Fachhochschule Köln
University of Applied Sciences Cologne

Fachbereich Nachrichtentechnik
Institut für Telekommunikation

Diplomarbeit

Programmierung eines Equalizers auf einem DSP-basierten Praktikumsaufbau

Student: Georg Elsenheimer
Matr. Nr.: 11002408
Referent: Prof. Dr.-Ing. Uwe Dettmar
Koreferent: Prof. Dr.-Ing. Heinrich Dederichs

Abgabedatum: 16. Oktober 2000

Hiermit versichere ich, daß ich die Diplomarbeit selbständig angefertigt und keine anderen als die angegebenen und bei Zitaten kenntlich gemachten Quellen und Hilfsmittel benutzt habe.

_____

(Name)

Inhaltsverzeichnis

1 Einleitung **1**
1.1 Aufgabenbeschreibung 1

2 Digitale Datenübertragung **2**
2.1 Die digitale Übertragungsstrecke 2
2.2 Der Kanal 2
2.3 Intersymbol-Interferenz 4
2.4 Pulsformung 6
2.4.1 Filterbestimmung 9

3 Entzerrer **11**
3.1 Zeitdiskretes Filtermodell 12
3.2 Lineare Entzerrer 14
3.2.1 Peak-distortion-Kriterium 14
3.2.2 MSE-Kriterium 16
3.2.3 Performance von MSE-Entzerrern 19
3.3 Decision-Feedback Equalizer 21
3.3.1 Performance von DFE-Equalizern 22
3.4 Adaptive Entzerrer 22
3.4.1 LMS-Algorithmus 23
3.5 Adaptiver DFE-Equalizer 25

4 Zusammenfassung **26**

5 Umsetzung **27**
5.1 In Matlab 27
5.1.1 MSE-Equalizer, mse.m 27
5.1.2 DFE-Equalizer, dfe.m 30
5.1.3 Ersatzfiltermodell, kanalmodell.m 31
5.1.4 Bitfehlerrate, msetest.m 32
5.1.5 Bitfehlerrate, dfetest.m 32
5.1.6 Vergleich MSE zu DFE, msedfe.m 34
5.2 In C 35
5.2.1 Oszilloskopausgabe MSE 39
5.2.2 Oszilloskopausgabe DFE 40

6 Hard- und Software **41**
6.1 Praktikumsaufbau 41
6.2 AD-Wandler 42
6.2.1 Fehlersuche 42
6.2.2 Fehlerbehebung 43
6.3 Software 47

7 Literaturverzeichnis **48**

A m.-files **49**
A.1 mse.m 49
A.2 dfe.m 51
A.3 kanalmodell.m 54
A.4 msetest.m 56
A.5 dfetest.m 60
A.6 msedfe.m 65

B C-Code **70**
B.1 mse.c 70
B.2 dfe.c 76

C Layout AD-Wandler **83**
C.1 Bestückung 84

D Praktikumsanleitung **85**

Abbildungsverzeichnis

2.1 Basisbandübertragung von der Quelle bis zur Senke 2
2.2 (a) idealer Kanal (b) nicht idealer Kanal 3
2.3 Amplitudenantwort und Gruppenlaufzeit eines Telefonkanals 4
2.4 Impulsantwort eines Telefonkanals 4
2.5 Augendiagramm 6
2.6 (a) si-Funktion (b) Frequenzspektrum 7
2.7 (a)RC-Funktion (b) Frequenzspektrum 8
3.1 Entzerrerauswahl 11
3.2 Blockschema eines Kanalmodells 12
3.3 Kanalmodell 12
3.4 Blockschema mit Noise-Whitening-Filter 13
3.5 Ersatzfilter mit Noise-Whitening-Filter 14
3.6 Lineares transversales Filter 15
3.7 Blockschema eines Kanals mit anschließendem *zero-forcing equalizer* 16
3.8 (a) Puls vor Equalizer (b) Equalized Puls 18
3.9 Drei verschiedene Kanalmodelle aus Proakis 19
3.10 Spektren der drei Kanäle 20
3.11 Fehlerwahrscheinlichkeit 20
3.12 Blockschema eines DFE 21
3.13 Funktion der einzelnen Filter des DFE 21
3.14 Fehlerwahrscheinlichkeit bei DFE-Equalizern 22
3.15 Adaptiver MSE-Equalizer 24
3.16 Adaptiver DFE-Equalizer 25
4.1 Entzerrerauswahl 26
5.1 Kanäle 29
5.2 Amplitudenspektren der Kanäle 29
5.3 Ausgabe für mse(1,0,20) 30
5.4 Ausgabe für dfe(3,0,20) 31
5.5 Ausgabe msetest 32
5.6 Ausgabe dfetest 33
5.7 Ausgabe msedfe 34
5.8 Meßpunkte MSE 39
5.9 Ausgabe mse.c 39
5.10 Meßpunkte DFE 40
5.11 Ausgabe dfe.c 40
6.1 Schematischer Aufbau des ADSP 21061. 41
6.2 Layout EZ-KIT Lite. 42
6.3 OPV NE5534 43
6.4 Auszug aus Datenblatt AD8042 (Versorgungsspannung + 5V) 44
6.5 Timing (richtig für DA-, falsch für AD-Wandler) 44
6.6 Schaltplan AD-Wandler mit Änderungen 45
6.7 Schaltplan AD-Wandler 46
C.8 Layout Oberseite 83
C.9 Layout Unterseite 83
C.10 Bestückung Oberseite 84

C.11 Bestückung Unterseite 84

1 Einleitung

1.1 Aufgabenbeschreibung

Aufgabe dieser Diplomarbeit ist es, Equalizer zu berechnen und auf einem DSP zu programmieren. Zwei verschiedene Arten von Equalizern, ein linearer Equalizer bzw. ein Decision-Feedback-Equalizer, sollen auf dem DSP-Board implementiert werden.
Zum Thema $Equalizer$ soll anschließend ein Praktikumsversuch ausgearbeitet werden.

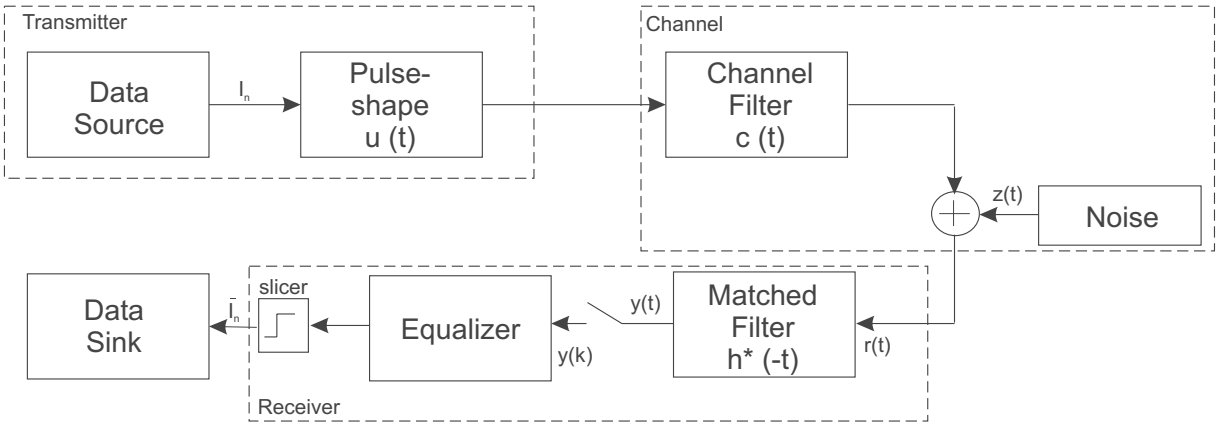


Abbildung 2.1: *Basisbandübertragung von der Quelle bis zur Senke*

2 Digitale Datenübertragung

2.1 Die digitale Übertragungsstrecke

Wie in Abb. 2.1 zu sehen ist, besteht eine digitale Übertragungsstrecke aus drei wesentlichen Komponenten.

Sie besteht aus einem Sender (Transmitter), dem Kanal (Channel) und dem Empfänger (Receiver). Den Eingang des Senders bildet die zeit- und wertdiskrete Wertefolge I_n. Im Transmitter werden aus dieser Wertefolge zu übertragene Pulsformen generiert. Diese Pulse werden dann über einen Kanal an den Empfänger gesendet. Bei dieser Übertragung kommt es dazu, daß die Pulse durch die Kanaleigenschaften verzerrt und verrauscht werden. Die Aufgabe des Empfängers besteht darin, diese Störungen so zu kompensieren, daß am Ausgang des Empfängers wieder die gleiche Wertefolge vorhanden ist.

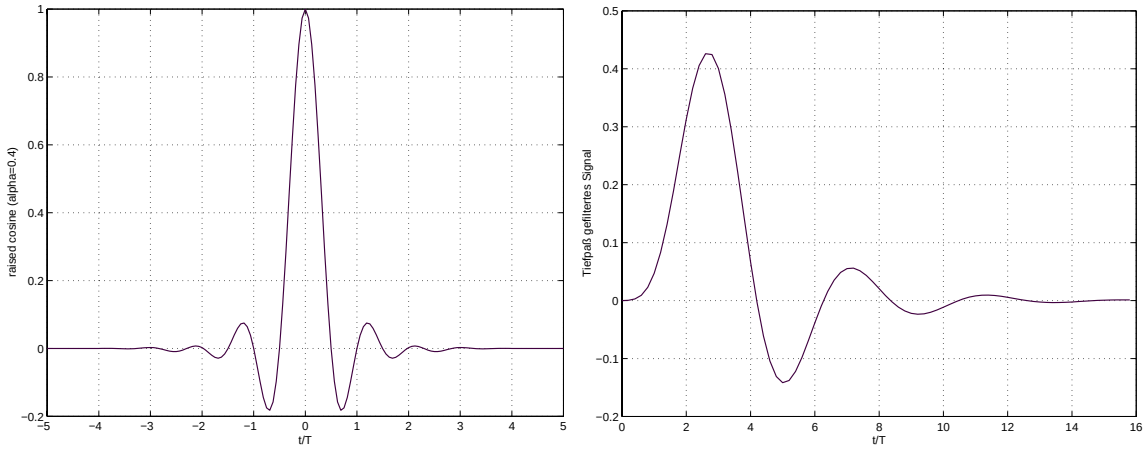
2.2 Der Kanal

Wie oben beschrieben führt der Kanal bzw. die Kanaleigenschaft zu Verzerrungen der übertragenen Signale. Deshalb sollen hier zunächst die wichtigsten Kanaleigenschaften beschrieben werden.
Jeder Übertragungskanal ist bandbegrenzt, d. h. er besitzt eine Bandbreite (z. B. Frequenzbereich, in dem Signale bis zu einem Amplitudenabfall von 3dB übertragen werden können) von W Hz. Viele dieser Kanäle, z. B. Telefonkanäle, können als lineare Filter mit Tiefpaßcharakteristik und der Frequenzantwort

$$C(f) = A(f) * e^{j\phi(f)} \tag{2.1}$$

beschrieben werden. Wobei $A(f)$ die Amplitudenantwort und $\phi(f)$ die Phasenantwort darstellt. Für die Phasenantwort wird auch häufig die Gruppenlaufzeit $\tau(f)$ angegeben.

$$\tau(f) = -\frac{1}{2\pi}\frac{d\phi(f)}{df} \tag{2.2}$$



(a) (b)

Abbildung 2.2: *(a) idealer Kanal (b) nicht idealer Kanal*

Ein Kanal gilt dann als ideal, wenn $A(f)$ und $\tau(f)$ konstant sind. Wenn der Kanal jedoch nicht diese idealen Eigenschaften besitzt, verzerrt er das Signal sowohl in der Amplitude als auch in der Phase.

In Bild 2.2 ist die Auswirkung des Kanals auf einen Empfangsimpuls zu sehen. Der Empfangsimpuls hat bei idealem Kanal periodische Nullstellen bei $\pm k \cdot T$ mit $k \neq 0$. Überträgt man solch eine Pulsfolge mit $f = 1/T$, so würde im Abtastzeitpunkt jeweils nur ein Impuls zum Abtastwert beitragen und es käme so zu keinen Überlagerungen benachbarter Signale. An der Kanalantwort kann man jedoch sehen, daß die Nullstellen nun nicht mehr periodisch angeordnet sind und es daher zu Überlagerungen von benachbarten Pulsen kommt. Die Folge ist, daß die Pulse an den Abtastzeitpunkten nicht mehr eindeutig voneinander zu unterscheiden sind.

Dieses Problem bei der Datenübertragung nennt man *Intersymbol-Interferenz* oder auch kurz ISI.

Desweiteren addiert sich zu dem Signal noch thermisches, durch Übersprechen oder elektromagnetische Einstreuung hervorgerufenes Rauschen. Wie später noch gezeigt wird, liegt ein Augenmerk im Empfänger darauf, das Signal-Rauschverhältnis zu optimieren.

Um eine Größenvorstellung von Intersymbol-Interferenz zu bekommen, ist in Abb. 2.3 die Amplitudenantwort und Gruppenlaufzeit eines üblichen Telefonkanals in Abhängigkeit von der Frequenz dargestellt. Man erkennt die starken Amplituden und Laufzeitverzerrungen bei tiefen und hohen Frequenzen. Abb. 2.4 zeigt die Impulsantwort eines solchen Kanals. Es ist zu sehen, daß erst nach ca. 10ms die Impulsantwort völlig abgeklungen ist. Angenommen es werden Daten mit einer Symbolrate von 2500 Symbolen/s über einen solchen Kanal übertragen, so erstreckt sich die Intersymbol-Interferenz über 20-30 Symbole.

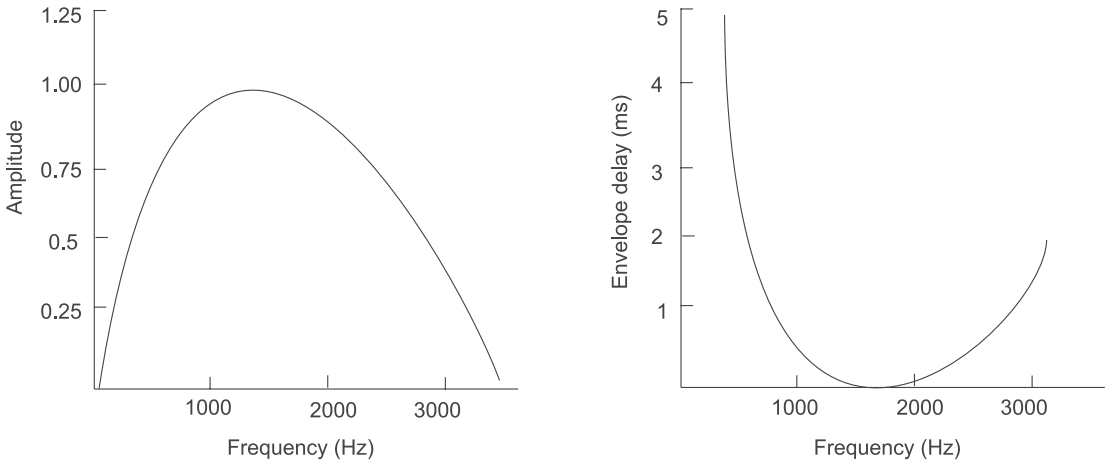


Abbildung 2.3: *Amplitudenantwort und Gruppenlaufzeit eines Telefonkanals*

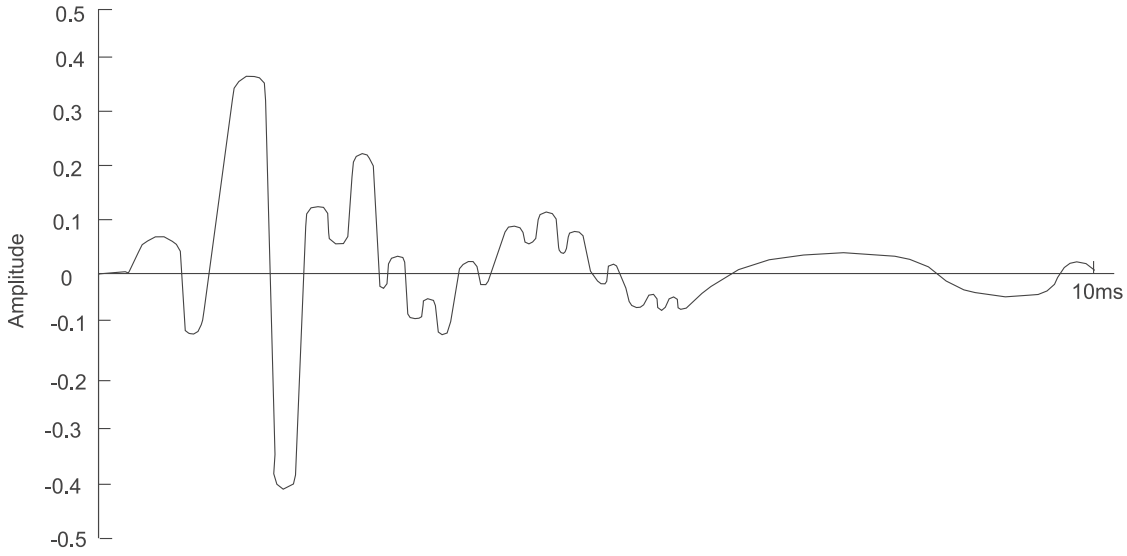


Abbildung 2.4: *Impulsantwort eines Telefonkanals*

2.3 Intersymbol-Interferenz

In Abschnitt 2.2 wurde auf anschauliche Weise das Entstehen und die Auswirkungen von Intersymbol-Interferenz dargestellt. Hier soll nun Intersymbol-Interferenz mathematisch beschrieben und definiert werden.

Am Eingang der Strecke liegt die zeitdiskrete Wertefolge I_n. Wird diese Wertefolge über ein Sendefilter mit der Impulsantwort g(t) geschickt, so erhält man:

$$\nu(t) = \sum_{n=0}^{\infty} I_n g(t - nT) \tag{2.3}$$

Nach Durchlaufen des Kanals ist das Signal am Empfänger

$$r(t) = \sum_{n=0}^{\infty} I_n h(t - nT) + z(t) \tag{2.4}$$

wobei h(t) die Impulsantwort aus Sender und Kanal darstellt.

$$h(t) = \int_{-\infty}^{\infty} g(\tau)c(t - \tau)d\tau \tag{2.5}$$

z(t) steht für das additiv hinzugekommene weiße gauß'sche Rauschen.
Wie bereits angedeutet, ist es für das weitere Vorgehen wichtig, das S/N-Verhältnis zu optimieren. Um im Empfänger ein optimales Signal/Rausch-Verhältnis zu erhalten, setzt man an den Anfang ein sogenanntes Matched-Filter. Wie der Name schon sagt, handelt es sich hierbei um ein signalangepaßtes Filter. In diesem Fall ist es auf die Sende-Kanal-Impulsantwort $h(t)$ angepaßt und besitzt die Übertragungsfunktion $h^*(-t)$.
Damit wird der Ausgang des Matchedfilters zu

$$y(t) = \sum_{n=0}^{\infty} I_n x(t - nT) + \nu(t) \tag{2.6}$$

wobei $x(t)$ die Antwort des Matchedfilters auf den Eingangsimpuls $h(t)$ darstellt. $\nu(t)$ ist die Antwort des Filters auf das Eingangsrauschen $z(t)$.
Das Signal wird jetzt zu den Zeiten $t = k \cdot T$ abgetastet.

$$y(kT) = y_k = \sum_{n=0}^{\infty} I_n x(kT - nT) + \nu(kT) \qquad k = 0, 1, ... \tag{2.7}$$

oder

$$y_k = \sum_{n=0}^{\infty} I_n x_{k-n} + \nu(k) \qquad k = 0, 1, ... \tag{2.8}$$

Zieht man aus der Summe $(x_0 \cdot I_k)$ heraus, so erhält man

$$y(k) = x_0(I_k + \frac{1}{x_0} \sum_{n=0, n\neq k}^{\infty} I_n x_{k-n}) + \nu_k \tag{2.9}$$

Wenn wir x_0 zu eins setzen, ergibt sich für das abgetastete Ausgangssignal

$$y(k) = I_k + \sum_{n=0, n\neq k}^{\infty} I_n x_{k-n} + \nu_k \tag{2.10}$$

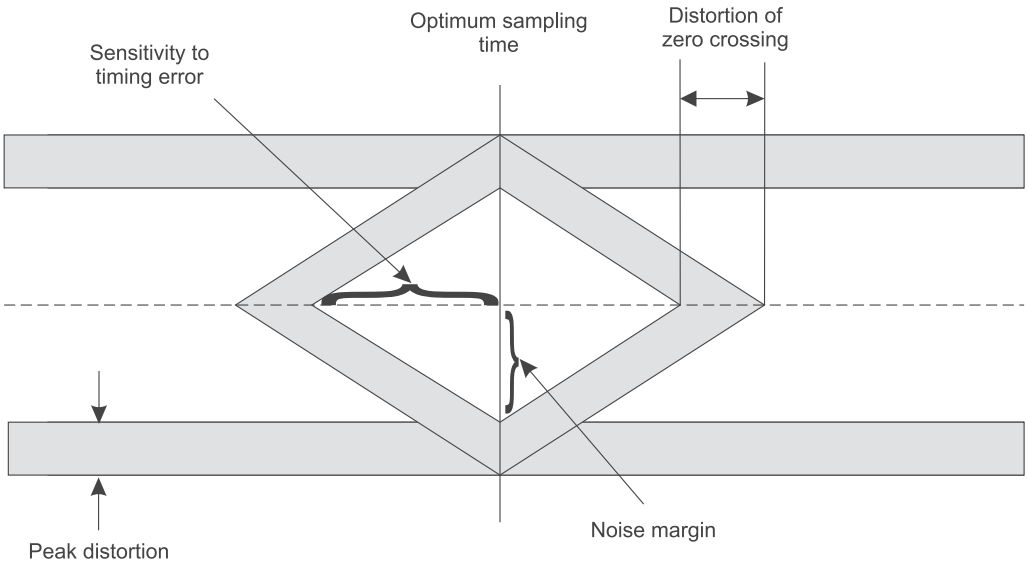


Abbildung 2.5: *Augendiagramm*

Hierbei stellt I_k das gewünschte Symbol am k-ten Abtastzeitpunkt dar, während die Summe

$$\sum_{n=0,n\neq k}^{\infty} I_n x_{k-n} + \nu_k \tag{2.11}$$

für die Intersymbol-Interferenz steht. ν_k steht für das additive gauß'sche Rauschen.
Die Auswirkungen von ISI und additivem Rauschen lassen sich anhand von sogenannten Augendiagrammen auf dem Oszilloskop darstellen. Stärkere Intersymbol-Interferenz und Rauschen drückt sich dadurch aus, daß sich das $Auge$ immer weiter schließt und somit eine eindeutige Trennung der Pulse nicht mehr möglich ist. Abb 2.5 zeigt die schematische Darstellung eines solchen Augendiagramms.

2.4 Pulsformung

Es soll untersucht werden, welche maximale Übertragungsrate über einen Kanal mit der Bandbreite W möglich ist, **ohne** das Auftreten von Intersymbol-Interferenz.
Nyquist hat sich genau mit dieser Thematik schon vor ca. 75 Jahren befaßt. Er wollte eine Pulsform $X(f)$ finden, mit $X(f) = G(f)C(f)R(f)$, bei der zu den Abtastzeitpunkten $t = k \cdot T$ keine ISI auftritt, wobei G(f), C(f) und R(f) jeweils die Frequenzantworten von Sender, Kanal und Empfangsfilter darstellen.
Das Ausgangssignal des Matched-Filters ist gegeben durch

$$y(k) = I_k + \sum_{n=0,n\neq k}^{\infty} I_n x_{k-n} + \nu_k \tag{2.12}$$

Wird ISI-Freiheit gefordert, so muß die Summe, welche die Intersymbol-Interferenz darstellt, Null ergeben. Daraus folgt die Bedingung

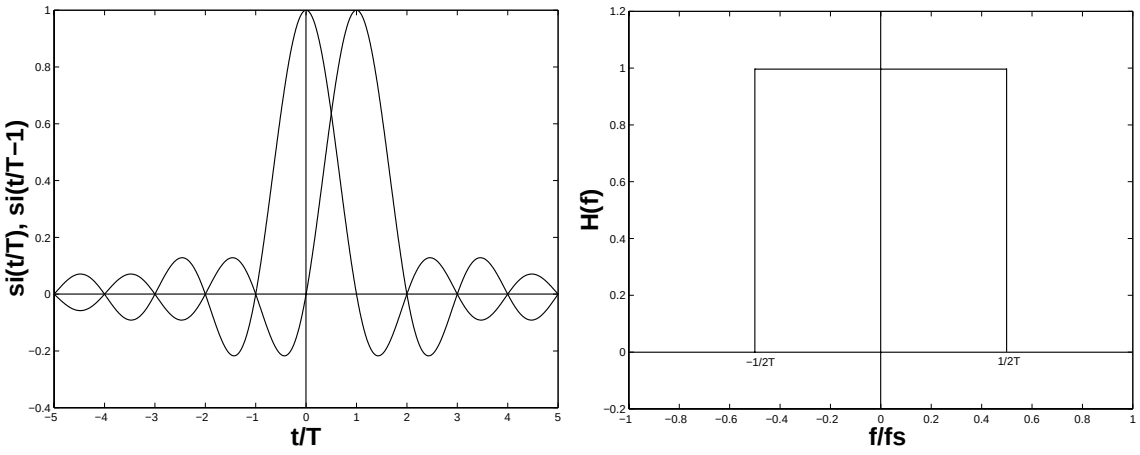
$$x(nT) = x_k = \begin{cases} 1 & (n = 0) \\ 0 & (n \neq 0) \end{cases} \tag{2.13}$$

Für die Fouriertransformierte von $x(t)$ muß dann gelten

$$\sum_{m=-\infty}^{\infty} X(f + \frac{m}{T}) = T \tag{2.14}$$

Nyquist hat gezeigt, daß mit einer rechteckigen Übertragungsfunktion $X(f)$, wie in Abb. 2.6b zu sehen ist, die maximale Übertragungsrate von $2W = 1/T = R_s symbol/s$ ohne ISI möglich ist. Diese Frequenz wird auch $Nyquistfrequenz$ genannt.
Für $x(t)$ ergibt sich in diesem Fall (Abb. 2.6a)

$$x(t) = \frac{\sin(\pi t/T)}{\pi t/T} = \text{sinc}(\frac{\pi t}{T}) \tag{2.15}$$



(a) (b)

Abbildung 2.6: *(a) si-Funktion (b) Frequenzspektrum*

Man erkennt, daß zu den Abtastzeitpunkten $t = k \cdot T$ die Symbole eindeutig unterscheidbar sind, da in den Abtastzeitpunkten $k \cdot T$ immer nur ein Puls zum Abtastwert beiträgt und die anderen Pulse dort ihre Nullstellen besitzen.
Ziel ist es nun, die benötigte Bandbreite eines Systems so weit wie möglich zu reduzieren, wobei die Nyquistfrequenz eine untere Grenze darstellt. Arbeitet man mit kleineren Bandbreiten als Nyquist vorschreibt, kommt es zu Fehlern bei der Übertragung, hervorgerufen durch

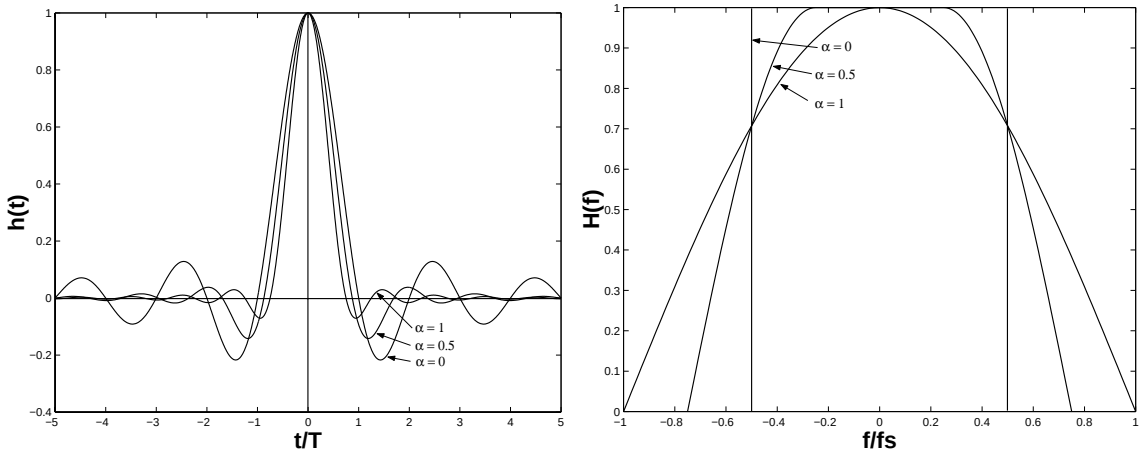
Intersymbol-Interferenz.
Die Nyquistbedingung für die zu empfangene Pulsform $x(t) = sinc(\pi t/T)$ ist aber physikalisch nicht realisierbar, da Filter mit solcher Frequenzcharakteristik nicht möglich sind. Deshalb mußten andere Pulsformen gefunden werden, die ähnliche Eigenschaften haben, wie die des $Nyquistimpulses$.
Eine Pulsform, mit einem Spektrum ähnlich des des Nyquistimpulses und $T > 1/2W$, welche in der Praxis oft Anwendung findet, ist der *raised-cosine-Impuls*.
Er ist definiert durch

$$H(f) = \begin{cases} \sqrt{\frac{E}{T}} & \text{für } 0 < f < \frac{(1-\alpha)}{2T} \\ \sqrt{\frac{E}{T}} \cos^2\left(\frac{\pi T}{2\alpha}\left(f - \frac{(1-\alpha)}{2T}\right)\right) & \text{für } \frac{(1-\alpha)}{2T} \leq f \leq \frac{(1+\alpha)}{2T} \\ 0 & \text{für } f > \frac{(1+\alpha)}{2T} \end{cases} \tag{2.16}$$

wobei α der sogenannte *Roll-off-Faktor* ist, dessen Werte zwischen 0 und 1 liegen können. Für $\alpha = 0$ ergibt sich die bekannte rechteckige Frequenzantwort. Abb. 2.7 zeigt den *raised-cosine-Impuls* im Zeit- und Frequenzbereich für verschiedene Werte von α.
Für den Fall $\alpha = 0$ wird zwar die geringste Bandbreite benötigt, aber man sieht, daß dieser Fall eine größere Zeitsensivität bzgl. des Abtastzeitpunktes mit sich bringt als bei Werten von $\alpha > 0$. Dies liegt daran, daß bei $\alpha = 0$ x(t) nur mit 1/t abfällt. Bei $\alpha > 0$ geschieht dies mit $1/t^3$. Da, wie gesagt, der $Nyquistimpuls$ physikalisch nicht realisierbar ist, reduziert sich auch damit die max. Übertragungsrate ohne das Auftreten von ISI.
Sie kann jetzt mit Hilfe des *Roll-off-Faktors* und der Bandbreite bestimmt werden zu

$$R_s = \frac{2W}{1+\alpha} \tag{2.17}$$

Für $\alpha = 0$ erhält man wieder die $Nyquistfrequenz$.



