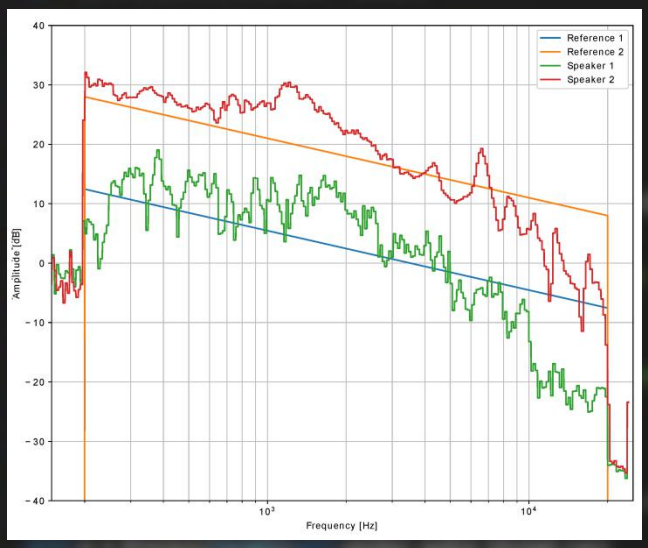
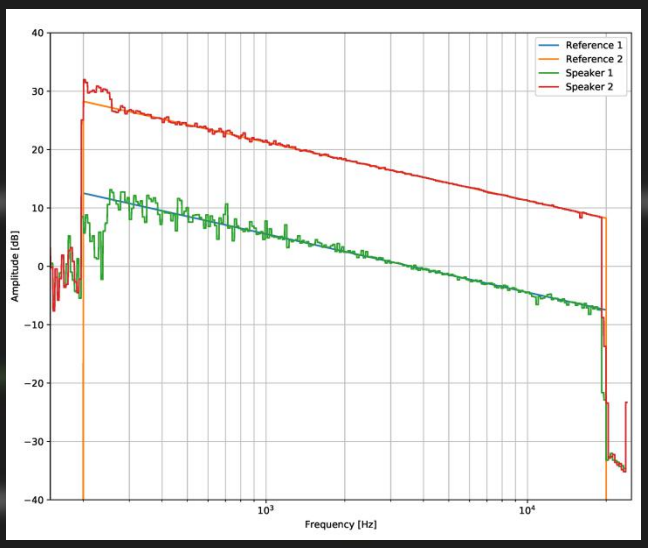
EQ en Delay Optimalisatie

Research & hardware-concept
voor automatische EQ en Delay optimalisatie





In opdracht van:

embedded ACOUSTICS

en

DE HAAGSE HOGESCHOOL

Auteur: Martijn van Essen
Versie: vA04
Datum: 10-10-2019

EQ en Delay Optimalisatie

Research & hardware-concept voor automatische EQ en Delay optimalisatie

Scriptie | Afstudeerstage
Embedded Acoustics

Algemeen:

Locatie:	De Haagse Hogeschool, Delft
Datum:	10-10-2019
Versie:	vA04
Auteur	Martijn van Essen

Opleidingsinformatie:

Onderwijsinstelling:	De Haagse Hogeschool
Opleiding:	Elektrotechniek
Code:	E-AFSTUD1-16
Studentnummer:	15086135
E-mail:	martijn.vanessen@hotmail.com
Eerste Begeleider:	Patrick Morley p.morley@hhs.nl
Tweede Begeleider:	Gerben Hoogendorp g.hoogendorp@hhs.nl

Bedrijf:

Opdrachtgever:	Embedded Acoustics
Stagebegeleider	Jan Verhave Jan@embeddedacoustics.com

Versie Historie

Versie Historie			
Versie	Datum	Wijzigingen	Auteur
vA01	08-04-2019	Eerste concept voor bespreking met begeleider.	M.T. van Essen
vA02	21-05-2019	Tweede concept voor feedback	M.T. van Essen
vA03	28-05-2019	Verwerking laatste feedback	M.T. van Essen
vA04	10-10-2019	Geredigeerd voor openbare publicatie	M.T. van Essen

Voorwoord

Dit verslag is geschreven in het kader van mijn afstudeerstage bij Embedded Acoustics. Deze stage is uitgevoerd in opdracht van de Haagse Hogeschool te Delft. De stage heeft plaatsgevonden in de periode van 4 februari tot en met 31 mei 2019. De activiteiten gedurende deze stage zijn verwerkt tot dit informatieve rapport.

Dit verslag is bedoeld als naslagwerk voor een eventueel vervolg van dit project binnen Embedded Acoustics. Hoewel dit verslag openbaar is, blijft van kracht dat de binnen het project opgedane kennis, algoritmiek en toepassingen hiervan, intellectueel eigendom blijven van Embedded Acoustics BV.

Verder dient dit verslag ook als basis voor de verslaglegging van de stage voor de Haagse Hogeschool.

Graag wil ik hierbij mijn begeleider, Jan Verhave, bedanken voor zijn begeleiding, hulp en leerzame uitleg tijdens de stage. Daarnaast wil ik de rest van het bedrijf bedanken voor de gezellige sfeer waarin ik deze stageopdracht heb mogen uitvoeren.

Verder wil ik ook mijn begeleidend docent, Patrick Morley, bedanken voor zijn begeleiding en feedback gedurende deze stage.

Ik vertrouw er op dat dit eindrapport inzicht biedt in het verloop van de stage en de vergaarde kennis en bevindingen vastlegt.

Delft, 28 mei 2019
Martijn van Essen
Student Elektrotechniek - cursusjaar 2018-2019
De Haagse Hogeschool, vestiging Delft.

Samenvatting

Dit verslag behandeld het research- en ontwikkelproces van het automatische EQ en Delay optimalisatie algoritme en het bijbehorende hardware advies. Dit document heeft als doel om informatie te verschaffen over het tot stand komen van het algoritme en het hardware advies.

Het doel van dit systeem is om door middel van een meting, automatisch de afwijking in frequentie-overdracht en vertragingstijden van een luidspreker installatie te detecteren en te corrigeren. Deze functionaliteit kan vervolgens gebruikt worden in diverse audio gerelateerde toepassingen.

Voor de ontwikkeling is allereerst gekeken naar de verschillende manieren om afwijkingen in de frequentie-overdracht te meten en hieruit een filter te genereren. Meerdere methoden zijn hiervoor gesimuleerd. Hieruit volgt dat het bepalen van een invers filter aan de hand FFTs en deze omzetten naar een FIR, de beste resultaten levert en het meest flexibel is.

Ook zijn hierbij meerdere testsignalen geëvalueerd. Hieruit volgt dat de gedetermineerde roze ruis beschikt over de gewenste eigenschappen.

Voor de meting van de vertraging wordt ook de gedetermineerde roze ruis gebruikt. De correlatie van dit signaal, levert een duidelijke piek. Met behulp van deze piek kan de vertraging tussen luidsprekers bepaald worden. Uit simulaties blijkt dat deze methode ook werkt onder ongunstige omstandigheden.

De methodes voor het meten van frequentie-overdracht en vertraging zijn vervolgens verwerkt in een enkele meting. Door het wisselen van frequentiebanden wordt het mogelijk om de frequentie-overdracht en vertraging van meerdere luidsprekers in één meting vast te stellen.

Dit geheel is verwerkt in een Python-applicatie. Met deze applicatie zijn de mogelijkheden van deze methodes getest. Hieruit volgt dat de applicatie de frequentie afwijking van de geteste speakers tot 360 keer kleiner kan maken en de vertraging tot één sample nauwkeurig kan bepalen.

Vervolgens wordt naar hardware gekeken. Hieruit volgt dat voor digitale uitbreiding van de audio, het beste een Dante Brooklyn II module gebruikt kan worden. Met betrekking tot filteren is het advies om 16 tot 24 bits fixed point variabelen te gebruiken met een samplefrequentie van 48 kHz. Dit kan gecombineerd worden met een FPGA met geïntegreerde hardcore processor en DSP elementen.

De conclusie die hieruit volgt is dat een werkend algoritme is ontwikkeld dat EQ en Delay kan corrigeren. Dit algoritme kan in combinatie met het hardware advies dienen als basis voor de verdere voortzetting van dit project binnen Embedded Acoustics.

Summary

This report covers the research and design process of the automatic EQ and Delay optimisation algorithm and the hardware advice associated with it. This document serves the purpose of giving information about the the design of the algorithm and to give a hardware advice.

The main goal of this system is to use measurements in order to automatically correct alterations in the frequency content of a signal, and to correct for added delay which a speaker installation introduces. These functions can be used in various audio related applications

First of all, different methods have been explored for detecting alterations in the frequency content of a signal. From this, an inverse filter is generated. Multiple methods have been simulated. This resulted in an FFT to calculate an inverse filter and transforming this to a FIR as being the best, and most flexible, method.

Multiple test-signals have been evaluated. This resulted in a determined pink noise, which satisfies all needs.

The determined pink noise is also used for measuring the delay. The correlation of this signal, results in a distinguishable peak. Using this peak, the delay between speakers can be determined. Simulations showed that this method worked well, even in sub-optimal conditions.

The methods for determining an inverse filter and determining the delay, are then combined in a single measurement. By also adding changing frequency bands, multiple speakers can be optimised at once using this method.

All of this is then converted into a Python-application. Using this application, the performance of the algorithm is tested. This resulted in the application being able to optimise the used speaker to reduce the frequency error by a factor of 360. The delay measurement was accurate up to one sample.

After this, the hardware has been researched. This resulted in the advice to use a Dante Brooklyn II module for digitally expanding the audio channels. It is also adviced to use 16 to 24 bits fixed point variables with a sample frequency of 48 kHz. This can be combined using an FPGA with integrated hardcore processor and DSP-slices.

The conclusion is that this process resulted in a working algorithm which is capable of correcting EQ and delay deviations. This algorithm can be combined with the hardware advice to further develop this project within Embedded Acoustics.

Inhoudsopgave

Lijst van figuren

Lijst van tabellen

Afkortingen en Begrippen

Afkorting	Betekenis
ADC	Analoog Digitaal Converter
ALU	Arithmatic and Logic Unit
DSP	Digital Signal Processing/Processor
EQ	Equalisatie (van geluid)
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
FPGA	Field Programmable Gate Array
GMAC	Giga Multiply Accumulate
GUI	Graphical User Interface
I/O	Input / Output
IIR	Infinite Impulse Response
MSE	Mean Square Error

Begrip	Betekenis
Callback	Functie die aangeroepen wordt als reactie op een event binnen software
Inter-process communication	Communicatie tussen threads
Multithreading	Meerdere taken parallel uitvoeren
Python	Programmeertaal
Thread	Losse taak binnen een programma

1. Inleiding

Vanuit ervaringen uit een eerder project, is bij Embedded Acoustics de wens ontstaan om een nieuwe ontwikkeling te starten op basis van het eerdere project. Hierbij gaat het om het automatisch corrigeren van de EQ en onderlinge vertragingstijden van luidsprekers.

In het kader van deze nieuwe ontwikkeling is onderzoek gedaan om voor deze correcties een algoritme te ontwikkelen. Daarnaast is gekeken naar de benodigde hardware om deze filters toe te passen.

Binnen dit verslag wordt allereerst de achtergrond van het project behandeld. Hierin wordt de basis van dit project en de doelstelling toegelicht. Dit wordt gevolgd door een beschrijving van de uitgevoerde onderzoeken in hoofdstuk 3. Hierin komt allereerst de concept werking aan bod, gevolgd door simulaties van verschillende oplossingsstrategieën.

Hierna wordt in hoofdstuk 4 de implementatie van het algoritme behandeld. Allereerst komen hierbij een aantal overwegingen aan bod die bij de implementatie van toepassing zijn. Dit wordt gevolgd door een overzicht van de gemaakte Python software.

De Python applicatie wordt vervolgens gevalideerd in hoofdstuk 5. Hierbij wordt aan de hand van een meetopstelling getest welke verbeteringen behaald kunnen worden door middel van deze applicatie.

Aan de hand van onderzochte concepten is vervolgens gekeken naar de hardware. Dit is beschreven in hoofdstuk 6. Hierin komen de benodigde hardware en mogelijkheden voor het digitaal uitbreiden van audio in- en uitgangen aan bod.

Dit alles wordt gevolgd door een conclusie van dit onderzoek en de uitgevoerde ontwikkeling, samen met aanbevelingen voor de voortzetting van dit project.

2. Achtergrond en Doelstelling

In dit hoofdstuk worden de achtergrond, doelstelling en verdere context van het project behandelt.

2.1. Achtergrond

Rond 2012 is Embedded Acoustics begonnen met de ontwikkeling van een 19" audiosignaalprocessor. Vanuit deze ontwikkeling zijn nieuwe ideeën voortgekomen over de mogelijke toepassingen hiervan. Hieruit is de vraag ontstaan om een toepassing te ontwikkelen voor het automatische meten van overdracht en impulsresponsie van een audioinstallatie en hieruit volgend de audio te compenseren door middel van EQ-Settings en/of FIR-filters.

Het project van Embedded Acoustics om audio-processing bord uit te breiden, kan hierbij onderverdeeld worden in twee primaire activiteiten:

- Het implementeren en testen van nieuwe EQ-Algoritmen voor de automatische EQ of FIR Instellingen
- Het vernieuwen van het ondertussen achterhaalde design van de audiosignaalprocessor

2.2. Doelstelling

Vanuit deze activiteiten zijn de volgende doelen gesteld binnen deze stage:

- Door middel van de aanwezige apparatuur experimenteren met algoritmen voor automatische EQ
- Vanuit de uitgevoerde testen een concept auto-EQ algoritme creëren
- Inventariseren van de eisen die de toepassing stelt aan de hardware en firmware
- Blokschema ontwerp voor de nieuwe audiosignaalprocessor
- Opstellen van functionele specificaties en een voorlopige technische specificatie voor een nieuwe versie van de audiosignaalprocessor
- Vergelijken van de hardware en firmware eisen met de huidige specificaties van de audiosignaalprocessor
- Vergelijken van de vereisten van het nieuwe EQ-algoritme met de nieuw-opgestelde specificaties
- *(optioneel)* Implementeren van het algoritme op de bestaande audiosignaalprocessor of andere hardware
- *(optioneel)* Herzien van schema's en eventuele PCB layout van de audiosignaalprocessor

Verdere informatie over de achtergrond, doelstelling en eisen van dit project, zijn te raadplegen in het Plan van Aanpak [1].

2.3. Toepassingen

Het toepassingsgebied van dit project zal zijn voor audio installaties die de volgende eigenschappen hebben:

- Grote onderlinge verschillen in de looptijd van geluid
- Sub-optimale geluidsoverdracht door afwijkingen van luidsprekers in het frequentiespectrum
- Een belang bij het verbeteren van de bovengenoemde factoren

Het doel is dat dit project deze factoren kan corrigeren om zo tot een optimale overdracht te komen binnen een audio installatie.

3. Research & Operationele concepten

In dit hoofdstuk is het onderzoek voor een EQ en Delay algoritme uitgewerkt. Daarnaast worden de achterliggende concepten voor het algoritme toegelicht.

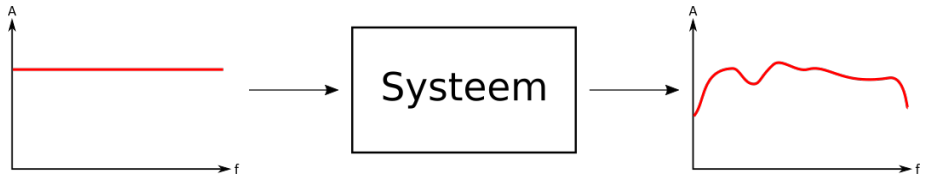
3.1. Concept werking EQ

De basis van de EQ-optimalisatie bestaat uit het corrigeren van de fouten die worden geïntroduceerd. Deze fouten ontstaan door de overdracht van de systemen die tussen de audio-bron en de luisteraar aanwezig zijn. Hierbij gaat het voornamelijk om de veranderingen in het frequentiespectrum van het signaal die veroorzaakt worden door de gebruikte luidspreker. Een voorbeeld hiervan is weergegeven in figuur 3.1. Hier wordt een signaal met een volledig vlak frequentiespectrum verstoord door het systeem. Hierdoor is het frequentiespectrum van het uitgaande signaal niet meer vlak.

Het doel van de EQ optimalisatie zal in eerste instantie zijn om deze overdracht zo goed mogelijk te corrigeren naar een vlakke, of een andere gedefinieerde, overdracht.

Voor deze optimalisatie zullen ook grenzen gesteld moeten worden. Een luidspreker is namelijk niet in staat om alle frequenties te produceren. Dit gaat voornamelijk om de laagste en hoogste frequenties die nog wel hoorbaar zijn voor mensen (20 Hz - 20 kHz). Het is dus ook niet relevant om deze frequenties hard te versterken in het filter, omdat de luidspreker ze toch niet kan produceren.

Daarnaast kunnen de correcties binnen het hoorbare gebied ook niet onbeperkt toegepast worden. Hierbij zullen namelijk vervormingen optreden. De mate van correctie die mogelijk is zonder vervorming is afhankelijk van de versterker en luidspreker combinatie.



Figuur 3.1: Veranderingen in het frequentiedomein door systemen

Het concept zoals hierboven beschreven, is van toepassingen op alle mogelijke optimalisatie-oplossingen. De verschillende mogelijkheden om deze optimalisaties uit te voeren, worden verder besproken in hoofdstuk 3.2.

3.2. EQ Filter berekeningen

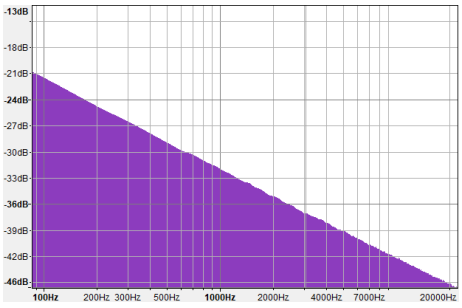
In dit onderdeel worden verschillende methodes voor het berekenen van de filters nader bekeken en toegelicht.

3.2.1. Tools en informatie

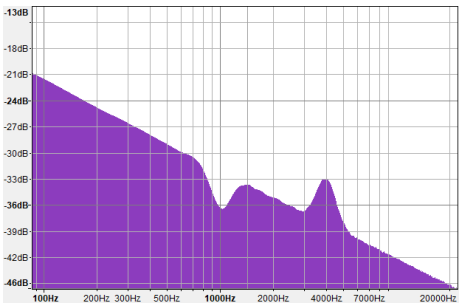
Voor het berekenen en simuleren van de filters wordt gebruik gemaakt van GNU Octave [2].

Voor de simulaties is met behulp van Audacity [3] een roze ruis signaal gegenereerd. Door middel van de Equalizer tools binnen Audacity zijn hier handmatig 'fouten' in aangebracht.

Het frequentie spectrum van het signaal dat voor de meeste testen gebruikt is, is weergegeven in figuur 3.2 en 3.3. Hierbij is de laatste de 'verstoorde' versie van het signaal. Hierin is met de Equalizer tool een dal van 7 dB toegevoegd bij 1 kHz en een piek van 7 dB bij 4 kHz. Vanuit het frequentiespectrum is zichtbaar dat de daadwerkelijke waarden rond de 6 dB liggen.



Figuur 3.2: Frequentiespectrum roze ruis



Figuur 3.3: Frequentiespectrum roze ruis met verstoringen

Om iets te kunnen zeggen over de kwaliteit van het filter, moeten de onderlinge signalen vergeleken worden. Dit zal gebeuren op basis van de Mean Squared Error (MSE). Met deze maat worden de onderlinge verschillen tussen het originele en het verstoorde signaal beoordeeld. Deze error zal worden berekend op basis van de absolute waarden van het signaal in het frequentiedomein. De waarden worden geschaald aan de hand van de FFT grootte.

Het aangepaste signaal heeft ten opzicht van het originele signaal een error van $6,70 \cdot 10^{-4}$.

Voor alle filterberekeningen, tenzij anders vermeld, is een Sample-grootte van 65536 samples gehanteerd (~1,5 seconden). Wanneer van toepassing is een FIR-filter berekend met een realistische grootte van 512 coëfficiënten, met uitzondering van de test voor verschillende filterlengtes.

3.2.2. Transferfunctie in het frequentiedomein

De basis van het probleem, zoals toegelicht in hoofdstuk 3.1, kan wiskundig worden uitgedrukt als de convolutie van het ingangssignaal $x(t)$ met de impulsreponse $h(t)$. Dit is weergegeven in vergelijking 3.1.

$$y(t) = h(t) * x(t) \tag{3.1}$$

In deze uitdrukking is $y(t)$ het uitgangssignaal, $x(t)$ het ingangssignaal en $h(t)$ de vervorming in de frequentie-overdracht, zoals geïntroduceerd door het systeem. Hierbij zijn afwijkingen als gevolg van reken-onnauwkeurigheden en effecten van verstoringen tijdens de meting niet meegenomen.

Het doel is hierbij dus om de term $h(t)$ te elimineren zodat de ideale situatie $y(t) = x(t)$ ontstaat. Dit is gemakkelijk te realiseren vanuit het frequentiedomein, doordat de convolutie hier een vermenigvuldiging wordt. Dit is te zien in vergelijking 3.2.

$$y(t) = h(t) * x(t) \xrightarrow{\mathscr{F}} Y(\omega) = H(\omega) \cdot X(\omega) \tag{3.2}$$

Vanuit deze vergelijking is $H(\omega)$ gemakkelijk weg te werkenm, zoals te zien in vergelijking 3.3.

$$Y(\omega) = H(\omega) \cdot \frac{1}{H(\omega)} \cdot X(\omega) = X(\omega) \tag{3.3}$$

Voordat dit mogelijk is, moet het systeem $H(\omega)$ eerst geïdentificeerd worden. Door metingen uit te voeren met vooraf gedefinieerde signalen en hierbij de uitgaande audio te meten, zijn $y(t)$ en $x(t)$ bekend. Verschillende mogelijkheden voor meetsignalen zullen verder worden behandeld in hoofdstuk 3.4.

Door de gemeten signalen te transformeren naar het frequentiedomein, kan vervolgens de overdracht $H(\omega)$ berekend worden. Dit is weergegeven in vergelijking 3.4.

$$H(\omega) = \frac{Y(\omega)}{X(\omega)} \tag{3.4}$$

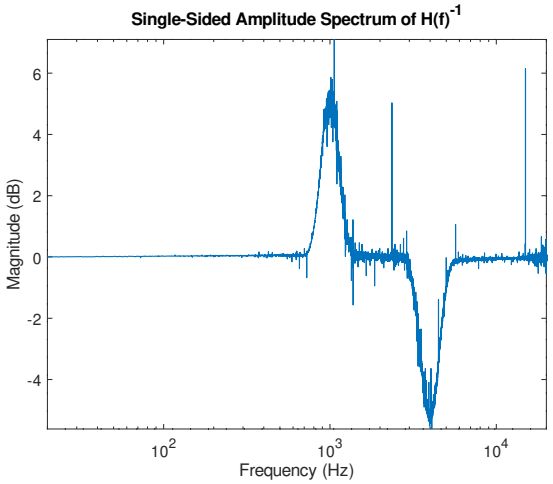
De verkregen $H(\omega)$ kan vervolgens gebruikt worden om de afwijkingen in het frequentiespectrum te corrigeren.

Simulaties

De bovengenoemde methode is gerealiseerd in Octave met behulp van codefragment 3.1. Hieruit resulteert de transferfunctie $H(f)$. De inverse van deze transferfunctie is weergegeven in 3.4.

```
1 %Take Fourier transform of input and
      output
2 Y = fft(outputaudio(1:sampleSize));
3 X = fft(inputaudio(1:sampleSize));
4 %Calculate H
5 H = Y./X;
6 %Reconstruct X using Y and H
7 OUT = Y .* H.^-1;
8 %Convert OUT back to the frequency
      domain
9 out = real(ifft(OUT));
```

Codefragment 3.1: Code voor Filteren in het Frequentiedomein



Figuur 3.4: Transferfunctie invers-filter

Bij deze transferfunctie is een duidelijke piek bij 1 kHz zichtbaar en een dal bij 4kHz. De grootte hiervan komt overeen met de aangebrachte veranderingen.

Ook is het zichtbaar dat er veel 'ruis' op het filter zit. Dit wordt veroorzaakt door meerdere factoren. De belangrijkste hierbij is de spectrale dichtheid van het testsignaal. Het testsignaal is zelf ook een ruissignaal, dat fluctueert in de tijd en als functie van de frequentie. Dit resulteert in de fluctuatie die zichtbaar is in de deling van Y en X.

