# Journal 2

## Indlejret Signal Behandling

### Gruppe 1


| Navn | Studienummer | Retning |
|---|---|---|
| **Mathias Ørnstrup Hvidberg** | 201905706 | E |
| **Niels Højrup Pedersen** | 201604812 | E |
| **Jakob Saugbjerg Lange** | 201907544 | E |

# Indhold

# Introduktion

I denne opgave implementeres der en digital equalizer, som baserer sig på seriekoblede IIR-filtrer af typen biquad. Disse filtrer implementeres først i Matlab, hvorefter de kodes på Blackfin-platformen BF533. Herefter vil resultatet af de to implementeringer blive sammenlignet, for at undersøge eventuelle afvigelser imellem dem. Slutteligt vil den implementerede equalizer på Blackfin blive demonstreret, med tilhørende billeder fra oscilloskop og frekvensgenerator.

## Implementering af makePeakEQ og BiquadFilter i Matlab

I Listing 1 ses Matlab-implementeringen for `makePeakEQ`. Denne funktion tager samplings-frekvensen, cut-frekvensen, Q og gain som parametre. Returværdien for denne funktion er a og b koefficienterne. For at lave dette filter, vil der kun blive brugt tre a-koefficienter og tre b-koefficienter.

```
function [b, a] = makePeakEQ(fs, fc, Q, A)
% Computes coefficients for peaking Equalizer
% b = [b0 b1 b2] Nominator
% a = [a0 a1 a2] Denominator
% fs = sampling frequency
% fc = Center EQ frequence
% Q = bandwidth of EQ
% A = gain
% Based on the Audio-EQ-Cookbook:
% http://www.musicdsp.org/files/Audio-EQ-Cookbook.txt

    w0 = 2*pi*fc/fs;
    alpha = sin(w0)/(2*Q);

    b = [0 0 0];
    a = [0 0 0];

    b(1) = 1+alpha*A;
    b(2) = -2*cos(w0);
    b(3) = 1-alpha*A;

    a(1) = 1+alpha/A;
    a(2) = -2*cos(w0);
    a(3) = 1-alpha/A;
end
```

*Listing 1 - makePeakEQ*

I Listing 2 ses `biquadFilter`-funktionen. Denne funktion tager et inputsignal, og de 6 udregnede a og b koefficienter, fra `makePeakEQ`-funktionen og returnerer et filtreret signal.

```
function [ y ] = biquadFilter( b, a, x )
%Performs Biquad filtering
% IIR 2. order, Direct Form 1
% x - sample input vector
% Biquad coefficients
% b = [b0 b1 b2] Nominator
% a = [a0 a1 a2] Denominator
    N = length(x);

    y = zeros(1,N)';

    x1 = 0;
    x2 = 0;

    y1 = 0;
    y2 = 0;

    for n=1:N
        x0 = x(n);
```

```
        y(n) = (b(1)*x0 + b(2)*x1 + b(3)*x2 - a(2)*y1 - a(3)*y2) / a(1);

        x2 = x1;
        x1 = x0;

        y2 = y1;
        y1 = y(n);
    end

end
```

*Listing 2 - biquadFilter Matlab*

For at teste det ideelle matlab-filter, anvendes der et testsignal. I dette tilfælde anvendes to sinus-signaler med to forskellige amplituder og to forskellige frekvenser. Testsignalet kan opskrives på følgende måde:

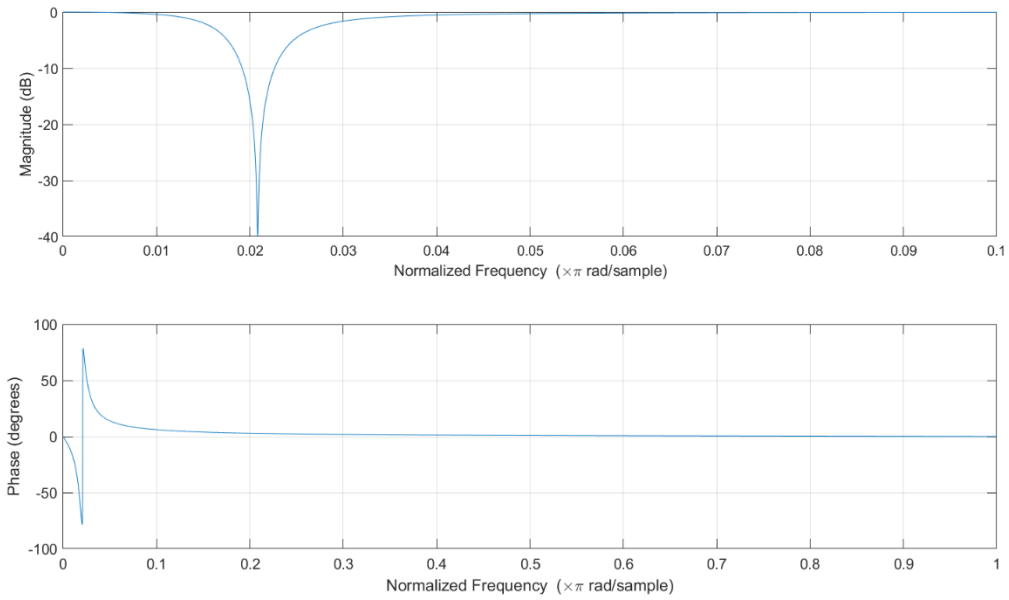
$$x(t) = 4 \cdot \sin(2\pi \cdot 500t) + 1 \cdot \sin(2\pi \cdot 4000t)$$

I den første test ønsker vi at fjerne den langsomme 500 Hz sinustone. Til at gøre dette, vil følgende parametre bliver brugt:

