

Diplomarbeit



Von:

Mario Jurcevic

Markus Gloor

Betreuung:

Prof. Dr. Guido M. Schuster



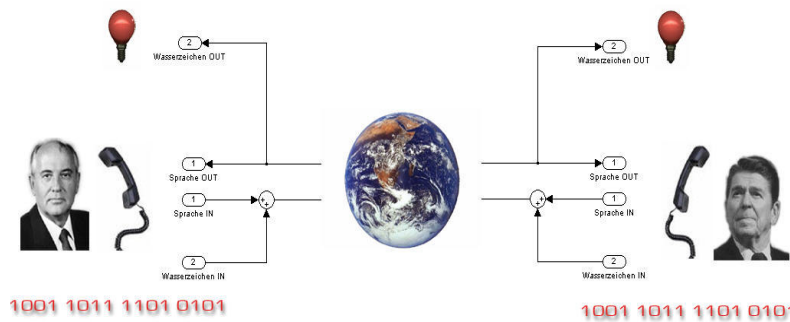
Verbessertes Echtzeit Wasserzeichensystem für Sprache

Rapperswil, 10. Dezember 2003

Abstract

Das Wasserzeichen

Unter einem Wasserzeichen kann man sich eine digitale Codesequenz, also eine bestimmte Folge von Nullen und Einsen vorstellen. Der Sender mischt dem Sprachsignal ein unhörbares Wasserzeichen bei. Weil das

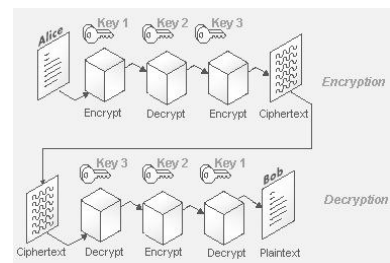


Wasserzeichen mit sehr kleiner Leistung beigegeben wird, kann man es nicht hören. Dieses Mischsignal wird nun über einen Telefonkanal gesendet. Das Wasserzeichen kann durch unser Verfahren am anderen Ende der Leitung im Empfänger wieder detektiert werden. Als original Wasserzeichen wird ein Counter verwendet. Das erste Datenpaket erhält so die Nummer eins, das zweite die Nummer zwei und so

weiter. Damit kann am Empfänger festgestellt werden, ob alle Datenpakete in der richtigen Reihenfolge angekommen sind.

3DES Codierung

Dieses original Wasserzeichen wäre auf der Leitung , wenn denn jemand unser Wasserzeichenverfahren kennt, leicht zu finden. Bevor nun das original Wasserzeichen geschickt wird, wird es bewusst nochmals mit dem 3DES Verfahren verschlüsselt. Am Empfänger kann diese Verschlüsselung nur mit denselben Keys wie sie der Sender benutzt hat, entschlüsselt werden. Auf der Leitung erscheint nun ein wildes Datendurcheinander, dass nicht mehr geknackt werden kann.

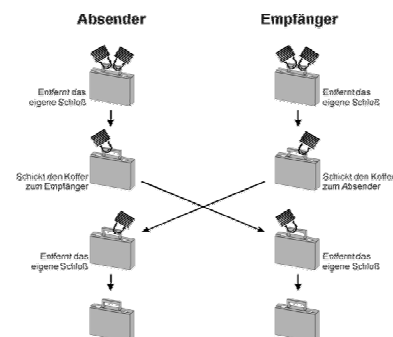


Diffie Hellmann Schlüsselaustausch

Damit der Sender und Empfänger mit denselben Schlüsseln arbeiten, übermittelt der Sender mit dem Diffie Hellmann Verfahren dem Empfänger die Schlüssel. Diffie Hellmann garantiert, dass niemand ausser dem Sender und Empfänger die Schlüssel kennt.

Bedeutung dieser Arbeit

Diese Diplomarbeit handelt über ein wichtiges Thema, ist Sicherheit in unserer Gesellschaft doch ein grosses Bedürfnis. Dank unserem Verfahren, ist man sicher, mit der richtigen Person ein ungeschnittenes Gespräch zu führen.



Inhaltsverzeichnis

Kapitel 1: Einleitung	5
Kapitel 2: Pflichtenheft	7
Kapitel 3: Projektplanung	8
3.1 Zeiteinteilung	8
3.2 Arbeitsaufteilung	9
Kapitel 4: Das Frequenzverfahren	10
4.1 Funktionsweise	10
4.2 Einbetten des Wasserzeichen im Sender	10
4.3 Detektion des Wasserzeichens im Empfänger	11
4.3.1 Synchronisation	12
4.4 Bitrate und Detektionszeit	12
Kapitel 5: 3DES	13
5.1 Die Geschichte von DES	13
5.2 Die Arbeitsweise von DES	13
5.2.1 Details zu DES	14
5.2.2 Betriebsarten von DES	17
5.3 Die Sicherheit von DES	19
5.4 Kombination von Blockchiffren	20
5.4.1 Doppelte Verschlüsselung	20
5.4.2 Dreifache Verschlüsselung	20
Kapitel 6: Diffie Hellmann	22
6.1 Das Protokoll	22
6.1.1 Zahlenbeispiel für DH	23
6.2 Berechnungsmethode	23
6.2.1 Die square and multiply Methode	23
Kapitel 7: Implementation	26
7.1 TMS320C6711 Entwicklungsumgebung	26
7.1.1 Hardware Development System Kit DSK	26
7.1.2 Analog Digital Wandler TDMX326040A	27
7.1.3 Code Composer Studio IDE	27
7.2 Versuchsaufbau	28
7.3 Programmaufbau	29
7.3.1 Das Projekt Tranceiver	29
7.3.2 Compilereinstellungen	30
7.3.3 Files und ihre Funktionen	31
7.3.4 Flussdiagramm	32
7.4 Ein- Ausgabe mit EDMA	32
7.4.1 Audio Daughter Card	32
7.4.2 Enhanced Direct Memory Access (EDMA)	33
7.4.3 Wie der EDMA funktioniert	34
7.4.4 EDMA Parameter (Options)	37
7.4.5 Ping Pong Buffer	39
7.4.6 Demoapplikation Daten Ein/Ausgabe	40
7.5 Das Frequenzverfahren	41

7.5.1 Einfügen des Wasserzeichens.....	41
7.5.2 Detektion des Wasserzeichens	42
7.5.3 Künstliches Rauschen.....	43
7.5.4 Synchronisation.....	43
7.6 3DES.....	45
7.7 Diffie Hellmann Schlüsselerzeugung	45
7.7.1 Einbindung der Library	46
7.7.2 Demoapplikation DH	47
Kapitel 8: Auswertung.....	48
8.1 Bitrate des Wasserzeichensignals.....	48
8.2 Störanfälligkeit und Fehler	49
8.2.1 Messungen	49
8.3 Tonqualität.....	53
8.4 Die beste Einstellung	53
Kapitel 9: Stand der Dinge.....	54
9.1 Allgemein.....	54
9.2 Das funktioniert.....	54
9.3 Das funktioniert nicht.....	54
Kapitel 10: Schlussfolgerungen	55
10.1 Kritischer Rückblick.....	55
10.2 Verbesserungen	55
10.3 Danksagung.....	56
Kapitel 11: Literaturverzeichnis.....	57
11.1 Bücher	57
11.2 PDF-Files	58
Kapitel 12: Anhang.....	59
12.1 Abbildungsverzeichnis.....	59
12.2 Glossary	60

Kapitel 1: Einleitung

Das vorliegende Dokument erklärt wie unser Wasserzeichensystem funktioniert, wie es in Matlab getestet und C / C++ implementiert wird. Hier ist der grobe Aufbau der Dokumentation.

Wir haben bereits in unserer zweiten Semesterarbeit zwei Wasserzeichensysteme mit Matlab und eines auch in C++ implementiert. Als Basis für die Diplomarbeit werden wir das Frequenzverfahren auf dem DSP umsetzen. Dazu sind aber noch weitere sicherheitsfördernde Ergänzungen wie 3DES und ein Diffie Hellmann Schlüsselaustausch eingebaut.

- 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. Die Funktionsweise des Systems sowie 3DES und Diffie Hellmann werden erklärt.
- 5 **Design:** In einem weiteren Schritt werden wir die getesteten Lösungen in C / C++ dokumentieren. Dazu gehört auch eine kleine Einführung in Code Composer Studio für die Programmierung des DSP.
- 6 Im Kapitel **Stand der Dinge** werden 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: *Verbessertes Echtzeit Wasserzeichensystem für Sprache*

Studenten: Mario Jurcevic und Markus Gloor

Betreuer: Prof. Dr. Guido M. Schuster

Kurzbeschreibung

Das Ziel dieser DA ist es das in der SA entwickelte Wasserzeichensystem für Sprache in Echtzeit zu implementieren und auszutesten. Als Wasserzeichen soll eine Uhr dienen, welche mittels 3DES codiert wurde. Der Schlüssel für diese 3DES soll mittels DH über den Wasserzeichenkanal zuerst zwischen den Stationen abgemacht werden.

Aufgabenstellung

- Erarbeitung und Darstellung der erforderlichen theoretischen Grundlagen
- Implementation des kompletten Systems auf der TI Plattform in Echtzeit

Erwartete Ergebnisse

- Dokumentation des Problems und der Lösung(en)
- 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. Theorie für 3DES erarbeiten und mögliche Implementationen suchen.
 - b. Diffie Hellmann Schlüsselaustausch für die Implementation vorbereiten und die nötige Theorie erarbeiten.
2. Implementation (Design):
 - a. So schnell wie möglich das Frequenzverfahren auf dem DSP umsetzen.
 - b. Eine Kommunikationsstrecke nachbauen und testen.
 - c. 3DES mit vorgegebenen Schlüsseln ins Projekt einbinden.
 - d. Diffie Hellmann ins Projekt einbinden und die Schlüssel für 3DES so festlegen.
3. Technische Pflichten:
 - a. Die Lösung soll auf dem TMS320 DSP Board in Realtime funktionieren.
 - b. Das eingebettete Wasserzeichen soll nicht hörbar sein.
 - c. Als Kanal gilt ein Telefonkanal mit 4 kHz Bandbreite.
 - d. Die gesendeten Informationen sollen mit 3DES verschlüsselt sein.
 - e. Die Schlüssel für 3DES werden beim Verbindungsaufbau mit einem Diffie Hellmann Schlüsselaustausch festgelegt.
4. Arbeitsweise
 - a. Step by Step.
 - b. Nach Arbeitsaufteilung gemäss Kapitel 3.
 - c. Laborbuch gemäss Aufgabenstellung.
5. Dokumentation:
 - a. Keine Einschränkungen, sehr frei!

Kapitel 3: Projektplanung

3.1 Zeiteinteilung

Tabelle 3-1: Übersicht Zeitplan

Bezeichnung		2003							
		KW43	KW44	KW45	KW46	KW47	KW48	KW49	KW50
		25.10	01.11	08.11.	15.11.	22.11.	29.11	06.12.	09.12.
EDMA	Daten werden eingelesen	■							
	Perfekte Ein-Ausgabe	■	■	■	■				
Sender	Soll konstanten Peak senden	■	■						
	Soll variablen Peak senden		■	■					
	Soll eine Uhr senden		■	■					
Empfänger	Soll konstanten Peak erkennen	■	■						
	Soll variablen Peak finden		■	■	■				
3DES	Funktioniert für sich alleine		■			■			
	Mit Sender und Empfänger				■	■	■	■	
Diffie Hellmann	Boards tauschen Daten aus					■	■	■	
	Diffie Hellmann funktioniert						■	■	■
Doku	Schreiben			■	■	■		■	■

Legende:

■	Ungefähr geplante Arbeiten
■	Effektiv ausgeführte Arbeiten

3.2 Arbeitsaufteilung

Tabelle 3-2: Arbeitsaufteilung

Aufgabe:	Person:
Eingabe – Ausgabe von Sprache EDMA	Mario Jurcevic
Sender	Markus Gloor
Empfänger	Markus Gloor
3DES	Mario Jurcevic
Kommunikation zwischen Sender und Empfänger	Markus Gloor
Diffie Hellmann	Mario Jurcevic

Kapitel 4: 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.

4.1 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.

4.2 Einbetten des Wasserzeichen im Sender

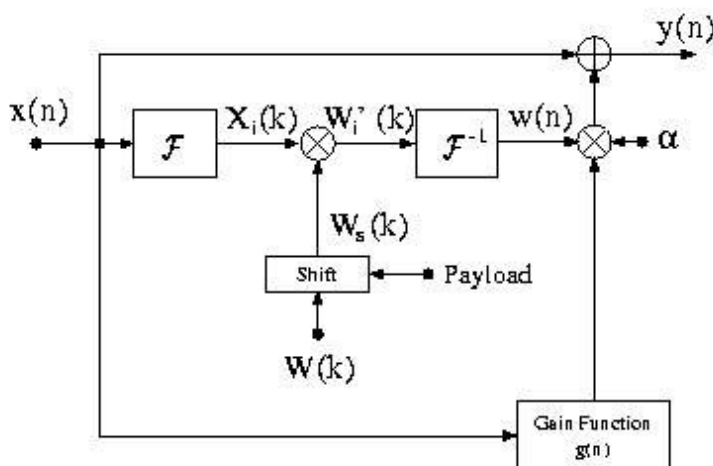


Abbildung 4-1: Einfügen des Wasserzeichen

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.

4.3 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.

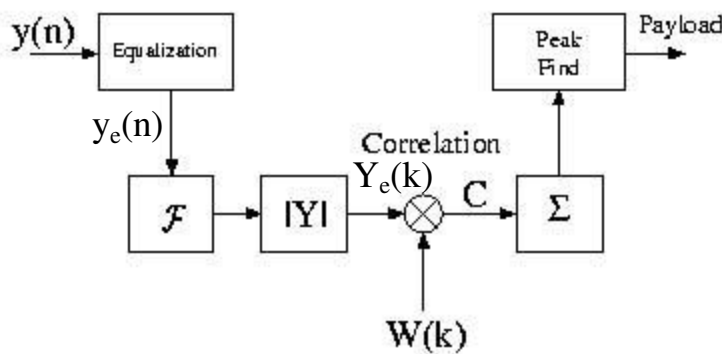


Abbildung 4-2: Detektion des Wasserzeichens

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.

4.3.1 Synchronisation

Bei diesem Verfahren ist normalerweise 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.

Wir haben uns in einem späteren Schritt entschieden doch eine Synchronisation einzubauen. Details dazu sind im Kapitel Implementation zu finden.

4.4 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 4-1: Daten für Bitratenberechnung

Abtastfrequenz:	f_o	24 kHz
Framelänge:	l	1024 Samples
Anzahl Frames für ein Codewort:	k	25

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

Der Codevektor darf nur von der Stelle 0 bis zur Stelle 511 verschoben werden. Das ist deshalb so, weil das Spektrum symmetrisch wird und zwei Peaks entstehen.

$$T = \frac{1}{f_o} \cdot l \cdot k = \frac{1}{25000} \cdot 1024 \cdot 25 = 1.024s$$

Nach dieser Zeit T können 9 Bits detektiert werden.

$$B = \frac{n}{T} = \frac{9}{1.024} = 8.79 \text{ Bit / s}$$

Eine höhere Bitrate erreicht man, wenn man zum Beispiel nur 10 Frames für ein Codewort benutzt. Dafür entstehen dann mehr Fehler.

Kapitel 5: 3DES

Eine der bekanntesten Blockschiffren ist der Data Encryption Standard (DES). Es ist der auch am meisten eingesetzte Verschlüsselungsalgorithmus. Der DES stellt seit mehr als zwanzig Jahren einen weltweit anerkannten Standard dar. Er hat jahrelanger Kryptoanalyse gut widerstehen können und bietet noch immer Schutz vor den meisten Angreifern sofern sie ihre Attacken nicht höchst aufwendig gestalten. 3DES bedeutet das der DES dreimal nacheinander ausgeführt wird. Typischerweise mit drei verschiedenen Schlüsseln.

