AALBORG UNIVERSITET
INSTITUT FOR ELEKTRONISKE SYSTEMER
AFDELING FOR KOMMUNIKATIONSTEKNOLOGI

Frederik Bajersvej 7 ▪ DK-9220 AALBORG Ø   Telefon 98 15 85 22

TITEL:	Frekvensanalysator
TEMA:	Maskinnær programmering
PROJEKTPERIODE:	4. semester, februar til juni 1999
PROJEKTGRUPPE:	453

GRUPPEMEDLEMMER:
Claus Albøge
Mads Græsbøll Christensen
Tonny Gregersen
Karsten Jensen
Peter Korsgaard
Lars Jochumsen Kristensen
Robert Stepien

VEJLEDER:
Ole Olsen

OPLAGSTAL:	10
ANTAL SIDER:	214
ANTAL BILAG:	13
FÆRDIGGJORT:	2. juni '99

Synopsis

Denne rapport omhandler udviklingen af både hardware og software til et dedikeret realtidssystem til frekvensanalyse af audio-signaler.
Hardwaremæssigt tager projektet sit udgangspunkt i en Motorola MC68331-microcontroller.
Realtidsafviklingen er bl.a. realiseret ved DMA-overførsel af det samplede signal til RAM og Power Spectrum til display.
Frekvensanalysen er baseret på en optimeret anvendelse af Fast Fourier Transform-algoritmen.

AALBORG UNIVERSITY
INSTITUTE OF ELECTRONIC SYSTEMS
DEPARTMENT OF COMMUNICATION TECHNOLOGY

Frederik Bajersvej 7 ▪ DK-9220 AALBORG Ø Telephone 98 15 85 22

TITLE:	Frequency Analyzer
THEME:	Low-level programming
PROJECT PERIOD:	4th semester, February to June 1999
PROJECT GROUP:	453

PARTICIPANTS:
Claus Albøge
Mads Græsbøll Christensen
Tonny Gregersen
Karsten Jensen
Peter Korsgaard
Lars Jochumsen Kristensen
Robert Stepien

SUPERVISOR:
Ole Olsen

COPIES:	10
NUMBER OF PAGES:	214
ENCLOSED PAGES:	13
FINISHED:	June 2nd '99

Abstract

This report documents the development of both hardware and software for a dedicated real-time system for frequency analysis of audio signals.
The system is based on a Motorola MC68331-microcontroller.
Real-time execution is realized by e.g. DMA-transfer of sampled signals to RAM and Power Spectrum to the display.
The frequency analysis is based on an optimized application of the Fast Fourier Transform-algorithm.

Forord

Denne rapport er dokumentationen af projektgruppe 453's arbejde på datateknik 4. semester 1999 på Institut for Elektroniske Systemer, Aalborg Universitet.
Rapporten henvender sig hovedsageligt til vejleder, censor samt senere studerende på datateknik.

Diagrammer og måleplot er at finde i appendiks F. I appendiks A er desuden en gennemgang af den relevante matematiske teori at finde, nærmere bestemt Fourierrækker, kontinuert og diskret Fouriertransformation og Fast Fourier Transformationen.

Aalborg, 2. juni 1999

Claus Albøge

Lars Jochumsen Kristensen

Tonny Gregersen

Mads Græsbøll Christensen

Karsten Jensen

Peter Korsgaard

Robert Stepien

Indhold

Kapitel 1

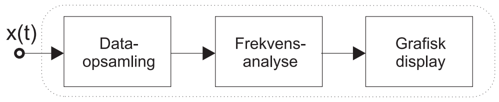
Kravspecifikation

1.1 Formål

Hardware og software til et dedikeret system til realtids frekvensanalyse af audiosignaler i det hørbare område udvikles.
Projektet har taget sit udgangspunkt i et projektforslag af Ole Olsen [22].
De formelle mål er at finde i studieordningen for 4. semester datateknik [2].

1.2 Systembeskrivelse

Systemet udvikles med udgangspunkt i en Motorola MC68331-mikrocontroller.



Figur 1.1: Et signal, x(t), opsamles. Frekvensindholdet af signal analyseres og præsenteres på et display.

Systemet kan groft beskrives som bestående af følgende dele:

- Opsamling af data
- Frekvensanalyse
- Visning af resultat

Dette er også beskrevet på figur 1.1.

1.3 Funktionalitetsbeskrivelse

En signalgiver tilsluttes indgangen. Signalet konverteres fra analog til digital. Frekvensanalysen foretages så af mikrocontrolleren på de opsamlede data, og resultatet præsenteres grafisk på et LCD-display.
Frekvensanalyse går ud på at opløse indholdet af et signal i et antal frekvenskomposanter, dvs. en række vægtede sinus- og cosinusfunktioner.
En frekvensanalyse er identisk med den matematiske transformation, Fourier Transformation. Her foretages denne af en mikrocontroller, og analysen bliver herfor tids- og frekvensdiskret. Den diskrete Fourier Transformation er også kendt som en DFT.

1.4 Begrænsninger

Det fremstillede system har en række begrænsninger:

- Det fremstillede produkt har karakter af en prototype
- Tilgængelig hardware
- Produktet bliver ikke optimeret med hensyn til strømforbrug, MMI og fysisk udformning.

Disse skal ses som et udtryk for prioritering i henhold til målene med 4. semester datateknik, samt de rent praktiske forhold på 4. semester, Institut for Elektroniske Systemer. Der er desuden en økonomisk grænse for projektet.

1.5 Udviklingsforløbet

1.5.1 Metode

Som generel metode til udvikling af software anvendes dele af Struktureret Program Udvikling [4].
Design foretages top-down, dvs. med udgangspunkt i funktionalitet, hvorimod test foretages bottom-up. Test vil med andre ord blive gennemført på modulplan og efterfulgt af integrationstests, efterhånden som de enkelte moduler samles.

1.5.2 Software

Software udvikles i ANSI C. Tidskritiske dele vil blive optimeret i CPU32 assembler.
CVS anvendes til revisionsstyring.
Af compiler anvendes GNU's C og C++ Compiler (gcc), også GNU's Assembler (as) og debugger (gdb) anvendes.

1.5.3 Logikkonstruktion

Til programmering af programerbare logikkredsløb (PLD'er) anvendes PEEL og MAX+PlusII.

1.5.4 Hardware

Motorolas microcontroller MC68331 skal benyttes. Et display (LCD) fra Seiko Instruments Inc., nærmere bestemt G1216, og en 10bit A/D Converter, ADC1061, er valgt. Disse danner udgangspunktet for systemet. Generelt anvendes den på Institut for Elektriske Systemer tilgængelige hardware.

1.5.5 Frekvensanalyse

Frekvensanalysen skal foretages ved hjælp af Fast Fourier Transform-algoritmen.

1.6 Definitioner

Realtid: Ikke sanselig, variabel tidsforsinkelse.

Clocktrue: Ikke sanselig, konstant tidsforsinkelse.

1.7 Specifikke krav

1.7.1 Obligatoriske

Systemet skal opfylde følgende krav:

- Realtids afvikling
- Frekvensområde: 20Hz - 20kHz

- Minimum 128 punkter (256 punkters DFT)
- Mono line-in input (2V peak to peak)
- Output på LCD display
- Grafisk repræsentation af power spectrum (lineær frekvensakse)

1.7.2 Optionelle

Følgende krav skal overvejes i designet og muligvis realiseres:

- Brugergrænseflade
- Logaritmisk frekvensskala
- Sample/buffer viewer

1.8 Fremtidige udvidelsesmuligheder

Nogle mulige funktioner og egenskaber, foruden de tidligere nævnte optionelle krav, er her anført.

- RS232/Parallelt interface
- 10" B/W display
- PC-software til videre behandling/præsentation af resultater
- Grafisk præsentation af Amplitude- og fasekarakteristik

Disse vil ikke blive realiseret.

1.9 Eksterne grænseflader

1.9.1 Brugergrænseflade

En minimal brugergrænseflade vil blive fremstillet i form af en reset-funktion, og resultatet af frekvensanalysen præsenteres på et grafisk display.

1.9.2 Hardwaregrænseflade

Inputsignalet på op til 2V peak to peak (0.707V rms).

1.10 Kvalitetsfaktorer

Her følger de kvalitetsfaktorer, der tillægges vægt under designet. Disse skal ses om et udgangspunkt for prioritering under udviklingsfasen. Bemærk, at kun relevate punkter er medtaget.

1.10.1 Pålidelighed

Grundet produktets karakter af stand-alone monitor vil konsekvensen af fejl (crash) være lille. Det vil altid være muligt for brugeren af genstarte systemet i anvendelsessituationen.

1.10.2 Vedligeholdelsesvenlighed

En udpræget modularisering og indkapsling gør de enkelte dele modificerbare, både med hensyn til specifikationsændringer samt fejlretning.

1.10.3 Udvidelsesvenlighed

Vigtigt i henhold til realisering af de optionelle krav. Opnåes, ligesom vedligeholdelsesvenligheden, igennem modularisering og indkapsling.

1.10.4 Brugervenlighed

Brugergrænseflade tillægges ingen større betydning. Output søges dog præsenteret på en overskuelig måde.

1.10.5 Effektivitet

Systemet skal fungere realtid. Clock-true afvikling er ikke mulig med den givne mikrocontroller, idet f.eks. multiplikationer vil have varierende længde.

Kapitel 2

Hardware/software co-design

2.1 Indledning

Temaet for dette projekt er maskinnær programmering [2]. Da anvendelsen af Programable Logic Device (PLD)'er i logik-konstruktion, gør grænserne mellem software-design og hardware-design flydende, er det nødvendigt med en række indledende overvejelser, vedrørende hardware/software co-design, der kan motivere en opdeling af projektet i en hardwaredel og en softwaredel, og definere grænseflader mellem disse.
Den overordnede motiverende faktor i denne aktivitet, er behovet for at maksimere throughput og minimere latens, samt at minimere arealforbrug (udmålt som størrelsen af chip, board og hukommelse) [11, s. 122, s. 227]. Disse mål udelukker ofte gensidigt hinanden. En forøgelse af throughput eller en formindskelse af latens vil som regel kun kunne opnåes ved at øge arealet. Resultatet af designfasen vil derfor være en afvejning af tidskrav overfor arealforbrug.

2.2 Design-kriterier

Realtidsafvikling er et obligatorisk krav, hvorfor minimering af kørselstid vægtes højest i designfasen. Først når realtidskravet er opfyldt, vil systemet blive optimeret med hensyn til kvantiseringsstøj. Hvad angår arealforbrug, så er den eneste specificerede begrænsning, at den hardware, der er til rådighed på Institut for Elektroniske Systemer, skal anvendes.

2.3 Generelle overvejelser

Det overordnede system består, som beskrevet i kravsspecifikationen, at tre dele:

- En opsamlingsdel.
- En behandlingsdel.
- En udlæsningsdel.

Dertil kommer en række mindre styringskredsløb og lagerkredse.
Det specificerede display har en bredde på 128 pixels[26] og udnyttes fuldt ud, hvis vi opererer på blokke med længden 256 (se appendix A).

2.3.1 Opsamlingsdel

Opsamlingsdelen består af antialiaserings-filter og ADC. Antialiseringsfilteret skal implementeres i analog hardware og er kun relevant for disse overvejelser, for så vidt det skal styres af en ekstern clock, der teoretisk kan genereres ved hjælp af MCU'ens GPT (General Purpose Timer). Hvad angår ADC'en, er det relevant at overveje om timingen af denne skal styres af software eller hardware. Endelig bør det overvejes, hvordan overførslen af data fra opsamlingsdel til behandlingsdel skal styres.
Den tid, der medgår til at opsamle data og overføre dem til RAM er med en samplerate på 40 kHz og en blokstørrelse på 256 lig $\frac{256}{40\cdot 10^3}$s$= 6.4 < cdot 10^{-3}$s.

2.3.2 Behandlingsdel

Den centrale del af vores system er frekvensanalysen. Denne bygger på en beregningstung algoritme med et stort antal multiplikationer.
Beregningen vil kræve $\frac{256}{2} < cdot log_2(256) = 1024$ (se appendix A) komplekse multiplikationer og ca. ligeså mange komplekse additioner. Antager vi, at ordlængden er 32 bit kræver multiplikationerne alene $2 \cdot 52 \cdot 1024 = 106496$ clock-cycles. Dertil kommer et antal clock-cycles der medgår til additioner og læsning og skrivning fra og til RAM, samt den indledende permutation A. For at få et overslag over antallet af clock-cycles, der ialt medgår til disse beregningen, ganger vi med en faktor 1.5 og får 159744 clock-cycles eller $\frac{159744}{16\cdot 10^6} \approx 10^{-2}$s.

2.3.3 Udlæsningsdel

Udlæsningsdelen håndterer overførslen af output fra FFT'en til LCD-display. Styringen af denne dataoverførsel kan både ligge i hardware og i software. Det specificerede display kan maksimalt opdateres ca. 11 gange/s [26]. Skulle udlæsningen styres via CPU'en ville udlæsningen af en blok lægge beslag på processoren i $90 \cdot 10^{-3}$s.
Da vi har et realtidskrav, vil vi gerne udnytte displayets kapacitet fuldt ud. Dette kan kun lade sig gøre, hvis vi aflaster CPU'en ved at flytte en del af systemets funktionalitet over i hardware.

2.4 Specifikke overvejelser

Ovenstående leder frem til at følgende punkter bør overvejes i relation til hardware/software co-design:

- Filter-clock.
- ADC-timing.
- Lagring af sample-data
- Frekvensanalyse
- Udlæsning på display

2.4.1 Filter-clock

Vi kan vælge at anvende et switch-capacitor filter (se afsnit 15.3.1), der tillader at afskæringsfrekvensen styres ved hjælp af en ekstern clock. Det giver os mulighed for at variere samplefrekvensen i tilfælde af, at softwaren ikke kan afvikles i realtid, ved den givne samplefrekvens.
Clock/afskæringsfrekvens forholdet er for det valgte filter 100 ([10, s. 6-40]. Den specificerede øvre grænsefrekvens er 20 kHz. For det tilfælde, at vi ville lade programmet generere denne clock, ved hjælp MCU'ens GPT, ville det medføre, at CPU'en i værste tilfælde afbrydes med et IRQ (Interrupt request) $2 \cdot 10^6$ gange pr. sekund. Der ville med en 16 MHz da kun være 8 clock-cycles mellem hvert IRQ. Tabel 2.1 angiver hvor mange clock-cycles, der medgår til et interrupt.

Operationer	Antal clock-cycles
Interrupt	37
ISR (ca.)	20
RTE	26
Ialt	83

Timing data er baseret på 3-clock reads og writes. [13, s. 8-16 - 8-28]

Figur 2.1: Overslag over antallet af clock-cycles, der medgår til et interrupt

2.4.2 ADC-timing

ADC'en skal times til den ønskede samplefrekvens. Vi har overvejet to mulige software-løsninger:

- Vi kan lade timingen være programstyret.
- Vi kan lade timingen være interruptstyret.

I det første tilfælde kræves det, at programmet er "busy waiting", og dermed lægger beslag på CPU'en i sampleperioden.
I det andet tilfælde belastes CPU'en i INTERRUPTTID*BLOCKSIZE. Antallet af clock-cycles i en sampleperiode er $\frac{16 \cdot 10^6}{44100} \approx 363$. CPU'en beslaglægges på denne måde i $\frac{83}{363} \cdot 100 \approx 23$ procent af sampleperioden.

2.4.3 Lagring af sampledata

Både program- og interruptdrevet ADC-timing kræver, at CPU'en tager aktivt del i overførslen af sampledata fra ADC'en til hukommelsen. Når en sample er klar, må CPU'en læse den og dernæst skrive den til RAM'en. Hvor ofte rutinen skal udføres bestemmes af sampleraten. Disse metoder har to ulemper:

1. Throughput begrænses af processorens hastighed, når den udfører ISR, samt læser sampledata og skriver dem til RAM'en.
2. Processoren kan ikke afvikle andre instruktioner, mens den er optaget af at overføre sampledata. Denne ulempe er specielt stor, når hovedprogrammet, som i vores tilfælde på grund af antallet af multiplikationer, er processor-bound (i modsætning til memory-bound).

DMA (Direct Memory Access) er en effektiv teknik til at løse dette problem. Ved hjælp af en DMA-controller overføres sampledata direkte fra ADC'en til hukommelse, uden at CPU'en involveres. Først når samplebufferen er fuld, sendes et IRQ til CPU'en. Denne løsning kræver en protokol, der bestemmer hvilken enhed, der har adgang til bussen på et givet tidspunkt. Dette kan igen bevirke, at der kan være tidspunkter, hvor processoren har brug for databussen, men må vente fordi den er optaget. MC68331'eren får automatisk altid laveste prioritet, når bussen deles [16, s. 4-41]. Løsningen kræver endvidere en relativt stor forøgelse af chip-arealet.
DMA har to store fordele:

1. Throughput forøges kraftigt.

2. Processoren kan afvikle andre instruktioner, mens der overføres sampledata. Dette er specielt en fordel, når hovedprogrammet er processor-bound.

Derudover kan DMA-controlleren styre ADC-timingen, via interrupts, der requestes hver gang sample-bufferen er fyldt op. Dermed aflastes CPU'en yderligere, og det bliver muligt at pipeline.

2.4.4 Frekvensanalyse

FFT'en har to sektioner (se appendix A). Den første sektion kan udføres som en in place permutation. Den anden sektion er en række indlejrede løkker med en avanceret adresseringsmekanik. Både permutation og adresseringsmekanik kan håndteres i hardware. I forhold til antallet af multiplikationer, der indgår i FFT'en, er tidsbesparelsen ikke særlig stor. Til gengæld er arealforøgelsen lille.

2.4.5 Udlæsning på display

Overvejelserne i forbindelse med lagring af sampledata gælder også for udlæsning på display. Denne kan med fordel udføres ved hjælp af DMA.

2.5 Konklusion

Vi har foretaget følgende valg af grænseflader mellem hardware og software:

- Filter-clocken styres med hardware. Med den givne processor er det ikke muligt, at styre den med software.

- Da vi har to Altera-kredse til rådighed, har vi valgt at anvende DMA til overførsel af sampledata og udlæsning på display. Vi får dermed en forøgelse af throughput og processoren aflastes. Permutationen af input-data, der udgør den ene sektion af FFT'en, kan udføres simpelt ved overførslen af sampledata fra ADC til RAM. Endvidere lægges ADC-timingen over i DMA'en for at aflaste CPU'en yderligere. Denne løsning tillader pipelining.

- Den komplicerede adresseringsmekanik i FFT'en vil ikke blive håndteret med hardware. Det vil kræve en grundig analyse af FFT'en, og tidsbesparelsen er ubetydelig, sammenlignet med eksekveringstiden for beregningsalgoritmen.

Kapitel 3

Generel software design

Følgende kapitel indeholder en beskrivelse af software designet. Udgangspunktet i selve designet er taget i det overordnede software/hardware-codesign, samt enkelte software-specifikke designkriterier. Formålet med kapitelet er at opstille designkriterier for implementationen af softwaren.

3.1 Designkriterier

Softwaremæssigt set er det primære designkriterie hastighed. Derfor vil softwaren i første omgang blive optimeret med hensyn til hurtig afvikling. Sekundært ønskes et modulopbygget softwaresystem med veldefinerede grænseflader, da det giver bedre udvidelsesmuligheder, samt et simpelt API(Application Programmers Interface).

3.2 Design af Operativsystem

3.2.1 Formål

Formålet med Operativsystemet (OS) er at abstrahere over den anvendte hardware, og stille et interface til rådighed der:

- er simpel at kommunikere med.
- har lav kobling til den underliggende hardware.
- giver beskyttet adgang til ressourcer.

- er effektiv (dvs. lille overhead)

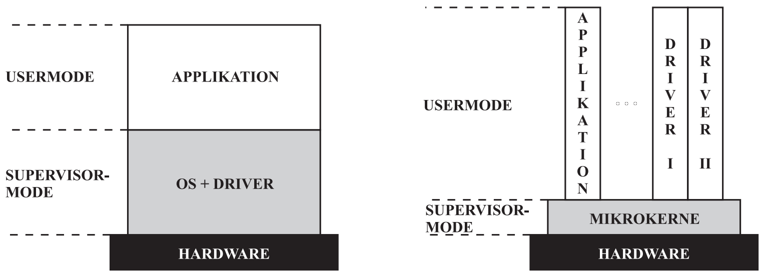
3.3 Anvendelse

Operativsystemer deles ofte op i singletasking og multitasking systemer. Multitaskingsystemer har klare fordele i generelle systemer som f.eks en PC, idet de højner effektivitetsgraden af CPU'en. I et dedikeret system som dette er det dog ikke altid tilfældet. Hvis programmets algoritme er udpræget seriel, og der ikke bruges en stor del af processortiden på at vente ved flaskehalse, vil et singletasking OS være ideel. Dette skyldes at et multitaskingsystem uundgåeligt vil have et vist skeduleringsoverhead.

3.3.1 Opbygning

Der er 2 grundlæggende designfilosofier indenfor design af operativsystemskerner:

- Monolitiske systemer, med meget funktionaltet indbygget i kernen.
- Mikrokerner, hvor så meget funktionalitet som muligt er flyttet ud af kernen.



Figur 3.1: Monolitisk system vs. Mikrokerne

Monolitiske systemer

Monolitiske systemer har forholdsvis høj kobling mellem de enkelte dele af OS'et, idet hele OS'et er samlet i én del. Hele OS'et kører i supervisormode. Dette betyder dels at ændringer i den underliggende hardware kan betyde store ændringer i OS'et, og dels at kommunikation mellem de enkelte dele af OS'et kan foregå med lille overhead.

Mikrokerner

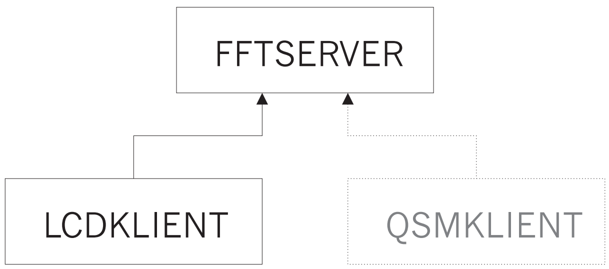
Mikrokerner besidder oftest højere eksporterbarhed end monolitiske systemer, idet de hardwarespecifikke drivere er indkapslet i selvstændige moduler. I disse systemer er kernen reduceret til en budbringer, der står for kommunikationen mellem brugerprogrammet og driverne. Desuden placeres så meget som muligt af operativsystemet i usermode. Mikrokerner har ofte dårligere performance end monolitiske systemer, idet kommunikation mellem de enkelte dele af operativsystemet skal foregå igennem kernen.

3.3.2 Valg

På grund af det specikke anvendelsesområde (FFT-beregning), har vi valgt at lave et singletasking, mikrokernebaseret OS. Dette er dels valgt på grund af det sekventielle forløb af algoritmen, og dels fordi de optionelle krav ligger op til udvidelser af softwaren.

3.4 Softwareopdeling

Som beskrevet i HW/SW co-design afsnittet, skal selve FFT beregningerne foretages af software. FFT'en må anses som den vigtigste del, men derudover skal også flytning af data fra A/D-konverter til RAM og fra RAM til LCD-display styres software mæssigt. Kronologisk set skal der laves styring af sampledel, beregning af FFT og til sidst skrivning på LCD-display. Disse trin lægger op til en modulær opbygning af softwaren, således at det består af 5 dele: sample styring, indlæsning, FFT beregning, udlæsning og display styring. For at opnå lav kobling mellem FFT beregning og udlæsning, er applikationen designet som et klient/server system, se figur 3.2.

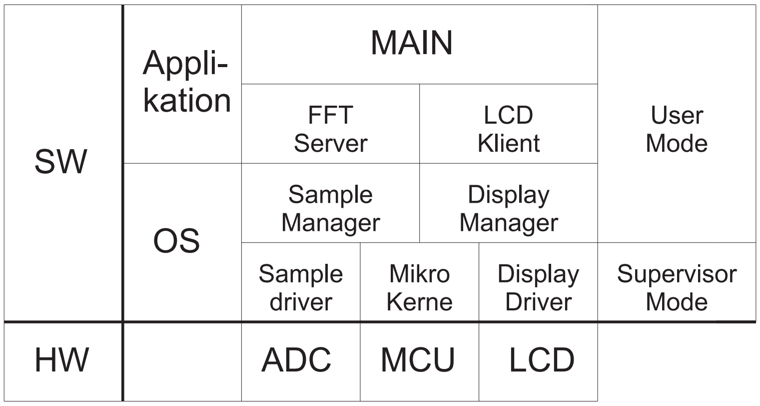


Figur 3.2: Klient/server opdeling af applikationen

Med denne opbygning kan man på simpel vis senere tilføje en klient til udlæsning eksempelvis via QSM interfacet. Styringen af ind- og udlæsning af data implementeres som drivere. Disse drivere er opdelt i 2 dele. En generel del (manager), og en hardwarespecifik del. Denne opdeling er lavet for igen at opnå lav kobling mellem hardware og applikation. Alt i alt giver det følgende opsplitning i software moduler:

- Operativ system
- Hardwarespecifik sampledriver
- Samplemanager
- Hardwarespecifik displaydriver
- Displaymanager
- LCDklient
- FFTserver
- Main

Selve opdelingen ser i lagdelt fremstilling ud som følgende:



Figur 3.3: Den lagdelte struktur i software opbygningen omkring en mikrokerne

Fordelene ved denne måde at strukturere systemet på er, at det er muligt at lave nye applikationer eller ændre eksisterende, uden af skulle lave om på andre dele. Samtidig foregår al kommunikation med hardware gennem mikrokernen, hvilken betyder at "uautoriserede" applikationer ikke får direkte adgang til hardwaren. Den lagdelte struktur bevirker også, at systemet bliver lettere at eksportere til nye "platforme" f.eks. et andet display, idet kun display driveren skal skrives om.

Kapitel 4

Software procesbeskrivelse

Dette kapitel omhandler det praktiske forløb i software udviklingen. Det være sig udvikling af prototype på PC, brug af kompiler, samt generelle overvejelser vedrørende optimering og tabelbrug.

4.1 PC prototype

Selve programmeringsfasen deles op i to delvis sideløbende processer, med hensyn til udarbejdelse af prototype og endelig version. Prototypen laves på PC, og de enkelte moduler eksporteres til MCU'en efterhånden, som de bliver færdige. Lavniveau driverne må nødvendigvis implementeres forskelligt afhængig af underlæggende arkitektur, men der stræbes efter at PC prototypen så vidt muligt kommer til at indeholde samme procedurekald som MCU versionen; hvorfor nogle funktioner vil være tomme funktionskald.

4.1.1 OpenPTC

Under udarbejdelsen af display-delen til PC prototypen, bliver der brug for at emulere LCD-displayet på monitoren. Til det formål findes et grafik bibliotek kaldet OpenPTC, [21], der kan bruges på UNIX systemer. Med dette grafik bibliotek er det muligt at udvikle og teste displaydelen på PC'en.

4.2 Programmering

Software programmeringen til projektet foregår så vidt muligt i en modul opbygget struktur, hvorved de enkelte dele kan implementeres og afprøves hver for sig, og efterhånden sættes sammen til større og større objekter.

4.2.1 Sprog

Størstedelen af softwaren skrives i ANSI C. Dog vil enkelte vitale dele som opstartskode blive skrevet direkte i CPU32 assembler. Selve FFT algoritmen vil ikke blive håndkodet i assembler. Istedet skrives den i C, hvorefter koden oversættes med GNU's C kompiler, som vi har valgt at benytte (se afsnit 4.2.2). Derefter bliver assemblerfilen gennemgået med henblik på yderligere optimering.

4.2.2 Kompilering

GNU's C Compiler bruges til al kompilering. Til kompilering af kode til Motorola, bruges en udviklingsversion af GCC, kaldet EGCS, der bl.a. er specielt anlagt til cross-compiling. Vi har kompileret en version af EGCS, med –target=m68k-coff, der gør det muligt at få kompileren til at generere S-RECORD's, som umiddelbart kan uploades til RAM'en eller ROM'en. En S-RECORD består af en sekvens af specielt formaterede ASCII strenge, og er opbygget som vist på figur 4.2.2

Type	Længde	Adresse	Data	Checksum

Figur 4.1: Opbygning af S-RECORD

De enkelte blokke i strengen beskrives på følgende måde.

Felt	Karakterantal	Beskrivelse
Type	2	SREC type S0 - S9
Længde	2	Antal af karakter-par i SREC, excl. Type og Længde
Adresse	4,6 eller 8	2,3 eller 4-byte hukommelses-adresse, hvortil Data/Kode skal downloades.
Data/Kode	0-2n	Eksekverbar kode, data eller informativ beskrivelse
Checksum	2	Checksum

Figur 4.2: Beskrivelse af S-RECORDEN's opbygning og indhold

For at linkeren kan lave S-RECORD'en skal den have information om systemets memorylayout. Der skal derfor laves et linkerscript med beskrivelse af memorymap, opstartskode mm. (Se afsnit 6.4)

Optimering

Optimering af software kode kan foregå på 2 niveauer. Først og fremmest er der optimering af algoritmer, hvor det alene er programmørens opgave at optimere algoritmerne. I relation til dette projekt er brugen af FFT algoritmen, fremfor DFT algoritmen, et godt eksempel på algoritmeoptimering. På et lavere niveau kan der laves optimering af assemblerkode. Dette foretages som regel at kompileren, men kan også gøres manuelt. Vi benytter i første omgang kompileren til optimering af assemblerkoden, hvorefter tidskritiske dele manuelt gåes igennem, med henblik på yderligere optimering. Kompileren kan optimere i flere trin, og bruger afhængig af det valgte trin forskellige optimeringsrutiner. Generelle optimeringsrutiner er fjernelse af unødvendig kode, herunder debugkode, samt erstatning af komplekse funktioner med ækvivalente funktioner, der er mere effektive med hensyn til størrelse eller tid. Mere komplekse optimeringsrutiner er eksempelvis optimering af eksekveringshastighed, på bekostning af størrelse af eksekverbar kode, og omvendt. Eksemplevis kan kompileren optimere ved at inline funktioner, det vil sige at mindre funktioner kopieres direkte ind, hvor de skal bruges, istedet for at blive kaldt. Gevinsten er hurtigere eksekvering, idet man undgår overhead ved funktionskald, ulempen er større mængde eksekverbar kode.

4.3 Tabelbrug

Følgende indeholder en beskrivelse af, hvordan vi fremstiller tabeller til tabelopslag.

4.3.1 Formål

Det primære formål ved at bruge tabeller er at nedsætte eksekveringstiden for tunge beregninger. Her tænkes specielt på floating point operationer, som ellers skulle foregå ved software emulering, da MC68331 ikke har FPU. Men også multiplikationer og divisioner, der forløber over mange clockcycles. Ulempen ved at bruge tabeller er begrænset opløsning, idet tabellen størrelsesmæssigt begrænses af mængden af tilgænglig RAM.

4.3.2 Tabelstørrelser

Størrelsen af de tabeller, der kan laves, afhænger som sagt af mængden af tilgængelig RAM. I vores tilfælde, hvor vi har 256kB til rådighed, eksklusiv den mængde, der går til OS, drivere og applikationer, kan der laves endimensionelle tabeller med en relativ stor opløsning, eksemplevis en 11 bit tabel med 2 gange 32 bit data, som vil komme til at fylde, $2048 \cdot 2 \cdot 32b = 131072b = 16kB$. Drejer det sig om todimensionelle tabeller, vil en 8 bit kvadratisk tabel med 8 bit data, optage følgende hukommelsesplads: $256^2 \cdot 8b = 524288bit = 64kB$. Tilsvarende vil en 9 bit kvadratisk tabel med 8 bit data optage 256kB, hvilket er mere end vi har til rådighed til tabelbrug, og vil derfor ikke kunne komme på tale.

4.3.3 Procedure

Selve proceduren, der skal gennemløbes inden tabellen kan tages i brug på MCU'en, består af en række trin. Først og fremmest skal tabellen beregnes. Til det formål laves et program til hver tabel, der beregner tabellen. Ved eksekvering af tabelberegningsprogrammet laves en rå datafil, der indeholder alle tabelværdier. Dette arbejde udføres på en PC, da MCU'en ikke har mikrokode, der direkte understøtter beregninger med floating points. Den rå data fil skal nu konverteres til assembler format, da det giver mulighed for at kompilere den, og få en objektfil, der kan linkes med det endelige program. Under konverteringen er det vigtigt at huske forskellen på Motorolas og Intels måder at beskrive ordlængder større end 8bit, f.eks. 16 bit ord.

Figur 4.3: Little endian vs. big endian

Intel bruger little-endian, mens Motorola bruger big-endian, det vil sige, at det er forskelligt i hvilken 8bit blok, man har MSB og LSB. Dette bør man være opmærksom på, når man laver konverteringen til assembler fil. Sidste led, inden den endelige linkning, er at assemble assemblerfilen med AS. Herefter har man en objektfil, der kan linkes med de resterende programdele, når den endelige S-RECORD laves.

Kapitel 5

MCU opbygning

I det følgende er MCU'en kort beskrevet, set fra et software mæssigt synspunkt. MCU'en består af 4 grundelementer:

- Central Processing Unit (CPU32)
- System Integration Module (SIM)
- Queued Serial Module (QSM)
- General Purpose Timer (GPT)

5.0.4 CPU32

CPU32 er en 32 bit mikroprocessor, dvs. at den internt arbejder med 32 bit data- og adressebus. CPU32 er baseret på MC68020 og er fuldt ud kompatibel med instuktionssættet til MC68000 og MC68010 samt størstedelen af instruktionerne til MC68020. Af speciel relevans for debugging, kan CPU32 bringes til at arbejde i en speciel Background Debugging Mode (BDM), hvor alle normale operationer suspenderes, for at tillade debugging komandoer fra et remote system. Denne mode er implementeret direkte i processorens mikrokode.

Registre

CPU32 har 8 32-bit dataregistre D[0:7] og 8 32 bit adresseregistre A[0:7], der kan bruges til generelle formål; derudover har den en 32-bit Program Counter (PC), hvoraf der reelt kun benyttes 24 bit, da det er størrelsen for det maksimale adresse rum. Et 16-bit Status Register (SR), der er opbygget som vist i figur 5.0.4.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T1	T0	S	0	0	IP2	IP1	IP0	0	0	0	X	N	Z	V	C

Figur 5.1: Opbygning af Status Register

Nederste byte er det såkaldte Condition Code Register (CCD), der indeholder information om sidst udførte aritmetiske eller logiske funktion på MCU'en. Øverste byte, kaldet system byte'n bestemmer, om MCU'en arbejder i supervisor eller user mode, og hvilket interrupt prioritetsniveau der arbejdes i, samt mulighed for operation i trace (singlestepping) mode. System byte'n kan kun ændres i supervisor mode, hvorimod CCD kan ændres i begge modes. Ved at udnytte supervisor / user mode, er det muligt at lave beskyttelse, idet man bl.a. kan begrænse hukommelses områder, så de kun kan tilgåes i supervisor mode. Yderligere beskyttelse understøttes via 2 forskellige stack pointere; i usermode er A7 User Stack Pointer (USP), og i supervisormode kan man både tilgå USP og en Supervisor Stack Pointer (SSP).

Instruktionssæt

Som sagt er CPU32 kompatibel med MC68000, og benytter derfor samme instruktions sæt, dog tager eksekveringen af operationerne en del færre clock cycles, idet CPU32, er optimeret til høj ydelse [15][s. 47]. Specifikationerne på instruktionerne henvises til CPU32 Reference Manual [13].

5.0.5 SIM

SIM'et står for integrationen mellem de enkelte dele af MCU'en. Dette inkluderer styring af ekstern bus, chip-selects og clock. Yderligere information kan findes i afsnit **??**.

5.0.6 QSM

QSM'et styrer 2 serielle interfaces, Queued Serial Peripheral Interface (QSPI) og Seriel Communication Interface (SCI).

5.0.7 GPT

GPT modulet består af en 11 kanals timer, der kan bruges som Programmable Interrupt Timers (PIT).

Kapitel 6

Opstartskode

6.1 Formål

At opsætte Motorola'ens konfigurationsregistre, vektor tabel, relokere kode og data fra ROM til RAM, initialisere OS'et og kalde main().

6.2 Konfigurationsregistre

Som omtalt i afsnit 5 består MCU'en af fire dele: CPU32, SIM, QSM og GPT. Disse skal hver især konfigureres for at sikre, at MCU'en befinder sig i den ønskede tilstand.

6.2.1 SIM

Formålet med System Integration Module er, som navnet antyder, at stå for integrationen mellem de enkelte dele i MCU'en. Til dette formål består SIM'et af fem enheder med følgende funktioner:

Enhed	Beskrivelse
System Configuration And Protection (SCAP)	Styring af Privilegie niveau, interrupts og generel opsætning af SIM
Clock Synthesizer (CLOCK)	Styring af clock
External Bus Interface (EBI)	Styring af perifer bus
Chip Select (CS)	Chip select af extern hardware
Factory Test (TEST)	Aftesting af MCU på fabrik

Tabel 6.1: System integration module

SCAP

I SCAP delen af SIM'et skal følgende sættes op:

- Mapping af SIM-registre
- Privilegie niveau
- Software Watchdog
- Opførsel under Freeze
- Interrupts

Mapping af SIM-registre Placeringen af SIM-registrene i hukommelsen kan styres med Module Mapping bitten. Enten ligger registrene som en 4kb blok fra adresse 0x7FF000 eller fra 0xFFF000. Vi har valgt at mappe registrene til 0xFFF000 (se figur 6.1). Bit'en sættes selvom den valgte indstilling er lig standardværdien, idet bitten kun kan ændres 1 gang. Derved er det ikke muligt for en eventuel fejl i en driver at ændre mappingen af SIM-registrene.

Privilegie niveau Med supervisor bitten kan det nødvendige privilegie niveau for adgang til SIM-registrene indstilles. Da vi kun ønsker at kunne tilgå hardwaren fra OS'et sættes privilegie niveauet til supervisor mode.

Software Watchdog Der er mulighed for at tilkoble en watchdog til systemet, hvis man ønsker at sikre sig mod hard- eller softwarerelaterede deadlocks. Da opdateringen af en watchdog nødvendigvis besværliggør systemet, og en deadlock i systemet ikke vil forårsage større skade, er watchdog'en slået fra.

Opførsel under freeze SIM indeholder to timere: watchdog'en og en intern bus monitor, som melder fejl hvis svartiden på bus'en har været for lang. Disse to timere kan enten standses når freeze bliver asserteret (når CPU32 går i background debugging mode), eller de kan fortsætte med at køre. Da det ikke er ønskværdigt at generere busfejl-exceptions under debugging, er timerne sat til at pause under freeze.

Interrupts Hvert modul der kan genere et interrupt har et Interrupt Arbitration Number (IARB), hvilket bruges til at skelne mellem flere moduler med samme prioritet.

CLOCK

CLOCK delen af SIM'et kan konfigureres på to måder:

- Referenceclock styrer intern frekvenssynthesizer med variabel clockfrekvens
- Ekstern clock styrer MCU frekvens direkte

Som beskrevet i afsnit 14.3, er løsning 2 valgt. Derfor er den interne frekvenssynthesizer slået fra.

EBI

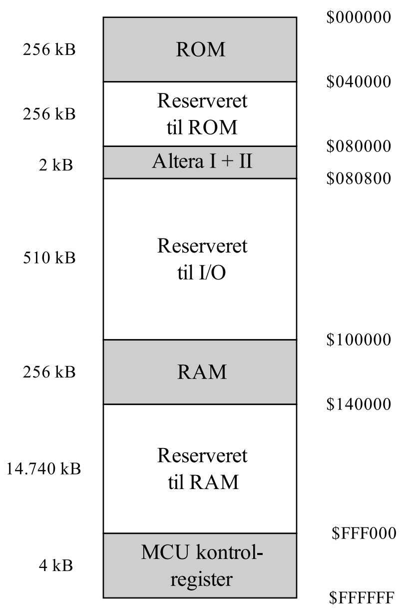
EBI'en forbinder den interne bus (IMB) til de perifære data- og adresseben. Denne del indeholder ingen konfigurationsregistre, og vil derfor ikke blive beskrevet yderligere.

CS

CS delen af SIM'et benyttes til at konfigurere hvor i adresseområdet eksterne enheder skal mappes ind, og hastigheden hvormed kommunikationen skal ske (antal waitstates).

Memorymap Memorymap'et er designet således, at man simpelt kan udvide mængden af RAM, ROM og tilføje eventuelle fremtidige udvidelser. Dette gøres ved at reservere ekstra adresserum mellem de placerede enheder.

ROM er placeret fra adresse 0x000000 til 0x03FFFF (256 kB), og der er reserveret yderligere 256 kB adresserum fra adresse 0x040000 til 0x07FFFF, for en eventuel fremtidig udvidelse af ROM'en.
RAM er placeret fra adresse 0x100000 til 0x13FFFF (256 kB). Desuden er der reserveret plads til yderligere udvidelser af RAM i adresseområdet 0x140000 til 0xFFEFFF. Ved adressen 0xFFF000 begynder MCU'ens konfigurationsregistre.
De to Alterakredse tilgåes med en enkelt chip-select, da de internt dekoder adressen. Man tilgår Alterakredsene vha. ganske få adresser, men da det mindste adresseområde for de programmerbare chipselects er på 2 kB, tildeles Alterakredsene adresseområdet fra 0x080000 til 0x0807FF.
Memorymappen kan ses på figur 6.1.
Det skal bemærkes, at MCU registerne bliver adresseret internt i MCU68331 og ikke bruger chip-select ben.
ROM'en er konfigureret til read-only på CSBOOT i både usermode og supervisormode, RAM'en bruger 3 chip-select 3,4 og 5, med adgang i både usermode og supervisormode, se afsnit 14.6.1. Alterakredsene bruger chip-select 9, og kun adgang i supervisormode.



Figur 6.1: Memorymap

Timing Den næste ting der skal sættes op er antallet af waitstates til den eksterne hardware. Ifølge afsnit 14.6.2 skal de konfigureres på følgende måde:

Enhed	Waitstates
ROM	1
RAM	0
DMA	13

Tabel 6.2: Waitstates for extern hardware

TEST

Testdelen skal ikke benyttes, og er derfor koblet fra.

6.2.2 QSM

QSM'et består af 2 serielle kommunikationsinterfaces, QSPI og SCI. Disse bliver ikke brugt i systemet, og er derfor slået fra.

6.2.3 GPT

GPT'en bliver anvendt som programmerbare interrupt timere (PIT). Disse bliver ikke brugt i systemet, og er derfor slået fra.

6.3 Exception vektor tabel

Ved opstart peger vector base registeret (VBR) på adresse 0, som er en adresse i ROM'en (se figur 6.1). Denne placering i ROM'en giver visse problemer:

- Da ROM'en er read-only, er der er lav konfigurerbarhed i systemet.
- Idet kode og data bliver relokeret er placeringen af koden ikke den samme under opstarten som efter. Dette gør, at man ikke kan lave en interrupt service rutine (ISR), der både virker fra ROM'en inden relokering og efter relokering i RAM'en.

