

Studienarbeit



Von:

Mario Jurcevic

Markus Gloor

Betreuung:

Prof. Dr. Guido M. Schuster



Echtzeit Wasserzeichensystem für Sprache

Rapperswil, 11. Juli 2003

Abstract

Lieber Leser, in diesem Dokument wollen wir Ihnen ein Verfahren zum Übertragen eines Wasserzeichens über einen Telefonkanal näher bringen. Unter einem Wasserzeichen kann man sich eine digitale Codesequenz, also eine bestimmte Folge von Nullen und Einer vorstellen.

Der Sender mischt in Echtzeit dem Sprachsignal ein unhörbares Wasserzeichen bei. Weil das Wasserzeichen mit sehr kleiner Leistung beigegeben wird, kann man es nicht hören. Dieses Mischsignal wird nun über einen Telefonkanal gesendet. Das Wasserzeichen kann durch unser Verfahren am anderen Ende der Leitung im Empfänger wieder sichtbar gemacht werden.

Diese Grafik zeigt die grobe Übersicht wie ein Sprachkanal mit Wasserzeichen funktioniert.

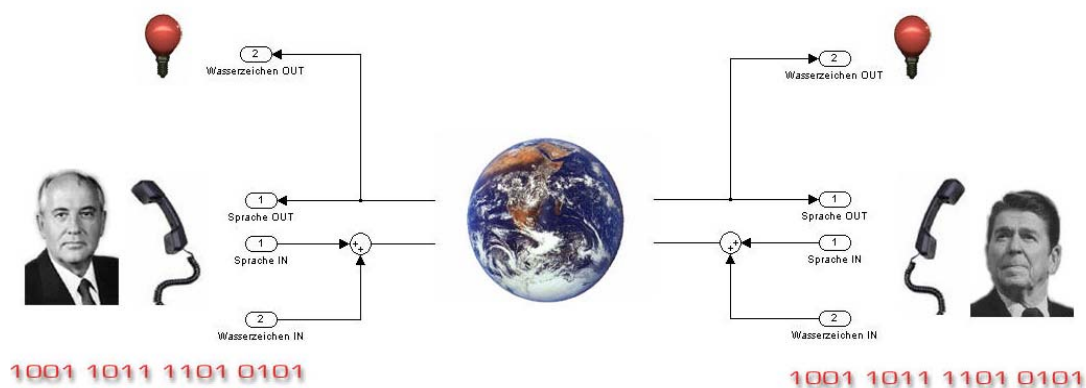


Abbildung 1-1: Übersicht Wasserzeichen

Was für Vorteile bringt dieses Wasserzeichen? Wie bei einer Geldnote kann man die Echtheit feststellen. Bei einem Ferngespräch ist man nun sicher, dass wirklich die richtige Person am anderen Ende der Leitung spricht. Falls das Wasserzeichen korrekt eingefügt wurde, kann der Empfänger davon ausgehen, dass es der Richtige Sender war. Eine Unsicherheit bleibt aber noch. Das gemeinsame Geheimnis wie das Wasserzeichen eingefügt wird und welche Codesequenz im Wasserzeichen verborgen ist, darf niemand anderer wissen.

Unsere zentrale Aufgabe besteht darin, ein geeignetes Wasserzeichenverfahren zu evaluieren und es mit Matlab auf Robustheit, Störanfälligkeit und Genauigkeit zu testen. In einem späteren Schritt werden wir das System auf einem Digital Signal Processing Board mit C / C++ in Echtzeit realisieren. In Echtzeit heisst, dass das Sprachsignal kaum verzögert wird und sich die ganze Kommunikation für den Hörer normal anhört.

Diese Semesterarbeit handelt über ein wichtiges Thema, ist Sicherheit in unserer Gesellschaft doch ein grosses Bedürfnis. Durch diese Arbeit erhält der Leser aber auch einen kleinen Einblick in digitale Signalverarbeitung. Wir haben uns auf zwei Verfahren beschränkt. Wie funktionieren sie und welches ist der bessere unserer möglichen Systemvorschläge? Entsprechen sie den Anforderungen? Nach diesem Bericht wissen Sie mehr.

Inhaltsverzeichnis

Kapitel 1: Einleitung	5
Kapitel 2: Pflichtenheft	7
Kapitel 3: Projektplanung	8
3.1 Zeiteinteilung	8
Kapitel 4: Wasserzeichenverfahren	9
4.1 Was gibt es für Verfahren	9
4.1.1 Notch Filter Technik	9
4.1.2 LSB Technik	10
4.1.3 Spread Spectrum	10
Kapitel 5: Das Frequenzverfahren	11
5.1 Funktionsweise	11
5.2 Einbetten des Wasserzeichen im Sender	11
5.3 Detektion des Wasserzeichens im Empfänger	12
5.3.1 Synchronisation	13
5.3.2 Bitrate und Detektionszeit	13
5.4 Auswertung des Frequenzverfahrens	14
5.4.1 Grundsätzliche Einstellungen und Voraussetzungen	14
Kapitel 6: Das Echoverfahren	18
6.1 Funktionsweise	18
6.1.1 Mathematische Separierung	19
6.1.2 Beispiel zur Cepstral Analyse	21
6.2 Einfügen des Wasserzeichens im Sender	23
6.3 Detektion des Wasserzeichens im Empfänger	24
6.3.1 Synchronisation	25
6.4 Auswertung des Echoverfahren	25
6.4.1 Bitrate und Detektionszeit	25
6.4.2 Messungen	26
Kapitel 7: Vergleich Frequenzverfahren – Echoverfahren	27
7.1 Vor- und Nachteile	27
7.1.1 Des Frequenzverfahren	27
7.1.2 Des Echoverfahren	27
Kapitel 8: Implementation	28
8.1 TMS320C6711 Entwicklungsumgebung	28
8.1.1 Hardware Development System Kit DSK	28
8.1.2 Code Composer Studio IDE	29
8.1.3 DSP Real Time Operating System (RTOS)	29
8.2 Programme	30
8.2.1 Der Sender	31
8.2.2 Der Empfänger	31
8.2.3 Interruptkonfiguration	32
Kapitel 9: Stand der Dinge	34
9.1 Allgemein	34
9.2 Das funktioniert	34
9.3 Bemerkungen	34

Kapitel 10: Schlussfolgerungen	36
10.1 Kritischer Rückblick	36
10.2 Verbesserungen	36
10.3 Danksagung	37
Kapitel 11: Literaturverzeichnis	38
11.1 Bücher	38
11.2 PDF-Files	38
Kapitel 12: Anhang	39
12.1 Abbildungsverzeichnis	39
12.2 Glossary	40

Kapitel 1: Einleitung

Das vorliegende Dokument erklärt wie unsere zwei Wasserzeichensysteme funktionieren und wie sie in Matlab und C / C++ implementiert werden. Hier ist der grobe Aufbau der Dokumentation.

- 1 Original der **Aufgabenstellung**
- 2 Das **Pflichtenheft** beschreibt unsere Ziele und das Vorgehen bei der Entwicklung und Realisierung unseres echtzeit Wasserzeichensystems.
- 3 Die **Projektplanung** umfasst die Zeit- und Arbeitsaufteilung.
- 4 **Analyse:** Wir erarbeiten die erforderlichen theoretischen Grundlagen und stellen sie dar. Mit Matlab testen wir die Robustheit der zwei Systeme und werten sie aus. Ausserdem wollen wir einen Vergleich der zwei Systeme machen.
- 5 **Design:** In einem weiteren Schritt werden wir die getesteten Lösungen von Matlab in C / C++ übersetzen. Dazu gehört auch eine kleine Einführung in Code Composer Studio für die Programmierung des DSP.
- 6 Im Kapitel **Stand der Dinge** werden evtl. Fehlerquellen des aktuellen Systems beschrieben. Es wird aufgezeigt was bis zuletzt funktionierte.
- 7 **Schlussfolgerungen:** Unsere Erkenntnisse und Erfahrungen bei der Entwicklung des echtzeit Wasserzeichensystems, sowie Verbesserungs- und Erweiterungsvorschläge sind in diesem Kapitel niedergeschrieben. Natürlich darf auch der Dank an unsere Unterstützer nicht fehlen.
- 8 **Literaturverzeichnis:** Beinhaltet nicht nur die von uns verwendeten Bücher und Unterlagen, sondern auch eine Auflistung der wichtigsten pdf Files.
- 9 Zuletzt folgt der **Anhang** mit allen Abhandlungen, Hilfsunterlagen und Datenblätter die wir benutzt haben. Hier ist auch das Abbildungsverzeichnis und ein Glossar mit den verwendeten Abkürzungen zu finden.

Thema: *Echtzeit Wasserzeichensystem für Sprache*

Studenten: Mario Jurcevic und Markus Gloor

Betreuer: Prof. Dr. Guido M. Schuster

Kurzbeschreibung

Das Ziel dieser SA ist es ein robustes Echtzeit Wasserzeichensystem für Sprache auf zwei TI 6711 DSKs zu implementieren. Dem Sender DSK wird ein Sprachsignal zugeführt, welches dann mittels eines zu entwickelnden Algorithmus mit einem unhörbaren Wasserzeichen versehen und ausgespielt wird. Das ausgespielte Signal wird dann dem zweiten DSK zugeführt, welcher versucht das Wasserzeichen in Echtzeit zu finden.

Aufgabenstellung

- Erarbeitung und Darstellung der erforderlichen theoretischen Grundlagen
- Entwicklung eines robusten Wasserzeichen Algorithmus in Matlab/Simulink
- Implementation des kompletten Systems auf der TI Plattform

Erwartete Ergebnisse

- Dokumentation des Problems und der Lösung(en)
- Matlab/Simulink Programm
- C Programme für DSP Sender und Empfänger
- Dokumentation der Implementierungen

Arbeitsweise

- Sie führen ein persönliches Laborbuch, wo Sie aufschreiben **wann** Sie **was** für **wie lange** machen und was die Ergebnisse sind
- Sie schicken mir vor jedem Treffen eine Agenda und nach dem Treffen ein Protokoll

Kapitel 2: Pflichtenheft

1. Erarbeiten des Gesamtkonzeptes gemäss Aufgabenstellung (Analyse):
 - a. Abklären, welche Möglichkeiten es zur Einbettung eines Wasserzeichen gibt.
 - b. Evaluation eines geeigneten Verfahrens. Wichtige Anforderungen sind Robustheit und Geschwindigkeit. Zwei Verfahren umsetzen.
 - c. Das eingebettete Wasserzeichen soll nicht hörbar sein.
 - d. Die ausgewählten Verfahren in Matlab nachbauen, testen und auswerten. Vergleich der zwei ausgewählten Systeme.
2. Implementation (Design):
 - a. Die Lösung aus Matlab in C / C++ umsetzen.
 - b. Eine Kommunikationsstrecke nachbauen und testen.
3. Technische Pflichten:
 - a. Die Lösung soll auf dem TMS320 DSP Board in Realtime funktionieren.
4. Arbeitsweise
 - a. Step by Step
 - b. Laborbuch gemäss Aufgabenstellung
5. Dokumentation:
 - a. Keine Einschränkungen, sehr frei!

