Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

Bachelorthesis

Benjamin Sellak

Development and Implementation of a TMS320C6713-based digital audio mixing including effects application with Matlab live remote control

Fakultät Technik und Informatik
Department Informations- und Elektrotechnik

Faculty of Engineering and Computer Science
Department of Information and Electrical Engineering

Benjamin Sellak

Development and Implementation of a TMS320C6713-based digital audio mixing including effects application with Matlab live remote control

Bachelorthesis eingereicht im Rahmen der Bachelorprüfung
im Studiengang Informations- und Elektrotechnik
am Department Informations- und Elektrotechnik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr.-Ing. Jürgen Vollmer
Zweitgutachter : Prof. Dr.-Ing. Ulrich Sauvagerd

Abgegeben am 29. August 2017

Benjamin Sellak

Title of the paper

Development and Implementation of a TMS320C6713-based digital audio mixing including effects application with Matlab live remote control

Keywords

DSP, Matlab, TMS320C6713, App Designer, Digital Biquad Filters, Nonlinear Systems, Dynamic Range Control, Multirate Signal Processing

Abstract

Subject of this thesis is the implementation of the signal chain of a generic digital mixing desk's channel. The focus hereby lies on the simulation of the most common signal processing units of such a device with Matlab, with subsequent implementation on a TMS320C6713 DSP development kit. Control over various processing parameters in real-time via serial interface is provided by a Windows software created with the Matlab App Designer

Benjamin Sellak

Thema der Bachelorthesis

Entwicklung und Implementierung einer TMS320C6713-basierten digitalen Audiomisch- und Effektanwendung mit Matlab-basierter Live-Steuerung

Stichworte

DSP, Matlab, TMS320C6713, App Designer, Digitale Biquadfilter, Nichtlineare Systeme, Dynamikumfang, Multiratensignalverarbeitung

Kurzzusammenfassung

Thema dieser Arbeit ist die Implementierung der Signalverarbeitungskette eines Digitalmischpultkanals. Hierbei liegt der Fokus auf der Simulation der gängigen Signalverarbeitungsblöcke solcher Geräte mit Matlab, sowie der anschließenden Implementierung auf einem TMS320C6713 Entwicklungskit. Verschiedene Parameter können dabei in Echtzeit über serielle Schnittstelle mittels in Matlab App Designer erstellter Windows-Software gesteuert werden.

Acknowledgement

Thanks to my family Dana, Atreyu and Peppa for showing patience and understanding for many nights spent in front of the computer and for all the support - I love you!

Thanks to my advisors Prof. Dr. Vollmer and Prof. Dr. Sauvagerd for providing extensive insight and frequent feedback during the course of this project.

Thanks to Friedemann Kootz and Andreas Schwarz for allowing me to use their photographies of beautiful audio gear and beautiful people.

Thanks to Jünger Audio GmbH for allowing me to use their facilities for measurements.

Thanks to Matthias Koelle, Maxim Zyrianov and David Ditter of Jünger Audio for explaining how things work in 'the real world'.

Thanks to Simon Nibbrig and Roman Quiring for providing two more pairs of eyes to spot errors and inaccuracies.

Thanks to Midas Consoles for looking out old XL4 press kits for me and letting me use them.

Contents

List of Tables

List of Figures

List of Acronyms

ADC	Analog-Digital Converter
ASCII	American Standard Code for Information Interchange
BNC	Bayonet Neill Concelman
CCS	Code Composer Studio
CD-ROM	Compact Disc-Read Only Memory
CPU	Central Processing Unit
DAW	Digital Audio Workstation
dB	Decibel
DAC	Digital-Analog Converter
DJ	Disk Jockey
DMA	Direct Memory Access
DSP	Digital Signal Processor
EMIF	External Memory Interface
EDMA	Enhanced Direct Memory Access
EQ	Equalizer
FFT	Fast Fourier Transformation
FIFO	First In, First Out
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
FS	Full Scale
GUI	Graphical User Interface
I^2C	Inter-Integrated Circuit

IEEE	Institute of Electrical and Electronics Engineers
IDE	Integrated Development Environment
IID	Inter-aural Intensity Difference
IIR	Infinite Impulse Response
ITD	Inter-aural Time Difference
I/O	Input/Output
LED	Light Emitting Diode
LTI	Linear Time-Invariant
McBSP	Multichannel Buffered Serial Port
MVC	Model-View-Controller
PA	Public Address
PFL	Pre-Fader Listening
PLL	Phase-Locked Loop
PPM	Peak Programme Meter
RMS	Root Mean Square
SDRAM	Synchronous Dynamic Random-Access Memory
SPI	Serial Peripheral Interface
SNR	Signal-to-Noise Ratio
SRAM	Static Random Access Memory
TDF2	Transposed Direct Form 2
THD	Total Harmonic Distortion
TRS	Tip-Ring-Sleeve
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
VLIW	Very Long Instruction Word
VU	Volume Unit

1. Introduction

Common working environments of audio professionals, like live concerts, recording studios, theaters and broadcasting stations have been subject to an ongoing revolution that came with the advent of digital mixing consoles in the 1990's. Were analogue consoles mainly designed for signal mixing and routing purposes while relying on a large amount of dedicated peripheral analog and digital signal processing units (the so-called *outboard gear*), digital desks potentially unite all these processing features in a single device, rendering most of additional outboard equipment obsolete.

1.1. Scope of this thesis

This thesis tackles the design and the implementation of a digital mixing and effects application prototype in the vein of the consoles thoroughly described in chapter 1.2.3. Focus hereby lies less on powerful routing capability and more on the various signal processing blocks found within such systems. The prototype itself will run on a Texas Instruments-powered **DSP** development kit with on-board audio codec and converters. There will be no hardware control surface to influence the signal processing. Instead, a remote control PC software will interface the DSP via **RS 232** and offers full control over a wide array of signal processing parameters. Figure 1.1 shows an overview of the system.

1.1.1. Functional specification

1.1.1.1. Signal processing functionality

Figure 1.2 displays the block diagram of the mixing and effects application. The signal processing features are as following:

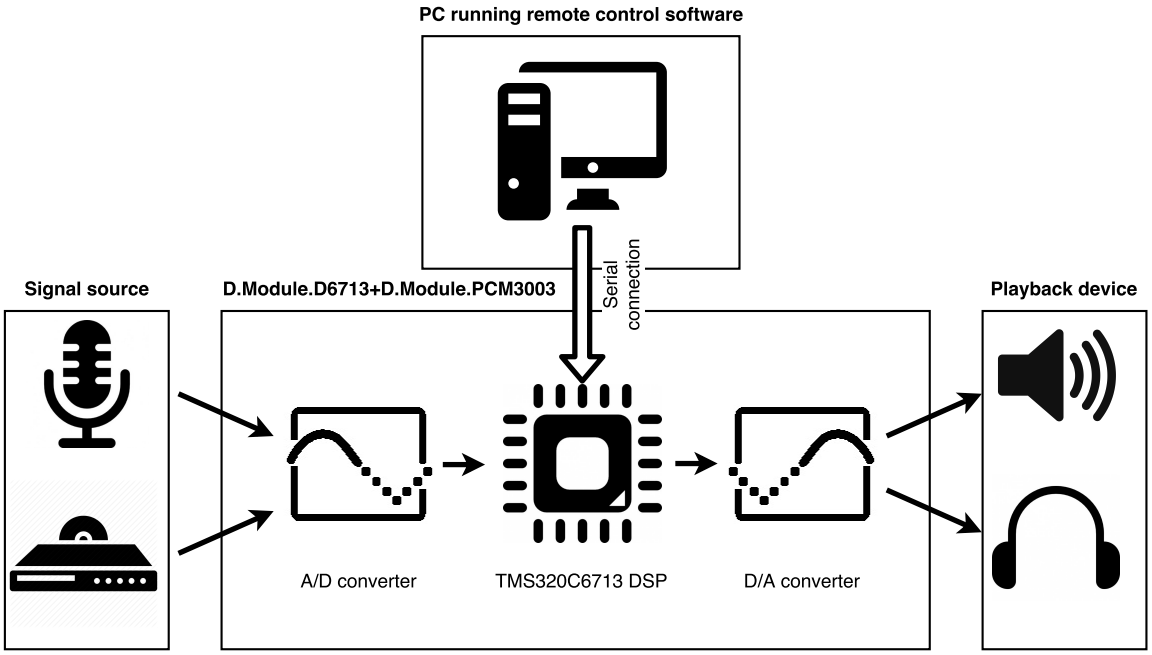


Figure 1.1.: *The digital mixing and effects application - system overview*

I/O

- Two mono inputs, one stereo output
- Sampling rate 48 kHz, 16 bit resolution

Input section

- Input gain from -36 dB to +18 dB
- Polarity switch
- High pass filter trimmable from 20 Hz to 400 Hz with bypass switch
- Parametric equalizer with bypass switch, four bands with variable gain g, corner/center frequency f_0 and quality factor Q and slope S respectively. Parameter ranges as in table 1.1
- Limiter with bypass switch and variable threshold
- Dynamic range control section with expander threshold, expander ratio, compressor threshold, compressor ratio, attack time, release time and makeup gain. Bypass

switches for compressor and expander. Range of dynamics parameters are shown in table 1.2

- Aux bus routing section with routing to the overdrive effect bus from $-\infty$ to 0 dB
- Master bus routing section with panorama knob and level slider

Effect section

- Overdrive effect with two different parametrized distortion characteristics. Mono input, stereo output to master bus

Master bus section

- Attenuation sliders from $-\infty$ to 0 **dBFS** for left and right outputs.

1.1.1.2. PC remote control software functionality

The PC software features are:

- precise real-time control of all aforementioned parameters via clear and appealing **GUI**
- Visual feedback of the current amplitude response of high pass filter and equalizer in effect per channel
- Visual feedback of the static characteristic in effect of each channel's dynamic range control section
- Visual feedback of the static characteristic of the overdrive.

Filter type	g **dB range**	f_0 **min**	f_0 **max**	Q / S **min**	Q / S **max**
low shelf	$\pm$18 dB	20 Hz	800 Hz	0.1	1.0
low-mid peak	$\pm$18 dB	100 Hz	2 kHz	0.7	3.0
mid-high peak	$\pm$18 dB	400 Hz	8 kHz	0.7	3.0
high shelf	$\pm$18 dB	1 kHz	20 kHz	0.1	1.0

Table 1.1.: *Parametric equalizer - range of parameters*

Parameter	Compressor	Expander	Limiter
threshold min	$-\infty$	-45 dBFS	-18 dBFS
threshold max	-45 dBFS	0 dBFS	0 dBFS
ratio min	0.01	1	fixed ∞
ratio max	1	5	fixed ∞
attack min	1 ms		fixed 500 μs
attack max	100 ms		fixed 500 μs
release min	1 ms		fixed 200 ms
release max	1 s		fixed 200 ms
makeup gain min	0 dB		
makeup gain max	$+36$ dB		

Table 1.2.: *Compressor, expander and limiter - range of parameters*

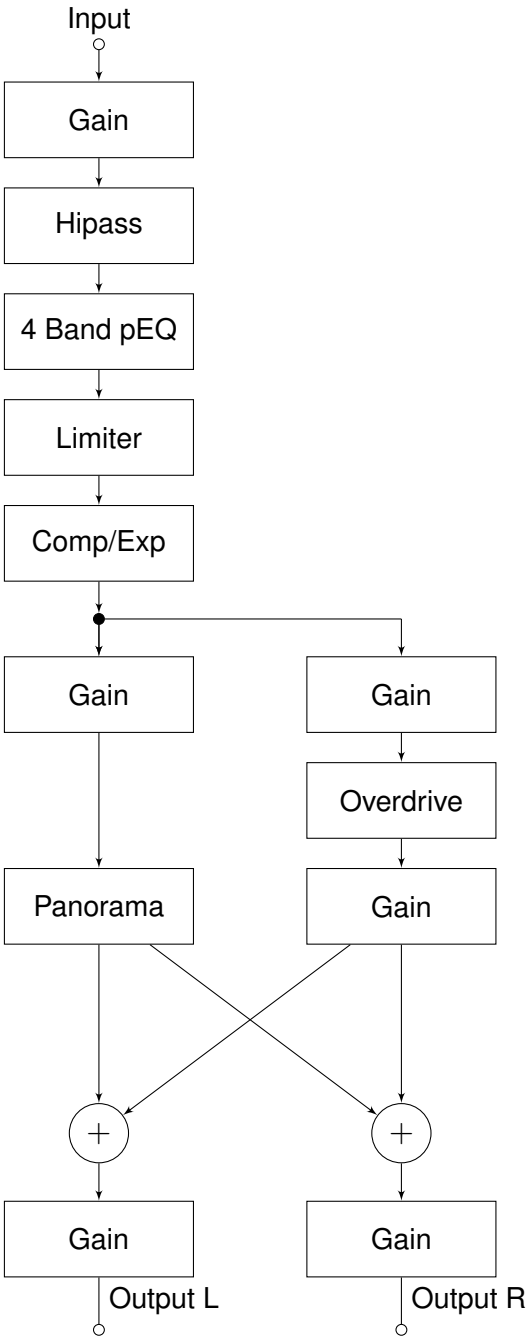


Figure 1.2.: *Comprehensive block diagram of the mixing and effects signal processing*

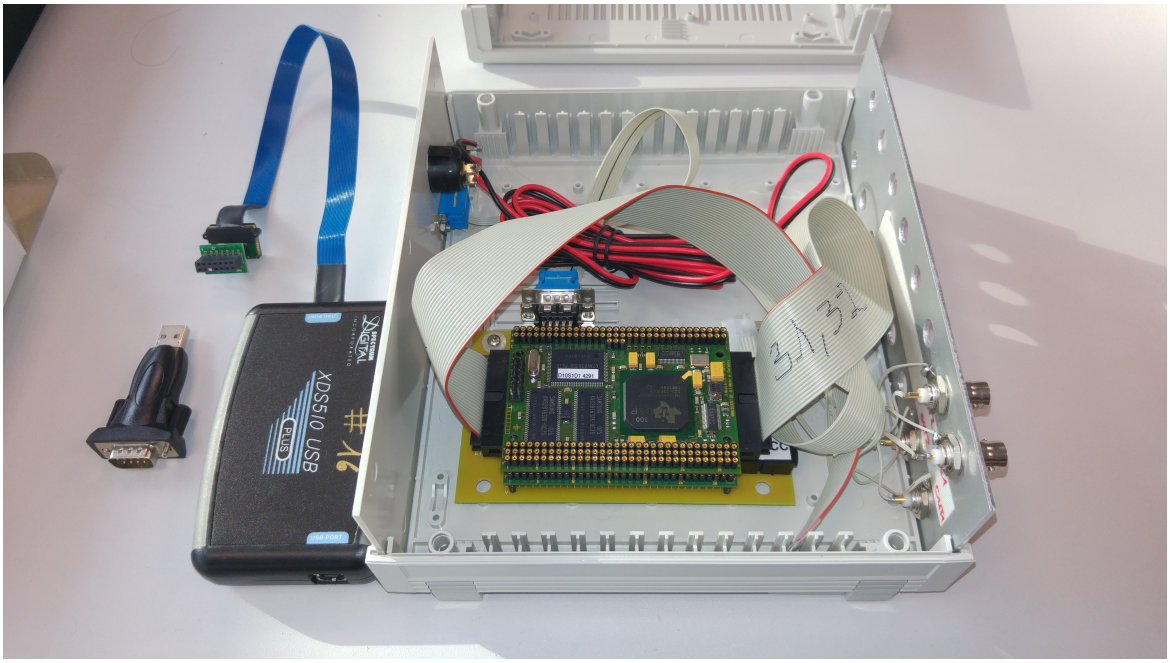


Figure 1.3.:
D.Module.C6713 wit D.Module.PCM3003, Spectrum Digital XDS510 USB JTAG emulator and generic USB-to-Serial Converter

1.1.2. Technology

1.1.2.1. D.Module.C6713 and D.Module.PCM3003

The D.Module.C6713 from D.Sign.T is a standalone **DSP** development board based on the Texas Instruments TMS320C6713 digital signal processor. This processor is clocked with 225 MHz, its **VLIW** architecture enables multiple parallel execution of fixed point operations and floating point operations with single (16 bit) or double (32 bit) precision. It offers 4 kByte of level 1 program cache and data **CPU cache** each with an additional 256 kByte level 2 cache.
Besides other interfaces like **SPI** and **I^{2}C**, the processor is equipped with two McBSPs (Multichannel Buffered Serial Port) that enable native handling of various audio stream formats like I^{2}S and an enhanced direct memory access (E**DMA**) to enable fast interfacing of peripherals while maintaining minimal CPU load.
These features render the TMS320C6713 an appropriate chip for real time audio signal processing.
The D.Module.C6713 extends the DSP by 16 MByte **SDRAM** and 512 kByte burst **SRAM** as well as a UART interface. The D.Module.C6713 is programmed and debugged via Spectrum Digital XDS510 USB JTAG emulator and the Texas Instruments **IDE CCS** (Code Composer

Studio) v.5.5. Programs for the DSP can be written in C and in Assembler. D.Sign.T ships a C library with the module, the D.Module.BIOS, which offers a set of functions to handle low-level programming tasks that involve on-board peripherals.
The D.Module.C6713 board gives another D.Sign.T module a piggyback, the D.Module.PCM3003. It features four name giving TI PCM3003 audio **codec**s that are connected to the DSP via the McBSP. The codecs itself offer 64x oversampling delta-sigma converters and do not require anti-**aliasing** filters: Passive first order RC networks designed for 48 kHz sampling rate/16 bit resolution can be found at the input and output stages. The input/output signals are accessible through DIN 41651 connectors and passed to the exterior of the casing to be laid out as female **BNC** connectors.
The configuration and setting of CCS comply with the guidelines of the digital signal processing laboratory of the Hochschule für Angewandte Wissenschaften Hamburg [1].

1.1.2.2. Matlab & App Designer

Matlab by *The MathWorks, Inc.* is both a high level programming language and an **IDE** likewise that has been constantly developed further since its initial release in 1984. Its origins lie in numerical computing and matrix manipulation as well as graphical display of data. The

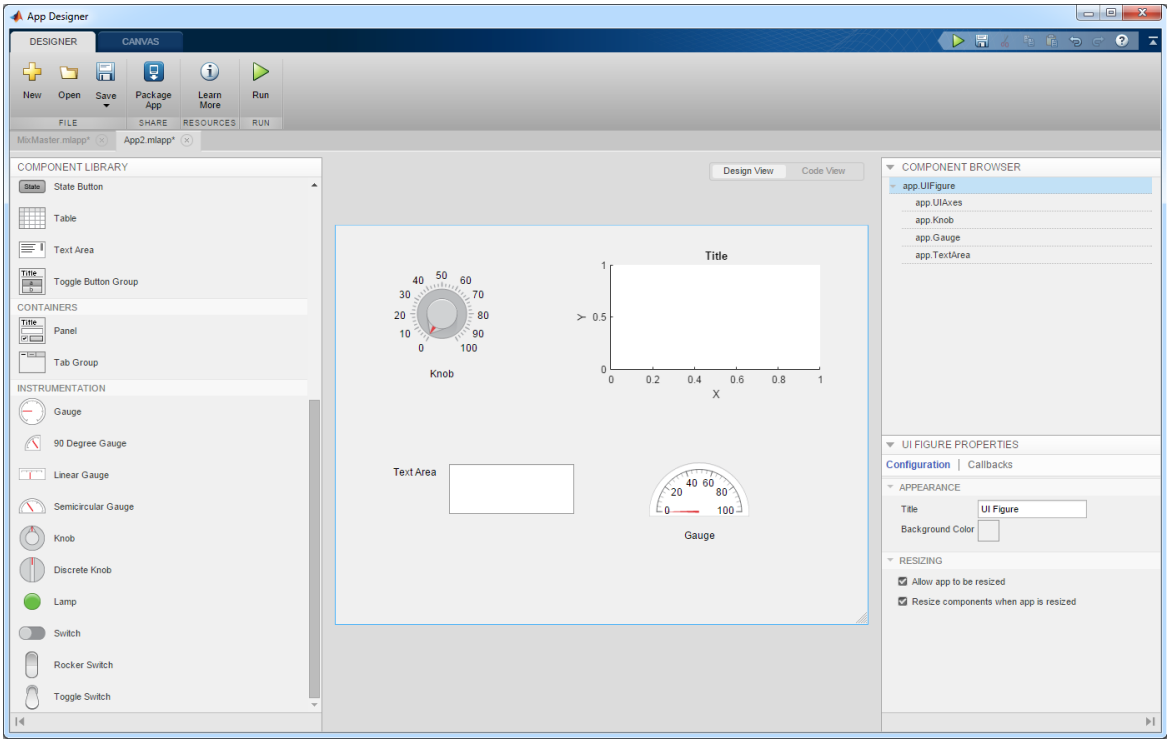


Figure 1.4.: *Matlab App Designer*

Digital Signal Processing Toolbox offers numerous functions to aid in the design, simulation and verification of digital systems. The Matlab language supports multiple programming paradigms.
This makes Matlab a suitable choice for simulation of the processing blocks outlined in section 1.1. The entire Matlab source code that has been developed during the making of this thesis was created with and is compatible to Matlab R2016b.
With the introduction of *App Designer* in Matlab R2016a, a new GUI framework was presented to supersede the outdated GUIDE environment. The very accessible App Designer allows the creation of visually pleasant user interfaces via drag-and-drop and offers a variety of interactive control elements like buttons, drop-down-menus and text fields, but also more application-specific gauges, rotary knobs and sliders in addition to the capability to embed the well-known Matlab figures. Software developed this way can be packaged into *apps* that are deployable as a single file and can be run directly from the Matlab toolbar.
The support of an object-oriented programming paradigm allows the utilization of modern software engineering practices and the application of effective design patterns in collaboration with a GUI framework that provides native support for audio mixing related UI elements, Matlab App Designer has been chosen to be the host application for this project's remote control software. This further reduces coding expenditure as Matlab modules can be reused that had been developed for simulation purposes.
A drawback of using App Designer is the low refresh rate of displayed elements. An app designed with this tool can not refresh faster than with approximately one frame per second. This is the case regardless of the software version or the host computer. An inquiry at MathWorks support revealed that this is a known issue that might be fixed in a future release of Matlab, but there is no workaround for any version available at the time this thesis is written. This renders the integration of any metering unfeasible since a refresh rate of less than 1 Hz is unacceptable for audio metering.
Nonetheless it is the author's belief that, given the project's time constraints, App Designer is the optimal choice for developing a GUI-driven control software for the DSP.

1.1.2.3. RS-232, Serial Port and UART Interface

The *RS-232* is a standard for serial communication that was defined in 1969 by the Electronic Industries Association [2] and describes the physical layer of this interface such as voltage levels, signal timing and connector properties.
A *serial port* characterizes an interface where information is transmitted in series, i.e. one bit at a time. In a wider sense, 'serial port' can refer to any hardware that is compliant to the RS-232 standard. The serial port mostly disappeared from consumer personal computers and would require an additional device such as a USB-to-Serial converter. It maintains an important role in the industry due to the relative lack of complexity of the underlying standard, the low cost of components and its robustness.

The *universal asynchronous receiver/transmitter (UART)* describes a transmission protocol as well as a hardware device implementing this protocol. Data can be sent and received in full duplex. The asynchronous mode of operation makes the use of a dedicated clock line unnecessary. This requires a receiver to be clocked at multiple times of the data rate. The transmission rate of symbols per second a serial connection is usually referred to as *baud rate* where one baud is the unit equivalent to one symbol transmitted. As the serial connection's dictionary consists of two symbols, *One* and *Zero*, baud- and bitrate are equivalent in this case. It is commonly set to 9.600 baud per second, although higher rates can be found as well as lower ones.
The *no data* state is high voltage (to enable detection of a faulty line). Each word begins with the start bit, followed by five to nine data bits, zero to one parity bit for error checking and one or two stop bits. There is no handshake process for communication participants to negotiate the data frame and rate - both must be set to identical values beforehand; otherwise communication will not be possible.
The D.Module.D6713 UART is equipped with a DE-9 connector that can transmit and receive RS232-compliant data streams and it features 32 byte deep transmit and receive **FIFOs**. By default, the UART is configured to 9600 baud, 8N1 (eight data bit, no parity, one stop bit, see table 1.3). The D.Module.BIOS provides library functions to access the UART.

Nr	0	1	2	3	4	5	6	7	8	9
Bit	start bit	LSB 0	bit 1	bit 2	bit 3	bit 4	bit 5	bit 6	MSB 7	stop bit

Table 1.3.: *Default D.Module.D6713 UART datagram*

Windows PCs can access a USB-to-Serial converter (or the serial port directly if applicable) via the generic COM-Port. Matlab provides an extensive serial port I/O support. Core of the communication is the *serial port object* with a wide array of functions and properties to enable asynchronous reading and writing via UART.

1.2. Mixing desks: an overview

There is a vast number of devices labelled as *mixing desks*, *consoles* or simply *mixer* that cater to various specific professions and purposes. A DJ who entertains an audience at a night club and an audio engineer who is recording an orchestra have quite different requirements to their mixing consoles regarding the number of in- and outputs, the form factor, the general robustness, the capabilities of sound processing and many more aspects.
In the context of this thesis, the main focus lies on the kind of consoles that is prevalent

Figure 1.5.: *The Midas XL4 analog mixing console*

in sound reinforcement, recording and broadcasting environments (they do have significant differences as well and usually can't be interchanged easily; however those differences are outside the scope of this thesis). The main function of these mixing desks can be summed up as [3, p. 330]:

- Amplification and normalization of signals of various sources, particularly amplification of microphone signals (Gain)
- Sound design and balancing by means of filters (EQ), level control (Fader) and dynamic range control processors (Dynamics)
- Distribution of signals between connected devices and internal mix buses(Routing)
- technical and acoustical monitoring of input and output signals with metering, headphones, and loudspeakers (Monitoring)
- Mixing of single- or multichannel input signals with individual levels and/or delays to target formats like Mono, Stereo, Surround (Mix)
- Communication between control room and stage/recording room (Talkback).

Figure 1.6.:
The author mixing a concert on a Crest X-VCA console. A side rack with an array of graphical equalizers and various other outboard processors can be seen on the left.

1.2.1. Structure of analog mixing desks

An analog mixing desk can be split up into three general sections: the input section, the output section and the mix buses. Additionally, there can be an arbitrary amount of outboard equipment involved in the signal processing for which the console provides several points to connect with.
In the following, the text is accompanied by detailed views of the Midas XL4 analog console to illustrate the structure of mixing desks and give a non-exhaustive overview of its various functions. The XL4 was released in 1993 but is still frequently encountered in large concert venues and in the stock of rental services. Pictures are taken from the original XL4 Live brochure [4] with courtesy from MUSIC Group.
The features described in this section as well as the range of their parameters are what the author deems as common based on his experience gained during his years of work as a professional live sound engineer.

1.2.1.1. Input section

An average, moderately sized analog mixing desk easily offers over 100 potentiometers to the user for operation, as well as a plethora of faders and buttons, due to the analog console's design principle *one knob for each function*. This might seem overwhelming to the layman and a question that console operators frequently get to hear is an incredulous 'Do you really know what every knob does?'.

The majority of any board surface's real estate is occupied by numerous input modules equal in layout and function, the *channel strips*. Each channel strip represents an input signal and takes care of that signal's processing. There are two main kinds of inputs, mono inputs which are usually microphones and stereo inputs, which are generally designed to handle line-level signals. Although they do have slight differences (for example the input gain range of stereo channels is usually lower than that of a mono channel and they can lack **phantom power**), operating a stereo channel feels the same as a monophonic one with a single button/potentiometer affecting both L and R channels. Console vendors often sell the same product line with numerous different input configurations. The internal signal flow of a channel strip is consistent with the layout of the operating elements and goes from the top to bottom.

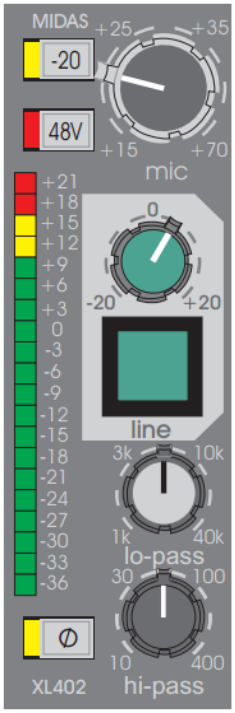


Figure 1.7.: *Head amplifier, phantom power, polarity reverse and high pass filter of an XL4 input channel*

Connectors The connectors are mostly found on the top or the rear end of an analog mixer. Mono inputs are fed with symmetric XLR connectors, TRS 6.3 mm jack, or XLR/jack hybrid connectors. Stereo inputs offer unbalanced RCA connectors and sometimes balanced TRS jack as well.

Head amplifier Depending on the source of the signal, an input channel must be capable of processing a wide level range. Delicate ribbon microphones may have an output less than +/-1 mV (-57.8 $\mathbf{dB}_U$ or -60 $\mathbf{dB}_V$)at 1 Pa sound pressure while line-level instruments like keyboards have an output up to 1 Volt ($\pm$0 $\mathbf{dB}_V$) when operated at usual range. A console input channel is expected to handle both kinds of signals equally without requiring any external signal conditioning beforehand. This is achieved through the first gain stage, the head amplifier. With a potentiometer the operator can trim the gain, commonly within a range of +20 to +80 dB. An additional switch, the *pad*, enables the user

to attenuate the signal by 20 dB. A headamp thus is required to be able to boost the signal by the factor 10.000 while keeping the added noise floor to a minimum and without adding distortion. It is easy to understand why the head amplifier is considered a mixing desk's most important sign of quality and number one indicator of whether a console sounds 'good' or not.

Phantom power Condenser microphones, ribbon microphones and active **D.I. boxes** require a supply voltage - the phantom power - to be able to operate; this source is 48 V DC, generated by the mixing console and applied to both balanced wires via switch.

Polarity Switch The polarity reverse button, or phase reverse, as it is often called quite inaccurately, flips the polarity of the input signal. This appliance comes into play when two signals with opposite polarity are to be mixed. A common example is the snare drum which is often recorded with two microphones: One directed at the top skin and one to the bottom skin and the snare wires. Since both skins are coupled, a concave deflection of one skin causes a convex deflection of the other and vice versa - , both signals are *out of phase* as it is called commonly. Mixing both without reversing one's polarity would cause an audible and undesirable cancellation of frequencies across the whole spectrum.

High-pass filter A high pass filter is usually applied to the signal in order to reject any undesired low frequency sound in the signal path, such as microphone handling noise or a microphone stand picking up floor vibrations. Cheaper mixing desks' high pass filters reject all low frequency rumble below a fixed cutoff frequency, usually between 50 and 90 Hz, while more professional consoles offer filters that are trimmable between 0 and 400 Hz or even above.

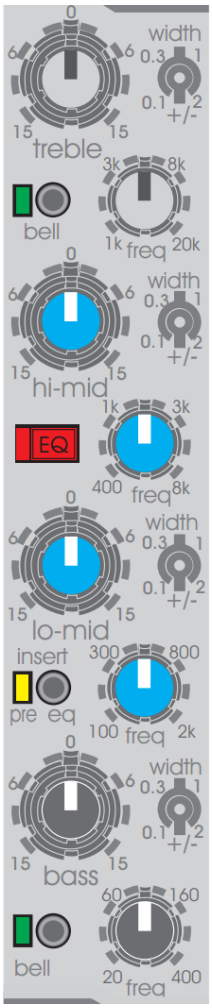


Figure 1.8.: *Parametric equalizer of an XL4 input channel*

Insert point The insert point enables the user to break the signal chain and loop in an external device (see section outboard gear). For this purpose, each input channel offers either one TRS jack (signal is sent to the outboard via tip and returned via ring) or two balanced TRS jacks (one dedicated send,

one return). This is the exception to the rule that the internal signal flow and the visual layout of control elements are equivalent as the loop-in happens before or after the equalizer section but the jack connectors are normally located next to the input XLR socket.

Equalizer While the high pass filter serves the purpose to filter out undesired signal components, the equalizer section is mainly used for shaping the signal. It consists of two types of filters, the shelving filter and the peak filter. The shelving type boosts or attenuates all signal content below (low shelf) or above (high shelf) a certain frequency by an amount typically in the +/- 12 dB range. The peak filter boosts or cuts frequencies around a center frequency with a given bandwidth. The equalizing section of a console is generally made up by a high and a low shelf as well as one up to four peak filters. Low end consoles' peak filters come with fixed center frequencies. If the center frequencies are trimmable by the operator, the equalizer is called *semi-parametric*, if the bandwidth of a peak filter is adjustable, too, it is a *(full-)parametric equalizer*.

Auxiliary send routing The auxiliary send routing (or aux) potentiometers allow the operator to split the equalized signal and route it to an aux bus (see section Buses). These sends come in the variant *pre-fader* and *post-fader*, meaning that either the amount of the signal routed to the bus is affected by the channel fader's position, or not. The former behavior is desired, for example, if the signal feed is supplied to artists performing on stage. If the console operator makes changes in the mix for the audience, the artist's monitor mix won't be affected this way. Post-fader aux sends on the other hand are preferred when the input is fed to an external effect processor like a digital reverberator. This way, if the input signal is altered in volume by the operator, the ratio *input signal:reverb signal* will stay consistent. Small mixing consoles might come with one or two auxiliary buses, bigger ones can have as much as 16 separate auxes and the pre/post behavior can be either fixed, can be switchable for certain or all auxiliary buses or can even be toggleable for each individual auxiliary send potentiometer.

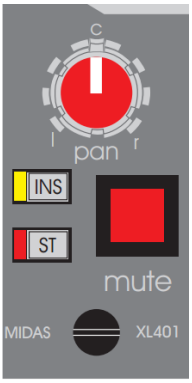


Figure 1.9.: *Panorama knob, mute button, insert enable and master bus routing button of an XL4 input channel*

Panorama The panorama potentiometer or *pan pot* determines the ratio with which the input signal will be routed to the left and the right stereo buses. A panorama potentiometer turned all the way to the left will cause the signal to be sent 100% to the left master bus (respectively group buses with uneven numbers) and 0% to the right

(or even-numbered group buses) and vice versa. The distribution in between is determined by the *pan rule* or *pan law* which defines the level attenuation of left and right signals at the center point. The predominant law in mixing desks is -3 dB at center; so the stereo signal will maintain consistent power across the whole panorama range.

Bus selection An array of buttons allows the operator to choose the buses the signal will be routed to.

Mute and PFL/Solo Each channel offers two additional buttons: the mute button inhibits the input signal to be sent to any bus. The PFL (*Pre Fader Listening*) splits the signal right before the fader and routes it onto a special bus that is fed directly to the console's headphone amplifier and enables the operator to rehearse isolated signals while the overall mixes are unaffected. Both buttons are typically accommodated with LEDs indicating whether they are pushed or not.

Fader On the bottom end of a mixing desk one can find an array of faders. The fader is effectively a sliding potentiometer that enables the operator to control the level the signal is sent to the mix buses (and post-fader auxiliary buses).

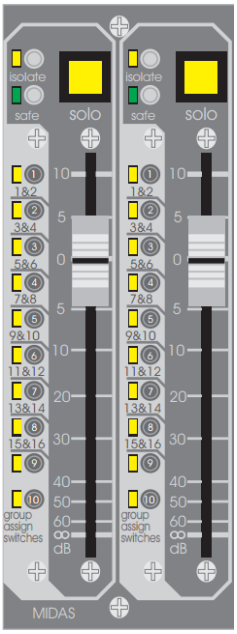


Figure 1.10.:
Faders, solo button and routing selection buttons of an XL4 input channel

1.2.1.2. Buses

Consoles feature a number of buses that enable the grouping of various input signals, offer similar, albeit not as extensive signal processing capabilities as input channels and partly allow subsequent routing to other mix buses. Control elements of the buses traditionally can be found on the right-hand side of a console or in the center of larger-frame consoles. They are easily distinguishable from the input channels by a slightly different potentiometer layout, a different color of the fader caps and other minor differences.

