Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
*Hamburg University of Applied Sciences*

# Masterarbeit

Florian Hoffmann

## Portierung und Post-Processing eines Algoritmus zur Lokalisierung von Audioquellen auf leistungsfähiger Hardware

*Fachhochschule Westküste*
*Fachbereich Technik*

*Hochschule für Angewandte Wissenschaften Hamburg*
*Fakultät Technik und Informatik*
*Department Informations- und Elektrotechnik*

*Fachhochschule Westküste - University of Applied Sciences*
*Faculty of Engineering*

*Hamburg University of Applied Sciences*
*Faculty of Engineering and Computer Science*
*Department of Information and Electrical Engineering*

Florian Hoffmann

# Portierung und Post-Processing eines Algoritmus zur Lokalisierung von Audioquellen auf leistungsfähiger Hardware

Masterarbeit eingereicht im Rahmen der Masterprüfung
im gemeinsamen Masterstudiengang Mikroelektronische Systeme
am Fachbereich Technik
der Fachhochschule Westküste
und
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Ing. Hans Peter Kölzer
Zweitgutachter : Prof. Dr. Hans Dieter Schütte

Abgegeben am 13. März 2018

**Florian Hoffmann**

## Thema der Masterarbeit

Portierung und Post-Processing eines Algoritmus zur Lokalisierung von Audioquellen auf leistungsfähiger Hardware


## Stichworte

D.Module2.C6657, Enhanced Direct Memory Access, Multi Channel Buffered Serial Port, Interprozessor Interrupt, PCM3003, Rekursive Regression,DSPF_sp_fftSPxSP, DSPF_sp_mat_mul_cplx



## Kurzzusammenfassung

In dieser Arbeit wird erklärt welche Arbeitsschritte nötig waren um ein bestehendes, auf dem auf dem UCA-ESPRIT Algorithmus basierendes, Programm von der D.Module.C6713 Hardware auf die D.Module2.C6657 Hardware zu portieren und was geändert werden musste um das Lokalisieren von zwei, dynamischen Schallquellen gleichzeitig zu ermöglichen


**Florian Hoffmann**

## Title of the paper

Porting and post-processing of a sound localization algorithm on powerful hardware


## Keywords

D.Module2.C6657, Enhanced Direct Memory Access, Multi Channel Buffered Serial Port, Interprocessor Interrupt, PCM3003, Recursive Regression,DSPF_sp_fftSPxSP, DSPF_sp_mat_mul_cplx



## Abstract

This thesis describes which steps had to be taken to port an existing, UCA-ESPRIT algorithm based, software from the D.Module.C6713 hardware to the D.Module2.C6657 hardware and which changes had to be applied to make the localization of two, dynamic sound sources possible

# Inhaltsverzeichnis

# Abbildungsverzeichnis

# Abkürzungs- und Begriffserklärung

| | |
|---|---|
| TI : | Texas Instruments |
| PC : | Personal Computer. Der Labor Computer auf dem die für die Arbeit benötigten Software Pakete installiert sind |
| DSP : | digitaler Signal Prozessor |
| RAM : | Random Access Memory. Der Arbeitsspeicher der Hardware |
| DDR3 : | Double Data Rate Synchronous Dynamic Random Access Memory 3. Eine RAM-Speicherchip Technologie mit 100-266 MHz interner Datenrate und der Möglichkeit 2 Datenpakete pro Takt zu übertragen. |
| SDRAM : | Synchronous Dynamic Random Acces Memory. Eine ältere Version der DDR Technolgie mit niedrigeren Taktungen und Datenübertragunsraten |
| CPU : | Central Processing Unit. Die zentrale Recheneinheit der Hardware die alle soft- und hardwareseitigen Anfragen verwaltet und bearbeitet. |
| L2 Speicher : | Level2 Cache Speicher. Ein Teil des Cache Speichers der die Daten des RAM zwischenspeichert um sie der CPU schnell zur Verfügung stellen zu können |
| Cache : | Bufferspeicher zwischen RAM und CPU. Hier werden die nächsten von der CPU zu behandelnden Daten in Paketen vorgehalten um nicht bei jedem Datenzugriff auf den langsameren, aber größeren, RAM Speicher warten zu müssen. |
| REVT : | Receiver Ready Event. Signalisiert der CPU oder dem EDMA Controller, dass neue Daten an den McBsp zur Abholung bereit liegen |

| | |
|---|---|
| XEVT : | Transmitter Ready Event. Signalisiert der CPU oder dem EDMA Controller, dass ein Datentransfer an die McBsp abgeschlossen wurde |
| IPR/IPRH : | Interrupt Pending Register bzw. Interrupt Pending Register High. Die Register des EDMA Controllers in denen eingetragen wird ob ein Interrupt Event aufgetreten ist. |
| SYNCDIM : | Transfer synchronization dimension. |
| NOP : | No Operation. Ein Assembler Befehl, der keine Operation ausführt |
| Com-Port : | serielle Schnittstelle des PC. |
| GUI : | Graphical User Interface. Eine grafische Benutzeroberfläche |
| USB : | Universal Serial Bus. Eine vielfach verwendete, serielle, bi-direktionale Datenübertragungsschnittstelle für multiple Anwendungen |
| ANSI : | American National Standard Institute. |
| ANSI C : | Eine textbasierte Programmierhochsprache in der der Code dieser Arbeit verfasst wurde. |
| DFFT;FFT : | ( diskrete ) Fast Fourier Transformation. Eine schnelle Implementierung der diskreten Fourier Transformation |
| JTAG : | Join Test Action Group. Eine Schnittstelle zum programmieren, testen und debuggen von Integrierten Schaltkreisen innerhalb ihrer Einsatzumgebung. |
| GPIO : | General Purpose Input Output. Frei Programmierbare Pins der Hardware die für multiple Zwecke genutzt werden können |
| EMIF : | External Memory Interface. Bus zum Anbinden von externem Speicher an die Hardware |
| D-Sub : | Steckerstandard für die serielle Schnittstelle des PC |
| RJ : | Steckerstandard für den Aufbau von lokalen Netzwerken |
| Little Endian : | bezeichnet die Ordnung in der Werte im Speicher abgelegt oder übertragen werden. Little Endian bedeutet, dass das am wenigsten signifikante Bit als erstes übertragen wird. |

| | |
|---|---|
| BaudRate : | Anzahl der per UART Übertragenen Symbole pro Sekunde. |
| DataBits : | Anzahl Bits die nacheinander über UART übertragen werden und zum gleichen Wert gehören. |
| StopBits : | definiert ob 1 oder zwei Stopbits am Ende jedes per UART übertragenen Wertes eingefügt werden sollen und dient der Synchronisierung zweier stark asynchron-arbeitender Busteilnehmer. |
| Parity : | Kontrollbit der UART Kommunikation, dass als Kontrollmöglichkeit der Kommunikation übertragen werden kann. Der Empfänger kann ebenfalls die Parität der Empfangenen Daten berechnen und sie mit der Empfangenen Parität vergleichen. |
| Handshake : | Kann zur Hardwaremässigen Synchronisierung von Sender und Empfänger bei der UART Kommunikation benutzt werden. Die Synchronisierung geschieht dann über zwei spezielle Datenleitungen zwischen den Busteilnehmern. |
| char : | ein 8 Bit breiter Datentyp. |
| string : | ein Array aus char. |
| handle : | die gespeicherte Adresse bestimmter Hardwarefunktionen zur Verwendung durch die CPU. |
| float : | ein 32 Bit breiter Datentyp für das Abspeichern von Gleitkommazahlen. |
| int : | eine 32 Bit breiter Datentyp für das Abspeichern von ganzzahligen Werten |
| short : | ein 16 Bit breiter Datentyp für das Abspeichern ganzzahliger Werte. |
| Array : | eine Matrix aus einem definierten Datentyp deren Werte alle hintereinander im Speicher abgelegt werden. Der Name des Array kann zum Abruf der Startadresse dieser Werte verwendet werden. |
| UCA-ESPRIT : | Uniform Circular Array - Estimation of Signal Parameters by Rotational Invariance Techniques. Der Algorithmus auf dem die lokalisierung der Audioquellen beruht. |

| | |
|---|---|
| Loop : | Eine Softwareschleife zur Abarbeitung wiederkehrender Aufgaben. |
| Look Up Table : | Eine fest im Speicher abgelegte Tabelle mit Vergleichswerten. |
| RMS : | Root Mean Square. Beschreibt den Energiegehalt eines Signals. Ergibt sich aus der Wurzel des Integrals über die quadrierte Funktion. |
| Peak : | ein Maximum innerhalb eines Array |
| MatLab : | Ein Software Tool für den Laborbetrieb |
| Histogrammarray : | Das Array in dem die während des aktuellen Messdurchgangs gefundenen Winkel ( und später auch die Frequenzen die zur Berechnung der Winkel gef"ührt haben ) abgelegt werden. |
| Winkelarray : | Das Array in dem die Auftrittshäufigkeit der aktuellen Winkel im Histogrammarray vor der Histogramanalyse abgespeichert wird |
| Clock Source : | Die Quelle der Taktung für bestimmte Prozesse. Zum Beispiel kann die Quelle für die Taktung der McBsp extern oder vom Chip selbst bezogen werden. |
| Clock Divider : | Skalierungsfaktor für die Taktung eines Prozesses. Mit seiner Hilfe können von einer Quelle fester Taktung auch Taktungen niedrigerer Frequenz bezogen werden. |
| Frame | Eine definierte Anzahl Taktzyklen in der eine definierte Anzahl Datenworte definierter Länge übertragen werden. |
| Falling Edge : | fallende Flanke. |
| Rising Edge : | steigende Flanke. |
| Active High : | Sensitivität auf das hohe Potential |
| Active Low : | Sensitivität auf das niedrige Potential. |
| Delay : | Verzögerung. |
| Audio Codec : | Die Vorgehensweise mit der ein Audio Signal abgetastet und unter Umständen komprimiert bzw. dekomprimiert wird. |

| | |
|---|---|
| Interrupt Event : | Ein Signal an die CPU, dass ein bestimmter Prozess abgearbeitet oder ein bestimmter Zustand erreicht wurde. |
| Interrupt Service Routine : | Ein Programmteil der nach dem Auftreten eines bestimmten Interrupt Events ausgeführt wird. |
| Header Datei : | Eine Hilfsdatei für die Programmierung in C. In dieser Datei werden alle, für ein bestimmtes Modul einer Software nötigen, globalen Variablen und Funktionen definiert. |

# 1 Einleitung

In dieser Arbeit wird beschrieben wie ein bereits für das D.Module.C6713 von DSignT entwickeltes Echtzeit-System zur Lokalisierung von Audioquellen auf die leistungsfähigere Hardware das D.Module2.C6657 von DSignT übertragen wird. Anschließend wird beschrieben wie durch ein geeignetes Post-Processing eine Erweiterung des Systems für bis zu zwei Schallquellen und das Lokalisieren von Sprechern realisiert werden kann.

Die Arbeit basiert auf der Vorgängerarbeit „Echtzeit-Lokalisierung von Audioquellen mittels zirkularen Mikrofonarrays und des UCA-ESPRIT Algorithmus“ die von Jens Reermann am 11.Oktober 2013 an der HAW Hamburg eingereicht wurde.

Im folgenden Kapitel wird kurz auf die Motivation und den Stand des Projektes eingegangen. Zusätzlich werden die Unterschiede zwischen den beiden Hardware Typen erläutert und ein kurzer Ausblick auf den Aufbau der Arbeit gegeben.

## 1.1 Motivation

Mit Fortschreitender Digitalisierung und Automatisierung des Alltags werden zuverlässige Sensoren der verschiedensten Art immer gefragter. Dies gilt natürlich auch für akustische Sensoren. Sind diese genau genug gibt es unzählige Anwendungsgebiete von Konferenz- und Eventtechnik, über Sicherheitssysteme, Augmented Reality und automatisiertes Fahren oder Fliegen.

Das vorliegende System dürfte dabei von besonderem Interesse sein, da es die Möglichkeit bietet mehrere Schallquellen in 3 Dimensionen zu triangulieren. Voraussetzung hierfür ist jedoch eine leistungsfähige Hardware. Mit der älteren, weniger leistungsfähigen Hardware, blieb keine Rechenzeit für ein Post Processing zwischen zwei EDMA Interrupts. Aus diesem Grund war es lediglich möglich Eine, und zwar die aktuell Lauteste, Schallquelle zu lokalisieren.

Die neuere, leistungsfähigere Hardware, soll es nun ermöglichen 2 Schallquellen zu klassifizieren und zu lokalisieren

## 1.2 Stand des Projektes

Zu Beginn der vorliegenden Arbeit lag ein funktionierendes System zur Echtzeit Lokalisierung von Schallquellen auf einem D.Module.C6713 von TI mit entsprechendem, zirkularem 8 Kanal Mikrofonarray und dazu gehörigen Mikrofonverstärkern vor.

Die weiteren Hardware Komponenten sind zum einen der D.Module.C6713, PCM3003 Audiocodec Platine und ein vorverkabelter 2xFlachbandkabel auf 16xKoaxial Kabel Adapter. (nähere Erläuterungen der eingesetzten Hardware Komponenten finden sich in Herr Reermanns Arbeit)

Die Hardware Programmierung wurde in Code Composer Studio 5.5.0 geschrieben. Außerdem gibt es noch eine in Java geschriebene, grafische Oberfläche die Daten der Hardware per UART übermittelt bekommt und die empfangenen Daten in eine grafische Darstellung der Vektoren relativ zum Mikrofonarray umsetzt.

Ein kurzer, zu Beginn durchgeführter, Probedurchlauf im alten Compiler und mit der alten Hardware zeigte, dass beide Softwarekomponenten und die Hardware im Rahmen ihrer Leistungsfähigkeit durchaus als funktionsfähig betrachtet werden können.

Um die Funktionsweise der grafischen Oberfläche überprüfen zu können wurde Oracle Eclipse als Java Entwicklungsumgebung auf dem PC installiert.

## 1.3 Aufgabenstellung

Aufgabe der Masterarbeit ist es den Quellcode von Herr Reermann so anzupassen, dass er anschließend auf der neueren Hardware genauso läuft wie auf der alten. Danach wird untersucht inwiefern das Projekt erweitert werden kann um mehr als eine Audioquelle gleichzeitig zu triangulieren

## 1.4 Unterschiede zwischen den beiden Hardware Systemen

Wie man anhand Abbildung 1.1 sehen kann, handelt es sich beim D.Module2.C6657 nicht nur um eine Entwicklungsplattform für ein Dual Core System. Die eingesetzten DSP Systeme sind auch 4 mal so schnell, besitzen 4 mal mehr Flash Speicher und RAM und sind mit 8 mal mehr Cache ausgestattet. Außerdem handelt es sich beim RAM des neuen Systems bereits um DDR3 RAM während die alte Hardware noch SDRAM verwendet. Zusätzlich steht noch ein Megabyte geteilter Speicher zur Verfügung.

| | | |
|---|---|---|
| DSP | TMS320C6657 | dual-core 1.25 GHz fixed- and floating-point, up to 40 GMAC / 20 GFLOP per core |
| Memory | DSP-internal | 32K bytes data cache, 32K bytes program cache per core, 1M byte direct mapped or level-2 cache per core, 1M byte shared RAM |
| | DDR3 | 512M bytes, DDR3-1333, 32-bit wide |
| | Flash | 8M bytes NOR (SPI interface, sector architecture), 64M bytes SLC NAND |

| | | |
|---|---|---|
| DSP | TMS320C6713 | 300 MHz floating-point DSP, 200 MHz in industrial grade |
| Memory | DSP-internal | 4K bytes data cache, 4K bytes program cache, 64K bytes level-2 cache, 192K bytes direct mapped memory |
| | SDRAM | 128M bytes, 100 MHz operation, 32-bit wide |
| | Flash | 2M bytes NOR |

Abbildung 1.1: Vergleich der beiden Hardware Systeme[1][2]

Die benötigte Peripherie, sprich die beiden MCBsp und die UART Schnittstelle sind bei beiden Systemen gleichermaßen vorhanden.

Weitere Unterschiede, wie zum Beispiel zusätzliche Schnittstellen, sind für diese Arbeit nicht entscheidend und sollen hiermit ausreichend erwähnt sein.

## 1.5 Aufbau der Arbeit

Die Arbeit ist in zwei Arbeitspakete eingeteilt. Im ersten Teil der Arbeit wird erläutert welche Schritte nötig waren um die bestehende Software funktionsfähig auf die neue Hardware zu portieren. Dabei wird auch darauf eingegangen an welchen Stellen der Software aufgrund der neuen Hardware Änderungen eingeführt mussten.

Der zweite Teil beschäftigt sich mit dem Postprocessing des Algorithmus und inwiefern es ermöglicht zwei Schallquellen gleichzeitig zu klassifizieren und lokalisieren. Zunächst wird dies am einfachen Beispiel zweier Rauschquellen untersucht. Im Weiteren wird dann versucht auch das klassifizieren zweier Sprecher zu ermöglichen.

Die theoretischen und praktischen Grundlagen zum Verständnis der Portierungsschritte werden im Kapitel *Grundlagen* erklärt. Ebenso in diesem Kapitel zu finden sind die theoretischen Grundlagen zur rekursiven Mittelwertbildung, der Multiplikation von Matritzen, Autokorrelation, Singulärwertzerlegung, der DFFT und zur Fensterung im Zeitbereich sowie die nötigen Grundlagen zum Verständnis der verwendeten Hardwarefunktionen

# 2 Grundlagen

## 2.1 Verwendete Hardwarefunktionen

### 2.1.1 Enhanced Direct Memory Access - EDMA

EDMA ist eine Möglichkeit Daten ohne Belastung der CPU von und zu jedem adressierbaren Speicherbereich zu bewegen. Der Einsatz von EDMA erhöht die CPU Rechenzeit die für weitere Schritte zur Verfügung steht. Ohne EDMA müsste die CPU bei 8 gleichzeitig zu sampelnden Schallsignalen am Eingang mindestens 48000 mal pro Sekunde 8 Werte in den Speicher schreiben und könnte in dieser Zeit keine anderen Aufgaben erfüllen. Mithilfe der EDMA Funktion können stattdessen zunächst immer die kompletten für den nächsten Berechnungsschritt nötigen Blöcke in den Speicher geschrieben und anschließend blockweise von der CPU verarbeitet werden. Die Belastung der CPU verringert sich dabei je nach Blockgröße um ein vielfaches.

Für die vorliegende Arbeit wurde ein Beispielprogramm von DSignT als Grundlage gewählt welches den EDMA Channel Controller so programmiert, dass er die Werte der McBsp Blockweise in den L2 Speicher schreibt. Die *Ping Pong Buffering* Konfiguration des EDMA Controllers stellt dabei eine nahtlose Sampling Kette für die CPU sicher. (siehe hierzu Kapitel *Ping Pong Buffering*)

Weitere Funktionen des EDMA Controllers werden im Rahmen dieser Arbeit nicht benötigt und werden deswegen auch nicht näher erläutert.

#### 2.1.1.1 Ping Pong Buffering

*Ping Pong Buffering* beschreibt eine bestimmte Methode mit einer endlosen Kette von Eingangssamples mit Hilfe von EDMA verlustlos umzugehen. Ohne diese Methode müsste der EDMA Transfer Controller bei jedem Lesezugriff der CPU, also nach jedem vollständig eingelesenen Block, die Dauer des Zugriffs abwarten. In dieser Zeit anfallende Samples wären verloren. Durch *Ping Pong Buffering* wird dieses Problem umgangen.

Der EDMA Channel Controller wird hierfür lediglich instruiert am Ende eines Blockes die Ziel-Startadresse im Speicher zu verändern Während der EDMA Transfer Controller die neu anfallenden Samples nun im neuen Speicherbereich ablegt hat die CPU Zeit um den alten Speicherbereich auszulesen. Am Ende wird dem EDMA Channel Controller als neue Ziel-Startadresse wieder die vom ersten Durchgang übergeben und die CPU kann nun auch die neuen Samples auslesen.

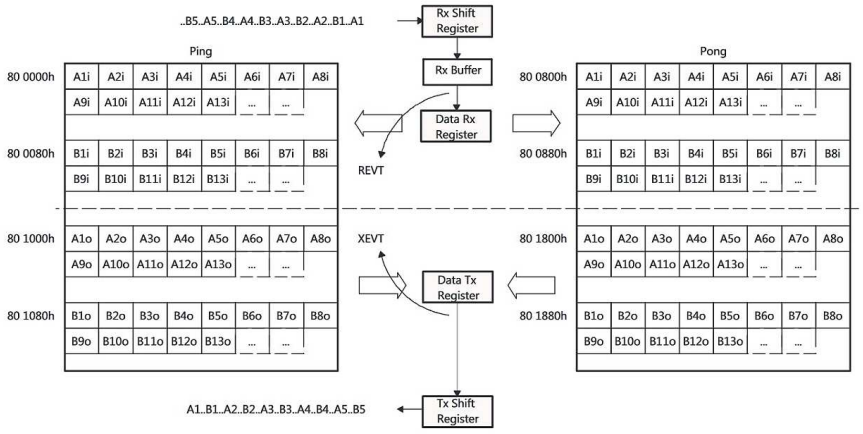


Abbildung 2.1: Visualisierung Ping Pong Buffering[3]

Im hier zu sehenden Beispiel gibt es jeweils zwei Speicherbereiche für das Einlesen(REVT) und das Ausgeben(XEVT). Wurde ein Zielbereich komplett gefüllt wechselt die Zielspeicheradresse. Der aktuell nicht beschriebene Zielbereich kann dann von der CPU in den nächsten auszugebenden Quellbereich kopiert und von dort ausgegeben werden.

Dieses Beispiel entspricht, bis auf die verwendeten Adressen, genau dem Vorgehen in der Basis Software

#### 2.1.1.2 EDMA Parametersets PaRAMSets

In den meisten Fällen wird der EDMA Controller für Aufgaben benötigt, bei denen sich gewisse Instruktionen während des Erfüllens der Aufgabe ändern müssen. Dies gilt auch für das *Ping Pong Buffering*. Um diese Änderungen zu realisieren ist der EDMA Controller mit Speicherbereich für 512 8x32 Bit breiten Registern ausgestattet, die im EDMA Controller eigenen RAM fest adressiert hinterlegt sind. In diesen Registern kann der

Programmierer alle Einstellungen vornehmen die für den jeweiligen Teilarbeitsschritt benötigt werden. Des Weiteren wird hier abgespeichert welches Parameterset am Ende des Teilarbeitsschrittes geladen werden soll.

Die Parametersets haben folgende Struktur:

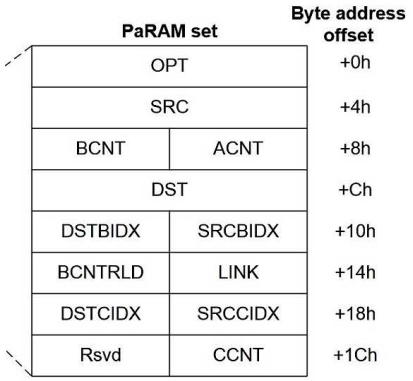


Abbildung 2.2: Struktur der Parametersets[3]

OPT : hier werden die Transfer Konfigurationsoptionen festgelegt. In der vorliegenden Arbeit werden die folgenden beiden Bits verändert. Der Rest kann den Wert 0 behalten:

1.) TCINTEN (Bit 20) - Transfer Complete Interrupt enabled Definiert, dass nach Abarbeiten eines Transfers der unter TCC definierte Bit im IPR gesetzt wird Dadurch wird der CPU mitgeteilt, dass die EDMA Interrupt Service Routine abgearbeitet werden kann

2.) TCC(Bits 17-12) - Transfer Complete Code. Durch diesen Wert wird festgelegt welches Bit im IPR nach Abarbeiten des Transfers gesetzt werden soll. Der Dezimalwert entspricht dabei der Bitposition

SRC: Die Quelladresse des DMA Transfers

BCNT: gibt an wieviele Elemente der Größe ACNT pro Frame übertragen werden sollen

ACNT: gibt an wieviele Byte einem zu übertragenden Wert entsprechen.

DST: Zieladresse des DMA Transfers

DSTBIDX: gibt an um wieviele Adressen die Zieladresse pro übertragenem Wert inkrementiert werden soll

SRCBIDX: gibt an um wieviel Adressen die Quelladresse pro übertragenem Wert inkrementiert werden soll

BCNTRLD: Definiert welcher Wert wieder in BCNT geladen werden soll nachdem alle Elemente eines Frame übertragen wurden und bevor alle Frames übertragen sind

LINK: die Adresse des nächsten zu ladenden Parametersets

DSTCIDX: gibt an um welchen Wert die Zieladresse nach Übertragung eines Frames inkrementiert werden soll

SRCCIDX: gibt an um welchen Wert die Quelladresse nach Übertragung eines Frames inkrementiert werden soll

CCNT: gibt an wie viele Frames übertragen werden sollen bevor das Parameterset abgearbeitet ist.

#### 2.1.1.3 DMA Channel to PaRAM Mapping

Die 64 DMA Kanäle des EDMA Controllers sind fest mit spezifischen Hardwareperipherie Events verknüpft um Transfers direkt durch diese Events triggern zu können. Das Mapping zwischen den entsprechenden DMA Kanälen und den Parametersets ist dabei frei programmierbar. Dies geschieht im EDMA Channel Controller. Hierfür wird lediglich in die Bits 5-13 des DCHMAP Registers des jeweiligen Kanals die Adresse des entsprechenden Parametersets geschrieben. Wird nun das Event des jeweiligen DMA Kanals ausgelöst verarbeitet der EDMA Controller die von dort anfallenden Daten in der Weise wie es im entsprechenden Parameterset beschrieben ist.

Beispiel:

Das Receive Ready Event des McBsp0 hat im EDMA Channel Controller fest die Eventnummer 36

| | | |
|---|---|---|
| 35 | TINT3H | Timer3 interrupt high |
| 36 | MCBSP0_REVT | McBSP_0 receive event |
| 37 | MCBSP0_XEVT | McBSP_0 transmit event |
| 38 | MCBSP1_REVT | McBSP_1 receive event |
| 39 | MCBSP1_XEVT | McBSP_1 transmit event |

Abbildung 2.3: Ausschnitt aus Channel Mapping Tabelle[7]

Dementsprechend ist diesem Event das DCHMAP36 Register zugeordnet

Um also beim Receive Ready Event des McBsp0 Parameterset 0 abzuarbeiten muss in das Register DCHMAP36 in Bit 5-13 die Adresse des Parametersets0 geschrieben werden.

#### 2.1.1.4 Interrupt Pending Register

In den Interrupt Pending Registern werden, unter der Voraussetzung, dass im TCINTEN Bit des OPT Registers eine eins steht, bei der Beendigung eines EDMA Transfers diejenigen Bits auf 1 gesetzt die in den TCC Bits des OPT Registers des verwendeten Parametersets definiert wurden. Falls der Wert in den TCC Bits kleiner als 31 ist werden die entsprechenden Bits im IPR gesetzt. Für Werte zwischen 32 und 63 werden Bits im IPRH Register gesetzt. Erst wenn diese Bits wieder zu null gesetzt wurden kann der nächste EDMA Interrupt ausgelöst werden. Die Bits werden wieder zu null gesetzt indem in ICR/ICRH ( Interrupt Clear Register / Interrupt Clear Register High ) in die korrespondieren Bits eine eins geschrieben wird.[3]

#### 2.1.1.5 A synchronisierter EDMA Transfer

Bei einem A Synchronisierten EDMA Transfer wird bei jedem auftretenden Event, im vorliegenden Fall also entweder das REVT oder XEVT der jeweiligen McBsp, die Anzahl Bits übertragen die im ACNT Register konfiguriert wurde und anschließend die Transferadressen um die gewünschten Werte inkrementiert. Um einen A synchronisierten Transfer zu konfigurieren muss das SYNCDIM Bit (Bit 2) im OPT Register des verwendeten Parametersets auf 0 gesetzt sein.[3]

### 2.1.2 Multichannel Buffered Serial Ports - McBsp

Der TMS320C6657 Chip von TI ist mit zwei frei programmierbaren McBsp Schnittstellen ausgestattet. Diese beiden Schnittstellen erlauben die direkte Verbindung zu anderen DSP, codecs und anderen Geräten im System. In der vorliegenden Arbeit werden die beiden McBsp als serielle Schnittstelle zur 8 Kanal D.Module.PCM3003 Audio Codec Platine verwendet.

Jeder McBsp stellt einen seriellen Datenkanal und einen Kontrollkanal zur Verbindung mit externen Geräten zur Verfügung. Die CPU kommuniziert mit den MCBsp über die 32-Bit Kontrollregister die über den internen peripherie Bus angesprochen werden können.

