Diplomarbeit

Entwicklung einer plattformunabhängigen echtzeitfähigen COFDM-Modulator-Software

Universität Hannover
Institut für Allgemeine Nachrichtentechnik

Robert Bosch Multimedia-Systeme GmbH & Co. KG
Entwicklung Multimedia-Systeme EMS–3

Verfasser:	Heiko Penschuck
Matr.-Nr.:	1440850
Tag der Ausgabe:	05.11.97
Tag der Abgabe:	05.05.98
Erstprüfer:	Prof. Dr.-Ing. H.-P. Kuchenbecker
Zweitprüfer:	Prof. Dr.-Ing. C.-E. Liedtke
Betreuer:	Dipl.-Ing. M. Schrader

Diplomarbeitsthema

Entwicklung einer plattformunabhängigen, echtzeitfähigen COFDM-Modulator-Software

Problemstellung:
Das europäische digitale terrestrische Hörfunksystem (DAB) benutzt das Multiträger-Modulationsverfahren COFDM (coded orthogonal frequency division multiplex). Die COFDM-Modulation findet dabei direkt am Senderstandort von sogenannten COFDM-Modulatoren statt. Allerdings ermöglichen neue Technologien mit immer höheren Integrationsgraden mittlerweile wesentlich kompaktere Konzepte für die Implementierung von COFDM-Modulatoren. Die Ergebnisse der Diplomarbeit sollen als Basis für ein zukunftssicheres Konzept für eine neue Generation von COFDM-Modulatoren dienen:

Zielsetzung:
Es soll eine Software für einen COFDM-Modulator entwickelt werden. Bei der Strukturierung der Software soll darauf geachtet werden, daß sie allgemeinen Ansprüchen von Signalprozessoren wie begrenztem Speicherplatz und Rechenzeitoptimierung genügen kann. Das bedeutet, es muß eine Struktur gefunden werden, die hierarchisch so angelegt ist, daß durch Eingriffe in die Kodierung (prozessorspezifischer Kode, Assembler) an wenigen Stellen hoher Rechenzeitgewinn erzielt werden kann. Außerdem müssen Algorithmen gefunden werden, die aufgrund ihrer mathematischen Eigenschaften bereits günstige Voraussetzungen für eine effiziente Programmierung mitbringen.

Aufgabenstellung:

- Unter UNIX ist ein Softwaredesign zu erstellen, das von der Zielhardware und dem Betriebssystem unabhängig ist. Dabei ist als besonderer Aspekt die Testbarkeit des Systems zu beachten.
- Es sin bezüglich der Rechenzeit optimierte Algorithmen zu entwickeln.
- Das Design ist in einer Programmiersprache (vorzugsweise C,C++) auf einem UNIX-System zu implementieren.
- Die Portierbarkeit ist auf einem anderen Zielsystem nachzuweisen (vorzugsweise Signalprozessor)
- Anhand von Performance-Untersuchungen ist festzustellen, inwieweit das System Echtzeitanforderungen genügt. Es sin Vorschläge für eine Verbesserung der Performance zu erarbeiten.

Hiermit erkläre ich, daß ich diese Arbeit selbständig angefertigt und keine Hilfsmittel als die angegebenen benutzt habe.

Hannover, den 5. Mai 1998

Inhaltsverzeichnis

Abkürzungsverzeichnis

BCH	Fehlerkorrigierender Code nach seinen Entdeckern Bose, Chaudhuri, Hocquenghem
CIF	Common Interleaved Frame: Eine Rahmenstruktur innerhalb des COFDM- Enkoders.
COFDM	"coded orthogonal frequency division multiplex": das in DAB verwendete Modulationsverfahren
CRC	"cyclic redundancy check": Fehlererkennungsverfahren, das auf zyklischen Codes basiert
DFT	diskrete Fouriertransformation
DSP	Digitaler Signalprozessor: Microprozessor, der auf hohe mathematische Verarbeitungsgeschwindigkeit optimiert ist
ETI	Ensemble Transport Interface: Eine Schnittstellenbeschreibung für die Datenübertragung zwischen Serviceprovidern und dem Senderstandort.
FFT	"fast fourier transformation": schnelle Fouriertransformation
FI	Fraktional Integer: Datenformat zur Darstellung von gebrochenen Zahlen mittels Integerzahlen.
FIC	Fast Information Channel: Übertragungskanal innerhalb des DAB-Multiplex, für den eine besondere Kanalkodierung angewendet wird.
FIFO	"first in, first out", hier: Dateityp des Betriebssystems
GNU	"Gnu's Not Unix": Name eines US-amerikanischen Projektes, das ein kostenfreies, UNIX-artiges System entwickelt
GF(q)	Galois Feld mit q Elementen
MSC	"main service channel": Datenkanal des ETI-Protokolls
MST	"main stream": Datenkanal im DAB-Rahmen
PRBS	"pseudo random binary sequence": pseudozufällige Bitfolge mit fester Periode
Q15	Zahlendarstellung im Fraktional-Integer-Format mit 15 Bit plus einem Vorzeichenbit
QPSK	"quad phase shift keying": Modulation der einzelnen Träger
RS-Code	Reed-Solomon-Code: Fehlerkorrekturverfahren, das auf der Mathematik in finiten Feldern basiert
TFPR	"transmission frame phase referenz": das zweite Symbol im DAB-Rahmen, welches die Feinsynchronisation ermöglicht.
TII	"transmitter identifikation information": Symbol, das anstelle des Nullsymbols gesendet wird und den Sender identifiziert.

1 Einleitung

1.1 Motivation

Im Rahmen der Entwicklung eines Senders nach der Eurekaspezifikation zu digitalem Radio, DAB, bei der Robert Bosch Multimedia Systeme Gmbh&CoKg sollte der bestehende COFDM-Encoder überarbeitet werden. Die Gründe dafür sind eine weitere Verkleinerung des Sendersystems sowie neue Verfahren bei der digitalen Vorverzerrung des Signals.

Ziele der erneuten Entwicklung einer Software sind:

1. die Verifikation der bestehenden Systeme und eine Verbesserung hinsichtlich der Wartbarkeit,
2. eine Grundlage für die Entwicklung einer zukünftigen neuen Hardwaregeneration des COFDM-Modulators zu schaffen,
3. Testumgebung hinsichtlich neuer Algorithmen zu erhalten,
4. eine Testumgebung hinsichtlich der Tauglichkeit einer möglichen Hardware für den COFDM-Modulator zu bekommen.

1.2 Ausgangspunkt der Weiterentwicklung

Die Arbeit fußt auf bereits funktionsfähigen Lösungen für einen COFDM-Encoder. Diese sind jedoch alle unter anderen Gesichtspunkten entwickelt worden. Ziel der erneuten Aufgabenstellung soll es sein, einen Ausgangspunkt für weitere Entwicklungen zu bilden, der den Stand der Technik wiederspiegelt.

Zum einen existiert eine Urversion in der Programmiersprache C. Diese basiert jedoch auf einer vorläufigen Spezifikation des Übertragungsverfahrens. Die Software ist deshalb besonders unter den Gesichtspunkten der Erweiterbarkeit entwickelt worden. Seit der Erstellung der Software sind viele Änderungen an dem Übertragungsstandard vorgenommen worden. Zweck dieser Software war es, den Aufwand des Modulationsverfahrens näher zu bestimmen und eine erste fehlerfreie Umsetzung vorzustellen. Der größte Unterschied zu der neuen Software ist jedoch die Forderung nach Echtzeitfähigkeit und der Anspruch, direkt auf einen beliebigen Prozessor übertragen werden zu können.

Zum anderen exisitiert eine MATLAB-Version des COFDM-Encoders, der weitestgehend den EU-Spezifikationen entspricht. Diese dient im wesentlichen als Ausgangspunkt für die Simulation des gesamten Verfahrens inklusive Übertragungsstrecke und Dekoders innerhalb von Matlab sowie für die Erzeugung von speziellen Testsignalen und Meßreihen. Diese MATLAB-Version ist deutlich langsamer in der Berechnung und ist nicht effizient auf Hardware zu übertragen.

Letztlich existiert ein vollständiger Hardwaremodulator, der auf der Basis von 5 Signalprozessoren arbeitet. Die Software ist für jeden Prozessor mit grossem Aufwand von Hand in Assembler geschrieben und optimiert worden. Dabei wurden vor allem spezielle Hardwareoptionen ausgenutzt und Fehler in den selbigen umgangen. Die Software ist damit hochgradig systemspezifisch und nur sehr schwer zu warten. Eine Übertragung auf andere Hardware ist unter marktwirtschaftlichen Gesichtspunkten nicht sinnvoll.

1.3 Aufbau des Textes

Zunächst soll ein Überblick über das Übertragungsverfahren von DAB gegeben werden. Dabei wird die COFDM-Modulation besonders hervorgehoben. Im Anschluß daran werden theoretische Grundlagen erörtert, die für das Erstellen der Arbeit herangezogen worden sind. Danach wird eine kurze Analyse des COFDM-Encoders gegeben, die auch die Basis für die Implementierung darstellt. Im Kapitel über die Implementierung werden einige Details aus der Umsetzung des COFDM-Encoders vorgestellt. Danach werden die Ergebnisse aus der Untersuchung des Systems beschrieben. Als letztes findet sich eine kurze Zusammenfassung der Arbeit.

2 Das Übertragungsverfahren von DAB

Innerhalb des Projekts Eureka 147 wurde ein System spezifiziert, welches die digitale Übertragung von Hörfunk gestattet. Damit kämen die Vorteile der digitalen Übertragung auch beim Rundfunk zum Zug. Wesentliche Merkmale der Entwicklung sind:

- die Übertragung in CD-naher Qualität. Im spezifizierten Verfahren wird dabei auf die sogenannte MPEG2-Kodierung der "Motion Picture Experts Group" zurückgegriffen, die eine variable Qualität bis nahezu CD-Qualität bei unterschiedlicher Bandbreite ermöglicht,
- die Übertragung anderer multimedialer Daten, wie zum Beispiel den Hörfunk begleitende Untertitel oder Bilder, Stauwarnungen und Nachrichten,
- der mobile Empfang wird in der Spezifikation des Hörfunk erstmals definiert und bis 200 km/h ausgelegt,

Auf der Seite der Senderbetreiber lassen sich noch weitere Vorteile ansprechen, wie:

- die Frequenzökonomie. Das verwendete Modulationsverfahren besitzt ein relativ scharf begrenztes Spektrum, dessen Bandbreite durch die Digitaltechnik sehr effizient genutzt wird. Gleichzeitig wird der Betrieb eines Gleichwellennetzes unterstützt, so daß Pufferbereiche zwischen zwei benachbarten Sendern entfallen können.
- ein sparsamer Energieverbrauch. Die Techniken der digitalen Fehlerkorrekturverfahren erlauben eine geringere Signal zu Rauschleistung.

Dieses System, "Digital Audio Broadcasting" oder auch kurz DAB genannt, benutzt ein breitbandiges Spektrum zur Übertragung. Dabei werden neben der Modulationsfrequenz eine Vielzahl an Unterträgern moduliert. Das Signal wird digital im Frequenzbereich zusammengesetzt und dann mittels einer Fouriertransformation in den Zeitbereich transformiert.

Die Kapazität von DAB reicht für mehr als ein Audioprogramm. Deshalb werden in einem Multiplex mehrere Hörfunkprogramme parallel übertragen. Die Qualität jedes einzelnen Programms ist dabei individuell bestimmbar. Sie ist, wie auch die Zusammensetzung des gesamten Programmixes, dynamisch veränderbar.

2.1 Einbettung des COFDM-Encoders in DAB

Die Audio- und Bilddaten werden vom Tonstudio eines Senders zu einem Service zusammengesetzt. Da mehrere davon parallel über ein DAB-Spektrum ausgestrahlt werden, werden diese Daten zunächst bei einem Ensembleprovider gesammelt. Von dort aus werden sie an die einzelnen Sendestationen übermittelt. Die Daten werden

broken Image

Abbildung 1: Allgemeine Struktur für ein Gleichfrequenznetzwerk

erst hier kanalkodiert, da dies die Datenrate erheblich vergrößert, was sonst auch Service- und Ensembletransportnetzwerk belasten würde. Die in dieser Arbeit vorgestellte Lösung für den COFDM-Enkoder beinhaltet auch die Anpassung an ein spezifiziertes Ensembletransportnetzwerk. Dazu gehören CRC's und RS-Code.

2.2 Aufbau des COFDM-Encoders

Das COFDM-Signal ist von einer Rahmenstruktur geprägt. Dabei werden zunächst ein Nullsymbol zur Grobsynchronisation und anschließend ein Phasenreferenzsymbol zur Feinsynchronisation versendet. Danach folgt eine definierte Zahl an Datensymbolen. Der realisierte COFDM-Encoder erzeugt dabei das zeitdiskrete komplexe Basisbandsignal. Der genaue Aufbau des Signals ist in [1] spezifiziert.

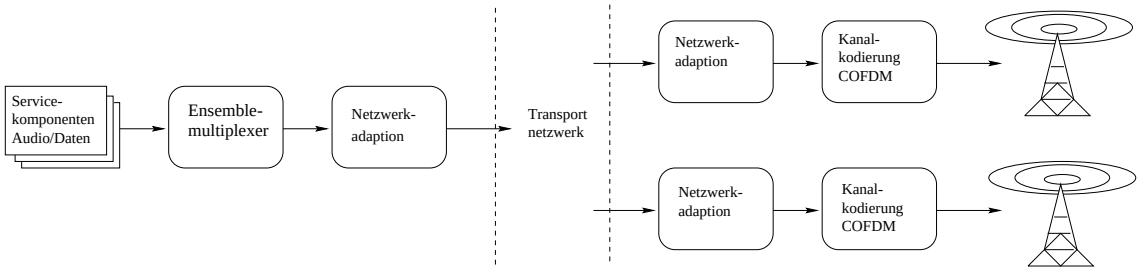

Abbildung 2: Blockschaltbild des COFDM-Encoders

2.2.1 PRBS

Die Daten der einzelnen Kanäle werden vor der Kanalkodierung zu einer pseudozufälligen Bitfolge addiert. Dadurch sollen Periodizitäten der Eingangsfolge reduziert werden. Das resultierende Signal besitzt eine gleichmäßigere Energieverteilung über das Spektrum.

2.2.2 Kanalkodierung

Die Kanalkodierung fügt den Daten den für den Transport über einen fehlerbehafteten Kanal notwendigen Fehlerschutz an. Da es sich bei der terrestrischen Übertragung um eine unidirektionale Verbindung handelt, wird dem Signal eine hohe Redundanz hinzugefügt. Die Redundanz wird von einem Faltungskodierer erzeugt. Um die Koderate variieren zu können, werden im Anschluß bestimmte Bits aus dem Datenstrom wieder gelöscht. Diesen Vorgang nennt man Punktieren. Die Koderate reicht bei dem verwendeten Verfahren von $\frac{8}{9}$ bis $\frac{1}{4}$.

2.2.3 Zeitinterleaver

Um den Einfluß von Bündelfehlern auf den Kanal zu mindern, findet eine Kodespreizung der Daten statt. Die Daten werden dabei gleichmäßig auf 16 DAB-Rahmen aufgeteilt. Die Daten müssen dabei über diesen Zeitraum zwischengespeichert werden.

2.2.4 Frequenzinterleaver

Da das Signal bei der Übertragung frequenzselektiven Störungen unterworfen ist, werden zusammenhängende Daten innerhalb eines Symbols über das gesammte Spektrum gestreut. Auch hierdurch soll das Entstehen von Bündelfehlern verhindert werden.

2.2.5 QPSK-Mapper

Die einzelnen Träger des Signals werden im Frequenzbereich moduliert. Dabei wird jedem Träger eine von 4 Phasen zugeordnet.

2.2.6 Differentielle Modulation

Es wird nicht die Phase der Träger direkt, sondern die Differenz zum jeweils vorangegangenen Symbol übertragen. Der differentiell modulierte Träger besitzt nun eine von 8 Phasen. Die maximale Phasendifferenz des Trägers in zwei aufeinanderfolgenden Symbolen ist damit nicht π sondern $\frac{3}{4}\pi$.

2.2.7 OFDM-Generator

Um das Symbol vom Frequenzbereich in den Zeitbereich zu transformieren, wird eine inverse Fouriertransformation angewendet. Anschließend wird das Symbol noch um ein Schutzintervall verlängert.

3 Theoretische Grundlagen

3.1 DFT

Hier soll eine kurze Einführung in die Berechnung der Fouriertransformation mittels eines Computers gegeben werden. Es soll dabei zunächst auf die Punkte eingegangen werden, die für eine korrekte Interpretation notwendig sind. In den folgenden Abschnitten werden Möglichkeiten zur Beschleunigung des Algorithmusses angesprochen. Als letztes kommen Überlegungen zur Genauigkeit der Berechnung bei endlicher Registerbreite hinzu.

Die Berechnung einer diskreten Fouriertransformation setzt zunächst ein paar Vereinbarungen für die Interpretation der Ergebnisse voraus. Die Fouriertransformation ist zunächst definiert als

$$F(x) = \int_{-\infty}^{+\infty} (f(x) \cdot e^{(-j\omega x)}) dx. \tag{1}$$

Dieses Integral läßt sich auf Grund der Grenzen von $-\infty$ bis $+\infty$ nicht numerisch lösen. Zunächst muß eine diskrete Entsprechung für die obige Gleichung gefunden werden. Es läßt sich zeigen, daß einem periodischen Zeitsignal eine Folge von Deltaimpulsen im Frequenzbereich entspricht. Ebenso läßt sich zeigen, daß einem im Frequenzbereich periodischen Signal eine Folge von Deltaimpulsen im Zeitbereich entspricht. Nimmt man beide Beobachtungen zusammen, läßt sich zu einem periodischen diskreten Signal im Zeitbereich ein periodisches diskretes Spektrum zuordnen. Die Berechnung der diskreten Fouriertransformation ist mit

$$X(n) = \frac{1}{T1} \cdot \sum_{k=1}^{N} x(k) \cdot e^{(-j\omega \frac{n \cdot k}{N})}, \tag{2}$$

und ihre Inverse mit

$$x(n) = \frac{1}{T2} \cdot \sum_{k=1}^{N} X(k) \cdot e^{(j\omega \frac{n \cdot k}{N})} \tag{3}$$

gegeben. Bei der Größe der Vorfaktoren $1/T1$ und $1/T2$ herrscht Uneindeutigkeit. Soll die Amplitude der analogen Fouriertransformation angenähert werden, wird üblicherweise $T1$ zu 1 und $T2$ zu N gesetzt. Um jedoch das Parsevalsche Theorem über die Energieerhaltung in Frequenz und Zeitdarstellung einzuhalten, muß $T1 = T2 = \frac{1}{\sqrt{N}}$ gelten. Bei der im COFDM-Modulator verwendeten IFFT geht es insgesamt nur um die relativen Beträge der Samples zueinander, da das Signal später noch verstärkt wird. Wichtig ist nur die Angabe der Maximalamplituden des Ausgangssignals, um die nachfolgenden Stufen korrekt auszusteuern.

In der Praxis geht es meistens darum, mit der diskreten Fouriertransformation ein reales Signal anzunähern. Um vom realen Signal mittels der DFT zum realen Spektrum zu kommen, muß das Eingangssignal diskretisiert, also abgetastet werden.

