

AARHUS UNIVERSITET

INDLEJRET SIGNALBEHANDLING

6. SEMESTER

ISB projekt

Gruppemedlemmer:

Søren Landgrebe

Stinus Lykke Skovgaard

Studienr:

201508295

201401682



19. maj 2018

Indhold

1	Indledning	3
2	Krav	4
3	Aktørbeskrivelse	4
4	Use case beskrivelse	5
4.0.1	UC 1 - Turn on filter	5
4.0.2	UC 2 - Turn off filter	5
4.0.3	UC 3 - Filter aktiv	5
5	Ikke-funktionelle krav	6
5.1	Problemrelateret krav	6
5.2	System og algoritme krav	6
5.3	Afledte krav	6
6	Accepttest	6
7	Struktur	7
7.1	BlackFin blokforklaring	7
7.2	Software arkitektur	8
8	Teori	10
9	Test	11
9.1	Matlab simulering	11
9.1.1	Første test	12
9.1.2	Anden test	15
9.2	Cross-core	17
9.2.1	Overordnet	17
9.2.2	Visuel test	17
9.2.3	Audio Test	20
10	Diskussion	21

Figurer

1	Konceptbillede for NSS	3
2	Aktør Kontekst diagram	4
3	Usecase diagram	5
4	Struktur NSS	7
5	Klassediagram over NSS	8
6	LMS adaptive filter	10
7	LMS filter i tid (3 sinus toner)	12
8	LMS filter i frekvens (3 sinus toner)	13
9	LMS filter(3 sinus toner)	14
10	LMS filter i tid (food processor)	15
11	LMS filter i frekvens (food processor)	16
12	LMS filter(food processor)	17
13	LMS filter implementeret på Blackfin	18
14	Simulering af figur 13 i matlab	19
15	Fejl fra matlab filter	20

1 Indledning

Under optagelse til et live madprogram, sker det ofte at værten skal bruge en blender/food-processor. Dette betyder at værten ikke kan kommunikere med seerne mens blenderen/food-processoreren kører. Denne problematik vil Noise Suppression System (NSS) kunne afhjælpe. Gennem en digital signal processing (DSP), vil vi dæmpe støjsignalet fra en køkkenmaskine dynamisk i realtid. Systemet består af to mikrofoner, et placeres tæt på støjen, et andet tæt på værten. De to mikrofoner fungerer som input til vores system (Blackfin), hvor processeringen og filtreringen foregår. Efter proceseringen bliver produktet afspillet på en højttaler, som erstatning for højttaleren fra et tv-apparat. Et overblik over systemet kan ses på figur 1



Figur 1: Konceptbillede for NSS

Med udgangspunkt i brugerens behov vil der blive opstillet en række brugsscenarier, der beskriver brugerens interaktion med systemet. Disse scenarier vil sammen med en række veldefinerede krav og afgrænsninger, danne grundlaget for designet af alle dele af systemet.

2 Krav

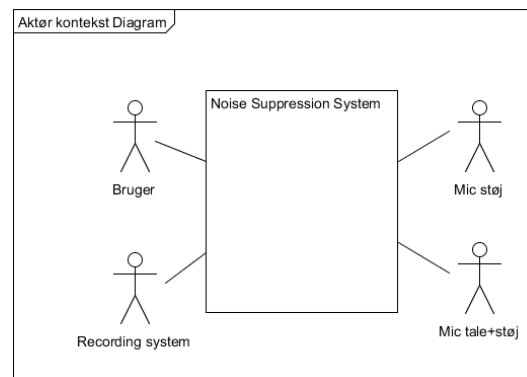
I dette afsnit beskrives kravene til hvilken funktionalitet systemet har.

I samarbejde med vejleder er der opstillet en række krav.

- Systemets skal kunne filtrere en støj fra et lydsignal, som indeholder et tale signal overlappet af et støjsignal.
- Brugeren skal manuelt kunne tænde og slukke for filtreringen.
- Output lydsignalet skal feedes til en højttaler som afspiller lyden fra mikrofonen.

Kravene er bygget op gennem use cases, som beskriver systemets funktionelle krav, samt en liste over alle ikke-funktionelle krav for systemet. Først vises aktørbeskrivelserne med tilhørende aktørkontekst diagram. Herefter vises use cases for systemet. Der er udfærdiget to use cases der tilsammen beskriver funktionaliteten af systemet. I use case afsnittet vises der også et use case diagram med alle use cases og hvordan aktørerne interagerer med dem. Til sidst gives et kort overblik over ikke funktionelle krav.

3 Aktørbeskrivelse



Figur 2: Aktør Kontekst diagram

På figur 2 ses aktør kontekst diagrammet som beskriver sammenhængen mellem aktørerne og det system de interagerer med. Aktørerne er som følger:

Primær

Bruger: Den aktør der interagerer med systemet og vælger den ønskede funktionalitet

Recording system: Den aktør der modtager det enedelige produkt

Sekundær

Mic støj: Et input til det samlede system

Mic tale+støj: Et input til det samlede system

4 Use case beskrivelse



Figur 3: Usecase diagram

På figur 3 ses usecase diagrammet som beskriver sammenhængen mellem aktørerne og de forskellige funktionaliteter der findes for systemet.

4.0.1 UC 1 - Turn on filter

Brugeren trykker på SW1 og filteret aktiveres.

4.0.2 UC 2 - Turn off filter

Brugeren trykker på SW1 og filteret deaktiveres.


4.0.3 UC 3 - Filter aktiv

Recording system modtager det filtrede lyd, hvis prækonditionen UC 1 er udført.


5 Ikke-funktionelle krav

Kravene er delt op i tre underkategorier. Krav der relaterer til problemet, krav der relaterer til DSP platform og algoritme. Til sidst er der en kategori der beskriver kravene til systemet på baggrund af de to første kategorier.



5.1 Problemrelateret krav

1. R1: Systemet skal have 2 mikrofoner og 1 højttaler
2. R2: Filteret skal gøre brug af LMS algoritmen
3. R3: Systemet skal kunne processerer lyd i frekvensbåndet 50-20000Hz.
4. R4: Systemet skal kunne dæmpe uønkset støj 30dB. 
5. R5: Systemet skal kunne dæmpe støj uden at dæmpe ønsket lydsignal.
6. R6: Systemet burde have en latency under 30ms.
7. R7: Systemet burde have et dynamikområde på min 80dB


5.2 System og algoritme krav

8. R8: Filter algoritmen skal implementeres med fixed point
9. R9: Filteret skal max bruge 10kByte memory
10. R10: Filteret skal implementeres på Blackfin BF533 
11. R11: Filteret må max benytte 98% DSP load

5.3 Afledte krav

12. DR1: DSP systemet skal kunne håndtere en samplingsrate på min 44100kHz(På baggrund af krav R3)
13. DR2: Filteret skal implementeres med 1.15 fixed point.(På baggrund af krav R7 og R8. Dette giver et dynamikområde på 96dB)
14. DR3: Filter latency må max forsinkes 1280 samples(På baggrund af R6. $(1/44100)*1280=30\text{ms}$) 
15. Filteret må max bruge 13333 cycles af DSP processing for hver sample. 