5.1 Die Geschichte von DES

Im Jahre 1971 machte das US Wirtschaftsministerium eine öffentliche Ausschreibung für einen standardisierten kryptographischen Algorithmus. Auflage an das Verfahren waren verschiedenste. Das Verfahren musste einen hohen Grad an Sicherheit bieten, die Sicherheit sollte auf den Schlüsseln beruhen, und nicht durch die Geheimhaltung des Algorithmus. Weiter sollte der Algorithmus für alle Anwender verfügbar sein. Er muss sich kostengünstig in elektronischen Komponenten implementieren lassen. Der einzige Vorschlag der in die engere Wahl kam war der von der Firma IBM entwickelte Lucifer-Algorithmus (eine Weiterentwicklung davon). Dieses Chiffrierverfahren wurde von H. Feistel entwickelt in den Jahren 1968 bis 1975. Es wurde in den siebziger Jahren zum Beispiel in Bankautomaten eingesetzt. Die Lucifer Chiffre ist eine symmetrische Blockchiffre die auf 128-Bit Daten Blöcken operiert. Es benutzt 128-Bit Schlüssel. Der eingereichte Vorschlag wurde durch das NBS (National Bureau of Standards) analysiert. Das NBS bat um die Unterstützung der NSA. Die NSA sollte die Sicherheit überprüfen. Es wurden einige Modifikationen vorgenommen. Schließlich wurde im November 1976 der Data Encryption Standard verabschiedet. Das American National Standards Institute (ANSI) erkannte DES 1981 als Standard für den privaten Sektor an.

5.2 Die Arbeitsweise von DES

Der DES ist eine Block-Produkt-Chiffre aus Permutationen und nicht linearen Substitutionen. Es werden jeweils 64 Bit Daten verschlüsselt. Aus einem Block Klartext (64 Bit) wird ein Block Chiffretext (64 Bit). Der Schlüssel besteht aus 64 Bit, jedoch sind die Bits 8,16,24,...,64 Paritätsbits. Es werden 7 Bits auf ungerade Parität ergänzt. Somit ist die tatsächliche Schlüssellänge 56 Bit. Der Schlüsselraum vom DES hat die Mächtigkeit 2^{56} .

Die Arbeitsweise von DES ist folgende: Jeder Block Daten (64 Bit) wird zunächst einer Eingangspermutation unterworfen. Diese ist immer die gleiche. Der resultierende Block wird nun aufgeteilt. In einen linken und rechten Teilblock. Diese beiden Blöcke werden nun sechzehnmal gleichartig verschlüsselt. Jeder Verschlüsselungsdurchlauf ist von einem 48-Bit Teilschlüssel des Externen (56 Bit) abhängig. Die Teilschlüssel werden durch ein Schlüsselauswahlverfahren erzeugt. Nach jedem Durchlauf wird ein neuer Teilschlüssel verwendet. Nach der Durchführung der 16 Verschlüsselungsschritte werden die beiden 32 Bit Blöcke wieder zu einem zusammengesetzt. Danach wird die inverse Permutation der Eingangspermutation durchgeführt. Die Dechiffrierung erfolgt analog zur Chiffrierung. Das einzige das sich ändert ist das die 16 Teilschlüssel in der umgekehrten Reihenfolge benötigt werden. Auf der nächsten Seite kann man ein Blockschaltbild vom DES sehen.

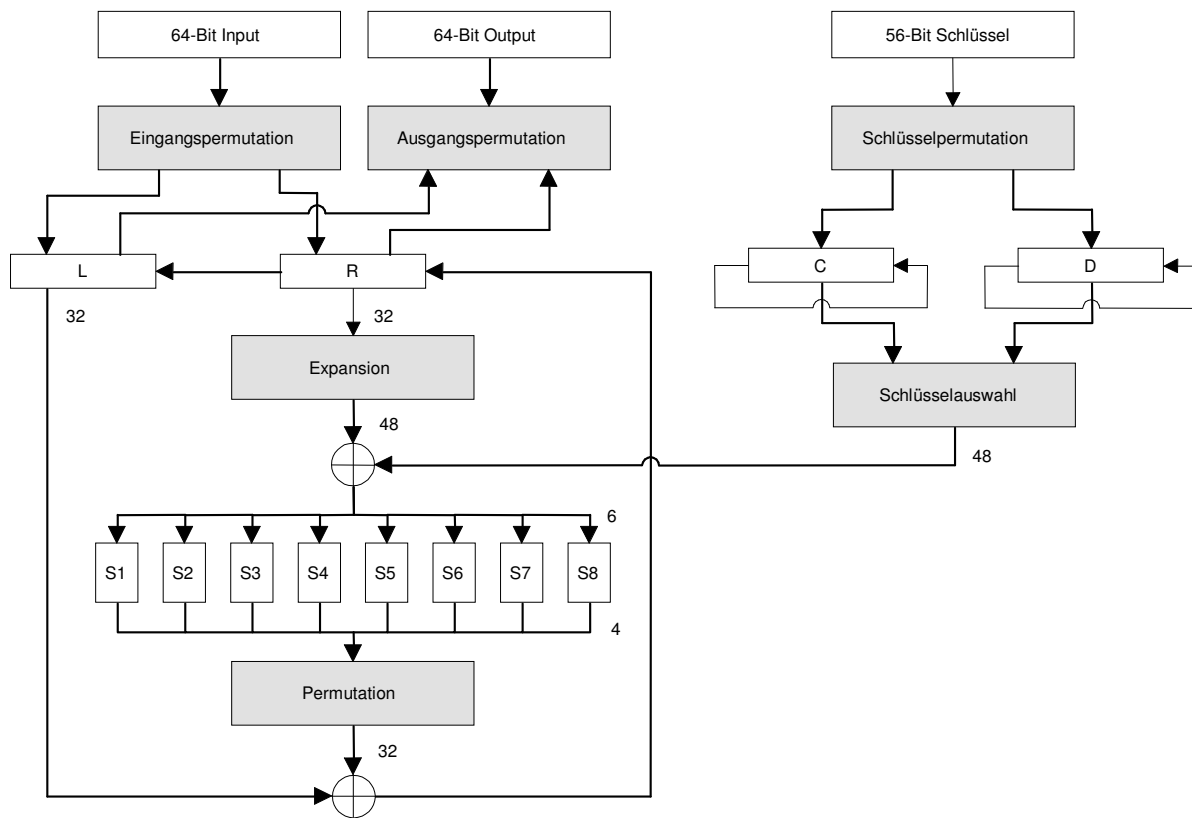


Abbildung 5-1: DES Algorithmus

5.2.1 Details zu DES

Die Funktionsweise von DES kann man in vielen Büchern nachschauen. Ich fasse das wichtigste kurz zusammen. Die Ein- Ausgangspertutationen beeinflussen die Sicherheit von DES überhaupt nicht. Danach folgen 16 Runden identischer Operationen. Die sogenannte Funktion F kombiniert den Schlüssel mit den Daten. In jeder Runde werden die Bits des Schlüssels verschoben. Dann werden 48 Bit aus den 56 Bits ausgewählt. Die rechte Hälfte der Daten wird mit Hilfe einer Expansionspermutation auf 48 verbreitert. Mit XOR werden nun 48 Bits expandierter Daten mit 48 Schlüsselbits kombiniert. Danach durch acht S-Boxen nochmals verändert. Das Resultat nach den S-Boxen sind 32 neue Bits. Schliesslich wird die Permutation des Eingangs wieder rückgängig gemacht. Das Ergebnis dieser Operation wird die neue rechte Hälfte, und die alte rechte Hälfte wird zur neuen linken Hälfte. Diese Operationen werden sechzehnmal wiederholt.

5.2.1.1 Eingangspertutation

Die Eingangspertutation wandelt einen 64-Bit Inputblock in die zwei 32 Bit Blöcke R und L.

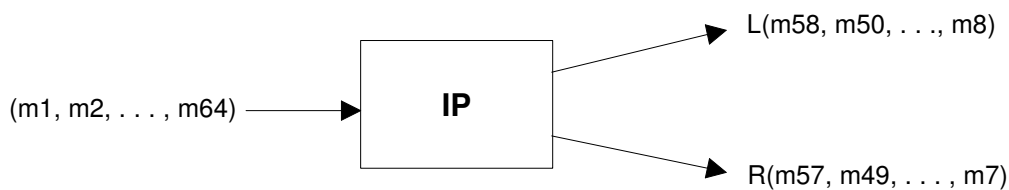


Abbildung 5-2: Eingangspertutation

L

58 50 42 34 26 18 10 2 60 52 44 36 28 20 12 4 62 54 46 38 30 22 14 6 64 56 48 40 32 24 16 8

R

57 49 41 33 25 17 9 1 59 51 43 35 27 19 11 3 61 53 45 37 29 21 13 5 63 55 47 39 31 23 15 7

Abbildung 5-3: DES Blöcke

5.2.1.2 Expansionspermutation

Diese Operation expandiert den R-Block auf 48 Bit. Sie heisst Expansionspermutation da sie die Reihenfolge der Bits ändert und einige davon wiederholt. Sie hat die Aufgabe den R Block auf die Schlüssellänge zu vergrössern und eine längere Bitfolge zu erzeugen. Diese werden später in den S-Boxen komprimiert. Diese Operation wird durchgeführt damit sich die Eingabebits schneller auf die Ausgabebits auswirken. Ein Ziel beim Entwurf von DES war, dass die Bits des Chiffretextes so schnell wie möglich von den Klartextbits und den Schlüsselbits abhängen sollten.

Tabelle 5-1: Expansionspermutation

32 1 2 3 4 5	4 5 6 7 8 9
8 9 10 11 12 13	12 13 14 15 16 17
16 17 18 19 20 21	20 21 22 23 24 25
24 25 26 27 28 29	28 29 30 31 32 1

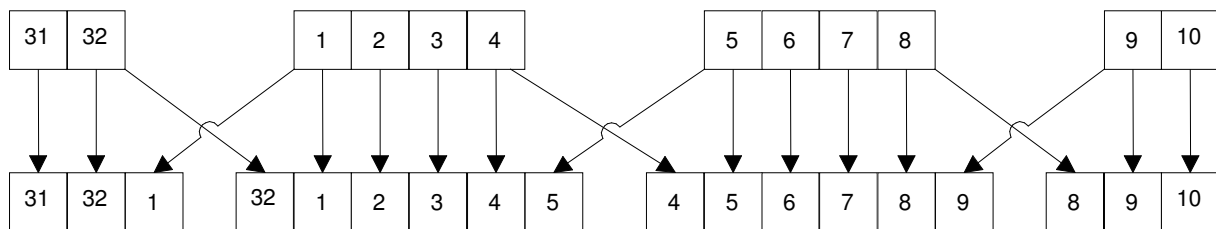


Abbildung 5-4: Expansionspermutation

5.2.1.3 S-Box-Substitutionen

Nach der Expansion folgt die Substitution. Diese erfolgt in den acht sogenannten S-Boxen. Jede S-Box substituiert einen 6-Bit Input Block (i1, i2, i3, i4, i5, i6) in einen 4-Bit Outputblock (o1, o2, o3, o4). Die 48 Bit (die nach der Expansion) werden in acht Teilblöcke der Länge 6 Bit zerlegt. Jeder Block geht durch eine S-Box. Der erste Block geht durch S-Box 1 der zweite durch S-Box 2 usw. Jede S-Box besteht aus einer Tabelle mit vier Zeilen und sechzehn Spalten. Die zuvor beigefügten Bits in der Expansion werden zur Auswahl der 4-Bit Substitution benutzt. Der Teilblock (i2, i3, i4, i5) bildet dann den Input für eine der vier Substitutionstabellen. Die Bits (i2, i3, i4, i5) wählen die Spalte und die Bits (i1, i6) wählen die Zeile. Ein kleines Beispiel veranschaulicht das nochmals.

Input (1, 1, 1, 0, 0, 0) (1, 0) -> 1Zeile und (1, 1, 0, 0) -> 3 Spalte Der Output ist also (0, 0, 1, 0) = 4.

Tabelle 5-2: S-Box 1

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
S1 0	E	4	D	1	2	F	B	8	3	A	6	C	5	9	0	7
1	0	F	7	4	E	2	D	1	A	6	C	B	9	5	3	8
2	4	1	E	8	D	6	2	B	F	C	9	7	3	A	5	0
3	F	C	8	2	4	9	1	7	5	B	3	E	A	0	6	D

Die S-Box Substitution ist die wichtigste Stelle von DES. Die anderen Operationen sind linear und lassen sich leicht analysieren. Die S-Boxen sind nichtlinear und gewährleisten die Sicherheit. Die Substitution mit den S-Boxen liefert als Ergebnis acht Blöcke der Länge 4 Bit. Man fasst diese zu einem 32 Bit Block zusammen. Dieser Block wird nun permutiert (P-Box).

5.2.1.4 Permutation

Die 32 Bit Ausgabe der S-Boxen bildet den Input für diese Permutation. Nach der Permutation wird das Ergebnis mit dem L Block XOR verknüpft. Dieses Ergebnis ist der neue Block R. Der alte Block R ersetzt den L Block.

Tabelle 5-3: P-Box

16	7	20	21	29	12	28	17	1	15	23	26	5	18	31	10
2	8	24	14	32	27	3	9	19	13	30	6	22	11	4	25

5.2.1.5 Ausgangspermutation

Die Ausgangspermutation ist die Inverse zur Eingangspermutation.

Tabelle 5-4: IP^{-1}

40	8	48	16	56	24	64	32	39	7	47	15	55
38	6	46	14	54	22	62	30	37	5	45	13	53
36	4	44	12	52	20	60	28	35	3	43	11	51
34	2	42	10	50	18	58	26	33	1	41	9	49

5.2.1.6 Schlüsselpermutation

Die Funktion PC-1 (permuted choice) wählt aus dem externen Schlüssel die 56 kryptographisch relevanten Schlüsselbits aus und permutiert sie. Die Bits 8, 16, 24, . . . , 64 sind Paritätsbits zwecks Fehlererkennung. Jedes Schlüsselbyte besitzt ungerade Parität.

Tabelle 5-5: Schlüsselpermutation C

57	49	41	33	25	17	9	1	58	50	42	34	26	18
10	2	59	51	43	35	27	19	11	3	60	52	44	36

Tabelle 5-6: Schlüsselpermutation D

63	55	47	39	31	23	15	7	62	54	46	38	30	22
14	6	61	53	45	37	29	21	13	5	28	20	12	4

Das Ergebnis der Permutation sind zwei 28 Bit Blöcke C und D. Die Register C und D werden vor jedem der 16 Verschlüsselungsschritte zyklisch um 1 oder 2 Bit nach links verschoben.

Tabelle 5-7: Shifts

Schritt	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Shifts	1	1	2	2	2	2	2	2	1	2	2	2	2	2	2	1

5.2.1.7 Schlüsselauswahl

Die Funktion PC-2 wählt aus den beiden Registern C und D 48 Schlüsselbits aus, und permutiert sie.

Tabelle 5-8: PC-2

14	17	11	24	1	5	3	28	15	6	21	10
23	19	12	4	26	8	16	7	27	20	13	2
41	52	31	37	47	55	30	40	51	45	33	48
44	49	39	56	34	53	46	42	50	36	29	32

5.2.1.8 Entschlüsselung

Bei DES kann man einen Block mit der gleichen Funktion ver- und entschlüsseln. Der einzige Unterschied besteht in der Reihenfolge der Schlüssel. Sind die Schlüssel für die Chiffrierung $K_1, K_2, K_3, \dots, K_{16}$ so sind $K_{16}, K_{15}, K_{14}, \dots, K_1$ die Schlüssel für die Dechiffrierung. Der Schlüssel wird immer nach rechts verschoben, und zwar jeweils 0, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 1

5.2.2 Betriebsarten von DES

Man kann DES in vier verschiedenen Betriebsarten benutzen. Das kann man mit allen blockorientierten Chiffren. Diese Modi unterscheiden sich in ihrer erreichbaren Verschlüsselungsrate, der Fehlerfortpflanzung und der Veränderung des Chiffretexts. Der Electronic Code Book Modus (ECB) wird zur Chiffrierung sehr kurzer Nachrichten verwendet. Typischerweise verschlüsselt man damit 1 Block. Der Cipher Block Chaining Modus (CBC) wird zur Verschlüsselung mehrerer Blöcke verwendet. Die Blöcke werden dabei miteinander verkettet. Die Modi Output Feedback (OFB) und Cipher Feedback (CFB) werden zur Verschlüsselung von sequentiell generierten Daten verwendet, oder wenn die Daten kleiner als die Blocklänge sind.

5.2.2.1 ECB

Bildet Klartextblöcke in Chiffretextblöcke ab.

$$M_1, M_2, M_3, \dots \rightarrow C_1, C_2, C_3, \dots \quad C_i = E(K, M_i) \quad M_i = D(K, C_i)$$

Jeder Chiffretextblock hängt nur vom zugehörigen Klartextblock und Schlüssel ab.