(a) (b)

Abbildung 2.7: *(a)RC-Funktion (b) Frequenzspektrum*

Beispiel zur Abschätzung der benötigten Bandbreite zur Übertragung von Sprache:

Es soll Sprache mit Frequenzanteilen bis ca. 3kHz über eine Telefonleitung übertragen werden. Einmal sollen die Sprachsignale direkt als analoges Signal, zum Anderen als digitalisiertes Signal übertragen werden.
Dazu wird das Signal mit 8kHz abgetastet und anschließend quantisiert (8-Bit Quantisierung).
Für die benötigte Bandbreite zur Übertragung ohne ISI gilt:

$$W \geq \frac{R_s}{2} \tag{2.18}$$

Für den Fall des abgetasteten Signals ergibt sich die benötigte Bandbreite:

$$W \geq \frac{(8bit/s) * (8000Symbols/s)}{2} \geq 32kHz \tag{2.19}$$

Für die reine analoge Übertragung benötigt man in etwa 4kHz (3kHz + einem gewissen Sicherheitsabstand zu benachbarten Kanälen). Die digitale Übertragung der Sprache erfordert also in diesem Fall eine um den Faktor 8 größere Bandbreite als die analoge Übertragung (bei binärer Übertragung und ohne Datenkompression).

2.4.1 Filterbestimmung

Anhand der bisher erlangten Erkenntnisse ist es nun möglich praktikable Filter zu bestimmen. Gehen wir zunächst von dem Fall des idealen Kanals mit $C(f) = 1, \mid f \mid \leq W$ aus.
Daraus folgt

$$X(f) = G_T(f)G_R(f) \tag{2.20}$$

wobei $X(f)$ die gewünschte Frequenzantwort, für *Null-ISI* darstellt. Im Fall des idealen Kanals ist $G_R(f) = G_T(f) = \mid G_T(f) \mid^2$.
Damit läßt sich die Aussage über das Sendefilter $G_T(f)$ herleiten zu

$$G_T(f) = \sqrt{\mid X(f) \mid} e^{-j2\pi ft} \tag{2.21}$$

Nun wird die Bedingung eines idealen Kanals fallen gelassen und eine Bestimmungsgleichung der Filter für diesen Fall bestimmt. Danach ändert sich die Ausgangsgleichung zu

$$X(f) = G_T(f)C(f)G_R(f) \tag{2.22}$$

Als Bestimmungsgleichung für $G_T(f)$ und $G_R(f)$ erhält man für den Fall von *Null-ISI (raised-cosine-Spektrum)*

$$G_R(f) = K_1 \frac{\mid X_{RC}(f) \mid^{1/2}}{\mid C(f) \mid^{1/2}} \qquad \mid f \mid \leq W, \quad \mid C(f) \mid > 0 \tag{2.23}$$

$$G_T(f) = K_2 \frac{\mid X_{RC}(f) \mid^{1/2}}{\mid C(f) \mid^{1/2}} \qquad \mid f \mid \leq W, \quad \mid C(f) \mid > 0 \tag{2.24}$$

wobei K_1 und K_2 beliebige Skalierfaktoren sind.
Es ist noch zu beachten, daß in diesem Fall $G_R(f)$ das auf $G_T(f)$ angepaßte Filter ist.

3 Entzerrer

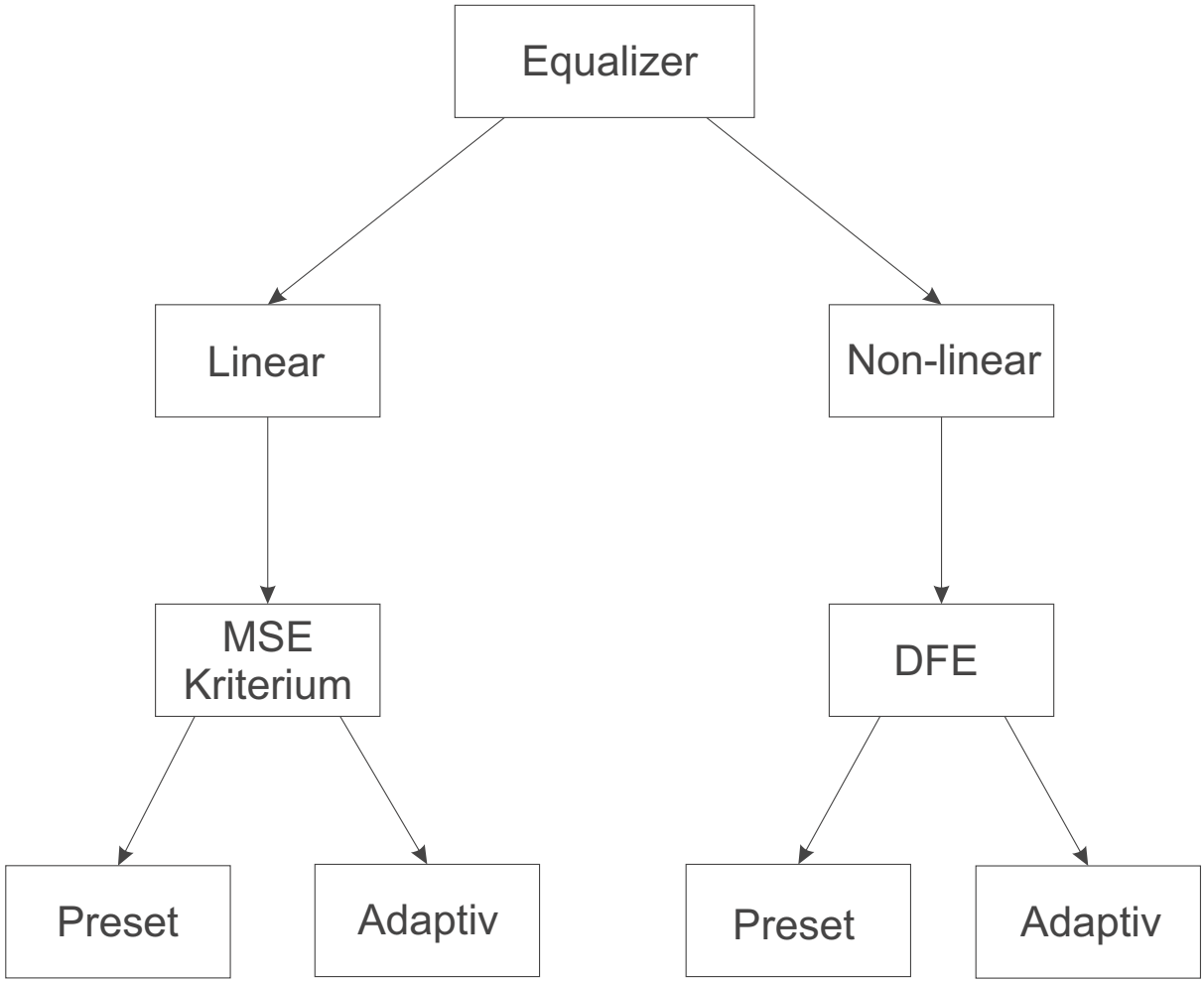


Abbildung 3.1: *Entzerrerauswahl*

Bei der bisherigen Betrachtung und Herleitung der Bestimmungsgleichung für Sende- und Empfangsfilter wurde immer angenommen, daß der Kanal mit seiner Frequenzantwort $C(f)$ bekannt ist. Die Regel ist aber, daß der Kanal und damit auch seine Frequenzantwort nicht ausreichend genau bekannt ist, so daß man die Filter nach obigen Prinzip nicht bestimmen kann.
Hier soll nun dargestellt werden, wie ein Empfangsfilter bestimmt werden kann, der die durch Kanaleinflüsse hervorgerufene Intersymbol-Interferenz kompensiert. Solch ein Filter nennt man $Entzerrer$ oder auch $Equalizer$.
Equalizer werden in zwei große Gruppen, die linearen und die nichtlinearen Equalizer, unterschieden. Hier sollen jeweils einer jeder Kategorie beschrieben und dimensioniert werden. Bild 3.1 zeigt die Spezifizierung der einzelnen Entzerrer.
Bei der Koeffizientenbestimmung der Filter unterscheidet man noch zwei verschiedene Methoden. Zum Einen werden die Koeffizienten einmal bestimmt und danach nicht mehr geändert. Solche Entzerrer nennt man $preset\ Equalizer$. Sie kommen dort zum Einsatz, wo sich der Übertragungskanal nicht zwischen den einzelnen Datenübertragungen verändert.
Die andere Methode zur Koeffizientenbestimmung nennt man $adaptiv$. Dort passen sich die Koeffizienten den ständig wechselnden Kanaleigenschaften automatisch an. Dieses Verfahren ist überall dort nötig, wo der Kanal lineares zeitvariantes Verhalten hat, wie es z. B. bei der

Telefonie vorkommt. Der Weg, den die Signale von einem zum anderen Teilnehmer nehmen, kann hierbei ständig variieren.
Die beiden Verfahren werden noch bei der expliziten Bestimmung der Koeffizienten genauer dargestellt.

3.1 Zeitdiskretes Filtermodell

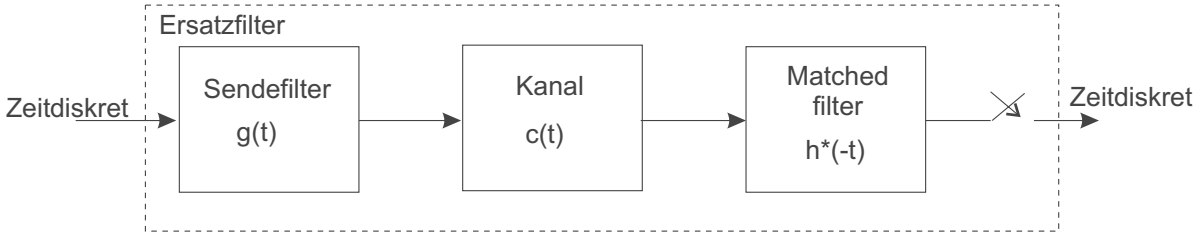


Abbildung 3.2: *Blockschema eines Kanalmodells*

Man bedient sich nun eines Hilfsmittels, um die Koeffizienten des Entzerrers berechnen zu können. Es wird ein sogenanntes *zeitdiskretes Filter- oder auch Kanalmodell* bestimmt. Dabei bildet man aus Sendefilter, Kanal und Empfangsfilter (Matched-Filter) mit anschließender Abtastung ein Ersatzfilter mit Koeffizienten x_k mit $\mid k \mid \geq L$.

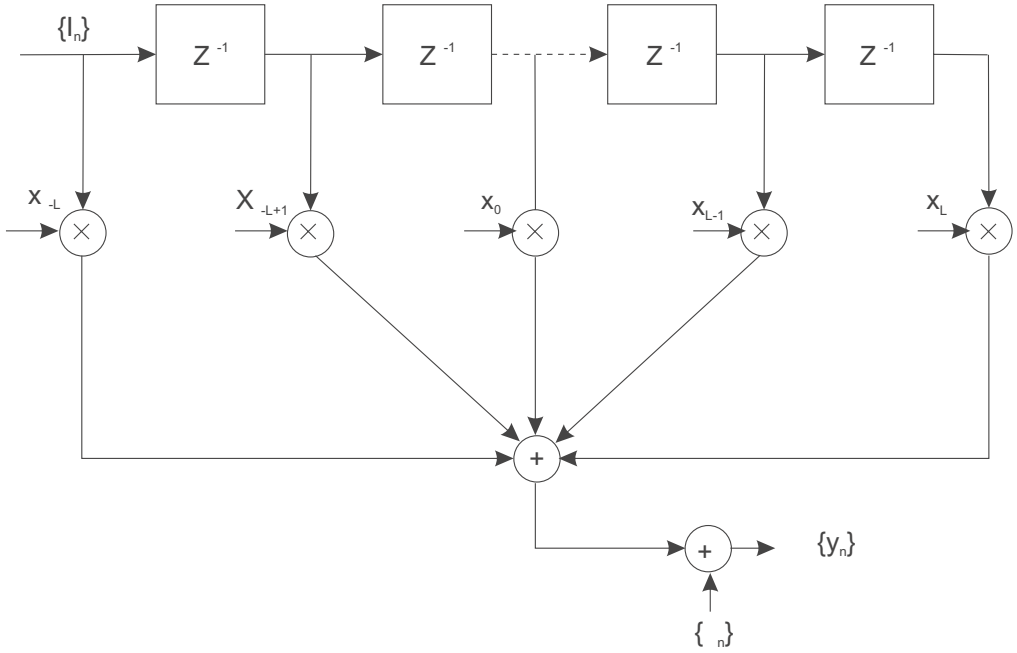


Abbildung 3.3: *Kanalmodell*

Bild 3.3 zeigt ein zeitdiskretes Filtermodell eines Kanals mit Intersymbol-Interferenz. Den Eingang bildet die Informationssequenz I_k und der Ausgang ist gegeben durch das abgetastete Ausgangssignal des Matchedfilters (2.8).
Mit Hilfe dieses Modells kann man nun die Berechnung der Koeffizienten durchführen. Es gibt aber noch eine Problematik zu beachten, die mit dem Rauschen und Matchedfilter zusammenhängt. Wie schon beschrieben, handelt es sich bei dem Kanalrauschen um weißes gauß'sches

Rauschen. Nach Durchlaufen des Matchedfilters hat das Rauschen diese Eigenschaft nicht mehr. Es ist nun farbiges Rauschen, d. h. es ist correliert, benachbarte Rauschwerte sind nicht mehr unabhängig. Dieses würde sich, wenn nicht geeignete Maßnahmen getroffen würden, in einer Performanceverschlechterung des Equalizers ausdrücken. Allerdings hält sich diese Beeinträchtigung in Grenzen. Ist sie aber dennoch inakzeptabel, so kann das Problem mit einem sogenannten *Noise-Whitening-Filter* beseitigt werden. Wie der Name schon sagt, sorgt dieses Filter dafür, daß das farbige wieder in weißes Rauschen gewandelt wird. Man kommt schließlich zu einem etwas anderem Kanalmodell, wie das Blockschema in Abb. 3.4 zeigt.

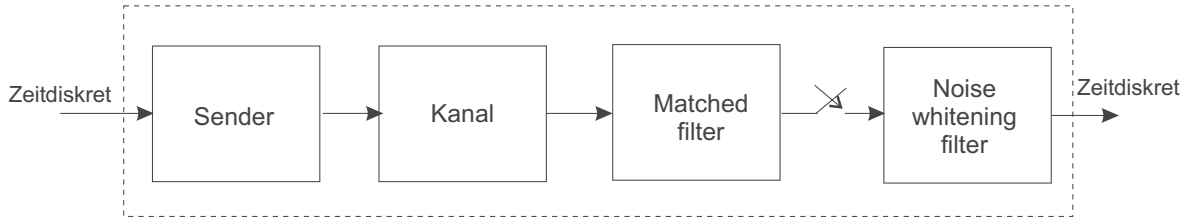


Abbildung 3.4: *Blockschema mit Noise-Whitening-Filter*

Das Noise-Whitening-Filter kann folgendermaßen bestimmt werden:
Sei $X(z)$ die zweiseitige Z-transformierte der abgetasteten Autocorrelationsfunktion x_k.

$$X(z) = \sum_{k=-L}^{L} x_k z^{-k} \tag{3.25}$$

Da $x_k = x^*_{-k}$ gilt (Autokorrelationsfunktion), besitzt $X(z)$ 2L Nullstellen, die die Symmetrie aufweisen, daß wenn p eine Nullstelle ist, auch $1/p^*$ einen Nullstelle darstellt. Damit kann nun $X(z)$ beschrieben werden mit

$$X(z) = F(z)F^*(z^{-1}) \tag{3.26}$$

wobei $F(z)$ ein Polynom in L mit den Nullstellen $p_1, p_2, ..., p_L$ und $F^*(z^{-1})$ ein Polynom in L mit Nullstellen $1/p^*{}_1, 1/p^*{}_2, ..., 1/p^*{}_L$ darstellt.
Nun läßt sich ein Noise-Whitening-Filter beschreiben durch seine Z-transformierte $1/F^*(z^{-1})$. Um an die Koeffizienten zu kommen, muß man sich noch um die Sortierung der Nullstellen kümmern. Es existieren 2^L Möglichkeiten für die Nullstellen von $F^*(z^{-1})$. Ein Stabilitätskriterium für Filter ist nun, daß sie ihre Polstellen im Einheitskreis haben. Auf diesem Hintergrund wählen wir die Nullstellen von $F^*(z^{-1})$ so, daß sie alle im Einheitskreis liegen. So besitzt die Funktion $1/F^*(z^{-1})$, wie gewollt, ihre Polstellen im Einheitskreis.
Durchläuft nun die bekannte Sequenz y_k (abgetastete Sequenz des Matchedfilterausgangs) das Noise-Whiteningfilter $1/F^*(z^{-1})$, erhält man eine neue Sequenz v_k

$$v_k = \sum_{n=0}^{L} f_n I_{k-n} + \eta_k \tag{3.27}$$

wobei η_k wieder eine weiße gauß'sche Rauschsequenz und f_k die Filterkoeffizienten des neuen diskreten Filtermodells mit der Übertragungsfunktion $F(z)$ darstellen.

Abb. 3.5 zeigt das zeitdiskrete Ersatzfilter mit Noise-Whitening-Filter.

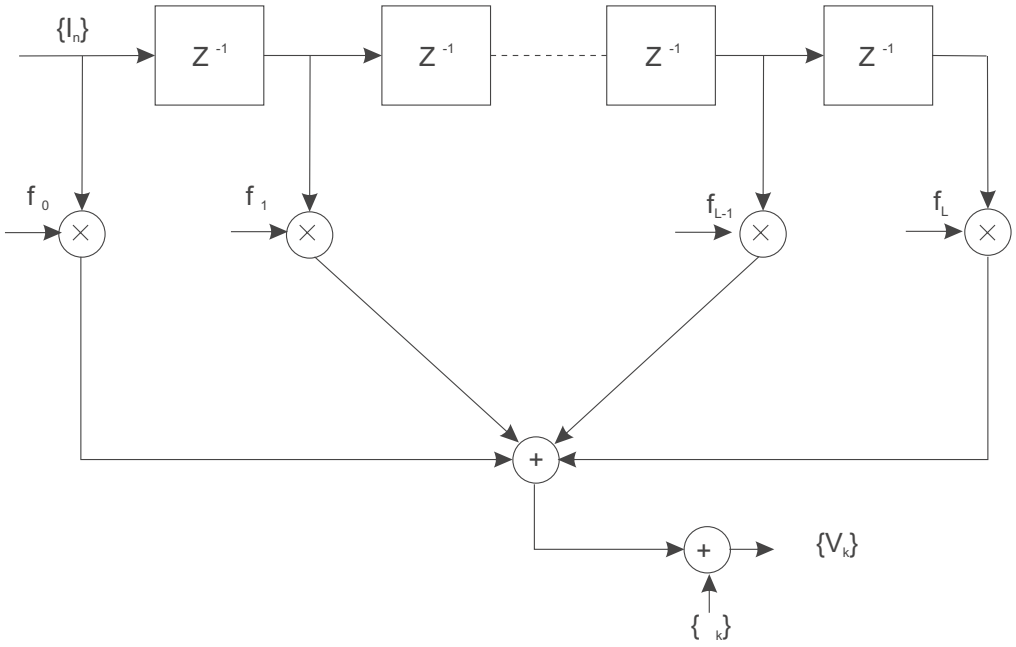


Abbildung 3.5: *Ersatzfilter mit Noise-Whitening-Filter*

3.2 Lineare Entzerrer

Nach dem bisherigen Vorgehen haben wir Erkenntnisse über die Ersatzfiltermodelle, sowohl des einfachen Ersatzfilters mit Koeffizienten x_k als auch über das Ersatzfilter mit NW-Filter und Koeffizienten f_k gewonnen. Nun soll daraus ein Equalizer als lineares transversales Filter beschrieben und seine Koeffizienten berechnet werden. Die Struktur eines solchen linearen Filters ist in Abb. 3.6 zu sehen.
Den Eingang bildet die Sequenz, gegeben durch (3.36). Der Ausgang läßt sich beschreiben durch

$$\hat{I_k} = \sum_{j=-K}^{K} c_j v_{k-j} \tag{3.28}$$

wobei c_k die $2K+1$ Koeffizienten des Equalizers sind. Man wählt $2K+1 \geq L$ mit $L = Ersatzfilterlänge$.
Es gibt nun zwei Verfahren, welche mit verschiedenen Ansätzen versuchen, die optimalen Filterkoeffizienten c_k zu finden. Es handelt sich dabei um das Verfahren des *peak-distortion-Kriteriums* sowie das *Mean-square-error-Kriterium*.

3.2.1 Peak-distortion-Kriterium

Dieser Ansatz geht von der größt möglich auftretenden Intersymbol-Interferenz am Ausgang des Equalizers aus. Die Aufgabe besteht darin, ISI in diesem *worst-case*-Fall zu minimieren.
Um dies zu erreichen, bildet man zuerst aus dem diskreten Ersatzfilter mit Koeffizienten $\{f_k\}$ und dem Equalizer mit Koeffizienten $\{c_k\}$ wiederum ein Ersatzfilter, das nun die Gesamtimpulsantwort beschreibt.

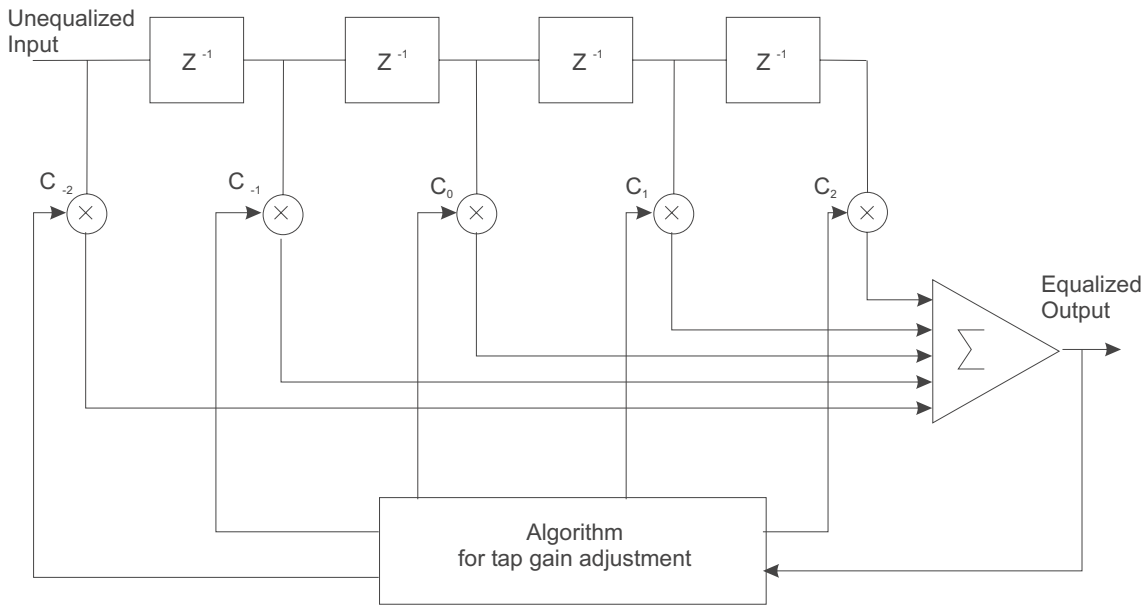


Abbildung 3.6: *Lineares transversales Filter*

$$q_n = \sum_{j=-\infty}^{\infty} c_j f_{n-j} \tag{3.29}$$

Damit kann nun der Ausgang des Equalizers beschrieben werden mit

$$\hat{I_k} = q_0 I_k + \sum_{n \neq k} I_n q_{k-n} + \sum_{j=-\infty}^{\infty} c_j \eta_{k-j} \tag{3.30}$$

Der erste Term steht für das gewünschte Symbol, wenn man q_0 zu 1 macht. Der zweite Term repräsentiert die Intersymbol-Interferenz. Soll diese zu Null werden, so müssen alle q_n bis auf q_0 zu Null werden. Darum werden solche Equalizer auch *zero-forcing equalizer* genannt.

$$q_n = \sum_{j=-\infty}^{\infty} c_j f_{n-j} = \begin{cases} 1 & (n = 0) \\ 0 & (n \neq 0) \end{cases} \tag{3.31}$$

Nehmen wir nun die Z-transformierte

$$Q(z) = C(z)F(z) = 1 \tag{3.32}$$

so folgt durch Umformen

$$\boxed{C(z) = \frac{1}{F(z)}} \tag{3.33}$$

Man sieht, daß in diesem Fall der Equalizer einfach das inverse Filter zu dem Filtermodell $F(z)$ ist.