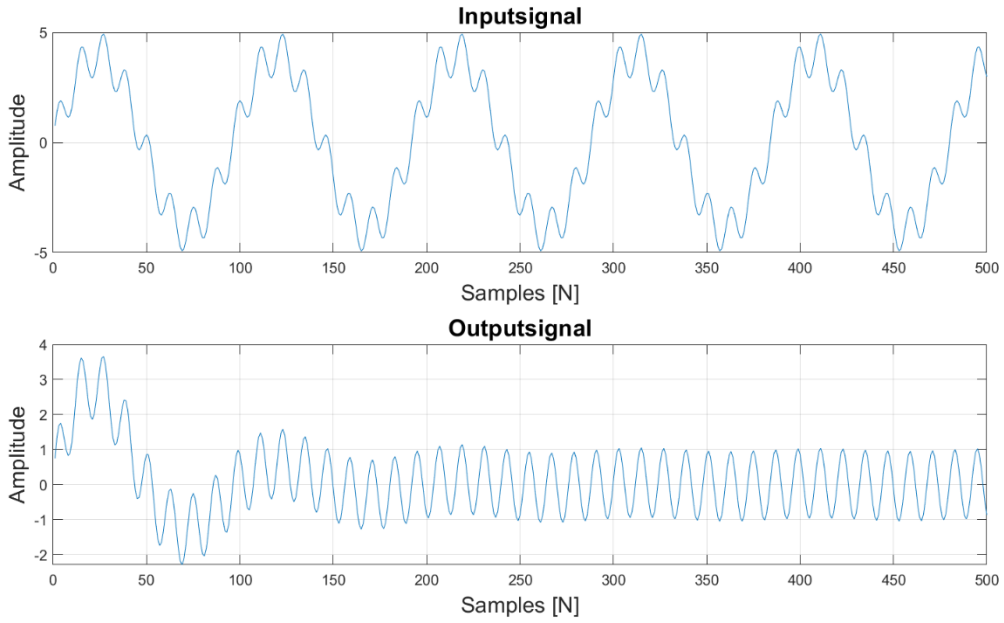
- $fs = 48000$
- $fc = 500$
- $A = 0.1$
- $Q = 20$

Ved at have en A(gain) på 0.1 vil 500 Hz blive dæmpet. Hvis gain > 1 vil signalet derimod bliver forstærket.

Frekvensresponsen for dette filter kan ses på Figur 1. Resultatet af signalet i tidsdomænet kan ses på Figur 2.



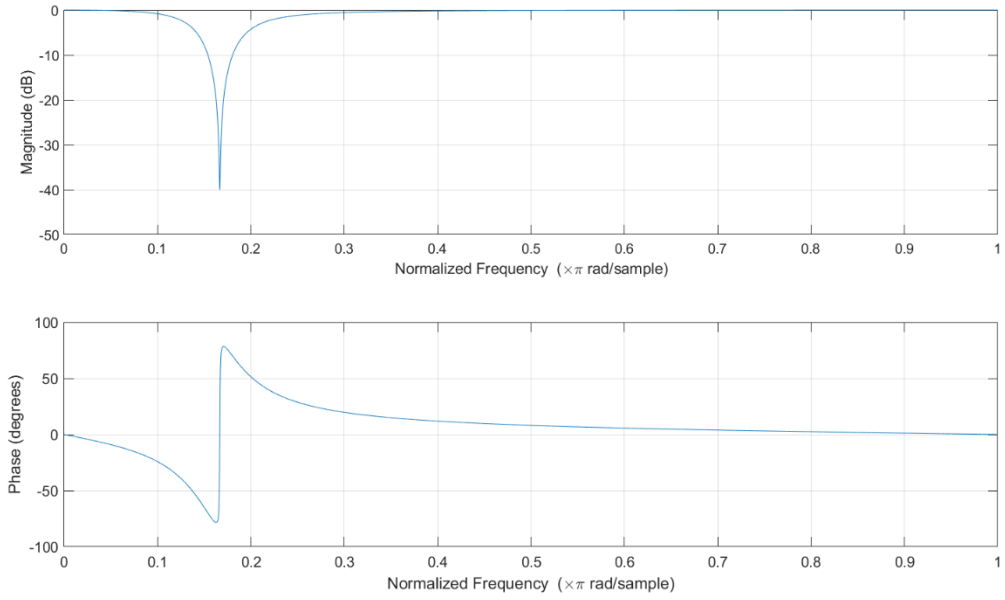

*Figur 1 - Frekvensrespons for filter, som fjerner 500 Hz.*



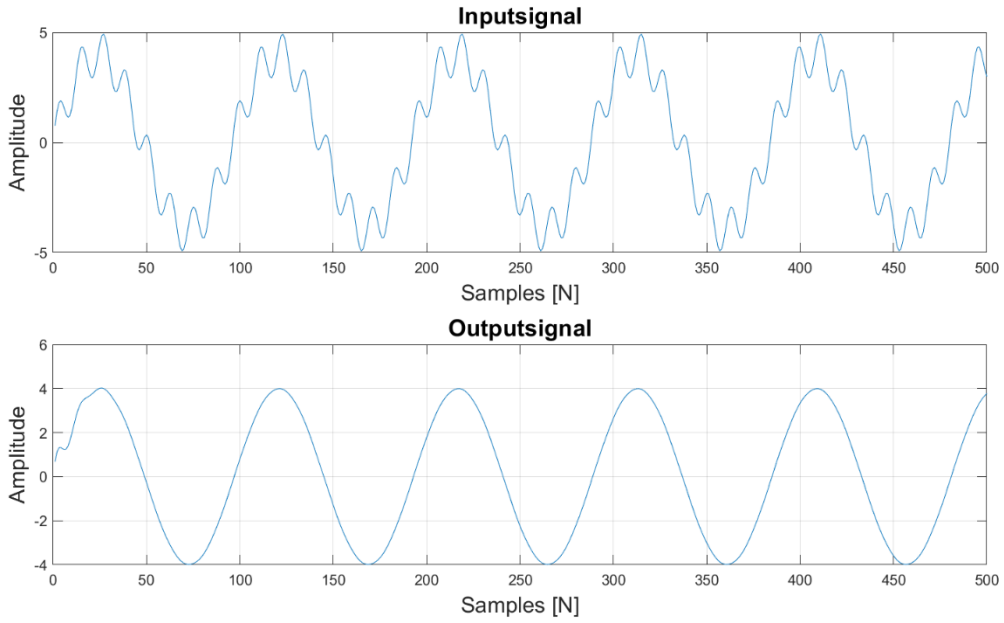

*Figur 2 - Resultat af filter i tidsdomænet, hvor de 500 Hz er fjernet.*

På Figur 2 ses det at de 500 Hz er fjernet fra signalet. Dog kan man se, at filteret først rigtig får fjernet de 500 Hz efter sample ~nr. 200. Dette giver dog god mening, da filteret der bliver brugt, har en delayline og derfor først virker efter en stykke tid. Det ses også at det resulterende sinus-signal ender med en amplitude på 1, hvilket også stemmer overens med testsignalet.

