3 Band Automatic Real-time Equalizer for Audio Amplifier Signal Correction

Diplom Rapport

Dennis Schmidt Nielsen

Electronic Engineering and IT (Aalborg)

AALBORG UNIVERSITY
DENMARK

Aalborg University

Titel:

3 Band Automatic Real-time Equalizer for Audio Amplifier Signal Correction

Projekt periode:

Februar 10st 2014 - Juni 4th, 2014

Student:

Dennis Schmidt Nielsen

Vejleder:

Søren Krarup Olesen

Mentor:

Bo Bjerrum

Number of pages in appendices: 5

Total number of pages: 82

Number of pages in report: 57

Abstract:

This report consists of a 7'th semester diploma project, which is the final project in the education to be an electronic diploma engineer. The assignment in this project is to optimize an tube amplifier, to get a flat audio spectrum response. The system is build on a Xilinx FPGA platform, which has an implemented Microblaze processor, which has been build up to be a 5 stage pipeline DSP. The input audio signal is the Spdif with CD quality format of 44.1Khz 16bit, and has been build from scratch in VHDL, and implemented inside the FPGA. The output, and feedback to, and from the amplifier has been build up around the Wolfson WM8731 CODEC, and all interface communication has also been build up in VHDL, such that it does not strain the Microblaze. All equalizer, and analyser filters implemented in Microblaze, has been modelled from calculated standardized octave bands.

The content in this report will first be an introduction, followed by project proposals, where an analyse and boundary will be described. Then two chapters, one for hardware solution, and one for signal processing solution, which both describe a deeper analyse, solution. Then a summarize part with conclusion, and perspective, then a appendix with diagrams, tests, measurements, and etc. And in the end, there will be a bibliography over sources used in this report and project.

Forwords

This report consist of the final engineer diploma project, by Dennis Schmidt Nielsen a student at Aalborg University, 7th semester from Electronic Engineering and IT(Aalborg). Conditions for reading this report is to have understanding for some basic signal processing, embedded systems, C, and VHDL programming.
I would like to make a specially thank to my supervisor Søren Krarup Olesen, and my mentor Bo Bjerrum for all guidance, constructive, and inspiring help and ideas throughout the hole process in making this project.

Reading guide:

Harvard method is in this report used as source reference. Sources are referenced by [number], and are placed under bibliography section. Books are specified by author, tittle, edition, and publisher. Internet references are specified by author, title and date.
Figures and tables are numbered by chapter, i.e. the first figure in chapter 4 is number 4.1, next 4.2 and so on. Abbreviations are used, and will appear inside parentheses just after name.

Aalborg University, 04.06-2014

Dennis Schmidt Nielsen

ACRONYMS

Aau	Aalborg University
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
VHDL	Very High speed Integrated Circuit
FPGA	Field Programmable Gate Arrays
LUT	LookUp Table
MIPS	Million Instruction Per Second
RISC	Reduced Instruction Set Computer
PLB	Processor Local Bus
LMB	Local Memory Bus
PCB	Printed Circuit Board
Spdif	Sony/Phillips Digital Interface
DSP	Digital Signal Processor
DSP	Digital Signal Processing
MAC	Multiplier Accumulator
TTL	Transistor Transistor Logic
BMC	Bi Phase Mark Code
PCM	Pulse Code Modulation
CD	Compact Disc
I2S	Integrated Interchip Sound or Inter Ic Sound
I2C	Inter Integrated Circuit
MCLK	Master Clock
LSB	Lowest Significant Byte
MSB	Most Significant Byte
LRC	Left Right Select
AUX	Auxiliary
RMS	Root Mean Square
ACK	Acknowledgement
BCLK	Bit Clock

CONTENTS

CHAPTER 1

INTRODUCTION

This report consists of Dennis Schmidt Nielsens 7th semester diploma project report. The projects project proposal is to optimize an tube amplifier output to get a flat frequency response. In chapter 2 there are the definition of the problem, which consists of a problem description with project solution proposals, and a second part describing the analyse, and choosing of solution proposal, and the the hardware platform.
In chapter 3 there is an hardware analyse, and solution part. The analyse part analyse the hardware components, and shows the hardware components function. The solution part will describe how the hardware has been developed, programet, and implemented.
In chapter 4 there is an signal processing solution part, which consists of an analyse, and solution part. In the analyse part there will be an analyse over audio theory, and other theory relevant for the chosen system. The solution part will describe how the methods has been calculated, programet, and implemented.
In chapter 5 there is a conclusion and a perspective of this project. Chapter 6 consists of a appendix, which consists of diagrams, interfaces, measurements, tests, and etc. In the end of this report there is a bibliography list over the reference sources used in this project.

CHAPTER 2

DEFINITION OF PROBLEM

2.1 Problem Description

The project proposal is to optimize a tube audio amplifier, to get a flat frequency spectrum response. When an audio signal enters an amplifier it will first go trough the preamp stage to gain the amplitude, after the preamp the signal will go trough a power stage to gain the current of the signal. A tube amplifier is normally build up of good quality components, and as few components as possible, however the components are far from ideal and flat in the frequency spectrum. To get, and match components so they compensate for each other is a extreamely expensive, and demanding task, and cost a lot of work time, and the result in the end will never be total perfect.
In this project the proposal is to compensate this by measuring the signal of the amplifier, and feed it back to some sort of Digital Signal Processing system, and analyse the difference between output and the input signal, and correct it to get a flat frequency spectrum response. The input source has to be at a high quality, because in this project it is the goal to get as close as possible to the input signal, in the best case scenario it will be very close to the sound of the recorded signal. The goal is therefore to remove the colouring of the signal added through the amplifier system, and compensate the signal by adjust an equalizer by the difference from analysing the output vs input signal.
The audio spectrum goes from approximately $20hz$ to $20khz$[1]. Below is 2 spectrum's the first is the uncorrected output signal, that shows the amplitude is different from frequency to frequency. The second spectrum shows an corrected(ideal) audio spectrum where the spectrum is flat, and has the same amplitude over all frequencies.

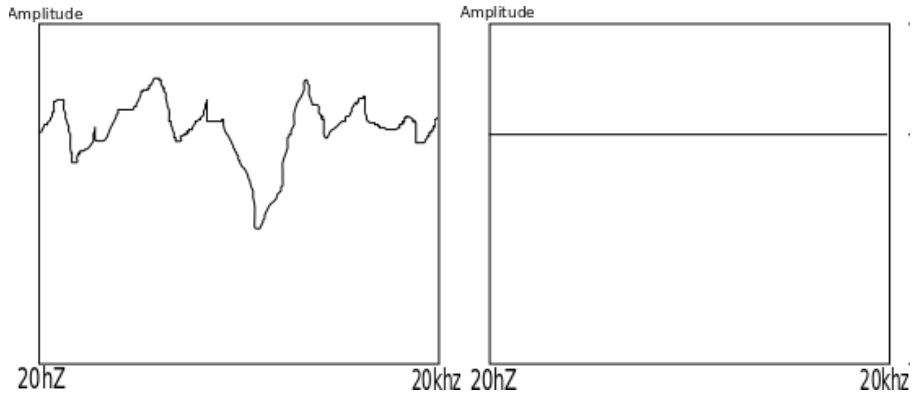

Figure 2.1: Audio spectrum over uncorrected signal vs corrected signal.

There is lined up 3 possible proposals solutions to focus on.

1)The first one is to focus only on signal correction from the output of the amplifier, and feedback the output signal to a signal processing unit, where there is implemented an analyser and an equalizer. The feedback from the output of the amplifier will go trough an analyser to measure the energy spectrum, and compare it with the input signal energy spectrum, and out from these data adjust an equalizer mounted on the signal line before output to the amplifier. Below is a sketch over a system of signal correction.

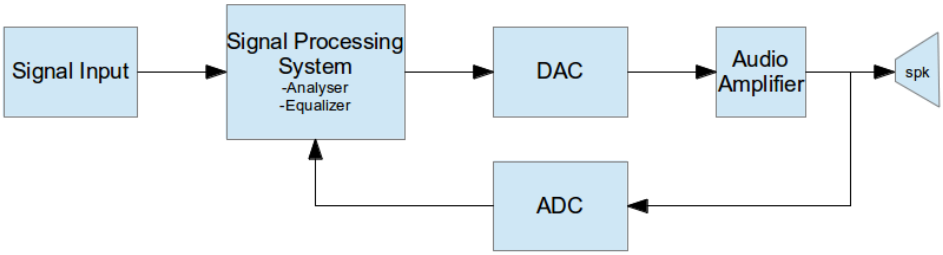

Figure 2.2: Signal correction by feedback of the output signal.

2)The second solution could be to focus on the output of the speaker(room correction). By installing a microphone where the listener of the music normally is placed(the sweet spot). The signal from the microphone will then be analysed, and thereby have the room characteristic, and from these characteristics adjust an equalizer mounted on the signal line before output to the amplifier. Below there is a sketch over how the room correction could be developed.

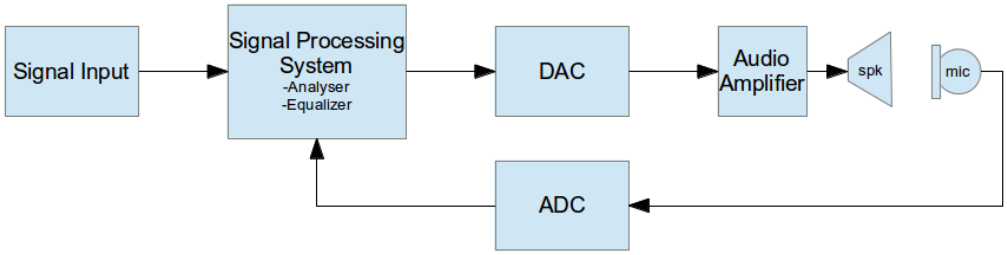

Figure 2.3: Room correction by feedback output from speaker via a microphone.

3)The third solution proposal could be a combination of signal and room correction. By first scan the room characteristic, and save these data in variables, and use these data in combination with the signal correction, and adjust in combination the equalizer mounted on the signal line before the output to the amplifier. Bellow is shown a sketch over how such a system solution could be developed.

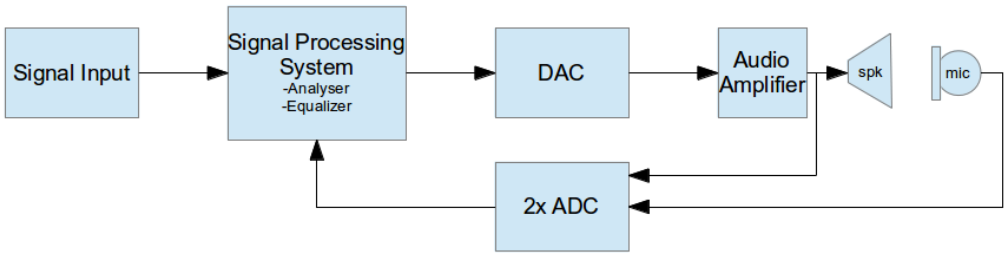

Figure 2.4: Room and signal correction by feedback of a room characteristics, and the output signal of the amplifier.

In the next section, there will be an analyse, boundary, and the goal over the system choice, and of what there has to be developed in each block.

2.2 Project Analyse, Boundary, And Goal

After some overall reflection, the solution was to go for the number 1), the signal correction of the amplifier by measure the output signal, and compare it by feedback the input signal to an analyser, and out from these data adjust an equalizer mounted on the signal line, before the output to the amplifier. The main reason to boundary to this solution are the time limitation of 3 months, given for this project. Below is number 1) sketch diagram again.

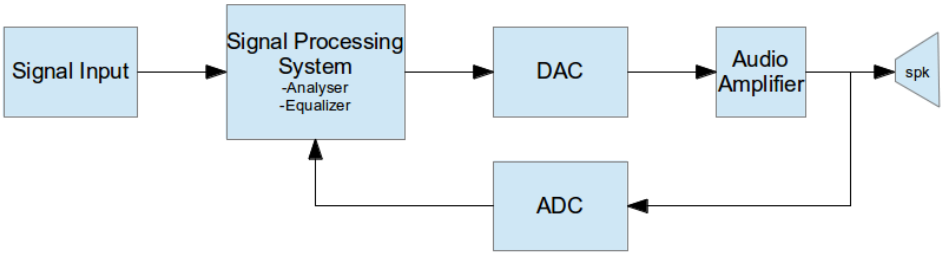

Figure 2.5: The chosen solution, Signal correction by feedback of the output from the amplifier.

So the job in this project is to find or create a platform where the analyser and equalizer can fit on. Below there is a sketch over what parts and choices there has be reflected over. The input could be either an ADC(Analogy To Digital Converter), or the digital audio interface Spdif(Sony/Philips Digital Interface Format). The Spdif is equipped in most audio systems on the market, an gives a good quality source of audio. The ADC will be an analogue input so the source quality will depend on equipment before this correction system. The platform could consist of a DSP(Digital signal Processor) or a FPGA(Field Programmable Gate Arrays). A benefit could be the FPGA, because of the support of parallel cores CPU's, and the VHDL(Very High speed Integrated Circuit) hardware implementation beside the cores. Below shows a sketch diagram over the different blocks and choices there has to be made.

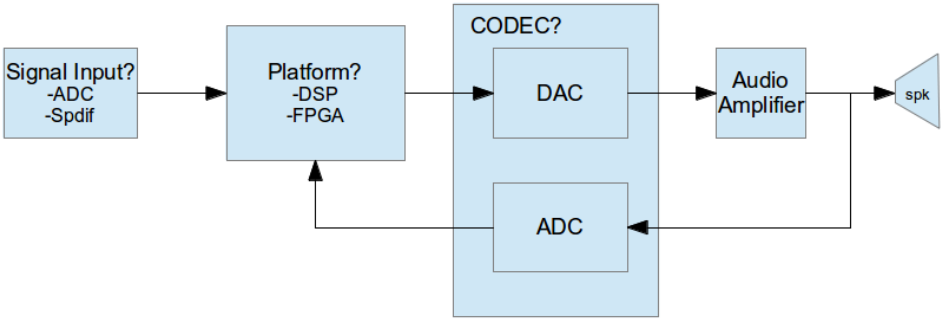

Figure 2.6: Sketch diagram over the signal correction and the choices of its parts there has to be taken.

As there can be seen on the sketch diagram there is different choices, below is the pros, cons, and the chosen goal.

Input ADC Or Spdif ?

An table is set up to compare the ADC input vs the Spdif input.

	ADC	**Spdif**
Development time	Short	Long
Noise added	There is noise added	No noise added to signal
Quality	Good	Super

Table 2.1: Compare ADC with Spdif

The choice was to go for Spdif because of the good quality and no noise is added to the signal as in the analogue ADC. In this project it is the goal to remove the colour of the audio signal not add, so the choice is therefore to go for the Spdif, because of its digital representation, and high quality.

Platform DSP Or FPGA ?

The platform to choose to implement the system on, requires a little reflection, but first a tabel se tup to compare the platforms.

	DSP	**FPGA**
Speed	Fast	Extremely Fast
Peripherals	Low	Theoretical Unlimited
Number of CPU's	1	Theoretical Unlimited
Development Time	Fast	Middle/Long
ADC/DAC's	Many	0

Table 2.2: Compare DSP vs FPGA

The choice is to go for the FPGA, especially the speed needed for the Spdif was an argument. An DSP is not fast enough to shift in a Spdif-signal(easily go over $6Mbit/s$)[2], and have power to also run the systems analysers, equalizers and other algorithms.The only advantages the DSP has over the FPGA is the smaller development time, and the included ADC and DAC's. But the ADC and DAC required will anyway be external, because of the wanted quality of the signal

ADC/DAC's or CODEC

The choice if it should be ADC and DAC's separately or a CODEC, will be open choice in this project and up to what hardware there is available on the market, for a reasonably price. But a CODEC will maybe be the best, because of the development time(shared interface), another benefit is that in a CODEC the ADC and DAC's will properly be closely matched.

The Desired System Set-up :

The desired system setup consist of a Spdif input, and a FPGA based platform to implement the system on, and ADC and DAC's or a CODEC. Below is a sketch diagram over the desired hardware set-up.

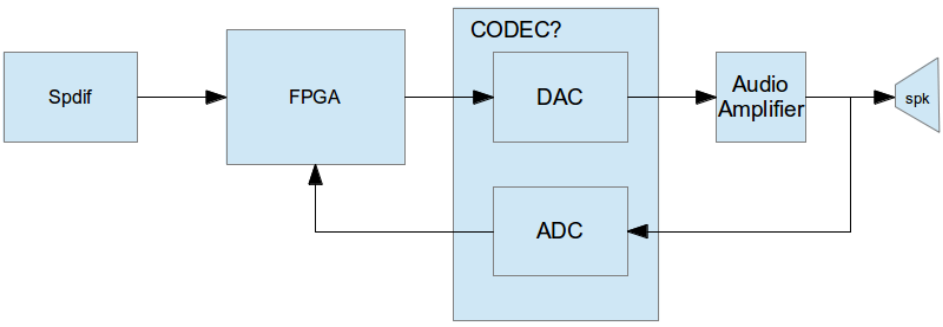

Figure 2.7: Sketch diagram over the desired system set-up.

Choice of platform :

After some researches on the internet, an interesting development board based on the Xilinx spartan6 lx9[3] FPGA showed up on ebay.com[4]. The board only consist of the FPGA system, but with a little extra fee, an expansion board with a Wolfson WM8731 CODEC[5] could be added to the ordre. This board are perfect to this project and the price is extremely low 80 dollars plus 11 dollars for ship, approximately 500 Dkr. Below shows the ordered boards.

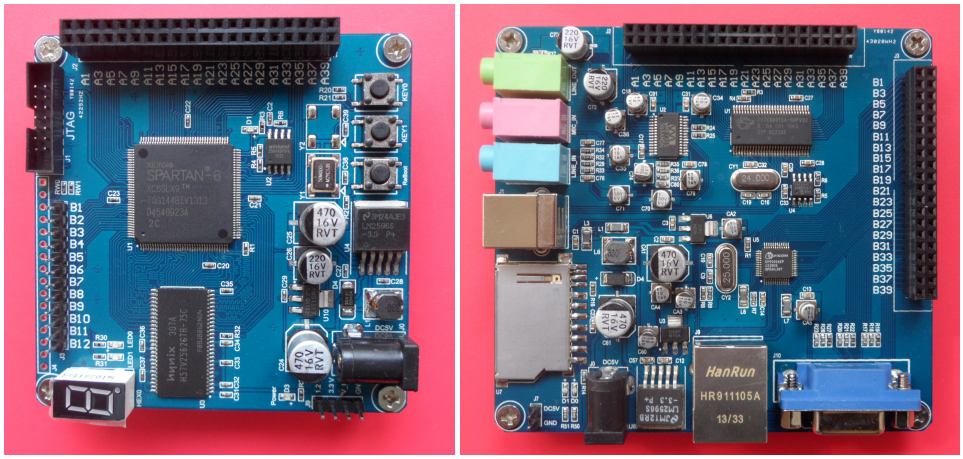

Figure 2.8: To the left the core board, and to the right the expansion board[4].