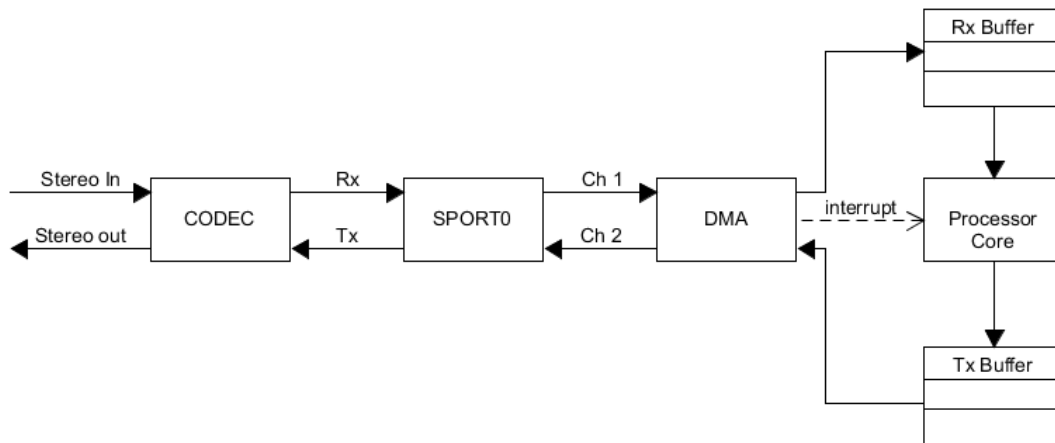
6 Accepttest

Til hver use case er der lavet en tilhørende accepttest, der tjekker om use casen bliver gennemført korrekt. Udover de tre accepttests er der oprettet en accepttest af de ikke-funktionelle krav til systemet. Alle accepttests kan ses i kravspecifikations dokumentation. 

7 Struktur

7.1 BlackFin blokforklaring


I dette projekt er der taget et valg ud fra læringsmålene om at vi bruger Blackfin platformen til at udføre projektets funktionalitet. Da en blackfin processor er bygget op af mange funktionelle blokke, vil der i det kommende afsnit laves et overblik over de **hardware** blokke som bliver brugt i dette projekt. [2]



Figur 4: Struktur NSS

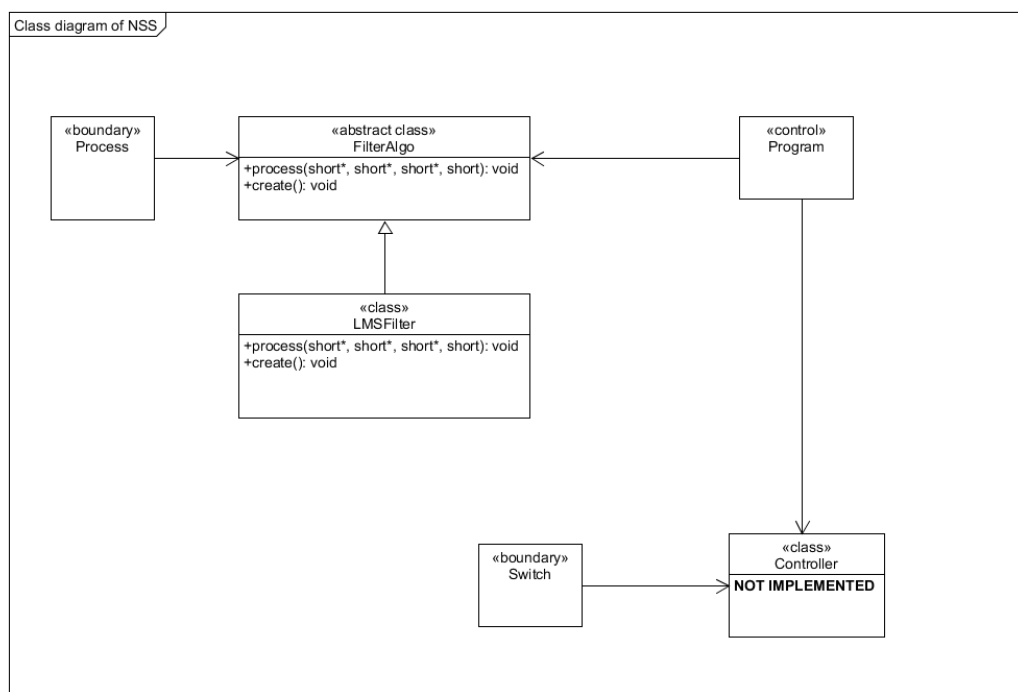
Igennem processen af vores funktionalitet, sendes lydsignalet gennem en codec1836, som har til opgave at sende inputtet gennem en ADC, hvorefter den sender det digitale signal videre til SPORT0, som står for at modtage og sende dataen.

DMA'en står for at sende dataen videre til Rx bufferen, når bufferen er fuld sender DMA'en interrupt til processeren om at procesere den modtagne data.

I den modsatte retning bliver det proceserede data flyttet til Tx bufferen, når Tx bufferen er fuld, sendes det til SPORT0 via DMA'en med Ch 2. Til sidst bliver dataen samlet gennem CODEC, som konverterer til et analogt signal vha. en DAC. De forskellige blokke i strukturen fra **figur 6**, er implementeret i **klassen Init**. 

7.2 Software arkitektur

Til dette projekt har det været muligt at benytte et udleveret framework til Crosscore. Dette framework er lavet af Kim Bjerger, og er blevet modificeret af gruppen, til at virke med vores LMS filter. På [Figure 5](#) kan man se hvilke klasser der er med og deres relationer til hinanden.



