INNHOLD

1. INNLEDNING

1.1 VoiceControl; En reise i fonetikkens verden

Talegjenkjenning; ordet som er synonymt med noe av det vanskeligste og mest avanserte innen dagens forskningsverden. De ressursene som stilles til disposisjon for å lage et fullgodt universal system for gjenkjenning av tale, er nærmest uten grenser. Rammene som foreligger for vårt prosjekt Voice Control er riktignok ikke helt i den samme størrelsesorden, men målet som ligger i bunnen er det samme: Vi skal utvikle et system som kan kjenne igjen talebaserte kommandoer, og utføre handlinger ut fra disse. I hvor stor grad vi klarer å oppnå dette målet, er helt avhengig av hvor gode vi er til å unytte den informasjonen som er tilgjengelig for oss på markedet. Området er så stort og uoversiktlig at det ikke handler om å utvikle nye måter å gjøre tingene på, men hvordan vi kan tilpasse, utnytte og utvikle de vertøyene og algoritmene som allerede er tilgjengelige for oss. Prosjektet blir som en reise tilbake i tiden, hvor vi stifter bekjentskap med noen av tidenes største eksperter innen fonetikk, og går i deres fotspor.

Hovedprosjekt er en obligatorisk del av ingeniør utdannelsen her ved Høgskolen i Gjøvik, og har et omfang på 15 studiepoeng. Da valget falt på en oppgave om talegjenkjenning, visste vi med en gang at det ikke ville mangle på utfordringer. Vi ønsker imidlertid å poengtere at valg av oppgave handler mer om interesse, enn å finne den enkleste veien fra A til B. Hovedprosjektet er på en måte den ultimate testen HiG har laget for oss, slik at vi kan sette på prøve det et to og et halvt års studieforløp har påført oss av kunnskap. Da sier det seg vel selv at det trengs utfordringer?!
Vi vil i denne sammenhengen takke vår veileder Halgeir Leiknes, og også Arne Wold for mye god input. Tilslutt vil vi takke oppdragsgiver Intelligent Control v\Ingolf Brudeli, for å ha gitt oss en interessant og utfordrende oppgave.

Rapporten vil ta for seg vår reise inn i fonetikkens verden, og hvordan vi har jobbet for å finne løsninger på de problemstillingene vi har satt oss, og som oppgaven i seg selv gir.

1.2 Spesifisering av oppgaven

Voice Control er et system for talegjenkjenning som skal være virksomt sammen med LON Works teknologien, og kunne styre ulike noder i et komplett system av talestyrte enheter. Grunntanken for konstruksjonen er å kunne få en taleinnput via en mikrofon, behandle disse data med avansert signalbehandling, og sende ut igjen en enkel eksekveringskommando til LON Works nettverket. I nettverket skal kommandoen styres til den aktuelle noden, slik at kommandoen utføres på denne. Systemet skal i første omgang designes for å benyttes i et normalt hjem, slik at et eksempel på effekten av en systemkommando kan være et lys som slås på. I utgangspunktet skal systemet kunne behandle og skille fem ulike stemme gitte kommandoer.

For at systemet skal være så mobilt som mulig innenfor den tenkte operasjonsplassen, ønsker vi en hardware løsning på sender siden som er bærbar. Denne spesifikasjonen stiller krav både til utformingen av selve produktet, og til hvilken type interface vi benytter sammen med LON Works nettverket. Trådløst grensesnitt er den eneste reelle muligheten for en enklest mulig bærbar løsning, og vi skal derfor benytte en trådløs RF tranciever fra Kongsberg Analogic for LON kommunikasjon. Nodene skal kobles sammen med tvinnet parkabel til et FTT-10 type nettverk, noe som gjør at vi trenger en ruter mellom RF signalene og FTT-10 signalene. Ruter teknologien leveres også av Kongsberg Analogic.

1.3 Målgruppen for rapporten

I og med at vi har kommet på et visst teknisk nivå under utdannelsesforløpet, vil rapporten være skrevet i lys av dette. Når det gjelder vanskelighetsgrad vil innholdet stort sett ligge på Bachelor Elektro Ingeniør nivå, med noen reservasjoner. Den teorien som ligger bak stemmegjenkjenning er såpass avansert, at det vil kreve en spesifikk innsikt på høyt nivå innenfor dette området, for å forstå alt som ligger bak. Dette vil først gjøre seg gjeldende når programkoden i vedlegg ***** , og algoritmen for stemmegjenkjenning, skal gås nærmere i sømmene. Utviklingen av algoritmen som er benyttet for å lage software som prosesserer stemmekommandoen har også et veldig avansert grunnlag, slik at det tekniske her er lagt som vedlegg for nærmere studie. Skal disse prosedyrene forstås fullt ut anbefaler vi å ta en titt på noen av de litteraturreferansene vi har satt opp, for å skaffe et tilstrekkelig grunnlag.

1.4 Organisering av rapporten

Siden Voice Control prosjektet vårt omfatter store områder innen flere forskjellige fagfelter, vil rapporten naturlig nok bli oppdelt i henhold til dette. Etter de innledende avsnittene vil vi forsøke å gi en konkret innføring i testkortet vi har jobbet med, og de verktøyer som er knyttet til dette. Dette blir den tekniske delen av rapporten. Testkortet vi har brukt, DSK6416320C, er presentert med sine grunnleggende funksjoner, uten å gå for mye inn på de mest tekniske delene av dette. Når det gjelder software er det Code Composer som kommuniserer direkte med testkortet. Vi forsøker å gi en oversikt over programmets funksjoner, samtidig som vi ønsker å si noe om hvordan vi setter opp kortet softwaremessig. Denne delen vil stort sett ta for seg oppsett av grunnleggende registre for hardware på kortet, og hvordan disse fungerer sammen*******. Til slutt I den tekniske seksjonen vil vi ta en kort diskusjon på det vi har oppfattet som problemer underveis, og hvilke forsalg til løsninger vi har hatt på disse.

Neste seksjon vil omhandle de aspektene som faller under stemmgjenkjenningsparolen.
Her vil vi gi en generell innledning til de vanligste innfallsvinklene, forklare hvordan vi har jobbet oss fram til den algoritmen vi har benyttet, og begrunne valg av endelig struktur. Forklaring til C++ koden som danner fundamentet for algoritmen blir den mest vitale biten, og vil bli tilegnet mye plass. Til slutt I avsnittet tar vi med en diskusjonsbit på de løsningene som er valgt, og hva som kan være alternative metoder.

LONWorks danner siste konkrete del av rapporten. Her presenteres den løsningen som er valgt for oppsett av nettverket, og hvilke bestanddeler dette består av. Helt til slutt vil vi bidra med en diskusjon rundt oppgaven rent konkret, og litt rundt de forholdene og rammene som ble lagt til grunn for utførelsen. Som en avrunding kommer konklusjonene vi har trukket, med avsluttende kommentarer.

Vedleggsdelen blir omfattende. Siden mye av stoffet som ligger til grunn for stemmegjenkjenning og algoritmen er i overkant teknisk til å ha med i rapporten, blir dette vedlegg. Rapporten presenterer de rent overfladiske aspektene under hvert punkt, og vil referere til vedleggene for ytterligere fordypning. Vedleggsdelen vil bestå av følgende momenter:

- Fullstendig programkode for stemmegjenkjenningsalgoritmen
- Programkode for Neuron chipen
- Tegninger for Hardware, med andre ord testkortet
- Utledninger av formler brukt i algoritmen og for å lage verktøys funksjoner I C++
- Logg over prosjektets gang
- Kildehenvisninger og henvisninger til aktuelle datablader

1.5 Arbeidsform

Vi håper at organiseringen skal gi et oversiktlig bilde av hva vi har gjort, og at rapporten skal bli lettfattelig å lese. Det tunge bakgrunnsstoffet er bevisst lagt som vedlegg, slik at dette ikke skal ødelegge den helhetlige leseropplevelsen.

Før vi gikk i gang med prosjektet hadde vi møte med arbeidsgiver, hvor vi fikk de siste spesifikasjoner på plass. Spesifikasjonene gikk stort sett på størrelse, overføringsmønster, og hvilke tranciever som skulle benyttes opp mot LON nettverket. For øvrig sto vi fritt til å velge løsninger selv, og når det gjaldt verktøyet for stemmegjenkjenning, hadde vi ingen konkrete retningslinjer.

Størrelsen på oppgaven førte til at vi med en gang fordelte arbeidsoppgaver, og har hele tiden jobbet mer eller mindre selvstendig hver vår del. Oppgavefordelingen kan stort sett sees i lys med de tre hoveddelene rapporten er delt inn i; Hardware/LON , Software med hovedvekt på Code Composer, og stemmegjenkjenning. Riktignok har grensene til tider vært noe flytende, men grunnstrukturen i arbeidet ligger på de områdene.

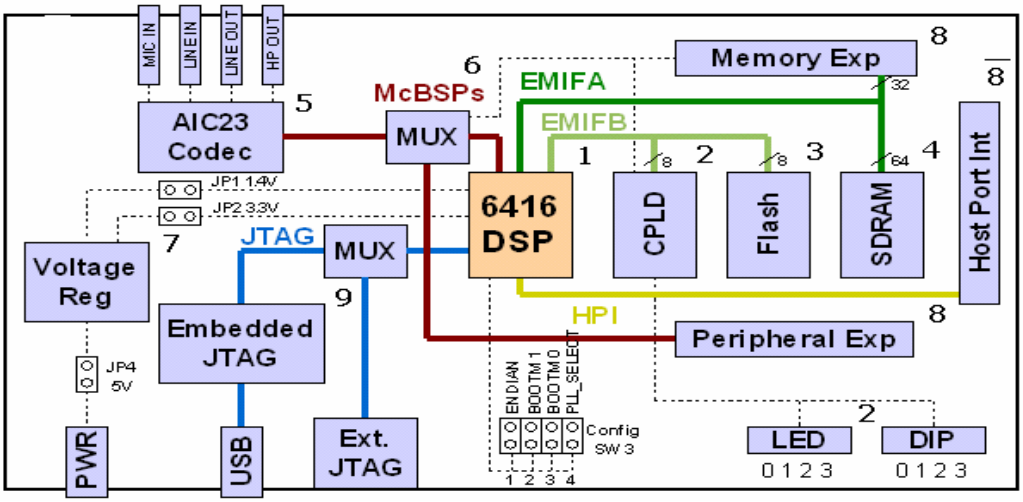
Arbeidsområdet har stort sett vært HiG, selv om enkelte deler innen programmering har blitt framstilt i det private. Utviklingsmateriell har skolen stilt til disposisjon, sammen med faglig kompetanse.

Kommunikasjon med oppdragsgiver underveis har stort sett foregått over telefon, e-mail og via oppdatering på prosjektets hjemmeside. Her har vi prøvd å legge ut statusrapporter jevnlig, på alle aspekter ved prosjektet. Statusmøter underveis har vi ikke hatt for mange av, men vi har kalt inn veileder når vi har følt dette nødvendig. Gruppen har jobbet selvstendig, og jevnlig hatt egne møter for å gjøre opp status. Det er i hovedsak resultater fra disse som er publisert på hjemmesiden.

2.
PRESENTASJON AV TESTKORT DSK6416C320

2.1 Sammendrag av kjernefunksjoner

Dette avsnittet vil i grove trekk ta for seg grunnleggende funksjoner på testkortet, rent hardwaremessig, uten å gå i for tekniske detaljer. Det er mer ment som en overordnet oversikt, slik at det raskt skal være mulig å sette seg inn i kortets kapasitet, og få en oversikt over tilgjengelige bruksområder. En illustrerende oversikt kan gjøre det enklere å se hvilke hovedområder det er snakke om. Disse områdene er nummerert på figuren for å gjøre det enklere å finne fram i teksten. En generell kommentar er at alle komponenter har 3.3V drivspenning på testkortet, og dette er derfor ikke kommentert under hver enkelt komponent. Skulle andre drivspenninger være benyttet vil heller dette poengteres.



Figur 2.1 (1) Oversikt over hovedelementene på DSK6416C320 testkort.
(Bildet er hentet fra Code Composer Help File, og modifisert)

1. TMS 320C6416 DSP - DSPen fra Texas Instruments danner kjernen i testkortet, og har samhandling med de andre komponentene på kortet. Klokkefrekvens på 720MHz gir formidabel datakapasitet, med 8 32bits instruksjoner per klokkepuls. 1MB Cache minne gir rask tilgang til de mest brukte funksjoner. Innebygd EDMA kontroller gjør at prosesser kan kjøres, uten at det trengs prosessorkraft fra DSPen. Kommuniserer med eksternt minne gjennom EMIFA og EMIFB bussene, og med lydcodecen gjennom McBSPene.

2. CPLD Altera EPM3128TC100-10 - Gjennom CPLDen har vi via Code Composer tilgang til å kontrollere flere funksjoner på kortet, via enkel software programmering. Dette kan for eksempel være styring av LEDs på kortet, og programmering av brytere. Interface mot et eventuelt ekstern kort, såkalt datterkort, styres også via CPLDen. Den siste viktige funksjonen er bindelogikken CPLDen inneholder mot de andre komponentene på kortet. Den tar plassen til en del diskret logikk, som ellers måtte ha vært implementert, og står for globale reset funksjoner. Dette gir CPLDen en sentral rolle i kommunikasjonsmønstret på testkortet.

3. Flash Memory Am29LV400B – Koblet til DSPen gjennom den 16bits brede EMIFB bussen. Hastigheten på bussen er som for EMIFA 120MHz. Flash minnet gir 512Kbyte til rådighet, og brukes som et Boot minne for Code Composer. Data som ligger her går ikke tapt selv om minnet mister spenningen.

4. SDRAM MT48LC2M32B2 – Koblet til DSPen gjennom den 64bits vide EMIFA bussen. Hastigheten på kommunikasjonen er 120MHz, som er hovedhastigheten på 720MHz dividert på 6. Kortet har to RAM brikker, hver på 32MB. Lesing og skriving til minnet skjer via en burst funksjon, beskrevet i avsnitt 2.2.1, seksjon IV.

5. TLV320AIC 23 AudioCodec – Audio Codecen fra Texas Instruments styrer alle input og output som skjer med lyd signaler. Den inneholder egne AD og DA konvertere, slik at den er fullstendig klar for å motta signaler, og prosessere disse. Kommunikasjon med DSPen skjer gjennom McBSP2 kanalen, og er toveis. Dette innbærer at lydsignaler som er samplet kan sendes til DSPen for prosessering, bli sendt tilbake via samme kanal, konverteres tilbake til analogt signal, og spilles av. McBSP1 brukes til å sette opp Codecens registre, gjennom et Serial Peripheral Interface (SPI).
Data sendes serielt, med en klokkehastighet på 12MHz og ordlengde 16bit per lydkanal. Codecen støtter med andre ord lyd i to kanaler (stereo).

6. McBSP – Står for Multichannel Buffered Serial Port. Det er 3 McBSPer på kortet, hver med sin funksjon. Felles er at de benyttes for sending av serielle data i høye hastigheter, mellom eksterne enheter og DSPen. McBSP2 sender og mottar lyddata fra AIC23 AudioCodec. McBSP 1 mottar Codecens kontrollsignaler, og setter de riktige registrene. McBSP0 brukes mot de eksterne kontaktene, for å gi signaler fra disse til DSPen. McBSP2 og -1 kan også brukes mot eksterne kontakter, noe som skjer gjennom setting av registre i CPLDen.

7. Spenningsforsyning – Forsyner seg av +5V spenning fra en ekstern kontakt, og konverterer spenningen til +1.4V og 3.3V ved å bruke TPS54310PWP Buck spenningsregulatorer. +1.4V brukes i forbindelse med DSPens kjernelogikk, mens 3.3V brukes til samtlige andre komponenter på kortet. Power til eksterne kort gis fra samme forsyning. Som ekstra funksjoner er det en hel del testpunkter på DSK kortet, slik at spenning og strømmer kan sjekkes.

8. Eksterne kontakter

- **Expansion Memory Interface (EMIF):** Denne kontakten gir oss tilgang til deler av EMIFA bussen. Vi kan lese av alle adressesignalene fra DSPen, og få tilgang til 32bit av datasignalene. Siden kontakten bidrar med en memory interface, er den ideell for tilkobling av flere RAM brikker eller lignende enheter. Gjennom kontakten får også eksterne minneenheter tilgang til asynkrone kontrollsignaler direkte fra DSPen, og drivspenning i størrelsesorden +3.3V og +5V.
- **Expansion Peripheral Interface (EPI):** Gir mulighet for utvidelse med serielle enheter, for eksempel i form av en ekstra audio codec. Kommunikasjon med DSP skjer gjennom de aktuelle styresignalene fra McBSP0 og -1 som er tilgjengelige for kontakten. Andre nøkkelsignaler som er tilgjengelige er timere, eksterne interrupt signaler og reset signaler. Drivspenning som for EMIF kontakten.
- **Expansion Host Port Interface (HPI):** Gjør signaler fra McBSP2 tilgjengelige for et eksternt datterkort. DSPens HPI signaler, både for kontrollsignaler og datasignaler, er tilgjengelige. HPI er en parallell port, og er optimal for tilkobling av en ekstern prosessor, for eksempel en annen DSP. Selvfølgelig er det også spenningsforsynings muligheter på kontakten, tilsvarende som for EPI.

9. JTAG - En avansert testrutine som består av et JTAG interface og en JTAG Emulator. Disse gjør det mulig å debugge hardware på kortet gjennom Code Composer, og sjekke registrene til enheten fortløpende. JTAG kommuniserer direkte med Code Composer via USB tilkobling, og all setting av registre går gjennom dette interfacet, før det finner veien til hardware rundt omkring på kortet.

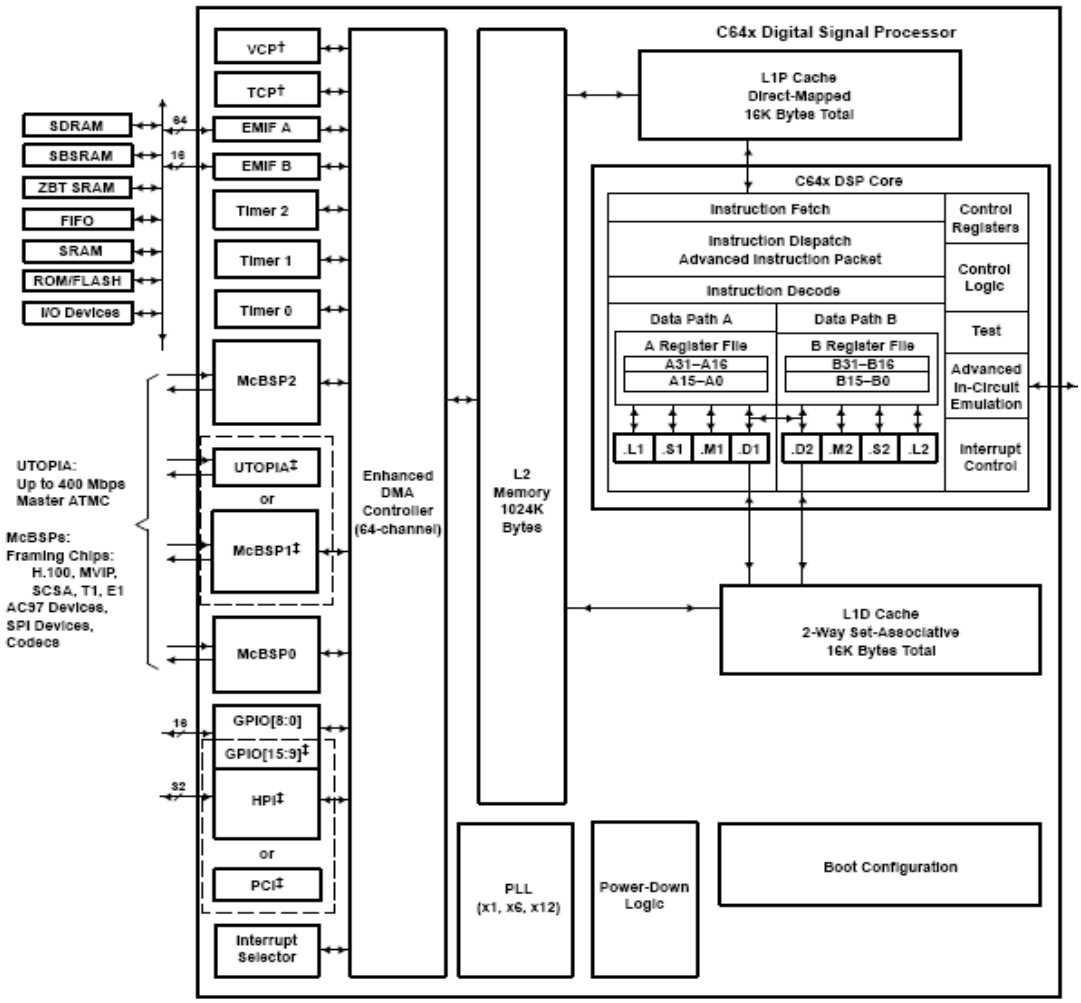
2.2 Utfyllende informasjon om kjernekomponenter

2.2.1 TMS 320C6416 DSP fra Texas Instruments

Etter den litt mer generelle presentasjonen av noen viktige funksjoner, under punkt 2.1,går vi litt mer i dybden på DSP`ens virkemåte. Enheten i seg selv er av høy kompleksitet, og den informasjonen som vil bli gitt i kommende avsnitt vil berøre viktig informasjon innen områdene:

I: DSP`ens kjernelogikk, og samspill mellom registre
II: Oppsett mot eksterne funksjoner, deriblant EDMA kontroller,
III: EMIFA (Eksternt minne i form av SDRAM)
IV: Lesing og skriving til SDRAM ved bruk av burst funksjon
V: EMIFB (Ekstern minne i form av Flash)

For øvrige timing diagrammer av både eksterne og interne enheter, refereres det til databladet for TMS 320C6416.



Figur 2.2.1 (1) Blokkskjematisk fremstilling av DSPen.
(Bildet er tatt fra databladet for TMS320C6416 DSP)

I:

For å beskrive oppførselen til TMS320C6416, er det naturlig å starte med registre og funksjoner knyttet til selve kjernen i prosessoren. Selve kjerneteknologien kalles velociTI. Denne benytter seg av en VLIW [Very Long Instruction Words] teknologi, noe som gir muligheten til å utføre mange instruksjoner for hver klokkepuls. For 64x serien kan vært instruksjonsord være på totalt 256bit, noe som gir rom for 8 32 bits instruksjonsblokker per klokkepuls. Alle de 8 blokkene trenger ikke være fylt med instruksjoner, da de kan eksekveres uavhengig av hverandre, og på forskjellige klokkepulser. Det første bitet i hver blokk bestemmer hvor mange av blokkene som har sammenhengende instruksjoner, og dermed skal eksekveres på samme klokkepuls. Det er også unikt for 64x serien at instruksjonene ikke trenger å oppfylle alle de reserverte 32bits plassene for å utføres. Dette fører selvfølgelig til en betydelig total besparelse i minnet som benyttes til å utføre instruksjoner. Den totale kapasiteten for aritmetiske operasjoner er som følge av dette enorm, og ligger i den funksjonsgruppen som DSPen er spesiallaget for. 5760 instruksjoner pr sekund med 720MHz klokkefrekvens sier det meste på dette punktet.

Fig. 2 viser innedelingen av kjernen i databanene A og B. Blokkene markert L,.M, .S, og .D er de funksjonelle blokkene. Disse eksisterer på begge de to databanene, sammen med 32 32 bits registre for hver bane. Hver av de 4 funksjonelle blokkene har tilgang til disse registrene, i tillegg til registrene for den andre databanen, via en databuss. Slik sikres enkel tilgang av alle registrene, for samtlige funksjonelle enheter.

