Audioeffekter

Datateknik

P3-projekt rapport

Gruppe 350

Aalborg Universitet

16. januar 2005

AALBORG UNIVERSITET

Institut for elektroniske systemer
Fr. Bajers Vej 7
Telefon 96 35 86 00
http://www.ies.aau.dk

Titel:

Audioeffekter

Tema:

Mikroprocessorsystemer

Projektperiode:
P3, efterårssemesteret 2004

Projektgruppe:
Gruppe 350

Deltagere:
Mikkel Yde Kjær
Gavin Francis Murray
Achuthan Paramanathan. K
Søren S. Larsen

Vejleder:
Mads G. Christensen

Oplagstal: 7

Sidetal: 82

Bilagsantal og –art: 0

Afsluttet den 20. december 2004

Synopsis:

Rapporten omhandler konstruktionen af et digitalt effektboard, baseret på et mikroprocessersystem, der opererer i realtid.
Dette omfatter hardwaredesign og konstruktion, såvel som design og implementation af software i mikroprocessoren. Derudover er der designet et interface til en klient-PC, som kan styre effekterne på boardet via seriel datakommunikation over bluetooth.
Der beskrives grundlæggende teori vedr. digital signalbehandling og de valgte audioeffekter der implementeres på boardet. Effektimplementationen omfatter en 3-bånds parametrisk equalizer. Der er ydermere beskrevet en implementation af en compressor, som ikke har været succesfuld. Testen af boardet har givet tilfredsstillende resultater, med henblik på equalizeren, både ved analyse af frekvensspektret, og i en uformel lyttetest.

Forord

Denne rapport er udarbejdet som et D3-projekt på institut for elektroniske systemer ved Aalborg Universitet, efterårssemesteret 2004. Temaet for semesterperioden er ”mikroprocessorsystemer”, med projektvalget ”audioeffekter”. Formålet med projektet er, gennem anvendelse af en mikroprocessor, samt digitale og analoge komponenter, at implementere og anvende beregningsalgoritmer til digital signalbehandling. Rapporten henvender sig til datateknikstuderende på 3. semester.

Rapporten er opbygget så nummerering af figurer, tabeller og ligninger er angivet gennem kapittelnummeret. For eksempel angiver ”figur 4.2”, at figuren er nr. 2 i kapitel 4. Bagerst i rapporten findes appendiks, der indeholde supplerende materiale og udreginnger, samt programmellets kildekode. Appendikset er angivet med bogstaver fra A - D. Kildehenvisninger i rapporten er markeret med ”[nummerering]”, der henviser til kildelisten, som er placeret bagerst i rapporten. I visse afsnit er benyttet hovedkilder, hvor disse angives ved begyndelsen af afsnittet. Figurernes kildehenvisninger er angivet som ”fra/efter [nummerering]”, afhængig af om figuren er kopieret fra en kilde, eller om den er tegnet efter kilden.

Indhold

Kapitel 1

Indledning

Digital signalbehandling kan spores tilbage til 1960-70'erne, da digitale computere begyndte at bliver mere tilgængelige [1]. På dette tidspunkt var teknologien forbeholdt statslige affærer og opgaver at stor økonomisk betydning, som rum- og medicinalforskning. Med introduktionen af mikroprocessoren i 1971 og den følgende udbredelse af privatcomputeren i 80'erne, voksede anvendelsesmulighederne af digital signalbehandling [2]. Produkter som CD-afspillere og mobiltelefoner gjorde sin indtræden på markedet, og i dag benytter størstedelen af kommercielle eletroniske produkter digital signalbehandling. I dag foregår lydbehandling i høj grad digitalt, da det reducerer omkostningerne og giver flere muligheder indenfor feltet. Ved audioeffekt forståes der, at et audiosignal påføres en modifikation for at opnå en ønsket virkning. Et eksmepel på en effekt er rumklang, som kan påføres et stykke musik, for at give opfattelsen at det spilles i et lille rum eller en katedral.
I dette projekt indgår digital signalbehandling, til implementation af audioeffekter i et realtidssystem. Effekterne påføres i et mikroprocessorsystem, hvor et audiosignal konverteres fra en analog spænding til digitale værdier. Det kvantificerede signal behandles i mikroprocessoren med udvalgte effekter, hvorefter det digitaliserede signal konverteres til et analogt signal igen.

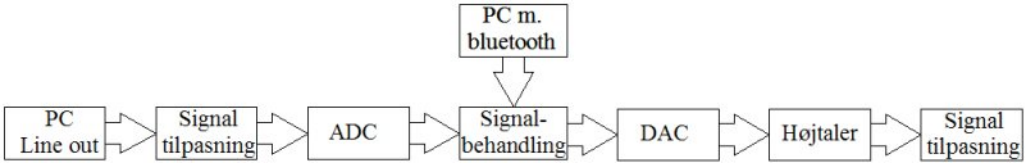
Kapitel 2

Systemoversigt

For at opnå en bedre forståelse for de funktioner, systemet skal være i stand til at udføre, beskrives det først på et oversigtsmæssigt niveau. Beskrivelsen udfærdiges til en kravspecifikation, som danner rammen om systemt. Ved benævnelsen systemet, menes der effektboardet som en helhed.

2.1 Systembeskrivelse

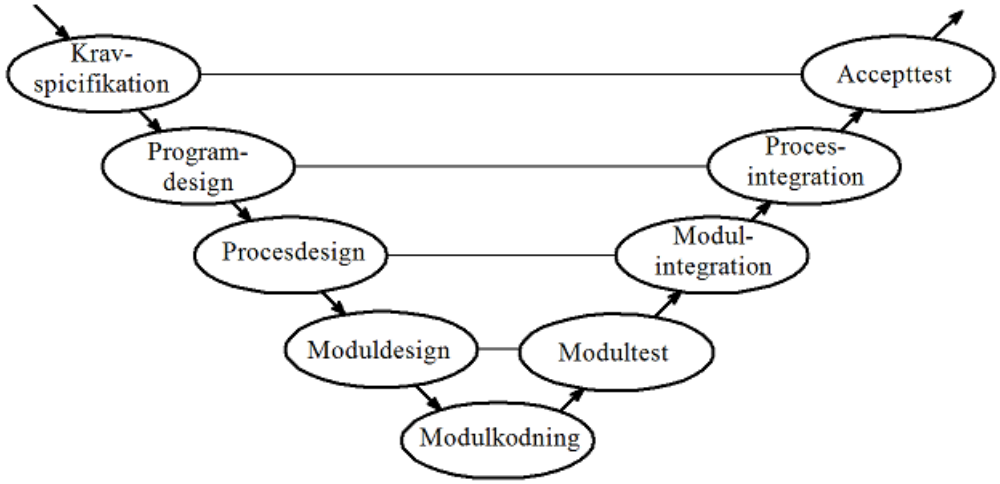
Systemet modtager et audiosignal fra en perifær enhed, som line-out fra en PC. Dette signal kan være musik, tale fra en mikrofon tilsluttet PC'en eller lignende. Oversigten er vist på figur 2.1.
Inputsignalet tilpasses, så det kan digitaliseres i en analog til digital converter (ADC), hvorefter det behandles i mikroprocessoren, der tager sig af audioeffekterne. Denne behandling kan være i form af en equalizer, der kan tilpasse lydens bas, mellemtone og diskant. Ydermere vil der være mulighed for at påvirke signalets dynamisk rækkevidde med en compressor.
Disse effekter skal være konfigurerbare fra en klient-PC, der afvikles på en Win32 kompatibel computer. Kommunikation mellem klient-PC'en og audiosystemet sker via RS-232 protokollen over et bluetoothmodul, forbundet til mikroprocessoren og en USB bluetooth dongle sat til computeren.
Efter signalet er blevet påført en effekt, føres signalet tilbage til det analoge domæne, via en digital til analog converter (DAC), og ud gennem en højtaler.



Figur 2.1: Blokdiagram der illustrerer systemet, som det kommer til at fungere.

De specifikke krav til grænsefladerne, herunder eksterne grænseflader, valg af ADC/DAC og forbindelsen mellem disse og mikroprocessoren, afdækkes i kravspecifikationen.

2.2 SPU



Figur 2.2: SPU's V-model, der viser de trin, som gennemgås under udvikling af software/systemer [3].

SPU-modellen er en metode til at strukturere projektforløbet med. Modellen indeholder en række værktøjer og teknikker, der kan være nyttige at anvende i et projekt, hvor der udvikles software. I dette projekt anvendes V-modellen fra SPU til strukturering samt udvikling af software og hardware. Formålet med anvendelse af SPU og V-modellen er:

- At opnå en god og overskuelig struktur af systemet.

- At finde fejl i systemet så tidligt som muligt.

V-modellen er vist i figur 2.2, og er opbygget på en sådan måde, at de trin der ligger på niveau med hinanden, illustreret med en vandret linje, skal udarbejdes sideløbende. De punkter som ligger til venstre for V-modellen er udviklingsfasen, og til hvert trin i udviklingsfasen, udfærdiges der en tests.

2.3 Kravspecifikation

Kravspecifikationen er stillet med udgangspunkt i SPU-modellen, og er anvendt, i det omfang der findes relevant.

1. Hardware.
   Effektboardet skal omfatte følgende enheder:
   - Texas Instruments MSP430F149 mikroprocessor.
   - Ekstern DAC da mikroprocessoren kun har en 12-bit ADC.
   - Bluecom Bluetooth modul samt dongle til kommunikation ml. klient-PC og mikroprocessor.
   - Single-supply spændingskilde på 300mA 9V DC-spænding.
   - 3,5 mm mono hun jackstik til input/output
2. Specifikke krav til effektboardet.
   Effektboardet skal kunne påføre lyden følgende audioeffekter:
   - En parametrisk equalizer.
   - Signalets dynamiske rækkevidde skal kunne påvirkes med en compressor.
3. Brugergrænseflader.
   Ved anvendelse af systemet skal der være mulighed for at justere effekternes parametre via software på klient-PC'en. Programmet skal omfatte:
   - Ændring af effektparametre.
   - Mulighed for valg/fravalg af effekter.
   - Mulighed for af oprette og afbryde kommunikationen over bluetooth.

- Et dialogvindue til statusbeskeder.

4. Systemets ydelse.
   Systemet skal operere i realtid, hvilket her betyder det at systemet skal nå at bearbejde en sample og outputte denne, inden der modtages en ny. For bedst mulig at kunne gengive musik og taget mikroprocessorens regnekraft i betragtning gælder:

   - En samplerate på 22,05 kHz.

2.4 Accepttestspecifikation

Her beskrives de tests der skal udføres for at undersøge hvorvidt kravspecifikationen er opfyldt. De konkrete krav til hardwaregrænsefladerne kan findes umiddelbart, uden nogle specifikke tests. Den endelige accepttest består af to tests der supplerer hinanden i en sådan grad at yderligere tests ikke anses som værende nødvendige.

- Der foretages en analyse af de forskellige effekters frekvensrespons, som stilles op mod det umodificerede signal. Denne mere specifikke test, afslører fejl i effekterne, eller i systemet, på et niveau der kan fortælle mere om hvor fejlen befinder sig. Frekvensresponsen kan imidlertid være i orden, mens lyden er af lav kvalitet.

- Der foretages en uformel lyttetest af effektboardets output, hvor en subjektiv vurdering af lydens kvalitet gives. Denne test skal afsløre fejl i systemet, og i effekterne. Den kan dog kun på et overfladisk niveau, fortælle noget om fejlens karakter og placering.

Kapitel 3

Digital signalbehandling

I dette afsnit beskrives den grundlæggende teori bag digital signalbehandling. Formålet med afsnittet er at give et indblik i nogle af de metoder, som anvendes til at digitalisere et signal, herunder sampling, kvantisering og rekonstruktion. Nedenstående afsnit er baseret på hovedkilde [4].

3.1 Digital signalbehandling

Et analogt signal er udtrykt som en kontinuert funktion af tiden x(t). Signalet kan analyseres i frekvensdomænet, vha. fouriertransformen af x(t). Funktionen X(Ω) beskriver signalet i frekvensdomænet, og kan udtrykkes på følgende måde:

$$X(\Omega) = \int_{-\infty}^{\infty} x(t) e^{-j\Omega t} dt. \tag{3.1}$$

Denne metode kan anvendes til at beregne frekvensresponsfunktionen $H(\Omega)$, og dermed beregne outputsignalet $Y(\Omega)$, ved følgende formel:

$$Y(\Omega) = H(\Omega) \cdot X(\Omega), \tag{3.2}$$

hvor Ω udtrykker den fysiske frekvens i enheden $[\mathrm{Radian/sek}]$, dvs. $\Omega = 2\pi f$.

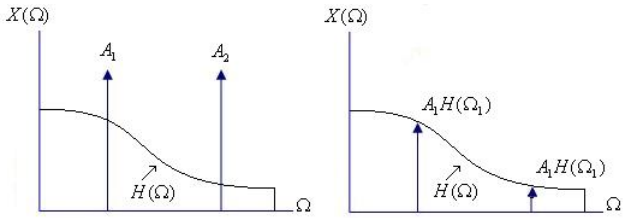
Et lineært superpositionsprincippet anvendt på to sinusfunktioner med frekvenserne på hhv. (Ω_1) og (Ω_2) og de tilhørende amplituder (A_1) og (A_2), kan udtrykkes som:

$$x(t) = A_1 e^{j\Omega_1 t} + A_2 e^{j\Omega_2 t}.$$

Ved anvendelse af formel 3.2 fås den filtrede funktion til:

$$y(t) = A_1 H(\Omega_1) e^{j\Omega_1 t} + A_2 H(\Omega_2) e^{j\Omega_2 t}. \tag{3.3}$$

Grafen for funktionen y(t) kan ses på figur 3.1.



Figur 3.1: Illustrerer formel 3.2, her kan det ses hvordan overføringsfunktionen H(Ω) påvirker signalet x(t) (efter [4]).

Af figur 3.1 og funktionen 3.3 fremgår det, at overføringsfunktionen H(Ω) kan styre den pågældende sinusfunktion x(t) mht. fase og amplitude. Det vil sige, et input af en sinusfunktion med frekvensen Ω, blive påvirket af faktoren $|H(\Omega)|$ mht. amplituden, og en ændring i fasen med en størrelse der svare til $argH(\Omega)$.
Dette kan udtrykkes som:

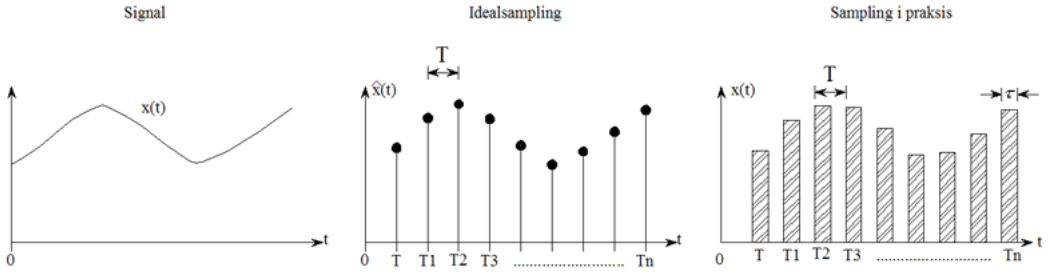
$$x(t) = e^{j\Omega t} \Rightarrow\ y(t) = H(\Omega)e^{j\Omega t} = |H(\Omega)|e^{j\Omega t + jargH(\Omega)}. \tag{3.4}$$

Dette princip kan bruges til at undgå aliasing under samplingen i A/D konverteringen. For eksampel, hvis komponenten Ω_1 er ønsket i det pågældende signal og Ω_2 ønskes at blive fjernet, kræve det en filtret der er designet på følgende måde: $H(\Omega_1)$ = 1 og $H(\Omega_2)$ = 0.

3.2 Sampling

Når et kontinuert signal x(t) samples, overføres det til det diskrete tidsdomæne x$[n]$. Det analoge signal samples til tidsintervallet T, som er målt i sekunder hvor t = nT,

og n = 0,1,2.... Dette kaldes samplingstiden f_s, og er givet ved: ($f_s = \frac{1}{T}$). Sampling kan udtrykkes på to måder, ideelt og i praksis se figur 3.2.



Figur 3.2: Illustrerer et periodisk sampling, ideelt og i praksis (efter [4]).

Formler for de to samplingsbegreber udtrykkes på følgende måde:

- Idealsampling

$$\sum_{n=-\infty}^{\infty} x(nT)\delta(t-nT). \tag{3.5}$$

- Sampling i praksis

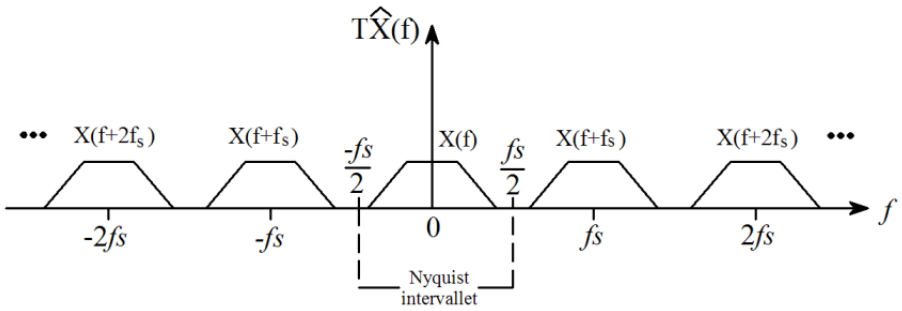
$$\sum_{n=-\infty}^{\infty} x(nT)p(t-nT). \tag{3.6}$$

Fra de to ovennævnte formler samt figur 3.2 fremgår det, at den ideele sampling forgår øjeblikkeligt, hvorimod sampling i praksis skal holde signalet i en kort periode, kaldet sample/hold. For funktionen p(t) med $\tau \rightarrow 0$ vil funktionen tilnærme den ideele sampling. Den samplede sinusfunktion har et periodisk spektrum af det oprindelige signal med samplingsintervallet f_s, kaldet spektrumreplikationer og er udtrykt ved: (f´= f + mf_s, m = $0, \pm 1, \pm 2, ...$) Se figur 3.3.

Aliasing

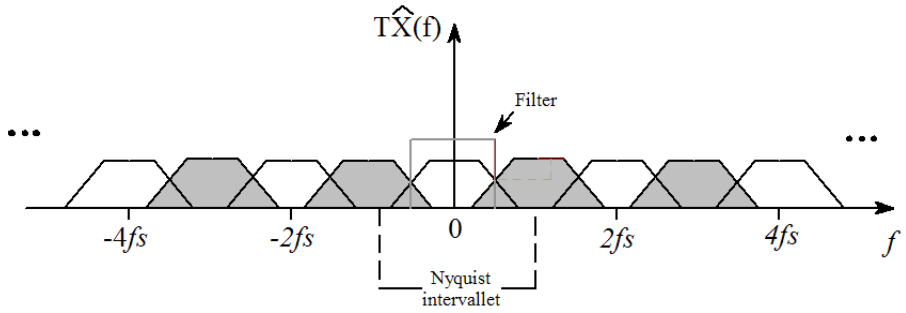
Et signal x(t) kan samples, således at outputtet y(t) ikke længere kan rekonstrueres dette kaldes aliasing. For at minimere aliasing, skal følgende kriterier være opfyldt:

1. Signalet x(t) skal være båndbegrænset, dvs. den maksimale frekvens (f_{max}) skal være kendt.
2. Samplingstiden f_s skal vælges til at være større end ($2 \cdot f_{max}$).



Figur 3.3: Illustrerer spectrumreplikationen som resultat af sampling (efter [4]).

Ud fra ovennævnte kriterier kan samplingsintervallet formuleres på følgende måde: $\left[-\frac{f_s}{2}, \frac{f_s}{2}\right]$, også kaldet nyquistintervallet. Nyqvistfrekvensen $\frac{f_s}{2}$ også kaldet cutoff-frekvensen bruges i filtre, eksempelvis i et analogt præfilter for at formindske aliasing. På figur 3.4, illustreres en spektrumreplikation af et ændret/aliaseret signal. Der fremgår, at aliasing-effekten fremtræder idet (f_{max}) bliver større end $\frac{f_s}{2}$ dvs. spektrumreplikationen falder over nyquist-intervallet. I det tilfælde bruges der et filter med overføringsfunktionen $H(\Omega)$ som filtrerer de signaler som falder over nyquist intervallet fra.

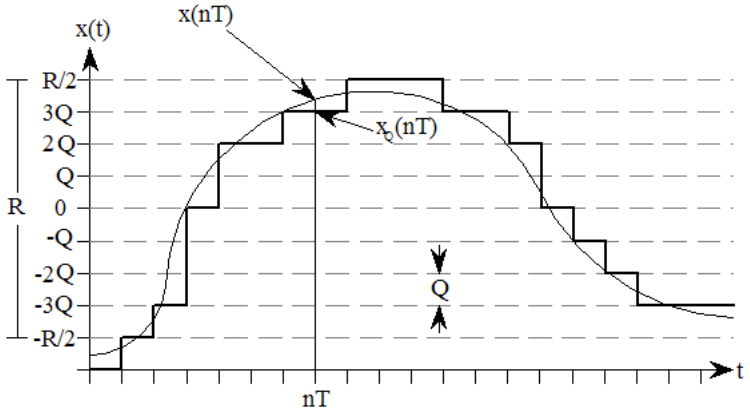


Figur 3.4: Illustrerer et ændret signal (aliasing), der opstår som følge af (f_{max} > nyquistintervallet), Et filter skærer de signaler som falder over nyquist intervallet fra, så aliasing bliver formindsket (efter [4]).