Deze transferfunctie is vervolgens toegepast op $Y(f)$ om $X(f)$ te reconstrueren. Bij de inverse FFT van het signaal ontstaan zeer kleine imaginaire delen(10^{-16}). Dit wordt veroorzaakt door afrondingsfouten binnen Octave en deze kunnen dus genegeerd worden.

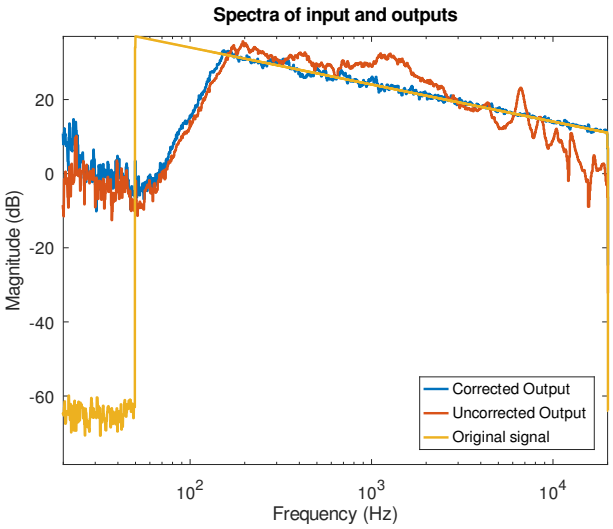
Van het resulterende signaal is vervolgens de error berekend . Dit resulteert in een error van $5 \cdot 10^{-33}$. De error is hiermee met een factor van $1,3 \cdot 10^{29}$ verbeterd ten opzichte van het ongecorrigeerde signaal.

Hoewel deze methode zeer goede resultaten levert, is het met het oog op de toepassing geen praktische oplossing. Doordat er gefilterd wordt in het frequentiedomein, moet bij implementatie alle audio geconverteerd worden naar het frequentiedomein en weer terug. Hoewel hier mogelijkheden voor zijn, is een implementatie voor meerdere kanalen op een FPGA zeer complex. Voor het gebruik in een FPGA is een FIR-filter een betere oplossing, omdat hier minder complexe wiskundige operaties voor nodig zijn.

Metingen

Om het berekenen van de transferfunctie in het frequentiedomein te testen, zijn metingen uitgevoerd. De meetopstelling die hierbij gebruikt is, wordt verder toegelicht in hoofdstuk 5.1. Bij deze metingen is gebruik gemaakt van een gedetermineerd roze ruis signaal. Hoewel dit signaal klink als ruis, zijn zowel de amplitude als fase gedefinieerd. Dit is verder toegelicht in hoofdstuk 3.4.3.

Het test-signaal wordt vervolgens door de luidsprekers afgespeeld en opgenomen door een meet-microfoon. Dit ingangs- en uitgangssignaal worden vervolgens omgezet tot een invers filter in het frequentiedomein aan de hand van de hierboven beschreven methode. Deze filters zijn bewerkt zodat er onder de 150 Hz en boven de 20 kHz niet geprobeerd wordt te filteren. Voor het genereren van het filter is een FFT lengte gebruikt van 480.000 samples.



Figuur 3.5: Gemeten ongecorrigeerde en gecorrigeerde output

Na het toepassen van dit invers filter op het ingangssignaal is het resultaat opnieuw gemeten. Dit resulteert in de frequentie-responses, zoals weergegeven in figuur 3.5. Hierin is te zien dat het gecorrigeerde signaal de lijn van de input volgt. Hierbij treden kleine schommelingen van $\pm 1,5$ dB op. Ook is te zien dat het filter geen invloed uitoefent op de frequenties onder de 150 Hz.

3.2.3. Transferfunctie in het tijddomein

Het is ook mogelijk om de inverse transferfunctie, die in onderdeel 3.2.2 berekend is, om te zetten naar het tijddomein. Dit resulteert in de impulse-response van het filter die vervolgens als FIR-filter toegepast kan worden om het signaal te filteren.

Simulaties

In codefragment 3.2 is de code weergegeven voor het converteren van de transferfunctie naar het tijddomein. Hoewel in eerste instantie gedacht kan worden dat het filter na de inverse fourier-transformatie direct bruikbaar is, is dit helaas niet mogelijk. Doordat in dit geval gebruik is gemaakt van discrete signalen, zal er aliassing optreden. Om deze reden wordt de verkregen transferfunctie geshift, getrunceert en gewindowed.

Het filter met aliassing is weergegeven in figuur 3.6. Het filter met shifting, truncering en windowing is weergegeven in figuur 3.7

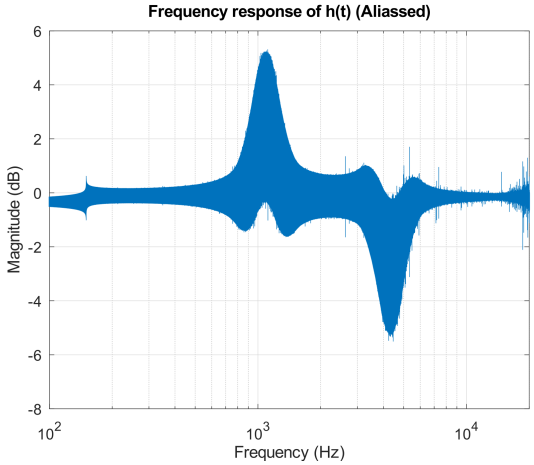
Hierin is te zien dat het filter een piek heeft bij 1 kHz en een dal bij 4 kHz. Beide met een amplitude van 5,1 dB. Dit filter is vervolgens toegepast op het uitgangssignaal om het ingangssignaal te reconstrueren. Ook bevat dit filter weinig 'ruis', dat te zien is in de geleidelijke curve van het filter.

Dit resulteert in een error van $1,42 \cdot 10^{-7}$. De error is met een factor van $4,93 \cdot 10^{3}$ verbeterd ten opzichte van het ongecorrigeerde signaal. De error is hierbij ook afhankelijk van de gebruikte windowing functie.

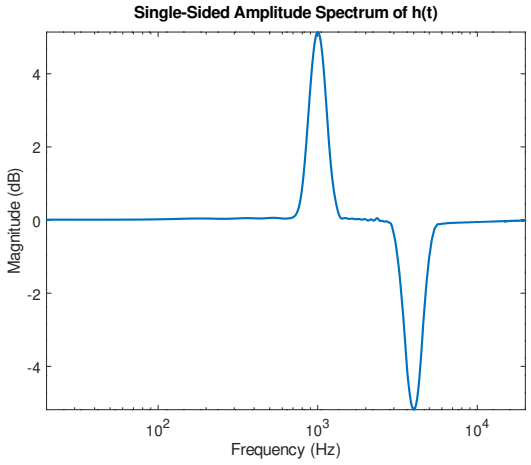
Aangezien deze techniek resulteert in gebruikelijke FIR-filters met 512 coëfficiënten, is het mogelijk om deze te implementeren in een FPGA. Wel moet er hierbij in het systeem een processor met voldoende rekenkracht aanwezig zijn om de berekening van het filter uit te voeren. De benodigde rekenkracht moet bij het implementeren onderzocht worden.

```
%Convert H back to timedomain
h = real((ifft(1./H)));
%Shift, truncate and window
M = numCoefficients;
T = zeros(1,sampleSize);
for I = 1:sampleSize
  index = I + M/2;
  if index > sampleSize
    index = index - sampleSize;
  end
  T(index) = h(I);
end
for I = 1:sampleSize
  h(I) = T(I);
end
for I = 1:sampleSize
  if I <= M
    %Rectangular window
    h(I) = h(I);
  end
  if I > M
    h(I) = 0;
  end
end
```

Codefragment 3.2: Filteren in het tijddomein



Figuur 3.6: Frequentie respons aliassed filter



Figuur 3.7: Frequentie respons filter zonder aliassing

Metingen

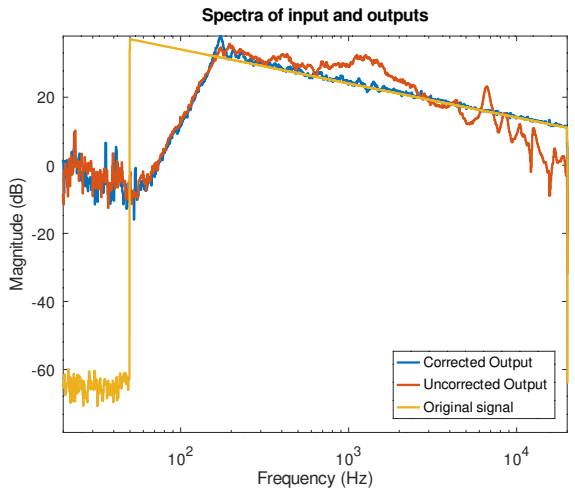
Voor het meten van deze filters zijn de zelfde condities gebruikt als in de voorgaande meting. Hierbij is alleen de manier van het berekenen van het filter vervangen. Hieruit resulteert een FIR-filter met een inverse response. Hier wordt verder aan gerefereerd als 'Invers FIR-filter'.

Dit invers filter is vervolgens op het ingangssignaal toegepast. Het resultaat hiervan is opnieuw afgespeeld en gemeten. De resultaten voor een filter van 4096 taps zijn weergegeven in figuur 3.8.

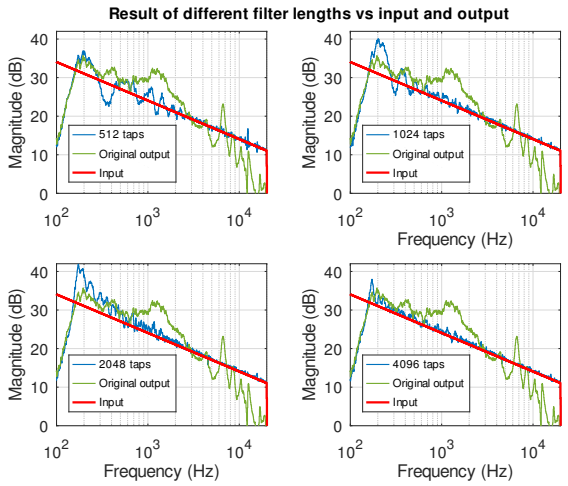
In dit figuur is te zien dat met name in de hogere frequenties het frequentiespectrum van het signaal goed gecompenseerd wordt. Aangezien de frequenties onder de 150 Hz niet gefilterd zijn, is het spectrum hier zoals verwacht gelijk gebleven. Bij 200 Hz is wel een overshoot zichtbaar ten opzichte van het originele spectrum. Verder is te zien dat het nieuwe spectrum de lijn van het originele signaal volgt, maar wel om deze lijn heen schommelt.

Ook is bij deze meting getest wat de invloed is van het aantal filtertaps op de kwaliteit van het resulterende frequentiespectrum. Hiervoor zijn filters gegenereerd met 512, 1024, 2048 en 4096 taps. Het resultaat hiervan is weergegeven in figuur 3.9. Een vergrote versie van deze afbeelding is te raadplegen in *Bijlage A: Invloed van het aantal filtertaps op de filterkwaliteit*. Hierin is te zien dat de filters met minder taps ook minder controle hebben over de lage frequenties.

Met 512 filtertaps is de resolutie van het filter ook slechts 100 Hz. In het lage gebied treden hierdoor grotere schommelingen op. Boven de 4 kHz worden de verschillen tussen de filters klein.



Figuur 3.8: Gemeten ongecorrigeerde en gecorrigeerde output(4096 taps)



Figuur 3.9: Vergelijking verschillende filter lengtes

3.2.4. Transferfunctie regulatie door Octaafbanden

Om meer controle te krijgen over hoeveel detail het uiteindelijke filter heeft ten aanzien van ons gehoor, kan gebruik gemaakt worden van een opdeling in relatieve octaafbanden. Relatieve octaafbanden hebben een bandbreedte die gerelateerd is aan de centrale frequentie van de band (f_c). Hoe hoger de centrale frequentie, hoe groter de bandbreedte. Voor octaafbanden is deze verhouding $\frac{f_c}{\sqrt{2}}$.

Deze octaafbanden zijn weer onder te verdelen in smallere octaafbanden, elk met een eigen relatieve bandbreedte. Hier wordt aan gerefereerd als $\frac{1}{N}$ octaafbanden. Voor deze applicatie kan een waarde worden gegeven om het spectrum op te delen in $\frac{1}{N}$ octaafbanden.

Berekening

Wanneer de octaafband-frequenties berekend zijn, kan het frequentiespectrum hierin opgedeeld worden. Hierbij wordt de energie van het ingangs- en uitgangssignaal binnen de frequentiebanden berekend en vervolgens aan deze frequentieband toegewezen.

Vervolgens worden deze twee verdeelde signalen door elkaar gedeeld waaruit de frequentierespons van het filter op basis van energie volgt. Dit wordt vervolgens verder verwerkt om er een impulse-response uit te laten volgen.

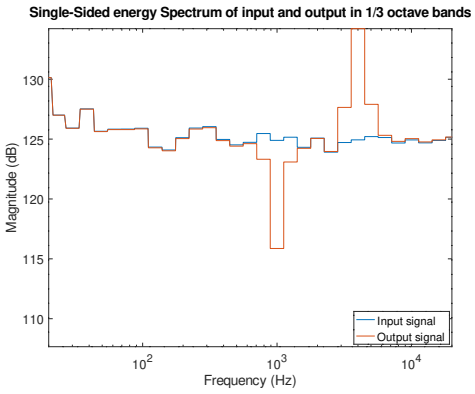
Een voorbeeld van de signalen opgedeeld in octaafbanden is weergegeven is figuur 3.10. Hierin zijn duidelijk de stappen in het frequentiespectrum te zien. In dit geval is gebruik gemaakt van $\frac{1}{3}$ octaafbanden. Hierdoor is elke 'bin' vrij groot.

Elk van deze secties zou apart geregeld kunnen worden, wanneer dat gewenst is. Deze methode maakt het dus gemakkelijker om bepaalde veranderingen in delen van het signaal toe te passen. Hierbij gaat het met name om het plaatsen van restricties voor de equalizer of het handmatig bijstellen hiervan. Ook zou dit kunnen helpen bij de ruisgevoeligheid tijdens het filterontwerp.

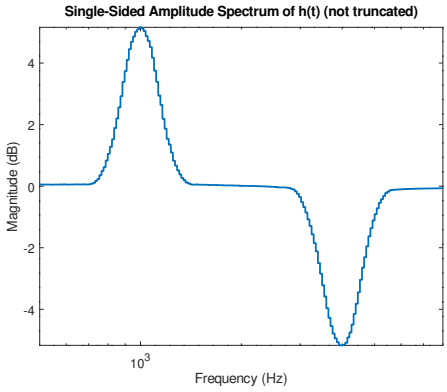
Een filter met een hogere resolutie van $\frac{1}{48}$ octaafbanden is weergegeven in figuur 3.11. Deze is uitvergroot, zodat hierin de stappen van het filter te zien zijn. Wanneer het filter getrunceert wordt naar 512 coëfficiënten, zijn de stappen voor dit filter niet meer zichtbaar. Dit is weergegeven in figuur 3.12.

Het filter met $\frac{1}{48}$ octaafbanden resulteert in een error van $5,30 \cdot 10^{-8}$. De error is met een factor van $1,32 \cdot 10^{4}$ verbeterd ten opzichte van het ongecorrigeerde signaal. Daarnaast is de error $2,7$ keer kleiner dan bij het direct bepalen van de transferfunctie in het tijdsdomein. Dit komt met name doordat de verstoringen door het ruiskarakter van het testsignaal met de octaafbanden worden uitgemiddeld.

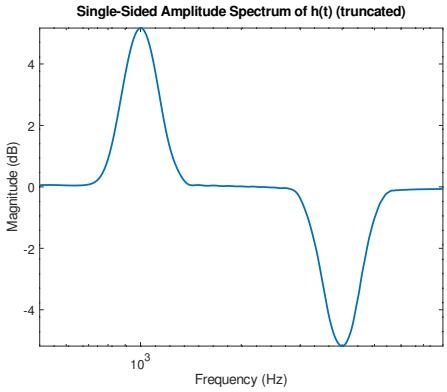
Deze techniek zou ook toegepast kunnen worden als extra verwerkingsstap bij de andere filtertechnieken.



Figuur 3.10: Energie van ingangs- en uitgangssignaal in $\frac{1}{3}$ octaafbanden



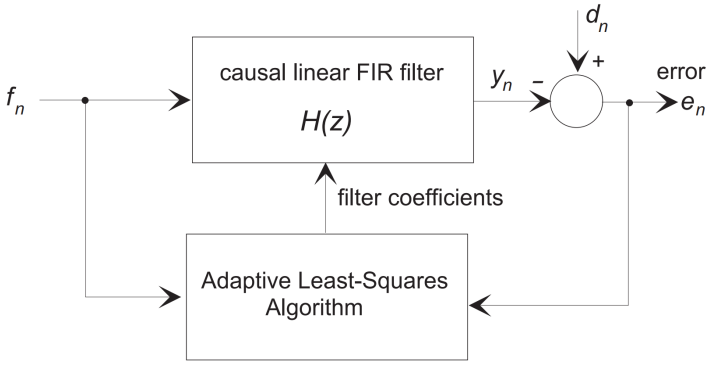
Figuur 3.11: Filter in $\frac{1}{48}$ octaafband stappen



Figuur 3.12: Getrunceert filter in $\frac{1}{48}$ octaafband stappen

3.2.5. Adaptive Filters

Een andere methode voor het bepalen van de filters, zijn de zogeheten 'adaptive filters'. In plaats van berekenen van een filter vanuit het in- en uitgangssignaal, worden hierbij de coëfficiënten berekend op basis van de error tussen de signalen. Twee voorbeelden hiervan zijn de Least Mean Square Error (LMS) en Recursive Least Square Error (RLS) filters. Een blokdiagram van de LMS is weergegeven in figuur 3.13.



Figuur 3.13: Architectuur Least Square Error filter [4]

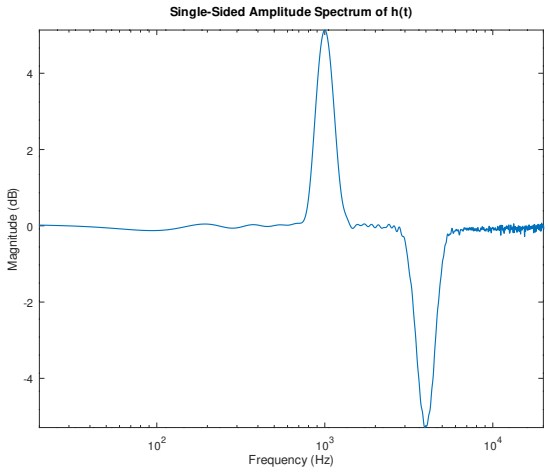
LMS Simulatie

Vanuit [4] is de voorbeeld applicatie overgenomen om een filter te berekenen. Deze filters worden berekend op basis van correlaties van de matrices. Aan de applicatie worden het ingangs- en uitgangssignaal meegegeven en het aantal coëfficiënten.

Dit resulteert in het filter, zoals weergegeven in figuur 3.14. Dit filter heeft een piek en dal bij de correcte frequenties. De amplitude is hierbij 5 dB. Ook zijn er duidelijk meer rimpelingen te zien dan in de andere filters.

Het toepassen van dit filter leid tot een error van $2,79 \cdot 10^{-6}$. De error is met een factor van $2,5 \cdot 10^{2}$ verbeterd ten opzichte van het ongecorrigeerde signaal.

Een groot voordeel van deze methode is dat de berekeningen van de filtercoëfficiënten relatief eenvoudige matrixbewerkingen zijn. Het zou dus mogelijk zijn deze berekeningen op een FPGA te implementeren. Een nadeel is dat dit filter met de gebruikte rekenmethode, niet resulteert in een lineaire filter.



Figuur 3.14: LMS filter response

RLS Simulatie

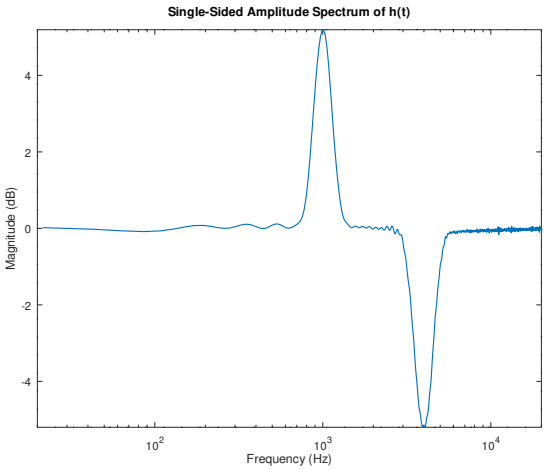
Vanuit [5] is een functie overgenomen voor een RLS filter. Dit filter wordt op een soortgelijke manier berekend als het LMS filter, maar heeft ook de mogelijkheid om recursief toegepast te worden. Hiermee zou dus ook, terwijl een systeem draait, de filtercoëfficiënten aangepast kunnen worden.

Het gebruik van deze filterfunctie resulteert in het filter, zoals weergegeven in figuur 3.15. In tegenstelling tot het LMS filter is de fase van dit filter wel lineair. Daarnaast heeft de filter een piek en dal van 5 dB bij de correcte frequenties. Ook zijn er duidelijk meer rimpelingen te zien dan in de andere filters.

Het toepassen van dit filter leid tot een error van $1,65 \cdot 10^{-6}$. De error is met een factor van $4,2 \cdot 10^{2}$ verbeterd ten opzichte van het ongecorrigeerde signaal.

Opvallend bij deze methode is dat met gekozen het aantal coëfficiënten en samples, de berekening ongeveer zes minuten duurt. Dit in tegenstelling tot alle andere filters die slechts 1 tot 30 seconden nodig hebben. Het is niet verder onderzocht of dit komt door de specifieke implementatie, of door het werkingsprincipe van het RLS filter.

Net als bij het LMS filter zijn ook hier de berekeningen relatief eenvoudige matrixbewerkingen. Het zou dus mogelijk zijn deze berekeningen in een FPGA te implementeren.



Figuur 3.15: RLS filter response

3.2.6. Maximum Length Sequence

Een andere mogelijkheid om het filter te berekenen, is door middel van een Maximum Length Sequence (MLS). Dit is een signaal met bepaalde eigenschappen die het mogelijk maken om de overdracht gemakkelijk te berekenen.

Berekening

Vanuit de signaaltheorie is bekend dat de kruis-correlatie tussen $x(n)$ en $y(n)$ van lineaire systemen, gerelateerd is aan de auto-correlatie van de input geconvolueerd met de impulsrespone [6]. Dit kan wiskundig als volgt aangegeven worden:

$$R_{xy}(n) = R_{xx}(n) * h(n) \tag{3.5}$$

Het convolueren van een impuls met een Dirac Delta-Functie resulteert in de originele impuls. Doordat de autocorrelatie van de MLS in essentie een Dirac Deltafunctie is, kan hier dus het volgende mee gesteld worden:

$$R_{xx}(n) \approx \delta(n) \rightarrow R_{xy}(n) = \delta(n) * h(n) = h(k) \tag{3.6}$$

Hieruit volgt dus dat de kruiscorrelatie van de ingang en uitgang (R_{xy}) bij een MLS signaal, resulteert in de impulsresponse van het systeem (h). Deze impulsresponse kan vervolgens gebruikt worden voor het filteren.

Simulatie

Voor het simuleren van deze methode is gebruik gemaakt van de MLS generator, zoals beschreven in [7]. Hiermee is een reeks gegenereerd met $N = 16$ en dus een lengte van 65535.

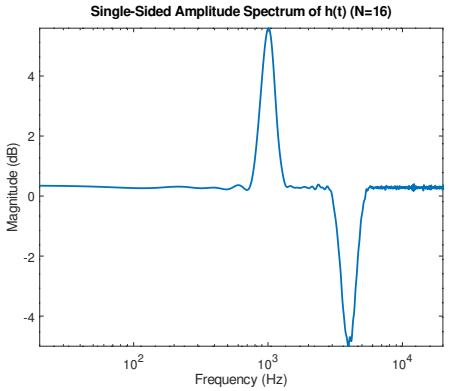
Vervolgens wordt er een kruiscorrelatie uitgevoerd tussen het ingangssignaal en het aangepaste uitgangssignaal. De transferfunctie van de resulterende impuls wordt vervolgens geïnverteerd, correct geschaald en getrunceerd. Dit resulteert in de filterresponse, zoals weergegeven in figuur 3.16.