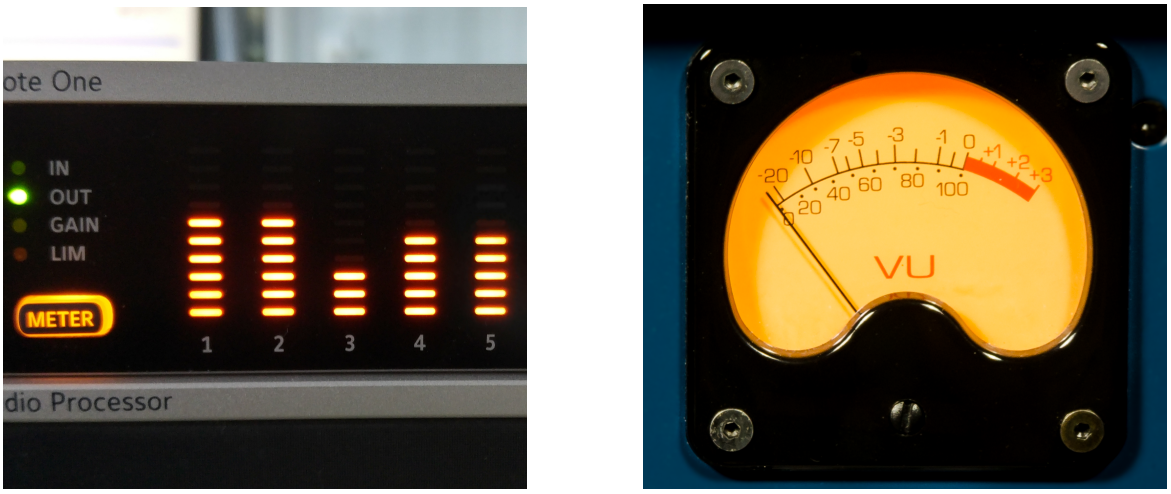
Auxiliary buses As mentioned before, auxiliary buses are a way to provide submixes for monitoring purposes or to feed external signal processors with input. They often provide little to no signal processing capabilities and are limited to an insert point and a master volume control potentiometer. Aux buses are monophonic in the majority of cases, but feature-rich consoles sometimes provide the ability to stereo-couple certain auxiliaries. A rerouting of auxiliaries onto other buses is uncommon, they are usually sent directly to the board outputs.

Group buses Group buses, or groups offer the possibility of submixes. A common use case would be to logically group instruments/signals to buses, e.g. the drum group, the guitar group, the vocal group, etc. This offers two advantages: via insert points it is possible to process several inputs at once with an outboard processor. Also, the operator can adjust the ratio in level between those groups with one (or two for stereo groups) without having to worry to inadvertently change the ratios within one group. Groups can subsequently be routed to the master bus as well as auxiliaries and offer an equalizing section as well, although compared to the input equalizers they commonly offer a reduced set of functionality. A group bus is monophonic and offers a panorama potentiometer as well. Like with input channels, a fader at the bottom of the group section is the final instance to control the level at which the group signal is routed to the master bus.

Master bus The master bus is made up of the left and the right bus (and sometimes a third *center* or *mono* bus) and its main purpose is to control the overall volume of the mix as well as providing an insert point for signal processing that is relevant for the whole sum of signals. That could be a limiting processor that makes sure the volume is compliant with local noise protection laws or an equalizer that counteracts the acoustical shortcomings of the control room and/or the p.a. speakers. Other than that, master buses offer limited control elements besides a fader and the space where one might expect the buses' equalizer section is usually occupied by special control elements like the headphone gain potentiometer and the volume knob for the talkback microphone.

1.2.1.3. Output section

Mixing consoles offer several outputs that are equally found at the rear end. Besides the obvious outputs of the master and auxiliary buses, all input channels and group buses provide an additional output, a so-called 'direct out'. This enables multi-track recording of all channels individually (respectively group-wise).
Another group of outputs are the matrices. They are driven by a matrix of potentiometers (hence the name) located above the controls for group and master buses. Matrix outputs are



(a) Peak Programme Meter **(b)** VU Meter

Figure 1.11.: *Peak Programme Meter and VU Meter*

utilized to provide additional mixes that usually don't require a lot of attention during the performance/recording, like routing the L/R feed to a stereo recording device or routing the vocal group towards sound systems located in other rooms of the venue to make sure important announcements are heard everywhere in case of emergency.

1.2.1.4. Metering

Since it is not feasible for the mixing desk operator to judge every signal solely with his ears, desks provide visual feedback about the in- and outgoing audio signals.

Peak programme meter (PPM) The PPM is a chain of LEDs indicating the current peak value of a signal and can be found at the beginning as well as the end of an input channel (often just a single LED for the former to signal overload of the input amplifier) and at each bus output.

Volume Unit Meter (VU meter) The electromechanical VU meters display the perceived loudness of the buses. Due to the needle's inherent inertia, the information displayed is indifferent towards short peaks as it integrates the signal over 300 ms [5].

1.2.2. Outboard Gear

There are several other signal processing applications and effects that play essential roles in most mixing and recording setups. These are not part of an analog mixing desk mainly for economical reasons: Since they are only applied to a couple of assorted signals, equipping every input channel or bus with such a unit would be wasteful while driving the selling price unreasonably high. Thus, the vast majority of these products are packaged as 19" rack-mount, of which several different devices are assembled into one or more side racks to be operated in close proximity to the console.
When talking about outboard gear, in general the distinction is made between *processors* and *effects*. The former replaces an incoming signal with the processed one. The processor is looped into the signal chain by connecting it with the aforementioned insert point. An effect is not supposed to replace a signal but rather to enrich a signal or multiple individual signals at once. It is connected in parallel to the signal chain. The most common way is to connect an auxiliary bus to the effect input while feeding the unit's output back to a stereo input of the console.
An outboard device is not inherently a processor or an effect by this definition, this is solely determined by the way the device is integrated into the signal chain. However, based on their intended impact on the incoming signal, specific devices are predestined - but not limited - to be utilized as either an effect or a processor predominantly. Thus, in the following section no hard distinction between those terms will be made. The following non-exhaustive list describes the outboard devices that are encountered the most frequently in mixing and recording setups. Although outboard equipment is not necessarily analog, it is presented in this section anyways since digital consoles do not rely on it much if at all, as will be explained in section 1.2.3.

1.2.2.1. Graphical Equalizers

Those processors have a function similar to the parametric channel equalizer in that they alter the spectral signature of the incoming signal. This device offers a bank of peak filters with fixed center frequency covering the whole audible spectrum, usually 31 bands with a spacing of a major third between adjacent bands ('1/3 octave equalizer'). The idea is that the vertical sliders controlling each bank's gain are are arranged in a way that their positions resemble the amplitude response [1], hence the name 'graphical' equalizer.
While a parametric equalizer is designed to deal with the insufficiencies of the signal source

[1] this can be misguiding however, since it does not take into account the bandwidth of individual bands. Setting three adjacent sliders to +6 dB for example could result in the center frequency of the mid slider being boosted by a value anywhere between +6 and +18 dB, depending on whether the filter is implemented as a simple RLC/gyrator network or a more sophisticated *Constant Q* design. For more details refer to [6]

Figure 1.12.:
Dynamic range processors. From top to bottom: UREI 1178 peak limiter, Valley People Dynamite stereo compressor, Drawmer DL251 stereo compressor/limiter

such correcting a microphone's frequency response and to (de-)emphasize individual signals within a mix, the graphical equalizer is supposed to compensate the deficits of the signal chain's end: The nonlinear responses of loudspeakers, headphones and finally the room itself in where the program is rehearsed. For speakers directed at the audience or the operator, the equalizer is used to achieve a more balanced spectrum. For the performing artists' foldback monitors, the equalizer has the additional task to combat audio feedback by attenuating those frequencies most susceptible to feedback.

1.2.2.2. Dynamics

Dynamic range control processors, or short *dynamics*, are used to limit or extend the dynamic range of signals. They can be distinguished by the range they are active in and the purpose they are serving and can be split up in three different groups: The limiter, the compressor and the expander/noisegate. They start operating once a signal exceeds (limiter, compressor) or is below (expander/noisegate) a certain level, the **threshold**. When talking about dynamics, one needs to discern the main signal path that gets weighted with the gain reduction factor from the signal sensing path, the side chain, that determines the amount of the gain reduction. This path is driven by the input signal as well and can be weighted with a filter before its peak or averaged value is evaluated. Many dynamic processors enable the sidechain to be driven by an entirely different signal, too. This advanced technique is not subject of this thesis.

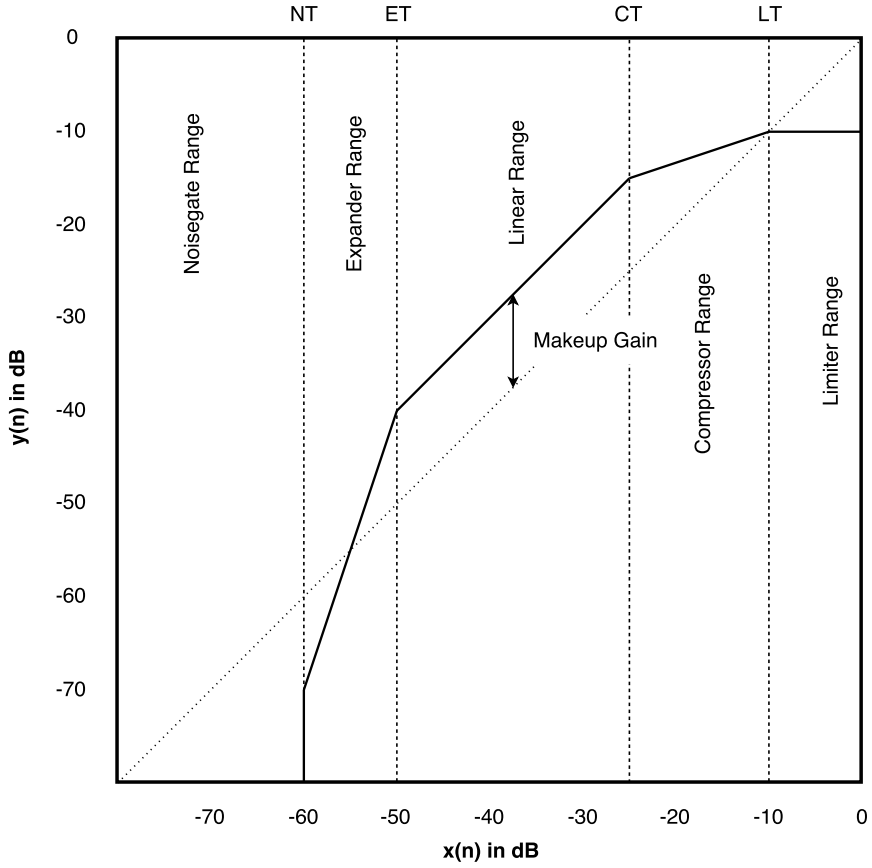


Figure 1.13.:
Static characteristic of a dynamic range processor. Dotted line: input level, solid line: output level.)

Limiter As the name suggests, the limiter has its purpose in limiting the output of a signal to a set threshold. There are two main reasons to limit a signal: First, as a safety measure to prevent sudden peaks in level to damage loudspeakers and ears. Second, they enable the possibility to further increase the loudness of a signal without having peaks excess any maximum level imposed by the underlying analog or digital recording system as well as any broadcast union standards (also known as the **Loudness War**[2]. The gain reduction is determined by the immediate peak value of the signal. Hard limiting achieves this by **clipping** any excess signal without compromises which causes audible distortion. Soft limiting processes the signal already before exceeding the threshold. This happens either by means of a curved

[2]A recent and famous example of excessive limiting is the album *Death Magnetic* from the band *Metallica*, which gathered generally unfavorable reviews from fans and critiques for this reason[7]

clipping which distorts the signal but does not clip it or by low pass filtering of the gain reduction and applying it to the delayed signal. The latter is only possible with a digital limiter due to the need of a delay. Soft limiting mitigates audible distortion but does not necessarily fully cancel it.

Compressor Similar to a limiter, a compressor reduces an incoming signal's level as it exceeds the threshold based on the ratio. A ratio of 2 means that for each two dB that an input signal is above the threshold, the output signal will be one dB above. Loud signal parts will be attenuated while sections lower in volume will be unaffected. After that, the overall level is brought up again with the **makeup gain**. The result is a signal whose perceived loudness is increased while the maximum level stays the same.
A compressor can be used for example to bring softly spoken syllables on par with those expressed in a firm tone within the same word. It also can be used to alter the envelope curve of a percussive signal, for example by letting the transients of a drum beat pass through but compressing the trailing decay portion of the sound. Both applications have very different requirements to a compressor's response speed. This introduces two more important parameters: the attack and the release time. The attack time determines, how quickly the gain reduction will be in full effect once the incoming signal surpasses the threshold, the release time states the time constant with which the gain reduction reverts back to zero once the signal is below the threshold again. Poor setting of attack and/or release time can result in audible processing artifacts that can be described as unnatural 'pumping'.
Compressors rarely operate based on the peak value but rather on sensing an averaged signal function, the root mean square (RMS).
Although even with the advent of digitization in professional audio, digital compressors rarely delay the main signal to catch brief peaks above threshold. There are several reasons for this:

- Unlike a limiter, a compressor's task is not to protect equipment or to make sure that a certain threshold is under no circumstance exceeded, but it is to increase a signal's loudness. Since human ears perceive volume by integrating the sound pressure level over several hundred ms [8], short bursts barely affect this perception and thus are tolerable.
- Since time constants are much longer than a limiter's ones, the compensation delay would need to be increased, too. This arises several new problems: Mixing a compressed and delayed signal with an unprocessed one can cause phase cancellation if both signals carry correlating content (I.e. an uncompressed bass drum and a compressed snare drum with the former **bleeding** in the latter's microphone). To compensate this, every channel would need to be retarded by the same amount of time, imposing an exorbitant additional demand for fast and expensive RAM. Also such a

delayed signal can not be fed back to a performing artist's rehearsal monitor since it would inevitably irritate him.

- After decades of listening to analog recordings, the human ear is quite simply accustomed to listening to the characteristic of non-perfect dynamics processors. As such, the demand is high for digital compressors that try to emulate the sound of their analog vintage counterparts.

Expander/Gate An expander is the counterpart to a compressor and further reduces signal levels that are below the threshold. That way, unwanted signal components can be combatted in the time domain without affecting the desired components itself. It shares the same parameters as the compressor with the attack time determining how quickly the expander ceases the gain reduction once the input climbs above threshold level.
A gate is an expander with an infinitive negative slope, i.e. beneath a certain threshold, no signal passes at all.
Like the compressor, an expander senses a signal by its RMS function and like with the compressor, sensible setting of attack and release is required. A slow-reacting expander may miss the transients of an incoming signal while one that operates too fast can cause 'breathing' artifacts by quickly reducing the level and rising it again in short succession.

1.2.2.3. Reverb

Reverb effects were one of the first category of devices profiting from audio digitization. Up to this point, reverberation units were either very delicate (spring reverberators) or very large and heavy (plate reverberators) electromechanical devices.
When digital reverberators were made widely available in the late 70's, they operated on simple delayed feedback and allpass networks but quickly became more sophisticated with the underlying algorithms being kept secret by the manufacturers.
As processors grew more powerful over the years, convolution reverberation started to become feasible where signals are convoluted with the pre-recorded or simulated impulse response of a real or virtual room.
Due to the fact that acoustical sound sources are generally recorded with microphones being put in very close proximity, the picked up signals can sound very dry and unnatural. To mitigate this, reverb is added to make the sound more 'lively'.

1.2.2.4. Distortion

The desire for artificial distortion effects originate from two historical states: Early sound reinforcement and recording was based on vacuum tubes and tape recorders. Both exhibited

Figure 1.14.: *TC Fireworx Multi-Effect Processor*

distinct nonlinear behavior with increasing drive that can be described as a pleasant 'warmth'. Also, before powerful **P.A.** systems were widely available, musicians playing concerts mostly had to take care themselves that they are being heard by the audience. This resulted in guitar players operating their amplifiers well over the intended limits and thus producing a heavily distorted guitar sound. Over the decades, this necessity has become a virtue and is now an integral part of many different music genres such as blues, rock and metal.
It is not uncommon to still see actual tubes used in modern music productions, predominantly part of microphone and guitar preamplifiers. Due to their high price, delicacy in handling and unfavorable form factor, there is demand for alternatives. Numerous devices are available that emulate the sound of those paragons. The nonlinearity can be reproduced digitally by means of a static characteristic or implemented as a Volterra series expansion[9].

1.2.2.5. Multi effect processors

There are numerous other effects and processors that have not been mentioned yet: pitch shift (raising and lowering the original pitch), tremolo(modulation of amplitude), vibrato (modulation of frequency), octaver (synthesis of signals one octave below or above the original signal) are just a few examples. While there are devices that emulate one single effect - especially in form of pedal effects for guitarists - , their use in general is not as common as the previously mentioned ones. Thus it is hardly economical for most music, broadcast and film productions to have each of those effects at their disposal as a dedicated device.
This can be mitigated through the use of multi effect processors. These digital contraptions offer a multitude of all kinds of effects for almost every purpose and somewhat serve as the 'swiss knife' in the sound engineer's side rack (See figure 1.14).

Figure 1.15.: *The Midas M32 digital mixing console*

1.2.3. Structure of digital mixing desks

Compared to consumer audio, where the compact disk surpassed the sales of vinyl LPs and music cassettes in the early 90's, the triumphal march of digital consoles in recording and live sound production took significantly longer. In fact, the demand for analog consoles from low to high end still exists, although digital boards are vastly more common and accepted at the time of writing this thesis than they have been ten years ago.
This section describes the most significant differences between analog and digital consoles.

New concept of operation The most obvious change is the entirely different layout of the board surface. The possibilities of digitization enabled the departure of the 'one knob for each function' paradigm of analog desks. Instead of having huge arrays of potentiometers and one slider for each channel and each bus, the number of controlling elements has been reduced drastically.
Input channels and mix buses are grouped into layers that can be swapped, motorized volume sliders automatically adjust when switching between layers. There is a single set of incremental encoders to control channel parameters like equalizer, panorama and so forth. The operator has to select the channel to which changes apply first. One or more graphical displays give information about the current state of parameters and provides additional visual information like the actual amplitude response of the parametric equalizer.

Routing A digital board's routing engine is powerful and much more flexible than an analog desk's capabilities. There is a sharp distinction between buses and outputs which are not required to be the same in numbers. The operator is free to route any input signal to any bus and he can route any bus to any output connector, internal effect engine, or fold it back to a stereo input without having to plug any cables.

Audio processing The heart of a digital board is generally one or more DSPs, although some manufacturers utilize FPGAs as the processing engine.
24 bit/96 kHz can be regarded as an industry standard for the AD/DA conversion, although the internal bit resolution is usually significantly higher. Unlike analog sound processing that can suffer from component variance and degradation, the digitally processed sound always stays the same, no matter the temperature or the age of the console.

Connectivity The digital nature of the console offers many options to be interfaced. Many boards offer the possibility to be remotely operated via Ethernet, WiFi or Bluetooth and a laptop or tablet. The need for AD conversion allows to physically dislodge the console from the converters: The incoming signals are converted close to the source by a *digital stagebox*, which is linked to the console via Ethernet or optical fiber. This drastically reduces the susceptibility of power line hum and other unwanted interspersals. In fact, more often than not the whole signal processing takes place in the digital stagebox as well with the console merely sending and receiving control information. Another aspect of the new connectivity is the rise of new digital protocols that enable the easy distribution of audio signals over existing infrastructure, such as the Audio over IP protocols *Dante* and *AES67* (*Ravenna*).

Effects Another revolutionary novelty is the capability of a digital console to include a library of all kinds of effects at the user's disposal. Most of today's consoles come with a complete set of dynamics processors included in each individual input signal's path. A graphical equalizer can be enabled for most if not all outputs (as a quality-of-life feature, the motorized channel sliders can function as the graphical eq's control elements). Dedicated effect engines can be the target and the source of mix buses. If the included effects library is still not sufficient, many consoles offer the possibility to host a multitude of commercial **DAW plugins**.
This effectively renders the need for a dedicated effect side rack moot.

Drawbacks Despite all these benefits, digital consoles have had a tough time getting accepted by operators, systems houses, technical directors and rental companies. This is due to several reasons:

- The learning curve is noticeable steeper. If an operator knows one analog board, he knows all analog boards. The difference is often marginal. Due to the reduced number and increased routing possibilities, digital consoles display more sophisticated concepts of usage and those can vary significantly between manufacturers and models.
- The principle of layering and the paradigm 'one knob, many functions' impose that the operator cannot any longer access any arbitrary function by pressing one single button/turning one single potentiometer. He is rather required to navigate through a set of menus and layers until he can control the desired parameter. This has a negative impact on the workflow, especially if the operator is not fully familiar with the desk (see above).
- The AD- and the DA-conversion supposedly has a negative effect on the sound quality. However, it is highly debatable whether this argument stands the test of time since it arose in the early days of digitalization when converters and DSPs had much weaker specifications than they have now.

2. Signal processing fundamentals

This chapter lays out the mathematical foundation on what the signal processing blocks of this project will rely on.

2.1. Linear processing

The systems introduced in the following section belong to the category of discrete linear time-invariant (LTI) systems that can be described in the time domain by their difference equation and in the spectral domain by their transfer function. The impulse response $h(n)$, the amplitude response $|H(z)|$ and the phase response $\angle H(z)$ visualize an LTI system and give information about its behavior.

2.1.1. Parametric biquadratic filters

The channels' high pass filters and equalizers will be laid out as second order systems. All transfer functions are derived from analog prototypes and transferred to the z-domain via bilinear transformation, resulting in the following equation:

$$H(z) = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a_2 z^{-2}} \tag{2.1}$$

with a_0 normalized to $a_0 = 1$ as a convention.

2.1.1.1. High pass filter

The second-order high pass filter, which rejects unwanted low frequency content, is described [10] by the analog transfer function

$$H(s) = \frac{H_0 s^2}{s^2 + \frac{w_0}{Q_\infty} s + \omega_0^2}. \tag{2.2}$$

$$w_0 = 2\pi f_0$$

H_0 is the pass-band gain, the quality factor of the filter is denoted by Q_∞ and determines the roll-off at f_0. With the Butterworth approximation ($Q_\infty = 1/\sqrt{2}$), the attenuation below f_0 is 12 dB per octave or 40 dB per decade respectively [11, p. 128]. f_0 is the resonance or cutoff frequency of the filter. If the conditions $H_0 = 1$ and $Q = 1/\sqrt{2}$ (Butterworth high pass with unity gain) are met, then $|H(j\omega_0)| = -3$dB.
A Chebyshev or elliptic filter characteristic could provide a narrower transition band. However, in audio signal processing a Butterworth filter is preferred due to its passband linearity [12] and insignificant amount of **ringing artifacts** [13]. These two characteristics are valued higher than a steep roll-off.
The second order Butterworth high pass filter normalized with $w_0 = 1$ and $H_0 = 1$ thus is

$$H_{HP}(s) = \frac{s^2}{s^2 + \sqrt{2}s + 1}. \tag{2.3}$$

2.1.1.2. Shelving filter

To emphasize the low and/or high frequency content of an audio signal, a parametric equalizer can be utilized. It changes the gain g in dB above (*high shelf*) or below (*low shelf*) a certain center frequency $\omega_c = 2\pi f_c$ with

$$|H_{shelving}(j\omega_c)|^2 = \frac{max(|H_{shelving}(j\omega)|^2)}{2}. \tag{2.4}$$

The transition width is defined by the slope S where the relationship is

$$\frac{1}{Q_\infty} = \sqrt{\sqrt{V_0} + \frac{1}{\sqrt{V_0}} \cdot (\frac{1}{S} - 1) + 2} \tag{2.5}$$

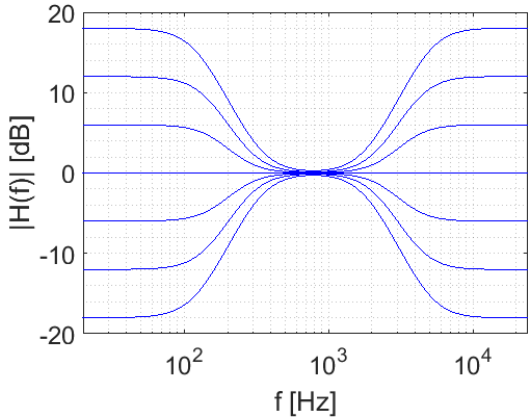
$$V_0 = 10^{\frac{g}{20}} \; g \geq 0$$

between S and the Q_∞ of the filter's underlying high-/low pass.[14]

Amplitude responses in dependence of those three parameters can be seen in equation 2.1.
A signal-boosting low shelf can be obtained with the transfer function

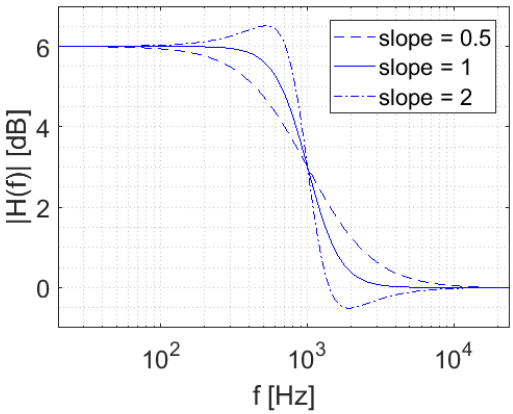
$$H_{LS}(s) = \frac{s^2 + \frac{\sqrt{V_0}}{Q_\infty}s + V_0}{s^2 + \frac{1}{Q_\infty}s + 1}. \tag{2.6}$$

The parameter V_0 is the gain at $\omega = 0$. Variations of V_0 allow boost if $V_0 > 1$ or attenuation if $V_0 < 1$. However, in the case of a cut ($V_0 < 1$), f_0 does not stay constant but becomes dependend on V_0 [15]. To circumvent this, the transfer function 2.6 has to be inverted.



(a)

High shelf ($f_{c,H} = 3$ kHz) and low shelf ($f_{c,L} = 200$ Hz) at $g = 0$ dB, ± 6 dB, ± 12 dB and ± 18 dB



(b)

Low shelf filter with parameters $g = +6$ dB at $f_{c,L} = 1$ kHz and slopes 0.5, 1 and 2

Figure 2.1.: *Parametric shelving equalizer*

A high pass - low pass transformation ($s \rightarrow \frac{1}{s}$) yields the transfer function for the parametric high shelf:

$$H_{HS}(s) = \frac{V_0 s^2 + \frac{\sqrt{V_0}}{Q_\infty} s + 1}{s^2 + \frac{1}{Q_\infty} s + 1}. \tag{2.7}$$

2.1.1.3. Peak filter

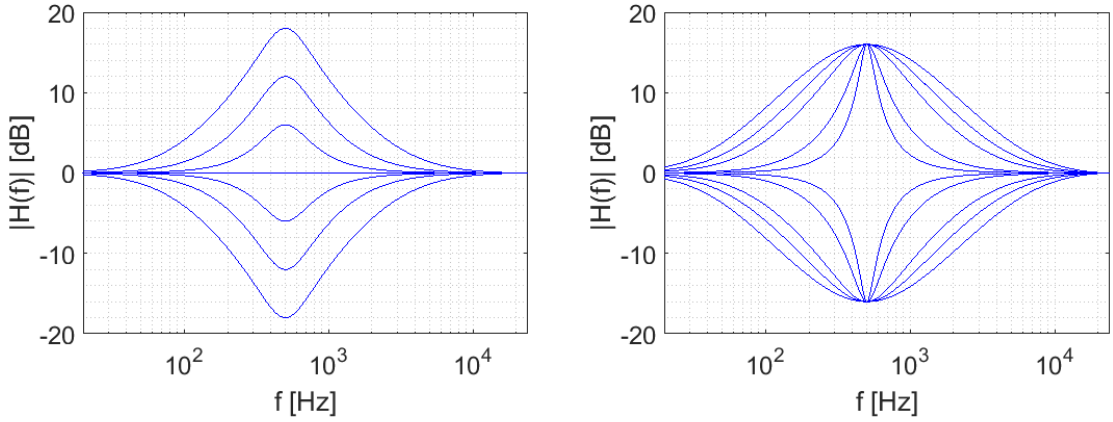
The peak filter is an equalizer designed to boost or attenuate an arbitrary frequency. The adjustable parameters are the center frequency $\omega_0 = 2\pi f_0$ with $|H_{BP}(j\omega_0)| = H_0$, the quality Q_∞ and the gain $g = 20 \cdot log_{10}(H_0)$. Based on the 2nd order band pass filter with constant $H_0 = V_0 - 1$ peak gain

$$H_{BP}(s) = H_0 \cdot \frac{\frac{1}{Q_\infty} s}{s^2 + \frac{1}{Q_\infty} s + 1} \tag{2.8}$$

the peak filter transfer function results in

$$H_{PK}(s) = 1 + H_{BP}(s) = \frac{s^2 + \frac{V_0}{Q_\infty} s + 1}{s^2 + \frac{1}{Q_\infty} s + 1} \tag{2.9}$$

The resulting amplitude responses under varying parameters are shown in 2.2.



(a)
$f_0 = 1000Hz$, $Q_\infty = 1.25$,
Gains: $g = 0$dB, ± 6dB, ± 12dB and ± 18dB

(b)
$f_0 = 1000Hz$, $g = \pm 16$dB,
Q_∞: 0.5, 0.7071, 1, 2.5, 5,

Figure 2.2.: *Parametric peak equalizer*

2.1.1.4. Bilinear transformation

To realize those analog prototype filters in the digital domain with sample rate $f_s = \frac{1}{T_s}$, the bilinear transformation[16, pp. 84-85] helps conveying their transfer functions to $H(z)$:

$$s = \frac{2}{T_s}\frac{z-1}{z+1} \tag{2.10}$$

The normalized analog frequency ω is transferred to the discrete normalized frequency Ω with

$$\omega = \frac{2}{T_s} tan\frac{\Omega}{2}. \tag{2.11}$$

Equations 2.10 and 2.11 map the imaginary axis of the s-Plane onto the unit circle of the z-plane. The trigonometric term in 2.11 causes a nonlinear distortion of the absolute system function. This is a necessity when mapping $\pm\infty$ to $\pm\pi$. By compressing the whole imaginary axis into $\pm\pi$, no aliasing by means of overlapping filter spectra can occur as would be the case with the impulse invariant design method [17, p. 61].
Applying the bilinear transformation to the systems 2.3, 2.6 and 2.9 results in the a and b coefficients for equation 2.1 listed in table 2.1.

High pass filter, $k = tan(\frac{\omega_0 T_s}{2})$

b_0	b_1	b_2	a_0	a_1	a_2
$\frac{1}{1+\sqrt{2}k+k^2}$	$\frac{-2}{1+\sqrt{2}k+k^2}$	$\frac{1}{1+\sqrt{2}k+k^2}$	1	$\frac{2(k^2-1)}{1+\sqrt{2}k+k^2}$	$\frac{1-\sqrt{2}k+k^2}{1+\sqrt{2}k+k^2}$

Peak filter, $k = tan(\frac{\omega_0 T_s}{2})$, boost $V_0 = 10^{g/20}$

b_0	b_1	b_2	a_0	a_1	a_2
$\frac{1+\frac{V_0}{Q_\infty}k+k^2}{1+\frac{1}{Q_\infty}k+k^2}$	$\frac{2(k^2-1)}{1+\frac{1}{Q_\infty}k+k^2}$	$\frac{1-\frac{V_0}{Q_\infty}k+k^2}{1+\frac{1}{Q_\infty}k+k^2}$	1	$\frac{2(k^2-1)}{1+\frac{1}{Q_\infty}k+k^2}$	$\frac{1-\frac{1}{Q_\infty}k+k^2}{1+\frac{1}{Q_\infty}k+k^2}$

Peak filter, $k = tan(\frac{\omega_0 T_s}{2})$, attenuation $V_0 = 10^{-g/20}$

b_0	b_1	b_2	a_0	a_1	a_2
$\frac{1+\frac{1}{Q_\infty}k+k^2}{1+\frac{V_0}{Q_\infty}k+k^2}$	$\frac{2(k^2-1)}{1+\frac{V_0}{Q_\infty}k+k^2}$	$\frac{1-\frac{1}{Q_\infty}k+k^2}{1+\frac{V_0}{Q_\infty}k+k^2}$	1	$\frac{2(k^2-1)}{1+\frac{V_0}{Q_\infty}k+k^2}$	$\frac{1-\frac{V_0}{Q_\infty}k+k^2}{1+\frac{V_0}{Q_\infty}k+k^2}$

Low shelf filter, $A = \sqrt{V_0}$, $\alpha = \frac{sin(\omega_0)}{2} \cdot \sqrt{(\sqrt{V_0} + \frac{1}{\sqrt{V_0}} \cdot \frac{1}{S} - 1) + 2}$

b_0	b_1	b_2
$\frac{A((A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha)}{(A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{2A((A-1)-(A+1)\cdot cos(\omega_0))}{(A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{A((A+1)+(A-1)\cdot cos(\omega_0)-2\sqrt{A}\cdot\alpha)}{(A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$
a_0	a_1	a_2
1	$\frac{-2((A-1)+(A+1)\cdot cos(\omega_0))}{(A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{(A+1)+(A-1)\cdot cos(\omega_0)-2\sqrt{A}\cdot\alpha)}{(A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$

High shelf filter, $A = \sqrt{V_0}$, $\alpha = \frac{sin(\omega_0)}{2} \cdot \sqrt{(\sqrt{V_0} + \frac{1}{\sqrt{V_0}} \cdot \frac{1}{S} - 1) + 2}$

b_0	b_1	b_2
$\frac{A((A+1)+(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha)}{(A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{-2A((A-1)+(A+1)\cdot cos(\omega_0))}{(A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{A((A+1)+(A-1)\cdot cos(\omega_0)-2\sqrt{A}\cdot\alpha)}{(A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$
a_0	a_1	a_2
1	$\frac{2((A-1)-(A+1)\cdot cos(\omega_0))}{(A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$	$\frac{(A+1)-(A-1)\cdot cos(\omega_0)-2\sqrt{A}\cdot\alpha)}{(A+1)-(A-1)\cdot cos(\omega_0)+2\sqrt{A}\cdot\alpha}$

Table 2.1.:

Computation formulas for biquad coefficients.

Refer to [14] (Low shelf and high shelf filter) and [18, p. 43, p. 55] (high pass filter, peak filter)

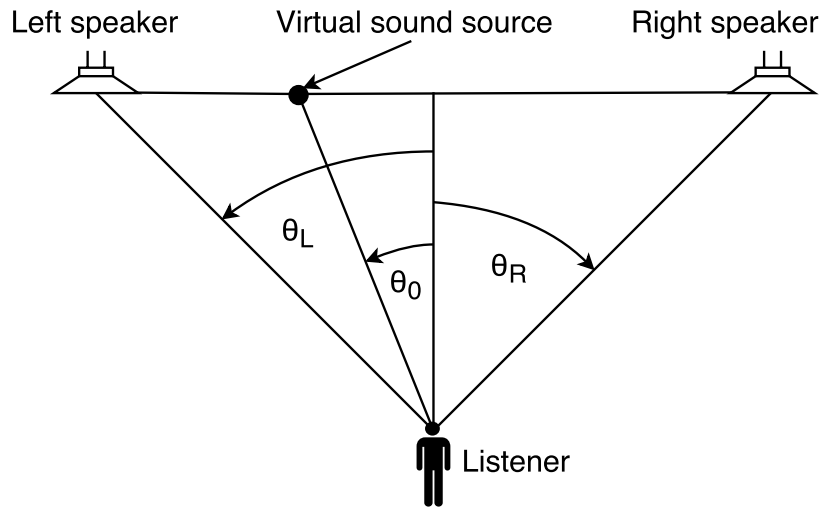


Figure 2.3.: *Azimuth of left and right loudspeaker and of virtual sound source*

2.1.2. Panorama

With a stereo sound reproduction system, it is possible to alter the *virtual sound source*, i.e. the perceived direction from where a sound seems to originate. There are two main principles that enable the human ears to locate a source[3, p. 337]: The first one is the *inter-aural intensity difference* (IID). The ear faced towards the source's direction hears a higher level than the one turned away as the listener's head dampens the incoming sound. From the difference in level, the brain interpolates its origin. The second is *inter-aural time difference*(ITD) and is closely related to the Haas effect[1]: here the brain evaluates the time difference at which sound arrives at the ears.
In this thesis, the panoramic effect will be designed on basis of IID.
Assume a central position in front of a stereo loudspeaker setup, forming an angle $2\theta_L$ where θ_L is the azimuth listener - left loudspeaker. The virtual sound source's azimuth shall be θ_0 with $\theta_R < \theta_0 < \theta_L$ (see figure 2.3). To achieve this, the approximation by the Blumlein law [20] provides the level difference needed:

$$sin\theta_0 = \frac{g_L - g_R}{g_L + g_R} sin\theta_L \tag{2.12}$$

[1] The Haas effect is based on the *law of the first wavefront* and states that if two identical sound sources are presented in short succession, they will be heard as one single sound. Further, the direction of the sound that arrived first will be perceived as its origin - even if the other one is up to 10 dB_{SPL} higher in level [19].

