FH Westküste
Wirtschaft und Technik

HAW HAMBURG

Masterarbeit

Max Keste

Realisierung eines Audio-Analyzers auf Basis eines DSP-Entwicklungsmoduls

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

Max Keste

Realisierung eines Audio-Analyzers auf Basis eines DSP-Entwicklungsmoduls

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

Betreuender Prüfer : Prof. Dr.-Ing. Hans-Dieter Schütte
Zweitgutachter : Prof. Dr.-Ing. Ulrich Sauvagerd

Abgegeben am 06.11.2019

Max Keste

Thema der Masterarbeit

Realisierung eines Audio-Analyzers auf Basis eines DSP-Entwicklungsmoduls

Stichworte

DSP, TI, C6000, C6657, Signalverarbeitung, Dual-Core, Inter-Prozessor-Kommunikation, D.Module2, d.signt, Audio-Analyzer, Ethernet, Webbrowser

Kurzzusammenfassung

Masterarbeit die die Realisierung eines Audio-Analyzers auf Basis des D.Module2.C6657 der Firma d.signt beschreibt. Hard- und Softwareentwicklung um ein eigenständiges Messsystem, inklusive eines User-Interfaces, zu erhalten.

Max Keste

Title of the paper

Realisation of an audio-analyzer based on a dsp-evaluationmodul

Keywords

DSP, TI, C6000, C6657, signal processing, dual-core, inter processor communication, D.Module2, d.signt, audio-analyzer, ethernet, webbrowser

Abstract

Masterthesis that describes the realisation of an audio-analyzer on the D.Module2.C6657 from d.signt. Hard- and software development to make a stand-alone measuring system including an user-interface.

Inhaltsverzeichnis

Abkürzungsverzeichnis

ADC	Analog-Digital Wandler
CCS	Code Composer Studio
CGI	Common Gateway Interface
DAC	Digital-Analog Wandler
DSP	digitaler Signalprozessor
DUT	Device Under Test
EFS	Embedded File System
EVM	Entwicklungsmodul
FFT	Fast-Fourier-Transformation
GPIO	General Purpose Input-Output
HE	Höheneinheit
HWI	Hardware-Interrupt
IPC	Inter-Processor-Communication
NDK	Network Development Kit
RTOS	Echtzeitbetriebssystem
SDK	Software Development Kit
SPI	Serial Peripheral Interface
SWI	Software-Interrupt
TI	Texas Instruments
UART	Universal Asynchronous Receiver and Transmitter
UI	User Interface

Abbildungsverzeichnis

Tabellenverzeichnis

Quellcodeverzeichnis

1. Einleitung

Digitale Signalprozessoren sind heutzutage ein wichtiger Bestandteil der digitalen Signalverarbeitung. Die Hersteller von DSPs entwickeln daher immer leistungsfähigere Prozessoren. In dieser Arbeit wird deshalb die Verwendung eines DSPs als eigenständiges Messsystem, anhand eines Audio-Analyzers untersucht.
Ziel ist es, anhand der Realisierung des Messsystems, eine Einschätzung zu erhalten, wie geeignet heutige DSPs als eigenständige Systeme sind, die sowohl die echtzeitkritische Signalverarbeitung übernehmen, als auch die Bedienung des Benutzers durch ein User-Interface ermöglichen. Ebenfalls soll die Kommunikation einzelner Rechenkerne in Mehrprozessorarchitekturen untersucht werden.

Um den Audio-Analyzer zu realisieren, wird zuerst eine geeignete Hardwareplattform ausgewählt, indem verschiedene EVMs untersucht werden. Dieses EVM bildet anschließend die Basis der weiteren Entwicklung und beinhaltet neben dem DSP die notwendigen digitalen Schaltungen, die zum Betrieb notwendig sind. Nach Auswahl eines EVM erfolgt die Entwicklung der zusätzlich benötigten analogen Hardware und eines Gehäuses, so dass am Ende ein fertiges Messgerät zur Verfügung steht. Dazu wurden bereits vorhandene Audio-Analyzer untersucht und anhand dessen eine Spezifikation für den eigenen Audio-Analyzer erstellt. Die Softwareentwicklung umfasst die Implementierung der Signalverarbeitung und die Realisierung eines User-Interfaces um den Audio-Analyzer zu bedienen.

2. Grundlagen

2.1. digitale Signalverarbeitung

Die digitale Signalverarbeitung befasst sich mit der Wandlung analoger Signale in eine digitale Form, der anschließenden Weiterverarbeitung und schlussendlich der Rückwandlung in ein analoges Signal (siehe Abbildung 2.1). Der wesentliche Unterschied des digitalen Signals zum analogen Signal besteht darin, dass es aufgrund der Abtastung zeitdiskret und aufgrund der Quantisierung wertdiskret ist.

Abbildung 2.1.: Signalfluss in der digitalen Signalverarbeitung

Entscheidend bei der digitalen Signalverarbeitung ist, dass aus dem digitalisierten Signal, das analoge Signal wieder exakt rekonstruiert werden kann. Das Abtasttheorem von Shannon, welches auch als Nyquist-Theorem bezeichnet wird, besagt, das dies gegeben ist, wenn ein analoges Signal mit der maximalen Signalfrequenz f_{max} mit einer Abtastfrequenz $f_A > 2 * f_{max}$ abgetastet wird. Die Wahl der Abtastfrequenz bestimmt demnach die maximal mögliche erfassbare Signalfrequenz.
Ebenfalls entscheidend bei der digitalen Signalverarbeitung ist die Quantisierung des Signals. Diese entsteht durch die wertdiskrete Darstellung des analogen Signals in der digitalen Welt. Als Beispiel soll hier ein analoges Sinussignal mit einer maximalen Amplitude von 1 V_{PP} digitalisiert werden. Während das analoge Signal jeden Wert zwischen -0,5 V und 0,5 V annehmen

kann, ist die maximal mögliche Anzahl an Werten, die das digitale Signal annehmen kann, abhängig von der Wortlänge des Analog-Digital Wandler (ADC). Beträgt diese beispielsweise 8 Bit, sind maximal 256 Werte zwischen -0,5 V und 0.5 V möglich. Der kleinstmögliche Abstand zwischen zwei Werten beträgt demnach $\frac{1V}{256Bit} * 1Bit = 3,9mV$. Aufgrund dieser Quantisierung wird dem Signal ein Fehler hinzugefügt, der als Quantisierungsrauschen bezeichnet wird. Je größer die Wortbreite des ADC ist, desto geringer ist das Quantisierungsrauschen.

Für die Rückwandlung des digitalen Signals in ein analoges Signal wird ein Digital-Analog Wandler (DAC) benötigt. Dieser erzeugt aus den digitalen Eingangswerten eine analoge Spannung und rekonstruiert somit das analoge Signal. Wie auch beim ADC ist die Abtastfrequenz und die Quantisierung entscheidend.

Mit dem technologischen Fortschritt und Erscheinen hochauflösender ADCs bzw. DACs sowie leistungsfähigen Prozessoren, ist es heutzutage möglich, Rechenoperationen in Echtzeit durchzuführen, die bis dahin, aufgrund fehlender Ressourcen, nur theoretisch existiert haben. Das digitale Signal kann somit im DSP mit verschiedenen Operationen manipuliert werden und anschließend wieder in ein analoges Signal gewandelt werden. So können beispielsweise Filter mit sehr hohem Grad realisiert werden, die in der analogen Welt nur unter einem enormen Schaltungsaufwand möglich sind. Dies ermöglichte auch die Anwendung neuer Algorithmen, wie beispielsweise der Fast-Fourier-Transformation (FFT), um das Spektrum eines digitalen Signals zu berechnen.

2.2. digitale Signalprozessoren

DSPs sind Prozessoren, die speziell für die Anwendung in der digitalen Signalverarbeitung optimiert sind. Sie besitzen einen auf die Signalverarbeitung optimierten Befehlssatz und Komponenten, welche Standardprozessoren nicht zwingend besitzen. Sie können, im Gegensatz zu normalen Mikrocontrollern, Rechenoperation wie die Multiplikation und Addition (MAC) von zwei Werten oder Gleitkommaberechnungen (FLOPs) optimiert ausführen. Um dies zu erzielen, beinhaltet ein DSP häufig mehrere arithmetische Einheiten, die Gleitkommaberechnungen und Multiplikationen, in einem Zyklus durchführen können. Als Leistungsmerkmal werden daher häufig die Anzahl an Rechenoperationen, die pro Sekunde ausgeführt werden

können, angegeben. Der in dieser Arbeit verwendete DSP C6657 von Texas Instruments (TI) erreicht somit beispielsweise 40 GMAC/s oder 20 GFLOP/s pro Kern.

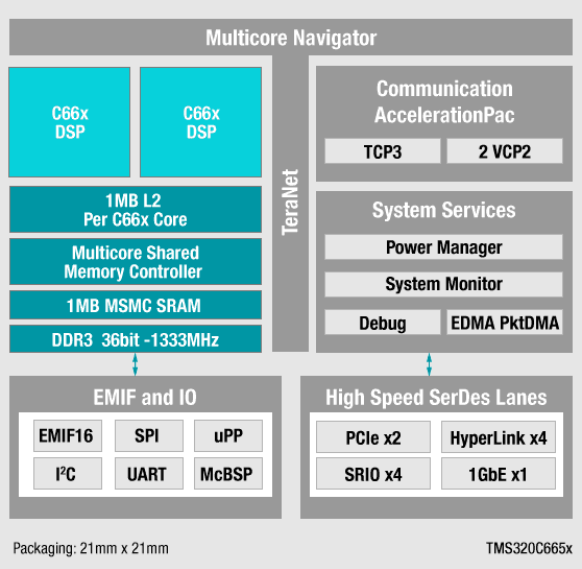


Abbildung 2.2.: Blockschaltbild der C665x DSPs von TI (Quelle: www.ti.com)

In Abbildung 2.2 ist als Beispiel für einen DSP das Blockschaltbild der C665x DSPs von TI zu sehen. Wie zu erkennen ist, sind hier zwei DSP-Kerne integriert. Sie besitzen großzügige schnelle Speicher, die direkt an den Prozessor angebunden sind (L2, DDR3). Häufig fehlt dafür aber der in Standardprozessoren vorhandene Flashspeicher in dem sich der Programmcode befindet. Dieser muss dann aus einem externen, nichtflüchtigen Speicher nachgeladen werden.
Weiterhin besitzen DSPs häufig spezielle Schnittstellen, die in der digitalen Signalverarbeitung genutzt werden und mit denen Analog-Interfaces angebunden werden können. Hier ist beispielsweise die Multichannel Buffered Serial Port(McBSP)-Schnittstelle zu nennen.

2.3. Audio-Analyzer

Audio-Analyzer sind Messgeräte, die in der Lage sind, audiotechnische Messungen durchzuführen. Dazu müssen sie Signale im Audiofrequenzbereich, dem vom Menschen hörbaren Bereich, erfassen und analysieren können. Dieser liegt im Bereich zwischen 20 Hz und 20 kHz. Oft reicht der messbare Bereich darüber hinaus, um Geräte für die Audioverarbeitung zu analysieren. Hier sind Abtastfrequenzen von 44,1 kHz (Audio-CD) bis hin zu 192 kHz (Tonstudio) üblich.
Um Geräte und Systeme zur Audioverarbeitung auf ihre Funktion hin zu überprüfen und zu spezifizieren, beinhaltet ein Audio-Analyzer verschiedene Messverfahren zur Analyse und meist einen Signalgenerator zur Erzeugung von Testsignalen.
Die Produktpalette von Audio-Analyzern reicht von einer Softwarelösung, die vorhandene PC-Hardware nutzt, bis zu eigenständigen Messgeräten, die alle notwendigen Komponenten beinhalten.

Als Beispiel für eine Softwarelösung ist hier der „audioTester“ zu nennen. Dies ist eine Software (siehe Abbildung 2.3), die in Zusammenarbeit mit einer hochwertigen Soundkarte und einem Windows-PC, einen Audio-Analyzer bildet. Die Spezifikation der Soundkarte bestimmt wesentlich die Qualität der Messung und die Messmöglichkeiten.

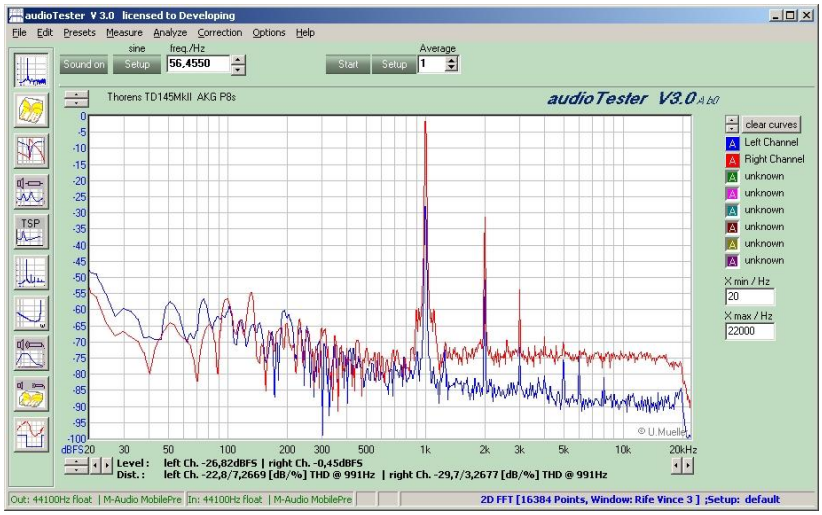


Abbildung 2.3.: Audio-Analyzersoftware „audioTester“(Quelle: www.audiotester.de)

Als Audio-Analyzer, der als Kombination aus Soft- und Hardware ein eigenständiges Messsystem bildet, ist der „UPV“ von der Firma „Rohde und Schwarz“ zu nennen. Dieser benötigt keine

zusätzlichen Komponenten, denn Signalerfassung und -erzeugung, Messung, Bedienung und Darstellung sind in einem Gerät zusammengefasst. Die Qualität der Messungen sind vom

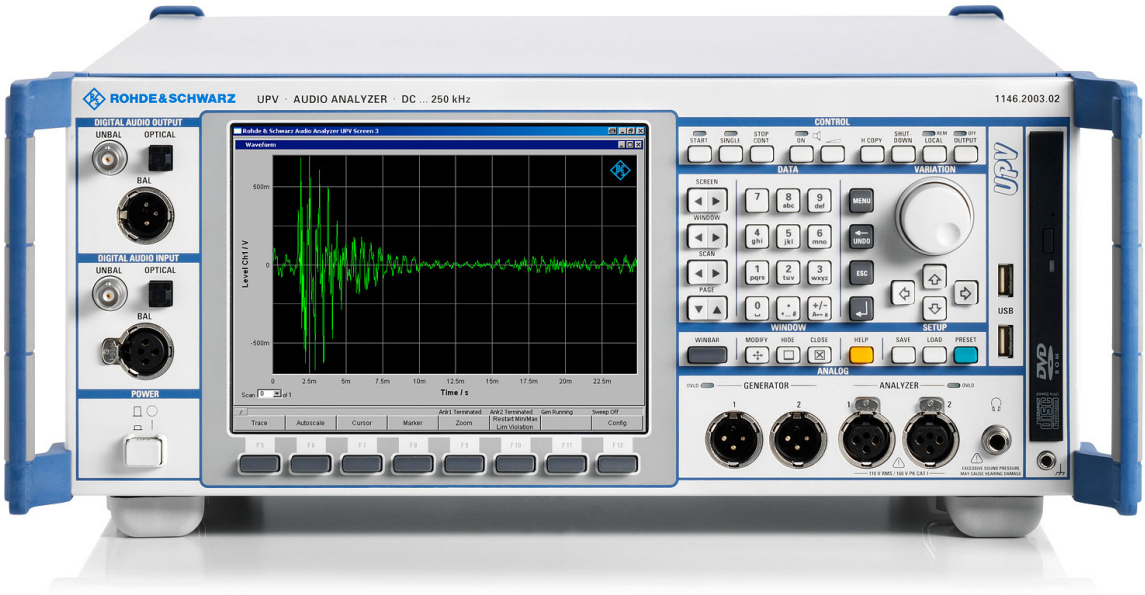


Abbildung 2.4.: Audio-Analyzer „Rohde und Schwarz UPV“(Quelle: www.rohde-schwarz.com)

Hersteller durch die verwendete Hardware vorgegeben und die Messmöglichkeiten können durch optionale Hardwarekomponenten erweitert werden. Dies beinhaltet beispielsweise die Erweiterung des Messgerätes um digitale Schnittstellen oder die Vergrößerung des Messbereichs.
Wie in Abbildung 2.4 zu sehen ist, erhält man mit dem UPV ein eigenständiges Messgerät. Es ist ein Monitor für die Anzeige vorhanden und Einstellungen können mit den Bedienelementen an der Frontseite erfolgen. Analoge sowie digitale Signale können über verschiedene Anschlüsse angeschlossen und analysiert werden.

2.4. Messungen im Audiobereich

In der Audiotechnik gibt es verschiedene Messungen, die genutzt werden, um eine Audioanalyse mithilfe eines Audio-Analyzers durchzuführen. Grundsätzlich wird dafür der in Abbildung 2.5 dargestellte, schematische Messaufbau benötigt.

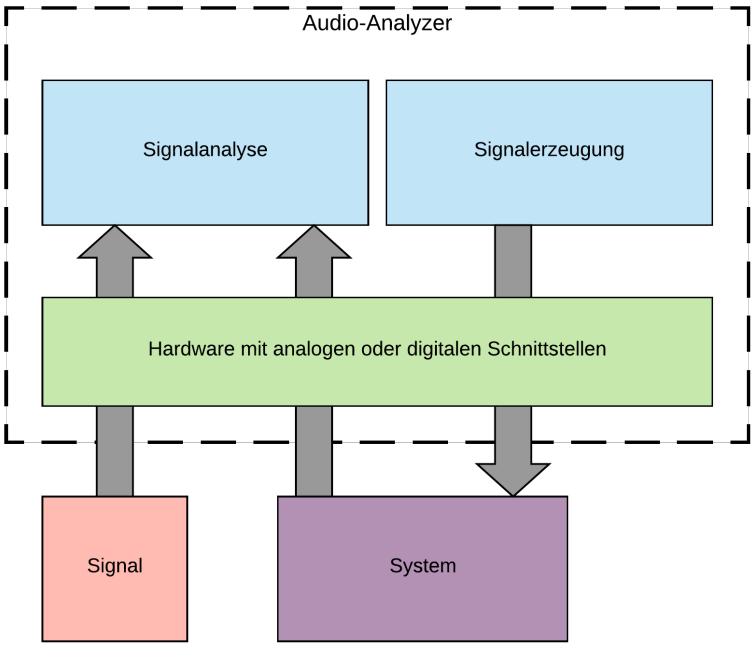


Abbildung 2.5.: prinzipieller Messaufbau eines Audio-Analyzers

Zur Messung von Signalen müssen diese über die Hardwarekomponente erfasst werden. Anschließend kann die Software des Audio-Analyzers eine Analyse dieser Signale durchführen. So kann mit einem Audio-Analyzer die Amplitude und die Frequenz des Signals bestimmt werden. Zusätzlich kann eine Spektralanalyse erfolgen, aus der sich weitere Signaleigenschaften bestimmen lassen. Dazu gehört unter anderem das Bestimmen des Signal-Rausch-Abstandes (SNR) oder die Messung der harmonischen Verzerrung (Total Harmonic Distortion (THD)). Sollen Systeme analysiert werden, benötigt der Audio-Analyzer zusätzlich eine Möglichkeit, Signale zu erzeugen. Mit diesen Signalen kann das System angeregt werden und die Antwort kann anschließend wieder analysiert werden. So sind Messungen des Frequenzgangs von Systemen möglich. Oft besitzen Audio-Analyzer mehrere Eingänge, so dass beispielsweise Stereosignale gleichzeitig erfasst werden können. Oft kann auch die Phasenverschiebung zwischen zwei Eingangssignalen gemessen werden.

3. Konzeptionierung

3.1. Entwurf des Audio-Analyzers

Im ersten Schritt dieser Arbeit ist ein Konzeptentwurf erstellt worden, der definiert, welche Komponenten der Audio-Analyzer haben soll und welche Entwicklungsschritte für die Realisierung notwendig sind. Der zu realisierende Audio-Analyzer soll ein eigenständiges Messgerät, also eine Kombination aus Soft- und Hardware, bilden. In Abbildung 3.1 ist der entstandene Konzeptentwurf dargestellt.

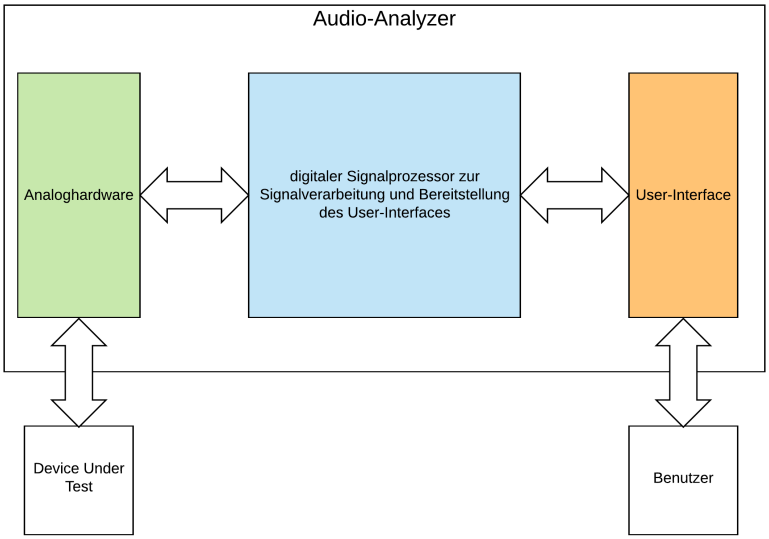


Abbildung 3.1.: Konzeptentwurf des Audio-Analyzers

Wie zu erkennen ist, beinhaltet der Audio-Analyzer drei grundlegende Komponenten.
Im Mittelpunkt steht der DSP, welcher die Signalverarbeitung durchführt und ein User-Interface bereitstellt. Die Analoghardware dient zur Verbindung des DSP mit dem zu testenden Gerät

(Device Under Test (DUT)). Hier sind geeignete Schnittstellen und analoge Schaltungen zu definieren um das DUT an den Audio-Analyzer anzuschließen.
Die Bedienung des Audio-Analyzers durch den Benutzer erfolgt über das User-Interface. Wie dieses User-Interface realisiert werden soll, wird später definiert, denn es ist abhängig von den Möglichkeiten des verwendeten EVM. Dies kann beispielsweise ein Display sein, das direkt am DSP angeschlossen ist oder die Implementierung eines Webservers auf den von einem Client zugegriffen werden kann.
Außerdem ist ein geeignetes Gehäuse zu entwerfen, in dem die Komponenten untergebracht sind und so ein vollwertiges Messsystem entsteht.

3.2. durchzuführende Entwicklungsschritte

Nachdem die einzelnen Komponenten im Konzept festgelegt wurden, konnten die weiteren Entwicklungsschritte festgelegt werden. Diese gliedern sich in:

- Auswahl einer Hardwareplattform
- Erstellung eines Gehäusekonzepts
- Entwicklung der Analoghardware
- Entwicklung der Softwarekomponenten
- Inbetriebnahme und Test

Da der DSP den Mittelpunkt des Audio-Analyzers bildet, musste zuerst ein geeignetes EVM ausgewählt werden, welches die Hardwareplattform bildet und den DSP sowie notwendige Analog-Interfaces, bestehend aus ADC und DAC, beinhaltet. Nach der Auswahl des EVM wurde die Analoghardware passend für das EVM entwickelt. Parallel dazu musste ein Gehäusekonzept erstellt werden, um die Komponenten unterzubringen. Abschließend erfolgte die Entwicklung der notwendigen Softwarekomponenten. Dazu gehört die Signalverarbeitung, die in einem Audio-Analyzer durchgeführt werden muss, sowie die Erstellung eines User-Interfaces.

3.3. Spezifikation des Audio-Analyzers

Um die Entwicklung des Audio-Analyzers durchzuführen, musste eine Spezifikation festgelegt werden. Diese wurde während der Konzeptionierung erstellt und im Verlauf der Arbeit auf Grundlage der gewonnenen Erkenntnisse erweitert und angepasst. Einige grundlegende Spezifikationen können an dieser Stelle bereits definiert werden. Als Vorlage dient der UPV, dessen Spezifikation einen Richtwert für die eigene Entwicklung bildet.

Der Audio-Analyzer muss grundsätzlich in der Lage sein, Signale im Audiofrequenzbereich erfassen und auch erzeugen zu können. Dazu sind analoge Ein- und Ausgänge notwendig, die einen Frequenzbereich von ca. 20 Hz bis mindestens 22 kHz abdecken. Dazu ist eine Abtastfrequenz von mindestens 44 kHz erforderlich. Die Spannungspegel von Signalen im Audiobereich liegen meist im Bereich von $1V_{eff}$. Die verwendeten ADCs und DACs müssen dementsprechend diesen Spannungsbereich abdecken können. Besitzt das DUT die Möglichkeit, das Signal zu verstärken, ist es sinnvoll auch höhere Eingangsspannungen erfassen zu können. Die endgültige Spezifikation des Audio-Analyzers ist im Anhang A dieser Arbeit zu finden.

4. Auswahl einer Hardwareplattform

4.1. Untersuchung vorhandener Hardware

Bei der Auswahl einer geeigneten Hardwareplattform wurden zuerst zwei an der FHW vorhandene EVMs untersucht. Dies ist zum einen das OMAP-LCDK und zum anderen das UniDAQ2. Beide EVMs werden anhand ihrer theoretischen Daten und einer selbst durchgeführten Messung miteinander verglichen. Anschließend wird die Tauglichkeit der EVMs für die Verwendung in dieser Arbeit diskutiert.

4.1.1. OMAP-LCDK

Das OMAP-LCDK ist ein von TI entwickeltes und vertriebenes EVM für den OMAP-L138 Prozessor. Dieser Prozessor besitzt laut dem User Guide [1] neben einem 456 MHz ARM-Prozessor zusätzlich einen 456 MHz DSP aus der C6000-Familie. Das EVM verfügt, wie in Abbildung 4.1 zu sehen, über diverse Anschlussmöglichkeiten für beispielsweise eine Ethernetverbindung oder den Anschluss eines Monitors über VGA. Für die Audiosignalverarbeitung ist ein Audio-Codec vom Typ TLV320AIC3106 vorhanden, dessen Line Aus- bzw. Eingang als 3,5 mm Klinkenbuchse ausgeführt ist. Es stehen jeweils ein Stereo Aus- und Eingang zur Verfügung.
Für die Programmierung stellt TI ein Linux Software Development Kit (SDK), sowie das betriebseigene Echtzeitbetriebssystem (RTOS) SYS/BIOS zur Verfügung.
Um das EVM zu untersuchen, wurde zuerst das auf einer mitgelieferten SD-Karte befindliche Linux-Betriebssystem installiert. Anschließend war es möglich, eine Netzwerkverbindung einzurichten. Da das OMAP-LCDK über einen VGA-Anschluss verfügt und somit die Möglichkeit bietet, einen Monitor anzuschließen, wurde dies als nächstes versucht. Vorteile die sich daraus ergeben würden, sind die Möglichkeit zur Erzeugung einer graphischen Oberfläche

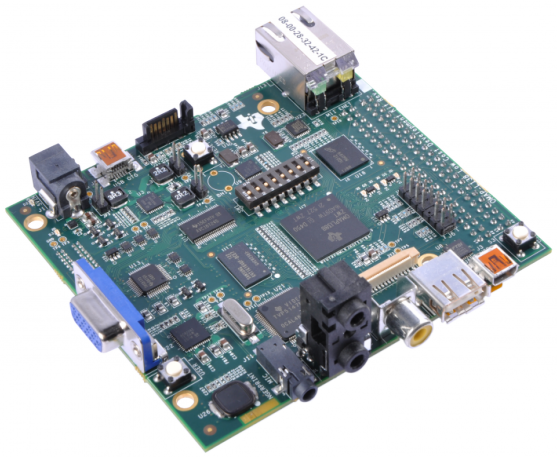


Abbildung 4.1.: OMAP-LCDK (Quelle: TI)

als User Interface (UI) mit beispielsweise QT. Hierbei stellte sich jedoch heraus, dass das verfügbare Linux-Betriebssystem nicht über die notwendigen Treiber zur Benutzung des auf dem OMAP-LCDK befindlichen Video-Codecs verfügt. Eine Implementierung über eine Erweiterung des Linux-Kernels mit dem Linux SDK wurde recherchiert und aufgrund der Komplexität und fehlenden Tutorials nicht weiter verfolgt.