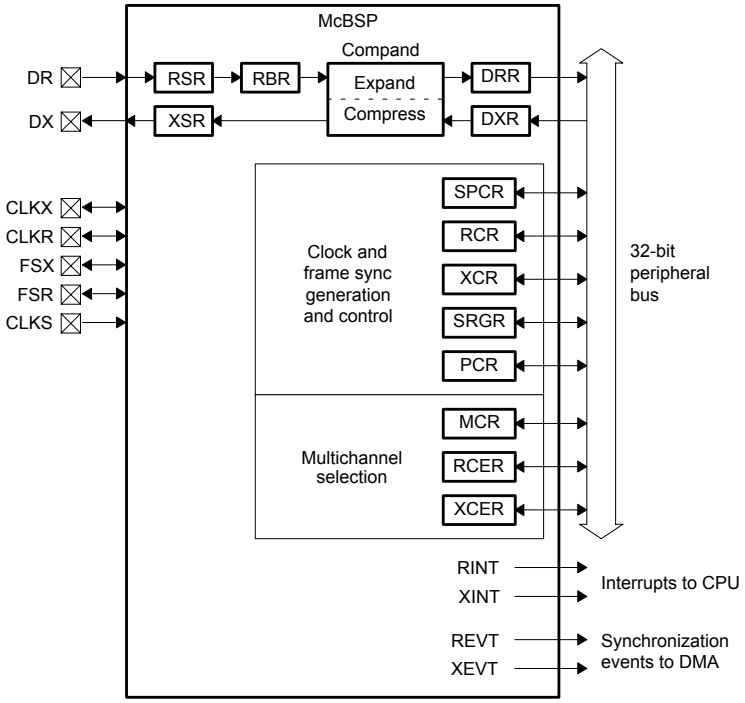


Abbildung 2.4: Blockdiagramm McBsp[4]

Über die Datenkanäle DR(Read) und DX(Write) können pro McBsp bis zu 128 Kanäle annähernd gleichzeitig ausgelesen und ausgegeben werden.

Clock Source und Frame Synchronization Source sowie die entsprechenden Clock Divider und Wartezeiten können mit Hilfe der *Clock and Frame Sync Generation and Control* Register für das Senden(X) und das Empfangen(R) von Daten separat gewählt und eingestellt werden. Die *Multichannel Selection Register* dienen der Einstellung des *Multichannel Mode*. Dieser wurde für die vorliegende Arbeit nicht benötigt und wird deswegen nicht weiter erläutert.

Jeder der beiden McBsp löst für das Empfangen und das Senden einen CPU Interrupt (RINT, XINT) und ein EDMA Synchronization Event (REVT, XEVT) aus.

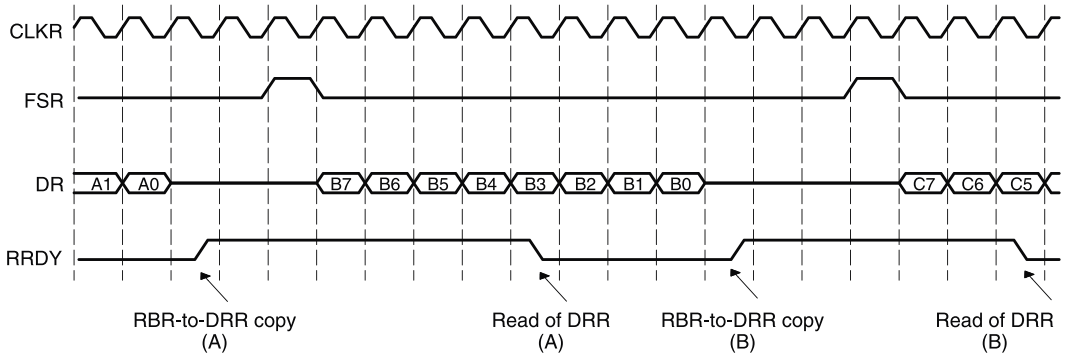


Abbildung 2.5: Timing McBsp[4]

Abbildung 2.5 zeigt den typischen Timing-Verlauf einer *single Frame, 1 Kanal, 8 Bit* Standard-Leseoperation eines McBsp. (Dies ist nur ein ganz einfaches Beispiel. Für den Einsatz eines PCM3003 Audio Codec an einem McBsp müssen diese Einstellungen an die Spezifikationen des AD Wandlers angepasst werden.)

Wie man sieht werden nach Auftreten eines *Frame Synchronization Events*, in diesem Fall *Falling Edge*, mit jeder steigenden Flanke des Clock Signals 1 Bit am Eingangskanal(DR) gelesen und ins RSR(Receiver Shift Register) geschoben. Entsprechend dem in den RDATDLY Bits (Read Data Delay) im RCR (Receiver Control Register) definierten Delay wird danach mit *Rising Edge* des Clock Signals der Inhalt des RSR, unter der Vorraussetzung, dass RBR (Receiver Buffer Register ) nicht noch mit Daten des vorherigen Durchgangs voll ist, ins RBR Register kopiert. Ist nun die Anzahl Bits die vom Programmierer festgelegt wurden erreicht erfolgt ein Kopieren der Daten aus RBR ins DRR ( Data Receive Register ), was dazu führt, dass das RRDY Bit ( Receiver Ready) im SPCR ( Serial Port Control Register ) in den High Zustand übergeht. Dies signalisiert nun der CPU bzw. dem EDMA Kontroller, dass Daten zum Lesen zur Verfügung stehen. Das Auslesen der Daten aus dem DRR Register setzt das RRDY Bit wieder zurück auf Null.

Durch das Buffern der Daten in DRR ergibt sich so die Möglichkeit, dass bereits neue Werte in RSR gelesen werden können während die alten Daten noch auf die Abholung durch die CPU oder den EDMA Kontroller warten. Das System ist also weniger empfindlich auf im Betrieb auftretenden timing Unterschiede.

Da zum Einen eine Transmit Operation sehr ähnlich abläuft und zum Anderen das Ausgeben von Samples für diese Arbeit nicht wichtig ist wird diese Operation hier nicht weiter erläutert.

### 2.1.3 Universal Asynchronous Receiver Transceiver - UART

Der TMS320C6657 Chip von TI ist mit zwei UART Schnittstellen ausgerüstet. Es handelt sich dabei um einen bi-direktionalen, asynchronen Datenbus, der sowohl Daten der CPU zunächst serialisieren und danach an ein peripheres Gerät senden, als auch seriell aus der Peripherie eintreffende Daten zunächst parallelisieren und dann an die CPU weiterreichen kann. Zusätzlich können Daten jedoch auch ohne zwischen Buffern gesendet und empfangen werden. Durch die ersten der beiden Möglichkeiten hat die CPU die Möglichkeit mit Peripherie zu kommunizieren , ohne dass ein kompliziertes, Softwareseitiges Management nötig wäre.[5]

UART ist ein weit verbreiteter Datenbus Standard und entspricht dem Format in dem der PC Daten über die serielle Schnittstelle empfängt und sendet.

In der vorliegenden Arbeit wird die UART Schnittstelle benutzt um Daten an den seriellen Com-Port des PC zu senden und diese in der GUI für eine grafische Darstellung der von der Hardware errechneten Vektoren zu verwenden. Es wird also lediglich die Transmitter-Funktion genutzt. Ein empfangen von Daten vom PC ist nicht vorgesehen.

### 2.1.4 Timer

Der TMS320C6657 stellt insgesamt 8 64 Bit Timer zur Verfügung. Jeder dieser Timer kann in drei verschiedenen Modi programmiert werden.[6]

- 64 bit general purpose timer
- dual 32 bit timer (chainned/unchained)
- watchdog timer

Im Rahmen dieser Arbeit wurde ein Timer benötigt und als 32 Bit Timer im *unchained Modus* eingerichet. Im *unchained* Modus kann ein 64 Bit Timer als zwei unabhängige 32 Bit Timer verwendet werden.

Im vorliegenden Fall wurden die unteren 32 Bit des Timer Period- und Timer Interrupt Registers verwendet. Abbildung 2.6 zeigt die Register der oberen. Für die Visualisierung macht dies keinen Unterschied.

Einmal gestartet wir im Prescale Counter des jeweiligen Timers bei jedem Clock Signal der aktuelle Wert um 1 erhöht. Hat der Prescaler den vorher festgelegten Wert erreicht wird ein Signal an den Timer Counter gegeben. Ist dies so oft passiert wie der Programmierer es vorher im Timer Period Register festgelegt hat wird ein Puls erzeugt, der

den Timer Interrupt an die CPU auslöst. Im Anschluss wird die vorher definierte Timer Interrupt Service Routine abgearbeitet.

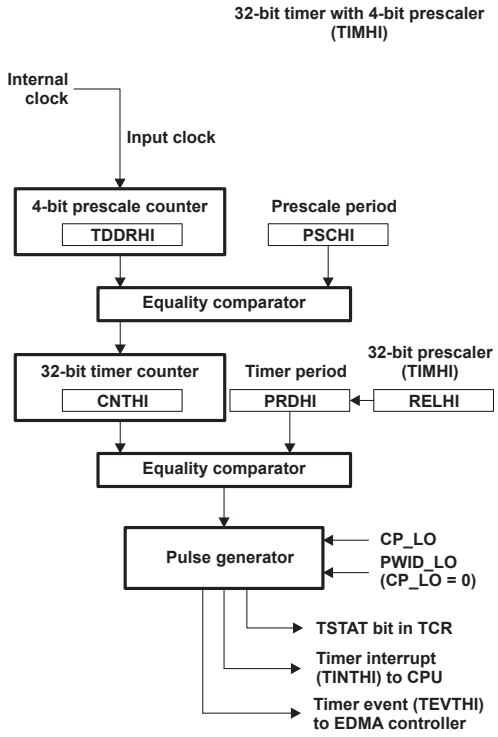


Abbildung 2.6: Blockdiagramm Timer 32 Bit Unchained[6]

Der Timer wird im vorliegenden Projekt im Rahmen der Systemkalibrierung eingesetz. Da die genannten Eigenschaften des Timers hierfür ausreichen sollen die weiteren Funktionen nicht weiter erläutert werden.

### 2.1.5 Interprozessor Interrupts

Da es sich beim TMS320C6657 um einen Dual Core System handelt gibt es auch die Möglichkeit Interprozessor Interrupts auszulösen um beide Prozessoren synchronisieren zu können während sie zusammen an einer Aufgabe arbeiten. Für das Handling dieser Interrupts stehen pro Prozessor je ein 32 Bit breites Interprocessor Interrupt Generation(IPCGR) und ein Interprocessor Acknowledgement Register(IPCAR) zur Verfügung.

Wenn beide Prozessoren arbeiten und mindestens beim Empfänger Prozessor eine Interrupt Service Routine für den entsprechenden Interprozessor-Interrupt eingerichtet ist, wird diese ausgelöst sobald in das erste Bit des IPCGR eine eins geschrieben wurde. 28 der restlichen 32 Bit stehen zur Übermittlung einer Source ID zur Verfügung, um der Empfänger CPU signalisieren zu können in welchem Kontext der Interrupt ausgelöst wurde. Die Empfänger CPU bestätigt dem Sender das Abarbeiten der Interrupt Service Routine in dem in die Korrespondierenden Bits des IPCAR eine 1 geschrieben wird. Dadurch werden beide Register wieder zurück auf Null gesetzt und die CPU ist bereit für den nächsten Interrupt.[7]

## 2.2 Konfiguration und Bibliotheken

### 2.2.1 Target Configuration TMS320C6657.ccxml

Die Datei TMS320C6657.ccxml enthält alle Konfigurationsinformationen zur Hardware, dem Emulator, der Compiler Version, dem Output Format und den benötigten Treibern. Sie kann automatisch in Code Composer Studio erzeugt oder selbst, entweder mit einem Texteditor oder der grafischen Oberfläche in Code Composer Studio modifiziert oder erzeugt werden. Sie befindet sich in im Unterordner *targetConfigs* im Projektordner. Es hat sich gezeigt, dass es sich, bei Fehlern die während der Verbindung,während des Programmierens oder während des Setzens von Haltepunkten auftreten, anbietet, den Emulator vom USB Port zu trennen und nachdem die Verbindung erneut eingerichtet wurde die Ziel Konfiguration neu zu laden.

### 2.2.2 Dsplib.ae66

Seit Code Composer 5.3 ist das Output Format von C66xx Projekten standardmäßig auf ELF eingestellt. Die für dieses Format nötigen Bibliotheken sind , nicht wie beim älteren Format COFF in einer .lib Datei sondern in der Datei dsplib.ae66 hinterlegt. Wie bei der COFF Version handelt es sich hierbei um eine von TI bereitgestellte Bibliothek, die auf die Hardware optimierte Assembler Funktionen enthält die oft im Bereich der digitalen Signalverarbeitung eingesetzt werden müssen. Durch die Optimierung der Hardware auf den Assembler Befehlssatz reduziert sich die Ausführdauer von Programmen durch den Einsatz dieser Funktionen deutlich im Vergleich zum Einsatz von in ANSI C geschriebenen Funktionen

Wie bereits in den vorhergehenden Versionen lassen sich die Funktionen dieser Bibliothek in 6 Bereiche einteilen:[8]

- 1. Adaptives Filtern
- 2. Korrelation
- 3. FFT
- 4. Filtern und Falten
- 5. Vektoren und Matritzen
- 6. Sonstiges

In dieser Arbeit werden die folgenden drei Funktionen benutzt beziehungsweise diskutiert:

DSPF_sp_fftSPxSP: Radix 2 FFT mit Bit Reversal

DSPF_sp_maxidx: Indexbestimmung des Maximums eines Vektors

DSPF_sp_mat_mul_cplx: Komplexe Matrix Multiplikation

### 2.2.3 Output Format

Beim Kompilieren eines Programms in Code Composer Studio wird mit den gesammelten Einstellungen und Funktionen aus .ccxml, .cmd, .aee (bzw. .lib) und dem in Assembler übersetzten ANSI C Programm eine .out Datei erzeugt, die dann auf die Hardware übertragen und ausgeführt werden kann. Diese Datei kann in zwei verschiedenen Formaten angelegt werden.[9]

1. ELF: Executable and Linking Format. Das aktuell am meisten verwendete Format und das Format, dass am ehesten als standardisiert gesehen werden kann. Programme die in CCS6 für die C66xx Familie Kompiliert werden haben per default dieses Output Format. COFF ist zwar wählbar, man muss jedoch bedenken, dass alle Bibliotheken und Einstellungen für dieses Format ausgelegt sein müssen. Ansonsten wird kein .out File erzeugt und der Compiler wirft eine Fehlermeldung

2. COFF: Common Object File Format. Ein ehemals für Unix Systeme entworfenes Dateiformat das Debugging und Portierung von Programmen vereinfachen sollte. Wie viele andere Firmen ( z.B. Microsoft) hat TI dieses Dateiformat aufgegriffen und dann den eigenen Ansprüchen angepasst. COFF ist noch weniger

als Industriestandard zu sehen als ELF und die COFF Formate verschiedener Firmen sind meist nicht im geringsten kompatibel.

Da für das Beispiel Programm von DsignT auf dem die Portierung der Vorgängerarbeit aufbaut als Output Format ELF gewählt war ist man im weiteren Verlauf dabei geblieben. Das andere Format soll nur der Vollständigkeit wegen erwähnt sein.

### 2.2.4 Linker Command File

Im jedem Linker Command File sind mindestens die Informationen zur Speicher Namens Konvention des Systems hinterlegt. Grundsätzlich könnte diese Datei auch noch weitere Informationen über prinzipiell alles was in der Kommando Zeile vorkommt enthalten. Dies ist jedoch im vorliegenden Fall nicht gegeben.[10]

In jedem Linker Command File sind zwei hauptsächliche Direktiven enthalten.[10]

- Memory Directive (Speicher Direktive)
- Section directive (Sektionen Direktive)

Die Aufgabe der Speicher Direktive ist es den Speicherbereichen des Chips Namen zuzuweisen die dann von der Sektionen Direktive benutzt werden können

Die Sektionen Direktive hat zwei Aufgaben.

- aus Output Sektionen werden Input Sektionen gebilded
- den erzeugten Output Sektionen werden Speicherbereiche zugeordnet

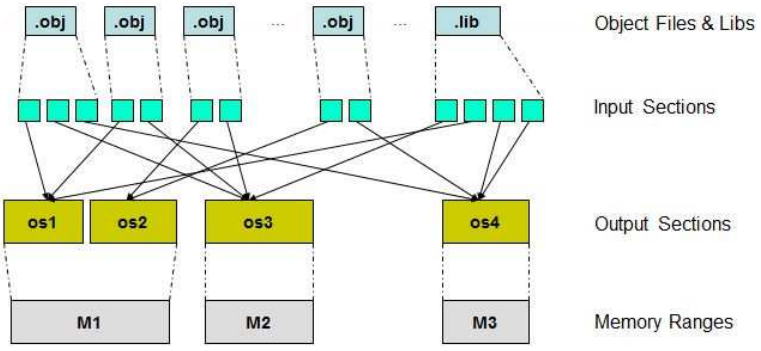


Abbildung 2.7: Funktionsweise Linker Command File[10]

Wie man in Abbildung 2.7 sehen kann besteht zum Beispiel die .lib Datei aus 4 Input Sektionen. Jede dieser Input Sektionen wird zunächst einer Output Sektion zugeordnet. Anschliessend werden diese Output Sektionen auf Speicherbereiche verteilt.

In der Sektions Direktive steht also welche Information über vorhandene und benutzte Funktionen an welcher Stelle im Speicher abgelegt werden.

Für das vorliegende Projekt wird die von D.SignT bereit gestellte Datei dm2c6657.cmd verwendet, in der bereits alle wichtigen Einstellungen richtig vorgenommen wurden.

Im Verlauf wird sie nur noch als Nachschlagewerk genutzt um zu überprüfen welche Adressbereiche in geteiltem oder nicht geteiltem Speicher liegen. Dies war wichtig für Experimente mit den Interprozessorinterrupts

## 2.3 Theoretische Grundlagen

### 2.3.1 Matritzenmultiplikation

Die Multiplikation von Matritzen wird an diversen Stellen des Quellcodes benötigt. Zum besseren Verständnis des Codes soll hier kurz auf die Vorgehensweise und den Effekt den die Operation auf eine gegebene Matrix hat eingegangen werden.

Grundsätzlich werden Matritzen Elementweise miteinander multipliziert und die Ergebnisse dieser Multiplikation in der Zielmatrix aufsummiert. Dabei muss beachtet werden, dass Reihenanzahl und Spaltenanzahl beider Matritzen übereinstimmen.[11]

Die folgende Grafik soll dies verdeutlichen:

Bei der Multiplikation der 3xn Matrix A mit mx3 Matrix B entsteht also eine nxm Matrix C. Die Elemente der Matrix C werden nach folgenden Schema berechnet:

Hier Beispielhaft für $c_{00}$

$$c_{00} = a_{00} \cdot b_{00} + a_{01} \cdot b_{10} + a_{02} \cdot b_{20} \tag{2.1}$$

Fast man die Spaltenelemente der Matrix A und C als Vektoren auf so entsprechen die Elemente der Matrix B den Skalaren mit denen die jeweiligen Spaltenvektoren der Matrix A vor der Addition multipliziert werden.

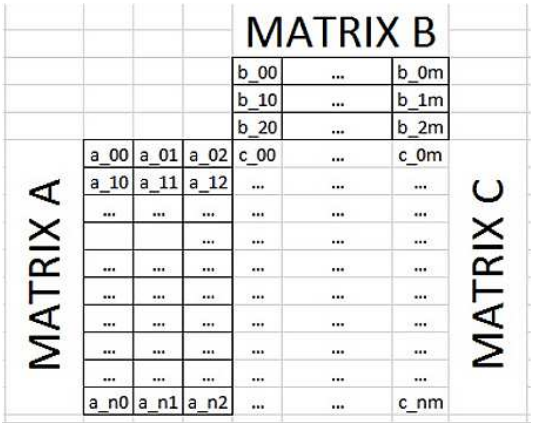


Abbildung 2.8: Multiplikation von Matritzen

$$\begin{pmatrix} c_{00} \\ \dots \\ c_{n0} \end{pmatrix} = b_{00} \cdot \begin{pmatrix} a_{00} \\ \dots \\ a_{n0} \end{pmatrix} + b_{01} \cdot \begin{pmatrix} a_{01} \\ \dots \\ a_{n1} \end{pmatrix} + b_{02} \cdot \begin{pmatrix} a_{02} \\ \dots \\ a_{n2} \end{pmatrix} \tag{2.2}$$

Durch die Multiplikation wird also der Vektorraum der Matrix A enstprechend der Elemente in Matrix B gezerrt und entsprechend der neuen Vektorelementlängenverhältnisse gedreht.

Diese Eigenschaften der Multiplikation von Matritzen macht man sich zur Faktorisierung, zum Beispiel bei der Singulärtwertzerlegung, oder während des Beamforming zunutze. (siehe Kapitel : *Singulärwertzerlegung*)

Die Multiplikation von Matritzen ist nicht kommutativ. Es gilt also, abgesehen davon, dass, aufgrund der Dimensionen, im obigen Beispiel so oder so nur $A \cdot B$ berechnet werden könnte.

$$A \cdot B \neq B \cdot A \tag{2.3}$$

#### 2.3.1.1 komplexe Matritzen

Bei der Multiplikation von komplexen Matritzen ist zu beachten, dass jedes Element für sich einen komplexen Zeiger der Form $x_{nm} + i \cdot y_{nm}$ darstellt. Es werden also Vektoren, deren Elemente komplexe Zeiger darstellen mit weiteren komplexen Zeigern multipliziert.[11] Die Rechenoperation soll hier beispielhaft für ein Element eines solchen

Vektors gezeigt werden. (Es wurde die kartesische Form gewählt, da dies der Form entspricht in der komplexe Zahlen vom DSP verarbeitet werden).

$xa_{nm}, ya_{nm}$ : Elemente des komplexen Zeigers an der Position nm in Matrix A

$xb_{ml}, yb_{ml}$ : Elemente des komplexen Zeigers an der Position mi in Matrix B

$$(xa_{nm}+i\cdot ya_{nm})\cdot(xb_{mi}+i\cdot yb_{mi}) = xa_{nm}\cdot xb_{mi}-ya_{nm}\cdot yb_{mi}+i\cdot(ya_{nm}\cdot xb_{ml}+xa_{nm}\cdot yb_{mi}) \tag{2.4}$$

Das Ergebnis der Rechenoperation ist ein neuer komplexer Zeiger dessen Betrag dem Produkt der Beträge, und dessen Phase der Summe beider Phasen der Eingangszeiger entspricht. Dies wird deutlich wenn man die Multiplikation in der eulerschen Form betrachtet.

$$|a|e^{j\omega\varphi_a}\cdot|b|e^{j\omega\varphi_b} = |a||b|\cdot e^{j\omega(\varphi_a+\varphi_b)} \tag{2.5}$$

Bei der Multiplikation komplexer Matritzen wird also der komplexe Zeiger jedes Elementes der von links mutliplizierten Matrix entsprechend der komplexen Zeiger in der von rechts multiplizierten Matrix gestreckt und gedreht.

Einen Sonderfall stellt die Multiplikation zweier, zueinander komplex-konjugierter Zeiger dar. Oben genannten Gleichung vereinfacht sich dann zu

$$|a|e^{j\omega\varphi_a}\cdot|a|e^{-j\omega\varphi_a} = |a|^2\cdot e^{j\omega(\varphi_a-\varphi_a)} = |a|^2 \tag{2.6}$$

Durch die Multiplikation zweier komplex-konjugierter Zeiger erhält man also das Betragsquadrat des Zeigers.

### 2.3.2 Autokorrelation

Für das Anwenden des UCA-ESPRIT Algorithmus ist es nötig die Autokorrelationsfunktionen der Signale der 8 Mikrofone untereinander nach der DFFT in einem bestimmten Bereich um ausgewählte Frequenzen herum zu berechnen.

Die Autokorrelationsfunktion beschreibt die Ähnlichkeit eines Signals zu sich selbst zu einem späteren Zeitpunkt oder, wie im vorliegenden Fall, zum Signal an einer anderen Messposition.

Aus Gründen der Verständlichkeit soll hier zunächst ein Beispiel mit nur zwei, reellwertigen Signalen besprochen werden.

Die Autokorrelationsfunktion für ein deterministisches, kontinuierliches Signal in Abhängigkeit von t wird beschreiben durch:[12]

$$r_{(t)} = \int_{-\infty}^{\infty} x_{(t)} \cdot x_{(t+\tau)} \, \mathrm{d}\, t \tag{2.7}$$

Wobei $\tau$ den zeitlichen Abstand der Messungen des Signals beschreibt.

Beim Übergang vom unendlichen, kontinuierlichen zum endlichen, diskreten wird daraus:

$$r_{(n)} = \sum_{n=0}^{N} x_{(n)} \cdot x_{(n+i)} \tag{2.8}$$

Hier beschreibt i den diskreten zeitlichen Abstand zwischen den Messungen eines Signals. Allerdings ist es auch ohne weiteres Möglich für n verschiedene Messpositionen ein und des selben Signals zu verwenden um ein Maß dafür zu erhalten wie stark sich zum Beispiel die Energie eines Signals zwischen zwei Punkten geändert hat.

Die Funktion erreicht ihr Maximum immer für i bzw. $\tau$ = 0, also wenn das Signal genau mit sich selbst korreliert wird. Im genannten Beispiel würde dies Bedeuten, dass die Energie an beiden Messpositionen identisch war. Je größer die Differenz der Energien, desto kleiner wäre im Beispiel die Autokorrelationsfunktion im Verhältnis zur Autokorrelationsfunktion des Signals mit sich selbst.

Betrachtet man die diskrete Autokorrelationsfunktion genauer so fällt auf, dass sie im Prinzip einer diskreten Faltung der beiden Signale entspricht. Alle Elemente werden zunächst miteinander multipliziert und die Ergebnisse der Multiplikation aufaddiert.

Dies ist auch das Vorgehen bei der Multiplikation von Matritzen. (siehe Kapitel: *Matritzenmultiplikation*)

Um also die Autokorrelation der Energie von M, an N Positionen gemessenen Frequenzen zu bestimmen genügt es, die Messwerte als N-dimensionalen Vektor in eine M Spalten breite Matrix einzutragen und diese Matrix mit ihrer eigenen Transponierten zu multiplizieren.

Beispiel für M = 3 und N = 2;

$$\begin{pmatrix} a_1 & a_2 & a_3 \\ b_1 & b_2 & b_3 \end{pmatrix} \cdot \begin{pmatrix} a_1 & b_1 \\ a_2 & b_2 \\ a_3 & b_3 \end{pmatrix} = \begin{pmatrix} \sum_{n=1}^{3} a_n \cdot a_n & \sum_{n=1}^{3} a_n \cdot b_n \\ \sum_{n=1}^{3} b_n \cdot a_n & \sum_{n=1}^{3} b_n \cdot b_n \end{pmatrix} \tag{2.9}$$

Das Ergebnis ist eine quadratische Matrix deren Hauptdiagonalelemente immer der Summe aller Autokorrelationswerten bei $r_{nm}$ bzw. $i_{nm}$ = 0 entsprechen. Alle anderen Werte sind ein Maß dafür wie stark sich die Energien der jeweiligen Frequenzen an den entsprechenden Positionen ähneln.

#### 2.3.2.1 komplexe Autokorrelationsmatrix

Um die Autokorrelation einer komplexen Matrix zu berechnen muss sie von links mit ihrer Adjungierten multipliziert werden. Die Adjungierte einer Matrix ist ihre konjugiert--transponierte. Das Ergebnis ist eine Matrix in deren Hauptdiagonalen die Summen der Betragsquadrate der komplexen Zeiger der enstprechenden Reihenelemente aus der ursprünglichen Matrix stehen. Alle anderen Elemente sind komplexe Zeiger.

### 2.3.3 Singulärwertzerlegung

Mit Hilfe der Singulärwertzerlegung wird vor der endgültigen Berechnung der resultierenden Azimut und Elevationswinkel während des UCA-ESPRIT-Algorithmus eine Entscheidungsschwelle definiert.

Die Singulärwertzerlegung ist eine Möglichkeit Matritzen zu faktorisieren. Dabei macht man es sich zunutze, dass der Vektorraum jeder Matrix beschrieben werden kann durch zwei Drehungen und eine Verzerrung der Einheitsmatrix.[17]

$$A = U\Sigma V^T \tag{2.10}$$

Die beiden Matritzen U und V sind orthogonal bzw. unitär. Das heißt bei der Multiplikation mit ihrer transponierten entsteht die Einheitsmatrix. Matritzen dieser Art haben die Eigenschaft, dass sie Verzerrungsfreie Drehungen und Spiegelungen an den Matritzen durchführen können die mit ihnen multipliziert werden.

Die Matrix $\Sigma$ ist eine Diagonalmatrix deren Elemente den Singulärwerten der Matrix A entsprechen. Durch sie wird die zur Darstellung der Matrix A nötige Verzerrung der durch $V^T$ gedrehten Einheitsmatrix erreicht.

An der Matrix $\Sigma$ lässt sich ablesen wie stark die Ausprägung des Vektorraums der durch $V^T$ gedrehten Einheitsmarix in einer bestimmten Dimension ist. Hohe Werte bedeuten der Vektorraum erstreckt sich entlang dieser Dimension eher weit, niedrige Werte bedeuten das Gegenteil.

