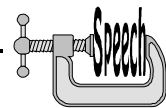


---

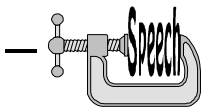
# Inhaltsverzeichnis

<b>1.</b>	<b>AUFGABENSTELLUNG</b>	<b>1</b>
<b>2.</b>	<b>EINLEITUNG</b>	<b>5</b>
2.1.	EINLEITUNG INS KAPITEL	5
2.2.	EINFÜHRUNG IN DIE DIPLOMARBEIT	5
2.3.	PFLICHTENHEFT	6
2.4.	ZEITPLAN	7
<b>3.</b>	<b>KONZEPT GUI</b>	<b>9</b>
3.1.	EINLEITUNG INS KAPITEL	9
3.2.	ANFORDERUNGEN AN DAS GUI	9
3.2.1.	KOMPRIMIERUNG	9
3.2.2.	DOWNLOAD	9
3.3.	DETAILKONZEPT GUI	11
3.3.1.	GUI'S MIT MATLAB	11
3.3.2.	DAS MATLAB TOOL GUIDE	11
3.3.3.	DER 1. PROTOTYP	13
3.3.4.	DER 2. PROTOTYP	13
3.3.5.	DAS SOFTWAREKONZEPT	16
3.3.6.	AUFNAHME EINES SPRACHSAMPLES	16
3.3.7.	LADEN EINES SPRACHSAMPLES	17
3.3.8.	KOMPRIMIERUNG DES SPRACHSAMPLES	17
3.3.9.	CODIERUNG DES KOMPRIMIERTEN SPRACHSAMPLES	17
3.3.10.	SPEICHERN UND LADEN VOM CODIERTEN SPRACHSAMPLE	17
3.3.11.	PROBEHÖREN	17
3.3.12.	DOWNLOAD DES SPRACHSAMPLES	17
3.3.13.	DATENFORMAT BEIM DOWNLOAD	18
<b>4.</b>	<b>KONZEPT HARDWARE</b>	<b>21</b>
4.1.	EINLEITUNG INS KAPITEL	21
4.2.	ANFORDERUNG AN DIE HARDWARE	21
4.2.1.	DOWNLOAD/SPEICHERORGANISATION	21
4.2.2.	AUSGABE MIT DEKOMPRIMIERUNG	21
4.2.3.	SPEZIFIKATIONEN	21
4.3.	DETAILKONZEPT HARDWARE	22
4.3.1.	AUFBAU UND EINSATZMÖGLICHKEIT VOM ENTWICKLUNGSBOARD SPACE	22
4.3.2.	DOWNLOAD/SPEICHERORGANISATION	23
4.3.3.	DEKOMPRIMIERUNG	23
4.3.4.	AUSGABE	25



---

<b>5.</b>	<b>IMPLEMENTATION GUI</b>	<b>31</b>
<b>5.1.</b>	<b>EINFÜHRUNG</b>	<b>31</b>
<b>5.2.</b>	<b> Globale Variablen</b>	<b>31</b>
<b>5.3.</b>	<b>Funktionsstruktur</b>	<b>31</b>
<b>5.4.</b>	<b>Funktionsbeschreibungen</b>	<b>33</b>
5.4.1.	DIE FUNKTION SQUEEZE()	33
5.4.2.	KOMPRIMIER-FUNKTIONEN	35
5.4.3.	GUI-FUNKTIONEN	37
5.4.4.	DIE FUNKTION MOTS2()	38
5.4.5.	DIVERSE MATLAB-FUNKTIONEN	39
<b>6.</b>	<b>IMPLEMENTATION HARDWARE</b>	<b>41</b>
<b>6.1.</b>	<b>Einleitung ins Kapitel</b>	<b>41</b>
<b>6.2.</b>	<b>HC-11 Software</b>	<b>41</b>
6.2.1.	EINFÜHRUNG	41
6.2.2.	ÜBERSICHT DER PROGRAMMKOMPONENTEN	41
6.2.3.	ERLÄUTERUNGEN ZU DEN KOMPONENTEN	42
6.2.4.	BESCHREIBUNG DER SOFTWARE	42
<b>6.3.</b>	<b>Implementation FPGA</b>	<b>49</b>
<b>6.4.</b>	<b>Implementation Print</b>	<b>50</b>
<b>7.</b>	<b>AUSWERTUNG DER RESULTATE</b>	<b>51</b>
<b>7.1.</b>	<b>Einleitung ins Kapitel</b>	<b>51</b>
<b>7.2.</b>	<b>Auslastung des Prozessors</b>	<b>51</b>
7.2.1.	MESSMETHODE	51
7.2.2.	MESSUNG DER RECHENZEIT UND PROZESSORAUFLASTUNG	53
7.2.3.	AUSWERTUNG DER MESSRESULTATE	53
<b>7.3.</b>	<b>Akustische Bewertung</b>	<b>54</b>
<b>8.</b>	<b>BEDIENUNGSANLEITUNG</b>	<b>55</b>
<b>8.1.</b>	<b>Installation</b>	<b>55</b>
<b>8.2.</b>	<b>BEDIENUNG DES MAGIC VOICE SQUEEZERS</b>	<b>56</b>
	<b>BEDIENUNG DER SPACE HARDWARE</b>	<b>57</b>
<b>9.</b>	<b>SCHLUSSFOLGERUNG</b>	<b>59</b>
<b>9.1.</b>	<b>RÜCKBLICK</b>	<b>59</b>
<b>9.2.</b>	<b>SCHLUSSWORT</b>	<b>59</b>



---

## ANHANG

---

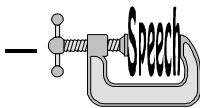
- ANHANG A: SITZUNGSPROTOKOLLE
- ANHANG B: LITERATURVERZEICHNIS
- ANHANG C: TABELLEN FÜR EXPANSION VON  $\mu$ -LAW KOMPR. SAMPLES
- ANHANG D: SOURCE-CODE M68HC11
- ANHANG E: GUI M-FILES
- ANHANG F: SCHEMAS VON SPACE
- ANHANG G: SCHEMA VON FPGA
- ANHANG H: SCHEMA VON PRINT
- ANHANG I: DATENBLATT LM4860 BOOMER
- ANHANG J: DATENBLATT CRYSTAL CS4430/31/33

## Abbildungsverzeichnis

Abbildung 1	Einsatz der Komprimierungsverfahren .....	5
Abbildung 2	Zeitplan .....	7
Abbildung 3	Grobkonzept des GUI's, schematisch dargestellt.....	10
Abbildung 4	Das Matlab-Tool GUIDE .....	12
Abbildung 5	WinAmp MP3 Player.....	13
Abbildung 6	Handskizze für Prototyp 1 .....	13
Abbildung 7	Prototyp 1 .....	13
Abbildung 8	Handskizze für Prototyp 2.....	14
Abbildung 9	Verwendung von Listbox.....	15
Abbildung 10	Prototyp 2, Fenster nach Aufstarten von Squeeze .....	15
Abbildung 11	Softwarekonzept, Struktogramm .....	16
Abbildung 12	Einstellung der Parameter .....	17
Abbildung 13	Aufbau des Registers.....	19
Abbildung 14	Textfile im Motorola S2-Format .....	19
Abbildung 15	Grobkonzept des Mikrokontrollersystems, schematisch dargestellt.....	21
Abbildung 16	Aufbau der Hardware .....	22
Abbildung 17	Memorymap von SPACE und HC11 .....	23
Abbildung 18	Umrechnungstabelle von unsigned auf signed Integer .....	24
Abbildung 19	Design-Flow eines Xilinx-Projekts .....	25
Abbildung 20	'Options' Fenster bei der Synthese .....	26
Abbildung 21	Struktur des FPGA-Designs .....	27
Abbildung 22	Funktionsstruktur .....	32
Abbildung 23	Programmstruktur von Squeeze .....	33
Abbildung 24	Mögliche Codes im Sprachdatenvektor .....	36
Abbildung 25	Beispiel eines Ablaufs von mots2 .....	39
Abbildung 26	Übersicht über alle Komponenten der HC11-Software .....	41
Abbildung 27	Ablauf von einem Download eines Sprachsamples .....	43
Abbildung 28	Ablauf von Lesen und Dekompirmieren eines Sprachsamples .....	45
Abbildung 29	8-Bit Datenformat für verschiedene Bittiefen .....	46
Abbildung 30	Lesen von Samples bei 4-Bit Quantisierung aus dem PEROM .....	47
Abbildung 31	Lesen von Samples bei 5-Bit Quantisierung aus dem PEROM .....	47
Abbildung 32	Lesen von Samples bei 6-Bit Quantisierung aus dem PEROM .....	48
Abbildung 33	Zeitsimulation .....	49
Abbildung 34	Ansteuerung des Ports A zur Messung der Auslastung des Prozessors ..	51
Abbildung 35	Messung der Auslastung des Prozessors am KO .....	52
Abbildung 36	The Magic Voice Squeezer.....	56
Abbildung 37	Die SPACE Hardware.....	57

## Tabellenverzeichnis

Tabelle 1	Expansionstabelle für 4-Bit Samples .....	24
Tabelle 2	Funktionen für die Sprachausgabe .....	46
Tabelle 3	Messung der Prozessorauslastung für Samples mit $\mu$ -Law-Komprimierung.....	53
Tabelle 4	Berechnung der Rechenzeit für einen Sprachsample .....	53



# 1. Aufgabenstellung



ABTEILUNG FÜR ELEKTROTECHNIK  
LABOR FÜR DIGITALE SIGNALVERARBEITUNG (DS)

CH-8640 RAPPERSWIL, OBERSEESTRASSE 10  
TELEFON: 055 / 222 41 11  
TELEFAX: 055 / 222 44 00

Nr. D98-06

Diplomarbeit für Herrn S. Kaufmann, R. Lienhardt

## **Sprachkomprimierungsverfahren für Microcontroller-Anwendungen**

### **Einleitung**

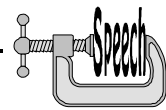
Werden Sprachsignale mit dem in der Telefonie gebräuchlichen PCM-Verfahren (Pulse Code Modulation) digitalisiert und aufgezeichnet, so fallen pro Sekunde Aufzeichnung 64kBit Daten an. Mit geeigneten Komprimierungsverfahren kann diese Datenrate weiter reduziert werden, um bei einer Aufzeichnung Speicherplatz oder bei einer Übertragung Signalbandbreite zu sparen. Mit ADPCM (Adaptiv Differential Pulse Code Modulation) kann man, im Vergleich zu PCM, eine Halbierung der Datenrate (32kBit/s) erreichen. ADPCM basiert auf Prädiktionsalgorithmen [1], welche für die Komprimierung und auch für die Dekomprimierung relativ viel Rechenleistung voraussetzen. Dieses Verfahren eignet sich deshalb kaum für den Einsatz in Geräten, wo auf leistungsfähige Signalprozessoren verzichtet werden muss. In der vorangegangenen Studienarbeit [2] wurden verschiedene andere Verfahren entwickelt, die sich mit einem Microcontroller – System realisieren lassen. Mit dieser Diplomarbeit sollen nun diese Verfahren auf einem Microcontroller - System als Echtzeit – Anwendung implementiert und ausgetestet werden.

### **Aufgabe**

Mindestens zwei Varianten der in der DS - Studienarbeit [2] entworfenen Komprimierungsverfahren sollen auf dem Microcontroller – Board "SPACE" [3] , mit einem Microcontroller aus der Familie 68HC11 von Motorola, als Echtzeit – Anwendungen realisiert werden. Dieses Prototyp – System soll folgende Teilfunktionen umfassen:

- Aufzeichnen und Komprimieren von kurzen Sprachsequenzen auf dem PC.
- Sprachsequenzen auf das SPACE – Board übertragen.
- Sprachsequenzen auf dem SPACE – Board speichern und verwalten.
- Sprachsequenzen in Echtzeit dekomprimieren und abspielen.

Falls spezielle Hardware ( z.B. D/A-Wandler, Filter usw.) verwendet wird soll diese als aufsteckbares SPACE – Modul realisiert werden.



### **Ziel der Arbeit**

Mit dieser Diplomarbeit soll ein funktionierendes Prototyp - System für die Aufzeichnung und Wiedergabe von komprimierten Sprachsignalen mit PC bzw. Microcontroller realisiert werden. Die erreichten Ergebnisse sollen mit der Theorie und mit den anlässlich der Studienarbeit [2] ausgeführten Simulationen verglichen werden. Es ist wichtig, dass die gesamte Arbeit umfassend dokumentiert wird und dass sämtliche Resultate (positive wie auch negative) diskutiert werden.

### **Vorgehen**

Kennenlernen der Microcontrollers MC68HC11

Pflichtenheft erstellen

Konzept ausarbeiten

evtl. Hardwareentwicklung

Softwareentwicklung

Durchführung von eingehenden Tests und Messungen zur Überprüfung der korrekten Funktionsweise und der Charakterisierung der realisierten Prototypen.

In Anbetracht der geringen Zeit, die zur Verfügung steht, wird dringend empfohlen, frühzeitig eine sorgfältige Arbeits- und Zeitplanung durchzuführen und diese mit dem Betreuer zu besprechen!

### **Literatur**

- [1] Lawrence R. Rabiner, Ronald W. Schafer, Digital Processing of Speech Signals, Prentice Hall 1978, ISBN 0-13-213603-1
- [2] Steven Kaufmann, Reto Lienhardt, Sprachkomprimierungsverfahren für Microcontroller – Anwendungen, HSR Studienarbeit SS98-6
- [3] Ascom Systec AG, Dokumentation SPACE  
Ascom Systec AG 1997
- [4] Ludwig Orgler, MC68HC11 Mikrocontroller, Franzis-Verlag 1996,  
ISBN 3-7723-5724-5

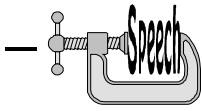
### **Bericht**

Über die Arbeit ist ein Bericht zu verfassen. Dabei sind die Richtlinien der Abteilung für Elektrotechnik für das Erstellen von Berichten einzuhalten. Alle verwendeten Quellen sind im Literaturverzeichnis anzugeben (Hinweise im Text). Der Bericht ist im Doppel abzugeben; ein Exemplar verbleibt am Labor für Digitale Signalverarbeitung des HSR. Die erstellten Programme und der Text des Berichts sind auf einer beschrifteten Diskette (mit Nummer der Diplomarbeit) beizulegen.

### **Termine**

Ausgabe der Aufgabenstellung:  
19. Oktober 1998

Abgabe des Berichts inkl. Zusammenfassung  
4. Dezember 1998, 17:00 Uhr



**Kontaktadresse**

Andreas Ehrensperger, El.-Ing. HTL  
Ascom Systec AG  
Abteilung CMEU (DSP)  
H43315  
8634 Hombrechtikon  
Tel.: 055 254 66 07  
Fax.: 055 254 67 62  
E-Mail: [Andreas.Ehrensperger@ascom.ch](mailto:Andreas.Ehrensperger@ascom.ch)

Grüt, 17. Oktober 1998

A.Ehrensperger



## 2. Einleitung

### 2.1. Einleitung ins Kapitel

Das Kapitel Einleitung verleiht einen Überblick über die Diplomarbeit Sprachkomprimierung. Ausserdem sind darin organisatorische Aspekte wie Anforderungsspezifikation, Projektstruktur und Zeitplan enthalten.

### 2.2. Einführung in die Diplomarbeit

Diese Arbeit ist eine Weiterführung der Semesterarbeit SS-98-06 [2]. Es wird daher nochmals kurz in die Problematik der Sprachkomprimierung eingegangen.

In einem mobilen Notrufsystem soll die Menüführung durch die Ausgabe von Sprachsequenzen gesteuert werden. Die Sprachdaten sind in digitalisierter Form auf einem Speicherchip gespeichert. Während der Benutzerführung werden die Sprachsequenzen von diesem Speicherchip geladen und durch einen Lautsprecher sofort wiedergegeben.

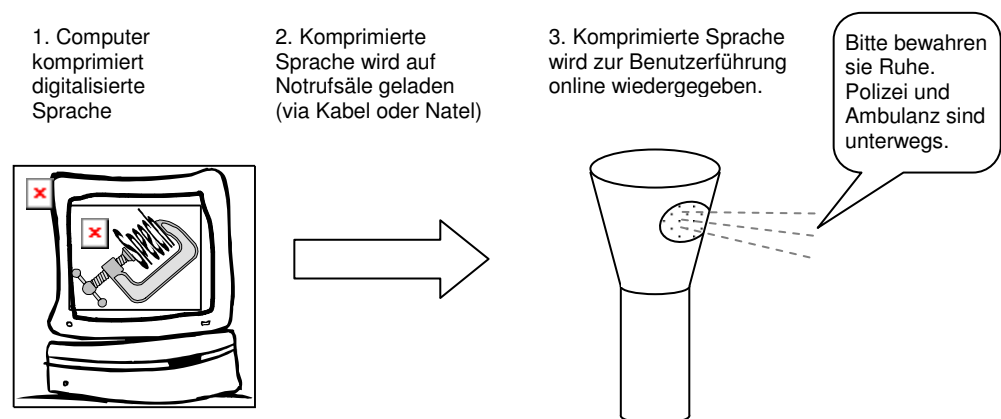
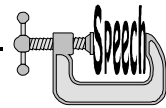


ABBILDUNG 1 EINSATZ DER KOMPRIMIERUNGSVERFAHREN

Durch den begrenzten Speicherplatz, ist es nicht möglich, die Sprachdaten in Telefonqualität zu speichern. Darum bearbeitet man die Sprachsequenzen nach der Aufnahme mit dem Computer. Es ist möglich, durch geschickte Beeinflussung, die benötigte Datenmenge der digitalisierten Sprache massiv zu verringern. Dabei nimmt man einen geringen Qualitätsverlust in Kauf. Die Sprache muss aber noch gut verständlich sein und darf nicht zu fest rauschen.

Da ein mobiles System möglichst lange ohne Energieversorgung funktionieren soll, ist man auf einen niedrigen Energieverbrauch aller Komponenten angewiesen. Der Einsatz von schnellen Digitalen Signalprozessoren (DSP) kommt daher nicht in Frage, denn DSP haben einen Stromverbrauch, der denjenigen von einfachen Mikroprozessoren um ein vielfaches übersteigt. Es ist also zu beachten, dass man bei der Echtzeitwiedergabe der Sprachsequenzen eine sehr beschränkte Rechenleistung zu Verfügung hat.