Data adresseringsblokkene .D1 og .D2 har ansvaret for all dataoverføring mellom de ulike registre, og minnet. I grunnversjon støtter de opp til 32bit størrelse på overført data pr. instruksjon, men øvrig oppgradering til velociTI.2 teknologi gir støtte for 64bit overføring pr. instruksjon.
Multiplikasjonsblokkene .M1 og .M2 utfører to 16x16-bits multiplikasjoner per klokkepuls, eller fire 8 x 8-bits multiplikasjoner. Addisjons og logikk blokkene .S og .L utfører aritmetiske og logiske operasjoner. Tilgjengeligheten er ved hver klokkepuls.

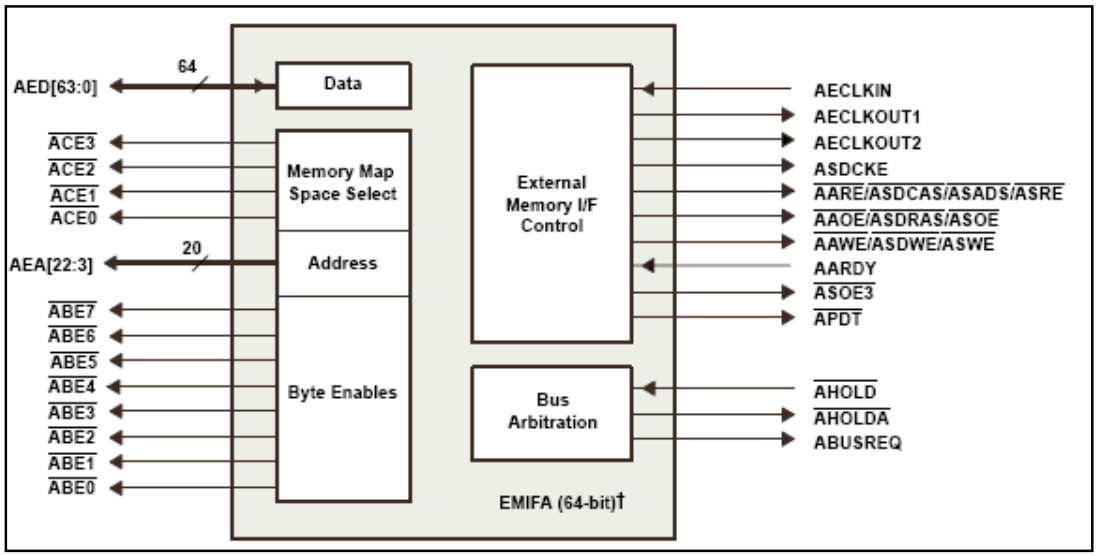
Alle instruksjoner starter når en 256 bits pakke, såkalt fetch pakke, blir hentet fra program minnet, med 32bit tilgjengelig for hver funksjonsblokk. Som nevnt kan det eksekveres 8 32bits pakker pr. klokkepuls, og pakker som skal eksekveres på samme klokkepuls, blir linket sammen via LSB. Det er også LSB som bestemmer hvilke pakker som skal sende videre til neste klokkepuls. Når 8 av 32bits pakkene er buntet sammen, og de inneholder instruksjoner fra funksjonsblokkene, blir de fylt inn i fetch pakken. Siden denne nå inneholder et konkret sett med instruksjoner, bytter den navn til execution pakke. Neste 256bits fetch pakke sendes ikke før den execution pakken har sendt ut alle sine 32 bits instruksjonspakker til de riktige adressene.

II:

EDMA (Enhanced Direct Memory Access) kontrolleren gjør DSPen i stand til å håndtere mer komplekse datasett, og gir samtidig muligheten for å koble prosesser utenom DSPens prosessor. EDMA kontrolleren overvåker trafikk i alle kanalene den er tilkoblet til, og skal skille mellom prioriteringer slik at datatrafikken flyter. Dette gjøres ved to prosedyrer som kalles preemption og pipelining. Preemption får vi når EDMA kontrolleren sender et interrupt for å stoppe en forsendelse med lav prioritering, til fordel for en med høy, og så overføre resten av den avbrutte sendingen etterpå. Dette gir data fra busser med høy hastighet prioritet foran lavhastighetsbusser, slik at disse ikke skal sinke systemet. Et eksempel kan være at data fra EMIF bussene prioriteres foran data fra den tregere Audio Codecen, som benytter seg av McBSP2. For ikke å overbelaste bussen med interrupt, har EDMA kontrolleren mulighet for pipelining. Mens en forsendelse foregår, sendes informasjon om den neste i en egen pipeline, slik at denne er klar til å starte umiddelbart etter at den første forsendelsen er ferdig. Da trengs det ikke noe reset tid etter interruptet, og data hastigheten øker betraktelig. EDMA kontrolleren er ikke direkte koblet til kjernen i DSPen, slik at den kan programmeres som en hvilket som helst ekstern enhet. For vårt testkort er den selvfølgelig ferdig programmert av Texas Instruments, for å fungere optimalt.

III:

Siden DSP kun er utstyrt internt med 1Mb Cache minne, trengs det ekstern RAM for innlegging av kode, og andre funksjoner som krever minneplass. Dette kan være lagring av sampler fra Audio Codecen, samt avanserte prosesseringsalgoritmer. Aksessen til SDRAM fra DSP gjøres via EMIF(External Memory InterFace) bussen. EMIF genererer buss-signalene via en programmerbar kontroller. Oppsettet av kontrolleren gjøres utelukkende i software, og det kan ikke foregå noen kommunikasjon over bussen før de rette registrene er satt opp via Code Composer. I praksis betyr dette at hvis kode skal legges over på SDRAM, så må de rette registrene for EMIF settes først, og buss kommunikasjon etableres.



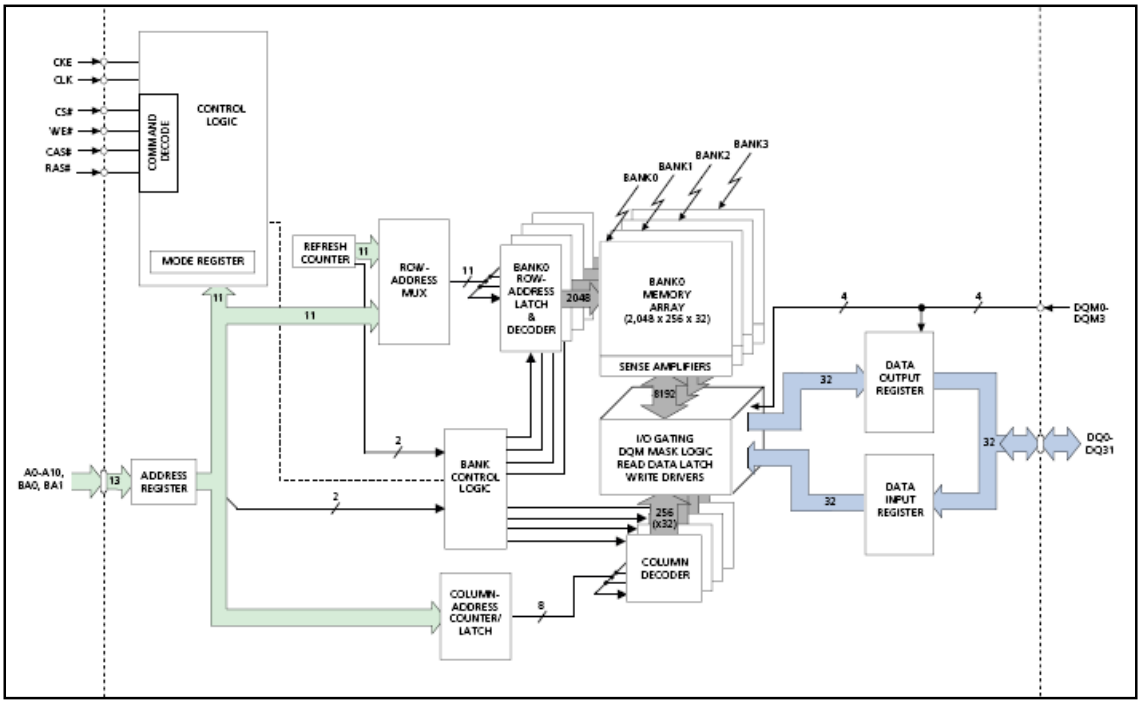
Figur 2.2.1 (2) Signalering mellom DSPen og EMIFA bussen
(Bildet er tatt fra databladet til TMS320C6416)

Resten av dette avsnittet skal dedikeres til noen av kjernefunksjonene og registrene som må settes opp for å få kommunikasjon over bussen til å fungere. Det vil også foreligge forklaring på de signalene som fungerer dynamisk mellom DSP og SDRAM, for å velge riktige minnesektorer, oppsett av burst for lesing fra/skriving til RAM og andre nøkkelfunksjoner.
Fig. 2.2.1 (2) viser en overordnet oversikt over de signalene som stilles til rådighet fra EMIFA kontrolleren på DSP. Disse signalene styrer i sin helhet kommunikasjon og oppsett med SDRAM. Tabellen under gir en oversikt over hvilke signaler som styrer hvilke funksjoner på den eksterne RAM brikken. Det er viktig å påpeke at dette kun er en oversikt over samspillet mellom brikkene, og ikke en detaljert beskrivelse med pinnenummer og andre detalj data. For disse refereres det til databladet for de enkelte komponenter.

Pinne(r) DSP	Pinne(r) SDRAM	Funksjon
AECLKOUT1	CLK	Klokken for SDRAM drives av DSP systemklokken. AECLKOUT1 gir ut en 120MHz klokkefrekvens.
ASDCKE	CKE	CLK Enable. Ved logisk høy aktiveres klokkesignalet, og ved logisk lav slås klokkesignalet av.
ACE0	CS	Chip Select. Denne er aktiv lav, og slår av kommandosignal dekoderen når den går høy. Er pinnen registret lav, er alle kommandosignaler på
SDWE,,SDCAS,SDRAS	WE,CAS,RAS	Kommandosignal er, som blant annet bestemmer read/write sekvenser, burst sekvenser og refresh sykluser. Nærmere bruk av disse kommer fram av eksempler på oppsett av viktige kommandoer.
ABE[0..7]	DQM[0..3]	DQM er Input\Output maske for RAMen. Hver av DQM inngangene styrer sin Data I\O seksjon på brikken. Dette kan være DQM0 styrer read/write for I\O seksjon DQ0-DQ7. Slik styrer også de andre DQM input pinner hver sin I\O sektor. De går under Byte Enable blokken for EMIFA.
AEA[14..15]	BA[0..1]	Styrer hvilken bank på RAMen ønsket kommando eksekveres. Hver brikke er delt opp i fire banker det kan velges mellom.
AEA[3..13]	A[0..10]	RAMens adresselinjer. Adressen leses inn når kommandoen aktiv er lagt inn på de ulike kommandosignaler. Da leses rad i aktiv minnebank(Gitt av B0 og B1) inn. Adressen leses også inn når read/write kommandoen er gjeldene, men da for kolonne i den raden som ble merket med aktiv kommandoen.
AED[0..31]	DQ[0..31]	RAMens Data I\O pinner.

Tabell 2.2.1 (1) Samspillet mellom DSP og SDRAM, på signal nivå.

IV:



Figur 2.1.1 (4) Internt RAM oppsett, med oversikt over kontrollsignaler og minnebanker
(Figuren er hentet fra databladet til MT48LC2M32B2)

Før lesing og skriving med burst funksjonen kan initieres, må RAM brikken sette i aktiv status. Det er i aktiv status at adressebussen EMIFA inneholder de data som bestemmer hvilken minnebank det skal leses fra, og hvilken rad i minnebanken som skal være aktiv. Det er viktig å ha i bakhodet at aksess til minneceller i RAMen skjer som en todimensjonal array, der det først velges rad, og så kolonne i valgt rad. Både kolonne og rad må være lokalisert i den samme minnebanken. Prinsippet illustreres godt av Fig. 2.2.1 (4).

Disse betingelsene gjør at for hver lesing og skriving som skal utføres, må det utføres to initialiseringsoperasjoner, så lenge det ikke er samme minneområde det skal leses fra. Aktiv kommandoen som skal settes opp først kan illustreres med følgende eksempel:

NAME (FUNCTION)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQs
ACTIVE (Select bank and activate row)	L	L	H	H	X	Bank/Row	X

Tabell 2.2.1 (2): Oppsett av aktiv kommando

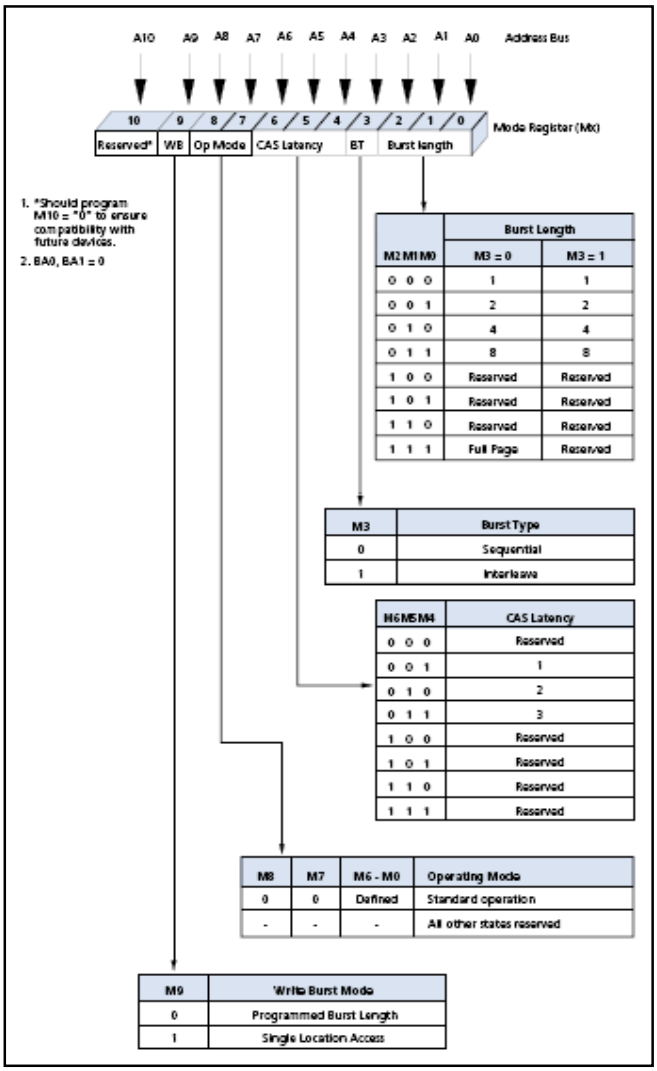
Det går tydelig fram av tabell 2.2.1 (2) hvordan de fire kommandosignalene skal settes opp, og det er derfor koblingen til adressebussen som er det vesentlige her. Som nevnt i Tabell 2.2.1 (1) så er det BA0 og BA1 som bestemmer hvilken minnebank som er aktiv, og adressebitene A0-A10 som bestemmer hvilken rad i aktiv minnebank det skal leses fra når les/skriv kommando initieres. Siden minnet kan sees på som en todimensjonal array, trengs det også en kolonne adresse. Denne adressen bestemmes av adressebitene A0-A10 når kommandoaignalene angir en les- eller skriv-kommando. Adressebitene setter altså både opp rad og kolonne adresse, avhengig av hvor i prosessen vi er.

Først må det imidlertid settes opp hvilke opsjoner som er aktuelle for den burst sekvensen som skal gjennomløpes ved lesing/skriving til minnet. Disse opsjonene settes ved konfigurasjon av Mode Registret. Åpning av Mode Registret skjer ved å legge inn bitmønstret fra tabell 2.2.1 (2) på kommandosignalene.

NAME (FUNCTION)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQs
LOAD MODE REGISTER	L	L	L	L	X	Op-Code	X

Tabell 2.2.1 (3): Bitmønster laster inn mode registret.

Det er viktig at det ikke foregår andre operasjoner på minnebankene mens Mode Registret skal settes, siden Mode Registret gjør bruk av adressebussen, og endrer på de burst innstillingene som ble brukt i forrige les/skriv operasjon. I bruk sammen med Mode Registret har adressebitene A0-A10 andre funksjoner enn å angi en spesifikk rad/kolonne adresse i en valgt minnebank. Funksjonen av bitene kan sees i sammenheng med Fig. 2.2.1 (5), på neste side.



Figur 2.2.1 (5): Adressebitens funksjon ved burst oppsett (Bildet er hentet fra databladet til til MT48LC2M32B2)

Burst basert lesing/skriving fungerer i prinsippet ved at bearbeiding av data skjer innenfor en valgt minneblokk, det såkalte burstområdet. Den nevnte Aktiv kommandoen velger minnebanken det skal opereres på, og hvilke rader det er aktuelt å lese fra. Når så Les/skriv kommando blir initialisert, angis det hvilken kolonne det skal leses fra. Mode Registrets rolle her, er å velge startkolonne, og hvor mange kolonner som skal være med å danne en minneblokk. Denne minneblokken er det minneområdet det vil bli utført burst basert lesing fra/skriving til. Fig. 2.2.1 (4) angir hvilken funksjon de ulike adressebitene har under oppsett. Hvis det skal leses fra to-kolonner, angir A0(LSB) startkolonne, mens A1-A7 angir en unik blokk med kolonnene som valgt burst funksjon vil operere på. Ønskes det lesing fra 4-kolonner velges dette med A0 og A1, mens A2-A7 angir unik minneblokk burstet opererer innenfor. A0-A3 velger 8-kolonners burst, og A4-A7 gir unik minneblokk.

Neste steg for å aksessere riktig minneceller, og utføre enten skriving eller lesing av data, er å legge inn les/skriv kommandoen. Også her gjør en liten tabell forklaringen lettere.

NAME (FUNCTION)	CS#	RAS#	CAS#	WE#	DQM	ADDR	DQs	NOTES
READ (Select bank and column, and start READ burst)	L	H	L	H	L/H[8]	Bank/Col	X	4
WRITE (Select bank and column, and start WRITE burst)	L	H	L	L	L/H[8]	Bank/Col	Valid	4

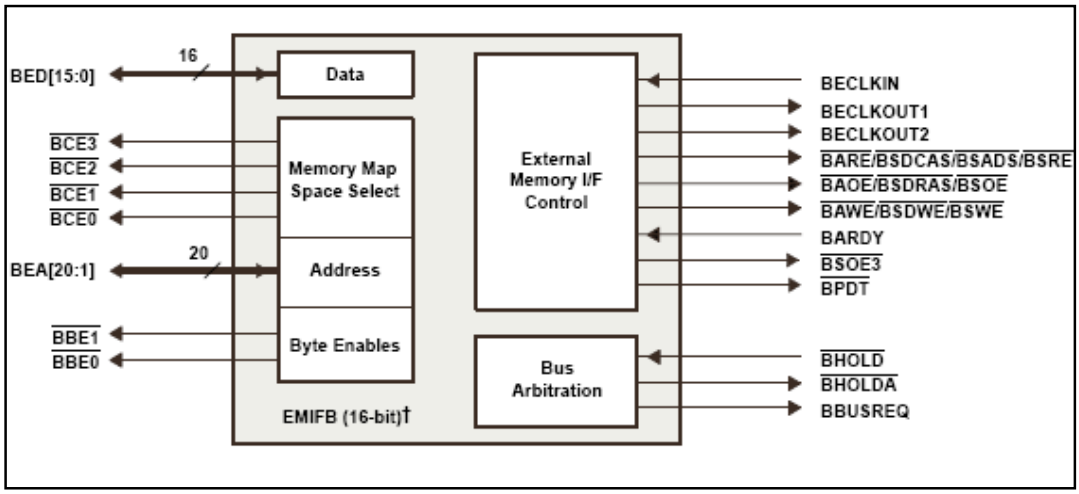
Tabell 2.2.1 (4): Konfigurering av les/skriv kommando

Eksempelvis kan en tenke seg at kommandosignalene angir at en les kommando skal benyttes.

Hva slags type burst som skal benyttes er satt opp i Mode Registret, så adressebitene har nå som funksjon å angi kolonneadresser. Det er A0-A7 som angir adresse, mens A10 betjener en annen viktig funksjon, nemlig ”precharge”. Etter at en aktiv kommando har blitt tilgitt en rad, vil raden være åpen helt til et ”precharge” signal sendes. Inntil dette signalet forkommer, vil det ikke være mulig å endre rad i den aktuelle minnebanken for les/skriv. Ved å sette A10 ved les/skriv kommando, vil raden automatisk lukkes, slik at en ny rad kan åpnes i neste les/skriv sekvens.

Dette skulle være en kort innføring i oppsett og bruk av SDRAM, i samarbeid med EMIFA fra DSP. Standardoppsett av RAMen, såkalt oppstartsmodus, initialiseres automatisk av Code Composer. Det er kun opp til brukeren og sørge for at de rette signaler sendes fra DSP, og inn på RAMens interne registre. Når dette oppsettet gjøres riktig, vil EMIFA bygge opp et flytende grensesnitt mellom DSP og SDRAM.
For ytterligere dyptgående registeroppesett refereres det til databladet for til MT48LC2M32B2, og TMS 320C6416 DSP.

V:



Figur 2.2.1 (6): Signalering mellom DSP og EMIFB

De signalene som er gjort tilgjengelige for Flash minnet er visualisert ved Figur 2.2.1 (6). Hvordan signalene fra DSP styrer funksjoner sammen med Flash illustreres best med en signaltabell. Tabellen tar utelukkende for seg navnet på signalene som kummuniserer, og ikke reellt pinnenummer på den enkelte brikke.

Pinne(r) DSP	Pinne(r) Flash	Funksjon
BEA2 – BEA20	A0-A17	Adresse bit
BED0-BED7	DQ0-DQ14	Input/Output databit
BEA1	DQ15	Word/Byte mode for input/output data
DGND	BYTE#	Byte/Word mode
BCE1, B_AOE, B_AWE	CE#, OE#, WE#	ChipEnable, OuputEnable, WriteEnable
BRD_RST#	RESET#	Hardware reset, aktiv lav
NV	RY/BY#	Ready/Busy# output

Tabell 2.2.1 (5): Samspill mellom DSP og Flash på signalnivå.

DSPen bruker Flash minnet som boot seksjon, i den betydning at vitale funksjoner lagres unna her. Bakgrunnen for dette er at flash minnet er non-volatile, som betyr at det ikke mister innholdet i minnet ved tap av drivspenning. Det er 8bits interface mellom Flash minnet og EMIFB.

2.2.2 AIC23 Audio Codec – Kommunikasjon og konfigurering

Audio Codecen er en vital del av testkortet for vår del, siden den er involvert i all prosessering av lydsignaler til og fra testkortet. Kommunikasjon med DSPen foregår via to McBSP kanaler, en kanal for kontroll signaler og en for data signaler. Kontrollsignalene blir styret av McBSP1, og inneholder interne klokke- og synkroniseringssignaler. McBSP2 styrer de signalene som omhandler datatrafikk mellom AudioCodec og DSP, noe som vil si datatrafikk i begge retninger. Styringen foregår i MasterMode, med klokkefrekvens 12MHz. At Codecen kjører i master mode betyr kort fortalt at det er den selv som bestemmer når frames skal sende og mottas. Det er med andre ord Codecen som sender initieringsignaler til DSPen, og ikke motsatt, som skjer i Slave Mode.