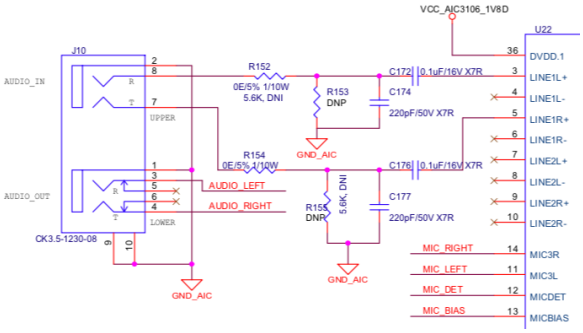


Abbildung 4.2.: Schaltplan des analogen Eingangspfads des OMAP-LCDK (Quelle: TI)

Der Ausschnitt des Schaltplans in Abbildung 4.2 zeigt den für die analoge Signalführung relevanten Schaltungsteil. Zu erkennen ist, das im Signalpfad des Eingangs ein RC-Tiefpass, bestehend aus R152 und C174, bzw. aus R154 und C177, vorhanden ist.
Mit der Formel

$$f_g = \frac{1}{2 \cdot \pi \cdot RC} \tag{4.1}$$

zur Berechnung der Grenzfrequenz eines RC-Gliedes, ergibt sich durch Einsetzen der Werte für $R = 5,6k\Omega$ und $C = 220pF$ aus dem Schaltplan eine Grenzfrequenz von

$$\frac{1}{2 \cdot \pi \cdot 5,6k\Omega \cdot 220pF} = 129,18kHz.$$

Desweiteren bildet der Kondensator C172, bzw. C176, die DC-Entkopplung am Eingang. Mit dem Eingangswiderstand des Audio-Codec, der laut Datenblatt [2] $20k\Omega$ beträgt, entsteht so ein Hochpass, dessen Grenzfrequenz sich ebenfalls durch Einsetzen der Werte in die Formel 4.1 berechnen lässt. Es ergibt sich eine Grenzfrequenz von

$$\frac{1}{2 \cdot \pi \cdot 20k\Omega \cdot 0,1\mu F} = 79,6Hz.$$

Im Signalpfad des Audioausgangs befindet sich, wie in Abbildung 4.3 zu erkennen ist, ebenfalls ein RC-Hochpass, bestehend aus C178 und R156 für den linken und C179 und R157 für den rechten Kanal. Die Grenzfrequenz nach Formel 4.1 beträgt

$$\frac{1}{2 \cdot \pi \cdot 20k\Omega \cdot 10\mu F} = 0,796Hz.$$

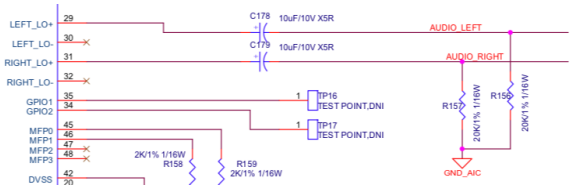


Abbildung 4.3.: Schaltplan des analogen Ausgangspfads des OMAP-LCDK (Quelle: TI)

Dem Datenblatt des Audio-Codec kann ebenfalls entnommen werden, dass die internen AD-Wandler mit einem Oversampling arbeiten. Die interne Abtastrate ist um das 128-fache größer, als die eigentlich eingestellte. Nachgeschaltete, integrierte Dezimationsfilter führen eine Antialias-Filterung durch und teilen die Abtastrate wieder auf den eingestellten Wert herunter. Der Audio-Codec hat außerdem weitere digitale Komponenten, um das Signal zu verändern. Als Beispiel kann hier die automatische Verstärkung genannt werden. Diese kann aktiviert werden, um bei der Aufnahme von Audiosignalen gleichbleibende Pegel zu erhalten.

4.1.2. UniDAQ2

Das UniDAQ2 ist ein von der Firma d.signt vertriebenes EVM auf Basis eines von TI hergestellten DSP der C6000-Familie. Es besitzt denselben DSP-Prozessorkern wie das OMAP-LCDK, jedoch ohne den ARM-Prozessor. Das UniDAQ2 ist, wie in Abbildung 4.4 zu sehen, mit einem Ethernet-Anschluss ausgestattet. Ein Display kann über einen dafür vorgesehenen Platinenstecker angeschlossen werden. Eine kurze Recherche hat allerdings ergeben, dass das passende Display vom Hersteller nicht weiter produziert wird und somit nicht verfügbar ist. Die Audiosignalverarbeitung erfolgt über A/D- bzw. D/A-Wandler, die an den Prozessor, über

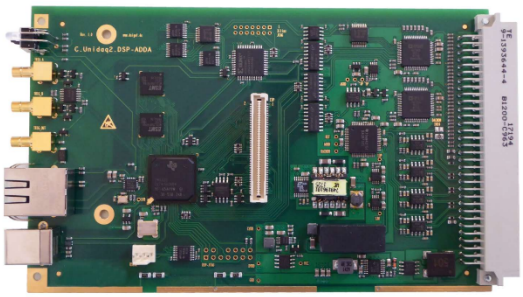


Abbildung 4.4.: d.signt UniDAQ2 (Quelle: d.signt)

eine Schnittstelle, angeschlossen sind. Es stehen hier 16 Aus- und 8 Eingänge zur Verfügung. Als Softwarepaket stellt d.signt einige Beispielprojekte bereit, die auf eigenen, zum Betrieb des EVM notwendigen Bibliotheken, basieren und zusätzlich das Prozessor SDK von TI nutzen.
Als Beispiel ist hier das bereits implementierte Demoprogramm zu erwähnen, welches bei Anschluss des EVM an ein bestehendes Netzwerk einen Webserver bereitstellt und dem Nutzer die Möglichkeit bietet, die analogen Eingänge grafisch, wie bei einem Oszilloskop anzusehen.

Dem Datenblatt des UniDAQ2 [3] kann entnommen werden, dass zur Antialias-Filterung an den AD-Wandlern ein Butterworth-Tiefpass 2. Ordnung mit einer Grenzfrequenz von 15 kHz (5V), bzw. 23 kHz (10V), abhängig von der Versorgungsspannung, integriert ist. An den Ausgängen befinden sich ebenfalls Butterworth-Tiefpässe mit einer Grenzfrequenz von 50 kHz.

4.1.3. Theoretischer Vergleich

Für den theoretischen Vergleich beider EVMs sind in der Tabelle 4.1 die Kernspezifikationen nebeneinander aufgelistet.

Tabelle 4.1.: Kernspezifikation von OMAP-LCDK und UniDAQ2

	OMAP-LCDK	UniDAQ2
Prozessor	456 MHz (DSP) + 456 MHz (ARM)	456 MHz (DSP)
A/D Interface	TLV320AIC3106 Audio Codec	AD7606 (Analog Devices)
Anzahl Eingänge	1x Stereo	16
D/A Interface	TLV320AIC3106 Audio Codec	DAC8718 (TI)
Anzahl Ausgänge	1x Sereo	8
Abtastrate	bis 96 kHz	0 - 175 kHz
Antialias Filter	Digitaler FIR TP bei Fs/2	2. Ord BW TP bei 15 kHz(5V) / 23 kHz (10V)
Kopplung	AC	DC

Wie zu erkennen ist, verwendet das OMAP-LCDK einen Prozessor, der einen DSP- und einen ARM-Prozessor beinhaltet, wohingegen das UniDAQ2 lediglich einen DSP-Kern besitzt. Dieser ist jedoch bei beiden Prozessoren identisch. Die Vorteile der Zweiprozessorarchitektur des OMAP-LCDK sind z.B., dass die zeitkritische Signalverarbeitung und die Benutzeroberfläche auf unterschiedlichen Kernen realisiert werden können. Hier muss anschließend jedoch ein größerer Aufwand betrieben werden, um Daten zwischen diesen beiden Kernen auszutauschen. Das bereits erwähnte Demoprogramm des UniDAQ2 zeigt, dass die Realisierung einer Benutzeroberfläche über einen Webserver und eine einfache Signalverarbeitung in einem Kern implementiert werden kann. Ob die Verwendung von zwei Prozessorkernen notwendig ist oder der DSP-Kern ausreichend ist, lässt sich somit nicht eindeutig sagen.
Ein weiterer großer Unterschied ist das Analog-Interface, welches beim OMAP-LCDK mit einem Audio-Codec und beim UniDAQ2 mit reinen AD- bzw. DA-Wandlern realisiert wurde. Die weiteren Spezifikationen zur Abtastrate, Antialias Filterung und der Kopplung ergeben sich aus dem verwendeten Audio-Interface. Da der Audio-Codec des OMAP-LCDK für die Aufnahme und Wiedergabe von Audiosignalen konzipiert ist, können die gängigen Abtastraten von 44,1 kHz, 48 kHz und 96 kHz eingestellt werden. Weil im Audiobereich kein Gleichspan-

nungsanteil im Signal vorhanden sein sollte, wird dieser intern durch die Vorschaltung eines Koppelkondensators abgeblockt. Das UniDAQ2 ist hier flexibler in der Wahl der Abtastrate, da die Wandler über einen externen Trigger angesteuert werden. Dieser erlaubt es, Signale mit Abtastraten bis zu 175 kHz zu erfassen. Ebenfalls ist kein Koppelkondensator vorhanden, so dass es möglich ist, Gleichspannungsanteile zu erfassen. Die Antialias-Filterung ist ebenfalls, bedingt durch das verwendete Analog-Interface, sehr unterschiedlich realisiert. Der im OMAP-LCDK verwendete Audio-Codec nutzt hier ein Oversampling-Verfahren, bei dem mit einer wesentlich höheren Abtastrate als eingestellt, abgetastet wird und anschließend eine digitale Filterung und Dezimation erfolgt um bei der gewünschten Abtastrate ein korrektes Spektrum zu erhalten.
Das UniDAQ2 verwendet für die Antialias-Filterung einen analogen Tiefpass zweiter Ordnung, dessen Grenzfrequenz je nach angelegter Versorgungsspannung bei 15 kHz oder 23 kHz liegt.

4.1.4. Talk-Through Messung der EVMs

Um die Eigenschaften beider EVMs genauer zu untersuchen, wurde eine sogenannte Talk-Through Messung mit dem UPV durchgeführt. Dazu wurde zunächst eine Software im DSP beider EVMs implementiert, die das digitale Signal eines Eingangs direkt auf einem Ausgang wieder ausgibt. Die Abtastrate wurde hierbei bei beiden EVMs größtmöglich gewählt. Anschließend wurde mit dem UPV eine Messung von Frequenz- und Phasengang durchgeführt um die wesentlichen Eigenschaften des Signalpfades mit minimaler Latenz zu bestimmen. Die in der Spezifikation angegebenen Filter lassen sich so überprüfen und weitere Effekte können sichtbar gemacht werden.
Der für die Messung notwendige Aufbau und die notwendigen Einstellungen im UPV sind in Abbildung 4.5 zu sehen.

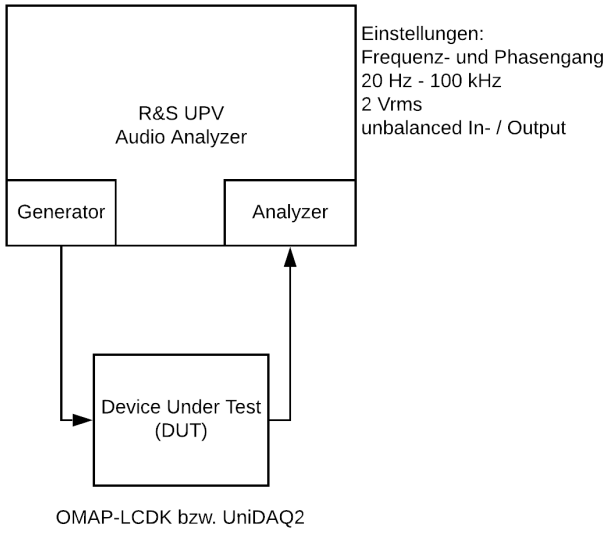


Abbildung 4.5.: Messaufbau zur Talk-Through Messung

Mit diesem Aufbau wurde für beide EVMs eine Messung durchgeführt. Diese sind in Abbildung 4.6 dargestellt, wobei links der Amplitudengang des OMAP-LCDK und rechts der des UniDAQ2 aufgezeichnet ist.
Zu erkennen ist, dass das OMAP-LCDK, wie erwartet eine DC-Entkopplung besitzt. Dies macht sich durch das Hochpassverhalten bis ca. 300 Hz bemerkbar. Anschließend verläuft der Amplitudengang sehr linear, bis er ab der Grenzfrequenz von 48 kHz stark gedämpft wird.

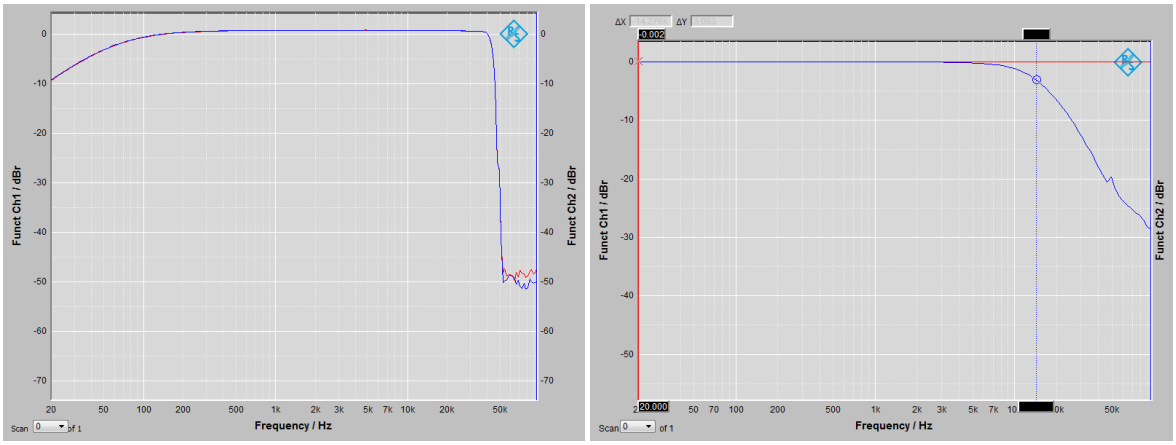


Abbildung 4.6.: Amplitudengang der Talk-Through Messung des OMAP-LCDK (links) und des UniDAQ2 (rechts)

Das UniDAQ2 hat wie erwartet, beginnend bei 20 Hz, einen linearen Amplitudengang, da kein Hochpass am Eingang vorhanden ist. Das nachgeschaltete Antialias-Filter mit einer Grenzfrequenz von 15 kHz lässt sich im Amplitudengang gut erkennen.

4.1.5. Fazit

Der theoretische Vergleich zeigt bereits, dass beide EVMs sich stark voneinander unterscheiden. Das OMAP-LCDK bietet mit seinen zwei Prozessorkernen eine flexiblere und leistungsfähigere Plattform an. Allerdings ist die Programmierung des ARM-Kern mit dem verfügbaren Linux nicht sehr ausgereift und schlecht dokumentiert, so dass aufgrund der fehlenden Erfahrung in diesem Bereich und dem schwer abzuschätzenden Mehraufwand dieses EVM nicht optimal für die Realisierung eines Audio-Analyzers ist. Der verwendete Audio-Codec erlaubt ebenfalls nur Abtastraten bis 96 kHz, so dass eine maximale Bandbreite von 48 kHz vorgegeben ist. Die vorhandene DC-Entkopplung bildet ebenfalls ein weiteres Element im Signalpfad, welches nicht an die Bedürfnisse angepasst werden kann. Weiterhin begrenzt die Anzahl der Ein- und Ausgänge, von denen jeweils nur zwei vorhanden sind, die Möglichkeiten der Signalerfassung.
Die Messung des Audiosignalpfades hat zusätzlich gezeigt, dass die digitale Filterung des Audio-Codec zwar eine gute Antialias-Filterung ergibt, aber dadurch das Signal im inneren des Audio-Codec in einer „Blackbox“ manipuliert wird. Dies ist für eine Wiedergabe oder Aufnahme von Audiosignalen nicht weiter hinderlich, aber für die Verwendung als Audio-Analyzer nicht

geeignet, da hier möglichst alle Elemente des Signalpfades bekannt sein müssen um eine korrekte Messung von Amplitude und Phase von Ein- und Ausgangssignal durchführen zu können.

Das UniDAQ2 ist, im Gegensatz zum OMAP-LCDK nicht so leistungsstark, da es nur einen DSP-Kern besitzt. Dennoch könnte, bei entsprechender Programmierung, dieser ausreichend sein, um einen Audio-Analyzer zu realisieren. Das Beispielprogramm hat hier gezeigt, dass ein Webserver und eine einfache digitale Signalverarbeitung auf einem Kern implementiert werden können. Die von d.signt mitgelieferten Softwarepakete sind besser dokumentiert und ermöglichen somit einen schnelleren Einstieg in die Programmierung. Die Talk-Through Messung des UniDAQ2 hat gezeigt, dass der Signalpfad einen linearen Amplitudengang von ca. 20 Hz bis 5 kHz aufweist. Anschließend macht sich das Antialias-Filter bemerkbar, welches das erwartete Verhalten eines Tiefpass 2. Ordnung hat. Mit einer maximalen Abtastrate von 175 kHz wären theoretisch Bandbreiten bis ca. 87,5 kHz möglich, jedoch nur unter Anpassung des auf dem Board befindlichen Tiefpass.

Der Vergleich von OMAP-LCDK und UniDAQ2 und die Bewertung im Hinblick auf die Verwendung in dieser Arbeit, haben gezeigt, dass beide EVMs nicht gut geeignet sind. Es erfolgt deshalb eine weitere Recherche nach geeigneten EVMs, die die gewünschten Anforderungen besser erfüllen und eine gute Basis für die Entwicklung eines Audio-Analyzers bieten. Der durchgeführte Vergleich der verfügbaren EVMs ist hier hilfreich, da Erfahrungen gesammelt wurden, die bei der Auswahl einer Hardware berücksichtigt werden können und auf die bei der Spezifikationen besonders geachtet werden muss.

4.2. Untersuchung neuer Hardware

4.2.1. Anforderungen

Um eine Hardware zu finden, welche die Anforderungen an die Aufgabenstellung in dieser Arbeit gut erfüllen, sind die notwendigen Spezifikationen kurz zusammengefasst:

- A/D- und D/A-Wandler mit möglichst hoher Abtastrate (> 200 kHz)
- keine, bzw. anpassbare Antialias-Filter
- Schnittstellen für die Benutzerkommunikation (Ethernet, Display)
- leistungsstarker DSP (nach Möglichkeit mehr als ein Kern)
- verfügbare Softwarepakete und Support

4.2.2. D.Module2

Anhand der in 4.2.1 genannten Spezifikation wurde ein passendes EVM gesucht. Da an der Hochschule für angewandte Wissenschaften Hamburg (HAW) sowie der Fachhochschule Westküste (FHW) bereits gute Erfahrungen mit EVMs der Firma d.signt gemacht wurden und sich auch bei der Untersuchung des UniDAQ2 gezeigt hat, dass der Support und auch die Dokumentation der Software sehr gut ist, wurde dort zuerst eine Anfrage gestellt.
Mit Berücksichtigung der gestellten Spezifikation wurden zwei Angebote von d.signt zugeschickt. Die Basis beider Angebote bildet das D.Module2-Konzept von d.signt, welches eine Basisplatine beinhaltet, auf die passende Module übereinander zu einem Gesamtsystem zusammengesetzt werden können. Die Basisplatine hat folgende Spezifikation, die dem Datenblatt [4] entnommen wurden:

- 100x100mm Basisplatine für DSP- und Peripherieboards
- 3,3V Spannungsversorgung
- +-5V Erzeugung für Peripherie
- Ethernet, RS232, USB und SD-Kartensteckplatz
- I2C, SPI und GPIO Pmod Adapter

Für die analoge Signalerfassung und -ausgabe ist in beiden Angeboten das D.Module2.ADDA500K16 enthalten. Das Datenblatt [5] liefert die folgenden Merkmale:

- 6 analoge Eingänge / 3 differentielle mit 16-bit Auflösung
- simultane Abtastung bis 500kS/s
- 2 analoge Ausgänge mit 16-bit Auflösung
- +/- 3V Eingangs- und +/- 2,5V Ausgangsspannungsbereich

Der Unterschied zwischen beiden Angeboten liegt bei dem verwendeten Hauptprozessor. In beiden Angeboten werden DSPs von TI verwendet. Diese sind bereits aus den Untersuchungen der verfügbaren EVMs bekannt. Im ersten Angebot kommt ein Single-Core DSP vom Typ C6747 zum Einsatz [6], im zweiten Angebot ein Dual-Core DSP vom Typ C6657 [7]. Die Eigenschaften der beiden Prozessoren sind in Tabelle 4.2 nebeneinander aufgelistet.

Tabelle 4.2.: Spezifikation des C6747 und des C6657 DSP

	C6747	C6657
Prozessor	Single-Core 1x 456 MHz (DSP)	Dual-Core 2x 1,25 GHz (DSP)
Speicher (intern)	32 kB Data Cache 32 kB Program Cache 128 kB RAM	32 kB Data Cache (pro Kern) 32 kB Program Cache (pro Kern) 1 MB RAM
Speicher (extern)	64 MB SDRAM	500 MB DDR3 RAM
Ethernet	100 Mbit	1 Gbit
Schnittstellen	2x USB, 2x UART, 1x SPI, 1, I2C, 2x McASP	1x USB, 3x UART, 1x PCIe, 1x SRIO, 1x SPI, 1x I2C, 2x McASP
Stromaufnahme	750 mA typ. (3,3V)	2A typ., 3A peak (3,3V)
Firmware	D.Module2.BIOS	D.Module2.BIOS

Wie zu erkennen ist, unterscheiden sich die DSPs hauptsächlich in der Art des verwendeten Prozessors. Während im C6747 ein Kern mit 456 MHz zum Einsatz kommt, befinden sich im C6657 zwei Kerne mit je 1,25 GHz. Weiterhin bietet der C6657 500 MB externen RAM und Gigabit Ethernet, während im C6747 nur 64 MB RAM und eine 100 MBit Ethernet-Schnittstelle untergebracht sind. Die Stromaufnahme ist bei dem C6657 mit 2 Ampere deutlich höher als beim C6747 mit 750 Milliampere. Dies resultiert hauptsächlich aus der höheren

Prozessorgeschwindigkeit und der leistungsfähigeren Peripherie.
Als Firmware kommt bei beiden Prozessoren das von d.signt mitgelieferte D.Module2.BIOS zum Einsatz, das einen Bootloader, eine Boardinitialisierung und -konfiguration und eine Möglichkeit zum Flashen der Software beinhaltet. Die Software kann so über USB oder die UART-Schnittstelle von einem Host-PC übertragen werden.

4.2.3. Auswahl

Aufgrund der bereits gesammelten Erfahrung mit dem UniDAQ2 EVM von d.signt wurde die Auswahl auf die zwei Angebote, die d.signt vorgelegt hat, begrenzt. Die mitgelieferten Softwarepakete des UniDAQ2 und deren gute Dokumentation, sowie die einfache Implementierung eigener Software lassen darauf schließen, dass bei der Verwendung des D.Module2 keine unerwarteten Schwierigkeiten in der Softwareentwicklung auftreten. Das in beiden Angeboten vorhandene D.Module2.ADDA500K16 erfüllt die in Kapitel 4.2.1 gestellten Anforderungen. Die Abtastrate ist ausreichend groß und es sind keine analogen oder digitalen Filter implementiert, die eine Nutzung von vornherein einschränken. Die notwendigen analogen Schaltungen können so den Anforderungen entsprechend entworfen werden.

Anhand der Aufgabenstellung dieser Arbeit und den in Kapitel 4.2.1 festgelegten Anforderungen, wurde sich für die Variante mit dem C6657 DSP entschieden. Dieser bietet mit seinen zwei Kernen, dem großen Speicher und den vielfältigen Schnittstellen eine gute Basis, um einen Audio-Analyzer zu realisieren. Die ausgewählten Module des D.Module2-Konzepts sind in Abbildung 4.7 zu sehen.

Abbildung 4.7.: ausgewählte Hardwaremodule von d.signt (Quelle: d.signt)

5. Hardwareentwicklung

5.1. Planung

Die Hardwareentwicklung beinhaltet die Entwicklung von analogen Schaltungen, um das ausgewählte EVM für den Einsatz als Audio-Analyzer zu erweitern. Des Weiteren muss ein Gehäusekonzept entworfen werden, um die verschiedenen Komponenten sinnvoll unterzubringen. Daher wurde zuerst eine Anforderungsliste an die Hardware zusammen gestellt, anhand derer die Entwicklung erfolgt. Als Vorbild dient hier der UPV, anhand dessen Merkmalen und Schaltbildern die Anforderung erstellt wird.

5.1.1. Gehäuse

Die Anforderungen an das Gehäuse ergeben sich aus der Überlegung, ein fertiges Messgerät zu erhalten, welches in Laborversuchen eingesetzt und in möglichen weiterführenden Arbeiten als Basis genutzt werden kann.
Sie ergeben sich somit folgendermaßen:

- geschlossenes Gehäuse für die Verwendung des Audio-Analyzers im Labor
- einfacher Austausch / Erweiterung der einzelnen Komponenten bzw. Platinen
- standardisierte Anschlussmöglichkeiten der Testgeräte
- Anschlussmöglichkeiten für Spannungsversorgung und Schnittstellen (Ethernet, USB, Display)

5.1.2. analoge Schaltungen

Die Anforderungen an die analogen Schaltungen ergeben sich aus den Ein- und Ausgangsschaltungen des UPV [8], der hier als Vorbild dient. Daraus und mit Berücksichtigung der verwendeten Hardware, ist die in Anhang A enthaltene Spezifikation erweitert worden.

Die Spezifikation beruht, wie schon erwähnt, auf dem Vorbild des UPV und unter Berücksichtigung der verwendeten Hardware. Das verwendete Analog-Interface D.Module2.ADDA500K16 bietet insgesamt 3 symmetrische Eingangspaare, so dass zwei davon für den Analyzer genutzt werden können. Das dritte Eingangspaar wird, wie in Kapitel 5.2.2.3 näher erläutert, für die Messung des Generatorausgangs genutzt. Der definierte Eingangsspannungsbereich des Analyzers ist vom Analog-Interface vorgegeben. Aus dem Datenblatt des Analog-Interfaces [9] geht hervor, dass der Eingangsspannungsbereich des ADC von +-1V bis +-3V konfiguriert werden kann.
Ein zweiter höherer Spannungsbereich soll durch einen Spannnungsteiler ermöglicht werden. Die Bandbreite wurde hier auf maximal 100 kHz festgelegt, da bis zu dieser Frequenz ausreichend gute Messerergebnisse erwartet werden.
Die Eingangssignale können entweder direkt eingespeist, oder über einen Koppelkondensator von ihrem Gleichspannungsanteil bereinigt werden. Die Eingangsimpedanz von 100 kΩ, sowie die Möglichkeit der DC-Entkopplung sind dem UPV nachempfunden.

Die Generatorspezifikation ergibt sich ebenfalls größtenteils aus den Möglichkeiten des D.Module2.ADDA500K16. Dieses besitzt zwei analoge Ausgänge, so dass dementsprechend zwei unsymmetrische Ausgänge realisiert werden können. Der Ausgangsspannungbereich ist, wie bei den Eingängen fest vorgegeben und beträgt laut Datenblatt +-2,5V. Als Ausgangsimpedanz wurden 50 Ω gewählt. Der Ausgangswiderstand des DAC auf dem D.Module2.ADDA500K16 beträgt ebenfalls 50 Ω, so dass eine Leistungsanpassung erzielt wird. Die Bandbreite beträgt, passend zum Eingang, 100 kHz und muss analog begrenzt werden.

5.2. Entwicklung

5.2.1. Gehäuse

Um die Anforderungen in Kapitel 5.1.1 zu erfüllen, wurde sich für die Verwendung eines 19-Zoll Gehäuses entschieden. Für diese standardisierte Gehäuseform gibt es von unzähligen Herstellern verschiedene Systeme, die für die Verwendung auf einem Labortisch oder den Einbau in einen Systemschrank vorgesehen sind. Hier wurde sich für ein Tischgehäuse mit einer Standardhöhe von 3 Höheneinheiten (HEs) entschieden, so dass Platinen im Europakartenformat (100 x 160 mm) verbaut werden können. Diese Platinen sind über einen rückwärtigen Bus miteinander verbunden und können jederzeit ausgetauscht werden. An der Gehäusefront können Frontplatten montiert werden, die in Verbindung mit einer Platine einen Einschub bilden. So besteht die Möglichkeit, einzelne Platinen für die analogen Schaltungen zu entwerfen, die später weiterentwickelt oder ausgetauscht werden können.
Auf der Rückseite des Gehäuses können ebenfalls einzelne Frontplatten eingebaut werden, um Anschlussmöglichkeiten für die Spannungsversorgung und die notwendigen Schnittstellen bereitzustellen. Abbildung 5.1 zeigt das entstandene Konzept.