Es lassen sich also Schlüsse über die Vektorlängenverhältnisse der Matrix A aus den Singulärwerten ziehen.

### 2.3.4 Diskrete Fast Fourier Transformation

Die DFFT ist eine schnelle Implementierung der diskreten Fourier Transformation. Durch die besondere Gestalt der für die diskrete Fourier Transformation nötige Fourier Matrix läßt sich durch geschickte Faktorisierung der Rechenaufwand von $N^2$ Multiplikationen auf $\frac{1}{2} \cdot N \cdot log_2 N$ vermindern. Dabei beschreibt N die Länge des Eingangsvektors der Transformation und NxN die Größe der entsprechenden Fourier Matrix.[13]

Die diskrete Fourier Transformation beschreibt eine diskrete Faltung des Eingansvektors mit den Lösungen der Gleichung $e^{\frac{-2\pi jnk}{N}}$ mit $n; k \in [0; N-1]$

Sie lässt sich damit wie folgt schreiben:[12]

$$X_{(k)} = \sum_{n=0}^{N-1} x_{(n)} e^{\frac{-2\pi jnk}{N}} \tag{2.11}$$

Wie im Kapitel *Autokorrelation* bereits angesprochen entspricht eine diskrete Faltung im Prinzip einer Matritzenmultiplikation. Für die diskrete Fourier Transformation muss also eine NxN Matrix, deren Elemente die Lösungen der Abtastfunktion $e^{\frac{2\pi jnk}{N}}$ beschreiben,von links mit dem Eingansvektor multipliziert werden.

Diese NxN Matrix ist die sogenannte Fourier Matrix $F_N$ und ihre Elemente berechnen sich ganz einfach nach der Vorschrift j = Zeile, k = Spalte mit $j; n \in [0; N-1]$[13]

Die Fast Fourier Transformation zielt nun darauf ab, diese Matrix zu minimieren um möglichst viel Rechenaufwand einzusparen.

Im ersten Schritt der DFFT wird der Eingangsvektor in zwei Eingangsvektoren halber Länge zerlegt. Hierfür werden für den ersten Eingangsvektor lediglich alle Elemente mit gerade Indizes und für den zweiten lediglich die mit ungeraden Indizes verwendet. Die beiden Eingangsvektoren stellen nun zwei Phasenversetzte Samplesequenzen halber Abtastrate des gleichen Signals dar:

$$X_{(k)} = \sum_{n=0}^{\frac{N}{2}-1} x_{(2n)} e^{\frac{-2\pi j \cdot 2nk}{\frac{N}{2}}} + \sum_{n=0}^{\frac{N}{2}-1} x_{(2n+1)} e^{\frac{-2\pi j \cdot (2n+1)k}{\frac{N}{2}}} \tag{2.12}$$

Durch die Periodizität der Abstastfunktion mit $2\pi$ kann dies umgeformt werden zu:

$$X_{(k)} = \sum_{n=0}^{\frac{N}{2}-1} x_{(2n)} e^{\frac{-2\pi jnk}{\frac{N}{2}}} + e^{\frac{-2\pi jk}{N}} \sum_{n=0}^{\frac{N}{2}-1} x_{(2n+1)} e^{\frac{-2\pi jnk}{\frac{N}{2}}} \tag{2.13}$$

Wie man sieht können die beiden Anteile also getrennt voneinander mit einer $\frac{N}{2} x \frac{N}{2}$ Fourier Matrix bearbeitet werden.

Anschliessend muss lediglich dafür Sorge getragen werden, dass bei der Zusammenführung der beiden resultierenden Teil-Ausgangsvektoren der fehlende Phasenversatz zwischen den beiden Teil-Eingangsvektoren wieder ausgeglichen wird.

Die Vorfaktoren, die diesen Phasenversatz beschreiben sind die Lösungen der Gleichung $e^{\frac{-2\pi jk}{N}}$ mit $k \in [0; N-1]$ und werden als Twiddle-Faktoren bezeichnet. Da sie nur von k, also der Eingangsvektorlänge der DFFT, abhängig sind können sie bei Systemstart berechnet, im Speicher abgelegt und als letzter Schritt der DFFT mit den resultierenden Ausgangsvektoren der aufgeteilten DFT multipliziert werden.

Betrachtet man die Twiddle-Faktoren genauer so fällt auf, dass gilt:

$$e^{\frac{-2\pi jk}{N}} = -e^{\frac{-2\pi j(k+\frac{N}{2})}{N}} \tag{2.14}$$

Es müssen also tatsächlich nur $\frac{N}{2}$ Twiddle-Faktoren berechnet werden. Für Werte von $k \geq \frac{N}{2}$ werden einfach wieder die Werte von $k \in [0; \frac{N}{2} - 1]$ mit umgekehrten Vorzeichen verwendet.

Um nach der Aufteilung schliesslich zum endgültigen Ergebnis zu kommen müssen zunächst die beiden Teil-Ausgangsvektoren wieder zu einem Vektor zusammengeführt werden. In diesem Vektor stehen nun an den ersten $\frac{N}{2}$ Positionen die Ergebnisse der Faltung der reduzierten Fourier Matrix mit den Elementen gerader Inidizes des Eingangsvektors, in den letzen $\frac{N}{2}$ Elementen die der ungeraden Indizes

Der so erhaltene Vektor wird dann mit folgender Matrix multipliziert um den fehlenden Phasenversatz auszugleichen und die Samples wieder in die richtige Reihenfolge zu bringen:[13]

$$\begin{pmatrix} E_{\frac{N}{2}} & D_{\frac{N}{2}} \\ E_{\frac{N}{2}} & -D_{\frac{N}{2}} \end{pmatrix} \tag{2.15}$$

In der Hauptdiagonale von D stehen die Hauptdiagonal-Elemente der Fourier Matrix $F_N$ zwischen 0 und $\frac{N}{2}$, die genau den Lösungen der Gleichung $e^{\frac{-2\pi jk}{N}}$ mit $k \in [0; N/2]$ entsprechen. Alle anderen Elemente von D sind null. E ist die Einheitsmatrix.[13]

Würde man dieses Vorgehen nur einmal anwenden wäre die Anzahl nötiger Multiplikationen bereits um ca. die Hälfte reduziert worden.

Durch rekursives Anwenden dieses Vorgehens auf $\frac{N}{4}$; $\frac{N}{8}$; $\frac{N}{16}$ und so weiter; verringert sich die Anzahl der benötigten Multiplikationen am Ende dann um den zu Begin des Kapitels genannten Faktor von $\frac{1}{2} \cdot N \cdot log_2 N$

### 2.3.5 Fenstern im Zeitbereich und Sample-Overlap

Da die Länge der Eingangsvektoren der DFFT nicht unendlich sein kann wird das Signal während des Sampling und der Vorbereitung der DFFT zweimal im Zeitbereich mit Funktionen multipliziert. Zunächst während des Sampling mit der diskreten Samplingfunktion. Dann, bedingt durch die endliche Länge des Eingangsvektors der DFFT, mit einem Rechteckfenster dessen Breite der Länge des Eingangsvektors entspricht. Analog zur Multiplikation im Frequenzbereich, die einer Faltung im Zeitbereich entspricht, entspricht eine Multiplikation im Zeitbereich aber einer Faltung im Frequenzbereich

Während des Abtastens wird also das Spektrum des Signals mit dem Spektrum der Samplingfunktion, und die so erhaltene Funktion dann bei der Vorbereitung der DFFT mit dem Spektrum des Rechteckfensters gefaltet.[14]

Bei der DFFT des Signals entstehen so Artefakte die im eigentlichen Signal nicht enthalten waren.

Anschaulich dargestellt kann man sagen, dass bei der DFFT davon ausgegegangen wird, dass sich der durch das Sampling erhaltene Aussschnitt des Signals periodisch in der Vergangenheit und der Zukunft des Signals fortsetzt. Dies wird dann zu einem Problem, wenn das erste und das letzte Sample des aktuellen Fensters stark unterschiedliche Beträge haben.

Folgende Grafik soll dies verdeutlichen:

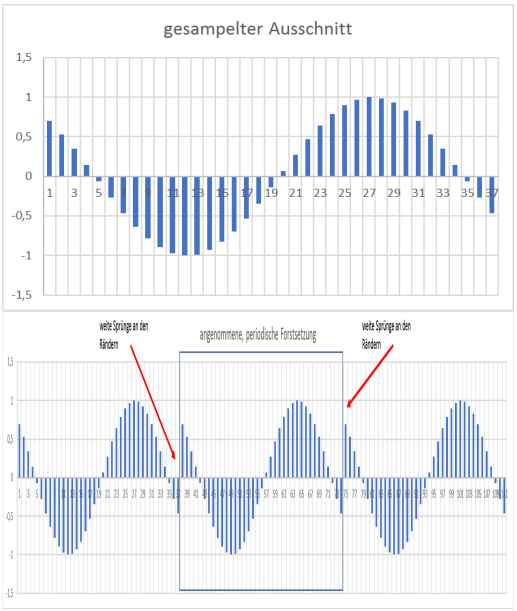


Abbildung 2.9: Sprünge durch periodische Wiederholung des gesampelten Ausschnitts

Wie man sieht entstehen durch die periodische Wiederholung dieses Signals weite Sprünge an den Rändern. Diese Sprünge werden im Spektrum durch Frequenzanteile repräsentiert die im ursprünglichen Signal nicht enthalten sind. Man spricht hierbei vom sogenannten Leck-Effekt.

Um den Leck-Effekt auszugleichen müssen die einzelnen Samples vor der DFFT entsprechend ihrer Position in der Sampling Kette gewichtet werden. Ähnlich wie beim Bandpass-Filter im Frequenzbereich darf das Fenster für die Vorbereitung der DFFT darf also nicht Rechteckig sein sondern muss eine Funktion beschreiben, die Samples an den Rändern weniger stark gewichtet als die Samples in der Mitte der Samplingkette.

Das Fenster, dass hier die schmalsten Peaks im Spektrum bei gleichzeitig höchster Dämpfung im Leck-Effekt-Bereich verspricht ist ein Tschebyscheff-Fenster.[8] Durch die Multiplikation des gesampelten Ausschnitts mit diesem Fenster werden die genannten Sprünge vermieden.

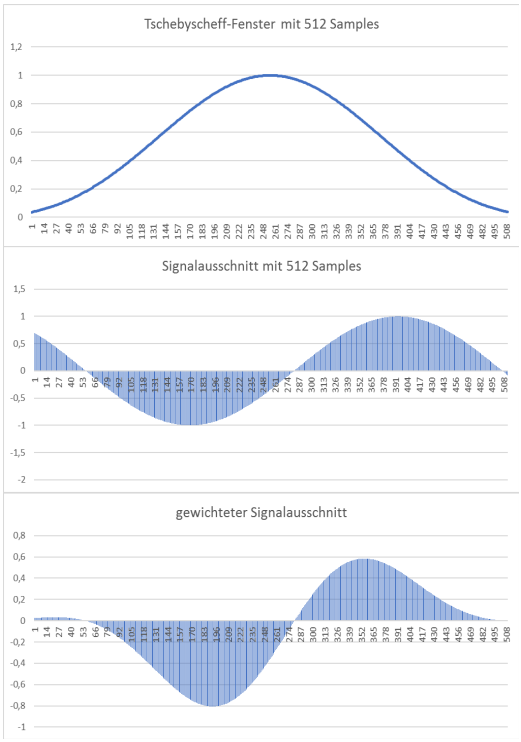


Abbildung 2.10: Tschebyscheff-Fenster im Zeitbereich

Dieses Vorgehen wäre ausreichend für, wie in den Beispielen gezeigt, periodische Signale. Echte Schallereignisse sind jedoch nicht periodisch sondern bestehen aus der Überlagerung vieler, periodischer Funktionen unterschiedlicher Frequenz und Phasenlage. Das heisst, dass sich unter Umständen gerade an den Übergängen zwischen zwei aufeinanderfolgenden DFFT des selben Signals für die vollständige Darstellung des Signals wichtige Information befindet. Würde man ein solches Signal einfach wie ein periodisches Signal Fenstern und behandeln so erhielte man ein falsches Spektrum, da die Samples an den Übergängen unterdrückt werden.

Es ergäbe sich folgendes Bild:

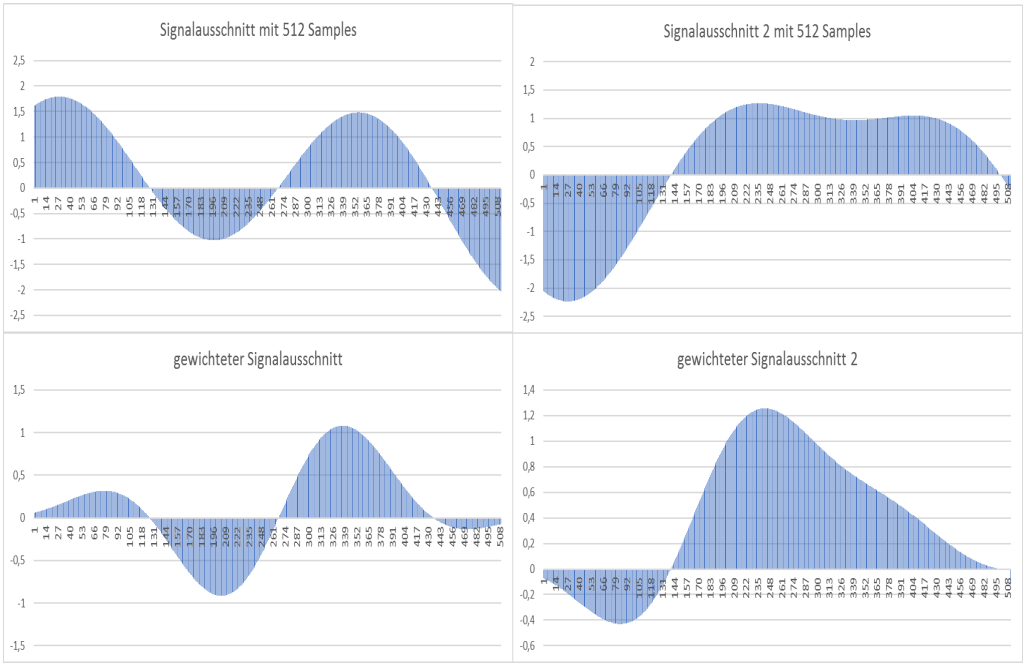


Abbildung 2.11: zwei aufeinanderfolgende Sample Ausschnitte nach Gewichtung

Wie man sieht verzerrt in diesem Fall die Gewichtung das Signal sehr stark und das resultierende Spektrum würde vom Original abweichen. Um dieses Problem einzudämmen empfiehlt es sich eine Überlappung anzuwenden. Das heisst im vorliegenden Fall, dass für die DFFT und die Gewichtung des zweiten Sample Ausschnitts nicht die Samples 1 - 512 vom zweiten Sample Ausschnitt, sondern die Samples 312 - 512 vom ersten Sample Ausschnitt und die Samples 1 - 200 vom zweiten Sample Ausschnitt benutzt werden. Durch die Symmetrie des Tschebyscheff Fensters werden so immer die Samples, die im vohergehenden Durchgang unterdrückt wurden noch einmal verwendet. Wobei nun eben diejenigen Samples die stark unterdrückt wurden eher in der Mitte des Fensters und diejenigen die weniger stark unterdrückt wurden eher am Rand des Fensters liegen.

Über die Zeit mittelt sich auf diese Art und Weise der durch das Tschebyscheff-Fenster entstandene Fehler heraus.

### 2.3.6 Rekursive Regression

Die rekursive Regression ist eine Möglichkeit im laufenden Messbetrieb neu eintreffende Messdaten dem aktuellen Mittelwert, mathematisch korrekt hinzuzufügen.

Es handelt sich dabei also um eine Methode die Gauß'sche Normalverteilung verrauschter Messdaten zu ermitteln ohne zuerst alle, zur Mittelung benötigten, Messdaten zu sammeln.[15]

Um eine rekursive Regression zu implementieren ist Kenntniss darüber nötig ob men eine konstante, eine lineare Gleichung oder eine quadratische Gleichung zu ermitteln versucht.

Wählt man hier eine Regression zu niedriger Ordnung kommt es zum sogenannten Abbruchfehler. Wählt man die Ordnung der Regression zu hoch werden Änderungen ermittelt die nicht tatsächlich exisitieren. [15]

Da im vorliegenden Projekt die rekursive Regression einer konstanten verwendet wird um um bereits detektierten Schallquellen die neuen Winkelmessdaten rekursiv hinzuzufügen, soll hier kurz der Zusammenhang zwischen der klassischen, Gauß'schen Mittelwertberechnung und der rekursiven Form erläutert werden.

Bekanntermaßen berechnet sich der Mittelwert einer Gauß'schen Normalverteilung mit n Messwerten nach der Formel:

$$\bar{x}_n = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.16}$$

Und bei einem zusätzlichen Messdurchgang

$$\bar{x}_{n+1} = \frac{\sum_{i=1}^{n+1} x_i}{n+1} \tag{2.17}$$

Durch Aufspalten der Summe im Zähler wird daraus:

$$\bar{x}_{n+1} = \frac{\sum_{i=1}^{n} x_i + x_{i+1}}{n+1} \tag{2.18}$$

Und durch Umformen und Einsetzen schliesslich

$$\bar{x}_{n+1} = \frac{\bar{x}_n \cdot n + x_{i+1}}{n+1} \tag{2.19}$$

Man hat somit eine Formel erhalten, die aussagt, dass der nächste zu erhaltende Mittelwert sich aus der Multiplikation des bereits vorhandenen Mittelwertes mit der bereits gemachten Anzahl Messungen, addiert zu dem nächsten zu erhaltenden Messwert im Verhältnis zu der neuen Anzahl Messungen ergibt

Setzt man nun n zu n-1 um den gleichen Zusammenhang für den aktuellen Mittelwert zu erhalten und formt etwas um so erhält man schlussendlich die Formel:

$$\bar{x}_n = \bar{x}_{n-1} + \frac{1}{n} \cdot (x_n - \bar{x}_{n-1}) \tag{2.20}$$

Mit der im laufenden Betrieb eintreffende Messdaten rekursiv dem bereits vorhandenen Mittelwert hinzugefügt werden können.

-

# 3 Beschreibung der Hardwarekomponenten

## 3.1 D.Module2.C6657

Das D.Module2.C6657 ist ein Entwickler-Board von DsignT auf der Basis des TMS320C6657 Dual Core 1.25 Ghz fixed and floating point CPU.[1] Das Entwickler Board ist fertig ausgerüstet mit einem 14 Pin Anschluss für die JTAG Schnittstelle eines Emulators. An der Unterseite der Platine werden alle weiteren Anschlüsse wie zum Beispiel GPIO, UART, McBsp, EMIF so herausgeführt, dass sie Problemlos auf die D2.Base Platine von DsignT aufgesetzt werden kann. Auf diese Art stehen die Anschlüsse in der standardisierten Form zur Verfügung (D-Sub, RJ....). Durch den Einsatz des Moduls mit der Base Platine kann auch das D-Module.PCM3003 einfach an das Entwickler Board angeschlossen werden.

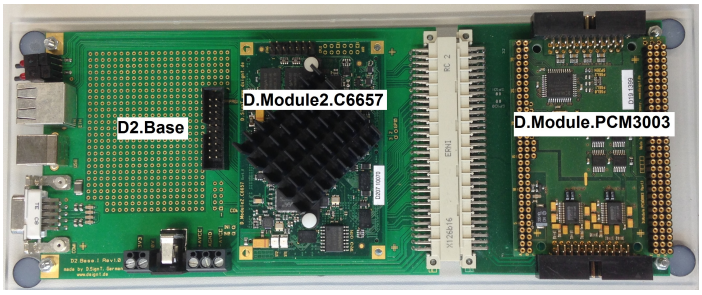


Abbildung 3.1: D.Module2.C6657 mit PCM3003 auf D2.Base Platine

## 3.2 D.Module.PCM3003

Das D.Module.PCM3003 ist ein Entwickler Board von DsignT auf Basis des PCM3003 Audio Codecs von TI. Mit den 4 vorhandenen PCM3003 Chips kann dieses Modul 8 Analoge Inputs und Outputs nahezu gleichzeitig bei bis zu 48kHz Abtastrate mit einer Auflösung von 16 Bit verarbeiten. Die Platine ist an der Unterseite mit den entsprechenden Pins ausgestattet um sie ohne Weiteres in ein System aus D.Module2.C6657 und D2.Base Platine zu integrieren. Aufgrund der Abtastrate ist dieses Modul perfekt für Audioanwendungen geeignet

DsignT empfiehlt dieses Board für Neuentwicklungen nicht weiter zu verwenden.[18]

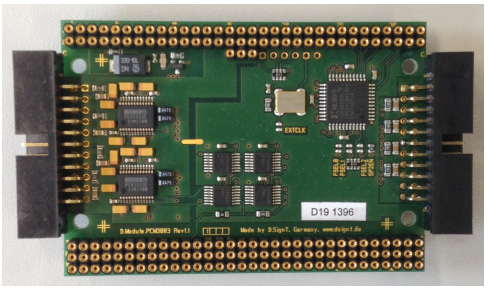


Abbildung 3.2: D.Module.PCM3003

## 3.3 8 Kanal Mikrofonverstärkerplatine

Die 8 Kanal Mikrofonverstärkerplatine dient als Schnittstelle zwischen den 8 Elektret Mikrofonen des Mikrofonarrays und den Eingängen des D.Module.PCM3003 Die dazu gehörigen 8 Mikrofonverstärker wurden von meinem Vorgänger so auf der Platine installiert, dass das Array mit einem 8 kanaligen Kabel direkt auf der Platine angeschlossen werden kann. Des Weiteren gibt es einen Anschluss für ein 26 Kanal Flachbandkabel um die verstärkten Signale an die für das Sampling verantwortliche PCM3003 Audio Codec Platine weiter reichen zu können und ein Netzteil zur Stromversorgung der Verstärker.[8]

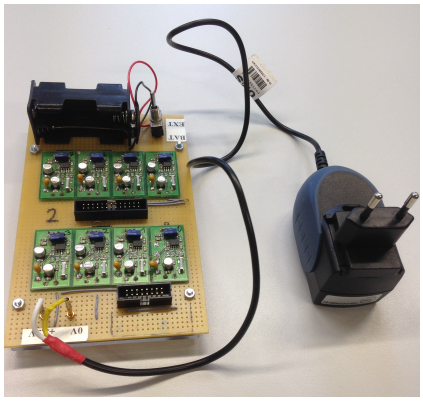

Abbildung 3.3: Mikrofonverstärkerplatine mit Netzteil

## 3.4 Zirkulares 8 Kanal Mikrofonarray

Bei dem Mikrofonarray handelt es sich um eine zirkulare Anordnung mit 9cm Durchmesser die aus 8, zirkular gleichverteilten, omnidirektionalen Elektret-Kondensator Mikrofonen besteht. Die Anordnung ist in einen Kunststoffrahmen eingebettet der auf 4, 5 Zentimeter langen, Metallfüßen steht. In der Mitte der Anordnung füllt ein Stück Teppich den Kreis aus. Die Anschlüße der Mikrofone sind so auf einen Stecker geführt, dass sie direkt auf der Mikrofonverstärkerplatine angeschlossen werden können.[8]

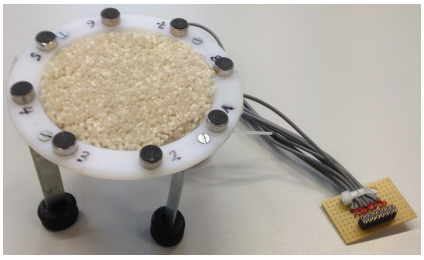

Abbildung 3.4: Mikrofonarray mit Anschlusstecker

# 4 Portierung

Die Portierung der Software auf die neue Hardware wird schrittweise durchgeführt. Zunächst wird eine funktionierende Basis–Software kreiert die mittels EDMA und in Ping Pong Buffer Technik Werte des an die beiden McBsp angeschlossenen PCM3003 in den Level 2 Speicher schreibt und sie zurück ausgibt. Anschließend werden die Schritte und Funktionen der alten Software Stück für Stück der neuen Basis Software hinzugefügt bis am Ende ein Programm entstanden ist, das auf der neuen Hardware genauso funktioniert wie das alte Programm auf der alten Hardware. Nachdem dies erreicht ist, werden diverse Einstellungen, wie Array Größen, EDMA Block Längen, Schwellwerte und vieles mehr so angepasst, dass mehr Messdaten verarbeitet werden können.

## 4.1 Basis Software

Für das Erzeugen einer funktionierenden Basis Software wurde zuerst versucht ein von Texas Instruments für den TMS320C6657 bereitgestelltes Beispielprogramm welches mit Hilfe von EDMA zunächst Werte in den Speicher schreibt und sie dann, ebenfalls mittels EDMA an die McBsp zur Ausgabe weiterreicht, so zu verändern, dass es benutzt werden kann um Werte im vom PCM3003 Audio Codec vorgegebenen Format an den McBsp auszulesen und sie von dort mittels EDMA in den Speicher zu schreiben und umgekehrt. Trotz zwei–wöchiger Bemühungen sind diese Versuche gescheitert. Glücklicherweise bekam man durch Herr Sauvagerds gute Verbindungen zu DsignT eine funktionierendes Beispielprogramm zugesendet welches genau diese Aufgabe erfüllen kann.

Alle Portierungsschritte werden also auf dem Beispielprogramm *pcm3003_edma_DM2C6657* aufgebaut, welches von Claus Hermbusche für DsignT entwickelt wurde.

### 4.1.1 Funktionsweise pcm3003_edma_DM2C6657

Um Daten vom PCM3003 empfangen zu können müssen zunächst die beiden McBsp entsprechend eingestellt werden. Das Format in dem die Daten vom PCM3003 zur Verfügung gestellt werden sieht folgendermaßen aus.

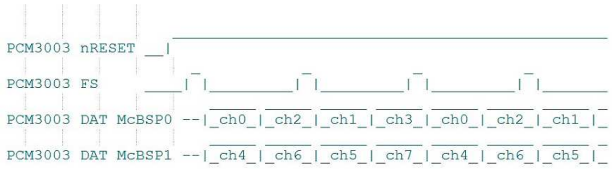


Abbildung 4.1: Datenformat PCM3003[16]

Frame und Clock Signal(hier nicht zu sehen) kommen extern vom PCM3003. In jedem Frame werden pro McBsp zwei Datenworte mit jeweils 16 Bit Little Endian übertragen. Jeder Frame besteht nur aus einer Phase, in der nacheinander 32 Bit in der selben Ordnung übertragen werden.

#### 4.1.1.1 McBsp Einstellungen

**Externes Clock Signal** Die Polarität und Quelle der beiden Clock Signale für In und Output können getrennt voneinander definiert werden. Für jedes der beiden Clock Signal existieren hierfür zwei Bits im Pin Control Register(PCR) des jeweiligen McBsp. Die beiden Bits 9 (CLKXM – Transmitter Clock) und 8 (CLKRM – Receiver Clock) definieren ob das Clock Signal vom internen Clock Generator bezogen werden soll oder von einer externen Quelle. Die beiden Bits 0 (CLKRP – Receiver Clock Polarität) und 1(CLKXM – Transmitter Clock Polarität) definieren ob bei fallender oder steigender Flanken des Clock Signals Daten übertragen werden. Um den entsprechenden Pin des McBsp als Eingang mit Empfindlichkeit auf die fallende Flanke zu konfigurieren muss in jedem dieser Bits eine 0 stehen.

Des Weiteren muss für die Receiver Clock das Bit 15 (DLB – Digital Loop Back) im Serial Port Control Register (SPCR) auf 0 gesetzt sein, da sonst der Receiver Clock Pin hoch–ohmig würde.

**Externes Frame Signal** Wie bei den Clock Signalen können auch die Frame Signal Quellen und Polaritäten sowohl für das Empfangen als auch das Ausgeben von Daten frei und getrennt voneinander Konfiguriert werden. Die beiden Bits 10 (FSRM Frame

Synchronization Receiver) und 11 (FSXM Frame Synchronization Transmitter) im PCR müssen auf 0 gesetzt werden um die McBsp für externe Frame Signale einzustellen. Die Polarität der Frame Signale wird wieder festgelegt durch die beiden Bits 2 (FSXP Frame Sync Polarität Transmitter) und 3 (FSRP Frame Sync Polarität Receiver). Durch das Setzen von Nullen in beide Bits wird verursacht, dass *Active High* für die Frame Sync Signale am Eingang gilt

**Zwei 16 Bit Datenworte pro Frame** Die Receiver-Datenwort-Anzahl und -Länge pro Frame wird im RCR (Receive Control Register) eingestellt. Um für den Receiver eine Datenwortlänge von 16 Bit festzulegen muss in die Bits 5-7 (RWDLEN1 - Receiver Wort Länge für Phase 1) der Wert 2 oder 0010 geschrieben werden. Die Anzahl der Worte pro Frame definieren die Bits 8-14 (RFRLEN1 - Receiver Frame Länge in Phase 1). Der hier festgelegte Wert entspricht genau der Frame Länge. Es muss also für zwei Worte pro Frame auch hier der Wert 2 oder 0010 eingetragen werden.