Resultatet av en lydsampling blir to arrayer med 16bit i hver, en for hver lydkanal. Til sammen danner disse en frame på 32 bit, som prosesseres av DSPen. Default oppsett på testkortet 320C6416 DSK er 16bit pr. Kanal. Samplingen av de analoge signalene kan reguleres mellom 8kHz og 96kHz, alt etter hvor god sampling som ønskes. For vårt prosjektformål kommer vi til å sette samplingen til 48kHz. Dette skal være mer enn godt nok for å ekstrahere den informasjonen vi trenger ut av et talesignal. Codecen inneholder som følge av kommunikasjon begge veier med DSPen, både ADC og DAC, slik at audiosignaler også kan spilles av igjen etter prosessering hos DSP. Både kanalen for kontrollsignaler og for datasigneler er serielle.
En bedre beskrivelse av Codecen kan hentes ut av fig. 4 hvor også de mest aktuelle registrene er vist.

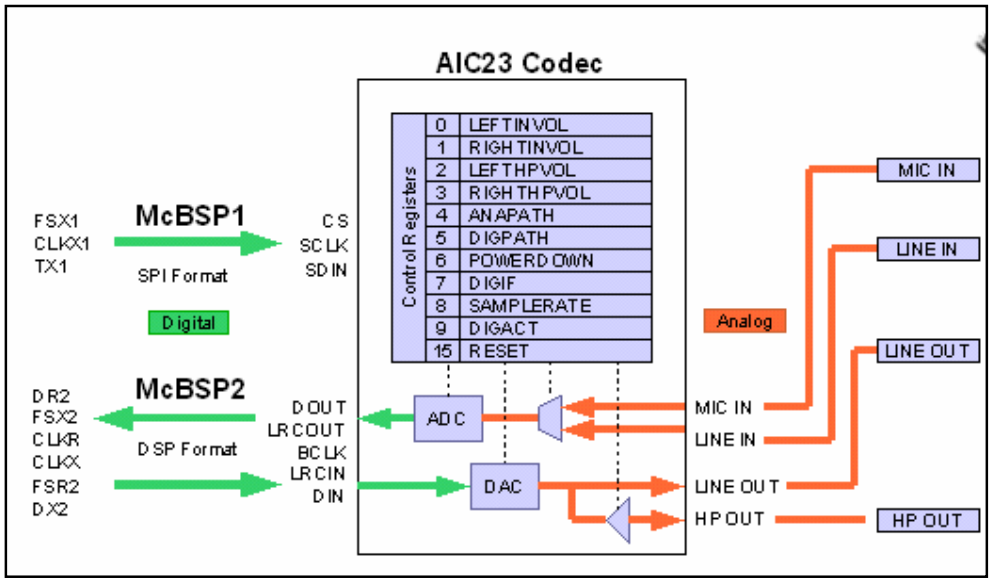


Fig. 2.2.2 (1): Oversikt over kommunikasjon signaler for AIC23 Audio Codec.
(Bildet er hentet fra hjelp filen til testkortet,under AIC23 seksjonen)

Mikrofon innganen Mic IN er den interessante for prosjektets formål. Inngangsforsterkningen kan justeres opp til 5 ganger forsterkning, og det kan i tillegg benyttes en mikrofonforsterker som har forsterkning mellom 0- og 20db. Under bruk til lyd formål hender det at oversampling av signalene kan være aktuelt. Codecen støtter industristandard verdier for oversampling, i tillegg til noen spesielle oversamplingsrater, tilpasset samplingsfrekvens til vanlige CD plater. Både vanlig Audiosampling(48kHz) og sampling i CD kvalitet (44.1kHz) støttes ved de spesielle oversamplingsratene $250f_s$ og $272f_s$. Disse ratene gir heltallige divisjoner med systemklokken.

Kontrollsignalene som styres av McBSP1 brukes til å sette de aktuelle registrene i Codecen. SPI kommunikasjonen muliggjør kun kommunikasjon i én retning, slik at setting av registre er den eneste opsjonen. Det er med andre ord ikke mulig å få tilgang til de data som ligger på registrene, gjennom kontrollsignalene.

Selve settingen av registrene gjøres ved å klokke inn et dataord på 16bit, der de 7 første bitene bestemmer hvilke register som skal settes, og de siste 9 bit bestemmer hvilke data registret skal

inneholde. Overføringen med SPI er veldig enkel. Den initialiseres av av Chip Select settes til aktiv, CS er aktiv lav, så klokkes det inn ett bit per klokkepuls. Et timing diagram illustrerer poenget bedre.

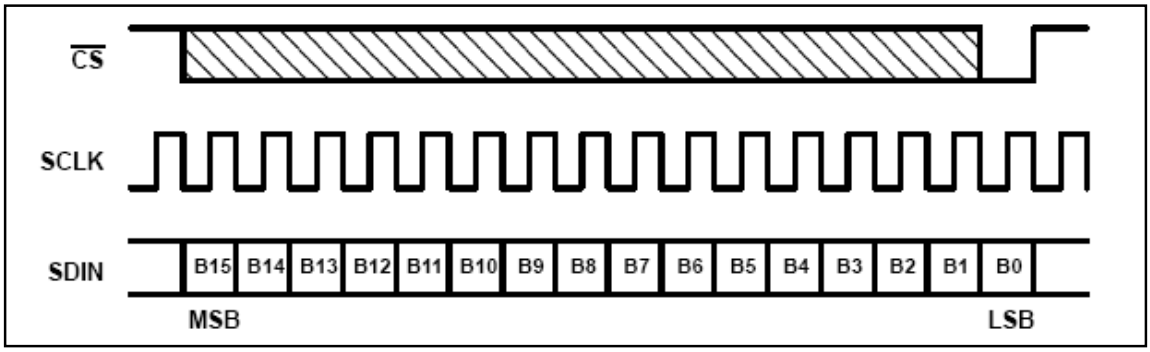


Fig. 2.2.2 (2): Timingdiagram for sending av data i SPI modus
(Figuren er hentet fra databladet til AIC23)

For ordens skyld kan det være greit med en oversikt over hvilke signaler fra McBSP som kommuniserer AIC23 sine kontrollsignaler:

McBSP 1	AIC23 Codec
FSX	LRCIN
DX	BCLK
CLKX	DIN

Alle data som skal fungere for konfigurasjon av registre sendes gjennom DX1. Disse bitene konfigureres software messig gjennom Code Composer. Når data sendes er det LSB som sendes først, og det CS som danner rammen for dataordet, ved å angi stopp og start for sending, se fig. 2.2.2 (2). All konfigurering av hvilket modus McBSP og AIC23 skal fungere i, må settes software messig. Konfigureringen gjør at pinnene som er aktuelle ut fra DSP og McBSP for kommunikasjon settes automatisk av programmet. Den eneste brukerinputen som trengs er bestemmelse av konfigureringsordet som skal sette registrene, med riktig adresse og databit. Bitene fylles inn og legges direkte inn i Code Composer koden.

Det kan legges til at kontrollsignalene også kan sendes i I2C modus, selv om dette er en mer uvanlig konfigurasjon, og ikke blir benyttet i vårt testoppsett. CS bestemmer nå hvilken adresse som sendes ut fra AIC23, og det er enheten som responderer på adressen som starter overføring av data. Dette gjøres ved at SDIN settes lav, mens systemklokken SCLK er høy. Synkroniseringen her er automatisk, og AIC23 opererer i slave mode, så lenge sending forgår i I2C modus. Bytting mellom SPI og I2C modus foregår enkelt ved å sette fast potensial på Mode pinnen til AIC23. Jord for I2C, og +3.3V for SPI modus.

En like vital del av Audio Codecen, er data kanalen. Det er denne som kommuniserer med DSPen, via en bi-direksjonell kommunikasjonskanal. All prossesering av audio signaler foregår her, og styringen av signaler skjer via McBSP2. Software muliggjør forskjellige innstillinger for dataformat gjennom kanalen, men på vårt kort er det DSP mode som er satt som standard, for å få et flytende grensesnitt med gjeldene McBSP konfigurering.

En forklaring på hvordan transaksjon av data foregår, blir enklere om vi først ser litt på hvilke signaler som spiller sammen, mellom McBSP og AIC23:

McBSP 1	AIC23 Codec
FSX	LRCIN
FSR	LRCOUT
CLKX	BCLK
DX	DIN
DR	DOUT

Overføring av data over kanalen styres av signalene LRCIN og LRCOUT, hvor LRCIN er for audio input, og LRCOUT er for audio output. En høy-til-lav puls på en av disse bitene gjør at dataoverføring initieres, i den retningen som er valgt. Dataordet som overføres er på totalt 32 bit, 16 for hver høyttaler kanal. Innklokking av ordet og synkronisering av systemklokken initieres automatisk, når codecen er innstilt på mastermode. Som for kontrollsignalene og McBSP1, stilles også datasignaler og McBSP2 inn software messig. For nærere informasjon henvises det til hjelpfilen for codecen.
Overføringen i Master mode er gitt av følgende tidsdiagram:

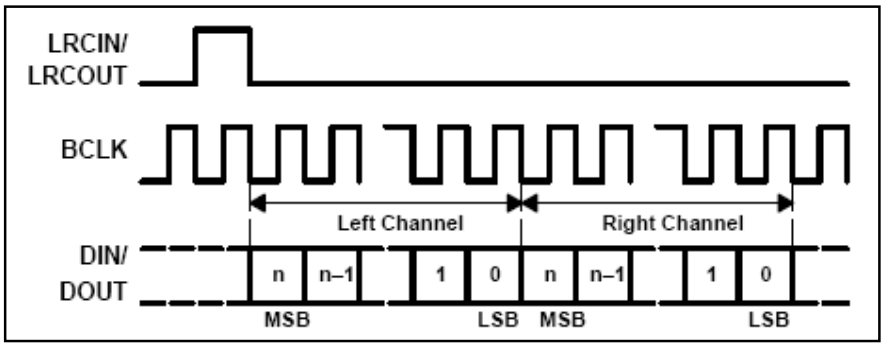


Fig. 2.2.2 (3): Initialisering av datasending mellom DSP og AIC23 i Master Mode. (Figuren er hentet fra databladet til AIC23)

AIC23 inneholder 10 registre, som det mulig å sette via kontrollsignalene på McBSP1. For å gi en rask oversikt over de aktuell registrene, og overfladisk funksjonalitet, vil tabell 2.2.2 (1) være relevant:

Adresse	Register navn og funksjon
0000000	Audiokontroll for venstre lydkanal
0000001	Audiokontroll for høyre lydkanal
0000010	Audiokontroll for venstre hodetelefonkanal
0000011	Audiokontroll for høyre hodetelefonkanal
0000100	Analog audiobane kontroll
0000101	Digital audiobane kontroll
0000110	"Power-Down" kontroll
0000111	Format til digitalt audiointerface
0001000	Protokoll for justering av samplingsrate
0001001	Aktivering av digitalt interface
0001111	Register for reset funksjon

Tabell 2.2.2 (1): Konfigurerbare registre for AIC 23, med dataadresser

I kolonnen som angir adressebitene, er de 7 første bitene som skal sendes med i kontrollordet på totalt 16bit. De neste 9 bitene er databitene, som angir registrets innstilling. For oversikt over mulige data kommandoer for hvert register refereres det til databladet for AIC23, og til hjelp filen for testkortet. Setting av registrene i SPI mode er forklart et par avsnitt ovenfor.

2.2.3 McBSP – Multichannel Buffered Serial Port

McBSPene er selve sentralen for kommunikasjonsstyring mellom DSPen og flere av de eksterne kjernekomponentene på kortet. Kommunikasjonen foregår i full duplex, med doble databuffer på registrene. Slik kan sending og mottak av data foregår kontinuerlig, i 128 ulike kanaler.
I utgangspunktet er konfigurasjonen slik at McBSP1 og -2 kommuniserer med AIC23 Codec, se for øvrig avsnitt 2.2.2, mens McBSP0 kommuniserer med de eksterne kontaktene, kommentert i avsnitt 2.2.5. Dette oppsettet er standard fra Texas Instruments. McBSP 1 og 2 kan som alternativ settes opp for ekstra kommunikasjon med eksterne kontakter, via MISC registret i CPLDen. CPLDen med tilhørende viktige registre er presentert i avsnitt 2.2.8.

Felles for alle McBSP er oppsettet med både en data kanal og en kontroll kanal, jfr. koblingen til Audio Codecen. Kommunikasjonen med DSPen skjer gjennom en 32bits bus, Internal Peripherl Bus, hvor setting av kontrollregistre internt i McBSPen bestemmer operasjonene. Det er likevel bare 7 signalpinner som kobler hver McBSP til sin hardware. Dette er signaler for klokkesynkronisering, og sending/mottak av data og frames. For et eksempel på funksjonen av disse signalpinnene, refereres det til avsnitt 2.2.2, og samspillet med AIC23.

2.2.4 Expansion Memory Interface

Memory Interfacet gjør det mulig å koble til en ekstern minneenhet hvis dette er ønskelig. Konfigureringen skal i utgangspunktet fungere smidig ved tilkobling av for eksempel mer RAM, eller enheter konfigurert for å jobbe opp mot minnefunksjoner. Det er imidlertid et poeng at hvis ekstra minne skal fungere, må det være et asynkront minne. Dette for å kunne benyttes sammen med det asynkrone minnet på EMIFA bussen. Alle de eksterne adresselinjene fra DSPen er også tilgjengelig, det vil si de samme adressene som konfigurerer minnetilgangen på EMIFA. En eksternt tilkoblet minneenhet vil derfor kunne styres like godt av DSPen, som minnebrikkene på kortet. Dette gjelder også i henhold til timing for dataoverføring, i og med at kontakten gir full tilgang til de asynkrone timingsignalene som genereres av DSPen. Spenningsforsyning er sikret gjennom kontaktpinner for +3.3V, +5V og jord.

2.2.5 Host Port Interface

Mens Peripheral Interface gir tilgang til McBSP 0 og 1, gir Host Port Interface tilgang til McBSP2 sine konfigureringssignaler. HPI er en parallell port som gjør at en ekstern prosessor kan få tilgang til DSPens minne område. Den mest essensielle funksjonen for HP interfacet er HPI-bussen, som gir tilgang til alle HPI data på DSPen. Bussen sikrer rask dataoverføring mellom DSP og HPI, fra Host Port Expansion kontakten. Dataoverføringen blir støttet av en del kontrollsignaler fra HPI til DSP, som gjør det mulig for forskjellig typer prosessorer og kunne benyttes som datterkort. Alle kontrollsignalene overføres asynkront .
HPI har tilkobling til +3.3V, +5V og jord for alle datterkort, samt en HPI_reset# funksjon, som gjør at ethvert datterkort tilkoblet HPI, kan resette DSPen direkte.

2.2.6 Expansion Peripheral Interface

Gjør de signalene DSPen disponerer til perifere enheter, altså tilkoblede datterkort, tilgjengelige. I hovedsak er det snakk om signalene fra McBSP-0 og -1, noe som tilsier at det ligger tilrette for tilkobling av ekstra Audio Codecer og lignende enheter. Skal datterkortet fungere i kontakten uten ekstra logikk, må det ha et serielt interface. Se for øvrig avsnitt 2.2.2 for McBSP1 sitt samarbeid med codecen. DSPen timere er tilgjengelige, sammen med eksterne interrupt signaler og reset signaler. Reset signalet kan opereres direkte fra CPLDen, og programmeres via denne. Klokkesignaler for synkronisering med DSPen, samt power betingelser som for HPI, er også tilgjengelige.

2.2.7 JTAG Interface og JTAG Emulator

JTAG interfacet og emulatoren er en av de viktigste delene på vårt testkort. I prinsippet er JTAG interfacet et sett med design regler som omfatter testing, programmering og debugging av mikroprosessorer.
I praksis betyr det at vi kan teste vitale funksjoner på kortet, ved bare å ha tilgang til en enkel kontakt, som kommuniserer med det eksterne JTAG interfacet. Det er med andre ord mulig å få direkte kontakt med signaler og registre på en brikke knyttet til testkortet. Debugging skjer gjennom JTAG emulatoren, som er en hardware brikke på kortet. Code Composer bruker JTAG emulatoren til å debugge kode direkte opp mot den mikroprosessoren eller komponenten koden er rettet mot. Noen av fordelene med å benytte en JTAG emulator på et utviklingskort:

- Det er mulig å plukke ut hardware på kortet, og debugge dette som er ferdig produkt, uten at det trengs noen ekstra testprogrammer. JTAG gjør det mulig å legge inn kode på ønsket hardware, og teste funksjoner og registre. Denne funksjonsmåten av JTAG er den vi benytter oss av på vårt testkort. Debugging skjer gjennom Code Composer, via USB kommunikasjon, og inn i aktuell hardware.
- Fleksibilitet og hastighet er nøkkelord når vi snakker om JTAG. I stedet for å skrive en kode i et eksternt program, og så legge denne inn i en EEPROM som kobles til hardware uten å egentlig vite hva koden gjør på forhånd, kan koden lastes direkte inn i JTAG emulatoren. Dette muliggjør sjekking av de interne registrene på hardware, samtidig som koden kjøres, slik at eventuelle feil kan rettes opp med en gang. Oversikten over registrene er tilgjengelig fra Code Compser, noe som gjør dette til en raskere prosedyre enn å brenne koden inn i separate EEPROM brikker, og teste .

Selv om JTAG interfacet gir store fordeler når det gjelder debuggingsmuligheter og god oversikt over aktuelle register, er det en del begrensninger når det gjelde de utviklingsmuligheter vi har med dette systemet.

2.2.8 CPLD

CPLDen er en slags alt-mulig komponent på kortet. De funksjonene den omhandler går mest på kontroll av andre enheter, ved å sette noen av de interne registrene. Fordelen med å bruke en CPLD i en sånn posisjon, er konfigureringsmulighetene. Kilde koden er skrevet i standard VHDL, og kan for eksperter innen programmeringsspråket muligens endres utover standardoperasjonene. CPLDen gir oss tilgang til fire programmerbare registre. Vi kommenterer bare de to som er mest relevante for oss, og de funksjonsbehovene vi behøver av teskortet. Registrene heter USER_REG og MISC. De er memory mapped, noe som betyr at vi har tilgang til å manipulere disse gjennom Code Composer. Siden et av registrene er særdeles viktige for hvordan vi kommuniserer med LONWorks nettverket. I denne omgang blir det kun en overfladisk presentasjon.

Index	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
0	USER_ REG	USER_ SW3 R	USER_ SW2 R	USER_ SW1 R	USER_ SW0 R	USER_ LED3 R/W 0 (Off)	USER_ LED2 R/W 0 (Off)	USER_ LED1 R/W 0 (Off)	USER_ LED0 R/W 0(Off)
6	MISC	MCBSP2_ EN R 1 (MCBSP2 enabled)	SCR_4 R/W 0	SCR_3 R/W 0	DSPPLL_ SELECT R 1	DSPPLL_ ENABLE R 1	FLASH_ PAGE R/W 0 (A19=0)	MCBSP2 ON/OFF BOARD R/W 0 (ONBOARD)	MCBSP1 ON/OFF BOARD R/W 0 (ONBOARD)

Tabell 2.2.8 (1): Konfigurasjonsbit for USER_REG og MISC

USER_REG tar for seg input fra DIP switchen på kortet, og funksjonen av lysdiodene. Ved å bruke Code Composer kan vi sette diodenes tilstand manuelt, eller kjøre sekvenser på disse ved brukerinput på DIP switchen. MISC styrer funksjoner for McBSP2 og 1, for eksempel om de skal styre sine vanlige funksjoner mot Audio Codecen, eller om de skal resettes for å styre et datterkort.

Utenom registerfunksjonene har CPLDen virkemåte som logiske kretser. Ved å programmere inn enkle logiske funksjoner i VHDL koden, spares det plass på kortet. Logikken binder hardware på kortet sammen, og kalles derfor bindelogikk. Flere av de reset funksjonene komponentene har, styres fra CPLDen. Reset til datterkort kan også styres, hvis kortet er koblet til Peripheral Expansion kontakten. Manuell reset av testkortet via en resetbryter kjøres også gjennom CPLDen. Kort sagt har CPLDen en sentral rolle i all kommunikasjon som skjer mellom komponenter på testkortet.

2.3 Kommentarer til skjematiske tegninger

2.3.1 Introduksjon

En del av produktspesifikasjonen gikk ut på å lage en taleprosesseringsenhet, hvor vi hadde en DSP til signalprosessering, samt hardware for mottak av talesignaler. På det samme kortet skulle det også sitte en trådløs tranciever fra Kongsberg Analogic, slik at kommunikasjon med LON nettverket kunne opprettes. Rammene for designet vårt foreligger altså ut fra den størrelsen disse enhetene trenger, samt at det skal være mulig å bære med seg det ferdige designet. Den bærbare enheten skal være ferdig konfigurert med oppladbare batterier.

Utgangspunktet for å lage et slikt design fant vi i et testkort for lydbruk, som personell ved HiG har brukt til lignende prosjekter før. Grunnfunksjonene til kortet, samt kapasiteten, er beskrevet under hovedpunktene 2.1 og 2.2. Kompleksiteten til talegjenkjenning er så stor, at utgangspunktet for prototypen ble derfor å sikre oss tilstrekkelig prosessorkraft. Testkortet gir oss alle funksjoner vi kan drømme om for å prosessere lyd, deriblant tale, og DSPen er spesiallaget for å drive med tunge aritmetiske operasjoner av den typen vi trenger. Det naturlige valget for oss ble derfor å ta utgangspunkt i testkortet, når vi skulle utvikle vår egen prototyp.

Grunntanken var å gå grundig gjennom testkortets dokumentasjon, og håndplukke de funksjoner vi ville behøve, for så å implementere funksjonene i en ny design. Kretstegninger for testkortet forelå som en del av dokumentasjonen, noe som sikret det vi mente var et godt utgangspunkt for å lage en modifisert design. Historien videre forklarer hvordan dette bildet endret seg drastisk, etter hvert som prosessen skred fram, og hvorfor vi var nødt til å gå bort fra å lage et eget kortdesign.

2.3.2 Skjema tegninger i Isis

Basisen for designet vårt legges i ISIS, som er skjemategningsprogram i Proteus Professional pakken. Våre tegninger er basert på de skjematiske tegningene som følger med i testkort dokumentasjonen, og følger stort sett malen fra disse. Noen introduksjon til arbeidet med å lage kretstegninger og print skjemaer vil ikke bli gitt i denne omgang. Da ber vi heller om å lese instruksjonsbøkene til ISIS og ARES.

Ingen av de komponentene som skulle tegnes, hadde definerte pakker og designmodeller innlagt i ARES og ISIS. For oversikt over pakkene som ble tegnet i ARES, se avsnitt 2.3.3. Hver komponent vi skulle tegne, måtte først designes i ISIS, og legges inn i USER_DVC biblioteket i programmet, som designmodell. Alle komponentmodellene er vedlagt på CDen som medfølger rapporten, i det aktuelle USER_DVC biblioteket. Hvis det ønskes ytterligere informasjon om hvordan design av modeller fungerer, referere vi nok en gang til brukerhåndboken for ISIS. De neste punktene vil ta for seg litt om hver enkelt side i vedlegget, og knytte noen kommentarer til hver enkelt tegning, samt si litt komponentenes pakketildeling. Det anbefales å bruke denne oppsummeringen sammen med vedleggene for å få oversikt over de ulike signalbanene.

Ark 1 Kjernen i testkortet, TMS 320C6416 DSP. Samtlige funksjoner denne innehar på testkortet signalmessig er representert, og angitt som en seksjonsoppdeling. Siden pakken som blir benyttet er en 532 pinns BGA (Ball Grid Array), blir tegningen noe uoversiktlig.

Ark 2 EMIFA bussen, med oppkobling til SDRAM. Bussen henter sine signaler fra DSPen, og disse går gjennom et motstandsnettverk før de hentes av bussen. Se for øvrig Ark 3. Begge Ram brikkene har pakketype TSOPII med 86 pinner.