3.3 Kvantisering

I afsnittet ”Sampling” blev det antaget, at det var muligt at sample et signal med uendelig præcision. Dette er af årsager som hardwarebegrænsninger, ikke muligt at udføre i praksis. Derfor behøves en proces til at oversætte de samples til endelig

præcision, denne proces kaldes kvantisering. Anvendes der en AD-converter med en præcision på 12 bit, opløses signalets amplitude i 4096 trin. Kvantiseringsopløsningen kan udtrykkes som $Q = \frac{R}{2^B}$[1] se figur 3.5.
Figur 3.5 illustrerer sampling og kvantisering af en analog sinusfunktion x(t). Den horisontale linje indikerer kvantiseringsniveauet, og den vertikale angiver samplingstiden nT. Som det fremgår af figuren, kvantiseres signalet til nærmeste kvantiser-



Figur 3.5: Sampling samt kvantisering af en sinusfunktion (efter [5]).

ingsniveau, hvilket kaldes *rounding*, og medfører et tab af præcision. Det fremgår at præcisionstabet formindskes, hvis bit-niveauet hæves.
Tabet af præcision kan beskrives som forskellen mellem den kvantiserede værdi $x_q(nT)$ og samplet x(nT), kaldes for kvantiseringsfejl, og kan udtrykkes på følgende måde [4].

$$e(nT) = x_q(nT) - x(nT). \tag{3.7}$$

Det skal bemærkes, at den maksimale og den minimale værdi e kan antage er:

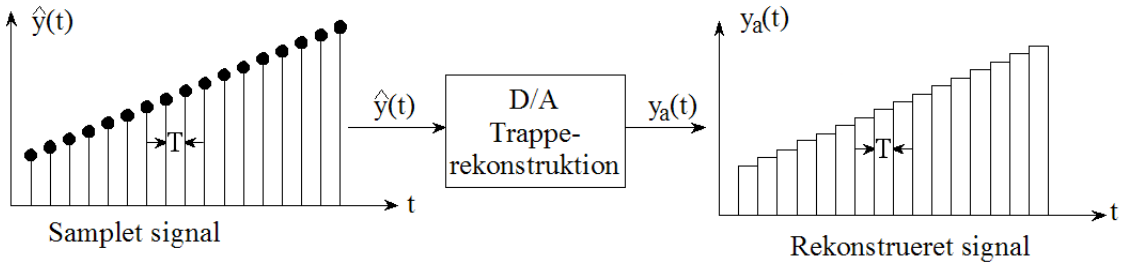
$$-\frac{Q}{2} \leq e \leq \frac{Q}{2}. \tag{3.8}$$

3.4 Rekonstruktion

Ved analog rekonstruktion konverteres de digitale samples til et analogt signal. En ideel rekonstruktion er et ideelt lowpass-filter med en knækfrekvens der svarer til nyquistfrekvensen $\frac{f_s}{2}$. Ved sampling opstår der nogle høje frekvenser, disse

[1] R er peak to peak værdien af inputsignalet, B er ADC bitopløsning.

frekvenser udglattes under rekonstruktionen. Figur 3.6 illustrerer et rekonstruktionsforløb. Den mest anvendte form for rekonstruktion er trappe-rekonstruktion.



Figur 3.6: Rekonstruktionsforløbet, hvor et samplet signal gennemgår en trappe-rekonstruktion i DAC (efter [4]).

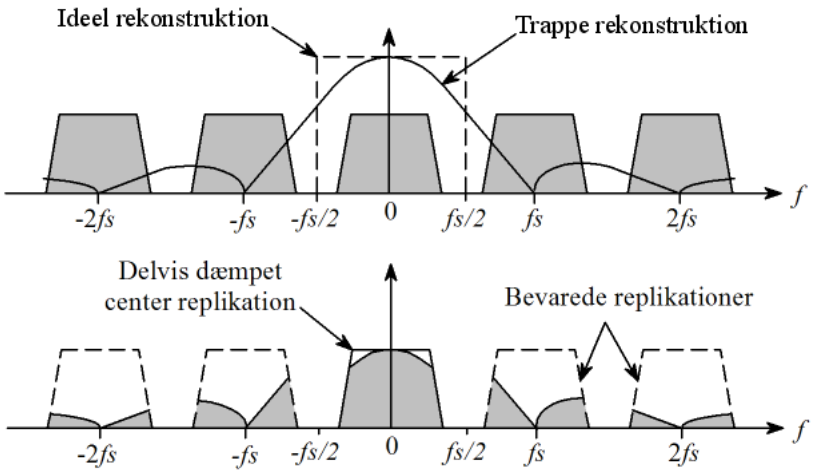
Her tilstræbes der en trappeapproksimering til det oprindelige signal, se figur 3.6. Denne operation kan sammenlignes med sampling i praksis, hvor trappe-rekonstruktionens impulsrespons h(t) har den samme funktionalitet som sampling pulset p(t). Trappe-rekonstruktionens impulsrespons h(t) er udtrykt på følgende måde:

$$h(t) = u(t) - u(t-T) = \begin{cases} 1, & \text{hvis } 0 \leq t \leq T \\ 0, & \text{ellers.} \end{cases} \tag{3.9}$$

Ved Laplacetransformationen af h(t) fås frekvensresponset til:

$$H(s) = \frac{1}{s} - \frac{1}{s}e^{-sT}.$$

Af figur 3.7 ses det, at ikke alle replikationer bliver fjernet af en trappe- rekonstruktion. Af denne grund anvendes endnu en rekonstruktion kaldet anti-image postfilter. Et anti-image postfilter fjerner slutligt de resterende replikationer vha. endnu et lowpass-filter, og dermed bliver den ønskede replikation mere udglattet [4].



Figur 3.7: Frekvensrespons af et ideelt- samt trapperekonstruktion (efter [4]).

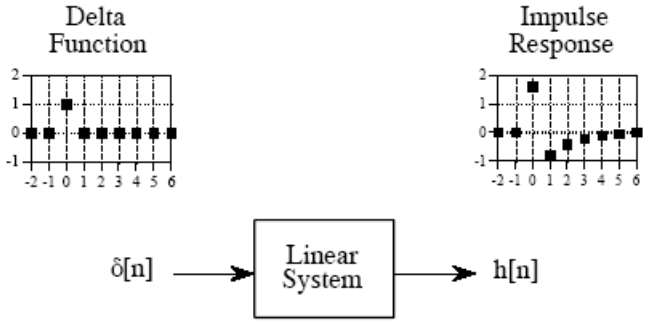
Kapitel 4

Digitale filtre og audioeffekter

Her forklares den grundlæggende teori bag digital filtrering, som er anvendt til design og implementation af effekterne i dette projekt. Dette afsnit er baseret hovedkilde [4].

4.1 Impulsrespons

Et digitalt filter kan karakteriseres ved dets impulsrespons $h[n]$, hvilket er outputtet når inputtet er kronecker delta $\delta[n]$. Signalet er karateriseret ved, at den til $n = 0$ har amplituden 1 og til andre værdier for n er lig 0. Dette er vist på figur 4.1.



Figur 4.1: Enhedsimpulsen føres ind i et filter, og outputtet er impulsresponsen som karateriserer filteret (fra [1]).

4.2 Infinite impuls response filtre

Filtertypen infinite impuls response (IIR) er kendetegnet ved anvender feedback til bestemmelse af output. Det betyder at den bruger både tidligere input og output-samples til at bestemme nyt output. Feedback egenskaben medfører at signalet kan svinge i det uendelige selvom der kun inputtes nul værdier, hvilket giver en uendelig frekvensrespons og heraf navnet. Differensligningen for et vilkårligt 2.ordens IIR-filter kunne se således ud:

$$y[n] = b_0x[n] + b_1x[n-1] + b_2x[n-2] + a_0 + a_1y[n-1] + a_1y[n-2]. \quad (4.1)$$

Dog er dette ikke altid tilfældet, da impulsrespons kan dø ud efter nogen tid (dens størelse orden bliver mindre end en bit), hvorefter den kan anses for endelig.

4.3 Z-transformation

Z-transformation kan betegnes som den digitale ækvivalent til Laplace-transformationen. For et diskret signal $x(n)$ er z-transformationen givet ved:

$$X(z) \; = \; \sum_{n=-\infty}^{\infty} x(n)\, z^{-n}. \quad (4.2)$$

$$z \; = \mathrm{e}^{\jmath\,\omega}. \quad (4.3)$$

Z-planet er repræsenteret i polære koordinater som ses af formel 4.3. Punkter indenfor enhedscirkelen i z-planet svarer til s-planets venstre halvdel og forholdet mellem z- og s-planet kan beskrives ved at den vertikale akse $Im\, s$ "bøjes"omkring enhedscirkelen. Vinkel ω spænder fra π til $-\pi$ radianer. De positive frekvenser er repræsenteret fra 0 til π og de komplekskonjugerede fra 0 til $-\pi$.

4.4 Bilineær transformation

Den bilineære transformation er en designmetode til digitale IIR filtre. Med udgangspunkt i et analogt filter er det muligt at designe et tilsvarende digitalt filter, hvilket betyder,

at alt filtrering der kan laves analogt, også kan laves digitalt.

Transformationen fra s- til z-domænet er givet ved:

$$s = f(z) = \frac{1 - z^{-1}}{1 + z^{-1}}. \tag{4.4}$$

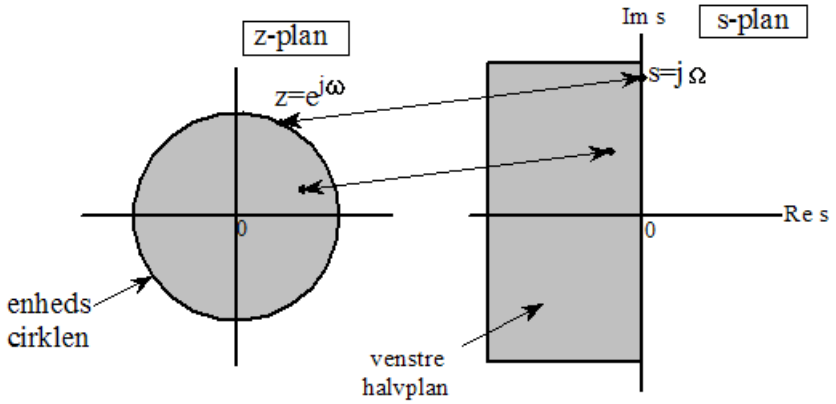
Kortlægning mellem den analoge frekvens og den digitale frekvens fåes ved indsættelse af følgende udtryk i 4.4:

$$s = j\Omega. \tag{4.5}$$

$$z = \mathrm{e}^{j\omega}. \tag{4.6}$$

$$j\Omega = f(\mathrm{e}^{j\omega}) = \frac{1 - \mathrm{e}^{-j\omega}}{1 + \mathrm{e}^{-j\omega}} = \frac{\mathrm{e}^{j\omega/2} - \mathrm{e}^{-j\omega/2}}{\mathrm{e}^{j\omega/2} + \mathrm{e}^{-j\omega/2}} = j\frac{\sin(\omega/2)}{\cos(\omega/2)} = j\tan\left(\frac{\omega}{2}\right). \tag{4.7}$$

Ved denne metode kan punkter i s-planets rektangulære koordinater, bestående af $s = \sigma + j\Omega$, mappes til z-planets polære koordinater, givet ved $z = e^{j\omega}$, og omvendt.



Figur 4.2: Punkter fra s-planets venstre halvplan mappes til det indre af enhedscirkelen og den imaginære akse til cirklens bue (efter [4]).

Paralleller mellem egenskaberne omkring stabilitet og kausalitet kan vises ved følgende formel, hvor den reelle akse i s-planet undersøges i forhold til z-planet. Ovenstående figur viser at venstre side af s-planet afbildes indenfor enhedscirklen i z-planet og at den imaginære akse $Im\ s$ plottes på enhedcirklens bue.

$$Re\ s = \frac{|z|^2 - 1}{|z + 1|^2} \Updownarrow \tag{4.8}$$

$$Re\ s < 0\ \ |z| < 1 \ \ \Leftrightarrow \ \ Re\ s = 0\ \ |z| = 1.$$

4.5 Equalizer

En equalizer kendetegnes ved at kunne dæmpe eller hæve forskellige frekvensområder i et signal. Hvis bestemte frekvensområder er svagt repræsenteret ved afspilling af musik, grundet et rums akustik, kan der anvendes en equalizer til at fremhæve de dæmpede dele af signalet. Herved opnåes en flad frekvensrespons [6].

Tone controls

Equalizereffekten kendes bedst fra musikanlæg med drejeknapperne bass og treble. Disse kaldes hhv. low- og highpass shelvingfiltre og giver mulighed for at hæve og sænke amplituden i de lave og høje frekvensområder, med udgangspunkt i en fastlagt cutoff-frekvens. Et lowpass shelvingfilter er forskellig fra et lowpass filter, da shelvingfiltret forstærker eller dæmper et frekvensområde, hvor lowpass filteret afskærer en del af det højfrekvente område.

Grafisk equalizer

En anden type equalizer er grafiske equalizere der virker ved at inddele frekvensspektret ind i et fast antal bånd, som hvert styres af båndpasfiltre der kan justeres på gainet. En 3-bånds grafisk equalizer vil være opbygget af tre båndpasfiltre som hver styrer en del af frekvensspektret, ud fra en fastlagt centerfrekvens. Disse filtre parallelforbindes da de hver isolerer et bestemt frekvensområde.

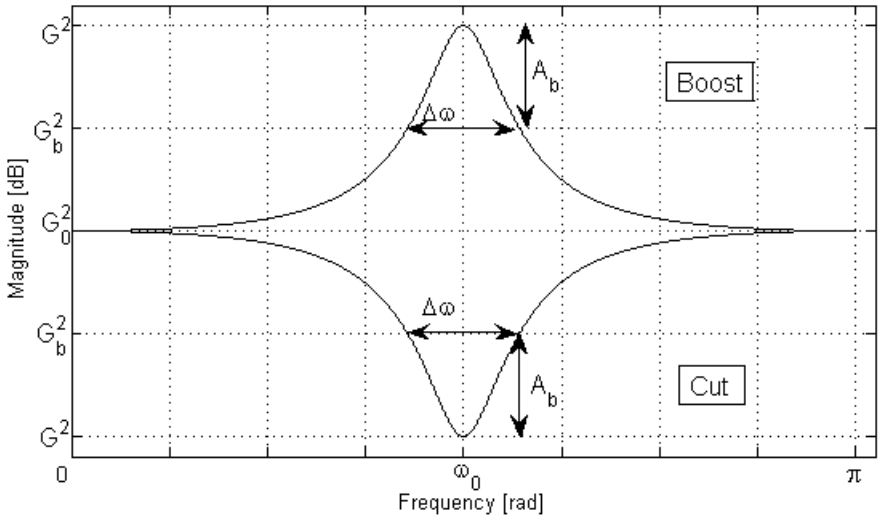
Parametrisk equalizer

En parametrisk equalizer giver flere muligheder for finindstilling. Baseret på et shelvingfilter kan der stilles på gainet, centerfrekvens, båndbredde og båndbreddens cutoff-frekvens. En parametrisk equalizer med boost kaldes også et peaking filter, og cut giver et notch filter. Med de større indstillingsmuligheder kan færre parametriske equalizers opnå den samme effekt som grafiske equalizere, der er bundet af en bestemt centerfrekvens og båndbredde. Dette giver den parametriske equalizer en fordel implementationsmæssigt og valget er derfor faldet på type.

4.5.1 Parametrisk equalizer

Den parametriske equalizer er givet ved en linearkombination af et peak og et notch filter. Filterparameter der kan stilles på omfatter centerfrekvensen w_0, båndbredden Δw, referencegain G_0, båndbreddens cutoff-niveau G_B og gain G. Filterets overføringsfunktion er givet ved:

$$\begin{aligned} H(z) &= H(s)_{\text{peak}} + H(s)_{\text{notch}}\big|_{s=\frac{1-z^{-1}}{1+z^{-1}}} \\ &= \frac{\frac{G_0+G\beta}{1+\beta} - 2\frac{G_0\cos(w_0)}{1+\beta}z^{-1} + \frac{G_0-G\beta}{1+\beta}z^{-2}}{1 - 2\frac{\cos(w_0)}{1+\beta}z^{-1} + \frac{1-\beta}{1+\beta}z^{-2}}. \end{aligned} \tag{4.9}$$



Figur 4.3: En parametrisk equalizer med boost og cut, hvor der for boost gælder at $G^2 > G_b^2 > G_0^2$ og cut $G^2 < G_b^2 < G_0^2$ (efter[4]).

Ifølge appendiks C sker der ved G_b en halvering af signalstyrken, ved en 3 dB dæmpning. For $G_0 = 1$ vil der være 0 dB og her er signalet uændret.

Low- og highpass shelvingfilter

Ved design af equalizeren skal der bruges low- og highpass shelvingfiltre for at kunne justere frekvenserne i yderpunkterne af frekvensspektret. Disse to filtre udledes relativt simplet fra den parametriske equalizer. For lowpass shelvingfiltret

sættes $w_0 = 0$.

Highpass shelvingfiltret sættes $w_0 = \pi$

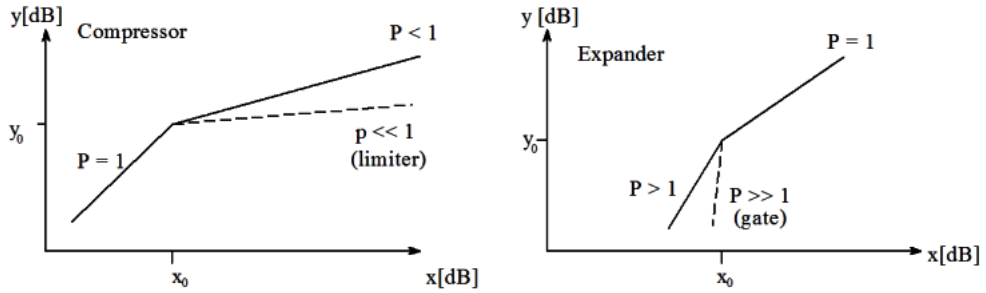
På kommercielle produkter inddeles centerfrekvensen ofte i spring på $\frac{1}{3}$ oktav. Oktavspring sker ved en fordobling af frekvensen for en oktav højere, og en halvering af frekvensen for en oktav lavere.

4.6 Compressor / expander

Der implementeres kun en compressor effekt på mikroprocessoren, men her gives en teoretisk beskrivelse af både compressors og expanders, da der er en klar sammenhæng mellem de to. Effekter som compressors / limiters og expanders / gates påvirker den dynamiske rækkevidde i et signal. En compressor har til formål, at dæmpe signaler med høj amplitude, som dæmpning af uønskede lyd, der kan opstå ved lydindspilning. Expanderen dæmper de lave amplituder i et signal, hvorved den høje lyd bliver mere markant [4].
Input- outputrelationen er vist på figur 4.4, hvor den horisontale akse angiver outputsignalet, og den vertikale akse inputsignalet. Denne relation kan udtrykkes i dB på følgende måde:

$$20\log_{10}\left(\frac{y}{y_0}\right) = \rho 20\log_{10}\left(\frac{x}{x_0}\right), \tag{4.10}$$

grænseværdien x_0 angiver amplitudegrænsen for, hvornår den enkelte effekt påvirker inputsignalet. Ved en expander effekt, vil signalet ikke have nogen ændring i amplituden så længe inputkonstanten $x \geq x_0$, og omvendt for compressoren $x \leq x_0$.



Figur 4.4: Input- outputrelation for hhv. en compressor og expander (efter [4]).

Af figur 4.4 fremgår det, at ved ændring af inputamplituden, sker der en ændring

i outputtet mht. ρ. Værdien ρ indikerer effektforholdet mht. dæmpning af signaletsamplitude. Et eksampel på effektforholdet for en expander med $\rho = 4$ vil dæmpningen af lave amplituder være 4:1, altså et 3 dB fald i inputtet vil resultere i et 12 dB fald i outputtet. Limiters og gates er ekstreme tilfælde for hvor $\rho << 1$ for compressor og $\rho >> 1$ for expander. I dette tilfælde vil input lyden blive dæmpet i en høj grad, eller bliver helt fjernet.

Niveaudetektoren

Et realiseringsdiagram for en compressor / expander er vist på figur 4.5, hvor niveaudetektoren c_n kontrollerer gainet G_n. Dette vil sige, at hvis gain på et givet signal overskrider den fastsatte grænseværdi, vil gainprocessoren dæmpe signalets amplitude i forhold til ρ. Niveaudetektoren afhænger af effektens formål, som kan bestå af :

1. Den øjeblikkelige peak værdi $|x_n|$.

2. RMS værdien af x_n.

Niveaudetektoren c_n kan udtrykkes på følgende måde:

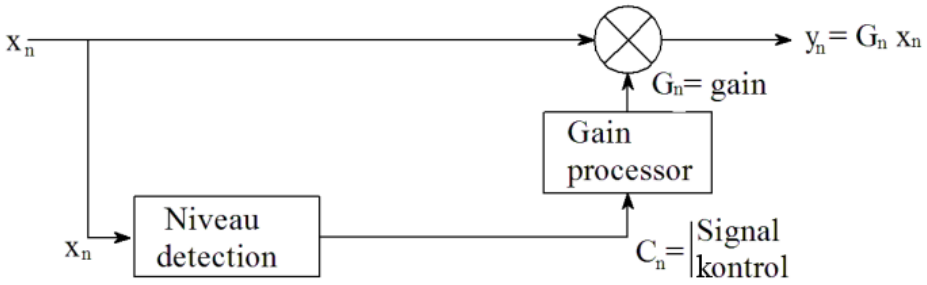
$$c_n = \lambda c_{n-1} + (1 - \lambda)|x_n| \ , \tag{4.11}$$

hvor (λ) udtrykker reaktionstiden for niveaudetektoren, hvilket angiver den tid niveaudetektoren bruger, til at justere attack- og releasetid. Denne metode anvendes på baggrund af udjævning af lydens ændring, se figur 4.6. I tilfældet hvor $(\lambda = 0)$ sker der en øjeblikkelig ændring i niveaudetektoren, og ligningen 4.11 vil være af typen peakdetektor.