Im Rahmen der Semesterarbeit SS-98-06 [2] wurden verschiedene Verfahren zur Komprimierung von Sprachsignalen evaluiert und realisiert. Die Verfahren zeichnen sich durch eine geringe benötigte Rechenleistung auf der Empfängerseite aus. Gegenüber Telefonqualität, welche bei einer Samplingrate von 8 kHz und einer Wortlänge von 8 Bit



64 kBit/s Daten produziert, wird ein Kompressionsfaktor von mindestens 2 erreicht. Die Datenrate wird also auf 32 kBit/s reduziert.

In der Diplomarbeit wird die Dekompression und Ausgabe mit verschiedenen Verfahren mit einem Mikroprozessor realisiert. Das Ziel ist, Sprachsequenzen aufzunehmen und zu komprimieren, die dann vom Mikroprozessor autonom und in Echtzeit wiedergegeben werden können.

### **2.3. Pflichtenheft**

- Sprachsequenzen werden auf dem PC aufgezeichnet und mittels der in der Studienarbeit [2] entwickelten Verfahren komprimiert.
- Für die Programmausführung wird Matlab mit der Signal Processing Toolbox verwendet. Der Anwender wird durch eine graphische Oberfläche durch die einzelnen Programmschritte geführt.
- Die komprimierten Daten werden vom PC über die serielle Schnittstelle in den Speicher des Mikrokontroller-Boards geladen.
- Die Sprachsequenzen werden auf dem Mikrokontroller-Board mit einem Dateisystem verwaltet. Es können einzelne oder mehrere Sprachsequenzen gespeichert und verwaltet werden.
- Es ist möglich, Sprachsequenzen zu speichern, die mit verschiedenen Verfahren komprimiert wurden und verschiedene Samplingraten zwischen 6kHz und 8kHz aufweisen.
- Die Sprachsequenzen werden bei Bedarf sofort in Echtzeit mit dem passenden Verfahren dekomprimiert und der richtigen Samplingfrequenz auf einem Lautsprecher abgespielt.

## 2.4. Zeitplan

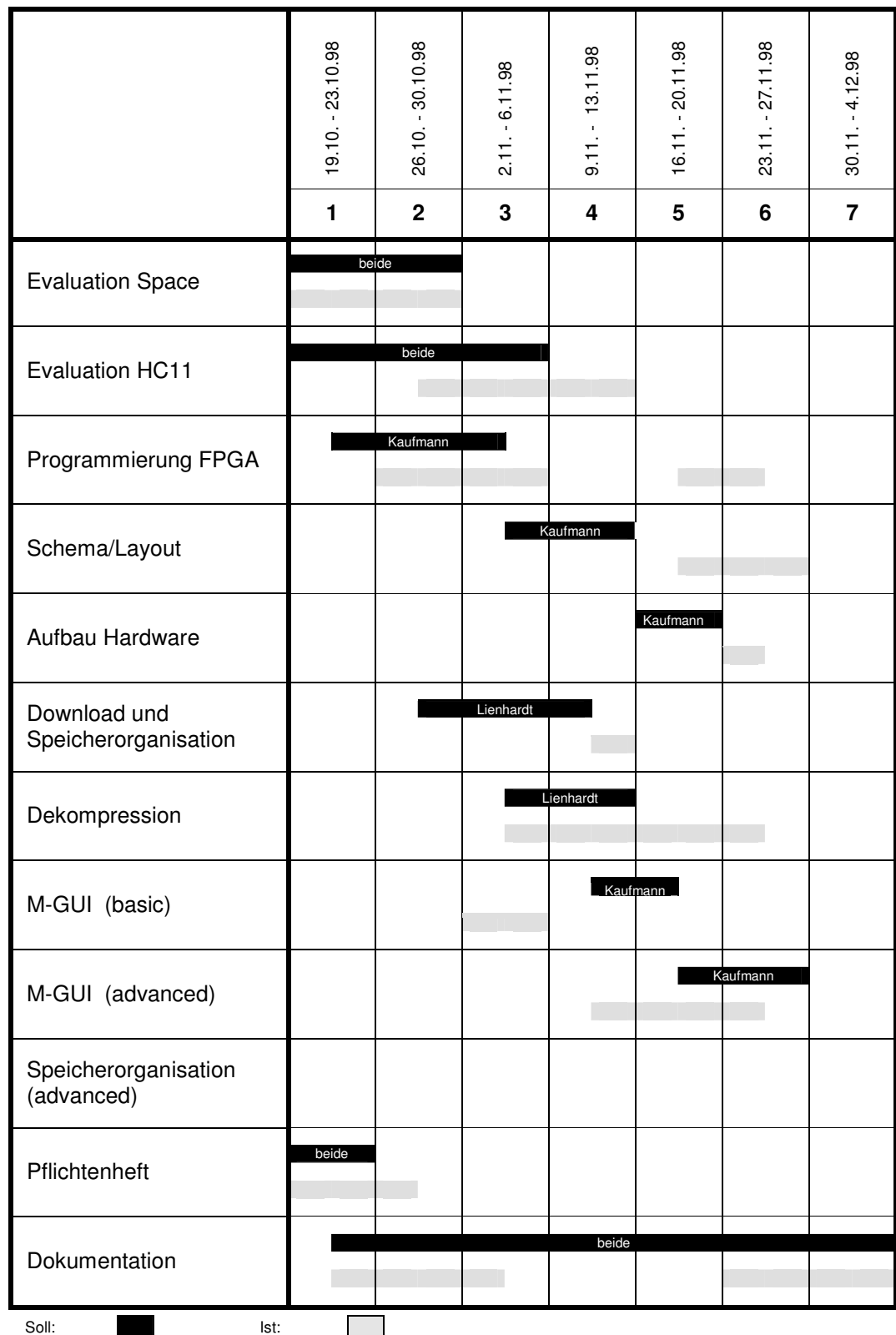
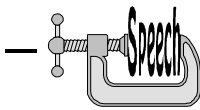


ABBILDUNG 2 ZEITPLAN





## **3. Konzept GUI**

---

### **3.1. Einleitung ins Kapitel**

Die Grundlage für die Konzeption der graphischen Benutzeroberfläche (GUI) auf dem PC liefern die Aufgabenstellung und das Pflichtenheft. In diesem Kapitel wird die Umsetzung der Aufgabenstellung und des Pflichtenhefts in ein detailliertes beschrieben. Zuerst wird ein Grobkonzept erstellt. Nachher werden die Bestandteile des Grobkonzepts genauer spezifiziert. Überlegungen und Entscheidungen, die in der Konzeptionsphase gemacht wurden, sind hier erläutert.

### **3.2. Anforderungen an das GUI**

#### **3.2.1. Komprimierung**

- Start der Graphischen Benutzerführung (GUI).
- Aufnahme eines Sprachsamples.
- Laden eines Sprachsamples.
- Komprimierung und Codierung des Sprachsamples mit verschiedenen Bitbreiten, Samplingraten und Komprimiermethoden.
- Probehören des originalen Sprachsamples.
- Speichern und Laden von codierten Sprachsamples.
- Probehören des codierten Sprachsamples.

#### **3.2.2. Download**

- Transfer des codierten Sprachsamples zum Mikrokontrollersystem.

Diese Komponenten sind in der folgenden Graphik schematisch zusammengefasst:

GUI (Graphical User Interface) auf dem PC

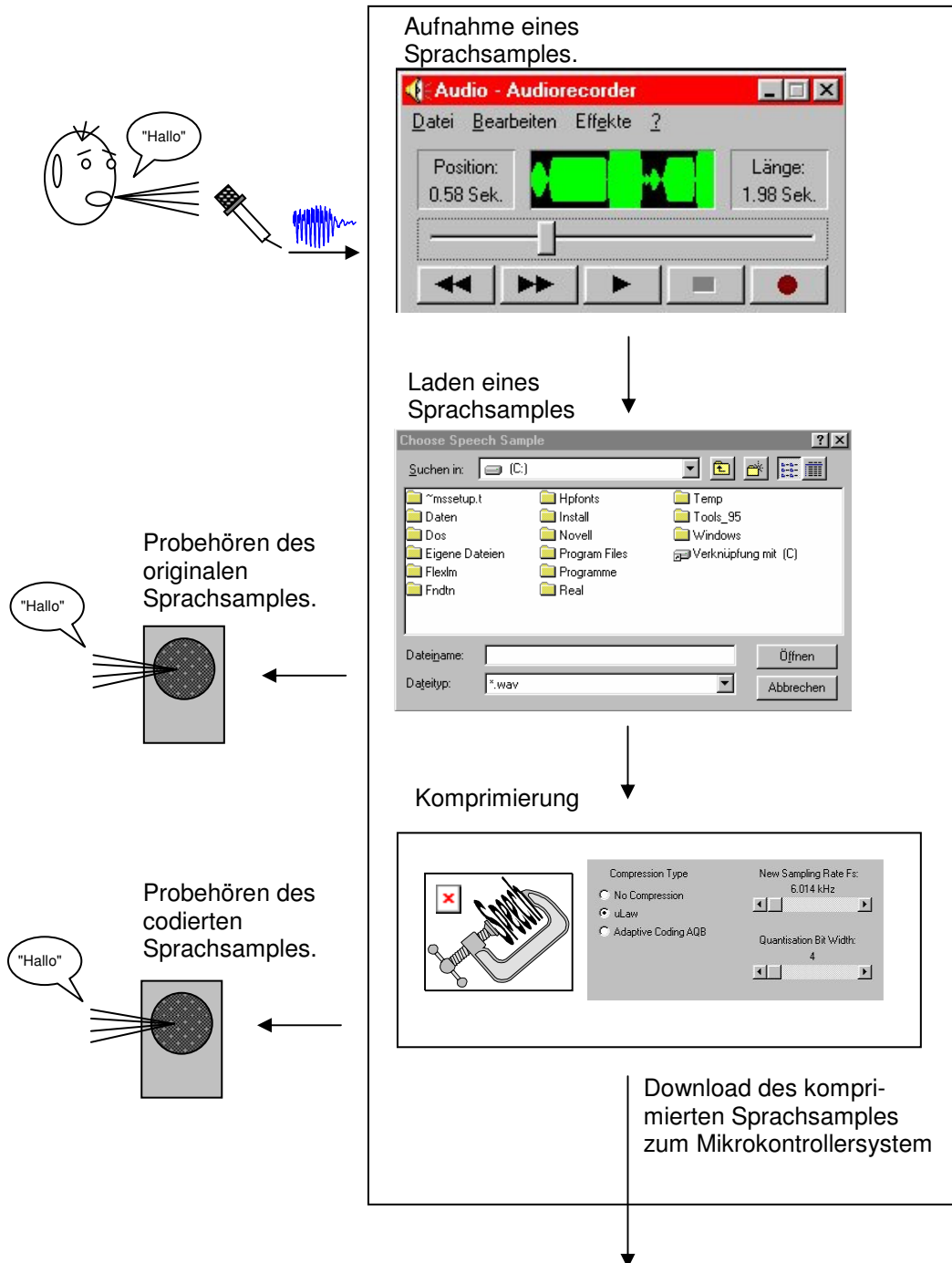
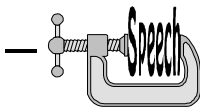


ABBILDUNG 3 GROBKONZEPT DES GUI'S, SCHEMATISCH DARGESTELLT



### 3.3. Detailkonzept GUI

Um die in der Semesterarbeit [2] entwickelten Komprimierungsverfahren einfach verwendbar zu machen, ist es sinnvoll, die Funktionen mit einer graphischen Benutzerführung zu steuern. Da die Funktionen zur Sprachkomprimierung in der Form von MATLAB M-Files zur Verfügung stehen, liegt es nahe, auch das GUI mit MATLAB Version 5.2 zu erstellen.

Bei einer Graphischen Benutzerführung ist es wichtig, dass die Oberfläche übersichtlich gestaltet ist. Darum wurde bei diesem GUI gänzlich auf ein Menu verzichtet. So sind alle Bedienelemente sofort ersichtlich, ohne sich zuerst durch zehn Menus durchklicken zu müssen. Die Bedienung soll möglichst ohne Bedienungsanleitung möglich sein. Eine solche intuitive Bedienung ist durch eine geschickte Wahl und Anordnung der Bedienelemente erreichbar. Mit Frames kann man Bedienelemente optisch gruppieren. In diesem GUI wurde darauf geachtet, dass der Ablauf eines Komprimiervorgangs im GUI-Design durch eine logische Anordnung der Bedienelemente berücksichtigt wurde.

#### 3.3.1. GUI's mit MATLAB

Eine MATLAB-Benutzeroberfläche besteht aus verschiedenen MATLAB-Objekten. Das Hauptfenster ist ein 'Figure'-Objekt. Graphische Darstellungen sind 'Axes'-Objekte und die Optionen der Menuleiste sind 'Uimenu'-Objekte. Alle übrigen Objekte, die im GUI eingesetzt werden, sind vom Typ 'Uicontrol'. Jedes Objekt besitzt ein Handle (Handling, Verwaltung). In [9] werden drei Arten der GUI-Programmierung unterschieden: Erstens können die Handles lokal bestehen, zweitens global, oder drittens können sie als Eigenschaften der Objekte (Userproperty) gespeichert werden. Wir haben uns hier für die Variante mit globalen Handles entschieden, weil sie eine übersichtliche Programmgestaltung möglich macht. Es muss aber darauf geachtet werden, dass nach der Ausführung von 'clear all' in der Matlab Konsole diese Handles gelöscht werden, und somit das Programm nicht mehr funktioniert!

Jedes Figure- oder Axes-Objekt hat eine ButtonDown-Funktion (ButtonDownFcn), welche eine Aktion initialisiert, wenn mit der Maus auf das betreffende Objekt geklickt wird. Bei den Uicontrol-Objekten heisst diese Funktion Callback anstatt ButtonDownFcn. Eine Aktion wird nur dann ausgelöst, wenn mit der Maus auf ein Objekt geklickt wird. D.h. ein Matlab GUI ist nicht in einer Endlosschleife (Polling Mode), wie es zum Teil andere GUI's sind.

Informationen zur Erstellung vom GUI findet man in der Matlab-Online-Hilfe. Dort gibt es auch eine "Solution Search Engine", die in Textform gestellte Fragen (natürlich in englisch) verblüffend gut beantwortet. Ein weiteres wichtiges Hilfsmittel bei unseren Lösungsansätzen war das Buch von Patrick Merchant [9 Building GUI's with MATLAB], das im DS-Labor vorhanden ist.

Das GUI kann mit dem MATLAB-Tool GUIDE, oder manuell erstellt werden.

#### 3.3.2. Das Matlab Tool GUIDE

In den Manuals von MATLAB 5.2 [10, Building GUI's] wird dieses graphische Tool von Matlab anhand eines Beispiels beschrieben. Es wird gestartet, indem man in der Matlab Konsole den Befehl 'guide' eingibt.

Mit Guide kann man mit Hilfe von Drag&Drop ein einfaches GUI erstellen.

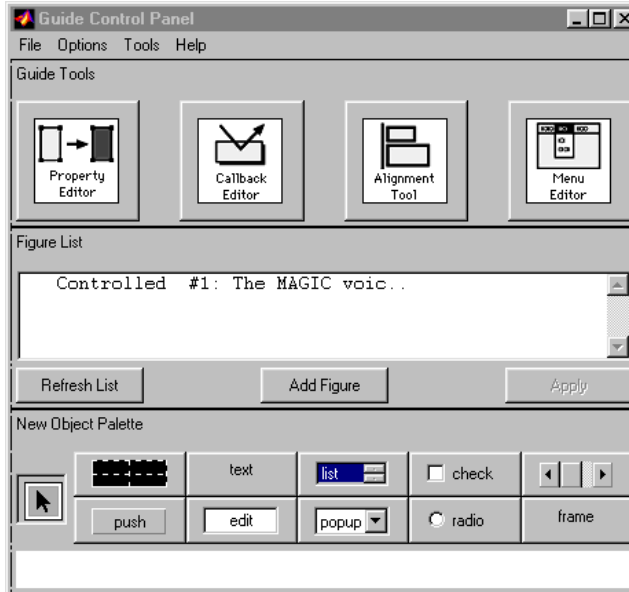


ABBILDUNG 4 DAS MATLAB-TOOL GUIDE

Kernstücke des Tools sind die Guide-Tools, die Figure List und die New Objects-Palette.

**Guide Tools:**

- Property Editor: Mit diesem Tool können die Eigenschaften der Objekte eingestellt werden. (Farbe, Position, ...)
- Callback Editor: Die Callbackfunktion des Objekts kann definiert und geändert werden.
- Alignment Tool: Die angewählten Objekte könne aufeinander ausgerichtet werden.
- Menu Editor: Die Menus der Figuren könne erstellt und editiert werden.

**Figure List:**

In dieser Figur werden alle Matlab-Fenster angezeigt, die offen sind. Es gibt zwei Zustände:

- "controlled" heisst, die Figur kann mit dem Guide-Tool verändert werden.
- "active" heisst, die Figur ist im aktiven Modus und wartet auf Eingaben des Benutzers.

**New Objects Palette:**

In diesem Fenster kann der GUI-Programmierer mit der Maus ein Uicontrol-Objekt packen und in der aktuellen Figur plazieren.

Mit Guide kann man einfach und schnell ein GUI erstellen. Für unsere Anwendung eignet es sich aber schlecht, da bei der Speicherung eines mit Guide erstellten GUI's ein M-File erstellt wird, das standardisierte Variablen enthält. Bei einem komplexen GUI ist dieses M-File sehr schlecht lesbar und darum auch schlecht erweiterbar.

### 3.3.3. Der 1. Prototyp

Beim ersten Prototyp ist es wichtig, dass ein lauffähiges, brauchbares Programm zur graphisch unterstützten Komprimierung besteht. Die Parameter der Komprimierung (Bitbreite, Abtastfrequenz, Komprimiermethode) können noch nicht eingestellt werden. Die Planung erfolgt durch eine Handskizze. Als Ideen-Vorlage für das Design dient der Winamp MP3 Player. Die Idee dahinter ist, eine Oberfläche ähnlich der eines CD-Players zu gestalten, damit die Bedienung dem Benutzer von Anfang an vertraut vorkommt.



ABBILDUNG 5 WINAMP MP3 PLAYER

Die Design Idee hält man am Besten mit einer Handskizze fest. Diese Vorgehensweise wird auch in [10, Building GUIs with MATLAB] beschrieben.