På samme måde kan de 4 kHz også fjernes, for at dette kan lade sig gøre, skal fc ændres til 4000, og nye koefficienter skal udregnes. Resultat af dette kan ses på Figur 3 og Figur 4.




*Figur 3 - Frekvensrespons for filter, som fjerner 4 kHz*




*Figur 4 - Resultat af filter i tidsdomænet, hvor de 4 kHz er fjernet.*

## Implementering af makePeakEQ på Blackfin

For at kunne lave en equalizer, hvor det er muligt at ændre koefficienterne on the go, uden behov for at compile en ny applikation, skal der laves en funktion, som udregner nye koefficienter afhængig af brugerens input. I dette tilfælde bliver koefficienterne udregnet ud fra: Sampling frekvensen, centerfrekvens, Q og gain. I Listing 3 ses `makePeakEQ()` funktionen, som gør netop dette. Formlerne som ligger til grund for disse udregninger, er taget fra: Audio EQ Cookbook[1].

I Listing 3 bliver der foretaget en normalisering af koefficienterne i forhold til a0. Dette er for at sikre at koefficienterne ikke overstiger deres maksimale værdi. Skaleringsfaktoren bliver påtrykket alle koefficienterne ved hjælp af et for-loop. Det endelige signal af filteret bliver tilsvarende skaleret tilbage. Hvis man eksempelvis dividerer alle koefficienterne med 2, så vil outputsignalet bliver gange med 2, for at neutralisere denne skalering.

Alle koefficienterne bliver udregnet som en `float`, og bliver derefter castet over i én `long fract`. Hvis der ikke var blevet foretaget en skalering/normalisering før koefficienterne blev caster over i `bg.a` og `bg.b`, så ville nogle af koefficienterne, alt efter input fra brugeren, muligvis få overflow. Hvis dette sker, vil det skabe et filter som ikke opfører sig efter hensigten.

```
void IIRFilter::makePeakEQ(const float fs,
                                        const float fc,
                                        const float Q,
                                        const float A)
{
	float w0 = 2*3.14159265358979*(fc/fs);
	float alpha = sin(w0)/(2*Q);

	float temp1[NUM_IIR_COEFFS];
	float temp2[NUM_IIR_COEFFS];

	temp1[0] = 1+alpha/A;
	temp1[1] = -2*cos(w0);
	temp1[2] = 1-alpha/A;

	temp2[0] = 1+alpha*A;
	temp2[1] = -2*cos(w0);
	temp2[2] = 1-alpha*A;


	for (short i = 0; i < NUM_IIR_COEFFS; i++) {
		bq.a[i] = temp1[i]/(2*temp1[0]);
		bq.b[i] = temp2[i]/(2*temp1[0]);
	}

}
```

*Listing 3 - makePeakEQ function*

Til at teste hvor god filterimplementeringen er på Blackfin, i forhold til et ideelt filter i Matlab, er der blevet genereret et sweep-signal som testsignal. Signalet bliver genereret som en txt-fil, som Blackfin er i stand til at indlæse. Blackfin påtrykker derefter det ønsket filter og generer en ny txt-fil som output. På samme måde påtrykkes det ideelle filter fra Matlab også testsignalet.

[1] Kilde Audio EQ Cookbook: https://webaudio.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html

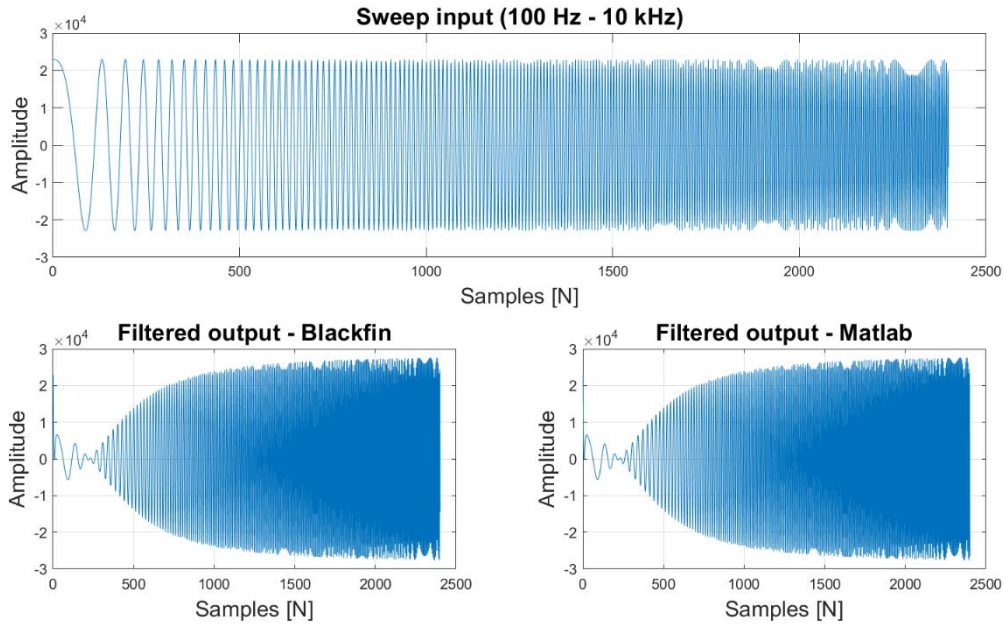
Outputfilen fra Blackfin og resultatet af det ideelle filter fra Matlab, bliver derefter sammenlignet for at se forskellen imellem de to.