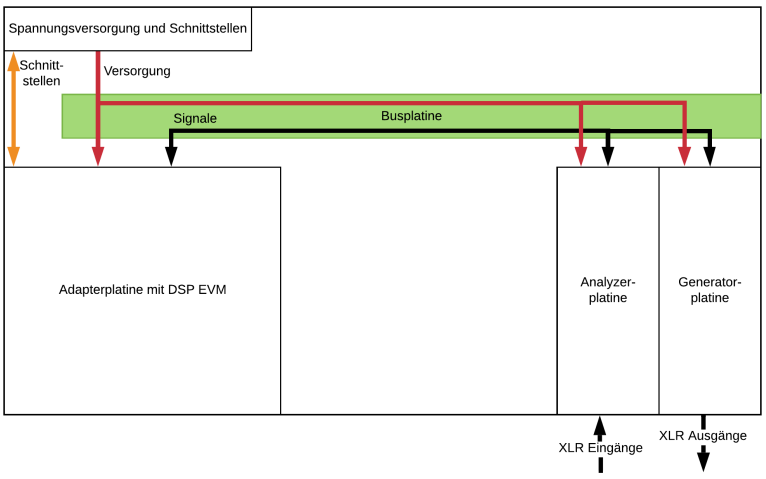


Abbildung 5.1.: Gehäusekonzept 19-Zoll (Ansicht von oben)

Es sieht vor, mehrere Einschübe und Platinen zu entwickeln, um eine spätere Weiterentwicklung einzelner Komponenten zu ermöglichen. Auch wurde in dem Konzept Platz für weitere Einschübe gelassen, so dass der Audio-Analyzer auch nach Abschluss dieser Arbeit um

Hardware ergänzt werden kann.
Das EVM wird über eine Adapterplatine mit der rückwärtigen Busplatine verbunden, so dass über diesen die analogen Signale sowie digitale Aus- und Eingänge an allen Platinen anliegen. Die Spannungsversorgung erfolgt über die Gehäuserückseite, an der auch Anschlüsse für die Schnittstellen des EVM eingebaut werden sollen.

Für die analogen Ein- und Ausgänge des Analyzers bzw. des Generators sind XLR-Steckverbindungen, wie sie auch im UPV Verwendung finden, vorgesehen. Mit diesen ist es möglich, symmetrische oder unsymmetrische Signale über einen Stecker zu übertragen.

An der Frontplatte des DSP-EVM Einschubs sind zusätzlich LEDs vorgesehen, um verschiedene Betriebszustände anzuzeigen.

5.2.2. analoge Schaltungen

Die analogen Schaltungsteile wurden so entwickelt, dass die beschriebenen Anforderungen erfüllt werden. Die Anforderung an das Gehäuse muss hier berücksichtigt werden, da für dieses, Platinen im Europakartenformat, entwickelt werden müssen. Es wurde sich entschieden, drei Platinen zu entwerfen:

- eine Adapterplatine, um die Signale des EVM auf die Busplatine zu führen
- eine Analyzerplatine, welche die notwendigen Schaltungen der analogen Eingänge beinhaltet
- eine Generatorplatine, welche die notwendigen Schaltungen der analogen Ausgänge beinhaltet

Jede Platine hat das erforderliche Europakartenformat von 100 x 160 Millimeter und besitzt an der Rückseite einen Busplatinenstecker. Außerdem wird für jede Platine eine Frontplatte entworfen und gefertigt, die die notwendigen Anschlüsse, Taster und Anzeigeeinrichtungen beinhaltet.

Um die Platinen zu entwickeln, musste zuerst die Pinbelegung des rückseitigen Busses festgelegt werden. Der Bus verfügt über 64 Signalleitungen und wird über einen standardisierten Steckverbinder (DIN41612) angebunden. Dieser verfügt über zwei Reihen(A und C) mit jeweils 32 Signalleitungen. Die Pinbelegung ist in Anhang B zu finden. Dabei wurde versucht, möglichst viele Signale des EVM auf der Busplatine zur Verfügung zu stellen. Dennoch sind genug freie Signalleitungen vorhanden, um spätere Erweiterungen zu ermöglichen.
Für die Schaltplan- und Layouterstellung wurde die Software Cadence OrCAD genutzt. Es wurde sich für ein zweiseitiges Platinenlayout entschieden, da diese an der FHW mit einem Fräsbohrplotter gefertigt werden können und ausreichend Platz für die Schaltungsentwicklung bieten. Die Projektverzeichnisse aller gefertigten Platinen, inklusive der für die Fertigung notwendigen Bohr- und Fräsdaten sowie die Schaltpläne sind auf der CD enthalten. Die Schaltpläne sind zusätzlich in Anhang D zu finden.

5.2.2.1. Adapterplatine

Die Hauptaufgabe der Adapterplatine ist es, die Signale des EVM auf den Bus zu führen. Des Weiteren soll die Spannungsversorgung, welche rückseitig in das Gehäuse geführt wird,

ebenfalls auf dem Bus zur Verfügung gestellt werden. An der Frontplatte der Adapterplatine sollen LEDs den Betriebszustand des Audio-Analyzers anzeigen.

Das EVM verfügt über mehrere zweireihige Stift- und Buchsenleisten, an denen die analogen Signale der Ein- und Ausgänge sowie General Purpose Input-Outputs (GPIOs) und Schnittstellen herausgeführt sind. Eine Pinbelegung dieser Steckverbindungen, inklusive des geplanten Verwendungszwecks, ist in Anhang C zu finden. In Abbildung 5.2 ist der gefertigte Einschub, bestehend aus Adapterplatine und Frontplatte gezeigt. Die Verbindungsleitungen zwischen

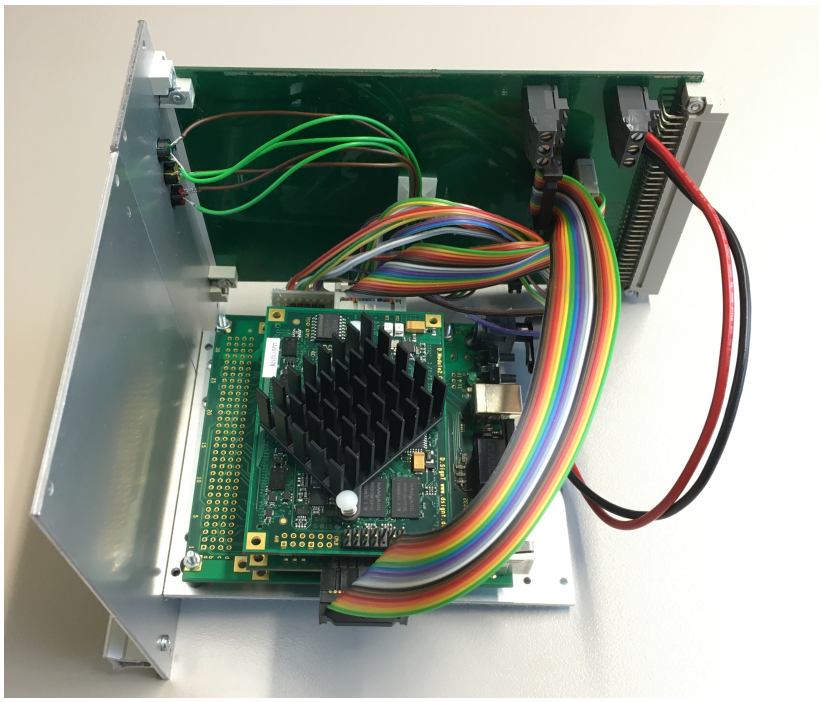


Abbildung 5.2.: gefertige Adapterplatine mit Frontplatte

dem EVM und der Adapterplatine sind als Flachbandkabel ausgeführt. Der Anschluss der Leitungen für die Spannungsversorgung, welche von der Rückseite des Gehäuses kommen, erfolgt über Schraubanschlüsse und wird dann zum einen von der Platine auf den Bus geführt und zum anderen zum EVM. Wie zu erkennen ist, besteht der Einschub neben der Adapterplatine auch aus einer Halterung, auf der sich das EVM befindet. So ist es möglich, den Einschub aus dem Gehäuse zu nehmen, ohne die Verbindungsleitungen zwischen Adapterplatine und EVM zu trennen. Dies ist nützlich, wenn zwischen dem Einschub und der Busplatine eine Verlängerungsplatine gesteckt wird um Messungen durchzuführen, da sich die Platine so außerhalb des Gehäuses befindet.

5.2.2.2. Analyzerplatine

Die Analyzerplatine beinhaltet alle Schaltungsteile, die notwendig sind, um die Eingänge des Audio-Analyzers entsprechend der Spezifikation zu verwenden. Als Vorlage dienen hier die im Bedienhandbuch des UPV [10] enthaltenen Schaltpläne.
In Abbildung 5.3 ist die DC-Entkopplung und die Umschaltung der Eingangsimpedanz, wie sie im UPV realisiert ist, zu sehen. Über die Pins 2 und 3 der XLR-Buchse, kann ein sym-

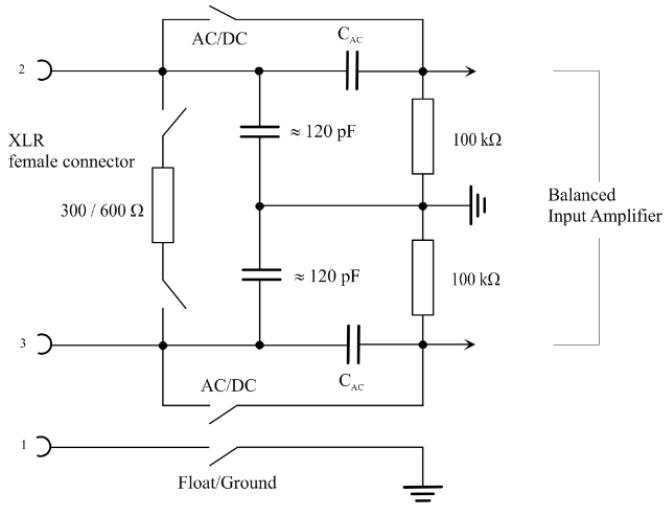


Abbildung 5.3.: Eingangsbeschaltung des UPV (Quelle: Rhode und Schwarz)

metrisches Signal eingespeist werden. Der Entkopplungskondensator C_{AC} kann in beiden Signalpfaden überbrückt werden, um auch den DC-Anteil des Signals in den Eingang einzukoppeln. Ein Widerstand von 100 kΩ gegen Signalmasse bildet die Eingangsimpedanz. Zusätzlich kann bei einem symmetrischen Eingangssignal der Eingangswiderstand auf 300 oder 600 Ω gesetzt werden, indem dieser zwischen die Signalleitungen geschaltet wird. Die parasitäre Koppelkapazität zwischen den Signalleitungen und der Masse ist hier mit 120 pF angegeben. Pin 1 der XLR-Buchse kann wahlweise mit der Signalmasse verbunden werden, oder offen bleiben.

Um die Anforderung an die Analyzerplatine zu erfüllen, wurde die in Abbildung 5.4 dargestellte Schaltung für jeden Eingangskanal entworfen. Die XLR-Pins 2 und 3 sind wie beim UPV für die Eingangssignale vorgesehen. Pin 1 ist dauerhaft auf die Signalmasse geschaltet. Es wurde hier auf eine Option, Pin 1 offen zu lassen, verzichtet. Die Schalter S1 und S2, welche durch Reed-Relais realisiert werden, ermöglichen die Umschaltung zwischen AC- und

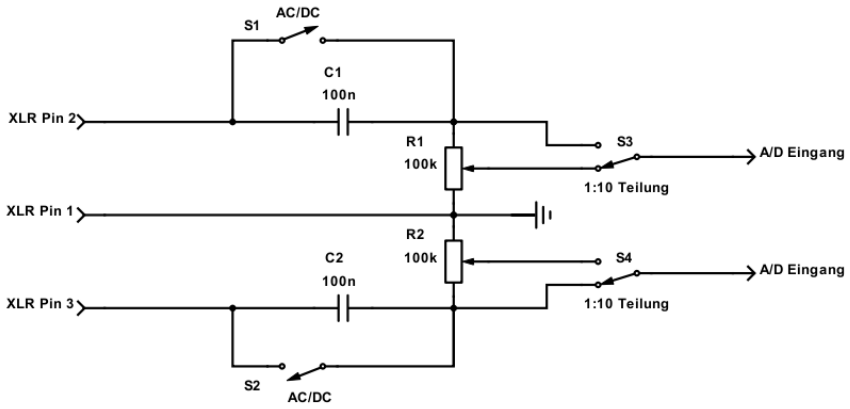


Abbildung 5.4.: Eigene Eingangsschaltung

DC-Kopplung. Sind die Schalter offen, bildet C1 mit R1, bzw. C2 mit R2 einen Hochpass, der den Gleichspannungsanteil des Signals entfernt. Die Werte für C1 und C2 wurden so gewählt, dass mit der vorgegebenen Eingangsimpedanz von 100 kΩ nach der Formel 4.1 die Grenzfrequenz des Hochpasses $f_g = 15,91Hz$ ist.
Ist S1 bzw. S2 geschlossen, ist der Kondensator überbrückt und der Gleichspannungsanteil wird nicht herausgefiltert.

Die Widerstände R1 und R2 sind als Potentiometer ausgelegt, um einen einstellbaren Spannungsteiler zu realisieren. Mit den Schaltern S3 und S4 kann ausgewählt werden, ob das Signal am oberen Punkt des Potentiometers, oder am Schleifer abgegriffen werden soll. Somit lassen sich, wie geplant, zusätzlich zum Eingangsspannungsbereich des ADC ein weiterer, höherer Bereich realisieren. Vorteil dieser Art der Schaltung ist, dass die Eingangsimpedanz konstant bei 100 kΩ bleibt, da der nachfolgende Eingang des A/D-Wandlers hochohmig (> 1MΩ) ist. Die Umschaltung auf eine Eingangsimpedanz von 300 oder 600 Ω, bei symmetrischem Betrieb, wurde nicht umgesetzt, da diese bei Bedarf extern erfolgen kann und so die, für die Ansteuerung der Reed-Relais, notwendigen GPIOs reduziert werden konnten.
Für die Ansteuerung der Reed-Relais, welche eine Betriebsspannung von 5V haben, kommt ein Treiberbaustein vom Typ ULN2003LVDR zum Einsatz. Dieser hat laut dem Datenblatt [11] bereits Freilaufdioden integriert, so dass diese nicht extra mit im Design vorgesehen werden mussten. Der Treiber ermöglicht es, die Reed-Relais mit den 3,3V Pegeln des EVM anzusteuern.
Die beschriebene Schaltung ist auf der Platine in zweifacher Ausführung, jeweils für Kanal

A und B, vorhanden. Der gefertigte Einschub, bestehend aus Platine und Frontplatte ist in Abbildung 5.5 zu sehen.

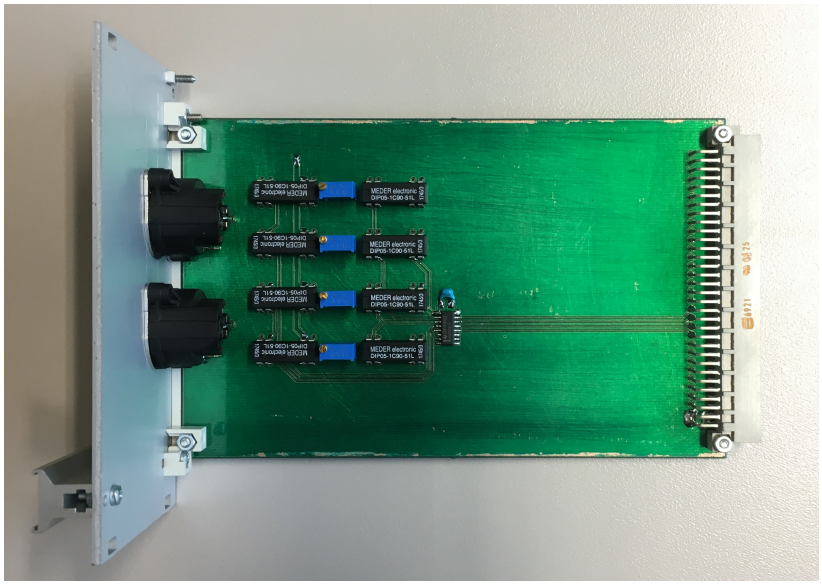


Abbildung 5.5.: gefertigte Analyzerplatine

5.2.2.3. Generatorplatine

Die Generatorplatine beinhaltet alle Schaltungsteile, die notwendig sind, um die D/A-Ausgänge des EVM entsprechend der Spezifikation zu verwenden. Auch hier dient die Schaltung des UPV als Vorlage für die eigene Entwicklung.
Im Bedienhandbuch des UPV [10] ist beschrieben, dass die beiden analogen Ausgänge, die als XLR-Stecker ausgeführt sind, eine Ausgangsimpedanz von 5Ω besitzen und symmetrisch ausgeführt sind. Es können damit Ausgangssignale mit bis zu 20V erzeugt werden. Weiterhin gibt es die Möglichkeit, die Ausgänge abzuschalten, bzw. vom Generator zu trennen, sollte eine Störung während der Messung auftreten. Somit können die angeschlossenen Geräte vor einem Schaden bewahrt werden.
Um eine vergleichbare Schaltung mit ähnlichem Funktionsumfang zu erhalten, wurde die in Abbildung 5.6 zu sehende Schaltung entwickelt.

Da das ADDA500K16 nur zwei DA-Wandler besitzt, konnten entweder zwei unsymmetrische Ausgänge oder ein symmetrischer Ausgang realisiert werden. Es wurde sich entschieden, zwei Ausgänge, die jeweils unsymmetrisch sind, zu realisieren. Ein einzelner symmetrischer

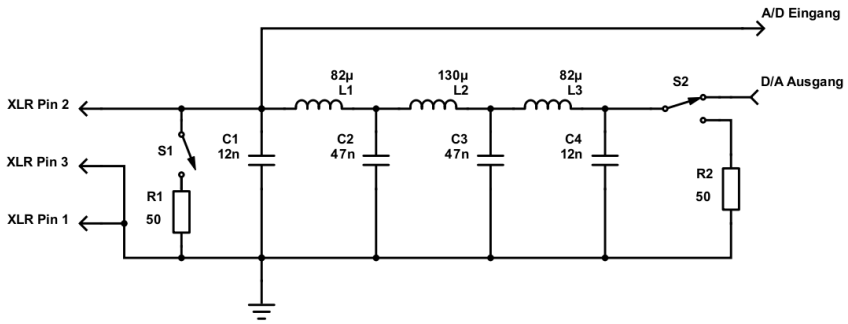


Abbildung 5.6.: Eigene Ausgangsschaltung

Ausgang kann bei Bedarf durch externe Verschaltung der beiden einzelnen Ausgänge erfolgen. Die Software muss an den beiden Ausgängen dann ein symmetrisches Signal erzeugen. Deshalb sind die XLR-Pins 3 und 1 jeweils auf die Signalmasse gelegt. XLR Pin 2 bildet den Signalausgang.
Um das Ausgangssignal des D/A-Wandlers zu glätten, wurde ein analoges Tiefpassfilter 7. Ordnung realisiert. Die Berechnung des Filters ist mit der Software „ELSIE“ durchgeführt worden. Dies ist ein Tool um analoge Filter nach vorgegebenen Parametern zu realisieren.
Als Parameter wurde hier die Grenzfrequenz $f_g = 120kHz$ gewählt. Das Ergebnis der Filterberechnung mit den Werten für Kapazitäten und Induktivitäten ist in Abbildung 5.7 zu sehen.

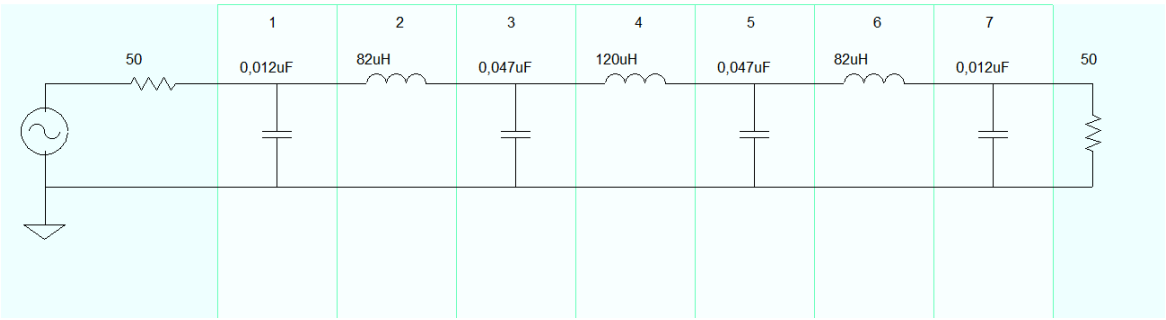


Abbildung 5.7.: Filterschaltung in ELSIE

Der Amplituden- und Phasengang der Simulation des Filters ist in Abbildung 5.8 dargestellt. Der Verlauf der Amplitude ist bis zu einer Frequenz von 10 kHz linear bei -6dB. Dies liegt an der festgelegten Quell- und Abschlussimpedanz. Diese wurde so gewählt, da im

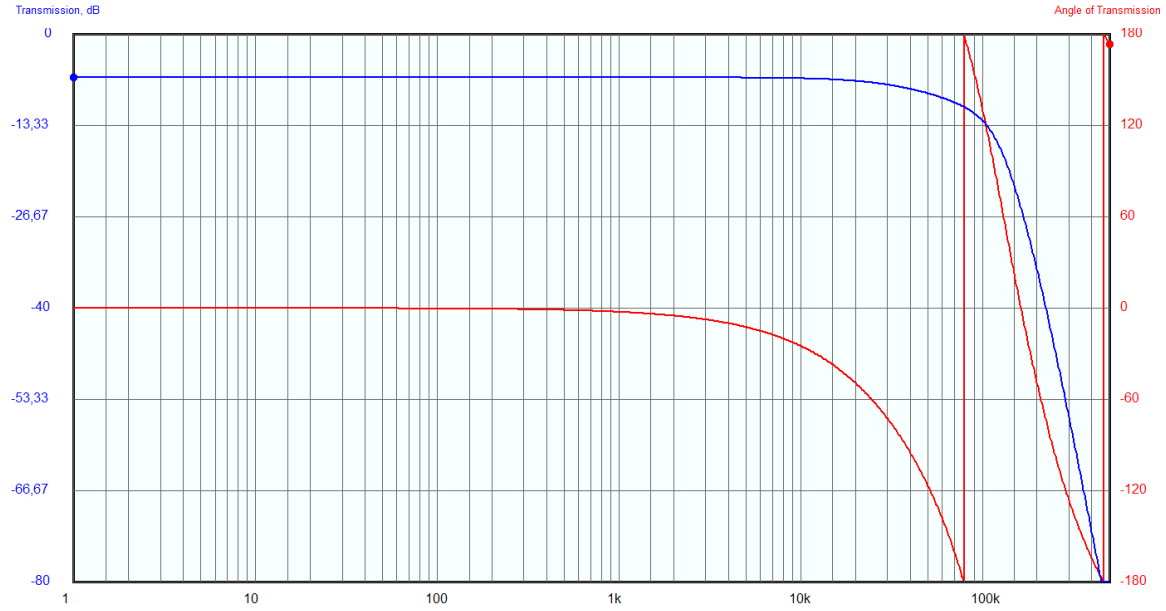


Abbildung 5.8.: Amplituden- und Phasengang des Filters in ELSIE

D.Module2.ADDA500K16 am Ausgang des D/A-Wandlers pro Ausgang ein 50Ω Widerstand in Reihe geschaltet ist, um den Stromfluss bei kapazitiven Lasten zu begrenzen. Die Abschlussimpedanz ist dementsprechend ebenfalls auf 50Ω festgelegt worden. Somit wird eine Leistungsanpassung am Ausgang der Filterschaltung erzielt.
Oberhalb von 10 kHz dämpft das Filter langsam, bis es bei ca. 120 kHz seine festgelegte Grenzfrequenz erreicht. Wie zu erkennen ist, verläuft die Phase bis 1 kHz linear bei 0°. Im Dämpfungsbereich des Filters dreht sie sich bis 140 kHz um 360°.
Um die Ausgangssignale in der Software zu erfassen und für die Signalverarbeitung zu nutzen, wurde vom dritten Eingangspaar des ADCjeweils ein Eingang auf den Filterausgang gelegt.

Da das analoge Ausgangsfilter mit einer Abschlussimpedanz von 50 Ω berechnet wurde, muss es dementsprechend abgeschlossen sein. Der Schalter S1, der wie bei der Analyzerplatine durch ein Reed-Relais realisiert wurde, schaltet bei Bedarf einen Abschlusswiderstand hinzu. Dies ist notwendig, wenn die Eingangsimpedanz des zu DUT hochohmig ist. Beträgt die Eingangsimpedanz des Gerätes 50Ω, kann der Schalter S1 offen gelassen werden. Die Zuschaltung des 50Ω Abschlusswiderstands erfolgt mit einem Schalter an der Frontplatte des Einschubs. Eine LED signalisiert, dass der Widerstand zugeschaltet ist. Eine Option, diesen Widerstand aus der Software heraus zu schalten, ist ebenfalls integriert.
Um wie beim UPV eine Ausgangsabschaltung zu realisieren, ist der Schalter S2, ebenfalls

als Reed-Relais ausgelegt, in der Schaltung vorhanden. Ist der Signalausgang aktiv, liegt das Ausgangssignal des D/A-Wandlers am Eingang des analogen Tiefpassfilters. Soll der Ausgang abgeschaltet werden, wird der Eingang des Tiefpasses über einen 50Ω Widerstand auf die Signalmasse gelegt. Somit wird erreicht, dass nach Abschalten das Ausgangssignal an XLR Pin 2 weiterhin definiert ist. Die Abschaltung der Ausgänge wird durch jeweils eine LED pro Ausgang angezeigt.
Die Ansteuerung der Reed-Relais erfolgt, wie auch bei der Analyzerplatine, über einen Treiberbaustein vom Typ ULN2003LVDR.
Die fertig hergestellte Platine mit Frontplatte ist in Abbildung 5.9 dargestellt.

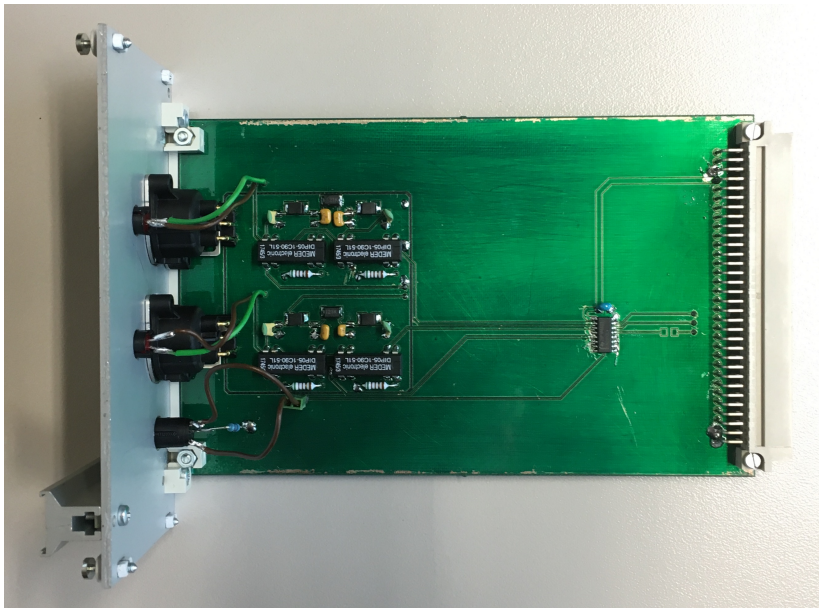


Abbildung 5.9.: gefertigte Generatorplatine

5.3. Zusammenbau der Hardware

Nachdem das Gehäuse beschafft und die einzelnen Platinen entwickelt wurden, konnte die Hardware zusammengebaut werden. Für die Platinen wurden jeweils Frontplatten angefertigt und mit Beschriftungen versehen. In Abbildung 5.10 ist die fertige Hardware zu sehen.

Abbildung 5.10.: zusammengebauter Audio-Analyzer

Das Gehäuse bietet wie geplant, Platz für die drei entwickelten Platinen und es besteht zusätzlich noch die Möglichkeit, weitere Platinen in die freien Steckplätze einzubauen. Diese sind mit Blindplatten verkleidet, um eine abgeschlossene Front zu erhalten. Die Positionierung der Platinen kann frei gewählt werden, so dass beispielsweise die Analyzer- und die Generatorplatine gegeneinander getauscht werden können und weiterhin ihre Funktion behalten.
An der Rückseite des Gehäuses wurden zum Anschluss der Spannungsversorgung Hohlsteckerbuchsen verbaut, über die zwei Steckernetzteile den Audio-Analyzer mit 5V und 3,3V versorgen. Des Weiteren ist Platz für den Anschluss weiterer Schnittstellen vorhanden.

5.4. Test der entwickelten Hardware

Für den Test der entwickelten Hardware wurden verschiedene Messungen durchgeführt, die die geforderte Funktion der einzelnen Platinen bestätigt. Dazu wurden zuerst die Platinen auf Kurzschlüsse und Schaltungsfehler untersucht. Anschließend wurden die Relais auf ihre Funktion überprüft. Nachdem diese Prüfungen erfolgreich waren, wurden die analogen Signalwege vermessen. Hierfür wurden Messungen der Analyzer- sowie die Generatorplatine mit dem UPV durchgeführt. Zusätzlich wurde eine Talk-Through Messung des Gesamtsystems durchgeführt.

5.4.1. Messung der Analyzerplatine

Die Analyzerplatine wurde zuerst auf Kurzschlüsse und Schaltungsfehler untersucht. Anschließend wurden die Relais auf ihre Funktion hin überprüft. Nachdem diese Prüfungen erfolgreich waren, konnte der analoge Signalpfad gemessen werden. Um diese Messung durchzuführen, wurden für jeden Kanal zwei Messungen mit dem UPV durchgeführt. Die erste Messung erfolgte im DC-Modus, bei dem Frequenzen bis 0 Hz und DC-Anteile durchgelassen werden. Die zweite Messung erfolgte im AC-Modus, mit zugeschaltetem Entkopplungskondensator. Bei der ersten Messung wird erwartet, dass der Amplituden- und Phasengang linear bei 0dB, bzw. 0° verläuft. Bei der zweiten Messung sollte sich der Hochpass am Eingang bemerkbar machen. Die für die Messungen verwendete Konfiguration des UPV ist auf der CD enthalten. Das Ergebnis der Messung im DC-Modus für jeden Kanal ist in Abbildung 5.11 dargestellt.

