

# DS-Studienarbeit

SS-2000-01

# QAM - Modem

## DOKUMENTATION

Betreuer:	Andreas Ehrensperger (ASCOM System AG)
Studenten:	Andy Zehnder E97c
	Armin Hüsler E97c
Abgabedatum:	14. Juli 2000

# 1 Inhaltsverzeichnis

<b>1</b>	<b>Inhaltsverzeichnis</b>	<b>2</b>
<b>2</b>	<b>Einführung</b>	<b>4</b>
<b>3</b>	<b>Aufgabenstellung</b>	<b>5</b>
<b>4</b>	<b>Zeitplan</b>	<b>7</b>
<b>5</b>	<b>Pflichtenheft</b>	<b>8</b>
<b>6</b>	<b>Allgemeines zur QAM</b>	<b>9</b>
6.1	Grundprinzip	9
6.2	Sender	10
6.3	Empfänger	13
<b>7</b>	<b>Matlab/Simulink Simulationen</b>	<b>14</b>
7.1	Entwicklung des Senders	14
7.2	Übergang zum Empfänger	20
7.3	Sender und Empfänger	24
7.3.1	Grundlegendes Modell	24
7.3.2	Automatic-Gain-Control (AGC)	26
7.3.3	Scrambler (Verwürfler)	31
7.4	Bitfehlerhäufigkeit in der Übertragung	35
7.4.1	Bitfehlerhäufigkeit ohne Scrambler	39
7.4.2	Bitfehlerhäufigkeit mit Scrambler	41
<b>8</b>	<b>Realisierung</b>	<b>43</b>
8.1	Aufbau	43
8.2	Entwicklungsumgebung	44
8.2.1	Hardware	44
8.2.2	Software	44
8.3	Definitionen und Protokoll	45
8.3.1	Taktraten	45
8.3.2	Synchronisation	45
8.3.3	Datenübertragung	47
8.3.4	Softwareschnittstellen	48
8.4	Sender	50
8.4.1	Datenorganisation	50
8.4.2	Synchronisation	51
8.4.3	Datenverarbeitung	52
8.5	Empfänger	58
8.5.1	Datenorganisation	58
8.5.2	Synchronisation	59
8.5.3	Datenverarbeitung	61
8.6	Kanal	68
8.6.1	Kanalfilter	68
8.7	Testprogramm	68
8.7.1	Hardwareanordnung	68
8.7.2	Programmiersprachen	69
8.7.3	Sendeanwendung	69
8.7.4	Empfängeranwendung	71
8.8	Messungen	72
8.8.1	KO-Messungen	72
8.8.2	Fehlermessung mit Testprogramm	73
8.9	Software	74
8.9.1	Verzeichnis- und File-Struktur	74

8.9.2	Kompilation und Ausführung der Programme .....	75
<b>9</b>	<b>Zusammenfassung .....</b>	<b>77</b>
9.1	Was realisiert wurde .....	77
9.2	Was nicht realisiert wurde.....	77
9.3	Probleme .....	77
<b>10</b>	<b>Schlusswort/Fazit .....</b>	<b>78</b>
<b>11</b>	<b>Anhang.....</b>	<b>79</b>
11.1	Literaturverzeichnis.....	79
11.2	m-Files der Simulationsphase .....	79
11.3	Sourcecode .....	79

## 2 Einführung

Die heute meist eingesetzten Modems, die ihre Daten analog über das Telefonband übertragen, arbeiten nach dem Prinzip der Quadratur-Amplituden-Modulation (QAM). Diese Modulationstechnik zeichnet sich dadurch aus, dass mit ihr über einen bandbegrenzten Übertragungsweg mit relativ hoher Geschwindigkeit Daten mit einer tiefen Bitfehler-Wahrscheinlichkeit gesendet werden können. Im Fach Digitale Signalverarbeitung soll nun im Rahmen einer Studienarbeit ein QAM-Modem entworfen und realisiert werden, welches auf DSP-Technologie basiert. Dabei wird je ein Digitaler Signalprozessoren für den Sender sowie den Empfänger eingesetzt. Gleichzeitig soll das Modem auch in Matlab/Simulink entworfen und eingehend simuliert werden.

## 3 Aufgabenstellung



HOCHSCHULE  
RAPPERSWIL  
HSR

SS-2000-01

Schuljahr 2000

**Studienarbeit** für Herrn Armin Hüsler und Herrn Andreas Zehnder

### QAM-Modem

#### Einleitung

Quadratur-Amplituden-Modulation (QAM) ist eine Modulationstechnik, die sich durch hohe Bandbreiten-Effizienz bei gleichzeitig tiefer Bitfehler-Wahrscheinlichkeit auszeichnet [1]. QAM eignet sich daher gut für Anwendungen wo digitale Daten mit hoher Übertragungsgeschwindigkeit über einen Kanal mit relativ kleiner Bandbreite übertragen werden sollen. Den erwähnten vorteilhaften Eigenschaften von QAM steht jedoch ein hoher Realisierungsaufwand gegenüber, welcher praktisch nur mit digitaler Signalverarbeitung bewältigt werden kann. Im Rahmen dieser Studienarbeit im Fache DS und der daran anschliessenden Diplomarbeit soll ein auf DSP-Technologie basierendes QAM-Modem entworfen und realisiert werden.

#### Ziel der Arbeit

Es soll eine in Echtzeit funktionierende QAM-Übertragung zwischen zwei Signalprozessor-Boards realisiert werden. Die Wertigkeit der übertragenen Symbole soll mindestens 16 betragen. Die Übertragung soll im Telefonieband (300Hz .. 3.4kHz) erfolgen. Die Synchronisation bezüglich des Trägersignals und des Symboltakts kann behelfsmässig über zusätzliche Verbindungen gelöst werden (die Rückgewinnung von Trägersignal und Symboltakt aus dem Empfangssignal ist Gegenstand der anschliessenden Diplomarbeit).

#### Ziel der Arbeit

Das Ziel dieser Arbeit ist nicht unbedingt ein perfekt funktionierendes Gerät, vielmehr geht es darum, die nötigen Kenntnisse und Erfahrungen zu erarbeiten. Wichtig ist eine Charakterisierung des realisierten Modems aufgrund von Messungen und Simulationen. Z.B. sollen Aussagen betreffend der Übertragungssicherheit (Bitfehlerrate) in Abhängigkeit von verschiedenen Parametern, wie z.B. Datenrate, Baudrate, Symbolwertigkeit oder Signal- zu Geräuschverhältnis auf dem Kanal gemacht werden können. Im Weiteren interessiert auch die Auslastung der beiden Signalprozessoren in der Empfangs- und in der Sendestelle. Es ist wichtig, dass die gesamte Arbeit umfassend dokumentiert wird und sämtliche Resultate (positive wie auch negative) im Bericht diskutiert werden!

#### Vorgehen

1. Einarbeitung in die Problematik z.B. anhand der unten aufgeführten Literatur.
2. Auswahl einer geeigneten Realisierungs-Plattform (Signalprozessor, Board, Entwicklungsumgebung) in Zusammenarbeit mit dem mit dem Betreuer.
3. Festlegung der groben Spezifikationen.
4. Erarbeitung des Konzepts sowie Erstellung eines kurzen Pflichtenhefts, das alle wesentlichen Anforderungen an den zu realisierenden Prototyp festlegt.
5. Ausarbeitung der Algorithmen. Verifikation z.B. durch Simulation mit MATLAB und/oder SIMULINK.

Ds-Studienarbeit SS-2000-01

1



6. Implementierung der DSP-Software nach den Gesichtspunkten der strukturierten Programmierung (→ hierarchischer Aufbau aus Modulen mit klaren Schnittstellen, einfach erweiterbar).
7. Durchführung von eingehenden Tests und Messungen zur Überprüfung der korrekten Funktionsweise sowie zur Charakterisierung der implementierten Algorithmen.
8. Vergleich der Messergebnisse mit der Theorie.

### Literatur

- [1] Steven A. Tretter, Communication System Design using DSP Algorithms  
Plenum Press New York 1995, ISBN 0-306-45032-1
- [2] Erich Pehl, Digitale und analog Nachrichtenübertragung  
Hüthig-Verlag Heidelberg 1998, ISBN 3-7785-2469-0
- [3] Bernard Sklar, Digital Communication Systems,  
Prentice Hall Englewood Cliffs 1988, ISBN 0-13-212713-X
- [4] Kaiser, Mitra, Handbook for Digital Signal Processing,  
John Wiley & Sons 1993, ISBN 0-471-61995-7

### Bericht

Über die Arbeit ist ein Bericht zu verfassen. Dabei sind die Richtlinien der Abteilung für Elektrotechnik für das Erstellen von Berichten einzuhalten. Alle verwendeten Quellen sind im Literaturverzeichnis des Berichts anzugeben (Hinweise im Text). Der Bericht ist in doppelter Ausführung abzugeben; ein Exemplar verbleibt am Labor für digitale Signalverarbeitung der HSR, das Doppel erhalten die Verfasser nach der Korrektur und der Bewertung zurück. Die erstellten Programme und der Text des Berichtes sind auf Diskette(n) (mit Nummer der Studienarbeit) beizulegen.

### Termine, Bedingungen

Gemäss Vorgaben und Terminplan des Vorstandes der Abteilung für Elektrotechnik.

Ausgabe der Aufgabenstellung: 10. April 2000  
Abgabe des Berichts, Ende der Arbeit: 14. Juli 2000  
Präsentation der Arbeit: 12. oder 13. Juli 2000  
Arbeitsplatz: DS-Labor (zuständig: P. Roffler, Laborassistent)

**Assistenz:** A. Rüegg

**Betreuer:** A. Ehrensperger  
Ascom Systec AG  
Abt. DEU, H43407  
CH-8634 Hombrechtikon

Tel. : 055 / 254 6607  
Fax.: 055 / 254 6764  
E-Mail: Andreas.Ehrensperger@ascom.ch

Grüt, 8. April 2000

## 4 Zeitplan

Da wir zu Beginn dieser Arbeit über keinerlei Erfahrung und Kenntnisse über den zeitlichen Aufwand einer solchen Arbeit verfügten, fanden wir es relativ sinnlos einen Soll-Zeitplan zu erstellen. Dieser hätte sowieso nicht eingehalten werden können. Es ist hier nachfolgend der Ist-Zeitplan aufgeführt, d.h. so wie die Arbeit tatsächlich abgelaufen ist.

Art der Tätigkeit	Woche													
	15	16	17	18	19	20	21	22	23	24	25	26	27	28
Erarbeitung der Grundlagen und Theorie	■													
Einarbeitung in Matlab/Simulink		■												
Einarbeitung in SHARC-Programmierung		■												
Einarbeitung in JAVA-Programmierung			■											
Matlab/Simulink Simulationen		■												
DSP-Algorithmen			■				■				■			
Testprogramme					■					■				
Dokumentation											■			

Ausgeführt durch A. Hüsler  
Ausgeführt durch A. Zehnder



## 5 Pflichtenheft

Es soll ein Modem beruhend auf dem Prinzip der Quadratur-Amplituden-Modulation realisiert werden. Zum einen ist das Modem auf Matlab/Simulink theoretisch zu simulieren und anhand verschiedener Parameter zu charakterisieren. Zum anderen ist zwischen zwei SHARC-DSP's eine unidirektionale, in echtzeit funktionierende QAM-Verbindung zu realisieren. Die Übertragung soll im Telefonieband (300Hz – 3.4kHz) erfolgen.

An das Modem und die Übertragung werden folgende Ansprüche gestellt:

- Sender und Empfänger funktionieren nach dem Prinzip der QAM
- Die Verbindung ist unidirektional
- Die Wertigkeit der zu übertragenden Symbole beträgt 16
- Die Bitrate beträgt 9600bps, die Baudrate 2400baud und die Trägerfrequenz 1800 Hz
- Um den Empfänger mit dem Sender zu synchronisieren, ist eine zur Datenleitung zusätzliche Verbindung zwischen den beiden Signalprozessoren zu verwenden
- Das Synchronisierungs- sowie das Datenübermittlungs-Protokoll ist frei wählbar.
- Sender und Empfänger sind je über ein graphisches-User-Interface zu steuern, auf welchen Auskunft über die gesendeten und empfangenen Daten gegeben wird
- Die empfangenen Daten werden dem Sender über eine Netzwerkverbindung rückgeführt, um die Korrektheit der empfangenen Daten zu überprüfen

## 6 Allgemeines zur QAM

Quadraturamplitudenmodulation ist eine weit verbreitete Methode um digitale Signale über bandpassbegrenzte Kanäle zu übertragen.

Da zum Bsp. Telefonleitungen nur ein eng begrenztes Frequenzband übertragen, können die Bits nicht direkt auf die Leitung gelegt werden. Daher wird ein permanentes Sinussignal übertragen, dessen Amplitude und Phase durch Modulation ständig verändert wird. Eine Amplituden-Phasen-Kombination wird Symbol genannt. Die Menge aller Symbole nennt man Konstellation. Sie kann als Punktmenge in der Ebene der komplexen Zahlen dargestellt werden. Enthält eine Konstellation 16 Symbole, so werden pro Schritt 4 Bit übertragen.

### 6.1 Grundprinzip

Das Prinzip der QAM besteht darin, dass zwei zueinander orthogonale Trägerkomponenten separat in ihrer Amplitude moduliert und dann addiert werden. Dies führt dazu, dass das resultierende Signal sich in seiner Amplitude und Phasenlage ändert. Auf diese Art und Weise können mit einem einzigen übertragenen Symbol mehrere Bits übermittelt werden. Am besten führt man sich das ganze anhand einer komplexen Ebene vor Augen (Abbildung 6.1-1).

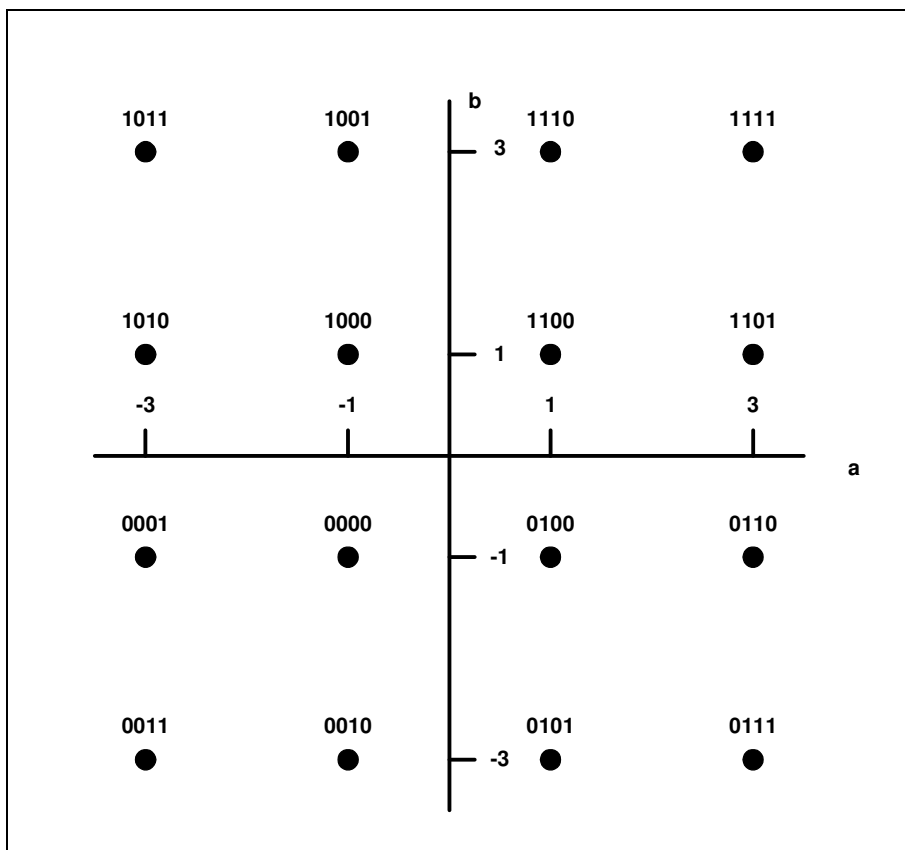


Abbildung 6.1-1 – Komplexe Ebene (Raumpunkte)

Den realen Teil,  $a_n$ , eines sogenannten Raumpunktes bezeichnet man als Inphasen-Komponente, den imaginären Teil,  $b_n$ , als Quadratur-Komponente. Überträgt man nun einen Raumpunkt, so sind dies wie hier gezeichnet bei der 16-QAM vier Bits ( $2^4 = 16$ ), die mit einem Symbol übertragen werden. Ein Symbol bzw. Zeichen setzt sich nun also folgendermassen als  $c_n = a_n + jb_n$  zusammen.

## 6.2 Sender

In Abbildung 6.2-6 ist das Prinzipschema eines QAM-Senders abgebildet. Die folgenden Überlegungen beziehen sich auf ein 16-QAM-Sender.

Am Eingang des Senders liegt ein seriell, binäres Datensignal an. Im Serie-Parallel-Konverter werden nun vier seriell ankommende Bits zusammengenommen und parallel als Vektor mit der Tiefe 4 dem Mapper übergeben. Dieser wandelt diesen Vektor in einen Real- und einen Imaginär-Teil um, d.h. er ermittelt den zu übertragenden Raumpunkt.

In Abbildung 6.2-2 ist ein Raumpunkt dargestellt, der sich aus dem Signal in Abbildung 6.2-1 ergibt.

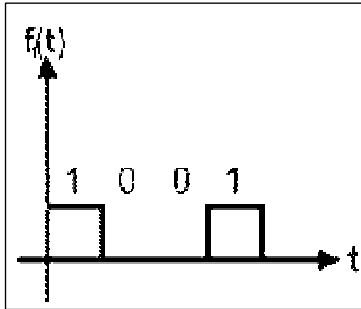


Abbildung 6.2-1 – 4 serielle Bits

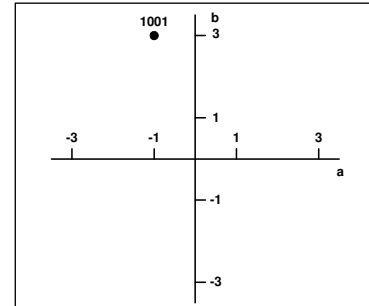


Abbildung 6.2-2 – Raumpunkt aus Abbildung 6.2-1

Abbildung 6.2-4 illustriert die ermittelten Raumpunkte 12 serieller Bits aus Abbildung 6.2-3.

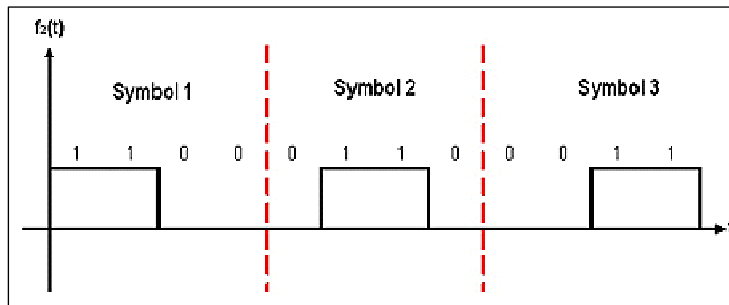


Abbildung 6.2-3 – 12 serielle Bits

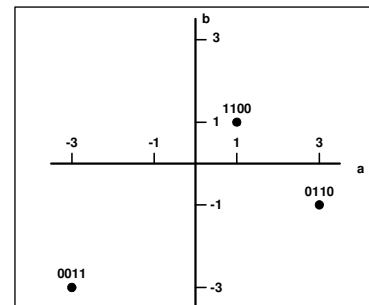


Abbildung 6.2-4 – Raumpunkte der Bits aus Abbildung 6.2-3

Hat man einmal die Inphasen- und die Quadratur-Symbolkomponenten anhand der vier Bits ermittelt, so werden diese je durch ein Puls-Amplituden-Modulationsverfahren mit den zwei orthogonalen Trägerkomponenten moduliert und danach voneinander subtrahiert. Zuvor jedoch müssen diese Komponenten über einen Impulsmodulator und ein Tiefpassfilter gelassen werden.

Der Pulsmodulator arbeitet so, dass er das am Eingang anliegende Signal nur während einem einzigen Taktimpuls am Ausgang erscheinen lässt und während der restlichen Zeit, bis das nächste Zeichen am Eingang erscheint, Nullen ausgibt.

Die Ausgangssignale der beiden Pulsmodulatoren lassen sich folgendermassen beschreiben [5]:

$$a^*(t) = \sum_{k=-\infty}^{\infty} a_k \cdot \delta(t - k \cdot T_{sym})$$

**Gleichung 6.2-1**

$$b^*(t) = \sum_{k=-\infty}^{\infty} b_k \cdot \delta(t - k \cdot T_{sym})$$

**Gleichung 6.2-2**

Das anschliessende Tiefpassfilter mit der Übertragungsfunktion  $g_T(t)$  reagiert darauf mit seiner Pulsantwort. Der nächste Zeichen-Puls ( $a^*$  bzw.  $b^*$ ) erscheint dann am Eingang des TP-Filters, wenn sich die Pulsantwort vom Filter des vorherigen Zeichens in einem Nulldurchgang befindet. Auf diese Weise hat man keine Symbolinterferenz, d.h. die aufeinanderfolgenden Zeichen beeinflussen sich gegenseitig nicht.

Das Tiefpassfilter muss also so dimensioniert sein, dass seine Pulsantwort sich dann in einer Nullstelle befindet, wenn das nächste zu übertragende Zeichen an seinem Eingang anliegt. Die Ausgangssignale dieser Filter, d.h. die Inphasen- und die Quadratur-Komponente lassen sich folgendermassen beschreiben:

$$a(t) = \sum_{k=-\infty}^{\infty} a_k \cdot g_T(t - k \cdot T_{sym})$$

**Gleichung 6.2-3**

$$b(t) = \sum_{k=-\infty}^{\infty} b_k \cdot g_T(t - k \cdot T_{sym})$$

**Gleichung 6.2-4**

Nach dem Tiefpassfilter wird die Inphasen-Komponente  $a(t)$  mit einem Kosinus-Signal und die Quadratur-Komponente  $b(t)$  mit einem Sinus-Signal multipliziert. Diese beiden Signale sind die Trägersignale und haben die Frequenz  $f_c$ , bzw.  $\omega_c$ . Nach dieser Multiplikation wird, um das endgültige, zu übertragende Zeichen zu erhalten, die Quadratur- von der Inphasen-Komponente subtrahiert.

$$s(t) = a(t) \cdot \cos(\omega_c \cdot t) - b(t) \cdot \sin(\omega_c \cdot t)$$

**Gleichung 6.2-5**

Abbildung 6.2-5 veranschaulicht das Spektrum des zu senden Signals.

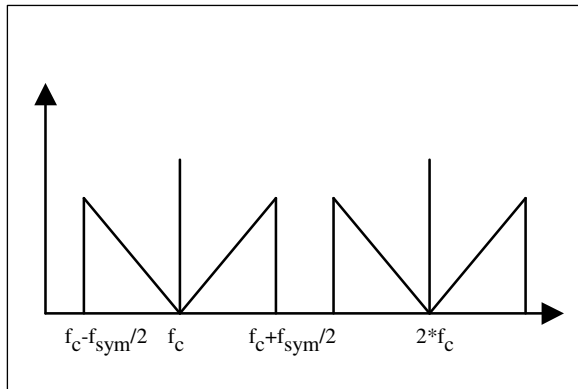


Abbildung 6.2-5 – Spektrum des Modulierten Signals

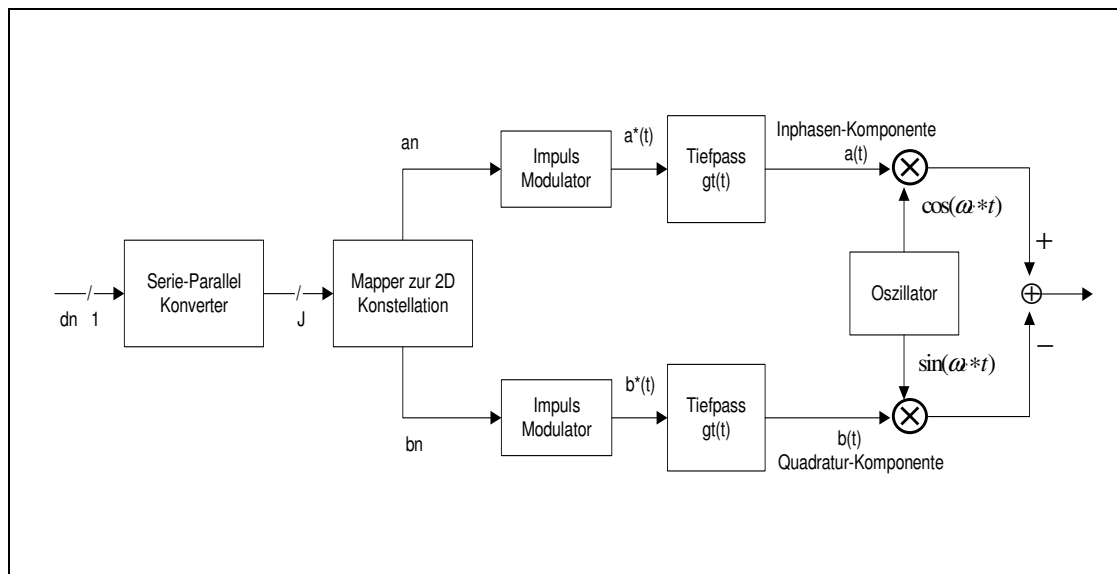


Abbildung 6.2-6 – Prinzipschema eines QAM-Senders

## 6.3 Empfänger

Der Empfänger ist so aufgebaut, dass das Empfangssignal nach einem Eingangsfiler, dessen Aufgabe es ist hochfrequente Störsignale auszufiltern, in zwei Zweige aufgesplittet wird (Abbildung 6.3-1). Das Ausgangssignal, des oberen Multiplikators, sieht folgendermassen aus:

$$s(t) \cdot 2 \cdot \cos(\omega \cdot t) = a(t) + a(t) \cdot \cos(2 \cdot \omega \cdot t) - b(t) \cdot \sin(2 \cdot \omega \cdot t)$$

Gleichung 6.3-1

Dementsprechend berechnet sich das Ausgangssignal des unteren Multiplikators zu

$$-s(t) \cdot 2 \cdot \sin(\omega \cdot t) = b(t) - b(t) \cdot \cos(2 \cdot \omega \cdot t) - a(t) \cdot \sin(2 \cdot \omega \cdot t)$$

Gleichung 6.3-2

Der zweite und dritte Term der beiden Gleichungen haben die Mitte ihres Spektrums bei  $2 \cdot \omega$ . Sie werden mit den nachgeschalteten Tiefpassfilter ausgefiltert, so dass dann an den Ausgängen nur noch  $a(t)$  und  $b(t)$  vorhanden ist.  $a(t)$  ist die zurückgewonnene Inphasen-Komponente, dementsprechend  $b(t)$  die Quadratur-Komponente.

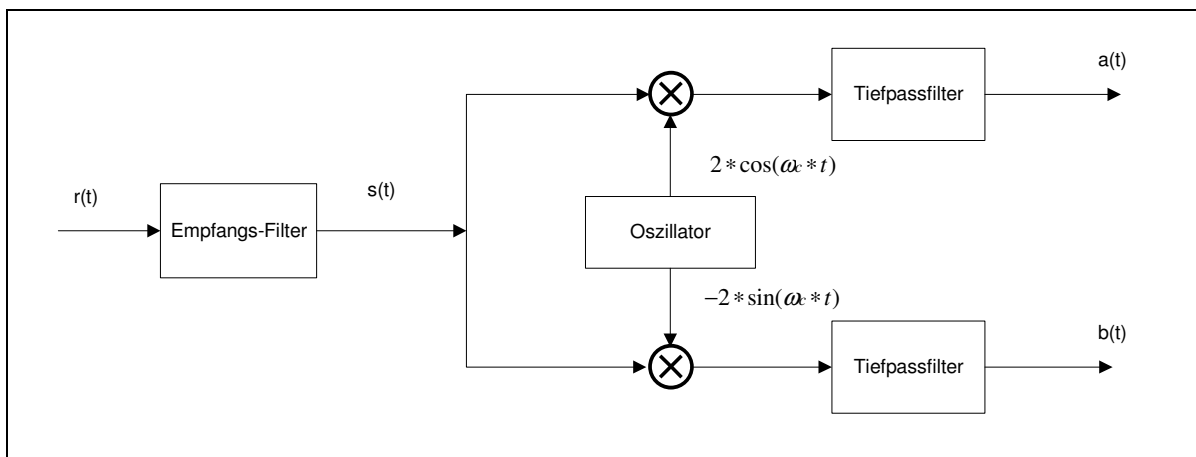


Abbildung 6.3-1 – Prinzipschema eines QAM-Empfängers

## 7 Matlab/Simulink Simulationen

### 7.1 Entwicklung des Senders

#### Simulationen ohne Impulsmodulator

In einer ersten Versuchsreihe soll der Sender auf Simulink aufgebaut werden. Die Funktionen der einzelnen Blöcke werden mit Matlab-Funktionen realisiert. Diese sind im Anhang aufgeführt. In Abbildung 7.1-1 ist der Sender, so wie er in Simulink realisiert wurde, dargestellt.

Die Data-Source ist die Datenquelle, die binäre Werte mit der Bitfrequenz  $R_b$  liefert. Im Buffer werden immer vier Bits zu einem Vektor zusammengefasst. Die Frequenz der Vektoren beträgt nun noch  $\frac{1}{4} \cdot R_b$ , d.h. die Zeichenfrequenz  $f_{sym}$ . Der Mapper ermittelt aus dem Datenvektor den zu übertragenden Raumpunkt. Er gibt einen komplexen Vektor aus, der den Raumpunkt 3 mal enthält.

Der Unbuffer taktet dann diesen drei Element langen Vektor mit der dreifachen Frequenz ( $f_s$ ) zu seinem Ausgang. Im nächsten Block wird ein solches komplexes Zeichen in Real- und Imaginärteil aufgeteilt. Diese Werte entsprechen den X- und Y-Werten im Koordinatensystem. Die Signale nach den Filtern sind dann die Quadratur-Komponente und Inphasen-Komponente. Bei diesen ersten Messungen wird noch kein Impuls-Modulator verwendet.

Die Tiefpassfilter sind FIR-Filter mit einem Hamming-Window. Die Grenzfrequenz muss hier relativ hoch gewählt werden, damit die zu übertragenden Raumpunkte nach diesen beiden Filtern noch zu erkennen sind. Die Grenzfrequenz liegt bei  $0.72 \cdot f_{ny}$ , d.h. bei ungefähr 2600. Die Ordnung wurde auf 61 gesetzt.

Es werden also folgende Parameter bei der Simulation verwendet:

$R_b$	=	9600	Bitrate
$f_s$	=	7200	Samplingfrequenz
$f_c$	=	1800	Trägerfrequenz
$f_{ny}$	=	$f_s/2$	Niquistfrequenz
$f_{sym}$	=	2400	Symbolfrequenz

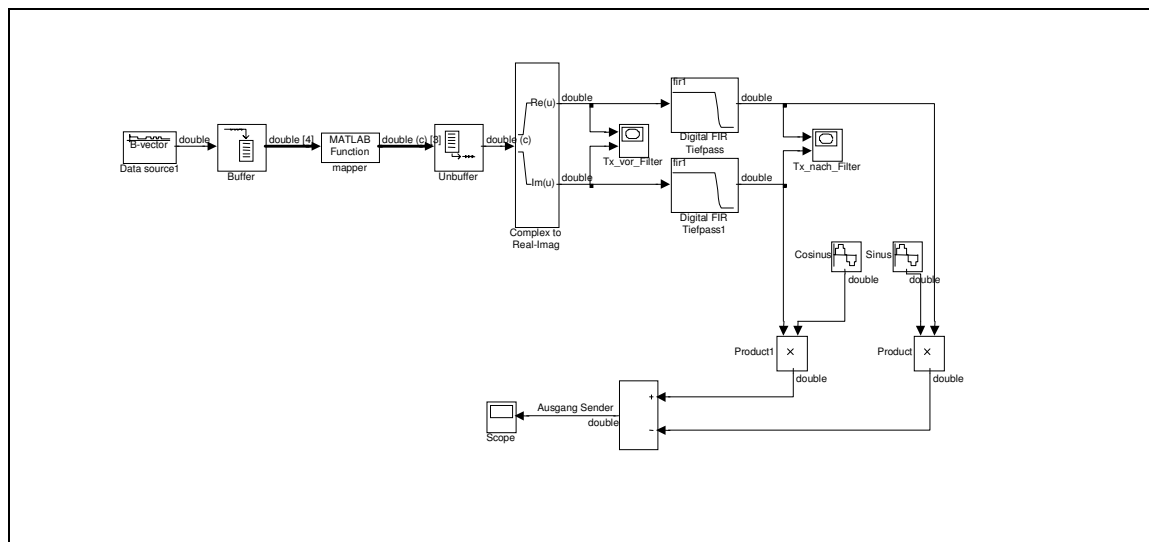


Abbildung 7.1-1 – Grundlegender Sender

In Abbildung 7.1-2 sieht man die von einem zufälligen binären Signal ermittelten Raumpunkte vor dem Filter. In Abbildung 7.1-3 sieht man wie das Filter das Bild „verschmiert“.

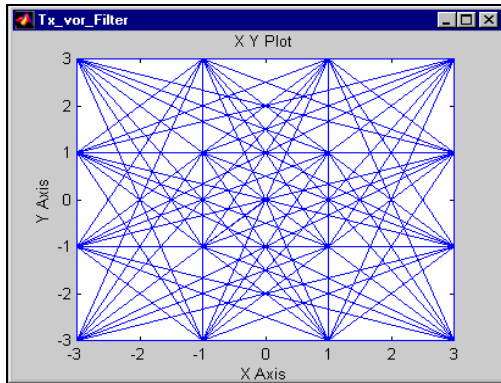


Abbildung 7.1-2 – Raumpunkte vor dem Filter

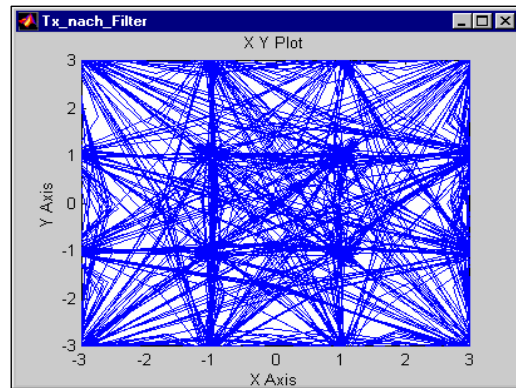


Abbildung 7.1-3 Raumpunkte nach den Filter,  $f_g=2400$

Die Grenzfrequenz sollte eigentlich so wie in [3] beschrieben auf der halben Symbolfrequenz liegen. Dies daher, weil man von diesem Signal ja nur die Grundwelle übertragen und alle höher gelegenen Frequenzen unterdrücken will. Sie wäre dann bei 1200 d.h. 1/3 der Nyquistfrequenz. In den nächsten beiden Plots (Abbildung 7.1-4 und Abbildung 7.1-5) sieht man wiederum die Raumpunkte vor und nach dem Filter. Diesmal jedoch ist die Grenzfrequenz der Filter bei  $0.3333 \cdot f_{ny}$ , d.h. bei 1200.

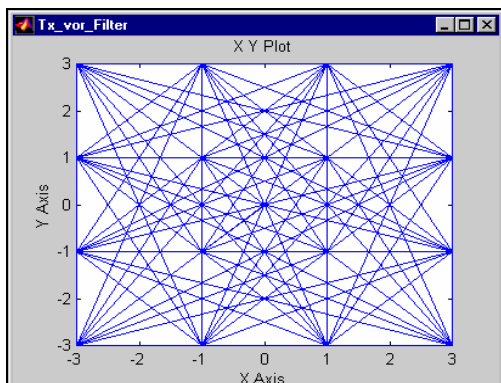


Abbildung 7.1-4 - Raumpunkte vor dem Filter

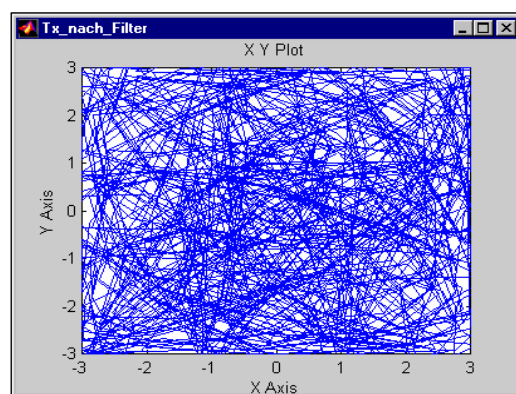


Abbildung 7.1-5 - Raumpunkte nach den Filter,  $f_g=1200$

Wie in Abbildung 7.1-5 zu erkennen ist, sind die Raumpunkte, welche man in Abbildung 7.1-3 deutlich sieht, fast nicht mehr zu erkennen.