Kapitel 3: Projektplanung

3.1 Zeiteinteilung

Tabelle 3-1: Übersicht Zeitplan

Bezeichnung		2003																	
		KW12	KW13	KW14	KW15	KW16	KW17	KW18	KW19	KW20	KW21	KW22	KW23	KW24	KW25	KW26	KW27	KW28	KW29
		17.03.	24.03.	31.03.	07.04.	14.04.	21.04.	28.04.	05.06.	12.05.	19.05.	26.05.	02.06.	09.06.	16.06.	23.06.	30.06.	07.07.	14.07.
Analyse	Systeme suchen und auswählen	■	■	■	■														
	System in Matlab implementieren				■	■	■	■	■	■									
	Auswerten und vergleichen							■	■	■	■								
Design	Umgang Code Composer üben									■	■	■	■	■	■				
	C/C++ Code implementieren											■	■	■	■	■	■	■	■
Doku	Schreiben																	■	■

Legende:

- Ungefähr geplante Arbeiten
- Effektiv ausgeführte Arbeiten

Kapitel 4: Wasserzeichenverfahren

In diesem Kapitel stellen wir einige mögliche Wasserzeichenverfahren vor. Wir haben uns für zwei dieser Verfahren entschieden und wollen später näher auf sie eingehen.

4.1 Was gibt es für Verfahren

Möglichkeiten bei Audiosignalen:

- Notch Filter Technik
- LSB-Techniken
- Spread Spectrum
- Frequenzverfahren
- Echo Markierung
- Kombinationen dieser Verfahren

Es gibt noch andere Verfahren auf die wir hier nicht weiter eingehen wollen.

4.1.1 Notch Filter Technik

Das Signal wird an bestimmten Stellen im Spektrum mit Kerbfiltern (Notchfilter) bearbeitet. Dabei müssen diese sehr schmalbandig sein.

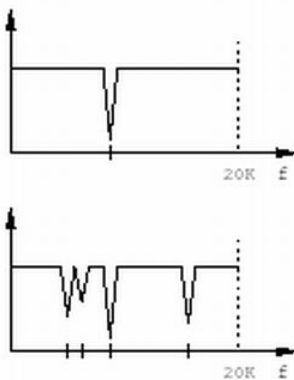


Abbildung 4-1: Signal nach Notchfilter

Wenn man die Filter schmal genug macht, kann man die Veränderung nicht wahrnehmen. Der Empfänger braucht nur das Spektrum des Signals anzusehen und erkennt sofort ob ein Wasserzeichen enthalten ist. Man kann das Wasserzeichen leicht entfernen. Deshalb ist dieses Wasserzeichen unsicher.

4.1.2 LSB Technik

Das Wasserzeichen wird im unwichtigsten letzten Bit der Nutzdaten kodiert. Das führt zu einem Qualitätsverlust, der vor allem bei leisen Passagen hörbar sein kann

```
sample[t] = sample[t] & 0xfffe  
+ mark[t] & 0x0001;
```

Dieses Verfahren ist anfällig gegen Rauschen zum Beispiel durch DA-AD Wandlung oder in unserem Fall ein Telefonkanal. Dafür ist es leicht zu realisieren und man kann eine hohe Bitrate des Wasserzeichens erwarten.

4.1.3 Spread Spectrum

Das Nutzsignal wird auf mehrere Frequenzbänder aufmoduliert. Dabei werden die Frequenzbänder ständig gewechselt. Die Auswahl der Frequenzbänder ist pseudozufällig. Der Sender und Decoder verwenden gleiche Zufallszahlen. Ohne Kenntnis der Zufallszahlen ist das Wasserzeichen nicht detektierbar. Dieses Verfahren ist unempfindlich gegen einfache Angriffe aber sehr empfindlich gegen Timing Veränderungen. Abbildung 4-2 zeigt wie die Frequenzbänder je nach Zeit an andere Orte verschoben werden.

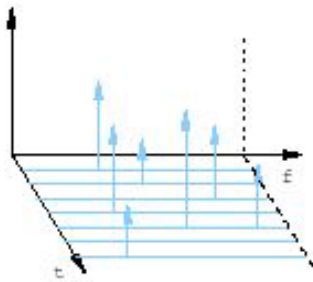


Abbildung 4-2: Spread Spectrum Verfahren

Kapitel 5: Das Frequenzverfahren

Es richtet sich stark nach der Publikation Audio Watermarking for Monitoring and Copy Protection welche im Anhang zu finden ist. Wir haben versucht diese Idee nachzubauen und sind dabei auf einige Probleme gestossen.

5.1 Funktionsweise

Das zentrale Element dieses Verfahrens ist ein Codevektor den sowohl Sender als auch Empfänger kennen. Dabei ist es egal ob Dritten dieser Codevektor auch bekannt ist. Der Sender fügt den Codevektor jedesmal anders zyklisch verschoben im Frequenzbereich des Sprachsignals ein. Wenn nun der Empfänger das Spektrum des empfangenen Sprachsignals mit dem bekannten Codevektor kreuzkorreliert, wird an einer bestimmten Stelle ein Peak auftreten. Diese Stelle entspricht genau der zyklischen Verschiebung des Codevektors des Senders und ist die gewünschte Information.

Die Macht der Kreuzkorrelation wird hier zweimal ausgenutzt. Einmal die Überlagerung zweier unabhängiger, unkorrelierter Signale und die Verzögerung eines Signals wie es im DS Skript auf der Seite 3-14 nachzulesen ist.

5.2 Einbetten des Wasserzeichen im Sender

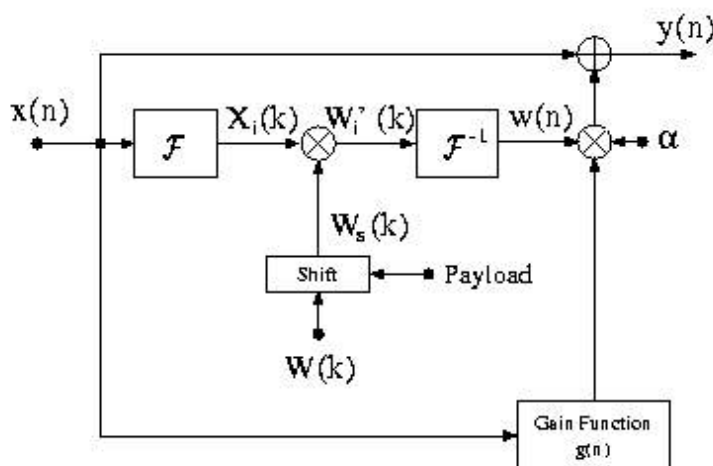


Abbildung 5-1: Einfügen des Wasserzeichen

Als erstes wird das Sprachsignal mit einem analog digital Wandler diskretisiert und in Frames mit einer Länge von 1024 Samples unterteilt. Dieses diskrete Signal $x(n)$ wird nun mit einem FFT in den Frequenzbereich $X_i(k)$ gebracht. Der Codevektor $W(k)$ entspricht einem Vektor mit 1024 Werten mit einem Durchschnitt von 0 und einer Varianz von 1. Ausserdem ist er gerade, damit er im Frequenzbereich keine Imaginärteile aufweist. Verschiebt man nun den Codevektor $W(k)$ zyklisch entsprechend dem Wert der Payload erhält man $W_s(k)$.

Beispiel von $W(k)$ mit 16 Werten. Das würde einem Frame mit 16 Samples entsprechen.

-2	3	4	-1	-6	5	2	-5	-5	2	5	-6	-1	4	3	-2
----	---	---	----	----	---	---	----	----	---	---	----	----	---	---	----

Beispiel von $W_s(k)$. Die zyklische Verschiebung ist hier 3.

-2	3	4	-2	3	4	-1	-6	5	2	-5	-5	2	5	-6	-1
----	---	---	----	---	---	----	----	---	---	----	----	---	---	----	----

$W_s(k)$ wird nun mit $X_i(k)$ multipliziert was zu $W_i'(k)$ führt. Nimmt man $W_i'(k)$ durch einen IFFT zurück in den Zeitbereich erhält man $w(n)$. Gesendet wird das Signal $y(n)$ nach folgender Formel:

$$y(n) = x(n) + a \cdot g(n) \cdot w(n)$$

Wir wählten bei unseren Versuchen den Gewichtungsfaktor a zwischen 0.15 und 0.25. Abhängig von der Leistung des Sprachsignals kann man mit der Funktion $g(n)$ die Gewichtung ebenfalls ändern. Wir haben $g(n)$ nicht benutzt.

5.3 Detektion des Wasserzeichens im Empfänger

Während der Hörer ohne etwas vom Wasserzeichen zu merken das Signal $y(n)$ des Senders hört, wird im Hintergrund gerechnet um das Wasserzeichen zu erkennen. Wir spalten das Signal wieder in Frames mit einer Länge von 1024 Samples auf. Zunächst wird $y(n)$ mit einem Hochpassfilter $F(z)$ gefiltert.

$$F(z) = -1 + 2 \cdot z^{-1} - z^{-2}$$

Dieses gefilterte Signal wird durch einen FFT in den Frequenzbereich gebracht und der Betrag davon genommen was zu $Y_e(k)$ führt.

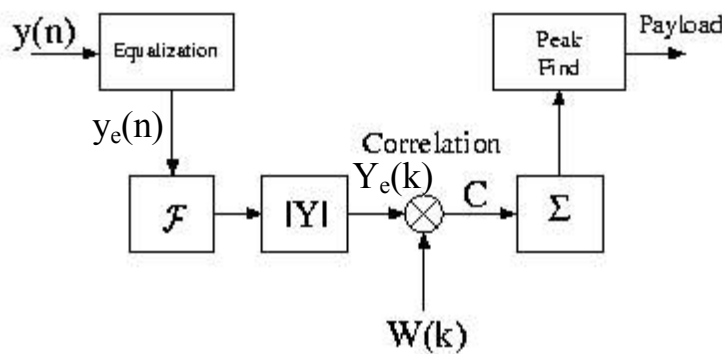


Abbildung 5-2: Detektion des Wasserzeichens

Nun wird $Y_e(k)$ mit dem ursprünglichen Codevektor $W(k)$ kreuzkorreliert. Man kann diese Kreuzkorrelation mittels FFT realisieren. Vorteilhaft ist dabei, dass die Kreuzkorrelation zirkulär wird und so der Peak stärker entsteht.

$$Y_{ef}(k) = fft(Y_e(k))$$

$$W_f(k) = fft(W(k))$$

$$C(i) = ifft(Y_{ef}(k) \cdot W_f(k))$$

An der Stelle wo der Codevektor $W(k)$ und das zu untersuchende Frame am ähnlichsten sind, entsteht in der Kreuzkorrelation ein Peak. Weil der Codevektor aber um die Information verzögert eingefügt worden ist, ist im Kreuzspektrum der Peak nicht in der Mitte. Genau diese Verschiebung gilt es noch zu finden. Das geschieht indem man das Maximum des Vektors $C(i)$ detektiert und danach diese i -te Stelle vom Nullpunkt subtrahiert.