Summarize System Goal:

A little quick summarize of this project system, and what there has to be developed.

- Choice 1) signal correction only
- The system input signal will consist of Spdif(Sony/Philips Digital Interface Format)
- The platform will consist of a Xilinx Spartan6 lx9 FPGA platform
- ADC/DAC's will consist of a Wolfson WM8731 CODEC on an extension board

In the 2 next chapters, there is an deeper analyse and solution of the system. The analyse/solution part are split up in 2 separately chapters, one for the hardware, and one for the signal processing part.

CHAPTER 3

HARDWARE SOLUTION

In this chapter there is an analyse section, followed by a solution section, over the different hardware components, and how these function, and how the hardware is developed. The sketch diagram below shows the desired hardware set up, and is the outcome from the analyse part, it is shown here for a better understanding of the developed hardware from the start.

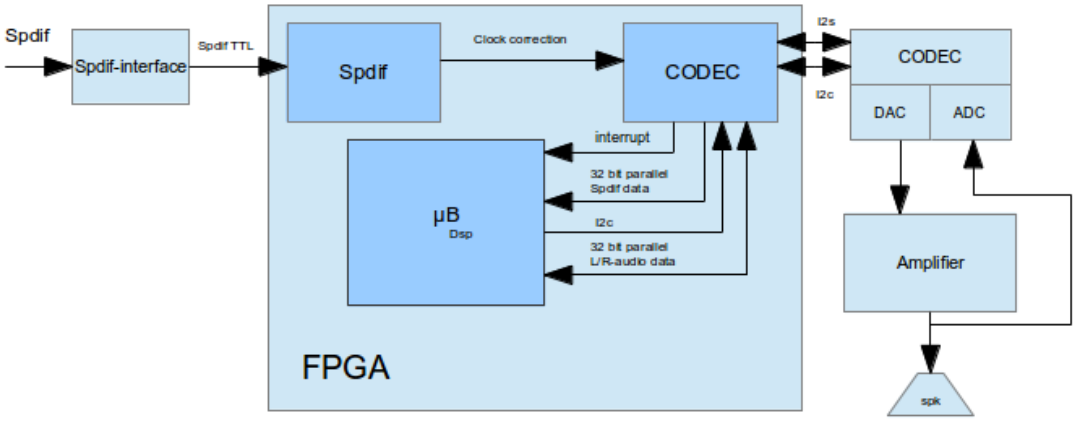

Figure 3.1: Overview Over Hardware

First will the Spdif(Sony/Phillips Digital Interface) go trough an interface, interfaced will convert the optical Spdif signal into TTL(ransistor Transistor Logic) level. The TTL Spdif signal will then enter an FPGA(Field Programmable Gate Arrays), where an hardware VHDL(Very High speed Integrated Circuit) program will take care of receiving, and split the signal up into the raw audio data. When the Spdif has been split up it will be send to the CODEC, there will take care of sending the data to the Microblaze there is set up as a DSP(Digital Signal Processor). The Spdif are controlling the system update clock cycles, which will be when a new Spdif audio sample is ready. When an update from the Spdif appears, the CODECs VHDL code will send the data from the signal processed signal, from the Microblaze to the CODEC DAC(Digital To Analog Converter) via I2S(Integrated Interchip Sound). At the same time the CODEC code will shift in the received output from the amplifier, via the ADC(Analog To Digital Converter) in the CODEC. The first time the system starts up the Microblaze will setup the registers in the CODEC by sending 2 wire I2C(Inter Integrated Circuit).

3.1 Analysing

In this section an deeper analyse of the Spdif(Sony/Phillips Digital Interface)[2], followed by Microblaze[6] set up as a DSP(Digital Signal processor), and last the CODEC WM8731[5], will be performed.

3.1.1 Spdif Protocol

Spdif stands for Sony/Phillips Digital Interface[2]. The purpose of the protocol is to interface, and transceive digital audio data. There is 3 standard interfaces[2], Coax(coaxial cable), Toslink(optical cable), and TTL. The Spdif protocol consist of 64 bit subframes, where the first 8 bita is an identification preamble, to detect whether it is the left or right audio channel, or a start of a block.

Preamble	

The preamble identification bits:

"0001.0111" or inverted signal "1110.1000" = Start of block, and is also left audio channel data.
"0001.1011" or inverted signal "1110.0100" = Left audio channel data.
"0001.1101" or inverted signal "1110.0010" = Right audio channel data.

Blocks and frames:

The protocol consists of a block, where each block consists of a group of 192 frames, each frame consists of two subframes, where one subframe is for the left(start-block/not start-block) channel, and the other is for the right channel. When the block has reached 192 frames a new block starts, this will continue infinite, or until the Spdif source has been turn off.

Frame	Sub Frame	Sub Frame
1	Start Block Left channel	Right channel
2	Left channel	Right channel
3	Left channel	Right channel
.	Left channel	Right channel
.	Left channel	Right channel
.	Left channel	Right channel
192	Left channel	Right channel

Table 3.1: Block with 192 frames, each frame consists of 2 subframes

BMC(Biphase Mark Code):

After the detection of the first 8bit subframe preamble, and which subframe channel there has been detected, then the rest of the bits(from bit 8 to bit 63) is BMC(Biphase Mark Code) encoded, which means that if two bits is "11" or "00" it is a 0, if the two bits is "10" or "01" then it is a 1:

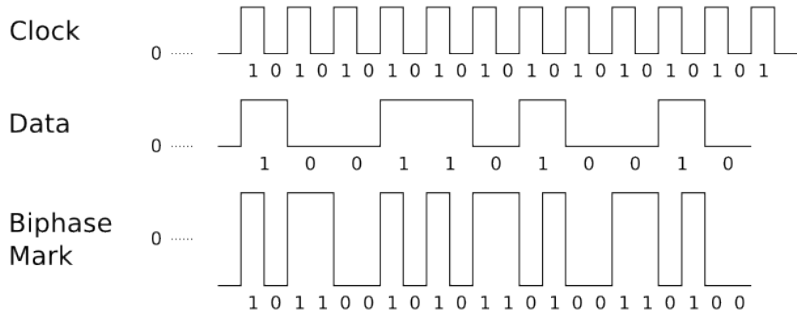

Figure 3.2: BMC(Biphase Mark Code) time diagram.[2]

Sub frame protocol:

After the BMC(Bi Phase Mark Code), the subframes is split up, from bit 4 to 7 a 4 bit AUX(auxiliary) which can consist of free usable bits, some use the 4 bits to expand the audio resolution from 20 bit to 24 bit. The next 20 bits consist of PCM(Pulse Code Modulation) audio data, with LSB first at bit 8 to MSB at bit 27. The last 4 bits from 28 to 31, consists of valid, user, control and parity check.

Preamble	AUX	LSB——20bit audio data——MSB	V	U	C	P

Table 3.2: Subframe Protocol

Where

V = Valid : Data ok to output(practical not useful because there is no time to detect errors).

U = User : Serial data stream each channel, with format specified.

C = Control = Channel status : Consists of various information such as tittle, format, etc.

P = Parity check : For detection of errors in data transmission, set to "1" if bit 4 to 30 have odd parity, and set to "0" if bit 4 to 30 have even parity.

Audio data format Normally CD(Compact Disk) audio data is send as 16bit PCM(Pulse Code Modulation) over the Spdif.

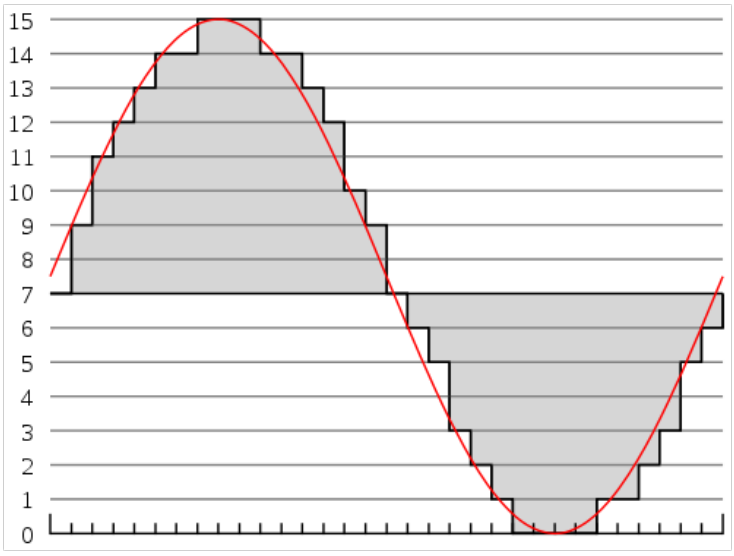

Figure 3.3: PCM(Pulse Code Modulation)[7]

This has to be taken care of if the signal have to be represented as 2's complement later.

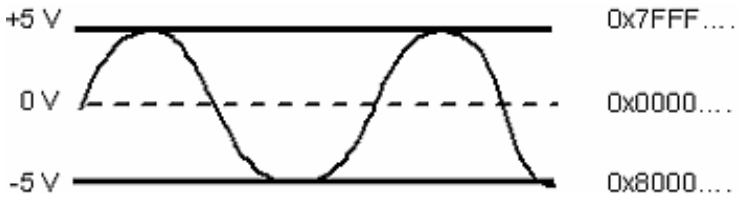

Figure 3.4: PCM(2's complement)[8]

Captured Spdif Signal:

A captured signal from a CD player used in this project, shows the sub frames with preamble and all the BMC encoded bits.

Figure 3.5: Spdif signal captured from the cdplayer coax output by an digital analyzer

As there can be seen 2 and ½ subframe is captured, the first"0001.1011" which shows that the CD-player is not inverted, and it is a not start of block, and it is the left audio channel. The next "00001.1101" shows the right audio subframe, the last is again "0001.1011" so no start of block and it is the left audio channel data.

3.1.2 Microblaze

Microblaze was chosen because of earlier experiences with the processor. Microblaze[6] is a 32bit RISC processor, and is by default set up to 3 stage pipeline, but support 5 stage pipeline. There is support of 64 bit MAC(multiplication accumulator). Microblaze architecture of the memory is Harward, and address space is 32 bit and can handle up to 4Gb instructions and data. I/O's has a mapped address from "0xC0000000" to "0xFFFFFFFF" in the memory space. The local memory stores the data and program instructions and the size varies between 8kb and 64kb and is chosen when building the core.

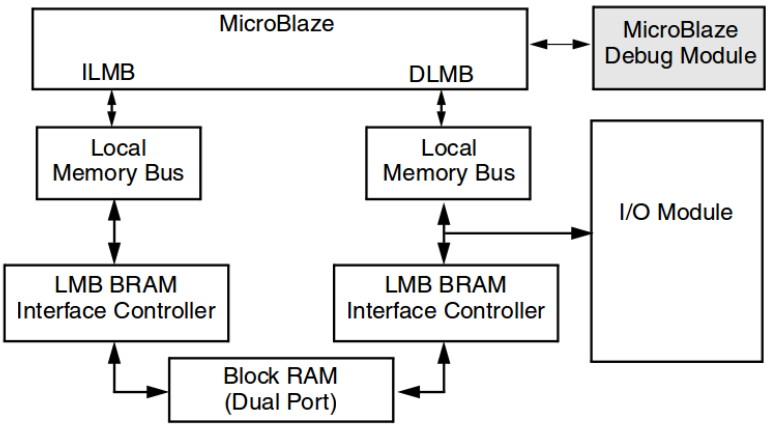

Figure 3.6: Overview over the Microblaze.[6]

Microblaze consumes 536 LUTS and 276 to 1619 flipflops without any pheripherals in the FPGA, and is optimized for being implemented on Xilinx FPGA's.

3.1.3 CODEC WM8731

The Wolfson WM8731 CODEC[5] is a stero CODEC, with 2xDAC, 2xADC, and a microphone mono input. The sampling frequency is between $8khz$ and $96khz$. It's audio stream interface is based on the I2S(Integrated Interchip Sound, or Inter IC Sound) developed by Philips[9] to interconnect audio digital stream between chips. The CODEC has selectable inputs of 16, 20, 24, and 32 bits word-length's(2's complement), and the device can be selected to operate in either slave or master mode. All the initialize of the peripherals set up in the CODEC, is interfaced by the 2 wire I2C(Inter-Integrated Circuit)[10] originally developed by Philips, to integrate communication between chips over a 2 wire line. Below is an overview over the WM8713 CODEC

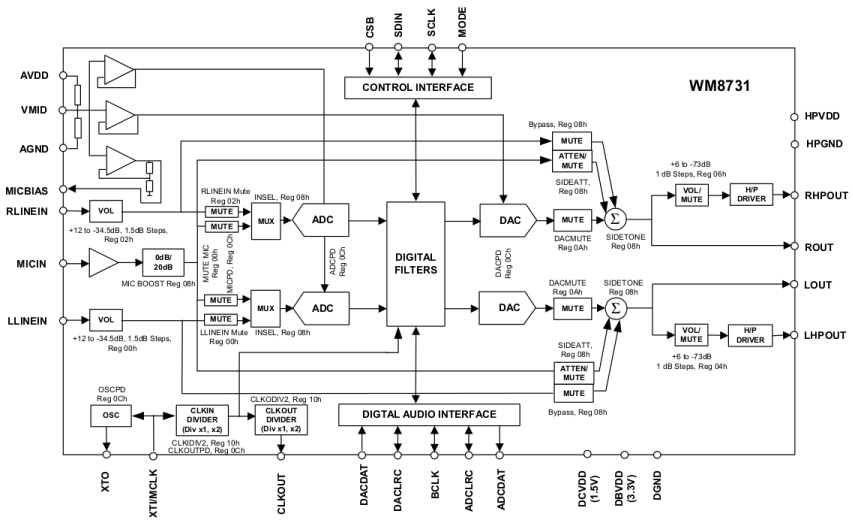

Figure 3.7: Overview over the Wolfson WM8731 CODEC[5].

Peripherals setup I2C:

Before using the CODEC, it's peripherals has to be initialized via the 2 wire I2C interface. I2C has 2 wires one for clock(between $0-400khz$), and the other for data.

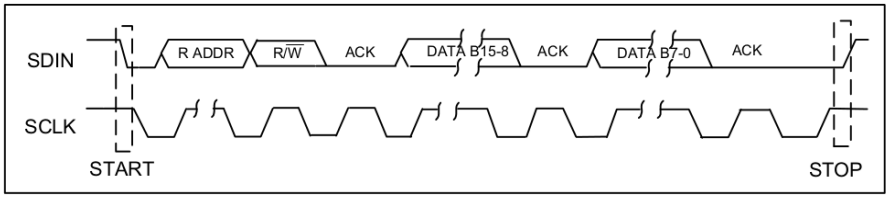

Figure 3.8: Overview over the I2C 2 wire[5].

The data sequence is shifted in with the I2C's clock one per bit, the sequence is first an address(WM8713: "0011010" or "0011011") consisting of 7 bit and a R/W-bit, the CODEC will

then respond by an ACK(acknowledgement). When an ACK has been send back, the data to the device can be send each packet is of size 8 bit every time the CODEC will respond with an ACK.

WM8731 CODEC registers:

WM8731 has a various of setup modes, below is a table with an overview of its different registers.

REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT		SIDETONE	DAC SEL	BYPASS	INSEL	MUTE MIC	MIC BOOST
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/ NORM

Figure 3.9: Overview over the WM8731 CODEC[5].

Every register has a 7bit register address, and 9 bit of register set up bits. The sequence has to be combined to 2x8bit packets to send over I2C, where the first 7bits is the register address, and the following 9bits is the register bits. In solution part there is specification, and set up of the registers.

I2S audio stream:

When the CODEC has been initialized, a MCLK(Master Clock) has to be added to the CODEC, this is for the cycle of the input stream of the I2S audio. There is a various of frequencies for the MCLK(see solution part). When the MCLK is added, the CODEC is ready to receive and transmit the audio stream via I2S. There is basically 2 modes, in the I2S, I2S-mode and DSP-mode.

I2S-mode:

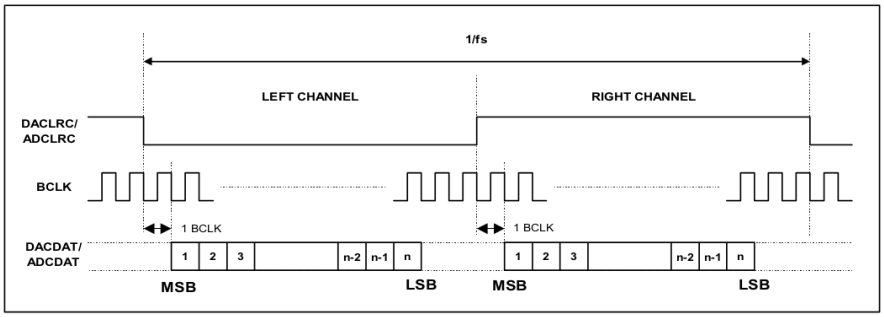

Figure 3.10: Overview over the I2S-mode[5].

The I2S I2S-mode consists of 3 signals, DAC/ADC-LRC(Left Right Select), BCLK, and DAC/ADC-data. LRC controls the left/right channel select, BLCK is the bit-stream clock one clock per bit, and the data is the audio data. When sending data, the data starts at the second clock of BCLK. The data is then sended in first with the left channel data with MSB first, then LRC change stage, and the right channel data is send.
DSP-mode:

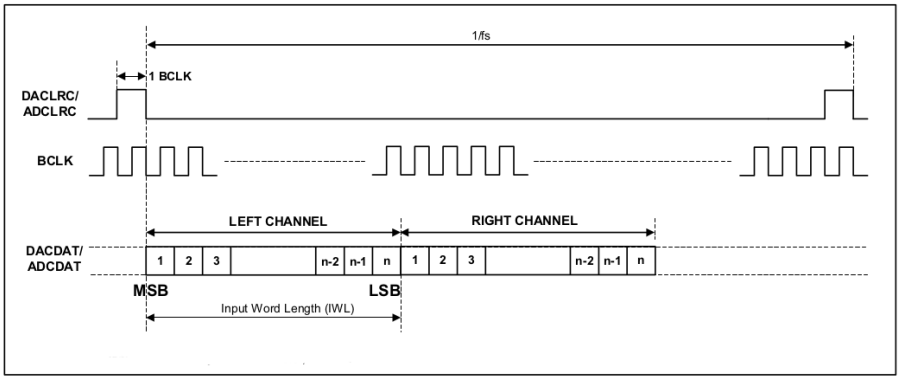

Figure 3.11: Overview over the DSP-mode[5].

In DSP-mode the LRC is a short pulse with the time of 1 BCLK(Bit Clock). The audio data stream is send with the left concatenation with the right. The first bit is the MSB and has to fall on the first BCLK after the pulse. After the data has been shifted in, the codec will process the data, and wait for a new pulse on LRC, all data send between the data length and the pulse will be ignored.