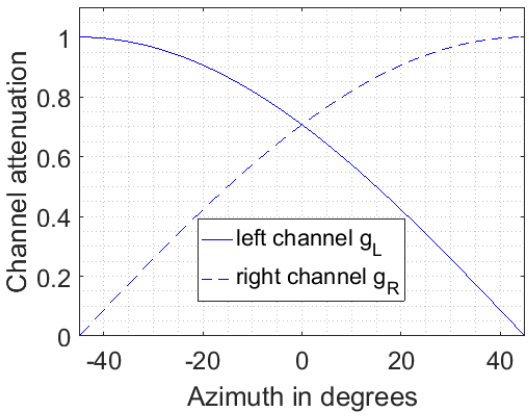


Figure 2.4.:
Channel attenuation g_L (solid line) and g_R (dashed line) over virtual sound source azimuth

This proves to be a valid [21] formulation for broadband signals and low frequencies. The gains can be obtained with

$$\begin{bmatrix} g_L \\ g_R \end{bmatrix} = \mathbf{A}_\theta \mathbf{u} \tag{2.13}$$

where $\mathbf{A}_\theta$ denotes a rotation matrix

$$\mathbf{A}_\theta = \begin{bmatrix} cos\theta_0 & sin\theta_0 \\ -sin\theta_0 & cos\theta_0 \end{bmatrix} \tag{2.14}$$

and $\mathbf{u}$ a unit magnitude stereo signal

$$\mathbf{u} = \begin{bmatrix} \frac{1}{\sqrt{2}} \\ \frac{1}{\sqrt{2}} \end{bmatrix}. \tag{2.15}$$

Assuming a $\theta_L = -\theta_R = 45°$ is advisable: only one of both channels is nonzero in the case of $\theta_0 = \pm\theta_L$ then. The norm $\frac{1}{\sqrt{2}}$ in equation 2.15 makes sure that $|max(g_L)| \leq 1 \; \forall \, \theta_0$ and thus avoids overload in the later implementation. This function follows the -3 dB pan law [22] to keep the sum of both channel's power constant across the complete range of θ_0.

2.1.3. Multirate signal processing

This section is largely based on [23]. Additional references will be declared explicitly.
In the implementation of digital signal processing blocks, it can be advisable to convert the sample rate f_s of the signal in question temporarily in certain situations, either by increasing the sampling rate or by decreasing it. There is no unified nomenclature across literature and terms are used interchangeably. For this thesis, the following definition of terms will be used:

Interpolation	The process of increasing a signal's sampling rate f_s by factor $L \in \mathbb{N}$
Decimation	The process of decreasing a signal's sampling rate f_s by factor $M \in \mathbb{N}$
Multirate processing	Interpolating a signal with sampling rate f_s by factor L, process it at sampling rate $L \cdot f_s$, then decimating it back to f_s. Alternatively: Decimating a signal with sampling rate f_s by factor M, process it at sampling rate $\frac{f_s}{M}$, then interpolating it back to f_s
Resampling	Interpolating a signal with sampling rate f_s by factor L and/or decimating it with factor M to obtain a signal with sampling rate $\frac{k}{m} f_s$([2])

(a) Upsampler **(b)** Downsampler

Figure 2.5.: *Up- and downsampling*

2.1.3.1. Interpolation

The process of interpolation consists of two steps: First, the original signal $x(m)$ at sampling rate $f_{s0} = \frac{1}{T_{s0}}$ with $m = k \cdot T_{s0}$ is *zero-stuffed*, meaning between each value of x, $k - 1$ equidistant zeros are placed. This step is called *upsampling* and yields the upsampled signal $y(n),\ n = k \cdot \frac{T_{s0}}{L}$ and L denotes the upsampling factor:

$$y(n) = \begin{cases} x(\frac{n}{L}) & n = mL,\ m \in \mathbb{Z} \\ 0 & \text{otherwise} \end{cases} \tag{2.16}$$

[2] Resampling is not solely limited to a rational factor but can have an arbitrary real-value ratio. However, interpolation that employs a discrete time filter as described in section 2.1.3.1 cannot be used in this case. Advanced interpolation techniques such as utilization of piecewise polynominal splines or Lagrange interpolation are required. These topics are beyond the scope of this thesis.

In the frequency domain, the zero stuffing causes a compression of the signal spectrum along the normalized frequency axis with the sampling frequency in radians ω_s

$$\Omega = \frac{\omega}{\omega_s} \tag{2.17}$$

by factor L as can be seen in figure 2.6. This is equivalent to the transformation

$$z \rightarrow z^L. \tag{2.18}$$

To distinguish both spectral domains, the following notation is introduced:

z denotes a complex variable in the Z-plane at f_{s0},

z_L is the variable at $L \cdot f_{s0}$.

The signal spectrum is now imaged L-fold in the new sampling rate $L \cdot f_{s0}$. To maintain the original signal and avoid aliasing, further measures are required. Thereby results the second interpolation step, the application of an *interpolation lowpass* $H_{LP}(z_L)$. As the original spectrum is now compressed, its normalized Nyquist frequency $\frac{\omega_{s0}}{2}$ is now located at $\frac{\pi}{L}$. This must be the center frequency ω_0 of the interpolation low pass filter to cut out the alias images while retaining the original spectrum.
It is easy to see that an upsampled signal does not retain the amplitude, since the zero-padding introduces a loss in the average signal power [24]

$$\overline{|y(n)|^2} = \lim_{N\rightarrow\infty} \frac{1}{2N-1} \cdot \sum_{n=-N}^{N} |y(n)|^2. \tag{2.19}$$

This is be mitigated by scaling $y(n)$ by factor L. This scaling can take place at the input signal of the interpolation $x(m)$, the output signal $y(n)$ or by scaling the interpolation filter transfer function $H_{LP}(z_L) \rightarrow L \cdot H_{LP}(z_L)$, whichever results in the most practical implementation.

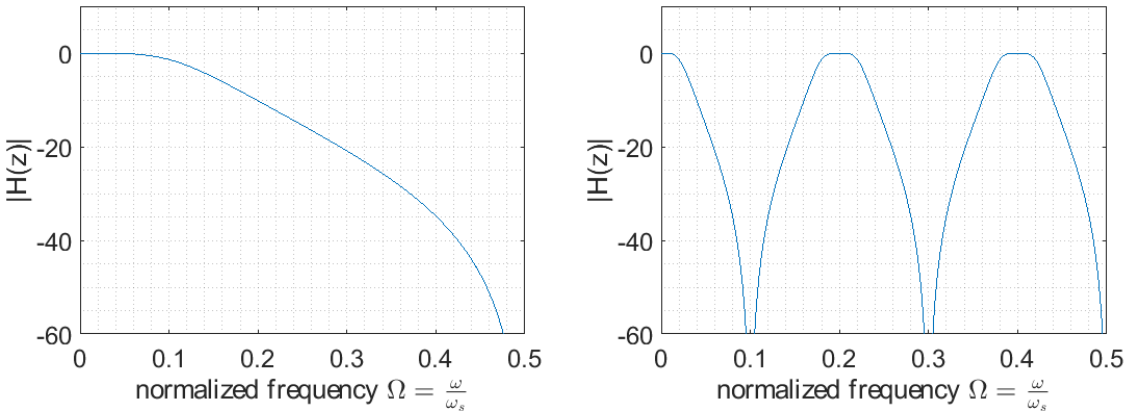
2.1.3.2. Decimation

On a brief glance, decimation appears to be the reversal of interpolation but has a few notable differences. The following section will use following conventions and definitions:

$$z \rightarrow z^{\frac{1}{M}} \tag{2.20}$$

z denotes a complex variable in the Z-plane at f_{s0},

$z_{\frac{1}{M}}$ is the variable at $\frac{f_{s0}}{M}$.



(a) Original spectrum of a lowpass signal

(b) Upsampled spectrum

Figure 2.6.:
Compression of signal spectrum after upsampling by factor $L = 5$ (2nd order Butterworth low pass, $\Omega_0 = 0.25$)

Decimation of a signal $x(n)$ at sampling rate f_{s0} with $n = k \cdot T_{s0}$ to

$$y(m) = x(m \cdot M), \; m = k \cdot M \cdot T_{s0} \tag{2.21}$$

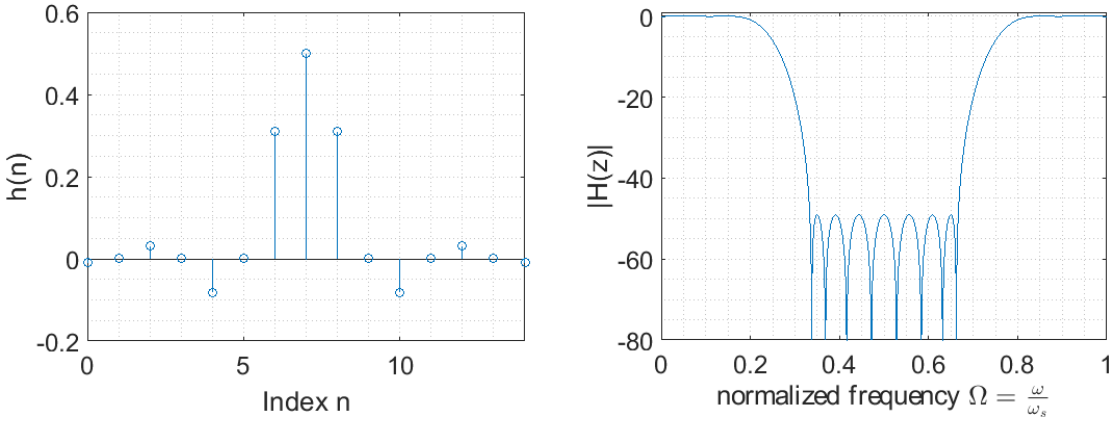
compresses the frequency axis of the signal spectrum

$$z \rightarrow z_{\frac{1}{M}} \tag{2.22}$$

rather than the spectrum itself as with interpolation. Since this means that the Nyquist frequency in $z_{\frac{1}{M}}$ is now located at $\frac{f_{s0}}{2 \cdot L}$ in z, the signal must be low pass filtered by a filter $H_{LP}(z)$ with cutoff frequency

$$f_0 \leq \frac{f_{s0}}{2 \cdot L} \tag{2.23}$$

in order to avoid aliasing. Once the signal $x(n)$ has been band limited, it can be downsampled to $\frac{f_{s0}}{M}$ by applying equation 2.21. Note that no further scaling is required opposed to the interpolation, since there will be no loss of average power when equation 2.23 is fulfilled sufficiently.



(a) Impulse response $h(n)$ **(b)** Spectrum of the filter

Figure 2.7.: *A half band low pass filter with* $N = 15$ *taps*

2.1.3.3. Half band FIR filters

A half band filter is an FIR filter with the special property that the center of its transition region is located at $\frac{f_s}{4}$, where f_s is the input sample rate. Interpolation filters for multirate signal processing purposes are usually implemented as linear phase *type 1*[3] finite impulse response filters. These filters can be designed with the *least squared error criterion*, the *window method*, the *Parks-McClellan Design* which uses the Remez exchange algorithm [25] or others.

The non-recursive filters in this thesis will be designed using the Parks-McClellan algorithm, which approximates a filter according to the parameters passband ripple δ_p, stopband attenuation δ_s and the cutoff frequencies of the passband Ω_p and stopband Ω_s. A detailed explanation of this procedure can be found in [26].

A notable subset of the FIR filter is the half band low pass filter. It is a type 1 FIR whose uneven coefficients $h(n)$ are zero, except for its symmetrical center $h(\frac{N-1}{2})$ which is 0.5 when normalized to $|H(0)| = 1$. This leads to the fact that the center Ω_0 of the transition width b is always

$$\Omega_0 = \frac{\Omega_s - \Omega_p}{2} = 0.25, \tag{2.24}$$

or in words: half the Nyquist frequency of the sampled signal. This makes the half band filter a good choice when implementing a multirate processing stage with an upsampling/-downsampling factor of $K = 2$, where it fulfills the requirement for the cutoff frequency $f_0 = \frac{f_{s0}}{2 \cdot K} = 0.25 f_s$.

The fact that close to half of the filter coefficients are zero effectively halves the number of

[3] Type 1 FIR filters have an odd number N of taps and have a symmetric impulse response around $h(\frac{N-1}{2})$

multiplications per sample. Considering the fact that discrete convolution has a computational complexity of $O(N^2)$, this has a large beneficial impact on computation speed.

2.1.3.4. Polyphase FIR filters

Instead of the common transversal FIR structure, a polyphase filter can be used for interpolation and decimation.

Polyphase filter for upsampling As described in section 2.1.3.1, the N-tap interpolation filter $H(z_L)$ is applied after the upsampling by factor L took place. This order of execution can be changed by converting the filter to a polyphase structure with L phases.
The transversal FIR is divided into L FIRs with $\frac{N}{L}$ taps (apply zero-padding to receive integer results) that are connected in parallel and fed by the input signal $x(m)$. The coefficients of each filter $H_l(z)$ with length $R = \frac{N}{L}$ are

$$h_l(m) = h(\frac{n}{L} + l),\ l = [0, 1, 2, ..., R-1]. \tag{2.25}$$

The filter can be described as

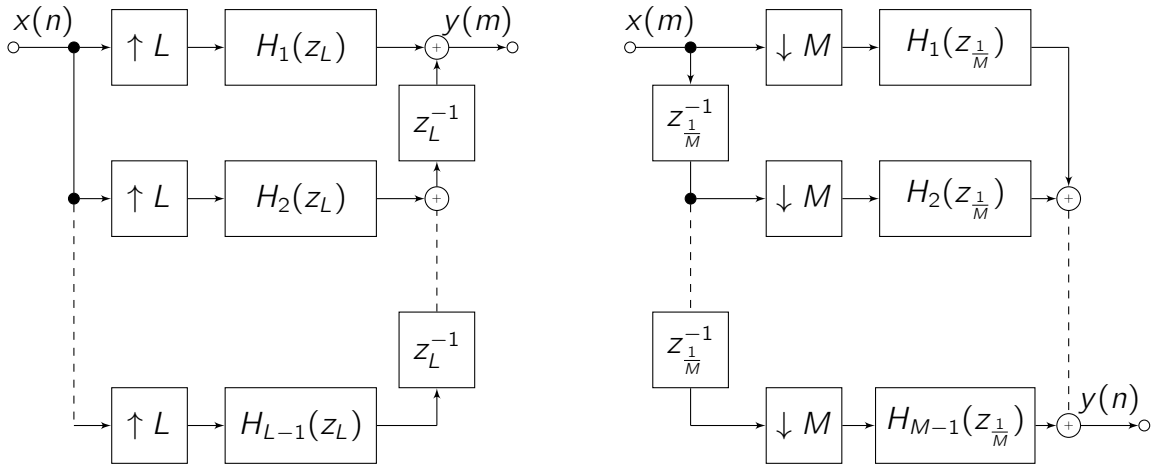
$$H(z) = \sum_{l=0}^{L-1} H_l(z) = \sum_{l=0}^{L-1} z^{-l} \sum_{r=0}^{R-1} h_l(r) z^{-Lr} \tag{2.26}$$

The upsampling then can be represented by a commuter: for each input sample$x(m)$, the L output samples $y(n),\ n = 0, 1, 2, ..., L-1$ are taken from the output of $H_l(z)$, $l = n$.
This reduces the number of multiplications and additions: While the transversal FIR in the z_L domain requires $L \cdot N$ multiplications and $L \cdot (N-1)$ additions, the same filter in the z domain and polyphase structure only needs the computation of N multiplications and $N-1$ additions.

Polyphase filter for downsampling Similarly to equation 2.26, the FIR for decimation by factor M can be represented as a polyphase filter with M filters of length $R = \frac{N}{M}$ by transposing it to the $z_{\frac{1}{M}}$ domain:

$$H(z_{\frac{1}{M}}) = \sum_{m=0}^{M-1} H_m(z_{\frac{1}{M}}) = \sum_{m=0}^{M-1} z_{\frac{1}{M}}^{-m} \sum_{r=0}^{R-1} h_m(r) z_{\frac{1}{M}}^{-Mr}. \tag{2.27}$$

The inputs of those filters are fed by a commuter that distributes the M input samples $x(n-m), m = 0, 1, 2, ..., M-1$ that occur during one period of $\frac{f_{s0}}{M}$ to the filters $H_m(z_{\frac{1}{M}})$. After



(a) Interpolation with polyphase FIR

(b) Decimation with polyphase FIR

Figure 2.8.: *Interpolation and decimation utilizing polyphase filters*

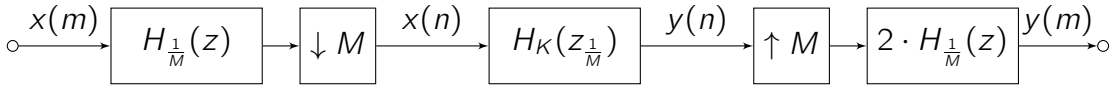
one commuter cycle, the outputs of the M filters are summed up and yield the decimated signal $y(m)$.
As with interpolation, the number of multiplications and additions are reduced by factor $\frac{1}{M}$ to N and $N-1$ respectively. See figure 2.8 for details.

2.1.3.5. Multirate Processing

Multirate processing can be advisable when applying a low pass[4], the *kernel filter*, with cutoff frequency $f_0 << f_s$ and the requirement for a narrow transition band and/or high stop band attenuation. Implementing this system in f_s as an FIR, it requires a large number of taps to fulfill this specification. As an IIR, the kernel filter faces the risk of suffering from limited arithmetic precision and entailing problems like limit cycles, quantization noise and possible instability.
Multirate filtering (see figure 2.9b) can avert these problems. With **Dyadic Cascading**, the sampling rate can be successively reduced. Utilizing half band filters (section 2.1.3.3), the absolute center frequency shifts with each cascade, while the required passband edge frequency is given by the kernel's cutoff frequency. This allows for a very broad transition band for the filter in f_{s0} and thus a comparably small amount of filter coefficients. As the decimation goes on, the transition band needs to narrow down and requires a higher FIR order. However, with decreasing sampling rate, the amount of calculations per time unit sinks as well.
For example, a low pass filter with the following specifications: passband frequency $f_{pass} = 30$ Hz, stopband frequency $f_{stop} = 40$ Hz, sampling frequency $f_{s0} = 8000$ Hz, passbband

[4] or likewise a high pass filter with f_0 close to $\frac{f_s}{2}$!

(a) Increase of samplerate



(b) Sample rate reduction

Figure 2.9.: *Multirate signal processing*

ripple $\delta_p = 0.01$, stopband attenuation $\delta_s = 0.001$ would require $N = 2033$ taps according to the Matlab estimation function *firpmord()*, resulting in over 16 million multiplications per second when implemented with discrete convolution. However, reducing the sampling rate to $f_{s'} = 125$ Hz, the number of required taps drops to $N = 31$ and 3875 multiplications per second. [23] shows that decimation by factor 64 (a dyadic cascade with 6 stages) and then applying the kernel can be achieved with less than 100.000 multiplications per second. Interpolation back to the original sampling rate approximately doubles this number.
Operation at a higher sampling rate can be necessary for highly nonlinear signal processing, such as described in section 2.2.2. Applying harmonic distortion to a discrete signal with a bandwith ranging from 0 Hz to $\frac{f_{s0}}{2}$ inevitably produces aliasing artifacts as harmonics at $f > \frac{f_{s0}}{2}$ fold back to below the nyquist frequency. Interpolating the signal with a factor high enough that any artifacts within the baseband are below a sufficient level avoids this effect.
The process of this is similar to the one before and is illustrated in 2.9a: Instead of decimation, the original signal will be interpolated first. After the nonlinear processing took place, the signal will be decimated back to its original sampling rate.

2.2. Nonlinear processing

Transfer functions are not defined for this class of systems. They are generally represented by a nonlinear function in the time domain. The output of the system is not proportional to changes of the input and its spectrum can contain components not present in the input (and vice versa). In case of a constant nonlinearity, the output signal is enriched with harmonics of the input. Thus, care must be taken that the output signal does not violate the Nyquist-Shannon sampling theorem, even if the input satisfies its requirement.

2.2.1. Dynamics processing

This section is largely based on the work of Udo Zölzer in [11, pp. 227-239] and [18, pp. 95-104]. Additional references will be declared explicitly.
Dynamics processing, the automatic gain control of a signal based on its current peak or averaged level is based on an amplitude detection algorithm, the *envelope follower*. If this follower detects a level above or below a certain predefined **threshold**, the signal gain gets changed in proportion to the threshold excess, the compression factor *ratio* R, defined as $R = \frac{\Delta L_{in}}{\Delta L_{out}}$. ΔL_{in} denotes the threshold excess of the input signal, ΔL_{out} the resulting excess of the output.
A derived item is the slope S:

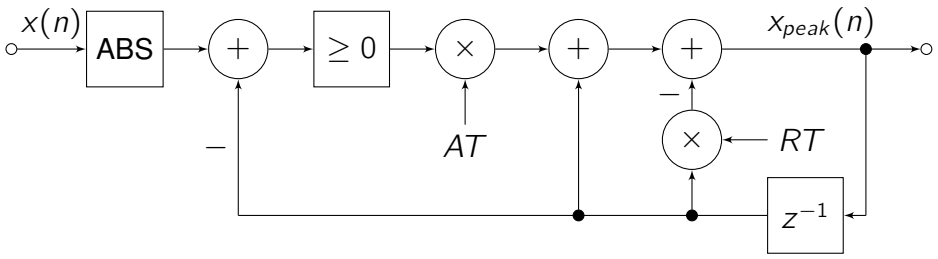
$$S = 1 - \frac{1}{R} \tag{2.28}$$

The four types of dynamics processor are *limiter, compressor, expander* and *noisegate*. They differ from each other by how the input signal is evaluated, whether levels below or above threshold trigger the gain reduction and where the ratio is located, see table 2.2.

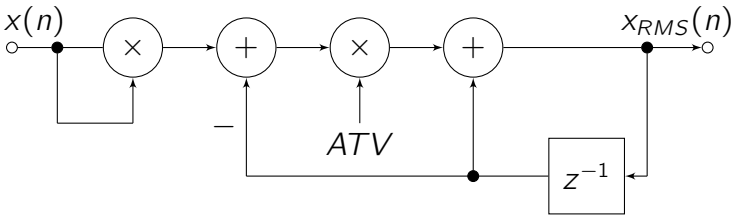
Limiter		R	$= \infty$		LS	$= 1$
Compressor	$1 <$	R	$< \infty$	$0 <$	CS	$= 1$
Linear		R	$= 1$		S	$= 0$
Expander	$0 <$	R	< 1	$-\infty <$	ES	$= 0$
Noise gate		R	$= 0$		NS	$= -\infty$

Table 2.2.: *Ratio and slope of dynamics processors*

These ratios result in a static characteristic that can be seen in figure 1.13. The slopes are tagged with their according dynamic range to make distinction easier: LS denotes the limiter slope, CS the compressor slope, ES the expander slope and NS the noise gate slope.



(a) Peak measurement block diagram



(b) RMS measurement block diagram

Figure 2.10.: *Level measurement*

2.2.1.1. Level measurement

To measure the current level of the input signal $x(n)$, the systems in figure 2.10 are used. The values AT, RT and AVT (see equations 2.30 to 2.32) are dimensionless coefficients that are based on the attack and the release time constants t_{at} and t_{rt} of the peak measurement $x_{peak}(n)$ and the averaging time t_{av} of the RMS measurement $x_{RMS}(n)$ which equal the rise time of the first order system

$$g(t) = 1 - e^{-\frac{t}{\tau}}$$
$$0.1 = 1 - e^{-\frac{t_{10}}{\tau}}$$
$$0.9 = 1 - e^{-\frac{t_{90}}{\tau}}$$

$$t_a = t_{90} - t_{10} = ln(0.9/0.1)\tau = 2.2\tau \tag{2.29}$$

The discrete coefficients sampled with $f_s = \frac{1}{T_s}$ thus are calculated with

$$AT = 1 - e^{-2.2T_s/t_{at}} \tag{2.30}$$

$$RT = 1 - e^{-2.2T_s/t_{rt}} \tag{2.31}$$

$$AVT = 1 - e^{-2.2T_s/t_{av}}. \tag{2.32}$$

These times should not be confused with the attack and release times of the automatic gain control as described in section 2.2.1.2.

It is suggested [18, p. 99] for limiter peak measurement $x_{peak}(n)$ to set the attack and release time constants to $20\mu s \leq t_{at} \leq 10ms$ and $1ms \leq t_{rt} \leq 5000ms$ respectively. For the RMS measurement $x_{RMS}(n)$ averaging time $t_{av} = 200ms$ is recommended [27, p. 40].
The difference equation for the peak measurement is

$$x_{peak}(n) = (1 - AT - RT) \cdot x_{peak}(n-1) + AT \cdot |x(n)| \tag{2.33}$$

and for the RMS measurement

$$x_{RMS}(n) = (1 - AVT) \cdot x_{RMS}(n-1) + AVT \cdot x^2(n). \tag{2.34}$$

This type of peak evaluation meter is called *Sample Peak Programme Meter*, or *SPPM*. It does not take into consideration the fact that sampling can mask the true signal amplitude by up to -3 dB [5]. To mitigate that error, a *True Peak Programme Meter (TPPM)* can be employed. A TPPM oversamples the signal by a factor of 4 to decrease the deviation of the sampled value from the actual analog amplitude down to a few tenths of a dB [28]. A true peak evaluation will not be part of this thesis, however. The operator does need to take this into account when deciding the threshold at which the limiter shall start to operate.

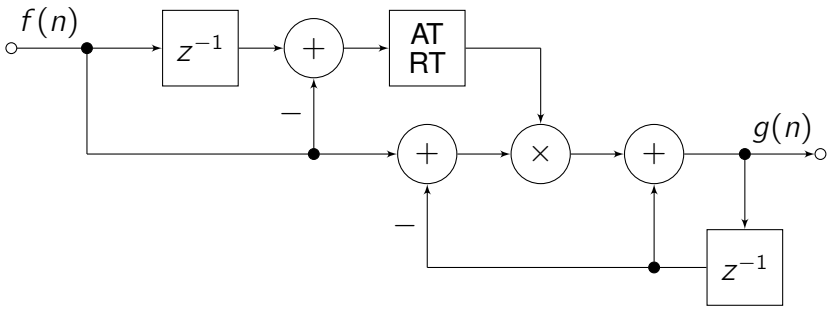


Figure 2.11.: *Smoothing filter block diagram*

[5] consider the sampling with $T_s = 1/f_s$ of a continuous-time signal $f(t) = A \cdot sin(2\pi f_0 t)$ with $A = 1$ and $f_0 = f_s/4$. If the sampling took place at $t = k \cdot T_s + 0.5T_s$, the sampled values would be $[0.707, 0.707, -0.707, -0.707, ...]$

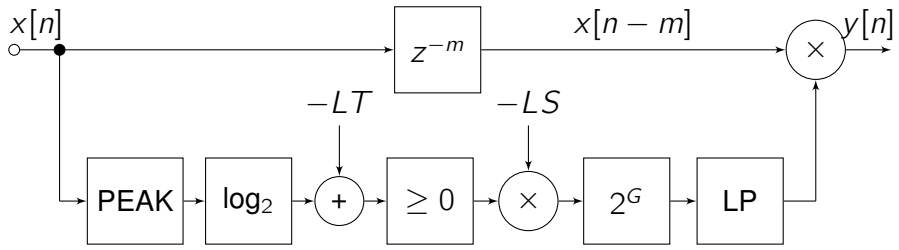


Figure 2.12.: *Limiter block diagram*

2.2.1.2. Smoothing filter

To realize parametric attack and release time of the automatic gain control $f(n)$ as specified in table 1.2, the system depicted in figure 2.11 is used. The output $g(n)$ is the smoothed gain factor. It follows the difference equation

$$g(n) = (1 - k) \cdot g(n - 1) + k \cdot f(n) \tag{2.35}$$

$$k = \begin{cases} 1 - e^{-2.2T_s/t_{attack}} & g(n) > g(n-1) \\ 1 - e^{-2.2T_s/t_{release}} & \text{otherwise.} \end{cases}$$

Where t_{attack}, $t_{release}$ denote the attack and release time and T_s the sampling period. The block *AT/RT* in the block diagram is a decision making entity that sets the appropriate system coefficient depending on whether $f(n-1) \leq f(n)$ (coefficient is AT) or $f(n-1) > f(n)$ (RT).

2.2.1.3. Limiter

The complete limiter block diagram can be seen in figure 2.12. The main signal path gets delayed by m samples. In the side chain, the peak gets evaluated (PEAK) according to section 2.2.1.1. LT is then substracted from the peak value's 2's logarithm. LT is the limiter threshold in the 2's logarithm domain. If the threshold is exceeded (≥ 0), it is multiplied with the negative limiter slope $-LS = -1$ (refer to 2.2) and transferred back into the linear domain (2^G). The gain factor now gets filtered by the smoothing low pass according to 2.2.1.2 (LP). The delayed input signal finally is weighted with the output of the smoothing filter.

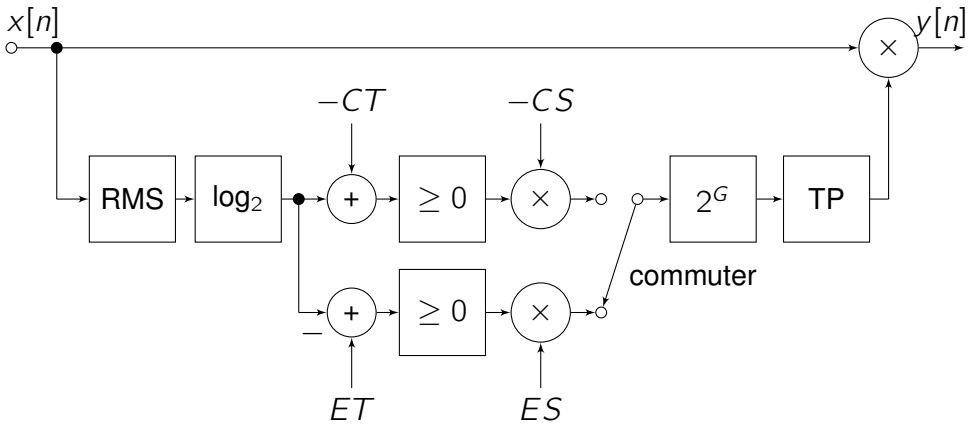


Figure 2.13.: *Compressor/Expander block diagram*

2.2.1.4. Compressor/Expander

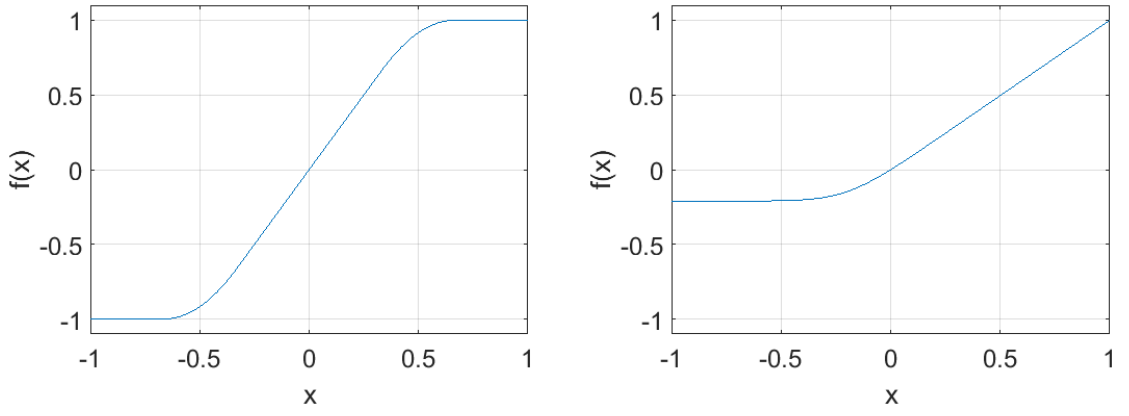
To enable an efficient computation, a compressor and an expander - which do not operate on intersecting input level ranges, see figure 1.13 - can be combined into a single processing unit. A range detector decides whether the upper or the lower branch of the side chain in figure 2.13 comes into effect.

2.2.2. Harmonic distortion and overdrive

Distortion is commonly measured and quantified as *total harmonic distortion*, or THD. It is the harmonic content of a waveform compared to the waveform's RMS value and calculates [29]

$$THD_R = \sqrt{\frac{\sum_{n=2}^{\infty} I_n^2}{\sum_{n=1}^{\infty} I_n^2}} \tag{2.36}$$

where I_1 is the RMS value of the fundamental frequency f_0 and I_k the RMS value of its k-th harmonic frequency $k \cdot f_0$ [6].
The musical application of distortion from valve amplifiers and overdrive effects can be found on virtually every recording that features electric guitar tracks. These distorting effects are based on various characteristics. In this thesis, two different kinds of digital overdrive are presented, the symmetrical soft clipping based on the Schetzen formula and a tube emulation based on the exponential function. Their static characteristics are displayed in figure 2.14.



(a) Symmetrical soft clipper, $k = 1$ **(b)** Tube distortion, $k = 2, Q = -0.2, dist = 8$

Figure 2.14.: *Static characteristic of distortion/overdrive effects*

2.2.2.1. Symmetrical soft clipping

This clipping algorithm aims to emulate the saturation curve of an analog tape recorder while maintaining efficient computability. It is based on the Schetzen formula $o_{symm}(x)$ [18,

[6] More commonly found is the THD_F definition where the denominator consists solely of the fundamental's RMS value I_1. However, to stay coherent with section 4.4.2.5, THD_R analysis is used here, as the distortion calculation of the *System Two* audio analyzer is based on this definition as well

p. 118]:

$$o_{symm}(x) = \begin{cases} 2x & 0 \leq x \leq \frac{1}{3} \\ \frac{3-(2-3x)^2}{3} & \frac{1}{3} < x \leq \frac{2}{3} \\ 1 & \frac{2}{3} < x \leq 1 \end{cases} \tag{2.37}$$

Weighting x with a constant drive factor dr before applying the function $o(dr \cdot x)$ enables control over the degree of saturation. This distortion enriches the original signal with uneven harmonics.

2.2.2.2. Tube clipping

The static characteristic of the tube clipping emulation is based on a triode, a vacuum tube commonly found at the input gain stage of a tube-based guitar amplifier. This asymmetric emulation algorithm is based on several parameters:

dr	the drive factor
Q	The operation point or zero offset, controls the linearity of low input signals
$dist$	The character of the distortion. Higher values provide harder distortion, i.e. higher THD.

The tube clipping function $o_{tube}(x)$ is

$$o_{tube}(x) = \frac{dr \cdot x - Q}{1 - e^{dist(dr \cdot x - Q)}} + \frac{Q}{1 - e^{dist(Q)}},\ Q \neq 0,\ dr \cdot x \neq Q \tag{2.38}$$

If the requirements $Q \neq 0,\ x \neq Q$ are not fulfilled, simplifications have to take place:

$$o_{tube}(x) = \frac{dr \cdot x}{1 - e^{-dist(dr \cdot x)}},\ Q = 0, \tag{2.39}$$

$$o_{tube}(x) = \frac{1}{dist} + \frac{Q}{1 - e^{dist(Q)}},\ dr \cdot x = Q. \tag{2.40}$$

These equations provide a signal distortion rich in both even and uneven harmonics. The resulting signal has to be high pass filtered to reject the dc component.

3. Simulation

Before implementing the processing blocks outlined in chapter 2, extensive simulation is required. The simulation files are provided in the folder *'Simulation Suite'* on the CD-ROM accompanying this thesis.