Für den Transmitter wird genauso vorgegangen. Hier müssen lediglich im XCR (Transmitter Control Register) des McBsp die entsprechenden Bits auf den gleichen Wert gesetzt werden. Die Bits haben die gleichen Adressen im XCR Register, sie heißen hier nur XWDLEN1 (Transmitter Wort Länge in Phase1) und XFRLEN1 (Transmitter Frame Länge in Phase 1)

**Bit Delay für Receiver und Transmitter** Mit den Bits 16-17 des RCR Beziehungsweise XCR kann das Data Delay für den Receiver und den Transmitter getrennt voneinander eingestellt werden. Das Data Delay kann hierbei Werte von 0 bis 2 annehmen und definiert wie viele Clock Zyklen nach dem Auftreten eines Frame Signals gewartet werden soll bevor damit begonnen wird Daten auszugeben oder einzulesen.

Folgende Grafik soll dies verdeutlichen.

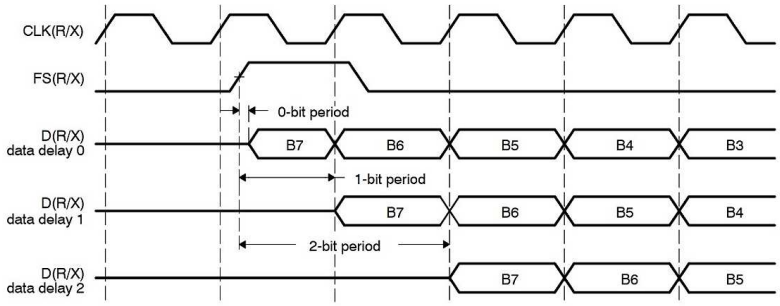


Abbildung 4.2: Bit Delay McBsp[4]

**Bit Reversal** Das Bit 4 im RCR oder XCR definiert ob 32 Bit Daten mit LSB First oder MSB First übertragen werden sollen. Laut Datenblatt haben diese Bits nur für C621x,C671x und C64x DSP einen Effekt. Im Beispielprogramm werden sie jedoch aktiv auf Null gesetzt. Vermutlich um sicher zu gehen, dass keine undefinierten Werte entstehen.

**Free Running** Mit Bit 25 im SPCR wird definiert ob sich der McBsp im Free Running Mode befindet oder nicht. Auch hier steht im Datenblatt eigentlich, dass dieses Bit nur für C621x,C671x und C64x DSP benötigt wird. Im Beispielprogramm wird es aktiv auf 1 gesetzt. Dies würde bedeuten, dass beim Emulationsstop die Serielle Clock weiterläuft. Inwiefern diese Einstellung einen Effekt hat oder nicht wurde nicht untersucht.

**Alle weiteren Register zu Null** Da weitere Einstellungen am McBsp für das Beipielsprogramm nicht benötigt werden, werden alle weiteren Register zu Null gesetzt.

#### 4.1.1.2 EDMA Einstellungen

Um den EDMA Channel Controller für die Funktion mit den McBsp in *Ping Pong Buffer Technik* einzurichten werden pro McBsp jeweils für das Einlesen und die Ausgabe der Daten drei Parametersets definiert. Es sind also pro McBsp jeweils 6 Parametersets vorhanden von denen immer drei mit dem selben DMA Channel verknüpft sind.

Ein Initial-Parameterset, ein Ping-Buffer und ein Pong-Buffer-Parameterset.

Der Einfachheit halber wird hier nur ein Parameterset für das Einlesen diskutiert. Die anderen Parametersets entsprechen in ihrem grundsätzlichen Aufbau dem Beschriebenen, nur weichen LINK Adressen, DMA Channel Adressen sowie Ziel– und Quelladressen ab, je nachdem ob es sich um Pong oder Ping Parametersets oder um Parametersets für das Ausgeben bzw. Einlesen von Daten handelt.

Desweiteren werden zwei, drei-dimensionale Arrays benötigt. Eines für das Einlesen und eines für das Ausgeben.

Die Arrays haben folgende Struktur:

$$< ArrayName > [a][b][c] \tag{4.1}$$

- a: erste Dimension kann den Wert 0 oder eins annehmen und definiert ob es sich um den Ping- oder den Pong-Buffer handelt

- b: zweite Dimension entspricht der EDMA Blocklänge. Hier werden die eintreffenden Samples abgelegt
- c: dritte Dimension definiert den Kanal, also das Mikrofon von dem die Samples eintreffen und kann Werte zwischen 0 und 7 annehmen.

**Channel Mapping** Im ersten Schritt muss der entsprechende DMA Channel mit den Parametersets verbunden werden. Dafür schreibt man in die Bits 5-13 des DCHMAP Registers, dass dem entsprechenden DMA Channel zugeordnet ist, die Adresse des gewünschten Parametersets.

**Interrupt** Als nächstes wird das Parameterset so eingerichtet, dass beim Abarbeiten eines Transfers, also bei Beendigung des Parametersets ein Bit im IPR gesetzt wird. Hierfür wird in das Bit 20 (TCINTEN) des OPT Registers des Parametersets eine eins und in die Bits 12-17 die Adresse des DMA Channels eingetragen. (Grundsätzlich wäre es nicht nötig hier die Adresse des DMA Channels zu benutzen. Jeder mögliche Wert ist denkbar und mehrere DMA Channels können Problemlos dasselbe Bit im IPR setzen und damit die selbe Interrupt Service Routine auslösen.)

**Ziel und Quelladresse** In den SRC bzw. DST Registern wird dann eingetragen welche Ziel- und Quelladressen der jeweilige Transfer verwenden soll. Für den Ping Buffer des McBsp0 Eingangsevents sind dies die Adresse des DRR0 Registers als Quell- und die Startadresse des Ping Buffer Arrays im Speicher als Zieladresse.

**Datenformat** Mit den Registern ACNT, BCNT und CCNT wird festgelegt in welchem Format die Daten in den Speicher geschrieben bzw. aus ihm gelesen werden sollen. Um die Struktur im Speicher an die Struktur der Daten vom PCM3003 anzupassen wird der Wert 2 für ACNT gesetzt und damit definiert, dass jeder ankommende Wert aus 2 Byte oder 16 Bit besteht. Da an jedem Eingang nacheinander 4 Kanäle übertragen werden, muss in BCNT der Wert 4 gesetzt werden. Im CCNT Register wird dann definiert wie viele Frames übertragen werden sollen bevor das nächste Parameterset geladen wird. Der EDMA Controller ist anschließend so eingerichtet, dass er so oft wie in CCNT definiert wurde Frames aus 4x16 Bit überträgt.

**Parameterset Link und B–Count Reload** In das Register LINK des Parametersets wird die Adresse des nächsten zu ladenden Parametersets eingetragen. Im Fall des Ping Buffers wird hier also die Adresse des Pong Buffer Parametersets eingetragen und umgekehrt. Außerdem muss in das BCNTRLD Register der Wert eingetragen, der in das BCNT Register geladen werden soll nachdem ein kompletter Frame übertragen wurde und noch nicht alle Frames abgearbeitet sind.

Letzterer Zusammenhang begründet sich darin wie die Hardware arbeitet. Im Falle eines ACNT synchronisierten Transfers, wie im vorliegenden Fall, wird nach jedem vollständig übertragenen Wert der Wert im BCNT Register so lange um eins dekrementiert bis der Wert 0 erreicht ist. Um für den nächsten Frame wieder die gleichen Einstellungen benutzen zu können wie beim Vorhergehenden muss zuerst der alte Wert wieder ins BCNT Register geladen werden. Dies geschieht automatisch aus dem BCNTRLD Registers am Ende jedes Frame. Ebenso wird am Ende jedes Frames der Wert in CCNT um eins dekrementiert bis auch hier eine 0 steht und das Parameterset abgearbeitet ist.

**Adressinkrementierung** Da die von den McBsp eintreffenden Daten zwar über zwei Kanäle eintreffen aber in einer definierten Reihenfolge ins selbe Array im Speicher geschrieben werden sollen muss besonderes Augenmerk auf die Register DSTBIDX, SRCBIDX, DSTCIDX und SRCCIDX gelegt werden.

In den xBIDX Registern wird jeweils festgelegt um wieviel Byte die Ziel- bzw Startadresse inkrementiert werden soll nachdem ein kompletter Wert übertragen wurde. Da in ACNT pro Wert 2 Byte definiert wurden, wird auch dieser Wert auf 2 Byte festgelegt. In den xCIDX Registern wird definiert um wieviel Byte die Adresse am Ende jedes Frames inkrementiert werden soll. Da immer Abwechselnd Werte vom McBsp0 und dann vom McBsp1 nacheinander im Speicher abgelegt oder ausgelesen werden sollen, wird hier der Wert 10 festgelegt.

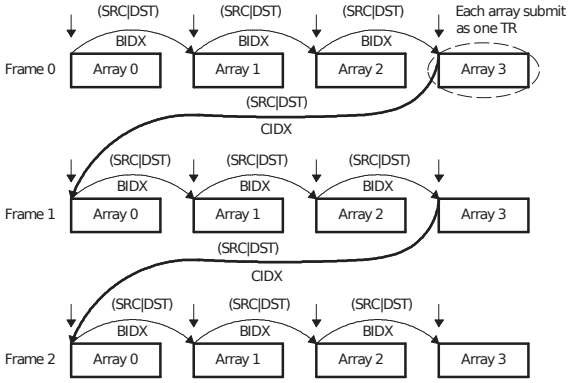


Abbildung 4.3: Darstellung Adressinkrementierung[3]

Wie man in Abbildung 4.3 sehen kann, wird am Ende des letzten zu übertragenden Wertes eines Frames der BIDX nicht noch einmal inkrementiert. Möchte man also nun zwischen zwei Frames des McBsp0 Platz im Speicher für 4x16 Bit Werte des McBsp1 freihalten muss der xCIDX Wert auf 10 gesetzt werden und nicht auf 8 wie man zuerst vermuten würde.

**Initialparameterset** Dieses Parameterset wird für jedes Event von jedem Kanal nur einmalig beim aller ersten Durchgang verwendet. Seine Einstellungen entsprechen denen des Ping Buffers. Nach dem Abarbeiten dieses Parametersets wird als nächstes das Pong Buffer Parameterset abgearbeitet. Von dort aus zeigt die Link Adresse auf den Ping Buffer.

Warum hier auf diese Weise vorgegangen wird und statt des Initial–Parametersets nicht direkt der Ping Buffer verwendet wird konnte leider nicht aus dem Code erschlossen werden.

**CPU Interrupt** Im Beispiel Programm ist kein Interrupt vom EDMA Controller an die CPU vorgesehen. Stattdessen wird am Anfang der Main Routine der Zustand der Bits im IPRH gepollt. Sind diese zu eins gesetzt worden werden sie zurückgesetzt und die Main Routine wird abgearbeitet. Hierdurch entstehen jedoch trotz des Post Processing keine Lücken in der Sampling Kette.

#### 4.1.1.3 UART

Für die UART Kommunikation sind im Beispielprojekt alle Vorkehrungen getroffen um dem Programmierer die Verwendung *Straight Forward* zur Verfügung zu stellen. Im dm2c6657 Header File des Projektes sind die hierfür benötigten Assembler Funktionen und die ihnen zugewiesenen Speicheradressen bereits definiert und können aus jeder Stelle des Codes aufgerufen werden. Die UART Schnittstellen werden beim Systemstart standardmäßig auf folgende Einstellungen Programmiert:

- BaudRate: 115200
- DataBits: 8
- StopBits: 1
- Parity: None
- HandShake: None

Zum Versenden von Daten stehen die beiden, aus C aufrufbaren Assembler Funktionen DM2_uartWrite() zum versenden einzelner char und DM2_uartWriteStr() zum Versenden von string zur Verfügung.

Die Funktion DM2_uartConfig() stünde zur Verfügung um die Schnittstellen nach den eigenen Wünschen zu Programmieren. Dies war jedoch im vorliegenden Fall nicht nötig.

Zur Verwendung einer UART Schnittstelle muss lediglich mit Hilfe der ebenfalls enthaltenen Assembler Funktion DM2_uartOpen() ein Handle zur gewünschten Schnittstelle erzeugt werden. Diese Handle wird dann benutzt um den zuerst genannten Funktionen mitzuteilen wo die Daten ausgegeben werden sollen

#### 4.1.1.4 Implementierungsreihenfolge

Um die Funktionsweise der verwendeten Komponenten sicher stellen zu können ist es ratsam bei der Implementierung eine definierte Reihenfolge einzuhalten.

Im ersten Schritt wird der PCM3003 in den Reset-Zustand versetzt. Dies geschieht durch den Aufruf der Funktion DM2_busConfig() mit dem Übergabewert 128. Danach wird das Handle zur UART Schnittstelle 0 durch den Aufruf der Funktion DM2_uartOpen mit dem Übergabe Wert 0 geöffnet.

Durch das Setzen aller Bits in den EDMA Event und Interrupt Clear Registern wird dann sichergestellt, dass keine undefinierten Werte in den EDMA Event oder Interruptregistern stehen. Dann werden alle 12 Parametersets vorbereitet und die jeweiligen DMA Channel mit ihnen verbunden.

Nach der Konfiguration des EDMA Channel Controllers wird der PCM3003 mit Hilfe der Funktion DM2_busConfig und dem Übergabewert 0 aus dem Reset Zustand genommen und anschließend die McBsp über die Funktion InitializeMcBsp mit den gewünschten Parametern konfiguriert. Hierbei ist es wichtig die McBsp weiter im Reset Zustand zu belassen. Das heißt die Bits XRST und RRST in den SPCR Registern der McBsp müssen auf 0 gesetzt bleiben.

Nun wird der PCM3003 wieder in den Reset Zustand versetzt und eine Sekunde abgewartet, bevor durch das Schreiben von Einsen an die benötigten Stellen im ICRH alle eventuell noch vorhandenen Einsen im IPRH Register gelöscht werden.

Jetzt ist sicher gestellt, dass alle beteiligten Komponenten, also PCM3003, McBsp und der EDMA Controller synchron laufen. Der PCM3003 und die McBsp können nacheinander aus dem Reset Zustand genommen werden.

An dieser Stelle muss einmalig die Funktion PCM3003_SETUP_DELAY() aufgerufen werden um eine stabile Bit Clock vom PCM3003 abzuwarten.

Diese Funktion führt, abhängig von der am PCM3003 eingestellten Samplingfrequenz, einen Loop aus NOP Befehlen aus.

#### 4.1.1.5 Hauptroutine

Im Main Loop des Beispielprogramms wird zu Beginn der Zustand des IPRH Registers überprüft. Wurde hier die Abarbeitung eines EDMA Blocks signalisiert kopiert die CPU lediglich die Werte aus dem soeben abgearbeiteten Block aus Eingangsamples an die Adresse des nächsten anstehenden Blocks für die Ausgabe.

Das so erhaltene Programm liest Werte von den McBsp aus, schreibt sie in den L2 Speicher und gibt sie dann wieder an die McBsp aus.

## 4.2 Schrittweise Portierung der vorhanden Software

Im ersten Schritt der Portierung musste der Aufbau der alten Software analysiert und verstanden werden. Da der Vorgänger eine sehr strukturierte und gut dokumentierte Software eingereicht hat konnte dies zügig erledigt werden. Es ergibt sich die im folgende vereinfachte Struktur:

Zunächst wird der aktuellste EDMA Block in ein, für komplexe Werte vorbereitetes Array kopiert. In diesem Schritt wird auch bereits der Overlap als Vorbereitung für die DFFT durchgeführt.

Dann wird das Tschebyscheff Fenster angesetzt und im selben Zug die Gesamtenergie des aktuellen Messdurchgangs berechnet. Ist diese zu klein werden die weiteren Schritte nicht durchgeführt.

Falls die Gesamtenergie groß genug war wird zuerst die DFFT durchgeführt und dann für jeden erhaltenen Frequenzbereich die Energie aller Mikrofone aufaddiert und in einem separaten Array abgelegt. Hierbei wird auch eine Filterung im Frequenzbereich durchgeführt.

Das Array wird im weiteren Verlauf benutzt um zu entscheiden welche Frequenzen für den UCA ESPRIT Algorithmus herangezogen werden. Dafür wird ein Loop solange durchlaufen wie Maxima ausreichender Höhe in diesem Array gefunden werden. Mit den Frequenzen an denen diese Maxima auftreten wird in jedem Durchgang einmal der UCA ESPRIT Algorithmus mit der Autokorrelationsmatrix eines definierten Ausschnitts des DFFT Ausgangsarray berechnet und die erhaltenen Winkel in ein weiteres Array abgelegt.

Wurden alle Maxima eines Durchgangs abgearbeitet werden die Winkel in diesem Array in einem Bereich von $10^{\circ}$ zusammengefasst, aus den Winkeln im am häufigsten auftretenden Bereich das zirkulare Mittel berechnet, per UART an den PC übermittelt und dort grafisch dargestellt.

Um während der schrittweisen Portierung überprüfen zu können ob die aktuell portierte Funktionen bei gegebenen Eingangswerten auch die richtigen Ausgangswerte erzeugt wurde das MatLab Skript *MatMulCmplx.m* geschrieben. Dieses Skript gleicht in den Durchgeführten Berechnungsschritten der tatsächlichen Software vom Eingang der DFFT bis zur Singulärwertzerlegung. Die hierfür nötigen Hilfsarrays wurden aus der Datei globals.c kopiert. Als Eingangsdaten werden die aus dem DSP Speicher in .txt files exportierten Daten aus dem DFFT Eingangsarray verwendet.

### 4.2.1 Vorbereitung des DFFT Eingangsarrays

Die Assembler Funktion die die DFFT durchführt erwartet am Eingang komplexe Werte, die Samples von den McBsp liegen jedoch nur als reelle Werte vor. Aus diesem Grund wird eine Struktur aus zwei floats definiert. Das Erste entspricht dem Realteil der Zahl, das Zweite dem Imaginärteil. Die Struktur hat die Bezeichnung *Complex* und wird in der Header Datei *math_func.h* definiert.

Für die Vorbereitung der DFFT werden zwei Arrays aus komplexwertigen Zahlen benötigt. Die Arrays haben folgende Struktur:

$$< ArrayName > [a][b] \tag{4.2}$$

- a: erste Dimension entspricht dem Kanal bzw. Mikrofon dem die Samples zugeordnet sind.
- b: zweite Dimension enstpricht der Anzahl zu speichernder Samples pro Mikrofon.

Die zweite Dimension des ersten Arrays entspricht der DFFT Länge. Die des zweiten der Länge des Overlap. Die Länge des ersten Arrays ist dabei genauso groß wie die EDMA Blocklänge und die Overlaplänge addiert.

Es werden nun nacheinander zwei Loops durchlaufen. Im ersten Loop werden für jedes Mikrofon alle Werte des aktuellen EDMA Blocks oberhalb des Overlap in das erste Array kopiert. Hierbei wird auch korrigiert, dass die im EDMA Block vorliegenden Werte nicht in der richtigen Ordnung vorliegen. (siehe Abbildung 4.1) Ein hierfür angelegter Look-Up-Table mit den korrigierten Positionen löst dieses Problem. Im weiteren Verlauf entspricht dann Kanal 0 in den entsprechenden Arrays Mikrofon 0.

Im zweiten Loop werden diejenigen Werte die aus dem vorhergehenden Durchgang im zweiten Array abgelegt wurden unterhalb des Overlap in das erste Array kopiert und anschließend die Werte die aus dem ersten Array für den Overlap im nächsten Durchgang benötigt werden ins zweite Array kopiert.

Die folgende Grafik soll dies verdeutlichen:

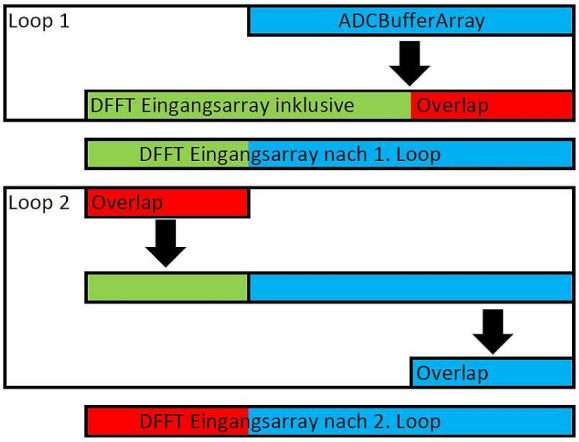


Abbildung 4.4: Visualisierung EDMA-Overlap

Dieses Vorgehen konnte für die Portierung übernommen werden. Die Arraygrößen werden während der Erweiterung angepasst.

### 4.2.2 Gesamtenergie und Fenstern im Zeitbereich

Für das Fenstern im Zeitbereich wurden Werte fest in einem Array aus floats in der Datei *globals.c* hinterlegt. Das Array entspricht in seiner Länge genau der Länge des Eingangsarrays für die DFFT. Jeder einzelne Wert in dem Array entspricht einem Gewichtungsfaktor mit dem der entsprechende Wert aus dem DFFT Eingangsarray multipliziert werden muss. Es wird also für jedes Mikrofon ein Loop durchlaufen und die beiden Arrays Elementweise miteinander multipliziert. Die erhaltenen Werte werden wieder ins DFFT Eingangsarray geschrieben. Beim Durchlaufen des Loops werden auch gleichzeitig alle Samples aller Mikrofone quadriert und aufsummiert. Aus diesen Werten wird am Ende der Durchschnitt gebildet. Dies entspricht nicht wirklich der RMS Energie der Signale, da nicht die Wurzel der Summe berechnet wird. Für die Entscheidungsschwelle ist dieser Rechenschritt aber auch nicht unbedingt nötig.

Für die Portierung mussten hier zunächst keine Änderungen vorgenommen werden. Erst während der Erweiterung des Programms wurde es nötig ein neues Tschebyscheff Fenster zu berechnen, das zu den neuen Arraygrößen passt.

### 4.2.3 DFFT

Im Vorgängerprojekt wurde zur Berechnung der DFFT die Funktion *DSPF_sp_cfftr2_dit* verwendet. Diese Funktion wird in der Datei dsp67x.lib definiert. Da das neue Projekt im ELF Format ausgegeben und in der dafür nötigen *dsplib.ae66* diese Funktion nicht mehr definiert wird, musste bereits für die Portierung ein Ersatz gefunden werden.

Es kommen zwei Möglichkeiten in Frage:

1. DSPF_sp_fftSPxSP_opt
2. DSPF_sp_fftSPxSP

Die beiden Funktionen entsprechen sich in ihrem grundsätzlichen Aufbau und bekommen alle Eingangswerte auf die gleiche Art und Weise übergeben.

#### 4.2.3.1 Untersuchung der DFFT Funktionen

Um untersuchen zu können welche der beiden Funktionen benutzt werden kann wurde eine Funktion geschrieben die ‚abhängig von der DFFT Länge, die Werte eines mit 48 kHz abgetasteten Sinus definierter Frequenz erzeugt.

Da die DFFT bei jedem Messdurchgang für 8 Kanäle durchgeführt werden muss wird als Ausgangsarray der DFFT ein zwei-dimensionales, komplexwertiges Array folgender Struktur benötigt:

$$< ArrayName > [a][b] \tag{4.3}$$

- a: erste Dimension definiert den Kanal dem die DFFT zugeordnet ist und kann Werte zwischen 0 und 7 annehmen.
- b: zweite Dimension entspricht der DFFT Länge

Die Untersuchung der beiden Funktionen wurde in zwei Schritten durchgeführt. Zunächst wurde nach der Berechnung der DFFT ein Haltepunkt gesetzt und auf einfache Art von Hand nachgeschaut ob die im Ausgangsarray der DFFT abgelegten Werte grundsätzlich realistisch sind. Anschließend wurden die Eingangswerte für die DFFT aus dem Speicher in ein Textfile exportiert und in MATLAB importiert um dort mit den gleichen Werten ebenfalls eine DFFT durchzuführen und zu überprüfen ob sich die Ergebnisse entsprechen.

Während der Untersuchung stellte sich heraus, dass die erste der beiden Funktionen nicht als vollständig funktionstüchtig betrachtet werden kann. Zwar scheint sie auf den ersten Blick durchaus stimmige Werte an den richtigen Stellen ins Ausgangsarray zu schreiben. Bei näherer Betrachtung fällt jedoch auf, dass diese Werte erstens mehr als nur leicht von den Werten abweichen die man durch die DFFT in MATLAB erhält und zweitens das Spektrum zwei überlagerten Spektren mit der halben bzw. viertel Abtastrate ähnelt.

Die zweite Funktion funktioniert jedoch wie erhofft.

Die folgenden Grafiken sollen die Beobachtungen untermauern.

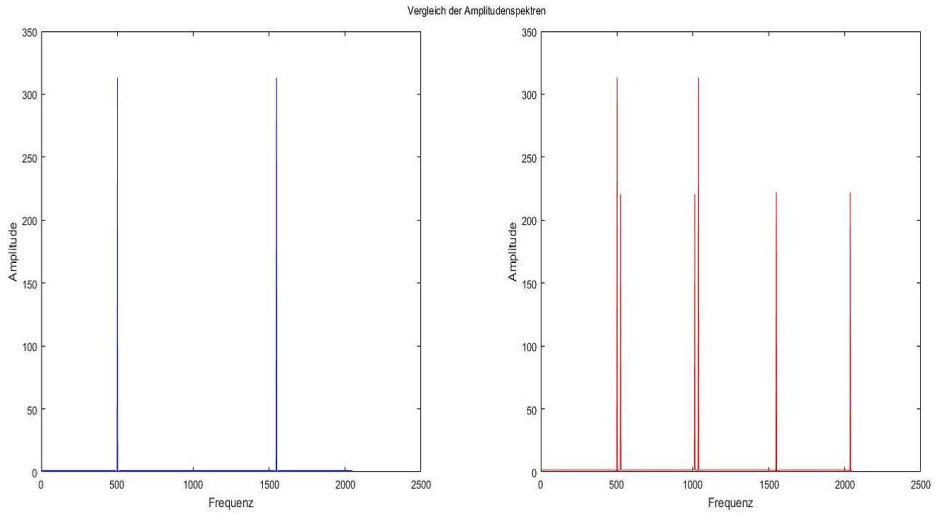


Abbildung 4.5: Vergleich der DFFT Funktionen

Wie man an der linken Grafik sieht wurde ein Sinus mit $500 \cdot 23,4375Hz = 11718,75Hz$ analysiert. Ebenfalls zu sehen ist wie sich das Spektrum mit $1024 \cdot 23,4375Hz = 24000Hz$ wiederholt. Dieses Spektrum wäre grundsätzlich in der rechten Grafik, mit unterschiedlichen Peaks, auch zu erkennen. Allerdings treten hier noch weitere Peaks auf die nichts mit dem eigentlichen Spektrum des Signals zu tun haben.

Am Ende der Untersuchung wurde entschieden auf eine Root-Cause-Analyse zu verzichten und mit der funktionierenden Funktion weiter zu arbeiten.

Der Compiler gibt zwar eine Warnung bezüglich einer implizit deklarierten Funktion. Diese kann jedoch ignoriert werden.

### 4.2.4 Filtern und vorbereiten der Maximumsuche

Im alten Projekt wurde an dieser Stelle während des Aufsummierens der Energien jeder Frequenz pro Mikrofon ein in der Datei *globals.c* hinterlegtes Hochpassfilter angewendet. Das Filter unterdrückt alle Frequenzen unterhalb von 4 kHz, so dass diese im Array für die Maximumsuche nicht mehr auftauchen und dementsprechend diese Frequenzen auch nicht mehr für den Algorithmus herangezogen werden. Da das neue Projekt für Sprachsignale verwendet werden soll und ein Transformieren des Spektrums zurück in den Zeitbereich nicht vorgesehen ist wurde entschieden, dass dieses Filter für die Portierung nicht weiter benötigt wird. Lediglich das Aufsummieren der Werte für die spätere Maximumsuche blieb erhalten.

Für die Erweiterung des Programms wird hier die Kalibrierung des Systems realisiert.

### 4.2.5 Maximumsuche

Für die Maximum-Suche in einem Array steht in beiden Bibliotheken die Funktion DSPF_sp_maxidx zur Verfügung. Eine kurze Überprüfung mit Hilfe von MATLAB ergab, dass diese auf der neuen Hardware immer noch fehlerfrei den Index eines Maximums innerhalb eines eindimensionalen Arrays zurück gibt.

Der Funktion wird einfach das für die Maximumsuche vorbereitete Array und seine Länge übergeben.

Mit Hilfe des zurückgegebenen Index kann dann die Höhe des Peaks im Maximumsuche-Array überprüft werden und entweder der UCA-ESPRIT Algorithmus durchgeführt oder, sollte das Maximum zu niedrig sein, die Suche weiterer Peaks unterbrochen werden.