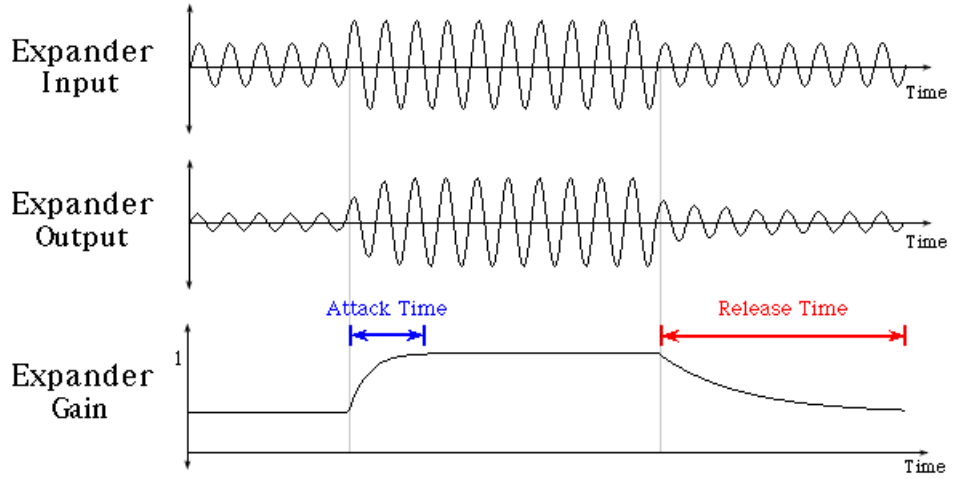
Gainprocessor

En gainprocessor kan udtrykkes som en ulineær funktion i forhold til den lineære funktion 4.10. Gainprocessor funkgionen kan udtrykkes på følgende måde :

$$Compressor: \quad f(c) = \begin{cases} (c/c_0)^{\rho-1}, & \text{hvis } c \geq c_0 \\ 1, & \text{hvis } c \leq c_0. \end{cases} \tag{4.12}$$



Figur 4.5: Filterrealiseringsdiagram af en compressor og expander (efter [4]).



Figur 4.6: Et signal modificeres af en expander hvor det dynamiske span udviddes. Høje områder af signalet dæmpes også under attacktime hvor gainet er stigende, og under releasetime dæmpes de lave områder ikke så markant (fra [6]).

$$Expander: \quad f(c) = \begin{cases} 1, & \text{hvis } c \geq c_0 \\ (c/c_0)^{\rho-1}, & \text{hvis } c \leq c_0. \end{cases} \tag{4.13}$$

hvor c_0 er den ønskede grænseværdi og $\rho < 1$ for en compressor og $\rho > 1$ for en expander. Outputtet udtrykkes ved :

$$G_n = f(c_n). \tag{4.14}$$

$$y_n = G_n x_n. \tag{4.15}$$

Kapitel 5

Hardwaredesign

I dette kapitel beskrives hardwaren, der skal integreres på effekboardet. Hardwaren udvælges med henblik på en løsning der belaster mikroprocessoren mindst muligt, derfor indledes med en undersøgelse af mikroprocessorens cycles til rådighed og lagerkapacitet. Hertil udvælges en passende DAC og slutteligt beskrives den endelige integration af hardwaren.

5.1 Mikroprocessoren

Kernen i det digitale effektboard er mikroprocessoren. Der arbejdes med en 16-bit RISC[1]-CPU som kan operere med en clockfrekvens på op til $\approx$ 8 MHz. I det lyden behandles i realtid er cycles til rådighed pr. samplingsinterval et altafgørende aspekt, da eventuelle fejl og forsinkelser vil være hørbare.
En anden vigtig faktor er boardets hukommelse, hvor der er 60 kB ROM som bruges til afviklingskoden og anden statisk data. Desuden er der 2 kB RAM til rådighed til lagring af de dynamiske data [7].

5.1.1 Cycles og interrupts

Der skal minumum læses fra ADC'en og skrives til DAC'en i én interrupt-rutine, som kaldes hver gang der samples. Da processoren opererer med en "system main clock" på 7,3728 MHz og der samples ved en hastighed på 22,05 kHz, er antal af

[1] "Reduced Instruction Set Computer"

cycles mellem 2 interrupts givet ved:

$$\text{cycles pr. interrupt} = \frac{\text{system main clock}}{\text{samplerate}} = \frac{7,3728\text{MHz}}{22,05\text{kHz}} \approx 334\,\text{cycles}. \quad (5.1)$$

5.2 ADC/DAC

ADC'ens opgave er at konvertere det kontinuerte signal fra line-in til et diskret signal, som Mikroprocessoren kan behandle.

5.2.1 Specifikke krav

Udover kravene opstillet i kravspecifikationen, er der nogle specifikke krav, som ADC/DAC komponenterne skal opfylde for at kunne anvendes i systemet.

1. Der opereres ved mindst 12 bit.

2. Det skal være muligt at synkronisere ADC og DAC med hinanden og mikroprocessoren.

Da mikroprocessoren har en integreret ADC er det oplagt at anvende den. Den har dog ikke en integreret DAC og er derfor afhængig af en ekstern af slagsen. Mikroprocessoren er udstyret med to USART[2] moduler, hvoraf den ene anvendes til den serielle kommunikation mellem klient-PC'en og boardet. Det andet modul kan anvendes til *"Serial Peripheral Interface"* (SPI), som er en veldefineret protokol til kommunikation mellem forskellige kredse i et system.

Dette er ikke umiddelbart noget problem, men for at kunne styre den eksterne serielle DAC, kræves det at den udstyres med en clockfrekvens til at styre den ønskede bitrate på den serielle bus. Denne bitrate udregnes som følger:

$$F_s \cdot \textit{Bitopløsning på DAC} = \textit{clockfrekvens}.$$

[2]USART står for "Universal Synchronous/Aynchronous Receive/Transmit" og er en chip der kan læse fra en data bus og outputte det på en seriel linie. To enheder forbundet via USART kan sende tegn til hinanden, i henhold til ASCII tabellen

Da den integrerede ADC har en opløsning på 12 bit, vil denne opløsning være et naturligt valg til den eksterne DAC.
SPI-protokollen foreskriver imidlertid, at der sendes otte bits ad gangen, og man er derfor nødt til at sende fire dummybits ved transmissionen af de 12 databits [8].
Tages ovenstående formel i anvendelse med en opløsning på de 16 bits som skal sendes, gives en clockfrekvens på 352,8 kHz - en frekvens mikroprocessoren er ikke umiddelbart i stand til at outputte. For at give DAC'en den påkrævede frekvens kan et oscillatorkredsløb konstrueres, eller en pin på mikroprocessoren kan sættes til at outputte frekvensen ved at danne perioder med skiftevis high-low tilstand via. en af de indbyggede timermuligheder, så der gives et interrupt der skifter tilstand på et pin. Da et tomt interrupt som tidligere nævnt tager 11 cycles, vil dette medføre at der anvendes $3,88 \cdot 10^6$ clocks per sekund til interrupts, hvilket er mere end havldelen af mikroprocessorens kapacitet.
Det efterlader muligheden for et eksternt oscillatorkredsløb, der kan generere en passende clockfrekvens. Denne løsning er i modsætning til ovenstående mulig, problemet er dog, at det er besværligt at få denne synkroniseret med resten af systemet. Med det menes, at sikre at den får sendt alle bits over til DAC'en inden den næste frame er nået.
Ydermere betyder anvendelsen af SPI, at det koster 2 interrupts fra USART til at sende de 16 bits af 2x8 bit, hvilket betyder at der kræves som minimum 22 cycles ekstra pr. frame, udover det interrupt, der skal styre sampleraten på ADC'en.
En mere fordelagtig løsning, set i forhold til forbruget af cycles, er at bruge en parallel bus, som eksternt konverteres til en seriel bus. Dette forklares mere detaljeret senere i rapporten. Fordelen ved den parallelle bus, frem for den serielle er, at den kun behøver et interrupt for at sende og modtage [9] [7].

En anden mulighed er at anvende en 12 bit parallel DAC, så der sendes data ud til DACen via. halvanden port af 8 pins på mikroprocessoren, direkte over til DAC'en. Dette er ligeledes en oplagt mulighed, men laboratoriet har ikke haft nogle af typen, der opfylder kravet om at kunne drives med en 9V spændingsforsyning. I begge løsningsforslag skal det bemærkes, at der reelt set benyttes 16 bits eller der er muligheden for at benytte 16 bits, uden at det koster mere CPU-tid, da det ikke gør nogen forskel om man skriver til halvanden eller til to porte.

Den valgte ADC/DAC er en CS4218, dette valg er taget på baggrunden af ovenstående overvejelser, samt inspiration fra [10]. Denne opfylder alle de stillede krav, og har

ydermere andre meget ønskværdige egenskaber såsom 64x oversampling og decimering, der nedsætter kravene til et anti-aliasing præfilter betydeligt, så kun et 1. ordens RC-filter er påkrævet. Denne ADC/DAC er på 16 bits og kører serielt. Denne ADC/DAC understøtter ikke SPI, så en anden måde for at sende og modtage dataene er påkrævet.

5.2.2 CS4218 konfiguration

I dette afsnit beskrives opsætningen af CS4218 mht. anvendelsen i systemet. Diagrammet for opsætningen er vist i appendiks B.

Mode setting

Overførsel af data fra CS4218 sker serielt, og denne undersøtter ikke SPI som nævnt tidligere. Den er dog meget fleksibel mht. opsætning af forskellige modes til dataoverførsel mv. Der er tre forskellige serielle interfaces; SM3, SM4 og SM5 der vælges udfra tre pins på kredsen, SMODE1, SMODE2 og SMODE3. Udover de tre modes har SM3 og SM4 begge submodes i form af master eller slave-mode. Master/slave submodes angiver om CS4218 skal være den styrende chip til resten af kredsløbet/delkredsløbet, eller om denne skal styres af en anden enhed.
I praksis betyder det om CS4218 kontrollerer sampleraten der opereres med, og om den skal outputte, eller have inputs til at styre synkroniseringen og bitrate i kredsløbet [11].
De tre forskellige primære serielle interfaces har hver deres funktioner [11]. Disse beskrives her:

- SM3 Mode
  I SM3 mode, bruges der 4 serielle pins, SDOUT, SDIN, SCLK og SSYNC. Pin SDOUT er det A/D konverterede output fra CS4218. Dette output er på 64 bits per frame (BPF); de første 32 bit til audiosignalet, og derefter 32 kontrolbit som bruges til at indikere clipping i lydsignalet, fejlkoder til hurtigere debugging, og at fortælle om andre tilstande i CS4218. SDIN som er det input til D/A konverteren på CS4218, er ligeledes på 64 bits, og er opstillet på samme måde som SDOUT. De første 32 bit er audiosignalet fra mikroprocessoren, og de sidste 32 er kontrolbit.
  Forskellen er dog at de 32 kontrolbits nu styrer CS4218, med indstillinger som

mute, gain og til styring af den interne MUX, der bestemmer hvilken af de to tilstedeværende input-kanaler der skal anvendes. SCLK styrer dataflowet til og fra CS4218, hvor SSYNC indikerer begyndelsen og slutningen på en frame. De to sidstnævnte kan være output eller input pins, afhængig om CS4218 konfigureres som hhv. master eller slave.

- SM4 Mode
  SM4 mode fungerer på samme vis som SM3 mode, med den undtagelse af den bruger en separat port til at sende de 32 kontrolbits. Det medfører, at den påkrævede databåndbredde halveres, da det så bliver muligt at fastlåse konfigurationen via pins på CS4218.

- SM5 Mode
  Mode SM5 benytter kommunikationsprotokollen I^2S, men er ellers identisk med SM3. Denne protokol er inkompatibel med mikroprocessoren der arbejdes med, og er derfor uden relevans.

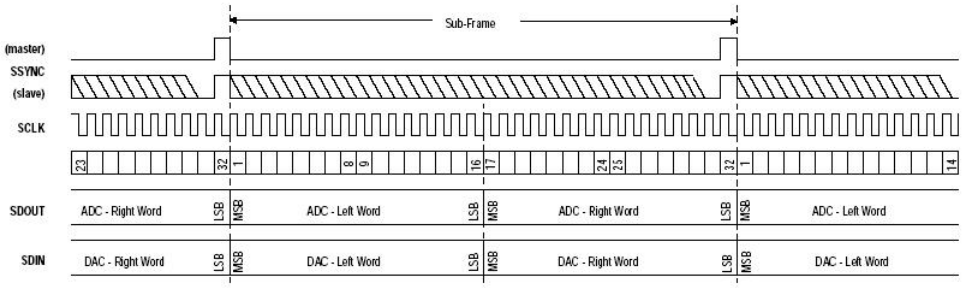
Den valgte mode til det serielle interface er SM4 mode, med master som submode. Det er et oplagt valg, da de 32 kontrolbits er separeret fra den serielle kommunikation. Disse kontrolbits kan hardwires da der er behov for ændringer i indstillingerne. Tabel 5.1 beskriver indstillingerne for mode SM4 og dataflowet kan ses på figur 5.1.

SMODE pins 1 2 3	Seriel mode	SCLK BC	BPF	Clock valg SCLK/SSYNC	Master frekvens
1 0 0	SM4	Fald	32 bits	M/S	11,2896 MHz

Tabel 5.1: Indstillingerne ved valgte SM4 mode (efter [11]).

Her gælder det at:
-SCLK BC[3] angiver hvilken tilstand SCLK-frekvensen er i, når bittet er halvvejs, se figur 5.1.
-BPF er den samlede framestørrelse.
-Clockvalg angiver valget af SCLK og SSYNC mht. master- og slave mode.
-Master frekvens er den primære clockfrekvens.

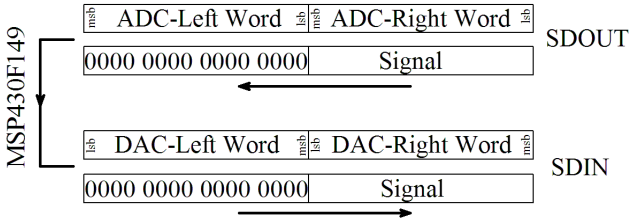
[3]Bit center



Figur 5.1: Figuren beskriver dataflowet til og fra CS4218 (af [11]).

Analogt input/output

Af kravspecifikationen fremgår det at, signaloutput fra effektboardet er mono. CS4218 chippen er udstyret med to 32 bit stereo line indgange med kanalerne LIN1, RIN1 eller LIN2, RIN2, hvor hver kanal er på 16 bit. Kanalerne der ikke bliver brugt eller valgt af mode settings, har et højt input impedans ($> 1M\Omega$), og skal derfor forbindes direkte til den analoge ground (AGND) for at forhindre DC strøm. Ligeledes skal kanaler som vælges af mode settings, og ikke bliver brugt, sættes til AGND via en 0,1 uF kondensator. Ved den valgte mode setting SM4, bliver LIN1 og RIN1 sat til lydinputkanaler, og de resterende kanaler sættes til AGND igennem en 0,1 uF kondensator. Da signalet skal være i mono, groundes LIN1 ligeledes hvilket giver RIN1 som eneste lydinputkanal. Dataframet, på baggrund af disse valg, er vist på figur 5.2.

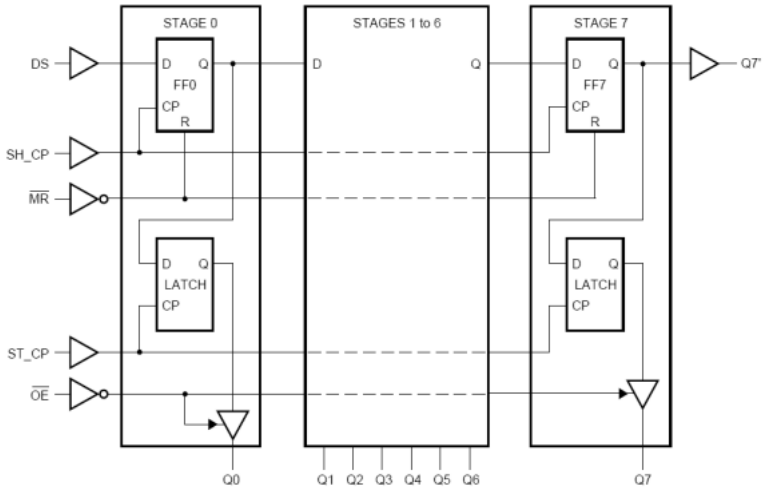


Figur 5.2: Signalet i SDIN er på DAC'ens left word. Der er to line-out kanaler i CS4218, left-out (LOUT) og right-out (ROUT), LOUT sættes som udgangskanal (efter [11]).

5.2.3 Kommunikation ml. ADC/DAC-kreds og mikroprocessor

For at kommunikere med CS4218, skal mickoprocessoren outputte parallelt, som eksternt konverteres til serielt output. Dette er hensigtsmæssigt i det at en parallel data-bus, for mikroprocessoren, er en mindre cycle-tung kommunikationsform.
Til konverteringen fra CS4218's serielle output, og til mikroprocessorens parallelle input, bruges to kredse af typen Philips 74HC595, som er 8 bit skifteregistere, med output latch (se figur 5.3) [12].

Når serielclocken SCLK[4] går høj, indlæses det første bit i det serielle data input (DS), i det første bit i skifteregisteret. Ved næste SCLK flyttes inputtet til det næste bit i skifteregisteret, og et nyt bit indlæses fra DS til det første bit. Dette gentages for hvert puls fra SCLK. Indlæsningen til et 8 bit skifteregister er vist i tabel 5.2.



Figur 5.3: Strukturdiagram over 74HC595. FF0 - FF7 er skifteregistrene, hvis output gemmes i en latch, der sender samplen ud parallelt, når SSYNC (ST_CP) går høj (af [12]).

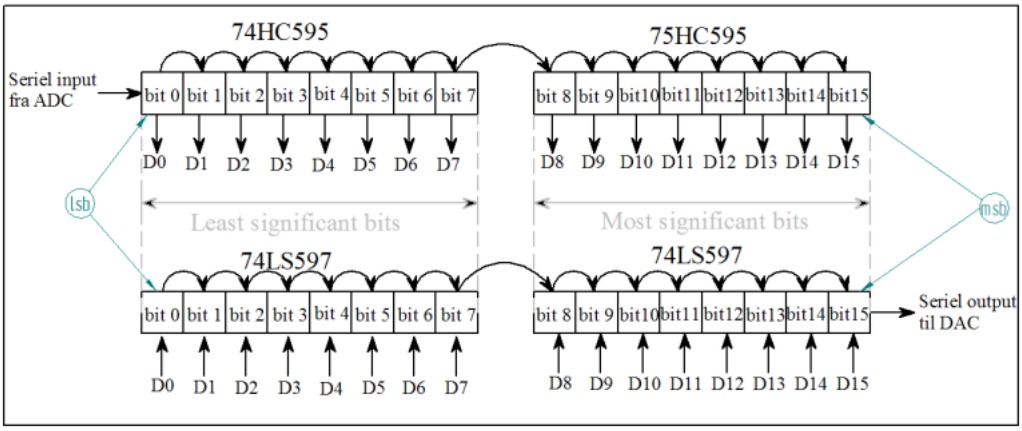
Når SSYNC[5] går høj, indlæses de binære værdier tli et array af latches, som outputter dataene parallelt til mikroprocessoren. Latchene forbliver uændrede, indtil den næste puls fra SSYNC, hvor de nye værdier indlæses fra skifteregistrene.
Konverteringen fra parallel til seriel virker omvendt, hvor det parallelle signal gemmes

[4]SH_CP på figur 5.3 har en frekvens på 705kHz
[5]ST_CL på figur 5.3 har en frekvens på 22,05 kHz

SH_CP	DS	Q'_0	Q'_1	Q'_2	Q'_3	Q'_4	Q'_5	Q'_6	Q'_7
1	1	1	0	0	0	0	0	0	0
2	0	0	1	0	0	0	0	0	0
3	1	1	0	1	0	0	0	0	0
4	1	1	1	0	1	0	0	0	0
5	0	0	1	1	0	1	0	0	0
6	1	1	0	1	1	0	1	0	0
7	0	0	1	0	1	1	0	1	0
8	1	1	0	1	0	1	1	0	1

Tabel 5.2: En 8 bit seriel datastrøm [10110101] skiftes i takt med SCLK, der ved hvert plus "skifter" dataene en plads til højre i registret.

i et array af latches, som sendes serielt bit for bit, i takt med SCLK. Til konverteringen anvendes to kredse af typen Texas Instruments 74LS597 [12][13]. Hver kreds kan gemme 8 bit, så der anvendes to af hver kredstype (74HC595 og 75LS597). Ved at forbinde dem parvis fås et samlet skifteregister med en kapacitet på 16 bit, hvor den ene udgører de 8 lsb (least significant bit) og den anden udgør de 8 msb (most significant bit) (se figur 5.4).