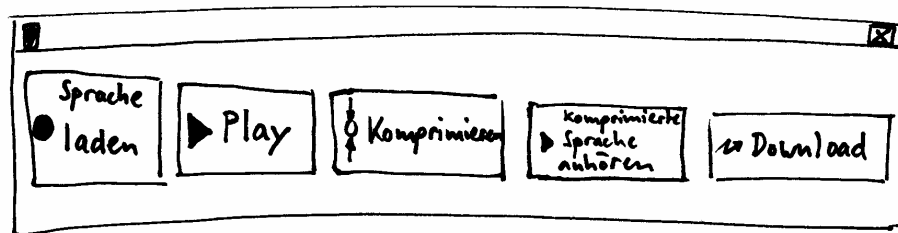


ABBILDUNG 6 HANDSKIZZE FÜR PROTOTYP 1

Da die Erzeugung der Symbole auf den Buttons nicht ganz einfach ist, wird in diesem Prototyp darauf verzichtet.



ABBILDUNG 7 PROTOTYP 1

### 3.3.4. Der 2. Prototyp

Die Erweiterung des ersten Prototyps ist nicht so einfach, wie erwartet. Das vom MATLAB Tool 'Guide' erstellte M-File lässt sich schlecht erweitern, und das graphische Konzept des 1. Prototyps lässt wenig Ideen für Erweiterungen offen. Darum wird der 2. Prototyp komplett neu gestaltet. Wieder wird das Menu ausgeblendet, um eine optimale Übersicht zu gewährleisten. Als Neuerung musste für diesen Prototyp ein geeigneter Name gefunden werden. Da der Begriff Komprimierung schon lange als schwerfällig empfunden wurde, hat man dafür einen neuen Begriff gesucht. Dabei kam das englische SQUEEZE (auspressen, drücken, quälen) hervor. Das GUI hat dann den Namen "The

Magic Voice Squeezer" (engl. Die magische Stimmen-Pressen) bekommen, weil die Komprimierung eine annähernd "magische" Komprimierleistung zeigt. Der Prototyp 2 hat folgende Handskizze als Grundlage:

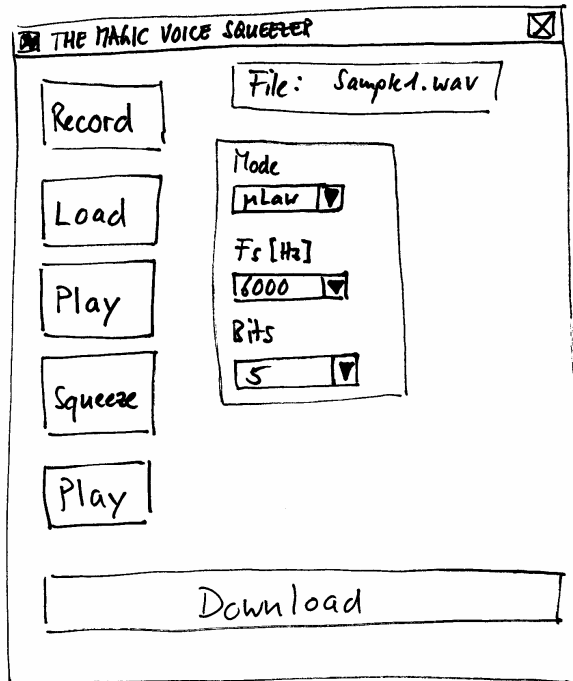


ABBILDUNG 8 HANDSKIZZE FÜR PROTOTYP 2

Dieses Design wird dann Schritt für Schritt erweitert, bis dann die endgültige Oberfläche resultiert.

Der Nachteil dieses Prototyps ist die schlechte Transparenz der ausgeführten Aktionen. Es ist nicht möglich, die Aktionen dem Benutzer anzuzeigen. Darum wird der Prototyp um eine Konsole erweitert. Die Konsole zeigt dem Benutzer für ihn wichtige Informationen an und macht das GUI somit sehr benutzerfreundlich, weil dieser immer weiss, was im Moment passiert. Zusätzlich kann der Benutzer seine Aktionen nachvollziehen, indem er in der Konsole die Meldungen anschaut. Dafür ist es nötig, dass man mit einem Scrollbar blättern kann.

Schwierig war es, einen geeigneten Uicontrol Typ für die Realisierung eines Konsolenfensters zu finden.

Eine Lösung ist in [9] beschrieben. Die dort gebrauchte Methode benutzt ein Uicontrol Objekt vom Typ 'Edit' und fügt den Scrolbar selber zu. Die beschriebene Methode ist aber für MATLAB 4 geschrieben, und kann ab MATLAB 5 nur mit aufwendigen Erweiterungen verwendet werden.

Die andere Lösung ist die Verwendung eines Uicontrols vom Typ 'Listbox'. Die Listbox ist, wie der Name sagt, gedacht, um ein Auswahlfenster zu erzeugen. Im folgenden Screenshot aus den Matlab-Demos (Befehl demo im Matlab-Fenster) sieht man, dass die Listbox auch von Matlab als Textfenster mit Scrollfunktion gebraucht wird (oben).

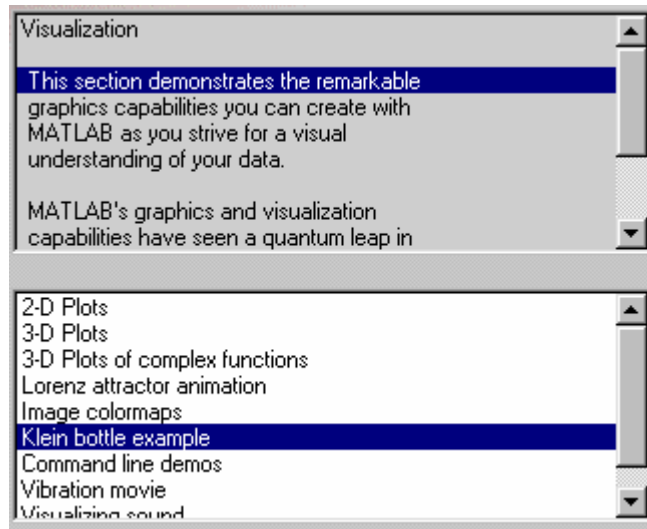


ABBILDUNG 9 VERWENDUNG VON LISTBOX

Unten ist die Listbox im herkömmlichen Sinn als Auswahlfenster gebraucht. Damit ist eine gute Lösung für die Erzeugung einer Mini-Konsole gefunden. Da sich der blaue Auswahlbalken nicht deaktivieren lässt, setzt man diesen immer ans untere Ende des Fensters.

Die Bedienelemente der Handskizze wurden im Laufe der Entwicklung durch funktionellere Typen angepasst: Für den Kompressionstyp wurden Radio-Buttons verwendet und für die Samplingrate und Quantisierungsbitbreite wurden Sliders verwendet.

So entsteht ein GUI, dass von einem Grossteil der Uicontrol Typen gebrauch macht.

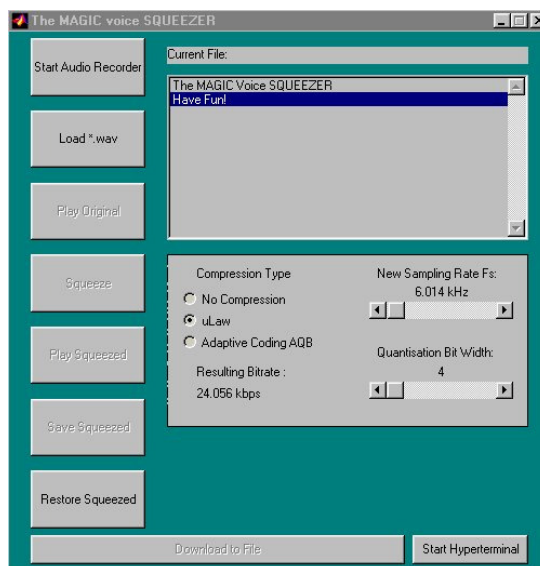
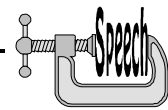


ABBILDUNG 10 PROTOTYP 2, FENSTER NACH AUFSTARTEN VON SQUEEZE

Die Funktionen der Bedienelemente sind in der Bedienungsanleitung dokumentiert. (Siehe Kapitel 8.2).



### 3.3.5. Das Softwarekonzept

Das GUI besteht aus **einem** M-File, in dem alle Figure-Handles und Programmvariablen global sind. Alle Callbackfunktionen, ausgelöst durch einen Mausklick, werden durch einen Aufruf des Hauptprogramms mit dem Callback-Kommando als Parameter ausgeführt. Defaultparameter ist 'Initialize', d.h. wenn kein Parameter übergeben wird, wird das GUI neu initialisiert.

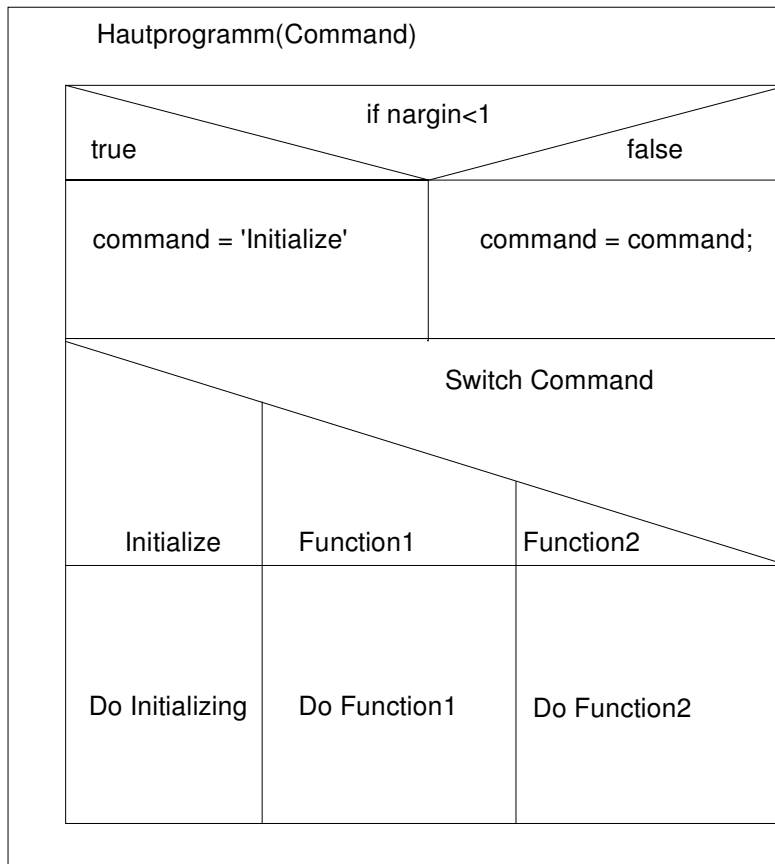
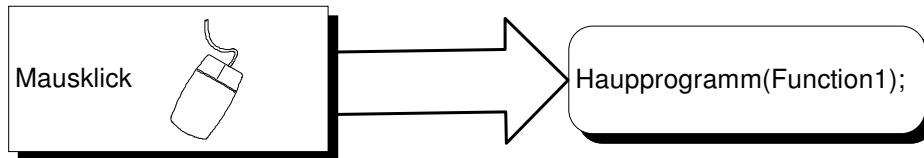


ABBILDUNG 11 SOFTWAREKONZEPT, STRUKTOGRAMM

### 3.3.6. Aufnahme eines Sprachsamples

Da Matlab keine Funktion bietet, um Sprachsamples mit einem Mikrofon direkt in MATLAB aufzunehmen, bietet sich der Microsoft Audiorecorder an. Dieser ist im Windows 95 Paket enthalten. Der Audiorecorder lässt sich zwar aus Matlab direkt mit dem dos() Befehl starten, der aufgenommene Sprachsample muss aber auf der Festplatte als Wave-File (\*.wav) gespeichert werden, und von dort in Matlab mit wavread() eingelesen werden.

### 3.3.7. Laden eines Sprachsamples

Mit der Matlab Funktion `Uigetfile()` wird ein Windows Benutzerdialog angezeigt. In diesem Fenster kann der Benutzer das gewünschte File mit der Endung `*.wav` auswählen. Mit dem Matlab-Befehl `wavread()` wird die Datei dann in den Matlab-Workspace geladen.

### 3.3.8. Komprimierung des Sprachsamples

Die Komprimierung erfolgt mit Hilfe der in [2] entwickelten Verfahren. Vor der Komprimierung kann man verschiedene Parameter einstellen.

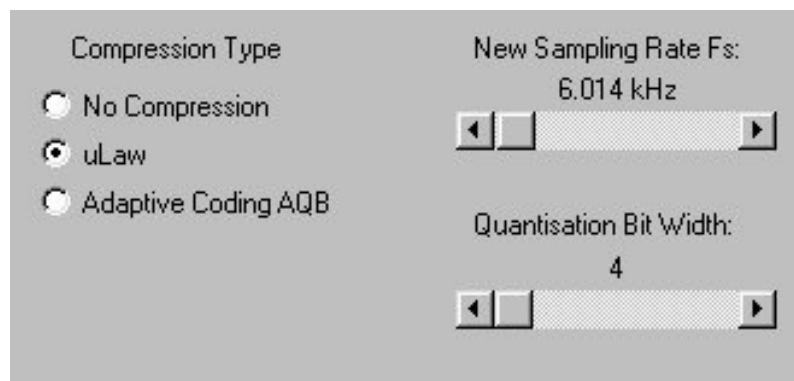


ABBILDUNG 12 EINSTELLUNG DER PARAMETER

Mit der ersten Softwareversion (Prototyp1) ist es möglich, ein Sample mit 6kHz, 4 Bit und  $\mu$ Law zu komprimieren. Die anderen Funktionen werden später implementiert.

### 3.3.9. Codierung des komprimierten Sprachsamples

Die Codierung setzt den komprimierten Sprachsample in das richtige Format, so dass die Daten im Mikrokontroller problemlos verarbeitet werden können. Das heisst die einzelnen Samplewerte werden in unsigned Integer zwischen 0 und  $2^{\text{Bittiefe}} - 1$  umgerechnet. Dazu wird noch eine Codierung eingefügt, die Sprachpausen und Sprachende anzeigt. (Details siehe Kapitel 5.4.2.2)

### 3.3.10. Speichern und laden vom codierten Sprachsample

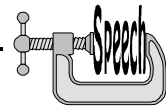
Die Variablen des Matlab-Workspace werden mit dem Befehl `save()` in einem File mit Endung `*.mat` gespeichert, und mit `load()` wieder hergestellt. Der Dateiname wird jeweils mit `Uigetfile()` abgefragt.

### 3.3.11. Probehören

Den **originalen** und den **codierten** Sprachsample kann der Benutzer per Tastendruck anhören. Dabei wird die Funktion `soundsc()` verwendet.

### 3.3.12. Download des Sprachsamples

Der Transfer der Sprachsamples zum Mikrokontrollersystem erfolgt über die serielle Schnittstelle vom PCs. Das direkte Schreiben und Lesen der Daten von der Schnittstelle ist mit Matlab Standard Routinen leider nicht möglich. Damit die Daten trotzdem mit Matlab transferiert werden können, wurde nach weiteren Hilfsprogrammen und Lösungsansätzen gesucht. Schlussendlich untersuchte man drei verschiedene Interface Programme.



#### 3.3.12.1. PortCom

Auf der MathWorks Inc. Web-Seite (Matlab Help Desk) besteht ein [Solution Search](#) Link. Mit Hilfe dieses Links konnte das Programm PortCom gefunden werden, welches die direkte Kommunikation über COM und LPT Ports mit allen Optionen möglich macht. Geschrieben wurde dieses Interface Programm von DSP Technology Inc. Nach eingehenden Tests mit diesen Interface Routinen stellte man fest, dass Daten auf die RS232 geschrieben werden konnten, aber beim Lesen von der Schnittstelle Speicherbereichverletzungen auftreten, und somit verschiedenste Windows Anwendung abstürzen.

Diese Problem wurde gleichzeitig noch von einem weiteren Diplomteam erkannt, und gemeinsam kam man zum Entschluss, dass sich dieses Programm nur zum Schreiben auf die Schnittstelle eignet, und nicht zum Lesen.

#### 3.3.12.2. WinWedge

Software Wedge ermöglicht anderen Windows-Programmen über serielle Schnittstellen (COM-Schnittstellen) zu kommunizieren. Es liest RS-232-Daten ein, konvertiert diese in das für die Applikation nötige Format und transferiert die Daten in die Applikation, entweder durch Simulation von Tastenanschlägen (Keystroke) oder durch den sog. Dynamic Data Exchange (DDE). Der Einsatz von WinWedge wäre sinnvoll, wenn die Applikation von Matlab automatisch gestartet werden könnte und dann unbemerkt im Hintergrund ablaufen würde. Aber leider ist auch nach längerem Betrachten der Manuals nicht herauszufinden, wie man WinWedge automatisch starten kann. Statt dessen muss man beim Start von Winwedge mit der Maus den Menüpunkt "activate" drücken.

#### 3.3.12.3. HyperTerminal

Mit dem HyperTerminal von Hilgraeve Inc., welches mit dem Windows 95 Paket mitgeliefert wird, können Daten über die Ports gesendet und empfangen werden. Hier müssen Sendedaten zuerst in ein Textfile abgespeichert werden, bevor sie mit dem Hyperterminal manuell über com1 heruntergeladen werden.

Trotzdem bleibt das Hyperterminal für diese Anwendung die beste Variante, da einerseits der programmiertechnische Aufwand für den Download der Daten sehr gering ist (Start des Hyperterminals aus Matlab), andererseits können Rückmeldungen des Mikrokontrollers über com0 direkt beobachtet werden. Der Nachteil liegt darin, dass man nebst den Matlab-GUI nochmals ein Fenster offen hat, und der Datendownload vom Benutzer manuell über "Übertragung/Textdatei senden..." ausgeführt werden muss.

### 3.3.13. Datenformat beim Download

Die Sprachdaten werden zuerst in eine Folge von 4 Bit Werten verwandelt, damit sie als Hex-Zahlen dargestellt werden können. Den Sprachdaten wird vor dem Download ein Servicetext und eine Register angefügt. Die Daten des Sprachsamples werden für den Download in ein Textfile geschrieben. Auf dem Mikrokontroller existiert ein Loaderprogramm, das die Daten im Motorola S2 Format lesen und in den Speicher schreiben kann. Folglich muss für die Verwendung dieses Loaders das Textfile dem S2 Format entsprechen.