Dette problem kan løses på 2 måder:

Løsning	Fordele	Ulemper
Alle interrupt rutiner ligger i ROM	Nemt at implementere	Langsomt (Se figur 6.2)
Vektortabel flyttes til RAM	Høj konfigurerbarhed	Der kræves 2 vektortabeller, en i ROM og en i RAM, for at sikre sig at interrupts håndteres planmæssigt under opstart

Tabel 6.3: Løsningsforslag til vektortabel

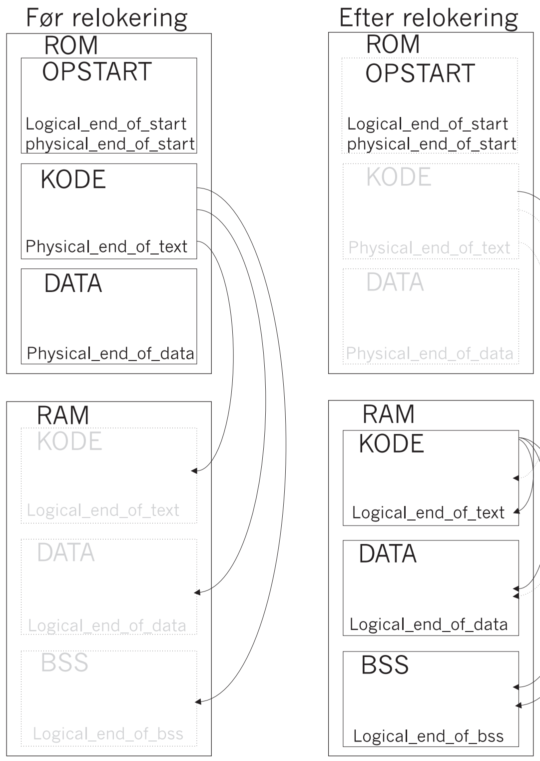
Af disse muligheder har vi valgt løsning 2, med 2 vektor tabeller. I ROM'en ligger der en minimumstabel, og efter relokeringen bliver der skiftet til den nye tabel i RAM.

6.4 Relokering

Da MCU'en starter med at læse fra ROM'en ville det simpleste være at lade program og data ligge i ROM'en. Dette giver dog flere problemer:

- Lav konfigurerbarhed - hvis vektortabellen ændres skal EPROM'en brændes om
- Initialiseret data vil ikke kunne overskrives. Det vil opføre sig som konstanter
- Hastighed - Som omtalt i afsnit 6.2.1 er tilgangstiden til ROM'en højere (1 waitstate) end ved RAM'en.

Derfor har vi valgt at relokere kode og data fra ROM'en til RAM'en under opstart. Relokeringen består i at få alle referencer til kode og data til at pege på de adresser, der relokeres til, dvs. i RAM'en.



Figur 6.2: Placering af kode og data før og efter relokering

Dette kan enten gøres runtime eller ved linktime via linkeren. Den simpleste mulighed af disse er linktime relokering. I GNU's linker har man mulighed for at give et ekstra tag til sine sections: AT(fysisk adresse). Med dette tag kan man få relokeringsadressen til være forskellig fra den fysiske adresse. Ud fra dette kan man lave et linkerscript, der placerer kode og data i ROM, men med logiske adresser som om de lå i RAM.

```
.startup romstart :
{
  boot.o
  logical_end_of_start = .;
  physical_end_of_start = .;
}

.text ramstart : AT (physical_end_of_start)
{
  *(.text)
```

```
    logical_end_of_text = .;
    physical_end_of_text = (logical_end_of_text - codestart)
                            + physical_end_of_start;
  }

  .data : AT (physical_end_of_text)
  {
    *(.data)
    logical_end_of_data = .;
    physical_end_of_data = (logical_end_of_data - codestart)
                            + physical_end_of_start;
  }

  .bss :
  {
    *(.bss)
    logical_end_of_bss = .;
    physical_end_of_bss = (logical_end_of_bss - codestart)
                           + physical_end_of_start;
  }
}
```

Som man kan se bliver opstartskoden (boot.o) ikke relokeret, men placeres først i ROM'en. Derefter følger resten af koden (*.text) med logiske adresser som om det er placeret i RAM'en. På grund af AT(physical_end_of_start) bliver det dog placeret lige efter opstartskoden i ROM'en. På samme måde følger data og bss (uinitialiseret data) (.bss segmentet placeres selvfølgelig ikke i ROM'en). Til hjælp for relokeringen defineres 2 labels for hver del. Den fysiske og den logiske adresse efter hver del. Disse benyttes i opstartskoden, hvor der kopieres (physical_end_of_data - physical_end_of_start) bytes fra adresse physical_end_of_start til adresse ramstart.

Kapitel 7

Operativ system

7.1 Opbygning

For at sikre systemets stabilitet og beskytte hardwaren mod uhensigtmæssige skrivninger/læsninger er applikationer placeret i usermode, og kan derfor ikke tilgå alle konfigurationsregistre i MCU'en og DMA controlleren (se afsnit 6.2.1.

7.2 Systemkald

Da OS'et kører i supervisormode, og applikationen kører i usermode, kan OS'et ikke kaldes direkte. Man kan ikke direkte skifte fra usermode til supervisormode, men man kan benytte en trapgate (TG), idet exceptionhandlere bliver udført i supervisormode. Denne TG kan så kalde systemfunktionen fra supervisormode og til sidst returnere til applikationen. Systemkaldet vil således bestå i at fremkalde en exception via trap instruktionen.
TG'en skal vide, hvilket systemkald vi ønsker at kalde, hvilket kan ske på to måder:

- Der benyttes en unik trap for hver systemfunktion
- Et trap bliver multiplexet til alle kald

Unik trap for hver systemfunktion

Man kunne benytte en unik TG for hver systemfunktion. Fordele: Simpel TG. Ulemper: Der er et begrænset antal traps til rådighed (16).

Et trap bliver multiplexet til alle kald

Man kunne benytte et enkel trap til alle systemkald og signalere, hvilken funktion man ønsker at kalde med et af MCU'ens registre. Dette register bruges derefter til at slå adressen på OS-funktionen op i en hop-tabel. Fordele: TG'en sætter ikke nogen begrænsning på antallet af mulige systemkald. Ulemper: TG'en bliver mere kompleks.

Valg af kaldemetode

Da løsningmulighed 1 begrænser mulighederne for fremtidige udvidelser, er løsningsmulighed 2 valgt. Dataregister d0 bruges til at signalere hvilken funktion, der ønskes kaldt, og system-trapnummeret er valgt til 13.

7.2.1 Parameteroverførsel

For at forstå problemerne bag parameteroverførsel til OS'et er det nødvendigt først at forstå konventionen for parameteroverførsel i C.

Parameteroverførsel i C

Konventionen for parameteroverførsel i C er:

- Parametre overføres via stacken.
- Parametre pushes på stacken fra højre mod venstre.
- Eventuel returværdi ligger i d0.
- Det er kalderens ansvar at fjerne parametre fra stacken efter kaldet.

Parameteroverførsel gennem TG'en kan ske på 3 måder:

- I registre.
- Kopiering af parametre fra USP til SSP.
- Skift af stackpointer.

I registre

Parametre kunne placeres i registre inden trap kald. Fordele: Hurtigt, idet RAM ikke skal tilgåes. Ulemper: Antal parametre kan ikke overstige antallet af registre, understøtter ikke variabel antal parametre, inkompatibel med C standard for parameteroverførsel.

Kopiering af parametre fra USP til SSP

Et antal parametre kunne kopieres fra USP til SSP. Fordele: Kompatibel med C standard for parameteroverførsel. Ulemper: TG skal kopiere et antal parametre afhængigt af hvilket systemkald, der kaldes, og skal derfor have specialtilfælde for alle systemkald. Variabel antal parametre understøttes ikke. Langsomt, pga. ekstra parameterkopiering.

Skift af stackpointer

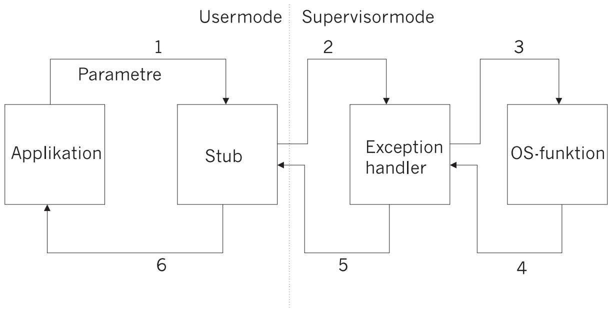
USP kunne kopieres til SSP. Fordele: Kompatibel med C standard for parameteroverførsel. Variabel antal parametre understøttes automatisk. Ulemper: Komplekst at gøre OS re-entrant, idet SSP skal gemmes i TG (i global variabel).

Valg af parameteroverførselsmetode

Idet løsningsmulighederne 1 og 2 har store ulemper, det være sig i overhead eller kompleksitet, er løsningsmulighed 3 valgt. Problemet med at gøre kaldet re-entrant er ikke stort i dette system, idet OS'et er et singletaskingssystem. Kommunikationen driverne imellem kan løses ved at kompromittere OS'ets modulopbyggethed, og lade de enkelte systemfunktioner kalde hinanden direkte.

7.2.2 Trapgate

Efter dette kan TG'en designes og implementeres.
Systemkaldet fungerer ved, at der findes en procedure (en stub) i usermode, som applikationen kalder. Denne procedure tager de samme parametre som systemkaldet, men inderholder kun et kald af exceptionhandleren, som så igen kalder OS-funktionen (se figure 7.1). Brugen af denne stub har den funktion, at OS'et kan kaldes direkte fra et C program.



Figur 7.1: Rækkefølge i OS kald

Ved indgangen af TG'en ser usermode-stacken således ud:

Figur 7.2: Usermode stack ved indgangen til TG

I TG'en skal følgende funktioner udføres:

- Skift af stack
- Ændring af returadresse og kald af OS-funktion
- Genindsættelse af returadresse
- Skift af stack og afbrydelse af exceptionhandler

Skift af stack

Supervisor stacken skal gemmes i hukommelsen, for at denne senere kan genetableres. Dernæst skal indholdet af USP kopieres over i SSP, således at stub'ens parametre kan tilgåes.

Ændring af returadresse og kald af OS-funktion

Toppen af stacken indeholder stub'ens returadresse. Ved at fjerne denne adresse og kalde OS-funktionen fra exceptionhandleren vil parametrene være korrekt placeret på stacken (se figur 7.2). Adressen på OS-funktionen findes ved at slå op med d0 i hop-tabellen.

Genindsættelse af returadresse

Efter kaldet af OS-funktionen skal returadressen genindsættes på stacken, således at stub'en kan returnere til den korrekte adresse.

Skift af stack og afbrydelse af exceptionhandler

Til sidst skal supervisor stacken hentes ind fra hukommelsen igen, således at exceptionhandleren kan returnere korrekt.
Den endelige exceptionhandler ser således ud.

```
move.l  %ssp,(save_supervisor_stack) /* gem supervisorstack  */
move.l  %usp,%ssp                    /* skift stack          */
move.l  (%sp)+,(save_return_address) /* gem returadresse     */

lea.l   os_jumptable,%a0             /* indlæs startadresse  */
move.l  (%a0,%d0*4),%a1              /* beregn adresse       */
jsr     (%a1)                        /* hop til OS rutine    */

move.l  (save_return_address),-(%sp) /* gendan returadresse  */
move.l  (save_supervisor_stack),%sp  /* gendan stack pointer */
rte                                  /* returner fra trap    */
```

7.2.3 Performance

Brugen af en TG giver et vist overhead i forhold til et almindeligt procedurekald. Dette overhead er $C_{trap} + C_{TG}$ hvilket ca. bliver til $40 + 100$ ektra clockcycles. På grund af dette overhead er OS-rutinerne og de hardwarespecifikke drivere designet således, at de arbejder på hele blokke af gangen, idet det minimerer antallet af systemkald, og dermed overhead'et.

7.3 Funktioner

OS'et stiller funktioner til at sætte og hente adressen på en given interrupt rutine (se appendiks E.2). Disse funktioner er kun tilgængelig for drivere, og ikke applikationer der kører i usermode, idet TG'en ikke tillader at disse funktionskald passerer.

Kapitel 8

Displaymanager

8.1 Formål

Formålet med displaymanageren er at håndterer det hardwarespecifikke interface til det tilsluttede LCD display. Desuden tilbyder det højniveau kommandoer til at styre output på LCD (printf m.m.), hvor det ikke er nødvendigt at kende til specifikationerne af LCD displayet.

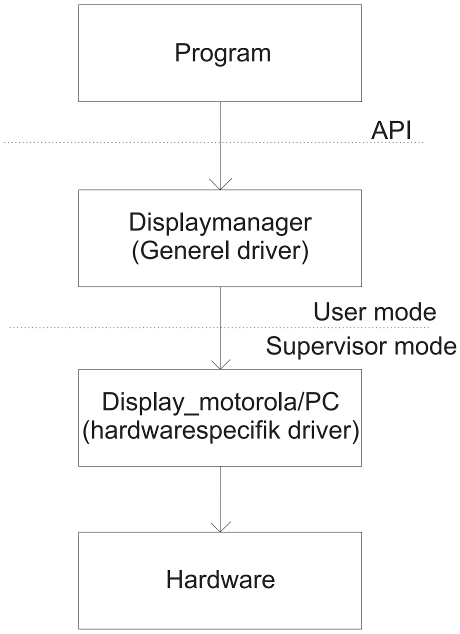
8.1.1 Opbygning

Displaymanageren er opdelt i 2 dele:

- øverst er der en generel del som står for kommunikationen med programmet gennem det definerede API (se appendiks E.3).

- nedenunder dette lag findes der en hardwarespecifik del som står for den direkte kommunikation med hardwaren. Dette lag findes i to udgaver, en til PC og en til MCU'en.

Det vil sige, at programmet kun snakker med den øverste 'generelle' del af driveren. Det er kun den nederste del, der er hardwarespecifik og derfor skal ændres, hvis hardwaren ændres.



Figur 8.1: Opbygning af displaymanageren

8.1.2 Overvejelser

Displaymanageren skal kunne udskrive både tekst og grafik. Tekst skal primært bruges under debugging, men er også vigtig hvis der senere skal implementeres brugerinterface på MCU'en. grafikudskriving skal primært bruges til at udlæse de beregnede frekvenskomposanter gennem LCD-klienten.
For at hastighedsoptimere udskriften er der foretaget følgende valg:

- Idet displayet er kompliceret (og dermed langsomt) at kommunikere med, benyttes der DMA overførsel af skærmdata fra systemhukommelsen til displayet's hukommelse (se afsnit 16.7).
- For at kunne arbejde på nye skærmdata mens de nuværende skærmdata bliver overført til displayet, benyttes der doublebuffering (desuden bevirker det at vi kun viser skærmbilledet når det er tegnet helt færdigt, dvs. ingen flimmer)

Doublebuffering giver dog et problem med tekstudskrift. I modsætning til grafikudskrift (hvor der beregnes et helt skærmbillede af gangen, hvorefter det vises på skærmen) vil man gerne kunne se teksten, lige så snart den bliver skrevet til skærmen. Dette kan løses på forskellige måder:

1. Skærmbufferen overføres til skærmen for hvert nyt tegn. Fordele: Skærmen bliver opdateret, lige så snart der bliver skrevet til den. Ulemper: pga. skærmens langsomme opdateringstid er der stort overhead ved tekstudskrivning og kraftig begrænsning på udskriftshastighed (ved det anvendte display: max 11 tegn/sec).

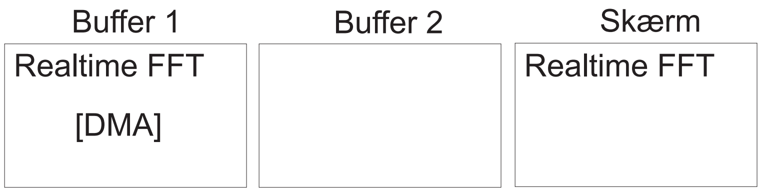
2. Skærmbufferen overføres til skærmen efter denne er blevet fyldt. Fordele: Lille overhead. Ulemper: Meget langt interval mellem opdatering af skærm, egner sig ikke til et realtidssystem som vores. Besværliggør software til udlæsning.

3. Skærmen opdateres ved hvert \n(newline) tegn. Fordele: Mindre overhead end ved (1) Ulemper: Skærmen bliver ikke opdateret ved hvert tegn (denne fremgangsmåde er almindelig i *NIX systemer)

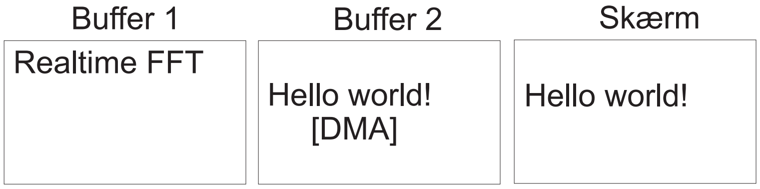
4. Skærmopdatering styres af specialtegn/kommando: Fordele: Programmet kan selv styre hvor ofte skærmen opdateres. Ulemper: API bliver mere kompliceret.

5. Der skrives direkte til skærmen udenom DMA. Fordele: Skærmopdatering sker øjeblikkelig. Ulemper: Komplicerer hardware. Kompliceret at udskrive grafik samtidig med tekst. Meget større overhead end ved DMA løsning.

Fælles for løsning 1-4 gælder det, at doublebufferen skaber problemer med skærmopdateringen. Forstiller man sig, at man vil skrive 2 linjer:
linje1 = ’Realtime FFT’
linje2 = ’Hello world!’
vil de 2 buffere se således ud i de 2 situationer (1=mellem linje 1 og 2, 2=efter udskrift)

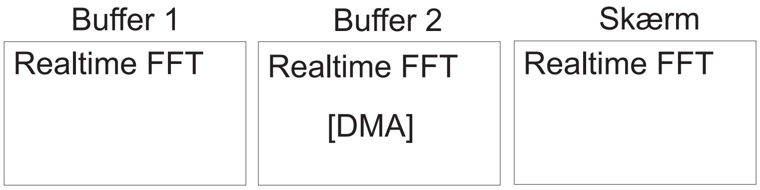


Figur 8.2: buffer 1 overføres til skærm via DMA



Figur 8.3: buffer 2 overføres til skærm via DMA

Efter første linje (se figur 8.2) bliver buffer 1 sendt til skærmen via DMA, og der skiftes til buffer 2. Derefter skrives næste linje til buffer 2, hvorefter buffer 2 sendes til skærmen. Dette sletter dog linje 1. Problemet er, at vi skifter skærmbuffer for hver opdatering af skærmen. Problemet kan løses på 2 måder. Enten kopierer man indholdet af den nuværende skærmbuffer over i den anden skærmbuffer, og lader så DMA kontrolleren opdatere skærmen fra buffer 2 istedet for buffer 1.
På den måde vil forløbet se således ud:



Figur 8.4: buffer 2 overføres til skærm via DMA

Figur 8.5: buffer 2 overføres til skærm via DMA

Denne metode løser vores problem, men den giver desværre et overhead ved tekstudskrivning. En anden metode kunne være at gøre hardwaren mere avanceret. Man kunne have et flag, der fortæller, hvilke linjer der er blevet ændret, og dermed skal opdateres. Dette kunne gemmes som en bitmaske i en byte (idet der kan være 8 tekstlinjer på skærmen (hvert tegn er 8x8 pixel)), som kunne sendes til vores DMA kontroller eller placeres et foruddefineret sted i hukommelsen forud for en DMA overførsel.

Når der tegnes grafik skal hele skærmen selvfølgelig opdateres (idet der tegnes et helt skærmbillede af gangen).
Af de 5 muligheder har vi valgt at implementere løsning 3 (opdatering for hver newline), med avanceret hardware (dvs kun opdatering af ændrede linjer).

8.2 Implementation

Under implementationen af funktioner til displaymanageren er der lagt vægt på at lave simple funktioner, der i så høj grad som praktisk muligt ligner de kald som normalt findes til tekst- og grafikudskrivning på en standard PC, uden at det går ud over eksekveringstiden. For at optimere eksekvenringstiden er meget funktionalitet placeret i displaymanageren, og ikke i den hardwarespecifikke driver, idet man på den måde minimerer overhead'et ved OS-kald. Funktionerne kan deles op i 3 dele (se appendiks E.3)

- Generelle funktioner.
- Tekstrelatered funktioner.
- Grafikrelatede funktioner.

8.2.1 Generelle funktioner

Af generelle funktioner er der implementeret initialiserings- og deinitialiseringsfunktioner, som henholdsvist initialiserer og deinitialiserer displaymanageren og den hardwarespecifikke driver. Desuden er der implementeret en funktion til at slette indholdet af skærmen, og en funktion til at fremtvinge en opdatering af skærmen.

8.2.2 Tekstrelaterede funktioner

Til at udskrive tekst er der implementeret en standard printf funktion, med den relevante funktionalitet (udskrivning af floating-point tal er ikke understyttet)

8.2.3 Grafikrelatede funktioner

2 funktioner er implementeret til at udskrive grafik: En generel putpixel, og den dedikeret funktion til at udskrive lodrette streger (bars) til brug i LCD-klienten.

8.3 hardwarespecifik driver

Den hardwarespecifikke driver er lavet så simpel som muligt. Mest mulig funktionalitet er istedet placeret i den generelle driver, displaymanageren. Til PC implementationen er der brugt grafiksystemet OpenPTC ([21]).

- Generelle funktioner.
- Tekstrelaterede funktioner.
- Grafikrelaterde funktioner.

8.3.1 Generelle funktioner

De generelle funktioner består af funktioner til at initialisere og deinitialisere (init og deinit), funktioner til at tænde og slukke displayet (on og off) og til at opdatere skærmen (updatescreeen)

8.3.2 Tekstrelatede funktioner

De tekstrelatede funktioner i den hardwarespecifikke driver består af funktioner til styring af cursoren (gotoxy, getxpos, getypos og scroll), og til simpel udskriving af enkelttegen (putchar).

8.3.3 Grafikrelaterede funktioner

Den hardwarespecifikke driver indeholder ingen primitiver til udskriving af grafik, men en mulighed for at få adressen på den aktive skærmbuffer (getscreenbuffer).

Kapitel 9

Samplemanager

9.1 Formål

Sampledriverens primære opgave at håndtere det hardwarespecifikke interface til sample DMA kontrolleren, og stille samplet data til rådighed for FFT-serveren.

9.2 Opbygning

For at benytte princippet i et operativ systems user / supervisor mode opbygning, som beskrevet i afsnit 3.2, bør sampledriveren opbygges som en lavniveaudriver, der kører i supervisormode. Alle funktionskald skal gå gennem en samplemanager, der kører i user-mode. På denne måde bliver den underlæggende hardware transparent, det vil sige at applikations programmøren ikke behøver at kende noget til hardwaren, eller tænke over, hvordan man specifikt kommunikerer med denne.

9.3 Funktion

Selve overførslen af data fra A/D-konverterens latch til RAM'en, sker ved DMA. Derfor skal samplemanageren kunne styre DMA overførslen, det vil sige mængden af data, der skal kopieres inden der sendes interrupt, og hastigheden, der skal samples med, kaldet samplerat, samt data'ens beliggenhed i rammen, altså fra hvilken adresse DMA'en skal overføre til. Ydermere bør man lave en funktion, der har til opgave at checke om A/D-konverteren har overført sekvensen af data, eller om den stadig arbejder.

For at kunne teste systemet laves der først en prototype på PC. Dog er der stor forskel på hvordan, softwaren skal implementeres på de 2 platforme, idet PC'en og Motorola'ens hardware ikke skal tilgåes på samme måde. Dette gøres ved at implementeres specifikke lavniveaudrivere til de 2 platforme. PC-prototypen af sampledriveren skal kun implementeres i sådan en grad, at resten af systemet kan testes. Derfor DMA-kontrollen blive emuleret via en datafil, der indeholder de enkelte samples.

9.4 Implementation

Sampledriveren skal implementeres forskelligt afhængig af den underlæggende hardware, derfor implementeres højniveau funktionerne fra samplemanageren først. Samplemanageren har 4 grundfunktioner. Samplemanager_init til initialisering. Samplemanager_start til start og styring af DMA overførsel. Samplemanager_wait_for_new_data, der skal checke for ny data fra A/D-konverteren, og til sidst samplamanager_deinit, der deinitialiserer samplemanageren.

9.4.1 Samplemanager_init

Initfunktionen, samplemanager_init, bruges til at initialiserer samplemanageren. Den kaldes med parameteren samplerate, der bestemmer, hvilken hastighed A/D-konverteren skal sample med. Samplemanageren afgør herefter, hvilken lowlevel driver, der skal kaldes, afhængig af om compiler optionen TARGET sættes til _PC_ eller _MOTOROLA_. For TARGET=_PC_ åbnes en datafil, der indeholder en række samples, derfor benyttes samplerate ikke. For TARGET=_MOTOROLA_ programmeres DMA kontrolleren med den givne samplefrekvens.

9.4.2 Samplemanager_transfer

Samplemanager_transfer funktionen tager 2 parametre: Numberofsamples, til bestemmelse af mængden af samples, DMA'en skal overføre fra A/D-konverteren til en buffer i RAM'en, og en pointer *buffer til et buffer-array med størrelsen Numberofsamples * 4 bytes, idet samples læses ind i 32bit ord, da det er formatet fftserveren benytter. (se afsnit 10) Igen er der forskel på om den underlæggende platform (TARGET) er _PC_ eller _MOTOROLA_, derfor skelnes der i det følgende mellem de 2.

- _PC_
  Samplemanager_transfer sender kaldet videre til lowleveldriveren sample_pc, med parameterne numberofsamples og buffer. Herefter allokerer lavniveaudriveren hukommelse i PC'ens RAM til de samples, der læses fra datafilen: diskbuffer, samt en midlertidig buffer til permutering og pakning: tempbuffer. Datafilen læses som en blok over i diskbufferen, og konverteres, afhængig af om FFT'en skal regnes med fixed eller floating point, til den valgte type, og lægges i tempbuffer'en. Diskbuffere'en frigives og der laves bitreversering (permutering) og pakning i tempbufferen, som beskrevet i appendiks A om FFT teori. Tilsidst frigives diskbuffer'en, og der returneres til applikationen.

- _MOTOROLA_
  På Motorola'en sendes kaldet videre til lowleveldriveren sample_motorola, som kalder videre til mikro kernen, med samme parametere som til _PC_. I den hardwarespecifikke del programmeres DMA kontrolleren med de givne data, hvorefter der returneres til applikationen, således at programmet kan køre videre mens DMA kontrolleren overfører samples til RAM'en. Som beskrevet i afsnit 16.6.2 om permutering, bliver alle data hardware permuteret og pakket og kommer til at ligge i denne rækkefølge i RAM'en.

9.4.3 Samplemanager_wait_for_new_data

Samplemanager_wait_for_new_data kaldes for at tjekke om A/D-konverteren har nye data klar. Funktionen er implementeret som en while-løkke og poll'er derfor konstant den hardwarespecifikke driver for nye data, hvorfor der laves et timestrap, i form af en timeoutværdi. While-løkken består af et kald til lowleveldriveren sample_is_data_ready, der udføres op til SAMPLEMANAGER_TIMEOUTVALUE gange. SAMPLEMANAGER_TIMEOUTVALUE er den maksimale tid, der kan gå før alle data er klar på latchen, givet ved maksimal bloksize og minimal samplerate. $MAXTIME = \frac{BLOKSIZE_{max}}{SAMPLERATE_{min}} = \frac{2048}{11K} = 0.19s$. Størrelsen på SAMPLEMANAGER_TIMEOUTVALUE skal da være den tid det tager at gennemløbe while-løkken for at få en tid, svarrende til MAXTIME. Sample_is_data_ready implementeres forskelligt afhængig af underlæggende hardware.

- _PC_
  Implementationen af sample_is_data_ready på _PC_ returnerer altid SAMPLE_DATA_READY,

da det ikke bliver aktuelt at vente på nye data.

- _Motorola_
  funktionen sample_is_data_ready på _MOTOROLA_ tjekker for om et flag er blevet sat. Dette flag bliver sat af en ISR rutine, der bliver trigget af DMA-kontrolleren hver gang den er færdig med at overføre en blok samples.

9.4.4 Samplemanager_deinit

Samplemanager_deinit funktionen kaldes, når man ønsker at stoppe sampleprocessen. På samme måde som med de 2 andre funktioner kalder denne funktion videre til henholdsvist sample_pc og sample_motorola. Arbejdes der på _PC_ lukkes data-filen med samples. På _MOTOROLA_ er deinit funktionen ikke implementeret, idet sampledelen aldrig skal stoppes.

Kapitel 10

Implementation af FFT-server

10.1 Formål

Formålet med FFT-serveren er, at konvertere de samplede tidssignaler fra A/D-konverteren til frekvenskomposanter.

10.2 Opbygning

FFT-serveren indeholder 3 funktioner, fftserver_init og fftserver_deinit, der henholdsvis initialiserer og deinitialiserer serveren, samt beregnings-algoritmen, fftserver_calcfft.

10.3 Metode

FFT-algoritmen kan implementeres rekursivt eller iterativt. I en rekursiv implementation kalder funktionen sig selv $2^{(log_2(N)+1)} - 2$ gange, hvor N betegner antallet af punkter i FFT'en. Det giver et overhead sammenlignet med den iterative implementation, idet funktionen, ved hvert kald af sig selv, udfører en række instruktioner. Tabel 10.1 giver et overslag over hvilke instruktioner, der er relateret til et rekursivt kald, og hvor mange clock-cycles disse strækker sig over.
Lægger man hertil, at funktionen har en initialiseringsdel, der også skal udføres for hvert kald af funktionen, er det ikke urimeligt at antage at antallet af instruktioner, der er relateret til det rekursive kald, er ca. 100. For en 1024-punkts FFT betyder det $(2^{(log_2(1024)+1)} - 2) * 100 = 204600$ clock-cycles. Med en 16 MHz processor er det ca. $12,8$ ms, hvilket er

Instruktion	Antal clock-cycles
LINK	12
MOVE.L (fra stack)	10
MOVE.L (til stack)	10
JSR fft	17
UNLK	11
RTS	16
Ialt	76

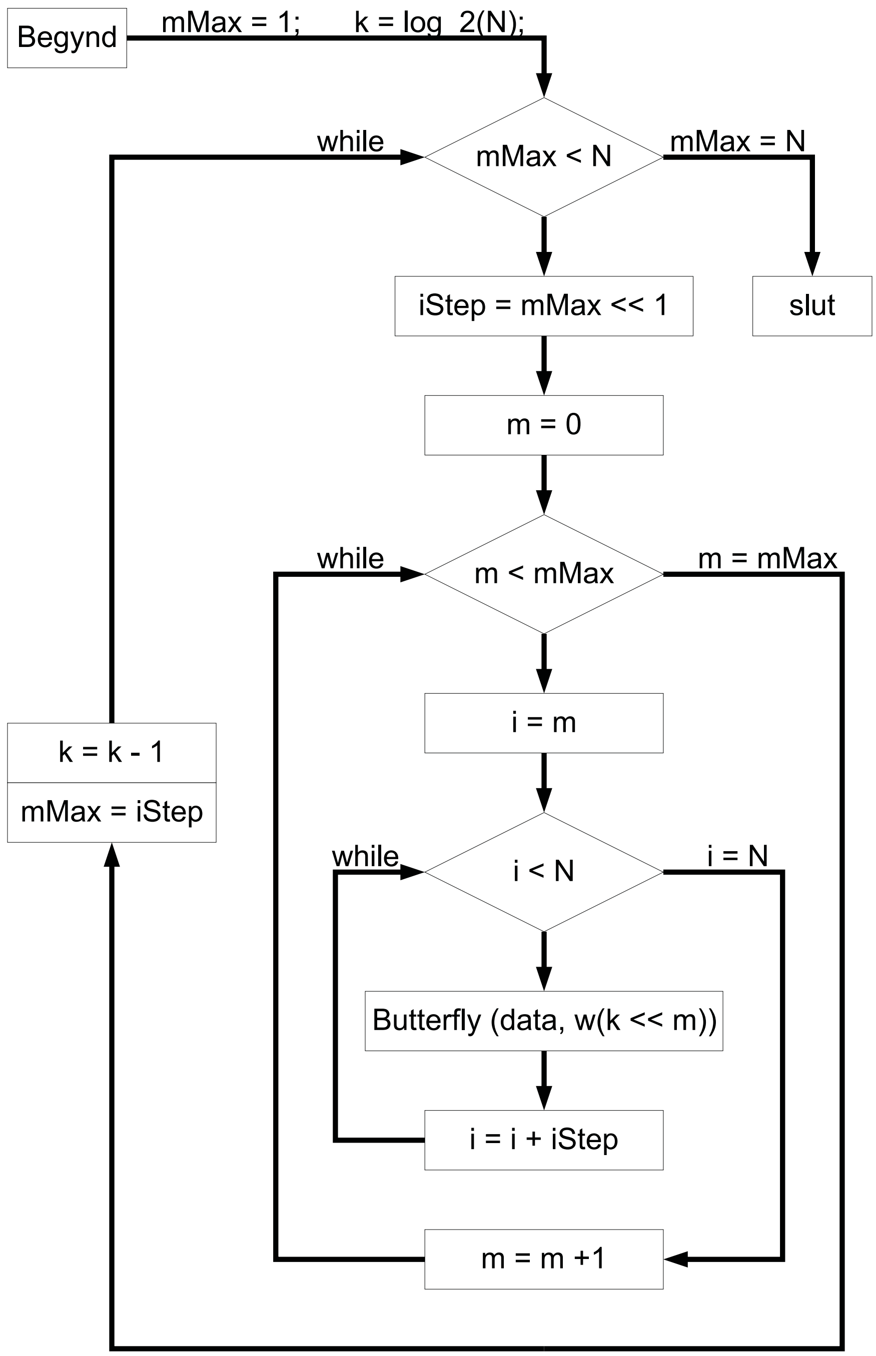
Timing data er baseret på 3-clock reads og writes. [13, s. 8-16 - 8-28]

Figur 10.1: Overslag over antallet af clock-cycles, der medgår til et rekursivt kald

et betragteligt overhead, idet FFT'en skal beregnes i løbet af det tidsinterval, der er mellem 2 opdateringer af displayet (se kapitel 3), ca. 100 ms. Disse overvejelser motiverer valget af en iterativ implementation af FFT'en.

10.4 Beregning

Som beskrevet i appendix A har FFT'en to sektioner. Den første sektion sorterer dataene i bit-reverseret orden. Dette trin udføres i hardwaren og vil ikke blive beskrevet her.
Den anden sektion har i den iterative implementation en ydre løkke, der gennemløbes $log_2(N)$ gange. Ved hvert gennemløb beregnes transformationer af længden $2, 4, 8, ..., N$. To indre indlejrede løkker håndterer beregningen af de mindre transformationer, således at den ydre kontrollerer antallet af iterationer, $1, 2, 4, ..., N/2$, der er nødvendige for at generere inputvektoren til næste stadie. Den indre løkke beregner den grundlæggende to-punkts transformation (butterfly). Nedenfor ses et rutediagram for den centrale del af FFT-rutinen.



Figur 10.2: Rutediagram, der beskriver den centrale del af FFT-algoritmen

Rutinen udfører en in-place beregning af en kompleks output-vektor med længden N. Efter beregningen indeholder data[0] DC-frekvenskomposanten. Derefter følger de positive frekvenskomposanter i stigende rækkefølge. Data[$N/2$] indeholder den mest positive

komposant (der samtidig er den mest negative, da frekvensspektret er periodisk), hvorefter de konjugerede negative frekvenskomposanter følger i stigende rækkefølge.
w[$m << k$] er et tabelopslag af twiddle-faktoren, $W(j\omega)$. På den måde undgår vi det overhead, der er forbundet med at skulle udregne denne faktor efter hvert gennemløb af m-løkken.

10.5 Optimering

10.5.1 Transformation af to relle funktioner samtidigt

Det er muligt at optimere beregningen. I vores system er input en sekvens af samplede reelle værdier. I ovenstående rutine behandles input som en sekvens af komplekse værdier med imaginærdelene sat til 0. Dette er ineffektivt med hensyn til udførselstid og udnyttelse af hukommelsen. Det er muligt at øge effektiviteten ved at udnytte symmetrien, der gælder for Fourier-transformationen af reelle funktioner,

$$F_{N-n} = F_n^*, \tag{10.1}$$

hvor stjernen betegner kompleks konjugation. Tænker man sig nu en diskret Fourier-transformation af en funktion, der udelukkende indeholder imaginære værdier, vil denne transformation opfylde følgende symmetri:

$$G_{N-n} = -G_n^* \tag{10.2}$$

[5, s. 511]. To reelle sekvenser kan derfor pakkes som henholdsvis den relle og imaginære del af en enkelt sekvens af data. Den resulterende transformation kan dernæst pakkes ud ved hjælp af de to symmetrier.

10.5.2 Transformation af enkelt reel funktion

For at undgå den redundans, der er forbundet med at transformere to sekvenser på samme tid kan man opdele input-sekvensen i to, og dernæst pakke de to halvdele som henholdsvis relle og imaginære dele i et komplekst array med den halve længde. Antager man at denne opdeling foregår på en sådan måde, at de lige data pakkes i den relle del af det komplekse array, og de ulige data pakkes i den imaginære del, $f_{2j} + if_{2j+1}$, $j = 0, ..., N/2 - 1$, vil resultatet af transformationen blive en kompleks array $H_n = F_n^{lige} + iF_n^{ulige}$, $n = 0, ...N/2 -$

1, hvor

$$F_n^{lige} = \sum_{k=0}^{N/2-1} f_{2k} e^{2\pi ikn/(N/2)} \tag{10.3}$$

$$F_n^{ulige} = \sum_{k=0}^{N/2-1} f_{2k+1} e^{2\pi ikn/(N/2)}. \tag{10.4}$$

Ved hjælp af ovennævnte symmetrier er det nu muligt at udskille de to transformationer, F_n^{lige} og F_n^{ulige}. Følgende sammenhæng (se ligning A.24 i Appendiks A) angiver hvorledes de to transformationer kan arbejdes sammen til en enkelt transformation, F_n:

$$F_n = F_n^{lige} + e^{2\pi in/N} F_n^{ulige}, \qquad n = 0, ..., N-1. \tag{10.5}$$

Ønsker vi at udtrykke denne sammenhæng ved H_n bliver resultatet:

$$F_n = \frac{1}{2}(H_n + H_{N/2-n}^*) - \frac{i}{2}(H_n - H_{N/2-n}^*)e^{2\pi in/N}, \qquad n = 0, ...N-1 \tag{10.6}$$

[5, s. 512]. Da $F_{N-n}^* = F_n$ er det ikke nødvendigt at gemme hele spektret. Det er tilstrækkeligt at gemme den positive frekvens-halvdel. Denne kan lagres i samme array som de oprindelige data, d.v.s. operationen kan foregå in-place.

10.6 Fixed-point FFT

10.6.1 Generelle overvejelser

68331'eren har ikke nogen hardware-understøttelse af floating-point. Det er muligt at understøtte floating-point med software, men det er ineffektivt, hvorfor vi har valgt at lave en fixed-point implementation. Dette nødvendiggør bl.a. en række overvejelser vedrørende data-repræsentation, således at overflow undgåes og der opnåes størst mulig nøjagtighed i beregninger med endelig ordlængde. Endvidere er det nødvendigt at programmet holder styr på kommaerne i beregningerne.

10.6.2 Skaleringsfaktor

Reelle og imaginære tal er i beregningen repræsenteret i 2's-komplement form,

$$x = Y_m(-b_0 + \sum_{i=1}^{31} b_i 2^{-i}) \tag{10.7}$$

[23, s. 329]. b_i er enten 0 eller 1 og b_0 er sign bit'en, der er 0 for positive tal og 1 for negative tal. Y_m angiver skaleringsfaktoren, og skal bestemmes således, at vi minimerer fejl fra kvantisering og undgår overflow.
Kommatalsdelen af x kan repræsenteres ved følgende notation

$$x_{31} = b_0 \diamond b_1 b_2 b_3 ... b_{31}, \tag{10.8}$$

hvor $\diamond$ repræsenterer det binære komma [23, s. 329].

10.6.3 Maximumværdi

Da vi arbejder med et 10-bit system vil alle input-værdier være repræsenteret i intervallet $\{-512, ...511\}$.
Antager vi at input-sekvensen er $x[n] = konstant$, for alle n, vil den Fourier-transformerede,

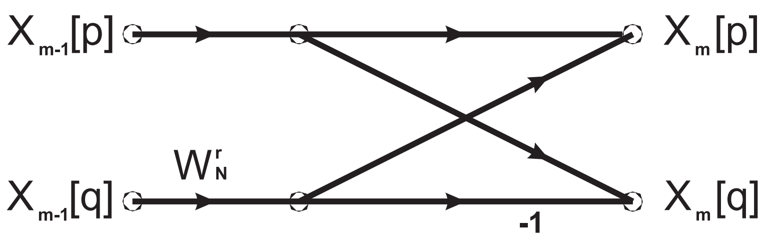
$$X[k] = \sum_{n=0}^{N-1} x[n] W(N)^{kn}, \tag{10.9}$$

blive $konstant * N$, og denne værdi vil være repræsenteret ved DC-komposanten i frekvensspektret, hvor $k = 0$. For alle andre værdier af k, er $X[k] = 0$. Den teoretiske baggrund for denne sammenhæng er, at det er muligt at fortolke den Fourier-transformerede, $X(e^{j\omega})$, af $x[n] = konstant$, for alle n, som et periodisk impulstog, der har uendelig højde, bredden 0 og arealet 1 [23, s. 50]. Størrelsen af $X[0]$ vil derfor gå mod uendelig, for N gående mod uendelig.
Den grundlæggende to-punkts DFT beregning for en radix-2 DIT algoritme har formen:

$$X_m[p] = X_{m-1}[p] + W(N)^r \cdot X_{m-1}[q], \tag{10.10}$$

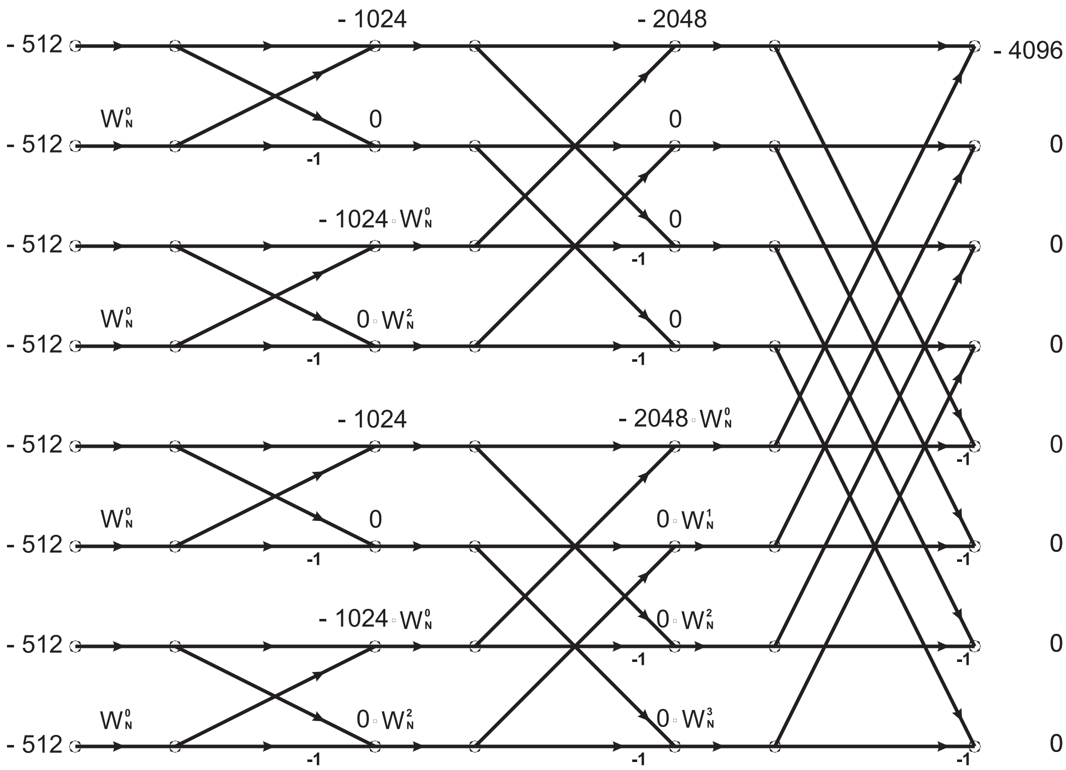
$$X_m[q] = X_{m-1}[p] - W(N)^r \cdot X_{m-1}[q]. \tag{10.11}$$

m og $(m-1)$ refererer til henholdsvis den nuværende og forrige vektor i beregningen i et bestemt stadie. $m-1 = 0$ refererer til input-vektoren, $x[n]$, og $m = log_2(N)$ refererer til output-vektoren. p og q udtrykker placeringen af de enkelte elementer i vektorerne.
Figur 10.3 er en grafisk afbildning af den centrale 2-punkts-transformation.