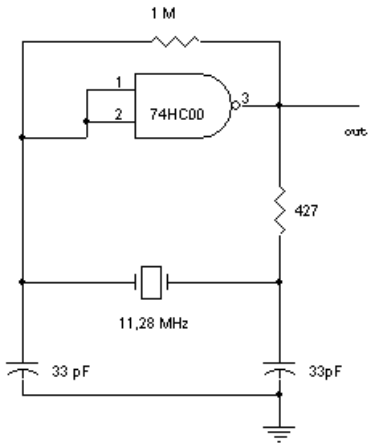
Figur 5.4: De fire kredse er forbundet parvist serielt, og udgør derved en 16 bit seriel til parallel konverter, og en 16 bit parallel til seriel konverter. D0 til D15 er de enkelte bits på de to 16 bit busser til mikroprocessoren.

CS4218, som transmitere SCLK og SSYNC, arbejder med stereo lyd frames af 32. Derfor påvirker det ikke båndbredden at der arbejdes med mono lyd. Framelængden er stadig på 32 bit, men output fra venstre kanal er er kun nuller (se figur 5.2). Det betyder, at 74HC595(seriel til parallel) i realiteten gennemgår 32 bit, hvor de første

16 bit er den venstre kanal og de sidste 16 er den højre kanal. De 16 første bit skiftes ud af registret, så kun værdier for den højre kanal læses.
For 74LS597(parallel til seriel) virker det lidt anderledes. Mikroprocessoren outputter 16 bit til latchene, der flytter disse over i skifteregisteret. Disse data bliver nu shifted over i CS4218, der læser dem til venstre outputkanal. Da den serielle indgang på første skifteregister er sat til GND, shiftes der nuller ind i registeret som så shiftes over i CS4218's højre outputkanal, som ikke tages i anvendelse. Det vil sige, at lyden skifter kanal fra input til output.

5.2.4 Oscillator

Kredsen CS4218 kan konfigureres til at operere ved forskellige samplerates. Dette indstilles ved at stille på tre pins på kredsen; F1, F2 og F3[6]. For at generere en samplerate på 22,05 kHz (SSYNC) og en seriel clockfrekvens (SCLK) på 705,6 kHz, kræver CS4218 en inputclock på 11,2896 MHz [11]. Da mikroprocessoren ikke understøtter en clock ved denne hastighed anvendes en ekstern oscillator.
Da der ikke var muligt at skaffe en ekstern oscillator med en frekvens på 11,2896 MHz, anvendes en krystal, ved den givne frekvens, og et kredsløb for oscillatoren konstrueres. Kredsløbet for oscillatoren er vist på figur 5.5 [14].



Figur 5.5: Her afbildes kredløbet for den eksterne oscillator med 11,2896 MHz [14].

[6]Disse pins varierer i de forskellige SM modes

5.3 Analogt præfilter

Ifølge databladet for CS4218 anbefales det at bruge et eksternt anti-alising præfilter. På bagrund af disse oplysninger designes der et eksternt præfilter bestående af et lowpass- og et highpass filter. Egenskaben ved en lowpass filter er, at den lader alle frekvenser der er lavere end knækfrekvensen f_{max} passere, og et highpass filter lader alle frekvenser der er højere end knækfrekvensen passere. Kravene til det analoge præfilter opstilles i forlængelse af hvad der står i kravspecifikationen i afsnit 2.3 mht. sampling.

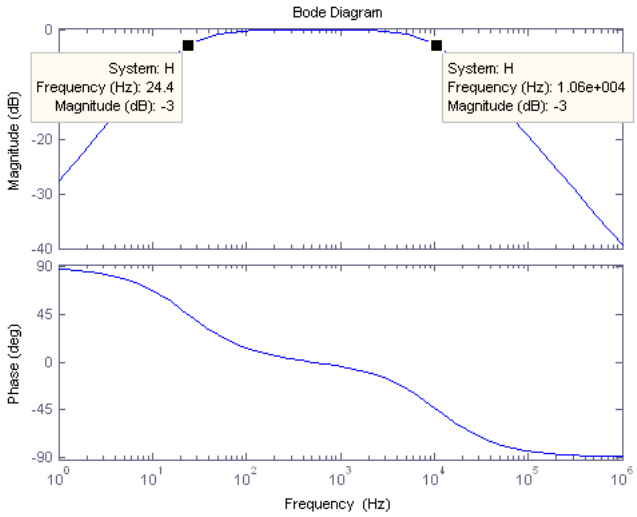
1. **Lowpass filter**
   Det fremgår af kravspecifikationen, at signalet skal samples med 22,05 kHz, dermed fås stopbånd til 11,025 kHz.

2. **Highpass filter**
   DC ønskes frafiltreret og stopbåndet vælges til at være en passende lav frekvens på 25 Hz.

Beregningsmetoden for filtret og de forskellige komponenter, beskrives nærmere i appendiks A. Ud fra disse beregningerne vælges der nogle passende komponenter, som tilfredsstiller båndpass filtrets krav og indsættes i den fundne overføringsfunktion. De valgte komponenters størrelse er:
$R_1 = 5600\,\Omega$, $R_2 = 40000\,\Omega$, $C_1 = 1,33\,nF$ og $C_2 = 330\,nF$.
Beregningerne viser at knækfrekvensen for lowpass filteret er på 10,6 kHz og highpass filteret på 24,4 Hz.
Bodediagrammet af overføringsfunktionen med disse komponenter kan ses på figur 5.6. Det fremgår af figuren, at båndpassfilteret overholder de stillede krav.

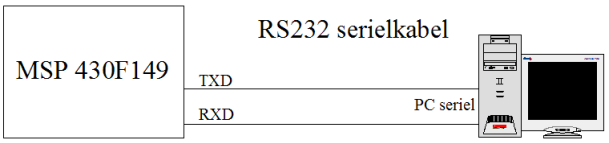
5.4 Kommunikationsgrænseflade

Til styring af effekterne på effektboardet er der en kommunikationsgrænseflade mellem boardet og en klient-PC, via en seriel forbindelse over RS-232 standarden.
Pin 32 og 33 på mikroprocessoren, er hardwaregrænsefladen for den indbyggede USART, hvor pin 32 er UTXD0 (USART transmit data) og pin 33 er URXD0 (US-



Figur 5.6: Bodeplot over en båndpass filteret med en knækfrekvens på hhv. 24,4 Hz og 10,6 kHz ved 3 dB dæmpning.

ART receive data). Disse kan via en linedriver kobles direkte til en klient-PC's serielport (se figur 5.7).

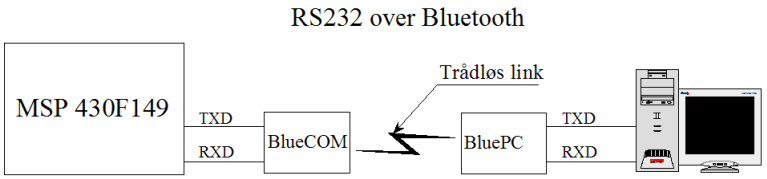


Figur 5.7: Seriel kommunikation mellem mikroprocessor og klient-PC. UTXD0 er transmit data og URXD0 er receive data.

Der anvendes et indlejret bluetoothmodul, som er forbundet til mikroprocessorens URXD0 og UTXD0 og har en rækkevidde på op til 10 m. Bluetoothmodulet kommunikerer med en bluetooth USB-dongle, som er placeret i klient-PC'ens USB port, se figur 5.8. På denne måde opnås en trådløs serielforbindelse mellem mikroprocessoren og klient-PC'en [15][16].

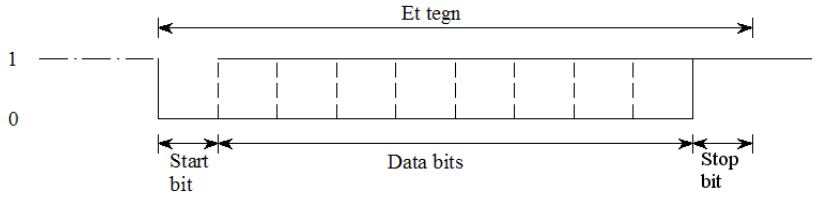
5.4.1 Seriel kommunikation

Data sendt over en seriel forbindelse, kan sendes synkront eller asynkront. Asynkron seriel kommunikation foregår uafhængig af en fælles clockfrekvens mellem sender



Figur 5.8: Seriel kommunikation mellem mikroprocessoren og klient-PC over bluetooth.

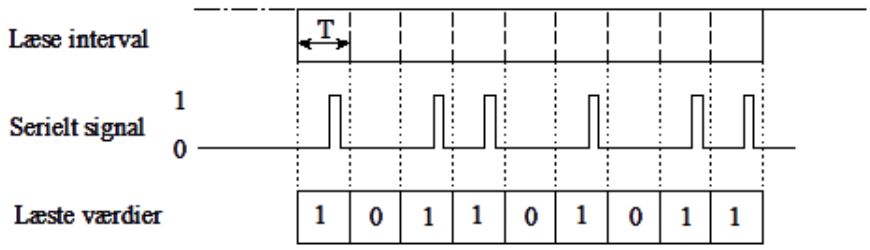
og modtager. Det gøres i stedet ved at modtageren venter på at signalet går fra høj til lav, et såkaldt start, som efterfølges af 8 databit og et stop bit (se figur 5.9).



Figur 5.9: Et tegn opdeles i 10 bit, et start bit, 8 data bit og et slut bit. Start bit er det første bit, efter at signalet går fra højt til lavt (efter [15]).

Et tegn kan sendes over en seriel forbindelse, i sin binære repræsentation i henhold til ASCII tabellen. Når et tegn sendes, sker det med least-significant-bit først og most-significant-bit til sidst.
Asynkron seriel kommunikation anvendes oftest mellem to adskilte kredsløb, da de typisk ikke deler en synkroniseringsclock. I stedet har de hver en clock med den samme clockfrekvens. For at sikre at de enkelte bit ikke går tabt, er det nødvendigt for modtageren, at have et tidsinterval T, hvor et bit læses. Dette tidsinterval er defineret som:

$$baud\ rate\ [\text{bps}] = \frac{1}{T}\ [\text{bps}]$$



Figur 5.10: Modtageren venter på et bit i intervallet T, komme der et impuls indenfor dette interval sættes bittet til 1 ellers sættes det til 0, hvilket betyder at det ikke er nødvendigt at de to clocks er synkrone, blot de tikker med samme frekvens.

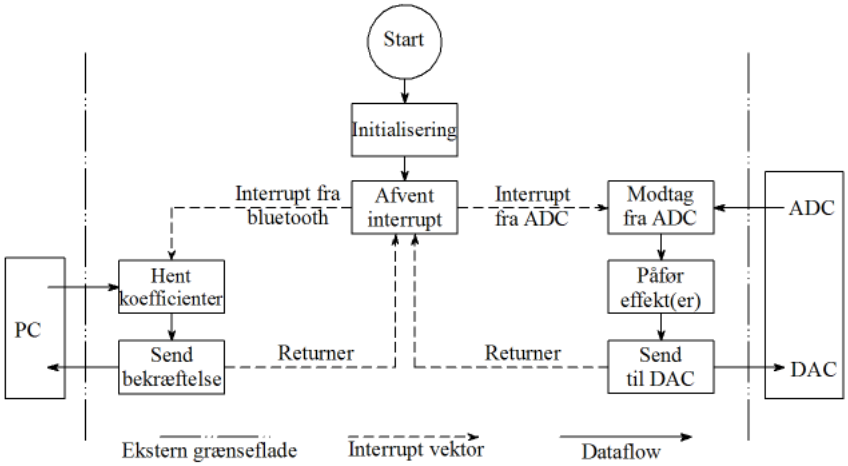
Kapitel 6

Softwaredesign

Programmellet er opdelt i to dele. Den ene del er placeret på mikroprocessoren og den anden på klient-PC'en.
Softwaren på mikroprocessoren er programmeret delvist i C, og delvist i assemblerkode. Årsagen til dette er, at i realtids-routinen er der flere muligheder for at optimere i assemblerkode, da det er nærmere hardware-laget. Programmet på klient-PC'en er designet til, at udregne koefficienterne til effekterne, og videresende dem til mikroprocessoren.

6.1 Programstruktur på mikroprocessoren

På mikroprocessorens softwareside er det valgt at inddele programmet i processer, som hver har en minimal grænseflade hinanden imellem. I praksis løses opgaverne på mikroprocessoren sekventielt, men ved at nedbryde systemet i pseudoparallelle processer overskueliggøres opgaven.
Softwaren kan opdeles i to pseudoparallelle processer, med udgangspunkt i en uendelig main-løkke, hvor der afventes et af to interrupts. Interruptet kan enten være et periodisk interrupt bundet til den serielle-synkroniserings-clock SSYNC, der er styres fra CS4218. Alternativt kan det være et ikke periodisk interrupt, når der modtages noget over den serielle forbindelse over bluetooth (se figur 6.1).



Figur 6.1: Strukturdiagram over mikroprocessorsoftware. Main er kaldemiljøet, hvor der under initialiseringen aktiveres moduler, som muliggører interrupts. Programmet venter i en uendelig løkke, hvorfra der kan interruptes. Linjerne [Interrupt fra ADC] og [Interrupt fra bluetooth] skal forståes som et programhop fra main, når der indtræffer et interruptkald.

6.1.1 SSYNC-interruptet

Dette interrupt starter en kodesekvens, der gennemløber tre funktioner. Det første består i at indlæse en sample fra ADC'en og placere den i et forud bestemt register. Efterfølgende behandles samplen med udvalgte effekter og outputtes til DAC'en. Herefter returnerer programmet til main løkken, hvor der afventes et nyt interrupt.

6.1.2 Bluetooth-interruptet

Dette interrupt er ikke periodisk, da det aktiveres ved indkommende data i USART'ens receive-buffer, hvilket kræver en brugerinteraktion.
De indkommende data er koefficienter, der er beregnet i klientsoftwaren, ud fra brugerindtastede oplysninger. Disse koefficienter flyttes til designerede adresser i hukommelsen, hvor de bliver benyttes ved påførelse af audioeffekter.

6.1.3 Initialisering

Her initialiseres alle variable i systemet, som input og output, samt de forskellige koefficienter til effekterne. De enkelte moduler i mikroprocessoren aktiveres, og

portene konfigureres til en funktion, det værende input/output eller anden funktionalitet som USART. Et pin sættes som reset-pin til CS4218, da den kræver et reset på mindst 50 ms. Endvidere sættes koefficienterne til de respektive effekter til en default værdi, mens input- og outputsamples sættes til 0. Som det sidste led i initialiseringen aktiveres muligheden for interrupts, og systemet går i venteløkken, hvor der afventes et interrupt.

6.2 Interrupts

Et interrupt er en metode til at afbryde en kørende proces, hvorpå man kan køre en alternativ proces, og efterfølgende vende tilbage til den oprindelige proces, hvor den blev afbrudt.
Et interrupt kan styres enten af en af mikroprocessorens timere eller med et eksternt interruptkald på et pin [9].

Når der kaldes en interruptrutine, er der en forsinkelse på 6 cycles. Denne forsinkelse skyldes initialisering af interruptet, hvor der udføres følgende:

1. Igangværende instruktioner afsluttes.
2. Program counter (PC), som peger på den næste instruktion i stakken, bliver "pushed" ind i stakken.
3. Statusregisteret (SR) bliver "pushed" ind i stakken.
4. Interruptet med den højeste prioritet bliver valgt, hvis der er flere i kø efter den sidste instruktion.
5. Interruptflaget for det valgte interrupt resettes, så det ikke vælges igen.
6. SR bliver nulstillet, med undtagelse af SCG0[1] som forbliver urørt. General interrupt enable (GIE) som ligger i SR slåes fra.
7. Indholdet af interruptvektoren, som peger på den rutine der skal køres, lægges ind i PC.

[1] SCG0 = system clock generator 0

PC og SR bliver ”pushed” ind i stakken, således at det er muligt at vende tilbage til udgangspunktet, når interruptet er afsluttet. Idet at GIE bliver nulstillet, ophæves muligheden for at kalde et interrupt. Det er derfor ikke muligt at kalde et interrupt mens et allerede kører. Skulle der komme et eller flere interrupts imens, afventer de indtil ophøret af det igangværende interrupt. Derefter eksekveres det interrupt der har den højeste prioritet og der fortsættes fra punkt 4 ovenfor.
Det er dog muligt at tillade interrupts i et interrupt, som kaldes et nested interrupt. Dette gøres ved at inkludere et ”interrupt enable” i interruptrutinen, for det enkelte interrupt. Med udgangspunkt i anvendelsen af interrupts i dette projekt, er det ikke hensigtsmæssigt at tillade nestede interrupts, da det f.eks. kan forårsage datatab ved en seriel transmission.
Når interruptrutinen er udført, er der igen en forsinkelse på 5 cycles, før det oprindelige program fortsættes. Dette skyldes RETI (return from interrupt) hvor følgende udføres:

1. SR med værdierne fra før interruptet ”poppes” fra stakken, og derved aktiveres GIE igen.

2. PC ”poppes” fra stakken og fortsætter det oprindelige program, hvor den slap før interruptet.

6.3 Grænseflader

Udfra figur 6.1 identificeres programmets eksterne grænseflader til:

- Seriel kommunikation til klient-PC.
- Input fra ADC.
- Output til DAC.

Af interne grænseflader findes:

- Koefficient indlæsning fra seriel interrupt til audiointerruptet.
- Input/output relation filtrene imellem.

De eksterne grænsefladerne adskiller softwaren og hardwaren. Den interne grænseflade adskiller software og software, hvilket skal forstås på den måde, at de adskilte moduler blot kommunikerer på tværs af grænsefladerne.

Kapitel 7

Softwareimplementation

Dette afsnit omhandler implementation, af det designede software. Software'en skal overholde de grænseflader/moduler der er opstillet i designfasen. Da softwaren er opdelt i moduler og processer, kan disse implementeres og testes enkeltvis, inden der arbejdes videre i implementationsfasen. Den første proces der er implementeret er hhv. ADC/DAC håndtering. Disse testes enkeltvis med fastprogrammerede inputs /outputs, og kobles senere sammen med henblik på at lave en "talkthrough" så audiosignalet føres direkte igennem systemet, hvorefter effekterne med relativ enkelthed kan indføres og testes på samme vis med fastprogrammerede værdier.

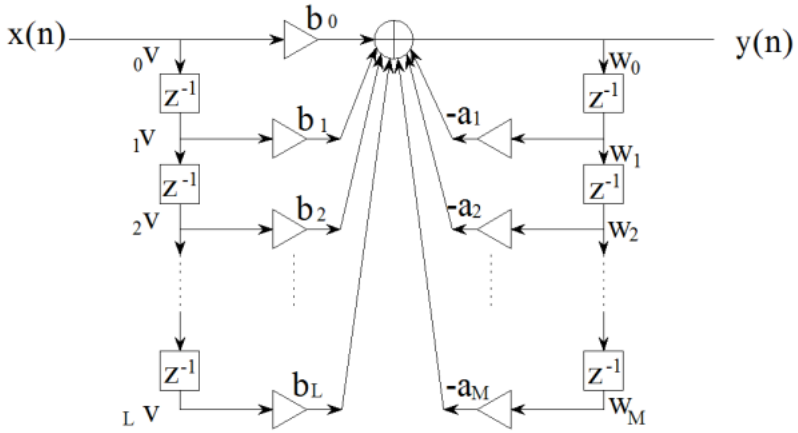
7.1 Direkte form I

Et digitalt filter kan realiseres, hvis overføringsfunktionen opfylder følgende krav: funktionen er lineær, tidsuafhængigt og kausalt.
I denne rapport anvendes der "direkte form I" til realiseringen af equalizeren.
En given overføringsfunktion af et M'te-ordens filter, bestående af filter koefficienter $a_1, a_2, a_M, b_0, b_1, b_2$ *og* b_L, kan udtrykkes i en I/O differensligning på følgende måde:

$$y_{[n]} = -\sum_{i=1}^{M} a_i y_{n-i} + \sum_{j=1}^{L} b_j x_{n-j}. \tag{7.1}$$

Koefficienterne på højre side af realiseringsdiagrammet på figur 7.1 er repræsenteret ved inputsamples og venstre side er outputsamples, disse kan benævnes som delay lines, der svarer til filterets orden. Koefficienterne b_L svarer til tællerkoefficinterne

i overførningsfunktionen og a_L svarer til nævnerkoefficinterne. Disse kaldes hhv. feed forward, og feeding back koefficienter. Til hver sample n, ganges filterkoefficienter på indholdet af delayregistrene og samples akkumulers i en enkelt adder.



Figur 7.1: Implementation af et digitalt filter, afbildes i et filter-realiseringsdiagram. Her illustreres realiseringen af et M'te-ordens filter (efter [4]).