På Listing 4 ses testkoden som bliver brugt til at lave et filter, med forudbestemte a og b-koefficienter. De samme koefficienter er også brugt i Matlab-modellen, for at kunne lave en realistisk sammenligning.

```
// Only for testing
// Setting a default notch filter 1 kHz, fs = 48 kHz, r = 0.8
const fract aTest[NUM_IIR_COEFFS] = {0.5r, -0.7932r, 0.3200r};
const fract bTest[NUM_IIR_COEFFS] = {0.5r, -0.9914r, 0.5000r};

void IIRFilter::create(void)
{
	// Initialization of coefficients
	// TODO change method to make a pass through filter (a0 = 1.0, b0 = 1.0)
	bq.a[0] = 1.0;
	bq.a[1] = 0;
	bq.a[2] = 0;

	bq.b[0] = 1.0;
	bq.b[1] = 0;
	bq.b[2] = 0;


	for (short i = 0; i < NUM_IIR_COEFFS; i++) {
		bq.a[i] = aTest[i];
		bq.b[i] = bTest[i];
	}

}
```

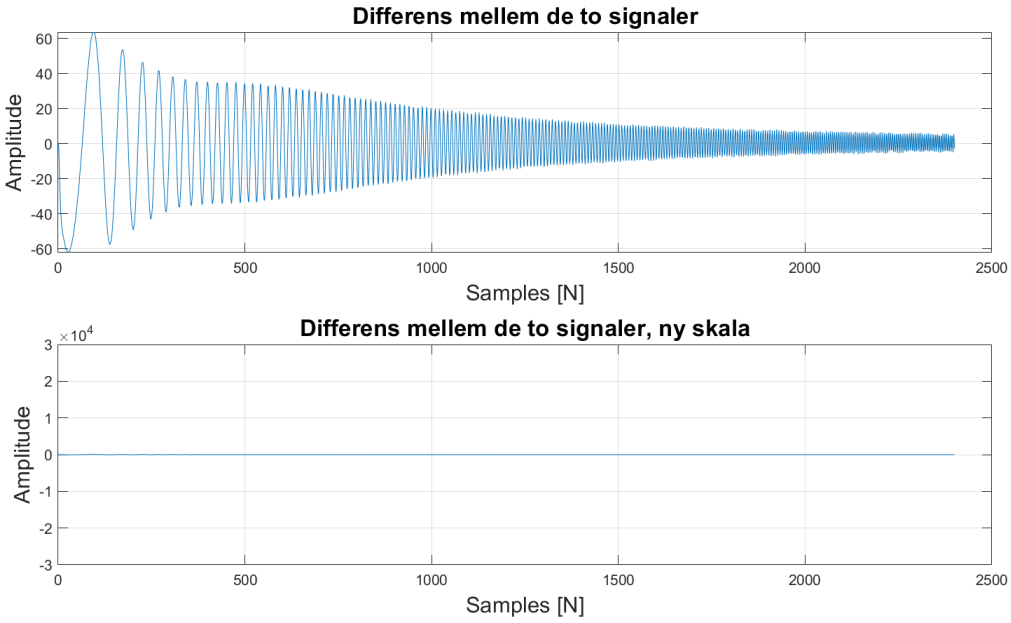
*Listing 4 - Konstante koefficienter til test af filteret.*

På Figur 5, ses det genereret testsignal som går fra 100 Hz til 10 kHz. De to nederste grafer viser henholdsvis resultatet fra Blackfin og resultatet fra Matlab. Ved første øjekast ser de to signaler umiddelbart ud til at være ens.




*Figur 5 - Matlab filter VS Blackfin filter*

For at finde ud af hvor stor forskel der er imellem de to signaler, tages differensen af dem, hvilket kan ses på Figur 6. Den øverste graf viser den direkte differens med automatisk indstillet y-skala. Her ligner det umiddelbart, at der er stor forskel imellem de to signaler. Forskellen er størst i starten af signalet, hvilket også give mening, da det er her filteret bliver påtrykket signalet, og der derfor vil være en yderligere kvantiseringsfejl i form af filter-koefficienterne. Derfor bliver forskellen imellem de to signaler også mindre, jo mere filteret aftager. Hvis man derimod kigger på den nederste graf, hvor der er samme y-skala som på Figur 5, ligner det umiddelbart ikke at er er noget mærkbar forskel.




*Figur 6 - Differens mellem de to signaler*

## Implementering af 4 band EQ

På Listing 5 ses `create()` funktionen som bliver kaldt for hver EQ-objekt. I dette tilfælde er der valgt at denne EQ skal have fire bånd. Hver af de fire bånd får nogle default værdier. Et gain på 0.3 og en Q på 10. Hver af de fire bånd bliver derefter indstillet med en default cut-frekvens. Derefter bliver alle koefficienterne udregnet for hver af disse fire bånd, med funktionen: `updateEQParameters()`.

```
void Equalizer::create(void)
{
	for (short band = 0; band < NUM_EQ_BANDS; band++)
	{
		m_IIRfilter[band].create();
		m_EQParams[band].gain = 0.3; // Default gain
		m_EQParams[band].Q = 10; // Default Q
	}
	// Setting default peak bands center frequency
	m_EQParams[0].fc = 200;
	m_EQParams[1].fc = 600;
	m_EQParams[2].fc = 2000;
	m_EQParams[3].fc = 6000;

	updateEQParameters(0);
	updateEQParameters(1);
	updateEQParameters(2);
	updateEQParameters(3);
}
```

*Listing 5 - Alle fire bånd bliver oprettet, og bliver indstillet med deres default parametre*

På Listing 6 ses hvordan inputsignalet bliver påtrykket alle fire bånd i EQ. For at spare på hukommelsen, bliver der benyttet en `tmpBuf`. Denne buffer bliver skiftevis fyld og tømt, sammen med `output`. Denne buffer er defineret i den tilhørende .h-til til `Equalizer`.

```
void Equalizer::process(short* input, short* output, short len)
{
	// TODO change code to processing all IIR filters
	if (m_filterOn)	{
		m_IIRfilter[0].process(input, tmpBuf, len);
		m_IIRfilter[1].process(tmpBuf, output, len);
		m_IIRfilter[2].process(output, tmpBuf, len);
		m_IIRfilter[3].process(tmpBuf, output, len);
	}
	else
		// Pass through, just copy input block to output
		for (short i = 0; i < len; i++)
			output[i] = input[i];
}
```

*Listing 6 - Alle fire bånd bliver påtrykt inputsignalet*

På Listing 7 og Listing 8, ses koden for hvordan man enten øger eller sænker gain i et bånd. Alt efter hvilket bånd der er valgt, ændres gain ud fra værdien der er valgt som `delta`. Efterfølgende bliver funktionen `updateEQParameters()` kaldt, som går ind og genudregner a og b koefficienterne for filteret.

```
float Equalizer::incParameter(short band, PARAMETER param, float delta)
{
    float value = 0;
    //m_filterOn = true; // For test only

    setParameter(band, param, m_EQParams[band].gain + delta);
    updateEQParameters(band);

    value = m_EQParams[band].gain;


       return value;
}
```