### 4.2.6 Berechnung der Autokorrelationsmatrix

Für die Berechnung des UCA-ESPRIT Algorithmus muss zuerst die Autokorrelationsmatrix eines definierten Bereichs des DFFT Ausgangsarrays berechnet werden.

Um möglichst gut geeignete Bereiche identifizieren zu können wird bei der Maximusuche gefundene Index verwendet und dann aus einem Bereich +/- 2 im zwei-dimensionalen DFFT Array um diesen Index herum die Autokorrelationsmatrix gebildet.

Das Ergebnis ist eine 8x8 Autokorrelationsmatrix mit der anschliessend der UCA-ESPRIT Algorithmus durchgeführt werden kann.

### 4.2.7 Anwendung des Algorithmus

Zu Begin des Algorithmus wird an der vorbereiteten Korrelationsmatrix ein Beamforming durchgeführt. (Details zum Thema Beamforming finden sich in Herr Reermanns Arbeit). Dazu wird die Korrelationsmatrix nacheinander mit zwei konstanten, in der Datei *globals.c* hinterlegten, Matrizen multipliziert. Durch die beiden Multiplikationen wird der Realteil der Korrelationsmatrix derart gedreht und gezerrt, dass die erste Spalte des Realteils der resultierenden Matrix dafür verwendet werden kann um, mit Hilfe weiterer Transformationen, den Elevations- und Azimutwinkel einer an den Mikrofonen eintreffenden Frequenz relativ zur Mikrofonarray-Oberfläche zu berechnen. (Details zum UCA-ESPRIT- Algorithmus finden sich in Herr Reermanns Arbeit )

Als Entscheidungsschwelle ob die aktuellen Messungen geeignet zur Berechnung der Winkel sind wird hier auch eine Singulärwertzerlegung durchgeführt. Ist der erste Singulärwert um einen in der Datei *global_params.h* definierten Faktor größer als der zweite werden die Winkel berechnet im anderen Fall wird diese Messung ignoriert.

Entscheidend für die vorliegende Arbeit ist, dass für die Vorbereitung und das Durchführen des Algorithmus viele komplexe Matrix-Multiplikationen durchgeführt werden müssen.

Hierfür wurde im alten Projekt die Assembler Funktion DSPF_sp_mat_mul_cplx verwendet. Diese Funktion ist zwar in der neuen Bibliothek immer noch definiert, eine Untersuchung mit MATLAB ergab jedoch, dass die durch sie erhaltenen Werte Fehler ab der zweiten Nachkommastelle enthalten.

Dies scheint zunächst nicht sehr problematisch, führt jedoch dazu, dass sie für die Vorliegende Arbeit nicht geeignet ist.

Es ergibt sich folgendes Problem:

Vor der Berechnung der der Winkel aus der ersten Spalte der transformierten Korrelationsmatrix wurden bereits 4 Matrix-Multiplikationen an der Korrelationsmatrix durchgeführt.

Benutzt man hierfür die Assembler-Funktion und vergleicht die sich ergebende Diagonalmatrix der Singulärwertzerlegung mit Hilfe von MATLAB so fällt auf, dass die Werte mehr als nur leicht voneinander abweichen. (Die Singulärwerte sollen hier nur der Veranschaulichung dienen. Die Singulärwertzerlegung selbst wird während der Erweiterung des Systems obsolet.)

| Assembler Funktion | |
|---|---|
| SVD[W] DSP | SVD[W] MATLAB |
| 0 | 0 |
| 10,645827 | 18,3445 |
| 0,1670457 | 0,6277 |
| 0,0238699 | 0,1375 |
| 0,0146479 | 0,0243 |
| 0,0006633 | 0,0078 |
| 0,0037679 | 0,004 |
| 0 | 0,001 |

Abbildung 4.6: Vergleich Singulärtwertzerlegung DSP - MATLAB (Assembler)

Da im alten Projekt in der Datei *math_func.c* bereits eine ANSI C Funktion vorlag die komplexe Matritzen miteinander multiplizieren kann wurde der Vergleich noch einmal mit dieser Funktion durchgeführt.

| Assembler Funktion | | | ANSI C Funktion | |
|---|---|---|---|---|
| SVD[W] DSP | SVD[W] MATLAB | | SVD[W] DSP | SVD[W] MATLAB |
| 0 | 0 | | 0 | 0 |
| 10,645827 | 18,3445 | | 23,609438 | 23,6094 |
| 0,1670457 | 0,6277 | | 1,5482526 | 1,5483 |
| 0,0238699 | 0,1375 | | 0,4560311 | 0,456 |
| 0,0146479 | 0,0243 | | 0,1583973 | 0,1584 |
| 0,0006633 | 0,0078 | | 0,0323584 | 0,0324 |
| 0,0037679 | 0,004 | | 0,0027707 | 0,0181 |
| 0 | 0,001 | | 0,018071 | 0,0028 |

Abbildung 4.7: Vergleich Singulärtwertzerlegung DSP - MATLAB (ANSI C)

Da dieser Schritt dazu führte, dass das System zum ersten Mal fehlerfrei funktionierte, wurde entschieden auf die Geschwindigkeitsvorteile der Assemblerfunktionen zu verzichten und stattdessen die ANSI C Funktionen zu benutzen.

An dieser Stelle fällt noch ein weiteres Problem auf. Betrachet man die resultierenden Singulärwerte aus DSP und MATLAB genauer so sieht man, dass die letzten beiden Werte bei der Berechnung durch den DSP in der falschen Reihenfolge auftauchen. (in der Grafik rot eingerahmt)

Für die Berechnung der Singulärwertzerlegung wird die vom Vorgänger aus dem Internet geladene und angepasste Funktion *svdcmp_opt* benutzt. Anscheinend ist diese Funktion leicht fehlerbehaftet.

Weitere Untersuchungen dieses Umstandes ergaben, dass das Problem nur bei den beiden niedrigsten Singulärwerten auftritt. Da für die Entscheidungsschwelle nur die beiden größten Singulärtwerte entscheidend sind schien dies kein Problem zu sein und wurde als gegeben aber unwichtig akzeptiert.

### 4.2.8 Histogrammanalyse

Am Ende der Bearbeitung des aktuellsten EDMA Blocks wurden neue Winkelmessdaten im Histogrammarray abgelegt. In der anschließenden Histogrammanalyse wird dieses Array dann auf die am häufigsten vorkommenden Azimut Winkel untersucht. Dafür wird das Array zunächst einmal komplett durchlaufen und die Azimut Winkel mit der einer Auflösung von 10° im Winkelarray zusammengefasst. Im so entstandenen Array aus nach Größe sortierten Winkeln wird dann der Modalwert gesucht und aus allen, in einem Bereich +/- 10° um den Modalwert im Histogrammarray auftretenden Winkeln, das zirkulare Mittel berechnet. Dabei wird davon ausgegangen, dass durch die Verarbeitung der dazu gehörigen Elevationswinkel auf dieselbe Art und Weise automatisch auch für diese Winkel der passende Mittelwert entsteht. Inwiefern dies stimmt konnte mit den gegebenen Mitteln nicht untersucht werden. Diese Mittelwerte, und im Fall der Elevationswinkel auch die dazu gehörenden Maximal- und Minimalwerte werden dann im Array abgelegt, welches per UART an die GUI weitergegeben werden soll. Die Modalwertsuche wird ebenfalls solange durchgeführt, bis kein Maximum größer ist als ein vorher einstellbarer Wert. Am Ende der Berechnung jedes einzelnen gefundenen Modalwerts wird der Wert und der Bereich von +/- 10° um den Modalwert aus dem Winkelarray gelöscht und das nächste Maximum gesucht.

Der Rückgabewert der Histogrammanalyse-Funktion entspricht der Anzahl der detektierten Schallquellen. Im Vorgängerprojekt wurde hier also lediglich dafür gesorgt, dass der höchste Wert der zurück gegeben werden kann eine 1 ist. Dadurch wird immer nur die Schallquelle mit den am meisten vorkommenden Winkeln auf der GUI dargestellt, auch wenn eigentlich noch weitere Messungen außerhalb des definierten Bereichs weitere Vektoren ergeben hätten.

Ausreißer und Messunsicherheiten werden damit einfach ausgeblendet beziehungsweise ignoriert. Allerdings kann so auch keine zweite Schallquelle angezeigt werden.

### 4.2.9 Versenden der Daten per UART

Die grafische Oberfläche verarbeitet eintreffende Daten mit einem Pseudo--Zustandsautomat. Für das Versenden der Daten per UART stehen auf der Hardware zwei Arrays zur Verfügung. Mit Hilfe des ersten Arrays wird lediglich übermittelt wieviel Schallquellen gefunden wurden. Die restlich Elemente des Arrays dienen nur dazu die Übertragung dieser Information an den Zustandsautomaten anzupassen. Im zweiten Array werden nacheinenander für jede Schallquelle der Azimutwinkel, die Häufigkeit mit der er gefunden wurde sowie die resultiernden Elevationswinkel und die entsprechenden maximal und minimal Werte abgelegt und dann übertragen. Hierfür werden vor der Übertragung in einem Loop die jeweiligen Werte in das entsprechende Array eingetragen und die Arrays dann Elementeweise an den PC übertragen. Der Loop wird so häufig durchlaufen wie Schallquellen detektiert wurden. Der Pseudo-Zustandsautomat ist so aufgebaut, dass die Winkel die im vorhergehenden Durchgang in einem Buffer gespeichert wurden, vor dem erneuten Beschreiben des Buffers im aktuellen Durchgang auf der grafischen Oberfläche angezeigt werden. Auf der grafischen Oberfläche ist also eigentlich immer die Vergangenheit der laufenden Messungen zu sehen.

Alle Werte werden mit einem Offset von 40 an den PC übertragen.

Obwohl es nicht Teil der Arbeit sein soll die grafische Oberfläche näher zu beschreiben war dennoch ein oberflächliches Verständnis des Pseudo-Automaten nötig um erfolgreich Daten an die GUI übermitteln und so die Funktionsweise des Programms testen zu können.(siehe nächste Seite Abbildung 4.8)

- intAct = RS232 Data (gesendet von Hardware)
- inputBuffer.size = Anzahl empfangener Winkel (PC)
- MaxDoas = Anzahl gefundener Schallquellen (gesendet von Hardware)
- inputBuffer.add = Winkel zu Buffer hinzufügen (PC)
- handleBuffer = Buffer grafisch anzeigen (PC)
- clearBuffer = Buffer löschen (PC)

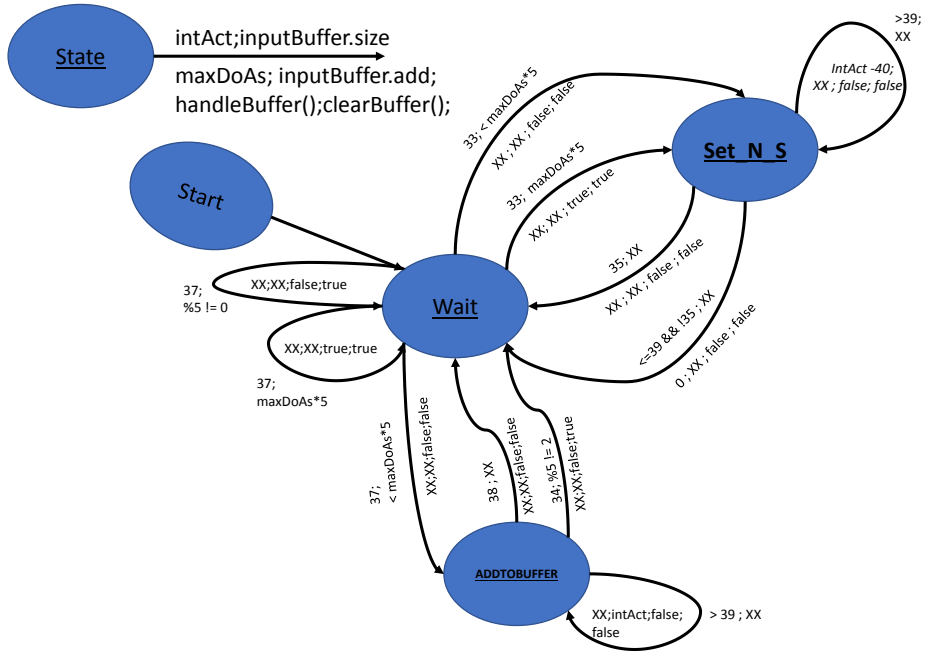


Abbildung 4.8: Pseudo-Zustandsautomat UART Kommunikation

Wie man sieht ist dies kein echter Zustandsautomat. Die Funktionen handleBuffer, clear-Buffer und inputBuffer.add stellen eigentlich eigene, diskrete Zustände dar. Hier werden sie jedoch, nur unter Umständen, beim Übergang zwischen Zuständen aufgerufen.

# 5 Erweiterung und Post-Processing

## 5.1 Erweiterung

Nachdem alle Teile der alten Software auf der neuen Hardware kompiliert werden konnten wurde erneut ein Funktionstest durchgeführt. Danach konnte damit begonnen werden die Software Stück für Stück zu Erweitern.

Der erste Teil der Erweiterung zielt darauf ab eine bessere Frequenzauflösung durch Erhöhung der DFFT Länge zu erzielen. In den weiteren Schritten wird dann untersucht welche Parameter verändert werden müssen um mehr als eine Schallquelle lokalisieren zu können.

Außerdem wird gezeigt wie durch den Einsatz von Interprozessorinterrupts eine Parallelverarbeitung erreicht werden kann.

### 5.1.1 Frequenzbereich

Wie bereits im Kapitel *Portierung* beschrieben muß der verwendete Frequenzbereich angepasst werden um auch die Lokalisierung von Sprachsignalen zu ermöglichen. Statt hierfür ein Filter zu entwerfen wurde entschieden auf das Ergebnis der DFFT während der Vorbereitung der Maximumsuche eine adaptives Rechteckfilter anzuwenden. Das Filter besteht lediglich aus einem Array, dessen Länge der des Eingangsarrays für die Maximumsuche entspricht. Da an keiner Stelle des Projekts wieder zurück in den zeitbereich transformiert werden muss schien dies ausreichend.

Das Rechteckfilter wird während der Vorbereitung der Maximumsuche mit den Ausgangsarrays der DFFT multipliziert. Auszublendende Frequenzbereiche werden hierbei mit 0 multipliziert, alle anderen mit 1.

Es wurde entschieden für Sprachsignale Frequenzen zwischen ca. 200 Hz und ca 4000 Hz zuzulassen. Hierfür wird lediglich während der Initialisierung des Programs in das Filterarray an den benötigten Stellen eine eins geschrieben. Die adaptiven Eigenschaften dieser Vorgehensweise werden im Kapitel *Kalibrierung* beschrieben.

### 5.1.2 Untersuchung des Systems und Veränderung Parametern

Bevor damit begonnen werden kann sich Gedanken über ein geeignetes Post-Processing zu machen muss das System auf seine Erweiterbarkeit um weitere Schallquellen untersucht werden. Da mein Vorgänger bereits gute Vorkehrungen hierfür geschaffen hat geht es dabei in erster Linie darum zu untersuchen wie sich die anfallenden Messdaten im HistogrammArray verändern wenn bestimmte, global in der Datei *global_params.h* definierte Parameter verändert werden.

Um dies zu untersuchen werden zwei Rauschschallquellen verwendet die mit definierten, wenn auch unbekannten, Winkeln aus zwei verschiedenen Richtungen auf das Mikrofonarray gerichtet sind. Da mein Vorgänger bereits gezeigt hat, dass der UCA-ESPRIT Algorithmus in der gegebenen Implementierung realistische Werte berechnet schien es unnötig exaktes Wissen über die verwendeten Winkel bei der Untersuchung des Systems zu haben. Das Augenmerk wurde lediglich darauf gerichtet Messdaten zu erzeugen die, bei geeignetem Post-Processing, verwendet werden können um zwei Schallquellen gleichzeitig zu detektieren.

Als Rauschquellen werden ein Laptop und ein mobiles Telefon benutzt. Beide Quellen werden auf Mono-Betrieb eingestellt um so nah wie möglich an die Eigenschaften einer Punktschallquelle heranzukommen.

#### 5.1.2.1 global_params.h

Die Header Datei global_params.h enthalt alle global benötigten Parameter für die wichtigsten Arraylängen, Schwellwerte und Schleifenzähler.

Sie wurde von Herr Reermann gut strukturiert und lässt sich in 4 Bereiche einteilen.

- Arraylängen und -adressen für EDMA und DFFT
- Arraylängen und Schwellwerte für Preprocessing
- Arraylängen und Schwellwerte für Postprocessing
- Schwellwerte und Arraylängen für den UCA-ESPRTI Algorithmus

### Arraylängen und -adressen für EDMA und DFFT

**N_CHANNEL** definiert die Anzahl Kanäle die von den McBsp verarbeiten werden. Da es sich um ein 8 Kanal Mikrofonarray handelt beträgt dieser Wert 8 und muss auch so belassen werden.

**N_BLOCK** definiert die Länge eines EDMA Blocks und hat somit indirekten Einfluss auf die Auflösung der DFFT. Dieser Wert wurde angepasst von 300 auf 1200 zur Verbesserung der Frequenzauflösung

**N_ALGO** definiert die Anzahl Eingangssamples die pro Kanal von der DFFT verarbeitet werden. Um die gewünschte Frequenzauflösung zu erhalten wurde dieser Wert von 512 auf 2048 erhöht.

**N_OVERLAP** definiert die Länge der Überlappung zweier nacheiander auftretender EDMA Blöcke zum Ausgleich der Verzerrung des Signals durch das Tschebyscheff Fenster im Zeitbereich. Da die beiden zuletzt genannten Werte vervierfacht wurden wurde auch dieser Wert vervierfacht von 212 auf 848.

### Arraylängen und Schwellwerte für Preprocessing

**N_MAG2** definiert die Länge des Array in das während der Vorbereitung der Maximumsuche die quadrierten Energien aller Mikrofone pro Frequenz aufaddiert werden. Da oberhalb der halben Abtastrate und damit oberhalb der halben DFFT-Länge Aliasing auftritt genügt es hier ein Array mit ungefähr der halben DFFT-Länge vorzubereiten. Auch hier wurde der alte Wert von 258 lediglich mit 4 multipliziert und auf 1216 festgelegt. Dies ist tatsächlich etwas kleiner als die Hälfte der neuen DFFT Länge , aber dennoch ausreichend. Vor allem, da für die vorliegende Arbeit Frequenzen oberhalb von 4687 Hz, also Adressen im Array größer 200 mit Null multipliziert werden.

**F_SW** definiert den Bereich um einen Peak im Frequenzspektrum des Signals, der für die Berechnung der Autokorrelationsmatrix als Basis für den UCA-ESPRIT Algorithmus herangezogen werden soll und damit die benötigten Hilfsarraygrößen. Der alte Wert 2 wurde beibehalten.

Zur Autokorrelation und damit auch für den UCA-ESPRIT Algorithmus wird also immer ein Bereich von +/- 46,875 Hz um einen Peak im Frequenzspektrum herangezogen.

**F_DW** definiert den Bereich um einen Peak im Ausgangsarray der Maximumsuche Vorbereitung der nach der Anwendung des Algorithmus aus dem Array gelöscht werden soll. Auch dieser Wert wurde auf 2 belassen. Aus dem Array wird also immer ein Bereich von +- 46,875 Hz gelöscht bevor nach einem neuen Peak gesucht wird.

**X_PEAK_MIN_POW2** definiert den Schwellwert den ein Peak im Ausgangsarray der Maximumsuche Vorbereitung mindestens erreichen muss um als potentiel zur Anwendung des Algorithmus geeignet klassifiziert zu werden. Er entspricht also sozusagen der Geräschempfindlichkeit des Systems. Je kleiner der Wert desto leisere Geräusche werden zur Berechnung des Algorithmus zugelassen. Der alte Wert von 8 hat sich als für Räume mit starkem Hall ungeeignet erwiesen. Er wurde auf 16 erhöht und kann eventuell auf 32 gesetzt werden falls es zu sehr lautem Hall oder Echos kommt.

**X_ENERGY_MIN** definiert den Schwellwert den die Gesamtenergie des Eingangssignals mindestens erreichen muss um die Vorbereitung der Maximumsuche einzuleiten. Er wurde auf 0,5 belassen, was bedeutet, dass die Summe aller quadrierten Samples im aktuellen DFFT Eingangsarray größer als 0,5 sein muss um weitere Schritte einzuleiten.

**N_PEAKS** definiert die maximale Anzahl Frequenzen die pro Messdurchgang zur Anwendung des UCA-ESPRIT Algorithmus herangezogen werden soll. Der alte Wert von 10 hat sich als ungeeignet für das Lokalisieren von zwei Schallquellen erwiesen und wurde auf 80 erhöht

### Arraylängen und Schwellwerte für Postprocessing

**N_HIST** definiert die Histogrammarraylänge. Dieser Wert definiert wieviele vergangene Messwerte für die Histogrammanalyse im Post-Processing vorgehalten werden sollen. Der Wert wurde von 50 auf 300 erhöht um eine Lokalisierung von zwei Schallquellen zu ermöglichen

**N_HIST_WAIT** definiert wieviele Messdurchgänge durchgeführt werden sollen bevor das Histogrammarray für die auf die am häufigsten auftretenden Winkel untersucht wird. Der Wert wurde auf 1 belassen.

Die Laufvariable mit der der Wert verglichen wird, wird mit dem Wert Null initialisiert und mit jedem Messdurchgang um eins erhöht. Ist ihr Wert größer als eins wird sie auf den Wert zwei gesetzt. Auf diese Art wird lediglich nach dem aller ersten Messdurchgang ein Durchgang abgewartet. Danach wird nach jedem Messdurchgang eine Histogrammanalyse durchgeführt.

**HIST_PEAK_MIN** definiert wie häufig ein Winkel bei der Untersuchung des Histogrammarrays vorkommen muss um ihn als potentielle, temporäre Schallquelle klassifizieren zu können. Dieser Wert wird im neuen Projekt verwendet um die Mindest-Höhe der Peaks der Autokorrelation der vergangenen 10 Winkelarrays zu definieren. Es hat sich herausgestellt, dass hierfür der Wert 60 am geeignetsten ist.

**N_HISTPEAKS** definiert wieviele Peaks pro Messdurchgang maximal im Winkelarray gefunden werden dürfen bevor die Histogrammanalyse abgeschlossen wird. Der Wert wurde von 7 auf 10 erhöht um eine Lokalisierung von zwei Schallquellen zu ermöglichen.

**HIST_ACCURACY** definiert mit welcher Auflösung die Winkel bei der Untersuchung des Histogrammarrays auf die am häufigsten auftretenden Winkel zusammengefasst werden sollen. Der Wert wurde von 10 auf 1 geändert um den Winkeln zugeordnete Frequenzen maximal-selektiv abspeichern zu können.

Die Winkel werden also nicht mehr zusammen gefasst sondern jeder einzelne für sich betrachtet.

**HIST_DW** definiert in welchem Bereich die Werte um einen Peak im Winkelarray nach der Klassifizierung des Peaks und der dazugehörenden Frequenzen gelöscht werden sollen. Abhängig von der Histogrammauflösung entspricht dies einem Bereich von:

$$HIST_{DW} * HISTACCURACY \tag{5.1}$$

Der Wert wurde von 2, bei einer Histogrammauflösung von 10 auf 1, bei einer Histogrammauflösung von 1 reduziert, so dass tatsächlich nur noch der Peak selbst und keine benachbarten Werte gelöscht werden.

Dadurch ist sicher gestellt, dass alle im Winkelarray auftretenden Peaks selektiv verarbeitet werden.

**HIST_SW** definiert in welchem Bereich um einen Peak im Winkelarray Messwerte aus dem Histogrammarray zur Mittelung herangezogen werden. Auch dieser Wert wurde von 2, bei einer Histogrammauflösung von 10 auf eins, bei einer Histogrammauflösung von 1 herab gesetzt. So ist wieder sicher gestellt, dass beim Auffinden dieser Winkel und der dazugehörenden Frequenzen im Histogrammarray nur diejenigen Winkel herangezogen werden, die tatsächlich dem Peak entsprechen.

**HIST_SIZE** definiert die Größe des Winkelarrays. Der Wert wurde so verändert, dass er von der Histogrammauflösung abhängig ist. Bei der alten Histogrammauflösung ergab sich so eine Größe von

$$\frac{360}{10} + 1 = 37 \tag{5.2}$$

Die neue Histogrammauflösung ergibt eine Größe von 360.

Die zusätzliche Position im Array des alten Projekts diente dem Vermeiden von Rundungsfehlern beim zusammenfassen der Winkel. Da im neuen Projekt die Winkel nicht mehr zusammen gefasst werden wird die zusätzliche Position nicht mehr benötigt.

**Schwellwerte und Arraylängen für den UCA-ESPRTI Algorithmus**

**N_M** definiert die zulässige Anzahl Phasenmoden. Dieser Wert wird für das Beamforming während des Anwendens des UCA-ESPRIT Algorithmus benötigt. Die Koeffizienten für das Beamforming müssen passend zu den Mikrofonarray-Ausmaßen berechnet und fest hinterlegt werden. Da bereits durch meinen Vorgänger gezeigt wurde, dass der Algorithmus in der vorliegenden Implementierung funktioniert und jede Änderung an diesem Parameter Einfluß auf die Implementierung hat wurde dieser Wert beibehalten. Nähere Informationen zum Beamforming sind Herr Reermanns Arbeit zu entnehmen

**SVD_VALUE_DIVISOR** definiert wie groß das Verhältniss zwischen den beiden größten Singulärwerten im Realteil des Beamformingausgangsarrays mindestens sein muß um die daraus resultierenden Winkel zu berechnen und im Histogrammarray abzulegen. Die Entscheidungsschwelle wird während der Erweiterung des Systems so weit herabgesetzt, dass die Berechnung der Singulärwerte obsolet wird.

**EV_ABS_MAX** Um die Elevationswinkel zu berechnen muss der Arkussinus eines sich aus dem Beamformingausgangsarray ergebenden Wertes gebildet werden. Da der Arkussinus für Werte größer 1 gegen unendlich geht wurde in der Vorgängerarbeit definiert, dass für Werte zwischen 1 und EV_ABS_MAX der Elevationswinkel 90° in Histogrammarray abgelegt wird. Der Wert wurde bei 1,2 belassen, da im Frequenzbereich von Sprache so oder so Probleme bei der Elevationswinkelgenauigkeit auftreten.

#### 5.1.2.2 DFFT Länge

Die Länge der DFFT hat direkten Einfluß auf die Frequenzauflösung des erhaltenen Spektrums. Je länger der Eingangsvektor ist desto kleiner wird der Bereich der in einem Ausgangswert des Spektrums Frequenzen zusammenfasst. Die allgemeine Formel die diesen Zusammenhang beschreibt ist:

$$Res_f = \frac{f_s}{N_{samples}} \tag{5.3}$$

Je länger also die DFFT ist, desto präziser können einzelnen Frequenzen im Spektrum voneinander unterschieden werden. Allerdings erhöht sich dadurch natürlich auch jedesmal der Rechenaufwand und damit auch die für die DFFT aufzuwendende Zeit.

In der Vorgängerarbeit wurde, aus Mangel an Rechenkapazität, lediglich mit einer DFFT Länge von 512 Samples gearbeitet. Daraus ergab sich eine Frequenzauflösung von 93,75 Hz. Da in der vorliegenden Arbeit darauf abgezielt wird Schallquellen aufgrund ihres charakteritischen Spektrums klassifiziern zu können wurde diese Auflösung als zu gering eingestuft.

Man entschied sich für eine DFFT Länge von 2048 Samples und damit für eine Frequenzauflösung von 23,4375 Hz. Die Gründe hierfür sind folgende:

1. Die niedrigste Frequenz die verarbeitet werden soll ist ca. 200 Hz für Sprachsignale. Bei dieser Auflösung kann man also bereits in der untersten, verwendeten Oktave 10 Frequenzbereiche differenzieren
2. Die EDMA Blocklänge und die DFFT Länge sind über die Overlap Länge voneinander abhängig. Durch Erhöhung der EDMA Blocklänge erhöht sich jedoch auch die Zeit zwischen zwei vollständigen EDMA Blöcken. Mit 2048 Samples DFFT Länge benötigt man, ohne weitere Vorkehrungen, eine EDMA Blocklänge von 1200 Samples. Bei einer Abtastrate von 48 kHz fallen also alle $\frac{1200}{48000\frac{1}{s}} = \frac{1}{40}s$ neue Werte an. Dies entspricht 40 Hz was bereits unterhalb der Wahrnehmungsschwelle liegt.

Durch das Anpassen der EDMA Block- und der DFFT-Länge musste auch der Überlappungsbereich angepasst und ein neues Tschebyscheff-Fenster berechnet werden. Dies konnte ganz einfach mit der MATLAB-Funktion *chebwin* erledigt werden. Die so entstandenen Werte wurde einfach in der Datei *globals.c* fest in einem Array hinterlegt. Diese Datei enthält bereits andere mit festen Werten vordefinierte Arrays und schien deshalb dafür geeignet. Die Datei enthält noch zwei weitere Tschebyscheff-Fenster-Arrays für DFFT Längen von 512 und 1024 Eingangssamples die während der Erweiterung des Systems verwendet wurden. Diese wurden auskommentiert.