Delayenheden z^{-1} udtrykker input/output delaylinet, disse beskrives ved:

$$v_i(n) = x(n-i), \;\; i = 0, 1, ..., L. \tag{7.2}$$

$$w_i(n) = y(n-j), \;\; j = 0, 1, ..., M. \tag{7.3}$$

7.2 Input/outputrutine

De samples der sendes til skifteregistrene, og derved ind på to porte på mikroprocessoren, kan aflæses i de to tilhørende adresser. Værdierne på portene er af en størrelse på 8 bit, og skal sættes sammen til den 16 bit værdi der er afsendt fra ADC'en. Dette foregår ved at indlæse tallene til to 16 bit registre, hvorefter registeret med de otte msb swapper bytes, svarende til et logisk shift 8 pladser mod venstre. Herefter foretages et logisk OR med msb registeret, og lsb registeret, hvorefter værdien er indlæst.
Output af det behandlede signal foregår på stort set samme måde. Værdien skrives ind i lsb outputportens adresse med et logisk AND, hvorefter der igen swappes bytes, hvilket nu svarer til et logisk shift til højre med otte pladser, og den nye værdi skrives til msb outputportens via logisk AND. Et kodeeksempel på dette findes

på figur 7.2.

/∗ Inputrutine ∗/			/∗ Outputrutine ∗/		
MOV.B	&0x34,	R4	BIS.B	R15,	&0x29
MOV.B	&0x30,	R5	SWPB	R15	
SWPB	R4		MOV.B	R15,	&0x1d
BIS.W	R5,	R4			

Figur 7.2: Kodeeksempel i assembler til en 16 til 2x8 bit input og output-rutine

7.3 Beregningskompleksitet

Da der er tale om realtidsbehandling af lyden, er forbruget af cycles et kritisk emne for hvilke filtre der kan tages i anvendelse. Filtrenes overføringsfunktioner konverteres til differensligninger, der giver et billede af beregningskompleksiteten af det pågældende filter, med henblik på multiplikationer.

Når der kaldes en interrupt-rutine bruges der 6 cycles på at initialisere interruptet og 5 cycles på at returnere til hovedprogrammet igen, altså 11 cycles pr. interrupt. Derudover bruges der også cycles til at læse fra ADC'en og skrive til DAC'en. Et eksempel på beregning af forbruget af cycles, er at bruge assembler instruktionen "MOV &input,&output" til at lave en såkaldt "talkthrough". Dette sker ved at flytte input signalet fra porten forbundet til ADC'en direkte til porten forbundet til DAC'en. Til ovenstående instruktion skal der bruges $2 \cdot 6$ cycles (til hhv. otte msb og otte lsb). Dette sker i et interrupt, og giver en total på $11 + 2 \cdot 6 = 23$ cycles pr interrupt, således er der 334-23 = 311 cycles tilbage til filtreringen af inputtet [9] [7].

7.3.1 Hardwaremultiplier

Mikroprocessoren besidder en integreret hardwaremultiplier, der med fordel kan tages i anvendelse når differensligningerne fra equalizeren og compressoren skal udregnes. Hardwaremultiplieren har fem 16 bit adresser der kan læses værdier ind

```
/* Differensligning /*

MPYS  = b0;
OP2   = x[0];
MACS  = b1;
OP2   = x[1];
MACS  = -a1;
OP2   = y[0];
```

Figur 7.3: Kodeeksempel i C til et 1. ordens filter ved brug af den integrerede hardwaremultiplier.

i. Fire til at indlæse den første operand, og samtidigt vælge hvilken mode man vil anvende på multiplikationen, og en til at indlæse den anden operand der skal multipliceres med (OP2).
De fire modes er:

- MPY: Unsigned multiplikation.
- MPYS: Signed multiplikation.
- MAC: Unsigned multiplikation med akkumulator.
- MACS: Signed multiplikation med akkumulator.

Da der opereres med værdier der kan være såvel positive som negative værdier, er det kun MPYS, og MACS der er aktuelle. Fælles for de fire modes er at resultatet bliver gemt i to 16 bit registre (RESHI og RESLO), da 32 bit er den maksimale størrelse en enkelt multiplikation kan antage. Resultatet kopieres derudover til en "adder" der anvendes hvis to tal skal multipliceres og adderes med resultatet fra forrig multiplikation [9].
Denne egenskab er oplagt ved behandling af differensligninger til at filtrering af audiosignalet. Differensligningen til et 1.ordensfilter $y[n] = b0x[n] + b1x[n-1] - a1y[n-1]$ kan skrives i C som vist i figur 7.3:

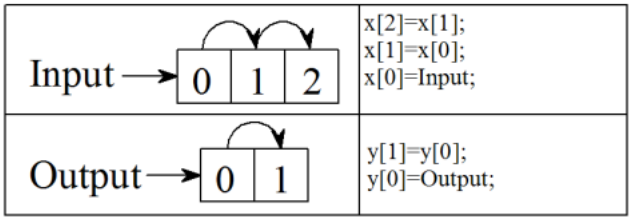
7.3.2 Equalizer

Equalizeren omfatter en summering af 9 multiplikationer hvis et 4. ordens filter anvendes. Et 4. ordens filter er dog mere tilbøjelig til at generere trunkeringsstøj [17]. Af denne grund, anvendes to 1. ordens shelving filtre og et 2. ordens peak/notch filter. I modsætning til 4. ordens filteret kræver dette 11 multiplikationer ($2 \cdot 3 + 5 = 11$). Til gengæld skal der ikke anvendes så meget hukommelse, da der gemmes færre samples.

Idet hardwaremultiplieren benyttes, skal der for hver multiplikation flyttes to værdier til to adresser tilknyttet hardwaremultiplieren. Til beregning af kompleksiteten anvendes et forsigtighedsprincip hvilket betyder, at der i de fleste tilfælde anvendes de mest cycle-tunge operationer muligt. Hvis det antages at de to værdier hentes fra hukommelsen, "koster" det $2 \cdot 6$ cycles.
Hardwaremultiplieren bruger 3 cycles til beregning, altså anvendes 15 cycles pr. multiplikation og der skal bruges mindst 3 cycles på at hente det endelige resultat [7]. Dette giver et maksimalt forbrug på 168 cycles.

Trukket fra de tilbageværende 333 cycles (minus 11 for interrupt), er der 165 cycles tilbage. Derudover skal input og output rutinerne også medregnes, hvilket beregnet udfra figur 7.2 giver 17 cycles. Tilbage er nu 148 cycles.

I forbindelse med equalizeren, skal tre tidligere inputsamples og to tidligere outputsamples gemmes. En måde er at lægge det i to arrays, som skifter en plads hver gang der kommer et nyt input og et nyt output se figur 7.4.



Figur 7.4: To arrays, som skifter en plads hver gang der kommer et nyt input og et nyt output, agere hukommelse for systemet. Til højre er funktionen repræsenteret i pseudokode og til venstre er de to arrayfunktioner illustreret.

Dette kræver 6 cycles pr flyt, hvilket resulterer i:

$$(3 \cdot 6) + (2 \cdot 6) = 30 \text{ cycles}. \tag{7.4}$$

Tilbage er nu 118 cycles til rådighed til det næste filter. Det skal her understreges at der er et fuldstændigt uoptimeret resultat, og at der er mange cycles at hente i optimeringsfasen ved at undgå de mange læse/skrive operationer til og fra hukommelsen, da dette er de mest cycle-tunge operationer der kan forekomme i systemet.

7.3.3 Compressor

Compressoren afviger fra andre effekter som equalizeren, ved at den er ulineær. Først del af effekten, som er niveaudetoktoren kan implementeres forholdsvist problemfrit, jævnfør direkte form I. Problemet med implementationen af formel 4.12 på side 24 opstår med eksponenten, i det der regnes med decimaltal repræsenteret ved fixed-point aritmetik. Af den grund er der valgt en løsning baseret på tabelopslag med udgangspunkt i brøken $\frac{c}{c_0}$. Her det er tiltænkt at der for værdier af eksponenten $\rho - 1$, skal kunne findes gainfaktoren G_n, udfra resultatet af brøken. Brøken giver imidlertid stor risiko for overflows da det kan give $\frac{x_n}{c_0}$, hvor $< 0 c_0 < 1$. Ved implementation omregnes c_0 til en værdi af signalets spænvidde, altså mellem 0 og 32768. For at undgå overflows, anvendes istedet $c \cdot c_0$ som opslag, hvilket der skal tages højde for tabellens talrækkefølge.
Opslagstabellen giver for $c \cdot c_0$, en konstant der skal ganges på x_n i stedet for eksponenten. Tabellen laves logaritmisk hvilket giver en bedre præcision når controlsignalet kun overstiger thresholden men en lille faktor, og mindre præcision som forskellen på c og c_0 stiger.
Sampleværdien $|x_n|$ i formel 4.11 side 24, laves absolut ved at udføre et logisk AND på denne med hex værdien 0x7FFF, dette gøres af hensyn til de negative udsving i signalet.

7.4 Fixed-point aritmetik

Under filtreringen arbejdes der med decimaltal hvilket ikke umiddelbart kan behandles i mikroprocessoren som arbejder med 16-bit tal. Ved at benytte fixed-point aritmetik kan decimaltallene omregnes til en binær representation. Notationen er givet ved $U(a, b)$, hvor a angiver antal bits der repræsenterer heltalsværdien, og b angiver hvor mange bits repræsenterer de fraktionelle tal [18]. Fremgangsmåden er at der vælges en repræsentation som passer bedst til den givne opgave, så der opnåes den bedst mulige præcision i decimaltallene, og ikke sker overflows i heltal. Multiplika-

tion ved to signed værdier er givet ved $U(a_0, b_0) \times U(a_1, b_1) = U(a_0+a_1+1, b_0+b_1)$

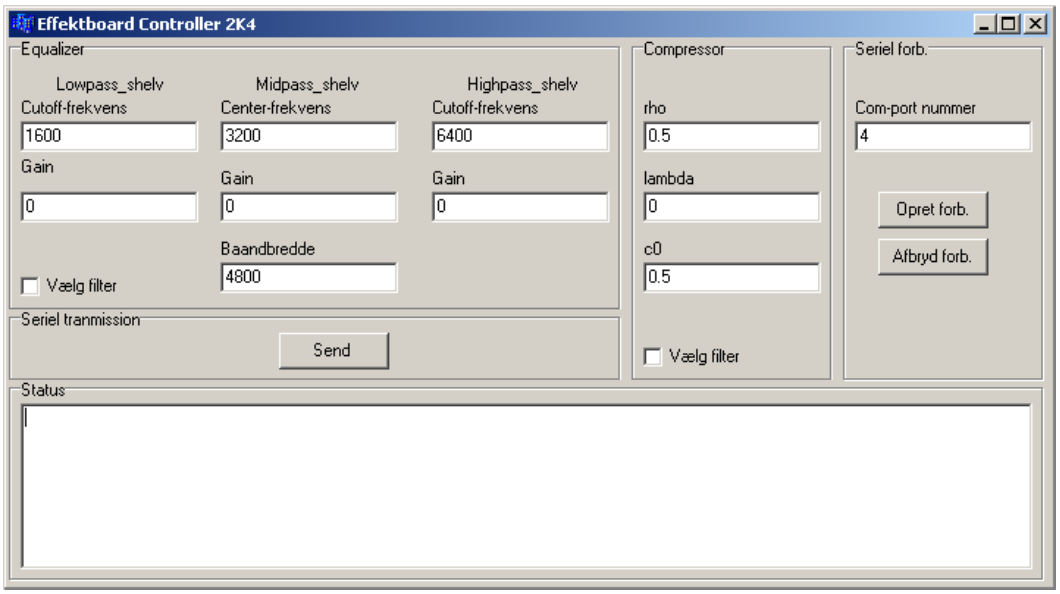
Et eksempel på en omregning med fixed-point aritmetik er en signed værdi af tallet 7,4. Dette tal ønskes repræsenteret med størst mulig præcision ud fra en 16 bit værdi. Derfor anvendes U(3,12) repræsentationen (1 bit til sign). Denne omregning kan udføres på følgende måde:
$\frac{N}{2^{-b}}$ eller $N \cdot 2^b$, hvor N er den værdi der ønskes omregnet.
På binær facon vil denne værdi se således ud [0111 0110 0110 0110]. Denne binære værdi svarer til 30310,4, hvilket afrundes til nærmeste heltal. Denne afrunding er det tab af præcision der kan forekomme ved brug af fixed-point aritmetik. Hvis tallet regnes tilbage til sin oprindelige form, er tallet nu 7,3999.

7.5 Klient-PC software

Til ændring af filterkoefficienter på effektboardet er der udviklet klientsoftware, som udregner koefficienterne og transmitterer disse til mikroprocessoren over en seriel forbindelse.
Programmet er bygget op omkring en C++ skabelon i Borland C++ builder, der leverer de grafiske elementer, men er primært skrevet i C. Det grafisk interface omfatter tekstbokse til parameterændring, checkbokse til filtervalg, et konsolvindue til statusbeskeder og en send-knappe der opretter en seriel forbindelse og påbegynder transmission.

7.5.1 Seriel kommunikation

Seriel kommunikation mellem mikroprocessoren og klienten sker i begge tilfælde med buffered input. På mikroprocessoren er der initialiseret en interruptroutine når der opstår input på U0RXBUF, hvilket er receiveregistret kommunikationen bruger. Når filterkoefficienterne er modtaget, nulstilles tidligere in- og outputsamples så der ikke kan opstå overflows.
Her skal det nævnes at ved oprettelse og afbrydelse af forbindelsen mellem bluetoothenhederne sendes der data, hvilket forårsager interrupt på effektboardet. Af den grund sendes et bestemt tegn før koefficientoverførslen påbegyndes, og samplingsprocessen afbrydes helt. På klientsiden benyttes en række funktioner fra windows-



Figur 7.5: Skærmbillede af den grafiskebrugerflade.

libraries, der behandler serielporten på samme må som der arbejdes med filpointere i C [19]. Ved specificering af en com-port kan der sendes og modtages. Her skal det nævnes at modtagelseskald virker indtil en buffer er fyldt op, eller til funktionen timer ud.

7.6 Optimering

Formålet med dette afsnit er, at afdække de åbenlyse områder, hvor det vil være fordelagtigt at optimere, og hvordan det kan optimeres.

7.6.1 Registre og hukommelse

Et af de tungeste operationer for mikroprocessoren, er hukommelsesoperationer. En flytning fra en adresse i hukommelsen, til en anden, kræver 6 cycles. Ved flytning fra et register til hukommelsen bruges 4 cycles og fra ram til register, 3 cycles. Operationer foretaget direkte i et regiester, eller fra register til register, bruger 1 cycle.
I tabel 7.1 ses i den venstre side en funktion i assembler, genereret af den anvendte

kompiler[1]. Denne funktion kan reduceres mht. antallet af cycles, ved så vidt muligt, at undlade at flytte data til og fra hukommelsen, men i stedet anvende de interne registre.

Input funktion	Cycles	Input funktion (optimeret)	Cycles
MOV.B &0x34, R15	3	MOV.B &0x34, R15	3
MOV.W R15, &input1	4	SWPB R15	1
MOV.B &0x30, R14	3	MOV.B &0x30, R14	3
MOV.W R14, &input2	4	MOV.W R14, R15	1
SWPB R15	1		
BIS.W R14, R15	1		
MOV.W R15, &x	4		
Total	20		8

Tabel 7.1: Til venstre ses assemblerudgaven, af den funktionen, der anvendes til at indhente data fra den externe ADC. Til højre er den optimerede udgave, hvor funktionen er reduceret fra 20 til 8 cycles.

[1]IAR Embedded Workbench for MSP430 V3.20A

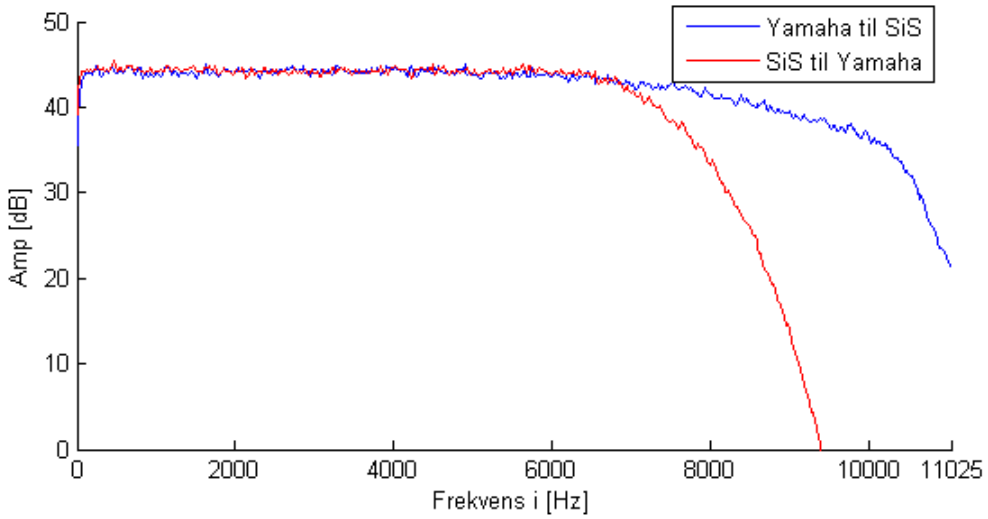
Kapitel 8

Systemtest

For at teste om systemet virker efter hensigten, er der lavet en analyse af systemets frekvensrespons ved at føre "white noise" gennem systemet. Whitenoise har den fordelagtige egenskab, at den er ligeligt repræsenteret i alle frekvensområder, og vil derfor afsløre om der sker nogle uhensigtsmæssige dæmpninger eller forstærkninger af signalet i frekvensspektret [20]. I praksis vil repræsentationen dog ikke være 100% lige, og af den grund kan der være en lille usikkehed i resultaterne.
Der føres white noise igennem systemt, fra en afsender PC og optages på en modtager PC. Resultatet af dette sammenholdes med en direkte PC til PC optagelse. Herved kan testen verificeres, så eventuelle usikkerheder i testudstyret udelukkes. Det vil sige, at der ikke må være nogle nævneværdige stigninger eller fald i amplituden, i vilkårlige frekvensområder i PC til PC testen.
Det anvendte testudstyr er en laptop med et "SiS 7018 Wave" integreret lydkort, og en stationær pc med et "Yamaha DS1x" lydkort. Til optagelse af lyden er line-in anvendt i begge tilfælde. Programmet der er anvendt til at generere den hvide støj, er Adobe's Audition 1.5, og til analyse af frekvensresponsen er MATLAB 7 anvendt. Udover tests af frekvensresponsen er der ligeledes udført uformelle lyttetests, hvor der gives en subjektiv vurdering af lydkvaliteten.

8.1 PC til PC

PC til PC testen på figur 8.1 viser at line-out på SiS 7018 har et meget upålideligt output, der allerede knækker ned ved ca. 7 kHz, hvorimod outputtet på Yamaha DS1x lydkortet håndterer de højere frekvenser langt bedre.

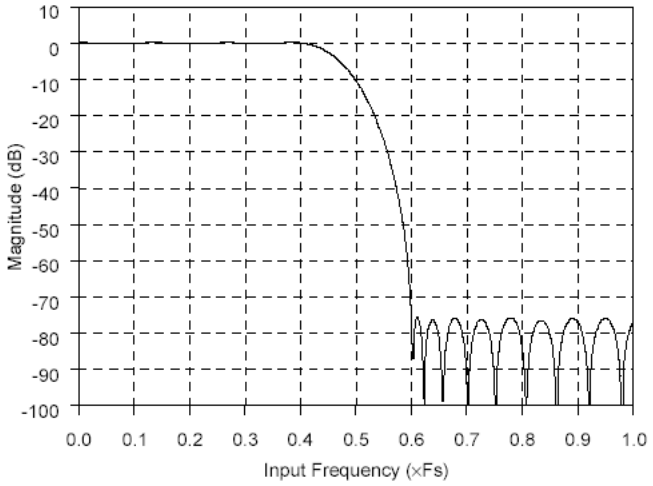


Figur 8.1: PC til PC test, der efterviser at line-out på SiS 7018 ikke er pålidelig til test af systemet, da der sker en kraftig dæmpning af signalstyrken ved en frekvens på over 7 kHz.

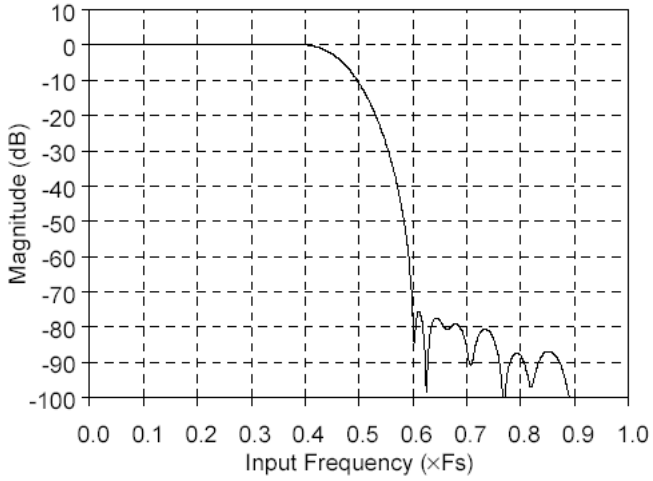
Efterfølgende er der udfærdiget en test med talkthrough kode for at eftervise at systemet har en frekvensrespons, der ligner PC til PC testen, denne kan ses på figur 8.4. I testen skal CS4218'erens egen frekvensrespons i ADC-leddet såvel som DAC-leddet også tages i betragtning. Jævnfør figurerne 8.2 og 8.3 ses det, at signalet ved den halve samplerate har en dæmpning på 10 dB for både ADC og DAC [11]. Testen viser at systemet har en pæn frekvensrespons med talkthrough, når CS4218's egendæmpning tages i betragtning.