Ark 3 Komponenter som er mellom DSPen og EMIFA bussen. Signalene som kommer fra disse komponentene, hovedsakelig motstandsarrayer, går inn i EMIFA bussen, og kan leses av SDRAM brikkene i neste omgang.

Ark 4 Den delen av JTAG intefacet som kan ses i sammenheng med 60-pinns Hurrycane kontakten på testkortet. Signaler både fra McBSP og DSPen er representert her.

Ark 5	CPLDen med tilkoblede enheter. Legg spesielt merke til USER_LEDs som vi benytter i kommunikasjonen med LON nettverket, og DIP switchene som kan styre LEDene manuelt. En annen ekstern funksjon er reset knappen, øverst til venstre på tegningen.
Ark 6	Kobling mellom EMIFA bussen og Memory expansion kontakten. EMIFA signalene går gjennom fire 16bits buss trancievere, før de gjøres tilgjengelige for minnekontakten. Det er 32 bit av den totalt 64bits EMIFA bussen som gjøres tilgjengelige her.
Ark 7	Gir en mer ryddig oversikt over spennings og jordtilkoblinger for DSPen. Det er lettere å skille pinne fra hverandre, og segmenteringen er mer tydelig, enn den på ark 1. Oversikt over samtlige avkoblingskondensatorer foreligger også.
Ark 8	Sammenkoblingen mellom DSPen og EMIFB bussen. Denne går videre til Flash minnet. Som for EMIFA passerer DSP signalene gjennom motstandsarrayer før de når EMIFB bussen. Flash brikken er i en TSOP 48 pins pakke.
Ark 9	Power Supply. Kjernen i oppsettet er Buck spenningregulatorene, som transformerer spenning ned til 1.4 og 3.3V, fra 5V. Det er ulike spenningsuttak, med ulik avkobling. Dette muliggjør spenningsforsyning til ulike formål på DSPen, og andre enheter på kortet. Regulatorene benytter en 20 pins PWP pakke, med heatsink.
Ark 10	Inn- og utgangsoversikt for AIC23 Audio Codec. Noen av de data signalene AIC23 opererer med for synkronisering og timing er også med på tegningen. Codecen benytter en TSOP 28 pins pakke.
Ark 11	Enkel oversikt over McBSP. Signalene styres via multipleksere, og inn på DSPen. Lettest å kjenne igjen er de signalene som benyttes i samspill med Audio Codecen. Se for øvrig avsnitt 2.2.2.
Ark 12	Kretsoppsettet som driver systemklokken i kretsen. Vi har komponentene rundt krystallen som genererer frekvensen, og multiplikatorer for denne.

Presentasjonen som har blitt gitt nå er selvfølgelig på et overfladisk nivå, men gir en grei oversikt over hvor komponentene er plassert i den store sammenhengen. Ved å se punktene i sammenheng med vedleggene og beskrivelsen av hver komponent i hovedavsnitt 2.2, vil det gis oversikt over de ulike signalbanene, og samspillet mellom komponentene. Å forklare alle sammenhenger i detalj ville tatt mer plass en vi har til gode for hele rapporten.

2.3.3 Tegning av pakker i ARES

Ingen av komponentene testkortet benytter finnes i ARES sine standardbiblioteker for komponentpakker. Resultatet av dette ble uunngåelig manuelle tegninger av nødvendige pakker. Selve prosessen er tidkrevende, og stiller store krav til nøyaktighet i eventuell oppmåling, og studering av datablader. De fleste pakker er tegnet ut fra produsentens datablad, hvor det står klare oppmålinger for hver komponent. Nok en gang refererer vi til brukerhåndboken for ISIS og ARES hvis det er spørsmål angående tegning av pakker.

De produserte pakken er lagt i USERPKG biblioteket, og alle dette biblioteket er vedlagt på medfølgende CD. For å gjøre det enklere å tilordne pakker til komponentene ved en senere anledning, gis det en liten oversikt over de ulike typer pakker, og hvilke komponenter de er tilordnet.

Navn på pakke i USERPKG	Antall Pinner	Komponent
BGA_GLZ	532	TMS 320C6416 DSP
TSOP_II	86	MT48LC2M32B2, SDRAM
SOLC_CONT	60	Hurrycane kontakt, JTAG
TSOP_48	48	Am29LV400B, Flash
TSSOP48	48	LVTH16245A Buss Tranciever
TSOP_28	28	AIC23, Audio Codec
PWP	20 + sink	TPS54310PWP regulator
RESAR	16	Motstands arrayer
SO5	5	OR, INV. logikk

Tabell 2.3.3 (1)

Pakker for motstander og kondensatorer i størrelse fra 1206 – 0604 ligger standard i biblioteket, slik at disse kan tilordnes direkte i ISIS. Dette gjøres ganske enkelt ved å skrive pakkestørrelse i boksen for PCB package, slik at denne opprettes automatisk ved netlist.

2.3.4 Hardware problematikken, og løsningen på denne

I introduksjonen til dette hovedavsnittet har vi hele tiden snakket om en prototyp design, og vi innser at det kan virke rart å ikke ha nevnt noe om print skjemaer i avsnitt 2.3.3 om ARES eller i vedleggsoversikten. Det er i hovedsak mangelen på disse vi skal ta opp i dette avsnittet, og hvorfor vi gikk bort fra tanken om å konstruere vår egen prototyp design.

Som nevnt i innledningen så vi på testkortet som en naturlig base for å utvikle et eget design rundt. Dokumentasjonen som forelå så god og fyldig ut, med oversiktlige tegninger for kretsdesign og fullstendig liste over komponenter med referanser. Prosessen med kretstegningene gikk omtrent som forventet i starten; Programmene måtte læres skikkelig, og mange nye funksjoner måtte tas i bruk. At alle pakker og komponentmodeller måtte tegnes manuelt både i ISIS og ARES var ikke noe problem, og hele designet begynte å ta form da det gjensto noen uker til påske.

Første problem var designets kompleksitet. Som første gruppe sprengte vi grensen for hvor mange pinner ARES kunne klare i sin Netlist, med den student versjonen vi benyttet. Betydelige tidsressurser ble brukt i forsøket på å oppdrive en fullversjon, som kunne håndtere den datamengden vi forsynte programmet med. Omkonvertering til fullversjon krevde tegning av hele designet på nytt siden filformatene i versjonene var ulike, og direkte konvertering ikke var mulig. Bibliotekene med pakker og komponenter måtte også oppdateres på nytt. Mer av tidsressursene gikk til spille i denne prosessen.

Neste problem, kanskje ikke overraskende, kom nok en gang som resultat av designets kompleksitet. DSPen er en av markedets kraftigste, og dermed også blant de mest avanserte. At testkortet inneholder ni lag for å kunne holde styr på alle de 532 pinnene er et problem i seg selv. Å lage et tilsvarende kort er et enda større problem. Skolen har ikke materiell som er i nærheten av å kunne brukes under utviklingen, og heller ikke utstyr til å lodde på komponentene. Kontakt med eksterne bedrifter med kompetanse på disse områdene ble derfor kontaktet. CapiNor og Topro er nærliggende bedrifter, med stor faglig tyngde på disse feltene. De forsikret oss om at hvis det var ønske om det fra vår side, så ville det ikke være noe problem å motta en ordre fra oss. Likevel begynte det å skinne gjennom at en design på dette nivået, muligens lå langt over det vi realistisk kunne få til.

Det som likevel ble tungen på vektskålen var manglende dokumentasjon på essensielle komponenter. I tidligere avsnitt har vi vært inne på Code Composer og hvordan denne kommuniserer med et JTAG interface og en JTAG Emulator, før data når riktig hardware på testkortet. Dokumentasjon på disse bitene, som MÅ være på plass for å få et funksjonelt kort med gjeldende design, foreligger ikke. I tillegg er komponentene som utgjør JTAG modulen gått ut av produksjon, og lignende komponenter klarte vi ikke å oppdrive noe sted. Vi var nødt til å foreta et valg, og valgte å skrinlegge prosjektet med å lage et eget prototyp design. Det ble rett og slett for mange ukjente variable til at det var forsvarlig å legge mer ressurser i prosessen.

Selvfølgelig en stor nedtur for oss, men absolutt et riktig valg å ta, forutsetnings og ressursmessig sett. Stemmegjenkjenning blir i litteraturen omtalt som ekstremt ressurskrevende, med tunge beregningsalgoritmer, så et av poengene våre innledningsvis var å ha nok datakraft. Stor datakraft krever en avansert DSP, som testkortet har, og 532pinner er den harde realiteten vi måtte stå ovenfor. Godt inne i prosjektperioden fikk vi imidlertid en dypere kjennskap til hvordan algoritmene utviklet seg i forhold til datakraft og kodekompleksitet. Ved å gjøre koden i algoritmen mer kompleks og finurlig, kunne vi spare datakraften en stor datastruktur ville trengt. I ettertid har det vist seg at vi muligens kunne sluppet unna med en enklere DSP variant, med redusert datakraft. Om dette hadde gjort utviklingsprosessen bak en prototyp enklere rent teknisk betviles ikke. Problemet med løsning av prototypen ville sannsynligvis ikke forsvunnet av den grunn, bare endret fokus.

Uten testkortet i bakhånd kunne vi ikke utviklet talegjenkjenningssoftware kontinuerlig under prosjektperioden. Skulle dette verktøyet legges inn på et annet kort, med en annen DSP, ville også interfacet blitt endret i forhold til testkortet. Code Composer ville ikke kunne bli benyttet, noe som ville gitt oss nær sagt ingenting å gå etter i software utviklingen, når det gjaldt faste holdepunkter på strukturen.

Kort sagt kan en si at det var erfaringen når det gjaldt store kretskonstruksjoner som innledningsvis ledet oss ut i litt feil tenkebaner.

3.
CODECOMPOSER STUDIO

Utviklingsmiljøet vi benyttet var Code Composer Studio. Intergrated Development Enviroment(IDE) er utviklingsprogram levert av Texas Instrument. CCStudio inneholder alle elementer for å bruke TMS320 DSP som vertsverktøy og runtime programstøtte.
IDE inkluderer DSP/BIOS, sanntidsanalyse, debugger og optimaliseringsverktøy, C/C++ compiler, assembler, linker integrert CodeWright editor, visual prosjektstyring og en rekke simulerings- og emulatordrivere. Texas presenterer programmet som enkelt med lett og oversiktig brukergrensesnitt og at det passer brukere på alle nivåer.

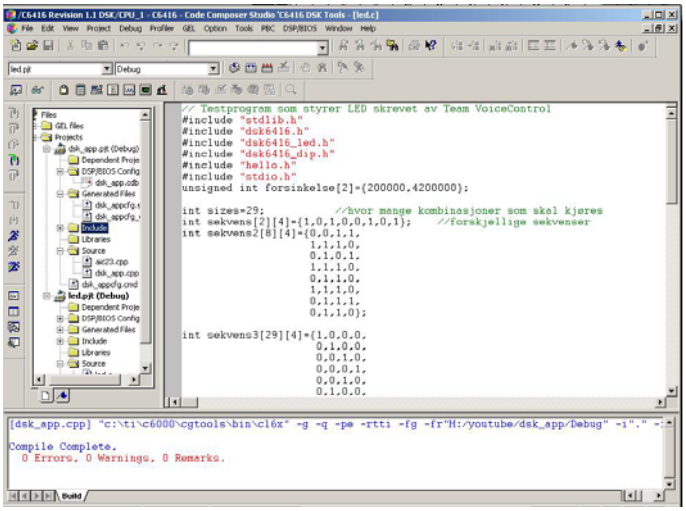


Fig 3: Brukergrensesnittet CCS

Spesifikasjoner fra texas:

- Integrated editor, debugger, profiler and project manager
- Probe points connected to the file I/O, graphical display or GEL scripts
- Advanced graphical signal analysis
- GEL scripting language for automated testing and customisation of environment
- Visual project management system
- On-line help with video tutorials and DSP chip specific information
- Annual update directly from TI's Web-site

3.1 Kode

3.1.1 Beskrivelse

Vi bruker et kodeeksempel som grunnlag for kommunikasjon med utviklingskortet. Eksemplet finnes under softwareeksempler på installasjonscden. Den modifiserte utgaven ligger som vedlegg på cd

Koden behandler digitale lyddata fra AIC23 codec, og behandler dataene gjennom en talegjenkjennelses algoritme. Output fra algoritmen sendes deretter ut som et 2x4-bits mønster til LonWorks. Flytting av informasjon på kortet skjer igjennom McBSP og EDMA for mest mulig effektiv dataoverføring uten bruk av selve DSP.

3.1.2 Dataoverføring

Sampledata blir sendt fram og tilbake til codec gjennom McBSP2, som er en bidireksjonell seriell port. EDMA er konfigurert til å ta hvert 16-bit lyd sampel som dukker opp på McBSP2 og lagre det i minne slik at det ligger klart til å behandles. Etter prosessering sendes data tilbake til codec igjennom McBSP2. tillegg brukes McBSP1 til å konfigurere codec. Codecen mottar en rekke serielle konfigurasjonsbit igjennom McBSP1 som setter all nødvendige parametere som volum, valg av lydutganger, sampelhastighet og dataformat.

I tillegg til grunnleggende EDMA, bruker vi to spesielle teknikker som gjør behandling av lyd mer effektiv.

1) Ping-pong databuffer i minne
Applikasjoner som bruker et enkelt buffer for å motta og sende data er vriene og veldig tidskritiske, siden ny data stadig overskriver data som sendes. Ping-pong bruker to buffere i stedet for ett.. EDMA er konfigurert til å først fylle Ping bufferet og så pong bufferet. På denne måten kan DSP behandle data som ligger i ping mens pong fylles opp og omvendt. Man kan altså behandle data uten å være redd for at data overskrives, samtidig som det fortsatt gir et inntrykk av å jobbe i sanntid.

2)Linked EDMA overføring
EDMA kontrolleren må bli konfigurert litt annerledes for hvert buffer. Når et buffer er fylt opp setter EDMA et intrrupt. Interruptet må resette konfigurasjonen for det andre bufferet før neste sample kommer. Det er fortsatt snakk om en ping-pong databuffer konfigurasjon. For å få dette mindre tidskritisk brukes en såkalt linket overføring. Hver busskonfigurasjon blir laget på forhånd. Dette betyr i praksis at mens en busskonfigurasjon er gjeldende, settes den andre konfigurasjonen opp. Denne ligger da klar for linking opp mot gjeldende konfigurasjon, gjennom EDMA. Det blir fortsatt satt et interrupt, men dette intrerruptet brukes bare av DSP som signal for at den kan begynne å behandle data i bufferet. Den eneste tidskritiske komponenten er at DSP må være ferdig med å prosessere før det aktive bufferet blir fylt opp. På denne måten er det mye enklere og tilfredstille real-time begrensninger.

3.1.3 Programmet

Når programmet kjører blir det laget en egen DSP/BIOS modul Resultatet av initsialiseringen er konfigurert i dsk_app1.cdb ved hjelp av DSP/BIOS konfigurasjons verktøy.
Main funksjonen blir kalt. edmaHWi() interruptrutine blir kalt når buffer er fylt opp. Den inneholder en tilstandsvariabel pingpong som indikerer om buffer er ping eller pong. dmaHwi skifter buffertilstand til det andre bufferet og kaller SWI processBuffer for å behandle lyddataen.

I starten blir alle nødvendige initsialiseringer gjort. Kodek sampler på 48kHz og formatet er 16bit. Når et interrupt settes lagres det unna et bittmønster i minnet. Størrelsesorden er XxX. Vi trenger egentlig bare og sample på 12kHz for å få med oss den viktigste informasjonen men vi bestemte oss for å sample med en hastighet på 24kHz. Dette kan lett fikses ved å bare lar annen hvert sample gjelde. Vi taper ikke noe prosesseringskraft på dette siden data kastes fram og tilbake fra rammen via CPLD og databusser og CPU er minimalt deltagende i denne prosessen. Når siste data er lagret unna setter gjenkjennelsesprosessen i gang.

3.1.4 LED
Programmet inneholder også en funksjon for styring av de fire Led som finnes på kortet. Ledene styres igjennom en CPLD som igjen styres igjennom en McBSP buss. Når det kommer til led trengs det ingen manuell konfigurasjon av McBSP siden denne ligger ferdig i DSK6416_led.h
Disse led utgangene bruker vi senere for å kommunisere med LonWorks. I den forbindelse lagde vi også et testprogram som lett styrte ledutgangene og hastigheten. På denne måten kunne vi teste både hvordan Neuronchipen reagerte og hvor fort neuron kunne oppfatte signaler levert på denne måten.

3.1.5 Codec
Codecen er kanskje den viktigste perifere enheten på kortet etter DSP'en selv. AIC23 har 10 registre som inneholder styring av f.eks volum, datafomat, samplefrekvens. Registrene blir satt gjennom den reserverte kontrollkanalen McBSP1. Vi beholdt lydutgangene til hodetelefon utgangen aktive, siden det er enkelt og greit å finne ut hva som ble lest inn. Ingen lyd ut betyr at noe gikk galt på veien.

3.1.6 Uskrift fra DSP
Hvis man analyserer data i realtime må man benytte RTDX kontroller som muliggjør skriving av data til og fra fil. Du åpner spesielle kanaler som verts pc lytter på og motsatt hvis du skal skrive fil til DSP. Vi prøvde lenge å få dette til å fungere ordentlig. Programmet opprettet filen, men nektet å skrive til denne. Dette problemet ble delvis løst ved å bruke LOG kanalen som skriver logg til fil. På denne måten kan man i hvert fall skrive en begrenset mengde data til fil, selv om problemet med å lese inn ikke blir løst. Det følger programeksempler med kortet som viser hvordan dette gjøres, men fremgangsmåten fungerer ikke når vi bruker samme fremgangmetode i vårt program.
Mange av disse problemene er ikke direkte knyttet mot resultatet, siden målet ideelt sett er å utvikle en bærbar enhet som ikke alltid er knyttet opp mot en pc.

3.1.7 Noen problemer i forbindelse med utviklingsmiljøet fra C til C++
Siden vi tidlig fant ut at det ideelle hadde vært å bruke C++ kode, var det nærliggende og endre miljøet til C++. Både fordi vi har en bedre bakgrunn i C++ programmering, og fordi vi etter hvert så at funksjoner i C++, som klasser og lister, gjør jobben med å behandle data både mer effektiv og oversiktelig.

Av en eller annen grunn dukket det opp en del uforutsette problemer forbundet med dette. Et krav fra DSP/BIOS som vi ikke visste om i starten, var at DSP/BIOS ikke kan kalle funksjoner i C++. Dette problemet ble løst ved å benytte såkalte eksterne C blokker. Ekstern C blokk isolerer en valgt del av koden og behandler denne delen som om den fortsatt var C. Prinsippet går ut på å "lure" DSP\BIOS til å tro at koden står skrevet i C, og IKKE i C++. DSP\BIOS støtter ikke *.CPP endelse på filer, slik at funksjoner i disse må settes i en C blokk for å bli kompilert som vanlig C kode. Syntaksen for sekvensen er ganske enkelt:

Extern "C" { //Funksjon };

I tillegg var det enkelte funksjoner i forbindelse av initsialiseringen av codec som sluttet å fungere. Løsningen var å kalle andre funksjoner som utførte samme oppgave. Akkurat denne delen har vi ikke funnet på noe god forklaring. Men selv om CCS skal ha full C++ støtte, virker det som om den fortsatt liker C best, og er lagt mest opp til at brukeren også skal benytte seg av C.

3.1.8 Implementering av kode på DSP
Talegjenkjennelsesalgoritmen ble etter hvert noenlunde klar og arbeidet med å implementere algoritmen inn på selve DSP kunne begynne. Vi valgte og legge inn funksjon for funksjon for å sikre at hvert ledd fungerte som det skulle. Men pågrunn av kodens komplekse natur og begrensede muligheter til debugging er det lett og miste oversikten, og dette førte igjen til at vi dessverre aldri rakk og implementere hele programmet på DSP. Men å få dette til å fungere er igjen et spørsmål om tid.

3.1.9 Dårlig Dokumentering

Et problem som etter hvert dukket opp er dokumentasjon når det kommer til Texas. Det fulgte med utrolig mye informasjon men kortet, men mye av denne dokumentasjonen er i beste fall vanskelig å forstå. Mange eksempler som fulgte med er beregnet på andre versjoner av kort eller faktisk ikke fungerer i det hele tatt. Mange eksempler bruker også syntaks som vi ikke finner forklaring på noe sted. Alt dette gjorde prosessen med å forstå og bruke CodeComposer mer komplisert og mer tidskrevende. Siden vi ikke har lang erfaring innen programmering og få på skolen har erfaring i tilknyting til bruken av dette programmet, har vi måttet gi opp enkelte funksjoner som vi gjerne ville ha hatt med.
I hjelpfunksjonen henvises det ofte til forskjellige linker som er knyttet til hjemmesiden til Texas. Men disse linkene er døde eller man blir sendt til førstesiden. Texas sin hjemmeside er utrolig uoversiktlig og rotete, og fører bare til irritasjon hos bruker. Derfor har mye av det vi har funnet ut basert seg på internettsøk og lesing av forskjellige forum på nettet som ikke er tilknyttet Texas.

Vi har unektelig tatt en del nyttig lærdom av de feilene som har blitt begått under konstruksjon av både hardware, og den prosessen det er å sette seg inn i nye utviklingsverktøy. Når det gjelder Code Composer fant vi dette programmet usedvanlig komplisert i forhold til de programmer vi var vant til å bruke fra før. Flere småproblemer voldte oss utrolige arbeidsressurser når det gjaldt tid. De fleste problemer som vi brukte tid på å løse viste seg å være enkle, men med den lille erfaringen vi har i bruk av utviklingsverktøy, blir prosessen med å finne disse feilene for lang i forhold til oppnådd resultat. Det er ingen ressurspersoner på skolen som har brukt dette verktøyet tidligere, og derfor ble de input som kom ikke særlig brukbare i problemløsningsprosessen.