Bei der Realisierung bezieht man das NW-Filter in den Equalizer mit ein. Ein NW-Filter wird also nicht explizit entwickelt. Dadurch kommt man nun zur endgültigen Vorschrift zur Bestimmung der Filterkoeffizienten

$$C^{'}(z) = \frac{1}{F(z)F^{*}(z^{-1})} = \frac{1}{X(z)} \tag{3.34}$$

Abb. 3.7 zeigt das Blockschema eines solchen Equalizers.
Anhand dieser Formel kann man jedoch bereits einen Nachteil bei dieser Methode zur Bestimmung der Filterkoeffizienten erkennen. Für kleine Werte von $X(z)$ wird $C(z)$ sehr groß und somit wird auch das Rauschen extrem gesteigert. Das S/N-Verhältnis geht dort gegen Null. Dadurch kommt es zu Fehlern bei der Symbolrückerkennung.
Solch ein Entzerrer kann somit überall dort eingesetzt werden, wo der Rauschanteil relativ gering ist. Eine Verbesserung hinsichtlich des Rauschens stellt die Ermittlung der Koeffizienten nach dem *mean-square-error*-Kriterium dar.

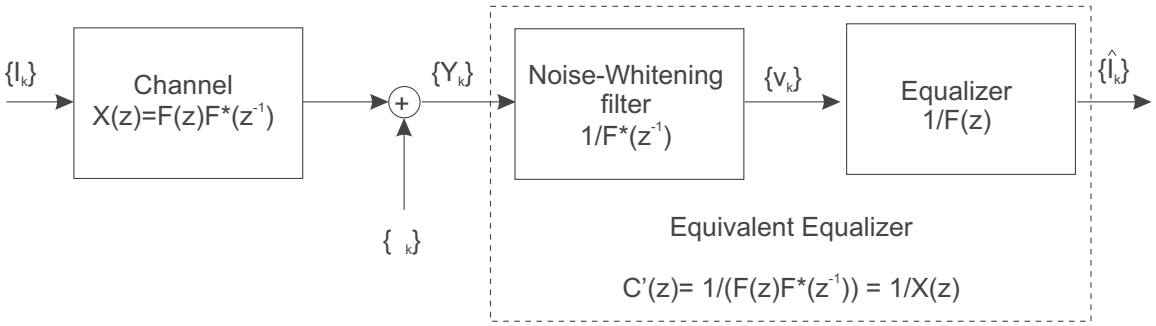


Abbildung 3.7: *Blockschema eines Kanals mit anschließendem zero-forcing equalizer*

3.2.2 MSE-Kriterium

Beim MSE-Kriterium werden die Koeffizienten des Equalizers so bestimmt, daß der Fehler

$$J = E\{\mid I_k - \hat{I}_k \mid^2\} \tag{3.35}$$

minimal wird. Es gilt also, die Differenz des zum k-ten Zeitpunkt übertragenen Symbols und des Symbols am Equalizerausgang, gegeben durch (3.37), möglichst klein zu machen.
Der Ausgang des Equalizers ist gegeben durch

$$\hat{I}_k = \sum_{j=-K}^{K} c_j v_{k-j} \tag{3.36}$$

Setzt man (3.45) in (3.44) ein, so erhält man

$$J(K) = E\{\mid I_k - \hat{I}_k \mid^2\} = E\{\mid I_k - \sum_{j=-K}^{K} c_j v_{k-j} \mid^2\} \tag{3.37}$$

Die Minimierung von **J** führt zu einem linearen Gleichungssystem, mit dem sich nun die Koeffizienten bestimmen lassen. Es hat folgende Form

$$\boxed{\sum_{j=-K}^{K} c_j \Gamma_{lj} = \xi_l \qquad l = -K, ..., -1, 0, 1, ..., K} \tag{3.38}$$

Die Matrix Γ_{lj} ist gegeben durch

$$\Gamma_{lj} = \begin{cases} x_{l-j} + N_0 \delta_{lj} & (\mid l - j \mid \leq L) \\ 0 & (sonst) \end{cases} \tag{3.39}$$

und

$$\xi_l = \begin{cases} f^*_{-l} & (l \leq l \leq 0) \\ 0 & (sonst) \end{cases} \tag{3.40}$$

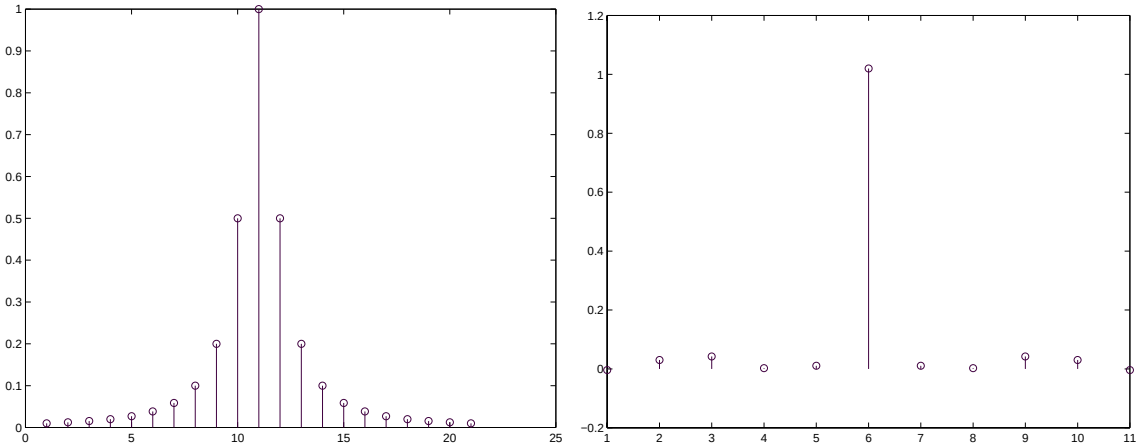
Drückt man die Gleichung in Matrixform aus, so erhält man

$$\mathbf{\Gamma C} = \xi \tag{3.41}$$

wobei nun $\mathbf{C}$ einen $2K + 1$ Zeilenvektor der Filterkoeffizienten, $\mathbf{\Gamma}$ eine $(2K + 1)$ x $(2K + 1)$ Matrix mit Elementen Γ_{lj} und ξ einen $2K + 1$ Zeilenvektor mit Elementen ξ_l darstellen.
Die Lösung für die Filterkoeffizienten lautet demnach

$$\boxed{\mathbf{C_{opt}} = \mathbf{\Gamma^{-1}} \xi} \tag{3.42}$$

Abb. 3.8 zeigt das Signal vor und nach einem Equalizer.



(a) (b)

Abbildung 3.8: *(a) Puls vor Equalizer (b) Equalized Puls*

3.2.3 Performance von MSE-Entzerrern

MSE-Entzerrer besitzen, wie bereits erwähnt, eine bessere Performance als *zero-forcing Entzerrer*. Dennoch sind auch diese Equalizer anfällig gegenüber sehr kleinen Werten im Spektrum der **Kanäle** (Zusammenfassung von Sender, Kanal und Empfänger), was zu einem starken Anstieg von *ISI* und damit zu höheren Bitfehlerraten führt. Dies läßt sich am besten anhand von verschiedenen Kanalmodellen aufzeigen, welche unterschiedliche Spektren besitzen. Abb. 3.9 zeigt Charakteristika drei verschiedener Kanalmodelle. Die dazugehörigen Spektren sind in Abb. 3.10 zu sehen. Während in Abb. 3.10(a) ein relativ konstanter Verlauf über den gesamten Bereich zu sehen ist, zeigen die anderen Kanäle verschieden große Nullstellenbereiche.
Das läßt vermuten, daß die Performance bei Kanal (a) am besten und bei Kanal (c) am schlechtesten ist. Genau dieses Verhalten zeigt Abb. 3.11, in der die Fehlerwahrscheinlichkeit über das S/N-Verhältnis dargestellt ist.

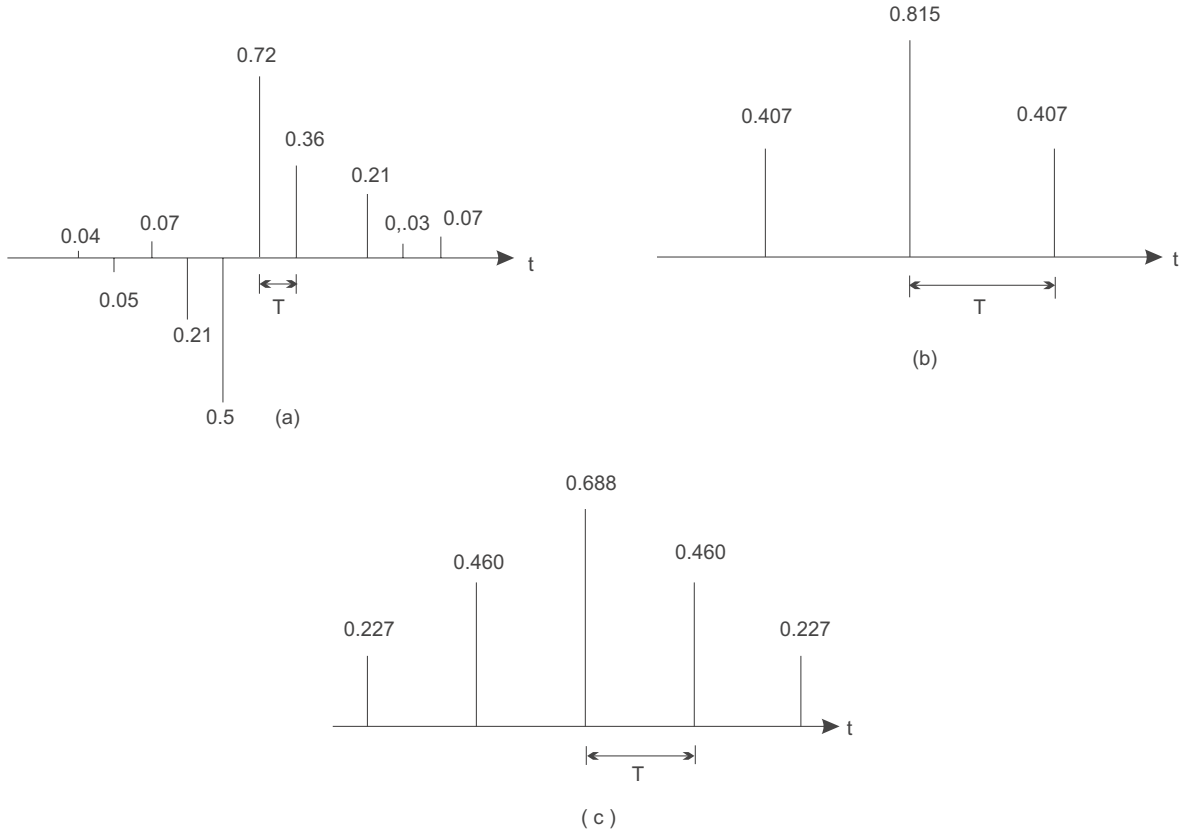


Abbildung 3.9: *Drei verschiedene Kanalmodelle aus Proakis*

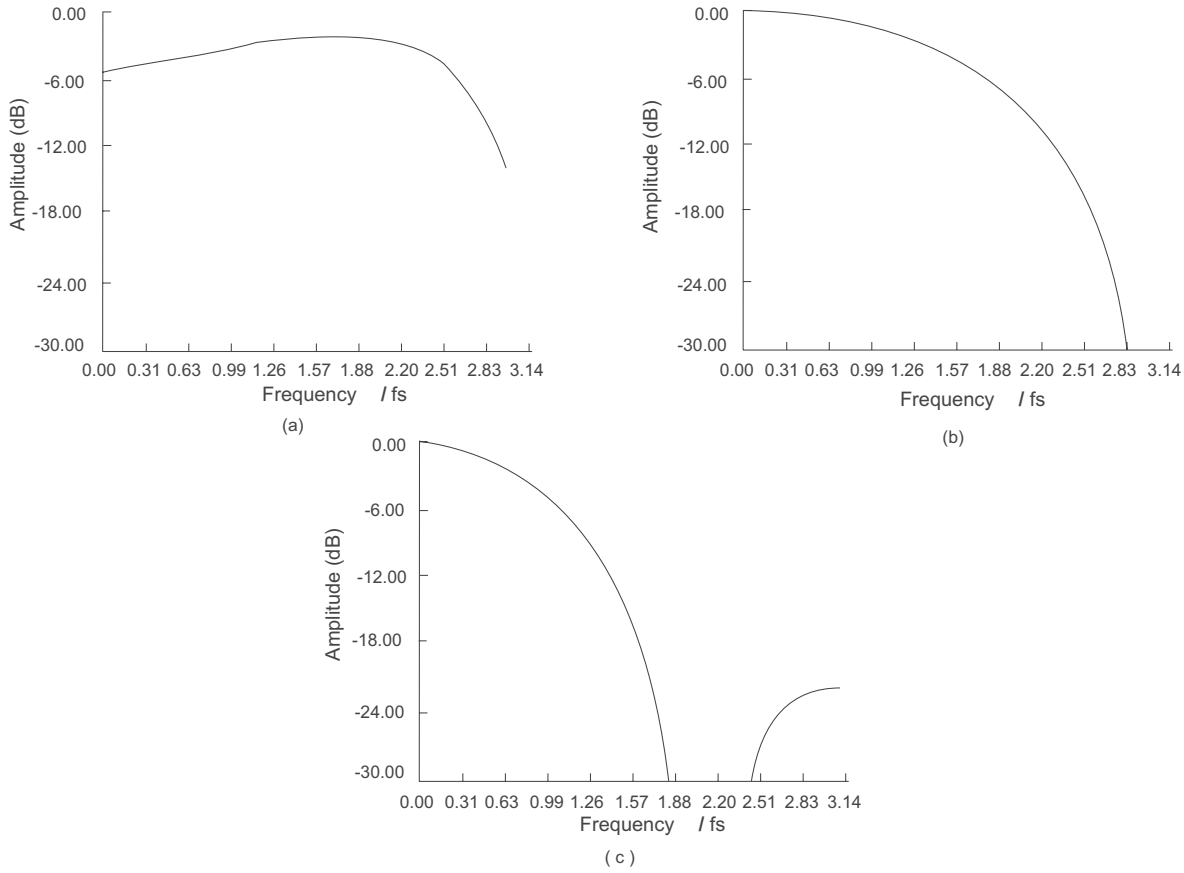


Abbildung 3.10: *Spektren der drei Kanäle*

Man erkennt die extreme Performanceverschlechterung bei den Kanälen mit spektralen Nullstellen. Solch ein MSE-Equalizer kann z. B. gut für Telefonkanäle, die ein ähnliches Spektrum wie Kanal (a) besitzen, eingesetzt werden.
Für andere Kanäle muß eine andere Entzerrerlösung gefunden werden. Eine solche Lösung stellt ein *decision-feedback-Equalizer* dar.

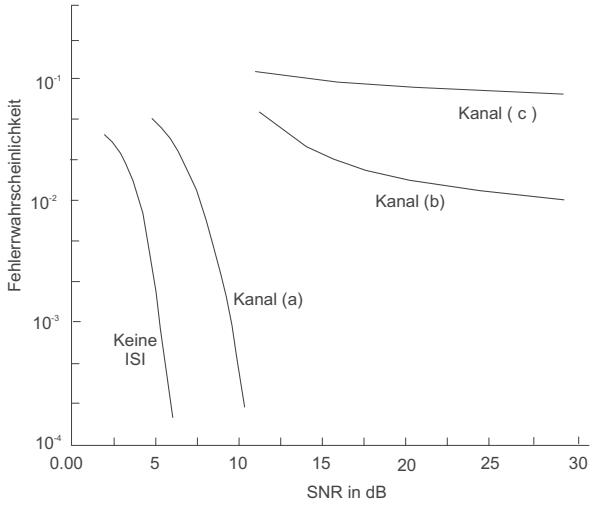


Abbildung 3.11: *Fehlerwahrscheinlichkeit*

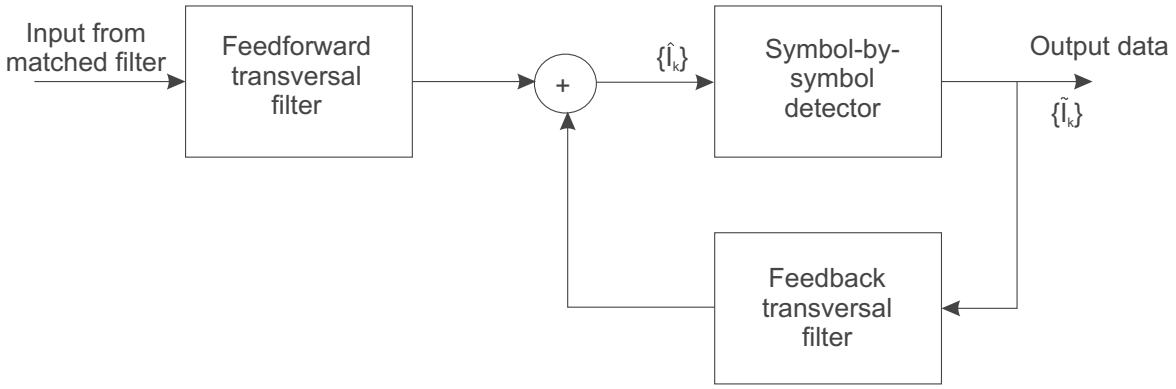


Abbildung 3.12: *Blockschema eines DFE*

3.3 Decision-Feedback Equalizer

Ein *decision-feedback Equalizer* ist ein nicht-linearer Entzerrer. Das Symbol I_{k-1} wird über das FB-Filter zurückgeführt und trägt zur Bestimmumg des Symbols I_k bei. Dies führt zu einer Performanceverbesserung bzgl. der Intersymbol-Interferenz. Abb. 3.12 zeigt das Blockschema eines *decision-feedback Equalizers* oder auch kurz DFE genannt. Ein DFE besteht aus zwei Teilen, einem *feedforward* und einem *feedback* Teil. Beide Bestandteile sind lineare transversale Filter, wie sie vorher schon besprochen wurden.
Das *feedforward* Filter beseitigt die Vorläufer, das *feedback* Filter die Nachläufer wie in Abb. 3.13 zu sehen ist.

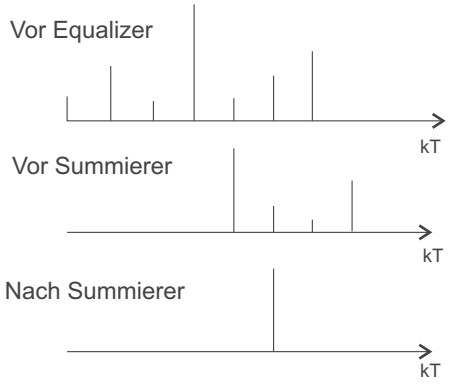


Abbildung 3.13: *Funktion der einzelnen Filter des DFE*

Das Ausgangssignal des DFE läßt sich aus den bisherigen Erkenntnissen beschreiben als

$$\hat{I}_k = \sum_{j=-K_1}^{0} c_j v_{k-j} + \sum_{j=1}^{K_2} c_j \tilde{I}_{k-j} \tag{3.43}$$

wobei c_j die Filterkoeffizienten, $\tilde{I}_k$ die vorher entschiedene Symbole und v_k den Equalizereingang darstellen. Das $feedforward$ Filter hat $K_1 + 1 \geq L$ und das $feedback$ Filter K_2 Koeffizienten. K_2 wird so gewählt, daß alle " Nachläufer ", welche das aktuelle Symbol stören, erfaßt werden. Es gilt $K_2 = L - 1$ ($L =$ Länge der Kanalimpulsantwort).
Zur Bestimmung der Koeffizienten wird das bekannte MSE-Kriterium angewendet. Es gilt also folgendes zu minimieren

$$J(K_1, K_2) = E\{\mid I_k - \hat{I}_k \mid^2\} \tag{3.44}$$

Dies führt wieder zu einem Gleichungssystem zur Bestimmung der *feedforward Koeffizienten*

$$\boxed{\sum_{j=-K_1}^{0} \psi_{lj} c_j = f_{-l}^* \qquad l = -K_1, ..., -1, 0} \tag{3.45}$$

wobei

$$\psi_{lj} = \sum_{m=0}^{-L} f_m^* f_{m+l-j} + N_0 \delta_{lj} \qquad l, j = -K_1, ..., -1, 0 \tag{3.46}$$

Die Koeffizienten des *feedback* Filters lassen sich berechnen durch

$$c_k = -\sum_{j=-K_1}^{0} c_j f_{k-j} \qquad k = 1, 2, ..., K_2 \tag{3.47}$$

3.3.1 Performance von DFE-Equalizern

Abb. 3.14 zeigt die Fehlerwahrscheinlichkeit eines DFE-Equalizers für die beiden Kanäle (b) und (c) aus Abb. 3.9. Vergleicht man dazu die Performance eines linearen Equalizers, dargestellt in Abb. 3.11, so sieht man den starken Performancegewinn bei den DFE-Entzerrern.

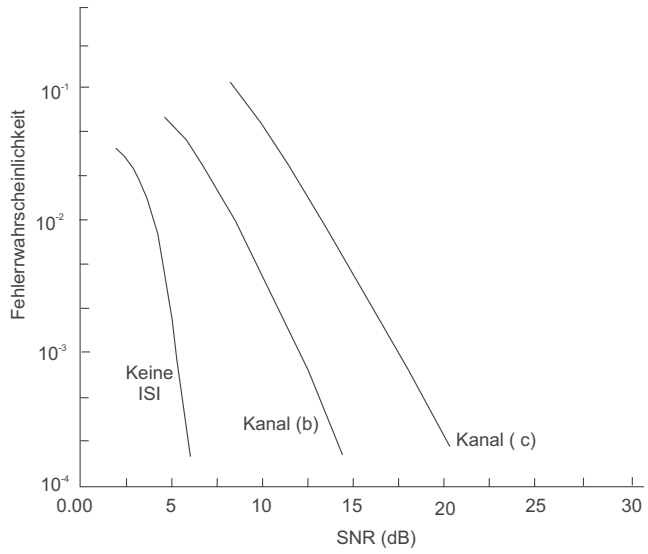


Abbildung 3.14: *Fehlerwahrscheinlichkeit bei DFE-Equalizern*

3.4 Adaptive Entzerrer

Bisher wurde immer davon ausgegeangen, daß die Kanäle **zeitinvariantes** Verhalten besitzen. Dies trifft aber nur für einen Teil der Kanäle zu. Weisen Übertragungskanäle **zeitvariantes** Verhalten auf, (d.h. ihre Eigenschaften ändern sich mit der Zeit), kommen sogenannte $adaptive$

Equalizer zum Einsatz. Solche Equalizer haben die Eigenschaft, sich an die wechselnden Kanaleigenschaften anzupassen.
Es soll mit Hilfe eines adaptiven Algorithmus die Koeffizientengewinnung sowohl für einen linearen- als auch für einen DFE-Equalizer beschrieben werden.

3.4.1 LMS-Algorithmus

Der *Least-Mean-Square*-Algorithmus arbeitet nach dem MSE-Kriterium. Er stellt eine iterative Prozedur zur Gewinnung der Filterkoeffizienten dar. Dabei wird anfangs willkürlich ein Startvektor C_0 vorgegben und anschließend mit der iterativen Prozedur des *steilsten Abstiegs* (*steepest descent*) immer weiter an C_{opt} angenähert. C_{opt} würde man durch Lösung des bekannten Gleichungssystems

$$\Gamma C = \xi \tag{3.48}$$

erhalten.
Diese Prozedur führt schließlich zu folgendem rekursiven Algorithmus

$$\boxed{\hat{C}_{k+1} = \hat{C}_k + \Delta \epsilon_k V_k^*} \tag{3.49}$$

wobei C_k der Koeffizientenvektor zum Zeitpunkt $k \cdot T$ ist. ϵ_k ist der Fehler, definiert durch

$$\epsilon_k = I_k - \hat{I}_k \tag{3.50}$$

und V_k ist ein Vektor von Equalizereingangswerten, welche zum Erhalt von $\hat{I}_k$ beitragen, z. B. $V_k = (v_{k+K} .... v_k .... v_{k-K})$.
Das Δ ist eine beliebige positive Zahl, die aber so klein gewählt werden muß, damit diese iterative Prozedur konvergiert. Von ihrer Größe hängt die Geschwindigkeit, mit der die Prozedur konvergiert, ab.
Wie in (3.50) zu sehen ist, benötigt man für die Bestimmung von ϵ_k eine bekannte Sequenz I_k. Dies kann dadurch erreicht werden, indem vor der eigentlichen Datenübertragung eine sogenannte Trainingssequenz zur ersten Bestimmung der Filterkoeffizienten benutzt wird. Diese liefert nun die benötigte, dem Empfänger bekannte, Sequenz I_k.
Später, bei der eigentlichen Datenübertragung, wird die Koeffizientenanpassung weitergeführt und der Fehler ϵ_k wird berechnet aus $\tilde{\epsilon}_k = \tilde{I}_k - \hat{I}_k$, wobei $\tilde{I}_k$ das aus $\hat{I}_k$ entschiedene Symbol darstellt. Dies wird als *decision-directed mode of adaptation* bezeichnet.
Abb. 3.15 verdeutlicht den Vorgang zur adaptiven Koeffizientengewinnung.