8.2 Equalizertest

Til at teste om equalizeren har den rette dæmpning i de forskellige frekvensområder, er der ligesom i den generelle systemtest, anvendt white noise til at afbilde frekvensresponsen. Testen er delt op i tre faser, tilsvarerne de tre filtre der udgør equalizeren; lavpas-dæmpning, båndpas-dæmpning og højpas-dæmpning. Knækfrekvenserne og båndbredden er tilfældigt udvalgt, da muligheden for dette er et stillet krav.



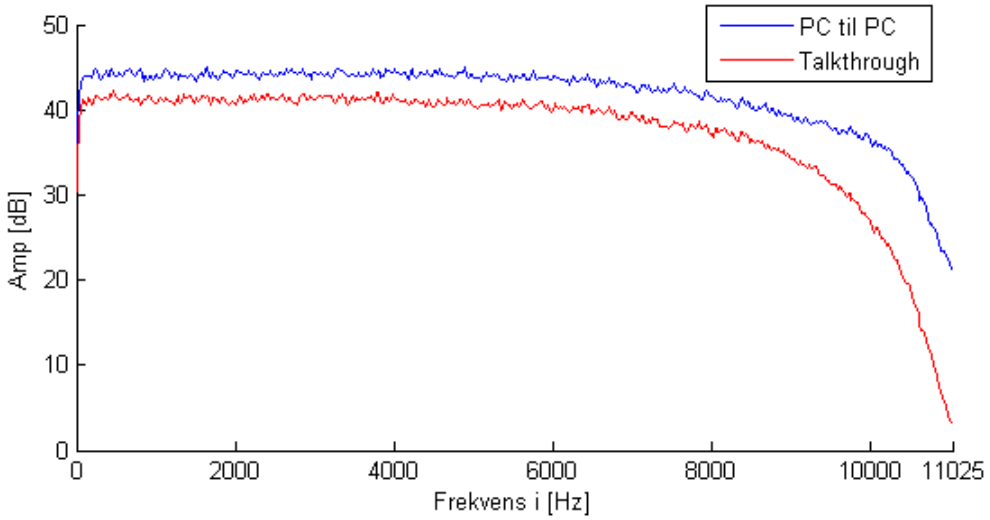
Figur 8.2: ADC'ens frekvensrespons, der viser en dæmpning på 10 dB ved en frekvens på 11.025 Hz (fra [11]).



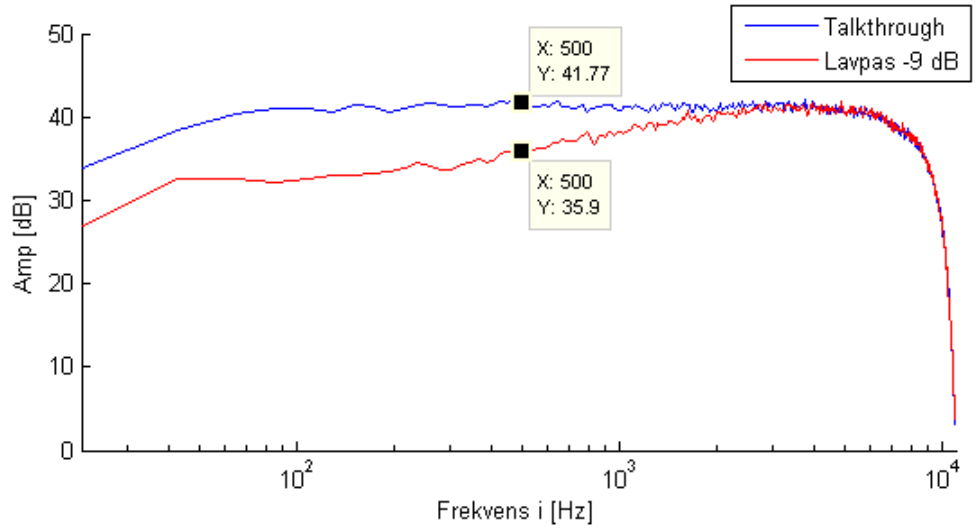
Figur 8.3: DAC'ens frekvensrespons, der ligesom ADC'en viser en dæmpning på 10 dB ved en frekvens på 11.025 Hz (fra [11]).

8.2.1 Lavpasfilter

Lavpasfilteret testes for at have den den rette dæmpning ved en given knækfrekvens, ved at indsætte værdier med et gain på -9 dB og med en knækfrekvens på 500 Hz. Resultatet som ses på figur 8.5 viser at det ønskede resultat med -6 dB gain ved knækfrekvensen på 500 Hz er opnået. Afvigelsen tilskrives white noise præcisionen.



Figur 8.4: Talkthrough testen viser at systemet har en pæn frekvensrespons med en lille dæmpning over hele frekvensspektret, udover CS4218's egendæmpning i de høje frekvenser.

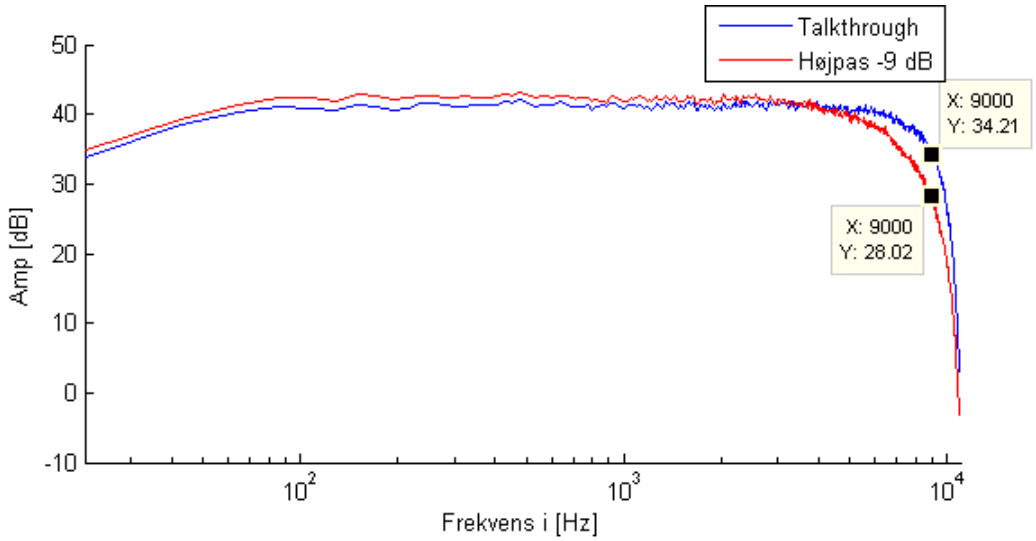


Figur 8.5: Lavpastesten viser at lavpasfilteret dæmper signalet de ønskede 6 dB ved knækfrekvensen på 500 Hz.

8.2.2 Højpasfilter

Dæmpningen der er testet for lavpasfilteret, er defineret som værende -9 dB gain, med en knækfrekvens på 9 kHz. Testresultatet ses på figur 8.6, og er ligesom lav-

pasfilteret meget tilfredsstillende, med en dæmpning på 6 dB i knækfrekvensen på 9 kHz som forventet.



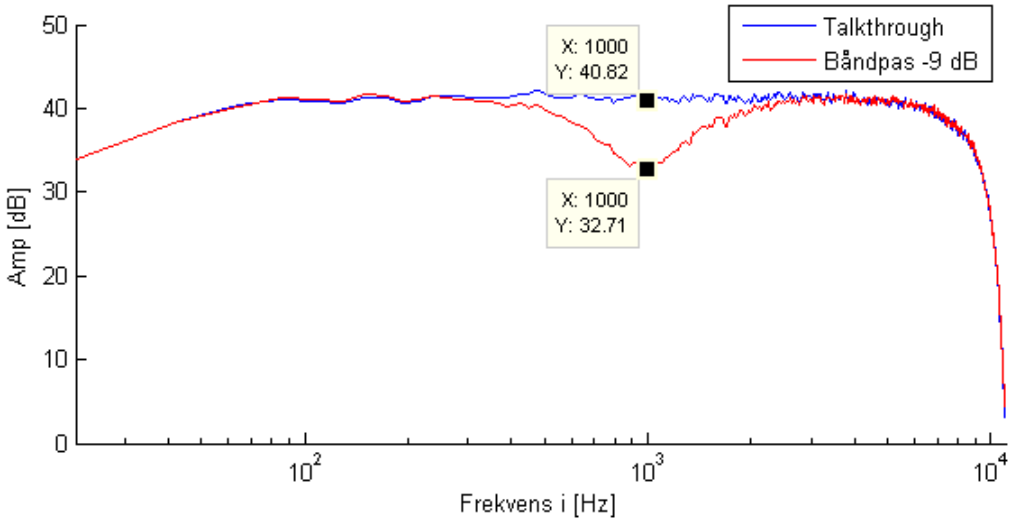
Figur 8.6: Højpastesten viser at filteret dæmper signalet de ønskede 6 dB ved knækfrekvensen på 9 kHz.

8.2.3 Båndpasfilter

Båndpasfiltertesten er udført med en dæmpning på 9 dB med en centerfrekvens på 1 kHz, og en båndbredde på 500 Hz. Båndpasfilterets resultat er i modsætning til de to foregående filtertests ikke helt så præcis som ønsket. Centerfrekvensen og båndbredden passer på det forventede, men dæmpningen i centerfrekvensen er 1 dB fra det ønskede resultat som ses på figur 8.7. Om dette kan tilskrives white noise afvigelsen er uvist. Testen betegnes dog stadigt som en succes.

8.2.4 Uformel lyttetest

Til den uformelle lyttetest, er forskellige stykker musik i genrer lige fra klassisk til techno, for at få så mange frekvensområder repræsenteret i testen som muligt. Lyttetestene er udført filter for filter, og til sidst en test med alle tre filtre aktiveret. Resultatet af denne test er overvejende positiv. Lyden menes at være god, og det er tydeligt at høre equalizerens effekt. Ydermere tilføjes det at equalizer-effekterne



Figur 8.7: Båndpasfiltertesten viser et resultat, der afviger med 1 dB fra den ønskede dæmpning på 9 dB. Centerfrekvensen på 1 kHz, og båndbredden på 500 Hz, passer dog.

karakteriseres som lydende ”rigtige”.
Det bemærkes at bluetooth modulet støjer i systemet, når der er oprettet forbindelse mellem klient-PC’en og boardet, denne støj beskrives som en svag hyletone med ripples.
Det samme gør sig gældende når ”JTAG - MSP430 Flash Emulation Tool” modulet[1] sidder på boardet. Her er støjen dog karakteriseret ved at sende inputsignalet meget svagt igennem systemet.

[1]Modul der anvendes når mikroprocessorens ROM skal flashes

Kapitel 9

Konklusion

I kravspecifikationen blev en række krav og succeskriterier opstillet til at udvikle systemet. Disse krav er udarbejdet med det formål at de skulle være test- og målbare. Kravene er efterlevet så vidt muligt, med en betydelig undtagelse. De udførte tests har dog været positive, og resultatet er som følger:

Hardware Hardwaren virker efter hensigten, og alle kravene til den er opfyldt. Ved talkthrough tests på boardet har det givet et pænt frekvensrespons unden videre påvirkning af signalet. Den serielle kommunikation mellem effektboard og klient-PC'en fungerer uden problemer, men bluetooth modulet har vist sig at støje i resten af kredsløbet, når den serielle forbindelse er oprettet.

Effekter Equalizeren fungerer efter hensigten, og har jvf. den uformelle lyttetest en god lyd. Den uformelle lyttetest understøttes godt af frekvensresponstesten, der viser at equalizeren fungerer efter hensigten. En iagtagelse ved equalizeren, er vedr. båndpasfilterets dæmpning i centerfrekvensen, som har en 1 dB afvigelse fra det ventede resultat. Dette antages at være grundet testmetoden, da centerfrekvensen og båndbredden stemmer overens med det ventede.

Compressoren har ikke vist sig funktionsdygtig. Der er gjort en række overvejlser omkr. implementationsmuligheder, men realiseringen har vist sig vanskeligere end ventet.

Brugergrænseflader Kravene stillet til brugergrænsefladen er i det store hele opfyldt. Det er muligt at oprette og afbryde kommunikationen over bluetooth fra klientprogrammet, der er en dialogboks der giver brugeren feedback, og det er muligt at indstille de forskellige parametre til effekterne. Det eneste krav der ikke er opfyldt, er til og fravalg af effekter. Denne funktionalitet er til rådighed på klient softwaren, men er slået fra på effektboardet, da der ikke er flere effekter at til eller fravælge.

Opbygningen af effektboardet har overvejende været en succes, der er dog visse mangler i systemet. Systemet sampler signalet korrekt, og fungerer efter hensigten i realtid. Compressoren der skulle implementeres, er imidlertid ikke funktionel, og er derfor udeladt fra produktet. Ses der bort fra den manglende effekt, er der ikke problemer med systemet.

Litteratur

[1] Steven W. Smith. *The Scientists and Engineers Guide to Digital Signal Processing*. California Technical Publishing, 1997.

[2] Frederik Nebeker. Digital signal processing and the rise of consumer electronics. *IEEE History of Technology*, juli 2002.

[3] Stephen Biering-Sørensen et al. *Struktureret Program-Udvikling*. Ingeniøren | bøger, 1999.

[4] Sophocles J. Orfanidis. *Introduction to signal processing*. Prentice-Hall, 1996.

[5] Dimitris Manolakis John G. Proakis. *Digital Signal Processing: Principles, Algorithms and Applications*. Prentice-Hall, 1996.

[6] Harmony Central. Harmony central, oktober 2004. http://www.harmony-central.com/Effects/effects-explained.html.

[7] Texas Instruments. Msp430f149 datablad, Juli 2003. http://www.control.aau.dk/~hr/d3_e04/datablad.pdf.

[8] Maxim. Max531 +5v low-power, voltage-output, serial 12-bit dac, februar 1997. http://komponenten.ies.aau.dk/fileadmin/komponenten/Data_Sheet/ ADC + DAC/MAX531-MAX539.pdf.

[9] Texas Instruments. Msp430x1xx family user's guide, 2004. http://www.control.aau.dk/~hr/d3_e04/users-guide.pdf.

[10] Jes Andersen et al. *Digital Signalbehandling: Design og konstruktion af e-qualizer og flanger.* Aalborg Universitet, 2003.

[11] Crystal. Cs4218 16 bit stereo audio codec, september 1996. http://komponenten.ies.aau.dk/fileadmin/komponenten/Data_Sheet/ ADC + DAC/CS4218.pdf.

[12] Philips. datablad for philips 74hc595; 74hct595,8-bit serial-in, serial or parallel-out shift register with output latches; 3-state, Juni 2003. http://komponenten.ies.aau.dk/fileadmin/komponenten/Data_Sheet/MOS-TTL/hc/74HC595.pdf.pdf.

[13] Texas instruments. datablad for texas instruments'; 8 bit shift register with input latch, Januar 1981. http://komponenten.ies.aau.dk/fileadmin/komponenten/Data_Sheet/MOS-TTL/ls/74LS597.pdf.pdf.

[14] Claus Walther. Hjælp til design af oscillator, november 2004. http://komponenten.ies.aau.dk/fileadmin/komponenten/Data_Sheet/Application/OCL.pdf.

[15] Datamaters arkitektur og programmering Forelæsnings materiale udleveret af Henrik Rasmussen. Asynchromous serial data transmission.

[16] Merlin Systems Corp. Ltd. Bluecom - embedded bluetooth module, datasheet, v2.0a, 2002. http://www.control.auc.dk/~hr/d3_e04/bluecomembedded.pdf.

[17] Robert Piché. Low-order vs. high-order. http://virtual.cvut.cz/odl/partners/tut/unit2/node9.html.

[18] Charles Randall Yates. Randy's dsp page, november 2004. http://home.earthlink.net/ yatescr/fp.pdf.

[19] Raymond McKendall. Rays notes, november 2004. http://www.cis.upenn.edu/.

[20] Wikipedia. White noise, december 2004. http://en.wikipedia.org/wiki/White_noise.

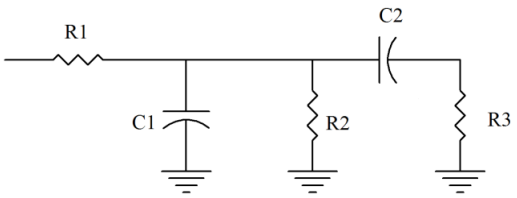
[21] David E.Johnson et al. *Electric Circuit Analysis*. Prentice Hall, 1997.

[22] Adel S.Sedra Kenneth C.Smith. *Microelectronic Circuits*. Oxford University Press, 1998.

Bilag A

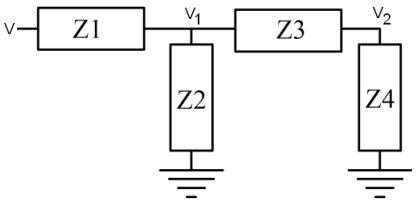
Analogt præfilter

I dette appendiks beskrives beregninger samt metoder der ligger til grund for opbygningen af et analogt filter i forbindelse med afsnit 5.3. For at kunne bestemme de enkelte komponenter som skaber filtret, tegnes der en anden ordens bandpass filter se figur A.1.



Figur A.1: Illustrerer en lowpass filter samt en highpass filter der er i serie.

Fra figur A.1 fremgår det, at C1 og R2 sidder parallelt. Disse kan omskrives til impedanser, derved kan C1 og R2 adderes, se figur A.2.



Figur A.2: Illustrerer en bandpass filter, bestående af impedanser,

hvor:
Z1 = R_1.

Z2 = $\frac{R_2}{sC_1R_2+1}$.
Z3 = $\frac{1}{sC_2}$.
Z4 = R_3.

Der anvendes spændingsdeling:

$$V_1 = V\frac{Z2}{Z1+Z2}. \tag{A.1}$$

$$V_2 = V_1\frac{Z4}{Z3+Z4} = \frac{Z2}{Z1+Z2}\frac{Z4}{Z3+Z4}V \tag{A.2}$$

$$= \frac{Z1Z4}{Z1Z3+Z1Z4+Z2Z3+Z2Z4}V. \tag{A.3}$$

Det reducerede udtryk for V_2 kan skrives på følgende måde:

$$V_2 = V\frac{\frac{R_2R_3}{sC_1R_2+1}}{\frac{R_1}{sC_2}+R_1R_3+\frac{R_2}{sC_1R_2+1}\frac{1}{sC_2}+\frac{R_2}{sC_1R_2+1}R_3}$$

$$= V\frac{sR_2R_3C_2}{S^2R_1R_2^2C_1C_2+R_2R_3C_2s+R_2+C_1R_2R_1}. \tag{A.4}$$

Overføringsfunktionen er udtrykt ved [21]:

$$H(s) = \frac{V_{out}(s)}{Vin(s)} \tag{A.5}$$

Ud fra disse opløsninger kan, overføringsfunktionen opstilles. Ydermere forkortes der med $(R_1R_2^2C_1C_2)$ i alle lederne. Dermed kommer overførings funktion til se således ud:

$$H(s) = \frac{s\frac{R_3}{R_1R_2C_1}}{s^2+s\frac{R_3}{R_1R_2C_1}+\frac{1+R_1C_1}{R_1R_2C_1C_2}}. \tag{A.6}$$

Da overføringsfunktionen i dette tilfælde gælder for en andenordens bandpass filter, skal følgende være opfyldt [22]:

$$H(s) = \frac{a_1(s)}{s^2+s\frac{\omega_0}{Q}+\omega_o^2}. \tag{A.7}$$

$$Center-frekvensgain = \frac{a_1Q}{w_0}. \tag{A.8}$$

Af formlen A.8 fremgår det, at for at finde center frekvens gainet, er det nødvendigt at kende komponenterne a_1, Q og w_0. For dette sættes funktionen A.6 og A.7 overfor hinanden, hvorfra tre ligninger med dertilhørende antal ubekendte kan opskrives.