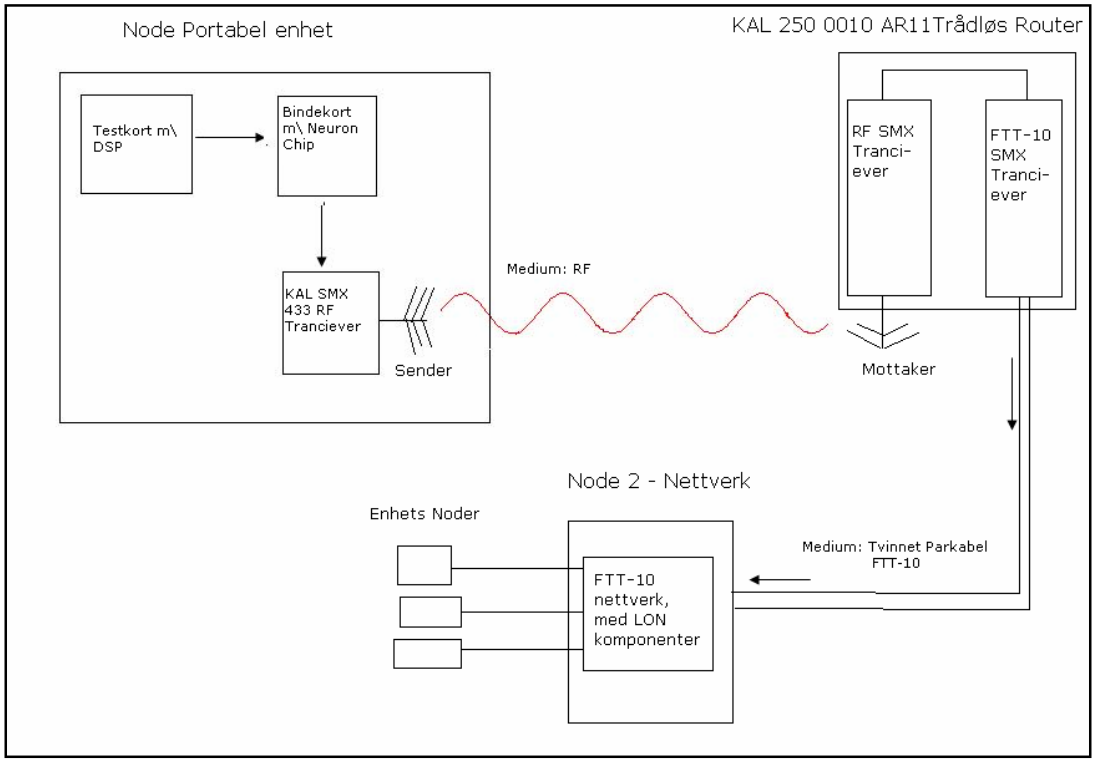
Tilbake på hardware mener vi selv at prosjektet i utgangspunktet var beregnet på en utviklings innfallsvinkel. Med dette tenker vi oss at vi skal utvikle verktøyene som skal benyttes fra bunnen, noe som inkluderer den hardware som skal benyttes. Uten erfaring på området med konstruksjoner i noen som helst skala, blir det umulig å vurdere innledningsvis hva vi kan forvente å oppnå. Prosjektet har vi hele tiden sett på som en arena for utvikling, og at vanskelighetsgraden skal settes der etter. At vi la terskelen for høyt i utgangspunktet hersker det vel liten tvil om nå i avslutningsfasen, men likevel, det er slik utvikling foregår.

4.
LON WORKS

4.1 Introduksjon til LON nettverket

For å muliggjøre en portabel forflytning av stemmegjenkjenningssystemet, er det nødvendig å sette opp trådløs kommunisering mellom den bærbare enheten og det sentrale distribusjonsnettverket. For å realisere dette har vi satt opp et distribusjonssytem bestående av en trådløs SMX trancieveren fra Kongsberg Analogic på sender siden, en ruter fra samme produsent som mellomstasjon, og en del enkeltnoder på mottaker siden. Ruteren har som funksjon å rute datapakker fra den formen som RF-trancieveren benytter, og over til FTT-10 type pakkedata. FTT-10 tranciever i ruteren sender så data videre til en sentral, som distribuerer signalene ut til de forskjellige nodene på nettverket. I et tenkt arbeidsmiljø kan disse enkeltnodene være alt fra lyspaneler, til kontrollpaneler for varmestyring i et hus eller for en stekeovn.

Signalering mellom ruteren, nettverket i Node 2 og de ulike enhetsnodene, skjer ved bruk av en tvinnet parkabel som overførings medium. Før det blir gitt en nærmere beskrivelse av hver enkelt byggestein i distribusjonssystemet, kan en visuell oversikt klargjøre litt av hovedtankene. Disse er forsøkt illustrert på figur 4.1 (1)



Figur 4.1 (1): Prinsippet bak LON nettverket, med tilhørende noder

Starten på distribusjonssystemet tar logisk nok utgangspunkt i vårt testkort, hvor dataene fra stemmegjenkjenningen blir prosessert og videresendt av DSPen. Her vil det nok kunne knyttes noen kommentarer til den litt utradisjonelle tilkoblingsmetoden vi har valgt, men vi kan begrunne dette på flere måter.

Det er nevnt i avsnitt 2.1 under DSP introduksjonen, at hastigheten på EMIFA og – B bussene er 120MHz. Dette er langt over de 10MHz vår Neuron Chip har som klokkefrekvens, og den frekvensen data kan mottas med sikkerhet på. Det viste seg også umulig å justere hastigheten på EMIF-bussene, da denne lages av heltallige divideringer av systemklokken for DSP Core modulen på 720MHz. Signalet AECLK_IN og BECLK_IN er styringssignaleter for EMIFenes klokkehastighet, og er en dividering av kjernehastigheten med 6 ganger, altså til 120MHz.

Neste mulighet for å komme rundt problemet med klokkehastigheten var å se på de perifere kontaktene. Disse er for øvrig beskrevet mer utførlig i avsnittene 2.2.4 – 2.2.6. Kontakten som kobler oss direkte til minne-bussen, Expansion Memory Interface, var første alternativ vi vurderte for tilkobling. Her ville vi imidlertid fått samme hastighetsproblemer som med EMIF bussene, siden kontakten gir direkte adgang til disse. Et annet element i denne sammenhengen er at kontakten kun er direkte kompatibel med andre minneenheter, for eksempel ekstern RAM. Skulle vi fått kontakten til å fungere med Neuron Chipen direkte ville dette vært nesten umulig, uten å ha en del komplisert logikk imellom. Da konstruksjon av et sånt system sannsynligvis ville ta mye tid, i tillegg til at det ikke er sikkert det ville fungere i det hele tatt, utelukket vi muligheten til å benytte Expansion Memory Interface i oppsettet.

De andre eksterne kontaktene, Expansion Peripheral Interface(EPI) og Host Port Interface(HPI), viste seg også å være utilgjengelige for den type informasjon vi skulle sende. Kompleksiteten til disse kontaktene er stor, som for testkortet generelt, og det var etter konsultering med fagpersonell på skolen at vi slo fra oss tanken om å benytte disse kontaktene. Host Port interfacet er spesielt laget for at flere DSPer skal kunne jobbe sammen om en oppgave, men adgang til de signaler dette krever. Når det gjelder Expansion Peripheral Interface får vi samme problem som for med HPI. De signalene som kommer ut er ikke beregnet for vårt formål, med større vekt på mulighet for eksterne interrupt. Interfacet er også laget for å fungere direkte med serielle enheter, noe som strider mot vårt ønske om å sende data parallelt til Neuron chipen.

Med også disse kontaktene ute av bildet, måtte vi ty til ukonvensjonelle metoder for å få vist de prosesserte data. Realiteten er at med 8bits dataord, så kreves det kun en simpel måte å overføre på, med hovedvekt på at vi har full kontroll og styring av denne overføringen. Dette har jo vært et av hovedargumentene for å gå bort fra de eksterne kontaktene, i tillegg til at de ikke hadde funksjonen vi var ute etter. Kontakter på 75pinner er litt i meste laget når vi trenger 8 signalutganger, i tillegg til drivspennings og jord tilgang. Grunnleggende funksjoner vi trenger for å overføre data:

- Vi må ha enkel tilgang til de data fra stemmegjenkjenningen som er prosessert av DSPen. Tilgangen til dette minneområdet må være rask og enkelt konfigurerbar Gjennom software.
- Software basert styring av registre som vi kan sette til ønsket datamønster, for så å sende dette videre til en egnet utgang.
- Utgangen må være lett tilgjengelig for tilkobling, og kunne sende data parallelt. En forutsetning for at prinsippet skal virke, er at vi har en frekvens ved sending godt under 10MHz.

Etter en vurdering av tilgjengelig kapasitet på kortet, kom vi fram til at ved å benytte lysdiodene som styres av CPLDen, ville få til en tilfredsstillende forsendelse av data. Et viktig kriterium for at vi valgte denne løsningen er den enkle konfigureringen via software. Den skjematiske koblingen mellom diodene og CPLDen er beskrevet i vedlegg.

Begrensningen ved oppkoblingen går på antall bit vi kan sende parallelt. Det er fire såkalte USER_LEDS som vi har tilgang til direkte, noe som kun muliggjør sending i fire bit parallelt per syklus. For hver syklus latches fire bit inn på I/O inngangene til Neuron Chipen, og lagres unna for sammensetning til det totale ordet. Når latchingen av de første fire bitene er fullført, sendes de neste fire bit til diodene, og latches inn etter samme mønster som de forrige bitene.
Total blir ordet som er lagret i Neuron chipen på 8bit, den datalengden vi i utgangspunktet ønsker.
Vi vil bruke I/O inngang 0-3 på Neuron Chipen til dette formålet.

Siste tilkobling til selve teskortet går på riktig jording, og på spenningsforsyning 5V. Dette er forutsetninger for at vi skal få lik referansejord på både binde koret og RF-Trancieveren, samt riktig spenning. Testkortet er rikelig utstyrt med uttak for begge disse nødvendighetene, slik at implementeringen blir enkel.

4.2 Node 1

4.2.1 Tankegangen bak bindekortet

For å få etablert en suksessfull kommunikasjon mellom en tranciever og testkortet, trenger vi et mellomledd, eller et såkalt binde kort. Dette kortet skal i utgangspunktet inneholde det interfacet som skal til, for å få en neuronchip til å kommunisere med den tranciever typen som skal benyttes. Denne logikken er spesifisert gjennom standarder lansert av Echelon, og via databladene av trancieverens produsent. I vårt tilfelle var oppbygningen av tranciever logikken forenklet, siden Kongsberg Analogic bruker et eget Neuron Interface, kalt Special Purpouse Neuron Interface. Dette interfacet er inkludert på tranciever kortet, noe som gjør ekstra logikk fra kommunikasjonsportene på Neuron Chipen overflødig. For øvrig går KAL SMX 433 Trancieveren som en standard SMX tranciever etter Echelon standarder.
Nøkkelfunksjonene på bindekortet kan oppsummeres i noen enkle punkter:

I ▪ Avkoblings muligheter for Neuron chipen
II ▪ Implementering av en Krystall, for å sikre rett klokkefrekvens
III ▪ Oppsett av reset funksjon for Neuron Chipen, også med ekstern bryter
IV ▪ Interface for kommunikasjon med Tranciever
V ▪ Valg av Neuron Chip

I:

Som alle andre elektroniske enheter trenger Neuron chipen riktig avkobling mellom jord og +5V spenningsforsyning. Det er viktig at dette forekommer mellom alle tilkoblingspinner for +5V og jord. Når det gjelder verdien på avkoblingskondensatorene er det ingen faste verdireferanser i de beskrivende datablad og øvrige konfigurasjons skrifter. Vi vil uansett ikke trenge noen stor verdi, siden det ikke går høyfrekvente datasignaler på binde kortet som må avledes. På relativt ukomplekse konstruksjoner har vi gode erfaringer med avkoblingskondensatorer på 10nF. Denne verdien er derfor benyttet på binde kortet.

II:

For å sikre at Neuron Chipen har riktig klokkefrekvens på 10MHz, er vi nødt til å implementere en kobling med en krystall som ordner dette. Anbefalte koblingsoppsett for dette er grundig spesifisert i de fleste bøker som omhandler LonWorks teknologi, hvor vår referanse er en variant fra Motorola. Denne står oppført over litteratur referanser som er benyttet. For å sikre rett oscillasjonsfrekvens til klokkeinngangene må vi sette opp komponenter rundt krystallen, etter fabrikkmessig spesifikasjon. Tabell oppslag i Motorolas LonWorks bok angir verdien på kondensatorer og motstander for oppkoblingen.

Det kan knyttes en ekstra kommentar til bruken av motstand Rf (se printutlegg), som ligger mellom chipens CLK1 og CLK2 pinner. Denne står i parallell med 1MΩ impedansen som ligger mellom CLK1 ogCLK2. Med en verdi på 100kΩ skal Rf senke den reelle impedansen, slik at vi som et alternativ til krystall, kan lage klokkefrekvensen med en keramisk resonator. Siden Rf kan benyttes sammen med en krystall har vi tatt den med i konfigurasjonen, i tilfelle vi ved en annen anledning skulle ønske å bytte oscillasjonskilde.

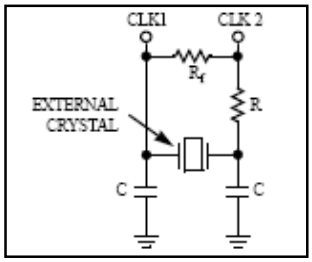


Fig. 4.2.1 (1) Generelt oppsett av krystall

III:

I flere sammenhenger kan det være nyttig å ha kontroll over en ekstern resetfunksjon av Neuron chipen, i samhandling med en automatisk reset, som tiltrer hvis gitte grenser overstiges.
For automatisk reset bruker vi en IC fra Dallas, DS1233 5V Econoreset. Denne er koblet til +5V spenningsforsyning og jord, og passer konstant på at spenningsnivået er i akseptabelt intervall. Dette skjer ved å sammenlikne gjeldende nivå med et referansenivå, ved å benytte en enkel komparator kobling. Kommer en uakseptabel verdi for spenning sendes et aktivt lavt resetsignal til chipen. Dette holdes i 350ms, til spenningen er tilbake i riktig intervall.

Komponenter rundt reset kretsen, angis også i databladet. Avkoblingskondensator mellom +5V og jord i området 10nF er innenfor anbefalt område, sammen med en kondensator mellom reset og jord på 1nF. Det bør også sitte en kondensator mellom Vcc pinnene og GND pinnene på reset kretsen. Dette er for ytterligere sikker avkobling. Verdien på 10nF er fin også til denne avkoblingen.

IV:

Tranciever interfacet er en vital bit av ethvert kort som skal fungere som grensesnitt opp mot en tranciever. Normale grensesnitt som er spesifisert knyttes som regel opp mot Direct Connect og diverse varianter mot Tvinnet Parkabel, for eksempel FTT-10. Disse er godt definert i boken vi har introdusert før i dette avsnittet. Siden vi skal ha grensesnitt opp mot en RF SMX tranciever, stiller situasjonen seg litt annerledes. Det finnes ingen ferdig spesifiserte interface som vi selv kan bygge inn på et binde kort, opp mot RF. Produsentene av de ulike RF trandscieverne implementerer sin interface logikk på trancieverkortet selv, innenfor standard rammer for SMX Trancievere. Vi har derfor ikke trengt å ta hensyn til dette på binde kortet. Koblingen av kommunikasjonsportene på Neuron chipen skjer derfor dirkete inn på kontakt pinnene som RF trancieveren benytter.

V:

For en kretskonstruksjon er det ikke likgyldig hvilken type Neuron chip som velges. De forutsetningene som legges til grunn for valget, er hvor komplisert kode vi skal implementere, og hvor enkel design vi ønsker på kortet. Forskjellen på en konstruksjon med 3120 varianter av chipen og 3150 varianten er muligheten for intern ROM på 3120. Siden den koden vi skal implementere er forholdsvis enkel, og mest skal sette de rette SNVTene ut fra det dataordet som blir sent inn til chipen, falt valget på en 3120 chip. Nærmere bestemt MC1431210E2DW fra Motorola. Vi har erfaring med denne chipen fra tidligere prosjekter, og E" varianten er den med størst internminne.

4.2.2 Design av bindekort i ISIS

Bindekortet er tegnet i ISIS etter de spesifikasjoner som er nevnt i avsnitt 4.2.1. Ut fra disse skal det være ukomplisert å få riktige komponenter på plass. Designet for øvrig er hentet fra de kortene som HiG bruker i Lon City. Forskjellen er at dette kortet er modifisert fra å benytte et interface mot en direct connect tranciever, til å interface mot vår RF tranciever. Designet er i vedlegg C.

4.2.3 Printutlegg i ARES

Det første som ble gjort i denne prosessen var å ta et valg når det gjaldt tilkoblingskontakter, både til testkortet og til trancieveren. Valget falt på en 14 pins mot testkortet, siden vi kun trenger begrenset med signaloverføring, og en standard 20 pinns SMX kontakt mot trancieveren. Disse ble målt omhyggelig med skyvelær, slik at det kunne lages pakker for kontaktene. Pakke til Neuron chipen ble laget fra datablad for Motoral sin 3120 modell. Krystallen fikk også laget pakke ut fra fysiske mål.
Alle motstander og kondensatorer ligger i standardbiblioteket, og vi benyttet størrelsen 0805 som gjennomløpende standard på disse.
Printutlegg og Top-silk er presentert i vedlegg C.

4.2.2 KAL SMX 433 tranciever

KAL SMX 433 RF tranciever fra Kongsberg Analogic er en godt kjent tranciever for trådløs overføring til LonWorks teknologi. Spesifikasjonsmessig faller den under de standarder som Echelon har fastsatt for SMX-trancievere, i extended versjon, det vil si med større kortlengde. Det er ikke nødvendig å oppgi all dokumentasjon på dette her, siden fyldige datablad er tilgjengelig. Referanser til disse ligger under litteraturhenvisninger. Det er imidlertid noen spesifikasjoner for KAL SMX 433 som er særskilt for denne. I forrige avsnitt ble det nevnt at vi ikke behøvde noen ekstra logikk opp mot kommunikasjonspinnene, siden trancieveren inneholder dette.

4.3 Neuron C koden

4.3.1 Innledning til koden

Neuron Chipen er kjerneenheten i den delen av kommunikasjonen, som formidler DSPens omgjorte talekommando, til et utførbart og synlig resultat på nettverket. Når vi snakker om synlig resultat, kan dette for eksempel være at et lys tennes, slukkes eller dimmes. DSPen tar imot en talekommando gjennom audio-codecen, prosesserer data som er mottatt, og sender ut et 8-bits mønster basert på hvilken kommando som har blitt gitt. Det er dette 8 bits mønstret som neuron chipen behandler. I Neuron C koden ligger det definert et sett med dataord, som DSPens kommandoord blir sammenlignet med. Hvilken operasjon som blir utført på nettverket avhenger av hvilken match Neuron chipen finner i sitt bibliotek av kommandoer, mot den som ble sent fra DSPen. I utgangspunktet har vi fire grunnfunksjoner for styring av et lys, som basisplattform for testing av vårt talegjenkjenningssystem: On, off, dimm_20 og dimm_50. Disse skal dekke det vanlige spektret av grunnfunksjoner for et lys, med muligheter for dimming der rett releé typen er implementert. En ting som er viktig å huske på er at den koden vi har konstruert, er beregnet for å utføre testing av grunnfunksjoner i systemet vi har utviklet, samtidig som det ligger plass til utvidelse av dette funksjonssettet. Dette blir nærmere kommentert i neste avsnitt.

4.3.2 Forklaring av koden

Første valg som ble tatt var hvilken metode vi skulle bruke for å legge inn bitmønstret fra DSPen, og hvordan vi skulle lagre dette med minst mulig opptak av internt minne på Neuron Chipen. Vi fant ut at den beste måten å lagre bitene på ville være i en array, hvor hver skuff skulle representere ett bit. Dette blir et naturlig valg, da en variabel per bit ville opptatt unødig minne, i tillegg til at det ville blitt vanskelig å holde styr på de ulike bitene. Hver skuff angis som typen bit. Angivelse som integer gjør at hver skuff i teorien kan inneholde 8 bit, noe som er sløsing med minne, så lenge vi kun trenger plass til ett bit.
Arrayer har også fordelen av enkel tilgang til spesifikke skuffer, og rask gjennomgangsmulighet via for løkker.

Å jobbe med bitmønstret i arrayen direkte, uten å gå veien om integer variable, viste seg etter hvert vanskelig. Grunnen til dette er programmeringsspråkets svakhet når det gjelder å kjenne igjen bitmønstre. Det finnes ingen binære regneregler som gjør at vi enkelt kan sette sammen det mønstret som ligger i de indekserte skuffene, og dermed sammenligne med et annet binært dataord direkte.Vi valgte derfor å gå veien om vekting, fra det binære systemet, og over til det desimale. For å få ut integer verdien av bitmønstret i arrayen, må vi vekte hver skuff, i henhold til det binære systemet. Arrayene viser her sin styrke, ved å gi mulighet for enkel flyt i vektingen. Ved å operere med to arrayer, kan vi løse vektingen som følgende illustrasjon viser:

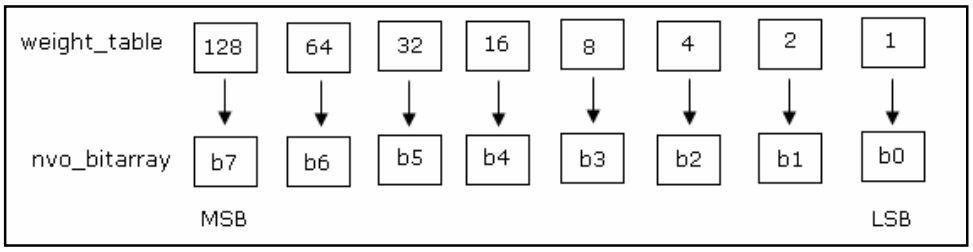


Fig. 4.3.2

Så Lenge nvo_bitarray inneholder vårt 8 bits dataord, kan vi opprette arrayen weight_table til å inneholde de faste vektingsverdiene. Den åpenbare fordelen er at ved å kjøre en enkel for løkke, kan vektingskoeffisienten i weight_table multipliseres direkte med bitet i tilsvarende skuff I nvo_bitarray. Da slipper vi ytterligere prosessering, og kan addere sammen hvert element direkte i en summeringsvariabel. Alt dette skjer I samme for løkke, i en programsetning. Se for øvrig fullstendig kode med kommentar, i vedlegg D.

Siden koden er enkel for denne sekvensen, tar den lite plass i minnet. Da dette også er en enkel løsning, mener vi at dette er en god måte å løse overgangen mellom DSPen og nettverket på. I bakhodet kan en også ha at integer variable er lettere å håndtere, som vi har vært inne på, enn et bitmønster. En vanlig short integer inneholder verdier som omfattes av 8 bit, noe som passer akkurat til det totale dataordet vi benytter. Når det gjelder verdi vil integer variabelen gå mye høyere enn vi trenger i første omgang, men det betyr at det finnes muligheter for utvidelse, når det skal legges inn flere unike kombinasjoner for stemmekommandoer.

Valget med bruken av en struct for de tilstandene vi opererer med på et lys, kan også tillegges litt forklaring. Prinsippet med structer, eller structures, er at base strukturen inneholder et sett med relevante variable, structens medlemmer. Oppretter vi variable av struct typen, har alle som er av en viss type struct tilgang til alle dens medlems variable. For en struct av typen lys kan medlemsvariablene være tilstander vi ønsker at lyset skal befinne seg i på ulike tidspunkt.
Fordelen med å legge variable i stucter er at det er veldig fleksibelt, med tanke på hvor mange variable vi kan opprette av struct typen vi lager. Vi kan definere så mange variable av en struct type som vi ønsker, og så mange medlemsvariable av structen som vi trenger.

Structen i vårt program inneholder de tilstandene vi har definert som mulige for et lys. I første omgang dreier dette seg om på og av funksjoner, og prosentvise dimmefunksjoner. Det er mulig å utvide structen med alle de ekstra funksjoner for lys vi kan tenke oss, for eksempel flere intervaller for dimming. Selv om vi i gjeldende program konfigurasjon kun benytter oss av ett lys for å teste funksjonen til stemmegjenkjenningsverktøyet vi har laget, vil structen forenkle syntaks ved tilkobling av flere lys til nettverket.

Alle medlemsvariable i structen følger integer definisjonene til den SNVT_switch som vi benytter. Dette betyr at struct medlemmene som refererer til .state typen i SNVT_swithc er unsigned integers, mens medlemmene som refererer til .value typen i SNVT_switch er signed integers.
State er knyttet til av og på funksjoner for lyset, mens value er knyttet til dimmefunksjoner. For ytterligere referanser på .state og .value medlemmene, refereres det til SNVT_master list.

Mens vi er inne på SNVT-switch kan vi begrunne vårt valg av denne. For det første er den en SNVT som inneholder tilstander spesielt tilpasset tilsvarende funksjoner som de lysfunksjonene vi ønsker å styre. Dette innebærer en diskret av og på mulighet, sammen med en funksjon, som gir mulighet for skrittvis styring av verdien som sendes ut. Dette er perfekt for en dimme funksjon. SVNT_switch.state har verdiene av for state = false=0, og på for verdiene state = True > 0. SNVT_switch.value har verdien 100% ved value =200, og 50% ved 100 som verdi. Dette gir en total oppløsning på 0.5% per step.