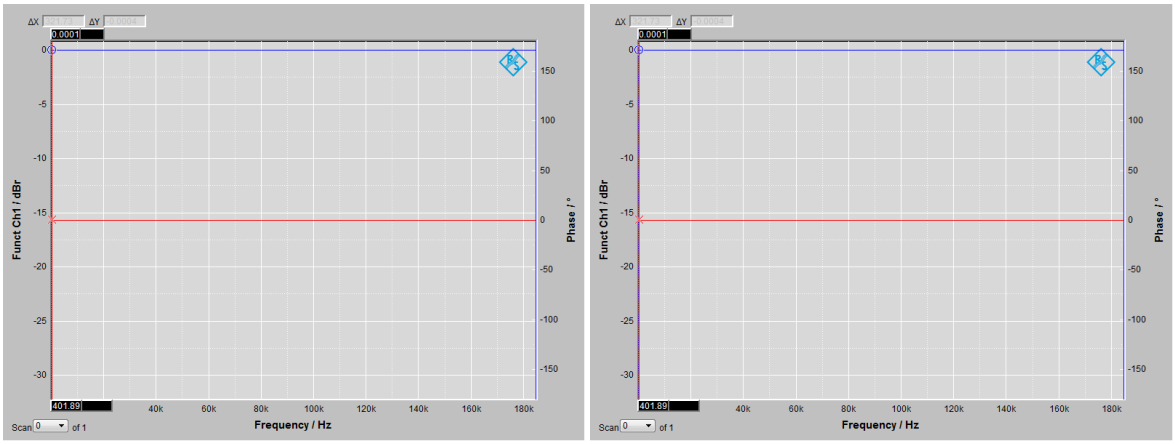


Abbildung 5.11.: Messung der Analyzerplatine im DC-Modus

Links ist Kanal A, rechts Kanal B zu sehen. Wie erwartet, zeigt sich der lineare Verlauf von Amplitude und Phase.

Die zweite Messung, dieses mal im AC-Modus ist in Abbildung 5.12 zu sehen. Um den

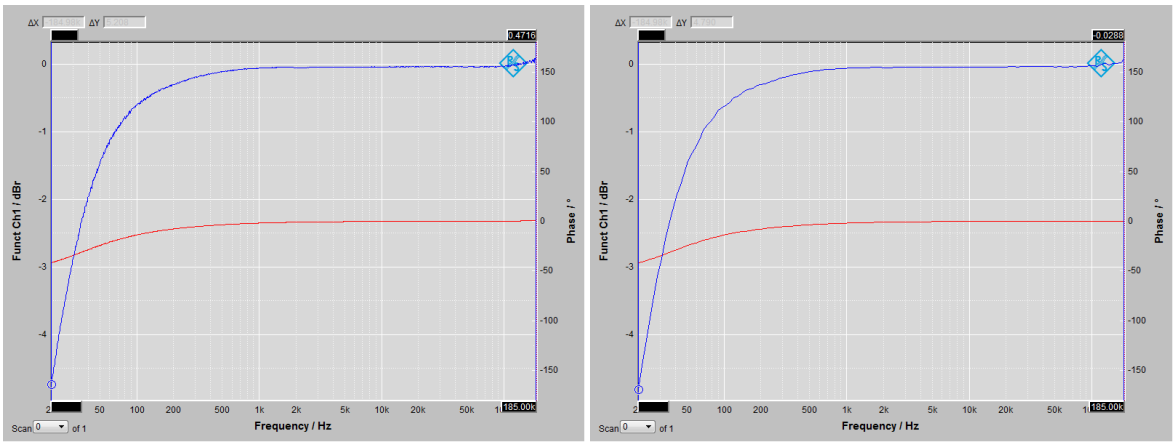


Abbildung 5.12.: Messung der Analyzerplatine im DC-Modus

Frequenzverlauf bei niedrigen Frequenzen besser zu erkennen, ist hier eine logarithmische Einteilung der Frequenzachse gewählt worden. Wie zu erkennen ist, macht sich die AC-Kopplung wie erwartet bemerkbar. Die Grenzfrequenz des Hochpassfilters liegt bei ca. 30 Hz. Ein nahezu linearer Verlauf von Amplitude und Phase wird ab ca. 1 kHz erreicht.

5.4.2. Messung der Generatorplatine

Wie auch bei der Analyzerplatine ist bei der Generatorplatine zuerst eine Überprüfung der Schaltung auf Fehler oder Kurzschlüsse erfolgt. Anschließend wurden Messungen mit dem UPV durchgeführt, um die analogen Signalwege zu überprüfen. Hierbei sollte die Charakteristik des analogen Tiefpassfilters zu erkennen sein. Die Einstellungen des UPV sind identisch mit denen der Analyzerplatinenmessung. Das Ergebnis der Messung ist in Abbildung 5.13 dargestellt. Wie zu erkennen ist, zeigt sich ein Signalverlauf, der sich nicht mit der Simulation in ELSIE deckt. Dies liegt an der Ausgangsimpedanz des UPV, die nicht 50Ω, sondern lediglich 5 Ω beträgt. Die Generatorplatine kann somit nur in Kombination mit dem EVM gemessen werden.

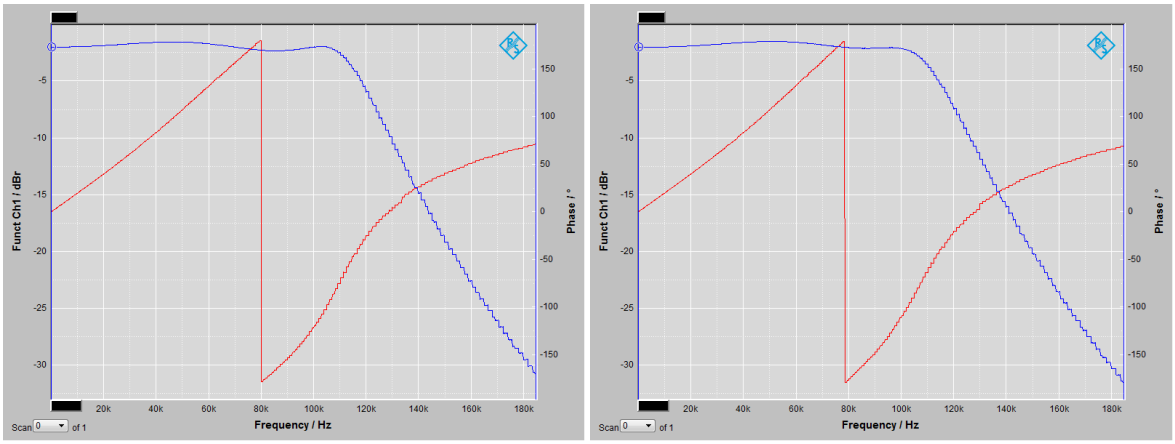


Abbildung 5.13.: Messung der Generatorplatine

5.4.3. Messung des Gesamtsystems

Um die Generatorplatine korrekt messen zu können, muss sie so betrieben werden, wie es im Filterdesign vorgesehen ist. Deshalb und auch um das Gesamtsystem, bestehend aus Analyzerplatine, EVM und Generatorplatine zu messen, erfolgte die folgende Messung.
Wie auch bei der Auswahl der Hardware, wird eine Talk-Through Messung durchgeführt. Dazu wird auf dem DSP eine Software implementiert, welche die Eingangssignale direkt wieder auf den Ausgängen ausgibt. Als Parameter wurden die gleichen Einstellungen wie in den vorangegangenen Messungen gewählt. In Abbildung 5.14 ist das Ergebnis dieser Messung zu sehen. Wie zu erkennen ist, zeigt sich der erwartete Verlauf von Amplitude und Phase. Da

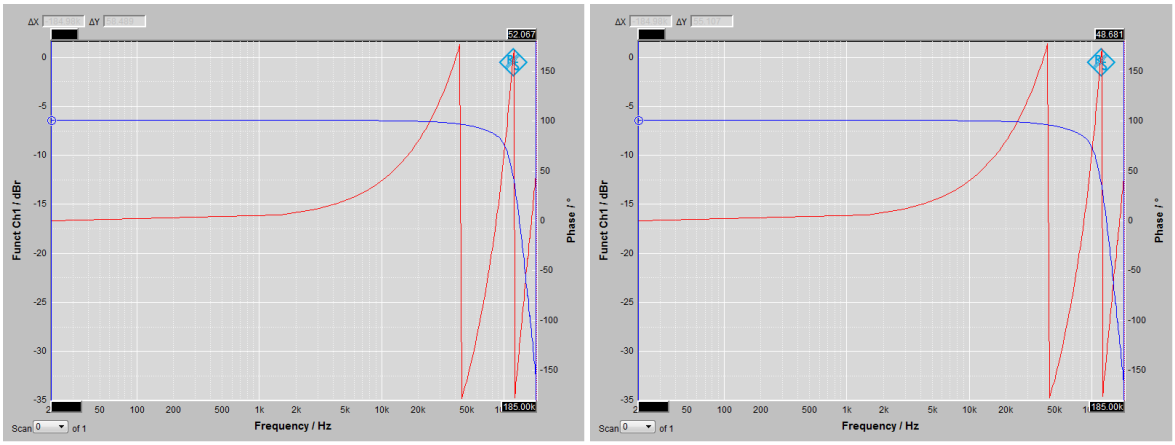


Abbildung 5.14.: Messung des Gesamtsystems

die Analyzerplatine über einen linearen Frequenzgang verfügt, ist hier der Amplitudengang des analogen Filters der Generatorplatine zu erkennen. Die Phase dreht sich etwas schneller als in der Simulation des Filters. Hier macht sich die Verzögerung des EVM bemerkbar, die das Signal aufgrund der Abtastfrequenz von 500 kHz konstant um 2 µs verzögert.

6. Softwareentwicklung

6.1. Planung

Die Softwareentwicklung beinhaltet die Entwicklung der notwendigen Software, um aus der entwickelten Hardware und dem EVM, einen funktionsfähigen Audio Analyzer zu erstellen. Als Programmiersprache wurde C genutzt.
Um die Softwareentwicklung durchzuführen wurde zuerst, wie auch bei der Hardwareentwicklung, eine Anforderung gestellt. Die Software muss in der Lage sein, die notwendige Signalverarbeitung zur Audioanalyse durchzuführen und diese dem Benutzer zur Verfügung zu stellen. Dazu muss ein User-Interface erstellt werden, über das der Benutzer die Messungen parametrieren kann und in welchem anschließend die Messergebnisse dargestellt werden. Die zu entwickelnde Software gliedert sich deshalb in drei Komponenten:

- Signalverarbeitung zur Audioanalyse
- Bereitstellung des User-Interaces und Kommunikation mit dem User-Interface
- User-Interface

Da der verwendete DSP zwei Prozessorkerne enthält, können die Komponenten auf diese beiden Kerne aufgeteilt werden. Es wurde sich entschieden, die Bereitstellung des User-Interfaces, sowie die Kommunikation des Benutzers über dieses Interface, auf dem einen Kern (Kern 0) und die zeitkritische Signalverarbeitung in dem anderen Kern (Kern 1) zu implementieren.
Als User-Interface stehen theoretisch zwei Varianten zur Verfügung. Zum Ersten gibt es die Möglichkeit, ein Display mit Eingabemöglichkeit direkt an das EVM anzuschließen. Die zweite Variante ist die Nutzung eines webbasierten User-Interfaces, indem ein Webserver auf dem DSP implementiert wird.

Es wurde sich für die webbasierte Lösung entschieden, da es keine direkte Anschlussmöglichkeit eines Displays an das EVM gibt. Weiterhin müssten zusätzlich spezielle Displaytreiber für die Ansteuerung des Displays und die grafische Darstellung implementiert werden. Ob und in welchem Umfang diese für das EVM verfügbar sind ist abhängig vom verwendeten Display. Bei der webbasierten Lösung findet die Darstellung in einem Webbrowser eines Client-PCs statt, der lediglich eine Verbindung zum Webserver herstellen muss. Die Implementierung eines Webservers im Kern 0 ist durch das Network Development Kit (NDK) von TI möglich. Der Vorteil ist, dass für das User-Interface verschiedene Clients genutzt werden können und eine Bedienung des Audio-Analyzers auch aus der Ferne erfolgen kann.
In Abbildung 6.1 ist zu sehen, wo im DSP die einzelnen Softwarekomponenten angesiedelt sind und wie diese miteinander kommunizieren.

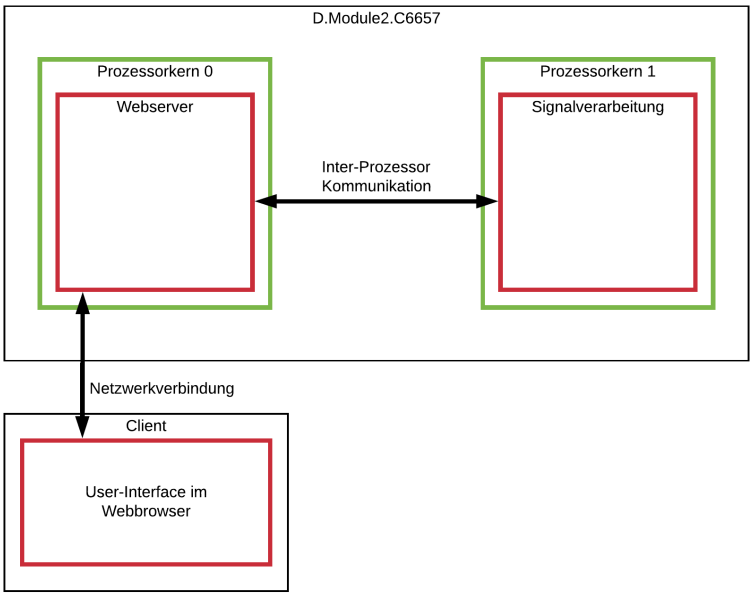


Abbildung 6.1.: Einzelne Softwarekomponenten und ihre Aufteilung

Damit beide Prozessoren untereinander kommunizieren und Daten ausgetauscht werden können, müssen Mechaniken der Inter-Prozessor-Kommunikation genutzt werden. Die Nutzung der Prozessorressourcen sowie die Abarbeitung von Aufgaben mit Echtzeitanforderungen muss ebenfalls verwaltet werden. Hierzu wird der von TI bereitgestellte RTOS-Kernel „SYS/-BIOS“ verwendet, welcher in Kapitel 6.2.3 genauer beschrieben ist.

6.2. verwendete Software

6.2.1. Entwicklungstools

6.2.1.1. Code Composer Studio

Um die Software für den C6657 DSP zu entwicklen, wurde das von TI zur Verfügung stehende Code Composer Studio (CCS) genutzt. Dies ist eine Entwicklungsumgebung auf Eclipse-Basis, die für die Softwareentwicklung von TI-Prozessoren angepasst ist. Sie unterstützt die Programmierung und das Debugging des verwendeten C6657-DSP über verschiedene JTAG-Debugger. Codegenerierung mit einem, für den Prozessor optimierten, Compiler ist im CCS integriert. Als Debug-Interface wurde in dieser Arbeit der ebenfalls von TI hergestellte „XDS-200 Debugger“ genutzt.

6.2.1.2. TeraTerm

„TeraTerm“ ist ein Terminalprogramm, welches eine Verbindung mit einer Universal Asynchronous Receiver and Transmitter (UART)-Schnittstelle herstellt. Es unterstützt zusätzlich zur Ein- und Ausgabe von Zeichenstrings das Übertragen von Binärdateien. Dies wird im Verlauf dieser Arbeit benötigt, um den Programmcode auf den nichtflüchtigen Speicher des D.Module2.C6657 zu übertragen und über die UART mit dem Prozessor zu kommunizieren.

6.2.1.3. binsrc

„binsrc“ ist ein MS-DOS Programm zum Umwandeln von Textdateien in Datenarrays. Es ist Bestandteil des NDK von TI. Die Datenarrays liegen nach der Konvertierung als C-Code vor, der dann in ein Projekt integriert werden kann. Es wurde genutzt um die Website des User-Interfaces so zu konvertieren, dass sie vom Webserver eingebunden werden kann.

6.2.1.4. TortoiseSVN

TortoiseSVN ist eine Versionsverwaltungssoftware basierend auf Subversion. Damit ist es möglich, Software zu versionieren und somit Projektstände abzuspeichern und bei Bedarf zu laden. Es wurdce verwendet, um die CCS-Projekte für Kern 0 und Kern 1 zu verwalten.

6.2.2. D.Module2.BIOS

Das D.Module2.BIOS ist ein von d.signt entwickeltes BIOS, welches eine Softwareschnittstelle zu der auf dem D.Module2.C6657 befindlichen Hardware bereitstellt und zusätzlich die Funktion eines Bootloaders übernimmt. Es ist in den Beispielprojekten, welche von d.signt mitgeliefert werden, bereits integriert. Ist das BIOS in ein Projekt integriert, können seine Funktionen genutzt werden. Dazu gehört zum Beispiel das Neustarten eines Prozessorkerns, die Kommunikation mit auf dem EVM befindlicher Peripherie oder das Senden und Empfangen von Daten über Schnittstellen.

Beim Start des EVM wird die im BIOS enthaltene Bootloadersoftware gestartet. Über diese ist es möglich, eine neue Software in den nichtflüchtigen Speicher zu laden. Die dazu notwendige Prozedur ist in Kapitel 6.4 zu finden.
Erfolgt ein normaler Prozessorstart, wird anhand einer Konfigurationsdatei, die Software für die Prozessorkerne geladen und eine Initialisierung der Peripherie vorgenommen.

6.2.3. SYS/BIOS

„SYS/BIOS“ ist der von TI bereitgestellte RTOS-Kernel des TI-RTOS. Er bietet die Funktionalität eines echtzeitfähigen Schedulers, der Aufgaben entsprechend ihrer Priorisierung und auftretende Interrupts abarbeitet. Die Priorisierung erfolgt anhand des in Abbildung 6.2 dargestellten Schemas. Wie zu erkennen ist, gibt es verschiedene Arten von ausführbaren Threads,

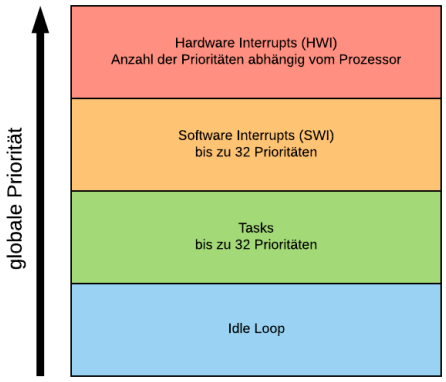


Abbildung 6.2.: Prioritäten im SYS/BIOS

die unterschiedlich priorisiert sind. Die höchste Priorität haben die Hardware-Interrupts (HWIs). Dies sind Interrupts, die im Interruptcontroller des Prozessors auftreten und abhängig vom verwendeten Prozessor sind. Sie können im SYS/BIOS aktiviert oder deaktiviert werden und mit einer aufzurufenden Funktion versehen werden. Tritt ein HWI auf und ist dieser aktiviert, wird diese Funktion ausgeführt. Die Ausführung kann vom SYS/BIOS nicht unterbrochen werden und nur verspätet geschehen, wenn ein höher priorisierter HWI gerade ausgeführt wird. Sie müssen daher immer bis zum Ende ausgeführt werden, da ansonsten der Prozessor in seiner weiteren Ausführung blockiert wird.
Die nächstniedrigere Priorität haben die Software-Interrupts (SWIs). Dies sind Interrupts, die nicht am Interruptcontroller anliegen, sondern vom SYS/BIOS in der Software definiert sind. Sie unterscheiden sich in der Verwendung nicht von HWIs, sondern haben lediglich eine geringere globale Priorität.
Die HWIs und SWIs können somit nur drei Zustände haben. Sie können inaktiv sein (Inactive), wenn sie deaktiviert wurden, ausgeführt werden (Running), oder auf ihre Ausführung warten (Ready).
Unterhalb der SWIs liegen die Tasks. Sie können im Gegensatz zu den Interrupts in ihrer Ausführung blockiert werden, wenn eine höher priorisierte Task oder ein Interrupt ausgeführt werden muss. Hier muss deshalb bei der Programmierung besonders darauf geachtet werden, dass eine Unterbrechung der Task zu jedem Zeitpunkt möglich ist. Ist dies nicht möglich, müssen vor der kritischen Stelle die Interrupts deaktiviert und anschließend wieder aktiviert werden.
Im Gegensatz zur klassischen Programmierung, ohne die Verwendung eines RTOS, bietet sich bei Verwendung des SYS/BIOS die Möglichkeit, mehrere Tasks mit Endlosschleifen zu programmieren. Der Scheduler sorgt dafür, dass alle Tasks abhängig von ihrer Priorität ausgeführt werden. Deshalb haben Tasks, im Gegensatz zu den Interrupts andere mögliche Zustände, die in Abbildung 6.3 aufgezeigt sind. Tasks werden ausgeführt, wenn sie sich im Zustand Running befinden. Warten sie auf ihre Ausführung, da höher priorisierte Threads ausgeführt werden, befinden sie sich im Zustand Ready. Eine Task wird in den Zustand Blocked versetzt, wenn entsprechende SYS/BIOS-Funktionen aufgerufen werden. Die Task kann damit für eine festgelegte Zeit, oder durch das Warten auf ein Event blockiert werden. Müssen höher priorisierte Threads ausgeführt werden, werden Tasks ebenfalls in den Zustand Blocked versetzt. Der Zustand Terminated wird erreicht, wenn eine Task beendet wird.

Das CCS bietet die Möglichkeit, ein Projekt, welches das „SYS/BIOS“ nutzt, grafisch zu

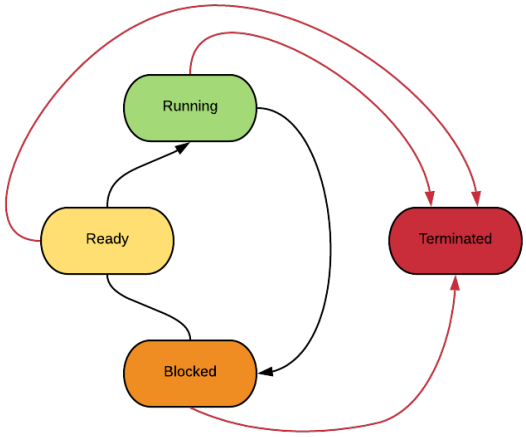


Abbildung 6.3.: Taskzustände im SYS/BIOS

konfigurieren. Notwendiger Initialisierungsquellcode wird automatisch erzeugt und in das Projekt integriert. Zusätzlich können weitere Softwarebibliotheken des TI-RTOS einfach eingebunden werden. Dazu zählen unter anderem ein Ethernetstack und die Inter-Prozessor-Kommunikation.

Um das SYS/BIOS für ein Projekt zu konfigurieren, kann dies im grafischen Editor „XGCONF“ des CCS erfolgen. Als Beispiel ist in Abbildung 6.4 die Konfiguration eines SWI dargestellt, für dessen Erstellung verschiedene Parameter festgelegt werden müssen. Dazu gehört die Definition eines Handles, einer aufzurufenden Funktion, eine Priorität sowie ein Initialwert.

Aus den Eingaben im XGCONF wird ein Konfigurationsskript erstellt, aus dem, vor dem

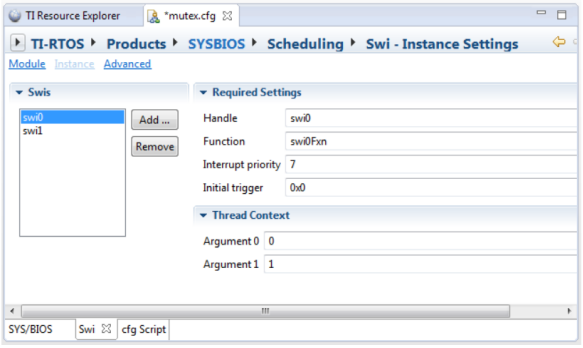


Abbildung 6.4.: Konfiguration eines Software-Interrupts im XGCONF (Quelle: [12])

Kompilieren, Programmcode in C erstellt wird. Dieser wird anschließend automatisch eingebunden und beinhaltet die Erstellung der notwendigen Objekte wie z.B dem Handle oder dem

Funktionsprototyp einer aufzurufenden Funktion. Sie dürfen deshalb im Programmcode nicht selbst definiert werden.

6.3. Entwicklung

6.3.1. Projekterstellung

Um die Software für beide Prozessorkerne zu entwickeln, wurden einzelne Projekte im CCS erstellt, die jeweils die Software für einen Kern beinhalten. Als Vorlage diente das von d.signt mitgelieferte Beispielprojekt „dualcore“ für das D.Module2.C6657. Beide Projekte nutzen das SYS/BIOS für die Verwaltung der Prozessorressourcen und beinhalten dementsprechend eine Konfigurationsdatei, die über XGCONF bearbeitet werden kann. Die zu programmierenden Softwareteile wurden in Module, die jeweils eine Aufgabe erfüllen, gegliedert. Ein Modul besteht immer aus einer Headerdatei (.h) und einer C-Datei (.c). Als Beispiel ist hier der Treiber für die I2C-Schnittstelle zu nennen. Das Modul beinhaltet alle Funktionen, um diese Schnittstelle zu initialisieren und anschließend zu nutzen.

Für die Entwicklung des User-Interfaces wurde eine Website mit HTML, CSS und Javascript erstellt. Diese wurde anschließend mit dem in Kapitel 6.2.1.3 beschriebenen Tool in das Projekt für Kern 0 integriert. Die erstellten CCS-Projekte für beide Prozessorkerne und der Quellcode des User-Interfaces sind auf der CD, die dieser Arbeit beiliegt, enthalten.
Die verwendeten Softwarepakete sind im Anhang E mit ihrer verwendeten Version aufgelistet.

6.3.2. Speicheraufteilung

Um die verfügbaren Speicherressourcen des C6657 zu nutzen, muss für jedes Projekt eine Speicherzuteilung erfolgen. Dies erfolgt über eine Linker-Command Datei, in der die verwendbaren Speicherbereiche mit ihren Adressen bekannt gegeben werden. Das CCS bietet hier ebenfalls die Möglichkeit, diese Speicherzuteilung über eine grafische Oberfläche durchzuführen. Die Linker-Command Datei wird anschließend automatisch generiert. Dem

User Guide des D.Module2.C6657 [13] ist die vorgeschriebene Speicherzuordnung entnommen worden.
Für die Speicheraufteilung beider Projekte wurde die in Tabelle 6.1 zu sehende Konfiguration gewählt. Die Startadressen sind globale Adressen und können somit theoretisch von beiden Prozessoren aus genutzt werden. Voraussetzung ist, dass diese dem Prozessor bekannt sind. Wie zu erkennen ist, sind einige Speicherbereiche in beiden Kernen angegeben. Der MSMCSRAM, L1DSRAM und L1PSRAM. Ein Zugriff ist somit von beiden Kernen aus möglich. Diese Zuteilung ist für den MSMCSRAM notwendig, um die Inter-Prozessor-Kommunikation, welche in Kapitel 6.3.5 beschrieben wird, zu realisieren. Der auszuführende Programmcode für jeden Kern befindet sich im L2SRAM. Damit ein unabsichtliches Zugreifen beider Prozessoren

Tabelle 6.1.: Speicherbelegung der Software

Kern 0	**Startadresse**	**Länge**
MSMCSRAM	0x0C002000	0x000FE000
L1DSRAM	0x00F00000	0x00008000
L1PSRAM	0x00E00000	0x00008000
L2SRAM_0	0x10800000	0x00100000
DDR3_0	0x80000000	0x04000000
L2SRAM_1	0x11800000	0x00100000
Kern 1		
MSMCSRAM	0x0C002000	0x000FE000
L1DSRAM	0x00F00000	0x00008000
L1PSRAM	0x00E00000	0x00008000
L2SRAM_1	0x11800000	0x00100000
DDR3_1	0x84000000	0x04000000

weitestgehend vermieden wird, sollte darauf geachtet werden, dass Speicherbereiche, die von einem Kern genutzt werden, dem anderen Kern gar nicht erst bekannt gemacht werden. Eine Ausnahme bildet der L2SRAM des Kern 1 (L2SRAM_1), der ebenfalls dem Kern 0 bekannt ist. Dies ist notwendig, da der D.Module2.BIOS Bootloader im Kern 0 ausgeführt wird und die Software für den Kern 1 in dessen L2SRAM lädt. Dem Kern 1 ist hingegen nur sein eigener L2SRAM bekannt, so dass dieser nicht den Speicher des Kern 0 nutzen kann. Der

DDR3-RAM wurde so aufgeteilt, das beide Prozessorkerne ihren eigenen Speicherbereich besitzen.
Die L1-Speicher werden nicht genutzt, sind aufgrund der Vollständigkeit der Speicherzuteilung aber trotzdem enthalten.