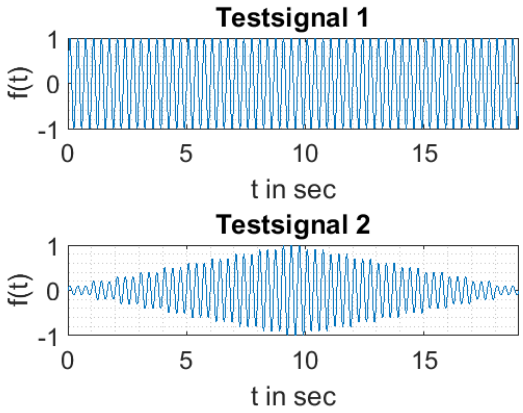
3.1. Concept and goal of the simulation

As the signal processing application as shown in figure 1.2 is highly modular, the author will refrain from modeling the complete system but rather just simulate and verify single blocks. This is realized in Matlab which does not only offer powerful numerical tools for the purpose of the simulation, but also enables graphic preparation of its results.
Simulation serves several purposes: First, it validates the concepts introduced in chapter 2. Further, successful simulation provides a high level reference implementation as a guideline for reimplementing the simulated blocks on the target host in a low level programming language such as C. The results can then be compared to the simulation outcome to prove the its correctness and to reveal any problems of the implementation.
In this specific case, the simulation needs to address the following points:

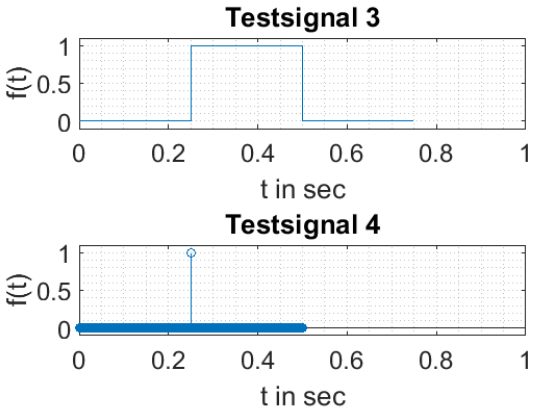
- formulas in table 2.1 need to be validated, i.e. ensure that the center/cutoff frequencies, the gain and the Q/slope are appropriate. Also, the filters may not become unstable when setting extreme frequencies with single precision coefficients
- squared sums of left and right signals must stay equal across the whole panoramic range
- dynamic range control must display the correct static behavior
- timing of the gain control must represent the desired rise and fall times set by attack and release
- overdrive equations have to be validated and their impact in the frequency analyzed
- halfband FIR filters for multirate signal processing have to be specified and designed

- the impact of interpolation on nonlinear processing and suppression of **aliasing** images needs to be examined

The script *testsignal.m* provides two sinusoid, an impulse and a boxcar function signal to aid the simulation. Their frequency and duration respectively can be changed by a function parameter, see figure 3.1



(a)
Sinusoid test signals with parameter $p = 3$ ($p \equiv$frequency in Hz)



(b)
Boxcar and impulse test signals with $p = 12000$ ($p \equiv$ no. of padded zeros and step length)

Figure 3.1.: *Test signals used for simulation*

3.2. Results

3.2.1. Parametric biquad filter simulation

Three functions, *highpass.m*, *peak.m* and *shelving.m* calculate the coefficients of a 2nd order recursive digital filter based on the formulas given in table 2.1.
The Matlab script *filtercascade.m* enables the user to specify the desired parameters of a filter cascade consisting of one high pass filter, one low shelf filter, one high shelf filter and two peak filters (see listing 3.1). The according coefficients are then calculated, converted to single precision (32 bit float) and the resulting amplitude response is plotted.

Listing 3.1: *filter_cascade.m: User parameters for filter cascade*

```
% FILTERCASCADE
% Author: Benjamin Sellak
% Last modified: 8.8.2017
%
% generates 32 bit floating-point coefficients
% for a cascade of 5 filters and displays the amplitude response.
clear all; close all;

%****************** User Parameters *************************
f_hp = 40;              % Highpass filter cutoff frequency in Hz

f_s1 = 180;             % Low shelf filter center frequency in Hz
g_s1 = +6;              % Low shelf filter gain in dB
s_s1 = 1;               % Low shelf filter slope

f_s2 = 6000;            % High shelf filter center frequency in Hz
g_s2 = -3;              % High shelf filter gain in dB
s_s2 = 1;               % High shelf filter slope

f_p1 = 400;             % 1st peak filter center frequency in Hz
g_p1 = -6;              % 1st peak fliter gain in dB
q_p1 = 2.5;             % 1st peak filter q factor

f_p2 = 3000;            % 2st peak filter center frequency in Hz
g_p2 = +3;              % 2st peak fliter gain in dB
q_p2 = 0.7;             % 2st peak filter q factor
%****************** User Parameters END**************************
```

A critical parameter is the high pass filter cutoff frequency f_0, which can go as low as 20 Hz as specified in section 1.1.1.1. This equals to $\frac{1}{2400}$ of the sampling frequency $f_s = 48000$.

Filter type	gain g	cutoff/center frequency f_0	Q/Slope
high pass	n.a.	40 Hz	n.a.
low shelf	+6 dB	180 Hz	1
low-mid peak	-6 dB	400 Hz	2.5
mid-high peak	+3 dB	3000 Hz	0.7
high shelf	-3 dB	6000 Hz	1

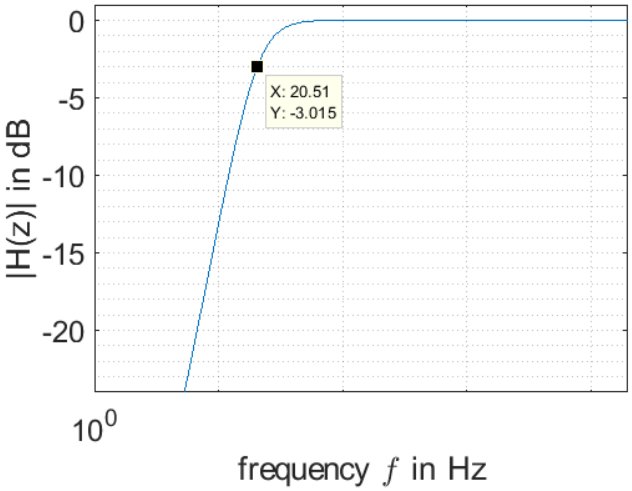
Table 3.1.: *Parametric equalizer coefficients for filtering a bass drum signal*

Since filters with cutoff frequencies $f_0 << f_s$ are especially prone to rounding errors [11, p. 148], it is essential to make sure that a high pass with the minimum allowed $f_0 = 20$ Hz specified results neither in an inaccurate amplitude response, nor does it become unstable. To verify this, a simulation of the filter response can be seen in figure 3.2a. As can be seen, the amplitude response at the cutoff frequency f_0 is at $\approx 0.5\%$ and within an acceptable range.
A more comprehensive cascade parametrization is given in table 3.1, which is a typical setting for equalizing a bass drum signal: The low end is emphasized by the low shelf filter, as is the attack by the peak filter at $f_0 = 3000$ Hz. The other peak filter is set to a narrow cut at 400 Hz, a frequency range where bass drums often display an unpleasant 'wooden'-sounding resonance. The high pass filter at 40 Hz and the high shelf at 6000 Hz attenuate frequencies in ranges where the bassdrum does not produce sound of its own. This way, the amount of rumble and **bleeding** can be reduced without affecting the desired signal components.
Figure 3.2b depicts the resulting amplitude response and validates the results of the three modules that calculate filter coefficients.

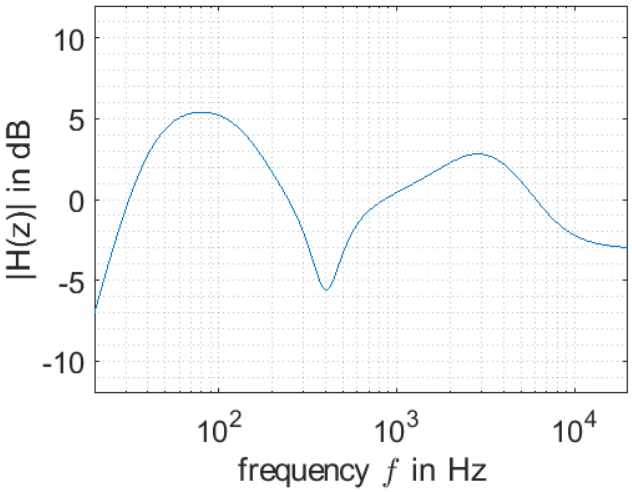
3.2.2. Panorama simulation

The panorama simulation consists of function *panorama.m* and script *panoramatest.m*. The requirements for the system are:

- the system input shall accept an azimuth θ_0 ranging from $\theta_R = -45°$ (signal source located at rightmost direction) to $\theta_L = 45°$ (signal source located at leftmost direction)
- at $\theta_0 = \theta_L$, the left attenuation factor shall be $g_L(\theta_L) = 1$, the right attenuation factor shall be $g_R(\theta_L) = 0$.
- Likewise, for $\theta_0 = \theta_R$, the left attenuation factor shall be $g_L(\theta_R) = 0$, the right attenuation factor shall be $g_R(\theta_R) = 1$.



(a)
High pass filter with $f_0 = 20$Hz, other filters disabled (gain $g = 0$dB)



(b) Filter cascade with parameters corresponding to table 3.1

Figure 3.2.: *Simulation of the biquad filter cascade*

- at $\theta_0 = 0°$, both attenuation factors shall be $g_L(0°) = g_R(0°) = \frac{1}{\sqrt{2}}$.
- the power sum of both gains shall be 1 across all azimuths $g_L(\theta)^2 + g_R(\theta)^2 = 1 \quad \forall \quad |\theta| \leq 45°$.

A run of *panoramatest.m*, as displayed in figure 3.3, shows that those constraints are met with the model presented in section 2.1.2.

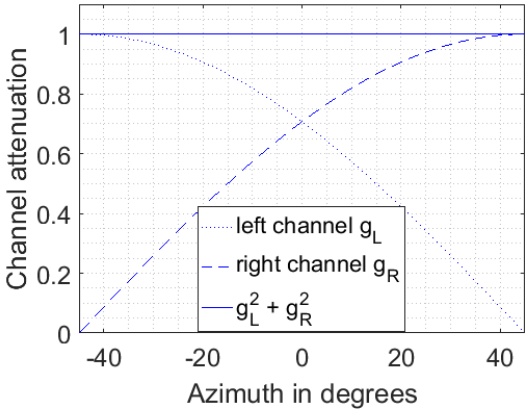


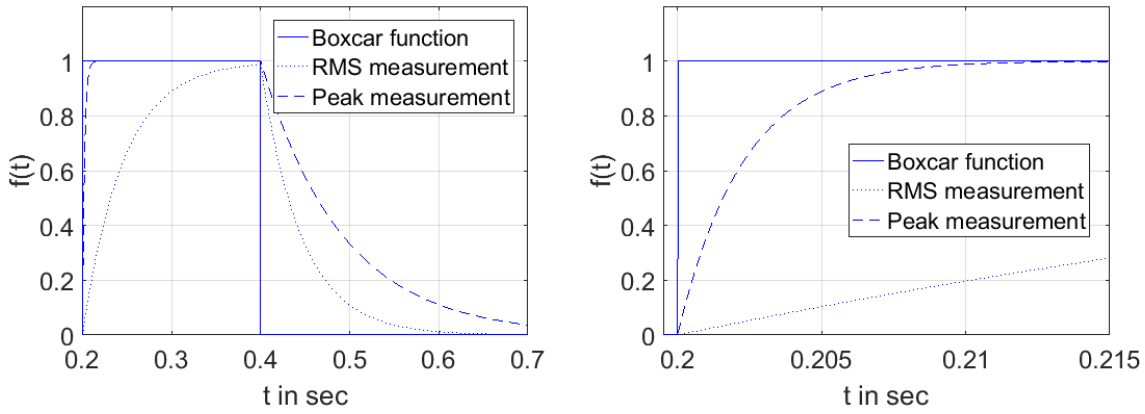
Figure 3.3.:
The simulated panorama functions $g_L(\theta)$ and $g_R(\theta)$ and the power sum of both.

3.2.3. Dynamic range control simulation

The dynamic range control simulation consists of several parts. First, the correct evaluation of peak and RMS levels has to be asserted. Second, a functional compression, expansion and limiting based upon those levels has to be proven. The measurement is handled in the Matlab functions *peakcalc.m* and *rmscalc.m*, the dynamic range control takes place in the functions *limiter.m* and *compander.m*. The simulation of the dynamic range control can be found in script *compressortest.m* and offers user parameters for all thresholds and time constants.

Measurement	$t_{10,R}$	$t_{90,R}$	Δt_R	$t_{90,D}$	$t_{10,D}$	Δt_D
Peak	$0.2003s$	$0.2052s$	$0.0049s$	$0.4096s$	$0.6093s$	$0.1997s$
RMS	$0.2048s$	$0.3048s$	$0.1000s$	$0.4042s$	$0.5042s$	$0.1000s$

Table 3.2.: *Measurement rise and decay times*



(a) Peak and RMS measurement of attack and release **(b)** Detailed view of peak measurement attack phase

Figure 3.4.: *Peak and RMS measurement timing*

3.2.3.1. Level measurement

Both types of level measurement two characteristics: The level itself and the time constant with which this level is reached. To verify the time constants, the peak and RMS measurement of a boxcar function

$$boxcar(t) = u(t_1) - u(t_2) \tag{3.1}$$

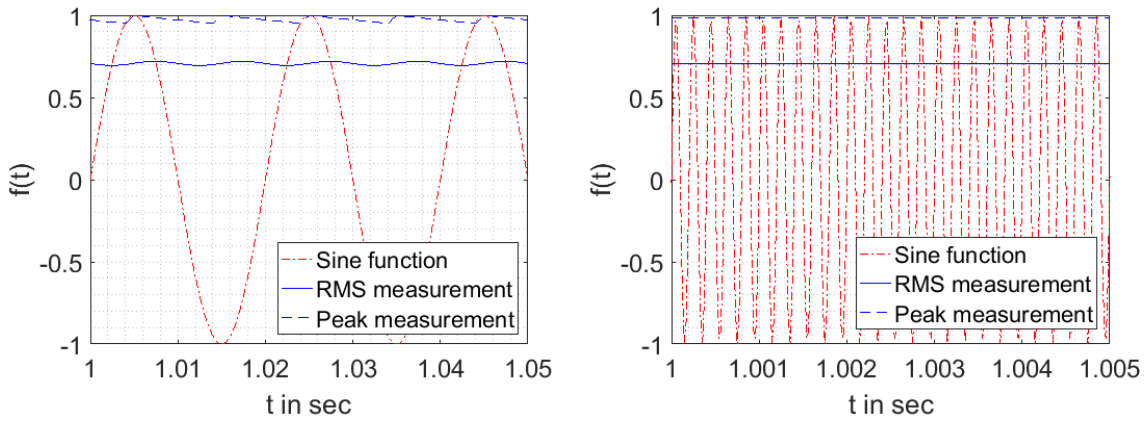
is taken and analyzed, with $u(t)$ being the unit step function, $t_1 = 0.2s$ and $t_2 = 0.4s$. The time constants for this simulation are $t_{average} = 100ms$ for RMS and $t_{attack} = 5ms$, $t_{release} = 200ms$ for peak measurement. The time it takes rising from 10% ($t_{10,R}$)to 90% ($t_{90,R}$) and to decay from 90% ($t_{10,D}$) back to 10% ($t_{10,D}$) is evaluated and should result in these attack, release and average time constants. As can be seen in figure 3.4 and in table, the time difference accurately matches the desired time constants.

To verify correct levels of the measurement functions $peak(x)$ and $RMS(x)$, a sine signal $f(t) = A \cdot sin(2\pi f)$ is evaluated by both functions. At a time $t_0 \gg max(t_{attack}, t_{release}, t_{average})$, the resulting values

$$peak(f(t_0)) \approx A \tag{3.2}$$

$$RMS(f(t_0)) \approx \frac{A}{\sqrt{2}} \approx 0.7071 \cdot A \tag{3.3}$$

are expected and may not be dependent on the frequency f_0. This is verified in figure 3.5.



(a) Peak and RMS level at $f_0 = 50$Hz **(b)** Peak and RMS level at $f_0 = 5000$Hz

Figure 3.5.: *Peak and RMS measurement of a sinusoid signal with amplitude* $A = 1$

3.2.3.2. Limiter

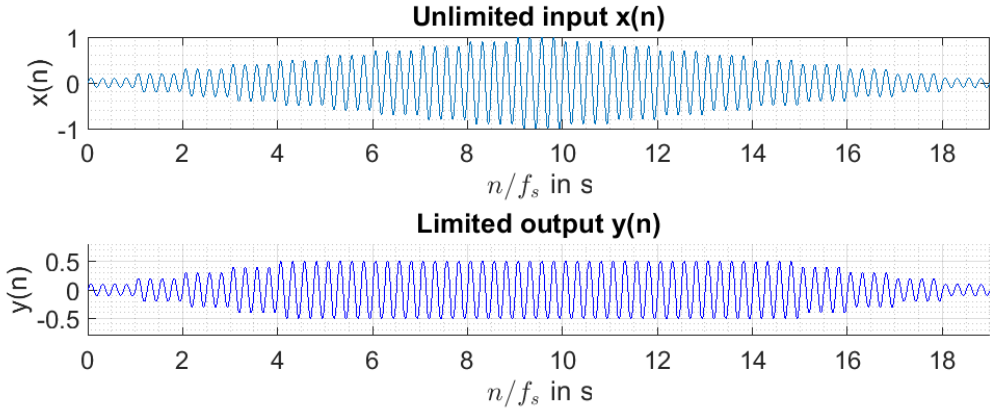


Figure 3.6.:
Limiter simulated operating at $th = -6.02$*dB. The input signal is a sinusoid with* $f = 4Hz$ *and an amplitude change of* 0.1 *each second*

The Limiter is simulated in the Matlab function *limiter.m* and will be set to $t_{attack} = 2$ ms, $t_{release} = 100$ ms, threshold $th = -6.02$ dB and ratio $R = \infty$. This effectively prohibits the output signal to exceed the value 0.5 To evaluate the performance of the limiter, the *test signal 1* will be fed into the module and the amount of overshoot is measured.

As the results in table 3.3 suggest, the limiter error is less than 1%. This margin is negligible with respect to the lack of true peak detection (see section 2.2.1.1).

Input sine frequency	20 Hz	200 Hz	1 kHz	2 kHz	20 kHz
Overshoot	0.703%	0.726%	0.676%	0.393%	0.267%

Table 3.3.: *Limiter overshoot over allowed threshold*

3.2.3.3. Compander

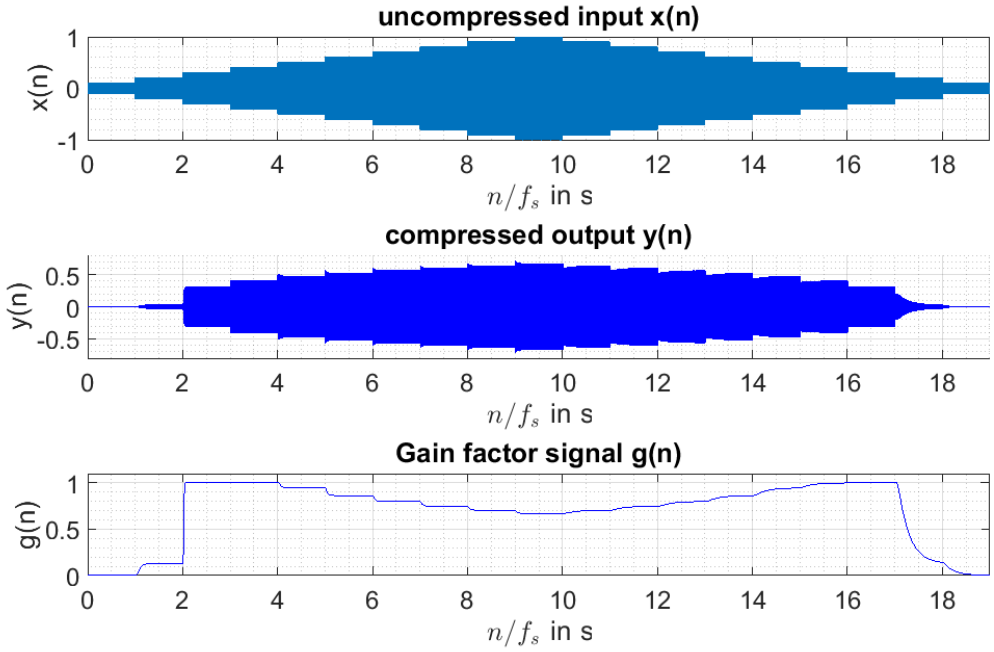


Figure 3.7.:
Compander simulated with parameters as in 3.4. The input signal $x(n)$ is based on test signal 1 at $f = 1000$ Hz.

The initial simulation of the combined expander/compressor reveals an issue when implementing a single smoothing filter for both units: The attack time of a compressor refers to the time it takes to reduce the gain factor g to a value < 1 once the threshold has been exceeded. The attack time of an expander or noisegate on the other hand denotes how fast the gain factor returns back to 1 (no gain reduction). For both compressor and expander, a short attack time compared to the release time is desired. A straightforward implementation would result in swapped release and attack times for the expander, which produces unacceptable results.
Therefore, a range check to determine whether the current gain reduction stems from the compression range or expansion range is included.

Listing 3.2: *compander.m: Range check for attack/release coefficients*

```
persistent range;
    %...
if (rms >= compressor_thresh_ld) %above compressor threshold
    f = -compressor_slope * (rms - compressor_thresh_ld);
    range = 1;
elseif (rms <= expander_thresh_ld) %below expander threshold
    f = expander_slope * (expander_thresh_ld - rms);
    range = 0;
else    % within linear range
    f = 0;
end
    %...
if (range == 1) % last gain reduction stems from compression
    if (g_temp > g_old)
        coeff = rt;
    else
        coeff = at;
    end
else    % last gain reduction stems from compression, swap at and rt
    if (g_temp > g_old)
        coeff = at;
    else
        coeff = rt;
    end
end
    %...
```

The compander simulation is based on the parameters given in table 3.4 and results in the output signal as seen in figure 3.7. The first two seconds, the signal is expanded downwards by the rather steep ratio R_{exp}. At $n/f_s = 2$s, the signal's RMS value exceeds the expander threshold th_{exp}, the short attack t_{at} reverts the gain reduction almost instantaneously. Until $n/f_s = 4$s, the signal stays in the linear range $th_{exp} < RMS(x(n)) < th_{comp}$, no gain reduction takes place. At second 4, the compressor threshold is exceeded and the output is attenuated by a factor that increases with each second. At $n/f_s = 10$, this process successively reverts, albeit slower since the release time is set much higher. As the RMS level falls below expander threshold at second 17, the signal is finally expanded to near-zero again.

This also shows the effectiveness of the range check introduced in listing3.2 as the fast attack both applies when the gain reduction is applied when the RMS value enters compression range as well as the gain reduction is reverted once the RMS leaves the expansion range.

Parameter	Value
compressor threshold	$th_{comp} = -10$ dB
expander threshold	$th_{exp} = -15$ dB
compressor ratio	$R_{comp} = 2$
expander ratio	$R_{exp} = 0.1$
attack	$t_{at} = 10$ ms
release	$t_{rt} = 500$ ms

Table 3.4.: *Parameters for compander simulation*

3.2.4. Distortion

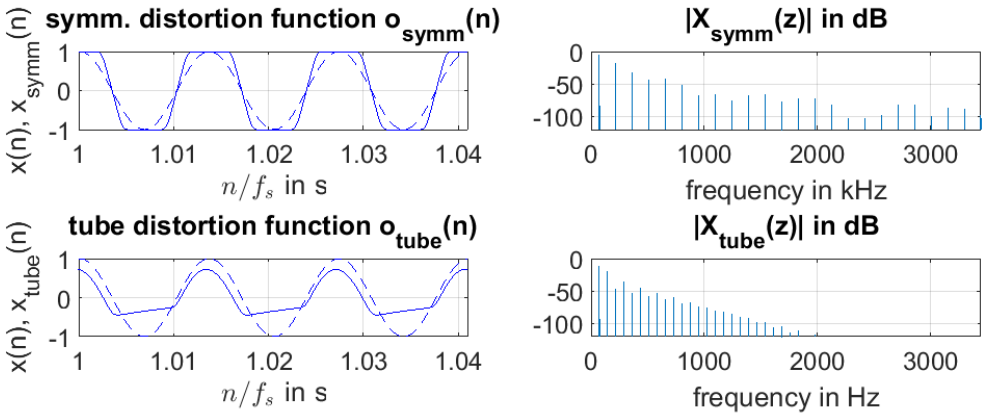


Figure 3.8.:
The symmetrical Schetzen clipper and the tube overdrive effect applied on a sinusoid with $f = 73.3$Hz. Dashed line: input signal $x(n)$. Solid line: overdrive signals $o(n)$

The distortion effect algorithms as lined out in section 2.2.2 are coded in *overdrive.m*, *overdrivetest.m* offers simulation of those effects. The outcome of this nonlinear processing block in the time domain and the frequency domain can be seen in figure 3.8. The simulation parameters are:

- working point $Q = -0.1$,
- drive $dr = 2.0$,
- distortion $dist = 8$.

The input signal $x(n)$ is a sine with frequency $f = 100$ Hz, sampled with $f_s = 48$ kHz. The results are the distorted signals $x_{symm}(n) = o_{symm}(x(n))$ and $x_{tube}(n) = o_{tube}(x(n))$. The symmetric overdrive effect function based on the Schetzen formula $o_{symm}(n)$ solely

adds uneven harmonics to the input, while the asymmetric tube distortion function $o_{tube}(n)$ provides for uneven harmonics as well as even ones.
As can be seen in the spectral domain, even a full range amplitude sinusoid with $f_1 = 100$ Hz $\approx 0.00208 f_s$ produces harmonics above the noise floor of the PCM3003 codec at $k = 35$ or $f_{35} = 3500$ Hz $= 0.073 * f_s$ when distorted by $o_{symm}(n)$ with said parameters. This means that already a sine with $f_1 \approx 700$ Hz introduces aliasing artifacts above the noise floor, which only gets worse with increasing fundamental frequency.
This clearly indicates that employment of multirate signal processing is necessary to combat undesired audible aliasing, as will be covered in the following section.

3.2.5. Multirate signal processing simulation

The previous simulation demonstrated that aliasing images occur when distorting signals to produce harmonics above $f_s/2$. To enable nonlinear signal processing as lined out in section 2.2.2, an interpolation of the input signal is required beforehand to avoid aliasing. The Matlab script *upsampling_filter.m* provides means to specify the FIR constraints for two consecutive stages of interpolation/downsampling. It also utilizes the function introduced in section 3.2.4 to simulate the overdrive effect and analyze the resulting aliasing.

3.2.5.1. Interpolation and half band filter design

To aid with the design of the half band FIRs, the Matlab function *firhalfband(n, fp)* is utilized, where n denotes the desired filter order and fp the passband edge frequency. The specification follows these stipulations in order of importance:

- The computational load is to be kept low, so the number of FIR taps N needs to be low as well.
- The stopband attenuation δ_s should generally be below -75dB across the stopband.
- The audible frequency range should not be affected by the FIR, i.e. the passband edge frequency f_p should be above 16 kHz.

For a single interpolation stage with upsampling factor $L = 2$ from sampling rate $f_{s0} = 48$ kHz to $2f_{s0} = 96$ kHz, these constraints for the interpolation filter $H_1(z_2)$ can be met by setting the passband frequency to $f_p = 16$ kHz and calling *firhalfband()* with varying filter orders until the stopband attenuation criterion is barely met.
However, when interpolating by an upsampling factor of $L = 4$ with **Dyadic Cascading**, the filter $H_2(z_4)$ requires a different approach. It is tempting to allow the same passband edge frequency (thus effectively doubling the transition width)to achieve a lower filter order.

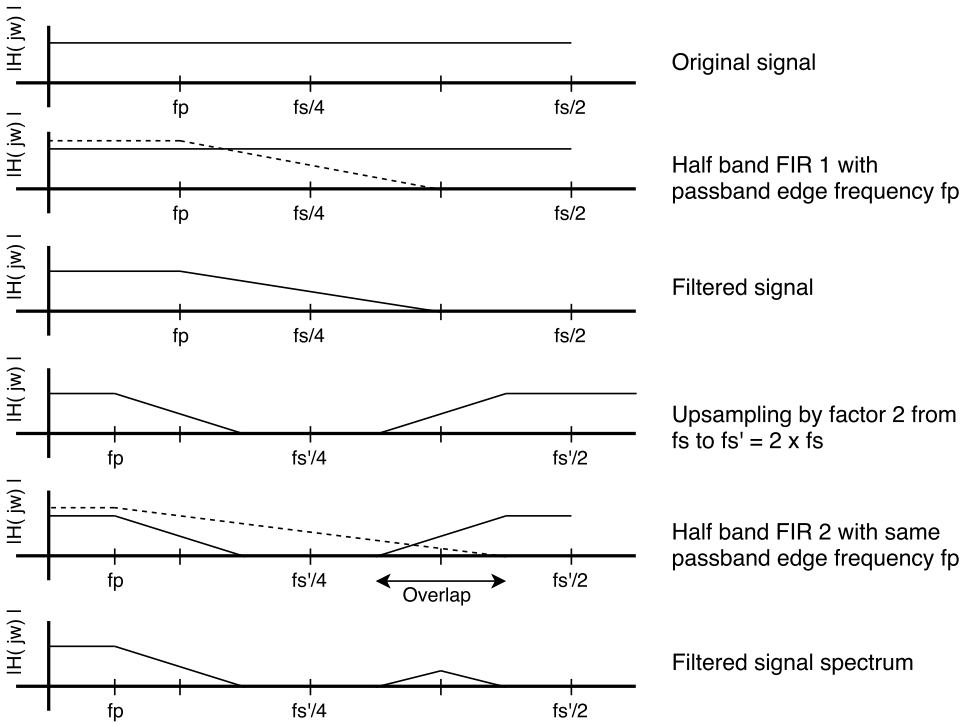


Figure 3.9.:
Transition band overlap occurs when two halfband FIR filters of a dyadic cascade share the same passband edge frequency f_p

This causes side effects, however. The upsampling from $2 \cdot f_s$ to $4 \cdot fs$ mirrors the original spectrum around the new Nyquist frequency $4f_{s0}/2 = 96$ kHz. This means that there is an image of the baseband whose lower passband frequency is $f_p' = 80$ kHz and lower stopband frequency is $f_s' = 64$ kHz, see figure 3.9.
If the second FIR $H_2(z_4)$ with a sufficient filter order to obtain a stopband attenuation $\delta_s = 75$ dB now maintained a passband frequency of $f_p = 16$ kHz, the result would be a stopband frequency of $4f_{s0}/2 - f_p = 80$ kHz - the transition regions of $H_2(z_4)$ and the baseband image would intersect! That would cause a significantly less attenuation than 75 dB in the region from 64 kHz to 80 kHz. The dilemma is pictured in figure 3.10a.
A possible solution would be to raise the passband edge frequency with the same rate as the sampling rate, but this disables the possibility to reduce the filter order of $H_2(z_4)$ compared to $H_1(z_2)$. By doing this, the number of taps needs to stay the same in order to achieve -75dB attenuation around the Nyquist frequency, yet there are side lobes that are attenuated much more than specified, see figure 3.10b. This is inefficient filter design.
To keep the order of $H_2(z_4)$ low while maintaining nearly constant sidelobe maxima levels in the stopband region and fulfilling aforementioned specifications, an experimental approach was chosen: The interpolation filter $H_1(z_2)$ was designed to fit the criteria with $N = 27$ taps

Filter	No of Taps	Passband edge frequency
$H_1(z_2)$	$N_1 = 27$	$f_{p,1} = 16$ kHz
$H_1(z_2)$	$N_2 = 19$	$f_{p,2} = 26.4$ kHz

Table 3.5.: *Halfband FIR filter parameters*

and a passband edge frequency of 16kHz. The parameters for the second FIR are the order n and the passband edge frequency $k \cdot 16$ kHz with $1 < k < 2$. Next, following algorithm was established and applied to find an appropriate $H_2(z_4)$:

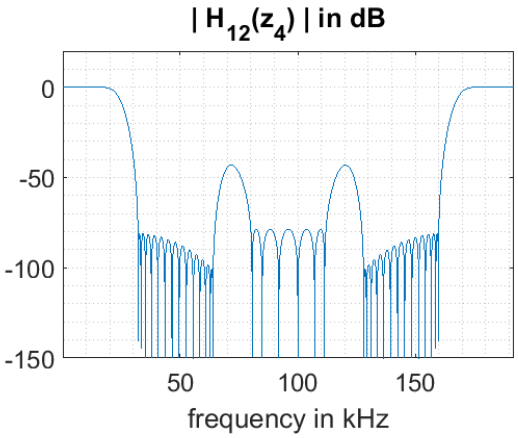
1. start with the sub minimum filter tap number $N = 3$ and passband and $k = 1$.
2. Raise filter number of taps by $N = N + 4$.
3. If $|H_2(96kHz)| > -75$ dB, go back to 2. Else 4.
4. Increase k until $|H_2(96kHz)| \approx -75$ dB.
5. If there are sidelobes in the stopband with $|H_2(z_4)| > -75$ dB, go back to 2. Else 6.
6. Apply further fine tuning of k at leisure.

This manual optimization yields the FIR design parameters in table 3.5. The impulse and amplitude responses of the filters can be seen in figure 3.11, while the response of the dyadic cascade of both filters $H_{12}(z_4)$ is displayed in figure 3.12.