In dit figuur zijn een piek en dal te zien bij de correcte frequenties met een amplitude van 5 dB. Ook zijn er wat schommelingen in de response te zien voor en na de pieken en rond de hogere frequenties.

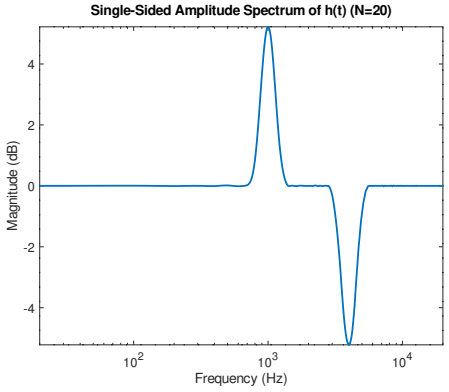
Opvallend bij dit filter is dat de gain van het filter niet op 0 dB ligt. Er is sprake van een kleine offset. Voor een uiteindelijke implementatie zou de oorzaak hiervan verder onderzocht en gecorrigeerd moeten worden.

Voor het berekenen van de error is het filter tijdelijk handmatig gecompenseerd, zodat de gain wel op 0 dB ligt. Dit leid tot een error van $5,70 \cdot 10^{-6}$. De error is met een factor van $9,17 \cdot 10^{4}$ verbeterd ten opzichte van het ongecorrigeerde signaal.

Wanneer de MLS reeks wordt uitgebreid, naar bijvoorbeeld $N = 20$, worden de kleine schommelingen in het filter die bij $N = 16$ aanwezig gereduceerd. Dit filter is weergegeven in figuur 3.17. Hierbij kan een verbetering van $5,83 \cdot 10^{6}$ gerealiseerd worden ten opzichte van het ongecorrigeerde signaal.



Figuur 3.16: Frequentie respons MLS filter ($N = 16$)



Figuur 3.17: Frequentie respons MLS filter ($N = 20$) met gecompenseerde gain offset

Nadelen MLS

De MLS heeft ook nadelen die niet direct naar voren komen in de simulatie. Omdat de MLS uit een bloksignaal bestaat, heeft deze zeer scherpe overgangen. Deze overgangen kunnen niet gerepresenteerd worden door luidsprekers. Ook veroorzaken deze overgangen oscillaties (ringing en overshoot). Hierdoor kunnen afwijkingen ontstaan in het uiteindelijke filter. Mogelijk kan dit gelimiteerd worden door de MLS reeks in bandbreedte te beperken.

3.3. Afweging voor filtermethodes

Aan de verschillende filtermethodes zitten voor- en nadelen. Wanneer het gewenst is om alles op de FGPA te implementeren, is het advies om de Adaptive Filters te gebruiken. De berekeningen hierbij blijven relatief eenvoudig. De Maximum Length Sequence is ook goed realiseerbaar op een FPGA.

De berekening van het filter in het frequentiedomein en omzetten naar het tijddomein, geeft veruit de meeste flexibiliteit. Bij deze methode kunnen veranderingen in het frequentiedomein, zoals frequentie- of gain-limieten, makkelijk aangebracht worden.

Er is gekozen om verder te werken met berekeningen van het filter in het frequentiedomein en deze om te zetten naar een FIR. Dit vanwege de toegevoegde flexibiliteit. De berekeningen voor deze methode zijn het meest intensief, maar met de toevoeging van een processor binnen het systeem haalbaar.

3.4. Meetsignalen

Voor het meten van de overdracht van een systeem is het van belang dat in het gehele spectrum voldoende energie van het meetsignaal aanwezig is. Wanneer er ergens te weinig energie van het testsignaal aanwezig is, is het moeilijker om op dat punt de overdracht vast te stellen.

In dit onderdeel worden verschillende opties voor meetsignalen uiteengezet. Hierbij worden de belangrijke eigenschappen van elk signaal toegelicht en wordt een vergelijking tussen de signalen uitgevoerd.

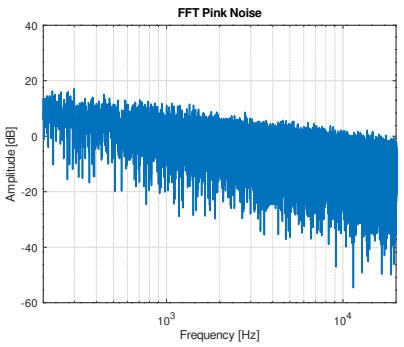
3.4.1. Impuls

Voor het meten van een overdracht, kan een impuls signaal gebruikt worden. Hierbij wordt één puls afgespeeld en wordt de respons van het systeem opgenomen. De opname is ook direct de impuls respons van het systeem.

Het nadeel bij deze methode is dat er slechts een korte tijd energie in het systeem gestopt wordt. Hierdoor is de totale energie over de gehele lengte van de opname relatief laag. Dit maakt deze methode gevoeliger voor achtergrondlawaai.

3.4.2. Willekeurige Ruis

Een goede manier om energie in het gehele spectrum te genereren, is door middel van een ruisachtig testsignaal. Ruis wordt vaak aangeduid met een kleur. Veel voorkomend hierbij zijn witte en roze ruis. Bij witte ruis is de gemiddelde energie per frequentieband gelijk. Bij roze ruis is de gemiddelde energie per relatieve frequentieband gelijk. Roze ruis is dus vlak op een octaafband analyse. Witte ruis loopt hierbij met 3 dB per octaafband op.



Figuur 3.18: FFT roze ruis

Roze ruis sluit hierbij beter aan bij het menselijk gehoor omdat deze ook een logaritmisch verloop heeft. Daarnaast is signaal ook prettiger om naar te luisteren dan witte ruis. Om deze reden is er verder gebruik gemaakt van roze ruis, maar in feite kan voor elk type ruis gekozen worden.

Ruis wordt doorgaans gegenereerd door het maken van een reeks willekeurige getallen. Wanneer een reeks van voldoende lengte gegenereerd wordt, resulteert dit in een signaal met energie verdeeld over het gehele spectrum.

Doordat de reeks willekeurig is, zal de energie van het gehele spectrum nooit exact vlak zijn. Ook zal het niet consistent zijn wanneer een nieuw ruissignaal gegenereerd wordt met andere getallen.

Een voorbeeld van de FFT van een willekeurig ruis signaal is weergegeven in figuur 3.18. Hierin is te zien dat het momentane spectrum inderdaad niet 'vlak' is.

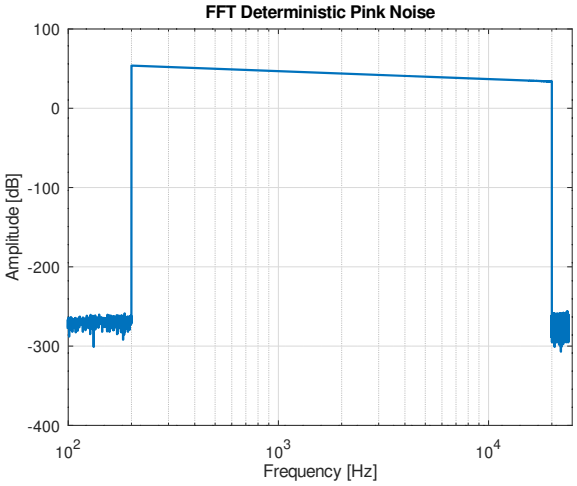
3.4.3. Gedetermineerde Ruis

Een ruis-signaal kan ook pseudo-willekeurig gegenereerd worden. Hierbij klinkt het signaal als een willekeurig ruis-signaal, maar in werkelijkheid wordt er een bepaald patroon herhaald.

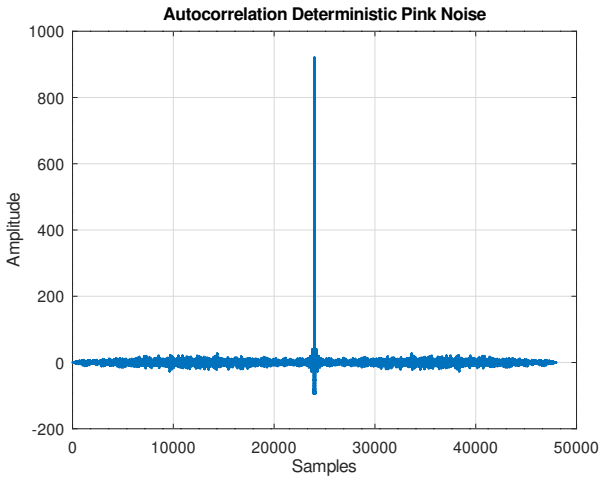
De ruis kan worden gegenereerd vanuit het frequentiedomein. Hierbij kunnen zowel de amplitude als de fase onafhankelijk van elkaar worden gegenereerd. De amplitude volgt hierbij een vooraf gedefinieerde vector afhankelijk van het type ruis. Hiermee wordt gezorgd dat op alle frequenties een bepaalde hoeveelheid signaal aanwezig is.

Voor de fase-vector worden waarden van $-\pi$ tot π distribueert over de lengte van de vector. Deze waarden worden vervolgens gepermuteerd. Het geheel wordt vervolgens omgezet naar het tijddomein. Het resulterende signaal klinkt net als normale ruis. Het voordeel hierbij is dat het spectrum constant en volledig vlak is. Dit is weergegeven in figuur 3.19.

Een ander voordeel treedt op bij de correlatie van het signaal. Doordat het signaal gedetermineerd is, geeft de autocorrelatie één duidelijke piek wanneer de signalen exact in fase zijn. Daarnaast geeft een correlatie met een ander signaal een kleine correlatie-coëfficient. Van deze eigenschap zal verder gebruik gemaakt worden tijdens het bepalen van de delay in hoofdstuk 3.5



Figuur 3.19: FFT gedetermineerde roze ruis



Figuur 3.20: Autocorrelatie gedetermineerde roze ruis

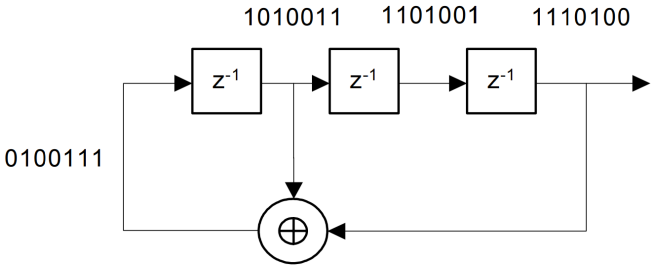
3.4.4. Maximum Length Sequence

Een Maximum Length Sequence (MLS) is een pseudo-willekeurige binaire reeks met een lengte van $2^N - 1$. De reeks kan gegenereerd worden door middel van schuifregisters en XOR operaties. Dit is weergegeven in figuur 3.21. Door deze op de correcte manier te koppelen, wordt een audiosignaal gegenereerd welke 1 is als het schuifregister 1 oplevert en -1 als het schuifregister 0 oplevert. Het spectrum van de MLS is gelijk aan dat van witte ruis. Het fase-gedrag is gekoppeld aan de gegenereerde reeks. Deze is daarmee ook pseudo-willekeurig

De MLS heeft een bijzondere eigenschap waarvan gebruik gemaakt kan worden voor filterberekeningen. De circulaire convolutie van de reeks resulteert namelijk in een Delta-functie (met een kleine offset). Een voorbeeld voor een MLS met $N = 2$ is weergegeven in figuur 3.22. Hierin is de bovenste rij de ingangs-MLS reeks en de onderstaande rijen de circulair verschoven variant. Hierbij is het eerste nummer van de reeks gemarkeerd. De bovenste rij wordt vervolgens vermenigvuldigd met elk van de onderstaande rijen. De optelling van de vermenigvuldiging is per rij weergegeven in de rechtse kolom.

Hierin wordt zichtbaar dat het resultaat van de circulaire convolutie van het MLS signaal een Delta-functie is met een offset van -1.

Doordat de MLS een deterministische reeks is, is het ook mogelijk om te middelen. Hierbij kan de reeks meerdere malen worden afgespeeld, waarna een gemiddelde van de gecombineerde reeksen bepaald kan worden. Hierdoor kan het effect van ruis en verstoringen in de metingen sterk gereduceerd worden.



Figuur 3.21: Schematische voorstelling MLS generator [7]

-1	-1	1 x	Result
1	-1	-1	-1
-1	1	-1	-1
-1	-1	1	3

Figuur 3.22: Circulaire convolutie MLS

3.4.5. Afweging meetsignalen

Vanuit deze set meetsignalen, bied de gedetermineerde roze ruis de meeste voordelen. Dit signaal relatief prettig om naar te luisteren, heeft een goede gedetermineerde basis en een vlak spectrum. Daarnaast bied het mogelijkheden om ook vertraging te kunnen meten door de goed gedefinieerde autocorrelatie. De verder genoemde meetsignalen beschikken niet over al deze eigenschappen. Om deze reden wordt in deze applicatie verder gegaan met de gedetermineerde roze ruis.

De gedetermineerde ruis heeft als nadeel dat het lastig te genereren is. Hierbij is het gebruik van FFTs nodig. Het genereren van deze signalen zal in het uiteindelijke systeem dus op een processor moeten gebeuren, of het signaal moet vooraf gegenereerd worden.

3.5. Delay meting

In dit onderdeel wordt het meten van vertraging tussen luidsprekers behandeld.

3.5.1. Doel

Het doel van deze optimalisatie is om de onderlinge verschillen in looptijd tussen luidsprekers te detecteren en corrigeren. Voor het meten van deze vertragingen tussen luidsprekers, zijn verschillende methoden mogelijk. Eén van de meest eenvoudige hierbij is het afspelen van een impuls-signaal. Hiermee kan de vertraging van het gehele systeem gemeten worden en, wanneer goed gebruikt, ook de vertraging tussen onderlinge luidsprekers.

Deze methode werkt minder goed bij negatieve signaal-ruis verhoudingen en is daarnaast ook onaangenaam om naar te luisteren. In het kader hiervan is gekeken naar een alternatief meetsignaal en meetmethode.

De volgende parameters zijn hierbij voorop gesteld:
- Goede performance bij negatieve signaal-ruis verhouding (Achtergrondlawaai harder dan signaal)
- Overlast van het meetsignaal minimaliseren
- Mogelijkheden open houden voor het gelijktijdig inregelen van EQ

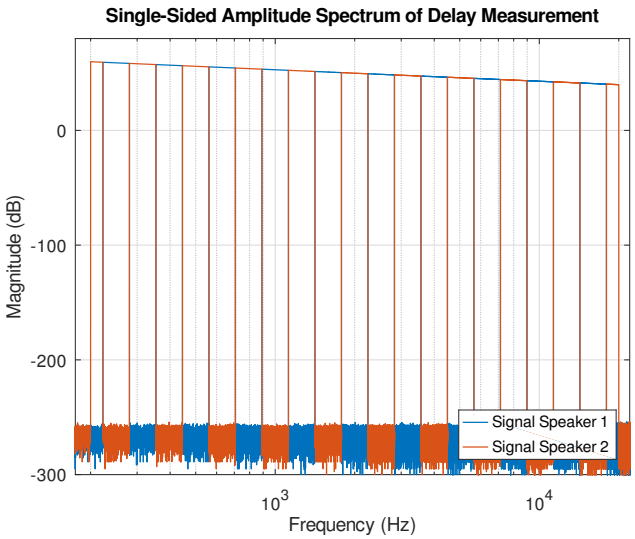
3.5.2. Concept werking

In hoofdstuk 3.4.3 is de gedetermineerde roze ruis geïntroduceerd. Door de gepermuteerde fase van dit signaal, volgde hieruit de eigenschap dat de autocorrelatie van het signaal op één enkel punt een onderscheidende piek heeft. Dit is wanneer de signalen exact over elkaar heen liggen. Daarnaast heeft het signaal een nauwkeurige frequentieoverdracht.

Wanneer dit signaal gebruikt wordt, kan de delay tussen twee of meer luidsprekers bepaald worden. Hierbij kan dan alleen geen onderscheid gemaakt worden tussen welke luidspreker bij welke delay-tijd hoort.

Om deze reden is gekozen om een uniek meetsignaal per luidspreker aan te bieden. Om de mogelijkheden voor de EQ open te houden, is het van belang dat de signalen goed te onderscheiden zijn, maar ook over het gehele gewenste frequentiespectrum energie bevatten.

Hiervoor is de techniek uit hoofdstuk 3.2.4 toegepast. De ruis wordt opgedeeld in octaafbanden. Vervolgens wordt de helft van de banden door de eerste luidspreker weergegeven, en de andere helft door de andere luidspreker. Dit is geïllustreerd in figuur 3.23. Hierbij is een verdeling in $\frac{1}{3}$ octaafbanden gemaakt. De onderverdeling in $\frac{1}{N}$ octaafbanden, kan hierbij gevarieerd worden.

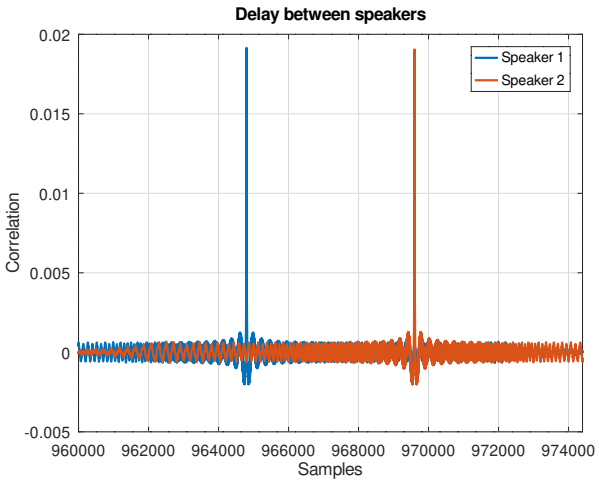


Figuur 3.23: Frequentiespectrum meetsignalen met afwisselende frequentiebanden

Doordat de twee signalen gecombineerd weer het volledige spectrum vullen, klinken ze samengevoegd het zelfde als de originele ruis.

De twee meetsignalen kunnen gelijktijdig afgespeeld worden over twee luidsprekers. De opname hiervan kan vervolgens gecorreleerd worden met de originele signalen. Dit is gesimuleerd door een audiofragment te genereren met beide signalen met een onderlinge offset van 100 ms.

De kruiscorrelaties tussen de meetsignalen en het gesimuleerde signaal zijn deels weergegeven in figuur 3.24. Hierin is te zien dat er een vertraging van 4800 samples tussen de signalen zit. Dit komt overeen met 100 ms. Verder is te zien dat de correlatiecoëfficient op andere punten laag is. In simulatie werkt deze methode dus naar verwachting



Figuur 3.24: Kruiscorrelatie meetsignalen en uitgangssignaal

3.5.3. Akoestische simulaties

Drie belangrijke factoren die van invloed kunnen zijn op deze metingen zijn: galm, achtergrondlawaai en luidspreker-bandbreedte. De invloed van deze factoren wordt daarom in simulatie getest. Om de gevoeligheid voor deze factoren te bepalen, wordt de Crest-factor bepaald. Deze wordt bepaald op basis van de piek van de correlatie ten opzichte van de RMS waarde het verdere signaal. Ook wordt de invloed van de lengte van het meetsignaal onderzocht.

De onderstaande simulaties zijn met de volgende parameters uitgevoerd, tenzij anders vermeld:

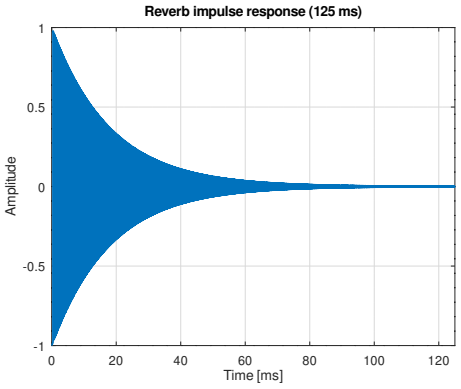
- Frequentiespectrum meetsignaal: 200-20000 Hz
- Lengte meetsignaal: 20 s
- Aantal luidsprekers: 2
- Vertraging tussen luidsprekers: 100 ms

Verder worden alle simulaties uitgevoerd met meetsignalen opgedeeld in $\frac{1}{3}$ en $\frac{1}{48}$ octaafbanden. Dit om de invloed van de grootte van de frequentiebanden te onderzoeken.

Gevoeligheid voor Galm

Voor het simuleren van de galm, moet dit effect toegevoegd worden aan het gesimuleerde meetsignaal. Dit wordt bereikt door een impuls-response van een galm te genereren en het meetsignaal hier vervolgens mee te convolueren.

Deze impuls-response kan gesimuleerd worden door een ruis met een amplitude die exponentieel afneemt in de tijd. De tijdconstante waarbij het niveau van de ruis 60 dB lager ligt is dan evenredig met de nagalmtijd. Hierbij is het van belang dat deze ruis geen correlatie heeft met de gedetermineerde ruis die als testsignaal gebruikt wordt. Een voorbeeld van een galm impuls-response met een galm-tijd van 125 ms, is weergegeven in figuur 3.25. Voor de simulatie zijn verschillende galm-tijden gebruikt van 125 tot 8000 ms.

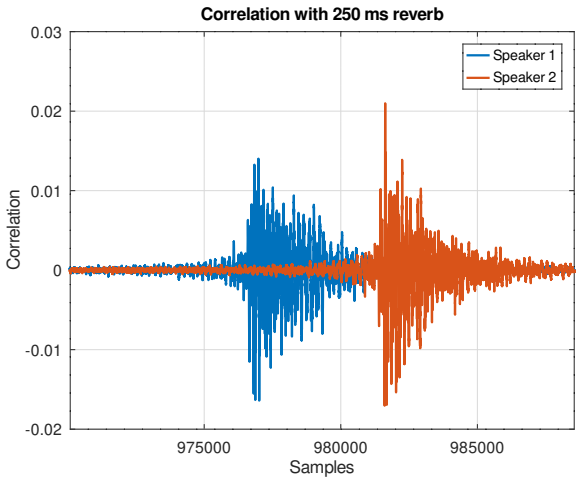


Figuur 3.25: Impuls respons galm 125 ms

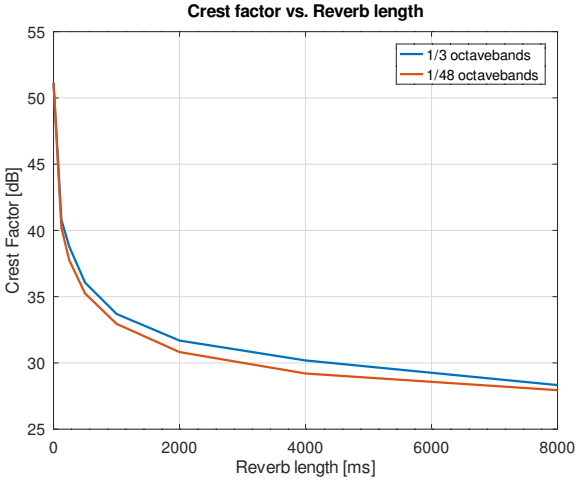
Het effect van de galm is weergegeven in figuur 3.26. Dit meetsignaal is verstoord met een galm van 250 ms. Wanneer dit figuur wordt vergeleken met het niet verstoorde signaal in figuur 3.24, is te zien dat er meerdere pieken ontstaan. Dit is te vergelijken met de reflecties tegen verschillende wanden in een ruimte.

Hierbij is het belangrijk op te merken dat de correcte delaytijd niet meer per definitie de hoogste piek in de correlatie heeft. Hierdoor kunnen afwijkingen optreden wanneer alleen gekeken wordt naar de hoogste piek. Om deze methode correct te implementeren moet gezocht worden naar de eerste significante piek in de correlatie.

Verder is ook te zien dat er meer variatie in de correlatie zit. Hierdoor is er meer energie in het geheel aanwezig waardoor de Crest-factor verslechterd. De Crest-factor is in figuur 3.27 uitgezet ten opzichte van de galm-lengte. Hierin is te zien dat naar mate de galm-lengte groter wordt, de Crest-factor verder afneemt. De invloed van de grootte van de octaafbanden is hierbij klein.



