Fachhochschule Westküste
Hochschule für Wirtschaft & Technik

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Masterthesis

Tobias Morten Wenzel

Entwicklung eines synchronen hochkanaligen Messsystems für Luft- und Körperschall

Fachhochschule Westküste
Fachbereich Technik

Fachhochschule Westküste - University of Applied Sciences
Faculty of Engineering

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

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

Tobias Morten Wenzel

Entwicklung eines synchronen hochkanaligen Messsystems für Luft- und Körperschall

Masterarbeit eingereicht im Rahmen der Masterprüfung

im gemeinsamen Studiengang Mikroelektronische Systeme

am Department Informations- und Elektrotechnik

der Fakultät Technik und Informatik

der Hochschule für Angewandte Wissenschaften Hamburg

und

am Fachbereich Technik

Tobias Morten Wenzel

Thema der Masterarbeit

Entwicklung eines synchronen hochkanaligen Messsystems für Luft- und Körperschall

Stichworte

Multisensorsystem, hochkanalige akustische Schallortung, Sensorfusion, integrierte Messtechnik, intelligente Mikrofone, hierarchische Topologie von Messsystemen, MEMS-Mikrofone, Beschleunigungssensor

Kurzzusammenfassung

Diese Arbeit beschreibt die Entwicklung eines hochkanaligen Messsystems in dem synchrone Daten von Sensoren für Luft- und Körperschall aufgezeichnet und verarbeitet werden. Es wird ein Verfahren entwickelt die Sensordaten in einer hierarchischen Topologie geeignet zusammenzuführen.

Tobias Morten Wenzel

Title of the paper

Development of a synchronous multichannel measurementsystem for air-borne and soild-borne sound

Keywords

multisensorsystem, mulitchannel sound beamforming, sensorfusion, measurement technology, hierarchical structure

Abstract

This thesis describes the multichannel measurement technology development in which synchronic air-borne and solid-born sensor data are recorded and processed. A method for the hierarchic sensor data topology is going to be developed.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ASF	Atmel Software Framework
ASIC	anwendungsspezifische integrierte Schaltung
DASA	Delay and Sum Algorithmus
FFT	Fast Fourier Transformation
MEMS	mikromechanische Systeme (engl. microelectromechanical systems)
WEA	Windenergieanlage

1 Einleitung und Motivation

Hochkanalige synchrone Messsysteme für Luftschall werden in der akustischen Messtechnik zur Vermessung der Emission von Luftschall technischer Anlagen und bei der Bestimmung von Schallbelastungen eingesetzt. Ein Messsystem mit einer großen Anzahl von Sensoren erlaubt insbesondere die Ermittlung der Position der Schallquelle und ermöglicht damit eine Analyse der Schallemissionsursache.

Die derzeit am Markt verfügbaren Systeme zur hochkanaligen Luftschallmessungen weisen jedoch einige Nachteile auf und entsprechen in einigen Punkten nicht dem Anspruch an ein integriertes und portables Messsystem. Insbesondere ist die Zusammenführung aller Signale in einen Messdatenaufnehmer verbesserungswürdig.

Ein Bild einer akustischen Messung einer Windenergieanlage (WEA) mit einem mehrkanaligen System wird zur Anschauung in Abbildung 1 gezeigt. Im Vordergrund ist das Array der Sensoren in Sternform zu erkennen, dieses ist auf die WEA gerichtet und misst die ankommenden Schallwellen. Diese werden in dem Kamerabild farblich hervorgehoben.

Abbildung 1: Akustische Messung an einer WEA [1]

Mehrkanalige Messsysteme für Körperschall werden zur Schwingungsanalyse von Strukturen und technischen Systemen eingesetzt. Dabei wird der Entstehungsort und die Auswirkungen auf das System untersucht, die Abstrahlung von Schallwellen sind von untergeordneter Bedeutung. Die vorhandenen Systeme sind wie die Messsysteme für Luftschall in der Art der Messdatenaufnehmer und Verkabelungsweise verbesserungswürdig. Bei den beschriebenen Messsystemen werden die verwendeten Sensoren in einer zentralen Einheit zusammengeführt. Dazu wird jeder Sensor einzeln mit der zentralen Einheit verbunden.

Die beschriebenen, derzeit am Markt verfügbaren Systeme weisen folgende Eigenschaften auf:

- Große Abmessungen der Auswertehardware
- Einzelverbindung zu jedem Sensor
- Lange Rüstzeiten beim Auf- und Abbau
- Analoge Sensoren

- Aufwendiger Transport der Systeme wegen der Größe und des Gewichts

Die Motivation dieser Arbeit begründet sich mit dem Anspruch ein höher integriertes System zu entwickeln, dass den gleichen Leistungsumfang der Sensoren hat aber deutlich kleiner und kostengünstiger ist.

Der Kern der Arbeit ist die Entwicklung eines hochkanaligen synchronen Messsystems mit folgenden Eigenschaften:

- Verringerung des Rüstaufwands und der Verkabelung
- Verwendung von Sensoren aus dem Consumerbereich
- Bildung von modularen Gruppen, Zusammenfassung von Einheiten
- Implementierung von Rechenhardware in der Messkette
- Verringerung der Kosten

2 Aufgabenstellung

In dieser Masterarbeit wird ein synchrones hochkanaliges Messsystem für Luft- und Körperschall entwickelt. Es werden folgende technische Möglichkeiten geprüft und umgesetzt:

- Der Einsatz von MEMS-Sensoren aus dem Consumerbereich
- Aufbau einer neuartigen hierarchischen, dezentralen Systemarchitektur

Ziel der Entwicklung ist es dabei folgende Eigenschaften des Systems zu erreichen. Das System soll im zerlegten Zustand, ohne die Struktur des Arrays, in einer Transportbox mit den Abmaßen 600mm x 400mm x 300mm Platz finden. Dabei soll das System ein Gewicht von 10 kg nicht überschreiten. Ein Aufbau der Sensoren und der Auswertehardware soll in 10 Minuten möglich sein. Das Gesamtsystem soll die Kosten von 100,- € pro Kanal im Gesamtsystem nicht überschreiten.

3 Grundlagen

3.1 Akustik

Die Lehre der Akustik befasst sich mit der Ausbreitung, Entstehung, Erzeugung, Wahrnehmung, Messung und Anwendung von Schall [2 S. 1].

3.2 Luftschall

Schall sind mechanische Schwingungen eines elastischen Mediums, im für den Menschen hörbaren Frequenzbereich, die sich in Gasen und Flüssigkeiten ausbreiten. Dabei äußern sich die Schwankungen als Überlagerung des statischen Druckes [3 S. 1]. Ist kein Medium vorhanden ist keine Schallausbreitung möglich. Schall tritt nur in Wellenform auf und bewegt sich fortschreitend im Raum und entfernt sich dabei von seiner Quelle. Zur Beschreibung von Wellenfeldern wird das räumliche und das ebene Wellenfeld verwendet. Das räumliche Wellenfeld breitet sich in alle Raumrichtungen gleichmäßig aus, beim ebenen Wellenfeld erflogt die Ausbreitung in eine Richtung und hat damit Verktorcharakter [4].

Das Luftschallspektrum wird in drei Bereiche unterteilt. Die Unterteilung erfolgt an den Grenzen des vom Menschen hörbaren Bereichs von 16Hz und 16KHz.

Der Bereich im Spektrum unter 16Hz wird als Infraschall und der Bereich oberhalb von 16kHz als Ultraschall bezeichnet. In Abbildung 2 werden mehrere Schallquellen in dem unterteilten Spektrum dargestellt.

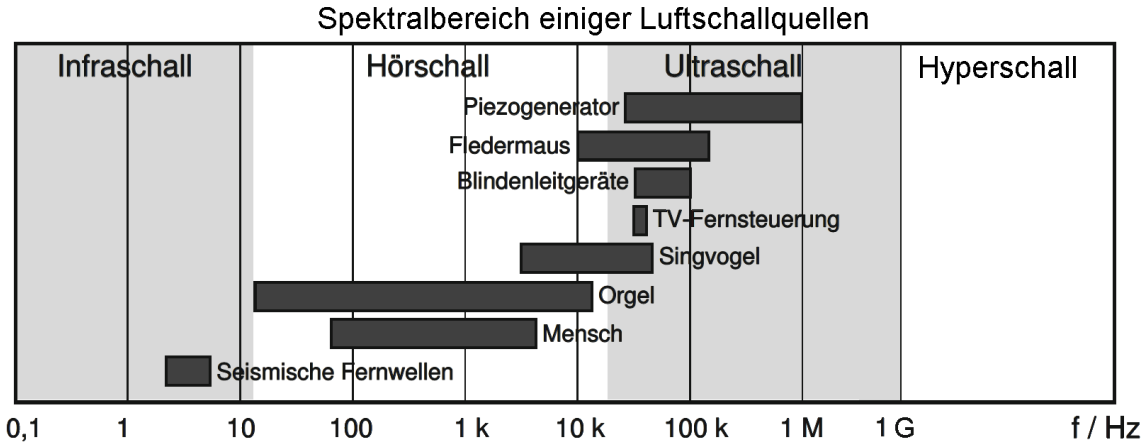


Abbildung 2: Spektralbereich einiger Schallquellen, nach [5 S. 3]

Der Bereich im Spektrum der vom Mensch erzeugt werden kann, liegt in der Mitte des hörbaren Bereichs. Es kann neben den Genannten Bereichen im Spektrum ein weiterer hinzugefügt werden, dieser wird als Hyperschall bezeichnet und beschreibt Schall mit einer Frequenz >1GHz.

Einheiten und Begriffe der Akustik

Luftschall, oder allgemein Schall in Gasen, breitet sich nur in Longitudinalwellen aus, damit schwingen die Teilchen in Richtung der Ausbreitungsrichtung der Schallwelle.

Es werden in der Akustik zur Beschreibung des Schallfeldes folgende Grundgrößen verwendet.

Schallschnelle: $\vec{v}$, in $[\mathrm{m/s}]$, schwingende Geschwindigkeit der bewegten Teilchen um ihre Ruhelage

Schalldruck: p, in [N/m^2 = Pa], veränderlicher Druck aufgrund von Teilchenbewegung

Schallgeschwindigkeit: $\vec{c}$, in [m/s], Ausbreitungsgeschwindigkeit der Schallwellen

Auslenkung: $\vec{x}$, in [m], Auslenkung der Teilchen von ihrer Ruhelage

Mit den gezeigten Grundgrößen werden in den folgenden Formeln die Zusammenhänge erarbeitet.

Ein lineares Wellenfeld bildet den einfachsten Fall der Ausbreitung von Schall. Die Ausbreitung des Schalls in einem unendlich langen Rohr wird als lineares Wellenfeld beschrieben. Die allgemeine Lösung der Wellengleichung für p= p(x,t) im linearen Wellenfeld lautet, nach [4 S. 8 ff].

$$\frac{\partial^2 p}{\partial x^2} = \frac{1}{c}\,\frac{\partial^2 p}{\partial dt^2} \qquad (3.1)$$

Für die Schallschnelle v= v(x, t) lautet die Wellengleichung im linearen Wellenfeld.

$$\frac{\partial^2 v}{\partial x^2} = \frac{1}{c}\,\frac{\partial^2 v}{\partial dt^2} \qquad (3.2)$$

Wird das Schallfeld auf eine ebene Ausbreitung erweitert, entfallen die Rohwandungen des linearen Schallfeldes. Das Wellenfeld wir nun als eben bezeichnet und basiert auf einer ungehinderten Schallausbreitung im freien Raum. In diesem erhält man für den Wechseldruck p= p(x, y, z, t), mit einsetzten des Laplace-Operators $\Delta = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2}$ anstelle von $\frac{\partial^2}{\partial x^2}$ in die Gleichung 3.1, folgende Gleichung:

$$\Delta \mathrm{p} = \frac{1}{c}\,\frac{\partial^2 p}{\partial dt^2} \qquad (3.3)$$

Die Wellengleichungen für p und v bleiben erhalten. Für eine Ausweitung in das räumliche Wellenfeld, die Kugwelwlle, wird auf die Quelle [4] verwiesen. Die Schallgeschwindigkeit in Gasen wird mit [4 S. 18]

$$c_{Gas} = \sqrt{\kappa_{Gas} R_{Gas}} * \sqrt{T} \qquad (3.4)$$

beschrieben. Diese Größe c_{Gas} wird durch die Gaskonstante R_{Gas} und den Isentropenexponent κ_{Gas} bestimmt, hinzukommt die absolute Temperatur T. Für Gasgemische ändern sich die Größen R_{gas} und κ_{gas} wie folgt.

$$R_{Gas} = \sum_i y_i * R_i \qquad (3.5)$$

$$\kappa_{Gas} = \sum_i y_i * \kappa_i \qquad (3.6)$$

Für die Beschreibung des Schalls an einem Ort sind beide Messgrößen Schalldruck und Schallschnelle ausreichend. Beide Größen lassen sich durch einfache Messaufbauten bestimmten und finden bei der größten Zahl akustischer Untersuchungen Anwendung. Nach [5 S. 5 ff] ist die Schallschnelle durch die nach EULER benannte Gleichung

$$v = -\int \frac{1}{\rho}\frac{\partial p}{\partial x} dt \qquad (3.7)$$

Diese Gleichung zeigt, wie über die Druckdifferenz die Schnelle in Messrichtung über einen beliebigen Bereich berechnet werden kann.

Schallwahrnehmung und Pegelgrößen

Die menschliche Wahrnehmung oder Psychoakustik von Schall ist subjektiv, alters- und geschlechtsabhängig, und ist nur näherungsweise mit Messtechnik nachzubilden. Die Abbildung 3 zeigt das Abnehmen der Empfindlichkeit des Ohres bei sinkender Frequenz und einen nicht linearen Zusammenhang von Frequenz und Hörschwelle [4]. Die Hörschwelle beschreibt den niedrigsten Schalldruck der vom Menschen noch wahrgenom-

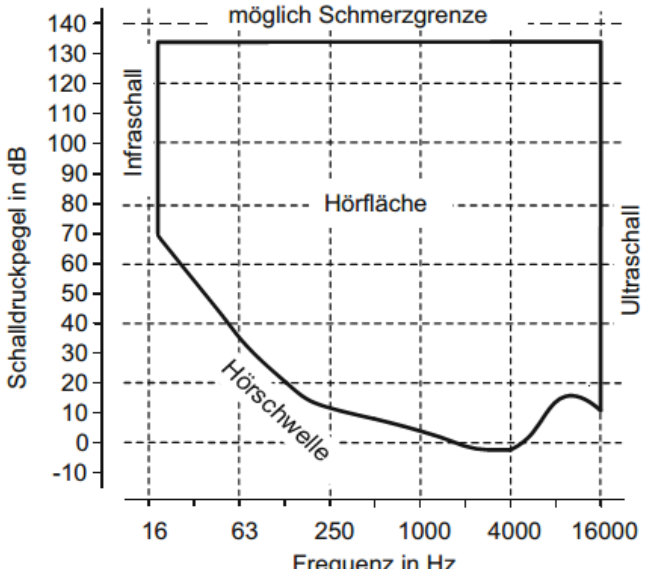


Abbildung 3: Hörfläche des menschlichen Ohres [4]

men wird. Zur Veranschaulichung der Bewertungskurven zur Nachbildung des menschlichen Hörverhaltens sind in Abbildung 4 die Bewertungskurven nach DIN EN 61672-1 gezeigt. Diese werden in Form von Filtern angewandt um die Schallwahrnehmung des Menschen messtechnisch nachzubilden.

In der Akustik ist es üblich Werte in Bezugsgrößen oder Pegelgrößen, wie z.B. dBA[1] anzugeben. Diese Umrechnung hat den Vorteil, dass der große Wertebereich, der bei Schallmessung entsteht, mit handhabbaren Zahlen ausgedrückt werden kann.

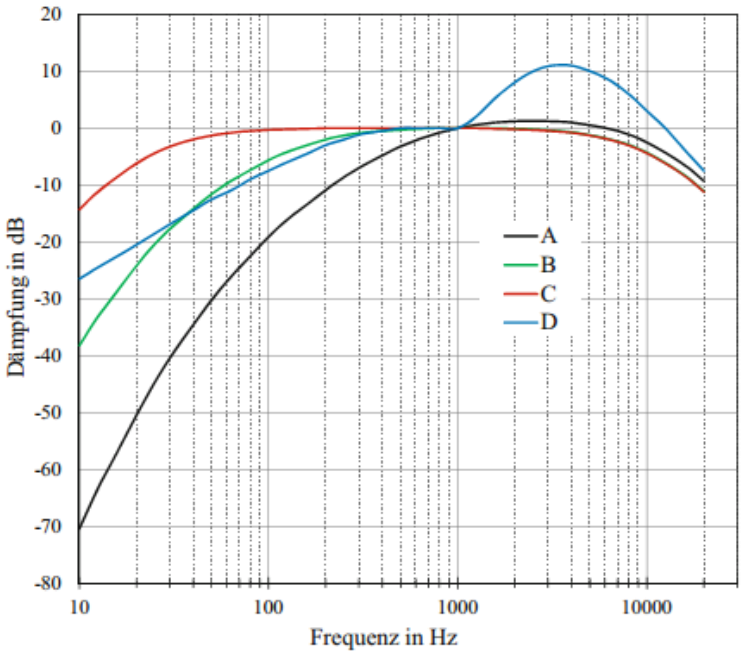


Abbildung 4: Bewertungskurven A,B,C,D [4]

[1] „Der A-bewertete Schalldruckpegel ist der gewichtete Mittelwert des Schalldruckpegels (dB) in Abhängigkeit von der Frequenz eines Geräusches. Diese Gewichtung berücksichtigt die Eigenschaft des menschlichen Gehörs, Schalldruckpegel bzw. Töne unterschiedlicher Frequenzen unterschiedlich stark wahrzunehmen.“ [22]

Zur Berechnung des Schalldruckpegelwertes mit gemessenen Druck wird

$$L_p = 20\ \log_{10}\left(\frac{p_{eff}}{p_0}\right) = 10\log_{10}\left(\frac{p_{eff}^2}{p_0}\right) dB \qquad (3.8)$$

genutzt, wobei p_0 den Bezugsruck von p_0 = 20 µPa bezeichnet. Neben den Pegeln für den Schalldruck wird analog der Pegel für die Schallschnelle, mit der Bezugsschnelle von v_0 = 50nm/s = $5*10^{-8}$ m/s, berechnet. Es gilt eine Verdopplung des Schalldrucks bedeutet eine Pegelerhöhung um +6dB. Eine Verdopplung der Schallleistung bedeutet eine Pegelerhöhung um +3dB.

3.3 Akustische Wellen in festen Körpern

Akustische Wellen in festen Körpern beschreiben die Ausbreitung von Störungen im Körper, dabei werden die Teilchen des Körpers aus der Ruhelage ausgelenkt.

Einheiten und Grundbegriffe

Die Schallausbreitung in festen Körpern unterscheidet sich erheblich von der in Gasen. Eine Welle kann sich in einem festen Körper nur durch mechanische Verschiebung ausbreiten. In festen Körpern können neben Longitudinalwellen auch Transversalwellen entstehen. Damit wird die Beschreibung von Wellenausbreitungen in festen Köpern deutlich schwieriger als bei Gasen.

Die Grundgrößen zur Beschreibung von Wellen in festen Körpern sind die Ausbreitungsgeschwindigkeit, die Frequenz der Schwingung und die Amplitude der Schwingung.

Ausbreitung von Wellen in Körpern

Die Ausbreitung von Wellen in festen Körpern wird durch mehrere Wellenformen beschrieben [6 S. 119] . Diese sind nachfolgend aufgelistet.

a) Reine Longitudinalwellen

   Die Materialteilchen bewegen sich wie beim Luftschall in Richtung der Ausbreitung. Diese Wellenform entsteht bei Körpern deren Ausdehnung groß gegenüber der Wellenlänge ist.

b) Reine Transversalwellen

   Bei Transversalwellen werden die Teilchen senkrecht zur Ausbreitungsrichtung ausgelenkt. Es wird eine Formänderung (Schubdeformation) in die Struktur eingeprägt. Aber es entsteht keine lokale Volumenänderung. Die Ausbreitung der Wellen erfolgt in Körpern deren Ausdehnung groß gegenüber der Wellenlänge ist.

c) Mischtypen

   Mischtypen der Wellenformen entstehen durch eine Überlagerung der beiden beschriebenen Wellenformen. Es treten Dehnungswellen, Biegewellen, Torsionswellen und Rayleighwellen[2] auf.

Die Ausbreitungsgeschwindigkeit von Longitudinalwellen in festen Körpern [3 S. 110] wird mit

$$\boldsymbol{c_L} = \sqrt{\frac{2*\mu+\lambda}{\varrho_0}} = \sqrt{\frac{2G(1-\nu)}{\varrho_0(1-2\upsilon)}} \qquad (3.9)$$

beschrieben. Setzt man in die Formel die Sofftgrößen für Aluminium, die Poissonzahl $\mu = 2{,}4*10^{10}\,N/m^2$, die Dichte $\rho_0 = 2700\ kg/m^3$ und die Lamé-Konstante[3] $\lambda = 6{,}1*10^{10}\ N/m^2$,ein so erhält man c_L=6353,76m/s.

$$\boldsymbol{c_L} = \sqrt{\frac{2*\mu+\lambda}{\varrho_0}} = \sqrt{\frac{2*2{,}4*10^{10}\frac{N}{m^2}+6{,}1*10^{10}\frac{N}{m^2}}{2700\frac{kg}{m^3}}} = \boldsymbol{6353{,}76}\,\frac{m}{s} \qquad (3.10)$$

[2] Die Rayleighwellen wurden nach Lord Rayleigh benannt. Rayleighwellen sind Oberflächenwellen oder Grenzflächenwellen an einer freien Oberfläche eines Körpers.

[3] Die Lamé-Konstante beschreibt den Elastizitätstensor eines festen Stoffes, mit $\lambda = 2G\frac{\nu}{1-2\nu}$

Die Ausbreitungsgeschwindigkeit von Transversalwellen wird nach [3 S. 112] durch

$$c_T = \sqrt{\frac{G}{\varrho_0}} = \sqrt{\frac{\mu}{\varrho_0}} \qquad (3.11)$$

beschrieben und ist damit wie bei der Longitudinalwelle stoffabhängig. Setzt man hier die Stoffkonstanten für Aluminium ein, erhält man:

$$c_T = \sqrt{\frac{\mu}{\varrho_0}} = \sqrt{\frac{2{,}4 * 10^{10}\,\frac{\mathrm{N}}{\mathrm{m}^2}}{2700\,\frac{kg}{m^3}}} = 2981{,}42\ m/s \qquad (3.12)$$

Neben der Stoffabhängigkeit wird eine Frequenzabhängigkeit beobachtet. Diese äußert sich in der Dispersion die auftritt wenn eine Welle aus mehreren Frequenzen besteht und diese mit unterschiedlichen Geschwindigkeiten durch einen Körper wandern [2 S. 132 ff].

3.4 Messtechnik

3.4.1 Messtechnik für Luftschall

Akustische Wandler wandeln eine akustische Größe in eine Elektrische oder umgekehrt. Die Wandlung nimmt dabei immer eine mechanische Größe als Zwischengröße an und erfolgt wie Abbildung 5 gezeigt in zwei Stufen. Diese Zwischengröße kann im Fall eines Mikrofones, also der Pfad von der akustischen Größe zur elektrischen, z.B. die Schwingung einer Membran sein. Im umgekehrten Fall in dem der Wandler als Sender arbeitet, wird z.B. wieder eine Membran zum Schwingen angeregt um in diesem Fall eine akustische Größe zu erzeugen.

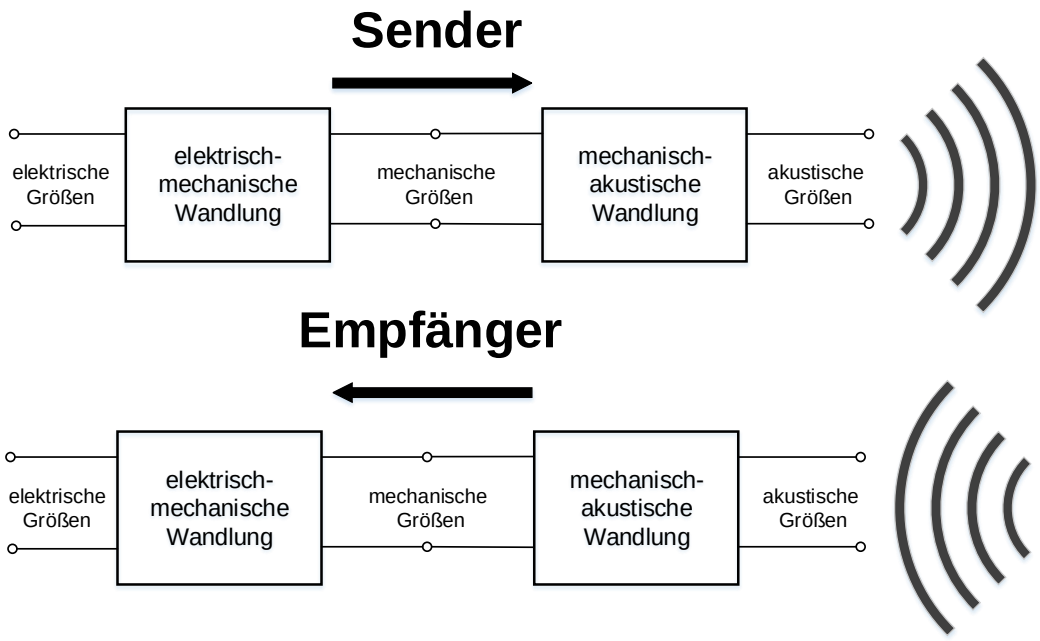


Abbildung 5: Elektroakustische Wandler, Sender und Empfänger

Im Allgemeinen kann ein akustischer Wandler reversibel und nichtreversibel gebaut werden [3]. Ein reversibler Wandler ermöglicht eine Übertragung in beide Richtungen nach Abbildung 5. Bei einem nichtreversiblen Wandler ist nur ein Betrieb in eine Richtung möglich. Die meisten verwendeten Wandler sind nicht reversibel, oder nahezu nicht reversibel, gebaut.

3.4.2 Mikrofone

Nachfolgend wird auf das Prinzip der Schalldruckmessung mittels elektrostatischer Wandler und MEMS-Mikrofonen eingegangen. Dieses Prinzip wird in den meisten erhältlichen Mikrofonen verwendet. Es gibt neben dem hier gezeigten Prinzip auch weitere Prinzipien. Diese haben eine geringere Zahl von Anwendungen und werden nicht weiter betrachtet. Aufzuführen sind elektrodynamische Wandler, Piezowandler, thermische Wandler und optische Wandler.

Elektrostatisches Mikrofon

Das elektrostatische Mikrofon oder dielektrische Mikrofon basiert auf dem Prinzip des elektrostatischen Wandlers. Dieser bildet als wandelndes Element einen Kondensator in Form einer frei schwingenden Membran aus. Die Membran arbeitet als eine Elektrode des Kondensators und befindet sich in der umgebenden Luft. Die Membran wird von dem eintreffenden Schall in Schwingung versetzt. Die Gegenelektrode ist durch einen Luftspalt von der Membran getrennt und bildet so einen Kondensator mit veränderlicher Kapazität C. Wird die Membran durch ein Schallereignis in Bewegung versetzt, ändert sich die Kapazität proportional zur dem eintreffenden Schall. Zur Auswertung der Schwingung der Membran wird folgender prinzipieller Schaltungsaufbau genutzt.

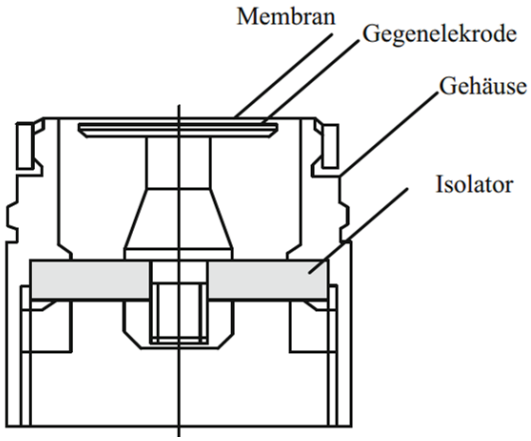


Abbildung 6: Aufbau eines Kondensatormikrofons [4 S. 160]

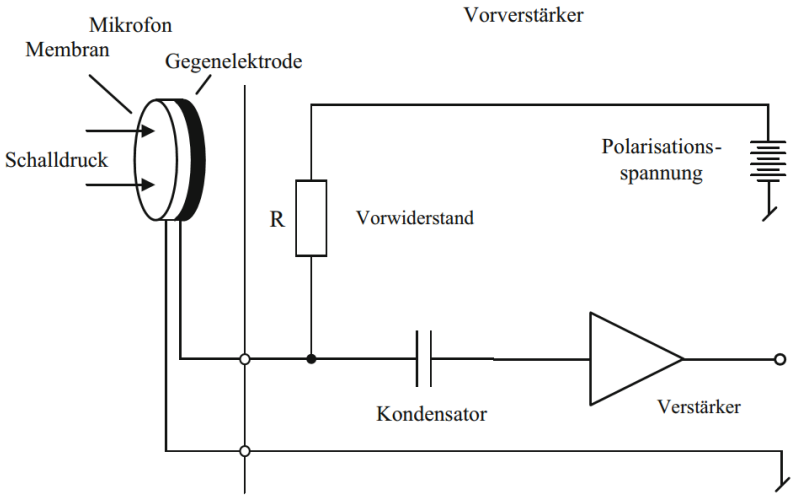


Abbildung 7: Prinzipielle elektrische Schaltung eines Kondensatormikrofons [4]

Das Frequenzverhalten des Mikrofons wird durch eine mechanische Tiefpassfilterung durch die Membran selbst und eine elektrische durch den Vorwiederstand gebildet. Um einen linearen Zusammenhang zwischen dem anliegenden Schalldruck und der Ausgangsspannung zu erhalten wird eine Polarisationsspannung, typischerweise 200V, über den Vorwiederstand R verwendet.

MEMS Mikrofon

Der Begriff des MEMS Mikrofons bezeichnet ein Membranmikrofon bei dem alle benötigten Komponenten zur Signalwandlung in Mikrostrukturen auf einem Siliziumwafer integriert sind. Diese Mikrofone werden in SMD Gehäusen ausgeliefert und sind damit direkt auf einer Platine verlötbar.

Der Schallwandler wird bei MEMS Mikrofonen durch ein Kondensatormikrofon gebildet. Dieses besteht aus einer starren Platte und einer beweglichen Membran unterhalb der starren Platte. In Abbildung 8 ist im linken Teil die Wandlereinheit zu erkennen, in der Mitte der Wandlereinheit befindet sich die mit Löchern versehene starre Kondensatorplatte. Durch die starre Platte treten die Schallwellen hindurch und treffen auf die bewegliche Membran. Diese wird angeregt und bewirkt dadurch eine proportionale Änderung der Kapazität. Die empfangenen Schwingungen werden in der im Mikrofon enthaltene Auswerteeinheit verstärkt, ausgewertet und mit der digitalen Schnittstelle weitergeleitet.

Im Gehäuse sind die Auswerteeinheit als ASIC und der Wandler meist mechanisch getrennt, dies ist in Abbildung 8 deutlich zu sehen. Die Trennung erfolgt auf Grund unterschiedlicher Herstellungsverfahren beider Einheiten. Die Maße des gezeigten Gehäuses sind 3mm in der y-Richtung und 4mm in der x-Richtung mit einer Höhe von 1,25 mm.

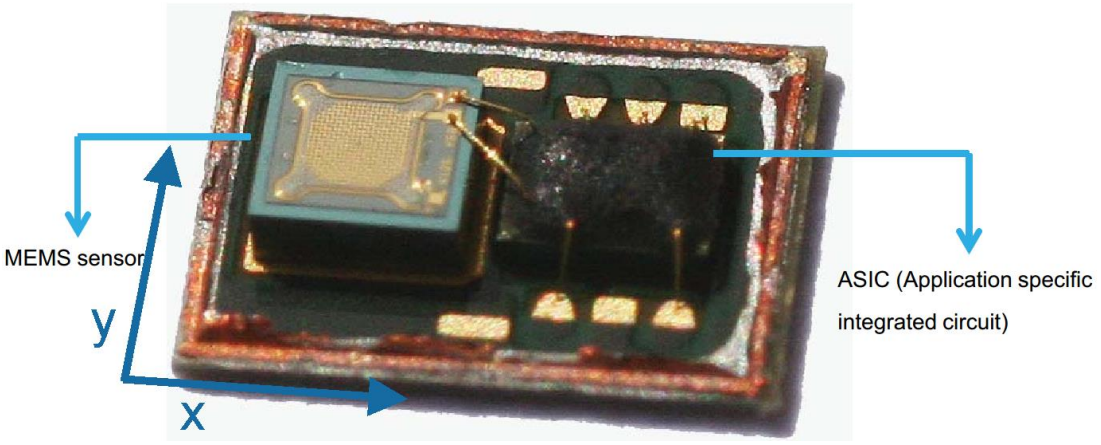


Abbildung 8: Geöffnetes Gehäuse eines MEMS Mikrofons [7]

Es werden top-port und bottom-port Mikrofone angeboten. Top-port Mikrofone haben den Schalleinlass auf der von der Platine weggewandten Seite und sind somit unabhängig von Platinendicke und Lötverfahren. Bottom-port Mikrofone haben den Schalleinlass auf der Unterseite, der Platinen zugewandten Seite. Damit läuft der Schallpfad durch die Platine. Somit ist ihre Charakteristik abhängig von der Dicke der Platine, dem Druckmesser der Bohrung, der Platzierung über der Bohrung und der Dicke der Verlötung. Beide Bauarten sind in Abbildung 9 dargestellt. Die in beiden Darstellungen gezeigte back chamber dient als Referenzdruck und ist über eine Bohrung mit der Umgebungsluft verbunden. Mit einer Veränderung dieser Bohrung ist es möglich das Schwingverhalten der Membran zu beeinflussen, hier wird der Gegendruck beeinflusst. Der Gegendruck dient darüber hinaus als Dämpfungselement der Membran.