#### 3.3.13.1. Servicetext

Der Benutzer kann vor dem Download des Sprachsamples den Sprachdaten eine Information in Textform anfügen. Diese Funktion ist für Servicezwecke vorgesehen. So kann die Person, die den Sprachsample gesprochen und komprimiert hat, ihren Namen und das Datum des Downloads anfügen. Der Mikrokontroller stellt eine Funktion zur Verfügung, um diesen Text anzuzeigen. Der Dateiname des S2-Datei wird dem Servicetext automatisch angefügt.





## 4. Konzept Hardware

### 4.1. Einleitung ins Kapitel

In diesem Kapitel werden die Anforderungen an das System erläutert, und die verschiedenen Möglichkeiten zur Realisation aufgezeigt. Für das HC11-Softwarekonzept werden einige Hinweise und Lösungen erwähnt. Für das Hardwarekonzept werden mögliche Lösungen diskutiert und eine davon ausgewählt

### 4.2. Anforderung an die Hardware

Das Grobkonzept erklärt die grobe Struktur der Hardware und die Anforderungen, die an das System gestellt werden.

#### 4.2.1. Download/Speicherorganisation

- Der Mikrokontroller verwaltet die Sprachsamples (Downloaden, Löschen, Verschieben und Umbenennen).

#### 4.2.2. Ausgabe mit Dekomprimierung

Die Sprachausgabe muss online erfolgen. Das heisst, die Anforderung an das System ist, dass der Mikrokontroller die Samplewerte fortlaufend berechnen kann, und dass die Ausgabe an einen DA-Wandler mit der richtigen Abtastfrequenz erfolgt.

- Der Mikrokontroller dekomprimiert die Daten und regelt die Ausgabe.
- Durch Tastendruck auf dem Mikrokontrollersystem kann ein Sprachsample über einen Lautsprecher abgespielt werden.

Diese Komponenten sind in der folgenden Graphik schematisch zusammengefasst:

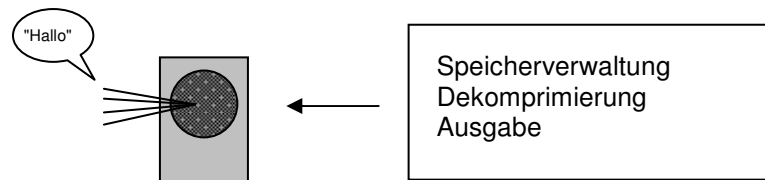


ABBILDUNG 15 GROBKONZEPT DES MIKROKONTROLLERSYSTEMS, SCHEMATISCH DARGESTELLT

#### 4.2.3. Spezifikationen

Das System soll Sprachsamples dekomprimieren und ausgeben können, die wie folgt komprimiert sind.

- Samplingfrequenzen zwischen 6kHz und 8kHz
- Bittiefen von 4,5 und 6 Bit
- mit oder ohne  $\mu$ -Law

Es soll automatisch erkannt werden, welche Spezifikationen der auszugebende Sprachsample hat, so dass er im richtigen Format ausgegeben wird.

### 4.3. Detailkonzept Hardware

Das Konzept für die Realisierung wird in drei Teile unterteilt:

- **Download/Speicherorganisation**
- **Dekomprimierung**
- **Ausgabe (Hardware, DA-Wandler und Lautsprecher)**

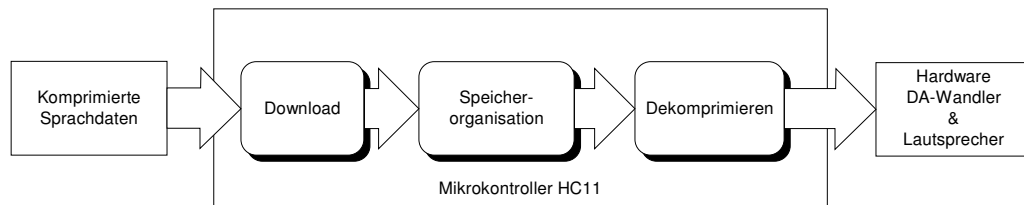


ABBILDUNG 16 AUFBAU DER HARDWARE

Für die Realisierung steht das Entwicklungsboard SPACE von der Firma Ascom System AG zur Verfügung (siehe Kapitel 4.3.1 und Unterlagen SPACE [3]).

#### 4.3.1. Aufbau und Einsatzmöglichkeit vom Entwicklungsboard SPACE

Das SPACE ist ein multifunktionales Board, das sich vor allem gut eignet, digitale Signale zu verarbeiten. Es besteht aus mehreren Blöcken:

- Mikrokontroller Motorola MC68HC11
- DSP Texas Instruments TMS320C50
- FPGA XC5201-6
- Diverse Kommunikationsschnittstellen (DUART, Parallel Ports, SPI)
- Speicheransteuerung- und verwaltung
- AD- und DA-Wandler

Die Berechnung bzw. Dekomprimierung der Samplewerte muss laut Aufgabenstellung mit dem Motorola M68HC11 erfolgen (siehe Kapitel 1). Die Software dafür wird mit der Entwicklungsumgebung "IAR Embedded Workbench Version 2.10C" erstellt, die von der ASCOM System AG zur Verfügung gestellt wird. Die genaue Beschreibung ist im "UserGuide" [11] vom IAR ersichtlich.

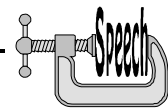
Für die Ausgabe der Sprachsamples, gibt es verschiedene Möglichkeiten. Es steht zur Auswahl:

- Die Samples parallel oder mit der SPI-Schnittstelle auf einen externen DA-Wandler geben.
- Den DA-Wandler auf dem SPACE verwenden. Das hat aber zur Folge, dass das FPGA dafür modifiziert werden muss.

Die zweite Lösung gibt weniger Aufwand an Hardware. Darum wird diese Variante im System realisiert.

Damit man an das System direkt einen passiven Lautsprecher anschließen kann, wird das analoge Ausgangssignal vom DA-Wandler mit einem Audio-IC verstärkt.





- Wenn möglich für das Dekomprimieren keine Multiplikationen ausführen. Sie brauchen sehr viele Taktzyklen. Der HC11 benötigt 10 Taktzyklen, um zwei 8-Bit-Integer Zahlen zu multiplizieren.

#### 4.3.3.1. Datenformat der Sprachsamples für den DA-Wandler

Der DA-Wandler wandelt nur 'Signed Integer'-Werte. Die Sample sind aber als 'Unsigned Integer' gespeichert. Das heisst, bevor sie dem DA-Wandler übergeben werden, müssen sie in signed umgerechnet werden.

	kleinste Zahl						grösste Zahl		
unsigned Integer:	00	01		7F	80	81		FE	FF
signed Integer:	FF	FE		80	00	01		7E	7F
Dezimalzahl:	-128	-127		-1	0	+1		+126	+127

ABBILDUNG 18 UMRECHNUNGSTABELLE VON UNSIGNED AUF SIGNED INTEGER

#### 4.3.3.2. Expansion von Sprachsamples, die mit $\mu$ Law komprimiert wurden

Sprachsamples, die mit der  $\mu$ Law Kennlinie komprimiert worden sind, müssen vor dem Ausgeben an den DA-Wandler, expandiert werden. Am schnellsten geht dies, wenn man ein Array im Speicher initialisiert, das schon die berechneten Ausgangswerte enthält. Die Tabelle 1 zeigt die Umrechnung von Samples, die mit 4-Bit quantisiert und komprimiert wurden. Tabellen für 5 und 6 Bit sind im Anhang C angefügt.

Gespeicherter Sample	Expandierter Sample	mit Vorzeichen
0	0	80
1	28	A8
2	45	C5
3	59	D9
4	67	E7
5	71	F1
6	78	F8
7	7D	FD
8	81	01
9	86	06
A	8E	0E
B	98	18
C	A6	26
D	BA	3A
E	D7	57
F	FF	7F

TABELLE 1 EXPANSIONSTABELLE FÜR 4-BIT SAMPLES

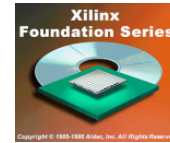
#### 4.3.3.3. Sprachsamples, die ohne $\mu$ -Law komprimiert wurden

Bei diesen Samples fällt das Expandieren weg. Sie müssen lediglich in das 'Signed Integer'-Format gebracht werden. (Siehe Abbildung 15)

#### 4.3.4. Ausgabe

Hier wird das Konzept der Hardware und die dafür benötigten Tools erläutert.

##### 4th3.4.1. Die Xilinx Foundation Series Software F1.4



Auf dem SPACE befindet sich ein FPGA XC5201\_6 von Xilinx. Unser Betreuer, Herr A. Ehernsperger, hat für das FPGA Design die Xilinx Foundation Series Software besorgt. Allerdings gab es bei der Installation Probleme. Die Software liess sich nicht registrieren, weil dafür eine Nummer nötig war, die sich aber auf dem Postpaket der Software befunden hat. Dieses Paket war leider nicht mehr vorhanden. Nach längerer Verhandlung wurde vom Xilinx-Vertreter eine temporäre Lizenz ausgestellt, die bis zum Ende der Diplomarbeit gültig ist.

Mit diesem Software-Paket von Aldec Inc. kann ein beliebiges XILINX FPGA programmiert werden. Mit dem Schematic Editor kann man bequem ein Digitaldesign zeichnen. Mit dem HDL Editor werden logische Funktionen in HDL programmiert. Mit dem State Editor kann man State-Machines zeichnen. Diese drei Komponenten kann man auch kombinieren. Im Schema können Blöcke vorkommen, die in HDL programmiert, oder als State Machine erzeugt wurden. Dieses Logik-Design kann man nun simulieren. Achtung, es handelt sich lediglich um eine logische Simulation! Die Laufzeiten werden hier nicht eingerechnet. Nach der Fertigstellung des Logik-Design synthetisiert man es mit dem Button 'Implement'.

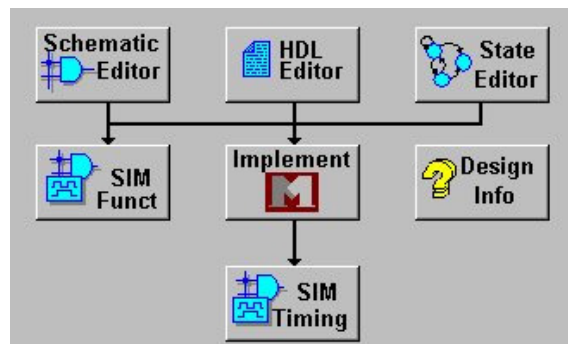


ABBILDUNG 19 DESIGN-FLOW EINES XILINX-PROJEKTS

Dann wird der Design Manager geöffnet. Hier kann man Versionen und Revisionen des Projekts verwalten. Es gibt immer eine neue Version, wenn man etwas am Logik-Design geändert hat. Eine neue Revision ist lediglich eine erneute Synthetisierung mit anderen Einstellungen. Für die Synthese betätigt man die Play Taste. Man kommt in ein Auswahlmenu. Hier kann man über die Taste 'Options...' die Optionen einstellen. Wenn man etwas simulieren will, muss unter 'Optional Targets' 'Produce Timing Simulation Data' angewählt werden.

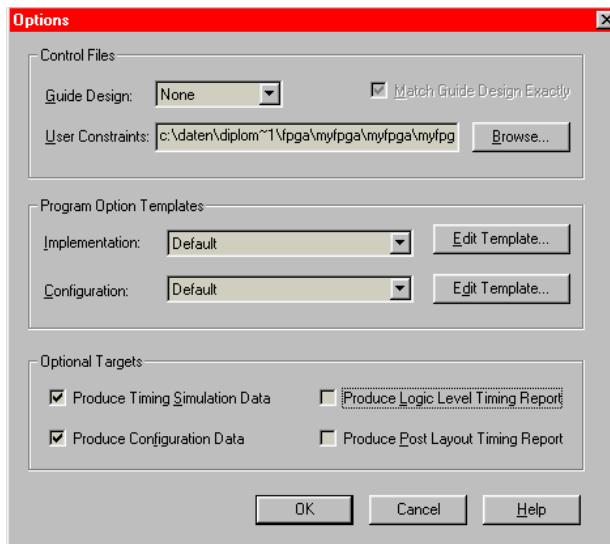


ABBILDUNG 20 'OPTIONS' FENSTER BEI DER SYNTHESE

Wenn die Synthetisierung nicht erfolgreich sein sollte, werden Reports erstellt, die alle Errors und Warnings beschreiben. Nach erfolgreicher Synthese kann man ein File für den EEPROM-Brenner in verschiedenen Dateiformaten erzeugen.

#### 4.3.4.2. Konzept für die FPGA Programmierung

Anforderungen:

- Vorhandene Funktionen des FPGAs für die Steuerung der verwendeten SPACE Hardware bleiben erhalten.
- Register für die Sprachdaten und die Frequenzwahl bereitstellen.
- Takterzeugung für eine Samplingfrequenz zwischen 6kHz und 8 kHz für den DA-Wandler.
- Serielle Ausgabe der Sprachdaten.
- Interrupt, wenn neue Daten benötigt werden.

Aus diesen Anforderungen resultierte am Ende eine grobe Struktur:

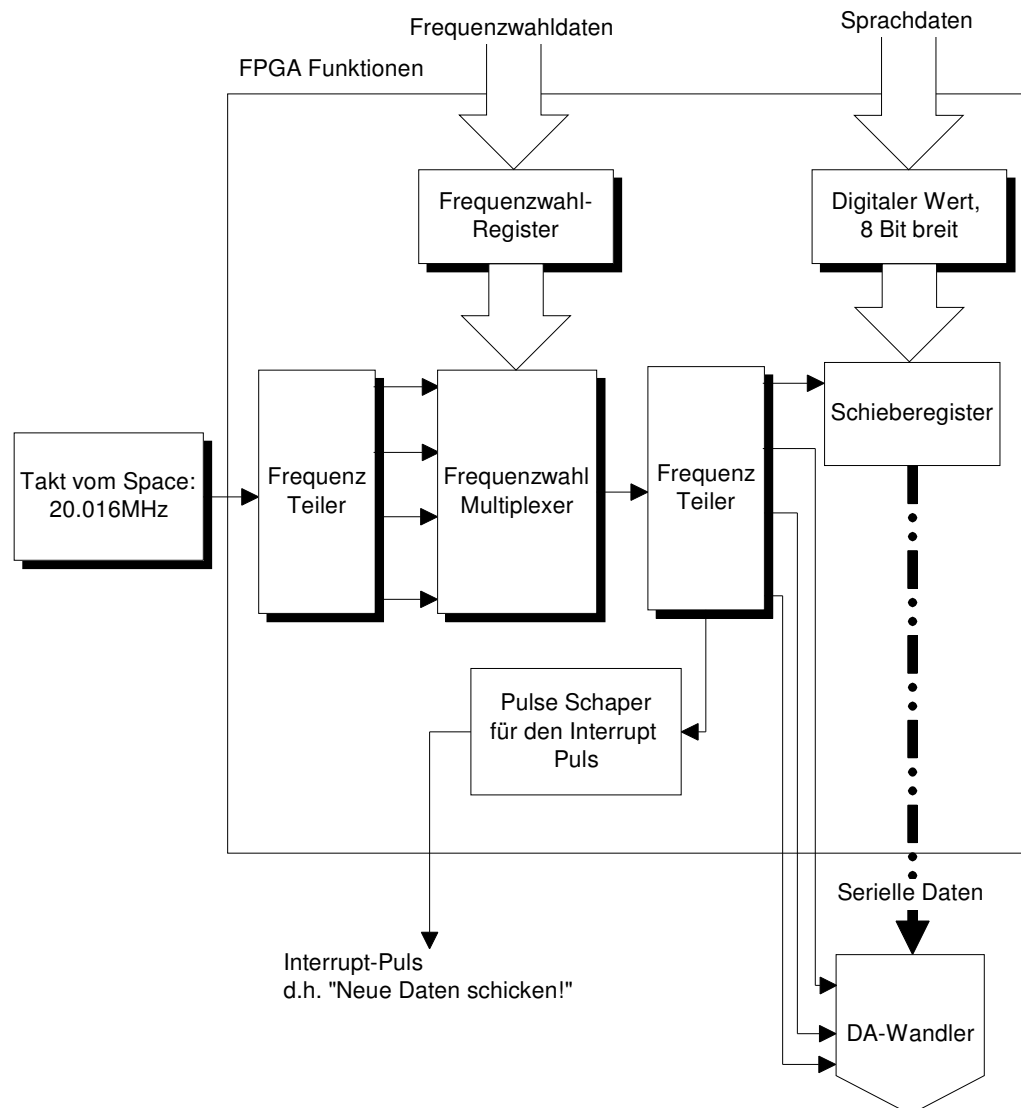
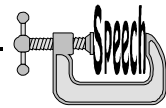


ABBILDUNG 21 STRUKTUR DES FPGA-DESIGNS

Auf dem SPACE steht ein Takt von 20.016 MHz zur Verfügung. Dieser wird heruntergeteilt. Die Teiler sind so gewählt, dass schlussendlich am DAC vier verschiedene Samplingfrequenzen zwischen 6kHz und 8kHz vorhanden sind, und gleichzeitig jeweils zwei weitere Frequenzen im Verhältnis 1:256 zur Verfügung stehen, um den Modus des DAC einzustellen (genaue Spezifikationen Anhang J). Daraus folgen im ersten Frequenzteiler Teilerraten von 10, 11, 12 und 13.

Berechnung der Frequenzen:

$$\begin{aligned} \text{LRCK0} &= 20.016\text{MHz} : 13 : 256 = 6.014 \text{ kHz} \\ \text{LRCK1} &= 20.016\text{MHz} : 12 : 256 = 6.516 \text{ kHz} \\ \text{LRCK2} &= 20.016\text{MHz} : 11 : 256 = 7.108 \text{ kHz} \\ \text{LRCK3} &= 20.016\text{MHz} : 10 : 256 = 7.819 \text{ kHz} \end{aligned}$$



Beim Design wurde darauf geachtet, dass alle Bausteine mit einem synchronen Takt laufen. Dies ist bei einem Digitaldesign immer anzustreben. Das FPGA bietet zudem auf der Hardware vernetzte Clockleitungen, die bei solchen Synchronclocks verwendet werden sollten. Dazu dient im Schematic Editor der Foundation Software das Bauteil 'BUFG'.