*Listing 7 - Øger gain i det pågældende bånd med delta*

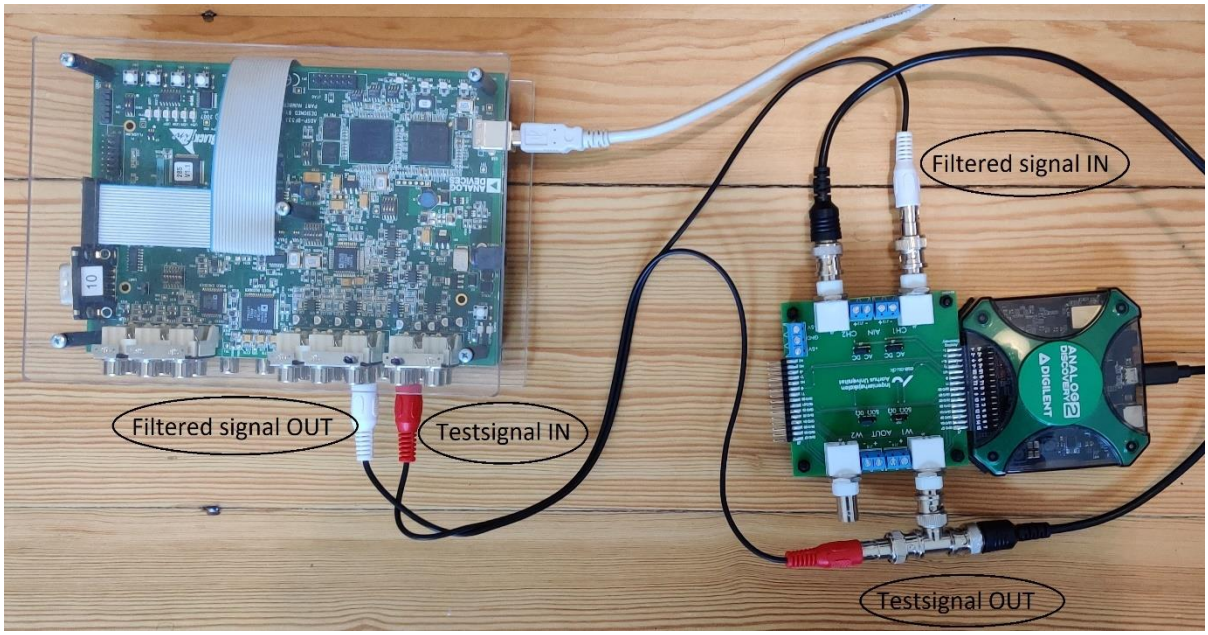
```
float Equalizer::decParameter(short band, PARAMETER param, float delta)
{
    float value = 0;
    //m_filterOn = false; // For test only

    setParameter(band, param, m_EQParams[band].gain - delta);
    updateEQParameters(band);

    value = m_EQParams[band].gain;

       return value;
}
```

*Listing 8 - Sænker gain i det pågældende bånd med delta*

Koden i Listing 7 og Listing 8, bliver aktiveret via et interrrupt, når der bliver trykket på henholdsvis SW4 eller SW5 på Blackfin.

## Test og verificering af EQ på Blackfin

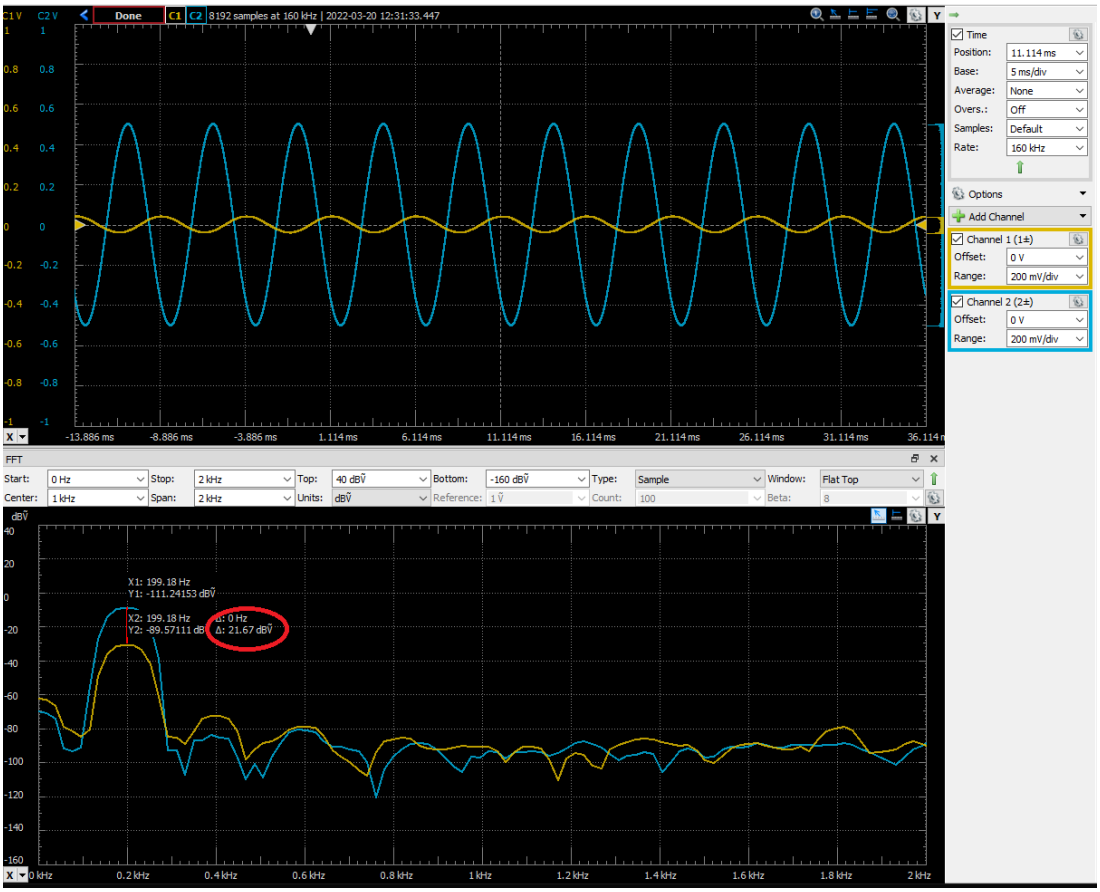
På Figur 7 ses et testsetup hvor Analog Discovery 2 både bliver brugt som signalgenerator og som oscilloskop.