Figur 10.3: Flow graf for den grundlæggende buttefly beregning.

Antager vi nu, at værdien af både $X_0[p]$ og $X_0[q]$ har værdien -512, for alle p og q, og at $W(N)^r$ har maksimum-værdien 1, vil $|X_m[p]|$ maksimalt kunne fordobles i beregningen af hvert de $log_2(N)$ stadier. $|X_{log_2(N)}[p]|$ kan derfor maksimalt blive $|512 * 2^{log_2(N)}| = |512 * N|$.
Figur 10.4 beskriver ovenstående, eksemplificeret ved $N = 8$.



Figur 10.4: Flow graf for en 8-punkts DFT, der beskriver et eksempel på, at output af FFT'en antager sin absolutte maximum værdi.

10.6.4 Skalering af input-værdier

Som beskrevet i Appendix 10 kan sammenhængen mellem DFT'en og den inverse DFT udtrykkes som

$$X[n] = Nx[-k]. \tag{10.12}$$

Ønsker man at genskabe de oprindelige input-data, udfører man blot en indeks-reverseret FFT på $X[n]$, og skalerer med $\frac{1}{N}$.
Det er også muligt at skalere input-data med $\frac{1}{N}$ inden Fourier-transformationen. Man vil i dette tilfælde stadig have information om det relative indhold af hver frekvens-komposant. Ønsker man at genskabe den oprindelige input-vektor, udfører man nu blot en indeks-reverseret FFT på sekvensen af frekvens-komposanter, uden at skalere med $\frac{1}{N}$.
Fordelen ved denne fremgangsmåde er i vort tilfælde, at vi kan skalere vores fixed point efter en absolut maximum-værdi i beregningen på $|512|$. En skalering af input-vektoren øger ganske vist kvantiserings-støjen, idet denne akkumuleres gennem FFT-beregningerne. Til gengæld øges nøjagtigheden i beregningerne, da den mindste forskel mellem tal i beregningerne er givet ved

$$\Delta = Y_m 2^{-31} \tag{10.13}$$

[23, s. 329]. Samtidig øges beregningshastigheden, idet programmet efter hver multiplikation skal skifte et antal gange, svarende til heltalsdelen. Vi har derfor valgt at skalere input-data med $\frac{1}{N}$.

10.6.5 SWAP

I MCU'ens instruktions-sæt findes der en instruktion, SWAP, der ombytter to halvdele af et 32 bit dataord. Den anvendes bl.a., når der skiftes 16 bits. Det betyder, at det er mere effektivt at skifte 16 bits frem for et vilkårligt antal bits (16 undtaget). Ønsker vi at skifte et vilkårligt antal bits, skal der udføres 2·MOVEQ.L (3), 2·LSd (11), og en OR.L (3). (Tallene i parentes angiver antallet af clock-cycles, der medgår til at udføre en instruktion af den pågældende type ved et 10-bit skifte [13, s. 8-14 - 8-28]). 3-clock reads og writes er forudsat). Ialt kræver ovenstående instruktioner 31 clock-cycles. Til sammenligning skiftes 16 bit ved hjælp af instruktionerne SWAP (7) og MOVE.L (3), ialt 10 clock-cycles.

10.6.6 Konklusion

Med en max-værdi i beregningerne på $|512|$ eliminerer vi risikoen for overflow hvis $Y_m \geq 512$. Da der skal skiftes et antal bits, svarende til heltalsdel $+$ signbit, efter hver multiplikation, og da dette skifte udføres mest effektivt når heltalsdel $+$ signbit $= 16$, har vi valgt at skalere således at $Y_m = 2^{15}$.

10.7 Test

Den grundlæggende algoritme er implementeret på PC med floating point i den optimerede version. Funktionaliteten er black-box testet med en variation af sinus-signaler. Resultatet er dernæst sammenlignet med MATLAB-beregninger. På denne baggrund har vi konkluderet at funktionaliteten er korrekt.
Algoritmen er ligeledes implementeret med fixed-point på PC, både i den ikke-optimerede version og den optimerede version, og dernæst black-box testet. Der er ikke udført white-box test, eller målinger af kvantiseringsstøj.

Kapitel 11

LCD-klient

Følgende kapitel indeholder en funktions- og implementationsbeskrivelse af LCD-klienten.

11.1 Funktion

LCD-klientens primære funktion er at omsætte output'et fra FFT'en til LCD-displayet. Selve skrivningen til LCD-displayet foregår ved et kald til displaymanageren. Dette forårsager at klienten har lav kobling til hardwaren. Derfor har det under design og implementation været muligt at skrive til et emuleret LCD-display på PC-monitoren.
Lcdklienten skal efter vores ønske beregne powerspektum, amplitudespektrum og eventuelt fasespektrum, som valgfrit skal kunne vises på LCD-displayet. Powerspektret kan betegnes som et mål for effekt-indholdet i indgangssignalet; og er matematisk beskrevet som summen af kvadraterne af imaginærdelen og realdelen: $P = re^2 + im^2$. Det skal her fremhæves at signalets faseinformation går tabt i et powerspektrum. Det kan alt afhængig af, hvad man ønsker af sit output, have stor betydning, men da vores signal allerede i antialiaseringsfilteret bliver fasedrejet, tjener det intet formål at vise. Amplitudespektret beskrives matematisk som modulus af koefficienterne $|X| = \sqrt{P} = \sqrt{Re^2 + Im^2}$, og fasespektret som argumentet $\phi = arctan\frac{im}{re}$.
Vi har i første omgang valgt kun at implementere præsentation af powerspektrum og amplitudespektrum. Selve beregningen af de 2 spektre kan foretages på flere måder. Ser vi først på powerspektret, ville een måde være at beregne $im^2 + re^2$ for hver bar, der skal vises på displayet, altså 2 mulitplikationer og 1 addition, der i clockcycles er $T_{mul} \cdot 2 + T_{add} = 52 \cdot 2 + 8 = 112$. [13] En anden måde ville være at slå resultatet op i en prekalkuleret tabel;

hvor sidstnævnte kan implementeres på flere måder:

- 1. En endimensionel tabel, hvor output er kvadradtet af input. D.v.s. at der laves opslag på den absolutte værdi af både realdel og imaginærdel, hvorefter resultaterne adderes. Det giver omregnet i clockcycles $(T_{abs} + T_{read}) \cdot 2 + T_{add}) = (10 + 6) \cdot 2 + 8 = 40$ altså en besparelse på 72 clockcycles. Størrelsen af denne tabel vil med en opløsning på 8 betydende bits fylde 512b, idet hver "tal" fylder 16 bit (2bytes).

- 2. En todimensionel tabel, hvor output er summen af kvadraterne på absolutværdien af input, realdel og imaginærdel. I clockcycles giver det $(T_{abs} + T_{read}) \cdot 2) = (10 + 6) \cdot 2 = 32$, en besparrelse på 80 clockcycles. Størrelsen af denne tabel er jævnfør afsnit 4.3 begrænset til en opløsning på 8 bit (256 værdier) i hver dimension, hvilket giver det et ramforbrug på 64kB.

Samme gør sig gældende for amplitudespektret, her vil besparelsen i clockcycles blot være større, idet vi med tabelopslag sparer beregning af kvadratroden.
Uden at tage stilling til hvilken type tabel der skal implementeres, er der endnu en mente, der skal overvejes. Ønsker man at kunne se værdier mellem 0 og 1301072 (256x256+256x256) på et display med en vertikal opløsning på 64 punkter, må det nødvendigvis skaleres derefter. Lineær skalering kan anvendes, men giver en dårlig udnyttelse af præcisionen i FFT'en, hvilket ikke er hensigtsmæssigt. Derimod vil en logaritmisk skalering og dermed logaritmisk afbilning vise flere nuancer, idet lave værdier vægtes mere end høje værdier. Da m68331 ikke har en fpu-del, (der kan laves floating-point emulation i software, men det er langsom) er det ikke hensigtsmæssigt at beregne logaritmen til powerspectret. Derfor bør der også være en tabel til dette. For at implementere en logaritmetabel, der kan bruges sammen med de to ovennævnte tabeller, skal der til metode eet, laves en endimensionel logaritme-tabel, der har en størrelse på: $256^2 + 256^2 = 128kB$. Til metode to, kunne power-tabellen let udbygges så output'et er logaritmen til summen af kvadraterne på input, realdel og imaginærdel. Til amplitude-tabellen vil det tilsvarende være logaritmen til roden af summen af kvadraterne på input. Denne metode giver kun et tabelopslag, og må herfor være at foretrække.

11.2 Implementation

Lcdklienten opbygges af 3 basis funktioner, en init, en update og en deinit. Init-funktionen, lcdklient_init, bruges til at initialisere lcdklienten, det vil sige valg af tabeltype. Update-funktionen, lcdklient_update, bruges til at beregne / lave tabelopslag for et amplitude eller powerspectret til et helt skærmbillede. Deinit-funktionen, (lcdklient_deinit), har reelt set ingen funktion på nuværende tidspunkt, men er en tom funktion, der blot skal være med til at give alle programdele samme interface, samt hindre store ændringer i softwaren ved senere udvidelser.

11.2.1 Lcdklient_init

Init funktionen tager en parameter *displaymode*, der bestemmer om output'et fra FFT'en skal vises som enten amplitude- eller powerspektrum. Selve tabellen bliver beregnet i en speciel programstump *lcdklient_calculate_tables*, efter samme procedure, som er beskrevet i 4.3. Størrelsen af tabellen, der beregnes i *lcdklient_calculate_tables* fastsættes som globale konstanter i headerfilen til lcdklienten, *LCDKLIENT_TABLESIZE_RE* og *LCDKLIENT_TABLESIZE_IM*. Power-tabellen laves ved at gemme logaritmen til $(re \cdot re + im \cdot im)$ vægtet med et skaleringsfaktor, der for powerspektruet er givet ved:

$$DISPLAY\_YRES/(log10((/LCDKLIENT\_TABLESIZE\_RE - 1^2) \tag{11.1}$$

$$\cdot LCDKLIENT\_TABLESIZE\_IM - 1^2)) \tag{11.2}$$

og for amplitudespektret:

$$DISPLAY_Y RES/(log10 \cdot \sqrt{(LCDKLIENT\_TABLESIZE\_RE - 1^2)} \tag{11.3}$$

$$\cdot \sqrt{(LCDKLIENT\_TABLESIZE\_IM - 1^2))} \tag{11.4}$$

11.2.2 Lcdklient_update

Update-funktionens primære opgave er at beregne et nyt skærmbillede, til LCD-displayet. Skærmbilledet laves ved at beregne *DISPLAY_XRES* (128) tabelopslag med sekvensen af komplekse frekvenskomposanter *re* og *im* som input, og få *output* som det output, der skal bruges i kaldet af *samplemanager_writebar*. Derudover skal der tages højde for at $FFTLENGTH/2$ (vi plotter kun første halvdel af FFT'en, da anden halvdel blot er en spejling af første, se FFT teori i appendiks A) ikke altid behøves at svare til bredden på displayet *DISPLAY_XRES* (128 punkter), men godt kan være f.eks. det halve. Er dette tilfældet skal hver punkt på LCD-displayet tegnes 2 gange efter hinanden. Dette kan gøres med følgende pseudo kode:

```
int barwidth, tmp;
barwidth = DISPLAY\_XRES/(FFTLENGTH/2);
for(count2=0;count2<barwidth;count2++){
   writebar(count1*barwidth+count2,output);
}
```

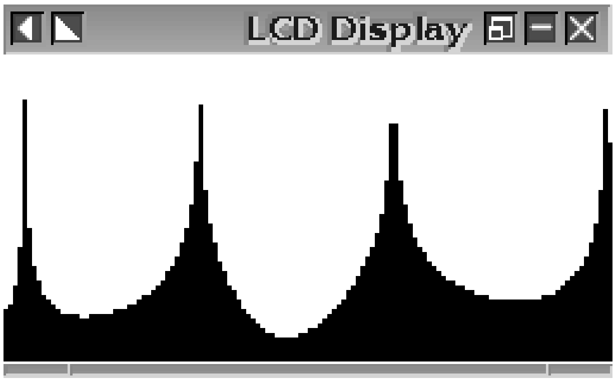
Her sættes *barwidth* til bredden af hver enkelt bar, beregnet udfra bredden af displayet delt med den halve FFTlængde. Derefter løber variablen $count2$ fra 0 til *barwidth* i en forløkke, der så længe den er true, sætter samme (output) i *writebar*.

11.2.3 Lcdklient_deinit

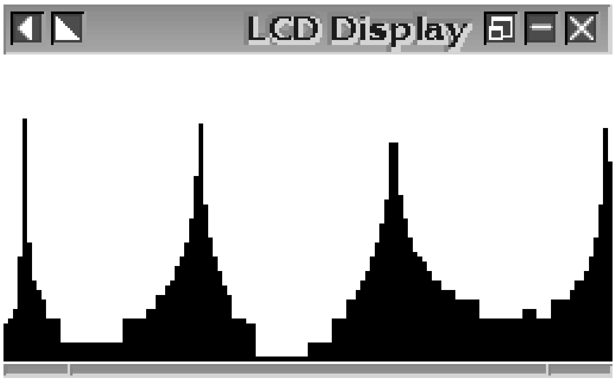
Som sagt er *lcdklient_deinit* kun en dummy funktion, og har derfor ikke nogen egentlig betydning.

11.3 Test

Den færdige lcdklient skal for at kunne testes, kompileres med FFT'en, displaymanageren og grafik bibliotekerne fra PTC. Herefter kan præcisionen af tabelbrugen testes grafisk. Der laves et grafisk output, hvor input'et er 4 sinus funktioner, mellem 0 og 2π, uden brug af tabel, samt et output af samme med brug af en 256*256 punkters opslagstabel.



Figur 11.1: Output fra fft vist på emuleret LCD-display, med direkte beregning af et logaritmisk powerspectrum



Figur 11.2: Output fra FFT vist på emuleret LCD-display, med tabelbrug til beregning af et logaritmisk powerspectrum

Det ses, at de laveste værdier bliver grove, når der benyttes tabelopslag til beregning af logaritmisk powerspectrum. Umiddelbart har vi valgt ikke at gøre noget for at finpudse dette problem, da det i første omgang beror på den fysiske størrelse af tabellen. Samtidig er der andre områder, der er vigtigere at bearbejde.

Kapitel 12

Hovedprogram

12.1 Formål

Formålet med hovedprogrammet er at initialisere systemet, og starte programmets hovedløkke, som i en uendelig løkke overfører data fra ADC'en, starter beregning af FFT, samt udlæser de beregnede data til LCD-displayet..

12.2 hovedløkke

For at udnytte DMA controlleren bedst muligt sættes hovedprogrammet til at fylde en buffer (2) allerede under initialiseringen. Egentlig fyldes bufferen en gang før den skal bruges, hvorved man undgår "ventetid" senere. I hovedprogrammets uendelige løkke venter man først på nye data fra ADC'en. Når disse er klar, begynder overførslen af disse til buffer 1, samtidig startes FFT-beregningen af de data, der er i buffer 2. Efter endt beregning opdateres LCD-displayet, med nye data, hvorefter løkken forsætter med at vente på nye data fra ADC'en. Umiddelbart efter klarsignal fra ADC'en startes kopieringen af data til buffer 2, mens der laves FFT på samples fra buffer 1. De buffere, der tager imod data fra ADC'en er, som vist i figur 16.6, delt op i blokke af 8kB, det skal der selvfølgelig tages højde for, når systemet implementeres på MCU'en.

Kapitel 13

System test

Følgende indeholder beskrivelse af testmetoder, samt en gennemgang af forventet systemtest. Kapitlet skal ses som et eksempel på, hvorledes samtlige dele af softwaren (og tildels hardwaren) kan testes.Vi har ikke haft det samlede system oppe at køre, på et tilpas tidelig tidspunkt til at foretage systemtesten. Dog er de fleste modultest gennemført, hvad angår validering af funktionalitet.

13.1 Software testmetoder

Til test af software findes en række forskellige metoder, hvoraf de to mest benyttede er White Box og Black Box tests. De to test metoder bygger på vidt forskellige principper, og er ligeså forskellige at gennemføre.

13.1.1 White Box

White Box testen bygger på et princip om at identificere og gennemløbe samtlige uafhængige stier i en programenhed, det vil sige alle tænkelige forløb gennemløbes mindst een gang. Dette forårsager selvsagt en dybdegående og tidskrævende testproces, hvorfor White Box test bliver meget kompleks at udføre på større programenheder.

13.1.2 Black Box

Black Box testen bygger på et princip om at gennemløbe en programenhed med en passende mængde udvalgte input data. Formålet med testen er primært at afsløre funktinalitetsfejl og fejl i overførsel af data mellem grænseflader. Udfordringen i Black Box tests er at udvælge passende input data. Typisk vil man vælge at teste på kritiske værdier, såsom maksimum og minimum, samt værdier udenfor det afgrænsede område.

13.2 Modultest

LCD-klient er testet med henblik på tabelbrug vs. beregning af logaritmiske værdier for amplitude og powerspektrum, se afsnit 11.3. Opstartskode er testet på MCU ved hjælp af BDM-interfacet, hvormed det, via singel-stepping gennem opstartskoden, har været muligt at verificere at kode og data blev kopieret til de rette adresser.

13.3 Testbeskrivelse

Den endelige systemtest planlægges udført, så snart det samlede system er køreklart. Testen tager udgangspunkt i Black Box metoden, og vil således blive udført ved at lade kritiske input data gennemløbe systemet. Disse data vil eksempelvis være sinussignaler med samme frekvens som yderværdierne i, det i kravsspecifikationen anførte, frekvensområde 20Hz og 20kHz, samt minimum og maksimum værdier for indgangssignalets amplitude. Sideløbende kunne samme data indsættes i MatLab, for at have et referencepunkt, at forholde sig til.

13.4 Forventet resultat

Resultatet af testen forventes at vise at systemet kan fungere i realtid, jævnfør kravene i kravsspecifikationen.

Kapitel 14

Minimumsystem

14.1 Indledning

Microkontrolleren kræver følgende fundamentale ting, for at den kan fungere:

- Spændingsforsyning
- Clock
- Resetkredsløb
- ROM
- RAM

I det følgende er beskrevet, hvorledes disse er designet og implementeret.

14.2 Spændingsforsyning

MCU, RAM, ROM, oscillator samt logik-kredse skal ifølge databladene forsynes med en spænding på 5V $\pm$ 10%. Denne spændingsforsyning fås fra en udleveret stabiliseret 5V forsyning. Når spændingsforsyningen er under 5V - 10%, resettes systemet for at undgå uforudsigelige tilstande (Se afsnit 14.4).
Ifølge Motorola [17, s. 11] skal der gøres en række ting, for at MCU'en får en støjfri spændingsforsyning. MCU'ens forsyningsspænding er delt op i to grupper:

- V_{DDE}/V_{SSE}, som driver de eksterne udgangstrin (f.eks. databus og adressebus)

- V_{DDI}/V_{SSI}, som driver MCU'ens kerne (CPU32) og andre interne enheder

De perifære udgangstrin støjer meget, mens de interne enheder er følsomme over for støj. Derfor skal spændingsforsyningen til de interne enheder forbindes, således at støj undgås. Hvis et flerlags print benyttes, anbefales det af Motorola [17, s. 15] at have et stellag og et forsyningslag, hvortil alle spændingsforsyninger forbindes direkte. Vi bruger et såkaldt "power plane board" beregnet til wrapning. Dette print har et stellag og et forsyningslag, men pga. wrapninger er en direkte forbindelse til lagene ikke mulig. Vi har derfor forsøgt at forbinde forsyningsbenene til så få punkter som muligt og har forbundet alle V_{DDI}/V_{SSI} ben til samme punkt.

14.2.1 Opstilling

Alle V_{DDI}/V_{SSI} og V_{DDE}/V_{SSE} forsyningsben skal have afkoblingskapacitorer. Dette er opfyldt, da MCU'en udleveres på et print, hvor kapacitorerne er påmonteret.
VDDSYN er et ben på MCU'en, som skal sikre støjfri spændingsforsyning til MCU'ens clock synthesizer. Da vi benytter en ekstern clock, forbindes VDDSYN direkte til VDD efter Mororolas anbefaling [16, s. 4-12].
Alle andre enheder på printet forbindes til samme stel- og forsyningslag som MCU'en og der placeres en afkoblingskondensator på 100 pF mellem komponenternes stel- og forsyningsben.

14.3 Clock

Det for MCU'en nødvendige clocksignal kan frembringes på to måder:

- Et krystalkredsløb med en frekvens på 20-50 kHz forbindes til MCU'ens interne frekvenssynthesizerkredsløb for at frembringe den høje systemclockfrekvens

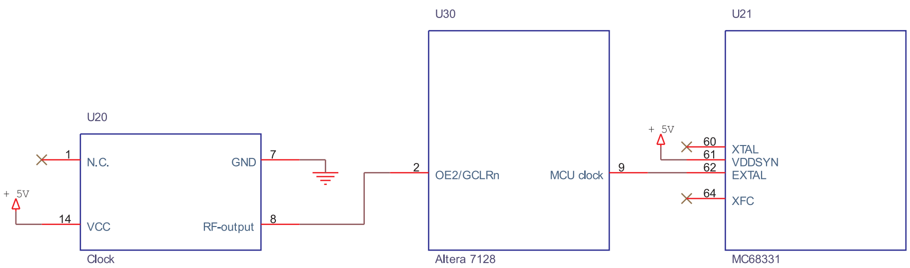
- En færdig, ekstern oscillator med systemfrekvensen forbindes direkte til MCU'en

Den første løsning stiller store krav til opbygningen af krystalkredsløbet, herunder print-layout, spændingsforsyning samt forbehold mod snavs og støv. Da vi ikke har haft undervisning om disse emner, og da et godt print-layout er svært at lave på et power plane board, har vi først valgt at bruge løsning nummer to.

Denne løsning er ikke realiserbar på en 16,78 MHZ MC68331 ved at bruge en 16,78 MHz oscillator, da der er et krav til clocksignalets duty cycle (symmetri). Dette problem løses ved at bruge en 32,000 MHz oscillator, som neddivideres i en Altera 7128 EPLD.

14.3.1 Opstilling

Oscillatoren og Alteraen placeres så tæt på MCU'en som muligt. Oscillatoren tilsluttes en forsyningsspænding på 5V (ben 14) og GND (Ben 7) forbindes til stel. "RF-output"-benet (ben 8) på oscillatoren forbindes til Alteraen's ben 2. Fra Alteraens ben 9 sendes det, af Alteraen, neddividerede clocksignal til MCU'ens EXTAL-ben (ben 62). XTAL (ben 60) og XFC (ben 64) på MCU'en skal ikke forbindes, mens V_{DDSYN} (ben 61) skal forbindes til 5V [17, s. 12].



Figur 14.1: Figur af clock forbundet til MCU'en

Under reset skal MODCLK holdes lav på MCU'en (ben 78) for at signalere til MCU'en, at den skal bruge en ekstern clock på EXTAL-benet (Se figur 14.2 og afsnit 14.4).

14.3.2 Timing

Ved en 16,78 MHz MC68331 er der et minimumskrav til clocksignalets periode og høj/lav tid.
Perioden skal mindst være 1/16,78 MHz = 59,6 ns, hvilket kan opfyldes ved at vælge en oscillator på 16,780 MHz. En sådan oscillator vil dog normalt ikke overholde høj/lav tiden på 29,8 ns [16, s. A-9, t_{XCHL}], da den vil have en duty-cycle, der afviger fra de perfekte 50%. Duty-cyclen på en typisk oscillator er på henholdsvis 45% og 55%. Vi kan herudfra beregne den højeste clockfrekvens, vi måtte forsyne MCU'en med, hvis vi brugte en typisk oscillator:

$$Maks.\ clockfrekvens = \frac{Min.\ ekstern\ duty\ cycle}{Min.\ hoej/lav\ tid} = \frac{45\%}{29,8ns} = 15,1MHz \qquad (14.1)$$

Da vi vil have mest mulig processorkraft til rådighed til beregning af FFT'en, er denne løsning ikke tilfredsstillende. Vi har derfor valgt at benytte en oscillator med den dobbelte frekvens, som neddivideres i en Altera 7128. Herved undgåes problemet med variabel dutycycle, da clockperioden er fast og Alteraen trigger på den opadgående flanke. Vi har valgt en oscillator på 32.000 MHz, da denne frekvens ligger tættest på den dobbelte maksimumfrekvens, 2*16.78=33.56 MHz. Med denne løsning forøges frekvensen med 0,9 MHz i forhold til, hvis vi havde brugt en typisk 15,1 MHz oscillator.
Fra MC68331 stilles krav om at den eksterne clocks stige- og faldetid skal være på hver max 5 ns [16, s. A-9]. Dette overholdes af Alterakredsen, som har en stige/faldetid på 4-5 ns [27, s. 18].

14.3.3 Elektrisk dimensionering

MCU'en kræver CMOS-niveauer på alle inputs [16, s. 3-6], altså under $20\% \cdot V_{DD} = 1V$ for lav og over $70\% \cdot V_{DD} = 3,5V$ for høj. Alteraen følger disse CMOS-niveauer (AN74 s.3).

14.4 Resetkredsløb

Resetkredsløbet er en nødvendig del af systemet, som skal sørge for, at de kritiske kredse bliver startet rigtigt, når der tændes eller resettes for systemet. Endvidere skal det resette systemet, hvis spændingsforsyningen bliver ustabil.
Kredsløbet består af to dele:

- Konfigurationskredsløbet, som sørger for, at bestemte ben på MCU'en bliver sat til ønskede logiske niveauer under reset.
- Low Voltage Inhibit-kredsløbet, som bl.a. sørger for, at MCU'en og andre kritiske kredse kun arbejder, når de får tilført en tilstrækkelig spændingsforsyning.

I det følgende er beskrevet, hvorledes konfigurationskredsløbet og LVI-kredsløb-et er designet og implementeret.

14.4.1 Konfigurationskredsløb

Pga. MC68331'erens mange muligheder og brede anvendelsesområde er MCU'en udstyret med en række registre, som konfigurerer MCU'en til forskellige opgaver. Nogle af disse registre konfigurerer hardwarespecifikke ting, som er vigtige for, at MCU'en kan køre. Indholdet af disse registre skal derfor kunne bestemmes inden MCU'en startes. MCU'en har derfor en opstartsfase under reset, hvor bl.a. databenenes logiske niveauer bruges til konfiguration af registerne. Figur 14.2 viser, hvilke funktioner MCU'en får ved at konfigurere de angivede ben under opstart. Endvidere er $\overline{BERR}$ og $\overline{HALT}$ blevet sat til logisk 1 og TSC er blevet sat til inaktiv tilstand ved at indsætte en 10 kΩ's modstand mellem benene og forsyningsspændingen på 5V efter Motorolas anvisninger [17, s. 5].

Mode Select Pin	Default Function (Pin Left High)	Alternate Function (Pin Pulled Low)	Used Settings
DATA0	$\overline{CSBOOT}$ is 16-bit port	$\overline{CSBOOT}$ is 8-bit port	High
DATA1	$\overline{CS0}$ $\overline{CS1}$ $\overline{CS2}$	$\overline{BR}$ $\overline{BG}$ $\overline{BGACK}$	Low
DATA2	$\overline{CS3}$ $\overline{CS4}$ $\overline{CS5}$	FC0 FC1 FC2	High
DATA3 DATA4 DATA5 DATA6 DATA7	$\overline{CS6}$ $\overline{CS[7:6]}$ $\overline{CS[8:6]}$ $\overline{CS[9:6]}$ $\overline{CS[10:6]}$	ADDR19 ADDR[20:19] ADDR[21:19] ADDR[22:19] ADDR[23:19]	???
DATA8	$\overline{DSACK0}$,$\overline{DSACK1}$,$\overline{AVEC}$,$\overline{DS}$,$\overline{AS}$,$\overline{SIZE}$	PORTE I/O pins	Low
DATA9	$\overline{IRQ[7:1]}$, MODCLK	PORTF I/O pins	Low
DATA11	Slave Mode Disabled	Slave Mode Enabled	High
MODCLK	VCO = System Clock	EXTAL = System Clock	Low
$\overline{BKPT}$	Background Mode Disabled	Background Mode Enabled	Low/High

Figur 14.2: Tabel over opsætningspins betydning under reset

Når MCU'en ikke er under reset, skal databussen være frigivet til andre enheder på bussen. Denne funktionalitet kan realiseres på forskellige måder:

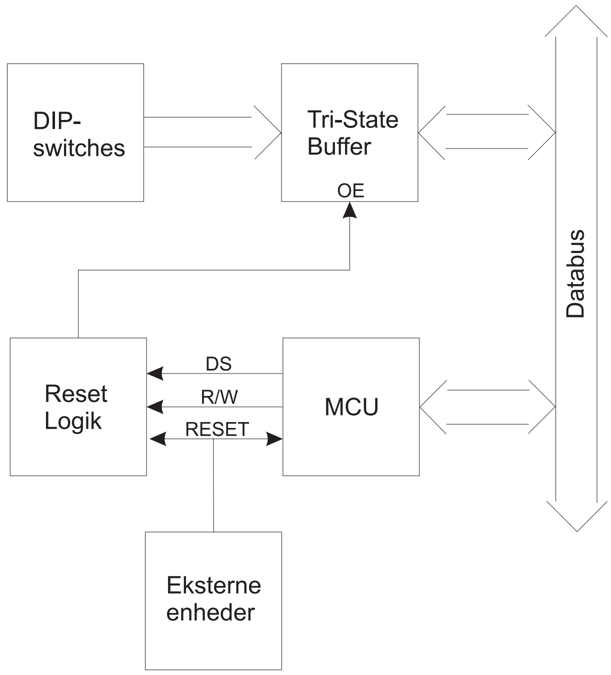
- Simple pull-up/pull-down modstande på databenene
- Databenene drives af et kredsløb med tristate buffere, hvorpå udgangene aktiveres under reset

Den første løsning kan bruges, men belaster bussen. Dette kan kun anbefales i små og billige systemer, hvor der kun er få enheder på bussen. Den anden løsning belaster bussen langt mindre, men kræver lidt ekstra HW i form af Tri-State Buffere og styrelogik, da bufferen ikke må benytte bussen samtidig med andre enheder.

Da vi har flere enheder, som skal benytte bussen, vælger vi den sidste løsning, da denne giver et mere stabilt system. Princippet i denne løsning beskrives i det følgende afsnit.
En konfigurationsbitmaske sættes op til de værdier, som MCU'en skal sættes op med under opstart. Konfigurationsbitmasken forbindes til en Tri-State Buffer, som skal sende den ud på databussen, når MCU'en skal sættes op. Under opstart og reset får Tri-State Bufferen således et signal fra reset logikken, når databussen er ledig. Dette sker, når følgende krav er opfyldt:

- R/$\overline{W}$ er logisk høj
- $\overline{DS}$ er logisk høj
- $\overline{RESET}$ er logisk lav

R/$\overline{W}$ og $\overline{DS}$ er med i reset logikken, da de angiver, om bussen er ledig, og dermed om Tri-State Bufferen må tilgå bussen. Resetsignalet kan enten asserteres af MCU'en eller en ekstern enhed (f.eks. LVI'en, se afsnit 14.4.2). Principdiagrammet for konfigurationskredsløbet kan ses på figur 14.3



Figur 14.3: Princippet i konfigurationskredsløbet

Konfigurationskredsløbet kan enten implementeres vha. alm. logikkredse (to buffered line driver, en NAND-gate og en inverter) eller i en Altera 7128 EPLD.

Opstilling

I vores opstilling bruger vi to buffered line drivere af typen 74HC244, som er en otte udgangs ikke inverterende buffer/line drivere. Endvidere benytter vi en NAND-gate af typen 74HC10, som også bruges som inverter. Indstilling af de logiske niveauer, som af line driveren sendes til MCU'en under reset, foretages vha. DIP-switches. Konfigurations-kredsløbets opstilling kan ses på figur F.3.

Timing

For at MC68331 kan genkende de logiske niveauer på databussen under opstart, er der fra Motorola stillet krav om, at niveauerne på konfigurationspins'ene skal holdes i mindst 5 ns efter at systemet har "sluppet" reset signalet. Dette overholdes idet 74HC244'ern først begynder at "floate" efter 14 ns [31, s. 472].
Et andet krav er, at den samlede holdetid for den bufferede line driver, NAND-gaten og inverteren ikke må blive større end 2 clockcycles (125 ns ved 16,000 MHz), da andre enheder kan begynde med at benytte databussen efter 2 clockcycles. Dette overholdes, da den samlede holdetid er på 14 ns + 9 ns + 9 ns = 32 ns [30, s. 125].

Elektrisk dimensionering

Da MCU'en arbejder med CMOS-niveauer, har vi valgt buffere fra 74HCxxx-serien, som er CMOS-kompatibel. Bufferens outputs kan maksimalt belastes med 35mA, hvilket er nok til at drive alle bussens enheder.

14.4.2 Low Voltage Inhibit-kredsløb

For at undgå, at MCU'en og andre enheder kommer i udefinerbare tilstande under opstart og drift, anvendes en LVI (Low Voltage Inhibit), som sørger for at resette systemet, hvis spændingsforsyningen er under systemets minimale tilladelige spændingsniveau.
Vi har valgt at implementere denne funktionalitet ved at anvende en 7705-ACP fra Texas Instruments. Denne LVI har en ekstra funktionalitet, idet den har en $\overline{RESIN}$-indgang, hvor man eksternt kan trigge reset. Vi har valgt at udnytte denne funktionalitet ved at sætte en resetknap til LVI'ens $\overline{RESIN}$-indgang.

Opstilling

LVI'en forbindes som beskrevet i det følgende.
LVI'ens $\overline{RESET}$-udgang sættes på MCU'ens $\overline{RESET}$, således at den kan assertere et resetsignal i reset logikken. Endvidere sættes en kondensator C_T på 2,2 uF på 7705'erns CT-indgang (Se afsnit 14.4.2 Timing). Efter Texas Instruments anbefaling placeres en 0,1 uF kondensator mellem LVI'ens REF-ben og stel.
Resetknappen forbindes til LVI'ens $\overline{RESIN}$-indgang. Endvidere sættes en kondensator på 22 uF over resetknappen for at undgå støj.
Den fysiske opstilling kan ses på figur F.3.

Elektrisk dimensionering

LVI'en negerer $\overline{RESET}$, når spændingsforsyningen når over 4,5 V. Denne forsyningsspænding passer med alle de andre enheder (MCU, Altera, RAM, ROM, logik og Display), da disse stiller krav om en spændingsforsyning på 5 V $\pm$ 10%.
LVI'ens $\overline{RESET}$-udgang arbejder med CMOS-niveauer, og er dermed kompatibel med MCU'en og de andre enheder, som er sat til $\overline{RESET}$-signalet.
På systemets $\overline{RESET}$-linie bruges en 820 Ω's pull up modstand efter Motorolas anbefaling. Dette er en meget lav pull-op modstand og vi skal derfor sikre os, at LVI'en kan tåle strømmen fra den. Denne giver en strøm på $\frac{5V}{820\Omega} = 6mA$, når LVI'en forsøger at trække $\overline{RESET}$ lav. Dette er lavere end LVI'ens maksimumstrøm på 30 mA [32, s. 2-345].

Timing

Hvis LVI'en resetter systemet pga. for lavt spændingsniveau eller ekstern anmodning fra resetknappen, skal $\overline{RESET}$ holdes længe nok til at de andre enheder registrerer, at systemet skal resettes. Ifølge LVI'ens datablad (reference) har resetsignalets holdetid t_d følgende sammenhæng med den eksterne kondensator C_T på LVI'ens CT-ben:

$$C_T = \frac{t_d}{1.3 \cdot 10^4} \tag{14.2}$$

Motorola 68331 kræver, at resetsignalet holdes i mindst 590 CLKOUT-cycles [16, s. A-13], hvilket svarer til ca. 35 us. Displayet kræver en holdetid på mindst 1 us [28, s. 8] og Altera 7128'ern kræver en holdetid på mindst 6,2 ns [27, s. 34]. Der stilles endvidere krav til spændingsforsyningens opstartsfase, da den her vil have fluktioner i spændingen. En holdetid t_d på 10 til 20 ms vil normalt afværge dette problem [29, s. 7-5]. På baggrund af ovenstående overvejelser vælger vi holdetid en periode t_d på ca.28 ms, svarende til en kondensator C_T på 2,2 uF.

14.5 ROM

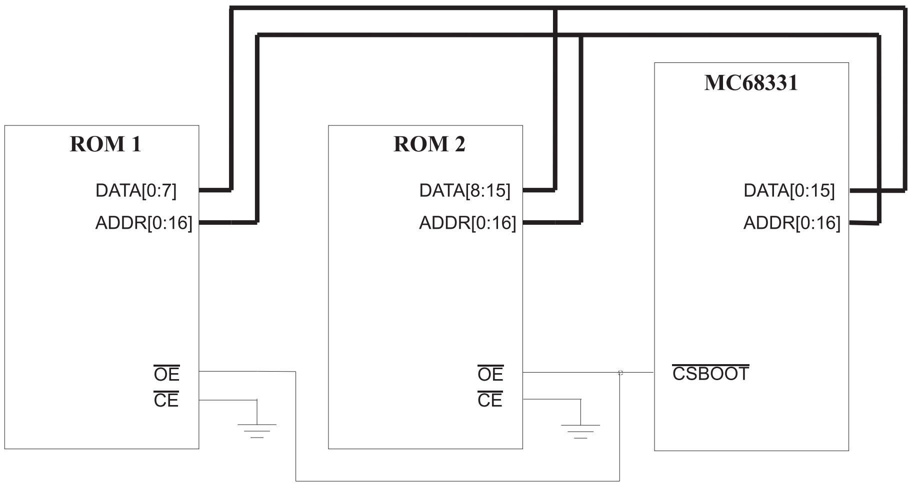
68331'eren giver mulighed for både 8-bit og 16-bit databredde til ROM. Vi har besluttet at benytte 16-bit databredde.
ROM kredsene fås i både 8- og 16-bit bredde, men den sidstnævnte type er langt dyrere end den første. Vi har derfor valgt to 8-bit ROM-kredse af typen NS27C010Q-15 fra National Semiconductor. Hastigheden er 150 ns og størrelsen er på 128kB i hver kreds. Den ene kreds bruges til ulige adresser, mens den anden bruges til lige adresser.

14.5.1 Opstilling

Kredsene forbindes til MCU'en ved at forbinde MCU'ens adresseben [17:1] til kredsenes adresseben [16:0], hvilket gør kredsene adresserebare i words. Når MCU'en skal læse bytes læser den på samme måde som ved word læsning, men bruger kun informationen fra den ene kreds. Den ene halvdel af MCU'ens databen [7:0] forbindes til den ene kreds' databen [7:0], mens den anden halvdel [15:8] forbindes til den anden kreds' databen [7:0]. Kredsenes $\overline{CE}$ ben forbindes til stel og $\overline{OE}$ benene bruges som chipselect. Begrundelsen for netop denne opstilling og ikke den omvendte er, at det kun tager 40 ns før data bliver gyldige på databenene, når kredsene vælges med $\overline{OE}$ benene, mens det tager 150 ns, hvis kredsene vælges med $\overline{CS}$ benene. Denne løsning øger dog effektforbruget, hvilket er iorden, da vi ikke har prioriteret optimering af effektforbruget. $\overline{CS}$-benene forbindes til $\overline{CSBOOT}$-benet på MCU'en. Dette chip-select ben bruges, da det er dette ben MCU'en chip-selecter efter reset. Der vælges 16-bit databredde ved at konfigurere dip-switch 0 (DATA0) til OFF (Pin Left High).
Opstillingen for ROM kan ses på figur 14.4.

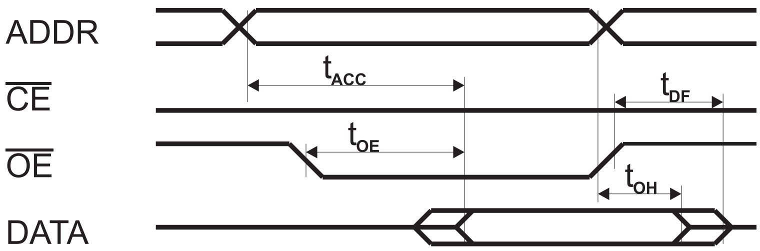
14.5.2 ROM timing

I det følgende er timingen i forbindelse med læsning af ROM beskrevet. Dette er gjort ud fra ROM'ens [3, s. 3-97 - 3-98] og MCU'ens datablad [16, s. A-9 - A-10 og A-15]



Figur 14.4: Figur af ROM forbundet til MC68331

Læsecyklus'en starter ved, at adressen sættes på adressebussen. Normalt sættes $\overline{CE}$ (chip enable) lav. Vi har dog valgt konstant at holde denne lav, da det giver en tilgangstid til ROM'en på 40 ns mod ellers 150 ns, hvilket forøger effektforbruget. Data kommer på databussen max. 150 ns efter at adressen er sat på adressebussen, såfremt $\overline{OE}$ (output enable) er sat lav max. 40 ns forinden. Kravet fra MCU'en er her 89,2 ns. På figur 14.5 ses diagrammet for en ROM-læsecyklus.