Der Vorteil der MEMS-Mikrofone liegt in ihrer kompakten Bauform und der geringen Leistungsaufnahme im Betrieb. Dadurch lassen sie sich besonders gut in mobilen Systemen einsetzten.

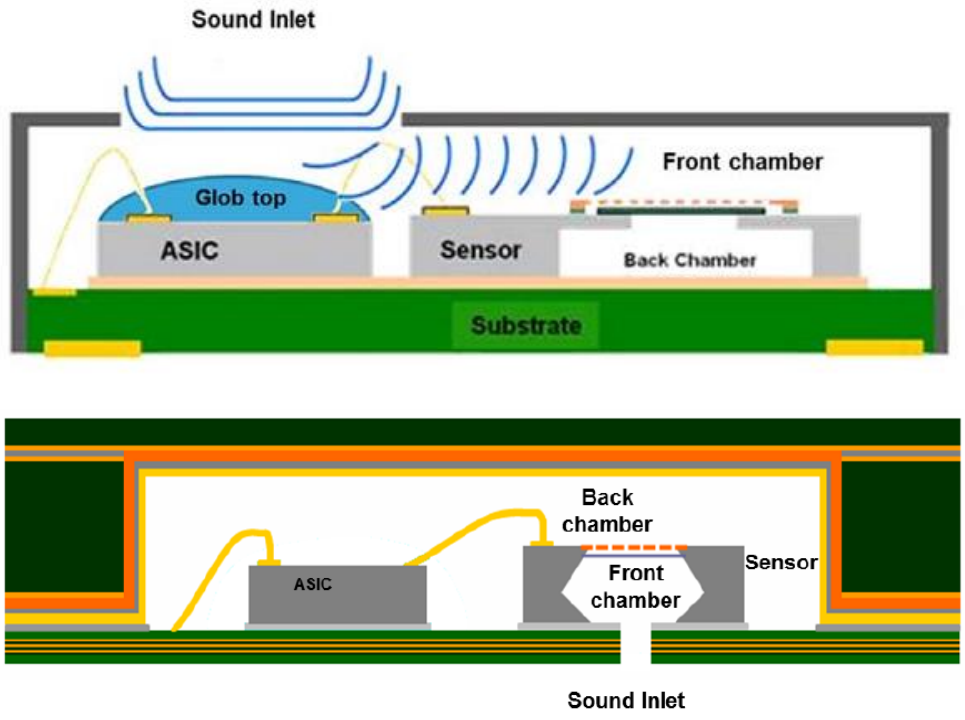


Abbildung 9: Querschnitt durch ein top-port (oben) und bottom-port (unten) Mikrofon [7]

Die Ausgabe der Signale erfolgt als PDM modelliertes Signal. Ein PDM Signal ist ein überabgetastetes Signal, welches auf die Wortlänge von 1 beschränkt ist. Das bedeutet der aktuelle Wert des Signals ist nicht durch den Wert der Ausgabe, sondern durch die Dichte der Werte beschrieben. Dies wird in Abbildung 10 deutlich.

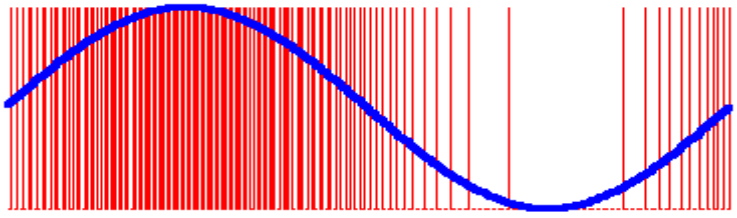


Abbildung 10: Veranschaulichung des PDM-Signals [8]

Durch eine gezielte Überabtastung ist es möglich eine bestimmte Audiofrequenz zu erhalten. Üblich ist eine Frequenz von 48 kHz*64 = 3,072MHz. Das

bedeutet das Mikrofon liefert ein 64 Mal überabgetastetes Signal, welches dezimiert wird um die Wortbreite von 1 bit auf einen Wert von 16 bit zu erhöhen. Ein Spektrum eines PDM-Signals ist in Abbildung 11 gezeigt. Der für die Anwendung interessante Bereich des Spektrums liegt am linken Rand und ist durch Filtern des Signals zu erhalten.

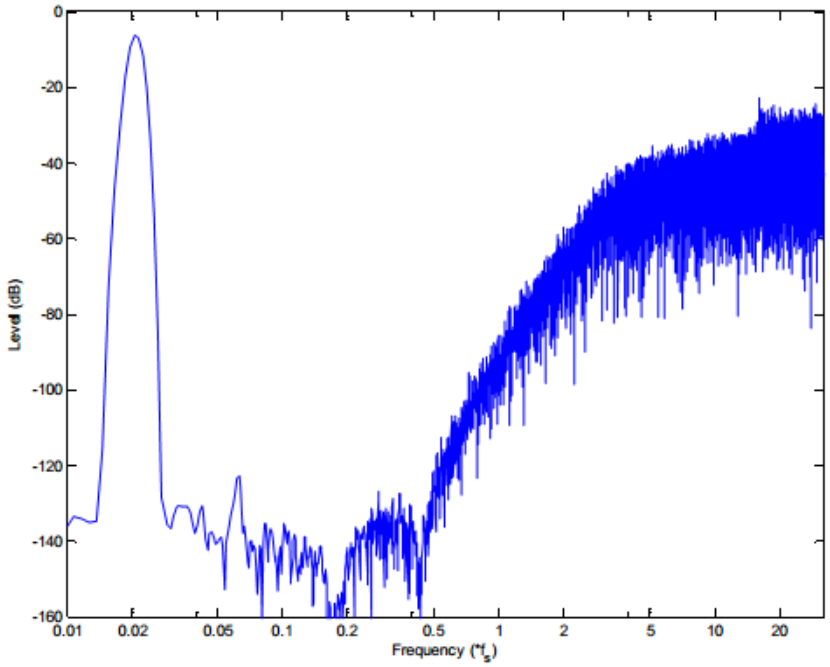


Abbildung 11: Spektrum eines PDM-Signals

Eine Besonderheit von MEMS-Mikrofonen mit einer PDM-Schnittstelle ist die Möglichkeit zwei Mikrofone mit nur einer Takt- und Datenleitung betreiben zu können. Dabei legt das erste Mikrofon seinen Ausgang bei einer steigenden Flanke auf die Datenleitung und schaltet bei einer fallenden Flanke des Taktes seinen Ausgang hochohmig. Das zweite Mikrofon arbeitet in umgedrehter Reihenfolge mit den Taktflanken. Die beschriebene Verwendung von gemeinsamen Daten und Taktleitungen ist in Abbildung 12 gezeigt. Darin ist zu sehen, dass der Pin 2 der Mikrofone bei Mic1 mit der Masse und bei Mic2 mit der Versorgungsspannung verbunden ist. Damit wurde die Auswahl der Taktflanken gemacht.

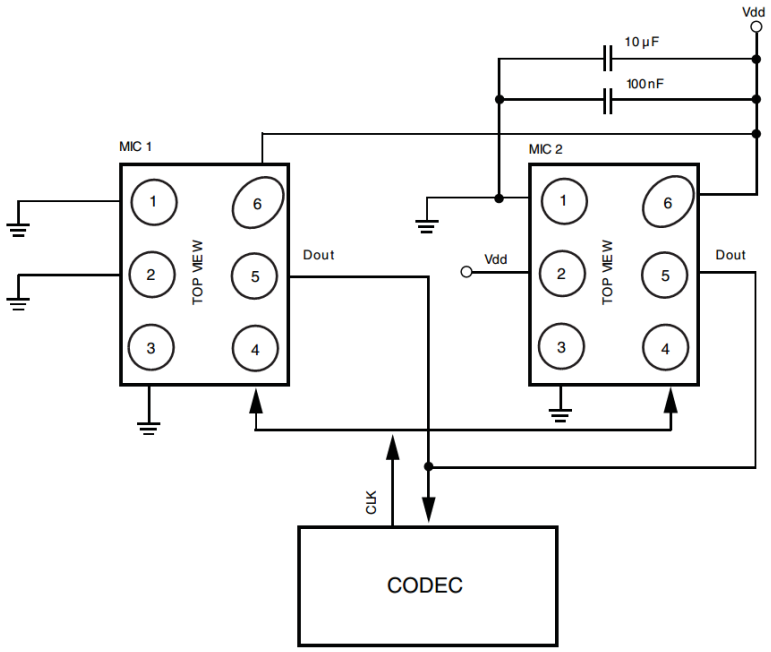


Abbildung 12: Schaltbild zweier MEMS-Mikrofone an gemeinsamen Leistungen [9]

Das zu der Beschaltung in Abbildung 12 passende Verhalten beim Senden der Daten ist in Abbildung 13 gezeigt. Es ist zu erkennen, dass im nicht aktiven Zustand eines Mikrofons dieses seinen Ausgang hochohmig schaltet und das zweite die Daten übermittelt.

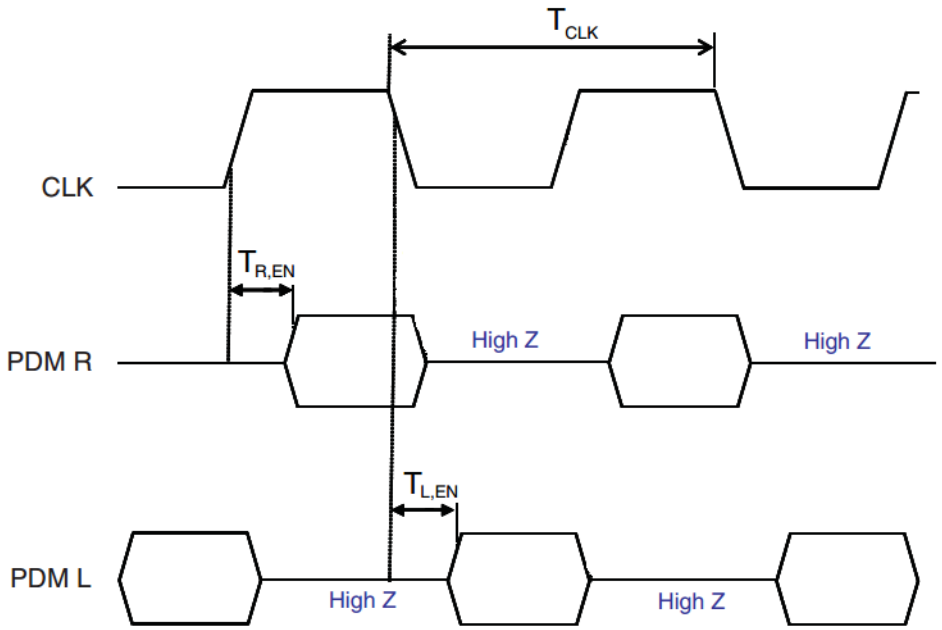


Abbildung 13: Verhalten der Mikrofone auf die Taktflanken [9]

Zur Überführung der Zusammenhänge der akustischen Größen in Größen der Messtechnik gilt nach [7]:

$$1\,Pa = 1\frac{N}{m^2} \equiv 94\,dBSPL = \tilde{p} \qquad (3.13)$$

Der Zusammenhang zwischen den akustischen Werten in dBSPL (engl. Sound Pressure Level in dB) und der digitalen Entsprechung in dBFS (engl. Full Scale) in dB ist in Abbildung 14 gezeigt.

Ein Mikrofon mit einen Acoustic Overload Point (AOP) von 120 dBSPL, einer Signal to Noise Ratio (SNR) von 63 dB und einer Sensitivität von -26 dBFS (db Full Scale), hat einen kleinsten messbaren Schallpegel von 31 dbSPL. Dieser wird bestimmt durch die Subtraktion des Dynamikbereichs von den AOP. Der Dynamikbereich wird aus der Summe von SNR und der Sensitivität gebildet. Die Rechnung lautet: 120 dBSPL (-26-63) dB =31 dBSPL. Die Abkürzung EIN seht dabei für equivalent input noise.

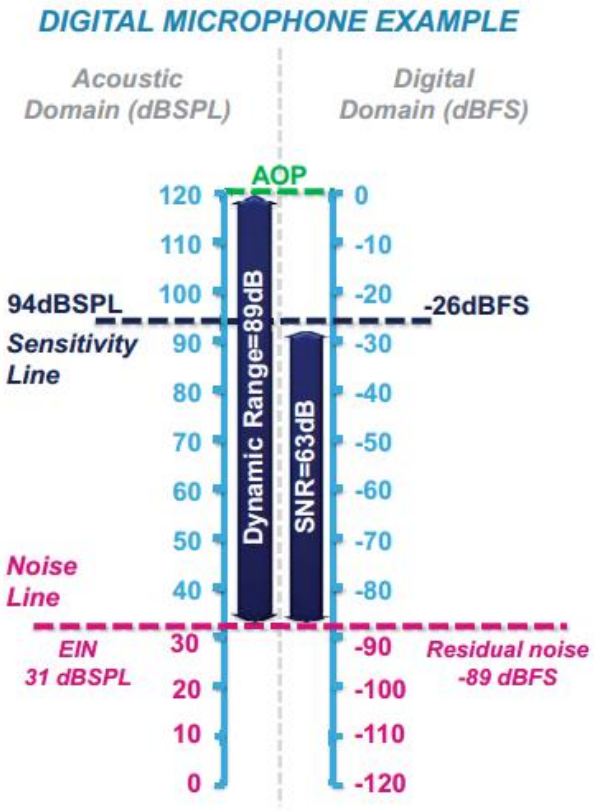


Abbildung 14: Beziehung zwischen Akustik und digitalen Werten [7]

MEMS Mikrofone werden in vielen Produkten, wie Smartphones, Headsets und Kameras, eingesetzt. Sie bieten überall einen Vorteil, wo die EMV-Robustheit und der geringe Preis von Bedeutung sind.

3.4.3 Schallintensitätsmessung

Als Schallintensität wird das zeitlich gemittelte Produkt von Schalldruck und Schallschnelle bezeichnet:

$$\vec{I} = \int_{t_0}^{t_1} \boldsymbol{p}(\boldsymbol{t}) * \vec{\boldsymbol{v}}(\boldsymbol{t}) * \boldsymbol{dt} \qquad (3.14)$$

Das bedeutet zur Bestimmung der Schallintensität sind sowohl der Schalldruck als auch die Schallschnelle messtechnisch zu erfassen. Es kommen so genannte Schallintensitätssonden zum Einsatz. Diese bestehen aus zwei Kondensatormikrofonen welche in einem bestimmten Abstand, meist 12 mm oder

50 mm, fest zu einander angebracht werden. Der Abstand zwischen den Mikrofonen bestimmt den Frequenzbereich der Sonde auf Grund der unterschiedlichen Wellenlängen die gemessen werden können. Das Messprinzip beruht auf

$$\frac{\partial v}{\partial t} = -\frac{1}{\rho}\frac{\partial p}{\partial x} = -\frac{1}{\rho}\frac{\partial p}{\partial r} \approx -\frac{1}{\rho}\frac{\Delta p}{\Delta r} \qquad (3.15)$$

vgl. [4 S. 164ff]. Sind nun zwei druckmessende Mikrofone mit einem kleinem Abstand r zu einander angeordnet, ist durch die lineare Näherung über den Druckgradienten wie folgt auf die Schnelle zu schließen.

$$\vec{I} = -\frac{p_A + p_B}{2\rho\Delta r} \int (p_B - p_A)\, dt \qquad (3.16)$$

Allgemein wird dieser Aufbau zur Messung der Schallintensität als „Zweimikrofonverfahren“ bezeichnet. Die Schallintensität wird als Zwischengröße zur Ermittlung der abgestrahlten Leistung von Quellen eingesetzt.

3.4.4 Messtechnik für Körperschall

Bei der Messung von Köperschall wenden die Schwingungen einer Struktur oder Körpers gemessen. Dies erfolgt in den meisten Fällen an der Oberfläche eines Körpers oder Struktur. Dazu wird das Messmittel mit einer geeigneten Befestigung schwingungsübertragend befestigt. Die Befestigung ist in der Praxis in den meisten Fällen reversibel, geklebt oder magnetisch haftend, und damit nach der Messung wieder entfernbar. Im Motor eines Automobils hingegen sind Körperschallsensoren zur Messung von Kopfgeräuschen der Zylinder fest integriert.

Messgrößen der Körperschallsensorik

Bei der Messung von Schwingungen eines Körpers oder einer Struktur wird die Beschleunigung an der Oberfläche gemessen. Neben der Beschleunigung sind auch die Ableitungen (Geschwindigkeit oder Auslenkung) durch Integration berechenbar.

Außerdem ist es mögliche eine Verstärker- und Auswerteschaltung mit auf dem Wafer zu integrieren was die beschriebenen Vorteile weiter positiv unterstützt.

Beschleunigungssensoren

Sensoren zur Messung von Körperschall messen vorrangig die Schwingbeschleunigung der Struktur. Zur Erfassung der Strukturschwingung ist eine möglichst kraftschlüssige Verbindung zu der Struktur zu verwenden.

Am häufigsten werden zwei verschiedene Sensortypen verwendet, piezoelektrische Sensoren und MEMS Sensoren.

Wesentliche Bestandteile eines piezoelektrischen Beschleunigungssensors sind der Piezokristall und die schwingende Masse, die mit dem Kristall verbunden ist. Übt die Masse bei einer wirkenden Beschleunigung durch ihre Trägheit eine Kraft auf den Kristall aus, so wird dieser verformt. Dabei entsteht im Kristall eine Ladungsverschiebung und eine Spannung ist messbar. Eine Prinzipskizze ist in Abbildung 15 gezeigt, neben den Funktionseinheiten ist ein robustes Gehäuse mit Schraubverbindungen dargestellt. Der Vorteil dieser Sensoren liegt in ihrer Robustheit, der Langzeitstabilität und der Einsetzbarkeit bei hohen Temperaturen.

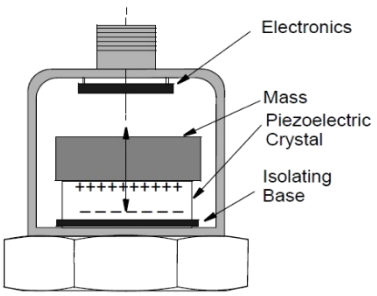


Abbildung 15: Piezo Beschleunigungssensor Prinzipskizze [21]

Der mikroelektronische Beschleunigungssensor, kurz MEMS, basiert ebenfalls auf dem Trägheitsprinzip. Hier wird die entstehende Kraft nicht durch einen Kristall gewandelt, sondern es wird die Auslenkung der Masse gegen eine Federspannung durch eine Kapazitätsänderung eines Kondensators gemessen. Der Kondensator nach dem Prinzip aus Abbildung 16, ist dabei auf einem Siliziumwafer in Mikrostrukturen realisiert. Es sind zwei Strukturen in der Ab-

bildung zu erkennen. Jede Struktur misst die Beschleunigung in einer Raumrichtung. Die dritte Raumrichtung ist in der Abbildung nicht gezeigt. Diese steht orthogonal auf der den in der Ebene angeordneten Richtungen. Durch die aus der Ebene ragende Struktur wird, bedingt durch den Fertigungsprozess, eine andere Charakteristik der dritten Achse erreicht.

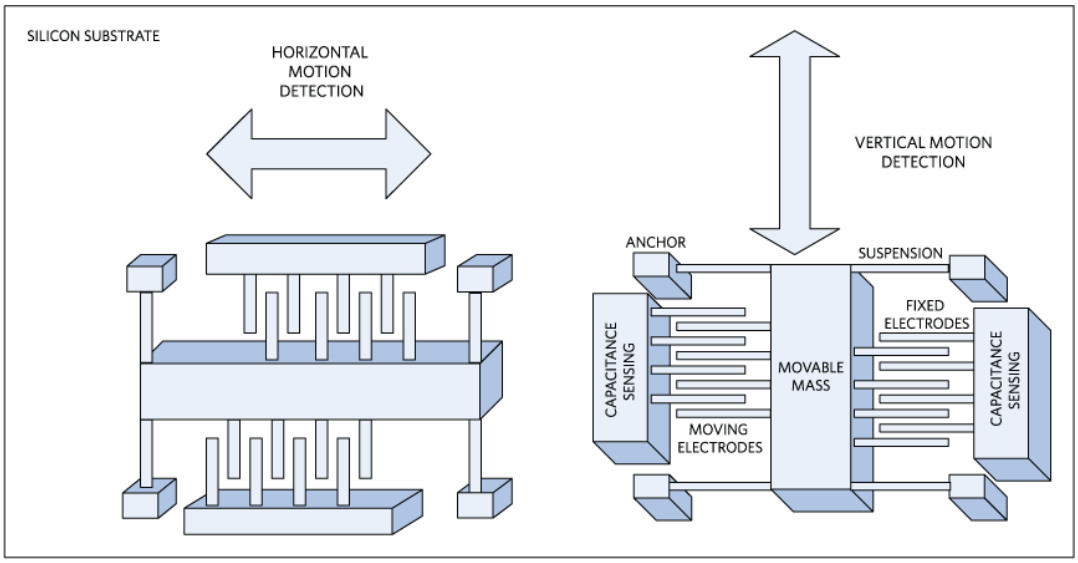


Abbildung 16: Prinzipskizze MEMS Beschleunigungssensor [10]

Die MEMS Beschleunigungssensoren bieten durch die kleinen Strukturen die Möglichkeit der Integration in kleine SMD Gehäuse. Im Gehäuse lassen auch die Auswerteelektronik und die Ausgangstreiber mit integrieren. Neben dem Vorteil der Größe ist es möglich die Auswertelektronik mit im Gehäuse zu integrieren und die Daten über eine digitale Schnittstelle aufbereitet bereitzustellen. Durch die Fertigung mit Siliziumprozessen ist es möglich die Sensoren in Massen und mit einem kleinen Stückpreis anbieten.

3.4.5 Kalibrierung akustischer Sensoren

Kalibrierung bezeichnet die Erstellung einer rückführbaren Bezugsgröße zwischen dem gemessenen Wert und der tatsächlich auftretenden Messgröße. Eine Kalibrierung ermöglicht eine absolute Aussage über die gemessenen Werte.

Abbildung 17: Schallkalibrator [23]

In der Praxis werden die Mikrofone mit Hilfe eines Schallkalibrators kalibriert, vgl. Abbildung 17. Dieser erzeugt einen definierten Schalldruck von 1 kHz. Es sind zwei Pegel am Kalibrator einstellbar 94dBSPL und 114dbSPL. Das zu kalibrierende Mikrofon wird in die Öffnung des Kalibrators eingeführt und bildet dann einen abgeschlossenen Raum in dem der eingestellte Schalldruck herrscht.

Mit einem geeichten Kalibrator wird jede Messung mit anderen Messungen des absoluten Schallpegels vergleichbar und es kann eine gleiche Messqualität über die Zeit erreicht werden.

3.5 Mehrkanalsysteme zur Messung akustischer Wellen

Der Begriff akustische Mehrkanalsysteme beschreibt Systeme zur Messung von Schallereignissen mit mehreren geeigneten parallelen Sensoren. Ihr besonderes Merkmal ist die parallele Aufzeichnung und Verarbeitung aller Sensorsignale.

Lineare Mikrofonarrays bilden die einfachste Form eines Mehrkanalsystems. Hier werden die Mikrofone in einer geraden Linie nebeneinander mit einem bestimmten Abstand angeordnet. Das kleinste Array besteht aus zwei Sensoren die im Abstand r zueinander versetzt montiert sind. Mit einem Array von zwei Sensoren ist eine einfache Lokalisierung von Schallquellen in einer Ebene, mit der Berücksichtigung von Mehrdeutigkeiten, möglich.

3.5.1 Akustische Kamera/ Antenne

Der Begriff akustische Kamera oder akustische Antenne beschreibt ein mehrkanaliges Mikrofonsystem in welchem die akustische Information einem Bild einer Kamera überlagert wird. Die Überlagerung kann dabei live während der Messung geschehen oder im Postprocessing. Dabei werden beide Informationen getrennt verarbeitet und später zur Anzeige zusammengeführt, siehe Abbildung 1.

Delay and Sum Beamforming-Algorithmus

Der Delay and Sum Beamforming-Algorithmus, kurz DSB, berechnet aus den Sensorsignalen eines Arrays den Winkel zu der Schallquelle.

Der Algorithmus zur Auswertung der Mikrofonsignale beruht auf der Summierung der zeitverschobenen Mikrofonsignale. Dazu werden die Signale zeitlich gegeneinander verschoben bis das Maximum der Summe gefunden wird. Die zeitliche Verschiebung hängt dabei von dem Einfallswinkel des Schalls ab. Wird nun das Maximum der Summe gefunden, so kann mit der durchgeführten Verschiebung der Einfallswinkel bestimmt werden. Die nachfolgende Grafik erläutert noch einmal bildlich die beschriebene Summenbildung ohne zeitliche Verschiebung

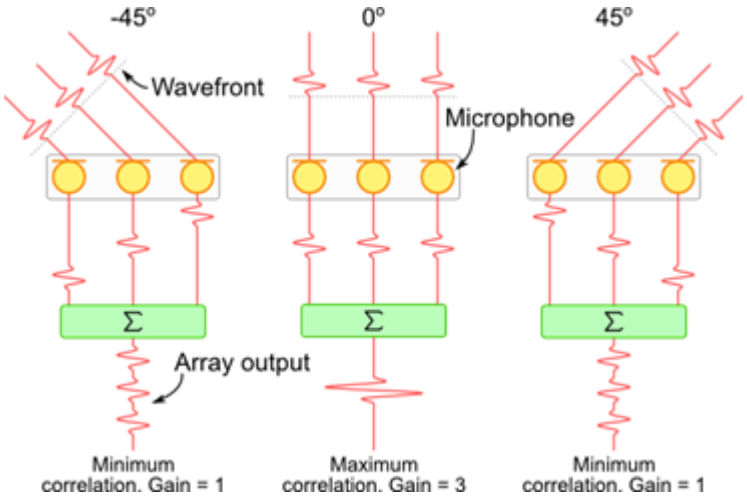


Abbildung 18: Veranschaulichung der zeitlichen Verschiebung [11]

Eine Erweiterung des Algorithmus findet bei akustischen Kameras Anwendung. Hier wird eine Messebene mit bestimmten Abstand zum Array bestimmt. Diese Ebene wird in ein Raster unterteilt. Mit dem Abstand zur Ebene ist auch der Abstand zu jedem Punkt in dem Raster bekannt und damit auch die Laufzeit zu den Mikrofonen. Mit der Kenntnis des Aufbaus des Arrays ist es möglich durch gezielte zeitliche Verschiebung der Mikrofonsignale einen Punkt im Raster der Messebene zu fokussieren und dann die resultierende Summe für diesen Punkt zu errechnen. Wird die bestimmte Summe durch die Anzahl der verrechneten Mikrofone geteilt, so erhält man den Schallpegel für das fokussierte Element. Dieses Verfahren wird für jeden Punkt im Raster angewendet um somit ein akustisches diskretisiertes Bild erzeugt.

Das beschriebene Verfahren wird in kommerziellen Produkten durch weitere Algorithmen erweitert und verbessert. Dabei kommen Verfahren mit adaptiven Filtern, Richtfunktionen über die Gewichtung einzelner Sensoren und weiteren Verfahren zum Einsatz. Des Weiterem werden zur Untersuchung von bestimmten Frequenzbereichen nur bestimmte Untergruppen der Arrays aktiviert für mit der eintreffenden Wellenlänge eine optimale Abbildung zu erreichen.

3.6 Synchronisierung von verteilten, eingebetteten Systemen

In einem eingebetteten System (engl. Embedded Systems) ist bei der Erfassung von Messwerten die Synchronität dieser besonders wichtig. Am Beispiel einer Audiomessung mit zwei getrennten Systemen werden zwei Strategien zur Synchronisierung der Signale beschrieben.

Eine Synchronisierung ist nötig, da jedes System einen eigenen Takt hat und über laufzeitbelastete Leitungen angeschlossen ist. Die verwendeten Taktgeber haben eine zeitliche Drift und weisen dadurch nach einer Zeit t eine Abweichung in der Zeit zueinander auf.

Verwendung einer Referenzzeit mit periodischer Kontrolle

Bei diesem Verfahren wird eine Referenzzeit für das System festgelegt. Alle in dem System enthaltenen Einheiten werden mit dieser Referenzzeit versehen und es gibt eine periodische Überprüfung über die Einhaltung der Zeit. Die Messwerte werden mit einem Zeitstempel versehen. Damit ist eine Verrechnung der Messwerte über das gesamte System möglich.

Es besteht die Möglichkeit die Zeit von einem externen Zeitgeber z.B. einem GPS-Empfänger in das System einzuspielen. Damit sind Systeme über weite Entfernungen zu synchronisieren.

Eine Implementierung eines Referenzzeitsystems ist das Precision Time Protocol (PTP). Die Linuximplementierung dieses Protokolls wird von ptpdv2 bereitgestellt und ist frei verfügbar auf jedem System zu installieren.

Verwendung eines Systemweiten Taktes

Mit der Verwendung eines systemweiten Taktes werden alle im System enthaltenen Komponenten mit dem gleichen Taktgeber über eine Taktleitung versorgt. Die Taktleitung kann dabei ein periodisches Signal oder ein unregelmäßiges und nicht periodisches Signal sein. Beide werden an alle angeschlossenen Einheiten verteilt. In beiden Fällen entsteht auf der Taktleitung eine längenabhängige Laufzeit welche, vermessen und korrigiert werden muss, um eine geringe zeitliche Abweichung zu realisieren.

4 Marktanalyse

4.1 Marktanalyse und Stand der Technik

In diesem Teil der Arbeit werden die am Markt verfügbaren Systeme zur ortsaufgelösten akustischen Messung beschrieben und analysiert. Hierbei liegt das Augenmerk auf der technischen Realisierung und der Signalverarbeitung dieser Systeme.

Die Systeme werden unter der Bezeichnung „akustische Kamera“ angeboten. Eine akustische Kamera beschreibt ein System bei dem ein akustisches Mehrkanalsystem mit einem Kamerabild überlagert wird. Dies erlaubt den Emissonsort und damit die Quelle sichtbar zu machen, siehe Abbildung 1. Mit einem solchen System wird die Suche nach Schallquellen von technischen Einrichtungen und Anlagen deutlich vereinfacht. Diese Systeme bilden den Stand der Technik bei akustischen Messsystemen in den Bereichen Signalverarbeitung und Sensortechnik ab. Aus diesem Grund werden diese hier untersucht.

Es werden mehrere Merkmale zur Bewertung und zum Vergleich der Systeme aufgeführt. Diese werden für jedes System, wenn verfügbar, dargestellt. Folgenden nach der Wichtigkeit geordnete Merkmale werden betrachtet:

1) Kanalanzahl
2) Typ der Sensoren

3) Art der Signalverarbeitung
4) Abtastfrequenz
5) Wortbreite
6) Frequenzbereich
7) Gewicht des Gesamtsystems
8) Leistungsaufnahme
9) Besondere Merkmale des Systems

4.1.1 Akustische Kameras mit analogen Mikrofonen

Die akustische Kamera der Firma *GFAI (Gesellschaft zur Förderung angewandter Informatik)* und deren Tochterfirma *GFAItech*, stellt den heutigen Stand der Technik mit analogen Mikrofonen im Array dar [1]. Das angebotene System besteht aus einem Mikrofonarray mit differentiellen analogen Mikrofonen, welches in mehreren Varianten für bestimmte Anwendungen angeordnet ist, einem zentralen Datenaufnahmegerät, welches die Elektronik zur Aufzeichnung der Signale enthält und einem PC auf dem die anzeigende Software und die Analysesoftware implementiert sind. Die drei Funktionseinheiten sind in Abbildung 19 gezeigt. Auf dem abgebildeten Ring sind die Mikrofone äquidistant befestigt. Die Datenaufnahme des Systems wird in einem 19 Zoll Datenrecoder zeitsynchron druchgeführt. In diesem werden die Kabel der Mikrofone zusammengeführt und die Signale gewandelt. Der Recorder kann bis zu 144 differentielle Mikrofonsignal mit 192 kHz aufnehmen und verarbeiten, dabei sind mehrere Steckkarten, mit 24 Kanälen pro Karte, eingesteckt.

Abbildung 19: Akustische Kamera GFAI [1], mit Ringarray

Die Software der akustischen Kamera stellt hier das Alleinstellungsmerkmal dar. Sie ist mit sehr vielen Funktionen ausgestattet und ermöglicht eine auf die Anwendung angepasste Analyse der akustischen Gegebenheiten. Es ist z.B. möglich im Bild lokalisierte Schallquellen auszublenden um dahinterliegende Quellen sichtbar zu machen.

Die Firma *CAE Software and Systems* bietet ebenfalls eine analoge akustische Kamera an. Der Grundaufbau dieses Systems ähnelt dem System von *GFAI* sehr. Es wird ein einzelner Datenrecorder mit Steckkarten in Verbindung mit einem Array aus Mikrofonen angeboten und die Berechnungssoftware wird auf einem separaten PC ausgeführt. Der Vorteil der von *CAE* bei den analogen Arrays herausgestellt wird, ist die hohe und variable, aber nicht näher erläuterte, Abtastrate der Mikrofone. Mit dieser Freiheit der Einstellungen können bestimmte und für die Messung interessant Frequenzbereiche insbesonderes betrachtet werden. Diese Eigenschaft ist vor allem beim Messen schneller Vorgänge, wie z.B. im Windkanal, von Vorteil [12].

Merkmal	GFAI Tech	CAE
Kanalanzahl	48 bis 144	56 bis 112
Sensoren	Analoge Mikrofone	Analoge Mikrofone
Signalverarbeitung	Auf PC nach Aufnahme/ live möglich	NI Messkarten
Abtastfrequenz	Bis zu 192 kHz	k.A.
Wortbreite	24 oder 30 bit	Bis zu 24 bit
Frequenzbereich	Max. 20 Hz bis 20 kHz	20 Hz bis 20 kHz
Gewicht	Arrays 1,2 bis 100kg,	-
Leistungsaufnahme	Bis zu 200W	-
Besondere Merkmale	Umfangreiche Software, verschiedenste Arrayanordnungen	Frei konfigurierbares Array

Tabelle 1: Übersicht analoge akustische Kameras

4.1.2 Akustische Kameras mit digitalen MEMS-Mikrofonen

Eine akustische Kamera mit digitalen MEMS-Mikrofonen hat gegenüber den analogen den Vorteil, dass die Baugröße der Mikrofone deutlich kleiner ist und die Signalübertragung rein digital erfolgt. Damit ist diese weniger störanfällig. Dies reduziert jedoch die Flexibilität durch die vorgegebene Abtastfrequenz und Wortbreite der Mikrofonsignale. Denn mit einer geforderten Wortbreite sind nur bestimmte Abtastfrequenzen möglich, vgl. Kapitel 3.4.2.