Eine eindeutige Abbildung ist hierbei nach Shannons Abtasttheorem nur bei bandbegrenzten Signalen möglich. Um die DFT anwenden zu dürfen, muß das erhaltene Signal periodisch angenommen werden. Um mit dem Ergebnis der DFT Aussagen über das reale System machen zu können, behilft man sich mit einem weiteren Trick. Da das reale Signal nicht periodisch ist, nimmt man an, es wäre ein Ausschnitt einer Periode davon. Das entspricht einer Multiplikation der periodischen Funktion mit einem Rechtecksignal. Es läßt sich zeigen, daß eine Multiplikation im Zeitbereich einer Faltung im Frequenzbereich entspricht. Die Fouriertransformierte eines Rechtecks ist die SI-Funktion $sin(x)/x$. Ersetzt man also im Frequenzbereich jeden Delta-Impuls durch eine SI-Funktion, erhält man ein Bild des realen Spektrums.

3.2 FFT-Algorithmus

Die grundlegende Idee dabei ist, die Summe in zunächst zwei Teile zu zerlegen. Das setzt voraus, daß N selber teilbar ist. Allgemein läßt sich eine Fouriertransformation um so schneller berechnen, je hochgradiger die Zahl N teilbar ist. Gehen wir zunächst von einer Teilbarkeit durch zwei aus. Die Summe zerfällt dabei in zwei Teile, einen von 1 bis 2, und einen von 1 bis $N/2$:

$$N = 2 \cdot R;\ R = N/2;\ n = (2 \cdot n_1 + n_0);\ k = (R \cdot k_1 + k_0);$$

$$X(2 \cdot n_1 + n_0) = \sum_{k_1=0}^{1} \sum_{k_0=0}^{N/2} x(2 \cdot k_1 + k_0) \cdot e^{(-j\omega \frac{(2 \cdot n_1 + n_0)(2 \cdot k_1 + k_0)}{N})}.$$

Substituiert man nun

$$W = e^{(-j\omega \frac{1}{N})}$$

und trennt den Term $e^{()}$ erhält man

$$X(2 \cdot n_1 + n_0) = \sum_{k_1=0}^{1} \sum_{k_0=0}^{N/2} x(2 \cdot k_1 + k_0) \cdot W^{n_1 k_0 R} \cdot W^{n_0 k_1 \cdot 2} \cdot W^{n_0 k_0}.$$

Der fehlende Term $W^{2Rn_1k_1}$ nimmt wegen $N = 2R$ immer den Wert 1 an und ist deshalb gleich weggelassen worden. Der Faktor $W^{n_1k_0R}$ nimmt immer nur die Werte 1 oder -1 an.

Diese Summe läßt sich nun in zwei Schritte aufteilen, die nacheinander berechnet werden können. Der Faktor $W^{n_0k_0}$ kann dabei wahlweise dem ersten oder dem zweiten Schritt zugeordnet werden oder als unabhängiger Zwischenschritt ausgeführt werden. Im letzten Fall spricht man vom sogenannten Twiddle-Schritt und von $W^{n_0k_0}$ als sogenanntem Twiddle-Faktor. Es fällt dabei auf, daß die Reihenfolge der Ergebnisse nicht mehr stimmt. Um eine lineare Beziehung wieder herzustellen, müssen die Ergebnisse umgeordnet werden. Ist dieses ein separater Berechnungsschritt, spricht man vom FFT-Reversal. Da die Ergebnisse im Frequenzbereich umgeordnet werden müssen, spricht man auch von “decimation in frequency”-Algorithmen.

Erster Berechnungsschritt:

$$x_1(2 \cdot n_0 + k_0) = \sum_{k_0=0}^{N/2} x(2 \cdot k_1 + k_0) \cdot W^{N/2 \cdot n_0 k_1} \ \cdot \ W^{n_0 k_0}$$

Zweiter Berechnungsschritt:

$$x_2(2 \cdot n_0 + k_0) = \sum_{k_1=0}^{1} x_1(2 \cdot n_0 + k_0) \cdot W^{n_1 k_0 \cdot 2}$$

Reversal:

$$X(2 \cdot n_1 + n_0) = x_2(2 \cdot n_0 + n_1)$$

Nach dem gleichen Prinzip läßt sich die innere Summe erneut unterteilen, wenn auch $N/2$ sich in weitere Faktoren zerlegen läßt. Kann man N als Potenz von 2 mit $N = 2^q$ darstellen, erhalten wir durch fortgesetztes Teilen durch zwei den sogenannten Radix-2 oder Basis-2 Algorithmus, der nach seinen Entdeckern auch Cooley-Tukey-Algorithmus genannt wird. Bei der weiteren Zerlegung wird aus dem Faktor $W^{n_1 k_0 \cdot 2}$ in den anschließend aufeinanderfolgenden Schritten jeweils ein Faktor, der ebenfalls immer nur die Werte 1 und -1 annimmt. Multiplikationen finden bei diesem Algorithmus nur mit den Twiddle-Faktoren statt. Alle Stufen können sequentiell nacheinander berechnet werden. Die Berechnung kann dabei "in-place" geschehen, was bedeutet, daß die jeweils nächste Stufe die Ergebnisse der vorherigen überschreibt. Dadurch wird kein zusätzlicher Speicher für die Berechnung benötigt. Es läßt sich auch zeigen, daß die Reihenfolge der Stufen unter Berücksichtigung anderer Twiddle-Faktoren umgekehrt werden kann. Dies führt zu einem "decimation in frequency"-Algorithmus. Zu beachten ist, daß der Algorithmus nur für Eingangsdaten mit N als Zweierpotenz angewendet werden kann.

Der Radix-2 Algorithmus besitzt $log_2(n)$ Stufen und läuft also in einer Zeit proportional zu $N \cdot log_2(N)$ ab. Der Algorithmus ist hinsichtlich der Additionen optimal. Ein besseres Verfahren zum Aufaddieren von $N \cdot N$ unabhängigen Werten als ein logarithmisches gibt es nicht. Bei der Zahl der Multiplikationen läßt sich jedoch noch einiges verbessern. Für eine 2048 Punkte FFT finden nämlich viele der Multiplikationen mit den Faktoren W^N,$W^{N/2}$ und $W^{N/4}$ mit den entsprechenden Werten 1,-1,j statt. Ebenso lassen sich komplexe Multiplikationen mit $W^{N/8}$ mit nur zwei statt üblicherweise vier reellen Multiplikationen berechnen, da Real- und Imaginärteil von $W^{N/8} = 1/\sqrt{(}2) + j/\sqrt{(}2)$ gleich groß sind.

Nimmt man statt einer Zerlegung der Summe durch zwei eine Zerlegung durch 4 vor, erhält man den Radix-4 Algorithmus. Anstelle der Werte 1 und -1 nehmen hier die Faktoren der Summenstufen die Werte 1,-1,j und $-j$ an. Auch hier finden Multiplikationen nur mit den Twiddle-Faktoren statt. Da hier jedoch nur $log_4(N)$

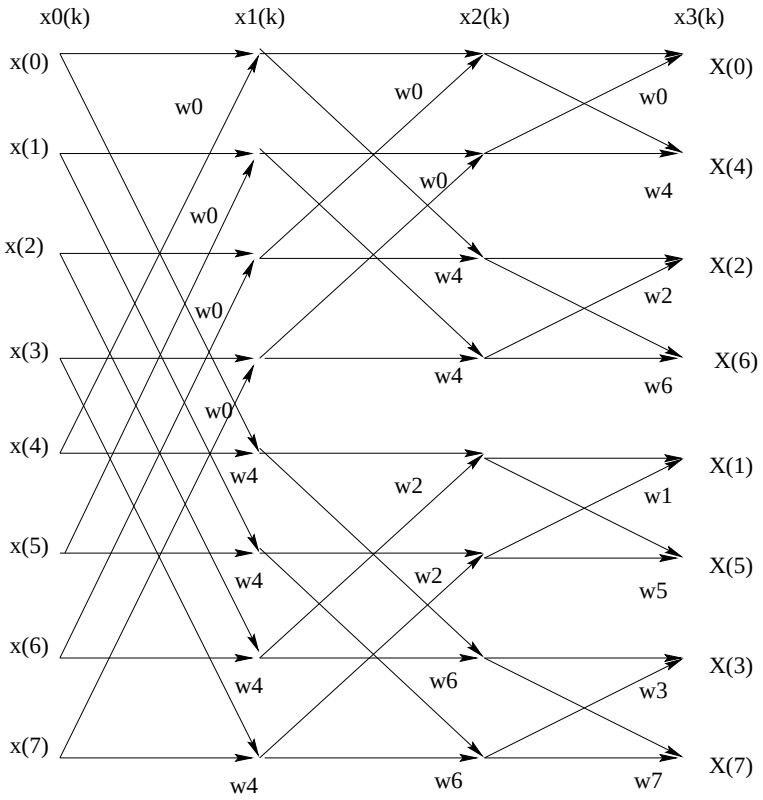

Abbildung 3: Signalflußgraph einer 16 Punkte Radix-2 FFT mit "decimation in frequency".

Stufen benötigt werden, reduziert sich die Zahl der nötigen Multiplikationen theoretisch auf die Hälfte. Da jedoch viele Multiplikationen mit 1 ersetzt werden, entfallen in der Praxis nur etwa 20% der Multiplikationen. Bei einer Implementation eines Radix-4-Algorithmus ist dabei darauf zu achten, daß die innere Summe von $4 \cdot 4$ Werten auch logarithmisch optimiert ausgeführt wird. Ansonsten verschenkt man schnell die Rechengeschwindigkeit, die man durch die Multiplikationen gewonnen hat, wieder mit zusätzlichen Additionen.

Der Radix-4 Algorithmus läßt sich nur bei Eingangsdaten mit der Zahl N als Viererpotenz anwenden. Aufgrund der stufenweisen Berechnung, die voneinander unabhängig durchgeführt werden kann, lassen sich jedoch Mischalgorithmen finden. Bei diesen wird den Basis-4 Stufen eine Basis-2 Stufe vorangestellt. Dadurch läßt bei einer beschleunigten Berechnung derselbe Eingangsraum wie bei dem Radix-2 Algorithmus erschließen.

Prinzipiell läßt sich die Zahl der Multiplikationen durch weiteres Erhöhen der Basis noch weiter reduzieren. Der Gewinn ist jedoch nur marginal und beträgt für einen Radix-16 Algorithmus ungefähr 5% gegenüber Radix-4. Zu einem Ende kommt diese Möglichkeit der Optimierung, wenn die linke und rechte Stufe der FFT gleich groß

sind. Die Multiplikationen, die in der Mitte der beiden Stufen liegen, lassen sich so optimal minimieren. Der Algorithmus würde einem Radix-$sqrt(N)$ Algorithmus entsprechen.

3.3 Berechnung der IFFT mittels FFT

Um nicht die gesamte Optimierungsarbeit doppelt machen zu müssen, wäre es wünschenswert, einen einzigen Algorithmus zu besitzen, der sowohl FFT wie auch die Inverse berechnen kann. Bei genauerer Betrachtung des FFT-Algorithmus fällt leicht auf, unter welchen Bedingungen eine FFT zur Berechnung einer IFFT verwendet werden kann. Betrachtet man die Gleichungen für FFT und IFFT, ist der einzige Unterschied im Produkt von x und dem Drehfaktor W zu sehen. Formulieren wir das Produkt ausführlich:

FFT:

$$x(k) \cdot e^{-\phi} \rightarrow$$
$$x(k) \cdot (cos(-\phi) + jsin(-\phi)) \rightarrow$$
$$(a + jb) \cdot (cos(\phi) - jsin(\phi))$$

IFFT:

$$x(k) \cdot e^{\phi} \rightarrow$$
$$(a + jb) \cdot (cos(\phi) + jsin(\phi))$$

Zunächst fällt auf, daß sich die IFFT mittels einer FFT berechnen läßt, wenn man entweder die Eingangswerte x, oder die Drehfaktoren W komplex konjugiert und das Ergebnis ebenfalls komplex konjugiert. Formuliert man weiter

FFT:

$$Re = acos(\phi) + bsin(\phi)$$
$$Im = bcos(\phi) - asin(\phi)$$

IFFT:

$$Re = acos(\phi) - bsin(\phi)$$
$$Im = asin(\phi) + bcos(\phi)$$

sieht man, daß auch durch einen Tausch von Real- und Imaginärteil der Eingangswerte vor und nach der FFT eine inverse Transformation berechnet wird. Dies läßt sich durch Implementation mit Hilfe eines Zeigers auf Real- und Imaginärteil in zwei Befehlen realisieren. Die IFFT ist im Rahmen des COFDM-Modulators auf diese Weise realisiert worden.

3.4 Definition eines Echtzeitsystems

Eine gute Definition eines Echtzeitsystems findet sich in [8]. Die wesentlichen Gedanken sollen hier zusammengefaßt werden, Ein Echtzeitsystem ist ein System, in dem die korrekte Funktion nicht nur von den Ausgaben, sondern auch von der Zeit, zu der diese Ausgaben erzeugt werden, abhängt. Die Zeit zwischen einer Eingabe und den daraus resultierenden Ausgaben heißt Antwortzeit.

Damit die Umwelt auf ein System reagieren kann, muß es eine Ausgabe erzeugen. Ein Echtzeitsystem erzeugt immer – auch im Fehlerfall – eine definierte Antwort (Reaktivität). Das impliziert, daß das System vorhersagbar ist und formal beschrieben werden kann. Im einfachsten Fall wird einfach eine Zustands- oder Fehlermeldung an den Benutzer ausgegeben, der dann das System daraufhin genauer untersucht.

Ebenfalls garantiert ein Echtzeitsystem eine Antwort innerhalb einer maximalen Zeit auf eine Eingabe hin. Die maximal zulässige Antwortzeit bestimmt die physikalische Umgebung, in der das System betrieben wird. Man unterscheidet hier noch zwischen harten und weichen Echtzeitsystemen. Bei harten Systemen ist die Einhaltung der maximal zulässigen Antwortzeit zwingend vorgeschrieben. Eine Antwort nach dieser Grenze ist unbrauchbar und wertlos. Bei einem harten System, welches diese Grenzen überschreitet, kann dies Gefahr für Menschen oder Material bedeuten. Ein weiches System ist ein System, bei dem die Antwortzeiten nur im zeitlichen Durchschnitt eingehalten werden müssen. Einzelne Überschreitungen können in der Auswirkung abgefedert werden oder sind durch einen Pufferspeicher kurzfristig vom realen System getrennt.

Für ein Echtzeitsystem mit harter Grenze sind zum Beispiel Fahrzeug- oder Flugzeugsteuerungen vorstellbar. Als Beispiel für ein System mit weicher Grenze mag man sich einen Prozeß an einem Fließband vorstellen. Das Fließband kann vor und hinter dem Prozeß ein Zwischenlager für ankommende Teile besitzen. Der eigentliche Prozeß hat also einen Spielraum, innerhalb dessen seine Durchlaufzeit pro Teil schwanken darf. Wie man daran sieht, treten in der Realität meistens harte und weiche Grenzen zusammen auf. In jedem Fall sollte für das System ein Zustand, wie zum Beispiel das Auslösen eines Alarms, vorgesehen werden, mit dem es auf das Überschreiten der zulässigen Antwortzeiten reagiert.

Ebenfalls wird an den Beispielen deutlich, daß die Verarbeitungszeit, die dem Prozeß zur Verfügung steht, eine relative Größe ist. Sie ist einerseits durch die gesetzte Aufgabe begrenzt. Auf der anderen Seite läßt sie sich jedoch durch die eingesetzte Hardware stark beeinflussen. Generell ist dabei immer eine effiziente Lösung gesucht. Diese soll sowohl den größtmöglichen Raum bei der Wahl der zu verwendenden Hardware bieten, als auch die Robustheit bieten, die durch Verwendung eines einfachen, minimalen Systems entsteht.

3.5 Das Fraktional-Integer Format

3.5.1 Darstellung und Auflösung von Zahlenformaten

Komplexe mathematische Formeln lassen sich numerisch auf Computern nur innerhalb eines vorgegebenen Wertebereichs lösen. Das liegt daran, daß nur eine begrenzte Zahl von Stellen im Computer gespeichert werden kann. Aus diesem Grunde lassen sich komplexe mathematische Formeln auch nur mit begrenzter Genauigkeit lösen. In diesem Fall kann nur eine begrenzte Zahl an Nachkommastellen gespeichert werden. Bei der Genauigkeit spricht man auch von Auflösung. Damit wird die Anzahl der numerischen Stellen einer Zahl, die noch korrekt dargestellt sind, bezeichnet. Die Stellen sind in der Digitaltechnik binär. Die Auflösung wird bei Computern deshalb in Bit angegeben. Die in der Nachrichtentechnik verbreitete Angabe von Dezibel läßt sich leicht daraus ableiten. Ein Bit entspricht dem Faktor zwei, daraus folgt $1Bit \hat{=} 20Log(2) = 6.02db$.

In der fortschreitenden Entwicklung von Rechenmaschinen haben sich zwei Darstellungsformen von Zahlen durchgesetzt. Zum einen der Integerdatentyp, der sich durch die einfache Repräsentation in der Hardware auszeichnet, sowie der Fließkommadatentyp, der sich durch größere Fehlertoleranz und Flexibilität in der Anwendung hervorhebt. Beide Datentypen werden von vielen Hochsprachen in verschiedenen Auflösungen unterstützt, die jedoch synonym verwendet werden können. Dadurch sind Optimierungen von Algorithmen hinsichtlich des Speicherbedarfs möglich.

Die Anzahl der Bits, die für die Darstellung von Zahlen verwendet wird, ist prinzipiell frei wählbar. In der Praxis stellt die verwendete Hochsprache dabei Datentypen zur Verfügung, die den Ansprüchen der meisten Algorithmen genügen. Unterschiede gibt es jedoch in der Abarbeitungsgeschwindigkeit von Berechnungen. Grundsätzlich gilt, je mehr Bits für die Darstellung von Zahlen verwendet werden, desto aufwendiger, und damit zeitraubender, ist auch eine Berechnung.

3.5.2 Integerarithmetik

Besonders schnell lassen sich Berechnungen mit ganzzahligen Werten ausführen. In der Hardware ist dabei für jedes Bit an Genauigkeit ein Addiergatter vorhanden. Dadurch kann die Berechnung komplett parallel durchgeführt werden. Da diese Darstellungsform nur einfache Hardwarestrukturen voraussetzt und auch sehr schnelle Berechnungen stattfinden, werden viele Prozessoren ausschließlich mit dieser Technik ausgerüstet. Fließkommaberechnungen können auch auf dieser Hardware algorithmisch implementiert werden, benötigen jedoch einen wesentlich höheren Zeitaufwand für eine mathematische Operation. Um den Geschwindigkeitsvorteil der Festkomma-Arithmetik zu nutzen und dennoch Zahlen mit Nachkommastellen darstellen zu können, existiert das sogenannte Fraktional-Integer-Format. Dabei wird ein Teil des Zahlenstrahls mittels Multiplikation mit einer Konstante auf den Darstellungsbereich einer Festkommazahl abgebildet.

3.5.3 Rechenvorschriften bei Festkomma-Arithmetik

Additionen und Subtraktionen von zwei Fraktional-Integer-Zahlen sind dadurch mit einfacher Integeraddition bzw. -subtraktion darstellbar. Einziges Problem dabei ist ein eventueller Über- oder Unterlauf des darstellbaren Zahlenbereichs. Dieses Problem tritt generell bei jeder Berechnung mit einem Computer auf, jedoch ist der Bereich, innerhalb dessen mathematische Operationen durchgeführt werden dürfen, durch die Transformation stark eingeschränkt. Dies ist auch Kern bei der effizienten Implementierung eines Algorithmus mittels FI-Zahlendarstellung. Der Programmierer muß dabei den Wertebereich, innerhalb dessen sich die darzustellenden Variablen bewegen, genau kennen, und seine Veränderung über den Ablauf des Algorithmus verfolgen. Nur so ist eine Implementation, die sowohl schnell als auch genau ist, möglich.