6.3.3. Software Prozessorkern 0

Die Bereitstellung des User-Interfaces erfolgt über einen Webserver, der im Kern 0 des DSP eingerichtet wird. Dazu wurde zusätzlich das NDK von TI, welches eine Erweiterung des SYS/BIOS ist, in das Projekt eingebunden. In Abbildung 6.5 ist das Softwarelayout für den Kern 0 dargestellt. Die Software nutzt, wie schon beschrieben das SYS/BIOS und das

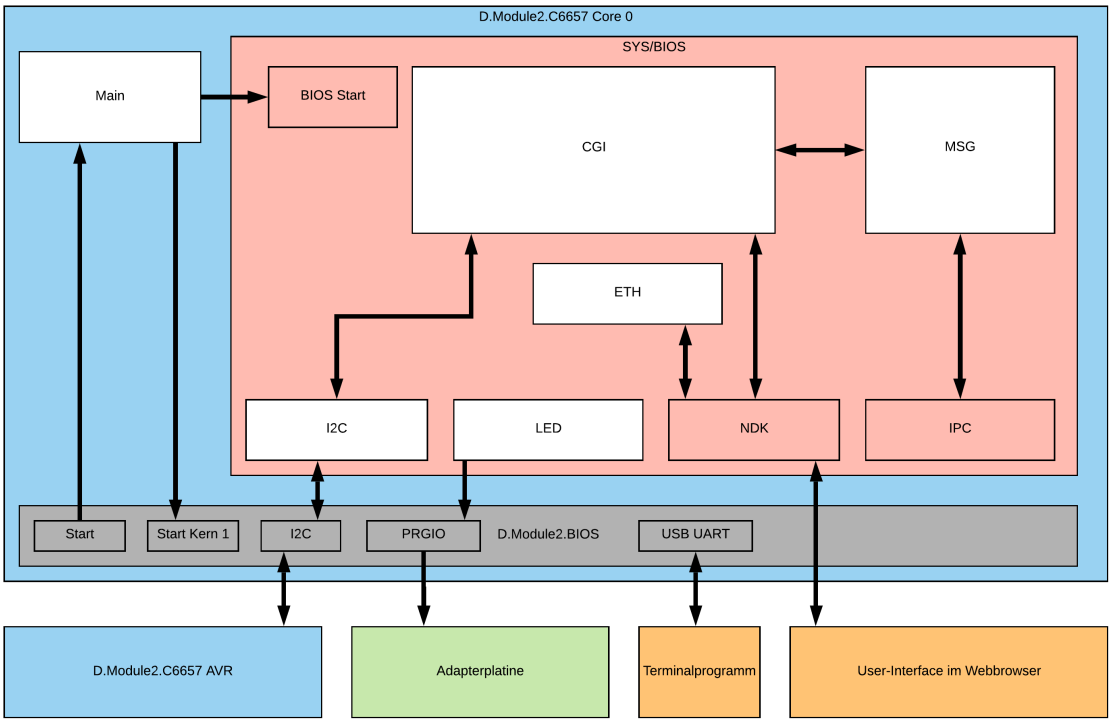


Abbildung 6.5.: Layout der Software für den Kern 0

D.Module2.BIOS. Als Erweiterung des SYS/BIOS sind das Inter-Processor-Communication (IPC) und das NDK-Modul in die Software integriert. Der Start der Software erfolgt aus dem

D.Module2.BIOS, nach Aufruf des Bootloaders. In der Main-Funktion wird eine Initialisierung des Prozessors durchgeführt. Die USB-Schnittstelle des EVM ist als UART konfiguriert um eine Ausgabemöglichkeit bei bestehender Verbindung zu einem Terminalprogramm zu realisieren. Nach der Initialisierung wird über das D.Module2.BIOS ein Start des Kern 1 eingeleitet. Anschließend erfolgt der Start des SYS/BIOS-Schedulers.
Das NDK ist so konfiguriert, dass es eine Netzwerkverbindung herstellt, wenn im angeschlossenen Netzwerk ein DHCP-Server vorhanden ist, der dem EVM eine IP-Adresse vergibt. Der Verbindungsstatus und die IP-Adresse werden bei jeder Änderung über die UART ausgegeben. Die Konfiguration des NDK und Definition von Callback-Funktionen sowie die Einrichtung des Webservers erfolgen im Modul ETH. Ist der Webserver eingerichtet, kann durch Eingabe der IP-Adresse im Browser eines Client-PCs eine Verbindung hergestellt und auf das User-Interface zugegriffen werden.
Für die Ansteuerung der LEDs auf der Adapterplatine ist das LED-Modul zuständig. Dieses setzt die GPIOs der LEDs je nach Systemzustand über den im D.Module2.BIOS enthaltenen Treiber.
Das I2C-Modul ist entwickelt worden, um mit dem auf dem EVM befindlichen Mikrocontroller zu kommunizieren. Über diesen kann die Temperatur des EVM erfasst werden. Für die Kommunikation mit dem User-Interface wird das sogenannte Common Gateway Interface (CGI) genutzt. Dafür ist ein Modul (CGI) entwickelt worden, welches die Callback-Funktion für Nachrichten über das CGI enthält. In der Callback-Funktion werden die Nachrichten abgearbeitet und eine Antwort an das User-Interface im Browser des Clients geschickt. Um Daten zum Kern 1 zu schicken oder von diesem zu erhalten, ist das Message-Modul (MSG) implementiert worden. Dieses initialisiert das IPC-Modul um mit dem Kern 1 zu kommunizieren.

6.3.4. Software Prozessorkern 1

Die Signalverarbeitung soll, wie in der Planung vorgesehen, im Kern 1 des DSP durchgeführt werden. Das Softwarelayout hierfür ist in Abbildung 6.6 zu sehen. Der Kern 1 wird von der

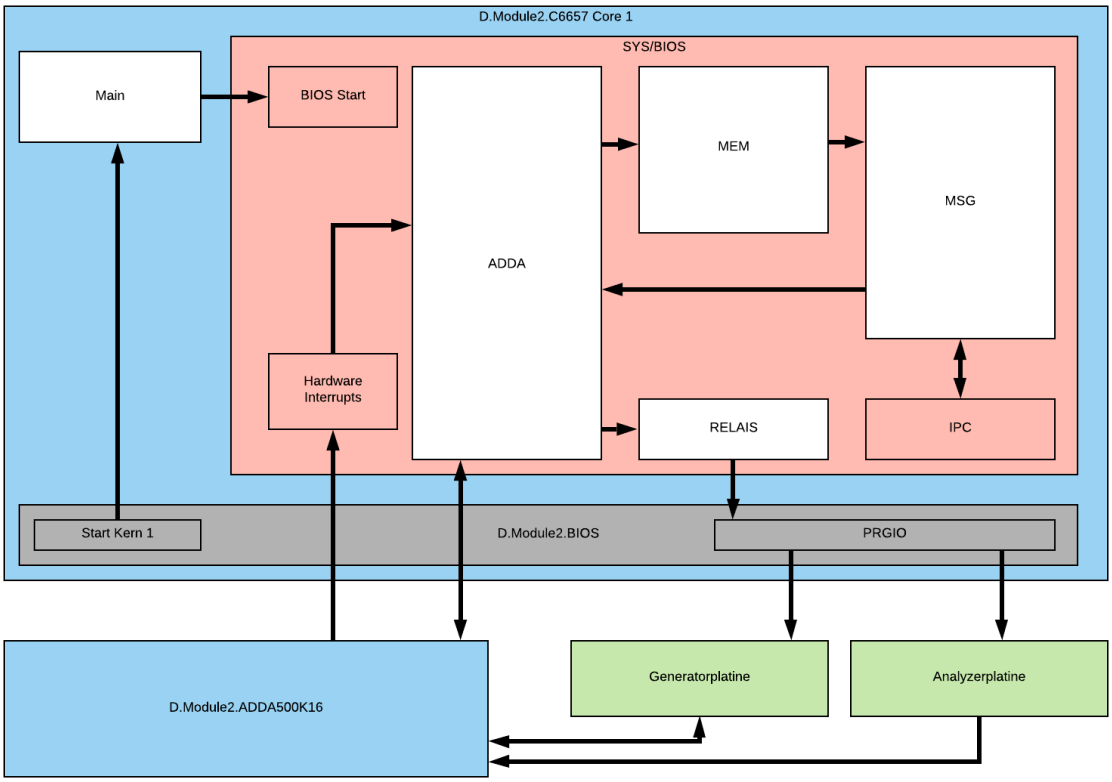


Abbildung 6.6.: Layout der Software für den Kern 1

Software im Kern 0 mithilfe des D.Module2.BIOS gestartet. Anschließend findet, wie im Kern 0, eine Initialisierung statt und es wird der SYS/BIOS-Scheduler gestartet. Das IPC-Modul des SYS/BIOS ist ebenfalls integriert, um die Kommunikation mit dem Kern 0 herzustellen. Dazu ist, wie auch in Kern 0, das MSG-Modul implementiert worden. Des Weiteren werden HWIs genutzt, damit das D.Module2.ADDA500K16 der Software signalisieren kann, dass Daten am A/D-Wandler verfügbar sind und der D/A-Wandler beschrieben werden kann. Der Interrupthandler hierfür ist im Modul ADDA enthalten. Zusätzlich sind in diesem Modul sämtliche Algorithmen für die Signalverarbeitung enthalten. Weiterhin findet hier die Konfiguration der Generator- und Analyzerplatine statt. Um die Relais der analogen Hardware zu setzen, ist das RELAIS-Modul zuständig. Dieses verwendet Funktionen des D.Module2.BIOS zum

Setzen der GPIOs.
Für die Speicherung von Messdaten wurde das MEM-Modul entwickelt. Dieses enthält einen Ringspeicher, der vom ADDA-Modul mit Daten beschrieben wird.

6.3.5. Inter-Prozessor-Kommunikation

Für die Inter-Prozessor-Kommunikation wird das im TI-RTOS vorhandene IPC-Modul genutzt. Der User Guide hierzu ist unter [14] zu finden. Das IPC-Modul stellt unter anderem eine sogenannte MessageQ für die Kommunikation zur Verfügung. Mit dieser MessageQ kann ein beliebiges Datenpaket von einem Prozessor zu einem anderen geschickt werden. Das IPC-Modul benötigt dafür einen Speicherbereich, der allen Prozessoren, die untereinander kommunizieren möchten, bekannt ist. Wie im Kapitel 6.3.2 beschrieben, ist dazu der MSMCSRAM verwendet worden. Dieser muss im Konfigurationsskript beider Projekte als „Shared-Region “ festgelegt werden. Anschließend kann in der „Shared-Region“ eine MessageQ eingerichtet und genutzt werden. Dazu muss ein Prozessor eine MessageQ im gemeinsamen Speicherbereich anlegen und über einen eindeutigen Namen bekannt machen. Der zweite Prozessor kann nun diese erstellte MessageQ unter Angabe des Namens öffnen und darauf warten, dass über diese MessageQ ein Datenpaket gesendet wird. Hat er das Datenpaket erhalten, kann er seinerseits ein Datenpaket als Antwort senden. Das SYS/BIOS bietet Funktionen an, um auf das Erhalten einer Nachricht zu warten. Diese können im Kontext einer Task aufgerufen werden, um die Task an dieser Stelle zu blockieren. Sie wird dann erst wieder ausgeführt, wenn die MessageQ erhalten wurde.

Die Kommunikation über das IPC-Modul ist in beiden Kernen im Modul MSG realisiert. Um die Kommunikation durchzuführen, muss allerdings zuerst eine Synchronisation beider Prozessorkerne erfolgen. Die notwendige Prozedur dazu ist in Abbildung 6.7 dargestellt.

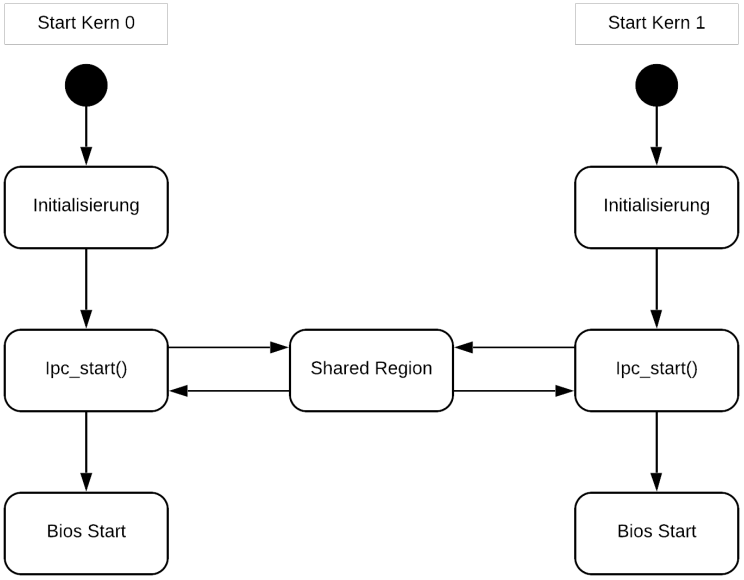


Abbildung 6.7.: Synchronisation der Prozessoren über das IPC-Modul

Beide Prozessoren müssen die Funktion „Ipc_start()“ vor dem Start des Bios-Schedulers aufrufen. Über die, beiden Prozessoren bekannte, „Shared-Region “ synchronisieren sie sich und können anschließend kommunizieren. Die Funktion wird erst verlassen und der Scheduler gestartet, wenn alle Prozessoren die das IPC-Modul nutzen, diese Funktion aufgerufen haben.
Die weitere Initialisierung und der Datenaustausch findet in jeweils einer Task in jedem Prozessor statt. Dabei übernimmt der Kern 0 die Steuerung der Kommunikation. Abbildung 6.8 zeigt, wie die Einrichtung der MessageQ und die Kommunikation abläuft. Der Heap für das Anlegen einer MessageQ wird vom Kern 1 erstellt. Dieser ist danach von beiden Kernen aus zu erreichen. Anschließend erstellen beide Kerne in diesem Heap abwechselnd eine lokale MessageQ. Danach ist die Initialisierung abgeschlossen und die Tasks laufen fortan in einer Endlosschleife. Kern 1 wartet zuerst darauf, dass eine MessageQ empfangen wird. Solange dies nicht der Fall ist, befindet sich die Task im Zustand Blocked.
Kern 0 wartet, im Gegensatz zu Kern 1 darauf, dass eine MessageQ gesendet werden soll. Dies ist durch eine Semaphore realisiert, die von beliebigen anderen Threads freigegeben werden kann, wenn sie eine MessageQ versenden wollen. Solange dies nicht der Fall ist, befindet sich die Task ebenfalls im Zustand Blocked.

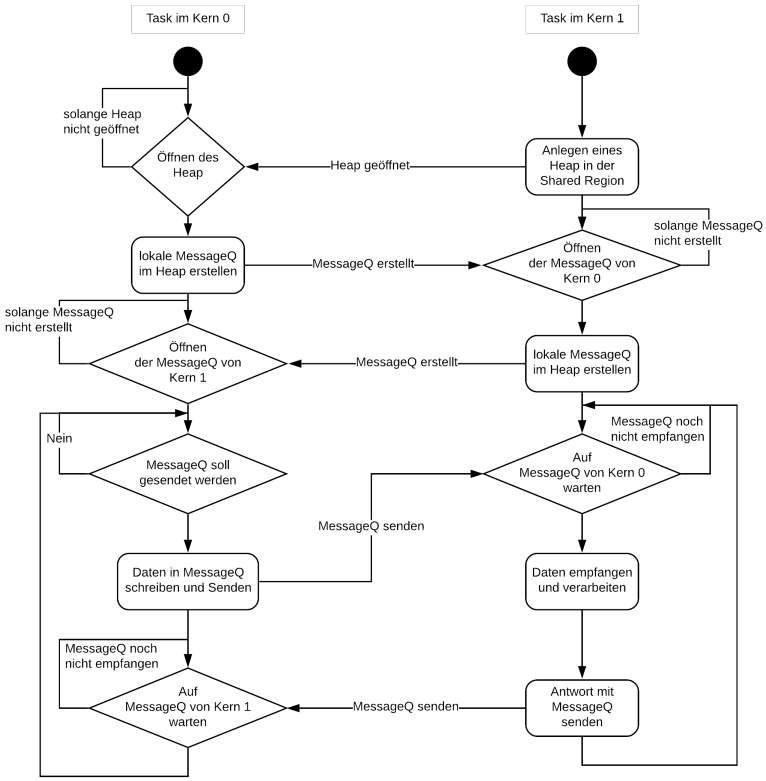


Abbildung 6.8.: Initialisierung der MessageQ und Kommunikationl

Soll eine MessageQ gesendet werden, muss diese zuerst mit Daten gefüllt werden. Dafür ist eine global bekannte Struktur im MSG-Modul vorhanden, die von allen Threads mit Daten gefüllt werden kann. Anschließend gibt dieser die Semaphore frei, die Task wechselt zum nächstmöglichen Zeitpunkt in den Zustand Running und die zu übertragenden Daten werden in die MessageQ kopiert. Abschließend wird die MessageQ gesendet und auf dem Empfang der Antwort gewartet. Die Task im Kern 1 wechselt ebenfalls in den Zustand Running, wenn die MessageQ empfangen worden ist. Die Daten werden dann verarbeitet und anschließend wird eine Antwort gesendet. Ist die Antwort im Kern 0 angekommen, werden die Daten weitergeleitet und die Kommunikation beginnt von vorne, indem die Task auf das Freigeben der Semaphore wartet.

6.3.6. Netzwerkverbindung und Webserver

Für die Realisierung eines Webservers im Kern 0 und die Initialisierung einer Netzwerkverbindung wurde das von TI, als Erweiterung des SYS/BIOS, entwickelte NDK genutzt. Die Einbindung in die Software wurde nach dem Reference Guide [15] durchgeführt.
Um die Netzwerkverbindung herzustellen, wurde zuerst der DHCP-Client aktiviert. Die dafür notwendigen Einstellungen im XGCONF sind in Abbildung 6.9 dargestellt. Ist der DHCP-Client

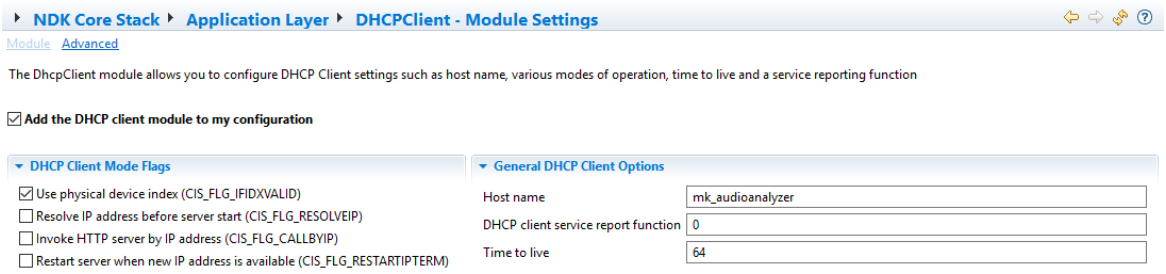


Abbildung 6.9.: DHCP Konfiguration im XGCONF

in das Projekt, durch Setzen des Hakens bei „Add the DHCP client module...“ eingebunden, wird beim Prozessorstart automatisch versucht, eine IP-Adresse von einem DHCP-Server im Netzwerk zu erhalten. Damit der Audio-Analyzer im Netzwerk identifiziert werden kann, wird ein Hostname angegeben. Ist eine IP-Adresszuteilung vom DHCP-Server erfolgt, ist er anschließend unter diesem Namen im Netzwerk zu erreichen. Weitere Einstellungen müssen an dieser Stelle nicht getroffen werden. Der Webserver, welcher ebenfalls Bestandteil des NDK ist, kann auf die gleiche Weise aktiviert werden („Application Layer->HTTP“).
Ist der Webserver aktiviert, müssen diesem jedoch noch Daten hinzugefügt werden. Dies erfolgt über das Embedded File System (EFS), welches ebenfalls im NDK integriert ist. Die genaue Vorgehensweise dazu ist im Reference Guide [15] beschrieben. Das EFS bietet Funktionen um Dateien, welche zuvor als Datenarray konvertiert wurden, hinzuzufügen. Die Konvertierung von HTML-, CSS- oder Javascript-Dateien in Datenarrays erfolgt mit dem in Kapitel 6.2.1.3 erwähnten binsrc-Tool. Damit das NDK das Hinzufügen der Dateien nach der Initialisierung der Netzwerkverbindung durchführt, wurden Callback-Funktionen definiert, die in der Konfiguration des NDK wie in Abbildung 6.10 eingetragen wurden. Die Callback-Funktionen für das Hinzufügen oder Entfernen von Dateien vom Webserver liegen im Modul ETH und sind als „Stack thread initialization hook“ und „Stack thread delete hook“ definiert.

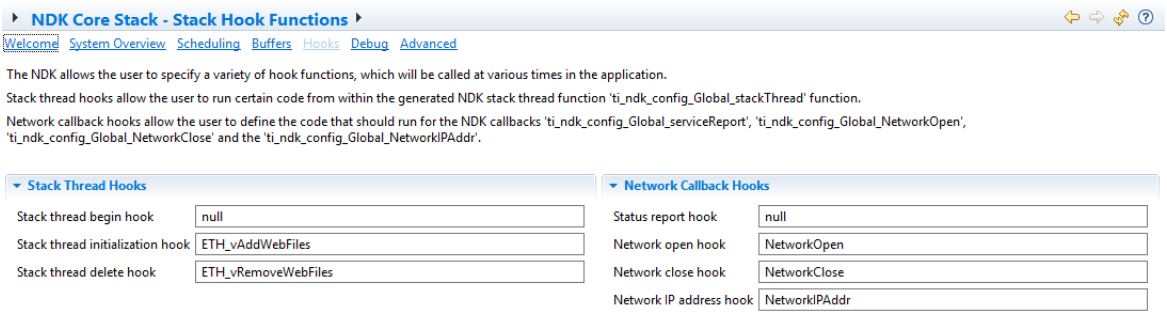


Abbildung 6.10.: Konfiguration der Callback-Funktionen des Network Development Kit

Bei Aufruf dieser Funktionen wird dann das EFS genutzt. In Quellcode 6.1 ist als Beispiel der C-Code für das Hinzufügen von Dateien gezeigt.

```
1 #define INDEX_SIZE 13628
2 static const unsigned char INDEX[] = {};
3 void ETH_vAddWebFiles(void)
4 {
5    efs_createfile("index.html", INDEX_SIZE, (UINT8 *)INDEX);
6    efs_createfile("systemstatus.cgi", 0, (UINT8 *)&CGI_vServeRequest);
7 }
```

Quellcode 6.1: Hinzufügen von Dateien mit dem Embedded File System

Die Funktion „ETH_vAddWebFiles “ ist als Callback-Funktion des NDK festgelegt, wenn eine Netzwerkverbindung erfolgreich hergestellt wurde. Wird sie aufgerufen, werden hier beispielshalber zwei Dateien zum Webserver hinzugefügt. Die erste Datei „index.html“ liegt als Datenarray vor (Zeile 1 und 2). Ist sie dem Webserver hinzugefügt, wird bei einem Zugriff der Inhalt der Datei an den Client geschickt. Die zweite Datei „systemstatus.cgi“ ist eine Pseudodatei, die kein Datenarray, sondern ein Zeiger auf eine Funktion, im Webserver ablegt. Diese wird genutzt um das in Kapitel 6.3.8 erklärte CGI zu nutzen.

6.3.7. User-Interface

Das User-Interface bildet eine Website, die auf dem Webserver im Kern 0 des DSP hinterlegt ist. Für die Programmierung wurde HTML, CSS und Javascript benutzt. Für die Verwendung dieser Programmiersprachen ist als Literatur das Buch „HTML5 und CSS3“ [16] genutzt worden.
Für die grafische Darstellung von Messergebnissen wurde zusätzlich die Javascriptbibliothek „Chart-JS“ genutzt. Diese benötigt für die Ausführung „jQuery“ welches ebenfalls eine Javascriptbibliothek ist, die Funktionen zur DOM-Navigation und -Manipulation enthält. Die Nutzung von „jQuery“ erfolgt normalerweise über das Laden dieser Bibliothek aus dem Internet während der Laufzeit der Website. Da der Audio-Analyzer jedoch auch ohne aktive Internetverbindung funktionsfähig sein soll, wurde der Quellcode der Bibliothek statisch in den Webserver als Datei integriert. Die verwendeten Versionen der Bibliotheken sind in Anhang E zu finden.

Für das Layout des User-Interfaces wurde ebenfalls der UPV als Vorlage verwendet. Dieses besitzt mehrere Fenster, die jeweils eine Funktion erfüllen. Um in einem Webbrowser auf einer Website mehrere Fenster darzustellen, wurde ein freies HTML-Template [17] genutzt. Dieses ermöglicht es, mehrere Fenster zu erstellen und diese mit Inhalt zu füllen. Es enthält eine HTML-, CSS- und Javascriptdatei, in denen das User-Interface programmiert wurde.
Abbildung 6.11 zeigt den Aufbau des User-Interfaces. Es stehen Fenster zur Konfiguration des Analyzers und des Generators zur Verfügung. Eine Messung kann über das „Control“-Fenster gestartet und gestoppt werden. Des Weiteren kann hier der Graph, der in einem eigenen Fenster dargestellt wird, gelöscht werden. Ein numerisches Display, welches ebenfalls ein eigenes Fenster besitzt, zeigt die aktuellen Messwerte von Mess- und Referenzkanal an. Weitere Informationen werden zusätzlich in einem Fenster angezeigt. Die Fenster können innerhalb des Browsers verschoben, minimiert oder maximiert werden. Die Leiste am unteren Rand zeigt einem alle geöffneten Fenster und ermöglicht es, diese in den Vordergrund zu holen. Ein geschlossenes Fenster kann über die Links im oberen linken Bereich erneut geöffnet werden. Der Aufruf und die weitere Bedienung des User-Interfaces ist in Kapitel 8.2 erklärt.

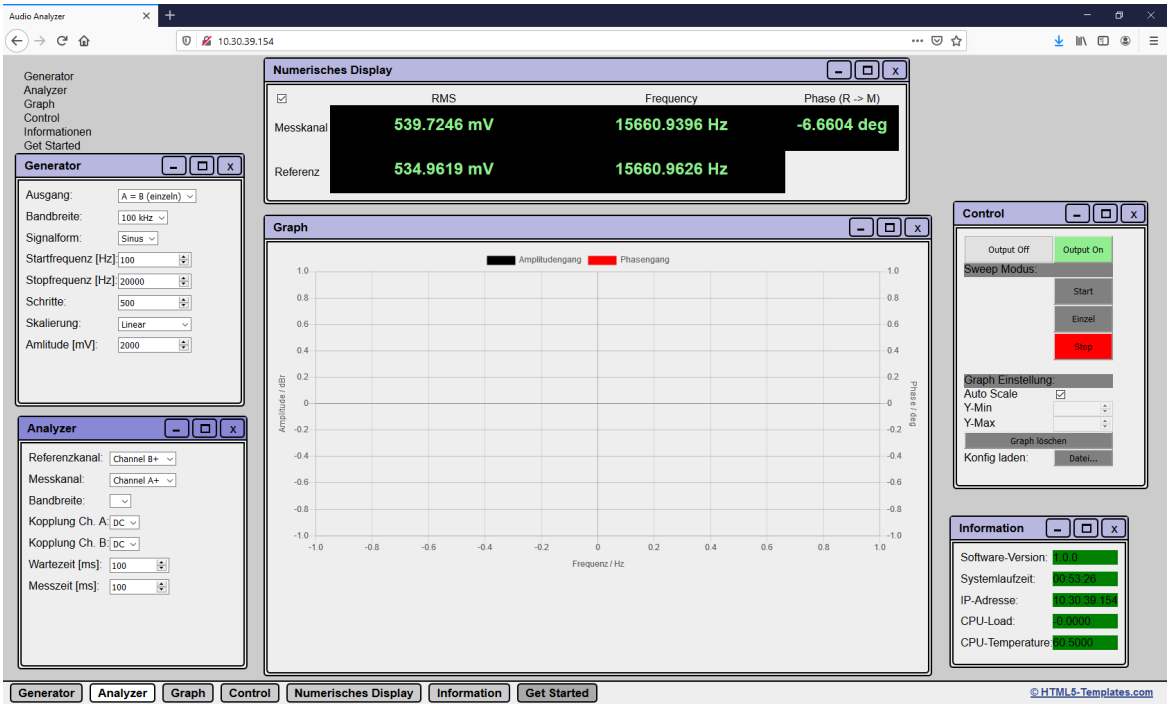


Abbildung 6.11.: Layout des User-Interfaces

6.3.8. Kommunikation mit dem User-Interface

Die Kommunikation mit dem User-Interface, welches im Webbrowser eines verbundenen Clients läuft, erfolgt über das CGI. Dieses Interface wurde gewählt, da es direkt vom NDK unterstützt wird und somit keine weiteren Softwarepakete für die Kommunikation integriert werden müssen.

Das CGI ist ein relativ altes Interface, welches hauptsächlich dafür entwickelt wurde, Websites dynamisch zu erzeugen. Die Kommunikation wird deshalb vom Client, der die Website aufruft, initiiert. Es wird hier die HTML-Anfragemethode POST genutzt. Dieser POST-Methode können Daten in Form von Name-Wert-Paaren angehängt werden. Eine bekannte Anwendung dafür ist beispielsweise das Senden von Daten aus einem Formular einer Website. Der Benutzer füllt hier Felder wie Name, Adresse oder Wohnort aus und sendet diese an den Server. Dieser kann anschließend auf diese Anfrage antworten.