Die akustische Kamera der Firma *Norsonic* ist in drei Varianten erhältlich. Diese unterscheiden sich in dem Durchmesser und der Anzahl der Mikrofone. Die Mikrofone sind bei *Norsonic* in einer Scheibe fest integriert und haben keine filigranen Halter, siehe Abbildung 20. In der Scheibe sind alle Verbindungen zu den Mikrofonen und die der Singnalverarbeitungshardware untergebracht. Somit ist das System sehr handlich und gut mobil einsetzbar. Die Verbindung zum PC erfolgt über eine Ethernetschnittstelle und ist damit einfach zu bedienen. Neben der Datenverbindung ist nur die Spannungsversorgung an dem Array anzubringen um es zu betreiben.

Abbildung 20: Scheibenarray Norsonic [18]

Das Unternehmen *CAE* bietet neben den analogen Arrays auch eine Produktreihe mit digitalen MEMS-Mikrofonen an. Die digitalen Arrays von *CAE* sind in besonderer Weise bionisch angeordnet, siehe Abbildung 21. Diese Anordnung erhöht die mögliche messbare Dynamik der Signale, da die durch die Verrechnung entstehenden Richtcharakterisiken aus den Mikrofoneigenschaften minimiert werden.

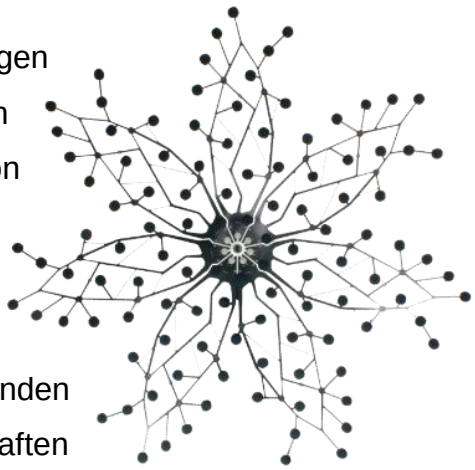


Abbildung 21: bionisches Array L-112 [20]

Zur Datenerfassung wird die Messhardware von *National Instuments* in der Narbe des Arrays von *CAE* verbaut. Diese zeichnet die Mikrofonsignale auf und gibt diese über eine Ethernetverbindung an den PC weiter.

Das Unternehmen *Distran* wählt ein anderes Design ihrer akustischen Kamera. Das angebotene System ist komplett in einem Gehäuse integriert und benötigt nur eine Verbindung zu einem PC. Es werden drei Varianten mit 64/120/128 Mikrofonen angeboten. Das Array mit 64 Mikrofonen und einer Sternanordnung ist in Abbildung 22 dargestellt. Leider wurden vom Hersteller keine Datenblätter zur Verfügung gestellt, so dass nur die auf der Website verfügbaren Eigenschaft aufgeführt werden.

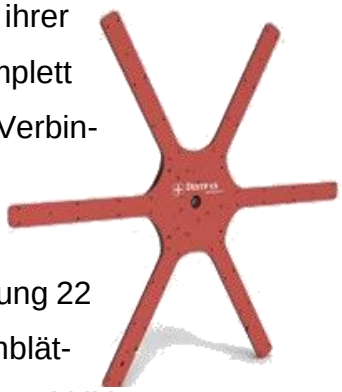


Abbildung 22: Array von Distran Switzerland [17]

Merkmal	Norsonic	CAE	Distran
Kanalanzahl	128 bis 384	40 bis 112	64/120/128
Sensoren	MEMS	MEMS	MEMS
Signalverarbeitung	Integriert in das Array	FPGA und Controller pro 40 Kanäle	k.A.
Abtastfrequenz	48 kHz	48 kHz	k.A.
Wortbreite	16bit	24bit	k.A.
Frequenzbereich	20 Hz bis 20 kHz	20 Hz bis 20 kHz	k.A.
Gewicht	Arrays 2,4 bis 16Kg	Arrays 1 bis 10,5Kg	1Kg
Leistungsaufnahme	Max 20W	<15W	k.A.
Besondere Merkmale	Integrierte Auswertehardware, Scheibendesign	verschiedenste Arrayanordnungen, handheld Version	Inegration des Gesamtsystem in einem kompakten, robusten Gehäuse

Tabelle 2: Übersicht digitale akustische Kameras

Weitere Hersteller von Systemen, wie z.B. *Mirkroflown*, mit ähnlichen oder erweitertem Funktionsumfang wurden untersucht, aufgrund fehlender oder zu geringer Daten aber nicht weiter berücksichtigt.

4.2 Analyse der bestehenden Systeme

Bei der Analyse der bestehenden Systeme sind folgende Systeme berücksichtigt worden:

- GFAItech
- CAE (analoges System)
- Norsonic
- CAE (digitales System)

Nach der Zusammenfassung der Eigenschaften der gezeigten Systeme sind folgende relevante Grundeigenschaften bei den Systemen zu erkennen:

- Mindestkanalanzahl von 40 Kanälen
- Fester Aufbau der Arrays, starre Verbindungen
- Robuste Signalübertragung, digital oder analog differenziell
- Einfacher Transport des Systems
- <20W Leistungsaufnahme eines digitalen Arrays,
- <200W Leistungsaufnahme eines analogen Arrays
- Frequenzaufnahmebereich von 20 Hz bis 20kHz
- Mindestabtastfrequenz von 48 kHz bei 16 bit Wortbreite
- Zusammenspiel mit Auswerte- und Konfigurationssoftware

Konkrete Kosten für ein System sind nur bei Kaufinteresse und nach einem Beratungsgespräch zu erfahren. Ein System von *GFAItech* ist mit Hard- und Software im Bereich von 100.000.-€ bis 200.000.-€ zu erwerben.

5 Anforderungsentwicklung

Zur Entwicklung der Anforderungen an das System werden die Erfahrungen von professionellen Anwendern mehrkanaliger Mikrofonsysteme und akustischer Kameras, neben der Markanalyse, herangezogen.

5.1.1 Anwendungsszenarien

Zur Erweiterung und Konkretisierung des Anforderungsprofils an das zu entwickelnde System wurde mit einem Partner in der Industrie und einem Competence Center der Hochschule zwei Anwendungsbespiele erarbeitet. Beide Gesprächspartner sind Anwender professioneller mehrkanaliger akustischer Messtechnik und haben mehrjährige Erfahrung mit diesen Systemen.

Die Anwendungsbespiele werden zur Erweiterung der Anforderungen an das zu entwickelnde System genutzt. Dazu sind Erfahrungen und erkannte Verbesserungsmöglichkeiten der kommerziell erhältlichen Systeme aus dem realen Messbetrieb im Gespräch aufgenommen worden.

Das Zentrum für angewandte Luftfahrtforschung (ZAL), der Partner in der Industrie, führt Freigabeflüge zur Zulassung von Passagierflugzeugen durch. Diese sind bei Neuentwicklungen und großen Veränderungen an bestehenden Flugzeugen vorgeschrieben. Bei diesen Flügen wird die Kabine des Flugzeugs mit bis zu 300 Mikrofonen ausgestattet, diese werden in einer zentralen Einheit geführt, dort verwaltet und aufgezeichnet. Dabei werden nur akustische Messungen durchgeführt. Dieser zentrale Recorder wird mit Messkarten von *National Instruments* bestückt und ist als 19 Zoll Rack ausgeführt. Der zentrale Aufbau erzeugt beim Einrichten einen großen Personal- und Zeitaufwand. Neben dem zeitlichen Aufwand behindert die aufwendige Verkabelung das Personal im Flugzeug. Im Gespräch am ZAL wurden die Inhalte des Gesprächs in einer Mindmap festgehalten, diese ist in Anhang B angefügt. Dabei sind die Anforderungen und erweiterte Funktionen aufgeteilt. Die erweiterten Funktionen beschreiben Eigenschaften, die das System für die Anwendung vom ZAL zusätzlich haben sollte, die durch die eingesetzten Systeme nicht abgedeckt werden.

Der zweite Erfahrungsträger am Competece Center CC4E in Bergedorf arbeitet mit einer kommerziellen akustischen Kamera. Dabei handelt es sich um das Array Star 48 AC Pro der Firma *GFAItech*, dieses ist in Abbildung 23 gezeigt. Bei diesem Modell werden 48 analoge Mikrofone auf drei Armen mit je 16 Mikrofonen verwendet. Die akustische Kamera wird zur Untersuchung von Windkraftanlagen im Betrieb eingesetzt. Bei den Untersuchungen der Windenergieanlagen wird die Geräuschemission mit Hilfe der akustischen Kamera identifiziert und lokalisiert. Damit ist eine Aussage über die Wirkung der Anlage auf die Umwelt möglich. Die Messungen dienen unter anderem der Erforschung der gesundheitlichen Einflüsse von Windenergieanalgen auf die im Umfeld lebenden Bevölkerung.

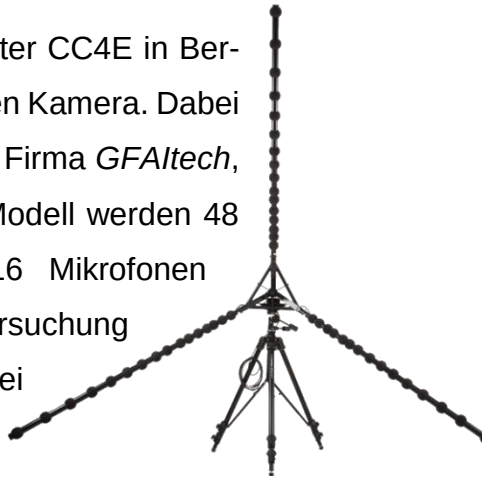


Abbildung 23: Array Star 48 AC Pro von *GFAItech* [1]

Neben der Untersuchung von Windenergieanlagen sollen die Populationen von Fledermäusen untersucht werden. Diese Tiere emittieren Schallwellen, welche im Ultraschallbereich liegen. Für diese Untersuchungen reicht die obere Grenzfrequenz der bestehenden akustischen Kamera nicht aus. Es ist das Ziel die obere Grenzfrequenz auf über 20 kHz zu erhöhen.

Die Messungen der Windenergieanlagen finden im Freifeld an Orten statt, an denen keine Infrastruktur, wie z.B. Steckdosen zur Energieversorgung, vorhanden ist, sodass die Energieversorgung des Messsystems durch portable Systeme wie Akkumulatoren und DCDC Wandlern bereitgestellt werden muss. Bei dem jetzigen System muss mit mehreren Akkumulatoren gearbeitet werden.

Die Ergebnisse des Gesprächs am CC4E sind neben denen vom ZAL in Anhang B angefügt. Dabei ist wieder die Unterteilung in Anforderungen und erweiterte Funktionen gemacht worden.

Neben den Gesprächen wurde die vorhandene akustische Kamera am CC4E in Bergedorf untersucht. Eine weitere Anforderung an das in dieser Arbeit zu entwickelnde System ist die Integrierbarkeit in das bestehende System, dazu ist es beispielsweise möglich die vorhandenen Mikrofone durch die zu entwickelnden Mikrofone zu ersetzten. Damit wäre die vorhandene Hard- und Software der akustischen Kamera nutzbar.

Die Anforderungen an das System des hochkanaligen synchronen Messsystems werden aus der Marktanalyse, den Gesprächen am ZAL und CC4E und der Analyse der akustischen Kamera von GFAItech entwickelt.

Die zusammengeführten Anforderungen sind in folgender Tabelle zusammengefasst worden.

Die Anforderungen wurden in mehreren Untergruppen unterteilt. Die erste Untergruppe beschreibt die funktionellen Anforderungen an das System, Anfor-

derungen an die Art der Messungen und die Einsatzmöglichkeiten. Eine weitere Gruppe formuliert die technischen Anforderungen an das System, hier werden Eigenschaften an die Sensoren und die Ausgabesignale beschrieben. Die letzte Gruppe zählt die nicht funktionalen Eigenschaften auf, hierzu zählen Transportfähigkeit und Kosten.

Lfd.	F/W	Anforderungen	Werte/Daten	
1		**Messgrößen**		
1.1	F	Luftschall	Frequenzbereich	100Hz bis 10kHz
1.2	F	Körperschall	Frequenzbereich	<10Hz bis 1Khz
2		**Messanforderungen**		
2.1	F	Kanäle	erweiterbare Anzahl	128/256
2.2	F		Mindestzahl	9
2.3	F	Auflösung	Wortbreite	16bit
2.4	F	Kalibrierbarkeit	Mit Kalibrator	94/114dB @ 1kHz
2.5	F	Abtastrate der Mikrofone	Frequenz	48 kHz
2.6	F	Snychronität der Signale	max. zeitliche Abweichung	10µs
2.7	F	Messachsen Beschl.		3
3		**Ausgabe**		
3.1	F	Datei mit den Daten aller Mikrofone	universal lesbares Format	
3.2	W2	Analoger Output an jedem Mikrofon		
4		**Auswertung**		
4.1	F	Ausgabe	Zeitsignal aller Mikrofone	
4.2	W2	Spektralanalyse		
5		**Geometrie**		
5.1	F	Abstand zwischen zwei Mikrofonen	möglichst gering	

5.2	W2	Aufbau	starrer Aufbau	
6		**Umwelt**		
6.1	W1	Temperatur		minus 10°C bis 50°C
6.2	W1	Feuchtigkeit	Schutzart für Außeneinsatz	IP20
7		**Kosten**		
7.1	F	Kosten pro Kanal		100,- €
7.2	F	Gesamtsystem	ohne Software	80.000,- €
8		**Energie**		
8.1	F	Energieverbrauch Gesamtsystem		<380 W
9		**Transport und Aufbau**		
9.1	W2	Einfacher Auf- und Abbau		
9.2		Handheldversion		

Tabelle 3: Anforderungen an das zu entwickelnde System

6 Konzept

In diesem Kapitel wird die Erarbeitung eines Lösungskonzeptes beschrieben. Zur Konzeptentwicklung werden die in Kapitel 5 erarbeiteten Ergebnisse der Anforderungsanalyse herangezogen.

Das erarbeitete Lösungskonzept beschreibt ein hierarchisches System zur hochkanaligen Messung von Körper- und Luftschall. Hierbei wird eine große Anzahl an Sensoren in dezentralen Einheiten zusammengeführt und ausgewertet. Nur verarbeitete und ausgewertete Signale werden übermittelt. Der Vorteil dieses Aufbaus liegt in der Verkleinerung der zentralen Einheit. Es werden Sensoren aus dem Consumerbereich verwendet. Diese Sensoren arbeiten digital und ermöglichen kompakte Abmaße der Sensormodule, bei einer geringen Leistungsaufnahme.

6.1 Systemarchitektur

Das Konzept sieht eine hierarchische Struktur der einzelnen Komponenten vor. Die unterste Ebene enthält die Stages 1, in dieser sind die Sensoren angebunden. Die nächste Ebene wird durch die Stages 2 gebildet, diese verbinden bis zu 12 der darunterliegenden Einheiten. Die höchste Ebene wird von der Stage 3 dargestellt. Diese ist nur einmal im System vorhanden und stellt das Userinterface und die Systemzeit bereit. Die Gesamtstruktur ist bildlich in

Abbildung 24 dargestellt. In dieser Abbildung ist neben den einzelnen Einheiten auch die Verbindungsstruktur in hierarchischer Weise erkennbar.

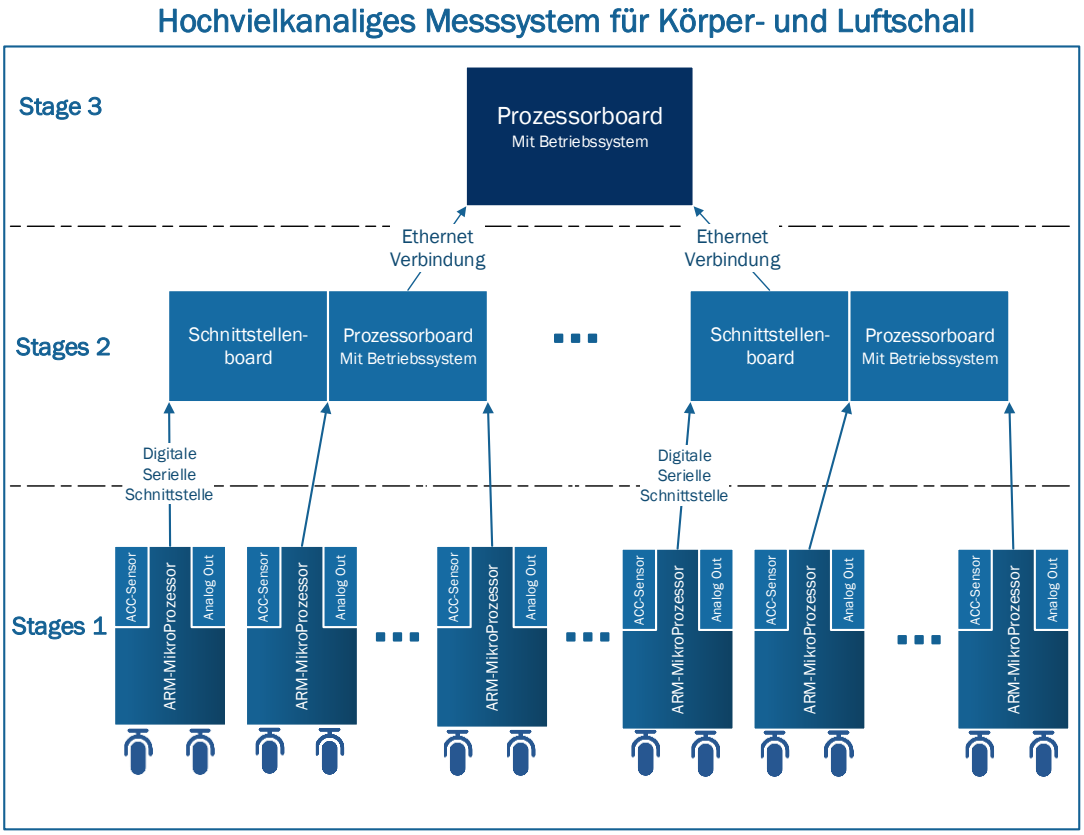


Abbildung 24: Konzept des Gesamtsystems

Im Folgenden werden die einzelnen Ebenen mit ihren Funktionen näher beschrieben.

Ebene 1 mit Stage 1:

Die unterste Ebene wird von mehreren Einheiten der Stage 1 gebildet. Eine Stage 1 besteht aus den Mikrofonen und dem Beschleunigungssensor zur Messung des Körperschalls. Neben den Sensoren wird eine Auswerte- und Kontrolleinheit in Stage 1, in Form einer Recheneinheit, integriert. Diese hat die Aufgabe die Auswertung der Sensoren zu starten und das Verhalten der Stage 1 zu kontrollieren, sowie die digitale serielle Schnittstelle zur nächsten

Ebene bereitzustellen. Des weiterem wird optional eine Möglichkeit zur analogen Ausgabe der Sensorsignale realisiert. Damit ist ein einzelnes Modul von Stage1 in der Lage, ein analoges Mikrofon in einem bestehenden System zu ersetzten. Es ist darüber hinaus möglich, den Beschleunigungssensor anstelle oder zusätzlich zu einem Mikrofon einzusetzen.

Anwendungsspezifisch wird das Verhalten der einzelnen Stages 1 angepasst, es können mehrere Sensoren zusammengefasst ausgegeben werden oder Signale gefiltert werden.

Von der Stage 1 werden mehrere über eine serielle Schnittstelle an der nächsten Ebene angebunden.

Ebene 2 mit Stages 2

In der Ebene 2 werden die Stages 2 angeordnet, eine Stage 2 ist mit mehreren Stages 1 verbunden und nimmt Daten von diesen auf. Die Stage 2 besteht aus zwei Bausteinen. Die Erste ist das Schnittstellenboard welches die digitalen seriellen Schnittstellen zur darunterliegenden Ebene bereitstellt und vorverarbeitet. Der zweite Baustein ist ein Prozessorboard auf dem ein Unix-basiertes Betriebssystem installiert ist. Die Verbindung der beiden Einheiten erfolgt über einen SPI-Bus.

Das Schnittstellenboard wird mittels eines FPGA realisiert. Der Vorteil der FPGA-Realisierung liegt in der flexiblen Verwendung der vorhandenen Anschlüsse. Ein FPGA bietet durch die frei programmierbare Logik die Möglichkeit serielle Schnittstellen zu realisieren. Damit ist die Realisierung der benötigten seriellen Schnittstellen möglich.

Das Prozessorboard wird in der Stage 2 als Kontrolleinheit der Schnittstellen und zur Synchronisierung mit den anderen Einheiten der Ebene genutzt. Es empfängt die Systemzeit von der darüber liegenden Ebene und versieht jede Messung mit dieser. Auf den Befehl des Masters wird eine Messung gestartet und mit der Aufzeichnung begonnen.

Höchste Ebene Stage 3

In der höchsten Ebene wird das Userinterface und die Kontrolle aller darunterliegenden Ebenen bereitgestellt. Des Weiteren fungiert die Stage 3 als Zeitgeber des Gesamtsystems.

Die Stage 3 kann aus einem Prozessorboard mit einem Unix-basiertes Betriebssystem bestehen. Es ist aber auch möglich das Prozessorboard nur zur Zeitsynchronisierung zu nutzten und das System von einem beliebigen Gerät über eine SSH-Schnittstelle zu steuern.

6.2 Komponentenauswahl

Die Komponentenauswahl wurde auf der Grundlage der in Kapitel 4 aufgestellten Anforderungen gemacht.

6.2.1 Komponentenauswahl Stage 1

Das Blockschaltbild der Stage 1 ist in Abbildung 25 dargestellt. Die darin gezeigten, auszuwählenden Komponenten sind ein Mikrofon für den Frequenzbereich von 100 Hz bis 10kHz, ein Mikrofon für den Frequenzbereich ab 10kHz und der Beschleunigungssensoren zur Messung des Körperschalls. Zudem soll ein Digital-Analog-Wandler optional die aufgenommenen Signale analog ausgeben.

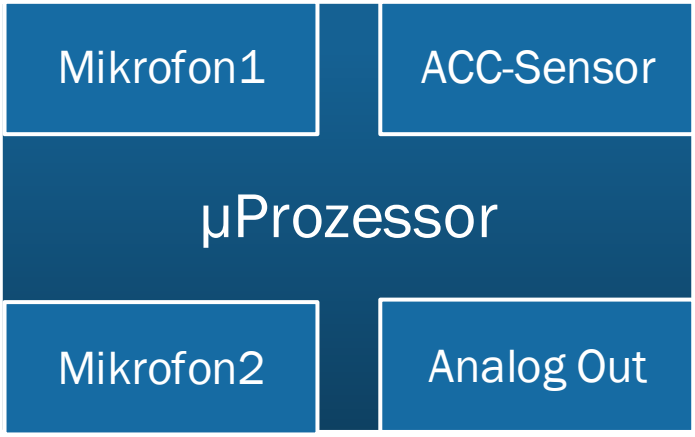


Abbildung 25: Komponenten Stage 1

MEMS-Mikrofone

Die Eigenschaften der Mikrofone lauten zusammengefasst:

- Digitale PDM-Schnittstelle
- Frequenzbereich 100Hz bis 10kHz
- Betrieb bei 3,3 V
- Hohe Sensitivität von < -20dB
- Hohe Signal-to-Noise Ratio von min 57 dB
- Breiter Frequenzbereich

Die nachfolgende Tabelle enthält die Auswahl der am Markt erhältlichen Mikrofone, welche die beschriebenen Anforderungen erfüllen.

Hersteller	Bezeich-nung	Empfind-lichkeit [dBFS] [4]	Schnitt-stelle	Signal-to-Noise ratio [dB]	Fre-quenz-bereich	Listen-preis Einzel-kauf [€][5]
ST Micro-electronics	MP45DT02	-26	PDM	61	20Hz - 10kHz	1,54
Akustika	AKU440	-26	PDM	63	50Hz-10kHz	2,25
Knowles	SPK0838HT4H	-26	PDM	64	100Hz - 10kHz	1,65
Knowles	SPH0641LU4H	-26	PDM	64,3	100 Hz -80kHz	2,91
Wolfson Microelectro-nics	WM7220	-36	PDM	58	35Hz - 9KHz	4,22

Tabelle 4: Auswahl von digitalen Mikrofonen

Mit der Tabelle und den Erfahrungen aus meiner Studienarbeit „Vier-Kanal-Mikrofon“ [13] wurde das Mikrofon MP45DT02 von *ST Microelectronics* ausgewählt. Das Mikrofon bietet gute Leistungsdaten bei einem geringen Preis und ist auf Grund der Anordnung der Lötpads gut für eine Handbestückung geeignet.

Neben dem Mikrofon für den unteren Frequenzbereich wir das SPH0641LU4H MEMS-Mikrofon von *Knowles* für den Frequenzbereich von über 10 kHz bis 80 kHz verwendet. Dieses Mikrofon ist das einzige, welches vom Hersteller für einen Betrieb bei Frequenzen über 20 kHz spezifiziert wurde. Das SPH0641LU4H ist ein bottom-port Mikrofon und nimmt die Schallwellen durch ein Loch in der Platine auf.

[4] [dBFS] engl. „Dezibel full scale“ , bezeichnet den logarithmischen Abstand zum Maximalwert des Pegels (0dB) [23]

[5] Stand 25.03.2016

Beide beschriebene Mikrofone sind in SMD-Bauweise ausgeführt und lassen sich damit flexibel auf der Platine der Stage1 platzieren.

Beschleunigungssensor

Zur Messung des Körperschalls wird ein Beschleunigungssensor ausgewählt. Die Anforderungen für die Auswahl des Sensors sind nachfolgend aufgeführt:

- Frequenzbereich kleiner 10Hz bis 1kHz
- Beschleunigungsbereich mindestens ±4g
- 3 Messachsen, selektiv abschaltbar
- 16 bit Wortbreite
- I2C Schnittstelle
- 3.3V Versorgungsspannung

Hersteller	Bezeichnung	Anz. Achsen	Schnittstelle	Messbereich [g]	Wortbreite [bit]	Listenpreis Einzelkauf [€][6]
NXP	MMA8452 QT	3	I2C	±2 bis ±8g	12/8 bit	1,20
Analog Devices	ADXL343B CCZ	3	I2C/SPI	±2 bis ±16g	10 bis 13 bit	2,73
ST Microelectronics	LIS3DSH	3	I2C/SPI	±2 bis ±16g	16 bit	2,17
Kionix	KXTJ2-1009	3	I2C	±2 bis ±8g	8/12/14 bit	2,04

Tabelle 5: Auswahl von Beschleunigungssensoren

Die Auswahl des Sensors zur Messung der Beschleunigung wurde nach drei Hauptkriterien durchgeführt: der Größe, I2C Schnittstelle und der im Sensor

[6] Stand 25.03.2016

vorhandenen Funktionen, wie z.B. Interruptgeber bei Vibrationen. Alle gezeigten Sensoren arbeiten mit der bei den anderen Komponenten verwendeten Versorgungsspannung von 3,3V.

Es wurde LIS3DSH von *ST Microeletronics* ausgewählt. Dieser Sensor erfüllt die Anforderungen und bietet eine Vielzahl an Funktionen, wie eine integrierte programmierbare State Machines, einen Temperatursensor und einen Selbsttest aller Funktionen. Des Weiterem werden die Werte mit 16 bit übermittelt, dies passt zu den Daten der Mikrofone und vereinfacht die Verrechnung der Sensorsignale. Somit fügt sich dieser Sensor gut in das System ein und ergänzt die Funktionen der Stage 1.

Digital-Analog-Wandler

Es ist der PCM5121 Digital-Analog-Wandler (DAC) von *Texas Instruments* ausgewählt worden. Dieser bietet die geforderten Schnittstellen auf der Prozessorseite und ermöglich damit eine einfache Integration in das System. Auf der analogen Ausgabeseite besteht die Möglichkeit eine Vielzahl an Einstellungen vorzunehmen. Der DAC empfängt Audiodaten im Stereoformat mit 16/14/32 bit und ermöglich somit eine Anpassung an die spezifischen Aufgaben.

6.2.2 Prozessorauswahl

Bei der Wahl des Prozessors für die Stage 1 wurden folgende Randbedingungen berücksichtigt:

- Hardwareschnittstelle für das PDM-Signal der Mikrofone
- 3,3 V Versorgungsspannung
- QFP-Format des Gehäuses
- I2S-Ausgabe
- Mehrere konfigurierbare serielle Schnittstellen
- Flexible Programmierbarkeit

Die verfügbaren Prozessoren basieren auf der ARM Cortex-M Plattform und werden von mehreren Herstellern angeboten. Die ARM Cortex-M bietet eine hohe Performance bei gleichzeitig geringer Leistungsaufnahme, welche grundsätzlich mit der gewählten Taktfrequenz variiert. Konkret werden von *NXP* und *Atmel* Prozessoren mit geeigneter PDM-Schnittstelle angeboten.

Bei *NXP* bietet die Serie LPC54xxxx eine solche Schnittstelle. Diese Serie bietet darüber hinaus eine Vielzahl an Schnittstellen und alle geforderten Eigenschaften.

Atmel bietet mit der Serie ATSAM G5x ebenfalls Produkte mit einer integrierten PDM-Schnittstelle und der dazugehörigen Verarbeitung in Hardwarebausteinen. Auch die *Atmel* Serie bietet alle geforderten Eigenschaften.

Es ist das Model *Atmel*-SAM G53N19 gewählt worden. Dieses wurde aufgrund der vorhanden Kenntnisse mit *Atmel*-Prozessoren und die vorhandene Programmierhardware ausgewählt. Es basiert auf dem ARM Cortex-M4 Chip und verfügt über eine integrierte DSP-Einheit, eine dreistufige Pipeline mit Sprungvorhersage, eine 32bit Multipliziereinheit und einem Takt von 48Mhz. Mit der Wahl des G53 wurde ein Mittelklassemodel der Serie ausgewählt. Damit ist die Leistungsaufnahme geringer. Das Modell G54 bietet eine noch größere Anzahl an Schnittstellen, die für diese Anwendung nicht benötigt werden, und einen höheren Takt von bis zu 95 MHz.

Die Kosten für den Prozessor liegen bei 5,28 € bei einem Einzelkauf. Damit ist das Modell wie bei den anderen Eigenschaften im Mittelfeld der Produktreihe.

6.3 Auswahl der Komponenten für Stage 2+3

Für die Stage 2 wird eine Kombination aus einem Prozessorboard und einem FPGA gesucht. Dies ist in Abbildung 26 dargestellt. Die Stage 3 wird aus dem gleichen Prozessorboard gebildet und kann mit der gleichen Peripherie betrieben werden.

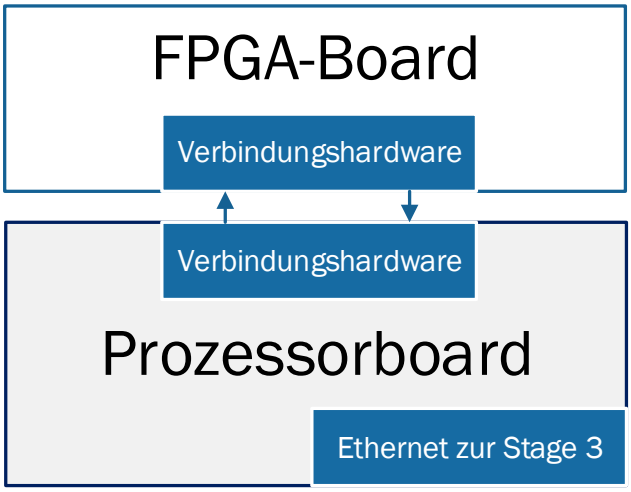


Abbildung 26: Komponenten der Stage 2

Es stehen zwei Lösungen zur Auswahl:

- LogiPi mit einem Raspberry Pi
- LigiBone mit einem BEAGLEBONE BLACK

Beide Lösungen haben die gleiche Leistungsfähigkeit. Beide zur Auswahl stehenden FPGA-Boards werden mit dem gleichen FPGA ausgeliefert und *ValentFx* gefertigt. Die Lösung mit der Kombination aus LogiPi und Raspberry Pi werden eingesetzt, beide sind fest über eine GPIO-Leiste verbunden und bilden eine kompakte Einheit. Diese Lösung bietet, aufgrund der größeren Verbreitung des Raspberry Pis und des geringeren Preises für den Raspberry das bessere Preis-Leistungs-Verhältnis.

Das FPGA-Board LOGIPi von *ValentFx* beherbergt einen Spartan 6 von *Xilinx* und bietet 24 I/O-Pins auf PMOD-Buchsen. Mit diesem Board bekommt der angeschlossene Raspberry Pi die Möglichkeit parallele Datenverarbeitung in programmierbarer Logik durchzuführen.

Aufgrund der Wahl für die Stage 2 wird für die Stage 3 das gleiche Prozessorboard verwendet. Hier kommt ebenfalls ein Raspberry Pi zum Einsatz.

6.4 Vergleich des Konzeptes mit dem Stand der Technik

Der Vergleich des entwickelten Konzeptes mit den am Markt verfügbaren Systemen.

Das neu entwickelte System verwendet digitale MEMS-Mikrofone. Daneben wird ein Beschleunigungssensor in dem System neben den zwei Mikrofonen integriert. Es ist in dem entwickelten Konzept möglich eine Verrechnung der drei Sensorsignale auszuführen und Fehler oder Störungen eines Sensors herauszufiltern. Diese Möglichkeit bietet keine am Markt verfügbare Lösung.

Es werden die angeschlossenen Sensoren in Untereinheiten zusammengeführt. Nach der Zusammenführung werden die Sensordaten an die darüber liegende Ebene weitergereicht. Mit diesem Aufbau ist es möglich einzelne Untereinheiten flexibel in das System zu integrieren um dabei den Anforderungen der Messaufgabe gerecht zu werden. Eine solche Flexibilität beim Aufbau des Systems bietet nur das digitale System von *CAE,* welche in der Narbe modular Sensorarme aufnimmt.