3.2.5.2. Multirate nonlinear processing

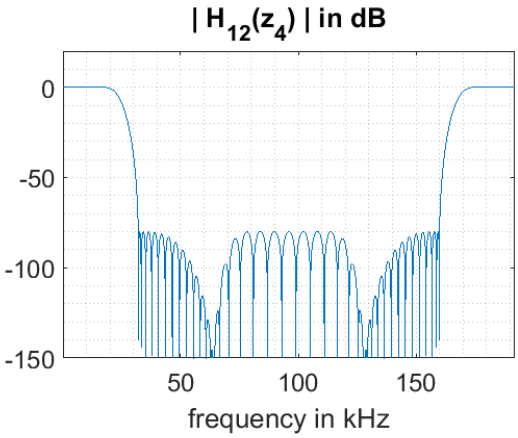
To get a sense of how the multirate processing affects the amount of aliasing artifacts after nonlinear distortion is applied to a signal, the script *upsampling_filter.m* is used again. To achieve this, a discrete sine signal $x_1(n) = sin(2\pi f_1 T_s n)$ with sampling rate $f_s = \frac{1}{T_s} = 48$ kHz, $n = c \cdot T_s$, $c \in \mathbb{Z}$ is upsampled twice to yield signals $x_2(m)$, $m = c \cdot \frac{T_s}{2}$ and $x_4(r)$, $r = c \cdot \frac{T_s}{4}$. The three signal vectors are then fed to the symmetrical overdrive function $o_{symm}(x)$ (see equation 2.37) and subsequently their FFT with length N is computed, resulting in the signals

$$\begin{aligned} d_1(n) = o_{symm}(x_1(n)), \quad & D_1(l) = \mathcal{F}(d_1(n)) \\ d_2(m) = o_{symm}(x_2(m)), \quad & D_2(l) = \mathcal{F}(d_2(m)) \\ d_4(r) = o_{symm}(x_4(r)), \quad & D_4(l) = \mathcal{F}(d_4(n)) \\ & l = 0, \ldots, N-1. \end{aligned}$$



(a)

Cascaded half band filters with constant passband edge frequency $f_{p1} = f_{p2} = 16$kHz, number of taps $N_1 = 26$, $N_2 = 11$



(b)

Cascaded half band filters with same number of taps $N_1 = N_2 = 26$ and passband edge frequencies $f_{p1} = 16$ kHz, $f_{p2} = 32$ kHz

Figure 3.10.:
Half band filter amplitude response of a dyadic cascade $H_{12}(z_4) = (H_1(z_2) \uparrow 2) \cdot H_2(z_4)$

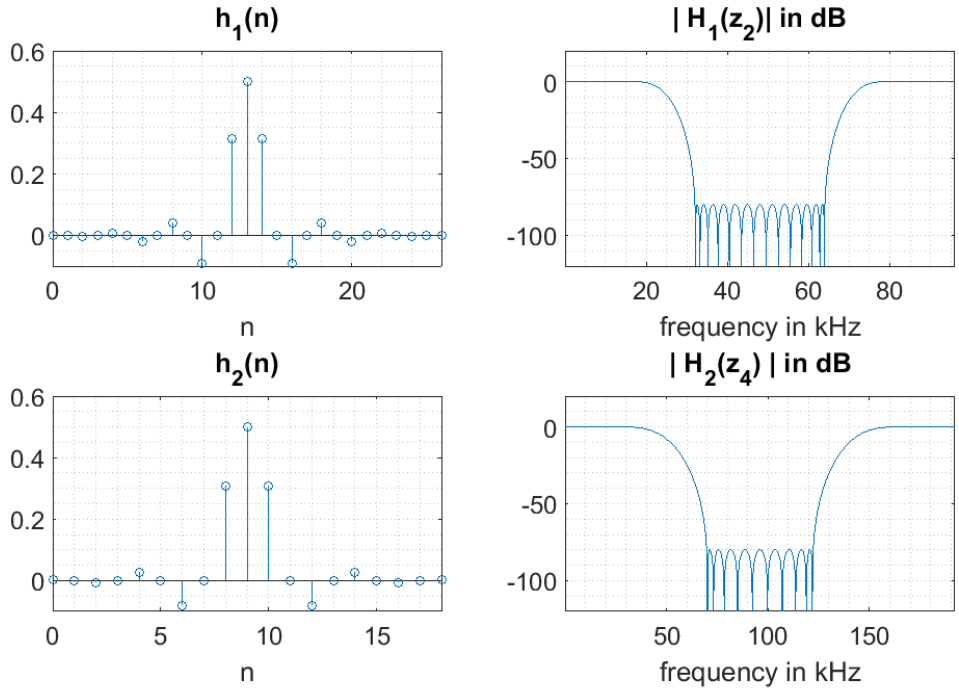


Figure 3.11.: *Half band Filters* $H_1(z_2)$ *and* $H_2(z_4)$

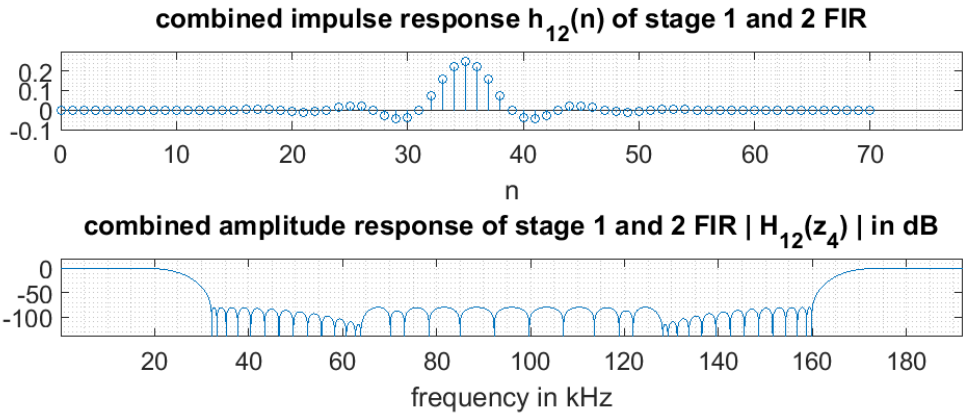


Figure 3.12.: *Dyadic cascade interpolation filter* $H_{12}(z_4) = (H_1(z_2) \uparrow 2) \cdot H_2(z_4)$

The spectra are then subjected to the calculation of the THD_R as in equation 2.36. However, only harmonics are taken into account that are located below the Nyquist frequency of the base sampling period T_s, because harmonics above this frequency are either already mirrored back due to violation of the sampling theorem, or they will be filtered out by the decimation FIR. As the SNR of the system is limited to -96 dB by the 16 bit quantization of the DSP codec, any FFT bins with absolute value below $1.585 \cdot 10^{-5} \cdot |D(f_1)|$ will be ignored for this simulation.
As the symmetric overdrive is a static function, the THD_R of the distorted signal should be constant for any fundamental frequency f_1 as long as

$$f_k < \frac{f_s}{2} \quad \forall \quad k \quad | \quad 20 \cdot log_{10}(|D(f_k)|) > -96dB$$

is fulfilled. In words: as long as there are no harmonics above the Nyquist frequency with level higher the system's SNR.
As the fundamental harmonic f_1 rises, overtones will start folding back into the audible range once they violate the sampling theorem at f_s; the THD_R between 0Hz and 24kHz stays the same. The same harmonics of the signals $d_2(m)$ and $d_4(r)$ do not violate the Shannon theorem of their respective domain yet - they are not taken into account of the THD_R anymore, which therefore increases. Table 3.6 shows the varying THD_R depending on fundamental frequency f_1 and the upsampling factor. An overdrive function with drive $d = 5$ was chosen, which makes for an extreme distortion with relevant harmonics up to $k = 71$. The odd-looking fundamental frequencies are chosen with respect to the FFT length to avoid falsification of results due to spectral leakage.
A fundamental frequency of $f_1 = 16.11$ kHz does not have any harmonics in the audible range. Hence, the THD values in the lowest column exclusively represent aliasing artifacts. The interpolation by factor $L = 2$ reduces aliasing by $\approx 38\%$ in this specific setting, in-

Fundamental frequency f_1	$\mathbf{THD}_\mathrm{R}(\mathbf{D}_1(\mathbf{k}))$	$\mathbf{THD}_\mathrm{R}(\mathbf{D}_2(\mathbf{k}))$	$\mathbf{THD}_\mathrm{R}(\mathbf{D}_4(\mathbf{k}))$
146.5 Hz	39.33 %	39.33 %	39.33 %
1.464 kHz	39.33 %	39.11 %	39.11 %
4.394 kHz	39.33 %	35.77 %	35.53 %
7.324 kHz	39.33 %	32.52 %	31.24 %
10.25 kHz	39.33 %	12.68 %	4.472 %
13.18 kHz	39.33 %	17.48 %	6.981 %
16.11 kHz	39.33 %	24.49 %	9.192 %

Table 3.6.: *Effect of interpolation on nonlinear distortion*

terpolation with $L = 4$ by $\approx 77\%$. This still seems to be a high amount of aliasing. But considering that an overdrive effect is usually applied to a full range signal rather than one band-limited to high frequencies, the alias images are not relevant from a psycho-acoustical point of view. They are masked by the audible lesser-k overtones of the lower fundamental frequencies; the human ear will not notice them.

4. Implementation

This chapter explains the details of the implementation of the project application outlined in the previous section. All source files can be found on the accompanying CD-ROM. A brief instruction on how to set up and use the application can be found in appendix A.

4.1. *MixMaster* - D.Module.C6713 Implementation

In this section, the implementation of the DSP routine will be discussed. The program, which has been christened *MixMaster* is written by the author in ANSI-C (C-99), with following exceptions:

- The files *uartio.h*, *uartio.c* and *bios.h* contain support functions shipped with the D.Module.C6713 kit
- The assembler function `FIR_filter_sc()` in the file *FIR_filter_asm_sc.asm* was received from the digital signal processing laboratory of the Hochschule für Angewandte Wissenschaften Hamburg and was included in this project with courtesy of Prof. Dr.-Ing. Sauvagerd
- The **EDMA** and audio codec handling routines in *main.h* are based on the CCS project *dmod_c6713_dmod_pcm3003_EDMA*, which is provided by the board manufacturer *D.Sign.T*

4.1.1. Overview of the program structure

The general program flow can be seen in figure 4.1. After the initialization of all subsystems in *main.h* (see section 4.1.2), the program enters the main loop `for (;;)`. There, the UART receiver is polled word-wise; if a transmission is received completely, the message is processed. Once a complete ADC sample block is received and stored by the EDMA controller (see section 4.1.3), the interrupt routine `edma_complete_int()` is called and sets the `block` flag. The main loop stops polling the UART and begins with the sample-wise processing of the received audio signals (see section 4.1.4). After the sample blocks are processed and the output is provided for DA conversion, metering data is transmitted

over UART to the control application. Due to the problems explained in section 1.1.2.2, the data will not be handled by the application. Therefore, a detailed explanation of the process is omitted.
The architecture of this program is highly modularized. Individual signal processing blocks are externalized into own source and header files, as is the code for handling UART communication (detailed explanation in section 4.3). All global variables are qualified with the keyword **`static`** and thus cannot be accessed from outside their respective **translation unit** [30, p. 42]. This ensures that no accidental modification or assignment of invalid values takes place. Variables can only be accessed via getter and setter functions. Explicit mentioning of those accessors methods in section 4.1.4 will generally be omitted.

4.1.2. Subsystem initialization

The startup routine of the program consists of two parts - the setup of peripherals and the initialization of variables and buffers.
All relevant board modules, such as the **PLL**, **EMIF**, the **SDRAM** (Not applicable here) and the file table for UART support are set up with the *bios.h* function `init_module()`. The McBSP buses are configured with

Listing 4.1: *main.c: Initialization of McBSP buses*

```
1  /*****************************************************************
2    configure McBSP0 and McBSP1 transmitter and receiver:
3      external clock and framesync
4      two 16 bit data words per frame
5      frame sync active high, one clock delay to data MSB
6      write data on CLK falling edge, read on rising edge
7  *****************************************************************/
8  MCBSP(0)->spcr = MCBSP(1)->spcr = 0x02000000;
9  MCBSP(0)->rcr  = MCBSP(1)->rcr  = 0x00010140;
10 MCBSP(0)->xcr  = MCBSP(1)->xcr  = 0x00010140;
11 MCBSP(0)->pcr  = MCBSP(1)->pcr  = 0x03;
12 MCBSP(0)->spcr = MCBSP(1)->spcr = 0x02010001;
```

The initialization of EDMA is covered separately in section 4.1.3.
Initial values and variable declaration for all signal processing blocks are provided in the files *initial_parameters.h* and *initial_parameters.c*. Each signal processing block offers an initialization function, such as `void limiter_init(void)` or `void overdrive_init(void)` to clear all state variables and buffers and initialize parameters.

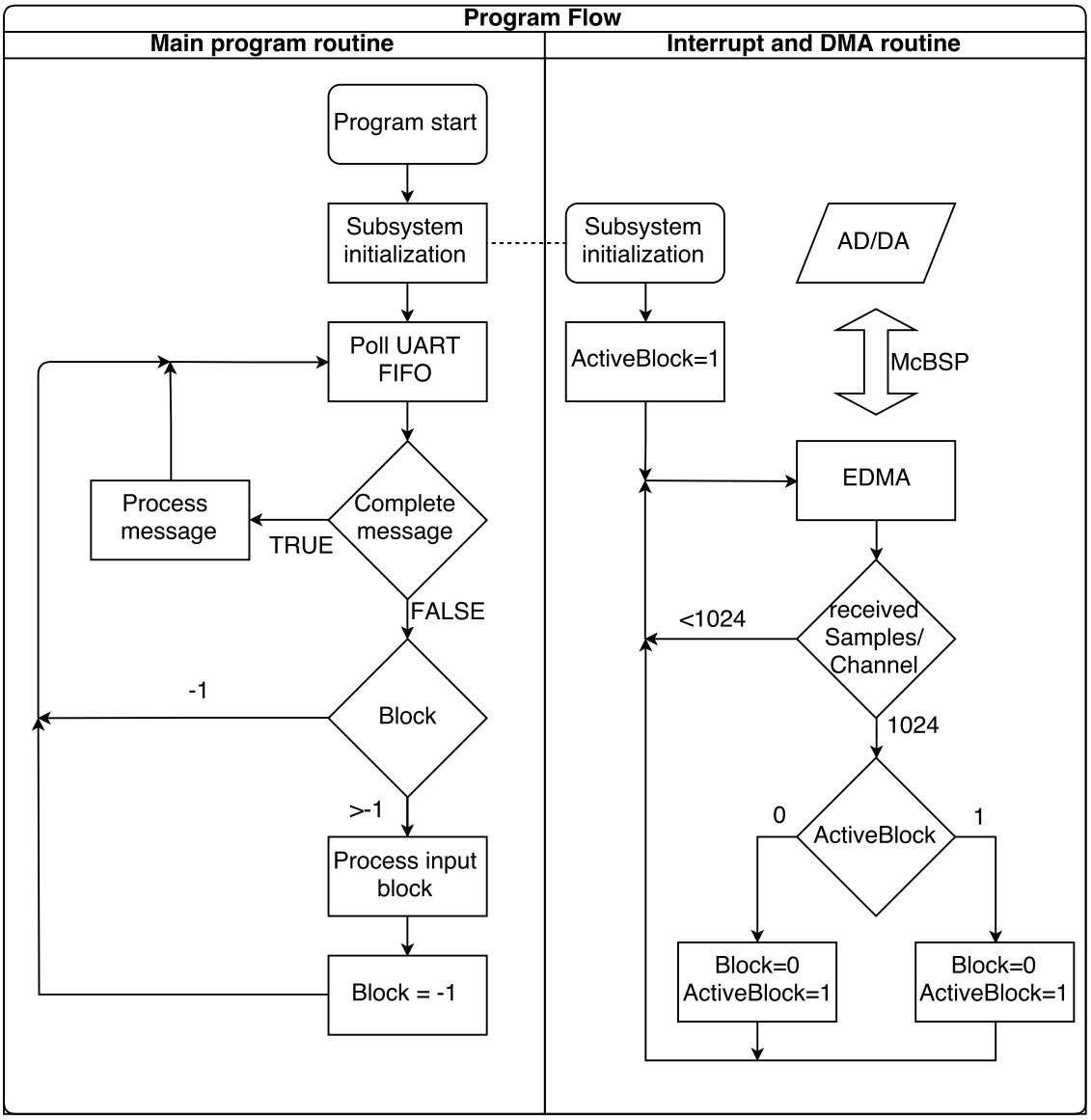


Figure 4.1.: *Overall program flow*

4.1.3. EDMA

The EDMA setup and handling was taken mostly unaltered from the board vendor's demo project *dmod_c6713_dmod_pcm3003_EDMA*. The direct memory access is configured in a way so that the PCM3003 ADC and DAC data transmitted on the McBSP0 and McBSP1 buses are stored in four ping-pong buffers[1] (two input, two output buffers) per channel. This approach greatly reduces processor load when handling converter in/out. Without DMA, the DSP would have to handle an interrupt each time a new sample value at one of the ADC occurs. Each interrupt involves the necessity of a **context switch** and produces a large computational overhead.
The buffers are arranged as two three-dimensional arrays `adcbuffer[channel][block][index]` and `dacbuffer[channel][block][index]`. `channel` denotes the ADC/DAC channel the data comes from or is destined to, `block` is the index of the ping-pong block (either `0` or `1`). `index` points to one of the elements of the block. The blocksize of these buffers is set to 1024 `short` data words, resulting in a requirement of $1024 \cdot 8 \cdot 2 \cdot 2 \cdot 2 = 65534$ byte for input/output buffers.
The channels are transmitted in the order 0, 2, 1, 3, 4, 6, 5, 7. This needs to be corrected before processing starts:

Listing 4.2: *dmod_c6713_dmod_pcm3003_EDMA: Correction of channel order*

```
1  int channel_correction[8] = {0, 2, 1, 3, 4, 6, 5, 7};
2  data = buffer[channel_correction[channel]][block][index];
```

4.1.4. Signal processing

The actual signal processing happens within the main loop of the program. Once a block of 1024 input samples per channel has been received, the processing takes place sample-by-sample as seen in figure 4.2. The processing itself is handled outside of *main.c* by individual translation units. These are discussed briefly. For full documentation, refer to the source code on the CD-ROM accompanying this thesis.
Each input sample is received as a `short` 16 bit signed integer value. It is then converted and further processed as a single precision `float` until it is truncated back to `short` when stored in the output buffer.

[1] One buffer is being written to/from the converters by the DMA controller, while the other is available to the main function for signal processing purposes

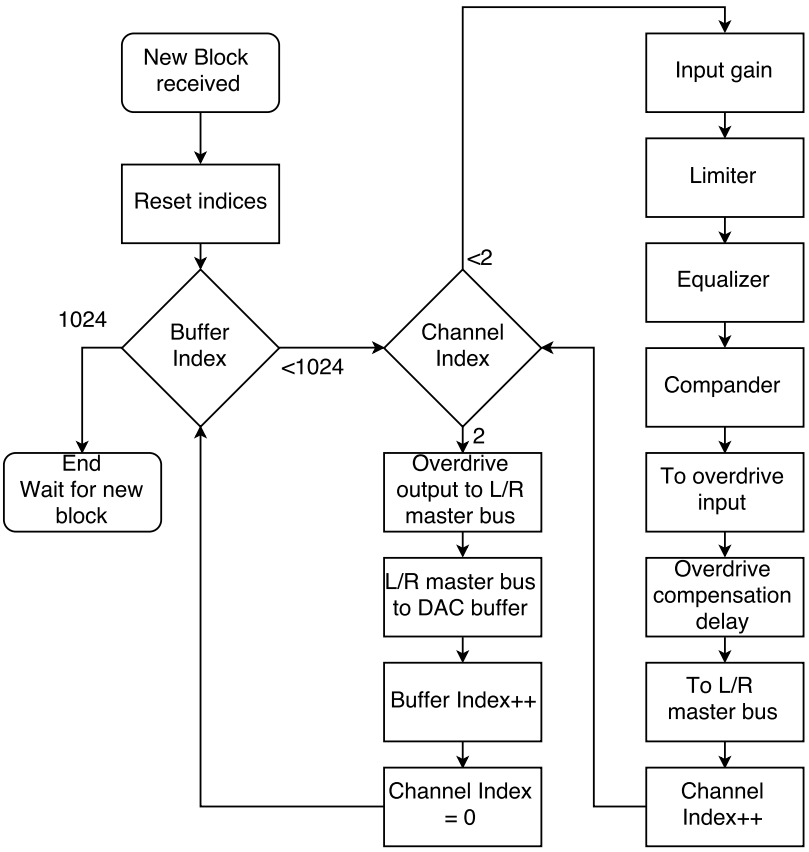


Figure 4.2.: *Program flow of signal processing loop*

4.1.4.1. *initial_parameters.c/h*

These files do not provide any functions. The purpose is to pre-allocate memory for buffers and provide initial values for filter coefficients, gain factors, dynamic range processing thresholds and others. Several global constants are defined in these two files as well, such as the sample rate and the number of input channels. All other source files include *initial_parameters.h*.

This module provides getters that either provide a single `float` or a structure `struct stereo_value` which aggregates two `float`s. The former getters are used to boost or attenuate the channel input (and possibly flip the polarity) and to determine the channel level that is routed to the overdrive module. The latter are used to obtain the values with which a single signal (input channel, overdrive) will be routed to the left and right master buses.

4.1.4.2. *pan_and_level.c/h*

The information about gain factors for all purposes are contained in these files as well as appropriate functions to obtain them. The gain factor of the input channels has a negative value if the *polarity flip* switch in the control surface has been enabled.

4.1.4.3. *peak_meter.c/h* and *rms_meter.c/h*

These modules are responsible for calculation of the peak and RMS values of a channel (see section 2.2.1.1). Since they are depended on the current sample as well as on its history, the calculation must be invoked by specifying the appropriate channel. This is done by calling the functions `float get_peak(int channel_no, float value)` and `float get_rms(int channel_no, float value)`.

4.1.4.4. *limiter.c/h*

This is the implementation of the system thoroughly described in 2.2.1.3. The Limiter is used by calling the function `struct dynamics_mono_return limit_value(int channel, float value, float peak)` with `channel` denoting the channel the sample is from, `value` being the current sample value and `peak` the calculated peak value. The return structure contains the delayed and limited value and the gain reduction factor. The input is internally delayed by an amount of samples defined by the constant `LOOKAHEAD_BUFFER_SIZE`.

4.1.4.5. *sos_filter.c/h*

To equalize the input by a cascade of five biquadratic filters, the function `float filter_cascade(int channel, float value)` of this module is called for each sample. Internally, it is checked whether the high pass(filter with index 0) and/or the two peak equalizers, the high and the low shelving equalizer(indices 1 to 4) are enabled. Then, the input value is filtered with all equalizers of the specified channel that haven't been switched off in the Matlab control software.
Each of the filters is a second order system implemented with Transposed Direct Form II structure [31, pp. 214-215].

4.1.4.6. *compander.c/h*

Similar to limiter.c, this module processes the dynamic range of the input and returns a structure containing the processed value as well as the gain reduction factor. The funcion to call is `struct dynamics_mono_return compand_value(int channel, float value, float rms)` and compresses or expands the argument `value` based on the `rms` value. Compressor and expander can be enabled or disabled individually via Matlab control software. Opposed to the limiter, the output of this function is not delayed.

4.1.4.7. *overdrive.c/h*

The overdrive module splits the processing into two parts: the input buffering and the distortion calculation/output. Since the overdrive effect is potentially driven by more than one channel, the function call `void overdrive_input (float value)` adds the function argument to the input buffer. There will be no further processing until the function `float overdrive_output()` is called: The input buffer is then taken and interpolated by a factor depending on the currently selected overdrive type. The upsampled values are then distorted (and high pass filtered in the case of tube distortion to reject DC components) and decimated back to the original sampling frequency. Then the input buffer is reset to zero and the distorted and decimated sample is returned.

4.1.4.8. *fastmath.c/h*

This module provides approximation algorithms for various mathematic operations. See section 4.4.1.2 for details.

4.1.4.9. *param_exchange.c/h*

Functions to communicate with the control software via UART are provided by this module. It receives new parameters from the Matlab application and calls the other modules' appropriate setter functions to put those new parameters into effect. Details can be found in section 4.3.

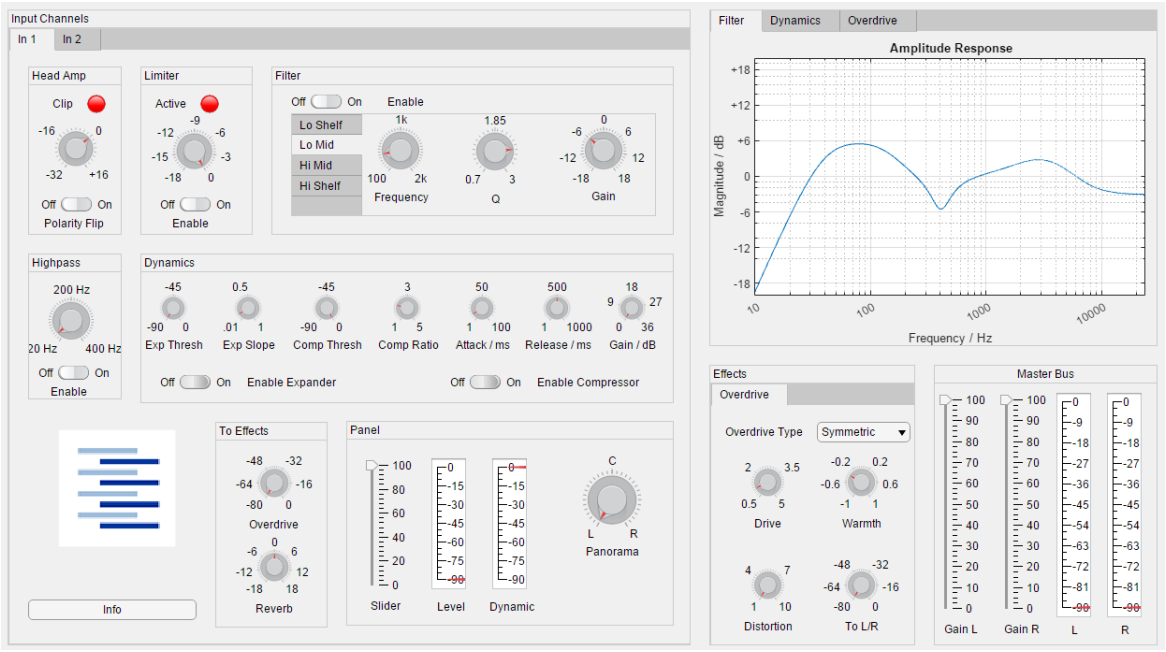


Figure 4.3.: *The GUI of the Matlab control software*

4.2. Matlab App Designer GUI for DSP remote control

The Matlab App Designer, basically an **IDE** within the Matlab IDE offers a quick way to prototype user interfaces for complex Matlab programs as well as the possibility to package them into *Apps* that can be distributed easily. The heart of such an application is a binary file with *.mlapp* extension. This file cannot be read or altered with an external text editor. Instead, the App Designer pre-generates function stubs and provides the ability to edit these functions' body. While it would be possible to program the entire control application with in this file that has been christened *MixMaster Control Utility*, the individual components are externalized into separate classes and functions. The control software user interface can be seen in figure 4.3.

4.2.1. Purpose of the control software

The goal of the *MixMaster Control Utility* is to provide the operator with a clear and self-explanatory interface to control the parameters of the signal processing on the D.Module.C6713. The application can be seen in figure 4.3. A detailed explanation of its functions is given in appendix A.

The leftmost two third of the UI represent the controls for an individual input channel, the signal flow of the channel can be traced roughly from top left to bottom right. There is an

individual layer of controls for each input channel.
In the upper right corner, the operator can select between three plots to visualize current characteristics. The first two display the equalizer's amplitude response and dynamics' static curve of the currently selected channel. The third plot displays the overdrive characteristic. It has to be kept in mind that the plots are calculated within the Matlab software without feedback from the DSP. However, section C show that those simulated blocks can be regarded as identical to the actual processing.
The lower right part of the GUI offers the user to control the parameters of the overdrive effect and the master levels of the DAC outputs.

4.2.2. Software architecture overview

The application's general approach features object-oriented design and its associated traits such as encapsulation of data and inheritance. The connection between classes is outlined in the class diagram seen in figure 4.2.2. Entry point of this program is the aforementioned *MixMasterControlUtility*, which in turn instantiates the classes *filter_manager*, *dynamics_manager*, *gain_manager*, *overdrive_manager*, *serial_manager* and *UIAxes_view_handler*.
The files *parameter_id.m*, *channel_id.m* and *filter_id.m* provide enumerations to distinguish between different types of filters, channels and parameters. This avoids **magic numbers** or invalid values in method calls. *transmission_codes.m* provides a lookup table for identifier characters used in serial communication with the D.Module.C6713.

4.2.3. Object-oriented design paradigm and software patterns in Matlab

As Matlab was not originally created with object orientation in mind, some features are rather haphazardly implemented compared to established programming languages that are designed around the concept of this paradigm, such as C++ or Java. Nevertheless it is possible to realize familiar concepts like interfaces, inheritance and collections of objects with the Matlab syntax.

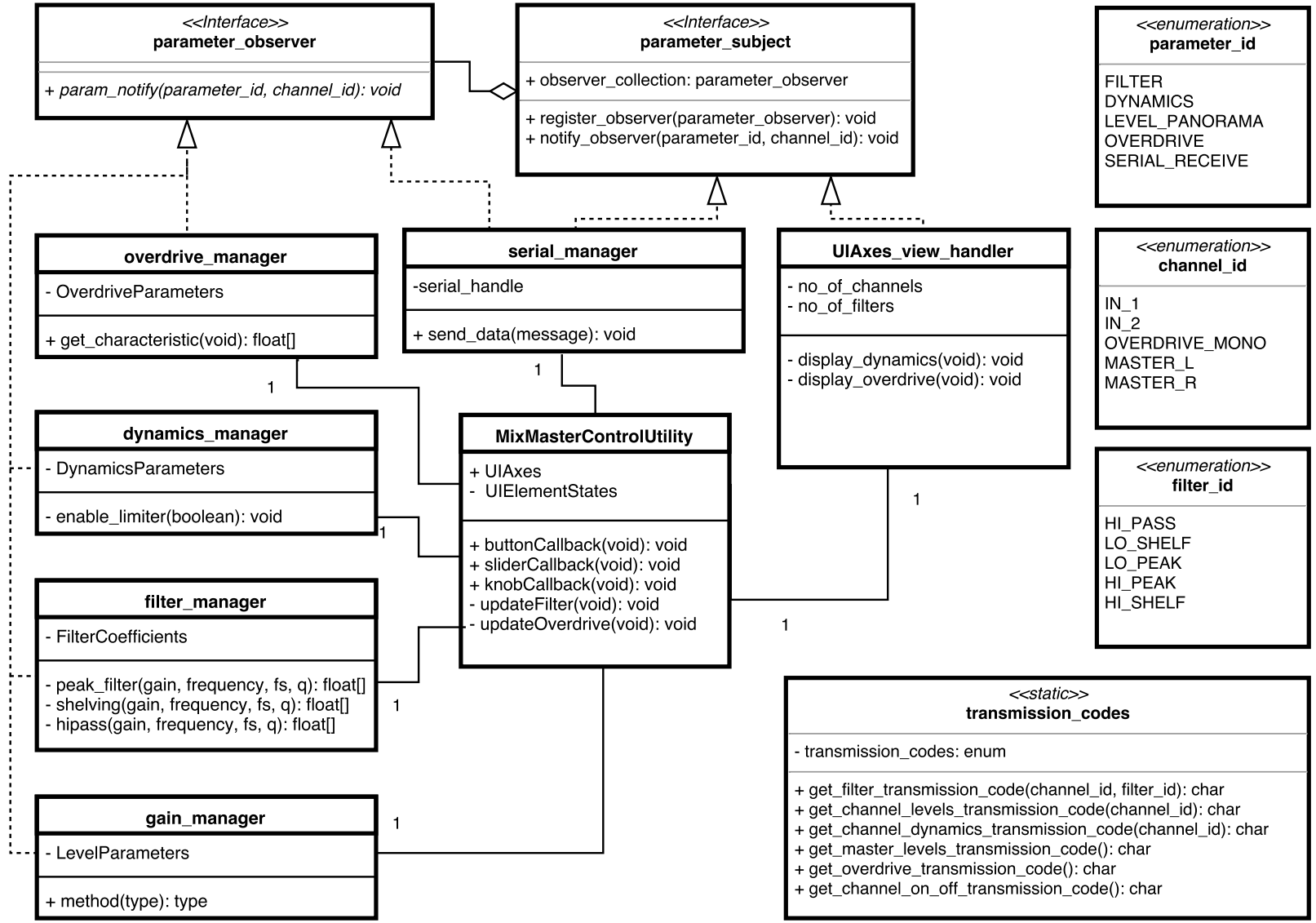


Figure 4.4.: *Simplified class diagram of the Mix Master Control Utility*

4.2.3.1. Object orientation

To obtain functions with consistent states over multiple calls, the keyword `persistent` can be utilized. Objects with persistent states must be defined as classes in separate files and then instantiated. The following listing shows an example class:

Listing 4.3: *demo_class.m: Example of a Matlab class*

```
1  classdef demo_class < handle
2      properties (Access = private)
3          hidden_value;
4      end
5      properties (Access = public)
6          public_value = 5;
7      end
8      methods (Access = public)
9          function obj = demo_class(initial_value)
10             obj.hidden_value = initial_value;
11         end
12         function [new_value, public_value] = do_something(obj, value)
13             obj.hidden_value = obj.hidden_value + value;
14             new_value = obj.hidden_value;
15             public_value = obj.public_value;
16         end
17     end
18 end
```

The file begins with the definition `classdef demo_class`. The `< handle` denotes that this class inherits from the super class `handle`. This is necessary in order to obtain object references. It is possible to inherit from multiple classes.
Similar to C++, class members have to be defined in blocks that qualify their visibility (line 2 and 5 `Access = private` and `Access = public`). The class methods are defined equally.
To instantiate an object of this class, a constructor must be present which needs to have the same name as the class itself. Its return value is a reference to the object itself, in this case called `obj`. This reference is also used when accessing instance variables, such as in line 10 and 13. As the Matlab syntax does not provide a universal way for an object to reference to itself (similar to the Java and C++ keyword `this`), methods that access instance variables must contain an object reference in the argument, see line 12.

Listing 4.4 shows how to instantiate this demonstration class and how to call its methods:

Listing 4.4: *demo_class_test.m: Instantiation and usage of the demonstration class*

```
1  %demo_class_test.m
2  % specify a format to print to command window
3  formatSpec = 'Private member hidden_value: %d; public member public_value
       : %d\n';
4  % instantiate demo_class with value 1
5  demo_instance = demo_class(1);
6  % call do_something
7  [a, b] = demo_instance.do_something(1);
8  fprintf(formatSpec, a, b);
9
10 % change the public member other_value:
11 demo_instance.public_value = 23;
12
13 % other possibility to call do_something:
14 [a, b] = do_something(demo_instance, 1);
15 fprintf(formatSpec, a, b);
16
17 % try to change private member directly, causes error!
18 demo_instance.private_value = 42;
19 fprintf(formatSpec, a, b);
```

Lines 7 and 14 show the two different ways to call a method on an object. The first makes use of the dot (`.`) operator familiar from other object-oriented languages. The second method treats the method as if it was a static function and passes the object reference as the first argument. Both calls are equivalent. Running this script produces the output

```
» demo_class_test
Private member hidden_value:2 public member public_value:5
Private member hidden_value:3 public member public_value:23
No public property private_value exists for class demo_class.


 Error in demo_class_test (line 18)
demo_instance.private_value = 42;
```

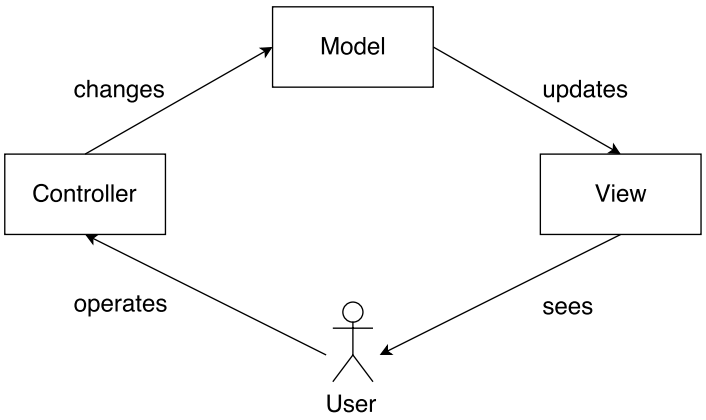


Figure 4.5.: *The MVC software pattern*

4.2.3.2. The MVC pattern

The Model-View-Controller (MVC) is a software pattern for the implementation of graphical user interfaces [32, pp. 186-191]. It divides software components into the three name-giving units

Model the underlying data and/or behavior model of the software

View the graphical representation of the data

Controller the user interface elements that enable the operator to alter the data or behavior

This distinction is useful to decouple the data model separate from the GUI elements and enables to develop an interface prototype without previous knowledge of the underlying mechanics such as data transfer to and from the DSP, component interaction and calculation of derived parameters.
With this pattern in mind, the control software implements the three components as follows:

Model The model consists of the underlying *manager* classes. These classes hold the current states of the signal processing blocks and calculate derived parameters. As an example, the instance of the *filter_manager* class maintains member variables that hold the states of all filters. For each channel, it keeps an array of five biquad filter coefficients. Those get individually recalculated whenever the method `set_parameters(obj, channel, filter_no, gain, q, frequency)` gets called and a new filter parametrization is passed.

View The *MixMasterControlUtility* shows figures (precise: Matlab builtin *UIFigure* objects) that display the state of the current channel's filter cascade, its dynamic range processor characteristic and the characteristic of the overdrive effect. These figures are controlled by the class *UIAxes_view_handler* that updates the *UIFigure* objects whenever the class receives a broadcast message (see section 4.2.3.3) from the Model classes.

Controller All changeable elements of the *MixMasterControlUtility* GUI belong to the Controller component of the MVC pattern. Those are all sliders, knobs and buttons, but also tab groups to switch between input channel controls, equalizer bands and the *UIFigure* layers. Each control element is associated with a specific callback function that is invoked whenever the user changes the element's state. Multiple elements share a single callback function. For instance, all equalizer gain knobs share the callback function `KnobFilterGainValueChanged(app, event)`. If a filter gain knob is turned by the user, the callback first checks which channel is currently active (i.e. which channel tab is in the foreground) and which filter is active (which tab is selected: *lo shelf, lo mid, hi mid, hi shelf*). Then it calls the *filter_manager* method `set_parameters()` with the appropriate channel and filter number as argument as well as the new gain value.