5.3.1 Synchronisation

Bei diesem Verfahren ist erstaunlicherweise keine Synchronisation nötig. Das Wasserzeichen wird im Frequenzbereich eingebettet. Wird nun das gesendete Signal verzögert oder nicht an den gleichen Orten in Frames unterteilt, macht das nichts. Der Betrag des Spektrums bleibt nämlich gleich. Nur die Phase erfährt eine Änderung. Aber da sind ja keine Informationen, weil der Codevektor ja gerade ist und nur Realteile besitzt.

Für eine erhöhte Robustheit wird Redundanz eingefügt. Der Codevektor wird nicht nach jedem Frame zirkulär verschoben, sondern nur nach jedem zehnten. Damit erniedrigt sich aber die Bitrate und auch die Zeit, bis ein Peak detektiert werden kann.

5.3.2 Bitrate und Detektionszeit

Entscheidend für die Bitrate B und die Detektionszeit T ist die Framelänge sowie die Abtastfrequenz und die Anzahl Frames bis ein Codewort detektiert werden kann. Die Codewortlänge n ist abhängig von der Framelänge.

Tabelle 5-1: Daten für Bitratenberechnung

Abtastfrequenz:	f_o	8000 Hz
Framelänge:	l	1024 Samples
Anzahl Frames für ein Codewort:	k	10

$$n = \log_2(l) - 1 = \log_2(1024) - 1 = 9$$

Der Codevektor darf nur von der Stelle 0 bis zur Stelle 512 verschoben werden. Das ist deshalb so, weil das Spektrum symmetrisch wird und zwei Peaks entstehen.

$$T = \frac{1}{f_o} \cdot l \cdot k = \frac{1}{8000} \cdot 1024 \cdot 10 = 1.28s$$

Nach dieser Zeit T können 9 Bits detektiert werden.

$$B = \frac{n}{T} = \frac{9}{1.28} = 7.03 \text{ Bit/s}$$

Eine höhere Bitrate erreicht man, wenn man zum Beispiel nur 5 Frames für ein Codewort benutzt. Dafür entstehen dann mehr Fehler.

5.4 Auswertung des Frequenzverfahrens

5.4.1 Grundsätzliche Einstellungen und Voraussetzungen

Die Auswertung erfolgte mit dem in Matlab vorhandenen `laughter wavfile`. Es wird nicht vorausgesetzt, dass die empfangenen Daten synchron sind. Deshalb wurde beim Testen der Soundvektor bewusst um 1000 Samples verschoben. Zu beachten ist, dass die Rauschquelle zufällig ist, was zu leicht unterschiedlichen Resultaten führen kann.

Bei den Messungen variieren:

- Der Einfügefaktor (wie stark der Codevektor eingemischt wird)
- Die Framelänge
- Der Singnalrauschabstand SNR
- Die Anzahl Frames, die für einen Peak benutzt wurden

5.4.1.1 Messerie 1: Variation des Einfügefaktors

Im File `freqkorr17.m` wird auf Zeile 61 der Einfügefaktor verändert.

Tabelle 5-2: Versuchseinstellungen 1

Matlabfile:	<code>freqkorr17.m, fixcode10x1024.m</code>
Framelänge:	1024 Samples
Einfügefaktor:	0.15
Anzahl Frames für ein Peak:	10
Bitrate:	7.03 Bits pro Sekunde

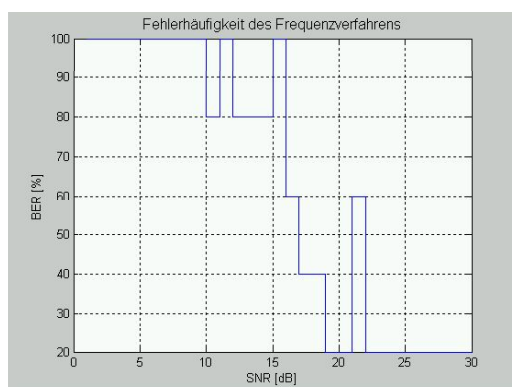


Abbildung 5-3: Fehlerhäufigkeit 1

Tabelle 5-3: Versuchseinstellungen 2

Matlabfile:	freqkorr17.m, fixcode10x1024.m
Framelänge:	1024 Samples
Einfügefaktor:	0.20
Anzahl Frames für ein Peak:	10
Bitrate:	7.03 Bits pro Sekunde

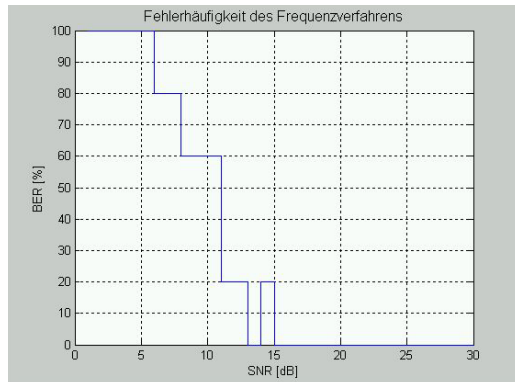


Abbildung 5-4: Fehlerhäufigkeit 2

Tabelle 5-4: Versuchseinstellungen 3

Matlabfile:	freqkorr17.m, fixcode10x1024.m
Framelänge:	1024 Samples
Einfügefaktor:	0.25
Anzahl Frames für ein Peak:	10
Bitrate:	7.03 Bits pro Sekunde

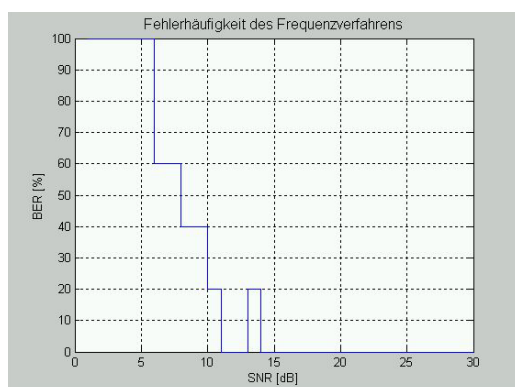


Abbildung 5-5: Fehlerhäufigkeit

5.4.1.2 Messerie 2: Variation der Anzahl Frames für einen Peak

Im File fixcodex1024.m wird auf Zeile 17 der Faktor geändert.

Tabelle 5-5: Versuchseinstellungen:

Matlabfile:	freqkorr17.m, fixcodex1024.m
Framelänge:	1024 Samples
Einfügefaktor:	0.25
Anzahl Frames für ein Peak:	5
Bitrate:	14.06 Bits pro Sekunde

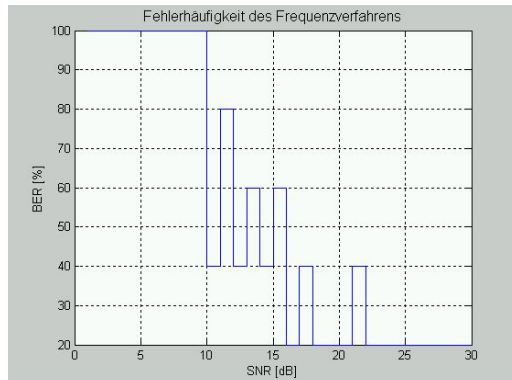


Abbildung 5-6: Fehlerhäufigkeit

Tabelle 5-6: Versuchseinstellungen

Matlabfile:	freqkorr17.m, fixcodex1024.m
Framelänge:	1024 Samples
Einfügefaktor:	0.25
Anzahl Frames für ein Peak:	6
Bitrate:	11.71 Bits pro Sekunde

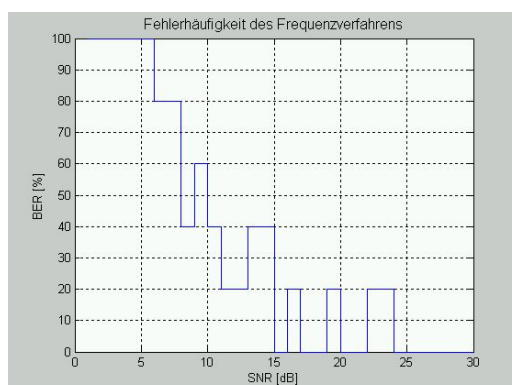


Abbildung 5-7: Fehlerhäufigkeit

Tabelle 5-7: Versuchseinstellungen

Matlabfile:	freqkorr17.m, fixcodex1024.m
Framelänge:	1024 Samples
Einfügefaktor:	0.25
Anzahl Frames für ein Peak:	8
Bitrate:	8.79 Bits pro Sekunde

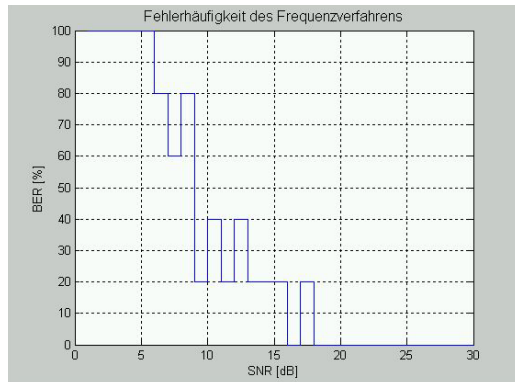


Abbildung 5-8: Fehlerhäufigkeit

Kapitel 6: Das Echoverfahren

6.1 Funktionsweise

Beim Sender wird ein Echo mit einer bestimmten Verzögerungszeit ins Sprachsignal eingefügt. Die Verzögerungszeit entspricht dabei dem Wasserzeichen. Das Echo ist vom menschlichen Gehör nicht wahrnehmbar, weil es mit sehr kleiner Verzögerung eingebettet wird. Der Empfänger muss nun diese Verzögerung des Echos wieder finden. Dies geschieht durch die Cepstrumanalyse des Signals.

Unter dem Begriff Cepstrum versteht man das rücktransformierte, logarithmierte Leistungsdichtespektrum eines Signals. Aufgrund des Echos entsteht im Leistungsdichtespektrum eine zusätzliche periodische Komponente. Die Fourier rücktransformierte des logarithmierten Leistungsdichtespektrum sollte daher ein Peak bei der Echo Laufzeit aufweisen.