Figuur 3.26: Correlatie bij meetsignaal met galm



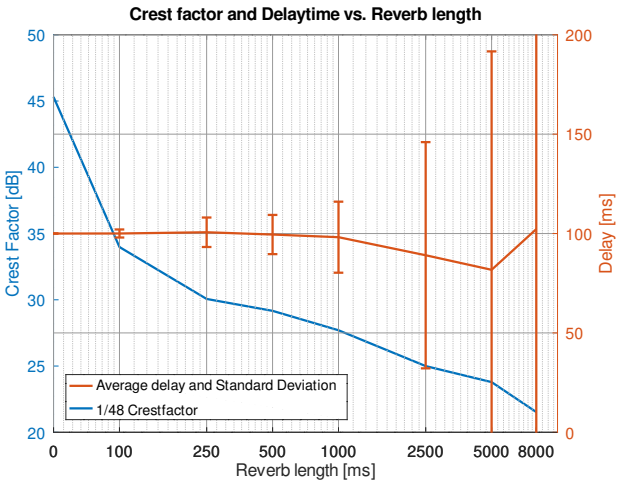
Figuur 3.27: Crestfactor ten opzichte van galm-lengte

De Crest-factor is in deze situatie niet volledig toereikend om de invloed van galm te beoordelen. De piek van de correlatie is namelijk uitgesmeerd en in sommige gevallen is de hoogste piek van locatie veranderd.

Om deze reden is de simulatie herhaald met voor elke galm-lengte dertig willekeurig gegenereerde galm impuls-responsies.

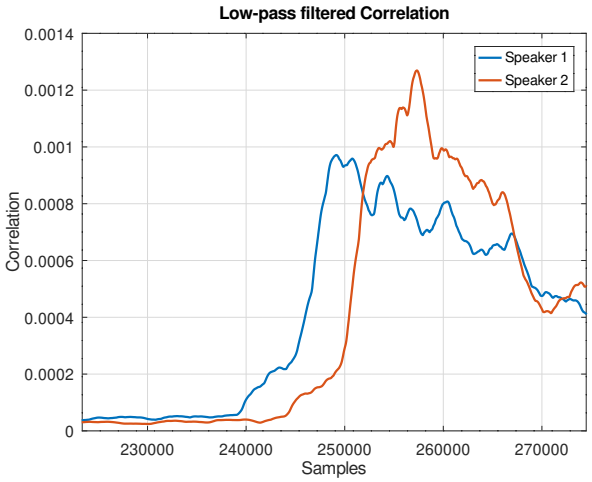
Hierbij is de Crest-factor bepaald samen met de gemiddelde delay-tijd en de standaarddeviatie is berekend.

Dit is weergegeven in figuur 3.28. Hierin is te zien dat hoewel de gemiddelde delay-tijd rond 100 ms blijft, de standaarddeviatie toeneemt tot 100 ms. Dit wordt veroorzaakt doordat er meerdere pieken in de correlatie ontstaat. Hierdoor werkt de methode van het selecteren van de hoogst piek niet correct.

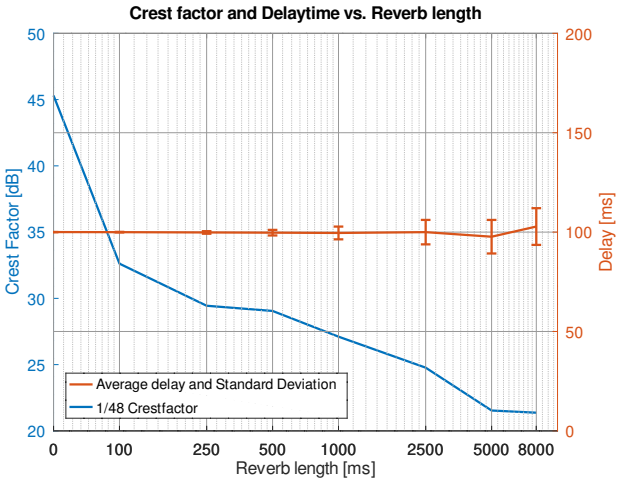


Figuur 3.28: Crest-factor en delay-tijd ten opzichte van galm-lengte

Om deze reden is een alternatieve detectiemethode gebruikt. Eerst wordt de absolute waarde van correlatie low-pass gefilterd. De resulterende curve is weergegeven in figuur 3.29. Vervolgens wordt gekeken wanneer de curve voor de eerste keer de helft van de piek-waarde overschrijd. Dit punt wordt gebruikt als referentie. Deze methode is vervolgens toegepast om opnieuw de delaytijden te bepalen. Het resultaat hiervan is weergegeven in figuur 3.30. De gemiddelde afwijking is hierbij gereduceerd tot 1 ms en de standaarddeviatie tot 10 ms. Door deze verbetering is het dus mogelijk om de delaytijd met een grotere precisie te benaderen.

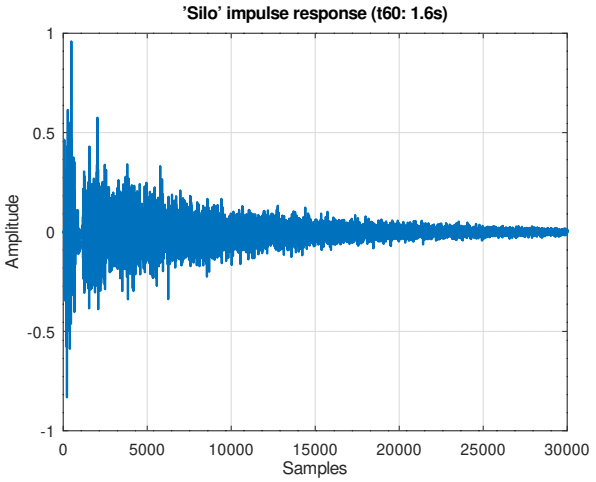


Figuur 3.29: Low-pass gefilterede correlatie

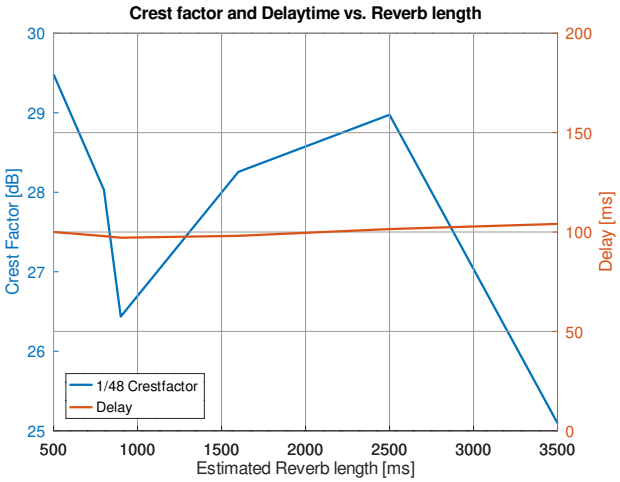


Figuur 3.30: Crest-factor en delay-tijd ten opzichte van galm-lengte met verbeterede piek-detectie

De galm zoals gebruikt voor de simulaties, weergegeven in figuur 3.25, is niet volledig representatief voor de echte galm van een ruimte. De gesimuleerde galm kan gezien worden als een ideale galm, maar heeft voor deze metingen een groter impact op het resultaat dan de galm van een daadwerkelijke ruimte. Om deze reden is ook een set van meer realistische gesimuleerde ruimte impuls-responses [8] gebruikt om de invloed van galm te onderzoeken. In figuur 3.31 is een impulse-response weergegeven met een galmtijd van $1,6\,s$. In vergelijking met de vorige galm, is te zien dat er hier na de eerste piek een vertraging is tot de eerste reflecties binnen komen. Vervolgens is het niveau van deze reflecties ook lager dan de gesimuleerde galm.



Figuur 3.31: Impuls respons van ruimte met $t60 = 1.6s$



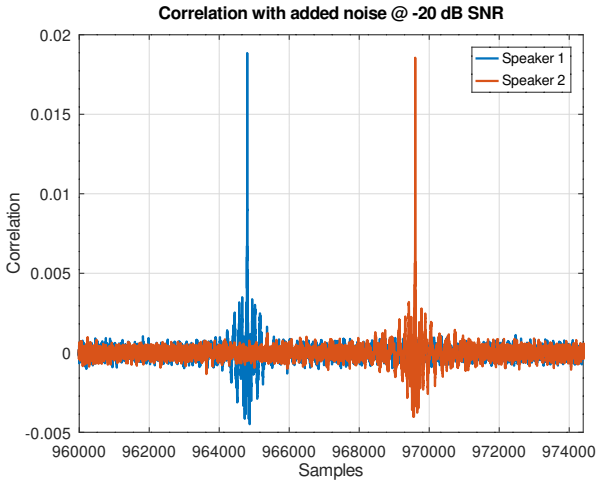
Figuur 3.32: Crest-factor en delaytijd met ruimte impuls-responses

Voor de gebruikte responses is de galmtijd benaderd. De plot van de delaytijd en Crest-factor ten opzichte van de galmtijd is weergegeven in figuur 3.32. Hierin is te zien dat de delaytijd een maximale afwijking heeft van $5\,ms$. Daarnaast is te zien dat Crest-factor geen geleidelijke curve meer is, maar wel bruikbaar blijft zolang de waarde boven de 20 dB ligt. Hieruit blijkt dat de Crest-factor geen directe verband heeft met de galmtijd. De Crest-factor is sterker afhankelijk van de vorm van de galm impulse-response.

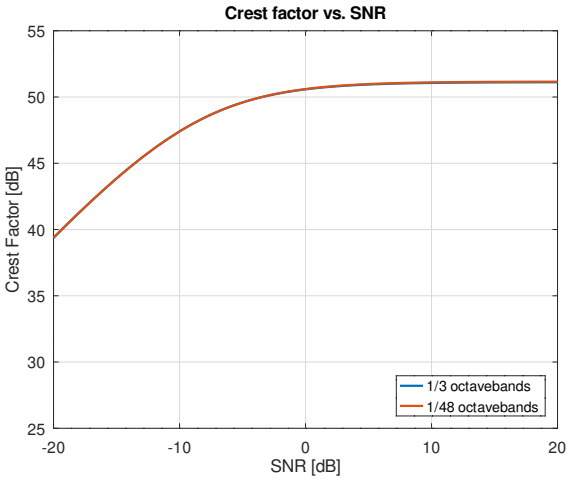
Gevoeligheid voor achtergrondlawaai

Voor het testen van de gevoeligheid voor achtergrondlawaai, is er witte ruis in verschillende signaal-lawaaiverhoudingen toegevoegd. Hiervoor is elke keer het zelfde ruis-signaal gebruikt en is alleen de amplitude hiervan gevarieerd.

Het gevolg van deze ruis voor de Crest-factor is weergegeven in figuur 3.34. Het resultaat van de correlatie is weergegeven in figuur 3.33. Hier is te zien dat, in tegenstelling tot figuur 3.24, de correlatie buiten de pieken niet snel uitdooft. Hierdoor is in de gehele correlatie meer energie aanwezig waardoor de Crest-factor verslechterd. De piek blijft in dit geval echter wel duidelijk zichtbaar. Hierdoor is het mogelijk om ook met een grote negatieve signaal-ruis verhouding, de delay tussen luidsprekers te meten. De grootte van de octaafbanden heeft hier verder geen invloed op de Crest-factor.



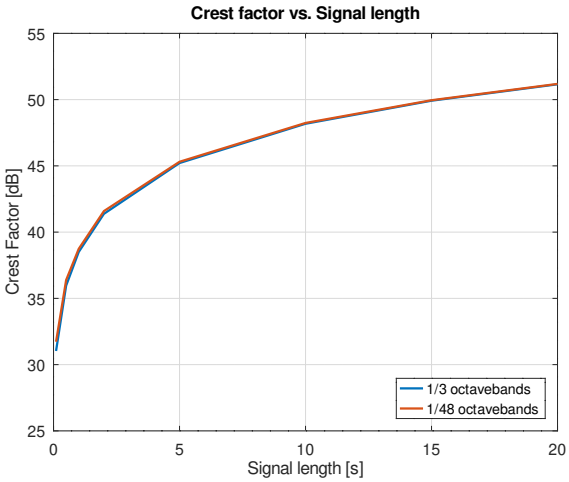
Figuur 3.33: Correlatie bij meetsignaal met toegevoegde ruis



Figuur 3.34: Crest-factor ten opzichte van SNR

Signaal-lengte

Om de invloed van de signaal-lengte te testen, zijn verschillende signaal-lengtes gesimuleerd. Hiervoor zijn lengtes van 0,1 tot 20 seconden genomen. In figuur 3.35 is de invloed hiervan op de Crest-factor weergegeven. Hierin is te zien dat bij een korte lengte de Crest-factor kleiner wordt. Dit wordt veroorzaakt doordat het niet-uitgedoofde deel van de correlatie een relatief groter deel van het geheel inneemt doordat de gehele lengte kort is. Hierdoor is de RMS waarde van het signaal hoger. Daarnaast ligt bij een kortere lengte de hoogte van de piek ook lager. Hieruit volgt een kleinere Crest-factor.



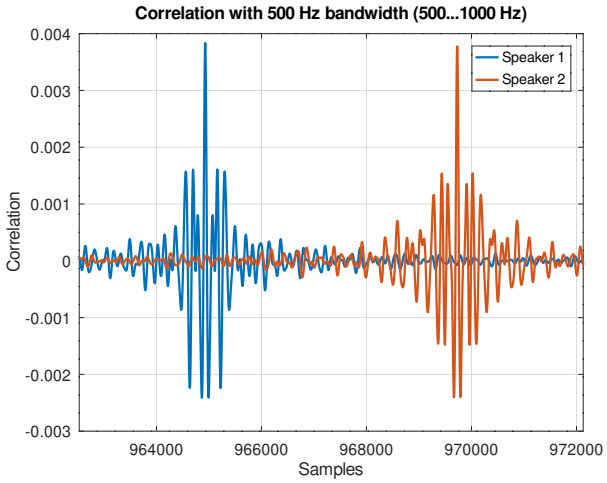
Figuur 3.35: Crest-factor ten opzichte van lengte meet-signaal

Bandbreedte

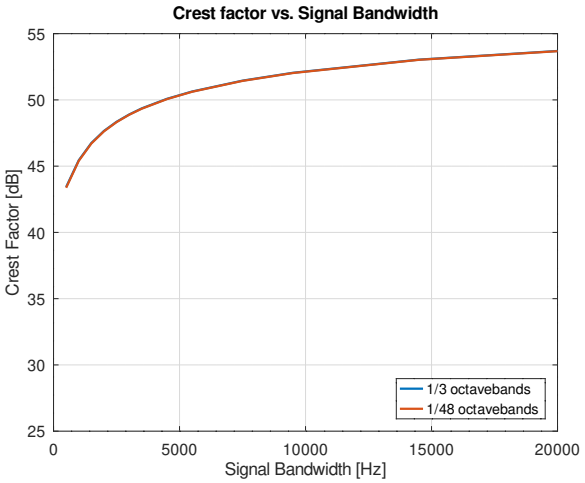
Voor het testen van het effect van de bandbreedte, is het meetsignaal door verschillende band-pass filters gehaald. Als startfrequentie is hierbij 500 Hz genomen en de bandbreedte is gevarieerd van 500 tot 20000 Hz.

Het gevolg van de bandbreedte voor de Crest-factor, is weergegeven in figuur 3.37. Hierin is te zien dat de bandbreedte een vergelijkbare invloed op de Crest-factor heeft als de SNR. De Crest-factor neemt namelijk sterk af bij kleine bandbreedtes.

In figuur 3.36 is een deel van de correlatie bij een bandbreedte van 500 Hz weergegeven. In de voorgaande gevallen verslechterde de Crest-factor doordat de correlatie rondom de piek veranderde. In dit geval is te zien dat juist de hoogte van de piek is veranderd. Ook dit resulteert in een kleinere Crest-factor. Opnieuw heeft de grootte van de octaafbanden geen invloed op de Crest-factor.



Figuur 3.36: Correlatie bij meetsignaal met gelimiteerde bandbreedte



Figuur 3.37: Crest-factor ten opzichte van Bandbreedte

3.6. Gecombineerde EQ en Delay

De hiervoor beschreven methodes kunnen op zichzelf staand goed functioneren. Echter is er ook gekeken om de methode voor delay metingen te combineren met de filter bepaling via inverse FFT.

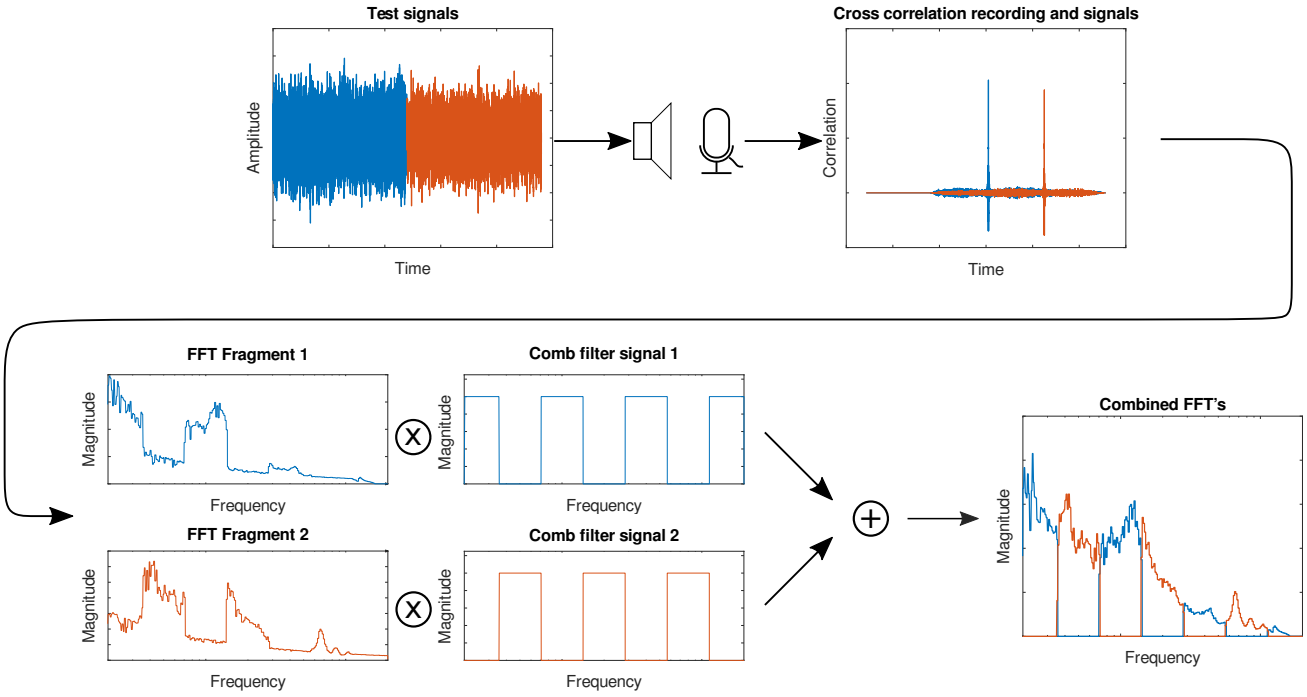
In figuur 3.23 is een signaal weergegeven waarbij voor één luidspreker in de helft van de frequentiebanden signaal aanwezig is. Voor het verkrijgen van de frequentie response in de overige banden, zijn er twee mogelijkheden.

- Frequentiebanden voldoende klein maken en interpoleren in de banden waar geen signaal is
- Meetsignaal van frequentiebanden laten wisselen zodat bij elke frequentieband per luidspreker signaal aanwezig is

In dit geval is de methode van het wisselen van frequentiebanden gekozen. Dit is gedaan zodat het volledige spectrum gemeten wordt, in plaats van dat er delen benaderd worden.

Om het meten met wisselende frequentiebanden te realiseren, wordt een signaal gegeneerd waarbij in de eerste helft de even frequentiebanden gevuld worden en in de tweede helft de oneven banden (in het geval van twee luidsprekers). De FFT van het totale signaal levert hierbij weer een volledig gevuld spectrum.

Dit concept is geïllustreerd in figuur 3.38. Hierin is te zien dat eerst een testsignaal gegenereerd wordt bestaande uit twee fragmenten. Dit wordt afgespeeld en gelijktijdig opgenomen. In de opname wordt met behulp van kruiscorrelaties met de originele twee fragmenten, de delay en de locatie van de fragmenten in de opname bepaalt. De fragmenten worden in het tijddomein gescheiden en vervolgens wordt de FFT hiervan genomen. De frequentiebanden die niet in het fragment zitten worden weg gefilterd met een kamfilter. De resulterende FFT's worden bij elkaar opgeteld en vormen samen weer het volledige spectrum van één van de luidsprekers. Dit wordt herhaald voor de andere luidspreker. Hier kunnen vervolgens de in hoofdstuk 3 besproken technieken op worden om een inverse filter te genereren.



Figuur 3.38: Conceptwerking gelijktijdige EQ en Delay

Deze techniek kan toegepast worden op meerdere luidsprekers. Dit kan door te zorgen dat elke luidspreker zijn eigen frequentiebanden heeft. Hierdoor interfereren de luidsprekers niet met elkaar. Door vervolgens net zo veel fragmenten te gebruiken als het aantal luidsprekers, kan van elke luidspreker een volledig spectrum opgebouwd worden.

Deze methode is verder uitgewerkt en getest binnen een Python-applicatie. Dit is beschreven in hoofdstuk 4 en hoofdstuk 5.

3.6.1. Uitbreidingen en Limitaties

Vanuit de testen van met dit concept zijn een aantal limitaties en mogelijkheden tot uitbreiding voortgekomen.

Uitbreiding: Scrambling van frequentiebanden

Wanneer er geoptimaliseerd wordt met meer de twee luidsprekers, wordt het testsignaal duidelijk tonaal. Dit wordt veroorzaakt door dat alle banden in het signaal een constant aantal octaafbanden uit elkaar liggen. Dit effect kan mogelijk voorkomen worden door de frequentiebanden willekeurig te verdelen onder de signalen. In plaats van een constante afstand wordt de afstand hierbij willekeurig. Wel is het van belang dat hierbij uiteindelijk alle banden aanwezig zijn binnen de fragmenten van een luidspreker.

Limitatie: Overvloeien frequentiebanden

Wanneer de delay nog niet gecompenseerd is tussen de luidsprekers, is er een kort moment waarop de signalen van de verschillende luidsprekers zich in de zelfde frequentiebanden bevinden. Wanneer dit gebeurt raken de overdracht-curves van de luidsprekers dus ook vervuild met het signaal van andere luidsprekers.

Om dit te voorkomen moet bij het bepalen van de audiofragmenten rekening worden gehouden met de vertraging. Het fragment moet zo ingeperkt dat er geen overlap tussen de signalen plaatsvind.

Een andere optie is om een elk fragment iets te verlengen en een fragment uit het midden van de opname van het fragment te pakken. Hoe veel het signaal hierbij verlengd moet worden moet hierbij goed afgestemd worden op de maximaal verwachtte vertraging tussen luidsprekers.

Limitatie: Signaal-ruis verhouding

Hoewel de delay-meting goed functioneert bij een lage signaal-ruis verhouding, doet de EQ meting dit niet. Voor het volledig optimaliseren van een audiosysteem is het dus vereist om het testsignaal met een goede signaal-ruis verhouding af te spelen.

3.7. Subconclusie Simulatie resultaten

Vanuit de opgestelde concept-werking zijn een aantal ideeën voortgekomen voor de optimalisaties. Voor de EQ betreffen dit een aantal methodes gebaseerd op matrixbewerkingen. Deze methodes zouden goed implementeerbaar zijn op een FPGA. Deze methodes zijn echter minder effectief en minder flexibel voor het optimaliseren.

De meest effectieve methode met een FIR-filter als resultaat, is het berekenen van het filter in het frequentiedomein en deze omzetten naar het tijddomein. Deze methode geeft ook de meeste flexibiliteit met betrekking tot het aanbrengen van correcties, zoals bandbreedte limieten, in het filter. Om deze redenen is gekozen om met deze methode verder te werken.

Voor het meten van de EQ is het ook van belang om een goed testsignaal te gebruiken. Hierbij is de gedetermineerde roze ruis naar voren gekomen. Door het instelbare spectrum is dit signaal geschikt voor het uitvoeren van EQ metingen.

Daarnaast bezit dit signaal een handige eigenschap die een goed te onderscheiden piek bij de autocorrelatie geeft. Hierdoor is het mogelijk om dit signaal te gebruiken voor delay-metingen.

Op basis hiervan is een methode ontwikkeld om deze delay te meten. Deze methode is vervolgens getest met behulp van simulaties. In deze simulaties zijn verschillende factoren getest die invloed kunnen hebben op deze metingen. De methode is hierbij verder bijgewerkt om met al deze verstoringen te blijven werken.

Aangezien zowel de EQ en delay werken op basis van het zelfde testsignaal, zijn deze samen geïntegreerd tot een enkele meting. Ook is dit aangepast om gelijktijdig op meerdere luidsprekers te werken. Hiervoor is het signaal opgedeeld in verschillende banden. Hierbij is gezorgd dat in elke frequentieband op een moment slechts één luidspreker actief is. Door het combineren van de verschillende spectra, kunnen uiteindelijk meerdere luidsprekers gelijktijdig voor EQ en Delay gecompenseerd worden.