Problematisch ist vor allen Dingen die Darstellung der Multiplikation zweier FI-Zahlen mittels einer normalen Integermultiplikation. Sinn der Transformation war es ja gerade, den darstellbaren Wertebereich voll auszuschöpfen. Eine Multiplikation bedeutet jedoch, daß der Wertebereich nach der Multiplikation doppelt so groß ist wie vorher. Eine Lösung dieses Problems ist eine Normierung der größten Eingangsdaten auf Eins. Bei den sogenannten Fraktional-Integer-Formaten (Qxx Formaten) wird einfach die größte darstellbare Zahl zu Eins gesetzt. Das bedeutet für Multiplikationen, daß das Ergebnis nie den Wertebereich des modifizierten FI-Datentyps überschreitet. Das Integerergebnis hat zwar nach wie vor einen doppelt so großen Wertebereich, kann aber durch das Verwerfen aller irrelevanten Nachkommastellen wieder auf den ursprünglichen Wertebereich angepaßt werden. Dies geschieht mittels Division oder einer Shift-Operation. Problematisch ist in diesem Verfahren jedoch ein möglicher Überlauf bei Additionsoperationen. Eine grundsätzliche Lösung für die Anpassung eines FI-Algorithmus an Integeroperationen bietet eine dynamische Anpassung des FI-Wertebereiches an den des Integerformates. Dies ist für viele Algorithmen, die stufenweise ablaufen, wie zum Beispiel auch den der schnellen Fouriertransformation, kein Problem. Bei der auf Eins normierten Darstellung muß zum Beispiel nach jeder Addition und Subtraktion das Ergebnis durch zwei geteilt werden, um einen möglichen Überlauf zu verhindern. Dieses Verfahren wird auch Blockgleitkommaarithmetik genannt (siehe auch [7]). Bei der Interpretation der Ergebnisse muß der sich ergebende Vorfaktor, der durch den Algorithmus vorgeschrieben wird, berücksichtigt werden.

3.5.4 Beschreibung des Q15-Formates

Für schnelle Berechnungen mit eingeschränkter Genauigkeit und sparsamem Speicherverbrauch bietet sich der Q15-Datentyp an. Er basiert auf dem häufig unterstützten 32-Bit Integer Datenformat. Ein Bit zählt als Vorzeichenindikator. Die verbleibenden 31 Bit müssen, um eine Multiplikation in einem Schritt durchzuführen, doppelt so groß sein, wie der Integerbereich einer Zahl. Es werden darum 15 Bit für die Repräsentation der Zahlen gewählt. Diese können inclusive ihrem Vorzeichenbit platzsparend in einem 16-Bit Integer Datenformat gespeichert werden. Das Ergebnis

einer Multiplikation belegt 30 Bits. So können zwei Zahlen nach ihrer Multiplikation noch addiert werden, bevor ein anschließender Shift den Zahlenraum wieder auf 15 Bit beschränkt. Der bei der Addition entstehende Fehler ist somit vernachlässigbar gering. Genau diese Rechenweise läßt sich für den FFT-Algorithmus des COFDM-Modulators ausnutzen. So läßt sich der entstehende Fehler gering halten.

Der als Beispielhardware gewählte Prozessor unterstützt zudem einen präziseren 40-Bit Integer Datentyp. Mit diesem als Basis für eine Multiplikation könnten 4 Bits an Genauigkeit hinzugewonnen werden. Allerdings existiert kein 19-Bit Datentyp der gespeichert werden könnte. So müssen Abstriche beim Speicherverbrauch und auch bei der Geschwindigkeit gemacht werden. Die in 5.8 gemachten Aussagen über den zu erwartenden Fehler in der Berechnung zeigen, daß das Q15 Datenformat den Anforderungen genügt. Sollen doch noch Rechnungen mit mehr Bits an Auflösung durchgeführt werden, reichen die von der hardwareseitig angebotenen und unterstützten Datenformate nicht mehr aus. Die Multiplikation zweier 19 Bit Zahlen miteinander ergibt die größten von der Hardwareseite darstellbaren Zahlen. Um noch größere Auflösungen darstellen zu können, müßten jetzt die Zahlen durch mehr als eine 30 Bit Integer Variable zusammengesetzt werden. Für das Q30 Datenformat reicht es jedoch völlig aus, einen Algorithmus zur Verfügung zu stellen, der das Ergebnis einer Multiplikation korrekt im Q30-Format zurückgibt. Dabei werden die beiden Faktoren in je zwei gleichgroße Q15 Zahlen zerlegt, die als hochwertiger und als niederwertiger Summand behandelt werden. Das Ergebnis der Multiplikation ergibt sich gemäß dem Kommutativgesetz zu

$$(A_{hi} * 2^{15} + A_{lo}) * (B_{hi} * 2^{15} + B_{lo}) =$$
$$A_{hi} * B_{hi} * 2^{30} + A_{lo} * B_{hi} * 2^{15} + B_{lo} * A_{hi} * 2^{15} + A_{lo} * B_{lo}$$

Um das Ergebnis korrekt im Q30-Format darzustellen, werden nur die oberen 30 Stellen des Ergebnisses benötigt. Die Berechnung des letzten Produktes kann unberücksichtigt bleiben, da dabei ein vernachlässigbar kleiner Fehler entsteht. Insgesamt werden jedoch drei Multiplikationen und drei Additionen benötigt, um das Ergebnis zu erhalten. Der Rechenaufwand für einen Algorithmus steigt bei Verwendung des Q30 Formates entsprechend.

3.6 Galois-Felder

"Ein Galois-Feld ist ein Körper mit einer endlichen Anzahl von Elementen. Ein Galois-Feld mit q Elementen wird als GF(q) bezeichnet." ([20],Seite 100)

Ein Körper ist ein Zahlenraum, über dem zwei Funktionen definiert sind. Beide Funktionen müssen ein Element für Identität besitzen, die üblicherweise mit Null und Eins bezeichnet werden. Zu der ersten Funktion muß die Inverse existieren und eindeutig sein. Für die zweite Funktion gilt dasselbe mit Ausnahme des Nullelements.

Beispiel: Die ganzen Zahlen, die Addition und die Multiplikation bilden einen Körper. Die inversen Funktionen sind Subtraktion und Division. Das Identitätselement bezüglich der Addition ist die Null, das der Multiplikation die Eins.

Rechentabellen für die Modulo-2-Arithmetik in GF(2)

+/-	0	1	*	0	1
0	0	1	0	0	0
1	1	0	1	0	1

Es sind zwei Arten von Galois-Feldern bekannt, die Restklassen modulo einer Zahl und die Restklassen modulo eines Polynoms. Wird der zulässige Zahlraum bei einer Addition oder einer Multiplikation überschritten, wird der Rest nach einer Division als Ergebnis der Operation definiert. Bekanntestes Beispiel ist die Bool'sche Algebra, die auch als Restklassenarithmetik modulo Zwei in GF(2) aufgefaßt werden kann. Die ODER-Verknüpfung entspricht dabei der Addition, die UND-Verknüpfung der Multiplikation. Addition und Subtraktion sind identisch, ebenso Multiplikation und die Division, die jedoch nur für $1 \cdot 1$ definiert ist.

Bei der Restklassenrechnung modulo einem Polynom wird für jeden Koeffizienten des Polynoms ein Bit als Zahl in $GF(2)$ benutzt.

Rechentabellen für die Modulo-2-Arithmetik in $GF(2^2)$
mit Generatorpolynom $d^2 + d + 1$

+/-	0	1	d	d+1	*	0	1	d	d+1
0	0	1	d	d+1	0	0	0	0	0
1	1	0	d+1	d	1	0	1	d	d+1
d	d	d+1	0	1	d	0	d	d+1	1
d+1	d+1	d	1	0	d+1	0	d+1	1	d

In jedem Galois-Feld existiert mindestens ein primitives Element. Dieses zeichnet sich dadurch aus, daß alle Elemente des GF mit Ausnahmen der Null als Potenz des primitiven Elements dargestellt werden können. In obiger Tabelle sind d und $d+1$ primitive Elemente. Für d sind die Potenzen $d^0 = 1$,$d^1 = d$,$d^2 = d \cdot d = d + 1$, $d^3 = d \cdot (d+1) = 1$. Es läßt sich mit dieser Beziehung durch Angabe eines primitiven Elements α auch ein diskreter Logarithmus definieren: $\alpha^i = x \rightarrow log_\alpha(x) = i$. Das Ergebnis einer Multiplikation in GF(q) kann so auch über den diskreten Logarithmus und eine Addition berechnet werden. Es gilt:

$$x \cdot y = \alpha^{log_\alpha(x)+log_\alpha(y)}.$$

Sind die Logarithmen der Zahlen und die Exponenten von α in Tabellen abgelegt, läßt sich die Multiplikation effizient durch eine Addition und 3 Tabellennachschläge implementieren. Dadurch entfällt die Polynommultiplikation und die anschließende Division durch das Generatorpolynom. Die Größe der Tabellen liegt für GF(q) bei jeweils q Einträgen. Natürlich kann das Ergebnis einer Multiplikation auch direkt in der Tabelle nachgeschlagen werden. In diesem Fall benötigt man jedoch q^2 Einträge. Auch hier läßt sich wieder die Gesetzmäßigkeit aus Kapitel 3.7 anwenden.

3.7 Performance-Memory Modell

Die Optimierung einer Software hinsichtlich Speicherverbrauch und Geschwindigkeit muß normalerweise mit einem Kompromiß enden. Bei einem gewissen Grad der Optimierung läßt sich die Abarbeitung von Algorithmen nur noch beschleunigen, wenn man den Datenstrukturen mehr Platz einräumt. Zum Beispiel könnten zu speichernde Daten komprimiert abgelegt werden. Dazu wird aber Rechenzeit für das Packen und entpacken benötigt.

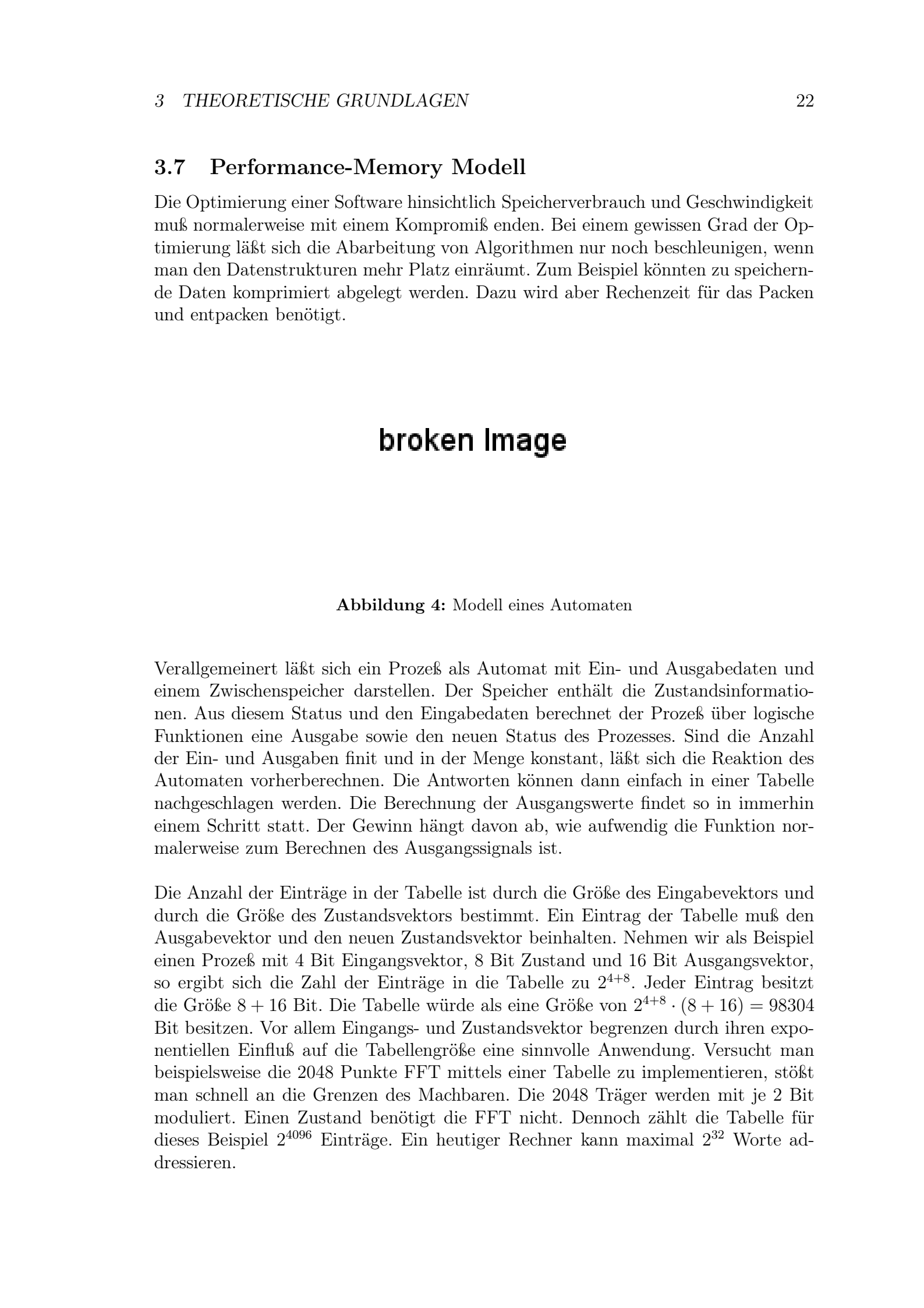

Abbildung 4: Modell eines Automaten

Verallgemeinert läßt sich ein Prozeß als Automat mit Ein- und Ausgabedaten und einem Zwischenspeicher darstellen. Der Speicher enthält die Zustandsinformationen. Aus diesem Status und den Eingabedaten berechnet der Prozeß über logische Funktionen eine Ausgabe sowie den neuen Status des Prozesses. Sind die Anzahl der Ein- und Ausgaben finit und in der Menge konstant, läßt sich die Reaktion des Automaten vorherberechnen. Die Antworten können dann einfach in einer Tabelle nachgeschlagen werden. Die Berechnung der Ausgangswerte findet so in immerhin einem Schritt statt. Der Gewinn hängt davon ab, wie aufwendig die Funktion normalerweise zum Berechnen des Ausgangssignals ist.

Die Anzahl der Einträge in der Tabelle ist durch die Größe des Eingabevektors und durch die Größe des Zustandsvektors bestimmt. Ein Eintrag der Tabelle muß den Ausgabevektor und den neuen Zustandsvektor beinhalten. Nehmen wir als Beispiel einen Prozeß mit 4 Bit Eingangsvektor, 8 Bit Zustand und 16 Bit Ausgangsvektor, so ergibt sich die Zahl der Einträge in die Tabelle zu 2^{4+8}. Jeder Eintrag besitzt die Größe $8 + 16$ Bit. Die Tabelle würde als eine Größe von $2^{4+8} \cdot (8 + 16) = 98304$ Bit besitzen. Vor allem Eingangs- und Zustandsvektor begrenzen durch ihren exponentiellen Einfluß auf die Tabellengröße eine sinnvolle Anwendung. Versucht man beispielsweise die 2048 Punkte FFT mittels einer Tabelle zu implementieren, stößt man schnell an die Grenzen des Machbaren. Die 2048 Träger werden mit je 2 Bit moduliert. Einen Zustand benötigt die FFT nicht. Dennoch zählt die Tabelle für dieses Beispiel 2^{4096} Einträge. Ein heutiger Rechner kann maximal 2^{32} Worte addressieren.

Ist die Tabelle zu groß, ist der Programmierer gefragt. Er kann prüfen, ob sich die Größe des Eingangsvektors noch reduzieren, oder sich wenigstens eine Teilfunktion in der Tabelle speichern läßt. Oft läßt sich die Funktion in zwei Teile trennen, die dann jeweils für sich in einer Tabelle nachgeschlagen werden. Für das obige Beispiel würde das zum Beispiel bedeuten, daß man zwei Automaten hintereinander schaltet, die jeweils einen Zustandvektor der Größe 4 Bit besitzen. Die Tabellen würden dann eine Größe von je $2^{4+4} \cdot (4 + 16) = 5120$ Bit besitzen, zusammen also ein Zehntel der ursprünglichen Tabelle. Während es für rein logische Funktionen Optimierungsverfahren zur Minimierung der Funktionszahl gibt, sind mir jedoch Verfahren für andere Funktionen wie <,>,if() und Tabellen nicht bekannt. Hier ist immer noch Kreativität und Programmierwissen gefragt.

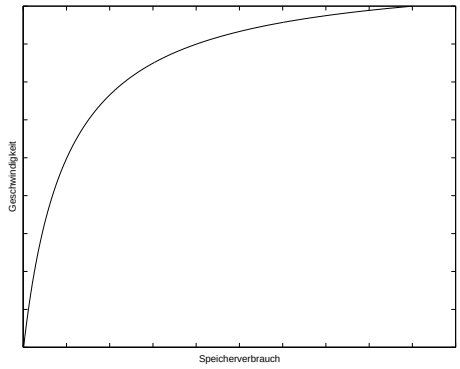

Abbildung 5: Qualitativer Verlauf der Geschwindigkeit über dem zur Verfügung stehenden Speicher

Durch das Aufteilen des Automaten sind zur Berechnung des Ausgangssignals jetzt zwei Tabellenzugriffe nötig. Dadurch ist der Speicherverbrauch um den Faktor zehn gesunken. Dieses Prinzip läßt sich nicht immer anwenden, soll jedoch, um eine allgemeine Aussage treffen zu könnnen generalisiert werden. Gehen wir also davon aus, daß jedes nicht interaktive Programm, welches Ein- und Ausgaben erzeugt, durch eine variable Zahl an Tabellenzugriffen realisiert werden kann. Trägt man die Größe aller Tabellen über die Zahl der Tabellenzugriffe auf, erhält man ein ungefähres Performance-Memory-Modell. In diesem Diagramm läßt sich ablesen, welche Möglichkeiten man als Kompromiß zwischen Speicherverbrauch und Geschwindigkeit offen hat. Generell wird eine effiziente Lösung immer in der Nähe der stärksten Rundung zu finden sein.