#### 5.1.2.3 Winkelgenauigkeit

Wie bereits im Kapitel *Portierung/ Histogram Analyse* erwähnt, werden in der Vorgängerarbeit bei der Vorbereitung der Histogramanalyse Winkel mit einer Auflösung von +/- 10° im Winkelarray zusammengefasst. Auch dieses Vorgehen ist für eine spätere Klassifizierung der Schallquellen anhand geeigneter Frequenzen nicht mehr praktikabel. Führt es doch dazu, dass, unter Umständen, Frequenzen, die zwar in diesen Bereich fallen aber nicht von der eigentlichen Schallquelle kommen, mit in den Klassifikator aufgenommen werden.

Aus diesem Grund entschied man sich für eine Auflösung von 1°.

Ein Peak im Winkelarray bedeutet dann also, dass genau dieser Winkel mit der Häufigkeit die der Höhe des Peaks entspricht im Histogrammarray gefunden wurde.

Im weiteren Verlauf werden dann auch nur genau die diesem Winkel zugeordneten Frequenzen dem Klassifikator für die Schallquelle hinzugefügt.

#### 5.1.2.4 Histogrammlänge

Nachdem die gewünschten Änderungen an den Grundeinstellungen vorgenommen waren wurde als erster zu untersuchender Parameter die Histogrammlänge ausgewählt. Man erhofft sich davon, dass durch eine Erhöhung auch ausreichend Winkel der zweiten Schallquelle im Histogramarray anfallen um beide gleichzeitig darstellen zu können. Um feststellen zu wie sich die Histogrammlänge auf die Detektierbarkeit mehrerer Schallquellen auswirkt wurde sie Schrittweise erhöht und das resultierende Winkelarray nach 10 Messdurchgängen grafisch dargestellt.

Es ergaben sich folgende Ergebnisse:

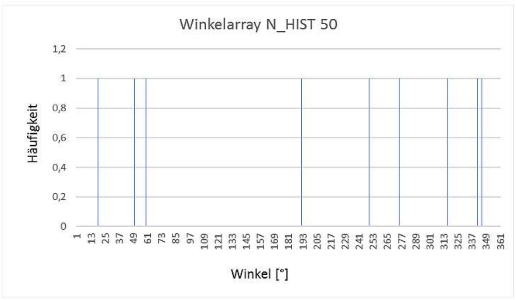


Abbildung 5.1: Winkelarray nach 10 Messdurchgängen mit N_HIST = 50

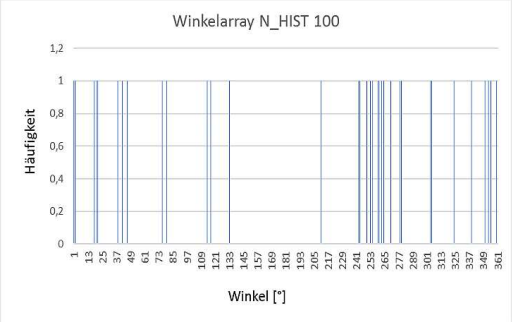


Abbildung 5.2: Winkelarray nach 10 Messdurchgängen mit N_HIST = 100

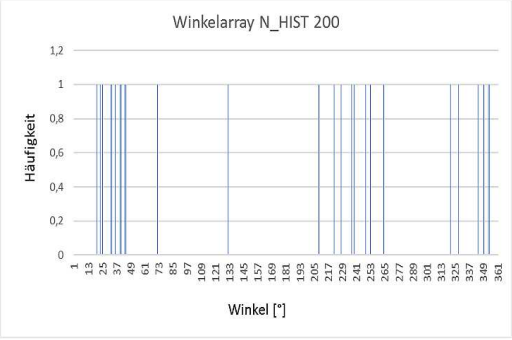


Abbildung 5.3: Winkelarray nach 10 Messdurchgängen mit N_HIST = 200

Wie man sieht, erhöht sich zwar die Auftrittsdichte bestimmter Winkelbereiche im Winkelarray durch das Erhöhen der Histogrammlänge. Da jedoch zwei Schallereignisse gleichzeitig auftreten scheinen schon während des Anwendens des UCA-ESPRIT Algorithmus nur wenige, der gefundenen Maxima als für die Berechnung geeignet eingestuft zu werden.

#### 5.1.2.5 Singulärwert-Entscheidungsschwelle

Da ein reines Verlängern der Histogrammlänge nicht zum gewünschten Ergebniss geführt hat wird als nächstes untersucht ob das Herabsetzen der Singulärwert-Entscheidungsschwelle das Auftreten verwendbarer Winkel im Histogrammarray erhöht oder ob dadurch lediglich das Rauschen verstärkt wird. Das Vorgehen ist wieder das gleiche. Der Präklassifikator wird Schrittweise verkleinert und das jeweils erhaltenen Winkelarray grafisch untersucht. Es ergeben sich folgende Darstellungen:

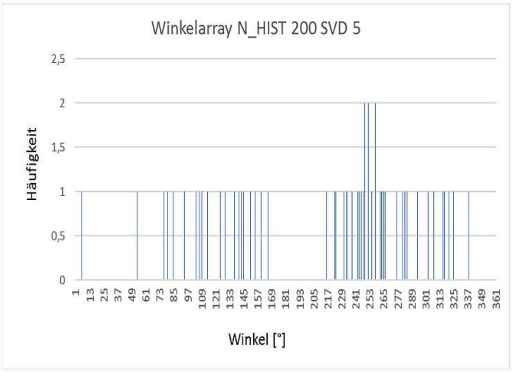


Abbildung 5.4: Winkelarray nach 10 Messdurchgängen mit N_HIST = 200 und SVD = 5

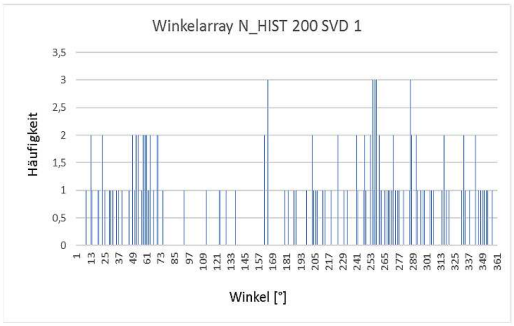


Abbildung 5.5: Winkelarray nach 10 Messdurchgängen mit N_HIST = 200 und SVD = 1

Man erkennt, dass die in der Vorgängerarbeit gewählte Einstellung des Singulärwert--Entscheidungsschwelle nicht geeignet ist wenn man mehr als eine Schallquelle gleichzeitig triangulieren will.

Lockert man diese Präklassifizierung jedoch auf erhält man zwar mehr Rauschen aber ebenfalls eine deutliche Erhöhung der Detektierbarkeit zweier Schallquellen gleichzeitig.

Dennoch sind auch so noch nicht ausreichend deutliche Peaks im Winkelarray entstanden um eine Aussage darüber treffen zu können, was davon Ausreisser und was davon verwertbare Messungen sind.

An dieser Stelle lässt sich bereits absehen, dass man, möchte man mehr als eine Schallquelle gleichzeitig detektieren, eine größere Messungenauigkeit in Kauf nehmen muss. Desweiteren fällt auf, dass das Durchführen der Singulärwertzerlegung zur Prüfung ob

der größte Singulärwert kleiner ist als der zweitgrößte nicht viel Sinn macht. Falls sie nicht genau gleich groß sein sollten werden die Winkel mit der aktuellen Einstellung immer berechnet. Die Singulärwertzerlegung ist damit obsolet.

#### 5.1.2.6 Maximale Anzahl zu findender Peaks im Spektrum

Aus zeitlichen Gründen wurde bei der Vorgängerarbeit definiert, dass pro Messdurchgang mit maximal 20 Peaks im Spektrum der UCA-ESPRIT Algorithmus durchgeführt werden darf. Bei gegebener Systemtaktung und EDMA-Block-Länge blieb nicht mehr Zeit für den rechenintensiven Algorithmus.

Da die Ergebnisse des Verlängerns der Histogrammlänge und des Anpassens des Singulärwert-Entscheidungsschwelle darauf hindeuten, dass lediglich mehr Messungen vorhanden sein müssen um in den Daten Aussreisser, Rauschen und Messungen voneinander zu unterscheiden wird als nächstes untersucht inwiefern einer Erhöhung dieses Parameters die Detektierbarkeit erhöht. Das Winkelarray wird nun bereits nach 5 Messdurchgängen untersucht, da durch die Veränderung dieses Parameters natürlich pro Messdurchgang potentiel doppelt bzw. viermal soviele Winkel anfallen. Es entstehen folgende Grafiken:

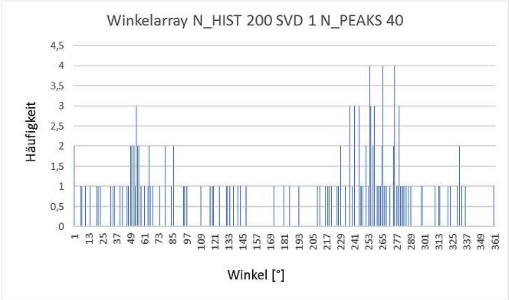


Abbildung 5.6: Winkelarray nach 5 Messdurchgängen mit N_HIST = 200, SVD = 1 und N_PEAKS = 40

Die Grafiken zeigen deutlich, dass bei einer Erhöhung der maximalen Anzahl zu findender Peaks auf den Wert 80 tatsächlich noch einmal einer Erhöhung der Detektierbarkeit erreicht werden kann. Eine weitere Erhöhung macht jedoch keinen Sinn, da man hierdurch zwar die Auftrittsdichte der zu findenden Winkelbereiche erhöht, im selben Maß allerdings auch die Peaks der Außreisser und das Grundrauschen höher werden.

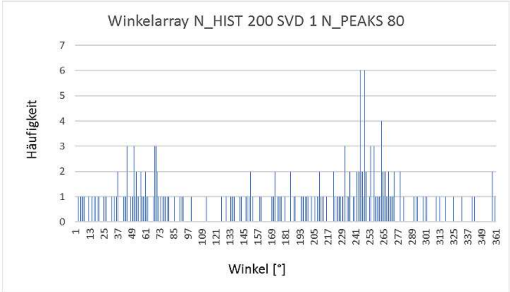


Abbildung 5.7: Winkelarray nach 5 Messdurchgängen mit N_HIST = 200, SVD = 1 und N_PEAKS = 80

Betrachtet man die Ergebnisse der Untersuchung genauer, so fällt auf, dass bei der letzten der beiden Untersuchungen nach 5 Messungen tatsächlich bereits 200 verwertbare Winkel im Winkelarray abgelegt wurden.

Aus diesem Grund wird an dieser Stelle noch ein letztes mal die Histogrammlänge auf 300 erhöht ohne die anderen Parameter zu verändern.

Dadurch ergibt sich folgende Darstellung:

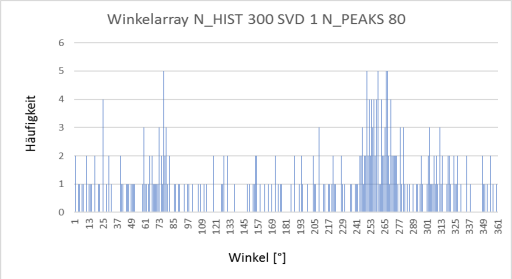


Abbildung 5.8: Winkelarray nach 5 Messdurchgängen mit N_HIST = 300, SVD = 1 und N_PEAKS = 80

Anscheinend kann nun durch eine weiter Erhöhung der Histogrammlänge tatsächlich noch einmal ein Vorteil gewonnen werden. Diesen erkauft man sich jedoch mit steigender Trägheit des Systems. Es wurde entschieden, dass diese Einstellungen als gute Kompromisslösung zwischen schneller Reaktionszeit und hoher Detektierbarkeit gesehen werden kann.

#### 5.1.2.7 Anzahl und Höhe zu findender Maxima im Winkelarray

Da mein Vorgänger sich für eine nicht-selektive Bearbeitung des Winkelarrays entschieden hatte war, vor allem nach der Singulärwert-Entscheidungsschwelle während des UCA-ESPRIT Algorithmus, auf diesen Wert kein besonderes Augenmerk zu legen. Es genügte ein Maximum zu finden und alle Winkel um das Maximum herum zur Mittelung heranzuziehen. Wie bereits erwähnt ist dieses Vorgehen für eine Klassifizierung anhand des charakteristischen Spektrums während des Post-Processing nicht geeignet. Um das gewünschte Ergebnis zu erhalten sollen lediglich die Peaks, selektiv für die Mittelung und Klassifizierung herangezogen werden.

Um herauszubekommen wieviele der bei laufenden Messungen im Winkelarray auftretenden Peaks jeweils einer Schallquelle zugeordnet werden können werden 30 Messdurchgänge durchgeführt und erneut das resultierende Winkelarray grafisch dargestellt. Da hierbei das Histogrammarray bei gegebener Histogrammlänge mit Sicherheit mehrfach komplett mit Werten vollgeschrieben wurde kann man davon ausgehen das eingeschwungene System zu betrachten.

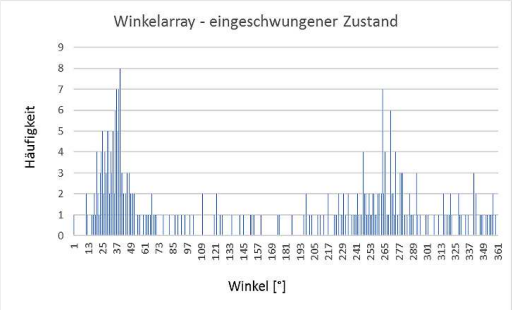


Abbildung 5.9: Winkelarray im eingeschwungenen Zustand

Wie man sieht wäre es für das lokalisieren zweier Schallquellen nötig bis zu 20 Peaks mit einer Höhe von 3 zuzulassen. Dieser Umstand begründet sich in der Art und Weise wie die Maximumsuche durchgeführt wird und soll an dieser Stelle anhand eines Beispiels erklärt werden.

Wenn man die Peaks in Abbildung 5.9 in der Reihenfolge duchnummeriert in der sie von der Maximumsuche aufgefunden werden ergibt sich folgendes Bild: (Aus Gründen der Übersichtlichkeit werden nur die relevanten Bereiche dargestellt)

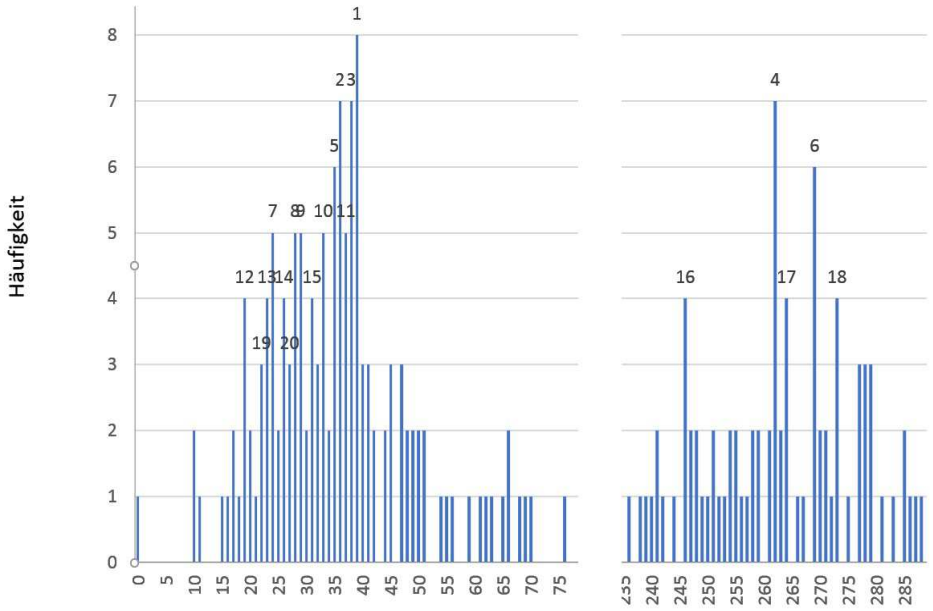


Abbildung 5.10: Reihenfolge der Maximumbearbeitung

Ebenfalls bereits zu erkennen ist, dass es bei dieser Vorgehensweise während des Post-Processing nötig sein wird Aussreisser zu identifizieren und die gefundenene Peaks den jeweiligen Schallquellen zuzuordnen.

Im weiteren Verlauf der Arbeit wurde festgestellt, dass am wenigsten falsch Messungen auftreten, wenn zunächst maximal 10 Peaks im Winkelarray gesucht werden und erst wenn sicher zwei Schallquellen detektiert wurden den Wert auf 20 zu erhöhen. Zusätzlich wurde festgestellt, dass deutlich weniger Aussreisser auftreten wenn statt mit den Winkelarrays selbst mit den autokorrelierten Werten aus mehreren Messdurchgängen gearbeitet wird. Der Schwellwert wird also, wie bereits erwähnt, an die höheren Werte der Autokorrelation angepasst.

#### 5.1.2.8 Frequenzabhängigkeit der Elevationswinkelgenauigkeit

Während der Untersuchung des Systems fiel auf, dass die Elevationswinkel nie kleiner als 45 Grad werden. Eine Untersuchung dieses Umstandes ergab eine Frequenzabhängigkeit der Elevationswinkeldetektierbarkeit die vermutlich mit den Ausmaßen des Mikrofonarrays zu tun hat.

Untersucht man das Histogrammarray der Elevationswinkel im eingeschwungenen Zustand mit Rauschquellen hoher und tiefer Frequenzen bei konstant flachem Einfallswinkel auf das Array ergibt sich folgendes Bild:

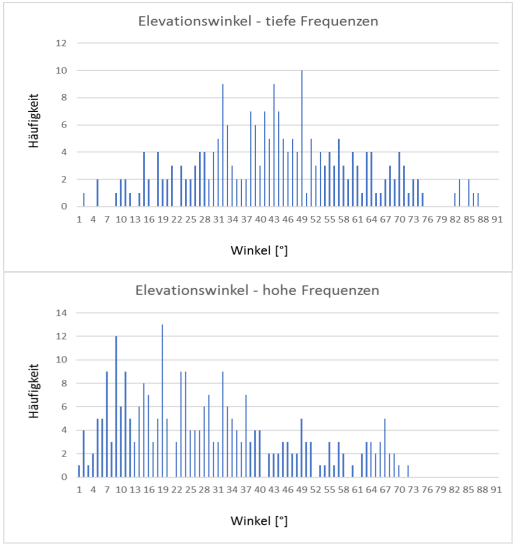


Abbildung 5.11: Elevationswinkel bei hohen und tiefen Frequenzen

Wie man sieht lassen sich bei hohen Frequenzen durchaus auch Elevationswinkel kleiner 45° lokalisieren. Bei einer Rauschquelle tieferer Frequenzen ist dies nicht möglich.

Theoretisch sollte sich hier bei genauerer Untersuchung folgendes Ergeben:

$$d_{array} = 0,09m \tag{5.4}$$

$$V_{schall} = 340\frac{m}{s} \tag{5.5}$$

$$f = \frac{V_{schall}}{\lambda} \tag{5.6}$$

$$f_{min} = \frac{340\frac{m}{s}}{0,09m} = 3777,78Hz \tag{5.7}$$

Da sich Schallwellen deren Wellenlänge größer oder gleich der Ausmaße eines Körpers sind um den Körper herum beugen. Es wird vermutet, dass dieser physikalische Zusammenhang zu den beobachteten Ergebnissen führt und erklärt warum mein Vorgänger sich für den gewählten Frequenzbereich oberhalb 4 kHz entschieden hatte.

Leider war es mit den gegebenen Mitteln nicht möglich diese Theorie näher zu untersuchen oder zu untermauern. Allerdings deckt sich die Beobachtung ziemlich genau mit den theoretischen Überlegungen zum räumlichen Alias-Effekt in der Arbeit meines Vorgängers.

#### 5.1.2.9 Galvanische Trennung vom Netz

Bereits zu Beginn der Arbeit an dem Projekt fiel auf, dass die Mikrofonverstärker Platine Netz- und Signalseitig ohne weitere Abschirmungsvorkehrungen angeschlossen ist. Im weiteren Verlauf des Projekts fiel dann auf, dass selbst in einem stillen Raum gelegentlich Vektoren berechnet und angezeigt werden. Diese stammen oft von Frequenzen die sich aus dem Netz in den Signalweg gestreut haben und vom System selbstverständlich nicht von Schallereignissen unterschieden werden können. Zunächst wurde hierfür die Kalibrierung des Systems eingeführt (siehe Kapitel *Post-Processing/Kalibrierung*).

In sehr ungünstigen Fällen kann dies jedoch nicht ausreichen.

Benutzt man das System zum Beispiel an einem Stromkreis an dem viele, induktive 230V/12V Transformatoren angeschlossen sind ergibt sich in einem stillen Raum folgendes Spektrum:

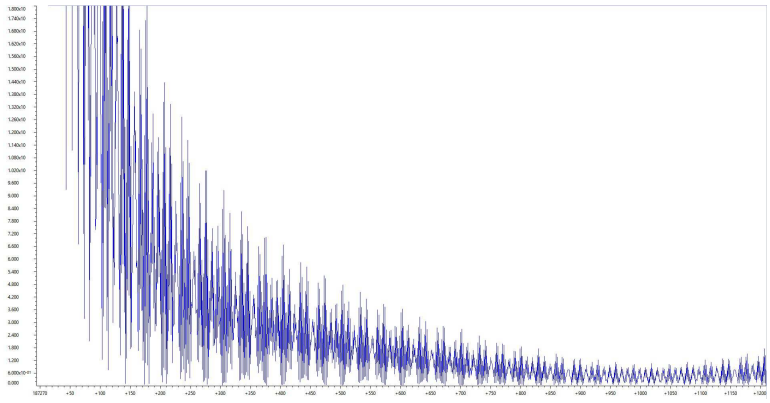

Abbildung 5.12: Ausgangsspektrum der DFFT bei starken Störungen aus dem Netz

Durch das Anschliessen eines 400W Schalt-Netzteils an den selben Stromkreis ergibt sich, ohne weitere Änderungen, dann dieses Bild:

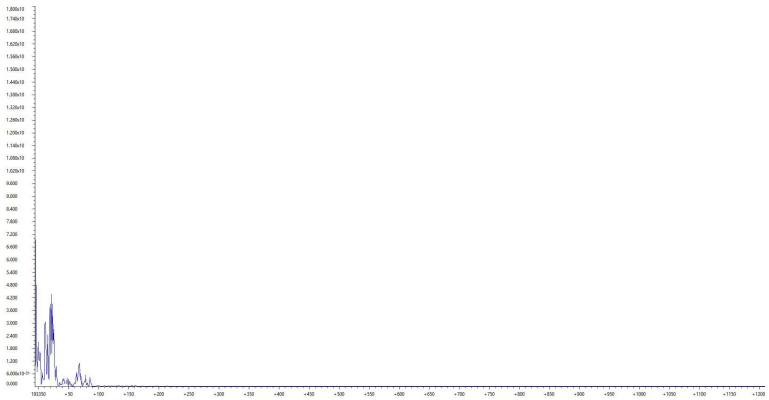

Abbildung 5.13: Ausgangsspektrum der DFFT mit „Blindleistungskompensation“

Ohne auf die Zusammenhänge eingehen zu müssen die diese Ergebnisse verursachen wird doch klar, dass es nötig ist die Mikrofone galvanisch vom Netz zu trennen um den Einfluss von Störfrequenzen zu unterbinden.

## 5.2 Post-Processing

Nachdem alle Einstellungen so geändert wurden, dass zumindest potentiel ausreichend häufig Winkel von zwei Schallquellen im Histogrammarray auftreten, konnte damit begonnen werden sich Gedanken über ein geeignetes Post-Processing zu machen. Folgende Probleme galt es dabei zu beseitigen:

- Grundgeräusche von zum Beispiel Rechnern oder ähnlichem bzw. Störfrequenzen aus dem Netz verfälschen die Messung
- Durch die veränderten Einstellungen ist weniger Zeit am Ende jedes Messdurchgangs für ein Post-Processing übrig
- In nicht-schalltoten Räumen wird es immer zu Hall und Echos kommen
- Selektives Verarbeiten der Peaks im Winkelarray macht eine Klassifizierung nötig
- Selektives Verarbeiten der Peaks im Winkelarray sorgt für weniger Messwerte für die Mittelung pro Messdurchgang

### 5.2.1 Kalibrierung

Um Probleme durch Störfrequenzen oder -Geräusche einzudämmen wurde ein Timer eingerichtet der beim Start des Systems zu zählen beginnt und jede Sekunde einen Interrupt auslöst. Solange dieser Timer aktiv ist wird das Histogrammarray zwar nach jedem Messdurchgang auf resultierende Winkel untersucht, die dazu gehörende Frequenzen werden jedoch im Frequenzfensterarray zu null gesetzt. Wurde der Interrupt 30 mal ausgelöst wird der Timer ausgeschaltet und die Kalibrierung des Systems ist vorbei.

Während der Kalibrierung sollten, außer der Störgeräusche, kein Signal an den Mikrofonen ankommen. Außerdem sollte nach der Kalibrierung überprüft werden wieviele Frequenzen tatsächlich nicht mit 0 multipliziert werden. Unter Umständen waren die Störeinflüsse so groß, dass das System nicht mehr funktionieren kann.

Sollten sehr viele Frequenzen ausgeblendet werden obwohl es im Raum eigentlich ruhig war liegt dies vermutlich an den Störfrequenzen aus dem Netz. Ein Blick auf das DFFT Ausgangsarray bei einem zweiten Versuch bringt hierbei Klarheit.

### 5.2.2 Dual Core Parallelverarbeitung

Bei der Untersuchung des Systems auf das zeitliche Verhalten mit dem System eigenen Timer fällt auf, dass, bereits ohne die Histogrammanalyse mit den neuen Einstellungen im ungünstigsten Fall 0,023 Sekunden vergehen bis alle gefundenen Peaks durch den UCA-ESPRIT Algorithmus verarbeitet wurden. Da bei gegebener EDMA Blocklänge alle 0,025 Sekunden neue Messwerte anfallen könnte dies eventuel dazu führen, dass Messdaten verloren gehen. Um dieses Problem von vornherein zu umgehen macht man es sich zu Nutze, dass es sich beim TMS320C6657 um einen Dual Core System handelt.

Hierfür wurde ein Programm für die zweite CPU vorbereitet, dass lediglich aus einer leeren Dauerschleife und einer Interprozessor Interrupt Routine besteht. Das Programm basiert auf dem Beispielprogramm *dual_core1*[19], dass von A.Klemenz für DSignT entwickelt wurde und im D.Module2.C6657 Software Paket enthalten ist

Am Ende eines Messdurchgangs kopiert die erste CPU das komplette Histogrammarray aus dem eigenen Speicher in den geteilten Speicher, indem ein, beim Start des Programms mit definierter Adresse im geteilten Speicher erzeugter, Pointer, innerhalb einer Schleife nach jedem Kopiervorgang um einen Adresswert erhöht wird.

Am Ende der Schleife setzt die erste CPU dann das entsprechende Bit im IPCGR, signalisiert damit der zweiten CPU, dass Daten zur Abholung zur Verfügung stehen und setzt danach die Pointeradresse zurück auf den Anfangswert für den nächsten Durchgang

In der Interruptroutine der zweiten CPU wird dann zunächst auf die selbe Art und Weise das Histogrammarray in den eigenen Speicher kopiert, anschliessend die Histogrammanalyse durchgeführt, der gleitende Mittelwert der resultiernden Winkel berrechnet und die so erhaltenen Winkel per UART an den PC übermittelt.

In dieser Zeit kann die erste CPU parallel neue Daten verarbeiten.

Es hat sich gezeigt, dass dieses Vorgehen am besten funktioniert wenn man eine bestimmte Reihenfolge beim Start des Systems einhält

- Verbinden der ersten CPU mit dem Emulator
- Verbinden der zweiten CPU mit dem Emulator
- Reset auf beiden CPU durchführen
- pcm3003_edma auf CPU1 laden
- dual_core1 auf CPU2 laden

- Haltepunkt in Timer Interrupt Routine von CPU1 so setzen, dass die CPU am Ende der Kalibrierung hier anhält
- Wurde der Haltepunkt erreicht CPU2 starten, danach CPU1 Starten

Durch die Aufteilung der Aufgaben auf zwei CPU mussten die Dateien *algo_func.c* und *global_params.h* so aufgeteilt werden, dass nur noch die für die jeweilige CPU wichtigen Werte und Funktionen vorliegen. Bei zukünftigen Untersuchungen des Codes muss dies beachtet werden.

### 5.2.3 Histogrammanalyse

Betrachtet man das Winkelarray im eingeschwungenen Zustand so fällt auf, dass in einem bestimmten Bereich um die wahren Winkel der beiden Schallquellen vermehrt Peaks einer bestimmten Höhe anfallen.