3.2 Solution

In this section the solution of and how Spdif, Microblaze, and CODEC is implemented. First there is Spdif, then Microblaze, and last CODEC.

3.2.1 Spdif

The clock frequency for the Spdif[2], has to be one clock cycle per bit of the Spdif frame. One frame consist of 2 subframes where each consists of 64 bit so $2 \cdot 64 = 128$. In this project a normal CD(Compact Disk) is chosen to be the Spdif input. A normal CD has the sampling frequency of $44.1khz$ per channel so $44.1khz \cdot 2 = 88.2khz$, and the format is 16 bit PCM(Pulse Code Modulation). So the clock frequency for the Spdif has to be:

$$64bit \cdot 2_{channels} \cdot 44.1 \cdot 10^3 = 5.644800Mhz \tag{3.1}$$

By looking at values of crystals oscillators there is available at the market, there was no $5.644800Mhz$, so an $16.934400Mhz(3 \cdot 5.644800Mhz = 16.934400Mhz)$ was chosen instead because it could be divided up by 3:

$$\frac{16.934400Mhz}{3} = 5.644800Mhz \tag{3.2}$$

The VHDL code for dividing the clock:

```
.  VHDL for the clock divider:
.
.  if rising_edge(Mclk) then
.     clk_div <= clk_div + 1;
.        if clk_div = 0 then
.           clk <= '1';
.        elsif clk_div = 1 then
.           clk <= '1';
.        elsif clk_div2 = 2 then
.           clk <= '0';
.           clk_div <= "0000";
.        end if;
.  end if;
```

The Spdif bit-stream is shifted in, in a 128 bit buffer, so there can be 2 subframes in it:

```
.   VHDL for shifting the Spdif bit-stream in:
.
.    if rising_edge(clk) then
.       Spdif_Frame <= Spdif_Frame(126 downto 0) & Spdif_bitStream;
.   end if;
```

Every time the frame is shifted one place in the Spdif_Frame buffer, the hole buffer is checked if there is preambles on place 127 down to 120(for the left) and place 63 down to 56. The VHDL code looks:

```
.   if rising_edge(clk) then
.       CLOCK_correction <= '0';
.          if (Spdif_Frame(127 downto 120) = "00011101" OR Spdif_Frame(127 downto 120) =
.           "00010111") AND (Spdif_Frame(63 downto 56) = "00011011") then
--left channel subframe
.              Gneric_GPIO_GPIO_IO_pin_latchL(0) <= Spdif_Frame(73) XOR Spdif_Frame(72);
.              Generic_GPIO_GPIO_IO_pin_latchL(1) <= Spdif_Frame(75) XOR Spdif_Frame(74);
.                                                    .
.                                                    .
.                                                    .
.            Generic_GPIO_GPIO_IO_pin_latchL(15) <= Spdif_Frame(103) XOR Spdif_Frame(102);
--Right channel subframe.
.              Generic_GPIO_GPIO_IO_pin_latchR(0) <= Spdif_Frame(9) XOR Spdif_Frame(8);
.              Generic_GPIO_GPIO_IO_pin_latchR(1) <= Spdif_Frame(11) XOR Spdif_Frame(10);
.                                                    .
.                                                    .
.                                                    .
.              Generic_GPIO_GPIO_IO_pin_latchR(15) <= Spdif_Frame(39) XOR Spdif_Frame(38);
--Convert to 2's complement and shift the 2 subframes in the parallelport of microblaze
.              SPDIF_GPIO <= ((not Generic_GPIO_GPIO_IO_pin_latchL) + 1) &
.               ((not Generic_GPIO_GPIO_IO_pin_latchR) + 1);
--Setclockcorectionto1toactivateintteruptinCODECfunction
.          CLOCK_correction <= '1';
.          end if;
.   end if;
```

After this every thing should be okay, but problems with jitter does that the the two clocks in this asynchronous serial receiving is not the same, and the result is a really bad sound with lots of clicks and noise.

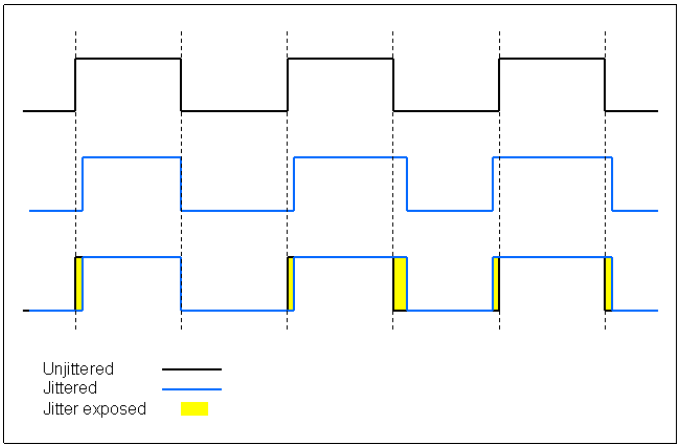

Figure 3.12: Jitter in a signal[11].

In this project the solution was, not perfect but acceptable, when using good matched clocks, to shift in the Spdif bit-stream 3 times by a clock phase of 1/3 of each other, and luckily the 16.934400Mhz was perfect. here is a sketch over the clocks:

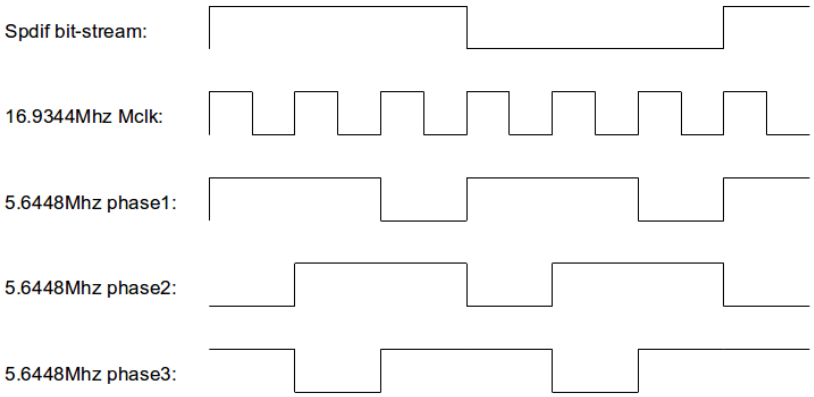

Figure 3.13: The 3 clocks with phases for the Spdif shift in

The VHDL code for the 3 clocks by a phase of 1/3:

```
.   VHDL for the 3 phases clocks:
.
.   if rising_edge(Mclk) then
.       clk_div <= clk_div + 1;
.          if clk_div = 0 then
.             clk_phase1 <= '1';
.             clk_phase2 <= '0';
.             clk_phase3 <= '1';
.          elsif clk_div = 1 then
.             clk_phase1 <= '1';
.             clk_phase2 <= '1';
.             clk_phase3 <= '0';
.          elsif clk_div2 = 2 then
.             clk_phase1 <= '0';
.             clk_phase2 <= '1';
.             clk_phase3 <= '1';
.             clk_div <= "0000";
.          end if;
.   end if;
```

The VHDL code for the 3 times buffer spdif bit-stream shift in is then:

```
.   VHDL for shifting 3 times the Spdif bit-stream in:
.
.    if rising_edge(clk_phase1) then
.       Spdif_Frame1 <= Spdif_Frame1(126 downto 0) & Spdif_bitStream;
.   end if;

.    if rising_edge(clk_phase2) then
.       Spdif_Frame2 <= Spdif_Frame2(126 downto 0) & Spdif_bitStream;
.   end if;

.    if rising_edge(clk_phase3) then
.       Spdif_Frame3 <= Spdif_Frame3(126 downto 0) & Spdif_bitStream;
.   end if;
```

The VHDL code has now the job to check 3 buffers for preambles, and chose one, and skip they others. The VHDL code below shows the how this is implemented, most of the code is included because of understanding:

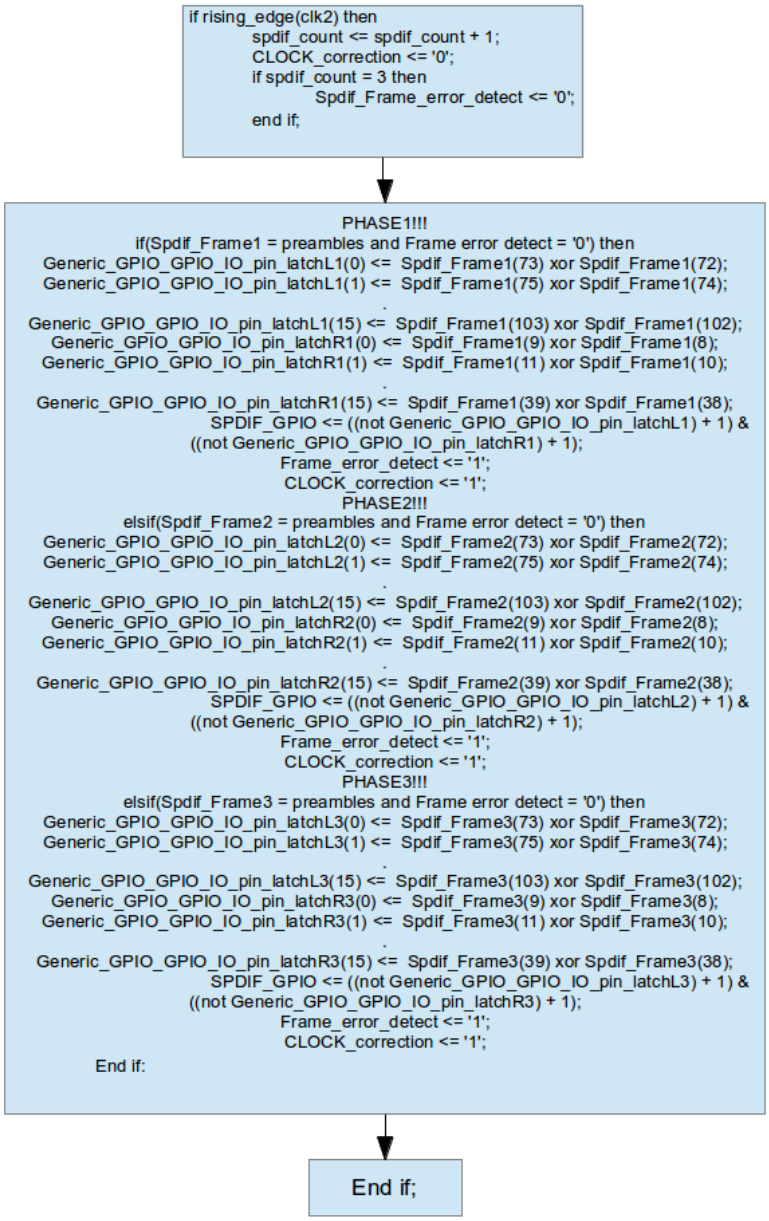

Figure 3.14: The 3-phase detecting preamble code

Point description of the VHDL code of the 3 phase preamble detecting:.

- $Spdif\_count$ is increasing by one on each clk
- $CLOCK\_correction$ is set to zero(will only be one when updating CODEC)
- When $Spdif\_count$ reach 3 the $Spdif\_Frame\_error\_detect$ is set to zero. It is for detecting when one buffer has been detected, and used for ignore other detects
- There is 3 detectors, one for each buffer. When one of them has detect a preamble it will set the $Spdif\_Frame\_error\_detect$ to high so they others will not execute. In each detector it will do the following
  - Sort out and inverse BMC(Bi Phase Mark Code) the specific buffer, and convert it to 2s complement, and store it in $Generic\_GPIO\_GPIO\_IO\_pin\_latchL$(which is the signal port to the CODEC)
  - Sort out and inverse BMC the specific buffer, and convert it to 2s complement, and store it in $Generic\_GPIO\_GPIO\_IO\_pin\_latchR$(which is the signal port to the CODEC)
  - $Frame\_error\_detect$ is equal to one
  - $CLOCK\_correction$ is equal to one(to tell the codec to update)

This VHDL code and all its functions will be running on every clock event, this shows the main reason why a FPGA(Field Programmable Gate Arrays) is chosen in this project, the rate this VHDL program has to run is 5644800 times a second, and beside that it has to switch in 3 times 5.644800 Mbit/s = 16.934400Mbit/s Spdif signal, and divide and route all the clocks, run the codec, etc.
A DSP(Digital Signal Processor) will not have this sort of power, and also have power to do all the calculations of the filters and other algorithms.

3.2.2 Microblaze

Microblaze[6] is build up to support the following peripherals.

- $System\ clk\ is\ set\ to\ maximum\ of$ 86.333333Mhz
- 32$kb\ instructions/data\ local\ memory$
- 32$bit\ GPIO(out)\ for\ output\ to\ DAC$
- 32$bit\ GPIO(in)\ for\ input\ of\ ADC$
- 32$bit\ GPIO(in)\ for\ input\ of\ Spdif$
- 1$bit\ GPIO(in)\ input\ for\ interrupt$
- 1 $UART\ to\ debug,\ etc.$.
- 1 $iic(I2C)\ for\ intializing\ registers\ in\ CODEC$

Beside the above peripherals, the processor is set up to be a 5 stage pipeline and 64 bit MAC, and is set to floating point support.

3.2.3 CODEC WM8731

The CODEC WM8713[5] is coded by shifting I2S[9] audio data stream in and out at the same time, by using the DSP-mode, and is set in slave-mode. WM8713 has been added the $MCLK = 16.9344Mhz$. It's interface is programmed in VHDL, so it runs parallel with the Spdif and the Microblaze[6] processor, thereby not take resources from the processor. $BCLK = 5.6448Mhz$ this is the bit-rate of the I2S audio data bit-stream.
"$CLOCK\_correction$" from the Spdif is controlling the LRC witch determine when a new audio samples has to be written, and a new one shifted in the CODEC(ADC from the amplifier output). The code below followed by step by step trough the code, explain how the CODEC interface are programmed:

```
.VHDL for I2S interface
.
.   if falling_edge(BCLK) then
.       DAC_count <= DAC_count + 1;
.       DACLRC <= '0';
.       ADCLRC <= '0';
.       DAC_out <= DAC_DATA(31);
.       DAC_DATA <= DAC_DATA(30 downto 0) & '0';
.       ADC_DATA <= ADC_DATA(30 downto 0) & ADC_in;
.
.       if (DAC_count = 1) then
.           Mb_interrupt_controller <= "0";
.       elsif (DAC_count = 33) then
.           ADC_DATA1 <= ADC_DATA;
.           ADC_GPIO <= ADC_DATA1;
.       elsif (CLOCK_correction = '1') then
.           Mb_interrupt_controller <= "1";
.           DAC_DATA <= DAC_GPIO;
.           DACLRC <= '1';
.           ADCLRC <= '1';
.           DAC_count <= "0000000000000000";
.       end if;
.    end if;
```

Steps for the code is:

- BCLK: bit-stream clk of 5.6448Mhz
- DAC_count: increases by one every BCLK $'event'$
- DACLRC/ADCLRC: is set to zero when no LRC pulse
- DAC_out(31): is the bit at the given time there is send to the CODEC DAC I2S
- DAC_DATA: is shifted one place
- ADC_DATA: is being shifted in by the bit at the given time, from the CODEC I2S ADC
- if DAC_count=1: Microblaze interrupt is set to zero
- if DAC_count=33: ADC_DATA is stored inADC_GPIO in the Microblaze ADC data port(32bit parallel data port)
- if CLOCK_correction=1: Comes from the Spdif and tell that the system has to update
- Mb_intterupt_controller: is set to one to generate interrupt in Microblaze
- DAC_DATA <= DAC_GPIO: The processed data(Signal processing) from Microblaze DAC_-GPIO is stored in DAC_DATA
- DACLRC/ADCLRC is set to one to generate the pulse in DSP-mode
- DAC_count: is set to zero and the process start again

A screen capture of the analyser shows that the ADC_DATA, DAC_DATA, DAC_LRC and ADC_-LRC works. The BCLK looks like running faster than the MCLK this is not the case, and this is due to the analyser is set bellow the nyquist rate for the MCLK.

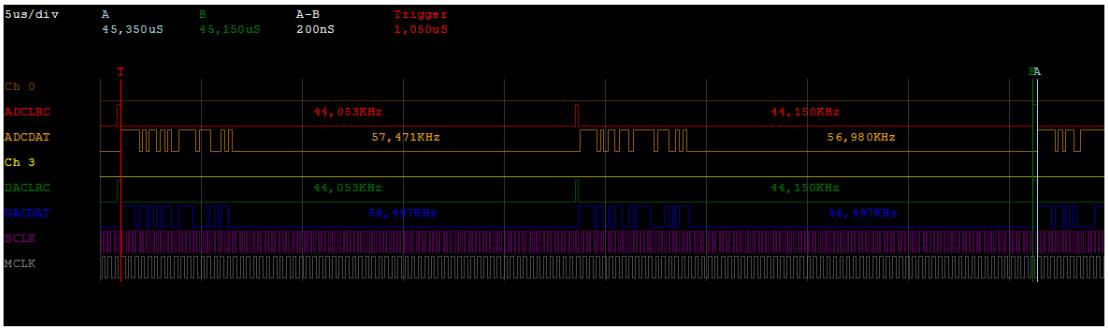

Figure 3.15: Overview Over I2S communication to and from CODEC

I2C setup:

A look again of the registers inside the CODEC, shows the functionality of the CODEC, and what has to be set in its registers.

REGISTER	B 15	B 14	B 13	B 12	B 11	B 10	B 9	B8	B7	B6	B5	B4	B3	B2	B1	B0
R0 (00h)	0	0	0	0	0	0	0	LRIN BOTH	LIN MUTE	0	0	LINVOL				
R1 (02h)	0	0	0	0	0	0	1	RLIN BOTH	RIN MUTE	0	0	RINVOL				
R2 (04h)	0	0	0	0	0	1	0	LRHP BOTH	LZCEN	LHPVOL						
R3 (06h)	0	0	0	0	0	1	1	RLHP BOTH	RZCEN	RHPVOL						
R4 (08h)	0	0	0	0	1	0	0	0	SIDEATT		SIDETONE	DAC SEL	BY PASS	INSEL	MUTE MIC	MIC BOOST
R5 (0Ah)	0	0	0	0	1	0	1	0	0	0	0	HPOR	DAC MU	DEEMPH		ADC HPD
R6 (0Ch)	0	0	0	0	1	1	0	0	PWR OFF	CLK OUTPD	OSCPD	OUTPD	DACPD	ADCPD	MICPD	LINEINPD
R7 (0Eh)	0	0	0	0	1	1	1	0	BCLK INV	MS	LR SWAP	LRP	IWL		FORMAT	
R8 (10h)	0	0	0	1	0	0	0	0	CLKO DIV2	CLKI DIV2	SR				BOSR	USB/ NORM

Figure 3.16: Overview over the WM8731 CODEC[5].

Basically in this project the overall set up of the CODEC has to be set to MCLK=16.934400Mhz, sampling rate to 44.1Khz for both the ADC and DAC, and the word length to 16bit(Compact Disk), and no microphone. The I2S interface has to be set in DSP-mode and in slave-mode.
A lot of other stuff has to be set up in the registers, below there is a table which explains how the registers is set, and how the data is send over I2C. After the table a point explanation has been set up to se the set up in deeper explanation.