Figur 5: Klassesdiagram over NSS

Arkitekturen består af en abstrakt klasse `FilterAlgo` som har to funktioner, `process()` og `create()`. Disse to nedarves af `LMSFilter` klassen og implementeres heri. `Create()` bliver kaldt først i `main` og nulstiller alle koefficienter, udover den første som bliver sat til 1. Grunden til dette er for at få det fulde støjsignal igennem første gang filteret kører. Dette vil blive uddybet mere i diskussion afsnittet. `process()` funktionen står for at filtrere uønsket støj fra tale signalet($d(n)$). Koden er lavet så den følger vores matlab model så godt som overhovedet muligt. Dette kan også ses i [7.2](#)

```

1  for(short i = 0; i < len; i++)
2  {
3      long fract yn = 0;
4
5      for(short j = 0; j < NUM_WEIGHTS; j++)
6      {
7          if(i > j)
8          {
9              yn = yn + Filter.W[j]*x[i-j];
10         }
11     }
12     y[i] = (fract)yn;
13     e[i] = d[i] - yn;
14
15     long fract tmp_W = 0;
16
17     for(short k = 0; k < NUM_WEIGHTS; k++)
18     {
19         if(i > k)
20         {
21             Filter.W[k] = Filter.W[k] + my*x[i-k]*e[i];
22         }
23     }
24 }

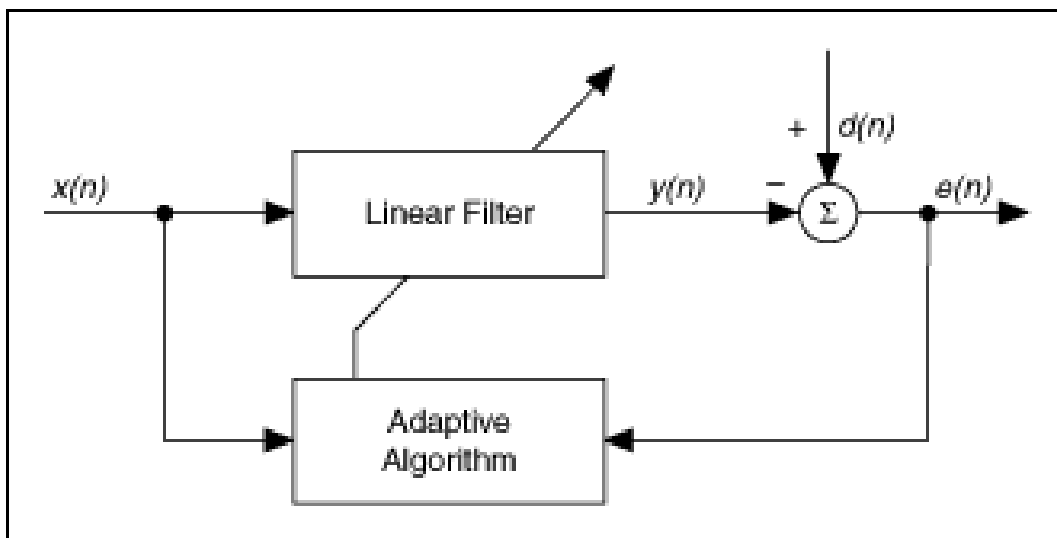
```

For at forsimple koden har vi valgt at splitte vores stereo kanal op i to, så det er muligt at få støjsignalet($x(n)$) ind på højre kanal og tale signal($d(n)$) ind på venstre kanal.

Vi har valgt ikke at implementere controller klassen, da vores 1. prioritet har været at få et fungerende LMS filter. Derfor står den som "NOT IMPLEMENTED". Istedet for at brugeren kan tænde for filteret med en switch, har vi valgt at lade filteret være aktiv så snart man starter programmet.

8 Teori

I procceseringsfasen af projektet, har vi valgt at bruge et adaptivt filter - LMS (Least Mean square) algoritme. LMS er et adaptivt filter som består af 2 funktionelle blokke, hvor den ene blok (Linear Filter) fungerer som et filter, det andet (Adaptive Algorithm) som et dynamisk beregner af nye koefficienter til første blok. Filteret trækker herefter de beregnede filter fra det samlede lydssignal.



Figur 6: LMS adaptive filter

På figur 6 ses et overblik over det adaptive system, hvor $x(n)$ er støjsignalet, $y(n)$ er det filtrede støjsignal, med koefficienter som opdateres fra blokken "Adaptive Algorithm". $d(n)$ er det ønskede signal inklusiv det støjende signal. $e(n)$ er forskellen mellem $d(n)$ og $y(n)$ og derved støjen fratrasket fra det samlede signal af ønsket og støj.[1]

Det digitale filter bliver beregnet ud fra formlen:

$$y(n) = \sum_{l=0}^{L-1} W(n) * x(n-l) \quad (1)$$

Hvor $W(n)$ er den værdi, som dynamisk opdateres. Dette sker ved at vi beregner den næste værdi ud fra formlen:

$$W(n+1) = W(n) - \mu * X(n) * e(n) \quad (2)$$

Hvor W er den nye koefficient til filteret, $X(n)$ er input signalet, og μ er en faktor, som bestemmer hastigheden af filteret, samt styrer infaldstiden. Hvis μ er lav bliver filteret langtsommere, mens settling time stiger jo højere vi kommer. Typisk må denne værdi ikke overstige 1.

Dette giver os et endeligt udtryk som stemmer overens med figur 6, ift summeringspunktet:

$$e(n) = d(n) - y(n) \quad (3)$$

9 Test

9.1 Matlab simulering

Efter den teoretiske undersøgelse, vil vi i dette afsnit udvikle og vise en simuleret version af vores filter, hvor der er taget udgangspunkt i Teori afsnittet. Da vi tager udgangspunkt i teoriafsnittet starter vi med at vise hvordan vi har implementeret vores filter.

```
1 %Create LMS FIR filter
2 my = 0.01; % some number 0.01
3 W = zeros(1,256);
4
5 for n = 1:length(d) %run every sample
6     yn = 0;
7     for m = 1:length(W) %make new filteret sample
8         if n > m
9             yn = yn + fixed32(W(m)*noise(n-m));
10        end
11    end
12    y(n) = yn;
13    e(n) = d(n) - y(n);
14    for m = 1:length(W) %make new koefficient
15        if n > m
16            W(m) = W(m) + fixed32(my*noise(n-m)*e(n));
17        end
18    end
19 end
```

Først i filteret vælger vi en my , som ift til teori afsnittet bestemmer hvor hurtig filteret er. I dette eksempel bruger vi en værdi som er testet frem til at have et god forhold mellem hastighed og settling time. Herefter bestemmes hvor mange koefficienter filteret skal have. Vi har valgt et forholdsvis højt tal, da vi simulerer. Dette vil ikke nødvendigvis være muligt på Cross-core.