Um die Anfragen abzuarbeiten, wurde in der Software für Kern 0 das CGI-Modul entwickelt. Dieses enthält eine Funktion, die über das EFS im Webserver als Callback-Funktion für CGI-Anfragen eingetragen ist.

Im User-Interface gibt es grundsätzlich zwei Arten von Anfragen:

- Der Client sendet eine Anfrage um Daten zu erhalten (bspw. Messdaten)
- Der Client sendet eine Anfrage um Daten zu schicken (Konfiguration des Analyzers)

Um für jede Anfrage die richtige Aktion auszuführen, sind im CGI-Modul alle gültigen Anfragenamen in Form einer Struktur hinterlegt. Als Beispiel ist in Quellcode 6.2 ein Ausschnitt dieser Struktur zu sehen.

```
1 const CGI_sttyCOMMAND CGI_stCommand[CGI_COUNT] =
2 {
3   // KEY         | CMP length | VALUE angehaengt? | ENUM
4   {"stat_uptime"    , 11         , FALSE           , CGI_UPTIME    },
5   {"gen_freq_stop" , 13      , TRUE            , CGI_GEN_FREQ_STOP }
6 }
```

Quellcode 6.2: angelegte Struktur um POST-Anfragen abzuarbeiten

Die Strukur beinhaltet für jede Anfrage neben dem Namen, die Zeichenlänge des Namens, ein Flag, dass kennzeichnet, ob ein Wert angehängt ist und eine Nummer. Die Anfrage mit dem Namen „stat_uptime“ ist zum Beispiel für das Anfragen der Systemlaufzeit, so dass diese im User-Interface angezeigt werden kann. Ihr ist kein Wert angehängt, da hier vom Client Daten angefordert werden. Die Anfrage „gen_freq_stop“ hingegen enthält einen Wert, denn sie dient zum Setzen der Stopfrequenz des Generators.
Die Nummer, welche in Form eines Aufzählungstypen implementiert ist, dient der Identifizierung der Anfrage im weiteren Programmablauf. Dieser Programmablauf ist in Abbildung 6.12 dargestellt. Wird eine Anfrage erhalten, wird zuerst anhand der in der Struktur hinterlegten Namen verglichen, ob dies eine gültige Anfrage ist. Anschließend wird, wenn in der Struktur hinterlegt ist, dass diese Anfrage einen Wert enthält, dieser extrahiert und für die spätere Verarbeitung gespeichert. Anschließend erfolgt für jede Anfrage die entsprechende Aktion. Für die Anfrage der Systemlaufzeit ist dies beispielsweise das Errechnen der Laufzeit im Format hh:mm:ss aus der Laufzeit in Sekunden. Danach wird die Antwort an den Client gesendet und die Funktion beendet. Bei einer weiteren Anfrage wird die Funktion erneut aufgerufen.

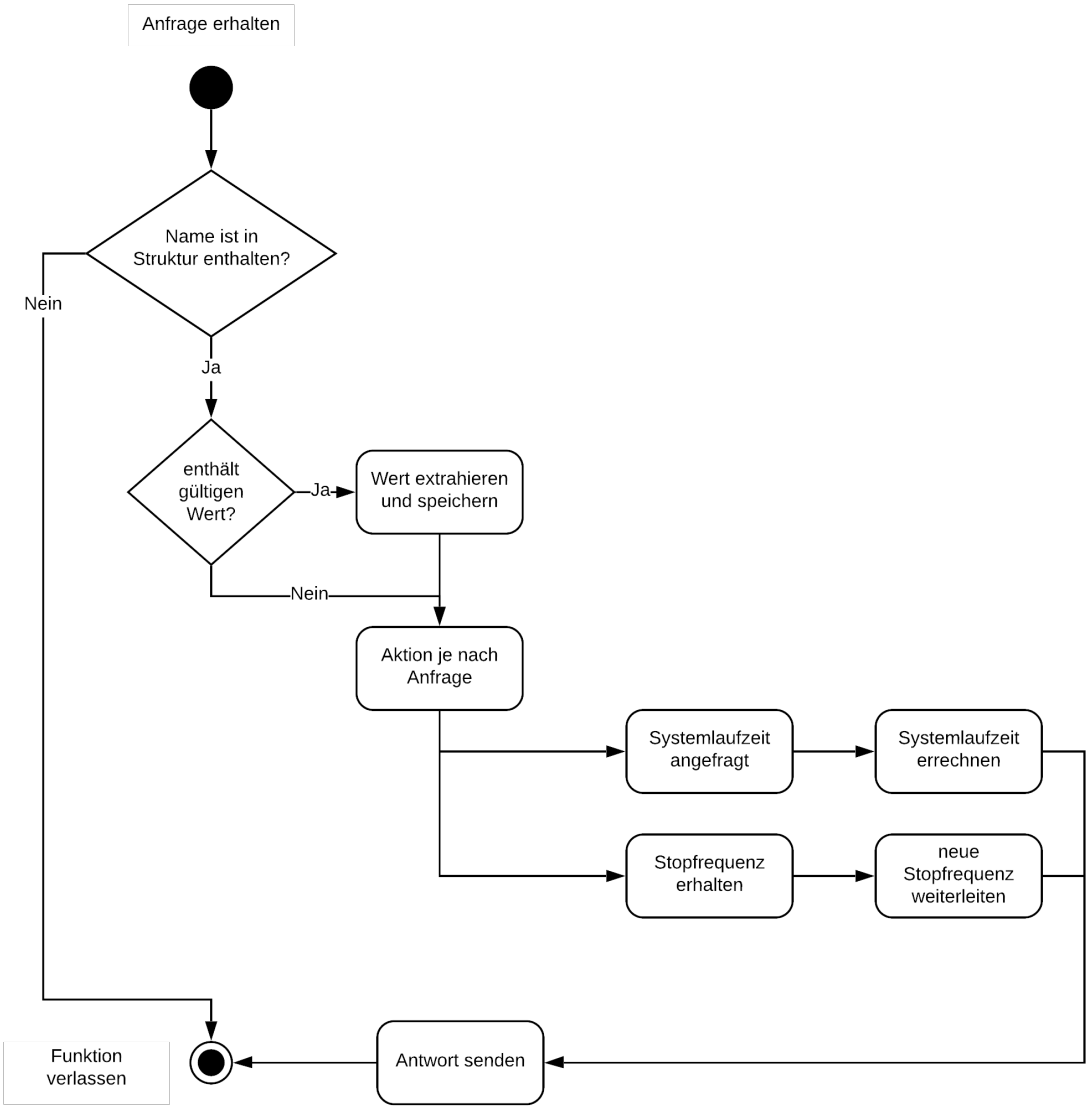


Abbildung 6.12.: Programmablauf einer CGI-Anfrage

6.3.9. Kommunikation mit dem D.Module2.ADDA500K16

Um die Signalverarbeitung im ADDA-Modul des Kern 1 durchzuführen, wurden die in diesem Kapitel erläuterten Schritte in der Software durchgeführt, um das D.Module2.ADDA500K16 zu konfigurieren und Daten auszulesen, bzw. zu schreiben.

6.3.9.1. Konfiguration des D.Module2.ADDA500K16

Um die A/D- und D/A-Wandler des D.Module2.ADDA500K16 zu nutzen, müssen diese zuerst konfiguriert werden. Die dafür notwendigen Schritte sind im User Guide des EVM [9] erklärt. Der daraus resultierende C-Code ist in Quellcode 6.3 gezeigt. Zusätzlich muss die Headerdatei für das D.Module2.ADDA500K16 eingebunden sein. In ihr sind die verfügbaren Register als Struktur hinterlegt und ermöglichen so den Zugriff auf diese Register.

```
1 ADDA500K16->clockdiv = ADDA500K16_CLKDIV(FSAMP);
2 ADDA500K16->cfgstat = ADDA500K16_CNVST_INT
3                     | ADDA500K16_LDAC_SYNC
4                     | ADDA500K16_INT_ADC0 + 16*DNUM
5                     | ADDA500K16_MUX_INP;
6 ADDA500K16->adc = ADDA_ADDA500K16_SETUP >> 16;
7 ADDA500K16->adc = (ADDA_ADDA500K16_SETUP & 0xFFFF);
```

Quellcode 6.3: Konfiguration des D.Module2.ADDA500K16

In der ersten Zeile des Quellcodes wird die Abtastfrequenz (FSAMP) gesetzt. Sie wurde mit 500 kHz auf den maximal möglichen Wert gesetzt. In den darauf folgenden Zeilen wird dann das Konfigurationsregister beschrieben und folgende Konfiguration festgelegt:

- Wandlung auf internen Takt triggern
- D/A Update synchron zum A/D
- Prozessorinterrupt festlegen (DNUM = Nummer des Kerns)
- Analogmultiplexer auf die Eingänge legen

In den letzten beiden Zeilen wird die erweiterte Konfiguration für den A/D-Wandler gesetzt. Die Konfiguration ist so gewählt worden, dass alle Eingangspaare (A - C) aktiviert sind und einen Eingangsspannungsbereich von +/- 2,5 V haben. Weitere Konfigurationsmöglichkeiten sind dem Datenblatt des D.Module2.ADDA500K16 [9] zu entnehmen.

6.3.9.2. Lesen des A/D-Wandlers und Schreiben des D/A-Wandlers

Nachdem die Konfiguration des D.Module2.ADDA500K16 wie in 6.3.9.1 beschrieben durchgeführt wurde, wird im Kern 1 des DSP ein HWI ausgelöst, wenn Daten des D/A-Wandlers verfügbar sind. Um darauf entsprechend zu reagieren, muss im SYS/BIOS für diesen HWI

ein Interrupthandler festgelegt werden. Dies wurde über das in Kapitel 6.2.3 beschriebene XGCONF durchgeführt.
Dieser Interrupthandler befindet sich ebenfalls im ADDA-Modul und liest, wenn er aufgerufen wird, die anliegenden Daten aus dem A/D-Wandler in einen temporären Speicher ein. Da in der Konfiguration festgelegt wurde, dass synchron zum Einlesen auch ein Schreiben des D/A-Wandlers durchgeführt werden soll, geschieht dies ebenfalls im Interrupthandler. Der dafür notwendige Quellcode ist in 6.4 zu sehen.

```
1  void ADDA_vAdcEventHandler ( void *handle)
2  {
3     uint8_t channel;
4     for (channel = 0; channel < 6; channel++)
5     {
6        adda_i16Input[channel] = ADDA500K16->adc;
7     }
8     // hier Ping-Pong Buffer fuellen
9     ADDA500K16->dac_a = adda_i16Output[0];
10    ADDA500K16->dac_b = adda_i16Output[1];
11 }
```

Quellcode 6.4: Konfiguration des D.Module2.ADDA500K16

Das Einlesen der Werte erfolgt für alle aktivierten Kanäle des A/D-Wandlers indem das Register entsprechend oft ausgelesen wird. Für die beiden D/A-Kanäle gibt es jeweils eigene Register, welche anschließend mit Werten beschrieben werden.
Um die Werte des A/D-Wandlers für die Weiterverarbeitung zu speichern, wurde ein Ping-Pong-Buffer eingerichtet. Dieser besteht aus zwei Datenarrays, die Speicherplatz für eine größere Anzahl Messwerte haben. Die Größe ist flexibel, so dass Messwerte bis zu einer Zeit von 100 ms gespeichert werden können. Der Programmablauf ist in Abbildung 6.13 dargestellt. Während der eine Buffer mit Daten beschrieben wird, kann der andere Buffer ausgelesen und mit den Daten gerechnet werden. Ist der aktuell beschriebene Buffer vollständig neu beschrieben worden, findet eine Umschaltung statt. Es wird nun aus dem gerade beschriebenen Buffer gelesen und der andere beschrieben. Um echtzeitfahig zu sein, steht für die Berechnung, die mit den Daten erfolgen soll, einem somit die Zeit zur Verfügung, die für das Füllen eines Buffers gebraucht wird. Ist der Buffer vollständig neu beschrieben worden, erfolgt das Setzen einer Semaphore, um die Task, in der die Berechnungen durchgeführt

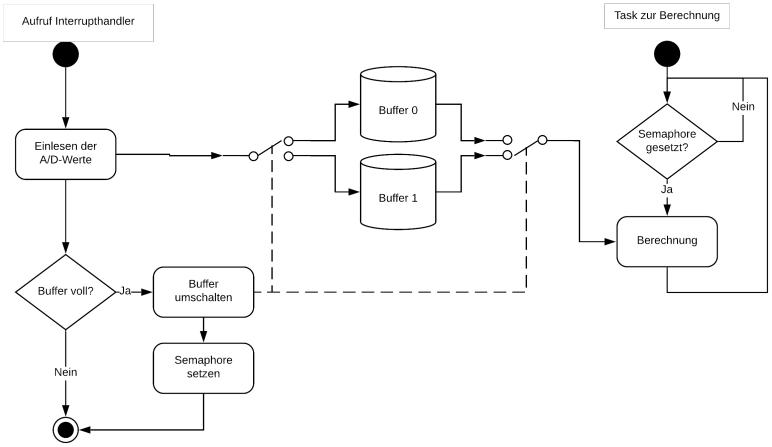


Abbildung 6.13.: Bufferstruktur der Signalverarbeitung

werden, auszuführen. Die Erläuterung der in dieser Task durchgeführten Berechnungen für die Signalverarbeitung erfolgt aufgrund des Umfangs in Kapitel 7.

6.3.10. Datentransfer der Messergebnisse

Um die Messergebnisse der Signalverarbeitung, die im Kern 1 stattfindet, im User-Interface anzuzeigen, sind mehrere Kommunikationsschritte erforderlich. Abbildung 6.14 zeigt den Datentransfer, wenn Daten der Frequenzgangmessung angefragt werden. Wird aus dem User-Interface heraus eine Messung gestartet, wird zyklisch über das CGI beim Kern 0 nachgefragt, ob Messdaten verfügbar sind. Dieser muss wiederum erst einmal über die MessageQ den Kern 1 fragen, ob Daten verfügbar sind. Da dies eine gewisse Zeit dauern kann und die CGI-Anfrage schnellstmöglich beantwortet werden muss, wird die erste Anfrage immer mit keinen Daten beantwortet. Währenddessen wurde die MessageQ im Kern 1 abgearbeitet und Messdaten aus dem Ringspeicher ausgelesen. Diese Messdaten werden über die MessageQ Antwort zum Kern 0 gesendet. Bei der nächsten CGI-Anfrage sind diese Daten verfügbar und können zum User-Interface gesendet und dargestellt werden. Kern 0 sendet wieder eine MessageQ, um bei der nächsten CGI-Anfrage neue Daten zur Verfügung zu haben.
Der Ringspeicher wird mit Daten aus der Signalverarbeitung gefüllt. Falls alle Daten, die in den Ringspeicher geschrieben worden ausgelesen sind und neue Messdaten noch nicht verfügbar sind, wird dies in der MessageQ Antwort des Kern 1 eingetragen. Das User-Interface fordert dann einfach weiter neue Daten an, bis diese wieder verfügbar sind.

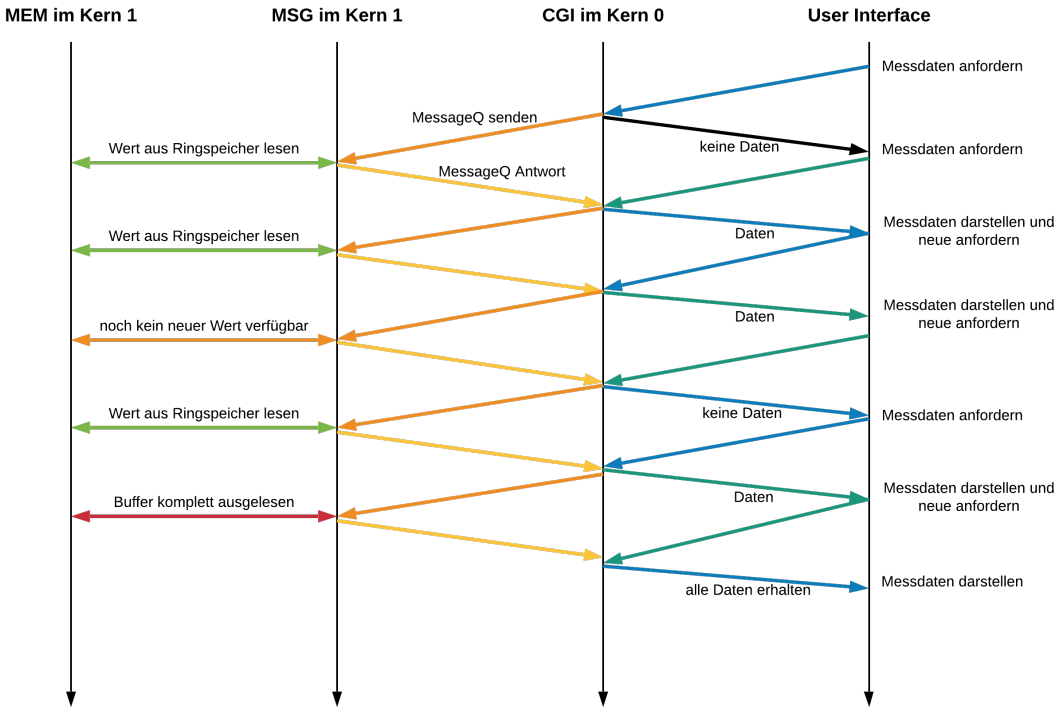


Abbildung 6.14.: Zeitlicher Ablauf eines gesicherten Datentransfers

Ist der Ringspeicher vollständig ausgelesen worden und die Signalverarbeitung abgeschlossen, wird dies in der Antwort auf die CGI-Anfrage eingetragen. Das User-Interface weiß somit, dass die Messung vollständig ist und fragt keine neuen Daten mehr an.

Wie zu erkennen ist, ergibt sich dadurch bei der Kommunikation eine Verzögerung. Diese Verzögerung lässt sich aber bei Nutzung der MessageQ für die Inter-Prozessor-Kommunikation nicht vermeiden. Der Vorteil dieser Variante ist, dass der Datenaustausch zwischen beiden Kernen gesichert abläuft und keine gleichzeitigen Zugriffe beider Kerne auf den gleichen Speicherbereich entstehen.
Nachteilig wirkt sich der Overhead aus, der durch die Nutzung der verschiedenen Kommunikationsprotokolle entsteht. Außerdem ist die Erweiterung der Daten um weitere Messwerte umständlich, da an mehreren Stellen im Programm Änderungen vorgenommen werden müssen.

Aufgrund der beschriebenen Nachteile wurde eine zweite Variante des Datentransfers implementiert. Hierbei wird ein Bereich des DDR3-Speichers genutzt, der aktuell nicht genutzt wird.

In diesem ist eine Struktur angelegt, die beiden Kernen bekannt ist. Kern 1 kann Messdaten dort abspeichern und Kern 0 kann diese direkt auslesen. Das IPC soll dazu auch Funktionen anbieten, die diese Methode sicher machen und gleichzeitige Speicherzugriffe verhindern. Da diese Implementierung jedoch zu einem fortgeschrittenen Zeitpunkt der Entwicklung durchgeführt wurde und die Dokumentation des IPC an dieser Stelle nicht einfach zu verstehen war, wurden die Speicherzugriffe direkt, also ungesichert durchgeführt. Dies ist aufgrund der Tatsache, dass ein Kern nur schreibt und der andere nur liest akzeptabel.
Die Kommunikation wie in Abbildung 6.14 dargestellt vereinfacht sich und ergibt den in Abbildung 6.15 zu sehenden zeitlichen Ablauf. 6.12 dargestellt. Das User-Interface fragt wieder

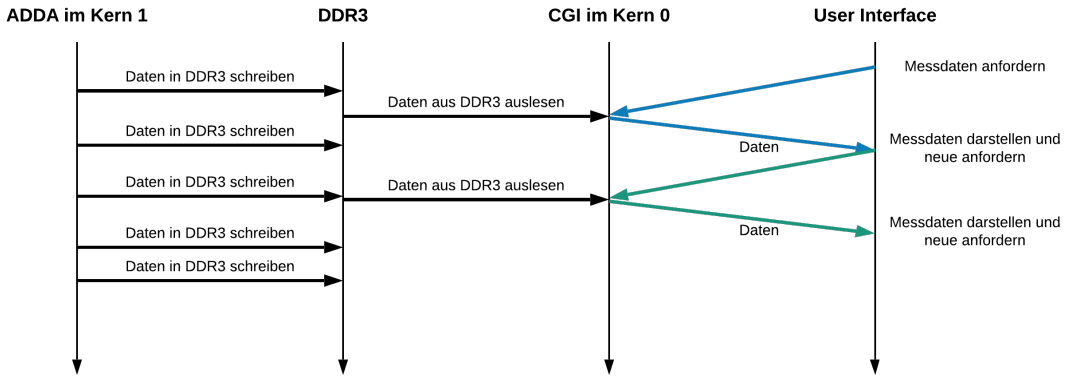


Abbildung 6.15.: Zeitlicher Ablauf eines Datentransfers mit DDR3

Daten über das CGI an. Diese Anfrage kann direkt mit Daten beantwortet werden, denn diese werden direkt aus dem DDR3 ausgelesen. Neue Messergebnisse des ADDA-Modul im Kern 1 werden von diesem sofort nachdem sie berechnet wurden in den DDR3 geschrieben. Sind keine neuen Messwerte verfügbar, stehen solange die letzten Messwerte im Speicher.

Beide Varianten werden in der Software genutzt. Die erste Variante dient zur Datenübertragung der Messergebnisse für eine Frequenzgangmessung. Denn hier gibt es mehrere Messwerte, die jeweils aus Frequenz, Amplitude und Phase bestehen und aufeinander folgend den Frequenzgang ergeben. Für die Anzeige im numerischen Display, die unabhängig von der Frequenzgangmessung erfolgt, werden nur die zuletzt errechneten Werte benötigt. Hier wird die zweite Variante genutzt.

6.4. Flashen des Entwicklungsmoduls

Um die Software des EVM zu flashen, ist eine spezielle Prozedur notwendig. Der DSP besitzt nur flüchtigen Speicher, so dass das Programm nach Abschalten der Spannungsversorgung nicht mehr verfügbar ist. Deshalb ist auf dem EVM ein nichtflüchtiger Speicher vorhanden, der über das D.Module2.BIOS beim Start des DSP das Programm aus diesem ausliest und in den flüchtigen Speicher lädt. Der nichtflüchtige Speicher ist ein NOR-Flash, der über eine Serial Peripheral Interface (SPI)-Schnittstelle angebunden ist. Für den Flash-Vorgang wird eine Konfigurationsdatei benötigt, die angibt, wo im nichtflüchtigen Speicher das Programm für Kern 0 und Kern 1 abgelegt ist. Diese ist ebenfalls im nichtflüchtigen Speicher abgelegt.

Der Flash-Vorgang beinhaltet somit drei Dateien, die in den NOR-Flash geschrieben werden müssen:

- Konfigurationsdatei
- Programm für den Kern 0
- Programm für den Kern 1

Um die Dateien in den NOR-Flash zu übertragen, wird die serielle Schnittstelle des EVM genutzt. Das auf dem EVM befindliche Setup Utility muss, wie im User Guide [13] beschrieben, gestartet werden und anschließend müssen alle Dateien hochgeladen werden.

6.5. Test der Software

Um die entwickelte Software zu testen, erfolgte zuerst die Überprüfung, ob alle Softwarekomponenten wie gewünscht arbeiten. Besonders wichtig ist hier die Echtzeitfähigkeit. Der Aufruf des Interrupthandlers zum Einlesen der Daten des ADC und Schreiben des DAC muss beendet sein, bevor der nächste Interrupt auftritt. Eine einfache Methode, dies zu überprüfen, ist das Setzen eines GPIO beim Aufruf des Interrupthandlers. Vor dem Verlassen wird dieser GPIO wieder zurückgesetzt. Mit einem Oszilloskop lässt sich so die Zeit für die Ausführung des Interrupthandlers ermitteln.

Der Versuch, einen GPIO innerhalb des Interrupthandlers zu setzen schlug jedoch fehl. Wurde die Funktion zum Setzen eines GPIO in den Programmcode eingefügt, wurde der Interrupthandler nicht oder nur gelegentlich ausgeführt. Die Gründe dafür sind nicht weiter analysiert worden. In anderen Threads funktionierte der Aufruf der GPIO-Funktionen problemlos.
Deshalb wurde hier eine andere Variante gewählt, um die Ausführungsdauer des Interrupthandlers zu ermitteln. Bei dessen Aufruf wurde der aktuelle Zeitstempel des Prozessors abgespeichert. Dieser wird intern mit dem CPU-Takt inkrementiert. Am Ende des Interrupthandlers wird dieser Wert erneut ausgelesen, und die Differenz beider Werte ergibt die Ausführungsdauer als Angabe in Taktzyklen. Bei einer Abtastrate von 500 kHz und einem CPU-Takt von 1,25 GHz darf die Ausführung des Interrupthandler nicht länger als 2500 Zyklen. Wiederholte Messungen ergaben hier einen Mittelwert von ca. 500 Zyklen. Die Auslastung der CPU durch das Abarbeiten des Interrupts beträgt somit ca. 20 Prozent.

Die Werte des ADC werden in einem Ping-Pong-Buffer abgelegt und in einem anderen Thread berechnet. Je mehr Werte in diesem Buffer gespeichert werden können, bevor dieser voll ist, desto mehr Zeit bleibt für die Berechnung. In der Software kann die Buffergröße so gewählt werden, dass ein Zeitfenster zwischen 25 ms und 100 ms für die Berechnung zur Verfügung steht. In Tabelle 6.2 ist das Ergebnis der Messung für die Berechnungsdauer zu sehen. Die Dauer der Berechnung wurde über das Setzen eines GPIOs und Messung mit einem Oszilloskop bestimmt. Wie zu erkennen ist, beträgt sie unabhängig von der Messdauer immer um die 80 Prozent. Um weitere Berechnungen durchzuführen sollten deshalb die bestehenden Messungen auf ihre Ausführungsdauer hin optimiert werden.

Neben der Signalverarbeitung erfolgte auch eine Überprüfung der CGI-Anfragen. Diese werden vom User-Interface aus gesendet und müssen zeitnah abgearbeitet werden. Um dies

Tabelle 6.2.: Dauer der Signalanalyse des Buffers

Messzeit [ms]	Dauer der Berechnung[ms]	Dauer / verfügbare Zeit [%]
25	20	80
50	41	82
100	79	79

zu überprüfen, wurde wie auch bei der vorangegangenen Messung ein GPIO-Pin und ein Oszilloskop für die Messung der Ausführungszeit genutzt. Das Ergebnis dieser Messung ist in Abbildung 6.16 zu sehen. Das Oszilloskop ist im „Persistance-Mode“ genutzt worden, so dass alle Ausführungszeiten über den Messzeitraum in einem Bild zu sehen sind. Gemessen wurde für einen Zeitraum von ca. 2 Sekunden. Wie zu erkennen ist, dauert die Beantwortung

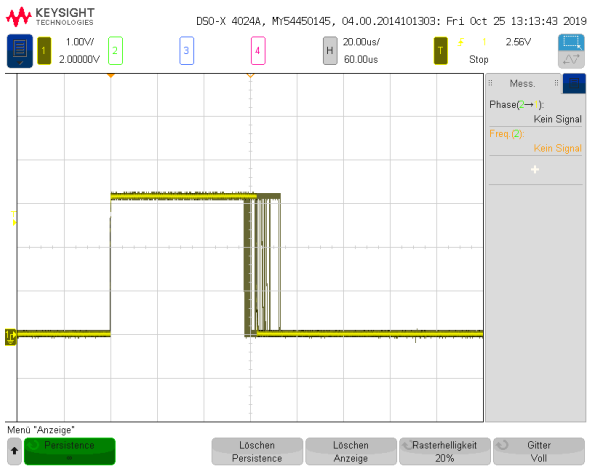


Abbildung 6.16.: Dauer der Beantwortung einer CGI-Anfrage

einer Anfrage über das CGI nicht länger als ca. 75 µs.

Zuletzt wurde die Kommunikation mit der MessageQ und des IPC-Moduls überprüft. Hier kam auch wieder die Messung eines GPIO-Pins in Verbindung mit einem Oszilloskop zum Einsatz. Der GPIO wurde gesetzt, wenn im Kern 0 eine MessageQ gesendet werden soll und zurückgesetzt, wenn die Antwort von Kern 1 gesendet und im Kern 0 empfangen wurde. Abbildung 6.17 zeigt diese Messung. Wie zu erkennen ist, werden einige MessageQs sehr schnell beantwortet (< 5ms) und einige werden erst nach ca. 50 ms vollständig abgearbeitet. Dies ist darauf zurückzuführen, dass die Beantwortung der MessageQ in einer Task geschieht.