4. Implementatie EQ-Algoritme

De concepten, zoals omschreven in hoofdstuk 3, zijn verder uitgewerkt tot een Proof of Concept. Dit is gedaan door het realiseren van een Real-Time implementatie van de EQ en Delay concepten. Dit hoofdstuk beschrijft deze implementatie samen met een aantal overwegingen voor een uiteindelijke implementatie.

4.1. Implementatie overwegingen

Bij het implementeren zijn afwegingen gemaakt over de invloed van bepaalde factoren en hoe dit mogelijk geïmplementeerd kan worden. Een aantal relevante factoren zijn hier verder toegelicht.

4.1.1. Processing delay

Voor het corrigeren van de EQ, moet de binnenkomende audio bewerkt worden. Hierbij ontstaat een vertraging. De grootte van deze vertraging is afhankelijk van het aantal filter-taps en de te corrigeren delay.

Wanneer er alleen audio afgespeeld wordt, vormt deze vertraging geen probleem. Wanneer de audio echter van een live-spreker komt of in combinatie met een video getoond wordt, kan dit een probleem vormen. Vanaf een bepaalde vertraging zal de luisteraar dit gaan opmerken

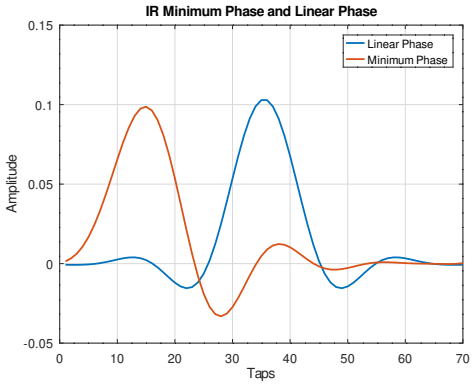
In het kader van televisie-uitzendingen, zijn meerdere onderzoeken gedaan naar video- en audio-synchronisatie. Hieruit volgen meerdere adviezen en standaarden met uiteenlopende specificaties.

De International Telecommunications Union zegt hierbij dat een vertraging pas merkbaar is vanaf 125 ms en dat 185 ms nog als acceptabel ervaren wordt [9]. Een ander voorbeeld is de European Broadcasting Union. Zij geven als advies een maximale vertraging van 60 ms [10].

Binnen Embedded Acoustics wordt vaak 15 meter, ofwel 50 ms, gehanteerd. Daarnaast is het bij Hifi, stereo en 3D audio van belang dat deze vertraging zeer klein is.

Minimum-Phase FIR

Eén van de methodes om de vertraging te minimaliseren is het gebruik van Minimum-Phase FIR-filters. Bij een standaard filter ligt de piek op het middelste filtercoëfficient. Bij een Minimum-Phase filter ligt deze meer richting het begin van het filter. Een voorbeeld hiervan is weergegeven in figuur 4.1. Hierdoor kan een kleinere vertraging gerealiseerd worden met een verder vergelijkbare filterkarakteristiek.



Figuur 4.1: Impulse response minimum phase en linear phase Low Pass Filter

Ter vergelijking zijn twee filters gegenereerd. De impuls response van deze filters is weergegeven in figuur 4.1. Beide filters hebben 4096 coëfficienten. De piek van het Linear-Phase filter ligt hierbij op het 2048^{e} coëfficient. Bij het Minimum-Phase filter ligt deze op het 27^{e} coëfficient. Bij een samplefrequentie van 48 kHz wordt hiermee de vertraging gereduceerd met 42 ms.

Het nadeel van deze Minimum-Phase filters is dat ze niet meer symmetrisch zijn. Hierdoor is het niet meer mogelijk het filter te optimaliseren door het gebruik van een Pre-Adder. Bij deze optimalisatie worden samples opgeteld voor de vermenigvuldiging, om zo het aantal vermenigvuldigingen te halveren.

Omdat dit niet mogelijk is bij een asymmetrisch filter, is meer rekenkracht vereist voor een filter van dezelfde lengte.

Omdat binnen dit Proof of Concept de absolute vertraging van het signaal niet van direct belang is, is deze methode niet meegenomen. Wanneer in een FPGA-implementatie de absolute vertraging wel van belang is, kan overwogen worden om deze methode te implementeren.

Weergave niveau

Met de gebruikte methode is het ook mogelijk om het verschil in weergave niveau tussen de luidsprekers te meten. Het zou dus ook mogelijk zijn om dit te compenseren.

Hierbij zouden mogelijk grote gain-factoren kunnen ontstaan in de EQ. Daarom zal het in de meeste gevallen verstandiger zijn om dit verschil op te lossen met de volumeregeling van de installatie. Om deze redenen is gekozen om deze functionaliteit voorlopig niet te implementeren.

Een tweede factor is de versterking van het totale filter. Door de menselijke perceptie van geluid kan het zijn dat het geluid harder of zachter gaat klinken wanneer de verhouding tussen hoog en laag veranderd, ondanks dat de totale energie gelijk blijft.

Om dit te compenseren kan gezorgd worden dat de A-gewogen [11] energie in het filter gelijk blijft. De A-weging compenseert hierbij voor het verschil in geluidniveau-beleving van het menselijk gehoor. In de praktijk zijn meerdere perceptieve modellen voorhanden om een indruk te krijgen van de over de beleving van luidheid van signalen.

Filter overlap factor

Bij het berekenen van een filter kan het nieuw berekende filter direct gebruikt worden. Dit maakt het filter wel gevoelig voor meetfouten of verstoringen in de meting. Om deze reden is ervoor gekozen om een instelbare overlap te gebruiken tussen het oude en nieuwe filter.

Hiermee kan een factor gekozen worden die de invloed van het oude en nieuwe filter instelt. Wanneer veel invloed van het oude filter gebruikt wordt, zal het filter minder gevoelig voor verstoringen zijn. Het nadeel hierbij is dat het filter langzamer reageert.

Wanneer veel invloed van het nieuwe filter gebruikt wordt, is het effect tegenovergesteld. Het filter zal zich snel aanpassen, maar is gevoeliger voor verstoringen. Deze factor kan afgesteld worden aan de hand van de meetomstandigheden.

4.2. Software implementatie

Vanuit Octave zijn de mogelijkheden voor Real-Time audio gelimiteerd. Om deze reden is gekozen om de functionaliteit in een andere programmeertaal uit te werken.

4.2.1. Programmeertaal

Om de Octave code makkelijk om te zetten, is het de wens om een programmeertaal te gebruiken waar zo veel mogelijk van dezelfde wiskundige functies in zitten. Daarnaast is het van belang dat er op een simpele manier een UI gemaakt kan worden waarin plots weergegeven kunnen worden. Ook is enige bekendheid met de programmeertaal van belang.

Uit deze eisen volgen twee primaire mogelijkheden:
- C++ met uitbreiding van de Armadillo library [12]
- Python met de NumPy [13] en SciPy [14] libraries

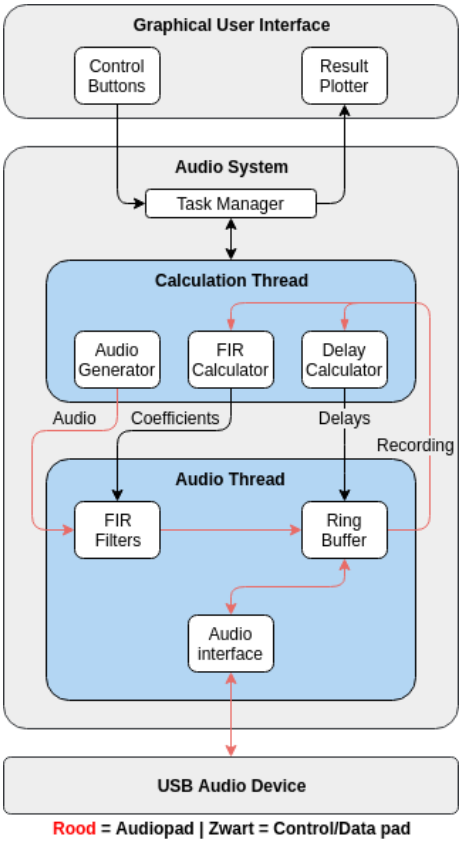
Met beide opties kan de basisfunctionaliteit gerealiseerd worden. Het creëren van een UI binnen C++ met plot-mogelijkheden is echter complexer dan binnen Python. Om deze reden is er gekozen om de software implementatie verder uit te werken binnen Python.

4.2.2. Architectuur

Vanuit de reeds bestaande Octave code is de architectuur opgesteld voor een real-time implementatie. Deze wordt onderstaand verder beschreven.

Overzicht

In figuur 4.2 is de globale software-architectuur weergegeven.



Figuur 4.2: Globale software architectuur

Gebruikersinterface

De gebruikersinterface is gerealiseerd door middel van PyQt [15] in combinatie met MatPlotlib [16].

PyQt wordt hierbij gebruikt voor het genereren van de interface. Matplotlib wordt gebruikt voor het plotten van de berekening-resultaten. De interface wordt verder toegelicht in onderdeel 4.3.

Multithreading

Binnen de applicatie wordt gewerkt met meerdere threads voor de GUI, berekeningen en audio. Dit is gedaan zodat de real-time audio taak gescheiden kan worden van de rest van het programma, tevens voor de toepassing op de FPGA. Dit is eerst gedaan door middel van het PyQt multithreading framewerk.

In standaard Python is er echter een limitatie die ervoor zorgt dat slechts één Python-thread tegelijk uitgevoerd kan worden. Ook kan hierdoor geen gebruik gemaakt worden van meerdere processor-cores. Daarnaast is het niet mogelijk om een prioriteit aan threads mee te geven.

Dit alles zorgt ervoor dat ondanks dat PyQt multithreading gebruikt is, de zware rekentaken nog steeds zorgen voor een geblokkeerde UI en een audio-interface die zijn timing eisen niet haalt.

Als oplossing voor dit probleem beschikt Python over het Multiprocessing-framewerk. Dit framewerk omzeilt de limitatie van standaard python, waardoor er wel meerdere processen gelijktijdig, en op meerdere processor-cores, uitgevoerd kunnen worden.

De Audio Thread en Calculation Thread zijn gerealiseerd met een multiprocessing thread. Doordat deze volledig gescheiden werken, is het niet langer een probleem om aan de real-time audio timing te voldoen en de GUI responsief te houden. De andere threads zijn gerealiseerd met de PyQt Threads omdat deze over makkelijkere communicatie-methodes beschikken.

Inter-process communication
De communicatie tussen de GUI en de Task Manager wordt verzorgd door Qt Signals en Slots. Dit is een Event-gebaseerd communicatie systeem waarbij een signaal, met eventuele data, verzonden wordt. Dit signaal wordt vervolgens op een andere plek opgevangen. Aan de hand hiervan wordt een functie aangeroepen. Dit versimpeld de asynchrone communicatie tussen de GUI en Task Manager.

De verdere communicatie verloopt via Queues van het Multiprocessing framewerk. Hiermee wordt onder andere de audiodata uitgewisseld tussen de verschillende threads.

Audio Thread

De audio-thread bestaat uit drie onderdelen

FIR-filters
Wanneer de audio-thread detecteert dat er audio in de Queue beschikbaar is, wordt deze ingeladen. De data gaat vervolgens naar de FIR-filters. Hier wordt elk kanaal met de bijbehorende coëfficienten gefilterd. Vervolgens wordt de gefilterde data doorgegeven aan de Ringbuffer.

Ringbuffer
De Ringbuffer heeft in deze applicatie twee toepassingen. Allereerst vormt deze buffer de tussenstap tussen audio-generatie en het afspelen van de audio. Hierbij wordt de normale Ringbuffer-functionaliteit gebruikt.

Daarnaast wordt met deze ringbuffer ook de delay-compensatie gerealiseerd. Dit wordt gedaan door met het gewenste aantal samples vertraging naar de buffer te schrijven.

De data uit deze buffer wordt vervolgens klaargezet in een Queue om af te spelen.

Audio Interface
Voor het afspelen en opnemen van audio is gebruik gemaakt van het Python SoundDevice framewerk [17]. Via dit framewerk wordt een I/O Stream opgezet waarmee gelijktijdig audio afgespeeld en opgenomen kan worden. Dit gebeurt op basis van callbacks. Vanuit de onderliggende systemen wordt een Python-callback aangeroepen wanneer er nieuwe audio-data nodig is. In deze callback wordt er een audio-blok uit de Queue gehaald. Deze wordt vervolgens naar het onderliggende systeem doorgestuurd en daar uiteindelijk afgespeeld. De opgenomen data wordt door de onderliggende systemen teruggegeven in deze callback. Deze opgenomen data wordt vervolgens in een Queue geplaatst om later door te geven aan de Calculation Thread.

Calculation Thread

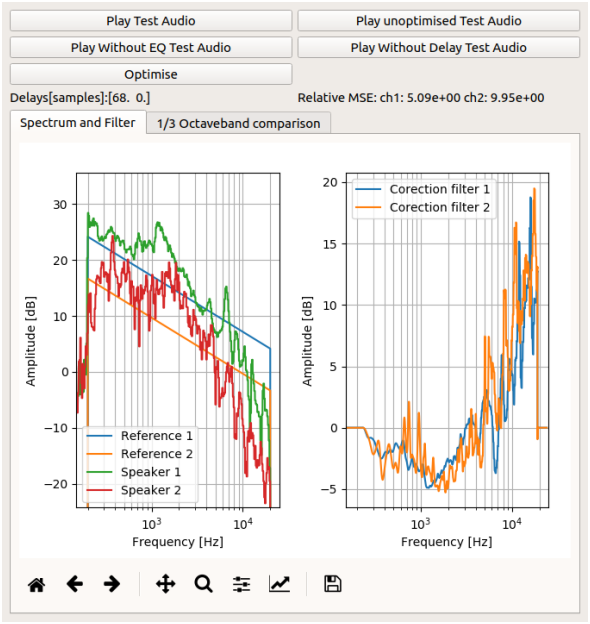
De opgenomen audio-data komt uiteindelijk in de Calculation Thread terecht. Hier zijn ook de gegenereerde audiofragmenten bewaard. De opname en originele signalen worden vervolgens met elkaar vergeleken. Aan de hand van de methodes zoals omschreven in hoofdstuk 3, worden vervolgens nieuwe filters en delay-tijden berekend. Deze worden vervolgens weer doorgegeven aan de Audio Thread. Bij het genereren van een nieuw filter, wordt het oude filter als basis genomen. Het oude en het nieuwe filter worden met een instelbare factor gecombineerd. Door deze vorm van middelen wordt het filter minder afhankelijk van meetfouten.

4.3. Applicatie overzicht

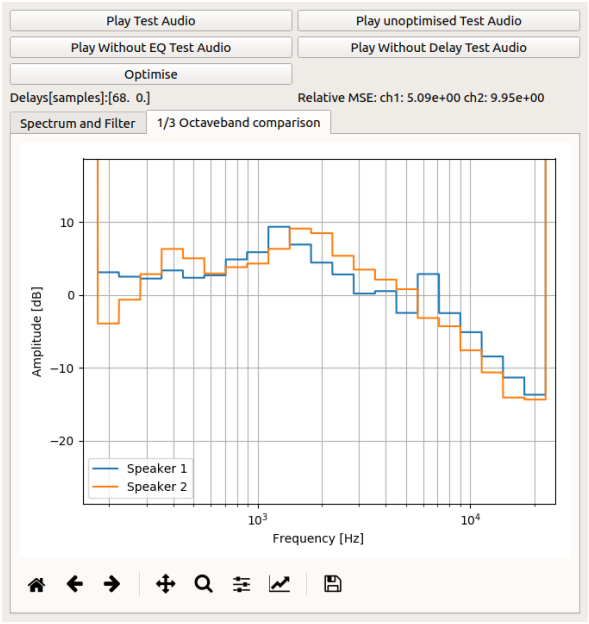
Al deze functionaliteiten samengevoegd, resulteren uiteindelijk in de gebruikersinterface die weergegeven is in figuur 4.3 en 4.4. Hierin zijn knoppen voor de aansturing verwerkt en grafieken voor de visualisatie voor de meet-data.

Door middel van de knoppen kan een optimalisatie gestart worden, of testsignaal afgespeeld worden. Dit testsignaal betreft een spraaksignaal om de geluidkwaliteit te kunnen beoordelen. Dit testsignaal kan met en zonder optimalisatie afgespeeld worden. Ook kan er afgespeeld worden met alleen EQ of alleen delay optimalisatie.

In de grafieken zijn de actuele FFTs van de opname weergegeven, het actueel toegepaste FIR-filter en een $\frac{1}{3}$ octaafband vergelijking. In deze vergelijking zijn de opname en referentie opgedeeld in $\frac{1}{3}$ octaafbanden en vervolgens van elkaar afgetrokken. Hierin kan dus gezien worden hoe dicht het signaal nadert tot een ideale overdracht. In het ideale geval zal deze lijn vlak blijven en op 0 dB liggen.



Figuur 4.3: Gebruikers interface pagina met signaal FFT (links) en correctiefilters (rechts)



Figuur 4.4: Gebruikers interface pagina met $\frac{1}{3}$ octaafband vergelijking

5. Verificatie EQ-Algoritme

In dit hoofdstuk wordt de Python-applicatie zoals omschreven in hoofdstuk 4 gevalideerd. Hierbij wordt gekeken naar de verbetering die behaald worden door het toepassen van de optimalisatie.

5.1. Meetopstelling

In dit onderdeel worden de fysieke opstelling en gebruikte apparatuur voor de metingen beschreven.

5.1.1. Apparatuur

Bij het uitvoeren van de metingen is de apparatuur gebruikt zoals weergegeven in tabel 5.1.

Figuur 5.1: Gebruikte apparatuur testopstelling

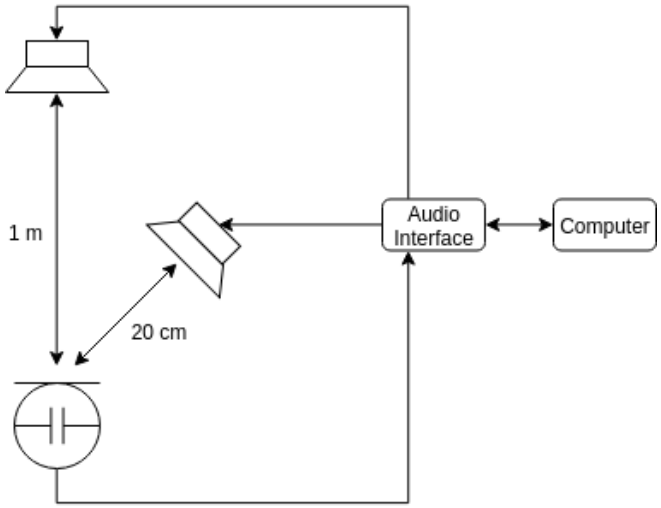
Apparaat	Type
USB Audio Interface	Maya44 (4ch)
Microfoon	Electret microfoon
Luidspreker 1	Behringer Behritone C5A
Luidspreker 2	Philips Box410 30-watt

Bij deze opstelling wordt de aanname gedaan dat de overdracht van het afspeel- en opname-systeem vlak is. Hoewel dit in de praktijk niet zo is, vormt het voor deze metingen geen probleem. Dit is omdat toegewerkt wordt naar een relatieve verbetering.

Wanneer een absolute verbetering gewenst is, moet de overdracht van de gebruikte apparatuur bepaald worden. Hiervoor kan vervolgens gecompenseerd worden.

5.1.2. Opstelling

De opstelling voor de metingen is weergegeven in figuur 5.2. Op de computer is de eerder beschreven software gebruikt om de audio te optimaliseren.



Figuur 5.2: Testopstelling verificatie metingen

5.2. Verbeteringsfactor

Om de verbeteringen inzichtelijk te maken, zijn een aantal methoden gebruikt. De methoden worden hier verder beschreven

5.2.1. FFT Error definitie

Voor het bepalen van de afwijking op frequentiegebied tussen het testsignaal en het opgenomen signaal, wordt de Mean Squared Error gebruikt. Hierbij wordt binnen de bandbreedte van het testsignaal de afwijking tussen de FFT van het testsignaal en de opname bepaalt. Dit gebeurt op basis van de energie in de FFT.

Deze methode op zichzelf is sterk afhankelijk van het volume van de signalen. Bij een hoger volume zou de afwijking groter worden. Om deze reden wordt de MSE als laatste stap gedeeld door de gekwadrateerde gemiddelde waarde van de energie van de FFT van het testsignaal. Dit is weergegeven in vergelijking 5.1

$$MSE_{relatief} = \frac{\frac{1}{n} \cdot \sum_{i=1}^{n} (|\, ref_i \,|^2 - |\, input_i \,|^2)^2}{\frac{1}{n} \cdot \sum_{i=1}^{n} (|\, ref_i \,|^4)} \tag{5.1}$$

Op deze manier is de MSE relatief en niet meer afhankelijk van de volumes van de signalen. Dit vereenvoudigd het vergelijken van verschillende opstellingen

5.2.2. Octaafband afwijking

Een tweede manier waarop gekeken wordt naar de afwijking tussen de FFTs, zijn octaafbanden. Beide signalen worden onderverdeeld in $\frac{1}{3}$ octaafbanden en vervolgens van elkaar afgetrokken. Dit resulteert in een lijn die in het ideale geval over de gehele bandbreedte van het testsignaal nul is. Deze lijn geeft een indicatie over eventuele grove afwijkingen.

5.2.3. Delay Error

Om te controleren of de toegepaste delay klopt, wordt een stereo impuls vanuit het Python programma afgespeeld. Tussen de twee kanalen is hierbij een kleine vertraging toegevoegd zodat de pulsen goed van elkaar te onderscheiden zijn. Vervolgens wordt de opname vanuit het Python programma opgeslagen. Deze opname wordt vervolgens met behulp van Audacity geanalyseerd. Bij deze analyse wordt gekeken naar de vertraging tussen de luidsprekers en of deze overeenkomt met de gemeten vertraging uit het Python programma.

5.3. Verificatie tests

De gebruikte methodes en uiteindelijke resultaten van de verificatie tests zijn hieronder weergegeven.

5.3.1. Test methode

Elke test is uitgevoerd met behulp van de testopstelling zoals omschreven in hoofdstuk 5.1. Hierbij zijn de verbeteringsfactoren zoals omschreven in hoofdstuk 5.2 waargenomen.

Binnen de software is de optimalisatie-stap vijf keer uitgevoerd voor twee luidsprekers met gelijktijdige EQ- en delay-meting. Dit is gedaan voor verschillende aantallen filtertaps: 1024, 2048, 4096, 8192. De overige gebruikte parameters zijn weergegeven in tabel 5.1.

Tabel 5.1: Gebruikte parameters verificatie tests

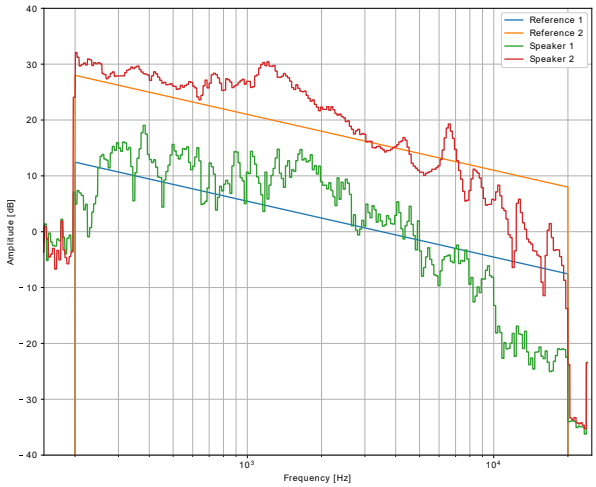
Parameter	Waarde
Aantal kanalen	2
Ruis frequentie range	200 - 20000 Hz
Optimalisatie range	250 - 19000 Hz
Ruis fragment-lengte	1,5 s
Totale signaalduur	3 s
Ruis octaafbanden	1/48
Filter smoothing octaafbanden	1/36
Filter overlap factor	0,75

5.3.2. EQ

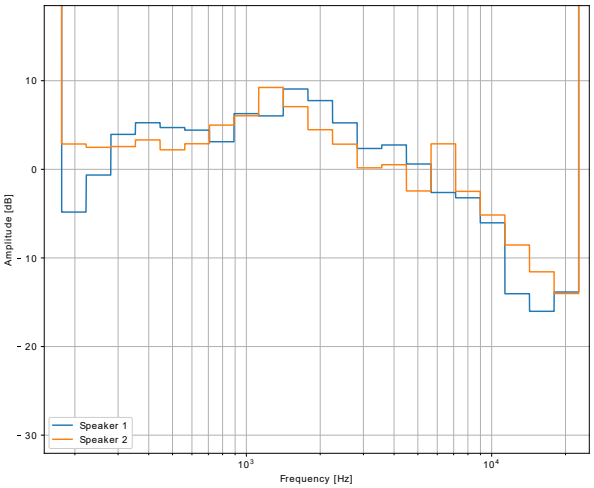
In dit onderdeel zijn de resultaten van de optimalisatie weergegeven. Hierin zijn alleen de gevallen voor 1024 en 8192 taps weergegeven. De resultaten voor de andere filtertaps zijn te raadplegen in *Bijlage B: Verificatie Tests*. Hierbij zijn ook de bijbehorende FIR correctie filters weergegeven.