#### 4.3.4.3. Der D/A Wandler CS4331 von Crystal (Datenblatt Anhang J)

Dieser DAC ist für CD-Player und portable Geräte gedacht. Er benutzt eine Delta-Sigma Modulation um die seriellen Eingangsdaten Analog zu Wandeln. Die Samplingrate kann stufenlos mit der Master Clockfrequenz (MCLK) zwischen 2kHz und 50kHz eingestellt werden. Der DAC arbeitet mit einer Betriebsspannung von +3V oder +5V. Der DAC arbeitet nach einem bestimmten Ablauf (Siehe Datenblatt Seite 7):

- Wandlung der seriellen Daten in parallele Daten.
- Digitale Interpolation mit dem Faktor 128 ( 32 mal 4 )  
Faktor 32: Den Eingangswerten werden jeweils 31 Nullen angefügt, dann wird mit einem digitalen Filter Interpoliert.  
Faktor 4: Vierfaches Digitales Sample&Hold. Damit wird die Samplingfrequenz nochmals um Faktor 4 erhöht.
- Delta-Sigma Modulation wandelt die Daten in 1 Bit Daten.
- Delta-Sigma Analog Konversion. Die 1 Bit Daten werden in eine Serie von Ladungspaketen verwandelt. Ein 1 Bit Wert bestimmt jeweils die Polarität des Ladungspakets. Die Ladung wird anhand einer hochpräzisen Spannungsreferenz ermittelt.
- SC-Filterung: Das SC-Filter wird der Samplingfrequenz angepasst.

Am Ausgang reicht nun ein Tiefpass erster Ordnung, um die Vielfachen des Eingangsspektrums bei der 128-fachen Samplingfrequenz wegzufiltern. Diese Vielfachen entstehen durch das SC-Filter.

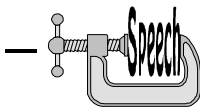
Den DAC kann man mit verschiedenen Betriebsmodi betreiben. Diese können mit dem MCLK eingestellt werden.

Die Pins des DAC haben folgende Funktionen:

- SDATA: Eingang für die seriellen Daten.
- SCLK: Serieller Takt. Dieser Pin wird verwendet, um die Daten seriell einzulesen. Wenn kein Takt angelegt wird, kommt der DAC in den 'Internal SCLK' Modus. Der Serielle Clock wird dann vom DAC intern erzeugt. Beim Internal Mode wird mit diesem Pin Deemphasis aktiviert oder deaktiviert. Deemphasis ist eine Absenkung der hohen Frequenzen, um den Rauschabstand zu verringern. Diese Funktion wird für Aufnahmen mit Preemphasis verwendet. Es wird im Datenblatt empfohlen, wenn möglich den 'Internal SCLK' Modus zu verwenden, um Clock Jitter zu vermeiden.
- LRCLK: Legt fest, wann der rechte und wann der linke Kanal verwendet wird.
- MCLK: Mit diesem Pin legt man fest, welchen Betriebsmodus man verwendet.

#### 4.3.4.4. EDA/Client 98 von Protel

Da im Labor für Mikroelektronik die neue Software EDA/Client 98 von Protel vorhanden ist, und dort auch von anderen Diplomteams gebraucht wird, haben wir uns für dieses Layouttool entschieden. Protel bietet ausser dem Zeichnen von Schemas und Layouts noch viel mehr Funktionen. Die elektrischen Schemas können vor dem Schema Design simuliert werden. Das Schema Layout Tool hat eine Autoroute Funktion, die bei sachgemässer Anwendung sehr leistungsfähig ist. Durch die Komplexität dieser Umgebung braucht es eine gewisse Zeit, um sich einzuarbeiten. In dieser Arbeit wurde vor dem Design ein Tutorial durchgearbeitet (Befindet sich als PDF-File auf der Diplom-CD), was die Arbeit nachher sehr erleichtert hat.



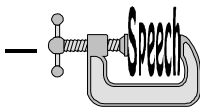
#### 4.3.4.5. Konzept Huckepackprint

Der Print soll folgende Funktionen erfüllen:

- Alle benötigten Anschlüsse sollen über diesen Print zugänglich sein.
- Der Analogausgang des SPACE wird mit einem Audioverstärker verstärkt und auf eine Lautsprecherbox ausgegeben.
- Zwei Taster, 2 rote LEDs und 2 grüne LEDs werden an einem PORT des HC11 angeschlossen.

Als Audioverstärker wurde uns von Herr Hinn aus dem Mikroelektronik-Labor ein Analogverstärker aus der Boomer-Familie von National vorgeschlagen. Diese Familie erscheint auf Anhieb ideal für unsere Anwendung. Diese Komponenten sind für Leistungen zwischen 0 und 2 Watt konzipiert und verbrauchen wenig Strom, was für den Einsatz in einem mobilen System wichtig ist. Zudem ist die äussere Beschaltung denkbar einfach. Wir haben uns für den Typ LM4860 entschieden, der eine Ausgangsleistung von 1 Watt gewährleistet und für Einkanal Anwendungen (mono) gemacht ist.





## 5. Implementation GUI

### 5.1. Einführung

Bei der Implementation des GUI wird vor allem darauf geachtet, dass möglichst schnell ein lauffähiger Prototyp besteht. Diesen Schritt können wir plangemäss mit dem Guide-Tool von MATLAB realisieren. Nachher geht es darum, das GUI möglichst benutzerfreundlich und komfortabel zu machen.

Das GUI besteht aus einem Hauptprogramm mit Namen `squeeze()`. Dieses wird als Callbackfunktion mit dem Kommando als Parameter aufgerufen. Einzig beim Aufstarten wird `squeeze` ohne Parameter aufgerufen (→ Softwarekonzept Kapitel 3.3.5).

### 5.2. Globale Variablen

Folgende Variablen werden **global** gebraucht:

Alle Uicontrol Handles	Class: double array Diese Variablen sind mit <code>h_</code> am Anfang der Variable gekennzeichnet (z.B. <code>h_recordbutton</code> )
<code>currentpathname</code>	Class: char array Speichert aktuellen Pfad. Am Programmanfang ('Initialize') wird der aktuelle Pfad auf 'c:\' gesetzt.
<code>wave</code>	Class:struct array Enthält alle Informationen des mit dem 'Load *.wav' - Button eingelesenen Wave-Files. (Sprachdaten, Samplingfrequenz, Bitbreite, Filename, Pfad).
<code>mat</code>	Class:struct array Enthält alle Informationen des komprimierten Sprachsamples. Wird beim Betätigen des 'Save Squeezed' – Buttons gespeichert und beim Betätigen von 'Load Squeezed' aktualisiert.
<code>decod</code>	Class:struct array Beim Abspielen des komprimierten Sprachsamples wird der decodierte Sprachsample mit den zugehörigen Informationen gespeichert.
<code>s2</code>	Class:struct array Filename und Pfad der <code>s2</code> -Datei.
<code>sqzpar</code>	Parameter für die Komprimierung. Wird durch das Betätigen der Uicontrols im Parameter-Frame verändert. Wird beim Betätigen des 'Save Squeezed' – Buttons gespeichert und beim Betätigen von 'Load Squeezed' aktualisiert.

### 5.3. Funktionsstruktur

Das Hauptprogramm ruft folgende Funktionen auf:

GUI-Funktionen	werden gebraucht, um die Oberfläche darzustellen. Es sind alles Matlab-Funktionen.
Komprimierfunktionen	werden gebraucht, um die Sprachdaten zu komprimieren. Zum Teil handelt es sich um Matlab Funktionen. Die meisten Funktionen sind in der Semesterarbeit [2] entwickelt worden.
Konvertierungsfunktion	wird gebraucht, um die codierten Sprachsamples ins Motorola <code>s2</code> Format zu konvertieren und in einem File mit Endung <code>s2</code> automatisch zu speichern.
Allgemeine Matlab Funktionen	werden von allen Programmteilen gebraucht. Hier werden nur ein paar ausgesuchte, nicht alltägliche Funktionen erläutert.

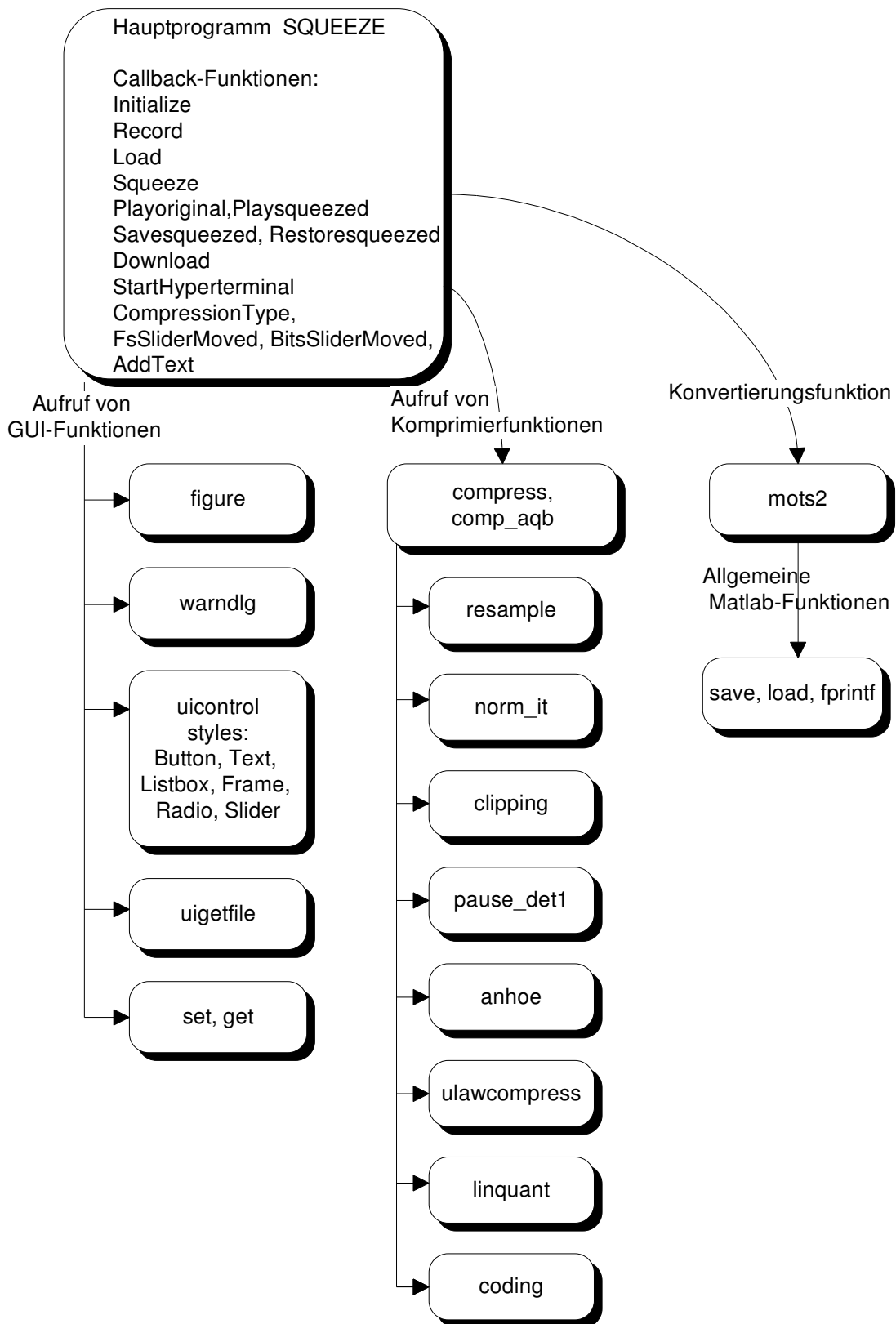


ABBILDUNG 22 FUNKTIONSSTRUKTUR

## 5.4. Funktionsbeschreibungen

### 5.4.1. Die Funktion squeeze()

Übergabewert: Command (Name der Callbackfunktion)  
Arg1 (Wird abhängig von Command verwendet)  
Rückgabewert: Keiner

Beschreibung:

Hauptprogramm für die Erstellung und interaktive Steuerung des GUIs. Die Funktion squeeze besteht, wie schon in (Kapitel 3.3.5) erwähnt wurde, aus einem Case. Der Case in Matlab kann auch mit Char array gebraucht werden.

Die Programmstruktur sieht dann so aus:

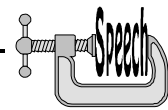
squeeze.m

```
function squeeze(Command, arg1)

if nargin < 1
    Command = 'Initialize';
end

switch Command
case 'Initialize'
    % Initialisierung der GUI Objekte und Variablen
    .
    .
case 'Record'
    % Callbackfunktion
    .
    .
case 'Load'
    % Callbackfunktion
    .
    .
end
```

ABBILDUNG 23 PROGRAMMSTRUKTUR VON SQUEEZE



#### 5.4.1.1. Callbacks von squeeze

Die Callback Funktionen der Uicontrol Objekte sind alle von der Form **squeeze('Record')**. Der Befehl **squeeze Record** hat die gleiche Bedeutung.

CallBacs von Squeeze	Initialize	Initialisierung aller Variablen, Figuren und Uicontrol-Objekte(Buttons, Sliders, Text, RadioButtons,...)
	Record	Startet den Audiorecorder
	Load	Lädt eine Wave Datei in den Workspace
	Squeeze	Komprimiert und Codiert den Sprachsample mit den Komprimier-Funktionen. Wenn als Komprimiermethode AQB aktiviert wurde, kommt eine Fehlermeldung, dass die Dekomprimierung von Sprachsamples, die mit AQB komprimiert wurden, auf dem Mikrokontroller nicht möglich ist. Der komprimierte Sprachsample wird dann sofort abgespielt.
	Playoriginal	Spielt den Sprachsample im Originalformat ab.
	Playsqueezed	Spielt den komprimierten Sprachsample (nach squeeze) ab.
	Savesqueezed	Speichert den komprimierten Sprachsample in einem Mat-File.
	Restoresqueezed	Stellt einen gespeicherten Sprachsample wieder her und aktualisiert die Squeezeparameter.
	Download	Speichert den komprimierten Sprachsample im Motorola s2-Format und verlangt nach einem Servicestring. Der Benutzer wird dazu Aufgefordert, das Hyperterminal zu starten und das s2-File herunterzuladen.
	StartHyperterminal	Startet das Hyperterminal.
	CompressionType	Wird beim anklicken eines RadioButtons aufgerufen.
	FsSliderMoved	Wird beim anklicken des Frequenzwahlsiders aufgerufen.
	BitsSliderMoved	Wird beim anklicken des Bits –Sliders aufgerufen.
	AddText	Schreibt Text auf die Unterste Zeile des Squeeze-Statusfensters. Wird von den meisten Funktionen gebraucht, um ausgeführte Aktionen im Statusfenster anzuzeigen.

## 5.4.2. Komprimier-Funktionen

Komprimier-Funktionen	compress	Wird von squeeze aufgerufen, um den Sprachsample zu komprimieren. Der Sample kann mit oder ohne $\mu$ Law komprimiert werden.
	comp_aqb*	Wird von squeeze aufgerufen, wenn als Komprimiermethode die adaptive Quantisierung gewählt wurde. Komprimiert das Signal mit Hilfe der adaptiven Quantisierung.
	resample	Matlab-Funktion für die Änderung der Abtastrate eines Signals.
	norm_it*	Entfernt den Gleichanteil eines Signals und skaliert es auf den maximalen Spitzenwert von 1.
	clipping*	Schneidet Signalspitzen ab (weiche Begrenzung).
	pause_det1*	Schneidet Pausen am Anfang und Ende ab. Detektiert Pausen während dem Signal.
	anhoe*	Hebt die hohen Frequenzen an.
	uLawcompress*	Nichtlineare, exponentielle Verformung der Signalamplitude.
	linquant*	Lineare Quantisierung des Signals.
	coding	Codierung der Pausen.
Die mit * gekennzeichneten Funktionen wurden unverändert von der Semesterarbeit [2] übernommen		

### 5.4.2.1. Die Funktion compress()

Übergabewert: Sprachdatenvektor  
 Originalsamplingfrequenz  
 Originalbittiefe  
 Komprimiermodus  
 neue Samplingfrequenz  
 neue Bittiefe  
 Clipping-Faktor

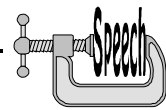
Rückgabewert: Codierter Sprachdatenvektor

Beschreibung:

Komprimiert und codiert das Sprachsignal. Die Methode wird mit dem Übergabewert *Komprimiermodus* bestimmt. Mögliche Methoden sind Komprimierung mit und ohne  $\mu$ Law. Als Pausendetektion wird die Funktion `pause1()` verwendet. Diese Funktion wurde nicht neu geschrieben. Sie entstand durch Modifikation der Funktion `ulawcompress()` aus der Semesterarbeit [2]. Der grosse Unterschied ist die Anzeige der ausgeführten Schritte: die Ausgabe erfolgt nun in die Minikonsole des Voice Squeezers, anstatt wie vorher ins Matlab-Fenster.

### 5.4.2.2. Die Funktion coding()

Die genaue Beschreibung dieser Funktion ist in der Dokumentation der Semesterarbeit SS-98-06 [2] Seite 47 beschrieben. Neu hinzu kommt, dass das Ende eines Sprachsamples anhand der Codierung des Sprachsamples im Mikrokontrollersystem erkannt werden kann. Das bedeutet, die Codierung des Sprachdatenvektors muss modifiziert werden.