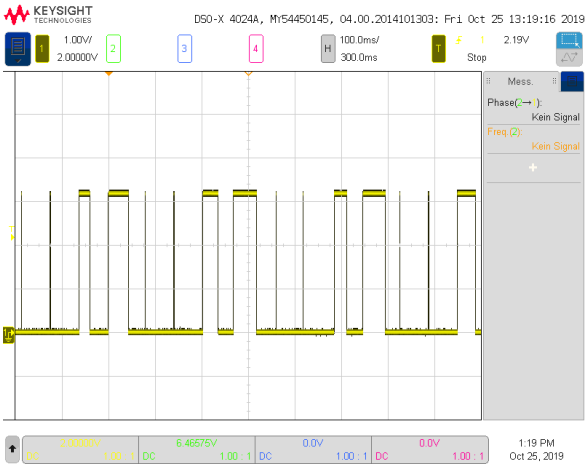


Abbildung 6.17.: Dauer einer MessageQ

Diese ist niedriger priorisiert, als z.B die Beantwortung von CGI-Anfragen im Kern 0 oder die Durchführung der Signalverarbeitung im Kern 1.

7. Signalverarbeitung

7.1. Planung

Um mit dem EVM einen Audio-Analyzer zu realisieren, müssen verschiedene Signalverarbeitungsalgorithmen in der Software implementiert werden. Dazu wurde zuerst eine Anforderung erstellt, welche Funktionen in der Software realisiert werden sollen:

- Signalgenerator mit einstellbarer Frequenz und Amplitude
- Effektivwertmessung (RMS) von Signalen
- Frequenzmessung
- Bestimmung der Phase von zwei Signalen

Es muss ein Signalgenerator vorhanden sein, der in der Lage ist, ein Sinussignal mit einstellbarer Frequenz und Amplitude zu erzeugen. Für die Bestimmung des Amplitudengangs eines DUT muss dessen Ausgangssignal mit einem Referenzsignal ins Verhältnis gesetzt werden. Dazu wird eine Effektivwertmessung des Signals genutzt. Des Weiteren muss die Signalfrequenz und die Phasenverschiebung zwischen zwei Signalen gemessen werden können. In den folgenden Kapiteln wird erklärt, wie die erforderlichen Berechnungen in der Software umgesetzt wurden.

7.2. Parametrierung

Für die Signalgenerierung und die Analyse der eingelesenen Signale, können verschiedene Parameter festgelegt werden. Diese sind im ADDA-Modul, jeweils für die Konfiguration des Generators und des Analyzers, hinterlegt.

Für den Generator gibt es folgende Parameter, die über das User-Interface angepasst werden können:

- Startfrequenz
- Stopfrequenz
- Anzahl der Frequenzschritte
- Aufteilung der Frequenzschritte (linear / logarithmisch)
- Ausgangsamplitude
- Wahl des Ausgangs

Diese Parameter bestimmen die Start- und Stopfrequenz des Signalgenerators. Die Anzahl der Frequenzschritte gibt an, wie viele Frequenzen zwischen diesen beiden Werten erzeugt werden sollen. Die Aufteilung der Frequenzschritte kann logarithmisch oder linear erfolgen. Bei der Wahl des Ausgangs kann gewählt werden, an welchen Ausgängen das Signal ausgegeben werden soll. Es kann auf beiden Ausgängen gleichzeitig oder nur auf einem Ausgang ausgegeben werden. Zusätzlich kann die Ausgabe auch als differenzielles Signal erfolgen, indem die Signalamplitude pro Ausgang halbiert und auf beiden Ausgängen ausgegeben wird. Dabei wird ein Ausgang invertiert, um durch Subtraktion des einen Kanals mit dem anderen die eingestellte Signalamplitude zu erhalten.

Für den Analyzer können folgende Parameter im User-Interface festgelegt werden:

- Wahl des Referenzkanals
- Wahl des Messkanals
- Wartezeit
- Messzeit
- Kopplung des Eingangs (AC / DC)

Die Parametrierung des Analyzers erlaubt die Auswahl von Referenz- und Messsignal. Als Quelle kann aus jedem Eingang und den beiden Generatorausgängen gewählt werden.
Die Wartezeit gibt an, wie lange eine Frequenz gesendet wird, bevor eine Messung bei dieser

Frequenz durchgeführt wird. Die anschließende Messung erfolgt für die in der Messzeit angegebene Zeit. Als letztes kann gewählt werden, ob das Eingangssignal gleichspannungsfrei sein soll (AC), oder auch DC-Anteile gemessen werden sollen.

7.3. Steuerung der Signalverarbeitung

Die Signalverarbeitung findet, wie im Kapitel 6.3.4 beschrieben, im ADDA-Modul des Kern 1 statt. Ist ein neuer Datensatz im Ping-Pong-Buffer verfügbar, wird eine Semaphore freigegeben und damit eine Task aufgeweckt. Diese Task führt dann eine Signalanalyse mit dem Buffer durch. Dieser beinhaltet je nach gewählter Messzeit, eine feste Anzahl an Messwerten für den Referenz- und den Messkanal. Für beide Kanäle erfolgt die Berechnung des Signalpegels und eine Frequenzmessung. Zusätzlich wird die Phasenverschiebung zwischen beiden Kanälen gemessen.
Die Signalerzeugung und die Zuordnung der Messdaten wird von einer Task gesteuert, welche aufgerufen wird, wenn der Ping-Pong-Buffer neu beschrieben wurde.
Beim Start des DSP wird in dieser Task zusätzlich einmalig eine Kalibrierung des ADC durchgeführt. Dazu wird der Eingang des ADC auf die Signalmasse gelegt und anschließend der Mittelwert für jeden Kanal über einen Zeitraum von zwei Sekunden gebildet. Dieser Mittelwert stellt den DC-Offset des ADC dar und wird anschließend nach dem Einlesen des AD-Wertes für jeden Kanal abgezogen.

Nach der Kalibrierung übernimmt die Task die Steuerung der Frequenzgangmessung. Ist diese nicht aktiv, wird kein Frequenzsweep am Generatorausgang erzeugt und alle Messungen erfolgen lediglich anhand der eingestellten Messzeit. In Abbildung 7.1 ist der Ablauf der Signalverarbeitung dargestellt. Die Signalanalyse erfolgt fortlaufend während der gesamten Systemlaufzeit. Die Ergebnisse der Messungen werden im numerischen Display angezeigt. Soll eine Frequenzgangmessung durchgeführt werden, wird der rechte Handlungstrang ausgeführt. In ihm wird zuerst, abhängig von der gewählten Start- und Stopfrequenz und anhand der Anzahl der Frequenzschritte die Schrittweite für den Frequenzsweep errechnet. Anschließend erfolgt der Programmablauf für alle zu erzeugenden Frequenzen. Der Signalgenerator wird mit der Frequenz initialisiert und anschließend die vorgegebene Wartezeit gewartet. Wenn diese Zeit abgelaufen ist, erfolgt der Start einer Messung für die Dauer der eingestellten Messzeit. Ist die Messung abgeschlossen, wird das Ergebnis im Ringspeicher des MEM-Moduls abgelegt.

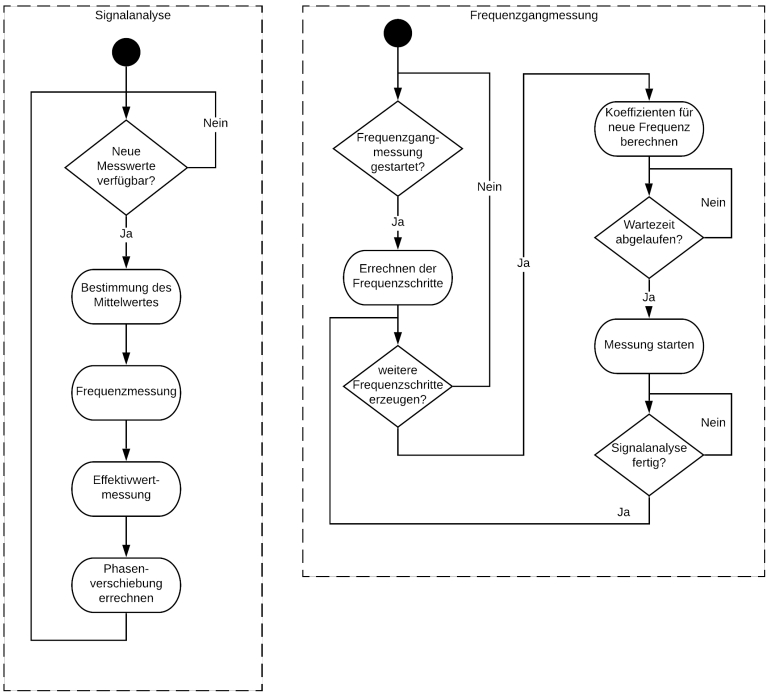


Abbildung 7.1.: Ablauf der Signalverarbeitung

Anschließend beginnt der Ablauf von vorne und eine neue Frequenz wird erzeugt, oder, wenn alle Frequenzen erzeugt wurden, wird auf den Start einer neuen Messung gewartet.

7.4. Signalgenerator

Der Signalgenerator basiert auf einem oszillierenden Filter, wie in [18] beschrieben. Dieses basiert auf einem IIR-Filter 2. Grades, bei dem die Koeffizienten so gewählt sind, dass es oszilliert. Das Signalflussdiagramm dazu ist in Abbildung 7.2 zu sehen. Wie zu erkennen ist, wird das Filter mit einem Dirac-Stoß angeregt. Die Koeffizienten können so gewählt werden, dass am Ausgang eine beliebige Frequenz erzeugt wird. Die Formel für die Berechnung des

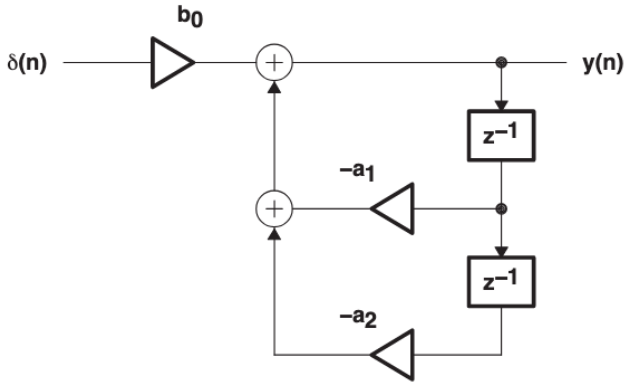


Abbildung 7.2.: Signalflussdiagramm des Oszillators

Ausgangs lautet wie folgt:

$$y[n] = \delta[n] * b_0 - a_1 * y[n-1] - a_2 * y[n-2] \tag{7.1}$$

wobei für die Berechnung der Koeffizienten gilt

$$a_1 = -2 * cos(\omega_0),\ a_2 = 1,\ b_0 = sin(\omega_0) \text{ mit } \omega_0 = \frac{2*\pi*f_s}{f_A}$$

Die Frequenz f_s gibt dabei die Frequenz an, bei welcher der Oszillator schwingt. f_A ist die Frequenz, mit der die Ausgangswerte $y[n]$ generiert werden. Der Dirac-Stoß $\delta[n]$ ist folgendermaßen definiert:

$$\delta[n] = \begin{cases} 1 & wenn\, n = 0 \\ 0 & sonst \end{cases}$$

Damit ergibt sich für die Berechnung des ersten Ausgangswertes:

$$y[n] = 1 * sin(\omega_0) - (-2 * cos(\omega_0)) * y[n-1] - 1 * y[n-2]$$
$$y[n] = sin(\omega_0) + 2 * cos(\omega_0) * y[n-1] - y[n-2]$$

für alle nachfolgenden Ausgangswerte gilt folgendes:

$$y[n] = 2 * cos(\omega_0) * y[n-1] - y[n-2]$$

7.5. Frequenzmessung

Die Analyse der Signalfrequenz ist durch die Messung von Nulldurchgängen des im Ping-Pong-Buffer befindlichen Signals realisiert. Als Nulldurchgang wird der Zeitpunkt erkannt, bei dem das Signal die Nulllinie vom Negativen ins Positive passiert. Für die Frequenzmessung werden mindestens zwei Nulldurchgänge benötigt. Sind diese erkannt worden, kann aus der Zeitdifferenz der Nulldurchgänge die Signalfrequenz mit der Formel 7.2 bestimmt werden:

$$f = \frac{1}{T_{diff}} \tag{7.2}$$

Da es sich um ein abgetastetes Signal handelt, kann T_{diff} nur ein Vielfaches der Abtastrate T_A sein. Diese errechnet sich aus dem Kehrwert der Abtastfrequenz f_A. Bei der gewählten Abtastfrequenz von $500kHz$ sind dies $2\mu s$.
Die entstehende Ungenauigkeit in der Bestimmung der Signalfrequenz ist abhängig von der zu messenden Frequenz. Angenommen der Nulldurchgang wird einen Abtastwert zu spät erkannt und T_{diff} ist somit $2\mu s$ zu groß. Während sich bei einer Signalfrequenz von $100Hz$ lediglich ein Fehler von $0,02Hz$ ergibt, entsteht bei einer Signalfrequenz von $50kHz$ ein Fehler von ca. $5,5kHz$. Um diese steigende Ungenauigkeit bei steigender Signalfrequenz zu minimieren, wurde die Nulldurchgangsmessung erweitert. Werden anstatt zwei Nulldurchgängen mehrere erfasst, lässt sich T_{diff} wie folgt errechnen:

$$T_{diff} = \frac{T_{Stop} - T_{Start}}{N_{Null} - 1} \tag{7.3}$$

T_{Start} und T_{Stop} sind die Zeiten des ersten und letzten Nulldurchgangs. N_{Null} ist die Anzahl an Nulldurchgängen, die insgesamt erfasst wurden.

Wurden bei einer Messzeit von $20ms$, für eine Signalfrequenz von $50kHz$, nun 1000 Nulldurchgängen erkannt, sollte T_{diff} einen Wert von genau $20ms$ haben. Ist dieser Wert nun fälschlicherweise um einen Abtastwert falsch und beträgt $19,998ms$, ergibt sich für f durch Einsetzen der Formel 7.3 in 7.2 und umformen:

$$f = \frac{N_{Null} - 1}{T_{Stop} - T_{Start}} = \frac{1000}{19,998ms} = 50kHz \tag{7.4}$$

Die Frequenzmessung kann somit für hohe Frequenzen deutlich verbessert werden.

7.6. Effektivwertmessung

Der Effektivwert eines Signals ist definiert als der Wert, den eine Gleichspannung haben muss, um dieselbe elektrische Leistung in einem ohmschen Widerstand umzusetzen. Wenn es sich bei dem Signal um ein periodisches Signal handelt, erfolgt die Berechnung für die Periodendauer T. Die Berechnung des Effektivwerts eines diskreten periodischen Signals, erfolgt nach folgender Formel:

$$U_{eff} = \sqrt{\frac{1}{n}(\sum_{i=0}^{n} u[i]^2)} \tag{7.5}$$

wobei n die Anzahl an Abtastwerten pro Periode ist. Dieser Wert ist aus der vorangegangenen Frequenzmessung bereits bekannt.
Da auch hier mit steigender Signalfrequenz eine erhöhte Ungenauigkeit entsteht, da bei höheren Frequenzen weniger Abtastwerte pro Periode vorhanden sind, erfolgte die Messung ebenfalls für die Zeit zwischen erstem und letztem Nulldurchgang.

7.6.1. Phasenmessung

Die Phasenmessung erfolgt ebenfalls anhand der ermittelten Nulldurchgänge. Hier wird die Differenz zwischen dem Nulldurchgang des Referenzsignals T_{Ref} und dem Nulldurchgang des Messignals T_{Mess} ermittelt, und anschließend kann bei bekannter Signalfrequenz f_s die Phasenverschiebung ϕ wie folgt errechnet werden:

$$\phi = \frac{T_{Ref} - T_{Mess}}{T_{Freq}} * 360^\circ \tag{7.6}$$

wobei gilt

$$T_{Freq} = \frac{1}{f_s}$$

Um auch hier die Genauigkeit der Messung zu erhöhen, wurden die Differenzen aller Nulldurchgänge addiert und anschließend durch die Anzahl geteilt.

8. Der fertige Audio-Analyzers

Nachdem die Entwicklung der notwendigen Hard- und Software abgeschlossen wurde, konnte der fertige Audio-Analyzer in Betrieb genommen und getestet werden. In diesem Kapitel wird beschrieben, wie die Inbetriebnahme durchgeführt wurde und wie zukünftig Messungen mit dem Audio-Analyzer durchgeführt werden können. Anschließend erfolgte die Überprüfung der Funktionalität des Audio-Analyzers, indem verschiedene Messungen durchgeführt wurden.

8.1. Test

8.1.1. Numerisches Display

Das numerische Display liefert kontinuierlich Messergebnisse für den ausgewählten Mess- und Referenzkanal. Um diese Messergebnisse zu überprüfen, wurden mit einem Frequenzgenerator zwei Sinusschwingungen erzeugt und in den Audio-Analyzer eingespeist. Die

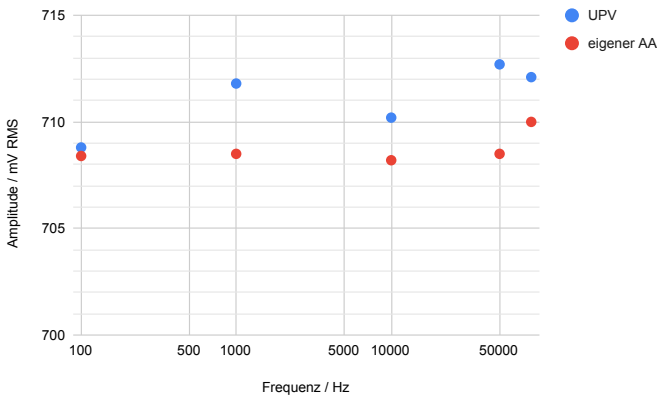


Abbildung 8.1.: Test des numerischen Displays

Überprüfung erfolgte mit dem UPV, dessen Messergebnisse als Referenz verwendet wurden. Abbildung 8.1 zeigt die Ergebnisse dieser Messung. Mit dem Frequenzgenerator wurden verschiedene Frequenzen von 20 Hz bis 100 kHz bei einer Amplitude von $2V_{PP}$ (0,707 V_{RMS}) erzeugt. Die Messung der Frequenzen ergab beim UPV sowie beim eigenen Audio-Analyzer gleich gute Ergebnisse. Die Amplitude wurde ebenfalls mit einer maximalen Abweichung von 5 mV (bei 50 kHz) sehr genau gemessen.

8.1.2. Frequenzgangmessung

Um die Frequenzgangmessung zu überprüfen, wurden Frequenzgänge von mehreren DUTs ermittelt. Diese wurden ebenfalls mit dem UPV gemessen, um eine Aussage über die Qualität der Messung mit dem Audio-Analyzer treffen zu können.
Als erstes DUT wurde ein Benchmaster 8 der Firma „Kemo Electronic Filters“ genutzt. Dieses besteht aus zwei einstellbaren Filtern, die hintereinander geschaltet werden können. Es wurde so konfiguriert, dass sich eine Reihenschaltung aus einem 5 kHz Hochpass und einem 40 kHz Tiefpass ergibt.
Die Messung der Filterschaltung mit dem Audio-Analyzer hat den in Abbildung 8.2 dargestellten Frequenzgang ergeben. Die Referenzmessung des UPV ist in Abbildung 8.3 dargestellt.

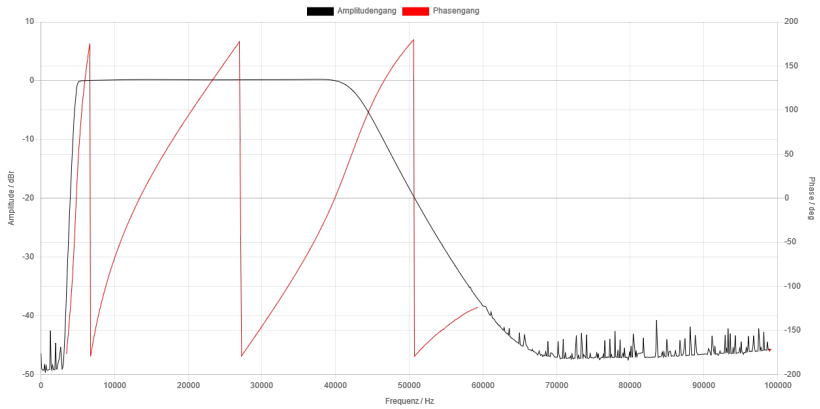


Abbildung 8.2.: Frequenzgangmessung einer Filterschaltung mit dem Audio-Analyzer

Die Messung zeigt, dass die Frequenzgangmessung mit dem eigenen Audio-Analyzer den erwarteten Verlauf hat. Der Hoch- sowie der Tiefpass sind zu erkennen und der Phasenverlauf ist stetig. Die durchgeführte Messung mit dem UPV bestätigt das gemessene Ergebnis. Amplituden- und Phasenverlauf sind im Durchlassbereich weitestgehend identlisch. Lediglich

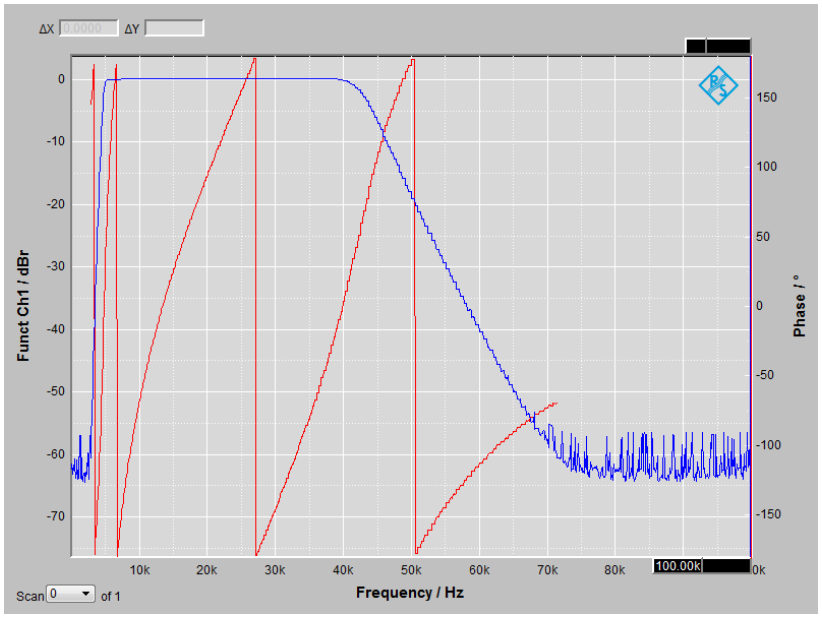


Abbildung 8.3.: Referenzmessung der Filterschaltung mit dem UPV

die messbare Dämpfung im Sperrbereich der Filterschaltung ist bei der Messung mit dem UPV besser. Während mit dem eigenen Audio-Analyzer hier maximal -48dB gemessen werden, erreicht der UPV ca. -60dB.
Die Gründe dafür sind einerseits, dass der maximale Ausgangspegel des eigenen Audio-Analyzers aufgrund des Ausgangsfilters nur $+-1,25V_{PP}$ beträgt und andererseits, das der UPV eine automatische Skalierung des Eingangs integriert hat. Die Skalierung besteht aus einer Schaltung, die je nach Eingangssignal dieses so skaliert, dass der nachgeschaltete ADC immer optimal ausgesteuert wird. Die analogen Schaltungen des eigenen Audio-Analyzers bieten diese Möglichkeit nicht, sodass mit einem festen Spannungsbereich gemessen wird.

Wird diese automatische Skalierung deaktiviert und der Eingangsspannungsbereich des UPV auf einen festen Wert gelegt, zeigt sich auch hier eine Verschlechterung der maximal messbaren Sperrdämpfung. In Abbildung 8.4 ist eine Messung mit dem UPV zu sehen, bei der die Eingangsskalierung fest auf 3V gesetzt wurde. Zu erkennen ist, dass die maximal messbare Dämpfung sich im Vergleich zur Messung mit automatischer Skalierung um ca. 15 dB verschlechtert hat.

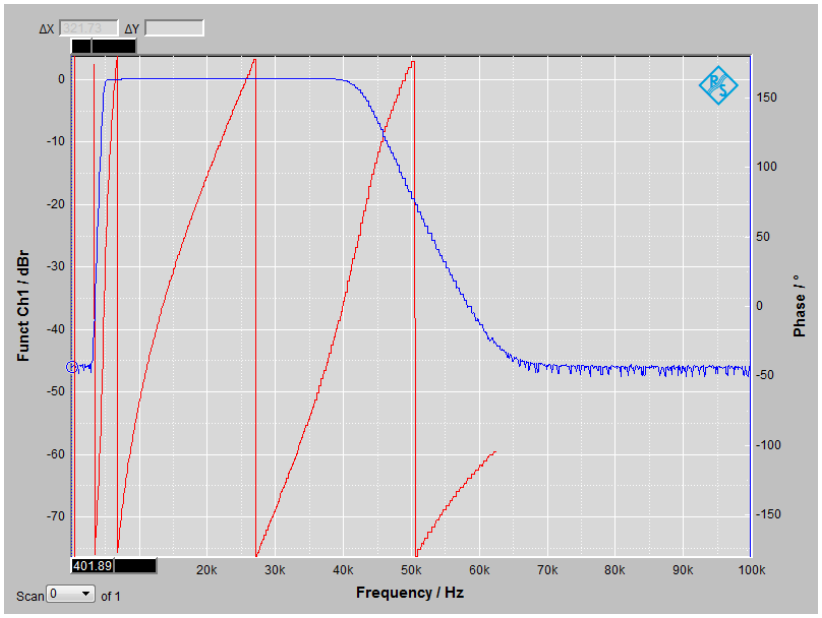


Abbildung 8.4.: Messung der Filterschaltung mit dem UPV und fester Eingangsskalierung

8.2. Inbetriebnahme

Um den Audio-Analyzer in Betrieb zu nehmen, wird zusätzlich ein Computer benötigt. Dieser muss über folgendes verfügen:

- Ethernet-Anschluss
- USB-Schnittstelle
- Webbrowser
- Terminalprogramm

Als Webbrowser wird „Mozilla Firefox“ empfohlen. Mit diesem wurde die Bedienung des Audio-Analyzers über das User-Interfaces während der Entwicklung getestet. Als Terminalprogramm wird „teraTerm“ empfohlen. Dieses wurde in dieser Arbeit zur Kommunikation mit dem Audio-Analyzer genutzt.

Als Erstes muss die Spannungsversorgung des Audio-Analyzers hergestellt werden. Der Audio-Analyzer benötigt zwei Spannungsversorgungen (5 V und 3,3 V), die von zwei Steckernetzteilen über Hohlsteckerbuchsen an der Rückseite des Gehäuses eingespeist werden. Diese müssen entsprechend der Beschriftung an den Hohlsteckerbuchsen angeschlossen

werden. Die grüne LED an der Frontplatte leuchtet daraufhin dauerhaft und zeigt an, dass die Software gestartet ist und die gelbe LED blinkt. Die rote LED darf nicht leuchten, da ansonsten ein Fehler vorliegt.

Anschließend muss der Audio-Analyzer per USB mit dem Computer verbunden werden. Der Audio-Analyzer erscheint dort als serielle Schnittstelle. Mit einem Terminalprogramm kann dann eine Verbindung hergestellt werden. Die Verbindungseinstellungen sind wie folgt zu wählen:

- Baudrate: 115200
- Datenbits: 8
- Stopbit: 1
- Parität: keine (N)

Diese stehen auch zusätzlich auf der Rückseite des Gehäuses, oberhalb des USB-Anschlusses.

Nachdem eine Verbindung mit dem Terminalprogramm hergestellt wurde, muss der Audio-Analyzer mit dem Netzwerk verbunden werden. Dies erfolgt über den Netzwerkanschluss an der Rückseite des Audio-Analyzers. Dieses Netzwerk muss über einen DHCP-Server verfügen, der dem Audio-Analyzer eine IP-Adresse zuweist. Ist kein DHCP-Server vorhanden, bekommt der Audio-Analyzer keine IP-Adresse und ist somit nicht erreichbar.
Wenn eine IP-Adresse zugewiesen wurde, leuchtet die gelbe LED dauerhaft und im Terminalprogramm erscheint die in Abbildung 8.5 zu sehende Ausgabe. Der Link Status gibt an, mit

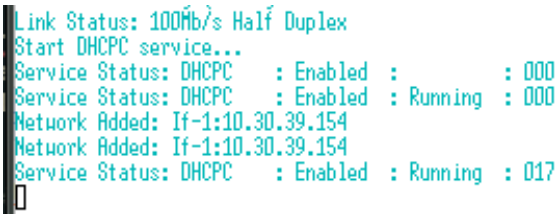


Abbildung 8.5.: Terminalausgabe bei Netzwerkverbindung

was für einem Netzwerk sich der Audio-Analyzer verbunden hat. Maximal sind hier 1000Mbit/s möglich. Die zugewiesene IP-Adresse ist in der Zeile „Network Added:“ zu finden. Sie lautet hier 10.30.39.154 und kann je nach Netzwerk unterschiedlich sein. Sie wird benötigt, um über den Browser des Computers auf das User-Interface zuzugreifen. Ist der DHCP-Server so

eingestellt, dass Geräten immer wieder dieselbe IP-Adresse zugewiesen wird, kann auf die USB-Verbindung und den Start des Terminalprogramms nach dem ersten Start verzichtet werden. Sie dient ausschließlich dazu, die IP-Adresse zu erfahren. Der Status der Netzwerkverbindung kann auch über die gelbe LED an der Frontplatte erkannt werden. Diese leuchtet bei bestehender Verbindung durchgehend.