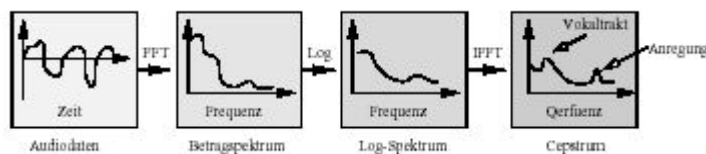


Abbildung 6-1: Cepstrumanalyse

Das Cepstrum ist definiert als die Rücktransformation des logarithmierten Leistungsdichtespektrums des Zeitsignals $s(t)$:

$$c(q) = F^{-1} \{ \log |S[\omega]|^2 \}$$

mit

$$S(\omega) = F \{ s(t) \},$$

Durch diese zweimalige Fouriertransformation (bzw. die Rücktransformation des logarithmierten Leistungsdichtespektrums) erhält man die "Frequenz" des Spektrums, das heißt den "Rhythmus" oder den "Takt" der harmonischen Frequenzanteile. "Cepstrum" entsteht durch Invertierung der ersten vier Buchstaben des Wortes "Spectrum" und wurde geprägt, um Verwechslungen zwischen den signalimmanenten Frequenzen einerseits und den sich erst über nochmalige Spektraltransformation ergebenden "cepstralen Frequenzen" andererseits zu vermeiden:

"Spektrum (*spectrum*)" wird zum "Cepstrum"

"*spectral analysis*" wird zu "*cepstral analysis* (cepstrale Analyse)"

"*frequency*" wird zu "*quefrequency* (Quefrequenz)"

"*harmonic*" wird zu "*rahmonic* (Rahmonische)"

"filtering" wird zu "liftering (Lifterung)"

Die Queffrenz q hat die Einheit Zyklen pro Hertz oder einfach Sekunden. Da die Queffrenz der Verzögerung (*delay*) der Autokorrelierten entspricht, wird sie häufig auch mit "d" abgekürzt.

Die Anwendungen der Cepstralanalyse sind ziemlich raffiniert: Fouriertransformiert man etwa ein Zeitsignal, das aus der Überlagerung aus den Klängen einer Flöte, einer Klarinette und eines Fagotts entsteht, erhält man eine Riesenanzahl von enthaltenen Frequenzen. Die Cepstralanalyse hiervon produziert lediglich drei Einzellinien, die den drei Instrumenten entsprechen.

Die cepstrale Analyse erweist sich aufgrund der zweimaligen Fourier-Transformation und der Logarithmus-Operation als sehr rechenaufwendig.

Treten im Spektralbereich große harmonische Intervalle auf, schlägt sich das im Cepstralbereich als niederqueffreter Anteil nieder, während geringe harmonische Intervalle etwa eines Rauschsignals zu einem hochqueffreteren Anteil transformiert werden. Bei der menschlichen Sprache besitzt die niedrige Grundfrequenz von z.B. 100 Hz eine große Anzahl spektraler Harmonischer, deren Abstand voneinander der relativ hohen Queffrenz von 10 ms entspricht.

6.1.1 Mathematische Separierung

Wir wollen die mathematische Separierung zweier Signale anhand des vereinfachten linearen Modells der Spracherzeugung zeigen.

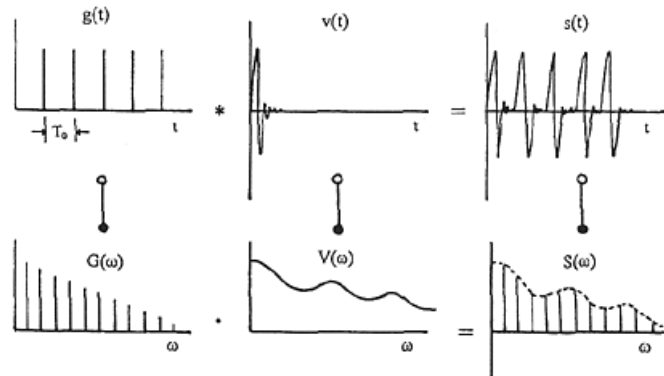


Abbildung 6-2: System zur Erzeugung stimmhafter Sprachlaute

Das vereinfachte lineare Modell der Spracherzeugung beschreibt die Erzeugung stimmhafter Sprachlaute im Zeitbereich durch die Faltung der glottalen Impulsfolgen-Anregungsfunktion $g(t)$ der Periode T_0 (inklusive additiv überlagerten Rauschens) mit der Impulsantwort $v(t)$ des Vokaltraktes:

$$s(t) = g(t) * v(t).$$

Dabei wird die glottale Anregung als echte Pulsfolge beschrieben, obwohl sie als Ergebnis der Faltung einer echten Pulsfolge mit der Impulsantwort des glottalen Systems eigentlich mehr dreieck- als deltaförmig ist. Die vorliegende Idealisierung ist aber eine im allgemeinen legitime Näherung.

Im Frequenzbereich entspricht dies der Multiplikation des glottalen Linienspektrums der Grundfrequenz $F_0 = 1/T_0$ mit der Vokaltraktübertragungsfunktion:

$$S(\omega) = G(\omega) \cdot V(\omega)$$

mit

$$G(\omega) = F\{g(t)\} \quad \text{und} \quad V(\omega) = F\{v(t)\}.$$

Für stimmhafte Sprachlaute ist das Anregungssignal und somit das Sprachsignal quasi-periodisch. Mit der Grundperiode T_0 ergibt sich der Abstand der Harmonischen im Spektrum bzw. Leistungsdichtespektrum zu F_0 , der sich über die Fourier Rücktransformation als Peak in der hierdurch entstehenden Autokorrelationsfunktion

$$\Phi(q) = F^{-1}\{|S(\omega)|^2\}$$

niederschlägt. Diese Gleichungen zusammengesetzt ergeben als weiteren Ausdruck für die Autokorrelationsfunktion:

$$\begin{aligned} \Phi(q) &= F^{-1}\{|G(\omega)|^2 |V(\omega)|^2\} \\ &= F^{-1}\{|G(\omega)|^2\} * F^{-1}\{|V(\omega)|^2\} \\ &= \Phi_g(q) * \Phi_v(q) \end{aligned}$$

mit den jeweiligen Autokorrelationsfunktionen von $g(t)$ und $v(t)$.

Um die Auswirkungen von Anregungssignal und Vokaltrakt-Impulsantwort voneinander trennen zu können (was über Entfaltung der Autokorrelationen nicht möglich ist), wird das Leistungsdichtespektrum vor der Fourier-Rücktransformation logarithmiert. Das Cepstrum ergibt sich zu

$$\begin{aligned} c(q) &= F^{-1}\{1 \circ \log [F\{\Phi(q)\}]\} \\ &= F^{-1}\{1 \circ \log |S[\omega]|\} \\ &= F^{-1}\{1 \circ \log |G(\omega)|^2\} + F^{-1}\{1 \circ \log |V(\omega)|^2\} \\ &= 2 F^{-1}\{1 \circ \log |G(\omega)|\} + 2 F^{-1}\{1 \circ \log |V(\omega)|\}. \end{aligned}$$

Anstatt der Faltung erhält man die mathematische Separierung zweier unabhängiger Funktionen. Die prinzipiellen Berechnungen sind in Abbildung 6-3 skizziert (Berechnung über Leistungsdichtespektrum).

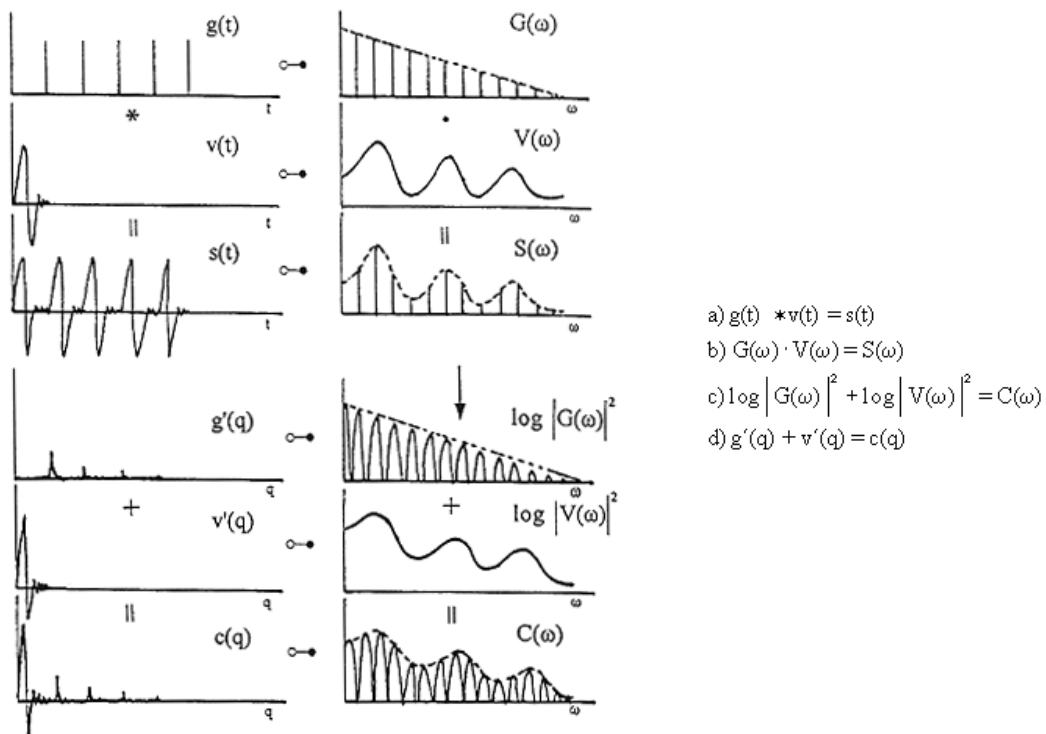


Abbildung 6-3: Die Berechnungen des Cepstrums eines harmonischen Sprachsignals

w = Spektrum des Cepstrums

$g'(q)$ = glottaler Cepstralanteil

$v'(q)$ = Vokaltrakt-Cepstralanteil

6.1.2 Beispiel zur Cepstral Analyse

Mit dem M-File echomark.m kann man gut die Vorteile der Cepstrumanalyse zeigen. Zunächst wird einmal das Wavefile laughter geladen.

```
load laughter;
z=y(1:10000);
neu = [z(1:5000)', (z(5001:10000)+z(1:5000))'];
figure(1);
stairs(xcorr(neu));
neu2 = [z(1:40)', (z(41:10000)+z(1:9960))'];
figure(2);
stairs(xcorr(neu2));
figure(3);
neu3 = circshift(neu2, [100,0]);
neu3(1:100)=0;
stairs(rceps(neu3(1:1000))' .* hann(1000));
```

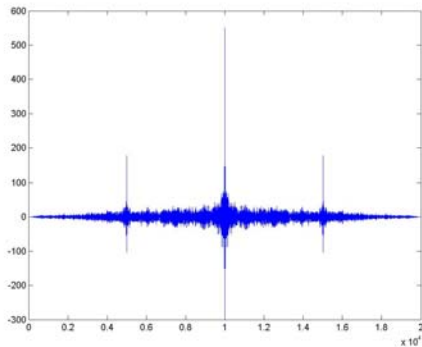


Abbildung 6-4: Autokorrelation mit grosser Verzögerung des Echos

Abbildung 6-4 weist drei markante Peaks auf. Wie bei jeder Autokorrelation ist der Peak in der Mitte am grössten. Die beiden kleineren Peaks stammen vom Echo. Weil die Autokorrelation symmetrisch ist, entstehen zwei Peaks mit gleicher Entfernung zum Nullpunkt. Das Echo ist hier mit einer Verzögerung von 5000 Samples, was 625 ms entspricht, eingefügt. Man kann es wegen dieser langen Verzögerung gut hören.