En annen tanke bak valget av SNVT-switch er at denne er kompatibel med de reléene og bryterne som kontrollerer lysene vi skal styre. Programmet er laget for å styre en koffert med et FTT-10nettverk, som inneholder en rekke av de lysfunksjonene som vi ønsker å kontrollere med stemmekommandoer. Lysene er sertifisert via Echelon, og er satt opp til å virke sammen med et visst sett SNVTer. SNVT_switch er en av de som er kompatibel både med vanlige bryterreléer og dimmereléer. At vi i utgangspunktet går til bruk av en SNVT fremfor en vanlig definert nettverksvariabel er naturligvis på grunn av nevnte kompatibilitet, men også fordi det er en fordel å ha de funksjonene vi trenger klart definert, og standardisert. Alle SNVTer er standardiserte i forhold til å benyttes mot de fleste typer interface, noe som gir oss ønsket fleksibilitet.nle

Som beskrevet i avsnitt 6.1 er det eneste vi benytter for å hente inn ønsket bit mønste, de fire I/O inngangene fra IO0 – IO3. Bakgrunnen for å kun benytte fire innganger er at testkortet i den konfigurasjonen vi benytter det, kun sender ut fire bit parallelt. For å sette dette sammen til det totale dataordet på åtte bit, benytter vi oss av timede sekvenser mellom DSPen og Neuron chipen. Neuron chipen setter opp et tidsvindu som DSPen kan sende inn første 4 bits del av dataordet. En ny timer setter i gang den sekvensen som lagrer de første fire bitene på rett plass i arrayen vi lagrer det totale dataordet i. To like timer sekvenser laster inn og lagrer også de 4 neste bitene på rett plass.

Når det gjelder nullstilling av arrayen der dataordet ligger, og initialisering av innlesningsprosedyre, så styrer vi disse med henholdsvis med MSB og LSB. Et høyt nivå på LSB markerer at vi starter innlesning av dataord, mens høy verdi på MSB starter nullstillingsprosedyren. Bakgrunnen for valg av en slik mekanisme, har bakgrunn i at vi i utgangspunktet ikke skal kjenne igjen mer enn 5 kommandoer. Ytterligere utvidelse av kommandoer skal heller ikke være noe problem, da vi kan ha 256 forskjellige kombinasjoner i dataordsvariabelen, som kan brukes til å trigge ulike kommandoer. Vi legger bitmønsteret for våre kommandoer godt spredt i verdi spektret, slik at det ikke skal være noen tvil om at en høg verdi i MSB og LSB har en unik betydning.

4.4 Brenning av Neuron Chip

Neste skritt på veien for å implementere Node 1 i det totale oppsettet, er å brenne inn den informasjon som er nødvendig, for at Neuron chipen skal kommunisere med Trancieveren. Det som i hovedsak trengs er en image fil for 3120, der det ligger Tranciever info, og den kilde koden i Neuron C som skal legges på chipen. Filformatet er *.nei (Neuron 3120 Image), og formatet lages i Node Builder. Det kan gis en kort gjennomgang av prosessen som det må kjøres gjennom for å få generert denne filtypen.

Først må neuron C koden legges inn i Lon Builderen. Legger filen i hovedkatalogen, og sjekker under File menyen om filen ligger der. Neste skritt er å lage den noden chipen skal virke inn i, og betingelsene rundt denne. Hardware Noder legges inn ved å benytte App Node knappen i menyen, trykke på Target HW, og så Create fra toppmenyen. En del data om noden må legges inn i denne sekvensen. Noden er av typen Coustum node, med channel KRTM_2400. Det er denne innstillingen som sier noe om hvilken tranciever noden skal fungere sammen med. KRTM_2400 er den trådløse konfigurasjonen som ligger nærmest opp mot den hastigheten KAL 433 SMX benytter. Under menyen HW.Prop Name skal vi spesifisere typen Neuron Chip som benyttes. For vår del en 3120_E2_5 variant.

Neste trinn er å lage det image som skal brennes ut på chipen, av kilde koden. Undermenyen App Image muliggjør dette, ved å trykke create, velge source code som image versjon, og kalle imaget det samme som neuron c filen koden hentes fra. Siste steg er å gå inn på Node Specs, og trykke create fra menyen. Her legger vi inn navnet på noden, velges nå, navnet på Imaget, samme navn som App Image, og tilslutt hvilket navn noden har som HW, navnet som ble gitt i Target HW seksjonen.
Etter å ha laget Node Specs trykket vi export i toppmenyen, og får dannet det filformatet vi ønsker, en *.nei fil. Denne kopierte vi over til en PC med software for brenning, slik at image filen kunne overføres til Neuron Chipen.

4.4.1 Videre gang i nettverksbyggingen

Å lage Image filen var det siste vi rakk innenfor de tidsrammene som var satt for prosjektet. Verktøyet for å brenne imaget over på filen ble overlevert lovlig sent, og uten kompetanse til å bruke den medfølgende programvaren, med lite hjelp tilgjengelig på det nødvendige tidspunktet, brukte vi resterende tidsressurser på andre aspekter ved prosjektet. Den videre gangen i nettverksbyggingen blir å sette den ferdig brente neuron chipen tilbake på binde kortet, og sørge for at det blir gjort riktig signalering mot trancieveren. Den noden vi har skal plugges til det FTT-10 nettverket vi ønsker å kommunisere med. I utgangspunktet er det ønskelig å benytte de kompatible lysenhetene på HiG sin LonWorks koffert. Installasjon på denne foregår ved å koble til LonMaker for Windows programmet, og installere nodene gjennom det ruter tilkoblede nettverket. Kommunikasjon mellom nodene skulle ikke være noe problem, i og med at SNVT_switch er en standard nettverks variabel som er kompatibel med de vanlige lys releene vi har benyttet på skolen fra før. Dette gjelder både releer for dimming, av vanlige av/på relee.

5.
TALEGJENKJENNING

5.1 Om tale og gjenkjenning av denne

5.1.1 Hvorfor et problem?

Som nevnt innledningsvis er talegjenning et teknisk område som har vært under lupen i snart et århundre. Teorier og hypoteser har har blitt oppfunnet, tatt i bruk og med tiden ansett som foreldet. De ble derfor forkastet og glemt. Det viser seg imidlertid at man stadig går tilbake og plukker opp gamle tråder for så å følge disse og at man gjenoppfinner gamle algoritmene på nytt. Det hele er i det hele tatt et veldig tydelig tegn på at talegjenkjenningen som teknologisk område er langt fra modent.

I fremtidens visjon ser man for seg at maskiner og mennesker skal kunne kommunisere verbalt med hverandre, og at maskinene skal være minst like dyktige som oss selv til å tolke og respondere på tiltale. Problemet er som vi skal komme mer detaljert inn på ikke hvordan man kan behandler snakkede signaler og trekker fram de essensielle data fra disse, men hva man så gjør med dataene man har i hende. Vi har å gjøre med hva som kalles tvetydighet (ambiguity). Ta setningen "I made her duck", denne kan tolkes på for eksempel følgende måter:

1. Jeg tilberedte en and til henne.
2. Jeg tilberedte anden hennes.
3. Jeg lagde (den kunstige) anden hennes.
4. Jeg fikk henne til å dukke (hyppig senke hodet).
5. Jeg slo henne med min tryllestav og forvandlet henne til en and.

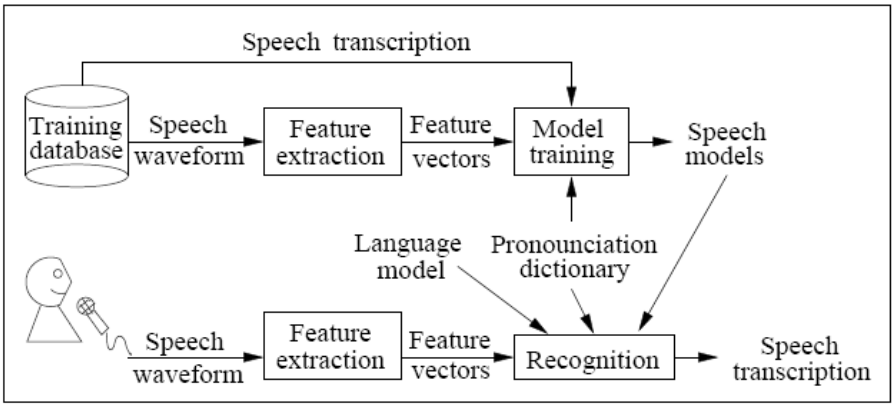
Og det trenger selvfølgelig ikke ende der. Hvordan i alle dager skal en datamaskin kunne tolke dette på samme måte som et menneske? Vi tar diskusjonen litt videre og sier at tvetydighet kan oppstår i følgende deler i læren om tale og språk:

- Fonetikk (Phonetics) og Fonologi (Phonology). Studien av lingvistiske lyder (tale).
- Morfologi (Morphology). Studien av meningsfylte ord.
- Syntaks (Syntax). Studien av strukturelle forhold mellom ord.
- Semantikk (Semantics). Studien av mening.
- Pragmatikk (Pragmatics). Studien av hvordan språk brukes til å nå et mål.
- Diskurs / Samtaler (Discourse). Studien av lingvistiske enheter større enn én ytring.

Sleng på ironi, sarkasme, dobbelmoral, personlige særegenheter osv. Poenget er følgende: *Hvordan* hjernen tolker tale har vi nærmest ikke den fjerneste anelse om, men kan vi lage noe som fungere godt nok?

5.1.2 Modeller og metoder.

Det finnes mange forskjellige angrepsvinkler og filosofier på området, dog er det visse retningslinjer som verserer. Vi trekker opp en generell modell for å illustrere:



Figur 5.1.2 (1)
Generell modell av talegjenkjenningsalgoritmers struktur.

I bunnen av algoritmen ser vi blokkene kalt ”feature extraction”. Her dreier det seg om å foredle lydsignalet på en slik måte at vi får fjernet alt av informasjon som ikke direkte har noe med det talte ordet å gjøre. I øvre halvdel benyttes statistiske modeller, dynamisk programmering og neurale nett for å håndtere og forme data på en slik måte at algoritmen med tiden blir stadig flinkere til oppgaven sin. ”Recognition” boksen kan inneholde så mangt, men generelt gjelder det at den sammenligner fonemer eller stavelser med den modellen som enhver tid er gjeldende. Deretter benyttes N-gram metoder som for eksempel Hidden-Markov og Viterbi-algoritmer for å kalkulere sannsynligheten for hvilke ord som blir snakket. Ordet algoritmen antar det er størst sannsynlighet for er det faktiske, velges omsider ut.

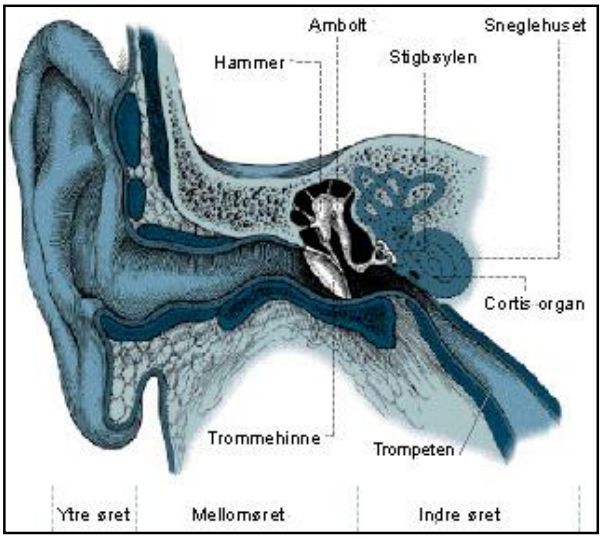
I denne oppgaven har vi verken tid eller ressurser til å gyve løs på hele systemet. Vi har i utgangspunktet konsentrert oss om egenskapsuthentingsboksen og knyttet denne sammen med en kodebok og en autokorrelasjonsalgoritme. Vi har også utviklet en generell datastruktur som passer formålet samt verktøy for å håndtere de mengder av data fort får for hånden.

5.1.3 Om egenskapsuthenting / foredling av det talte signalet

Det er særlig to fremtredende ideer om hvordan man best angriper ”feature extraction” boksen. Den ene omhandler talesystemet og hvordan man kan benytte viten om denne til å generere sine egne ord man kan bruke som ”referanse” ved senere gjenkjenningsalgoritmer. Den andre tar for seg viten om hvordan øret filtrerer signalet slik at hjernen kun skal forsynes med informasjonen som er vesentlig for å identifisere tale. Begge metoder benyttes i stor utstrekning, og hva man velger virker å være likegyldig. Vi har grunnet noe bedre kilder og en generell oppfatning av at sistnevnte metode gir en noe raskere implementering valgt å gå for nettopp denne.

5.2 Egenskapsuthenting

5.2.1 Modellering av øret.



Figur 5.2.1 (1)
Viser de fysiske aspektene ved øret.

Hørselssystemet kan deles opp i et ytre og et indre system. Det ytre systemet er den mekaniske delen av øret, og fungerer i hovedsak som et filter som modifiserer lydbølger fra omgivelsene slik at disse blir overkommelige for hjernen å håndtere. Det indre hørselssystemet er mer rettet mot hvordan hjernen mottar og behandler impulsene den mottar, og den generelle viten på dette området er svært mangelfull.

Det ytre mekaniske hørselssystemet består av ytre tre øret, mellomøret og det indre øret. De to førstenevnte har som hovedmål å fange opp og forsterke lydbølgene. Når lydbølgene når trommehinnen begynner denne å vibrere. Vibrasjonene forplanter seg videre via hammer, ambolt og stigbøylen til det indre øret.

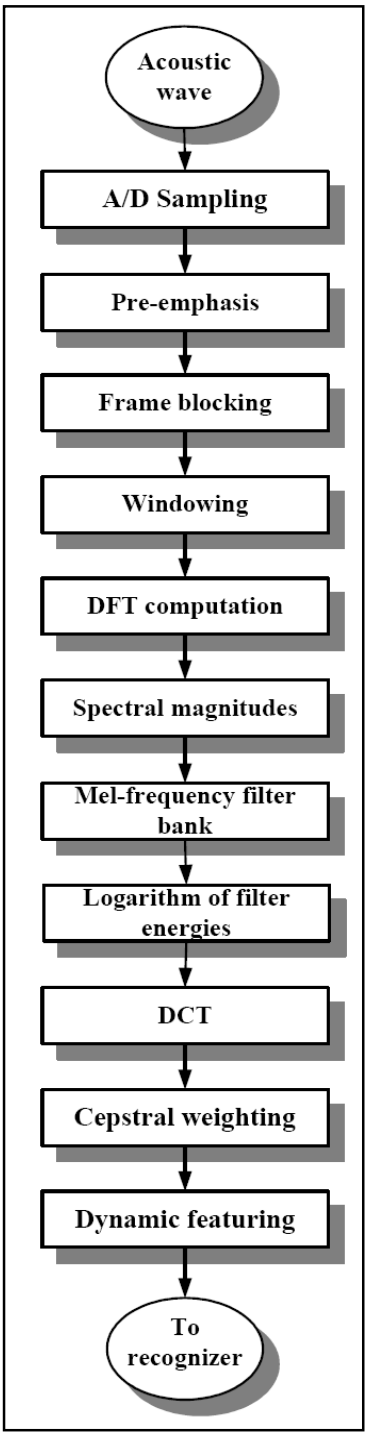
Det indre øret er en væskefylt kanal formet som et sneglehus. Veggene på innsiden av denne kanalen er fylt med mange små flimmerhår. Når vibrasjonene forplanter seg fra mellomøret begynner væsken å bevege seg. Flimmerhårene oversetter denne bevegelsen til elektriske *impulser* som sendes til hjernen.

Et argument som unektelig virker svært godt er at det ytre øret gjennom evolusjonen nok er utformet på en slik måte at hjernen mottar svært gunstige impulser fra det ytre hørselssystemet. Vi kan forstå hva som blir sagt selv om signal/støy forholdet er under 0dB. Med dette argumentet i bakhånd blir det naturlig å spørre seg hvorfor man ikke bare kan lage et filter som utfører de samme operasjoner på tidssignalet som det ytre øret ville gjort? Klarer man dette sitter man jo tross alt igjen med akkurat den samme informasjonen som hjernen mottar og tydeligvis greier seg fint med.

Helt i tråd med dette har vi altså valgt å bygge talegjenkjenningsdelen rundt en ”egenskapsuthenting” som ene og alene har som oppgave å ligne mest mulig på det ytre ørets egen filterfunksjon. Rekkefølgen for implementering av de forskjellige operasjonene følger stort sett den vandringen signalet gjør fra det ytre øret til hjernen. La oss ta for oss nettopp denne vandringen.

5.2.2 Forbehandling.

Se figur 5.2.2 (1) hvilke trinn egenskapsuthentingsboksen inneholder. For å starte på toppen; datainnsamlingsdelen består av en mikrofon og ADC som digitaliserer lyden. Vi benytter oss av et DSP utviklingskort som har egen ADC og egen mikrofoninngang. Samplingshastigheten ble lagt til 24kHz og ordlengden til 16-bits. Såfremt man har prosesseringskraft nok er det aldri ungustig med høy oppløsning. Dette gir lavere standardavvik på kvantiseringsstøyen noe som er meget fordelaktig når vi ved gjentatte anledninger multipliserer verdiene med hverandre. Desto flere kalkulasjoner på de samme verdiene desto større utslag til kvantiseringsstøyen få. Høy samplingshastighet er også gunstig da man slik ved å "midle ned" frekvensen i etterkant kan nytte seg av sentralgrenseteoremet, desto større datasamlinger i empiriske forsøk desto bedre nøyaktighet.



Figur 5.2.2 (1).
Oversikt over de vanligste punktene i "Feature Exraction".

5.2.3 Pre-emphasis.

Lyden forvrenges noe av den fysiske utformingen av øret. Særlig de høyere frekvensene dempes noe. Dette skal vi kompensere for i pre-emphasis filteret. Vi benytter et førsteordens FIR filter for dette.

$$H(z) = 1 - \alpha z^{-1} \qquad 0.9 \le \alpha \le 1.0$$

Ligning 5.2.3 (1). FIR variant representert Z-transformert.

$$\tilde{s}[n] = s[n] - \alpha s[n-1]$$

Ligning 5.2.3 (2). FIR variant representert med differensligning.

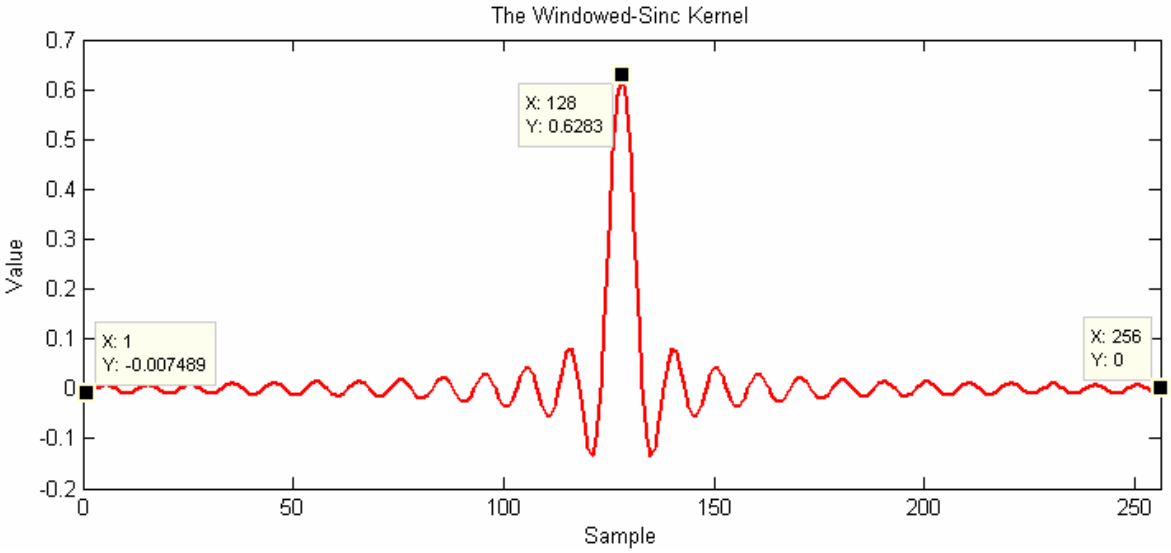
α er pre-emphasis parameteren. Typisk verdi er 0.97. Ved å benytte oss av denne metoden får vi en 20dB forsterkning i de øvre frekvensene og 32dB ved Nyquist frekvensen.

5.2.4 Frame blocking og windowing.

Et ideelt filter har både ideell step-, impuls- og frekvensrespons. En step-respons er ideell når flanken er uendelig bratt (dvs. filteret har ingen tidsforsinkelse), er uten "overshoot" og har symmetrisk fase (oppadgående og nedadgående flanke er like). Impulsresponsen er egentlig bare den deriverte av stepresponsen og vi ønsker altså at arealet under den skal være nøyaktig lik 1, at den er uendelig høy og uendelig smal, og at den ikke innfører oscillasjoner ellers. I frekvensplanet ønsker vi å skape så gode skiller mellom frekvensbånd som mulig, og at båndene ikke blir forsterket eller dempet av filteret for øvrig.

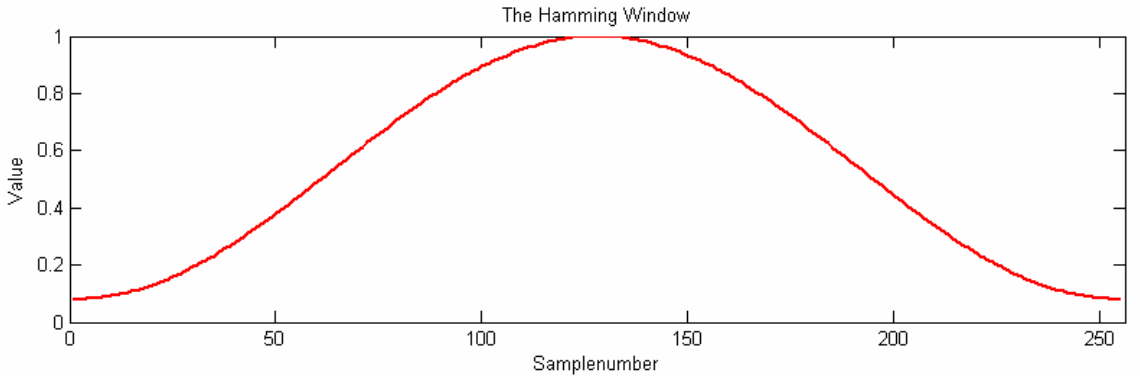
For god frekvensrepons kan vi benytte oss av windows-sinc filteret. Her tenker man seg at en ideell frekvensrespons oppnås når man multipliserer hele frekvensspekteret med et rektangel. Denne operasjonen tilsvarer å folde tidssignalet med en sinc-funksjon som er symmetrisk om tiden 0. Siden man gjerne regner verden som kausal er ikke negative tider noe vi ønsker å operere med. Vi forskyver derfor hele sinc-funksjonen fram i tid slik at vi kan adressere med kun positive indekser. Ved å gjøre dette får man imidlertid litt "frynsete"

kanter på frames'ene, og med på lasset en del uønskede effekter. Vi multipliserer derfor med et eller annet slags vindu for å "pynte" på endene av den forskyvde windows-sinc funksjonen. Det optimale vinduet å benytte i forbindelse med vår type oppgave er Hamming vinduet, ergo er dette benyttet.
I oppgaven valgte vi og bruke frames på 256 sampler men vi lot være å overlappe dem. Vi forkastet en av de 256 samplene i hvert samplesett, og sentrerer rundt sampel 127. Den ene sampelverdien vi forkastet kan betraktes som en 0-padding og vi vil derfor ikke påvirke frekvensresponsen. På denne måten lager vi et bratt digitalt lavpass samtidig som vi oppnår lineær fase. For de som ikke har helt kontroll på hvordan sinc-funksjonen ser ut kan vi herved informere om at den ser slik ut:



Figur 5.2.4 (1)
En sinc-kjerne hvor knekkfrekvens er valgt lik 0.4Fs. Sinc-funksjonen er skiftet 128 sampler ut i datasettet for å unngå negative indekser på samplene.