$$s\frac{w_0}{Q} = s\frac{R_3}{R_1R_2C_1} \quad \Rightarrow \tag{A.9}$$

$$Q = \frac{R_1R_2C_1}{R_3}. \tag{A.10}$$

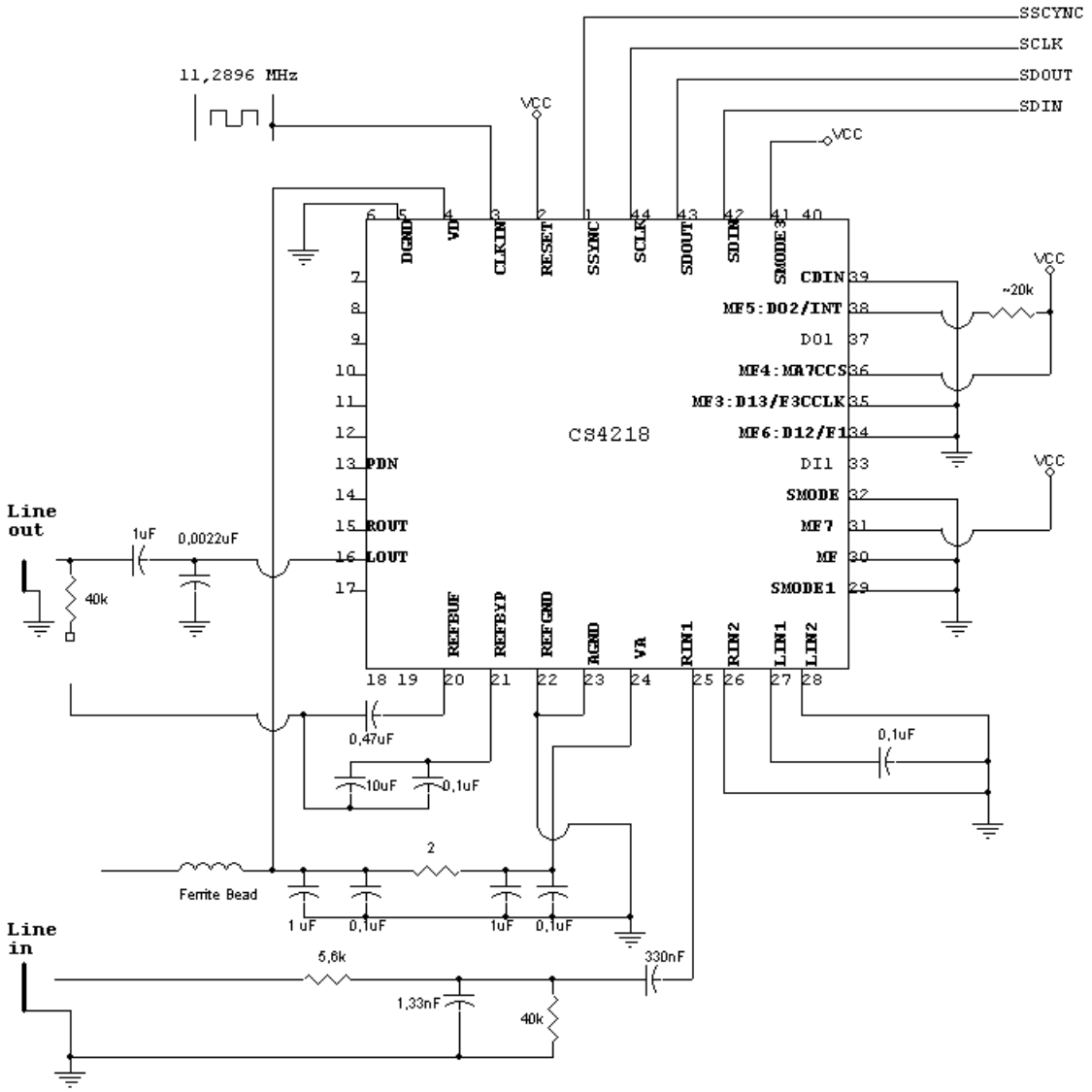
$$w_o = \sqrt{\frac{1 + R_1C_1}{R_1R_2C_1C_2}}. \tag{A.11}$$

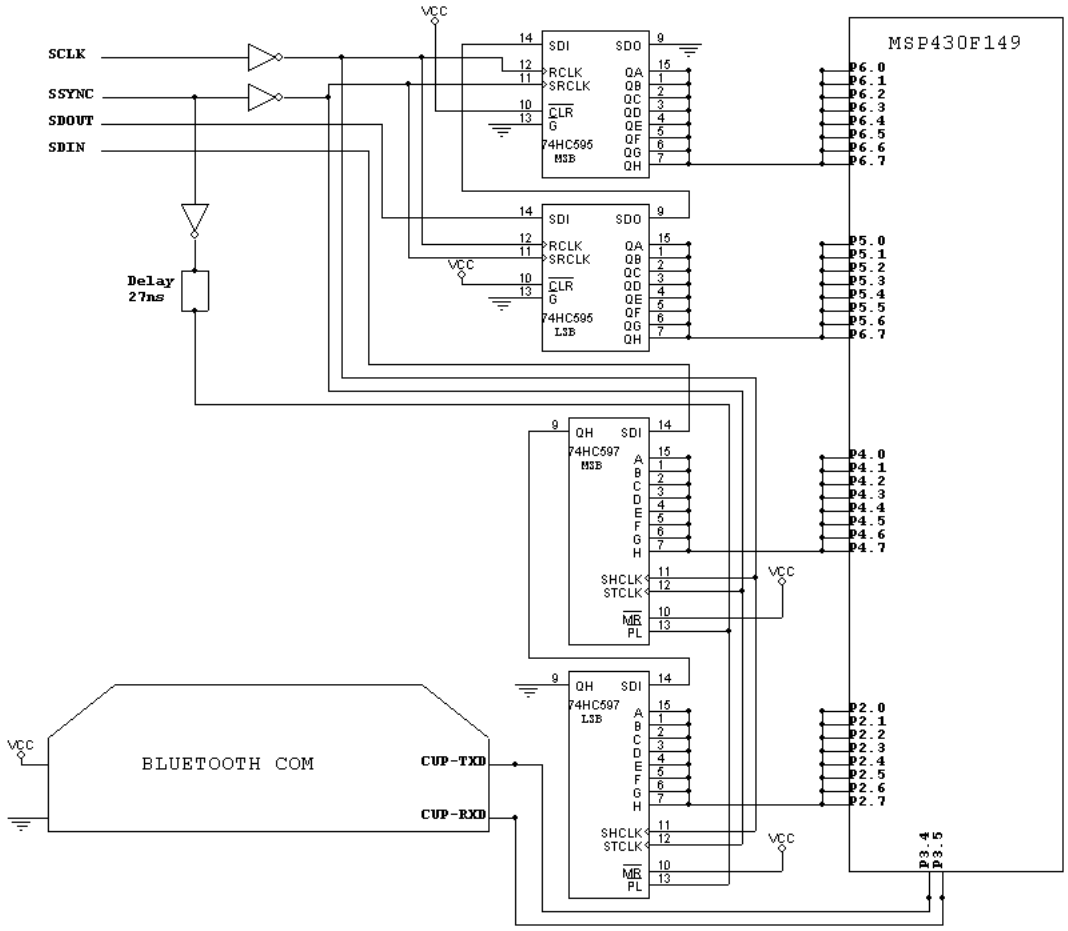
$$a_1 = \frac{R_3}{R_1R_2C_1}. \tag{A.12}$$

For at kunne løse disse tre ligninger, sættes komponenten $R_1 \ og \ C_2$ til hhv. 5600 og 330 nF. Disse to komponentstørrelser er valgt ud fra databladet for CS4218. De resterende komponenter $R_2 \ og \ C_1$ vælges ud fra en tilnærmelse af funktionen A.8 hvor center-frekvens gainet sættes til 1.

Bilag B

Ekstern kredsløb


SSCYNC
SCLK
SDOUT
SDIN
11,2896 MHz
VCC
VCC
VCC
VCC
~20k
CS4218
DGND
VD
CLKIN
RESET
SSYNC
SCLK
SDOUT
SDIN
SMODE3
CDIN
MF5:DO2/INT
DO1
MF4:MA7CCS
MF3:D13/F3CCLK
MF6:D12/F1
DI1
SMODE
MF7
MF
SMODE1
PDN
ROUT
LOUT
REFBUF
REFBYP
REFGND
AGND
VA
RIN1
RIN2
LIN1
LIN2
Line out
1uF
0,0022uF
40k
0,47uF
10uF
0,1uF
0,1uF
2
Ferrite Bead
1 uF
0,1uF
1uF
0,1uF
Line in
5,6k
330nF
1,33nF
40k


SCLK
SSYNC
SDOUT
SDIN
Delay
27ns
VCC
SDI
SDO
RCLK
SRCLK
CLR
G
74HC595
MSB
LSB
QA
QB
QC
QD
QE
QF
QG
QH
MSP430F149
P6.0
P6.1
P6.2
P6.3
P6.4
P6.5
P6.6
P6.7
P5.0
P5.1
P5.2
P5.3
P5.4
P5.5
P5.6
P5.7
QH
SDI
74HC597
A
B
C
D
E
F
G
H
SHCLK
STCLK
MR
PL
P4.0
P4.1
P4.2
P4.3
P4.4
P4.5
P4.6
P4.7
P2.0
P2.1
P2.2
P2.3
P2.4
P2.5
P2.6
P2.7
P3.4
P3.5
BLUETOOTH COM
CUP-TXD
CUP-RXD

Bilag C

Effekt og decibel

Når et filters påvirkning af et signal skal analyseres med henblik på f.eks. knækfrekvens, er det nødvendigt at se på amlitudekarakteristikken. Ved knækfrekvenser er vi også interesseret i et signals effekt (P) som for et signal $H(\omega)$ er givet ved $|H(\omega)|^2$ som angives i decibel (dB).
Et signals amplitudeændring udtryk i dB er givet ved [1]:

$$\text{gain } [\text{dB}] = 20 \log_{10}\left(\frac{a_{out}}{a_{in}}\right) [\text{dB}] = 10 \log_{10}\left(\frac{P_{out}}{P_{in}}\right) [\text{dB}]. \tag{C.1}$$

En typisk knækfrekvens som er defineret ved en 3dB dæmpning af et signal, giver derfor en halvering af styrken som vist ved følgende:

$$-3\,\text{dB} = 10 \log_{10}\left(\frac{P_{out}}{P_{in}}\right) [\text{dB}] \Leftrightarrow \frac{P_{out}}{P_{in}} = \log_{10}^{-1}\left(\frac{-3}{10}\right) = \frac{1}{2}. \tag{C.2}$$

Bilag D

Kildekode

D.1 Klient-PC software

main.h

```
//---------------------------------------------------------------------------
#ifndef mainH
#define mainH
//---------------------------------------------------------------------------
#include <Classes.hpp>
#include <Controls.hpp>
#include <StdCtrls.hpp>
#include <Forms.hpp>
#include <Menus.hpp>
//---------------------------------------------------------------------------
class TForm1 : public TForm
{
__published: // IDE-managed Components
        TGroupBox *eqalizer_;
        TGroupBox *comp_exp_;
        TLabel *Label4;
        TLabel *Label5;
        TEdit *f_c_lp;
        TEdit *gain_lp;
        TLabel *Label7;
        TLabel *Label8;
        TEdit *rho_in;
        TEdit *lambda_in;
        TCheckBox *CheckBox1;
        TCheckBox *compcheckbox_;
        TGroupBox *seriel_send_;
        TButton *sendkoeff_;
        TGroupBox *debuginfo;
        TMemo *Memo1;
        TLabel *Lowpass_shelv;
        TLabel *Highpass_shelv;
        TEdit *f_c_hp;
        TLabel *gain;
        TEdit *gain_hp;
        TEdit *f0_mid;
        TLabel *Label1;
        TLabel *f_c;
        TLabel *Midpass_shelv;
        TLabel *Label2;
        TEdit *gain_mid;
        TEdit *delta_f;
        TLabel *Baandbredde;
        TCheckBox *eqcheckbox_;
        TGroupBox *serielforb_;
```

```
        TButton *opretforb_;
        TButton *afbrydforb_;
        TLabel *Label3;
        TEdit *comport;
        TEdit *c0_in;
        TLabel *Label6;
        void __fastcall sendkoeff_Click(TObject *Sender);
        void __fastcall compcheckbox_Click(TObject *Sender);
        void __fastcall eqcheckbox_Click(TObject *Sender);
        void __fastcall opretforb_Click(TObject *Sender);
        void __fastcall afbrydforb_Click(TObject *Sender);
private: // User declarations
        void lp_shelf(void);
        void hp_shelf(void);
        void mid_shelf(void);
        double gainb_calc(double G0, double G, double gain, double width);
        void serial_send_eq(double *koefA, double *koefB, int elements);
        void serial_send_comp(void);
        void serial_recive(void);
        void serial_init_eq(char* equ);
        double LP_A[2];
double LP_B[2];
        double HP_A[2];
double HP_B[2];
        double MID_A[3];
double MID_B[3];
        HANDLE hPort;   /* port handle */
        int cnt;
public: // User declarations
        __fastcall TForm1(TComponent* Owner);
};
//---------------------------------------------------------------------------
extern PACKAGE TForm1 *Form1;
//---------------------------------------------------------------------------
#endif
```

main.cpp

```
//---------------------------------------------------------------------------
#include <vcl.h>
#include <math.h>
#pragma hdrstop

#include "main.h"
#include <windows.h>
#include <stdio.h>
#include <string.h>
//---------------------------------------------------------------------------
#pragma package(smart_init)
#pragma resource "*.dfm"
TForm1 *Form1;
//---------------------------------------------------------------------------
__fastcall TForm1::TForm1(TComponent* Owner)
        : TForm(Owner)
{
}
//---------------------------------------------------------------------------
/* funktion  for knappen 'send' */
void __fastcall TForm1::sendkoeff_Click(TObject *Sender)
{
        int i;
        /* init tegn til switchcase på boardet */
        char *eq = "a";
        char *comp = "b";

        /*funktioner til udregning af filterkoefficienter */
        lp_shelf();
        hp_shelf();
        mid_shelf();

        /* afsender filterkoefficienter */
```

```
        if(eqcheckbox_->Checked)
        {
                serial_init_eq(eq);
                serial_send_eq(LP_A, LP_B, sizeof(LP_A)/sizeof(LP_A[0]));
                serial_send_eq(HP_A, HP_B, sizeof(HP_A)/sizeof(HP_A[0]));
                serial_send_eq(MID_A, MID_B, sizeof(MID_A)/sizeof(MID_A[0]));
        }
        if(compcheckbox_->Checked)
        {
                serial_send_comp();
        }

        Memo1->Lines->Add("Koefficinter afsendt");
}

//---------------------------------------------------------------------------

/* koeficientberegning til lowpass shelvingfilter*/
void TForm1::lp_shelf(void)
{
        /* henter indtastede værdier fra brugerfladen */
double fc = f_c_lp->Text.ToDouble();           /* cutoff freq */
        double gain = gain_lp->Text.ToDouble();        /* gain (dB) - nul dæmpning ved 0 -> da G = 1 = G0 */
double daemp = -10.0*log10(.5);          /* hvor høj dæmpning/stigning der sker før cutoff nåes (dB) */
        double fs = 22050.0;            /* samplingsrate*/
double beta;
double G0 = 1.0;                /* reference gain */
        double Gc;                      /* cutoff-dampning*/
        double G;                       /* filtergain */
double pi = 4.0*atan(1.0);
double dw = (2.0*pi*fc)/fs; /* udregner den digitale delta-frek */

        if(gain != 0)
        {
                G = pow(10,gain/20);
                Gc = gainb_calc(G0, G, gain, daemp);
                beta = tan(dw/2.0) * sqrt(fabs(pow(Gc,2.0) - pow(G0,2)))/sqrt(fabs(pow(G,2.0) - pow(Gc,2.0)));
                LP_A[0] = 1.0;
                LP_A[1] = -(1-beta)/(1+beta);
                LP_B[0] = (G0 + (G*beta))/(1+beta);
                LP_B[1] = -(G0 - G*beta)/(1+beta);
        }
        else
        {
                LP_B[1] = LP_A[1] = LP_A[0] = 0;
                LP_B[0] = 1;

        }
}
//---------------------------------------------------------------------------

/* koeficientberegning til highpass shelvingfilter*/
void TForm1::hp_shelf(void)
{
        /* henter indtastede værdier fra brugerfladen */
        double fc = f_c_hp->Text.ToDouble(); /* cutoff freq */
        double gain = gain_hp->Text.ToDouble(); /* gain (dB) - nul dæmpning ved 0 -> da G = 1 = G0 */
double daemp = -10.0*log10(.5);                 /* hvor høj dæmpning/stigning der sker før cutoff nåes (dB) */
        double fs = 22050.0;
double beta;
double G0 = 1.0; /* reference gain */
        double Gc;
        double G;
double pi = 4.0*atan(1.0);
double dw = (2.0*pi*fc)/fs; /* udregner den digitale delta-frek */


        if(gain != 0)
        {
                G = pow(10,gain/20);
                Gc = gainb_calc(G0, G, gain, daemp);
                beta = tan((pi-dw)/2.0) * sqrt((fabs(pow(Gc,2.0) - pow(G0,2.0))/fabs(pow(G,2.0) - pow(Gc,2.0))));
```

```
                HP_A[0] = 1.0;
                HP_A[1] = (1-beta)/(1+beta);
                HP_B[0] = (G0 + (G*beta)) / (1+beta);
                HP_B[1] = (G0 - G*beta) / (1+beta);
        }
        else
        {
                HP_B[1] = HP_A[1] = HP_A[0] = 0;
                HP_B[0] = 1;
        }
}
//---------------------------------------------------------------------------

/* koeficientberegning til mellempass shelvingfilter*/
void TForm1::mid_shelf(void)
{
        /* henter indtastede værdier fra brugerfladen */
double f0 = f0_mid->Text.ToDouble();   /* centerfrekvens */
double deltaf = delta_f->Text.ToDouble(); /* deltafrek - bredden på peak/notchen */
        double gain = gain_mid->Text.ToDouble();  /* gain (dB) - nul dæmpning ved 0 -> da G = 1 = G0 */
double daemp = -10.0*log10(.5);           /* hvor mange dB der dæmpes indenfor intervallet df */
        double beta;
double G0 = 1.0;         /* refecegain (dB) */
double Gb;
double G;
double pi = 4.0*atan(1.0);
double fs = 22050.0;
double w0 = (2.0*pi*f0)/fs; /* udregner den dititale centerfrekvens */
double dw = (2.0*pi*deltaf)/fs;

        if(gain != 0)
        {
                G = pow(10,gain/20.0);
                Gb = gainb_calc(G0, G, gain, daemp);
                /* koefficientberegning */
                beta = tan(dw/2.0) * sqrt(fabs(pow(Gb,2.0) - pow(G0,2)))/sqrt(fabs(pow(G,2.0) - pow(Gb,2.0)));
                MID_A[0] = 1;
                MID_A[1] = (-2*cos(w0))/(1+beta);
                MID_A[2] = (1-beta)/(1+beta);
                MID_B[0] = (G0 + G*beta)/(1+beta);
                MID_B[1] = -2*G0*cos(w0)/(1+beta);
                MID_B[2] = (G0 - G*beta)/(1+beta);
        }
        else
        {
                MID_B[1] = MID_B[2] = MID_A[2] = MID_A[1] = MID_A[0] = 0;
                MID_B[0] = 1;
        }
}
//---------------------------------------------------------------------------

/* beregning af niveauet for cutoff-frekvensen og båndbredden */
double TForm1::gainb_calc(double G0, double G, double gain, double daemp)
{
        if(gain < daemp && gain > -daemp && gain != 0)
                return sqrt((pow(G0,2.0)+pow(G,2))/2.0); /*hvis gain er er mindre end daemp */
        else if(gain > 0 || gain == 0)
                return pow(10.0,(gain-daemp)/20.0);  /*hvis gain er == 0 eller gain > 0 */
        else
                return pow(10.0,(gain+daemp)/20.0); /*hvis gain < 0 */
}
//---------------------------------------------------------------------------

/* afsendelse af equalizerkoefficienter */
void TForm1::serial_send_eq(double *koefA, double *koefB, int elements)
{
        int i,s;
        signed short send;
        char msg_out[15];            /* besked der skal skriver til port */
        int n_msg_out;            /* tæller bytes i besked til port */
        unsigned long n_out_out;     /* tæller bytes der er skrevet til porten */
        int ok;                 /* retur-kode fra funktionskald */
```

```
        for(i=0;i<elements;i++)       /* sender først B-koefficienter*/
        {
                send = short((koefB[i]/pow(2.0,-14.0))+.5);
                sprintf(msg_out, "%d\r", send);
                n_msg_out = strlen(msg_out);
                WriteFile(hPort, msg_out, n_msg_out, &n_out_out, NULL);
        }

        for(i=1;i<elements;i++)         /* sender A-koefficienter*/
        {
                send = short((koefA[i]/pow(2.0,-14.0))+.5);
                sprintf(msg_out, "%d\r", send);
                n_msg_out = strlen(msg_out);
                WriteFile(hPort, msg_out, n_msg_out, &n_out_out, NULL);
        }
}
//---------------------------------------------------------------------------


void TForm1::serial_send_comp(void)
{
        double rho = rho_in->Text.ToDouble();
        double lambda = lambda_in->Text.ToDouble();
        double c0 = c0_in->Text.ToDouble();
        short signed rho_fp, lambda_fp, c0_fp;
        char *msg_out;            /* besked der skal skriver til port */
        int n_msg_out;            /* tæller bytes i besked til port */
        unsigned long n_out_out;     /* tæller bytes der er skrevet til porten */
        int ok;                 /* retur-kode fra funktionskald */

        rho_fp = short((rho/pow(2.0,-15.0))+.5);
        lambda_fp = short((lambda/pow(2.0,-15.0))+.5);
        c0_fp = short((c0/pow(2.0,-15.0))+.5);
        sprintf(msg_out, "%d\r", rho_fp);
}
//---------------------------------------------------------------------------


/*opret seriel forbindelse*/
void __fastcall TForm1::opretforb_Click(TObject *Sender)
{
        int ok;
        int comnr = comport->Text.ToInt(); /* indlæser port.nr. fra brugerflade*/
        char port[2];

        sprintf(port, "com%d", comnr);     /* konverterer port.nr. til streng*/
        hPort = CreateFile(port, GENERIC_ALL, 0, NULL, OPEN_EXISTING, 0, NULL); /* åber port som en filehandler*/
        if(hPort == INVALID_HANDLE_VALUE)
        {
                Memo1->Lines->Add("Fejl ved åbning af port");
        }
        else
                Memo1->Lines->Add("Forbindelse oprettet");
}
//---------------------------------------------------------------------------

/* afbryd seriel forbindelse*/
void __fastcall TForm1::afbrydforb_Click(TObject *Sender)
{
        int ok;

        ok = CloseHandle(hPort);        /* lukker porten */
        if(!ok)
        {
                Memo1->Lines->Add("Fejl ved lukning af porten");
        }
        else
                Memo1->Lines->Add("Forbindelse afbrudt");
}
//---------------------------------------------------------------------------
```

```
/* init til afsendelse af equalizerkoefficinter*/
void TForm1::serial_init_eq(char *equ)
{
        char *msg_out;            /* besked der skal skriver til port */
        int n_msg_out;            /* tæller bytes i besked til port */
        unsigned long n_out_out;     /* tæller bytes der er skrevet til porten */
        int ok;                 /* retur-kode fra funktionskald */

        msg_out = equ;
        n_msg_out = strlen(msg_out);
        ok = WriteFile(hPort, msg_out, n_msg_out, &n_out_out, NULL);
        if(!ok)
        {
                Memo1->Lines->Add("Fejl i equalizer init");
        }
}

/* checkbox-valg til equalizer*/
void __fastcall TForm1::eqcheckbox_Click(TObject *Sender)
{
        if(eqcheckbox_->Checked)
        {
                Memo1->Lines->Add("Equalizer valgt");
        }
        else
                Memo1->Lines->Add("Equalizer fravalgt");
}
//---------------------------------------------------------------------------

/* checkbox-valg til compressor */
void __fastcall TForm1::compcheckbox_Click(TObject *Sender)
{
        if(compcheckbox_->Checked)
        {
                Memo1->Lines->Add("Compressor valgt");
        }
        else
                Memo1->Lines->Add("Compressor fravalgt");
}
```