Die Codierung muss folgendes anzeigen können:

- Sprachpausen und deren Länge
- Ende eines Sprachsamples

Aufbau des Codes:

Der Wert '0' im Sprachdatenvektor gibt an, dass ein Code folgt der wie folgt aussehen kann:

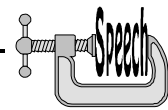
- a) '0 1 H L' → Folgt nach der Null eine '1', bedeutet es, dass danach eine Sprachpause folgt. Die Länge der Pause ist so festgelegt, dass 'H' die höherwertigen Bits und 'L' die niederwertigen Bits plus die Anzahl Werte für die Minimalpausenlänge von 5ms ist. Dieser Wert entspricht gerade der Anzahl Pausenwerte, die der Mikroprozessor wieder einfügen muss.
- b) '0 3' → Eine '3' nach der Null zeigt das Ende des Sprachsamples an.
- c) '0 0' → Eine '0' nach der Null bedeutet, dass an dieser Stelle im Sprachdatenvektor wirklich die Null steht.

x	x	x	0	1	H	L	x	x	x	0	0	x	x	x	x	x	0	3
			Pause							Wert '0'						Ende		

ABBILDUNG 24 MÖGLICHE CODES IM SPRACHDATENVEKTOR

### 5.4.3. Gui-Funktionen

GUI-Funktionen	figure	Erzeugt ein neues Figur Fenster und gibt den Handle zurück. Mit dem Befehl <b>figure(handle)</b> kann man ein bestimmtes Fenster aktivieren.
	warndlg	Erzeugt ein neues Fenster mit einer Warnmeldung.
	uicontrol	Erzeugt ein Objekt vom Typ 'Uicontrol'. Es gibt 10 verschiedene Typen, die mit dem Attribut 'Style' unterschieden werden.  Folgende Styles sind möglich:
	pushbutton <sup>1)</sup>	Viereckiger Knopf
	togglebutton	Viereckiger Knopf, der on oder off ist.
	radiobutton <sup>1)</sup>	Runder Knopf; schwarz, wenn angewählt.
	checkbox	Eckiger Knopf; schwarz, wenn angewählt.
	edit	Textfeld, das vom User editiert werden kann.
	text <sup>1)</sup>	Textfeld
	slider <sup>1)</sup>	Scrolbar
frame <sup>1)</sup>	Eckige Fläche; Erleichtert die Übersicht.	
listbox <sup>1)</sup>	Auswahlbox mit Scrollbar; Wird hier als Konsolenfenster "Misbraucht".	
popupmenu	Erzeugt ein Pop-Up Menu	
uigetfile	Erzeugt ein Windows Standard Dialogfenster, um ein File zu öffnen und gibt den Filenamen und den Pfad des ausgewählten Files zurück.	
set	Setzt die Eigenschaften (Farbe, Position, Callbackfunktion) einer Figur oder eines Uicontrol-Objekts.	
get	Gibt die Eigenschaften einer Figur oder eines Uicontrol-Objekts in einem struct zurück.	
		<sup>1)</sup> Diese Styles wurden verwendet.



### 5.4.4. Die Funktion mots2()

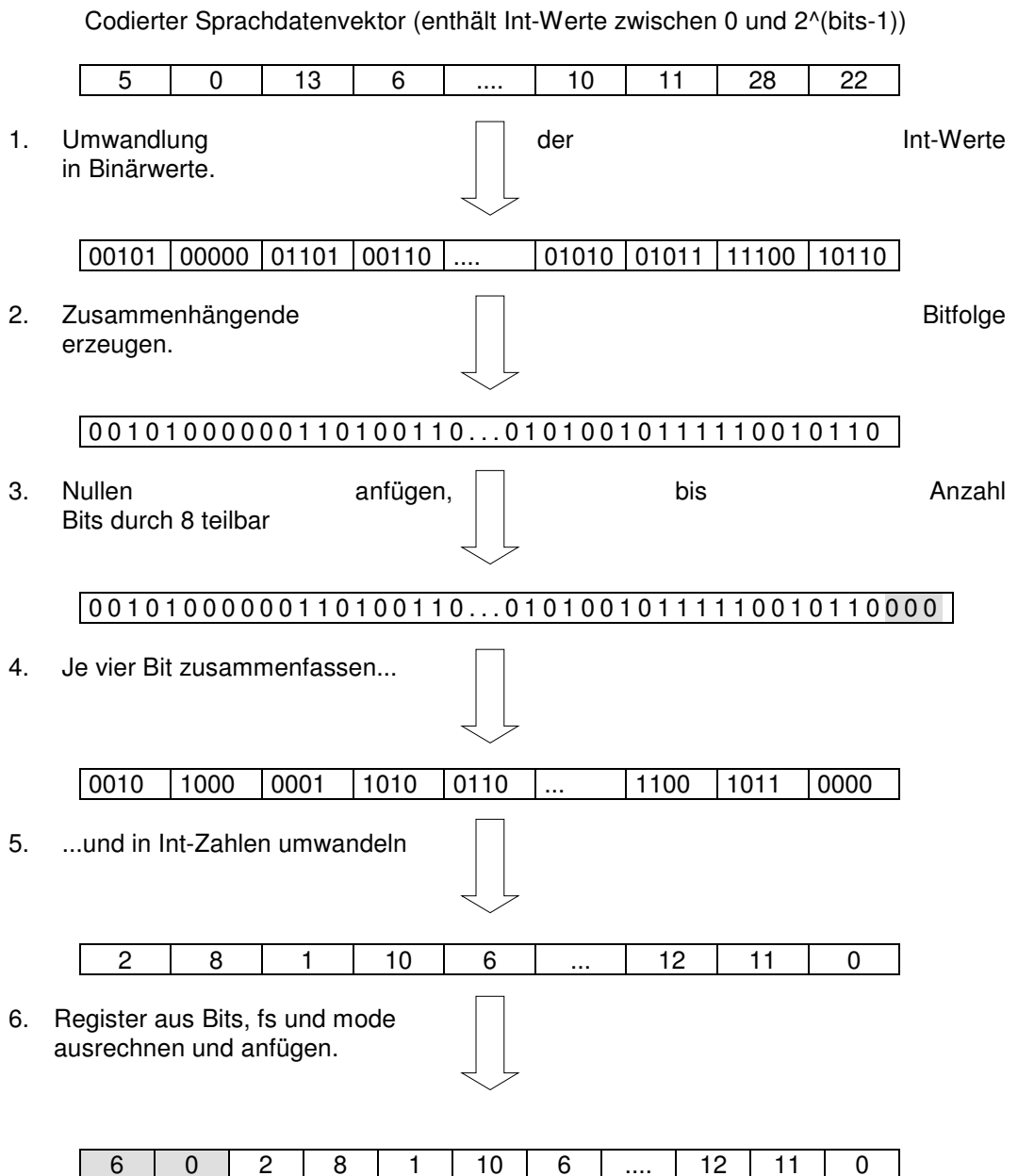
Übergabewert: codierter Sprachdatenvektor (Zeilenvektor)  
 Samplingfrequenz  
 Quantisierungsbitbreite  
 Servicetext  
 Pfad des S2 Files  
 Filename des S2 Files

Rückgabewert: Keiner

Beschreibung:

Diese Funktion mots2() hat die Aufgabe, den Sprachsample im **Motorola S2** Format in ein File zu schreiben.

Der Aufbau der Funktion ist hier anhand eines Beispiels mit der Quantisierungs-Bitbreite 5 Bit durchgespielt: (Servicetext: 'ab')

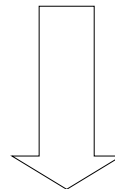


7. Servicetext anfügen  
und mit 00 abschliessen



6	1	6	2	0	0	6	0	2	8	1	10	6	....	12	11	0
---	---	---	---	---	---	---	---	---	---	---	----	---	------	----	----	---

8. Diese Daten in Hex-Zahlen wandeln und mit den Zusatzzeichen und den Adressen in File speichern (mit dem Befehl `fprintf(fid, '%X', Daten)` ). Als Checksumme wird 0xFF eingesetzt.



Test.s2

S001FF	S20D00000061620068281A6...
S801FF	...52F96FF

ABBILDUNG 25 BEISPIEL EINES ABLAUFS VON MOTS2

#### 5.4.5. Diverse Matlab-Funktionen

Diverse Matlab-Funktionen	save	Speichert den Matlab Workspace in einem File mit Endung '*.mat'.
	load	Stellt den mit save gespeicherten Workspace wieder her.
	fprintf	Schreibt eine Variable mit einer bestimmten Formatierung in eine Textdatei.
	struct	Erzeugt eine Variable mit dem Datentyp Struct. Ermöglicht eine sehr übersichtliche Programmierung.



## 6. Implementation Hardware

### 6.1. Einleitung ins Kapitel

In diesem Kapitel wird die Implementation der HC11-Software schematisch erklärt. Wichtige Informationen des FPGA und Huckepackprintdesign sind dokumentiert.

### 6.2. HC-11 Software

#### 6.2.1. Einführung

Bei der Implementation wird darauf geachtet, dass das Dekomprimieren und Ausgeben, zeitoptimiert ist. Daher sind alle zeitkritischen Routinen so programmiert, dass sie möglichst wenig Speicherzugriffe und Subrutinenaufrufe machen müssen. Das hat aber zur Folge, dass das Programm erheblich mehr Speicherplatz braucht.

#### 6.2.2. Übersicht der Programmkomponenten

Die einzelnen Komponenten sind nach der folgenden Abbildung erläutert.

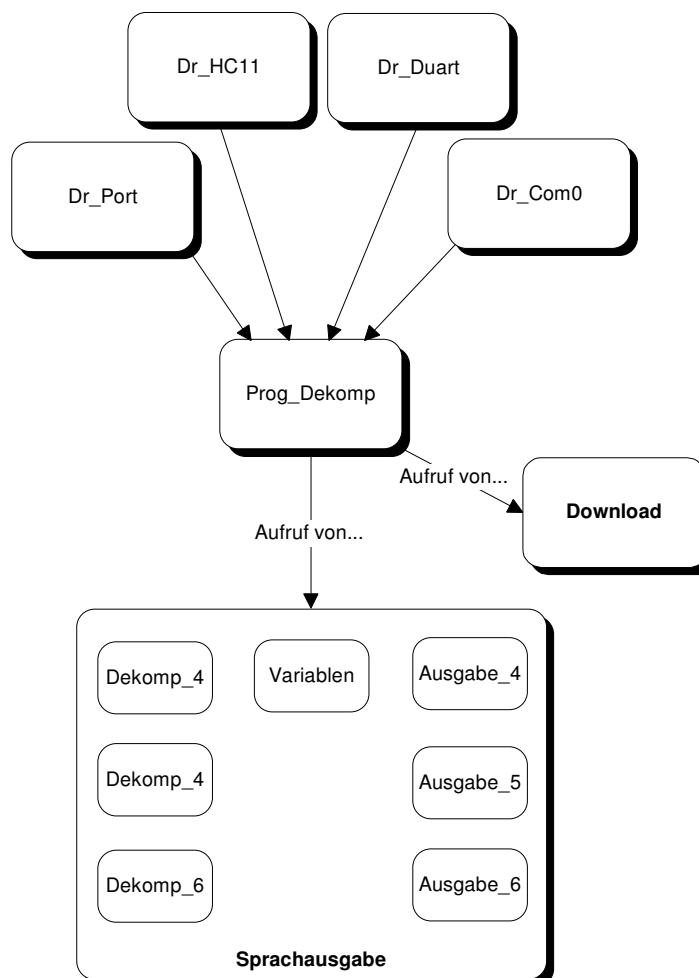
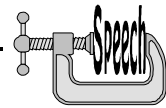


ABBILDUNG 26 ÜBERSICHT ÜBER ALLE KOMponentEN DER HC11-SOFTWARE



### 6.2.3. Erläuterungen zu den Komponenten

#### 6.2.3.1. Prog\_Dekomp

Diese Komponente beinhaltet das Main. Sie initialisiert die benötigten Ports und Hardware und steuert die Benutzer- bzw. Menüführung.

#### 6.2.3.2. Dr\_Port

Steuert die Funktionen des Portbausteins "MC 146823"

#### 6.2.3.3. Dr\_HC11

Erstellt die Interruptvektortabelle und beinhaltet Interruptsubroutinen.

#### 6.2.3.4. Dr\_Duart

Steuert die Funktionen der UART "68C681".

#### 6.2.3.5. Dr\_Com0

Steuert die Funktionen der seriellen Schnittstelle des HC11.

#### 6.2.3.6. Download

Download von Sprachsamples. Ladet Sprachsamples von maximaler Grösse von 64 kB über die serielle Schnittstelle 'Com0' in das ROM.

#### 6.2.3.7. Sprachausgabe

Decodiert heruntergeladene Sprachsamples, und gibt sie an den DA-Wandler aus.

Dekomp\_x: Decodiert und expandiert die Sprachsamples, die mit x-Bittiefe quantisiert worden sind.  
Die Sprachsamples sind mit  $\mu$ Law komprimiert worden.

Ausgabe\_x: Hier werden die Sprachsamples nur decodiert, die mit x-Bittiefe quantisiert worden sind.  
Das Expandieren fällt weg, weil die Sprachsamples nicht mit  $\mu$ Law komprimiert worden sind.

Variablen: Variablendeklarationen, die von den Komponenten Dekomp\_x und Ausgabe\_x benötigt werden.

### 6.2.4. Beschreibung der Software

Das Programm besteht aus den Funktionen: Benutzermenü, Download und Sprachausgabe von Sprachsamples. In dieser Version der Software ist es nur möglich, jeweils einen Sprachsample herunterzuladen und auszugeben.

### 6.2.4.1. Benutzermenü

Aus dem Benutzermenü können drei verschiedene Aktionen gestartet werden. Anhören eines Sprachsamples (siehe Kapitel 6.2.4.3), Ausgeben des Servicetextes und Starten eines Download (siehe Kapitel 6.2.4.2).

Die Ausgabe des Servicetextes dient dazu, um zu überprüfen, welcher Sprechtext auf das System heruntergeladen worden ist. Der Servicetext ist im Kapitel 3.3.13 beschrieben.

### 6.2.4.2. Download

Der Download wurde grösstenteils von Herrn Altwegg übernommen. Folgende Abbildung zeigt den groben Ablauf von einem Download eines Sprachsamples:

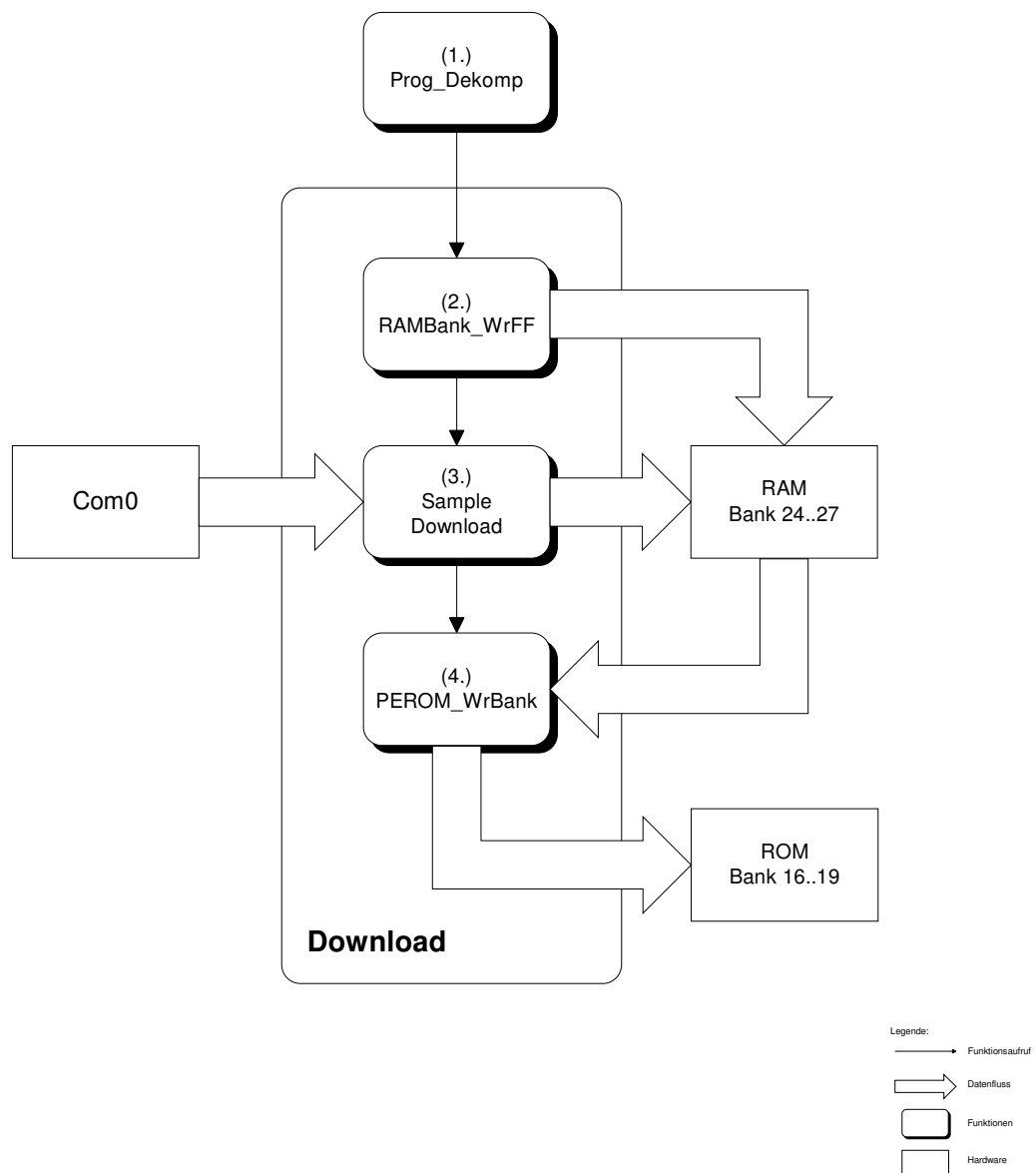
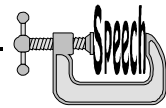


ABBILDUNG 27 ABLAUF VON EINEM DOWNLOAD EINES SPRACHSAMPLES



Ablauf:

- (1.) Aus dem Benutzermenü wird die Routine 'Download' aufgerufen.
- (2.) Die RAM-Banken, in die die Samples kopiert werden, werden zuerst mit 'FF' beschrieben.
- (3.) Die Sprachdaten können jetzt vom Terminal aus der seriellen Schnittstelle 'Com0' gesendet werden. Der Funktionsteil Sampledownload wartet auf den Anfang der Daten. Sobald ein 'S0' gesendet wird (siehe Kapitel 3.3.13) beginnt der Download der Sprachdaten. Diese werden zuerst in die RAM-Banken 24..27 geschrieben. Überschreiten die Daten die Grösse von 64 kByte, oder wird eine Codeverletzung des Motorola S2 Format entdeckt, wird der Download abgebrochen, und ein Error wird ausgegeben.
- (4.) Sind die Daten vollständig heruntergeladen worden, werden sie jetzt eins zu eins von den RAM-Bänke 24..27 in die ROM-Bänke 16..19 kopiert. Danach ist der Download abgeschlossen, und der Sprachsample kann vom Benutzermenü ausgegeben werden.

#### 6.2.4.3. Sprachausgabe

Die Sprachausgabe kann vom Benutzermenü aus oder mit der grünen Taste auf dem Print gestartet werden.

Das Ausgeben von Sprachsamples an den DA-Wandler ist interruptgesteuert. Die dekomprimierten Sprachsamples werden jeweils in der Variable 'SprachDat' gespeichert. Wenn das FPGA an PA2 einen Interrupt auslöst, ladet der HC11 diesen Sprachsample mittels einer Interruptsubroutine ins Register im FPGA, von dort aus er dann mit dem DA-Wandler ausgegeben wird. Dazwischen hat der Prozessor Zeit, um einen weiteren Sample vom ROM zu lesen, zu dekomprimieren und ihn in die Variable 'SprachDat' zu speichern. Dieser Vorgang wurde mit eine Flag synchronisiert, das weiter unten erklärt wird.