4.2.3.3. The observer pattern

A glance at the class diagram in figure 4.2.2 depicts an implementation of the observer pattern[32, pp. 192-213]. This design enables the communication between classes by sending and receiving specific broadcast messages. A *MixMasterControlUtility* class that provides data which is of interest for multiple classes can inherit from the **abstract** class *parameter_subject*.

Listing 4.5: *parameter_subject.m: The parameter subject super class*

```
1  classdef (Abstract = true) parameter_subject < handle
2      properties
3          observer_collection
4          collection_length = 0;
5      end
6      methods
7          function obj = register_observer(obj, observer)
8            obj.collection_length = obj.collection_length + 1;
9            if (obj.collection_length == 1)
10               obj.observer_collection = observer;
11           else
12               obj.observer_collection = [obj.observer_collection,
                     observer];
```

```
13            end
14        end
15        function obj = notify_observer(obj, param_id, ch_id)
16            for i = 1:obj.collection_length
17                obj.observer_collection(i).param_notify(param_id, ch_id);
18            end
19        end
20    end
21    methods (Abstract = true)
22    end
23 end
```

This class holds an object collection, the `observer_collection`. Two methods are provided by the super class, `register_observer(parameter_observer)` and `notify_observer(parameter_id, channel_id)`. The counterpart to this class is the *parameter_observer* interface[2]:

Listing 4.6: *parameter_observer.m: The parameter observer interface*

```
1 classdef (Abstract = true) parameter_observer < matlab.mixin.
      Heterogeneous & handle
2     methods (Abstract = true)
3         y = param_notify(param_id, ch_id)
4     end
5 end
```

This interface does only contain an abstract method `param_notify()` that is implemented by the child class.
An object that implements the *parameter_observer* interface can register itself at any *parameter_subject* by calling `register_observer()` passing itself as argument. This way, the subject can inform about changes of its internal state by calling `param_notify()` on all registered observers without having to know anything about the specifics of the observer objects.
An example in this specific case is the class *filter_manager*, which inherits from *parameter_subject* and the classes *UIAxes_view_handler* and *serial_manager*, both registered observers of *filter_manager*. If any filter coefficients change due to user interaction, the subject calls the method `param_notify()` with `param_id` being `FILTER` to point to the broadcast origniator and `ch_id` specifying the channel of which the parameters changed. The *UIAxes_view_handler* class now updates the *UIFigure* depicting the filter amplitude response while the *serial_manager* starts transmission of the new filter coefficients to the DSP.

[2] Matlab does not support a dedicated *interface* type in the narrower sense of object orientation, however the *parameter_observer* class fulfills the common definition of an interface[33, p. 42]

Listings 4.5 and 4.6 also demonstrate a way of holding object references in a collection: The class to be aggregated must inherit from the Matlab built-in classes *matlab.mixin.Heterogeneous* and *handle*. Now references to different objects of the same type can be stored in a variable that can be accessed and modified like a common Matlab array.

4.3. RS232

The RS232 interface of the D.Module.C6713 supports baud rates of up to 460 kbaud/s. A high baud rate makes for faster adaption of new parameters and improves the user experience as changes made are perceived as being adapted immediately. On the other hand, this might cause the UART FIFO to overflow since the incoming data words are not read with adequate frequency. A lower rate mitigates this hazard but might cause a noticeable time delay between the change of user parameters in the Matlab user interface and the adaption of the change in the signal processing.
To figure out an appropriate baud rate, following considerations are made: All signals are sampled with 48000 Hz. Thus, the EDMA provides the main loop with a buffer of 1024 new samples per channel every $21.3ms$ to process. The processing of samples has priority over the serial communication and should not be interrupted. Therefore the baudrate must be low enough to ensure that the FIFO does not overflow during those $21.3ms$. It can hold up to 32 bytes of data and each received byte is framed with a stop bit, so a maximum of $32 \cdot 9 = 288$ baud are allowed during this time frame. The result is a maximum rate of ≈ 13.5 kbaud/s. The next lower standard rate is 9.6 kbaud/s, which also happens to be the value the D.Module.C6713 is configured to by default.

4.3.1. Serial communication from Matlab to DSP

The core of communication with the DSP in Matlab is the serial port object created with *serial* [34]. In the control software, the object is constructed and maintained by the class *serial_manager.m* with the method `serial_init()` on application startup:

Listing 4.7: *Initialization of the serial object*

```
1 function obj = serial_init(obj)
2     fprintf('Creating serial object ...\n');
3 obj.serial_handle = serial('COM6', 'BaudRate', 9600, 'Terminator', 'CR',
      'DataBits', 8);
4     obj.serial_handle.BytesAvailableFcn = @obj.receiving_data;
5     fprintf('Opening serial object ...\n');
6     fopen(obj.serial_handle);
7 end
```

serial_manager implements the *parameter_observer* interface. Its instance is subscribed to *filter_manager*, *dynamics_manager*, *gain_manager* and *overdrive_manager*, thus it receives broadcast messages whenever a change of parameters in the user interface is made.
Upon broadcast reception, *serial_manager.m* calls the sender's appropriate getter function to obtain the new parameters in a D.Module.C6713-specific format (i.e.

`get_processed_params(obj, channel)` for the dynamic range control parameters).
The majority of data to be transmitted will be arrays of floating point values that represent numerical data, the number of elements depend on the specific module that had its properties changed. Each value is converted into an ASCII string that represents the value as IEEE-conform single precision hexadecimal. For example, the number 1.234 is converted to the character array '3f9df3b6'. To identify the purpose of such an array, an additional identifier is needed. This is a single character which is obtained from the lookup class *transmission_codes.m*. This identifier leads the message, followed by the converted hexadecimal strings. A list of identifiers and values can be seen in table 4.1.
After a message of identifier, hex-encoded values and terminator character `'\r'` has been assembled, it is sent over the COM-port to the DSP by calling the Matlab built-in function `fprintf()`:

Listing 4.8: *Writing data via COM port*

```
58    fprintf(obj.serial_handle, message);
```

`obj.serial_handle` references the member variable that points to the *serial* object, `message` contains the string to be sent.
An example: The operator changes a parameter of the overdrive. The lookup table returns the transmission code 'R' for overdrive parameters. The actual values are: Overdrive type is 2 (the tube overdrive), drive factor $dr = 0.5$, working point $q = -0.4$, distortion $dist = 10$. These values represented as IEEE single precision hexadecimals are $0x4000\,0000$ (type), $0x3f00\,0000$ (drive factor), $0xbecc\,cccd$ (working point) and $0x4120\,0000$ (distortion). Thus, the message sent to the DSP would be the string 'R40000003f000000beccccd41200000\r'. On the DSP side, the module *param_exchange.c/h* is responsible for receiving and dispatching these messages. While the main loop is **busy waiting** for the EDMA to provide a new block of input samples, it repeatedly calls the function `void check_for_messages(void)`. Each call polls the board's UART FIFO and, if data has been received, reads one character. The character is stored in a message buffer. If the received character is the agreed upon terminator (carriage return, `0x0D`), the transmission is complete. An internal dispatcher function is called that looks up the appropriate message destination according to the identifier character.
The received string is then segmented into substrings of eight characters beginning with the first after the identifier - the floating point values converted to ASCII strings. Those `char` arrays are converted back to `float` data types. A check is then performed whether the received floating point values are valid numbers (neither `+Inf`, `-Inf` or `NaN`). If one or more floating point numbers are invalid, the message is ignored. If each number passed this check, the transmission is deemed successful and the setter function that is associated with the identifier character is called with the received values as arguments. In both cases, the message buffer is cleared afterwards and the FIFO is emptied.

4.3.2. Serial communication from DSP to Matlab

Since the data sent from the DSP will not be used by the control software in a meaningful way, this section will be kept brief.
An array of values relevant to the metering will be stored in the memory, such as highest peak value of a channel or master bus or flags whether a limiter became active. After processing a block of samples, one of those values or an aggregation of flags will be transmitted. The functionality is provided by *param_exchange.c*, which takes care of UART communication.
Entry point to the transmission are the functions `send_int(char param_id, int data)` and `send_float(char param_id, float data)`. Both convert the argument `data` to a string consisting of the according ascii number characters. `param_id` is an identifier for the control software. The resulting string is then sent by calling the *bios.h* function `int send_char (char c)` character by character.
The class *serial_manager* of the control software takes care of the reception of data. Upon creation, the serial object is assigned a callback function which is called whenever data has been received via UART (`BytesAvailableFcn`):

Listing 4.9: *Assignment of a callback function to the serial object*

```
26  obj.serial_handle.BytesAvailableFcn = @obj.receiving_data;
```

The referenced function `receiving_data` assembles the message by buffering it until the agreed upon terminator is transmitted (`'\r'`), then notifies *UIAxes_view_handler*, a subscriber of the *serial_manager*, via broadcast about new metering data.

ID	Type	Parameters in order
'A'	Channel 1 high pass filter	a_1, a_2, b_0, b_1, b_2
'B'	Channel 1 low shelving filter	a_1, a_2, b_0, b_1, b_2
'C'	Channel 1 low peak filter	a_1, a_2, b_0, b_1, b_2
'D'	Channel 1 high peak filter	a_1, a_2, b_0, b_1, b_2
'E'	Channel 1 high shelving filter	a_1, a_2, b_0, b_1, b_2
'F'	Channel 2 high pass filter	a_1, a_2, b_0, b_1, b_2
'G'	Channel 2 low shelving filter	a_1, a_2, b_0, b_1, b_2
'H'	Channel 2 low peak filter	a_1, a_2, b_0, b_1, b_2
'I'	Channel 2 high peak filter	a_1, a_2, b_0, b_1, b_2
'J'	Channel 2 high shelving filter	a_1, a_2, b_0, b_1, b_2
'K'	Channel 1 levels	input gain , to overdrive , to master bus L, to master bus R
'L'	Channel 2 levels	as above for ch 2
'M'	Overdrive and master levels	overdrive to master bus L gain factor, overdrive to master bus R gain factor, master bus L to DAC 0, master bus R to DAC 1
'N'	Channel 1 on/off switch states	ch 1 polarity flip, ch 1 high pass enabled ch 1 EQ enabled ch 1 limiter enabled ch 1 expander enabled
'O'	Channel 2 on/off switch states	as above for ch 2
'P'	Channel 2 Dynamics parameters	ch 1 limiter threshold ch 1 compressor threshold ch 1 expander threshold ch 1 compander attack ch 1 compander release ch 1 compressor slope ch 1 expander slope ch 1 makeup gain
'Q'	Channel Dynamics parameters	as above for ch 2
'R'	Overdrive parameters	overdrive type drive factor dr tube working point q tube distortion $dist$

Table 4.1.: *Transmission codes and trailing values*

4.4. Performance analysis and measurements

The implementation is accompanied by an ongoing judgement of the quality and correctness of the signal processing with electronic measurement devices as well as biological ones, namely the ears. Problems during the implementation can be discovered quickly by a trained ear. However, certain problems are discovered quickly even by the layman when listening carefully: for example, performance problems due to inefficient algorithms cause audible **dropout**s.
This section covers the detection and solving of problems that occurred during the project and measurements wrapping up and validating the implementation.

4.4.1. Performance considerations

During the process of the implementation, several adjustments had to be taken to improve the result. They can be divided in two general groups: those made with intention to improve the performance in terms of quality, and those to improve performance in terms of speed.

4.4.1.1. Quality performance adjustments

These adjustments were undertaken in order to yield a better quality of the signal processing by increasing SNR, reducing distortion or by making the real result match the theoretical more closely.

Alternative filter structure for biquad filter section The transposed direct form II (TDF2), which is the underlying structure of this project's biquadratic filter implementation, derives its difference equation directly from the coefficients of the corresponding transfer function. This results in the fact that a real system with finite precision has a low density of possible poles along the real axis and around z = +/- 1 compared to the high density around z = +- j. This potentially affects the high pass and low shelf filters in this project, whose cutoff/center frequencies can be trimmed very low compared to the sampling frequency[11].
This can be mitigated through the use of a different filter structure, such as the Gold&Rader structure[35]. Opposed to the TDF2, it alters the denominator feedback path of the filter as seen in figure 4.6, so that the multipliers are not equivalent to the transfer function coefficients a_0, a_1 and a_2 but from the real and imaginary parts of the poles $r \cdot cos(\phi)$ and $r \cdot sin(\phi)$. This makes for an even distribution of poles across the whole area within the unit circle.
Measurement shows that an implementation does not yield better results in terms of cutoff/center frequency accuracy nor SNR. The latter makes sense since the biquadratic filters operate on single precision floating point arithmetic and have an ideal SNR of 138 dB due

to the 23-bit mantissa, while the audio codec quantizes with 16 bit and offers an SNR of 90 dB [36]. Both structures make for a quantization noise that is well below the noise floor introduced by the converters and does not influence the audio quality.
As this structure does not yield benefits but increases the number of multiplications per output sample and introduce a delay into the signal path, the Gold&Rader filter is not employed.

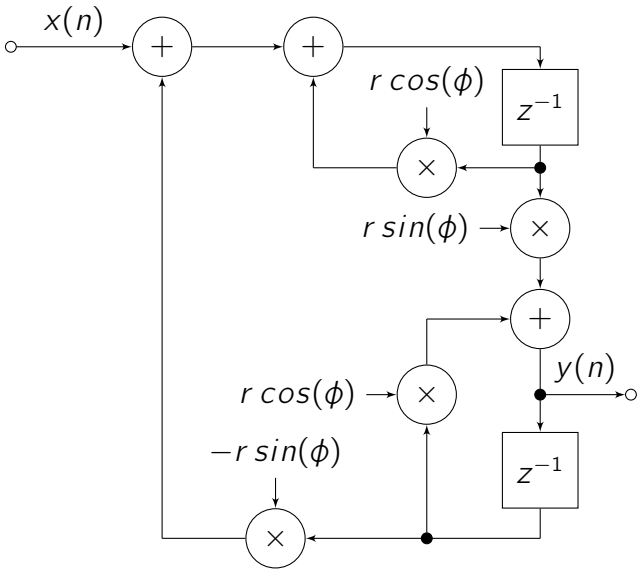


Figure 4.6.: *Gold & Rader filter structure of recursive IIR part*

4.4.1.2. Speed performance considerations

As the processing blocks grow in number during implementation, it is inevitable that the computation time needed for certain blocks increases the maximum time available given by the real time constraints of processing a continuous audio stream. This is the case when the EDMA finishes receiving a full block of audio samples from the codec before the main loop finished processing the current block. This causes audible **dropouts**. In this case, an identification of the bottleneck in the processing chain is required first, then the problematic block must be optimized if possible.
A helpful tool for identifying code parts that require a high amount of cpu time is the CCS Simulator. It allows the execution of code on a virtual C6000 Hardware and provides detailed analysis about the load that individual functions cause. The simulator for code profiling is set up according to the guidelines of the digital signal processing laboratory of the Hochschule für Angewandte Wissenschaften Hamburg [37]. Further modifications must be made to the project to simulate it successfully; a run with the project as-is results in the error message

Console | Expressions | Profile Setup | Line Coverage | Function Coverage | Profile

	Name	Calls	Excl Count Min	Excl Count Max	Excl Count Average	Excl Count Total
1	set_averaging(float)	1	12	12	12.00	12
2	get_channel_gain(int)	27800	16	16	16.00	444800
3	overdrive_init()	1	18	18	18.00	18
4	get_input_to_overdrive(int)	27800	21	21	21.00	583800
5	overdrive_input(int, float)	27800	22	22	22.00	611600
6	decimate_by_four(short *, short *)	13900	32	32	32.00	444783
7	interpolate_by_four(short, short *)	13900	35	35	35.00	486500
8	get_rms(int, float)	27800	44	44	44.00	1223200
9	get_peak(int, float)	27800	52	52	52.00	1445600
10	filter_cascade(int, float)	27800	56	56	56.00	1556800
11	interpolate_by_two(short, short *, short, short)	41700	80	80	80.00	3336000
12	get_pan_and_level(enum unknown, int, float)	55598	80	92	84.50	4698022
13	rms_init()	1	90	90	90.00	90
14	compander_init()	1	92	92	92.00	92

Figure 4.7.: *Profiling of functions of a CCS project*

```
TMS320C671X: Error:  Illegal opcode (403a0000) at
pc = 0xb00085ec
```

This issue occurs due to the memory limitations of the profiler [38] and can be resolved by setting the constant `BLOCKSIZE = 1024;` to a smaller value such as `64`. This limits the size of the buffer per channel the EDMA reads and writes. Since peripherals such as the audio codec are not simulated, this does not have any impact on the simulation. Figure 4.7 shows an example of profiling with CCS.
With this tool at disposal, several critical functions have been identified and optimized to be able to process the incoming audio without risking dropouts.

Variable upsampling factor for overdrive effect The distortion effects calculation and the accompanying interpolation have the highest demand on CPU performance by a big margin. As can be seen in equation 2.38, the tube overdrive routine requires the calculation of two exponential functions and two divisions per call, and due to the upsampling by factor $L = 4$ the routine is called four times per input sample.
Figure 3.8 shows that the envelope of harmonics produced by the tube overdrive decays significantly faster with increasing harmonic number k than the symmetric distortion. Therefore, a reduction of the upsampling factor to $L = 2$ when the tube distortion is active while maintaining $L = 4$ for the symmetric effect is a viable option. This way, performance is increased without compromising the suppression of aliasing artifacts unduly.

Fixed-point implementation of Overdrive effect To further optimize the symmetric overdrive effect, a fixed point implementation is undertaken instead of the initial floating point calculation:

Listing 4.10: *Fixed point implementation of symmetric overdrive*

```
129 short schetzen_distortion(short value) {
130     temp_i = value * overdrive_saturation;
131     if (temp_i > SCHETZEN_THRESHOLD_2) {
132         temp_i = FULL_SCALE;
133     } else if (temp_i < -SCHETZEN_THRESHOLD_2) {
134         temp_i = -FULL_SCALE;
135     } else if (abs(temp_i) < SCHETZEN_THRESHOLD_1) {
136         temp_i *= 2;
137     } else {
138         if (temp_i > 0) {
139             temp_i = (triple_fs - fast_sq_short((short) (double_fs_sqrt -
                      ((triple_fs_sqrt * temp_i) / FULL_SCALE))))/3;
140         } else {
141             temp_i = -(triple_fs - fast_sq_short((short) (double_fs_sqrt
                     + ((triple_fs_sqrt * temp_i) / FULL_SCALE))))/3;
142         }
143     }
144     return (short) temp_i >> 1;
145 }
```

This ensures that this effect can be utilized at $4\times$ Upsampling without causing dropouts.

Approximation of mathematical functions The side chain signal paths of the dynamic range processors are based on calculations of power and exponential functions, the square root and the binary logarithm. These operations are very expensive in terms of CPU cycles and although the native *math.h* C-library is highly optimized, it provides more accuracy than is needed: The side chain does not affect the audio signal directly, but via a gain factor that is low pass filtered with a time constant that is long compared to the period of low frequency, so a moderate margin of approximation errors in this path is acceptable.
For this reason, the library *fastmath.h* was developed and included in the project that provides approximations for the aforementioned functions

- `float fast_sq(float value)` simply multiplies the argument with itself returns the result. This is not an approximation. The library *math.h* does not provide a dedicated function to square a value, instead the function `double pow (double base, double exponent)` is supposed to be used. Since the exponent is evaluated at runtime, calling `pow()` is not the optimal choice.

- `float fast_sq_short(short value)` is the equivalent to `float fast_sq(float value)` for computing the square of 16 bit signed integer values.
- `void fast_sqrt(float number, float *result)` calculates the square root of the passed number and stores the result at the address in the argument. It is based on the fast inverse square root algorithm first established in the video game *Quake III Arena* from *id Software*. This algorithm shifts the IEEE-conform floating point value and subtracts a magic number to obtain a good first estimation for the following two iterations of the well-known Newton method [39].
- `void fast_log2(const float *value, float *result)` has the purpose of calculating the binary logarithm of a single precision number. The function achieves this by setting the result's integer part to the exponent of the argument; the fractional part is calculated by evaluating the logarithm's taylor series at the value of the argument's mantissa.

4.4.2. Final measurements

With the DSP source code being undergone several re-factorizations and aforementioned optimizations, concluding measurements to validate the expected outcome of the simulations are required. Plots of the measurements can be found in Appendix C.

4.4.2.1. Measurement setup and equipment

The measurements performed in this project fall in the two categories that require different setups: frequency domain analysis and time domain analysis. Both are shown in appendix B.

Time domain analysis, development and debugging This domain is analyzed by recording the DSP response of test stimuli. Those are wave files created in Matlab. Playback and recording of signals is done via *Windows 7* PC with the **DAW** *Cockos Reaper*. In/Out is handled by a *Focusrite Saffire PRO 24* audio interface. Rehearsal of audio was done via *HEDD Type 07* studio monitors.
The same setup was used for the entire development and debugging stage of the project as well.

Frequency domain analysis For measurements in the frequency domain, such as $THD + n$ and amplitude response, the *Audio Precision System Two Cascade Plus* audio analyzer is utilized. The audio codecs PCM3003 operate at a full scale input/output level of $1.9V_{pp}$ which equals to -1.29 **dB**$_U$. All measurements with the *Audio Precision System Two* will be conducted with a test signal level of -3 dB$_U$ unless noted otherwise.

4.4.2.2. Reference measurement

The first measurements are undertaken in order to evaluate the capabilities of the D.Module.C6713. For this, the CCS project *dmod_c6713_dmod_pcm3003_EDMA* is utilized. When programmed with this example provided by the board vendor, no signal processing will take place apart from routing all input channels of the PCM3003 codecs to the according output channels. The frequency response can be seen in figure C.1. This shows that a low pass filter with noticeable passband ripple is in effect. This is presumably caused by the digital decimation and interpolation filters of the codec input/output stages and lies within the specifications.
Figures C.2 and C.3 show the noise floor and the distortion of the DSP when excited with a sine with $f = 1$ kHz at -3 dB$_U$. The *System 2* reading gives a noise floor of 0.057% and a THD of 0.006%.

4.4.2.3. Parametric equalizer

The critical point of the filter section is the high pass filter with cutoff frequency $f_0 = 20$ Hz, since this is the lowest frequency to which the filter can be tuned and here the negative effects of finite precision are potentially most significant. However, figure C.4 proves that concerns are not necessary and the -3 dB center frequency lies exactly at 20 Hz as was set via Matlab control surface.
To evaluate the performance of the parametric equalizer, it is set according to the specifications in table 3.1. Compared to figure 3.2b, the behavior as seen in plot C.5 is nearly identical. There is a significant difference in the range 50 Hz to 150 Hz: The DSP shows a 'flattened' curve with a maximum at ≈ 4.8 dB opposed to the simulated ≈ 5.2 dB. Reason for this is that the graph was recorded at -6 dB$_u$ by mistake, disregarding the fact that the expected maximum result of $+5.2$ dB exceeds the maximum output of the codec and causes clipping. Nevertheless - with this in mind, interpolating the plot at this frequency region, it can be assumed that the behavior of the DSP is equally as expected when given appropiate level adjustments.

4.4.2.4. Overdrive

The overdrive functions are measured with fundamentals f_1 at several different frequencies. The higher frequencies will be covered in section 4.4.2.5 in the context of multirate processing. Here, figure C.6 displays the resulting symmetric distortion function of an input sine with $f = 250$ Hz and drive factor $dr = 2$. Figure C.12 shows a tube distortion with fundamental frequency $f_1 = 500$ Hz, drive factor $dr = 2$, working point $Q = -0.1$ and distortion $dist = 8$. These parameters are equivalent to the ones in 3.2.4. Since the resulting total harmonic distortion is highly depending on the input signal level, the direct THD values of simulation and measurement won't be compared; a visual comparison has to suffice.
The distorted signals strongly resemble the spectra plotted in 3.8, the envelope of the harmonics behave similar. The tube distortion produces odd and even harmonics with steep roll-off, while the symmetric distortion provides a slower decay of exclusively odd harmonics.

4.4.2.5. Multirate processing

To evaluate the effect of multirate processing on aliasing suppression with a nonlinear system, the two overdrive functions are re-evaluated with higher fundamental frequencies of $f = 10$kHz and $f = 14$ kHz respectively. The results are shown in figures C.6 to C.17.
The parameters are the same as in the previous section. The results are listed in table 4.3 and show that interpolation of a signal before distorting it is in fact a very helpful technique for avoiding aliasing.
If a sine with a level -3 dB_U and a fundamental frequency f that is small compared to the sampling rate f_s is distorted with the symmetric overdrive effect with drive factor $dr = 2$, a spectrum as seen in figure C.6 is the result. The levels of the harmonics relative to the fundamental ($k = 1$) are read and gathered in table 4.2.
Now, considering a sine of $f = 10$ kHz with the same level, the highest overtone can be found at $k = 3$ or 30 kHz with a level relative to the fundamental of $-11dB$, followed by $k = 5$/50 kHz with -17 dB. With a sampling frequency of 48 kHz, both harmonics violate the Nyquist theorem and theoretically fold back into the audible spectrum to $|f_s - 30$ kHz $| = 18$ kHz ($k = 3$) and $|f_s - 50$ kHz $| = 2$ kHz ($k = 5$). And indeed, looking at figure 4.8a reveals that this configuration produces the exact levels of aliasing at the frequencies 18 kHz and 2 kHz.
If the same signal is upsampled to $4 \cdot f_s = 192$ kHz before distortion, other results are expected: As the passband edge frequency of the interpolation FIR filters is set to $f_p = 16$ kHz, aliasing is supposedly suppressed by a big margin. Only harmonics within the band $c \cdot 192$ kHz $\pm f_p$, $c \in \mathbb{N}$ will fold back into the base band unattenuated. For $c = 1$, those are the harmonics $k = 19$ and $k = 21$ which are expected at the baseband frequencies 2 kHz and 18 kHz respectively.

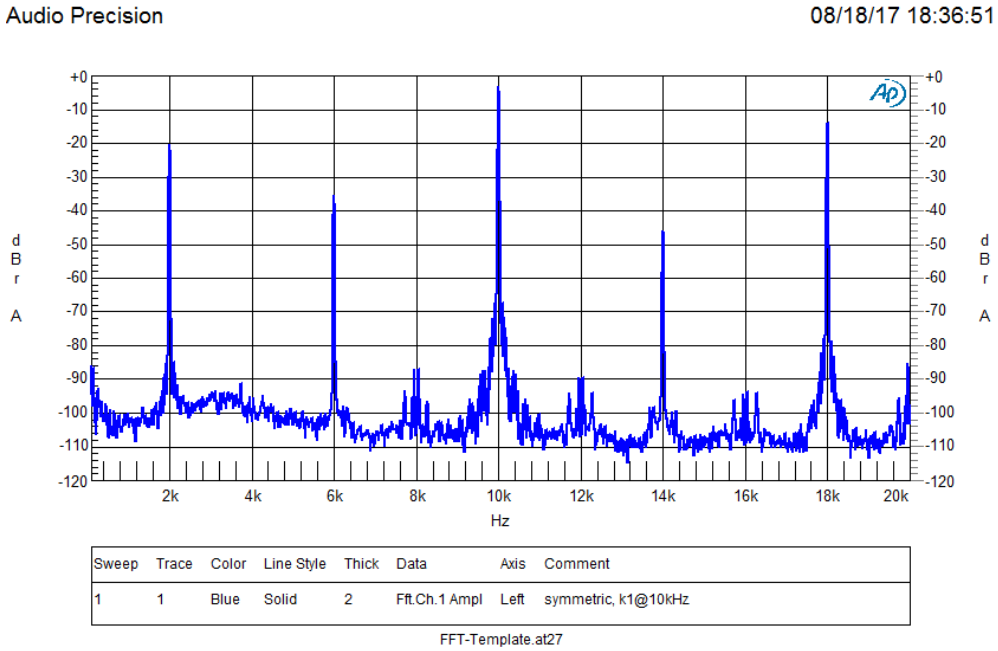
k	relative level	k	relative level
1	0 dB	13	-42 dB
3	-11 dB	15	-40 dB
5	-17 dB	17	-42 dB
7	-24 dB	19	-47 dB
9	-32 dB	21	-56 dB
11	-54 dB	23	-71 dB

Table 4.2.:
Amplitude of the k-th harmonics relative to the first harmonic of a distorted sine with -3 dB_U. Distortion: symmetric overdrive effect, drive factor $dr = 2$.

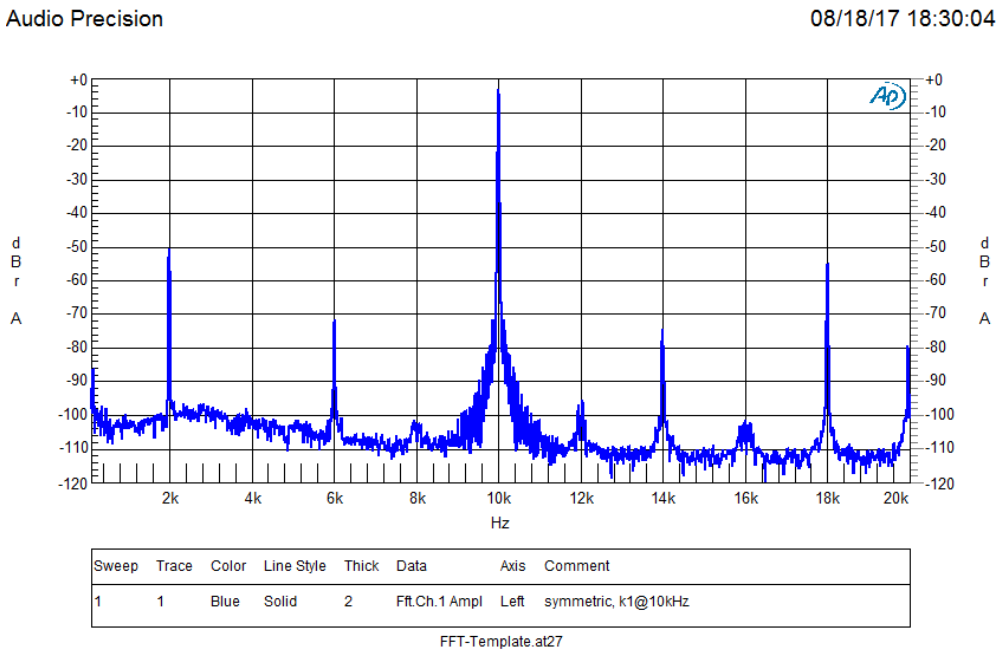
Fundamental frequency f_1	250 Hz	500 Hz	10 kHz	14 kHz
THD + n Symmetric, no upsampling	31.55%	-	31.55%	31.51%
THD + n Symmetric, 4x upsampling	31.58%	-	0.92%	1.99%
THD + n Tube overdrive, no upsampling	-	35.28%	34.25%	28.94%
THD + n Tube overdrive, 2x upsampling	-	35.26%	32.19%	2.78%

Table 4.3.: *Effect of interpolation on nonlinear distortion*

As 4.8b shows, this is indeed where the aliasing artifacts with the highest level show up, with the image of $k = 19$ matching exactly the value from table 4.2, while the artifact produced by $k = 21$ is approximately 3 dB higher than expected. This may be due to superposition of several artifacts.
When re-simulating with *upsampling_filter.m* (see section 3.2.5) under given conditions (Input level at -1.7 dBFS, fundamental frequency $f = 10$ kHz, drive factor $dr = 2$), the simulated THD_R shows the following: $THD_R = 30.6\%$ without upsampling and $THD_R = 0.59\%$ when the input signal is interpolated by factor $L = 4$ beforehand. These values are in the same order of magnitude as the actual measurements and strongly suggest the correctness of our implementation of both the distortion effect as well as the multirate processing As could be shown, by means of upsampling with factor $L = 4$, a significant amount of the aliasing artifacts as seen in figure 4.8a could be reduced to the spectrum in figure 4.8b. In these plots, the symmetric distortion of a sinusoid with frequency $f = 10$ kHz and a drive factor $dr = 2$ is shown. The *System Two* reading indicates a $THD + n$ of 30.12% for the setup without any upsampling. As the input signal does not have uneven harmonics in the audible range, the total harmonic distortion entirely stems from aliasing artifacts. Interpolation by factor $L = 4$ before applying the distortion effect to the same signal results in figure 4.8b. The $THD + n$ reading dropped to less than 1%.



(a) Symmetric overdrive without interpolation



(b) Symmetric overdrive with $4\times$ interpolation

Figure 4.8.:
Spectrum of distorted sinusoid $f = 10$ kHz, -3 dB_U with and without interpolation. Distortion: symmetric overdrive effect, drive factor $dr = 2$.

4.4.2.6. Dynamic range processing

The setup for measurements in the time domain turned out to be suboptimal: The *Focusrite Saffire PRO 24* audio interface introduced significant noise when recording the DSP responses. The lack of calibration tools made exact level control difficult. Nevertheless, the results strongly suggest a correct implementation when visually compared to the outcomes of the dynamics simulation.
To measure the limiter, a test stimulus with the following properties was create: A sine with frequency $f = 1$ kHz is increased in level from 0.0 to 1.0 with steps of 0.1 every second, then decreases back to 0.0 with the same step frequency/magnitude (compare to simulation test signal 2). The result can be seen in figure C.18. The upper graph displays the stimulus, the center is the limiter response with threshold $th = -3$ dBFS. There is noticeable variance in the gain reduction when compared to the simulated limiter. The suspicion arises that this is caused by the approximation of the *fastmath.h* functions. The lower plot reinforces this impression: Swapping out the function `fast_log2()` in favor of `log2()` from the standard *math.h* library produces far more accurate results. The deviation of the gain control when the input signal's amplitude is at full scale is approximately 0.05 or 6.8%. This is a significant difference and bears the question whether the binary logarithmic approximation is acceptable.
Figure C.19 shows the DSP response to the stimulus with activated compressor (center) or expander (bottom). The parameters are as in table 3.4. As said before, the measurement setup skews results and absolute threshold values are estimated. But comparing the recorded file's amplitudes and envelopes to the simulation outcome in figure 3.7 likely suggests a correct implementation of the dynamic range processors.

5. Conclusion

A system model was conceptualized that emulates the most common foundational blocks of modern digital audio mixing consoles while enabling an operator to control various system parameters via remote control. Based on this model, a prototype was first simulated module-wise and afterwards implemented on a D.Module.C6713 DSP development kit in combination with a Matlab App Designer remote control application

The individual signal processing blocks *parametric equalizer*, *panorama*, *dynamic range control* and *overdrive effect* were successfully simulated in Matlab. The blocks can be parametrized with properties that are familiar to a console operator without having to know about the internals of such systems.

During simulation of the overdrive effect, it became clear that the nonlinear distortion causes aliasing that cannot be neglected. Thus, a model for multirate signal processing was conceived and simulated to mitigate aliasing artifacts. The interpolation filters were optimized for a compromise between computational efficiency and stopband rejection. For this purpose, the concept of half band FIR filters has been utilized.

The successful simulation enabled an effective implementation on the D.Module.C6713 DSP development kit. Temporary performance issues have been combated with various optimization techniques, such as approximation of processor taxing mathematical functions, partial refactoring of algorithms as fixed-point implementation and variable interpolation factors.

The Matlab-based parameter control application was implemented following an object-oriented paradigm and utilizes modern software design principles and patterns. This ensures readability of the code base, provides scalability and simplifies the addition of new functional blocks.

The DSP and the Matlab application communicate via serial connection and employ a simple, proprietary protocol to exchange data. The user experience proves that the communication of new signal processing is reliable, changes in the control software are adopted quickly by the DSP without noticeable delay.

The transmission and decoding of data from DSP to control software has been implemented but due to the bad performance of Matlab App Designer applications for live display of measurement data, the topic of metering has eventually not been pursued further. Concluding measurements proved the successful implementation of both control application and DSP program. The measured results match the simulations perfectly. The interpolation of the audio signal prior to its distortion reduces aliasing to a negligible amount.