Figur 14.5: Timingsdiagram for læsning af ROM.

Timingen af ROM'en er designet således, at denne kan tilgås hurtigst muligt. Dette gøres ved at anvende mindst mulig antal waitstates. Timingstiderne ved 0 waitstates kan ses i figur 14.6 (* angiver hvilken enhed, som stiller kravene i det pågældende tilfælde).
Det ses af figur 14.6, at t_{ACC} ikke overholder kravene ved 0 waitstates. Der indsættes derfor 1 waitstate. Herved opfyldes kravene til at $t_{ACCMCU,max}$ skal være større end $t_{ACCROM,max}$. Timingstiderne ved anvendelse af 1 waitstate kan ses på figur 14.7, hvor alle timingskravene er overholdt.

Symbol	ROM		MCU	
	Min.	Max.	Min.	Max.
t_{ACC}		150 ns		115 ns*
$t_{\overline{OE}}$		40 ns		89,2 ns*
$t_{\overline{OH}}$	0 ns		0 ns*	
$t_{\overline{DF}}$		35 ns		55 ns*

Figur 14.6: Timingstider for læsning af ROM med 0 waitstates

Symbol	ROM		MCU	
	Min.	Max.	Min.	Max.
t_{ACC}		150 ns		175 ns*
$t_{\overline{OE}}$		40 ns		148,8 ns*
$t_{\overline{OH}}$	0 ns		0 ns*	
$t_{\overline{DF}}$		35 ns		55 ns*

Figur 14.7: Timingstider for læsning af ROM med 1 waitstate

14.5.3 SIM-registre

Efter reset starter MCU'en med at tilgå ROM'en med 13 wait-states. Ved en adgangstid på 150 ns kan vi tilgå ROM'en med kun 2 wait-states [17, s. 19].
SIM-registret CSORBT ($FFFA4A) er knyttet til benet $\overline{CSBOOT}$ og definerer følgende: Synkron/asynkron busoperation, øvre/nedre byte-adgang, læse/ skrive adgang, address strobe/data strobe synkronisering af chipselect, antallet af waitstates, CPU/user/supervisor adgang, interrupt prioritets niveau, autovector on/off.
Registret konfigureres til følgende:

- Asynkron operation. Dette giver mulighed for at tilføje waitstates
- Øvre og nedre byte adgang. Dette gør at ROM-kredsene vælges både når der læses i byte og word-bredde
- Læse adgang. Det skal ikke være muligt for MCU'en at forsøge at skrive til ROM
- Address strobe synkronisering af chip-select. Under læsning kommer $\overline{DS}$ signalet forskudt i forhold til $\overline{AS}$ signalet med mellem -15 og 15 ms (afhængig af belastningen på benene). Derfor vælges Adress strobe synkronisering, da det giver mere

præcist valg af chipselect (ellers har det ikke nogen betydning om vi vælger AS eller DS).

- 2 waitstates. Dette felt virker kun i asynkron busoperation. Der kan vælges et eksternt data-acknowledge signal. Vi har valgt en intern kilde til dette signal, som venter med 2 waitstates

- Supervisor og user adgang. MCU'en starter i supervisor mode. Det er ikke udelukket at vi vil bruge user mode senere, derfor vælges denne også. CPU space er en special type adgang, som ikke bruges under normal læse/skrive adgang

- Interrupt prioritets niveau = alle. Dette felt bruges når et chipselect ben bruges som interrupt acknowledge. Når dette ikke er tilfældet, sættes feltet til "all" = %000

- Autorvector off. Dette felt har kun betydning i forbindelse med interrupts. Er feltet sat til "off" bruges en ekstern interrupt vektor, ellers bruges en autovektor

Disse indstillinger giver CSORBT-værdien %0110100010110000 = $68B0 (se figur 14.8).

CSORBT - Chip Select Option Register Boot ROM $FFFA44

MODE	BYTE		R/$\overline{W}$		STRB	$\overline{DSAK}$				SPACE		IPL			$\overline{AVEC}$
0	1	1	0	1	0	0	0	1	0	1	1	0	0	0	0

Figur 14.8: Figur af instillinger i registeret CSORBT

SIM-registret CSBARBT ($FFFA48) er også knyttet til $\overline{CSBOOT}$ og bestemmer fra hvilken adresse, der skal chip-selectes samt, hvor stor chip-select blokken er. Der kan vælges blokstørrelser på 2, 8, 16, 64, 128, 256, 512 og 1024 KB. Blokstørrelsen er bestemt af de tre mindst betydende bits i registret. Bits 3 til 15 bestemmer blokkens startadresse-bits 11-23. Vi vælger 256 KB blokstørrelse til ROM'en. Blokken skal starte ved adresse 0. Disse indstillinger giver CSBARBT-værdien %0000000000000101 = $0005 (se figur 14.9).

14.6 RAM

RAM'en laves ligesom ROM'en med to kredse med en databredde på 8 bit. Dette har et bestemt formål ved RAM'en, idet det giver mulighed for at skrive til den øvre byte, den

CSBARBT - Chip Select Base Address Register Boot ROM $FFFA48

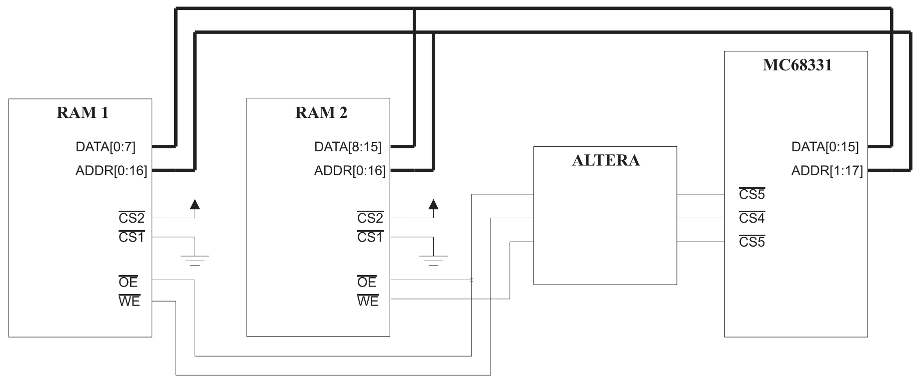
ADDR 23	ADDR 22	ADDR 21	ADDR 20	ADDR 19	ADDR 18	ADDR 17	ADDR 16	ADDR 15	ADDR 14	ADDR 13	ADDR 12	ADDR 11	BLKSZ		
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1

Figur 14.9: Figur af instillinger i registeret CSBARBT

nedre byte eller begge bytes. Her kan vi ikke nøjes med een chipselect, men vælger en løsning med tre chipselects. Ved skrivning bruges der to chipselects (en til hver kreds) og under læsning bruges en tredie chipselect som vælger begge kredse. Hvis MCU'en under læsning kun skal læse fra den ene kreds, bruger den blot de data på databussen som er nødvendige, selv om begge kredse chipselectes.
Vi har valgt at bruge to 128 KB RAM kredse af typen SRM20100LC70 fra SEIKO. Denne RAM er af den statiske type, som ikke skal genopfriskes. RAM'en har en adgangstid på 70 ns. Denne adgangstid svarer til en konfiguration med 0 wait states [17, s. 19, Tabel 3]. MCU'en kan arbejde i en konfiguration kaldt "fast termination mode" som kræver RAM med en adgangstid på mindre end 29.6 ns. Denne type RAM er dyr i forhold til den valgte type.
RAM kredsene fra SEIKO er udstyret med seperate Output Enable ($\overline{OE}$) og Write Enable ($\overline{WE}$) indgange. Vi har derfor valgt ikke at bruge MCU'ens R/$\overline{W}$ udgang, da dette ville kræve at et inverteret R/$\overline{W}$-signal skulle bruges samtidig med en ikke inverteret form. Vi har dermed udnyttet den interne logik i MCU'en, sparet en inverter og forhindret en ekstra forsinkelse af signalet. Motorola anbefaler til denne type RAM-kredse en løsning med tre chip-selects, som beskrevet ovenover [17, s. 22].
RAM-kredsene har udover $\overline{OE}$ og $\overline{WE}$ også to chipselect indgange $\overline{CS1}$ og CS2, som gør det muligt at konfigurere skrivning til RAM på tre forskellige måder. Ved læsning gælder at $\overline{CS1}$ og $\overline{OE}$ skal være lave mens CS2 og $\overline{WE}$ skal være høje. Ved skrivning skal $\overline{CS1}$ og $\overline{WE}$ være lave mens CS2 skal være høj. De tre forskellige måder at skrive på består i at man bruger eet af de sidstnævnte ben til at styre skrivningen med, mens man holder de to andre ben på de nævnte niveauer. Hvis der styres med $\overline{WE}$ benet, skal man ifølge databladet huske at sætte $\overline{OE}$ til høj. I vores tilfælde er det lettest altid at holde $\overline{CS1}$ lav og CS2 høj, både under skrivning og læsning. Vi vælger dermed at styre læsening med $\overline{OE}$-benet og at styre skrivning med $\overline{WE}$-benet.

14.6.1 Opstilling

RAM'en implementeres ved at forbinde $\overline{CS1}$ til 5V og CS2 til stel (pull up/down modstande er her ikke nødvendige, da disse ben kun fungerer som input ben). Vi bruger $\overline{CS3}$, $\overline{CS4}$ og $\overline{CS5}$ på MCU'en til valg af RAM-adgang på følgende måde:
$\overline{CS3}$ forbindes til $\overline{WE}$ på den RAM-kreds, der svarer til den øvre byte. Denne kreds' databen forbindes til data[15:8] på MCU'en.
$\overline{CS4}$ forbindes til $\overline{WE}$ på den RAM-kreds, der svarer til den nedre byte. Denne kreds' databen forbindes til data[7:0] på MCU'en.
$\overline{CS5}$ forbindes til $\overline{OE}$ på begge ramkredse.
Vi har valgt ikke at benytte Chipselects 0 til 2 på MCU'en til RAM, da disse ben kan konfigureres til bus-overtagelses funktioner, som senere skal bruges af vores DMA-kredsløb. Opstillingen for RAM kan ses på figur 14.10.

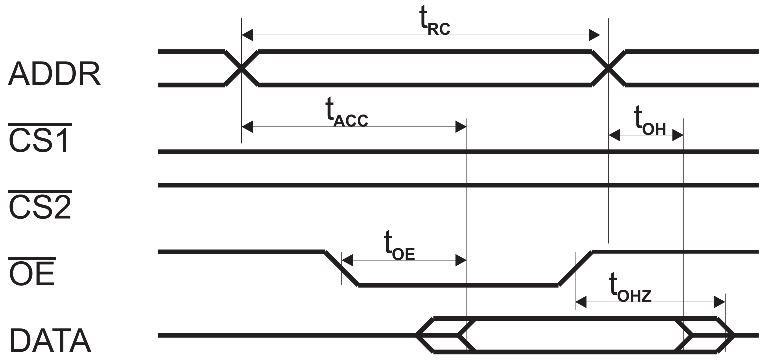


Figur 14.10: Figur af RAM forbundet til MC68331

14.6.2 RAM timing

I det følgende er timingen i forbindelse med læsning og skrivning af/til RAM beskrevet. Dette er gjort ud fra RAM'ens [25, s. 0-72 - 0-73] og MCU'ens datablad [16, s. A-9 - A-10 og A-15].
RAM'en kan tilgåes på flere måder. Vi har valgt en, hvor $\overline{CS1}$ og $\overline{CS1}$ fastholdes henholdsvis lav og høj. Vi har valgt at benytte denne konfiguration, da denne er den hurtigste (5 ns hurtigere end de andre), og da den er den mest simple at forbinde. RAM'en kan tilgåes hurtigere, hvis man anvender Fast Termination RAM (svarende til -1 waitstate), men desværre var det ikke muligt, at få denne RAM-type udleveret.

Læsecyklus'en starter ved at adressen sættes på adressebussen. Denne skal holdes i min. 70 ns. Der må maks. gå 115 ns fra addressen er sat på adressebussen til dataene er klar på databussen. Disse data holdes i mindst 10 ns efter at adressebussen er fri. Endvidere går der min. 0 ns fra $\overline{OE}$ bliver sat til dataene er klar på databussen. Disse bliver desuden holdt i max. 55 ns fra $\overline{OE}$ bliver negeret.
På figur 14.13 ses diagrammet for en RAM-læsecyklus.



Figur 14.11: Timingsdiagram for læsning af RAM.

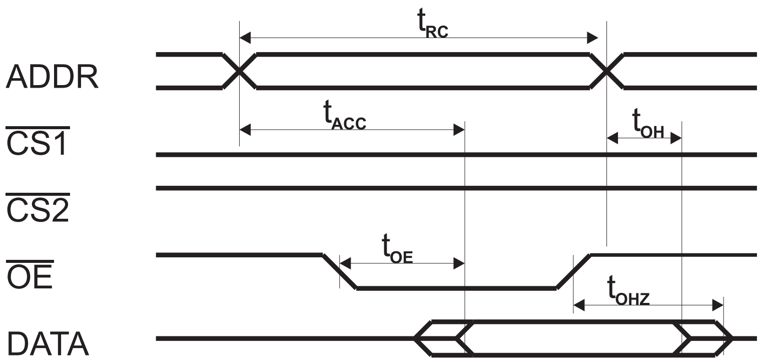
Timingen af RAM'en er designet således, at denne kan tilgås hurtigst muligt. Dette gøres ved at anvende mindst mulig antal waitstates. Timingstiderne ved 0 waitstates kan ses i figur 14.12 (⋆ angiver hvilken enhed, som stiller kravene i det pågældende tilfælde).

Symbol	RAM		MCU	
	Min.	Max.	Min.	Max.
$t_{\overline{OE}}$		40 ns		89,2 ns⋆
t_{ACC}		70 ns		115 ns⋆
t_{RC}	70 ns⋆		178,8 ns	
t_{OH}	10 ns		0 ns⋆	
t_{OHZ}		30 ns		55 ns⋆

Figur 14.12: Timingstider for læsning af RAM med 0 waitstates.

Det ses af figur 14.12, at alle kravene er overholdt, og der behøves derfor ikke yderligere waitstates.
Skrivecyklus'en starter ved at adressen sættes på adressebussen. Denne skal holdes i min. 70 ns. Der må maks. gå 115 ns fra addressen er sat på adressebussen til dataene er klar på databussen. Disse data holdes i mindst 10 ns efter at adressebussen er fri. Endvidere

går der min. 0 ns fra $\overline{OE}$ blier sat til dataene er klar på databussen. Disse bliver desuden holdet i max. 55 ns fra $\overline{OE}$ bliver negeret.
På figur 14.13 ses diagrammet for en RAM-skrivecyklus.



Figur 14.13: Timingsdiagram for læsning af RAM.

Timingen af RAM'en designet således, at denne kan tilgås hurtigst muligt. Dette gøres ved at anvende mindst mulig antal waitstates. Timingstiderne ved 0 waitstates kan ses i figur 14.14.

Symbol	RAM		MCU	
	Min.	Max.	Min.	Max.
t_{AW}	60 ns*		115 ns	
t_{WP}	55 ns*		100 ns	
t_{WC}	70 ns*		178,8 ns	
t_{DW}	30 ns*		60 ns	
t_{DH}	0 ns*		15 ns	
t_{WR}	0 ns*		15 ns	
t_{AS}	0 ns*		15 ns	

Figur 14.14: Timingstider for skrivning til RAM med 0 waitstates

Det ses af figur 14.14, at alle kravene er overholdt, og der behøves derfor ikke yderligere waitstates.

14.6.3 SIM-registre

Sim registrene til chipselect 3 til 5 fungerer på samme måde som registrene til rom'ens chip select.
SIM-registretne CSBAR3 ($FFFA58), CSBAR4 ($FFFA5C) og CSBAR5 ($FFFA60) bestemmer baseadressen og blokstørrelsen for chipselects 3, 4 og 5. Vi lader RAM'en starte ved $100000 og blokstørrelsen sættes til 256 kB (Se Figur 6.1). Disse indstillinger er ens for de tre chipselects og der skrives derfor en værdi på $1005 svarende til indstillingerne til registrene CSBAR3, CSBAR4 og CSBAR5.
CSOR3 ($FFFA5A), CSOR4 ($FFFA5E) og CSOR5 ($FFFA62) for chipselects 3,4 og 5 svarer til registret CSORBT for ROM'ens chipselect. CSOR5 konfigureres til 16-bit læsning med samme indstillinger som i CSORBT med den forskel at antallet af waitstates sættes til 0 (passende til en 150ns RAM-kreds [17, s. 19, Tabel 3]). Vi skriver en værdi på %0110100000110000 = $6830 til registret CSOR5 (Se figur 14.15).

CSOR5 - Chip Select Option Register 5 $FFFA62

MODE	BYTE		$R/\overline{W}$		STRB	$\overline{DSAK}$				SPACE		IPL			$\overline{AVEC}$
0	1	1	0	1	0	0	0	0	0	1	1	0	0	0	0

Figur 14.15: Figur af instillinger i registeret CSOR5

CSOR3 og CSOR4 skal konfigureres til skrivning til hhv. øvre og nedre byte. Disse registre får tildelt samme bitmønster bortset fra upper/lower byte feltet. Registrene konfigureres til følgende:

- Asynkron operation. Dette giver mulighed for at bestemme waitstates
- Øvre byte adgang for CSOR3 og nedre byte adgang for CSOR4. Denne konfiguration gør, at begge RAM-kredse vælges, når der skrives 16-bit, og at der vælges een kreds, når der skrives 8-bit.
- Skrive adgang. Disse chipselects skal kun vælges, når der skal skrives.
- Address stobe synkronisering af chip-select. Ved skrivning kommer $\overline{DS}$ signalet 55 ms senere end $\overline{AS}$ signalet (331UM/AD, s. A-9, t_{SWA}-t_{SWAW}). Ifølge databladet for RAM-kredsene skal skrivepulsbredden være mindst t_{WP} = 55 ns. Data strobe

synkronisering giver en bredde på t_{SWAW} = 45 ms, hvilket er for lidt. Derfor vælges Adress strobe synkronisering, som giver en skrive puls bredde på t_{SWA} = 100 ms.

- 0 waitstates. Dette felt virker kun i asynkron busoperation. Der kan vælges et eksternt data-acknowledge signal. Vi har valgt en intern kilde til dette signal, som venter med 0 waitstates
- Supervisor og user adgang. MCU'en starter i supervisor mode. Det er ikke udelukket, at vi vil bruge user mode senere, derfor vælges denne også. CPU space er en special type adgang, som ikke bruges under normal læse/skrive adgang
- Interrupt prioritets niveau = alle. Dette felt bruges, når et chipselect ben bruges som interrupt acknowledge. Når dette ikke er tilfældet, sættes feltet til "all" = %000
- Autorvector off. Dette felt har kun betydning i forbindelse med interrupts. Er feltet sat til "off" bruges en ekstern interrupt vektor, ellers bruges en autovektor

CSOR3 får værdien %0101000000110000 = $5030 (Se figur 14.16).

CSOR3 - Chip Select Option Register 3 $FFFA5A

MODE	BYTE		R/$\overline{W}$		STRB	$\overline{DSAK}$				SPACE		IPL			$\overline{AVEC}$
0	1	0	1	0	0	0	0	0	0	1	1	0	0	0	0

Figur 14.16: Figur af instillinger i registeret CSOR3

CSOR4 får værdien %0011000000110000 = $3030 (Se figur 14.17).

CSOR4 - Chip Select Option Register 4 $FFFA5E

MODE	BYTE		R/$\overline{W}$		STRB	$\overline{DSAK}$				SPACE		IPL			$\overline{AVEC}$
0	0	1	1	0	0	0	0	0	0	1	1	0	0	0	0

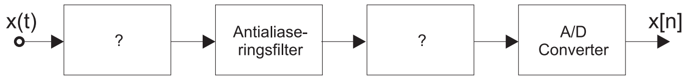
Figur 14.17: Figur af instillinger i registeret CSOR4

Kapitel 15

Dimensionering af analoge dele af dataopsamlingen

15.1 Indledning

Udgangspunktet er vores inputsignal. Dette kontinuerte signal skal konverteres til digital (sampling) , dvs. tids- og amplitudediskret form. Sampling forårsager et fænomen, der er kendt som aliasering, hvor frekvenser spejles om den halve samplefrekvens, og således ødelægger signalet. Disse kritiske frekvenser (over den halve samplefrekvens) skal bortfiltreres med et antialiaseringsfilter, et lavpasfilter med stopbåndsfrekvensen lig den halve samplefrekvens. Inputsignalet er på 2V peak to peak. Dette signal passer ikke umiddelbart



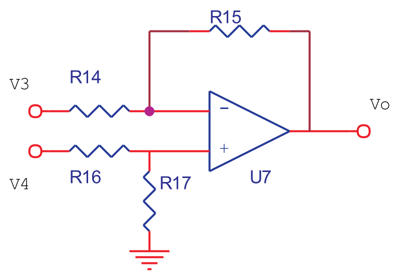
Figur 15.1: Signalets tidslige vej fra kontinuert analogt til diskret digitalt.

på de valgte komponenter (antialiaseringsfilter og analog/digital converter) og skal herfor tilpasses. Figur 15.1 viser den tidslige sammenhæng i et blokdiagram over dette.
F.eks. varierer kravene til størrelsen af peak to peak-signalet, DC-mæssig hævning af dette og impedanser.
Dette kapitel omhandler dels de valgte komponenters karakteristika, standardkoblingen af disse, og føromtalte tilpasninger.
Et aktivt-RC antialiaseringsfilter er designet i appendiks B.

15.2 Differentiel indgang

15.2.1 Differentialforstærker

For at bortfiltrere støj fra omgivelserne anvendes et såkaldt differens-/differential-forstærkerled. Denne ses på figur 15.2 Den fungerer således, at kun forskellen, differensen, på de to



Figur 15.2: Differensforstærker.

indgange, dvs. inputsignalet, forstærkes. Støj fra f.eks. lysnettet fjernes, da dette vil forekomme på begge indgange.

Operationsforstærkerne, vi har anvendt, er alle af typen LF411, hvor V_+ tilsluttes $+15V$ og V_- $-15V$.

LF411-operationsforstærkeren, har en Common Rejection Ratio på 100dB. Denne angiver forholdet mellem forstærkningen af differensen, dvs. det påtrykte signal, og forstærkningen af Common Mode (fælles-) signalet, dvs. støjen. Alle oplysninger angående LF411'eren stammer fra [18].

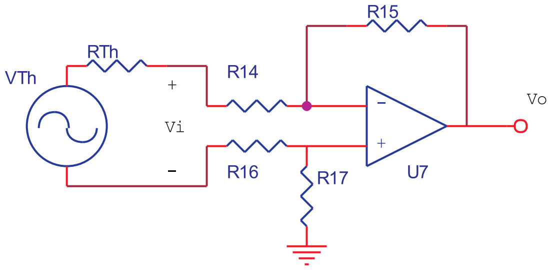
15.2.2 Isolering

Det ses af figur 15.3, at det tilkoblede udstyr, signalgiveren, vil påvirke forstærkningen i koblingen.

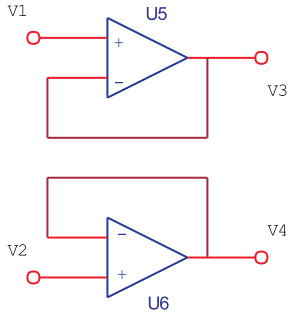
Dette kan beskrives således:

$$|A| = \frac{R_{15}}{R_{14} + R_{Th}} \tag{15.1}$$

Da dette vil forårsage en dæmpning af signalet og forringe signal-støjforholdet i filteret, tilkobles signalet direkte på plus-indgangen på to operationsforstærkere, såkaldte spændingsfølgere, hvis store indgansmodstand, $10^{12}\Omega$, således isolerer systemet. Koblingen ser ud som vist på figur 15.4



Figur 15.3: Thevenin-ækvivalent til signalgiver tilkoblet differensforstærkeren.

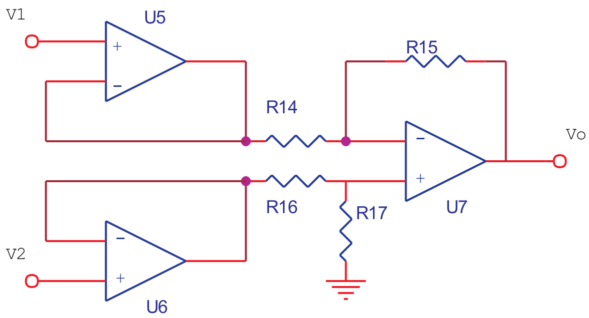


Figur 15.4: Isolering af systemet.

Indgangsspændingen, v_i, vil da være lig $v_1 - v_2$, og denne vil også ligge over $v_3 - v_4$, grundet den virtuelle kortslutning mellem plus- og minus-indgangene på en operationsforstærker.

15.2.3 Kobling

Den endelige kobling af differensforstærkeren ser med anvendte komponentværdier ud som vist på figur 15.5.



Figur 15.5: Samlet kobling.

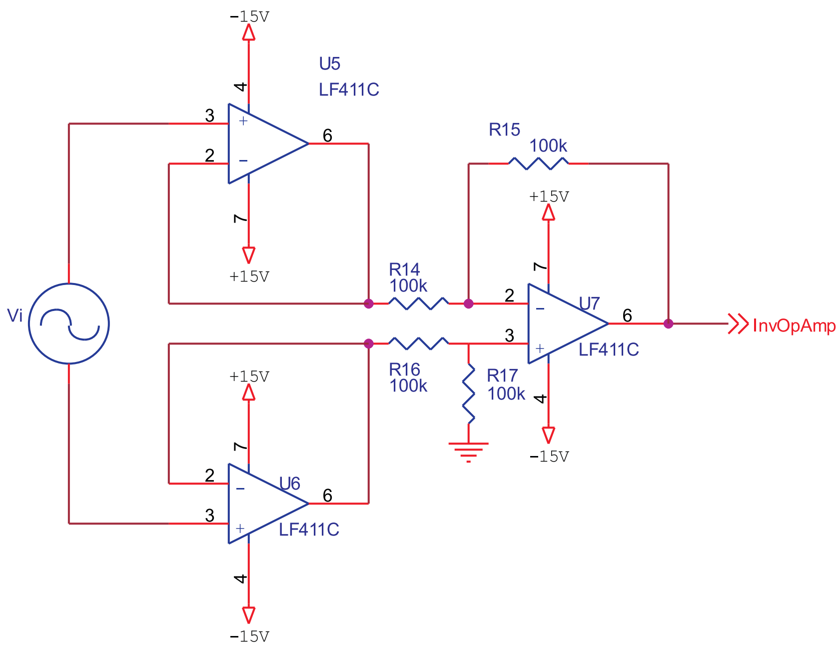
Udgangen på dette led tilsluttes så indgangen på den inverterende operationsforstærker, dvs. C_1, tilsluttet indgangen på MAX294 filteret.
Dette vil ikke påvirke dimensioneringen af C_1 i synderlig grad, som vist i afsnit 15.3.8, da den indre udgangsmodstand i operationsforstærkeren jo er lille.

15.2.4 Dimensionering

Forstærkningen igennem den samlede kobling er, når $R_{14} = R_{16}$ og $R_{15} = R_{17}$:

$$|A| = \frac{R_{15}}{R_{14}}(v_2 - v_1) \tag{15.2}$$

Vælges $|A| = 1$ er selve signaltilpasningen overladt til den inverterende operationsforstærkerkobling umiddelbart før MAX294'eren. Instrumentforstærkerens opgave er da at beskytte systemet og bortfiltrere støj.
Alle modstandene i denne kobling er valgt til $100k\Omega$.
Med anvendte komponentværdier kommer koblingen til at se ud som vist på firgur 15.6



Figur 15.6: Samlet kobling med anvendte komponentværdier.

15.3 Anti-aliaseringsfilter

15.3.1 Karakteristika

Vi har valgt et Switched-Capacitor filter til lavpasfiltrering af inputsignalet, nærmere bestemt MAX294. Alle oplysninger angående denne stammer fra [10]. Dette har den fordel, at afskæringsfrekvensen kan styres ved en ekstern clock, og filteret vil herfor kunne indgå i et system med variabel samplefrekvens op til 25kHz, der er filterets øvre grænse.
Filteret er et 8. ordens elliptisk filter. Det har en normeret stopbåndsfrekvens på $1,2$ og en stopbåndsdæmpning på $60dB$, passende til et 8-10bit system, og pasbåndsripple på $0,27dB$. Ved 10 bit er ADC'ens opløsningsevne (mindste registrerbare signalamplitudeændring [8, s. 561]) $0,098\%$, svarende til $60,2dB$.
Det elliptiske filters ringe fasekarakteristik (dvs. ikke lineær) er uden betydning, da kun amplitudekarakteristikken er af interesse (se kravspecifikationen).
Den normerede stopbåndsfrekvens på 1.2 og ripple bevirker, at amplitudespektret ikke vil modsvare inputsignalet (under stopbåndsfrekvensen).
Frekvenskomposanterne nær stopbåndsfrekvensen vil være behæftet med de mest betydelige aliaseringsfejl, og disse bør bortskæres i visningen af resultatet. Aliaseringsfejl vil dog altid forekomme, grundet den endelige dæmpning ved stopbåndsfrekvensen.

15.3.2 Clocking

Med en samplefrekvens, f_s på 44,1kHz, der er vores fastlagte maximum, skal stopbåndsfrekvensen være:

$$f_s = \frac{44,1kHz}{2} = 22,05kHz \tag{15.3}$$

Denne er også kendt som Nyquist-frekvensen. Dette giver følgende afskæringsfrekvens:

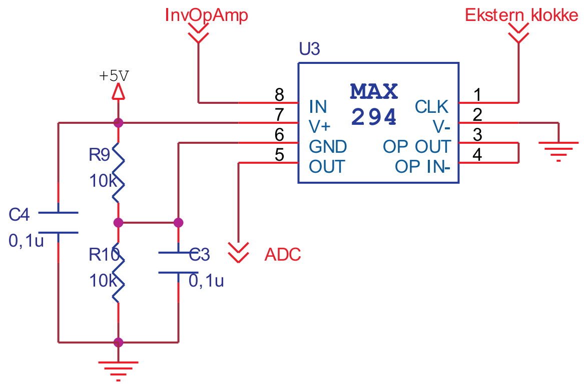
$$f_a = \frac{22,050kHz}{1,2} = 18,375kHz \tag{15.4}$$

Da forholdet mellem afskæringsfrekvensen og den eksterne klokke er $1:100$ fåes en klokfrekvens på:

$$f_{clk} = 100 \cdot 18,375kHz = 1,8375MHz \tag{15.5}$$

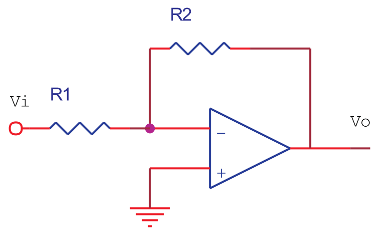
15.3.3 Kobling

Den i databladene anbefalede kobling (mht. afkoblingskapacitorer), og den beskrevne "Single supply operation" er anvendt. Dette ses på figur 15.7. Filterets V_- er forbundet til jord, V_+ til $5V$ spændingsforsyning og GND til 2,5V.



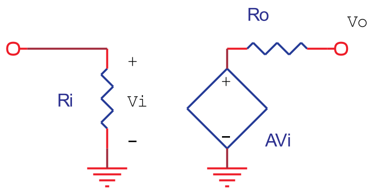
Figur 15.7: Anbefalet kobling af MAX294.

Til realisering af kravene til input til filteret og systemets input (2V peak to peak) anvendes en standardkobling af en inverterende operationsforstærker, som vist på figur 15.8. v_o tilsluttes indgangen på filteret.



Figur 15.8: Inverterende operationsforstærker.

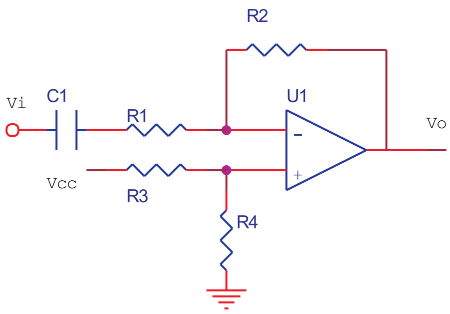
Operationsforstærkerens virkemåde kan illustreres med ækvivalentdiagrammet på figur 15.9. Spændingen over den uendeligt store, indre modstand, R_i, forstærkes med den i princippet uendeligt store råforstærkning, A. Udgangsmodstanden, R_o i operationsforstærkeren er lille, typisk af størrelsesordenen 70Ω.



Figur 15.9: Operationsforstærkerens ævkivalentdiagram.

15.3.4 DC-forspænding

Grundet forsyningsspændingen på MAX294'eren skal input på denne ligge imellem $V_{-} - 0.3V$ og $V_{+} + 0.3V$, og forstærkerkoblingen således hæve inputspændingen 2,5V DC-mæssigt.



Figur 15.10: DC-hævning af signal.

DC-forspændingen laves ved at placere en spændingsdeler, R_3 og R_4, på operationsforstærkerens plus-indgang (og en capacitor før R_1), som figur 15.10 viser. Følgende skal være opfyldt:

$$V = V_{cc} \frac{R_4}{R_3 + R_4} = 2,5V \tag{15.6}$$

Forsyningsspændingen, V_{cc}, er på +5V. Den DC-mæssige hævning på 2,5V af signalet fåes da ved $R_3 = R_4$. Disse er valgt til $100k\Omega$.

$$V = 5V \frac{100k\Omega}{100k\Omega + 100k\Omega} = 2,5V \tag{15.7}$$

Spændingsdeleren vil hæve $-$ indgangen på operationsforstærkeren, og capacitoren forhindrer DC'en i at løbe i tilbagekoblingen, og udgangspændingen vil herfor også være hævet de 2,5V.

15.3.5 AC-forstærkning

Når input på filteret varierer med 3V peak to peak er den harmoniske forvrængning og støj på sit laveste, dvs. det bedste signalstøjforhold opnåes. Forstærkningsfaktoren for AC-signaler for koblingen bliver -1,5, da der er tale om en inverterende kobling.
Forstærkningen realiseres som forholdet mellem R_1 og R_2.

$$|A| = \frac{R_2}{R_1} \tag{15.8}$$

Vælges R_1 til 100kΩ fåes

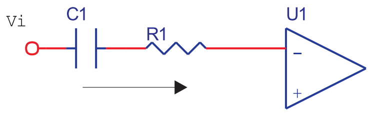
$$R_2 = R_1 \cdot |A| = 100k\Omega \cdot 1,5 = 150k\Omega \tag{15.9}$$

15.3.6 DC-mæssig afbrydelse

For at afskære et eventuelt input DC-offset eller -hævning fra systemet (og DC-forspændingen skal virke) indsættes en kapacitor, C_1. Denne kan dimensioneres ud fra følgende:

$$C_1 = \frac{1}{\omega \cdot R_1} \tag{15.10}$$

Princippet kaldes for tidskonstantmetoden. ω er afskæringsfrekvensen, her valgt til 10Hz. Modstanden, R_1, er den modstand C_1 ser ind i, som vist på figur 15.11.

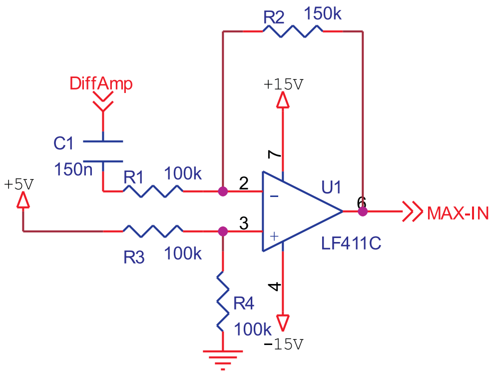


Figur 15.11: Modstanden, C_1 ser ind i.

C_1 bliver da:

$$C_1 = \frac{1}{20\pi\frac{rad}{s} \cdot 100k\Omega} \approx 160nF \tag{15.11}$$

Dette skulle give en 3dB grænse ved 10Hz. Tilpas lavt til at opnå en meget lille indvirkning på frekvenser over 20Hz.
Bemærk at denne faktiske afskæringsfrekvens vil variere i forhold til den teoretiske på grund af den indre indgangsmodstand i operationsforstærkeren.
Indsættelsen af denne kapacitor har den konsekvens, at det ikke vil være muligt at måle DC-komposanten i signalet gennem frekvensanalysen. Figur 15.12 viser den anvendte kobling af operationsforstærkeren til realisering af kravende til input på filteret.



Figur 15.12: Anvendt kobling af operationsforstærker.

15.3.7 Overføringsfunktion

Output som funktion af inputspændingen, der ses bort fra DC-hævningen, kan skrives:

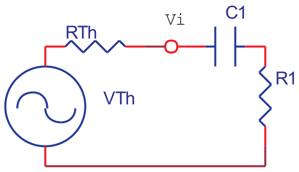
$$v_o = -v_i \frac{R_2}{R_1 + \frac{1}{sC_1}} = -v_i \frac{\frac{R_2}{R_1} s}{s + \frac{1}{R_1 C_1}} \tag{15.12}$$

Overføringsfunktionen for AC-signalet kan da opstilles som:

$$H(s) = \frac{v_o}{v_i} = \frac{-\frac{R_2}{R_1} s}{s + \frac{1}{R_1 C_1}} \tag{15.13}$$

15.3.8 Impedanser

Figur 15.13 viser Thevenin-ækvivalenten til differensforstærkeren, tilkoblet den inverterende forstærkerkobling.



Figur 15.13: Indgangsimpedans.

Spørgsmålet er, i hvilken grad dette påvirker koblingen, specifikt med hensyn til den nedre grænsefrekvens.

Som det ses af diagrammet, sidder theveninmodstanden, R_{Th}, i serie med R_1. Sammenhængen mellem afskæringsfrekvensen og komponenterne skrives:

$$\omega = \frac{1}{C_1 \cdot (R_1 + R_{Th})} \tag{15.14}$$

Dette viser, at afskæringsfrekvensen er omvendt proportional med summen af R_1 og R_{Th}. Afskæringsfrekvensen bliver da aldrig højere end den, der er dimensioneret ud fra. I praksis er det intet problem, da den indre udgangsmodstand i operationsforstærkeren, som C_1 jo ser ind i, er lille. Af samme årsag er udgangsimpedansen på den inverterende forstærkerkobling og dennes påvirkning af MAX294'eren ingen grund til bekymring.

15.4 Analog/Digital Converter

15.4.1 Karakteristika

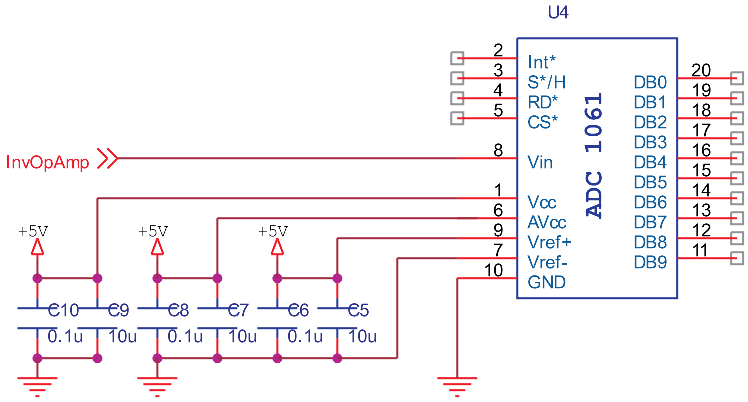
AD1061 er en 10-bit analog til digital converter baseret på CMOS-teknologi fra National Semiconductor. Det vil sige, at vi med denne komponent kan sample med en opløsning op til 10-bit (1024 kvantiseringstrin), hvilket giver en signalstøjforhold på 60.2dB, hvilket svarer til en opløsningsevne på 0,098%. Op til, da man f.eks. kan få en 8-bit converter ved at ignorere de to mindst betydende bit (LSB's). Alle oplysninger angående A/D converteren stammer fra [19].
Signalstøjforholdet kan bestemmes ud fra:

$$\frac{S}{N}[dB] = 20 \cdot log2^n[dB] \tag{15.15}$$

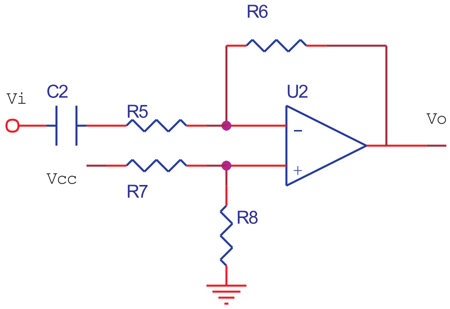
Hvor n er antal bit.
Samplefrekvensen kan gå op til 160kHz, hvilket er langt over vores behov.

15.4.2 Kobling

Den i databladene anbefalede kobling af den valgte A/D Converter, ADC1061, er med hensyn til afkoblingskapacitorer, forsynings- og referencespændinger anvendt. Figur 15.15 viser dette.
Bemærk de separate jordforbindelser for henholdsvis analoge og digitale dele.



Figur 15.14: Anbefalet kobling ad ADC1061.



Figur 15.15: Operationsforstærker til tilpasning af input med valgte komponent navne/numre.

For at tilpasse outputsignalet fra MAX294-filteret indsættes en inverterende operationsforstærker i en kobling som vist på figur 15.15.
Denne dimensioneres efter samme princip som operationsforstærkeren mellem signalgiveren og MAX294'eren.