In der Literatur [3], [2],[1] findet man oft auch ein sogenanntes Raised-Cosine-Filter beschrieben, welches sich durch eine enorm kurze Sprungantwort auszeichnet.

Wird nun ein solches Raised-Cosine-FIR-Filter statt die normalen FIR-Filter verwendet, so sehen die Raumpunkte nach den Filtern folgendermassen aus (Abbildung 7.1-7):

Folgende Parameter sind am Raised-Cosine-Filter eingestellt:

Grenzfrequenz:  $0.333 \cdot f_{ny} = 1200$   
 Ordnung: 48  
 Rolloff-Faktor: 0.5

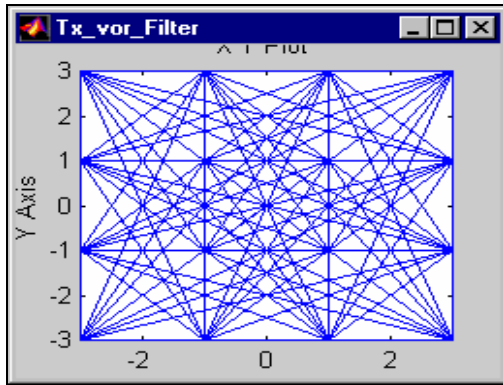


Abbildung 7.1-6 - Raumpunkte vor dem Filter

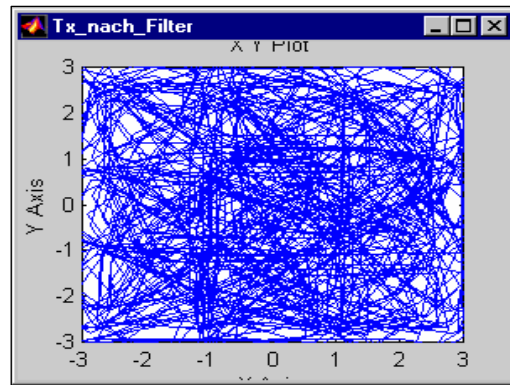


Abbildung 7.1-7 - Raumpunkte nach dem Filter,  $f_g=1200$ , Raised-Cosine-Filter

Wie in Abbildung 7.1-7 zu sehen ist, sind die Raumpunkte relativ schwer zu erkennen, wie es schon in Abbildung 7.1-5 der Fall war.

Man stellt also fest, dass die Filter, egal welches man verwendet, die Signale zu sehr zu verzerren. Dies kommt daher, dass die „Antworten“ der Filter, die ja so wie es bei den Messungen gemacht wurde, keine Impulsantworten sind, sich gegenseitig überlappen, d.h. es tritt Symbolinterferenz ein.

Die aufeinanderfolgenden Symbole dürfen sich am Ausgang des Filters also nicht überlappen, soviel steht fest. Dies bedeutet, dass wie schon in der Theorie beschrieben, ein Impulsmodulator vor das Filter geschaltet werden muss. Seine Aufgabe besteht darin, aus dem Symbol, welches mehrere Takte andauert einen Impuls einer Taktlänge (Sampleperiode) zu machen. Das Filter reagiert dann auf diesen Impuls mit seiner Impulsantwort. Damit sich nun die aufeinanderfolgenden Zeichen gegenseitig nicht beeinflussen, muss sich der Ausgang des Filters dann in einer Nullstelle befinden, wenn am Eingang der neue Zeichenpuls anliegt.

### Raised-Cosine-Filter

Das Raised-Cosine Filter hat seine erste Nullstelle bei  $T$  [5].  $T$  ist also die Zeit zwischen zwei Zeichenpulsen. Es ergibt sich also folgender Formalismus:

$$f_N = \frac{1}{2 \cdot T} = f_g = \frac{f_{sym}}{2}$$

Gleichung 7-1

$$T = \frac{1}{f_{sym}}$$

Gleichung 7-2

Die Formel für ein Raised-Cosine-Filter sieht wie folgt aus:

$$g^T(t) = \frac{\sin(\pi \cdot t/T)}{\pi \cdot t/T} \cdot \frac{\cos(\alpha \cdot \pi \cdot t/T)}{1 - 4(\alpha \cdot t/T)^2}$$

Gleichung 7-3

Der Parameter  $\alpha$  beschreibt die Breite des Überganges vom Durchlass- in den Sperrbereich, also die Flankensteilheit. Er wird als Roll-off-Faktor bezeichnet;  $\alpha=0$  beschreibt den idealen Tiefpass,  $\alpha=1$  die reine Kosinusform.

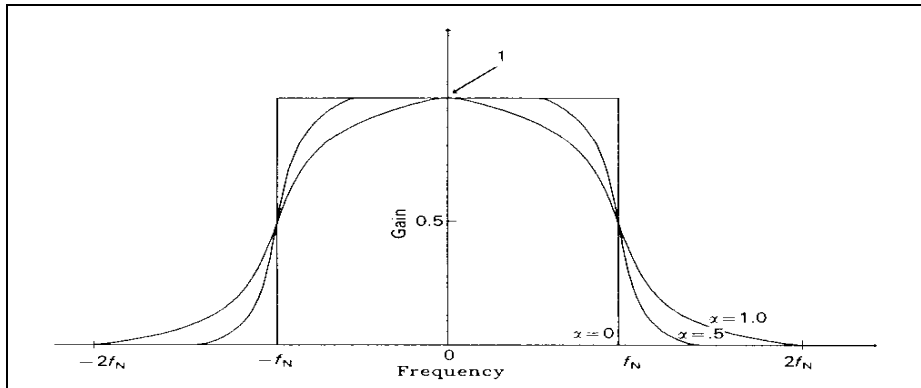


Abbildung 7.1-8 – Frequenzgang Raised-Cosine-Filter mit versch. Rolloff-Faktoren

In Abbildung 7.1-8 ist der Frequenzgang eines Raised-Cosine-Filters mit verschiedenen Rolloff-Faktoren zu sehen. Zu beachten ist, dass bei der Grenzfrequenz,  $f_g=f_N$  das Signal schon beachtlich geschwächt wird, wenn ein Faktor von 0.5 verwendet wird. In Abbildung 7.1-9 erkennt man aber, je grösser der Faktor gewählt wird, um so weniger lang dauert das Ausschwingen der Sprungantwort.

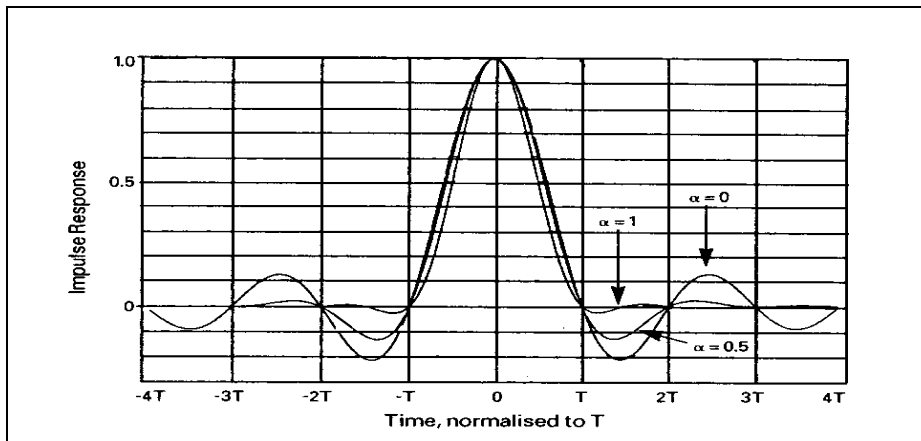


Abbildung 7.1-9 – Sprungantwort eines Raised-Cosine-Filters mit versch. Rolloff-Faktoren

## Simulation mit Raised-Cosine-Filter und Impulsmodulator

In Abbildung 7.1-10 ist der Sender nun mit Impulsmodulator abgebildet. Diese Impulsmodulatoren werden ganz einfach mit Upsamplern realisiert. Die zuvor verwendete Funktion ist nun weggefallen. Der Vector-to-Scalar-Converter wandelt den 4-Bit-Vektor in eine Skalar um. Der Data-Mapper konvertiert diesen Skalar dann aufgrund eines Vergleichsvektors in die entsprechende X- bzw. Y-Koordinate.

Aufgrund der im letzten Abschnitt beschriebenen Dämpfung des Signals durch Raised-Cosine-Filter, muss also zusätzlich nach dem Impulsmodulator ein Verstärker vor das Filter geschaltet werden, damit am Ausgang dann die Raumpunkte am selben Ort zu liegen kommen wie bei den vorangegangenen Messungen.

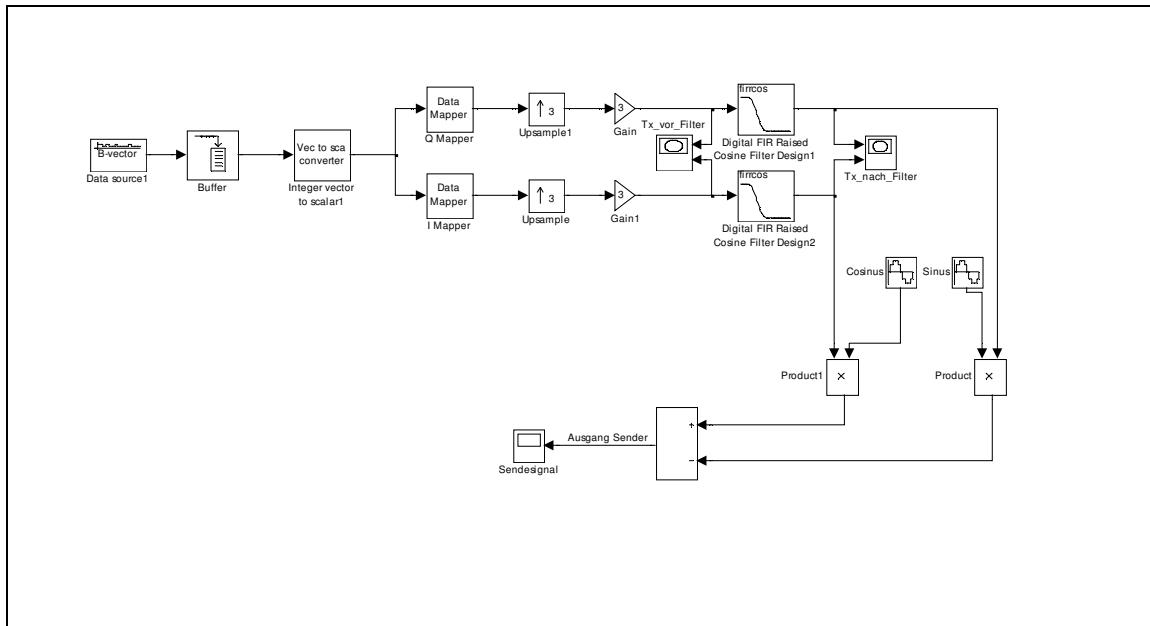


Abbildung 7.1-10 – Sender mit Impulsmodulator

Wie man in Abbildung 7.1-12 erkennt, sind die Raumpunkte wesentlich besser sichtbar als bei den Versuchen zuvor. Das Signal vor den Filtern (Abbildung 7.1-11) sieht nun anders aus als zuvor. Aufgrund des Impulsmodulators geht das Signal nun immer wieder auf den Nullpunkt zurück.

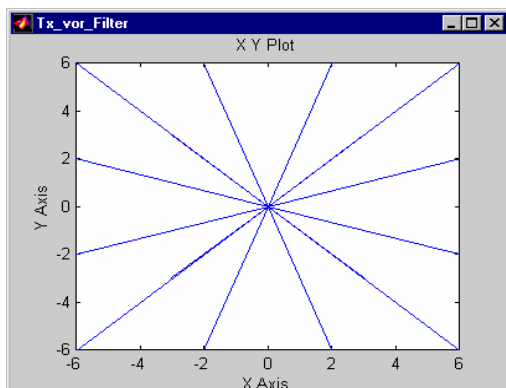


Abbildung 7.1-11 – Raumpunkte nach Impulsmodulator

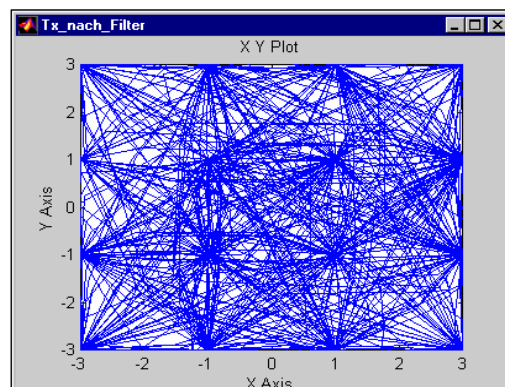


Abbildung 7.1-12 – Raumpunkte nach den Filter,  $f_g=1200$ , Raised-Cosine-Filter

Eigentlich sind es ja nicht die Raumpunkte die man übermitteln will. Die Raumpunkte setzen sich ja aus der Inphasen- und Quadraturkomponente zusammen. Diese müssen nun noch moduliert werden. Dies geschieht, wie in Abbildung 7.1-10 zu erkennen ist, nach den Filtern. Wie das definitive, zu übermittelnde Signal einer zufälligen Bitsequenz aussieht, zeigt Abbildung 7.1-13.

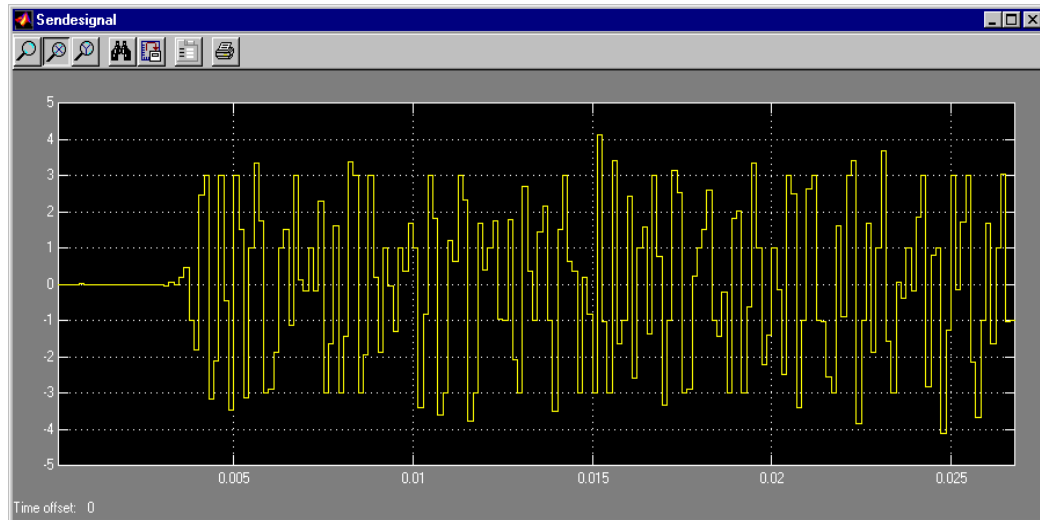


Abbildung 7.1-13 - Sendesignal

Der Sender funktioniert also grundsätzlich.

## 7.2 Übergang zum Empfänger

### Aufbau des Empfängers

Wie der Empfänger in Simulink realisiert wird, zeigt Abbildung 7.2-1.

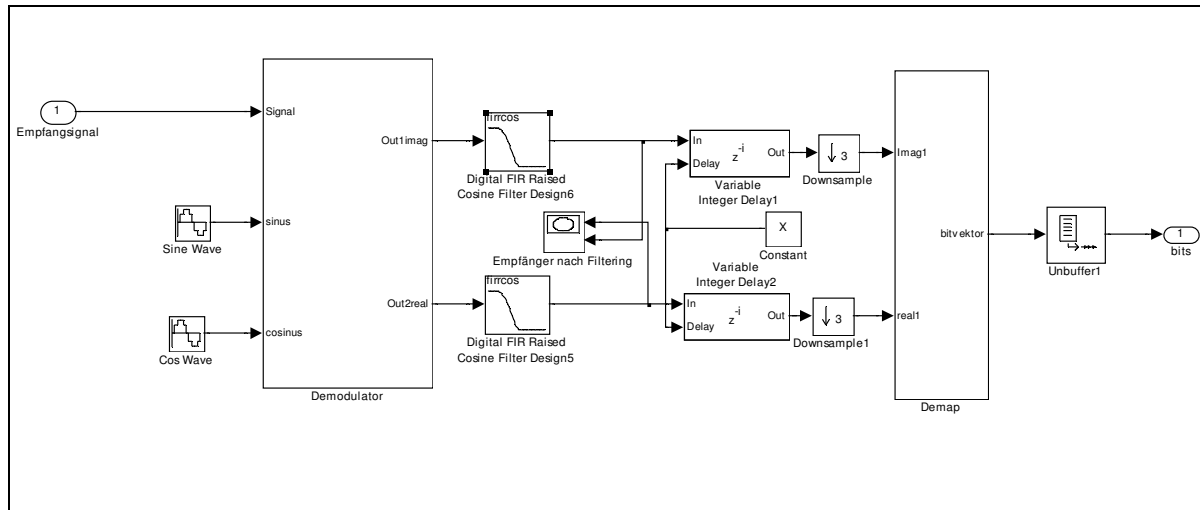


Abbildung 7.2-1 - Empfänger

Der Aufbau ist vom Prinzip her der Selbe wie in der Theorie beschrieben. Das Empfangssignal gelangt direkt auf den Demodulator (Abbildung 7.2-2). Hier wird am Empfängereingang kein Eingangsfiler verwendet. Der Sinn dieses Filters wäre, hochfrequente Signale vom Empfangssignal wegzufiltern. Diese entstehen jedoch hier in Simulink nicht, deshalb wird es weggelassen. Nach dem Demodulator gelangen die Inphasen- und die Quadraturkomponente je über einen Tiefpass. Danach sind die Raumpunkte im XY-Plot sichtbar. Um nun die Signale im richtigen Moment abzutasten, wird ein verstellbares Verzögerungsglied (Variable-Integer-Delay) in die Leitung geschaltet. Über eine Konstante (X) kann dieser Delay eingestellt werden. X bewegt sich zwischen 0 und 2. Dies daher, da ja die Samplingfrequenz ( $f_s=7200$ ) drei mal höher liegt als die Zeichenfrequenz ( $f_{sym}=2400$ ). Jedes übermittelte Zeichen dauert also 3 Samples lang an. Der Wert dieser drei Samples ist jedoch nicht konstant, da wir ja beim Sender das Signal über einen Pulsmodulator gelassen haben.

Der anschliessende Downsampler verringert die Samplingfrequenz auf einen drittel, d.h. wieder auf die Zeichenfrequenz. Er nimmt jeden dritten Wert und hält ihn dann während den nächsten zwei Taktzyklen (Sampleperioden) an. Deshalb muss eben das Integer Delay vorgeschaltet werden, um den richtigen Wert während den nächsten zwei Zyklen durchzuschalten. Dieser Wert wird dann im Demapper wieder in einen 4-Bit Vektor gewandelt. Der Demapper ist in Abbildung 7.2-3 abgebildet. Es ist eine reine Vergleichssache, welche auf booleschen Gleichungen beruht, die mit Leichtigkeit aus dem Raumpunktdiagramm im Theorieteil hergeleitet werden können. Es wird hier darauf verzichtet diese Gleichungen aufzuführen. Um entscheiden zu können, zu welchem Raumpunkt das empfangene Zeichen gehört, müssen Schwellwerte festgelegt werden. Am besten wählt man die Schwellwerte bei  $-2$ ,  $2$  und  $0$ , da diese Werte jeweils in der Mitte von  $1$  &  $3$ , bzw.  $-1$  &  $-3$  und  $-1$  &  $1$  liegen

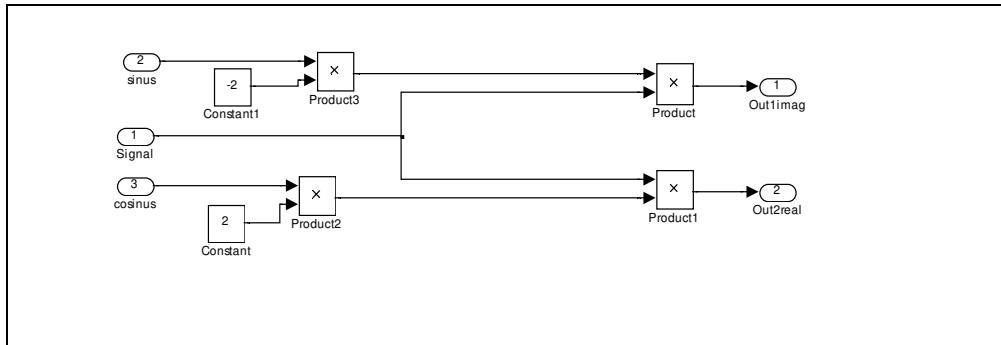


Abbildung 7.2-2 - Demodulator

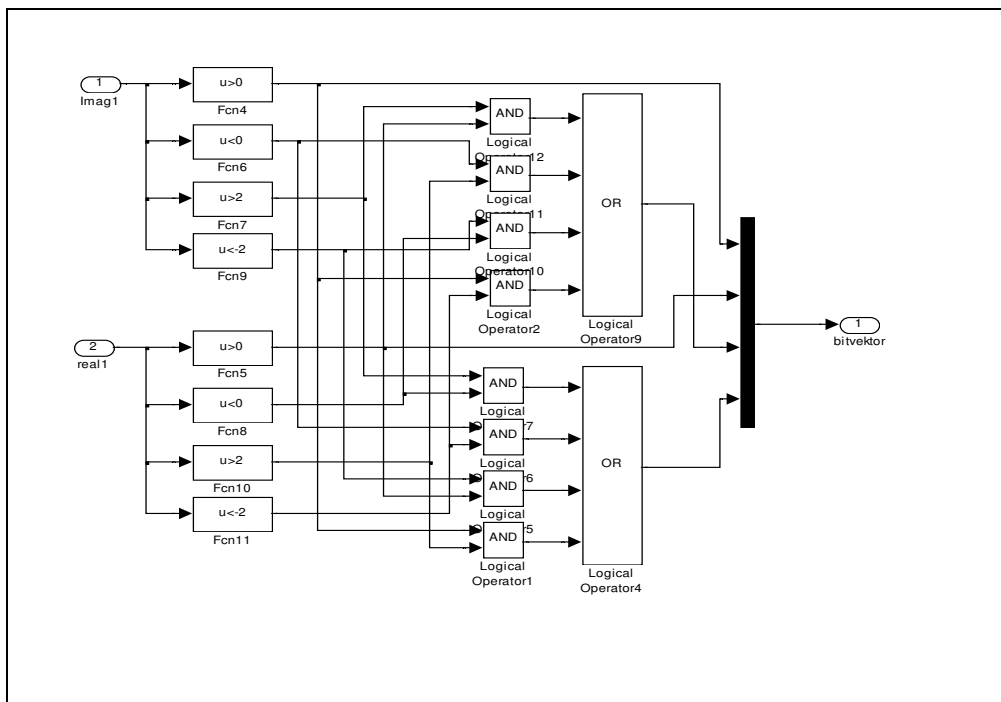


Abbildung 7.2-3 - Demapper

Das Ausgangssignal des Demappers ist, wie schon oben erwähnt ein 4-Bit Vektor. Dieser Vektor wird mit einem Unbuffer wieder in einen kontinuierlichen Bitstrom gewandelt. Der Bitstrom hat dann die vierfache Zeichenfrequenz, d.h. wieder die Bitrate  $R_b$ .

## Ergänzung zu den Filtern

Wie schon beim Sender und in der Theorie erwähnt, ist das Ausgangssignal des Sendefilters im Prinzip ja die Sprungantwort des Filters, da ja der Real- bzw. der Imaginärteil des zu sendenden Signals mit einem Impulsmodulator moduliert wird. Das Ziel ist es, dass sich der Ausgang dann in einem Nulldurchgang befindet, wenn das neue Zeichen, bzw. der neue Zeichenpuls am Eingang anliegt.

Da ja aber im Sender und im Empfänger ein solches Filter vorhanden ist, müsste sich ja eigentlich die Sprungantwort über zwei Filter hinweg, d.h. am Ausgang des zweiten Filters in einer Nullstelle befinden, wenn beim ersten Filter der neue Zeichenpuls erscheint.

Konkret ausgedrückt muss sich in unserem Fall, wo die Samplingfrequenz ( $f_s=7200$ ) drei mal grösser ist als die Zeichenfrequenz ( $f_{\text{sym}}=2400$ ), der Ausgang des Filters im Empfänger bei jedem dritten Wert vom Peak weg gezählt in einer Nullstelle befinden.

Es soll nun überprüft werden, ob dies mit den Raised-Cosine-Filtern der Fall ist. Es werden dazu zwei Raised-Cosine-Filter hintereinander geschaltet und die Ausgänge je mit einem KO (Scope) betrachtet. Das Eingangssignal ist ein Vektor, der sich aus 28 Nullen und einer Eins zusammensetzt, d.h. ein Dirac-Impuls. Der Aufbau ist in Abbildung 7.2-4 dargestellt. Auf diese Weise können die Sprungantworten der Filter betrachtet werden.

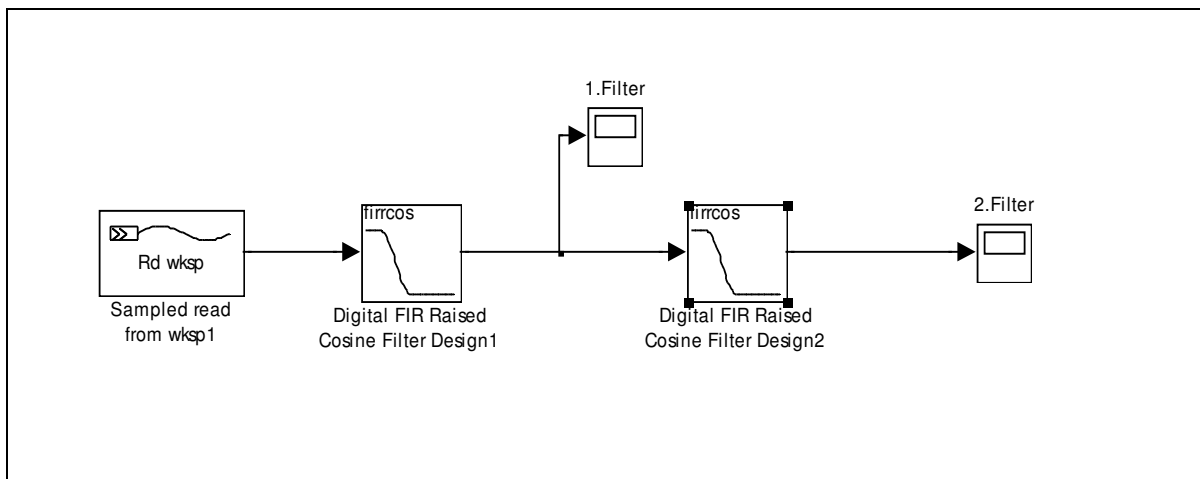


Abbildung 7.2-4 – Aufbau zur Ermittlung der Sprungantwort

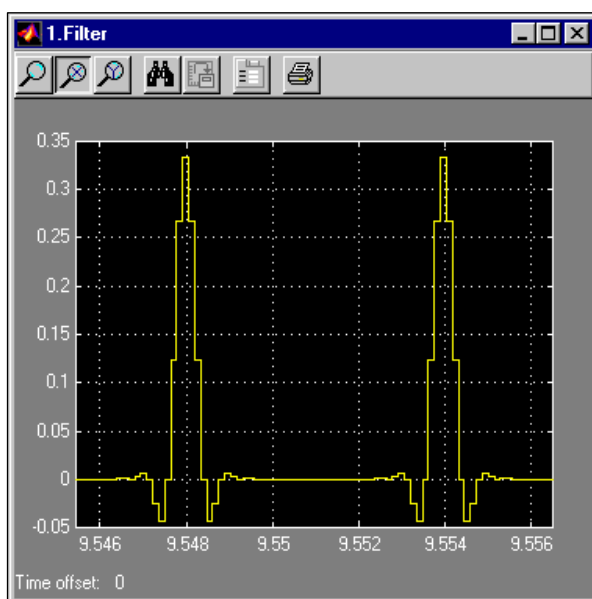


Abbildung 7.2-5 – Sprungantworten des 1. Raised-Cosine-Filter

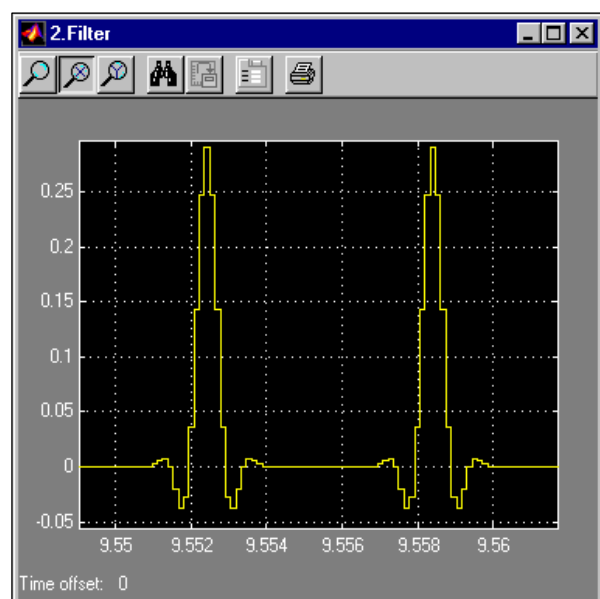


Abbildung 7.2-6 – Sprungantworten des 2. Raised-Cosine-Filter

Wie man in Abbildung 7.2-5 erkennt, ist der Ausgang des ersten Filters bei jedem dritten Wert vom Peak weg gezählt in einem Nulldurchgang. Der Ausgang des zweiten Filters (Abbildung 7.2-6) befindet sich hingegen nicht dort wo man es gern hätte, nämlich auch in einem Nulldurchgang. Das würde bedeuten dass sich im Empfänger die aufeinanderfolgenden Zeichen beeinflussen. Um dies zu verhindern, verwendet man in der Praxis oft auch Root-Raised-Cosine-Filter. Schaltet man nämlich zwei solche Filter hintereinander, so ist die Sprungantwort über beide Filter betrachtet, die Selbe wie bei einem Raised-Cosine-Filter.

Die Formel für das Root-Raised-Cosine-Filter sieht im Zeitbereich folgendermassen aus [4] (Gleichung 7-4):

$$g_T(t) = \frac{4 \cdot \alpha \cdot t/T \cdot \cos(\pi \cdot t/T \cdot (1-\alpha))}{1 - \pi \cdot t \cdot T \cdot (4 \cdot \alpha \cdot t/T)^2}$$

Gleichung 7-4

In Simulink kann das Raised-Cosine-Filter durch einen kleine Veränderung in ein Root-Raised-Cosine-Filter umgewandelt werden. Verwendet man nun zwei solche Filter hintereinander, so kann man in Abbildung 7.2-8 erkennen, dass sich nun beim Ausgangssignal des zweiten Filters jeder dritte Wert vom Peak weg in einem Nulldurchgang befindet, hingegen beim Ausgang des ersten Filters (Abbildung 7.2-7) ist dies nicht der Fall.

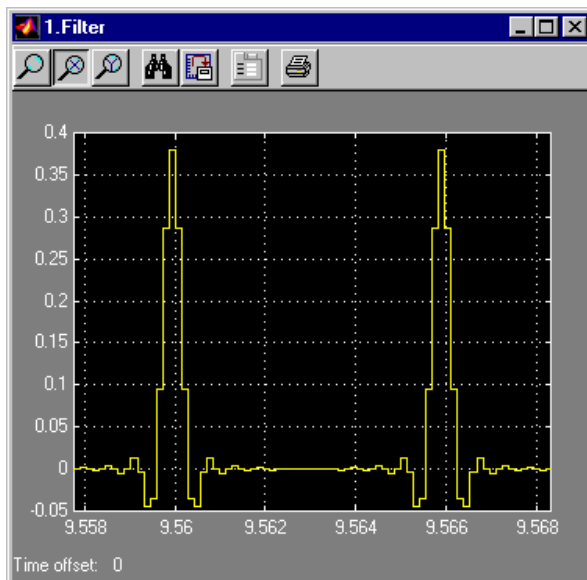


Abbildung 7.2-7 – Sprungantworten des 1. Root-Raised-Cosine-Filters

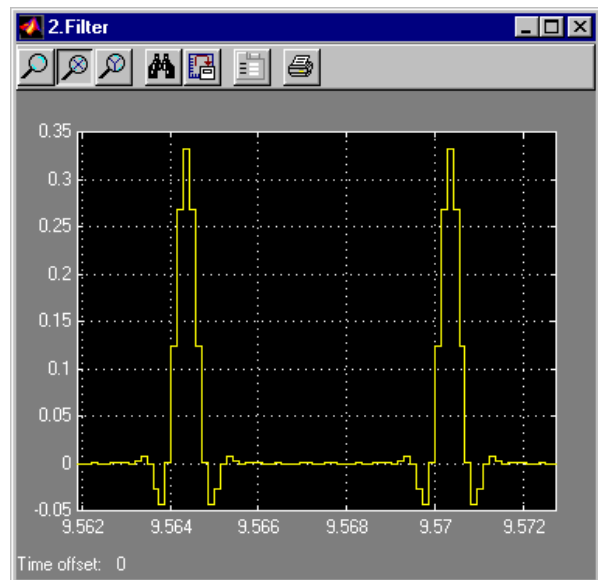


Abbildung 7.2-8 - Sprungantworten des 2. Root-Raised-Cosine-Filters

## 7.3 Sender und Empfänger

### 7.3.1 Grundlegendes Modell

#### Aufbau

In Abbildung 7.3-1 ist nun das Modem, so wie es in Simulink realisiert wurde, abgebildet. Der Sender und der Empfänger sind separat in Abbildung 7.3-2 und Abbildung 7.3-3 aufgeführt.

Um die Fehler in der Übertragung zählen zu können, hat man in Simulink mit einem Error-Rate-Calculation Block die Möglichkeit die gesendeten und die empfangenen Bits zu zählen. Es ist jedoch zu beachten, dass man je nach Versuchsaufbau eine kürzere oder längere Verzögerung aufgrund der FIR-Filter und anderer Verzögerungsglieder miteinberechnen muss. In diesem Aufbau beträgt die Verzögerung 80 Taktzyklen.

Der Wert des Integer-Delay (Abbildung 7.3-3) im Empfänger wird durch Ausprobieren ermittelt. Der Wert, bei dem die wenigsten Fehler in den Übertragung vorkommen, wird eingesetzt. In diesem Fall ist es der Wert 0. Mit diesem Wert treten am wenigsten Fehler auf.