Bei mehrfacher Untersuchung hintereinander fällt dann auf, dass Hall, Echo und Messausreisser zwar mit ähnlich hohen Peaks auftreten können. Die Auftrittsdichte dieser Peaks allerdings deutlich geringer ist.

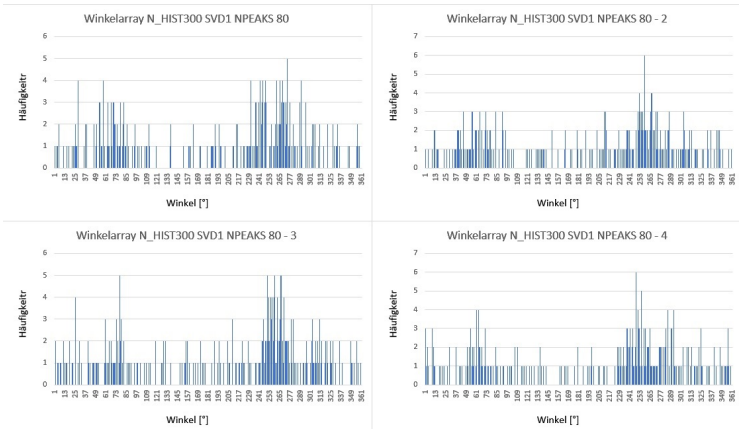


Abbildung 5.14: Winkelarray bei konstanten Schallquellen zu verschiedenen Zeitpunkten

Dies wird noch deutlicher bei Betrachtung der Hauptdiagonale der Autokorrelationsmatrix aus den gezeigten vier Messungen.

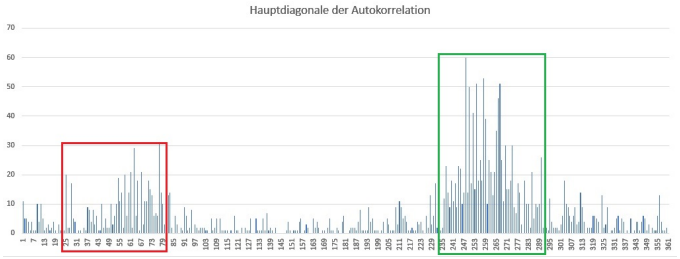


Abbildung 5.15: Autokorrelation der vier Messdurchgänge

Verwertbare Winkel scheinen also immer in Rahmen von ca. 60° aufzutauchen ( in der Abbildung farblich markiert ). Ausreißer treten zwar auf, im Verlauf mehrerer Messungen ist ihr Auftreten jedoch nicht mit den vorhergehenden Messungen korreliert.

Für die Histogrammanalyse wurden sich diese zwei Eigenschaften des durch zwei Schallquellen resultierenden Winkelarrays zu nutze gemacht.

#### 5.2.3.1 Autokorrelation der Winkelarrays

Zur Stabilisierung der Messwerte und zum Vermeiden von Aussreissern ist es von Vorteil die Histogrammergebnisse mehrerer Messdurchgänge miteinander zu korrelieren. Da bei einer Frequenzauflösung von 1 insgesamt 360 Winkel miteinander korreliert werden müssen, was unter Umständen hohen Rechenaufwand bedeutet, wird zunächst untersucht wieviele Messdurchgänge mindestens korreliert werden sollten um stabile Messergebnisse zu erhalten. Es ergeben sich folgende Grafiken:

Wie man sieht erhält man bei der Korrelation der letzten 10 Messdurchgänge miteinander bereits relativ aussgeprägte Peaks und deutlich geringere Aussreisser. Eine weitere Erhöhung würde dies zwar noch verbessern, da man hierfür jedoch bereits $10 \cdot 360$ Additionen und Multiplikationen durchführen muss wurde entschieden, dass so eine ausreichend hohe Aussagekraft erreicht ist.

#### 5.2.3.2 Höhe der Histogrammaxima

In der Vorgängerarbeit entsprach dieser Wert der Mindesthöhe des Modalwertes im Winkelarray nach der Analyse des Histogrammarrays am Ende eines Messdurchgangs. Da

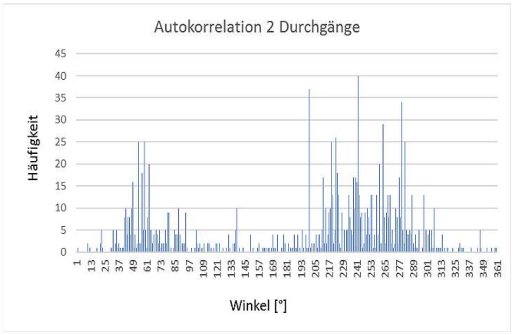


Abbildung 5.16: Autokorrelation 2 Messdurchgänge

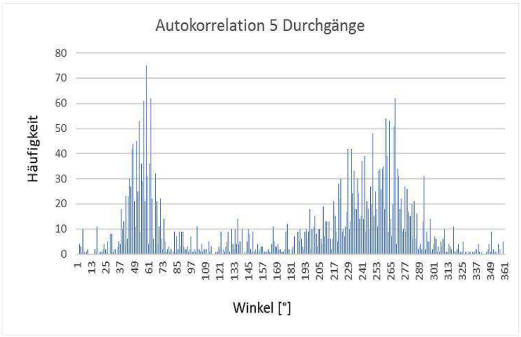


Abbildung 5.17: Autokorrelation 5 Messdurchgänge

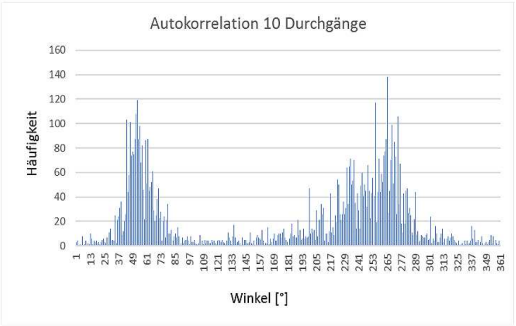


Abbildung 5.18: Autokorrelation 10 Messdurchgänge

im vorliegenden Projekt nicht mehr das jeweilige Winkelarray selbst sondern die autokorrelierten Werte aus 10 Messdurchgängen auf ihren Modalwert untersucht werden, musste dieser Wert entsprechend angepasst werden.

Betrachtet man Abbildung 5.18 so sieht man, dass eine Peakhöhe von 60 gut geeignet sein dürfte um echte Messwerte von Aussreissern zu unterscheiden.

#### 5.2.3.3 Selektive Maximaverarbeitung

Wie bereits eingehend erwähnt sollen während des neuen Post-Processing lediglich diejenigen Winkel zur Mittelung herangezogen werden die tatsächlich häufig genug vorkommen. So soll eine spätere Klassifizierung anhand charakteristischer Frequenzen möglich sein. Abbildung 5.10 kann dies verdeutlichen.

Im vorliegenden Fall kann man also 20 Winkel finden die Häufig genug im Winkelarray vorkommen. ( es wären noch weitere möglich. Aber die Maximale Anzahl zu findender Peaks wurde auf 20 beschränkt ) 15 dieser Winkel gehören zur ersten Schallquelle, 5 dieser Winkel gehören zur zweiten.

Da nicht zwangsläufig zuerst alle, der ersten Schallquelle zugeordneten Winkel und dann die die der zweiten Schallquelle zugeordnet werden können, aufgefunden werden ist es nötig zu unterscheiden ob es sich bei dem gefundenen Maximum um eine neue Schallquelle, eine bereits bekannte Schallquelle oder einen Ausreisser handelt.

#### 5.2.3.4 Speichern geeigneter Frequenzen

Zur späteren Klassifizierung der Schallquellen ist es nötig die charakteristischen Frequenzen abzuspeichern. Um diese Idee umzusetzen wurde das Histogrammarray so erweitert, dass zusätzlich zu den resultierenden Winkeln auch die Frequenzen abgelegt werden die zur Berechnung des entsprechenden Winkels geführt haben.

Werden nun während der Histogrammanalyse die entsprechenden Winkel einer bestimmten Schallquelle zugeordnet, wird diese Frequenz und die Häufigkeit mit der sie im Kontext dieser Schallquelle aufgetreten ist in einem Klassifikator abgespeichert.

Man entschied sich dafür pro Schallquelle bis zu 24 Frequenzen abzuspeichern. Dieser Wert ist beliebig gewählt und ist sicher ein Parameter dessen Einfluss noch untersucht werden kann.

#### 5.2.3.5 Unterscheidung zwischen echten Audioquellen und Ausreißern

Obwohl auch Messaussreisser oder Hall- und Echoeffekte ausreichend hohe Peaks im Winkelarray erzeugen können unterscheiden sie sich von echten Schallquellen dennoch durch die Häufigkeit mit der diese Winkel in $60^\circ$ Grad Rahmen während eines Messdurchgangs im Winkelarray auftreten. Desweiteren werden im Verlauf mehrerer Messungen hintereinander echte Schallquellen immer wieder im gleichen Rahmen erscheinen, während Hall- und Echoeffekte im Verlauf mehrerer Messdurchgänge unkorreliert sind.

Um das Auftreten von Aussreissern also noch weiter einzudämmen entschied man sich dazu, zusätzlich zur Autokorrelation der letzten 10 Messdurchgänge auch eine händische Unterscheidung zwischen Audioquellen und Außreissern durchzuführen.

Händisch heisst an dieser Stelle, dass die Histogrammanalyse um Funktionen und, sowohl globale als auch lokale, Variablen erweitert wird, die es ermöglichen die auftretenden Peaks während laufender Messungen so zu sortieren, klassifizieren und zu verarbeiten wie es ein Mensch machen würde der nacheinander die laufenden Winkelarrays grafisch dargestellt bekommt.

Die Idee dabei ist es alle Winkel die mit ausreichender Häufigkeit im Histogrammarray aufgefunden wurden und die dazu gehörenden Frequenzen zunächst in definierten Bereichen temporären Schallquellen zuzuordnen. Diejenigen temporären Schallquellen denen am Ende des Messdurchgangs die meisten Peaks zugeordnet wurden werden dann global abgespeichert und im nächsten Messdurchgang überprüft ob ihnen erneut ausreichend viele Peaks zugeordnet werden konnten.

Ist dies der Fall werden die dieser Schallquelle zugeordneten Azimut- und Elevationswinkel an den PC übermittelt. Falls nicht wird mit den neu detektierten temporären Schallquellen genauso Verfahren.

Es hat sich gezeigt, dass die wenigsten Aussreisser entstehen wenn pro Messdurchgang nur eine neue Schallquelle hinzukommen kann. Treten also zwei Schallquellen auf werden mindestens 3 Messdurchgänge benötigt bis sie beide Dargestellt werden.

1. Schallquelle 1 wurde sicher, temporär detektiert und global abgespeichert
2. Schallquelle 1 wurde detektiert und wird dargestellt. Schallquelle 2 wurde sicher, temporär detektiert und global abgespeichert
3. Schallquelle 1 und Schallquelle 2 wurden detektiert und werden dargestellt.

Es vergehen also mindestens 0,075 Sekunden bis beide Schallquellen dargestellt werden.

#### 5.2.3.6 Klassifizierung der Audioquelle

**Klassifizierung anhand der Winkelgenauigkeit**

Ganz zu Beginn hat das System keinerlei Wissen über die Schallquellen. Um geeignete Frequenzen für die spätere Klassizizierung zu finden muss zuerst eine Art Präklassifizierung statt finden.

Diese Präklassifzierung kann nur über Winkelbereiche geschehen. Bei gegebener Systemungenauigkeit kann davon ausgegangen werden, dass die einer Schallquelle zugeordneten Winkel in einem bestimmten Bereich auftreten. Wurde also ein Peak im Winkelarray gefunden wird überprüft ob er in einem definierten Bereich um einen bereits vorher gefundenen Winkel liegt und dementsprechend klassifiziert.

Um diese Präklassifizieruzng umzusetzen wurde festgelegt, dass, falls ein Peak im Bereich +/- 45° um einen bereits gefundenen Winkel liegt, er dieser Schallquelle zugeordnet werden kann. Liegt der Wert im Bereich von +/- 45° bis +/- 60° um einen bereits gefundenen Winkel wird er als Aussreisser klassifiziert und ignoriert. Liegt der Wert ausserhalb der +/- 60° Grenze wird er als neue Schallquelle klassifiziert.

Unter diesen Gesichtspunkten wird bei jedem Messdurchgang das Winkelarray und das Histogrammarray untersucht und die anfallenden Winkel, unter Umständen, rekursiv miteinander gemittelt

**Klassifizierung anhand geeigneter Frequenzen**

Die im vorhergehenden Unter-Kapitel beschriebene Vorgehensweise funktioniert zwar gut um zwei Schallquellen voneinander zu Unterscheiden. Diese müssen jedoch mindestens 45° voneinander entfernt sein um eine Vermischung der Messdaten zu verhindern. Um im weiteren Verlauf zu ermöglichen, dass sich die Schallquellen einander weiter annähern können wurde darauf abgezielt sie aufgrund ihres charakteristischen Spektrums klassifizieren zu können.

Wurden beide Schallquellen lang genug Präklassifiziert und die jeweiligen Frequenzen sowie ihre Auftrittshäufigkeit in ihrem Klassifikator abgelegt wird versucht sie Anhand dieser Information zu unterscheiden.

Dafür werden die Frequenzen die dem aktuellen Peak im Winkelarray zugeordnet sind auf ihre Auftrittshäufigkeit in beiden echten Klassifikatoren untersucht. Sind die Frequenzen in einem der Klassifikatoren doppelt so häufig aufgetreten und der Winkel nicht weiter als 45° vom Winkel der entsprechenden Schallquelle entfernt so wird ihr dieser Winkel rekursiv hinzugefügt.

#### 5.2.3.7 Dynamische, rekursive Regression

Da durch die selektive Maximaverarbeitung nicht bei jedem Messdurchgang zwangsläufig ausreichend viele Winkel anfallen um einen Mittelwert mit kleiner Abweichung zu berechnen entschied man sich dafür für bei bereits eindeutig detektierten Schallquellen neu anfallende Messwerte rekursiv dem bereits vorhandenen Mittelwert hinzuzufügen.

Da im vorliegenden Fall in der Regel Schallquellen mit konstanter Position lokalisiert werden sollen, diese sich jedoch durchaus auch bewegen können wurde entschieden die Regression einer konstanten mit dynamischer Wichtung zu implementieren.

Betrachetet man die Formel zur rekursiven Regression eines konstanten Wertes:

$$\bar{x}_n = \bar{x}_{n-1} + \frac{1}{n} \cdot (x - \bar{x}_{n-1}) \tag{5.8}$$

So fällt auf, dass mit steigender Messdauer und damit steigendem n der Einfluss neuer Messwerte auf den Mittelwert immer kleiner wird. Dies würde dazu führen, dass die Bewegung einer Schallquelle, die zunächst still stand, nicht oder nur sehr schwach in den Mittelwert einfließen würde.

Um dieses Problem zu umgehen wird der Gewichtungsfaktor dynamisch an die Höhe der Abweichungen der Messwerte vom akutellen Wert angepasst. Ist ein Messwert innerhalb

eines Bereichs von +/- 10° um den aktuellen Mittelwert so wird der Gewichtungsfaktor um eins erhöht. Ist der Messwert einmal ausserhalb dieses Bereichs wird dies zunächst als Aussreisser akzeptiert, geschieht es mehr als zweimal wird der Gewichtungsfaktor wieder um eins verkleinert. Hierbei wird auch dafür Sorge getragen, dass der Gewichtungsfaktor nicht größer als 100 und nicht kleiner als 2 werden kann.

Das System stellt den Gewichtungsfaktor so von selbst auf seine Unsicherheit ein. Eine nähere Betrachtung des Verhaltens zeigte, dass sich bei zwei Schallquellen konstanter Position die Gewichtungsfaktoren auf ca. 30 einstellen. Die Messwerte haben auf den Mittelwert also ungefähr einen Einfluss der $\frac{1}{30}$ seiner Differenz zum aktuellen Mittelwert entspricht.

Durch dieses Vorgehen kann die Lokalisierung der Schallquellen bereits stark stabilisiert werden. Allerdings reagiert das System hierdurch selbstverständlich auch träger, handelt es sich doch um ein integrierendes Filter.

#### 5.2.3.8 Gleitende Mittelwertbildung

Um die Darstellung der ermittelten Werte noch etwas stabiler darstellen zu können werden zusätzlich immer die letzten 10 ermittelten Werte gleitend gemittelt bevor der daraus resultierende Mittelwert an den PC übermittelt wird.

Dazu werden sie einfach in einem Array abgelegt. Bevor die Winkel dann an den PC übertragen werden werden alle in diesem Array befindlichen Werte aufaddiert und durch ihre Anzahl geteilt. Ist das Array voll wird es erneut von vorn befüllt und die alten Werte nach und nach überschrieben.

#### 5.2.3.9 Unterschiede zwischen Rausch und Sprachsignalen

Ein wichtiger Unterschied zwischen Sprachsignalen un Rauschen ist die Dynamik. Während wir sprechen geht unseres Stimmlage auf und ab (zum Beipsiel hoch am Ende einer Frage) und wir machen Pausen um Luft zu holen oder einfach nur aus Gründen der Verständlichkeit.(eine Pause bedeutet hier auch nur Bruchteile von Sekunden)

Dies führt dazu, dass beim Lokalisieren von Sprechern nicht unbedingt bei jedem Messdurchgang wieder neue Werte anfallen, obwohl er noch nicht zu ende gesprochen hat. Sollte jedoch aufgrund einer Pause sein bisheriger Klassifikator und sein bisheriger Mittwelwert verworfen werden so muss nach der Pause, die wie gesagt nur Bruchteile von Sekunden lang sein muss, neu mit der Mittelung begonnen und ein neuer Klassifikator erzeugt werden.

Um dieses Problem zu umgehen wurde ein globaler Indikator eingeführt der jedesmal wenn ein Peak im Winkelarray einer Schallquelle während eines Messdurchgangs zugeordnet werden kann um eins erhöht wird. Gleichzeitig wird er aber auch zu Anfang jedes Messdurchgangs um eins verkleinert. Wird also während eines Messdurchgangs die Schallquelle nicht detektiert, im Messdurchgang davor aber 3 mal, hat dieser Indikator am Ende des Messdurchgangs immer noch den Wert 2 und der Wert vom vorherigen Messdurchgang wird nocheinmal übermittelt.

Es wurde dafür gesorgt, dass dieser Indikator nicht größer werden kann als 10. Sollte ein Sprecher also tatsächlich aufhören zu sprechen dauert es maximal 250 ms bis er nicht mehr angezeigt wird.

## 5.3 Realisierung

Um die die Histogrammanalyse auf die beschriebene Art zu erweitern waren folgende Änderungen nötig

### 5.3.1 Zusätzliche Arrays und Variablen

#### 5.3.1.1 global

- int SourceCount: dient dem Abspeichern der Anzahl echter gefundener Schallquellen und wird als Rückgabewert der Histogrammanalyse verwendet. Dieser Rückgabewert definiert wieviel Winkel an den PC übermittelt werden sollen.
- int SourceCountTemp: dient dem Abspeichern der Anzahl echter temporärer Schallquellen
- int SampleCount[2]: dient dem Abspeichern der Gewichtungsfaktoren für die rekursive Regression der Schallquellen
- short SourceIndicator[2]: Indikator für die Häufigkeit mit der eine echte Schallquelle innerhalb mehrerer Messdurchgänge gefunden wurde. Dient dem ausgleich der Dynamik von Sprechern.
- int OutlierCount[2]: Ist der Antagonist zu SampleCount. Wurde ein Winkel ausserhalb der 10° Grad Grenze gemessen wird zunächst dieser Wert um 1 erhöht und erst beim dritten mal SampleCount um 1 erniedrigt.

- float Classifier[2][25[2]: Klassifikatorarray zur klassifizierung der Schallquellen. Erste Dimension entspricht der Schallquelle. In der zweiten Dimension wird der erste Wert zum Abspeichern des aktuellen Winkelmittelwerts benutzt. Die restlichen zum Abspeichern der charakteristischen Frequenzen. Die dritte Dimension speichert die Informationen über die Häufigkeit mit der die entsprechende Frequenz der Schallquelle zugeordnet werden konnte.
- int histPhi[11][360]: das Winkelarray wurde um 10 Elemente erweitert um die Autokorrelation der letzten 10 Messdurchgänge realisieren zu können
- AutoCorrCount: die Laufvariable zum Ansprechen der richtigen Positionen im Autokorrelationsarray.

#### 5.3.1.2 lokal

- short Multiple: dient der dynamischen Anpassung der Anzahl zu findender Peaks im Winkelarray an die Anzahl der gefundenen Schallquellen
- float ClassifierTemp[10][25[2]: entspricht dem Aufbau des globalen Array Classifier. Wird aber nur für die temporären Schallquellen, also zur Unterscheidung von temporären Schallquellen während eines Messdurchgangs, verwendet.
- short SourceIndicatorTemp[10]: Dient der Identifizierung der am häufigsten aufgetretenen temporären Schallquelle. Wird mit jedem Peak im Winkelarray der einer temporären Schallquelle zugeordnet wurde um 1 erhöht. Diejenige Schallquelle mit dem höchsten SourceIndicatorTemp Wert wird am Ende des Messdurchgangs, unter der Vorraussetzung, dass nicht bereits zwei Schallquellen bekannt sind, als potentielle, echte Schallquelle global abgespeichert.
- int sim: dient dem Abspeichern des Rückgabewerts der Klassifizierungen.
- float Classifier_l[25][2]: lokaler Klassifikator. Gleicht ebenfalls dem globalen Array Classifier. Hier werden zunächst alle Frequenzen und ihre Häufigkeit während des Auffindens des Winkels im Histogrammarray der dem aktuellen Peak im Winkelarray entspricht abgespeichert. Ausserdem wird hier auch der Winkel hinterlegt. Anschliessend wird geprüft ob er einer temporären Schallquelle oder einer echten Schallquelle hinzugefügt werden kann oder eine ganz neue temporäre Schallquelle definiert. Vor dem Auffinden des nächsten Peaks im Winkelarray werden alle Elemente des Array auf 0 gesetzt um es für den nächsten Peak vorzubereiten.

### 5.3.2 Zusätzliche Funktionen

#### 5.3.2.1 CircularDistance

Sowohl die Präklassifizierung als auch die dynamische, rekursive Regression machen es nötig die zirkulare Distanz zweier Winkel zu berechnen. Da dies an mehreren Stellen des Codes geschieht machte es Sinn eine Funktion zu schreiben, die den Abstand zwischen zwei Winkeln abhängig vom Quadranten in dem sich beide befinden richtig zurück gibt.

Dabei ist zu beachten, dass die übergabe der Azimut Winkel an die Funktion in der Form -180° - 180° geschieht.

Das Flow-Chart soll den Ablauf der Funktion verdeutlichen:

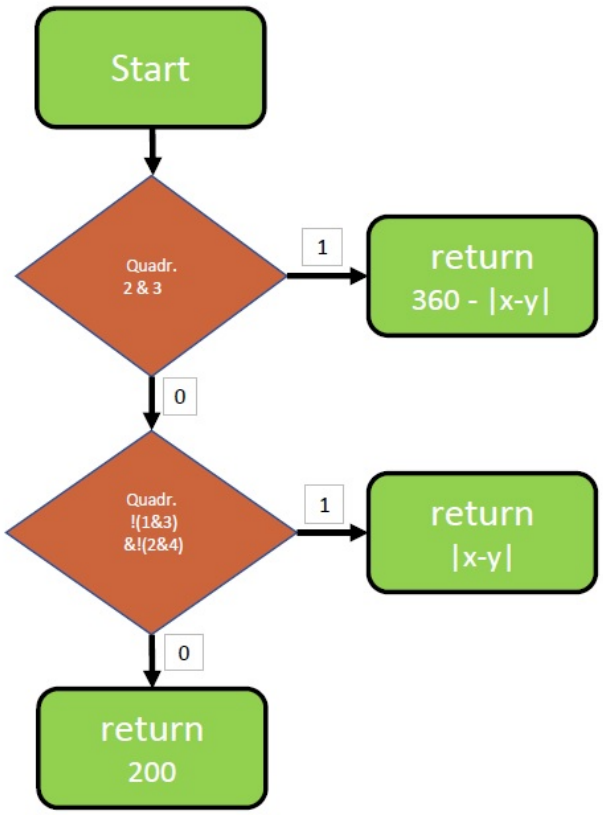


Abbildung 5.19: Flow Chart Circular Distance

Wie man sieht wurde die Vorgehensweise so vereinfacht, dass, falls sich die beiden Winkel in zwei im Einheitskreis diagonal gegenüber liegenden Quadranten befinden,

keine Berechnung durchgeführt sondern einfach der Wert 200 zurück gegeben wird. Diese Winkel wären grundsätzlich für jede Abfrage die in dieser Arbeit benutzt wird zu weit auseinander. Der genaue Wert interessiert in diesem Fall nicht.

#### 5.3.2.2 RecursiveMean

Die Funktion *RecursiveMean* dient der Berechnung der rekursiven Regression der 4 Winkel die einer Schallquelle zugeordnet sind.

Ihr werden als Call-by-Pointer die aktuellen Mittelwerte, die neuen Messwerte und, als Call-by-Value die Variable *sim* zur Definition der Schallquelle übergeben.

Die Variable *sim* entspricht hierbei immer dem Rückgabewert der Funktionen *SimCheck* bzw. *SimCheckTemp*

*ResursiveMean* passt die globalen Variablen *SourceIndicator, SampleCount* und *OutlierCount* an, berechnet dann das neue rekursive Mittel der entsprechenden Winkel und legt die neuen Mittelwerte sowohl im Klassifikatorarray der enstprechenden Schallquelle als auch an der richtigen Stelle im Array zur späteren Übermittlung an den PC ab.

Zu beachten ist, dass der aktuelle Azimutwinkelmesswert für die Verrechnung mit dem aktuellen Azimutwinkelmittelwert vorbereitet werden muss falls sich einer der beiden in Quadrant 2 und der andere in Quadrant 3 befindet.

In diesem Fall muss, entsprechend des Vorzeichens des Azimutwinkelmittelwerts entweder 360 zum Azimutwinkelmesswert hinzuaddiert oder abgezogen werden.

$$\phi^*_{corr} = sgn(\bar{\phi}) * 360 + \phi^* \tag{5.9}$$

#### 5.3.2.3 setClassifier

Mit Hilfe der Funktion *setClassifier* werden während des Auffindens der zum Peak im Winkelarray gehörenden Winkel im Histogrammarray die den Winkeln zugeordneten Frequenzen im lokalen Klassifikator abgespeichert.

Falls nicht bereits 24 unterschiedliche Frequenzen im lokalen Klassifikator abgelegt wurden wird er mit einer Schleife durchlaufen die abgleicht ob die aktuell aufgefundene Frequenz im Bereich +– 4*23,4375 Hz um eine der bereits abgelegten Frequenzen liegt. Ist dies der Fall wird die Häufigkeit der Frequenz, also die zweite Dimension der aktuellen Position im Klassifikatorarray, um 1 erhöht. Wird keine ähnliche Frequenz gefunden, wird die aktuelle Frequenz an der freien Position abgelegt und die dazugehörige Häufigkeit um 1 erhöht.

#### 5.3.2.4 SimCheckTemp

Diese Funktion dient der Präklassifizierung der echten sowie der temporären Schallquellen untereinander. Zur Fallunterscheidung werden der Funktion zusätzlich zu den entsprechenden Klassifikatoren entweder der Wert 2 oder der Wert 10 übergeben. Dieser Wert entspricht der Anzahl an Klassifikatoren mit denen der lokale Klassifikator verglichen werden muss. ( bis zu 10 temporäre Schallquellen. Maximal 2 echte Schallquellen)

Wurden alle einem Peak im Winkelarray zugeordneten Winkel und Frequenzen im Histogrammarray gefunden und im lokalen Klassifikator abgespeichert wird mit ihrer Hilfe überprüft ob der entsprechende Winkel in einem Bereich fällt der bereits einer echten Schallquelle zugeordnet ist.

Zunächst wird überprüft ob der aktuelle Azimutwinkelmesswert von einem der bereits bekannten Azimutwinkel maximal 45° entfernt ist. Ist dies der Fall werden die in den Klassifikatoren abgelegten Frequenzen miteinander verglichen. Liegt eine Frequenz im lokalen Klassifikator im Bereich +- 4*23,4375 Hz wird im entsprechenden temporären bzw. echten Klassifikator die Häufigkeit dieser Frequenz im 1 erhöht. Handelt es sich um eine Frequenz die noch nicht im nicht-lokalen Klassifikator abgelegt wurde wird sie ihm hinzugefügt und ihre Häufigkeit auf 1 gesetzt.

Sind alle Frequenzen überprüft und gegebenenfalls ausgetauscht gibt die Funktion einen Zahlenwert zurück, welcher der Position der zugehörigen Schallquelle im entsprechenden nicht-lokalen Klassifikatorarray plus 1 entspricht.