Baseline

In figuur 5.3 en 5.4 zijn de baseline metingen weergegeven voor de FFT en octaafbanden. Hierin zijn afwijkingen te zien van ruim ± 10 dB. De MSE in deze situatie is 8,80 voor luidspreker 1 en 5,02 voor luidspreker 2.



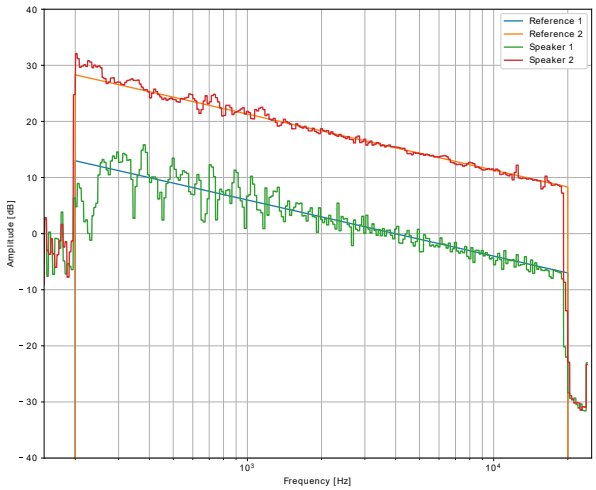
Figuur 5.3: FFT resultaat zonder optimalisatie (luidspreker 1: Groen; luidspreker 2: Rood)



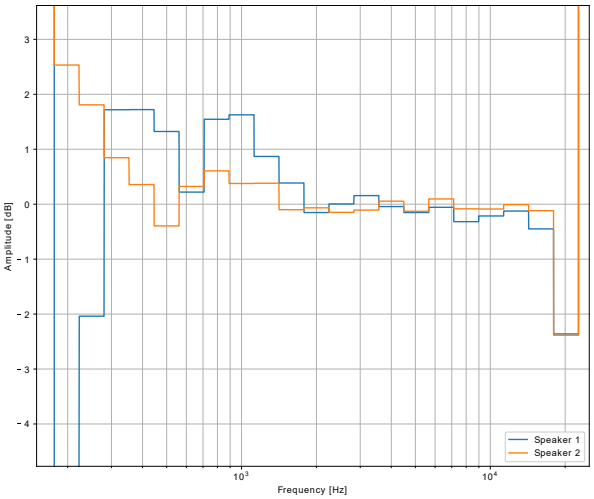
Figuur 5.4: $\frac{1}{3}$ octaafband resultaat zonder optimalisatie (luidspreker 1: Blauw; luidspreker 2: Oranje)

Optimalisatie met 1024 taps filter

De optimalisatie is vijf keer toegepast met een FIR-filter van 1024 taps. De resultaten hiervan zijn weergegeven in figuur 5.5 en 5.6. Hier is te zien dat de maximale afwijking tussen 300 Hz en 18 kHz gereduceerd is tot +2 dB en -0,5 dB. De MSE in deze situatie is 0,55 voor luidspreker 1 en 0,056 voor luidspreker 2. De MSE is hierbij dus met een factor 16 en factor 90 gereduceerd.



Figuur 5.5: FFT resultaat met 1024 taps optimalisatie (luidspreker 1: Groen; luidspreker 2: Rood)

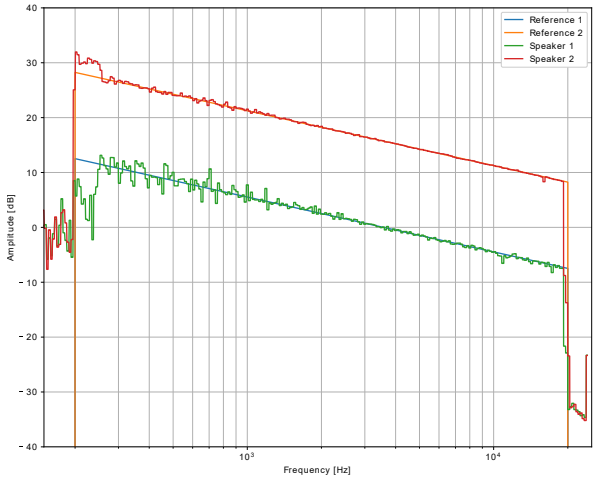


Figuur 5.6: $\frac{1}{3}$ octaafband resultaat met 1024 taps optimalisatie (luidspreker 1: Blauw; luidspreker 2: Oranje)

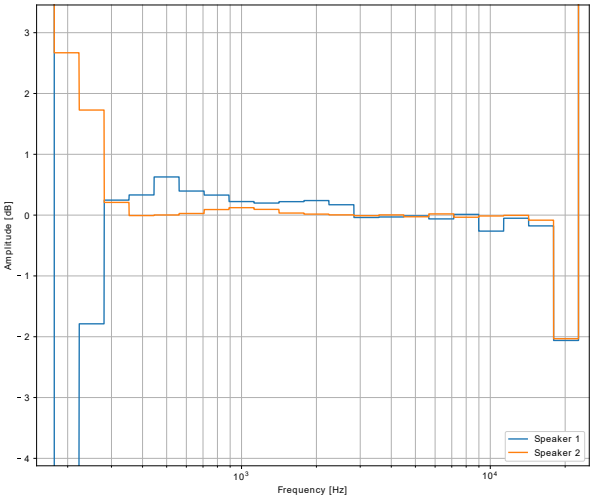
Optimalisatie met 8192 taps filter

De optimalisatie is vijf keer toegepast met een FIR-filter van 8192 taps. De resultaten hiervan zijn weergegeven in figuur 5.7 en 5.8. Hier is te zien dat de maximale afwijking tussen 300 Hz en 18 kHz gereduceerd is tot +0,6 dB en -0,2 dB. De MSE in deze situatie is 0,095 voor luidspreker 1 en 0,014 voor luidspreker 2. De MSE is hierbij dus met een factor 93 en factor 359 gereduceerd ten opzichte van de baseline.

Ten opzichte van de 1024 taps FIR is de situatie verbeterd met een factor 6 en een factor 4.



Figuur 5.7: FFT resultaat met 8192 taps optimalisatie (luidspreker 1: Groen; luidspreker 2: Rood)



Figuur 5.8: $\frac{1}{3}$ octaafband resultaat met 8192 taps optimalisatie (luidspreker 1: Blauw; luidspreker 2: Oranje)

Waarnemingen

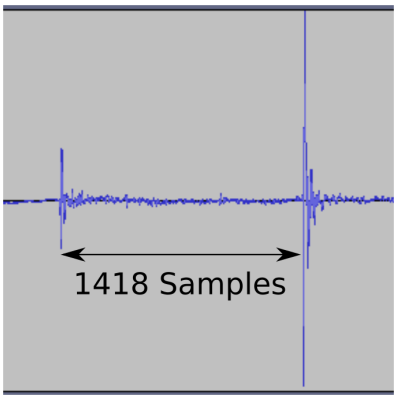
In deze tests is het opvallend dat luidspreker 2 bij 1024 taps een grotere verbetering ondervind dan luidspreker 1. Dit wordt veroorzaakt doordat het spectrum van luidspreker 2, zoals weergegeven in figuur 5.3, gelijkmatiger verloopt. luidspreker 1 heeft hierbij vele sterke overgangen. Deze steile overgangen vereisen meer filtertaps om te corrigeren. om deze reden is luidspreker 2 beter te optimaliseren dan luidspreker 1.

Hieruit blijkt dus ook dat het benodigd aantal filtertaps, afhankelijk is van welke luidspreker geoptimaliseerd moet worden.

5.3.3. Delay

De delay metingen zijn gelijktijdig uitgevoerd met de EQ optimalisaties. Hierbij werd in alle gevallen na 3 optimalisaties een delay van 22 samples aangegeven. Dit bleef hierna consistent.

Voor de validatie is de delay gecontroleerd met Audacity. De opname van eerder genoemde impuls is weergegeven in figuur 5.9. Hieruit volgde een delay van 1418 samples. De delay in het originele bestand is 1440 samples. De gemeten delay in Audacity is dus $1440 - 1418 = 22$ samples. Dit bevestigd dus de resultaten die uit het Python programma volgden.



Figuur 5.9: Opname vertraagde impuls

Waarnemingen

Gedurende de meting was het opvallend dat de delay-tijd de eerste drie cyclussen niet correct was. Deze varieerde tussen de 17 en 21 samples.

In hoofdstuk 3.5.3 is een methode geïntroduceerd waarbij de correlaties van de signalen gefilterd werden om de pieken beter te detecteren. Deze methode is in de Python applicatie niet geïmplementeerd. Mogelijk kan met deze verbeterde methode vanaf de eerste meting al een nauwkeuriger resultaat bepaald kunnen worden.

6. Vernieuwing hardware

In dit hoofdstuk worden de benodigdheden en voorstellen voor vernieuwingen van de hardware voor de audiosignaalprocessor gedaan. Dit zal gebeuren aan de hand van de benodigde specificaties die in dit hoofdstuk zullen worden uitgewerkt. Deze specificaties worden opgesteld aan de hand benodigdheden van het EQ-algoritme zoals besproken in hoofdstuk 4.

6.1. Digitale uitbreiding via Dante

Voor de nieuwe audiosignaalprocessor is het gewenst naast analoge IO ook digitale IO te bieden. Op de huidige audiosignaalprocessor wordt dit gerealiseerd door middel van Cobranet [19]. Cobranet geeft de mogelijkheid om via een netwerkconnectie audio in- en uitgangen te realiseren. Deze technologie wordt echter steeds minder toegepast.

Tabel 6.1: Dante specificaties (Maximum) [18]

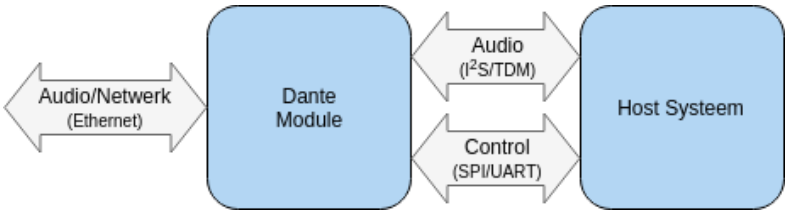
Specificatie	Waarde
Minimum latency	150 μs
Maximum channels per link	1024 (512 in; 512 uit)
Maximum sampling rate	192 kHz
Maximum bit-diepte	32 bits

Om deze reden is gekozen om onderzoek te doen naar het uitbreiden van de IO door middel van Dante [18]. Dante is één van de vele protocollen waarmee audio via een standaard ethernet netwerk verstuurd kan worden. Het is zelfs mogelijk om Dante gedeeltelijk te mengen met normaal netwerk verkeer. De specificaties van dit protocol zijn weergegeven in tabel 6.1.

Dit protocol is gekozen omdat het in de actuele markt een goede compatibiliteit verzorgt. Dit komt doordat Dante op steeds meer plekken wordt ingezet. Voor de nieuwe audiosignaalprocessor is het de wens om minimaal 48 digitale ingangen en 48 digitale uitgangen te hebben. Dit valt ruim binnen het maximaal aantal kanalen van Dante.

6.1.1. Algemene systeemarchitectuur Dante

Voor het integreren van Dante in een eigen systeem zijn verschillende opties beschikbaar. Allereerst zal kort de plaats van een Dante module binnen een systeem worden besproken.



Figuur 6.1: Algemene systeemarchitectuur Dante

In figuur 6.1 is een blokdiagram van een generieke Dante implementatie weergegeven. Hierin is te zien dat de Dante Module de brug vormt tussen het Audio/Netwerk verkeer en de digitale audio binnen het systeem. Deze digitale audio interface wordt kort toegelicht in hoofdstuk 6.1.2.

Met betrekking tot de plek en functionaliteit in het systeem, is elke Dante module identiek. De verschillen tussen de modules zitten in het aantal kanalen en de audiokwaliteit die de modules kunnen leveren. Andere onderscheidende functies worden vermeld in de beschrijvingen van de modules in hoofdstuk 6.1.3.

6.1.2. Audio Interface

Voor het overdragen van de audio wordt door alle Dante modules gebruikt gemaakt van I^2S/TDM. De originele specificatie voor I^2S is opgesteld door Philips [20] (nu NXP). Deze specificatie wordt echter niet actief bijgehouden. Hierbij zijn er variaties ontstaan op het I^2S protocol.

Eén van de grotere variaties hierbij is TDM (Time Division Multiplexing). Normaal ondersteund I^2S twee audio kanalen. Een linker en rechter kanaal. Het onderscheid wordt hierbij aangegeven door middel van een

Links/Rechts kloksignaal.

TDM maakt gebruik van dit Links/Rechts kloksignaal om de start van een segment aan te geven. Vervolgens worden op de data lijn de audiosamples van meerdere kanalen achter elkaar verstuurd. Door middel van deze methode kunnen er meer dan twee kanalen gebruikt worden via de interface.

Met de Dante modules kan gebruikt gemaakt worden van I^2S of TDM. Naast de standaard interface signalen bieden de Dante modules hierbij ook een 'Master Clock' aan. Deze kan gebruikt worden voor het synchroniseren van de verschillende elementen binnen het systeem.

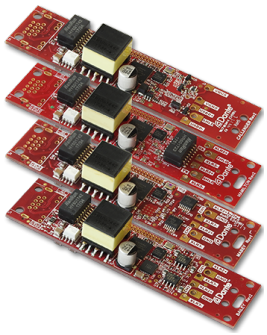
6.1.3. Systeem implementaties

De modules om Dante te integreren, variëren van volledige printplaten tot FPGA hardware-beschrijvingen. De verschillende opties zullen hieronder besproken worden. Dit betreffen alle relevantie opties van de originele fabrikant die op het moment van schrijven beschikbaar zijn.

Na een overzicht van de modules, worden de voor- en nadelen van elke module tegen elkaar afgewogen. De opties die relevant zijn binnen de gewenste toepassing, zijn tegen elkaar uitgezet in tabel 6.2.

Adapter Module

Eén van de beschikbare opties is de Dante Adapter Module [21] zoals weergegeven in figuur 6.2. Deze module beschikt over één netwerkaansluiting met ondersteuning voor Power over Ethernet. Daarnaast beschikt hij over maximaal twee audio ingangen en twee audio uitgangen. Vanwege het lage aantal in- en uitgangen, is deze module niet geschikt in deze toepassing.



Figuur 6.2: Adapter module [21]

Brooklyn II

De Dante Brooklyn II [22] is een module met daarop de benodigde hardware om deel te nemen aan een Dante netwerk. Deze module is weergegeven in figuur 6.3. De module beschikt over een Mini-PCI connector waarmee de verbinding met de rest van het systeem wordt gemaakt.

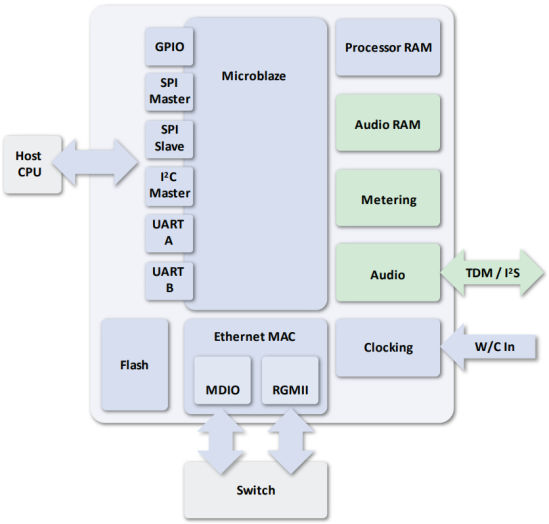
Figuur 6.3: Brooklyn II [22]

Voor de processing wordt gebruik gemaakt van een Xilinx Spartan-6 FPGA waar ook een soft-core processor op draait. Door middel van deze hardware kan het systeem 64 audio ingangen en 64 audio uitgangen realiseren(Bij een sample rate van 48 kHz). De maximale sample rate is 192 kHz en de bit-diepte 32 bits.

Op de soft-core van het systeem, draait een variant van Linux. Door middel van de bijbehordende SDK is het mogelijk om applicaties binnen deze omgeving uit te voeren. Verdere mogelijkheden voor het implementeren van applicaties in de FPGA worden niet door de fabrikant aangegeven.

Voor de connectie met het netwerk beschikt de module over een RGMII/MII interface. Hiervoor is verder een Ethernet PHY chip en netwerkconnector nodig om daadwerkelijk te verbinden met het netwerk.

Een blokdiagram van de systeem-architectuur is weergegeven in figuur 6.4.



Figuur 6.4: Dante Brooklyn II & HC blokdiagram [22]

HC

De Dante HC [23] is vergelijkbaar met de Brooklyn II. Dit betreft ook een Xilinx Spartan-6 FPGA. Een belangrijk verschil hierbij is dat bij de HC enkel de FPGA geleverd wordt en geen ondersteunende hardware. De FPGA wordt geleverd in de vorm van een 676 pins BGA.

Ook is het maximaal aantal kanalen anders. De HC kan 512 ingangen en 512 uitgangen ondersteunen (Bij een sample rate van 48 kHz). De maximale sample rate is opnieuw 192 kHz. Ook is het door middel van de beschikbare netwerk interfaces mogelijk om een redundant netwerk op te zetten met deze module. Aangezien de HC op veel vlakken lijkt op de Brooklyn II, wordt hierop de zelfde blokdiagram van toepassing zoals weergegeven in figuur 6.4. Ook op deze FPGA draait een Soft-Core processor waarop applicaties uitgevoerd kunnen worden.

Voor het integreren van deze FPGA binnen een eigen product heeft Dante design resources en engineers beschikbaar om begeleiding te bieden gedurende het ontwerpproces.

Ultimo

De Dante Ultimo [24] is op de zelfde manier opgezet als de Dante HC. Ook dit betreft een enkele chip, in de vorm van een 144 pins BGA, die binnen een eigen ontwerp geïntegreerd kan worden. De capaciteiten van deze chip zijn echter vele malen lager. De Ultimo ondersteund maximaal 4 ingangen en 4 uitgangen (Bij een sample rate van 48 kHz). De maximale sample rate is 96 kHz.

Door het lage aantal in- en uitgangen, is dit product niet geschikt voor deze toepassing.

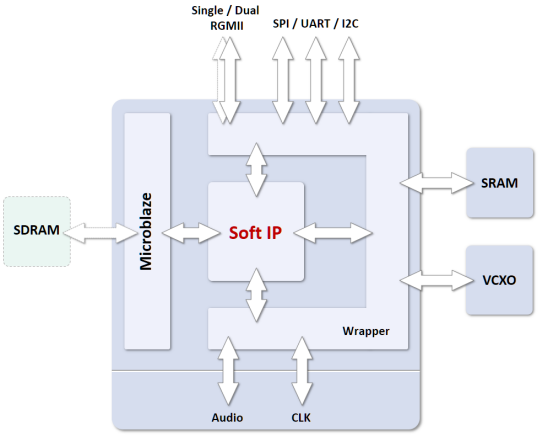
Broadway

De Dante Broadway is de tussenstap tussen de Ultimo en de Brooklyn II. Ook dit is een enkele chip in de vorm van een 256 pins BGA. Hierin zijn de benodigde faciliteiten aanwezig om met een Dante netwerk verbinding te maken. Deze chip geeft de mogelijk voor 16 ingangen en 16 uitgangen (Bij een sample rate van 96 kHz). Hoewel dit niet het aantal gewenste kanalen is, kan deze chip een goed alternatief zijn wanneer een kleinere variant van de audiosignaalprocessor gewenst zou zijn.

IP Core

Wanneer volledige integratie in een reeds aanwezige FPGA nodig is, is het mogelijk om gebruik te maken van de Dante IP Core [25]. Deze IP Core kan geïmplementeerd worden op FPGA's uit de Xilinx Spartan-6 of Artix-7 serie. Door gebruik te maken van een IP Core, is het mogelijk om een oplossing te creëren die exact aansluit bij de behoeften van het systeem. Dit is mogelijk doordat de gehele IP Core schaalbaar is.

Met de IP Core is het mogelijk om maximaal 512 ingangen en 512 uitgangen te realiseren (bij een sample rate van 48 kHz). De maximaal haalbare sample rate is 192 kHz. Een blokdiagram van het systeem is weergegeven in figuur 6.5. Ook op deze FPGA draait een Soft-Core processor waarop applicaties uitgevoerd kunnen worden.



Figuur 6.5: Dante IP Core architectuur [25]

Bij deze IP Core komt ook een support pakket. Dit bestaat uit advies en toegang tot Reference Projects en designs voor het implementeren van zowel software als hardware.

6.1.4. Afweging oplossingen

De relevante oplossingen zijn weergegeven in tabel 6.2. Voor elk van deze oplossingen is externe hardware nodig. Bij de Brooklyn II gaat dit alleen om de ethernet en voeding. Voor de andere twee moeten ook alle faciliteiten voor het functioneren van de chip gerealiseerd worden. Daarnaast moet er natuurlijk ook hardware aanwezig zijn die de audio in digitaal formaat van het Dante netwerk kan verwerken.

Tabel 6.2: Overzicht Dante oplossingen

Oplossing	Aantal kanalen	Sample Rate @max kanalen	Sample Rate @48x48	Hardware
IP Core	512x512	48 kHz	192 kHz	Implementeren in Xilinx Spartan-6 of Artix-7
HC	512x512	48 kHz	192 kHz	676 pins BGA
Brooklyn II	64x64	48 kHz	48 kHz	Mini PCI Card edge module

Bij het kiezen voor een module komen drie aspecten voornamelijk aan bod:
- Voldoet de oplossing aan de gewenste specificatie?
- Integratie gemak van de oplossing
- Kosten van de oplossing

De oplossingen uit tabel 6.1 voldoen allen aan de gestelde specificaties voor de Dante uitbreiding. Er kan dus verder worden gekeken naar het gebruiksgemak.

Qua integratie gemak zal de Brooklyn II het makkelijkst zijn om in een product toe te passen. Aangezien dit een kant en klare module is waar enkel de voeding, communicatie interfaces en ethernet bij gerealiseerd moet worden. Doordat het verdere ontwerp al aanwezig is op de module, zal dit de ontwikkeltijd verkorten.

Bij de andere twee opties zal meer hardware ontwikkeling komen kijken. Dit kan de ontwikkeltijd dan ook langer maken. In ruil hiervoor is de oplossing wel flexibeler. Met name wanneer gebruik gemaakt wordt van de IP Core. Het is ook mogelijk deze IP Core te combineren met de rest van de audio-processing applicatie. Hierbij zou dan slechts één FPGA gebruikt hoeven te worden. Hierbij moet wel onderzocht worden of er naast de Dante implementatie genoeg ruimte over blijft op de FPGA voor de audio-processing applicatie.

Het laatste aspect is de prijs van de verschillende oplossingen. Deze prijzen zijn echter alleen op aanvraag beschikbaar. Om deze reden zijn er op het moment nog geen prijzen bekend. Op basis hiervan kan dus geen afweging gemaakt worden.

Advies

Met het oog op integratie gemak en een zo kort mogelijke ontwikkeltijd, komt de Brooklyn II module het meest in aanmerking. Doordat deze module 'Plug and Play' is, gaan er minder ontwerp werkzaamheden gepaard met de integratie. Verder sluit deze module goed aan bij de wensen voor de audiosignaalprocessor, zonder dat de module te veel functionaliteiten en kanalen bied.

6.2. Filtering

In dit onderdeel worden de vereisten besproken voor het filteren van de audiosignalen.

6.2.1. Fixed Point/Floating Point

Een belangrijke overweging bij het uitvoeren van digitale signaalbewerking, is de overweging voor Fixed Point of Floating Point variabelen. Aan beide typen variabelen zitten voor en nadelen verbonden.