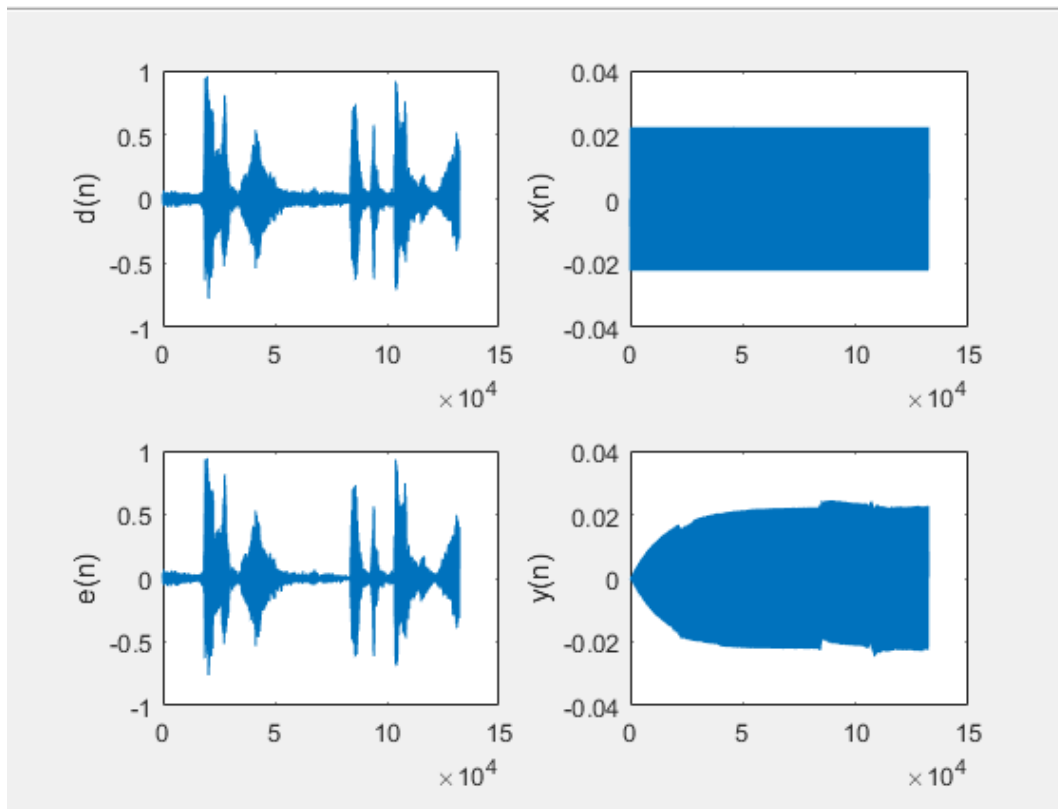
Igenennem genereringen af filteret laves der 3 forloop. Et som kører hver sample igennem, et som laver det filtrerede nye sample, og et som opdaterer koefficienterne ift tilbagekoblingen. De nye filter koefficienter bliver herved opdateret til det ønskede filter, hvor $e(n)$ bliver fejlen ift figur 6, som er det talesignal der ønskes.

Koden er bygget op sådan at vi kører en `fixed16()` og `fixed32()`, for at simulere det bedst muligt ift blackfin. Fixed16 svarer til fixed point 1.15 og fixed32 til fixed point 1.31.

Dette giver os et filter som fungerer efter hensigten. Først testes der med 3 sinus toner, som ligger indover et tale signal, disse 3 toner skal derved gerne blive filtreret gennem filteret.

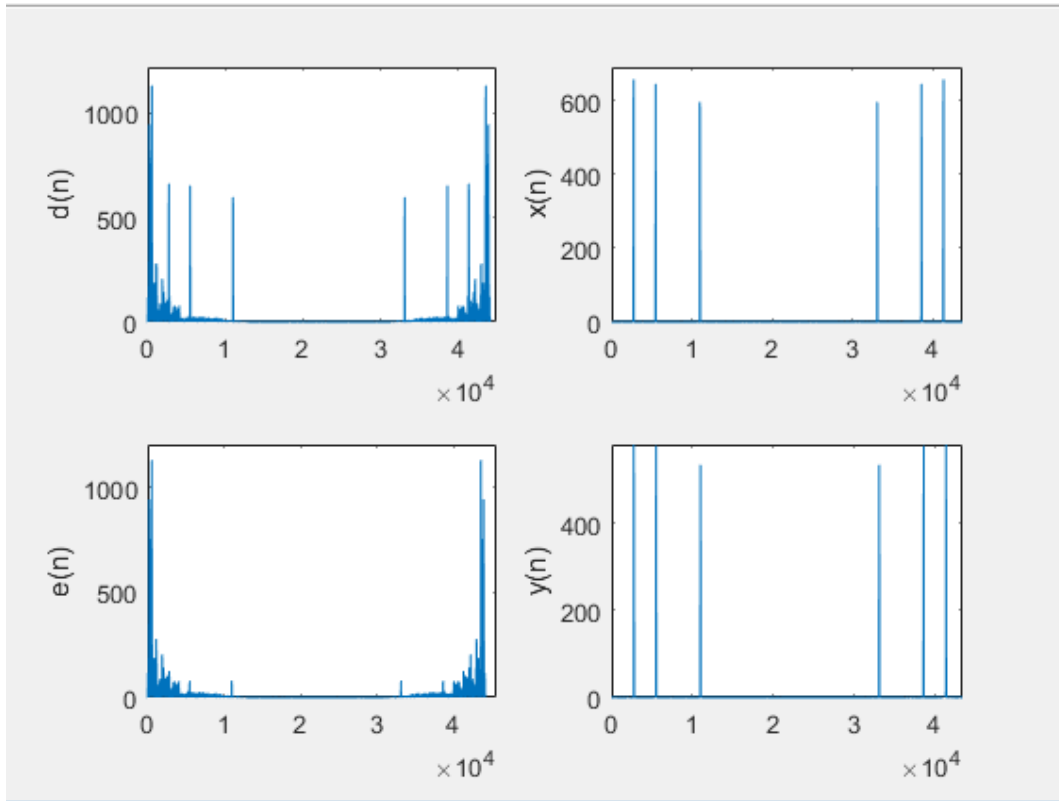
9.1.1 Første test

Den første test der blev udført var så simpel som mulig. I forsøget genereres et talesignal med 3 sinustoner indover. Figur 7 viser signalet som bliver filtreret i tidsdomænet. Her er det værd at observere settling tiden på filteret som ses i $y(n)$. Denne settling tid vil kunne gøres mindre ved at justere på μ . Der er valgt til opgaven her at teste med 0.01 my.



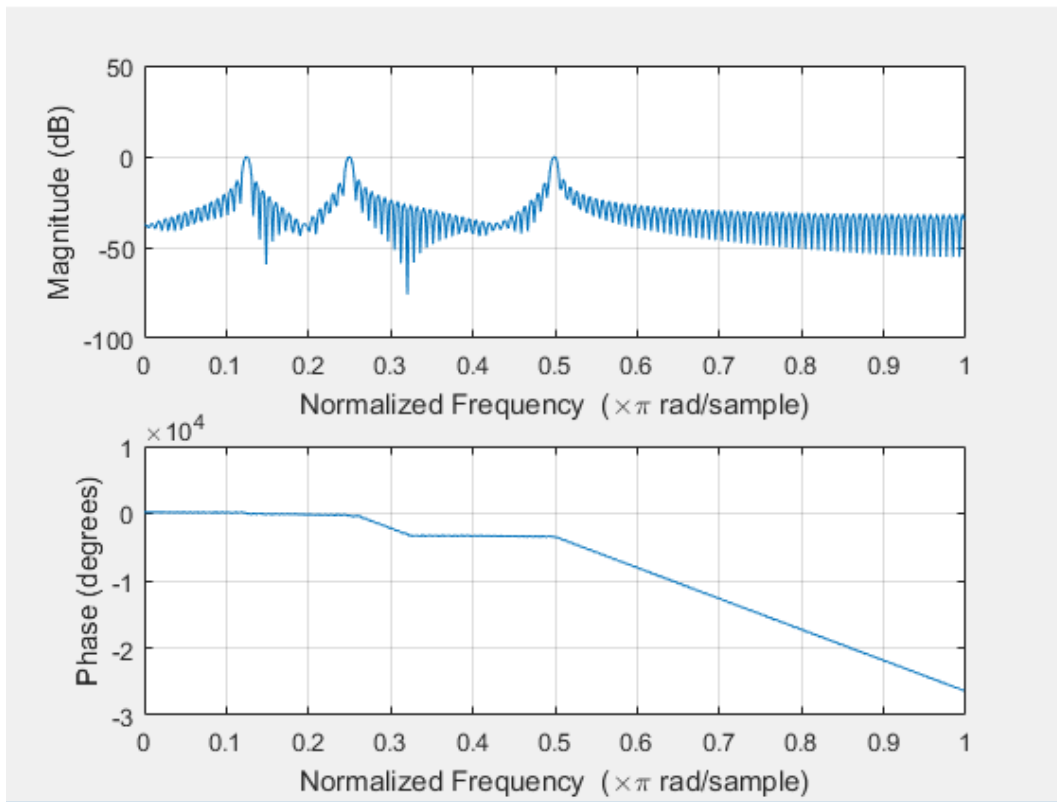
Figur 7: LMS filter i tid (3 sinus toner)