*Figur 7 - Testsetup*

Som testsignal vil der først blive genereret et sinussignal, som har frekvensen, der er lig med fc i hvert bånd. Derefter bliver der lavet et sweepsignal, for at se alle bånd i aktion på samme tid. I alle tilfælde er der tale om en dæmpning, da gain er 0.3.

På Figur 8, Figur 9, Figur 10 og Figur 11, ses alle de fire fc-frekvenser som inputsignal. Det gule signal på figurerne er det filtreret signal igennem Blackfin, hvor det blå signal er signalet fra signalgeneratoren. Det ses tydeligt hvordan den pågældende frekvens bliver dæmpet i tidsdomænet. Ved at bruge den indbygget FFT på Analog, ses det også at det er præcis den frekvens der bliver påvirket i frekvensdomænet.




*Figur 8 - 200 Hz sinussignal*

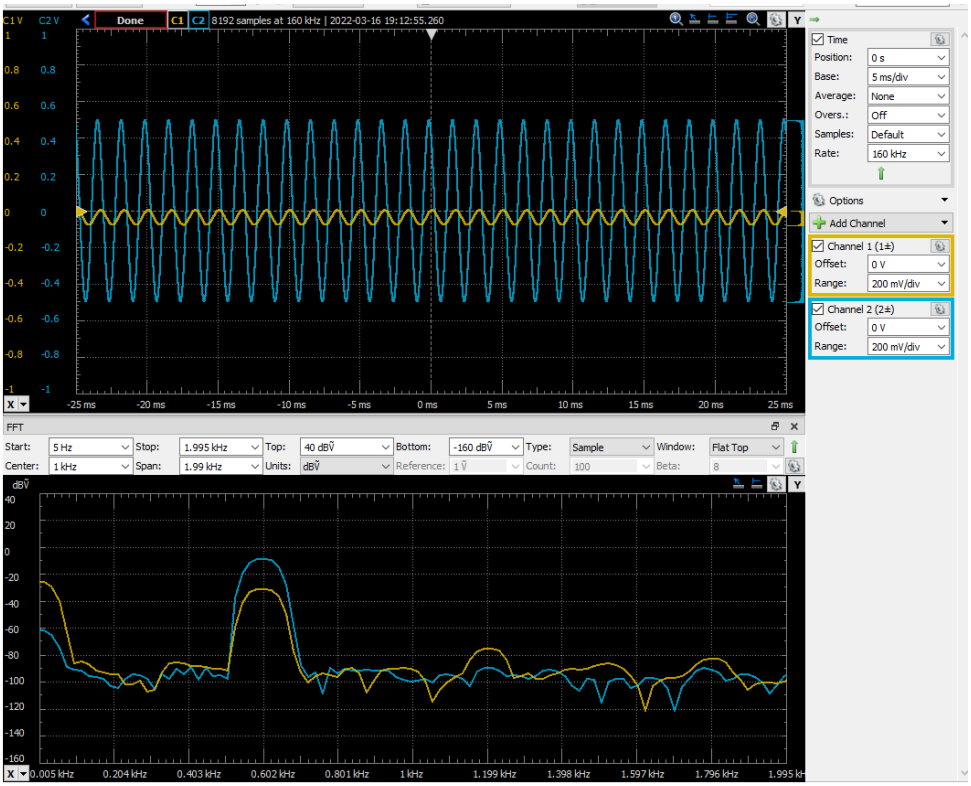
På Figur 8 fremgår det af den røde cirkel, at der er ~21.67 $dB$ til forskel imellem de to signaler, ved 200 Hz. Det svarer til et gain på:

$$A_{gain} = 10^{\frac{dBgain}{40}}$$

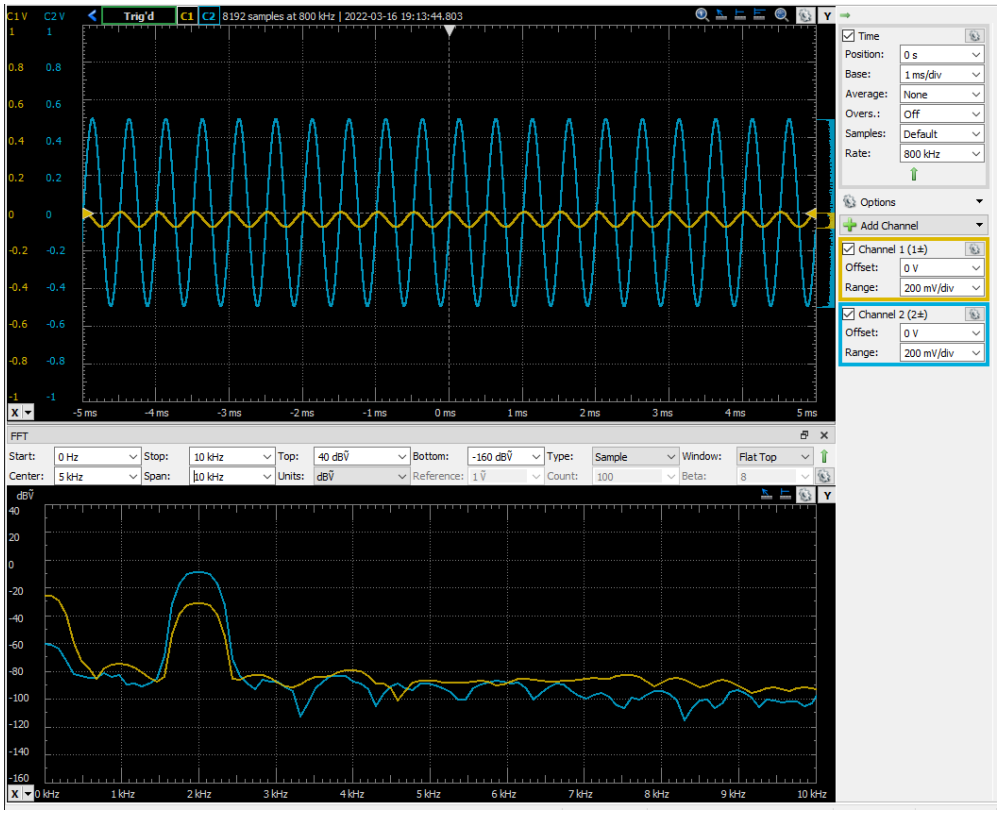
$$\downarrow$$

$$A_{gain} = 10^{\frac{-21.67\ dB}{40}} = 0.28\ \approx 0.3$$

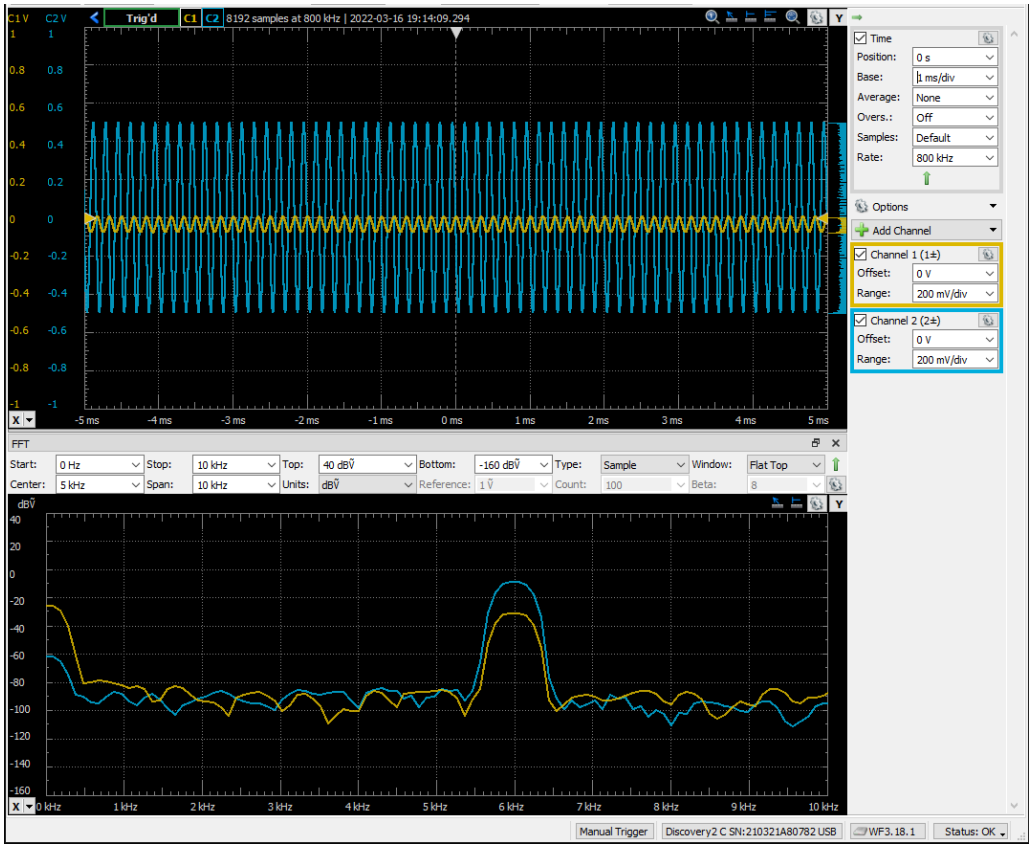
Hvilket stemmer overens med det gain, som er brugt til at udregne koefficienterne.