Abbildung 5-5: ECB

5.2.2.2 CBC

In dieser Betriebsart wird das Resultat einer Verschlüsselungsoperation dazu benutzt den folgenden Klartextblock durch Bitweise XOR – Verknüpfung zu modifizieren. Dadurch hängt ein Chiffretextblock C_i nicht allein vom zugehörigen Klartextblock M_i , sondern auch von allen vorhergehenden Klartextblöcken ab. Zur Veränderung des ersten Blocks der keinen Vorgänger besitzt, benutzt man einen Initialisierungsvektor (IV).

$$M_1, M_2, M_3, \dots \rightarrow E(K, M_1+IV)=C_1, E(K, M_2+C_1)=C_2, E(K, M_3+C_2)=C_3, \dots$$

$$D(K, C_1) + IV = M_1, D(K, C_2) + C_1 = M_2, D(K, C_3) = M_3, \dots$$

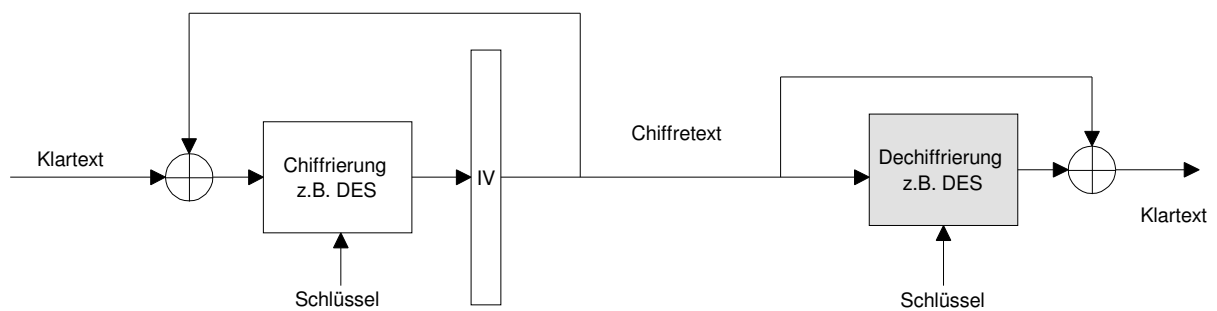


Abbildung 5-6: CBC Modus

5.2.2.3 OFB

In dieser Betriebsart wird eine Blockchiffre als Pseudozufallsgenerator eingesetzt und dient zur Erzeugung eines Schlüsselstromes. Es wird ein m-Bit Block Daten mit einem m-Bit Schlüssel XOR verknüpft. Der Schlüssel wird aus dem Outputblock berechnet. Es werden jeweils m-Bit auf den Eingang rückgekoppelt. Dadurch werden m-Bits des alten Inputs ersetzt. Die Dechiffrierung erfolgt gleich wie die Chiffrierung. Tritt im OFB Modus ein Synchronisationsfehler auf, so kann der Empfänger alle nachfolgenden Zeichen nicht korrekt dechiffrieren.

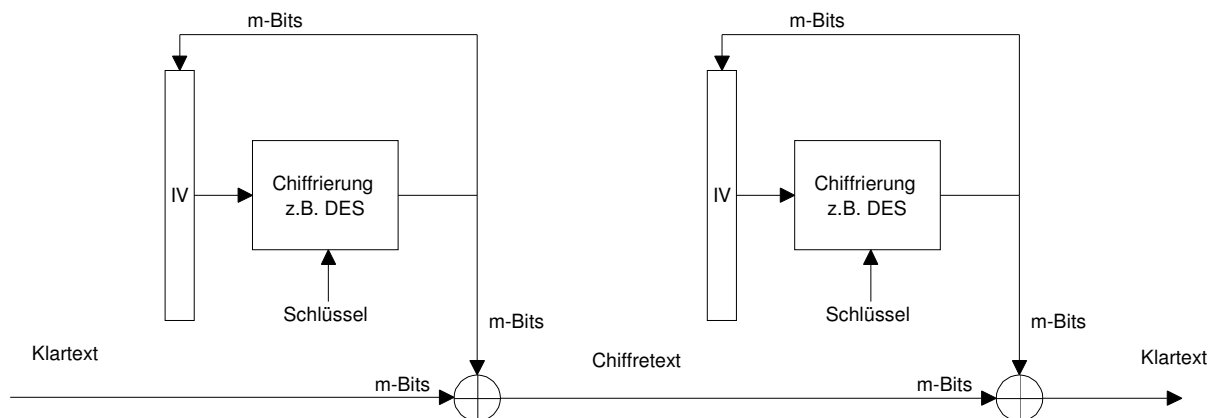


Abbildung 5-7: OFB Modus

5.2.2.4 CFB

Auch in diesem Modus wird eine Blockchiffre als Pseudozufallsgenerator verwendet. Es wird jeweils ein m-Bit Block des Output mit einem m-Bit Block Klartext XOR verknüpft. Der Chiffretext wird danach in das Initialisierungsregister geladen. Es ersetzt dort m-Bits des alten Inputs. Ein Chiffretextzeichen hängt vom zugehörigen Klartext, Schlüssel, Initialisierungsvektor und von allen vorhergehenden Klartextzeichen ab. Wegen der Rück- und Vorwärtskopplung des Chiffretextes gibt es im CFB-Modus im gegensatz zum OFB-Modus eine Fehlerfortpflanzung. Ein auf dem Übertragungsweg auftretender Fehler im Chiffretext wirkt sich zunächst auf das zugehörige Klartextzeichen so aus als ob es direkt übertragen worden wäre. Danach wird das fehlerhafte Chiffretextzeichen beim Empfänger durch das Inputregister der Blockchiffre geschoben. Solange es dieses Register nicht verlassen hat, werden aufgrund des Avalanche - Effektes etwa 50 % der dechiffrierten Klartextbits verfälscht.

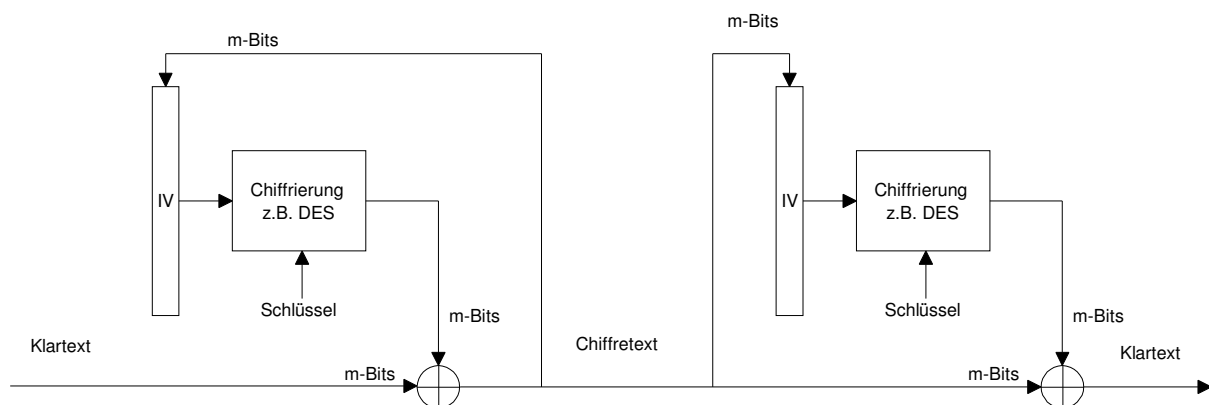


Abbildung 5-8: CFB Modus

5.3 Die Sicherheit von DES

Die Sicherheit von DES wurde lange Zeit in Frage gestellt. Es gab viele Spekulationen über die Schlüssellänge, Anzahl Iterationen und die S-Boxen. Gerade die S-Boxen galten als besonders mysteriös. Viele Zahlen und kein ersichtlicher Grund weshalb genau diese Zahlen an den und den Positionen in den Tabellen stehen. Obwohl IBM behauptet das die Gestalt der S-Boxen das Resultat von 17 Jahren Kryptanalyse sei, haben viele Leute den Verdacht das die NSA eine Hintertür in den Algorithmus eingebaut hat. Im Jahre 1978 untersuchte ein Komitee des US-Senates die Angelegenheit. Das Komitee sprach die NSA von dem Verdacht frei auf irgendeine unzulässige Art am Entwurf des Algorithmuses beteiligt gewesen zu sein. Die Regierung veröffentlichte die Einzelheiten jedoch nie, darum sind noch immer viele Leute skeptisch.

Aus diesem Grund entwickelte man verschiedenste Varianten von DES. z.B. 3DES, DES mit unabhängigen Teilschlüsseln, DESX, Crypt(3), GDES, DES mit alternativen S-Boxen, RDES und DES mit schlüsselabhängigen S-Boxen.

Die Technik macht laufend fortschritte. Und immer schnellere Prozessoren werden verfügbar. Beachtet man die Schlüssellänge so kann man sagen das man mit einem Brute-Force Angriff den Schlüssel finden kann. (man probiert alle 2^{56} Schlüssel aus). Alles was man dazu braucht ist ein Klartext und ein Chiffretext und natürlich eine Maschine die alle Schlüssel ausprobert. Um eine Maschine zu bauen die durchschnittlich 3,5 Stunden für einen Schlüssel braucht hätte man 1993 1 Mio. Dollar bezahlen müssen. DES ist so weit verbreitet, dass es sicherlich irgendwo in einem Keller einen DES – Knacker hat. Außerdem fallen die Kosten für eine solche Maschine laufend. Darum wird der DES im Laufe der Zeit nur unsicherer.

5.4 Kombination von Blockchiffren

Blockalgorithmen kann man auf viele verschiedene Arten miteinander kombinieren. Der Grund dafür ist, dass man Versucht die Sicherheit zu erhöhen ohne dabei einen neuen Algorithmus verwenden zu müssen. DES ist relativ sicher, obwohl der Schlüssel nicht gerade sehr lang ist. Wie ich im vorherigen Abschnitt erwähnt habe, kann man mittels Brute Force Angriff den DES in einer relativ kurzen Zeit knacken. Hätte man jedoch längere Schlüssel wäre dieser Angriff nicht gerade sehr praktisch.

Es gibt verschiedenste Kombinationstechniken. Eine davon ist die Mehrfachverschlüsselung. Dabei Verschlüsselt man den gleichen Klartext mehrmals mit dem gleichen Algorithmus, aber mit unterschiedlichen Schlüsseln. Bei der Kaskadierung benutzt man verschiedene Algorithmen.

5.4.1 Doppelte Verschlüsselung

Man kann die Sicherheit eines Blockalgorithmus durch doppelte Chiffrierung eines Blocks mit zwei verschiedenen Schlüsseln erhöhen.

$C = E(K_2, E(K_1, M))$ für die Chiffrierung

$M = D(K_2, D(K_1, C))$ für die Dechiffrierung

Falls der Blockalgorithmus eine Gruppe bildet so gib es immer einen Schlüssel K_3 mit

$C = E(K_2, E(K_1, M)) = E(K_3, M)$

Das würde bedeuten, dass eine Doppelte Verschlüsselung mit K_1 und K_2 einer Verschlüsselung mit K_3 entspräche. Darum wäre eine Doppelverschlüsselung mit K_1 und K_2 sinnlos. Man hätte genauso gut mit K_3 verschlüsseln können. Dadurch würde man den Aufwand eines Angriffes nicht erhöhen. Dies trifft aber für den DES nicht zu. Für einen Brute Force Angriff wären 2^{2n} Versuche erforderlich. (doppelte Chiffrierung mit 2 Verschiedenen Schlüsseln) Bei DES ergäbe das 2^{112} Versuche.

Es gibt andere Angriffsmethoden um die Doppelverschlüsselung zu knacken. Und diese Varianten brauchen nur noch $2^{(n+1)}$ Versuche. Es gibt zwar einige Schwierigkeiten bei der Umsetzung. Jedoch reicht das aus um die Doppelverschlüsselung für nutzlos zu erklären.

5.4.2 Dreifache Verschlüsselung

Eine Dreifachverschlüsselung kann mit zwei oder mit drei verschiedenen Schlüsseln erfolgen. Bei beiden wird die Verschlüsselung nach folgendem Schema gemacht: encrypt-decrypt-encrypt (EDE). Entschlüsselt wird in der umgekehrten Reihenfolge. Also entschlüsselt man wie folgt: decrypt-encrypt-decrypt. Diese Verfahren wurden entwickelt um die Sicherheit zu erhöhen. Dank dieser Methode wird das Knacken von Blockchiffren erheblich erschwert. Dieses Schema zur Ver- und Entschlüsselung wird vom Triple-DES (3DES) verwendet.

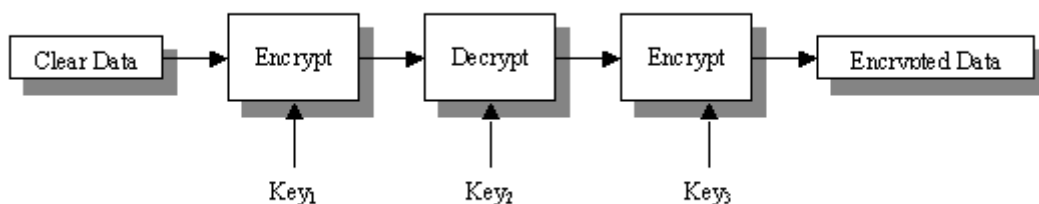


Abbildung 5-9: EDE-Schema

5.4.2.1 Dreifachverschlüsselung mit zwei Schlüsseln

Hat man einen Schlüssel der Länge n so kann man mit der Dreifachverschlüsselung einen Schlüssel der Länge $2 \cdot n$ haben. Die Ver- und Entschlüsselung gehen wie folgt:

$$C = E(K_1, D(K_2, E(K_1, M))) \quad \text{für Verschlüsselung}$$

$$M = D(K_1, E(K_2, D(K_1, C))) \quad \text{für Entschlüsselung}$$

Wählt man beide Schlüssel gleich, so entspricht das einer Einfachen Verschlüsselung. Aber auch dieses Verfahren kann man in $2^{(n-1)}$ Schritten knacken. Jedoch braucht man dazu 2^n gespeicherte Blöcke von Vorberechneten Resultaten. Es existieren jedoch Möglichkeiten um mit zwei Schlüsseln trotzdem sicher zu sein. Das Verfahren heisst Triple Encryption with Minimum Key (TEMK). Der Trick besteht darin aus zwei Schlüsseln K_1 und K_2 den neuen Schlüssel K_3 abzuleiten.

5.4.2.2 Dreifachverschlüsselung mit drei Schlüsseln

Bei der Dreifachverschlüsselung sollte man drei verschiedene Schlüssel verwenden. Dadurch erhöht man den Aufwand eines Angriffs.

$$C = E(K_3 D(K_2, E(K_1, M))) \quad \text{für Verschlüsselung}$$

$$M = D(K_3 E(K_2, D(K_1, C))) \quad \text{für Entschlüsselung}$$

Der beste Angriff der mittels Speicherplatz Laufzeit spart braucht $2^{(2n)}$ Schritte und Speicherplatz für 2^n Blöcke.

Kapitel 6: Diffie Hellmann

Der Algorithmus von Diffie Hellmann eignet sich für die Verteilung von Schlüsseln über unsichere Leitungen. Dies ist z.B. bei symmetrischen kryptographischen Verfahren nötig. So können zum Beispiel Person A und Person B einen gemeinsamen Schlüssel erzeugen. Der Algorithmus war der erste Public-Key-Algorithmus der Patentiert wurde (1976). Die Sicherheit beruht auf dem Problem der Berechnung diskreter Logarithmen.

6.1 Das Protokoll

Die Mathematik hinter dem Ganzen ist denkbar einfach. Alice und Bob einigen sich zunächst auf eine Grosse Primzahl p und eine Zahl g . An die Zahl g werden noch Bedingungen gestellt. Die Zahl g muss modulo p primitiv sein. Was bedeutet das nun? g ist primitiv zu p wenn wir eine Ganze Zahl i finden können für die gilt $g^i = j \pmod p$ für alle Werte von j von 1 bis $p-1$. Ein kleines Zahlenbeispiel: 2 ist nicht primitiv zu 7, weil die Menge der Potenzen von 2 von 1 bis 6 mod 7 = {2,4,1,2,4,1} ist. {3,5} kommen nicht vor. Darum ist die Bedingung nicht für alle j erfüllt. 3 ist primitiv zu 7, weil die Menge der Potenzen von 3 von 1 bis 6 mod 7 = {3,2,6,4,5,1} ist. Und $2 < g < p-2$

Das Protokoll verläuft so ab:

- Alice wählt eine grosse zufällige Zahl x und sendet Bob $X=g^x \pmod p$
- Bob wählt eine grosse zufällige Zahl y und sendet Alice $Y=g^y \pmod p$
- Alice berechnet $ka=Y^x \pmod p$
- Bob berechnet $kb=X^y \pmod p$
- ka und kb sind gleich Alice und Bob haben nun den Gleichen Schlüssel

Die Zahlen die sich Alice und Bob gedenkt haben (a bzw. b) wurden beim Schlüsselaustausch nie übermittelt!