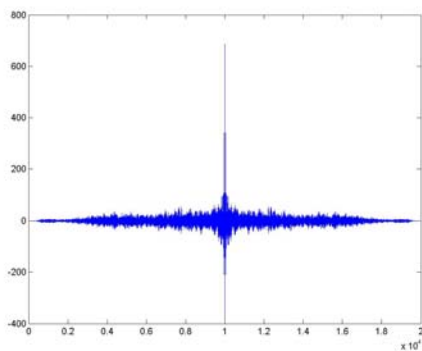


Abbildung 6-5: Autokorrelation mit kleiner Verzögerung des Echos

Nun wurde das Echo nur noch mit einer Verzögerung von 41 Samples, was 5.125 ms entspricht, eingefügt. In dieser Autokorrelation kann man Peaks die vom Echo stammen nicht mehr sehen, weil sie im Grundpeak verschwinden. Dafür ist das Echo nicht mehr hörbar.

Abhilfe schafft hier das Cepstrum. Auch wenn das Echo mit kleiner Verzögerung eingefügt wurde, entsteht ein markanter Peak, welcher leicht zu detektieren ist. Er ist an der Position 41, also an der Stelle die der Verzögerung des Echos entspricht.

Der Vektor neu3 ist eine Verschiebung des Sprachsignals mit Wasserzeichen. Trotz dieser Verschiebung um hundert Samples ist der gewünschte Peak klar ersichtlich.

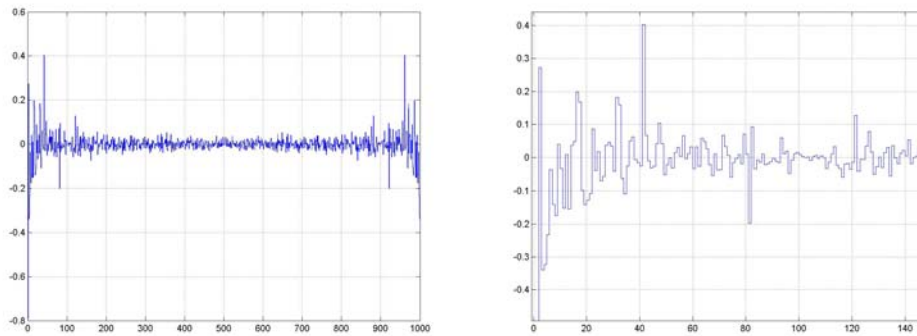


Abbildung 6-6: Cepstrum des Signals mit Echo

6.2 Einfügen des Wasserzeichens im Sender

Der Sender zerstückelt das originale Sprachsignal $x(n)$ in Frames und fügt dem Sprachsignal Echos mit der Abschwächung α , β dazu. Eines nach der Offsetzeit und ein zweites nach der Offsetzeit plus Delta. Je nach Verzögerung handelt es sich dabei um eine Eins oder eine Null.

$$y(n) = x(n) + \beta * p(n - \delta_1) \quad \text{für die Wasserzeicheninformation Null}$$

$$y(n) = x(n) + \alpha * p(n - \delta_2) \quad \text{für die Wasserzeicheninformation Eins}$$

Die Verzögerungszeit ist so klein, dass beide Echos vom menschlichen Gehör nicht wahrgenommen werden. Abbildung 4-10 zeigt wie das gemeint ist.

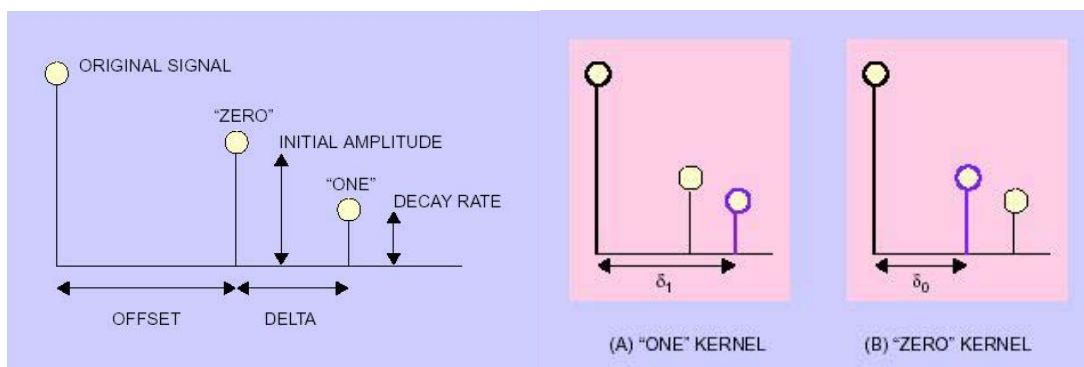


Abbildung 6-7: Einfügen des Echos

Gesendet wird das codierte Signal $y(n)$. Abbildung 4-11 zeigt das Blockschaltbild des Senders. Nach jedem Frame kann zwischen einfügen einer Null oder einfügen einer Eins gewechselt werden. Wann gewechselt wird, entscheidet der Wasserzeichenvektor der den One Mixer, und invertiert dem Zero Mixer steuert.

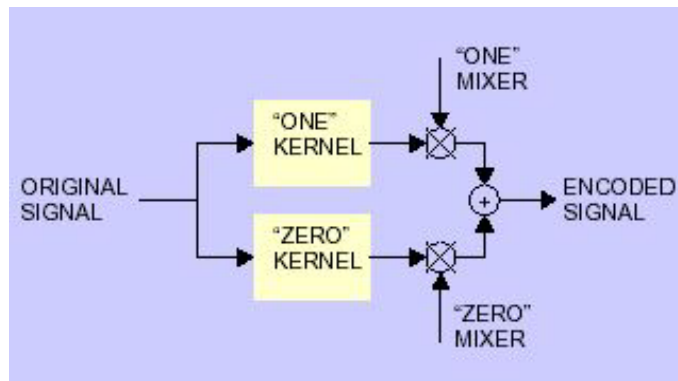


Abbildung 6-8: Aufbau des Senders

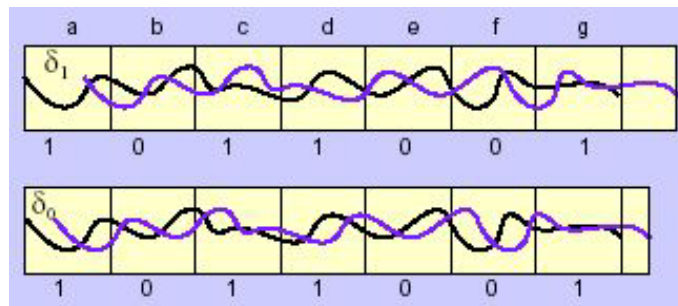


Abbildung 6-9: Echos des Originalsignals

6.3 Detektion des Wasserzeichens im Empfänger

Das ankommende Signal $y(n)$ wird in Frames unterteilt. Diese Frames werden ins Cepstrum gebracht.

$$c[n] = IFFT(\log|Y[w]|^2) \quad \text{mit } S[w] = FFT(y[n])$$

In $c[n]$ entstehen wiederum markante Peaks entsprechend der Verzögerung des Echos. Danach wird die Distanz zwischen dem ersten und zweiten Peak ermittelt. Ist sie zum Beispiel 8 Samples = 1 ms entspricht es einer Null, ist sie 11 Samples = 1.3 ms entspricht das einer Eins.

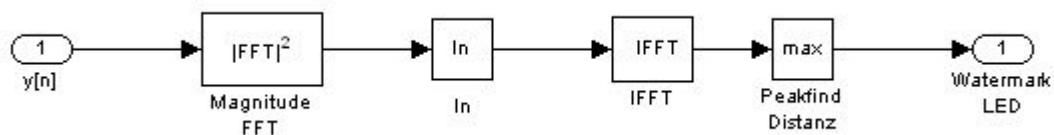


Abbildung 6-10: Blockschaltbild Empfänger

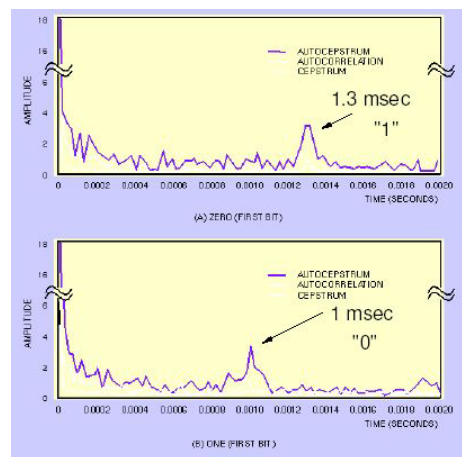


Abbildung 6-11: Peakfind im Cepstrum

6.3.1 Synchronisation

Der Sender fügt zum Beispiel während 590 Samples eine Eins ein. Beim Empfänger fährt ein moving Window über das empfangene Signal. Dieses moving Window hat nur eine Grösse von 512 Samples. Wird kein Peak gefunden, schreitet das moving Window 4 Samples vor und untersucht das Cepstrum erneut. Sobald ein Peak eindeutig gefunden wurde, rastet das System ein und untersucht fortan alle 600 Samples ob eine Eins oder eine Null im Datenstrom ist.

Ob nun das Cepstrum über den ersten 512 Werten der gesendeten 600 erfolgt oder erst einige Samples später spielt keine Rolle.

6.4 Auswertung des Echoverfahren

6.4.1 Bitrate und Detektionszeit

Entscheidend für die Bitrate B und die Detektionszeit T ist die Framelänge sowie die Abtastfrequenz bis ein Codebit detektiert werden kann.

Tabelle 6-1: Daten für Bitratenberechnung

Abtastfrequenz:	f_0	8000 Hz
Framelänge:	l	300, 590, 1080

$$B = \frac{f_0}{l} = \frac{8000}{590} = 13.6 \text{ Bit/s}$$

$$T = \frac{l}{f_0} = \frac{590}{8000} = 73 \text{ ms}$$

Eine höhere Bitrate erreicht man, wenn man die Framelänge verkleinert oder die Taktrate erhöht.

6.4.2 Messungen

Die Auswertung erfolgte mit dem in Matlab vorhandenen `laughter wavfile`. Zu beachten ist, dass die Rauschquelle zufällig ist, was zu leicht unterschiedlichen Resultaten führen kann.