Neben der Integration weiterer Sensoren ist es ebenso möglich, Sensoren oder Untereinheiten zu entfernen. Durch die Definition einzelner Untergruppen ist es möglich die benötigten Verbindungen kurz auszulegen und nur eine einzelne Verbindung an darüber liegende Ebene weiterzugeben. Bei den bestehenden analogen werden die einzelnen Sensorsignale in einer zentralen Einheit zusammengeführt. Änderungen im Aufbau sind nur durch Veränderung dieser Einheit möglich. Daneben ist der Aufwand der Kabelführung deutlich größer, denn alle angeschlossenen Sensoren haben eine exklusive Verbindung zur zentralen Einheit. Ist die räumliche Ausdehnung des Systems groß steigt der Aufwand weiter. Über die konkrete Struktur der Verbindungen liegen bei den digitalen Systemen keine Informationen vor.

Das entwickelte Konzept sieht neben der Verwendung als reine Schallmesstechnik auch eine Verwendung zur Körperschallmessung vor. Es ist möglich nur die integrierten Körperschallsensoren auszuwerten. Dieses ist wie bei der Luftschallmessung mit mehreren Kanälen und bei einer großen räumlichen Ausdehnung möglich. Eine Integration anderer Sensoren zur Erfüllung anderer Messaufgaben ist bei dem bestehenden System nicht vorgesehen, hier sind nur Mikrofone verwendbar.

Die Auswertungshardware lässt sich fest in der Arraystruktur integrieren. Somit ist es nicht notwendig ein externes Gerät aufzustellen. Durch die Verwendung von kleinen stromsparenden Einheiten ist es möglich ein System mit deutlich geringerer Leistungsaufnahme zu bauen und das Gesamtsystem über Stunden mit handlichen Akkumulatoren zu versorgen. Eine Überschlagsrechnung mit einem Array von 48 Mikrofonen, mit 4 Prozessorboards (je 3W), 3 FPGA-Boards (je 1,5 W) und 48 Stage 1 Boards (je 70mV), ergibt eine Leistungsaufnahme von ca. 20W. Im bestehenden analogen System wird der größte Teil der Leistung, bis zu 200W, in der zentralen Einheit verbraucht. Verfügbare digitale Systeme haben für das Array einen Verbrauch von 20 W. Durch den Einsatz der beschriebenen Sensoren und der kleinen Einheiten lässt sich der Gesamtpreis des Systems, auf ca. 2.000,- € für die Hardware reduzieren.

7 Realisierung

7.1 Multisensorboard Entwicklung (Stage 1)

Die Entwicklung des Multisensorboards wurde auf der Grundlage der in Kapitel 5 formulierten Anforderungen durchgeführt. Das Board soll dabei alle notwendigen Sensoren und eine Einheit zur Verarbeitung der eintreffenden Signale beherbergen. Die entwickelte Struktur der Stage 1 wird in Abbildung 27 dargestellt. Es werden die verwendeten Sensoren und der Prozessor mit den unterschiedlichen Verbindungsbussen abgebildet. Die gezeigte Ebene der Signalausgabe befindet sich nicht, oder nur bei Bedarf, auf der Platine.

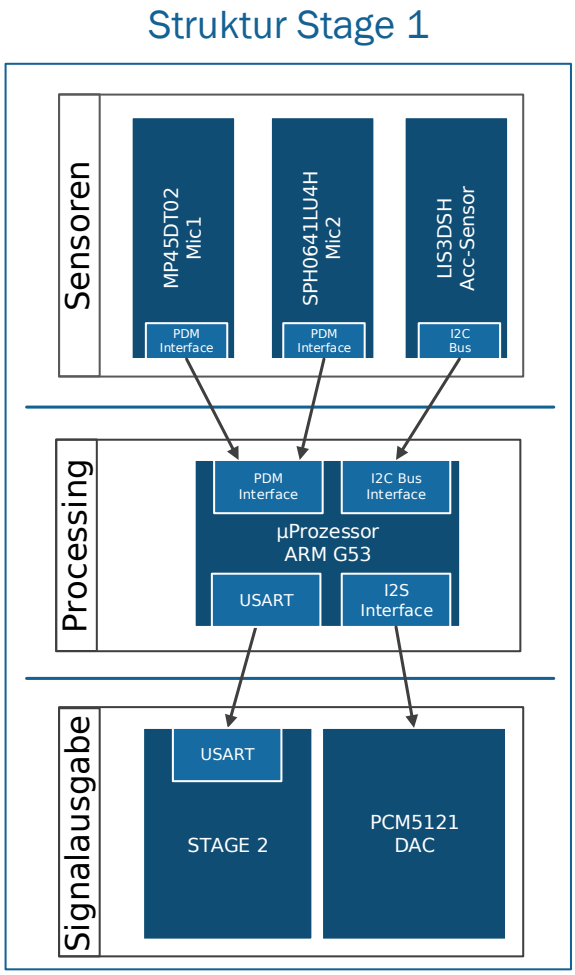


Abbildung 27: Struktur der Stage 1

7.1.1 Verbindungsbussystem auf dem Multisensorboard

Die Verbindung vom Prozessor zum Beschleunigungssensor wird über eine I2C-Schnittstelle realisiert. Diese wird zur Konfiguration und zur Übertragung der Daten genutzt.

Die Mikrofone sind nicht konfigurierbar. Der Ausgang der Mikrofone ist ein auf das Taktsignal modulierter Pegelwert des gemessenen Schalls.

Der optionale Audio-Digital-Analog-Wandler ist über eine Stiftleiste verbunden, diese kann bei Bedarf verlötet und der DAC angeschlossen werden. Die

Konfiguration des DAC wird über eine I2C-Schnittstelle durchgeführt, die Audiodaten werden über die I2S-Schnittstelle übertragen. Die I2S-Schnittstelle bietet dabei den Vorteil der kontinuierlichen Übertragung von zwei separaten Audio-Kanälen. Weitere Informationen zu den Schnittstellen bitte dem Datenblatt des PCM5121-Audio Stereo DAC [14] entnehmen.

7.1.2 Boarddesign

Das Borddesign wurde mit dem Ziel entworfen, die geforderten Komponenten geeignet zu platzieren und dabei eine möglichst kleine Fläche zu benutzten. Weiter wurden die Mikrofone isoliert von den anderen Komponenten am oberen Rand der Platine platziert. Es wurden die oberen Ecken der Platine frei gelassen um die Möglichkeit zu haben diese nachträglich zu entfernen. Mit dieser Maßnahme stehen die Mikrofone auf einem schmalen Steg im Raum und haben damit bessere akustische Eigenschaften. Die Reflektionen der umliegenden Platinenfläche mit dieser Maßnahme verringert werden.

Der schematische Entwurf der Stage 1 ist in Abbildung 27 gezeigt. Darin zu sehen sind die Sensoren welche die Messgrößen aufzeichnen und über die entsprechenden Schnittstellen an den Prozessor weitergeben. Der Prozessor nimmt die Sensorsignale über die Schnittstellen auf und verarbeitet diese. Der DAC befindet sich optional auf der Stage1, ist aber in Abbildung 27 dargestellt. Aus dem Prinzipentwurf wurde der in Anhang C gezeigte Schaltplant entwickelt. Dieser enthält, neben den beschriebenen Einheiten, die benötigten passiven Bauelemente und Verbindungselemente. Hervorzuheben sind die Programmier- und Debugschnittstelle über den JTAG-Standard, sowie die Schnittstelle zur Stage 2. Diese wird in Abbildung 28 dargestellt, die Pins haben die folgender Tabelle aufgeführten Funktionen.

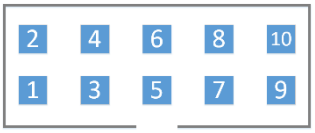


Abbildung 28: Connector zu Stage 2

PIN	Funktion
1	GND
2	VCC
3	TDX- USART
4	RDX- USART
5	SCK- USART
6	CTS- USART
7	Ext. Interrupt
8	Ext. Interrupt
9	GPIO
10	GPIO

Tabelle 6: Pinfunktionen

Die Pins 7 bis 10 haben keine direkt zugewiesene Funktion, werden jedoch bei dem Debuggen des Programms als GPIO nützlich sein.

Die größte Herausforderung des Platinendesigns bestand in der geeigneten Platzierung der Entstörkondensatoren. Die Kondensatoren wurden mit Hilfe der in der Application Note des Herstellers [15] beschrieben Hinweise platziert. Es wurde auf die Nähe der einzelnen Kondensatoren zu dem jeweiligen Versorgungspin am Prozessor geachtet. Neben den Kondensatoren wurden die Programmierschnittstelle, der Resetpin und die Spannungsversorgung nach den Beschreibungen von *Atmel* in [15] aufgebaut:

In der Abbildung 29 wird das Design im Editor aufgeführt. In diesem Bild sind folgende Baugruppen mit den entsprechenden Nummern gekennzeichnet.

1. Prozessor ATSAMG53 mit umliegenden Entstörkondensatoren
2. Die Mikrofone mit Lötpads zur beidseitigen Montage
3. Der Connector zur Stage 2, USART, Versorgungsspannung und ext. Interrupts
4. Die JTAG-Programmierschnittstelle mit benötigten PullUp-Widerständen
5. Der Beschleunigungssensor mit benötigten PullUp-Widerständen
6. Unterhalb vom 6 der Connector zum DAC, mit I2S, I2C und Versorgungsspannung

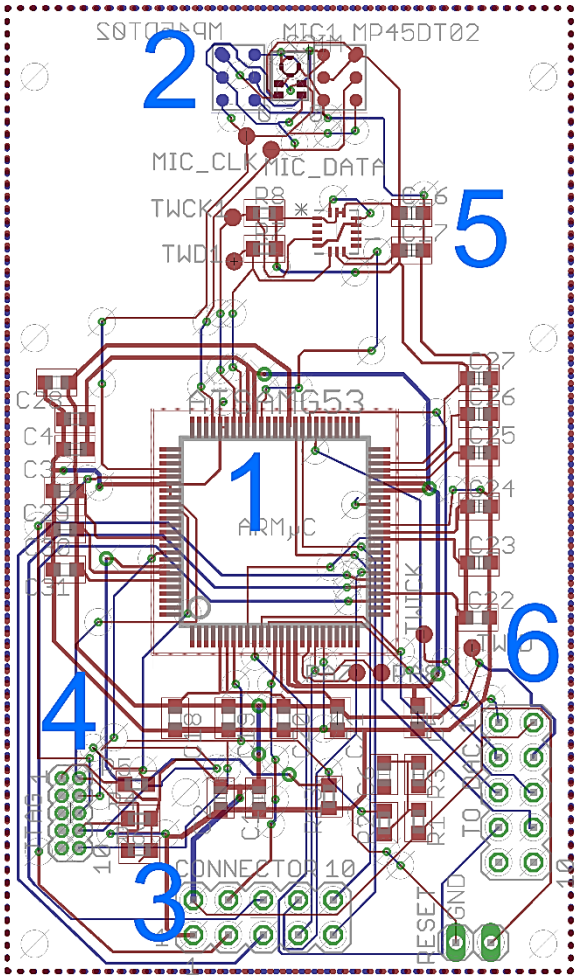


Abbildung 29: MSB im Designeditor

Neben der Platzierung aller Komponenten wurde auf die Möglichkeit einer nachträglichen Korrektur der einzelnen Leitungen geachtet. Dazu wurde der Platz zwischen den einzelnen Komponenten groß genug gehalten. Der dadurch vergrößerte Zwischenraum ermöglicht außerdem eine einfachere Platzierung der Bauteile beim Bestücken.

7.1.3 Hardwareaufbau

Die Platinen wurden nach dem in 7.1.2 beschriebenen Design, bei *BetaLayouts*®, einem Auftragsfertiger von Platinen, gefertigt. Neben den Platinen wurde für jede Platinenseite eine Rakelmaske bestellt. Diese Masken haben

an allen Stellen an denen Lötpaste aufgetragen werden muss eine Aussparung. Mit der Maske wird Lötpaste gezielt auf der Platine verteilt.

Der Aufbau der Platinen wurde in folgenden Arbeitsschritten durchgeführt:

1) Reinigung der Platinen
2) Positionierung der Rakelmaske über den Lötpads
3) Fixierung der Rakelmaske
4) Auftragen der Lötpaste und abnehmen der Rakelmaske
5) Platzierung der Bauteile
6) Verlöten der Bauteile mit Heißluft
7) Eventuelle Korrekturen oder Nacharbeiten
8) Anbringen der Stiftleistungen mit einem konventionellen Lötkolben

Es wurden insgesamt 11 Platinen bestellt und aufgebaut. Auf der ersten Platine wurde nur der Prozessor verlötet um einen ersten Prototyp und ein Programmierziel zu erhalten. Die Abbildung 30 zeigt im linken Bild die Rohplatine und im rechten die fertig aufgebaute Platine. Auf dem rechten Bild ist die freie Stiftleiste zum Anschluss des DAC am unteren rechten Rand der Platine zu sehen.

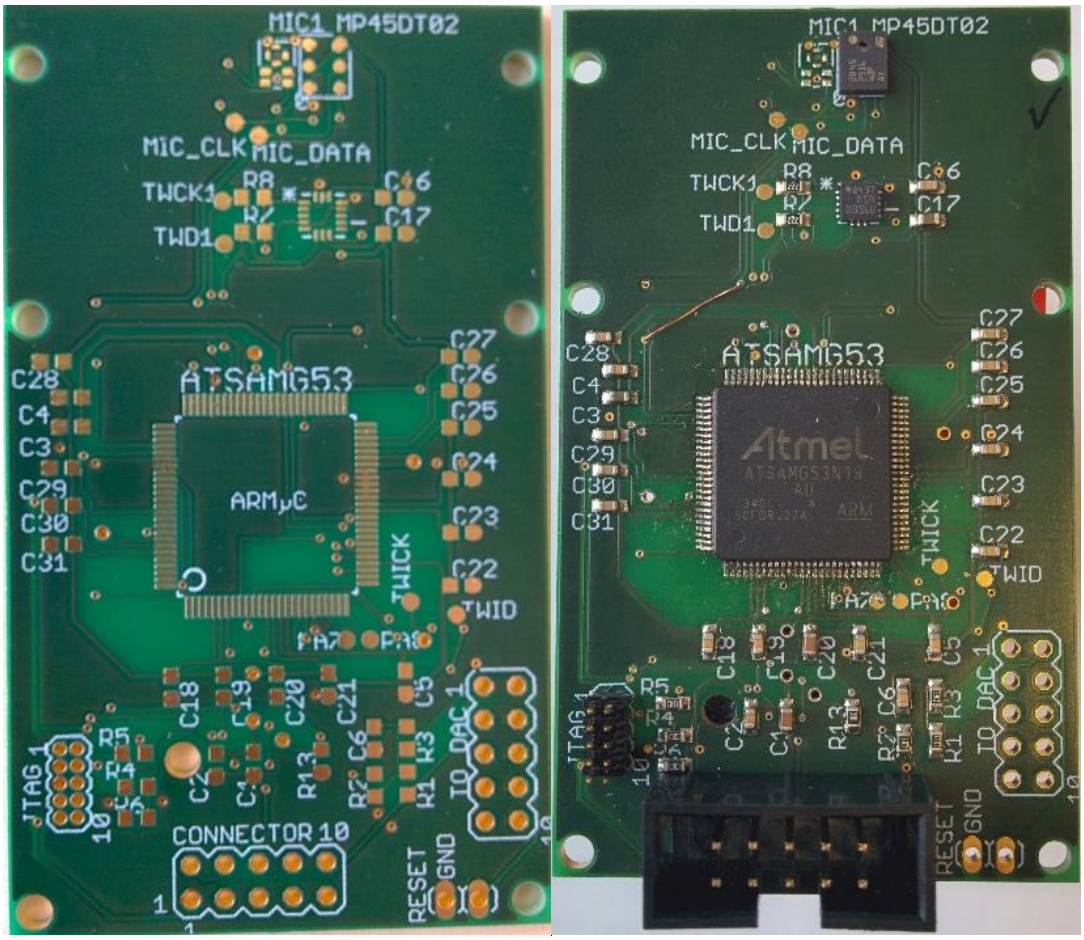


Abbildung 30: Aufbau und Lötung der Platinen (l) Rohplatine (r) fertige Platine

Beim Löten der einzelnen Boards sind trotz der gründlichen Arbeitsweise bei dem Prozessor Lötbrücken entstanden, das bedeutet es wurden mehrere Pins des Prozessors kurzgeschlossen, dies ist in Abbildung 31 in Rot umkreist. Diese Brücken mussten nach dem Löten entfernt werden. Bei allen aufgebauten Platinen entstanden Lötbrücken beim Löten mit Heißluft.

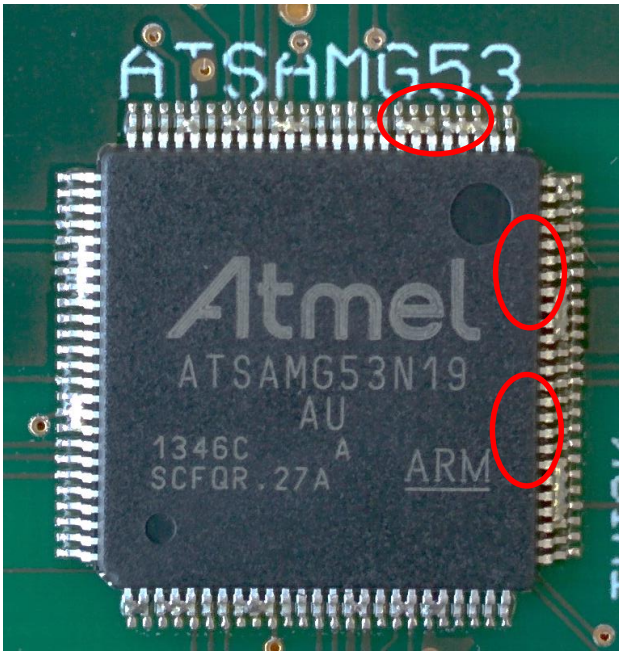


Abbildung 31: Verlöteter Prozessor mit Lötbrücken

7.1.4 Programmierung der Stage 1

Die Programmierung des Prozessors auf Stage 1 erfolgt in C. Dabei wurden mit der Hilfe des Atmel Software Frameworks (ASF) die Hardwareblöcke angesprochen. Das ASF bietet die Möglichkeit für bestimmte Hardwareblöcke spezifische Initialisierungsroutinen und Konstantendefinitionen einzubinden und zu verwenden. Dadurch wird der erzeugte Quellecode deutlich übersichtlicher und leichter nachvollziehbar.

Das Programm auf Stage1 arbeitet nach dem in Abbildung 32 gezeigten Ablaufdiagramm. An der HAW Hamburg ist ein Schallmessraum vorhanden, in welchem Referenzmessungen durchgeführt werden können. Aus diesem Grund wurde ein Programm zur Messung von Luftschall implementiert. Dieses ist mit den beschriebenen Möglichkeiten an der HAW später auf seine Leistungsfähigkeit zu überprüfbar.

Das Programm beginnt mit der Initialisierung des Prozessors, hier werden z.B. die Clockquelle, der Tankt und die Pin-I/O-Portmap und die entsprechenden

Register geladen. Dazu werden die Funktionen, sysclk_init(), system_board_init() und ioport_init() ausgeführt.

Im nächsten Schritt werden die verwendeten Hardwarekomponenten initialisiert, zu diesen zählen das PDM-Interface, die USART-Schnittstelle, die I2C-Schnittstellen und die Konfigurationen für die Sensoren. Hierbei wird die Taktrate des PDM-Taktes mit 3,072 MHz eingestellt.

Das USART-Modul wird als SPI-Schnittstelle konfiguriert. Damit ist eine einfache Anpassung des Bustaktes auf die Gegebenheiten möglich.

Die Ausgabe der Mikrofonsignale erfolgt bei einem Interrupt des USART-Moduls. Dieser Interrupt wird bei einer fallenden Flanke der CS-Leitung ausgelöst.

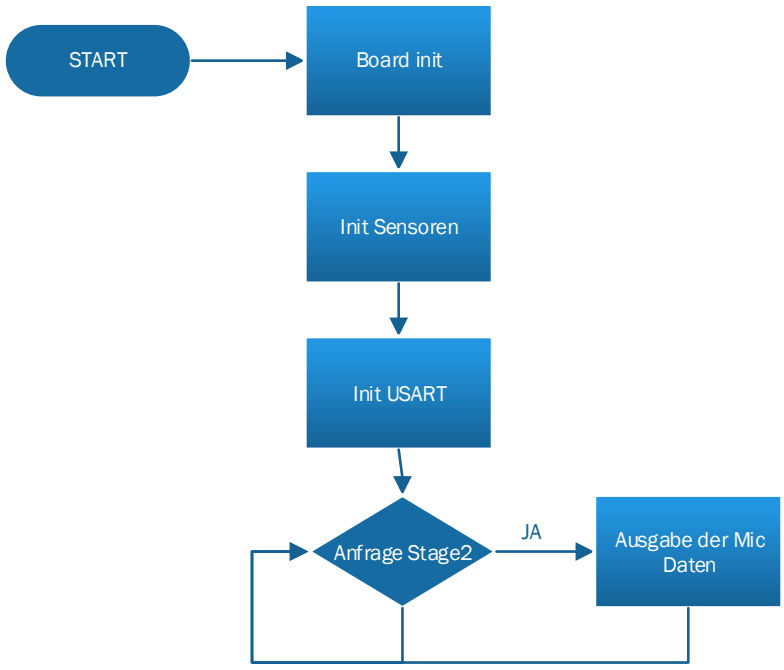


Abbildung 32: Ablaufdiagramm Stage 1

Eine Implementierung zum Auslesen der Daten des Beschleunigungssensors ist ebenfalls in Anhang E gezeigt. Die Daten des Beschleunigungssensors

werden darin in festen Zeitabständen abgefragt und über die USART-Schnittstelle weitergeleitet. Dieses Vorgehen ist notwendig, da der Sensor nur auf die Anfrage mit einem bestimmten Befehl Daten übermittelt.

7.1.5 Hardwaretests

Nach dem Aufbau der Platinen und der Erstellung des Programms wurde die Funktion der Stage 1 überprüft. Dazu wurde im ersten Schritt die bereitgestellte Versorgungsspannung für den Prozessor gemessen. Im nächsten Schritt wurde der JTAG-Programmer angeschlossen und das in Kapitel 7.1.4 entwickelte Programm aufgespielt. Nach der erfolgreichen Programmierung des Prozessors wurden die Schnittstellen zu den Sensoren und der Stage 2 mit einem Oszilloskop auf das richtige Verhalten untersucht. Nachdem alle Komponenten einer Platine das richtige Verhalten gezeigt haben, ist die Platine weiterverwendet worden. Bei diesen Tests haben, neben der zum Testen ohne Sensoren bestücken Platine, drei weitere nicht das richtige Verhalten gezeigt. Bei der Fehleranalyse dieser drei Platinen konnte der Prozessor nicht angesprochen werden. Dieser wurde vermutlich durch eine falsche Löttemperatur oder –Dauer beschädigt.

Das SPH0641LU4H Mikrofon sendet keine Daten, dies ist in Abbildung 33 gezeigt. Die eingesetzten Lineale zeigen die Veränderung auf der steigenden Flanke des Clocks. Auf die fallende Flanke, auf der das SPH0641LU4H Mikrofon konfiguriert ist, finden keine Pegeländerungen im blauen Graphen statt. Somit ist das zweite Mikrofon nicht verwendbar.

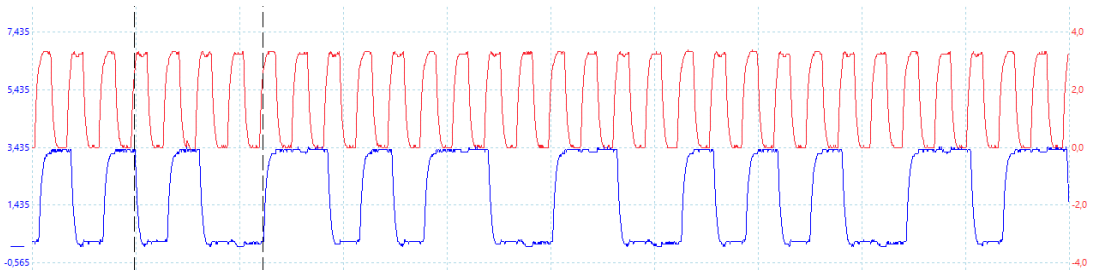


Abbildung 33: Fehlerhaftes Verhalten des Ultraschallmikrofons

7.2 Signalübertragungspfadentwicklung (Stage 2)

Die Stage 2 befindet sich eine Ebene über der Stage 1 und nimmt Verbindung mit mehreren Stages 1 auf. Dazu wird die in Abbildung 34 gezeigte Struktur implementiert. Die Struktur zeigt das Verbindungsnetzwerk zum Einlesen der Werte aus den angeschlossenen Stages 1. Dabei werden jeweils zwei Stages 1 von einem SPI-Master im FPGA bedient. Die empfangenen Signale werden im Anschluss zusammengeführt, um diese an den Raspberry Pi weiterzugeben.

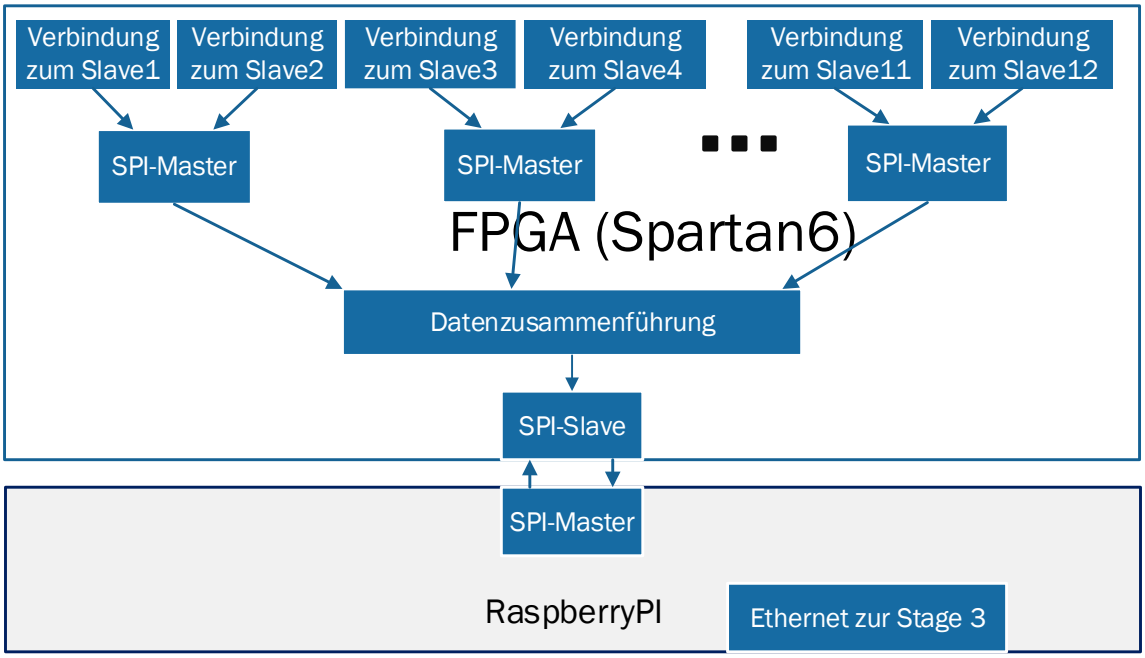


Abbildung 34: Struktur der Stage 2

Auf Grund der Möglichkeit des FPGA eine große Anzahl digitaler Schnittstellen parallel in Hardware zu realisieren, wird der FPGA als Schnittstellenboard verwendet. Durch die Verwendung geeigneter Logik auf dem FPGA ist es möglich die benötigten SPI-Master auf den I/O-Pin abzubilden.

Die Abbildung 35 zeigt das schwarze FPGA-Board fest mit dem Raspberry Pi verbunden. Der Vorteil dieser Lösung ist die gute Kompatibilität der beiden Komponenten und die elektrische und mechanische Verbindung zwischen den

Boards. Die Programmierung des FPGA wird über den Raspberry ausgeführt. Dieser lädt das synthetisierte bit-File auf den FPGA.

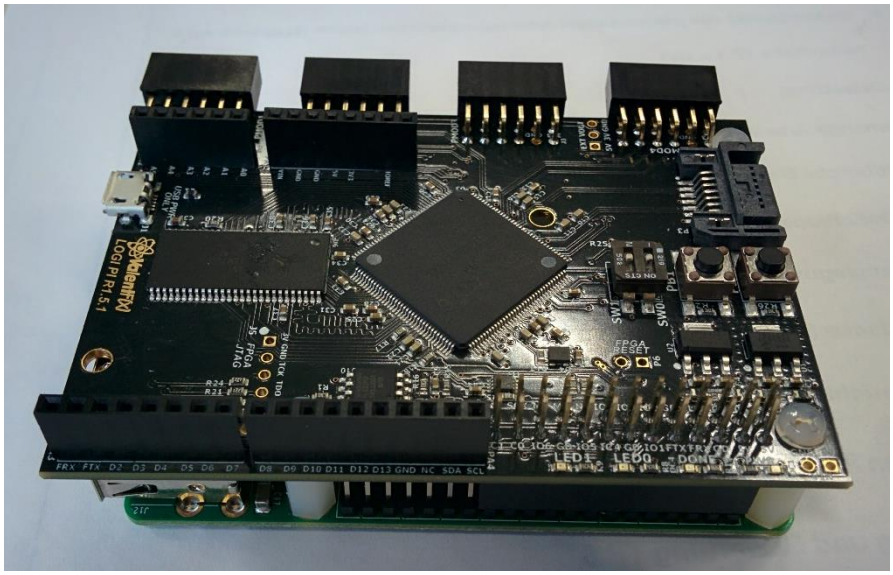


Abbildung 35: FPGA-Board auf dem Raspberry Pi

Auf dem FPGA werden nach Abbildung 34 6 SPI-Master mit einer Taktfrequenz von 2 MHz implementiert. Es werden 16 bit pro Sample von jedem Slave empfangen, mit je 2 Slaves pro Master sind 32 bit pro Sample und Master zu empfangen. Die Audiosamplerate beträgt 48kHz, das bedeutet: *48kHz * 32 bit = 1,536MHz* Mindesttakt. Mit den 2 Mhz wird ein ausreichend großer Abstand zum Mindesttakt erreicht.

Durch die hohe Abtastrate werden immer garantiert 2*16bit mit dem Audiotakt von 48kHz empfangen. Die Synchronität aller Slaves, in Form der Stages 1, wird durch die durch das Design garantierte Synchronität sichergestellt. Das Einlesen der Daten erfolgt wie beschrieben in 16 bit Blöcken. Diese werden abwechselnd von jedem Slave einzeln eingelesen.

In Abbildung 36 wird eine Aufnahme mit einem Ozilloskop eines SPI-Masters dargestellt. Der Master stellt den Takt (blau) und das Slave-Select Signal (rot) bereit. Das gezeigte grüne Signal sind statische Daten die zu Testzwecken von zwei Stages 1 auf den Bus gelegt wurden.

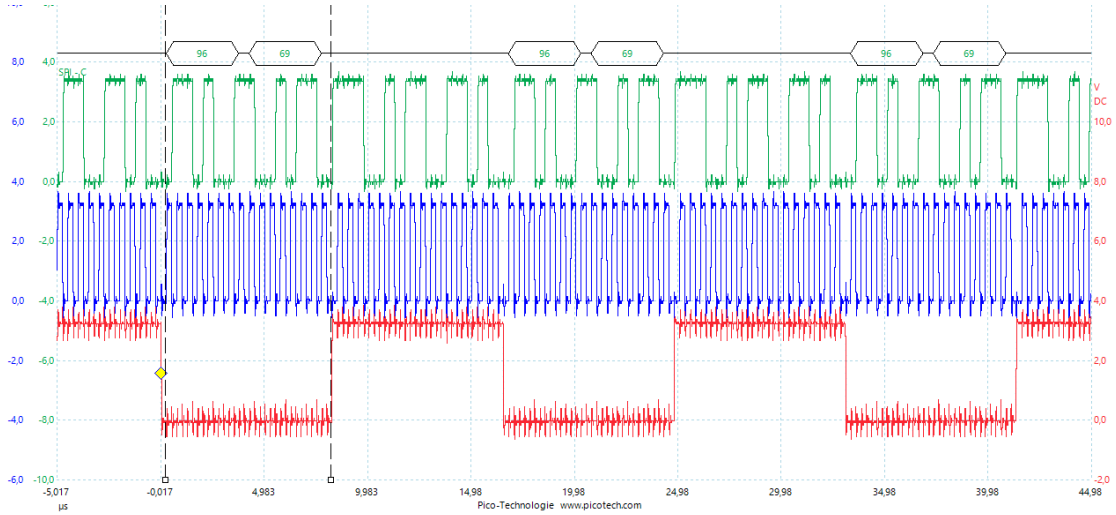


Abbildung 36: SPI-Master mit statischen Daten

Pro Master werden fünf Pins belegt. Diese Lauten SCK, MISO, MOSI, CS0, CS1 und definieren jeweils alle Pins die für einen SPI-Master benötigt werden. Mit der Verwendung von fünf Pins pro SPI-Master bleiben mit sechs Mastern bei 32 verfügbaren Pins auf den PMOD-Buchsen zwei Pins übrig, welche für andere Zwecke verwendet werden können. Eine Stage 2 mit dem gezeigten Aufbau nimmt Verbindung zu 12 Slaves, in Form von Stages 1, auf.

In folgender Abbildung 37 wird ein Aufbau von acht Mikrofonen in einem festen Array mit äquidistanten Abstand von 70mm gezeigt. Das Array lässt eine Untersuchung eintreffender Schallwellen auf ihre Richtung zu. Das dargestellte Stativ dient der besseren Standfestigkeit und der Möglichkeit das Array auszurichten. Die in dieser Abbildung nicht erkennbare Stage 2 ist in Abbildung 38, zusammen mit der Adapterplatine, dargestellt. Die Adapterplatine stellt für jedes Board der Stage 1 die Spannungsversorgung und die SPI-Schnittstelle exklusiv, über den in Kapitel 7.1.2 beschrieben Steckverbinder, bereit. Dabei erhält jeder Slave ein Slaveselect Signal. Die Spannungsversorgung wird von einem externen Stecknetzteil bereitgestellt.