Dieses Modell ist leider noch zu primitiv, um der Wirklichkeit zu entsprechen. Die Zugriffszeit ist nämlich für alle Tabellen gleich angenommen worden. In der Realität muß jedoch, um eine um Größenordnungen verschiedene Speichermenge zu realisieren, auf unterschiedliche physikalische Technologien zurückgegriffen werden. Lassen sich heute mehrere Kilobytes in kleinen Caches mit denselben Taktraten

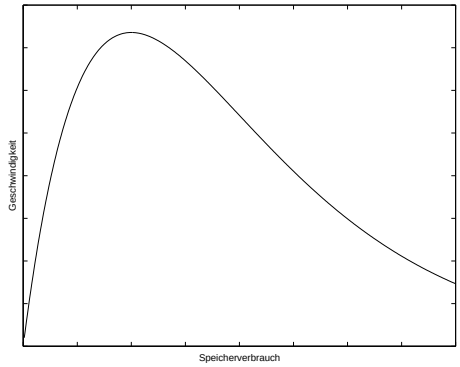

Abbildung 6: Qualitativer Verlauf der Geschwindigkeit über dem zur Verfügung stehenden Speicher unter Berücksichtigung von zunehmenden Zugriffszyklen

wie der Prozessor ansteuern, muß man zur Realisierung mehrerer Megabytes bereits mehrere Wartezyklen der CPU einkalkulieren. Bringt man dieses Wissen in das Geschindigkeits-Speicherverbrauch-Diagramm mit ein, erhält man eine Kurve mit einem Maximum. Ziel des Optimierungsverfahrens ist es jedoch nicht, dieses Maximum zu treffen. Schließlich gibt es noch andere Gründe, wie zum Beispiel Kosten, Systemgröße und Verfügbarkeit, welche bei der Realisierung eines Systems eine Rolle spielen. Es sollte jedoch bei der Implementierung berücksichtigt werden. Weiterhin sollte man sich merken, daß Funktionen schon durch Einsatz kleiner Tabellen deutlich beschleunigt werden können und daß große Tabellen durch Trennung verkleinert werden können. Bei der Optimierung der Funktionsblöcke ist jeweils versucht worden, den Arbeitsaufwand mit Hilfe von Tabellen zu verringern.

4 Systemanalyse

4.1 Informelle Analyse

Ziel der informellen Analyse soll es sein, einen Überblick über die gestellte Aufgabe zu geben. Dabei sollen vor allem die Grenzen des Systems abgesteckt werden und absehbare Probleme angesprochen werden. Es soll informiert werden, jedoch noch nicht spezifiziert, um der Kreativität bei der Umsetzung genug Freiraum zu lassen.

4.1.1 Modularer Aufbau

Einer der Hauptansprüche an das zu entwickelnde System ist die Forderung, es dynamisch weiterentwickeln zu können. Dabei wird davon ausgegangen, daß für eine optimale Ausnutzung der Hardwareressourcen auch eine Anpassung der Software in unterster Ebene notwendig ist. Das Grundsystem soll also durch austauschbare Module gebildet werden. Die Größe eines Moduls ist zunächst völlig willkürlich wählbar, kann aber durch eine rationale Trennung des Systems ein in sich strukturiertes Design bestimmen. Die Grenzen eines Moduls werden also durch vom System logisch definierte Grenzen vorgegeben. Um dennoch eine Flexibilität in der Größe der auszutauschenden Softwareteile zu bieten, wird die funktionale Programmierung, die Hochsprachen anbieten, zu Hilfe gezogen. Das Problem wird also zunächst in logisch zusammenhängende Blöcke unterteilt. Diese werden dann wieder funktional mittels der Hochsprache realisiert und bilden eine weitere hierarchische Schicht. Zwischen den einzelnen Modulen liegen definierte Schnittstellen, deren Anforderungen beim Austausch eines Moduls erfüllt werden müssen. Bei einer Neudefinition einer Schnittstelle müssen beide Module auch neu angepaßt werden. Das bedingt einen höheren Aufwand aber auch den größtmöglichen Freiheitsgrad. Grundsätzlich sollten die Module jedoch in sich geschlossen bleiben.

4.1.2 Evolutionäre Softwareentwicklung

Das freie Entwicklungskonzept ohne detailiert vorgeschriebene Spezifikation entspricht dem evolutionären Softwarekonzept. Vorteil dieses Vorgehens ist vor allem die schnelle Entwicklungszeit. Da an Spezifikation, Design und Validierung gleichzeitig gearbeitet werden kann und keine explizite Kontrollphase vorgeschrieben ist, können Änderungen und Evolution sehr schnell stattfinden. Der besondere Nachteil der fehlenden Kontrollphasen ist ein zunehmend unstrukturierter Sourcecode. Diesem soll durch die vorangehende Modularisierung und die Einführung definierter Schnittstellen entgegengewirkt werden. Sie ermöglichen dem Entwickler, sich auf einen begrenzten Bereich des Softwareprojekts zu beschränken. Die Definition des evolutionären Konzepts sowie andere Konzepte läßt sich in [23] nachlesen.

4.1.3 Unterscheidung zwischen Entwicklungs- und Zielsystem

Grundsätzlich ist Die hardwareunabhängigkeit des Konzepts auf Systeme beschränkt, die von den verfügbaren Werkzeugen unterstützt werden. Oberste Priorität genießt

dabei der Hochsprachencompiler, ohne den das Softwarekonzept nutzlos ist. Durch den Einsatz von Crosscompilern kann das Konzept in ein Entwicklungssystem und ein Zielsystem getrennt werden. Die Ansprüche an das Entwicklungssystem sind dabei andere als die des Zielsystems. Die an das Entwicklungssystem gestellten Anforderungen beinhalten die Forderung den Compiler, das Make-Utility und ein hierarchisches Filesystem zu unterstützen. Bei der Portierung auf eine andere Entwicklungsplattform müssen Änderungen an der Software vorgenommen werden. Das Konzept läßt sich jedoch übernehmen.

4.1.4 Benötigte Module des COFDM-Encoders

Der COFDM-Encoder besitzt vier Schnittstellen zur Außenwelt: zum einen die Eingabeschnittstelle des ETI-Datenstromes, zum zweiten die Ausgabe des zeitdiskreten Ausgangssignals und zum dritten das Bedienteil mit Status Ein- und Ausgabe. Dazu kommt noch eine Zeitreferenz. Die Ein- und Ausgaben sind prinzipiell von der verwendeten Hardware abhängig. Sie sollten trotzdem als jeweils eigenständiges Modul gekapselt werden, um der nachfolgenden Berechnung den größtmöglichen Freiraum zu gewähren. Die funktionalen Blöcke der eigentlichen Modulatorsoftware können nach dem Festlegen der Außenschnittstellen frei gewählt werden und sollten nach logischer Zusammengehörigkeit gewählt werden.

4.2 Strukturierte Analyse

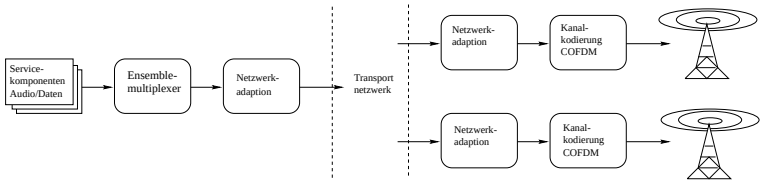

Abbildung 7: Oberstes Datenflußdiagramm für den COFDM-Encoder

Der Sinn der strukturierten Analyse ist es, eine präzise und vollständige Beschreibung des Systems zu liefern, um eine systematische oder parallele Umsetzung der einzelnen Teile zu ermöglichen, ohne dabei mit Überraschungen konfrontiert zu werden. Eine vollständige strukturierte Analyse des COFDM-Encoders findet sich in [21] und lag zu Beginn der Arbeit vor. Die Analyse bezieht sich dabei auf das realisierte Gerät, welches zusätzliche Funktionalität bietet, die über das eigentliche Modulationsverfahren hinausgeht. Bei der nun realisierten Aufgabe wurde die Analyse auf die zur Kanalkodierung und OFDM-Modulation benötigten Teile reduziert.

4.3 Design

4.3.1 Grenzen des Designs

Da das Design der COFDM-Software selber einem dynamischen Prozeß unterworfen werden soll, sind einer endgültigen Form der strukturierten Darstellung Grenzen gesetzt. Das realisierte System stellt eine Umsetzung aller zur Erzeugung des Ausgangssignals nötigen Teile dar. Gerade die hardwareabhänigen I/O-Prozesse sind jedoch nur in Ansätzen implementiert, sodaß sich das System testen läßt. Das entwickelte System ist als Grundlage zu verstehen, auf dessen Basis endgültige Versionen für einen COFDM-Modulator entstehen können. Dementsprechend stehen viele EXE-Files zur Verfügung, die unterschiedliche Funktionen erfüllen.

4.3.2 Aufbau der Hierarchien des Entwicklungssystems

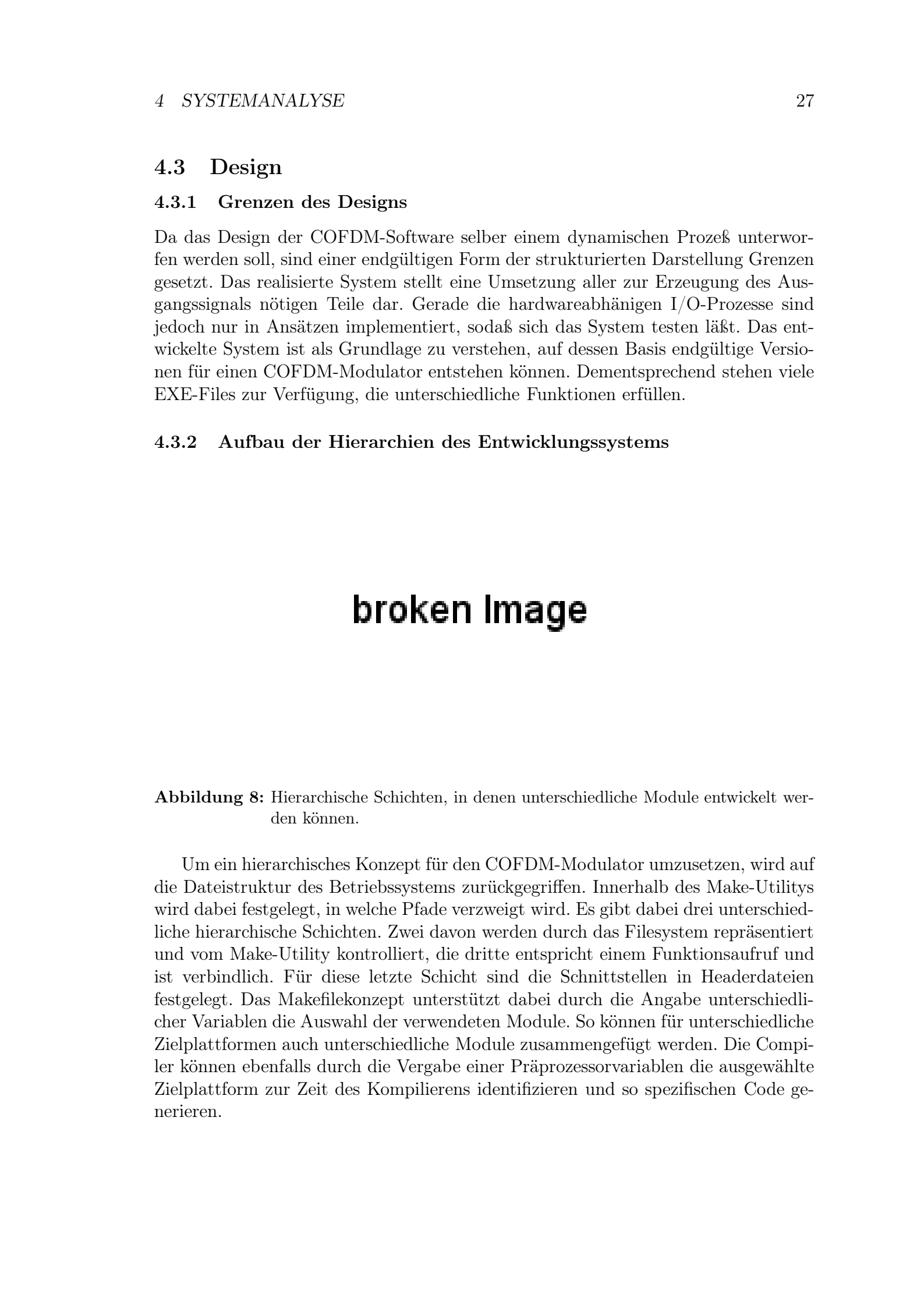

Abbildung 8: Hierarchische Schichten, in denen unterschiedliche Module entwickelt werden können.

Um ein hierarchisches Konzept für den COFDM-Modulator umzusetzen, wird auf die Dateistruktur des Betriebssystems zurückgegriffen. Innerhalb des Make-Utilitys wird dabei festgelegt, in welche Pfade verzweigt wird. Es gibt dabei drei unterschiedliche hierarchische Schichten. Zwei davon werden durch das Filesystem repräsentiert und vom Make-Utility kontrolliert, die dritte entspricht einem Funktionsaufruf und ist verbindlich. Für diese letzte Schicht sind die Schnittstellen in Headerdateien festgelegt. Das Makefilekonzept unterstützt dabei durch die Angabe unterschiedlicher Variablen die Auswahl der verwendeten Module. So können für unterschiedliche Zielplattformen auch unterschiedliche Module zusammengefügt werden. Die Compiler können ebenfalls durch die Vergabe einer Präprozessorvariablen die ausgewählte Zielplattform zur Zeit des Kompilierens identifizieren und so spezifischen Code generieren.

Abbildung 9: Struktur des COFDM-Encoders

4.3.3 Aufbau des COFDM-Encoders

Der COFDM-Encoder ist im wesentlichen durch eine Bibliothek repräsentiert. Die Bibliothek ist in sich geschlossen und enthält neben den Modulationsfunktionen noch Module zum Messen der Zyklenzahlen einer Routine und zum Speichern von Ausgaben. Die Ausgabe Ablaufinformationen kann an oder ausgeschaltet werden. Normale Ausgaben benötigen jedoch die vom Laufzeitsystem oder Betriebssystem zur Verfügung gestellten Ein- und Ausgabefunktionen. Alle sonstigen Ein- und Ausgabeoperationen sind aus der Bibliothek ausgeschlossen worden, da sie von der verwendeten Zielplattform abhängen.

4.4 Hardwareunabhängigkeit

Eine weitere Anforderung an das System ist die Portierbarkeit auf andere Hardwareplattformen. Prinzipiell gibt es sehr unterschiedliche Formen der Hardware, zum Beispiel 'Programmable Grid Arrays','PLD's' 'ASIC's und 'Digitale Signal Prozessoren', DSP's. Um das Softwarekonzept einschränken zu können, wurden als mögliche Zielplattformen allem voran "all purpose DSP's" gewählt. Es ist möglich die Software oder Teile davon, als Grundlage für andere Hardwarebeschreibungssprachen, wie zum Beispiel VHDL, zu verwenden. Dafür ist jedoch die Erstellung eines neuen, eigenständigen Konzeptes notwendig.

4.5 Programmiersprache

Für DSP-Plattformen gibt es in der Regel Entwicklungsumgebungen, welche Hochsprachencompiler und Simulator enthalten. Eine direkte Hardwareanpassung ist deshalb nicht notwendig. Als mögliche Hochsprachen sollen hier C, Ada und Java miteinander verglichen werden.

4.5.1 Java

Das Konzept der hardwareunabhängigen Software verfolgt auch JAVA. Bei dieser Sprache wird das JAVA-Programm zunächst in einen Byte-Code übersetzt. Erst dieser Byte-Code wird auf der Hardware ausgeführt oder emuliert. Dadurch wird im Prinzip eine weitere Schicht eingeführt, zu der sich auf beiden Seiten eine standardisierte Softwareschnittstelle befindet. JAVA ist eine C++ ähnliche Sprache. Sie ist jedoch überwiegend für die Entwicklung graphischer Oberflächen gedacht. Problematisch bei einer Umsetzung in JAVA ist vor allen Dingen die Abarbeitungsgeschwindigkeit. Da die Sprache auf einer standardisierten Schnittstelle aufsetzt, bleibt für hardwarenahe Maschinenbefehle und Optimierungen kaum Platz. Ebenso ist die Erweiterung eines auf JAVA aufbauenden Softwarekonzepts um maschinennahe Konstrukte wie zum Beispiel Assembler nicht möglich. Da generell eine effiziente Methode gesucht wird, ist JAVA aufgrund der Ausführungsgeschwindigkeit nicht geeignet.

4.5.2 Ada

ADA ist aus dem Wunsch nach Vereinheitlichung bei unterschiedlichen Softwareprojekten entstanden und findet vor allem in vielen staatlichen Projekten Verwendung. Ada ist eine ausgezeichnete Large-Scale Programmiersprache. Sie enthält beispielsweise Sprachkonstrukte für Interprozeßkommunikation und Synchronisation. Die Umsetzung eines effizienten COFDM-Modulators setzt jedoch vor allem die Entwicklung kleiner, schneller und speichersparender Funktionen voraus.

4.5.3 C

C ist von sich aus sehr hardwarenah. In dieser Hochsprache geschriebene Programme können sehr direkt auf die anschließende Assemblerebene übertragen werden. So ist eine Optimierung des Programms bereits in der Hochsprache möglich. Eine Anbindung von Assemblersprache und die damit verbundene hardwarenahe Anpassung auf niedriger Ebene ist ebenfalls möglich. Weiterhin ist C eine sehr verbreitete Sprache. Ein entsprechender Compiler ist für nahezu jede Prozessorumgebung vorhanden. Eine weitere Spezialität der Sprache ist die Unterstützung verschiedener Wortgrößen bei Integer- und Floatingpointdatentypen, die eine effiziente Ausnutzung des verfügbaren Speichers ermöglichen. Aus diesen Gründen wurde C für die Implementierung des COFDM-Enkoders gewählt.

5 Implementierung

5.1 Benutzte Werkzeuge/Umgebung

Die für die Softwareentwicklung nötigen Werkzeuge wurden von der Firma Bosch zur Verfügung gestellt. Zum Aufbau des hardwareunabhängigen Konzeptes diente hauptsächlich das GNU 'make'-tool. Die Flexibilität und die Menge an Kommandos, die dieses Werkzeug beinhaltet, machten ein ebenso flexibles all-in-one Gerüst für die Bibliothek möglich. Grundsätzlich läßt sich das Problem auch mit weniger komfortablen 'make'-Programmen lösen, eine Portabilität auf dieser Ebene ist jedoch nicht vorgesehen.

Als Compiler diente sowohl für Linux- als auch für Sun-Systeme der GNU-Compiler. Für das Testsystem mit dem Signalprozessor wurden spezielle Crosscompiler der Herstellerfirma zur Verfügung gestellt. Das Make-File Gerüst mußte für diese Compiler extra angepaßt werden. Die Anpassung der Makefiles an die Kommandozeilenoptionen sowie die Syntax für spezielle Einstellungen des Compilers muß für jeden neuen Compiler vorgenommen werden. Prinzipiell sind den Möglichkeiten jedoch kaum Grenzen gesetzt. Der Aufwand ist beschränkt, jedoch muß das Makefile-konzept bekannt sein. Zum Linken von Bibliotheken sind Archivierungstools der jeweiligen Zielplattform zum Einsatz gekommen.

Als Beispielhardware ist der Prozessor TMS320C6201 von TexasInstruments zu verwenden. Da Prozessoren einer sogenannten Familie angehören, die vom Befehlssatz und den verfügbaren Werkzeugen her nahezu identisch sind, können die Ergebnisse auf alle Schwesterprozessoren übertragen werden. Zu der Familie des C6201 gehören alle Prozessoren, deren Bezeichnungen mit TMS320C6 beginnen. Besonders interessant ist dabei die Ankündigung des Floatingpoint-Prozessors TMS320C67xx. Der C6201 ist ein reiner Integer-Prozessor. Floatingpointoperationen werden zwar unterstützt, werden jedoch durch Software emuliert und benötigen dementsprechend viel Rechenzeit.