Herefter vises på figur 8 signalet i frekvens domænet, her ses hvordan filteret filtrere den fejl som skabes fra, og derved skaber et signal $e(n)$, som er talesignalet med dæmpet sinus tonerne.



Figur 8: LMS filter i frekvens (3 sinus toner)

Ved at kigge på selve filteret ses at filteret (figur 9) skaber et filter som kun lukker de 3 sinus toner igennem, hvilket er hensigten. Dette signal bliver så fratrasket det endelige samlede signal, for at få en error som er det endelige $e(n)$.

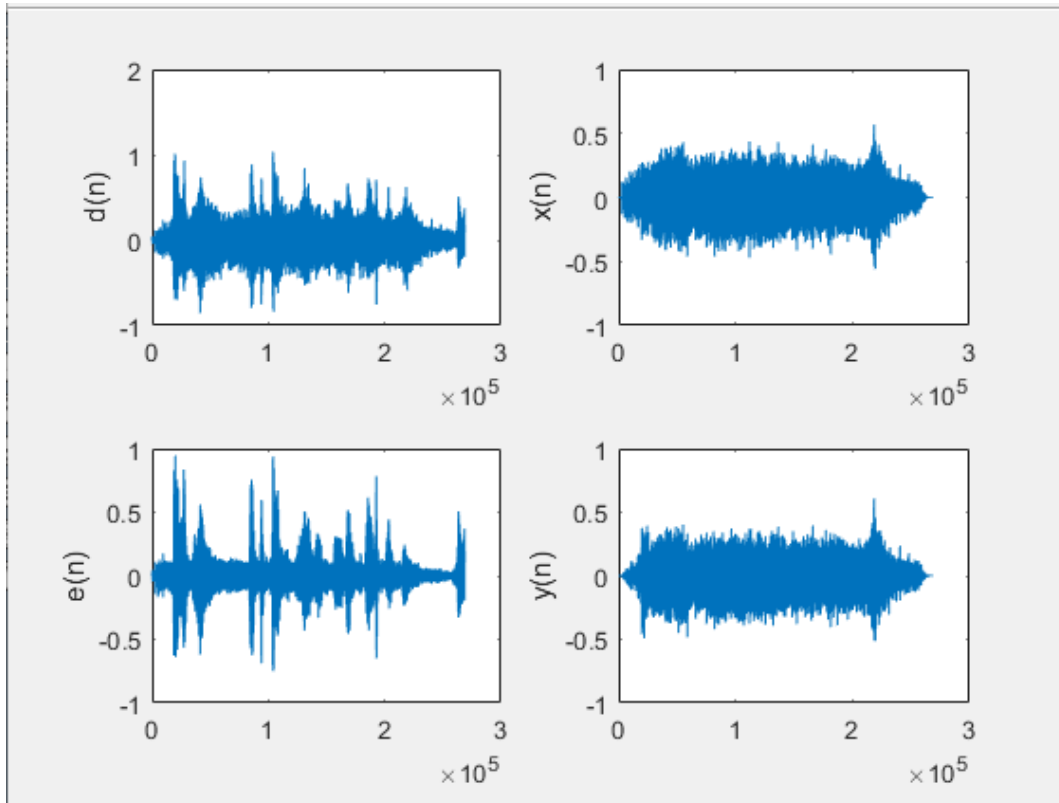


Figur 9: LMS filter(3 sinus toner)

9.1.2 Anden test

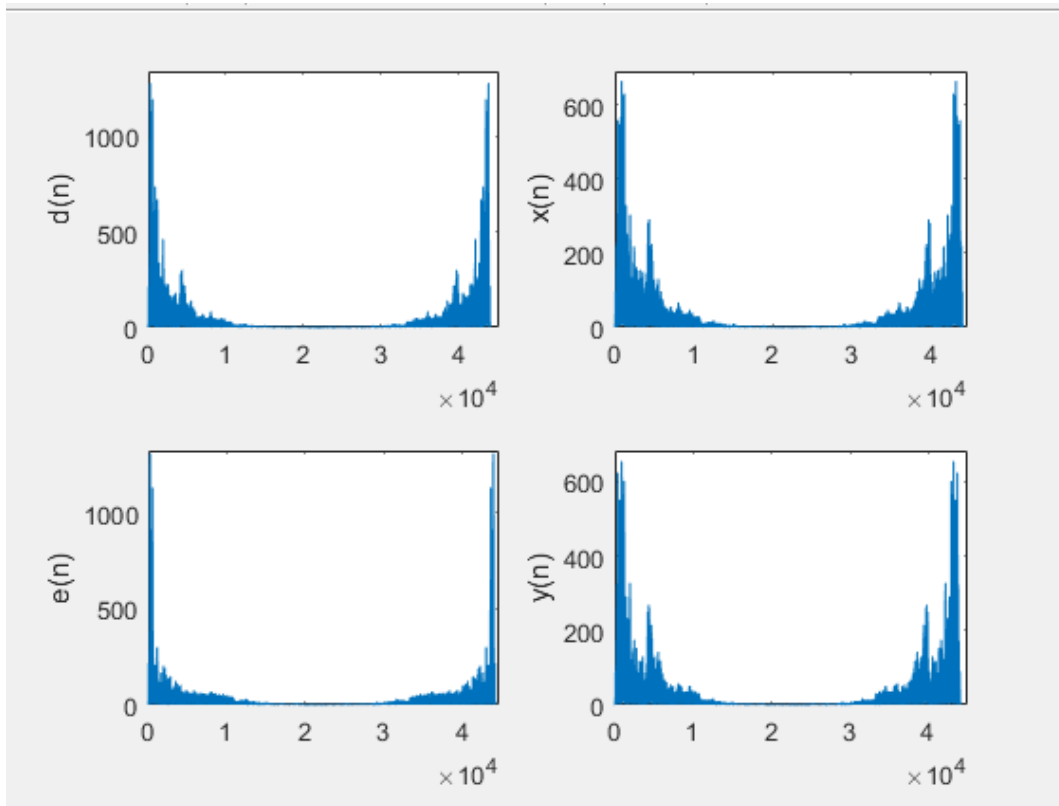
Første test var i den nemme og simple ende, da vi kun skulle filtrere rene toner fra. Der vil i anden test forsøges at filtrere en foodprocessor som bekræftet i indledningen. Denne opgave er del mere udfordrende for filteret da den nu skal filtrere mange toner og ikke kun 3, samtidig med den skal lade nogle bestemte toner gå igennem, som kommer fra talesignalet.