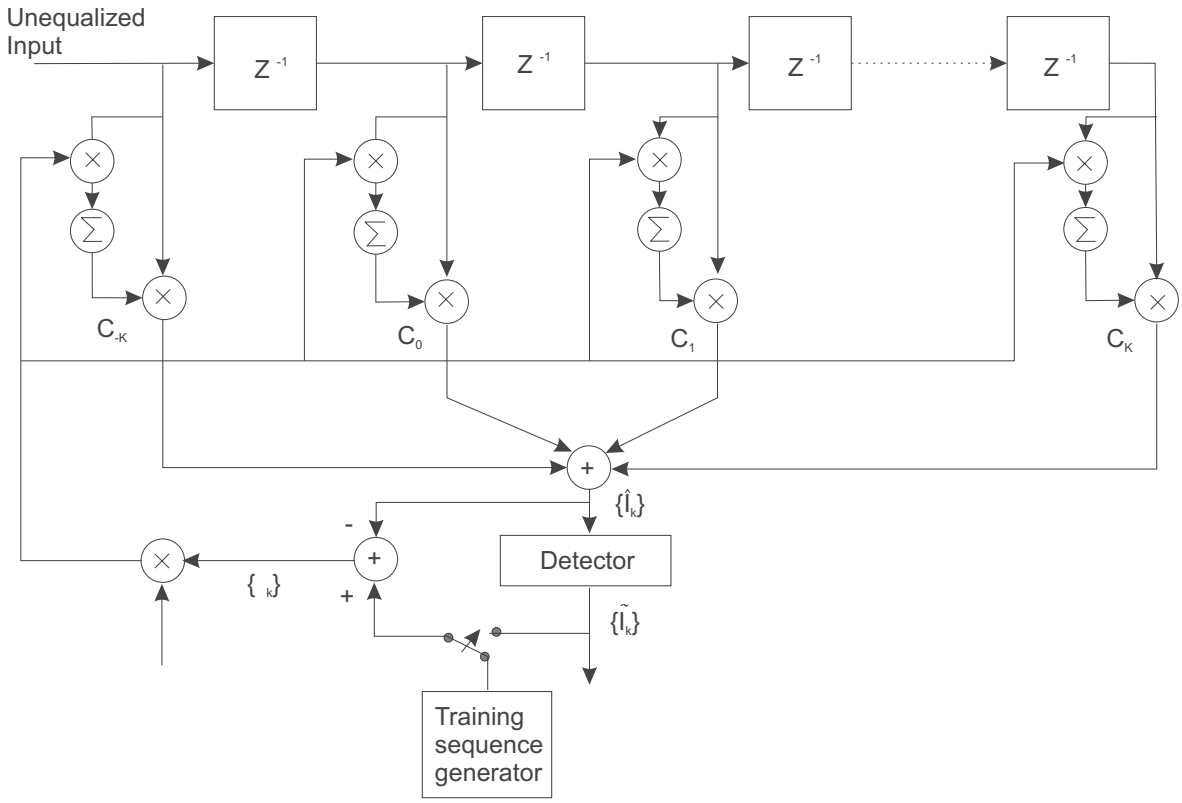


Abbildung 3.15: *Adaptiver MSE-Equalizer*

3.5 Adaptiver DFE-Equalizer

Die Koeffizientengewinnung bei einem DFE-Equalizer läuft im Prinzip genauso, wie bei einem MSE-Equalizer. Es wird der gleiche Algorithmus angewendet.

$$\tilde{C}_{k+1} = \tilde{C}_k + \Delta\tilde{\epsilon}_k V_k^* \tag{3.51}$$

wobei $\tilde{\epsilon}_k = \tilde{I}_k - \hat{I}_k$ und $V_k = (v_{k+K_1}...v_k, \tilde{I}_{k-1}...\tilde{I}_{k-K_2})$.
Auch hier werden mit Hilfe einer Trainingssequenz die ersten Koeffizientenwerte des Equalizers bestimmt. Bild 3.16 zeigt einen adaptiven DFE-Equalizer.

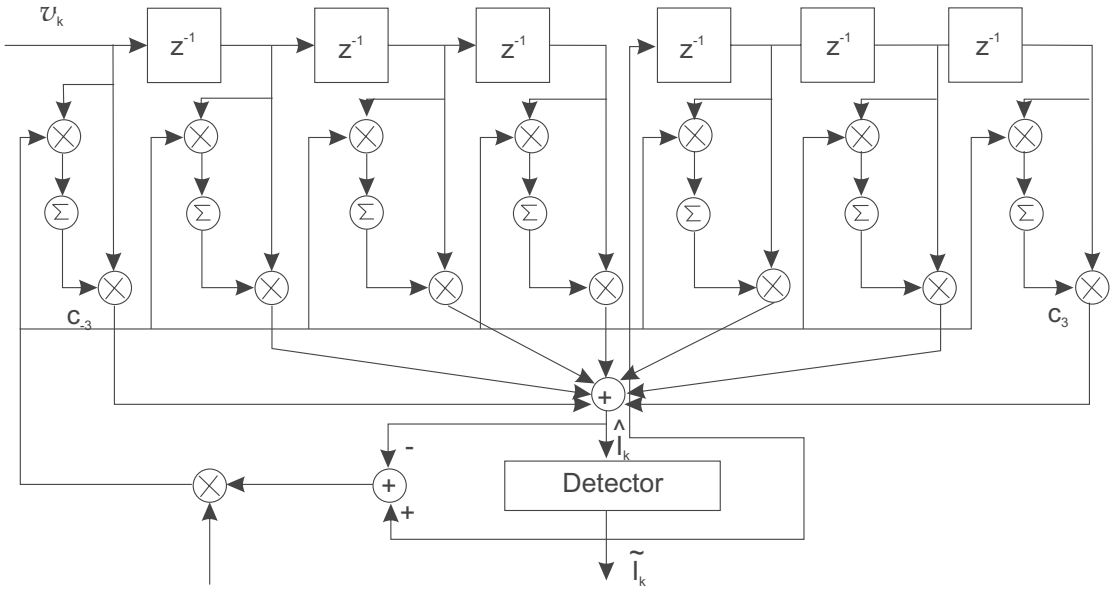


Abbildung 3.16: *Adaptiver DFE-Equalizer*

4 Zusammenfassung

Für die Auswahl des richtigen Equalizerstyps sind die Eigenschaften des Übertragungskanals entscheidend. Weist das Kanalmodell in seinem Amplitudenspektrum keine Nullstellenbereiche auf und ist es zeitinvariant, dann kann dort ein **linearer preset-Entzerrer** eingesetzt werden. wie z. B. ein MSE-Equalizer. Ändern sich hingegen die Kanaleigenschaften mit der Zeit, so ist die **adaptive** Methode zur Koeffizientenanpassung notwendig.
Die folgende Tabelle zeigt eine mögliche Entzerrerauswahl in Abhängigkeit verschiedener Kanaleigenschaften.

Kanaleigenschaften	Equalizertyp
Keine Nullstellenbereiche im Amplitudenspektrum, zeitinvariant	Linearer Equalizer preset-Methode
Keine Nullstellenbereiche im Amplitudenspektrum, zeitvariant	Linearer Equalizer adaptive Methode
Nullstellenbereiche im Amplitudenspektrum, zeitinvariant	DFE-Equalizer preset-Methode
Nullstellenbereiche im Amplitudenspektrum, zeitvariant	DFE-Equalizer adaptive-Methode

Abbildung 4.1: *Entzerrerauswahl*

5 Umsetzung

5.1 In Matlab

5.1.1 MSE-Equalizer, mse.m

Das Programm $mse.m$ liest die Kanalnummer bzw. einen benutzerdefinierten Kanal und das S/N-Verhältnis ein. Es werden die Koeffizienten des MSE-Equalizers und eine Grafik mit Kanal, Amplitudenspektrum und dem Signal nach dem Equalizer ausgegeben.
Abb. 5.1 und 5.2 zeigen die vier verschiedenen Kanäle und deren Spektren. Abb. 5.3 zeigt die Ausgabe für *mse(1,0,20)*.
Im Folgenden die wichtigsten Programmsequenzen.

- Umsetzung der Formel (3.51) zur Berechnung der Koeffizienten

```
%Berechnung von Copt
% Aus diesen Werten wird die Matrix T(lj) erstellt
x=xcorr(f);
x=x/x((length(x)+1)/2);% x(0)=1;

 d=length(f);%Erstellung der Matrix T(lj) nach (3.48)
 r=d-1;
 m=zeros(length(f),length(f));
 for i=1:d
    r=r+1;
    a=r;
    for l=1:d
       m(i,l)=x(a);
       if(i==l)
          %Auf Hauptdiagonalen wird N0 addiert
          m(i,l)=m(i,l)+10^(-SNR/10);
       end
       a=a-1;
    end
 end
 f1=fliplr(conj(f));% Vektor f*(L)-f*(0)
 copt=inv(m)*f1.'; %Berechnete Koeffizienten nach (3.51)
```

- Die Erzeugung der gestörten Datensequenz nach (3.36)

```
%Erzeugung von gestörten Daten
 data1=conv(info,f);
 var=0.5*10^(-SNR/10);
 AWG1=randn(1,size(data1,2))*sqrt(var);
 AWG2=AWG1;
 AWG=AWG1+j*AWG2;
 data=data1+AWG;
```

- Berechnung Equalizerausgang nach (3.37)

```
%Berechnung Equalizerausgang
for k=1:length(info) %Bestimmumg von I(k)=summe [c(j)*v(k-j)] j=-
K...K (Proakis Seite 602)
     v1=fliplr(data(1,k:(k+length(copt)-1))); %Sequenz v(k-j)
     I1(k)=copt.'*v1.';
end
```

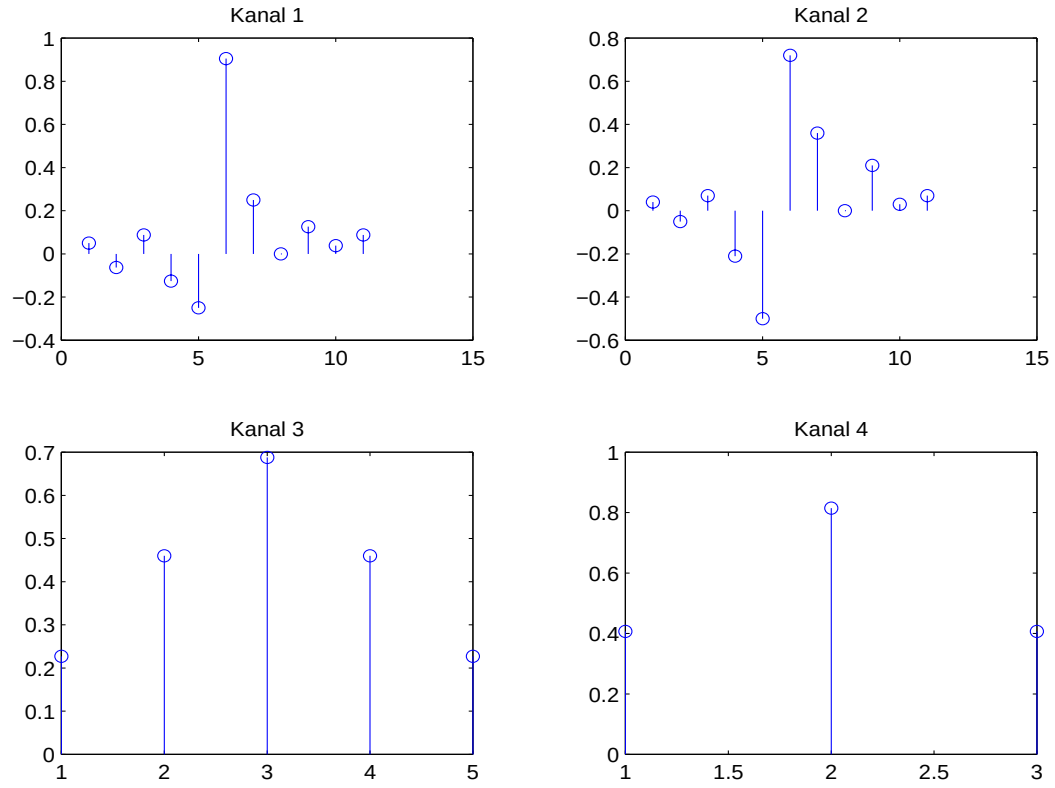


Abbildung 5.1: *Kanäle*

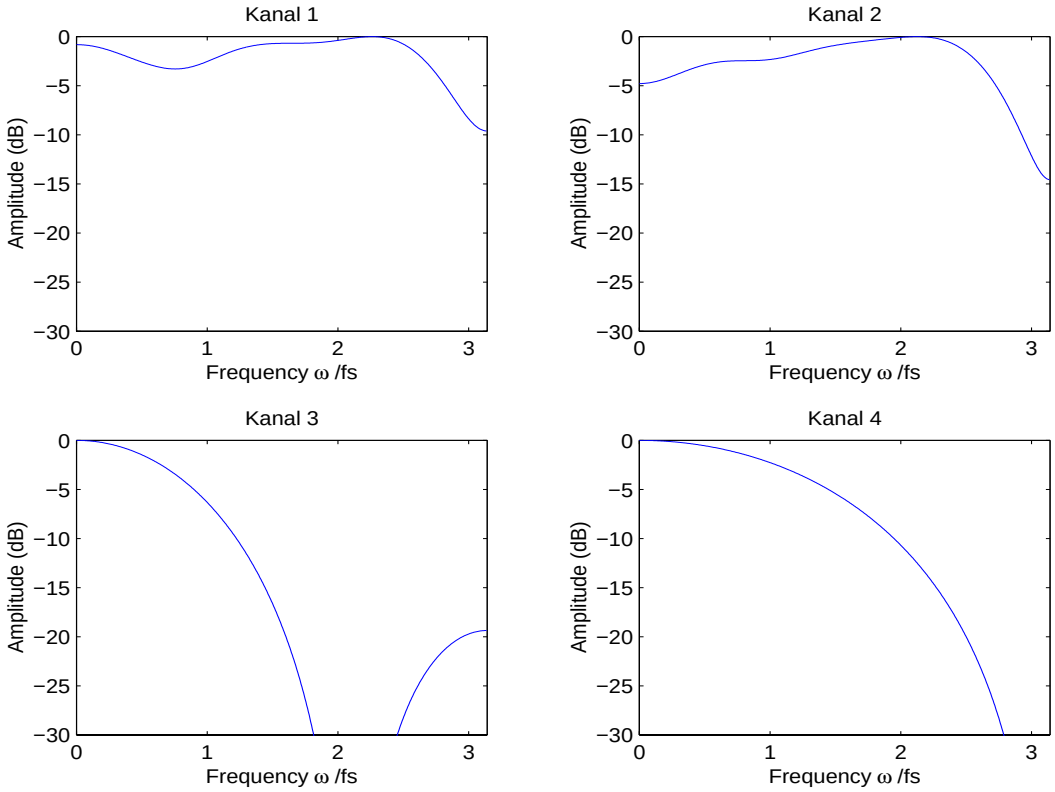


Abbildung 5.2: *Amplitudenspektren der Kanäle*

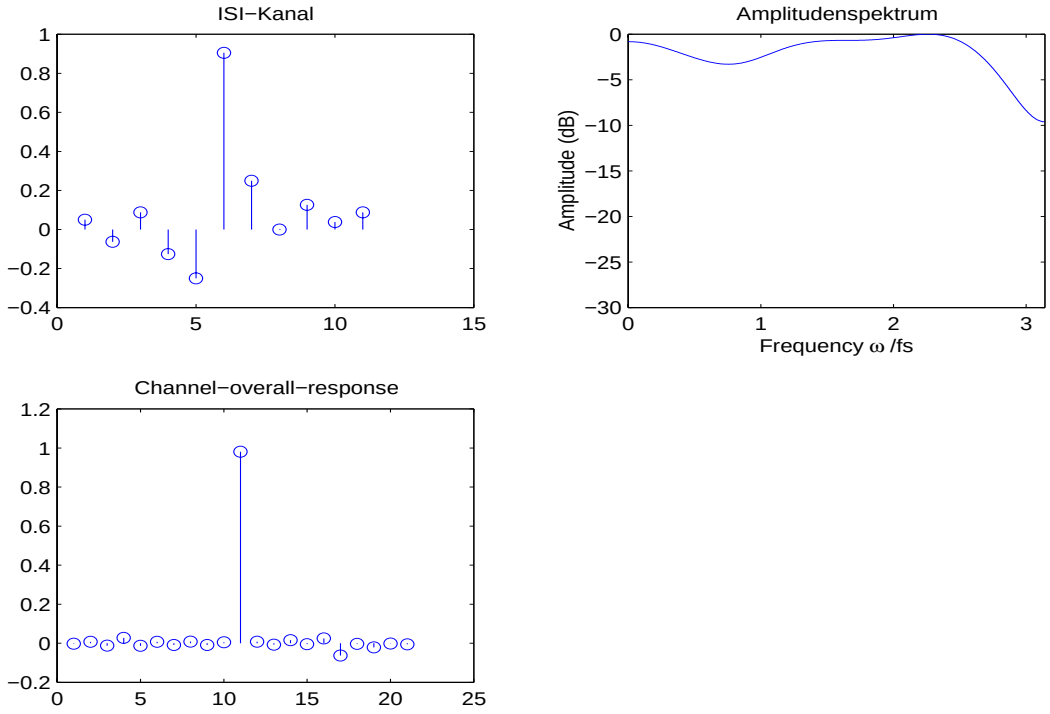


Abbildung 5.3: *Ausgabe für mse(1,0,20)*

5.1.2 DFE-Equalizer, dfe.m

Dieses Programm arbeitet nach demselben Prinzip wie beim MSE-Equalizer. Ausgabe ist das Signal vor dem Equalizer, nach dem feedforward-Filter und nach der Additionsstelle. Die Programmsequenz zur Berechnung der Koeffizienten nach (3.55) ist:

```
%Bestimmung der psi-matrix
psi = zeros(K1+1,K1+1);
 for k=-K1:0
  for jo=-K1:0
   for i=0:-k
     if ( ((i+1+k-jo)>0) & ((i+1+k-jo)<L+1) & (i<L))
       psi(k+K1+1,jo+K1+1)=psi(k+K1+1,jo+K1+1)+(conj(f(i+1))...
     ...* f(i+1+k-jo));
      end
    end
  end
 end

    for k=-K1:0
     psi(k+K1+1,k+K1+1) = psi(k+K1+1,k+K1+1) + 10^(-SNRdfe/10);
    end

    %Koeffizienten des FF-Filters
    cDFE = (psi \ fliplr(f(1:K1+1))').';
```

```
%Koeffizienten des FB-Filters
 bDFE=zeros(1,K2);
  for k = 1:K2
   for jo=-K1:0
    if( (k-jo+1) <= L )
       bDFE(k) = bDFE(k) - cDFE(jo+K1+1) * f(k-jo+1);
    end
   end
  end
```

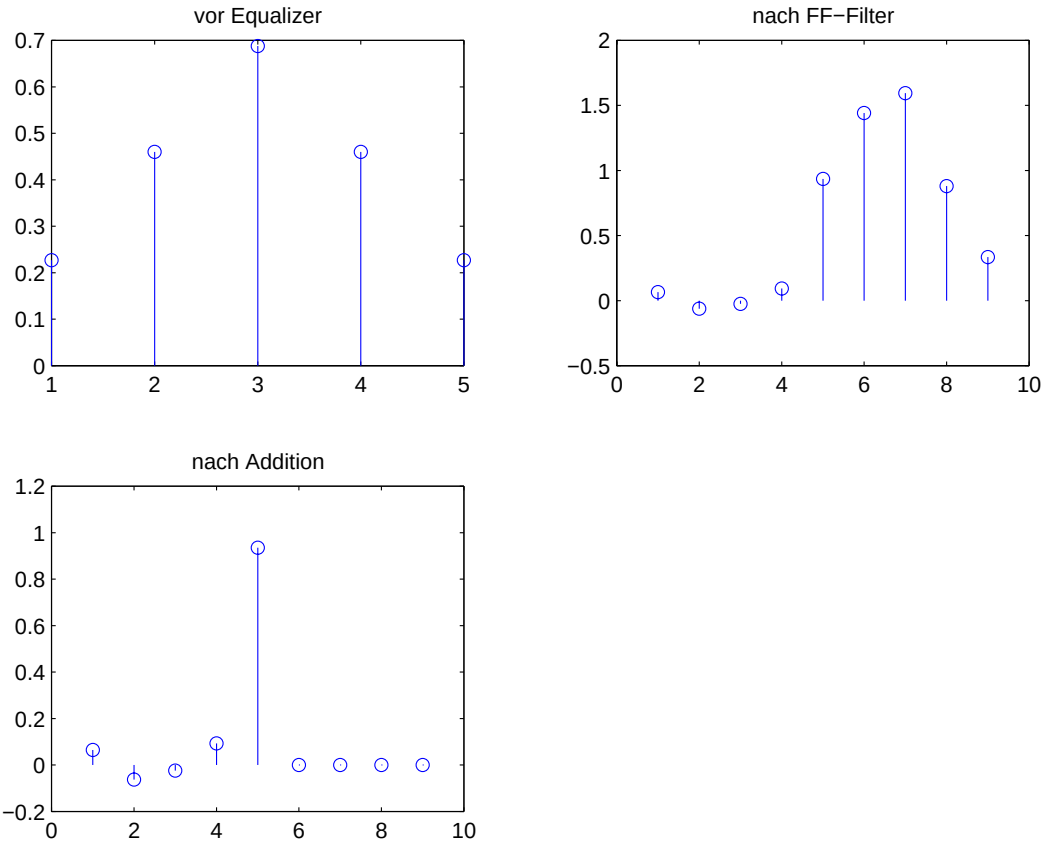


Abbildung 5.4: *Ausgabe für dfe(3,0,20)*

5.1.3 Ersatzfiltermodell, kanalmodell.m

Will man ein eigenes Ersatzfilter bestimmen, so kann dies mit Hilfe des Programms *kanalmodell.m* geschehen. Es wird dabei mit einer achtfachen Überabtastung gearbeitet. Aus Sender, einem Kanal und dem Matchedfilter mit anschließendem Abtaster wird das Ersatzfiltermodell gewonnen. Als Sendefilter wird ein *root-raised-cosine* gewählt. Der Kanal wird von der Funktion *Fetsichan* geliefert und soll zwei Symbole stören (bei achfacher Überabtastung entspricht dies ca. 16 Kanalkoeffizienten). Das Matchedfilter schließlich ist auf $Sender + Kanal$ gematched. Das Programm liefert durch Aufruf *a=kanalmodell* die Koeffizienten des Ersatzfiltermodells. Dieses kann nun z. B. in die Funktion $mse.m$ mit $mse(0, a, 20)$ eingelesen werden.

5.1.4 Bitfehlerrate, msetest.m

Mit diesem Programm kann man die Bitfehlerrate eines MSE-Equalizers ermitteln. Als Parameter werden bis zu zwei verschiedene Ersatzfiltermodelle eingegeben und können somit verglichen werden. Man hat die Auswahl zwischen den vorgegebenen Kanälen nach Abb. 5.1 oder eigenen Kanälen, z. B. erzeugt mit $kanalmodell.m$. Desweiteren wird noch die Länge der Testsequenz eingegeben. Diese sollte im Praktikum min. 10000 und max. 30000 betragen, da sonst die Rechenzeit zu lang wird. Allerdings ergeben sich für höhere Werte auch genauere Ergebnisse. Der genaue Aufruf der Funktion lautet *msetest(k1,kan1,kan2,N)*. Im Matlab Kommandofenster werden jeweils die aktuellen S/N-Werte mit dazugehörigen Bitfehlerraten angezeigt. Die erzeugte Grafik zeigt das Ersatzfiltermodell (ISI-Kanal), dessen Amplitudenspektrum, ein entzerrtes Bit und die Bitfehlerwahrscheinlichkeit in Abhängigkeit vom S/N-Verhältnis. Abb. 5.5 zeigt ein Beispiel.

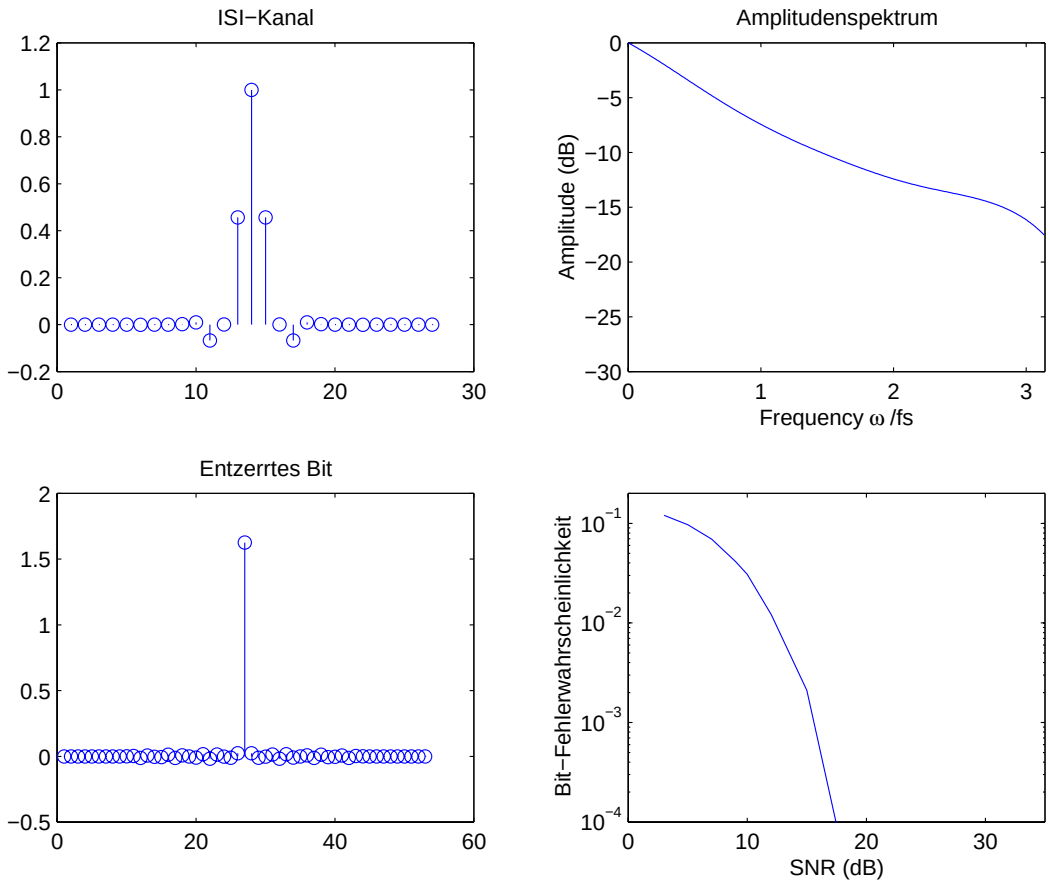


Abbildung 5.5: *Ausgabe msetest*

5.1.5 Bitfehlerrate, dfetest.m

Diese Funktion arbeitet nach gleichem Prinzip wie $msetest.m$ aber nun für einen DFE-Equalizer. Es gelten dieselben Parameter. Ausgabe hier ist das Signal vor dem Equalizer, nach dem Feedforward-Filter und nach der Additionsstelle und die Bitfehlerwahrscheinlichkeit. Abb. 5.6 zeigt ein Beispiel.