Register Address	8-bit Address	Reg-data(9-bit)	I2C(2x8-bit)	Description
R0(R1):	0x00	0x11f	0x01,0x1f	Line-in (both L/R)
R2(R3):	0x04	0x17f	0x05,0x7f	Line-out (both L/R)
R4:	0x08	0x016	0x08,0x16	Analog Audio Path Control
R5:	0x0a	0x000	0x0a,0x00	Digital Audio Path Control
R6:	0x0c	0x062	0x0c,0x62	Power Down Control
R7:	0x0e	0x003	0x0e,0x03	Digital Interface Format
R8:	0x10	0x022	0x10,0x22	Sampling Control
R9:	0x12	0x001	0x12,0x01	Active Interface
R15:	0x1e	Not Set	Not Set	Reset All

Table 3.3: CODEC Register Setup table.

Where

- R1 set to: "1.0000.1111" LRIN BOTH=1, LIN MUTE=0, N/A=0, N/A=0, LINVOL="11111"(ful vol): 0x11f ={0x01,0x1f}
- R2 set to: "1.0111.1111" LROUT BOTH=1, LZCEN(zero cross detect)=0, LOUTVOL="1111111"(ful vol): 0x57f ={0x05,0x7f}
- R4 set to: "0.0001.0110" N/A=0, SIDEATT=00, SIDE TONE=0, DACSEL=1, BYPASS=0, IN-SEL=1, MUTE MIC=1, MIC BOOST=0: 0x816 ={0x08,0x16}
- R5 set to: "0.0000.0000", N/A, N/A, N/A, N/A, N/A, HPOR=0, DACMU=0, DEEMP=00, ADCHPD=0: 0xa00 ={0x0a,0x00}
- R6 set to: "0.0110.0010", N/A, PWROFF=0, CLKOUTPD=1, OSCPD=1, OUTPD=0, DACPD=0, ADCPD=0, MICPD=1, LINEINPD=0: 0xc62={0x0c,0x62}
- R7 set to: "0.0000.0011", N/A, BCLKINV=0, MS(master/slave)=0, LRSWAP(right chan data right)=0, LRP(MSB available 1'st clk)=0, IWL(input data length (16bit))=00, Format(DSP-mode)=11: 0xe03 ={0x0e,0x03}
- R8 set to: "0.0010.0010", N/A, CLKOUTDIV=0, CLKDIV2=0, SR[3:0](ADC and DAC set 44.1khz)="1000", BOSR(mclk=16.9344Mhz)=1, USB/NORMAL=0: 0x1022={0x10,0x22}
- R9 set to: "0.0000.0001", N/A, N/A, N/A, N/A, N/A, N/A, N/A, N/A, ACTIVE=1: 0x1201={0x12,0x01}
- R15 set to: "not do anything" if "0.0000.0000" then registers resets to default

The I2C is send from the Microblaze processor, and is only needed to be send once, because the CODEC in this project project has no need to change functionality under running.

CHAPTER 4

SIGNAL PROCESSING SOLUTION

In this chapter there is an analyse, and a solution part of the signal processing, and audio theory. There will be going trough how, and why, the different components has been chosen, and developed. This chapter is build up of 2 sections, an analyse part, and a solution part. In the analyse part there is focus a lot of the audio spectrum, and effects. The solution part focus on the calculations, and implementations of the methods based on the analyse part.

4.1 Analysing

In this section there is a analyse of audio spectrum methods, and theory used in this project. First there will be some analyse of the audio spectrum and how these can be divided up into octave bands. Then a short description of effects of equalization different bands. Then there will be a short description of the mean energy formula used to calculate the mean energy in different bands.

4.1.1 Octaves And the Audio Band

The audio band spectrum goes from approximately 20Hz to 20Khz[1]. The spectrum can then be divided into 11 octave bands. The octaves has been refereed to the "basic miracle of music"[12]. The use of dividing the audio spectrum up into octaves is usually used in audio systems, and may came from the harmonic series between an octave start and stop attenuation. First the 7^{th} octave is set to the ISO standard[13] $f_{ctr_7} = 1Khz$. Then all lower and higher centre frequencies is calculated by:

lower:

$$f_{ctr_{n-1}} = \frac{f_{ctr_n}}{2} \tag{4.1}$$

higher:

$$f_{ctr_{n+1}} = 2 \cdot f_{ctr_n} \tag{4.2}$$

By calculating f_{ctr_7} = 1Khz the result of the lower f_{ctr_6} and the higher f_{ctr_8} is:

Lower f_{ctr_6}:

$$f_{ctr_{n-1}} = \frac{f_{ctr_n}}{2} \Longrightarrow f_{ctr_{7-1}} = \frac{f_{ctr_7}}{2} \Longrightarrow f_{ctr_6} = \frac{1Khz}{2} = 500hz \tag{4.3}$$

Higher f_{ctr_8}:

$$f_{ctr_{n+1}} = 2 \cdot f_{ctr_n} \Longrightarrow f_{ctr_{7+1}} = 2 \cdot f_{ctr_7} \Longrightarrow f_{ctr_8} = 2 \cdot 1Khz = 2Khz \tag{4.4}$$

By continuing calculating all the centre frequencies the $f_{ctr_{1\rightarrow 11}}$ centre frequencies becomes:

15.62hz, 31.25hz, 62.50hz, 125hz, 250hz, 500hz, 1Khz, 2Khz, 4Khz, 8Khz, 16Khz

Then the n^{th} octave attenuation frequencies f_{a1_n}, and f_{a2_n} is the half octave below f_{ctr_n} for the low f_{a1_n}, and a half octave above f_{ctr_n} for the high f_{a2_n} is:

Lower f_{a1_n}:

$$f_{a1_n} = \frac{f_{ctr_n}}{2^{1/2}} \tag{4.5}$$

Higher f_{a2_n}:

$$f_{a2_n} = 2^{1/2} \cdot f_{ctr_n} \tag{4.6}$$

By calculating the attenuations for the f_{ctr_7} become:

Lower f_{a1_7}:

$$f_{a1_n} = \frac{f_{ctr_n}}{2^{1/2}} \Longrightarrow f_{a1_7} = \frac{f_{ctr_7}}{2^{1/2}} \Longrightarrow f_{a1_7} = \frac{1Khz}{2^{1/2}} = 707.103hz \tag{4.7}$$

Higher f_{a2_7}:

$$f_{a2_n} = 2^{1/2} \cdot f_{ctr_n} \Longrightarrow f_{a2_7} = 2^{1/2} \cdot f_{ctr_7} \Longrightarrow f_{a2_7} = 2^{1/2} \cdot 1Khz = 1414.210hz \tag{4.8}$$

By calculating all the attenuations become:

Lower $f_{a1_{1\rightarrow 11}}$:

11.049, 22.0971, 44.1942, 88.3883, 176.777, 353.553, 707.107, 1414.21, 2828.43, 5656.85, 11313.7

Higher $f_{a2_{1\rightarrow 11}}$:

22.097, 41.1942, 88.3883, 176.777, 353.553, 707.107, 1414.21, 2828.43, 5656.85, 11313.7, 22627.4

Now all the centre and attenuations frequencies are calculated, they can be substituted in a table:

$AudioOctave_{band_n}$	f_{a1_n}	f_{ctr_n}	f_{a2_n}
1	11.049	15.625	22.097
2	22.0971	31.25	41.1942
3	44.1942	62.5	88.3883
4	88.3883	125	176.777
5	176.777	250	353.553
6	353.553	500	707.107
7	707.107	1Khz	1414.21
8	1414.21	2Khz	2828.43
9	2828.43	4Khz	5656.85
10	5656.85	8Khz	11313.7
11	11313.7	16Khz	22627.4

Table 4.1: This table shows the calculated octave audio spectrum.

4.1.2 Major Octave Bands Effects:

The audio spectrum octaves can be broken down into 6 major ranges[14], each of the 6 bands can have a big impact on the sound. Below is a point explanation of each.

- Sub-Bass($16hz-60hz$): Gives sound a feeling of power, to much boost will make sound muddy
- Bass($60hz-250hz$): Basic rythm in the sound, can make the sound thin or fat. To much boost will make sound boomy
- Low-Mids($250hz-2Khz$): Boosting $500hz-1Khz$ makes instruments sound like horn, boosting $1Khz-2Khz$ will make sound tiny
- High-Mids($2Khz-4Khz$): Boosting make lisping quality of voices like v, m, and v, to much boost makes sound fatigue, and puts instruments in background
- Presence($4Khz-6Khz$): The clarity and definition of instruments, and voices. Boosting can feel like the sound comes closer, reducing $5Khz$ makes sound more transparent and distant
- Brilliance($6Khz-16Khz$): Boosting this area creates clarity and brilliance, to much creates whistles in vocals

4.1.3 Mean Energy

The energy in a signal has been chosen to see what difference there is in the filters. The formula for energy is:

$$E = \sum |x[n]|^2 \implies E[n] = x[n]^2 \tag{4.9}$$

By using the formula of mean from statistical:

$$\bar{x} = \frac{1}{N}\sum x[n] \tag{4.10}$$

By substitute the two together, and the update of mean is selected to be every sample, then N=2, gets:

$$\bar{E}[n] = \frac{\bar{E}[n-1] + E[n]}{N} = \underline{\underline{\frac{\bar{E}[n-1] + x[n]^2}{2}}} \tag{4.11}$$

4.2 Solution

This section consist of the solution of the digital signal processing part. First there will be an overview over the system, next is the chosen frequencies for the filter from the data calculated from the octaves part. Then there will be going trough the filters calculation, and simulations. After this a description over a small algorithm program made to adjust the filters, to get the maximum performance. In the last part of this section, the implementations of the filters and other algorithm in the Microblaze.

4.2.1 Overview Over DSP Solution

After a lot of reflection, and sketching with a pen, the desired signal processing system looks.

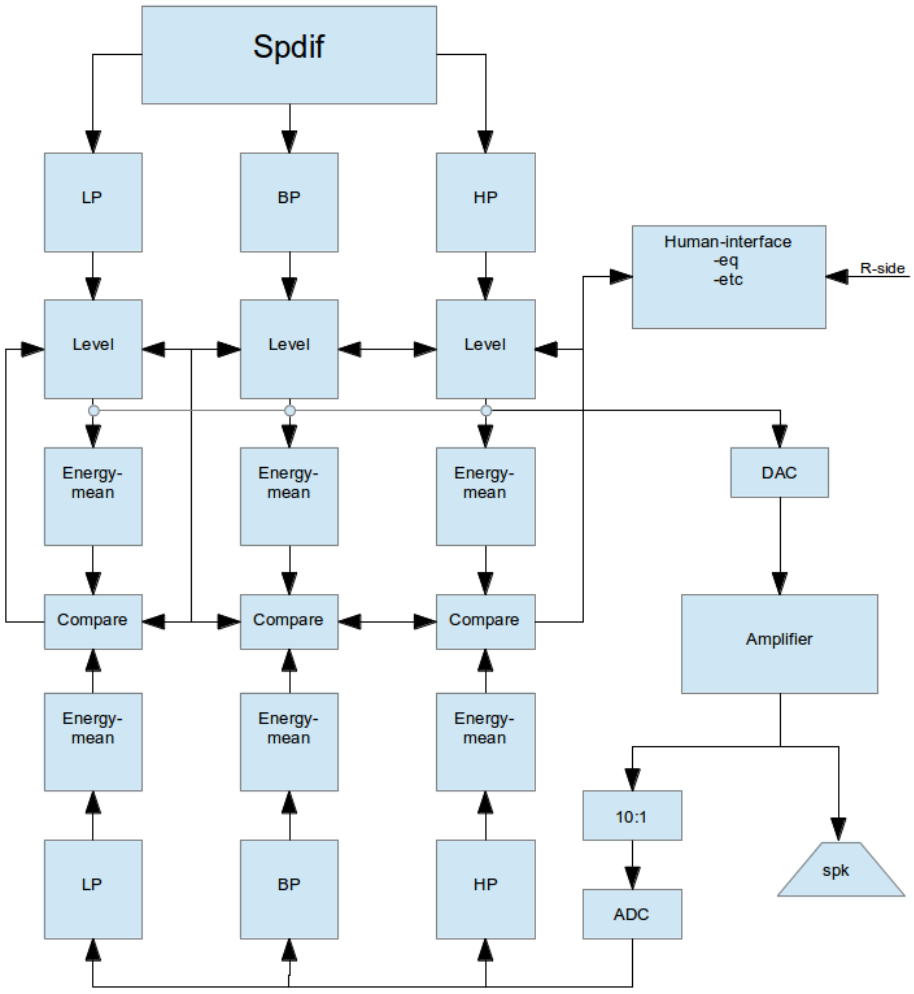

Figure 4.1: Overview Over DSP Solution(only left side)

The Spdif(Sony/Phillips Digital Interface) input signal will go trough 3 filters, a low, band, and high-pass. After the filters the signal will go trough a gain control, to control the wanted gain in the specific band. After this the signal will split, one to the DAC output(to amplifier), and the other will go through a energy mean algorithm to calculate the mean energy. From the other side the ADC(from output of amplifier) signal will pass 3 identical filters, the signal from the ADC low, band, and high-pass filters will then also enter a mean energy algorithm. When the mean energy has been calculated from both the Spdif and the ADC, energy data will be compared. The compared band-pass filters are the reference, and the low and high-pass filters has to adjust out from the band-pass reference.
The gain adjustment of the filters in the Spdif section works as an equalizer, and can be adjusted by an interface from the human user. When the signal go through the DAC, and amplifier it will get feedback to the ADC through a voltage splitter, the ratio of the splitter 10 to 1 so it not saturate the ADC. In appendix there is a diagram over the voltage splitter, and a table shown measurements of the input(from DAC) and the output of the amplifier.
The filters are chosen to be first order butterworth filters. The choice of the butterworth filters is because of the flat frequency response in the pass band.

4.2.2 Choosing Of Octave Frequencies For The Filters

The goal is to create a 3 band equalizer(tone control). The choose of filters has been one low-pass, one band-pass filters, and one high-pass filter. One low-pass for the low region, one band-pass for the middle region and a high-pass for the high region.

$AudioOctave_{band_n}$	f_{a1_n}	f_{ctr_n}	f_{a2_n}	**S**elected Band
1	11.049	15.625	22.097	Low-band
2	22.0971	31.25	41.1942	Low-band
3	44.1942	62.5	88.3883	Low-band
4	88.3883	125	176.777	Low-band
5	176.777	250	353.553	Mid-band
6	353.553	500	707.107	Mid-band
7	707.107	1Khz	1414.21	Mid-band
8	1414.21	2Khz	2828.43	Mid-band
9	2828.43	4Khz	5656.85	Mid-band
10	5656.85	8Khz	11313.7	High-band
11	11313.7	16Khz	22627.4	High-band

Table 4.2: This table shows the chosen octaves for the filters low, mid, high-band

A look at the calculated octave bands scheme gives the above choices:

Low Low-pass: Band 1 to 4 gives f_a = 176.777hz for the bass region.
Mid Band-pass: Band 5 to 9 gives f_{a1} = 176.777hz and f_{a2} = 5656.85hz for the middle region.
High High-pass: Band 10 and 11 gives f_a = 5656.85hz for the high region.

The low, mid, and high-band frequencies is chosen out from the description in the analyse part where the low band is defined to be the bass area, and the mid is the mid tone area, and the high is the highs area.

4.2.3 Filter Calculations

In this part the filters a calculated. First is the analogies filter calculated, simulated and adjusted. Next the z transfer functions of filters is calculated by using bilinear z-transform method, followed by simulation and adjustment. Then the filters will be transformed into difference equations, and simulated in Matlab, and scaled up by 2^{15}, and a sinus wave will go through the hole audio spectrum to check if any overflow occur the the equation. If an overflow is detected the the gain in the difference equation will be scaled.

4.2.3.1 Low, Mid, and High Band Calculations

All analogies filters are calculated, out from the octave table and the chosen bands.

Low-pass: f_a=176.777hz

$$
\begin{aligned}
H(S)_{norm} &= \frac{1}{S+1} \qquad & H(S)_{norm\rightarrow lp} &= H(S)_{norm}\Big|_{S=\frac{s}{\Omega_a}} \\
H(s)_{lp} &= \frac{\Omega_a}{s+\Omega_a} \qquad & where: \Omega_a &= 2\pi \cdot f_a \\
H(s)_{lp} &= \frac{2\pi \cdot 176.777hz}{s+2\pi \cdot 176.777hz} = \underline{\underline{\frac{1110.72}{s+1110.72}}}
\end{aligned}
\tag{4.12}
$$

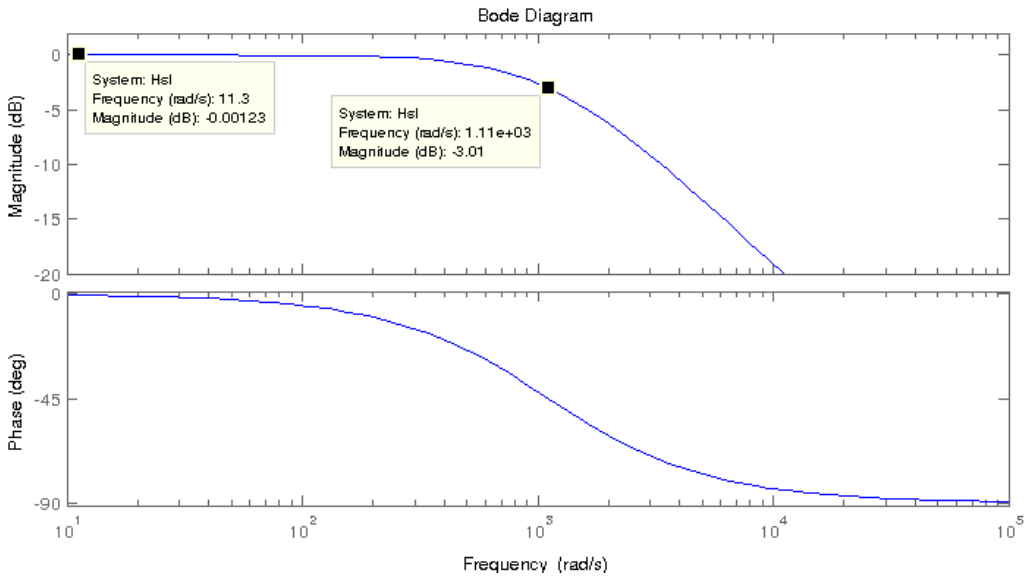

Figure 4.2: Bode-plot over the Low-pass filter for the low band.

