsbandbreite gleich verteilt ist (Abbildung 7a links A). Das frequenzselektive Fading erkennt man daran, dass es auf Signale unterschiedlicher Frequenz eine unterschiedliche Wirkung hat (Abbildung 7a links B). Die mathematischen Beschreibungen dazu befinden sich in [6] und [7]. Die frequenzselektive Art verursacht durch die unterschiedlich langen Verzögerungen der Signale, die am Empfänger ankommen, eine nichtlineare Störung und Inter-Symbol-Interferenzen (ISI). Denen kann mit geeigneter Wahl der Modulation (z.B. OFDM) entgegengewirkt werden. Bei allen anderen Arten werden die Störungen mit Leistungsreserven gemindert. Die folgende Auflistung enthält einige Beispiele des Fadings, die für DVB-SH von Bedeutung sind:

- Fast-Fading: Simuliert schnelle Schwankungen des Signalpegels, die sich aus dem Wechsel zwischen konstruktiver und destruktiver Interferenz bei Mehrwegeausbreitung ergeben.
- Slow-Fading: Simuliert langsame Pegeländerungen, die z.B. durch Abschattungseffekte (z. B. in Tunneln) auftreten können.
- Reines Doppler-Fading: Simuliert einen direkten Übertragungspfad, bei dem durch die Bewegung des Empfängers eine Dopplerverschiebung auftritt.
- Rayleigh-Fading: Bildet ein Funkfeld nach, wie es durch Streuung an Hindernissen im Signalweg entsteht (Gebäude, etc.).
- Rice-Fading: Modelliert ein Rayleigh-Funkfeld zusammen mit einem starken direkten Signal.
- Log-Normal-Fading: Simuliert eine zusätzliche, langsame Schwankung der Empfangsamplitude eines bewegten Empfängers, wie sie z.B. durch landschaftliche oder topographische Gegebenheiten (Fahrt durch eine Senke, mehr oder weniger Abschattung durch Bäumen) auftreten kann.

Um die typischen DVB-SH Übertragungswege zu beschreiben, wird eine Mischung der eben genannten Fading-Erscheinungen benötigt. Der terrestrische Pfad kann zum Beispiel mit dem Rice- oder Loo-Modell simuliert werden. Bei Beiden existieren, neben dem direkten Pfad, Echowege mit Rayleigh-Verteilung. Der Unterschied zwischen den beiden Modellen besteht darin, dass der direkte

Pfad im Rice-Kanal konstant ist und im Loo-Kanal log-normal verteilt. Die genaue Beschreibung der Fading-Erscheinungen in terrestrischen DVB-SH Kanälen ist Gegenstand aktueller Forschungsprojekte und Feldversuche. In der Praxis wird momentan das *„Typical Urban 6-Path“* (TU6) genutzt. Es repräsentiert die typischen Bedingungen beim Mobilempfang bei einer Dopplerfrequenz von über 10 Hz. Dieses Kanalmodell definiert insgesamt sechs Ausbreitungswege, davon sind Fünf mit und Einer ohne Verzögerung. Die Beschreibung der möglichen Kanalmodelle befindet sich in [16]. Im vorliegenden Dokument wurde der OFDM-Satellitenweg, neben dem AWGN-Kanal, auch mit dem Rice-Kanalmodell simuliert.

1.3.2.1 Rice Kanal

Beim Rice-Fading kann davon ausgegangen werden, dass während des Mehrwegempfangs ein direkter Signalweg dominiert. (Abbildung 8) Beim Model der Mehrwegeausbreitung setzt sich das Empfangssignal im Allgemeinen aus zwei Anteilen zusammen. Diese sind zum einen der direkte Pfad $r(t)_{LOS}$ und die Echowege $r(t)_{Echo}$. [9]

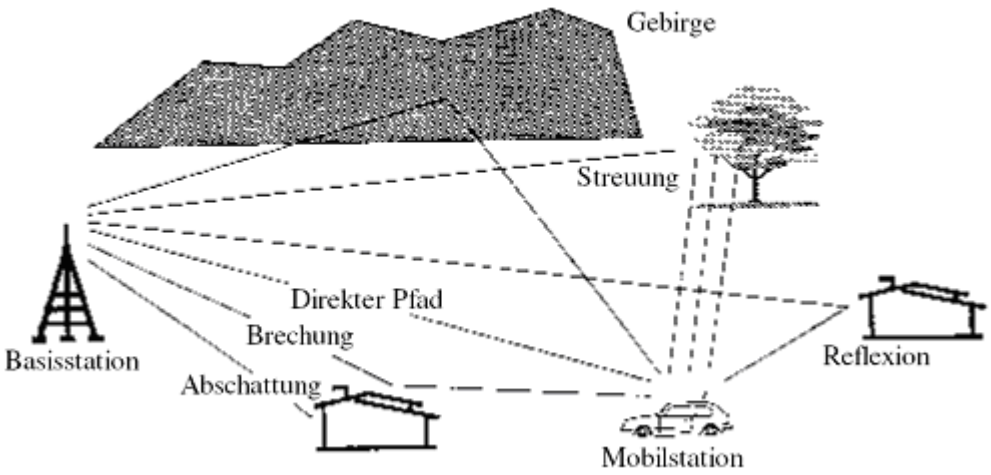


Abbildung 8: Mehrwegeausbreitung im Fadingkanal (Quelle: [j])

Die Gleichung (1.3.3) beschreibt das sinusförmige Sendesignal, welches über ein Mehrwegempfangs-Fading Kanal übertragen wird.

$$s(t) = \cos(\omega_c t + \theta) \qquad (1.3.3)$$

Den Empfänger erreicht eine Linearkombination vieler einzelner Signalanteile (1.3.5). Jede dieser Komponenten besitzt aufgrund eines komplexen Gauß’schen

Prozesses (1.3.4) eine individuelle Dämpfung der Amplitude und Phasen-Verschiebung.

$$u(t) = \sum_{n=1}^{N} c_n \exp[j(\omega_n t + \lambda_n)] \tag{1.3.4}$$

$$r(t) = u(t)\cos(\omega_c t + \theta) \tag{1.3.5}$$

Mit Gleichung (1.3.6) wird die direkte *„Line Of Sight“* (LOS) Komponente beschrieben. [9]

$$r(t)_{LOS} = A\cos(\omega_c t + \omega_n t) \tag{1.3.6}$$

Die Statistik der Echo-Komponente wird durch eine Rayleigh-Verteilung approximiert. Laut [10] lässt sich $r(t)_{Echo}$ mit der Gleichung (1.3.7) bestimmen.

$$r(t)_{Echo} = \frac{\rho_0 * s(t) + \sum_{i=1}^{N} \rho_i e^{-j2\pi\Theta_i} s(t-\tau_i)}{\sqrt{\sum_{i=0}^{N} \rho_i^2}} \tag{1.3.7}$$

mit

N – die Zahl der Echos

ρ_0 – die Dämpfung auf dem direkten Signalweg

ρi – die Dämpfung auf dem Echoweg i

Θ_i – die Phasendrehung im Echoweg i

τ_i – die Zeitverzögerung im Echoweg i

<u>K Faktor für Rice-Kanal</u>

$$K = \frac{\rho_0^2}{\sum_{i=1}^{N} \rho_i^2} \tag{1.3.8}$$

Der Rice-Faktor kennzeichnet das Verhältnis der Leistung auf dem direkten Signalpfad zur Summe der Leistung auf allen Echopfaden. Der Wertebereich dieses Faktors liegt zwischen 0 und ∞. Ein Wert von K=0 bedeutet Rayleigh-Fading. Beträgt K=∞ handelt es sich um einen AWGN-Kanal. Abbildung 9 zeigt einen typischen Verlauf der empfangenen Leistung bei einem Rice-Kanal.

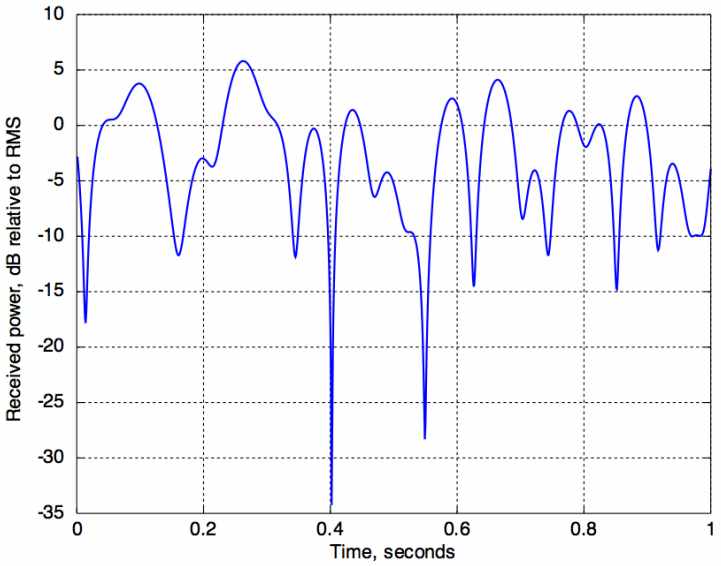


Abbildung 9:Rice-Fading (Quelle [k])

2. Kapitel: Digitale Übertragungstechnik

Das Modell eines digitalen Übertragungssystems zeigt Abbildung 10. Grundlegende Funktionsblöcke sind die Quellenkodierung und vom spezifischen Über-tragungsverfahren abhängige Funktionen, wie die Kanalkodierung, Modulation oder Multiplexing. Sender und Empfänger sind über den Übertragungskanal verbunden. Um den Bezug zwischen Abbildung 10 und DVB-SH herzustellen, wurden den Funktionen konkrete Techniken zugeordnet.

Als Ergebnis der digitalen Datenverarbeitung (A/D-Wandlung, Daten-kompression), digitaler Kodierungs- und Modulationsverfahren, sowie digitaler Übertragungstechniken wurde die effizientere Ausnutzung des elektro-magnetischen Spektrums, der zur technischen Kommunikation verwendeten Funkwellen, erreicht. Im Zuge dieser Digitalisierung wurden höhere Datenraten und schmalere Signalbandbreiten, bei gleichen oder sogar schlechteren SNR wie in analogen Übertragungssystemen, erreicht. Digitale Medientechniken bedeuten nicht nur die Stabilität der Übertragung, sondern ermöglichen auch zahlreiche Zusatzdienste. Dazu gehören unter anderem Datenverschlüsselung, Rückkanal-Übertragung, hochwertige Bild und Tonsignale (HDTV, Dolby Digital) und Mobilempfang.

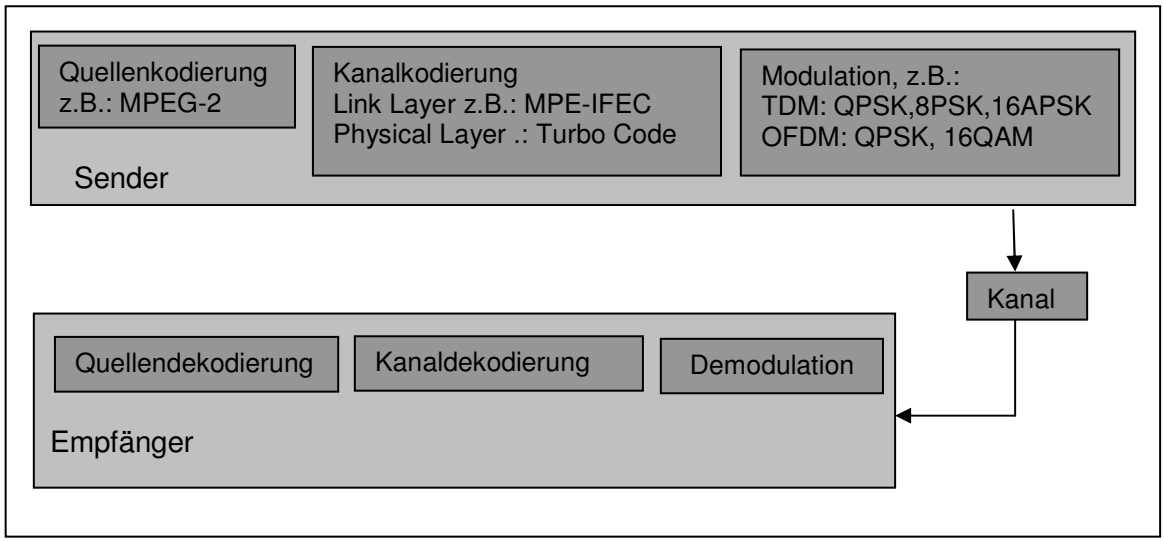


Abbildung 10: Übertragungssystem

2.1 DVB steht für Digitaler Rundfunk

Das europäische Projekt zur Entwicklung des digitalen Fernsehens startete im Jahr 1993 mit dem Namen „Digital Video Broadcasting“ (DVB). Diese Kooperation zwischen Programmanbietern, Geräteherstellern, Netzbetreibern und Behörden hat über 270 Mitglieder. Außerordentliche Mitglieder sind die Normungsinstitute ETSI, CENELEC, ISO und IEC, welche die Spezifikationen erarbeiten. Ein weiteres Mitglied ist die MPEG-Arbeitsgruppe, diese beschäftigt sich mit der Quellenkodierung von Bild- und Tonsignalen. [11]

Für die verschiedenen Übertragungswege und Dienste mussten unterschiedliche Übertragungsverfahren entwickelt und standardisiert werden. Die DVB-Standards regeln die Verfahren zur Übertragung von digitalen Inhalten durch digitale Technik. Die grundlegenden DVB-Standards sind:

- DVB-T (Terrestrisch) Quelle:[12]
- DVB-C (Kabel) Quelle:[13]
- DVB-S (Satellit) Quelle:[14]
- DVB-H (Handheld) Quelle:[15]

2.2 DVB-SH

Der Übertragungsstandard DVB-SH wurde im März 2008 von der ETSI veröffentlicht [11]. Die Kernanwendung ist das Verbreiten digitaler mobiler TV-Dienste. In diesem Sinne arbeitet DVB-SH im S-Band und nutzt eine hybride Netzarchitektur bestehend aus einer Satelliten- und einer terrestrischen Komponente. Digitale TV-Dienste in Echtzeit zu liefern stellt hohe Anforderungen an die Übertragungstechnologie. Um eine gleich bleibend guten *„Quality of Service“* (QoS) zu erreichen, muss der Empfang fehlerrobust sein. Beim mobilen Empfang hat man mit längeren Signalausfällen durch Abschattungen ebenso zu kämpfen wie mit den Effekten der Mehrwegeausbreitung. Spektrale Effizienz und eine gute Fehlerkorrektur gehören zu den Stärken von DVB-SH. In diesem Kapitel werden die einzelnen Systemkomponenten näher betrachtet.

2.2.1 Architektur

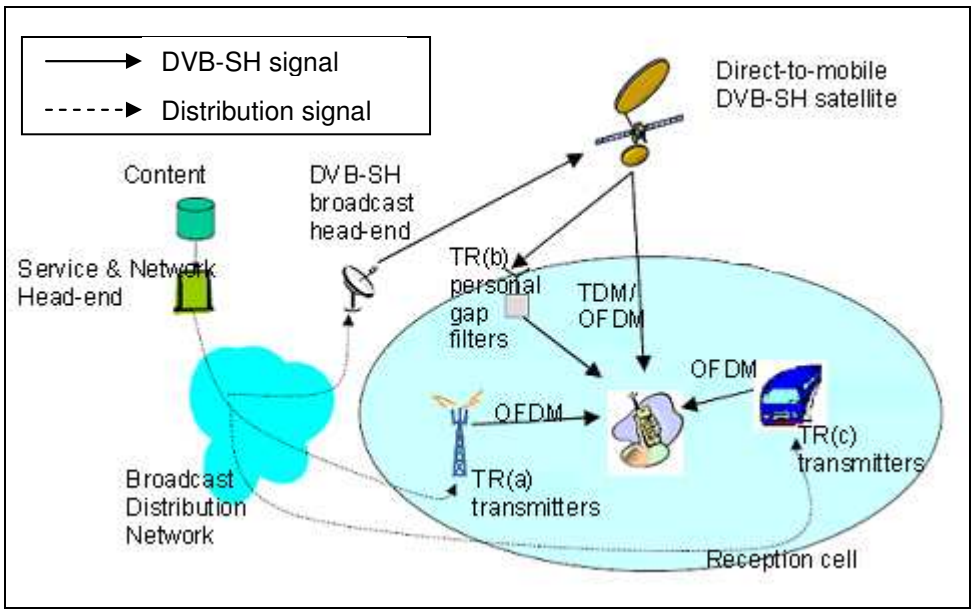


Abbildung 11: DVB-SH Architektur (Quelle: [I])

Die Architektur von DVB-SH, wie in Quelle [16] beschrieben ist, besteht aus einer terrestrischen und einer Satelliten Übertragungskomponente (SC).

Die Übertragung per Satellit stellt die großflächige Abdeckung sicher. Damit ist es möglich eine Vielzahl von Nutzern kosteneffizient zu erreichen. Der Nachteil eines Satellitensignals ist der schlechte Empfang in Gebäuden und in Städten, bedingt durch den grossen Anteil von Abschattungen. Als Lösung dieses Problems wird die DVB-SH Architektur um terrestrische Sendeanlagen (Complementary Ground Component - CGC) am Boden erweitert. Dadurch können die Inhalte auf mehreren Wegen zum Nutzer gelangen. Abbildung 11 veranschaulicht den grundsätzlichen Aufbau von DVB-SH. Die Inhalte werden in ein *„Broadcast Distribution Network“* eingespeist. Dieses Versorgungsnetz verteilt die Nutzdaten weiter auf den DVB-SH-Satelliten, die terrestrischen Sendeanlagen oder auf die mobilen Verstärker. Dabei kommen verschiedenste Technologien, wie xDSL, DVB-S/S2, WiMAX und Glasfaser zum Einsatz. Je nach Einsatzgebiet unterscheidet man drei CGC. Die terrestrischen Sendeanlagen, TR(a) verteilen das Signal in einer zellartigen Netzstruktur ähnlich dem Mobilfunknetz. Sie verstärken das Satellitensignal und bieten die Möglichkeit zusätzlich lokale Inhalte einzuspeisen. Durch die Nutzung des S-Bandes kann für die terrestrische Ausstrahlung die bestehende 3G-Infrastruktur genutzt werden. In Gebäuden werden kleinere Anlagen („personal Gap-fillers“), TR(b) zur Verstärkung des

Satellitensignals genutzt. An diesen Stellen wird kein lokaler Inhalt hinzugefügt, sondern lediglich eine Verstärkung oder Frequenzumwandlung vorgenommen. Schließlich gibt es noch mobile Sendestationen, TR(c), die in Transportmitteln, wie Zügen und Autos, eingesetzt werden. Hier können ebenfalls lokale Inhalte eingespeist werden.

Auf der physikalischen Schicht unterscheidet man zwischen zwei Konfigurationen. Die Variante DVB-SH-A und -SH-B. Die Modulationsparameter für die beiden Konfigurationen sind in Tabelle 1 aufgelistet. Die genannten Parameter und ihre Wirkung sind in [17] erklärt.

SH-A verwendet OFDM für die terrestrischen und satelliten-Übertragungswege. Dadurch ist es möglich die SH-A Konfiguration als Gleichwellennetz (SFN) zu betreiben. Dies bietet die Vorteile, dass die Sendeanlagen frequenzökonomisch arbeiten und die einzelnen Signale sich nicht gegenseitig stören. Der Nachteil besteht jedoch darin, dass *„Forward Error Correction*“ (FEC), *„Channel Interleaver“*, Modulation und *„Guard Interval“* nicht individuell an SC und CGC angepasst werden können.

Die zweite Variante DVB-SH-B arbeitet mit TDM auf dem satellitischen Übertragungsweg und OFDM auf der terrestrischen Komponente. Durch die Verwendung zweier unterschiedlicher physikalischer Schichten benötigt SH-B je ein Frequenzband für SC und CGC. Der große Vorteil von dieser Variante ist die individuelle Anpassungsfähigkeit jeder Komponente. Dadurch können die Anforderungen für jeden Übertragungsweg optimal erfüllt werden.

	OFDM (SH-A und SH-B)				TDM (SH-B)
IFFT-Modus	8K	4K	2K	1K	–
Kanalband-breite	8 / 7 / 6 / 5 MHz			1,7 MHz	8 / 7 / 6 / 5 / 1,7 MHz
Signalband-breite	7,61 / 6,65 / 5,70 / 4,75 MHz			1,52 MHz	roll-off 0,15 / 0,25 / 0,35
OFDM-Träger	8192	4096	2048	1024	–
Anzahl Träger	6817	3409	1705	853	–
Schutzintervall	1/4, 1/8, 1/16, 1/32				–
Konstellation	QPSK, 16QAM ($\alpha = 1$), 16QAM ($\alpha \neq 1$)				QPSK, 8PSK, 16APSK

Tabelle 1: Modulationsparameter von DVB-SH (Quelle: [16])

2.2.2 DVB-SH Empfänger

In diesem Abschnitt wird die allgemeine Architektur von DVB-SH Empfängern beschrieben. An einen DVB-SH Empfänger werden die folgenden Anforderungen gestellt:

- Es muss der nahtlose Empfang des Signals beim Übergang zwischen Satelliten- und terrestrischen Sendeanlagen gewährleistet sein.
- Aufgrund des mobilen Einsatzes müssen die Geräte energiesparend sein und einen Empfangsmodus besitzen, indem sie Signale aus verschiedenen Quellen parallel verarbeiten können (*„Diversity“*).
- Die Systemkomponenten müssen zusammen mit anderen Standdards für die Funkübertragung, wie zum Beispiel GSM, UMTS, WLAN und Bluetooth, im Endgerät untergebracht werden.
- Insgesamt muss der Empfänger robust sein, gegenüber den Einflüssen der 2G/3G Sendeanlagen und/oder den zahlreichen ISM-Systemen.

Je nach verwendeter Systemvariante wird zwischen einer DVB-SH-A und einer SH-B Empfängerarchitektur unterschieden. Die Abbildungen 12 und 13 stellen den schematischen Aufbau der Komponenten dar. Es ist erkennbar, dass der SH-B Empfänger um eine TDM-Demodulator erweitert wurde.

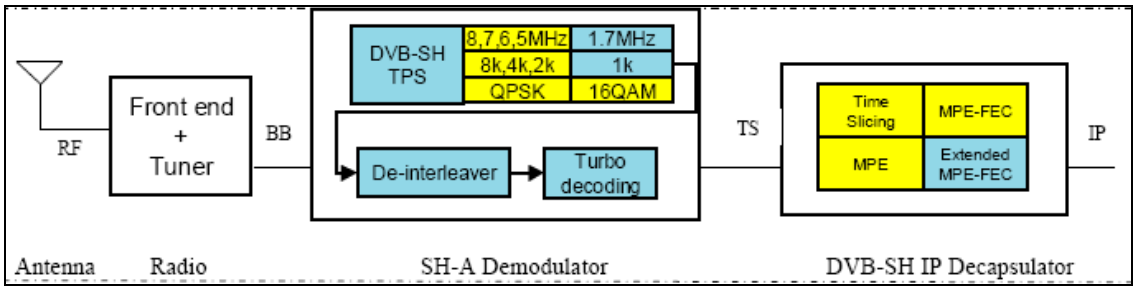


Abbildung 12: DVB-SH-A Empfänger (Quelle: [m])

Für OFDM besitzen beide Varianten eine identische Konfiguration, dadurch können SH-B Empfänger auch in einer SH-A Systemarchitektur verwendet werden. Grundsätzlich sind die SH-A Empfänger darauf ausgelegt in einem SFN zwischen Satellit und terrestrischen Sender genutzt zu werden. Es ist jedoch von Vorteil, wenn sie auch in einer *„Multiple Frequency Network“* (MFN) Umgebung arbeiten können, die für das OFDM-Signal vom Satelliten ein anderes Sub-Band

benutzt als für die terrestrische Ausstrahlung. Genauere Betrachtungen über die Empfängertechnik stehen in [16].

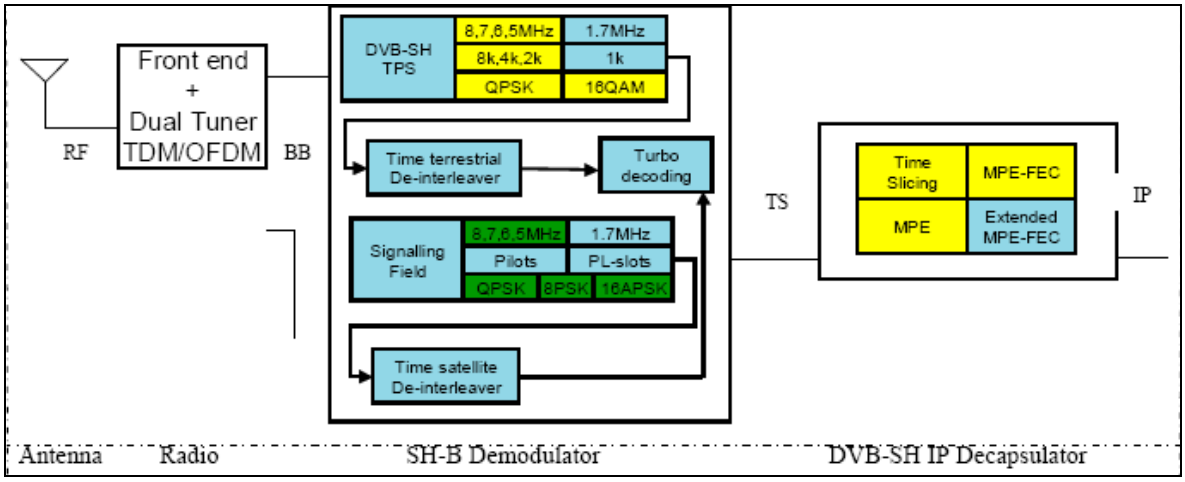


Abbildung 13: DVB-SH-B Empfänger (Quelle: [n])

2.2.3 Übertragungswege

Die unterschiedlichen Übertragungswege der DVB-SH Technik verlangen eine getrennte Betrachtung der Ausbreitungscharakteristiken von Satelliten- und terrestrischen Kanälen. Bezüglich DVB-SH unterscheidet man zwischen drei verschiedenen Umgebungsarten, die in Kapitel 1.2.4 beschrieben werden.

- ländlich: Die meist großflächigen Areale werden durch den Satelliten versorgt.
- vorstädtisch: Hier wird die lückenlose Signalverbreitung zu gleichen Anteilen durch die Satelliten- und terrestrischen Sendeanlagen erreicht.
- städtisch: In einer solchen Umgebung ist eine Versorgung per Satellit nicht lückenlos möglich. Daher werden CGC eingesetzt, um die nötige Signalqualität zu gewährleisten.

Allgemein teilt man die, auf dem Übertragungsweg, auftretenden Effekte in drei Typen ein. Diese Gruppierung wird anhand der Entfernung vorgenommen und gilt für Land Mobile Satellit (LMS)- und terrestrische Kanäle. [16]

- Sehr langsame Signalleistungsschwankungen verursacht der Pfadverlust. Dieser wurde in Kapitel 1.2.1. beschrieben.

- Etwas schnellere Leistungsänderungen treten aufgrund von Abschattungen durch Gebäude, Bäume, etc. auf.
- Schließlich treten kleinräumig die Effekte des Mehrwege-Fadings auf (Schnelles Fading).

Um die Verluste, mit denen auf den beiden DVB-SH Übertragungswegen gerechnet werden muss, näher betrachten zu können, werden die grundlegenden Gleichungen des Link Budget für diese Übertragungswege erklärt. Als Quelle wurde [16] verwendet.