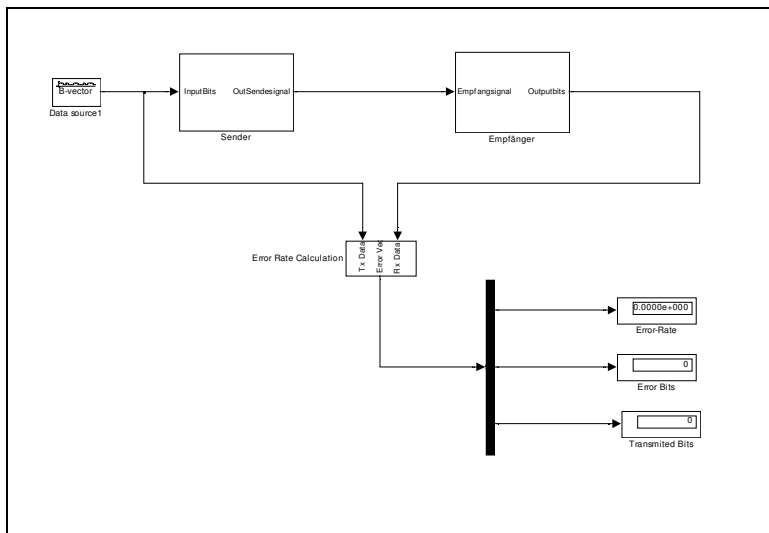


Abbildung 7.3-1 – Aufbau des Modems

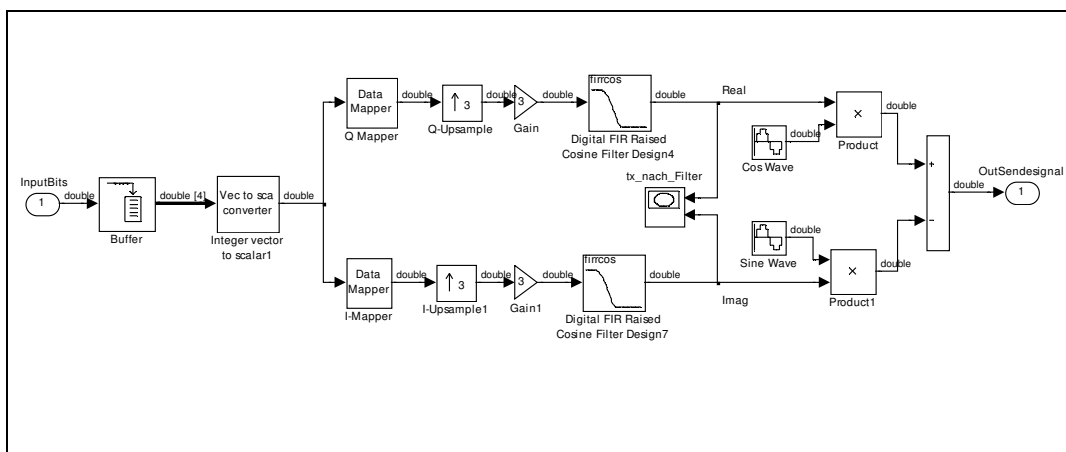


Abbildung 7.3-2 - Sender

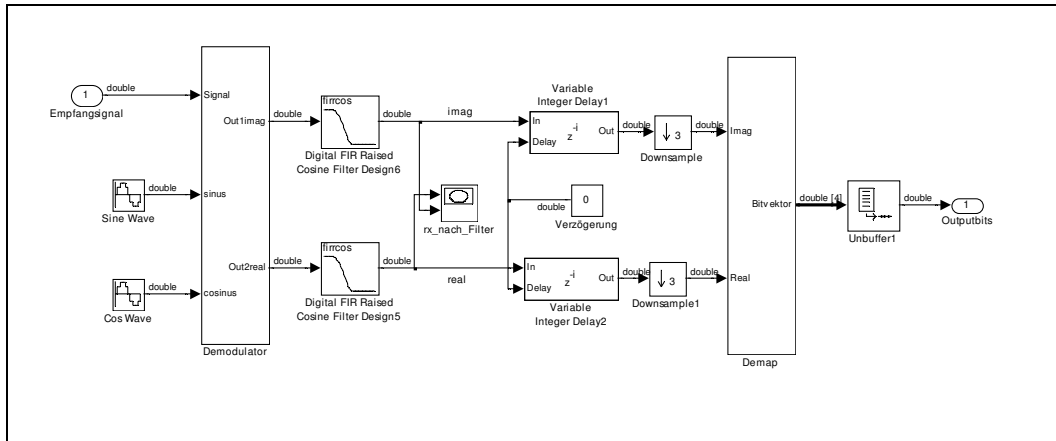


Abbildung 7.3-3 - Empfänger

Der Demapper und der Demodulator sind in Abbildung 7.2-3 und Abbildung 7.2-2 aufgeführt.

### Simulation des Senders und des Empfängers

Als Datensignal wird wieder eine rein zufällige Bitsequenz verwendet. Betrachtet man nun in Abbildung 7.3-4 die Raumpunkte nach den Filtern vom Sender, so stellt man fest, dass diese recht stark um den jeweiligen Punkt herum verteilt sind. Nach den Filtern im Empfänger Abbildung 7.3-5 ist zu erkennen, dass die jeweiligen Punkte relativ dichtgedrängt dort zu liegen kommen, wo sie auch erwartungsgemäss sein sollten. Konkret heisst das, um eine relativ fehlerfreie Datenübertragung zu haben, müssen die Raumpunkte im Empfänger wirklich auch dort sein, wo man sie erwartet. Dies wird durch die Verwendung von Root-Raised-Cosine-Filtern eher garantiert, als mit einem anderen Filter.

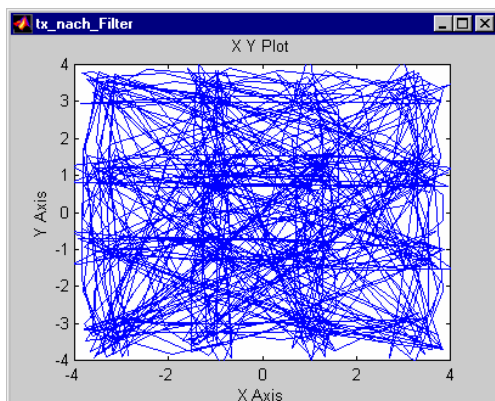


Abbildung 7.3-4 – Raumpunkte Sender nach den Filter

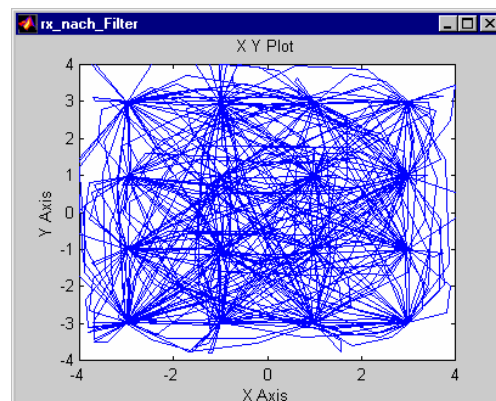


Abbildung 7.3-5 – Raumpunkte Empfänger nach den Filter

Eingehende Tests ergeben, dass diese Datenübertragung fehlerfrei funktioniert.

## 7.3.2 Automatic-Gain-Control (AGC)

### Prinzip des AGC

Verwendet man nun ein Modem im Realen, so wird aufgrund des Übertragungsweges das Signal abgeschwächt und es wird dem Signal Rauschen dazu addiert.

Um nun am Empfänger immer mit den selben Pegeln arbeiten zu können, d.h. um nicht die Entscheidungsschwellen beim Demapper dem jeweiligen Signallevel anpassen zu müssen, verwendet man am Eingang einen Automatic-Gain-Control, der das Signal auf einen bestimmten Mittelwert reguliert.

Prinzipiell arbeitet der AGC so, dass er das Eingangssignal  $In(z)$  zuerst quadriert und dann daraus den zeitlichen Mittelwert bildet, um so eine mittlere Leistung des Signals  $Pm(z)$  zu erhalten. Berechnet man die Wurzel von  $Pm(z)$ , so erhält man den mittleren Effektivwert  $Um(z)$  des Empfangssignals. Dieses  $Um(z)$  wird nun von einem Sollwert, der empirisch ermittelt wurde, subtrahiert. Diese Differenz  $D(z)$  wird nun noch mit einer Verstärkungs-konstante  $a$  multipliziert und dann integriert. Die Verstärkungs-konstante  $a$  muss sehr klein gewählt werden. Je grösser sie ist, um so flinker reagiert die ganze Regelung und das ganze könnte zu schwingen anfangen. Das Ausgangssignal des Integrators  $K(z)$  ist die Korrekturverstärkung und wird mit dem Eingangssignal multipliziert, um das Ausgangssignal des AGC zu erhalten.

Die Mittelwertbildung (Abbildung 7.3-7) erfolgt eigentlich über einen Tiefpass erster Ordnung mit der Übertragungsfunktion beschrieben in Gleichung 7-5.

$$H(z) = \frac{MWout(z)}{MWin(z)} = \frac{c}{1 - (1 - c) \cdot z^{-1}}$$

Gleichung 7-5

Das Ausgangssignal des AGC lässt sich wie folgt beschreiben

$$\begin{aligned} Out(z) &= In(z) \cdot K(z) \\ K(z) &= Dm(z) + K(z^{-1}) \\ Dm(z) &= a \cdot (Sollwert - \sqrt{Pm(z)}) \\ Pm(z) &= Out(z)^2 \cdot \frac{c}{1 - (1 - c) \cdot z^{-1}} \end{aligned}$$

Gleichung 7-6

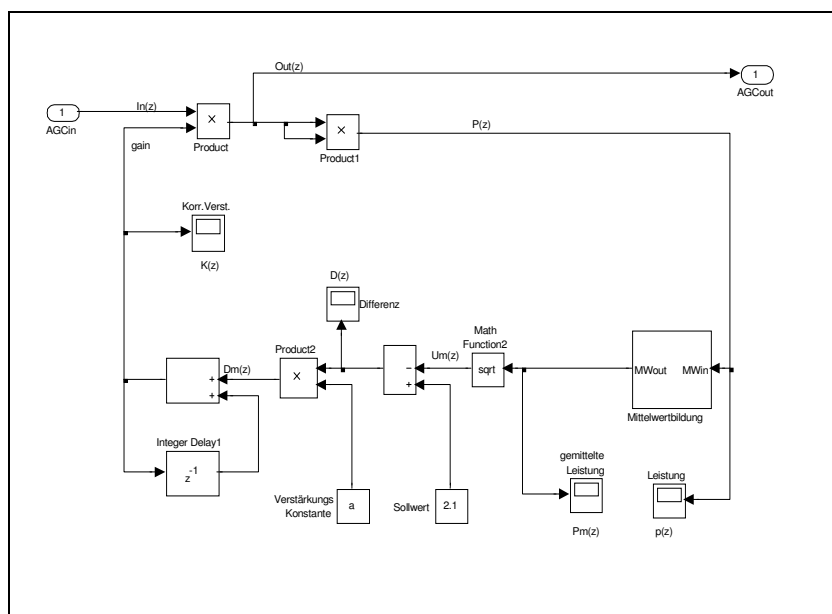


Abbildung 7.3-6 – Automatic-Gain-Control

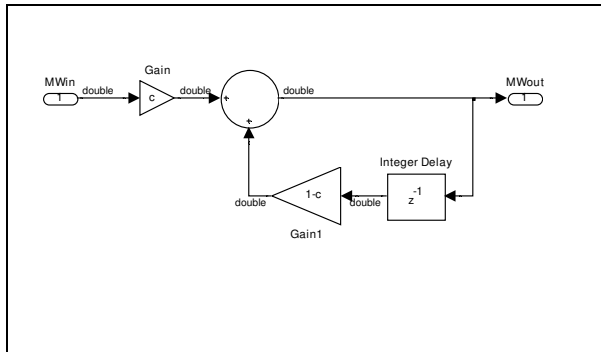


Abbildung 7.3-7 – Mittelwertbildung

### Simulationen mit AGC

Der AGC wird beim Empfänger am Eingang vor dem Demodulator plaziert. Um im realen eine ideale Aussteuerung des A/D-Wandlers zu erreichen, wäre es am sinnvollsten, man würde den AGC vor dem A/D-Wandler plazieren. Er müsste dann in analoger Schaltungstechnik realisiert werden und via DSP in der Verstärkung reguliert werden können. In Abbildung 7.3-8 sieht man nun den Empfänger mit AGC.

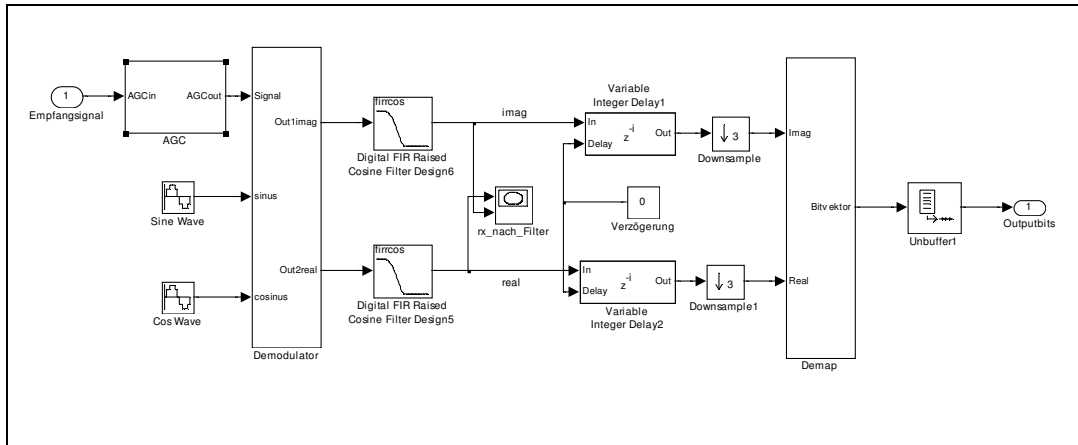


Abbildung 7.3-8 – Empfänger mit AGC

Um zu sehen wie der AGC seine Verstärkung reguliert, wird beim Sender zunächst eine schwache Signalamplitude eingestellt. Die Verstärker vor den Filtern im Sender (Abbildung 7.3-2) werden auf 1 gesetzt. Vergleicht man nun die Raumpunkte im Sender nach den Filtern Abbildung 7.3-9 mit denen z.B. von Abbildung 7.3-4, so erkennt man, dass bedeutend weniger Amplitude vorhanden ist.

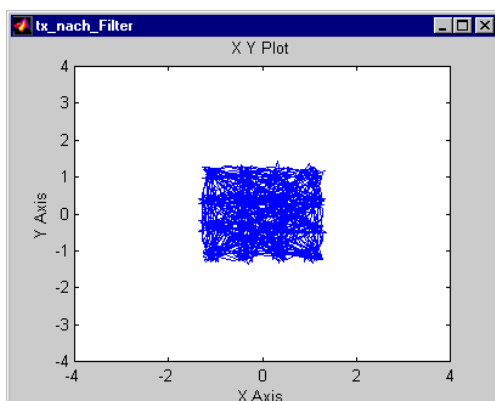


Abbildung 7.3-9 – Raumpunkte Sender

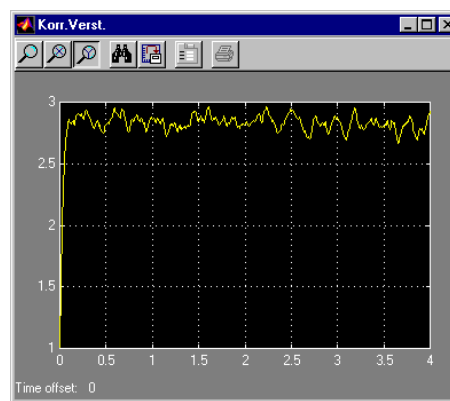


Abbildung 7.3-10 Verstärkung bei  $c=0.01$  &  $a=0.05$

Betrachtet man in Abbildung 7.3-10 den Verlauf der Verstärkung, so erkennt man, dass sie sich schon nach kurzer Zeit auf einen relativ konstanten Wert einreguliert. Bei dieser Simulation wurde  $c = 0.01$  und  $a = 0.05$  gesetzt. Dabei wurden die ersten 59 übertragenen Bits falsch interpretiert. D.h. die ersten 14 Zeichen lagen an einem absolut falschen Ort und das 15. lag dann schon in der Nähe des Raumpunktes den es sein sollte.

Macht man  $c$  10 mal kleiner ( $c = 0.001$ ) so resultiert der Verlauf der Verstärkung in Abbildung 7.3-11 dargestellt. Man sieht, je kleiner die Konstante  $c$  gewählt wird, um so stärker ist das Überschwingen. Zu beachten ist aber auch, dass sie schon nach sehr kurzer Zeit relativ konstant ist und hier schon nach 58 übertragenen Bits kein Fehler mehr auftritt.

Wird hingegen  $a$  10 mal kleiner ( $a = 0.005$ ) gemacht und  $c$  wieder auf 0.01 gesetzt, so resultiert der Verlauf der Verstärkung in Abbildung 7.3-12 dargestellt. Hier sieht man, dass sich die Verstärkung ganz langsam ihrem Sollwert nähert. Dementsprechend ist auch die Fehlerzahl viel grösser. Es wurden 666 Bits falsch interpretiert.

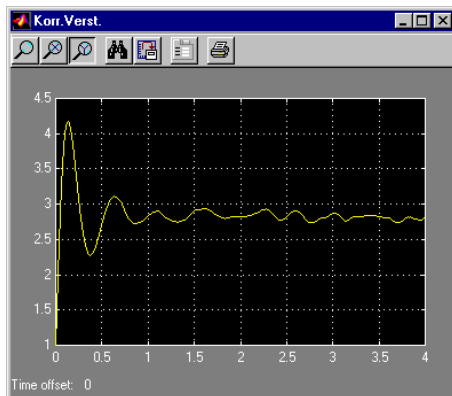


Abbildung 7.3-11 - Verstärkung bei  $c=0.001$  &  $a=0.05$

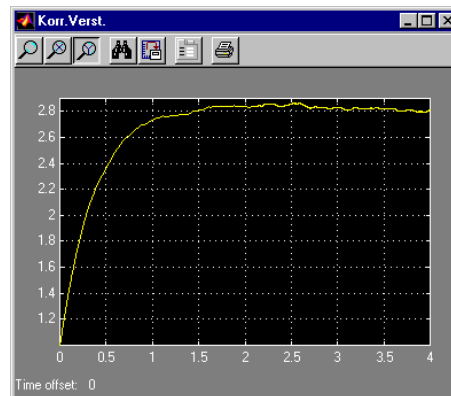


Abbildung 7.3-12 Verstärkung bei  $c=0.01$  &  $a=0.005$

Konkret bedeutet es also, je kleiner  $a$  gewählt wird und je grösser  $c$  um so langsamer wird die Verstärkung sich auf seinen Sollwert einstellen, oder anders herum, je grösser  $a$  und je kleiner  $c$  ist, um so schneller ist die Regelung, aber auch um so sensibler auf Änderungen in der Signalamplitude.

In einem nächsten Schritt soll herausgefunden werden, wie der AGC auf zu grosse Signalamplituden reagiert. Es wird wieder  $c = 0.01$  und  $a = 0.005$  gesetzt. Die Verstärker im Sender werden auf 15 gesetzt. Bei diesem Experiment stellt man nach kurzer Zeit fest, dass sich die Verstärkung nicht auf einen Wert einpendelt, sondern dass sie gegen minus unendlich hinläuft (Abbildung 7.3-13).

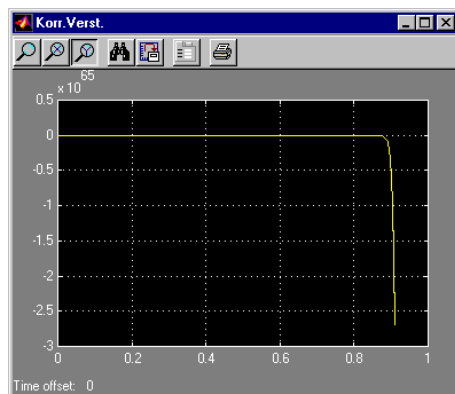


Abbildung 7.3-13 - Verstärkung bei  $c=0.01$  &  $a=0.005$

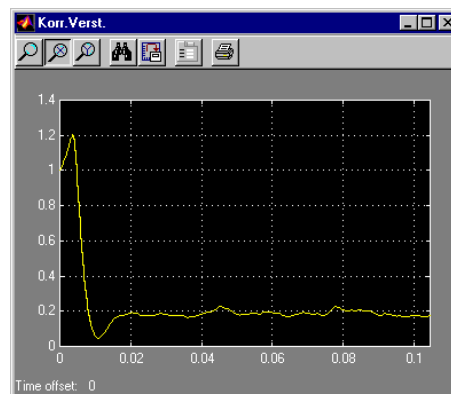


Abbildung 7.3-14 - Verstärkung bei  $c=0.1$  &  $a=0.005$

Nach einigem experimentieren mit diese beide Werten stellt man bald fest, dass die Regelung relativ träge gemacht werden muss. Dazu wird  $c = 0.1$  und  $a = 0.005$  gesetzt. In Abbildung 7.3-14 ist der Verlauf der Verstärkung dargestellt. Bei dieser Simulation wurden die Verstärker im Sender auf 15 gesetzt. Dies entspricht sozusagen einer 5-fachen Übersteuerung, da um im normalen Pegel zu senden eine Verstärkung von 3 eingesetzt werden muss. Man erkennt, dass die Verstärkung nach relativ kurzer Zeit schon einen konstanten Wert erreicht. Der Empfänger macht dabei 16 Fehler in der Erkennung. Wird hingegen mit einer zu kleinen Amplitude gesendet, so zeigt sich, dass der ganze Regelvorgang nun länger andauert. Dies veranschaulicht Abbildung 7.3-15. Hier wurden die Verstärker im Sender auf 0.5 gesetzt, was einem sechstel des normalen Sendepiegels entspricht. Dabei macht das Modem 184 Fehler.

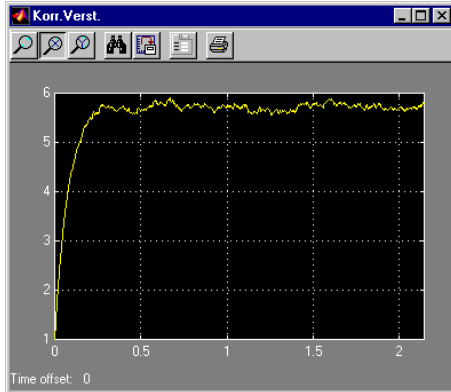


Abbildung 7.3-15 - Verstärkung bei  $c=0.1$  &  $a=0.005$

Es werden also die beiden Parameter  $a$  und  $c$  während den restlichen Simulationen bei denen die Quadratwurzel im AGC vorkommt, folgendermassen gesetzt:  $a = 0.005$  und  $c = 0.1$

In einem letzte Schritt soll nun herausgefunden werden, ob die ganze Regelung auch funktioniert, wenn die im Signalpfad des AGC vorhandene Wurzelberechnung weggelassen wird. Dazu muss aber der Sollwert quadriert werden, da ja nun mit der mittleren Leistung  $P_m(z)$  weitergerechnet werden muss. In Abbildung 7.3-16 sieht man wiederum den Verlauf der Verstärkung im AGC bei einer Verstärkung im Sender von 0.5. Die Regelung scheint nun schneller zu reagieren, wenn man mit Abbildung 7.3-15 vergleicht. Der Empfänger macht hierbei 58 Fehler. Wird nun beim Sender wiederum eine zu grosse Sendeamplitude eingestellt, so resultiert der Verstärkungsverlauf in Abbildung 7.3-17. Hierbei kann jedoch nicht eine zu grosse Verstärkung im Sender eingestellt werden (maximal 7), da mit den eingestellten  $c$  und  $a$  Werten die jeweiligen neuen Ausgangswerte des AGC zuwenig bei der Mittelwertbildung gewichtet werden. Es tritt eine Fehlermeldung im Simulink auf. Das Programm habe Probleme bei der Multiplikation um die Signalleistung  $P(z)$  zu ermitteln.

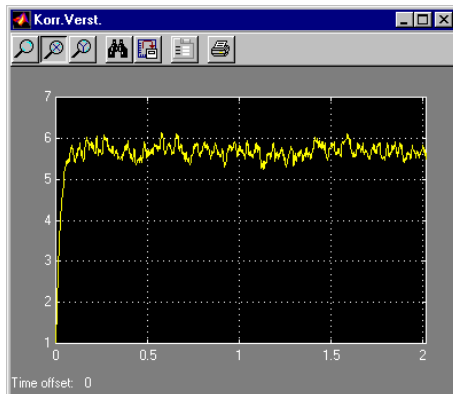


Abbildung 7.3-16 – Verstärkung bei zu kleiner Sende-Amplitude,  $c=0.1$  &  $a=0.005$

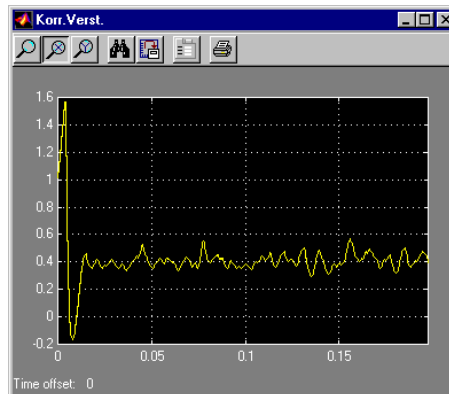


Abbildung 7.3-17 – Verstärkung bei zu grosser Sende-Amplitude,  $c=0.1$  &  $a=0.005$

Um diesen Fehler zu umgehen, muss nun der jeweilig neu berechnete Wert der Leistung mehr ins Gewicht fallen. Hierzu wird  $c = 0.5$  gesetzt. Dies scheint nun, wie in Abbildung 7.3-18 und Abbildung 7.3-19 zu sehen ist, zu funktionieren. Bei der Übertragung treten nun im ersten Fall bei zu kleiner Amplitude - Verstärkung im Sender auf 0.5 gesetzt - 60 Fehler auf. Im zweiten Fall bei zu grosser Amplitude (Abbildung 7.3-18) - Verstärkung im Sender auf 15 gesetzt - treten bei der Übertragung noch 3 Fehler auf.

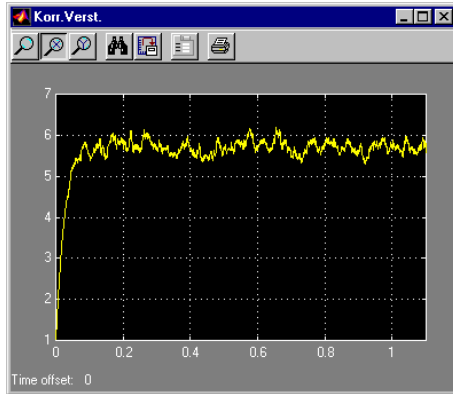


Abbildung 7.3-18 - Verstärkung bei zu kleiner Sende-Amplitude,  $c=0.5$  &  $a=0.005$

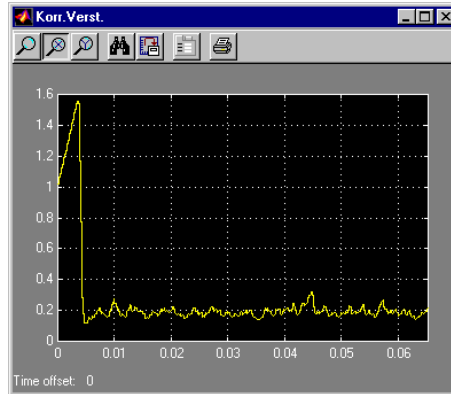


Abbildung 7.3-19 - Verstärkung bei zu grosser Sende-Amplitude,  $c=0.5$  &  $a=0.005$

Wird also die Wurzel im AGC weggelassen, so verwendet man für  $a$  0.005 und für  $c$  0.5. Ob dann diese Werte auch für den DSP gelten, muss dann am DSP selber ausprobiert werden. Verwendet man die Wurzel im AGC, so wird  $a = 0.005$  und  $c = 0.1$  gesetzt. Grundsätzlich kann gesagt werden, dass die Regelung schneller, aber auch heikler ist, wenn die Wurzel weggelassen wird. Im DSP wäre es von Vorteil ohne Quadratwurzel arbeiten zu können, um den Rechenaufwand im Prozessor zu minimieren. Dies jedoch muss dann am DSP selbst ausprobiert werden.

### 7.3.3 Scrambler (Verwürfler)

#### Grundprinzip eines Scramblers/Descramblers

Da ja bei einem realen Modem das Taktsignal irgendwie aus den übertragenen Zeichen herausgefunden muss, verhindert man mit einem Scrambler (Verwürfler) das Auftreten von bestimmten Bitmustern, die wenig Taktinformationen enthalten. Das Prinzip eines Scramblers besteht darin, dass er eine Datenfolge in eine Pseudozufallsfolge umwandelt [2].

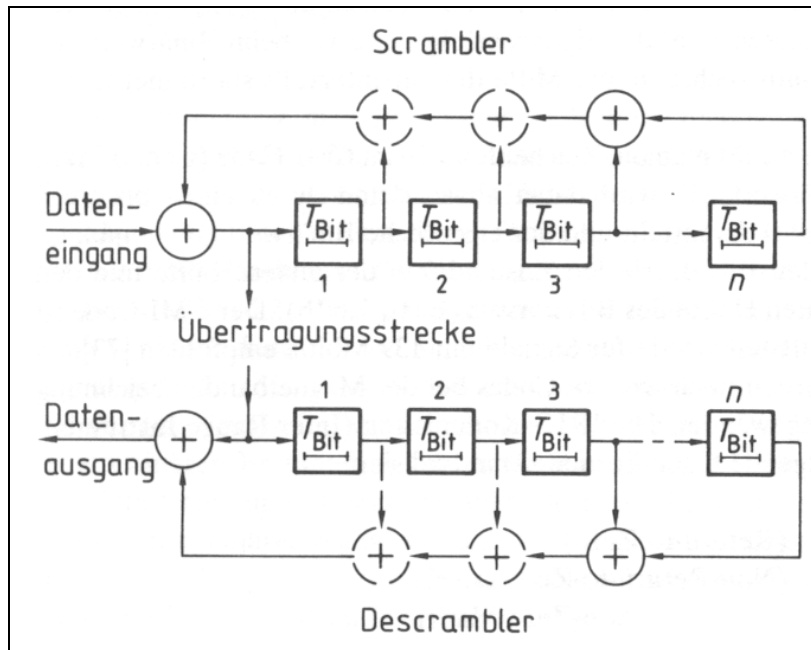


Abbildung 7.3-20 – Grundprinzip des Scramblers & Descramblers

Der Scrambler (Abbildung 7.3-20) besteht aus einem n-stufigen Schieberegister mit Rückkopplung an wenigstens zwei Stellen, von denen aus eine Exklusiv-ODER-Verknüpfung mit dem binären Sendesignal erfolgt. Bei geeigneten Anzapfungen des Schieberegisters wird der abgehende Datenstrom hinreichend statistisch verteilt sein. Insbesondere wird jede sich in kurzen Abständen wiederholende Bitmusterfolge ein Ausgangsmuster erzeugen, dessen Wiederholperiode um den Faktor  $2^n - 1$  länger ist gegenüber dem Eingangsmuster. Der Faktor  $2^n - 1$  sollte eine Primzahl sein. Schieberegisterlängen von  $n = 7, 9, 11, 13, 15, \dots$  sind typisch.

Der Descrambler im Empfänger wandelt diese Pseudozufallsfolge wieder in die ursprünglich gesendete Bitfolge um, d.h. auf der Empfangsseite läuft der umgekehrte Vorgang ab. Das System ist selbstsynchronisierend und benötigt keinen Abgleich, weil dem vom Scrambler erzeugten Bitmuster ein eindeutiges Bildungsgesetz zugrunde liegt.

## Simulation mit Scrambler

Matlab/Simulink besitzt in seiner Bibliothek einen Scrambler und einen Descrambler. In deren Block-Parameter können die jeweiligen Stellen vom Schieberegister angegeben werden, die für die EXOR-Verknüpfung verwendet werden sollen. Zudem kann auch der Initialisierungsvektor angegeben werden. Mit einem zweiten Eingang kann der Scrambler bzw. der Descrambler ein- oder ausgeschaltet werden.

Beim Sender (Abbildung 7.3-21) wird der Scrambler beim Dateneingang vor dem Buffer platziert. Beim Empfänger (Abbildung 7.3-22) wird der Descrambler nach dem Unbuffer, dort wo wieder der kontinuierliche Bitstrom vorhanden ist, platziert.

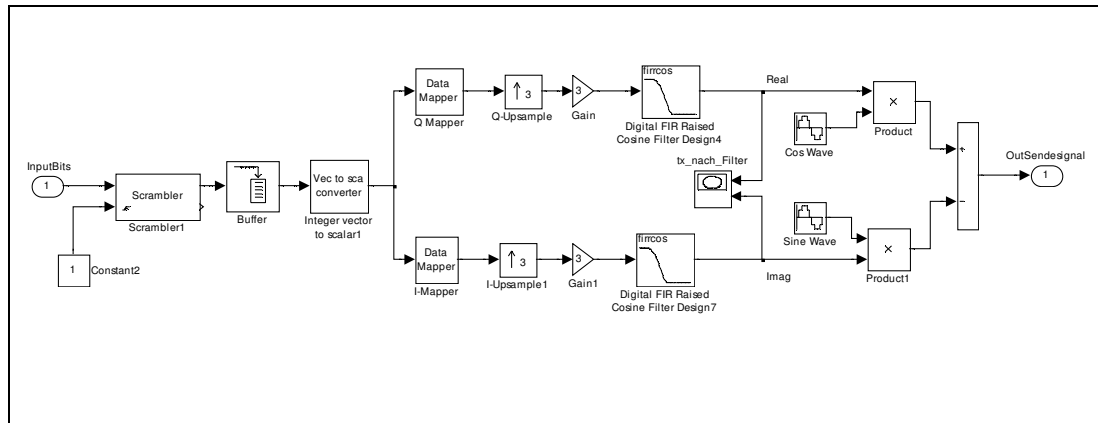


Abbildung 7.3-21 – Sender mit Scrambler

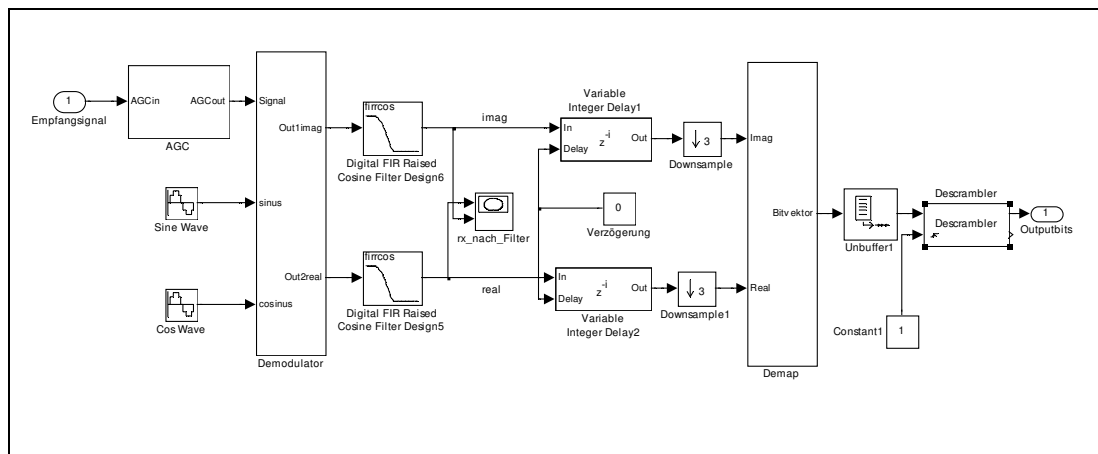


Abbildung 7.3-22 – Empfänger mit Descrambler

Um zu sehen wie nun das gescrambelte Signal aussieht, werden zwei Vergleichsmessungen gemacht. Die eine Messung soll mit dem Scrambler geschehen, die andere ohne. Bei beiden Messungen soll eine konstante Bitfolge übermittelt werden, d.h. es soll immer nur ein Raumpunkt gesendet werden. Dazu muss beim Modem die Data-Source durch ein Sampled-read-from-wksp ersetzt werden. Bei den folgenden Simulationen wird der Vektor (1;1;1;1) gesendet. Dieser Vektor entspricht dem Raumpunkt (3;3).

Bei den folgenden Abbildungen ist jeweils auf der linken Seite die Simulation ohne Scrambler und auf der rechten Seite die Simulation mit Scrambler abgebildet.