Legg merke til at grafen ikke går til 0 i høyre side av vinduet. Slike småfeil er svært kritiske da vi ved en folding har løpende summer der slike småfeil dras med gjennom hele prossesen og adderes opp til slik at de blir særdeles uheldige. Løsningen er som nevnt å multiplisere med et vindu for å "pynte på" endene. Det valgte Hammingvinduet har en størrelse valgt lik window-sinc kjernen slik at disse blir lette å multiplisere sammen.



Figur 5.2.4 (2)
Hamming vinduet, har en størrelse som matcher window-sinc kjernen.

$$h[n] = \begin{cases} 0.54 - 0.46 * \cos(2\pi n / N_0) & 0 \le n \le N_0 \\ Ellers\ 0 \end{cases}$$

Ligning 5.2.4 (1)
Hammingfunksjonen. Ligninger er alltid kjekt.

5.2.5 Discrete Fourier Transform

I neste steg ønsker vi å hente ut frekvensinformasjonen til tidsdomenets data. Utgangspunktet er basisfunksjonen:

$$X(k) = \sum_{n=0}^{N-1} x[n] e^{-j2\pi nk/N} \qquad 0 \le k \le N$$

Ligning 5.2.5 (1)

I prinsippet dreier det seg enkelt og greit om å vekte tidsdomenet med basisfunksjoner. Vi har her implementert the mer "regnekraftvennlige" Fast Fourier Transform(FFT). Ved bruk av FFT sparer vi en del operasjoner, mens DFT krever N^2 operasjoner, krever FFT bare $N\log_2 N$ operasjoner.

5.2.6 Spectral magnitudes

Generelt er signal X(k) en kompleks verdi bestående av en reell del og en kompleks del. Når det kommer til talegjenkjennelse blir de komplekse verdiene nesten alltid ignorert. Derfor blir bare størrelsen av den komplekse verdien av X(k) utnyttet i denne situasjonen. Hvis vi antar at reelle og imaginære delen av X(k) er henholdsvis Re(X(k)) og Im(X(k)) , så blir absoluttverdien av X(k):

$$\left|X(k)\right| = \sqrt{\mathrm{Re}(X(k))^2 + \mathrm{Im}(X(k))^2}$$

Lingning 5.2.6(1)

5.2.7 Mel-frequency filterbank

For å representere de statisk akustiske egenskapene, Mel-Frequency Cepstral Coefficient(MFCC) brukes som de akustiske kjennetegn i det cepstral domain. Dette er et fundamentalt konsept som bruker et sett med ikke-lineære filtre til å beregne oppførselen til et audio system. Det menneskelige øret hører frekvenser stort sett mellom 300Hz til 3400Hz, systemet er laget for å herme etter disse egenskapene. Dessuten er det menneskelige øret mer følsomt og har høyere oppløsning på lave frekvenser. Derfor burde filterbanken være slik at den vekter lave frekvenser høyere enn de høye frekvensene.

Filtrene i filterbanken er trekantformet som vist på fig5:

$$H_m(k) = \begin{cases} 0, & k < f(m-1) \\ \dfrac{k - f(m-1)}{(f(m) - f(m-1))}, & f(m-1) \le k \le f(m) \\ \dfrac{k(m+1) - k}{(f(m+1) - f(m))}, & f(m) \le k \le f(m+1) \\ 0, & k > f(m+1) \end{cases}$$

Ligning 5.2.7(1)

som oppfyller:

$$\sum_{m=0}^{M-1} H_m(k) = 1$$

Ligning 5.2.7(2)

Cepstralfrekvensen av hvert mel-skalert filter har en uniform fordeling under 1kHz og følger en logaritmisk fordeling over 1kHz.. Flere filtre brukes på frekvenser under 1kHz siden det meste av den brukbare informasjonen ligger her.

$$\mathrm{Mel}(f) = 2595\log_{10}(1 + \frac{f}{700})$$

Ligning 5.2.7(3)

Hvis vi definerer fl og fh til å være laveste og høyeste frekvens i filterbanken i Hz, Fs er samplingsfrekvensen i Hz, M er antall filtre og N er størrelsen på FFT, senterfrekvensen f(m) av mth filterbank

$$f(m) = \left(\frac{N}{F_s}\right) Mel^{-1}\left(Mel(f_l) + m\frac{Mel(h_h) - Mel(f_l)}{M+1}\right)$$

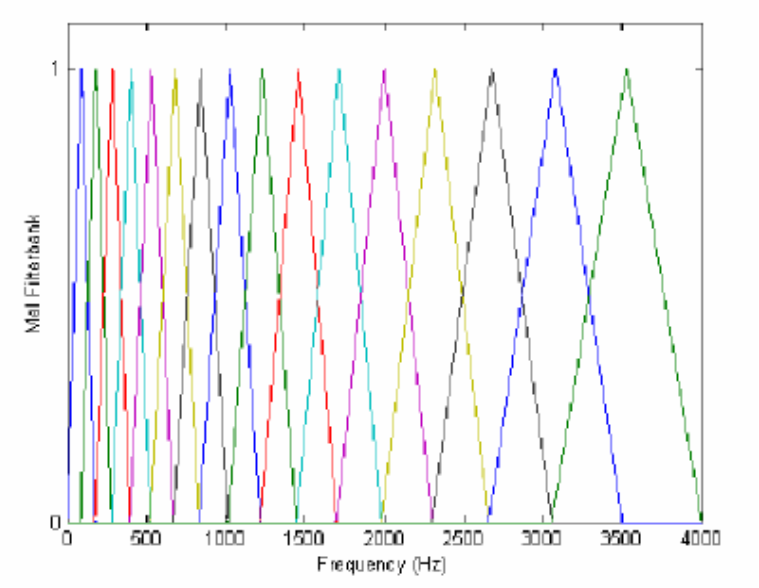
Ligning 5.2.7(4)

hvor Mel-scale Mel er gitt ved Mel frequency

Mel^{-1} er gitt ved.

$$Mel^{-1}(b) = 700(10^{b/2595} - 1)$$

Ligning 5.2.7(5)



Figur 5.2.7(1)::viser formen på filtrene i filterbanken. Her ser man også tydelig hvordan bredden på filtrene strekker seg ut etter hvert som frekvensen øker.

5.2.8 Logaritmen av filterenergier

Etter at signalet har passert igjennom filterbanken beregnes den logaritmiske verdien av hvert filter.

$$s(m) = \log_{10}\left[\sum_{k=0}^{N-1}|X(k)|^2 * H_m(k)\right], 0 \le m \le M$$

Ligning 5.2.8(1)

Det menneskelige øret glatter også ut spektrumet på en måte som ligner den logaritmiske skalaen.

5.2.9 Discrete Cosinus Transform

Etter filterbanken skal vi tilbake i tidsdomene. For å utrette dette tar vi i bruk Discrete Cosinus Transform(DCT). Cosinustransform har mange fellestrekk med fouriertransform. Men mange foretrekker denne metoden fordi man kan korte ned på behovet for datakraft. I praksis betyr det at man kan beskrive en større del av signalet med færre koeffisienter. Dessuten transformerer den reelle signaler til reelle signaler.

Hvis vi skulle ha brukt Invers Fourier transform måtte vi i så fall tatt vare på den komplekse delen av signalet. Ett typisk talegjenkjennelsesprogram bruker 10-15 koeffisienter. Vi bruker 24 siden vi regner med at vi har datakraft til å håndtere dette.

$$C(k) = \sum_{m=0}^{M-1} S(m) * \cos(\pi * k(m + 0{,}5) / M) \ ,0 \le k \le K$$

Ligning 5.2.9(9)

DCT ble i utgangspunktet brukt for å skille mellom vokaltrakten og glottis (stemmebåndspalten i strupehodet) Disse to komponentene blir additive i ceptraldomenet, og kan med hell fjernes ved å kjøre windowing i ceptraldomenet. Denne prosessen blir kalt lifting(løfting)

5.2.10 Dynamic Feature(Dynamiske egenskaper)

Denne delen tar for seg det faktum at ordene som ytres ikke alltid er like i tid. Stavelser blir ikke sagt på samme måte med samme lengde. Ved å innføre den deriverte eller høyere ordens deriverte kan man ta hensyn til disse endringene. Det vanligste er å bruke førsteordens dynamiske egenskaper, altså den deriverte av de statiske signalene, også kalt deltaegenskaper. Statiske signaler er de signalene vi fikk ut etter DCT. Teoretisk sett kan man finne deltaegenskapene ved å finne differansen mellom to frames, men dette gir ofte et resultat med mye støy. Det som ofte er lurt er å benytte et glattet estimat som beregnes over flere etterfølgende frames. Her kan man for eksempel velge å beregne over de tre neste og de tre foregående frames fra den statiske egenskapen. Hvis man velger å bruke høyere orden dynamiske egenskaper beregnes disse fra de lavere ordens egenskaper. Når det gjelder hvor mange estimat man vil ha med er det vanlig å velge like mange som de statiske. Altså hvis man har 24 statiske verdier velger man å beregne 24 delta- og 24 dobbeltdeltaegenskaper.
Vi valgte og ikke gå inn på denne delen, mest fordi det ikke var tid, men også fordi de statiske dataene fortsatt er de beste. Hvis man bare bruker dynamiske egenskaper får man ofte et dårligere resultat. Det aller beste resultatet oppnås ved bruke statiske verdier sammen med de dynamiske.

5.2.11 Andre metoder for å forbedre resultatet

Andre metoder for å forbedre N/S forholdet. Effektiviteten av metodene er mye bestemt av type støy. Våres program er beregnet på å fungere i under noenlunde ideelle forhold, så disse metodene hadde bare vært nyttige hvis vi ønsket å gjøre algoritmen våres mer robust. Dessuten virker det som filtrene våres tidligere i oppgaven effektivt fjerner mye av støyen.

5.2.11.1 Ceptralmidling

Også kalt Ceptral Mean Normalization(CMN). Målet er å fjerne konstant eller saktevarierende konvolvert støy. Eksempler er støy fra mikrofon og romakustikk.
CMN benytter seg av at konvolusjon er additivt i ceptraldomenet. De betyr at en konstant støykanal vil være additivt konstant i ceptraldomenet. Man kan derfor midle i ceptraldomenet og trekke gjennomsnittet fra alle statiske egenskaper. Teknikken fungerer best hvis man midler over en lengre tid.

5.2.11 Relativ Spectral metode(RASTA)

Dette er en annen metode for å fjerne støykilder. Man har funnet ut at tale har en modulasjonsfrekvens som ligger noe mellom 4-16Hz. På grunn av fysiske forutsetninger er det lite trolig at stemmebåndene kan ligge noe særlig over. Derfor kan man vha en båndpass fjerne uønsket støy. Metoden er likevel ikke så veldig effektiv hvis det forligger mye konvolvert additivt støy.

5.2.12 Gjenkjennelse

Veldig generelt lagrer denne delen unna resultatet fra forbehandlingsdelen av programmet i et dynamisk bibliotek. Det vil si at biblioteket "lærer" mens det brukes, dette er nødvendig fordi selv ikke den samme personen kan uttale det samme ordet helt likt fra gang til gang. Biblioteket beregner et snitt som brukes til sammenlikning, og dette oppdateres etter hver gang ordet uttales. Prosessen som sammenlikner de forskjellige ordene er stort sett bygget på tunge statistiske beregninger. Det finnes utallige måter å ta for seg denne delen, man kan bruke delvis ferdige biblioteker som finnes i Matlab eller Windows, eller man kan prøve å finne en egen løsning. Kjente oppskrifter for å utføre dette i praksis kan være:

Denne delen av prosessen har vi i liten eller ingen grad berørt i denne oppgaven. Omfanget av teorien er ganske enkelt for stort i forhold til tidsrammene. Vi valgte å gå så langt som å transformere signalet tilbake til tidsdomene ved hjelp av DCT og deretter kjøre en simpel korrelasjon med et datasett som har gått igjennom samme prosess. Deretter har vi bare sett på hvilket ord som har gitt den høyeste amplituden etter korrelasjonen og gått ut fra at det er det riktige ordet.

5.2.13 Testresultater/diskusjon

Systemet viser visse svakheter når man prøver å sammenligne med korte ord. Det virker som disse har en tendens til og slå ut med høye amplitude etter korrelasjonen. En mulig årsak er at korte ord har en temmelig konsentrert energi over et kort tidsrom. Dette vil igjen føre til at det vil se ut som match hvis et annet kort ord blir ytret. Igjen har vi hatt stort helle med ord som er omtrent like lange men som har energiene spredt litt mer spredt over spekteret. For å forbedre dette vil et naturlig valg være å jobbe videre med de forskjellige metodene som vi ikke kom inn på. Ved å benytte dynamiske egenskaper vil nok resultatet forbedre seg noe men den største biten er nok å velge en annen måte og sammenligne ord enn og bare autokorrulere signalene etter at vi er tilbake i tidsdomenet. Som tidligere nevnt finnes det utallige metoder og gjøre dette på, men siden vi i liten grad har satt oss inn i dette stoffet er det vanskelig å si hvilke alternativer som hadde vært naturlig å velge.

Til slutt kommer også en autokorrelasjonsalgoritme benyttet for å få et håndfast resultat å vise til. Det er imidlertid enorme forbedringspotensialer her, da dette er nettopp hva det indre hørselssystemet "dreier seg om". Teoriene og hypotesene her bygger seg mer på tunge statistiske analyser som implementeres i dynamisk programmerte neurale nett (selvlærende algoritmer). Vi var imidlertid nødt til å sette grensen et sted så dette kommer vi verken til å implementere eller drøfte konsekvensene av.

6.
KONKLUSJON/DISKUSJON

6.1 Diskusjon

Denne diskusjonen er beviset på at vi nå er ferdig med vårt utviklingsprosjekt, Voice Control. Teamarbeidet vi har lagt ned i prosjektperioden har i våre øyne vært tilnærmet optimalt. Vi har jobbet med hver våre fraksjoner av oppgaven, men har likevel fungert som en enhet. Det selvstendige arbeidet har aldri tatt fra oss følelsen av å være del av et arbeidende team,

Når du velger prosjektoppgave ved en Høgskole, er det innebygget i underbevisstheten at det er læringsutbyttet av prosjektet som er det viktigste. Det var ikke uten grunn at vi tok den oppgaven som ble regnet som den mest utfordrende i årets utvalg. Oppgaven ble presentert som en utviklingsoppgave, og det var nettopp dette aspektet som tente alle deltakerne umiddelbart. Fra dag én startet vi med å utvikle grunnideer, og stadig bygge på disse den første tiden. Alles vilje til å fullføre prosjektet med best tenkelig resultat, har gjort at gruppen som helhet har fungert utmerket. Vi har alle hele tiden jobbet med forskjellige aspekter av prosjektet, ofte helt selvstendig, men likevel klart å holde fokus på et endelig produkt. Fokuset har i tillegg til å motivere til innsats, økt viljen til å sette seg inn i nytt stoff. Denne læringsprosessen var noe av det vi i utgangspunktet ønsket å oppnå, og som vi håpet skulle ligge i en utviklingsoppgave.

I starten hadde vi ingen muligheter til å se det fulle omfanget av oppgaven. Først etter en omfattende research fase på ca. to og en halv måned, innså vi det totale omfanget av oppgaven. Dette sier litt om vår uerfarenhet i utviklingssammenheng, men mest om hvilket enormt område talegjenkjenning representerer. Talegjenkjenning er et av de områdene innenfor moderne teknologi med størst utviklingspotensiale. Det foreligger et utall forskningsrapporter på ulike tilnærminger til området, uten gode konkrete resultater på et fungerende system.

6.2 Konklusjon

Vi har klart å utvikle et verktøy for talegjenkjenning, som under visse forutsetninger klarte å gjenkjenne den taleinput vi gav systemet. Programmet ble ikke så robust som vi hadde ønsket, men kunne gjenkjenne 3 ulike ord med temmelig stor sikkerhet. For at robustheten skulle vært på et tilfredsstillende nivå, måtte vi sannsynligvis ha implementert en algoritme for selve gjenkjennelsen. Vi har stort sett tatt for oss den delen av talegjenkjennelse som går på å hente ut visse egenskaper fra signalet. Algoritmen for denne signaluthentingen kan sees i sammenheng med den ytre delen av ørets virkemåte. Verktøyet vi har utviklet gir oss altså et bilde av talesignaler, som om det skulle vært et digitalt øre. Det vi imidlertid mangler, er den delen som tar for seg videre signalering mot hjernen. Dette området er så stort og avansert, at det ikke foreligger noen god metode for å få til dette teknisk i dag. De algoritmene som eksisterer bygger på tunge statistiske beregninger, som ligger langt utenfor grensene for vår kompetanse og vårt fagfelt. Basisen er store biblioteker, som kun finnes i store software systemer som Windows, og enda er det ting å hente på virkemåten. Hovedlinjen er at denne siste delen, ikke har noe som helst med det vi utdanner oss til å gjøre, og er på et for høyt nivå.

Når det gjelder hardware delen, var vi for optimistiske i starten. Testkortet vi skulle utvikle vår egen versjon av var i utgangspunktet for komplisert, noe vi ikke innså før et stykke ut i prosjekt perioden. Vi gikk tidlig over de grensene skolens software kunne takle i kompleksitet, og måtte skaffe utbedrede versjoner av dette. Testkortet vi brukte fungerte bra til sitt formål, men var veldig komplekst å forstå. Med 50 000 dokumentasjon på forskjellige registre og komponenter, og en compiler hvor C, C++ og Assembly brukes om hverandre, ble det brukt masse unødvendig tid og krefter på småting. Hadde vi hatt noen på skolen som kunne gitt mer direkte veiledning i bruk av dette programmet, hadde det hjulpet veldig. Konklusjonen på hardware seksjonen blir altså at vi ikke fikk utviklet en egen prototyp, som vi hadde ønsket innledningsvis.

LonWorks biten hadde vi bra framgang på, helt til det stoppet opp når vi skulle ha tak i en brenner til 3120 chipen. Vi utviklet imidlertid et grensesnitt mellom testkortet og Neuron Chipen, og testet dette til riktig funksjonalitet. Vi kunne fint styre output med de kommandoene vi sendte fra testkortet. Hadde vi fått brenneren noen uker tidligere, kunne vi testen interfacet ut mot LonWorks nettverket. Så lenge koden kommuniserer tilfredsstillende med testkortet i en tenkt setting, så har vi oppnådd grunnmålet i denne sammenhengen.

Det siste målet vi hadde, var på et personlig plan. Vi ville lykkes med en avansert utviklingoppgave, og kjenne følelsen av å ha avsluttet utdannelsen med bravur. I stedet sitter vi igjen med en litt bitter smak i munnen. Prosjektet underveis har vært en utrolig opplevelse for oss alle, hvor vi har oppnådd målene rundt teamarbeid mot et felles mål. Utviklingen fagmessig har også vært på et høyt nivå, og totalt sett har vi hatt et utrolig læringsutbytte på alle fronter. Eneste skåret i gleden kom de siste dagene. Feilberegning av tidsbruk på rapporten gjorde at vi ikke fikk med alt stoffet vi skulle, og noe av det som ble lagt ved, ikke var kommentert så fyldig som ønskelig. Et prosjekt som kunne vist vår kapasitet helt inn, endte i stedet i en skuffelse over at vi måtte levere en rapport som absolutt ikke lever opp til den standarden vi vanligvis holder. Men det er en del av den prosessen som kalles læring, og det er ofte de mest smertelige feil man lærer mest av.

Dette med tidsbruk er alltid et tilbakevendende tema i hovedprosjekter, men det er så vanskelig å ha oversikt på alle fronter. Vi har sittet kontinuerlig de siste ukene med både hovedprosjekt og eksamener, og hadde håpet å få full uttelling. Det klarte vi imidlertid ikke, men regner med at denne lærdommen sitter i til framtidige storprosjekter.