Band-pass: f_{a1}=176.777hz, and f_{a2}=5656.85hz:

$$H(S)_{norm} = \frac{1}{S+1} \qquad H(S)_{norm\to bp} = H(S)_{norm}\Big|_{S=\frac{s^2+\Omega_c^2}{\Delta\Omega\cdot s}}$$

$$H(s)_{bp} = \frac{\Delta\Omega\cdot s}{s^2+\Delta\Omega\cdot s+\Omega_c^2} \qquad where:$$

$$\Omega_a = 2\pi\cdot f_a$$
$$\Delta\Omega = \Omega_{a2}-\Omega_{a1}$$
$$\Omega_c = \sqrt{\Omega_{a1}\cdot\Omega_{a2}} \tag{4.13}$$

$$\Omega_{a1} = 2\pi\cdot f_{a1} = 2\pi\cdot 176.777hz = \underline{1110.78}[rad/s]$$
$$\Omega_{a2} = 2\pi\cdot f_{a2} = 2\pi\cdot 5656.85hz = \underline{35543}[rad/s]$$
$$\Omega_c = \sqrt{\Omega_{a1}\cdot\Omega_{a2}} = \sqrt{1110.78\cdot 35543} = \underline{6283.35}[rad/s]$$
$$\Delta\Omega = \Omega_{a2}-\Omega_{a1} = 35543-1110.78 = \underline{34432.2}[rad/s]$$

$$H(s)_{bp} = \frac{34432.2\cdot s}{\underline{s^2+34432.2\cdot s+39480487.2225}}$$

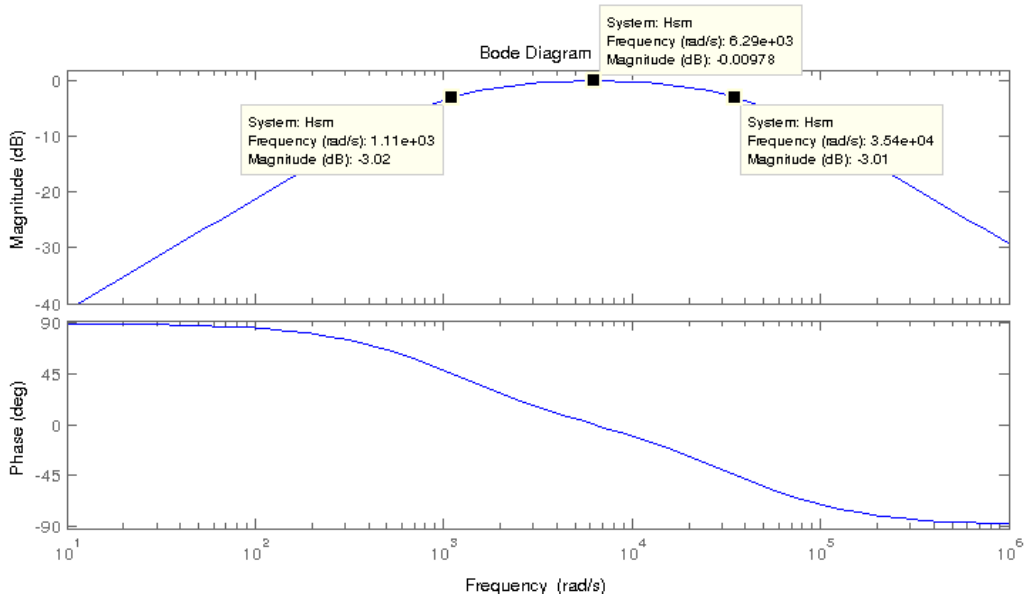

Figure 4.3: Bode-plot over the Band-pass filter for the mid band.

High-pass: f_a=5656.85hz:

$$H(S)_{norm} = \frac{1}{S+1} \qquad H(S)_{norm \to hp} = H(S)_{norm}\Big|_{S=\frac{\Omega_a}{s}}$$

$$H(s)_{hp} = \frac{s}{s+\Omega_a} \qquad where: \quad \Omega_a = 2\pi \cdot f_a \tag{4.14}$$

$$H(s)_{hp} = \frac{s}{s+2\pi \cdot 5656.85} = \frac{s}{\underline{\underline{s+35543}}}$$

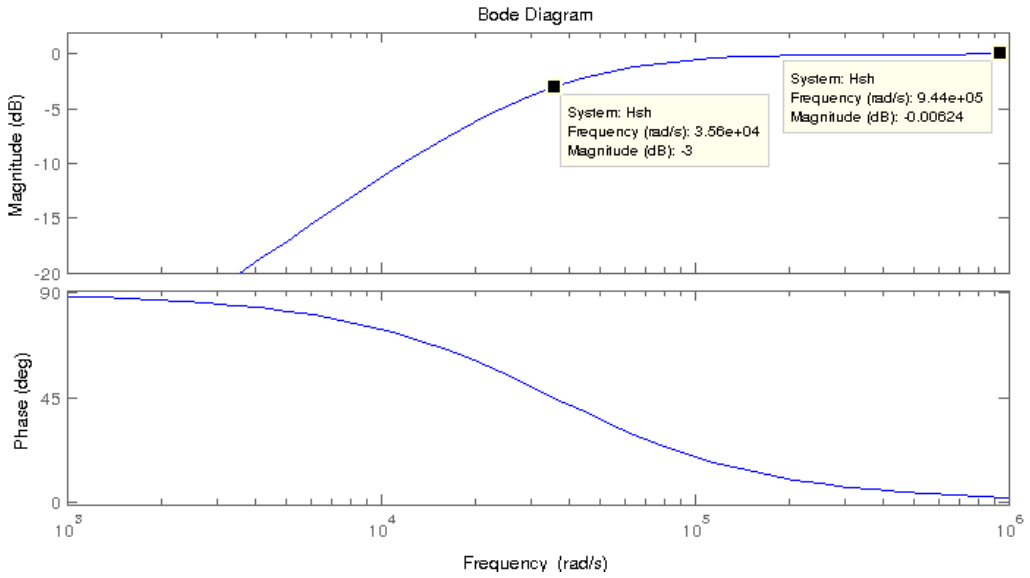

Figure 4.4: Bode-plot over the High-pass filter for the high band.

Adding Low, Mid, and High together:

Now all 3 filters has to be added together for an analyse of the frequency response. Below there is a diagram explaining the addition.

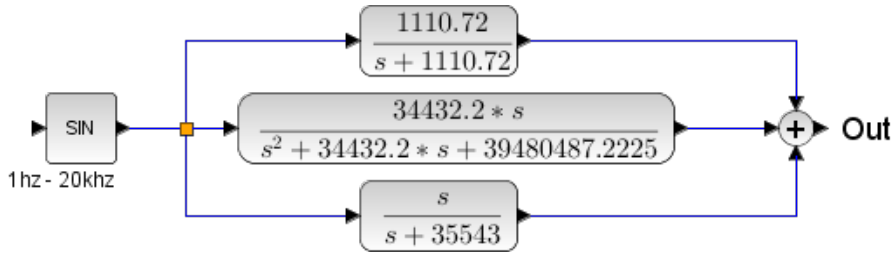

Figure 4.5: Diagram over Low, Band, and High-pass added together.

Looking at the Bode-plot of the 3 filters added together, reveals an overshoot at the Ω_c=6283.35[rad/s] of the band-pass filter, with a magnitude of 0.511dB.

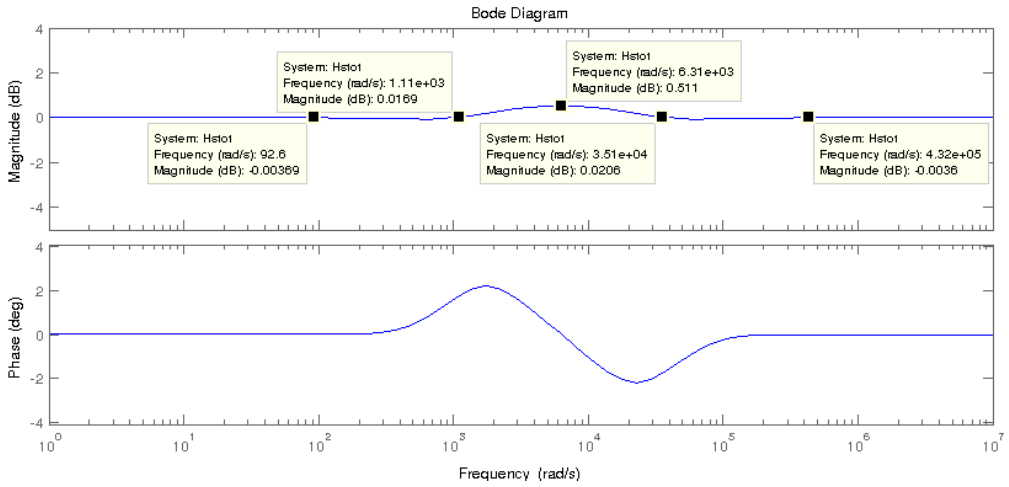

Figure 4.6: Low, Band and High-pass added together, there is an overshoot at Ω_c=6283.35[rad/s] in the Band-pass region there has to be corrected.

Scaling the Mid gain:

To scale the gain in the mid-bands band-pass filter, the gain scaling factor has to be calculated by adding the overshoot magnitude of 0.511dB, by the length of the original magnitude of -0.00978. By a litle bit of experiment it has to get a little more so the original was set to -0.08dB this gives the bellow.

$$
\begin{aligned}
&20 \cdot \log(A) = dB \implies \\
&20 \cdot \log(\Delta A) = \Delta dB \implies \\
&20 \cdot \log(\Delta A) = dB_{max} + |dB_{min}| \implies \\
&20 \cdot \log(\Delta A) = 0.511 + |-0.08| \implies \\
&20 \cdot \log(\Delta A) = 0.591 \implies \\
&\log(\Delta A) = \frac{0.591}{20} \implies \\
&\Delta A = 10^{(0.591/20)} = \underline{\underline{1.07041}} \\
&A_0 = \frac{1}{\Delta A} = \frac{1}{1.07041} = 0.934 \approx \underline{\underline{0.93}}
\end{aligned}
\qquad
\begin{aligned}
&where: \\
&\quad dB_{max} = 0.511dB \\
&\quad dB_{min} = -0.08dB \\
&\quad \Delta dB = dB_{max} + |dB_{min}| \\
&\quad \Delta A = \Delta magnitude \\
&\quad A_0 = \frac{1}{\Delta A} = scale_{factor}
\end{aligned}
$$

$$
H(s)_{bp} = A_0 \cdot \frac{\Delta\Omega \cdot s}{s^2 + \Delta\Omega \cdot s + \Omega_c^2} = 0.93 \cdot \frac{34432.2 \cdot s}{s^2 + 34432.2 \cdot s + 39480487.2225} = \underline{\underline{\frac{32021.9 \cdot s}{s^2 + 34432.2 \cdot s + 39480487.2225}}} \tag{4.15}
$$

A look at the bode-plot for the 3 bands with mid-band scaled down, bellow shows that the filters added together is now correct and the gain is now below 1.

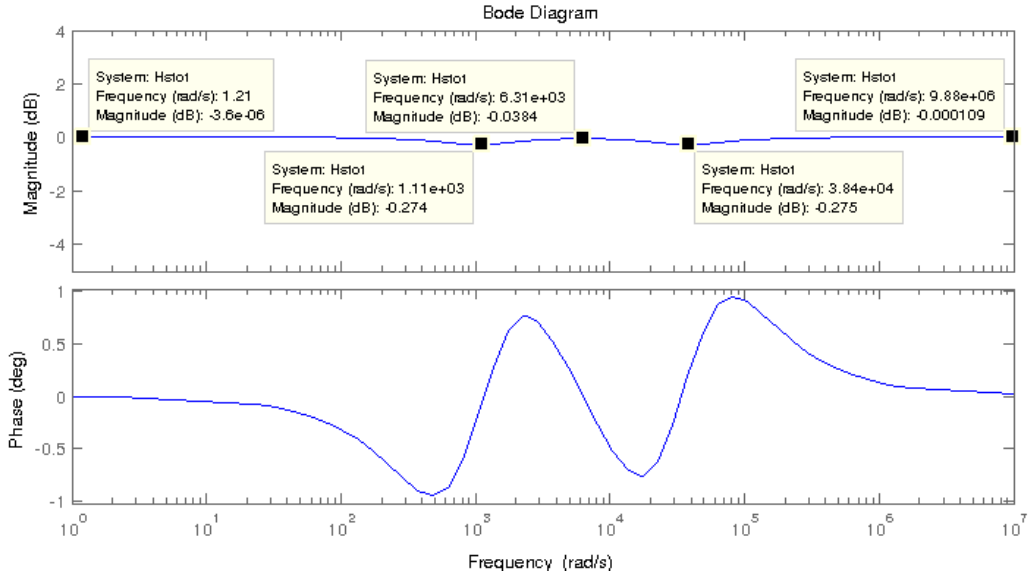

Figure 4.7: Low, Band and High-pass added together, where the Band-pass gain has been scaled by 0.93.

4.2.4 bilinear Z-transform(Tustin's method)

4.2.4.1 Pre-warping:

Before the 3 filters in S-domain can be transformed over in the Z-domain, the attenuation frequencies has to be pre-warped to fit over in the Z-domains unit circle, after this the bilinear z-transform(Tustin's method) can be calculated. The formulas for the pre-warping, and bilinear z-transform is:

Pre-warping and z-transform formulas:

$$\omega_a = \tfrac{2}{T_s} \cdot \tan \frac{2\pi \cdot f_a \cdot T_s}{2} = \tfrac{2}{T_s} \cdot \tan \frac{\Omega_a \cdot T_s}{2}$$

$$H(z) = H(s) \underset{s \approx}{} \frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}$$

$Where:$

$\Omega_a = Analogue_{attenuation}$

$\omega_a = Digital_{attenuation}$

$f_s = 1/T_s = Sampling_{frequency}$

$T_s = 1/f_s = Sampling_{period}$

(4.16)

Pre-Warped Transfer Function Low-pass filter:

Thus is the transfer function pre-warped for the low-pass with the $\Omega_a = 1110.72, f_s = 44.1khz$.

$$\omega_a = \frac{2}{T_s} \cdot \tan \frac{\Omega_a \cdot T_s}{2} = \frac{2}{1/44100} \cdot \tan \frac{1110.72 \cdot (1/44100)}{2} = \underline{\underline{1110.78}}$$

$$H(s)_{lp_{pre}} = \frac{\omega_a}{s + \omega_a} = \frac{1110.78}{\underline{\underline{s + 1110.78}}}$$

$$H(z)_{lp} = \frac{1110.78}{\frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} + 1110.78} = \frac{1110.78}{\frac{2}{1/44100} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} + 1110.78} = \frac{1110.78 + 1110.78 \cdot z^{-1}}{\underline{\underline{89310.8 - 87089.2 \cdot z^{-1}}}}$$

(4.17)

A look at the bode-plot over the $H(z)_{lp}$ shows that the filter is mapped fine on the $H(s)_{lp}$:

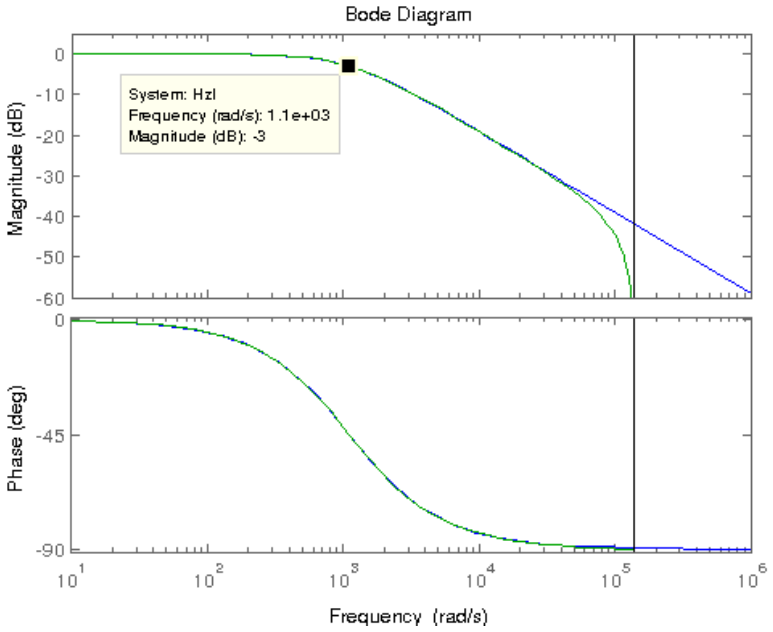

Figure 4.8: Bode-plot over $H(z)_{lp}$(green) maps fine on the $H(s)_{lp}$(blue).

Pre-Warped Transfer Function Band-pass filter:

The transfer function pre-warped for the band-pass with the $\Omega_{a1} = 1110.72$, $\Omega_{a2} = 35543$, $f_s = 44.1khz$, and the scaling factor $A_0 = 0.93$ gives.

$$\omega_{a1} = \frac{2}{T_s} \cdot \tan \frac{\Omega_{a1} \cdot T_s}{2} = \frac{2}{1/44100} \cdot \tan \frac{1110.72 \cdot (1/44100)}{2} = \underline{\underline{1110.78}}$$
$$\omega_{a2} = \frac{2}{T_s} \cdot \tan \frac{\Omega_{a2} \cdot T_s}{2} = \frac{2}{1/44100} \cdot \tan \frac{35543 \cdot (1/44100)}{2} = \underline{\underline{37600.81}}$$

$$\omega_c = \sqrt{\omega_{a1} \cdot \omega_{a2}} = \sqrt{1110.78 \cdot 37600.81} = \underline{\underline{6462.68}}$$

$$\Delta\omega = \omega_{a2} - \omega_{a1} = 37600.81 - 1110.78 = \underline{\underline{36490}}$$

$$H(s)_{bp_{pre}} = A_0 \cdot \frac{\Delta\omega \cdot s}{s^2 + \Delta\omega \cdot s + \omega_c^2} = 0.93 \cdot \frac{36490 \cdot s}{s^2 + 36490 \cdot s + (6462.68)^2} = \underline{\underline{\frac{33935.7 \cdot s}{s^2 + 36490 \cdot s + 41766232.78}}}$$

$$H(z)_{bp} = \frac{33935.7 \cdot \frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}}{(\frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}})^2 + 36490 \cdot \frac{2}{T_s} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} + 41766232.78} =$$

$$\frac{33935.7 \cdot \frac{2}{1/44100} \cdot \frac{1 - z^{-1}}{1 + z^{-1}}}{(\frac{2}{1/44100} \cdot \frac{1 - z^{-1}}{1 + z^{-1}})^2 + 36490 \cdot \frac{2}{1/44100} \cdot \frac{1 - z^{-1}}{1 + z^{-1}} + 41766232.78} = \underline{\underline{\frac{0.650315 - 0.650315 \cdot z^{-2}}{z^{-2} - 3.36222 \cdot z^{-1} + 2.39853}}}$$

(4.18)

A look at the bode-plot over the $H(z)_{bp}$ shows that the filter is mapped fine on the $H(s)_{bp}$:

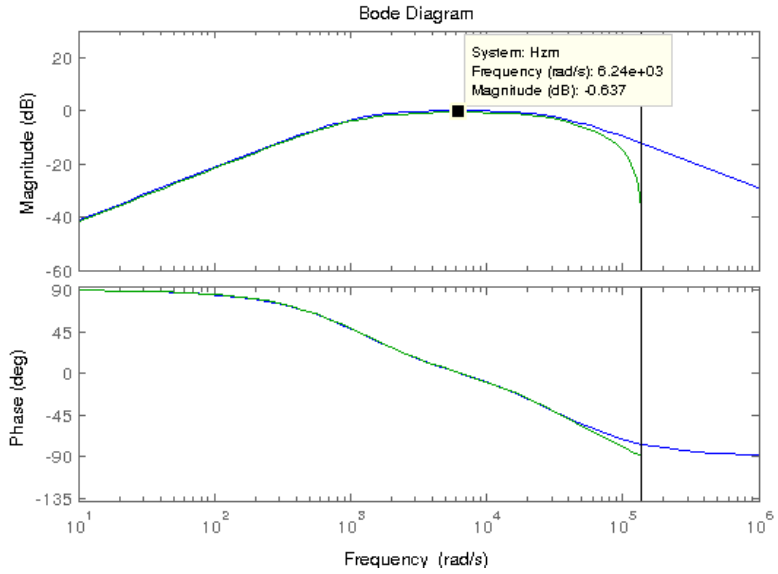

Figure 4.9: Bode-plot over $H(z)_{bp}$(green) maps fine on the $H(s)_{bp}$(blue).