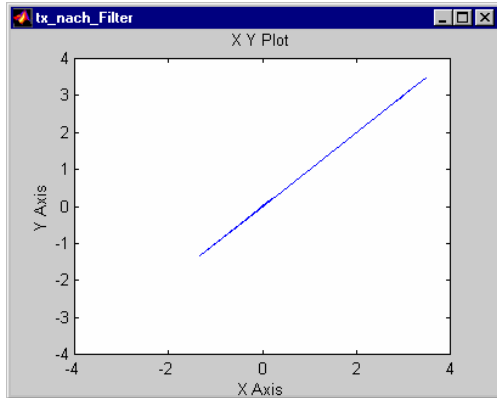


Abbildung 7.3-23 – Sender mit konstantem Raumpunkt, ohne Scrambler

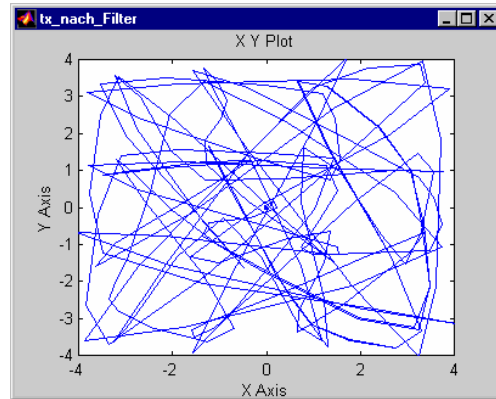


Abbildung 7.3-24 - Sender mit konstantem Raumpunkt, mit Scrambler

Vergleicht man den Sender nach den Filtern ohne Scrambler mit dem mit Scrambler, so stellt man fest, dass ohne Scrambler wirklich nur der Raumpunkt (3;3) gesendet wird, jedoch mit Scrambler alle Raumpunkte gesendet werden. Man stellt auch fest, dass sich das Signal nach einer gewissen Zeit wiederholt und wieder den selben Linien entlang fährt.

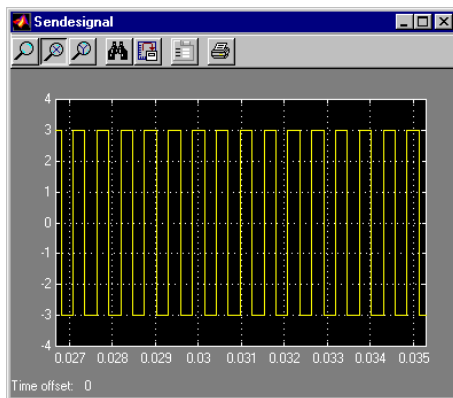


Abbildung 7.3-25 – Sendesignal ohne Scrambler

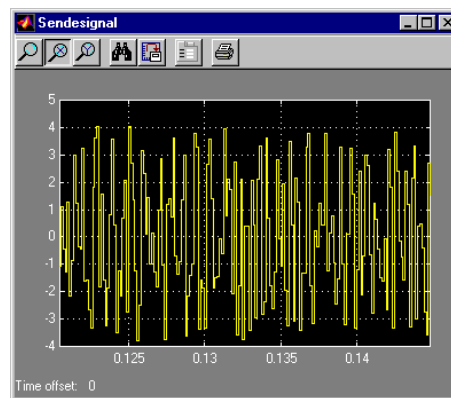


Abbildung 7.3-26 – Sendesignal mit Scrambler

Vergleicht man die beiden gesendeten Signale (Abbildung 7.3-25 und Abbildung 7.3-26), so stellt man fest, dass ohne Scrambler (Abbildung 7.3-25) ein periodisches Signal vorliegt.

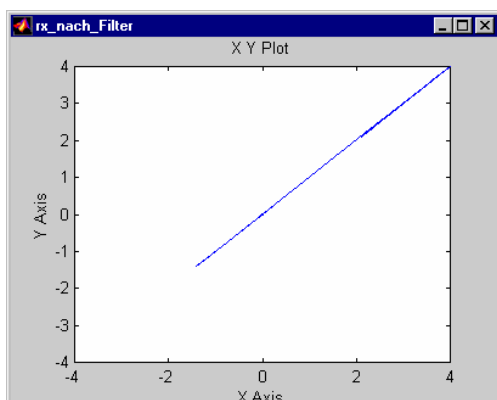


Abbildung 7.3-27 – Empfänger ohne Scrambler

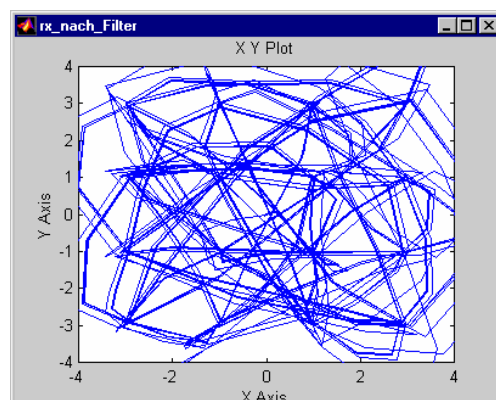


Abbildung 7.3-28 – Empfänger mit Scrambler

Betrachtet man Abbildung 7.3-27 und Abbildung 7.3-28, welche die jeweiligen Raumpunkte im Empfänger darstellen, so stellt man wiederum fest, dass ohne Scrambler wirklich nur der Raumpunkt empfangen wird. Mit Scrambler hingegen fällt auf, dass alle Raumpunkte empfangen werden. Vergleicht man Abbildung 7.3-28 mit dem Sender (Abbildung 7.3-24), so fällt auf, dass die Selbe Struktur der Zeichenreihenfolge zu erkennen ist.

Dies daher, wie schon weiter oben gesagt, weil der Scrambler eine gewisse Periodizität besitzt.

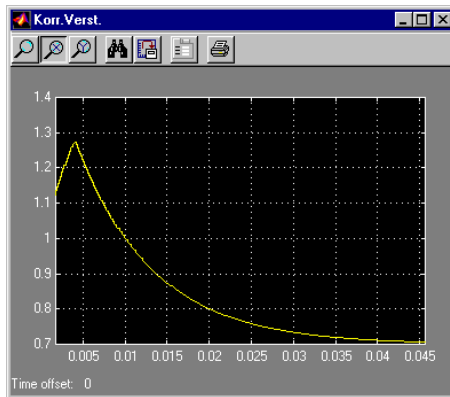


Abbildung 7.3-30 – Verstärkung (AGC) ohne Scrambler

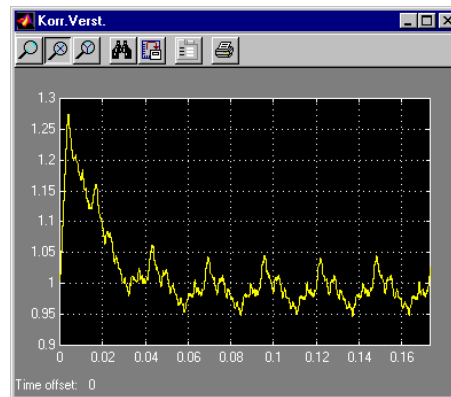


Abbildung 7.3-29 – Verstärkung (AGC) mit Scrambler

Vergleicht man noch die beiden Verstärkungen im AGC (Abbildung 7.3-30 und Abbildung 7.3-29) so fällt auf, dass ohne Scrambler (Abbildung 7.3-25) die Verstärkung fortlaufen abnimmt. Mit Scrambler nimmt sie auch ab, sie schwankt jedoch um einen Wert herum, der sich von dem ohne Scrambler unterscheidet. Ohne Scrambler wird ja nur ein einziger Raumpunkt gesendet, und dieses Signal wird dann im AGC so Verstärkt, dass ein Mittelwert von 2.1 resultiert. Mit Scrambler hingegen werden alle Raumpunkte statistisch gleich viel gesendet, deshalb auch der etwas höhere Wert der Verstärkung, da ja der Mittelwert aller Raumpunkte kleiner als drei ist.

## 7.4 Bitfehlerhäufigkeit in der Übertragung

### Allgemeines

Um eine qualitative Aussage über die Bitfehlerhäufigkeit einer Übertragung machen zu können, wird in Büchern vielfach ein Diagramm dargestellt, in welchem die Bit-Error-Rate, sprich die Bitfehlerhäufigkeit als Funktion des Signal/Rauschabstandes im Übertragungskanal aufgetragen ist. Es sollen nun in einigen Messungen solche Kurven aufgenommen, bzw. ermittelt werden. Dazu wird zum einen das Modem ohne Scrambler betrieben, und zum andern mit Scrambler. Es soll herausgefunden werden, wie sich der Rolloff-Faktor der Root-Raised-Cosine-Filter im Sender und Empfänger auf die Übertragungssicherheit auswirken.

Da es schwierig ist in der Literatur solche Kurven zu finden, sollen die aus den Messungen gewonnenen Kurven mit der, auf einer theoretischen Betrachtung basierenden Kurve verglichen werden.

### Theoretische Betrachtung.

Wird ein Zeichen beim Empfänger ausgewertet, so muss entschieden werden zu welchem Raumpunkt dieses Zeichen nun gehören soll. Wir betrachten nun im folgenden die beiden Raumpunkte (1;1) und (1;-1). Soll nun zwischen diesen beiden Zeichen entschieden werden, so muss eine Entscheidungsschwelle festgelegt werden, da ja dem Signal aufaddiertes Rauschen zu einer Verschiebung des Punktes führen kann. In Abbildung 7.4-1 sind nun diese beiden Punkte links im Koordinatensystem aufgetragen. Rechts davon ist die Wahrscheinlichkeitsdichteverteilung der beiden Zustände 1 und -1 mit der überlagerten Rauschspannung aufgezeichnet. Die Entscheidungsschwelle liegt in diesem Fall bei null, d.h. dort wo der Imaginärteil null ist. Es wird nun also nur der Imaginärteil des Signals betrachtet um einige grundlegende Überlegungen anzustellen, mit welchen man auf die Fehlerwahrscheinlichkeit schliessen kann.

Wird nun z.B. der Punkt (1;-1) übertragen und kommt im Empfänger durch aufaddiertes Rauschen oberhalb der x-Achse zu liegen, so entscheidet man falsch, nämlich dass der Punkt (1;1) übertragen worden ist. Die schraffierte Fläche oberhalb der x-Achse bei der Wahrscheinlichkeitsverteilung entspricht genau der Fehlerwahrscheinlichkeit ein falsches Zeichen zu interpretieren.

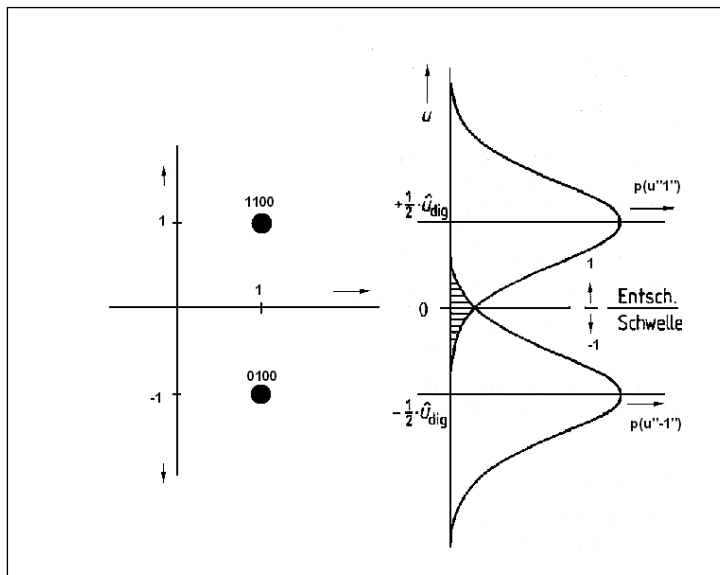


Abbildung 7.4-1 - 2 Raumpunkte und durch Wahrscheinlichkeitsdichte der überlagerten Rauschspannung bei den Zuständen 1 und -1

Da in [2] aber ein übertragenes Zeichen auch nur ein Bit ist, muss man für die Berechnung bei der 16-QAM einige Überlegungen anstellen um auf die tatsächliche theoretische Bitfehlerhäufigkeit schliessen zu können. Jedoch mit der in [2, S. 157] beschriebenen Methode lässt sich hier auf die Zeichenfehlerhäufigkeit in Abhängigkeit vom Signal/Rauschabstand im Übertragungskanal schliessen.

Die Zeichenfehlerhäufigkeit  $p_w$  berechnet sich aus Gleichung 7.4-1 [2].  $\hat{u}_{dig}$  ist in unserem Fall der Abstand zweier Raumpunkte, d.h.  $\hat{u}_{dig}=2$ , und  $U_R$  entspricht dem Effektivwert der Rauschspannung.

$$p_w = \frac{1}{2} \cdot \operatorname{erfc}\left(\frac{\hat{u}_{dig}}{2 \cdot \sqrt{2} \cdot U_R}\right)$$

Gleichung 7.4-1

Anstelle des Verhältnisses  $\hat{u}_{dig}/2 \cdot \sqrt{2} \cdot U_R$  soll nun der auf einen solchen Punkt bezogene Signal/Rauschabstand zur Berechnung der Zeichenfehlerhäufigkeit herangezogen werden. Dazu benötigt man die durchschnittliche Signalleistung. Sie lässt sich bei unserem Raumpunktdiagramm Abbildung 7.4-2 folgendermassen berechnen:

Punkt x:  $P_x = d^2/2 + d^2/2 = d^2$

Gleichung 7.4-2

Punkt y:  $P_y = 9d^2/2 + d^2/2 = 5d^2$

Gleichung 7.4-3

Punkt z:  $P_z = P_y = 5d^2$

Gleichung 7.4-4

Punkt q:  $P_q = 9d^2/2 + 9d^2/2 = 9d^2$

Gleichung 7.4-5

Signalleistung:  $P_s = (P_x + P_y + P_z + P_q)/4 = 5d^2 = 5 \cdot \left(\frac{\hat{u}_{dig}}{2}\right)^2 = \frac{5 \cdot \hat{u}_{dig}^2}{4}$

Gleichung 7.4-6

Erklärung: d ist ja der Peak-Wert der Spannung. Will man die Leistung berechnen, so muss man den Effektivwert haben. Die Leistung des Punktes (1;1) berechnet sich demnach zu  $(d/\sqrt{2})^2$  an einem Einheitswiderstand von  $1\Omega$ , der hier für alle Leistungen gelten soll.

Die Leistung des Rauschens berechnet sich zu

$$P_N = U_R^2$$

Gleichung 7.4-7

Der Signal/Rauschabstand linear dargestellt, ist definiert als

$$S/N = \frac{P_s}{P_N} = \frac{5 \cdot \hat{u}_{dig}^2}{4 \cdot U_R^2}$$

Gleichung 7.4-8

Nach der Rauschspannung aufgelöst erhält man

$$U_R = \sqrt{\frac{5 \cdot \hat{u}_{dig}^2}{4 \cdot S/N}}$$

Gleichung 7.4-9

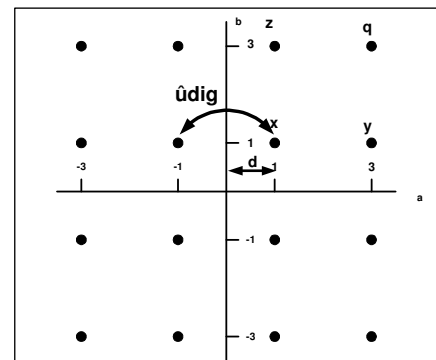


Abbildung 7.4-2 - Raumpunktdiagramm

Setzt man Gleichung 7.4-9 in Gleichung 7.4-1 ein und kürzt was gekürzt werden kann, dann erhält man den Ausdruck

$$p_w = \frac{1}{2} \cdot \operatorname{erfc}\left(\sqrt{\frac{S/N}{10}}\right)$$

Gleichung 7.4-10

Mit Gleichung 7.4-10 lässt sich nun eine Aussage über die theoretische Signalfehlerhäufigkeit in Abhängigkeit des Signal/Rauschabstandes machen.

Das Ziel jedoch ist, eine Aussage über die theoretische Bitfehlerhäufigkeit machen zu können.

Betrachtet man das Raumpunktdiagramm, und nimmt die bestmögliche Verteilung der vier Bits auf die Raumpunkte an, so wäre es doch möglich, dass sich je zwei benachbarte Raumpunkte in ihren Bitmustern um maximal ein Bit unterscheiden. Dies wäre der optimalste Fall und von ihm aus soll bei der folgenden Überlegung gegangen werden.

Gelangt nun also ein Zeichen durch Rauschen auf die andere Seite der Entscheidungsschwelle, so wird es als den benachbarten Raumpunkt interpretiert. Dieser jedoch unterscheidet sich aufgrund unserer vorangegangenen Voraussetzung um nur ein Bit vom ursprünglich gesendeten Raumpunkt. Die drei restlichen Bits werden richtig übertragen.

Wird also bei der Zeichenübertragung ein Fehler gemacht, so verursacht dies bei nur einem von vier im Zeichen enthaltenen Bits einen Bitfehler. Diese Erkenntnis lässt darauf schliessen, dass die Bitfehlerhäufigkeit viermal kleiner ist, als die Signalfehlerhäufigkeit.

Die Bitfehlerhäufigkeit in Abhängigkeit des Signal/Rauschabstandes im Übertragungskanal lässt sich also folgendermassen berechnen:

$$p_e = \frac{p_w}{4} = \frac{1}{8} \cdot \operatorname{erfc}\left(\sqrt{\frac{S/N}{10}}\right)$$

Gleichung 7.4-11

Es ist noch zu bemerken, dass in Gleichung 7.4-11 das Verhältnis S/N linear eingesetzt werden muss.

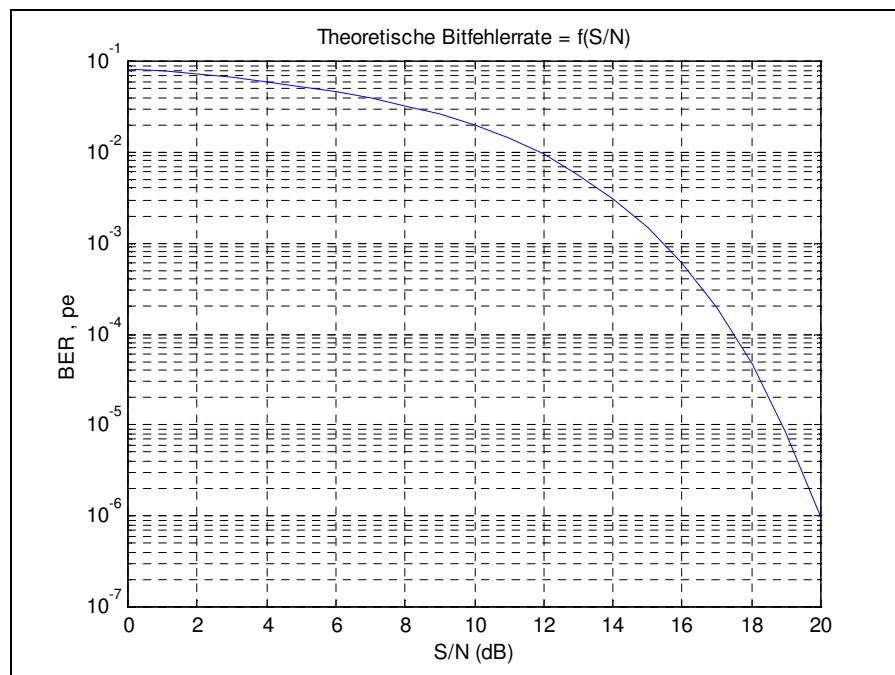


Abbildung 7.4-3 – Theoretische Bitfehlerhäufigkeit in Abhängigkeit vom S/N

## Messaufbau

In Abbildung 7.4-4 ist das Modem wie es zur Ermittlung der anschliessenden Messresultate verwendet wird. In der Leitung ist ein Bandpassfilter ( $f_u=300$  und  $f_o=3300$ ) zur Simulation einer Telefonleitung plziert. Das Rauschen wird dem Signal durch einen AWGN-Kanal (Additive-white-gaussian-noise-Channel) zugeführt. Der Block ist so konfiguriert, dass ihm die Varianz, d.h. die Rauschleistung über einen separaten Eingang zugeführt werden muss. Dies daher, da man im Sender durch die Verstärker vor den Filter die Signalamplitude variieren kann. Das Subsystem (Abbildung 7.4-5) berechnet aus dem eingestellten S/N(dB) und dem Signal die, dem AWGN-Channel zuzuführende Varianz. Die Konstante  $q$  beträgt hier 0.01. Sie wurde empirisch ermittelt.

Die Verzögerung bei der Error-Rate-Calculation muss nun aufgrund der Laufzeit im Bandpassfilter auf 124 gesetzt werden. Um nicht die Fehler, die beim Einstellen der Verstärkung des AGC's auftreten, mitzuzählen, hat man die Möglichkeit die Error-Rate-Calculation an einem Reset Port mit einem Step erst nach einer gewissen Zeit zu aktivieren. Dieser zusätzliche Reset Port ist über das Menu dieses Blockes zu aktivieren.

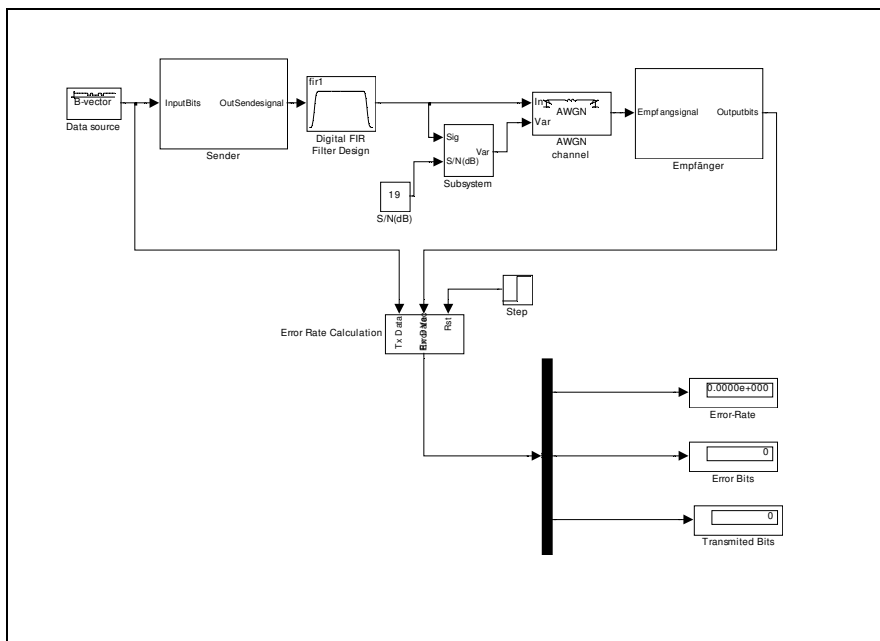


Abbildung 7.4-4 - Modem mit Bandpassfilter und AWGN-Channel

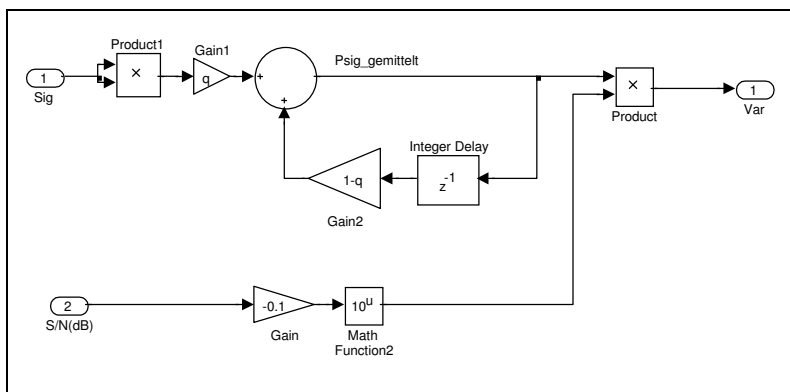


Abbildung 7.4-5 – Subsystem

## 7.4.1 Bitfehlerhäufigkeit ohne Scrambler

### Messdaten

#### Rolloff-Faktor = 0

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	1.21E+04	2.62E-01
3	4.61E+04	1.01E+04	2.18E-01
5	4.61E+04	8.06E+03	1.75E-01
8	4.61E+04	4.99E+03	1.08E-01
11	4.61E+04	2.41E+03	5.24E-02
13	4.61E+04	1.42E+03	3.07E-02
14.5	4.61E+04	8.66E+02	1.88E-02
16	9.41E+04	1071	1.11E-02
18	9.41E+04	589	6.26E-03
19	9.41E+04	441	4.69E-03

#### Rolloff-Faktor = 1

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	1.24E+04	2.65E-01
3	4.61E+04	1.03E+04	2.24E-01
5	4.61E+04	8.35E+03	1.81E-01
8	4.61E+04	4.92E+03	1.07E-01
11	4.61E+04	2.28E+03	4.95E-02
13	4.61E+04	1.16E+03	2.52E-02
14.5	4.61E+04	6.28E+02	1.36E-02
16	9.41E+04	306	6.64E-03
18	9.41E+04	218	2.32E-03
19	1.90E+05	242	1.27E-03

#### Rolloff-Faktor = 0.5

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	1.14E+04	2.57E-01
3	4.61E+04	9.78E+03	2.12E-01
5	4.61E+04	7.69E+03	1.67E-01
8	4.61E+04	4.32E+03	9.37E-02
11	4.61E+04	1.59E+03	3.46E-02
13	4.61E+04	5.92E+02	1.28E-02
14.5	4.61E+04	1.69E+02	3.67E-03
16	1.90E+05	204	1.07E-03
18	7.66E+05	91	1.19E-04
19	3.82E+05	6	1.57E-05

#### Rolloff-Faktor = 0.25

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	1.20E+04	2.60E-01
3	4.61E+04	9.84E+03	2.14E-01
5	4.61E+04	7.53E+03	1.63E-01
8	4.61E+04	4.05E+03	8.79E-02
11	4.61E+04	1.43E+03	3.10E-02
13	4.61E+04	5.36E+02	1.16E-02
14.5	9.41E+04	3.38E+02	3.59E-03
16	1.90E+05	155	8.15E-04
18	1.15E+06	199	1.03E-04
19	1.25E+06	21	1.69E-05

Tabelle 7.4-1 – Messung ohne Scrambler

**Graphische Darstellung**

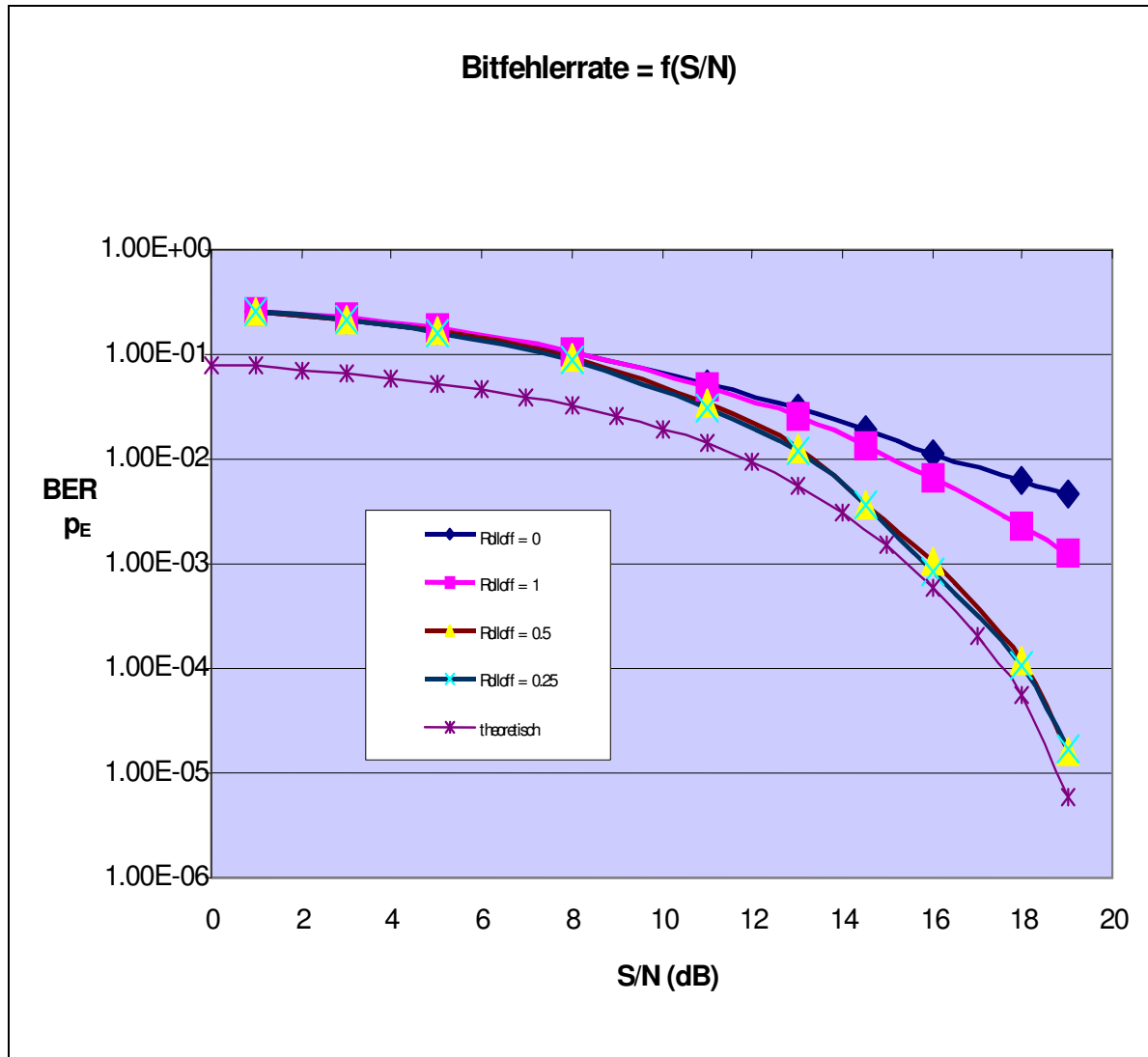


Abbildung 7.4-6 – Messung ohne Scrambler

Vergleicht man die gemessenen mit der theoretischen Kurve, so stellt man fest, dass die gemessenen Kurven schlechter ausfallen als die theoretische. Die beiden Messkurven mit den Rolloff-Faktoren 0.5 und 0.25 liegen sehr nahe beieinander und gelangen bei besserem S/N sehr nahe an die theoretische Kurve.

Ein Grund warum das simulierte Modell nicht so gut ist wie das im letzten Abschnitt hergeleitete, ist weil ja die Voraussetzung gemacht wurde, dass sich zwei benachbarte Raumpunkte um nur ein Bit unterscheiden. Im simulierten Modell ist dies jedoch nicht der Fall. D.h. zwei benachbarte Raumpunkte unterscheiden sich in einigen Fällen um mehr als nur ein Bit, was denn dazu führt, dass die Bitfehlerwahrscheinlichkeit grösser wird, als  $\frac{1}{4}$  der Zeichenfehlerwahrscheinlichkeit.

## 7.4.2 Bitfehlerhäufigkeit mit Scrambler

### Messdaten

#### Rolloff-Faktor = 0

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	2.06E+04	4.46E-01
3	4.61E+04	1.88E+04	4.41E-01
5	4.61E+04	1.63E+04	3.55E-01
8	4.61E+04	1.13E+04	2.45E-01
11	4.61E+04	6.09E+03	1.32E-01
13	4.61E+04	3.29E+03	7.14E-02
14.5	4.61E+04	2.08E+03	4.51E-02
16	4.61E+04	1234	2.68E-02
18	4.61E+04	664	1.44E-02
19	4.61E+04	477	1.04E-02

#### Rolloff-Faktor = 1

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	2.10E+04	4.56E-01
3	4.61E+04	1.94E+04	4.20E-01
5	4.61E+04	1.66E+04	3.61E-01
8	4.61E+04	1.13E+04	2.47E-01
11	4.61E+04	5.79E+03	1.26E-01
13	4.61E+04	3.04E+03	6.59E-02
14.5	4.61E+04	1.67E+03	3.63E-02
16	4.61E+04	911	1.98E-02
18	3.34E+05	2126	6.36E-03

#### Rolloff-Faktor = 0.5

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	2.05E+04	4.45E-01
3	4.61E+04	1.89E+04	4.09E-01
5	4.61E+04	1.59E+04	3.43E-01
8	4.61E+04	9.79E+03	2.12E-01
11	4.61E+04	3.77E+03	8.18E-02
13	4.61E+04	1.33E+03	2.89E-02
14.5	4.61E+04	5.15E+02	1.18E-02
16	9.41E+04	296	3.15E-03
19	2.86E+05	104	3.64E-04

#### Rolloff-Faktor = 0.25

S/N [dB]	Übertragene Bits	Fehler	Bit-Erroro-Rate
1	4.61E+04	2.05E+04	4.44E-01
3	4.61E+04	1.84E+04	3.98E-01
5	4.61E+04	1.54E+04	3.34E-01
8	4.61E+04	9.81E+03	2.13E-01
11	4.61E+04	3.80E+03	8.24E-02
13	4.61E+04	1.43E+03	3.20E-02
14.5	9.41E+04	4.19E+02	9.09E-03
16	9.41E+04	215	2.29E-03
18	5.74E+04	144	2.46E-04
19	5.74E+04	29	5.05E-05

Tabelle 7.4-2 – Messung mit Scrambler

## Graphische Darstellung

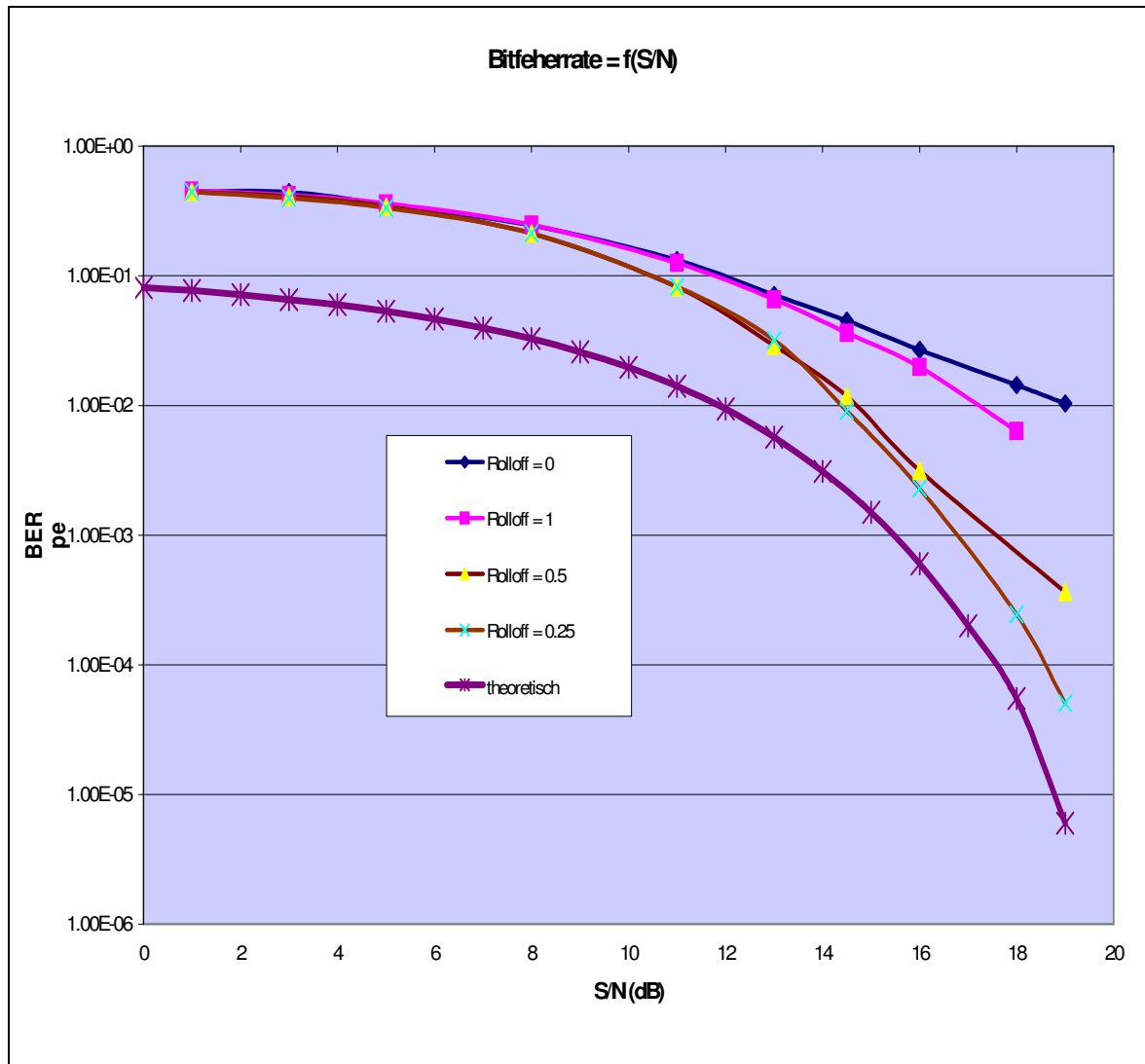


Abbildung 7.4-7 – Messung mit Scrambler

Vergleicht man diese gemessenen Kurven mit denen von Abbildung 7.4-6, so stellt man fest, dass die Messung hier schlechter ausfällt. Dieses Ergebnis, dass diese Messung schlechter ausfällt als die vorangegangene, war zu erwarten. Der Grund liegt darin, dass wenn nun ein falsches Zeichen empfangen wird, und von diesem Zeichen ein Bit nicht so ist wie beim ursprünglich gesendeten, dass sich dieses eine falsche Bit im Descrambler fortpflanzt und einige zusätzliche Bits verfälscht, obwohl diese eigentlich vom Zeichen her richtig erkannt worden wären.