5.2 Der COFDM-Modulator als Echtzeitsystem

Der COFDM-Modulator erhält als Eingaben einen ETI-Strom, ein Zeitsignal und Modusinformationen durch ein Bedienpult. Die dazu gehörenden Aufgaben sind die Erzeugung des Basisbandsignals, die Synchronisation mit dem Zeitsignal und die Interpretation und Ausgabe der Reaktion auf Bedienereingaben hin. Ist die Verarbeitung der einzelnen Schritte schnell genug, kann auf ein System mit konkurrierenden Prozessen verzichtet werden. Die einzelnen Arbeitsschritte können als getrennte Prozeduren realisiert werden, die nacheinander aufgerufen werden.

Die Bearbeitung der Bedienerinformationen stellt dabei keine hohen Anforderungen an das System. Es handelt sich dabei in erster Linie um das Setzen und Lesen des Systemzustandes. Die Zeittoleranzen dürfen für einen Bediener im Bereich von

Sekunden liegen und sind damit beim Vergleich mit den anderen Aufgaben unkritisch.

Die Synchronisation mit dem Zeitsignal stellt eine harte Grenze für die Erzeugung des Basisbandsignals dar. Grundsätzlich gilt, daß der Prozeß zur Berechnung des Signals schon vor der Synchronisation mit dem Zeitsignal angestoßen werden muß. Die Synchronisation kann dann über einen Zwischenpuffer erfolgen, dessen Inhalt synchron zum Zeitsignal ausgegeben wird. Die Dimensionierung dieses Speichers hängt sowohl von der Berechnungsweise des Signals, als auch von deren Berechnungsdauer ab. Durch einen entsprechenden Algorithmus ist hier eine Minimierung an die Systemvoraussetzungen möglich.

Die Berechnung des Basisbandsignals stellt die Hauptaufgabe des Systems dar. Sie ist auch die aufwendigste Aufgabe und bestimmt somit maßgeblich die Dimensionierung des Systems. Grundsätzlich muß das Signal kontinuierlich erzeugt werden. Um diese kontinuierliche Aufgabe in endliche Schritte endlicher Berechnungszeit zu unterteilen, muß das Signal im Zeitbereich ebenfalls in endliche Segmente, die unabhängig voneinander berechnet werden können, unterteilt werden. Hinzu kommt ein Pufferspeicher, aus dem das Signal gelesen werden kann, während der Prozessor sich den anderen Aufgaben widmet. Dieser Pufferspeicher kann gleichzeitig zur Abschwächung der sonst harten Grenze für die Erzeugung des Signals dienen.

Prinzipiell läßt sich die Größe eines zu berechnenden Signalabschnitts im Zeitbereich frei wählen. Je nach Größe werden jedoch unterschiedliche Anforderungen an die Berechnungsgeschwindigkeit und den Zwischenspeicher gestellt. Die Antwortzeit hängt von der Berechnungszeit für ein Segment ab. Sie ist unabhängig von der Hardware und logischerweise minimal für einen möglichst kurzen Teilabschnitt. Folglich sollte das Signal in möglichst kleine Teilstücke heruntergebrochen werden. Setzt man für die Erzeugung des Signals die Berechnung mittels einer Fouriertransformation voraus, ist das kleinste unabhängig voneinander berechenbare Teilstück des Signals ein Symbol. Der Funktionsaufruf für die Berechnung eines Symbols ist endlich und mit einem finiten Automaten darstellbar.

Für die realisierte COFDM-Modulator-Software ist somit ein Symbol als kleinster zu berechnender Signalabschnitt gewählt worden. Da die Struktur des COFDM-Signals einen logischen Aufbau von mehreren Symbolen, einen DAB-Rahmen, vorsieht, ist im Programm ein aufwendigerer Algorithmus nötig, der den Zustand der Symbolberechnung innerhalb eines Rahmens speichert und auswertet.

Obwohl der größte Teil der Berechnungen für die Symbole aufgewendet werden muß, hängt die Einhaltung der Zeitschranken auch von zufälligen Ereignissen ab. Insbesondere ist hier die Berechnung des Reed-Solomon-Encoders zu erwähnen. Der benötigte Aufwand hängt davon ab, ob die Daten fehlerfrei oder fehlerbehaftet am COFDM-Encoder ankommen. Die Rechenzeit ist bei maximaler Fehlerzahl etwa doppelt so hoch wie bei Fehlerfreiheit. Das gleiche Symptom verursacht ein Wechsel

im Multiplex oder ein fehlerhafter CRC. Prinzipiell läßt sich das System so definieren, daß auch im ungünstigsten Fall die harte Grenze immer eingehalten werden kann. Es gäbe allerdings auch die Möglichkeit, durch ein gezielt fehlerhaftes Signal die "firm-time-requirements" weiter abzuschwächen. So könnte die Berechnung der Senderkennung (TII) entfallen oder statt einer Fehlerkorrektur ein "muting", ein Überdecken des fehlerhaften Signals, durchgeführt werden. Prinzipiell soll der COFDM-Encoder jedoch auch bei fehlerhaftem Signal ein möglichst stabiles Signal liefern.

5.3 Prozeßoptimierung

In den folgenden Kapiteln sollen Ansätze zur Optimierung des COFDM-Enkoders aufgeführt werden. Die Optimierungen beziehen sich dabei auf die algorithmische Ebene. Es Übersicht über den generellen trade-off zwischen Speicher und Geschwindigkeit wird in [**?**] gegeben. Anschließend werden die arbeitsintensiven Funktionen des COFDM's einzeln besprochen.

5.4 Cyclic Redundancy Check

Der CRC "cyclic redundancy check" gehört zu den linearen, zyklischen Blockcodes. Blockcodes heißen so, weil jeweils einem Block mit K Eingangsbits ein Block mit N Ausgangsbits zugeordnet wird, wobei $N > K$ gilt. Zyklisch heißt der Code, weil ein Rotieren des Codewortes nach links oder nach rechts wieder ein Codewort ergibt. Linear heißt der Code, weil er mit Hilfe der Modulo-2-Arithmetik algebraisch beschrieben werden kann. Der im ETI Multiplex verwendete CRC ist ein systematischer Code. Das heißt, daß die Eingangsfolge von Datenbits auch genauso im zugeordneten Codewort wieder vorkommt. Das Codewort wird praktisch durch Anfügen von zusätzlichen Bits gewonnen. Diese Bits werden auch Paritybits oder Prüfstellen genannt.

Zyklische Codes zeichnen sich durch eine einfache Berechnungsmöglichkeit mittels der Polynomarithmetik aus. Die Koeffizienten des Polynoms sind die Binärstellen eines Vektors. Zwei Koeffizienten werden über die Modulo-2-Arithmetik verknüpft. Diese Rechenweise entspricht dem Rechnen in einem finiten Feld. Die Koeffizienten für den CRC stammen dabei aus dem Galois-Feld GF(2) mit den Elementen 0 und 1. Die Codeworte des CRC-Codes werden prinzipiell durch Multiplikation eines Eingangspolynoms mit einem Generatorpolynom $g(d)$ erzeugt:

$$f(d) = h(d) \cdot g(d)$$

Der Grad von h(d) ist K und der von f(d) ist N. Daraus folgt, daß der Grad von g(d) $k = N - K$ beträgt. Bei der Dekodierung wird das erhaltene Codewort $f'(d) = f(d) + e(d)$ wieder durch das Generatorpolynom geteilt und kontrolliert, ob das Restpolynom gleich 0 ist. Ist das der Fall, wird davon ausgegangen, daß kein Übertragungsfehler stattgefunden hat.

Dieses Vorgehen erzeugt jedoch in der Regel keinen systematischen Code. Um die Eingangsfolge in das Codewort zu integrieren, wird etwas anders vorgegangen. Die Eingangsfolge wird zunächst durch Multiplikation mit d^k auf die Größe eines Codewortes gebracht. Anschließend wird das nächstgrößere Polynom gesucht, welches durch das Generatorpolynom teilbar ist und dieses als Codewort dem Eingangspolynom zugeordnet. Dieses geschieht einfach durch Division der vergrößerten Eingangsfolge $d^k \cdot h(d)$ durch das Generatorpolynom. Der Rest der Division wird einfach zu der vergrößerten Eingangsfolge hinzuaddiert und ergibt das gesuchte Codewort.

$$f(d) = d^k \cdot h(d) + modulo(d^k \cdot h(d), g(d))$$

Die Multiplikation d^k bedeutet einfach das Anfügen von k Nullen an das Polynom. Das Generatorpolynom für den im ETI-Mux verwendeten CRC lautet:

$$g(d) = \ d^{16} \ + \ d^{12} \ + \ d^5 \ + \ d^1$$

Eine eindeutige Zuordnung zu Codewörtern ist nicht für beliebig lange Eingangssequenzen möglich. Das Polynom g(d) stellt zugleich das Polynom niedrigsten Grades dar, welches ein Codewort ist, da $g(d) = g(d) * 1$. Das erste Polynom, welches durch g(d) teilbar ist, jedoch kein Polynom des Codes mehr sein kann, ist $d^{2^{16-1}} + 1$. Würde es ein Codewort sein, müßte auch seine zyklische Vertauschung $d + 1$ Codewort sein. Dieses Polynom besitzt jedoch einen geringeren Grad als g(d). Die Codewortlänge beträgt also $2^{16-1} - 1 = 32767$ Bits.

Für die Berechnung der CRC Paritybits sind die führenden Nullen eines Eingangspolynoms nicht relevant. So können auch kurze Eingangssequenzen effizient codiert werden, indem die Nullen einfach übersprungen werden. Werden die führenden Nullen erst gar nicht übertragen, spricht man von einer Verkürzung des Codes.

Für die Berechnung des CRC's verlangt die Spezifikation außerdem, daß das CRC-Register zu Beginn und am Ende der Division invertiert wird. Dadurch werden die zyklischen Eigenschaften des Codes wieder aufgehoben. Ein um ein Bit verschobenes Codewort wird jetzt als Fehler erkannt. Weiterhin bestehen die Paritybits einer Nullfolge nicht aus Nullen.

Dieser CRC detektiert:

1. alle Fehlermuster mit ungeradem Gewicht
2. alle Fehlermuster mit Gewicht< 5
3. alle Bündelfehler mit einer Länge von 16 Bit
4. wenigstens 99.997% aller Fehlerbündel der Länge 17 und größer

5.4.1 Optimierung des CRC's

Die Polynomdivision kann mittels eines rückgekoppelten Schieberegisters erfolgen. Die bitweise Nachbildung dieses Prozesses in Software ist jedoch ineffizient. Besser ist es, eine größere Zahl an Eingangsbits zu betrachten. Geht man der bitweisen Nachbildung analytisch für mehrere Bits nach, erhält man ein Bool'sches Gleichungssystem, mit dem sich der nächste Zustand des CRC-Registers berechnen läßt. Diese Funktionen lassen sich nun zum Beispiel mit Quine-McClusky-Verfahren optimieren. Ebenso können sie in einer Tabelle nachgeschlagen werden. Die im Rahmen dieser Diplomarbeit erstellte Bibliothek stellt diesen 16 Bit CRC mit einem tabellengestützten Verfahren zur Verfügung. Dabei werden die Eingaben in Blöcken von je 8 Bit verarbeitet. Die Tabelle besitzt demnach eine Größe von 256 Einträgen mit jeweils 16 Bit pro Eintrag für den Zustand des CRC-Registers.

5.5 Pseudo Random Binary Sequence

Ähnlich wie der CRC besteht auch die PRBS aus einer Divisionsschaltung. Das Generatorpolynom muß in diesem Fall ein primitives Polynom sein. In der Folge der Berechnung treten alle möglichen Polynome mit Ausnahme des Nullpolynoms als Rest auf. Das verwendete primitive Polynom lautet

$$f(d) = d^9 + d^5 + 1$$

Die Periode der damit erzeugten Bitfolge beträgt $2^9 - 1 = 511$ Bit. Neben der Möglichkeit, die PRBS durchgehend zu berechnen, steht hier die Möglichkeit offen, einfach die gesamte Sequenz im Speicher abzulegen. Damit läge der Speicheraufwand für die PRBS auch bei 511 Bit. Da jedoch zum Verknüpfen eines Wortes mit einer beliebigen Sequenz der PRBS die Wortgrenzen der Speicheraddressierung überschritten werden können, sind zusätzliche Shift-Operationen nötig. Um dies zu vermeiden, ist in der realisierten PRBS nicht eine, sondern 32 PRBS-Folgen hintereinander abgespeichert. So ist jede beliebige PRBS-Sequenz auch in passenden Wortgrenzen zur Addressierung vorhanden. Der Speicherverbrauch liegt zwar jetzt bei $511 \cdot 32$ Bit, zum Verknüpfen eines Wortes mit der PRBS ist jetzt jedoch nur ein Speicherzugriff und eine Exor-Funktion nötig.

5.6 Convolutional Coder und Punktierer

Der Convolutional Coder erzeugt den Fehlerschutz für die Übertragung. Die Coderate verringert sich dabei auf 1/4. Zu jedem Eingangsbit wird eine Folge aus 4 Ausgangsbits in Folge gebildet. Um die Coderate anschließend wieder anheben zu können, werden nach bestimmten Mustern wieder bis zu 3 der 4 Bits verworfen. Diesen Vorgang nennt man auch Punktieren. Die Coderate wird damit von 8/9 bis 8/32 in 24 Stufen einstellbar.

Auch hier wurde ein tabellengestütztes Verfahren entwickelt. Betrachten wir dazu zuerst alle nötigen Ein- und Ausgabedaten. Der Zustand des Coders ist durch die

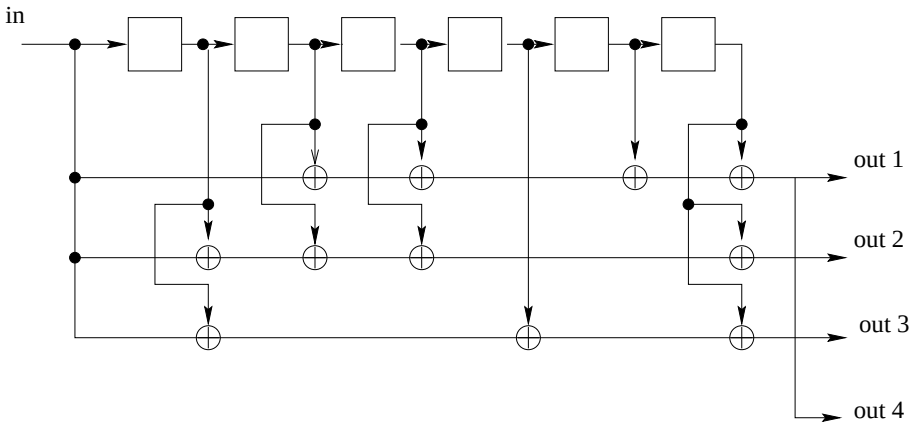

Abbildung 10: Berechnungsvorschrift des Faltungskodierers

sechs Schieberegister bestimmt. Versucht man, Coder und Punktierer zusammen in einer Funktion zu realisieren, benötigt man den Punktierindex von 1 bis 24 dazu, die mit 5 Bits codiert werden können. Das macht zusammen schon $6+5 = 11$ Bit. Dazu kommt noch eine frei wählbare Zahl an Eingabebits. Die Tabelle wächst dabei sehr schnell. Um die Größe zu reduzieren, wurde versucht, Coder und Punktierer wieder zu trennen. Als Eingangsgröße wurden 4 Bits gewählt, so daß sich die Tabelle für den Convolutional Coder auf $2^{6+4} = 1024$ Einträge beläuft. Jeder Eintrag besitzt aufgrund der Coderate von 1/4 eben $4 \cdot 4 = 16$ Bits als Ausgang. Die Tabelle für den nachfolgenden Punktierer würde also 2^{16+5} Einträge benötigen. Dies läßt sich, da jeweils 2 Ausgangsbits pro Eingangsbit des Codierers identisch sind, noch auf 2^{12+5} verringern. Da das immer noch zuviel ist, wurde auf ein Tabellenverfahren verzichtet und der Punktierer rein algorithmisch implementiert.

Ergebnis des tabellengestützten Verfahrens für den Convolutional-Coder ist eine Veränderung des Worst-Case-Verhaltens. Die ursprüngliche Funktion, die die 4 Ausgangsbits sequentiell berechnete, besaß den größten Aufwand für die Berechnung der ersten beiden Ausgangsbits. Das entspricht einer Coderate von 8/16. Da die Zeit zur Berechnung der Convolutional Coded Bits durch den Tabellennachschlag konstant gehalten wird, hängt der Zeitaufwand für den neuen Coder von der Zahl der Tabellennachschläge insgesamt ab. Werden 3 der 4 Ausgangsbits wieder verworfen, muß pro Ausgangsbit in diesem Fall ein Tabellennachschlag durchgeführt werden. Der maximale Aufwand muß also bei einer Coderate von 8/9 betrieben werden. Dieses muß bei der Untersuchung des COFDM-Modulators auf seine Echtzeitfähigkeiten hin berücksichtigt werden. Insgesamt ergab sich eine Speed-Up-Verbesserung des Worst-Case um den Faktor 1,5 für das tabellengestützte Verfahren. Für den Best-Case bei einer Coderate von 1/4 ergab sich sogar ein Speed-Up von Faktor 3.

5.7 Reed-Solomon Dekoder

Auch der Reed-Solomon-Code gehört zu den linearen, zyklischen Blockcodes. Dazu sind RS-Codes eine Untermenge der BCH-Codes.

$$RS \subset BCH \subset zyklisch \subset linear \subset Blockcode$$

Auch der Eingangsvektor des RS-Code wird als Polynom aufgefaßt. Genau wie beim CRC lassen sich die Paritybits durch Polynomdivision berechnen. Wesentlicher Unterschied ist jedoch, daß die Koeffizienten des Polynoms nicht aus dem Galois-Feld $GF(2)$ sondern aus $GF(2^8)$ stammen. Es werden also jeweils 8 Bit zu einem Koeffizienten zusammen gefaßt. Das Generatorpolynom wird als Produkt von k Wurzeln gebildet:

$$g(d) = \prod_{i=0}^{k} (d - \alpha^{l+i})$$

5.8 FFT Fehlerberechnung

Natürlich ist die Berechnung der FFT mit einer endlichen Registerbreite mit einem Fehler behaftet. Art und Größe des Fehles hängen dabei von dem verwendeten Algorithmus ab. Als Grundlage für die Berechnung soll vom Radix-2 Algorithmus ausgegangen werden. Betrachten wir aber zunächst das Eingangssignal der FFT, wie es vom ersten Teil des COFDM-Encoders erzeugt wird. Auffällig ist hierbei, daß alle Träger die gleiche auf Eins normierte Amplitude besitzen. Sie sind nur in der Phase gedreht. Problematisch ist die dazu notwendige Darstellung von $frac1\sqrt{}(2)$. Diese läßt sich nämlich nicht genau darstellen. Gleiches gilt für Träger, die linear vorverzerrt wurden. Auch ihre Werte lassen sich nur fehlerbehaftet darstellen. Der Fehler beträgt dabei $\frac{1}{2}2^{-B}$ wenn B die Zahl der Bits angibt, die für die Darstellung der Zahl benutzt werden. Die einzelnen Fehler sind unkorreliert und gleichverteilt. Mit diesem fehlerbehafteten Signal führen wir nun die Rechenoperationen der FFT durch. Da in jeder Stufe dieselben Rechenoperationen durchgeführt werden, reicht es, den Fehler für eine Stufe zu berechnen und dann das Ergebnis zu verallgemeinern. Der Fehler ist für jede Rechenart individuell zu bestimmen. Es sollen dabei Gleitkomma- und Fixkommaberechnung sowie spezielle Fixkommaverfahren untersucht werden.