15.4.3 DC-forspænding

Med V_{ref-} forbundet til jord og V_{ref+} forbundet til $+5V$ spændingsforsyning skal input til ADC'en ligge mellem 0 og $5V$. Dette gøres ved, som før, at hæve inputtet $2,5V$ DC-mæssigt.
Den eneste forskel er, at vi her har valgt at anvende et $22k\Omega$ potentiometer (R_8), der tillader os af finjustere DC-hævningen. Dette forbedrer kvaliteten af A/D konverteringen.
R_7 er valgt til $10k\Omega$

15.4.4 AC-Forstærkning

Signalet, der kommer fra filteret, er på $3V$ peak to peak. Dette skal så forstærkes op til $5V$ peak to peak. Dette giver følgende forstærkning:

$$|A| = \frac{v_o}{v_i} = \frac{R_6}{R_5} = \frac{5V}{3V} \tag{15.16}$$

Der er krav til indgangsimpedansen af den inverterende forstærkerkobling, fra MAX294'eren. Vi har herfor valgt $R_5 = 20k\Omega$, hvilken svarer til den belastning, der er anvendt i databladene. R_6 bliver da:

$$R_6 = R_5 \cdot |A| = 20k\Omega \cdot \frac{5}{3} = 33,3k\Omega \tag{15.17}$$

Forstærkningen i denne kobling udgør den største i systemet, og det bør undersøges, om LF411'eren går i mætning eller Slew Rate'en vil påvirke koblingen. Forstærkningen i dB er:

$$20 \cdot log|A| = 20 \cdot log\frac{5}{3} = 4,44dB \tag{15.18}$$

Betrages Open Loop Gain grafen i databladet ses det, at de 4,44dB først vil afstedkomme påvirkning af Slew Rate'en omkring 1MHz, det vil sige langt uden for vort frekvensområde. De 4,44dB er desuden langt under mætningsgrænsen.

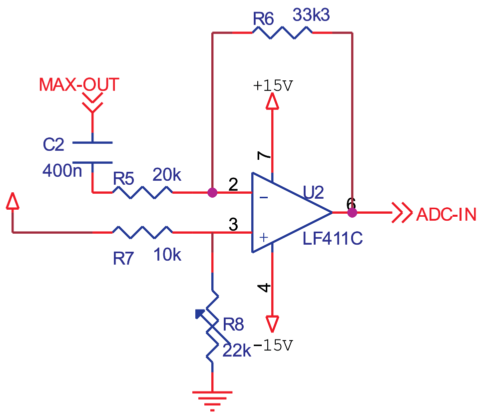
15.4.5 DC-mæssig afbrydelse

Da MAX294-komponenten har et DC-offset, og dette vil medføre dårligere præcision i A/D konverteringen, indsættes en kapacitor mellem indgangen på den inverterende forstærkerkobling og udgangen på filteret. Kapacitoren, C_2, bestemmes ganske som C_1. Dette giver:

$$C_1 = \frac{1}{20\pi\frac{rad}{s} \cdot 20k\Omega} \approx 800nF \tag{15.19}$$

Den faktiske afskæringsfrekvens vil afvige, da både udgangen på MAX294-filteret og den indre modstand i operationsforstærkeren også har indflydelse. Resultatet af dimensioneringen ses på figur 15.16

Det bemærkes, at de to signaltilpasningskoblinger, tilsammen vil give en dæmpning på 6dB, dvs. en halvering af signalet, i 10Hz, da der jo er tale om en kaskadekobling af to 1. ordenssektioner.



Figur 15.16: Anvendt kobling af operationsforstærker.

Kapitel 16

Controller i Altera

16.1 Controllerens opdeling

Controlleren er den del af systemet, som skal sørge for at styre en række forskellige specifikke opgaver. Dens primære opgaver er:

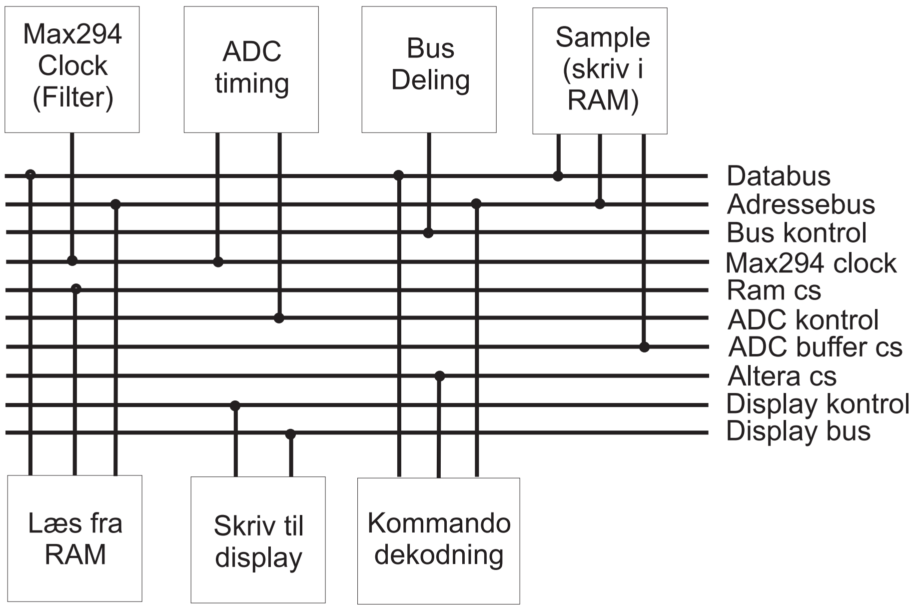
- Direct Memory Access fra ADC til RAM, herunder:
  - Styring af ADC
  - Bus deling
  - Skrivning i RAM (herunder bitreversering og pakning)
- DMA fra RAM til Display, herunder:
  - Bus deling
  - Læsning fra RAM
  - Skrivning til display

Derudover er der nogle sekundære opgaver, som er med i controlleren fordi de er stærkt relaterede til de primære funktioner, eller fordi de kan udnytte den plads der er tilovers i den programmerbare logik:

- Kommando dekodning:
  - Skal aktivere de primære funktioner via kommandoer fra MCU'en

- Filter clock:
  - Skal sende et clocksignal med en frekvens proportional med ADC-samplefrekvensen til det analoge filter
- Neddividering af systemclock
  - Dividerer 32MHz clock til MCU'ens 16MHz. Kunne godt have været en ekstern enhed, men er med i controlleren, da det kun kræver kun een flipflop

På figur 16.1 ses en oversigt over de forskellige moduler og deres tilhørende signaler/busser.



Figur 16.1: Controllerens moduler forbundet til de tilhørende signaler/busser.

16.2 Busdeling - Enhedernes prioritet

Fælles for de to DMA-dele er at de sammen med MCU'en skal dele data- og adressebus, som er forbundet til RAM-kredsene. Dette betyder i praksis at kun een af delene kan skrive eller læse fra hukommelsen ad gangen. For at fordele denne resource (bussen) bedst muligt, er det nødvendigt at prioritere DMA-enhederne:

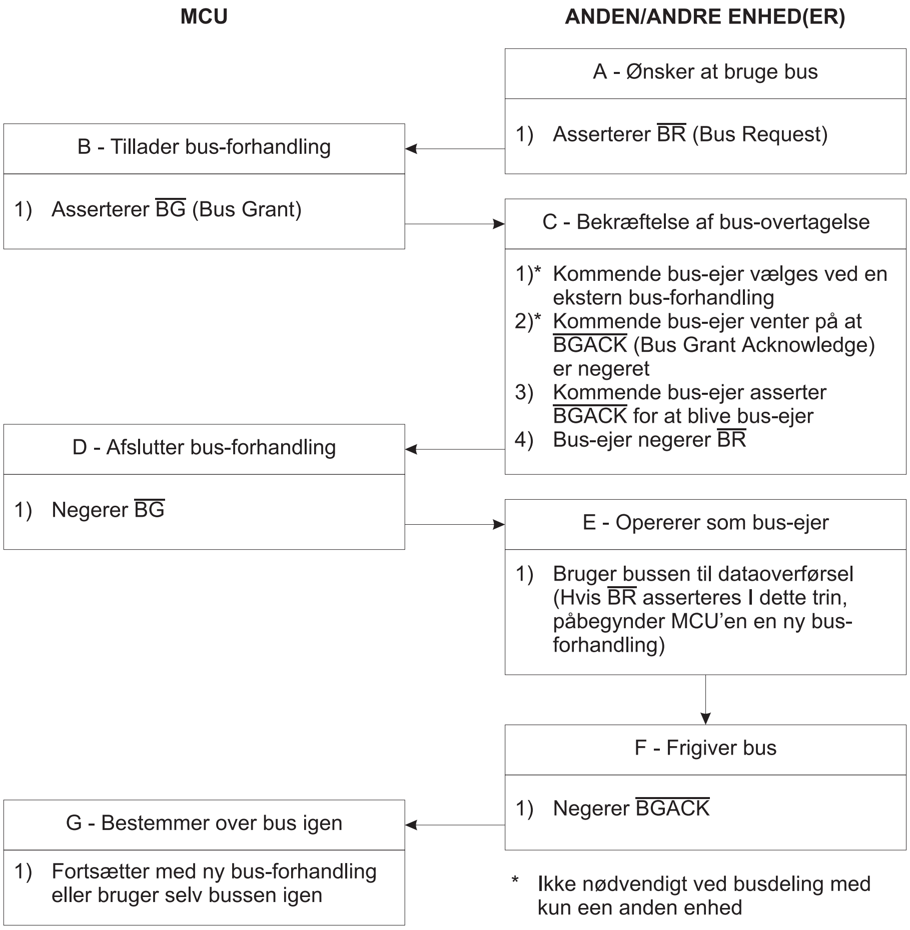
Sampledelen skal bruge bussen et bestemt antal gange i sekundet (afhængigt af samplefrekvensen) og skal kunne nå at skrive data i hukommelsen inden for en bestemt periode (som afhænger af ADC'ens sampleperiode, integrationstid og konverteringstid). Hvis disse tidskrav ikke overholdes, mistes der samples, og målingerne bliver ubrugelige. Derfor vil vi tildele denne del den højeste prioitet på bussen.
Formålet med display-DMA er at aflaste MCU'en, så denne kan bruge tiden på FFT-beregningerne, og dermed forøge dataoverførselshastigheden til displayet. DMA-controlleren skal derfor læse fra hukommelsen og skrive til displayet så ofte som muligt. Dette betyder dog ikke at bussen vil blive belastet af display-DMA hele tiden, da displayet har en flaskehals, som sætter en grænse for hvor ofte det er muligt at overføre data til displayet. Displayet ligner altså med sine periodiske dataoverførsler sampledelen, men i modsætning til i sampledelen er en forsinkelse er ikke katastrofal i displaydelen. Vi tildeler derfor displayet anden prioritet på bussen.
MCU'en skal regne på FFT når de andre enheder ikke bruger bussen. MCU'en får laveste prioritet på bussen. MC68331-MCU'en får automatisk altid laveste prioritet når bussen deles, da denne er opbygget på en måde så den altid er villig til at frigive bussen efter hver data-overførsel [16, s. 4-41].

16.3 Busdeling - Mellem MCU og een enhed

Busdeling mellem MCU'en og een anden enhed foregår ved firevejs handshake mellem enhederne. På MC68331-MCU'en bruges tre ben til formålet: $\overline{BR}$ (Bus Request), $\overline{BG}$ (Bus Grant) og $\overline{BGACK}$ (Bus Grant Acknowledge). Busdelingen foregår ved, at den eksterne enhed fortæller MCU'en, at den vil benytte bussen ved at assertere $\overline{BR}$. Når MCU'en er klar til at frigive bussen, fortæller den til enheden, at bussen er fri med en assertering af $\overline{BG}$. I den periode, hvor enheden benytter bussen, skal dette bekræftes ved at assertere $\overline{BGACK}$. Når enheden er færdig med at benytte bussen, skal $\overline{BGACK}$ negeres for at give bussen tilbage til MCU'en.

16.4 Busdeling - Mellem MCU og flere enheder

Ved busdeling mellem MCU'en og to eller flere enheder fungerer den omtalte busdelingsprotokol på en udvidet måde. $\overline{BG}$ signalet skal i dette tilfælde ikke sendes direkte til alle



Figur 16.2: Princip for busdeling mellem to eller flere enheder.

enheder, men skal igennem en ekstern fordelingsenhed, som står for prioriteringen af enhederne. Når en enhed ønsker at bruge bussen, asserterer den $\overline{BR}$. MCU'en svarer med $\overline{BG}$ signalet, når den er klar til priotiteringsenheden. Prioriteringsenheden sender nu $\overline{BG}$ signalet videre til den rigtige enhed. Da MCU'en godt kan sende $\overline{BG}$ signalet mens den selv ikke ejer bussen, skal enheden tjekke, om en anden enhed benytter bussen ved at se om $\overline{BGACK}$ er asserteret. Hvis ingen andre enheder bruger bussen, asserterer enheden nu $\overline{BGACK}$ for at vise overfor MCU'en og andre enheder, at den nu ejer bussen. Når enheden ikke ønsker at bruge bussen mere, negerer den $\overline{BGACK}$, og MCU'en eller en anden enhed overtager nu bussen.

16.5 Busdeling - Implementationsmuligheder for prioriteringsenhed

Prioriteringsenheden kan implementeres på forskellige måder, hvoraf vi har overvejet to mulige grundprincipper:

1. Central, parrallel prioritering, hvor alle enhederne parallelt forbindes til prioriteringsenheden, med hver sine busdelings-signaler.

   Fordele:

   - Høj hastighed.
   - Ingen problemer med udelukkelse af enheder med lav prioritet.

   Ulemper:

   - Mere kompleks HW.

2. "Daisy Chain", seriel prioritering, hvor enhederne forbindes gennem en kæde, hvor enheden med højest prioritet placeres øverst i kæden, tættest på MCU'en. $\overline{BR}$ og $\overline{BGACK}$ signalerne forbindes som en bus direkte til alle enhederne i en open drain / collector konfiguration. $\overline{BG}$ signalet sendes ned gennem enhederne således, at hvis en enhed med højere prioritet modtager signalet, og enheden ikke har bedt om bussen, sendes signalet længere ned til enheder med lavere prioritet.

   Fordele:

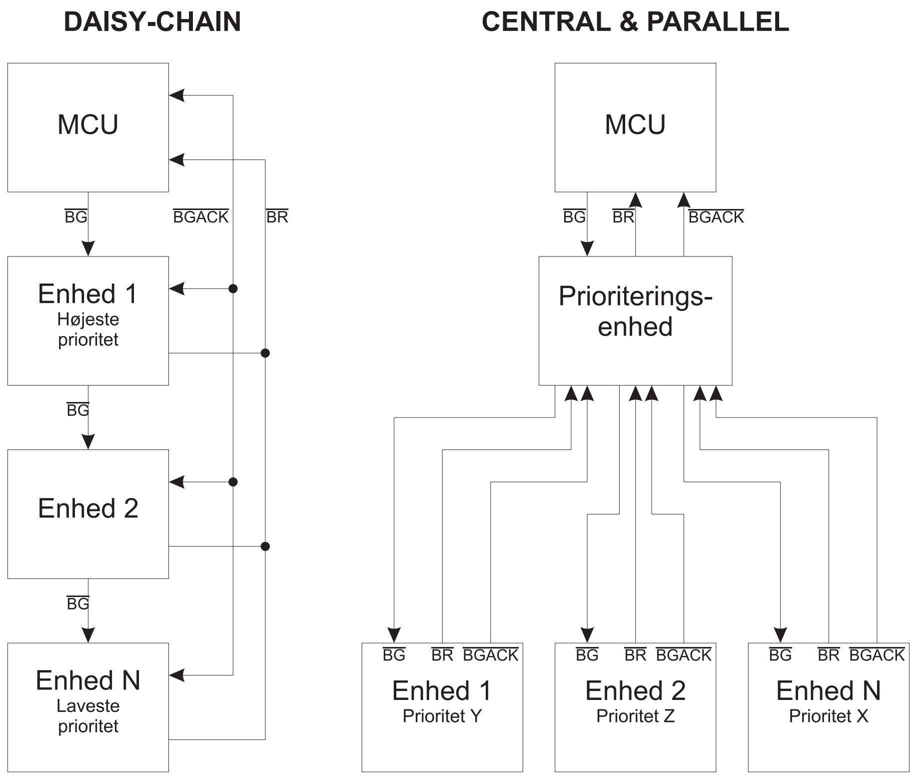
   - Simpel HW.

   Ulemper:

   - Enheder med lav prioritet kan blive lukket helt ud af bussen, hvis enheder med højere prioritet bruger bussen for ofte (starvation).
   - Hastighed; der kommer en forsinkelse for hvert led i kæden.

På figur 16.5 ses et diagram over de to konfigurationers opbygning.
Vi har valgt at benytte "Daisy-Chain" konfigurationen, da:

- Hastighedsproblemet har næsten ingen betydning ved kun to enheder i kæden.



Figur 16.3: Mulige implementationer af prioritering ved busdeling.

- Enheden med højest prioritet skal bruge bussen med fast interval og kun i korte perioder ad een skrivecyklus, hvilket giver garanti for, at enheden med anden prioritet ikke bliver lukket ude.
- Vi har begrænset areal i den benyttede programmerbare logik.

16.6 Sampling DMA

16.6.1 Opdeling

Denne del af controlleren skal sørge for dataopsamlingen fra ADC. Vi har valgt en løsning, hvor vi fra MCU'en kan bestemme samplefrekvensen, samplebufferstørrelsen og placeringen af samplebufferen i hukommelsen, da dette er en mere generel løsning. Dette giver f.eks. mulighed for en fremtidig MMI (Man-Machine Interface), hvorfra man vil kunne stille på faktorer som samplefrekvens og FFT-størrelse.
Samplebufferstørrelsen skal kunne sættes til størreslser der direkte passer til FFT-algoritmen,

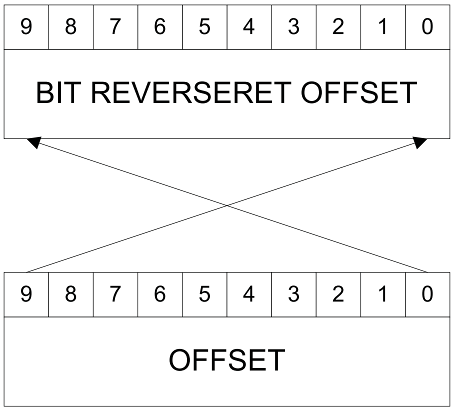
dvs størrelsen 2^n. Den maksimale nødvendige bufferstørrelse er blevet fastsat til 2048 samples udfra estimeringer af udførselstiden af en 2048 punkts FF.
Vi har valgt at lave DMA-controlleren på en måde, hvor een samplebuffer i systemhukommelsen fyldes op med sampledata, hvorefter controlleren sender et interrupt til MCU'en for at indikere at der er sampledata klar. Der er i vores system ikke brug for en kontinuert strøm af data. Ønskes en sådan, kan ISR'en sætte DMA-controlleren i gang med at sample til en anden buffer, mens MCU'en regner på data fra den første samplebuffer. (Dette kræver dog at ISR'en er hurtig nok til at der ikke mistes samples, hvilket vi ikke har undersøgt.)
Dataopsamlingen er delt op i to hoveddele, hvor den første står for timingen af ADC'en (afsnit 16.6.3). Når ADC'en er klar med sampledata, går del to i gang med at overtage bussen og skrive samplen i RAM (afsnit 16.6.4).

16.6.2 Samplebuffer addressering

Før DMA-overførslen af en sample skal den adresse som samplen skal placeres i beregnes. Ved en lineær skrivning af sampledata kan beregningen af sampleadressen skrives som: $P(m) = BASE + m$, hvor $P(m)$ er den beregnede sampleadresse, $BASE$ er baseadressen, og m er index som tæller fra 0 til $2^n - 1$.
Denne funktionalitet kan i en Altera PLD implementeres vha. en full-adder. Det viser sig dog at man ved ret små ændringer kan implementere permutation (se beskrivelse i FFT-teorien, A) med variabel bufferstørrelse.

Permutation med fast bufferstørrelse

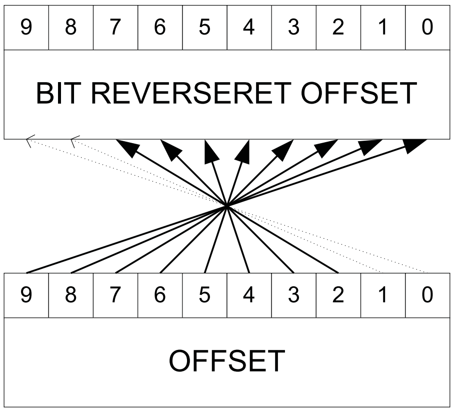
Permutation med fast bufferstørrelse kan, som beskrevet i Appendix A, udføres som en bitreversering, dvs. en ombytning af adressetællerens bits, så de højestbetydende bits bliver til mindstbetydende bits og omvendt. Hvis vi lader $BR(m)$ være en bitreversering af m, kan vi skrive den beregnede baseadresse som $P(m) = BASE + BR(m)$.
Denne bitreversering kan i hardware laves ved at forbinde de enkelte adressebits i adressetælleren omvendt som på figur 16.4.



Figur 16.4: Bitreversering med fast bufferstørrelse.

Permutation med variabel budfferstørrelse

Skal bitreverseringen udføres med variabel bufferstørrelse kan man ikke umiddelbart implementere dette i hardware, da dette ville kræve en forskellig krydsning af adressebit'ene ved de forskellige bufferstørrelser. Dette kunne løses vha. et skifteregister, men en sådan løsning er både langsom og kræver meget chipareal. Brugen af skifteregister kan undgåes, hvis en anden teknik benyttes:
Alle bits i m reverseres. Men i stedet for at indexregistret m løber fra 0 til 2^n med spring på 1, skal springene være $s = 2^k$, hvor k er forskellen mellem antallet af bits i m og antallet af bits som skal bruges. Se eksempel figur 16.5



Figur 16.5: Bitreversering med variabel bufferstørrelse.

På figur 16.5 ses bitreverseringen for 1024 samples, dvs. 10 bit. Hvis bitreverseringen skulle udføres på 256 samples, skal der kun bruges 8 bits. Foreskellen i antallet af bits

i m er $k = 10 - 8 = 2$. Det ses tydeligt at hvis den bitreverserede adresse skal have et adresserum fra 0 til 255, skal de to mest betydende bits ikke være sat. Altså skal de to mindst betydende bits på den ikke reverserede adresse ikke tælles op. Tælningen skal starte på bit 2. En sådan tælning betyder at der skal tælles $s = 2^k = 2^2 = 4$ op for hver adresseberegning.
Læg mærke til, at 2^n nu kun er bufferstørrelsen, hvis springene s er 1. Hvis s er > 1 er bufferstørrelsen lig $\frac{2^n}{s}$, hvorved s fungerer som en bufferstørrelse-divider. Ud fra dette kan vi se at tælleren altid skal tælle ved at sammenligne med tallet 2^n, som er den største bufferstørrelse hvis n bits benyttes.
Beregningen af den nye pointer $P(m)$ kan beskrives med følgende pseudokode:

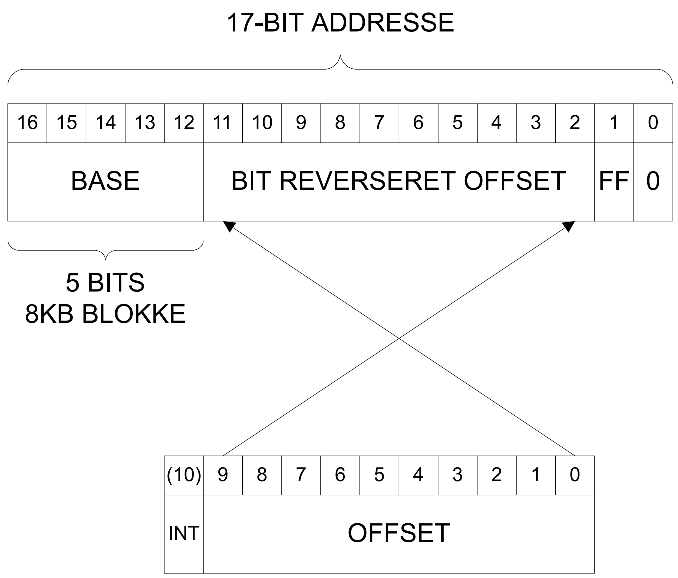
```
m:=0;
repeat
 P(m):=BASE+BR(m);
 WRITE_TO_MEM;
 m:=m+s;
until m>=(2^n);
SEND_INTERRUPT;
```

Permutation med variabel budfferstørrelse og pakning

Den i appendix 10 beskrevne pakning kan i hardware implementeres med en rulning af $BR(m)$ med 1 bit til venstre. Da en rulning er en tids- og arealkrævende funktion i hardware, har vi fundet en anden teknik:
Vi kender "opførslen" af den mest betydende bit i $BR(m)$, idet vi ved at den mindst betydende bit i m altid flipper. Efter en rulning til venstre vil denne mest betydende bit blive skubbet ud og blive sat ind som den mindst betydende bit. Vi kender altså opdførslen af den mindst betydende bit af den rullede $BR(m)$, og erstatter denne med en flip-flop, som flippes for hver adresseberegning. En forskydning af de resterende bits laves ved kun at tælle m op hver anden gang. Dette svarer til et skift til højre i m, og altså et skift til venstre i $BR(m)$.
Pseudokoden for denne beregning er følgende:

```
m:=0;
FF:=0;
```

```
repeat
 P(m):=shl(BASE+BR(m))+FF;
 WRITE_TO_MEM;
 FF:=(FF+1) and 1;
 if FF=0 then m:=m+s;
until m>=(2^n);
SEND_INTERRUPT
```



Figur 16.6: Bitreversering med variabel bufferstørrelse og pakning.

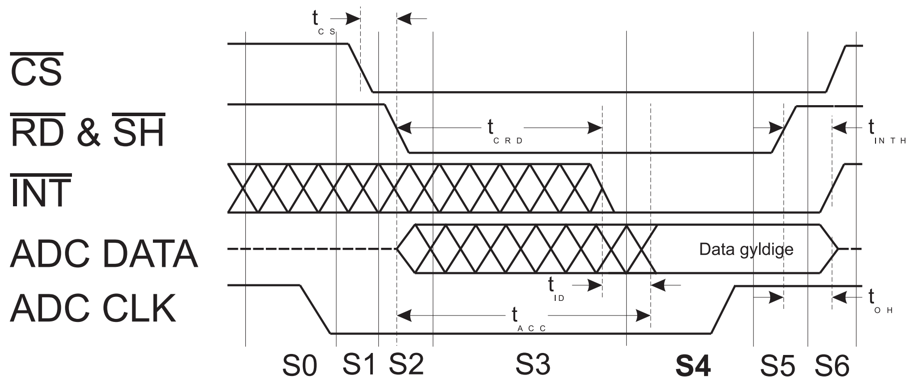
På figur 16.6 ses princippet i den færdige implementation af adresseberegningen med permutation og pakning. Adressen er beskrevet med 17 bits, som svarer til et adresserum på 128kWords. Bit 0 er altid 0, da de samplede data skal benyttes i en 16:16-fixed point konfiguration i softwaren, og kommatalsdelen ønskes sprunget over. Bit 1 er Flip-flop'en, Bits 2 til 11 er det bitreverserede offset, og de 5 øverste bits bestemmer baseadressen.

Da der kun bruges 5 bits til baseadressen, kan der kun bestemmes 32 forskellige baseadresser, hvilket svarer til spring på 4kWords. Denne begrænsning har ikke praktisk betydning i softwaren, men sparer en ekstra full-adder i hardwaren. Det ikke bitreverserede offset består af i alt 11 bits, hvor den 11. bit bruges for at simplificere sammenligningen

af tælleren med den maksimale bufferstørrelse. Når den 11. bit bliver sat, betyder det at tælleren har nået op på bufferstørrelsen og at der skal sendes et interrupt.
Beregningen af den nye pointer skal ske lige efter hver skrivning til RAM, således at den nye adresse er klar så så tidligt som muligt.

16.6.3 ADC-styring

ADC'en skal sample med en fast periode, som vi lader blive bestemt af en samplefrekvens-clock. Denne bliver genereret i controlleren ud fra en hurtigere clock (se 16.8.1). Vi lader ADC'en sample hele tiden for at undgå opstartsproblemer og bruger kun dens data, når vi har brug for dem. Samplingsforløbet ses på figur 16.7.



Figur 16.7: ADC timing.

Betegnelse	Beskrivelse	Værdi
t_{CS}	Forsinkelse fra $\overline{CS}$ til $\overline{RD}$ og $\overline{SH}$	min 0 ns
t_{CRD}	Integrationstid + Konverteringstid	max 2,4 μs
t_{ACC}	Access-tid	$t_{CRD} + 50ns$
t_{ID}	Forsinkelse fra $\overline{INT}$ til outputdata gyldig	max 50 ns
t_{INTH}	Forsinkelse fra $\overline{RD}$ til $\overline{INT}$	max 50 ns
t_{OH}	Holdetid fra $\overline{RD}$ til outputdata i højimpedans	max 50 ns

Figur 16.8: ADC timingsværdier.

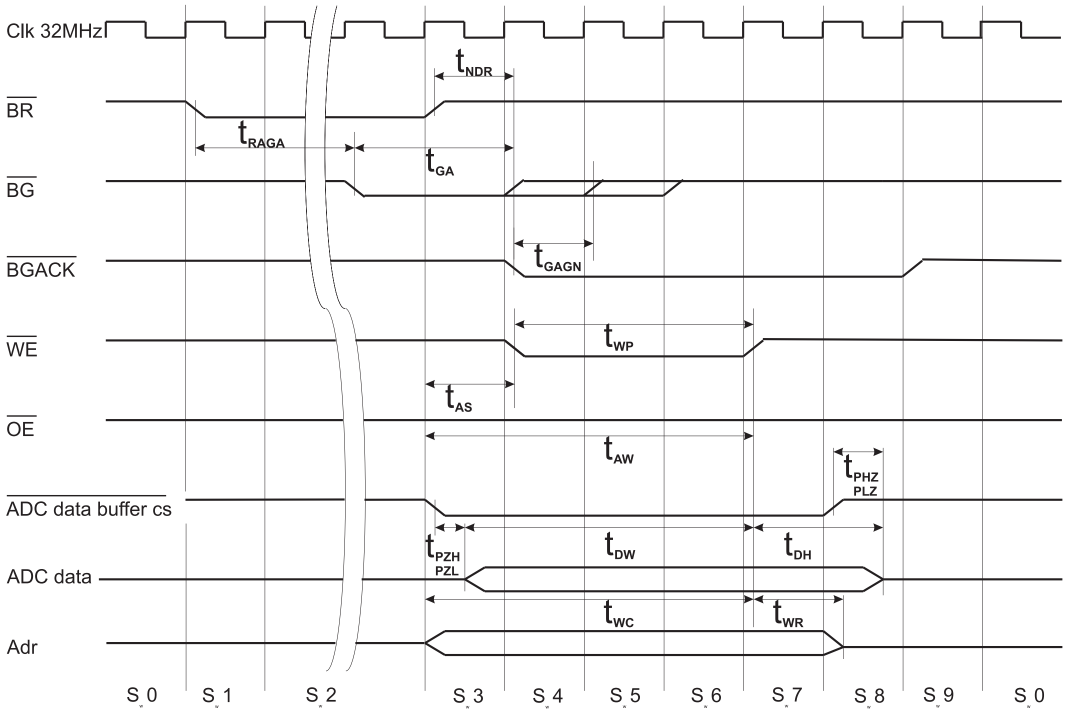
På figur 16.8 ses timingsværdierne. Vi har valgt at opfylde timingen vha. en state-machine, som i State 0 starter med at vente på samplefrekvens-clock'ens nedadgående flanke. S1

og S2 bruges til at sætte ADC'en i gang ved at sætte $\overline{CS}$ lav i S1 og umiddelbart efter sætte $\overline{RD}$ og $\overline{SH}$ lav i S2. I S3 ventes der nu indtil ADC'en har sampledata klar ved at vente på $\overline{INT}$ signalet fra ADC'en. Når data er klar, kan der gås i gang med at overtagelse af bussen og skrivning til RAM i en anden statemaskine (se 16.6.4), hvilket sker i S4. I S4 ventes der nu på saplefrekvens-clock'ens opadgående flanke. Når dette sker fortsættes til S5 og S6, hvor hele samplingcyklusen afsluttes ved at sætte $\overline{RD}$ og $\overline{SH}$ høj i S5 og sætte $\overline{CS}$ høj i S6.
Statemaskinen styres af en 32 MHz clock, som gør at forsinkelsen mellem to states mindst kan blive 31,25 ns, hvilket gør, at ADC'ens timingskrav for overgangen fra S1 til S2 og S5 til S6 overholdes.
Det ses på figur 16.7 at der er en forsinkelse, t_{ID} = max 50 ns, på ADC-outputdataene. Det er dog ikke noget problem, at DMA og skrivning til RAM startes under denne forsinkelse (i S4), da selve busovertagelsen tager minimum 3 states svarende til minimum 84,375 ns. På figuren ses det også, at ADC'en sender data ud på bussen under hele samplingsforløbet, selvom vi kun skal bruge dens data under skrivning til RAM. For ikke at blokere databussen i samplingsforløbet har vi placeret tristate-buffere mellem udgangen af ADC'en og databussen (se 16.6.5).

16.6.4 Busovertagelse og RAM-skrivning

Vi har placeret busovertagelsessekvensen og RAM-skrivningssekvensen i een stor statemachine, da disse altid skal udføres lige efter hinanden. En opdeling i to state-machines ville give et (lille) delay mellem de to dele, hvilket ikke ønskes, da vi skal benytte bussen i så kort tid som muligt.
Sample-DMA'en starter med en overtagelse af bussen. Dette sker i overensstemmelse med den tidligere beskrevne protokol for busdeling mellem flere enheder (se. 16.4), samt i en daisy-chain konfiguration (se. 16.5). På figur 16.9 ses et timingsdiagram over busovertagelsen og RAM-skrivningen, med tilhørende timingsværdier på figur 16.10.
State 0 i sekvensen (S_w0) bruges udelukkende når DMA-sekvensen ikke er i gang. Der fortsættes først til S_w1 når følgende er opfyldt:

- Der ønskes DMA-sampling (hertil benyttes et bestemt sampling-flag, som bliver aktiveret ved en bestemt kommando fra MCU'en, se 16.8.2)
- State-machinen for ADC-styring er i S4, se 16.6.3



Figur 16.9: Sample-DMA timing.

I S_w1 asserteres $\overline{BR}$ og der fortsættes til S_w2, hvor der nu ventes på at $\overline{BG}$ bliver asserteret af MCU'en, og testes om $\overline{BGACK}$ er negeret. Er kravene opfyldt, fortsættes der til S_w3, hvor man antager at man ejer bussen. Ifølge busdelingsprotokollen (se 16.2) skal $\overline{BGACK}$ nu asserteres efterfulgt af en negering af $\overline{BR}$. Denne løsning vil dog volde problemer, hvis der er et delay, som gør at $\overline{BG}$ negeres af MCU'en før controlleren negerer $\overline{RB}$. Dette delay må maksimalt være på t_{GAGN}, hvilket kunne opstå hvis $\overline{BR}$ blev negeret i samme state som $\overline{BGACK}$ blev asserteret i, og den benyttede PLD havde forskellige propagation delays for de to signaler. Vi har på dette grundlag tilføjet et timingskrav, t_{NDR}, på minimum 0 ns, som i state-machinen kan implementeres ved først at negere $\overline{BR}$ i S_w3 og derefter assertere $\overline{BGACK}$ i S_w4. Vi har valgt at sende data og addresse ud på data- og adressebussen allerede i S_w3, da det reelt er i denne state controlleren ejer bussen. $\overline{WE}$ asserteres først i S_w4 for at overholde adressesetuptiden t_{AS} på 0 ns. De efterfølgende to states, S_w5 og S_w6 er inkluderet for at sikre RAM-kravene t_{WP}, t_{AW}, t_{DW} samt t_{WC}. I S_w7 afsluttes selve skrivecyklen ved at negere $\overline{WE}$. Den efterfølgende S_w8 er der for at sikre RAM-kravene t_{DH} og t_{WR} på 0 ns. S_w8 udnyttes også til at flippe filp-flop'ens tilstand ved adresseberegning, se 16.6.2. I S_w9 frigives bussen igen ved at negere $\overline{BGACK}$. S_w9 udnyttes også til

Betegnelse	Enhed	Beskrivelse	Værdi
t_{NDR}	MCU (krav)	$\overline{BR}$ skal negeres inden MCU'en negerer $\overline{BG}$	min 0 ns
t_{GA}	MCU	$\overline{BG}$ bredde asserteret	min 1 $t_{cyc} = 59,6ns$
t_{BRAGA}	MCU	$\overline{BR}$ asserteret til $\overline{BG}$ asserteret	min 1 $t_{cyc} = 59,6ns$
t_{GAGN}	MCU	$\overline{BGACK}$ asserteret til $\overline{BG}$ asserteret	min 1 $t_{cyc} = 59,6ns$ max 2 $t_{cyc} = 119.2ns$
t_{WP}	RAM (krav)	Write Pulse Width	min 55 ns
t_{AS}	RAM (krav)	Address Setup Time	min 0 ns
t_{AW}	RAM (krav)	Address Enable Time	min 60 ns
t_{DW}	RAM (krav)	Input Data Setup Time	min 30 ns
t_{DH}	RAM (krav)	Input Data Hold Time	min 0 ns
t_{WC}	RAM (krav)	Write Cycle Time	min 70 ns
t_{WR}	RAM (krav)	Address hold time	min 0 ns
t_{PZH}/t_{PZL}	Buffer	3-state output enable time	max 38 ns
t_{PHZ}/t_{PLZ}	Buffer	3-state output disable time	max 38 ns

Figur 16.10: Sample-DMA timingsværdier.

at inkrementere adressetælleren i adresseberegningen, se 16.6.2. Der hoppes først igen til S_w0 når state-machinen for ADC-styring er i S4, se 16.6.3.

16.6.5 Hardwareopbygning

Sampling DMA-delen implementeres i en Altera Max7128SLC84-15 PLD. På denne kreds laves en tristate adressebus med 17 ben, som skal bruges til at sende adressen på den sample som skal skrives til RAM'en. Bussen bruges også i forbindelse med kommandodekodereren, se 16.8.2. Der implementeres en 16-bit input-databus, som kun bruges til læsning i forbindelse med kommandodekodereren. Herudover laves der buskontrolben, herunder ben til busdeling, ben til Chip-Select til RAM samt et inputben til Chip-Select af controlleren.

ADC'en er forbundet til databussen gennem to eksterne 3-state buffere, da ADC'en er konfigureret på en måde, hvor den ellers under hele sampleforløbet ville have brugt bussen. Inden buffer'ene sker der desuden en konvertering af sampledata fra 10 bit uden fortegn til 16 bit med fortegn. Dette skere ved at invertere den mest betydende bit (den 10. bit) af sampledataen, og sende denne værdi til outputbuffernes 6 mest betydende bits.

Begrundelsen for at denne konvertering er mulig ligger i den af MCU'en benyttede talrepresentation, som er 2's komplement.

16.7 Display DMA

16.7.1 Opdeling

Denne del af controlleren skal sørge for dataudlæsningen. Vi har, ligesom ved ADC-delen, valgt en fleksibel løsning, hvor der fra MCU'en kan bestemmes en række faktorer. Displayet kan tændes og slukkes, da dette ikke kræver ekstra HW (skal alligevel ske efter RESET). Baseadressen på displaydata i system-RAM'en er gjort konfigurerbar. Derudover er der valgt en opdeling af displayet i 8 linier, således at udlæsning af tekst på displayet er optimeret. Optimeringen består i at man fra MCU'en med en bitmaske kan bestemme hvilke linier der skal opdateres.
Controlleren kopierer altid $128 \cdot 64 = 8192 bits = 1024 bytes$, og efter en kopiering sendes et interrupt til MCU'en.
Dataudlæsning udføres i to trin. Først hentes et der data fra hukommelsen ved DMA. Dernæst sker der en udlæsning af data til displayet via en seperat bus.

16.7.2 Intern displaystyring

Display'ets udskrivningscyklus aktiveres med en kommando fra MCU'en, som indeholder informationer om hvilke linier, der skal opdateres og hvilken baseadresse data ligger på. Når kommando'en er blevet sendt, går en større state-machine i gang med at udføre DMA-kopieringen til displayet gennem et forløb med flere løkker. Denne state-machine ses på figur 16.11.
State-machinen "kalder" andre state-machines, som sørger for mindre opgaver, som f.eks. DMA-læsning af data fra RAM, læsning af status fra display og skrivning af data til display.
Displayets opbygning gør en del løkker nødvendige. Displayet har følgende data:

- To lige store displayhalvdele på 64 x 64 pixels, som vælges med hver sin chipselect.
- Hver halvdel består af 8 linier med 64 x 8 pixels i hver linie.
- Der skrives 8 pixels i een byte ad gangen.

- Hver gang der skrives et 1x8-pixels søjle forøges skrivepositionen til søjlen til højre for

- Der er specielle instruktioner til at sætte skrivepositionen på displayet.

I det følgende beskrives display-DMA sekvensen.
Når kommandoen kaldes ,der får display-DMA'en til at opdatere display'et ud fra de betingelser, der er angivet i parameteren til kommandoen, gennemløber state-machinen bitmasken, der indeholder de linier, der skal opdateres. Når første linie er identificeret, checker display-DMA'en om display'et er busy. Dette gøres hver gang en kommando er sendt til display'et. Denne funktion udføres efter princippet vist i figur 16.14. Når det er konstateret, at display'et ikke er busy sættes x-adressen på. Den angiver hvilen linie i display'et, man ønsker at tilgå. Denne kommando er betinget af hvilken linie, der aktuelt opdateres. Efter x-adressen er specificeret og display'et ikke længere er busy, sættes den y-adresse, der skal skrives i. Første gang y-adressen sættes er det længst til venstre i venstre display-halvdel(asserteret $\overline{CS1}$). Da linien automatisk fyldes op er det kun nødvendigt at sætte y-adressen, når der skiftes linie eller display-halvdel. Når display'et melder ikke-busy, beregnes adressen til det sted i RAM'en, hvor data'ene til udskrivning findes. Adressen beregnes ud fra baseadressen + linie-offset'et + pladsen i linien. Denne beregning sker i state 6. Efter endt beregning læses data'ene fra RAM'en. Dette sker efter principperne i afsnit 16.7.3. I forbindelse med læsning af data'ene, state 7, læses der 16 bit. Det er det dobbelte af displaybussen, og dermed indeholder en RAM læsning to gange display skrivning.
De første 8 bit, der skrives i display'et, er de 8 mest betydende bit. Dette sker efter princippet vist i figur **??**. Når display'et har behandlet de modtagne data, og bekræftet det ved at melde ikke-busy, skal den anden del af de, fra RAM'en, læste data skrives ud. Der er her tale om de 8 mindst betydende bit. Display'et sørger selv for at skifte til næste række pixel i linien. State-machinen befinder sig nu i state 12. I denne state sker checket af følgende betingelser:

- Hvis y-adresse er 31, og dermed befinder sig i sidste række pixel i en linie, checkes der om den er i venstre dislay-halvdel. Er det tilfældet skiftes der til den højre display-halvdel. Udskrivningen genoptages fra state 2. Derefter udskrives der efter ovenstående princip. State 2 til 12 gennemløbes, med y-adressen initialiseret til 0.

- Hvis y-adressen er 31 og er i højre display-halvdel, er linien færdig udskrevet. Bittet, svarende til den udskrevne linie, i opdaterings-bitmasken maskes ud så linien ikke opdateres flere gange. Udskrivningen fortsætter i state 1, hvor der checkes om der er flere liner, der skal udskrives.