Der minimale Empfangssignalpegel, mit dem der Empfänger am Ende eines terrestrischen Übertragungsweges das Signal dekodieren kann wird mit der Gleichung (2.2.1) bestimmt.

$$P_{\min\ Empfänger} = NF + 10 * \lg(k * T_0 * B) + \frac{C}{N} \qquad (2.2.1)$$

NF – Rauschzahl des Empfängers
k – Boltzmann Konstante
T_0 – Empfängertemperatur
B – Bandbreite des Rauschen
C/N – Minimales C/N des Empfängers

Die ersten beiden Summanden charakterisieren den Empfänger. Der C/N-Wert ist das minimal erforderliche Träger-Rausch-Verhältnis. Diese Größe ist von den Kanaleigenschaften abhängig. Um den C/N auszurechnen, wird die Gleichung (2.2.2) verwendet. Nähere Betrachtungen der Wirkung dieses Wertes werden im Kapitel 2.3 erläutert.

$$\left(\frac{C}{N}\right) = EIRP_{Sender} - PL - Ms - L_b - L_p - L_v + G_{SFN} - NF - 10 * \lg(k * T_0 * B) + G_{Empfänger} \qquad (2.2.2)$$

PL - Pfadverlust
Ms - Abschattungsreserve
L_b - der Verlust der entsteht, wenn die Welle durch ein Gebäude muss
L_v - der Verlust der entsteht, wenn die Welle durch Fahrzeuge muss
L_p - Verlust wegen der Änderung der Wellenpolarisation
G_{SFN} – Gewinn aufgrund der Verwendung in einem SFN

EIRP bedeutet *„Equivalent Isotropic Radiated Power“* und wird für den Sender mit der Gleichung (2.2.3) ausgerechnet.

$$EIRP_{Sender} = P_{Sender}[dBm] + G_{Sender} \qquad (2.2.3)$$

Auf einem Satelliten-Downlink gibt es ebenfalls zahlreiche Einflussfaktoren. Die DVB-SH-Satelliten bewegen sich auf einer geostationären Bahn um die Erde. Deshalb spielen die Verluste im Freiraum L_{FS} eine wichtige Rolle. Detaillierte Informationen über die Umlaufbahnen von Satelliten befinden sich in Anhang B. Des Weiteren kommen auf der satellitischen Übertragungsstrecke atmosphärische Verluste L_A hinzu. Das Verhältnis aus Antennengewinn zu Rauschtemperatur $\left(\frac{G}{T}\right)_{Empfänger}$ des Empfangsgerätes spielt auch eine Rolle. Mit (2.2.4) kann der, unter Berücksichtigung der eben genannten Einflussfaktoren, notwendige C/N errechnet werden. Für $EIRP_{SAT}$ wird (2.3.5) eingesetzt. Dieser Wert setzt sich zusammen aus der Sendeleistung P_{SAT} und dem Gewinn G_{SAT} des Satelliten, sowie Verlusten aufgrund einer nicht optimalen Ausrichtung des Satelliten L_T und Verlusten auf dem Weg vom Sender zur Empfängerantenne L_{FTX}.

$$\left(\frac{C}{N}\right)_{DownLink} = \frac{1}{kB} EIRP_{SAT} \frac{1}{L_{FS} * L_A} \left(\frac{G}{T}\right)_{Empfänger} \qquad (2.2.4)$$

$$EIRP_{SAT} = \frac{P_{SAT} * G_{SAT}}{L_T * L_{FTX}} \qquad (2.2.5)$$

Für beide Übertragungsarten gilt das AWGN-Kanalmodell als idealer Kanal. Das praktische Modell für den terrestrischen Weg und damit für eine OFDM Signalübertragung ist das Rice-Kanalmodell.

2.3 Qualität des Übertragungssystems

Das Ziel der Übertragungstechnik ist, ein Maximum an Daten fehlerfrei zu übertragen. Die höchste Bitrate, welche fehlerfrei über einen Kanal geschickt werden kann, heißt Kanalkapazität und wird durch das Shannon-Hartley-Gesetz definiert. Laut dieses Gesetzes wird die obere Grenze der Kanalkapazität durch die zwei Kanaleigenschaften Bandbreite und Signal-zu-Rausch Verhältnis

bestimmt. Um Informationen effizient innerhalb einer bestimmten Bandbreite übertragen zu können, werden Quellenkodierung, Kanalkodierung und Modulationsverfahren eingesetzt. Als wichtiges Maß für die Qualität einer Nachrichtenverbindung hat sich der Störabstand SNR (*„Signal to Noise Ratio“*) etabliert, wobei S für die Basisbandsignalleistung und N für die Störleistung steht. Mit Gleichung (2.3.1) kann der SNR in dB ausgerechnet werden.

$$SNR = 10\lg\left(\frac{S}{N}\right) \qquad (2.3.1)$$

Die Wahrscheinlichkeit (BER), dass ein Bit verfälscht wird ist umso größer, desto kleiner der SNR ist. Modulationsformate höherer Ordnung können nur in Kanälen eingesetzt werden, die einen hohen SNR haben. Abbildung 14 zeigt die Quadratur-Amplituden-Modulation in verschiedenen Ordnungen. Dabei lässt sich erkennen, dass sich, mit steigender Ordnung, der Abstand zwischen den Zuständen verringert. Dadurch ist das Modulationsverfahren empfindlicher gegenüber Störeinflüssen, da der Empfänger die Zustände schwerer eindeutig detektieren kann.

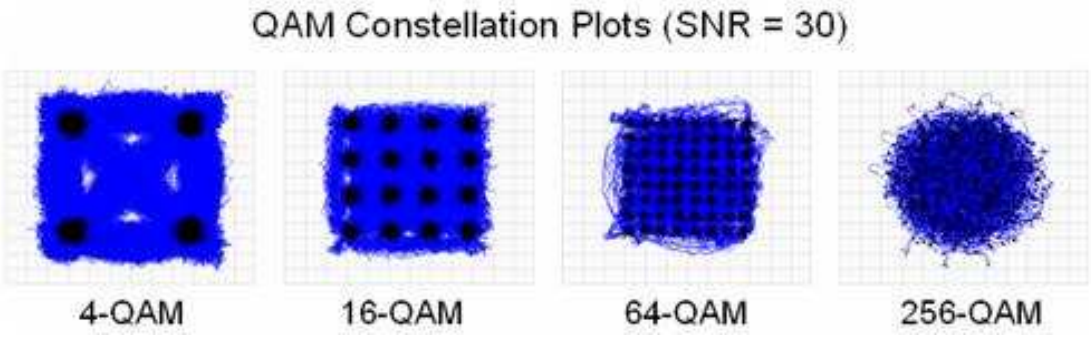


Abbildung 14: Konstellationsdiagramme für Modulationsformate steigender Ordnungen (Quelle: [o])

Um die Qualität einer Übertragungsstrecke beurteilen zu können, werden einige Begriffe gebraucht. Neben dem SNR, der sich auf das gesamte Signal bezieht, beschreibt das Träger-Rausch-Verhältnis (C/N_0) den Zusammenhang zwischen der empfangenen modulierten Trägerleistung und der durchschnittlichen empfangenen Rauschleistungdichte. C/N_0 zeigt wie stark die Trägersignalleistung im Verhältnis zur Rauschleistungdichte im Kanal ist. Je größer dieser Wert ist, desto sicherer ist die Übertragung. Oder umgekehrt, je robuster ein System gegenüber Übertragungsfehlern ist, desto besser können die Signale, auch bei

kleinem C/N_0, mit ausreichend guter Qualität dekodiert werden. Ein Empfänger benötigt ein bestimmten minimalen C/N_0, um aus dem Empfangssignal das ursprüngliche Sendesignal mit genügend kleiner BER herstellen zu können. Mit einigen wichtigen Faktoren kann das C/N_0 positiv beeinflusst werden. Dazu gehört eine angemessene hohe Abstrahlleistung (EIRP), sowie eine geringe Dämpfung und Eigenrauschleistung der Komponenten im Empfangsprozess. Des Weiteren ist speziell im Satellitenempfangsszenario eine genaue Einrichtung von Azimuth, Elevation und Polarisation der Empfängerantenne notwendig.

Mit Hilfe der Gleichung (2.3.2) kann man aus C/N_0 das Verhältnis zwischen der Energie pro Bit E_b und N_0 ermitteln. Dafür subtrahiert man von der modulierten Signalenergie C die Energien, welche aufgrund der Kodierung und der Symbolerzeugung hinzugefügt wurden.

$$\left(\frac{E_b}{N_0}\right) = \left(\frac{C}{N_0}\right) - 10 * \lg c - 10 * \lg m \qquad (2.3.2)$$

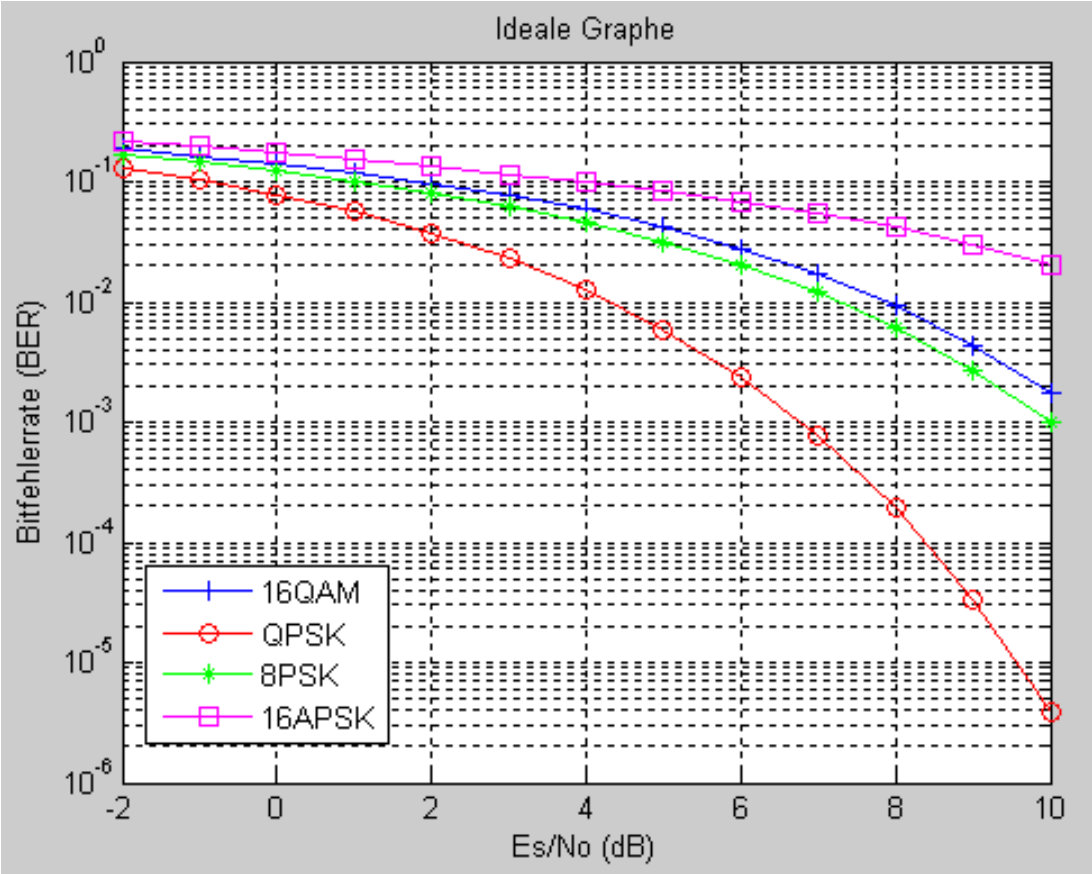
c - Kodierungsrate

m - Anzahl der Informationsbits pro Symbol

$$\left(\frac{E_s}{N_0}\right)_{Modulation} = \left(\frac{C}{N_0}\right) - 10 * \lg c \qquad (2.3.3)$$

Wenn die Wirkung der Modulation (2.3.3) oder der Kodierung (2.3.4) untersucht werden muss, wird mit dem Verhältnis aus Energie pro Symbol E_s zu N_0 gerechnet.

$$\left(\frac{E_s}{N_0}\right)_{Kodierung} = \left(\frac{C}{N_0}\right) - 10 * \lg m \qquad (2.3.4)$$



Graph 1: Verhalten der BER bei verschiedenen Modulationsarten

Ein weiteres Qualitätsmerkmal einer Übertragungsstrecke ist die BER in Relation zum E_s/N_o-Verhältnis. Graph 1 veranschaulicht diesen Zusammenhang am Beispiel eines idealen AWGN-Kanalmodells mit verschiedenen Modulationsverfahren, aber ohne Kanalkodierung. Das Diagramm beschreibt in welchem Ausmaß sich die Qualität verschlechtert, wenn sich E_s/N_0 verringert. Genau diese Darstellung von Messergebnissen ist Kernbestandteil der vorliegenden Bachelorarbeit. In Kapitel 3.6 wird die Durchführung der Messungen erläutert und in Kapitel 3.7 folgt die Darstellung der Resultate.

3. Kapitel: Automatisiertes Messsystem

Ziel dieses Projektes ist es, die Qualität des DVB-SH Empfängers zu beurteilen. Die Kernaufgabe dieser Bachelorarbeit ist die Implementierung einer Test-Software, welche die Steuerung der Messgeräte sowie die Speicherung und Auswertung der Messdaten übernimmt. Das angewendete Testszenario hat die Aufgabe bei verschiedenen Systemparametern und Kanalmodellen die Rate der Fehlerhaften Pakete in Abhängigkeit von C/N_0-Werte zu messen.

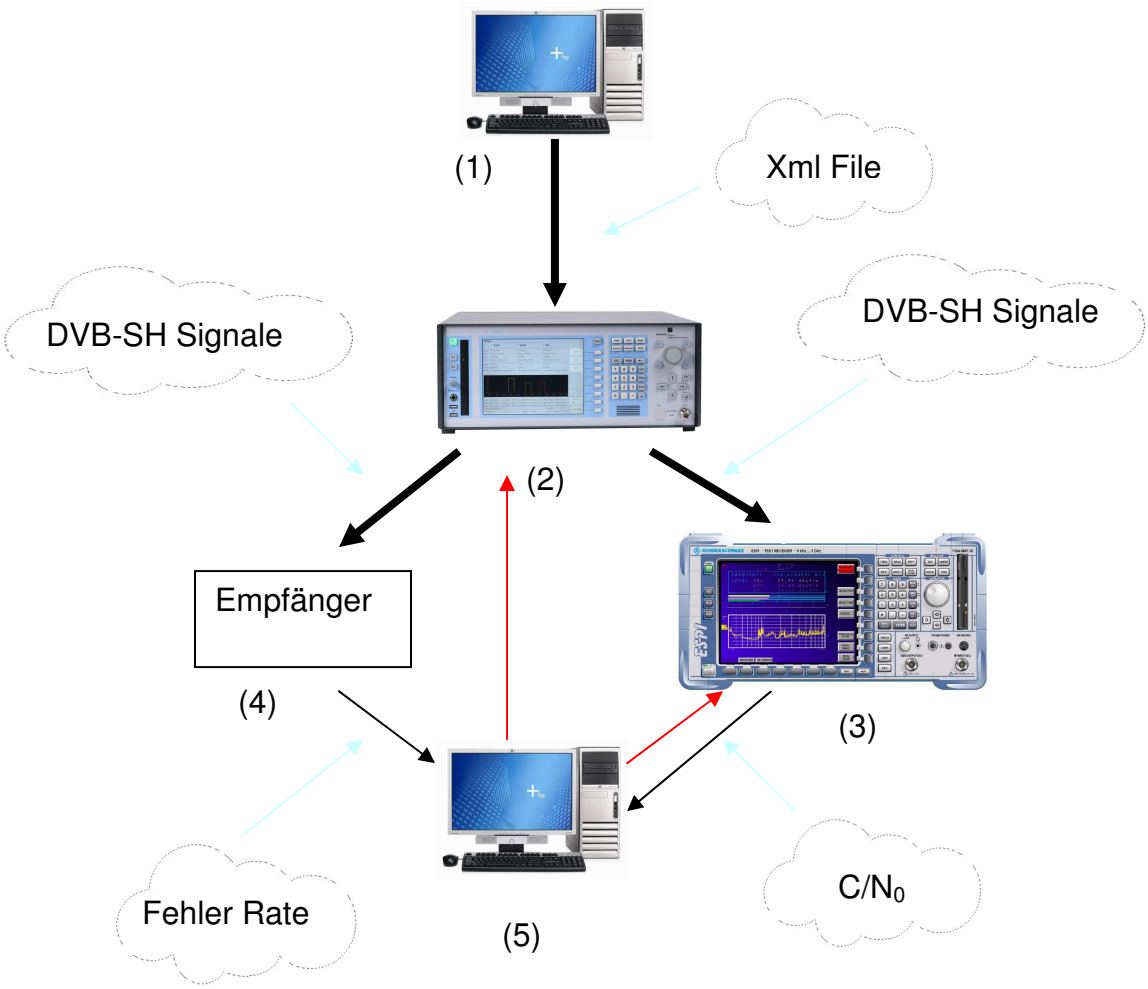


Abbildung 15: Messaufbau

Abbildung 15 beschreibt grafisch die einzelnen Schritte des implementierten Meßverfahrens. Der Messaufbau besteht aus einem Spektrumanalysator ESPI der Firma Rohde & Schwarz, einem Testempfänger des Fraunhofer IIS und einem Signalgenerator DT4500 ebenfalls gebaut vom Fraunhofer IIS. Der DT4500 übernimmt die Rolle des Modulators und der Sendeanlage (Tabelle 2).

Name	Typ	Hersteller
ESPI	Funkstörmessempfänger	R&S
DT4500	DVB-SH Signalgenerator	Fraunhofer Gesellschaft
Testempfänger	DVB-SH Empfänger	Fraunhofer Gesellschaft

Tabelle 2: Verwendete Messgeräte

Gesteuert wird der gesamte Messvorgang über eine Softwarelösung. In den folgenden Kapiteln wird detaillierter auf die einzelnen Komponenten eingegangen. Die folgenden Erläuterungen können der in der Darstellung (Abbildung 15) angegebenen Nummerierung zugeordnet werden und beschreiben die stattfindenden Vorgänge.

(1) - Für die Messung werden in einer Datei gespeicherte MPEG-TS-Pakete verwendet. Mit Hilfe einer XML-Datei wird der TS aufgerufen und zusammen mit den Konfigurationsparametern dem DT4500 Signalgenerator übergeben.

(2) - Der DT4500 Signalgenerator erzeugt OFDM- und TDM-Signale im S-Band. Dazu nutzt er die Konfigurationsinformationen aus der XML-Datei und die TS-Pakete, welche die Nutzdaten enthalten.

(3) - Der ESPI Spektrumanalysator überwacht das Signal und misst den C/N-Wert, der am Empfänger anliegt Die Messergebnisse werden zu einem Computer weitergeleitet und gespeichert.

(4) - Parallel zur C/N-Messung werden die OFDM- und TDM-Signale vom DT4500 zu einem Testempfänger weitergeleitet. Nach dem Empfang wird eine Statistik erstellt in der die Anzahl der insgesamt gesendeten und fehlerhaften Pakete gespeichert sind.

(5) - Das Messprogramm fasst alle Schritte des Messverfahrens zusammen. Dabei übernimmt es die Steuerung der beiden Messgeräte sowie Speicherung und Auswertung der Messdaten.

Tabelle 3 beinhaltet die verwendeten Konfigurationsparameter für die Test-durchläufe. Im Detail ist ein Testdurchlauf definiert als eine bestimmte Kombination aus den möglichen Parametern. Beispielsweise wird OFDM mit einer Coderate R von 1/3, einer Bandbreite B von 5 MHz und einem Guard Interval von

0,25 verwendet. Diese Konfiguration wird insgesamt viermal durchlaufen. Je ein Testlauf mit den Modulationsarten QPSK und 16QAM, sowie ein Lauf mit einem AWGN- und Rice-Kanalmodell.

OFDM		TDM	
R = 1/3	R = 2/3	R = 1/3	R = 2/3
B = 5 MHz	B = 5 MHz	B = 5 MHz	B = 5 MHz
G.I.= 0.25	G.I.= 0.25	Roll-off Faktor: 0.15	Roll-off Faktor: 0.15
Modulation: QPSK, 16QAM	Modulation: QPSK, 16QAM	Modulation: QPSK, 8PSK, 16APSK	Modulation: QPSK, 8PSK, 16APSK
Kanalmodelle: AWGN, Rice	Kanalmodelle: AWGN,Rice	Kanalmodelle: AWGN	Kanalmodelle: AWGN
Interleaverlänge: ≈ 5 sec.		Interleaverlänge: ≈ 5 sec.	

Tabelle 3: Verwendete Konfigurationsparameter für Testläufe

3.1 ESPI Spectrumanalysator

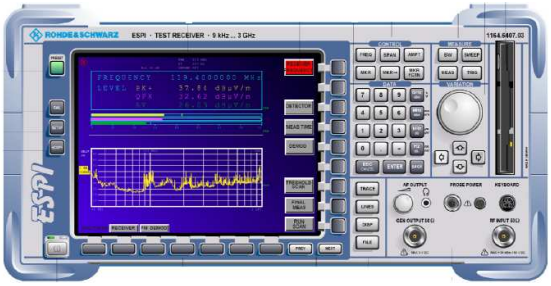


Abbildung 16: ESPI (Quelle: [p])

Grundsätzlich ist der ESPI ein Funkstörmessempfänger. Im Frequenzbereich können Signale ab 9 kHz bis zu 3 GHz detektiert werden. Das Gerät kann mit der LAN-Schnittstelle über ein lokales Netzwerk ferngesteuert werden. Der ESPI unterstützt die SCPI-Version 1997.0 (Standard Commands for Programmable Instruments).

3.1.1 C/N_0-Messung

Die C/N_0-Messung wird mit dem Spektrumanalysator ESPI durchgeführt. Dieses Messgerät hat für die Kanal- und Nachbarkanal-Leistungsmessungen eine extra

Funktion. Anhang III. zeigt die dazugehörende Menüübersicht. Während der Messung wird die *„Integrated Bandwidth Method“* (IBW-Methode) verwendet. Diese Methode wird in [19] ausführlich beschrieben. Für die Ermittlung der benötigten C/N_0-Werte wird für C innerhalb der *„Tx Channel Bandwidth“* und für N_0 innerhalb der *„Adjacent Channel Bandwidth“* die Leistung gemessen. Als Ergebnis erhält man den Unterschied zwischen den gemessenen Leistungen. In linearer Form kann es mit (3.1.1) dargestellt werden.

$$\frac{C_{TxChannelBandwidth} + N_{AdjacentChannelBandwith}}{N_{AdjacentChannelBandwith}} \qquad (3.1.1)$$

Am Ende wird dieses Resultat korrigiert, um den C/N_0-Wert zu erhalten. Während eines Testdurchlaufs ist es von Vorteil, wenn die beiden Messbandbreiten die gleichen Werte besitzen. Dadurch können die gemessenen Leistungswerte, nach der angesprochenen Korrekturberechnung, als C/N_0-Verhältnis verwendet werden.

Für die Beispielmessung eines TDM-Signals wird die verwendete Signalbandbreite mit B = 5 MHz und der Roll-off Faktor mit α = 0,15 festgelegt. Mit Hilfe dieser beiden Größen können die Messbandbreiten nach Gleichung (3.1.2) errechnet werden.

$$\text{„Channel Bandwidth“} = \left[B_{[MHz]} * (1-\alpha) \right] * (1-\alpha) = 3,6125 [MHz] \quad (3.1.2)$$

Die ermittelte *„Tx Channel Bandwidth"* von 3,6125 MHz stellt sicher, dass der Signalverlauf innerhalb dieser Bandbreite konstant ist. Dadurch kann die Bandbreite von 3,6125 MHz auch für die *„Adjacent Channel Bandwidth"* verwendet werden.

Bei der C/N_0-Messung eines OFDM-Signals muss lediglich eine Messbandbreite innerhalb der Signalbandbreite festgelegt werden, in der der Signalverlauf nahezu konstant ist. Diese *„Tx Channel Bandwidth"* wird ebenfalls als *„Adjacent Channel Bandwidth"* verwendet.

Abbildung 17 zeigt, welche Werte vom ESPI geliefert werden. Es ist erkennbar, dass die Rauschleistung in einer oberen („upper“) und in einer unteren („lower“) *"Adjacent Channel Bandwidth"* gemessen wird. Um den relevanten Wert zu bekommen, muss aus den beiden Rauschleistungen ein Mittelwert gebildet

werden. In Kapitel 3.6.1 wird die Notwendigkeit dieser Mittelwertbildung anhand praktischer Messergebnisse nachgewiesen.

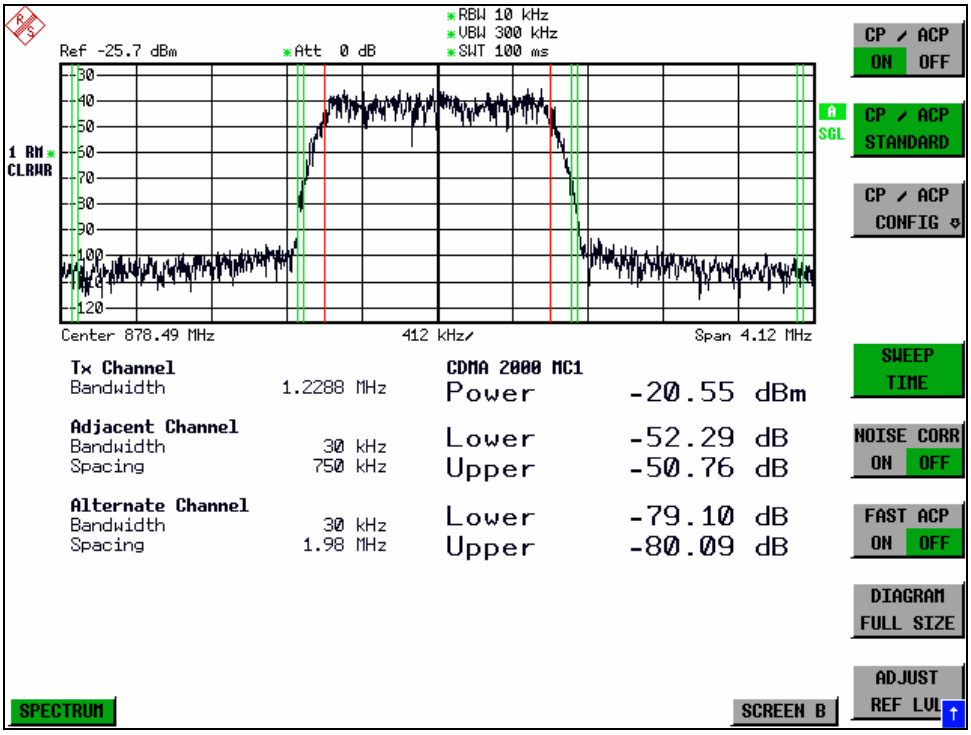


Abbildung 17: Nachbarkanalleistungsmessung nach der IBW-Methode (Quelle [q])

3.1.2 Kommunikation mit ESPI

Quelltext 1 zeigt wie der Kontakt mit dem ESPI über einen Java-Socket realisiert wird. Dazu muss eine Adresse definiert werden. Diese besteht aus einem Hostnamen und einem Port. Das Senden und Empfangen von Befehlen oder Daten erfolgt über die Eingabe- (iStream) und Ausgabedatenströme (oStream).

```
1  socket = new Socket(this.hostname, this.port);
2          iStream = socket.getInputStream();
3          oStream = socket.getOutputStream();
```

Quelltext 1: Java-Socket

Die, in Quelltext 2 definierte, Methode *„sendCommand*" ermöglicht es einen Befehl zu senden. Der Eingabewert *„cmd"* ist ein String und muss als gültige SCPI-Syntax eingegeben werden. Eine Übersicht der gültigen SCPI-Befehle für das ESPI befindet sich in [19].

```
1     public void sendCommand(String cmd) throws IOException {
2         System.out.println("ESPI Command: " + cmd);
3         cmd += "\n";
4         oStream.write(cmd.getBytes());
5     }
```

Quelltext 2: Funktion *„sendCommand*"

Der in Quelltext 3 dargestellte Java-Programmabschnitt veranschaulicht am Beispiel des Parameters *„Center Frequenz“*, wie Werte beim ESPI mittels eines Java-Befehls eingestellt werden können. Andere ESPI Funktionen werden mit ähnlicher Syntax in Java realisiert.

```
1  public void setCenterFreq(double freqHz) {
2      executeCmd("FREQUENCY:CENTER" + " "
3                                    + String.valueOf(freqHz));
4      executeResponse("FREQUENCY:CENTER", freqHz);
5      }
```

Quelltext 3: Java-Befehl für Einstellung der Center Frequenz

3.2 DT4500

Abbildung 18: DT4500 (Quelle: [r])

Der DT4500 ist ein Signalgenerator für den DVB-SH Standard. Er ist in der Lage komplexe digitale Basisbanddateien von einer internen Festplatte abzuspielen. Zudem bietet der DT4500 eingebaute TDM- und OFDM-Echtzeit-Encoder (Inner Physical layer) [18]. Jeder arbeitet mit einem MPEG-Transport-Stream als Eingangssignal. Alle Möglichkeiten können parallel mit einer eingebauten Störgeräuschquelle arbeiten. Der DT4500 unterstützt zwei verschiedene Ausgangssignale. Er ist in der Lage entweder direkt drei erzeugte Signale in ein Frequenzband von 2.14 GHz bis 2.20 GHz (S-Band) umzuwandeln oder das erzeugte Signal auf einer Zwischenfrequenz auszugeben. Der DT4500 wird über ein Webinterface konfiguriert. Die Adresse lautet "http://<dt4kXXX>". Für XXX wird die IP-Adresse oder der Hostname des DT4500 eingetragen. Das Setzen von