Nachdem der Audio-Analyzer im Netzwerk erreichbar ist und die IP-Adresse bekannt ist, kann diese im Webbrowser des Computers eingegeben werden, um das User-Interface aufzurufen. Voraussetzung dafür ist natürlich, dass sich Computer und Audio-Analyzer im selben Netzwerk befinden und sich erreichen können.
Abbildung 8.6 zeigt den Aufruf des User-Interfaces im Webbrowser. Als erstes Fenster wird das „Get Started“ angezeigt. Hier wird kurz die Benutzung des Audio-Analyzers erklärt.

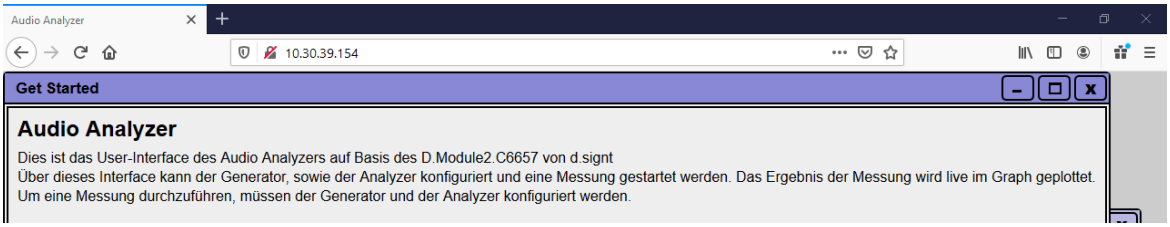


Abbildung 8.6.: Erster Aufruf des User-Interfaces im Browser

Um die anderen Fenster des User-Interfaces zu sehen, kann das „Get Started“ Fenster geschlossen oder minimiert werden. Anschließend sind alle weiteren Fenster, dargestellt in Abbildung 8.7, sichtbar. Das numerische Display kann aktiviert werden, indem in dessen Fenster das Kästchen oben links angeklickt wird. Anschließend werden dort die Messergebnisse des Referenz- und des Messkanals angezeigt und alle 100 ms erneuert. Die Auswahl dieser Kanäle erfolgt im Fenster „Analyzer“. Hier kann zusätzlich die Eingangskopplung gewählt und die Warte- und Messzeit eingestellt werden. Die Auswahl einer Eingangsbandbreite ist nicht möglich, da diese nicht in der Software implementiert ist.

Um eine Frequenzgangmessung zu starten muss zusätzlich der Generator im Fenster „Generator“ parametriert werden. Hier muss eine Auswahl der Ausgänge, an denen das Signal erzeugt werden soll, erfolgen. Die Signalform ist immer ein Sinussignal. Weiterhin können die Start- und Stopfrequenzen sowie die Frequenzschritte dazwischen eingegeben werden. Die Skalierung kann dabei logarithmisch oder linear eingestellt werden. Eine Wahl der Bandbreite

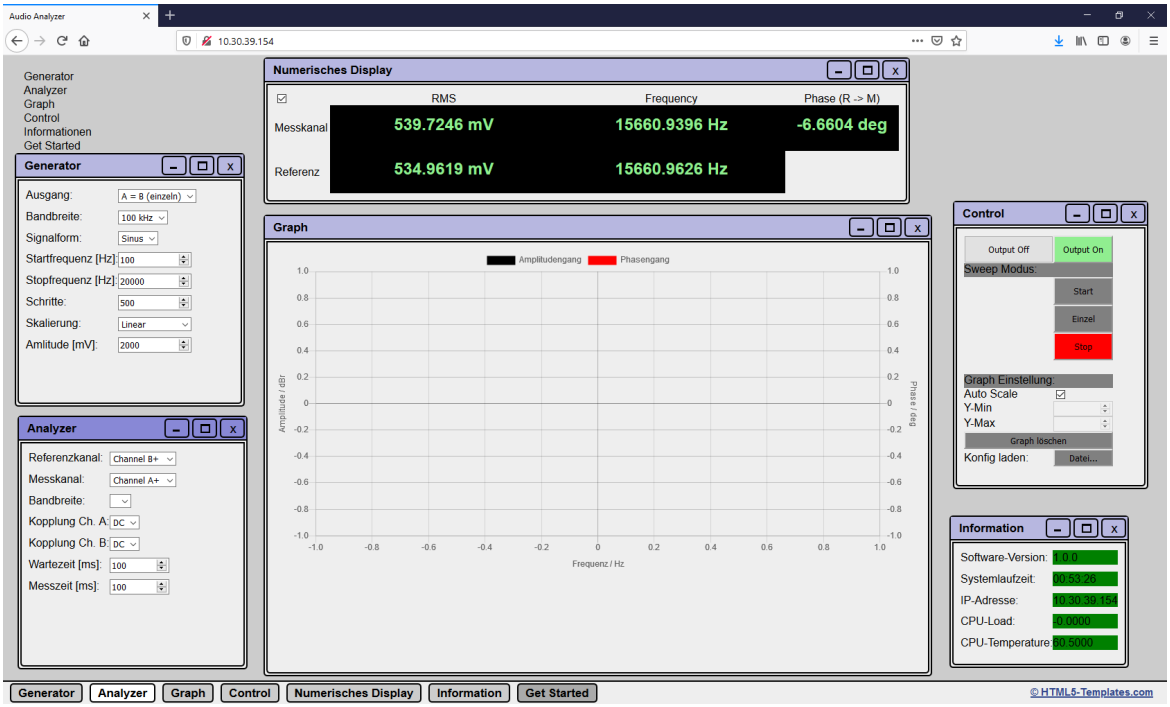


Abbildung 8.7.: Aufbau des User-Interfaces im Browser

ist auch hier nicht möglich und fest auf 100 kHz eingestellt, da dies nicht implementiert wurde. Als letzten Parameter kann die Ausgangsamplitude gewählt werden.

Sind alle Einstellungen gemacht worden, kann die Frequenzgangmessung über das Fenster „Control“ gestartet werden. Hierzu muss der Button „Einzel“ einmal gedrückt werden. Anschließend wird im „Graph“-Fenster der Frequenzgang in Amplitude und Phase dargestellt. Der Fortschritt der Messung ist an einem grünen Balken, unterhalb der Buttons zu erkennen. Soll die Messung vorzeitig abgebrochen werden, kann dies mit einem Klick auf den Button „Stop“ erfolgen.
Das Abspeichern des aufgenommenen Frequenzgangs als Bild, erfolgt über einen Rechtsklick auf den Graphen. Soll die Amplitude oder die Phase nicht angezeigt werden, kann eine Deaktivierung durch Klick auf den Namen in der Legende des Graphen erfolgen. Das Löschen der Daten des Graphen erfolgt über den Button „Graph löschen“.

9. Diskussion und Fazit

Die Aufgabe, einen Audio-Analyzer als eigenständiges Messsystem zu entwickeln, konnte weitestgehend erfüllt werden. Es ist ein Messgerät entstanden, dass alle notwendigen Hardwarekomponenten in einem Gehäuse vereint und somit flexibel eingesetzt werden kann. Es können Frequenzgangmessungen von Systemen durchgeführt und Effektivwert, Frequenz und Phase von zwei Signalen gemessen werden.

Die Untersuchung der verfügbaren EVMs und die anschließende Bewertung haben gezeigt, dass diese aufgrund ihrer Spezifikationen nicht flexibel genug sind, um die Anforderungen für die Verwendung in einem Audio-Analyzer zu erfüllen. Dies liegt größtenteils an den verwendeten Analog-Interfaces. Im Fall des OMAP-LCDK kommt hier ein Audio-Codec zum Einsatz, der intern bereits eine digitale Vorverarbeitung vornimmt, die nicht vollständig bekannt ist. Dies ist für die einfache Audiowiedergabe oder Aufnahme nicht relevant, sollen jedoch Signale ausgegeben und anschließend analysiert werden, müssen alle Teile der Signalverarbeitung bekannt sein. Zusätzlich besitzt der Audio-Codec lediglich einen Stereo Aus- und Eingang.
Das UniDAQ2 nutzt keinen Audio-Codec, sondern einfache ADCs und DACs. Hier findet keine digitale Vorverarbeitung statt, die eine Verwendung in dieser Arbeit erschweren. Allerdings ist die Bandbreite durch analoge Filter bereits so stark begrenzt, dass eine Nutzung nicht sinnvoll ist.
Bei der Auswahl einer Hardwareplattform wurde sich deshalb für das D.Module2.C6657 als DSP-Modul und das D.Module2.ADDA500K16 als Analog-Interface entschieden. Dieses besitzt keinen der Nachteile der anderen EVMs und bietet zudem eine ausreichend leistungsfähige Basis. Der auf diesem EVM befindliche DSP besitzt zwei Prozessoren, so dass hier ebenfalls die Inter-Prozessor-Kommunikation untersucht werden konnte.

Die Entwicklung der erforderlichen Hardware erfolgte mit dem UPV als Vorlage. Grundlegende Schaltungselemente konnten aus dessen Dokumentation übernommen werden, jedoch ist die vollständige Schaltung des UVP nicht bekannt und so musste die weitere Entwicklung

selbstständig erfolgen. Da die Hardwareentwicklung nur einen Teil dieser Arbeit ausmacht, ist diese klein gehalten worden und könnte in nachfolgenden Arbeiten weiter entwickelt werden. Das gewählte 19-Zoll Gehäuse bietet dafür die Möglichkeit, da Platinen austauschbar sind und noch Platz für weitere Platinen verfügbar ist.

Die Entwicklung der Software umfasst einen größeren Teil der Arbeit, da diese für zwei Prozessorkerne und das User-Interface erfolgen musste. Das Zusammenspiel der einzelnen Softwarekomponenten hat zusätzlich viel Zeit in Anspruch genommen, da die notwendigen Schnittstellen implementiert werden mussten und der Informationsfluss gesteuert werden musste. Die Verwendung des SYS/BIOS als RTOS hat sich bewährt, da so ein zusätzlicher Programmieraufwand, um die Ressourcen des Prozessors zu verteilen, nicht erforderlich war. Zusätzlich konnten Softwarepakete, wie das NDK oder das IPC genutzt werden. Die Konfiguration des SYS/BIOS verlief mit den verfügbaren Dokumentationen von TI und den Beispielprogrammen von d.signt problemlos.
Es hat sich jedoch gezeigt, dass das vom Webserver verwendete CGI nicht optimal geeignet ist, da die Kommunikation mit dem User-Interface nur vom Client aus initiiert werden kann. Die Verwendung der MessageQs des IPC um Daten zwischen beiden Kernen auszutauschen funktioniert sicher, jedoch ist hier der Aufwand, um einzelne Messwerte abzufragen, sehr groß. Der direkte Zugriff auf den DDR3 von beiden Prozessoren aus ist eine Variante, die mit weniger Aufwand realisiert werden kann. Bei größeren Datenmengen sollte hier noch zusätzlicher Programmieraufwand erfolgen, um den Zugriff beider Prozessoren auf diesen Speicher zu steuern.

Abschließend lässt sich sagen, dass die Aufgabenstellung einen großen Themenbereich umfasst, der in nachfolgenden Arbeiten weiter untersucht werden kann. Die Signalverarbeitung kann um weitere Messungen ergänzt und die Möglichkeiten des User-Interfaces erweitert werden. Der entstandene Audio-Analyzer bietet eine gute Grundlage dafür und die Hardware sowie die Software wurden so konzeptioniert, dass eine Erweiterung einzelner Teile möglich ist.

Literatur

[1] Texas Instruments. (2019). OMAP-LCDK User Guide, Adresse: http://www.ti.com/lit/ug/spruil2a/spruil2a.pdf (besucht am 23. 10. 2019).

[2] Texas Instruments. (2014). TLV320AIC3106 Datenblatt, Adresse: http://www.ti.com/lit/ds/symlink/tlv320aic3106.pdf (besucht am 23. 10. 2019).

[3] d.signt. (2018). UniDAQ2 Datenblatt, Adresse: https://www.dsignt.de/files/dsignt/media/datasheets/dsunidaq2_dsp-adda.pdf (besucht am 23. 10. 2019).

[4] d.signt. (2016). D2.Base Datenblatt, Adresse: https://www.dsignt.de/files/dsignt/media/datasheets/dsdm2basep.pdf (besucht am 23. 10. 2019).

[5] d.signt. (2016). D.Module2.ADDA500K16 Datenblatt, Adresse: https://www.dsignt.de/en/peripheral-modules/daq-board-d-module2-adda500k16.html (besucht am 23. 10. 2019).

[6] d.signt. (2014). D.Module2.C6747 Datenblatt, Adresse: https://www.dsignt.de/files/dsignt/media/datasheets/tdd2c6747.pdf (besucht am 23. 10. 2019).

[7] d.signt. (2018). D.Module2.C6657 Datenblatt, Adresse: https://www.dsignt.de/files/dsignt/media/datasheets/tdd2c6657.pdf (besucht am 23. 10. 2019).

[8] Rohde und Schwarz. (2013). D.Module2.C6657 Datenblatt, Adresse: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_common_library/dl_brochures_and_datasheets/pdf_1/UPV_dat_sw_en_0758-1306-22_v0400.pdf (besucht am 23. 10. 2019).

[9] *D.Module2.ADDA500K16 User Guide*, Rev 1.0, d.signt, Marktstr. 10, 47647 Kerken, 2011.

[10] Rohde und Schwarz. (2015). UPV Bedienhandbuch, Adresse: https://scdn.rohde-schwarz.com/ur/pws/dl_downloads/dl_common_library/dl_manuals/gb_1/u/upv_1/UPV_Bedienhandbuch_de_13.pdf (besucht am 22. 10. 2019).

[11] Texas Instruments. (2015). ULN2003LV Datenblatt, Adresse: http://www.ti.com/lit/ds/symlink/uln2003lv.pdf (besucht am 23. 10. 2019).

[12] Texas Instruments. (2018). TI-RTOS Kernel (SYS/BIOS) User's Guide, Adresse: http://www.ti.com/lit/ug/spruex3u/spruex3u.pdf (besucht am 22. 10. 2019).

[13] *D.Module2.C6657 User Guide*, Rev 1.12, d.signt, Marktstr. 10, 47647 Kerken, 2013.

[14] Texas Instruments, *IPC User Guide*. Adresse: http://processors.wiki.ti.com/index.php/IPC_Users_Guide# (besucht am 23. 10. 2019).

[15] Texas Instruments. (2017). NDK Reference Guide, Adresse: http://www.ti.com/lit/ug/spru524k/spru524k.pdf (besucht am 23. 10. 2019).

[16] J. Wolf, *HTML5 und CSS3: das umfassende Handbuch*. Rheinwerk, 2018.

[17] HTML5-Templates.com. (2019). HTML-Template Windows, Adresse: https://html5-templates.com/preview/windows.html (besucht am 25. 10. 2019).

[18] Texas Instruments. (2004). Generation of a Sine Wave Using a TMS320C54x DSP, Adresse: http://www.ti.com/lit/an/spra819/spra819.pdf (besucht am 25. 10. 2019).

Glossar

Entwicklungsmodul

Ein Entwicklungsmodul ist eine Prototypenplattform für spezielle Hardware. Hersteller bieten diese an, damit man ohne aufwendige Hardwareentwicklung für einzelne Chips, einen Prototyp anfertigen und testen kann.

Real Time Operating System

Ein Echtzeitbetriebssystem (engl. Real Time Operating System) ist ein Betriebssystem, das die Programmierung von Anwendungen mit Echtzeit-Anforderungen erleichtert. Echtzeit bedeutet hier, dass auf Ereignisse in einer festgelegten Zeit reagiert werden muss. Das Echtzeitbetriebssystem unterstützt den Programmierer der Anwendung darin, dass es Scheduling-Verfahren zur Verfügung stellt und Aufgaben priorisieren lässt. Je nach gewähltem Scheduling-Verfahren kümmert sie das Echtzeitbetriebssystem um die korrekte zeitliche Ausführung und Einhaltung der Echtzeitanforderungen.

Software Development Kit

Ein Software Development Kit ist eine Zusammenstellung von Software für einen bestimmten Anwendungszweck. Sie enthalten alle für die Entwicklung einer Anwendung notwendigen Bestandteile.

Anhang

A. Spezifikation des Audio-Analyzers

Analyzer		
Eingänge	2x symmetrisch	XLR Pin 2 und 3
	2x unsymmetrisch	über XLR-BNC Adapter
Spannungsbereich	+/- 3V Fullscale	
	+/- 30V Fullscale	1:10 Spannungsteiler
Bandbreite	100 kHz	digital begrenzt
Kopplung	AC / DC	Koppelkondensator zuschaltbar
Eingangsimpedanz	100 kΩ	jeder Pin nach GND

Generator		
Ausgänge	2x unsymmetrisch	
Spannungsbereich	+/- 2,5V Fullscale	
Ausgangsimpedanz	50 Ω	
Bandbreite	100 kHz	analog begrenzt

B. Pinbelegung der Busplatine

DIN 41612 Type C 64 P Belegung

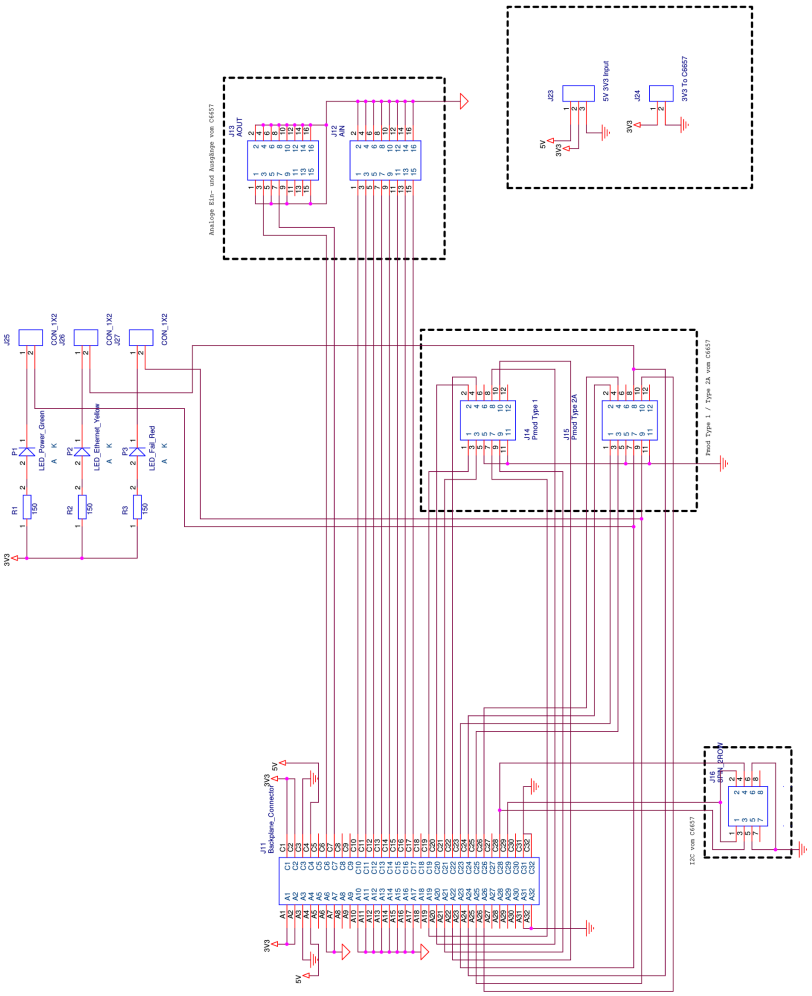
Nummer	A		C
1	3V3		3V3
2	3V3		3V3
3	GND		GND
4	5V		5V
5			
6	AGND		OUT_A
7	AGND		OUT_B
8			
9			
10	AGND		IN_A0
11	AGND		IN_A1
12	AGND		CAL
13	AGND		IN_B0
14	AGND		IN_B1
15	AGND		CAL
16	AGND		IN_C0
17	AGND		IN_C1
18			
19	IO1_2		IO1_1
20	IO2_2		IO2_1
21	IO3_2		IO3_1
22	IO4_2		IO4_1
23	IO1		SS_N
24	IO2		MOSI
25	IO3		MISO
26	IO4		SCK
27			
28			SDA
29			SCL
30			
31	GND		GND
32	GND		GND

C. Pinbelegung des EVM

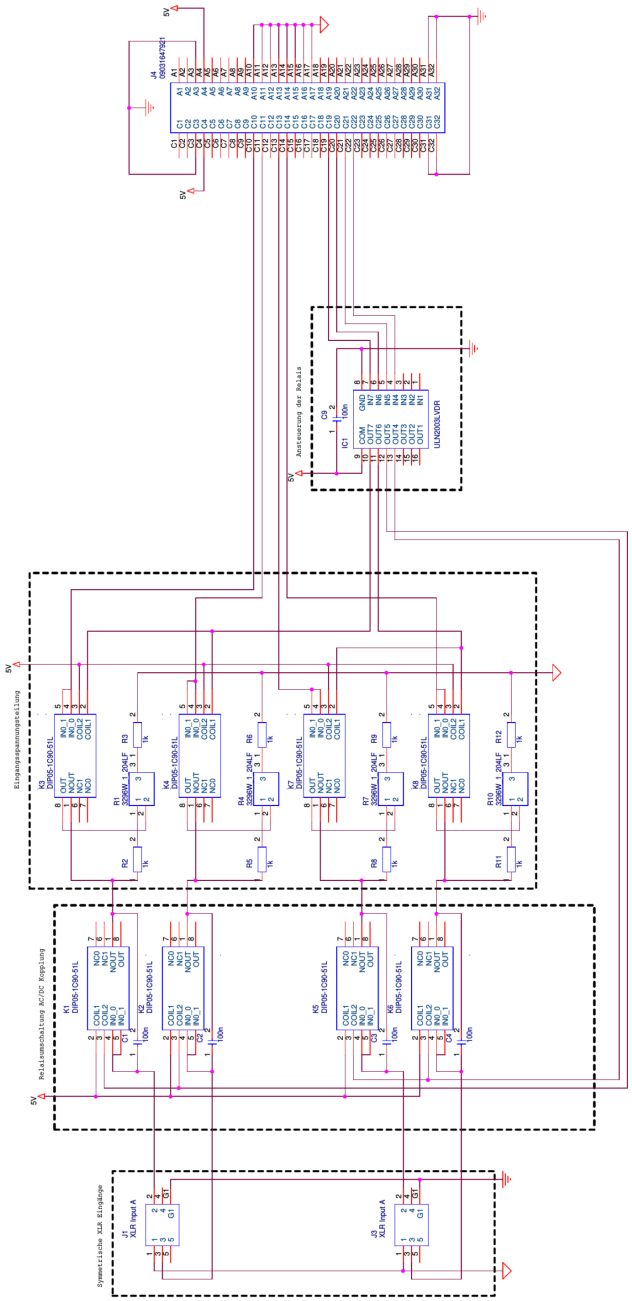
Header	Pin	Name	Beschreibung	Funktion
Pmod Type 1	1	IO1_1	PRGPIO[0]	Relais Ch. A Skalierung
	2	IO2_1	PRGPIO[1]	Relais Ch. B Skalierung
	3	IO3_1	PRGPIO[2]	Relais Ch. A AC DC
	4	IO4_1	PRGPIO[3]	Relais Ch B AC DC
	5	GND		
	6	VCC	3V3	
	7	IO1_2	PRGPIO[4]	Relais Ch. A Off
	8	IO2_2	PRGPIO[5]	Relais Ch. B Off
	9	IO3_2	PRGPIO[6]	Relais 50 Ohm
	10	IO4_2	PRGPIO[7]	
	11	GND		
	12	VCC	3V3	
Pmod Type 2A	1	SS_N	ChipSelect	
	2	MOSI	Master Out Slave In	
	3	MISO	Master In Slave Out	
	4	SCK	Clock	
	5	GND	Ground	
	6	VCC	3V3	
	7	IO1	PRGIO[8]	Led Power
	8	IO2	PRGIO[9]	Led Ethernet
	9	IO3	PRGIO[10]	Led Error
	10	IO4	PRGIO[11]	
	11	GND	Ground	
	12	VCC		
AIN	1	IN_A0	analog in A0	Eingang Ch. A+
	2	AGND	analog gnd internal connection to GND	
	3	IN_A1	analog in A1	Eingang Ch. A-
	4	AGND	analog gnd internal connection to GND	
	5	CAL	ext. Calibration input	
	6	AGND	analog gnd internal connection to GND	
	7	IN_B0	analog in B0	Eingang Ch. B+
	8	AGND	analog gnd internal connection to GND	
	9	IN_B1	analog in B1	Eingang Ch. B-
	10	AGND	analog gnd internal connection to GND	
	11	CAL	ext. Calibration input	
	12	AGND	analog gnd internal connection to GND	
	13	IN_C0	analog in C0	Eingang Gen. A
	14	AGND	analog gnd internal connection to GND	
	15	IN_C1	analog in C1	Eingang Gen. B
	16	AGND	analog gnd internal connection to GND	
AOUT	1	AGND	analog gnd internal connection to GND	
	2	AGND	analog gnd internal connection to GND	
	3	OUT_A	analog out A	Ausgang Gen. A
	4	AGND	analog gnd internal connection to GND	
	5	AGND	analog gnd internal connection to GND	
	6	AGND	analog gnd internal connection to GND	
	7	OUT_B	analog out B	Ausgang Gen. B
	8	AGND	analog gnd internal connection to GND	
	9	AGND	analog gnd internal connection to GND	
	10	AGND	analog gnd internal connection to GND	
	11	EXTCLK_OUT	clock output to sync other ADDA500K	
	12	AGND	analog gnd internal connection to GND	
	13	EXTCLK_IN	ext sampling clock input	
	14	AGND	analog gnd internal connection to GND	
	15	AGND	analog gnd internal connection to GND	
	16	AGND	analog gnd internal connection to GND	

D. Schaltpläne der Platinen

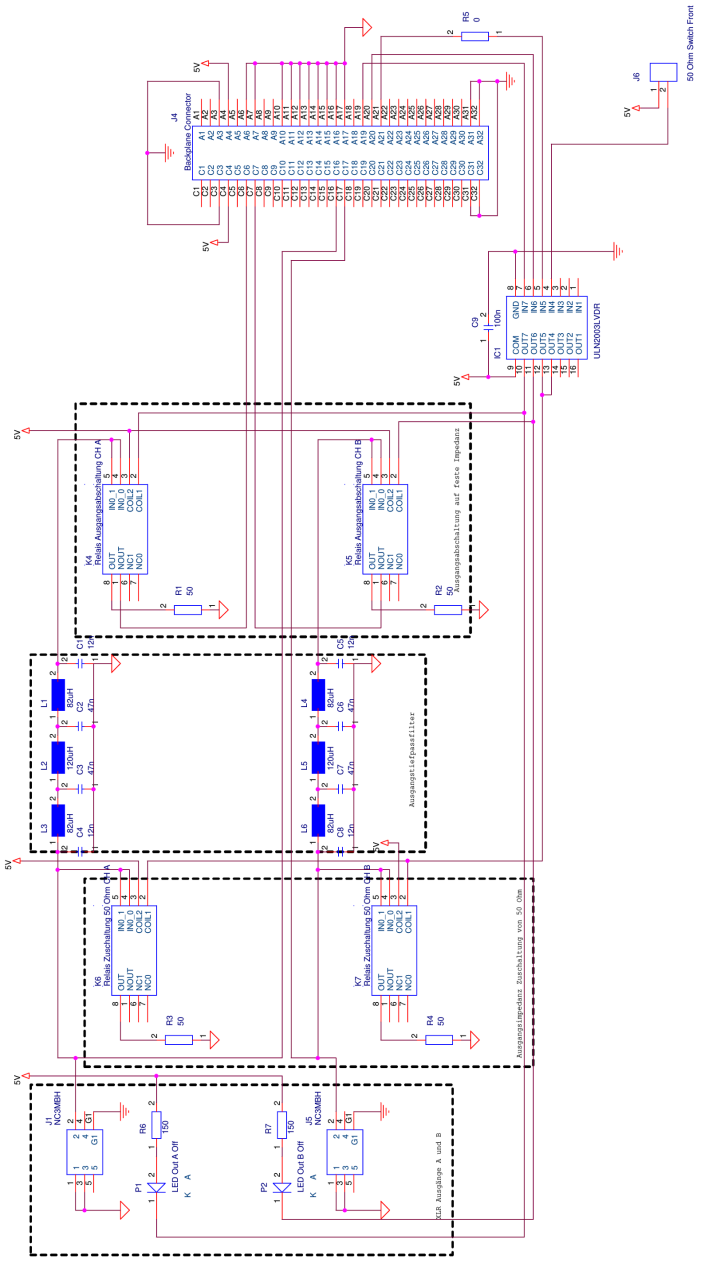
Adapterplatine



Analyzerplatine



Generatorplatine



E. verwendete Softwarepakete

Softwarepaket	Version
CCS	9.0.1
XDCTools	3.23.6.96
IPC	3.21.0.07
NDK	2.24.0.11
SYS/BIOS	6.37.4.32
DSPLIB	3.4.0.3
jQuery	1.12.4
Chart-JS	2.8.0

Versicherung über die Selbstständigkeit

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

Hamburg, 6. November 2019
Ort, Datum

Unterschrift