- Er ingen af de to ovenstående betingelser opfyldt er en linie under udskrivning. Y-adressen inkrementers med 1. Y-adressen bruges i ovenstående betingelser og i beregning af adresse til læsning fra RAM'en.

Når alle linier er udskrevet interuptes MCU'en. Det bruger MCU'en til at aktivere udskrivning med en ny baseadresse, og en opdateringsbitmaske. State-machinen venter i state 0 (Start i figur **??**)

16.7.3 Busovertagelse og Ram-læsning

Busovertagelsen i displaydelen foregår efter præcis samme principper som i ADC-delen. Displaydelen er blot placeret længere nede i prioritetskæden.
Busovertagelsessekvensen og RAM-læsningssekvensen er (ligesom i ADC-delen) placeret i een stor state-machine, da disse altid skal udføres lige efter hinanden. På figur 16.12 og 16.13 ses timingsdiagrammet og timingsværdierne for sekvensen.
Læsningen fra RAM'en foregår efter samme princip som skrivning til RAM'en, og startes og afsluttes i samme states som ved skrivning til RAM'en, dvs state 3 til state 8 (se. figur 16.9).

16.7.4 Kommunikation med display

Displayets skrive/læse-cyklus er i modsætning til ved de tidligere enheder synchron, hvilket vil sige at alt skal times i forhold til et bestemt clocksignal. Displayet kræver et clocksignal med en minimumsperiode på 1000 ns, som genereres i PLD ved at neddividere 32MHz-clock'en med 34. Da displayet har en forholdsvis langsom clock, er en meget præcis skrive/læse-cyklus ikke nødvendig, hvilket betyder at vi kan nøjes med forholdsvis korte state-machines til formålet. Figurerne 16.14 og 16.15 viser timingsdiagrammer for hhv. læse- og skrivecyklus.
På figur 16.16 ses timingsværdierne for displayet.

16.7.5 Hardwareopbygning

DisplayDMA-delen er blevet implementeret i en seperat Altera-kreds. Dette skyldes dog udelukkende logik-pladsmangel internt i den første PLD, da antallet af nødvendige ben var blevet fordelt således, at begge DMA-dele kunne være i een chip.
Display-DMA PLD'en er forbundet til adressebussen vha en tristatebus. Databussen forbundet i en læsekonfiguration med en bredde på 16-bit. Der er lavet en seperat displaybus med en databredde på 8-bit, da denne bus bruges næsten konstant pga "BUSY-poll'ning. Herudover er PLD'en forbundet til busdelingssignaler, RAM-chipselectsignaler og et PLD-chipselet signalet.

16.8 Andre moduler

16.8.1 Filter clock

Implemtationen af clocken til det Switched-Capacitor filter, som er beskrevet i afsnit 15.3.1, sker i en PLD. Dette skyldes at clocken, ud over at anvendes til filteret, også anvendes til timing af andre dele af samplingen, herunder ADC timing. Clocken genereres ved at dele en extern clock, her implementeret i form af en X-tal oscillator, med et antal gange svarende til den ønskede clock.
Systemet er implementeret, så der kan vælges imellem 8 frekvensområder. Dette giver en større frihedsgrad til indførelse af forskellige funktioner til systemet, f.eks zoom funktion.
Den valgte X-tal oscillator er på 7.3728 MHz, den er valgt på baggrund af sortimanget og ønsket om en omkring 20 kHz. I figur 16.17 fremgår den 8 valgte frekovensområder.
I den analoge del af målejournalen afsnit D.4 er testen af antialiaseringsfilteret og dermed implecit en test af implementationen af filter clock generator i PLD'en.
Med udganspunkt i filter clocken genereres ADC timings clock i form af samplefrekvensen. For at generer samplefrekvensen divideres filter clocken med 19. Samplefrekvensen passer forholdsmæssig med den øvre grænsefrekvens, i alle frekvensområder, med en faktor > 2, så der undgåes aliasering (Nyquist-frekvensen overholdes).

16.8.2 Kommando-dekoder

Pld'ens funktionalitet kommer til udtryk i en række kommandoer. For at eksekvere en kommando gennemløbes følgende sekvens, se figur 16.18.
MCU mapper først adressen, på kommadoen, ud på adressebussen og derefter parameteren ud på databussen. Når både adressebussen og databussen er stabil, asserterer MCU'en $\overline{CS}$ i 100nS [16, s. A-13] ved 0 waitstates. MCU'en fastholder databussen i min. 15 nS, t_{SNDOI} på figur 16.18. Da PLD'en bruger max 5nS,t_{XZ} [27, s. 28] , på latche at parameteren , fra databussen, er det i orden at trigge latchingen på $\overline{CS}$'s opadgående flanke. Det er i overensstemmelse med den tid MCU holder databussen stabil.
Instruktionssættet til PLD'erne findes i afsnit 16.9.

16.8.3 Systemclock

For at øge ydelsen på MCU'en skal en duty cycle på 50% tilstræbes. Dette skyldes, som fremgår af afsnit 14.3.2, krav fra MCU'en. For at generere en duty cycle på 50% halveres en frekvens på 32 MHz til 16 MHz. I det tilfælde den 32 MHz clock har en duty cycle, der afviger fra 50%, vil en halvering af clocken med en Flip flop give en duty cycle på 50%, da flip flop'en altid trigger på samme flanke. Dette princip anvendes i systemet.

16.9 Controller kommandosæt

Den programerbare logik indeholder en række funktioner, som stilles til rådighed for MCU'en. For at give MCU'en mulighed for at styre funktonaliteten af den programerbare logik (Altera kredsene) er der implementeret et kommandosæt. Kommandosættet består af 5 kommandoer til styring af samplingen og displayet. Hver kommando skal angives med en parameter. Denne parameter angiver kommandoens handlingsforløb. Funktionaliteten er fordelt på to kredse, men set fra MCU'ens vil de optræde som een. For at aktivere kommandodekoderen skal MCU'en assertere alteraenes chip-select ben. Under aktiveringen af kommandofortolkeren dekodes adressebussen for hvilken kommando, der skal udføres. Når kommandoen er bestemt, dekodes databussen for parameteren til kommanoden.

16.9.1 Sæt sample base adresse og start sampling

Kommandoen har til formål at bestemme base adressen for de samplede målinger. Efter bestemmelse af base adressen startes samplingen.
Samplingen vil stoppe efter at have fyldt en buffer med målinger. Kommandoen skal gentages for hver gang, der skal samples.

16.9.2 Sæt divider for samplebuffer

Kommandoen har til formål at bestemme divideren for samplebufferen.
Divideren bestemmes ud fra nedestående formel.

$$Divider = \frac{2048(maxbuffersize)}{antalpunkterFFT} \tag{16.1}$$

For at være en gyldig værdi skal divideren være af størrelsen 2^n, hvor $0 \leq n \leq 7$.

16.9.3 Sæt frekvensområdet

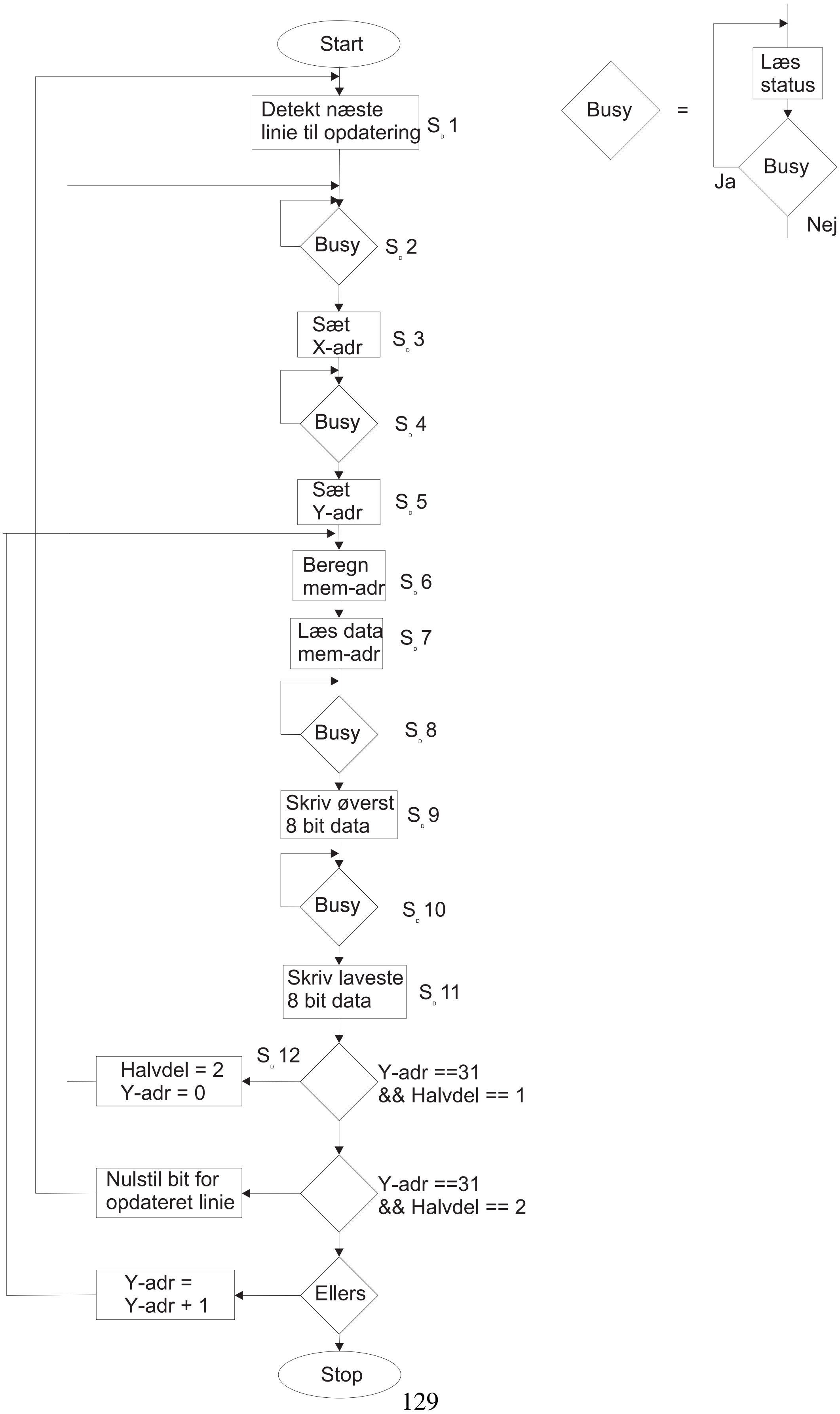
Kommandoen har til formål at bestemme frekvensområdet. Områdenr vælges ud fra nedestående tabel.

16.9.4 Sæt opdatering af display

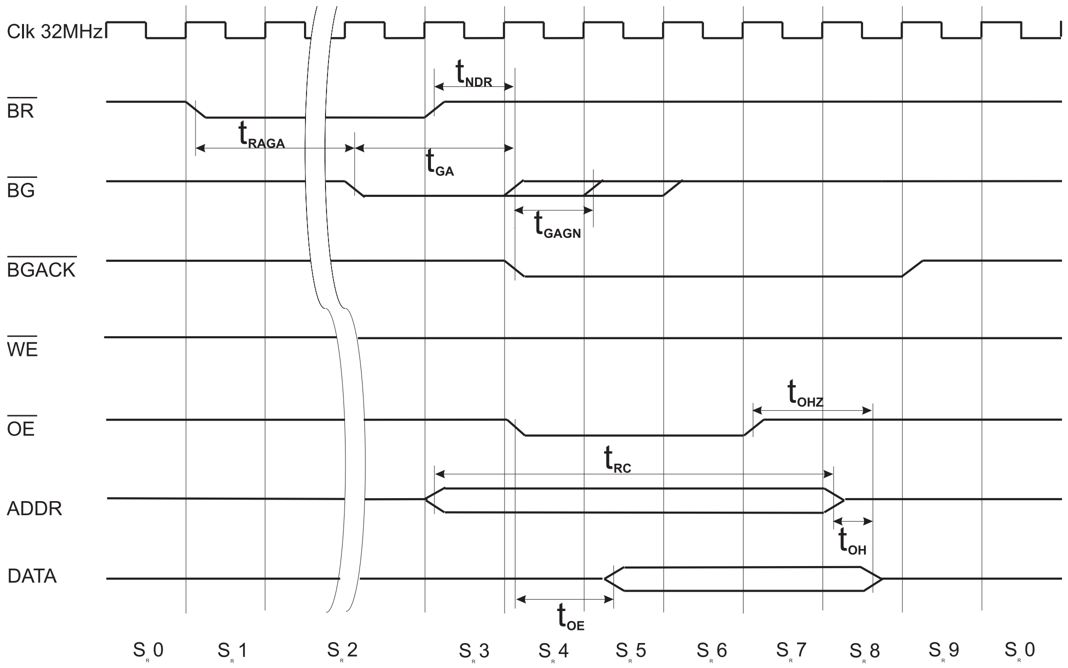
Kommandoen har til formål at bestemme baseadressen for data, samt hvilke linier, der skal udskrives på displayet.
For at opdatere en linie skal den aktuelle bit asserteres med '1'.

16.9.5 Tænd/sluk for displayet

Kommandoen har til formål at tænde eller slukke displayet. Kommandoen virker på begge halvdele af displayet.



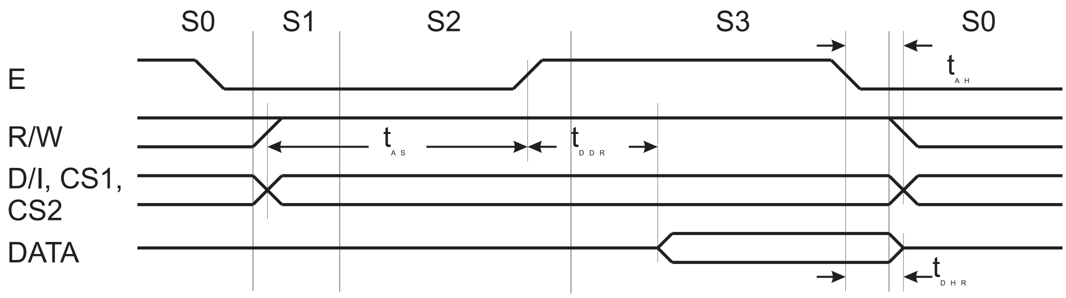
Figur 16.11: Rutediagram over udskrivningsprocessen for displayet.



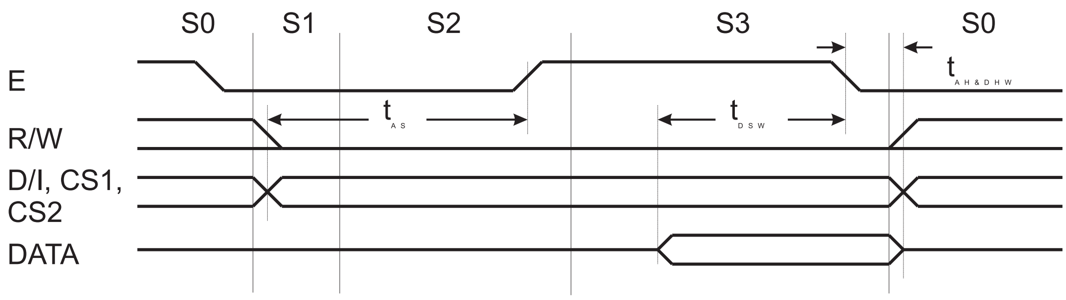
Figur 16.12: Display-DMA timing.

Betegnelse	Enhed	Beskrivelse	Værdi
t_{NDR}	MCU (krav)	$\overline{BR}$ skal negeres inden MCU'en negerer $\overline{BG}$	min 0 ns
t_{GA}	MCU	$\overline{BG}$ bredde asserteret	min 1 $t_{cyc} = 59,6ns$
t_{BRAGA}	MCU	$\overline{BR}$ asserteret til $\overline{BG}$ asserteret	min 1 $t_{cyc} = 59,6ns$
t_{GAGN}	MCU	$\overline{BGACK}$ asserteret til $\overline{BG}$ asserteret	min 1 $t_{cyc} = 59,6ns$ max 2 $t_{cyc} = 119.2ns$
t_{OHZ}	RAM	Output Enable Output Floating	max 30 ns
t_{RC}	RAM (krav)	Read Cycle Time	min 70 ns
t_{OH}	RAM	Output Hold Time	min 10 ns
t_{OE}	RAM	Output Enable Access Time	max 40 ns

Figur 16.13: Display-DMA timingsværdier.



Figur 16.14: Timingsdiagram for læsning fra display.



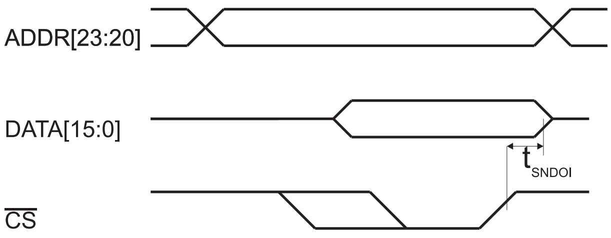
Figur 16.15: Timingsdiagram for skrivning til display.

Betegnelse	Beskrivelse	Værdi
t_{AS}	Address Setup Time	min 140 *ns*
t_{DDR}	Data Delay Time	max 320 *ns*
t_{DHR}	Data Hold Time (Read)	min 20 *ns*
t_{DSW}	Data Setup Time	min 200 *ns*
t_{DHW}	Data Hold Time (Write)	min 10 *ns*
t_{AH}	Address Hold Time	min 10 *ns*

Figur 16.16: Display-DMA timingsværdier.

Frekvensområde	Filter frekvensen	Øvre grænsefrekvens	Stopbånds frekvensen	division af X-tal oscillatoren
0	1,84 MHz	18,43 kHz	22.1 kHz	4
1	1,29 MHz	12,29 kHz	14,75 kHz	6
2	922 kHz	9,22 kHz	11,06 kHz	8
3	737 kHz	7,37 kHz	8,85 kHz	10
4	614 kHz	6,14 kHz	7,37 kHz	12
5	527 kHz	5,27 kHz	6.32 kHz	14
6	461 kHz	4,61 kHz	5,53 kHz	16
7	419 kHz	4,10 kHz	4,92 kHz	18

Figur 16.17: Tabel over de 8 frekvensområder.



Figur 16.18: Timing af kommandokald i pld.

Bitmaske på adresse bussen
Altera baseadresse + 2

Figur 16.19: Kald af kommandoen.

Bitmaske på databussen					
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bits 10 - 0
adr16	adr15	adr14	adr13	adr12	ikke i brug

Figur 16.20: Parameteren.

Bitmaske på adresse bussen
Altera baseadresse + 4

Figur 16.21: Kald af kommandoen.

Bitmaske på databussen	
Bits 15 - 10	Bits 9 - 0
ikke i brug	divideren

Figur 16.22: Parameteren.

Områdenr	Frekvensområde
0	20 Hz - 22,1 kHz
1	20 Hz - 14,7 kHz
2	20 Hz - 11,1 kHz
3	20 Hz - 8,8 kHz
4	20 Hz - 7,4 kHz
5	20 Hz - 6,3 kHz
6	20 Hz - 5,5 kHz
7	20 Hz - 4,9 kHz

Figur 16.23: Valg af parameter.

Bitmaske på adresse bussen
Altera baseadresse + 8

Figur 16.24: Kald af kommandoen.

Bitmaske på databussen	
Bits 15 - 3	Bits 2 - 0
ikke i brug	Områdenr

Figur 16.25: Parameteren

Bitmaske på adresse bussen
Altera baseadresse + 16

Figur 16.26: Kald af kommandoen.

Bitmaske på databussen														
Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bits 9 - 8	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
display base adr							Linier der skal opdateres							
adr16	adr15	adr14	adr13	adr12	adr11	ikke i b	l7	l6	l5	l4	l3	l2	l1	l0

Figur 16.27: Parameteren.

Bitmaske på adresse bussen
Altera baseadresse + 32

Figur 16.28: Kald af kommandoen.

Kapitel 17

Udvidelsesmuligheder

17.1 Indledning

Dette kapitel omhandler udvidelsesmulighederne for det fremstillede system. At de ikke er realiseret skal ses som et udtryk for afgrænsning af projektet. En del af disse muligheder, som vi fra starten har set i systemet, er omtalt i kravspecifikationen, og afgrænsningen er foretaget ved en inddeling i obligatoriske og optionelle krav samt fremtidige udvidelsesmuligheder , andre er mere løsningspecifikke, tekniske, såsom anvendelsen af vinduesfunktioner i forbindelse med DFT'en.

17.2 Kravspecifikationen

17.2.1 Optionelle krav

Disse krav er overvejet i designet af systemet i henhold til kravspecifikationen:

- Brugergrænseflade
- Logaritmisk frekvensskala
- Sample/buffer viewer

Alle kræver forholdsvis små ændringer og/eller tilføjelser, hovedsageligt i software. Kun brugergrænseflade, der jo kunne være en skrueknap eller lignende, kræver hardwaretilføjelse. Brugergrænsefladen skulle sætte brugeren i stand til at skifte mellem de forskellige visninger (lineære eller logaritmiske akser, zoom) og samplefrekvensen og opløsning.

Sample/buffer vieweren, der skal vise indholdet af bufferen, dvs. inputsekvensen, er umiddelbart til at implementere. Dette kræver en SW-permutation af inputdata, da inputsekvensen er ombyttet og "pakket" af DMA-controlleren. Desuden kræves en skalering.
Den logaritimiske frekvensskala kræver større ændringer: Dels skal en logaritmetabel genereres, dels skal antallet af punkter i DFT'en tilpasses, da denne frekvensligt foretages lineært. Dette er tilfældet, hvis frekvenskomposanterne skal optræde med fast interval i visningen).

17.3 Fremtidige udvidelsesmuligheder

En række fremtidige udvidelsesmuligheder er specificeret i kravspecifikationen:

- RS232/Parallelt interface
- 10" B/W display
- PC-software til videre behandling/præsentation af resultater
- Grafisk præsentation af Amplitude- og fasekarakteristik

Disse kræver alle, relativt til de optionelle krav, omfattende tilføjelser/ændringer.
10" displayet (her er tale om et specifikt display, vi kunne låne) er komplekst og kræver omfattende interfacing.
Faciliteter til implementation af et RS232-interface er inkluderet i mikrocontrolleren (QSM), hvorimod et parallelt interface til PC vil kræve yderligere hardware. Disse ville kunne anvendes til præsentation af frekvensanalysen i f.eks. MATLAB.
PC-software forudsætter eksistensen af et RS232 eller parallelt interface og en kravspecifikation. Softwaren kunne så inkludere port-drivere afhængigt af OS. og dele til visning, evt. udprintning, og analyse. Omfanget af en sådan udvidelse vil afhænge af kravspecifikationen.
Amplitudespektret kræver en implementation af en kvadratrod i det nuværende system, da amplituden af de enkelte frekvenskomposanter er lig kvadratroden af power spektret. Dette kan gøres på utallige måder, f.eks ved lineær approksimation, Newton-Raphson-metoden, tabeller evt. med lineær interpolering, hvilket MCU'en understøtter (se TBLS [14, s. 7-10]).

En visning af fasekarakteristik i det nuværende system vil være uanvendelig grundet fasekarakteristikken af antialiaseringsfilteret (elliptisk eller Chebyshev). Eventuelt kunne et udtryk for denne opstilles og kompensering foretages.
Rent praktisk vil beregningen af fasekarakteristikken kræve en arcustanges-funktion - evt. i form af en tabel.

17.4 Windowing

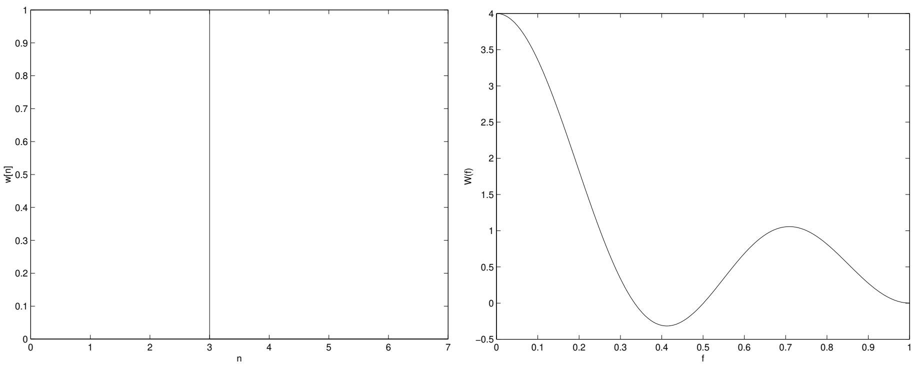
17.4.1 Leakage

Når en DFT udføres, foretages den på et endeligt udsnit af en i princippet uendeligt sekvens. Dette svarer til, at indgangssekvensen, $x[n]$ multipliceres med en rektangulær vinduesfunktion. I frekvensdomænet er dette lig foldningen mellem indgangssekvensens og vinduessekvensens frekvensspektrum.
Hvis det rektangulære vindue har sekvensen:

$$w[n] = 1 \qquad \text{for } -M \leq n \leq M \tag{17.1}$$

Har det rektangulære vindue har følgende frekvensspektrum [8, s. 460]:

$$W(f) = \frac{sin 2\pi f M T}{\pi f} \tag{17.2}$$



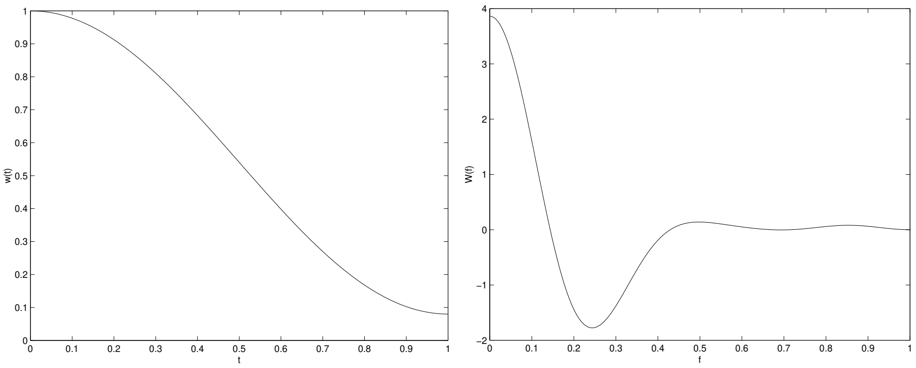
Figur 17.1: Den rektangulære vinduesfunktion, $w(t)$, og dennes frekvensspektrum, $W(f)$.

Denne "ringning", som ses på figur 17.1, er også kendt som Gibbs fænomen. Årsagen til dette er at finde i diskontinuiteten ved overgange fra 1 til 0 (og omvendt) i sekvensen. I

DFT'en manifesterer dette sig som en forvrængning af frekvensspektret, beskrevet som "udsivning", heraf "leakage", af energien bort fra de oprindelige frekvenskomposanter og danner nye falske.
Dette må anses som den største mangel i det fremstillede system.

17.4.2 Vinduesfunktioner

Problemet med leakage kan reduceres ved at anvende en vinduesfunktion. Disse multipliceres med indgangsekvensen, $x[n]$, og sikrer, at den resulterende sekvens varierer jævnt.



Figur 17.2: Eksempel på vinduesfunktion, Hamming vindue.

Figur 17.2 viser Hammingvinduesfunktionen. Af andre vinduesfunktioner kan nævnes Hanning, Kaiser, Bartlett, Boxcar og Blackman. Disse har hver deres fordele og ulemper.

17.5 Andre

Analyse og optimering med hensyn til kvantiseringsstøj ved trunkering og afrunding i softwaren er et emne, vi gerne ville have gjort mere ud af. Specielt har vi savnet dette i forbindelse med optimerede anvendelse af FFT'en (transformationen af reel funktion af den dobbelte længde), og generelt har vi savnet værtøj til belysning (og løsning) af problemstillingerne.

Kapitel 18

Studierapport

18.1 Indledning

Følgende kapitel indeholder en studierapport, der har til formål at dokumentere projekt relaterede emner som overvejelser vedrørende projektvalg, opgavefordeling i gruppen, planlægning af projektforløbet, erfaringer med gruppearbejde og samarbejde med vejleder, samt udbytte af PE-kurser. Da der ikke foreligger overordnede krav til at studierapporten skal indgå i den samlede rapport, laves den udelukkende på projektgruppens eget initiativ, idet den giver mulighed for at reflektere over projektforløbet og samle erfaringer.

18.2 Projektvalg

Da alle medlemmer i projektgruppen har arbejdet sammen i tidligere projekter, har vi opnået et vist kendskab til hinandens arbejdsmetoder, samt erfaring i hvorledes projektarbejdet bør struktureres, for at udnytte gruppens ressourcer. Projektgruppen har i tidligere projekter haft en svaghed, hvad angår afgrænsning af projektets omfang, hvorfor der var stor enighed om, at dette projekt fra starten skulle være så afgrænset og veldefineret som muligt.

18.2.1 Projektforeslag

Temaet for dette 4.semesters projektet "Maskinnær programmering" har affødt 4 projektforeslag.

- Autonom styringsenhed
- Frekvensanalysator
- Adgangs-, overvågnings- og alarmsystem
- EKG Monitor

Efter gennemlæsning af forslagene og mundtlig oplæg af foreslagsstillerne, blev gruppen hurtigt enig om at projektforslaget med titlen "Frekvensanalysator" passede fint til gruppens ønsker, idet dette projekt var tilpas afgrænset og umiddelbart kunne danne grundlag for en kravsspecifikation; i modsætning til de øvrige projektforslag, hvor det først var nødvendigt at udarbejde en mere eller mindre omfattende systembeskrivelse. Ydermere var der en vis interesse for emnet, idet hovedparten af gruppen på 2. semester har arbejdet med digital filtrering i forbindelse med udvikling af en digital audio mixer med equalizering på PC.

18.3 Opgavefordeling

For at gøre projektarbejdet mere effektivt, besluttede vi ret tidligt i projektforløbet (d. 3/3-99) at dele gruppearbejdet op i tre undergrupper med følgende opgaver:

Gruppe 1

- FFT-algoritmen på PC i C
- Div. managers på PC
- Div. managers på mikro
- FFT på mikro
- Optimere FFT i asembler

Gruppe 2

- Minimumsystem: Mikro, RAM, ROM, mem-map/CS, LCD
- DMA-controller + ADC-styring (digital)

Gruppe 3

- Filtre
- Signaltilpasning
- ADC (analog)

Pga. denne opdeling bestemte vi desuden følgende: Vi skulle have koordineringsmøder hver fredag, så alle gruppemedlemmerne kunne følge med i, hvor langt vi var nået med de forskellige dele. Endelige arbejdsblade skulle laves når en hel opgave var afsluttet. Vi regnede med at grupperne ville "smelte sammen" sidst i arbejdsprocessen.
En sådan opdeling af gruppen kan have følgende konsekvenser:

- Fordel: Projektet kan blive dybt og samtidig bredt
- Ulempe: Ikke alle i gruppen kan få samme kendskab til alle dele af projektet

Vi føler i gruppen, at vi i dette projekt alle har haft meget at lave. Tiden er ikke blevet spildt med store diskussioner mellem alle gruppemedlemmerne, som i nogle tidligere projekter. Undergrupperne har selv foretaget de fleste beslutninger. Kun vigtige beslutninger med konsekvenser for resten af projektet er blevet taget op på fælles møder. Alt dette har resulteret i en langt mere effektiv arbejdsproces, som har gjort projektet både bredt og dybt.
Dette kan dog resultere i, at gruppemedlemmerne ikke alle kender alle hjørner af projektet lige godt. Dette har vi forsøgt at forbedre ved at lave gruppemøder og interne fremlæggelser af gruppernes arbejde/arbejdsblade. Sidst i projektet begyndte grupperne desuden at arbejde mere på tværs om opgaverne.
Hvad kunne gøres bedre:

- Fordelingen af grupperne var skæv på den måde, at der i starten af projektet var for få folk på arbejdet med grundsystemet i HW.
- Arbejdsblade skulle have blevet udarbejdet tidligere
- Arbejdsblade kunne have været fremlagt tidligere

18.4 Planlægning

I forbindelse med opdelingen af gruppen i undergrupper, blev der lavet en overordnet tidsplan.
Formålet med den overordnede tidsplan var dels at have et redskab til at planlægge en række milestones, samt at koordinere indsatsen i de forskellige undergrupper, herunder fordeling af ressourcer. Detaljeringen af den overordnede tidsplan blev overladt til undergrupperne.
Fællesmøder blev holdt med jævne mellemrum, både med og uden vejleder, hvor status i undergrupper blev rapporteret til resten af gruppen.
Milestones blev brugt til at overveje status i forhold til tidsplanen, og afvigelser blev konstateret. På et tidspunkt blev afvigelserne på visse områder så store at en korrektion var nødvendig. En sådan korrektion var mulig, fordi den oprindelige tidsplan havde indbygget elasticitet, med henblik på at tage højde for uforudsete forsinkelser.

18.5 Gruppearbejde

Internt i gruppen har vi benyttet følgende til at koordinere og informere hinanden om arbejdet i gruppen:

- Pligtfordeling
- Mødeteknik
- Arbejdsblade

I det følgende er beskrevet, hvorledes vi har effektueret disse punkter.

18.5.1 Pligtfordeling

I gruppen uddelte vi fra starten de organisatoriske pligter til de enkelte gruppemedlemmer. Mødekoordinatoren har haft til opgave at styre både interne møder i gruppen og eksterne møder med vejleder. Dette blev gjort ud fra en dagsorden, som er blevet mailet til de implicerede senest dagen før. Al kommunikation med vejleder har gået igennem mødekoordinatoren for at undgå multiple modstridende aftaler.

Referanten har skullet tage referat af møderne i en detaljegrad, så alle vigtige oplysninger, valg og aftaler kan gendannes. Disse referater er blevet indskrevet af referanten og givet til webmasteren.
Webmasteren har løbende haft til opgave at opdatere gruppens hjemmeside med de sidste nye informationer, så de andre i gruppen løbende har kunnet holde sig orienteret om de forskellige dele af projektet på internettet.
Datakoordinatoren har haft til opgave at holde orden i gruppedir'et, hvor alle gruppens filer har ligget. Det er således datakoordinatorens opgave at rydde op i gamle filer og sørge for en ordentlig struktur i gruppedir'et.
Papirkoordinatorens opgave har været at sørge for at al udefrakommende post er blevet kopieret og fordelt til alle gruppens medlemmer. Endvidere har han haft til opgave at informere de andre i gruppen om disse.
Den hardwareansvarlige har haft til opgave at finde ud af, hvilke komponenter som var til rådighed. Desuden har han indsamlet information om de anvendte komponenter.
Den softwareansvarlige har haft til opgave at sørge for at den nødvendige software var til rådighed f.eks. i forbindelse med rapportskrivningen. Der er således blevet lavet CVS og makefiles, som har lettet arbejdet i forbindelse med rapportskrivningen.
De organisatoriske pligter er af alle gruppens medlemmer blevet udført ansvarligt. I enkelte tilfælde har der dog været små forsinkelser i udførelsen af pligterne. De enkelte medlemmer har taget mere ansvar end tidligere.

18.5.2 Mødeteknik

I forbindelse med møderne har der været en bestemt arbejdsgang.
Møderne er blevet ledet "løst" af mødekoordinatoren. Senest dagen før mødet har alle gruppens medlemmer modtaget en mail fra mødekoordinatoren med dagsordenen for mødet. Herefter var det op til de enkelte medlemmer at læse den relevante information på gruppens hjemmeside (Referat fra tidligere møde, ny dokumentation osv.) eller de udfærdigede arbejdsblade.

18.5.3 Arbejdsblade

Under udviklingen af de forskellige dele af projektet har de implicerede lavet arbejdsblade om emnerne. Disse har været været til rådighed til læsning på gruppens hjemmeside. I enkelte tilfælde er der foretaget præsentation af relevante dele af projectet som supplement til arbejdsbladene.

De interne forpligtigelser i forbindelse med projektet, møderne og udfærdigelse af arbejdsbladene har fungeret godt. Der har dog været forsinkelse i udfærdigelse af arbejdsbladene som følge af problemer med udviklingen af hardwaren.

18.6 Samarbejde med vejleder

Et af succeskriterierne, vi har opstillet for dette projekt, har været tidlig og ordentlig afgrænsning. Med et forholdsvis afgrænset projektforslaget har Ole Olsen allerede her hjulpet os godt på vej.
I tidligere projekter har vi stort set selv styret projektet og formuleret kravene - Disse har selvfølgelig været der, omend uformulerede. Denne gang har vi fået modspil i form af formulerede krav fra vejlederen, f.eks. angående det teoretiske, matematiske indhold (Fourier) i projektet - Et krav, der har vist sig ganske udbytterigt for de enkelte medlemmer af projektgruppen.
Endvidere har vi haft stort individuelt udbytte af fremlæggelse af de enkelte undergruppers arbejde og arbejdsblade undervejs i projektet, hvilket også er sket på opfordring fra vejleder.
Vi har i dette projekt anvendt vores vejleder i langt højere grad, end vi tidligere har gjort, og vejlederen har også udvist større engagement, end vi har været vant til.

18.7 Projektenhedskurser

I projektforløbet har vi deltaget i en række kurser med det formål at støtte os i projektarbejdet. Dels kurser i form af projektenhedskurser, der udgør projekt-pensum og evalueres igennem projektevalueringen, og andre kurser, SE- og VF-kurser.

18.7.1 Analog og digital elektronik

Formålet med dette kursus lyder [2]:

> "At give de studerende så meget kendskab til analog og digital elektronik, at de kan konstruere og realisere enkle, digitale systemer og interface dem til såvel en digital som en analog omverden."

Udbyttet af dette kursus har generelt været stort. Indholdet var relevant og anvendeligt, ikke bare for dette projekt. Kurset indholdt gennemgang af anvendte teknologier, eksempelvis CMOS og TTL, og metoder til fremstilling af kombinatorisk såvel som sekventiel logik.

18.7.2 Systemarkitektur og -integration

Dette kursus har følgende formål [2]:

1. "At give kendskab til datamaters opbygning og funktion."
2. "At give indblik i de programmelsystemer. der skal være til stede, for at en given datamat kan tilbyde omgivelser for afvikling af brugerprogrammel."
3. "At kunne udvikle struktureret programmel til håndtering af en centralenheds ydre enheder."

De generelle emner, der blev behandlet i dette kursus var ganske udmærkede, men havde karakter af baggrundsviden - og havde herfor ikke direkte betydning for projektet.
Mere konkrete emner, såsom software-delen, var ikke særlig specifik.
Kurset har udbyttemæssigt været præget af, at det tilsyneladende henvender sig til 4. semester på E-sektoren. Her anvendes et færdigt 68000-board, hvortil en VMEbus-enhed udvikles. De praktiske problemer med at få et minimumssystem (mikrocontroller, RAM og ROM) op at køre behandles overhovedet ikke. Vi har afset forholdvist mange ressourcer (2 mand i 2/3 af projektperioden) til dette. Desuden har BDM-interfacet voldt store problemer og blev aldrig helt stabilt - på E4 anvendes indbyggede faciliteter i 68000-boardet, PEPbug.

18.7.3 Andre kurser

Af andre kurser, vi har deltaget i, som har vist sig nyttige i projektarbejdet, er de valgfrie kurser i C-programmering og Programmerbar logik og SE-kurset Styresystemer og parallellitet.

Softwaren er hovedsageligt udviklet i C, og kurset i Programmerbar logik, der specifikt omhandlede Max+PlusII og Altera-kredse, fandt anvendelse i forbindelse med DMA-dataopsamlingen og -skrivningen til displayet.
Principper fra SE-kurset, Styresystemer og parallellitet, viste sig at være værdifulde i forbindelse med konstruktionen af en mikrokerne og organiseringen af programafvikling.

18.8 Konklusion

Når vi skal vurdere vore erfaringer med dette projekt er vi nødt til at se tilbage på vores tidligere erfaringer. For flertallet af gruppemedlemmernes vedkommende er det vort 4. semester sammen. Vi har derfor mulighed for at vurdere vores fremskridt på baggrund af et længere udviklingsforløb.
Tidligere har vi haft for vane at gemme vores arbejdsindsats til sidst i projektforløbet. Desuden har en stor del af projektarbejdet hvilet på enkeltpersoners præstationer, hvilket ind imellem har medført en nervepirrende afslutning på vores projekter. Før vort D3-projekt var vi opmærksomme på ulemperne ved denne arbejdsform, men vi formåede ikke at føre vores intentioner om et mere struktureret projektforløb ud i livet. Dette skyldtes ikke kun manglende vilje, men havde ligeledes sin årsag i vanskeligheder med den anvendte metode. Resultatet var endnu en nervepirrende afslutning.
Dette semester startede med en åbenhjertig diskussion, hvor vi tog vores arbejdsform grundigt op til overvejelse. Der blev talt direkte, og hvor det var nødvendigt blev enkeltpersoners indsats på forskellige områder diskuteret.
Resultatet af diskussionen blev en mundtlig aftale som omfattede en række punkter.

- Alle gruppemedlemmer forpligtede sig til at arbejde kontinuerligt gennem hele semesteret.
- Enkelte gruppemedlemmer forpligtede sig til at gøre en ekstra indsats, tidligt i semesteret, på områder, hvor deres kundskaber var mindre end det, der kræves på dette niveau.
- Vi ville tilstræbe en opdeling af projektgruppen i mindre enheder for at opnå en bedre udnyttelse af gruppens ressourcer.

Denne aftale er blevet overholdt. Resultatet er blevet det mest tilfredsstillende projektforløb vi endnu har oplevet. Det skyldes både ændringen af vores arbejdsform og indholdet

af kurserne, samt det projektforslag som vi valgte.
Opdelingen af gruppen i mindre enheder har bevirket, at vort projekt har fået både bredde og dybde. I vore individuelle valg af problemområder, har vi som hovedregel valgt områder, der gav udfordringer.
Vi kan opsummere med at konkludere, at dette har været et projekt, hvor vi har haft mulighed for at gøre fremskridt, som enkeltpersoner og som gruppe.

Kapitel 19

Konklusion

Temaet for 4. semester datateknik er maskinnær programmering og følgende formål er formuleret i studieordningen:

> "At sætte de studerende i stand til på maskinnært niveau at bygge og programmere dele til/af et mikrodatamatsystem [2]."

Dette må siges at være opfyldt igennem arbejdet med MC68331'eren og Altera PLD'erne, men også andre emner er berørt, såsom analog elektronik, parallellitets- og sandtidsproblematikker, opbygningen af operativsystemer og matematisk teori. Projektet har fagligt favnet meget bredt.

Fast Fourier Transformationen er blevet gennemarbejdet rent teoretisk og implementeret i C og optimeret i CPU32 assembler.

En systemtest er i skrivende stund endnu ikke blevet gennemført, og opfyldelsen af realtidskravet er herfor ikke eftervist men forventes opfyldt. Systemet er gennemtestet på PC.