6. Outlook

As this project was conceived as a prototype rather than as a fully fleshed out product, the code base is occasionally frail and delicate when operated outside of the specification. The first step to further improve the mixing including effects application would be to make the code more robust. An improved data transmission protocol with implementation of 'ACK'/'NAK' flags to acknowledge successful transmission of data (or request retransmission respectively) comes to mind. A handshake protocol between DSP and control application could be established to resynchronize parameters when inconsistencies happen due to a fault in the transmission line. Comprehensive range checks of parameters could be employed in order to ensure no invalid values will be assigned to any variables relevant for signal processing.
Further, the system was designed with scalability in mind. The amount of input channels is not hard coded into the software and is expandable easily. As the overdrive effect (which is the processing block that requires the most cpu time) does not scale with the number of channels, the DSP should be capable to process more input channels without requiring much more optimization.
A glance at the DSP's noise floor reveals that noise is biased towards low frequencies. This is suboptimal from a psychoacoustic point of view, as low frequent noise is perceived as more disturbing as at high frequencies. A noise shaping algorithm in combination with dithering at the output stage could improve the sound quality further.
Different other effects, such as a reverberator could be added to either replace the overdrive block or - after further performance optimization - be operated in parallel with the existing effect.
Finally, the question has to be asked whether the audio codec PCM3003 is suitable for this kind of application. A sampling rate of 48 kHz and 16 bit quantization is not really competitive in this day and age as even digital mixing desks in the lowest price segment offer 24 bit resolution and sampling rates of 96kHz are becoming a norm rather than an exception.

Bibliography

[1] A. Kupke, “Aufsetzen und Debuggen eines CCSv5.5-Projektes für D.module.C6713 DSP mit CODEC D.module.ADDA16,” February 2015, accessed: 2017-07-28. [Online]. Available: https://www.haw-hamburg.de/fileadmin/user_upload/TI-IE/Daten/Labore/Digitale_Signalverarbeitung/pdfs/CCS5.5_dmod6713adda16-Projekt.pdf

[2] Electronic Industries Association, Engineering dept., “Interface between data terminal equipment and data communication equipment employing serial binary data interchange,” Electronic Industries Association, Washington, Tech. Rep., 1969.

[3] T. Görne, *Tontechnik*, 2nd ed. Hamburg: Carl Hanser, 2008.

[4] MIDAS Consoles, *MIDAS XL4 Live Performance Console*, Klark Teknik Group, Kidderminster, 1994.

[5] H. Schmidt, “Audio Program Level, the VU Meter, and the Peak-Program Meter,” *IEEE Transactions on Broadcasting*, vol. 22, no. 1, March 1977.

[6] D. A. Bohn, “Constant-Q Graphic Equalizers,” *Journal of the Audio Engineering Society*, vol. 34, no. 9, September 1986.

[7] E. van Buskirk, “Analysis: Metallica’s Death Magnetic Sounds Better In Guitar Hero,” September 2008, accessed: 2017-07-27. [Online]. Available: https://www.wired.com/2008/09/does-metallicas

[8] E. B. Union, “Loudness Metering: ’EBU Mode Metering To Supplement EBU R 128 Loudness Normalization,” European Broadcasting Union, Geneva, TECH 3341, January 2016, accessed: 2017-07-28. [Online]. Available: https://tech.ebu.ch/docs/tech/tech3341.pdf

[9] F. T. Agerkvist, “Volterra Series Based Distortion Effect,” in *Proc. 129th AES Convention*, no. 8212. San Francisco: Audio Engineering Society, November 2010.

[10] H. Zumbahlen (Editor), *Basic Linear Design*. Analog Devices, 2007.

[11] U. Zölzer, *Digitale Audiosignalverarbeitung*, 2nd ed. Stuttgart: B.G.Teubner, 1997.

[12] R. Clark, E. Ifeachor, G. Rogers, and P. van Eetvelt, "Techniques for Generating Digital Equalizer Coefficients," *Journal of the Audio Engineering Society*, vol. 48, no. 4, April 2000.

[13] H. Korhola and M. Karjalainen, "Perceptual Study and Auditory Analysis on Digital Crossover Filters," *Journal of the Audio Engineering Society*, vol. 57, no. 6, June 2009.

[14] R. Bristow-Johnson, "The Audio EQ Cookbook," 2004, accessed: 2017-07-23. [Online]. Available: https://shepazu.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html

[15] U. Zölzer and M. Holters, "Parametric Recursive Higher-Order Shelving Filters," in *Proc. 120th AES Convention*, no. 6722. Paris: Audio Engineering Society, May 2006.

[16] K.-D. Kammeyer and K. Kroschel, *Digitale Signalverarbeitung*, 5th ed. Wiesbaden: Springer Fachmedien, 2002.

[17] M. E. Frerking, *Digital Signal Processing In Communication Systems*, 1st ed. New York: Van Nostrand Reinhold, 1994.

[18] U. Zölzer (Editor), *DAFX Digital Audio Effects*, 2nd ed. Chichester: John Wiley & Sons, 2003.

[19] H. Haas, "The Influence of a Single Echo on the Audibility of Speech," *Journal of the Audio Engineering Society*, vol. 20, no. 2, March 1972.

[20] J. Blauert, "Räumliches Hören," in *Handbuch der Audiotechnik*, 1st ed., S. Weinzierl, Ed. Berlin, Heidelberg: Springer, 2008, ch. 3.

[21] V. Pulkki, "Virtual Sound Source Positioning Using Vector Base Amplitude Panning," *Journal of the Audio Engineering Society*, vol. 45, no. 6, June 1997.

[22] A. D. Blumlein, "British Patent Specification 394,325 (Improvements in and relating to Sound-transmission, Sound-recording and Sound-reproducing Systems," *Journal of the Audio Engineering Society*, vol. 6, no. 2, April 1958.

[23] N. Fliege, *Multirate Digital Signal Processing*, 1st ed. Chichester: John Wiley & Sons, 1994.

[24] H. W. Schüßler, *Digitale Signalverarbeitung 1 - Analyse diskreter Signale und Systeme*, 4th ed. Berlin, Heidelberg: Springer, 1994.

[25] L. R. Rabiner, J. H. McClellan, and T. W. Parks, "Chebyshev Approximation for Nonrecursive Digital Filters with Linear Phase," *IEEE Transactions on Circuit Theory*, vol. 19, no. 2, March 1972.

[26] W. Fraser, “A Survey of Methods of Computing Minimax and Near-Minimax Polynominal Approximations for Functions of a Single Independent Variable,” *Journal of the ACM*, vol. 12, no. 3, July 1965.

[27] T. Sandmann, *Effekte und Dynamics*, 6th ed. Bergkirchen: PPV Medien, 2007.

[28] I. Dash, “True Peak Metering - a Tutorial Review,” in *Proc. 136th AES Convention*, no. 9041. Berlin: Audio Engineering Society, April 2014.

[29] D. Shmilovitz, “On the Definition of Total Harmonic Distoriton and Its Effects on Measurement Interpretation,” *IEEE Transactions on Power Delivery*, vol. 20, no. 1, January 2005.

[30] P. van der Linden, “Expert c programming: Deep c secrets,” accessed: 2017-08-15. [Online]. Available: http://www.electroons.com/8051/ebooks/expert%20C%20programming.pdf

[31] R. Chassaing and D. Reay, *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK*, 2nd ed. Chichester: John Wiley & Sons, 2008.

[32] S. Kleuker, *Software Engineering*, 3rd ed. Wiesbaden: Springer Vieweg, 2013.

[33] A.-T. Schreiner, *Object-Oriented Programming with ANSI-C*. Axel-Tobias Schreiner/Lulu, October 2011.

[34] “serial (Official Matlab documentation),” accessed: 2017-08-24. [Online]. Available: https://de.mathworks.com/help/matlab/ref/serial.html

[35] R. Lyons, “Improved narrowband low-pass IIR filters in fixed-point systems [DSP tips & tricks],” *IEEE Signal Processing Magazine*, vol. 26, no. 2, March 2009.

[36] Texas Instruments, *PCM3002 / PCM3003 Data Sheet*, October 2004.

[37] A. Kupke, “Code Profiling im Simulator,” February 2013, accessed: 2017-08-15. [Online]. Available: https://www.haw-hamburg.de/fileadmin/user_upload/TI-IE/Daten/Labore/Digitale_Signalverarbeitung/pdfs/CCS5_ProfilingCode_de.pdf

[38] T. Support, “Tms320c671x: Error: Illegal opcode (403a0000) at pc = 0xb00085ec illegal opcode at pc = 0xb00085ec,” January 2012, accessed: 2017-08-15. [Online]. Available: https://e2e.ti.com/support/dsp/tms320c6000_high_performance_dsps/f/115/p/157856/575226

[39] C. Lomont, “Fast Inverse Square Root,” February 2003, accessed: 2017-08-15. [Online]. Available: http://www.lomont.org/Math/Papers/2003/InvSqrt.pdf

Glossary

abstract a type of classes, properties and methods related to the object-oriented software paradigm. Classes with abstract members or declared as abstract cannot be instantiated and must be inherited from instead. 89

aliasing violation of the Nyquist sampling theorem. Any signal with frequency $f > f_s/2$ folds itself back to an aliasing artifact $f' < f_s/2$. If $f_s/2 < f < 3 \cdot f_s/2$, then $f' = |f_s - f|$. 18, 60

bleeding when recording several acoustic sources simultaneously with microphones in close proximity to the sources, these microphones will also pick up the sources farther away. This kind of signal crosstalk is called *bleeding*. 32, 62

BNC Bayonet Neill Concelman, a lockable connector used for coaxial cable. 18

busy waiting repeatedly checking whether the state of a variable/flag/mutex has been changed by another thread, process or interrupt. 93

CCS Code Composer Studio. An **IDE** for programming and debugging Texas Instruments devices. Based on the open source IDE *Eclipse*. 17

clipping strictly limiting a signal to a certain threshold, *clipping* off anything that exceeds this level. Causes high amount of harmonic distortion. 31

codec Portmanteau of *coder* and *decoder*. A device for en- and decoding digital data streams, such as digital audio. 18

context switch occurs when a CPU switches to a different environment/thread, i.e. to handle an interrupt. Program counter, stack pointer and register values are stored away and later reloaded to commence the computing once the interrupt handling is finished. 79

CPU cache fast on-chip memory. CPUs often have several levels of cache. These are denoted by an *L1, L2, L3,..* prefix. Lower numbers are equivalent to faster memory. 17

D.I. box Direct Injection box, small devices that are put in between signal source and mixing desk for reasons like impedance conversion, galvanic isolation or balancing of an unbalanced signal. 24

DAW a Digital Audio Workstation is a computer software for mixing, recording, editing and producing audio files. 36, 100

dB$_U$ Level with reference value $0.775V$. Predominant in Europe. 23, 101

dB$_V$ Level with reference value $1V$. Predominant in USA. 23

dBFS Full Scale level, amplitude of a signal relative to the maximum amplitude a device can handle before **clipping** occurs. 14

DMA Direct Memory Access allows for peripherals to read from and write to memory independent from the CPU. A DMA controller handles the interaction of hardware subsystems and central processing unit, relieving the latter from the task of data transmission. 17, 76

dropout Partial loss of samples in a signal stream. Cause can be a faulty connection or a processor that is not powerful enough to handle the signal stream without interruption. 96, 97

DSP Digital Signal Processor, specialized microprocessor, optimized for signal processing applications. 12, 17

Dyadic Cascading Interpolation or decimation with factor 2^k, $k \in \mathbb{N}$ by cascading k interpolation/decimation stages with factor 2. 50, 70

EMIF External Memory InterFace, bus protocol for communication between processors and memory devices such as SDRAM or memory-mapped peripherals. 77

FIFO First In, First Out describes a data buffer and the method with which data is handled on the buffer: The oldest ('first') data in the buffer will be processed first. 20

GUI Graphical User Interface, interaction with program by enabling the use of icons or other visual indicators. 14

I^{2}C Inter-Integrated Circuit, serial data bus for communication between a microcontroller and peripheral integrated circuits. 17

IDE Integrated Development Environment, a software application for software development that provides a comprehensive toolchain from source code editing to build automation and debugging. 17, 18, 83

Loudness War The desire of audio engineers and producers to mix their music as loud as possible. This resulted in an ongoing trend of increasing audio levels in records over the years that has been ill-received by many critics. 31

magic numbers An anti-pattern in software architecture. This refers to the use of arbitrary numbers in source code (arguments of function calls or calculations) instead of unambiguously named constants. This practice severely impairs the intelligibility of code. 84

makeup gain a compressor attenuates signal segments above a certain threshold. The compressor's makeup gain then boosts it in level again so the signal increased in loudness overall. 32

P.A. Public Address system, a sound amplification system meant to play back music or speech at high volume to a big audience at a conference, a concert hall, a stadium or similar. 34

phantom power switchable dc voltage source applied to the balanced wires of a microphone input. Intended to power condenser microphones, **D.I. boxes** and other active circuits. 23

PLL Phase Locked Loop, control system utilized (among other uses) for generating stable clock signals. 77

plug-in additional software for **DAW**s that expand on their functionality and provide digital effects or virtual instruments. 36

ringing artifacts the artifact that occurs due to a non-monotonic, oscillating step response of a filter. They can be audible, especially when filtering percussive or other signals rich in transients. 39

RS 232 Standard for serial communication. 12

SDRAM Synchronous Dynamic Random-Access Memory, a type of random access memory. 17, 77

SPI Serial Peripheral Interface, protocol for communication between devices in a master/slave configuration. 17

SRAM Static Random-Access Memory, a type of random access memory, generally faster than dynamic RAM. 17

static (C keyword) functions and global variables qualified as `static` are only visible within the translation unit they are declared in. Local variables qualified as `static` is allocated at compile time and keeps its value beyond its scope, i.e. stays persistent between multiple function calls. 77

threshold the level at which a dynamics processor starts operating, either when the signal exceeds the threshold or when it falls below. 30, 52

translation unit the output file of the C preprocessor that is to be compiled. Contains the .c source file (with all macros expanded) as well as all header files specified by an `#include` directive. 77

VLIW Very Long Instruction Word processor architecture allows for execution of multiple instructions in parallel. 17

A. Instructions for setup and operation of the *MixMaster* application

This section provides a brief instruction on how to start and operate the thesis project on DSP and on PC. For further questions, feel free to inquire Prof. Dr. Jürgen Vollmer at the Hochschule für Angewandte Wissenschaften Hamburg about the contact details of the author.

Prerequisites To start the MixMaster mixing and effects application, following electronic and physical resources are needed:

- D.Module.C6713 DSP development kit with D.Module.PCM3003 audio codec daughter card
- Windows 7 PC with *CodeComposerStudio* v.5.5 and *Matlab* v.R2016b
- *Spectrum Digital* XDS510 USB JTAG emulator
- USB-to-Serial converter
- The Code Composer Studio project *MixMaster V2*
- The Matlab App Designer project *MixMasterControlUtility*

As the COM port for communication with the DSP is hard coded into the *MixMasterControlUtility*, the first step is to find out which COM port the USB-to-Serial converter occupies on the pc hosting the App Designer application. This can be done via Windows device manager. Then, an appropriate modification must be undertaken in the file *serial_manager.m*:

Listing A.1: *serial_manager.m: Setting up COM port*

```
25 obj.serial_handle = serial('COM6', 'BaudRate', 9600, 'Terminator', 'CR',
       'DataBits', 8);
```

Here, `'COM6'` has to be replaced with the actual port.
In the unfortunate case that the CCS project is supposed to run on a DSP board where the ADC and DAC connectors have been poorly soldered, specifically: where the pin rows 0 and

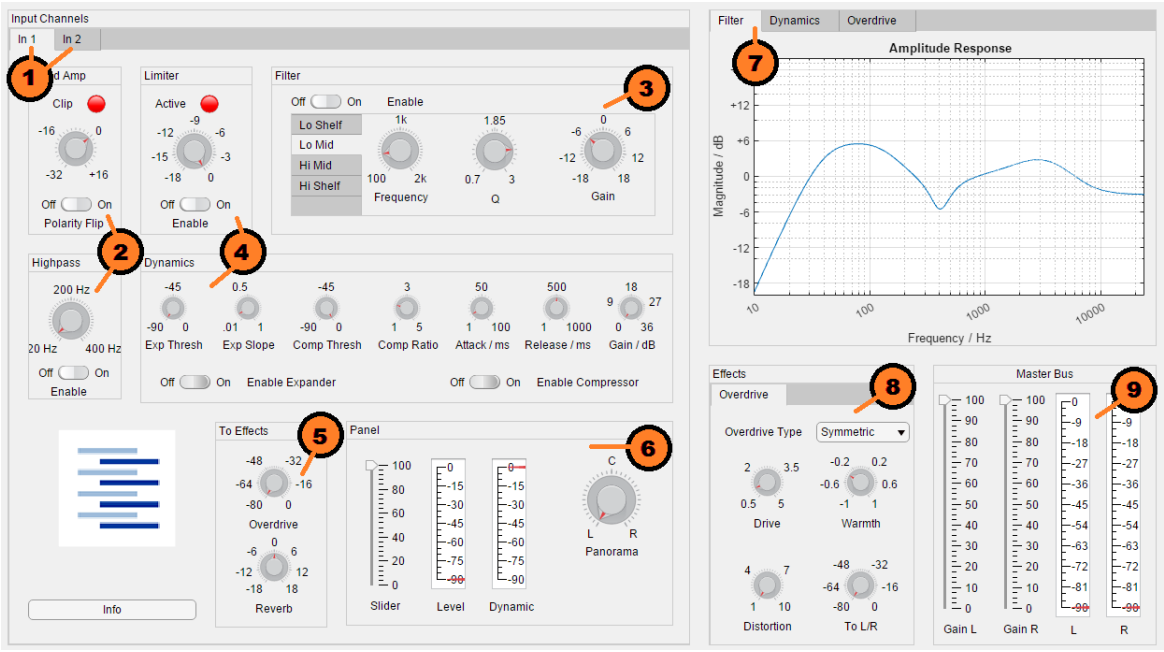


Figure A.1.: *Overview of the MixMasterControlUtility user interface*

1 as well as the rows 31 and the 32 have been swapped by accident, an offset has to be set:

Listing A.2: *main.c: offset for in/out port correction*

```
32 #define MORON_OFFSET 4
```

With this constant, input/output ports 4 and 5 of the D.Module.PCM3003 will be utilized instead of 0 and 1. If not needed, set `MORON_OFFSET` to 0.

Program start To ensure proper audio pass-through without dropouts, select the *MixMaster v2* project in CCS and navigate to *Project* → *Properties* → *Optimization*. Make sure that 'Optimization level' is set to '3'. Start debugging the program (F11) but do not yet run it.
The *MixMasterControlUtility* can be started by double-clicking the *MixMaster.mlapp* file. Once the control surface has loaded, run the DSP program (F8).

Operation of the *MixMasterControlUtility* Once both the *MixMaster* and the control utility are running, operation is straightforward. The *MixMasterControlUtility* presents itself as seen in figure A.1.

1. With this tab group, the operator can switch between the input channel layers. Doing so sets all channel-specific control elements to the position according to the selected

channel. If 'Filter' or 'Dynamics' is selected in the figure view, the plot changes according to the selected channel's settings as well

2. The input gain and high pass section precondition the signal. Both high pass and polarity flip can be enabled and disabled via the switches.

3. On the left side of this box, the parametric equalizer band can selected by clicking on one of the four tabs. The on/off button enables disables all equalizers of this channel, not just the currently selected band.

4. The limiter and compander section can be enabled individually. The limiter processes the signal before the compander gain factor is calculated. The attack and release applies to both compressor and expander; the limiter operates on hard coded time constants.

5. The upper of the two knobs in the aux routing section determines the attenuation with which the channel signal will be routed to the overdrive effect input. The lower of the two knobs does not have any use yet.

6. The slider determines the attenuation factor applied to the signal before routed onto the master bus, the panorama knob distributes between left and right master bus.

7. This tab group enables switching between the figures to display. Filter shows the current amplitude response of the selected channel's high pass and equalizer section. Dynamics display the static curve of limiter, compressor and expander. Overdrive displays the static curve of the distortion effect.

8. In this box, the overdrive effect parameters can be set. The drop down menu enables the choice of overdrive, however, the type 'exponential' is not yet implemented. Enabling it basically bypasses the effect. For 'Symmetric' type, only the knobs 'Drive' affects the characteristic of the distortion. 'To L/R' routes the distorted signal back onto the master bus.

9. The master bus section enables control over the attenuation with which the master buses are

Known issues As the *MixMaster* is not a fully finished application, it can sometimes display erratic behavior or lead to program crashes. A list of known issues and possible workarounds:

- After startup, the DSP may not react to any parameter changes made in the *MixMasterControlUtility*. This is likely caused by an optimization issue where the compiler changes the behaviour of the parsing function `parse_float_from_hex_string(char *string)` in *param_exchange.c* so that floating point values can no longer successfully be parsed from ASCII strings. A workaround is to restart CCS, set compiler optimization to 'off', clean, build and run the project. Run the program and verify that parameter changes in the GUI are now adopted by the DSP. Quit debugging and reset optimization to '3', rebuild and run. The system should now behave as expected.
- The parameters of GUI application and DSP are not synchronized. If one of both components are restarted, or changes in the *MixMasterControlUtility* are made while the serial connection is interrupted, the parameters shown in the GUI are not consistent with the actual DSP parameters. Thus it is recommended to always restart both application when one needs to be reset or when the connection has been interrupted.
- The figures displaying the filter amplitude response and the dynamics characteristic do not take into account the state of the various on/off switches.
- Rarely it occurs that due to a crash or a software bug, the Matlab serial object is not deleted properly. This results in a *Port not available* error when restarting the *MixMasterControlUtility*. Matlab (not just the App Designer!) has to be restarted in order to run the control software again.
- Aforementioned error can be triggered by trying to re-run the application while there still is an instance running. Always close the control software before attempting to run it again.
- As discussed before, elements for metering and visual feedback (i.e. 'Limiter Active' lamp) purposes are in place but not in use. This may or may not change with future updates of the Matlab App Designer.

B. Pictures of measurement setups

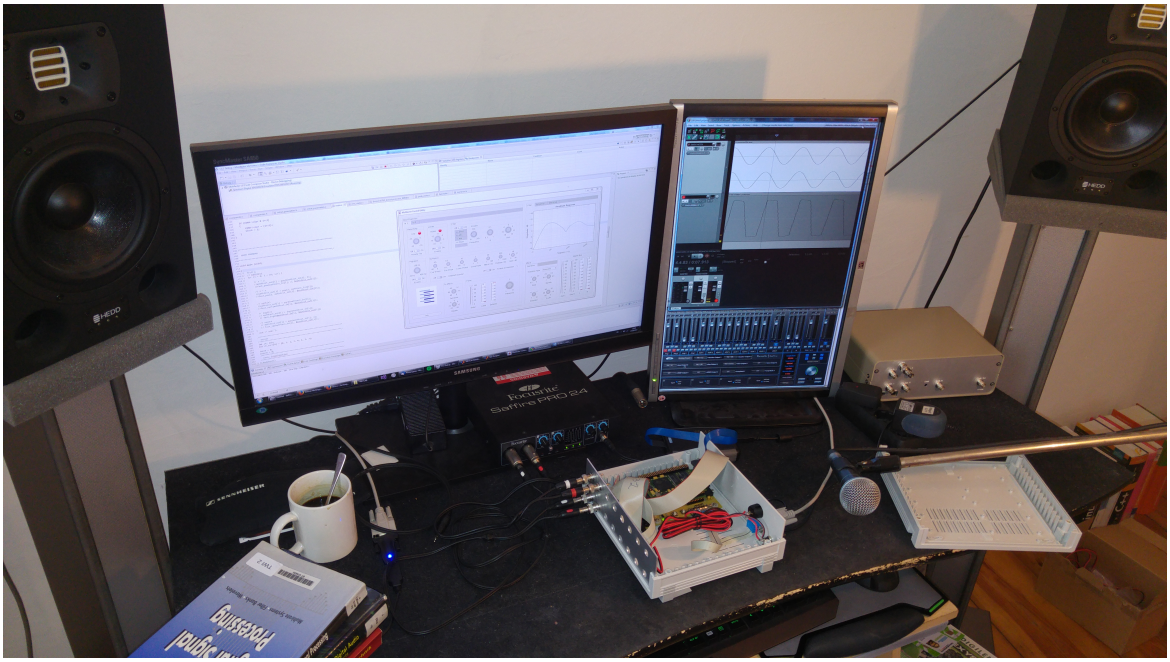


Figure B.1.: *Setup for development, debugging and time-domain measurements*

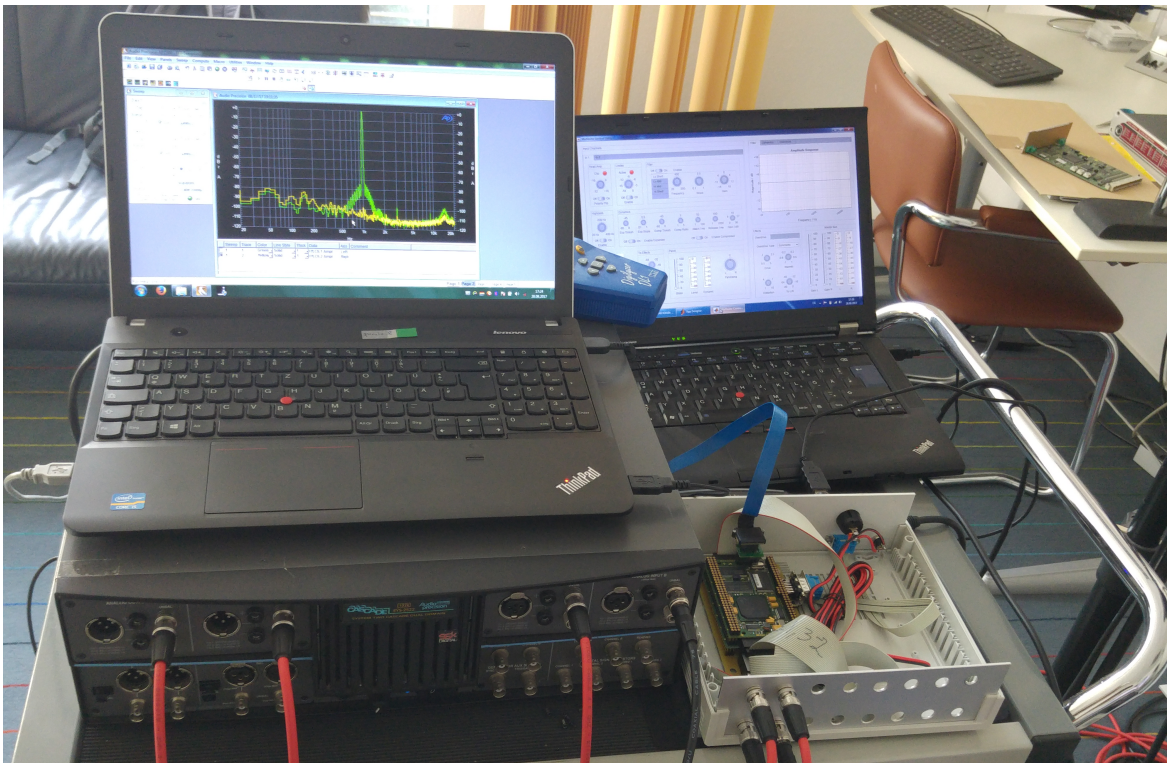


Figure B.2.: *Setup for frequency-domain measurements at Jünger Audio GmbH*

C. Measurements

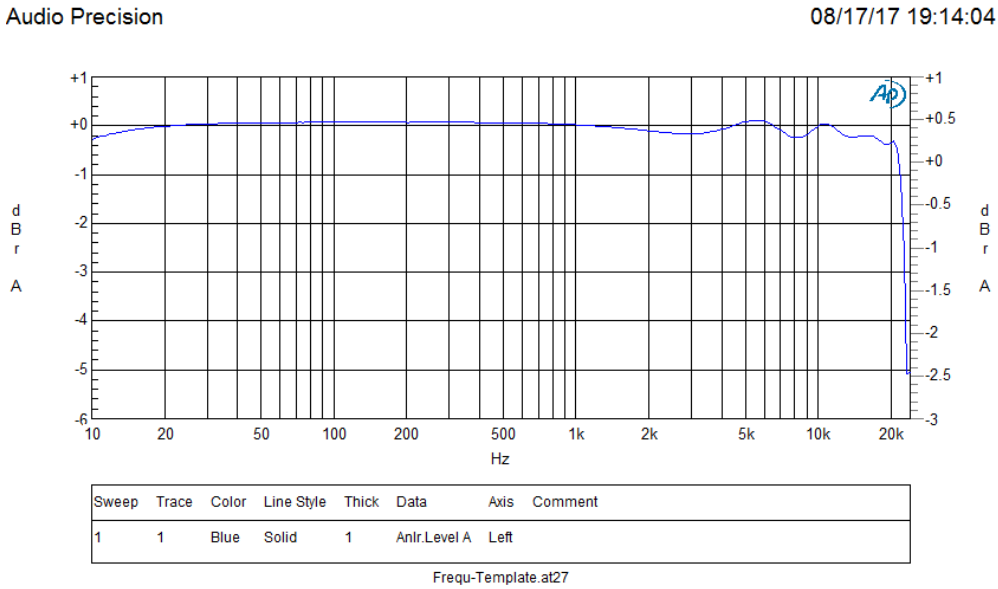


Figure C.1.:

Magnitude response of the DSP with all signal processing blocks disabled. $0dBr = -3dBu$

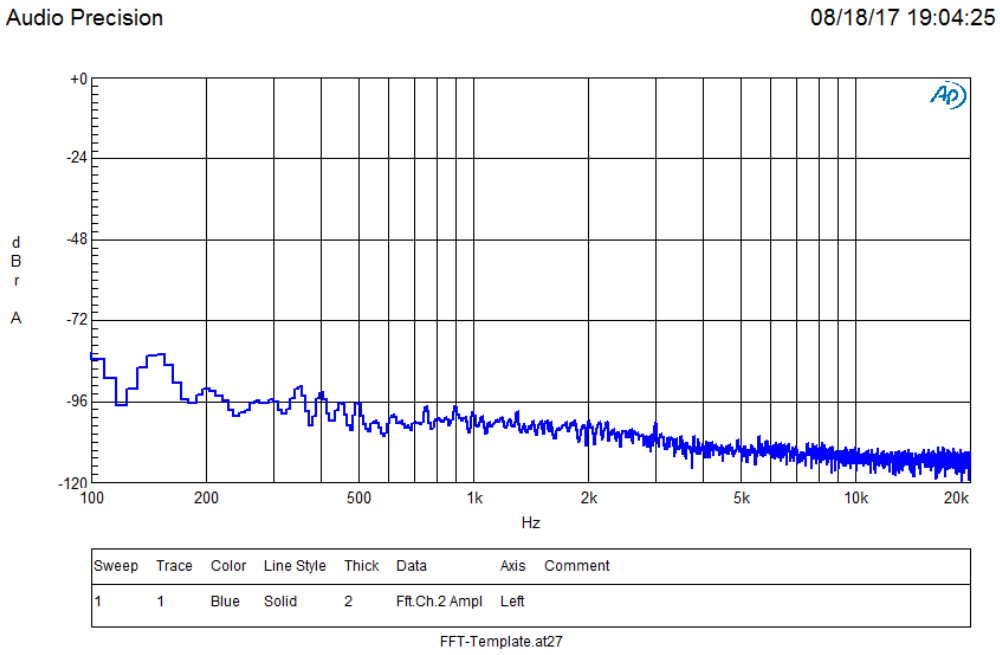


Figure C.2.: *Noise floor of DSP. 0dBr = −3dBu*

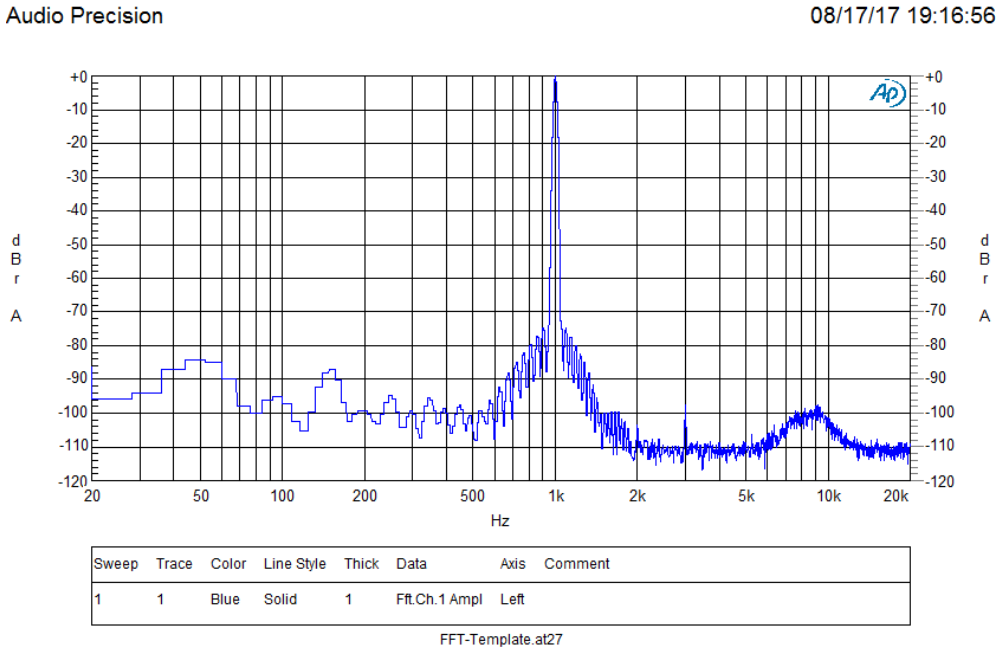


Figure C.3.:
THD+n measurement of DSP with all signal processing blocks disabled. 0dBr = −3dBu

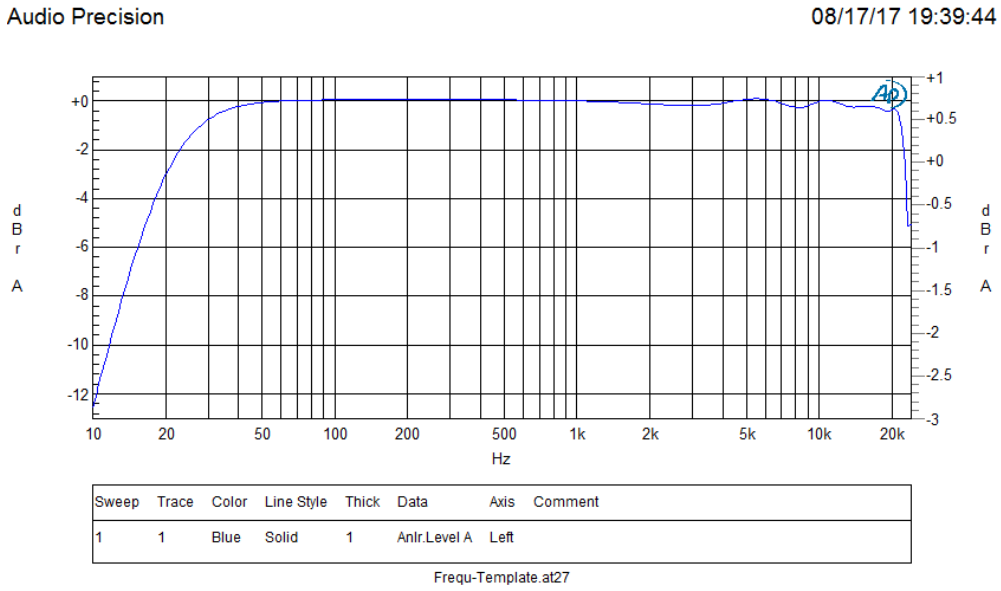


Figure C.4.: *High pass filter with cutoff frequency $f_0 = 20Hz$. $0dBr = -3dBu$*