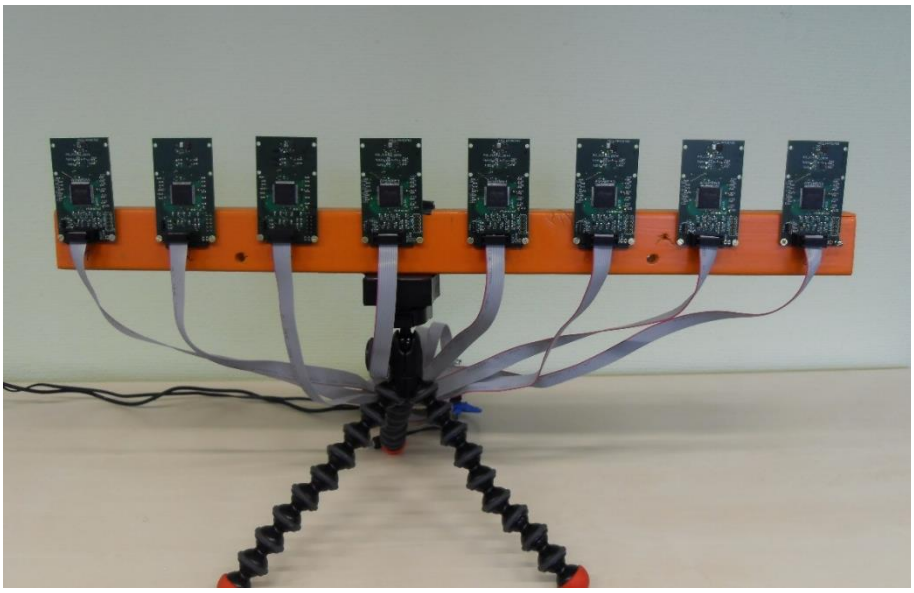


Abbildung 37: Acht Mikrofonen Array auf Stativ

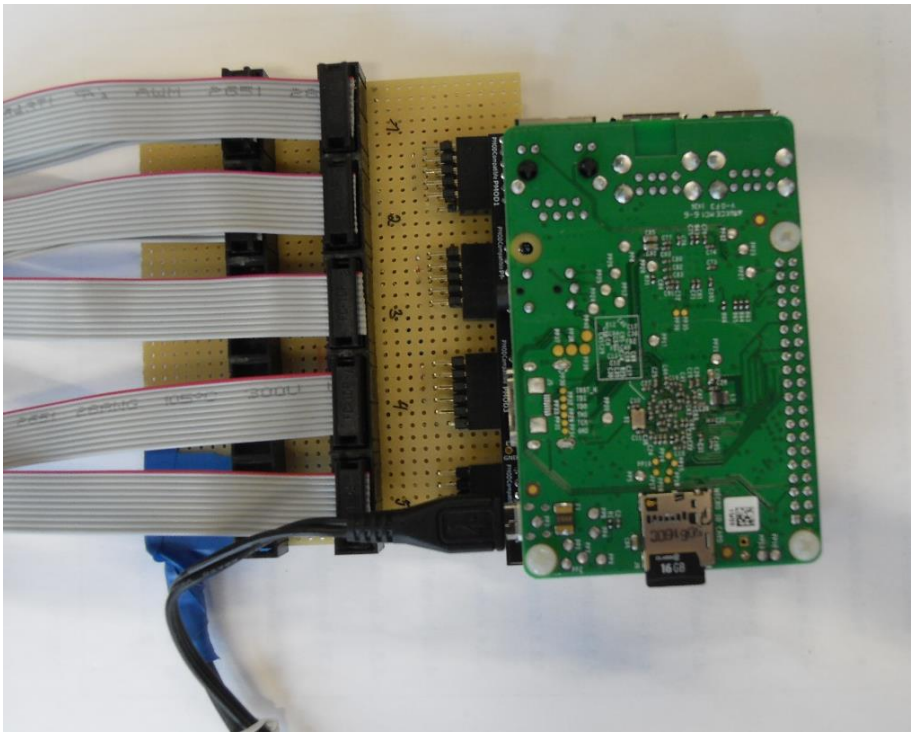


Abbildung 38: Stage 2 mit angeschlossenem Adapterboard zu Stage1

7.2.1 Sensorsignalzusammenführung

Die Sensorsignale werden nach dem Einlesen durch die SPI-Master in 16bit Blöcken der Reihenfolge nach hintereinander angeordnet. In dieser Anordnung werden diese über die SPI-Schnittstelle an den Raspberry Pi übertragen. Bei dieser Schnittstelle agiert der FPGA als SPI-Slave und der Raspberry Pi als SPI-Master.

Nach der Übertragung der Daten auf den Raspberry Pi stehen diese in Form der Rohdaten zu Verfügung.

Das auf dem Raspberry Pi implementierte C-Programm versieht jedes Sample mit einem Zeitstempel. In einem Sample sind die Daten aller angeschlossener Mikrofone gespeichert. Die Zeitstempel enthalten die von der Stage 3 verteilte Systemzeit.

7.3 Signalverarbeitung und -Speicherung (Stage 3)

Die Hauptaufgabe der dritten Stage ist die Synchronisierung der darunterliegenden Ebenen. Dazu wird ein Master mit dem precision-time-protocol in der Version 2 (ptpv2, oder auch kurz PTP) installiert. Dieser setzt eine Systemzeit und verteilt diese an die angeschlossenen Teilnehmer. Dabei wird die vorhandene Ethernetschnittstelle der Raspberry Pis verwendet.

Das PTP Protokoll nutzt ein TCP/IP basiertes Netzwerkprotokoll zur Synchronisierung einer Zeit systemweit. Dabei sendet der Master adressierte Nachrichten an einen Teilnehmer und misst dabei die Laufzeit der Übermittlung und die Zeit bis zur Antwort des Teilnehmers. Beide Zeiten werden bei der Übermittlung der Systemzeit verrechnet, sodass die Teilnehmer die Zeit des Masters mit wenigen µs Genauigkeit annehmen können. Mit diesem Verfahren ist es möglich auch in einem komplexeren Ethernetnetzwerk mit mehreren Switches und vielen Teilnehmern eine Systemzeit auf allen Teilnehmern zu setzen.

Neben der Zeitsynchronisierung arbeitet die Stage 3 als Userinterface. Es ist möglich die Messungen auf den darunterliegenden Ebenen von der Stage 3 aus anzustoßen und mit allen Sensoren parallel und synchron Messungen aufzuzeichnen.

7.4 Kostenaufstellung

Folgend ist die Kostenaufstellung für ein System mit 48 Sensorsboards, in vier unabhängigen Arrays mit je 12 Sensorboards, aufgeführt.

Komponente	**Anzahl**	**Einzelpreis**	**Summe**
Raspberry Pi B+	5	22,04 €	110,20 €
LogiPi	4	112,49 €	449,96 €
Sensorplatinen	48	11,35 €	544,80 €
MP45DT02	48	1,54 €	73,92 €
ATSAMG53	48	5,28 €	253,44 €
LIS3DSH	48	2,17 €	104,16 €
passive Komponenten	48	2,00 €	96,00 €
Verbinder	48	0,84 €	40,32 €
Kabel	48	1,00 €	48,00 €
Ethernet	1	50,00 €	50,00 €
opt. SPH0641LU	48	2,91 €	139,68 €
	Gesamt:		1.910,48 €

Für den beschriebenen Aufbau, belaufen sich die Hardwarekosten auf 1.910,48 €. Bei den Einzelpreisen handelt es sich um Tagespreise, sodass eine Summe von 2.000,- € für 48 Kanäle angenommen wird.

8 Test und Validierung

8.1 Messaufbau

Es werden zur Darstellung der Funktionen der einzelnen Einheiten mehrere Messaufbauten erstellt. Dazu werden die Funktionseinheiten soweit möglich isoliert betrachtet.

Zur Darstellung einer Funktion eines einzelnen Boards der Stage1 wurde ein Board direkt ausgelesen. Dies bedeutet es wurde ein Board mit einem Raspberry Pi und dessen SPI-Schnittstelle ausgelesen. Bei diesem Aufbau liegt das Augenmerk auf den akustischen Eigenschaften der Mikrofone. Insbesondere ist die Korrelation mit dem in Abbildung 14 gezeigten Eigenschaften eines digitalen Mikrofons interessant.

Nachdem die Grundfunktion der Stage 2 in Abbildung 36 gezeigt wurde, wird eine akustische Messung im reflexionsarmen Raum durchgeführt. Bei dieser Messung wird die über die Phasenlage mit dem „delay-and-sum-Algorithmus“ eine Richtung der Schallquelle bestimmt. Vor der Anwendung des „delay-and-sum-Algorithmus“ wird über eine Prüfung der zeitlichen Verschiebungen überprüft ob die gemessenen Ergebnisse zu den Abständen der Mikrofone passen.

Der Funktionstest der Stage 3 beschränkt sich auf die Überprüfung der korrekten Funktion des PTP-Protokolls.

8.2 Messdurchführung

8.2.1 Messungen Stage 1

Die Messung eines einzelnen Multisensorboards wird in den folgenden Grafiken gezeigt. Zur Erzeugung eines Referenzsignals wird der Kalibrator mit den Pegeln von 94 dbSPL und 114 dbSPL verwendet. Dabei ist das Mikrofon gegenüber dem Kalibrator mit einer Dichtmasse abgedichtet. Dies entspricht nicht der Anbindung nach Herstellerangaben, bietet jedoch ein akzeptables Ergebnis. Der Hersteller schreibt eine durch einen Dichtring abgeschlossene Kammer zwischen Mikrofon und Kalibrator vor. Dazu wird das Mikrofon, wie in Abbildung 39, in die Öffnung des Kalibrators eingeführt.

Abbildung 39: Mikrofon wird kalibriert

Die Ergebnisse der Messungen sind in folgenden Grafiken dargestellt.

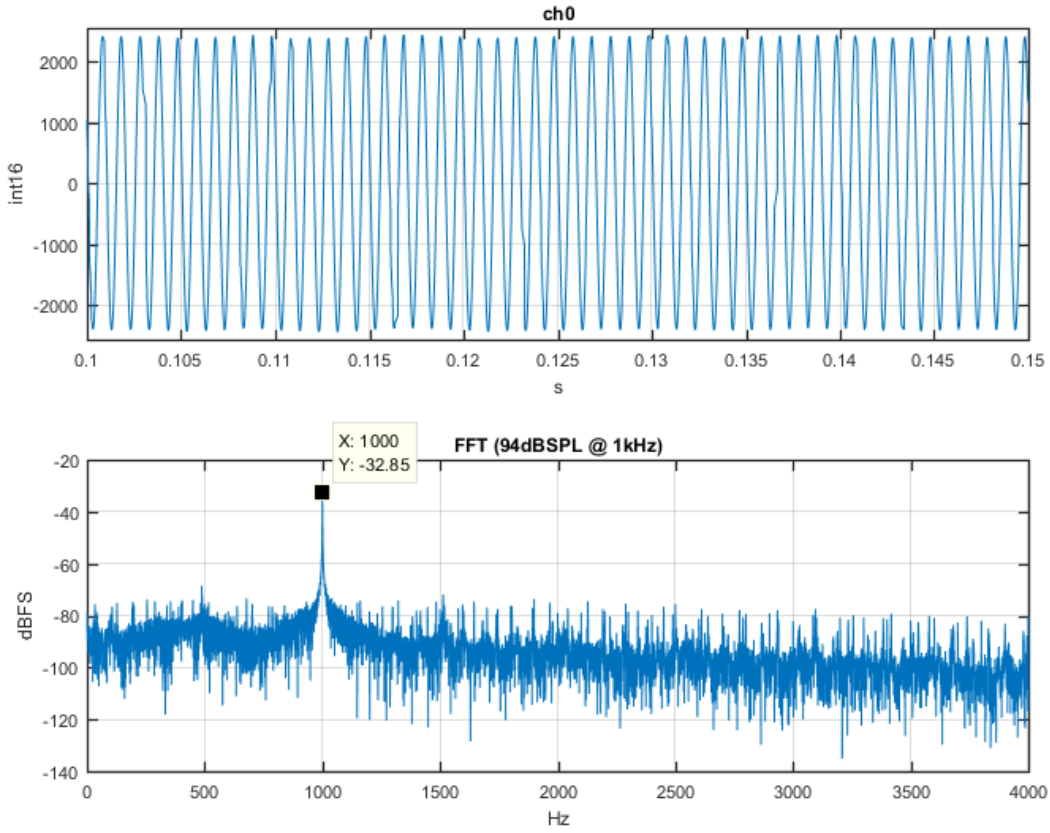


Abbildung 40: Einzelmikrofon mit Referenzsignal 94dB

In den Abbildungen zu sehen sind jeweils ein Ausschnitt der Zeitfunktion und die resultierende FFT. Die gezeigten Pegel in der FFT sind Vergleichbar mit den Pegeln der Abbildung 14. Die Pegel erreichen nicht die Werte aus der Abbildung. Es fehlen in beiden Fällen ca. 6dB, dies lässt sich auf die schlechtere Abdichtung der Mikrofone gegenüber dem Kalibrator und den größeren Abstand zwischen Mikrofon und Membran des Kalibrators zurückführen. Dadurch, dass die Membran nicht direkt wie in Abbildung 39 gezeigt vor die Schallerzeugende Membran des Kalibrators gehalten werden kann entsteht der größere Abstand.

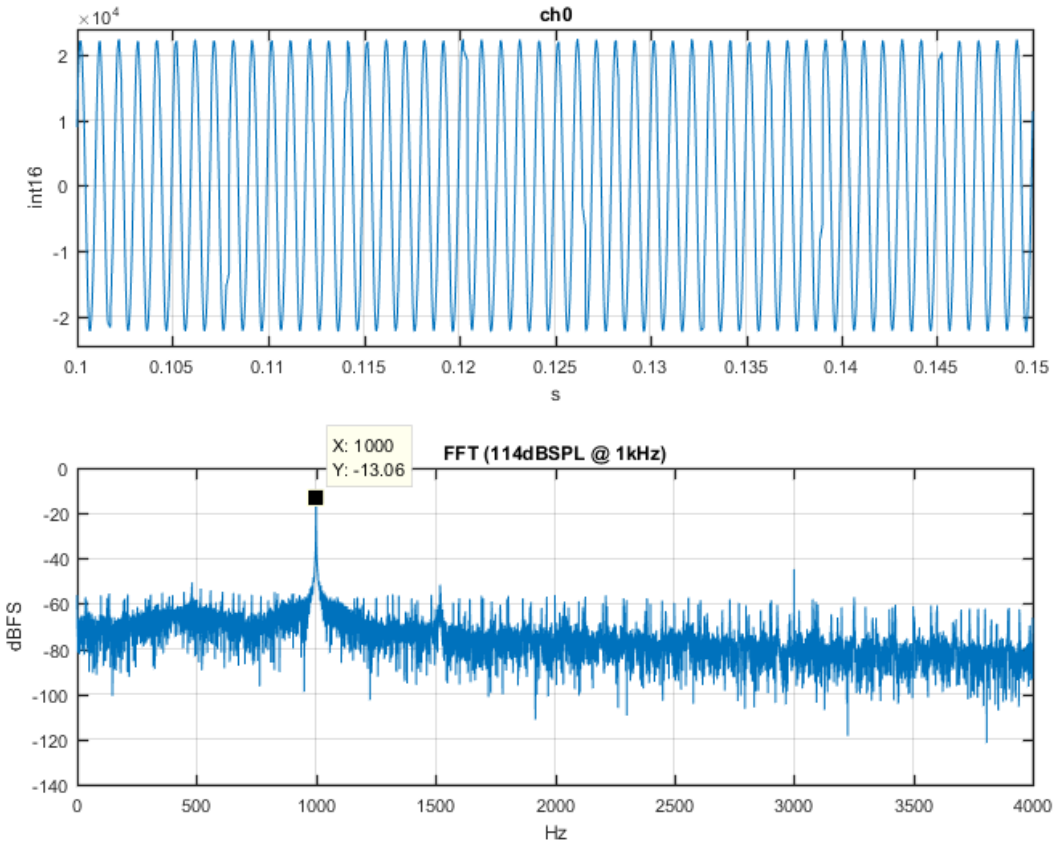


Abbildung 41: Einzelmikrofon mit Referenzsignal 114dB

Mit den gezeigten Messungen ist die Umrechnung der gemessenen Werte in Schallpegel möglich. Damit lassen sich die Mikrofone für den gezeigten Aufbau kalibrieren. Ein Schallpegel von 94dBSPL entspricht einem Wert von 2400. Ein Schallpegel von 114dBSPL entspricht einem Wert von 22300. Erwartet wurde bei der Erhöhung von 20dBSPL eine Verzehnfachung des resultierenden Wertes. Bei den Messungen wurde eine Erhöhung um den Faktor 9,3 erreicht und genügt mit den Randbedingungen der Annahme.

Es ist der Frequenzgang des Mikrofons aufgezeichnet worden. Dieser ist in Abbildung 42 oben gezeigt. Der blaue Verlauf in dieser Abbildung zeigt den gleichzeitig aufgezeichneten Frequenzgang eines Referenzmikrofons von

Brüel & Kjær vom Type 4190. Die Verläufe beider Mikrofone über die steigende Frequenz ähneln sich. In der unteren Grafik in der Abbildung ist die Differenz beider Mikrofone über die Frequenz dargestellt. Dazu wurde der Frequenzgang des Type 4190 als ideal angenommen, um eine Korrekturkurve für das entwickelte Mikrofon zu erhalten. Wird das Signal des entwickelten Mikrofons mit der Differenzkurve gefiltert, so entspricht der Verlauf dem des Referenzmikrofons.

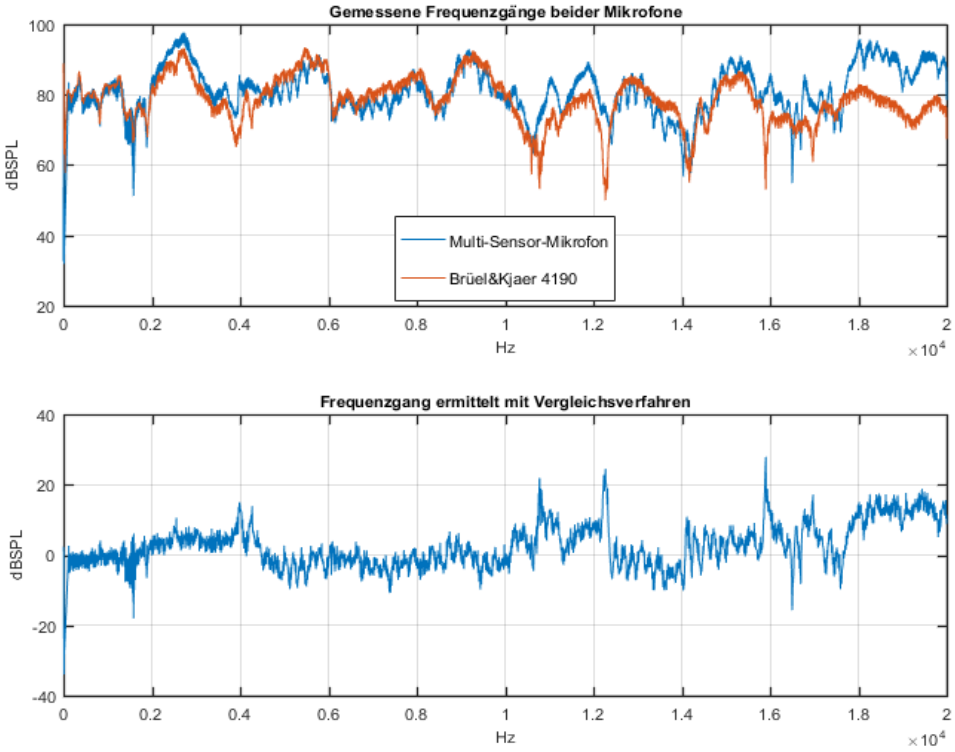


Abbildung 42: Aufgenommener Frequenzgang

Abbildung 43 stellt den Frequenzgang des Type 4190 Mikrofons von Brüel & Kjær aus dem Datenblatt dar. Es ist ein glatter Verlauf bis zu 10 kHz zu sehen, der Verlauf ist bei der durchgeführten Messung auf Grund der nicht linearen Schallquelle nicht erreicht worden.

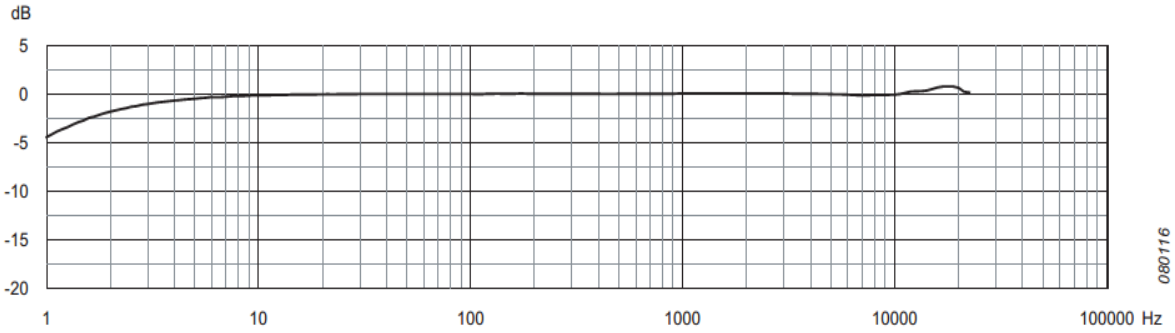


Abbildung 43: Frequenzgang der Mikrofons Type 4190 Brüel & Kjær [16]

Neben den Messungen mit den Mikrofonen wurde zum Nachweis der Funktion, eine exemplarische Messung mit dem Beschleunigungssensor auf der Stage 1 durchgeführt. Dabei ist der Sensor händisch gedreht worden um die drei Achsen des Sensors auszulenken. Die Messung ist in Abbildung 44 dargestellt.

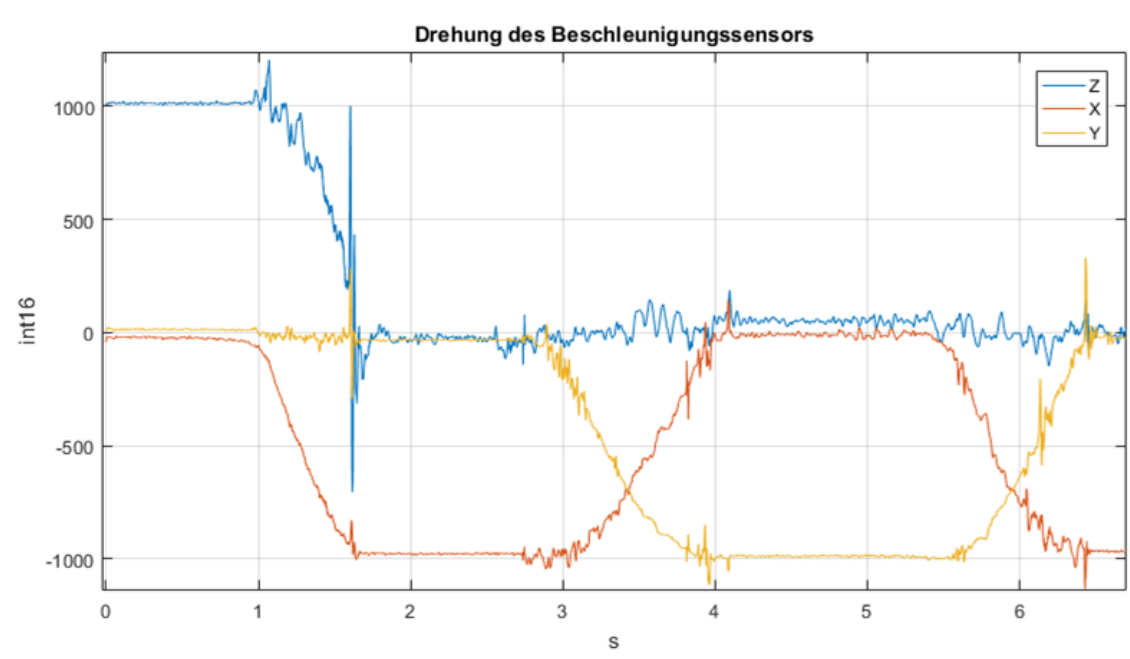


Abbildung 44: Beispielmessung des Beschleunigungssensors

Die Beispielmessung zeigt die Auslenkung der drei Achsen des Sensors, dabei ist der parallele Verlauf der Steigungen zu erkennen.

8.2.2 Messungen Stage 2

Die Messungen der Stage 2 sind zur Lokalisierung der Schallquelle durchgeführt worden. Dabei wurde das Array mit mehreren Winkeln zu der Schallquelle aufgestellt. Die Positionen der Messungen werden in Abbildung 46 aufgezeigt.

Die Abbildung 45 dient der Verdeutlichung des entstehenden zeitlichen Versatzes zwischen der Mikrofone des Arrays bei den Messungen unter einem Winkel von 45°. Die Schallwelle läuft mit der Schallgeschwindigkeit von 343 m/s über das Array von Mikrofonen und erreicht das Nächste durch den Abstand von 7 cm mit einer Zeitverzögerung von 144µs.

$$zeitl.\ Verschiebung = \frac{0{,}07m * \cos(45°)}{343\frac{m}{s}} = 144\mu s \qquad (8.17)$$

Bei dieser Rechnung wird ein ebenes Wellenfeld angenommen. Die Abstand „dist“ in Abbildung 45 ist die Strecke auf der das Signal verzögert wird.

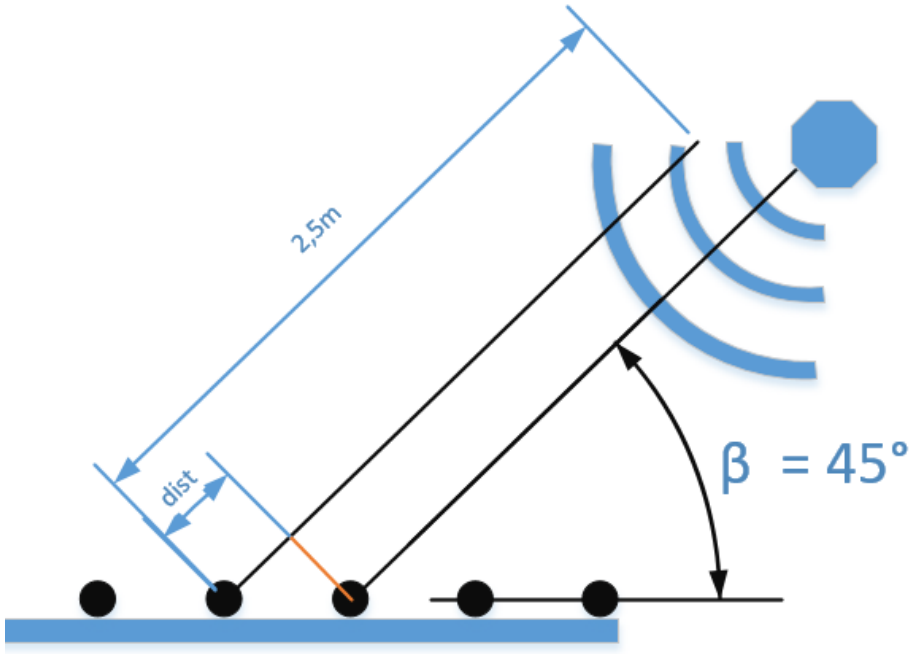


Abbildung 45: Eintreffender Schall auf das Array

Die Messungen im Schallmessraum, wie in Abbildung 47 dargestellt, sind mit den drei dargstellten Positionen durchgeführt worden. Die Position 2 dient als Referenz um eine Synchronität der Signalaufzeichnung zu gewährleisten.

Während der Messungen im Schallmessraum ist das Sensorboard 4 ausgefallen, sodass in den Abbildungen der Sensordaten nur sechs der sieben Aufgebauten Mikrofone zu sehen sind.

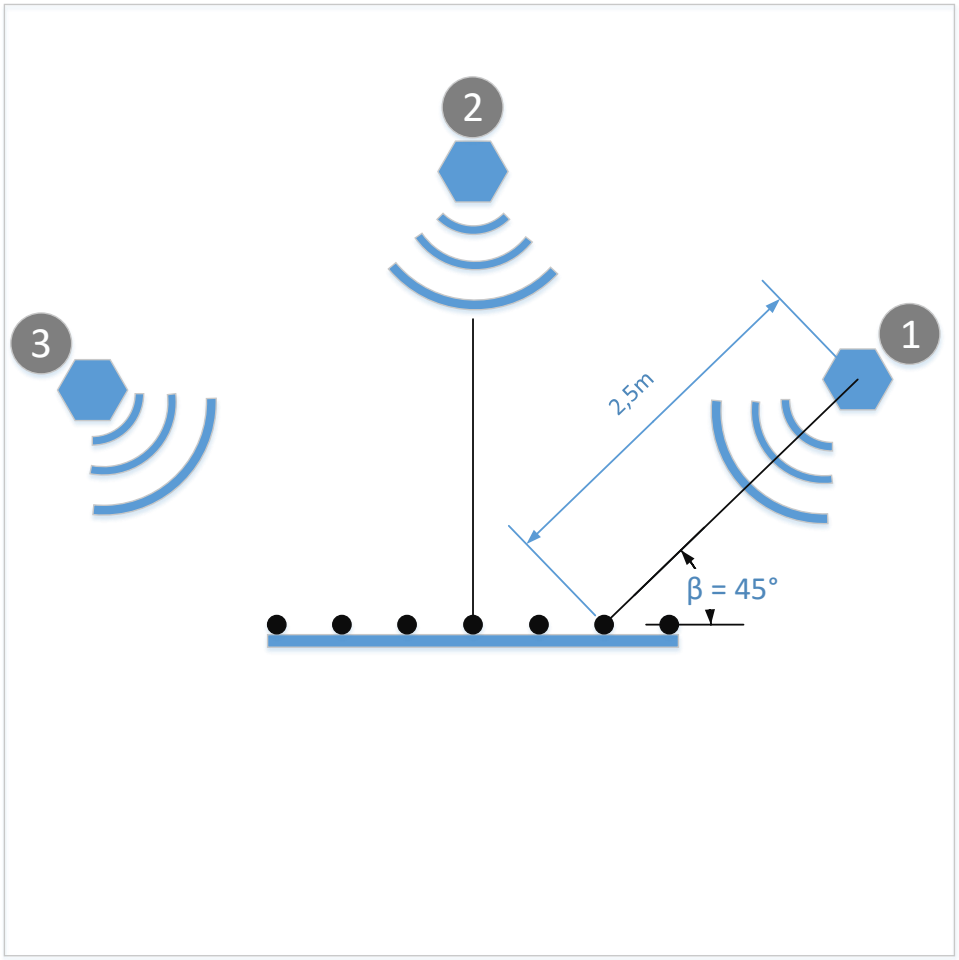


Abbildung 46: Schematische Darstellung des Messaufbaus

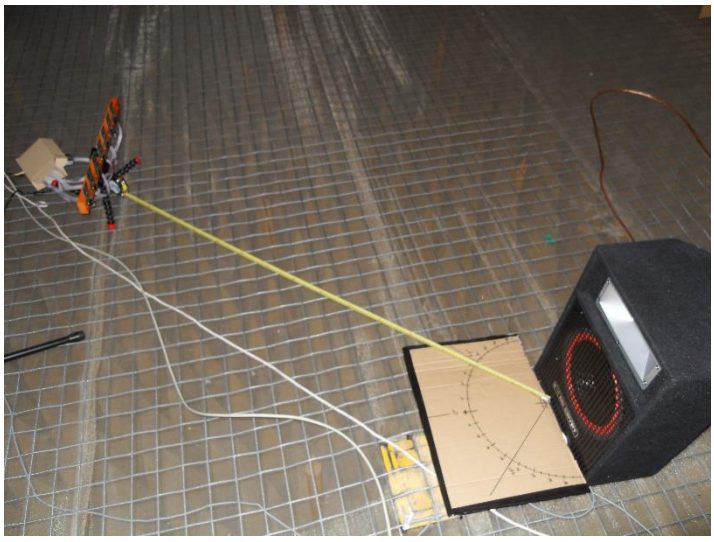


Abbildung 47: Aufbau im Schallmessraum (Pos1)

Die Synchronität der Signale ist in Abbildung 48 dargestellt. Zwei auf dem Array auseinanderliegende Mikrofone haben bei der Messung mit der Schallquelle auf Position 2 einen fast identischen zeitlichen Verlauf. Die Sensorsignale haben damit den erwarteten Verlauf bei einer orthogonal auftreffenden Schallwelle gezeigt. Ein zeitlicher Fehler bei der Aufzeichnung und der Weitergabe der Sensordaten ist nicht messbar.