Jemand der den Kommunikationskanal abhört kennt nur die Werte p , n , X , Y . Nun stellt sich die Frage kann ein Aussenstehender daraus den Schlüssel $g^{(x*y)} \pmod p$ berechnen?

- Könnte die aussenstehende Person den diskreten Logarithmus von $g^a \pmod p$ zur Basis g berechnen, so hätte sie die Zahl a und somit den Schlüssel $(g^b)^a \pmod p$. Das sollte aber praktisch sehr schwer sein.
- Es ist nicht bekannt das man aus der Kenntnis von g , $g^a \pmod p$ und $g^b \pmod p$ den Wert $g^{ab} \pmod p$ berechnen kann ohne Berechnung von diskreten Logarithmen zur Basis g .

Die Sicherheit des Systems hängt sehr stark von der Wahl von g und p ab. Die Zahl $(p-1)/2$ sollte auch eine Primzahl sein. Am wichtigsten ist, dass die Zahl p groß genug ist. Denn die Sicherheit des Systems beruht auf dem Problem der Faktorisierung von Zahlen derselben Größenordnung wie p . g kann beliebig gewählt werden.

6.1.1 Zahlenbeispiel für DH

g hat den Wert 4 und p hat den Wert 11

- Alice denkt sich für a die Zahl 3
- Bob denkt sich für b die Zahl 5
- Alice berechnet $X=4^3 \bmod 11 = 9$
- Bob berechnet $Y=4^5 \bmod 11 = 1$
- Alice sendet Bob $X=9$
- Bob sendet Alice $Y=1$
- Alice berechnet $ka=1^3 \bmod 11 = 1$
- Bob berechnet $ka=9^5 \bmod 11 = 1$

Beide haben nun den gleichen Schlüssel. ($ka = kb = k = 1$)

Nehmen wir nun mal an eine Drittperson würde sich die vier Zahlen $p = 11$, $g = 4$, $X = 9$ und $Y = 1$ schnappen. Wie könnte sie nun die Geheimnisse von Bob oder Alice herausfinden? Der einzig bekannte Weg das zu tun ist die Berechnung von den diskreten Logarithmen zur Basis g .

1. $4^1 \bmod 11 = 4$
2. $4^2 \bmod 11 = 5$
3. $4^3 \bmod 11 = 9$ Bingo! Der Angreifer hat nun das Geheimnis von Alice! $a = 3$

Nun kann er leicht den gemeinsamen Schlüssel berechnen. $k = Y^a \bmod p \rightarrow 1^3 \bmod 11 = 1$ Der Angreifer kennt nun den Schlüssel k .

Man sieht, dass die Sicherheit vom DH sehr stark von der Länge der verwendeten Zahlen abhängig ist. Typischerweise verwendet man für die Zahl p 1024-Bit. Das entspricht 309 Dezimalstellen im 10er System.

6.2 Berechnungsmethode

Um einen sicheren DH zu machen muss man in der Lage sein $a^d \bmod n$ auch für große d und n zu berechnen. Mit den Standarddatentypen die man in C hat, stößt man schnell einmal an Grenzen wenn man es eins zu eins programmiert. Eine der möglichen Berechnungsmethoden ist die square and multiply methode.

6.2.1 Die square and multiply Methode

Wir wollen für $a \in \mathbb{Z}$ und $d, n \in \mathbb{N}$ die Potenz $a^d \bmod n$ berechnen. Ist

$d = d_0 + d_1 * 2 + \dots + d_r * 2^r$ mit $d_i \in \{0,1\}$ die Binärentwicklung von d , so gilt

$$a^d \equiv \prod_{j=0}^r (a^{2^j})^{d_j} \bmod n$$

und $r = \lfloor \log_2 d \rfloor$, falls $d_r \neq 0$ ist. Wir definieren für $i = 0, 1, 2, \dots, r$

$$c_i = \left\lfloor \frac{d}{2^i} \right\rfloor = d_i + d_{i+1} \cdot 2 + \dots + d_r \cdot 2^{r-i}, \quad x_i \equiv a^{2^i} \pmod{n}, \quad y_i \equiv \prod_{j=0}^i (a^{2^j}) \pmod{n}.$$

Wie man sieht sind c_i, d_i, x_i und y_i durch folgende Rekursionsformeln gegeben

$$c_0 = d, d_0 \equiv c_0 \pmod{2}, x_0 = a, y_0 = a^{d_0} \text{ und}$$

$$c_i = \left\lfloor \frac{c_{i-1}}{2} \right\rfloor, d_i \equiv c_i \pmod{2}, x_i \equiv x_{i-1}^2 \pmod{n}, y_i \equiv y_{i-1} x_i^{d_i} \pmod{n}.$$

Man hat dann natürlich $y_r \equiv a^d \pmod{n}$, also kann man $a^d \pmod{n}$ mit obigen Rekursionsformeln berechnen. Wegen $d_i \in \{0, 1\}$ muss man dabei nur quadrieren und multiplizieren. Daher nennt man das Verfahren auch die square-and-multiply-Methode zum Potenzieren.

Wir wollen z.B. $2^{123456789} \pmod{987654321}$ berechnen

Tabelle 6-1: Berechnung

i	c_i	d_i	x_i	y_i
0	123456789	1	2	2
1	61728394	0	4	2
2	30864197	1	16	32
3	15432098	0	256	32
4	7716049	1	65536	2097152
5	3858024	0	344350012	2097152
6	1929012	0	392547562	2097152
7	964506	0	282069526	2097152
8	482253	1	706746679	149611286
9	241126	0	123137257	149611286
10	120563	1	851417971	279277817
11	60281	1	345041113	911451224
12	30140	0	769901713	911451224
13	15070	0	759344764	911451224
14	7535	1	483551350	189213602
15	3767	1	923405899	206271470
16	1883	1	964006387	212405648
17	941	1	91103341	969951539
18	470	0	469653595	969951539
19	235	1	528234745	155914325
20	117	1	142714780	566254001
21	58	0	282045658	566254001
22	29	1	109238839	264855578
23	14	0	982955794	264855578
24	7	1	106586737	970718579
25	3	1	652313308	483053927
26	1	1	924386425	804307517

Also ist das Resultat 804307517

Algorithmus: Für $n, d \in \mathbb{N}$ und $a \in \mathbb{Z}$ soll $a^d \bmod n$ berechnet werden.

1. Setze $c := d, x := a$. Setze $y := 1$, falls $c \equiv 0 \bmod 2$, sonst $y := a$.
2. Setze $c := \left\lfloor \frac{c}{2} \right\rfloor, x := x^2 \bmod n$.
3. Ist $c \equiv 1 \bmod 2$, setze $y := xy \bmod n$.
4. Ist $c = 1$, gib y als Ergebnis aus und beende das Verfahren, sonst gehe zu Schritt 2.

In Programmpaketen mit Langzahlarithmetik ist die das square-and-multiply Verfahren zur Berechnung von $x = a^d \bmod n$ normalerweise implementiert. (GMP, NTL, MIRACL)

Kapitel 7: Implementation

7.1 TMS320C6711 Entwicklungsumgebung

7.1.1 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



Abbildung 7-1: TMS320 DSP Board

Für unsere Anwendung stellt die Fliesskomma Rechenleistung der CPU (A) und der, an der CPU aufgesteckte Analog/Digital Wandler (C1 / C2) die zentralen Einheiten dar. Auf den Position (B1) wäre noch ein langsamerer Analog/Digital Wandler. Wir haben uns aber für den leistungsfähigeren Wandler entschieden weil:

- Die Taktrate höher ist und so auch eine höhere Bitrate des Wasserzeichens erreicht werden kann.
- Wir mindestens zwei Einlese- und ein Ausgabekanal brauchen. Nur der aufgesteckte Analog/Digital Wandler ist in Stereo (zwei Kanäle) ausgeführt.

7.1.2 Analog Digital Wandler TDMX326040A

Einige Eckdaten dieser Karte:

- 16 Bit Auflösung
- 24 kHz Abtastung oder mit externem Clock variabel einstellbar
- Stereo

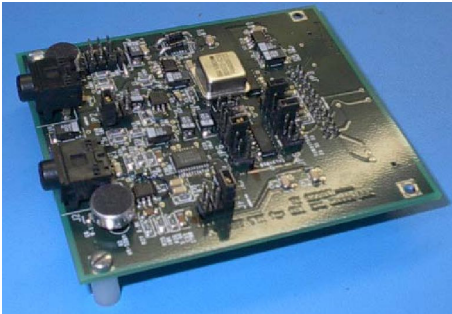


Abbildung 7-2: Analog Digital Wandler TDMX326040A

7.1.3 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.

7.2 Versuchsaufbau

Für unseren Versuchsaufbau benutzen wir zwei DSP Boards. Für das Mikrofon und den Kopfhörer haben wir die Kopfhörer Sennheiser m@b 40, welche ein integriertes Mikrofon haben, genommen. Wichtig ist, dass das Mikrofon eine Vorspannung erhält. Dazu bastelten wir einen Adapterstecker. Das Pluskabel muss auf die AVDD Speisung des AD Wandlers gesteckt werden.

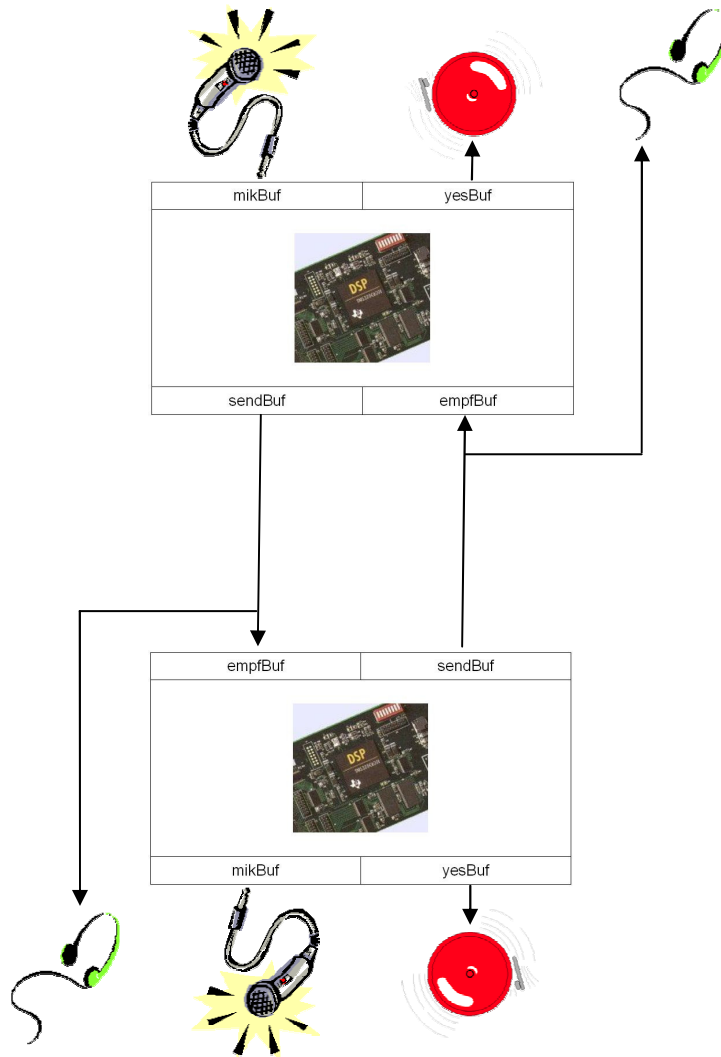


Abbildung 7-3: Versuchsaufbau

Im Programm sind die verschiedenen Buffer mit den Ein- und Ausgängen des AD Wandlers verknüpft. Die folgende Tabelle zeigt die Zuordnungen.

Tabelle 7-1: Bufferzuordnung

Buffer:	Zugehörige Buchse auf AD Wandler TDMX326040A:
empfBuf	IN, linker Kanal
mikBuf	IN, rechter Kanal
sendBuf	OUT, linker Kanal
yesBuf	OUT, rechter Kanal

7.3 Programmaufbau

7.3.1 Das Projekt Tranceiver

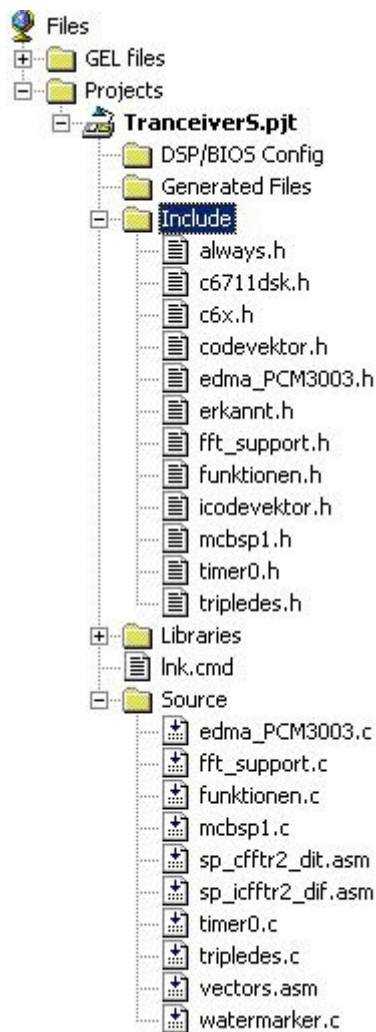


Abbildung 7-4: Programmaufbau

Das Projekt ist recht umfangreich. Wichtigstes File ist watermarker.c. Hier befindet sich das Main sowie sämtliche entscheidende Funktionen.

7.3.2 Compilereinstellungen

Wir haben gemerkt, dass man sich mit den Compilereinstellungen die ganze Freude kann verderben. Für verschiedene Projekte sind verschiedene Compilereinstellungen zu empfehlen. Nötig sind in unserem Fall die nachher abgebildeten.

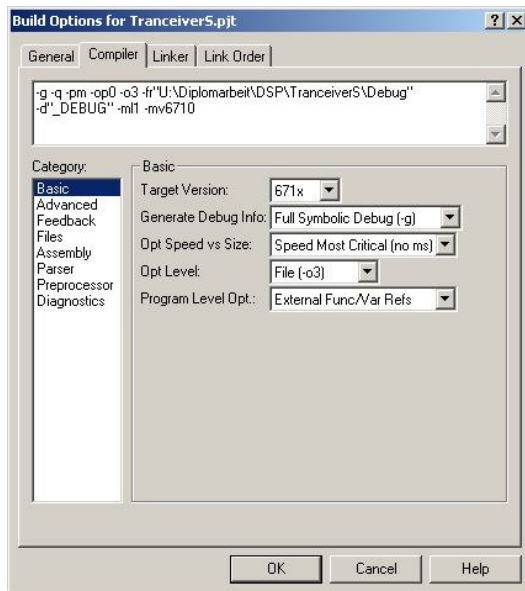


Abbildung 7-5: Compilereinstellung Basic

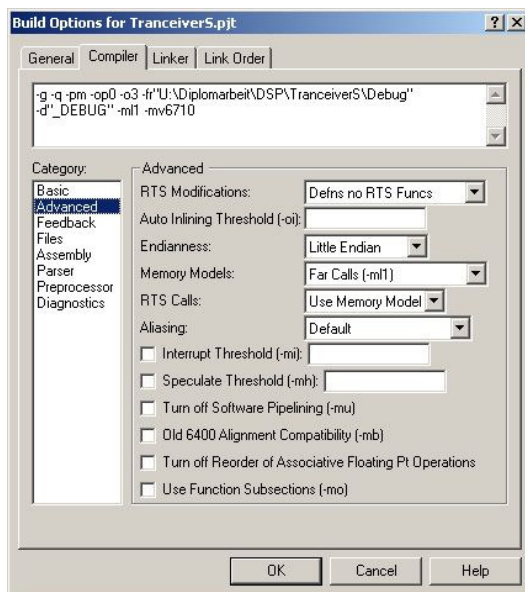


Abbildung 7-6: Compilereinstellung Advanced

Da wir einige Funktionen aus Platzgründen ins SRAM verschoben haben, müssen wir beim Compiler besonderen wert auf Geschwindigkeit geben. Sonst bearbeitet uns der Prozessor die Daten nicht mehr in der gewünschten Zeit und die ausgegebenen Daten klingen verhackt. Auch ist der Programmcounter PC sehr beschränkt und kann nicht ins SRAM springen, da das SRAM riesig im vergleich zur Adresse des IRAM sind. Deshalb ist unter der Option Advanced ebenfalls noch auf Far Calls (-ml1) zu schalten. Nicht aber (-ml3), welches wieder zu einer zu langsamen Maschine führt.

7.3.3 Files und ihre Funktionen

In der folgenden Tabelle sind alle Files aufgeführt und es ist bemerkt wozu sie dienen.