Parametern aus dem Testprogramm heraus erfolgt über das Hypertext Transfer Protocol (HTTP) mit der Syntax 1. Für das Abfragen der gesetzten Parameter gilt Syntax 2.

```
http://DT4500_Hostname/remote-access?Parameter_Name=Value
```

Syntax 1: Parameter setzen im DT4500

```
http://DT4500_Hostname/remote-access?GetParameter=Parameter_Name
```

Syntax 2: Parameter abfragen im DT4500

3.2.1 Kommunikation mit DT4500

Quelltext 4 zeigt, wie der Kontakt mit dem Signalgenerator DT4500 aufgebaut wird und Befehle gesendet werden. Die Adressierung erfolgt wie beim ESPI über eine Kombination aus Hostname und Port. In Zeile 2 wird die Adresse in einem String-Objekt zusammengefasst. Die Zeilen 5 und 8 bauen die Verbindung mit Hilfe vordefinierter Java-Klassen auf.

```
1  public String sendCommand(String cmd) {
2          String url = new String("http://" + host + ":"
3          + Integer.toString(port) + "/remote-access?" + cmd);
4          System.out.println("DT4500K URL:" + url);
5          GetMethod get = new GetMethod(url);
6          client = new HttpClient();
7          try {
8              client.executeMethod(get);
9              Thread.sleep(100);
10         } catch (Exception e) {
11             e.printStackTrace();
12         }
13
```

Quelltext 4: Parameter Eingabe

Die Parameterabfrage vom DT4500 durch das Testprogramm wird in Quelltext 5 angegeben.

```
14          try {
15              String response = get.getResponseBodyAsString();
16              System.out.print("DT4500K Response: " + response);
17              get.releaseConnection();
18              return response;
19          } catch (Exception e) {
20              e.printStackTrace();
21              get.releaseConnection();
22              return null;
23          }
24      }
```

Quelltext 5: Parameter Ausgabe

3.2.2 Die XML-Konfiguration

Die XML-Datei konfiguriert den Kanalkoder des DT4500 als *„Outer Physikal Layer“* (OPL) und den Modulator des DT4500 als *„Inner Physical Layer“* (IPL). Das folgende Beispiel zeigt den grundsätzlichen Aufbau der XML-Datei, mit Parametern einer Messung. An dieser Stelle werden lediglich die für diese Arbeit relevanten Parameter näher erläutert. Weiterführende Informationen zu den einzelnen Angaben findet man in [18]. Quelltext 6 zeigt den Abschnitt für die TDM-Konfiguration. Die Zeilen 5 bis 8 definieren die IPL-Parameter „*Modulation“* und „*RollOffFactor“*. Der Wert für die Modulation steht für die Anzahl der Bits pro Symbol, das heißt im gezeigten Beispiel handelt es sich um eine QPSK-Modulation. Ab Zeile 10 erfolgt die Angabe der OPL-Parameter. Dazu gehört die *„Puncturing Pattern ID“* (PunctPatID), welche für eine Coderate aus der Tabelle 4 steht. Alle weiteren Daten werden für die Einstellung des Interleavers benutzt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <ParameterConfig>
3      <TDM>
4
5          <IPLConfig>
6              <Modulation>2</Modulation>
7              <RollOffFactor>0.15</RollOffFactor>
8          </IPLConfig>
9
10         <OPLConfig>
11             <PunctPatID>4</PunctPatID>
12             <CommonMultiplier>23</CommonMultiplier>
13             <NumberOfLateTaps>0</NumberOfLateTaps>
14             <NumberOfSlices>1</NumberOfSlices>
15             <SliceDistance>0</SliceDistance>
16             <NonLateIncrement>6</NonLateIncrement>
17             <TSID>1</TSID>
18         </OPLConfig>
19
20     </TDM>
```

Quelltext 6: TDM-Konfiguration aus der XML-Datei

Punct_Pat_ID	Code Rate	Pattern Name	Puncturing Pattern ($X;Y_0;Y_1;X';Y'_0;Y'_1;\ X;Y_0;\ldots$)
0	1/5	Standard	1;1;1;0;1;1
1	2/9	Standard	1;0;1;0;1;1; 1;1;1;0;1;1; 1;1;1;0;0;1; 1;1;1;0;1;1
2	1/4	Standard	1;1;1;0;0;1; 1;1;0;0;1;1
3	2/7	Standard	1;0;1;0;0;1; 1;0;1;0;1;1; 1;0;1;0;0;1; 1;1;1;0;0;1
4	1/3	Standard	1;1;0;0;1;0
5	1/3	Complementary	1;0;1;0;0;1
6	2/5	Standard	1;0;0;0;0;0; 1;0;1;0;0;1; 0;0;1;0;0;1; 1;0;1;0;0;1; 1;0;1;0;0;1; 0;0;1;0;0;1; 1;0;1;0;0;1; 1;0;1;0;0;1; 0;0;1;0;0;1; 1;0;1;0;0;1; 1;0;1;0;0;1; 0;0;1;0;0;1
7	2/5	Complementary	1;1;0;0;1;0; 0;1;0;0;1;0; 1;1;0;0;1;0; 1;1;0;0;1;0; 0;1;0;0;1;0; 1;0;0;0;0;0; 1;1;0;0;1;0; 0;1;0;0;1;0; 1;1;0;0;1;0; 1;1;0;0;1;0; 0;1;0;0;1;0; 1;1;0;0;1;0
8	1/2	Standard	1;1;0;0;0;0; 1;0;0;0;1;0
9	1/2	Complementary	1;0;0;0;1;0; 1;1;0;0;0;0
10	2/3	Standard	1;0;0;0;0;0; 1;0;0;0;0;0; 1;0;0;0;0;0; 1;0;1;0;0;1
11	2/3	Complementary	1;0;0;0;0;0; 1;0;1;0;0;1; 1;0;0;0;0;0; 1;0;0;0;0;0

Tabelle 4: Zusammenhang Punct_Pat_ID und Coderate (Quelle: [16])

Der Abschnitt der XML-Datei, in dem die OFDM-Konfiguration festgelegt wird, kann in Quelltext 7 eingesehen werden. Die IPL-Parameter befinden sich in den Zeilen 24 bis 30. Die Angaben für OPL werden in den Zeilen 33 bis 41 definiert.

```
21
22      <OFDM>
23
24          <IPLConfig>
25              <FFTLength>2048</FFTLength>
26              <GuardRatio>1/4</GuardRatio>
27              <SampleRateInMHz>40/7</SampleRateInMHz>
28              <Modulation>2</Modulation>
29              <Hierarchical>0</Hierarchical>
30              <Alpha_h>1</Alpha_h>
31          </IPLConfig>
32
33          <OPLHPConfig>
34              <PunctPatID>4</PunctPatID>
35              <CommonMultiplier>17</CommonMultiplier>
36              <NumberOfLateTaps>0</NumberOfLateTaps>
37              <NumberOfSlices>1</NumberOfSlices>
38              <SliceDistance>0</SliceDistance>
39              <NonLateIncrement>14</NonLateIncrement>
40              <TSID>1</TSID>
41          </OPLHPConfig>
42
43          <OPLLPConfig>
44              <PunctPatID>4</PunctPatID>
45              <TSID>2</TSID>
46          </OPLLPConfig>
47      </OFDM>
```

Quelltext 7: OFDM-Konfiguration aus der XML-Datei

Für die Messungen werden lediglich die Parameter *„Modulation“* (TDM: Zeile 6; OFDM: Zeile 28) und *„PunctPatID“* (TDM: Zeile 11; OFDM: Zeile 34) variiert. Der abschließende Teil der XML-Konfigurationsdatei (Quelltext 8) beinhaltet die Angaben über den verwendeten Transportstrom.

```
48
49      <TS>
50          <TSID>1</TSID>
51          <TSFileName>ts/NullPacketsWithCounter65k.ts</TSFileName>
52          <TSDataRate>3900210</TSDataRate>
53
54          <TSID>2</TSID>
55          <TSFileName>ts/NullPacketsWithCounter65k_360_TS_per_frame.ts</TSFileName>
56          <TSDataRate>3900210</TSDataRate>
57      </TS>
58
59
60  </ParameterConfig>
```

Quelltext 8: Angaben über den verwendeten TS

3.3 Der DVB-SH Testempfänger

Zum Testempfänger gehört eine grafische Benutzeroberfläche (Abbildung 22), wo die Konfiguration des Testempfängers vorgenommen wird.

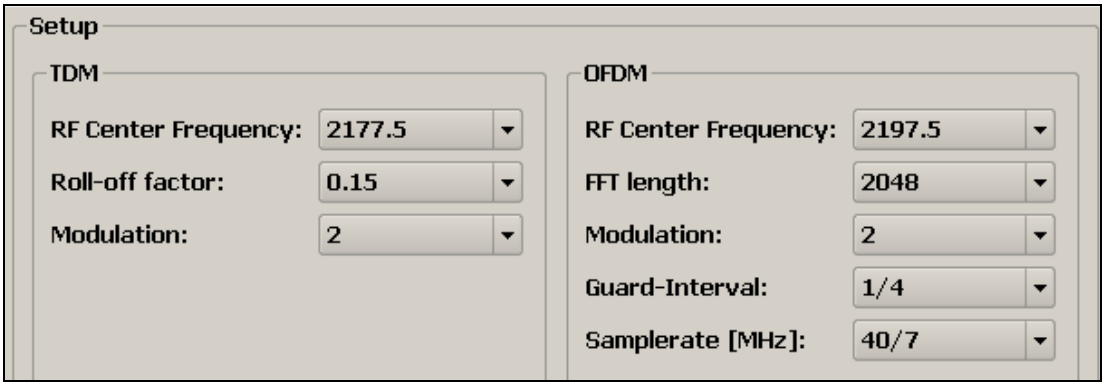


Abbildung 19: Benutzeroberfläche des Empfängers

3.3.1 Kommunikation mit dem Testempfänger

Der Kontakt mit dem Testempfänger wird ebenfalls über einen Java-Socket aufgebaut. Im Programm wird ein Buffer definiert, in dem die TS-Pakete kurzzeitig gespeichert werden. Das Programm identifiziert die einzelnen Stellen, an denen ein *„Header“* beginnt. Danach untersucht das Programm ob die Werte, die an den in Zeile 3 und 5 des Quelltextes 9 definierten Stellen stehen, mit den vordefinierten Werten aus Quelltext 10 übereinstimmen. Wenn diese Vorrausetzung nicht erfüllt ist, wird das aktuelle Paket als fehlerhaft angesehen. In Zeile 1 (Quelltext 9) wird der Zähler der Gesamtanzahl der Pakete berechnet. Die fehlerhaften Pakete werden in Zeile 4 und 7 gezählt.

```
1  packetCounter++;
2
3  if (buf[posInBuf + PKG_TS_HEADER_POS] != PKG_TS_HEADER) {
4      packetErrorCounter++;
5  } else if ((buf[posInBuf + PKG_TS_ERR_FLAG_POS] &
6              PKG_TS_ERR_FLAG_MASK) != 0) {
7      packetErrorCounter++;
8  }
```

Quelltext 9: Zähler der fehlerhaften Pakete

```
1  private static int BUF_LEN = 1024 * 1024 * 16;
2  private static int PKG_LEN = 244;
3  private static byte PKG_HEADER[] = { 0x53, 0x49, 0x47,
4          0x4e, 0x00, 0x00, 0x00, 0x20, 0x01, 0x00, 0x01,
5          0x00, 0x44, 0x41, 0x54, 0x41, 0x00,0x00, 0x07,
6          0x00, (byte) 0xee, (byte) 0xee, (byte) 0xee,
7          (byte) 0xee };
8  private static int PKG_TS_HEADER_POS = 0x38;
9  private static byte PKG_TS_HEADER = 0x47;
10 private static int PKG_TS_ERR_FLAG_POS = 0x39;
11 private static byte PKG_TS_ERR_FLAG_MASK = (byte) 0x80;
```

Quelltext 10: TS-Paket Variablen

3.4 Datenverarbeitung

Das Testprogramm liefert die folgenden Werte:

- der im DT4500 eingestellte Sendesignalpegel
- der vom ESPI gemessene Empfangssignalpegel
- vom ESPI gemessene C/N_0-Wert
- Anzahl der insgesamt gesendeten Pakete
- Anzahl der fehlerhaften Pakete
- Paketfehlerrate

Wenn ein Paket fehlerhaft ist, werden alle Bits innerhalb dieses Paketes als fehlerhaft betrachtet. In diesem Fall ist die Paketfehlerrate gleich mit der Bitfehlerrate (BER). Der gemessene C/N_0-Wert wird in die Verhältnisse E_b/N_0 und E_S/N_0 umgerechnet. Die BER wird als Funktion dieser beiden Verhältnisse dargestellt.

3.4.1 E_b/N_0 aus C/N_0 ausrechnen

In diesem Kapitel wird die Umrechnung des C/N_0-Verhältnisses zum E_s/N_0-Wert anhand zweier praktischer Messbeispiele erläutert. Als Grundlage dient die Gleichung (2.3.3) aus Kapitel 2.3. Für das beschriebene Projekt werden die notwendigen Berechnungen in Matlab durchgeführt.

OFDM 2k Mode

Im ersten Beispiel wird ein OFDM-Signal mit 2k FFT verwendet. In diesem Mode existieren 1705 Trägerfrequenzen. Davon sind 1529 Datenträger und 176

Pilotträger. Die Leistung der Pilotsignale beträgt 16/9 der Leistung der Datenträger. Für die Messung ist es wichtig die exakte Leistung der Datenträger zu bestimmen. Der Spektrumanalysator kann diese Leistung nicht direkt bestimmen. Stattdessen liefert er einen Wert, der zwischen den beiden Leistungswerten von Daten- und Pilotträgern liegt. Dieser Messwert muss mittels einer Berechnung korrigiert werden. Laut der unten stehenden Gleichung (3.4.1) liegt der vom Spektrumanalysator gelieferte C-Wert, in unserem Beispiel, ca. 8% über dem tatsächlichen Leistungswert der Datenträger.

$$\left[1529*1+176*\frac{16}{9}\right]*\frac{1}{1705*1}\approx 1.08 \qquad (3.4.1)$$

in dB:

$$10*\lg(1.08)=0.335 \qquad (3.4.2)$$

Mit Gleichung (3.4.3) wird der E_s/N_0-Wert mit dem korrigierten C/N_0-Wert für OFDM 2k errechnet.

$$\left(\frac{E_s}{N_0}\right)=\frac{C}{N_0}-0.335-10\lg c \qquad (3.4.3)$$

TDM

Beim Messbeispiel mit einem TDM-Signal muss keine Korrektur des gemessenen C/N_0-Wertes durchgeführt werden, da nur ein Träger existiert. Es gilt die Gleichung (2.3.2).

3.4.2 Wirkung der Funktionsparameter

Während des Tests werden QPSK, 8PSK, 16APSK und 16QAM Signale verwendet. Abbildung 20 zeigt die Konstellationsdiagramme für diese Modulationsarten. Aus der Darstellung ist ersichtlich, dass mit steigendem Grad der Modulationsart sich der Signalraum pro Symbol verringert. Die spektrale Effizienz ist bei Modulationsarten höherer Ordnung besser, jedoch steigt auch die Anfälligkeit gegenüber Übertragungsfehlern. Zusammenfassend kann man sagen, dass mit der QPSK-Modulation die angestrebte Signalqualität bei einem niedrigeren E_S/N_0 erreicht wird als mit der 16APSK Modulation, allerdings bei geringerer Bandbreiteneffizienz. Als Schätzwert könnte man durch die Verdopplung der Bitanzahl pro Symbol von einem Unterschied von 3 dB

ausgehen. Bei einer Coderate von 1/3 wird ein steilerer Verlauf des BER(C/N_0)-Graphen erwartet, als bei einer Coderate von 2/3. Der Grund dafür ist, dass die Coderate 1/3 eine bessere Fehlerkorrektur ermöglicht und dadurch das Sendesignal, trotz stärkerer Störeinflüsse, mit einer niedrigeren Fehlerrate dekodiert werden kann. Für den Übergang vom AWGN- zum Rice-Kanalmodell muss, bei gleich bleibender Modulationsart und Coderate, der Störabstand notwendigerweise erhöht werden, um die gleiche Qualität zu erreichen.

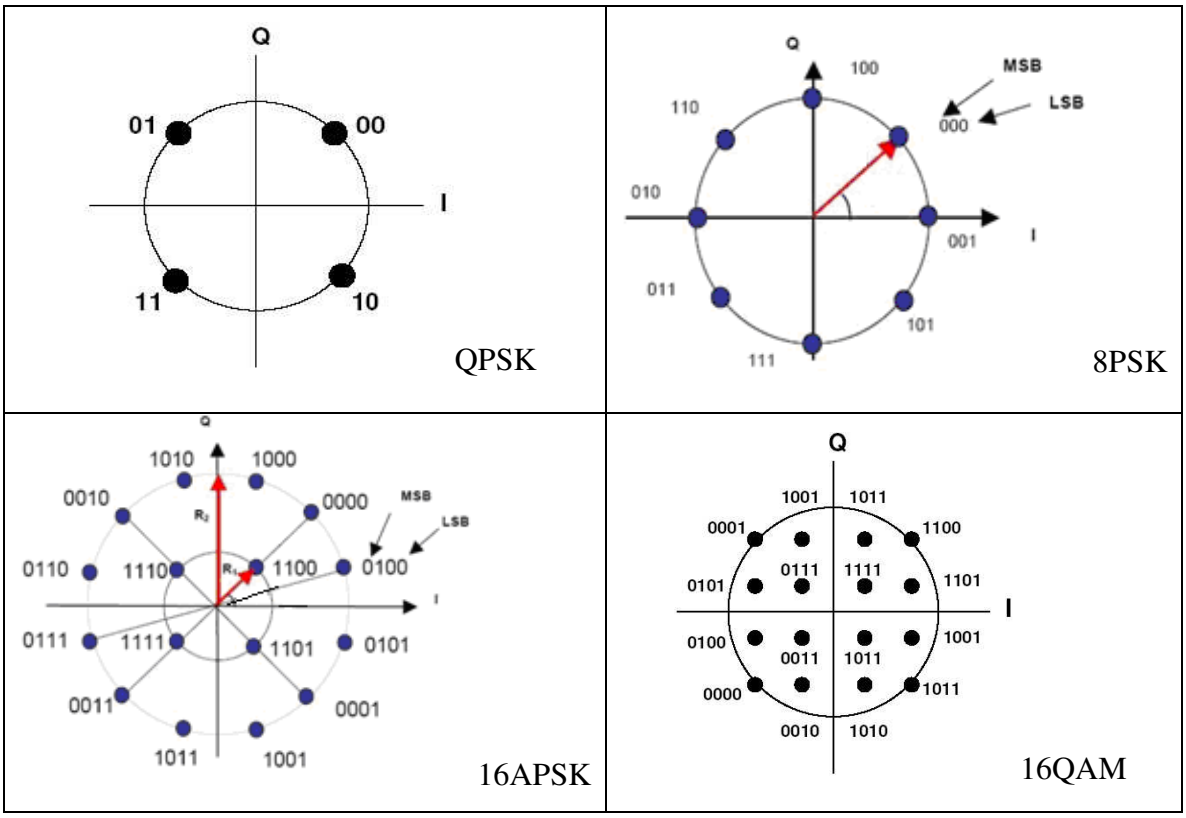


Abbildung 20: Modulationsarten in DVB-SH (Quelle:[s,t,u,v])

3.4.3 Graphische Darstellung

Die Darstellung der Messwerte und der ideale BER(E_s/N_0)-Graph, wie zum Beispiel in Graph 1 (Kapitel 2.3) dargestellt, wurden mit Matlab gezeichnet. Um einen idealen AWGN-Kanal zu simulieren gibt es in Matlab den Befehl *„berawgn“* nach Syntax 3. Das erste Eingabeargument *„EbNo“* ist ein Vektor. Beim ausgegebenen BER handelt es sich ebenfalls um einen Vektor der gleichen Größe, dessen Werte abhängig sind von den entsprechenden E_b/N_0-Werten. Der zweite Eingabeparameter definiert den Modulationstyp. Die Ordnung der Modulation wird mit dem letzten Wert *„M“* festgelegt.

```
ber = berawgn(EbNo,'modulationstyp',M)
```

Syntax 3: BER Werte für AWGN-Kanal in Matlab

Quelltext 11 zeigt den Matlab Programmcode, mit dem ein Beispielgraph erzeugt wird. In den Zeilen 2 bis 7 werden die Parameter für den oberen Befehl (Syntax 1) definiert. Die Zeilen 10 bis 12 symbolisieren die Signale mit unterschiedlichen Modulationsarten in einem idealen AWGN-Kanal.

```
1  %Idealer Graphik
2  M_QPSK = 4;
3  M_8PSK = 8;
4  M_16APSK = 16;
5  Eb_No_QPSK = 0:1:11;
6  Eb_No_8PSK = 0:1:12;
7  Eb_No_16APSK = 0:1:12;
8
9  %AWGN Kanal
10 ber_QPSK = berawgn(Eb_No_QPSK,'psk',M_QPSK,'nondiff');
11 ber_8PSK = berawgn(Eb_No_8PSK,'psk',M_8PSK,'nondiff');
12 ber_16APSK = berawgn(Eb_No_16APSK,'psk',M_16APSK,'nondiff');
```

Quelltext 11: BER Werte für idealen AWGN-Kanal mit Matlab ausrechnen

Mit dem Programmteil aus Quelltext 12 können die Daten aus einer externen Datei eingelesen werden. Solche Dateien werden vom Testprogramm während des Messdurchlaufs erstellt. Die Zeilen 1 bis 6 definieren die Matlab Variablen, welche in den Berechnungen ab Zeile 8 bis 12 verwendet werden.

```
1  TDM_QPSK_R_1_3 = csvread('TDM_QPSK_R_1_3.txt');
2  TDM_QPSK_R_1_3_level_dt4500 = TDM_QPSK_R_1_3(:,1);
3  TDM_QPSK_R_1_3_level_espi = TDM_QPSK_R_1_3(:,2);
4  TDM_QPSK_R_1_3_snr_lower= TDM_QPSK_R_1_3(:,3);
5  TDM_QPSK_R_1_3_snr_upper = TDM_QPSK_R_1_3(:,4);
6  TDM_QPSK_R_1_3_error_rate = TDM_QPSK_R_1_3(:,8);
7
8  TDM_QPSK_R_1_3_mitte = (TDM_QPSK_R_1_3_snr_lower
9                          + TDM_QPSK_R_1_3_snr_upper)/2;
10 TDM_QPSK_Es_N = (TDM_QPSK_R_1_3_C_N - 10*log10(1/3));
11 TDM_QPSK_Eb_N = (TDM_QPSK_R_1_3_C_N - 10*log10(1/3)
12                 - 10*log10(2));
```

Quelltext 12: Matlab-Daten aus externer Datei auslesen

Quelltext 13 zeigt wie, die aus Sicht der Messung, zusammengehörenden Werte in einem Graph dargestellt werden. In diesem Beispiel wird ein TDM QPSK Signal mit den Coderaten 1/3 (Zeile 2-3) und 2/3 (Zeile 8-9) gezeichnet. Als Referenz wird zusätzlich der Graph eines idealen AWGN-Kanals hinzugefügt (Zeile 10).

Nach diesem Schema werden alle Messungen grafisch ausgewertet. Das bedeutet es wird für ein Kanalmodell mit einem Modulationsverfahren je ein Graph für zwei Coderaten gezeichnet und der Referenzgraph des Kanalmodells.

```
% Plot TDM QPSK 1/3 und 2/3 AWGN
figure;semilogy(TDM_QPSK_R_1_3_C_N,TDM_QPSK_R_1_3_error_rate,...
                                                '-.+','LineWidth',2);
title('TDM QPSK Signal im AWGN Kanal');
xlabel('C/No (dB)');
ylabel('BER [%]');
hold on;
plot(TDM_QPSK_R_2_3_C_N,TDM_QPSK_R_2_3_error_rate,...
                                    '--ro','LineWidth',2);
plot(Eb_No_QPSK , ber_QPSK,'-y','LineWidth',2);
legend('TDM QPSK R:1/3', 'TDM QPSK R:2/3',...
                        'idealer QPSK Graph');
axis([-1 7 0 100]);
grid on;
hold off;
```

Quelltext 13: Matlab-Graph konstruieren

Neben der TDM Messung werden auch OFDM Signale gemessen. Für diese Signale wird nicht nur ein AWGN-, sondern auch ein Rice-Kanalmodell verwendet. Der Matlab-Befehl *„berfading“,* der für die Darstellung eines idealen Graphen des Rice-Kanals verwendet wird, befindet sich in Syntax 4. Die ersten Parameter sind identisch mit dem Befehl *„berawgn“*. Der Wert *„divorder“* ist die Anzahl der Ausbreitungswege bei einer Mehrwegeausbreitung. Abschließend wird der Rice-Faktor K definiert.

```
ber = berfading(EbNo,modulationstyp,M,divorder,K)
```

Sysntax 4: BER Werte für Rice-Kanal in Matlab

Im folgenden Programmcode (Quelltext 14) werden die BER-Werte eines idealen AWGN- und Rice-Kanals, mit einer QPSK- und 16QAM-Modulation, definiert.

```
%Idealer Graphik
M_QPSK = 4;
M_16QAM = 16;
Eb_No_QPSK = 0:1:8;
Eb_No_16QAM = 0:1:8;
K= 10^0.7;

%AWGN Kanal
ber_QPSK_AWGN = berawgn(Eb_No_QPSK,'psk',M_QPSK,'nondiff');
ber_16QAM_AWGN = berawgn(Eb_No_16QAM,'qam',M_8PSK,'nondiff');

%RICE Kanal
ber_QPSK_RICE = berfading(Eb_No_QPSK,'psk',M_QPSK,1,K);
ber_16QAM_RICE = berfading(Eb_No_16QAM,'qam',M_16QAM,1,K);
```

Quelltext 14: BER eines idealen AWGN- und Rice-Kanals, mit QPSK und 16QAM

3.5 Das Testprogramm

Wie in der Projektbeschreibung erwähnt, übernimmt das Testprogramm eine zentrale Rolle. Beim ESPI konfiguriert es die Messparameter wie zum Beispiel *„Center Frequenz“*, *„Span“* und *„RBW“* und aktiviert die C/N_0-Messung. Für den Signalgenerator DT4500 wählt die Software die Signalquelle und das Kanalmodell aus. Abschließend speichert die Testsoftware die Messinformationen und wertet diese aus. Diese Aufgaben werden von unterschiedlichen Modulen übernommen, welche ständig miteinander interagieren müssen. Aus diesem Grund eignet sich für die praktische Realisierung eine objektorientierte Programmiersprache wie Java am besten. Die Programmiersprache Java ist in [20] beschrieben. Ergänzend dazu wurde für die Auswertung und Darstellung der Messergebnisse das Programm Matlab eingesetzt. Abbildung 21 zeigt eine Übersicht der beteiligten Module.

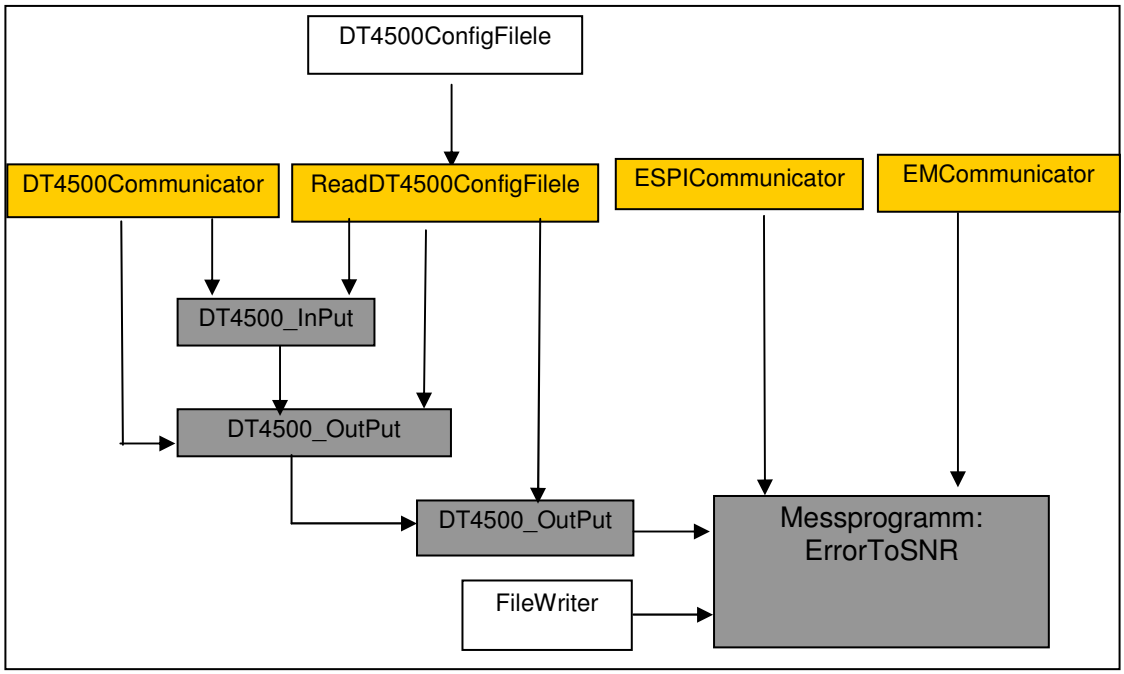


Abbildung 21: Programmaufbau

Das Testprogramm wurde so strukturiert, dass jedes Messgerät durch ein Java-Paket symbolisiert wird. Innerhalb des Paketes werden die einzelnen Hauptmenüfunktionen als Java-Klassen dargestellt. In den Klassen gibt es Java-Methoden, welche den einstellbaren Parametern entsprechen. Das Programm befindet sich in Anhang D.

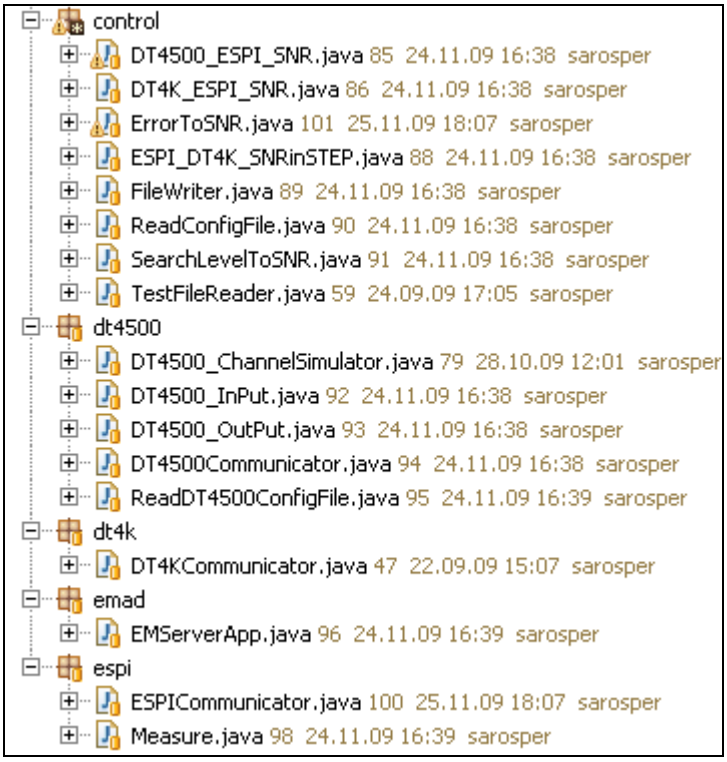


Abbildung 22: Übersicht Pakete und Klassen

In Abbildung 22 kann der oben beschriebene Aufbau nachvollzogen werden. Es gibt die Pakete *„dt4500“*, *„espi“* und *„emad“* als virtuelle Geräte. Die Klassen im Paket *„control“* greifen auf die virtuellen Maschinen zu und steuern die Messabläufe. In den folgenden Kapiteln werden die einzelnen Pakete und Klassen genauer betrachtet.

3.5.1 Das Messprogramm

In diesem Teil des Testprogramms werden die verwendeten Geräte konfiguriert und gesteuert. Quelltext 15 zeigt den Konstruktor der Klasse *„ErrorToSNR“.* Darin werden die Objekte der Geräte angelegt. Dazu zählen in Zeile 3 der dt4500, in Zeile 6 ESPI, Zeile 8 serverApp und f für die Klasse *„FileWriter“*. Abschließend wird die Methode *„runMeasure“* aufgerufen. Diese führt die anstehenden Messungen aus.