Bei den Messungen variieren:

- Die Framelänge
- Der Signalrauschabstand SNR

Tabelle 6-2: Versuchseinstellungen:

Matlabfile:	awgn.m, echomarking_3
Framelängen:	300, 590, 1080 Samples
Einfügefaktor:	0.8
Bitraten:	26.4, 13.6, 7.4 Bits pro Sekunde

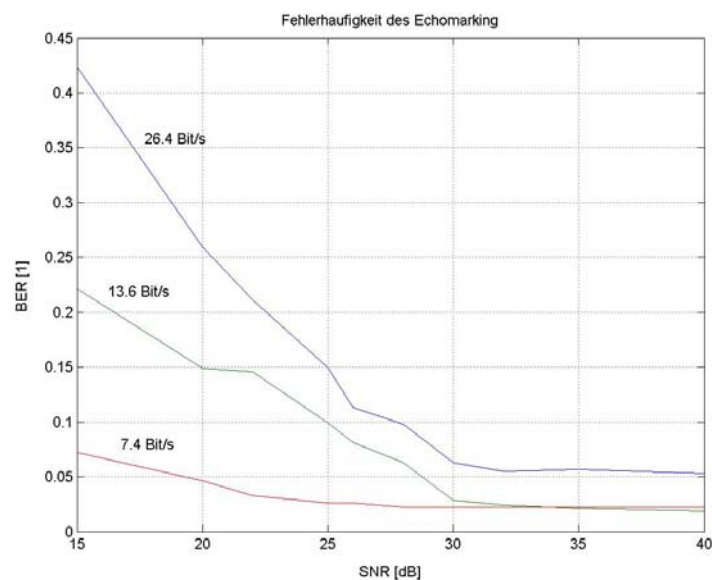


Abbildung 6-12: BER des Echoverfahren

Je kürzer die Framelänge gewählt wird, desto höher ist die Bitrate und der Fehler. Je mehr Rauschen eingefügt wird, desto schlechter sind die Resultate. Der Fehler wird nie zu Null, kann aber mit Redundanz im Wasserzeichen gelöst werden.

Kapitel 7: Vergleich Frequenzverfahren – Ecoverfahren

7.1 Vor- und Nachteile

7.1.1 Des Frequenzverfahren

Vorteile:

- Null Biterror sind möglich
- Sehr robust
- Braucht keine Synchronisation

Nachteile:

- Langsam
- Bitrate ist von der Framegrösse abhängig. Die Framegrösse muss mindestens 1024 sein.
- Einfügen im Sender benötigt deshalb schon mindestens 128 ms, was schon erheblich ist

7.1.2 Des Ecoverfahren

Vorteile:

- Schnell. Das Echo kann Sampleweise eingefügt werden, was zu praktisch keiner Verzögerung des Nutzsignals führt.
- Höhere Bitrate als beim Frequenzverfahren ist möglich.
- Mittlere Robustheit

Nachteile:

- Der Biterror kann nicht zu Null gemacht werden. Man braucht ein Wasserzeichen mit Redundanz.
- Cepstrum ist sehr rechenaufwendig
- Am Anfang ist eine „Synchronisation“ nötig.

Kapitel 8: Implementation

8.1 TMS320C6711 Entwicklungsumgebung

8.1.1 Hardware Development System Kit DSK

Einige Eckdaten des Entwicklungsboards:

- CPU: 150-MHz C6711DSP, mit der Leistung von 900 Millionen Floating Point Rechenoperationen pro Sekunde(MFLOPS)
- 16M Bytes Synchronous Dynamic Random Access Memory (SDRAM)
- 128K Bytes Flash
- 8-bit memory-mapped I/O Ports (3 User LED's, 3 Dip Switches und 2 I/O Ports)
- 16-bit, 8kHz audio codec



Tabelle 8-1: TMS320 DSP Board

Für unsere Anwendung stellt die Fließkomma-Rechenleistung der CPU (A) und der, an der CPU angeschlossene Analog/Digital Wandler (B) die zentralen Einheiten dar (rot eingefärbt). Auf den Positionen C1 und C2 wäre noch ein schnellerer Analog/Digital Wandler. Für unsere Anforderungen genügt aber der langsamere Wandler (B).

8.1.2 Code Composer Studio IDE

Die Entwicklungsumgebung Code Composer Studio, kurz CCS, ist ein sehr hilfreiches Tool für die Programmierung des DSP. Um das Programm effizient zu debuggen, hat man zahlreiche Hilfsmittel. Wir möchten diese Hilfsmittel auflisten und kurz erklären.

Breakpoints: Der klassische Breakpoint, wird von jeder Entwicklungsumgebung zur Verfügung gestellt und bewirkt ein Anhalten des Programmes an der Stelle, an welcher dieser Breakpoint gesetzt wurde. Im CCS kann man die einzelnen Breakpoints so steuern, dass sie sogar auf Bedingungen reagieren können. Das heisst, der Breakpoint ist nur dann gültig, wenn eine vorher angegebene Bedingung wahr ist.

Probepoints: Die klassische Variante eines Entwicklers ist die Ausgabe von Werten über eine Schnittstelle (Beispielsweise stdio). Mittels einem Probepoint kann man Datenwerte aus dem Speicher des DSP direkt in ein File auf dem Host PC speichern und dann mittels Matlab oder Excel analysieren. Zusätzlich ist auch die umgekehrte Datenrichtung möglich, man kann also Variablen auf der Target CPU vom Host PC aus in einen vordefinierten Zustand vorgeben.

File I/O: File I/O ist stark mit den Probepoints gekoppelt und stellt den Datentransfer zwischen einem File auf dem Host PC und dem DSP dar. Graph CCS bietet zusätzlich zu den Probepoints die Möglichkeit, Werte von Variablen (auch eindimensionale Arrays) grafisch darzustellen. Bei der Wahl der Grafik, besteht die Möglichkeit direkt das Frequenzspektrum der Werte zu betrachten. Dies war für uns hilfreich.

Realtime Data Exchange RTDX: Diese Schnittstelle ist sehr ähnlich zu den Probepoints. Der Unterschied besteht darin, dass bei einem Probepoint das Programm kurz unterbrochen wird, um während diesem Unterbruch die Daten zum Host PC zu übermitteln. Bei der RTDX Schnittstelle hingegen werden die Daten zwischengespeichert und im Hintergrund übermittelt. Die Ausführung des Programms wird also nicht unterbrochen. Für die Übermittlung selbst wird mehr Zeit benötigt, als vergleichsweise bei einem Probepoint. Die Daten werden dann übermittelt, wenn dies die Auslastung der Target CPU zulässt.

8.1.3 DSP Real Time Operating System (RTOS)

DSP RTOS ist ein skalierbarer realtime Kernel. Dieser Kernel stellt mächtige Mittel zur Verfügung, um ein Paralleles Softwaresystem auf dem DSP zu realisieren:

- Realtime Scheduling und Synchronisation
- Host-Target Kommunikation
- Real-Time Analysis

Kurz gesagt stellt uns Texas Instruments ein Betriebssystem zur Verfügung, welches uns ein Softwaresystem mit parallelen Prozessen ermöglicht. Dabei werden zwischen drei verschiedene Arten von parallelen Prozessen unterschieden, welche jeweils eine andere Prioritätsstufe beschreiben. Höchste Priorität haben Prozesse, welche von einem Hardware Interrupt (HWI) ausgelöst werden, eine tiefere Priorität haben Software Interrupts (SWI) und schliesslich die tiefste Priorität weisen Tasks (TSK) auf.

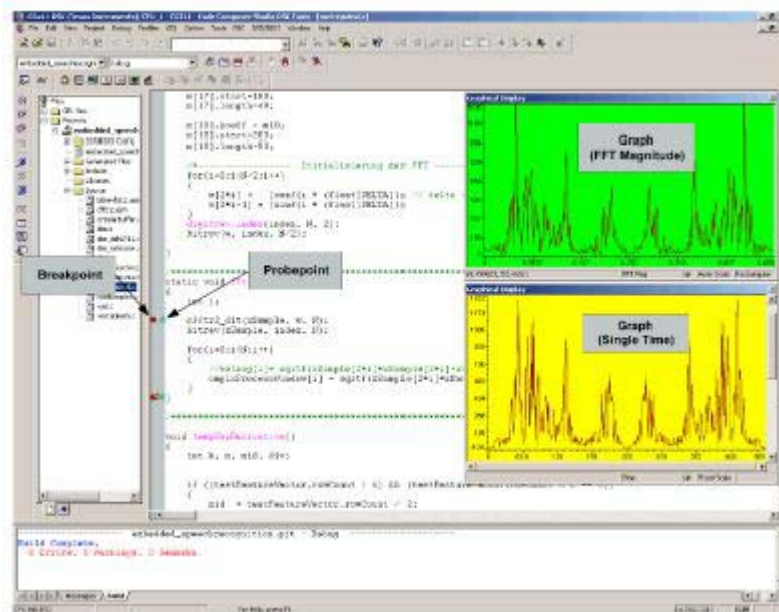


Abbildung 8-1: Code Composer Studio

8.2 Programme

Unten sind die Projekte EchoEmpfänger_2.pjt und Proj_3.pjt, welches den Sender beschreibt, aufgeführt.

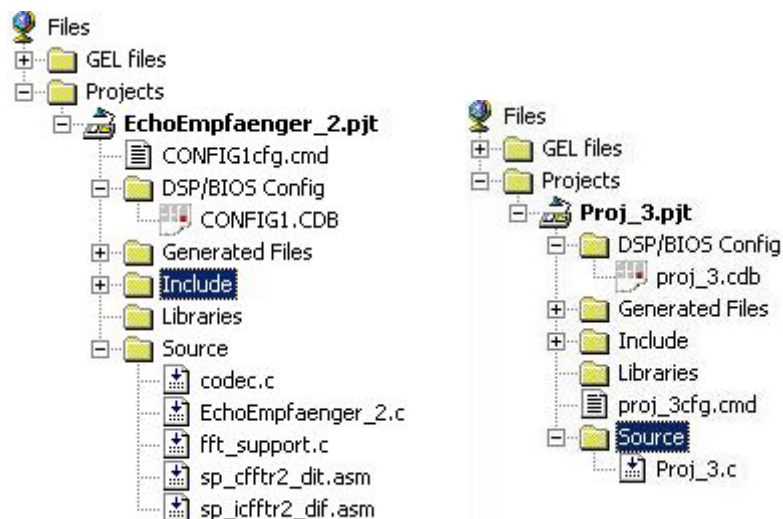


Abbildung 8-2: Projektaufbau

8.2.1 Der Sender

Im DSP Bios File proj3.cdb wird der Receive Interrupt des McBSP0 konfiguriert. Im File Proj_3.c wird der McBSP0 initialisiert und danach der AD / DA Wandler (AD535) ebenfalls initialisiert. Danach fällt das Programm in den Idle Loop. Sobald ein Interrupt durch den AD535 erfolgt, wird ein Sample eingelesen. Danach wird das Echo eingefügt und sofort wieder dem AD535 zurückgegeben. Der Sender ist noch auf die Ursprüngliche Idee programmiert das man Frameweise das Echo verändert und so „Daten“ Senden kann. (Wasserzeichen)

8.2.2 Der Empfänger

Im DSP Bios File CONFIG1.cdb wird der Receive Interrupt des McBSP0 konfiguriert. Im File fft_support.c werden die Hilfsfunktionen zur Ausführung des FFT/IFFT bereitgestellt. Im codec.c sind alle Funktionen die den AD535 und den McBSP ansteuern. Die FFT/IFFT Funktionen wurden aus der TMS320C67x DSP Library kopiert. Anfänglich wurde die ganze Library eingefügt. Jedoch funktionierte dann der McBSP0 receive Interrupt nicht.