D.2 Mikroprocessor software

main.c

```
#include <MSP430x14x.h>
#include <stdio.h>
#include <stdlib.h>
#include "release/basic_clock_module.h"
#include "release/serial.h"
#include "release/port_modules.h"
#include "release/delay.h"

#define LP_LENGTH 3 //  antallet af koefficienter i differenslignингerne
#define HP_LENGTH 3 //  11 multiplikationer = 11 koefficienter
#define BP_LENGTH 5

// Serielkommunikation initialisering
unsigned int control;
unsigned int cnt;
char chararray[7];
char ch;

// Equalizer initialisering
signed int LP_koeff[LP_LENGTH];
signed int HP_koeff[HP_LENGTH];
signed int BP_koeff[BP_LENGTH];

// Sample initialisering
signed int x[2];
signed int ylp[2];
signed int yhp[3];
signed int ybp[3];

void Delay(unsigned int i) //software delay
{
   do{
      i--;
   }while(i != 0);
}

void main(void)
{
//Modul initialisering
  basic_clock_init();
  serial_init();
  port_interrupt_init();
  lower_dac_init();
  reset_init();
  upper_dac_init();
  lower_adc_init();
  upper_adc_init();

//Startup tid til ADC (RESET tid på >50ms ~ 400.000 clockcycles)
  P3OUT &= ~0x80;      //Sæt port lav
  Delay(0xFFFF);
  Delay(0xFFFF);
  Delay(0xFFFF);
  P3OUT |= 0x80;     //Sæt port høj

//Default koefficienter (Talkthrough)
  LP_koeff[0] = 16384; //1 m. (1,14) repræsenation
  LP_koeff[1] = 0;
  LP_koeff[2] = 0;

  HP_koeff[0] = 16384;
  HP_koeff[1] = 0;
  HP_koeff[2] = 0;

  BP_koeff[0] = 16384;
  BP_koeff[1] = 0;
  BP_koeff[2] = 0;
```

```
  BP_koeff[3] = 0;
  BP_koeff[4] = 0;

  //Nulstilling af koefficienter
  for(cnt = 0; cnt < 2; cnt++)
  {
    x[cnt]   = 0x8000;
    ylp[cnt] = 0x8000;
  }
  for(cnt = 0; cnt < 3; cnt++)
  {
    yhp[cnt] = 0x8000;
    ybp[cnt] = 0x8000;
  }

  _EINT();
  while(1)
  {
  do
    {
      if (KeyHit()) //Ved keyhit slås audiointerrupts fra
      {
         ch=GetChar();
         switch (ch)
         {
            case 'a':
              //Modtag værdier fra klient
              //Lavpas shelving filter
              for(cnt = 0; cnt < LP_LENGTH; cnt++){
                scanf("%s",chararray);
                LP_koeff[cnt] = atoi(chararray);
              }
              //Højpas shelving filter
              for(cnt = 0; cnt < HP_LENGTH; cnt++){
                scanf("%s",chararray);
                HP_koeff[cnt] = atoi(chararray);
              }
              //Båndpas filter
              for(cnt = 0; cnt < BP_LENGTH; cnt++){
                scanf("%s",chararray);
                BP_koeff[cnt] = atoi(chararray);
              }
              //Nulstilling af samples
               for(cnt = 0; cnt < 2; cnt++)
               {
               x[cnt]   = 0x8000;
               ylp[cnt] = 0x8000;
               }
               for(cnt = 0; cnt < 3; cnt++)
              {
              yhp[cnt] = 0x8000;
              ybp[cnt] = 0x8000;
               }
              control = 0;
              break;
            default:
              control = 0;
         }
      }
   }while(control);
  P1IE  |= BIT1;          // Audiointerrupt enabled
  }
}
```

audiointerrupt.c

```
#include <MSP430x14x.h>

#define LP_LENGTH 3 //  antallet af koefficienter i differenslignингerne
#define HP_LENGTH 3 //  11 multiplikationer = 11 koefficienter
#define BP_LENGTH 5
```

```
// Input initialisering
unsigned int input1;
unsigned int input2;
unsigned int inputsample;

// Equalizer initialisering
extern signed int LP_koeff[LP_LENGTH];
extern signed int HP_koeff[HP_LENGTH];
extern signed int BP_koeff[BP_LENGTH];
extern unsigned int cnt;

// Sample initialisering
extern signed int x[2];
extern signed int ylp[2];
extern signed int yhp[3];
extern signed int ybp[3];

/* Port 1 Interrupt Vector */
#pragma vector=PORT1_VECTOR
__interrupt void audiointerrupt(void)
{
/* Inputrutine */
  input1 = P6IN;
  input2 = P5IN;

  x[0]  = input1 << 8; //læs msb og shift den til øvre byte
  x[0] |= input2;      //indlæs lsb til nedre byte

/* Equalizer rutine */

/* Lavpas shelving filter */
  // Differensligning for 1. ordens lavpas shelving filter - parametrisk equalizer:
  // ylp[n] = b0x[n] + b1x[n-1] - a1ylp[n-1]

    MPYS = LP_koeff[0];
    OP2  = x[0];
    MACS = LP_koeff[1];
    OP2  = x[1];
    MACS = -LP_koeff[2];
    OP2  = ylp[1];
    cnt = 0;        //NOP
    ylp[0] = RESHI;
    ylp[0] = RESHI << 2;  //korriger for (1,14) rep.

    x[1] = x[0];    //Inputswapping

/* Højpas shelving filter */
  //Differensligning for 1. ordens højpas shelving filter - parametrisk equalizer:
  //yhp[n] = b0ylp[n] + b1ylp[n-1] - a1yhp[n-1]

    MPYS = HP_koeff[0];
    OP2  = ylp[0];
    MACS = HP_koeff[1];
    OP2  = ylp[1];
    MACS = -HP_koeff[2];
    OP2  = yhp[1];
    cnt = 0;        //NOP
    yhp[0] = RESHI;
    yhp[0] = yhp[0] << 2;

    ylp[1] = ylp[0];  //Lavpas output-swapping

/* Båndpas filter */
  //Differensligning for 2. ordens båndpas filter - parametrisk equalizer
  //ypn[n] = b0yhp[n] + b1yhp[n-1] + b2yhp[n-2] - a1ypn[n-1] - a2ypn[n-2]

    MPYS = BP_koeff[0];
    OP2  = yhp[0];
    MACS = BP_koeff[1];
    OP2  = yhp[1];
    MACS = BP_koeff[2];
```

```
    OP2  = yhp[2];
    MACS = -BP_koeff[4];
    OP2  = ybp[2];
    MACS = -BP_koeff[3];
    OP2  = ybp[1];
    cnt = 0;          //NOP
    ybp[0] = RESHI;
    ybp[0] =  ybp[0] << 2;

    yhp[2] = yhp[1];  //Højpas output-swapping
    yhp[1] = yhp[0];
    ybp[2] = ybp[1];  //Båndpas output-swapping
    ybp[1] = ybp[0];

/* Outputrutine */
//  outputsample = ybp[0];
  P2OUT |= ybp[0];
  ybp[0] = ybp[0] >> 8;
  P4OUT = ybp[0];

  P1IFG &= ~0x01;   // P1.1 interruptflag cleared
}
```

serial.h

```
   void serial_init(void);      //initialiser USART0
   int  putchar(int);           //send karakter når bufferen er klar
   char KeyHit(void);           //sand ved tastetryk
   char GetChar(void);          //get char
   int  getchar(void);          //get char (buffered) (scanf kompatibilitet)
```

serial.c

```
#include <MSP430x14x.h>
#include "../release/common.h"
#include <stdio.h>

#define BS    '\x08'              // ASCII <BS>
#define LINE_LENGTH 128
char io_buffer[LINE_LENGTH + 1];  //input buffer til getchar
int  ptr;                         //pointer til næste bufferinput

#define RXBUFSIZE 32                          // receive buffer størrelse
unsigned char ucRXBuffer[RXBUFSIZE];          // receive buffer
unsigned char ucRXReadIndex, ucRXWriteIndex; // receive buffer indexes
unsigned char ucRXCharCount;                  // tæller af bytes modtaget

void serial_init(void)
{
   unsigned int br;
   unsigned char br1,br0;
   br = (unsigned int)(XT2/19200);
   br0=(unsigned char)(br & 0x00FF);
   br1=(unsigned char)(br >> 8);
   U0CTL  &=~ SWRST;          // software reset UCTL0.0:=0
   U0CTL   = 0x50;            // 8 databit, 1 stopbit ingen paritetscheck
   U0TCTL  = 0x30;            // anvend SMCLK
   U0BR0   = br0;             // SMCLK/baud=XT2/19200
   U0BR1   = br1;             //      -"-
   U0MCTL  = 0x00;            // ingen modulation
   ME1    |= UTXE0+URXE0;     // enable USART transmit/receive
   P3SEL  |= BIT4+BIT5;       // Pin P3.4/P3.5 USART TXD/RXD
   P3DIR  |= BIT4;            // Pin P3.4 sættes som output
   IE1    |= URXIE0;          // enable interrupt til modtagelse
   ucRXWriteIndex = ucRXReadIndex = ucRXCharCount = 0;
   for (ptr=0; ptr<(LINE_LENGTH+1); ptr++) {io_buffer[ptr]=0;}
   ptr=0;
}
```

```
int putchar(int c)
{
   while ((IFG1&UTXIFG0)!= UTXIFG0);  // vent til U0TXBUF er klar
   U0TXBUF=c;                         // send serial på P3.4
   return (1);
}

#pragma vector=UART0RX_VECTOR
__interrupt void uart0_rx(void)
{
   P1IE  &= ~BIT1;                        // P1.1 Interrupt disabled
   _EINT();
   ucRXBuffer[ucRXWriteIndex++] = U0RXBUF;// gem modtaget byte og
                                          // increment receive index
   ucRXWriteIndex &= RXBUFSIZE-1;         // reset index
   ucRXCharCount++;
}

char KeyHit(void)                         // sand ved tastetryk
{
   if(ucRXCharCount) return(1);
   else return(0);
}

char GetChar (void)
{
   char Byte;
   while (!KeyHit());                  // vent til RXBUF1 er klat
   Byte = ucRXBuffer[ucRXReadIndex++]; // hent byte fra buffer
   ucRXReadIndex &= RXBUFSIZE-1;       // juster index
   IE1 &= ~URXIE0;                     // disable receive interrupt
   ucRXCharCount--;                    // en karakter læst, decrement tæller
   IE1 |=  URXIE0;
   return (Byte);
}

int getchar(void) //buffered getchar (scanf kompatibilitet)
{
   char c;
   c=io_buffer[ptr];
   if(c){ptr++; return(c);}
   ptr = 0;
   do
   {
      c = GetChar();
      if ((c == BS)&&(ptr>0))
      {
         ptr--; putchar(BS); putchar(' '); putchar(BS);
      }
      else if (ptr < LINE_LENGTH)
      {
         if (c >= ' '){putchar (io_buffer[ptr] = c);ptr++;}
      }
   }while(c!='\r');
   putchar (io_buffer[ptr] = '\n');
   ptr++;
   io_buffer[ptr] = 0;
   ptr=1;
   return(io_buffer[0]);
}
```

port_modules.h

```
//Port 1
void port_interrupt_init(void);

//Port 2
void lower_dac_init(void);

//Port 3
```

```
void reset_init(void);

//Port 4
void upper_dac_init(void);

//Port 5
void lower_adc_init(void);

//Port 6
void upper_adc_init(void);
```

port_modules.c

```
#include "MSP430x14x.h"

// Initialiser port 1
// P1 er interrupt-port
void port_interrupt_init(void)
{
   P1DIR |= BIT0;
   P1IE  |= BIT1;   // audiointerrupt (P1.1) enabled
   P1IES |= BIT1;   // P1.1 trigges på high edge
   P1IFG &= ~0x01;   // P1.1 interruptflag cleared
}

// Initialiser port 2
// P2 er den nedre del af DAC word.
void lower_dac_init(void)
{
  P2DIR |= 0xFF;  // Sæt P4 til output retning
}

//Sæt port 3.7 til reset pin.
//port 3.4 og 3.5 sættes i serial_intr.c
void reset_init(void)
{
   P3DIR |= BIT7;       // Sæt port 3.7 til output retning
}

// Initialiser port 4
// P4 er den øvre del af DAC word
void upper_dac_init(void)
{
  P4DIR |= 0xFF;  // Sæt P4 til output retning
}

// Initialiser Port 5
// P5 er den nedre del af ADC word.
void lower_adc_init(void)
{
  P5DIR |= 0x00;  // Sæt P5 til input retning
}

// Initialiser Port 6
// P6 er den øvre del af ADC word
void upper_adc_init(void)
{
  P6DIR |= 0x00;  // Sæt P6 til input retning
}
```

basic_clock_module.h

```
void basic_clock_init(void); //modul til styring af clockvalg
```

basic_clock_module.c

```
#include "MSP430x14x.h"
```

```
void basic_clock_init(void)         //Modul til styring af clockvalg
{
  int i;
  WDTCTL = WDTPW + WDTHOLD;         // Stop watchdog timer
  BCSCTL2 |=0x08;                   // SMCLK = XT2
  do{
    IFG1 &= 0x0FD;                  // Clear OSCFault flag
    for (i = 0xFFFF; i > 0; i--);   // Tid for at flaget sættes
  }while ((IFG1 & OFIFG) != 0);     // OSCFault flag stadigt sat?
  BCSCTL2 |= 0x80;                  // MCLK = XT2
}
```

common.h

```
#ifndef common_h
#define common_h
#define XT2 7.3728e6 //den valgte clockfrekvens

#endif
```

delay.h

```
void Delay(unsigned int);  //software delay
```

Assembler-kode

audiointerrupt.s43

```
        NAME audiointerrupt

        RSEG CSTACK:DATA:SORT:NOROOT(1)

        EXTERN ?cstart_init_zero

        PUBWEAK ‘??INTVEC 8‘
        PUBWEAK MACS
        PUBWEAK MPYS
        PUBWEAK OP2
        PUBWEAK P1IFG
        PUBWEAK P2OUT
        PUBWEAK P4OUT
        PUBWEAK P5IN
        PUBWEAK P6IN
        PUBWEAK RESHI
        PUBLIC audiointerrupt
        EXTERN input1
        EXTERN input2
        EXTERN inputsample

        EXTERN LP_koeff
        EXTERN HP_koeff
        EXTERN BP_koeff


        ASEGN DATA16_AN:DATA:NOROOT,0132H
MPYS:
        DS8 2

        ASEGN DATA16_AN:DATA:NOROOT,0136H
MACS:
        DS8 2

        ASEGN DATA16_AN:DATA:NOROOT,0138H
```

```
OP2:
        DS8 2

        ASEGN DATA16_AN:DATA:NOROOT,013cH

RESHI:
        DS8 2

        ASEGN DATA16_AN:DATA:NOROOT,023H

P1IFG:
        DS8 1

        ASEGN DATA16_AN:DATA:NOROOT,029H

P2OUT:
        DS8 1

        ASEGN DATA16_AN:DATA:NOROOT,01dH

P4OUT:
        DS8 1

        ASEGN DATA16_AN:DATA:NOROOT,030H

P5IN:
        DS8 1

        ASEGN DATA16_AN:DATA:NOROOT,034H

P6IN:
        DS8 1

        RSEG DATA16_Z:DATA:SORT:NOROOT(1)
REQUIRE ?cstart_init_zero

        RSEG CODE:CODE:REORDER:NOROOT(1)
audiointerrupt:
PUSH.W R15
PUSH.W R14
PUSH.W R13
PUSH.W R12
PUSH.W R11
PUSH.W R10
PUSH.W R9
PUSH.W R8
        PUSH.W R7
PUSH.W R6
PUSH.W R5
PUSH.W R4

// Inputrutine
MOV.B &0x34, R15
MOV.B &0x30, R14
SWPB R15
BIS.W R14, R15


// Equalizer rutine

// Lavpas shelving filter
// Differensligning for 1. ordens lavpas shelving filter - parametrisk equalizer:
// ylp[n] = b0x[n] + b1x[n-1] - a1ylp[n-1]


// b0x[0]
MOV.W &LP_koeff, &0x132 //HW-multiplier MPYS
MOV.W R15, &0x138 //HW-multiplier OP2

// b1x[1]
MOV.W &LP_koeff + 2, &0x136 //HW-multiplier MACS
MOV.W R14, &0x138
```

```
// -a1ylp[1]
MOV.W &LP_koeff + 4, R5 //R5 er a1
XOR.W #0xffff, R5 //a1 inverteres og adderes 1 (sign skift)
ADD.W #0x1, R5
MOV.W R5, &0x136
MOV.W R12, &0x138
NOP

MOV.W #0x13c, R4 //R4 sættes som pointer til RESHI
MOV.W @R4, R13

RLA.W R13 //Korriger for (1,14 rep)
RLA.W R13

//Inputswapping
MOV.W R15, R14

// Højpas shelving filter */
// Differensligning for 1. ordens højpas shelving filter - parametrisk equalizer:
// yhp[n] = b0ylp[n] + b1ylp[n-1] - a1yhp[n-1]

// b0ylp[0]
MOV.W &HP_koeff, &0x132
MOV.W R13, &0x138

// b1ylp[1]
MOV.W &HP_koeff + 2, &0x136
MOV.W R12, &0x138

// -a1yhp[1]
MOV.W &HP_koeff + 4, R5 //R5 sættes som a1
XOR.W #0xffff, R5 //R5 inverteres og adderes 1 (sign skift)
ADD.W #0x1, R5
MOV.W R5, &0x136

MOV.W R10, &0x138
NOP

MOV.W @R4, R11

RLA.W R11 //Korriger for (1,14) rep.
RLA.W R11

// Lavpas output-swapping
MOV.W R13, R12

// Båndpas filter
// Differensligning for 2. ordens båndpas filter - parametrisk equalizer
// ypn[n] = b0yhp[n] + b1yhp[n-1] + b2yhp[n-2] - a1ypn[n-1] - a2ypn[n-2]

//  b0yhp[0]
MOV.W &BP_koeff, &0x132
MOV.W R11, &0x138

//    b1yhp[1]
MOV.W &BP_koeff + 2, &0x136
MOV.W R10, &0x138

//  b2yhp[2]
MOV.W &BP_koeff + 4, &0x136
MOV.W R9, &0x138

// -a1ybp[1]
MOV.W &BP_koeff + 8, R5
XOR.W #0xffff, R5
ADD.W #0x1, R5
MOV.W R5, &0x136
MOV.W R6, &0x138

// -a2ybp[2]
MOV.W &BP_koeff + 6, R5
```

```
XOR.W #0xffff, R5
ADD.W #0x1, R5
MOV.W R5, &0x136
MOV.W R7, &0x138
        NOP
MOV.W @R4, R8

RLA.W R8
RLA.W R8

//Højpas output-swapping
MOV.W R10, R9
MOV.W R11, R10

//Båndpas output-swapping
MOV.W R7, R6
MOV.W R8, R7


// Outputrutine
BIC.B R8, &0x29
SWPB R8
MOV.B R8, &0x1d

// P1.1 interruptflag cleared
BIC.B #0x1, &0x23

POP.W R15
POP.W R14
POP.W R13
POP.W R12
POP.W R11
POP.W R10
POP.W R9
POP.W R8
        POP.W R7
POP.W R6
POP.W R5
POP.W R4


RETI

        COMMON INTVEC:CONST:ROOT(1)
        ORG 8

‘??INTVEC 8‘:
DC16 audiointerrupt

        RSEG CODE:CODE:REORDER:NOROOT(1)
        END
```