Pre-Warped Transfer Function High-pass filter:

The transfer function pre-warped for the high-pass with the $\Omega_a = 35543$, $f_s = 44.1khz$ gives.

$$\omega_a = \frac{2}{T_s} \cdot \tan\frac{\Omega_a \cdot T_s}{2} = \frac{2}{1/44100} \cdot \tan\frac{35543 \cdot (1/44100)}{2} = \underline{\underline{37600.81}}$$

$$H(s)_{hp_{pre}} = \frac{s}{s+\omega_a} = \underline{\underline{\frac{s}{s+37600.81}}}$$

$$H(z)_{hp} = \frac{\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}}}{\frac{2}{T_s} \cdot \frac{1-z^{-1}}{1+z^{-1}} + 37600.81} = \frac{\frac{2}{1/44100} \cdot \frac{1-z^{-1}}{1+z^{-1}}}{\frac{2}{1/44100} \cdot \frac{1-z^{-1}}{1+z^{-1}} + 37600.81} = \underline{\underline{\frac{1-z^{-1}}{1.426313 - 0.573687 \cdot z^{-1}}}} \tag{4.19}$$

A look at the bode-plot over the $H(z)_{hp}$ shows that the filter is mapped fine on the $H(s)_{hp}$:

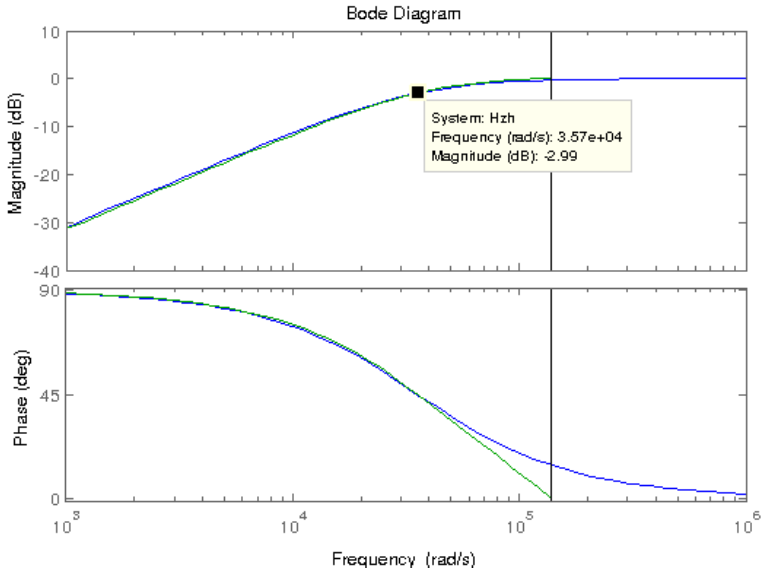

Figure 4.10: Bode-plot over $H(z)_{hp}$(green) maps fine on the $H(s)_{hp}$(blue).

Adding Low, Mid, and High together:

The 3 filters added together $H(z)_{tot}$ maps fine on the $H(s)_{tot}$:

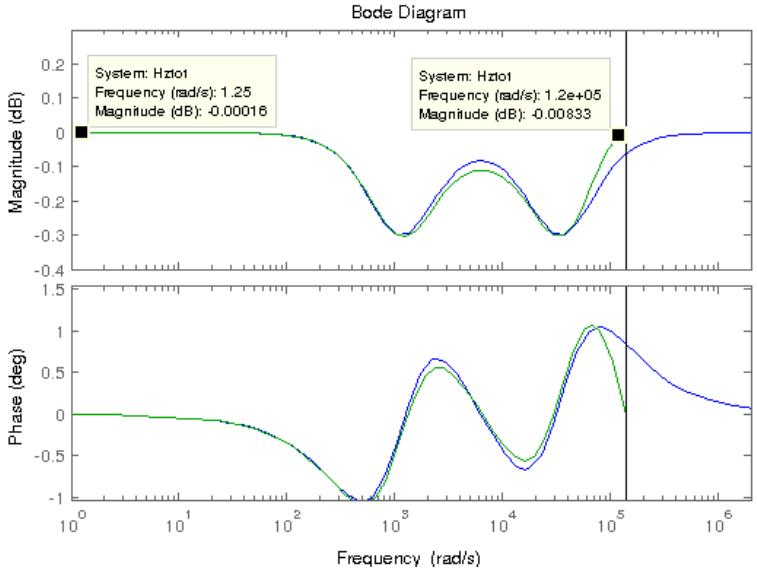

Figure 4.11: Bode-plot over $H(z)_{tot}$(green) maps fine on the $H(s)_{tot}$(blue).

4.2.5 Difference Equations

To convert the z transfer function over in the discrete time difference equation there is used the following z transform.

$$
\begin{aligned}
&X(z) = \sum_{n=-\infty}^{\infty} = X[n] \cdot z^{-1} && Let: \\
& && m = n - n_0 \\
& && n = m + n_n 0 \\
&\sum_{n=-\infty}^{\infty} X[n-n_0] \cdot z^{-1} = \sum_{m=-\infty}^{\infty} X[m] \cdot z^{-(m+n_0)} = \\
&\sum_{m=-\infty}^{\infty} X[m] \cdot z^{-m} \cdot z^{-n_0} = z^{-n_0} \cdot \sum_{m=-\infty}^{\infty} X[m] \cdot z^{-m} = z^{-n_0} \cdot X(z) \implies \\
&\underline{\underline{X[n-n_0] = z^{-n_0} \cdot X(z)}}
\end{aligned}
\tag{4.20}
$$

Difference equation for Low-pass filter:

$$
\begin{aligned}
&H(z) = \frac{Y(z)}{X(z)} = \frac{1110.78 + 1110.78 \cdot z^{-1}}{89310.8 - 87089.2 \cdot z^{-1}} \implies \\
&Y(z) \cdot (89310.8 - 87089.2 \cdot z^{-1}) = X(z) \cdot (1110.78 + 1110.78 \cdot z^{-1}) \implies \\
&89310.8 \cdot Y(z) = 1110.78 \cdot X(z) + 1110.78 \cdot X(z) \cdot z^{-1} + 87089.2 \cdot Y(z) \cdot z^{-1} \implies \\
&Y(z) = 0.012437 \cdot X(z) + 0.012437 \cdot X(z) \cdot z^{-1} + 0.975125 \cdot Y(z) \cdot z^{-1} \implies \\
&\underline{\underline{y[n] = 0.012437 \cdot x[n] + 0.012437 \cdot x[n-1] + 0.975125 \cdot y[n-1]}} \\
&\underline{\underline{y[n] = 407 \cdot x[n] + 407 \cdot x[n-1] + 31952 \cdot Y[n-1]}} \ by \cdot 2^{15}
\end{aligned}
\tag{4.21}
$$

By input of a sinus of $\Omega_a = 1110.72$, the filter is scaled by $2^{15} = 32768 bits_{max}$, looks like:

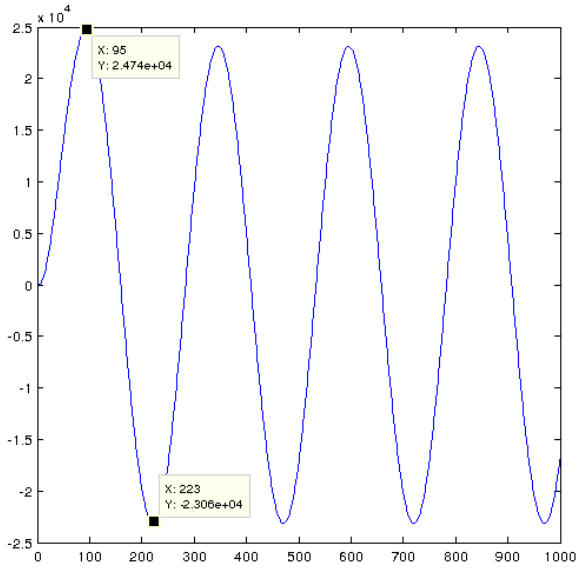

Figure 4.12: Sinus response Low-pass filters difference equation at $\Omega_a = 1110.72$.

A scan of frequencies from $1hzto20Khz$, the filter is scaled by $2^{15} = 32768 bits_{max}$, :

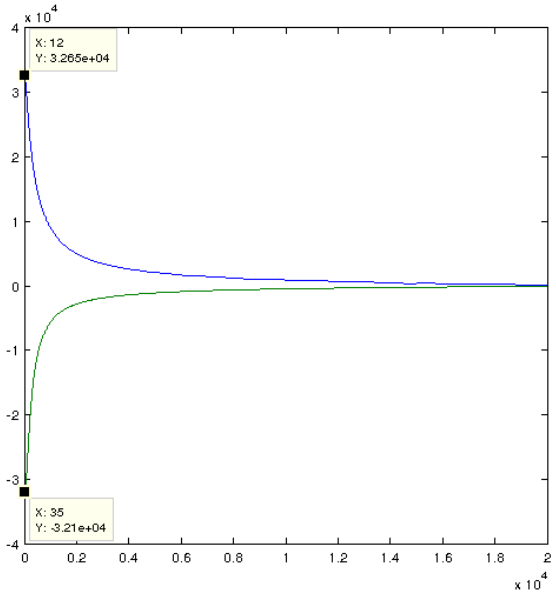

Figure 4.13: Sinus response Low-pass filters from $1hzto20Khz$.

Both plots most not exceed the limit of (+/-)32768 bits so the filter is accepted.

Difference equation for Mid-pass filter:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{0.650315 - 0.650315 \cdot z^{-2}}{z^{-2} - 3.36222 \cdot z^{-1} + 2.39853} \Longrightarrow$$

$$Y(z) \cdot (z^{-2} - 3.36222 \cdot z^{-1} + 2.39853) = X(z) \cdot (0.650315 - 0.650315 \cdot z^{-2}) \Longrightarrow$$

$$2.39853 \cdot Y(z) = 0.650315 \cdot X(z) - 0.650315 \cdot X(z) \cdot z^{-2} + 3.36222 \cdot Y(z) \cdot z^{-1} - Y(z) \cdot z^{-2} \Longrightarrow$$

$$Y(z) = 0.271131 \cdot X(z) - 0.271131 \cdot X(z) \cdot z^{-2} + 1.40178 \cdot Y(z) \cdot z^{-1} - 0.416922 \cdot Y(z) \cdot z^{-2} \Longrightarrow$$

$$\underline{y[n] = 0.271131 \cdot x[n] - 0.271131 \cdot x[n-2] + 1.40178 \cdot y[n-1] - 0.416922 \cdot y[n-2]}$$

$$\underline{y[n] = 8884 \cdot x[n] - 8884 \cdot x[n-2] + 45933 \cdot y[n-1] - 13661 \cdot y[n-2]} \; by \cdot 2^{15} \tag{4.22}$$

By input of a sinus of $\Omega_a = 1110.72$, the filter is scaled by $2^{15} = 32768 bits_{max}$, looks like:

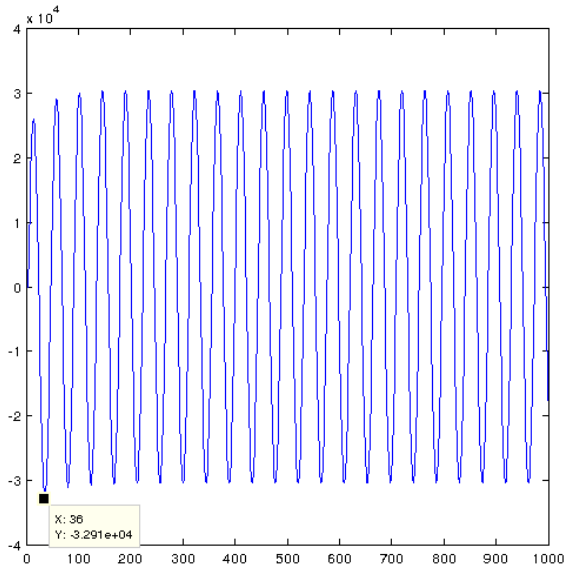

Figure 4.14: Sinus response band-pass filters difference equation at $\Omega_a = 1110.72$ shows a overflow with -32910.

A scan of frequencies from $1hzto20Khz$, the filter is scaled by $2^{15} = 32768bits_{max}$, :

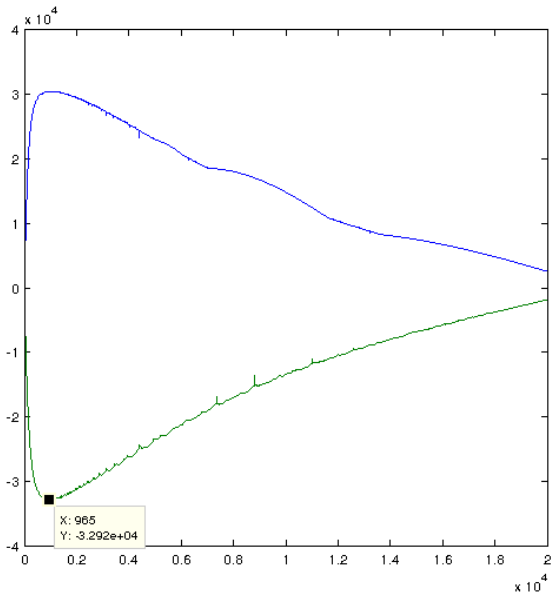

Figure 4.15: Sinus response Low-pass filters from $1hzto20Khz$ shows a overflow with -32920.

Both plots shows an overflow by -32917(Matlab terminal), and has to be corrected by:

$$A_0 = \frac{2^{15}}{|-32917|} = 0.995473$$

$$y[n] = A_0 \cdot 0.271131 \cdot x[n] - A_0 \cdot 0.271131 \cdot x[n-2] + 1.40178 \cdot y[n-1] - 0.416922 \cdot y[n-2] =$$
$$0.995473 \cdot 0.271131 \cdot x[n] - 0.995473 \cdot 0.271131 \cdot x[n-2] + 1.40178 \cdot y[n-1] - 0.416922 \cdot y[n-2] =$$
$$\underline{\underline{0.269904 \cdot x[n] - 0.269904 \cdot x[n-2] + 1.40178 \cdot y[n-1] - 0.416922 \cdot y[n-2]}}$$

$$\underline{\underline{y[n] = 8844 \cdot x[n] - 8844 \cdot x[n-2] + 45933 \cdot y[n-1] - 13661 \cdot y[n-2]}} \; by \cdot 2^{15} \tag{4.23}$$

After this correction the graph is okay and no overflow with a result of -32765_{max}

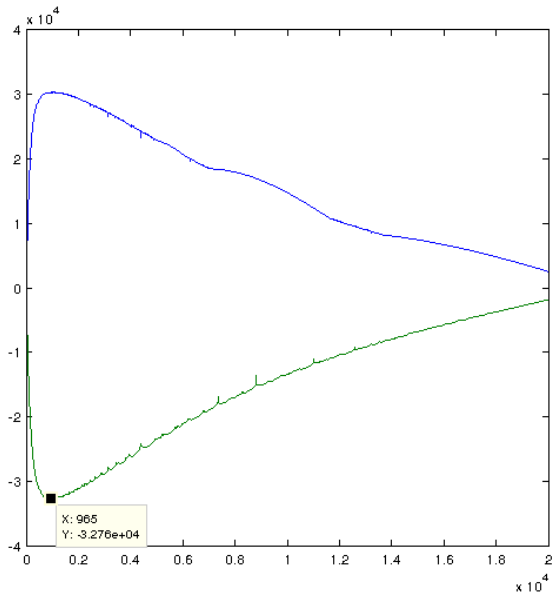

Figure 4.16: Sinus response band-pass filters corrected from $1hz to 20Khz$ shows no overflow by -32765_{max}.

Difference equation for High-pass filter:

$$
\begin{aligned}
&H(z) = \frac{Y(z)}{X(z)} = \frac{1 - z^{-1}}{1.426313 - 0.573687 \cdot z^{-1}} \Longrightarrow \\
&Y(z) \cdot (1.426313 - 0.573687 \cdot z^{-1}) = X(z) \cdot (1 - z^{-1}) \Longrightarrow \\
&1.426313 \cdot Y(z) = X(z) - X(z) \cdot z^{-1} + 0.573687 \cdot Y(z) \cdot z^{-1} \Longrightarrow \\
&Y(z) = 0.701108 \cdot X(z) - 0.701108 \cdot X(z) \cdot z^{-1} + 0.402217 \cdot Y(z) \cdot z^{-1} \Longrightarrow \\
&\underline{\underline{y[n] = 0.701108 \cdot x[n] - 0.701108 \cdot x[n-1] + 0.402217 \cdot y[n-1]}} \\
&\underline{\underline{y[n] = 22973 \cdot x[n] - 22973 \cdot x[n-1] + 13179 \cdot y[n-1]}} \ by \cdot 2^{15}
\end{aligned}
\tag{4.24}
$$

By input of a sinus of $\Omega_a = 1110.72$, the filter is scaled by $2^{15} = 32768 bits_{max}$, looks like:

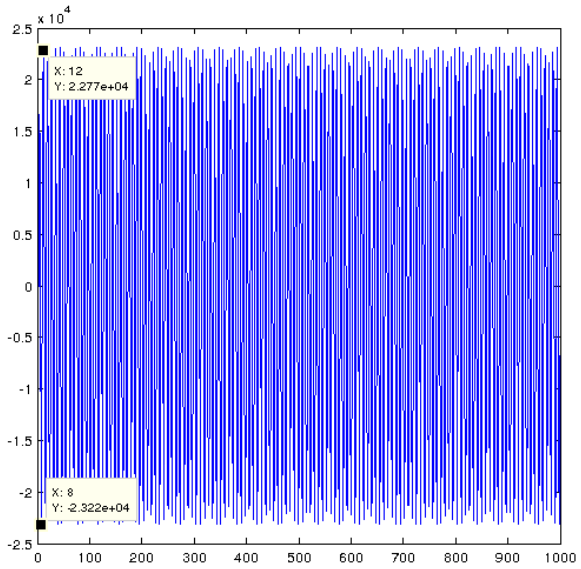

Figure 4.17: Sinus response high-pass filters difference equation at $\Omega_a = 1110.72$ shows no overflow with -23220 bits.