```
1  public ErrorToSNR(String file_name) {
2          try {
3              dt4500 = new DT4500_OutPut(DT4500_hostname,
4                                         DT4500_port);
5              f = new FileWriter(file_name);
6              espi = new ESPICommunicator(ESPI_hostname,
7                                          ESPI_port);
8              serverApp = new EMServerApp(EM_hostname,
9                                          dataPort);
10             runEMServer = new Thread(serverApp);
11             runEMServer.start();
12             runMeasure(time_a, time_b);
13         } catch (Exception e1) {
14             e1.printStackTrace();
15         }
16     }
```

Quelltext 15: Konstruktor der Klasse ErrorToSNR

Der Quelltextabschnitt 16 zeigt auf welche Weise die Daten aus der Klasse „*EMServerApp“* geholt werden und befindet sich in der Methode „*runEmServer“*. Die erste while-Schleife läuft 20 Sekunden lang und hat die Aufgabe die Messung zu stabilisieren. Der Grund für diese Vormessung wird in Kapitel 3.6.1 beschrieben. Danach werden, mit der Funktion „*resetCounter“*, die Zähler für Gesamtanzahl und fehlerhafte Pakete zurückgesetzt. Die zweite while-Schleife berechnet für eine beliebige Zeitdauer die Paketfehlerrate (Zeile 16).