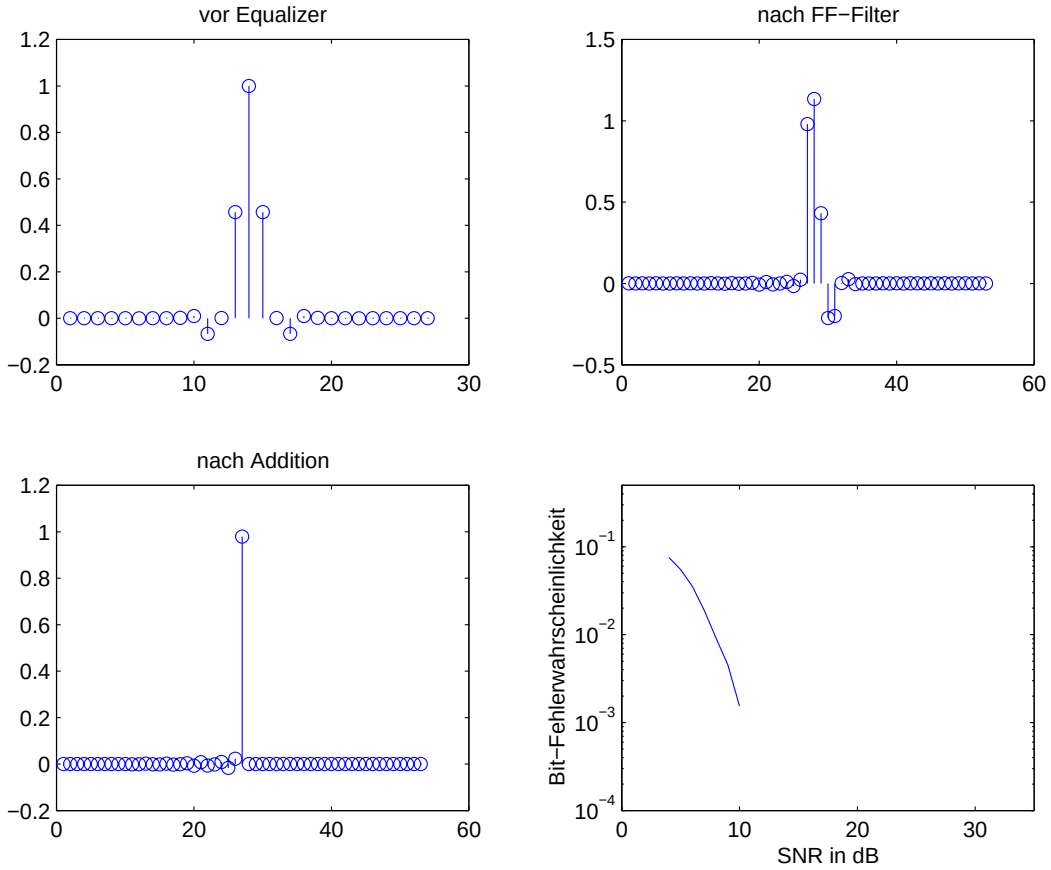


Abbildung 5.6: *Ausgabe dfetest*

5.1.6 Vergleich MSE zu DFE, msedfe.m

Die Funktion *msedfe(k,kan,N)* vergleicht die Performance der zwei Equalizertypen. Es kann wieder zwischen vorgegebenen und selbst bestimmten Ersatzfiltermodellen gewählt werden. Es wird jeweils eine Grafik für den MSE- bzw. DFE-Equalizer ausgegeben. Die Bitfehlerraten erscheinen in einem Diagramm.
Beispiel:

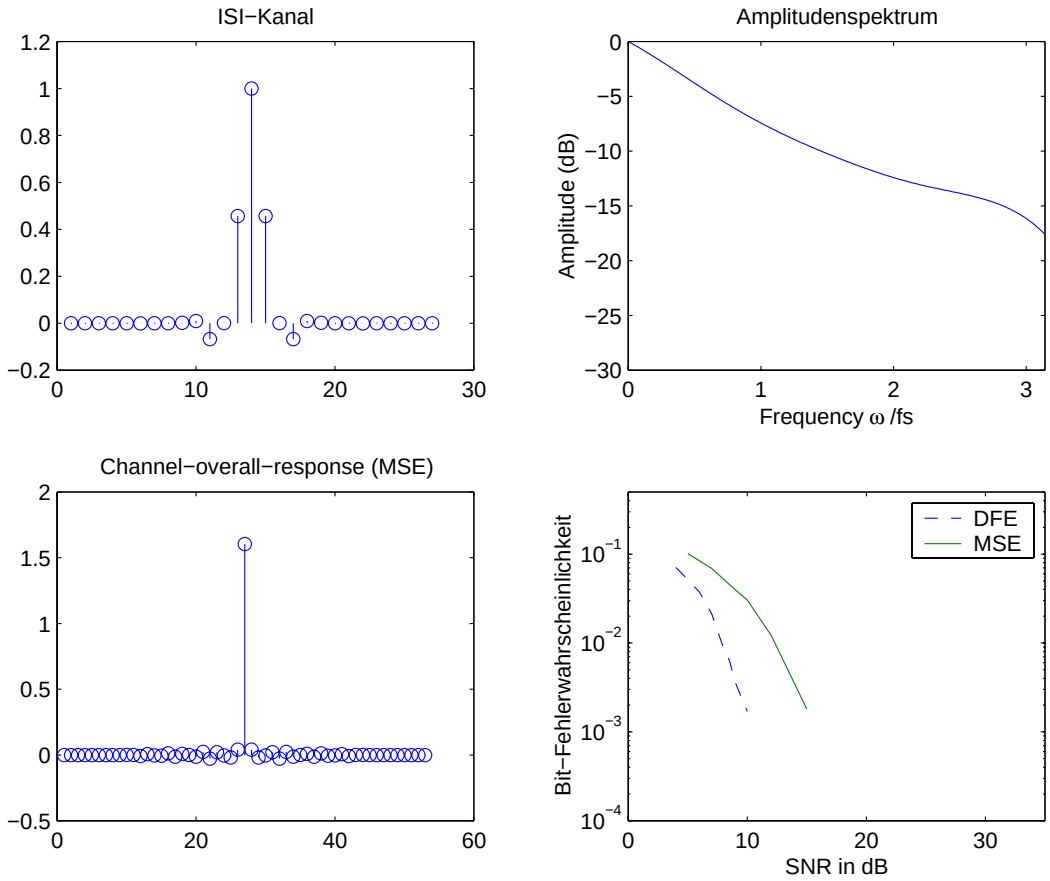


Abbildung 5.7: *Ausgabe msedfe*

5.2 In C

Als Grundlage zur Implementierung der Equalizer dient das Programm $nnrz.c$ aus [2]. Dort wurde aus einer PCM-codierten Zahl ein überabgetastetes NRZ-Signal (never-return to-zero) erzeugt und auf ein Sendefilter gegeben.
Auf das Ausgangssignal wurde noch Rauschen additiv hinzugegeben und passierte anschließend ein Matchedfilter. Dieses Programm mußte nun noch durch weitere Filter (Kanal, Equalizer) erweitert werden.
Ein Problem ergab sich noch dadurch, daß die vorhandene Filterfunktion $fir()$ nicht für komplexe Zahlen ausgelegt ist. Deswegen mußte eine eigene Filterfunktion erstellt werden, die diese Eigenschaft besitzt.
Die komplexe Filterfunktion $fir1()$ sieht folgendermaßen aus:

```
complex_float fir1(complex_float input,float pm coeffs_re[],
float pm coeffs_im[],
float dm state_re[],float dm state_im[],int taps)
{
temp.re=0.0;
temp.im=0.0;

state_re[0]=input.re;
state_im[0]=input.im;

   for(i=0;i<taps;i++)
   {
     temp.re+=(coeffs_re[i]*state_re[i])-(coeffs_im[i]*state_im[i]);
     temp.im+=coeffs_im[i]*state_re[i]+coeffs_re[i]*state_im[i];
   }

   for(j=taps-1;j>=0;j--)
   {
     state_re[j+1]=state_re[j];
     state_im[j+1]=state_im[j];
   }
return temp;
}
```

Nachteil dieser Funktion ist, daß sie um einiges langsamer ist als die orginale Filterfunktion $fir()$. Um in etwa die gleiche Ausführungsgeschwindigkeit zu erhalten, müßte $fir1()$ wie $fir()$ direkt in Assembler programmiert werden. Dies wurde aber im Rahmen dieser Diplomarbeit nicht durchgeführt.
Um aber an Geschwindigkeit zu gewinnen, wurde in den neuen Programmen die Sequenz der PCM-Codierung weggelassen und durch eine einfachere Erzeugung von zu übertragenen Daten ersetzt. Desweiteren wird mit einer achtfachen anstatt 20-fachen Überabtastung gearbeitet.
Die Ermittlung der Koeffizienten von *Sendefilter, Sendefilter u. Kanal, Machtedfilter* und *Ersatzfiltermodell* erfolgt mit Matlab durch Aufruf der Funktion *[sende,sendekan,mf,kanal]=kanalmodell*.
Die Koeffizienten der Equalizer erhält man durch Aufruf der Funktion *[c]=mse(0,kanal,20)* oder *[ff,fb]=dfe(0,kanal,20)*. Die Funktionen sind unter 5.1 beschrieben.

Um diese Koeffizienten in C zu verarbeiten, müssen sie getrennt nach Real- bzw. Imaginärteil abgespeichert werden. Solche Koeffizientedatein sind z. B. *sendere.h, mfim.h* usw.
Diese Koeffizienten werden am Anfang des Programms mit

```
//Sendefilterkoeffizienten real
float pm filterRE[TAPS]=
{
   #include "sendere.h"
}
```

in Arrays eingelesen.
Die Timerfunktion steuert die Erzeugung der zu übertragenen Daten. Bei jedem Durchlauf wird zuerst eine -1 bzw. 1 (repräsentiert durch einen äquivalenten Spannungspegel an den *DA-Ausgängen* von -1 bzw. 1V) erzeugt. Je nach eingesetztem Filter, gesteuert durch den Hardwareinterrupt (Taster), wird die Überabtastung durch Anhängen von Nullen realisiert.

```
//Erzeugung einer Zufallszahl und abhängig davon eine 1 bzw. -1
void timer_hi_prior(int sig_num)
{
   int aus=0;
   sig_num=sig_num;
   // os = OverSampling
   if(os==0)
   {
   // Takt = high
   SetIOP(dac_7,dac_high);

   // Die Zufallszahl wird erzeugt.
     if(rand()<1073741824)
     {
       level=-PEGEL;
     }
     else
     {
       level=PEGEL;
     }

   }

    // ZA-1 mal werden Nullen ereugt (Überabtastung)
   else
   {
     if(r_count==1||r_count==2||r_count==3) //Welche Filter?
     {
       level=dac_low;
       // Takt = low
       SetIOP(dac_7,dac_low);
```

```
      }
    }
```

Der Hardwareinterrupt inkrementiert eine Zählervariable, nach der die Filterauswahl erfolgt, und setzt verschiedene S/N-Werte.

```
// Interruptfunktion für IRQ1
// Mit dem Interrupt wird der Rauschabstand ausgewählt.
void press_button(int sig_num)
{
sig_num=sig_num;
output2.re=0.0;
output2.im=0.0;

r_count++;
  if(r_count==2)
  {
    snrdb=23;
  }
  if(r_count==3)
  {
    snrdb=15;
  }
  if(r_count==4)
  {
    snrdb=100;
  }
  if(r_count==5)
  {
  r_count=0;
  }

}
```

Je nach Wert der Zählervariablen erfolgt die Berechnung der verschiedenen Filterausgänge und Ausgabe der Werte über die *D/A-Wandler*(Auszug aus dfe.c).

```
// Werte werden auf das Filter gegeben.
input.re=(float)level;

if(r_count==1||r_count==2||r_count==3)
{
 //Sendefilter
 output=fir1(input,filterRE,filterIM,states_re,states_im,TAPS);
 //Sendefilter+Kanal
 output1=fir1(input,filter1RE,filter1IM,states1_re,states1_im,TAPS1);
 //Rauschen
```

```
 input1.re=(float)(int)output1.re+awgn(en,PEGEL,snrdb);
 input1.im=(float)(int)output1.im+awgn(en,PEGEL,snrdb);
 aus=(int)input1.re;
 //Ausgabe
 SetIOP(dac_2,(int)output.re+dac_zero);
 SetIOP(dac_3,aus+dac_zero);
}
else .....
```

5.2.1 Oszilloskopausgabe MSE

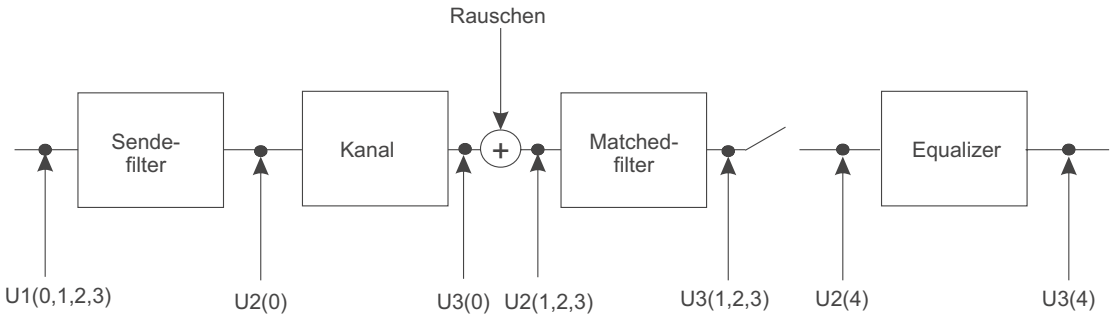
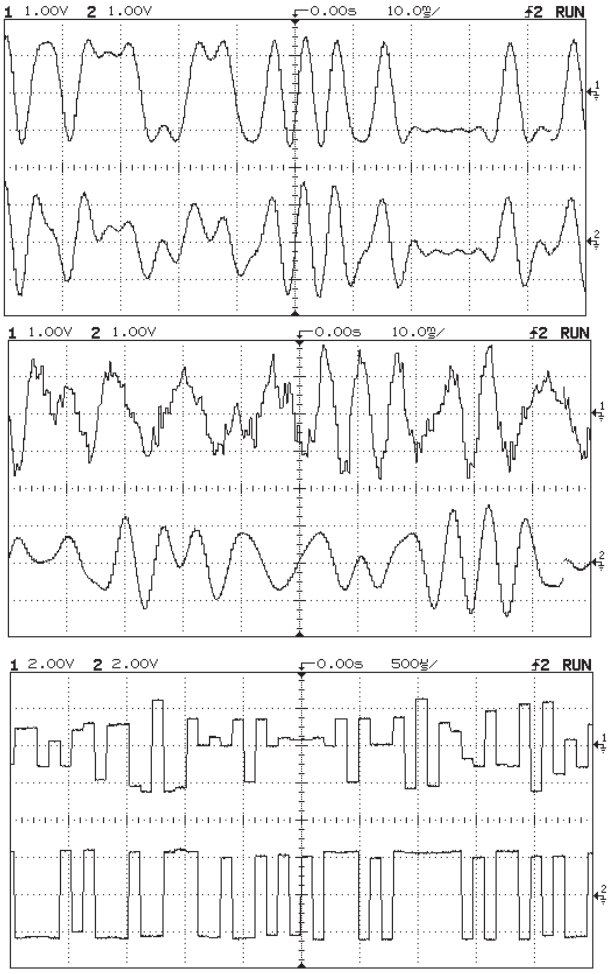


Abbildung 5.8: *Meßpunkte MSE*



Die Signale von oben nach unten:
1. Signal nach Sendefilter
2. Signal nach Sendefilter+Kanal
3. Signal 2+Rauschen
4. Signal nach Matchedfilter
5. Signal nach Abtaster
6. Signal nach Equalizer

Abbildung 5.9: *Ausgabe mse.c*

5.2.2 Oszilloskopausgabe DFE

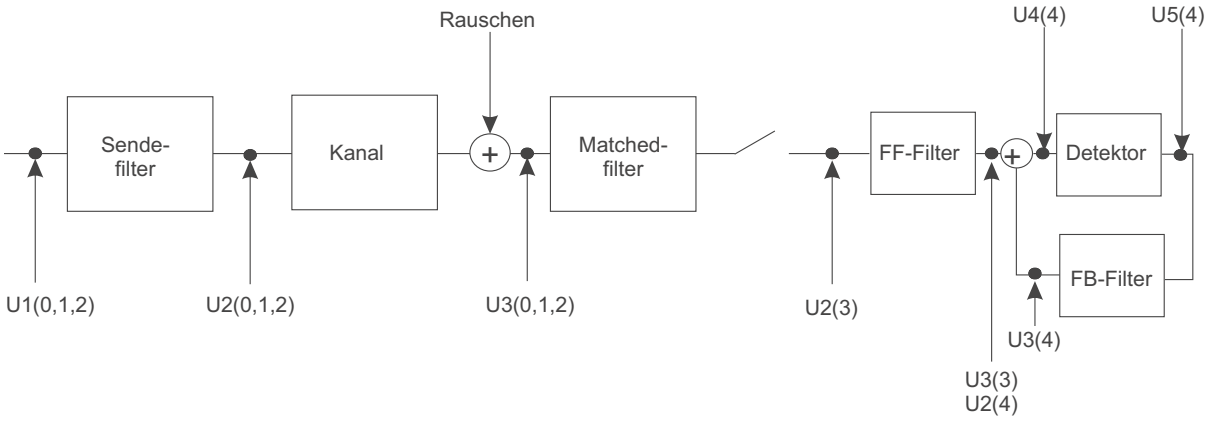
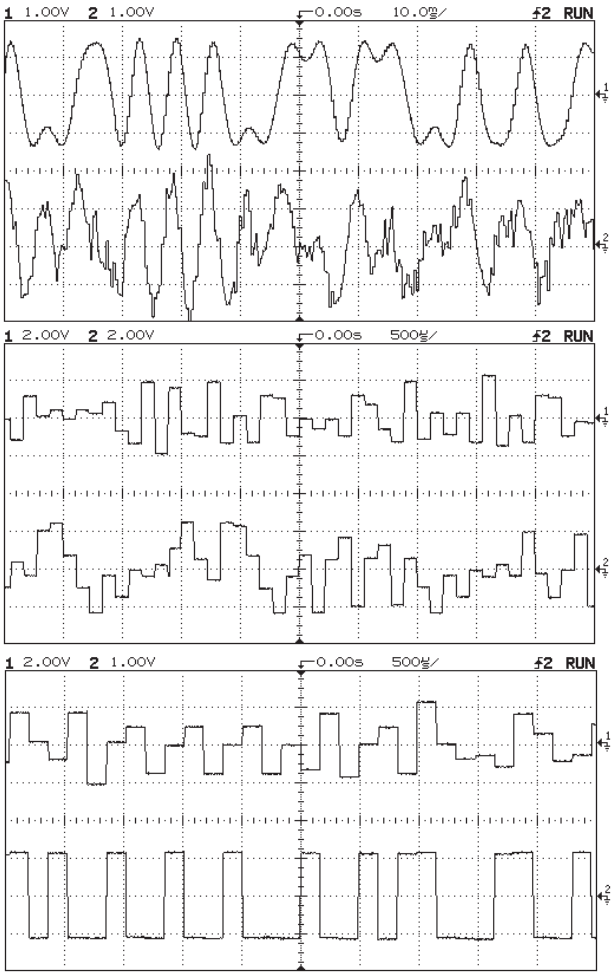


Abbildung 5.10: *Meßpunkte DFE*



Signale von oben nach unten:
1. Signal nach Sendefilter
2. Signal nach Sendefilter+Kanal+Rauschen
3. Signal nach Abtaster
4. Signal nach FF-Filter
5. Signal nach FB-Filter
6. Signal nach Additionsstelle

Abbildung 5.11: *Ausgabe dfe.c*

6 Hard- und Software

6.1 Praktikumsaufbau

Als Plattform zur Demonstration der geschriebenen Programme dient ein Praktikumsaufbau einer vorherigen Diplomarbeit. In dem Praktikumsaufbau ist das Evaluationboard $EZ-KitLite$ von $AnalogDevices$ mit dem Digitalen Signalprozessor $DSP21061$ integriert. Der DSP 21061 kann 32 und 40 Bit Gleitkommazahlen und 32 Bit Festkommazahlen verarbeiten. Er wird mit einer Frequenz von 40 MHz getaktet. Der Prozessor hat zusätzlich noch einen I/O-Bus, an den externe Einheiten angeschlossen werden können. Diese externen Einheiten bilden in dem Praktikumsaufbau sechs 8-Bit Digital-Analog-Wandler und zwei 12-Bit Analog-Digitalwandler.

Über die DA-Wandler können verschiedene Zwischensignale zur Anschauung bzw. Weiterverarbeitung ausgegeben werden. Die AD-Wandler bieten die Möglichkeit, mehrere Praktikumsaufbauten miteinander zu verbinden, um evtl. eine komplette Digital-Übertragungsstrecke darzustellen. Abbildung 6.1 zeigt die Architektur des $DSP21061$ und Bild 6.2 das EZ-Kit lite.

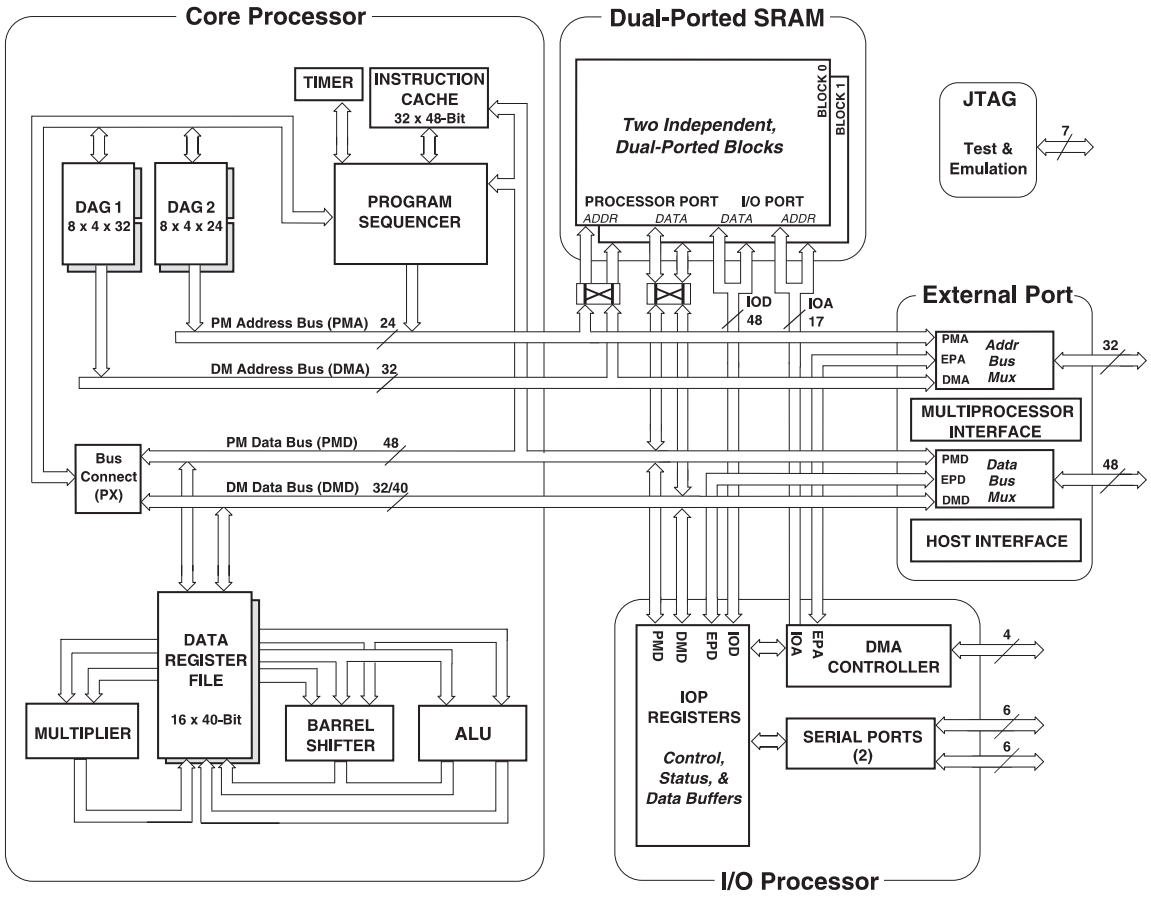


Abbildung 6.1: *Schematischer Aufbau des ADSP 21061.*

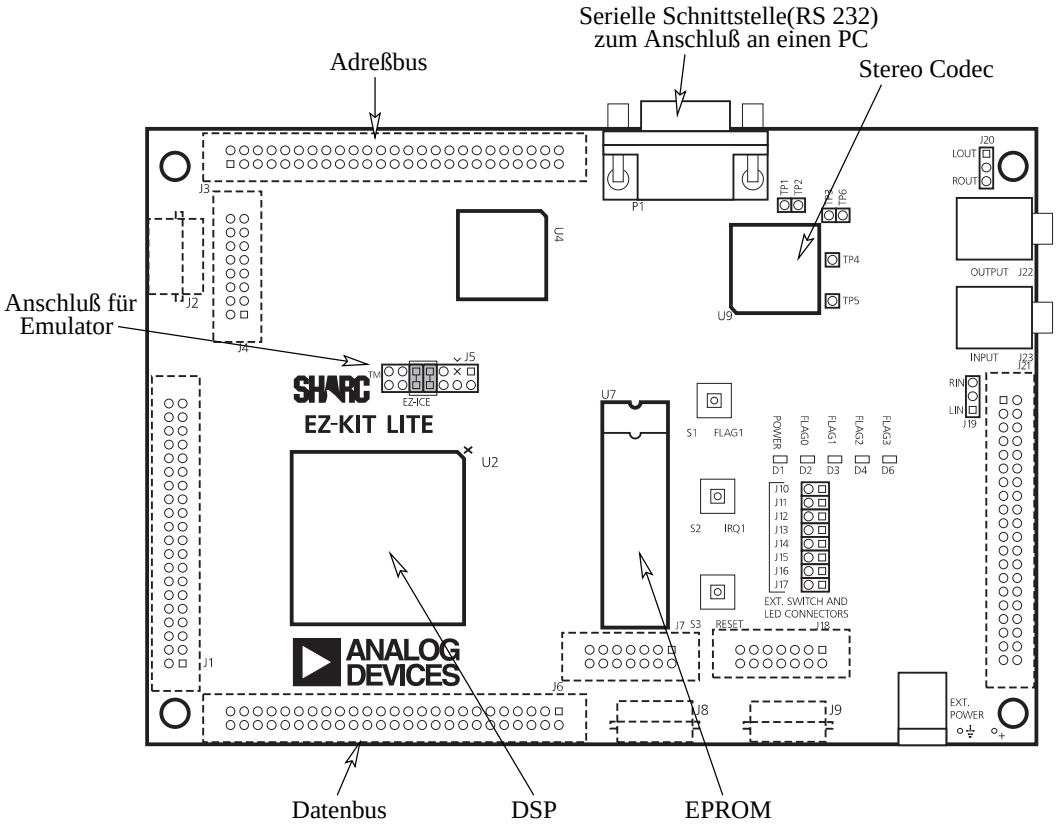


Abbildung 6.2: *Layout EZ-KIT Lite.*

Für weitergehende Informationen sei an dieser Stelle auf [1] [2] verwiesen.

6.2 AD-Wandler

Wie schon erwähnt, kommen in dem Praktikumsaufbau zwei 12-bit A/D-Wandler zum Einsatz. Diese konnten aber bisher nicht in Betrieb genommen werden, da der Wandler Störungen auf dem Datenbus verursachte.

6.2.1 Fehlersuche

Bei der Überprüfung von Schaltung und Layout des AD-Wandlers wurden folgende Fehler bzw. Verbesserungsmöglichkeiten gefunden:

- Wandler stört das Datenbussignal
- Beim Verbinden eines DA-Ausgangs mit dem AD-Eingang bricht die Spannung ein
- Der AD-Wandlerbaustein AD9221 benötigt eine CLK-Time von min. 360 ns. (Monoflopschaltung erforderlich)
- Das gewandelte Signal erscheint genau invertiert am Ausgang
- Der bisherige Operationsverstärker schafft nicht den erforderlichen Eingangsspannungsbereich von 0 - 5V für den AD-Wandler

- Timingproblem bei Erzeugung des RD (read)- und WR (write)-Signals zur Ansteuerung der Wandler.