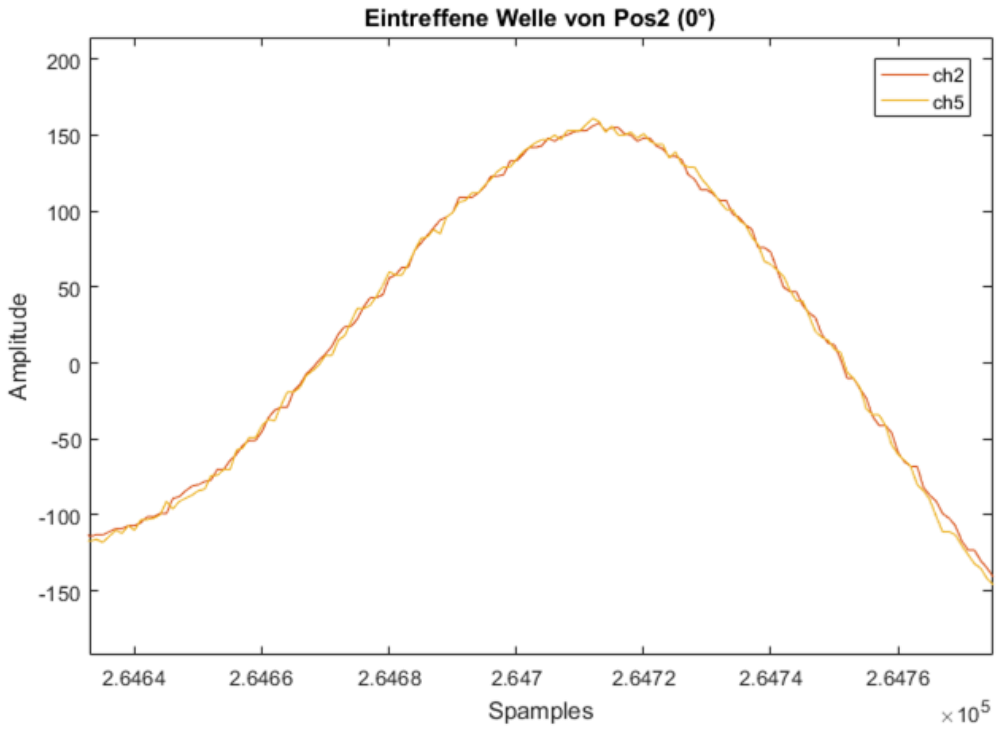


Abbildung 48: Eintreffende Welle Pos 2, Mikrofone 3 und 6

Nach der Messung auf der Position 2 wurde die Schallquelle auf die Positionen 1 und 3 versetzt. Auf dieser Position läuft die Welle, wie in Abbildung 45 dargestellt, über das Array und die Mikrofone zeichnen ein zeitlich versetztes Signal auf. Die von der Position 1 aufgezeichneten Signale sind in der Abbildung 49 und Abbildung 50 dargestellt. Es ist die Verschiebung in beiden Abbildungen zu erkennen.

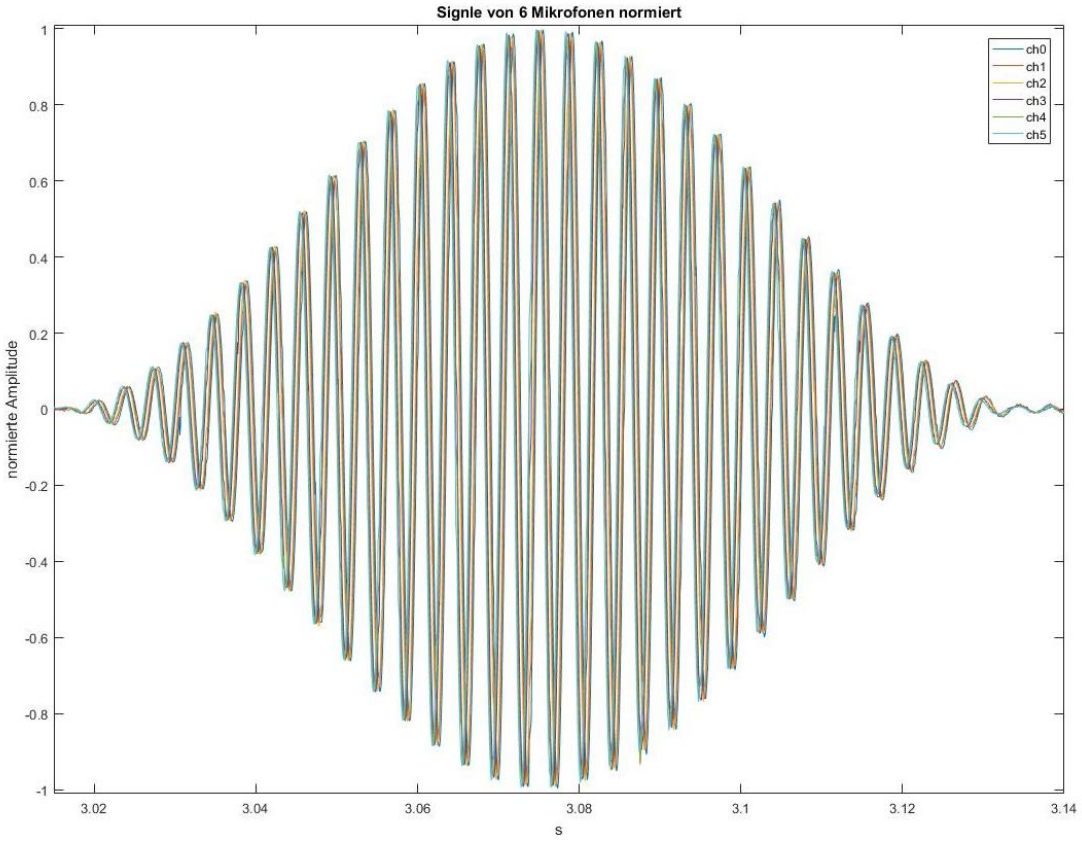


Abbildung 49: Darstellung der 6 Sensorsignale im Zeitbereich

Zur Bestimmung des Winkels unter der die Schallwellen eingetroffen sind, wird der „delay-and-sum-Algorithmus“ verwendet. Dazu werden die Mikrofonsignale im Auswerteprogramm zeitlich zu einander verschoben. Die Verschiebung bei der die Summe den größten Wert bildet, wird gespeichert und zur Berechnung des Winkels verwendet. Die mit dem beschriebenen Verfahren veränderten Signale sind in Abbildung 51 als Betrag dargestellt.

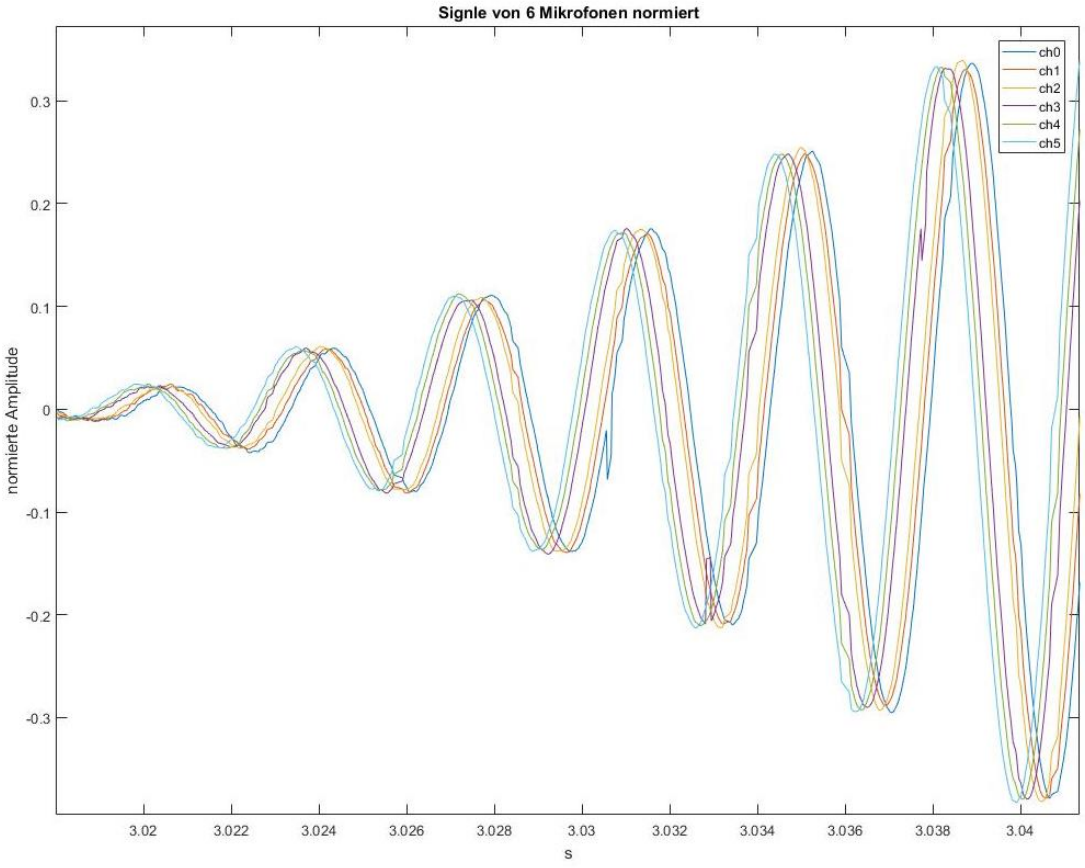


Abbildung 50: Darstellung der 6 Sensorsignale im Zeitbereich vergrößert

Mit folgender Formel wird aus der Verschiebung der Winkel β der eintreffenden Schallwelle berechnet:

$$\beta = \cos^{-1}\left(\frac{zeitl.Verschiebung * c}{Mikrofonabstand}\right) \qquad (8.18)$$

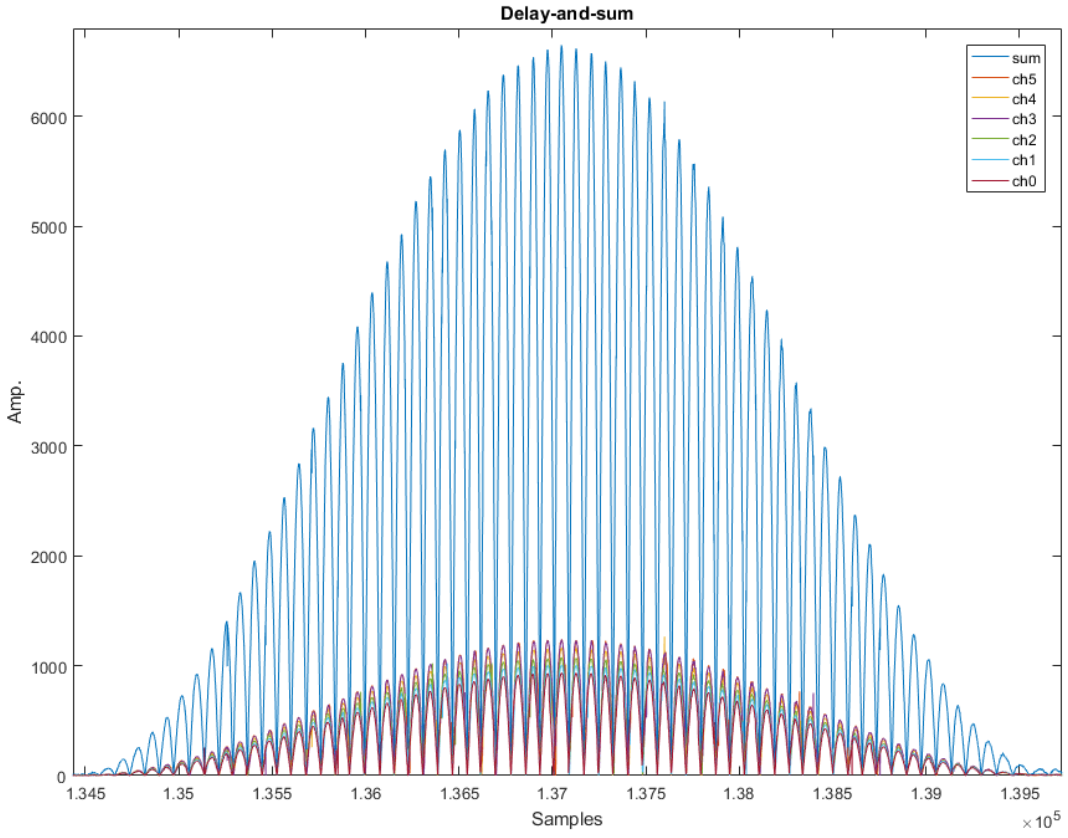


Abbildung 51: Verschobene und aufsummierte Signale

Mit der gezeigten Formel und den ermittelten Verschiebungen wurde bei der Position 1 ein Winkel von β=35.0538° und bei der Position 3 ein Winkel von β=135.7488° errechnet.

Mit diesen Ergebnissen wird der Nachweis erbracht, dass das System zur Lokalisierung von Schallquellen eingesetzt werden kann.

8.2.3 Test Stage 3

Die Tests der dritten Stage beziehen sich auf die Zeitgebungsfunktion des PTP-Protokolls. Dazu wird die Stage 3 als Master des PTP-Protokolls und damit als Zeitgeber definiert.

Es wurden Logfiles im Master während der Ausführung der Zeitsynchronisierung geschrieben. Beispielhaft wird folgender Eintrag im Log File des Masters gezeigt:

(slv) Stepped the system clock to: 04/21/16 12:41:48.907138270

Dieser zeigt die Anpassung der Zeit des Slaves auf die genaue Zeit des Masters. Zuvor wurde die Übertragungszeit in beide Richtungen gemessen um diese bei der Anpassung berücksichtigen zu können. Die Synchronisierung geschieht im Rahmen der Geforderten Genauigkeit von 10µs (1/2 einer Abtastperiode).

8.3 Vergleich der Messungen mit den Anforderungen

Die Messungen sind durchgeführt worden um zu prüfen ob die aufgestellten Anforderungen von dem System erfüllt werden.

Es ist mit den Messungen eines Boards aus der Stage 1 die korrekte Abtastung mit 48Khz gezeigt worden. Die eingegebene Frequenz durch den Kalibrator von 1 kHz findet sich in der Auswertung ohne Frequnzverschiebung wieder. Dabei wurde ebenfalls die prinzipielle Kalibarkeit des Mikrofons gezeigt.

Es ist gezeigt die Synchronität der Signale bei einer Messung mit mehreren Mikrofonkanälen gezeigt worden, dabei wurden 7 Signale mit der gewählten Abtastfrequenz von 48 kHz eingelesen. Die Synchronität lässt sich auch auf eine Messung mit Beschleunigungssensoren ausweiten, die Stage 2 den Takt der Abfragen vorgibt. Alle vom System verwendeten Prozessorboards werden mit der Systemzeit versehen und arbeiten mit der geforderten zeitlichen Genauigkeit untereinander.

Mit der Aufzeichnung des Frequenzgangs ist die Abdeckung des gesamten geforderten Frequenzbereichs von 100 Hz bis 10 kHz gezeigt worden. In diesem Frequenzbereich ist die Deckung mit dem analogen Referenzmikrofon

sehr groß. Es wurde ebenfalls gezeigt, dass Messung auch oberhalb von 10 kHz mit einer korrigierenden Filterung in Stage möglich ist.

Es wurde die Funktion des Beschleunigungssensors gezeigt. Dabei konnte keine Aussage über den Frequenzbereich gemacht werden, da für eine aussagekräftige Messung keine Vorrichtung vorhanden ist.

9 Zusammenfassung

Diese Arbeit zeigt, die Realisierung eines synchronen hochkanaligen Systems zur Messung von Luft- und Körperschall. Es wurde ein modulares, flexibel erweiterbares Messsystem entwickelt. Das gezeigte System fasst dezentral Einheiten zu Modulen zusammen und ermöglicht einen flexiblen Aufbau des Messsystems.

Es konnte nachgewiesen werden, dass die verwendeten MEMS-Sensoren den Anforderungen an ein Messsystem zur Vermessung von Schallemissionen genügen. Dabei ist es mit dem entwickelten System möglich, auftretende Schwingungen der Struktur oder der Mikrofone, durch die Messung der Beschleunigung mit dem integrierten Sensor, zu filtern.

Das entwickelte System zeigt, dass die aufgenommenen Signale synchron in mehreren Einheiten zusammengeführt werden können, um Aussagen über die Richtung und die Stärke des einfallenden Schalls errechnen zu können.

Bei Messungen im reflexionsarmen Raum wurde zur Lokalisierung einer Schallquelle ein lineares Array von Mikrofonen aufgebaut. Mit diesem Aufbau ist die Funktionsfähigkeit des Gesamtsystems zur Schalllokalisierung nachgewiesen worden.

Das entwickelte System enthält ausschließlich frei am Markt verfügbare Komponenten, zusammen mit der designten Platine und kann damit den Gesamtpreis des Systems gegenüber erhältlichen Lösungen deutlich verringern.

10 Ausblick

10.1 Hardwareoptimierung

Das entwickelte System verwendet zur Verbindung zweier Ebenen Ethernet Kabel. Nach einer Prüfung des Verhaltens der Zeitsynchronisierung, wäre die Verwendung von WLAN-Komponenten zur Verbindung der Ebenen denkbar. Eine drahtlose Lösung würde den Aufwand beim Einrichten des Systems am Messort verringern. Drüber hinaus ist es mit einer drahtlosen Lösung möglich die Mikrofone frei zu platzieren. Weiter wäre eine Ausstattung mit einem Akkumulator sinnvoll. Dieser Energiespeicher würde im System die Möglichkeit bieten, flexibel und Infrastrukturunabhängig, Messungen durchzuführen.

Der Entwickelte Aufbau der Sensoren auf einer einzelnen Platine ließe sich in einer anderen Form anordnen. Damit lassen sich alle technisch sinnvollen Anordnungen von Mikrofonen erzeugen. Neben der einfachen Schalldruckmessung ist es möglich mit je einem Paar von Mikrofonen mehrere Schallintensitätssonden in einer festen Struktur zu einander anzuordnen. Ein solcher Aufbau könnte die eintreffende Schallintensität in den drei Raumrichtungen gleichzeitig aufzeichnen.

10.2 Softwareoptimierung

Die implementierte Software auf allen drei Ebenen leitet die eingehenden Sensorsignale weiter und Verknüpft diese zu Datenpaketen. Die eingesetzte Rechenhardware der drei Ebenen ermöglicht eine Implementierung von Filtern und die Berechnung von Signalauswertungen. Denkbar ist die Implementierung eines „Delay-and-Sum-Algorithmuses“ in der Messkette. Es könnte nur der Richtungsvektor auf die Schallquelle übertragen werden. Dies würde die Datenrate und die zu speichernden Daten deutlich reduzieren. Denkbar ist eine Implementierung zur Berechnung einer „Delay-and-Sum-Ortung“ in der Messkette. Diese Berechnung könnte auf dem FPGA parallel ausgeführt werden.

Neben der Verrechnung von Signalen wäre eine Sensoroptimierung mittels einer Kennlinie sinnvoll. Mit einer solchen Optimierung würde eine flache Kennlinie der Sensoren über einen breiten Frequenzbereich möglich. So könnte die Qualität der Signale noch einmal deutlich gesteigert werden. Darüber hinaus könnte mit der Verwendung von Sensoren unterschiedlicher Hersteller, nach außen immer das gleiche Verhalten gezeigt werden.

10.3 Funktionserweiterung

Die Entwicklung bietet über standardisierte Schnittstellen, die Möglichkeit in jeder Ebene weitere Sensoren einzubinden und mit aufzuzeichnen. Möglich wäre die Einbindung eines GPS-Sensors, mit diesem könnten an mehreren Orten Messungen durchgeführt werden und eine Verrechnung wäre mit der Kenntnis der Messorte möglich.

Weiter wären ein Feuchtesensor sowie ein Windsensor für akustische Messungen im Freien sinnvoll. Diese Sensoren würden eine genaue Bestimmung der Schallgeschwindigkeit ermöglichen und mit der Windgeschwindigkeit und -richtung eine Abschätzung der zu erwartenden Störungen gestatten.

Eine Live-Messung ist eine Messung bei der die eintreffenden Sensorsignale in Echtzeit ausgewertet und angezeigt werden. Mit einer solchen Lösung sind Messungen mit einem Handheld-Array durchzuführen. Dabei bewegt sich eine Person im Raum und kann live mit einem Bildschirm nach Schallquellen suchen. Die Realisierung einer Möglichkeit zur Durchführung einer Live-Messung würde den Funktionsumfang der entwickelten Lösung noch einmal erweitern.

Literaturverzeichnis

1. Camera, Acoustic. GFAI Tech. [Online] [Zitat vom: 02. 04 2016.] http://www.acoustic-camera.com/en/applications/wind-turbine.html.
2. Möser, Michael. *Technische Akustik.* Berlin Heidelberg : Springer, 2015.
3. R. Lerch, G. Sessler, D. Wolf. *Technische Akustik.* Heidelberg : Springer, 2009.
4. Gh. R. Sinambari, S. Sentpali. *Ingenieursakustik - Physikalische Grundlagen und Anwendungsbeispiele.* Wiesbaden : Springer, 2014.
5. Mörser, Michael. *Messtechnik der Akustik.* Berlin : Springer, 2010.
6. M. Zollner, E. Zwicker. *Elektroakustik.* Heidelberg New York : Springer, 2003.
7. Microelectronics, ST. Application Note AN4426. [Online]
8. Beer, Daniel. http://dlbeer.co.nz. [Online] 08. 03 2011. [Zitat vom: 12. 03 2016.] http://dlbeer.co.nz/articles/pdm-sine.png.
9. Microeletronics, ST. *MP45DT02- Datasheet.*
10. maximintegrated. [Online] [Zitat vom: 25. 04 2016.] http://pdfserv.maximintegrated.com/en/an/AN5830.pdf.
11. labbookpages. [Online] [Zitat vom: 15. 04 2016.] http://www.labbookpages.co.uk/audio/beamforming/delaySum.html.
12. Systems, CAE. CAE Systems. [Online] [Zitat vom: 17. 04 2016.] http://www.cae-systems.de/produkte/akustische-kamera/analoge-arrays.html.
13. Wenzel, Tobias Morten. *Vier-Kanal-Mikrofon.* Hamburg : s.n., 23.06.2014.
14. Instruments, Texas. *Datasheet PCM5121.*
15. Atmel. Atmel-42310-SAM-G53-Schematic-Checklist_ApplicationNote_AT07215. [Online] [Zitat vom: 15. 10 2015.] http://www.atmel.com/devices/ATSAMG53.aspx?tab=documents.
16. Kjær, Brüel &. *½" Free-field Microphone — Type 4190- Datasheet.*
17. J. Blauert, NXiang. *Acoustics for Engineers.* Berlin-Heidelberg : Springer, 2009.
18. Microeletronics, ST. *AN4426-Tutorial for MEMS microphones.* 2014.
19. Ditran. distran.ch. [Online] [Zitat vom: 2016. 04 25.] http://www.distran.ch/en/products/.
20. Norsonic. Norsonic.com- Nor848A. [Online] [Zitat vom: 2016. 04 25.] http://www.norsonic.com/filestore/PDF-filer/Product_Data/PD848AEd3Rev1Eng0315web.pdf.
21. microflown. microflown.com. [Online] [Zitat vom: 2016. 04 24.] www.microflown.com.
22. CAE-systems. www.cae-systems.de. [Online] [Zitat vom:] datasheet-acoustic-camera-bionic-l-112.pdf.
23. bestech. bestech.com.au. [Online] bestech. [Zitat vom: 25. 04 2016.] bestech.com.au/sensors.

24. acousticgroup. acousticgroup.com -GRUNDLAGEN DER AKUSTIK. [Online] [Zitat vom: 15. 03 2016.] http://www.acousticgroup.com/cms/upload/Allgemein/Grundlagen_der_Akustik_de.pdf.
25. kjaer, bruel. bruelkjaer.de. [Online] [Zitat vom: 15. 03 2016.] http://www.bruelkjaer.de/Products/transducers/acoustic/calibrators/4231.

Abbildungsverzeichnis

Danksagungen

Jede Masterarbeit trägt die Handschrift des Erstellers, und doch ist sie niemals die Arbeit eines Einzelnen.

Meinen Professoren Herrn Prof. Dr. rer. nat. Rasmus Rettig und Herrn Prof. Dr.-Ing. Stephan Hußmann danke ich für die wissenschaftliche Unterstützung und die immer neuen wegweisenden und konstruktiven Vorschläge. Meinem Professor Herrn Prof. Dr. rer. nat. Rasmus Rettig bin ich für seine vielfältigen thematischen Anregungen besonders dankbar. Ohne die umfangreichen Diskussionen mit ihm und dem Team im Labor der HAW hätte ich diese Arbeit nicht in der hier dargebotenen Form entwickeln können. Und schließlich gebührt besonderer Dank meiner Familie, die zu jeder Zeit an mich geglaubt hat und die mich in dieser fordernden und spannenden Phase meines Studiums, begleitet hat.

Anhang

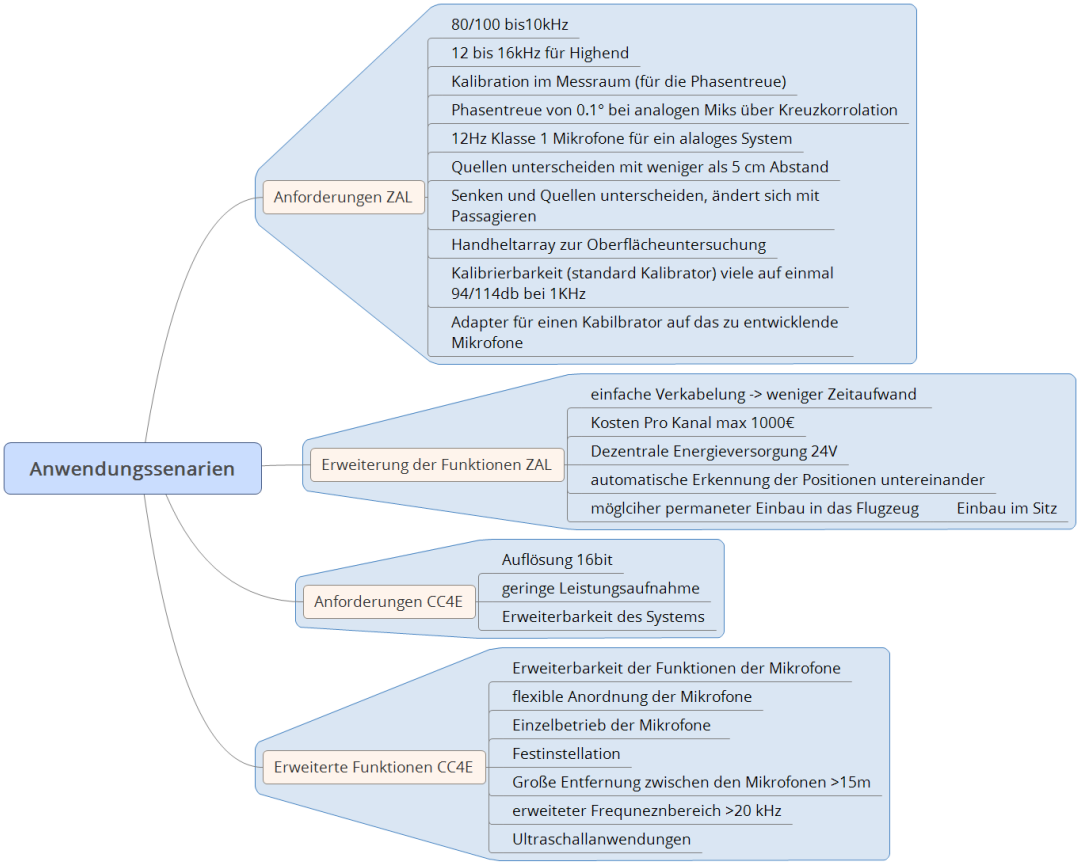
Der Anhang der Arbeit befindet sich auf einer DVD und ist einzusehen bei Prof. Dr. rer. nat. R. Rettig, HAW Hamburg, Berliner Tor 7, 20099 Hamburg.

Anhang A – Anforderungen

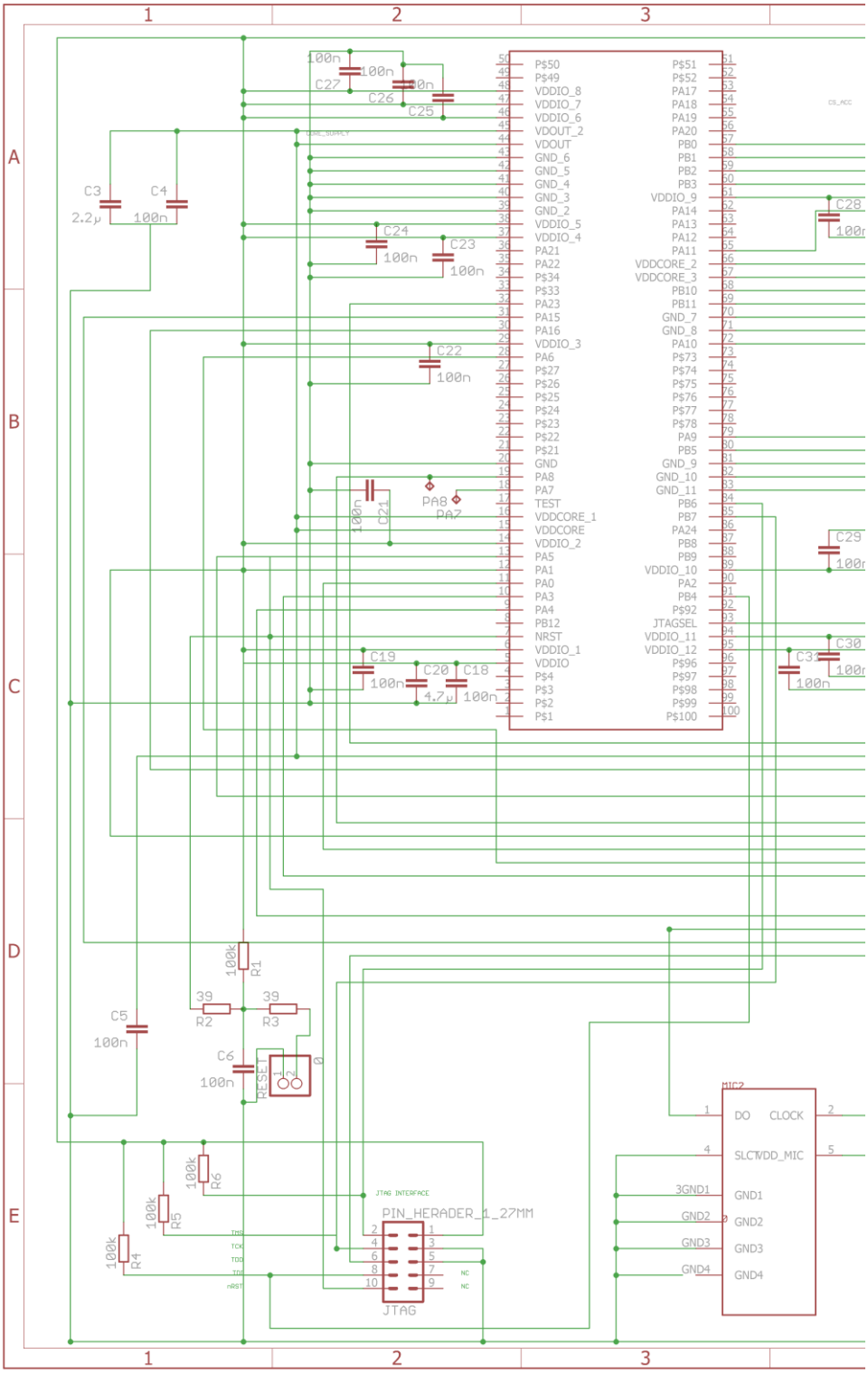
		Anforderungsliste		
		hochkanalige synchrone Messtechnik		
Lfd.	**F/W**	**Anforderungen**	**Werte/Daten**	
1		**Messgrößen**		
1.1	F	Luftschall	Frequenzbereich	100Hz bis 10kHz
1.2	F	Körperschall	Frequenzbereich	<10Hz bis 1Khz
2		**Messanforderungen**		
2.1	F	Kanäle	erweiterbare Anzahl	128/256
2.2	F		Mindestzahl	9
2.3	F	Auflösung	Wortbreite	16bit
2.4	F	Kalibrierbarkeit	Mit Kalibrator	94/114dB @ 1kHz
2.5	F	Abtastrate der Mikrofone	Frequenz	48 kHz
2.6	F	Snychronität der Signale	max. zeitliche Abweichung	10µs
2.7	F	Messachsen Beschl.		3
3		**Ausgabe**		
3.1	F	Datei mit den Daten aller Mikrofone	universal lesbares Format	
3.2	W2	Analoger Output an jedem Mikrofon		
4		**Auswertung**		
4.1	F	Ausgabe	Zeitsignal aller Mikrofone	
4.2	W2	Spektralanalyse		
5		**Geometrie**		
5.1	F	Abstand zwischen zwei Mikrofonen	möglichst gering	
5.2	W2	Aufbau	starrer Aufbau	
6		**Umwelt**		
6.1	W1	Temperatur		minus 10°C bis 50°C
6.2	W1	Feuchtigkeit	Schutzart für Außeneinsatz	IP20

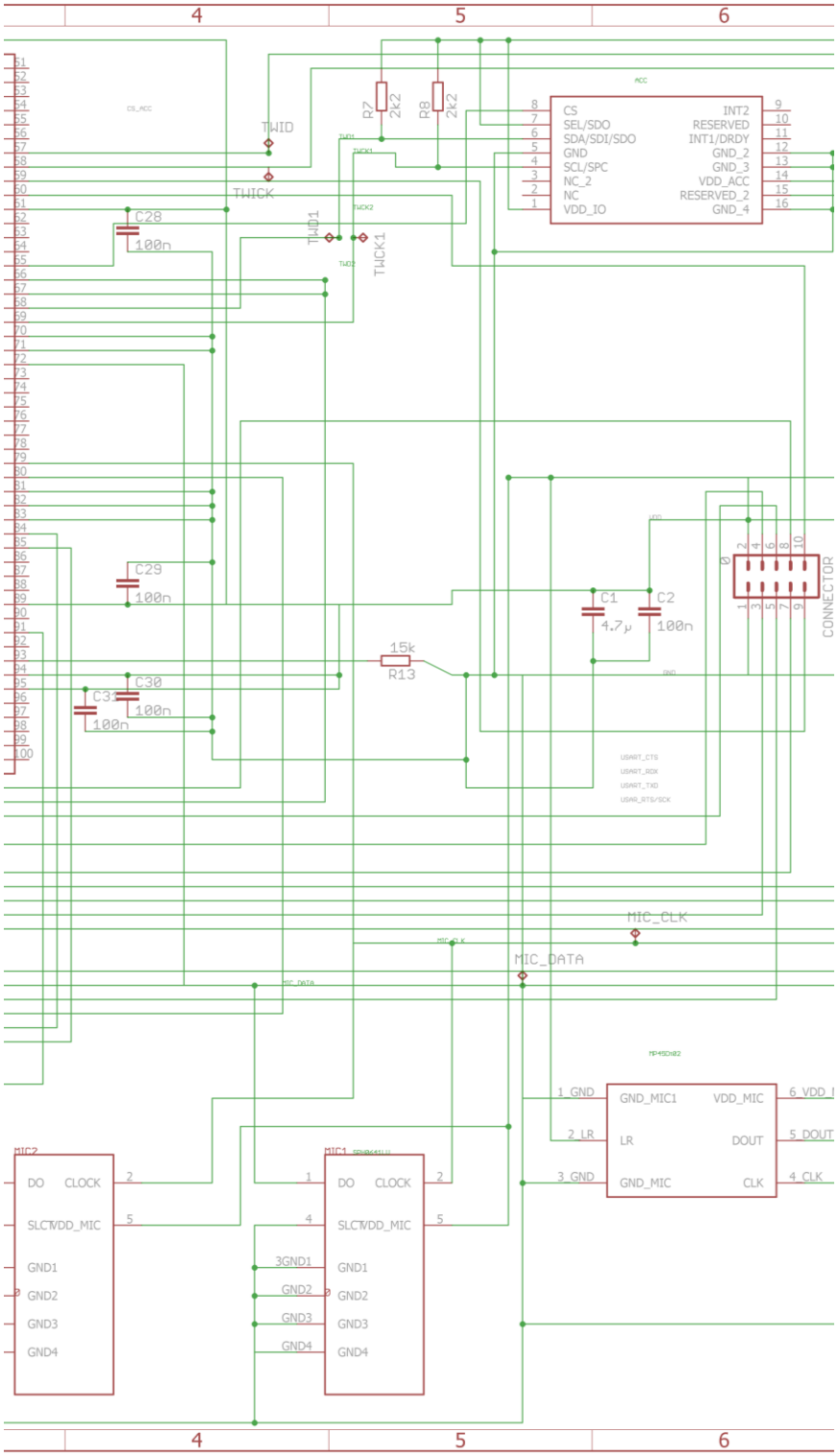
7		**Kosten**		
7.1	F	Kosten pro Kanal		100 €
7.2	F	Gesamtsystem	ohne Software	80.000 €
8		**Energie**		
8.1	F	Energieverbrauch Gesamtsystem		<380 W
9		**Transport und Aufbau**		
9.1	W2	Einfacher Auf- und Abbau		
9.2		Handheldversion		
	F=Forderung, W4=sehr wichtig, W3= wichtig, W2 interessant, W1=wenn möglich			