På figur 10, ses nu det endelige produkt i tidsdomænet, der ses her en kraftig filtrering af signalet fra $d(n)$ til $e(n)$. Igen ser vi også en $y(n)$ der har en settling time, inden filteret begynder at virke efter hensigten.

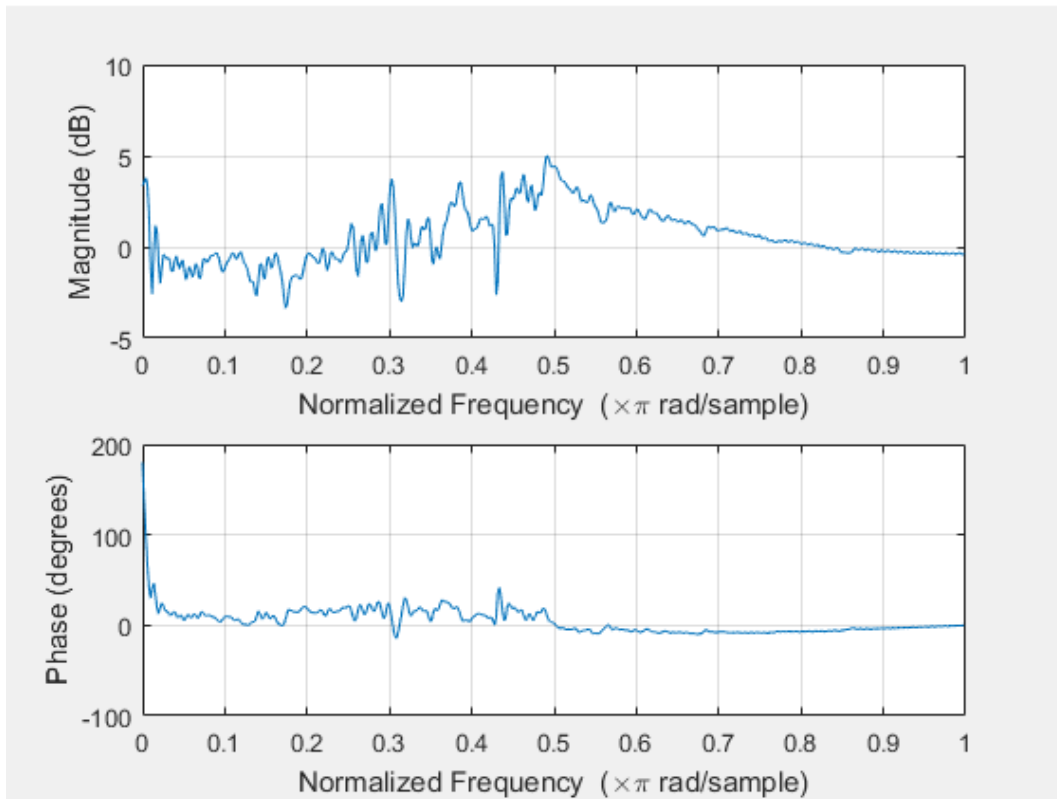


Figur 10: LMS filter i tid (food processor)

Herefter tager vi et blik på frekvens domænet som viser lidt det samme billede som tidsdomænet, at LMS filteret filtrerer kraftigt fra $d(n)$ til $e(n)$, dette kommer af at foodprocessoren indeholder mange frekvenser. Hvis vi kigger lidt på figur 11, og figur 12, kan man nærstudere figur 12, og se at filteret fungerer bedst når frekvenserne er udenfor talefrekvenserne, dette betyder også at filteret ikke prøver at filtrere støj fra som ligger i tale signalet, dette kan vi se fra figur 12.



Figur 11: LMS filter i frekvens (food processor)



Figur 12: LMS filter(food processor)

9.2 Cross-core

9.2.1 Overordnet

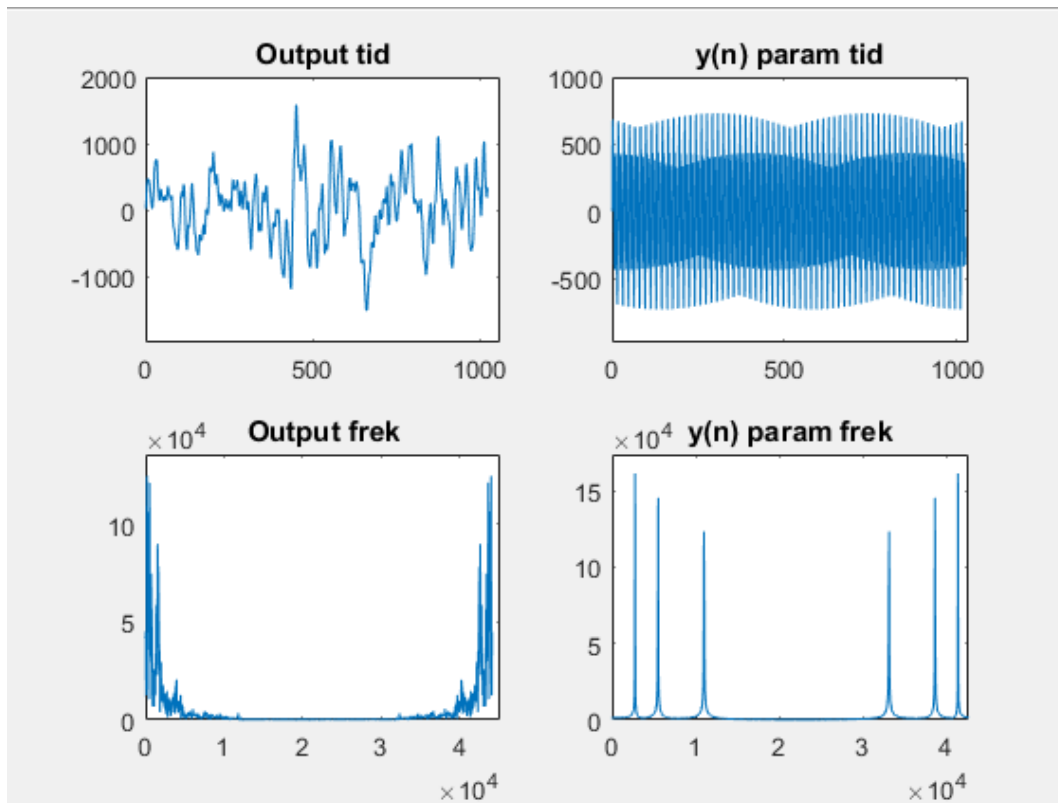
Gruppen har valgt at lave test ud fra funktionalitet frem for use cases, da mange af use casene ikke vil godkendes, og derfor giver en forklaring af testene mere mening.