## 8 Realisierung

### 8.1 Aufbau

Der Aufgabenstellung zu Folge ist es gestattet, neben dem Datensignal eine zweite Leitung zur Übertragung eines Synchronisationssignals zu verwenden. Aus diesem Grunde sind nachfolgend zwei Verbindungsleitungen zwischen Sender und Empfänger zu erkennen.

Abbildung 8.1-1 beschreibt die Anordnung der einzelnen Komponenten, wie sie beim Test eingesetzt wurden.

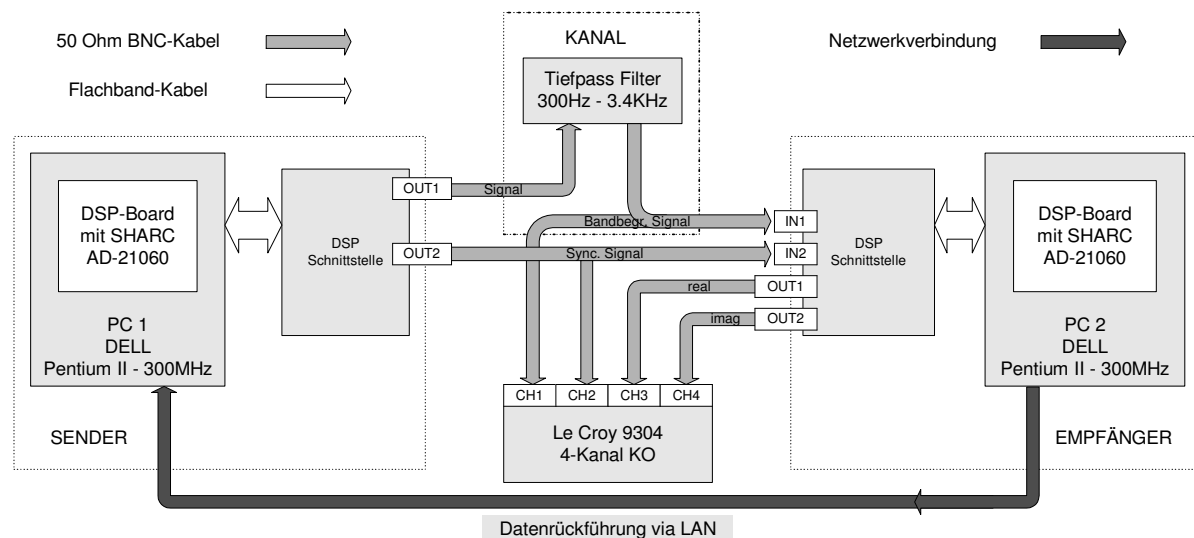


Abbildung 8.1-1 - Systemanordnung

PC1 bildet zusammen mit der eingebauten DSP-Karte und deren externem Anschluss-Adapter die QAM-Sende-Einheit. Das Gegenstück, der QAM-Empfänger wird mit einem identisch ausgestatteten PC (PC2) realisiert. Die Verbindungen (Signal und Synchronisation) erfolgen über 50 Ohm Koax-Kabel. Beim Sender werden nur die Ausgänge zur Übertragung des Daten- und des Synchronisationssignals verwendet. Beim Empfänger hingegen, werden neben den zwei Eingängen für den Empfang des Daten- und Synchronisationssignals zwei Ausgänge für die Visualisierung der aufgesplitteten Inphasen- (real) und Quadratur- (imag) Signale auf einen KO geführt. Daten- und Sync-Signal werden ebenfalls auf dem KO dargestellt. Ein Tiefpassfilter (Variables Filter) glättet das treppenförmige Datensignal am D-/A-Konverter des Senders und begrenzt gleichzeitig das Übertragungsband auf Telefoniebandbreite (300Hz...3.4kHz). Um das Datensignal nicht zu stark zu beschneiden, wurde das Filter auf eine Grenzfrequenz von 3.8 statt 3.4 kHz eingestellt.

## 8.2 Entwicklungsumgebung

### 8.2.1 Hardware

Die Realisierung erfolgte auf der von der HSR zur Verfügung gestellten Infrastruktur, welche sich aus folgenden Komponenten zusammensetzt:

- 2 Personalcomputer der Firma DELL mit Pentium II – 300MHz Prozessor
- 2 DSP-Boards mit SHARC AD-21060 DSP von Analog Devices und Bitsi-AD/DA-Converter-Board von Bittware (1 Board/PC)
- Programmierumgebung: MATLAB-DLL / SHARC-C-Compiler (Assembler)
- Variables Analogfilter für die Bandbegrenzung des Kanals
- 4 Kanal Kathodenstrahl Oszilloskop LE CROY 9340

### 8.2.2 Software

Die DSP-Algorithmen für Sender und Empfänger wurden in C implementiert. Für die Compilierung des Source-Codes wurde der g21k Compiler für die Prozessoren der ADSP-21k-Reihe eingesetzt.

Bei der Durchführung von Grobtests während der Implementation der DSP-Algorithmen, wurde MATLAB mit der SHARC-DLL für die Kommunikation zwischen DSP und PC verwendet.

Die Realisierung des Testprogrammes erfolgte in JAVA. Hierzu wurde ein JDK-Interpreter (V 1.2.2) mit KAWA-Benutzeroberfläche (V 3.21) für die Projektverwaltung verwendet. Dabei wird über das JAVA-Native-Interface auf die Sharc C++-Library zugegriffen, welche Befehle zum Austausch von Daten zwischen DSP und PC zur Verfügung stellt. Für die Kompilation der Native-Files wurde ein Visual-C++-Compiler von Microsoft eingesetzt.

- |                                |   |              |            |         |
|--------------------------------|---|--------------|------------|---------|
| • DSP-Algorithmen in C         | → | Compiler:    | g21k.exe   |         |
| • MATLAB mit SHARC-DLL         |   |              |            |         |
| • Testprogramm mit GUI in JAVA | → | Interpreter: | JDK        | V 1.2.2 |
|                                | → | Editor:      | KAWA       | V 3.21  |
|                                | → | Native C++:  | Visual C++ |         |

## 8.3 Definitionen und Protokoll

### 8.3.1 Taktraten

#### Samplingfrequenz

Der Sender arbeitet mit einer Takt-, bzw. Samplingfrequenz ( $f_s$ ) von 7'200 Hz. Der Empfänger hingegen, wird mit einem Vielfachen dieser Taktfrequenz betrieben, da das zur Zeit verwendete Synchronisationsverfahren eine Überabtastung des Empfangssignals erfordert. Die Verarbeitung jedoch erfolgt mit der Rate von 7'200 Hz.

#### Symbolfrequenz

Die Symbolrate ( $f_D$ ) von 2'400 Baud bzw. 2'400 Hz beträgt somit ein Drittel der Taktfrequenz, was zu bedeuten hat, dass nur jedes 3. Sample eine Information enthält.

#### Trägerfrequenz (Carrier)

Die Frequenz des Trägersignals ( $f_c$ ) beträgt 1'800 Hz, was einem Viertel der Taktfrequenz entspricht. Daraus folgt, dass eine Periode des Sinus-, bzw. Cosinus-Trägers, wie aus Gleichung 8.3-1 ersichtlich, viermal ( $N=4$ ) abgetastet wird.

$$N = \frac{T_{\text{Carrier}}}{T_{\text{Sample}}} = \frac{1/f_c}{1/f_s} = \frac{1/1'800\text{Hz}}{1/7'200\text{Hz}} = 4$$

$f_c$ : Trägerfrequenz  
 $f_s$ : Samplingrate

Gleichung 8.3-1

#### Bitrate

Die 16-QAM Modulation ermöglicht eine Übermittlung von 4 Datenbit pro Raumpunkt bzw. pro Symbol (Gleichung 8.3-2). Bei einer Symbolrate von 2'400 Hz ergibt dies eine Bitrate von  $4 \cdot 2'400 = 9'600$  Hz bzw. 9'600 Bit/s.

$$n_{\text{Raumpunkte}} = 2^{n_{\text{Bits}}} \Rightarrow n_{\text{Bits}} = \frac{\log_{10}(n_{\text{Raumpunkte}})}{\log_{10}(2)} = 4$$

Gleichung 8.3-2

### 8.3.2 Synchronisation

#### Frequenzsynchronisation

Als erster Schritt der Synchronisation muss die Frequenz des Empfängers mit der des Senders identisch sein. Dies erfolgt beim Empfänger unabhängig vom Zustand des Senders fortlaufend anhand eines separat geführten Synchronisationssignals. Dieses Signal wird mit der effektiven Abtastrate von 72'000 Hz (Vielfaches von  $f_c$  beim Empfänger) abgetastet.

Aufgrund der Samplingfrequenz des Senders von 7'200 Hz beträgt die Frequenz des bipolar rechteckigen Synchronisationssignals 3'600 Hz ( $1/2 \cdot f_c$ ), da der Sender pro Sample nur einen Pegelwechsel ausgeben kann.

Der Übertragungskanal verzögert das Empfangssignal gegenüber dem Sendesignal (Filter, Leitung, usw.), wodurch das Sende- und Synchronisationssignal nicht mehr in Phase liegen. Für die folgenden Synchronisationsschritte muss daher das Datensignal selbst als Referenz genommen werden.

## Träger(phasen)synchronisation

Die Trägersynchronisation erfolgt in zwei Schritten.

Zuerst wird eine Grobsynchronisation vorgenommen, bis alle demodulierten, d.h. alle mit dem jeweiligen Trägersignal (Cosinus für Inphase- und Sinus für Quadraturkomponente) multiplizierten Samples positive Werte aufweisen. Danach folgt eine Feinsynchronisation für die Phasenkalibrierung (Maximum-, bzw. Nullstellendetektion). Die Feinsynchronisation erfordert ein konstantes periodisches Sinus-, bzw. Cosinussignal.

Das fortlaufende Aussenden des Raumpunktes  $1111_{\text{Bin}}$  bzw.  $15_{\text{Dez}}$  (siehe Abbildung 8.3-1), wurde als Referenzsignal für die Trägerphasensynchronisation gewählt, da die Punkt-Koordinaten (real 3/imag 3) eine maximale Aussteuerung der Inphase-/Quadratur-Anteile und somit ein sinusförmiges Datensignal mit grosser Amplitude liefern.

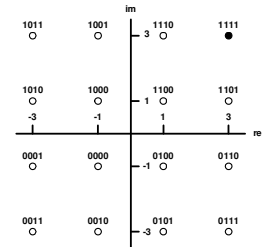


Abbildung 8.3-1

## Symbolsynchronisation

Bis anhin wurde jedes Sample beachtet und abgefragt, obwohl nur jedes dritte relevant wäre. Auch wurde kein Datenscrambling (wird im nächsten Kapitel beschrieben) vorgenommen. Von jetzt an werden die Datenbits beim Sender verwürfelt (Scrambler) und beim Empfänger "entwürfelt" (Descrambler), was verhindern soll, dass zwei identische Symbole nacheinander auf der Datenleitung auftreten können. Der Hauptgrund für das Scrambling ist, die Taktinformation des Datensignals zu jedem Zeitpunkt zu gewährleisten, was bei Periodischen Signalen nicht immer möglich ist.

Um nun die bedeutenden Samples zu ermitteln, wird zyklisch eine länger anhaltende Folge von 8 frei definierten Raumpunkten gesendet. Nach erstem Erkennen der Synchronisationsfolge ist der Empfänger synchronisiert und verarbeitet nur noch die relevanten Symbole.

Als Symbolsynchronisationsfolge wurde die kreuzweise Ausgabe der vier äusseren Eckpunkte und der 4 inneren Mittelpunkte gewählt (siehe Abbildung 8.3-2).

- $1100|0000|1000|0100|1111|0011|1011|0111|_{\text{Bin}}$
- $12|0|8|4|15|3|11|7|_{\text{Dez}}$

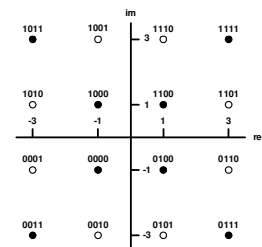


Abbildung 8.3-2

## Rahmensynchronisation

Die Datenübertragung erfolgt in Paketen, was heisst, dass eine gewisse Anzahl Symbole zusammengefasst und mit einem Kopf (Header) versehen werden. Der Kopf beinhaltet in diesem Anwendungsfall eine ebenfalls frei gewählte, über 8 Symbole andauernde Startsequenz zur Erkennung und eine Längeninformation zur Anzahl der nachfolgenden Zeichen. Im Unterschied zu den vorherigen Vorgehen wird hier die Startsequenz nur ein einziges Mal pro Paket gesendet.

Als Rahmensynchronisationsfolge wurde die Ausgabe der acht kreisförmig angeordneten Raumpunkte mit gleichem Abstand vom Mittelpunkt der Ebene gewählt (Abbildung 8.3-3).

- $|0001|0010|0101|0110|1101|1110|1001|1010|_{\text{Bin}}$
- $|1|2|5|6|13|14|9|10|_{\text{Dez}}$

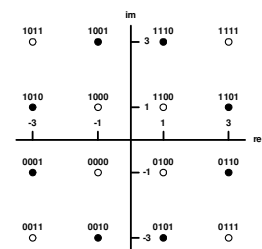


Abbildung 8.3-3

### 8.3.3 Datenübertragung

#### Datenpakete

Die Datenübertragung zwischen Sender und Empfänger erfolgt blockweise. Ein Block beinhaltet jeweils einen Header von 8 Symbolen und 400 Nutzzeichen bzw. 800 Symbole. Die ersten 8 Symbole im Header bilden dabei die Blockerkennungssequenz, auch Rahmensynchronisationssequenz genannt, die restlichen 8 Symbole die Anzahl der nachfolgenden gültigen Nutzzeichen im Block.

Die Grösse eines Nutzdatenblocks wurde so gewählt, dass ein Kommunikationsbuffer (Kapitel 8.3.4) von 400 Zeichen auf einmal in kompakter Weise übertragen werden kann.

Wichtig zu erwähnen ist, dass auch bei einer Nutzdaten-Zahl von weniger als 400 Zeichen pro Block, der ganze Kommunikationsbuffer übertragen bzw. gelesen wird. Damit also die maximale Bitübertragungsrate ausgeschöpft werden kann, sollte daher der ganze Block von 400 Zeichen bzw. der ganze Kommunikationsbuffer ausgenutzt werden. Abbildung 8.3-4 zeigt den Aufbau eines Datenblocks oder Pakets.

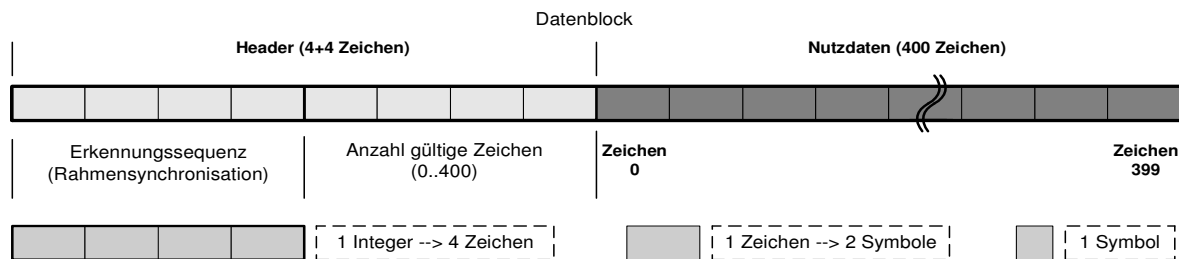


Abbildung 8.3-4 - Datenblock zwischen Sender und Empfänger

#### Übertragungsrate

Die implementierte QAM-Verbindung erreicht eine theoretische Übertragungsrate von **9'600 Bit/s**, welche sich nach Gleichung 8.3-3 errechnen lässt.

$$Bitrate_{Theoretisch} = 4Bit \cdot f_D = 4 \cdot 2'400Hz = 9'600Bit / s$$

Gleichung 8.3-3

Da aber zu den 400 Nutzzeichen zusätzlich noch ein Header einer Länge von 8 Zeichen hinzugefügt wird, reduziert sich die Übertragungsrate bei Vollaustattung des Kommunikationsbuffers nach Gleichung 8.3-4 auf **9'411.76 Bit/s**.

$$Bitrate_{Maximal} = 4Bit \cdot f_D \cdot \frac{Anz_{NutzzeichenMax}}{Anz_{Headerzeichen} + Anz_{NutzzeichenMax}} = 4 \cdot 2'400Hz \cdot \frac{400}{8 + 400} = 9'411.76Bit / s$$

Gleichung 8.3-4

Im schlechtesten Falle, mit einer Bufferauslastung von einem Zeichen, käme man mit der Rechnung nach Gleichung 8.3-5 auf eine Übertragungsrate von **23.52 Bit/s**.

$$Bitrate_{Minimal} = 4Bit \cdot f_D \cdot \frac{Anz_{NutzzeichenMin}}{Anz_{Headerzeichen} + Anz_{NutzzeichenMax}} = 4 \cdot 2'400Hz \cdot \frac{1}{8 + 400} = 23.52Bit / s$$

Gleichung 8.3-5

### 8.3.4 Softwareschnittstellen

---

Beim SHARC DSP-Board besteht die Möglichkeit, von Seite der PC-Applikation, globale Variablen oder Arrays direkt zu beschreiben bzw. auszulesen. Kommunikationsvariablen müssen global definiert werden, da deren Speicheradressen im DSP-Memory vom Start des Programms an fix sein müssen. Lokale Variablen (Funktionsvariablen) werden im Stack abgelegt und haben daher keine feste Adresse.

Für die Kommunikation zwischen dem PC und dem DSP stehen folgende zwei Interfaces zur Verfügung, die im Lieferumfang eines SHARC-Boards enthalten sind:

- **MATLAB-DLL**

Die Matlab-DLL enthält Befehle, um verschiedene Datentypen einzeln oder in Arrayform auf den DSP zu schreiben bzw. vom DSP zu lesen. Dabei kann der Name der gewünschten globalen DSP-Variable direkt angegeben werden.

- **C++-Library**

Die C++-Library umfasst die identischen Befehle in Form von Funktionen. Hier ist aber jeweils die Adresse einer gewünschten DSP-Variable mit einer zusätzlichen Funktion zu ermitteln.

Für den Grobtest des Senders und Empfängers wurde MATLAB verwendet. Die endgültige Testanwendung, welche in Java realisiert ist, braucht hingegen die C++-Library, welche über das JAVA-Native-Interface (plattformgebunden) angesprochen wird.

#### Sender

---

Aus Effizienzgründen werden auf dem DSP (Sender & Empfänger) 4 Zeichen in einen Integer gepackt, wodurch sich die Buffergrösse der Nutzdaten von 400 auf 100 reduziert.

Für den Daten-Download (PC→DSP) stellt der Sender einen Datenbuffer mit der maximalen Länge von 100 Integerwerten, sowie eine zugehörige Längenvariable zur Verfügung. Da die Verarbeitung im Sender dank DMA-Ausgabe (Kapitel 8.4.1) blockweise erfolgt, reicht ein Datenbuffer aus. Der Download muss jedoch innerhalb einer bestimmten Zeit erfolgen, um ein effizientes Arbeiten des Senders zu ermöglichen.

Bevor der Sendebuffer des DSP von der Anwendung (PC) beschrieben werden kann, muss die Längenvariable zurückgelesen werden. Beträgt deren Inhalt **-1**, so ist der Buffer bereit und kann beschrieben werden. Ist sie hingegen grösser oder gleich null, ist der Buffer möglicherweise in Verarbeitung und darf nicht beschrieben werden. Nach dem Beschreiben muss die Längenvariable gesetzt werden (**0...400**), wodurch der Buffer DSP-seitig zur Verarbeitung freigegeben wird. Unmittelbar nach der Verarbeitung wird die Längenvariable vom DSP-Programm jeweils wieder auf **-1** gesetzt und der Buffer somit für den nächsten Download freigegeben.

#### Schreibvorgang:

- |    |   |   |  |                                       |
|----|---|---|--|---------------------------------------|
| 1. | Zurücklesen von <b>dataCount</b>        | → | <b>-1</b>                                  | Buffer kann beschrieben werden        |
|    |   | → | <b>0..400</b>                              | Buffer in Verarbeitung (anz. Zeichen) |
| 2. | Schreiben des Buffers <b>dataBuf</b>    | → | 0..400 Zeichen bzw. 0..100 Integer möglich |                                       |
| 3. | Setzen der Bufferlänge <b>dataCount</b> | → | Zahl der geschriebenen Zeichen             |                                       |

Für die Tests in **MATLAB** wurden folgende m-Files (Tabelle 8.3-1) verwendet:

File	Aufruf	Bereich (x)	Auswirkungen
send.m	<b>send(x)</b>	0..4294967296	Buffer füllen mit x x=Integerwert des Buffers → 8 Symbole/Integerwert
send1.m	<b>send1(x)</b>	0..15	Buffer füllen mit xxxxxxxx x=Symbol bzw. Raumpunkt → 8 Symbole/Integerwert
send2.m	<b>send2(x)</b>	0..4294967296	1. Element mit x beschreiben (Integer) Rest mit 0 beschreiben (Integer)
send3.m	<b>send3(x)</b>	x -> keine Bedeutung	Buffer wird mit Integerwerten von 100..0 in absteigender Reihenfolge gefüllt → 8 Symbole/Integerwert

Tabelle 8.3-1

## Empfänger

Beim Empfänger werden ebenfalls 4 Zeichen (8 Symbole) in einem Integer vereint. Im Gegensatz zum Sender werden hier jedoch zwei Buffer mit zugehöriger LängenvARIABLE für den Daten-Upload benötigt, da die Verarbeitung sampleweise (einzeln) vonstatten geht. So wird gewährleistet, dass der eine Buffer ausgelesen werden kann, während der andere vom DSP bearbeitet wird. Die maximale Grösse der beiden Buffer beträgt ebenfalls 400 Zeichen bzw. 100 Integerwerte.

Vor dem Uploaden eines Buffers muss zuerst die zugehörige LängenvARIABLE abgefragt werden. Beträgt deren Wert **-1**, so ist der Buffer leer oder gerade in Bearbeitung. Liegt der Wert jedoch zwischen **0** und **400**, kann der Upload erfolgen. Nach Abschluss des Lesevorgangs, muss die LängenvARIABLE auf **-1** gesetzt werden, wodurch der jeweilige Buffer zurückgesetzt wird.

### Lesevorgang

1. Zurücklesen von **dataCountx** → **0..400** Buffer kann gelesen werden (anz. Z.)  
→ **-1** Buffer in Verarbeitung
2. Lesen des Buffers **dataBufx** → **0..400** Zeichen bzw. **0..100** Integer möglich
3. Setzen der Bufferlänge **dataCount** → **-1** Buffer Freigabe

Folgende MATLAB-Befehle wurden zum Auslesen der Daten beim Empfänger verwendet:

- `c0 = sharc('ulint','dataCount0') | d0 = sharc('ulints','dataBuf0',100) | sharc('dlint','dataCount0',-1)`
- `c1 = sharc('ulint','dataCount1') | d1 = sharc('ulints','dataBuf1',100) | sharc('dlint','dataCount1',-1)`

## Kommunikations-Variablen

Tabelle 8.3-2 erläutert zusammenfassend die Bezeichnungen der Variablen bzw. Adressen für die Datenkommunikation (PC ↔ DSP) auf den verschiedenen Software-Ebenen.

globale DSP-Variable (C-Code)	MATLAB Variablen (Adresse)	DLL Variablen (Adressen)
<b>Sender (1 Buffer)</b>		
dataBuf[100]	'dataBuf'	'_dataBuf'
dataCount	'dataCount'	'_dataCount'
<b>Empfänger (2 Buffer)</b>		
dataBuf0[100]	'dataBuf0'	'_dataBuf0'
dataCount0	'dataCount0'	'_dataCount0'
dataBuf1[100]	'dataBuf1'	'_dataBuf1'
dataCount1	'dataCount'	'_dataCount1'

Tabelle 8.3-2

## 8.4 Sender

Nach Definition arbeitet der Sender mit einer Takt- bzw. Samplingfrequenz von 7'200 Hz und einer Symbolrate von 2'400 Baud. Die Frequenz des Trägersignals beträgt 1'800 Hz. Die 16-QAM Modulation ermöglicht eine Übermittlung von 4 Datenbit pro Raumpunkt, bzw. pro Symbol.

### 8.4.1 Datenorganisation

#### Zeichengrösse

Die Datenübermittlung zwischen Anwendung und Modem erfolgt grundsätzlich in Form von ASCII-Werten. Da sich mit 8 Bit alle gängigen Zeichen (Charakter) darstellen lassen, hat man sich für eine Zeichengrösse bzw. Wortbreite von 8 Bit entschieden. Mit 4 Datenbits pro Raumpunkt kann also ein Zeichen mit zwei Raumpunkten übertragen werden.

#### Speichernutzung

Die Datenverarbeitung auf dem ADSP-21060 erfolgt unabhängig vom Datentyp mit Ausnahme von Float mit einer Breite von 32-Bit.

Um nun den Speicher auf dem DSP-Chip besser auszunutzen und eine effizientere Echtzeitverarbeitung zu gewährleisten, werden nun 4 Zeichen in einen 32-bit Integerwert verpackt. Somit können gleichzeitig 8 Symbole untergebracht werden. Wichtig zu erwähnen ist dabei, dass die Verarbeitung eines solchen Integers von links nach rechts abläuft. Das Zeichen oder Symbol ganz links wird also zuerst gesendet.

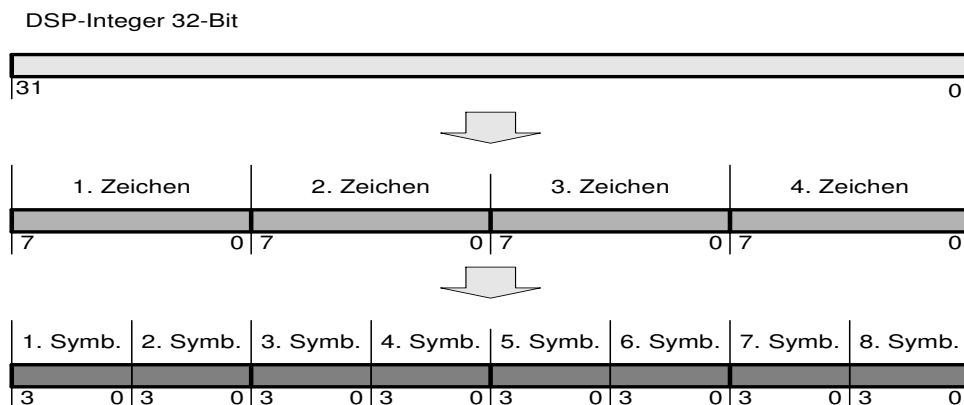


Abbildung 8.4-1 - Speicherausnutzung

#### DMA-Betrieb

Um garantiert genug Rechenleistung für die Datenverarbeitung zur Verfügung zu haben, erfolgt das Aussenden der Symbole bzw. Samples mittels DMA (Direct Memory Access). Durch diese Methode wird die Ausgabe der Samples im Hintergrund über einen separaten, aber im DSP-Chip integrierten, DMA-Controller abgewickelt, wodurch der Prozessor für andere Aufgaben freigehalten wird.

Beim DMA-Betrieb ist es notwendig, mindestens zwei DMA-Buffer zu installieren, um eine fehlerfreie und fließende Sampleausgabe zu gewährleisten. In dieser Anwendung wurden zwei Buffer implementiert, die abwechslungsweise ausgegeben werden (Abbildung 8.4-2). Beim automatischen Wechsel des DMA-Buffers wird ein Interrupt ausgelöst, wodurch die Verarbeitung der Daten und die Neubeschreibung des freien Buffers gestartet wird. Dieser Vorgang darf die Dauer einer Buffer-Ausgabe nicht übersteigen. Zudem muss zwischen der Verarbeitung mit dem Beschreiben des freien DMA-Buffers und dem nächsten DMA-Interrupt noch genügend Zeit für das Neubeschreiben des Datenbuffers übrig bleiben. Das heisst, das Downloaden der Sendedaten zum DSP sollte nach der Verarbeitung und vor dem nächsten DMA-Interrupt erfolgen.

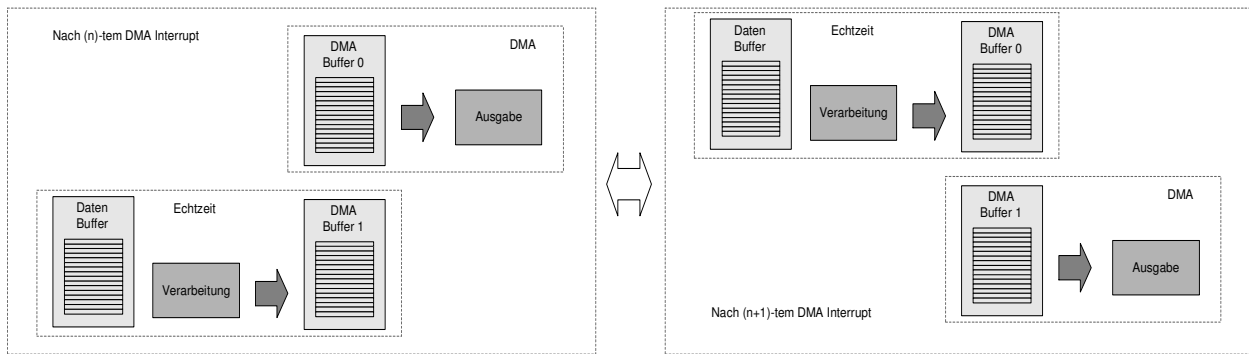


Abbildung 8.4-2 - DMA-Ausgabe und Neubeschreibung

## Buffergrößen

Wie bereits beschrieben, werden die Daten in Blöcken von maximal 400 Nutzzeichen oder 800 Symbolen übertragen. Ein Buffer für die Kommunikation zwischen PC und DSP kann 100 Integer und somit einen ganzen Block von 400 Nutzzeichen aufnehmen.

Der DMA-Buffer für die Sampleausgabe ist rund 24 mal grösser als der Kommunikations-Buffer, da für jeden Raumpunkt und jeden nichtrelevanten Zwischenwert ein Sample ausgegeben werden muss. Die genaue Grösse eines DMA-Buffers errechnet sich nach Gleichung 8.4-1.

Folgende Zusammenhänge werden miteinander berechnet:

- $N_{\text{Zeichen/Integer}}$  : Anzahl Zeichen pro Integerwert : 4
- $N_{\text{Symbol/Zeichen}}$  : Anzahl Raumpunkte (Symbole) pro Zeichen: 2
- $N_{\text{Samples/Symbol}}$  : Anzahl Samples pro Raumpunkt (nur 1. ist relevant): 3
- $N_{\text{Block}}$  : Maximale Blockgrösse: 400 Zeichen / 100 Integer / 800 Symb.
- $N_{\text{Header}}$  : Headergrösse: 8 Zeichen / 2 Integer / 16 Symbole

$$N_{\text{DMA}} = (N_{\text{Block(Int)}} + N_{\text{Header(Int)}}) \cdot N_{\text{Zeichen/Integer}} \cdot N_{\text{Symbol/Zeichen}} \cdot N_{\text{Sample/Symbol}} = 2400 + 48$$

Gleichung 8.4-1

## 8.4.2 Synchronisation

### Synchronisationssignal

Bei dieser Realisierung ist es gestattet, die Samplingfrequenz, oder ein Trägersignal getrennt vom Übertragungssignal über eine separate Leitung zu führen, um eine komplizierte Trägerrückgewinnung am Empfänger umgehen zu können. In unserem Falle wird ein Signal zur Synchronisation der Sampling-Frequenz übertragen, wodurch ein exakter Frequenzabgleich zwischen Sender und Empfänger möglich wird.

Das Synchronisationssignal ist als bipolares Rechtecksignal ausgelegt, welches mit jedem ausgesendeten Sample seine Polarität wechselt bzw. jeweils eine Flanke aufweist. Durch diese Gegebenheit entsteht ein Rechtecksignal mit der halben Samplingfrequenz von 3'600Hz (7'200Hz/2). Der Pegel wurde so gewählt, dass der DA-Wandler etwa bis zu zwei Drittel angesteuert wird.

Da die Verarbeitung DMA-bedingt blockweise erfolgt und ein DMA-Buffer-Element die Samples beider Kanäle (Ch1 & Ch2) vereint, muss das jeweilige Synchronisations- und Daten-Sample beim Schreiben des aktuellen DMA-Buffers miteinander verknüpft werden. Ein DMA-Element ist 32 Bit breit, wovon die oberen 16 Bits dem 1. Kanal und die unteren 16 Bits dem 2. Kanal zugewiesen werden.

## Träger(phasen)synchronisation

Zur Träger- und Trägerphasensynchronisation wird beim Start des DSP-Algorithmus eine bestimmte Zeit lang der Raumpunkt 1111<sub>BIN</sub> (3/3) ausgesendet, woraus auf der Datenleitung ein sinusförmiges Signal resultiert. Während diesem Vorgang, muss es dem Empfänger möglich sein, sich phasengleich auf den Träger zu synchronisieren. Zur Festlegung der Synchronisationszeit wurde ein Zähler (scrCnt) eingebaut, der inkrementiert wird, sobald eine DMA-Buffer-Verarbeitung abgeschlossen ist. Gleichung 8.4-2 zeigt die Berechnung der effektiven Synchronisationsdauer.

$$t_{Sync} = t_{Sample} \cdot N_{DMA} \cdot s_{Cnt} = \frac{1}{f_S} \cdot N_{DMA} \cdot s_{Cnt}$$

$N_{DMA}$ : Länge eines DMA Buffers (2'400+48)  
 $f_S$ : Samplingfrequenz (7'200 Hz)  
 $s_{Cnt}$ : Counterschwelle (6)

### Gleichung 8.4-2

Mit den bekannten Grössen ergibt sich für die Trägerphasensynchronisation eine Dauer von 2.04 Sekunden und eine Symbolzahl von 14'688.