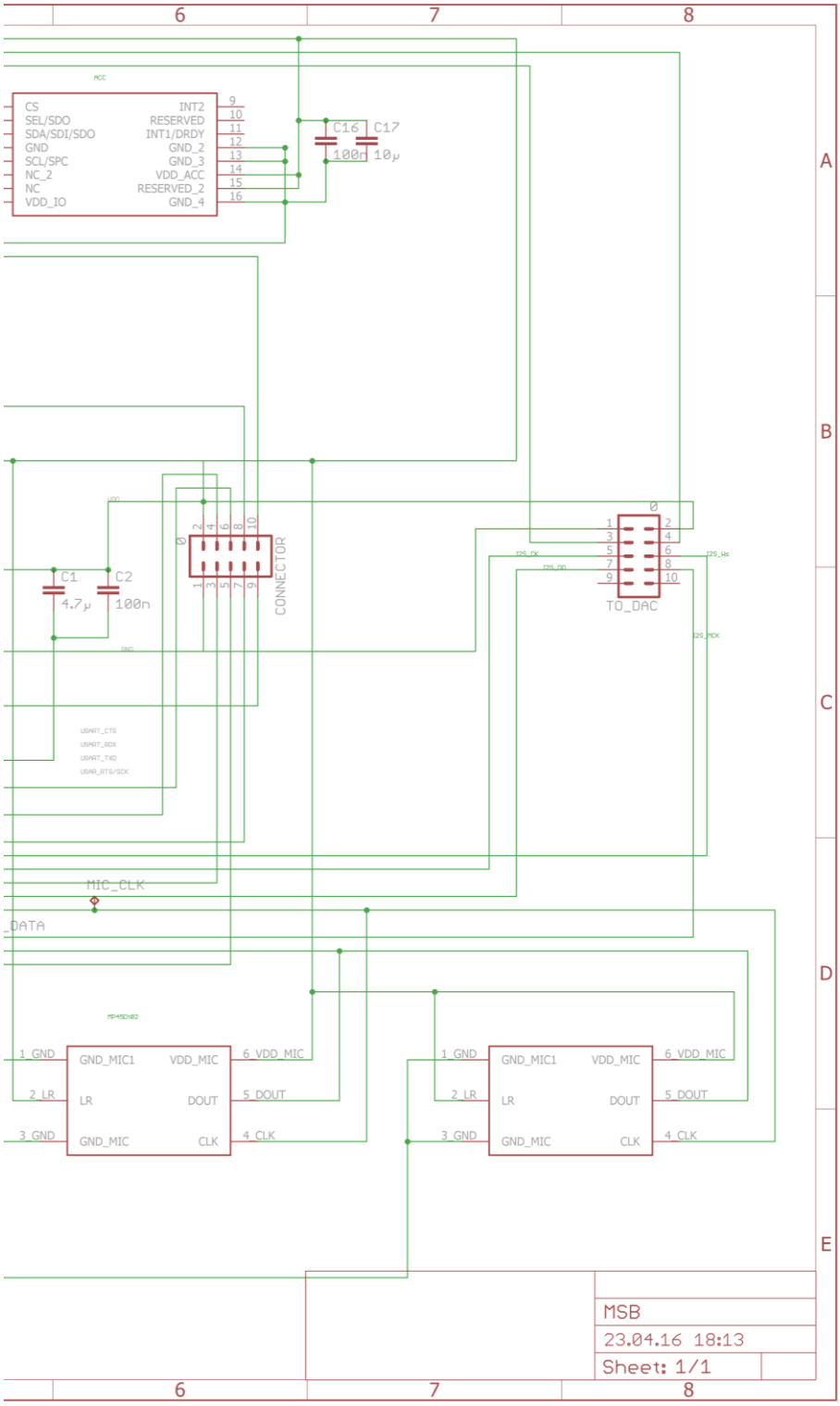
Anhang B – Anwendungen



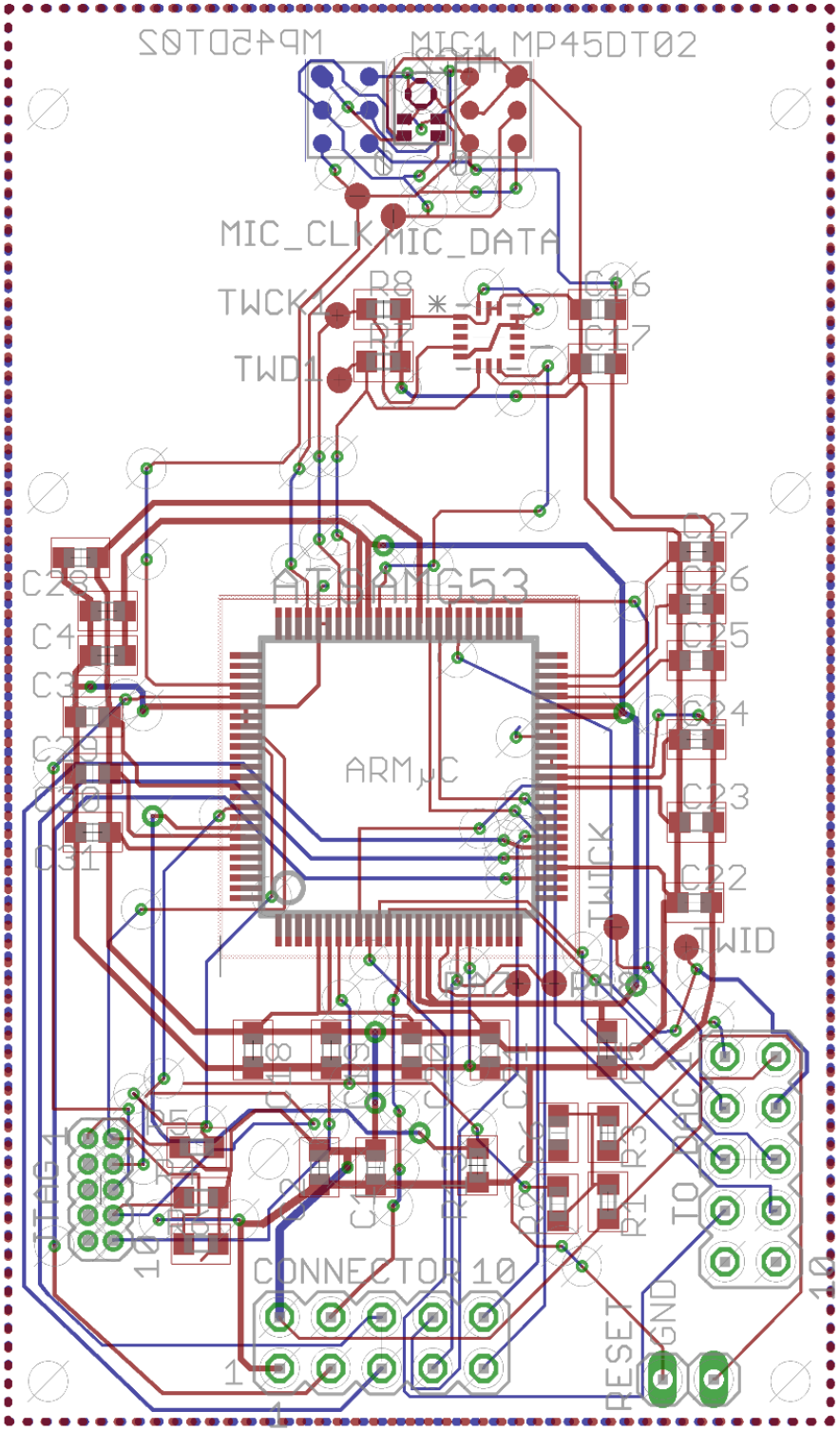
Anhang C – Schaltplan




4
5
6
ACC
CS
SEL/SDO
SDA/SDI/SDO
GND
SCL/SPC
NC_2
NC
VDD_IO
INT2
RESERVED
INT1/DRDY
GND_2
GND_3
VDD_ACC
RESERVED_2
GND_4
R7
2k2
R8
2k2
TWID
TWICK
TWID1
TWICK1
C28
100n
C29
100n
C30
100n
C31
100n
15k
R13
C1
4,7µ
C2
100n
CONNECTOR
MIC_CLK
MIC_DATA
MP45DH02
1_GND
2_LR
3_GND
GND_MIC1
LR
GND_MIC
VDD_MIC
DOUT
CLK
6_VDD_I
5_DOUT
4_CLK
MIC2
MIC1
DO
CLOCK
SLCT
VDD_MIC
GND1
GND2
GND3
GND4
3GND1
4
5


6
7
8
A
B
C
D
E
ACC
CS
SEL/SDO
SDA/SDI/SDO
GND
SCL/SPC
NC_2
NC
VDD_IO
INT2
RESERVED
INT1/DRDY
GND_2
GND_3
VDD_ACC
RESERVED_2
GND_4
C16
100n
C17
10µ
C1
4.7µ
C2
100n
CONNECTOR
TO_DAC
MIC_CLK
_DATA
1_GND
2_LR
3_GND
GND_MIC1
LR
GND_MIC
VDD_MIC
DOUT
CLK
6_VDD_MIC
5_DOUT
4_CLK
MSB
23.04.16 18:13
Sheet: 1/1

Anhang D – Boarddesign



Anhang E – Programm Stage1

Der Quellecode zur Programmierung des Prozessors in Stage 1 ist auf folgenden Seiten angehängt

```
/**
*       Multi-Sensor-Mikrofon (MSM)
*/
#include <asf.h>
#include "stdio_serial.h"
#include "conf_board.h"
#include "conf_clock.h"
#include "conf_uart_serial.h"
#include "lis3dsh2.h"

//#define DAC_CONNECTED
#define USART_INTERRUPT_PRIORITY    1
#define PDM_INTERRUPT_PRIORITY      2
#define TIMER_INTERRUPT_PRIORITY    3
#define TWI1_INTERRUPT_PRIORITY     4

/**************************************************************************
**/
/*                          Definitions
*/
/**************************************************************************
**/
#define MCLK                48000000
#define TWI1_BUS_SPEED      400000
#define TWI2_BUS_SPEED      400000
#define ACC_BUS_ADDR        0x1D        // LIS3DSH Bus Address
#define DAC_BUS_ADDR        0x4C        // PCM5121 Bus Address
#define ACC_SENSITIVITY     0.06
#define TWI1_BUFFER_SIZE    6
#define USART_BUFFER_SIZE   2
#define NB_BUFFERS          16

/**************************************************************************
**/
/*                      Function Prototypes
*/
/**************************************************************************
**/
static void init_timer(void);
static void init_pdm(void);
static void init_usart(void);
static void init_twi1(void);
static void init_acc(void);
static void acc_process_data(void);
static void acc_init_data_read(void);
#ifdef DAC_CONNECTED
static void init_i2s(void);
static void init_twi2(void);
static void init_dac(void);
static void write_dac_register(uint8_t page, uint8_t reg, uint8_t
data);
static uint8_t read_dac_register(uint8_t page, uint8_t reg);
#endif

/**************************************************************************
**/
/*                          Declarations
*/
/**************************************************************************
**/
```

```
61  /* PDC transfer buffer (DMA) */
62  volatile int8_t twi1_pdc_buffer[TWI1_BUFFER_SIZE];
63  volatile int8_t usart_buffer[NB_BUFFERS][USART_BUFFER_SIZE];
64  volatile uint8_t current_usart_buffer = 0;
65  volatile uint8_t usart_buffer_rdy_idx = 0;
66  volatile int8_t test_buffer[USART_BUFFER_SIZE] = {0xA9,0x63};
67  static volatile int16_t pdm0_data = 0;
68
69  const size_t spi_data_packet_lenght = USART_BUFFER_SIZE;
70  uint8_t spi_data_packet[USART_BUFFER_SIZE] = {0};
71  int16_t acc_data[3] = {0};
72
73  static volatile bool pdm0_data_rdy = false, pdm1_data_rdy = false;
74  static volatile bool timer_trigger = false;
75
76  /* PDC data packet for transfer */
77  pdc_packet_t twi1_pdc_packet;
78  pdc_packet_t usart_pdc_packet;
79
80  /* Pointer to Peripheral DMA Controller register base */
81  Pdc *p_twi1_pdc;
82  Pdc *p_usart_pdc;
83
84  struct pdm_instance pdm0;
85  twi_master_t TWIM1 = TWI1;
86
87  #ifdef DAC_CONNECTED
88  twi_master_t TWIM2 = TWI2;
89
90  /* I2S configuration structure */
91  struct i2s_config config_i2s;
92  /* I2S device structure */
93  struct i2s_dev_inst dev_inst_i2s0;
94
95  int16_t i2s_left_ch = 0x00;
96  int16_t i2s_right_ch = 0x00;
97  volatile int32_t i2s_data = 0x0000;
98  #endif
99  /************************************************************************
100 **/
101 /*                                main()
102 */
103 /************************************************************************
104 **/
105
106 int main (void){
107
108     /* Insert system clock initialization code here (sysclk_init()).
109 */
110     sysclk_init();
111     /* Initialize the SAM system */
112     system_board_init();
113     /* Insert application code here, after the board has been ini-
114 tialized. */
115     ioport_init();
```

```
116         delay_init(sysclk_get_cpu_hz());
117         init_pdm();
118         init_usart();
119         init_twi1();
120         init_acc();
121         init_timer();
122         #ifdef DAC_CONNECTED
123         init_i2s();
124         init_twi2();
125         init_dac();
126         i2s_enable(&dev_inst_i2s0);
127         #endif
128         /* Enable PDM module */
129         pdm_enable(&pdm0);
130         rtt_enable(RTT);
131
132         Enable_global_interrupt();
133
134         /* Insert application code here, after the board has been ini-
135     tialized. */
136         while(1) {
137             // Timer triggers data read sequence from Accelerometer
138             if(timer_trigger){
139                 timer_trigger = false;
140                 twi_enable_interrupt(TWI1, TWI_IER_RXBUFF);
141                 acc_init_data_read();
142             }
143         }
144     }
145     /*************************************************************
146     **/
147     /*                          Interrupt Handler
148     */
149     /*************************************************************
150     **/
151
152     /* data transfer on TWI1 completed */
153     void TWI1_Handler(){
154         /* Get TWI status and check if PDC receive buffer is full */
155         if((TWIM1->TWI_SR & TWI_SR_RXBUFF) == TWI_SR_RXBUFF) {
156             twi1_pdc_packet.ul_addr = (uint32_t) twi1_pdc_buffer;
157             twi1_pdc_packet.ul_size = TWI1_BUFFER_SIZE;
158             /* Configure PDC for next data transfer (RX) */
159             pdc_rx_init(p_twi1_pdc, &twi1_pdc_packet, NULL);
160             twi_disable_interrupt(TWI1, TWI_IER_RXBUFF);
161             acc_process_data();
162         }
163     }
164     /* @ 1,6kHz */
165     void RTT_Handler(){
166         NVIC_DisableIRQ(RTT_IRQn);
167         uint32_t status = RTT->RTT_SR;
168         if((status & RTT_SR_ALMS) == RTT_SR_ALMS){
169             RTT->RTT_MR |= RTT_MR_RTTRST;
170             timer_trigger = true;
```

```
171         }
172         NVIC_EnableIRQ(RTT_IRQn);
173     }
174
175     /* new data on PDMIC0 available (1.42us) */
176     void PDMIC0_Handler(void){
177         /* Read PDM Converted Data Register*/
178         pdm0_data = pdm0.hw->PDMIC_CDR;
179         usart_buffer[current_usart_buffer][0] = (uint8_t)(pdm0_data);
180         usart_buffer[current_usart_buffer][1] = (uint8_t)(pdm0_data >>
181     8);
182         /* store the index with the latest data */
183         usart_buffer_rdy_idx = current_usart_buffer;
184         pdm0_data_rdy = true;
185         /* next buffer index */
186         current_usart_buffer = (current_usart_buffer + 1) % NB_BUFFERS;
187     }
188
189     /* on every ChipSelect Input change (execution time: 0.38/1.86us)*/
190     void USART_Handler(void){
191         uint32_t status = USART_SERIAL->US_CSR; // clear interrupt
192         /* Check if CS is HIGH */
193         if( (status & US_CSR_CTS) == US_CSR_CTS ){
194             usart_pdc_packet.ul_addr = (uint32_t)
195     usart_buffer[usart_buffer_rdy_idx];
196             usart_pdc_packet.ul_size = USART_BUFFER_SIZE;
197             pdc_tx_init(p_usart_pdc, &usart_pdc_packet, NULL);
198         }
199     }
200
201
202
203     /*----------------------------------------------------------------------
204     **/
205     /*                        Function Implementations
206     */
207     /*----------------------------------------------------------------------
208     **/
209     /* Function to setup the PDM module */
210     static void init_pdm(void)
211     {
212         /* PDM Interface Controller 0 - */
213         struct pdm_config conf_pdmic0;
214         pdm_get_config_default(&conf_pdmic0);
215         conf_pdmic0.prescal = PDM_PRESCALER;
216         conf_pdmic0.gain = 1;
217         conf_pdmic0.oversampling_ratio = PDMIC_OVERSAMPLING_RATIO_64;
218         conf_pdmic0.conver_data_size = PDMIC_CONVERTED_DATA_SIZE_16;
219         pdm_init(&pdm0, PDMIC0, &conf_pdmic0);
220
221         /* PDM Interrupts */
222         NVIC_DisableIRQ(PDMIC0_IRQn);
223         NVIC_ClearPendingIRQ(PDMIC0_IRQn);
224         NVIC_SetPriority(PDMIC0_IRQn, PDM_INTERRUPT_PRIORITY);
225         NVIC_EnableIRQ(PDMIC0_IRQn);
```

```
226
227         pdm_enable_interrupt(&pdm0, PDMIC_INTERRUPT_SRC_DATA_READY);
228     }
229     /* Function to setup the USART module */
230     static void init_usart(void){
231         const usart_spi_opt_t usart_spi_settings = {
232             USART_SPI_BAUDRATE,
233             USART_SPI_CHAR_LENGTH,
234             USART_SPI_MODE,
235             USART_SPI_CHANNEL_MODE
236         };
237         sysclk_enable_peripheral_clock(USART_SERIAL_ID);
238         usart_init_spi_slave(USART_SERIAL, &usart_spi_settings);
239         usart_enable_tx(USART_SERIAL);
240         usart_enable_rx(USART_SERIAL);
241
242
243         /* Configure PDC Module for DMA Transfers */
244         p_usart_pdc = (Pdc *)((uint32_t)USART_SERIAL + 0x100);
245         /* Initialize PDC data packet for transfer */
246         usart_pdc_packet.ul_addr = (uint32_t)  test_buffer;
247         usart_pdc_packet.ul_size = USART_BUFFER_SIZE;
248         /* Configure PDC for data receive */
249         pdc_tx_init(p_usart_pdc, &usart_pdc_packet, NULL);
250         /* Enable PDC transfers */
251         pdc_enable_transfer(p_usart_pdc, PERIPH_PTCR_TXTEN);
252
253         NVIC_EnableIRQ(USART_IRQn);
254         NVIC_ClearPendingIRQ(USART_IRQn);
255         NVIC_SetPriority(USART_IRQn, USART_INTERRUPT_PRIORITY);
256         NVIC_EnableIRQ(USART_IRQn);
257
258         usart_enable_interrupt(USART_SERIAL, US_IER_CTSIC);
259     }
260
261     /* Function to setup the TWI1 module */
262     static void init_twi1(void){
263         /* TWI1 config */
264         twi_master_options_t opt = {
265             .speed      = TWI1_BUS_SPEED,
266             .chip       = ACC_BUS_ADDR
267         };
268         /* enable peripheral clock */
269         twi_master_setup(TWIM1, &opt);
270         /* Clear receipt buffer */
271         twi_read_byte(TWI1);
272
273         /* Configure PDC Module for DMA Transfers */
274         p_twi1_pdc = (Pdc *)((uint32_t)TWI1 + 0x100);
275         /* Initialize PDC data packet for transfer */
276         twi1_pdc_packet.ul_addr = (uint32_t) twi1_pdc_buffer;
277         twi1_pdc_packet.ul_size = TWI1_BUFFER_SIZE;
278         /* Configure PDC for data receive */
279         pdc_rx_init(p_twi1_pdc, &twi1_pdc_packet, NULL);
280         /* Enable PDC transfers */
```

```
281         pdc_enable_transfer(p_twi1_pdc, PERIPH_PTCR_RXTEN);
282         /* Enable TWI IRQ */
283         twi_enable_interrupt(TWI1, TWI_SR_RXBUFF);
284         /* Enable TWI interrupt */
285         NVIC_DisableIRQ(TWI1_IRQn);
286         NVIC_ClearPendingIRQ(TWI1_IRQn);
287         NVIC_SetPriority(TWI1_IRQn, TWI1_INTERRUPT_PRIORITY);
288         NVIC_EnableIRQ(TWI1_IRQn);
289     }
290
291     /* Function to setup the LIS3DSH Accelerometer */
292     static void init_acc(void){
293
294         // CS HIGH (LIS3DSH I2C Mode)
295         ioport_set_pin_level(ACC_CS_PIN, IOPORT_PIN_LEVEL_HIGH);
296
297         uint16_t InitStruct = 0x97;     // all axis enable, 1.6kHz ODR
298         LIS3DSH_Init(InitStruct);
299
300         /* Connection Test */
301         if(LIS3DSH_ReadID() != I_AM_LIS3DSH){
302
303         }
304
305         uint8_t ctrl_reg6 = 0x10;   // Register Auto Increment Enable
306         ACC_IO_Write(&ctrl_reg6,LIS3DSH_CTRL_REG6_ADDR,1);
307     }
308
309     /* RT Timer setup for Accelerometer Polling Timing*/
310     static void init_timer(void){
311         /* 32kHz / 3 / 1 = 10,66kHz (10,02kHz gemessen)
312         *
313         *       30060 / 1600 = 18,787 => 19     Messung: 1582 Hz
314         *                                18     Messung: 1666 Hz
315         */
316         uint16_t prescaler = 19;
317         uint32_t alarm_time = 1;
318
319         //rtt_sel_source(RTT, false);
320         rtt_init(RTT, prescaler);
321         rtt_write_alarm_time(RTT, alarm_time);
322         rtt_enable_interrupt(RTT, RTT_MR_ALMIEN);
323
324         /* Enable RTT interrupt */
325         NVIC_DisableIRQ(RTT_IRQn);
326         NVIC_ClearPendingIRQ(RTT_IRQn);
327         NVIC_SetPriority(RTT_IRQn, TIMER_INTERRUPT_PRIORITY);
328         NVIC_EnableIRQ(RTT_IRQn);
329     }
330
331     static void acc_init_data_read(void){
332
333         uint32_t status = TWIM1->TWI_SR;
334
```

```
335         /* Set read mode, slave address and 3 internal address byte
336     lengths */
337         TWIM1->TWI_MMR = TWI_MMR_MREAD | TWI_MMR_DADR(ACC_BUS_ADDR) |
338         ((sizeof(uint8_t) << TWI_MMR_IADRSZ_Pos) & TWI_MMR_IADRSZ_Msk);
339
340         /* Set internal address for remote chip (with auto address in-
341     crement)*/
342         TWIM1->TWI_IADR = 0x80 | LIS3DSH_OUT_X_L_ADDR;
343
344         TWIM1->TWI_CR = TWI_CR_START;
345     }
346
347     static void acc_process_data(void){
348         float valueinfloat = 0.0;
349         valueinfloat = ((twi1_pdc_buffer[1] << 8) + twi1_pdc_buffer[0])
350     * ACC_SENSITIVITY;
351         acc_data[0] = (int16_t)valueinfloat;
352         valueinfloat = ((twi1_pdc_buffer[3] << 8) + twi1_pdc_buffer[2])
353     * ACC_SENSITIVITY;
354         acc_data[1] = (int16_t)valueinfloat;
355         valueinfloat = ((twi1_pdc_buffer[5] << 8) + twi1_pdc_buffer[4])
356     * ACC_SENSITIVITY;
357         acc_data[2] = (int16_t)valueinfloat;
358     }
359
360     #ifdef DAC_CONNECTED
361     static void init_twi2(void){
362         /* TWI2 config */
363         twi_master_options_t opt = {
364             .speed      = TWI2_BUS_SPEED,
365             .chip       = DAC_BUS_ADDR
366         };
367         /* enable peripheral clock */
368         twi_master_setup(TWIM2, &opt);
369         /* Clear receipt buffer */
370         twi_read_byte(TWI2);
371     }
372
373     static void init_i2s(void){
374         i2s_get_config_defaults(&config_i2s);
375
376         config_i2s.data_format = I2S_DATE_16BIT_COMPACT;
377         config_i2s.fs_ratio = I2S_FS_RATE_256;
378         config_i2s.master_clock_enable = true;
379         config_i2s.master_mode = true;
380         config_i2s.loopback = false;
381         config_i2s.transmit_mode_underrun = true;   // previous sample
382     if underrun
383         config_i2s.rx_channels = I2S_CHANNEL_STEREO;
384
385         i2s_init(&dev_inst_i2s0, I2SC0 , &config_i2s);
386         i2s_enable_transmission(&dev_inst_i2s0);
387         i2s_enable_clocks(&dev_inst_i2s0);
388
389
```

```
390
391 }
392
393 static void write_dac_register(uint8_t page, uint8_t reg, uint8_t
394 data){
395     uint8_t data_pattern[] = {data};
396     uint16_t address = page;
397     address = (address << 8) | reg;
398     twi_package_t packet_write = {
399         .addr         = address,                // TWI slave memory
400 address data
401         .addr_length  = sizeof (uint16_t),      // TWI slave memory
402 address data size
403         .chip         = DAC_BUS_ADDR,           // TWI slave bus ad-
404 dress
405         .buffer       = (void *)data_pattern,   // transfer data
406 source buffer
407         .length       = sizeof(data_pattern)    // transfer data
408 size (bytes)
409     };
410     while (twi_master_write(TWIM2, &packet_write) != TWI_SUCCESS);
411 }
412
413 static uint8_t read_dac_register(uint8_t page, uint8_t reg){
414     uint8_t data_received;
415     uint16_t addr = (page << 8) & reg;
416     twi_package_t packet_read = {
417         .addr         = addr,               // TWI slave memory ad-
418 dress data
419         .addr_length  = sizeof (uint16_t),  // TWI slave memory ad-
420 dress data size
421         .chip         = DAC_BUS_ADDR,       // TWI slave bus address
422         .buffer       = &data_received,      // transfer data desti-
423 nation buffer
424         .length       = 1                   // transfer data size
425 (bytes)
426     };
427     if(twi_master_read(TWIM2, &packet_read) == TWI_SUCCESS){
428         }
429     return data_received;
430 }
431
432 static void init_dac(void){
433     // Datasheet p. 75
434     write_dac_register(0x00, 0x28, 0x00);           // I2S Word
435 Lenght: 16 bit
436     write_dac_register(0x00, 0x2B, 0x00);           // DSP Program
437 Selection
438     write_dac_register(0x00, 0x3B, 0x00);           // Auto Mute
439 Time: 21 ms (default)
440     write_dac_register(0x00, 0x3C, 0x00);           // Vol for L/R
441 are independent
442     write_dac_register(0x00, 0x3D, 0x30);           // Left Digital
443 Volume: 0.0dB (default)
```

```
444         write_dac_register(0x00, 0x3E, 0x30);            // Right Digital
445     Volume: 0.0dB (default)
446         // Output for testing
447         write_dac_register(0x00, 0x08, 0x04);        // GPIO3 Output ena-
448     ble
449         write_dac_register(0x00, 0x52, 0x02);        // GPIO3 Register
450     output (0x56, bit2)
451         write_dac_register(0x00, 0x56, 0x04);        // GPIO1 Output high
452
453     }
454     #endif
```