6.2.2 Fehlerbehebung

Da die einzig vorhandene bestückte Platine in einem schlechten Zustand war, konnte die Ursache der Datenbusstörung zunächst nicht genauer lokalisiert werden und wurde zurückgestellt. Das Problem des Spannungseinbruchs lag an einem 50Ω Widerstand am Eingang der Schaltung. Der Fehler konnte mit Entfernen dieses Widerstandes behoben werden.
Der bisherige AD-Wandlerbaustein AD9221 benötigt laut Datenblatt eine Pulsdauer am Clk-Eingang von min. 330 ns. Das aus den RD-, WR- und Adreßsignalen (Lese- und Schreibvorgang) des EZ-Kit Lite erzeugte $Chipselect$ Signal ist aber nur etwa 100 ns lang. Daher wurde ein Timerbaustein auf der Schaltung integriert, der dieses Signal auf ca. 500ns verlängert.
Der AD-Wandler $AD9220$ der gleichen Baureihe benötigt aber laut Datenblatt nur 45 ns Pulsdauer am Clk-Eingang. Daher wurde in der modifizierten AD-Wandlerschaltung der AD9221 gegen den AD9220 ausgetauscht. Damit kann jetzt auch die komplette Timerschaltung entfallen.
Desweiteren wurde festgestellt, daß das Signal am Ausgang der Schaltung invertiert erscheint. Bei max. Eingangsspannung (+5V) am AD9221 waren die Bits am Ausgang alle auf Null und umgekehrt bei min. Eingangsspannung (0V) waren alle Bits eins. Ursache dafür war, daß der Operationsverstärker zur Anpassung des Spannungspegels der DA-Wandler ($\pm$ 2,5V) auf den Eingangsbereich des AD9221 (0 - 5V), als invertierender Verstärker geschaltet ist. Einfache Abhilfe schafft das Vertauschen der Analogeingänge $VINA$ und $VINB$.
Der dazu eingesetzte Operationsverstärker hatte aber den Nachteil, daß seine max. Ausgangsspannung (bei $\pm$ 5V Versorgungsspannung) nicht ausreicht, um den Eingangsspannungsbereich von 0- 5V des AD922x zu gewährleisten. Das Diagramm 6.3 zeigt unter anderem auch den starken Spannungseinbruch bei Frequenzen höher ca. 80 kHz.

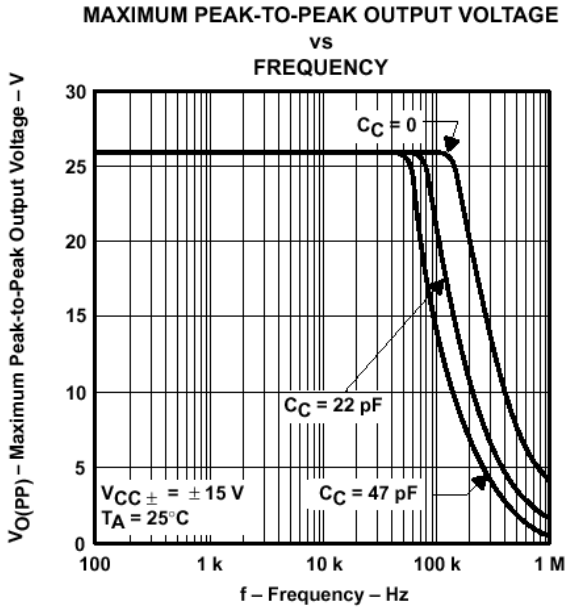


Abbildung 6.3: *OPV NE5534*

Anstelle des Ne5534 wurde in der neuen Schaltung ein *Rail-to-Rail*-Operationsverstärker von Analog Devices eingesetzt. Diese OPV haben die Eigenschaft, nahezu die Versorgungsspannung an den Ausgang zu bringen, wie der Datenblattauszug zeigt.

OUTPUT CHARAKTERISTICS			
Output Voltage Swing	R_L = 10 k to 2.5 V	0.03 to 4.97	V
	R_L = 1 k to 2.5 V	0.05 to 4.95	V

Abbildung 6.4: *Auszug aus Datenblatt AD8042 (Versorgungsspannung + 5V)*

Da sich in einem Dip8-Gehäuse zwei AD8042 befinden, ergibt sich als weiterer Vorteil, daß auf der Platine ein 8-poliger Sockel entfällt, der vorher nötig war, da es den NE5534 nur als $Single-$ und nicht als $Dual$-Typ gibt (es werden zwei OPV pro Platine benötigt).

Letztendlich existiert noch ein Timingproblem bei der Ansteuerung der AD-Wandlerplatine und zwar im Zusammenhang mit dem RD- bzw. WD-Signal des EZ-Kit lite und dem Chipselektsignal der einzelnen Platinen. Dieses Problem trat schon in der vorherigen Diplomarbeit [2] auf, jedoch im Zusammenhang mit den DA-Wandlern. Bei den DA-Wandlern ist es so, daß eine gewisse Zeit **bevor** der DA-Wandler ein CS-Signal erhält, um die Daten zu wandeln, diese Daten durch ein WD-Signal auf den Bus gelegt werden müssen. Dieses Timing wurde daraufhin auch so eingestellt.
Nun verhält es sich bei den AD-Wandlern bzgl. des Timings genau umgekehrt. Hier muß zuerst der Wandler durch ein CS-Signal aufgefordert werden, die analoge Spannung zu wandeln. Erst dann darf ein RD-Signal folgen, um diese Daten in den Speicher zu lesen. Da aber sowohl RD- als auch WR-Signale durch die gleiche Schaltung erzeugt werden, existiert kein unterschiedliches Timing für DA- bzw. AD-Wandler. Die Folge ist, daß der zuerst eingelesene Wert bei einer AD-Wandlung nicht korrekt ist. Dieser ist der letzte, vor Beginn der Wandlung, auf dem Datenbus anliegende binäre Wert.
Dieses Problem wurde nun hardwaretechnisch nicht behoben, da eine Layoutänderung der Hauptplatine nötig gewesen wäre. Dieses muß auf der Softwareseite berücksichtigt werden.

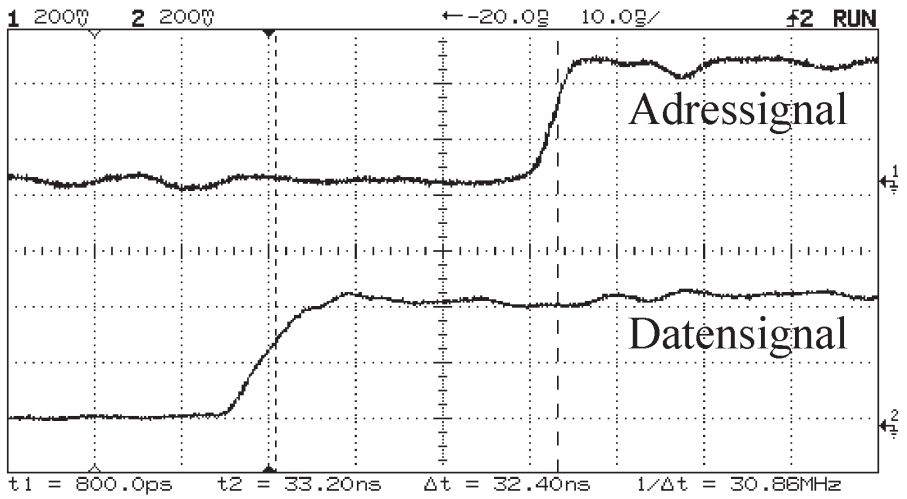


Abbildung 6.5: *Timing (richtig für DA-, falsch für AD-Wandler)*

Nach abgeschlossener Fehlersuche wurde der Wandler zu Testzwecken zunächst auf Lochraster aufgebaut. Nachdem dieser Test erfolgreich verlaufen ist, wurden die nötigen Layoutände-

rungen vorgenommen.
Die vorgenommenen Änderungen in der AD-Schaltung sind in Bild 6.6 zu sehen.

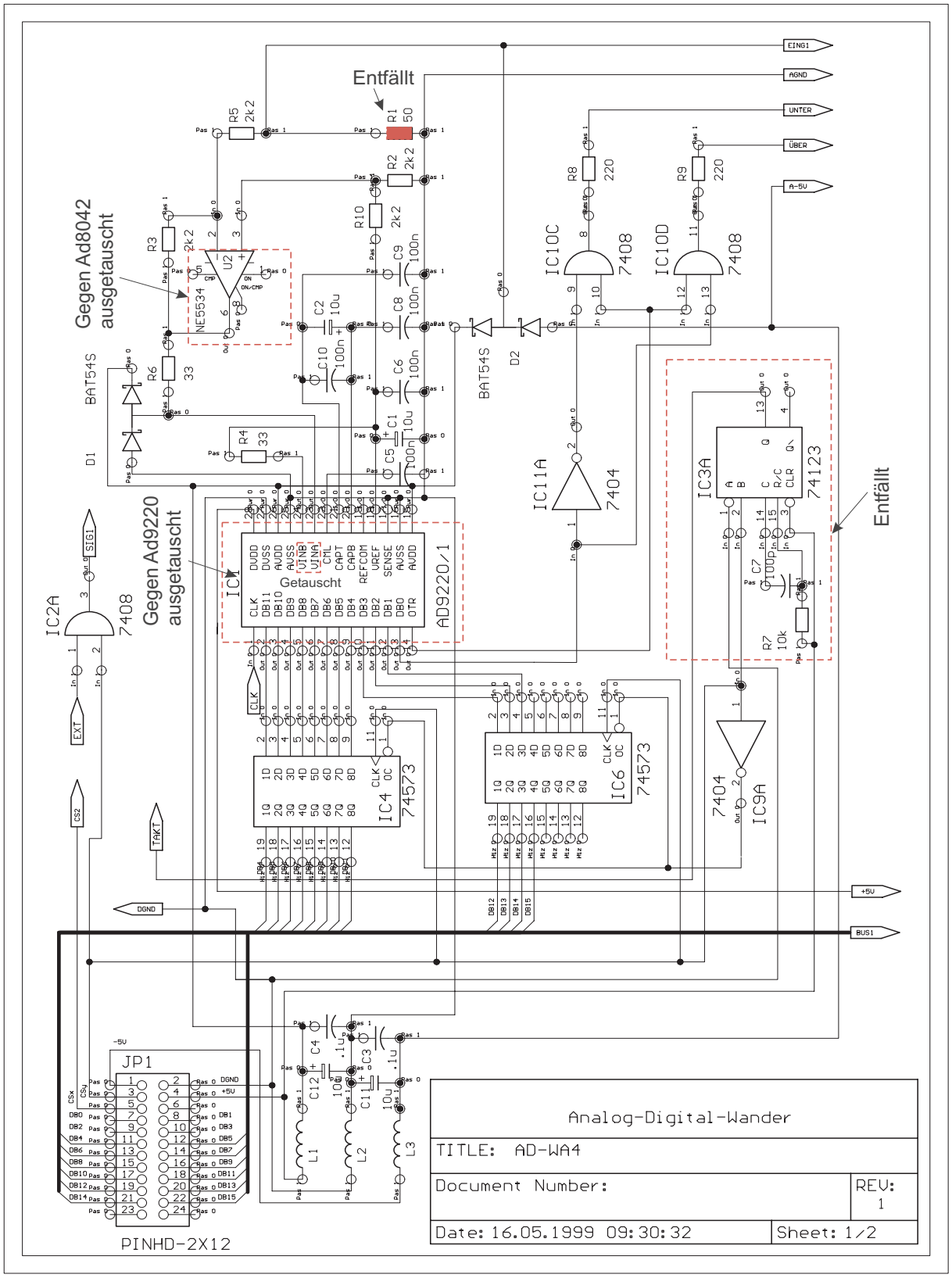


Abbildung 6.6: *Schaltplan AD-Wandler mit Änderungen*

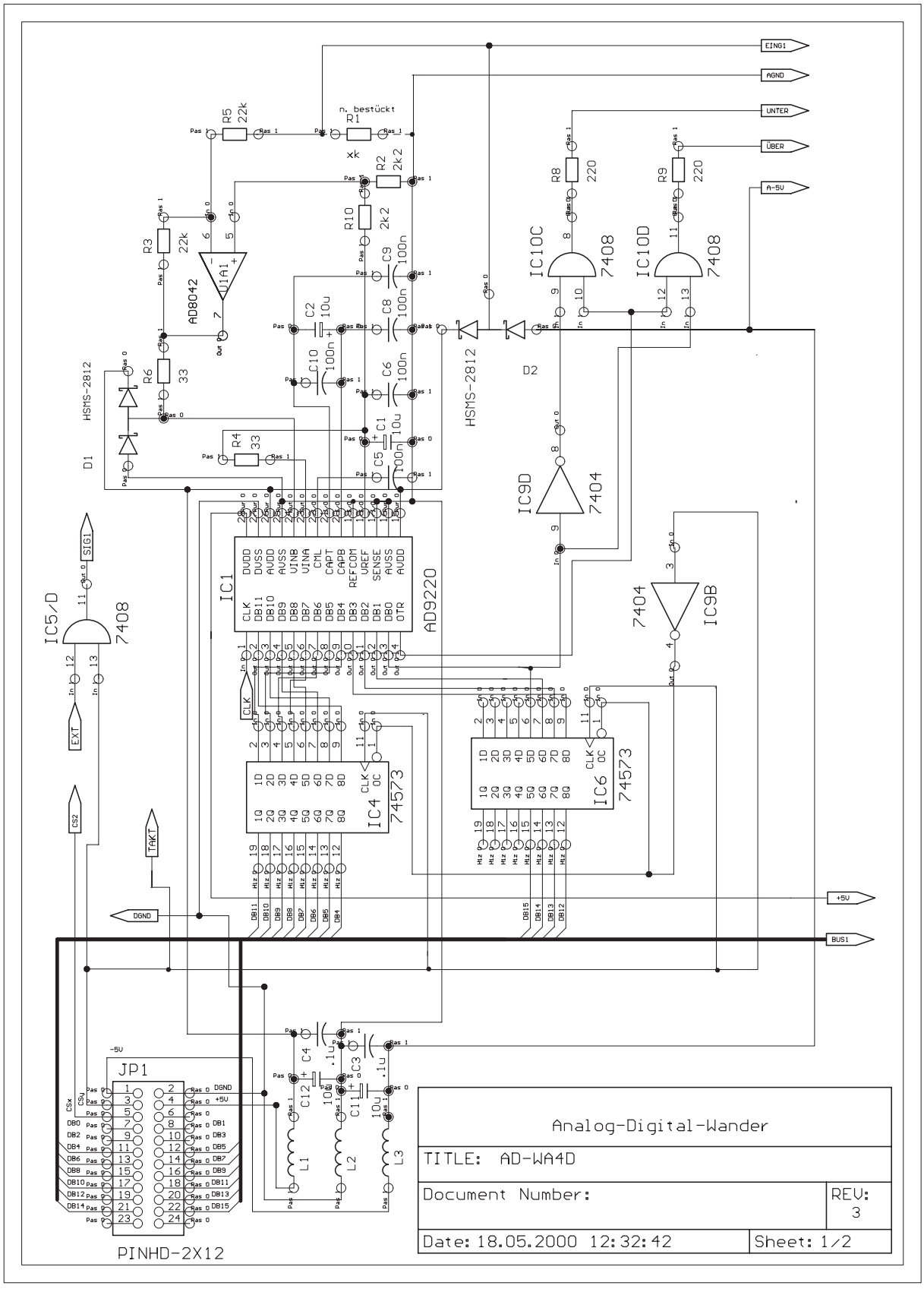


Abbildung 6.7: *Schaltplan AD-Wandler*

6.3 Software

Als Software kommt hier das Entwicklungstool *Visual-DSP* von *Analog Devices* zum Einsatz. Es beinhaltet einen Editor, einen Compiler und einen Debugger.
Die Programme können mit dieser Entwicklungsumgebung sehr komfortabel bearbeitet und in den DSP geladen werden. Da dieses Tool im Praktikum nicht zur Verfügung steht, müssen die Programme noch mit dem Compiler des *EZ-Kit lite* übersetzt und mit dem Programm *Win95Downloader* in den DSP geladen werden.
Weitergehende Informationen zur Software findet man unter [2].

Literatur

[1] *Diplomarbeit Jens Berger.*
Inbetriebnahme und Anpassung eines Evaluation-Kit mit Gleitkomma DSP für den Einsatz im Praktikum.
Fachhochschule Köln 1999

[2] *Diplomarbeit Stefan Lütz.*
Inbetriebnahme und Programmierung eines DSP-basierten Praktikumsaufbau
Fachhochschule Köln 2000

[3] *John G. Proakis.*
Digital communications. McGraw-Hill, New York

[4] *J.P. Morton.*
Adaptive Equalization for Indoor Channels. http://scholar.lib.vt.edu/theses/available/etd-7798-135711/

[5] *Simon Haykin.*
Digital communication. Wiley, New York

[6] *K.D. Kammeyer.*
Nachrichtenübertragung. B.G. Teubner, Stuttgart

A m.-files

A.1 mse.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktion  a=mse(k,kan,SN);
%
% Zeigt einen Ersatzkanal, dessen Spektrum und die Channel-overall-response
% (Kanal+Equalizer). Es werden die Koeffizienten eines linearen MSE-Equalizers
%  berechnet.
%
% Es stehen vier Kanäle zur Auswahl, welche
% unterschiedliche Amplitudenspektren aufweisen.
%
%´Input Parameter: k - Auswahl des Kanals (0-4; 0= eigener Kanal)
%                  kan - eigener Kanal sonst 0
%                  SN - S/N-Ratio (0-35 dB)
% Ausgabe: a=Koeffizienten des Entzerrers
%          Im Matlab-Kommando-Fenster wird die Bitfehlerrate für 1*10^4 Bits
%          angezeigt.
%          Abbildung zeigt ISI-Kanal, Amplitudenspektrum und COR.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function copt = mse(k,kan,SN)

echo off

SNR=SN;

N=10000;
for g=1:N  % Erzeugung einer Infosequenz mit 1 bzw. -1
   if(rand<0.5),
      info(g)=-1;
   else
      info(g)=1;
   end;
end;



% verschiedene Kanalmodelle
if(k==0)
   f=kan;
   elseif(k==1)
   % ISI-Kanal aus Matlab S.270
      f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
   elseif(k==2)
      f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];%ISI-Kanäle aus Proakis S. 616
   elseif(k==3)
      f=[.227 .460 .688 .460 .227];%Proakis
   else
      f=[.407 .815 .407];%Proakis
end


%Berechnung von Copt
% Aus diesen Werten wird die Matrix T(lj) erstellt (Autocorrelationfunktion)
x=xcorr(f);
x=x/x((length(x)+1)/2);% x(0)=1;

 d=length(f);%Erstellung der Matrix Xlj
 r=d-1;
 m=zeros(length(f),length(f));
 for i=1:d
    r=r+1;
    a=r;
    for l=1:d
       m(i,l)=x(a);
       if(i==l)
```

```
            m(i,l)=m(i,l)+10^(-SNR/10); %Auf Hauptdiagonalen wird N0 addiert
          end
          a=a-1;
    end
end
f1=fliplr(conj(f));% Vektor f*(L)-f*(0)
copt=inv(m)*f1.'; %Berechnete Koeffizienten


eq=conv(f,copt);% Channel-overall-response für Koeffizienten aus Berechnung


%Erzeugung von gestörten Daten
data1=conv(info,f);
var=0.5*10^(-SNR/10);
AWG1=randn(1,size(data1,2))*sqrt(var);
AWG2=AWG1;
AWG=AWG1+j*AWG2;
data=data1+AWG;

%Bestimmumg von I(k)=summe [c(j)*v(k-j)] j=-K...K (Proakis Seite 602)
for k=1:length(info)
     v1=fliplr(data(1,k:(k+length(copt)-1))); %Sequenz v(k-j)
     I1(k)=copt.'*v1.';
end


err=0;
 for i=1:length(I1)
   if real(I1(i))>=0
      bit(i)=1;
       if(bit(i)~=info(i))
         err=err+1;
       end
   else
      bit(i)=-1;
        if(bit(i)~=info(i))
         err=err+1;
        end
   end
  end

Bitfehler=err

%Frequenzantwort des Kanals
[G_T,W]=freqz(f,1);
G_t=20*log10(abs(G_T)/max(abs(G_T)));



subplot(2,2,3)
stem(abs(eq))
title('Channel-overall-response')
subplot(2,2,2)
plot(W,G_t)
axis([0 pi -30 0])
ylabel('Amplitude (dB)')
xlabel('Frequency \omega')
title('Amplitudenspektrum')
subplot(2,2,1)
stem(real(f))
title('ISI-Kanal')
set(1,'Name','Übersicht')
```

A.2 dfe.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktion  [a,b]=dfe(k,kan,SN);
%
% Zeigt für einen Ersatzkanal die Signale vor dem Equalizer, nach dem FF-Filter
% und nach dem Summierer. Es werden die Koeffizienten der Filter (FF- und FB-Filter)
% berechnet.
%
% Es stehen vier Kanäle zur Auswahl, welche unterschiedliche Amplitudenspektren
% aufweisen.
%
%´Input Parameter: k - Auswahl des Kanals (0-4; 0 = eigener Kanal)
%                  kan - eigener Kanal sonst 0
%                  SN - S/N-Ratio (0-35 dB)
% Ausgabe: a=Koeffizienten des FF-Filters
%          b=Koeffizientem des FB-Filters
%          Im Matlab-Kommando-Fenster wird die Bitfehlerrate für 1*10^4 Bits
%          angezeigt.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [cDFE,bDFE] = dfe(k,kan,SN);

echo off
%verschiedene Ersatzkanalmodelle
if(k==0)
   f=kan;
   elseif(k==1)
   % ISI-Kanal aus Matlab S.270
      f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
   elseif(k==2)
      f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];%ISI-Kanal aus Proakis
   elseif(k==3)
      f=[.227 .460 .688 .460 .227];
   elseif(k==4)
      f=[.407 .815 .407];
end

L=length(f);
K1=L-1; %K1+1=Anzahl der FF-Koeffizienten
K2=L-1; %K2=Anzahl FB-Koeffizienten
SNRdfe=SN;
err=0;

N=10000;
for g=1:N  % Erzeugung einer Infosequenz mit 1 bzw. -1
   if(rand<0.5),
      info(g)=-1;
   else
      info(g)=1;
   end;
end;

%Berechnung der Koeffizienten
%Bestimmung der psi-matrix
      psi = zeros(K1+1,K1+1);
      for k=-K1:0
         for jo=-K1:0
          for i=0:-k
              if ( ((i+1+k-jo)>0) & ((i+1+k-jo)<L+1) & (i<L))
               psi(k+K1+1,jo+K1+1)=psi(k+K1+1,jo+K1+1)+(conj(f(i+1))*f(i+1+k-jo))
              end
          end
         end
      end

      for k=-K1:0
          psi(k+K1+1,k+K1+1) = psi(k+K1+1,k+K1+1) + 10^(-SNRdfe/10);
      end
```

```
        %Koeffizienten des FF-Filters
        cDFE = (psi \ fliplr(f(1:K1+1))').';

        %Koeffizienten des FB-Filters
        bDFE=zeros(1,K2);
         for k = 1:K2
             for jo=-K1:0
                if( (k-jo+1) <= L )
                   bDFE(k) = bDFE(k) - cDFE(jo+K1+1) * f(k-jo+1);
                end
             end
         end

     %gestörte Datensequenz
     data1=conv(info,f);
     var=0.5*10^(-SNRdfe/10);
     AWG1=randn(1,size(data1,2))*sqrt(var);
     AWG2=AWG1;
     AWG=AWG1+j*AWG2;
     v=data1+AWG;


     fff = zeros(1,L);
     fbf = zeros(1,L-1);

    for i = 1:length(info)

            %nach FF-Filter
            fff = fliplr(v(1,i:(i+K1)));
            I1(i)=sum(cDFE.*fff);

            %nach FB-Filter
            I2(i)=sum(bDFE.*fbf);

            % Summe von FF + FB
            d_hat(i)=I1(i)+I2(i);

             %Bitentscheidung
             if real(d_hat(i)) >=0
               d_bar(i) = 1;
                if(d_bar(i)~=info(i))
                  err=err+1;
                end
               else
                d_bar(i) = -1;
                 if(d_bar(i)~=info(i))
                  err=err+1;
                 end
              end

            % Anpassen des FB_Filters
            fbf(1,2:K2)=fbf(1,1:(K2-1));
            fbf(1,1)=d_bar(i) ;
     end


    %Ausagabe Bitfehler
    Bitfehler=err

    %Grafiken
    t=conv(1,f);%Signal vor Equalizer
    p=conv(t,cDFE);%Signal nach FF-Filter
    p1=p;
    r=conv(1,bDFE);%Signal nach FB-Filter

    %Signal nach Addition
    for i=1:length(r)
       p(length(f)+i)=p(length(f)+i)+r(i);
    end

    subplot(2,2,1)
    stem(real(t))
```

```
    title('vor Equalizer')
    subplot(2,2,2)
    stem(real(p1))
    title('nach FF-Filter')
    subplot(2,2,3)
    stem(real(p))
    title('nach Addition')
150 set(1,'Name','Koeffizienten berechnet')
```

A.3 kanalmodell.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Funktion [a,b,c,d]=kanalmodell;
%
% Das Programm dient zur Ermittlung von Filterkoeffizienten.
% Es werden die Koeffizienten des Sendefilters ’tran’ (root raised cosine), des
% Filters aaus der Zusammenfassung aus Sneder und Knal (sendekan), des
% Matchedfilters ’mf’ und des Ersatzkanalmodells ’Kanal’ (Sender+Kanal+MF)
% ausgegeben. Es wird mit einer achtfachen Überabtastung gearbeitet.
% Das Downsampeln der Gesamtimpulsantwort zur Ermittlung der Ersatzfilter-
% koeffizienten richtet sich nach der max. Signalenergie, die erreicht werden kann.
%
% Ausgabe: a=Koeffizienten Sendefilter
%          b=     "        Sendefilter+Kanal
%          c=     "        Matchedfilter
%          d=     "        Ersatzkanalmodell
%
%          Grafiken zeigen das Sendefilter, Sendefilter+Kanal+MF,
%          Ersatzkanalmodell und dessen Amplitudenspektrum.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function [tran,sendekan,mf,kanal]=kanalmodell()

%Ermittlung des Übertragungskanal
%Fetsichan(taps,delay_spread/ns,1,fs/MHz)
%Störung über zwei Symbole-> ~16 Taps; taps=7*delay_spread/Ts -> Ts=1/fs=20ns
%-> taps~17
kan=Fetsichan(16,50,1,50);

%Transmitter
% achtfache Überabtastung bedeutet 8 Koeffizienten zwischen 0 und T
% -> Schrittweite 0.125
t=-5:0.125:5;
tran=rrcn(0.5,t)        ; %Transmitterfilter Root-Raised-Cosine (Stefan Lütz)

sendekan=conv(tran,kan);%Faltung Transmitterfilter und Kanal
z1=conj(sendekan);% z*
mf=fliplr(z1);%z*(-t) Matchedfilter
f=conv(sendekan,mf); % Faltung Matchedfilter-(Transmitter+Kanal)
f=f/max(abs(f));

%%%     Gewinnung der Koeffizienten f(k) des Ersatzkanalmodells %%%

%Abtasten 8-fach Oversampling -> jeder achte Wert

e=zeros(1,length(f));%max. Energie der Impulsantwort finden
for a=1:length(f)
   for i=a:8:(length(f)) %1,9,17..Abtastwert; 2,10,18..Abtastwert usw.
      e(a)=e(a)+(abs(f(i))^2); %Energie
   end
 a=a+1;
end

r=max(e);%max. Energie

for i=1:length(e);
   if(e(i)==r)
     k=i;% erster Abtastwert
      if(k>8)
        k=k-8;
      end
   end
end


d=fix((length(f)-k)/8);
kanal=zeros(1,d+1);
g=1;
```

```
for i=k:8:length(f) %Abtastung der Kanalimpulsantwort mit k*T
   kanal(g)=f(i); %Ersatzfiltermodell!!!
   g=g+1;
end