A scan of frequencies from $1hzto20Khz$, the filter is scaled by $2^{15} = 32768 bits_{max}$:

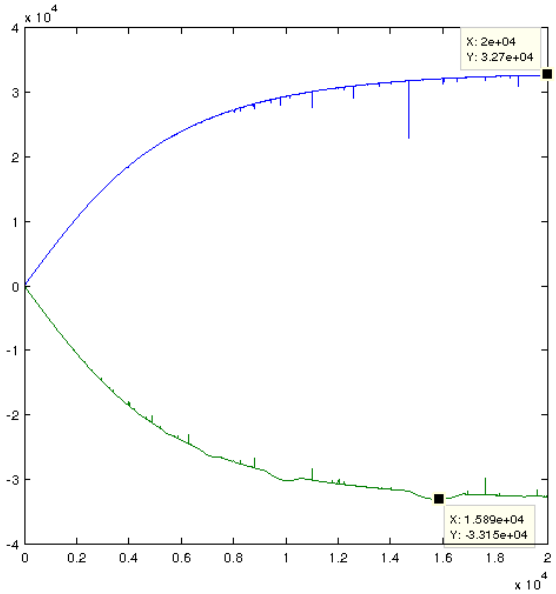

Figure 4.18: Sinus response high-pass filters from $1hzto20Khz$.

The first curve do not show any overshoot, but a full scan from 1 to 20khz shows an overflow by -33148(Matlab terminal), and has to be corrected by:

$$A_0 = \frac{2^{15}}{|-33148|} = 0.988536$$

$$\begin{aligned} &y[n] = A_0 \cdot 0.701108 \cdot x[n] - A_0 \cdot 0.701108 \cdot x[n-1] + 0.402217 \cdot y[n-1] = \\ &0.988536 \cdot 0.701108 \cdot x[n] - 0.988536 \cdot 0.701108 \cdot x[n-1] + 0.402217 \cdot y[n-1] = \\ &\underline{\underline{0.693071 \cdot x[n] - 0.693071 \cdot x[n-1] + 0.402217 \cdot y[n-1]}} \end{aligned} \tag{4.25}$$

$$\underline{\underline{y[n] = 22710 \cdot x[n] - 22710 \cdot x[n-1] + 13179 \cdot y[n-1]}} \ by \cdot 2^{15}$$

A scan over the frequencies from $1hzto20Khz$, shows no overflow, and the filter is now accepted:

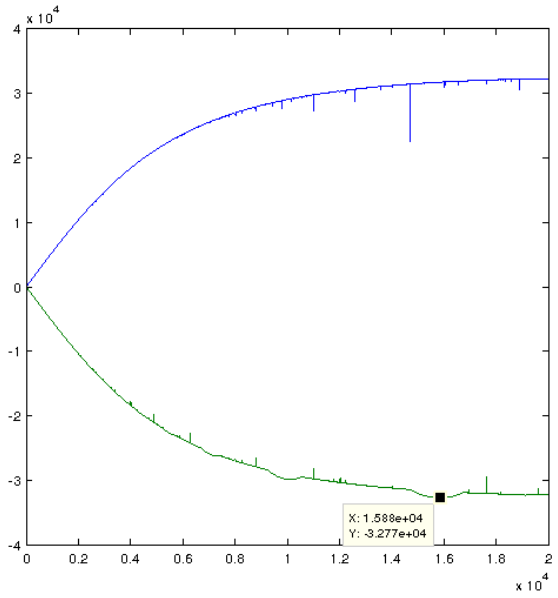

Figure 4.19: Sinus response high-pass filter corrected from $1hzto20Khz$ shows no overflow by -32767_{max}(Matlab terminal).

Difference equation added together:

A scan of the 3 filter added together, will determine if there is any overflow in the hole audio spectrum:

$$y[n] = y[n]_{low} + y[n]_{mid} + y[n]_{high} \tag{4.26}$$

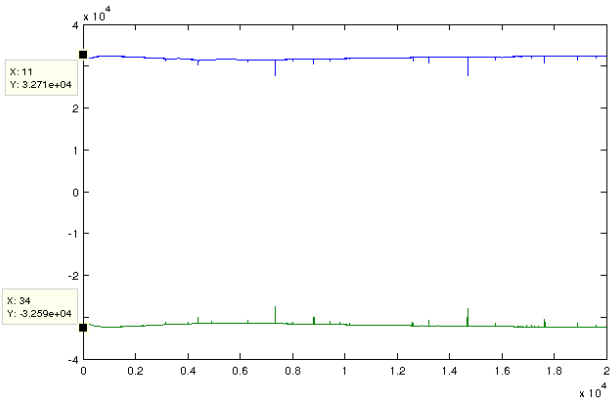

Figure 4.20: Sinus response for all 3 filters added together, from $1hzto20Khz$ shows no overflow by -32767_{max}(Matlab terminal).

The full scan shows no overflow in the filters, with a min at 1hz of -32591(Matlab terminal), and a max at 1hz of 32710(Matlab terminal). Filters are ready to be implemented!!!

4.2.6 Adjust gain in equalizer and lift to maximum

When the gain in equalizer is to be adjusted, the maximum gain point have to be at the max gain point, to get the maximum out of the filters. The 2 smallest filters has to follow the highest. Below is shown how the signal is lifted up to the maximum gain point. To the left the filters are not lifted, and to the right they are.

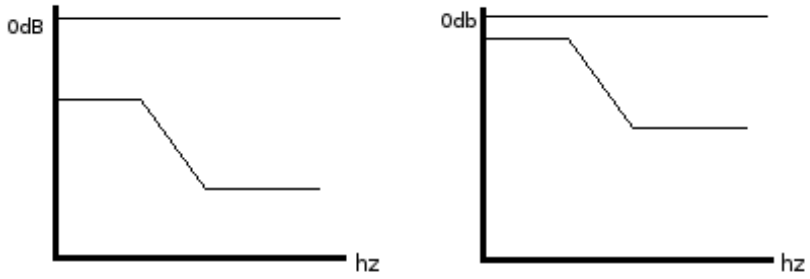

Figure 4.21: Left not lifted to maximum, Right lifted to maximum.

To do this job a small algorithm has been developed. The algorithm has not been implemented, but has been simulated in Matlab. Below is the small program:

```
.Gain and adjust to zero Matlab
.
.   boostL = gain_lp;
.   boostM = gain_bp;
.   boostH = gain_hp;
.
.   if(boostH > boostL)
.       if(boostH >= 1)
.           boostM = boostM - (boostH - boostM);
.           bufL = 1 - (boostH - boostL);
.           bufH = 1;
.       elseif(boostH < 1)
.           bufH = boostM - (boostM - boostH)
.           bufL = boostM - (boostM - boostL)
.           boostM = 1;
.       end
.
.   elseif(boostL > boostH)
.       if(boostL >= 1)
.           boostM = boostM - (boostL - boostM);
.           bufH = 1 - (boostL - boostH);
.           bufL = 1;
.       elseif(boostL < 1)
.           boostM = 1;
.           bufH = boostM - (boostM - boostH);
.           bufL = boostM - (boostM - boostL);
.           boostM = 1;
.       end
.
.   elseif(boostL == boostH)
.       if(boostH < 1)
.           bufH = boostM - (boostM - boostH)
.           bufL = boostM - (boostM - boostL)
.           boostM = 1;
.       elseif(boostH >= 1)
.           boostM = boostM - (boostH - boostM)
.           bufH = 1;
.           bufL = 1;
.       end
.   end
.   boostL = bufL;
.   boostH = bufH;
```

There is 3 main functions in the code. One when high is biggest, one when low is biggest, and one when they are equal. Inside each function there is 2 functions, one when high or low is bigger or equal to 1.
When high or equal to 1 means that they are bigger than the mid(in this project mid is the reference band), and is the dominate part. The other is saying when lower than 1, this means that the high or low is below the mid, and the mid is now the dominate part there has to be at the maximum point. When high, mid, or low is dominate, the other bands has to follow up, this is done by subtract the length from the dominate part.

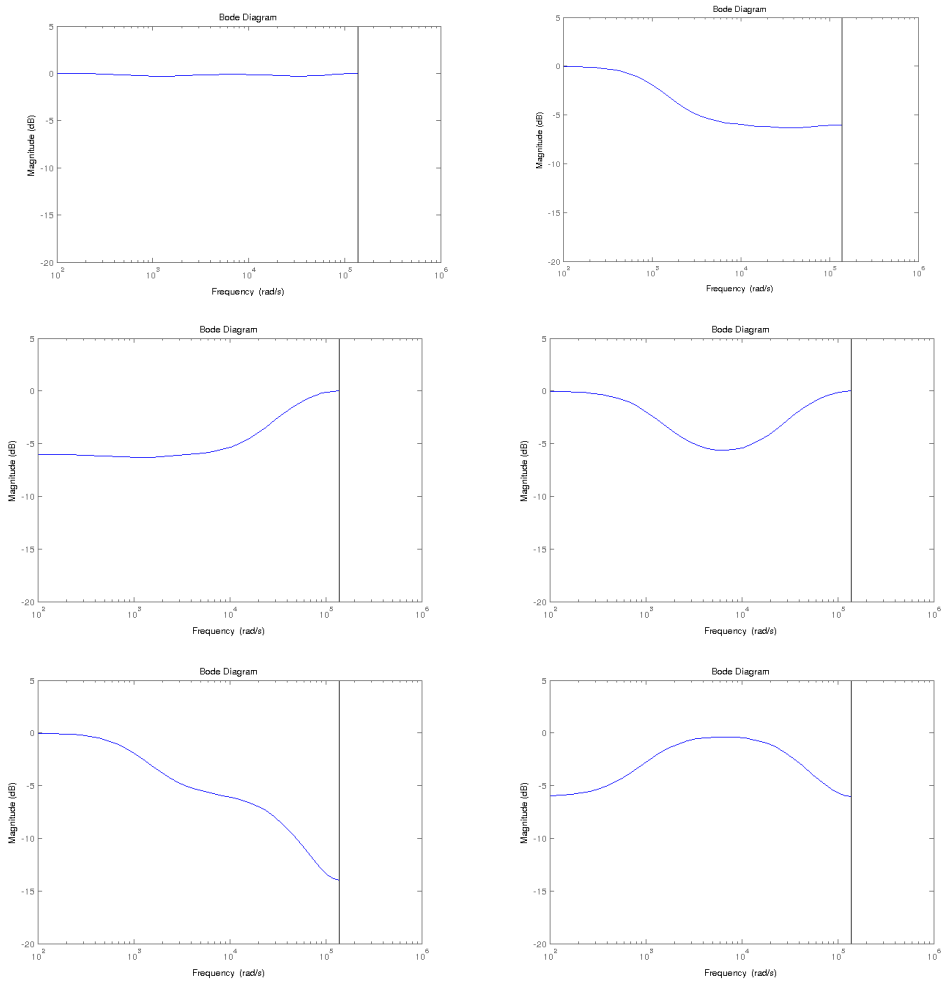

Figure 4.22: 1)lp, bp and hp is equal. 2)lp is higher, bp and hp is equal. 3)hp is higher, lp and bp is equal. 4)both lp and hp is higher than bp. 5)lp is higher, hp is lower 6)both lp and hp is lower than bp

Above is shown 6 screen shoots from Matlab over how the biggest is always at maximum gain point. The first is all the same, then low band is highest, then high is highest, then both low and high, then low is high and high is low, and last the mid is highest.

4.2.7 Software Microblaze In C

In this part the implementation of the filters, and other algorithm is shown.

4.2.8 Overview Of Software

The overall program consist of an infinite loop and a interrupt function. When the system starts up, it will first set up some global variables. All filter and energy variables are made global, so if a variable has to be set somewhere in the code, it will work easily everywhere. When the variables has been declared the code will initialize all ports, peripherals, and interrupt. Then the I2C will send the register set up to the CODEC and then activate interrupt, the code will now wait in the infinite loop for an interrupt. When a interrupt appears it will disable the interrupt, and read the incoming data on Spdif and ADC port. Then the data from the to ports will be signal processed, when the data has been processed, it will be send by the DAC port, then then interrupt will be activated again, and jump back to the infinite loop. Below is a simple sketch diagram over the overall system build up.

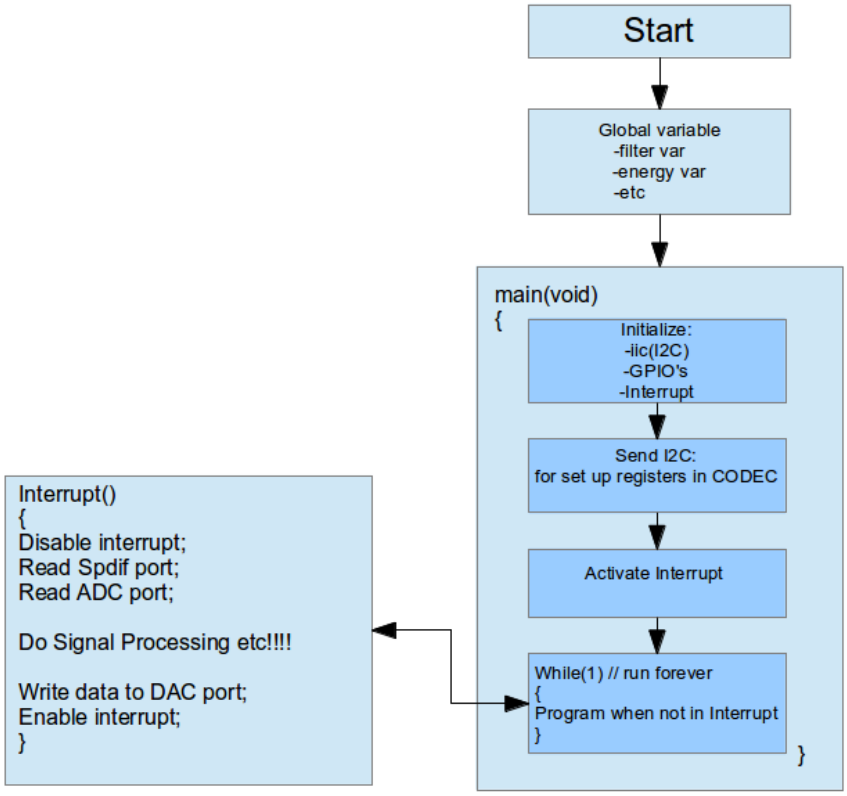

Figure 4.23: Overview over the C-code structure in the Microblaze DSP.

In appendix there is a speed test of the Microblaze to see how many filters there can be implemented.

4.2.9 Interrupt Code

Interrupt function is made up to be the main function, all algorithms and filter calculation etc is implemented here. When an interrupt occurs, a jump from the infinite while loop to the interrupt function will be performed. The interrupt function below shows the implementing of the filters etc in this project. There is only demonstrated(for convenience) the 2 mid band band-pass filters from the left side of the Spdif and ADC.

```
void GpioIsr(void *InstancePtr)
{

//All variables shown in this demonstration code, are global's. There is only shown the left mid band filters for the Spdif and ADC,
//this is for keeping the simplicity and understanding of the code.


//Disable the interrupt
	XGpio_InterruptDisable(GpioPtr, XPAR_GENERIC_GPIO_1_IP2INTC_IRPT_MASK);


//Read data from spdif and ADC
	Data_spdif = XGpio_DiscreteRead(&Gpio_spdif, GPio_CHANNEL);
	Data_ADC = XGpio_DiscreteRead(&Gpio_ADC, Gpio_CHANNEL);

//sort out spdif data  store it in Spdif_X0_Left
	Spdif_X0_Left = 0;
	Spdif_X0_Left |= (((Data_spdif >> 16) & 0xffff) << 16);
	Spdif_X0_Left = (Spdif_X0_Left >> 16);

//ADC out ADC data store it in ADC_X0_Left
	ADC_X0_Left = 0;
	ADC_X0_Left |= (((Data_ADC >> 16) & 0xffff) << 16);
	ADC_X0_Left = (ADC_X0_Left >> 16);

//Spdif left mid band filter
	Spdif_Y0_Left_mid = 8844* Spdif_X0_Left - 8844 * Spdif_X2_Left + 45933* Spdif_Y1_Left_mid - 13661* Spdif_Y2_Left_mid;
	Spdif_Y0_Left_mid = Spdif_Y0_Left_mid;
	Spdif_Y2_Left_mid = ((Spdif_Y1_Left_mid));
	Spdif_Y1_Left_mid = ((Spdif_Y0_Left_mid) >> 16);
	Spdif_Left_mid_mean_energy = ((Spdif_Left_mid_mean_energy + (Spdif_Y1_Left_mid * Spdif_Y1_Left_mid)) >> 1);
	Spdif_Left_mid_out = ((Spdif_Y0_Left_mid >> 16) * (256 * EQ_mid_Left));

//ADC_Left_mid band filter
	ADC_Y0_Left_mid =  8844 * ADC_X0_Left - 8844 * ADC_X2_Left + 45933* ADC_Y1_Left_mid - 13661 * ADC_Y2_Left_mid;
	ADC_Y2_Left_mid = ADC_Y1_Left_mid;
	ADC_Y1_Left_mid = (ADC_Y0_Left_mid >> 16);
	ADC_Left_mid_mean_energy = ((ADC_Left_mid_mean_energy + (ADC_Y1_Left_mid * ADC_Y1_Left_mid)) >> 1);
	ADC_Left_mid_out =  (ADC_Y0_Left_mid);

//Spdif save x[0],x[n-1],x[n-2]
	Spdif_X2_Left = Spdif_X1_Left;
	Spdif_X1_Left = Spdif_X0_Left;

//ADC save x[0],x[n-1],x[n-2]
	ADC_X2_Left = ADC_X1_Left;
	ADC_X1_Left = ADC_X0_Left;

//Spdif left filter total out signed
	Data_spdif_total_left_out = Spdif_Left_low_out + Spdif_Left_mid_out + Spdif_Left_high_out;


//shift left + right in 32bit out unsigned
	Data_spdif_out = 0; //empty spdif out
	Data_spdif_out |=  ((Data_spdif_total_left_out >> 16) << 16) | (((Data_spdif_total_right_out >> 16) & 0xffff) << 0);

//Output to DAC
	//XGpio_DiscreteWrite(&Gpio_DAC, GPio_CHANNEL, (Data_spdif_out));



//Clear the interrupt such that it is no longer pending in the GPIO
	(void)XGpio_InterruptClear(GpioPtr, XPAR_GENERIC_GPIO_1_IP2INTC_IRPT_MASK);
//Enable the interrupt
	XGpio_InterruptEnable(GpioPtr, XPAR_GENERIC_GPIO_1_IP2INTC_IRPT_MASK);



}
```

Figure 4.24: Overview over the interrupt code.