Als Referenz diente eine 64 Bit Floatingpoint FFT. Alle Messungen sind bei 1024 Punkten durchgeführt worden. Die abgebildeten Fehlerdichten stammen von 99 zufällig erstellten Eingangsmustern. Die einzelnen Träger sind dabei vergleichbar dem COFDM-Signal mit

$$F(k) = \pm maxin \pm j \cdot maxin$$

moduliert worden. Die Signalamplitude des COFDM-Ausgangssignals liegt, da nur $\frac{3}{4}$ der Eingangsträger moduliert werden, um diesen Faktor niedriger.

Betrachten wir zunächst eine Stufe einer Radix-2 FFT. Es findet eine komplexe

Multiplikation statt sowie eine komplexe Addition und eine komplexe Subtraktion. Addition und Subtraktion verdoppeln den Wertebereich des Ausgangssignals einer Stufe. Die Multiplikation entspricht nur einer Drehung der Träger und verändert den Wertebereich nicht. Die Multiplikation findet aber mit ebenfalls fehlerbehafteten Werten statt. Bei diesem reinen FFT-Algorithmus ist der Wertebereich des Ausgangssignals um den Faktor N größer als der Wertebereich des Eingangssignals. Dies ist vor allem bei der Fixkommaberechnung zu berücksichtigen. Es werden alle N Werte miteinander verknüpft. Die Summe aller Operationen hängt ebenfalls von der Zahl der Samples N ab. Der relative Fehler wird sich also für große Werte von N auch vergrößern. Bei der Fehlermessung ging es haupsächlich darum festzustellen, wieviele der für Real- und Imaginärteil verwendeten Bits nach der Berechnung noch brauchbar sind. Deshalb sind Real- und Imaginärteil als voneinander unabhängig betrachtet worden.

5.8.1 Floatingpoint

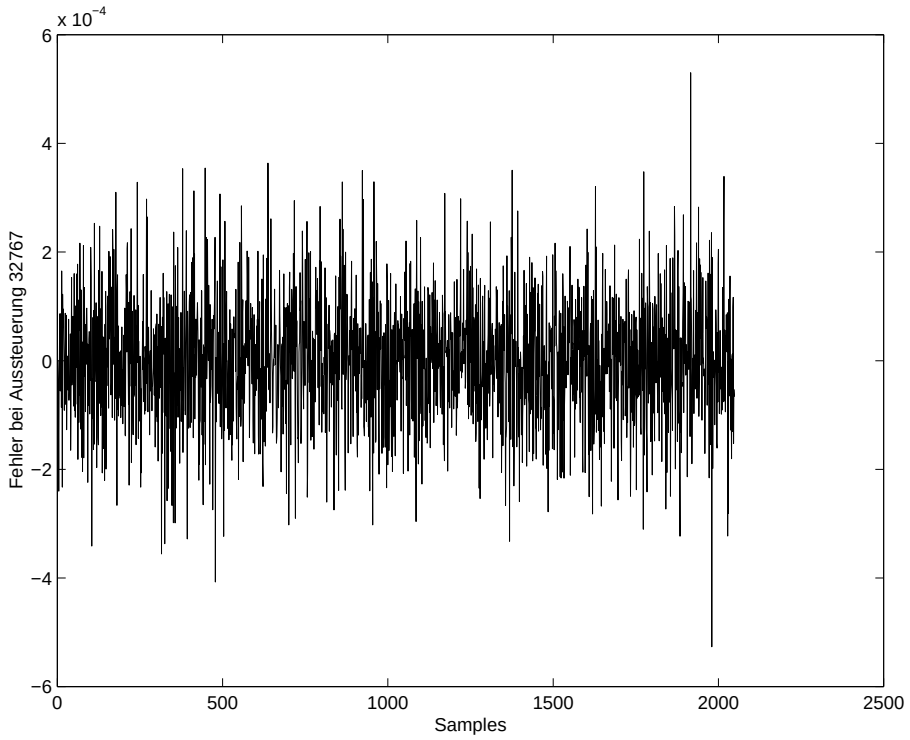

Abbildung 11: Fehler bei 32 Bit Floatingpoint Berechnung

Bei der Floatingpoint-Darstellung werden die Zahlen über Mantisse und Exponent dargestellt. Die Mantisse gibt dabei die Auflösung der Zahl an, und stellt Zahlen von $1,0$ bis $1,99...$ dar. Der Exponent gibt an, mit welchem Faktor die Mantisse noch zu multiplizieren ist, um die richtige Größenordnung zu haben. Die Auflösung ist dabei für jede Zahl aus dem darstellbaren Zahlbereich gleich. Bei einer Addition

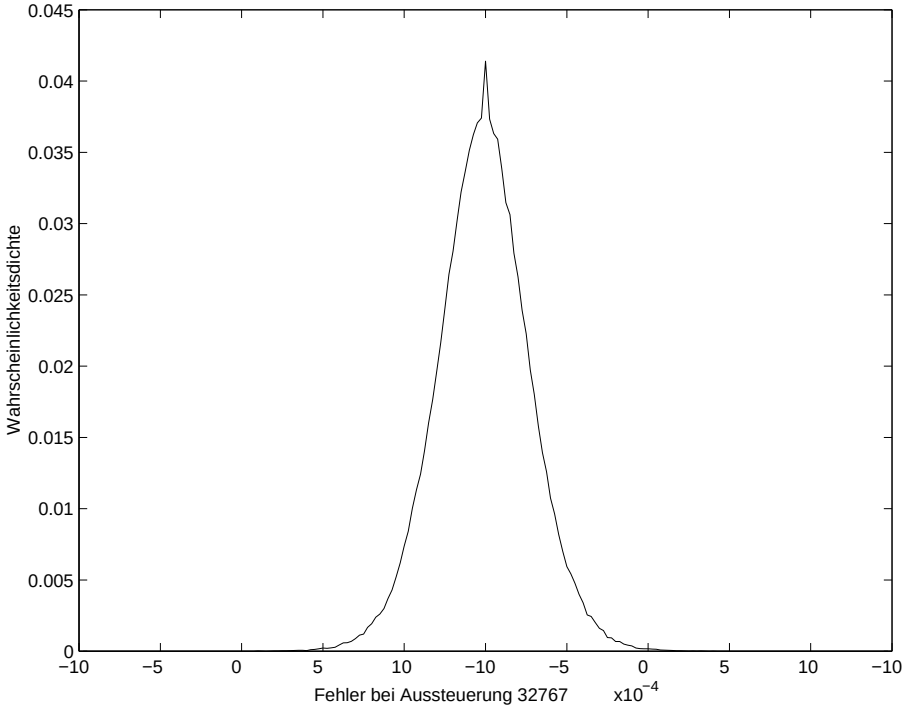

Abbildung 12: Fehlerdichte bei 32 Bit Floatingpoint Berechnung

addieren sich die absoluten Fehler der einzelnen Zahlen. Eine signifikante Vergrößerung des gesamten Fehlers findet nur statt, wenn beide Zahlen ungefähr die gleiche Größe besitzen. Bei einer Multiplikation addieren sich die relativen Fehler. Da die Multiplikation jedoch immer mit einem Twiddle-Faktor stattfindet, dessen Fehler immer in der Größenordnung der Auflösung liegt, kommt jedesmal nur ein prozentual ungefähr gleichgroßer Fehler hinzu.

Da das Ausgangssignal um den Faktor N größer ist als das Eingangssignal, relativieren sich auch die Fehler entsprechend. Es werden insgesamt N fehlerbehaftete Werte zueinander aufaddiert. Dazu kommt der Fehler durch die Multiplikationen. Der Fehler wird jedoch mit dem Faktor $\frac{1}{N}$ bedämpft. Insgesamt ist ein prozentualer Fehler zu erwarten, der nicht größer als das $log(N)/2$ fache des Eingangssignals ist. Bei der Messung wird der Fehler eines 32 Bit Floatingpoint-Formats mit dem eines 64 bittigem verglichen. Die Messung fand bei $N = 1024$ Punkten und einem Eingangssignal bestehend aus 1024 QPSK-modulierten Trägern statt. Es zeigt sich, daß der praktisch zu erwartende Fehler noch deutlich unter dem Schätzwert liegt.

5.8.2 Fixpoint

Die Rechenweise im Fixpoint-Format wurde in Kapitel 3.5 schon angesprochen. Da beim Addieren zweier Fixpunktwerte kein Rundungsfehler entstehen kann, läßt sich zur Berechnung des theoretischen Fehlers ein vereinfachtes Blockschaltbild angeben.

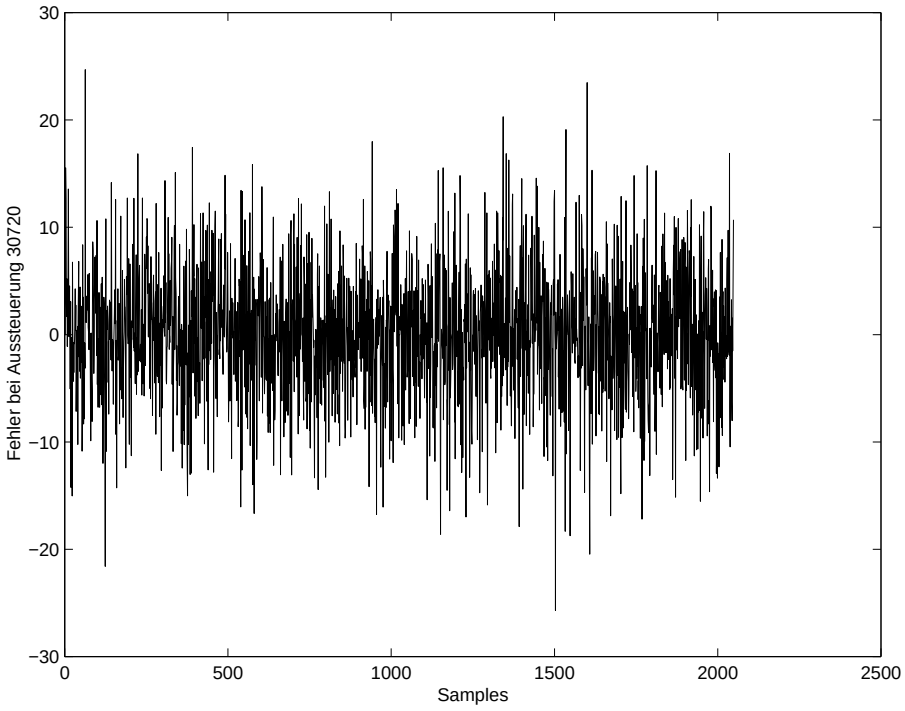

Abbildung 13: Fehler bei 16 Bit Fixpunkt Berechnung

Zusätzliches Rauschen wird dabei nur durch die Multiplikation erzeugt. Eine Berechnung des theoretischen Fehlers findet sich in [7] und [4]. In der Praxis kann der Fehler durch ein entsprechendes Berechnungsverfahren noch um die Hälfte reduziert werden. Dies folgt aus dem speziellen Berechnungsverfahren der komplexen Multiplikation. Wie in Kapitel 3.5 schon angesprochen, wird dabei das Ergebnis von zwei reellen Multiplikationen erst geshiftet, nachdem sie zusammen aufaddiert worden sind. Dadurch entsteht pro komplexer Multiplikation jeweils nur ein Rundungsfehler für Real- und Imaginärteil.

Bei dem Eingangszahlraum der FFT muß darauf geachtet werden, daß im Laufe des Algorithmus kein Überlauf stattfindet. Die maximale Signalamplitude nimmt um den Faktor N bei der Berechnung zu. Als Eingangswerte für 15 Bit Register darf deshalb maximal der Wert 15 gewählt werden. Das Ausgangssignal der FFT besitzt demnach für die 2048 Punkte-FFT des Mode 1 die maximale Amplitude von $2048 \cdot 15 = 30720$. Im COFDM-Encoder darf mit dem Wert 21 ausgesteuert werden, da die Maximalamplitude wegen der Nullträger um $\frac{1}{4}$ niedriger liegt. Als Maximalamplitude ergibt sich $2048 \cdot 21 \cdot \frac{3}{4} = 32256$. Die Fehler addieren sich von Stufe zu Stufe. Pro Stufe kann ein Fehler von ± 5 angenommen werden. Dementsprechend läßt sich der Fehler für die 2048 Punkte FFT des Mode 1 angeben.

5.8.3 Blockgleitkomma

Bei der Blockgleitkommarechnung ist der Zuordnungsfaktor zwischen Integerzahl und dargestellter Kommazahl nicht fest, sondern wird während des Ablaufes des

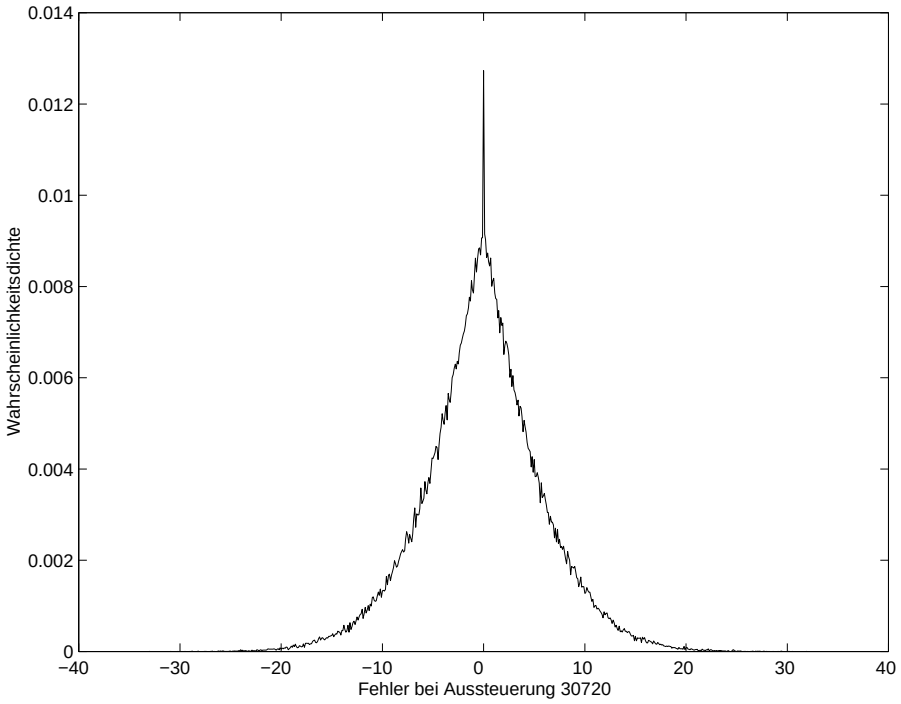

Abbildung 14: Fehlerdichte bei 16 Bit Fixpunkt Berechnung

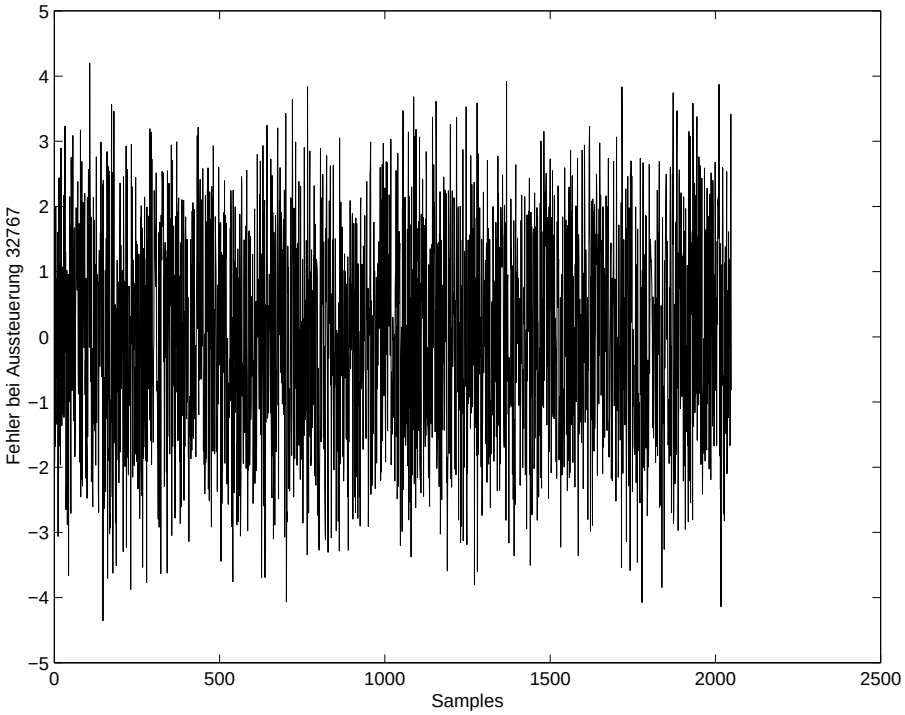

Abbildung 15: Fehler bei 16 Bit Blockgleitkomma Berechnung

Algorithmus dynamisch an den benötigten Zahlraum angepaßt. Dies ist für den FFT-Algorithmus kein Problem, da der Algorithmus stufenweise abläuft. Der benötigte Zahlraum verdoppelt sich dabei von Stufe zu Stufe. Die Anpassung der Ausgangszahlen erfolgt entsprechend durch eine Division durch Zwei am Ende einer Stufe. Insgesamt entspricht das Ergebnis dann einer FFT mit Vorfaktor $T1 = \frac{1}{N}$.

Dadurch, daß die Fehler in den ersten Stufen der FFT stark bedämpft werden,

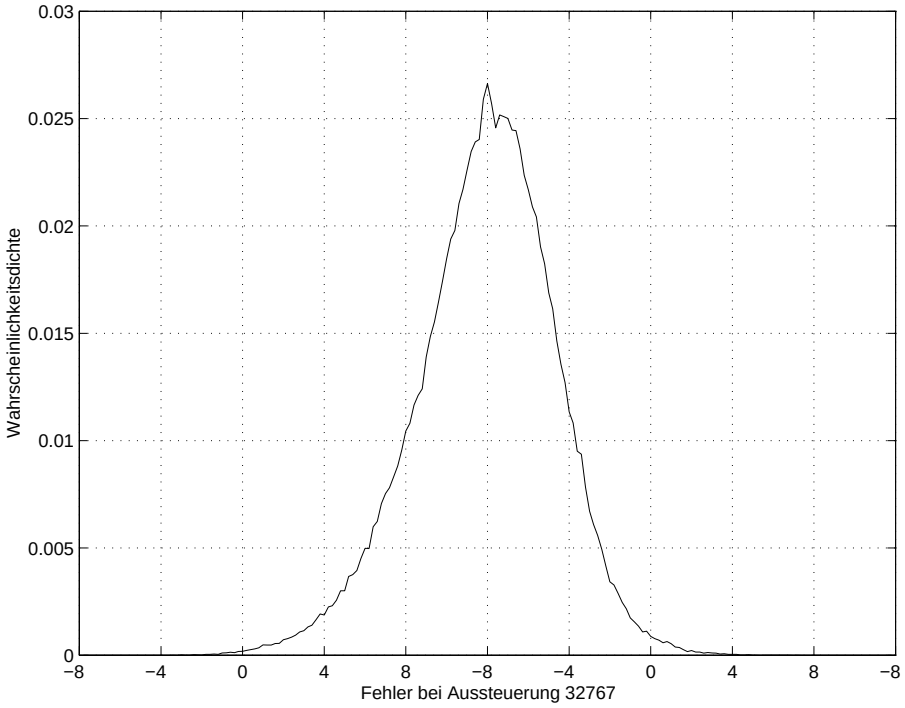

Abbildung 16: Fehlerdichte bei 16 Bit Blockgleitkomma Berechnung

sind die am Ende entstehenden Fehler ausschlaggebend. Der Fehler liegt nach der Fixpointmessung bei ±5 pro Stufe, wird hier jedoch mit dem Faktor 2 bedämpft. Als Fehler der letzten Stufe bleibt also der Wert 2,5. Die vorhergehende Stufe ist um den Faktor 2 bedämpft, und liefert also einen Beitrag von 1.25 Bit. Als maximalen Fehler läßt sich also nach 5 Stufen ein Wert von 4.84 Bit angeben. Da der Fehler der ersten Stufe nach wenigen Dämpfungen vernachlässigbar ist, sind die Diagramme auf die 2048 Punkte FFT des Mode 1 übertragbar.