9.2.2 Visuel test

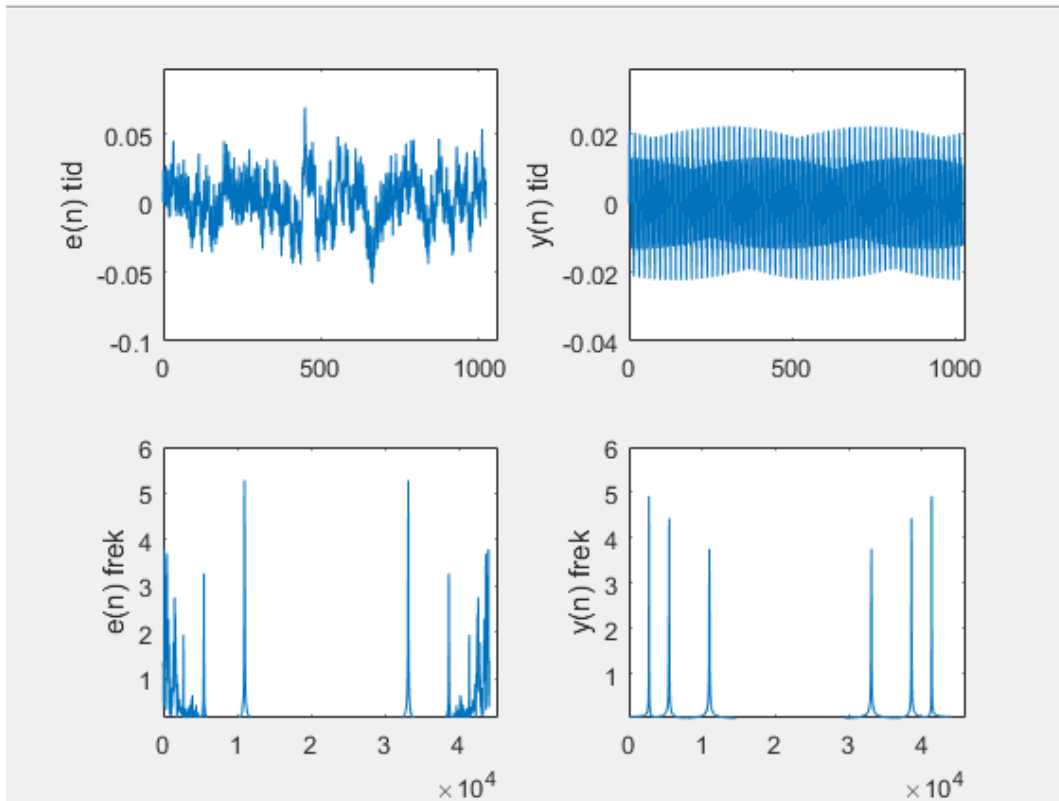
Cross-core er et IDE, som kan programmere software over på blackfin procesoren. Heri gennem er der lavet embedded software, som styre forbindelserne på blackfin, samtidig med at vi udfører NSS's funktionalitet. Igennem undervisningen er der blevet henvist til signal procesering framework, som også gøres brug af i denne opgave.

Gennem denne test var der flere ting som udfordrede funktionaliteten , og sammenligningsgrundlaget ift matlab simuleringen. Da det er nemmest at sammenligne med er grafer og figurer, blev vi nød til at udskrive txt filer fra blackfin, som udskrev hver sample. Da blackfin har et begrænset memory, kunne vi dog kun have 1024 sample gemt, hvilket er et meget kort lydsignal. Dette var især en ulempe, da matlab modellen var bygget op af lydsignaler med en længde på 268723 samples. Dette gør en signifikant forskel. Hvis vi kigger på figur 13, ser vi den realiserede proces med 3 toner, som også blev brugt i matlab modellen. Dette er valgt da det visuelt er meget nemmere at se på en graf frem for støj fra en food processor. Hvis der sammenlignes med de første 1024 samples i matlab figur 14, ser vi at matlab ikke

filtrerer sinus tonerne fra som ved realiseringen på blackfin. Det skal med at skaleringen af tidssignalerne fra blackfin og matlab, ikke er den samme, dette skyldes **fixedpoint**, og at en txt fil kun kan have integer værdier. Det har dog ingen indflydelse på resultaterne, da skaleringen er den samme. Ydermere er der også ændret i koden, så første værdi er 1 og ikke 0, mere om det emne i diskussionen.