```
1  while (System.currentTimeMillis() / 1000 - s_time < 20) {
2          Thread.sleep(1000);
3          pkg = serverApp.packetCounter();
4          err = serverApp.packetErrorCounter();
5          }
6          serverApp.resetCounter();
7          s_time = System.currentTimeMillis() / 1000;
8          System.out.println("Bitte warten Sie "
9                                  + time + " sec. lang");
10 while (System.currentTimeMillis() / 1000 - s_time < time) {
11         Thread.sleep(1000);
12         pkg = serverApp.packetCounter();
13         if (p != pkg) {
14             p = pkg;
15             err = serverApp.packetErrorCounter();
16             error_rate = (double) err / (double) pkg * 100;
17             valid = true;
```

Quelltext 16: Fehlerrate ausrechnen

Abbildung 23 beschreibt die Methode „*measureErrorRate*“. In den grafischen Symbolen steht zum einen die durchgeführte Tätigkeit und zum anderen die dazugehörige Java-Funktion. Nach den Gerätekonfigurationen kommt eine for-Schleife. Für diese Schleife legt der Benutzer ein „*startLevel*“ und ein „*stopLevel*“ fest. Das Programm beginnt die Messung mit dem „*startLevel*“. Innerhalb der for-Schleife werden zunächst die Signalpegel der beiden Messgeräte eingestellt. Danach erfolgt eine BER-Messung. Ist die BER (Java-Variable: *final_error_rate*) größer als 99,8 %, wird die Variable „*startLevel*“ um 1 dB erhöht und die for-Schleife erneut durchlaufen. Für den Fall das die BER kleiner ist als 99,8 % wird „*startLevel*“ um 1 dB verringert und die Schrittweite auf 0,1 dB eingestellt, zudem folgt eine genauere Messung in einer while-Schleife. Diese Schleife wird, mit einer Verringerung des Signalpegels um die neue Schrittweite von 0,1 dB, solange durchlaufen, bis die BER 0% erreicht. Innerhalb dieses Prozesses werden zuerst die Signalpegel eingestellt. Danach erfolgt eine C/N_0-Messung mit dem ESPI und eine BER-Messung. Abschließend werden die Werte in einer Datei gespeichert. Wenn der BER-Wert in der äußeren for-Schleife, aufgrund eines Fehlers, nicht die 99,8 % Grenze unterschreitet, wird das Programm bei der unteren Grenze (*stopLevel*) automatisch gestoppt. Dieser Prozess schützt die Geräte vor einem zu großen Signalpegel.

Die oben genannte Einstellung der Signalpegel ist eine getrennte Methode, weil ESPI und DT4500 unterschiedliche Pegel haben. Diese aber dennoch voneinander abhängig sind. Der ESPI Referenzpegel wird immer mit einer

AMPLITUDE_DIFFERENCE von 16 dB über dem DT4500 Signalpegel eingestellt. Damit wird eine bessere Genauigkeit der C/N_0-Messung erreicht, ohne dass der ESPI in *„overload“* gesteuert werden muss.

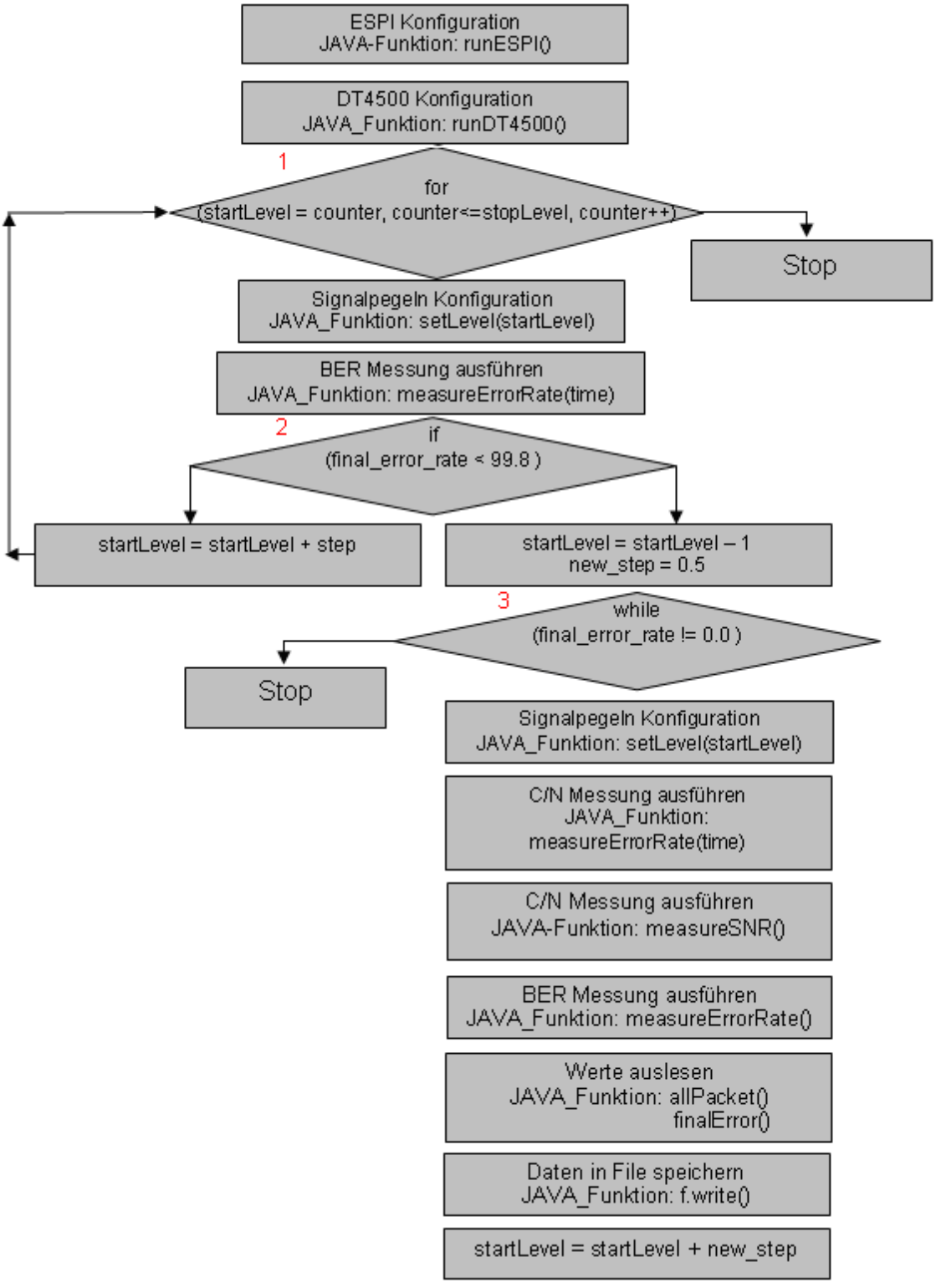


Abbildung 23: Funktion MaesureErrorRate

3.6 Durchgeführte Messungen mit dem Testprogramm

Es ist ein Framework aufgebaut, das die automatisierte Durchführung von Messreihen an einem DVB-SH Empfänger ermöglicht. Mit diesem Testprogramm werden die folgenden Aufgaben durchgeführt.

1. Der ESPI misst während der C/N_0-Messung einen C/N_0-Wert für "Upper"- und einen Wert für *"Lower Adjacent Channel Power"*. Welche Wirkung hat dieses Verhalten auf das Ergebnis?
2. Die zweite Messaufgabe beschäftigt sich mit dem Verhalten des Testempfängers, wenn das C/N_0-Verhältnis sich während des Messvorganges verringert oder erhöht.
3. Berechnung von E_b/N_0 und E_s/N_0 für TDM Signale mit Coderate 1/3
4. Der Testdurchläufe mit allen Konfigurationen aus Tabelle 2 durchführen und die Ergebnisse beurteilen.

3.6.1 Messverfahren

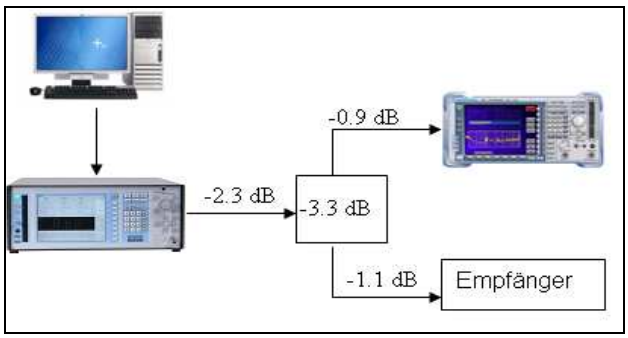


Abbildung 24: Messaufbau mit zusätzlichen Dämpfungen

Der DT4500 Signalgenerator erzeugt aus den TS-Paketen das gewünschte DVB-SH Signal und fügt die ausgewählte Rauschleistung hinzu. Dieses Signal wird zum Testempfänger weitergeleitet und von der ESPI überwacht. Die Java-Klasse *„EMServer"* zählt wie viele TS-Pakete, bei sich verändernden C/N_0-Werten, falsch detektiert werden. Die C/N_0-Werte werden mit Hilfe von Matlab in das E_s/N_0-Verhältnis umgerechnet. Die BER wird mit dem dazugehörenden E_s/N_0-Wert oder C/N_0 in einem Koordinatensystem dargestellt. Die verwendeten Parameter werden in den Tabellen 4 bis 10 aufgelistet. Die Tabellen 5-7 beinhalten die Geräte- und

Signalkonfigurationsparameter für die TDM-Messung, die Tabelle 8-11 für die OFDM-Messung.

ESPI			
Allgemeine Einstellungen:		Spezifische Einstellungen:	
Betriebsmode:	Spec.	Measure Mode:	[CP / ACP] ON
Center Freq. [MHz]:	2177.5	CP / ACP STANDARD	none
Span [MHz]:	20	NO. OF ADJ CHAN:	1
RBW [kHz]:	300	NO. OF TX CHAN:	0
VBW [kHz]:	1	CHANNEL BW [MHz]:	3.61
Sweep Time [ms]:	500	CHANNEL SPACING [MHz]:	5.5
Referent Level [dBm]:	?	CP / ACP ABS/REL:	REL

Tabelle 5: ESPI Konfigurationsparameter für TDM-Messungen

DT4500					
Input: CH1(TDM)		Channel Simulator		Output: DAC1 CH1	
Source	file	-	-	Signalpegel:	?
ChannelCoding	OPL_IPL_ TDM	-	-	Frequenz [MHz]:	2177.5
OPL/IPL XML file	*	-	-	Rauschpegel [dBm]:	-50
		-	-	$BW_{Rauschen}$ [kHz]	3610

Tabelle 6: DT4500 Konfigurationsparameter für AWGN-TDM-Messungen

DVB-SH Signalparameter			
TDM		OFDM	
Modulation	QPSK, 8PSK, 16APSK	Modulation	QPSK
Code Rate	1/3, 2/3	Code Rate	1/3
		FFT	2k
Rollof Faktor	0.15	G.I.	0.25
Interleaving Time	5 [s]	Interleaving Time	5 [s]

Tabelle 7: Signal Konfigurationsparameter für TDM-Messungen

*dvbsh_TDM_QPSK_R1_3.xml. Dieses XML File ist in dem Kapitel 3.2.2. vorgestellt.

ESPI			
Allgemeine Einstellungen:		Spezifische Einstellungen:	
Betriebsmode:	Spec.	Measure Mode:	[CP / ACP] ON
Center Freq. [MHz]:	2187.5	CP / ACP STANDARD	none
Span [MHz]:	20	NO. OF ADJ CHAN:	1
RBW [kHz]:	300	NO. OF TX CHAN:	0
VBW [kHz]:	1	CHANNEL BW [MHz]:	4.25
Sweep Time [ms]:	500	CHANNEL SPACING [MHz]:	5.
Referent Level [dBm]:	?	CP / ACP ABS/REL:	REL

Tabelle 8: ESPI Konfigurationsparameter für OFDM-Messungen

DT4500					
Input: CH1(TDM)		Channel Simulator		Output: DAC1 CH1	
Source	file	-	-	Signalpegel:	?
ChannelCoding	OPL_IPL_ TDM	-	-	Frequenz [MHz]:	2187.5
OPL/IPL XML file	*	-	-	Rauschpegel [dBm]:	-50
		-	-	$BW_{Rauschen}$ [kHz]	4250

Tabelle 9: DT4500 Konfigurationsparameter für AWGN-OFDM-Messungen

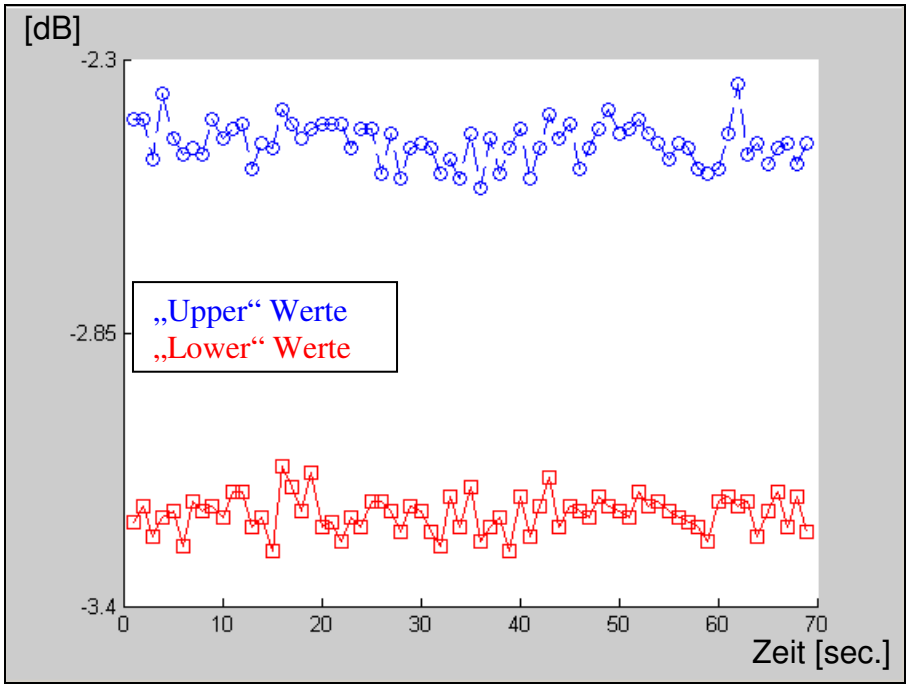
DT4500					
Input: CH1(TDM)		Channel Simulator		Output: DAC1 CH1	
Source	file	Group0	enable	Signalpegel:	?
ChannelCoding	OPL_IPL _TDM	Speed [km/h]	50	Frequenz [MHz]:	2187.5
OPL/IPL XML file	*	RICE K [dB]	7.0	Rauschpegel [dBm]:	-50
		Dop. Shift [Hz]	5-	$BW_{Rauschen}$ [kHz]	4250

Tabelle 10: DT4500 Konfigurationsparameter für RICE-OFDM-Messungen

DVB-SH Signalparameter			
TDM		OFDM	
Modulation	QPSK	Modulation	QPSK, 16QAM
Code Rate	1/3	Code Rate	1/3, 2/3
		FFT	2k
Rollof Faktor	0.15	G.I.	0.25
Interleaving Time	5 [s]	Interleaving Time	5 [s]

Tabelle 11: Signal Konfigurationsparameter für OFDM-Messungen

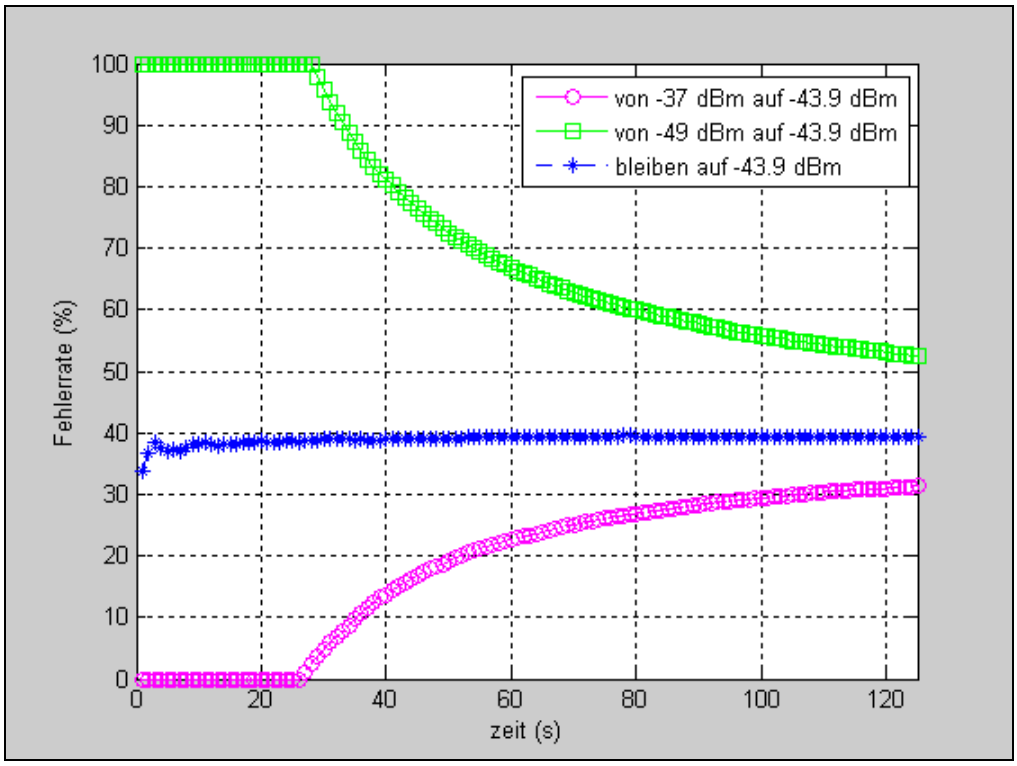
Aufgrund der Störungen wird sich die Leistung des Sendesignals ständig ändern. Diese Änderungen werden die beim ESPI gemessenen Empfangsleistungspegel und die daraus resultierenden C/N_0-Werte beeinflussen. Des Weiteren werden die $BER(E_b/N_0)$-Graphen keinen gleichmäßigen Verlauf haben. Um diese Erscheinung zu veranschaulichen, wurde eine C/N_0-Messung, mit einem bestimmten Sendesignalpegel 70 Sekunden lang durchgeführt. In jeder Sekunde wurden die *„Upper“*- und *„Lower“*-Werte gespeichert. Die zeitlichen Schwankungen der Leistung in der *„Adjacent Channel Bandwidth“* werden in den Graph 2 dargestellt. Idealerweise sollten diese Graphen eine gerade Linie bilden. Da dies auf die realen Messungen nicht zutrifft, ist es nicht egal, zu welchem Zeitpunkt die C/N_0-Messung durchgeführt wird.



Graph 2: Leistung in *„Adjacent Channel Bandwidth“.*

Zur Beseitigung dieser Fehlerquelle wurde ein Programmabschnitt im Testprogramm implementiert, der aus 20 C/N_0-Messergebnissen einen Mittelwert errechnet.

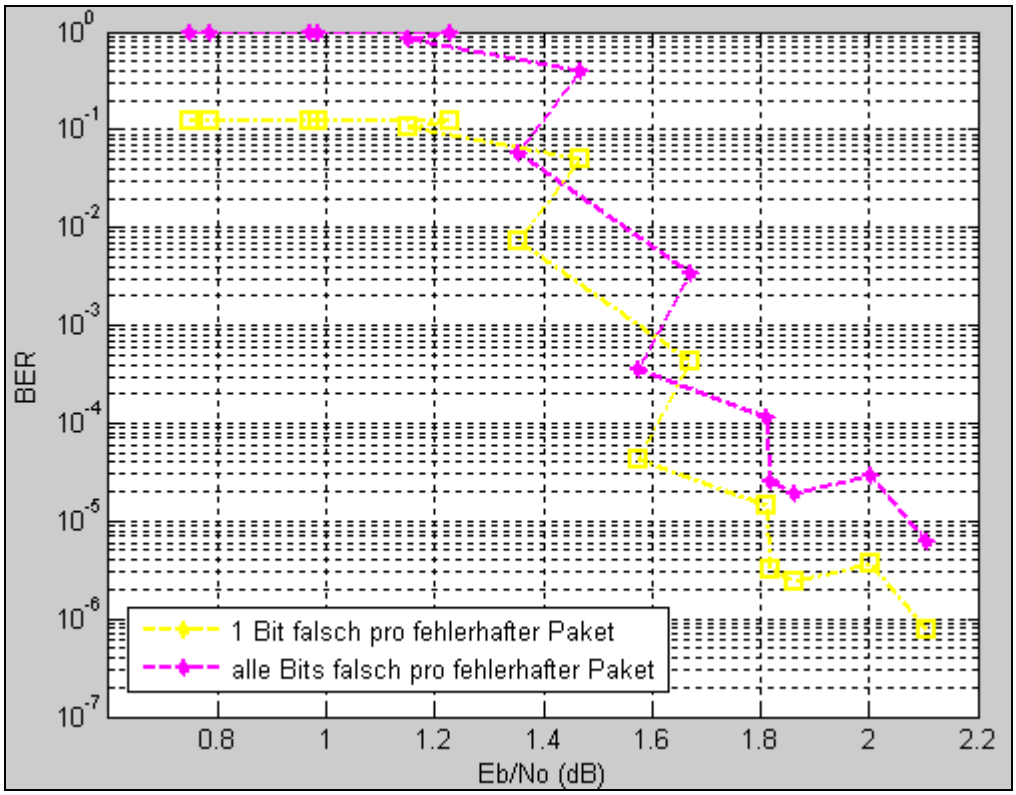
Während eines Testdurchlaufs beeinflussen die vorangegangenen Messwerte die aktuellen Messwerte, da ein Interleaver verwendet wird. Dieser Einfluss wird mit Hilfe einer Messreihe untersucht. Dabei wird ein TDM QPSK Signal mit Coderate 1/3 im AWGN-Kanal verwendet. Damit das Ergebnis besser erkennbar wird, wird und eine Interleaver-Länge von 10 Sekunden benutzt. Für diese Untersuchung wurden zunächst die drei Signalpegel -37 dBm (BER=0%), -49 dBm (BER=100%) und -43,9 dBm, zu dem ein mittlerer BER-Wert gehört, festgelegt. Danach wurde eine erste BER-Messung mit -37 dBm gestartet und nach 20 Sekunden auf den Signalpegel -43,9 dBm umgeschaltet. Das Ergebnis zeigt die magentafarbige Linie in Graph 3. Bei der zweiten BER-Messung war der Startpegel mit -49 dBm definiert und wurde nach 20 Sekunden auf den Wert -43,9 dBm verringert (grüne Linie in Graph 3). Im dritten Messdurchlauf lief die BER-Messung 20 Sekunden lang mit dem Signalpegel -43,9 dBm. Dieser wurde nach Ablauf der Zeit erneut eingestellt. (blaue Linie in Graph 3) Aus dem Ergebnis konnte abgelesen werden, dass sich beim gleich bleibenden Signalpegel die Werte stabilisieren. Der Grund dafür ist die Tatsache, dass im Interleaver noch ältere Messwerte gespeichert sind, die den Beginn der neuen Messung verfälschen.



Graph 3: Wirkung des vorherigen Wertes

Als Schlussfolgerung dieser Untersuchung wurde zur Verbesserung der eigentlichen Messdurchläufe eine 20 Sekunden lange Vormessung eingeführt. Nach dieser Zeit werden die Fehlerzähler zurückgesetzt und die eigentliche Messung beginnt.

Mit den Einstellungen aus den Tabellen 5-7 und mit einem TDM Signal (Coderate 1/3, QPSK, AWGN Kanal) wurde untersucht, welche Folgen es hat, wenn in einem fehlerhaften Paket, das noch nicht demoduliert wurde, entweder alle Bits des Paketes oder nur ein Bit als Fehler gezählt werden. Wie in Graph 4 dargestellt definieren diese beiden Fälle die theoretischen Qualitätsgrenzen der Übertragung. Wenn nur ein oder wenige Bits in einem Empfangspaket verfälscht wurden kann das Signal, aufgrund von Fehlerschutzkodierung, trotzdem fehlerfrei dekodiert werden. Für den Fall, dass die Fehlerrate 100% beträgt, kann das Signal einfach invertiert werden, um die richtigen Informationen zu erhalten. Somit kann festgestellt werden, dass der unglücklichste Fall bei einer Fehlerrate im Empfangspaket von genau 50% auftritt. Dann kann nicht mehr entschieden werden, welche Hälfte des Empfangssignals fehlerfrei ist.



Graph 4: Bitfehlerrate

Für die folgenden Messreihen wird angenommen, dass in einem fehlerhaften Paket alle Bits falsch sind.

3.7 Ergebnisse des Tests

1. Aufgabe

Der Pegel des TDM-Signals wird beim DT4500 schrittweise verringert, bis das Signal im Rauschpegel verschwindet oder die BER 0% erreicht. Die gezählten Werte der Paketfehlerrate werden als Funktion der folgenden C/N_0 Werte dargestellt:

- obere („Upper“)
- untere („Lower“)
- mittelwert

Parameters des Messprogramms:

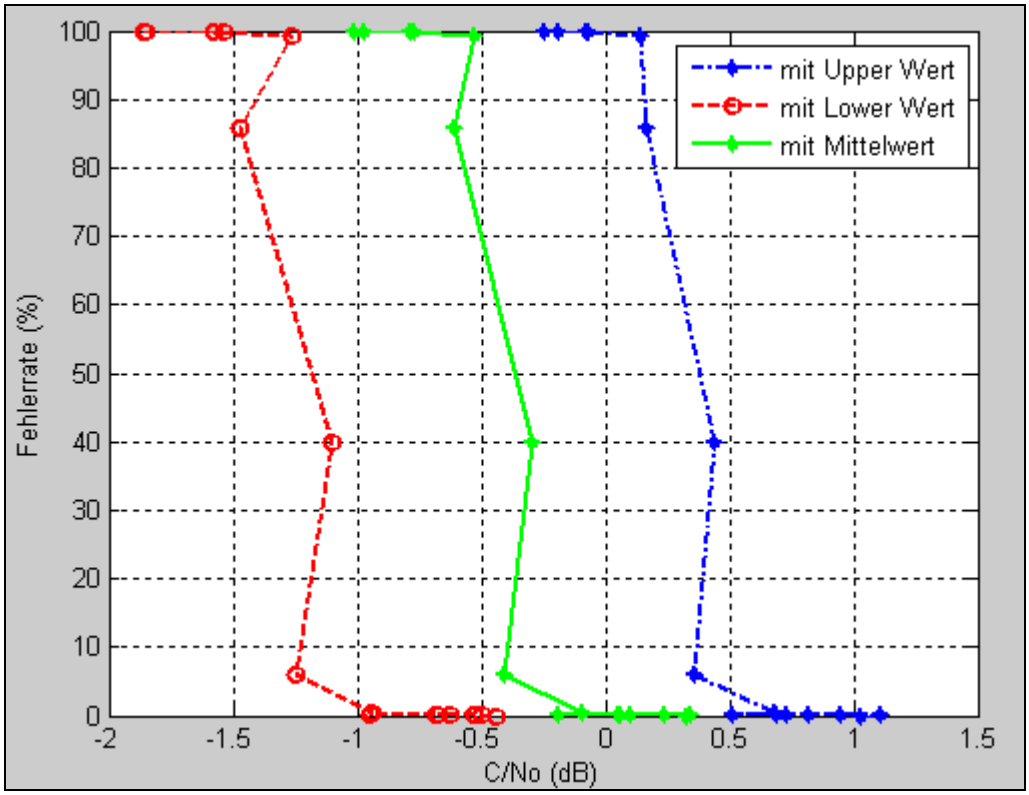
Start Level: -48 dBm

Step_1: -1

Laufzeit_1: 150 sec.

Step_2: -0.1

Laufzeit_2: 300 sec.



Graph 5: Lower, Upper und Mittelwert für C/N_0

Aus den Messergebnissen in Graph 5 ist erkennbar, dass die Verwendung des *„Upper“* (blau)- oder des *„Lower“* (rot)-Wertes, einen Unterschied von mehr als 1,5 dB verursacht. Diese Erscheinung kommt aufgrund der Ungenauigkeit des DT4500 Rauschgenerators zustande. Graph 5 veranschaulicht dass die Mittelwertbildung (grüne Linie) aus den *„Upper“*- und *„Lower“*-Werten eine praktische Lösung für dieses Problems ist, weil so am genauesten der Rauschgrund des Signals am Empfänger beschrieben wird. Des Weiteren ist es sichtbar, dass der Verlauf der Funktionen Fehlerrate(C/N_0) nicht monoton ist. Der Grund dafür wird in Kapitel 4 erklärt.

2. Aufgabe

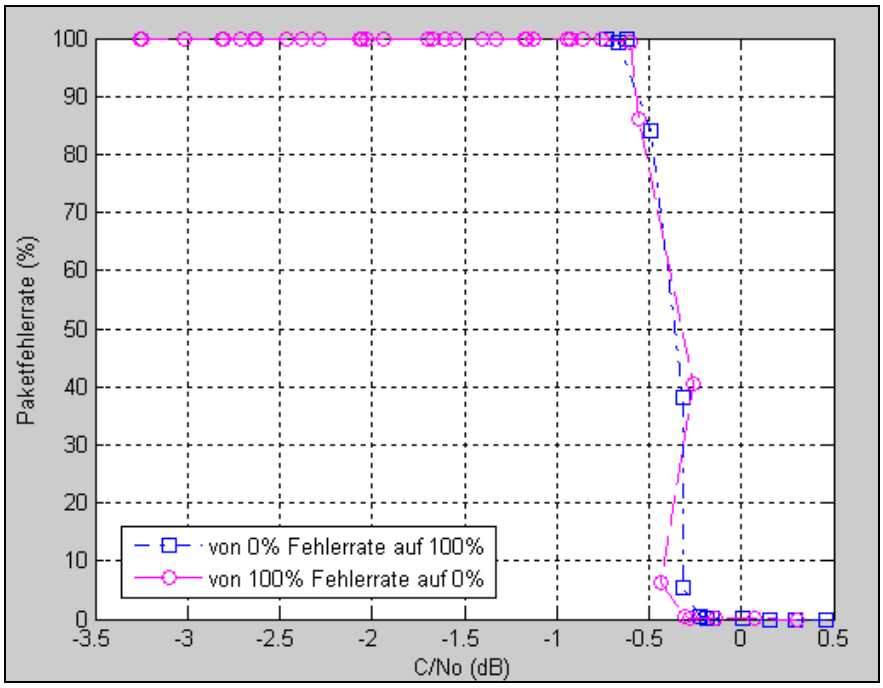
Zur Lösung von Aufgabe 2 werden zwei verschiedene Herangehensweisen benutzt. Bei der Ersten (Test A Tabelle 12) beginnt die Messung bei einem Signalpegel, bei dem der BER 0,0% beträgt. Während des Messvorganges wird der Signalpegel eines TDM-Signals (Coderate 1/3, QPSK, AWGN Kanal) vom DT4500 in 1dB-Schritten verringert. Bei jedem Schritt wird die BER bestimmt. Dieser Vorgang wird solange durchgeführt, bis das Signal im Rauschpegel

verschwindet oder der BER 100% erreicht. Der zweite Messansatz (Test B Tabelle 12) arbeitet nach dem gleichen Prinzip wie der Erste. Der Unterschied besteht darin, dass beim Startpegel der BER 100 % beträgt. Folglich läuft dieser Messvorgang bis zu dem Punkt, an dem der BER 0,0% ist.

Test A. (von 100% Fehlerrate):	Test B. (von 0% Fehlerrate):
Start Pegel: -37 dBm	Start Pegel: -48 dBm
Stop Pegel: -48 dBm	Stop Pegel: -37dBm
Schritt_1: 1	Schritt_1: -1
Laufzeit_1: 15 sec.	Laufzeit_1: 15 sec.
Schritt_2: 0.1	Schritt_2: -0.1
Laufzeit_2: 30 sec.	Laufzeit_2: 30 sec.

Tabelle 12: Parameters des Messprogramms:

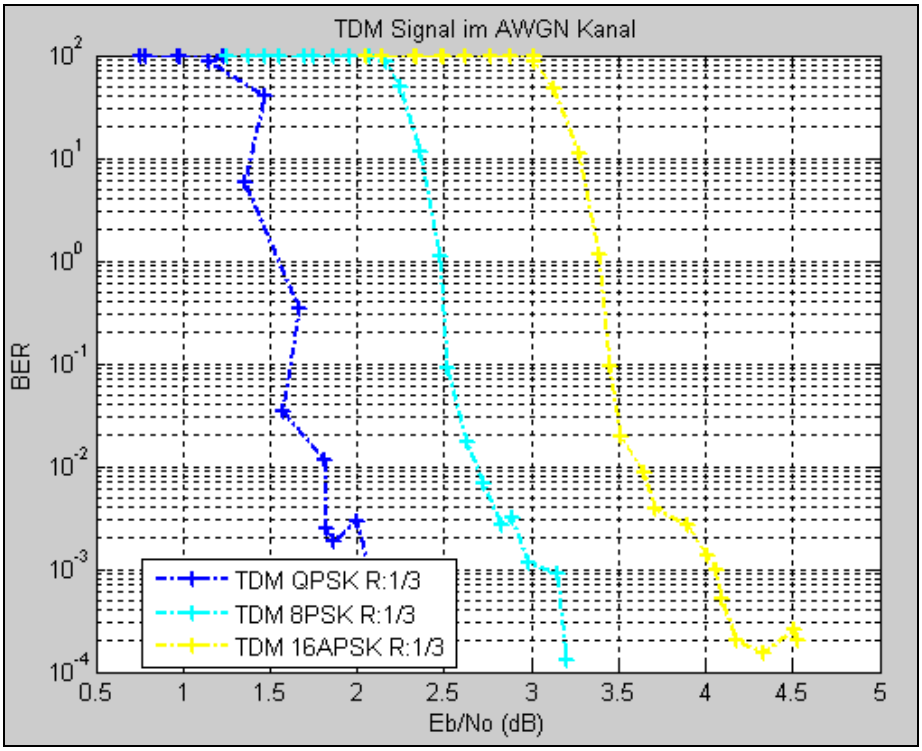
Graph 6 zeigt dass zwischen den Messkurven der Testvarianten A (blau) und B (magenta) ein sichtbarer Unterschied existiert. Diese Erscheinung wird als Hysterese des Messempfängers bezeichnet.



Graph 6: Hysterese des Empfängers

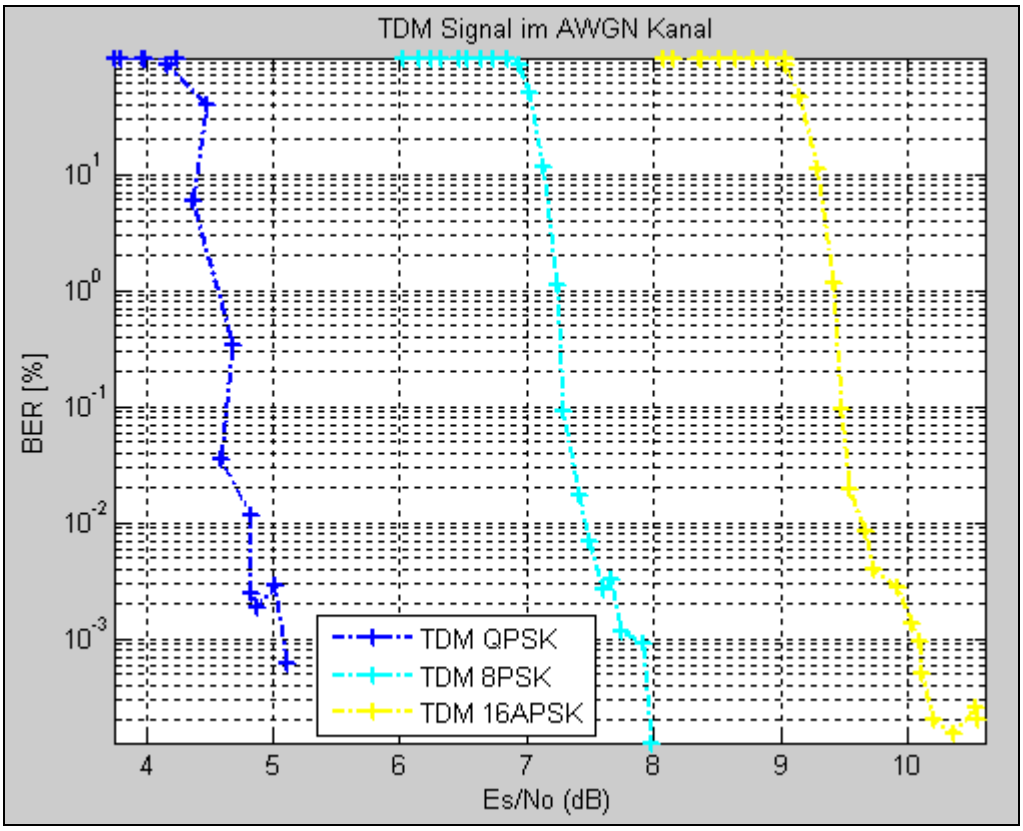
3. Aufgabe

Graph 7 zeigt die drei untersuchten TDM-Signale ohne Modulation und Coderate. Aus diesem Graph kann ermittelt werden, wie stark der Wert des BER steigt, wenn die Bitenergie in Abhängigkeit von der Rauschleistungsdichte sinkt.



Graph 7: TDM Signale (Modulation und Coderate herausgerechnet)

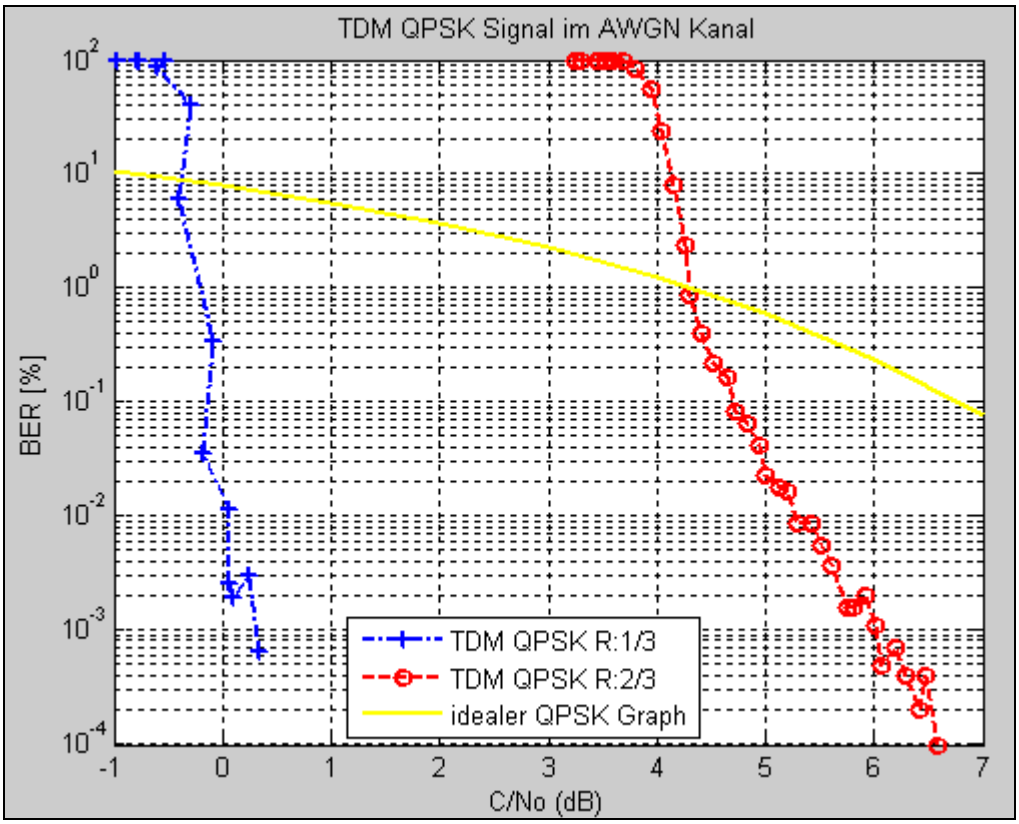
Graph 8 veranschaulicht die BER als Funktion der Symbolenergie pro Rauschleistungsdichte. In diesem Graph kann die Wirkung der unterschiedlichen Modulationen untersucht werden. Wie in Kapitel 3.4.2 erwähnt, existiert zwischen QPSK und 16APSK ein Unterschied von 3 dB. Die 8PSK Modulation liegt ungefähr in der Mitte, da sie 3 Bit pro Symbol enthält. Aus den Betrachtungen kann der Schluss gezogen werden, dass eine QPSK-Modulation eine sichere Übertragung gewährleistet, auch wenn starke Störungen im Kanal vorhanden sind. Ist der Übertragungskanal jedoch relativ störungsfrei, können mit 16APSK mehr Daten transportiert werden.



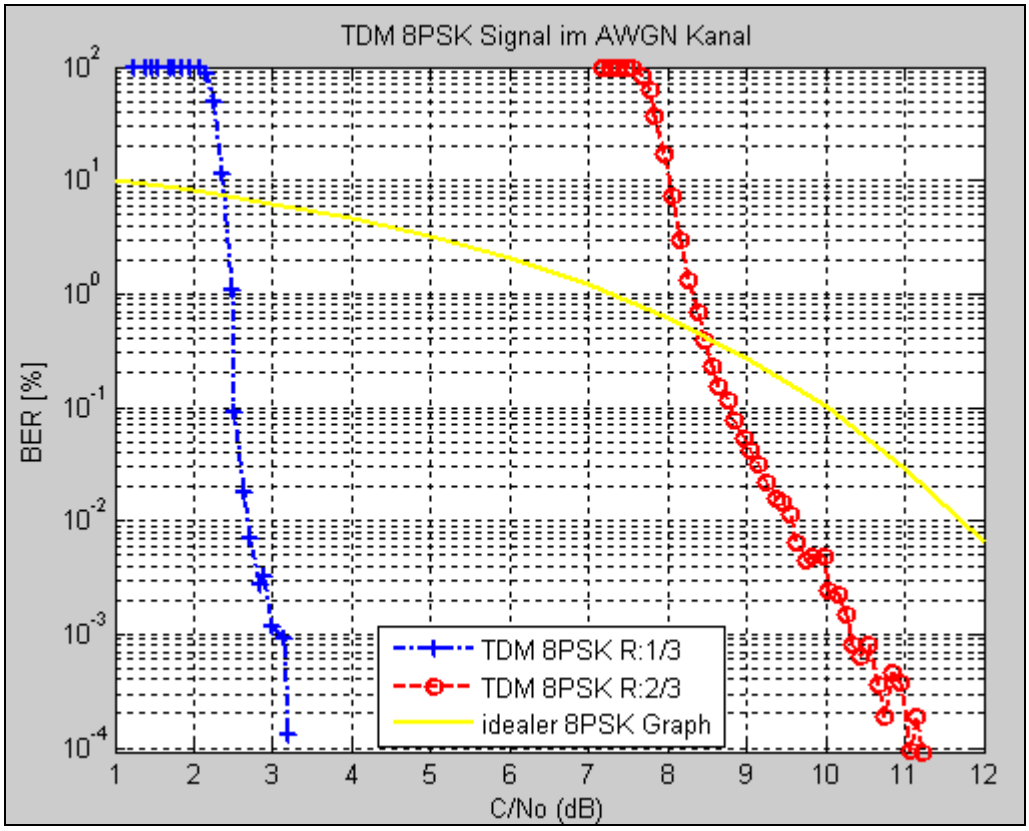
Graph 8: TDM Signale (Coderate herausgerechnet)

Aufgabe 4

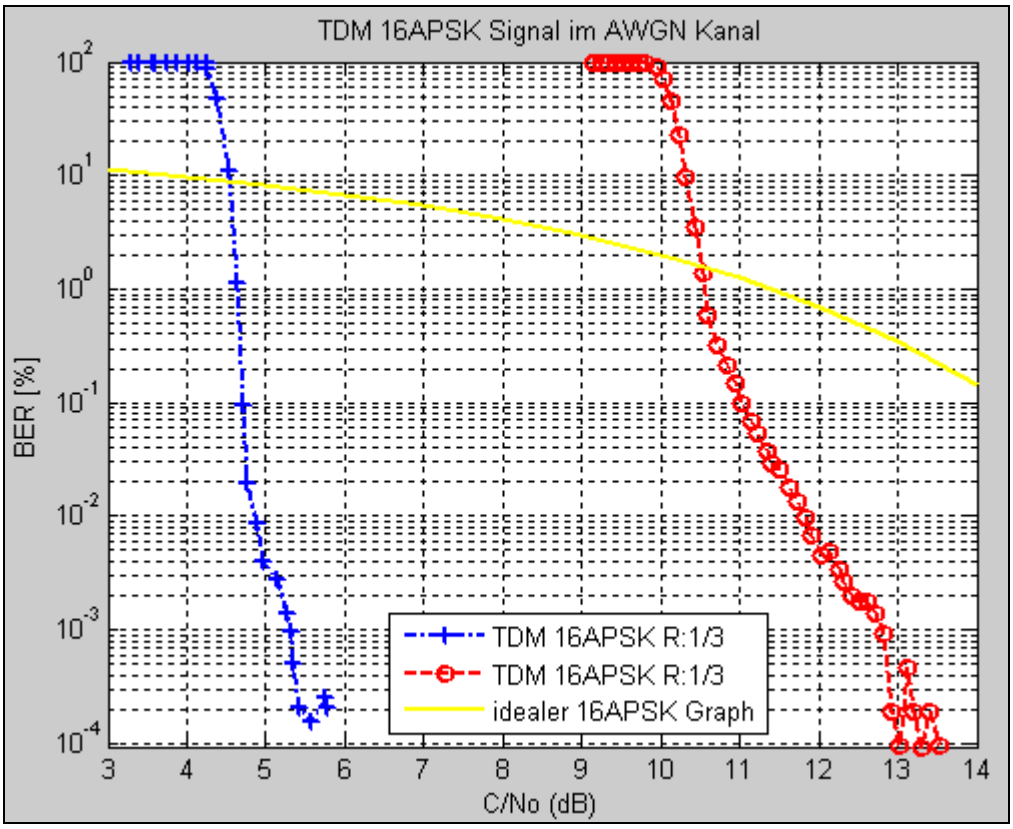
Die folgenden Graphen zeigen das Empfangssignal mit verschiedenen Konfigurationen. In einem Graph werden drei Signale dargestellt. Für diese gilt, dass Modulation und Kanalmodell innerhalb eines Graphen identisch sind. Die gelbe Linie ist der mit Matlab simulierte ideale Verlauf ohne Kanalkodierung.



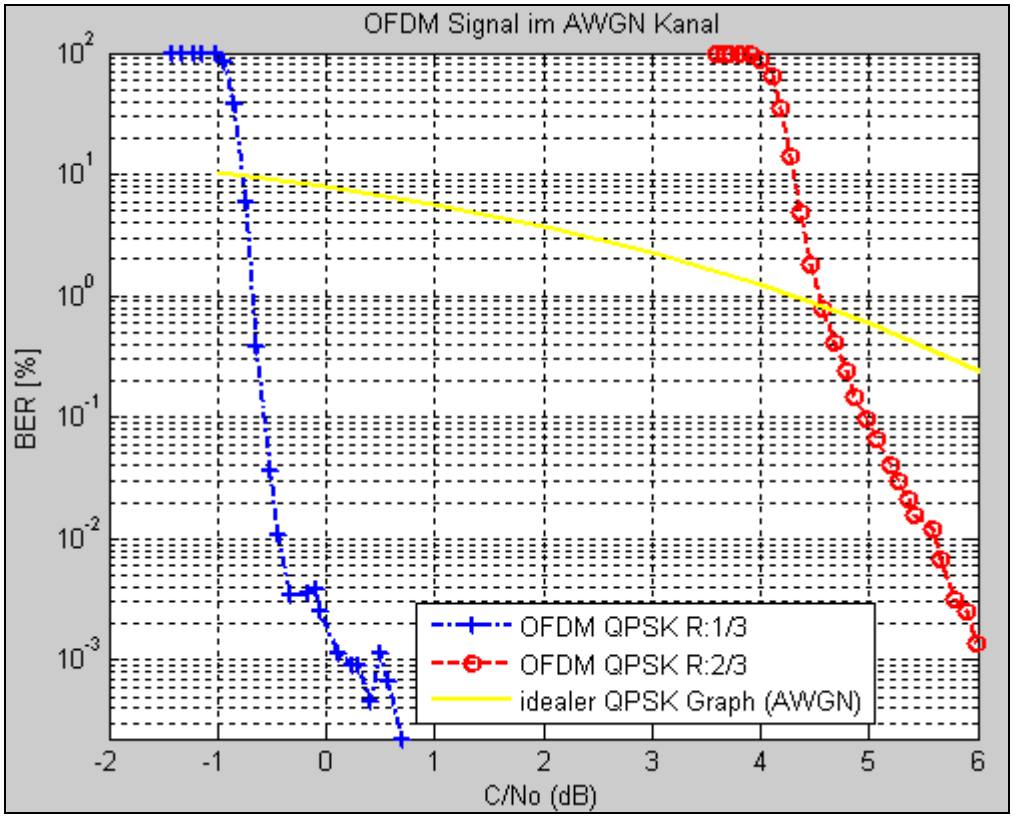
Graph 9: TDM QPSK 1/3 und 2/3



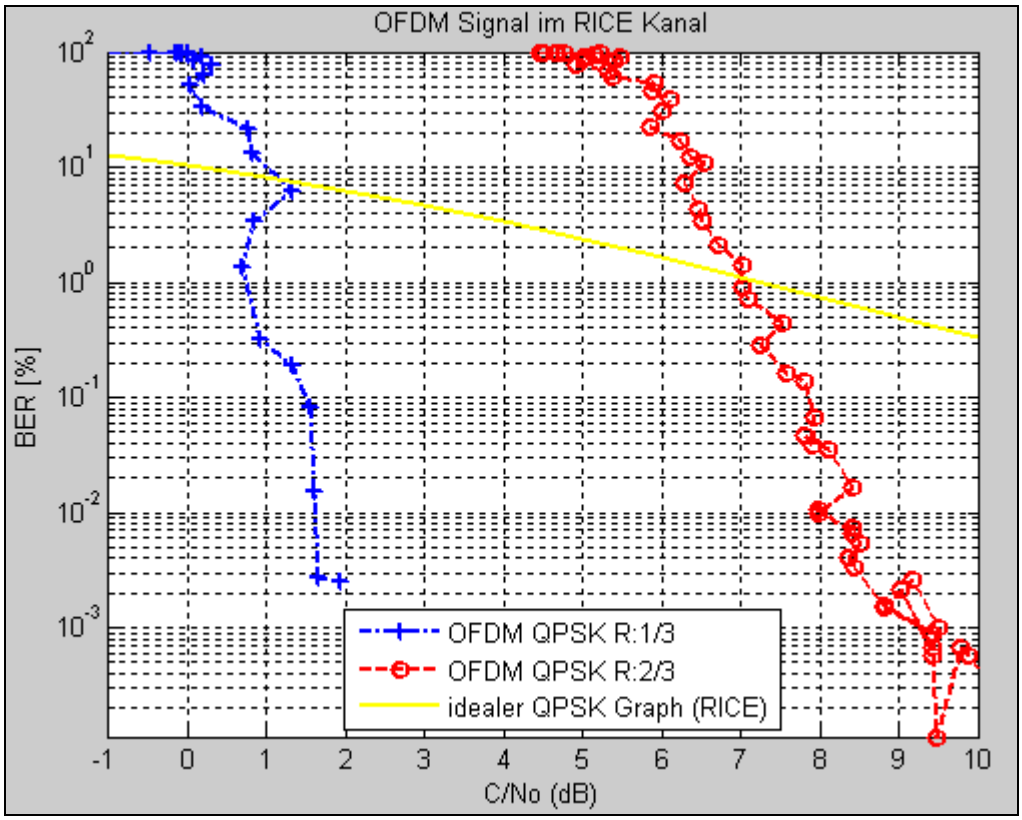
Graph 10: TDM 8PSK 1/3 und 2/3



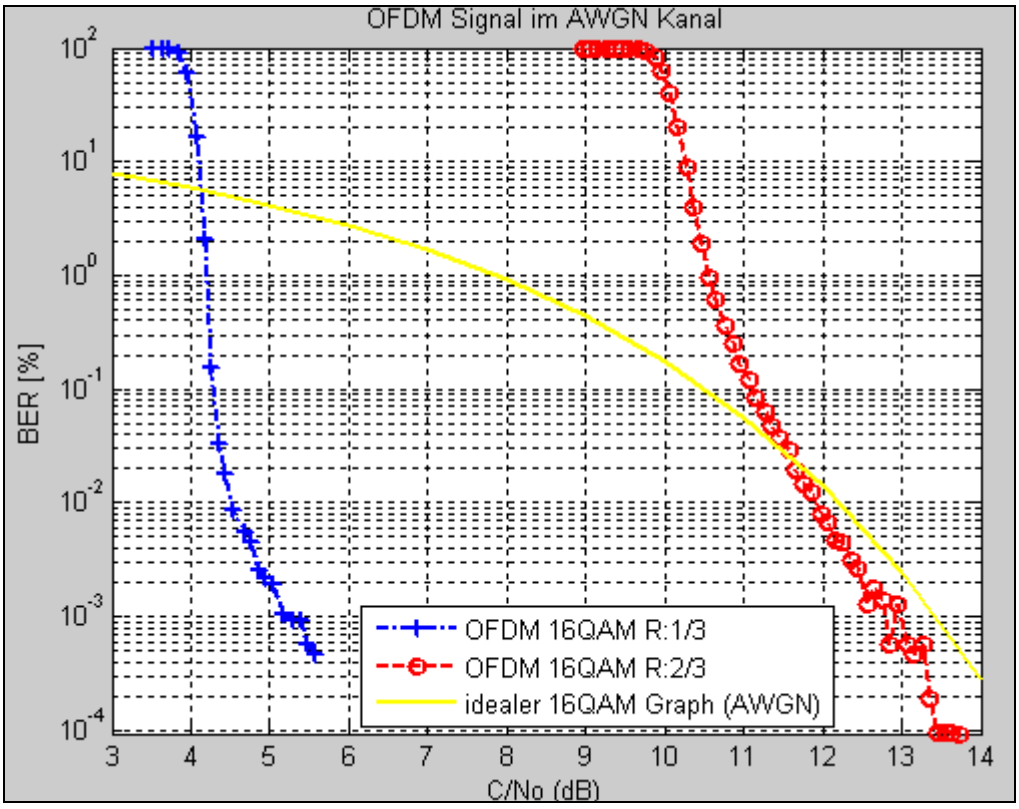
Graph 11: TDM 16APSK 1/3 und 2/3



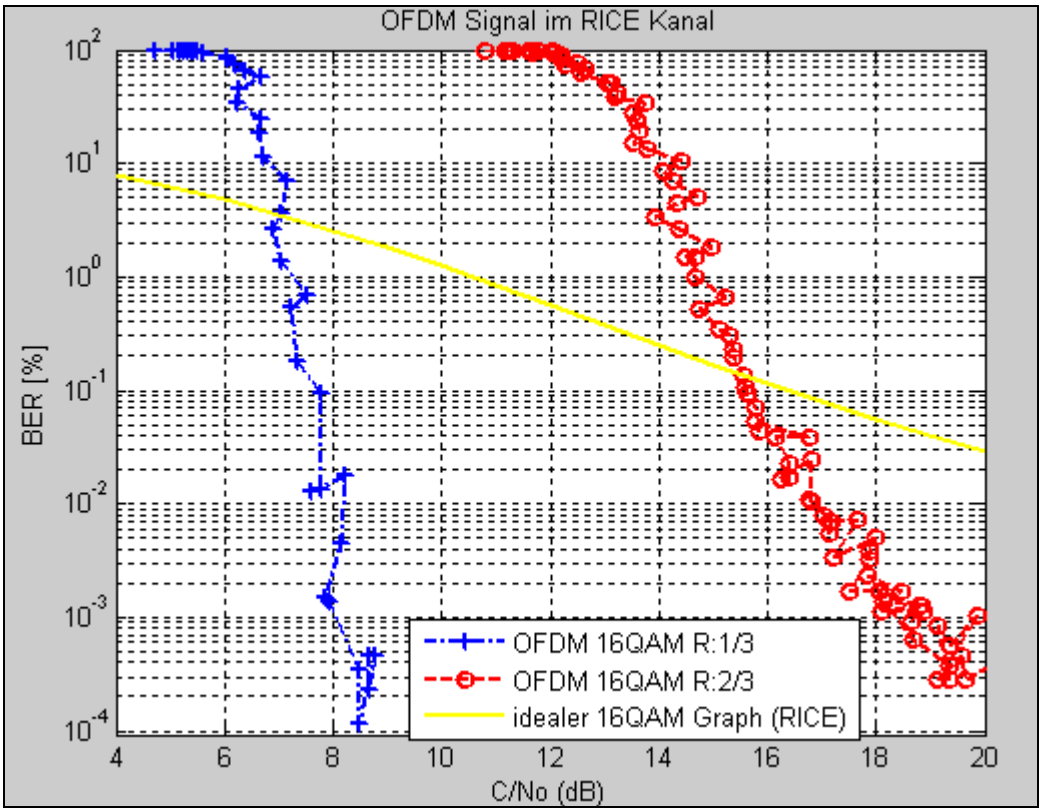
Graph 12: AWGN OFDM QPSK1/3 und 2/3



Graph 13: RICE OFDM QPSK 1/3 und 2/3



Graph 14: AWGN OFDM 16QAM 1/3 und 2/3



Graph 15: RICE OFDM 16QAM 1/3 und 2/3

4 Kapitel: Zusammenfassung und Ausblick

Das erstellte Programm bietet die Möglichkeit, die Paketfehlerrate in Abhängigkeit des C/N_o bzw. E_s/N_o zu messen und darzustellen und somit die Qualität des DVB-SH Testempfängers zu beurteilen. Mit dem aufgebauten Framework können Kanalmodelle simuliert werden, welche die Bedingungen der Übertragungswege vom Satelliten oder von den Repeatern beschreiben. Solche Kanalmodelle wurden in Kapitel 1.3.2 erläutert, Empfehlungen für weitere Kanalmodelle befinden sich in [16]. Das verwendete AWGN-Modell repräsentiert die realen DVB-SH Kanäle nicht ausreichend. Es bietet jedoch eine gute Grundlage, um das automatisierte Messsystem zu entwickeln und zu testen. Das gewählte Rice-Kanalmodell mit K=7 gilt für die Satellitenübertragung in ländlicher Umgebung mit einer langsamen Empfängergeschwindigkeit (Fußgänger mit ca. 3km/h). Die Messergebnisse dieser beiden Konfigurationen zeigen, dass das vorgestellte Messprogramm für den Test eines DVB-SH Empfängers vielseitige Testmöglichkeiten bietet. Die Wirkungen von Modulation und verschiedenen Coderaten sind erkennbar und richtig. Der steile Verlauf der Graphen entspricht den Erwartungen und ist typisch für digitale Übertragungssysteme. Er weist darauf hin, dass mit zunehmenden Störungen die Bitfehlerwahrscheinlichkeit stark ansteigt und die Übertragungsqualität schlagartig abnimmt.