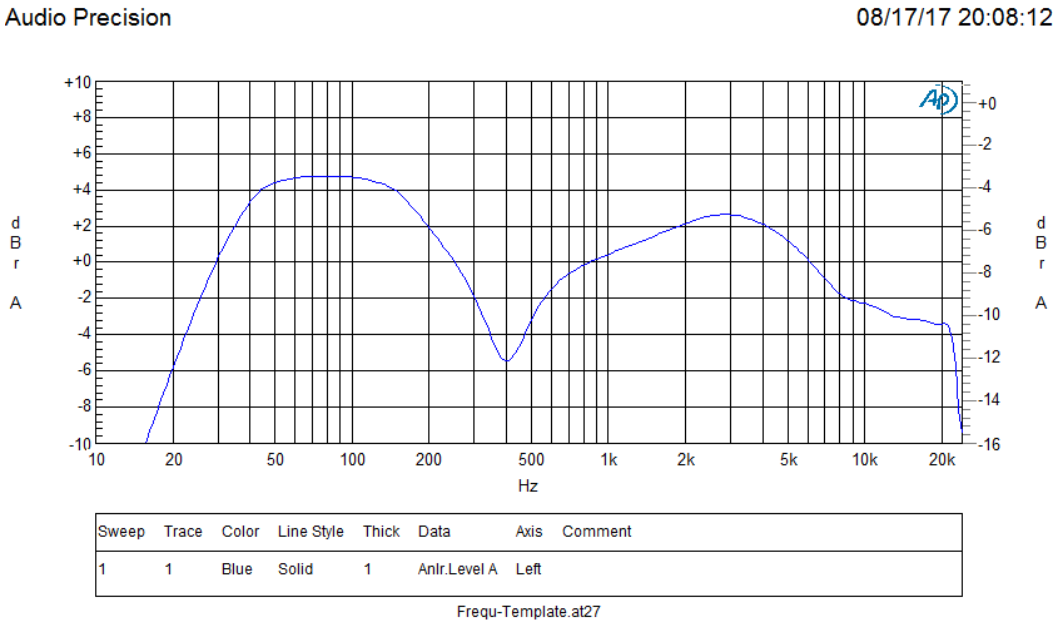


Figure C.5.: *Filter cascade with specification as in table 3.1. $0dBr = -6dBu$*

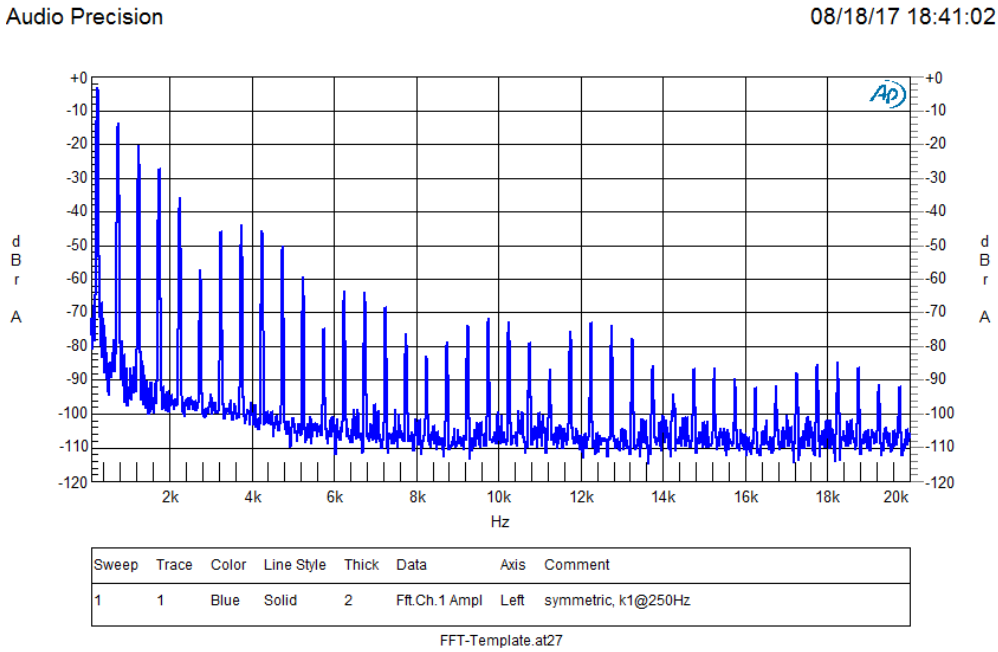


Figure C.6.:

Spectrum of symmetric overdrive with fundamental frequency $f_1 = 250$*Hz, drive* $dr = 2$*, no interpolation.* $THD + n = 31.55\%$

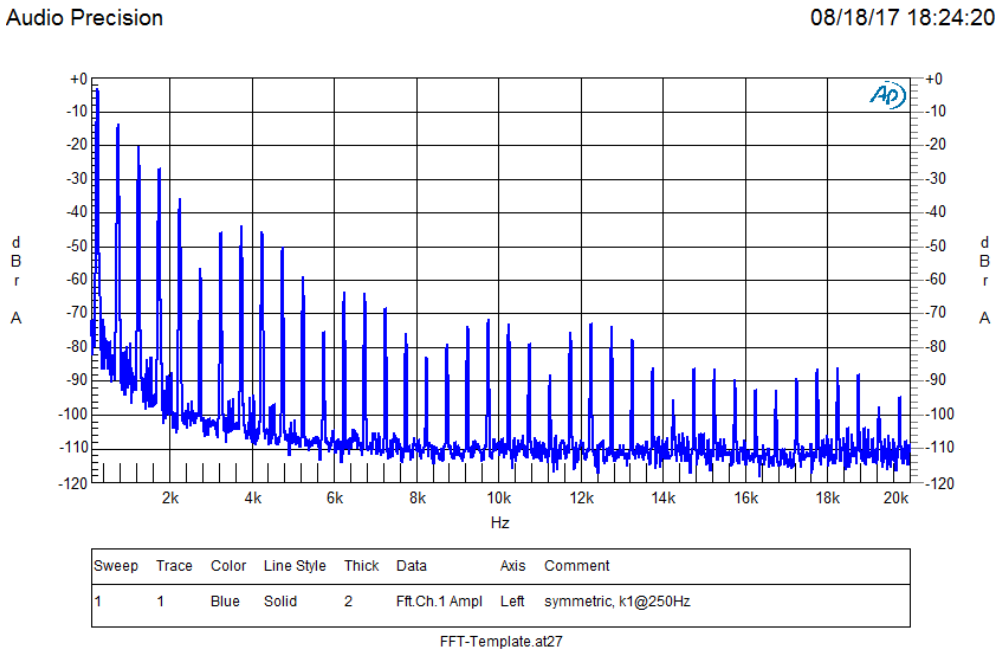


Figure C.7.:

Spectrum of symmetric overdrive with fundamental frequency $f_1 = 250$*Hz, drive* $dr = 2$*,* $4\times$ *interpolation.* $THD + n = 31.58\%$

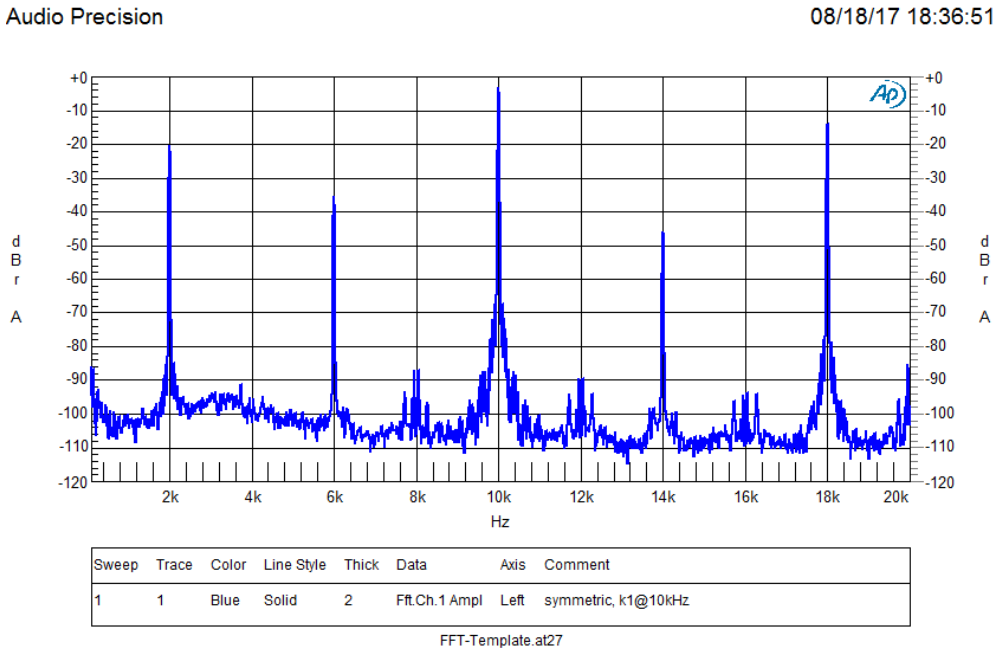


Figure C.8.:
Spectrum of symmetric overdrive with fundamental frequency $f_1 = 10$kHz, drive $dr = 2$, no interpolation. $THD + n = 30.12\%$

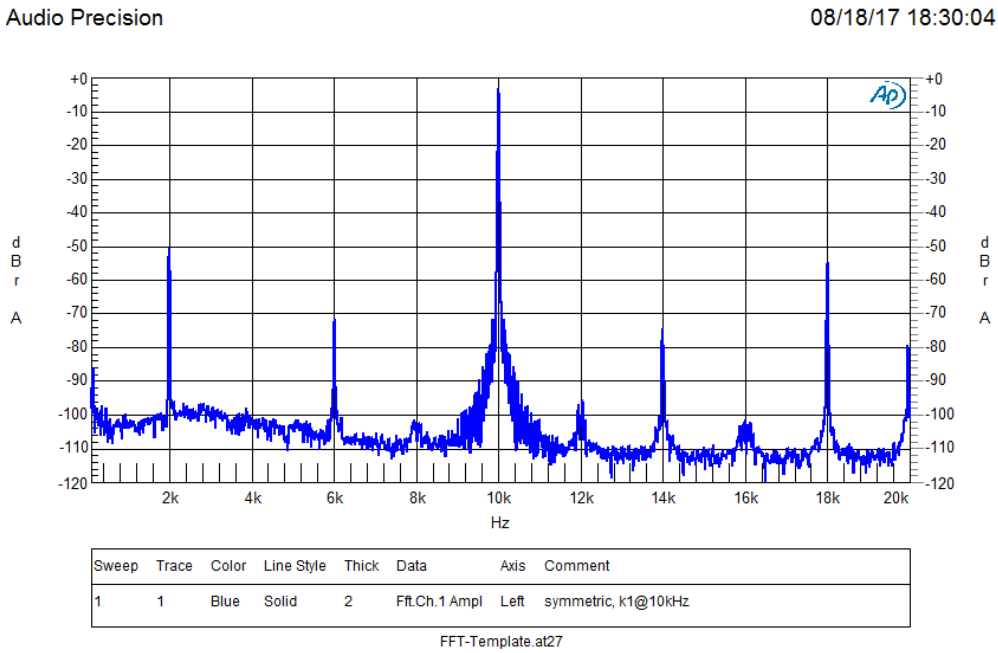


Figure C.9.:
Spectrum of symmetric overdrive with fundamental frequency $f_1 = 10$kHz, drive $dr = 2$, $4\times$ interpolation. $THD + n = 0.92\%$

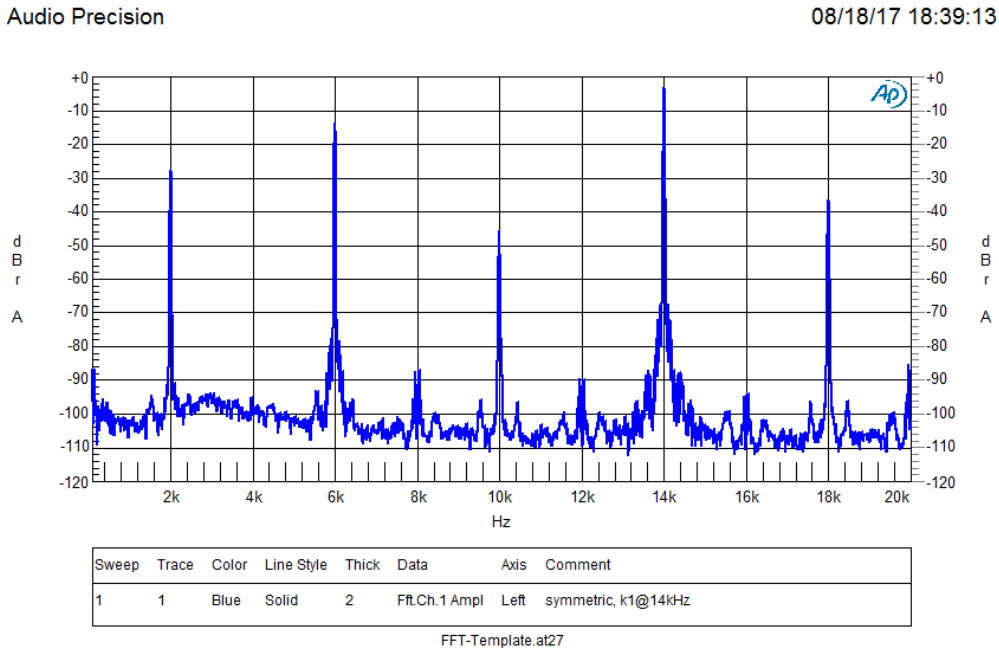


Figure C.10.:
Spectrum of symmetric overdrive with fundamental frequency $f_1 = 14$kHz, drive $dr = 2$, no interpolation. $THD + n = 31.51\%$

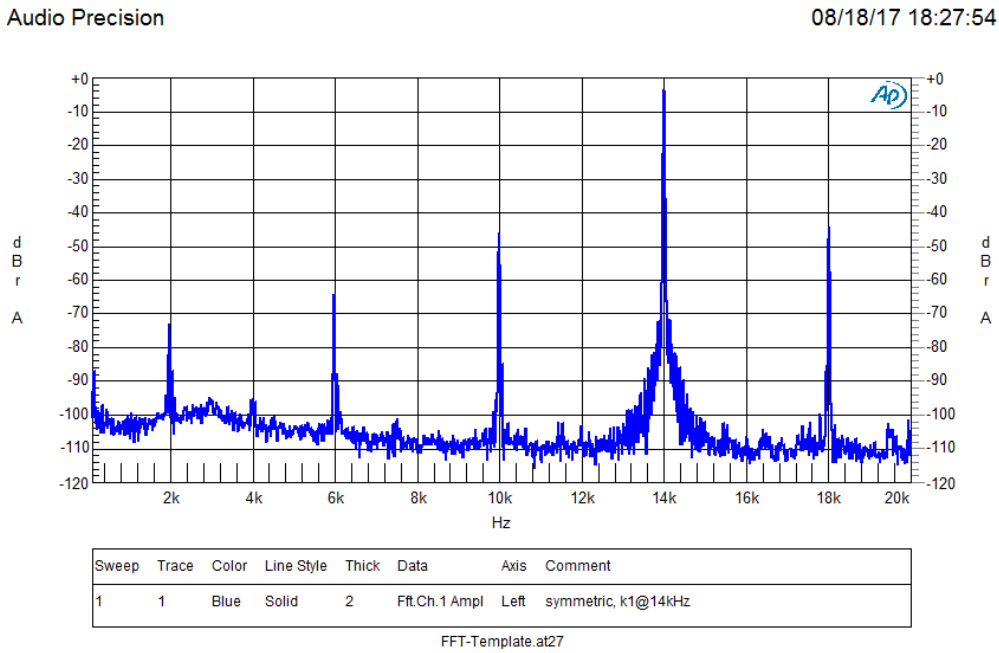


Figure C.11.:
Spectrum of symmetric overdrive with fundamental frequency $f_1 = 14$kHz, drive $dr = 2$, $4\times$ interpolation. $THD + n = 1.99\%$

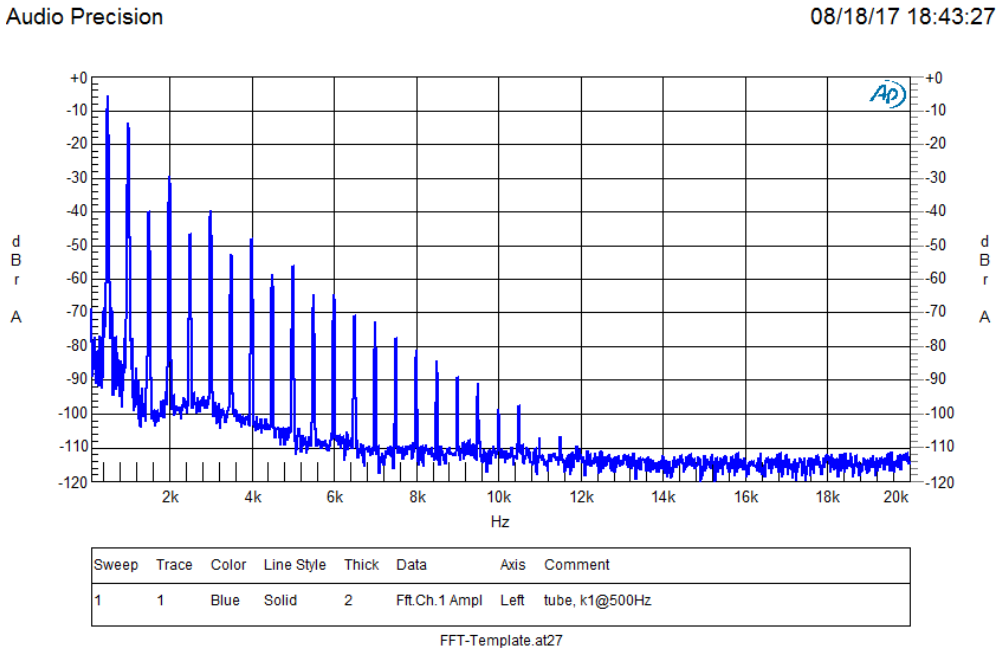


Figure C.12.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 500Hz$*, drive* $dr = 2$*, working point* $Q = -0.1$*, distortion* $dist = 8$*, no interpolation.* $THD + n = 35.28\%$

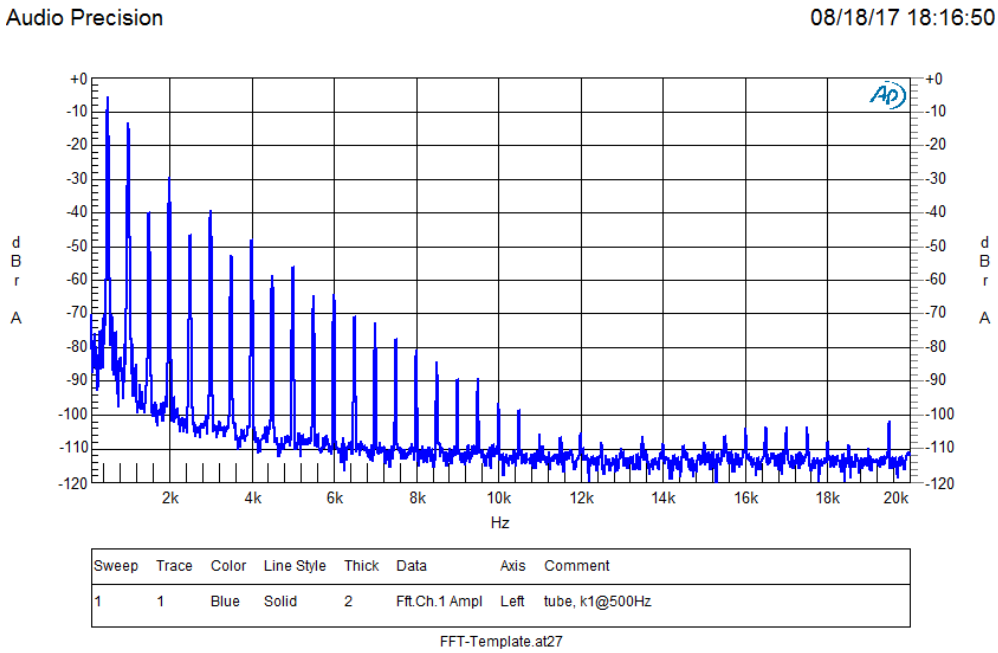


Figure C.13.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 500Hz$*, drive* $dr = 2$*, working point* $Q = -0.1$*, distortion* $dist = 8$*,* $2\times$ *interpolation.* $THD + n = 35.26\%$

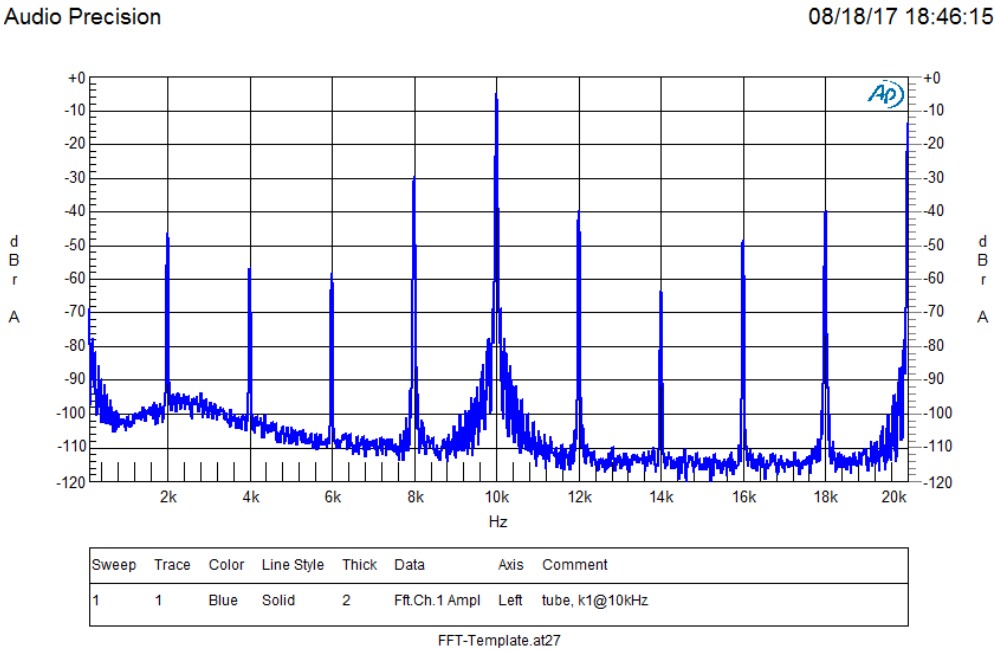


Figure C.14.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 10$kHz, drive $dr = 2$, working point $Q = -0.1$, distortion $dist = 8$, no interpolation. $THD + n = 34.25\%$

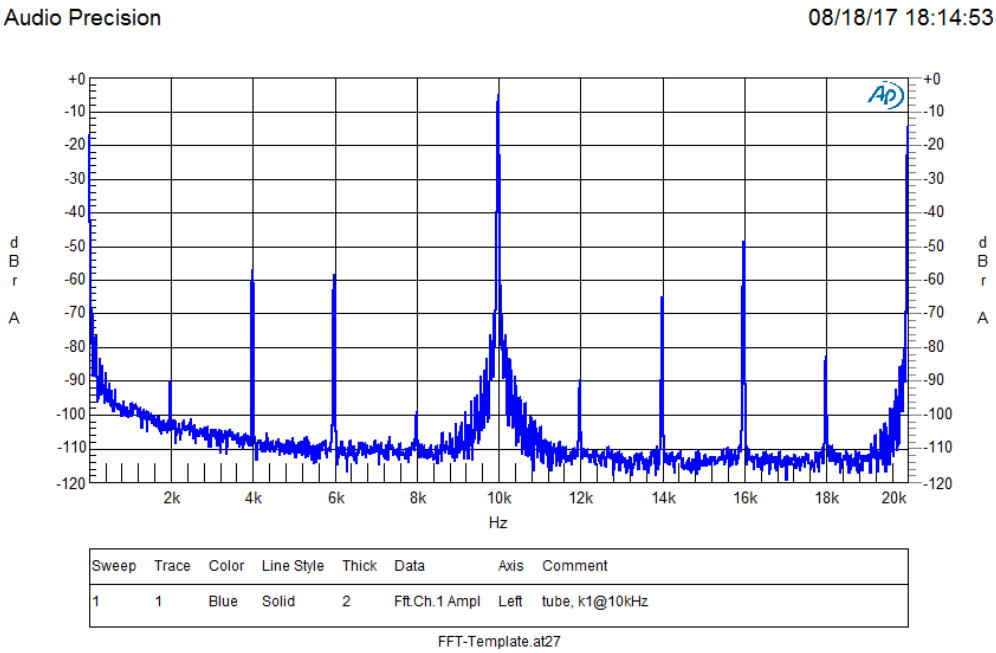


Figure C.15.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 10$kHz, drive $dr = 2$, working point $Q = -0.1$, distortion $dist = 8$, $2\times$ interpolation. $THD + n = 32.19\%$

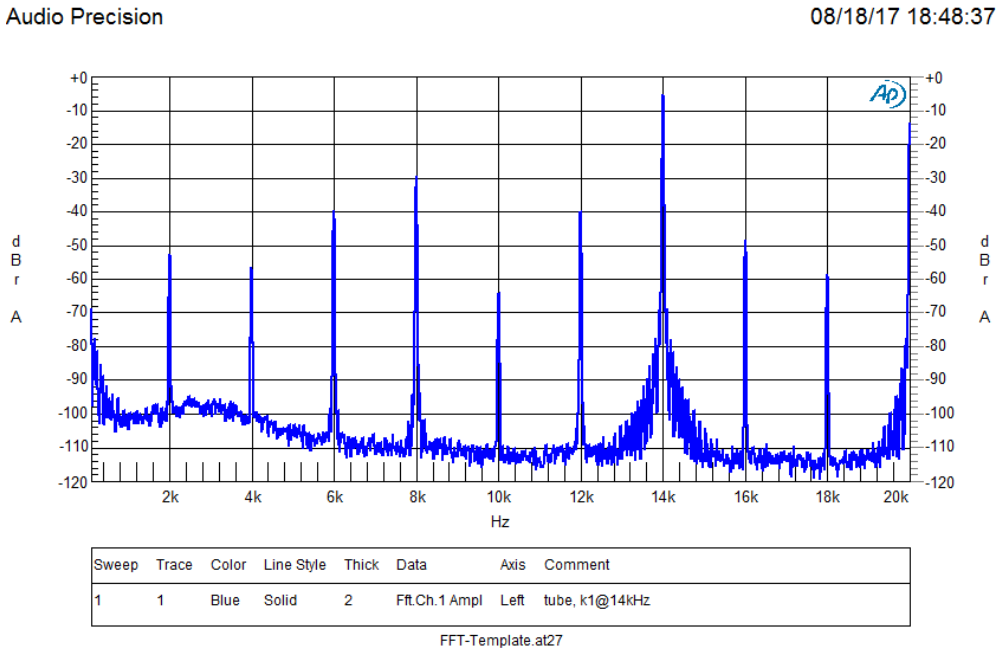


Figure C.16.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 14$*kHz, drive* $dr = 2$*, working point* $Q = -0.1$*, distortion* $dist = 8$*, no interpolation.* $THD + n = 28.94\%$

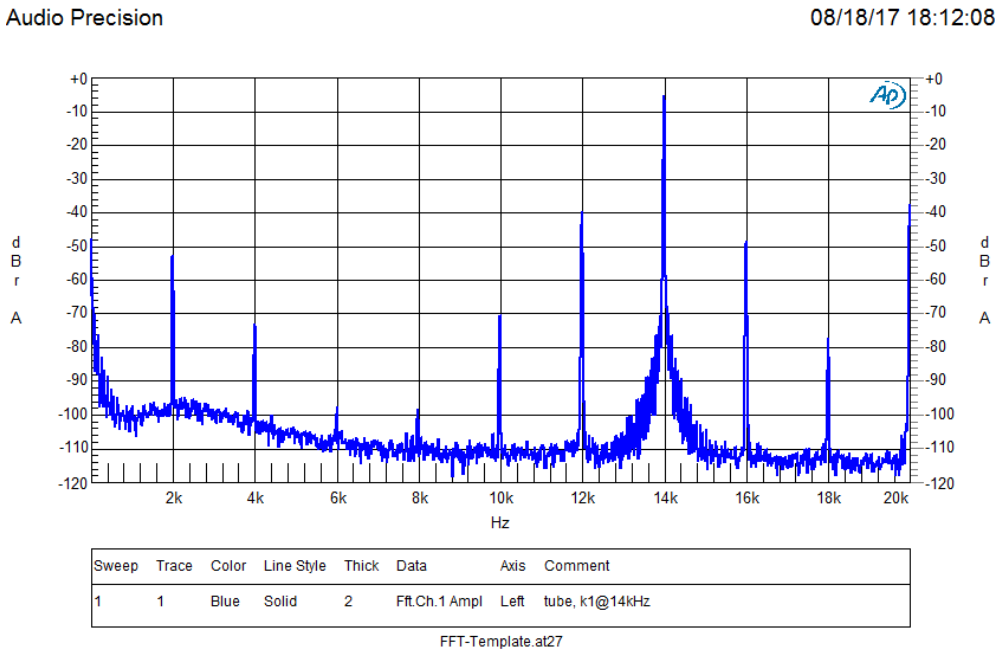


Figure C.17.:

Spectrum of tube overdrive with fundamental frequency $f_1 = 14$*kHz, drive* $dr = 2$*, working point* $Q = -0.1$*, distortion* $dist = 8$*,* $2\times$ *interpolation.* $THD + n = 2.78\%$

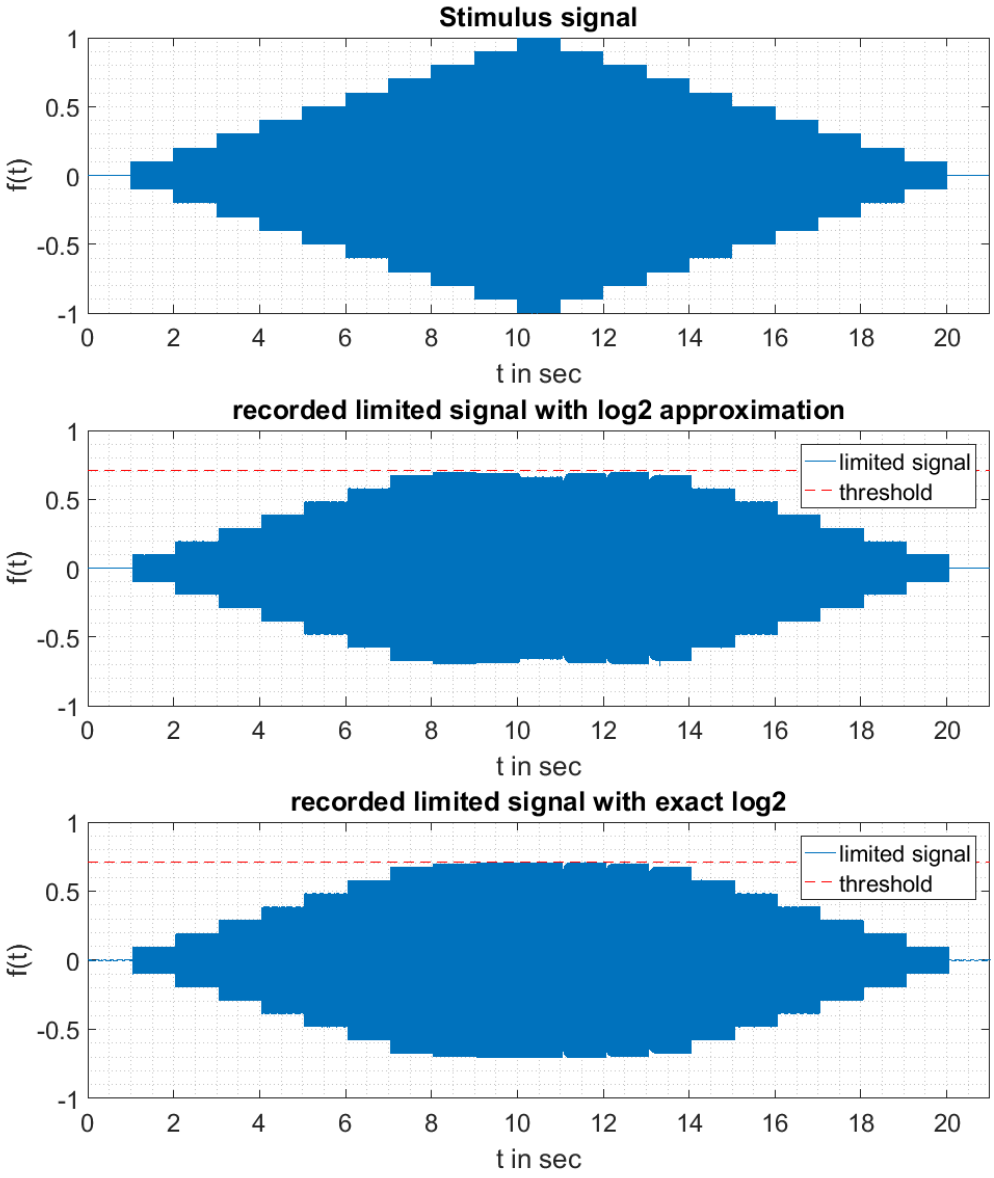


Figure C.18.: *Limiter measurement*

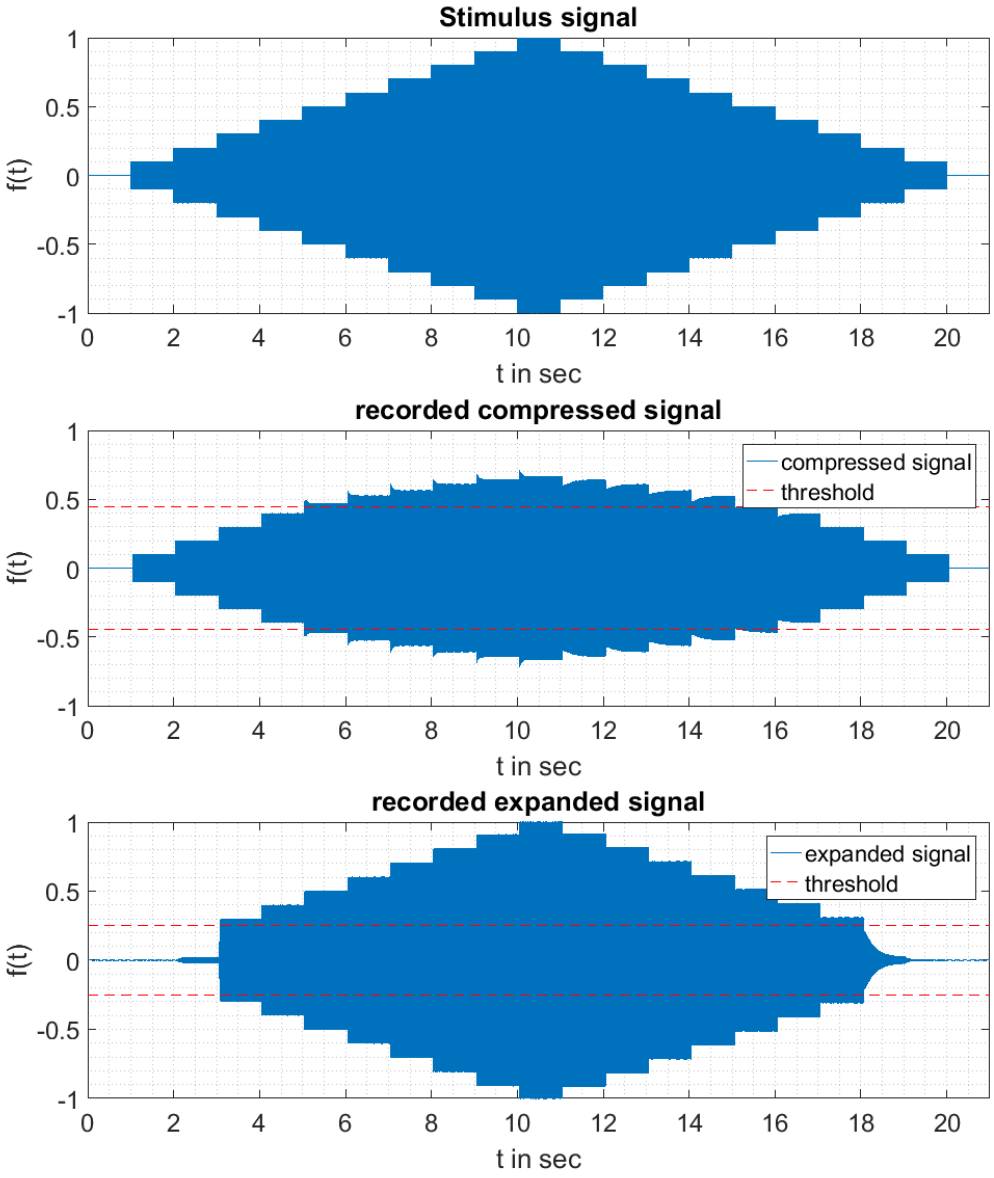


Figure C.19.: *Compander measurement*

Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM 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, August 29, 2017	
Ort, Datum	Unterschrift