Anhang F – FPGA Beschreibung

```
1  ----------------------------------------------------------------------------
2  --------------
3  -- Company: HAW Hamburg-FHW
4  -- Engineer: Tobias Wenzel
5  --
6  -- Create Date:    10:36:57 19/04/2016
7
8  -- Module Name:    Stage2 - Behavioral
9  -- Project Name:    Stage2
10 --
11 -- Revision:
12 -- Revision 1.00
13
14 ----------------------------------------------------------------------------
15 --------------
16
17 library IEEE;
18 use IEEE.STD_LOGIC_1164.ALL;
19
20 library work ;
21 use work.logi_wishbone_pack.all ;
22 use work.logi_wishbone_peripherals_pack.all;
23
24 library UNISIM;
25 use UNISIM.VComponents.all;
26 use IEEE.NUMERIC_STD.ALL;
27
28
29
30 entity Stage2 is
31
32     Port ( OSC_FPGA : in std_logic;
33
34          LED : out std_logic_vector(1 downto 0);
35
36          SPIM1_SCLK: inout STD_LOGIC;
37          SPIM2_SCLK: inout STD_LOGIC;
38          SPIM3_SCLK: inout STD_LOGIC;
39          SPIM4_SCLK: inout STD_LOGIC;
40          SPIM5_SCLK: inout STD_LOGIC;
41          SPIM6_SCLK: inout STD_LOGIC;
42
43          SPIM1_MISO: in STD_LOGIC;
44          SPIM2_MISO: in STD_LOGIC;
45          SPIM3_MISO: in STD_LOGIC;
46          SPIM4_MISO: in STD_LOGIC;
47          SPIM5_MISO: in STD_LOGIC;
48          SPIM6_MISO: in STD_LOGIC;
49
50          SPIM1_MOSI:  out STD_LOGIC;
51          SPIM2_MOSI:  out STD_LOGIC;
52          SPIM3_MOSI:  out STD_LOGIC;
53          SPIM4_MOSI:  out STD_LOGIC;
54          SPIM5_MOSI:  out STD_LOGIC;
55          SPIM6_MOSI:  out STD_LOGIC;
56
57
58          SPIM1_CS: inout std_logic_vector(1 downto 0);
59          SPIM2_CS: inout std_logic_vector(1 downto 0);
60          SPIM3_CS: inout std_logic_vector(1 downto 0);
```

```
61          SPIM4_CS: inout std_logic_vector(1 downto 0);
62          SPIM5_CS: inout std_logic_vector(1 downto 0);
63          SPIM6_CS: inout std_logic_vector(1 downto 0);
64
65          pmod2: out std_logic;
66
67          SYS_SPI_SCK, RP_SPI_CE0N, SYS_SPI_MOSI : in std_logic ;
68          SYS_SPI_MISO : out std_logic
69       );
70  end Stage2;
71
72  architecture Behavioral of Stage2 is
73
74
75      component clock_gen
76          port
77          (   CLK_IN1         : in     std_logic; -- Clock in ports
78              CLK_OUT1        : out    std_logic; -- Clock out ports
79              LOCKED          : out    std_logic  -- Status and con-
80  trol signals
81          );
82      end component;
83
84      component edge_detector is
85            Port
86            ( clk             : in  STD_LOGIC;
87              signal_in       : in  STD_LOGIC;
88              output          : out  STD_LOGIC);
89      end component;
90
91  component spi_slave is
92      Generic (
93          N : positive := 16;
94  -- 32bit serial word length is default
95          CPOL : std_logic:= '0';
96  -- SPI mode selection (mode 0 default)
97          CPHA : std_logic := '0';
98  -- CPOL = clock polarity, CPHA = clock phase.
99          PREFETCH : positive :=3);
100 -- prefetch lookahead cycles
101     Port (
102         clk_i : in std_logic ;                                    --
103 internal interface clock (clocks di/do registers)
104         spi_ssel_i : in std_logic ;                               --
105 spi bus slave select line
106         spi_sck_i : in std_logic;                                --
107 spi bus sck clock (clocks the shift register core)
108         spi_mosi_i : in std_logic ;                               --
109 spi bus mosi input
110         spi_miso_o : out std_logic;                              --
111 spi bus spi_miso_o output
112         di_req_o : out std_logic;
113 -- preload lookahead data request line
114         di_i : in  std_logic_vector (N-1 downto 0);  -- parallel
115 load data in (clocked in on rising edge of clk_i)
```

```
116         wren_i : in std_logic;                                          --
117 user data write enable
118         wr_ack_o : out std_logic;
119 -- write acknowledge
120         do_valid_o : out std_logic;
121 -- do_o data valid strobe, valid during one clk_i rising edge.
122         do_o : out  std_logic_vector (N-1 downto 0)
123 -- parallel output (clocked out on falling clk_i)
124     );
125 end component;
126
127
128     component spi_master IS  GENERIC(
129               slaves  : INTEGER := 4;  --number of spi slaves
130               d_width : INTEGER := 8); --data bus width
131           PORT(
132               clock   : IN     STD_LOGIC;
133 --system clock
134               reset_n : IN     STD_LOGIC;
135 --asynchronous reset
136               enable  : IN     STD_LOGIC;
137 --initiate transaction
138               cpol    : IN     STD_LOGIC;
139 --spi clock polarity
140               cpha    : IN     STD_LOGIC;
141 --spi clock phase
142               cont    : IN     STD_LOGIC;
143 --continuous mode command
144               clk_div : IN     INTEGER;
145 --system clock cycles per 1/2 period of sclk
146               addr    : IN     INTEGER;
147 --address of slave
148               tx_data : IN     STD_LOGIC_VECTOR(d_width-1 DOWNTO 0);
149 --data to transmit
150               miso    : IN     STD_LOGIC;
151 --master in, slave out
152               sclk    : INOUT STD_LOGIC;
153 --spi clock
154               ss_n    : INOUT STD_LOGIC_VECTOR(slaves-1 DOWNTO 0);
155 --slave select
156               mosi    : OUT    STD_LOGIC;
157 --master out, slave in
158               busy    : OUT    STD_LOGIC;
159 --busy / data ready signal
160               rx_data : OUT    STD_LOGIC_VECTOR(d_width-1 DOWNTO 0));
161 --data received
162     END component;
163
164     signal sys_reset, sys_resetn,sys_clk, clock_locked : std_logic ;
165     signal clk_100Mhz : std_logic ;
166
167     signal       data_out          :  STD_LOGIC_VECTOR(31 downto 0);
168 -- word read from SDRAM
169     signal       data_out_ready    :  STD_LOGIC;
170 -- is new data ready?
```

```
171
172     signal      enable_spi_reciever :STD_LOGIC := '0';
173     signal      SPIM_all_reset :STD_LOGIC := '0';
174     signal      SPIM_all_enable :STD_LOGIC := '0';
175     signal      SPIM_all_addr   :integer := 0;
176     signal      SPIM_all_cpha   :STD_LOGIC := '0';
177     signal      SPIM_all_cpol   :STD_LOGIC := '0';
178     signal      SPIM_all_clk_div : INTEGER := 0;
179     signal      SPIM_all_cont : STD_LOGIC := '0';
180     signal      SPIM_all_tx_data : STD_LOGIC_VECTOR(7 downto 0);
181
182     signal      write_done_slv1 : STD_LOGIC := '0';
183     signal      write_done_slv2 : STD_LOGIC := '0';
184     signal      write_done_slv3 : STD_LOGIC := '0';
185     signal      write_done_slv4 : STD_LOGIC := '0';
186     signal      write_done_slv5 : STD_LOGIC := '0';
187     signal      write_done_slv6 : STD_LOGIC := '0';
188     signal      write_done_slv7 : STD_LOGIC := '0';
189     signal      write_done_slv8 : STD_LOGIC := '0';
190     signal      write_done_slv9 : STD_LOGIC := '0';
191     signal      write_done_slv10 : STD_LOGIC := '0';
192     signal      write_done_slv11 : STD_LOGIC := '0';
193     signal      write_done_slv12 : STD_LOGIC := '0';
194
195     signal      SPIM1_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
196 "0000000000000000";
197     signal      SPIM2_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
198 "0000000000000000";
199     signal      SPIM3_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
200 "0000000000000000";
201     signal      SPIM4_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
202 "0000000000000000";
203     signal      SPIM5_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
204 "0000000000000000";
205     signal      SPIM6_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
206 "0000000000000000";
207     signal      SPIM7_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
208 "0000000000000000";
209     signal      SPIM8_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
210 "0000000000000000";
211     signal      SPIM9_data_in       :STD_LOGIC_VECTOR(15 downto 0):=
212 "0000000000000000";
213     signal      SPIM10_data_in      :STD_LOGIC_VECTOR(15 downto 0):=
214 "0000000000000000";
215     signal      SPIM11_data_in      :STD_LOGIC_VECTOR(15 downto 0):=
216 "0000000000000000";
217     signal      SPIM12_data_in      :STD_LOGIC_VECTOR(15 downto 0):=
218 "0000000000000000";
219
220     signal      SPIM1_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
221 "0000000000000000";
222     signal      SPIM2_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
223 "0000000000000000";
224     signal      SPIM3_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
225 "0000000000000000";
```

```
226     signal      SPIM4_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
227 "0000000000000000";
228     signal      SPIM5_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
229 "0000000000000000";
230     signal      SPIM6_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
231 "0000000000000000";
232     signal      SPIM7_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
233 "0000000000000000";
234     signal      SPIM8_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
235 "0000000000000000";
236     signal      SPIM9_data_out      :STD_LOGIC_VECTOR(15 downto 0):=
237 "0000000000000000";
238     signal      SPIM10_data_out     :STD_LOGIC_VECTOR(15 downto 0):=
239 "0000000000000000";
240     signal      SPIM11_data_out     :STD_LOGIC_VECTOR(15 downto 0):=
241 "0000000000000000";
242     signal      SPIM12_data_out     :STD_LOGIC_VECTOR(15 downto 0):=
243 "0000000000000000";
244
245     signal      bank1_vaild_data    :STD_LOGIC := '0';
246     signal      bank2_vaild_data    :STD_LOGIC := '0';
247     signal      bank1_vaild_output  :STD_LOGIC := '0';
248     signal      bank2_vaild_output  :STD_LOGIC := '0';
249     signal      bank1_data_written  :STD_LOGIC := '0';
250     signal      bank2_data_written  :STD_LOGIC := '0';
251     signal      write_disable       :STD_LOGIC := '0';
252     signal      Byte_to_recvice     :STD_LOGIC := '0';
253     signal      SLAVE_sel           :STD_LOGIC:= '0';
254     signal      data_recieved       :STD_LOGIC := '0';
255     signal      reset_counter       :STD_LOGIC := '0';
256     signal      counter             :integer    := 0;
257
258     signal      SDRAM_read      :STD_LOGIC  := '0';
259     signal      SDRAM_write     :STD_LOGIC  := '0';
260
261     signal SPIS_di_req_o_buf    :std_logic:= '0';
262 -- preload lookahead data request line
263     signal SPIS_di_req_o        :std_logic:= '0';
264 -- preload lookahead data request line
265     signal SPIS_di_i            :std_logic_vector (7 downto 0) :=
266 (others => '0');  -- parallel load data in (clocked in on rising
267 edge of clk_i)
268     signal SPIS_wren_i          :std_logic := '0';
269 -- user data write enable
270     signal SPIS_wr_ack_o        :std_logic:= '0';
271 -- write acknowledge
272     signal SPIS_do_valid_o      :std_logic:= '0';
273 -- do_o data valid strobe, valid during one clk_i rising edge.
274     signal SPIS_do_o            :std_logic_vector (7 downto 0);
275
276     signal SPIS_start_transfer:  std_logic:= '0';
277
278     signal      SPIM_1_cpha    : STD_LOGIC := '0';
279 --spi clock phase
```

```
280     signal      SPIM_1_cont    : STD_LOGIC := '0';
281 --continuous mode command
282     signal      SPIM_1_addr    : INTEGER := 0;
283 --address of slave
284     signal      SPIM_1_busy    : STD_LOGIC := '0';
285 --busy / data ready signal
286     signal      SPIM_1_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
287 data received
288
289     signal      SPIM_2_cpha    : STD_LOGIC := '0';
290 --spi clock phase
291     signal      SPIM_2_cont    : STD_LOGIC := '0';
292 --continuous mode command
293     signal      SPIM_2_addr    : INTEGER:= 0;
294 --address of slave
295     signal      SPIM_2_busy    : STD_LOGIC:= '0';
296 --busy / data ready signal
297     signal      SPIM_2_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
298 data received
299
300     signal      SPIM_3_cpha    : STD_LOGIC := '0';
301 --spi clock phase
302     signal      SPIM_3_cont    : STD_LOGIC := '0';
303 --continuous mode command
304     signal      SPIM_3_addr    : INTEGER:= 0;
305 --address of slave
306     signal      SPIM_3_busy    : STD_LOGIC := '0';
307 --busy / data ready signal
308     signal      SPIM_3_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
309 data received
310
311     signal      SPIM 4 cpha    : STD LOGIC := '0';
312 --spi clock phase
313     signal      SPIM_4_cont    : STD_LOGIC := '0';
314 --continuous mode command
315     signal      SPIM_4_addr    : INTEGER:= 0;
316 --address of slave
317     signal      SPIM_4_busy    : STD_LOGIC := '0';
318 --busy / data ready signal
319     signal      SPIM_4_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
320 data received
321
322     signal      SPIM_5_cpha    : STD_LOGIC := '0';
323 --spi clock phase
324     signal      SPIM_5_cont    : STD_LOGIC := '0';
325 --continuous mode command
326     signal      SPIM_5_addr    : INTEGER:= 0;
327 --address of slave
328     signal      SPIM_5_busy    : STD_LOGIC := '0';
329 --busy / data ready signal
330     signal      SPIM_5_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
331 data received
332
333     signal      SPIM_6_cpha    : STD_LOGIC := '0';
334 --spi clock phase
```

```
335     signal      SPIM_6_cont    : STD_LOGIC := '0';
336 --continuous mode command
337     signal      SPIM_6_addr    : INTEGER:= 0;
338 --address of slave
339     signal      SPIM_6_busy    : STD_LOGIC := '0';
340 --busy / data ready signal
341     signal      SPIM_6_rx_data : STD_LOGIC_VECTOR(7 DOWNTO 0); --
342 data received
343
344
345 begin
346
347     pll0 : clock_gen
348       port map
349         (-- Clock in ports
350          CLK_IN1 => OSC_FPGA,
351           --Clock out ports
352          CLK_OUT1 =>  clk_100Mhz,
353           --Status and control signals
354          LOCKED => clock_locked);
355
356         sys_clk <=   clk_100Mhz;
357
358     clk_det: edge_detector
359       Port map ( clk         => sys_clk,
360                  signal_in   => RP_SPI_CE0N,
361                  output   => SPIS_start_transfer
362     );
363
364     req_det: edge_detector
365       Port map ( clk         => sys_clk,
366                  signal in   => SPIS di req o,
367                  output   => SPIS_di_req_o_buf
368     );
369
370     SPI_SLAVE_to_RPI : spi_slave
371         Generic map(
372         N           => 8,
373 -- 32bit serial word length is default
374         CPOL        => '0',                                        -
375 - SPI mode selection (mode 0 default)
376         CPHA        => '1',                                        -
377 - CPOL = clock polarity, CPHA = clock phase.
378         PREFETCH    => 2)                                       --
379 prefetch lookahead cycles
380         Port map(
381         clk_i       => sys_clk,                                  --
382 internal interface clock (clocks di/do registers)
383         spi_ssel_i  => RP_SPI_CE0N,                             --
384 spi bus slave select line
385         spi_sck_i   => SYS_SPI_SCK,                              --
386 spi bus sck clock (clocks the shift register core)
387         spi_mosi_i  => SYS_SPI_MOSI,                            --
388 spi bus mosi input
```

```
389         spi_miso_o  => SYS_SPI_MISO,                                --
390 spi bus spi_miso_o output
391         di_req_o    =>  SPIS_di_req_o,
392 -- preload lookahead data request line
393         di_i        => SPIS_di_i,                                    --
394 parallel load data in (clocked in on rising edge of clk_i)
395         wren_i      => SPIS_wren_i,                                   --
396 user data write enable
397         wr_ack_o    => SPIS_wr_ack_o,
398 -- write acknowledge
399         do_valid_o  => SPIS_do_valid_o,
400 -- do_o data valid strobe, valid during one clk_i rising edge.
401         do_o        => SPIS_do_o
402 -- parallel output (clocked out on falling clk_i)
403
404     );
405
406     SPIM1 : spi_master
407           GENERIC map(
408              slaves  => 2,  --number of spi slaves
409              d_width => 8   --data bus width
410              )
411           PORT map(
412              clock      => sys_clk,                                  --
413 system clock
414              reset_n    => SPIM_all_reset,
415 --asynchronous reset
416              enable     => SPIM_all_enable,
417 --initiate transaction
418              cpol       => SPIM_all_cpol,
419 --spi clock polarity
420              cpha       => SPIM all cpha,
421 --spi clock phase
422              cont       => SPIM_all_cont,
423 --continuous mode command
424              clk_div    => SPIM_all_clk_div,
425 --system clock cycles per 1/2 period of sclk
426              addr       => SPIM_all_addr,
427 --address of slave
428              tx_data    => SPIM_all_tx_data,  --data to transmit
429              miso       => SPIM1_MISO,                                -
430 -master in, slave out
431              sclk       => SPIM1_SCLK,                                -
432 -spi clock
433              ss_n       => SPIM1_CS(1 downto 0),   --slave select
434              mosi       => SPIM1_MOSI,                                -
435 -master out, slave in
436              busy       => SPIM_1_busy,                               -
437 -busy / data ready signal
438              rx_data    => SPIM_1_rx_data --data received
439           );
440     SPIM2 : spi_master
441           GENERIC map(
442              slaves  => 2,  --number of spi slaves
443              d_width => 8   --data bus width
```

```
444                )
445             PORT map(
446                clock      => sys_clk,                                --
447   system clock
448                reset_n    => SPIM_all_reset,
449   --asynchronous reset
450                enable     => SPIM_all_enable,
451   --initiate transaction
452                cpol       => SPIM_all_cpol,
453   --spi clock polarity
454                cpha       => SPIM_all_cpha,
455   --spi clock phase
456                cont       => SPIM_all_cont,
457   --continuous mode command
458                clk_div    => SPIM_all_clk_div,
459   --system clock cycles per 1/2 period of sclk
460                addr       => SPIM_all_addr,
461   --address of slave
462                tx_data    => SPIM_all_tx_data,  --data to transmit
463                miso       => SPIM2_MISO,                              -
464   -master in, slave out
465                sclk       => SPIM2_SCLK,                              -
466   -spi clock
467                ss_n       => SPIM2_CS(1 downto 0),   --slave select
468                mosi       => SPIM2_MOSI,
469   --master out, slave in
470                busy       => SPIM_2_busy,                             -
471   -busy / data ready signal
472                rx_data    => SPIM_2_rx_data --data received
473           );
474       SPIM3 : spi_master
475             GENERIC map(
476                slaves  => 2,  --number of spi slaves
477                d_width => 8   --data bus width
478                )
479             PORT map(
480                clock      => sys_clk,                                --
481   system clock
482                reset_n    => SPIM_all_reset,
483   --asynchronous reset
484                enable     => SPIM_all_enable,
485   --initiate transaction
486                cpol       => SPIM_all_cpol,
487   --spi clock polarity
488                cpha       => SPIM_all_cpha,
489   --spi clock phase
490                cont       => SPIM_all_cont,
491   --continuous mode command
492                clk_div    => SPIM_all_clk_div,
493   --system clock cycles per 1/2 period of sclk
494                addr       => SPIM_all_addr,
495   --address of slave
496                tx_data    => SPIM_all_tx_data,  --data to transmit
497                miso       => SPIM3_MISO,                              -
498   -master in, slave out
```

```
499                sclk       => SPIM3_SCLK,                                -
500   -spi clock
501                ss_n       => SPIM3_CS(1 downto 0),   --slave select
502                mosi       => SPIM3_MOSI,                                -
503   -master out, slave in
504                busy       => SPIM_3_busy,                               -
505   -busy / data ready signal
506                rx_data    => SPIM_3_rx_data --data received
507           );
508       SPIM4 :     spi_master
509             GENERIC map(
510                slaves  => 2,  --number of spi slaves
511                d_width => 8   --data bus width
512                )
513             PORT map(
514                clock      => sys_clk,                                  --
515   system clock
516                reset_n    => SPIM_all_reset,
517   --asynchronous reset
518                enable     => SPIM_all_enable,
519   --initiate transaction
520                cpol       => SPIM_all_cpol,
521   --spi clock polarity
522                cpha       => SPIM_all_cpha,
523   --spi clock phase
524                cont       => SPIM_all_cont,
525   --continuous mode command
526                clk_div    => SPIM_all_clk_div,
527   --system clock cycles per 1/2 period of sclk
528                addr       => SPIM_all_addr,
529   --address of slave
530                tx data    => SPIM all tx data,  --data to transmit
531                miso       => SPIM4_MISO,                                -
532   -master in, slave out
533                sclk       => SPIM4_SCLK,                                -
534   -spi clock
535                ss_n       => SPIM4_CS(1 downto 0),   --slave select
536                mosi       => SPIM4_MOSI,                                -
537   -master out, slave in
538                busy       => SPIM_4_busy,                               -
539   -busy / data ready signal
540                rx_data    => SPIM_4_rx_data --data received
541           );
542       SPIM5 : spi_master
543             GENERIC map(
544                slaves  => 2,  --number of spi slaves
545                d_width => 8   --data bus width
546                )
547             PORT map(
548                clock      => sys_clk,                                  --
549   system clock
550                reset_n    => SPIM_all_reset,
551   --asynchronous reset
552                enable     => SPIM_all_enable,
553   --initiate transaction
```

```
554                cpol       => SPIM_all_cpol,
555   --spi clock polarity
556                cpha       => SPIM_all_cpha,
557   --spi clock phase
558                cont       => SPIM_all_cont,
559   --continuous mode command
560                clk_div    => SPIM_all_clk_div,
561   --system clock cycles per 1/2 period of sclk
562                addr       => SPIM_all_addr,
563   --address of slave
564                tx_data    => SPIM_all_tx_data,  --data to transmit
565                miso       => SPIM5_MISO,                              -
566   -master in, slave out
567                sclk       => SPIM5_SCLK,                              -
568   -spi clock
569                ss_n       => SPIM5_CS(1 downto 0),   --slave select
570                mosi       => SPIM5_MOSI,                              -
571   -master out, slave in
572                busy       => SPIM_5_busy,                             -
573   -busy / data ready signal
574                rx_data    => SPIM_5_rx_data --data received
575           );
576       SPIM6 : spi_master
577             GENERIC map(
578                slaves  => 2,  --number of spi slaves
579                d_width => 8   --data bus width
580                )
581             PORT map(
582                clock      => sys_clk,                                --
583   system clock
584                reset_n    => SPIM_all_reset,
585   --asynchronous reset
586                enable     => SPIM_all_enable,
587   --initiate transaction
588                cpol       => SPIM_all_cpol,
589   --spi clock polarity
590                cpha       => SPIM_all_cpha,
591   --spi clock phase
592                cont       => SPIM_all_cont,
593   --continuous mode command
594                clk_div    => SPIM_all_clk_div,
595   --system clock cycles per 1/2 period of sclk
596                addr       => SPIM_all_addr,
597   --address of slave
598                tx_data    => SPIM_all_tx_data,
599   --data to transmit
600                miso       => SPIM6_MISO,                              -
601   -master in, slave out
602                sclk       => SPIM6_SCLK,                              -
603   -spi clock
604                ss_n       => SPIM6_CS(1 downto 0),                  --
605   slave select
606                mosi       => SPIM6_MOSI,                              -
607   -master out, slave in
```

```
608             busy       => SPIM_6_busy,                                -
609 -busy / data ready signal
610             rx_data    => SPIM_6_rx_data
611 --data received
612         );
613
614     start_when_ready : pro-
615 cess(SPIS_do_valid_o,SPIS_di_req_o,sys_clk,sys_reset) is
616     begin
617
618         if (sys_clk='1' and sys_clk'event) then
619             if SPIS_start_transfer = '1' then
620                 enable_spi_reciever <= '1';
621                 LED(0)<='1';
622             end if;
623         end if;
624
625
626     end process start_when_ready;
627
628     control_spi_master : process(sys_clk, sys_reset) is
629         begin
630
631         if (sys_clk='1' and sys_clk'event) then
632             if enable_spi_reciever = '1' then
633
634                 SPIM_all_cpol <= '0';
635                 SPIM_all_cpha <= '0';
636                 SPIM_all_clk_div <= 25;
637                 SPIM_all_enable <= '1';
638
639                 if  SPIM 1 busy = '0' then --switch der Adresssen
640                     SPIM_all_cont <= '0';
641                     if data_recieved = '1' then
642                         data_recieved <= '0';
643                         if( SLAVE_sel = '0')then
644                             SPIM_all_cont <= '1';
645                             SPIM_all_addr   <= 0;
646                             SLAVE_sel <='1';
647
648                         elsif( SLAVE_sel = '1')then
649                             SPIM_all_addr   <= 1;
650                             SPIM_all_cont   <= '1';
651                             SLAVE_sel <='0';
652                         end if;
653
654                     elsif (data_recieved  = '0') then
655                         data_recieved <= '1';
656                     end if;
657
658                 end if;
659
660             else
661             SPIM_all_enable <= '0';
662             SPIM_all_reset <= '1';
```

```
663             data_recieved <= '0';
664             SLAVE_sel <= '0';
665             end if;
666           end if;
667     end process control_spi_master;
668
669     prepare_store_data : process (SPIM_1_busy, sys_clk, sys_reset)
670 is
671
672     begin
673
674       if (sys_clk='1' and sys_clk'event) then
675             if ( SPIM_1_busy = '0') then
676                 if ((SLAVE_sel = '0') and (data_recieved = '0'))
677 then
678
679                     SPIM1_data_in(7 downto 0) <= SPIM_1_rx_data;
680                     SPIM3_data_in(7 downto 0) <= SPIM_2_rx_data;
681                     SPIM5_data_in(7 downto 0) <= SPIM_3_rx_data;
682                     SPIM7_data_in(7 downto 0) <= SPIM_4_rx_data;
683                     SPIM9_data_in(7 downto 0) <= SPIM_5_rx_data;
684                     SPIM11_data_in(7 downto 0) <= SPIM_6_rx_data;
685                     bank1_vaild_data <='0';
686
687                 elsif ((SLAVE_sel = '0') and (data_recieved = '1'))
688 then
689
690                     SPIM1_data_in(15 downto 8) <= SPIM_1_rx_data;
691                     SPIM3_data_in(15 downto 8) <= SPIM_2_rx_data;
692                     SPIM5_data_in(15 downto 8) <= SPIM_3_rx_data;
693                     SPIM7_data_in(15 downto 8) <= SPIM_4_rx_data;
694                     SPIM9 data in(15 downto 8) <= SPIM 5 rx data;
695                     SPIM11_data_in(15 downto 8) <= SPIM_6_rx_data;
696                     bank1_vaild_data <='1';
697
698                 elsif ((SLAVE_sel = '1') and (data_recieved = '0'))
699 then
700
701                     SPIM2_data_in(7 downto 0) <= SPIM_1_rx_data;
702                     SPIM4_data_in(7 downto 0) <= SPIM_2_rx_data;
703                     SPIM6_data_in(7 downto 0) <= SPIM_3_rx_data;
704                     SPIM8_data_in(7 downto 0) <= SPIM_4_rx_data;
705                     SPIM10_data_in(7 downto 0) <= SPIM_5_rx_data;
706                     SPIM12_data_in(7 downto 0) <= SPIM_6_rx_data;
707                     bank2_vaild_data <='0';
708
709                 elsif ((SLAVE_sel = '1') and (data_recieved = '1'))
710 then
711
712                     SPIM2_data_in(15 downto 8) <= SPIM_1_rx_data;
713                     SPIM4_data_in(15 downto 8) <= SPIM_2_rx_data;
714                     SPIM6_data_in(15 downto 8) <= SPIM_3_rx_data;
715                     SPIM8_data_in(15 downto 8) <= SPIM_4_rx_data;
716                     SPIM10_data_in(15 downto 8) <= SPIM_5_rx_data;
717                     SPIM12_data_in(15 downto 8) <= SPIM_6_rx_data;
```

```
718                     bank2_vaild_data <='1';
719
720                 end if;
721             end if;
722         end if;
723
724     end process;
725
726
727     Buffer_data: process (sys_clk,bank1_vaild_data,bank2_vaild_data)
728 is
729
730     begin
731         if(rising_edge(sys_clk)) then
732             if bank1_vaild_data  = '1' then
733
734             SPIM1_data_out <= SPIM1_data_in;
735             SPIM3_data_out <= SPIM3_data_in;
736             SPIM5_data_out <= SPIM5_data_in;
737             SPIM7_data_out <= SPIM7_data_in;
738             SPIM9_data_out <= SPIM9_data_in;
739             end if;
740
741             if bank2_vaild_data = '1' then
742
743             SPIM2_data_out <= SPIM2_data_in;
744             SPIM4_data_out <= SPIM4_data_in;
745             SPIM6_data_out <= SPIM6_data_in;
746             SPIM8_data_out <= SPIM8_data_in;
747             SPIM10_data_out <= SPIM10_data_in;
748             end if;
749         end if;
750     end process;
751
752
753
754     write_to_RPI : process(SPIS_start_transfer,sys_clk,sys_reset) is
755         begin
756
757         if (rising_edge(sys_clk)) then
758
759             if SPIS_di_req_o_buf = '1' and RP_SPI_CE0N = '0' then
760
761             LED(1)<='1';
762                     if counter = 1 then
763                         SPIS_wren_i <= '1';
764                         bank1_data_written <= '0';
765                         SPIS_di_i <= SPIM1_data_out(15 downto 8);
766                         counter <= counter +1;
767                     elsif counter = 2 then
768                         SPIS_wren_i <= '1';
769                         SPIS_di_i <= SPIM1_data_out(7 downto 0);
770                         counter <= counter +1;
771                     elsif counter =3 then
772                         SPIS_wren_i <= '1';
```

```
773                         SPIS_di_i <= SPIM2_data_out(15 downto 8);
774                         counter <= counter +1;
775                     elsif counter =4 then
776                         SPIS_wren_i <= '1';
777                         SPIS_di_i <= SPIM2_data_out(7 downto 0);
778                         counter <= counter +1;
779                     elsif counter =5 then
780                         SPIS_wren_i <= '1';
781                         SPIS_di_i <= SPIM3_data_out(15 downto 8);
782                         counter <= counter +1;
783                     elsif counter =6 then
784                         SPIS_wren_i <= '1';
785                         SPIS_di_i <= SPIM3_data_out(7 downto 0);
786                         counter <= counter +1;
787                     elsif counter =7 then
788                         SPIS_wren_i <= '1';
789                         SPIS_di_i <= SPIM4_data_out(15 downto 8);
790                         counter <= counter +1;
791                     elsif counter =8 then
792                         SPIS_wren_i <= '1';
793                         SPIS_di_i <= SPIM4_data_out(7 downto 0);
794                         counter <= counter +1;
795                     elsif counter =9 then
796                         SPIS_wren_i <= '1';
797                         SPIS_di_i <= SPIM5_data_out(15 downto 8);
798                         counter <= counter +1;
799                     elsif counter =10 then
800                         SPIS_wren_i <= '1';
801                         SPIS_di_i <= SPIM5_data_out(7 downto 0);
802                         counter <= counter +1;
803                     elsif counter =11 then
804                         SPIS wren i <= '1';
805                         SPIS_di_i <= SPIM6_data_out(15 downto 8);
806                         counter <= counter +1;
807                     elsif counter =12 then
808                         SPIS_wren_i <= '1';
809                         SPIS_di_i <= SPIM6_data_out(7 downto 0);
810                         counter <= counter +1;
811                     elsif counter =13 then
812                         bank2_data_written <= '0';
813                         SPIS_wren_i <= '1';
814                         SPIS_di_i <= SPIM7_data_out(15 downto 8);
815                         counter <= counter +1;
816                     elsif counter =14 then
817                         SPIS_wren_i <= '1';
818                         SPIS_di_i <= SPIM7_data_out(7 downto 0);
819                         counter <= counter +1;
820                     end if;
821
822                 if counter = 15 then
823                     counter <= 1;
824                     SPIS_wren_i <= '0';
825                     LED(1)<='0';
826
827                 end if;
```

```
828             end if;
829         end if;
830
831             if SPIS_start_transfer = '1'  then
832                 counter <= 1;
833                 SPIS_wren_i <= '0';
834             end if;
835
836     end process write_to_RPI;
837
838 end Behavioral;
```

Anhang G – Stage 2 Raspberry Pi Programm

```c
#include <bcm2835.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <stdint.h>
#include <string.h>
#include <errno.h>
#include <time.h>

#define TRUE    (1==1)
#define FALSE   (!TRUE)

#define SPI_PACKET_BYTES    17
#define SAMPLE_RATE         50000

uint16_t convertFrom8To16 (uint8_t dataFirst, uint8_t dataSecond) {
    uint16_t dataBoth = 0x0000;

    dataBoth = dataFirst;
    dataBoth = dataBoth << 8;
    dataBoth |= dataSecond;
    return dataBoth;
}

int main (int argc, char** argv){

    time_t nowtime;
    struct tm *today;
    char buffer [30];
    char date[9];

    uint64_t sample_time = 1000000/SAMPLE_RATE;
    uint32_t sample_rate = SAMPLE_RATE;
    unsigned int sample_cnt = 0, ii, samples = 50000, sample_timer =
sample_time;
    uint64_t start_time, end_time, now, time_now;
    uint8_t data[SPI_PACKET_BYTES] = {0};
    int16_t ch0_data = 0;
    int16_t ch1_data = 0;
    int16_t ch2_data = 0;
    int16_t ch3_data = 0;
    int16_t ch4_data = 0;
    int16_t ch5_data = 0;
    int16_t ch6_data = 0;
    int16_t ch7_data = 0;
    int16_t ch8_data = 0;
    int16_t ch9_data = 0;
    int16_t ch10_data = 0;
    int16_t ch11_data = 0;
    uint32_t bytes_to_transfer = 2;

    nowtime = time(NULL);
    struct tm *t = localtime(&nowtime);

    strftime( buffer, sizeof(buffer), "data_%d%m%y_%H%M%S.txt", t );

    if (argc < 2) {
        fprintf(stderr, "Usage: %s [-N SAMPLES] [-fs SAMPLE
RATE]\n\n", argv[0]);
    }
```

```
61          while (*argv) {
62              if (strcmp(*argv, "-N") == 0) {
63                  argv++;
64                  if (*argv)
65                      samples = atoi(*argv);
66              }
67              if (strcmp(*argv, "-fs") == 0) {
68                  argv++;
69                  if (*argv)
70                      sample_rate = atoi(*argv);
71                      sample_time = 1000000/sample_rate;
72              }
73              if (*argv)
74                  argv++;
75          }
76          printf("MSM-Aufnahme von %i Samples (fs = %.2f kHz) wird ges-
77      tartet...\n", samples, (float)sample_rate/1000);
78
79          /* Create Output File */
80          FILE *output_file = 0;
81          output_file = fopen(buffer, "w");
82
83          if(output_file == 0){
84              fprintf (stderr, "Can't create output file: %s\n", strerror
85      (errno));
86              exit (EXIT_FAILURE);
87          }
88          /* SPI Initialization */
89          if (!bcm2835_init()){
90              printf("bcm2835_init failed. Are you running as root??\n");
91              return 1;
92          }
93          bcm2835_spi_begin();
94          bcm2835_spi_setBitOrder(BCM2835_SPI_BIT_ORDER_MSBFIRST);      //
95      The default
96          bcm2835_spi_setDataMode(BCM2835_SPI_MODE1);                   //
97      The default
98          bcm2835_spi_setClockDivider(BCM2835_SPI_CLOCK_DIVIDER_16);  //
99      128 = 1.953125MHz ; 64 = 3,9MHz
100         bcm2835_spi_chipSelect(BCM2835_SPI_CS0);                      //
101     The default
102         bcm2835_spi_setChipSelectPolarity(BCM2835_SPI_CS0 , LOW);
103     // the default
104
105         start_time = bcm2835_st_read();
106         fprintf(output_file, "%i,%i\n",start_time);
107         fprintf(output_file, "ch0, ch1, ch2, ch3, ch4, ch5, ch6, us\n");
108         while(sample_cnt < samples){
109             now = bcm2835_st_read();
110
111             sample_cnt++;
112             /* Request Data from MSM-Module */
113             bcm2835_spi_transfern(data, SPI_PACKET_BYTES);
114             time_now = bcm2835_st_read();
115             now = time_now - start_time;
```

```
116
117             ch0_data = convertFrom8To16(data[1],data[2]);
118             ch1_data = convertFrom8To16(data[3],data[4]);
119             ch2_data = convertFrom8To16(data[5],data[6]);
120             ch3_data = convertFrom8To16(data[7],data[8]);
121             ch4_data = convertFrom8To16(data[9],data[10]);
122             ch5_data = convertFrom8To16(data[11],data[12]);
123             ch6_data = convertFrom8To16(data[13],data[14]);
124
125             fprintf(output_file, "%i,%i,%i,%i,%i,%i,%i,%i, %i\n",
126     ch0_data, ch1_data, ch2_data, ch3_data, ch4_data, ch5_data,
127     ch6_data, now);
128         }
129
130         end_time = bcm2835_st_read();
131         float duration = (float)(end_time - start_time)/1000000;
132         printf("Dauer: %f s\n", duration);
133         float samplerate = ((float)samples/duration)/1000;
134         printf("Samplerate: %.2f kHz\n", samplerate);
135
136         fclose(output_file);
137         bcm2835_spi_end();
138         bcm2835_close();
139
140         return 0;
141     }
```

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „– bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] – ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Dieses Blatt, mit der folgenden Erklärung, ist nach Fertigstellung der Abschlussarbeit durch den Studierenden auszufüllen und jeweils mit Originalunterschrift als letztes Blatt in das Prüfungsexemplar der Abschlussarbeit einzubinden.
Eine unrichtig abgegebene Erklärung kann -auch nachträglich- zur Ungültigkeit des Studienabschlusses führen.

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: Wenzel

Vorname: Tobias Morten

dass ich die vorliegende Masterarbeit bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines synchronen hochkanaligen Messsystems für Luft- und Körperschall

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

- die folgende Aussage ist bei Gruppenarbeiten auszufüllen und entfällt bei Einzelarbeiten -

Die Kennzeichnung der von mir erstellten und verantworteten Teile der Masterarbeit ist erfolgt durch:

Hamburg	09.05.2016	
Ort	Datum	Unterschrift im Original