Het doel hierbij is om een formaat te kiezen waarbij de ruis, die inherent is aan de afrondingsfouten in de berekening, zo min mogelijk invloed heeft op het eindsignaal.

Floating Point

De notatie voor Floating Point variabelen is vergelijkbaar met de wetenschappelijke notatie van getallen. Bijvoorbeeld: $3,75 \cdot 10^3$. Een aantal bits binnen de variabele is hierbij toegewezen aan de 'Mantissa', ofwel het getal ($3,75$ in dit voorbeeld). Het andere deel van de variabele wordt toegewezen aan het 'Exponent'. In dit geval de 3. De verdeling voor het aantal Mantissa-bits en het aantal Exponent-bits is afhankelijk van de gebruikte standaard.

Het voordeel van deze notatie is dat het mogelijk is om een grote range aan getallen weer te geven. Hierdoor heeft een Floating Point variabele een grote dynamic range. Voor een IEEE-754 [26] 32 bits variabele is dit een range van $1529\,dB$. Voor een standaard 32 bits integer is dit slechts $193\,dB$. Een ander voordeel van deze grote range is dat er niet snel overflows op zullen treden bij berekeningen. Hierdoor hoeft bij het rekenen minder goed opgelet te worden voor deze overflows. Dit maakt het ontwerp van filter algoritmen makkelijker. Wel is het zo dat hoe groter het getal wordt, hoe meer detail verloren gaat.

De afrondingsfouten die bij berekeningen met Floating Point variabelen optreden, zitten in het achterste deel van de Mantissa. Doordat deze Mantissa vervolgens nog vermenigvuldigd wordt met het exponent, schaalt deze afrondingsfout mee met de grootte van het getal.

De grote dynamic range wordt ook aangedragen als limitatie van de Floating Point variabele [27]. Wanneer er een klein getal wordt opgeteld bij een groot getal, kan het kleine getal verloren gaan doordat er niet genoeg resolutie aanwezig is in de Floating Point representatie.

Eén van de belangrijkste nadelen is de hoeveelheid logica die nodig is voor een Floating Point reken-unit (ALU). Doordat de variabele complexer is, wordt de logica dit ook. Hoewel de exacte hoeveelheid logica ook sterk afhankelijk is van de gewenste mogelijkheden van ALU, is de benodigde hoeveelheid logica altijd groter dan dat van een Fixed Point ALU met vergelijkbare mogelijkheden.

Met het oog op integratie binnen een FPGA zal onderzocht moeten worden of het haalbaar is om genoeg Floating Point hardware te genereren voor het filteren van alle kanalen.

Fixed Point

Zoals eerder gemeld, is de dynamic range van een Fixed Point variabele kleiner dan die van een Floating Point. Daarentegen heeft een Fixed Point variabele geen problemen met het optellen van grote bij kleine getallen. Wel bestaat er een groter risico op overflow wanneer de filters niet correct ontworpen zijn.

Ook zijn de afrondingsfouten bij Fixed Point variabelen niet afhankelijk van de variabele grootte. Dit heeft als voordeel dat ze niet groter worden bij grote signalen, maar ook dat ze relatief groot kunnen zijn ten opzichte van kleine signalen.

Het grote voordeel is dat de Fixed Point ALU logica relatief eenvoudig is ten opzichte van de Floating Point. Hierdoor zullen er minder resources van de FPGA gebruikt worden ten opzichte van floating point.

Afwegingen

Over het gebruik van Floating Point of Fixed point signaal processing is een whitepaper geschreven door Xilinx [28]. Hierin is een voorbeeld gegeven voor een implementatie van 10 FIR-filters. De eerste variant met Floating point en de tweede variant fixed point. Hierbij was de resource utilisatie van de Fixed Point oplossing vijf keer kleiner. Verder bied de Fixed point oplossing de volgende voordelen:

- Kan met een hogere kloksnelheid werken
- Signaal latency gereduceerd
- Minder energie benodigd

Daarnaast wordt in de whitepaper aangetoond dat het verschil in filterkwaliteit zeer minimaal is. Hieruit volgt dat wanneer een geoptimaliseerde oplossing nodig is, Fixed Point processing een goede optie is.

6.2.2. Bit-grootte en Sample frequentie

Een veel voorkomende discussie rondom audio, is de audiokwaliteit. De beleving van audiokwaliteit is echter zeer subjectief. De benodigde bit-grootte en samplefrequentie zijn dan ook regelmatig onderwerp van discussie. Daarnaast hebben deze twee factoren verschillende invloeden afhankelijk van de plaats in het proces van opnemen tot afspelen.

Afspeel bit-grootte

Onderzoeken zijn gedaan om te kijken of mensen onderscheid kunnen maken tussen het afspelen van CD-kwaliteit audio (16-bits/44,1 kHz) en hogere kwaliteit audio. Eén van deze onderzoeken is uitgevoerd door de Boston Audio Society [29]. Hieruit volgde dat mensen bij het afspelen van de audio geen onderscheid kunnen maken tussen 16 bits/44,1 kHz en hogere kwaliteit audio. Voor het afspelen binnen dit systeem zou deze kwaliteit dus ook voldoende zijn.

ADC

Hoewel 16 bits/44,1 kHz voldoende is voor het afspelen van audio, zijn er argumenten om een ADC met hogere resolutie te gebruiken.

Vaak wordt gebruik gemaakt van een 24 bits ADC. Door het grotere aantal bits is het dynamisch bereik van de ADC groter. Hierdoor komt het instellen van de gain van het inkomende signaal minder nauwkeurig. Daarnaast zorgt het grotere aantal bits er ook voor dat de kwantisatie ruis van de ADC lager ligt.

Daarnaast werkt steeds meer audio apparatuur tegenwoordig met een samplerate van 48 kHz. Om het transformeren tussen verschillende kloksnelheden te vermijden, is het verstandig om 48 kHz te gebruiken.

Reken grootte

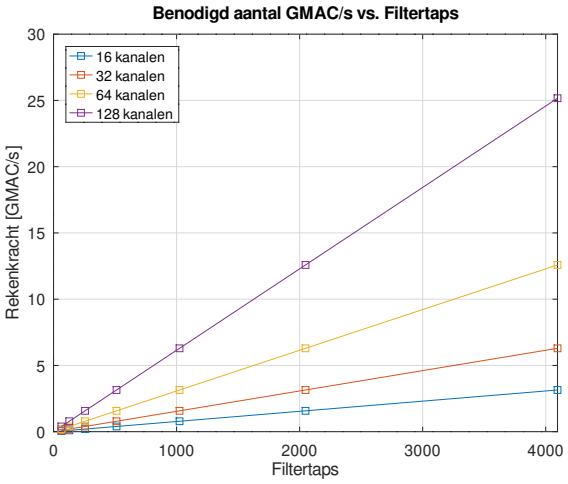
Voor het toepassen van signaalprocessing is een grotere bit-diepte nodig. Bij het uitvoeren van de vele vermenigvuldigingen en optellingen, kan een getal ontstaan dat vele malen groter is dan bijvoorbeeld 24 bits.

Dit bit-diepte voor de tussenresultaten bij audiobewerkingen valt hierbij vaak tussen de 48 en 56 bits. Deze is echter ook afhankelijk van de bit-grootte van de audio-samples

6.2.3. Filter lengtes

De hoeveelheid rekenkracht die nodig is, is afhankelijk van de lengte van de gebruikte filters. In figuur 6.6 is aantal filtertaps uitgezet tegenover het benodigd aantal GMAC/s (Giga Multiply Accumulate per Second). Dit is gedaan voor verschillende aantallen kanalen.

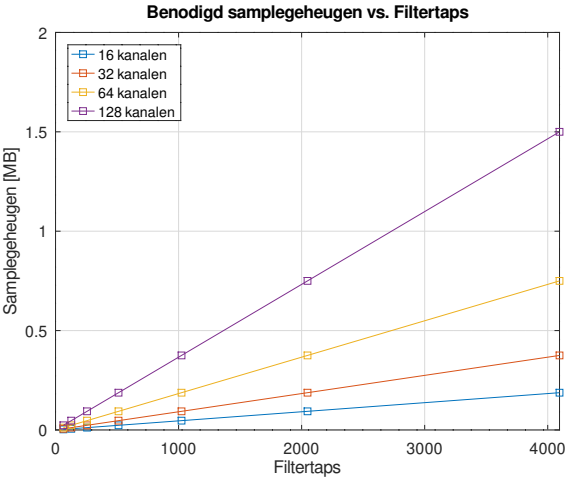
Deze benodigde rekenkracht is haalbaar met huidige FPGA technologie. Een voorbeeld hiervan is de Xilinx Spartan-7 XC7S15 [30]. Dit is een FPGA uit het midden van de 7 serie (~€20). Deze FPGA kan 18 GMAC/s[1] behalen. Rekenkracht zal hierin dus niet snel voor een limitatie zorgen.



Figuur 6.6: Rekenkracht versus filtergrootte en kanalen ($Fs = 48\,kHz$)

6.2.4. Geheugen

De hoeveelheid opslagruimte voor de audiosamples is ook afhankelijk van de filterlengtes en het aantal kanalen. Deze gegevens zijn tegen elkaar uitgezet in figuur 6.7. Hierbij is uitgegaan van een bit-grootte van 24 bits en een samplefrequentie van 48 kHz. Uit deze gegeven blijkt dat zelf bij een groot aantal kanalen (128) het om relatief kleine hoeveelheid geheugen gaat (1,5 MB).

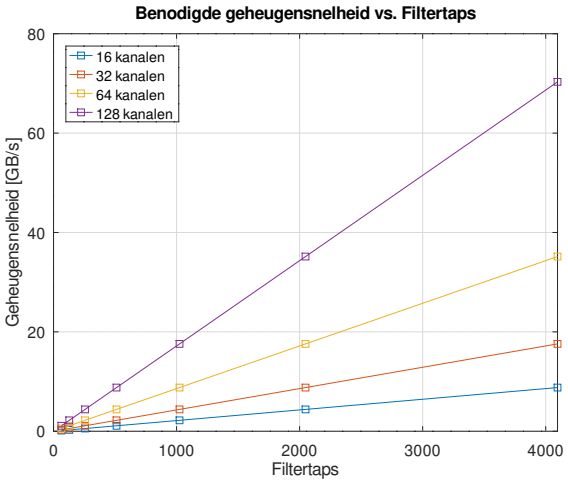


Figuur 6.7: Grootte samplegeheugen versus filtergrootes en kanalen (24 bit sampldiepte)

Ook is gekeken naar de snelheid van dit geheugen. Hierbij is uitgegaan van het 'worst-case' scenario waarin alle samples vanuit het geheugen opgehaald moeten worden. Dit is weergegeven in figuur 6.8. De berekening hiervoor is weergegeven in vergelijking 6.1

$$memspeed = \frac{taps \cdot channels \cdot Fs \cdot 3}{1024^3} \qquad (6.1)$$

Deze snelheden worden al snel groot, met een vereiste van 35 GB/s voor 64 kanalen met filters van 4096 taps. Dit is een te hoge snelheid om realistisch te implementeren. Hier zal bij de implementatie rekening mee gehouden moeten worden. De systeemarchitectuur moet zo worden opgezet dat de geheugensnelheden haalbaar blijven.



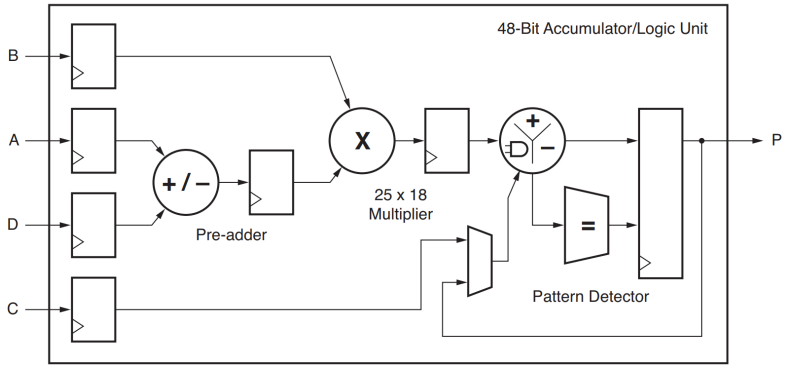
Figuur 6.8: Geheugensnelheid versus filtergroottes en kanalen (24 bit sampldiepte)

[1] Uitgaande van symmetrische FIR-filters en een snelheid van 450 MHz voor de DSP blokken.

6.2.5. DSP-Slices

Verschillende FPGA's beschikken over standaard DSP-Slices. Dit zijn standaard schakelingen binnen de FPGA die zijn geoptimaliseerd voor DSP operaties. Hierbij gaat het met name om de Multiply Accumulate (MAC) operatie.

Een voorbeeld hiervan is de DSP48E1 slice van Xilinx [31]. De opbouw van deze slice is weergegeven in figuur 6.9.



Figuur 6.9: Architectuur DSP48E1 Slice [31]

Deze slice beschikt over een 25x18 bits multiplier met een 48 bits accumulator. In combinatie met het gebruik van de geïntegreerde preadder, kunnen hier op een efficiente manier FIR-filters mee geïmplementeerd worden.

Het gebruik van deze slices reduceert ook het gebruik van de overige FPGA resources. Een voorbeeld van Xilinx is de implementatie van 10 FIR-filters [28]. Voor deze implementatie zorgde het gebruik van de DSP48 slices tot een drie keer lager resource gebruik.

6.2.6. Filtering advies

Vanuit de gegeven specificaties en bevindingen met betrekking tot de verschillende aspecten, volgen de volgende specificaties:

- Samplefrequentie: 48 kHz
- Data type: Fixed Point
- Filter grootte: 1024 taps (Als basis. Mogelijk afhankelijk van benodigde correctie)
- ADC Bit-diepte: 24 bits (Onderzoeken of 16 bits ook voldoende kwaliteit geeft)
- Interne rekengrootte: 16 bits
- Accumulator grootte: minimaal 32 bits (Grootte gebruiken van beschikbare hardware)

Verder is het aan te raden om gebruik te maken van een FPGA met geïntegreerde DSP-slices, zodat de filters efficienter geïmplementeerd kunnen worden.

6.3. Processor

Voor het maken van de filterberekeningen en het aansturen van het gehele proces, is een processor nodig. Aangezien het filteren op een FPGA geïmplementeerd wordt, zijn hier drie verschillende mogelijkheden:

- Hardcore: Externe processor verbinden met de FPGA
- Softcore: Processor configureren in de FPGA
- Integrated Hardcore: FPGA met ingebouwde hardcores

6.3.1. Hardcore

Bij het gebruik van een Hardcore processor zijn de processor en FPGA twee gescheiden elementen. Dit bied een aantal voor- en nadelen.

Voordelen:

- Flexibiliteit in processor keuze
- Kunnen high-performance zijn

Nadelen:

- Vereist het integreren van extra componenten
- Goede integratie tussen FPGA en processor vereist

6.3.2. Softcore

Bij een Softcore processor wordt de processor geïmplementeerd door middel van FPGA logica. Er zijn verschillende Softcore processors beschikbaar voor verschillende doeleinden.

Bij het kiezen van een processor is het van belang dat de Softcore over goede floating-point performance beschikt. Dit is nodig voor het uitvoeren van de filterberekeningen. Mogelijk kan dit veel FPGA resources kosten. Verder zitten aan deze oplossing de volgende voor en nadelen.

Voordelen:

- Processor te configureren aan de hand van behoeften
- Flexibiliteit in FPGA keuze

Nadelen:

- Lagere snelheid (30% tot 50% van de snelheid van een Hardcore [32])
- Verbruikt FPGA resources

6.3.3. Integrated Hardcore

Een steeds vaker voorkomende optie is een FPGA met daarin een Hardcore processor. Hierbij is het mogelijk om directe connecties te maken tussen de processor en de FPGA. Ook aan deze oplossing zijn een aantal voor en nadelen verbonden.

Voordelen:

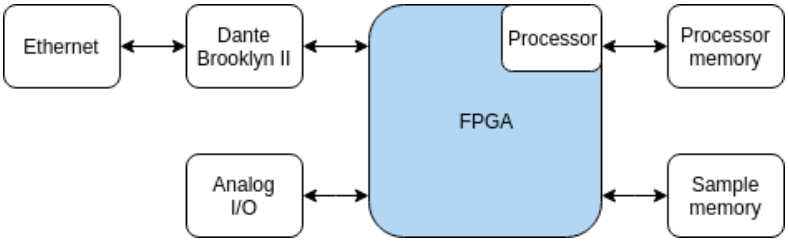
- FPGA en processor zijn direct aan elkaar te koppelen
- Minder hardware onderdelen nodig ten opzichte van losse processor en FPGA

Nadelen:

- Minder flexibiliteit in FPGA keuze
- Minder flexibiliteit in processor keuze

6.4. Voorstel technische specificatie audiosignaalprocessor

Vanuit de beschreven optimalisatie-concepten, verdere eisen en de hiervoor beschreven adviezen, volgt een concept-specificatie voor de hardware. Deze concept-specificatie is weergegeven in figuur 6.10.



Figuur 6.10: Globale concept specificatie audiosignaalprocessor

Er is hierbij gebruik gemaakt van een FPGA met geïntegreerde processor. Hierbij is een scheiding aangebracht tussen het werkgeheugen van de processor en het geheugen om de audio-samples op te slaan.

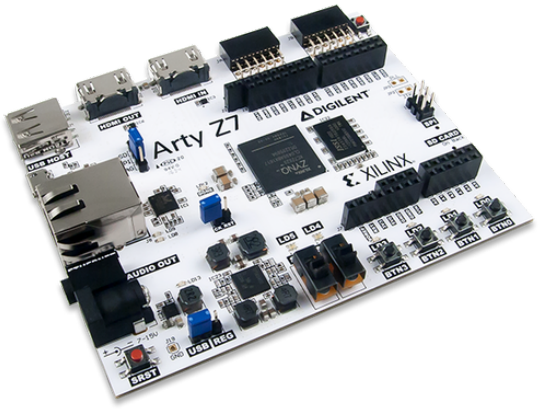
De processor zorgt hierbij voor de berekeningen voor een correctiefilter. Dit doet hij aan de hand van de audio-data die vanuit de FPGA wordt doorgegeven. Door de directe integratie van de FPGA en processor, kan dit met hoge snelheid gebeuren. Wanneer meer flexibiliteit nodig is met betrekking tot de processor keuze, kan beter een externe processor gebruikt worden.

6.4.1. Prototyping platform

Als mogelijke eerste stap voor verdere ontwikkeling kan gebruik gemaakt worden van de Digilent Arty Z7-20 [33]. Deze is weergegeven in figuur 6.11.

Dit is een development-bord die beschikt over een Xilinx ZYNC FPGA met een geïntegreerde dual core ARM processor. Daarnaast is een Artix-7 FPGA aanwezig met 220 DSP slices.

Dit platform kan gebruikt worden om te evalueren hoeveel FPGA resources en Processor power nodig zijn om de volledige applicatie te implementeren.



Figuur 6.11: Arty Z7-20 development board [33]

7. Conclusie en Aanbevelingen

Aan de hand van het uitgevoerde onderzoek volgt in dit hoofdstuk een conclusie. Daarnaast worden aanbevelingen gegeven voor eventuele voortzetting van dit project.

7.1. Conclusie

De volgende conclusies kunnen getrokken worden aan de hand van de optimalisatie research en uitwerking:
- Verschillende methodes zijn onderzocht en gesimuleerd voor het optimaliseren van de EQ.
- Een EQ-algoritme is gemaakt die de afwijking van een luidspreker met een factor 359 kan reduceren.
- De correctiefilters hebben minder autoriteit in de lage frequenties.
- Een Delay-algoritme is ontwikkeld om de onderlinge vertragingen tussen luidsprekers tot op enkele samples te corrigeren.
- Het EQ- en Delay-algoritme zijn gecombineerd om beide correcties gelijktijdig uit te voeren.
- De EQ en Delay kunnen gelijktijdig op meerdere speakers geoptimaliseerd worden.
- Een Python applicatie is ontwikkeld om de optimalisaties automatisch uit te voeren.

De volgende conclusies kunnen getrokken worden aan de hand van de uitgevoerde Hardware research:
- Voor het digitaal uitbreiden van de audio I/O is een Dante Brooklyn II geadviseerd, met het oog op een korte ontwikkeltijd.
- Voor de filtering is geadviseerd om met 16-24 bits fixed-point getallen te rekenen.
- Bij het kiezen van een FPGA is het aan te raden er een te zoeken die beschikt over standaard DSP elementen (MAC).
- Een FPGA met geïntegreerde Hard-Core is geadviseerd
- Wanneer een Hard-Core niet voldoende flexibiliteit bied, kan een externe processor overwogen worden

7.2. Aanbevelingen

De volgende aanbevelingen worden gegeven voor voortzetting van het project:
- FIR-filters hebben moeite met het corrigeren van lage frequenties. IIR filters kunnen gebruikt worden om de lage frequenties te corrigeren.
- Mogelijk adaptieve filter-lengtes om resources zo efficiënt mogelijk te benutten.
- Effectiviteit van de optimalisatie testen bij meer dan twee luidsprekers
- EQ optimalisatie verder verbeteren zodat deze ook bij slechtere signaal-ruis verhoudingen werkt.
- Frequentiebanden willekeurig verdelen in de testsignalen om tonaliteit te voorkomen.
- Overlap tussen signalen voorkomen bij gelijktijdige EQ en delay door langere fragmenten te gebruiken.

7.3. Eindconclusie

De basis voor het automatisch optimaliseren van EQ en Delay van luidsprekers is gelegd. Hierbij is het mogelijk om meerdere luidsprekers gelijktijdig te optimaliseren. Hiermee zijn de tools beschikbaar voor het optimaliseren van de audio op de positie van de microfoon. Vanuit deze basis kan de applicatie verder uitgewerkt worden en op een FPGA geïmplementeerd.

De volgende stap is om de voorgestelde hardware te gaan ontwikkelen, de optimalisatie algoritmes te finetunen en dit concept verder te testen en uit te werken.

Dit geheel maakt het project tot een goede basis voor het verder ontwikkelen van de automatische EQ en Delay optimalisatie.

Referenties

[1] M. T. van Essen, „Plan van Aanpak Afstudeerstage Embedded Acoustics", 2019-02-07_PVA_Afstuderen_Embedded-Acoustics_vA02.pdf, feb 2019.

[2] J. W. Eaton. (2019). GNU Octave, adres: https://www.gnu.org/software/octave/ (bezocht op 20-02-2019).

[3] Audacity. (2019). Audacity | Free, open source, cross-platform audio software for multi-track recording and editing., adres: https://www.audacityteam.org/ (bezocht op 20-02-2019).

[4] D. Rowell. (2008). lecture_25.pdf, adres: https://ocw.mit.edu/courses/mechanical-engineering/2-161-signal-processing-continuous-and-discrete-fall-2008/lecture-notes/lecture_25.pdf (bezocht op 22-02-2019).

[5] N. Westerlund. (2000). rapport.dvi, adres: https://www.diva-portal.org/smash/get/diva2:829580/FULLTEXT01.pdf (bezocht op 25-02-2019).

[6] O. H. Bjor. (2000). Maximum Length Sequence, adres: http://www.altracustica.org/docs/mls_theory.pdf (bezocht op 27-02-2019).

[7] M. Thomas. (2006). A Novel Loudspeaker Equaliser, adres: http://www.commsp.ee.ic.ac.uk/~mrt102/publications/Thomas2006.pdf (bezocht op 27-02-2019).

[8] A. Vaneev. (2018). Free Reverb Impulse Responses | Voxengo, adres: https://www.voxengo.com/impulses/ (bezocht op 16-04-2019).

[9] ITU. (1998). RELATIVE TIMING OF SOUND AND VISION FOR BROADCASTING, adres: https://www.itu.int/dms_pubrec/itu-r/rec/bt/R-REC-BT.1359-1-199811-I!!PDF-E.pdf (bezocht op 10-05-2019).

[10] EBU. (2007). The relative timing of the sound and vision components of a television signal, adres: https://tech.ebu.ch/docs/r/r037.pdf (bezocht op 10-05-2019).

[11] IEC, *Electroacoustics - Sound level meters*, 2013.

[12] C. Sanderson en R. Curtin. (2019). Armadillo: C++ library for linear algebra & scientific computing, adres: http://arma.sourceforge.net/ (bezocht op 10-05-2019).

[13] NumPy developers. (2019). NumPy - NumPy, adres: https://www.numpy.org/ (bezocht op 10-05-2019).

[14] SciPy developers. (2019). SciPy.org - SciPy.org, adres: https://www.scipy.org/ (bezocht op 10-05-2019).