At systemtesten ikke er gennemført skyldes dels det faktum, at vi grundet projektets omfang er kommet i tidsbeknep og dels praktiske problemer med BDM-interfacet, der kraftigt har hæmmet udviklingsforløbet.

Litteratur

[1] *Elektronik ståbi*. Teknisk Forlag A/S, 1995.

[2] Studieordning. 1999. http://www.kom.auc.dk/ESN/.

[3] ATMEL. *AT27C010/L*, 1994.

[4] Stephen Biering-Sørensen et al. *Håndbog i Struktureret Program Udvikling*. Teknisk Forlag A/S, 1. edition, 1996.

[5] William H. Press et al. *Numerical Recipes in C*. Cambridge University Press, 1992.

[6] GNU Project. *GNU C Compiler*, 1998.

[7] Per Brinch Hansen. The fast fourier transform. 1991.

[8] Erik Hüche. *Digital Signalbehandling*. Teknisk Forlag A/S, 1. edition, 1996.

[9] Erwin Kreyszig. *Advanced Engineering Mathematics*. John Wiley and sons, Inc., 1993.

[10] Maxim Integrated Products. *8th-Order, Lowpass, Switched-Capacitor Filters*, 1992.

[11] Giovanni De Micheli and Mariagiovanna Sami. *Hardware/Software Co-Design*. Kluwer Academic Publishers, 1996.

[12] Jacob Millman and Arvin Grabel. *Microelectronics*. McGraw-Hill, 2nd edition, 1987.

[13] Motorola. *CPU32 reference manual*, 1990.

[14] Motorola. *Programmer Reference Manual*, 1992.

[15] Motorola. *32-Bit Modular Microcontroller*, 1993.

[16] Motorola. *MC68331 User's Manual*, 1993.

[17] Motorola. *An Introduction to the MC68331 and MC68332*, 1996.

[18] National Semiconductor. *LF411A/LF411 Low Offset, Low Drift JFET Input Operational Amplifier*.

[19] National Semiconductor. *ADC1061 10-Bit High-Speed µP-Compatible A/D Converter with Track/Hold Function*, 1995.

[20] National Semiconductor. *Improving A/D Converter Performance Using Dither*, 1995.

[21] Christian Nentwich. Prometheus truecolour for x. 1999. http://www.cs.ucl.ac.uk/students/c.nentwich/ptc/.

[22] Ole Olsen. Projektforslag d4/99. 1999. http://www.kom.auc.dk/~oo/d4/99/fft-analyse.html 27. Maj 1999.

[23] Alan V. Oppenheim and Ronald W. Schafer. *Discrete-Time Signal Processing*. Prentice-Hall, 1989.

[24] Adel S. Sedra and Kenneth C. Smith. *Microelectronic Circuits*. Oxford University Press, 4th edition, 1998.

[25] SEIKO EPSON. *SRM20100LC 70/85/10*, 1994.

[26] Seiko Instruments. *Application Notes*, 1999. http://www.seiko-usa-ecd.com/lcd/pdf/graphic/appnotes.pdf.

[27] Seiko Instruments. *MAX 7000A Programmable Logic Device Family*, 1999. http://www.altera.com/document/ds/m7000a.pdf.

[28] Seiko Instruments. *Modules with Build-In Data RAM*, 1999. http://www.seiko-usa-ecd.com/lcd/pdf/graphic/builtinram.pdf.

[29] Texas Instruments. *LINEAR AND INTERFACE APPLICATIONS*, 1985.

[30] Texas Instruments. *Linear And Interface Circuits*, 1985.

[31] Texas Instruments. *High-Speed CMOS Logik*, 1986.

[32] Texas Instruments. *SUPPLY VOLTAGE SUPERVISORS*, 1991.

Appendiks A

Fast Fourier Transformation

A.1 Fourierrækker

En periodisk funktion kan skrives som en fourierrække,

$$f(t) = a_0 + \sum_{m=1}^{\infty} a_m \cdot cos(m\omega t) + b_m \cdot sin(m\omega t) \tag{A.1}$$

[9, s. 569], hvor $m\omega$ er den m'te harmoniske af grundfrekvensen, ω. Koefficienterne a_m og b_m er reelle tal og kaldes Fourier-koefficienter. Disse findes ved hjælp af Eulers formler,

$$a_0 = \frac{1}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)dt \tag{A.2}$$

$$a_m = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\cdot\cos(m\omega t)dt, \qquad m = \{0, 1, 2, 3...\}, \tag{A.3}$$

$$b_m = \frac{2}{T}\int_{-\frac{T}{2}}^{\frac{T}{2}} f(t)\cdot\sin(m\omega t)dt, \qquad m = \{0, 1, 2, 3...\} \tag{A.4}$$

[9, s. 571], hvor T er perioden.
Bruger man den eksponentielle notationsform for cosinus og sinus,

$$\cos(\omega t) = \frac{1}{2}(e^{j\omega t} + e^{-j\omega t}), \tag{A.5}$$

$$\sin(\omega t) = \frac{1}{2j}(e^{j\omega t} - e^{-j\omega t}), \tag{A.6}$$

kan f(t) også skrives som

$$f(t) = \sum_{m=0}^{\infty} \frac{a_m}{2}(e^{jm\omega t} + e^{-jm\omega t}) + \frac{b_m}{2j}(e^{jm\omega t} - e^{-jm\omega t}) \tag{A.7}$$

$$= \sum_{m=0}^{\infty} \frac{a_m - jb_m}{2} \cdot e^{jm\omega t} + \frac{a_m + jb_m}{2} \cdot e^{-jm\omega t} \tag{A.8}$$

$$= \sum_{m=0}^{\infty} c_m \cdot e^{jm\omega t} + c_{(-m)} \cdot e^{-jm\omega t}, \qquad c_m = \frac{a_m - jb_m}{2} \tag{A.9}$$

$$= \sum_{-\infty}^{\infty} c_m \cdot e^{jm\omega t}. \tag{A.10}$$

De komplekse Fourier-koefficienter kan nu beregnes som

$$c_m = \frac{a_m - jb_m}{2} \tag{A.11}$$

$$= \frac{1}{2} \cdot \frac{2}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot (\cos(m\omega t) - j\sin(m\omega t))dt \tag{A.12}$$

$$= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot (\frac{1}{2}(e^{j\omega t} + e^{-j\omega t}) - j \cdot \frac{1}{2j}(e^{j\omega t} - e^{-j\omega t}))dt \tag{A.13}$$

$$= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot (\frac{1}{2}e^{j\omega t} + \frac{1}{2}e^{-j\omega t} - \frac{1}{2}e^{j\omega t} + \frac{1}{2}e^{-j\omega t})dt \tag{A.14}$$

$$= \frac{1}{T} \int_{-\frac{T}{2}}^{\frac{T}{2}} f(t) \cdot e^{-j\omega t} dt. \tag{A.15}$$

A.2 Fourier-transformationen

Visse tids-diskrete sekvenser kan repræsenteres ved et Fourier-integrale med formen

$$x[n] = \frac{1}{2\pi} \int_{-\pi}^{\pi} X(e^{j\omega}) e^{j\omega n} d\omega, \tag{A.16}$$

hvor $X(e^{j\omega})$ er givet ved

$$X(e^{j\omega}) = \sum_{n=-\infty}^{\infty} x[n] e^{-j\omega n}, \tag{A.17}$$

[23, s. 45]. Ligningerne A.16 og A.17 udgør tilsammen en Fourier-repræsentation af sekvensen, $x[n]$. Ligning A.16, den inverse Fourier-transformation, er en syntese-formel, der repræsenterer $x[n]$ som en superposition af uendeligt små komplekse sinus-svingninger af formen

$$\frac{1}{2\pi}X(e^{j\omega})e^{j\omega n}d\omega, \tag{A.18}$$

hvor ω spænder over et interval med længden 2π, og hvor $X(e^{j\omega})$ angiver det relative indhold af hver kompleks sinus-komposant. Ligning A.17, Fourier-transformationen, udtrykker hvorledes $X(e^{j\omega})$ beregnes udfra sekvensen $x[n]$, og anvendes til at analysere sekvensen $x[n]$ for at bestemme hvor meget af hver frekvens-komposant, der kræves for at sammensætte $x[n]$ udfra ligning A.16.

A.3 DFT

For tids-diskrete sekvenser med endelig længde eksisterer der en alternativ Fourier-repræsentation. Denne kan opfattes som samples af $X(e^{j\omega})$ ved frekvenserne $\omega_k = 2\pi k/N$, hvor N angiver længden af en sekvens, $x[n]$ og $0 \leq k \leq N-1$. Ligesom den tids-kontinuerte Fourier-repræsentation har den diskrete Fourier-repræsentation både en syntese- og analyse-ligning:

$$\text{Analyse-ligning:} \qquad X[k] = \sum_{n=0}^{N-1} x[n]W(N)^{kn}, \tag{A.19}$$

$$\text{Syntese-ligning:} \qquad x[n] = \frac{1}{N}\sum_{n=0}^{N-1} X[k]W(N)^{-kn}. \tag{A.20}$$

$W(N) = -e^{2\pi/N}$ [23, s. 532]. I det tilfælde hvor $X[k] = 0$ for alle værdier af k, der ikke ligger i intervallet $0 \leq k \leq N-1$, og $x[n] = 0$ for alle værdier af n, der ikke ligger i intervallet $0 \leq k \leq N-1$, kaldes A.19 og A.20 for henholdsvis den diskrete Fourier-transformation (DFT) og den inverse diskrete Fourier-transformation (invers DFT). Det er værd at bemærke at skaleringen af den inverse DFT med $\frac{1}{N}$, kan absorberes i definitionen af $X[k]$, hvorved vi får følgende sammenhæng:

$$X[n] = Nx[-k] \tag{A.21}$$

[23, s. 539]

A.4 Kompleksitet

Antager vi at inputtet til en DFT er en sekvens af komplekse værdier, $x[n]$, vil det kræve N komplekse multiplikationer og $N-1$ komplekse additioner, at beregne hver værdi af

DFT'en, hvis vi bruger ligning (A.19) som formel til beregningen. Hver kompleks multiplikation kræver fire reelle multiplikationer og to additioner, og hver kompleks addition kræver to reelle additioner. Til beregningen af $X[k]$ skal der udføres $4N$ reelle multiplikationer og $(4N-2)$ reelle additioner. Da $X[k]$ skal beregnes for N forskellige værdier af k, vil den direkte beregning af den diskrete Fourier-transformation af sekvensen $x[n]$ kræve $4N^2$ reelle multiplikationer og $N(4N-2)$ reelle additioner. Foretages denne beregning på en digital computer skal der endvidere påregnes processortid til at lagre og tilgå de N komplekse input-værdier og værdierne af de komplekse koefficienter $W(N)^{nk}$.
Da antallet af beregninger og dermed også beregningstiden med god tilnærmelse kan siges at være proportional med N^2 bliver beregningen af DFT'en ved den direkte metode meget stor ved store værdier af N. Dette kan motivere en interesse i at finde metoder til beregning, der reducerer antallet af multiplikationer og additioner.
Der kan opnåes stor effektivitet ved at opløse beregningen af DFT'en i en række mindre DFT-beregninger. Man kan i denne forbindelse udnytte symmetrien og periodiciteten i den komplekse exponentialfunktion $W(N)^{nk} = e^{-j(2\pi/N)nk}$ [23, s. 583]. En klasse af algoritmer, der på effektiv vis udnytter denne mulighed er Fast Fourier-transformationen (FFT).

A.5 Fast Fourier Transformation

A.5.1 DIT-algoritmer

DIT-algoritmer (decimation-in-time algorithms) er en klasse af FFT-algoritmer der, ved hjælp af del-og-kombiner teknikken, opløser en sekvens af samples i en række mindre sekvenser [23, s. 587]. Dette princip kan anvendes til en effektiv beregning af DFT'en og kan nemmest illustreres ved at betragte tilfælde hvor N er en heltals-potens af 2, d.v.s. $N = 2^v$. Vi kan da beregne $X[k]$ ved at adskille $x[n]$ i to $(N/2)$-punkts sekvenser, der hver består af henholdsvis de lige og de ulige punkter i $x[n]$, og (A.19) kan omskrives til

$$X[k] = \sum_{n \text{ lige}} x[n] \cdot W(N)^{nk} + \sum_{n \text{ ulige}} x[n] \cdot W(N)^{nk}. \tag{A.22}$$

Ved at substituere $n = 2r$ for lige værdier af n og $n = 2r+1$ for ulige værdier af n fås

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] \cdot W(N)^{(2r)k} + \sum_{r=0}^{(N/2)-1} x[2r+1] \cdot W(N)^{(2r+1)k} \tag{A.23}$$
$$= \sum_{r=0}^{(N/2)-1} x[2r] \cdot (W(N)^2)^{rk} + W(N)^k \sum_{r=0}^{(N/2)-1} x[2r+1] \cdot (W(N)^2)^{rk}$$

Men $W(N)^2 = W(N/2)$ da

$$W(N)^2 = e^{-2j(2\pi/N)} = e^{-j2\pi/(N/2)} = W(N/2)$$

og (A.23) kan følgelig omskrives til

$$X[k] = \sum_{r=0}^{(N/2)-1} x[2r] \cdot W(N/2)^{rk} + W(N)^k \sum_{r=0}^{(N/2)-1} x[2r+1] \cdot W(N/2)^{rk} \tag{A.24}$$

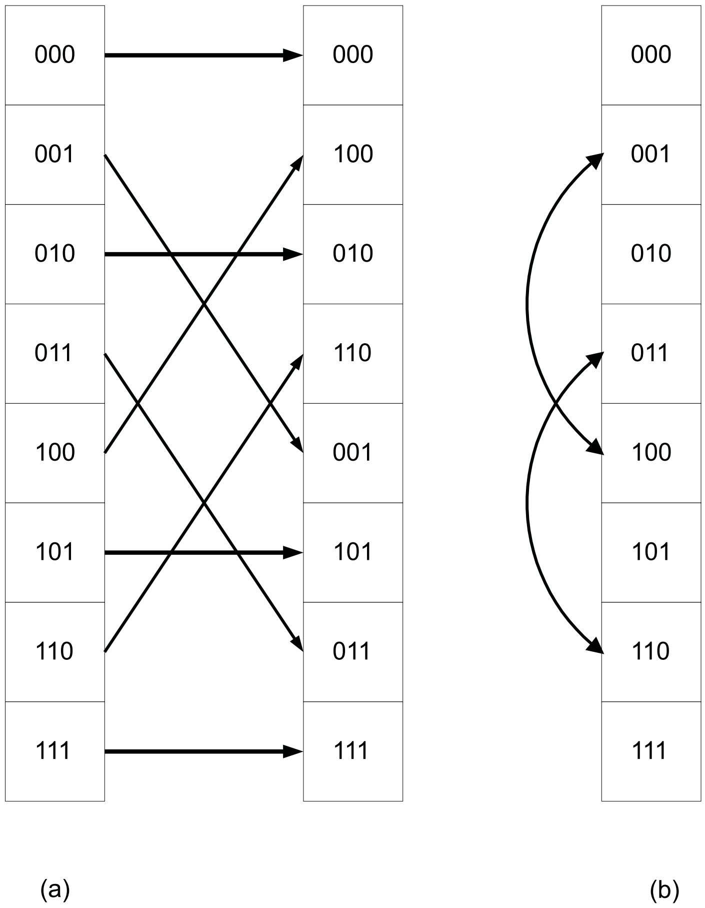
Da N er en heltals-potens af 2 kan denne procedure gentages for hver af de to $(N/2)$-punkts DFT'er, således at vi får fire $(N/4)$-punkts DFT'er og disse kan igen deles. Denne proces kan udføres ialt $log_2(N)$ gange.

A.5.2 Bitreversering

En simpel måde at lave denne opsplitning på er at foretage en indledende bit-reversering af rækken af input-samples. Dette virker fordi opsplitningen svarer til at teste input-vektoren på mindst betydende bit. Er denne 0, er der tale om et lige nummer i rækken af samples, og disse grupperes i øverste halvdel af listen. Er den 1, er det et ulige nummer, og disse grupperes i nederste halvdel. Dette gentages i hver af de to halvdele for næstmindst betydende bit og så fremdeles [5, s. 505-506], [23, s. 594-596]. Figur A.1 viser, hvorledes bitreverseringen foregår i et array med længden 8.

A.5.3 Butterfly

DFT'en foretages først på de enkelte punkter, men da den Fourier-transformerede af et enkelt punkt er punktet selv kan denne beregning udelades [5, s. 505]. Det næste skridt er at kombinere tilstødende punkter to og to, og foretage en to-punkts DFT på hver af disse. Den grundlæggende to-punkts DFT beregning (butterfly) for en radix-2 DIT algoritme har formen:



Figur A.1: Et array med længden 8 rearrangeres ved en bitreversering, (a) mellem to arrays, (b) in place. Bitreversering er en nødvendig del af FFT-algoritmen.

$$\begin{aligned} X_m[p] &= X_{m-1}[p] + W(N)^r \cdot X_{m-1}[q], \\ X_m[q] &= X_{m-1}[p] + W(N)^{r+N/2} \cdot X_{m-1}[q] \end{aligned} \tag{A.25}$$

[23, s. 594]. Udnyttes det at

$$W(N)^{N/2} = e^{-j2\pi/N)N/2} = e^{-j\pi} = -1,$$

kan faktoren $W(N)^{r+N/2}$ omskrives til

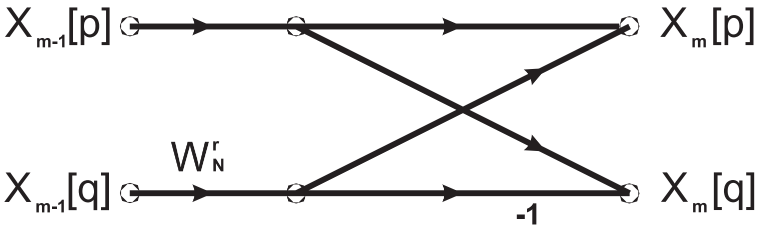
$$W(N)^{r+N/2} = W(N)^{N/2} \cdot W(N)^r = -W(N)^r.$$

(A.25) kan derfor også udtrykkes som

$$X_m[p] = X_{m-1}[p] + W(N)^r \cdot X_{m-1}[q], \tag{A.26}$$

$$X_m[q] = X_{m-1}[p] - W(N)^r \cdot X_{m-1}[q]. \tag{A.27}$$

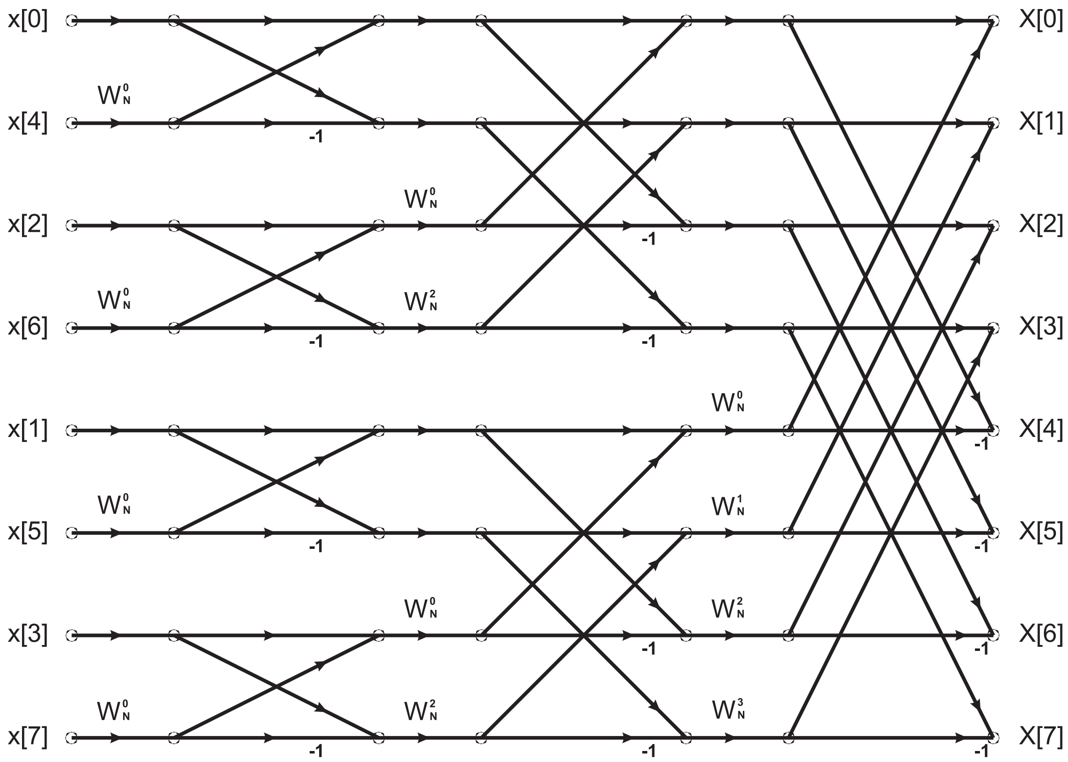
m og $(m-1)$ refererer til henholdsvis den nuværende og forrige vektor i beregningen i et bestemt stadie. $m = 0$ refererer til input-vektoren og $m = log_2(N)$ refererer til output-vektoren. p og q udtrykker placeringen af de enkelte elementer i vektorerne. I hvert stadie foretages $N/2$ beregninger af denne type for at generere den næste vektor. Heltallet r varierer med p, q og m på en måde, der afhænger af den specifikke FFT algoritme, der anvendes.
Figur A.2 viser butterfly beregningen i sin simplificerede udgave.



Figur A.2: Flow graf for den grundlæggende buttefly beregning.

A.5.4 Kombinering

De $N/2$ to-punkts DFT'er kombineres dernæst to og to til $N/4$ fire-punkts DFT'er, der igen kombineres til $N/8$ otte-punkts DFT'er. Denne proces, der ligeledes har $log_2(N)$ stadier, fortsætter indtil to $N/2$-punkts DFT'er kombineres til en N-punkts DFT.
I hver af de $log_2(N)$ stadier genereres der altså, ud fra en eksisterende vektor med længden N, en ny vektor med længden N, ved hjælp af lineære kombinationer af parrede elementer i de to vektorer. Den sidst beregnede vektor, i det sidste stadie, indeholder den ønskede DFT.
Flow-grafen i figur A.3 beskriver ovenstående algoritme eksemplificeret ved en 8-punkts DFT. Kompleksiteten af algorithmen er $O(\frac{n}{2}log_2(n))$.



Figur A.3: Flow graf for en 8-punkts DFT

Appendiks B

Antialiaseringsfilter

B.1 Krav

Et antialiaseringsfilter er et lavpasfilter, der har til formål at fjerne frekvenser, der kan forårsage aliasering. Disse er er frekvenser over den halve samplefrekvens, Nyquistfrekvensen.
Filteret skal anvendes til et 10 bit system, og et rimeligt krav ville være, at signalet bør dæmpes under A/D converterens opløsningevne i stopbåndsfrekvensen, svarende til en $60,2dB$ dæmpning - Aliaseringsfejl vil dog altid forekomme, da stopbåndsdæmpningen er endelig. Stopbåndsfrekvensen, ω_s, der er lig den halve samplefrekvens, skal være $22050Hz$. Den normerede stopbåndsfrekvens, der angiver forholdet mellem afskærings- (ω_a) og stopbåndsfrekvenserne, accepteres som $1,5$. Ripple kan minimalt holdes til $0,1dB$ i et 10. ordens filter uden at kompromittere andre krav. Dette er fundet ved iteration af $N = cheb1ord(1, 1.5, 0.1, 60.2, 's')$ i MATLAB.

B.2 Filtertype

Chebyshev type I [24, s. 897] er et All-pole filter, hvilket vil sige, at det ingen nulpunkter har. Filteret har rimelig flankestejlhed ved en given orden (sammenlignet med f.eks. Butterworth- eller Besselfiltre), dog forekommer ripple i pasbåndet.
Størrelsen ε er givet ved passbåndsripplen, A_{max}:

$$\varepsilon = \sqrt{10^{A_{max}/10} - 1} \tag{B.1}$$

Stopbåndsdæmpningen er givet ved:

$$A(\omega_s) = 10\log[1 + \varepsilon^2 \cosh^2(N \cosh^{-1}(\omega_s/\omega_a)] \tag{B.2}$$

Ordenen, N, der er nødvendig for at opnå en given stopbåndsdæmpning, kan da findes ved iteration af dette udtryk.
Den k'te komplekse pol kan findes ud fra følgende [24, s. 733]:

$$\begin{aligned} p_k = &-\omega_a \sin(\frac{2k-1}{N}\frac{\pi}{2})\sinh(\frac{1}{N}\sinh^{-1}\frac{1}{\varepsilon}) \\ &+ j\omega_a \cos(\frac{2k-1}{N}\frac{\pi}{2})\cosh(\frac{1}{N}\sinh^{-1}\frac{1}{\varepsilon}), \quad \text{hvor } k = 1, 2, ..., N \end{aligned} \tag{B.3}$$

Endelig har Chebyshevfilteret en overføringsfunktion, der ser således ud:

$$H(s) = \frac{A_{DC}\omega_a^N}{\varepsilon 2^{N-1}(s-p_1)(s-p_2)\cdots(s-p_N)} \tag{B.4}$$

Hvor A_{DC} er DC-forstærkningen.

B.3 Poler

De komplekse poler kan udfra kravene i B.1 findes i MATLAB vha. kommandoen $[Z, P, K] = cheb1ap(10, 0.1)$. Dette giver disse kompleks-konjugerede poler:

$$p_1, p_10 = \omega_a(-0,0408 \pm j1,0207) \tag{B.5}$$

$$p_2, p_9 = \omega_a(-0,1184 \pm j0,9208) \tag{B.6}$$

$$p_3, p_8 = \omega_a(-0,1844 \pm j0,7307) \tag{B.7}$$

$$p_4, p_7 = \omega_a(-0,2323 \pm j0,4692) \tag{B.8}$$

$$p_5, p_6 = \omega_a(-0,2575 \pm j0,1617) \tag{B.9}$$

Det ses, at alle polerne er negative, dvs. de ligger i venstre halvplan i s-domænet. Denne egenskab udgør stabilitetskravet til polerne.
Den returnerede K er svarende til $\frac{1}{\varepsilon 2^{N-1}}$ i B.4. Denne er $K = 0.0128$.

MATLAB returnerer de normerede værdier, svarende til $\omega_a = 1$. Vinkelfrekvensen bliver:

$$\omega_a = \frac{\omega_s}{1,5} = \frac{22050Hz \cdot 2\pi}{1,5} = 92,36k\frac{rad}{s} \tag{B.10}$$

Med afskæringsfrekvensen ganget på bliver polerne:

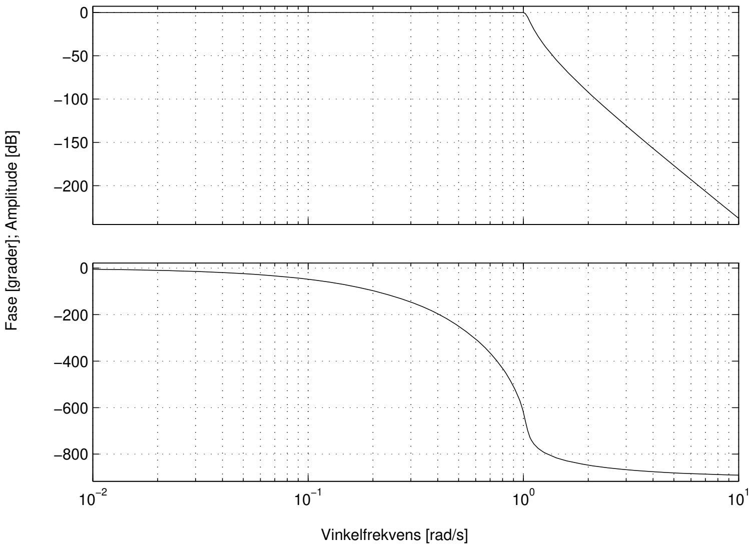
$$p_1, p_10 = (-0,3767 \pm j9,4276) \cdot 10^4 \tag{B.11}$$

$$p_2, p_9 = (-1,0933 \pm j8,5047) \cdot 10^4 \tag{B.12}$$

$$p_3, p_8 = (-1,7029 \pm j6,7494) \cdot 10^4 \tag{B.13}$$

$$p_4, p_7 = (-2,1458 \pm j4,3334) \cdot 10^4 \tag{B.14}$$

$$p_5, p_6 = (-2,3786 \pm j1,4932) \cdot 10^4 \tag{B.15}$$



Figur B.1: Amplitude- og fasekarakteristik for normeret 10. ordens Chebyshevfilter med $0,1dB$ ripple, normeret stopbåndsfrekvens $1,5$ og $60,2dB$ stopbåndsdæmpning.

Figur B.1 viser amplitude- og fasekarakteristikkerne for det normerede filter.
ε bliver, beregnet ud fra pasbåndsripplen:

$$\varepsilon = \sqrt{10^{0,1/10} - 1} = 0,1526 \tag{B.16}$$

B.4 2. ordenssektioner

Pol-parrene realiseres hver for sig i 2. ordenssektioner, også kendt som biqauds.
Foretages opdelingen i 2. ordenssektioner i form af komplekskonjugerede pol-par, ser nævneren i hver 2. ordenssektion, hvis polen $p_i = \alpha + j\beta$, således ud:

$$s^2 + s(-2\alpha) + (\alpha^2 + \beta^2) \tag{B.17}$$

Dvs. koefficienterne bliver reelle. Hver 2. ordenssektion er karakteriseret ved en polgodhed, Q, der angiver selektiviteten af sektionen, og polresonansfrekvensen, ω_o [24, s. 905].

$$s^2 + s(-2\alpha) + (\alpha^2 + \beta^2) = s^2 + (\omega_o/Q)s + \omega_o^2 \tag{B.18}$$

Q og ω_o kan da beregnes som:

$$\omega_o = \sqrt{\alpha^2 + \beta^2} \qquad Q = \frac{\sqrt{\alpha^2 + \beta^2}}{-2\alpha} \tag{B.19}$$

Beregnes disse for alle pol-parrene fåes:

$$Q_1(p_1, p_{10}) = 12,5222 \qquad \omega_o 1(p_1, p_{10}) = 9,4351 \cdot 10^4 \tag{B.20}$$

$$Q_2(p_2, p_9) = 3,9214 \qquad \omega_o 2(p_2, p_9) = 8,5747 \cdot 10^4 \tag{B.21}$$

$$Q_3(p_3, p_8) = 2,0438 \qquad \omega_o 3(p_3, p_8) = 6,9609 \cdot 10^4 \tag{B.22}$$

$$Q_4(p_4, p_7) = 1,1268 \qquad \omega_o 4(p_4, p_7) = 4,8356 \cdot 10^4 \tag{B.23}$$

$$Q_5(p_5, p_6) = 0,5904 \qquad \omega_o 5(p_5, p_6) = 2,8085 \cdot 10^4 \tag{B.24}$$

Disse størrelser anvendes i realiseringen af de enkelte 2. ordenssektioner.
Tælleren i den samlede overføringsfunktion er:

$$\frac{A_{DC}\omega_a^N}{\varepsilon 2^{N-1}} \tag{B.25}$$

Fordeles denne på på hver af de 5 2. ordenssektioner fåes, at tælleren, hvis $A_{DC} = 1$, på den i'te sektion er:

$$a_i = \sqrt[N/2]{\frac{A_{DC}\omega_a^N}{\varepsilon 2^{N-1}}} = \sqrt[5]{\frac{1 \cdot (92363\,\frac{rad}{s})^{10}}{0,1526 \cdot 2^9}} = 3,5681 \cdot 10^9 \tag{B.26}$$

B.5 Overføringsfunktion

Ud fra ligning B.4 fåes filterets samlede overføringsfunktion, der ser således ud:

$$H(s) = \frac{3,5681 \cdot 10^9}{s^2 + s0,7535 \cdot 10^4 + 8,9021 \cdot 10^9} \tag{B.27}$$

$$\cdot \frac{3,5681 \cdot 10^9}{s^2 + s2,1867 \cdot 10^4 + 7,3526 \cdot 10^9} \tag{B.28}$$

$$\cdot \frac{3,5681 \cdot 10^9}{s^2 + s3,4058 \cdot 10^4 + 4,8454 \cdot 10^9} \tag{B.29}$$

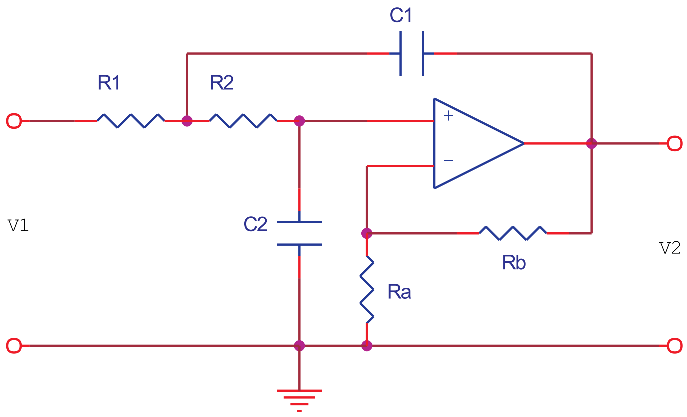
$$\cdot \frac{3,5681 \cdot 10^9}{s^2 + s4,2916 \cdot 10^4 + 2,3383 \cdot 10^9} \tag{B.30}$$

$$\cdot \frac{3,5681 \cdot 10^9}{s^2 + s4,7572 \cdot 10^4 + 0,7887 \cdot 10^9} \tag{B.31}$$

Disse 2. ordenssektioner benævnes $H_1(s), H_2(s), ..., H_5(s)$.

B.6 Realisering

Filteret realiseres i form af aktive-RC 2. ordens sektioner af typen Sallen & Key [12, s. 739]. Disse er Single-Amplifier Biquads (SAB), dvs. kun en enkelt operationsforstærker andvendes i modsætning til f.eks. Tow-Thomas 2. ordenssektioner, der anvender hele 3.



Figur B.2: Sallen & Key LP 2. ordenssektion.

Figur B.2 viser realiseringsmodellen for en 2. ordens lavpassektion.

At en ikke-inverterende kobling er anvendt skyldes, at den inverterende kun bør anvendes med $Q < 10$ [1, s. 222], hvorimod den ikke inverterende kan anvendes ved $Q < 50$. I den inverterende kobling kunne K være realiseret i forholdet $\frac{R_b}{R_a}$.
Vælges $C_1 = C_2 = C$ og $R_1 = R_2 = R$, gælder Følgende sammenhæng for polresonansfrekvesen i 2. ordenssektionen:

$$\omega_o = \frac{1}{RC} \tag{B.32}$$

Og for Q:

$$Q = \frac{1}{3 - A_v} \quad \Leftrightarrow \quad A_v = 3 - \frac{1}{Q} \tag{B.33}$$

Udgangspunktet for dimensioneringen er Q og ω_o for hver 2. ordenssektion, herefter vælges værdier af komponenterne C og R_a. $C = 0,5nF$ og $R_a = 4,87k\Omega$ (1%-række) vælges. Herudfra beregnes så de resterende komponenter.
R_b bestemmes ud fra DC-forstærkningen, A_v, og R_a, dvs. forstærkningen igennem den ikke inverterende kobling:

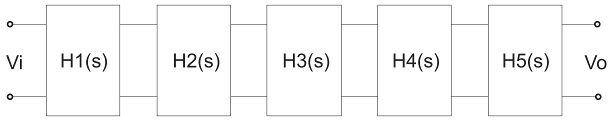
$$R_b = (A_v - 1)R_a \tag{B.34}$$

Figur B.3 viser valgte og beregnede komponentværdier for de enkelte 2. ordenssektioner.

Sektion	$C_1 = C_2 = C$	R_a	$R_1 = R_2 = R$	R_b
H_1	$0,5nF$	$4,87k\Omega$	$21,0k\Omega$	$9,31k\Omega$
H_2	$0,5nF$	$4,87k\Omega$	$23,2k\Omega$	$8,45k\Omega$
H_3	$0,5nF$	$4,87k\Omega$	$28,7k\Omega$	$7,32k\Omega$
H_4	$0,5nF$	$4,87k\Omega$	$41,2k\Omega$	$5,36k\Omega$
H_5	$0,5nF$	$4,87k\Omega$	$71,5k\Omega$	$1,50k\Omega$

Figur B.3: 2. ordenssektioner komponentværdier

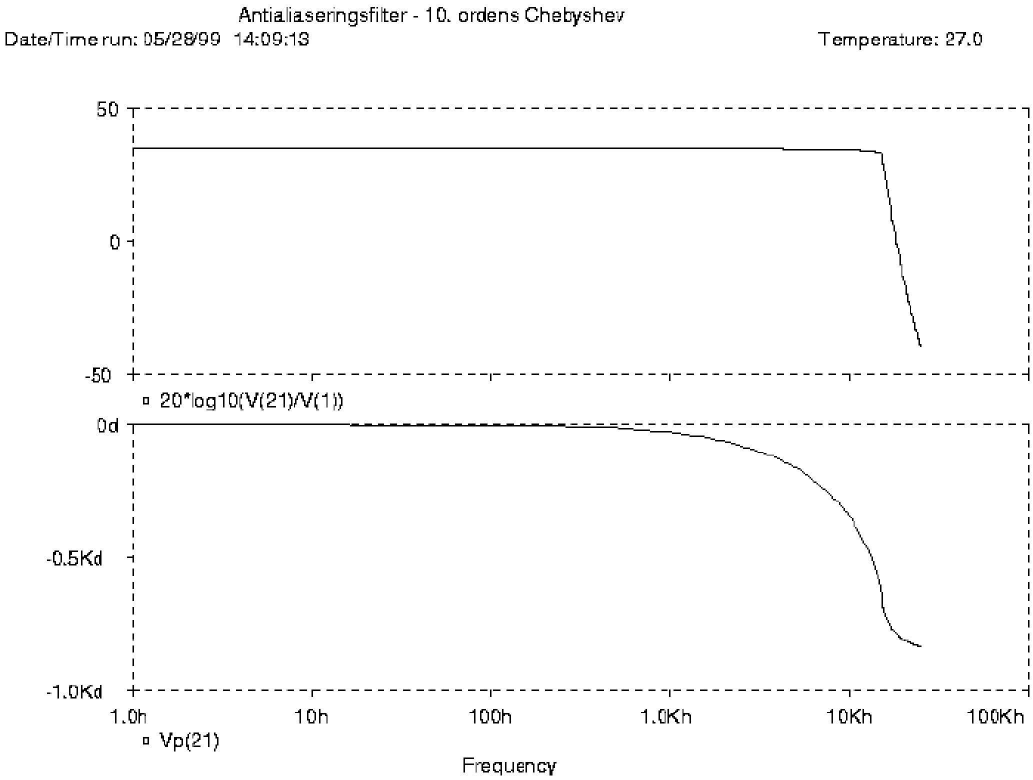
De 5 2. ordenssektioner kaskadekobles så i den endelige opstilling, som vist på figur B.4 - Output på en sektion tilsluttes input på den næste.



Figur B.4: Kaskadekobling af 2. ordenssektioner.

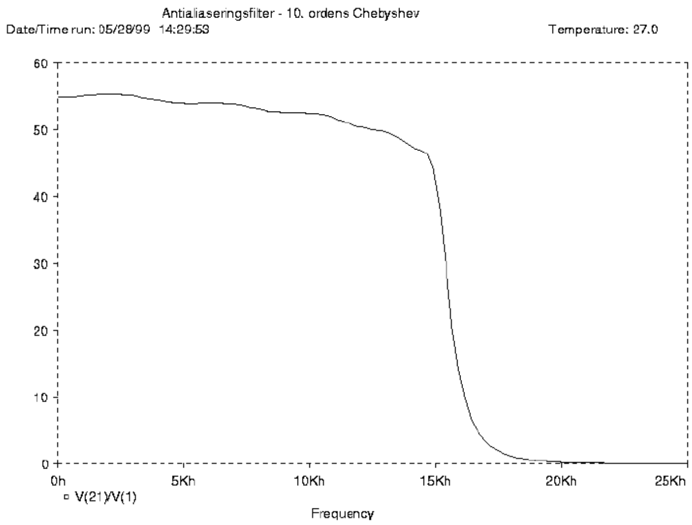
B.7 Simulering

Filteret er simuleret i SPICE. Simuleringen er foretaget med ideelle operationsforstærkere. Dette skyldes, at den tilgængelige version af PSPICE er en evalueringsversion, der kun tillader et vist antal symboler (100).



Figur B.5: Simulering af filter i SPICE - Fase- og amplitudekarakteristik.

Figur D.5 viser filterets fase -og amplitudekarakteristik. Bemærk, at der ikke er kompenseret for forstækningen igennem filteteret. Dette gøres ved faktoren K.
Figur B.6 viser filterets amplitudekarakteristik med lineære akser, hvilket gør aflæsning



Figur B.6: Simulering af filter i SPICE - Lineær amplitudekarakteristik

af afskærings- og stopbåndsfrekvenser nemmere.

B.8 Integration

For at integrere antialiaseringsfilteret i det allerede eksisterende kredsløb kræves en række ændringer. Filteret skal indsættes efter differensforstærkeren (U7) og signaltilpasningskoblingen (U2). Forstærkningen i denne kobling skal ændres til $2,5$.
Man kunne desuden spare en operationsforstærkerkobling ved at lægge kompenseringen for forstærkningen igennem filteret, $K = 0,0128$, i den inverterende kobling. Denne kan ikke realiseres i selve filteret, da der anvendes ikke-inverterende koblinger.
Forstærkningen skal altså være:

$$K \cdot |A| = \frac{R_6}{R_5} \tag{B.35}$$

Vælges $R_5 = 100k\Omega$, fåes:

$$R_6 = R_5 \cdot K \cdot |A| = 100k\Omega \cdot 0,0128 \cdot 2,5 = 3,2k\Omega \tag{B.36}$$

Den normerede stopbåndsfrekvens på 1.5 bevirker, at amplitudespektret under stopbåndsfrekvensen ikke vil modsvare indputsignalet. Ripple forvrænger også amplitudespektret. Frekvenskomposanterne nær stopbåndsfrekvensen vil være behæftet med de mest betydelige aliaseringsfejl, og disse bør bortskæres i visningen af resultatet.
Bemærk, at Switched Capacitor-filteret er anvendt i den endelige opstilling, grundet dets variable afskæringsfrekvens. Således er filteret, der her er beskrevet, ikke realiseret.

Appendiks C

Digital målejournal

Målejounalen for den digitale del af systemet indholder målinger på følgende:

- Spændingsforsyning
- Clockfrekvens
- Clock duty cycle
- LVI
- ROM
- RAM

Disse er beskrevet i det følgende.