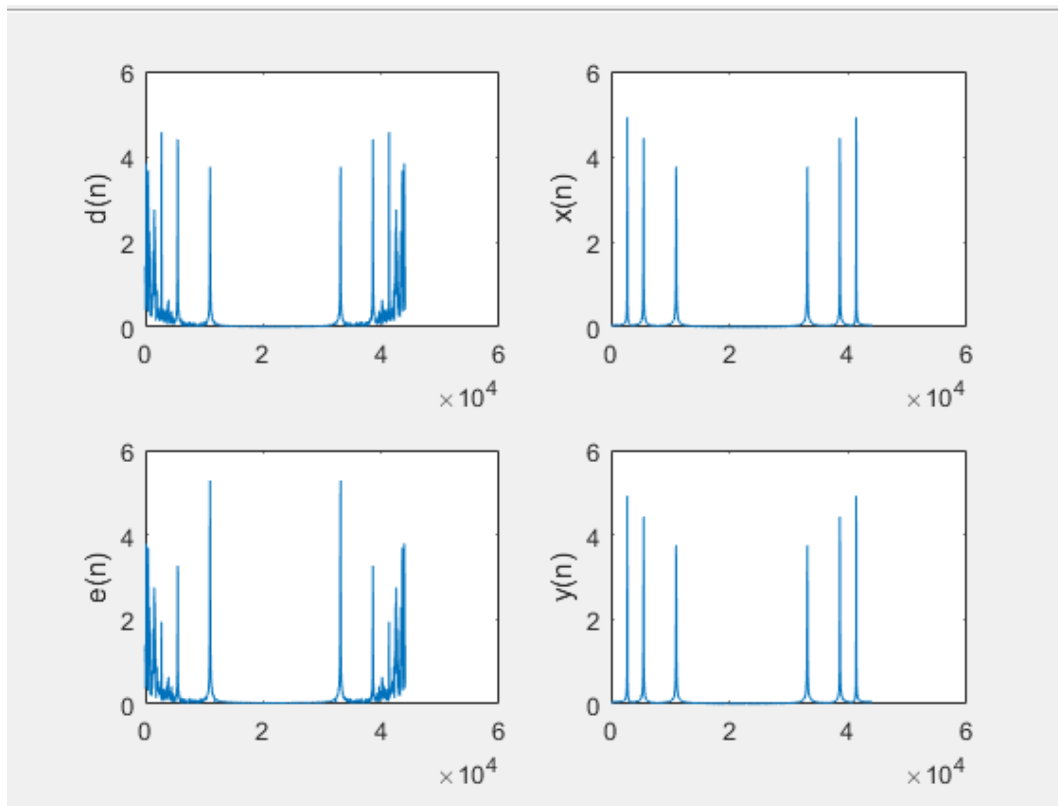


Figur 13: LMS filter implementeret på Blackfin



Figur 14: Simulering af figur 13 i matlab

Hvis vi kigger nærmere på matlab simuleringen, af de samme signal og samme antal samples, undrer gruppen sig over hvordan det kan være at matlab modellen bliver markant anderledes, og ikke ser ud til at virke efter hensigten. Dette kommer især til udtryk hvis vi kigger på figur 15, hvor vi ser at frekvensen fra $y(n)$ burde bliver trukket fra $d(n)$, det ser dog ikke ud til at virke efter hensigten. Dette emne vil kort blive taget op i diskussionen igen.



Figur 15: Fejl fra matlab filter

9.2.3 Audio Test

Gruppen har gennem testen lavet en audio test, hvor både sinus toner så vel som food processoren er blevet testet. Produktet og derved lyden virker til en vis grad igennem blackfin processoren. Hvis der sendes 3 sinus toner som både støj og implementeret i talesignalet, opleves at sinus tonerne fjernes, dog tilføres en "klik" lyd, som vil forsøges forklaret i diskussionen. Hvis vi derimod sender food processoren ind som støj og implementerer det i talesignal, bliver lyden forvrænget og filtrering af fejlen har ikke den ønskede funktionalitet ift at fjerne støjen. Dette vil igen blive diskuteret i diskussionen.



10 Diskussion

Igennem processen i projektet, mødte gruppen flere problemstillinger. Noget af det første var, at bestemme hvilket filter der skulle bruges for at løse opgaven. Efter søgen på nettet og gennem undervisningsbogen, blev et LMS filter valgt som den rette løsning. Hertil er der blevet produceret et filter, som er testet både i Matlab som simulering, og på blackfin processoren som realisering.

Igennem testen oplevede gruppen flere forskellige problemstillinger som vil redegøres for herunder.

Første sample = 1. Igennem testen blev den første sample sat til et i cross-core koden, dette gøres for at få det samme signal fra input til output. Dette betyder at fejlen allerede på første sample er stor, mens hvis første sample er 0, vil filteret lige så stille og roligt få en større fejl, og derved have en indjusteringstid. Derfor giver det bedre mening at bruge en værdi på en. Dog ser vi en at fejlen $e(n)$ forværes hvis den første værdi i matlab koden er 1 og ikke 0. Vi har ikke nogen god forklaring på hvorfor dette er tilfældet, vi mener dog at første værdi burde være en for at sende det samme signal igennem.

Filter koefficienter har stor betydning for filteret, især når vi kigger på matlab koden. For at filtrere ordentlig på de forskellige toner og food processeren, skal filteret have 256 filter koefficienter i matlab. Modsat høres væsentlig forskel på blackfin, når vi kommer over 32 koefficienter, hvor den omtalte ”klik”lyd forværes hvis vi kommer på 64 eller derover. Gruppen mener at dette skyldes at programstrukturen ændrer disse koefficienter på en gang, derfor giver det en kraftigere ”klik”lyd når vi overstiger et bestemt antal. Vi har dog ikke nogen forklaring på hvorfor matlab modellen ikke kan filtrere støjsignalerne fra ved lav filter koefficienter. Det burde være muligt at filtrere helt ned til blackfin implementeringen, især hvis man tester med 3 rene sinus toner.

Simulering af realiseringen. Da der blev simuleret det eksakte som vi realiserede på blackfin, var resultaterne meget mærkelig ref figur15, da det ikke ligner at funktionen $e(n) = d(n) - y(n)$ bliver gennemført ordentlig. Hvis man kigger på skaleringen burde filteret trække støj tonerne fra det oprindelige signal. Den bedste forklaring vi havde på dette var at der bliver brugt for lidt koefficienter, så filteret ikke lavede et filter peak på den rigtige frekvens. Dette tjekkede vi dog i matlab, og fandt at de var ens, derfor kunne den teori afvises.

Food processor realisering Da vi skulle teste om vores filter kunne filtrere på et mere komplekst signal end tre toner, har vi valgt at bruge en foodprocessor. Støjen fra foodprocessoren indeholder mange forskellige frekvenser, hvilket gør kravene til filteret højere. Da vi er begrænset til kun at have 32 koefficienter begrænser det hvor mange af disse frekvenser der kan dæmpes. Dette er en af grundene til at filteret ikke har haft den ønsket effekt. En anden grund kan også være at vores timing ikke har været god nok mellem støj signalet og tale-signalet. Da vi testede med enkelte toner fandt vi ud af at bare en lille ændring i støjsignalets frekvens gjorde filteret betydeligt dårligere til at dæmpe støjen. Så hvis timingen ikke har været præcis nok, har de forskellige frekvenser for støjsignalet ikke passet med støjen på talesignalet.

”klik”lyde. Som tidligere beskrevet opleves der ”klik”lyde når filteret køres på blackfin. Den eneste gode forklaring vi har på dette, er at filterkoefficienterne ændres så hurtigt at de ’ødelægger’ lyder med ”klik”lyde.

Bedst udenfor 300-3400 Hz. Vi fandt også ud af gennem projektet at LMS filteret fungerer bedst hvis det støjende signal ($x(n)$) og det ønskede signal ($e(n)$) ligger i forskellige frekvenser. Dette ses af figur 12, hvor vi har et tale signal, som normalt ligger imellem 300-3400 Hz, sammensat af et food processor som støjer på et meget bredt spektrum. Det ses at filteret skaber et gain af støjen som er hensigten, når vi kommer over 3400 Hz. Dette betyder også at filteret støjsignaler fra som ligger indenfor talesignalet. Dog har vi gennem projektet set at en ren sinus tone på en bestemt godt kan filtreres fra selvom den ligger inderfor tale båndet. Dette skyldes højst sandsanligt at tonen er så kraftig at filteret negligere talesignalet.



Litteratur

- [1] Gan and Kuo.
Embedded Signal Processing with the Micro Signal Architecture, Chapter 4.4.1
John Wiley 1st Ed. 2007.
- [2] Gan and Kuo.
Embedded Signal Processing with the Micro Signal Architecture, Chapter 7.2.2.1
John Wiley 1st Ed. 2007.