[15] Riverbank Computing Limited. (2019). Riverbank | Software | PyQt | What is PyQt?, adres: https://www.riverbankcomputing.com/software/pyqt/intro (bezocht op 14-05-2019).

[16] Matplotlib development team. (2019). Matplotlib: Python plotting — Matplotlib 3.0.3 documentation, adres: https://matplotlib.org/ (bezocht op 14-05-2019).

[17] M. Geier. (2019). Play and Record Sound with Python — python-sounddevice, version 0.3.13, adres: https://python-sounddevice.readthedocs.io/en/0.3.13/ (bezocht op 15-05-2019).

[18] Audinate. (2019). Dante Overview | Audinate, adres: https://www.audinate.com/solutions/dante-overview (bezocht op 11-02-2019).

[19] Cobranet. (2019). CobraNet HOME | CobraNet HOME, adres: https://www.cobranet.info/ (bezocht op 11-02-2019).

[20] Philips Semiconductors. (1996). I2SBUS, adres: https://web.archive.org/web/20080706121949/http://www.nxp.com/acrobat_download/various/I2SBUS.pdf (bezocht op 12-02-2019).

[21] Audinate. (2018). Dante Adapter Modules | Audinate, adres: https://www.audinate.com/products/manufacturer-products/dante-adapter-modules (bezocht op 12-02-2019).

[22] ——, (2016). Dante_Brooklyn_II_6-july-2016, adres: https://www.audinate.com/sites/default/files/datasheets/Dante_Brooklyn_II_6-july-2016.pdf (bezocht op 12-02-2019).

[23] ——, (2016). Dante HC | Audinate, adres: https://www.audinate.com/products/manufacturer-products/dante-hc (bezocht op 12-02-2019).

[24] ——, (2018). Dante Ultimo | Audinate, adres: https://www.audinate.com/products/manufacturer-products/dante-ultimo (bezocht op 12-02-2019).

[25] ——, (2018). Dante IP Core | Audinate, adres: https://www.audinate.com/products/manufacturer-products/dante-ip-core (bezocht op 12-02-2019).

[26] IEEE. (2008). IEEE 754-2008 - IEEE Standard for Floating-Point Arithmetic, adres: https://standards.ieee.org/content/ieee-standards/en/standard/754-2008.html (bezocht op 15-03-2019).

[27] G. Duckett en T. Pennington. (2005). Fixed-Point vs. Floating-Point DSP for Superior Audio, adres: https://recording.de/uploads/newbb/4bfcff8af46c790faee3be9da5342914.pdf (bezocht op 15-03-2019).

[28] A. Finnerty en H. Ratigner. (2017). Reduce Power and Cost by Converting from Floating Point to Fixed Point, adres: https://www.xilinx.com/support/documentation/white_papers/wp491-floating-to-fixed-point.pdf (bezocht op 22-05-2019).

[29] B. Meyer en D. Moran. (2017). Audibility of a CD-Standard A/D/A Loop Inserted into High-Resolution Audio Playback, adres: http://drewdaniels.com/audible.pdf (bezocht op 22-05-2019).

[30] Xilinx. (2018). 7 Series FPGAs Data Sheet: Overview (DS180), adres: https://www.xilinx.com/support/documentation/data_sheets/ds180_7Series_Overview.pdf (bezocht op 18-03-2019).

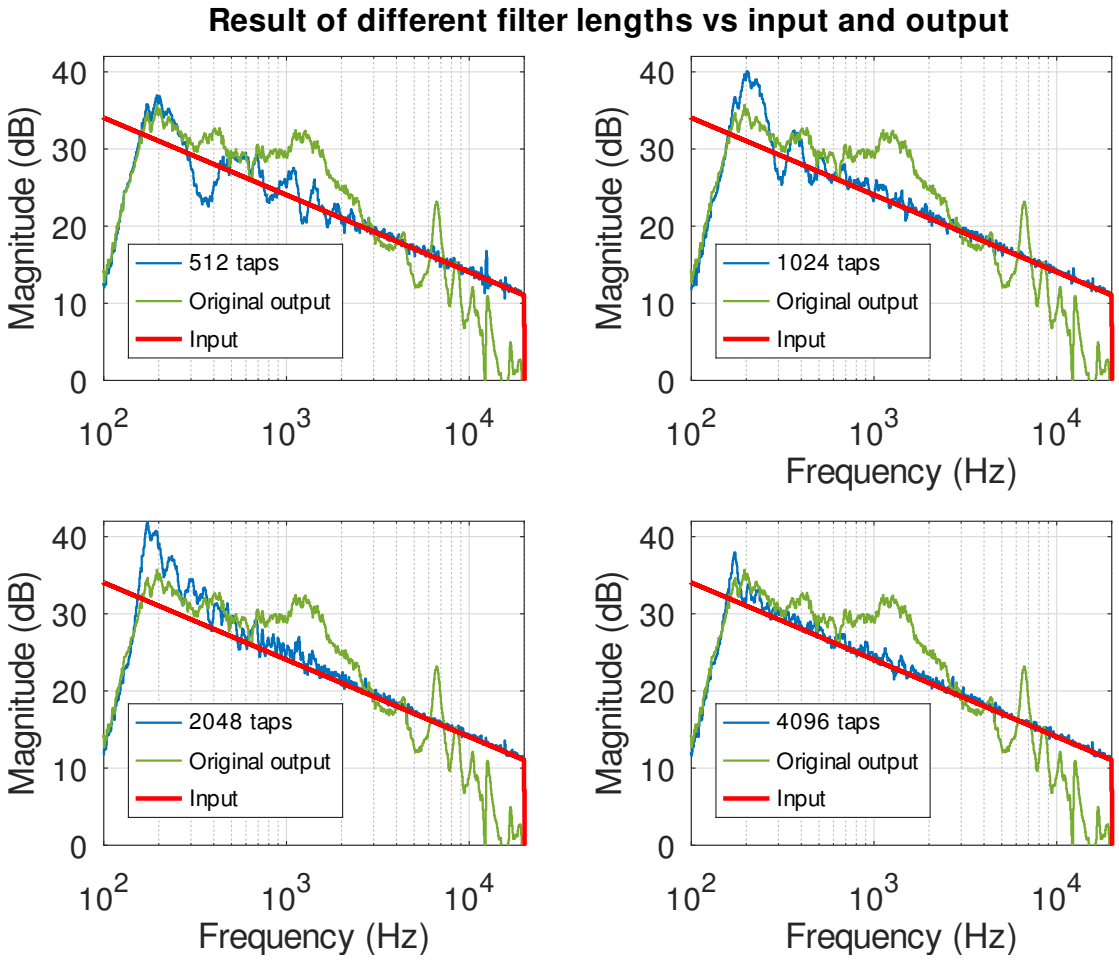
[31] ——, (2018). 7 Series DSP48E1 Slice User Guide, adres: https://www.xilinx.com/support/documentation/user_guides/ug479_7Series_DSP48E1.pdf (bezocht op 22-05-2019).

[32] V. J. V. Kanhiroth. (2017). Embedded processors on FPGA: Hard-core vs Soft-core, adres: https://scholarworks.gvsu.edu/cgi/viewcontent.cgi?article=1843&context=theses (bezocht op 23-05-2019).

[33] Digilent. (2019). Arty Z7: APSoC Zynq-7000 Development Board for Makers and Hobbyists - Digilent, adres: https://store.digilentinc.com/arty-z7-apsoc-zynq-7000-development-board-for-makers-and-hobbyists/ (bezocht op 23-05-2019).

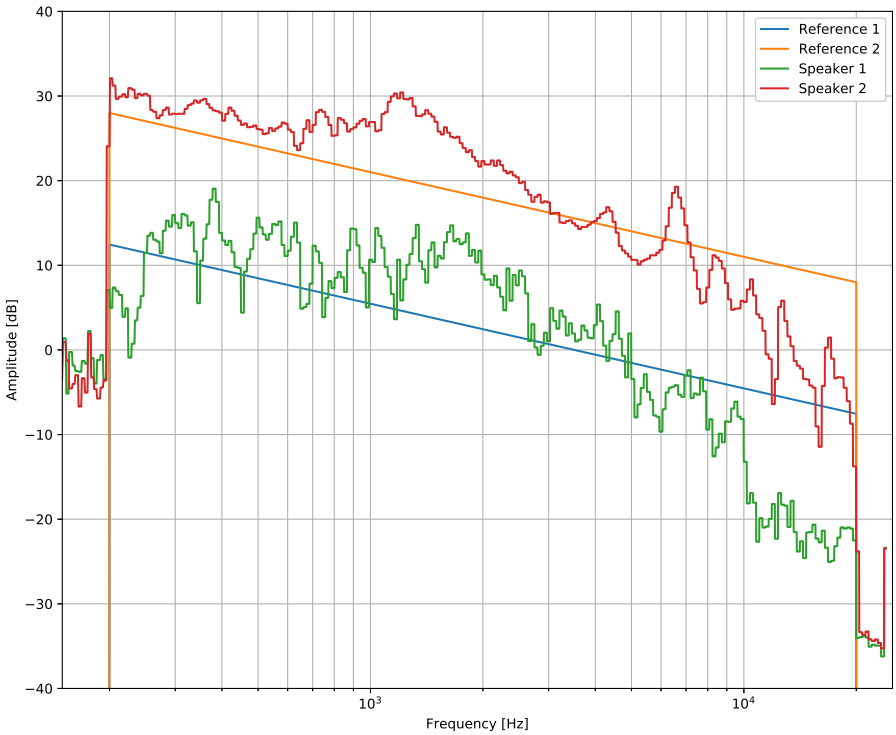
Bijlagen

Bijlage A: Invloed van het aantal filtertaps op de filterkwaliteit

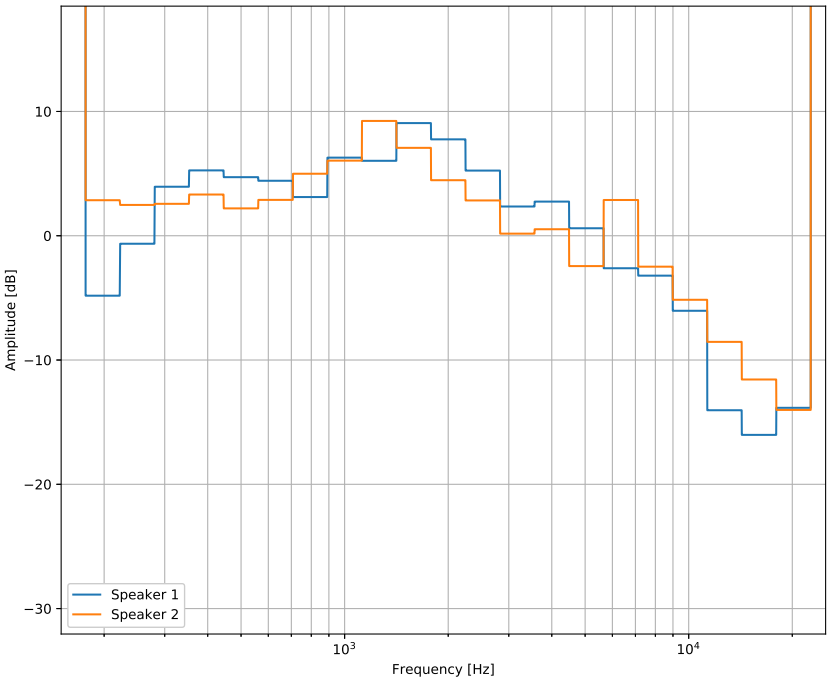


Bijlage B: Verificatie Tests

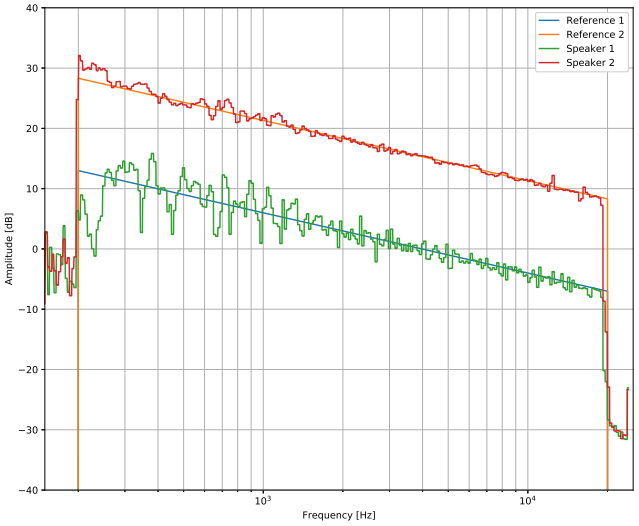
FFT zonder optimalisatie
MSE: ch1: 8,80; ch2: 5,02



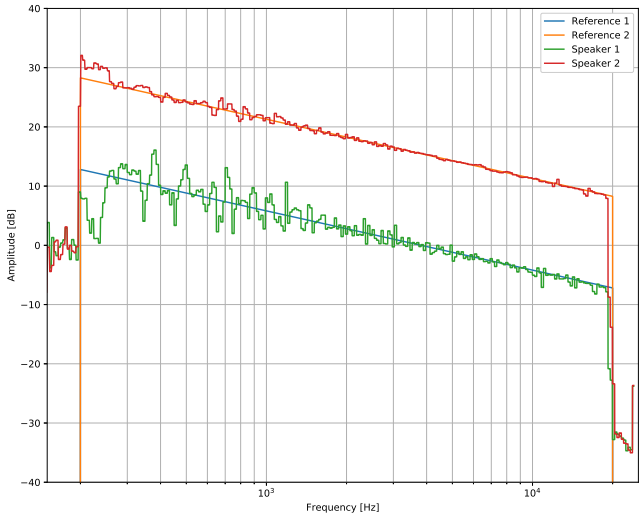
$\frac{1}{3}$ octaafband afwijking zonder correcties



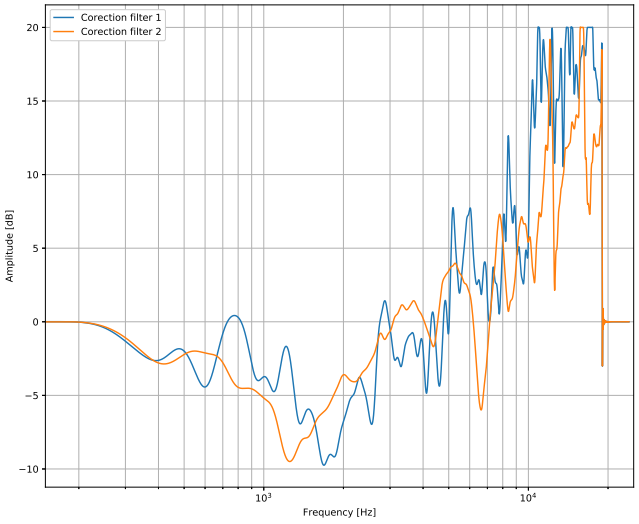
FFT 1024 taps optimalisatie
MSE: ch1: 0,55; ch2: 0,056
Verbetering: ch1: 15,9; ch2: 90,0



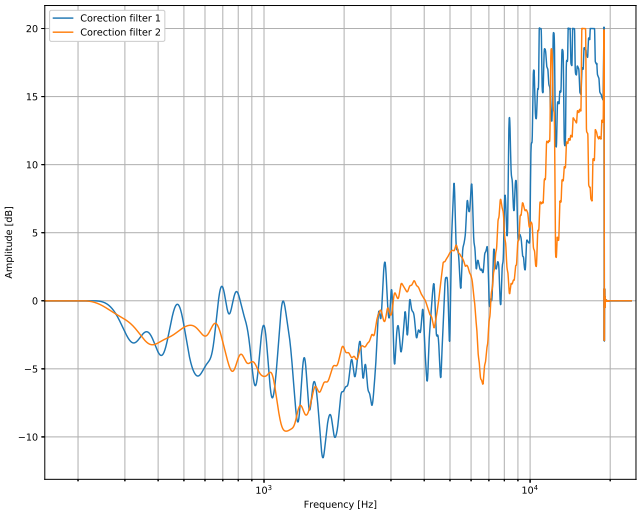
FFT 2048 taps optimalisatie
MSE: ch1: 0,54; ch2: 0,035
Verbetering: ch1: 16,3; ch2: 143,0



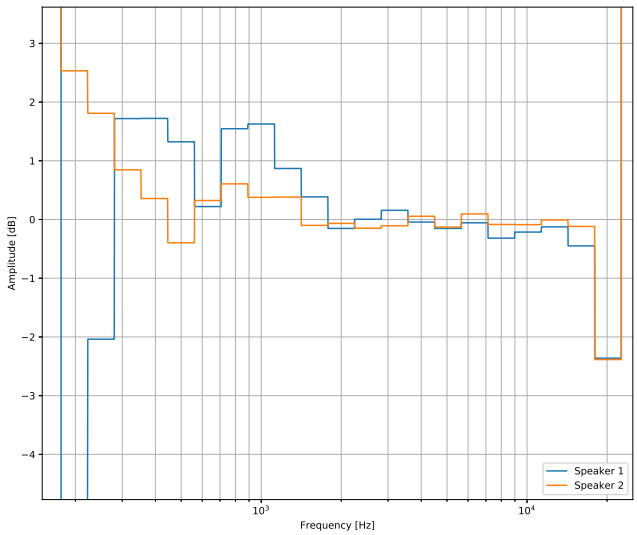
1024 taps FIR Correctie filter



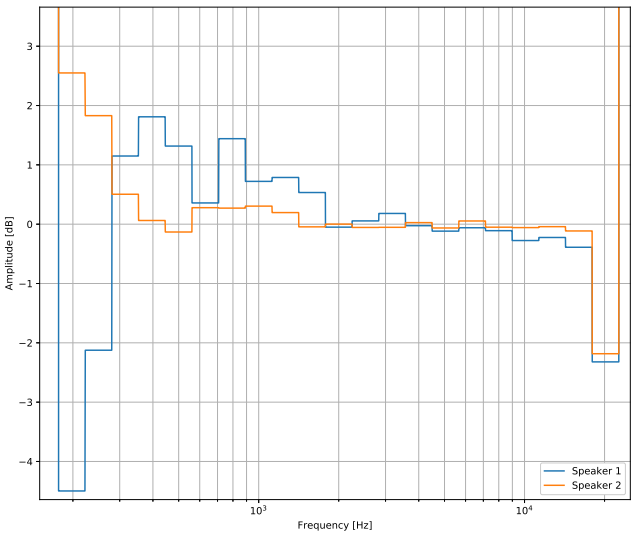
2048 taps FIR Correctie filter



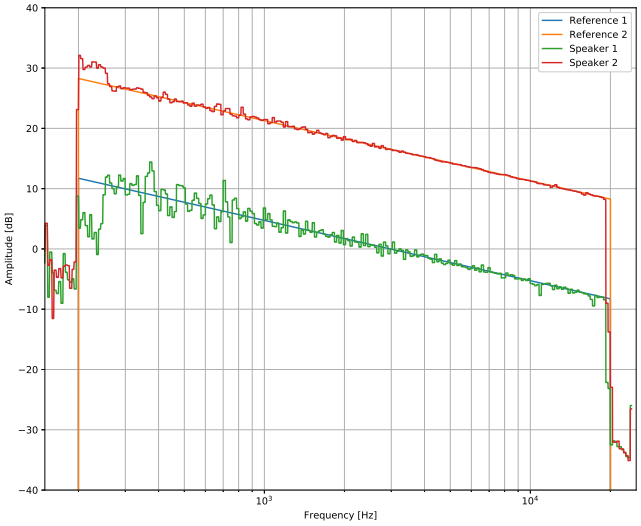
$\frac{1}{3}$ octaafband afwijking 1024 taps



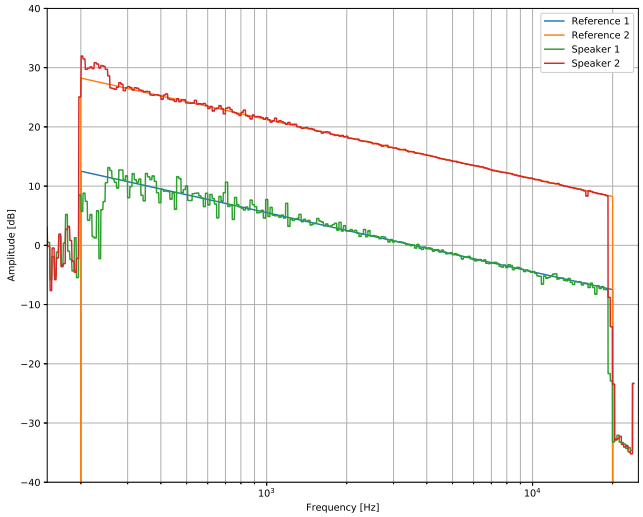
$\frac{1}{3}$ octaafband afwijking 2048 taps



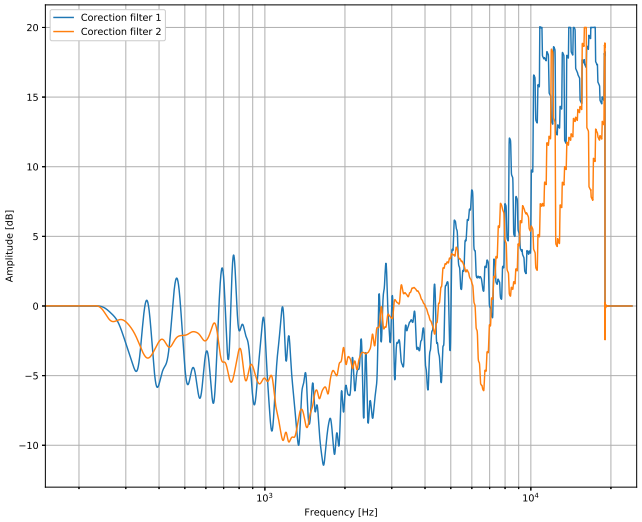
FFT 4096 taps optimalisatie
MSE: ch1: 0,22; ch2: 0,024
Verbetering: ch1: 39,8; ch2: 206,6



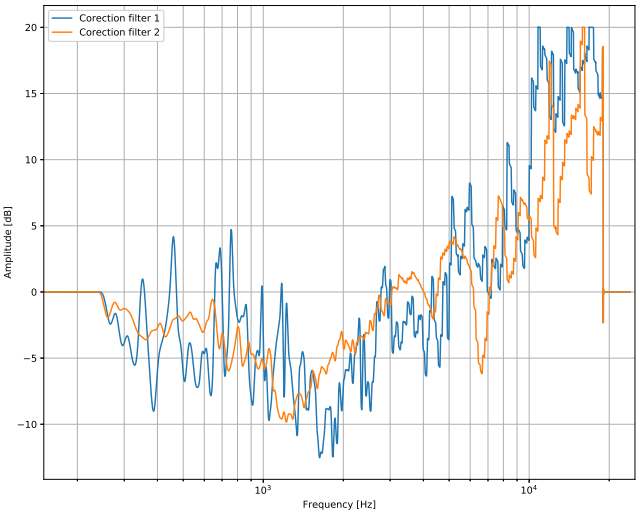
FFT 8192 taps optimalisatie
MSE: ch1: 0,095; ch2: 0,014
Verbetering: ch1: 92,2; ch2: 358,6



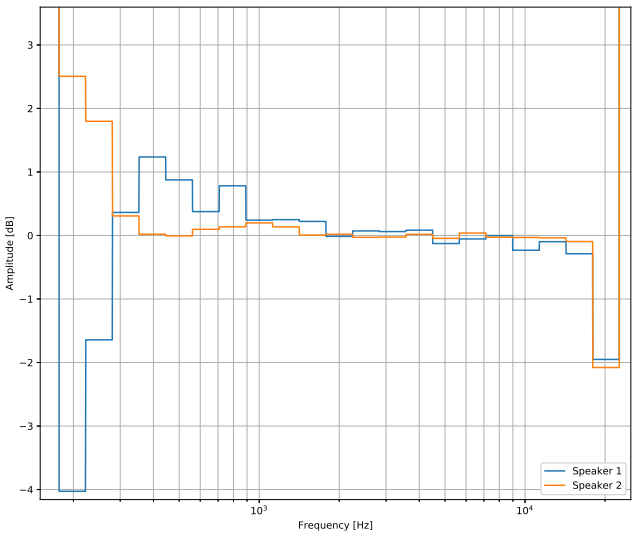
4096 taps FIR Correctie filter



8192 taps FIR Correctie filter



$\frac{1}{3}$ octaafband afwijking 4096 taps



$\frac{1}{3}$ octaafband afwijking 8192 taps