Ist der aktuelle Azimutwinkelmesswert im Bereich von 45° bis 60° um einen bereits bekannten Azimutwinkel gibt die Funktion den Wert -1 zurück um zu signalisieren, dass der Wert verworfen werden soll.

Ist zu keiner der bekannten, temporären oder echten, Schallquellen eine Ähnlichkeit gefunden worden, wird der Wert 0 zurückgegeben und damit signalisiert, dass es sich um eine noch unbekannte, temporäre Schallquelle handelt.

#### 5.3.2.5 exchangeClassifier

Sollte der lokale Klassifikator keiner bereits bekannten Schallquelle ähneln wird durch den Aufruf dieser Funktion der lokale Klassifikator in die letzte freie Stelle im Array für die temporären Klassifikatoren kopiert

#### 5.3.2.6 SimCheck

Sind zwei echte Schallquellen bekannt und ihre Klassifikatoren enthalten 24 Frequenzen die insgesamt 5000 mal gefunden wurden wird, statt den aktuellen lokalen Klassifikator mit Hilfe von SimCheckTemp mit den beiden Klassifikatoren abzugleichen, die Funktion SimCheck aufgerufen.

Diese Funktion vergleicht die Frequenzen des lokalen Klassifikators auf ihre Auftrittshäufigkeit in beiden echten Klassifikatoren. Sind die Frequenzen doppelt so häufig in einem der beiden echten Klassifikatoren aufgetaucht wie im anderen und der aktuelle Azimutwinkelmesswert maximal 45° von dem entsprechenden Azimutwinkelmittelwert entfernt, gibt die Funktion die Dimension der entsprechenden Schallquelle im echten Klassifikatorarray plus 1 zurück.

Ist der Unterschied in der Auftrittshäufigkeit geringer wird -1 zurückgegeben.

Die Vorgehensweise funktioniert für Rauschen. Leider gelingt es auf diese Art nicht Sprachsignale dauerhaft zu klassifizieren.

#### 5.3.2.7 Abschliessende Analyse der gefundenen Schallquellen

Wurden alle zulässigen Peaks im aktuellen Winkelarray klassifiziert und die variablen SourceIndicator und SourceIndicatorTemp entsprechend angepasst wird untersucht wieviele echte oder temporäre Schallquellen gefunden wurden.

Zunächst wird überprüft ob die beiden SourceIndicator Werte größer sind als 2 und somit innerhalb der letzten Messdurchgänge beiden echten Schallquellen im Schnitt ausreichend Peaks zugeordnet werden konnten. Gilt dies für beide Werte so bekommt SourceCount den Wert 2 und signalisiert damit, daß je zwei Azimut- und Elevationswinkel mit den entsprechenden Elevationswinkelgrenzen an den PC übermittelt werden sollen.

Gilt dies nur für den ersten der beiden Werte bekommt SourceCount den Wert 1 und alle Winkel, Frequenzen und Gewichtungsfaktoren die der zweiten echten Schallquelle zugeordnet waren werden zu null gesetzt.

Gilt die Abfrage nur für die zweite der beiden Schallquellen werden alle Winkel, Frequenzen und Gewichtungsfaktoren der zweiten Schallquelle an die Positionen der ersten Schallquelle kopiert und die der zweiten gelöscht. Auch in diesem Fall wird SourceCount zu 1 gesetzt.

Ist keine der bisher bekannten Schallquellen ausreichend oft detektiert worden werden alle genannten Werte auf null gesetzt und SourceCount bekommt den Wert 0.

Ist am Ende dieser Abfragen SourceCount kleiner als zwei, wurde also maximal eine echte Schallquelle sicher lokalisiert und soll dargestellt werden, wird überprüft ob es temporäre Schallquellen gibt, die ausreichend häufig detektiert wurden.

Hierfür wird das SourceIndicatorTemp-Array auf sein Maximum durchsucht. Ist das gefundene Maximum größer als 3 werden die Frequenzen und Winkel dieser temporären Schallquelle in den nächsten freien Platz im Klassifikator der echten Schallquellen kopiert, damit im nächsten Messdurchgang überprüft werden kann ob noch einmal ausreichend häufig Peaks dieser Schallquelle zugeordnet werden konnten.

### 5.3.3 Visualisierung Programmablauf

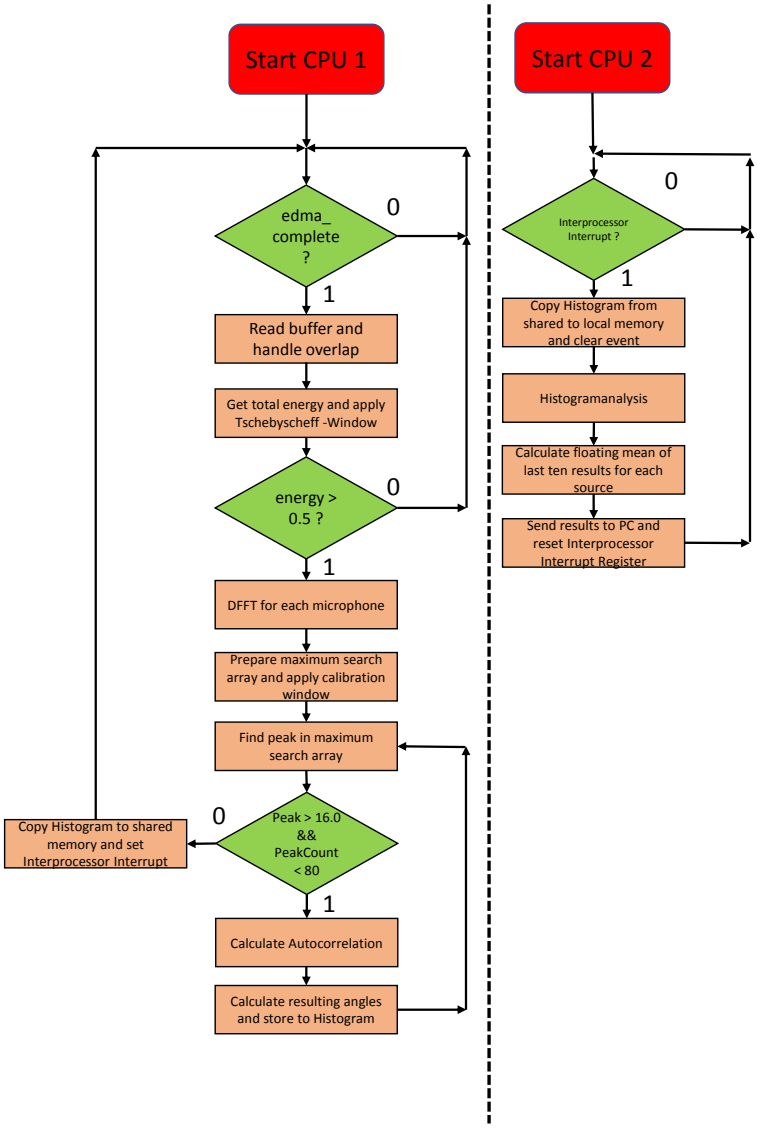


Abbildung 5.20: Programmablauf

### 5.3.4 Visualisierung der Histogrammanalyse

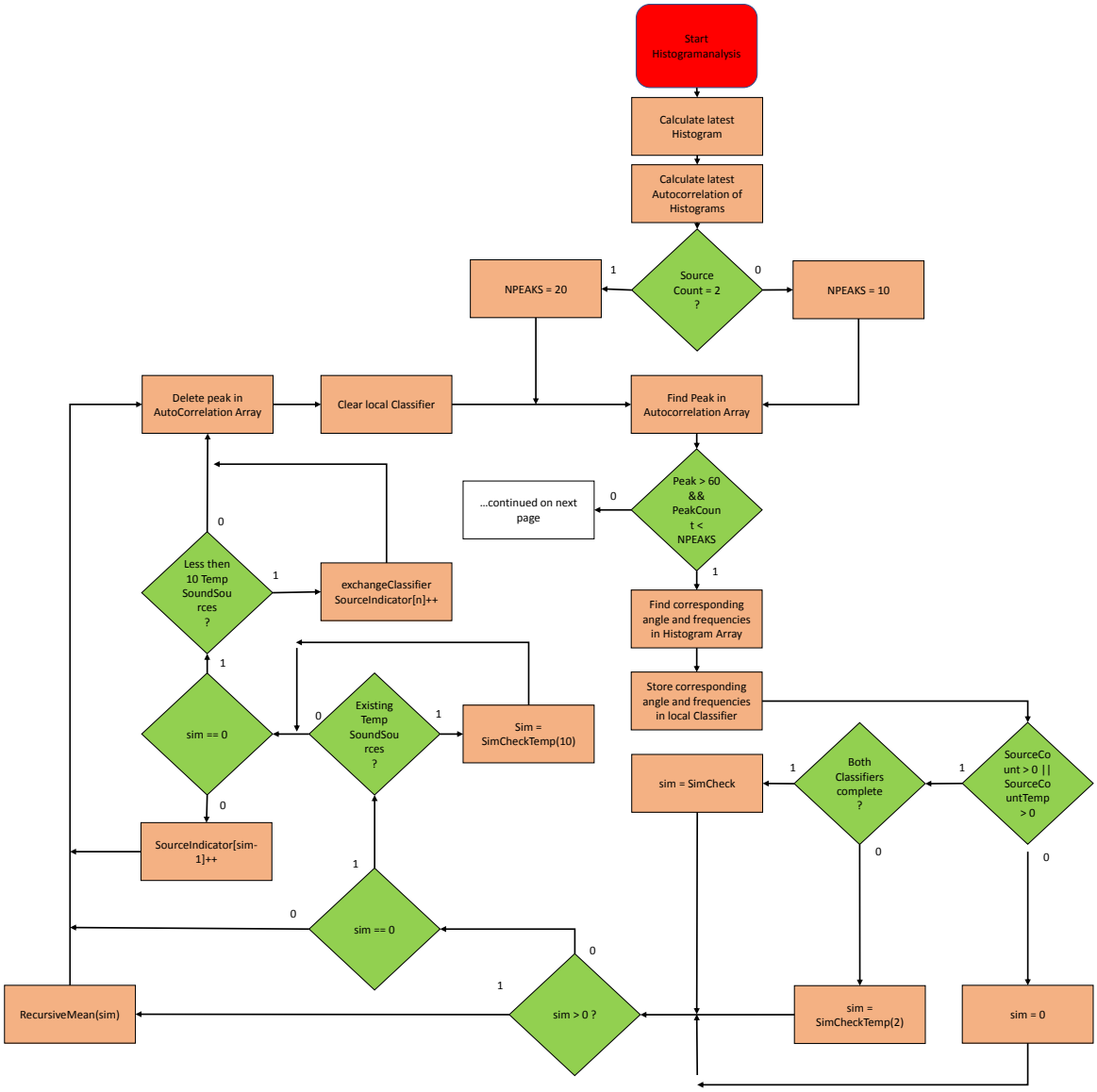


Abbildung 5.21: FlowChart Histogrammanalyse Teil 1

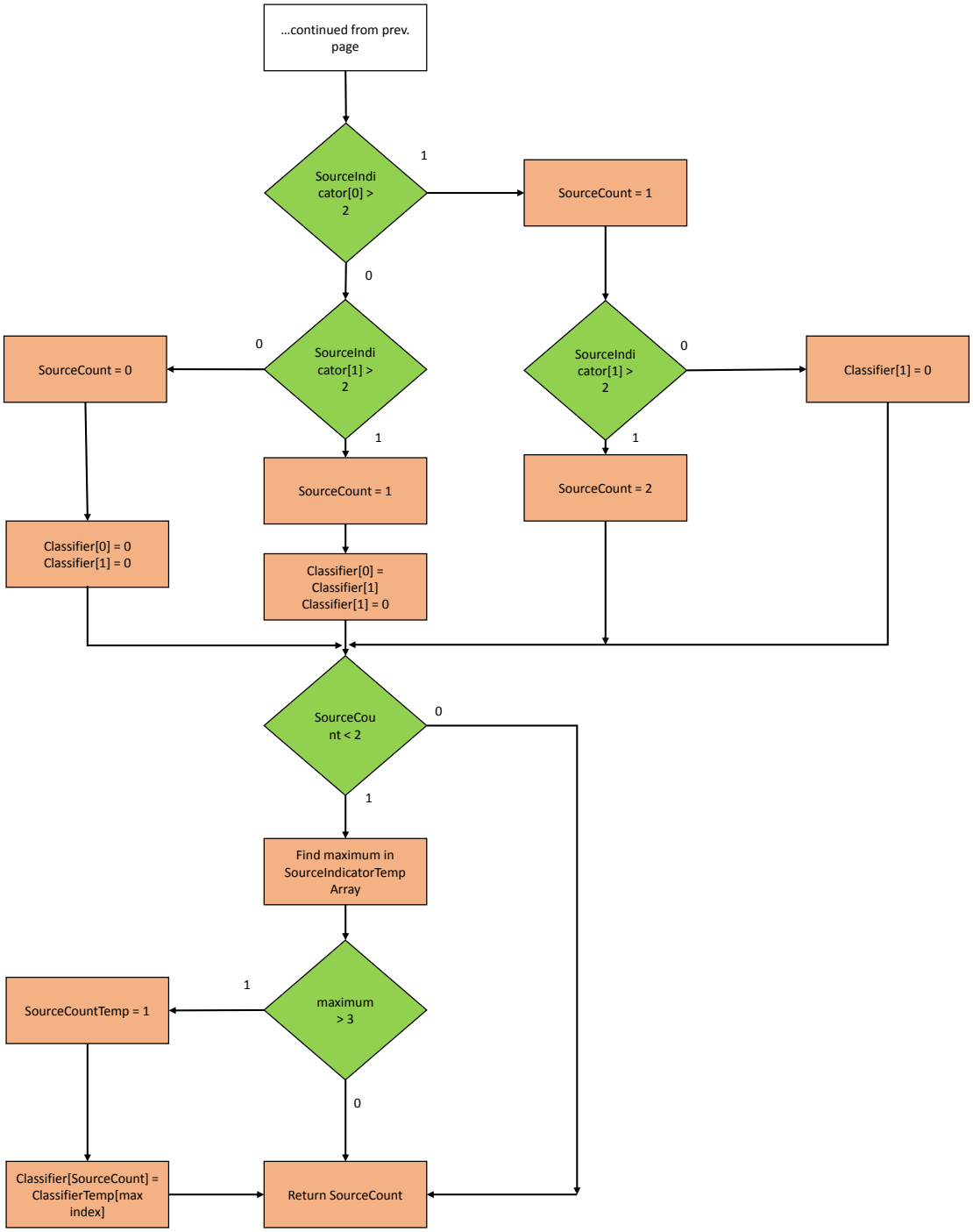


Abbildung 5.22: FlowChart Histogrammanalyse Teil 2

# 6 Abschlussbetrachtung und Ausblick

## 6.1 Abschlussbetrachtung

### 6.1.1 Portierung

Die alte Software konnte erfolgreich auf die neue Hardware portiert werden. Alle benötigten Funktionen konnten schlußendlich entweder genauso benutzt, ausgelassen oder durch neue, äquivalente Funktionen ersetzt werden.

Beim Einsatz der neuen Hardware für andere Projekte sollte auf jeden Fall immer ein Auge darauf geworfen werden wie genau die Ergebnisse der zur Verfügung gestellten Assemblerfunktionen sind. Ähnliche Probleme wie die, die sich in der vorliegenden Arbeit beim Einsatz der DFFT Funktion und der Matrix-Multiplikation ergeben haben könnten auch bei anderen, für das vorliegende Projekt nicht verwendeten, Assemblerfunktionen auftreten.

Sehr eigenartig ist aufgefallen, dass das alte Projekt auf der neuen Hardware ohne die Erweiterung der Histogramlänge und der Frequenzauflösung nicht in selben Maße funktioniert hätte wie auf der alten Hardware.

### 6.1.2 Erweiterung

Obwohl das alte Projekt erfolgreich auf die neue Hardware portiert wurde konnte man die Genauigkeit des Systems über das Klassifizieren der Schallquellen mit Hilfe ihres charakteristischen Spektrums nicht verbessern. Die Klassifizierung über Winkelbereiche funktioniert zwar zuverlässig, die Schallquellen dürfen sich aber nicht näher als maximal 45° kommen um sie voneinander unterscheiden zu können. Bei genauerer Betrachtung dieses Umstandes fällt auf, dass genau alle 45° ein Mikrofon auf dem Array angebracht ist. Anscheinend definiert die Anzahl der Mikrofone so etwas wie ein räumliches Raster und damit die Genauigkeit mit der die Richtungen aus denen Schallereignisse eintreffen unterschieden werden können.

Positiv zu vermerken ist, dass es gelungen ist das Vorgängerprojekt so zu erweitern, dass, statt nur eine, dynamik-freie Schallquelle oberhalb von 4 Khz sicher detektieren zu können, die Lokalisierung von zwei Sprach-Audioquellen ermöglicht wurde. Aufgrund der fest definierten Mikrofon-Array-Ausmaße ist dies allerdings mit Einschränkungen bei der Ermittlung der Elevationswinkel verbunden.

Betrachtet man das gesamte Vorgehen während der Histogramanalyse im großen Kontext so fällt auf, dass prinzipiel eigentlich nur zwei Dinge nötig sind um die anfallenden Messwerte zu verarbeiten. Zum einen ist eine Tiefpassfilterung der anfallenden Winkel nötig um nur diejenigen Winkel anzuzeigen deren Auftrittshäufigkeit sich im Verlauf mehrerer Messungen nur wenig ändert. Zum anderen wird für Sprecher eine Art Histerese nötig um die Dynamik auszugleichen.

Die Aufteilung der Software auf zwei CPU zur Parallelverarbeitung ist sicher eine interessante Idee und funktioniert, unter der Vorraussetzung, dass die Startsequenz richtig duchgeführt wird, zuverlässig. Es wurde jedoch nie untersucht ob man hierdurch wirklich einen Zeitvorteil erhält oder ob das hin und her kopieren der Werte im Speicher diesen wieder zunichte macht.

## 6.2 Ausblick

### 6.2.1 Entwicklung eines mathematischen Modells

Als nächsten Entwicklungsschritt kann man sich Gedanken darüber machen wie die gezeigte Tiefpassfilterung- und Histeresebearbeitung der Messwerte in eine mathematisches Modell überführt werden können.

### 6.2.2 EDMA Interrupt und Semaphore

Hardwarefuntionsseitig können Versuche unternommen werden die EDMA Funktion Interrupt gesteuert und den Datenaustausch zwischen den CPU mit Hilfe einer Semaphore zu implementieren.

### 6.2.3 Erweitern der Klassifizierung

Für die Zukunft könnte es sich lohnen zu untersuchen ob eine Erweiterung der Klassifikatoren auf 50 oder 100 Frequenzen pro Schallquelle die Klassifizierung über das charakteristische Spektrum verbessert. Ebenso könnten sich Gedanken über andere Klassifizierungsmöglichkeiten gemacht werden. Anbieten würde sich eventuel ein Bayes Klassifikator.

### 6.2.4 Implementierung Kalman Filter

In der vorliegenden Arbeit wird die Dynamik von Sprechern einfach durch eine Akkumulation während aktiver Phasen in den SourceIndicator Variablen erreicht. In Phasen in denen keine Messwerte auftreten, der SourceIndicator Wert der letzten Durchgänge aber darauf hinweist, dass gerade ein aktiver Sprecher vorhanden ist wird lediglich der letzte ermittelte Wert noch einmal an den PC übertragen bis der SourceIndicator endgültig leer gelaufen ist. Dies ist völlig ausreichend für räumlich-statische Schallquellen. Sollte sich ein Sprecher jedoch bewegen reicht dies nicht mehr aus.

Für diesen Fall könnte ein Kalman Filter implementiert werden, dass, basierend auf der Vergangenheit des Signals Vorhersagen über den aktuellen Verlauf machen kann obwohl die Messdaten für eine kurze Zeit ausgeblieben sind.

# Literatur/Quellen


[1] DSignT *D.Module.C6713 Spezifikationen* https://www.dsignt.de/de/embedded-dsp-boards/d-module2-c6713-dsp-module.html

[2] DSignT *D.Module2.C6657 Spezifikationen* https://www.dsignt.de/de/embedded-dsp-boards/d-module2-c6657-tms320c6657-dsp-modul.html

[3] TI *KeyStone Architecture Enhanced Direct Memory Access EDMA3 Controller User's Guide* http://www.ti.com/lit/ug/sprugs5b/sprugs5b.pdf

[4] TI *TMS320C6000 DSP Multichannel Buffered Serial Port (McBSP) Reference Guide* http://www.ti.com/lit/ug/spru580g/spru580g.pdf

[5] TI *KeyStone Architecture Universal Asynchronous Receiver/Transmitter (UART) User Guide* http://www.ti.com/lit/ug/sprugp1/sprugp1.pdf

[6] TI *KeyStone Architecture TIMER64P User Guide* http://www.ti.com/lit/ug/sprugv5a/sprugv5a.pdf

[7] TI *TMS320 C6655 and TMS320 C6657 Fixed and Floating-Point Digital Signal Processor* http://www.ti.com/lit/ds/symlink/tms320c6657.pdf

[8] Reermann, Jens: *Echtzeit-Lokalisierung von Audioquellen mittels zirkularen Mikrofonarrays und des UCA-ESPRIT-Algorithmus* HAW Hamburg, 2013

[9] TI *A Brief History of TI Object File Formats* http://processors.wiki.ti.com/index.php/A_Brief_History_of_TI_Object_File_Formats

[10] TI *Linker Command File Primer* http://processors.wiki.ti.com/index.php/Linker_Command_File_Primer

[11] Papula, Lothar: *Mathematische Formelsammlung für Naturwissenschaftler und Ingenieure* Vieweg+Teubner, 2009 ISBN: 978-3-8348-0757-1

[12] Strang, Gilbert: *Wissenschaftliches Rechnen* Springer Verlag, 2010 ISBN: 978-3-540-78495-1

[13] Mertins, Alfred: *Signaltheorie* Vieweg+Teubner, 2010 ISBN: 978-3-8348-0737-3

[14] Kölzer, Hans Peter und Sauvagerd, Ulrich: *Discrete Fourier Transform* HAW Hamburg, 2012

[15] Zarchan, Paul und Mussoff, Howard: *Fundamentals of Kalman Filtering: A Practical Approach* American Institue of Aeronautics and Astronautics, 2009 - ISBN: 978-1-600-86718-7

[16] Hermbusche, Claus: *pcm3003_edma_DM2C6657* DSignT, 2015

[17] Karpfinger, Christian: *Höhere Mathematik in Rezepten* Springer Verlag, 2017 - ISBN: 978-3-662-54808-0

[18] DSignT *D.Module.PCM3003 Spezifikationen* https://www.dsignt.de/de/peripherie-module/d-module-pcm3003.html

[19] Klemenz, A.: *dual_core1* DSignT, 2013

# Registerbezeichnungen und Adressen

## McBsp

| MCBSP0 BYTE ADDRESS | McBSP1 BYTE ADDRESS | ACRONYM | REGISTER DESCRIPTION |
|---|---|---|---|
| | | | **McBSP Registers** |
| 0x021B 4000 | 0x021B 8000 | DRR | McBSP Data Receive Register (read-only) |
| 0x021B 4004 | 0x021B 8004 | DXR | McBSP Data Transmit Register |
| 0x021B 4008 | 0x021B 8008 | SPCR | McBSP Serial Port Control Register |
| 0x021B 400C | 0x021B 800C | RCR | McBSP Receive Control Register |
| 0x021B 4010 | 0x021B 8010 | XCR | McBSP Transmit Control Register |
| 0x021B 4014 | 0x021B 8014 | SRGR | McBSP Sample Rate Generator register |
| 0x021B 4018 | 0x021B 8018 | MCR | McBSP Multichannel Control Register |
| 0x021B 401C | 0x021B 801C | RCERE0 | McBSP Enhanced Receive Channel Enable Register 0 Partition A/B |
| 0x021B 4020 | 0x021B 8020 | XCERE0 | McBSP Enhanced Transmit Channel Enable Register 0 Partition A/B |
| 0x021B 4024 | 0x021B 8024 | PCR | McBSP Pin Control Register |
| 0x021B 4028 | 0x021B 8028 | RCERE1 | McBSP Enhanced Receive Channel Enable Register 1 Partition C/D |
| 0x021B 402C | 0x021B 802C | XCERE1 | McBSP Enhanced Transmit Channel Enable Register 1 Partition C/D |
| 0x021B 4030 | 0x021B 8030 | RCERE2 | McBSP Enhanced Receive Channel Enable Register 2 Partition E/F |
| 0x021B 4034 | 0x021B 8034 | XCERE2 | McBSP Enhanced Transmit Channel Enable Register 2 Partition E/F |
| 0x021B 4038 | 0x021B 8038 | RCERE3 | McBSP Enhanced Receive Channel Enable Register 3 Partition G/H |
| 0x021B 403C | 0x021B 803C | XCERE3 | McBSP Enhanced Transmit Channel Enable Register 3 Partition G/H |
| | | | **McBSP FIFO Control and Status Registers** |
| 0x021B 6000 | 0x021B A000 | BFIFOREV | BFIFO Revision Identification Register |
| 0x021B 6010 | 0x021B A010 | WFIFOCTL | Write FIFO Control Register |
| 0x021B 6014 | 0x021B A014 | WFIFOSTS | Write FIFO Status Register |
| 0x021B 6018 | 0x021B A018 | RFIFOCTL | Read FIFO Control Register |
| 0x021B 601C | 0x021B A01C | RFIFOSTS | Read FIFO Status Register |
| | | | **McBSP FIFO Data Registers** |
| 0x2200 0000 | 0x2240 0000 | RBUF | McBSP FIFO Receive Buffer |
| 0x2200 0000 | 0x2240 0000 | XBUF | McBSP FIFO Transmit Buffer |

Die Strukturen der jeweiligen Register und nähere Erläuterungen sind im McBsp--Datenblatt ab Seite 87 zu finden. Die Tabelle mit den Adressen der McBsp Register befindet sich auf Seite 155 im TMS320C6657 Datenblatt

## EDMA

Die Adressen der EDMA Channel Controller Register und der Parametersets sind im EDMA-Datenblatt als Offsets zur Startadresse des EDMA Channel Controllers angegeben.

Die Startadresse des EDMA Channel Controllers ist auf Seite 164 des TMS320C6657--Datenblatts zu finden. Die Offsets der Register befinden sich auf Seite 90 bis 93 im EDMA-Datenblatt.

Die Offsets der Parametersets stehen auf Seite 28 des EDMA-Datenblatts.

Die Strukturen der Register müssen dem gesamten EDMA-Datenblatt entnommen werden.

## UART

Obwohl es nicht nötig ist Kenntnisse über die von den UART-Funktionen verwendeten Register zu haben um die Kommunikation über UART mit den gegebenen Funktionen zu implementieren sollen sie hier dennoch der Vollständigkeit wegen erwähnt sein.

Auch in diesem Fall werden die Adressen der Register im UART-Datenblatt als Offsets zur jeweiligen UART Startadresse angegeben. Die Startadressen der beiden UART Schnittstellen Register findet man auf Seite 164 des TMS320C6657-Datenblatts. Die Offsets der Register befinden sich auf der Seite 28 des UART-Datenblatts. Auf den darauf folgenden Seiten sind die Strukturen der jeweiligen Register erläutert.

## Timer

Auch die Timer Register und ihre Strukturen sollen nur oberflächlich angeschnitten werden.

Wieder werden im Timer-Datenblatt die Adress-Offsets der benötigten Register zur Startadresse des jeweiligen Timers angegeben. Die Startadressen der Timer befinden sich auf Seite 163 des TMS320C6657-Datenblatts ( Achtung: hier gibt es einen Druckfehler. Statt Timer1 steht hier in einer Zeile Reserved. Das ist jedoch falsch )

Die Offsets und Strukturen der Register findet man auf den Seiten 38 bis 48 im Timer-Datenblatt

# Interprozessor Interrupts

Die Interprozessor-Interrupt-Register-Adressen befinden sich auf Seite 201 des TMS320C6657 Datenblattes. Die entsprechenden Strukturen findet man auf den Seiten 214 bis 216.

# Inhalt der CD

- Masterarbeit_FlorianHoffmann.pdf - Die vorliegende Arbeit
- MatLab - enthält folgende MatLab Skripte
  - MatMulCmplx.m - Das MatLab Skript zum Testen der Ausgangswerte der Funktionen während der Portierung
  - Test.m - ein kleines MatLab Skript für ein besseres Verständnis der Fensterung im Zeitbereich. Wurde auch zur Erzeugung der neuen Tschebyscheff Fenster verwendet.
- CCS - enthält die Quellcodes für beide CPU
  - pcm3003_edma_DM2C6657 - Quellcode für CPU 1
  - dual_core1 - Quellcode für CPU2
- Java enthält den Quellcode der GUI (ReadMe.txt beachten!)

# Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

| Hamburg, 13. März 2018 | |
|---|---|
| Ort, Datum | Unterschrift |