Tabelle 7-2: Files im Projekt Tranceiver

Filename:	Funktion:
edma_PCM3003.c	Edma Einstellungen für die Ein- und Ausgabe von Frames über den AD Wandler TMDX326040A
fft_support.c	Hier stehen Funktionen zur Berechnung von: <ul style="list-style-type: none"> • Gen_w_r2(): Berechnet Twiddlefaktoren für FFT • Bit_Rev(): Muss nach jedem FFT und vor jedem IFFT gemacht werden, damit die Werte im Buffer wieder richtig geordnet sind.
funktionen.c	Hier sind Hilfsfunktionen wie: <ul style="list-style-type: none"> • tofftBuf(): Wandelt Buffer mit realen Werten in einen für den FFT brauchbaren Buffer • absolut(): Berechnet den Betrag • initBuf(): Löscht alle Werte und setzt = 0 • maxIndex(): Gibt die Maximalstelle eines Buffers zurück • circShift(): Zirkulare Verschiebung der Werte eines Buffers • get_ioport(): Für die Schalterstellung • get_bdrev(): Für die Schalterabfrage
mcbasp1.c	Interrupt Einstellungen für die Ein- und Ausgabe von Frames
sp_cfft2_dit.asm	FFT, handoptimiert in Assembler
sp_icfft2_dif.asm	IFFT, handoptimiert in Assembler
timer0.c	Erzeugt externen Clock für eine andere Abtastrate als 24 kHz. Kann mit Jumper auf der AD Karte eingestellt werden.
tripleDES.c	Stellt die Funktionen für eine 3DES Codierung zur Verfügung. Die wichtigsten sind: <ul style="list-style-type: none"> • OFB_Encrypt() • OFB_Decrypt()
vectors.asm	Hier werden der Interrupt für den AD Wandler sowie der Boardreset definiert
watermarker.c	Hier ist das Main und die Funktionen: <ul style="list-style-type: none"> • main() • Edma_Isr(): Definiert was bei einem Interrupt passiert • insertWatermark(): Fügt das Wasserzeichen ein • schalter(): Schaltereinstellungen und Entscheidungen wie der Peak einzufügen ist, 3DES • detektion(): Findet das Wasserzeichen • synchronisation(): Reagiert auf modem() • modem(): Gibt verschiedene Sinus aus
codevektor.h	Enthält den Codevektor für die Wasserzeichengenerierung
always.h	Datentypen Typedef
c6711dsk.h	Braucht es immer
c6x.h	Braucht es immer
erkannt.h	Enthält Array mit Computerstimme die „erkannt“ sagt

7.3.4 Flussdiagramm

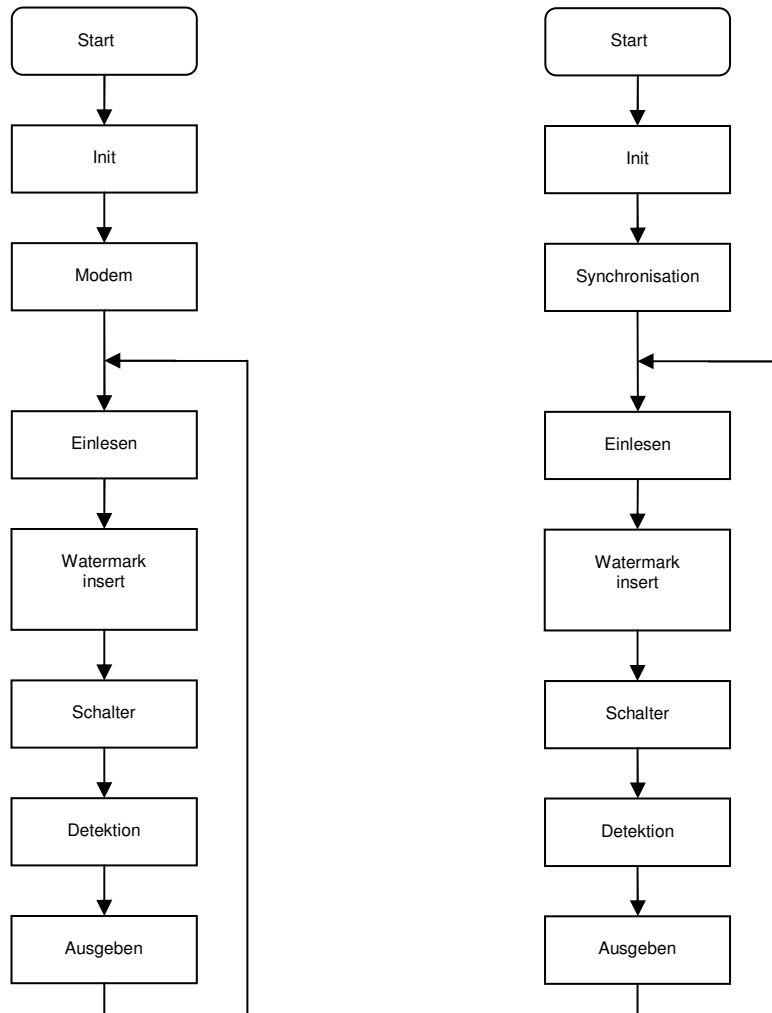


Abbildung 7-7: Flussdiagramme Sender, Empfänger

Wie man sieht, haben wir zwei Programme erstellt die sich im Kern aber nicht voneinander unterscheiden. Eigentlich wollten wir dasselbe Programm auf beiden DSPs laufen lassen. Wegen der Synchronisation beim Starten der Programme mussten wir aber eine Unterteilung in einen Sender und Empfänger vornehmen.

7.4 Ein- Ausgabe mit EDMA

Die Daten Ein- und Ausgabe wird mit der Audio Daughter Card TMDX 326040A gemacht. Weil wir zwei Kanäle benötigten, konnten wir den Codec AD535 der sich bereits auf dem DSK befindet nicht benutzen.

7.4.1 Audio Daughter Card

Die Daughter Card ist mit dem McBSP1 vom DSP verbunden. Die Daten werden via EDMA ein- und ausgegeben. Der Clock für die Abtastung kann vom DSP generiert werden oder man nimmt den Oszillator der bereits auf der Karte drauf ist. Jedoch ist dann die Abtastzeit nicht variabel sondern ist fest eingestellt auf 24 kHz für Stereo und 48kHz für Mono.

7.4.2 Enhanced Direct Memory Access (EDMA)

Wieso benutzt man den EDMA Controller um Daten zu transferieren? Man kann Daten auf zwei Arten bewegen:

1. CPU
2. EDMA

Wenn man den EDMA benutzt dann initialisiert die CPU den EDMA und kann danach andere Operationen ausführen. Müsste die CPU jedes Sempel einzeln abholen, würde sie erheblich an Rechenkapazität verlieren.

Mit dem EDMA Controller kann man Daten von irgendeiner Adresse zu einer anderen bewegen. Zum Beispiel aus dem Internen Speicher an eine Peripherie, oder aus dem Externen in den Internen Speicher. Der EDMA Transfer von Daten hat noch weitere Vorteile

- Datentransfer mit keinem Overhead
- Die CPU und der EDMA können unabhängig voneinander arbeiten
- EDMA und CPU können nie gleichzeitig auf den gleichen Speicherbereich zugreifen, der Programm Memory Controller verhindert das.

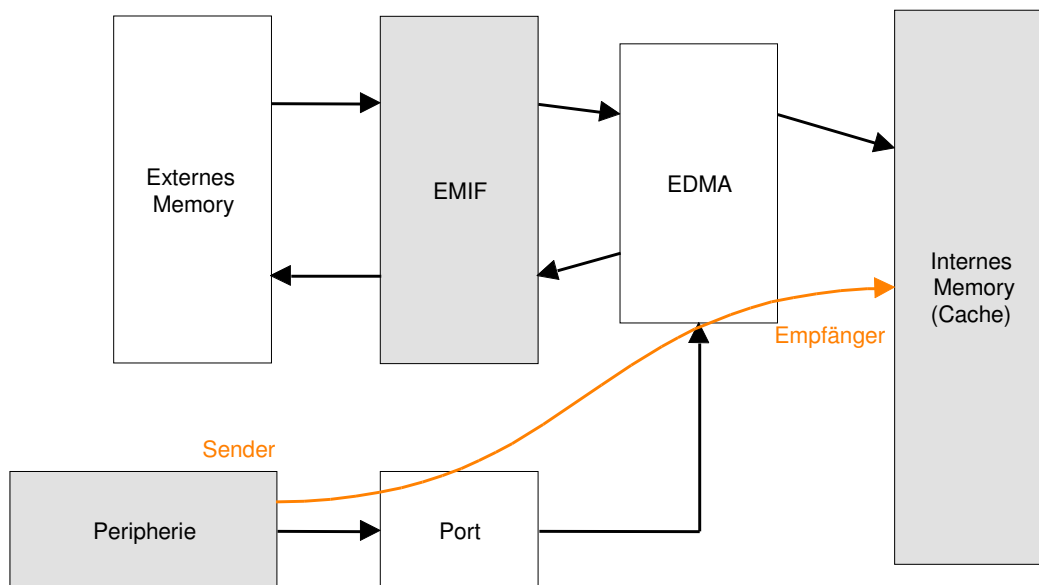


Abbildung 7-8: EDMA Transfer von Peripherie in internen Speicher

Der EDMA Controller auf dem C6711 hat folgende Features:

- 16 Kanäle
- 1 Hilfskanal für HPI (für den Programmierer nicht zugreifbar)
- einen Quick DMA (QDMA)
- Elementtransfer in einem Maschinenzyklus

Die Leistungsfähigkeit vom EDMA kann durch folgendes gemindert werden:

- EDMA Stalls durch gleichzeitiges anfordern von Transfers mit gleicher Priorität
- Wenn der EDMA tiefere Priorität als die CPU für den Zugriff auf das IRAM hat

7.4.2.1 Einige Definitionen

Um die Funktionsweise des EDMA zu verstehen ist es nützlich zuerst ein paar Definitionen zu machen.

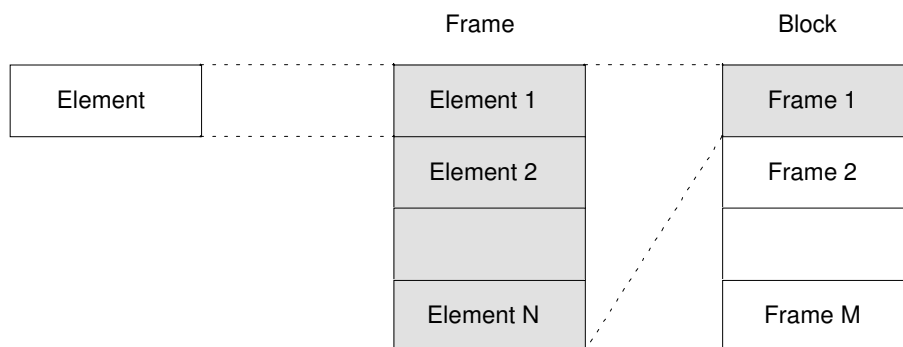


Abbildung 7-9: Unterschied zwischen Element, Frame und Block

Das Element ist der Grundbaustein. Ein Element kann 8, 16 oder 32 Bits sein. Die Grösse vom Element wird im OPT Register mit den Bits ESIZE definiert. Ein Frame besteht aus mehreren Elementen. Auch die Länge des Frames muss definiert werden. Dazu dienen die Bits ELECNT. Der Block schliesslich besteht aus mehreren Frames. Seine Grösse wird mit den Bits FRMCNT definiert. All diese Definitionen müssen vor dem EDMA Transfer gemacht werden.

7.4.3 Wie der EDMA funktioniert

Die verschiedenen EDMA Kanäle werden in der PaRAM Tabellen konfiguriert. Die Tabellen sind ein 2 Kbyte Block von internem Speicher. Jede Tabelle besteht aus 6 Langworteinträgen (6 mal 32-Bit). Man kann maximal 85 Einträge machen.

Das EDMA PaRAM besteht aus Einträgen für die Kanäle und Reload Kanäle.

Parameter RAM

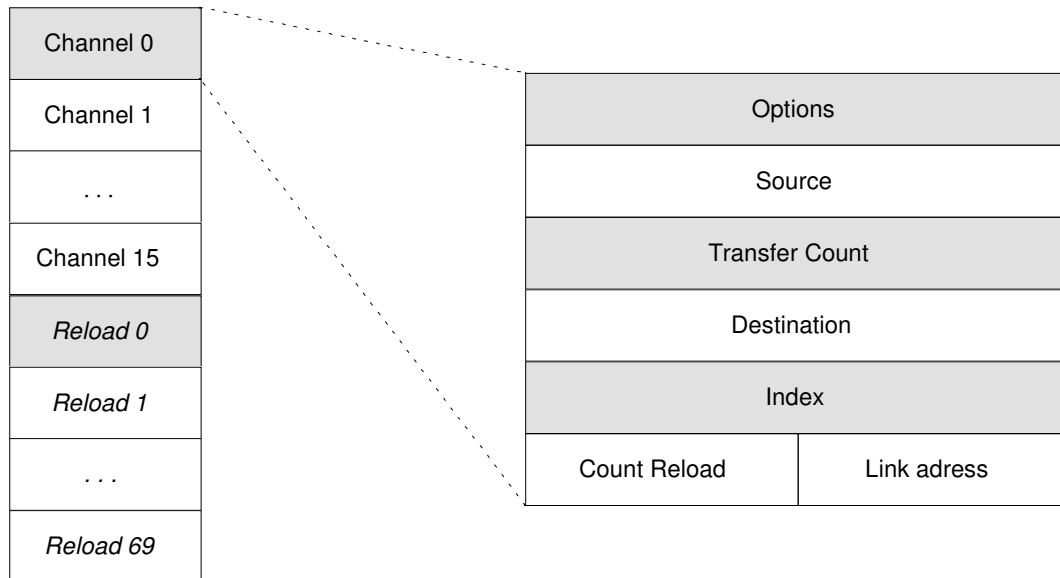


Abbildung 7-10: Parameter RAM (PaRAM)

Die Konfigurationstabelle für den EDMA enthält folgendes:

Options OPT	
SRC Adresse (Sender)	
Array/Frame count (FRMCNT)	Element count (ELECNT)
DST Adresse (Empfänger)	
Array/frame index (FRMIDX)	Element index (ELEIDX)
Element count reload (ELERLD)	Link adress (LINK)

Abbildung 7-11: Konfigurationstabelle EDMA

- Options Transfer Optionen
- Source Adresse Die Adresse von der die Daten Transferiert werden
- Element count Anzahl Elemente pro Frame
- Frame count Anzahl Frames pro Block minus 1
- Element index Adress Offset von Elementen in einem Frame
- Link Adresse Die PaRAM Adresse die die Parameter für den nächsten Transfer hat

Noch etwas zum Link Befehl. Der Link befehl ermöglicht es den EDMA nach erfolgtem Transfer zu reinitialisieren. Dabei werden die Parameter für den neuen Transfer mit der LINK Adresse angegeben. Dadurch wird dieser PaRAM Block geladen und ausgeführt. Das erlaubt einem sehr elegant die folgenden Transfers zu bestimmen. Diese Link Listen sind vor allem dann nützlich wenn man Ping - Pong Buffer hat, oder wenn man die eingelesenen Daten sortieren will.

Parameter RAM

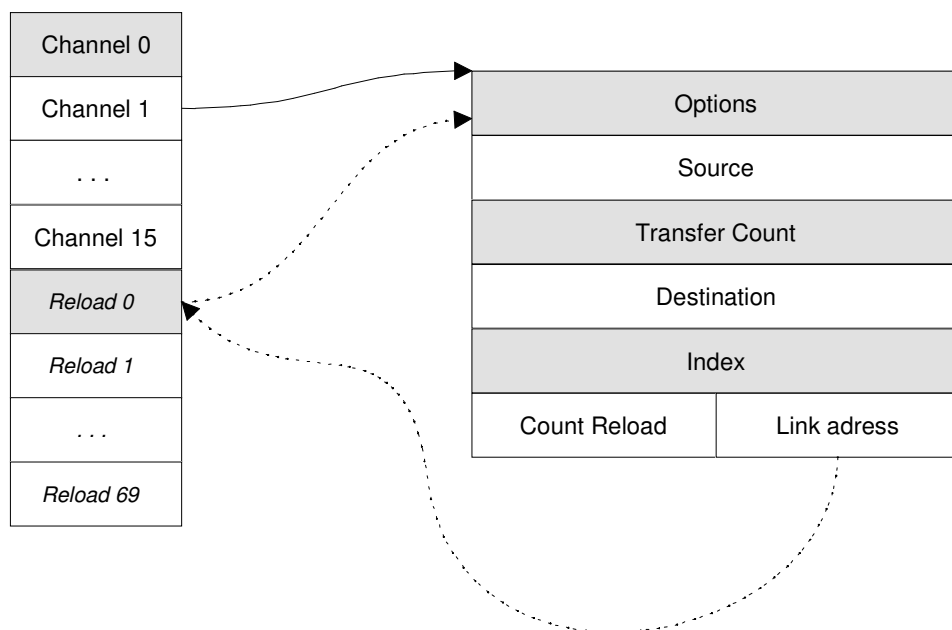


Abbildung 7-12: Linken von EDMA Transfers

7.4.4 EDMA Parameter (Options)

Im Channel Options Register sind die Transport Optionen.