C.1 Spændingsforsyning

På forsyningsspændingen måles de anvendte spændingsniveauer (5 V, 15 V og -15V) for at sikre at spændingsforsyningen leverer de nødvendinge spændinger. Endvidere måles udgangsspændingen på spændingsregulatoren til displayet (-8 V).
Målingen af spændingsniveauerne er foretaget med et multimeter (FLUKE 37 Multimeter LabNr. 08181) på udgangene af spaendingskilderne.
For at de i systemet anvendte enheder ikke kommer i en udefinerbar tilstand kræves det, at forsyningsspændingen på 5 V ikke afviger med mere end 10 %. De målte spændings niveauer skal dermed ligge mellem 4,5 V - 5,5 V. Displayet kræver en spændingsforsyning

på -8 V $\pm$ 0,3V. Der kræves ikke den samme præsicion for de andre forsyningsspændinger til operationsforstærkerne ($\pm$ 15 V).
De målte spændingsniveauer kan ses på figur C.1.

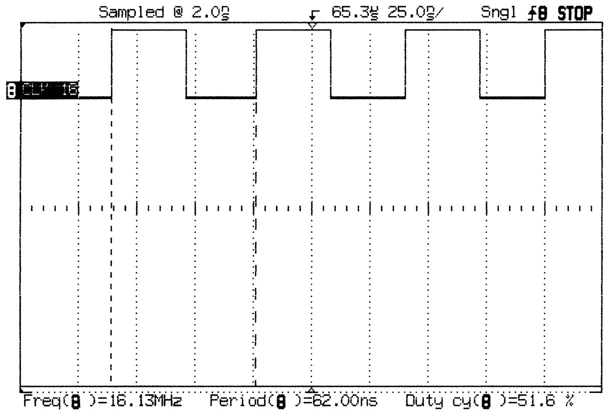
Forsyningsspænding	Målt spænding
-15 V	-15,36 V
-8 V	-8,02 V
5 V	5,03 V
15 V	15,48 V

Figur C.1: Målte spændingsniveauer fra forsyningspændingen

Det kan af målingerne konkluderes, at 5 V og 8 V spændingsforsyningerne ligger inden for det krævede område. Endvidere kan der konkluderes, at de øvrige forsyningsspændinger ligger inden for et acceptabelt niveau. Den samlede konklusion er således, at spændingsforsyningerne overholder de opstillede krav.

C.2 Clockfrekvens

For at sikre, at MCU'en drives med den rigtige clockfrekvens måles den i Altera 7128'erens neddividerede clock.
Målingen af clock'en foretages med en logik analysator (Hewlett Packard Logic Analyzer HP 54620A LabNr. 33726) på ben 55 på Alteraen, som styrer DMA delen (UX-REFERANCE FINDES).
Resultatet af målingerne burde i teorien give en clockfrekvens på nøjagtig 16,000 MHz.
På figur C.2 ses clockfrekvensen målt med den anvendte logiske analysator.
Den målte clockfrekvens var ikke på 16,000 MHz. Den svingede derimod i området mellem 15,13 MHz og 16,67 MHz. Grunden til denne store afvigelse kan være, at logik analysatoren har en begrænset samplefrekvens. Denne medfører at samplingen (kvantiseringen) giver en vis usikkerhed svarende til 1/samplefrekvensen = 1/500MHz = 2 ns. Grundet Logik Analyserens lave samplerate forventes det, at den målte clockfrekvens svinger omkringe de ønskede 16,000 MHz med en afvigelse på op til 2 ns fra den fra den optimale clockperiode. Den målte clockfrekvens skal dermed ligge mellem 15,504 MHz og 16,529 MHZ. At nogle af målingerne lå uden for dette område kan skyldes unøjagtighed i det anvendte måleapparats sampleperiode.



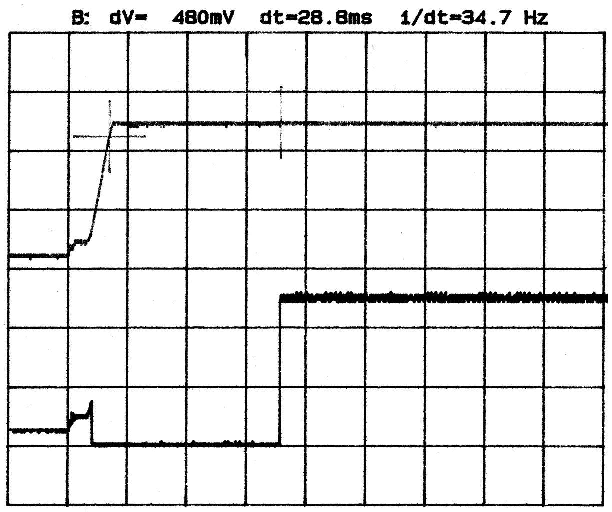
Figur C.2: Plot af clockperiode og duty cycle

C.3 Clock duty cycle

For at sikre, at den i Altera 7128 neddividerede clock's duty cycle er i overensstemmelse med de opstillede krav undersøges clock'ens periode i det følgende.
Målingen af clock'ens duty cycle foretages med en logik analysator (Hewlett Packard Logic Analyzer HP 54620A LabNr. 33726) på ben 55 på UX.
Den målte duty cycle bør i teorien ligge på 50 %.
Den målte duty cycle er på 51,6 % (Se figur C.2) Afvigelsen på 1,6 % fra den teoretiske kan tilskrives måleinstrumentet, da dennes samplinger kan afvige fra den ideelle med $\pm$ 2 ns (Se C.2) svarende til en duty cycle på 46,8 % - 53,2 %. Da duty cyclen ligger inden for den målbare grænse for den anvendte logik analysator, kan det konkluderes, at den neddividerede clock kan anvendes som reference clock til MCU'en.

C.4 LVI

For at sikre at de kritiske kredse i systemet (Altera, Display, MCU, RAM og ROM) startes korrekt, skal LVI'ens frigivelse af $\overline{RESET}$ signalet testes.
LVI'en testes ved at måle tiden reset signalet holdes af LVI'en efter, at der er sat spænding til systemet. Dette gøres ved at sammenligne 5 V spændingsforsyningens spændingsgraf med LVI'ens $\overline{RESET}$ udgang (Se figur C.3). Målingen er foretaget med (50 MHz oscilloscop med hukommelse + LABNR).
Det forventes, at LVI'ens $\overline{RESET}$ holdes i ca. 28 ms (Se afsnit 14.4.2).



Figur C.3: LVI'ens reset forsinkelse

$\overline{RESET}$ signalet bliver holdt i 28,8 ms. Dermed er kravet til LVI'en holdt, da denne skulle holde $\overline{RESET}$ i mindst 20 ms efter at spændingsforsyningen tændes.

C.5 ROM

I det følgende testes om ROM kredsene virker, og om de er korrekt forbundet til systemet. Testen udføres med et lille assembler program(reference), som brændes i ROM'en. Testprogrammet lader i en uendelig lykke MCU'en læse ROM'en igennem. Hver gang MCU'en har gjort dette flippes Port F1 (ben 77). Endvidere flippes MCU'ens Port F0 ben (ben 78), mens romtesten kører. Typen af eventuelle fejl vil kunne ses på MCU'ens Port F2 - F4 ben (ben 74 - 76), hvis ROM-testen går i stå. Således angiver Port F2, Port F3 og Port F4 henholdsvis om fejlen er en (long)word læsefejl, en nedre byte læsefejl eller en øvre byte læsefejl. Endelig angiver Port E benene (ben 80 - 82 og 85 - 89), i hvilken 1 kbyte blok fejlen fandt sted. Aktiviteten på disse ben måles med en logik analysator (Hewlett Packard Logic Analyzer HP 54620A LabNr. 33726).
Under testen forventes det, at man på logikanalysatoren kan se Port F0 og Port F1 på MCU'en flippe i hele testperioden.
ROM'en blev testet i over 10 minutter, hvor man på logikanalysatoren konstant kunne se, at Port F0 og Port F1 blev flippet. Da testen gav det forventede resultat, kan vi konkludere, at ROM kredsene virker og er korrekt forbundet til resten af systemet.

C.6 RAM

I det følgende testes om RAM kredsene virker, og om de er korrekt forbundet til systemet. Testen udføres med et lille assembler program(reference), som i en uendelig løkke lader MCU'en skiftevis læse og skrive data fra/til RAM'en. Under læsning fra RAM'en checkes der, om de læste data er de samme data, som blev skrevet i RAM'en. Hver gang MCU'en har gjort dette flippes Port F0 benet. Port F1 benet flippes hver gang hele RAM'en er blevet kørt igennem. Typen af eventuelle fejl vil kunne ses på MCU'ens Port F2 - F 5 ben, hvis RAM-testen fejler, da de i så fald enkeltvis begynder at flippe alt efter, hvilken fejl som er opstået. Endvidere skrives adressen ud på MCU'ens Port E ben (ben 80 - 82 og 85 - 89). Aktiviteten paa disse ben måles med en logik analysator (Hewlett Packard Logic Analyzer HP 54620A LabNr. 33726).
Under testen forventes det, at man på logikanalysatoren kan se en tæller køre kontinuert på signalerne fra MCU'ens Port F ben. Endvidere bør man kunne registrere aktivitet på bussen ved at betragte MCU'ens Port E ben.
ROM'en blev testet i over 10 minutter, hvor man på logikanalysatoren kunne se en tæller køre på Port F benene. Desuden kunne der registreres aktivitet på Port E benene. Da testen gav det forventede resultat, kan vi konkludere, at ROM kredsene virker og er korrekt forbundet til resten af systemet.

Appendiks D

Analog målejournal

D.1 Indledning

Målejounalen indholder følgende målinger :

- DC-målinger på udvalgte målepunkter.
- Signal/støj-forholdet for anti-aliaseringsfilteret.
- Plots af frekvensresponsen.

For at muliggøre disse målinger er måleopstillingen i bilag F.2 anvendt.
Det er praktisk ikke muligt at måle indgangsimpedansen, grundet operationsforstærkerne, der sidder på indgangen. Den indre indgangsmodstand i LF411 operationsforstærkerne er af størrelsesordenen $10^{12}\Omega$ [18].

D.2 DC-målinger

Måleopstillingen til DC-målingerne bilag F.2 indeholder 2 målepunkter. Indgangen på kredsløbet forbindes til stel, og frekvensområde 0 vælges, hvilket svarer til en stopbåndfrekvens på $22,1kHz$. DC-spændingerne i målepunkterne 1 og 2, måles med et oscilloscop. Spændingerne er peak-spændinger i forhold til stel. Resultatet af målinger er listet i figur D.1. Instrumenterne, der indgår i måleopstillingen, er angivet under punktet instrument indstillinger.

Målepunkt	Teoretisk værdi	Målt værdi	Enhed
1	2,5	2,5	V_p
2	2,5	2,5	V_p

Figur D.1: Tabel over DC-målingerne

D.3 Signal/støj-forhold

Måleopstillingen for måling af signal/støj-forholdet findes i bilag F.2. Til beregning af signal/støj-forholdet er følgende formel anvendt.

$$S/N = 20 \cdot log(\frac{udgangssignal(V_G = 2V_{pp}) - udgangssignal(V_G = 0V_{pp})}{udgangssignal(V_G = 0V_{pp})}) \quad \text{(D.1)}$$

Signalgeneratoren, V_G, afgiveren en sinusfrekvens på 1 kHz. Udgangssignalerne, målepunkt 2, er målt med et oscilloscop. Signalerne er peak-spændinger i forhold til stel. Bilag F.10 indeholder et plot af udgangssignalet for et generatorsignal på $0V_{pp}$, og bilag F.10 for et generatorsignal på $2V_{pp}$.
Udgangssignalerne er aflæst til følgende spændinger:

$$0V_{pp} \Rightarrow 70\mu V$$

$$2V_{pp} \Rightarrow 4,8V$$

Signal/støj-forholdet for anti-aliaseringsfilteret med ovenstående målinger er $37dB$.

D.4 Frekvensrespons

Måleopstillingen til måling af frekvensresponsen ses i bilag F.2. Indgangen, signalgenerator V_G, på kredsløbet forbindes til en sinus sweepgenerator (B&K 1051). Udgangen, målepunkt 2, forbindes til forstærkeren (B&K 2636). Resultatet af forstærkningen plottes ud på en recorder (B&K 2308). Samplefrekvensen findes ved at måle perioden mellem to nedadgående flanker på ADC'ens int ben, målepunkt 3, med et oscilloskop.
Den øvre og nedre grænsefrekvens er målt med forstærkeren (B&K 2636) ved at finde den maksimale forstærkning i pasbåndet. Med udgangspunkt i den maksimale forstærkning jursteres frekvensen (B&K1051) ned til forstærkningen er $6dB$ lavere en den maksimale.

Frekvensen aflæses som den nedre grænsefrekvens. Igen med udgangspunkt i den maksimale forstærkning justeres frekvensen (B&K1051) op til forstærkningen er $3dB$ lavere en den maksimale. Frekvensen aflæses som den øvre grænsefrekvens.
Ovenstående 3 målinger foretages for alle 8 frekvensområder, som frekvensanalysatoren kan indstilles til. Resultaterne fra målinger ses i figur D.2.

Måleområde	Filters frekvensområde	Målt nedre-grænsefrekvens	Målt øvre/ grænsefrekvens	sample-frekvens
0	20 Hz - 18,43 kHz	10 Hz	18,93 kHz	46,13 kHz
1	20 Hz - 12,29 kHz	10 Hz	12,61 kHz	30,71 KHz
2	20 Hz - 9,22 kHz	10 Hz	9,46 kHz	23,02 kHz
3	20 Hz - 7,37 kHz	10 Hz	7,57 kHz	18,44 kHz
4	20 Hz - 6,14 kHz	10 Hz	6,30 kHz	15,38 kHz
5	20 Hz - 5,27 kHz	10 Hz	5,40 kHz	13,18 kHz
6	20 Hz - 4,61 kHz	10 Hz	4,72 kHz	11,51 kHz
7	20 Hz - 4,10 kHz	10 Hz	4,20 kHz	10,25 kHz

Figur D.2: Tabel over samplefrekvens, nedre- og øvregrænsefrekvens

Der er valgt, at der vises, plots over frekvensresponsen for 2 måleområder (ydergrænserne nr. 0 og nr. 7). I nedenstående figur D.3 er betingelserne og placeringen af de enkelte plots angivet.

Måleområde	Sweepområde	Bilagsnr	Kommentar
0	1 Hz - 1 kHz	F.12	RMS averaging times = slow
0	30 Hz - 30 kHz	**??**	RMS averaging times = fast
7	10 Hz - 10 kHz	F.13	RMS averaging times =slow

Figur D.3: Tabel over plots af frekvensresponsen

Årsagen til at måleforstærkerens RMS averaging times er ændret til slow, ved måling af lave frekvenser, er unøjagtigheder i måleforstærkeren (B&K 2636). Denne unøjagtiged ses tydelig i plottet i bilag F.11. Plottet illustrerer et sweep fra 0,2 Hz til 200 Hz, hvor sweepgeneratoren (B&K 1051) er direkte forbundet med måleforstærkeren (B&K 2636). En anden test viser, at det ikke er sweepgeneratoren, der genererer unøjagtigheden. Med et oscilloscop observeres amplituden for ændringer i et sweepet af samme frekvensområde.

Da dette ikke er tilfældet, konkluderes der, at det er måleforstærkeren, som forårsaget unøjagtigheden.
Instrumenterne, der indgår i måleopstillingen samt deres indstillinger, findes i sektion D.5.

D.5 Instrumentindstillinger

Sinus sweep generator	B&K1051	Lbnr 08449
Funktion	Værdi	Enhed
output voltage	0,707	V_{rms}
sweep rate	25	mDec/s

Figur D.4: Aktuelle opsætning af sinus sweep generator B&K1051.

Måleforstærker	B&K2636	Lbnr 08451
Funktion	Værdi	Enhed
Input section gain	10	V
Output section gain	0	dB

Figur D.5: Aktuelle opsætning af måleforstærker B&K2636.

-X -Y Plotter	B&K2308	Lbnr 08450
Funktion	Værdi	Enhed
Range X & Y	20	mV/mm
Sweep X & Y	ext	
Polarity X & Y	norm	
Offset level plot af grænsefrekvenser	0	dB

Figur D.6: Aktuelle opsætning af -X -Y plotter B&K2308.

Funktionsgenerator	PM3212	Lbnr 07748
Funktion	Værdi	Enhed
Output voltage	5	V_p
sinus signal	1	kHz

Figur D.7: Aktuelle opsætning af funktionsgenerator PM3212.

Strømforsyning	Lbnr 124-20

D.6 Konklusion

Konklusionen på antialiasfilteret i henhold til DC-niveauerne er følgende: DC-spændingerne i målepunkterne er i overensstemmelse, som det fremgår af tabelen under punktet DC-målinger, med de teoretiske værdier. I henhold til frekvensresponsen har alle 8 områder en acceptabel afvigelse. Signal/støj-forholdet for anti-aliaseringesfilteret afviger fra kravet i kravspecifikationen. Kravet fra kravspecifikationen er anvendelsen af ADC1061. Denne har en opløsningevne på 0.098%, svarende til 60.2dB signalstøjforhold, hvor den målte/beregnede er 37 dB. Mulige fejlkilder til dette er:

- Aflæsningsfejl, da spændingerne er aflæst/beregnet ud fra plottene
- Grundet de lave spændinger, vil oscilloscopets nøjagtighed ved lave spændingerne, påvirke målingerne.
- Accumuleret støj i wire-wrap ledningerne (ikke afskærmede) til realiasering af filteret. Det kan dels stamme fra omgivelserne, og dels fra den digitale del af konstruktionen, da der findes, i forhold til de analog signaler, højhastigheds signaler i denne del.
- Støjen kan ligge overlejret i forsyningsspændingen til filteret. Forsyningsspændingen indgår, som en aktiv komponent i DC-offsetet, i pre og post behandlingen, i forhold til maxim filteret, af signalet fra terminalen til indgangen på ADC'en. I forsøg på at modvirke denne påvirkning er forsyningsledningerne i den analoge og digitale ført seperat i konstruktionen, med fælles punkt i terminalerne til boardet.
- Støj genereret af maxim filteret.

At foretage 10-bit A/D konvetering i dette system med det målte signal/støj-forhold, må betegnes som overkill. Dog bemærkes det, at systemet har karakter af en prototype - En massefremstillet udgave ville f.eks. ikke være wire-wrappet, og signal/støj-forholdet forventes da forbedret.

Da både DC-spændingerne i de enkelte målepunkter og grænsefrekvenserne er i overensstemmelse med de opstillede krav, kan designet anvendes i den samlede konstruktion.

Appendiks E

Headerfiler

Følgende indeholder headerfilerne fra kildekoden, til softwaren. Formålet hermed er på en overskuelig måde, at fremstille den API, de forskellige program dele stiller tilrådighed for hinanden.

E.1 OS_defines.inc

```
/*************************************************************************
 * Fil: OS-DEFINES.H
 * Beskrivelse: Knst. til OS kald - til 4. Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * Sidst ændret: 25-05-99 13:58/
/************************************************************************/

/* OS */
.equ OS_TRAP,13

/* Display driver */
.equ OS_CALL_DISPLAY_INIT,0
.equ OS_CALL_DISPLAY_DEINIT,1
.equ OS_CALL_DISPLAY_ON,2
.equ OS_CALL_DISPLAY_OFF,3
.equ OS_CALL_DISPLAY_GETSCREENBUFFER,4
.equ OS_CALL_DISPLAY_UPDATESCREEN,5
```

```
.equ OS_CALL_DISPLAY_GOTOXY,6
.equ OS_CALL_DISPLAY_GETXPOS,7
.equ OS_CALL_DISPLAY_GETYPOS,8
.equ OS_CALL_DISPLAY_SCROLL,9
.equ OS_CALL_DISPLAY_PUTCHAR,10

/* Sample driver */
.equ OS_CALL_SAMPLE_INIT,11
.equ OS_CALL_SAMPLE_DEINIT,12
.equ OS_CALL_SAMPLE_TRANSFER,13
.equ OS_CALL_SAMPLE_IS_DATA_READY,14
```

E.2 OS.h

```
/*************************************************************************
 * Fil: OS.H
 * Beskrivelse:
 * header fil til OS'et til 4.Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * Skal kaldes fra supervisor mode!
 * Sidst ændret: 26-05-99 17:20/
/************************************************************************/

#ifndef OS_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define OS_H_

 extern void os_init(void);
 /* initialiserer OS'et (bliver kaldt af opstartkode) (efter dette kald vil
    OS+drivere være initialiseret + vi befinder os i usermode) */

 extern void os_setvector(int vectornumber,void* isr);
 // installerer en ISR.

 extern void* os_getvector(int vectornumber);
 // returnerer adressen på den givne vektor.
```

```
#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.3 Displaymanager.h

```
/************************************************************************
 * Fil: DISPLAYMANAGER.H
 * Beskrivelse:
 * header fil til display Mgr. til 4.Semesters projekt (Frekvensanalyse)
 ************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * se errorcodes.h for beskrivelse af fejlkoderne
 * Sidst ændret: 21-05-99 10:31/
/************************************************************************/

#ifndef DISPLAYMANAGER_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define DISPLAYMANAGER_H_

 #include"errorcodes.h"

 extern error displaymanager_init(void);
 // init funktionen.

 extern error displaymanager_deinit(void);
 // deinit funktionen.

 extern error displaymanager_clrscr(void);
 // sletter skærmbufferen

 extern error displaymanager_printf(const char *format, ...);
 // skriver en formateret string til skærmbufferen (opdaterer skærm med det samme)

 extern error displaymanager_writebar(const int x,const int height);
 // tegner en lodret bar på position x med højden height (x=0..127, height=0..64)

 extern error displaymanager_writepixel(const int x,const int y,const char value);
 // sætter pixelen på position (x,y) lig value
```

```
extern error displaymanager_flush(void);
// opdaterer skærmen med indholdet af skærmbufferen

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.4 Display_PC.h

```
/*************************************************************************
 * Fil: DISPLAY_PC.H
 * Beskrivelse:
 * header fil til display på PC - 4. Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * se errorcodes.h for beskrivelse af fejlkoderne
 * Sidst ændret: 21-05-99 10:31/
/*************************************************************************/

#ifndef DISPLAY_PC_H_ // sørg for at header filen kun bliver inkluderet 1 gang
#define DISPLAY_PC_H_

#include"errorcodes.h"

 // størrelsen af LCD displayet
#define DISPLAY_XRES 128
#define DISPLAY_YRES 64

// font størrelse
#define DISPLAY_CHARWIDTH 8
#define DISPLAY_CHARHEIGHT 8
#define DISPLAY_CHARSIZE 8
#define DISPLAY_NUMBEROFCHARS 128

// filnavn på font
#define DISPLAY_FONTFILENAME "FONT_MOT.DAT"

// antallet af tegn på skærmen
#define DISPLAY_CHAR_XRES DISPLAY_XRES/DISPLAY_CHARWIDTH
```

```
#define DISPLAY_CHAR_YRES DISPLAY_YRES/DISPLAY_CHARHEIGHT

// lav vores c++ navne om til c navne (til linking)
#ifdef __cplusplus
 extern "C" {
#endif

/*************************************************************************
                    Generelle funktioner
 *************************************************************************/

extern error display_init(void);
// init funktionen.

extern error display_deinit(void);
// deinit funktionen.

extern error display_on(void);
// tænder for display,et.

extern error display_off(void);
// slukker for display,et.

extern void* display_getscreenbuffer(void);
// returnerer en pointer til skærmbufferen

extern error display_updatescreen(char bitmask);
// opdaterer skærmen med indholdet af skærmbufferen. (ifølge bitmask)

/*************************************************************************
                    Tekstrelaterede funktioner
 *************************************************************************/

extern error display_gotoxy(const int x,const int y);
// bevæger cursoren til (x,y)

extern int display_getxpos(void);
// returnerer nuværende x-position

extern int display_getypos(void);
```

```
// returnerer nuværende y-position

extern error display_scroll(void);
// scroller skærmbufferen en tekstlinje op (DISPLAY_CHARHEIGHT pixels)

extern error display_putchar(const char c);
// skriver et tegn på skærmbufferen på nuværende position (håndterer ikke kontroltegn) (kun ASCII 0..127)

#ifdef __cplusplus
 }
#endif

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.5 Display_Motorola.h

```
/*************************************************************************
 * Fil: DISPLAY_MOTOROLA.H
 * Beskrivelse:
 * header fil til display på MCU - 4. Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * se errorcodes.h for beskrivelse af fejlkoderne
 * Sidst ændret: 10-05-99 10:59/
/*************************************************************************/

#ifndef DISPLAY_MOTOROLA // sørg for at header filen kun bliver inkluderet 1 gang
 #define DISPLAY_MOTOROLA

 #include"errorcodes.h"

  // størrelsen af LCD displayet
 #define DISPLAY_XRES 128
 #define DISPLAY_YRES 64

 // font størrelse
 #define DISPLAY_CHARWIDTH 8
 #define DISPLAY_CHARHEIGHT 8
```

```
#define DISPLAY_CHARSIZE 8
#define DISPLAY_NUMBEROFCHARS 128

// antallet af tegn på skærmen
#define DISPLAY_CHAR_XRES DISPLAY_XRES/DISPLAY_CHARWIDTH
#define DISPLAY_CHAR_YRES DISPLAY_YRES/DISPLAY_CHARHEIGHT

/**************************************************************************
                          Generelle funktioner
 **************************************************************************/

extern error display_init(void);
// init funktionen.

extern error display_deinit(void);
// deinit funktionen.

extern error display_on(void);
// tænder for display,et.

extern error display_off(void);
// slukker for display,et.

extern void* display_getscreenbuffer(void);
// returnerer en pointer til skærmbufferen

extern error display_updatescreen(char bitmask);
// opdaterer skærmen med indholdet af skærmbufferen. (ifølge bitmask)

/**************************************************************************
                        Tekstrelaterede funktioner
 **************************************************************************/

extern error display_gotoxy(const int x,const int y);
// bevæger cursoren til (x,y)

extern int display_getxpos(void);
// returnerer nuværende x-position

extern int display_getypos(void);
```

```
// returnerer nuværende y−position

extern error display_scroll(void);
// scroller skærmbufferen en tekstlinje op (DISPLAY_CHARHEIGHT pixels)

extern error display_putchar(const char c);
// skriver et tegn på skærmbufferen på nuværende position (håndterer ikke kontroltegn) (kun ASCII 0..127)

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.6 Samplemanager.h

```
/*************************************************************************
 * Fil: SAMPLEMANAGER.H
 * Beskrivelse:
 * header fil til sample Manager til 4.Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * se errorcodes.h for beskrivelse af fejlkoderne
 * Sidst ændret: 25−05−99 21:01/
/*************************************************************************/

#ifndef SAMPLEMANAGER_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define SAMPLEMANAGER_H_

 #include"errorcodes.h" // til fejlkoder
 #include"fftserver.h"  // den komplekse type

 #define SAMPLEMANAGER_TIMEOUTVALUE 0x7FFFFFFF
 // timeout værdi − hvis hardwaren ikke har svaret inden mcu,en har talt til denne værdi vil kaldet

 extern error samplemanager_init(int samplerate);
 // init funktionen.

 extern error samplemanager_transfer(int numberofsamples,complex *buffer);
 // starter en DMA overførsel med numberofsamples samples fra a/d konverteren til buffe
 // buffer skal pege på et array af størrelsen sizeof(long)*blocksize (se hardware)
```

```
extern error samplemanager_wait_for_new_data();
// venter indtil bufferen bliver fyldt klar.

extern error samplemanager_deinit(void);
// deinit funktionen.

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.7 Sample_PC.h

```
/*************************************************************************
 * Fil: SAMPLE_PC.H
 * Beskrivelse:
 * header fil til sampler på PC - 4. Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * se errorcodes.h for beskrivelse af fejlkoderne
 * Sidst ændret: 16-05-99 20:05/
/*************************************************************************/

#ifndef SAMPLE_PC_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define SAMPLE_PC_H_

 #include"errorcodes.h"
 #include"fftserver.h"

 #define SAMPLE_DATA_NOT_READY 0
 #define SAMPLE_DATA_READY 1

 #define SAMPLE_FILENAME "inputfile.dat"
 // filnavnet på den fil der samples (læses) fra.

 extern error sample_init(int samplerate);
 // init funktionen.

 extern error sample_transfer(int numberofsamples,complex *buffer);
 // starter en DMA overførsel med numberofsamples samples fra a/d konverteren til buffer.
 // buffer skal pege på et array af størrelsen sizeof(complex)*blocksize
```

```
extern int sample_is_data_ready();
// returnerer SAMPLE_DATA_NOT_READY hvis data ikke er klar, og SAMPLE_DATA_READY hvis data er klar.

extern error sample_deinit(void);
// deinit funktionen.

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.8 Sample_Motorola.h

```
/*********************************************************************
 * Fil: SAMPLE_MOTOROLA.H                                            *
 * Beskrivelse:                                                      *
 * header fil til sampler på MCU - 4. Semesters projekt (Frekvensanalyse) *
 *********************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)                            *
 * Noter:                                                            *
 * se errorcodes.h for beskrivelse af fejlkoderne                    *
 * Sidst ændret: 01-06-99 12:48                                      *
*/

#ifndef SAMPLE_MOTOROLA_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define SAMPLE_MOTOROLA_H_

 #include"errorcodes.h"
 #include"fftserver.h"

 #define SAMPLE_DATA_NOT_READY 0
 #define SAMPLE_DATA_READY 1

 extern error sample_init(int samplerate);
 // init funktionen.

 extern error sample_transfer(int numberofsamples,complex *buffer);
 // starter en DMA overførsel med numberofsamples samples fra a/d konverteren til buffer.
 // buffer skal pege på et array af størrelsen sizeof(complex)*blocksize

 extern int sample_is_data_ready();
```

```
// returnerer SAMPLE_DATA_NOT_READY hvis data ikke er klar, og SAMPLE_DATA_READY hvis data er klar.

extern error sample_deinit(void);
// deinit funktionen.

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.9 FFTserver.h

```
/************************************************************************
 * Fil: FFTSERVER.H
 * Beskrivelse:
 * Header fil til FFT server til 4.Semesters projekt (Frekvensanalyse)
 ************************************************************************
 * Af Karsten Jensen(sunrisekom.auc.dk)&Peter Korsgaard(jacmetkom.auc.dk)
 * Noter:
 * Sidst ændret: 18-05-99 20:51/
/************************************************************************/

#ifndef FFTSERVER_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define FFTSERVER_H_

 #include"errorcodes.h" // til fejlkoder

 // antal bits til FFT (fft længde er 2^FFT_BITS)
 #define FFT_BITS 7

 // længden af vores FFT (skal være 2^n)
 #define FFT_LENGTH (1<<FFT_BITS)

 // skal vi bruge fixed point eller floating point?
 #define FFT_FIXEDPOINT

 #ifdef FFT_FIXEDPOINT // hvis fixed point

  // definér hvordan man ganger
  #define FFT_MULTIPLY(a,b) (long)(((long long)(a)*(long long)(b))>>FFT_FRACTIONBITS)

  // definér hvordan man konverterer fra float til format
```

```
 #define FFT_CONVERT(a) (long)((a)*FFT_FRACTIONSIZE)

 // definér hvordan man konverterer til float
 #define FFT_TO_FLOAT(a) ((double)(a))/FFT_FRACTIONSIZE

 // hvilken type skal FFT,en udføres på
 #define FFT_TYPE long

 // hvor mange bits til heltal
 #define FFT_FIXEDBITS 10

 // selvberegnende defines
 #define FFT_FIXEDSIZE (1<<FFT_FIXEDBITS)
 #define FFT_FRACTIONBITS (sizeof(FFT_TYPE)*8-FFT_FIXEDBITS)
 #define FFT_FRACTIONSIZE (1<<FFT_FRACTIONBITS)

#else // hvis floating point

 // definér hvordan man ganger
 #define FFT_MULTIPLY(a,b) ((a)*(b))

 // definér hvordan man konverterer fra float til format
 #define FFT_CONVERT(a) (a)

 // definér hvordan man konverterer til float
 #define FFT_TO_FLOAT(a) (a)

 // hvilken type skal FFT,en udføres på
 #define FFT_TYPE double
#endif

// en type der håndterer de komplekse frekvenskomposanter.
typedef struct{
 FFT_TYPE re,im;
} complex;

/*************************************************************************
                              funktioner
 *************************************************************************/
```

```
extern error fftserver_init();
// init funktionen.

extern error fftserver_deinit();
// deinit funktionen.

extern void fftserver_calcfft(complex *data);
// beregner fft og returnerer en pointer til en sekvens af data (størrelse: FFT_LENGTH/2)

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.10 LCDklient.h

```
/***************************************************************************
 *
 *        Header-fil til lcd_klient.c 4.Semesters projekt (Frekvensanalyse)
 *
 ***************************************************************************
 * Af Claus Albøge (tractrixkom.auc.dk)
 * Sidst ændret: 11-5/99 14:22/
/**************************************************************************/

#ifndef LCDKLIENT // sørg for at header filen kun bliver inkluderet 1 gang
 #define LCDKLIENT

 #include"errorcodes.h"
 #include"fftserver.h"

 #define LCDKLIENT_TABLEBITS_RE 8 //størrelse på "re-akse" i lcdklienttabel
 #define LCDKLIENT_TABLEBITS_IM 8 //størrelse på "im-akse" i lcdklienttabel

 #define LCDKLIENT_TABLESIZE_RE (1<<LCDKLIENT_TABLEBITS_RE) //størrelse på "re-akse" i lcdklienttabel
 #define LCDKLIENT_TABLESIZE_IM (1<<LCDKLIENT_TABLEBITS_IM) //størrelse på "im-akse" i lcdklienttabel

 #define LCDKLIENT_DISPLAYMODE_POWERSPECTRUM 1
 #define LCDKLIENT_DISPLAYMODE_AMPLITUDESPECTRUM 2

 extern error lcdklient_init(int displaymode);
 // initialiserer lcdklienten
```

```
  extern void lcdklient_update(complex input[]);
  // Update funktion til lcdklient.

  extern error lcdklient_deinit(void);
  // Deinit funktion til lcdklient.

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.11 Errorcodes.h

```
/***********************************************************************
 * Fil: ERRORCODES.H
 * Beskrivelse:
 * Header fil der definerer fejlkoder til 4. Semesters projekt
 ***********************************************************************
 * Af Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * Ingen.
 * Sidst ændret: 25-05-99 17:44/
/***********************************************************************/

 #ifndef ERRORCODES // sørg for at header filen kun bliver inkluderet 1 gang
  #define ERRORCODES

  // typen der bliver brugt til fejlkoder
  #define error int

  // forskellige fejlkoder
  #define ERROR_OK 0                              // ingen fejl
  #define ERROR_PARAMETER_OUT_OF_BOUNDS 1    // parameter udenfor lovlig række
  #define ERROR_TIMED_OUT 2                       // hardwaren svarede ikke indenfor den forventede tid.
  #define ERROR_FILE_NOT_FOUND 3                  // filen blev ikke fundet
  #define ERROR_END_OF_FILE_REACHED 4          // vi er nået til slutningen af filen
  #define ERROR_MEMORY_ALLOCATION_FAILED 5   // ikke nok hukommelse
  #define ERROR_UNKNOWN_PARAMETER 6             // ikke lovlig parameter
  #define ERROR_UNKNOWN_ERROR                     // ukendt fejl (aka Microsoft fejl ;)
 #endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.12 Displayfont.h

```
/***************************************************************************
 * Fil: DISPLAY_MOTOROLA_FONT.H
 * Beskrivelse:
 * header fil fonten der bruges i display_motorola.c
 ***************************************************************************
 * Af: Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * display_font.o er lavet af raw2s.
 * Sidst ændret: 21-05-99 10:31/
/***************************************************************************/

#ifndef DISPLAY_FONT_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define DISPLAY_FONT_H_

 #ifdef _PC_
  #include"display_pc.h"
 #else
  #include"display_motorola.h"
 #endif

 /***************************************************************************
                          Globale variable
  ***************************************************************************/

 extern unsigned char font[DISPLAY_NUMBEROFCHARS][DISPLAY_CHARSIZE];

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.13 FFTserver-koefficienter.h

```
/***************************************************************************
 * Fil: FFTSERVER_COEFFICIENTS.C
 * Beskrivelse:
 * Koefficienter til fftserveren til 4.Semesters projekt (Frekvensanalyse)
 ***************************************************************************
 * Af Peter Korsgaard (jacmetkom.auc.dk)
```

```
 * Noter:
 * Sidst ændret: 13−05−99 17:36/
/*************************************************************************/

#ifndef FFTSERVER_COEFFICIENTS_H_ // sørg for at header filen kun bliver inkluderet 1 gang
 #define FFTSERVER_COEFFICIENTS_H_

 #include"fftserver.h"

 /*************************************************************************
                                    globale variable
  *************************************************************************/

 extern complex coefficients[FFT_LENGTH];

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.14 LCDklient-amplitudetable.h

```
/*************************************************************************
 * Fil: LCDKLIENT_AMPLITUDETABLE.H
 * Beskrivelse:
 * Koeff. til log powertabel til 4.Semesters projekt (Frekvensanalyse)
 *************************************************************************
 * Af Peter Korsgaard(jacmetkom.auc.dk)&Claus Albøge(tractrixkom.auc.dk)
 * Noter:
 * Sidst ændret: 17-05-99 17:12/
/*************************************************************************/

#ifndef LCDKLIENT_AMPLITUDETABLE // sørg for at header filen kun bliver inkluderet 1 gang
 #define LCDKLIENT_AMPLITUDETABLE

 #include"lcdklient.h"

 /*************************************************************************
                                    globale variable
  *************************************************************************/

 extern unsigned char amplitudetable[LCDKLIENT_TABLESIZE_RE][LCDKLIENT_TABLESIZE_IM];
```

```
#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.15 LCDklient-powertable.h

```
/****************************************************************************
 * Fil: LCDKLIENT_POWERTABLE.H
 * Beskrivelse:
 * Koeff. til log powertabel til 4.Semesters projekt (Frekvensanalyse)
 ****************************************************************************
 * Af Peter Korsgaard(jacmetkom.auc.dk)&Claus Albøge(tractrixkom.auc.dk)
 * Noter:
 * Sidst ændret: 17-05-99 17:12/
/****************************************************************************/

#ifndef LCDKLIENT_POWERTABLE // sørg for at header filen kun bliver inkluderet 1 gang
 #define LCDKLIENT_POWERTABLE

 #include"lcdklient.h"

 /****************************************************************************
                              globale variable
  ****************************************************************************/

 extern unsigned char powertable[LCDKLIENT_TABLESIZE_RE][LCDKLIENT_TABLESIZE_IM];

#endif // sørg for at header filen kun bliver inkluderet 1 gang
```

E.16 Linker-script

```
/**************************************************************************
 * Fil: MCU-WITH-RELOC.LD
 * Beskrivelse: Linker script til 4.Semesters projekt (Frekvensanalyse)
 **************************************************************************
 * Af Peter Korsgaard (jacmetkom.auc.dk)
 * Noter:
 * Sidst ændret: 26−05−99 17:19/
```

```
/**********************************************************************/

/**********************************************************************
 * Memory map                                                          *
 **********************************************************************
 *
 * The memory map looks like this:
 *
 * +——————+ ROM: Base = 0x000000 - Length = 0x40000
 * |-Temp. vector table |
 * |-Startup code (boot)|
 * |-Code + data (not   |
 * | relocated)         |
 * |                    |
 * +——————+ RAM: Base = 0x100000 - Length = 0x40000
 * |Vector table (boot) |
 * +——————+
 * |Relocated code      |
 * +——————+
 * |Relocated data      |
 * +——————+
 * |bss (cleared by     |
 * |boot.o)             |
 * +——————+
 * |user mode stack     |
 * +——————+
 * |supervisor stack    |
 * +——————+
 *
 */

/**********************************************************************
 * Target processor and file format                                   *
 **********************************************************************/

/* Output architecture                                                */
OUTPUT_ARCH(m68k)

/* Generate S-records                                                 */
OUTPUT_FORMAT(srec)
```

```
/**********************************************************************
 * Search directories and library information                     *
 **********************************************************************/

SEARCH_DIR(.)

/* No shared (dynamic) libraries */
__DYNAMIC  =  0;

/**********************************************************************
 * Required files                                                   *
 **********************************************************************/

/* Start-up File that is always loaded first                       */
STARTUP(boot.o)

INPUT(os.o)

/**********************************************************************
 * Sections                                                         *
 **********************************************************************/

SECTIONS
{

/**********************************************************************
 * Ram size and location                                            *
 **********************************************************************/
  ramstart = 0x100000;
  ramsize  = 0x040000;
  ramend   = ramstart + ramsize;

/**********************************************************************
 * Rom size and location                                            *
 **********************************************************************/
  romstart = 0x000000;
  romsize  = 0x040000;
  romend   = romstart + romsize;
```

```
  save_supervisor_stack = ramend - 4;
  save_return_address = save_supervisor_stack - 4;

/**********************************************************************
 * Stack size and location                                            *
 **********************************************************************/
  supervisor_stacksize = 8192;
  user_stacksize = 8192;

  supervisor_stack = save_return_address;
  user_stack = supervisor_stack - supervisor_stacksize;

/**********************************************************************
 * div constants                                                      *
 **********************************************************************/
  exceptionvectorstart = ramstart;
  exceptionvectorsize = 1024;
  exceptionvectorend = 0x100000+exceptionvectorsize;
  codestart = exceptionvectorend;

/**********************************************************************
 * startup code (not relocatable)                                     *
 **********************************************************************/
  .startup romstart :
  {
    boot.o
    estart = .;
    real_estart = .;
  }

/**********************************************************************
 * all the rest of the code (relocatable)                             *
 **********************************************************************/
  .text codestart : AT (real_estart)
  {
    *(.text)
    etext = .;
    real_etext = etext + real_estart - codestart;
  }
```

```
/*************************************************************************
 * all initialized data (relocatable)                                    *
 *************************************************************************/
  .data : AT (real_etext + 2)
  {
    *(.data)
    edata = .;
    real_edata = edata + real_estart - codestart;
  }

/*************************************************************************
 * all uninitialized data (relocatable)                                  *
 *************************************************************************/
  .bss :
  {
    *(.bss)
    ebss = .;
    real_ebss = ebss + real_estart - codestart;
  }
}
```

Appendiks F

Bilag - diagrammer og måleplot

F.1 Dataopsamling

F.2 Diagram - Målopstilling til analog målejournal

F.3 Diagram - Resetkredsløb

F.4 Diagram - ROM kredsløbet

F.5 Diagram - RAM kredsløbet

F.6 Diagram - Sample DMA

F.7 Diagram - Display DMA

F.8 Diagram - Antialiaseringsfilter

F.9 Plot - Frekvensrespons af øvre grænsefrekvens (måleområde 0)

F.10 Plot - Signal/Støj forhold

F.11 Plot - Måleforstærker frekvensrespons

F.12 Plot - Frekvensrespons af nedre grænsefrekvens

F.13 Plot - Frekvensrespons af øvre grænsefrekvens (frekvensområde 7)