Für das in der vorliegenden Bachelorarbeit vorgestellte Messsystem gibt es noch eine Reihe von Verbesserungs- und Erweiterungsmöglichkeiten. Im Folgenden werden einige Vorschläge beschrieben, die sich aus den aktuellen Resultaten ableiten.

Während der Tests dauerte die BER-Messung für einen Signalleistungspegel 5 Minuten lang. Diese Zeitspanne reicht jedoch nicht aus, um die Qualität des Empfängers nach standardisierten Vorgaben zu beurteilen. Um die Grenzwerte des Empfangspegels zu finden, zu denen ausreichend kleine BER-Werte von z.B. 10^{-6} oder 10^{-7} gehören, muss eine BER-Messung eine bestimmte zeitliche Länge haben. Diese optimale Messdauer wird von den eingestellten Signalparametern bestimmt. Der Zeitfaktor ist für diesen Test sehr wichtig. Da der Zeitbereich eines Messschrittes, bei den durchgeführten Messungen, nicht genügend lang war, kann man in allen Graphen erkennen, dass der Funktionsverlauf nicht monoton ist.

Idealerweise sollten, bei steigenden C/N_0-Werten, die BER-Werte sinken. Es dürfte nicht vorkommen, dass in einer Messreihe zu einem kleineren C/N_0-Wert ein kleinerer BER-Wert gehört. Bei niedrigen BER-Werten ist diese Erscheinung noch ausgeprägter. Der Grund dafür ist, dass um niedrigere Bitfehlerraten mit hoher Genauigkeit messen zu können, viel größere Zeitbereiche (Tage) nötig sind. In 5 Minuten, wie in den durchgeführten Messungen, spielt die Wahrscheinlichkeit des Zufallfehlers eine entscheidende Rolle. Aus den oben genannten Gründen müssen die Messungen zur Beurteilung der Qualität des Empfängers mit deutlich längeren Zeitbereichen ausgeführt werden.

Im Testprogramm kann die Konfiguration für den ESPI und für die Module *„Input“* und *„Channel Simulator“* des DT4500 in Konfigurationsdateien ausgelagert werden. Die Klasse *„DT4500ChannelSimulator“* kann noch um zusätzliche Kanalmodelle erweitert werden. In einer zukünftigen Arbeit sollten diese Punkte noch angegangen werden.

IV. Verzeichnis

Abbildungen

Tabelle

Graph

Quelltext

Syntax

V. Abkürzungen

BER: Bit Error Rate

CENELEC: Europäisches Komitee für elektrotechnische Normung

CGC: Complementary Ground Component

DVB-C: Digital Video Broadcasting- Cable

DVB-H: Digital Video Broadcasting- Handheld

DVB-S: Digital Video Broadcasting- Satellite

DVB-SH: Digital Video Broadcasting- Satellite Services to Handheld Devices

DVB-T: Digital Video Broadcasting- Terrestrial in VHF-UHF Band

ESA: European Space Agency

ETSI: European Telecommunications Standards Institute

FEC: Forward Error Correction

HDTV: High Definition TeleVision

IEC: International Electrotechnical Commission

ISI: Intersymbolinterferenz

ISM: Industry Scientific Medical

ISO: International Organization for Standardization

LMS: Land Mobile Satellite

MPEG: Moving Pictures Experts Group

OFDM: Orthogonal Frequency Division Multiplex)

QoS: Quality of Service

SC: Satellite Component

SNR: Signal to Noise Ratio

TDM: Time Division Multiplex

TR: Terrestrial Repeater

VI. Zitate

Seite 1:
Quelle: http://www.like.e-technik.uni-erlangen.de/nachrichtenarchiv/2009/121009_SenderMobilerRundfunk.shtml

VII. Literatur

[1] http://www.dailynet.de/Telekommunikation/39424.php ;Eutelsat Satellit W2A erfolgreich vom russischen Weltraumbahnhof Baikonor gestartet; 06.04.2009

[2] www.like.e-technik.uni-erlangen.de/

[3] B.Szekeres, L. Nagy; „Antennák és Hullámterjedés"; Budapesti Műszaki és Gazdaságtudományi Egyetem

[4] J.Honfy, „Hullámterjedés és Antennák"; Széchenyi István Főiskola, Győr

[5] https://home.zhaw.ch/~rur/ntm/unterlagen/ntmkap6mobkanal.pdf, ZHW, NTM, 2005/06, Rur

[6] http://www.lnt.e-technik.tu-muenchen.de/download/LNTwin/Texte/MFK/MFK1.PDF

[7] N. Geng, W. Wiesbeck; „Planungsmethoden für die Mobilkommunikation"; Springer; ISBN: 3-540-64778-3

[8] www.nhh.hu

[9] J. Farserotu, R. Prasad; "IP/ATM Mobile Satellite Networks"; Artech House; ISBN: 1-58053-112-1

[10] U. Reimers; " DVB-Digitale Fernsehtechnik, Datenkompression und Übertragung für DVB"; Springer; 3. Auflage; ISBN: 978-3-540-43490-0

[11] Digital Video Braodcastin Project; www.dvb.org; 2003;

[12] ETSI EN 300 744 V1.6.1 (2009-01)

[13] EN 300 429 V1.2.1 (1998-04)

[14] EN 300 421 V1.1.2 (08/97)

[15] ETSI TR 102 377 V1.4.1 (2009-06)

[16] ETSI TS 102 584 V1.1.1 (2008-12)

[17] ETSI EN 302 583 V1.1.1 (2008-03)

[18] Fraunhofer Institute IIS; „DVB-SH Testequipment DT4500 TE User Manual“; Version 0.3

[19] Rhode&Schwarz; Bedienhandbuch: Funkstörmessempfänger ESPI7

[20] C. Ullenboom; „Java ist auch eine Insel“; Galileo Computing; 7.,aktualisierte und erweiterte Auflage; ISBN: 978-3-8362-1146-8

Literatur für Abbildungen

[a] http://upload.wikimedia.org/wikibooks/en/5/53/Tem_wave.gif

[b] J.Honfy, „Hullámterjedés és Antennák“; Széchenyi István Főiskola, Győr; Seite: 38

[c] https://home.zhaw.ch/~rur/ntm/unterlagen/ntmkap6mobkanal.pdf; ZHW, NTM, 2005/06, Rur;Seite: 6-3

[d] :B.Szekeres, L. Nagy; “Antennák és Hullámterjedés”; Budapesti Műszaki és Gazdaságtudományi Egyetem; Seite: 3

[e] http://www.wirelesscommunication.nl/reference/chaptr03/fading/doppler.htm

[f] Alcatel Lucent; Unlimited Mobile TV initiative in Europe ; International Workshop for B3G/4G Satellite Communications Seoul Education & Cultural Hall, 23 November 2006

[g] https://home.zhaw.ch/~rur/ntm/unterlagen/ntmkap6mobkanal.pdf; Seite:6-16

[h] B. Héder, R. Singliar, G. Szládek; „Csatornamodellek“; Budapesti Műszaki és Gazdaságtudományi Egyetem; 2005

[i] B. Héder, R. Singliar, G. Szládek; „Csatornamodellek“; Budapesti Műszaki és Gazdaságtudományi Egyetem; 2005

[j] G. Söder; Simulation digitaler Übertragungssysteme; Seite 2

[k] http://upload.wikimedia.org/wikibooks/en/f/fd/ Rayleigh_fading_doppler_10Hz.gif

[l] a120.DVB-SH_implementation_guide.pdf; Seite 11

[m] a120.DVB-SH_implementation_guide.pdf; Seite 35

[n] a120.DVB-SH_implementation_guide.pdf; Seite 36

[o] National Instruments; MIMO Kommunikationssysteme für Prüfanwendungen

[p] Rhode&Schwarz; Bedienhandbuch: Funkstörmessempfänger ESPI7; Seite 1

[q] Rhode&Schwarz; Bedienhandbuch: Funkstörmessempfänger ESPI7; Seite 4.146

[r] Fraunhofer Institute IIS; „DVB-SH Testequipment DT4500 TE User Manual“; Version 0.3; Seite 1

[s] DI Peter Knorr; Digitale Satellitenübertragungstechnik DI; September 2009; Seite 9

[t] DI Peter Knorr; Digitale Satellitenübertragungstechnik DI; September 2009; Seite 10

[u] DI Peter Knorr; Digitale Satellitenübertragungstechnik DI; September 2009; Seite 10

[v] http://images.google.de/imgres?imgurl=http://www.sat-steve.de/files/signale/qam.jpg&imgrefurl=http://www.sat-steve.de/signale.htm&usg=__KvvbJvA2kJN4S2CHxy6OsTINyi8=&h=412&w=410&sz=30&hl=de&start=5&um=1&itbs=1&tbnid=ZN-Gtg2AASPvyM:&tbnh=125&tbnw=124&prev=/images%3Fq%3DKonstellation%2B16QAM%26hl%3Dde%26client%3Dfirefox-a%26rlz%3D1R1GPEA_de___DE356%26sa%3DN%26um%3D1

Anhang A: Phänomene in der Erdatmosphäre

Reflexionen:

Im Allgemeinen versteht man unter Reflexionen das Zurückwerfen von Wellen an einer Grenzfläche. Im speziellen Fall der elektromagnetischen Welle treten diese Reflexionen an elektrisch leitfähige Flächen auf. Diese wirken dabei wie ein Spiegel, wobei gilt, dass der Einfalls- gleich dem Ausfallswinkel ist.

Brechungen:

Dich wichtigste Kennzahl dieses Effektes ist die Brechzahl von Medien. Beispielsweise haben die verschiedenen Luftschichten der Atmosphäre unterschiedliche Dichten und damit individuelle Brechzahlen. Dabei gilt je dünner das Medium, desto besser ist die Leitfähigkeit. Beim Durchqueren der Erdatmosphäre wird die elektromagnetische Welle mehrfach gebrochen. Die Brechung erfolgt immer in Richtung des dichteren Mediums.

Absorptionen:

Bei diesem Effekt wird die Energie der elektromagnetischen Welle teilweise oder vollständig absorbiert. Das passiert an Medien an denen die Welle weder gebrochen noch reflektiert wird. Zu diesen Medien zählen beispielsweise Berge, dichte Wolken, Schnee, Nebel, trockenes Erdreich oder Mauerwerk.

Abschattungen:

Die Ausbreitung elektromagnetischer Wellen ist ähnlich der Ausbreitung von Wasserwellen. Im Fall, dass sich absorbierende oder reflektierende Hindernisse im Ausbreitungsweg befinden kommt es hinter dem Hindernis zu einer Schattenzone. Die Größe dieser Zone hängt vom Verhältnis der Wellenlänge zur Ausdehnung des Hindernisses ab. Bei kleinen Hindernissen verformt sich lediglich kurzzeitig der geradlinige Frontverlauf und das Hindernis wird einfach umlaufen, ohne Ausbildung einer Schattenzone.

Beugungen:

Treffen elektromagnetische Wellen bei ihrer Ausbreitung auf Gitter, Spalte oder Kanten werden sie gebeugt. Beachten muss man hierbei, dass es beim Aufeinandertreffen der Direktwelle und ihrer gebeugten Anteile an Gittern zur

Mehrfachbeugung kommt. Dadurch entstehen Interferenzen, die sich oft als Fading-Störungen bemerkbar machen.

Ausbreitungswege:

Ein Kugelstrahler strahlt eine gewisse Größe der Energie parallel mit der Erdoberfläche aus. Der andere Teil der Energie wird unter verschiedenen Höhenwinkeln ausgestrahlt. Die sich entlang der Erde ausbreitenden Wellen heißen Bodenwellen, die anderen Raumwellen. Abbildung 25 stellt die verschiedenen Ausbreitungswege grafisch dar.

Boden- oder Oberflächenwelle

Die Oberflächenwelle folgt der Oberflächenstruktur der Erde und ist verschiedenen absorbierenden Einflüssen ausgesetzt. Die Dämpfung ändert sich mit der elektrischen Leitfähigkeit der Erde. Die Reichweite der Bodenwelle ist über Feuchtgebieten und Wasserflächen größer, da die Leitfähigkeit im Vergleich zum normalen Erdboden und Felsen besser ist. Die Bodenwelle hat für den Empfang von Langwellensendern die größte Bedeutung.

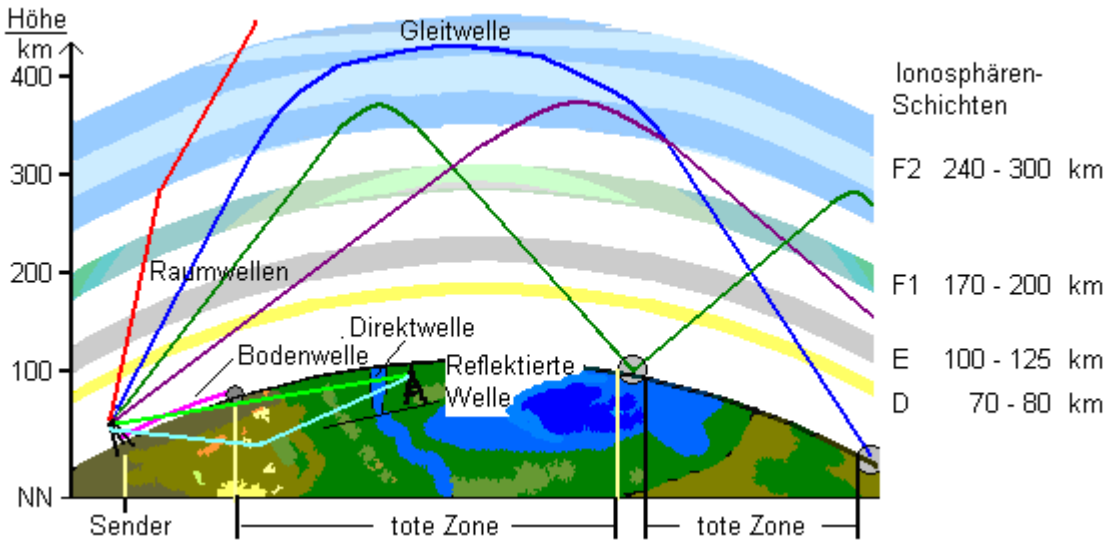


Abbildung 25: Ausbreitungswege

Direkte Welle

Die Direktwelle breitet sich Im VHF- und UKW-Bereich, 30 MHz ... 300 MHz und darüber im UHF-Bereich 300 MHz bis 3 GHz bis zum optische Horizont aus. Die Reichweite kann sich durch Beugungsvorgänge in der Troposphäre verschieben und über den optischen Horizont hinausgeschoben werden. Eine

Brechung oder Reflexion an Ionosphärenschichten findet mit zunehmender Frequenz nicht mehr statt.

Reflektierte Welle:

Die reflektierte Welle enthält den Teil der Energie, welcher bis zu einer Oberfläche gradlinig vorankommt, und davon geht reflektiert bis zum Antenne weiter. Reflektionsoberflächen sind der Erdboden, Ionosphäre und Troposphäre. Die direkten und reflektierten Wellen bilden zusammen die Raumwellen. Die Feldstärke bei der Empfängeranlage ergibt sich aus dem Zusammenspiel dieser Wellen. Über 30 MHz muss mit Raumwellen gerechnet werden. Die Anteile der elektromagnetischen Strahlung, die aufgrund des Abstrahlwinkels in den freien Raum abwandern und durch Dämpfung und Krümmung des Erdbodens nicht beeinflusst werden, bezeichnet man als Raumwelle. Die Ausbreitung der Raumwellen werden von drei Faktoren beeinflusst: Die verschiedenen Sonnenaktivitäten, der Sonnenstand und dadurch bedingt die Jahres- und Tageszeiten bestimmen ebenfalls die Frequenzbereiche, die sich für eine Raumwellenverbindung eignen.

Ionosphärische Welle

Die Ionosphärenschichten sind nicht einheitlich und haben von der Sonnen- und Höheneinstrahlung abhängig unterschiedliche Eigenschaften. Periodisch auftretende Sonnenaktivitäten verursachen mit den Partikeln des Sonnenwindes ein Zusammenbrechen einiger Ionosphärenschichten und verhindern so immer wieder für einige Frequenzbänder den Funkverkehr. Eine besonders stark ionisierte Gasschicht trägt den Namen Heavyside-Schicht. Die unsicherste D-Schicht, die nur am Tag wirksam ist, reflektiert LW-Signale und absorbiert KW-Signale. Die E-Schicht liegt über der D-Schicht. Ihre Intensität ist von Sonnenstand abhängig. Sie reflektiert die längeren Kurzwellen. Die F1-Schicht ist nur tagsüber und im Sommer wirksam. Ihr Verhalten ist ähnlich der E-Schicht, allerdings liegen ihre Grenzfrequenzen noch etwa 50% höher. Die F2-Schicht ist nachts auch wirksam und damit ist ein interkontinentaler Funkverkehr über 24 Stunden möglich. Sie ist die unregelmäßigste Schicht. Die Grenzfrequenzen sind abhängig von der Sonnenaktivität und von Sonnenstand. Die Aktivität der Sonne schwankt in einem elfjährigen Rhythmus. Dieser Rhythmus nennt man auch Sonnenfleckenzyklus. Aus dem berechenbaren Sonnestand und der

vorhergesagten Sonnenfleckenzahl lässt sich der Zustand der Ionosphäre voraussagen.

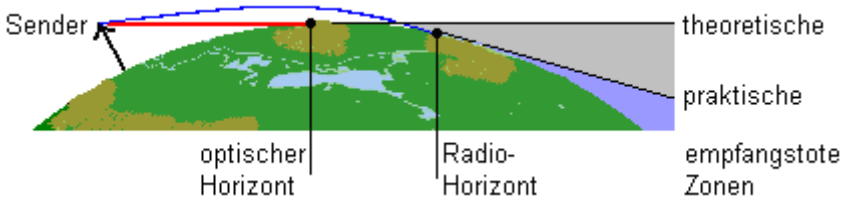


Abbildung 26: Horizonte

Eine Besonderheit ist die Gleitwelle, die durch eine Vielfachbrechung innerhalb der F2-Schicht erst nach einer sehr weiten Strecke wieder zur Erde zurück gelangt. Der Bereich zwischen der Reichweite der Bodenwelle und dem Wiederempfang des Senders über die erste reflektierte Raumwelle wird tote Zone genannt. Die Entfernung wird als Sprungdistanz bezeichnet. Sie ist abhängig von der Sendefrequenz, dem Abstrahlwinkel, der Tages- und Jahreszeit. Mit der F2-Schicht kann tagsüber eine maximale Sprungdistanz von 4000 km und für die E-Schicht bis 2000 km erreicht werden. Für jeden Einstrahlwinkel kann eine kritische Frequenz ermittelt werden, oberhalb der keine Beugung mehr erfolgt. Die Welle durchdringt die Ionosphärenschichten und strahlt in den freien Raum ab.

Anhang B: Satellitenlaufbahnen

GEO: Die Satteliten kreisen in einer Höhe von 36000 km. Die Umlaufzeit beträgt genau einen Sternentag, sie scheinen von der Erde aus gesehen also stillzustehen. Wegen der großen Entfernung von der Erde werden die Signale zwischen der Erdstation und dem Satellit stark gedämpft (-205 dB bei 10 GHz) und um zirka 240 (Erde-Satellit) ms verzögert.

MEO, *Medium Earth Orbit*: Entfernung von der Erde ist 10000-20000 km.

LEO, *Low Earth Orbit*: Die Laufbahnhöhe liegt zwischen 500 und 5000 km. Mit dieser Satellitenbahn können die Randgebiete (Nordpol/Südpol) abgedeckt werden. Wegen der kleinen Entfernung ist die Signaldämpfung- und -verzögerung wesentlich niedriger, aber die Satelliten „wandern“ auf dem Himmel. Das bedeutet, dass ein Satellit nur 10-30 Minuten lang von der Erde aus sichtbar ist, deshalb sind mehrere Satelliten zur ständigen Kommunikationsverbindung nötig.

HEO, *High Elliptical Orbit*: Auf der elliptischen Laufbahn, welche sogar 40000 km von der Erde entfernt sein kann, bewegen sich die Satelliten sehr langsam, aus diesem Grund haben sie eine lange Sichtbarkeitszeit. Ungefähr 70 % der Laufbahn befinden sie sich über dem zu versorgenden Gebiet. Nur 30 % der Umlaufzeit verbringen sie auf der anderen Seite der Erde. Aus diesem Funkschatten kehren die Satelliten relativ schnell wieder zurück, weil die Laufbahnhöhe hier auf einige 1000 km verringert wird. Ein spezieller Fall ist die Molnyija-Bahn, worauf 3 Satelliten mit 8 Stunden Umlaufzeit die Randgebiete versorgen. Globale Abdeckung wird bei GEO und HEO Umlaufbahnen mit 3 Satelliten, bei MEO 10-15, bei LEO 40-60 Satelliten erreicht.

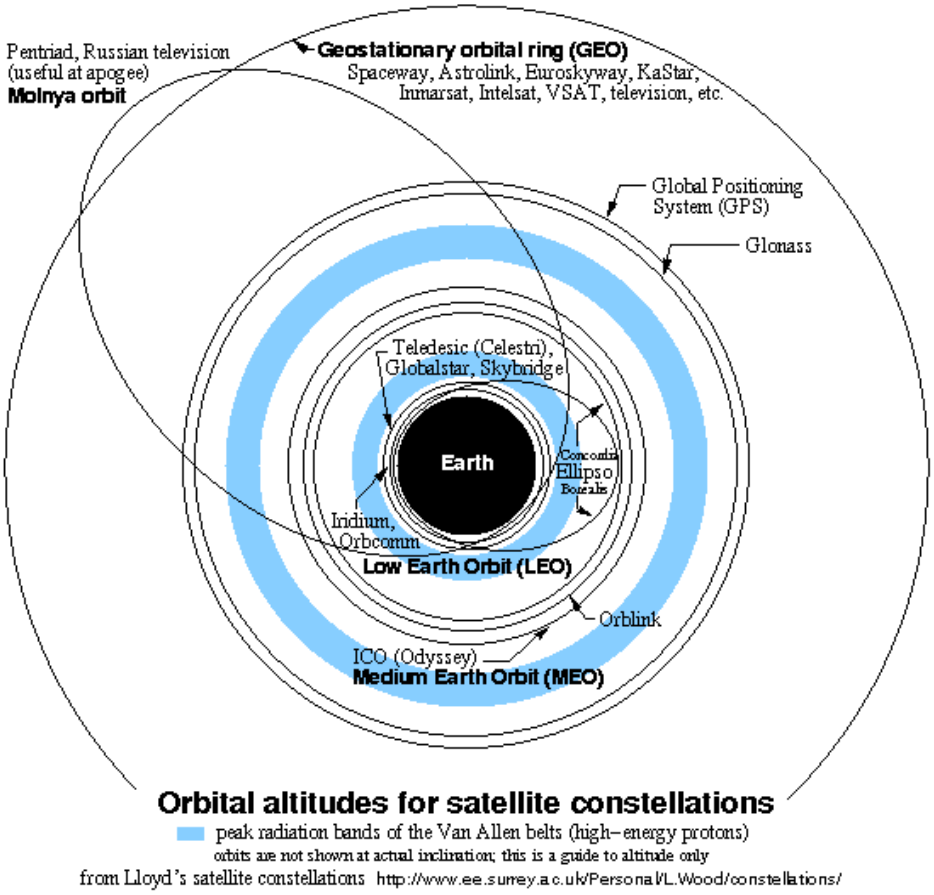
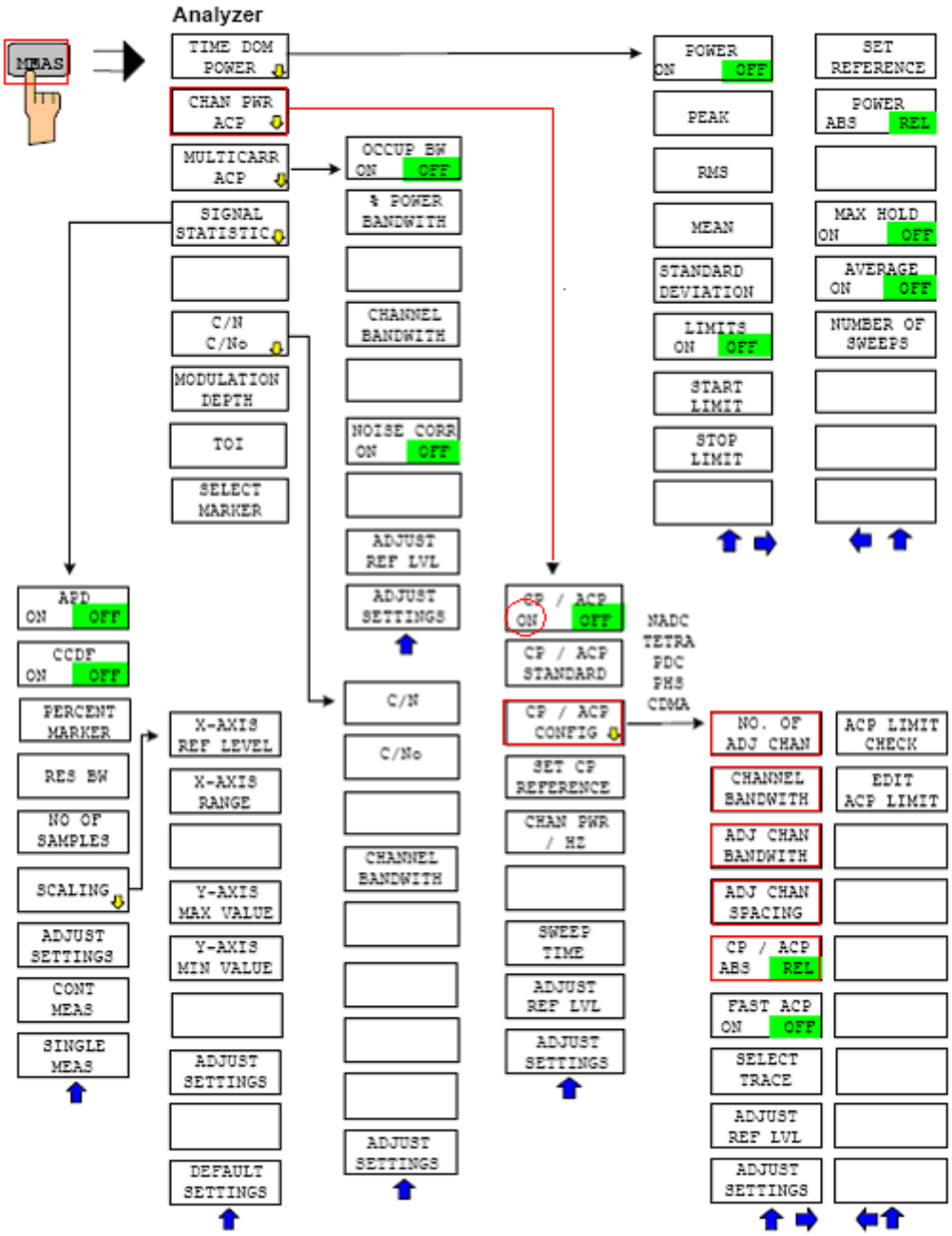


Abbildung 27: Satellitenlaufbahn

Anhang C: Menüübersicht



Anhang D.1: Package ESPI