## Symbolsynchronisation

Nach der Trägersynchronisation folgt die Symbolsynchronisation. Während dieser Phase wird pausenlos eine vordefinierte Folge gesendet (siehe Kapitel 8.3.2), welche sich nach 8 Symbolen wiederholt. Die Zeitdauer ist vom gleichen Zähler (scrCnt) abhängig wie die Dauer der Trägerphasensynchronisation. Hierbei wird für die Zeitberechnung die Differenz dieser und der letzten Counterschwelle eingesetzt. Die Synchronisationsdauer lässt sich schliesslich aus Gleichung 8.4-3 berechnen. Spätestens nach Ablauf dieser Zeit, sollte der Empfänger symbolsynchron sein und nur noch die relevanten Symbole (jedes 3. Sample) interpretieren.

$$t_{Sync} = t_{Sample} \cdot N_{DMA} \cdot s_{Cnt} = \frac{1}{f_S} \cdot N_{DMA} \cdot d_{sCnt}$$

$N_{DMA}$ : Länge eines DMA Buffers (2'400+48)  
 $f_S$ : Samplingfrequenz (7'200 Hz)  
 $d_{sCnt}$ : Schwellendifferenz (8-6=2)

### Gleichung 8.4-3

In obige Gleichung eingesetzt, erhalten wir für die Symbolsynchronisation eine Dauer von 680 Millisekunden und eine Symbolzahl von 4'896.

## Rahmensynchronisation

Nach den oben erwähnten einmaligen Synchronisationsphasen erfolgt die eigentliche Datenverarbeitung, worin die Rahmensynchronisation eingebettet ist. Mit der Rahmensynchronisation ist die Datenblockerkennung gemeint, die zusammen mit der Blockgrösse als Header den Nutzdaten vorausgeschickt wird. Wie im Kapitel 8.3.2 bereits erwähnt, besteht ein Header aus einer vordefinierten Erkennungssequenz von 8 Symbolen und der Anzahl der nachfolgenden Nutzzeichen. Für die Anzahl sind ebenfalls 8 Symbole reserviert, die jedoch für eine maximale Zeichenzahl von 400 nicht vollständig ausgenutzt werden.

Die Daten zur Symbolsynchronisation (48 Elemente) werden jeweils vor den Nutzdaten (2'400 Elemente) in den DMA-Buffer eingetragen.

## 8.4.3 Datenverarbeitung

Unter der Datenverarbeitung ist die Verarbeitung eines Datenblocks zu verstehen. Die Daten des aktuellen Interfacebuffers (PC→DSP) werden hierbei verwürfelt (Scrambler) und in die einzelnen Symbole (Raumpunkte) aufgesplittet (1 Integer → 8 Symbole). Diese werden anhand einer Tabelle dekodiert, wodurch die zwei zugehörigen Koordinatenpunkte ihrer Position in der komplexen Ebene resultieren. Jeder der beiden Koordinatenpunkte (Realteil/Imaginärteil) wird nun mit dem entsprechenden Träger (Cos/Sin) moduliert und gefiltert. Die so erhaltene Quadraturkomponente (Imaginärteil) wird von der Inphasenkomponente (Realteil) subtrahiert und das erhaltene Resultat als Sample in den freien DMA-Buffer geschrieben. Zu beachten ist dabei, dass aufgrund des Symbol-/Samplingratenverhältnisses nur bei jedem dritten Element des DMA-Buffers ein Symbol aus dem Interface-Buffer verarbeitet wird.

Abbildung 8.4-3 zeigt, wie die Verarbeitung im Programmablauf integriert ist und Abbildung 8.4-4 liefert eine Übersicht über den Verarbeitungsablauf.

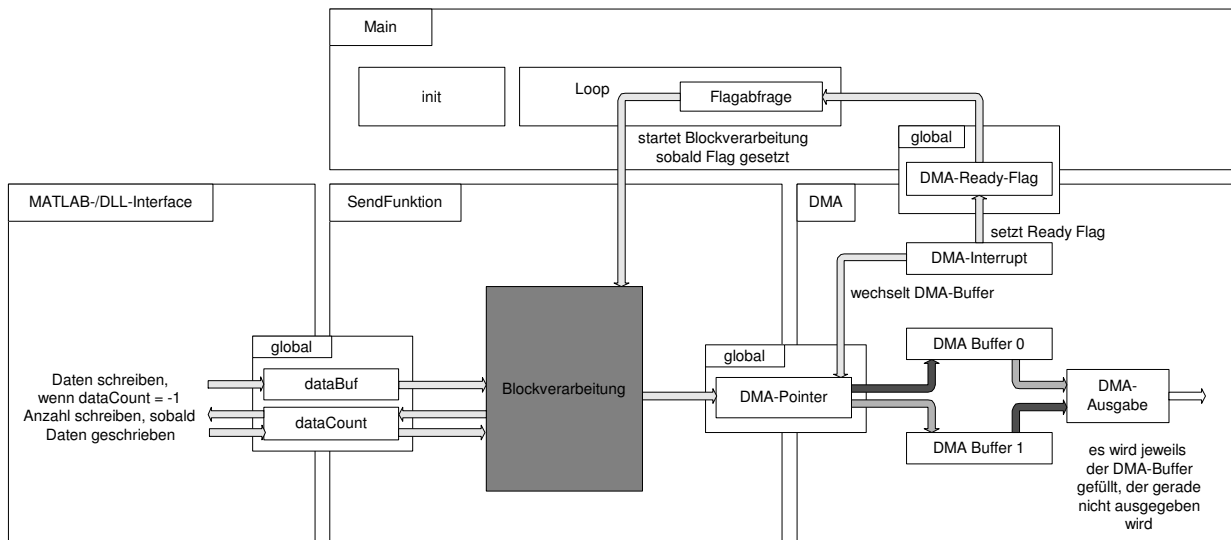


Abbildung 8.4-3 - Integration der Blockverarbeitung im Sender-Algorithmus

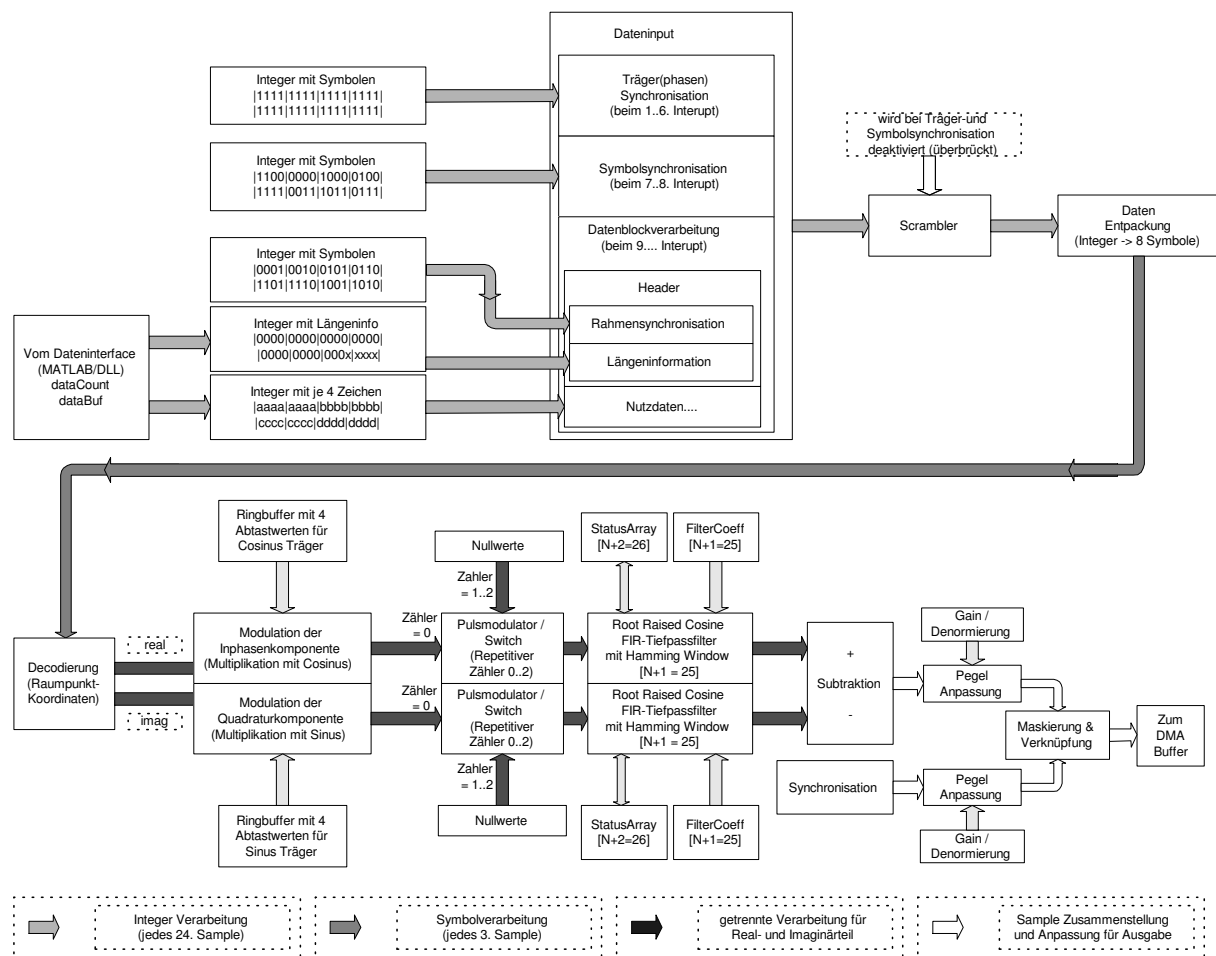


Abbildung 8.4-4 - Ablauf der Blockverarbeitung

## Scrambler

Der Scrambler sorgt dafür, dass nie zwei gleiche Symbole unmittelbar nacheinander über die Datenleitung geschickt werden. So können längere periodische Sequenzen ohne Taktinformationen vermieden werden. Für den gegenwärtigen Fall ist der Scrambler noch nicht von grosser Bedeutung, da die Synchronisation auch mit periodischen Signalen möglich ist. Da die Träger(phasen)synchronisation beim Empfänger jedoch auf Periodische Signale (z.B. Punkt 1111 oder 0000) anspricht, ist der Scrambler das ideale Instrument, um ein Ansprechen der Trägerdetektion während der Datenübermittlung auszuschliessen.

Für den Scramblevorgang wurde eine Struktur gewählt, die selbstsynchronisierend ist. Dadurch wird gewährleistet, dass der Aktivierungszeitpunkt und die Initialisierungswerte beim Sender und Empfänger verschieden sein können und sich Scrambler (Sender) und Descrambler (Empfänger) nach kurzer Zeit in den selben Zustand „eingeschwungen“ haben. Der verwendete Scrambler (Abbildung 8.4-5) ist als Schieberegister mit 17 Bit-Elementen anzusehen, von dem bestimmte Elemente exorverknüpft und rückgeführt werden. Die Abgriffe sollten jeweils bei einem Speicherelement erfolgen, dessen Stelle vom Eingang her gezählt einer Primzahl entspricht (z.B. 7.,13.,17,... Element). Wird nun eine periodische Bitfolge auf den Eingang gegeben, wiederholt sich die Ausgangssequenz nach  $2^n - 1$  Bits. „n“ ist dabei die Anzahl der Speicherelemente. In unserem Falle wäre  $n=17$ .

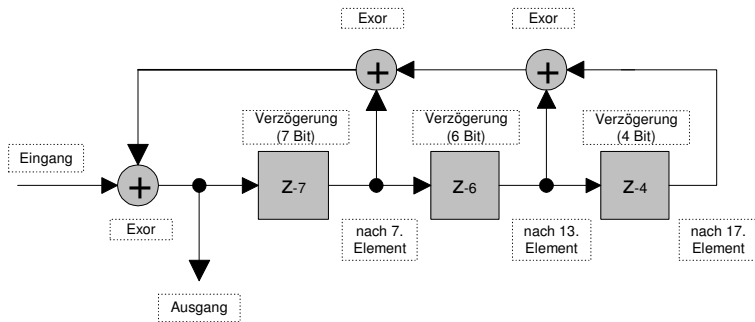


Abbildung 8.4-5 - Blockschaltbild Scrambler

Aus Effizienzgründen erfolgt das Scrambling vor der Aufteilung eines Integerwertes (32-Bit) in die Bestandteile von 8 Symbolen. Wegen der Abhängigkeit des jeweils errechneten Bits vom vorherigen Resultat (Rekursion), ist jedoch keine blockartige Verarbeitung des Integers möglich und es muss jedes Bit einzeln durch aufwendiges Maskieren verarbeitet werden, was relativ rechenintensiv ist. Abbildung 8.4-6 zeigt, wie ein Integerwert verwürfelt (scrambling) wird.

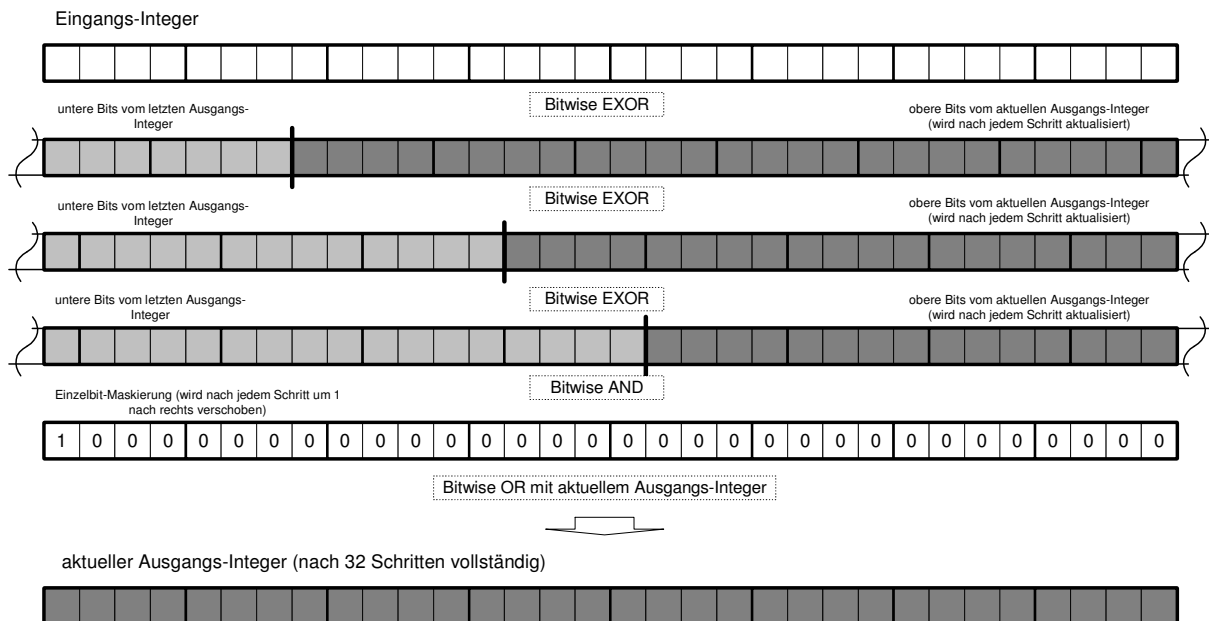


Abbildung 8.4-6 - Bitverarbeitung im Scrambler

### Raumpunkt-Kodierung

Bei der Raumpunktkodierung werden die Werte des Realteils und des Imaginärteils mittels einer Tabelle nach Abbildung 8.4-7 ermittelt. Die Tabelle basiert auf einem zweidimensionalen Array von 2\*16 Elementen. Mit dem ersten Index wird die Koordinatenachse bestimmt und mit dem zweiten Index der jeweilige Symbolwert angegeben. Der Rückgabewert entspricht dann der gewünschten Koordinate (Real oder Imag) des übergebenen Symbolwertes. Die dadurch erhaltenen Koordinaten mit den möglichen Werten -3, -1, 1 und 3 bilden die Ausgangslage für die weiteren Verarbeitungsschritte. Tabelle 8.4-1 zeigt die Inhalte des verwendeten Arrays rpv[2][16].

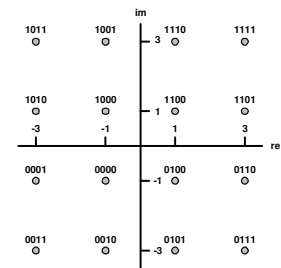


Abbildung 8.4-7

Realteil → rpv[0][symbol <sub>DEZ</sub> ]				Imaginärteil → rpv[1][symbol <sub>DEZ</sub> ]			
Symbolwert	Inhalt	Symbolwert	Inhalt	Symbolwert	Inhalt	Symbolwert	Inhalt
0	-1	8	-1	0	-1	8	1
1	-3	9	-1	1	-1	9	3
2	-1	10	-3	2	-3	10	1
3	-3	11	-3	3	-3	11	3
4	1	12	1	4	-1	12	1
5	1	13	3	5	-3	13	1
6	3	14	1	6	-1	14	3
7	3	15	3	7	-3	15	3

Tabelle 8.4-1 - Raumpunkt-Tabelle

### Pulsmodulator

Der Pulsmodulator sorgt dafür, dass ein Symbol nur während einer Sampleperiode Einfluss auf das Ausgangssignal hat. Da die Symbolrate ein Drittel der Samplefrequenz beträgt, wird nur bei jedem dritten Sample ein Symbol bzw. die daraus hervorgegangenen Koordinatenwerte verarbeitet. Dazwischen werden Nullwerte eingefügt (Abbildung 8.4-8), wodurch schliesslich pro Symbol ein Puls resultiert. Der Pulsmodulator befindet sich unmittelbar nach der Raumpunktkodierung in doppelter Ausführung einmal im Inphasen- und einmal im Quadraturkanal.

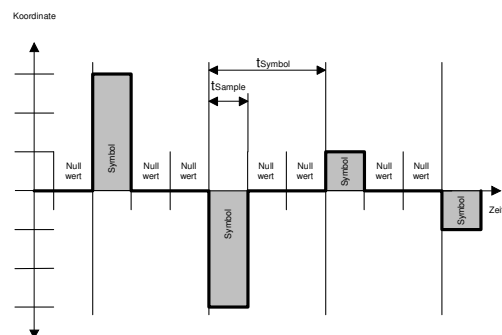


Abbildung 8.4-8 - Pulsmodulator

### Modulator

Nach der Pulsformung folgt die Trägermodulation. Dabei wird die Inphasenkomponente mit einem Cosinusträger und die Quadraturkomponente mit einem Sinusträger multipliziert. Aufgrund des Sampling-/Trägerfrequenzverhältnisses ( $N = 7 \cdot 200\text{Hz} / 1 \cdot 800\text{Hz} = 4$ ), wird ein Trägersignal mit 4 Abtastwerten konstruiert.

Mit dem SHARC-Prozessor lassen sich maximal drei Ringbuffer auf einfache Weise bilden. Der zyklischen Wiederholung der 4 Abtastpunkte wegen, wird für beide Trägerkomponenten je einer dieser Ringbuffer verwendet. Beide Buffer haben die Länge 4, wobei der eine mit den Werten **1,0,-1,0** für den Cosinusträger und der andere mit den Werten **0,1,0,-1** für den Sinusträger initialisiert wird. Die Modulation erfolgt mit jedem Sample.

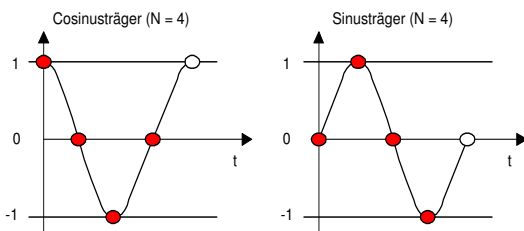


Abbildung 8.4-9

## Tiefpass-Filter

Das FIR-Tiefpassfilter hat in erster Linie die Aufgabe, das theoretisch unendliche Spektrum eines modulierten Symbolpulses auf seine Grundwelle zu begrenzen. Dabei besteht zusätzlich das Problem der Überschneidung zweier aufeinander folgender Impulse, der sogenannten Symbolinterferenz. Um diese so gering wie möglich zu halten, fiel die Entscheidung auf ein Raised-Cosine Tiefpassfilter. Wie im Simulationsteil bereits behandelt, können beim Raised-Cosine Filter die Nullstellen (Nulldurchgänge der Stossantwort) durch das Sampling-/Grenzfrequenzverhältnis so gelegt werden, dass der nächste Puls, also jedes 3. Sample auf eine Nullstelle zu liegen kommt (Abbildung 8.4-10). So werden die relevanten Samples nicht durch benachbarte Pulse beeinträchtigt. Da diese

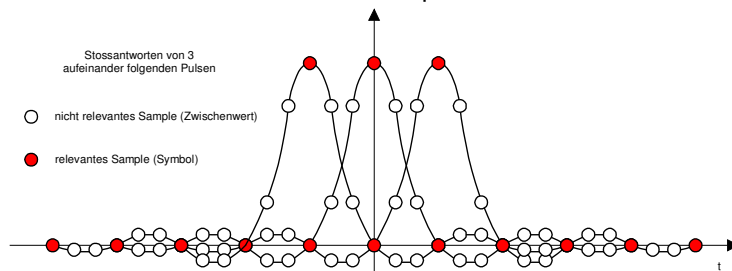


Abbildung 8.4-10 - Stossantworten dreier Pulse

Eigenschaft über die ganze Strecke, also Sender mit Empfänger zusammen, gelten soll, muss beim Sender, wie beim Empfänger ein Root-Raised-Cosine Filter eingesetzt werden. Beide Filter zusammen (Faltung der Stossantworten) haben dann diese Eigenschaft eines normalen Raised-Cosine-Filters.

Die Berechnung der Stossantwort bzw. der FIR-Koeffizienten erfolgt im

Programm nach Gleichung 8.4-4 und Gleichung 8.4-6. Um keine Division durch NULL zu erhalten, muss der Peakwert des Mainlobe (mittlerer Wert) separat behandelt werden. Dazu wird bei  $i=N/2$  die Hilfsvariable  $k[i]$  auf 0.00000001 gesetzt (Gleichung 8.4-7). Zur spektralen Begrenzung der Stossantwort wird zusätzlich ein Hamming-Fenster (Gleichung 8.4-5) eingesetzt. Die Filterlänge wurde zuerst nach Theorie (siehe Skript FIR-Filter) berechnet und nachträglich empirisch (Upload der FIR-Koeffizienten vom DSP) gekürzt, um keine unnötigen Rechenschritte durchführen zu müssen. Die Filterlänge  $N+1$  beträgt nun 25. Die Koeffizienten bleiben während der Echtzeitverarbeitung unverändert und werden beim Programmstart berechnet (Initialisierung in Mainfunktion).

$\alpha$ : Rolloff-Faktor (Flankensteilheit von 0.5)  
**N**: Filterordnung  $(N+1) - 1 \rightarrow$  immer gerade Zahl  
**T**: Periodendauer eines Samples  $(1/f_s)$   
 $f_g$ : Grenzfrequenz 1'200 Hz (halbe Symbolrate)  
**k**: Substitutionsvariable  
**i**: Index von 0...N (0...24)  
**c\_buf[i]**: FIR-Filterkoeffizienten  
**c\_ham[i]**: Koeffizienten des Hammingfensters

$$c\_buf[i] = \frac{[4 \cdot \alpha \cdot k[i] \cdot \cos(\pi \cdot [1 + \alpha] \cdot k[i])] + [\sin(\pi \cdot [1 - \alpha] \cdot k[i])]}{[1 - [4 \cdot \alpha \cdot k[i]]^2] \cdot \pi \cdot [i - N/2] \cdot T} \cdot c\_ham[i]$$

Gleichung 8.4-4 - Berechnung der Filterkoeffizienten

$$c\_ham[i] = 0.54 - 0.46 \cdot \cos\left(2 \cdot \frac{\pi}{N} \cdot i\right)$$

Gleichung 8.4-5 - Hamming-Window

$$k[i] = T \cdot 2 \cdot f_g \cdot [i - [N / 2]]$$

Gleichung 8.4-6 - Substitution

$$k[N / 2] = 0.00000001$$

Gleichung 8.4-7 - Division mit NULL

## Subtraktion und Anpassung

---

Haben Inphasen- und Quadraturwert die FIR-Filter passiert, wird die Quadraturkomponente von der Inphasenkomponente subtrahiert, wodurch ein einzelnes moduliertes Datensample entsteht. Um eine optimale Aussteuerung des D-/A-Wandlers erzielen zu können, wird das Signal nun noch einer Denormierung und Gain-Anpassung unterzogen. Das Synchsignal, welches bei jedem Sample das Vorzeichen wechselt (Rechtecksignal) wird ebenfalls angepasst und mit dem Datensignal verknüpft, wonach der neu erhaltene Einzelwert als Sample im DMA-Buffer abgelegt wird. Die Verknüpfung ist notwendig, da die Ausgabe beider Ausgangskanäle (Ch1 und Ch2) über ein 32-Bit Wort erfolgt. Die oberen 16 Bit bilden dabei das Sample für Kanal 1 und die unteren 16 Bit das Sample für Kanal 2. Mit der Implementierten Verknüpfung erfolgt die Ausgabe des Datensamples auf Kanal 1 und die des Sync-Signals auf Kanal 2.

## 8.5 Empfänger

Der Empfänger erfordert gegenüber dem Sender einen wesentlich höheren Aufwand in der Implementation. Die Synchronisation alleine, nimmt einen grossen Teil des Codeumfangs in Anspruch. Die Abtastung des Daten- und Synchronisationssignals erfolgt mit einer wesentlich höheren Rate als 7'200 Hz, um einen genauen Abgleich der Trägerphase zu erreichen.

### 8.5.1 Datenorganisation

#### Zeichengrösse

Die Zeichengrösse muss zwangsläufig der des Sender entsprechen, um eine korrekte Decodierung zu gewährleisten. Die Datenübermittlung zwischen Empfänger und PC-Anwendung erfolgt somit ebenfalls im ASCII-Format mit einer Zeichengrösse von 8 Bit. Dies entspricht zwei Raumpunkten (4 Bit pro Raumpunkt) pro Zeichen.

#### Speichernutzung

Aus gleichen Gründen wie beim Empfänger beschrieben, erfolgt die Verarbeitung der Daten wo möglich integerweise. So werden ebenfalls 4 Zeichen bzw. 8 Symbole in einem 32-bit Integerwert untergebracht (Abbildung 8.5-1). Auch hier läuft dabei die Verarbeitung eines Integers von links nach rechts ab. Das Zeichen oder Symbol ganz links wird also zuerst empfangen. Die Datenübertragung vom Empfänger zum Anwendungsprogramm (DSP→PC) erfolgt ebenfalls mit Integerwerten.

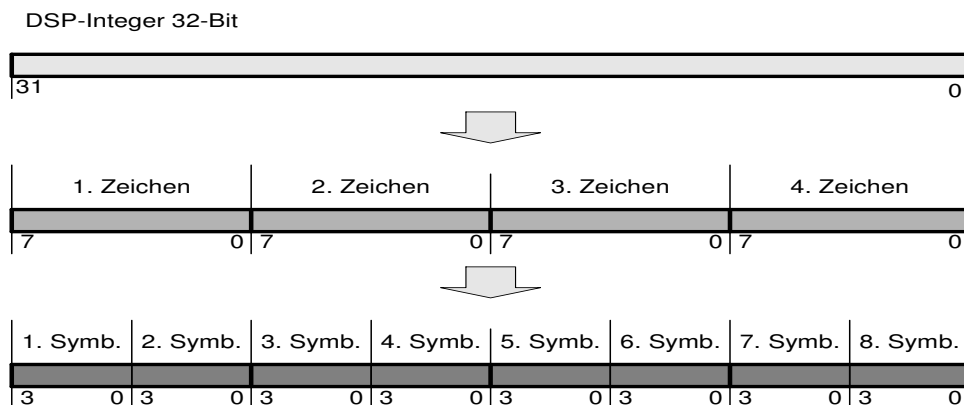


Abbildung 8.5-1 - Speicherausnutzung

#### Sample-Interrupt

Beim Empfänger läuft die Verarbeitung im Gegensatz zum Empfänger voll umfänglich in Echtzeit ab. Die Sampling-Interrupts erfolgen aufgrund eines Timerüberlaufs in periodischen Zeitabständen (Sampleperiode) mit einer Frequenz von mehreren Vielfachen der eigentlichen Samplingfrequenz von 7'200 Hz. Die dadurch erreichte Überabtastung zwingt sich mit dem gewählten Synchronisationsverfahren auf. Im normalen Betrieb (Datenübertragung) findet nicht bei jedem ausgelösten Sampleinterrupt eine Verarbeitung von Daten statt.

#### Buffergrössen

Der Empfänger muss die Blöcke von maximal 400 Nutzzeichen oder 800 Symbolen und dem zusätzlichen Header verarbeiten können. Ein Buffer für die Kommunikation zwischen DSP und PC kann 100 Integer und somit einen ganzen Block von 400 Nutzzeichen aufnehmen. Um einen gleitenden Datenaustausch zu ermöglichen, werden zwei Kommunikations-Buffer zur Verfügung gestellt.

## 8.5.2 Synchronisation

### Frequenzsynchronisation

Die Frequenzsynchronisation erfolgt mit Hilfe des über den zweiten Kanal übermittelten Synchronisationssignals. Um später eine möglichst genaue Phasensynchronisation zu ermöglichen, werden die Eingangssignale (Daten und Sync) mit einer Samplingfrequenz von 72'000 Hz abgetastet. Dies entspricht dem zehnfachen der Verarbeitungsfrequenz. Das Sync-Signal mit der Frequenz von 3'600 Hz wird somit  $N = 72'000\text{Hz} / 3'600\text{Hz} = 20$  mal pro Periode abgetastet. Die Frequenzsynchronisation ist direkt in der Interrupt-Routine implementiert, da sie immer erfolgen muss und keine Verzögerungen auftreten dürfen. Dabei werden die Samples pro Zustand (Peak) des Rechtecksignals gezählt (Abbildung 8.5-2). Weicht der Zähler nach einem Flankenwechsel vom Normwert von  $20/2 = 10$  ab, wird die Samplingfrequenz (Timer auf Bitsi-Board) nach folgendem Muster um 5Hz erhöht, oder verkleinert:

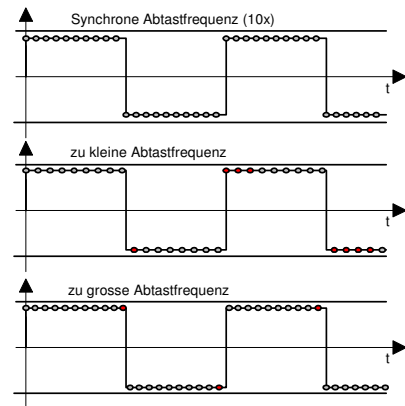


Abbildung 8.5-2

- Counterwert > 10 → Zu hohe Abtastrate → Frequenz um 5Hz reduzieren
- Counterwert < 10 → Zu kleine Abtastrate → Frequenz um 5Hz erhöhen
- Counterwert = 10 → Keine Änderung

Dieser Synchronisationsvorgang geschieht fortwährend, solange das Programm läuft.

### Träger(phasen)synchronisation

Die Trägerphasensynchronisation kann von Seite des Empfängers zu jedem Zeitpunkt erfolgen. Voraussetzung dafür ist ein periodisches Datensignal, welches durch permanentes Senden des Raumpunktes 1111<sub>BIN</sub> mit den Koordinaten 3/3 (siehe Sender) entsteht. Für die Detektion des periodischen Signals werden über eine Periode (4 Samples) die einzelnen Werte mit den jeweils um eine Periode zurückliegenden Abtastwerten verglichen. Liegt deren Abweichung (Mean Square Error MSE) unter einem bestimmten Grenzwert, so wird das Signal als periodisches Trägersignal interpretiert und die Trägerphasensynchronisation aktiviert. Abbildung 8.5-3 zeigt, wie die Abtastpunkte für die Trägerdetektion verarbeitet werden.

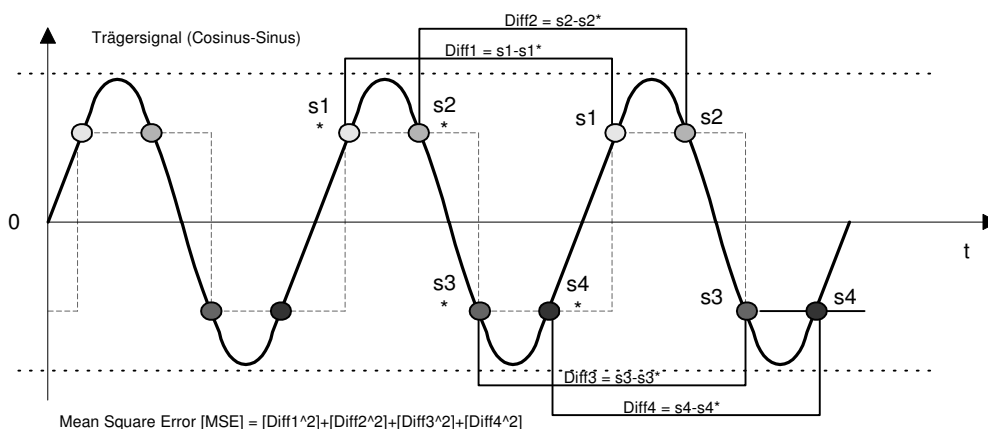


Abbildung 8.5-3 - Trägererkennung

Hat die Trägererkennung einmal angesprochen, wird zuerst eine Grobsynchronisation des Trägers am Empfänger vorgenommen. Dabei wird mit jedem Abtastwert das Trägersignal (Cosinus und Sinus) um ein Sample verschoben, bis weder Inphasen- noch Quadraturkomponente nach der Demodulation negative Vorzeichen aufweisen (nur positive Halbwellen nach Demodulation). Abbildung 8.5-4 veranschaulicht die demodulierten Signalkomponenten vor und nach der Grobsynchronisation.

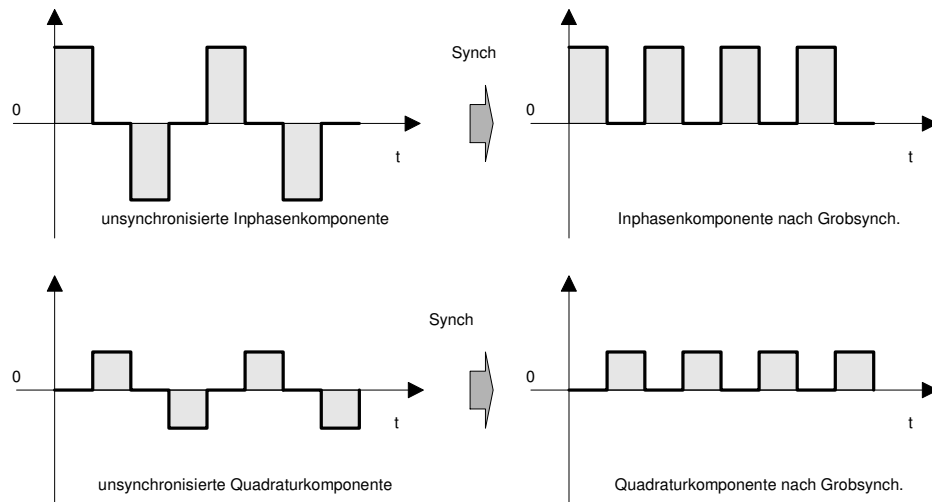


Abbildung 8.5-4 - Vorsynchronisation (Vorzeichen)

Um die Phase nun genau abzugleichen, wird anschliessend eine Feinsynchronisation vorgenommen. Dabei wird nicht mehr der Träger verändert, sondern der Abtastpunkt im Raster der Überabtastung (10x) verschoben, bis Inphasen- und Quadraturkomponente nach der Modulation die gleiche Pulshöhe aufweisen (siehe Abbildung 8.5-5). Beide Verfahren zur Trägerphasensynchronisation erfolgen, im Falle einer Trägerdetektion (periodisches Signal auf der Datenleitung), bei jedem regulärem Sample, also mit  $7 \cdot 200$  Hz.