*Figur 9 - 600 Hz sinussignal*

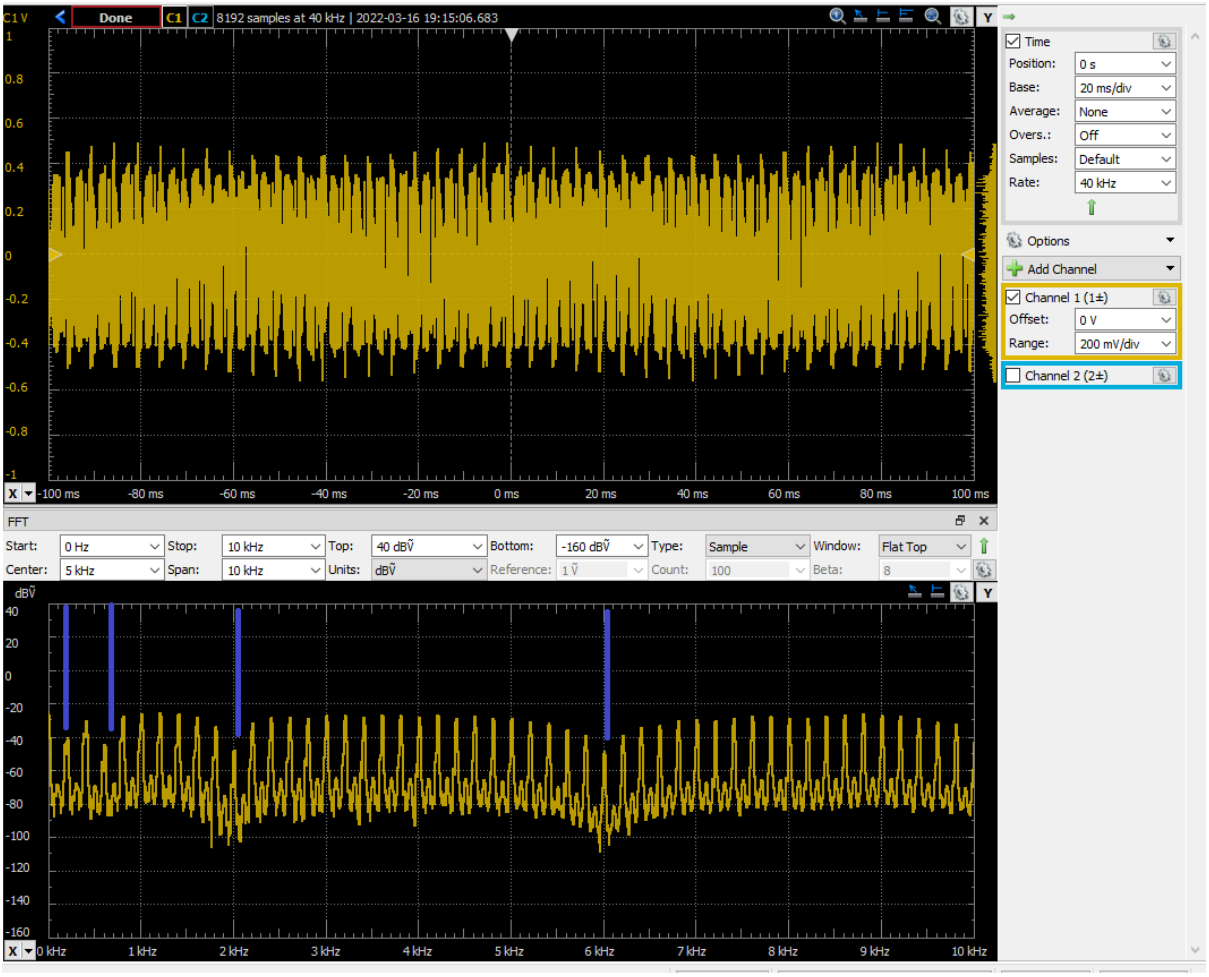



*Figur 10 - 2 kHz sinussignal*



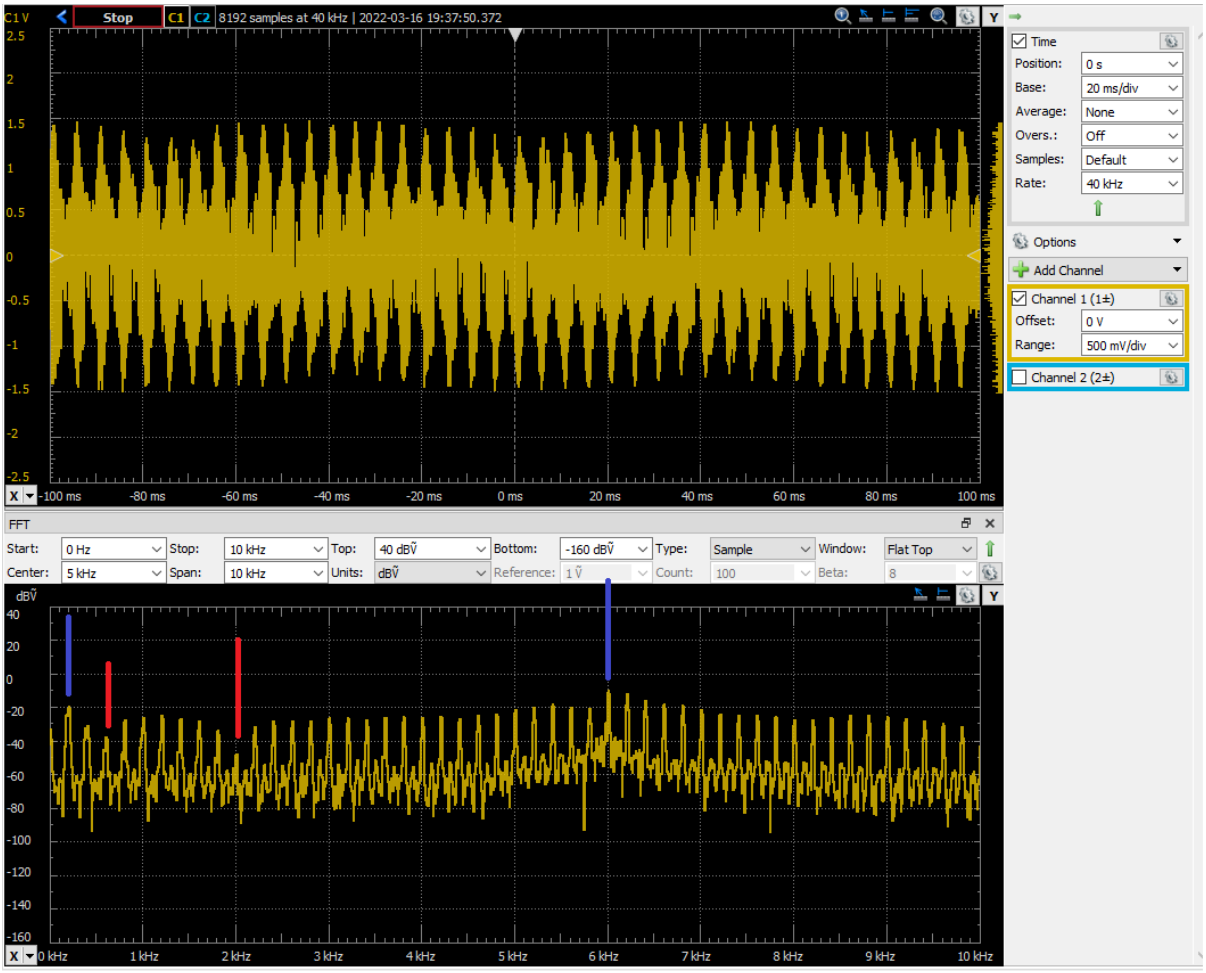

*Figur 11 - 6 kHz sinussignal*

På Figur 12 ses et sweepsignal der går fra 50 Hz til 10 kHz. I dette tilfælde er alle fire bånd på EQ aktiveret. Her ses bedst i frekvensdomænet hvordan de fire bånd påvirker hver deres frekvens. Det ses også hvordan de forskellige bånd dæmper nogle af de omkringliggende frekvenser. Dette kan dog gøres ved at øge Q, for at gøre selve båndbredde for filteret mindre.




*Figur 12 - Sweepsignal. 50 Hz - 10 kHz*

Ved at bruge SW7 til at vælge bånd, og SW4 til at øge gain, kan forstrækning øges på de forskellige bånd, imens programmet kører. Resultat af dette kan ses på Figur 13, hvor det forstærket frekvenser er markeret med blå, og de dæmpet frekvenser er markeret med rød. Det kan derfor konkluderes at de forskellige koefficienter, for det pågældende bånd, bliver udregnet og opdateret korrekt.




*Figur 13 - Forstrækning fra 200 Hz og 6 kHz*