```
package espi;

import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.net.Socket;

public class ESPICommunicator {
     static int MAX_RETRYS = 10;
     static String FREQ_SPAN = "FREQ:SPAN";
     static String FREQ_CENT = "FREQUENCY:CENTER";
     static String SWEEP_TIME = "SWE:TIME";
     static String RBW = "BAND:RES";
     static String VBW = "BAND:VID";
     static String REF_LEVEL = "DISP:WIND:TRAC:Y:RLEV";
     static String PREDEFINED_STANDARD = "CALC:MARK:FUNC:POW:PRES";
     static String NO_OF_ADJ_CHAN = "POW:ACH:ACP";
     static String CHANNEL_BANDWIDTH = "SENS:POW:ACH:BWID:CHAN";
     static String ADJ_CHANNEL_BANDWIDTH = "SENS:POW:ACH:BWID:ACH";
     static String ADJ_CHANNEL_SPACING = "SENS:POW:ACH:SPAC:ACH";
     static String CP_ACP_ABS_REL = "SENS:POW:ACH:MODE";
     static String CHAN_PWR_HZ = "CALC:MARK:FUNC:POW:RES:PHZ";
     static String INP_ATT = "INP:ATT";
     String hostname;
     int port;
     InputStream iStream;
     OutputStream oStream;
     Socket socket;

     public ESPICommunicator(String hostname, int port) throws Exception
{
          this.hostname = hostname;
          this.port = port;

          socket = new Socket(this.hostname, this.port);
          iStream = socket.getInputStream();
          oStream = socket.getOutputStream();
     }

     public void sendCommand(String cmd) throws IOException {
          System.out.println("ESPI Command: " + cmd);

          cmd += "\n";
          oStream.write(cmd.getBytes());
     }

     public String getResponse(String cmd) throws IOException {
          String response = new String();

          sendCommand(cmd);

          int c = 0;
          while (c != 0x0a && c != 0x0d) {
               c = iStream.read();
               if (c != 0x0a && c != 0x0d) {
                    response += (char) c;
               }
          }
```

```
            System.out.println("ESPI Response: " + response);

            return response;
        }

        private void executeCmd(String cmd) {
            for (int i = 0; i < MAX_RETRYS; i++) {
                try {
                    sendCommand(cmd);
                    return;
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            System.err.println("Write command failed: " + cmd);
            System.exit(1);
        }

        private void executeResponse(String cmd, double value) {
            for (int i = 0; i < MAX_RETRYS; i++) {
                try {
                    String response = getResponse(cmd + "?");
                    double res = Double.parseDouble(response);
                    if (res == value) {
                        // if (response.compareTo(value) == 0) {
                        System.out.println("ok");
                        return;
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }

            System.err.println("Read command failed: " + cmd + " " +
value);
            System.exit(1);
        }

        private String getString(String cmd, String variable) {
            String var = variable.toUpperCase();
            for (int i = 0; i < MAX_RETRYS; i++) {
                try {
                    String response = getResponse(cmd + "?");
                    if ((var.contains("ON")) && (response.contains("1")))
{
                        System.out.println("ok");
                        return response;
                    } else if ((var.contains("OFF")) &&
(response.contains("0"))) {
                        System.out.println("ok");
                        return response;
                    } else if (response.contains(var)) {
                        System.out.println("ok");
                        return response;
                    }
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
            System.err.println("Read command failed: " + cmd + " " +
variable);
            System.exit(1);
```

```
            return null;
        }

        private double getHz(String freq) {

            String lower = freq.trim().toLowerCase();

            if (lower.contains("ghz")) {
                return Double.parseDouble(lower.substring(0,
lower.indexOf("ghz"))) * 1000 * 1000 * 1000;
            } else if (lower.contains("mhz")) {
                return Double.parseDouble(lower.substring(0,
lower.indexOf("mhz"))) * 1000 * 1000;
            } else if (lower.contains("khz")) {
                return Double.parseDouble(lower.substring(0,
lower.indexOf("khz"))) * 1000;
            } else if (lower.contains("hz")) {
                return Double.parseDouble(lower.substring(0,
lower.indexOf("hz")));
            } else {
                System.err.println("Invalid frequency: " + freq);
                System.exit(1);
            }

            return 0;
        }

        private double getSec(String sweep_time) {

            String lower = sweep_time.trim().toLowerCase();

            if (lower.contains("ms")) {
                double i = Double.parseDouble(lower.substring(0, lower
                        .indexOf("ms"))) / 1000;
                return i;
            } else {
                System.err.println("Invalid sweep time: " + sweep_time);
            }

            return 0;
        }

        private double getdBm(String level) {

            String lower = level.trim().toLowerCase();

            if (lower.contains("dbm")) {
                double i = Double.parseDouble(lower.substring(0, lower
                        .indexOf("dbm")));
                return i;
            } else {
                System.err.println("Invalid level: " + level);
            }

            return 0;
        }

        // ------------------------------------------------------------------

        public void reset() {
            executeCmd("*RST");
        }
```

```
public void setSpan(double spanHz) {
     executeCmd(FREQ_SPAN + " " + String.valueOf(spanHz));
     executeResponse(FREQ_SPAN, spanHz);
}

public void setSpan(String span) {
     setSpan(getHz(span));
}

public void setCenterFreq(double freqHz) {
     executeCmd(FREQ_CENT + " " + String.valueOf(freqHz));
     executeResponse(FREQ_CENT, freqHz);
}

public void setCenterFreq(String freq) {
     setCenterFreq(getHz(freq));
}

public void setSweepTime(double sec) {
     executeCmd(SWEEP_TIME + " " + String.valueOf(sec));
     executeResponse(SWEEP_TIME, sec);
}

public void setSweepTime(String msec) {
     setSweepTime(getSec(msec));
}

public void setRBW(double freq) {
     executeCmd(RBW + " " + String.valueOf(freq));
     executeResponse(RBW, freq);
}

public void setRBW(String freq) {
     setRBW(getHz(freq));
}

public void setVBW(double freq) {
     executeCmd(VBW + " " + String.valueOf(freq));
     executeResponse(VBW, freq);
}

public void setVBW(String freq) {
     setVBW(getHz(freq));
}

public void setRefLevel(double level) {
     executeCmd(REF_LEVEL + " " + level);
     executeResponse(REF_LEVEL, level);
}

public void setRefLevel(String level) {
     setRefLevel(getdBm(level));
}

public void setRFAtten(int value) {
     executeCmd(INP_ATT + " " + value + " DB");
     executeResponse(INP_ATT, value);
}

public void predefinedStandard(String standard) throws IOException {
     executeCmd(PREDEFINED_STANDARD + " " + standard);
     getString(PREDEFINED_STANDARD, standard);
}
```

```
    public void numberOfAdjChan(int number) {
        executeCmd(NO_OF_ADJ_CHAN + " " + number);
        executeResponse(NO_OF_ADJ_CHAN, number);
    }

    public void chanBandwidth(double freq) {
        executeCmd(CHANNEL_BANDWIDTH + " " + freq);
        executeResponse(CHANNEL_BANDWIDTH, freq);
    }

    public void chanBandwidth(String freq) {
        chanBandwidth(getHz(freq));
    }

    public void adjChanBandwidth(double freq) {
        executeCmd(ADJ_CHANNEL_BANDWIDTH + " " + freq);
        executeResponse(ADJ_CHANNEL_BANDWIDTH, freq);
    }

    public void adjChanBandwidth(String freq) {
        adjChanBandwidth(getHz(freq));
    }

    public void adjChanSpacing(double freq) {
        executeCmd(ADJ_CHANNEL_SPACING + " " + freq);
        executeResponse(ADJ_CHANNEL_SPACING, freq);
    }

    public void adjChanSpacing(String freq) {
        adjChanSpacing(getHz(freq));
    }

    public void cpAcpAbsRel(String abs_rel) throws IOException {
        executeCmd(CP_ACP_ABS_REL + " " + abs_rel);
        getString(CP_ACP_ABS_REL, abs_rel);
    }

    public void chanPwrHz(String onoff) throws IOException {
        executeCmd(CHAN_PWR_HZ + " " + onoff);
        getString(CHAN_PWR_HZ, onoff);
    }

    public String getSNR() throws IOException {
        System.out.println("\n"
                + "TX Chanel Power and Adjacent Chanel Power
Lower/Upper:");

        executeCmd("CALC:MARK:FUNC:POW:SEL ACP"); // CHAN PWR ACP ON
        String response = getResponse("CALC:MARK:FUNC:POW:RES? ACP");
        System.out.println(" ");
        return response;
    }
    public void stopSNR(){
        executeCmd("CALC:MARK:FUNC:POW OFF");
    }

    public static void main(String[] args) throws Exception {
        ESPICommunicator espi = new ESPICommunicator("192.168.255.218",
5025);
        espi.getSNR();
    }
}
```

Anhang D.2: Package DT4500

Class DT4500Communicator

```
package dt4500;

import java.io.IOException;

import org.apache.commons.httpclient.HttpClient;
import org.apache.commons.httpclient.methods.GetMethod;
import org.jdom.JDOMException;


public class DT4500Communicator {
     // ------------------------------------variable--------------------
     ReadDT4500ConfigFile config;

     private static HttpClient client;
     private static String host;
     private static int port;

     protected final static int retrySend = 20;

     // ----------------------------------Constructor--------------------

     public DT4500Communicator(String host, int port) {
          DT4500Communicator.host = host;
          DT4500Communicator.port = port;
          client = new HttpClient();
          initWebFrontend();
          try {
               config = new ReadDT4500ConfigFile("DT4500ConfigFile");
          } catch (JDOMException e) {
               e.printStackTrace();
          } catch (IOException e) {
               e.printStackTrace();
          }
     }

     // ---------------------------------------------------------------

     protected void initWebFrontend() {
          String url = new String("http://" + host + ":" +
Integer.toString(port)
                    + "/remote-access");

          GetMethod get = new GetMethod(url);
          client = new HttpClient();
          try {
               client.executeMethod(get);
               return;
          } catch (Exception e) {
               e.printStackTrace();
          }

          System.err.println("Error");
          System.exit(1);
     }
```

```
    // --------------------------------------------------------------------

    public String sendCommand(String cmd) {
        String url = new String("http://" + host + ":" +
Integer.toString(port)
                + "/remote-access?" + cmd);
        System.out.println("DT4500K URL:" + url);
        GetMethod get = new GetMethod(url);
        client = new HttpClient();
        try {
            client.executeMethod(get);
            Thread.sleep(100);
        } catch (Exception e) {
            e.printStackTrace();
        }

        try {
            String response = get.getResponseBodyAsString();
            System.out.print("DT4500K Response: " + response);
            get.releaseConnection();
            return response;
        } catch (Exception e) {
            e.printStackTrace();
            get.releaseConnection();
            return null;
        }
    }

    public String getVariable(String variable) {
        String cmd = "GetParameter=" + variable;
        return sendCommand(cmd);
    }

    public String getValue(String adress) {
        String cmd = "GetParameter=" + adress;
        for (int i = 0; i < retrySend; i++) {
            try {
                String result = sendCommand(cmd);

                if (resultIsOK(result)) {
                    String value = new String("Value: ");
                    int pos = result.indexOf(value) + value.length();
                    return result.substring(pos);
                }
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
        System.exit(1);
        return null;
    }

    protected boolean resultIsOK(String result) {
        if (result.contains("Status: Ok")) {
            return true;
        } else {
            return false;
        }
    }

    public boolean control(String address, double value, String print) {
        Double control = Double.parseDouble(getValue(address));
        if ((control != value)) {
```

```
            System.err.println("Invalid Value");
            System.err.println(control);
            System.err.println(value);
            System.exit(1);
            return false;
        } else {
            System.out.println("ok");
        }
        return true;
    }

    public void reset() {
        System.out.println("\nReset:");
        sendCommand("Function.OnOff=0");
        sendCommand("Function.Relay1=0");
        sendCommand("Function.Relay2=0");
        sendCommand("Input.Ch0.Source=0");
        sendCommand("Input.Ch1.Source=0");
        sendCommand("Input.Ch2.Source=0");
        sendCommand("ChanSim.PlpEnable=0");
        sendCommand("ChanSim.Ch0.Enable=0");
        sendCommand("ChanSim.Ch0.Plp.Enable=0");
        sendCommand("ChanSim.Ch1.Plp.Enable=0");
        sendCommand("ChanSim.Ch2.Plp.Enable=0");
        sendCommand("Output.EnableRefClock=0");
        sendCommand("Output.Dac0.OutputSel=0");
        sendCommand("Output.Dac1.OutputSel=0");
        sendCommand("Output.Dac0.Ch0.Enable=0");
        sendCommand("Output.Dac0.Ch1.Enable=0");
        sendCommand("Output.Dac0.Ch2.Enable=0");
        sendCommand("Output.Dac1.Ch0.Enable=0");
        sendCommand("Output.Dac1.Ch1.Enable=0");
        sendCommand("Output.Dac1.Ch2.Enable=0");
        sendCommand("Output.Dac0.Noise.Enable=0");
        sendCommand("Output.Dac1.Noise.Enable=0");
        System.out.println("Reset End");
    }

    // --------------------------------FunctionStatus-------------------

    public String getFunctionStatus() {
        for (int i = 0; i < retrySend; i++) {
            String result = getVariable("Function.Status");
            if (result.contains("Status: Ok")) {
                if (result.contains("Value: ready")) {
                    return new String("ready");
                } else if (result.contains("Value: operating")) {
                    return new String("operating");
                } else if (result.contains("do_init_lo")) {
                    i--;
                } else if (result.contains("check")) {
                    i--;
                } else if (result.contains("prefill")) {
                    i--;
                } else if (result.contains("pre_operating")) {
                    i--;
                } else if (result.contains("armed")) {
                    sendCommand("Function.S2=1");
                    i--;
                } else if (result.contains("force_trigger")) {
                    i--;
                } else if (result.contains("error")) {
                    System.err.println("DT4K Error: " + result);
```

```
                    System.exit(-1);
                }
            }
        }

        return null;
    }

    public boolean setFunctionStatus(boolean onoff) throws Exception {
        String address = new String("Function.OnOff");
        String returnCmd = new String();
        System.out.println("\nSet: Function Status");
        if (onoff == true) {
            returnCmd = "operating";
        } else if (onoff == false) {
            returnCmd = "ready";
        }
        if (!getFunctionStatus().contains(returnCmd)) {
            for (int i = 0; i < retrySend; i++) {
                enableDisable(address, onoff);
                System.out.println("\nControl Function Status:");
                Thread.sleep(10000);
                if (getFunctionStatus().contains(returnCmd)) {
                    return true;
                } else {
                    System.out.println("Error");
                    sendCommand("Function.OnOff=Disabled");
                }
            }
        } else {
            return true;
        }

        return false;
    }

    // -------------------------------EnableDisenable-------------------

    public String enableDisable(String address, boolean value) throws
Exception {
        String expectedResponse = new String();
        String varBackup = address;

        if (value == true) {
            address += "=Enabled";
            expectedResponse = "Value: 1";
        } else if (value == false) {
            address += "=Disabled";
            expectedResponse = "Value: 0";
        }

        String returnCmd = sendCommand(address);
        for (int i = 0; i < retrySend; i++) {
            try {
                if (returnCmd.contains("Status: Ok")) {
                    if (returnCmd.contains(expectedResponse)) {
                        System.out
                                .println("DT4100 " + address + ":
successful");
                        return expectedResponse;
                    }
                } else if (returnCmd.contains("Error Timeout")) {
                    System.err.println("DT4100 " + address
```

```
                                        + ": timed out, retrying");
                              returnCmd = getVariable(varBackup);

                         } else if (returnCmd.contains("Warning Value
differs")) {
                              System.err.println("DT4100 " + address
                                        + ": value differs, retrying");
                              returnCmd = sendCommand(address);

                         } else {
                              throw new Exception();
                         }
                    } catch (Exception e) {
                         e.printStackTrace();
                    }
               }
               System.err.println("Error");
               System.exit(1);
               throw new Exception();
          }
}
```

Class DT4500InPut

```
package dt4500;

import dt4500.DT4500Communicator;

public class DT4500_InPut extends DT4500Communicator {


     static String CH0 = "Ch0";
     static String CH1 = "Ch1";
     static String CH2 = "Ch2";
     static double SAMPLE_RATE_NUM_OFDM = 4.000000;
     static double SAMPLE_RATE_DIV_OFDM = 1.000000;
     static double SAMPLE_RATE_NUM_TDM_1 = 5.000000;
     static double SAMPLE_RATE_DIV_TDM_1 = 1;
     static double SAMPLE_RATE_NUM_TDM_2 = 4.000000;
     static double SAMPLE_RATE_DIV_TDM_2 = 1;


     public DT4500_InPut(String host, int port) {
          super(host, port);
     }

     public enum Source {
          OFF, FILE, SINUS, DATA_FROM_CH0, CONFIG_FROM_CH0
     }

     private enum ChanelCoding {
          NONE, IPL_OFDM, OPL_IPL_OFDM, IPL_TDM, OPL_IPL_TDM
     }

     public int allSource(Source source) {
          int s = 0;
          switch (source) {
          case OFF:
               s = 0;
               break;
          case FILE:
```

```
                s = 1;
                break;
            case SINUS:
                s = 20;
                break;
            case DATA_FROM_CH0:
                s = 30;
                break;
            case CONFIG_FROM_CH0:
                s = 31;
                break;
            }
            return s;
        }

        public int Ch_Coding(ChanelCoding cc) {
            int s = 0;
            switch (cc) {
            case NONE:
                s = 0;
                break;
            case IPL_OFDM:
            case IPL_TDM:
                s = 1;
                break;
            case OPL_IPL_OFDM:
            case OPL_IPL_TDM:
                s = 2;
                break;
            default:
                System.out.println("default");
            }
            return s;
        }

        boolean setSource(String address, String sourceIn) {
            Source source = Source.valueOf(sourceIn.toUpperCase());
            String addr = "Input." + address + ".Source";
            System.out.println("\nSet: " + addr);
            int s = allSource(source);
            String cmd = addr + "=" + s;
            // for (int i = 0; i < retrySend; i++) {
            String result = sendCommand(cmd);
            if (resultIsOK(result)) {
                if (getValue(addr).contains(String.valueOf(s))) {
                    return true;
                } else {
                    System.err.println("Invalid Source");
                    System.exit(1);
                }
            }
            // }
            return false;
        }

        public boolean setSourceFile(String address, String cc, String file)
{
            Source source = Source.valueOf("FILE");
            int s = allSource(source);
            String addr = "Input." + address + ".Source";
            System.out.println("\nSet: " + addr);
            String cmd = addr + "=" + s;
            // for (int i = 0; i < retrySend; i++) {
```

```
            String result = sendCommand(cmd);
            chanelCoding(address, cc, file);
            if (resultIsOK(result)) {
                if (getValue(addr).contains(String.valueOf(1))) {
                    return true;
                } else {
                    System.err.println("Invalid Source");
                    System.exit(1);
                }
            }
            // }
            return false;
        }

        public String chanelCoding(String address, String cc,String file) {
            ChanelCoding chc = ChanelCoding.valueOf(cc.toUpperCase());
            int s = Ch_Coding(chc);
            String addr = "Input." + address + ".ChannelCoding";
            String result = sendCommand(addr + "=" + s);
            if (resultIsOK(result)) {
                if (getValue(addr).contains(String.valueOf(s))) {
                } else {
                    System.err.println("Invalid Source");
                    System.exit(1);
                }
            }

            if (address == CH0) {
                switch (chc) {
                case NONE:
                    sendCommand("Input.Ch0.IqFile=" + file);
                    sendCommand("Input.Ch0.IqSampleRateNum=" +
SAMPLE_RATE_NUM_OFDM);
                    sendCommand("Input.Ch0.IqSampleRateDiv=" +
SAMPLE_RATE_DIV_OFDM);
                    break;
                case IPL_OFDM:
                case OPL_IPL_OFDM:
                    sendCommand("Input.Ch0.OplIplXmlFile=" + file);
                    break;
                }
            } else if (address == CH1) {
                switch (chc) {
                case NONE:
                    sendCommand("Input.Ch1.IqFile=" + file);
                    sendCommand("Input.Ch1.IqSampleRateNum="
                            + SAMPLE_RATE_NUM_TDM_1);
                    sendCommand("Input.Ch1.IqSampleRateDiv="
                            + SAMPLE_RATE_DIV_TDM_1);
                    break;
                case IPL_TDM:
                case OPL_IPL_TDM:
                    sendCommand("Input.Ch1.OplIplXmlFile=" + file);
                    break;
                }
            } else if (address == CH2) {
                switch (chc) {
                case NONE:
                    sendCommand("");
                    sendCommand("Input.Ch2.IqFile=" + file);
                    sendCommand("Input.Ch2.IqSampleRateNum="
                            + SAMPLE_RATE_NUM_TDM_2);
                    sendCommand("Input.Ch2.IqSampleRateDiv="
```

```
                    + SAMPLE_RATE_DIV_TDM_2);
                break;
            case IPL_TDM:
            case OPL_IPL_TDM:
                sendCommand("Input.Ch2.OplIplXmlFile=" + file);
                break;
            }
        }
        return cc;
}

public void setOFDMSourceFile(String cc, String file_ofdm ) {
        setSourceFile(CH0, cc, file_ofdm);
}

public void setOFDMSource(String source) {
        setSource(CH0, source);
}

public void setTDM1SourceFile(String cc, String file_tdm1) {
        setSourceFile(CH1, cc, file_tdm1);
}

public void setTDM1Source(String source) {
        setSource(CH1, source);
}

public void setTDM2SourceFile(String cc, String file_tdm2) {
        setSourceFile(CH2, cc, file_tdm2);
}

public void setTDM2Source(String source) {
        setSource(CH2, source);
}

public void configDT4500InputOFDM() {
        String source = config.getOFDMConfig()[0];
        String cc = config.getOFDMConfig()[1];
        if (source.equalsIgnoreCase("file")) {
            String file_ofdm = config.getOFDMConfig()[2];
            setSourceFile(CH0, cc, file_ofdm);
        } else {
            setSource(CH0, source);
        }
}

public void configDT4500InputTDM1() {
        String source = config.getTDM1Config()[0];
        String cc = config.getTDM1Config()[1];
        if (source.equalsIgnoreCase("file")) {
            String file_tdm1 = config.getTDM1Config()[2];
            setSourceFile(CH1, cc, file_tdm1);
        } else {
            setSource(CH1, source);
        }
}

public void configDT4500InputTDM2() {
        String source = config.getTDM2Config()[0];
        String cc = config.getTDM2Config()[1];
        if (source.equalsIgnoreCase("file")) {
            String file_tdm2 = config.getTDM2Config()[2];
            setSourceFile(CH2, cc, file_tdm2);
```

```
          } else {
               setSource(CH2, source);
          }
     }

     public void print(){
          System.out.println(config.getOFDMConfig()[1]);
     }

     public static void main(String[] args) throws Exception {
          DT4500_InPut dt = new DT4500_InPut("192.168.254.13", 80);
          // dt.setCh0SourceFile("OPL_IPL_OFDM");
          // dt.setSource("Ch0","off");
dt.print();
     }
}
```

Class DT4500ChannelSimulator