%Frequenzantwort des Kanals
[G_T,W]=freqz(kanal,1);
G_t=20*log10(abs(G_T)/max(abs(G_T)));

subplot(2,2,1)
stem(t,tran)
title('Sendefilter')
subplot(2,2,4)
stem(real(kanal))
title('Ersatzfilter')
subplot(2,2,2)
stem(abs(kan))
title('Kanal')
subplot(2,2,3)
stem(real(f))
title('Transmitter+Kanal+Matchedfilter')

figure(2)
plot(W,G_t)
axis([0 pi -30 0])
ylabel('Amplitude (dB)')
xlabel('Frequency \omega')
title('Amplitudenspektrum Ersatzkanal')
```

A.4 msetest.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktion  msetest(k1,kan1,kan2,N);
%
% Zeigt für einen Kanal die Bitfehlerwahrscheinlichkeit
% in Abhänigkeit vom S/N-Verhältnis. Es stehen vier Kanäle zur Auswahl, welche
% unterschiedliche Amplitudenspektren aufweisen.
%
%´Input Parameter: k1 - Auswahl des Kanäle (0-4; 0 = eigener Kanal)
%                  kan1,kan2 - eigene Kanäle
%                  N - Länge der Testsequenz (min. 10000, max. 30000)
%                 (größere Werte verlangen zu viel Rechenzeit)
% Ausgabe: Im Matlab-Kommando-Fenster werden jeweils die aktuellen S/N-Werte
%          und die Bitfehlerrate angezeigt.
%          Die Grafiken zeigen den Ersatzkanal, dessen Amplitudenspektrum,
%          ein entzerrtes Bit und die Bitfehlerrate.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function mse = msetest(k1,kan1,kan2,N);

echo off


N1=N;
   for g=1:N1  % Erzeugung einer Infosequenz mit 1 bzw. -1
     if(rand<0.5),
        info(g)=-1;
       else
        info(g)=1;
     end
   end

%erster Kanal

   % verschiedene Kanalmodelle
   if(k1==0)
      f=kan1;
      SN1=[3 5  7  9 10 12 15 18 20 22 25 27 30 33 ];
      elseif(k1==1)
      % ISI-Kanal aus Matlab S.270
         f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
         SN1=[4  5  6  7  8  9  10 11];
      elseif(k1==2)
         %ISI-Kanäle aus Proakis S. 616
         f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];
         SN1=[5  6  7  8  9  10  11];
      elseif(k1==3)
         f=[.227 .460 .688 .460 .227];%Proakis
         SN1=[10 12 15 18 20 22 25 27 30 33 ];
      else
         f=[.407 .815 .407];%Proakis
         SN1=[10 12 15 18 20 22 25 27 30 33 ];
        end

%Berechnung von Copt
% Aus diesen Werten wird die Matrix T(lj) erstellt (Autocorrelationfunktion)
x=xcorr(f);
x=x/x((length(x)+1)/2);% x(0)=1;


fehler=zeros(1,length(SN1));


for h=1:length(SN1)%Berechnung der Bitfehlerraten
   err1=0;

   %Berechnung der Equalizerkoeffizienten
   d=length(f);%Erstellung der Matrix Xlj
   r=d-1;
   m=zeros(length(f),length(f));
```

```
      for i=1:d
         r=r+1;
         a=r;
          for l=1:d
             m(i,l)=x(a);
             if(i==l)
                m(i,l)=m(i,l)+10^(-SN1(h)/10); %Auf Hauptdiagonalen wird N0 addiert
             end
           a=a-1;
80        end
      end

    f1=fliplr(conj(f));% Vektor f*(L)-f*(0)
    copt=inv(m)*f1.'; %Berechnete Koeffizienten

    eq1=conv(1,f);
    eq=conv(eq1,copt);% Channel-overall-response für Koeffizienten aus Berechnung


90

    data1=conv(info,f); %Erzeugung von gestörten Daten
    var=0.5*10^(-SN1(h)/10);
    AWG1=randn(1,size(data1,2))*sqrt(var);
    AWG2=AWG1;
    AWG=AWG1+j*AWG2;
    data=data1+AWG;

   %Berechnung Equalizerausgang
     for k=1:length(info) %Bestimmumg von I(k)=summe [c(j)*v(k-j)] j=-K...K
100    v1=fliplr(data(1,k:(k+length(copt)-1))); %Sequenz v(k-j)
       I1(k)=copt.'*v1.';
     end


      %Bitentscheidung
      for i=1:length(I1)
        if real(I1(i))>=0
           bit(i)=1;
               if(bit(i)~=info(i))
110               err1=err1+1;
               end
        else
             bit(i)=-1;
                   if(bit(i)~=info(i))
                  err1=err1+1;
            end
        end
      end

120 fehler(h)=err1/N1;
    aktSN=SN1(h)
    Bitfehler=fehler(h)

    end

    Bitfehler=fehler

    %Frequenzantwort des Kanals
    [G_T,W]=freqz(f,1);
130 G_t=20*log10(abs(G_T)/max(abs(G_T)));

     %Grafik
     subplot(2,2,3)
     stem(real(eq))
     title('Entzerrtes Bit')
     subplot(2,2,2)
     plot(W,G_t)
     axis([0 pi -30 0])
     ylabel('Amplitude (dB)')
140  xlabel('Frequency \omega')
     title('Amplitudenspektrum')
     subplot(2,2,1)
```

```
stem(real(f))
title('ISI-Kanal')
subplot(2,2,4)
semilogy(SN1,fehler)
axis([0 35 0.0001 0.2])
xlabel('SN')
ylabel('Bit-Fehlerwahrscheinlichkeit')
set(1,'Name','Kanal 1')



%zweiter Kanal

% verschiedene Kanalmodelle
if(k1==0 & kan2~=0)
   f=kan2;
   SN=[3 5  7  9 10 12 15 18 20 22 25 27 30 33 ];


  %Berechnung von Copt
  % Aus diesen Werten wird die Matrix T(lj) erstellt (Autocorrelationfunktion)
  x=xcorr(f);
  x=x/x((length(x)+1)/2);% x(0)=1;

  fehler1=zeros(1,length(SN));

for h=1:length(SN)%Berechnung der Bitfehlerraten
  err1=0;

%Berechnung der Equalizerkoeffizienten
d=length(f);%Erstellung der Matrix Xlj
r=d-1;
m=zeros(length(f),length(f));
  for i=1:d
      r=r+1;
      a=r;
      for l=1:d
          m(i,l)=x(a);
          if(i==l)
            m(i,l)=m(i,l)+10^(-SN(h)/10); %Auf Hauptdiagonalen wird N0 addiert
          end
        a=a-1;
     end
  end

f1=fliplr(conj(f));% Vektor f*(L)-f*(0)
copt=inv(m)*f1.'; %Berechnete Koeffizienten

eq1=conv(1,f);
eq=conv(eq1,copt);% Channel-overall-response für Koeffizienten aus Berechnung


data1=conv(info,f);%gestörte Datensequenz
var=0.5*10^(-SN(h)/10);
AWG1=randn(1,size(data1,2))*sqrt(var);
AWG2=AWG1;
AWG=AWG1+j*AWG2;%Rauschen
data=data1+AWG;



  %Berechnung Equalizerausgang
  for k=1:length(info) %Bestimmumg von I(k)=summe [c(j)*v(k-j)] j=-K...K
      v1=fliplr(data(1,k:(k+length(copt)-1))); %Sequenz v(k-j)
      I1(k)=copt.'*v1.';
  end


%Bitentscheidung
for i=1:length(I1)
    if real(I1(i))>=0
       bit(i)=1;
```

```
            if(bit(i)~=info(i))
               err1=err1+1;
            end
         else
          bit(i)=-1;
            if(bit(i)~=info(i))
               err1=err1+1;
            end
      end
   end

   fehler1(h)=fehler1(h)+err1/N1;
   aktSN=SN(h)
   Bitfehler1=fehler1(h)

end

   error1=fehler1

   %Frequenzantwort des Kanals
   [G_T,W]=freqz(f,1);
   G_t=20*log10(abs(G_T)/max(abs(G_T)));

    %Grafik
    figure(2)
    subplot(2,2,3)
    stem(abs(eq))
    title('Entzerrtes Bit')
    subplot(2,2,2)
    plot(W,G_t)
    axis([0 pi -30 0])
    ylabel('Amplitude (dB)')
    xlabel('Frequency \omega')
    title('Amplitudenspektrum')
    subplot(2,2,1)
    stem(real(f))
    title('ISI-Kanal')
    subplot(2,2,4)
    semilogy(SN,fehler1,SN1,fehler)
    axis([0 35 0.0001 0.2])
    xlabel('SN')
    ylabel('Bit-Fehlerwahrscheinlichkeit')
    legend('Kanal(2)','Kanal(1)')
    set(2,'Name','Kanal 2')

end
```

A.5 dfetest.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Funktion  dfetest(k1,kan1,kan2,N);
%
% Zeigt für einen Kanal die Bitfehlerwahrscheinlichkeit
% in Abhänigkeit von S/N-Verhältnis. Es stehen vier Kanäle zur Auswahl, welche
% unterschiedliche Amplitudenspektren aufweisen.
%
%´Input Parameter: k1,k2 - Auswahl des Kanäle (0-5; 0 = eigener Kanal)
%                  kan - eigener Kanal sonst 0
%                  N - Länge der Testsequenz (min. 10000, max. 30000)
%                     (größere Werte verlangen zu viel Rechenzeit)
% Ausgabe: Im Matlab-Kommando-Fenster werden jeweils die aktuellen S/N-Werte
%          und die Bitfehlerrate angezeigt.
%          Die Grafiken zeigen das Signal vor dem Equalizer, nach FF-Filter
%          und nach der Additionsstelle und die Bitfehlerrate.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



function dfe = dfetest(k1,kan1,kan2,N);

echo off



N1=N;
  for g=1:N1  % Erzeugung einer Infosequenz mit 1 bzw. -1
     if(rand<0.5),
       info(g)=-1;
      else
       info(g)=1;
     end
  end

%erster Kanal

   % verschiedene Kanalmodelle
   if(k1==0)
      f=kan1;
      SN=[4 5 6 7 8 9 10 12 14 16 18 20];
   elseif(k1==1)
      % ISI-Kanal aus Matlab S.270
      f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
      SN=[4  5  5.5  6    7   8  10 ];
   elseif(k1==2)
      f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];%ISI-Kanäle aus Proakis S. 616
      SN=[4  5  5.5  6   7   8   10  ];
   elseif(k1==3)
      f=[.227 .460 .688 .460 .227];%Proakis
      SN=[4 6 8 10 12 15 17 20];
   else
      f=[.407 .815 .407];%Proakis
      SN=[4 6 8 10 12 15 17 20 ];
   end

L=length(f);
K1=L-1;
K2=L-1;


fehler=zeros(1,length(SN));


for h=1:length(SN)%Berechnung der Bitfehlerraten
   err1=0;

   %Berechnung der Filterkoffizienten
   %psi-matrix
       psi = zeros(K1+1,K1+1);
```

```
          for k=-K1:0
           for jo=-K1:0
            for i=0:-k
             if ( ((i+1+k-jo)>0) & ((i+1+k-jo)<L+1) & (i<L))
              psi(k+K1+1,jo+K1+1) = psi(k+K1+1,jo+K1+1) + (conj(f(i+1))*f(i+1+k-jo));
             end
            end
           end
          end
80
           for k=-K1:0
              psi(k+K1+1,k+K1+1) = psi(k+K1+1,k+K1+1) + 10^(-SN(h)/10);
           end

           %FF-Koeffizienten
           cDFE = (psi \ fliplr(f(1:K1+1))').';

           %FB Koeffizienten
           bDFE=zeros(1,K2);
90         for k = 1:K2
               for jo=-K1:0
                  if( (k-jo+1) <= L )
                     bDFE(k) = bDFE(k) - cDFE(jo+K1+1) * f(k-jo+1);
                  end
               end
           end



100 data1=conv(info,f); %gestörte Datensequenz
    var=0.5*10^(-SN(h)/10);
    AWG1=randn(1,size(data1,2))*sqrt(var);
    AWG2=AWG1;
    AWG=AWG1+j*AWG2;%Rauschen
    v=data1+AWG;



    %Berechnung Equalizerausgang
110 fff = zeros(1,L);
    fbf = zeros(1,L-1);

        for i = 1:length(info)


           %Nach FF-Filter
           fff = fliplr(v(1,i:(i+K1)));
           I1(i)=sum(cDFE.*fff);

120        %Nach FB-Filter
           I2(i)=sum(bDFE.*fbf);

           %Summe
           d_hat(i)=I1(i)+I2(i);

           %Bitentscheidung
           if real(d_hat(i)) >=0
              d_bar(i) = 1;
              if(d_bar(i)~=info(i))
130             err1=err1+1;
              end
            else
               d_bar(i) = -1;
               if(d_bar(i)~=info(i))
                 err1=err1+1;
               end
            end

           %Anpassung FB-Filter
140        fbf(1,2:K2)=fbf(1,1:(K2-1));
           fbf(1,1)=d_bar(i);
```

```
        end

    fehler(h)=err1/N1;

    aktSN=SN(h)
    Bitfehler=fehler(h)

150 end

    SN=SN
    Bitfehler=fehler

    %Frequenzantwort des Kanals
    [G_T,W]=freqz(f,1);
    G_t=20*log10(abs(G_T)/max(abs(G_T)));


160 t=conv(1,f);%gestörtes Bit
    p=conv(t,cDFE);%nach FF-Filter
    p1=p;
    r=conv(1,bDFE);%nach FB-Filter
    %nach Addition
    for i=1:length(r)
       p(length(f)+i)=p(length(f)+i)+r(i);
    end

    subplot(2,2,1)
170 stem(real(t))
    title('vor Equalizer')
    subplot(2,2,2)
    stem(real(p1))
    title('nach FF-Filter')
    subplot(2,2,3)
    stem(real(p))
    title('nach Addition')
    subplot(2,2,4)
    semilogy(SN,fehler)
180 axis([0 35 0.0001 0.5])
    xlabel('SN')
    ylabel('Bit-Fehlerwahrscheinlichkeit')
    set(1,'Name','Kanal1')


     %zweiter Kanal

    % verschiedene Kanalmodelle
    if(k1==0 & kan2~=0)
190    {
          f=kan2;
          SN1=[4 5 6 7 8 9 10 12 14 16 18 20];

    L=length(f);
    K1=L-1;
    K2=L-1;


    fehler2=zeros(1,length(SN1));
200


    for h=1:length(SN1)%Berechnung der Bitfehlerraten
       err1=0;

       %Berechnung der Filterkoeffizienten
       %psi-matrix
       psi = zeros(K1+1,K1+1);
       for k=-K1:0
        for jo=-K1:0
210      for i=0:-k
           if ( ((i+1+k-jo)>0) & ((i+1+k-jo)<L+1) & (i<L))
             psi(k+K1+1,jo+K1+1) = psi(k+K1+1,jo+K1+1) + (conj(f(i+1)) * f(i+1+k-jo));
           end
         end
```

```
        end
       end

          for k=-K1:0
             psi(k+K1+1,k+K1+1) = psi(k+K1+1,k+K1+1) + 10^(-SN1(h)/10);
220       end

          %FF-Koeffizienten
          cDFE = (psi \ fliplr(f(1:K1+1))').';

          %FB-Koeffizienten
          bDFE=zeros(1,K2);
          for k = 1:K2
              for jo=-K1:0
                 if( (k-jo+1) <= L )
230                 bDFE(k) = bDFE(k) - cDFE(jo+K1+1) * f(k-jo+1);
                 end
              end
          end



    data1=conv(info,f); %gestörte Datensequenz
    var=0.5*10^(-SN1(h)/10);
    AWG1=randn(1,size(data1,2))*sqrt(var);
240 AWG2=AWG1;
    AWG=AWG1+j*AWG2;
    v=data1+AWG;



    %Berechnung Equalizerausgang
    fff = zeros(1,L);
    fbf = zeros(1,L-1);

250   for i = 1:length(info)

          %Nach FF-Filter
          fff = fliplr(v(1,i:(i+K1)));
          I1(i)=sum(cDFE.*fff);


          %Nach FB-Filter
          I2(i)=sum(bDFE.*fbf);

260       %Summe
          d_hat(i)=I1(i)+I2(i);

           %Bitentscheidung
           if real(d_hat(i)) >=0
             d_bar(i) = 1;
               if(d_bar(i)~=info(i))
                 err1=err1+1;
               end
           else
270          d_bar(i) = -1;
               if(d_bar(i)~=info(i))
                 err1=err1+1;
               end
           end

           %Anapssung FB-Filter
           fbf(1,2:K2)=fbf(1,1:(K2-1));
           fbf(1,1)=d_bar(i) ;

280    end

          fehler2(h)=err1/N1;
          aktSN1=SN1(h)
          Bitfehler1=fehler2(h)

    end
```

```
 SN2=SN1
 Bitfehler1=fehler2
%Frequenzantwort des Kanals
[G_T,W]=freqz(f,1);
G_t=20*log10(abs(G_T)/max(abs(G_T)));


t=conv(1,f);%gestörtes Bit
p=conv(t,cDFE);%nach FF-Filter
p1=p;
r=conv(1,bDFE);%nach FB-Filter
%nach Addition
for i=1:length(r)
   p(length(f)+i)=p(length(f)+i)+r(i);
end

figure(2)
subplot(2,2,1)
stem(real(t))
title('vor Equalizer')
subplot(2,2,2)
stem(real(p1))
title('nach FF-Filter')
subplot(2,2,3)
stem(real(p))
title('nach Addition')
subplot(2,2,4)
semilogy(SN,fehler,SN1,fehler2)
axis([0 35 0.0001 0.5])
xlabel('SN')
ylabel('Bit-Fehlerwahrscheinlichkeit')
legend('Kanal1','Kanal2')
set(2,'Name','Kanal2')


end
```

A.6 msedfe.m

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Funktion  msedfe(k,kan,N)
%
% Vergleicht für einen Kanal die Bitfehlerwahrscheinlichkeit eines MSE- und DFE-
% Equalizers.
% in Abhänigkeit von S/N-Verhältnis. Es stehen vier Kanäle zur Auswahl, welche
% unterschiedliche Amplitudenspektren aufweisen.
%
%´Input Parameter: k - Auswahl des Kanals (0-4; 0 = eigener Kanal)
%                  kan - eigener Kanal sonst 0
%                  N - Länge der Testsequenz (min. 10000, max. 30000)
%                      (größere Werte verlangen zu viel Rechenzeit)
%
% Ausgabe: Im Matlab-Kommando-Fenster werden jeweils die aktuellen S/N-Werte und die
% Bitfehlerrate angezeigt.
% In Abbildung 1 wird der DFE-Equalizer, in Abb. 2 der MSE-Equalizer angezeigt.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


function msedfe(k,kan,N);

echo off

k1=k;

N1=N;
   for g=1:N1  % Erzeugung einer Infosequenz mit 1 bzw. -1
     if(rand<0.5),
        info(g)=-1;
     else
        info(g)=1;
     end
   end

%DFE-Berechnung

   % verschiedene Kanalmodelle
   if(k==0)
      f=kan;
      SN1=[4 5 6 7 8 9 10 12 15 17 20 25 30];
      elseif(k==1)
      % ISI-Kanal aus Matlab S.270
         f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
        SN1=[4 4.5 5 5.5  6 6.5 7 7.5  8 8.5  9];
      elseif(k==2)
         %ISI-Kanäle aus Proakis S. 616
         f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];
         SN1=[4  5   6  7   8   9];
      elseif(k==3)
         f=[.227 .460 .688 .460 .227];%Proakis
         SN1=[4 6 8 10 12 15 17 20];
      else
         f=[.407 .815 .407];%Proakis
         SN1=[4 6 8 10 12 15 17 20 ];
   end

L=length(f);
K1=L-1;
K2=L-1;


fehler=zeros(1,length(SN1));


for h=1:length(SN1)%Berechnung Bitfehlerraten
   err1=0;

   %Berechnung der Filterkoeffizienten(DFE)
   %psi-matrix
```

```
          psi = zeros(K1+1,K1+1);
           for k=-K1:0
             for jo=-K1:0
              for i=0:-k
                if ( ((i+1+k-jo)>0) & ((i+1+k-jo)<L+1) & (i<L))
                 psi(k+K1+1,jo+K1+1)=psi(k+K1+1,jo+K1+1)+(conj(f(i+1))*f(i+1+k-jo));
                end
               end
              end
80          end

            for k=-K1:0
               psi(k+K1+1,k+K1+1) = psi(k+K1+1,k+K1+1) + 10^(-SN1(h)/10);
            end

            %FF-Koeffizienten
            cDFE = (psi \ fliplr(f(1:K1+1))').';

            %FB-Koeffizienten
90           bDFE=zeros(1,K2);
            for k = 1:K2
                for jo=-K1:0
                   if( (k-jo+1) <= L )
                      bDFE(k) = bDFE(k) - cDFE(jo+K1+1) * f(k-jo+1);
                   end
                end
            end


100
    data1=conv(info,f); %gestörte Datensequenz
    var=0.5*10^(-SN1(h)/10);
    AWG1=randn(1,size(data1,2))*sqrt(var);
    AWG2=AWG1;
    AWG=AWG1+j*AWG2;%Rauschen
    v=data1+AWG;



110 %Berechnung Equalizerausgang
    fff = zeros(1,L);
    fbf = zeros(1,L-1);

         for i = 1:length(info)

           %Nach FF-Filter
           fff = fliplr(v(1,i:(i+K1)));
           I1(i)=sum(cDFE.*fff);

120
           % Nach FB-Filter
           I2(i)=sum(bDFE.*fbf);

            % Summe FF+FB
            d_hat(i)=I1(i)+I2(i);

             %Bit Entscheidung
            if real(d_hat(i)) >=0
              d_bar(i) = 1;
130            if(d_bar(i)~=info(i))
                  err1=err1+1;
               end
            else
              d_bar(i) = -1;
               if(d_bar(i)~=info(i))
                 err1=err1+1;
               end
          end

140         % Anpassung FB-Filter
            fbf(1,2:K2)=fbf(1,1:(K2-1));
            fbf(1,1)=d_bar(i) ;
```

```
      end

fehler(h)=err1/N1;

aktSNRDFE=SN1(h)
BitfehlerDFE=fehler(h)
end

SNDFE=SN1
BitfehlerDFE=fehler

t=conv(1,f);%gestörtes Bit
p=conv(t,cDFE);%nach FF-Filter
p1=p;
r=conv(1,bDFE);%nach FB-Filter

%nach Addition
for i=1:length(r)
   p(length(f)+i)=p(length(f)+i)+r(i);
end

%Grafik
figure(1)
subplot(2,2,1)
stem(real(t))
title('vor Equalizer')
subplot(2,2,2)
stem(real(p1))
title('nach FF-Filter')
subplot(2,2,3)
stem(real(p))
title('nach Addition')
subplot(2,2,4)
semilogy(SN1,fehler)
axis([0 35 0.0001 0.5])
xlabel('SN')
ylabel('Bit-Fehlerwahrscheinlichkeit')
set(1,'Name','DFE')


 %zweiter Kanal


   % verschiedene Kanalmodelle
   if(k1==0)
      f=kan;
      SN=[5 7 9 10 12 15 20 25 30 35];
      elseif(k1==1)
      % ISI-Kanal aus Matlab S.270
      f=[.05 -.063 .088 -.126 -.25 .9047 .25 0 .126 .038 .088];
         SN=[4  5  6  7  8  9  10  11];
      elseif(k1==2)
         %ISI-Kanäle aus Proakis S. 616
         f=[.04 -.05 .07 -.21 -.5 .72 .36 0 .21 .03 .07];
         SN=[4  5   6   7   8   9   10  11];
      elseif(k1==3)
         f=[.227 .460 .688 .460 .227];%Proakis
         SN=[10 12 15 17 20 25 30 33];
      else
         f=[.407 .815 .407];%Proakis
         SN=[10 12 15 17 20 25 30 33];

    end

%Berechnung von Copt
% Aus diesen Werten wird die Matrix T(lj) erstellt (Autocorrelationfunktion)
x=xcorr(f);
x=x/x((length(x)+1)/2);% x(0)=1;

fehler1=zeros(1,length(SN));
```

```
for h=1:length(SN)%Berechnung Bitfehlerraten
   err1=0;

%Berechnung der Filterkoeffizienten(DFE)
d=length(f);%Erstellung der Matrix Xlj
r=d-1;
m=zeros(length(f),length(f));
 for i=1:d
    r=r+1;
      a=r;
       for l=1:d
         m(i,l)=x(a);
           if(i==l)
             m(i,l)=m(i,l)+10^(-SN(h)/10); %Auf Hauptdiagonalen wird N0 addiert
           end
         a=a-1;
       end
   end

f1=fliplr(conj(f));% Vektor f*(L)-f*(0)
copt=inv(m)*f1.'; %Berechnete Koeffizienten

bit=conv(1,f);% Channel-overall-response für Koeffizienten aus Berechnung
eq=conv(bit,copt);%nach Equalizer


data1=conv(info,f); %gestörte Datensequenz
var=0.5*10^(-SN(h)/10);
AWG1=randn(1,size(data1,2))*sqrt(var);
AWG2=AWG1;
AWG=AWG1+j*AWG2;%Rauschen
data=data1+AWG;



%Berechnung Equalizerausgang
  for k=1:length(info) %Bestimmumg von I(k)=summe [c(j)*v(k-j)] j=-K...K
      v1=fliplr(data(1,k:(k+length(copt)-1))); %Sequenz v(k-j)
      I1(k)=copt.'*v1.';
  end


    %Bitentscheideung
    for i=1:length(I1)
     if real(I1(i))>=0
        bit(i)=1;
          if(bit(i)~=info(i))
             err1=err1+1;
          end
     else
        bit(i)=-1;
          if(bit(i)~=info(i))
             err1=err1+1;
          end
     end
    end

fehler1(h)=err1/N1;

aktSNRMSE=SN(h)
BitfehlerMSE=fehler1(h)

end

SNRMSE=SN
BitfehlerMSE=fehler1

%Frequenzantwort des Kanals
[G_T,W]=freqz(f,1);
```

```
G_t=20*log10(abs(G_T)/max(abs(G_T)));

% Ausgabe
figure(2)
subplot(2,2,3)
stem(real(eq))
title('Channel-overall-response (MSE)')
subplot(2,2,2)
plot(W,G_t)
axis([0 pi -30 0])
ylabel('Amplitude (dB)')
xlabel('Frequency \omega')
title('Amplitudenspektrum')
subplot(2,2,1)
stem(real(f))
title('ISI-Kanal')
subplot(2,2,4)
semilogy(SN1,fehler,SN,fehler1)
axis([0 35 0.0001 0.5])
xlabel('SN')
ylabel('Bit-Fehlerwahrscheinlichkeit')
legend('DFE','MSE')
set(2,'Name','MSE')
```