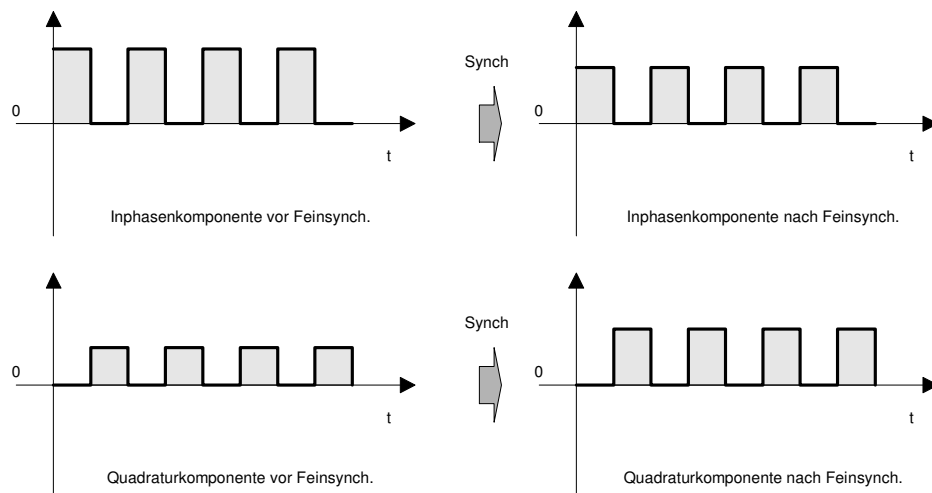


Abbildung 8.5-5 - Feinsynchronisation (Phase)

Die Feinsynchronisation kann je nach aktueller Lage des Abtastpunktes auf dem Datensignal, durch die Feinverschiebung über einen Nulldurchgang erneut ein negatives Vorzeichen bewirken, wodurch die Grobsynchronisierung anspricht und die ganze Synchronisation von neuem beginnt. Dies kommt im Normalfall aber nur einmal pro Synchronisationsphase vor. Abbildung 8.5-6 verdeutlicht die Lage der Abtastpunkte für einen ideal synchronisierten Träger.

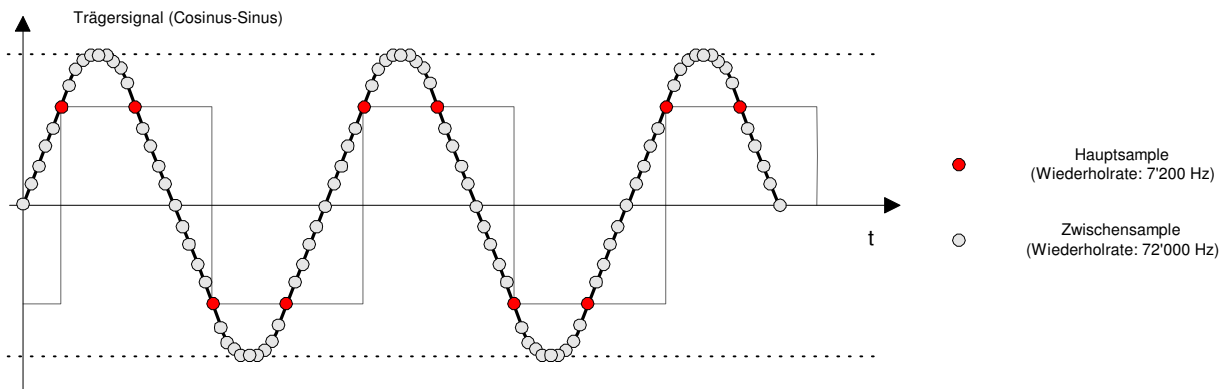


Abbildung 8.5-6 - Abtastung eines Datensignals (Träger) bei idealer Synchronisation

## Symbolsynchronisation

Ist der Träger einmal synchronisiert, kann die Symbolsynchronisation erfolgen. Dabei wird vom Sender eine repetitive Sequenz von 8 definierten Symbolen (siehe Sender) übertragen. Vor und während der Symbolsynchronisation arbeitet der Empfänger mit einer Verarbeitungsrate von 7'200 Hz. Es wird dabei also jedes Symbol und die dazugehörigen Zwischenwerte (3 Samples pro Symbol) verarbeitet. Sobald nun die bekannte Symbolsequenz erkannt worden ist, ist dem Empfänger der Zeitpunkt eines relevanten Symbolen bekannt, und die Zwischenwerte werden nicht mehr verarbeitet. Von nun an wird also nur noch jedes dritte Sample bzw. jedes 30. Sample der effektiven Samplingrate beachtet, womit die Verarbeitung mit der Symbolrate von 2'400 Hz erfolgt.

## Rahmensynchronisation

Mit Rahmensynchronisation ist die Blockerkennung gemeint. Sie ist im Gegensatz zu den oben erwähnten Synchronisationsphasen fester Bestandteil der Datenverarbeitung und erfolgt jeweils zu Beginn einer Blockverarbeitung. Wie bereits mehrmals angesprochen (Kapitel 8.3.2), wird jedem Nutzdatenblock ein Header mit der Erkennungssequenz und der Blocklänge vorangestellt. Wichtig zu erwähnen ist, dass das ganze mit aktiviertem Scrambler am Sender bzw. Descrambler am Empfänger vorstatten geht. Die Übertragung der Bits erfolgt jetzt also verwürfelt. Aufgrund dieser Tatsache ist die Blockerkennung nach dem Descrambler implementiert.

## 8.5.3 Datenverarbeitung

Unter der Datenverarbeitung ist der Empfang eines vom Sender übertragenen Datenblocks zu verstehen. Im Gegensatz zum Sender wird ein Block beim Empfänger Symbol für Symbol verarbeitet. Das heißt, dass jedes empfangene Symbol sofort verarbeitet und zwischengespeichert wird. Die Verarbeitungszeit muss wesentlich kürzer sein, als eine Taktperiode von  $1 / 7'200$  Hz, damit zusätzlich noch die Frequenzsynchronisation mit einer 10-fachen Überabtastung erfolgen kann.

Ein digitaler AGC (Auto Gain Control) sorgt dafür, dass unabhängig vom Signalpegel auf der Datenleitung der erforderliche Wertebereich nach dem A-/D-Wandler ausgenutzt wird. Danach folgt die Trägersynchronisationsstufe, welche beim Datenverkehr aufgrund des nichtperiodischen Signals überbrückt wird.

Durch Modulieren (multiplizieren) des geregelten Eingangssignals mit den jeweiligen Trägern werden die diskretisierten Raumkomponenten zurückgewonnen. Die Multiplikation mit dem Cosinusträger ergibt dabei die Inphasen- und die Multiplikation mit dem inversen Sinusträger die Quadraturkomponente. Über mehrere Samples betrachtet, weisen beide Signale einen pulsartigen Verlauf auf. Beide Signale werden nun durch ein Root-Raised-Cosine FIR-Filter geglättet, wodurch zusätzlich die Überlappung zweier benachbarter Symbole an den relevanten Punkten minimiert wird. Danach wird jede Komponente für sich nach Pegel und Vorzeichen ausgewertet, woraus die entsprechenden Raumpunktkoordinaten abgeleitet werden können.

Anhand einer Decodiertabelle werden die Raumpunkte oder Symbole ermittelt. Diese zwei Schritte der Signalauswertung geschehen nach erfolgter Symbolsynchronisation nur bei jedem dritten Sample mit der Symbolrate von 2'400 Hz. Um nun eine effizientere Datenverarbeitung zu ermöglichen wird das Symbol mit weiteren Symbolen in einen Integer (8 Symbole) verpackt und in dieser Form durch den Descrambler geschickt. Dieser wandelt den vorliegenden Integer bzw. die 32-Bits in die eigentlichen unverschlüsselten Bits um, wodurch ein anderer Integerwert resultiert. Der erhaltene Integer wird darauf im aktuellen Kommunikationsbuffer für den Datenupload (DSP→PC) abgelegt. Ist der ganze Block verarbeitet, wird die zum Kommunikationsbuffer gehörende Längenvariable geschrieben und somit der Buffer für den Upload freigegeben.

Abbildung 8.5-7 zeigt den Verarbeitungsablauf und Abbildung 8.5-7 liefert eine Übersicht über dessen Integration im Programm.

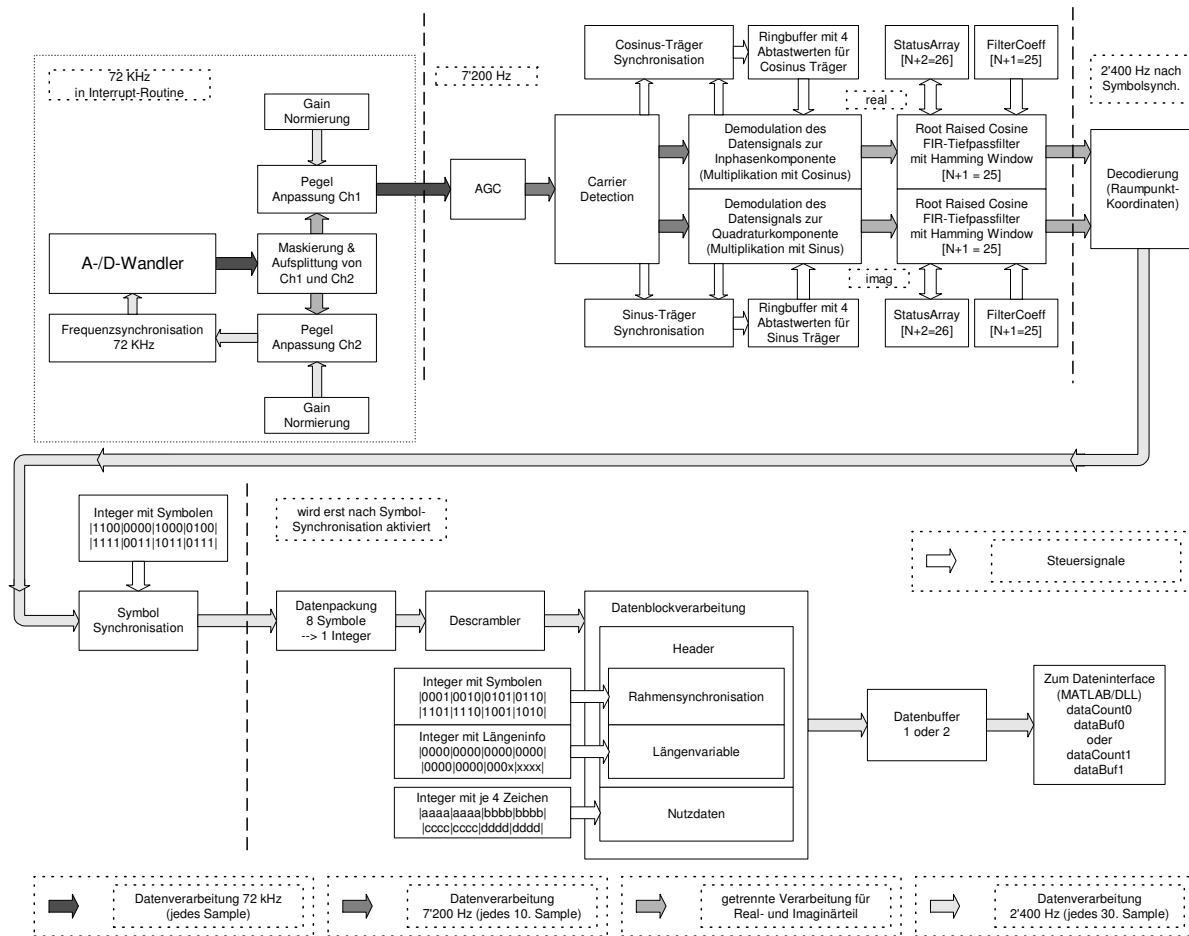


Abbildung 8.5-7 - Ablauf der Sampleverarbeitung bzw. Synchronisation

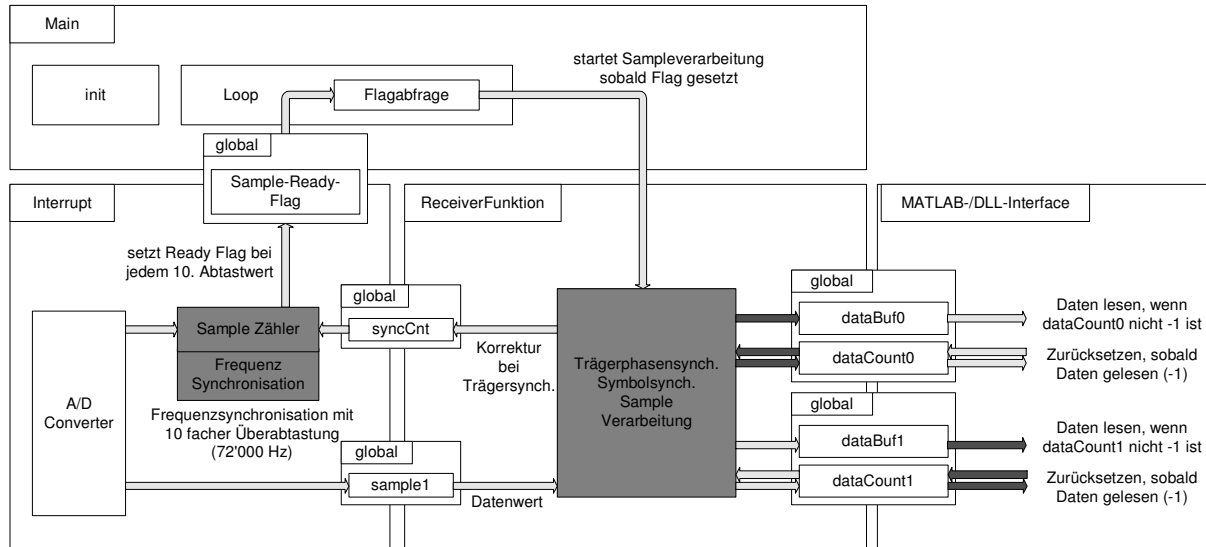


Abbildung 8.5-8 - Integration der Blockverarbeitung im Sender-Algorithmus

### Auto Gain Control (AGC)

Der AGC befindet sich am Anfang des Verarbeitungsflusses. Er hat die Aufgabe, Pegelabweichungen des gesendeten Datensignals auszugleichen.

Normalerweise erfolgt die Pegelkorrektur vor dem A-/D-Konverter in analoger Weise, um eine optimale Aussteuerung des Wandlers und somit eine möglichst grosse Auflösung des Signals zu erreichen.

Im gegenwärtigen Falle ist der AGC voll digital, also nach dem Konverter implementiert. Dabei wird der jeweilige Fließkomma-Zahlenwert eines Samples mit einem laufend neu errechneten Faktor multipliziert, wodurch Amplitudenschwankungen auf dem Übertragungskanal auf digitaler Ebene kompensiert werden. Das Ganze ist hauptsächlich notwendig, damit die vielerorts verwendeten Vergleiche der Signalwerte mit Schwellwerten, zuverlässig arbeiten. Der implementierte AGC (Abbildung 8.5-9) weist die selbe Struktur wie in der Simulation auf. Nur die Parameter (Gewichtung, Reaktionsgeschwindigkeit) variieren aufgrund der unterschiedlichen Wertebereiche.

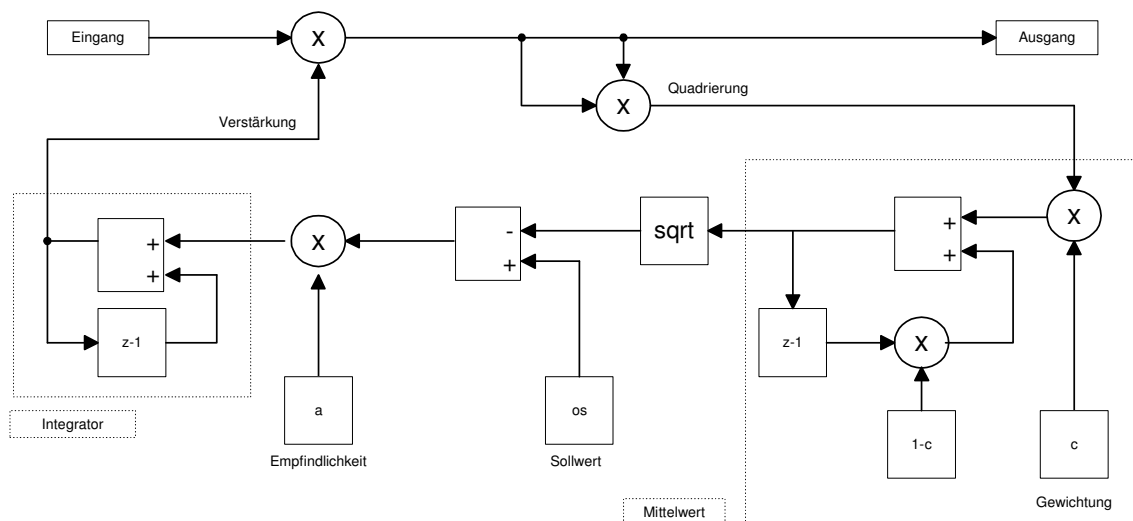


Abbildung 8.5-9 - Blockschema des AGC

Der AGC konnte aus zeitlichen Gründen noch nicht optimiert werden. Daher kann es vorkommen, dass das Modem durch Instabilität des AGC aus der Synchronisation läuft. Ein zweiter Schritt der Optimierung wäre die Realisierung ohne Quadratwurzel (Block  $\sqrt{\phantom{x}}$ ), denn die Wurzelrechnung bedeutet für den DSP einen grossen Rechenaufwand.

### Träger-Detektor

Der Trägerdetektionsblock erkennt die Trägersynchronisationssequenz nach im Kapitel 8.5.2 erläuterten Verfahren. Während der Detektierung wird der im selben Kapitel erwähnte Synchronisationsalgorithmus (Grobsynchronisation und Feinsynchronisation) freigegeben und ein Abgleich der Trägerphase vorgenommen. Der Synchronisationsalgorithmus nimmt einerseits Einfluss auf die Ringbuffer für Cosinus- und Sinusträgerwerte und andererseits Einfluss auf den Abtastzeitpunkt im Feinraster der effektiven Samplingrate von 72kHz (Interrupt-Routine).

### Demodulator

Die Demodulation erfolgt am Anfang des Signalpfades nach dem AGC bzw. dem Trägerdetektor. Dabei wird das Datensignal einmal mit dem invertierten Sinus- und einmal mit dem Cosinusträger multipliziert, wodurch die Inphasen- und die Quadraturkomponente zurückgewonnen wird. Die Trägersignale werden auf gleiche Weise generiert, wie es im Sender der Fall ist. Es stehen dazu ebenfalls je ein Ringbuffer mit  $N=4$  Abtastwerten (Abbildung 8.5-10) zur Verfügung. In unserem Falle liegen beide Komponenten nach der Demodulation in gepulster Form vor. Die Ringbuffer sind identisch dem Sender mit den Werten **1,0,-1,0** für den Cosinusträger und **0,1,0,-1** für den Sinusträger initialisiert. Die Modulation erfolgt mit der regulären Taktrate von 7'200 Hz.

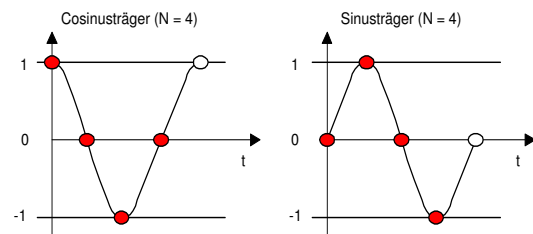


Abbildung 8.5-10

### Tiefpass-Filter

Das FIR-Tiefpassfilter hat primär die Aufgabe, den Trägeranteil in einem demodulierten Signal zu unterdrücken. Dadurch werden die Pulse der Inphasen- und Quadraturkomponente geglättet. Gleich wie im Sender, wird pro Komponente je ein Root-Raised-Cosine-Filter mit einer Grenzfrequenz von 1'200 Hz eingesetzt. Beide Filter zusammen (Sender und Empfänger) bilden nun ein Raised-Cosine-Filter, bei welchen die Nullstellen gezielt positioniert werden können. Diese Eigenschaft wird wie beim Sender beschrieben, zur Vermeidung von Symbolüberlappungen (Symbolinterferenz) ausgenutzt.

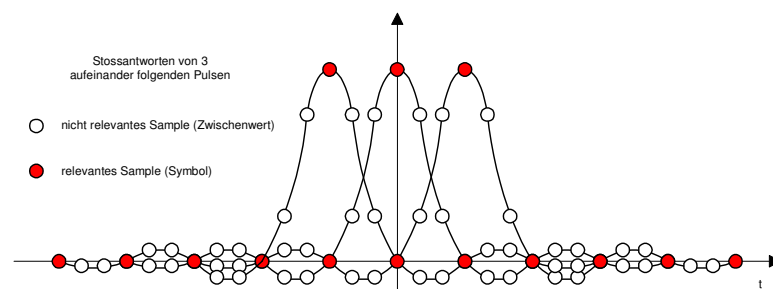


Abbildung 8.5-11 - Stossantworten dreier Pulse

reguläre Samplefrequenz von 7'200 Hz und  $f_g$  die Grenzfrequenz der beiden Filter von 1'200 Hz.

Abbildung 8.5-11 zeigt nochmals, wie die Nullstellen des gesamten Raised-Cosine-Filters für unseren Anwendungsfall zur Vermeidung von Symbolinterferenzen positioniert werden. Da nur jedes dritte Sample ein relevantes Zeichen liefert, müssen auch die Nullstellen bei jedem dritten Sample zu liegen kommen, was durch das Verhältnis  $f_s/2f_g$  bestimmt wird.  $f_s$  ist dabei die

Die Berechnung der Filterkoeffizienten erfolgt beim Programmstart (Initialisierung) nach dem gleichen Muster wie beim Sender. Die dazu notwendigen Formeln sind im Abschnitt Tiefpass-Filter des Senders (Kapitel 8.4.3) zu finden.

## Raumpunkt-Dekodierung

Die Raumpunkt-Dekodierung erfolgt in folgenden zwei Schritten:

- **Auswertung der Komponentenwerte nach Vorzeichen und Wertebereich** → -3 / -1 / 1 / 3
- **Ermitteln des Raumpunktes anhand einer Tabelle**

Zuerst erfolgt für beide Komponenten separat eine Normierung in einen bekannten Wertebereich, wonach anhand des Wertes und des Vorzeichens die Zuteilung eines Indexwertes (0/1/2/3) nach folgenden Kriterien vonstatten geht:

- |                        |                         |                       |                 |
|------------------------|-------------------------|-----------------------|-----------------|
| • Norm · InphaseData   | größer 2                | → RealKoordinate = 3  | → RealIndex = 0 |
| • Norm · InphaseData   | größer 0 und kleiner 2  | → RealKoordinate = 1  | → RealIndex = 1 |
| • Norm · InphaseData   | kleiner 0 und größer -2 | → RealKoordinate = -1 | → RealIndex = 2 |
| • Norm · InphaseData   | kleiner -2              | → RealKoordinate = -3 | → RealIndex = 3 |
|                        |                         |                       |                 |
| • Norm · QuadraturData | größer 2                | → ImagKoordinate = 3  | → ImagIndex = 0 |
| • Norm · QuadraturData | größer 0 und kleiner 2  | → ImagKoordinate = 1  | → ImagIndex = 1 |
| • Norm · QuadraturData | kleiner 0 und größer -2 | → ImagKoordinate = -1 | → ImagIndex = 2 |
| • Norm · QuadraturData | kleiner -2              | → ImagKoordinate = -3 | → ImagIndex = 3 |

Mit den neu erhaltenen ganzzahligen Indizes kann nun die Ermittlung des zugehörigen Symbols anhand einer Tabelle erfolgen. Die Tabelle ist dabei als zweidimensionaler Array einer Grösse von 4\*4 Elementen (rpv[4][4]) realisiert, wobei der erste Index die Realkoordinate und der zweite Index die Imaginärkoordinate in der Punktebene (Abbildung 8.5-12) darstellt. Zusammenfassend werden in Tabelle 8.5-1 die Zusammenhänge zwischen Raumpunkt-Koordinate, Array-Index und Raumpunkt dargestellt.

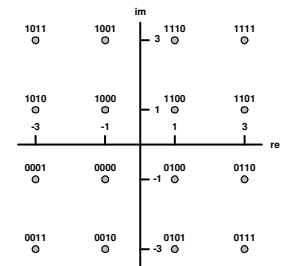


Abbildung 8.5-12

real Koordinate	imag Koordinate	real Index	imag Index	Symbol	real Koordinate	imag Koordinate	real Index	imag Index	Symbol
1	1	0	0	12	-1	1	2	0	8
1	3	0	1	14	-1	3	2	1	9
1	-1	0	2	4	-1	-1	2	2	0
1	-3	0	3	5	-1	-3	2	3	2
3	1	1	0	13	-3	1	3	0	10
3	3	1	1	15	-3	3	3	1	11
3	-1	1	2	6	-3	-1	3	2	1
3	-3	1	3	7	-3	-3	3	3	3

Tabelle 8.5-1 - Raumpunkt-Index-Abhängigkeits-Tabelle

## Descrambler

Der Descrambler wird erst nach der Träger- und Symbolsynchronisation aktiviert. Seine Aufgabe besteht darin, die durch den Scrambler des Senders verwürfelten Bits in die Ausgangsposition zurückzuwandeln. Der Descrambler ist wie der Scrambler selbstsynchronisierend, wodurch die Initialisierungswerte und der Zeitpunkt der Aktivierung verschieden sein dürfen. Nach einer bestimmten Angleichszeit erscheint am Ausgang des Descramblers der gleiche Bitstrom wie am Eingang des Scramblers beim Empfänger.

Der Descrambler weist eine dem Scrambler ähnliche Struktur mit 17 Verzögerungs-Gliedern und drei Abgriffen nach dem 7., 13. und 17. Element auf. Die Abgriffe werden jeweils untereinander und mit dem Eingangssignal exorverknüpft wodurch das Ausgangssignal gebildet wird (Abbildung 8.5-13).

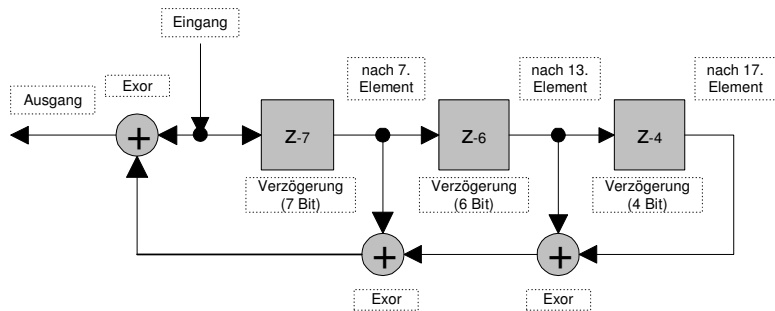


Abbildung 8.5-13

Wie beim Sender, erfolgt das Descrambling aus Effizienzgründen mit einer Breite von 32-Bit. Dabei werden die verwendeten Integerwerte so zurechtgeschoben, damit pro Integerwert nur eine Bitweise-EXOR-Verknüpfung erfolgen muss. Dies ist hier im Gegensatz zum Scrambler möglich, da ein errechnetes Bit nicht vom Resultat der vorherigen Verknüpfung abhängig ist. Abbildung 8.5-14 zeigt ergänzend den Descramblevorgang eines einzelnen Integerwertes.

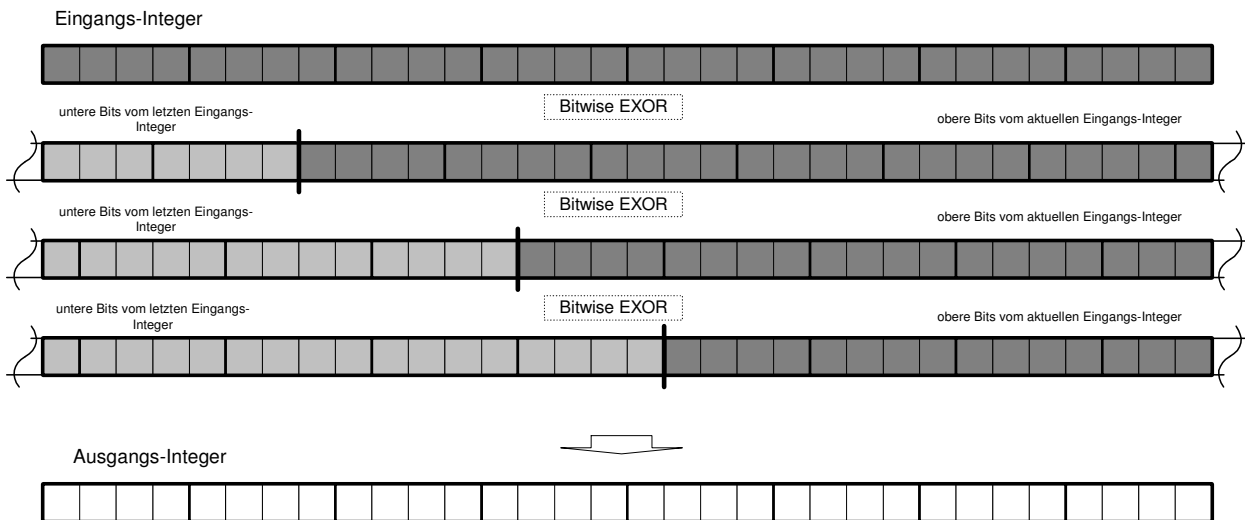


Abbildung 8.5-14 - Bitverarbeitung im Descrambler

## Datenbuffer

---

Ein Datenbuffer kann einen Datenblock der maximalen Länge von 400 Zeichen bzw. 100 Integerwerten aufnehmen. Um einen reibungslosen Ablauf garantieren zu können, wurden zwei Datenbuffer implementiert. So kann von einem Buffer gelesen werden, während der andere durch den DSP-Algorithmus beschrieben wird.

In der Datenblockverarbeitung werden die Nutzdaten, die in Form von Integerwerten vorliegen, an den aktiven Datenbuffer weitergegeben. Sind 100 Integerwerte (400 Nutzzeichen des Datenblocks) geschrieben worden, ist der Buffer voll und die Längenvariable aus dem Header des Datenblocks wird in die Längenvariable des jeweiligen Buffers geschrieben. Damit ist der Kommunikationsbuffer bereit für den Upload (DSP-PC).

Die Wahl des jeweiligen Kommunikationsbuffers erfolgt nach folgenden Kriterien:

- Buffer0 bereit (Längenvariable gleich -1) ?
  - Daten in Buffer0 schreiben und zugehörige Längenvariable setzen.
- Buffer0 nicht bereit (Längenvariable nicht gleich -1) ?
  - Daten in Buffer1 schreiben und zugehörige Längenvariable setzen.

Beim Lesen der Kommunikationsbuffer durch die PC-Anwendung müssen immer beide Buffer nach folgenden Kriterien abgefragt werden:

- Buffer0 bereit (Längenvariable grösser als -1) ?
  - Daten von Buffer0 lesen und zugehörige Längenvariable zurücksetzen (-1).
- Buffer0 nicht bereit (Längenvariable nicht gleich -1) ?
  - Buffer leer oder nicht bereit → keine Beachtung
- Buffer1 bereit (Längenvariable grösser als -1) ?
  - Daten von Buffer1 lesen und zugehörige Längenvariable zurücksetzen (-1).
- Buffer1 nicht bereit (Längenvariable nicht gleich -1) ?
  - Buffer leer oder nicht bereit → keine Beachtung

## 8.6 Kanal

Normalerweise werden QAM-Modem's zur Datenübertragung über das analoge Telefonnetz eingesetzt. Die Bandbreite eines analogen Telefonkanals reicht von 300 Hz bis 3.4 kHz und lässt heute durch erweiterte QAM-Verfahren eine maximale Bitübertragungsrate von 56 kBit/s zu.

### 8.6.1 Kanalfilter

Bevor das zeit- und wertdiskrete Datensignal am Ausgang des D-/A-Wandlers beim Sender über den Kanal (z.B. Telefonleitung) geschickt werden kann, muss es zuerst durch einen Tiefpass bandbegrenzt werden, um eine kontrollierte Übertragung zu ermöglichen. Zur Dämpfung hochfrequenter Störanteile, sollte am Eingang des Empfängers ebenfalls eine Tiefpassfilterung vorgenommen werden.

In der vorliegenden Testeinrichtung wird am Ausgang des Senders ein variables Tiefpassfilter eingesetzt, um das treppenförmige Signal zu glätten. Um garantiert keine zu grosse Dämpfung des Nutzbandes unter 3.4 kHz zu bewirken, wurden die Tests mit einer Filtergrenzfrequenz von 3.8 kHz durchgeführt. Auf das Entstörfilter am Eingang des Empfängers wurde verzichtet, da die Übertragung über 50 Ohm-Kabel erfolgt und es bis anhin nicht möglich war, die Einrichtung auf Rauschsignal-Einflüsse zu testen. Würde dem Kanal Rauschen beigegeben, müsste beim Empfänger unbedingt ein analoges Eingangsfilter eingesetzt werden.

## 8.7 Testprogramm

### 8.7.1 Hardwareanordnung

Für die Ausführung der Testprogramme ist die in Abbildung 8.7-1 dargestellte Anordnung der Komponenten zu empfehlen. Der KO ist dabei nicht notwendiger Bestandteil des Test, sondern dient lediglich der Visualisierung der Übertragungssignale und der Kontrolle des Empfängers (Real-/Imaginärkomponente).

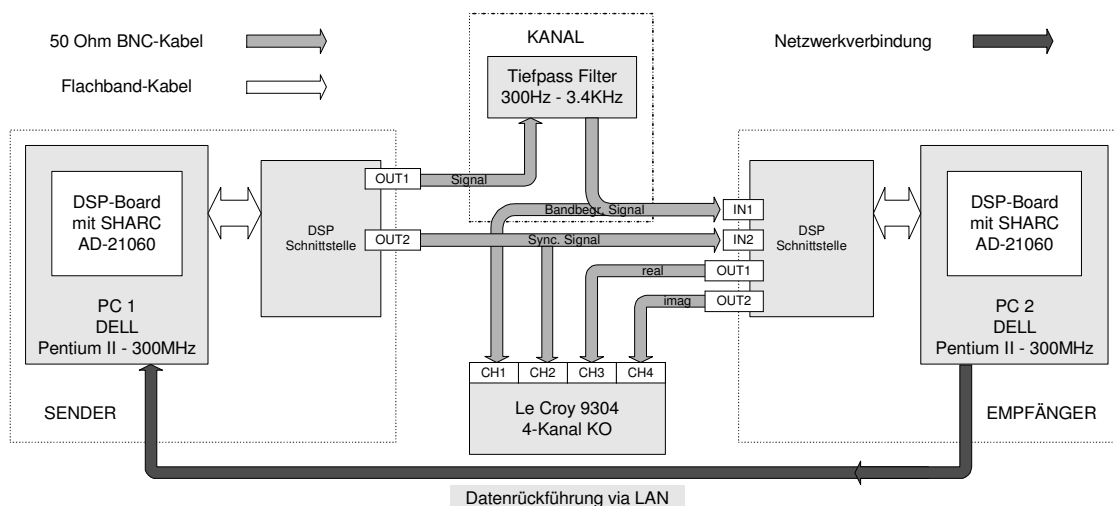


Abbildung 8.7-1 - Testanordnung

## 8.7.2 Programmiersprachen

Aufgrund der einfachen GUI-Programmierung erfolgte die Implementation des Testprogrammes in JAVA. Dabei wurde der JDK-Interpreter Version 1.2.2 verwendet, der von Sun-Microsystems (<http://java.sun.com>) als Freeware zur Verfügung gestellt wird. Als Editor und Projektmanager wurde KAWA 3.21 eingesetzt, welcher als 40 Tages Free-Version zur Verfügung steht.

Die Kommunikation mit dem DSP-Board geschieht mittels bereitgestellter C++-Funktionen. Für die Einbettung dieser Funktionen in das JAVA-Programm, musste ein JAVA-Native-Interface in C++ realisiert werden, welches im Gegensatz zu JAVA plattformspezifisch, also Maschinen- und Betriebssystemabhängig ist.

Dieses Native-Interface wird mit einem Microsoft Visual-C++ Compiler zu einer DLL-Datei (Dynamic Link Library) kompiliert, auf welche dann bei Ausführung der JAVA-Anwendung, im Falle einer Kommunikation mit dem DSP, zugegriffen wird.

## 8.7.3 Sendeapplication

Die Sendeapplication (Abbildung 8.7-2) hat in erster Linie die Aufgabe, über den Sender (DSP-Board) einen kontinuierlichen Datenstrom an den Empfänger zu schicken. Das Empfängerprogramm retourniert die empfangenen Daten über das HSR-interne LAN (Netzwerk) an die Sendeapplication zurück, worauf diese die gesendeten mit den erhaltenen Daten vergleicht. Die voneinander abweichenden Zeichen werden dabei gezählt und auf dem Benutzerinterface angezeigt.