31	0									
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINIT	TCC	RSVD	LINK	FS

Abbildung 7-13: EDMA Parameter Options (OPT Register)

Tabelle 7-3: EDMA Parameter Options

Label:	Beschreibung:
PRI	Prioritäts Levels für den EDMA <ul style="list-style-type: none"> ▪ 001 → hohe Priorität ▪ 010 → niedrige Priorität
ESIZE	Bestimmt die Element Grösse (8/16 oder 32 Bit) <ul style="list-style-type: none"> ▪ 00 → 32 Bit ▪ 01 → 16 Bit ▪ 10 → 8 Bit
2DS	Source Dimension
SUM	Source Adressen update Modus <ul style="list-style-type: none"> ▪ 00 → fixe Adresse ▪ 01 → Source Adresse wird inkrementiert ▪ 10 → Source Adresse wird dekrementiert ▪ 11 → Source Adresse wird durch das Frame/index bestimmt (FRMIDX)
2DD	Destinations Dimension
DUM	Destinations Adressen update Modus, gleich wie SUM
TCINT	Transfer complete interrupt enable <ul style="list-style-type: none"> ▪ wenn auf 0 werden die Bits im Register CIPR nicht gesetzt ▪ wenn auf 1 werden die Bits im Register CIPR gesetzt. Dabei wird der Wert der sich im TCC Register befand reingeschrieben. Daraus kann man herausfinden welcher Transfer gerade zuende ging.
TCC	Transfer complete code Mit diesen Bits wird das CIPR Register geladen wenn ein Transfer complete Interrupt erfolgt
LINK	Man muss LINK auf 1 setzen wenn man EDMA Transfer miteinander Linken will
FS	Frame Synchronisation

Source / Destination:

In diesem Feld wird die Startadresse des Speicherbeginns von Source und Destination angegeben.

Transfer Count:

Bei Transfer Count stehen zwei Sachen

- Frame Count [31:16] → Anzahl Frames in einem Block
- Element Count [1:15] → Anzahl Elemente in einem Frame, gültige Werte sind zwischen 1 und 65535

Element/Frame Index

Der Element Index bzw. Frame Index wird für Adressmodifikationen verwendet. Der Element Index wird für die Berechnung der Adresse des nächsten Elementes verwendet. Der Frame Index macht einen Versatz zum folgenden Frame in einem Block.

Element Count Reload (ELERLD)

Der Wert in diesem Feld wird in den Element Count geladen wenn ein Frame übertragen wurde.

Linkadresse

Die Linkadresse gibt an von wo aus dem Speicher (PaRAM) die Parameter wiedergeladen werden müssen. Dabei wird die Adresse (16-Bit) zum Offset 0x01A0xxxx addiert um die absolute Adresse zu berechnen.

EDMA Synchronisation

Es gibt zwei Möglichkeiten einen EDMA Transfer einzuleiten

- die CPU leitet den EDMA Transfer ein
- Ereignis gesteuert (Event triggered)

In diesem Fall wird das Ereignis in einem Register angezeigt. Das Event Register (ER) triggert dann den Transfer. In der folgenden Tabelle ist aufgelistet welche Events EDMA Transfers triggern können.

Alle EDMA Transfers sind an einen Event gebunden (Synchronisiert).

Tabelle 7-4: EDMA Channel

EDMA Channel Number	Event Acronym	Event Description
0	DSPINT	Host to DSP interrupt
1	TINT0	Timer 0 interrupt
2	TINT1	Timer 1 interrupt
3	SD_INT	EMIF SDRAM timer interrupt
4	EXT_INT4	External interrupt pin 4
5	EXT_INT5	External interrupt pin 5
6	EXT_INT6	External interrupt pin 6
7	EXT_INT7	External interrupt pin 7
8	EDMA_TCC8 †	EDMA transfer complete code 1000b interrupt
9	EDMA_TCC9 †	EDMA TCC 1001b interrupt
10	EDMA_TCC10 †	EDMA TCC 1010b interrupt
11	EDMA_TCC11 †	EDMA TCC 1011b interrupt
12	XEVT0	McBSP0 transmit event
13	REVT0	McBSP0 receive event
14	XEVT1	McBSP1 transmit event
15	REVT1	McBSP1 receive event

EDMA Interrupt Generierung

Der EDMA Controller ist dafür zuständig Interrupts auszulösen wenn ein Transfer beendet wurde. Der EDMA Controller löst einen Interrupt (EDMA_INT) für alle 16 Kanäle aus. Man kann mit dem CIPR Register herausfinden von welchem Kanal er ausgelöst wurde.

Jeder Kanal hat ein Options Register. In dem werden folgende Sachen definiert.

- TCINT → falls man Interrupts für diesen Kanal auslösen will (nach beendetem Transfer)
- TCC → 4-Bit Code der in das CIPR Register geladen wird nach erfolgtem Interrupt

Um die Interrupts freizugeben müssen folgende Register gesetzt werden

- CIEn muss auf ,1' gesetzt werden , für den entsprechenden Event
- EERn muss auf ,1' gesetzt werden für den entsprechenden Event
- TCINT und TCC wie oben beschrieben

Programmierung vom EDMA

Es gibt drei Methoden den EDMA zu programmieren

- 1: durch direktes schreiben in die EDMA Register
- 2: mit CSL
- 3: graphisch mit dem DSP/BIOS

Ich machte es mit der ersten Variante weil ich die CSL nicht benutzen wollte, da ich das DSP/BIOS für meine Applikation nicht brauchte.

7.4.5 Ping Pong Buffer

Der Ping Pong Buffer wird in der Echtzeitdatenverarbeitung häufig eingesetzt. Das Prinzip ist einfach. Während der Pong Buffer bearbeitet wird, kann der Ping Buffer mit neuen Daten geladen werden.

Da mein Programm Daten ein – und ausgibt benötige ich insgesamt vier Daten Buffer. Zwei für das Einlesen und zwei für die Ausgabe.

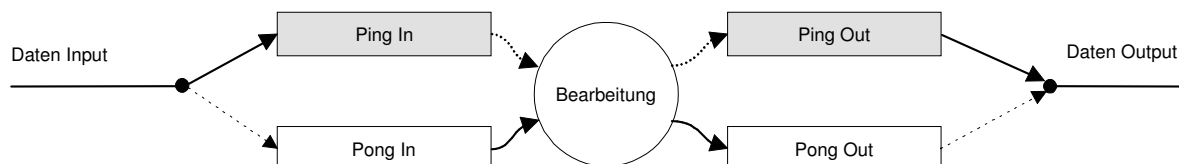


Abbildung 7-14 Ping Pong Buffer

Dabei ist der Ablauf folgender: Die Daten werden in den Ping In Buffer geladen. Gleichzeitig wird der Ping Out Buffer ausgegeben. Während dieser Zeit kann man den Pong buffer bearbeiten. Man nimmt die Daten aus dem Pong In Buffer, verarbeitet die Daten mit dem Algorithmus den man geschrieben hat und kopiert das Resultat in den Pong Out Buffer. Danach wechselt das Ganze. Ein heikler Punkt ist noch das man immer wissen muss welcher Buffer gerade gefüllt wird und welchen man bearbeiten kann.

7.4.5.1 Implementation vom Ping Pong Buffer

Die Implementation vom Ping Pong Buffer kann sehr elegant gelöst werden (dank EDMA reload Tabellen). Die Events werden durch McBSP1 receive/transmit ausgelöst.

Ablauf für den Empfang

Im Parameter RAM wird die Tabelle EVENT 15 für den Empfang der Daten geladen → Initialisierung vom EDMA. Die Source der Daten ist das Empfangsregister vom McBSP1- (McBSP1_DRR). Die Destination ist der Ping In Buffer. Wenn nun der EDMA gestartet wird, weiss der EDMA Controller von wo nach wo er die Daten transferieren muss. Wenn er den Transfer abgeschlossen hat, löst er einen Interrupt aus.

In der Interruptrutine muss das Bit vom Event 15 gelöscht werden. Danach sieht der EDMA Controller nach ob das Bit LINK gesetzt ist. Da es gesetzt ist, liest er die Adresse die im Link Register steht und ladet den entsprechenden Parameter RAM Block. In diesem Beispiel ist das der EDMA Reload 0 Block. Die Source der Daten ist noch immer die gleiche, aber die Destination hat geändert. Nun wird in den Pong In Buffer geschrieben. Wenn dieser Transfer beendet ist, löst er wieder einen Interrupt aus. Danach ladet er die PaRAM Tabelle EDMA Reload 1. Die Source ist wieder die gleiche, aber die Destination hat wieder geändert. Jetzt wird wieder in den Ping In Buffer geschrieben. Der EDMA befindet sich in einem Kreislauf der durch das Verlinken entstanden ist. Das garantiert einen kontinuierlichen Datenfluss. Die gleichen Überlegungen gelten für das Senden von Daten.

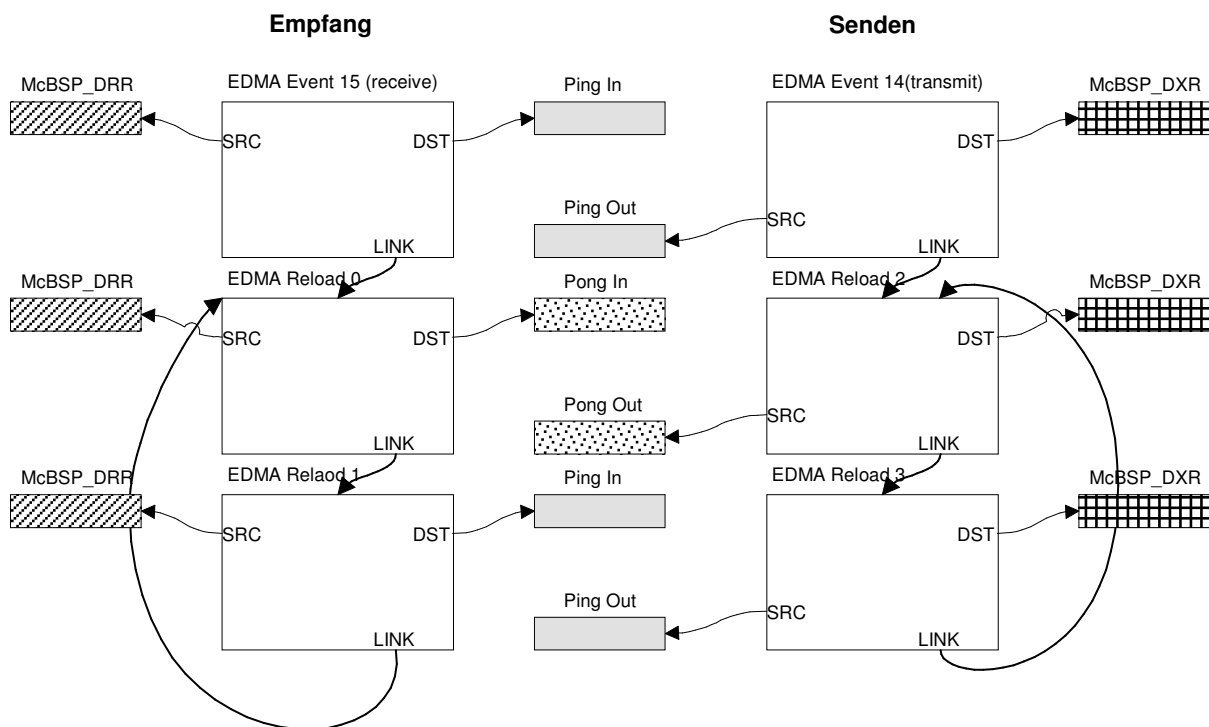


Abbildung 7-15: Ablauf Ping Pong

7.4.6 Demoapplikation Daten Ein/Ausgabe

Auf der CD findet man eine Demoapplikation die Daten mit dem Ping Pong Buffer ein- und ausliest. Der Ordner heisst PCM_EDMA_2. Der Ablauf des Programms ist folgender. EDMA wird initialisiert. Danach befindet sich das Programm in einem while Loop. In diesem while Loop gibt es zwei Abfragen. Die eine Abfrage kontrolliert ob das pong_ready gesetzt ist. Die andere ob ping_ready gesetzt ist. Wenn eines gesetzt ist geht das

Programm in die Abfrage und kopiert den In Buffer in den Out Buffer. Die beiden Variablen ping/pong_ready werden im EDMA Interrupt gesetzt.

Der folgende C Code zeigt einen Pseudocode vom Main in der Demoapplikation.

```
while(1)
{
    if( pong_ready )                // ist Pong Buffer bereit zur Bearbeitung
    {
        Pong Buffer Bearbeiten mit Algorithmus
        Resultat in Pong Out Buffer kopieren
    }
    if( ping_ready )                // ist Ping Buffer bereit zur Bearbeitung
    {
        Ping Buffer Bearbeiten mit Algorithmus
        Resultat in Ping Out Buffer kopieren
    }
}
```

7.5 Das Frequenzverfahren

7.5.1 Einfügen des Wasserzeichens

Genau wie im Matlab und in der Theorie beschrieben, wird in der Funktion insertWatermark() das Wasserzeichen eingefügt. Der wesentliche Unterschied zwischen Matlab und dem C Code ist nur die Sprache. Sämtliche Operationen sind in derselben Reihenfolge ausgeführt und auch die Resultate zeigen nur leichte Unterschiede zum Beispiel ist Einfügefaktor im C Code wesentlich höher. Das hängt mit der Normierung der Werte zusammen.

```
void insertWatermark(const float* inBuf, float* outBuf, int peakStelle)
{
    int i;

    circShift(codeVektor, codeSBuf, peakStelle, BLOCK_SIZE);

    // fft(mikBuf)
    tofftBuf(inBuf, fftBuf);
    DSPF_sp_cfftr2_dit(fftBuf, w, BLOCK_SIZE);
    Bit_Rev(fftBuf, BLOCK_SIZE);

    // Einbetten des Wasserzeichen
    for (i = 0; i < 2 * BLOCK_SIZE; i = i + 2)
    {
        fftBuf[i] = fftBuf[i] * codeSBuf[i/2];
        fftBuf[i+1] = fftBuf[i+1] * codeSBuf[i/2];
    }

    // Signal mit eingebettetem Wasserzeichen erstellen
    Bit_Rev(fftBuf, BLOCK_SIZE);
    DSPF_sp_icfftr2_dif(fftBuf, w, BLOCK_SIZE);
    for (i = 0; i < 2048; i = i + 2)
    {
        waterSBuf[i/2] = fftBuf[i];
    }

    for (i = 0; i < BLOCK_SIZE; i++)
    {
        outBuf[i] = inBuf[i] * 30000 + 20.0 * waterSBuf[i];
    }

    return;
}
```

7.5.2 Detektion des Wasserzeichens

Auch die Funktion `detektion()` entspricht im Wesentlichen dem Matlabcode. Gleichzeitig wird hier aber noch die Auswertung gemacht. Das heisst, die LEDs werden angesteuert. Da die Peaks mit 3DES verschlüsselt gesendet werden, muss diese 3DES hier wieder durch die Funktion `OFB_Decrypt()` rückgängig gemacht werden. Nach Aufgabenstellung sollen wir eine Uhr senden. Diese ist einfach als ein Counter realisiert.

```
void detektion(const float* inBuf)
{
    int i;
    // Schritt 1: Filtern
    filter(inBuf, waterEBuf, BLOCK_SIZE);

    // Schritt 2: abs(fft(y))
    tofftBuf(waterEBuf, fftBuf);
    DSPF_sp_cfft2r2_dit(fftBuf, w, BLOCK_SIZE);
    Bit_Rev(fftBuf, BLOCK_SIZE);
    for (i = 0; i < 2048; i = i + 2)
    {
        waterEBuf[i/2] = absolut(fftBuf[i], fftBuf[i+1]);
    }
    // Schritt 3: Kreuzkorrelation mit Codevektor
    tofftBuf(waterEBuf, fftBuf);
    DSPF_sp_cfft2r2_dit(fftBuf, w, BLOCK_SIZE);
    Bit_Rev(fftBuf, BLOCK_SIZE);
    for (i = 0; i < 2048; i = i + 2)
    {
        fftBuf[i] = fftBuf[i] * icodeVektor[i/2];
        fftBuf[i+1] = fftBuf[i+1] * icodeVektor[i/2];
    }
    Bit_Rev(fftBuf, BLOCK_SIZE);
    DSPF_sp_icfft2r2_dif(fftBuf, w, BLOCK_SIZE);
    for (i = 0; i < 2048; i = i + 2)
    {
        waterEBuf[i/2] = fftBuf[i];
    }
    for (i = 0; i < 1024; i++)
    {
        peakEBuf[i] = peakEBuf[i] + waterEBuf[i];
    }
    // Schritt 4: Nach 25 Frames -> Peak detektieren und peakBuf reseten
    if (icount % anzFrame == 0)
    {
        // Peakstelle
        i = maxStelle(peakEBuf, BLOCK_SIZE / 2);
        peakE[icount/anzFrame] = OFB_Decrypt(i, key); // 3DES retour
        // Schritt 5: LED Ausgabe
        *(volatile unsigned int*)IO_PORT = 0x07000000; // LEDs aus
        if (peakE[icount/anzFrame] == 300)
        {
            *(volatile unsigned int*)IO_PORT = 0x03000000; // USER_LED3
        }
        if (peakE[icount/anzFrame] == 200)
        {
            *(volatile unsigned int*)IO_PORT = 0x05000000; // USER_LED2
        }
        if (peakE[icount/anzFrame] == 100)
        {
            *(volatile unsigned int*)IO_PORT = 0x06000000; // USER_LED1
        }
        if ((peakE[icount/anzFrame] - peakE[(icount/anzFrame)-1]) == 1)
        {
            *(volatile unsigned int*)IO_PORT = 0x00000000; // alle LEDs
        }
        // Clean peakEBuf
        for (i = 0; i < BLOCK_SIZE; i++)
        {
            peakEBuf[i] = 0.0;
        }
    }
}
```

Für Testzwecke haben wir mit Schaltereinstellungen aber auch die Möglichkeit konstante Peaks an den Stellen 100, 200 und 300 zu schicken. Ausserdem füllen wir das peakE Array mit den empfangenen Peaks. So können wir im Watch Window von CCS die empfangenen Werte kontrollieren.