B C-Code

B.1 mse.c

```
/*****************************************************************************

   mse.c

   05.09.2000


   Diplomarbeit DSP-Programmierung

   Labor für Telekommunikation
   Institut für Hochfrequenz- und Übertragungstechnik
   Fachhochschule Köln

   Dieses Programm soll eine digitale Übertragungsstrecke mit Sendefilter,
   Kanal, Matchedfilter und MSE-Equalizer simmulieren. Zuerst werden die
   Koeffizienten der einzelnen Filter eingelesen. Je nach Interrupt (Taster)
   werden verschiedenen Filter dargestellt.
   Den Eingang der Strecke bildet eine zufällig erzeugte Bitfolge aus "1" bzw.
   "-1". Die 8-fach Überabtastung wird realisiert durch Anhängen von jeweils
   sieben Nullen an jedes Bit.

   Interrupt:
      1       Signal nach Sendefilter (RRC)                U2
              Signal nach Sendefilter+Kanal                U3
      2,3,4   Signal nach Sendefilter+Kanal+Rauschen (mit
              verschiedenen S/N-Verhältnissen)             U2
              Signal nach  Matchedfilter                   U3
      5       Signal nach Abtaster                         U2
              Signal nach Equalizer                        U3

              erzeugte Datensequenz (ständig)              U1



*****************************************************************************/

#include <def21060.h>
#include <21060.h>
#include <signal.h>
#include <macros.h>
#include <math.h>
#include <filters.h>
#include <complex.h>
#include "dac.h"


// Wert für die Überabtastung
#define ZA 8
// Ausgangspegel
#define PEGEL 15000
#define TAPS 81 //verschiedene Filterlängen
#define TAPS1 96
#define TAPS2 23
int os=0,i,j;  // Schleifenzähler für die Überabtstung
int level;     // Pegel

// Filter Ein- und Ausgangsvariablen
complex_float input,output,input1,output1,input2,output2;

// Arrays für Filtereingänge
float dm states_re[TAPS+1],states1_re[TAPS1+1],states_im[TAPS+1],states1_im[TAPS1+1];
float dm states2_re[TAPS1+1],states3_re[TAPS2+1],states2_im[TAPS1+1],states3_im[TAPS2+1];
float dm states4_re[TAPS2+1],states4_im[TAPS2+1];

complex_float rauschen,temp;// Verrauschtes Signal
int snrdb;          // Rauschabstand
```

```
int r_count=0;      // Interruptzähler
int snrdb=100;      // Rauschabstand in dB
float en;           // Energie des Pulses zur Berechnung des Rauschens

// Variablen für gasdev
static int iset=0;
static float gset;

// Koeffizienten Sendefilter (real)
float pm filterRE[TAPS]=
{
   #include "sende2.h"
};

// Koeffizienten Sendefilter (imag.)
float pm filterIM[TAPS]=
{
   #include "sende2im.h"
};


// Koeffizienten Sendefilter+Kanal (real)
float pm filter1RE[TAPS1]=
{
   #include "sekan2re.h"
};


// Koeffizienten Sendefilter+Kanal (imag.)
float pm filter1IM[TAPS1]=
{
   #include "sekan2im.h"
};

// Koeffizienten Matchedfilter (real)
float pm filter2RE[TAPS1]=
{
   #include "mf2re.h"
};

// Koeffizienten Matchedfilter (imag.)
float pm filter2IM[TAPS1]=
{
   #include "mf2im.h"
};

// Koeffizienten Ersatzfilter (real)
float pm filter3RE[TAPS2]=
{
   #include "kanal2re.h"
};


// Koeffizienten Ersatzfilter (imag.)
float pm filter3IM[TAPS2]=
{
   #include "kanal2im.h"
};

// Koeffizienten Equalizer (real)
float pm filter4RE[TAPS2]=
{
   #include "mse2re.h"
};


// Koeffizienten Equalizer (imag.)
float pm filter4IM[TAPS2]=
{
   #include "mse2im.h"
};
```

```
// Prototypen
void init_21k(void);
void timer_hi_prior(int);
void press_button(int);
void timer_16speed(int);
float ran1(void);
void init_filter(void);
float gasdev(void);
int awgn(int,int,int);
complex_float fir1(complex_float,float pm [],float pm [],float dm [],float dm [],int);


// Hauptprogramm
void main (void)
{
    input.im=0;

    // Initialisienung des Filters
    init_filter();
    // Initialisierung einiger SHARC-Register
    init_21k();

    // Timer wird eingeschaltet
    timer_on();

    // Endlosschleife
    for(;;)
    {

    };
}

void init_21k( void )
{
    // Timer Auschalten und Timer Initialisieren
    timer_off();
    timer_set(100,100);


    // Einige Assembler-Befehle zur Initialisierung von Interrupts
    asm( "#include <def21060.h>" );
    asm( "bit set mode2 IRQ1E;");
    asm( "bit clr mode1 NESTM;"  );


    // Timer-Interrupt wird initialisiert
    // Funktion: Wenn der Timer bis 0 runtergezählt hat,
    // wird die Funktion timer_hi_prior() aufgerufen.
    interruptf( SIG_TMZ0, timer_hi_prior );

    // Hardware-Interrupt IRQ1 wird initialisiert
    interruptf(SIG_IRQ1,press_button);

        // Lampen ausschalten
    set_flag(SET_FLAG0, CLR_FLAG);
    set_flag(SET_FLAG1, CLR_FLAG);
    set_flag(SET_FLAG2, CLR_FLAG);
    set_flag(SET_FLAG3, CLR_FLAG);

    r_count=1;
    snrdb=100;

    return;
}

// Interruptfunktion für IRQ1
// Mit dem Interrupt wird der Rauschabstand ausgewählt.
void press_button(int sig_num)
{
   sig_num=sig_num;
```

```
        r_count++;
            if(r_count==2)
            {
            snrdb=100;
            }
            if(r_count==3)
            {
            snrdb=23;
220         }
            if(r_count==4)
            {
            snrdb=15;
            }
            if(r_count==5)
            {
            r_count=0;
            }

230 }

    //Erzeugung einer Zufallszahl und abhängig davon eine 1 bzw. -1
    void timer_hi_prior(int sig_num)
    {
        int aus=0;
        sig_num=sig_num;
        // os = OverSampling
        if(os==0)
        {
240       // Takt = high
          SetIOP(dac_7,dac_high);

         // Die Zufallszahl wird erzeugt.
           if(rand()<1073741824)
           {
               level=-PEGEL;
           }
           else
           {
250           level=PEGEL;
           }

        }

        // ZA-1 mal werden Nullen ereugt (Überabtastung)
        else
        {
          if(r_count==1||r_count==2||r_count==3||r_count==4)
          {
260         level=dac_low;
            // Takt = low
            SetIOP(dac_7,dac_low);
          }
        }

        // Werte werden auf das Filter gegeben.
        input.re=(float)level;
        SetIOP(dac_2,(int)input.re+dac_zero);

270         if(r_count==1) // Sendefilter u. Sendefilter+Kanal
            {
                    output=fir1(input,filter_re,filter_im,states_re,states_im,TAPS);
                    output1=fir1(input,filter1_re,filter1_im,states1_re,states1_im,TAPS1);
                    SetIOP(dac_3,(int)output.re+dac_zero);
                    SetIOP(dac_4,(int)output1.re+dac_zero);
            }
            else //Sendefilter+Kanal+Rauschen u. Matchedfilter
            if(r_count==2||r_count==3||r_count==4)
            {
280
                    output=fir1(input,filter1_re,filter1_im,states1_re,states1_im,TAPS1);
                    input1.re=(float)(int)output.re+awgn(en,PEGEL,snrdb);
```

```
                input1.im=(float)(int)output.im+awgn(en,PEGEL,snrdb);
                aus=(int)input1.re;
                output1=fir1(input1,filter2_re,filter2_im,states2_re,states2_im,TAPS1);
                SetIOP(dac_3,aus+dac_zero);
                SetIOP(dac_4,(int)output1.re/7+dac_zero);
        }
        else // Ersatzfiltermodell u. Equalizer
        {
                output=fir1(input,filter3_re,filter3_im,states3_re,states3_im,TAPS2);
                output1=fir1(output,filter4_re,filter4_im,states4_re,states4_im,TAPS2);
                SetIOP(dac_3,(int)output.re/4+dac_zero);
                SetIOP(dac_4,(int)output1.re/9.5+dac_zero);
        }


        // Zählerverwaltung
        os++;
        if(os==ZA||r_count==0)
        {
           os=0;
        }


}


// Gleichverteiles Rauschen zwischen 0 und 1
float ran1(void)
{
   // gleichverteilte Zufallszahlen zwischen 0 und 1 zu erhalten,
   // wird die C-Zufallsfunktion auf 1 normiert
   return (float)rand()/2147483647;
}



// Initialisierung der Filter
// Einlesen der Koeffizienten
void init_filter(void)
{
    int i=0;
    en=0;


        for(i=0;i<TAPS+1;i++)
        {
           states_re[i]=0.0;
           states_im[i]=0.0;
        }


        for(i=0;i<TAPS1;i++)
        {
           // Energie des Sendeimpuls berechnen
           en+=pow(sqrt(pow(filter1RE[i],2)+pow(filter1IM[i],2)),2);
        }

        for(i=0;i<TAPS1+1;i++)
        {
            states1_re[i]=0.0;
            states1_im[i]=0.0;
            states2_re[i]=0.0;
            states2_im[i]=0.0;
        }


        for(i=0;i<TAPS2+1;i++)
        {
           states3_re[i]=0.0;
           states3_im[i]=0.0;
           states4_re[i]=0.0;
           states4_im[i]=0.0;
```

```
        }

    }

    // Erzeugung von normalverteilten, mittelwertfreien Zufallszahlen mit Varianz 1
360 // aus 'Numerical Recipes in C'
    float gasdev(void)
    {
       float fac,rsq,v1,v2;

       if(iset==0)
       {
          do
          {
             v1=2.0*ran1()-1.0;
370          v2=2.0*ran1()-1.0;
             rsq=v1*v1+v2*v2;
          }while(rsq>=1.0 || rsq==0.0);

          fac=sqrt(-2.0*log(rsq)/rsq);
          gset=v1*fac;
          iset=1;
          return (float)(v2*fac);
       }
       else
380    {
             iset=0;
             return (float)gset;
       }
    }

    // Erzeugung des Rauschsignals, die Varianz wird angepaßt.
    // Die Signalenergie, der Pegel des Eingangssignals und das
    // der Rauschabstand in dB wird eingelesen.
    int awgn(int energy, int scale, int snrdB)
390 {
       int sigma;
       scale=abs(scale); // Betrag vom Pegel, es gibt auch negative Pegel
       // Berechnung der Streuung
       sigma=(int)(energy*scale/sqrt(2*pow(10,snrdB/10)));
       // Berechnung und Rückgabe der Zufallszahlen
       return (int)(gasdev()*sigma);
    }


400 // Komplexe Filterfunktion
    complex_float fir1(complex_float input,float pm coeffs_re[],float pm coeffs_im[],
    float dm state_re[],float dm state_im[], int taps)
    {
    temp.re=0.0;
    temp.im=0.0;

    state_re[0]=input.re;
    state_im[0]=input.im;

410     for(i=0;i<taps;i++)
        {
          temp.re+=(coeffs_re[i]*state_re[i])-(coeffs_im[i]*state_im[i]);
          temp.im+=coeffs_im[i]*state_re[i]+coeffs_re[i]*state_im[i];
        }

        for(j=taps-1;j>=0;j--)
        {
          state_re[j+1]=state_re[j];
          state_im[j+1]=state_im[j];
420     }
    return temp;
    }
```

B.2 dfe.c

```
/*****************************************************************************

   dfe.c


   05.09.2000


   Diplomarbeit DSP-Programmierung

   Labor für Telekommunikation
   Institut für Hochfrequenz- und Übertragungstechnik
   Fachhochschule Köln

   Dieses Programm soll eine digitale Übertragungsstrecke mit Sendefilter,
   Kanal und DFE-Equalizer simmulieren. Zuerst werden die
   Koeffizienten der einzelnen Filter eingelesen. Je nach Interrupt (Taster)
   werden verschiedenen Filter dargestellt.
   Den Eingang der Strecke bildet eine zufällig erzeugte Bitfolge aus "1" bzw.
   "-1". Die 8-fach Überabtastung wird realisiert durch Anhängen von jeweils
   sieben Nullen an jedes Bit.

   Interrupt:
      1,2,3   Signal nach Sendefilter (RRC)                U2
              Signal nach Sendefilter+Kanal+Rauschen
              mit verschiedenen S/N-Werten                 U3
      4       Signal nach Abtaster                         U2
              Signal nach FF-Filter                        U3
      5       Signal nach FF-Filter                        U2
              Signal nach FB-Filter                        U3
              Addition der beiden Anteile                  U4
              Signal nach Entscheider                      U5

              erzeugte Datensequenz                        U1

*****************************************************************************/

#include <def21060.h>
#include <21060.h>
#include <signal.h>
#include <macros.h>
#include <math.h>
#include <filters.h>
#include <complex.h>
#include "dac.h"


// Wert für die Überabtastung
#define ZA 8
// Ausgangspegel
#define PEGEL 15000
#define TAPS 81
#define TAPS1 112
#define TAPS2 27
#define TAPS3 27
#define TAPS4 26

int os=0,i,j; // Schleifenzähler für die Überabtstung
int level;    // Pegel
// Filter Ein- und Ausgangsvariablen
complex_float input,output,input1,output1,input2,output2;

// Arrays für Filtereingänge
float dm states_re[TAPS+1],states1_re[TAPS1+1],states_im[TAPS+1],states1_im[TAPS1+1];
float dm states2_re[TAPS2+1],states3_re[TAPS3+1],states2_im[TAPS2+1],states3_im[TAPS3+1];
float dm states4_re[TAPS4+1],states4_im[TAPS4+1];

complex_float rauschen,temp;// Verrauschtes Signal
int snrdb;                  // Rauschabstand
int r_count=0;              // Zähler für Rauschen
```

```
int snrdb=100;              // Rauschabstand in dB
float en,input3;            // Energie des Pulses zur Berechnung des Rauschens

// Variablen für gasdev
static int iset=0;
static float gset;

// Koeffizienten Sendefilter (real)
float pm filterRE[TAPS]=
{
   #include "sende2.h"
};

// Koeffizienten Sendefilter (imag)
float pm filterIM[TAPS]=
{
   #include "sende2im.h"
};

// Koeffizienten Sendefilter+Kanal (real)
float pm filter1RE[TAPS1]=
{
   #include "sekan4re.h"
};

// Koeffizienten Sendefilter+Kanal (imag)
float pm filter1IM[TAPS1]=
{
   #include "sekan4im.h"
};

// Koeffizienten Ersatzfilter (real)
float pm filter2RE[TAPS2]=
{
   #include "kanal4re.h"
};

// Koeffizienten Ersatzfilter (imag)
float pm filter2IM[TAPS2]=
{
   #include "kanal4im.h"
};

// Koeffizienten FF-filter (real)
float pm filter3RE[TAPS3]=
{
   #include "DFEff4re.h"
};

// Koeffizienten FF-filter (imag)
float pm filter3IM[TAPS3]=
{
   #include "DFEff4im.h"
};

// Koeffizienten FB-filter (real)
float pm filter4RE[TAPS4]=
{
   #include "DFEfb4re.h"
};


// Koeffizienten FB-filter (imag)
float pm filter4IM[TAPS4]=
{
   #include "DFEfb4im.h"
};


// Prototypen
```

```
    void init_21k(void);
    void timer_hi_prior(int);
    void press_button(int);
    void timer_16speed(int);
    float ran1(void);
    void init_filter(void);
    float gasdev(void);
150 int awgn(int,int,int);
    complex_float fir1(complex_float,float pm [],float pm [],float dm [],float dm [],int);


    // Hauptprogramm
    void main (void)
    {
        input.im=0;
        output2.re=0;
        output2.im=0;
160     input2.im=0;

        // Initialisienung des Filters
        init_filter();
        // Initialisierung einiger SHARC-Register
        init_21k();

        // Timer wird eingeschaltet
        timer_on();

170     // Endlosschleife
        for(;;)
        {

        };
    }

    void init_21k( void )
    {
        // Timer Auschalten und Timer Initialisieren
180     timer_off();
        timer_set(100,100);


        // Einige Assembler-Befehle zur Initialisierung von Interrupts
        asm( "#include <def21060.h>" );
        asm( "bit set mode2 IRQ1E;");
        asm( "bit clr mode1 NESTM;"  );


190     // Timer-Interrupt wird initialisiert
        // Funktion: Wenn der Timer bis 0 runtergezählt hat,
        // wird die Funktion timer_hi_prior() aufgerufen.
        interruptf( SIG_TMZ0, timer_hi_prior );

        // Hardware-Interrupt IRQ1 wird initialisiert
        interruptf(SIG_IRQ1,press_button);

        // Lampen ausschalten
        set_flag(SET_FLAG0, CLR_FLAG);
200     set_flag(SET_FLAG1, CLR_FLAG);
        set_flag(SET_FLAG2, CLR_FLAG);
        set_flag(SET_FLAG3, CLR_FLAG);

        r_count=1;
        snrdb=100;

        return;
    }

210 // Interruptfunktion für IRQ1
    // Mit dem Interrupt wird der Rauschabstand ausgewählt.
    void press_button(int sig_num)
    {
       sig_num=sig_num;
```

```
       output2.re=0.0;
       output2.im=0.0;

        r_count++;
            if(r_count==2)
            {
            snrdb=23;
            }
            if(r_count==3)
            {
            snrdb=15;
            }
            if(r_count==4)
            {
            snrdb=100;

            }
            if(r_count==5)
            {

            r_count=0;
            }

   }

//Erzeugung einer Zufallszahl und abhängig davon eine 1 bzw. -1
   void timer_hi_prior(int sig_num)
   {
       int aus=0;
       sig_num=sig_num;
       // os = OverSampling
       if(os==0)
       {
        // Takt = high
        SetIOP(dac_7,dac_high);

        // Die Zufallszahl wird erzeugt.
          if(rand()<1073741824)
          {
           level=-PEGEL;
          }
          else
          {
           level=PEGEL;
          }

       }

           // ZA-1 mal werden Nullen ereugt (Überabtastung)
       else
       {
                if(r_count==1||r_count==2||r_count==3)
                {
                  level=dac_low;
                  // Takt = low
                  SetIOP(dac_7,dac_low);
                }
       }

       // Werte werden auf das Filter gegeben.
       input.re=(float)level;
       SetIOP(dac_2,(int)input.re+dac_zero);

       // Sendefilter u. Sendefilter+Kanal+Rauschen
           if(r_count==1||r_count==2||r_count==3)
           {

                 output=fir1(input,filterRE,filterIM,states_re,states_im,TAPS);
                 output1=fir1(input,filter1RE,filter1IM,states1_re,states1_im,TAPS1);
                 input1.re=(float)(int)output1.re+awgn(en,PEGEL,snrdb);
                 input1.im=(float)(int)output1.im+awgn(en,PEGEL,snrdb);
                 aus=(int)input1.re;
```

```
                SetIOP(dac_3,(int)output.re+dac_zero);
                SetIOP(dac_4,aus+dac_zero);
            }
290         else // Ersatzfilter u. FF-Filter
            if(r_count==4)
            {

              output=fir1(input,filter2RE,filter2IM,states2_re,states2_im,TAPS2);
              output1=fir1(output,filter3RE,filter3IM,states3_re,states3_im,TAPS3);
              SetIOP(dac_3,(int)output.re+dac_zero);
              SetIOP(dac_4,(int)output1.re+dac_zero);
            }
            else // FF-Filter, FB-Filter, nach Addition u. nach Detector
300
            {

              output=fir1(input,filter2RE,filter2IM,states2_re,states2_im,TAPS2);
              output3=fir1(output,filter3RE,filter3IM,states3_re,states3_im,TAPS3);
              //Nach FF-Filter
              SetIOP(dac_3,(int)output3.re+dac_zero);
              //Nach FB-Filter
              SetIOP(dac_4,(int)output2.re+dac_zero);
              input3=output3.re+output2.re;
310           //Nach Addition
              SetIOP(dac_5,(int)input3+dac_zero);

                if(input3>0)
                {
                    input2.re=(float)PEGEL;
                }
                else
                {
                    input2.re=-(float)PEGEL;
320             }

               //Nach Detector
               SetIOP(dac_6,(int)input2.re+dac_zero);
               output2=fir1(input2,filter4RE,filter4IM,states4_re,states4_im,TAPS4);
       }


            // Zählerverwaltung
            os++;
330         if(os==ZA||r_count==0||r_count==4)
            {
               os=0;
            }


    }


    // Gleichverteiles Rauschen zwischen 0 und 1
340 float ran1(void)
    {
       // gleichverteilte Zufallszahlen zwischen 0 und 1 zu erhalten,
       // wird die C-Zufallsfunktion auf 1 normiert
       return (float)rand()/2147483647;
    }



    // Initialisierung der Filter
350 void init_filter(void)
    {
        int i=0;
        en=0;

            for(i=0;i<TAPS+1;i++)
            {
                    states_re[i]=0.0;
                    states_im[i]=0.0;
```

```
            }

            for(i=0;i<TAPS1;i++)
            {

                    // Energie des Sendeimpuls berechnen
                  en+=pow(sqrt(pow(filter1RE[i],2)+pow(filter1IM[i],2)),2);

            }

            for(i=0;i<TAPS1+1;i++)
            {
                    states1_re[i]=0.0;
                    states1_im[i]=0.0;

            }


            for(i=0;i<TAPS2+1;i++)
            {
                    states2_re[i]=0.0;
                    states2_im[i]=0.0;

            }


            for(i=0;i<TAPS3+1;i++)
            {
                    states3_re[i]=0.0;
                    states3_im[i]=0.0;

            }


            for(i=0;i<TAPS4+1;i++)
            {
                    states4_re[i]=0.0;
                    states4_im[i]=0.0;

            }
    }

    // Erzeugung von normalverteilten, mittelwertfreien Zufallszahlen mit Varianz 1
    // aus 'Numerical Recipes in C'
    float gasdev(void)
    {
       float fac,rsq,v1,v2;

       if(iset==0)
       {
          do
          {
             v1=2.0*ran1()-1.0;
             v2=2.0*ran1()-1.0;
             rsq=v1*v1+v2*v2;
          }while(rsq>=1.0 || rsq==0.0);

          fac=sqrt(-2.0*log(rsq)/rsq);
          gset=v1*fac;
          iset=1;
          return (float)(v2*fac);
       }
       else
       {
             iset=0;
             return (float)gset;
       }
    }

    // Erzeugung des Rauschsignals, die Varianz wird angepaßt.
```

```
// Die Signalenergie, der Pegel des Eingangssignals und das
// der Rauschabstand in dB wird eingelesen.
int awgn(int energy, int scale, int snrdB)
{
   int sigma;
   scale=abs(scale); // Betrag vom Pegel, es gibt auch negative Pegel
   // Berechnung der Streuung
   sigma=(int)(energy*scale/sqrt(2*pow(10,snrdB/10)));
   // Berechnung und Rückgabe der Zufallszahlen
   return (int)(gasdev()*sigma);
}


complex_float fir1(complex_float input,float pm coeffs_re[],float pm coeffs_im[],
float dm state_re[],float dm state_im[], int taps)
{
temp.re=0.0;
temp.im=0.0;

state_re[0]=input.re;
state_im[0]=input.im;

    for(i=0;i<taps;i++)
    {
        temp.re+=(coeffs_re[i]*state_re[i])-(coeffs_im[i]*state_im[i]);
        temp.im+=coeffs_im[i]*state_re[i]+coeffs_re[i]*state_im[i];
    }

    for(j=taps-1;j>=0;j--)
    {
        state_re[j+1]=state_re[j];
        state_im[j+1]=state_im[j];
    }
return temp;
}
```

C Layout AD-Wandler

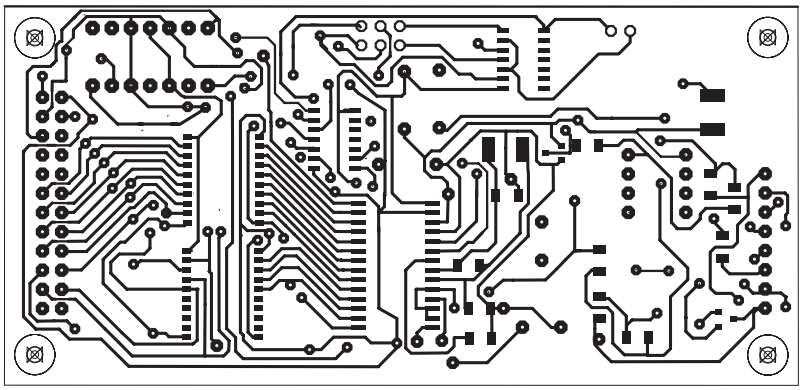


Abbildung C.8: *Layout Oberseite*

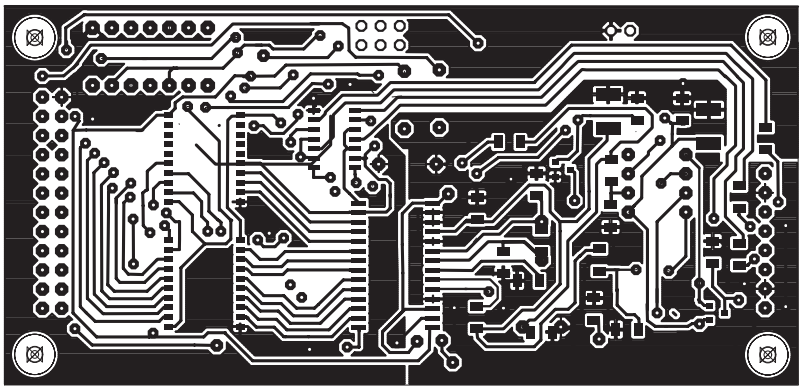


Abbildung C.9: *Layout Unterseite*

C.1 Bestückung

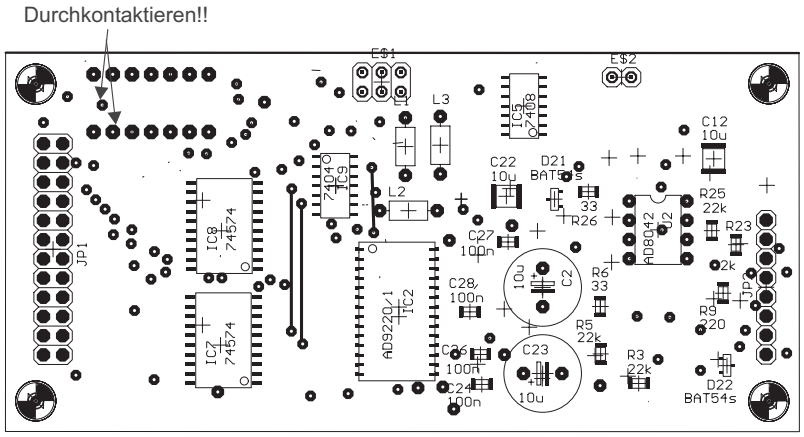


Abbildung C.10: *Bestückung Oberseite*

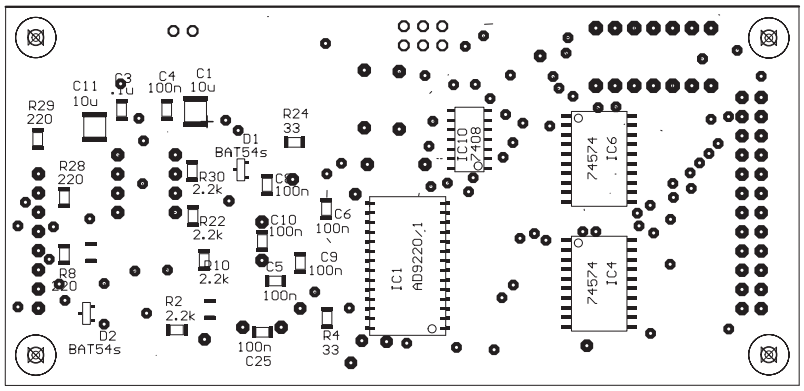


Abbildung C.11: *Bestückung Unterseite*

D Praktikumsanleitung

Nachfolgend ist die Anleitung für das Praktikum abgedruckt.