```
package dt4500;

public class DT4500_ChannelSimulator extends DT4500_InPut {

   private static String CHAN_SIM_FILE = "all.chansim";
   private static int CHAN_SIM_FREQ = 2187;

    public DT4500_ChannelSimulator(String host, int port) {
          super(host, port);
    }

    public void loadChanSimFile(boolean onoff)throws Exception{
     if (onoff == true){
     enableDisable("ChanSim.Ch0.Enable", onoff);
     sendCommand("ChanSim.Ch0.File="+CHAN_SIM_FILE);
     sendCommand("ChanSim.Ch0.Load=1");
     sendCommand("ChanSim.Ch0.Freq="+CHAN_SIM_FREQ+"MHz");
     } else if (onoff == false){
          enableDisable("ChanSim.Ch0.Enable", onoff);
     }
    }

     public static void main(String[] args) throws Exception{
          DT4500_ChannelSimulator dt = new
DT4500_ChannelSimulator("dt4k071.guests.iis.fhg.de", 80);
          dt.loadChanSimFile(false);
          dt.setFunctionStatus(false);
     }

}
```

Class DT4500OutPut

```
package dt4500;

public class DT4500_OutPut extends DT4500_ChannelSimulator {

     public DT4500_OutPut(String host, int port) {
          super(host, port);
     }

     // ----------------------------------------------------------------

     public String dac_OutputSel(int dac_number, int value) {
          // String os = set_OutputSel(value);
          String address = "Output.Dac" + dac_number + ".OutputSel";
          String cmd = address + "=" + value;
          sendCommand(cmd);
          System.out.println("\nControl Output Selection:");
          // control(address,String.valueOf(value), " ");
          return getValue(address);
     }

     public String dac_Bias(int dac_number, long value) {
          String address = "Output.Dac" + dac_number + ".Bias";
          String cmd = address + "=" + value;
          sendCommand(cmd);
          System.out.println("\nControl Bias:");
          // control(address,String.valueOf(value)," ");
          return getValue(address);
     }

     public String ch_Delay(int ch_number, int value) {
          System.out.println("\nSet: Output.Ch" + ch_number + ".Delay");
          String address = "Output.Ch" + ch_number + ".Delay";
          String cmd = address + "=" + value;
          sendCommand(cmd);
          System.out.println("\nControl Delay:");
          String str = (address + " [usec]= " + getValue(address));
          control(address, value, str);

          return getValue(address);
     }

     public String dac_Noise(int dac_number, boolean onoff) throws
Exception {
          System.out.println("\nSet: Output.Dac" + dac_number +
".Noise.Enable");
          String address = "Output.Dac" + dac_number + ".Noise.Enable";
          enableDisable(address, onoff);
          return address;
     }

     public void dac_Noise_Bw(int dac_number, double bandwith) throws
Exception {
          String addr = "Output.Dac" + dac_number + ".Noise.Enable";
          System.out.println("\nSet: Output.Dac" + dac_number
                    + ".Noise.Bandwidth");
          if ((getValue(addr).contains("1"))) {
               String address_1 = "Output.Dac" + dac_number +
".Noise.Bandwidth";
               String cmd_1 = address_1 + "=" + bandwith;
```

```
                sendCommand(cmd_1);
                String str = address_1 + " [kHz]:" + (getValue(address_1));
                System.out.println("\nControl Noise Bandwidth:");
                control(address_1, bandwith, str);
            } else {
                System.err.println("Output.Dac" + dac_number
                        + ".Noise isn´t Enable");
            }
        }

        public void dac_Noise_Level(int dac_number, double level) throws
Exception {
            String addr = "Output.Dac" + dac_number + ".Noise.Enable";
            System.out.println("\nSet: Output.Dac" + dac_number +
".Noise.Level");
            if ((getValue(addr).contains("1"))) {
                String address_2 = "Output.Dac" + dac_number +
".Noise.Level";
                String cmd_2 = address_2 + "=" + level;
                sendCommand(cmd_2);
                String str = address_2 + " [dBm]: " +
(getValue(address_2));
                System.out.println("\nControl Noise Level:");
                control(address_2, level, str);
            } else {
                System.err.println("Output.Dac" + dac_number
                        + ".Noise isn´t Enable");
            }
        }

        public String dac_Ch(int dac_number, int ch_number, boolean onoff)
                throws Exception {
            String address = "Output.Dac" + dac_number + ".Ch" + ch_number
                    + ".Enable";
            System.out.println("\nSet: " + address + "");
            enableDisable(address, onoff);
            return getVariable(address);
        }

        public void dac_Ch_Freq(int dac_number, int ch_number, double freq)
                throws InterruptedException {
            String adr = "Output.Dac" + dac_number + ".Ch" + ch_number +
".Enable";
            System.out.println("\nSet: Output.Dac" + dac_number + ".Ch" +
ch_number
                    + " Frequency");
            if ((getValue(adr).contains("1"))) {
                String address = "Output.Dac" + dac_number + ".Ch" +
ch_number
                        + ".Freq";
                String cmd = address + "=" + freq;
                sendCommand(cmd);
                String str = address + " [MHz]= " + (getValue(address));
                System.out.println("\nControl Frequency:");
                control(address, freq, str);

            } else {
                System.err.println("Output.Dac" + dac_number + ".Ch" +
ch_number
                        + " isn´t Enable");
                System.exit(1);
            }
        }
```

```
     public void dac_Ch_Level(int dac_number, int ch_number, double
level) {
          String adr = "Output.Dac" + dac_number + ".Ch" + ch_number +
".Enable";
          System.out.println("\nSet: Output.Dac" + dac_number + ".Ch" +
ch_number
                    + " Level");
          if ((getValue(adr).contains("1"))) {
               String address = "Output.Dac" + dac_number + ".Ch" +
ch_number
                         + ".Level";
               String cmd = address + "=" + level;
               sendCommand(cmd);
               String str = address + " [dBm]= " + (getValue(address));
               System.out.println("\nControl Level:");
               control(address, level, str);
          } else {
               System.err.println("Output.Dac" + dac_number + ".Ch" +
ch_number
                         + " isn´t Enable");
               System.exit(1);
          }
     }

     public double dac_Ch_RMS(int dac_number, int ch_number) {
          String adr = "Output.Dac" + dac_number + ".Ch" + ch_number +
".Enable";
          if ((getValue(adr).contains("1"))) {
               String cmd = "Output.Dac" + dac_number + ".Ch" + ch_number
+ ".Rms";
               double rms = Double.parseDouble(getValue(cmd));
               System.out.println(cmd + "= " + rms + "[dBm]");
               return rms;
          } else {
               System.err.println("Output.Dac" + dac_number + ".Ch" +
ch_number
                         + " isn´t Enable");
               System.exit(1);
          }
          return 0;
     }

     // ----------------------------------------------------------------

     public void configDac0Ch0() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac0Ch0Config()[0]);
          dac_Ch(0, 0, onoff);
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac0Ch0Config()[1]);
               dac_Ch_Level(0, 0, level);
               Double freq =
Double.parseDouble(config.getDac0Ch0Config()[2]);
               dac_Ch_Freq(0, 0, freq);
          }
     }

     public void configDac0Ch1() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac0Ch1Config()[0]);
          dac_Ch(0, 1, onoff);
```

```
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac0Ch1Config()[1]);
               dac_Ch_Level(0, 1, level);
               Double freq =
Double.parseDouble(config.getDac0Ch1Config()[2]);
               dac_Ch_Freq(0, 1, freq);
          }
     }

     public void configDac0Ch2() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac0Ch2Config()[0]);
          dac_Ch(0, 2, onoff);
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac0Ch2Config()[1]);
               dac_Ch_Level(0, 2, level);
               Double freq =
Double.parseDouble(config.getDac0Ch2Config()[2]);
               dac_Ch_Freq(0, 2, freq);
          }
     }

     public void configDac0Noise() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac0NoiseConfig()[0]);
          dac_Noise(0, onoff);
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac0NoiseConfig()[1]);
               dac_Noise_Level(0, level);
               Double bw =
Double.parseDouble(config.getDac0NoiseConfig()[2]);
               dac_Noise_Bw(0, bw);
          }

     }

     public void configDac1Ch0() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac1Ch0Config()[0]);
          dac_Ch(1, 0, onoff);
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac1Ch0Config()[1]);
               dac_Ch_Level(1, 0, level);
               Double freq =
Double.parseDouble(config.getDac1Ch0Config()[2]);
               dac_Ch_Freq(1, 0, freq);
          }
     }

     public void configDac1Ch1() throws Exception {
          Boolean onoff =
Boolean.parseBoolean(config.getDac1Ch1Config()[0]);
          dac_Ch(1, 1, onoff);
          if (onoff == true) {
               Double level =
Double.parseDouble(config.getDac1Ch1Config()[1]);
               dac_Ch_Level(1, 1, level);
               Double freq =
Double.parseDouble(config.getDac1Ch1Config()[2]);
```

```
                dac_Ch_Freq(1, 1, freq);
            }
        }

        public void configDac1Ch2() throws Exception {
            Boolean onoff =
Boolean.parseBoolean(config.getDac1Ch2Config()[0]);
            dac_Ch(1, 2, onoff);
            if (onoff == true) {
                Double level =
Double.parseDouble(config.getDac1Ch2Config()[1]);
                dac_Ch_Level(1, 2, level);
                Double freq =
Double.parseDouble(config.getDac1Ch2Config()[2]);
                dac_Ch_Freq(1, 2, freq);
            }
        }

        public void configDac1Noise() throws Exception {
            Boolean onoff =
Boolean.parseBoolean(config.getDac1NoiseConfig()[0]);
            dac_Noise(1, onoff);
            if (onoff == true) {
                Double bw =
Double.parseDouble(config.getDac1NoiseConfig()[2]);
                dac_Noise_Bw(1, bw);
                Double level =
Double.parseDouble(config.getDac1NoiseConfig()[1]);
                dac_Noise_Level(1, level);

            }

        }

        public void setDac0Ch1(boolean onoff) throws Exception {
            dac_Ch(0, 1, onoff);
        }

        public void setDac0Ch2(boolean onoff) throws Exception {
            dac_Ch(0, 2, onoff);
        }

        public void setDac1Ch0(boolean onoff) throws Exception {
            dac_Ch(1, 0, onoff);
        }

        public void setDac1Ch1(boolean onoff) throws Exception {
            dac_Ch(1, 1, onoff);
        }

        public void setDac1Ch2(boolean onoff) throws Exception {
            dac_Ch(1, 2, onoff);
        }

        public void setDac0Ch0Level(double level) throws Exception {
            dac_Ch_Level(0, 0, level);
        }

        public void setDac0Ch1Freq(double freq) throws Exception {
            dac_Ch_Freq(0, 1, freq);
        }

        public void setDac0Ch1Level(double level) throws Exception {
```

```
        dac_Ch_Level(0, 1, level);
    }

    public void setDac0Ch2Freq(double freq) throws Exception {
        dac_Ch_Freq(0, 2, freq);
    }

    public void setDac0Ch2Level(double level) throws Exception {
        dac_Ch_Level(0, 2, level);
    }

    public void setDac1Ch0Freq(double freq) throws Exception {
        dac_Ch_Freq(1, 0, freq);
    }

    public void setDac1Ch0Level(double level) throws Exception {
        dac_Ch_Level(1, 0, level);
    }

    public void setDac1Ch1Freq(double freq) throws Exception {
        dac_Ch_Freq(1, 1, freq);
    }

    public void setDac1Ch1Level(double level) throws Exception {
        dac_Ch_Level(1, 1, level);
    }

    public void setDac1Ch2Freq(double freq) throws Exception {
        dac_Ch_Freq(1, 2, freq);
    }

    public void setDac1Ch2Level(double level) throws Exception {
        dac_Ch_Level(1, 2, level);
    }

    public void setDac0Noise(boolean onoff) throws Exception {
        dac_Noise(0, onoff);
    }

    public void setDac1Noise(boolean onoff) throws Exception {
        dac_Noise(1, onoff);
    }

    public void setDac0NoiseBw(double bw) throws Exception {
        dac_Noise_Bw(0, bw);
    }

    public void setDac0NoiseLevel(double level) throws Exception {
        dac_Noise_Level(0, level);
    }

    public void setDac1NoiseBw(double bw) throws Exception {
        dac_Noise_Bw(1, bw);
    }

    public void setDac1NoiseLevel(double level) throws Exception {
        dac_Noise_Level(1, level);
    }

    public void setCh0Delay(int value) throws Exception {
        ch_Delay(0, value);
    }
```

```
    public void setCh1Delay(int value) throws Exception {
        ch_Delay(1, value);
    }

    public void setCh2Delay(int value) throws Exception {
        ch_Delay(2, value);
    }

    public void setRefClock(boolean onoff) throws Exception {
        String address = "Output.EnableRefClock";
        enableDisable(address, onoff);
    }

    public void setDac0Bias(long value) {
        dac_Bias(0, value);
    }

    public void setDac1Bias(long value) {
        dac_Bias(1, value);
    }

    public void getDac0Ch0RMS() {
        dac_Ch_RMS(0, 0);
    }

    public void getDac0Ch1RMS() {
        dac_Ch_RMS(0, 1);
    }

    public void getDac0Ch2RMS() {
        dac_Ch_RMS(0, 2);
    }

    public void getDac1Ch0RMS() {
        dac_Ch_RMS(1, 0);
    }

    public void getDac1Ch1RMS() {
        dac_Ch_RMS(1, 1);
    }

    public void getDac1Ch2RMS() {
        dac_Ch_RMS(1, 2);
    }

    public static void main(String[] args) throws Exception {

        DT4500_OutPut dt = new DT4500_OutPut("192.168.254.13", 80);
        dt.setDac1Ch1Level(-35);
        dt.setDac1NoiseLevel(-73);
    }
}
```

ReadDT4500ConfigFile

```
package dt4500;

import java.io.IOException;

import org.jdom.Document;
import org.jdom.Element;
import org.jdom.JDOMException;
import org.jdom.input.SAXBuilder;

public class ReadDT4500ConfigFile {

     private Element rootElement;
     private Element DT4500InputConfig;
     private Element DT4500OutputConfig;

     public ReadDT4500ConfigFile(String filename) throws JDOMException,
               IOException {
          Document doc = new SAXBuilder().build(filename);
          rootElement = doc.getRootElement();
          DT4500InputConfig =
rootElement.getChild("DT4500_InPut_Config");
          DT4500OutputConfig =
rootElement.getChild("DT4500_OutPut_Config");
     }

     public String[] getOFDMConfig() {
          Element ofdm = DT4500InputConfig.getChild("OFDM");
          String source = ofdm.getChildTextTrim("Source");
          String channelCoding = ofdm.getChildTextTrim("ChannelCoding");
          String sourceFile = ofdm.getChildTextTrim("SourceFile");
          return new String[] { source, channelCoding, sourceFile };
     }

     public String[] getTDM1Config() {
          Element tdm1 = DT4500InputConfig.getChild("TDM1");
          String source = tdm1.getChildTextTrim("Source");
          String channelCoding = tdm1.getChildTextTrim("ChannelCoding");
          String sourceFile = tdm1.getChildTextTrim("SourceFile");
          return new String[] { source, channelCoding, sourceFile };
     }

     public String[] getTDM2Config() {
          Element tdm2 = DT4500InputConfig.getChild("TDM2");
          String source = tdm2.getChildTextTrim("Source");
          String channelCoding = tdm2.getChildTextTrim("ChannelCoding");
          String sourceFile = tdm2.getChildTextTrim("SourceFile");
          return new String[] { source, channelCoding, sourceFile };
     }

     public String[] getDac0Ch0Config() {
          Element dac0 = DT4500OutputConfig.getChild("DAC0");
          Element dac0ch0 = dac0.getChild("CH0");
          String level = dac0ch0.getChildTextTrim("Level");
          String freq = dac0ch0.getChildTextTrim("Frequenz");
          String enable =dac0ch0.getChildTextTrim("Enable");
          return new String[] { enable,level, freq };
     }

     public String[] getDac0Ch1Config() {
```

```
        Element dac0 = DT4500OutputConfig.getChild("DAC0");
        Element dac0ch1 = dac0.getChild("CH1");
        String level = dac0ch1.getChildTextTrim("Level");
        String freq = dac0ch1.getChildTextTrim("Frequenz");
        String enable =dac0ch1.getChildTextTrim("Enable");
        return new String[] { enable, level, freq };
    }

    public String[] getDac0Ch2Config() {
        Element dac0 = DT4500OutputConfig.getChild("DAC0");
        Element dac0ch2 = dac0.getChild("CH2");
        String level = dac0ch2.getChildTextTrim("Level");
        String freq = dac0ch2.getChildTextTrim("Frequenz");
        String enable =dac0ch2.getChildTextTrim("Enable");
        return new String[] { enable, level, freq };
    }

    public String[] getDac0NoiseConfig(){
        Element dac0 = DT4500OutputConfig.getChild("DAC0");
        Element noise = dac0.getChild("Noise");
        String level = noise.getChildTextTrim("Level");
        String bandwidth = noise.getChildTextTrim("Bandwidth");
        String enable =noise.getChildTextTrim("Enable");
        return new String[] { enable, level, bandwidth };
    }

    public String[] getDac1Ch0Config() {
        Element dac1 = DT4500OutputConfig.getChild("DAC1");
        Element dac1ch0 = dac1.getChild("CH0");
        String level = dac1ch0.getChildTextTrim("Level");
        String freq = dac1ch0.getChildTextTrim("Frequenz");
        String enable =dac1ch0.getChildTextTrim("Enable");
        return new String[] { enable, level, freq };
    }

    public String[] getDac1Ch1Config() {
        Element dac1 = DT4500OutputConfig.getChild("DAC1");
        Element dac1ch1 = dac1.getChild("CH1");
        String level = dac1ch1.getChildTextTrim("Level");
        String freq = dac1ch1.getChildTextTrim("Frequenz");
        String enable =dac1ch1.getChildTextTrim("Enable");
        return new String[] { enable, level, freq };
    }

    public String[] getDac1Ch2Config() {
        Element dac1 = DT4500OutputConfig.getChild("DAC1");
        Element dac1ch2 = dac1.getChild("CH2");
        String level = dac1ch2.getChildTextTrim("Level");
        String freq = dac1ch2.getChildTextTrim("Frequenz");
        String enable =dac1ch2.getChildTextTrim("Enable");
        return new String[] { enable, level, freq };
    }

    public String[] getDac1NoiseConfig(){
        Element dac1 = DT4500OutputConfig.getChild("DAC1");
        Element noise = dac1.getChild("Noise");
        String level = noise.getChildTextTrim("Level");
        String bandwidth = noise.getChildTextTrim("Bandwidth");
        String enable =noise.getChildTextTrim("Enable");
        return new String[] { enable, level, bandwidth };
    }
```

```
     public static void main(String[] args) throws JDOMException,
IOException {
          ReadDT4500ConfigFile config = new
ReadDT4500ConfigFile("DT4500ConfigFile");
          String dgdg = config.getTDM1Config()[0];
          System.out.println(dgdg);
     }
}
```

Anhang D.3: Class ErrorToSNR

```
package control;

import java.io.IOException;

import dt4500.DT4500_OutPut;
import espi.ESPICommunicator;
import emad.EMServerApp;

public class ErrorToSNR {

     FileWriter f;
     final static String configFile = new String("TestFile.xml");

     ESPICommunicator espi;
     DT4500_OutPut dt4500;
     private EMServerApp serverApp;
     private Thread runEMServer;

     boolean emStarted = true;
     boolean valid = true;

     long final_error;
     long drop_out;
     long all_packet;

     // dt4k071.guests.iis.fhg.de
     final static String DT4500_hostname = new String("192.168.254.13");
     final static String ESPI_hostname = new String("192.168.255.218");
     final static String EM_hostname = new String("em0005.guests");

     final static int DT4500_port = 80;
     final static int ESPI_port = 5025;
     final static int dataPort = 55003;

     // Measure - Parameters
     int AMPLITUDE_DIFFERENZ = 15;
     int time_a = 60;
     int time_b = 300;
     double startLevel = -42.0;
     double stopLevel = -10;

     // ESPI:
     String CENTER_FREQ = "2177.5 MHz";
     String SPAN = "20MHz";
     String SWEEP_TIME = "500 ms";
     int RFATTEN = 10;
     String PREDEFINED_STANDARD = "None";
     int NO_OF_ADJ_CHAN = 1;
     String CHANNEL_BANDWIDTH = "3.61 MHz";
     String ADJ_CHANNEL_BANDWIDTH = "3.61 MHz";
     String ADJ_CHANNEL_SPACING = "5 MHz";
     String CP_ACP_ABS_REL = "ABS";
     String CHAN_PWR_HZ = "Off";
     String RBW = "300kHz";
     String VBW = "1kHz";
     String NOISE_CORR = "off";
```

```
public ErrorToSNR(String file_name) {
     try {
          dt4500 = new DT4500_OutPut(DT4500_hostname, DT4500_port);
          f = new FileWriter(file_name);
          espi = new ESPICommunicator(ESPI_hostname, ESPI_port);
          serverApp = new EMServerApp(EM_hostname, dataPort);
          runEMServer = new Thread(serverApp);
          runEMServer.start();
          runMeasure(time_a, time_b);
     } catch (Exception e1) {
          e1.printStackTrace();
     }
}

double rintScale2(double value) {
     return Math.rint(value * 100) / 100;
}

double rintScale4(double value) {
     return Math.rint(value * 100 * 100) / 100 / 100;
}

public void runDt4500() {
     try {
          System.out.println("\nDT4500: \n");
          dt4500.reset();
          System.out.println("\nConfiguration \n");
          dt4500.configDT4500InputOFDM();
          dt4500.configDac1Ch1();
          dt4500.configDac1Noise();
          dt4500.setFunctionStatus(true);
          System.out.println("Configuration End \n");

     } catch (Exception e) {
          e.printStackTrace();
     }
}

public void runEspi() throws Exception {
     try {
          System.out.println("\nESPI:\n");
          espi.reset();
          System.out.println("\nConfiguration \n");
          espi.setRFAtten(RFATTEN);
          espi.setSpan(SPAN);
          espi.setCenterFreq(CENTER_FREQ);
          espi.setSweepTime(SWEEP_TIME);
          espi.setRBW(RBW);
          espi.setVBW(VBW);
          System.out.println(" ");
          espi.numberOfAdjChan(NO_OF_ADJ_CHAN);
          espi.chanBandwidth(CHANNEL_BANDWIDTH);
          espi.adjChanBandwidth(ADJ_CHANNEL_BANDWIDTH);
          espi.adjChanSpacing(ADJ_CHANNEL_SPACING);
          espi.cpAcpAbsRel(CP_ACP_ABS_REL);
          espi.chanPwrHz(CHAN_PWR_HZ);
          System.out.println(" ");
          System.out.println("Configuration End\n");

     } catch (Exception e) {
          e.printStackTrace();
     }
}
```

```
    public double runEmServer(long time) {
        long pkg = 0;
        long p = 0;
        long err = 0;
        double error_rate = 0;
        long s_time = System.currentTimeMillis() / 1000;

        System.out.println("\nSearch Error Rate:");
        System.out.println("");

        // Start EMServer
        try {
            System.out.println("Resetting Counters");
            System.out.println("Measure Final Error Rate:");
            serverApp.resetCounter();

            System.out.println("Bitte warten Sie 20 sec. lang");

            while (System.currentTimeMillis() / 1000 - s_time < 20) {
                Thread.sleep(1000);
                pkg = serverApp.packetCounter();
                err = serverApp.packetErrorCounter();
            }
            serverApp.resetCounter();
            s_time = System.currentTimeMillis() / 1000;
            System.out.println("Bitte warten Sie " + time + " sec.
lang");
            while (System.currentTimeMillis() / 1000 - s_time < time) {
                Thread.sleep(1000);
                pkg = serverApp.packetCounter();
                if (p != pkg) {
                    p = pkg;
                    err = serverApp.packetErrorCounter();
                    error_rate = (double) err / (double) pkg * 100;
                    valid = true;

                } else {
                    drop_out++;
                    err = serverApp.packetErrorCounter();
                    error_rate = (double) err / (double) pkg * 100;
                    valid = true;
                }
                if (serverApp.isValid() != true) {
                    System.err.println("Receiver Error 2");
                    // valid = false;
                    serverApp.resetCounter();
                    break;
                }
            }
            System.out.println("\n\nFinal Error Rate: " + error_rate +
"\n\n");
            final_error = err;
            all_packet = pkg;
            return error_rate;

        } catch (Exception e) {
            e.printStackTrace();
        }
        return 0;
    }

    public long dropOut() {
```

```
            return drop_out;
      }

      public long allPacket() {
            return all_packet;
      }

      public long finalError() {
            return final_error;
      }

      public boolean valid() {
            return valid;
      }

      public void setLevel(double startLevel) throws Exception {
            double ref_level = 0;
            ref_level = startLevel - AMPLITUDE_DIFFERENZ;
            espi.setRefLevel(ref_level);
            dt4500.setDac1Ch1Level(startLevel);
            System.out.println(" ");
            Thread.sleep(3000L);
      }

      public String measureSNR() throws IOException, InterruptedException
{
            double final_level = 0;
            double final_lower = 0;
            double final_upper = 0;
            int count = 0;
            int i = 0;
            Thread.sleep(1500L);
            while (i != 20) {
                  i++;
                  String response = espi.getSNR();
                  String str = (" ");
                  int index_1 = response.indexOf(str) + str.length();
                  String koma = (",");

                  int index_2 = response.indexOf(koma);
                  int index_3 = response.lastIndexOf(koma);
                  int index_4 = response.length();
                  double level =
rintScale4(Double.parseDouble(response.substring(
                              index_1, index_2)));
                  double lower =
rintScale4(Double.parseDouble(response.substring(
                              index_2 + 1, index_3 - 1)));
                  double upper =
rintScale4(Double.parseDouble(response.substring(
                              index_3 + 1, index_4)));
                  count++;

                  final_level = (final_level + level);
                  final_lower = (final_lower + lower);
                  final_upper = (final_upper + upper);
            }
            espi.stopSNR();
            final_level = rintScale2(final_level / count);
            final_lower = rintScale2(final_lower / count);
            final_upper = rintScale2(final_upper / count);
            String string = (String.valueOf(final_level) + ","
                        + String.valueOf(final_lower) + "," + String
```

```
                    .valueOf(final_upper));
          System.out.println(string);
          return string;
     }

     public double measureErrorRate(long time) throws Exception {
          valid = false;
          double error_rate = 0;
          while (valid == false) {

               error_rate = runEmServer(time);
               valid = valid();

               if (valid == false) {

                    System.out.println("Signal auschalten");
                    dt4500.setDac1Ch0(false);
                    Thread.sleep(10000L);

                    dt4500.setDac1Ch0(true);
                    System.out.println("\n\n"
                              + dt4500.getValue("Output.Dac1.Ch1.Level") +
"\n\n");
                    Thread.sleep(500);
               }
          }
          return error_rate;
     }

     public void runMeasure(long time_1, long time_2) throws Exception {
          long drop_out = 0;
          long packet = 0;
          long error = 0;
          double final_error_rate = 0;

          System.out.println(startLevel + "," + stopLevel);

          // Configure dt4k and ESPI - read run... Functions
//        runDt4500();
          Thread.sleep(500L);
          runEspi();
          Thread.sleep(500L);

          // Measure
          long system_time = System.currentTimeMillis() / 1000;
          double counter;
          double step = 1;

          for (counter = startLevel; counter <= stopLevel; counter =
counter
                    + step) {

               System.out.println("\n Measure:\n");

               // set Reference Level for ESPI and DT4K
               setLevel(startLevel);

               // Measure Reciver Error_1
               final_error_rate = measureErrorRate(time_1);
               drop_out = dropOut();
               packet = allPacket();
               error = finalError();
               f.write(packet + "," + error + "," + drop_out + ","
```

```
                    + final_error_rate);

            // Measure with Step 0.1

            if (final_error_rate < 99.8) {
                startLevel = rintScale2(startLevel - (1));
                double new_step = 0.1;

                System.out.println("\n Measure with new startLevel:");
                System.out.println("New startLevel= " + startLevel +
"dBm");
                System.out.println("New step= " + new_step);
                final_error_rate = 0.1;
                while (final_error_rate != 0.0) {

                    System.out.println("\n Measure:\n");

                    // set Reference Level for ESPI and DT4K
                    setLevel(startLevel);
                    Thread.sleep(500L);

                    // Measure SNR
                    String resp = measureSNR();

                    // Measure Reciver Error_2
                    final_error_rate = measureErrorRate(time_2);
                    drop_out = dropOut();
                    packet = allPacket();
                    error = finalError();
                    f.write(startLevel + "," + resp + "," + packet +
","
                            + error + "," + final_error_rate);

                    startLevel = rintScale2(startLevel + new_step);
                }
                System.out.println("Ready");
                System.exit(1);
            }

            startLevel = rintScale2(startLevel + step);
        }
        f.close();
        System.out.println("End");
        System.exit(1);
    }

    public static void main(String[] args) throws Exception {
        ErrorToSNR measure = new ErrorToSNR("TDM_18PSK_R_1_3.txt");
    }
}
```

L