Beim Empfänger muss die Frametime synchronisiert werden. Da wir 512 Punkte für das Cepstrum benutzen, konnte dies nicht zwischen zwei Sampeln gemacht werden wie es Ursprünglich gedacht war. Das Problem lag nicht am FFT bzw. IFFT. Diese Funktionen waren in Assembler geschrieben und nutzten die Pipeline-Verarbeitung des DSPs. Das Langsame war die Berechnung des Betrages der Komplexen Zahl und danach der Logarithmus. Es wurden verschiedene Optimierungen gemacht. Beim Betrag wurde die Wurzel nicht mehr gezogen und der Logarithmus wurde nicht mehr während der Laufzeit berechnet sondern als Tabelle abgespeichert. Dadurch konnte die Geschwindigkeit der Cepstrumberechnung erheblich gesteigert werden. Jedoch war die ganze Berechnung immer noch zu langsam. Eine Mögliche Variante wäre noch gewesen es zwischen mehreren Sampeln zu berechnen, und dann das Frame um diese Anzahl Sempel weiterzuschieben. Eine weitere Möglichkeit das Problem in den Griff zu kriegen wäre die Daten über den EDMA direkt in den Speicher zu schreiben. Das ist sicherlich die elegantere und bessere Lösung des Problems. Diese Lösung konnte jedoch aus zeitlichen Gründen nicht Implementiert werden.

Der Empfänger funktioniert folgendermassen, er liest 512 Werte ein. Danach sperrt er den McBSP0 Interrupt. Dann wird das Cepstrum berechnet und nach dem Maximalwert gesucht. Wenn der Maximalwert bei der Indexposition 40 ist wird USER_LED3 geschaltet, wenn der Wert 30 ist wird USER_LED2 geschaltet. Falls an keiner der beiden Positionen ein Maximum gefunden werden konnte wird keine der beiden LEDs geschaltet.

8.2.3 Interruptkonfiguration

Um die Interruptvektortabelle aufzusetzen wurde das DSP/BIOS benutzt. Um so ein File zu erstellen geht man im CCS auf File -> New(dskc6711) -> DSP/BIOS Configuration. Dann erscheint folgendes Fenster.

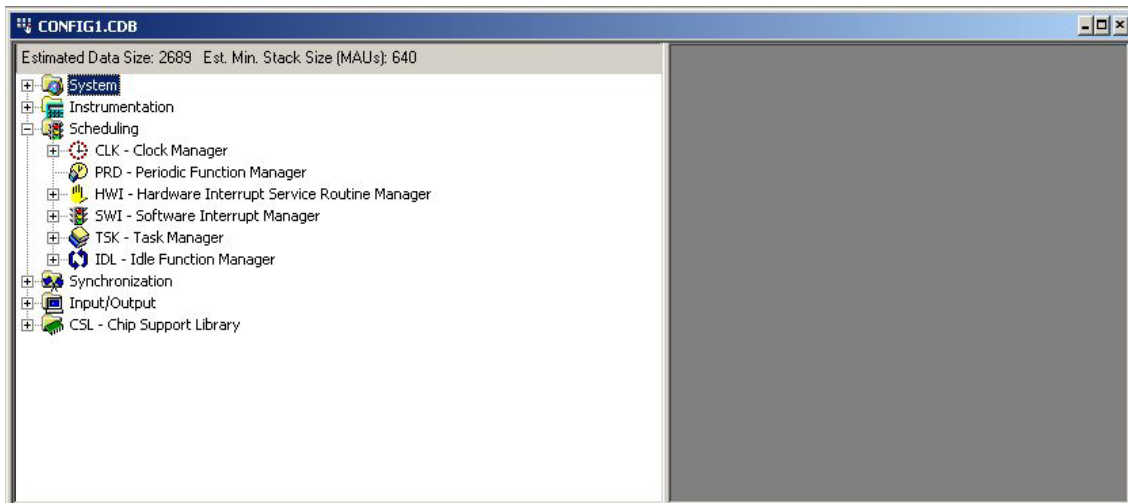


Abbildung 8-3: BIOSC Konfiguration

Danach führt man einen Doppelklick auf HWI aus. Dann erscheint das Untere Bild.

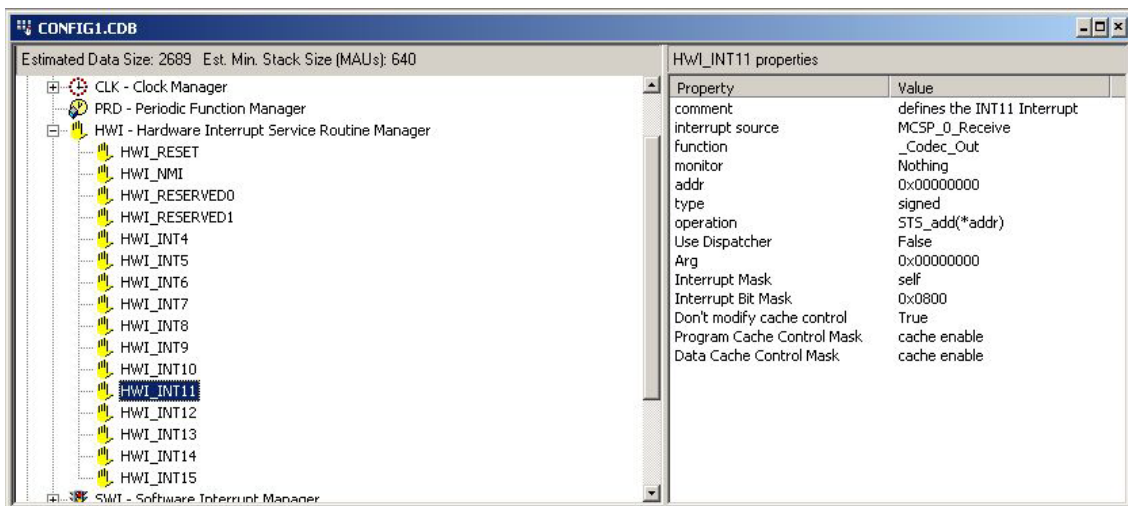


Abbildung 8-4: HWI Konfiguration

Jetzt muss man auf HWI_INT11 mit der rechten Maustaste klicken und wählt im erscheinenden Fenster auf Properties. Dann kann man im Feld function den Namen der Funktion angeben in die gesprungen werden soll wenn ein McBSP0 receive Interrupt erfolgt. Es ist darauf zu achten das ein Underscore vor den Funktionsnamen gemacht wird.

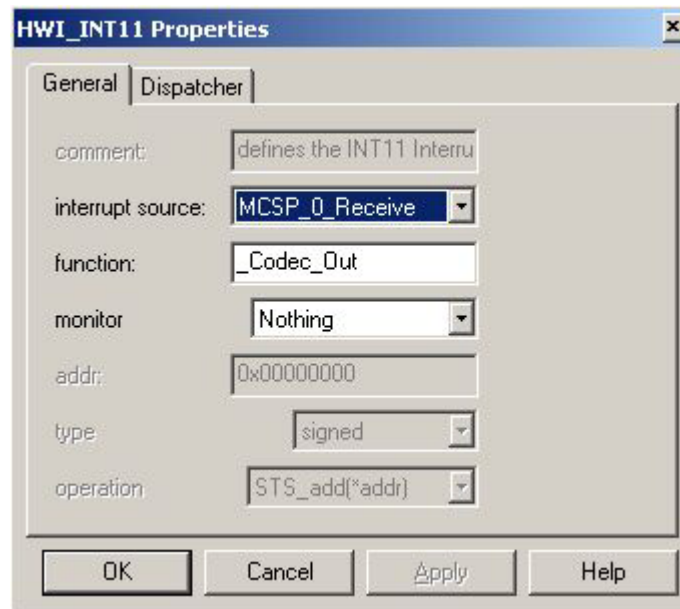


Abbildung 8-5: HWI Properties

Kapitel 9: Stand der Dinge

Stand vom 11.07.2003

9.1 Allgemein

Leider konnten wir das Projekt nicht vollständig bis zum gesetzten Termin zu Ende führen. Viele grundlegende Funktionen werden von der Software erfüllt.

9.2 Das funktioniert

Folgende Funktionen werden durch unsere Arbeit fehlerfrei ausgeführt:

- Interruptgesteuertes einlesen von Werten durch den McBSP
- Der DSP kann das Cepstrum berechnen
- Der DSP kann das Cepstrum auswerten und signalisieren das er ein Echo gefunden hat

9.3 Bemerkungen

Wenn das Programm ausgeführt wird sieht man das die LEDs ein und wieder ausgeschaltet werden. Die LEDs „flackern“. Die entsprechende LED wird eingeschaltet wenn ein „passendes“ Echo gefunden wurde. Wird jedoch keines gefunden so werden alle LEDs bis auf USER_LED1 ausgeschaltet.

Es wurde festgestellt, dass der Peak im Cepstrum nicht so hoch wird wie er im Matlab erscheint. Für den test wurde das Matlab File laughter geladen und danach das Echo eingefügt. Das File wurde über die Soundkarte des PC an den DSP abgespielt. Natürlich kann ich nicht sagen welches Frame der DSP für seine Berechnung genommen hat. Dann und nur dann wäre es vollkommen vergleichbar. Ich probierte jedoch Mehrehmahls verschiedene Zeitpunkte für die Aufnahme aus. Das Resultat war Praktisch immer das Gleiche. Der Echo Peak war da aber seine relative Differenzhöhe zu den umgebenden Werten war sehr viel kleiner. Eine mögliche Ursache für diesen Unterschied wäre das die FFT im DSP zu ungenau ist. Es ist eine Single Precision FFT. Im Matlab werden double benutzt.