7.5.3 Künstliches Rauschen

Wir haben in Tests festgestellt, dass wenn kein Signal anliegt oder nur sehr leise ist, das Wasserzeichen schlecht detektiert wird. Im Sender wird im Frequenzbereich eine Multiplikation des Codevektors mit dem Signal gemacht. Ist das Signal leise (gleich Null) entspricht das einer Multiplikation mit Null. Es ist also verständlich, dass so kein Wasserzeichen eingefügt wird.

Als Lösung geben wir dem Signal ein künstliches Rauschen bei. Es ist ein 1024 Array und in einem H-File abgelegt. Das Rauschen ist aber so leise, dass man es nicht hört. Die Resultate mit Rauschen waren wesentlich überzeugender.

7.5.4 Synchronisation

Wir haben festgestellt, dass unser System der Empfänger nicht immer korrekt detektiert. Schnell einmal wurde klar, dass es eine Synchronisation braucht.

7.5.4.1 Wieso braucht es eine Synchronisation

Wie im Theorieteil erwähnt, mussten wir doch eine Synchronisation einbauen. Das hat zwei wesentliche Gründe:

- Verbesserung der Detektionsqualität
- 3DES Maschine muss im exakt gleichen Moment gestartet werden.

Die Synchronisation muss aber nicht genau auf ein Sample sein. Frameweise genügt, weil sich sämtliche Berechnungen auf 25 Frames beziehen.

Ohne Synchronisation ist uns beim Testen ein Fehler aufgefallen, den wir unbedingt korrigieren mussten. Weder mehr Frames für einen Peak noch eine Vergrößerung des Einfügefaktors nützten etwas. Dank einem hohen Einfügefaktor ist zwar jeder Peak gefunden worden dafür zum Teil zweimal. Zum Beispiel peakE[5] und peakE[6] haben beide den Wert 5. Eigentlich müsste in peakE[6] der Wert 6 stehen.

Name	Value	Type	Radix
peakE	0x80007	int[100]	hex
peakE[0]	0	int	dec
peakE[1]	1	int	dec
peakE[2]	2	int	dec
peakE[3]	3	int	dec
peakE[4]	3	int	dec
peakE[5]	5	int	dec
peakE[6]	5	int	dec
peakE[7]	7	int	dec
peakE[8]	7	int	dec
peakE[9]	9	int	dec
peakE[10]	10	int	dec
peakE[11]	11	int	dec
peakE[12]	12	int	dec
peakE[13]	13	int	dec
peakE[14]	13	int	dec
peakE[15]	15	int	dec
peakE[16]	16	int	dec
peakE[17]	17	int	dec
peakE[18]	18	int	dec
peakE[19]	19	int	dec
peakE[20]	20	int	dec

Abbildung 7-16: Fehler bei der Detektion

Beim derzeitigen System werden 25 Frames mit demselben Peak gesendet. Der schlimmste Fall ist, wenn der Empfänger genau 12 Frames noch mit dem letzten Peak auswertet und 13 bereits den neuen Peak enthalten. Je nach gesendeten Daten gewinnt einer die Überhand und es kann möglich sein, dass ein Peak zweimal erkannt wird. Die einzig richtige Lösung ist hier eine Synchronisation.

Gesendet:

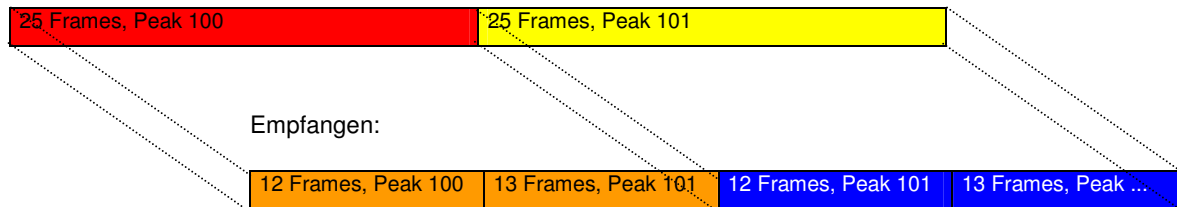


Abbildung 7-17: Verschiebung der auszuwertenden Frames mangels Synchronisation

7.5.4.2 Funktionsweise der Synchronisation

Eigentlich wollten wir in beiden DSP Boards dasselbe Programm laufen lassen. Zu Beginn ist aber eben eine Synchronisation nötig. Dazu teilen wir die Boards wieder in einen Sender und einen Empfänger auf mit unterschiedlichen Programmen. Hier erst einmal den entscheidenden C Code für den Sender. Gleich nach Programmstart wird der Sender in die Funktion `modem()` springen. Diese Funktion erzeugt 25 verschiedene Sinus mit einer für jedes Frame typischen Frequenz. Die Variable `icount` wird nach jedem gesendeten Frame im `mainloop` um eins erhöht.

```
void modem()
{
    int i;

    for (i = 0; i < BLOCK_SIZE; i++)
    {
        sendBuf[i] = 30000 * sin(2 * PI * i * 15 * icount / 1024);
        yesBuf[i] = 30000 * sin(2 * PI * i * 15 * icount / 1024);
    }
    if(icount >= 25)
    {
        icount = 0;
    }
    modem_ready--;
}
```

Der Empfänger wird vor dem Sender gestartet. Der Empfänger befindet sich nun in einem Wait Status. Er wartet, bis am Eingang Frames kommen, die einen Sinus enthalten. Dazu muss er nach jedem Frame in die Funktion `synchronisation()` springen. Ein Sinus ergibt im Spektrum einen Dirac Impuls an der Stelle der Frequenz des Sinus. Entscheidend ist, dass der Peak an einer Stelle ist, wo er auch erwartet werden darf. Das sind alle Stellen, die durch 15 dividierbar sind ohne Rest. Als Vorsichtsmaßnahme wird der Fall Peak an der Stelle Null ausgeschlossen. Tests haben gezeigt, dass sich sonst `synch_ready` dauernd erniedrigt, weil wenn kein Signal anliegt der Peak an die Stelle Null zu liegen kommt. Nach der Funktion `Synchronisation`, weiss der Empfänger genau, welchen Sinus er erwisch hat. Er weiss, dass der Sender 25 Sinus schickt und kann sich die Wartezeit bis zum Start der Kommunikation errechnen.

```
int synchronisation(const float* inBuf)

{
int i;
int modemicount;
int wait;

tofftBuf(inBuf, fftBuf);
DSPF_sp_cfft2_dit(fftBuf, w, BLOCK_SIZE);
Bit_Rev(fftBuf,BLOCK_SIZE);
for (i = 0; i < 2048; i = i + 2)
{
waterEBuf[i/2] = absolut(fftBuf[i],fftBuf[i+1]);
}
synchPeak = maxStelle(waterEBuf,BLOCK_SIZE / 2);
if((synchPeak % 15 == 0) && (synchPeak != 0))
{
modemicount = synchPeak / 15;
wait = 25 - modemicount;
synch_ready--;
}
return wait + 2 + 4 * 25;
}
```

Weil jedes Board wegen den Ping Pong Buffern noch eine Verzögerung eines Frames erzeugen und der Sender 5 mal das „Modem“ schickt, ist wait so gewählt worden. Ausserdem waren Tests mit dieser Konfiguration am erfolgreichsten.

7.6 3DES

Der 3DES Code wurde nicht selbst geschrieben. Ich fand entsprechende C-Files im Internet. Man kann sie von <http://people.qualcomm.com/karn/> oder von <http://www.ka9q.net/code/des/index.html> holen. Der OFB Modus wurde selbst implementiert. Der C Code musste noch leicht geändert werden. Alle Variablen die vom Datentyp long waren wurden auf int umgetauft. Man findet ein Demoprogramm im Ordner 3DES.

7.7 Diffie Hellmann Schlüsselerzeugung

Um einen sicheren DH zu machen muss man in der Lage sein mit Grossen Zahlen zu rechnen. Da es sehr aufwendig ist habe ich nach C-Code gesucht. Schliesslich wurde ich fündig. Man kann die Multiprecision Integer and Rational Arithmetic C/C++ Library (MIRACL) von <http://indigo.ie/~mscott> holen. Die Library war nicht dazu gedacht auf einem DSP zu laufen. Jedoch ist sie einfach aufgebaut und leicht portierbar. Die einzige Schwierigkeit bestand am Anfang das mirdef.h File richtig aufzusetzen. Darin sind die nötigen Definitionen für die Library. Die Library setzt voraus, dass der long Datentyp 64 Bit hat. Beim DSP c6711 ist das jedoch nicht der Fall. Bei ihm ist nämlich der long 40-Bit lang. Darum muss man als Grunddatentyp für die Berechnungen den short verwenden. Wenn man den long mit 64 Bit hätte, würden die Berechnungen auch noch einiges schneller gehen.

7.7.1 Einbindung der Library

Die Library verbraucht sehr viel Speicher. Sie hätte im Internen Speicher gar keinen Platz. Darum musste der Code der Library in den Externen Speicher gelinkt werden. Im Linker Command File haben wir eine Sektion mit dem Namen „gosram“ gemacht.

```
SECTIONS
{
    "vectors"    >    IRAM
    .cinit       >    IRAM
    .text        >    IRAM
    .stack       >    IRAM
    .bss         >    IRAM
    .const       >    IRAM
    .data        >    IRAM
    .far         >    SRAM
    .switch      >    IRAM
    .system      >    SRAM
    .tables      >    SRAM
    .cio         >    SRAM
    gosram       >    SRAM
}

```

Will man nun eine Funktion in das SRAM „zwingen“ so muss man im C-File vor der Definition der Funktion Folgendes schreiben:

```
#pragma CODE_SECTION(mr_padd,"gosram"); // dabei ist mr_padd der Funktionsnamen
```

Durch diesen Handgriff kann die Library verwendet werden ohne dabei die ganze .text Section in das SRAM zu verschieben. Aber in der Compileroption muss man noch Far Calls & Data (-ml3) auswählen. Sonst gibt es eine Fehlermeldung das der Program Counter (PC) nicht so weit springen kann. Leider stellten wir fest, dass das Projekt (Tranceiver.pjt) dann nicht mehr richtig läuft. Es scheint zu langsam zu sein. Obwohl der Code für diesen Teil des Programms im Internen RAM ist. Wahrscheinlich hat das mit dem PC zu tun.

7.7.2 Demoapplikation DH

Auf der CD befindet sich ein DH Projekt, das für sich alleine auf dem DSP funktioniert. Man findet die Demo im Ordner DH. Der Text den die Demo ausgibt sieht man im stdout Fenster. Der Text sieht wie folgt aus:

```
Diffie-Hellman Key exchange ....
```

```
Alice's offline calculation
```

```
Alice's secret is :1270050485000452110757173318331956417314622713174
```

```
Alice's public secret:
```

```
77399131521859548790055673215020818835909174771770796098022664782503250750588762481
78860469282728734446592515211426966028436898178919134043302568457076418351156293566
13971128117842161950752801766407427818999596267139151200427622651748119343709591418
29663170081001033468258577733513139271073088629515557325334
```

```
Bob's offline calculation
```

```
Bob's secret is :916486348387274072462284215915353216094578140965
```

```
Bob's public secret:
```

```
60284773815534388747952530178213382080220861256219083279567578233669804660194485883
19512448497954703333780744588835205223598312783531812807641678997215434490098050492
62317905554444106482677413482112213094339900750355650798876350478289222924798612254
21475362394087179893962302975022858340968435042195645203665
```

```
Alice calculates Key=
```

```
1346482694728074145410384539525548912028456613031730767834559842726840913359317
59003722335178840418659023147318394570295054913938238117982553049091272938777017692
57769594769205611797076366424606691733541608883737148135789580679849550631374441506
9963761051970454644341607385112835309249410419942056671478100596
```

```
Bob calculates Key=
```

```
1346482694728074145410384539525548912028456613031730767834559842726840913359317
59003722335178840418659023147318394570295054913938238117982553049091272938777017692
57769594769205611797076366424606691733541608883737148135789580679849550631374441506
9963761051970454644341607385112835309249410419942056671478100596
```

```
Alice and Bob's keys should be the same!
```

Kapitel 8: Auswertung

8.1 Bitrate des Wasserzeichensignals

Wir haben zwei Systeme realisiert. Einmal unser Standardmodell, welches 25 Frames für einen Peak benützt und andererseits eine Version welche nur 10 Frames für einen Peak auswertet. Wir empfehlen das Standardmodell weil die Genauigkeit und Störanfälligkeit wesentlich besser ist. Der Preis ist aber eine niedrigere Bitrate.

Tabelle 8-1: Daten für Bitratenberechnung des Standardmodells

Abtastfrequenz:	f_o	24 kHz
Framelänge:	l	1024 Samples
Anzahl Frames für ein Codewort:	k	25

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

$$T = \frac{1}{f_o} \cdot l \cdot k = \frac{1}{25000} \cdot 1024 \cdot 25 = 1.024s$$

$$B = \frac{n}{T} = \frac{9}{1.024} = 8.79 \text{ Bit / s}$$

Tabelle 8-2: Daten für Bitratenberechnung für Version mit 10 Frames

Abtastfrequenz:	f_o	24 kHz
Framelänge:	l	1024 Samples
Anzahl Frames für ein Codewort:	k	10

$$T = \frac{1}{f_o} \cdot l \cdot k = \frac{1}{25000} \cdot 1024 \cdot 10 = 0.4096s$$

$$B = \frac{n}{T} = \frac{9}{0.4096} = 21.97 \text{ Bit / s}$$

8.2 Störanfälligkeit und Fehler

8.2.1 Messungen

Es ist sehr schwierig, die Fehlerrate zu ermitteln, weil viele Faktoren einen Einfluss haben. Entscheidend sind:

- Die anliegenden Daten wie Sprache oder Musik.
- Der Einfügefaktor wie stark das Wasserzeichen beigemischt wird. Je stärker der Faktor desto eher hört man das Wasserzeichen, desto besser ist aber auch die Fehlerrate. Je nach anliegenden Daten kann man den Einfügefaktor variieren. Bei Musik hört man das Wasserzeichen wesentlich schlechter.
- Wie viele Frames für einen Peak verwendet werden.
- Die Synchronisation. Das System synchronisiert sich nur Frameweise. Je nach dem passen die einzelnen Samples genau oder nicht. Dieser Einfluss ist aber gering.