When the interrupt occurs the interrupt function will disable further interrupts, and read the data from Spdif, and DAC ports. When the data has been read it will be sorted out and OR in a signed integer. The OR is necessary because the data from the port is stored in an unsigned, therefore it will be OR in a signed integer to set the sign bit correct, after this it will be shifted down so the LSB is on zero bits place. After the data has been sorted out and stored in a signed integer(x_0), it can be used in the implemented filter difference equations.
When the processed signal leaves the filter algorithm, there will be calculated the energy in the signal. The energy in the signal will be summarised by the mean energy from in the past samples, and the new mean will be saved in the mean energy variable. All mean energy variables from all filters will be compared and a calculation of the gain of each filter will be performed, so the output to the amplifier will compensate for the no-flat to a flat spectrum.This correction of the signal will be real time. The mid-bands bandpass filters are the reference, a proportion between Spdif and ADC energy.
So the low and high-band filters has to have the same proportion mean energy as the mid-band. The calculation of proportions between filters, and gain adjust values, are planned to be located in the infinite while loop. This has not been implemented yet, because of the lack of time given for this project. The main reason to implementing the correction in the infinite loop instead of the interrupt, is because if lack of process time it will not matter if it takes 2 or 3 interrupt cycles before finishing. Another reason is the easy integration of a human interface can be quickly implemented and upgraded, for example when an human controlled equalizer is implemented.
After the calculation and storing of the mean energy, there are on the Spdif side implemented a small gain adjustment, this gain adjustment variable will be interacting with the human equalizer, and the correction interface.
When all algorithms for the filter, energy storing , and equalizing has been done, it is time for adding all the filters together. As there can be seen in the above reduced code all filters are added together(again only mid-band filters are shown, but here is added other filters together with the mid-band) to create a final output to the DAC(output to amplifier). The filter-total will be OR into a unsigned variable. The unsigned variable is a 32 bit variable, capable of having the capacity to both the left and right side channel. The signal will now be send out of the system, and the interrupt is set to enable again, and the program will jump back to the while loop, and wait there until the next audio sample generate an interrupt.

CHAPTER 5

SUMMATION

5.1 Conclusion

The ground idea in this project was to take the assignment of the project proposal like a job, and see how far the development of product will progress. Taken in account of the limit of time of 3 months(2months and 3weeks), the project was a success. However the desired real-time correction of the signal is not implemented.Just when the basic system with the hardware and implementation of the filter algorithms was finished, and the system development was ready to the next step of making and implement the signal correction the time rand out.
But the project is still seen as a success, because of the learned theory combined with the practice implementation, and the basic system seems to have a strong and simple structure.
There is used a lot of hours in this project, and time to time, it seems that this project will never get completed, and it did not totally, but then again it did because of the basic core works so well, and the equalizers can be regulated, and mean energy works.
The Spdif is build from scratch, and a lot of time was spend on it, it is not perfect, but either bad, and the sound is really good when using good matched clocks. There is a little click in the sound, approximately every 2-3 seconds but is very low. There could have been chosen an integrated chip with the Spdif receiver implemented, however there was a lot of practice and theory to gain here, and also very interesting.
The CODEC was more strait forward to implement, but still learning-full, and gave more routine in developing interface and protocol communication. The choice of an FPGA as platform seems to be the right choice. The FPGA is extreamly fast due to its parallelism , and combined with the Microblaze set up as an DSP, and the posibility of implementing coprocessors and parallel cores gives so much speed power, that a FPGA based system will total outperform a DSP processor.

5.2 Perspective

The future perspective of this project could be huge, under development progress there was constantly popping new ideas up. For example there could be developed correction of the left and right channels stereo perspective, so they will be at the same level output. More bands in the equalizer, and analysers could make a dramatically effect of the correction quality, and by implement room correction could lead the sound quality in a positive direction.
Keeping the FPGA platform, and develop as much hardware in VHDL as possible, could lead in making a high performance sound system at a low price. Other things like human interface to communicate and interact with the system, could also be implemented within a low price range. Actually the FPGA and its low price, could make a new product standard on the market. The low price gives the possibility of using more expensive, and higher quality components in the rest of the audio system beside FPGA. Also tacking in account of all the system wires, and analog components exchanged by digital algorithms, could make an high quality audiophile sound system, there normally cost thousand of dollars to entry the lower price end market.

CHAPTER 6

APPENDIX

6.1 Appendix

6.1.1 Toslink Interface

For the interface of the Spdif(Sony/Phillips Digital Interface) there has been chosen Toslink(optical). The receiver is Sharps GP1FAV50RK0F[15] optical receiver module, and has been simple build from the information from the data sheet. The resistors has the values defined by data sheets, $R_{so} = 2.2K\Omega$, $R_{si} = 3.3K\Omega$, beside the data sheet specification there has been mounted a capacitor with the value of 47uF over VCC and GND. The capacitor removed a lot of fail bits, which sounded like loud pops in the sound.

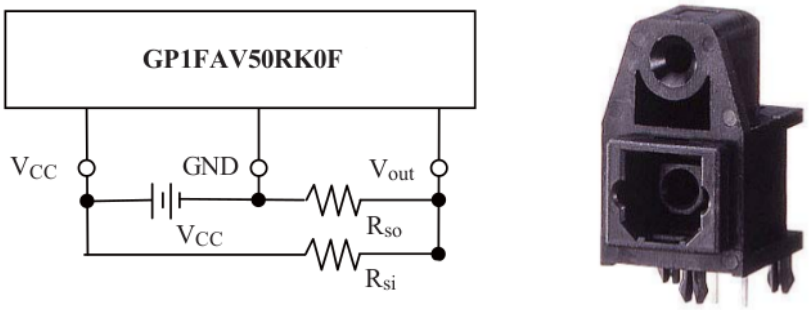

Figure 6.1: Sharp GP1FAV50RK0F optical circuit and receiver[15].

6.1.2 Voltage Splitter for Feedback

The voltage divider for the feedback from the amplifier is build of ratio [10:1]. The reason for the divider is not to saturate the ADC. There is chosen that $R_{tot} = 2K\Omega$, so $R_{tot}/10 = R2$, then $R2 = 2K/10 = 200\Omega$, then $R1 = 9 \cdot R2 = 9 \cdot 200 = 1800\Omega$.

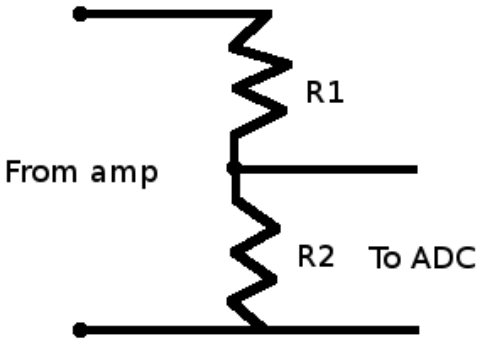

Figure 6.2: Circuit diagram over the voltage divider, for the [10:1] of the amplifier feedback to the ADC.

6.1.3 Modification of Development board

The WM8731 data sheet[5] specify that when the codec is used in line out mode the capacitors of the line out filter has to be 10uF instead of 220uF when using headphones. The expansion board was equipped with 220uF, so it was made for headphones. Below shows a picture of the modification.

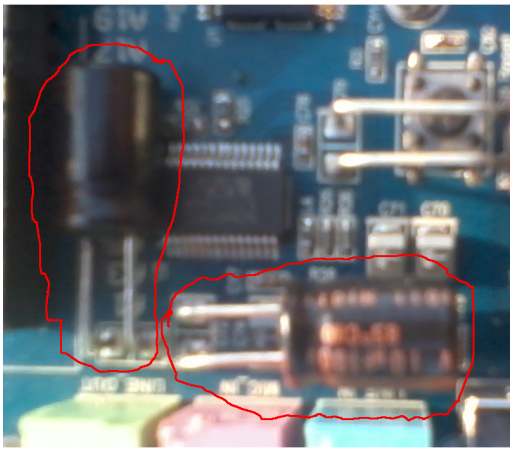

Figure 6.3: Expansion board mod of the 2 capacitors from 220uF to 10uF

6.1.4 Blink Audio Test CD

To test the difference between DAC output, and the output from the amplifier a audio test CD called "Blink Audio Test CD"[16] has been downloaded see source under bibliography. Blink audio test CD consist of the following test sinus with gain of -1db of each frequency. The test frequencies are:

{16, 20, 25, 31.5, 40, 50, 60, 63, 70, 80, 90, 100, 125, 160, 200, 250, 315, 400, 500,
630, 800, 1.25K, 1.6K, 2K, 2.5K, 3.15K, 4K, 5K, 6.3K, 8K, 10K, 12.5K, 16K, 20K} hz

6.1.5 Measurements Of Codec With Tube Amplifier

By using the Blink audio test CD, a measurement of the output of the DAC, and the tube amplifier has been performed, to get feeling of the complete audio system. Below is a table over measured amplitudes:

Hz	**DACL**	**AMPL**	**DACR**	**AMPR**
	$(max,min)VRMS$	$(max,min)VRMS$	$(max,min)VRMS$	$(max,min)VRMS$
16	(1.25,-1.25) 824mV	(3.87,-5.06) 2.83V	(1.37,-1.43) 822mV	(4.06,-3.93) 2.71V
20	(1.25,-1.25) 825mV	(4.06,-5.46) 3.07V	(1.37,-1.37) 816mV	(4.37,-5.31) 3.12V
25	(1.25,-1.25) 830mV	(4.37,-5.78) 3.35V	(1.43,-1.37) 824mV	(4.68,-5.46) 3.37V
31.5	(1.25,-1.25) 835mV	(5.00,-5.93) 3.68V	(1.37,-1.37) 826mV	(5.15,-5.62) 3.66V
40	(1.25,-1.25) 834mV	(5.31,-5.62) 3.80V	(1.43,-1.43) 835mV	(5.46,-5.46) 3.78V
50	(1.25,-1.25) 846mV	(4.84,-5.46) 3.60V	(1.43,-1.37) 840mV	(5.31,-5.31) 3.66V
60	(1.25,-1.25) 840mV	(4.84,-5.46) 3.49V	(1.43,-1.37) 838mV	(5.15,-5.15) 3.57V
63	(1.25,-1.25) 836mV	(4.84,-5.46) 3.52V	(1.25,-1.25) 821mV	(5.15,-5.31) 3.59V
70	(1.25,-1.25) 842mV	(4.84,-5.62) 3.65V	(1.25,-1.25) 828mV	(5.15,-5.31) 3.66V
80	(1.25,-1.25) 835mV	(5.31,-5.78) 3.77V	(1.25,-1.25) 819mV	(5.31,-5.46) 3.74V
90	(1.18,-1.25) 840mV	(5.31,-5.78) 3.89V	(1.25,-1.25) 826mV	(5.46,-5.62) 3.83V
100	(1.25,-1.31) 839mV	(5.46,-5.93) 3.92V	(1.25,-1.25) 822mV	(5.46,-5.62) 3.87V
125	(1.18,-1.31) 838mV	(5.46,-5.93) 3.94V	(1.18,-1.18) 817mV	(5.62,-5.78) 3.89V
160	(1.18,-1.31) 835mV	(5.31,-5.78) 3.82V	(1.25,-1.18) 816mV	(5.46,-5.46) 3.78V
200	(1.18,-1.31) 836mV	(5.00,-5.62) 3.71V	(1.18,-1.18) 818mV	(5.31,-5.31) 3.68V
250	(1.18,-1.31) 835mV	(5.00,-5.46) 3.61V	(1.18,-1.18) 818mV	(5.15,-5.15) 3.59V
315	(1.18,-1.31) 837mV	(4.68,-5.31) 3.52V	(1.18,-1.18) 819mV	(5.00,-5.15) 3.51V
400	(1.12,-1.31) 833mV	(4.68,-5.15) 3.45V	(1.31,-1.37) 819mV	(5.00,-5.00) 3.44V
500	(1.12,-1.31) 828mV	(4.53,-5.15) 3.40V	(1.18,-1.18) 813mV	(4.84,-4.84) 3.39V
630	(1.12,-1.31) 818mV	(4.68,-5.15) 3.41V	(1.12,-1.18) 804mV	(4.84,-4.84) 3.40V
800	(1.06,-1.25) 807mV	(4.53,-5.00) 3.32V	(1.12,-1.18) 796mV	(4.68,-4.84) 3.33V
1250	(1.00,-1.25) 784mV	(4.37,-4.84) 3.27V	(1.12,-1.12) 770mV	(4.68,-4.68) 3.26V
1600	(1.00,-1.18) 750mV	(4.37,-4.68) 3.16V	(1.06,-1.12) 737mV	(4.53,-4.68) 3.17V
2000	(0.937,-1.18) 716mV	(4.06,-4.53) 3.01V	(1.06,-1.06) 705mV	(4.21,-4.37) 2.97V
2500	(0.875,-1.06) 670mV	(3.90,-4.37) 2.84V	(1.00,-0.937) 660mV	(4.06,-4.06) 2.78V
3150	(0.812,-0.937) 613mV	(3.59,-3.90) 2.63V	(0.875,-0.875) 605mV	(3.75,-3.90) 2.55V
4000	(0.750,-0.875) 553mV	(3.12,-3.59) 2.40V	(0.812,-0.812) 548mV	(3.43,-3.59) 2.25V
5000	(0.625,-0.812) 495mV	(2.81,-3.43) 2.14V	(0.750,-0.750) 483mV	(3.12,-3.12) 1.94V
6300	(0.562,-0.750) 440mV	(2.50,-2.96) 1.88V	(0.625,-0.625) 428mV	(2.65,-2.81) 1.62V
8000	(0.500,-0.625) 391mV	(2.03,-2.65) 1.65V	(0.562,-0.562) 382mV	(2.34,-2.34) 1.33V
10000	(0.437,-0.562) 352mV	(1.87,-2.34) 1.47V	(0.562,-0.500) 340mV	(2.03,-2.18) 1.15V
12500	(0.375,-0.562) 325mV	(1.71,-2.18) 1.34V	(0.500,-0.500) 310mV	(1.87,-2.03) 1.06V
16000	(0.375,-0.562) 302mV	(1.56,-2.03) 1.24V	(0.437,-0.437) 290mV	(1.71,-1.87) 1.10V
20000	(0.375,-1.06) 316mV	(1.40,-1.87) 1.14V	(0.437,-0.875) 290mV	(1.71,-1.87) 1.11V

Table 6.1: This table shows the measured frequencies with blink audio CD, of the tube amplifier trough the CODEC WM8731

6.1.6 Test Microblaze Speed For Filters

For a test of the Microblaze[6] of how many filters it can handle, a small test has been made. The test is created by implement a filter inside the infinite loop(while{}), and a counter variable "Counter". "Counter" will increase by 1 every time the filter has completed, the filter will then run again, and again, until an interrupt occur. When interrupt occur it will write the counter variable "Counter" to the DAC port, and reset "Counter" to zero. Below is the test code.

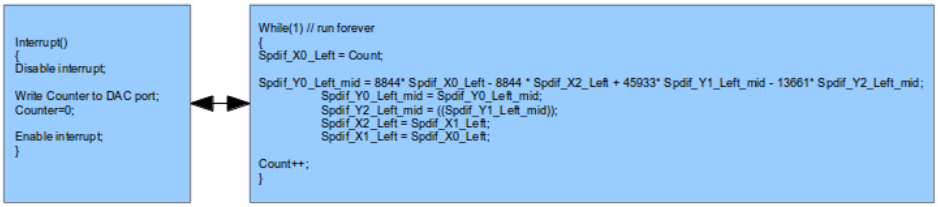

Figure 6.4: Overview filter test to check the speed of Microblaze.

With an digital analyser the DAC value can be read. The result was taken when the lowest value over a few seconds of analyse(1 second corresponds to 44100 interrupts), and the value was 38. So the Microblaze is fast enough for this project, and can run about 38 second order band-pass filters. This project requires 2x3 filters for Spdif(left/right), and 2x3 filters for the ADC(left/right), so total 12 filters, and taken in account that not all filters will be 2 order band-pass filters, plus all other algorithms. If more speed is needed, there could be made coprocessors or multi kernels.

6.1.7 Test Of Hole System

There has been performed a small test of the system. The system Spdif input works, however there are low clicks every 2-3 seconds in the sound, but is not annoying, and the sound quality is really good. The CODECs DAC, and ADC works perfect, there has been made a test where an audio signal has been input on the ADC, and output from the DAC, and the result sounded really good, but a little hum. The equalizers can be adjusted up, and down. The correction has not been implemented, because of the time rand out, but all energy mean values comes out correct. The basic system build up seems strong, simple, and very stable, So the system is ready for the signal correction implementation.

BIBLIOGRAPHY

[1] Illinois.edu Frequency limits for octave bands $https://courses.physics.illinois.edu/phys406/Lab_Handouts/Octave_Bands.pdf$ (2014).

[2] Book, H. Spdif $http://www.hardwarebook.info/$ (2014).

[3] Xilinx Spartan-6 family overview $http://www.xilinx.com/support/documentation/data_sheets/ds160.pdf$ (2011).

[4] $dong_dt$ Xilinx spartan6+extension boards $http://www.ebay.com/usr/dong_dt$ (2014).

[5] Wolfson Portable internet audio codec with headphone driver and programmable sample rates $http://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf$ (2014).

[6] Xilinx Logicore ip microblaze micro controller system (v1.1) $http://www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ds865_microblaze_mcs.pdf$.

[7] Wolfcrow Grandma moon speaks: Pulse code modulation (pcm) $http://wolfcrow.com/blog/grandma-moon-speaks-pulse-code-modulation-pcm/$ (2014).

[8] Devices, A. The relationship of data word size to dynamic range and signal quality in digital audio processing applications $http://www.analog.com/en/content/relationship_data_word_size_dynamic_range/fca.html$ (2014).

[9] Altium Wishbone i2s streaming audio $http://designcontent.live.altium.com/PluginDetail/CCC_I2S$ (2014).

[10] Quick2Wire I2c and spi $http://quick2wire.com/articles/i2c-and-spi/$ (2014).

[11] Audio, T. crash course on digital audio interfaces $http://www.tnt-audio.com/clinica/jitter1_e.html$ (2014).

[12] Wikipedia Octave $http://en.wikipedia.org/wiki/Octave$ (2014).

[13] IEC Iec1260 $IEC1260page47$ (1967).

[14] Owsinski, B. A description of the audio frequency bands $http://bobbyowsinski.blogspot.dk/2012/06/description-of-audio-frequency-bands.html$ (2014).

[15] Sharp Gp1fav50rk0f fiber optic receiver $http://www.sharp.co.jp/products/device/doc/opto/gp1fav50rk_e.pdf$ (2014).

[16] Blink Bink audio test cd available online $http://www.livesoundint.com/lab/lab/messages/archive5/41723.html$ (2003).