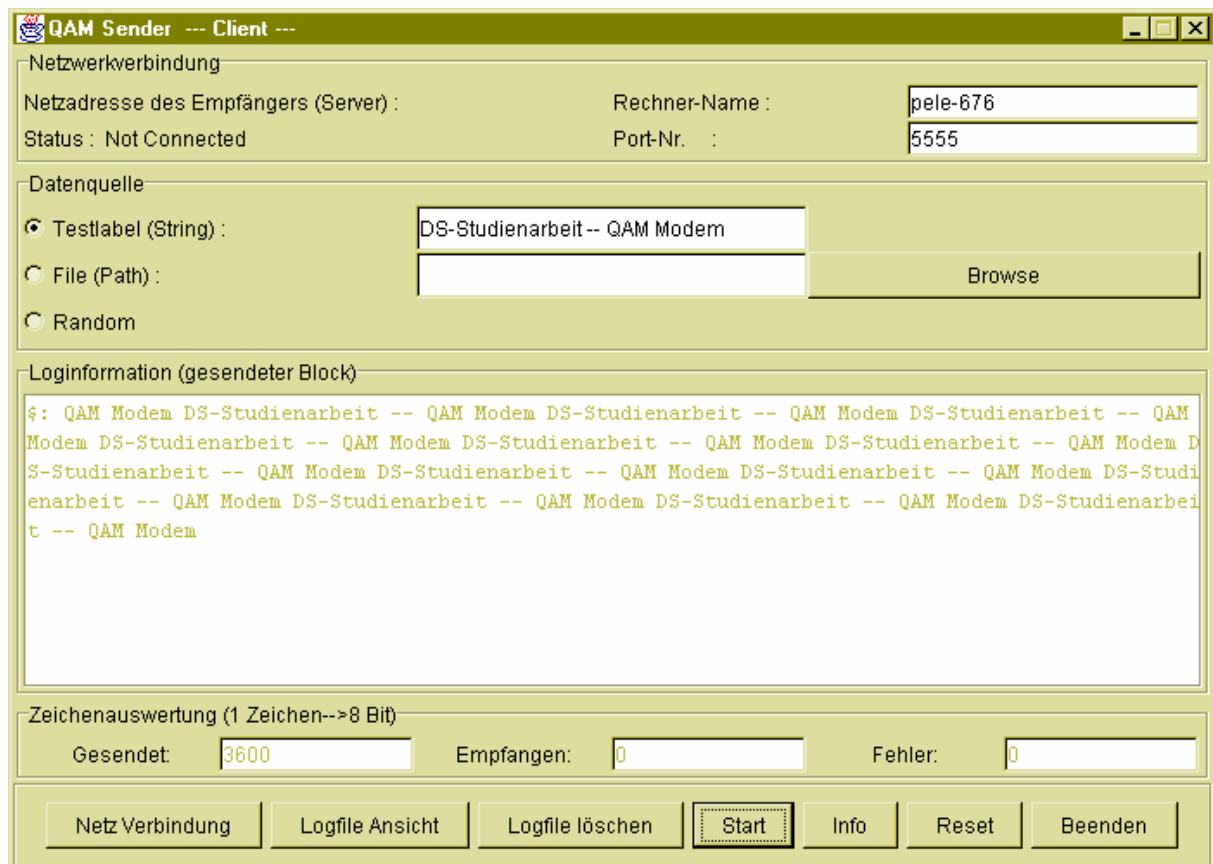


Abbildung 8.7-2 - GUI der Sendeapplication

Während der Datenübertragung wird laufend ein Logbuch geführt, wobei die verglichenen Daten zeichenweise unmittelbar hintereinander dargestellt und fortlaufend im Logfile [send.log] abgelegt werden. Alle Zeichen sollten immer Paarweise im Logfile dargestellt werden. Ist dies nicht der Fall, so liegt an dieser Stelle ein Fehler vor.

## Netzwerkverbindung

---

Bevor eine Netzwerkverbindung entstehen kann, muss die Portnummer [1024...65535 sind frei verfügbar] bestimmt werden, über die kommuniziert werden soll. Weiter muss der Zielrechner mit dem Serverprogramm angegeben werden. Dieser Name wird jeweils in die entsprechende IP-Nummer des Rechners umgewandelt. Im vorliegenden Fall wird eine sogenannte Socketverbindung zwischen dem Empfänger (Server) und dem Sender (Client) hergestellt. Dabei werden die Daten mit UDP, dem unsicheren Protokoll verschickt. UDP/IP ist im Gegensatz zum bekannten TCP/IP nicht gegen Verlust von Datenpaketen abgesichert, da keine Bestätigungen erfolgen.

In unserem Falle kommuniziert der Empfänger (Server) über die fixe Portnummer 5555. Der Rechnername ist allenfalls auf die Bezeichnung des Zielrechners anzupassen, auf dem die Empfängeranwendung läuft. Sind die Eingaben korrekt und ist der Empfänger betriebsbereit, kann die Verbindung mit dem Button *NetzVerbindung* hergestellt werden.

Es ist nicht zwingend eine Rückführung der Daten über das LAN notwendig, wenn auf eine Fehlerauswertung verzichtet werden kann. Sender und Empfänger können auch zum Datenaustausch eingesetzt werden, wobei die gesendeten Daten beim Empfänger im Logfile mit Zeitstempel abgelegt werden.

## Datenquelle

---

Die Datenübertragung kann von den folgenden drei Datenquellen erfolgen:

- Zyklische Ausgabe einer Textzeile
- Zyklische Ausgabe eines einzelnen Files
- Zufallswerte zwischen 0 und 127

Bei allen Quellen werden Blöcke einer Länge von 400 Zeichen zusammengestellt um die Kommunikations-Buffer optimal auszunutzen.

## Loginformation

---

In der Loginformation wird der gerade gesendete Block in ASCII Schreibweise dargestellt.

## Auswertung

---

Die Auswertung beinhaltet folgende drei Bereiche:

- Gesendet: Anzahl der seit dem letzten Reset gesendeten Zeichen
- Empfangen: Anzahl der seit dem letzten Reset empfangenen Zeichen
- Fehler: Anzahl der abweichenden ASCII-Werte (Gesendet/Empfangen)

## Logfile

---

Wie bereits erwähnt, wird über den Vergleich der ASCII-Zeichen ein Logbuch geführt. Das laufend ergänzte Logfile [send.log] kann über den Button *Logfile Ansicht* angeschaut werden. Der Button *Logfile Löschen* erstellt ein neues leeres File mit dem selben Namen. Es empfiehlt sich zur Zeit, keinen Button ausser *Stop* zu betätigen, während der Sendevorgang läuft, da die Echtzeitverarbeitung dadurch ins Stocken geraten kann.

## Start/Stop -- Info -- Beenden

---

- Über den Button *Start/Stop* lässt sich der Sendevorgang starten bzw. stoppen
- *Info* zeigt einen Kurzbeschrieb des Programms
- *Beenden* schliesst das Programm

Es blieb leider keine Zeit mehr, die Buttons je nach Betriebszustand zu sperren. So ist es zur Zeit noch möglich, alle Buttons zu betätigen, was im schlimmsten Falle zu einem Programmabsturz führt. Es empfiehlt sich, vor der Betätigung eines Buttons, den Sendeablauf zu stoppen. Das Programm ist gestoppt, wenn der Startbutton sichtbar ist.

## 8.7.4 Empfängeranwendung

Die Empfängeranwendung (Abbildung 8.7-3) hat die Hauptaufgabe, die empfangenen Daten blockweise mit Zeitstempel im Logfile abzulegen. Falls eine Netzwerkverbindung zum Sender-PC steht, werden die Daten dorthin für den Vergleich weitergeleitet. Die Anzahl der empfangenen Zeichen wird dabei laufend auf dem GUI angezeigt.

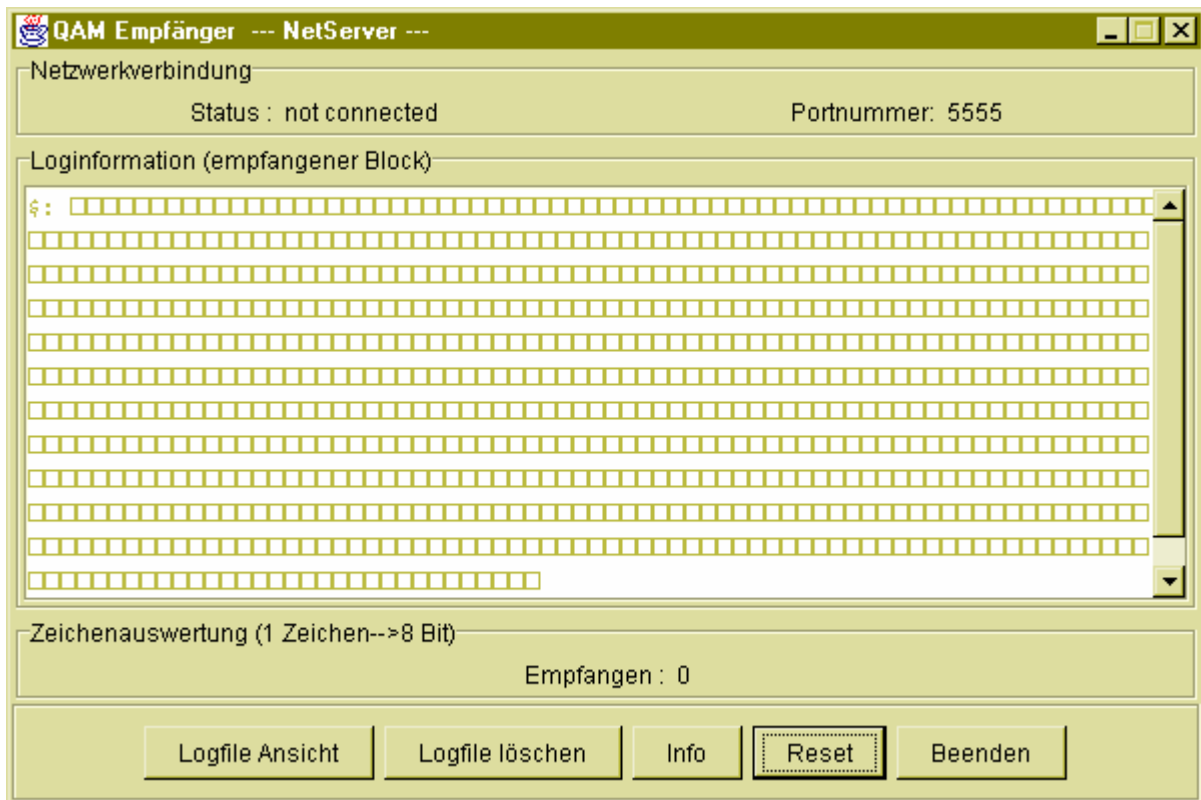


Abbildung 8.7-3 - GUI der Empfängeranwendung

### Netzwerkverbindung

Bei der Netzwerkverbindung nimmt der Empfänger den Serverstatus ein. Für diese Aufgabe wurde ein zusätzlicher Thread (Teilprozess) implementiert, der ständig auf eine Netzverbindung wartet und diese im Falle einer Anfrage des Senders (Client) bewerkstelligt. Der Netzstatus (*Connected/not Connected*) wird jeweils auf dem GUI dargestellt. Die Portnummer hat den fixen Wert von 5555. Sie muss beim Sender (Client) identisch gewählt werden.

### Loginformation

In der Loginformation wird ein gerade empfangener Block in ASCII Schreibweise dargestellt.

### Zeichenauswertung

Die Zeichenauswertung umfasst lediglich die Anzahl der empfangenen Zeichen. Ein Zeichen beinhaltet dabei 8 Bits.

## Logfile

Beim Empfänger wird jeder empfangene Block mit einem Zeitstempel in ein Logfile [receive.log] eingetragen. Dieses File kann wie beim Sender mit dem Button *Logfile Ansicht* angeschaut oder mit *Logfile löschen* zurückgesetzt werden.

## Info – Beenden

- *Info* zeigt einen Kurzbeschrieb des Programms
- *Beenden* schliesst das Programm

## 8.8 Messungen

### 8.8.1 KO-Messungen

Folgende KO-Bilder veranschaulichen die Signalformen während den verschiedenen Systemzuständen.

Das linke Bild zeigt jeweils das Signal unmittelbar am Ausgang des Senders und nach dem Kanalfilter.

- **Ch1** → ungefiltertes Ausgangssignal am Sender
- **Ch2** → gefiltertes Signal am Eingang des Empfängers
- **Ch3** → gefilterte Inphasenkomponente beim Empfänger
- **Ch4** → gefilterte Quadraturkomponente beim Empfänger

Das rechte Bild stellt jeweils die beiden Eingangssignale am Empfänger (Daten- und Synch-Signal) dar.

- **Ch1** → gefiltertes Signal am Eingang des Empfängers (Ch1)
- **Ch2** → Synchronisationssignal am Eingang des Empfängers (Ch2)
- **Ch3** → gefilterte Inphasenkomponente beim Empfänger
- **Ch4** → gefilterte Quadraturkomponente beim Empfänger

## Trägersynchronisation

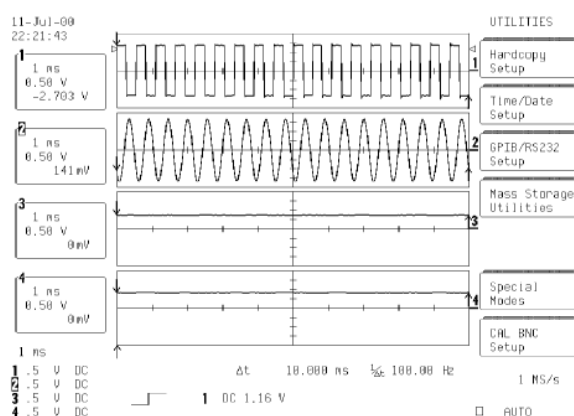


Abbildung 8.8-1 - Kanalfilter

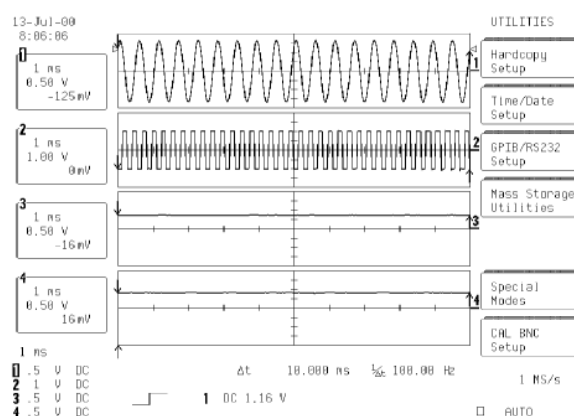


Abbildung 8.8-2 - Daten- und Synch.-Signal

Beim permanenten Senden des Raumpunktes **1111<sub>BIN</sub>** werden die Träger der Inphasen- und Quadraturkomponente überlagert, wodurch ein periodisches Signal entsteht. Der Sender dekodiert dabei diesen Raumpunkt als Koordinatenpaar 3/3 (siehe Ch3 und Ch4).

## Symbolsynchronisation

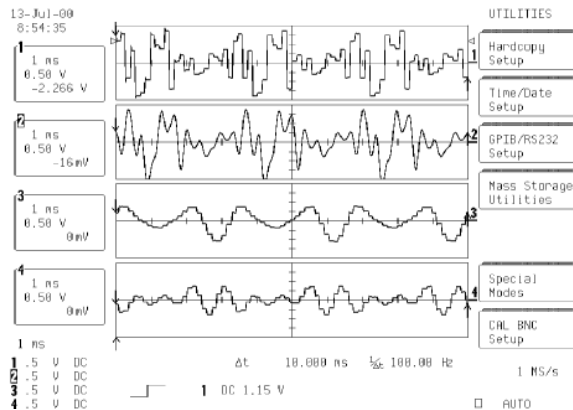


Abbildung 8.8-3 - Kanalfilter

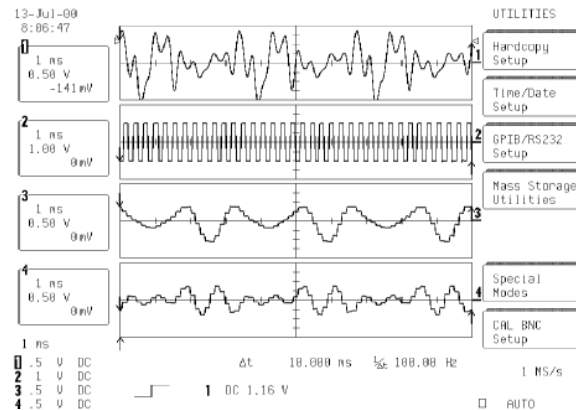


Abbildung 8.8-4 - Daten- und Synch.-Signal

Während der Synchronisation der Symbole wird eine sich wiederholende Sequenz von 8 vordefinierten Symbolen (1100|0000|1000|0100|1111|0011|1011|0111|Bin) ausgesendet.

## Datenverkehr mit Scrambler

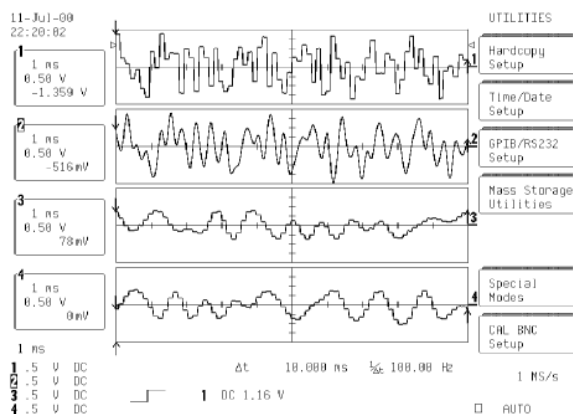


Abbildung 8.8-5 - Kanalfilter

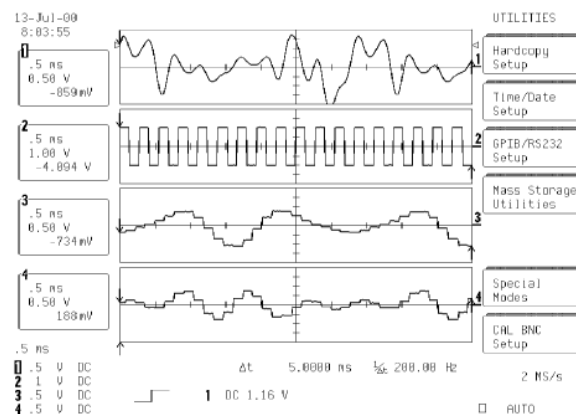


Abbildung 8.8-6 - Daten- und Synch.-Signal

Beim Datenverkehr dürfen keine periodische Sequenzen auftreten. Um dies zu bewerkstelligen, wird ein Scrambler (Verwürfler) eingesetzt.

## 8.8.2 Fehlermessung mit Testprogramm

Die Fehlermessung mit dem Testprogramm konnte leider wegen Stabilitätsmängeln bei der Datenübertragung und aus zeitlichen Gründen nicht seriös durchgeführt werden. Folgende Werte wurden bis anhin festgehalten:

- **Mit AGC beim Empfänger → bis 250'000 übermittelte Zeichen mit 0 Fehlern**  
nach 250'000 → Instabilität → Massive Anhäufung der Fehler bis Blockverratz (kein Vergleich mehr möglich)
- **Ohne AGC beim Empfänger → bis 2'500'000 übermittelte Zeichen mit einigen wenigen Fehlern**  
unterschiedliche Übertragungswerte aufgrund Synchronisation und Programm-Instabilitäten bei Testanwendung.

## 8.9 Software

### 8.9.1 Verzeichnis- und File-Struktur

#### DSP-Software (C)

..\Software\Sharc	
→ SourceCode für Sender-Algorithmus (DSP)	→ QAMsend.c
→ SourceCode für Empfänger-Algorithmus (DSP)	→ QAMrecv.c
→ Header-Files (Bitsiboard)	→ bitsi.h
	→ bitsibb.h
→ SHARC-Architektur-File	→ sharc6xc.ach
→ Konfigurationsfiles für Comptool-Anwendung (Kompilation und Download in MATLAB)	→ QAMsend.mat
	→ QAMrecv.mat
→ Batchdatei zur Kompilation der Algorithmen	→ make.bat
→ Ausführbarer Code für DSP-Download (Sender)	→ QAMsend.21k
→ Ausführbarer Code für DSP-Download (Empfänger)	→ QAMrecv.21k
→ m-Files für Test	→ send.m
	→ send1.m
	→ send2.m
	→ send3.m

#### Testanwendung (JAVA)

..\Software\Java\QAMSender	
→ Hauptprogramm Sender	→ SendMain.java
→ GUI	→ SendWindow.java
→ Fensterfunktionen	→ SendWindowListener.java
→ Sendeschleife mit Zusatzfunktionen	→ Send.java
→ Datenquelle	→ DataSource.java
→ Logfile	→ LogFile.java
→ Logausgabe	→ LogOut.java
→ Netzwerkverbindung	→ NetConnect.java
→ DSP-Zugriff (Native)	→ DSP.java
→ Projektfile (Kawa Editor)	→ QAMSend.kawa
→ DSP-Headerfile (wird aus DSP.java erstellt)	→ DSP.h
→ DSP-Native-Interface (in Visual C++ implementiert)	→ DSP.c
→ DSP-Algorithmus (gleiche Datei wie im Sharc-Verzeichnis)	→ QAMSend.21k
→ Diverse Include- und Library-Dateien	→ *.lib / *.h / ....

..\Software\Java\QAMReceiver	
→ Hauptprogramm Receiver	→ RecMain.java
→ GUI	→ RecWindow.java
→ Fensterfunktionen	→ RecWindowListener.java
→ Empfangsschleife mit Zusatzfunktionen	→ Rec.java
→ Logfile	→ LogFile.java
→ Logausgabe	→ LogOut.java
→ Netzwerkverbindung	→ NetConnect.java
→ DSP-Zugriff (Native)	→ DSP.java
→ Projektfile (Kawa Editor)	→ QAMReceiver.kawa
→ DSP-Headerfile (wird aus DSP.java erstellt)	→ DSP.h
→ DSP-Native-Interface (in Visual C++ implementiert)	→ DSP.c
→ DSP-Algorithmus (gleiche Datei wie im Sharc-Verzeichnis)	→ QAMRecv.21k
→ Diverse Include- und Library-Dateien	→ *.lib / *.h / ....

## 8.9.2 Kompilation und Ausführung der Programme

Die Kompilation und der Start der Programme erfolgt am besten von der Befehlszeile aus. Dabei sollte man sich immer im Verzeichnis befinden, wo der Sourcecode, bzw. die Anwendung abgelegt ist.

### Kompilieren

DSP-Algorithmen		
Sender	→ make QAMSend.c	→ QAMSend.21k
Empfänger	→ make QAMRecv.c	→ QAMRecv.21k
Sender-Anwendung aktualisieren	QAMSender.21k kopieren → Java\QAMSender	
Empfänger-Anwendung aktualisieren	QAMEmpfänger.21k kopieren → Java\QAMReceiver	

Testprogramm		
Sender-Anwendung aktualisieren	→ javac *.java	→ *.class
Empfänger-Anwendung aktualisieren	→ javac *.java	→ *.class
DSP.h aktualisieren (aus DSP.java)	→ makedsph.bat	zuvor Pfad mit setpath.bat setzen
DSP.c zu DSP.dll kompilieren	→ makedll.bat	zuvor Pfad mit setpath.bat setzen

### Ausführen

Testprogramm		
Sender-Anwendung	→ java SendMain	im Verzeichnis QAMSender
Empfänger-Anwendung	→ java RecMain	im Verzeichnis QAMReceiver

**Anmerkung:** Damit die Testprogramme ausgeführt werden können, muss auf beiden Rechnern der Java2 (JDK 1.2.2) Interpreter installiert sein.  
Im Falle einer Fehlerauswertung müssen beide PC's über ein LAN miteinander verbunden werden. Dazu wird der Rechnername des Empfängers benötigt.

Abbildung 8.9-1 auf der nächsten Seite, gibt eine Übersicht über den Zusammenhang der einzelnen Programmdateien und die einzelnen Übersetzungsschritte.

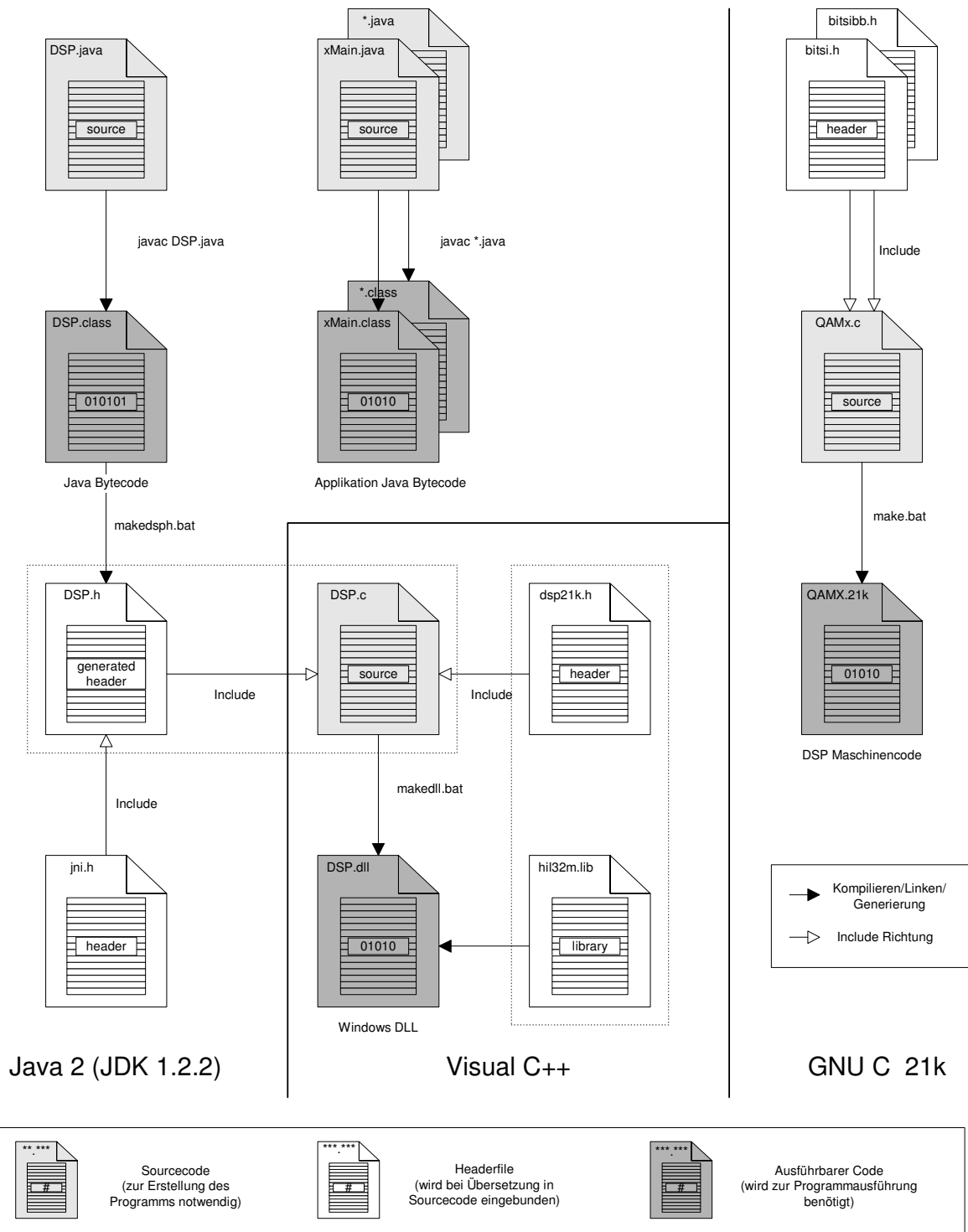


Abbildung 8.9-1 - Zusammenhänge der verwendeten Dateien

**Bemerkung:** Anstelle von x ist entweder Send oder Rec bzw. Sender oder Receiver einzusetzen.

## 9 Zusammenfassung

### 9.1 Was realisiert wurde

- Kompletter Sender und Empfänger auf Matlab/Simulink mit einer unidirektionalen Verbindung
- Automatic-Gain-Control (AGC) im Empfänger
- Diverse Simulationen und Messungen zur Bestimmung der Parameter der Filter und des AGC
- Unidirektionale Verbindung zweier SHARC-DSP's
- Implementation eines kompletten DSP-Algorithmus für den Sender
- Implementation eines DSP-Algorithmus für den Empfänger ohne AGC
- Testeinrichtung mit Datenrückführung via Netzwerk und Fehlerzählung (zeichenweise)

### 9.2 Was nicht realisiert wurde

- Genauere Charakterisierung des Modems bei der Simulation mit anderen Datenraten und Raumpunktkonfigurationen
- Einwandfreie Funktionsweise des AGC ohne Quadratwurzelberechnung im DSP sowie bei der Simulation
- Aufnahme der Bitfehlerwahrscheinlichkeitskurve in Funktion des Signal/Rauschabstandes bei der Implementation
- Button-Deaktivierung im GUI

### 9.3 Probleme

- Die empirische Ermittlung der verschiedenen Parameter des AGC waren recht langatmig und vom zeitlichen Aufwand her schwer einzuschätzen.
- Die ganze Theorie über die Signal/Geräuschverhältnisse findet man in jedem Buch anders definiert.
- Zu Beginn der Arbeit haben wir ein Buch bestellt und erst nach sehr langer Zeit die Rückmeldung bekommen, dass es unter der von uns angegebenen ISBN-Nummer vergriffen ist.
- Die in Matlab/Simulink ermittelten Parameter für den AGC konnten nicht direkt auf den DSP übernommen werden. Es benötigte wiederum lange Messungen um empirisch die Werte für den SHARC zu ermitteln.
- Aufgrund der etwas einseitigen Arbeitsteilung war auch die Intensität verschieden. Dadurch brach die Kommunikation im Team teilweise zusammen. Dies führte dazu, dass zum Beispiel A. Hüsler sich an einem Problem bei der Übertragung den Kopf zerbrach, während A. Zehnder bei seiner Simulation das doch schon gelöst hatte.
- Aufgrund eines Compilerfehlers konnten gewisse Passagen des Programms nicht kompiliert werden. Um diesen Fehler zu eruiieren wurde sehr viel Zeit benötigt (Analyse des Assembler-Zwischencodes). Um diesen Fehler zu umgehen mussten zusätzliche Variablen definiert um durch Umspeicherung gewisser Werte diesen Fehler zu umgehen.
- Um den Scrambler im DSP realisieren zu können, mussten Integerwerte bitweise nach rechts verschoben werden. Der C-Compiler unterscheidet Signed und Unsigned Datentypen und entscheidet sich je nach Typ für eine andere Schiebefunktion (logisch oder arithmetisch). Um zu dieser Erkenntnis zu kommen, wurde sehr viel Zeit aufgewendet.

## 10 Schlusswort/Fazit

Die Realisierung einer QAM-Modemverbindung erwies sich für uns im Laufe der Arbeit als aufwendiger, wie wir es zu Beginn angenommen hatten. Die ganze Theorie mit den verschiedenen Filtern hat seine Tücken, worauf im Fach DS nie wirklich eingegangen wird. Dies erforderte einen besonders hohen Aufwand, die fast nirgends aufzufindende Theorie über diese Art von Filtern zu erarbeiten. Nur schon das Zusammensuchen von Informationen nahm viel Zeit in Anspruch. Hätte man ein nützliches Buch gefunden gehabt, so war es entweder vergriffen oder ausgeliehen. Unser Betreuer, Herr Ehrensperger lieferte uns schlussendlich die benötigten Informationen, so dass wir mit unserer Arbeit nicht zu sehr ins Stocken gerieten.

Da wir eigentlich die zweite Studienarbeit im Fach SHE bestreiten wollten, uns dann aber aufgrund Platzmangels im SHE-Labor für diese Fachrichtung überreden liessen, war unsere Motivation zu Beginn der Arbeit gegen null strebend. Im Verlaufe der Arbeit jedoch, wurde bei uns beiden das Interesse an der Materie DS geweckt.

Die Einarbeitung in das Simulationsprogramm Matlab/Simulink erwies sich als sehr anstrengend, da es sehr viele Möglichkeiten bietet. Um nur einen Überblick zu bekommen, was dieses Programm alles kann, wird viel Zeit benötigt. In Simulink gibt es für fast jede erdenkbare mathematische Funktion einen Block der diese Operation ausführen kann. Um einen bestimmten Block zu finden, der eine gewisse Funktion ausführen soll, muss man sehr genau wissen worum es geht, um nicht am falschen Ort zu suchen. Nachdem ein Einblick über die Möglichkeiten des Programms verschafft werden konnte, bereiteten die verschiedenen Simulationen des Modems sogar Spass.

Was einen jedoch einige male stutzig werden liess, war zum Beispiel, dass nirgends eine genaue Zusammenstellung über die Definition des Signal/Rausch-Verhältnisses zu finden ist. In allen konsultierten Büchern fand man andere Definitionen und Notationen vor. Um dann aus all den verschiedenen Büchern und Unterlagen die wesentlichen Informationen, die man benötige herauszusuchen, wurde sehr viel Zeit benötigt.

Bei der Realisierung lag der Schwerpunkt vor allem bei der Frequenz- und Trägerphasensynchronisation. Trotz der Übertragung eines zweiten Signals zur Synchronisation, mussten einige nicht triviale Probleme gelöst werden, wodurch sich die Implementation des DSP-Algorithmus als enorm anspruchsvoll erwies. Die Datenübertragung war daneben eher einfach zu bewerkstelligen.

Es war sehr interessant, einen Einblick in die DSP-Programmierung erhalten zu haben. Die effiziente Programmierung von Echtzeitsystemen verlangt vom Programmierer einiges an Computertechnikwissen ab, was mir persönlich sehr gut gefallen hat.

Unseres Erachtens, kam in der Implementation die Digitale Signalverarbeitung bei dieser Arbeit fast etwas zu kurz. Wenn man bedenkt, dass eigentlich nur gerade ein Tiefpassfilter und Modulator bzw. Demodulator realisiert werden musste. Die ganze Datenverarbeitung war letztendlich ein programmiertechnisches Problem.

Unser Betreuer A. Ehrensperger nahm sich sehr viel Zeit, uns die DS-Materie näher zu bringen. Er verstand es, uns auch bei starken Rückschlägen wieder zu motivieren. Mit seinen Tips gelang es uns in den meisten Fällen wieder etwas Dynamik in die Arbeit zu bringen. Für seinen grosszügigen Betreuungseinsatz möchten wir Herrn Ehrensperger an dieser Stelle unseren Dank aussprechen.

Auch A. Rüegg, dem DS-Assistenten, lassen wir an dieser Stelle ein herzliches Dankeschön zukommen. Er war sehr hilfsbereit und stand uns bei Problemen zur Seite. Gerade bei der Implementation der DSP-Algorithmen, konnten wir enorm von seinem Wissen profitieren.

Herr Schuster nahm sich viel Zeit, uns die Materie über das Signal-/Geräuschverhalten und die Bitfehler-Wahrscheinlichkeit von Modems zu erläutern. Auch ihm wollen wir an dieser Stelle danken.

Rapperswil, 14. Juli 2000

Andy Zehnder:

Armin Hüsler:

## 11 Anhang

Nachstehende Zusatzinformationen werden in der selben Reihenfolge als Anhang folgen.

### 11.1 Literaturverzeichnis

Verwendete Unterlagen

---

### 11.2 m-Files der Simulationsphase

Init: File zur Initialisierung der Modemparameter

---

Mapper: Funktion zur Eruiierung des Raumpunktes

---

Biterrorrkurve: Funtktion zur graphischen Ausgabe  $BER=f(S/N)$

---

### 11.3 Sourcecode

DSP-Algorithmus des Senders

---

DSP-Algorithmus des Empfängers

---

Java Sourcecode der Sender-Testanwendung mit Visual C++ File

---

Java Sourcecode der Empfänger-Testanwendung mit Visual C++ File

---