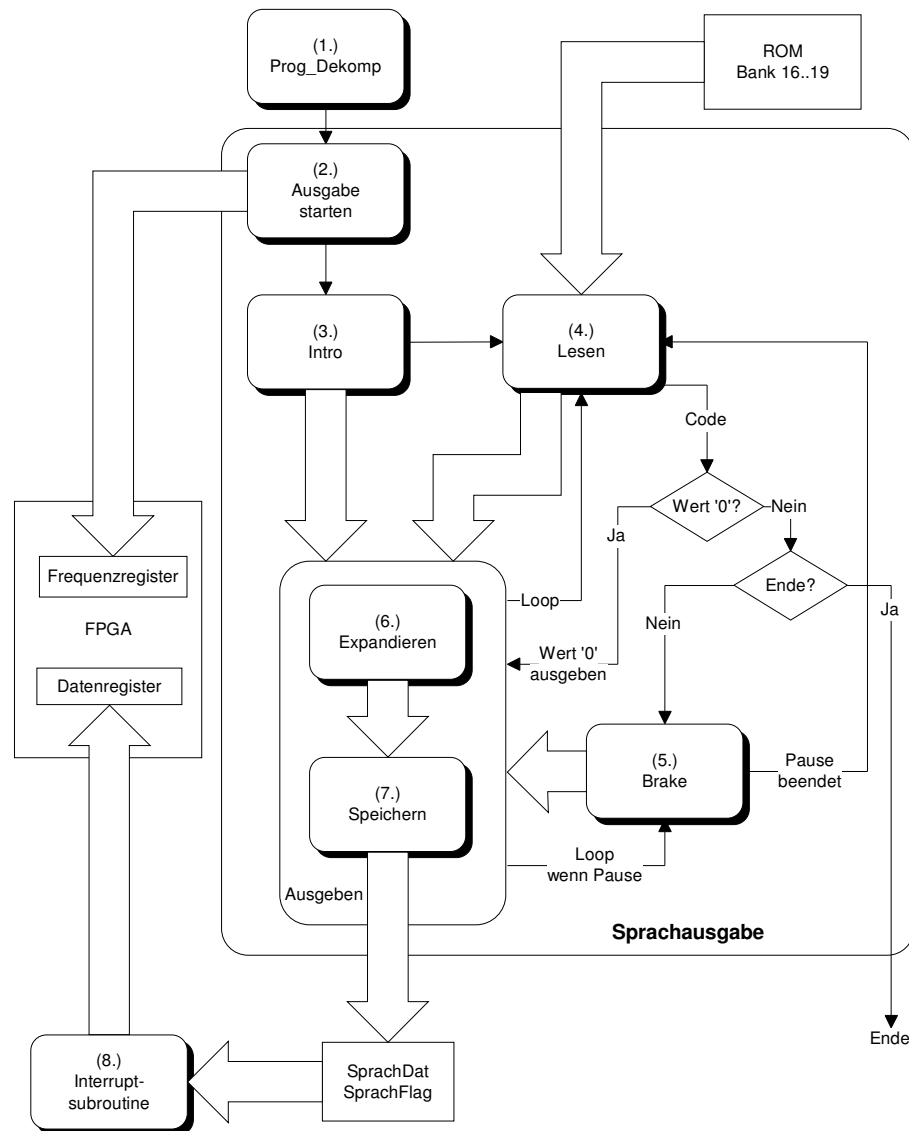
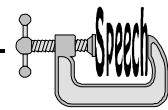


ABBILDUNG 28 ABLAUF VON LESEN UND DEKOMPRIEREN EINES SPRACHSAMPLES

Ablauf:

- (1.) Will man den geladenen Sample anhören, so kann die Ausgabe aus dem Benutzermenü gestartet werden. Als erstes wird der Servicetext ausgegeben, welcher als String am Anfang des Sprachsamples abgespeichert wurde. Das Ende des Strings ist mit dem Nullstring markiert. Das folgende Byte nach dem String ist das Register (siehe Kapitel 3.3.13). Dieses wird in die Variable 'Register' gespeichert, von dort man jederzeit Zugriff hat. Anhand dessen weiss man, mit welcher Abtastfrequenz und Bittiefe der Sample digitalisiert, und ob der Sample mit  $\mu$ Law quantisiert worden ist.



Je nachdem, wird für die Sprachausgabe folgende Funktion gestartet:

Bittiefe	$\mu$ -Law	Funktionsname:
4	Ja	Dekomp_4
5	Ja	Dekomp_5
6	Ja	Dekomp_6
4	Nein	Ausgabe_4
5	Nein	Ausgabe_5
6	Nein	Ausgabe_6

TABELLE 2 FUNKTIONEN FÜR DIE SPRACHAUSGABE

Es wurde für jede Art von Sprachausgabe eine eigene Funktion geschrieben, damit nicht ständig das Register abgefragt werden muss. Somit spart man Rechenzeit ein.

Die folgenden Aktionen werden alle aus der Funktion (siehe Tabelle 2) für die Sprachausgabe ausgeführt:

- (2.) Bevor die Ausgabe gestartet werden kann, muss das Frequenzregister im FPGA auf die richtige Abtastfrequenz gesetzt werden. Danach wird der Interrupt PA2 im Prozessor freigegeben, und das 'Intro' ausgeführt
- (3.) Das Intro ist dazu da, um zuerst 1000 Nullwerte auszugeben. Das ist nötig, weil sonst der Anfang der Sprachdaten verschluckt wird.
- (4.) Jetzt kann mit dem Lesen der Sprachsamples begonnen werden. Dieser Teil liest jeweils den nächsten Wert aus den ROM-Banken 16..19. Wird in einem Wert einen Code detektiert, das heisst den Wert '0' (siehe Kapitel 5.4.2.2), so wird überprüft, ob es a) wirklich um den Wert '0' handelt, b) das Ende der Sprachdaten erreicht wurde, oder c) es sich um eine Sprachpause handelt.
  - a) Der Wert '0' wird ausgegeben.
  - b) Die Sprachausgabe ist beendet und wird abgebrochen. Das heisst, es wird gewartet, bis der letzte Sample ins FPGA geladen wurde. Dann wird der Interrupt 'PA2' deaktiviert und zurück ins Benutzermenü gegangen.
  - c) Falls es sich um eine Sprachpause handelt wird die Pause eingefügt (siehe Punkt 5.)

Die Sprachsampledaten sind komprimiert worden. Sie wurden Bitweise aneinander gefügt (siehe Kapitel 5.4.4). Das heisst sie müssen beim Lesen in das 8-Bit Datenformat gebracht werden (siehe Abbildung 29).

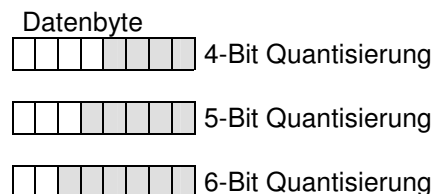


ABBILDUNG 29 8-BIT DATENFORMAT FÜR VERSCHIEDENE BITTIEFEN

Vom ROM kann man aber nur 1 oder 2 Byte auf einmal lesen. Je nachdem müssen die Samples geschoben und maskiert werden, um sie in die richtige Lage zu bringen. Unter Umständen muss ein gleiches Byte im ROM für den nächsten Wert noch einmal geladen werden. Folgend sind die verschiedenen Bittiefen dargestellt:

Das Lesen von Samples, die mit 4 Bit quantisiert sind:

Hier ist das höhere Nibble der 1. Wert und das niedere Nibble jeweils der 2. Wert. Beim Lesen wird immer ein ganzes Byte geholt. Mit Hilfe eines 1 Bit Pointers wird überprüft, welcher Wert gerade gelesen werden muss. (höheres oder niederes Nibble) Wird der 1. Wert gelesen wird dieser um 4 Bit logisch nach rechts verschoben, und beim 2. Wert wird dieser maskiert. Jedes Byte muss also immer zwei Mal gelesen werden.

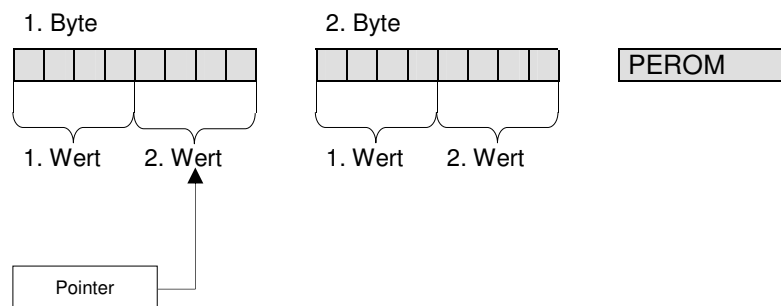


ABBILDUNG 30 LESEN VON SAMPLES BEI 4-BIT QUANTISIERUNG AUS DEM PEROM

Das Lesen von Samples, die mit 5 Bit quantisiert sind:

Diese Variante ist die Aufwendigste, um die Sprachsamples vom ROM zu lesen. Die 5-Bit Sprachsamples können in acht verschiedenen Positionen im Daten-ROM liegen. Das heißt, man muss mit einem 3-Bit Pointer unterscheiden, welcher Wert gerade gelesen werden muss. Die Werte können auch byteübergreifend liegen. Je nachdem wird ein Byte oder ein Doublebyte gelesen. Der gewünschte Sprachsample wird mit logisch Schieben und Maskieren in die richtige Lage gebracht.

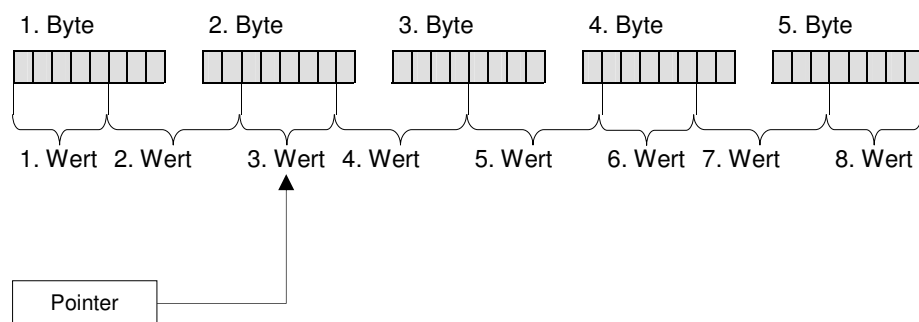


ABBILDUNG 31 LESEN VON SAMPLES BEI 5-BIT QUANTISIERUNG AUS DEM PEROM

Das Lesen von Samples, die mit 6 Bit quantisiert sind:

Bei dieser Variante muss man vier verschiedene Fälle unterscheiden, um die Sprachsamples vom ROM zu lesen. Diese werden mit einem 2-Bit Pointer unterschieden, welcher Wert gerade gelesen werden muss. Auch hier muss je nachdem ein Byte oder aber ein Doublebyte gelesen werden. Der gewünschte Sample muss dann mit logisch Schieben und Maskieren in die richtige Lage gebracht werden.

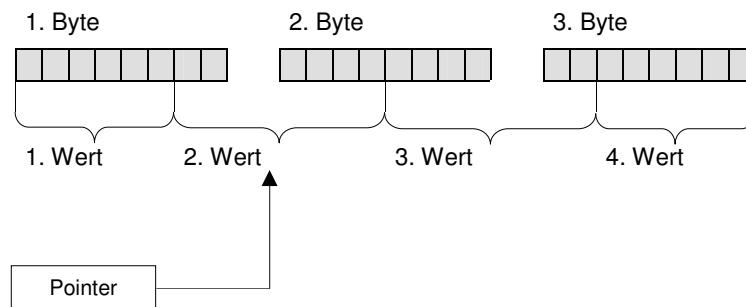


ABBILDUNG 32 LESEN VON SAMPLES BEI 6-BIT QUANTISIERUNG AUS DEM PEROM

- (5.) Ist eine Pause detektiert worden, so wird diese mit Mittelwerten aufgefüllt. Zuerst holt die Routine die Pausenlänge aus dem Daten-ROM und rechnet die nötigen Anzahl Werte aus, die der Pausenlänge entspricht. (Pausenberechnung siehe Kapitel 5.4.2.2). Danach wird die entsprechende Anzahl Mittelwerte dem 'Ausgeben' geschickt.
- (6.) Der gelesene Wert muss jetzt in das richtige Format (siehe Kapitel 4.3.3.1) gebracht werden, bevor er ausgegeben werden kann. Die Sampledaten sind im ROM als unsigned Integer geladen worden. Der DA-Wandler benötigt aber signed Integer  
Je nachdem, ob die Sprachsample mit  $\mu$ Law komprimiert worden sind, werden sie dabei noch expandiert.  
Dazu dient eine Tabelle, die schon mit den richtigen Werten aufgefüllt worden ist. Anhand vom gelesenen Wert kann dann aus der Tabelle der richtige 8-Bit Wert herausgelesen werden.
- (7.) Der Sample kann jetzt in der Variable 'SprachDat' gespeichert werden. An dieser Stelle wird der ganze Vorgang Sprachausgabe synchronisiert. Zusätzlich zum Wert, wird ein Flag (Variable: SprachFlag) gesetzt. Dieses wird erst wieder zurückgesetzt, wenn der Wert in das Datenregister vom FPGA geschrieben wurde. (siehe Punkt 8.) Der Mikrokontroller wartet dann jeweils an dieser Stelle, bis dieses Flag zurückgesetzt wurde, bevor er den nächsten Wert in die Variable 'SprachDat' schreibt. Danach beginnt er wieder bei Punkt 4: Lesen.
- (8.) Die Interruptsubroutine schreibt immer bei einer Interruptaufforderung vom FPGA an PA2 den Wert im SprachDat ins Datenregister vom FPGA. Danach wird das SprachFlag zurückgesetzt.

### 6.3. Implementation FPGA

Durch die Anforderungen des Systems, vier verschiedene Samplingfrequenzen zwischen 6 und 8 kHz zu betreiben, und die Tatsache, das auf dem SPACE alle anderen Frequenzen von 20.016 MHz heruntergeteilt werden müssen, entsteht das Digitaldesign (Schema s. Anhang G).

Der DAC wird im 'External SCLK' Modus betrieben, da die Taktfrequenzen vom FPGA sehr genau erzeugt werden, und so die Gefahr von Clock Jitter gering ist. De-emphasis kann im 'External SCLK' Modus nicht verwendet werden. Da wir aber keine Signale haben, die mit Pre-emphasis aufgenommen wurden, spielt das keine Rolle.

Der Chipselect für die zwei Register ist mit dem CS\_RES und dem LSB des Daten-Adress-Bus realisiert. Der CS\_RES ist schon in der Grundversion des Space FPGA Digitaldesigns vorhanden. Dieser Chipselect ist immer aktiv, wenn der Daten-Adress-Bus einen Wert zwischen \$0080 und \$009F annimmt. Durch die Verknüpfung mit dem LSB des Daten-Adress-Bus wird immer bei den geraden Adressen zwischen \$0080 und \$009F in das Datenregister, bei den ungeraden ins Frequenzwahlregister geschrieben.

Die Frequenzteiler wurden so ausgewählt, dass die Frequenz für den seriellen Clock 64 mal höher als die Samplingfrequenz ist. Damit könnten pro Halbperiode des LRCK-Signals 32 Bits serielle Daten zum DA Wandler übertragen werden. In unserer Anwendung werden pro Sample 8 Bit serielle Daten übertragen. Damit ist die serielle Datenübertragung schon nach einem Viertel der LRCK-Periode beendet (Siehe Figure 6 vom Datenblatt des DAC im Anhang). Nach der Übertragung der 8 Bit werden vom Schieberegister nur noch Nullen gesendet, da der Schiebemodus des Schieberegisters auf 'arithmetisch' eingestellt ist.

Das Schieberegister sendet jeweils das MSB der Daten zuerst.

Da wir uns mit einem Kanal begnügen, werden nur nach der positiven Flanke von LRCK parallele Daten geladen. Das heisst, es wird nur der rechte Kanal verwendet.

Die Zeit-Simulation zeigt das Timing der Signale:

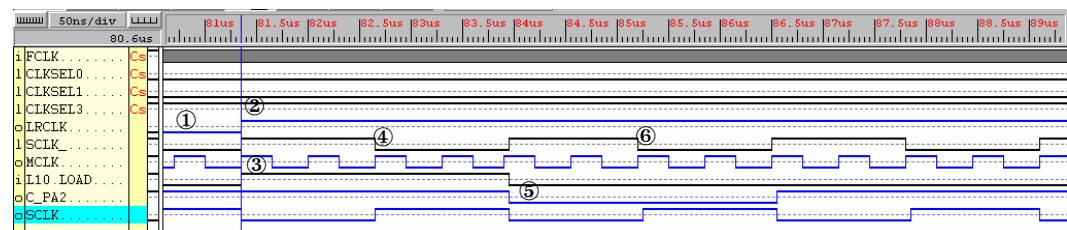
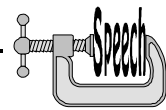


ABBILDUNG 33 ZEITSIMULATION

Die Signale CLKSEL0 und CLKSEL1 sind auf logisch 0 gesetzt. CLKSEL3 dient als Enable für die Frequenzen. Der Frequenzwahl-Multiplexer lässt somit das durch 13 geteilte Signal FCLK passieren, welches als MCLK an einen Pin herausgeführt wird.

Der Ablauf des Herausschreibens ist folgendermassen gegliedert:

- ① LRCK ist auf logisch 0. Das Schieberegister schreibt lauter Nullen.
- ② Positive Flanke von LRCK. Jetzt wartet der Puffer des rechten Kanals im DAC auf Daten.
- ③ LOAD geht auf logisch 1. Die parallelen Daten des Datenregisters werden mit...
- ④ ...der negativen Flanke von SCLK\_ ins Schieberegister geladen.
- ⑤ C\_PA2 wird ausgelöst. Dieses Signal wird als Interrupt für den HC11 gebraucht. Der Mikrokontroller hat jetzt eine Taktperiode Zeit, um das Datenregister mit neuen Daten zu füllen.
- ⑥ Negative Flanke von SCLK\_. Der erste Wert der geladenen Daten (das MSB) wird auf den DAC geladen.