Matlab befehle:

```
load laughter;
z = y(1:50000);
neu = [z(1:40)', (z(41:50000) + (z(1:49960))')'];
stem(rceps(neu(1+10000:512+10000))' .* hann(512));
```

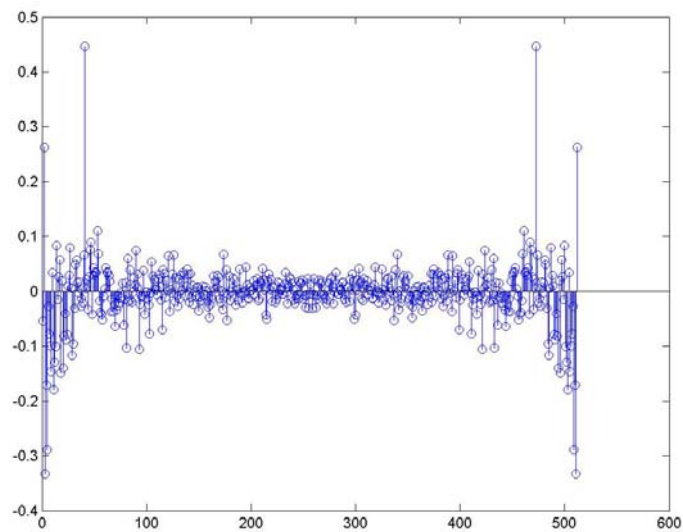


Abbildung 9-1: Cepstrum mit Matlab

Und zum vergleich ein Typisches Cepstrum wie es auf dem DSP gerechnet wird.

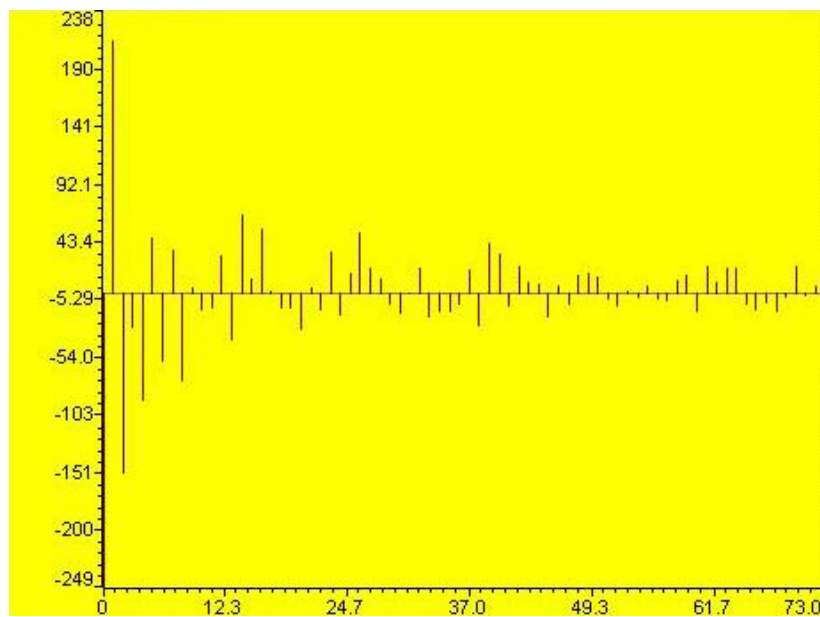


Abbildung 9-2: Cepstrum mit DSP

Man kann gut erkennen das der Echo Peak da ist, jedoch hat es um ihn noch viele andere die genauso Hoch sind wie er oder noch höher sind. Das Verursacht dann schlussendlich das „flackern“ des LEDs.

Kapitel 10: Schlussfolgerungen

10.1 Kritischer Rückblick

Die Arbeit war ohne Zweifel sehr interessant und abwechslungsreich. Da wir diese Arbeit wollten und mit Glück auch erhielten, war die Motivation auch dementsprechend hoch. Dennoch konnten wir leider, trotz enorm hohen Zeiteinsatzes vor allem gegen Ende der Arbeit, das Ziel nicht vollständig erreichen. Gerne hätten wir uns eines korrekten Wasserzeichens erfreut.

Rückblickend stellen wir fest, dass das Ziel hoch gesteckt war. Mit der Implementation in Matlab hat es gut geklappt. So fühlten wir uns sicher auf dem weiteren Weg. Ein harter, beschwerlicher Weg in den C Sümpfen von Code Composer Studio erwartete uns. Voller Elan haben wir dafür die Ferien geopfert und nach vier Tagen ohne Fortschritt ist es mir (MG) ausgefahren.

Die Arbeit war sehr lehrreich, konnten wir uns doch mit Problemen aus verschiedenen Gebieten der digitalen Signalverarbeitung befassen. Mit der Cepstrumanalyse sind wir in Gebiete vorgestossen, die wir im Unterricht nicht behandelten. Wir haben in der Programmierung von Matlab viel dazugelernt und auch unsere C-Kenntnisse verbessern können. Beide haben wir auch viele Messungen gemacht und uns in der Schule der logischen Schlussfolgerungen bemüht.

Vergessen darf man nicht die Teamarbeit. Zwei Dickschädel haben im gewissen Masse harmonisch miteinander arbeiten können und sich ergänzt. Nach dieser zweiten Semesterarbeit fühlen wir uns gewappnet für die Diplomarbeit.

10.2 Verbesserungen

Im Laufe der Arbeit haben wir festgestellt, dass es nicht so einfach ist einen DSP zu programmieren. Wir sind aber überzeugt, dass mit mehr Zeit beide Systeme erfolgreich zum laufen gebracht werden können.

Wir erfüllen nicht alle technischen Pflichten. Das Wasserzeichen wird nur einmal ausgewertet und nicht über den ganzen Datenstrom. Ausserdem ständig enthalten, nicht aber mit verschiedenen Echozeiten damit man Nuller und Einer übermitteln kann. Es beschränkt sich also nur auf das Detektieren eines Echos. Von hier aus ist der Schritt aber nicht mehr weit zu einem lauffähigen System.

10.3 Danksagung

Wir möchten es an dieser Stelle nicht unterlassen den Betreuer des DS-Labors, Herr Peter Roffler, den Laborassistenten Daniel Ostojic und Matthias Engelhardt herzlichst zu danken. An alle Studenten, die auch noch spätabends zu guten Raumklima beigetragen haben, sei ein weiteres Merci gewidmet.

Herr Schuster hat uns mit seinen Tipps einen Bärenienst erwiesen. Seine kompetente Beratung war stets ein Genuss. „Das ist der Moment wo sie danken dürfen.“ Danke.

Ein weiterer Dank gilt unserer Schulleitung. Sie hat uns liebevoll finanziell unterstützt und auch die professionelle, gute Infrastruktur zur Verfügung gestellt. Möge das unseren Nachfolgern erhalten bleiben!

Und nicht zuletzt ein mächtiger Dank allen Leser, die es geschafft haben bis hier hin zu lesen.

Rapperswil, 07.01.2003

Markus Gloor

Mario Jurcevic

Kapitel 11: Literaturverzeichnis

11.1 Bücher

- [1] Alan V. Oppenheim / Ronald W. Schafer, „Zeitdiskrete Signalverarbeitung“, R. Oldenburg Verlag München Wien, 1995
- [2] Lawrence R. Rabiner / Charles M. Rader, “Digital Signal Processing”, The Institute of electrical and electronics Engineers, New York, 1972
- [3] Matlab, „Matlab The Language of Technical Computing“, The MathWorks, 1996
- [4] Peter Vary / Ulrich Heute / Wolfgang Hess, “Digitale Sprachsignalverarbeitung”, Teubner, 1998
- [5] Rulph Chassaing, “DSP applications using C and the TMS320C6x DSK”, John Wiley & Sons Inc., New York, 2002

11.2 PDF-Files

- [1] Alex Schüeli, “Digitale Signalverarbeitung”, Skript HSR
- [2] Anthony Sibling / Simon Mark, „ Speech recognition using DTW on a TI DSP“, speechrecognition.pdf
- [3] Jaap Haitzma / Michiel van der Veen / Ton Kalker / Fons Bruekers, “Audio Watermarking for Monitoring and Copy Protection”, Haitzma.doc
- [4] Texas Instruments, “Using a TMS320C6000 McBSP for Data Packing”,McBSP_init.pdf
- [5] Texas Instruments, “TMS320C67x DSP Library Programmer’s Reference Guide“, c67_lib_spru_657.pdf
- [6] Texas Instruments, “TMS320C6000 McBSP Initialization“, McBSP.pdf
- [7] W. Bender / D. Gruhl / N. Morimoto / A. Lu, „Techniques for data hiding“, bender.pdf
- [8] Wolfgang Effelsberg / Ralf Steinmetz, “Digitale Wasserzeichen“, wasserzeichen.pdf

Kapitel 12: Anhang

12.1 Abbildungsverzeichnis

Abbildung 1-1: Übersicht Wasserzeichen.....	2
Abbildung 4-1: Signal nach Notchfilter	9
Abbildung 4-2: Spread Spectrum Verfahren.....	10
Abbildung 5-1: Einfügen des Wasserzeichen	11
Abbildung 5-2: Detektion des Wasserzeichens.....	12
Abbildung 5-3: Fehlerhäufigkeit 1.....	14
Abbildung 5-4: Fehlerhäufigkeit 2.....	15
Abbildung 5-5: Fehlerhäufigkeit.....	15
Abbildung 5-6: Fehlerhäufigkeit.....	16
Abbildung 5-7: Fehlerhäufigkeit.....	16
Abbildung 5-8: Fehlerhäufigkeit.....	17
Abbildung 6-1: Cepstrumanalyse.....	18
Abbildung 6-2: System zur Erzeugung stimmhafter Sprachlaute	19
Abbildung 6-3: Die Berechnungen des Cepstrums eines harmonischen Sprachsignals	21
Abbildung 6-4: Autokorrelation mit grosser Verzögerung des Echos	22
Abbildung 6-5: Autokorrelation mit kleiner Verzögerung des Echos.....	22
Abbildung 6-6: Cepstrum des Signals mit Echo	23
Abbildung 6-7: Einfügen des Echos	23
Abbildung 6-8: Aufbau des Senders	24
Abbildung 6-9: Echos des Originalsignals	24
Abbildung 6-10: Blockschaltbild Empfänger	24
Abbildung 6-11: Peakfind im Cepstrum.....	25
Abbildung 6-12: BER des Echoverfahren	26
Abbildung 8-1: Code Composer Studio.....	30
Abbildung 8-2: Projektaufbau.....	30
Abbildung 8-3: BIOSC Konfiguration.....	32
Abbildung 8-4: HWI Konfiguration.....	32
Abbildung 8-5: HWI Properties.....	33
Abbildung 9-1: Cepstrum mit Matlab.....	35
Abbildung 9-2: Cepstrum mit DSP	35

12.2 Glossary

A/D Wandler	Analog Digital Wandler
FFT	Fast Fourier Transformation
IFFT	Inverse Fast Fourier Transformation
C / C++	Programmiersprache
Matlab	Rechenprogramm
DSP	Digital Signal Prozessor
DSK	Design Starter Kit
LSB	Least Significant Bit
DA	Digital Analog
AD	Analog Digital
F	Fourier Transformation
F^{-1}	Inverse Fouriertransformation
SNR	Signal Rauschabstand
CPU	Central Processing Unit
MFLOPS	Millionen Floating Point Rechenoperationen pro Sekunde
I/O	Input Output
CCS	Code Composer Studio
PC	Personal Computer
RDTX	Real Time Data Exchange
RTOS	Real Time Operating System
HWI	Hardware Interrupt
SWI	Software Interrupt
TSK	Task