Der maximale Zahlraum des Eingangssignals beträgt bei 15 Bit Registerbreite plus einem Vorzeichenbit -32767 bis 32767. Nach der Rechnung werden von der Q15 Zahlendarstellung noch $ld(32767) - ld(4,84) = 12,7$ Bits fehlerfrei dargestellt. Dieses Ergebnis wird durch die Messung bestätigt.

Der Aufwand in der innersten Schleife der FFT ist bei diesem Algorithmus jedoch auch am höchsten. Es kommt zu jedem Operanden eine Shiftoperation hinzu. In der

Performanceanalyse wird darauf noch einmal eingegangen werden.

5.9 FFT-Implementation

Die IFFT, die im COFDM verwendet wird, besitzt mehr Punkte, als eigentlich für die Träger nötig wären. Es wird tatsächlich immer die nächste Zweierpotenz verwendet, was ungefähr $\frac{4}{3}$ mehr Frequenzen bedeutet. Die zusätzlichen Frequenzen werden dabei einfach zu Null gesetzt. Da dies in einer höheren Abtastrate resultiert, sind auch die am Anschluß an den COFDM-Encoder befindlichen Systeme wie der I/Q-Modulator auf die erhöhte Samplingrate eingestellt. Dadurch, daß ein $\frac{1}{4}$ der Frequenzen gleich Null ist, läßt sich auch ein spezieller FFT-Algorithmus finden, der in den ersten Stufen die Berechnung dieser Werte ausläßt. Der Geschwindigkeitsvorteil liegt jedoch nur bei etwa 10%.

Die Twiddlefaktoren der FFT können in einer Tabelle gespeichert werden. Die Algorithmen von Texas Instruments verwenden dabei eine spezielle Technik, bei der jeweils Real- und Imaginärteil des Twiddlefaktors gleichzeitig in ein Register geladen werden. Dazu ist es aber notwendig, daß die Tabelle eine komplette Sinus- und Cosinusschwingung enthält. Da die Funktionen bis auf eine Phasenverschiebung identisch sind, kann durch eine geschickte Addressierung die Größe um etwa die Hälfte reduziert werden. Unter Ausnutzung weiterer Symmetrien läßt sich die Tabelle sogar auf ein Achtel der ursprünglichen Größe reduzieren, dies erfordert jedoch einen nicht unerheblichen zusätzlichen Rechenaufwand.

5.10 OFDM-Signalpfadoptimierung

Eine Optimierung der FFT ist vor allem bei der Betrachtung des Gesamtsystems des OFDM-Generators möglich. Es soll dazu der Signalpfad hinsichtlich Optimierungsmöglichkeiten untersucht werden.

5.10.1 FFT-Reversal und FFT-Shift

Der Signalpfad zur OFDM-Generierung enthält mehrere Stufen, die das Signal lediglich umordnen ohne eine Berechung durchzuführen. Dazu gehören das Frequenzinterleaving, in welchem die Träger nach der ETI-Spezifikation [1] umgeordnet werden. Weiterhin gehört ein FFT-Shift dazu, der die negativen Träger des Basisbandsignals auf die positive Seite spiegelt und letztlich das FFT-Reversal, was die beim Radix-2 Algorithmus verwürfelten Samples wieder in die richtige Reihenfolge bringt. Sinn einer Optimierung ist es, diese Aufgaben in möglichst einem Schritt auszuführen. Wird als FFT-Algorithmus ein “decimation in time”-Verfahren benutzt, kann das FFT-Reversal vor die IFFT verschoben werden. Um FFT-Reversal und -Shift auch vor die anderen Funktionsblöcke ziehen zu können, muß nur beachtet werden, daß die zugeführten Signale insbesondere vom TII- und TFPR-Symbolgenerator schon in der umgeordneten Form vorliegen. Dann können alle Umordnungen zu einem Schritt

zusammengefaßt werden. Die Zuordnung der Träger zu ihrem Platz im umgeordneten Spektrum geschieht am effizientesten über eine Tabelle.

Abbildung 17: Blockschaltbild des realisierten OFDM-Symbolgenerators. Der endgültigen Version fehlt noch eine Vorverzerrung.

5.10.2 Digitale Differentielle Modulation

Das Signal verläßt im Prinzip bei QPSK-Codierung der einzelnen Träger den Bereich des Digitalen. Es wird anschließend differentiell moduliert und dann der IFFT zugeführt. Bei einer linearen Vorverzerrung des Signals im Frequenzbereich muß dieser Schritt hier berücksichtigt werden. Um den Rechenfehler möglichst gering zu halten, muß das wertkontinuierliche Signal möglichst weit am Ende des Graphen eingeführt werden. Es ist möglich, die Einführung von Floatingpoint-Werten bis hinter die differentielle Modulation zu verschieben. Die Differenzkodierung der einzelnen Träger findet dabei noch digital statt. Da das differentielle Symbol pro Träger nur acht mögliche Phasen besitzt, läßt es sich mit drei Bit pro Träger zudem sehr platzsparend darstellen. Im Gegensatz dazu fällt ein wertkontinuierliches Signal mit beispielsweise 16 Bit jeweils für Real- und Imaginärteil ins Gewicht.

Die Berechnung der Phasendifferenz läßt sich ebenfalls optimieren. Die kontinuierliche Berechnung erfordert hier eine komplexe Multiplikation bestehend aus vier reellen Multiplikationen und zwei Additionen. Es soll an dieser Stelle angemerkt werden, daß es zu der komplexen Multiplikation,

$$(a + jb) \cdot (c + jd) = R + jI$$

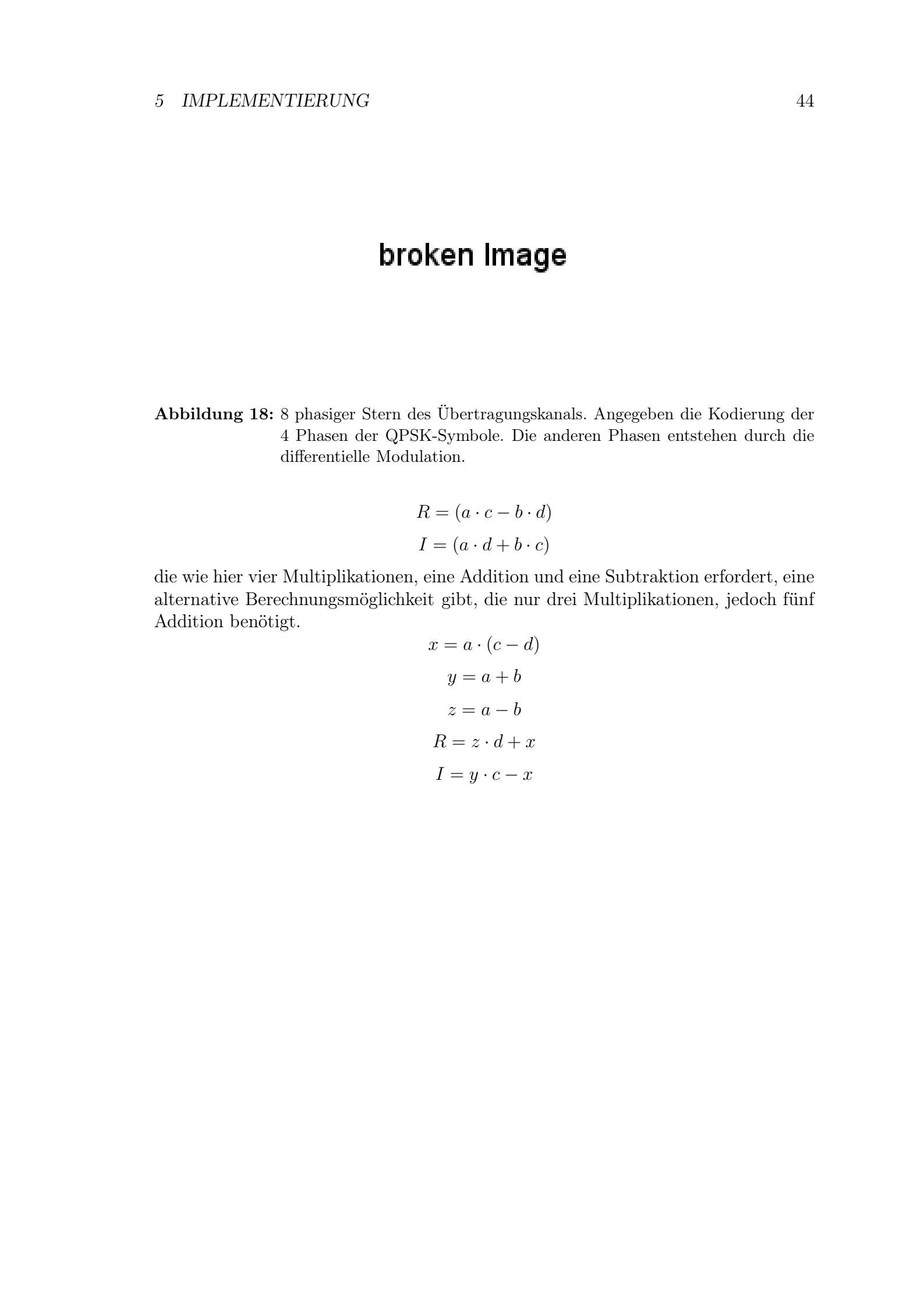

Abbildung 18: 8 phasiger Stern des Übertragungskanals. Angegeben die Kodierung der 4 Phasen der QPSK-Symbole. Die anderen Phasen entstehen durch die differentielle Modulation.

$$R = (a \cdot c - b \cdot d)$$
$$I = (a \cdot d + b \cdot c)$$

die wie hier vier Multiplikationen, eine Addition und eine Subtraktion erfordert, eine alternative Berechnungsmöglichkeit gibt, die nur drei Multiplikationen, jedoch fünf Addition benötigt.

$$x = a \cdot (c - d)$$
$$y = a + b$$
$$z = a - b$$
$$R = z \cdot d + x$$
$$I = y \cdot c - x$$

6 Ergebnisse

6.1 Test des Systems

6.1.1 Gründe für die Entwicklung eines Servers

Eines der Hauptprobleme moderner Softwareentwicklung ist die Validierung derselben. Für den COFDM-Modulator stehen dafür bestehende Lösungen in der MATLAB Programmierumgebung bereit, deren Berechnungen bloß mit den Ausgaben des neuen Modulators verglichen werden müssen. Natürlich lassen sich die Ausgaben beider Programme in eine Datei umleiten und anschließend miteinander vergleichen. Ein nicht so hohes Datenaufkommen und eine damit auch höhere Geschwindigkeit erreicht eine direkte Anbindung des C-Modulators an die Programmiersprache von MATLAB. Diese unterstützt jedoch nur einzelne C Funktionsaufrufe. Da der Modulator im Prinzip nur ein einfacher Automat ist, kann er natürlich in einzelne Funktionsaufrufe unterteilt werden. Dabei müssen aber alle Zustandsvariablen, die veränderlich sind, zurückgegeben und beim nächsten Aufruf wieder eingelesen werden. Das verkompliziert die Prozedur und macht sie vor allen Dingen von der spezifischen Implementation der Funktionen abhängig. Um dennoch eine Verbindung beider Modulatorprogramme zu erreichen, ist eine Client-Server Lösung entwickelt worden. Die Daten werden dabei über das Filesystem übertragen. Jedoch lassen sich die beiden Programme einfrieren oder in einen beliebigen Zustand versetzen.

Die Basisfunktionen, die den Kern des COFDM-Modulators darstellen, werden dabei an ein Userinterface herangeführt und so dem Benutzer individuell zur Verfügung gestellt. Die Vorteile liegen auf der Hand. Es ist so möglich, den COFDM-Encoder einzelne Zwischenschritte abarbeiten zu lassen und den jeweiligen Zustand des Encoders zu prüfen. So lassen sich nicht nur die Endergebnisse beider Programme, sondern auch Zwischenergebnisse, wie zum Beispiel das digitale Symbol oder das Symbol im Frequenzbereich, auf ihre Korrektheit hin überprüfen. Ebenfalls kann im Fehlerfall auch gleich der Istzustand der Software überprüft werden, was Hinweise auf Ort und Art des Fehlers gibt.

Insgesamt hat sich der COFDM-Server bei dieser Aufgabe als wertvolles Entwicklungswerkzeug erwiesen, dessen Einsatz den Abgleich beider COFDM-Versionen ungemein erleichtert hat. Auch schwierige Fehler, wie ”dangling pointers”, konnten schrittweise eingekreist werden. Die Zeit für das Debugging der Software hat sich drastisch verringert. Auch die Struktur der Software hat sich verbessert. So kann auf aufwendige Debug-Routinen oder unübersichtliche Debug-Ausgaben im Kerncode des Projektes verzichtet werden. Ausgaben und Debugging stehen sauber getrennt in den Kommandofunktionen des Servers.

Zusammenfassend läßt sich sagen, daß die Entwicklung einer Einzelschrittsteuerung per Kommandozeile für alle Software, die nicht standardmäßig über eine Benutzerschnittstelle verfügt, ein lohnender Schritt ist. Darunter fallen Bibliotheken, Signal- und Datenverarbeitung.

6.1.2 Aufbau des Servers

Basis des Datentransfers vom und zum Server ist das Filesystem des Betriebssystems. Der Server fällt damit nicht unter das Prinzip der Hardwareunabhängigkeit, sondern ist auf die Unterstützung des Betriebssystems angewiesen. Dies ist jedoch nicht nachteilig. Der Server soll ja nicht auf einer beliebigen Zielplattform laufen, sondern ist als Zusatz zu verstehen. Unterstützt werden sowohl Sun wie auch PC. Wichtig ist vor allem, daß das Betriebssystem Pipes unterstützt wird.

Die Ein- und Ausgabe der Daten erfolgt über spezielle Dateien, sogenannte FIFO's. Diese haben den Vorteil, daß gelesene Daten gleich wieder gelöscht werden, so daß der Speicherverbrauch der Dateien gering ist. Der Name stammt von "first in, first out" und beschreibt das Verhalten des FIFO's bei Erhalt und Anforderung von Daten. Die Daten, die den FIFO als erste erreichen, sind auch die ersten, die bei einem Lesezugriff auf den FIFO wieder ausgegeben werden. Sind keine Daten im FIFO vorhanden oder existiert kein Verbraucher, der lesend auf den FIFO zugreift, werden die Prozesse automatisch vom Betriebssystem blockiert. Das Warten erzeugt dabei keine Prozessorlast.

Ein Client greift also schreibend auf die Kommandopipeline zu und teilt hierüber seine Befehle an den Server mit. Diese Kommandopipeline muß dem Client allerdings beim Start namentlich bekannt sein. Man spricht hierbei von einem Well-Known-FIFO. Prinzipiell reicht es aus, wenn sich der Client hierüber beim Server anmeldet und hierbei die Namen der im folgenden verwendetetn Ein- und Ausgabedateien bekanntgibt. So könnten auch unabhängige Clients individuell bedient werden. Der COFDM-Server ist jedoch auf sequentielle Abarbeitung ausgelegt. Wenn mehrere Clients ihre Befehle an den COFDM-Server senden, werden sie vom Betriebssystem sequentialisiert und in dieser Reihenfolge abgearbeitet. Die Ergebnisse hängen dabei von der Summe aller gesendeten Befehle ab. Sind nicht alle Eingaben aufeinander abgestimmt, kann das zu Fehlern führen.

Die Eingaben der Clients werden auf einfache Weise verarbeitet. Beim Parsen des Textes werden die Eingaben in einzelne Worte, sogenannte Tokens, unterteilt. Alle Tokens sind einfache Zeichenfolgen, die durch Leerzeichen voneinander getrennt sind. Die Art des Befehls ist durch das erste Wort bestimmt. Es wird in einer Tabelle gesucht. Anschließend wird in den entsprechenden Programmcode verzweigt. Die Grammatik, die damit möglich ist, entspricht einer L1 Grammatik. Das von links nach rechts jeweils nächste Zeichen bestimmt dabei, welchem Zweig nachgegangen wird. Wird eine Übereinstimmung gefunden, ist der Befehl eindeutig gefunden. Dadurch, daß ein Befehl durch seinen linken Anfang erkannt wird, kann der Rest des Kommandos entfallen, sobald der Befehl eindeutig ist. Bei Mehrdeutigkeiten verzweigt der Server zum ersten passenden Eintrag in seiner Tabelle.

Anschließend werden die Zeichenketten an den Interpreter übergeben. Erst hier wird den Eingaben ein Sinn zugeschrieben. Der Interpreter muß zugleich prüfen, ob Ein-

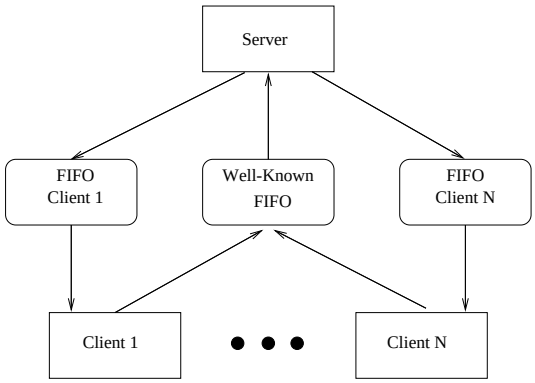

Abbildung 19: Abbildung eines Servers mit ausgezeichnetem FIFO zum Datenempfang und clientspezifischen Rückkanälen

gaben wie Zahlen oder Filenamen in korrekter Syntax stehen und andernfalls eine Fehlermeldung ausgeben. Dadurch, daß die Befehle in einer Tabelle gehalten werden, läßt sich der Interpreter sehr leicht durch neue Kommandos ergänzen. Es müssen nur der Name des Befehls und der dazugehörige Funktionsaufruf in der Tabelle ergänzt werden.

6.2 Performanceanalyse

Bei der Performanceanalyse geht es darum, eine Aussage darüber zu treffen, ob die gegebenen Aufgaben von dem Zielprozessor in der vorgeschriebenen Zeit bearbeitet werden können. Dazu soll zunächst die maximale Last, die dem Prozessor zugemutet wird, spezifiziert werden.