## 6.4. Implementation Print

Beim Layouten kann man bei einfachen Prints entweder direkt das Layout erstellen, oder zuerst ein Schema zeichnen, dann eine Netzliste erstellen, und aufgrund dieser Netzliste das Layout erstellen. Da man bei der direkten Methode schwer die Kontrolle behalten kann, ob das Layout auch wirklich dem gezeichneten Schema entspricht, haben wir uns für den zwar aufwendigen, dafür aber sicheren Weg entschieden. Probleme gab es beim 96 Pol Stecker. Man könnte meinen, das sei ein Standardelement. Das Schemasymbol konnte aber in den Libraries nicht gefunden werden und es musste manuell ein Symbol erstellt werden. Erstaunlicherweise war beim Layouten der Footprint für diesen Stecker aber vorhanden.

Probleme gab es auch bei der Platzierung des Boomer IC's. Da dieser IC nur im SMD SOP-16 Gehäuse erhältlich ist, gibt es ein gemischtes Layout aus "normalen" Elementen und SMD Bausteinen. Als die erste Layout-Version fertig war, wurde bemerkt, dass der SMD-Chip spiegelverkehrt plazierte war. Das Layout musste dann komplett neu erstellt werden!

## 7. Auswertung der Resultate

### 7.1. Einleitung ins Kapitel

In diesem Kapitel werden die erzielten Ergebnisse gezeigt. Im besonderen wird gemessen, wie stark der Prozessor ausgelastet ist, um die Sprachsamples zu dekomprimieren und auszugeben.

### 7.2. Auslastung des Prozessors

#### 7.2.1. Messmethode

Die Messung wird mit Hilfe des Ports A am Portbaustein vorgenommen. Bevor der Prozessor den neu berechneten Wert ausgeben kann, muss er warten, bis der alte Wert geholt worden ist (Synchronisation der Ausgabe). Diese Wartezeit wird angezeigt, indem der Port A währenddessen auf 'High' gesetzt wird.

Folgende Abbildung zeigt einen Ausschnitt vom Makro 'Ausgeben von einem Wert' aus der Testsoftware, wo der Port A angesteuert wird. (Die Portansteuerung ist in der Endversion der Software wieder entfernt worden)

```
      :
      :
      PSHA
      LDAA    #$ff
      JSR     Port_PAWrByte
      PULA
* --- *
*      Waitloop bis DA-Wandler vorhergehenden      *
*      Wert geholt hat                               *
*
Loop   LDAB    SprachFlag
      CMPB    #_FlagOff
      BNE     Loop           Waitloop
* --- *

      PSHA           Fuer Messung der Auslastung
      LDAA    #$00           der CPU
      JSR     Port_PAWrByte
      PULA
      :
      :
```

ABBILDUNG 34 ANSTEUERUNG DES PORTS A ZUR MESSUNG DER AUSLASTUNG DES PROZESSORS

Die Rechenzeit kann jetzt mit einem KO gemessen werden. Am besten geht es, wenn zusätzlich zum Port A noch die Interruptaufforderung an PA2 mitgemessen wird. Das KO-Bild kann auf diesen Kanal gut getriggert werden. Folgende Abbildung zeigt die Messung eines Sprachsamples.

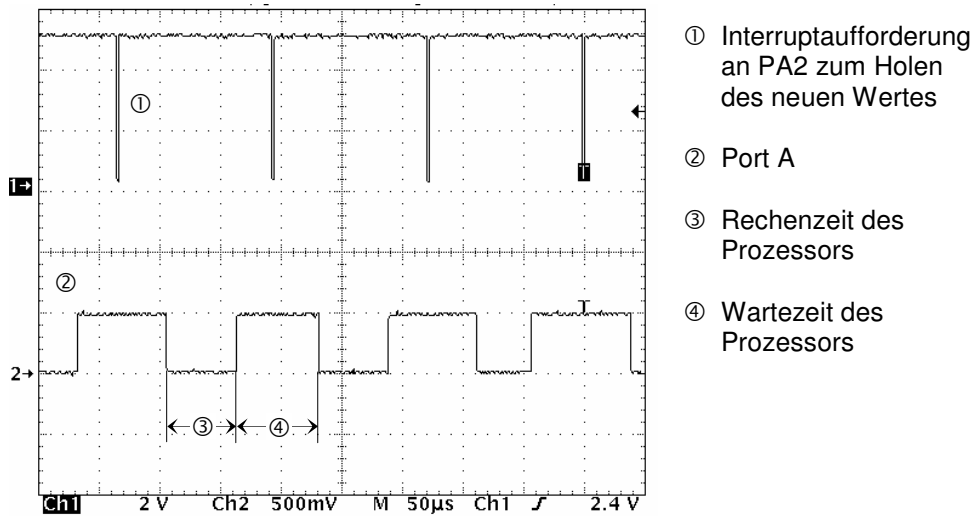


ABBILDUNG 35 MESSUNG DER AUSLASTUNG DES PROZESSORS AM KO

Das Setzen der Ports braucht natürlich auch Rechenzeit. Das verfälscht das eigentliche Ergebnis. Diese zusätzliche Rechenzeit ist mit einem Testprogramm gemessen worden. Der gemessene Wert kann man von der Rechenzeit abziehen, die man oben erhalten hat.

Der zusätzliche Code sieht folgendermassen aus:

```
PSHA
LDAA    #$ff
JSR     Port_PAWrByte
PULA
```

und

```
PSHA
LDAA    #$00
JSR     Port_PAWrByte
PULA
```

Die Rechenzeit dafür beträgt je **13µs**.

Das heisst, von der gemessenen Rechenzeit kann man diese 13µs abziehen.

### 7.2.2. Messung der Rechenzeit und Prozessorauslastung

Die Messung wurde mit verschiedenen Bittiefen und Samplingraten durchgeführt. Auf Messungen mit Samples, die ohne  $\mu$ Law komprimiert sind, wurde verzichtet.

Bittiefe	Samplingfrequenz [kHz]	Minimale Rechenzeit [ $\mu$ s]	Mittlere Rechenzeit [ $\mu$ s]	Maximale Rechenzeit [ $\mu$ s]	Maximale mögliche Rechenzeit [ $\mu$ s]	Prozessorauslastung [%]
4	6.014	32		51	166.3	19 – 31
4	6.516	32		51	153.5	21 – 33
4	7.108	32		51	140.7	23 – 36
4	7.819	32		51	127.9	25 – 40
5	6.014	32		87	166.3	19 – 52
5	6.516	32		87	153.5	21 – 57
5	7.108	32		87	140.7	23 – 62
5	7.819	32		87	127.9	25 – 68
6	6.014	32		65	166.3	19 – 39
6	6.516	32		65	153.5	21 – 42
6	7.108	32		65	140.7	23 – 46
6	7.819	32		65	127.9	25 – 50

TABELLE 3 MESSUNG DER PROZESSORAUFLASTUNG FÜR SAMPLES MIT  $\mu$ LAW-KOMPRIMIERUNG

Zur Kontrolle der Messungen, wurde beim Sample, der mit einer Samplingrate von 6.014kHz und einer Bittiefe von 5 Bit komprimiert wurde, die benötigte Rechenzeit ausgerechnet. Es sind die minimal und maximal benötigten Taktzyklen gezählt worden.

Bittiefe	Samplingfrequenz [kHz]	Minimale Anzahl Taktzyklen	Minimale gerechnete Rechenzeit [ $\mu$ s]	Maximale Anzahl Taktzyklen	Maximale gerechnete Rechenzeit [ $\mu$ s]
5	-	69	34.5	150	77.5

TABELLE 4 BERECHNUNG DER RECHENZEIT FÜR EINEN SPRACHSAMPLE

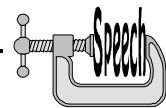
Die Rechenzeit ist mit folgender Formel berechnet worden:

$$\text{Rechenzeit} = \frac{\text{Taktzyklen}}{\text{Taktrate des Mikrokontrollers}}$$

### 7.2.3. Auswertung der Messresultate

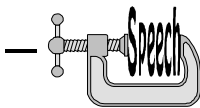
Jeder Sample hat praktisch die gleiche minimale Rechenzeit. Aber bei den maximalen gibt es zum Teil grosse Unterschiede. Das kommt daher, weil alle Methoden für den optimalen Fall zum Lesen von Samples aus der ROM-Bank etwa gleich viele Taktzyklen brauchen. Aber im schlechtesten Fall zum Teil beträchtlich unterschiedliche Anzahl benötigte Taktzyklen aufweisen. Es kommt darauf an, welcher Samples gerade geladen werden muss. Je nachdem muss er noch geschoben und maskiert werden. Am längsten geht es, wenn die ROM-Bank gewechselt werden muss.

Die Samplingfrequenz hat nur auf die Prozessorauslastung einen Einfluss. Man kann sagen, je höher die Samplingfrequenz, umso weniger Zeit hat der Prozessor zu Verfügung, um die Samplewerte zu dekomprimieren



### **7.3. Akustische Bewertung**

Die Ausgabe eines komprimierten Sprachsamples vom PC und vom SPACE-Board ist nicht zu unterscheiden. Das heisst, Simulation und Realität unterscheidet sich kaum. Es wurde deshalb darauf verzichtet, aufwendige Tests wie in der Semesterarbeit [2] durchzuführen.



## 8. Bedienungsanleitung

### 8.1. Installation

Hardware:

Verbinden Sie ...

... das SPACE Entwicklungsboard mit einem seriellen D-SUB9 Kabel mit der Com 1 Schnittstelle ihres Computers.

... die Bananenstecker mit einer geregelten +5V Gleichspannungsquelle.

... den Cinch-Stecker mit der Lautsprecherbox.

Beim Einschalten der Spannungsquelle leuchtet nun die rote Power-On-LED und die grüne Bereit-LED.

Software:

- Versichern sie sich, dass auf ihrem Computer MATLAB Version 5.2 installiert ist.
- Kopieren sie den ganzen Ordner 'MatlabGui' in ein Verzeichnis auf ihrer Festplatte. Notieren sie den Pfad, auf dem sich der Ordner 'MatlabGui' nun befindet.
- Kopieren sie die Datei 'startup.m' aus dem Ordner 'MatlabGui' in den Ordner 'C:\.....\Matlab52\bin'.
- Öffnen sie diese kopierte Datei durch Doppelklicken und fügen sie den notierten Pfad am markierten Ort ein. Datei Speichern und Schliessen.
- Kopieren sie die Datei space.ht in das Verzeichnis 'C:\Programme\Zubehör\Hyperterminal'.
- Starten sie Matlab.
- "The Magic Voice Squeezer " wird automatisch gestartet.

## 8.2. Bedienung des Magic Voice Squeezers

Die Bedienelemente des Voice Squeezers sind bei einem Komprimiervorgang tendenziell in der Reihenfolge von oben nach unten zu betätigen. Die Funktion der Bedienelemente ist im folgenden beschrieben.

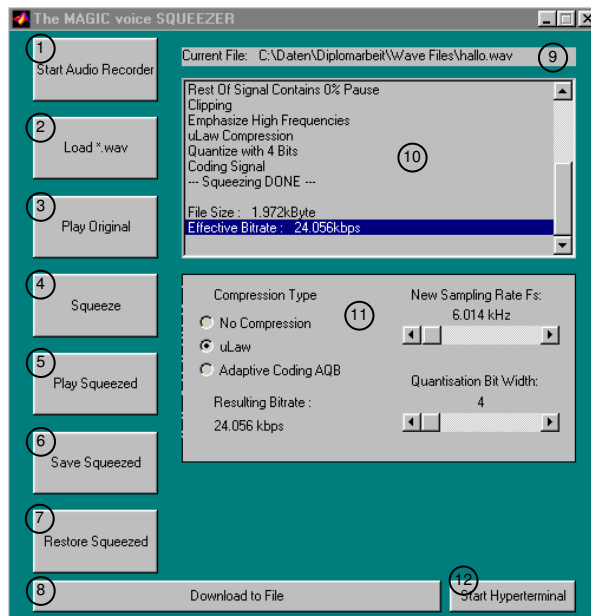


ABBILDUNG 36 THE MAGIC VOICE SQUEEZER

- ① Startet den Audiorecorder von Windows 95. Hier kann man mit dem PC-Mikrofon einen Satz aufsprechen und als Wave File speichern.
- ② Laden eines Wave Files.
- ③ Probehören des Originalsatzes.
- ④ Komprimieren des originalen Satzes mit den Einstellungen in ⑪.
- ⑤ Probehören des komprimierten Satzes. Zusammen mit ③ kann die Qualität des komprimierten Signals bestimmt werden.
- ⑥ Speichern des komprimierten Satzes.
- ⑦ Ein mit ⑥ gespeicherter Satz kann hier wiederhergestellt werden. Die Einstellungen in ⑪ werden automatisch aktualisiert.
- ⑧ Der komprimierte Satz wird im richtigen Download Format in eine Datei geschrieben. Der Benutzer kann eine Information in Textform beifügen.
- ⑨ Hier wird immer der Dateiname der aktuellen Datei angezeigt. Wenn ein Wave File geladen wird, ist die aktuelle Datei eine Wave Datei. Wenn ein komprimierter Satz mit ⑦ wiederhergestellt wird, hat die aktuelle Datei die Endung '\*.mat'.
- ⑩ Die Minikonsole zeigt wichtige Informationen über die ausgeführten Aktionen an. Hier können die Arbeitsschritte auch nachvollzogen werden.
- ⑪ Hier können die Einstellungen für die Komprimierung mit ④ gesetzt werden. Zusätzlich wird die theoretische Datenrate angezeigt. Diese Datenrate wird bei der Komprimierung durch die Pausendetektion in den meisten Fällen unterschritten.
- ⑫ Startet das Hyperterminal von Windows 95 mit den richtigen Einstellungen. Ein Dialog zeigt an, dass man nach Start des Hyperterminals die Taste 'C' drücken muss und die angezeigte Datei über die serielle Schnittstelle ausgeben muss. Dies erfolgt durch den Befehl 'Textdatei Senden' im Menü 'Übertragung'.

### 8.3. Bedienung der SPACE Hardware

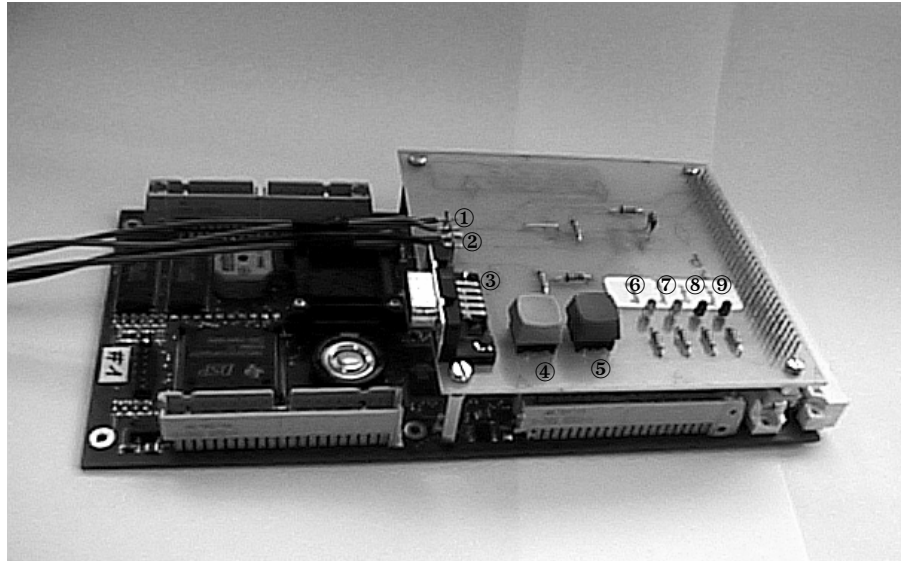
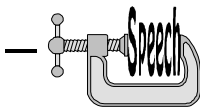


ABBILDUNG 37 DIE SPACE HARDWARE

- ① Anschluss für Lautsprecherbox
- ② Anschluss für Gleichspannungsquelle +5V.
- ③ 9 Poliger Anschluss für die serielle Datenübertragung.
- ④ Grüner Knopf löst Ausgabe des gespeicherten Satzes aus.
- ⑤ Reserveknopf (Wurde bei der Entwicklung verwendet)
- ⑥ Grüne Bereit-LED. SPACE Hardware ist bereit für die Sprachausgabe mit ④.
- ⑦ Grüne Download-LED. SPACE Hardware ist bereit für den Download. Blinkt während dem Download.
- ⑧ Rote Error-LED. Leuchtet, wenn beim Download ein Fehler auftritt.
- ⑨ Rote Power-LED. Leuchtet, wenn Spannungsversorgung eingeschaltet ist.

Die SPACE Hardware kann auch umfassend mit dem Hyperterminal gesteuert werden. Die Bedienung ist durch ein einfach verständliches Menü gewährleistet.





## 9. Schlussfolgerung

### 9.1. Rückblick

Die Diplomarbeit war eine Weiterführung der Semesterarbeit SS-98-06. Das hatte den Vorteil, dass wir uns nicht in ein neues, fremdes Gebiet einarbeiten mussten. Die Planung war dadurch etwas einfacher. Wir wussten schon in etwa, was uns erwarten wird. Eine Knacknuss waren die neuen Tools, in die wir uns einarbeiteten. Dies war zwar für die Semesterarbeit geplant gewesen. Dafür hat aber die Zeit nicht mehr gereicht.

Die Implementation verlief ohne grössere Probleme. Schritt für Schritt konnten wir unsere Hardware in Betrieb nehmen. Das einzige, was wir nicht implementiert haben, ist die Speicherorganisation (advanced). Das heisst, Verwaltung von mehreren Samples auf dem Mikrokontroller.

Das Ergebnis war für uns sehr erfreulich. Wir erreichten alle Punkte, die wir am Anfang geplant hatten.

### 9.2. Schlusswort

Durch diese Diplomarbeit haben wir einen Einblick in verschiedene Gebiete der Software- und Hardwareentwicklung erhalten. Wir lernten einige nützliche Tools kennen und anwenden. Durch die kompetente Betreuung von Herrn A. Ehrensperger und Herrn V. Altwegg von der ASCOM System AG, verlief die Einarbeitung in die für uns zum Teil neue Software ohne grössere Probleme. Wir danken unseren Studienkollegen Michael Brenneis für die Unterstützung bei der Implementation des GUI, Herrn P. Roffler für seine Hilfsbereitschaft und den sehr guten Support in allen Bereichen. Wir danken auch Prof. Dr. A. Schüeli für seine zuverlässige Organisation und seine Hilfsbereitschaft.

Rapperswil, 13. September 1998

Reto Lienhardt

Steven Kaufmann