Wir haben in den Messungen verschiedene Einstellungen vorgenommen. Es variieren:

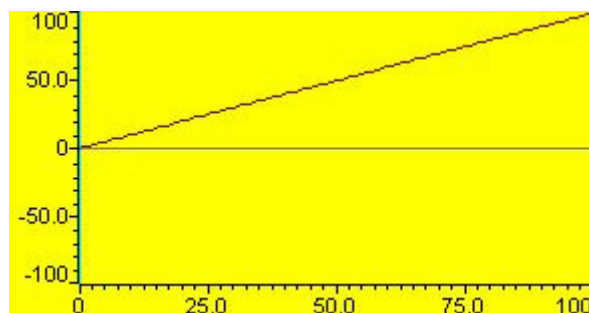
- Der Einfügefaktor
- Die anliegenden Daten
- Die Anzahl Frames für einen Peak

Bei jeder Messung wird die Uhr übertragen. Nach 103 Sekunden ist der peakE Buffer mit 100 empfangenen Werten gefüllt. Diese können wir grafisch darstellen oder auch mit dem Watch Window von CCS genau ansehen und so die Bitfehlerrate BER berechnen.

8.2.1.1 Messung 1

Tabelle 8-3: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	20
Daten:	Pink Floyd: Your possible pasts
Frames für ein Peak:	25



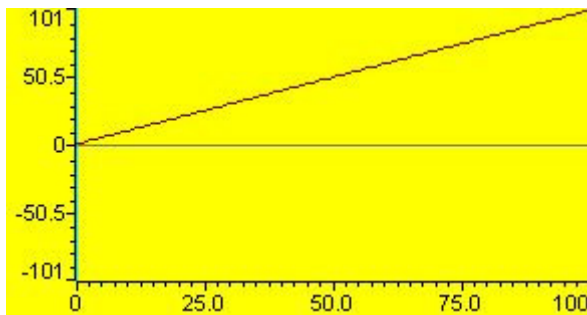
BER = 0 %

Abbildung 8-1: Messresultat 1

8.2.1.2 Messung 2

Tabelle 8-4: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	20
Daten:	Mikrofon angeschlossen, es wird von 1 bis 130 gezählt
Frames für ein Peak:	25



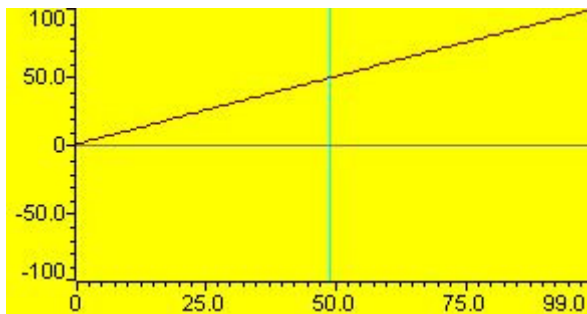
BER = 0 %

Abbildung 8-2: Messresultat 2

8.2.1.3 Messung 3

Tabelle 8-5: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	10
Daten:	Mikrofon angeschlossen, es wird von 1 bis 130 gezählt
Frames für ein Peak:	25



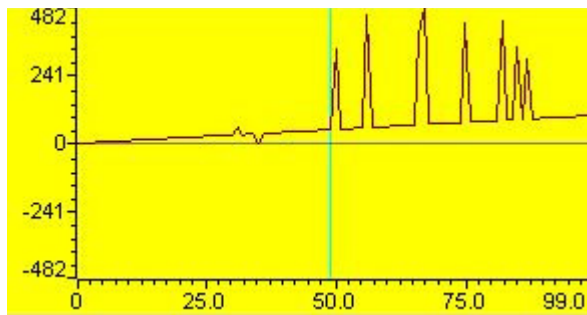
BER = 0 %

Abbildung 8-3: Messresultat 3

8.2.1.4 Messung 4

Tabelle 8-6: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	10
Daten:	Pink Floyd: Your possible pasts
Frames für ein Peak:	25



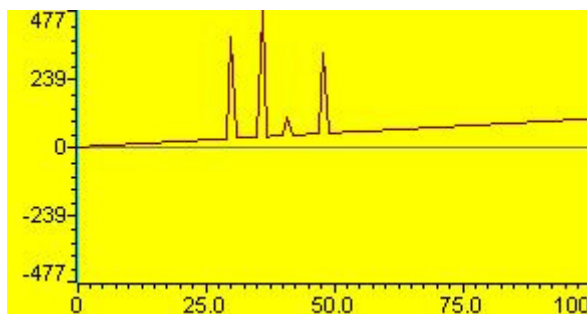
BER = 10 %

Abbildung 8-4: Messresultat 4

8.2.1.5 Messung 5

Tabelle 8-7: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	8
Daten:	Mikrofon angeschlossen, es wird von 1 bis 130 gezählt
Frames für ein Peak:	25



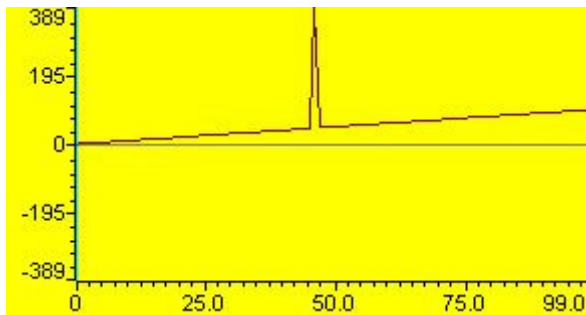
BER = 4 %

Abbildung 8-5: Messresultat 5

8.2.1.6 Messung 6

Tabelle 8-8: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	25
Daten:	Mikrofon angeschlossen, es wird von 1 bis 130 gezählt
Frames für ein Peak:	10



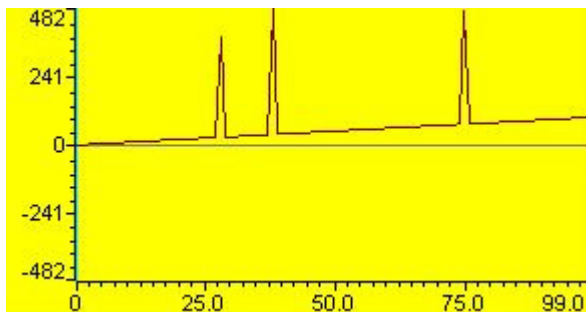
BER = 1 %

Abbildung 8-6: Messresultat 6

8.2.1.7 Messung 7

Tabelle 8-9: Messeinstellungen

Wer:	Einstellung:
Einfügefaktor	10
Daten:	Mikrofon angeschlossen, es wird von 1 bis 130 gezählt
Frames für ein Peak:	10



BER = 4 %

Abbildung 8-7: Messresultat 7

8.3 Tonqualität

Die Qualität der Verbindung ist stark vom Einfügefaktor abhängig. Etwa ab Faktor 10 ist das Wasserzeichen als leichtes Knacken wahrzunehmen. Die Kommunikation wird dadurch kaum gestört. Mühsam wird es ab einem Faktor der grösser als 25 ist. Subjektiv will ich behaupten, eine Kommunikation über ein Natel ist wesentlich mühsamer.

8.4 Die beste Einstellung

Wir empfehlen:

- Einfügefaktor 12
- 25 Frames für einen Peak

Damit hat man grosse Sicherheit, dass das Wasserzeichen gut detektiert wird und auch die Sprachqualität angemessen ist.

Kapitel 9: Stand der Dinge

Stand vom 10.12.2003

9.1 Allgemein

Leider konnten wir nicht alle geforderten Ziele bis zum gesetzten Termin erreichen. Viele grundlegende Funktionen werden von der Software erfüllt.

9.2 Das funktioniert

Folgende Funktionen werden durch unsere Arbeit fehlerfrei ausgeführt:

- Ein- Ausgabe von Frames mit EDMA Transfer
- Wir haben eine duale Kommunikation. Das heisst ein Telefongespräch ist mit unserem Versuchsaufbau möglich
- Das Wasserzeichen wird korrekt eingefügt
- Das Wasserzeichen wird richtig detektiert
- Das Wasserzeichen ist als Uhr implementiert
- Die übermittelten Peaks sind mit 3DES verschlüsselt
- Sender und Empfänger arbeiten synchron
- Das Wasserzeichen kann ein- und ausgeschaltet werden
- Das Wasserzeichen ist nicht hörbar

9.3 Das funktioniert nicht

Leider konnte bis zum gesetzten Zeitpunkt der:

- DH Schlüsselaustausch

nicht mehr realisieren werden. Als einzelnes Projekt funktioniert der DH mit 1024 Bit langen Zahlen. Beim Einfügen ins Projekt Tranceiver stellte uns CCS mehrere Steine in den Weg. Compilereinstellungen, Programmcounter und .text Section sind hier die Übeltäter.

Kapitel 10: Schlussfolgerungen

10.1 Kritischer Rückblick

Die Diplomarbeit war ohne Zweifel sehr interessant, abwechslungsreich und fordernd. Da wir dieses Thema bereits in der zweiten Semesterarbeit behandelten, war die Motivation auch dementsprechend hoch. Dennoch konnten wir leider, trotz enorm hohen Zeiteinsatzes das Ziel nicht vollständig erreichen. Gerne hätten wir uns eines korrekten Diffie Hellmann Schlüsselaustausch erfreut.

Rückblickend stellen wir fest, dass das Ziel hoch gesteckt war. Grosse Mühe bereitete uns zu Beginn der Arbeit eine seriöse Ein- und Ausgabe von Samples. Mehrfach änderten wir die verwendeten Methoden oder gar den zu verwendenden analog digital Wandler.

Mit der Implementation in C hat es gut geklappt bis der Datenplatz im IRAM aufgebraucht war und die Geschwindigkeit noch keine entscheidende Rolle spielte. Weitere C Files für den Diffie Hellmann Schlüsselaustausch haben uns aber die Leistungsgrenze des DSPs und des Entwicklungsboards aufgezeigt. Es folgte ein harter, beschwerlicher Weg mit Hilfe von Compilerereinstellungen und Map Files das Projekt zu optimieren.

Die Arbeit war sehr lehrreich, konnten wir uns doch mit Problemen aus verschiedenen Gebieten der digitalen Signalverarbeitung befassen. Mit 3DES und Diffie Hellmann sind wir auch in Gebiete von Digitale Medien vorgestossen. Wir haben in der Programmierung mit C viel dazugelernt und die Macht von Code Composer Studios kennen gelernt. Beide haben wir auch viele Versuche gemacht und uns in der Schule der logischen Schlussfolgerungen bemüht.

Vergessen darf man nicht die Teamarbeit. Wir haben in dieser Diplomarbeit hervorragend zusammen gearbeitet und uns ergänzt. Dank den Semesterarbeiten konnten wir unsere individuellen Stärken gekonnt umsetzen. Nach dieser Diplomarbeit fühlen wir uns gewappnet für weitere interessante Projekte im Berufsleben.

10.2 Verbesserungen

Im Laufe der Arbeit haben wir festgestellt, dass es nicht so einfach ist dem DSP ein grosses Projekt zu füttern. Wir sind aber überzeugt, dass mit mehr Zeit auch ein Diffie Hellmann Schlüsselaustausch erfolgreich zu unserem Projekt beigefügt werden kann.

Nicht optimal ist auch das Rauschen auf den Kanälen. Es liegt am Versuchsaufbau und den verwendeten Mikrofonen. Sie liefern ein schwaches Eingangssignal, so dass wir es enorm verstärken müssen. Durch geeignete Mikrofone oder einen Vorverstärker kann dieses Problem gut behoben werden.

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 und an trüben Sonntagen zu guten Raumklima beigetragen haben, sei ein weiteres Merci gewidmet.

Herr Schuster hat uns mit seinen Tipps einen guten Dienst erwiesen. Seine kompetente Beratung war stets ein Genuss und hat oft den gewünschten Erfolg gebracht. „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, 10.12.2003

Markus Gloor

Mario Jurcevic

Kapitel 11: Literaturverzeichnis

11.1 Bücher

- [1] Alan V. Oppenheim / Ronald W. Schaffer, „Zeitdiskrete Signalverarbeitung“, R. Oldenburg Verlag München Wien, 1995
- [2] Bruce Schneier, „Angewandte Kryptographie“, Addison-Wesley Publishing Company, 1996
- [3] Matlab, „Matlab The Language of Technical Computing“, The MathWorks, 1996
- [4] Norber Ryska / Siegfried Herda, „Kryptographische Verfahren in der Datenverarbeitung“, Springer Verlag, 1980
- [5] Peter Vary / Ulrich Heute / Wolfgang Hess, „Digitale Sprachsignalverarbeitung“, Teubner, 1998
- [6] Rulph Chassaing, „DSP applications using C and the TMS320C6x DSK“, John Wiley & Sons Inc., New York, 2002
- [7] W. Fumy / M. P. Riess, „Entwurf und Analyse symmetrischer Kryptosysteme“, Oldenburg 1988

11.2 PDF-Files

- [1] Alex Schüeli, "Digitale Signalverarbeitung", Skript HSR
- [2] Anthony Sibley / Simon Mark, „ Speech recognition using DTW on a TI DSP“,
speechrecognition.pdf
- [3] Jaap Haitsma / Michiel van der Veen / Ton Kalker / Fons Bruekers, "Audio Watermarking for
Monitoring and Copy Protection", Haitsma.doc
- [4] Mario Jurcevic / Markus Gloor, "Echtzeit Wasserzeichensystem für Sprache", HSR, 2003
- [5] Texas Instruments, "Using a TMS320C6000 McBSP for Data Packing",McBSP_init.pdf
- [6] Texas Instruments, "TMS320C67x DSP Library Programmer's Reference Guide",
c67_lib_spru_657.pdf
- [7] Texas Instruments, "TMS320C6000 McBSP Initialization", McBSP.pdf
- [8] W. Bender / D. Gruhl / N. Morimoto / A. Lu, „Techniques for data hiding“, bender.pdf
- [9] Wolfgang Effelsberg / Ralf Steinmetz, "Digitale Wasserzeichen", wasserzeichen.pdf
- [10] „Algorithmische Zahlentheorie“, algopdf_2.pdf

Kapitel 12: Anhang

12.1 Abbildungsverzeichnis

Abbildung 4-1: Einfügen des Wasserzeichens.....	10
Abbildung 4-2: Detektion des Wasserzeichens.....	11
Abbildung 5-1: DES Algorithmus	14
Abbildung 5-2: Eingangspemutation	14
Abbildung 5-3: DES Blöcke	15
Abbildung 5-4: Expansionspermutation	15
Abbildung 5-5: ECB.....	17
Abbildung 5-6: CBC Modus.....	18
Abbildung 5-7: OFB Modus	18
Abbildung 5-8: CFB Modus.....	19
Abbildung 5-9: EDE-Schema.....	20
Abbildung 7-1: TMS320 DSP Board	26
Abbildung 7-2: Analog Digital Wandler TDMX326040A.....	27
Abbildung 7-3: Versuchsaufbau	28
Abbildung 7-4: Programmaufbau	29
Abbildung 7-5: Compilereinstellung Basic	30
Abbildung 7-6: Compilereinstellung Advanced	30
Abbildung 7-7: Flussdiagramme Sender, Empfänger.....	32
Abbildung 7-8: EDMA Transfer von Peripherie in internen Speicher.....	33
Abbildung 7-9: Unterschied zwischen Element, Frame und Block.....	34
Abbildung 7-10: Parameter RAM (PaRAM).....	35
Abbildung 7-11: Konfigurationstabelle EDMA	35
Abbildung 7-12: Linken von EDMA Transfers.....	36
Abbildung 7-13: EDMA Parameter Options (OPT Register)	37
Abbildung 7-14 Ping Pong Buffer	39
Abbildung 7-15: Ablauf Ping Pong	40
Abbildung 7-16: Fehler bei der Detektion	43
Abbildung 7-17: Verschiebung der auszuwertenden Frames mangels Synchronisation.....	44
Abbildung 8-1: Messresultat 1	49
Abbildung 8-2: Messresultat 2	50
Abbildung 8-3: Messresultat 3	50
Abbildung 8-4: Messresultat 4	51
Abbildung 8-5: Messresultat 5	51
Abbildung 8-6: Messresultat 6	52
Abbildung 8-7: Messresultat 7	52

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
3DES	Triple DES
DH	Diffie Hellmann
DES	Data Encryption Standart
NBS	National Bureau of Standarts
ANSI	American National Standarts Institute
ECB	Electronic Book Modus
CBC	Cipher Block Chaining Modus
OFB	Output Feedback
CFB	Cipher Feedback
NSA	National Security Agency
IRAM	Internal RAM
EMIF	External Memory Interface
EDMA	Enhanced Dircet Memory Access
CIPR	Channel Interrupt Pending Register
ER	Event Register