Die größte Struktur, die dem DAB-Signal inne ist, ist der DAB-Rahmen. Es soll die Konfiguration gesucht werden, die für einen Rahmen den größten Arbeitsaufwand verursacht. Der Aufwand bei der Kanalkodierung ist dabei nicht vom DAB-Mode abhängig. Der aufwendigste DAB-Mode bestimmt sich damit aus der FFT, die für Mode 1 proportional $Symbolzahl \cdot FFT - Aufwand = 77 \cdot 2048 \cdot 11$ für 96ms Zeitsignal ist. Für Mode 4 muß beispielsweise ein Aufwand proportional $77 \cdot 1024 \cdot 10$ für 48ms Zeitsignal berechnet werden, also Faktor $\frac{10}{11}$ weniger. Mode 1 ist also der zu betrachtende Fall. Für die Kanalkodierung wurde bereits angegeben, daß die minimale Koderate von $8/9$ den größten Aufwand erzeugt. Der entsprechende minimale Fehlerschutz findet sich bei einem Protectionlevel von 5. Der Worst-Case kann jedoch auch aus einem beliebigen Profil interpoliert werden.

Der DAB-Rahmen des Mode 1 besteht aus 77 Symbolen: Einem Nullsymbol, einem TFPR-Symbol, 3 Symbolen des "Fast information Channels" (FIC) und 72 Symbolen des "Main Service Channels" (MSC). Weiterhin sind für jeden ETI-Rahmen die CRC-Checks einzurechnen. In Mode 1 sind 4 ETI-Rahmen zur Erzeugung des DAB-Rahmens nötig. Jeder besitzt einen Header-CRC und einen Body-CRC. Zusammen

sind also 8 CRC-Berechnungen für Mode 1 nötig. Dazu kommt eine eventuelle Rekonfiguration, deren Auswirkungen vom Modul <time config> berechnet werden. Insgesamt muß dieses Modul für jeden ETI-Rahmen einmal aufgerufen werden, jedoch wird sie ohne vorliegende Rekonfiguration nach wenigen Zyklen abgebrochen. Eine Rekonfiguration liegt technisch jedesmal beim Start des Programms vor.

Für den Nachweis der Funktionstüchtigkeit bedarf es nicht einer vollständigen Anpassung an das netzwerkabhängige ETI-Format. Zur Messung der Geschwindigkeit reicht es aus, den Reed-Solomon-Dekoder zu benchmarken, da er der mit Abstand aufwendigste Prozeß bei der Netzwerkanpassung (NA) ist. Die vollständige Anbindung des ETI-NA Protokolls ist im Rahmen der Arbeit nicht mehr erfolgt. Die Geschwindigkeit des RS-Kode ist jedoch auch außerhalb des COFDM-Modulators meßbar und kann auf einen DAB-Rahmen hochgerechnet werden. In der Tabelle ist der Reed-Solomon-Dekoder und die von ihm aufgerufene Routine <berlekamp>, die zum Finden des Fehlerpolynoms benutzt wird, enthalten. Für das Profilen des

Algorithmus	calls	Zyklen/call	Zyklen
<overhead>	1	235	235
<berlekamp>	1	17990	17990
<rs correct>	1	269366	269366

Tabelle 1: Profiling Tabelle mit den Zyklen zur Dekodierung eines Reed-Solomon-Codes der Länge 240 mit 14 Paritybytes und 7 fehlerhaften Bytes

Programms ist ein eigenes Modul in der COFDM-Bibliothek vorgesehen. Dabei muß berücksichtigt werden, daß der Aufruf der Funktionen und die Abfrage der CPU-Zyklen selber Rechenzeit benötigt. Die ungefähre Größe dieses Overheads wird durch die <overhead>-Zeile im Benchmarkprofil angegeben. Sie ist gering, so daß sie prinzipiell vernachlässigt werden kann. Zu beachten ist jedoch, daß die Funktionen verschachtelt sein können. So erhöht sich der Fehler für die aufrufende Funktion proportional zur Anzahl der <calls>.

Interessiert man sich jedoch trotzdem für die genaue Zyklenzahl einer Funktion, kann man die Messung durch Herausrechnen des Overheads verfeinern. Jeder Aufruf (call) einer Funktion, der in der Tabelle angegeben ist, verursacht einmal den in <overhead> angegebenen Zyklenfehler. Der Overhead ergibt sich aus zwei Aufeinanderfolgenden Profilingaufrufen, zwischen denen kein weiterer Programmcode steht. Prinzipiell ist diese eine Messung nicht ausreichend, um den verursachten Fehler genau herauszurechenen. Es müßte auch der durch einen verschachtelten Aufruf der Profilingfunktionen verursachte Fehler gemessen werden. Für die Angabe einer Größenordnung reicht es jedoch aus. Der wirkliche Fehler kann doppelt so groß sein, wie der geschätzte. Die korrigierte Messung wird also immer noch zu viele Zyklen anzeigen.

Die Messung der Performance wurde zunächst auf den Entwicklungsplattformen Sun

Algorithmus	Zyklen
Radix-2	20815
Radix-4	13228

Tabelle 2: Angabe der Zyklenzahlen für 1024 Punkte FFT's von TI (ohne reversal, 16 Bit integer)

und Linux durchgeführt. Dort zeigte sich vor allem die Relation der Geschwindigkeiten von den Funktionen zueinander. Da die Simulationszeit für 10 Symbole DAB-Signal bei etwa 8 Stunden lag, fanden Optimierungen und Debugging ausschließlich auf den Entwicklungsplattformen statt. Optimierungen wurden dabei vornehmlich auf algorithmischer Ebene durchgeführt. Es ist sicher noch Spielraum für Verbesserungen vorhanden. Dies bezieht sich zum einen auf die Betrachtung der Befehlsebene von C, zum anderen könnte man direkt für die Zielplattform optimieren und dabei auf Assemblerebene wechseln.

Der erzeugte Assemblercode ist einsehbar. Es zeigt sich, daß der Compiler gut optimierten sequentiellen Code erzeugt. Der Parallelitätsgrad ist jedoch durchgehend gering und liegt bei ca. 1,5 Befehlen pro Zyklus. Der maximale Parallelitätsgrad der Zielplattform liegt bei 8. Bei handoptimiertem Assemblercode liegt der Parallelitätsgrad bei bis zu 6,3 Befehlen, wie sich aus Beispielen von Texas Instruments ablesen läßt. Daraus läßt sich absehen, daß ein theoretischer Speed-Up mit dem Faktor 4 durch eine bessere Ausnutzung des Prozessors noch möglich ist. In der Praxis wird man nur die innersten Schleifen und die aufwendigsten Algorithmen per Hand optimieren. Aus den Benchmarks ergibt sich, daß auch hier noch ein Speed-Up von Faktor 3 zu erwarten ist.

Auffällig bei den Benchmarks ist die herausragende Position der Fouriertransformation. Sie zeigt sich auch nach allen Verbesserungsversuchen als bestimmendes Element der Performance des gesamten Algorithmus. Es wurden deshalb auch mehrere Versuche unternommen, diese Funktion zu optimieren. Die FFT ist jedoch aus algorithmischer Sicht eine tief geschachtelte, relativ kurze, mathematische Funktion. Versuche, die Berechnungszeit auf dieser Ebene zu beschleunigen, erreichen einen Speed-Up von beispielsweise Faktor 1,5 für den Einsatz einer Radix-4 FFT. Wie bereits beschrieben, läßt sich ein wesentlich größerer Speed-Up durch Parallelisierung und spezielle Hardwareanpassung auf Assemblerebene erreichen. Um eine Abschätzung für eine handoptimierte FFT-Routine angeben zu können, wurden Beispiele von Texas-Instruments verwendet.

Texas Instruments hat selber hochoptimierten Assemblercode für eine FFT zur Verfügung gestellt. Mit diesem läßt sich abschätzen, wieviel Zeit für die FFT auf dieser speziellen Plattform veranschlagt werden muß. Die Zyklenzahlen wurden für die Radix-4 FFT überprüft. Eine Fehlermessung ist jedoch nicht erfolgt. Für die 2048 Punkte FFT des Mode 1 läßt sich eine minimale Zyklenzahl für einen Mix-Radix-

Algorithmus	calls	Zyklen/call	Zyklen
<overhead>	1	241	241
<conv_punct>	19	10942	207905
<ifft>	11	683811	7521921
<mst symbol>	7	85897	601281
<fic symbol>	3	61236	183708
<interleaver>	9	20226	182038
<time config>	1	110679	110679
<symbols complete>	11	1085831	11944141
<crc_calc>	8	9229	73834
<dqpsk>	10	36438	364380
<fft reversal>	11	289116	3180276

Tabelle 3: Tabelle mit den Zyklenzahlen, wie sie von dem Profilingmodul der COFDM-Bibliothek ausgegeben werden.

Algorithmus von ca. 34000 Zyklen veranschlagen. Nimmt man den den Radix-2 Algorithmus als Basis, sind etwa 40000 Zyklen zu veranschlagen. Die innerste Schleife des Algorithmus hat eine Länge von 4 Zyklen. Sollten Veränderungen am Algorithmus nötig sein, ist eine eventuelle Verdopplung der Zyklenzahl nicht auszuschließen. In diesem Fall läßt sich die Zyklenzahl mit 80000 Zyklen angeben. Zusammenfassend läßt sich sagen, daß für eine Fouriertransformation im Mode 1 nicht mehr als 100000Zyklen veranschlagt werden müssen. Mit dieser Aussage läßt sich das Ergebnis des Benchmarks neu bewerten.

Anhand der Benchmarks läßt sich ausrechnen, daß für den Algorithmus ohne FFT etwa 100000 Zyklen im Worst-Case benötigt werden. Zusammen mit der FFT werden also 200000 Zyklen benötigt, was einer Zeit von 1 ms für ein Worst-Case Symbol entspricht. Das entspricht hochgerechnet 77ms für die Berechnung aller 77 Symbole eines DAB-Rahmens. Es darf also gesagt werden, daß es technisch möglich ist, den COFDM-Encoder auf einem TMS320C6201 in Echtzeit zu implementieren! Die verbleibende Zeit von maximal 19ms reicht jedoch nach dem jetzigen Stand der Arbeit nicht mehr für den Reed-Solomon-Dekoder aus. Mit den Aussagen über die verbliebenen Optimierungsmöglichkeiten darf man jedoch optimistisch sein, daß nach einer Beschleunigung der Kernroutinen auch diesem genug Zeit zur Verfügung gestellt werden kann.

In der Tabelle 3 sind die wichtigsten Funktionen mit ihren Zyklenzahlen angegeben.

In Kapitel 5.10 wurde darauf hingewiesen, daß sich das FFT-Reversal unter bestimmten Bedingungen mit dem Frequenzinterleaver zusammenfassen läßt. Dadurch kann diese Funktion völlig entfallen. Es wird davon ausgegangen, daß diese Zusammenfassung stattfindet, sobald ein endgültiger FFT-Algorithmus gefunden worden

ist. Diese Funktion wurde deshalb nicht hinsichtlich der Geschwindigkeit optimiert.

Für die Zyklen eines DAB-Rahmens in Mode 1 sind zu rechnen: 3 FIC-Symbole, 72 MST-Symbole, ein Null- und ein TFPR-Symbol, entsprechend 77 IFFT's, 4 CRC's für die ETI-Header, 4 CRC's für den ETI-Body, eine Rekonfiguration und insgesammt 36 Reed-Solomon-Codes. Besonders IFFT und RS-Code verbrauchen die meiste Rechenzeit. Die Benchmarks sind mit den geschwindigkeitsoptimierten Algorithmen auf dem Simulator durchgeführt worden. Verbesserungen können sich noch durch die nächste Generation des Compilers ergeben.

6.3 Speicheranalyse

Modul	Speicherplatz (Bytes)
Convolutional Coder	2240
Protectionlevel Tabellen	750
CRC	512
IFFT	10240
ETI-Demuxer	512
TFPR-Symbol Generator	218
TII-Symbol Generator	76
OFDM Generator	5120
Reed-Solomon	512
Symbol Control	408
Time Interleaver	52096
Time Configuration	3460
Time Symbol	21248
4x ETI	24576
printtools*	200
profile*	1000
Größe des EXE-Files	201080

Tabelle 4: Tabelle mit dem Speicherverbrauch der einzelnen Module für den TMS320C6201

Die Algorithmen sind überwiegend auf Rechenzeit optimiert. Der von den Algorithmen benötigte Speicher ist in Tabelle 4 wiedergegeben. Der Speicherplatz ist dabei jedesmal auf den größten Bedarf im Mode 1 ausgelegt. Dadurch wird im gesammten Programm keine dynamische Speichervergabe benötigt. Die Algorithmen lassen sich auch hinsichtlich des Speicherverbrauchs optimieren, es lassen sich jedoch nur geringe Mengen sparen. Das Gros des Speichers benötigt der Time Interleaver. Dieser Speicher läßt sich nicht weiter reduzieren. Das gleiche gilt für den Platz, den das Symbol im Zeitbereich benötigt (Time Symbol) und den Platz für die ETI-Eingabedaten. Der IFFT-Algorithmus benötigt für die Tabelle mit den Twidd-

lefaktoren den nächstgrößeren Speicherblock. In der Tabelle sind 5/4 der Periode einer Sinusschwingung abgespeichert. Aufgrund der enthaltenen Symmetrien läßt sich hier der Speicher prinzipiell noch auf 1/4 reduzieren. Die IFFT ist jedoch auch einer der aufwendigsten Algorithmen. Zusätzliche Operationen können hier die Rechenzeit empfindlich in die Höhe treiben. Alle anderen Module benötigen dazu nur geringe Mengen an Speicher. Es lassen sich zusammen ungefähr 10KByte einsparen. In der Größe des Executables sind Heap, Stack, Code und Daten mit berücksichtigt. An der Rechnung fehlt allerdings noch ein Speicherbereich für I/O-Buffering, der für die Benchmarks nicht benötigt wurde. Ein Speed-Up der Algorithmen ist erst möglich, wenn deutlich mehr Speicher zur Verfügung gestellt wird.

Der TMS320C6201 ist mit 2x64KByte Speicher ausgerüstet, einmal für den Code, das andere mal für Daten. Dieser Speicher reicht nicht für den COFDM-Modulator aus. In dieser Version muß der Prozessor auf externen Speicher zurückgreifen, der jedoch zusätzlich Wartezyklen erzeugt. Es sind jedoch auch Versionen des Prozessors geplant, die mehr Speicherplatz besitzen. Mit der doppelten Menge an Speicher könnte man auskommen.

7 Zusammenfassung/Ausblick

Im Rahmen der Diplomarbeit ist eine COFDM-Modulatorsoftware in der Programmiersprache C entwickelt worden. Die grundsätzliche Funktion wurde durch den Vergleich mit anderen Lösungen gezeigt. Auf vollständige Fehlerprotokolle wurde jedoch verzichtet, da dies nicht Hauptziel der weiteren Entwicklung sein sollte.

Bei der Erstellung der Software wurde Wert auf Modularisierung gelegt. Einzelne Komponenten sind leicht austauschbar. Im Rahmen der Optimierung sind Module neu geschrieben worden. Hierbei wurde das Konzept getestet, ob es die Anforderung der dynamischen Entwicklung erfüllt. Weiterhin wurden hardwareabhängige FFT-Module eingebunden, um den nächsten Schritt zur Anpassung an eine spezielle Hardware zu demonstrieren.

Für die Ausgangsdaten ist eine Genauigkeit von 12 Bit seitens Bosch gefordert worden. Daraus ergibt sich die Wortbreite, mit der die Algorithmen berechnet werden müssen. In der Arbeit ist gezeigt worden, daß 15 Bit Genauigkeit zur Berechnung der FFT ausreichend sind.

Die erstellten Algorithmen sind auf die Zielplattform des TMS320C6201 übertragen worden. Dort wurden Untersuchungen zur Laufzeit und zum Speicherverbrauch durchgeführt. Hinsichtlich der Geschwindigkeit läßt sich sagen, daß die Kernroutinen des COFDM-Modulators in Echtzeit ausgeführt werden können. Dazu bedarf es lediglich einer Anpassung der FFT in Assembler. Um noch zusätzliche Aufgaben, wie die Netzwerkadaption zu bewältigen, müssen weitere Teile des Programms durch Assemblerroutinen ersetzt werden.

Um aus dem Konzept ein voll einsetzbares Produkt zu entwickeln, müssen die noch unvollständigen Module für die Netzwerkadaption ergänzt werden. Ebenfalls soll noch eine lineare Vorverzerrung im Frequenzbereich implementiert werden. Der Assembler-FFT-Algorithmus muß in einen “dezimation in time”-Algorithmus umgeschrieben und an das Blockgleitkommaverfahren angepaßt werden. Letztendlich müssen I/O-Routinen für die Ein- und Ausgabe und Synchronisation auf der Zielplattform erstellt werden.

Literatur

[1] *Radio broadcast systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers.* European Telecommunication Standard Institute ETS 300 401, November 1994

[2] *DAB distribution, Ensemble Transport Interface.* European Telecommunication Standard Institute ETS 300 799, 1995

[3] Thomas Lauterbach: *Digital Audio Broadcasting – Grundlagen, Anwendungen und Einführung von DAB.* Feldkirchen: Franzis, 1996

[4] Marc Schrader: *Softwaremodell des OFDM-Modulators für einen DAB-Prototypsender.* Diplomarbeit, Hannover, 1994

[5] C.S. Burrus, T.W. Parks: *DFT, FFT and Convolution Algorithms* . Wiley and Sons, 1985

[6] E. Oran Brigham: *FFT: schnelle Fourier-Transformation* Oldenbourg, 1995

[7] Oppenheim, Schafer: *Zeitdiskrete Signalverarbeitung* R. Olderbourg Verlag, 1995

[8] Alan Burns, Andy Wellings: *Real-Time Systems and Programming Languages.* Addison Wesley, 1996

[9] *TMS320C6xx Assembly Language Tools, User's Guide.* Handbuch, Texas Instruments, 1997

[10] *TMS320C6xx Optimizing C Compiler, User's Guide.* Handbuch, Texas Instruments, 1997

[11] *TMS320C6xx Programmer's Guide.* Handbuch, Texas Instruments, 1997

[12] *Evaluation of the performance of the C6201 Processor & Compiler.* Loughborough Sound Images, Draft 1996

[13] Robert Matusiak: *Implementung Fast Fourier Transform Algorithms of Real-Valued Sequences with the TMS320 DSP Family.* Application Report, 1997

[14] Rolf Veith: *Softwaremodell der Modulationszuführung zu einem DAB-Gleichwellensender.* Diplomarbeit, 1995

[15] Volker Eberlein: *Studie zur Echtzeit-Realisierung eines DAB-Ensemble-Multiplexers auf einer UNIX-Plattform.* Diplomarbeit, 1996

[16] Stephen B. Wicker: *Error Control Systems, Communication and Storage.* Prentcie Hall, 1995

[17] Tenkasi V. Rambadran, Sunil S. Gaitonde: *Tutorial on CRC Computations.* IEEE Micro, 1988 vol.8

[18] Guy Gastagnoli, Jürg Ganz, Patrick Graber: *Optimum Cyclic Redundancy Check Codes with 16-Bit Redundancy*. IEEE Transactions on Communications, 1990 vol.38

[19] Richard Stevens: *Advanced Programming in the UNIX Environment*. Addison Wesley, 1992

[20] H.G. Musmann: *Einführung in die Kanalkodierung*. Skript des Instituts für Nachrichtentechnik und Informationsverarbeitung, Hannover 1996

[21] Marc Schrader: *Structured System Analysis COFDM-Encoder* Robert-Bosch Multimedia Systeme, Hildesheim, 1996

[22] Ian Sommerville: *Software Engeneering*. Addison Wesley, 1995

[23] Edward Yourdon *Moderne strukturierte Analyse* Prentice Hall, 1992