7.
KILDER

Fra nett:

www.echelon.com
www.dspguide.com
http://www.sundance.com/web/files/productpage.asp?STRFilter=TMDSCCSALL-1
www.fysikknett.no
http://www.dsprelated.com/groups/code-comp/11.php
http://www.eng.iastate.edu/ee424/labs.htm http://www.cs.aue.auc.dk/~akbar/2006/dsp-1-06.html
http://www.ti.com
http://umsis.miami.edu/~tplummer/PROJECT.pdf
http://www.bioingenieria.edu.ar/grupos/cibernetica/milone/download.htm
http://en.wikipedia.org/wiki/Divide_and_conquer_algorithm
http://en.wikipedia.org/wiki/Cooley-Tukey_FFT_algorithm
http://en.wikipedia.org/wiki/Fast_Fourier_transform
http://en.wikipedia.org/wiki/Twiddle_factor
http://en.wikipedia.org/wiki/Butterfly_diagram
http://www.cmlab.csie.ntu.edu.tw/cml/dsp/training/coding/transform/fft.html
http://www.relisoft.com/Science/Physics/fft.html
http://recherche.ircam.fr/projects/cuidado/wg/management/deliverables/IRCAM_D211_M2.pdf

Bøker:

- Speech and language processing, Daniel Jurafsky & James H. Martin
- Echelon Neuron C Programmers Guide
- Motorola LonWorks
- Speech Processing, edited by Chris Rowden
- Fundamentals of Speech Recognition, Lawrence Rabiner & Biing-Hwang Juang
- Computer Speech Processing, edited by Frank Fallside & William A. Woods
- Signals & system second edition, Alan V. Oppenheim & Alan S. Willsky

Annet:

- Dokumentene som fulgte med CD fra Texas og hjelpefunksjonen i CCS.
- CD DSP teaching room
- (Pdf) Feature Extraction for Automatic Speech Recognition in Noisy Acoustic Environments.
- (Pdf) Signalrepresentasjoner for Automatisk Talegjenkjenning

VEDLEGG A
KODE TALEGJENKJENNING

A.1 Filen Stemmegjenkjenning.cpp

```
  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Comments                                    ///
////////////////////////////////////////////////
                                                                                    #pragma region
/*
Last edit: 30.5.2007, 13:55 gmt.

<<General>>
This code is meant implemented on a Texas Instruments 320C6416 DSP, as a voice recognition
software. It has some functionality like the 'iostream' related functions 'cout' and 'cin',
the 'stdafx.h' header and 'fstream' functionality that probably will need to be removed on most
DSP's.

<<Special occurences>>
The 'stdafx.h' header is Visual Studio 2005 specific, but can be removed whenever one
emigrate to another system. The same goes for the '#pragma regions' which is a system for code
outlining. I kept them present as the reader might have the Visual Studio Express 2005 software
installed. If not it is downloadable for free at www.microsoft.com. The _getch_nolock() function
declared and defined in 'conio.h' has one purpose only; break the program until a key i pressed.

<<Structure>>
I chose to split the code up in many "sections", each section being represented by an own file.
Look at the comments where I include the 'DataSet.h' file. The general philosophy is relative
simple, the first come first serve applies. The Complex is a class used by ComplexMatrix, therefore
the file 'Complex.h' is included from the 'Matrix.h' file asf.

<<Tweaking>>
To add flexibility while still preserving the overall structure and readability of the code, i chose
to put the adjustable variables with the functions that take them as parameters. In this part of
program you can find the feature extraction parameters in the FeatureExtraction() function for
instance. Constants that apply to several parts of the program are defined first on the file,
right after the necessary headerfile includes.

For more details the reader is advised to look up the report accompanying this code.

Written by Jo Inge Buskenes,
with lots of help and general support from team VoiceControl. :)
*/
                                                                                    #pragma endregion
  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Include                                     ///
////////////////////////////////////////////////
                                                                                    #pragma region
#include "stdafx.h"     //Visual Studio specific! Remove when not in this environment.
#include <iostream>     //Basic IO, cin / cout etc.
#include <iomanip>      //Setprecision(), for access to file- / screenstream etc.
#include <fstream>      //File IO, fstrean, ifstream, ofstream.
#include "stdlib.h"     //abs().
#include "math.h"       //sin(), cos(), ...
#include <conio.h>      // _getch_nolock()

#include "DataSet.h"    //All the functionality used when dealing with the data in this project.
                        //Contains (listed in the order they are included, from last to first):
                        //DataSet -> a class which holds time/frequency domain + related functions.
                        //Toolbox -> a class which acts as a container for datarelated operations.
                        //Matrix  -> contains the class ComplexMatrix, the basic structure of data.
                        //Complex -> a class which acts as a complex number and holds the operator
                        //           overloads to imitate the complex number nature.

using namespace std;    //std:: is the standard namespace.
                                                                                    #pragma endregion
```

```
  ///////////////////////////////////////////////////////////////////////////////////////////////////
 /// Global Variables                            ///
///////////////////////////////////////////////////
                                                                                   #pragma region

List *liCodebook = new List;                    //This is where all data from the time-domain and
                                                //frequency-domain will be stored. The objects it
                                                //holds is of type ComplexDataSet.

                                                                                   #pragma endregion

  ///////////////////////////////////////////////////////////////////////////////////////////////////
 ///Global Function Declarations                 ///
///////////////////////////////////////////////////
                                                                                   #pragma region

void ReadCodebookFromFile();                    //Reads codebook from 'codebook.txt'.
void WriteCodebookToFile();                     //Reads cobebook to   'codebook.txt'.
void ReadWordFromFile(char *cWordName);         //Reads a word from  'lyddata.txt'.
void WriteMenu();                               //Write a fancy menu to screen.
void CompareWord();                             //Compare codebook words with the one in lyddata.txt
void NewWord();                                 //New word? Put it in 'lyddata.txt' and off we go!
void ViewCodebook();                            //Writes codebook to screen.
void FeatureExtraction();                       //The very essence of the voicerecognition algorithm

                                                                                   #pragma endregion

  ///////////////////////////////////////////////////////////////////////////////////////////////////
 /// Main                                       ///
//////////////////////////////////////////////////

int main()
{
  ReadCodebookFromFile();                       //First we read the codebook from file.

  char cChoice;

  do
  { WriteMenu();                                //Then we enter the menu.
    cin >> cChoice; cin.ignore();              //The user have the following options:
    switch (toupper(cChoice))
    {
    case ('C'): CompareWord();  break;          // 'C' - Compare word in 'lyddata.txt' with codebook.
    case ('N'): NewWord();      break;          // 'N' - Read word from 'lyddata.txt' into codebook.
    case ('V'): ViewCodebook(); break;          // 'V' - Write the whole codebook to screen.
    }
  }while ( (toupper(cChoice) != 'Q') );         // 'Q' - Quit and store the current codebook to file.

  WriteCodebookToFile();

  return 0;
}

  ///////////////////////////////////////////////////////////////////////////////////////////////////
 /// Global Function Definitions                ///
//////////////////////////////////////////////////

  //-------------//
 // Informative //
//-------------//

void WriteMenu()                                                   //  Skriver ut menyen, evt. hjelpemenyen.
{        cout <<
"\n\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\"
       << "\n
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\"
       << "\n  \\\\\\\\\\\\\\\\\\\\     ¤¤¤       ¤¤¤    ¤¤¤¤¤¤¤     ¤¤¤      ¤¤¤¤¤¤   ¤¤¤¤¤¤¤   \\\\\\\\\\\\\\\\\\\\"
       << "\n   \\\\\\\\\\\\\\\\\\\\     ¤¤¤     ¤¤¤    ¤¤¤   ¤¤¤    ¤¤¤     ¤¤¤       ¤¤¤        \\\\\\\\\\\\\\\\\\\\"
       << "\n    \\\\\\\\\\\\\\\\\\\\     ¤¤¤   ¤¤¤    ¤¤¤     ¤¤¤   ¤¤¤    ¤¤¤        ¤¤¤¤¤¤      \\\\\\\\\\\\\\\\\\\\"
       << "\n     \\\\\\\\\\\\\\\\\\\\     ¤¤¤ ¤¤¤      ¤¤¤   ¤¤¤    ¤¤¤     ¤¤¤       ¤¤¤          \\\\\\\\\\\\\\\\\\\\"
       << "\n      \\\\\\\\\\\\\\\\\\\\     ¤¤¤¤¤        ¤¤¤¤¤¤¤     ¤¤¤      ¤¤¤¤¤¤   ¤¤¤¤¤¤¤       \\\\\\\\\\\\\\\\\\\\"
       << "\n      /////                                                                   /////"
       << "\n     /////    ¤¤¤¤   ¤¤¤¤   ¤¤  ¤¤  ¤¤¤¤¤¤ ¤¤¤¤¤    ¤¤¤¤   ¤¤          /////"
       << "\n    /////    ¤¤      ¤¤  ¤¤  ¤¤¤ ¤¤    ¤¤    ¤¤  ¤¤  ¤¤  ¤¤  ¤¤         /////"
       << "\n   /////     ¤¤      ¤¤  ¤¤  ¤¤ ¤¤¤    ¤¤    ¤¤¤¤    ¤¤  ¤¤  ¤¤        /////"
       << "\n  /////       ¤¤¤¤   ¤¤¤¤   ¤¤  ¤¤    ¤¤    ¤¤ ¤¤¤   ¤¤¤¤   ¤¤¤¤¤¤   /////"
       << "\n ///////////////////////////////////////////////////////////////////////////"
       << "\n///////////////////////////////////////////////////////////////////////////"
              << "\n"
       << "\n Key | Description               >>> Codebook currently containing " << liCodebook-
>NrOfElements() << " words <<< "
```

```
       << "\n=============================================================================|"
       << "\n  C  | Compare a new word with the ones currently in database.               |"
       << "\n  N  | New word? Add it to the database!                                     |"
       << "\n  V  | What's in the Codebook? Let's have a look!                            |"
       << "\n  Q  | I can't take this anymore! Let me out!                                |"
       << "\n=============================================================================|"
       << "\n"
       << "\n"
       << "\nWhat path would you like to embark on? ";
}

  //---------------------//
 // Codebook Management //
//---------------------//

void ReadCodebookFromFile()
/*****************************************************************************************************/
/* Notice: The structure of the ANSI-formatted file 'codebook.txt' can be found in appendix  .   */
/*****************************************************************************************************/
{
  ComplexMatrix *matTime, *matFreq;

  ifstream InFile("codebook.txt");              //The codebook data is in the 'codebook.txt' file.

  if(InFile)                                    //If this file exists:
  {
    int    iNrOfDataSets, iNrOfSampleSets, iNrOfSamples, i, j, k;
    double dNewSample;
    char   cTemp[STRLEN],
           *cWordName;

    InFile >> iNrOfDataSets; InFile.ignore();   //First read the NrOfDataSets in the codebook.

    for(k = 1; k <= iNrOfDataSets; k++)         //For every DataSet
    {
      InFile.getline(cTemp, STRLEN);            //there is a name of the word inherent to the DataSet,
      cWordName = new char[strlen(cTemp)+1];
      strcpy(cWordName, cTemp);

      //Read timedomain data:
      InFile >> iNrOfSampleSets;                //aswell as a number of rows and columns that define
      InFile >> iNrOfSamples;                   //the structure of the matrix storing the data.

      matTime = new ComplexMatrix(iNrOfSampleSets, iNrOfSamples);

      for(i = 1; i <= iNrOfSampleSets; i++)
        for(j = 1; j <= iNrOfSamples; j++)
        {
          InFile >> dNewSample;                 //The data itself consists of complex data. Since
          (*matTime)(i,j)->dRe = dNewSample;    //these are presented in rectangular form we need
          InFile >> dNewSample;                 //to write both real and imaginary parts to the file.
          (*matTime)(i,j)->dIm = dNewSample;    //All values are spaced with a tab.
        }

      //Read frequency domain data:
      InFile >> iNrOfSampleSets;                //Next we also need to do the same with the
      InFile >> iNrOfSamples;                   //frequency domain data. Rows/columns need to be
                                                //noted..
      if( (iNrOfSampleSets > 0) && (iNrOfSamples > 0) )
      {                                         //..a datastructure need to be made..
        matFreq = new ComplexMatrix(iNrOfSampleSets, iNrOfSamples);

        for(i = 1; i <= iNrOfSampleSets; i++)
          for(j = 1; j <= iNrOfSamples; j++)
          {
            InFile >> dNewSample;               //..and both real and imaginary information has to
            (*matFreq)(i,j)->dRe = dNewSample;  //be read.
            InFile >> dNewSample;
            (*matFreq)(i,j)->dIm = dNewSample;
          }
      }
      else
        matFreq = NULL;                         //We might also have the specialcase where frequency
                                                //domain structure has 0 rows / column, this means
      InFile.ignore();                          //there simply are no frequency information and we
                                                //set the matFreq pointer to NULL.
      liCodebook->Add(new ComplexDataSet(liCodebook->NrOfElements()+1, cWordName, matTime, matFreq));
    }                                           //Finally the newly acquired data are added to the
  }                                             //codebook.
  else                                          //If 'lyddata.txt' didn't exist we notify user.
    cout << "\n'lyddata.txt' does not exist!";

  InFile.close();
}
```

```
void WriteCodebookToFile()
/*********************************************************************************************/
/* Notice: The structure of the ANSI-formatted file 'codebook.txt' can be found in appendix  .   */
/*********************************************************************************************/
{
  ofstream OutFile("codebook.txt");             //Codebook information is to be stored into
                                                //'codebook.txt'.
  ComplexDataSet *tmpData;
  int iNrOfDataSets = liCodebook->NrOfElements();

  OutFile << iNrOfDataSets << "\n";             //First we output how many DataSets there are in
                                                //total in the codebook.
  for(int i = 1; i <= iNrOfDataSets; i++)
  {                                             //Then we simply ask them all to write themselves
    tmpData = (ComplexDataSet*)liCodebook->RemoveNr(i);//to file.
    tmpData->WriteToFile(&OutFile);
    liCodebook->Add(tmpData);
  }

  OutFile.close();
}
void ViewCodebook()
{
  ComplexDataSet *tmpData;                      //This function just outputs the textstrings of
  int iNrOfDataSets = liCodebook->NrOfElements();//what words are in the codebook to screen.
                                                //Nothing fancy, but very handy.
  cout << "\nCodebook currently contains these words:";

  for(int i = 1; i <= iNrOfDataSets; i++)
  {
    tmpData = (ComplexDataSet*)liCodebook->RemoveNr(i);
    cout << '\n' << i << ". " << tmpData->GetWordName();
    liCodebook->Add(tmpData);
  }                                             //The _getch_nolock() simply breaks the program
  cout << "\n\nPress a key to continue"; _getch_nolock();//till some key are pressed.
}
void NewWord()
{
  char cTemp[100];                              //Want a new word added to the codebook?
  char *cWordName;                              //Put it in 'lyddata.txt' and hit this function!

        cout << "\n\n\n"
       << "\n                    //=========================//"
       << "\n                   // Add new word to database //"
       << "\n                  //=========================//"
       << "\n"
       << "\n----------------------------------------------------------------------------"
       << "\n Notice: Make sure 'lyddata.txt' contains the desired word. "
       << "\n----------------------------------------------------------------------------"
       << "\n\n\n\n\n\n\n\n\n\n\n\n\n"
       << "\n The word is: ";

   cin.getline(cTemp, 100);                     //To keep track of the words we have stored we
   cWordName = new char[strlen(cTemp)+1];       //assign them all a textstring describing what
   strcpy(cWordName, cTemp);                    //word we are talking about.

   ReadWordFromFile(cWordName);                 //Then we read the new word and add it to codebook.

   cout << "\nProcessing...";
                                                //After that we process it, that is run feature-
   FeatureExtraction();                         //extraction.
}
void ReadWordFromFile(char *cWordName)
/*********************************************************************************************/
/* Notice: The structure of the ANSI-formatted file 'lyddata.txt' can be found in appendix  .    */
/*********************************************************************************************/
{
  ComplexMatrix *matTime;

  ifstream InFile("lyddata.txt");               //The core function that reads in words, and adds
                                                //them to database.
  if(InFile)                                    //If the source-file 'lyddata.txt' exists, we
  {                                             //assume proper data structure and run ahead with
    int    iNrOfSampleSets, iNrOfSamples, i, j; //the information-gathering.
    double dNewSample;

    InFile >> iNrOfSampleSets;                  //First we read the NrOfSampleSets and
    InFile >> iNrOfSamples;                     //the NrOfSamples.
                                                //Then we create the datastructure necessary to
    matTime = new ComplexMatrix(iNrOfSampleSets, iNrOfSamples); //store the data.
```

```
    for(i = 1; i <= iNrOfSampleSets; i++)       //Finally we run through the file as it was a
      for(j = 1; j <= iNrOfSamples; j++)        //a matrix (it is formatted that way),
      {
        InFile >> dNewSample;                   //writes a new value and assign this to the time-
        (*matTime)(i,j)->dRe = dNewSample;      //domain. It is simply best to think of time-domain
      }                                         //data as real. The imaginary part is automatically
                                                //set to zero by the '=' operator. Finally we add
    liCodebook->Add(new ComplexDataSet(liCodebook->NrOfElements()+1, cWordName, matTime));//the word
  }                                             //to codebook.
  else                                          //If 'lyddata.txt' did not exist of could not be
    cout << "\n'lyddata.txt' does not exist!";  //read we notify user.

  InFile.close();
}

  //------------------------//
 // Word Computing Section //
//------------------------//

void CompareWord()
/*********************************************************************************************/
/* Notice: Takes whatever word is present in the 'lyddata.txt' file and compares it with the     */
/*         words in the codebook. It uses a pretty straightforward autocorrelation algorithm to  */
/*         do this. Whether this method justifies the cause in a good way is however doubious.   */
/*********************************************************************************************/
{
  int            iNrOfDataSets = liCodebook->NrOfElements(),
                 iSize, i, j;
  double         *dCorrConstant = new double[iNrOfDataSets],
                 dLargestCorrConst;
  ComplexDataSet *zdTestData, *zdData;
  ComplexMatrix  *matTest, *matTime, *matResult;

  struct sData                                  //A temporarly structure to hold the data we need
  { char   *cWordName;                          //from a ComplexDataSet.
    double dCorrConstant;
  } *Data = new sData[iNrOfDataSets];

  ReadWordFromFile(NULL);                       //First we read the word currently present in
                                                //'lyddata.txt' but don't mind entering a name
  cout << "\nProcessing...";                    //for it.

  FeatureExtraction();                          //Then its feature extraction time! I uses the last
                                                //word added to codebook as source. Furthermore we
  zdTestData = (ComplexDataSet*)liCodebook->RemoveNr(iNrOfDataSets+1);//remove it from the codebook
  matTest    = zdTestData->GetTime();           //as it's not supposed to be there. The time-domain
                                                //data is what we are interested in. Autocorrelation
  matTest->S->Rotate180();                      //is basically the same as convolving two signals
                                                //together where one has inverted phase. In this
  //Correlate all DataSets with the testword:   //case we simply rotate the testmatrix 180 to
  for(i = 1; i <= iNrOfDataSets; i++)           //accomplish the phaseinvertion. Then, with the
  {                                             //formalities sorted we are ready for business!
    zdData = (ComplexDataSet*)liCodebook->RemoveNr(i); //One by one the ComplexDataSets are removed
    matTime = zdData->GetTime();                //from codebook and time data adress are noted.
                                                //For the presentation later on we want to read
    Data[i-1].cWordName = zdData->GetWordName();//the word name/description aswell.

    matResult = *matTime->S * *matTest;         //Finally, the convolution.
                                                //We use the "peak of correlation" as an indicator
    iSize = matResult->NrOfColumns()*matResult->NrOfRows();//of how good the words match. There are
                                                //lot of weaknesses with this particular method, but
    //Find the "peak" of match:                 //then again it will do for now.
    for(j = 1; j <= iSize; j++)                 //The peak are very easy to find, we simply run
      if(Data[i-1].dCorrConstant < (*matResult)(j)->dRe)//through all values and find the largest.
        Data[i-1].dCorrConstant = (*matResult)(j)->dRe;

    liCodebook->Add(zdData);                    //When done we put the ComplexDataSet back into the
  }                                             //codebook.
  dLargestCorrConst = 0.0;

  //Find largest correlationconstant:
  for(int i = 0; i < iNrOfDataSets; i++)        //So what word had the largest correlation factor /
    if(dLargestCorrConst < Data[i].dCorrConstant)//"peak of correlation"? Well this is pretty easy to
      dLargestCorrConst = Data[i].dCorrConstant;//find out.

  //Normalize with this:
  for(int i = 0; i < iNrOfDataSets; i++)        //Also, we want the results presented as relative
    Data[i].dCorrConstant /= dLargestCorrConst; //compared to the "greatest match".
```

```
  //Present results:
  cout << "\nResults presented as relative compared to the best match:";
  for(int i = 0; i < iNrOfDataSets; i++)
  {                                             //Then it is simple outputting to screen:
     cout << '\n' << i+1 << ". " << Data[i].cWordName;
     iSize = (int)(strlen(Data[i].cWordName)+3);//This "method" will do just fine as long as we
     if(iSize < 8)                              //have no more than 9 words in the codebook.
       cout << "\t\t\t";
     else if(iSize < 16)
       cout << "\t\t";
     else
       cout << "\t";

     cout << Data[i].dCorrConstant*100 << "% match";
  }
  delete zdTestData;
  cout << "\n\nPress a key to continue"; _getch_nolock();
}

void FeatureExtraction()
/*********************************************************************************************/
/* Notice: Runs a feature extraction on the last word added to Codebook. This is where you might */
/*         want to "tweak" the setup a bit in order to possibly achieve better performance.      */
/*********************************************************************************************/
{
  ComplexDataSet *pData = (ComplexDataSet*)liCodebook->Remove(liCodebook->NrOfElements());

  //Preemphasis:
  double dAlpha = 0.97;                         //Alpha represent the emphasis-factor.
  pData->PreEmphasis(dAlpha);

  //Windowed-sinc:
  double Fc  = 0.3;                             //Falloff frequency.
  int    M   = 254;                             //Filter resolution.
  pData->WindowedSinc(Fc, M);

  //Padding for greater frequency resolution.
  int iPads = 4;                                //How many times greater frequency resolution?
  ComplexMatrix *matTime = pData->GetTime();
  matTime->S->ResizeMatrix(matTime->NrOfRows(), iPads*matTime->NrOfColumns());

  //Fft:                                        //Finds the frequency information from the time-
  pData->FFTAnalysis();                         //domain data.

  //Mel Frequency Ceptral Coefficients.         //Transforms the frequency spectrum from N pts.
  int     iNrOfMelCoeff  = 24;                  //to NrOfMelCoeff(icients) pts. The function will
  double  dSampleFreq    = 24000,               //also need to know the samplefrequency and "base-
          dBaseBandwidth = 100;                 //bandwidth" - the bancdwith between pts. in the
  pData->MFCC(24, 24000, 100);                  //linear field of the Mel-frequency spectra.

  //Discrete Cosinus Transform.                 //Uses a cosinus function as basis for weighting
  pData->DCT();                                 //the frequency domain signal into the time-domain.

  liCodebook->Add(pData);                       //When done the data are readded into the codebook.
}
```

A.2 Filen DataSet..h

```
#include "Lists.h"      //A tool to make dynamic data structures.
#include "Toolbox.h"    //A class with more artithmetic-related tools.


  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Global Class Declarations                 ///
////////////////////////////////////////////////

                                               /*************************************************/
       ////////////////////////////            /* This is the container class that holds arrays */
      /// class ComplexDataSet ///             /* with sampled data and the corresponding       */
     ////////////////////////////              /* frequency spectras.                           */
                                               /*************************************************/
class ComplexDataSet: public NumElement
{
private:
  char *cWordName;                             //The name of the word the object holds.
  ComplexMatrix *matTime, *matFreq;            //A ComplexMatrix holding time- and frequency
                                               //domain data.

public:                                        //The ComplexDataSet may be contructed in two ways.
  ComplexDataSet(int iDataSetNr, char *cWordNameIn, ComplexMatrix *matTimeData);//Either by passing
                                               //only time-domain data, or by passing both time-
  ComplexDataSet(int iDataSetNr, char *cWordNameIn, ComplexMatrix *matTimeData, //and frequency
                                                    ComplexMatrix *matFreqData);//domain data to it.
  ~ComplexDataSet();                           //Cleanup when object is killed.
  char* GetWordName();                         //Returns the name of the word stored in the object.
  ComplexMatrix* GetTime();                    //Returns the matrix holding time-domain data.
  ComplexMatrix* GetFreq();                    //Returns the matrix holding frequency-domain data.

  void WriteFreqToFile();                      //Writes frequency-domain data to 'freqresponse.txt'
  void WriteTimeToFile();                      //Writes timedomain data to 'timeresponse.txt'.
  void WriteToFile(ofstream *OutFile);         //Writes both time-/frequency domain data to a file
                                               //the user chose.
  //Functions that call tools in the toolbox:
  void PreEmphasis(double dAlpha);             //Preemphasis of the frequencydomain.
  void WindowedSinc(double Fc, int M);         //Lowpass filter for the frequencydomain.
  void FFTAnalysis();                          //The path from time- of frequency domain.
  void MFCC(int KmelTotal, double dSampleFreq, double dBaseBandwidth); //Find the MEL Frequency Cepstral
Coefficients.
  void DCT();
};


  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Class Function Definitions                ///
////////////////////////////////////////////////

  //---------------------------//               /*************************************************/
 // Constructors / destructors //               /* Creation and destruction of the ComplexDataSet*/
//---------------------------//                 /*************************************************/

ComplexDataSet::ComplexDataSet                 //Constructs the ComplexDataSet with time-domain
  (int iDataSetNr, char *cWordNameIn, ComplexMatrix *matTimeData): //data, index and a textstring
  NumElement(iDataSetNr)                       //to show what word we are talking about.
{ cWordName = cWordNameIn;                     //The frequency domain are left empty. This is
  matTime   = matTimeData;                     //desireable when the size of the frequency domain
  matFreq   = NULL;                            //is not yet known.
}
ComplexDataSet::ComplexDataSet                 //Constructs the ComplexDataSet with data from both
  (int iDataSetNr, char *cWordNameIn, ComplexMatrix *matTimeData, ComplexMatrix *matFreqData):
  NumElement(iDataSetNr)                       //time- and frequency domain data. It will also need
{ cWordName = cWordNameIn;                     //to know its index in the list holding it, aswell
  matTime   = matTimeData;                     //as a textstring to show what word we are talking
  matFreq   = matFreqData;                     //about.
}
ComplexDataSet::~ComplexDataSet()              //We dynamically allocate memory for the textstring
{ delete cWordName, matTime, matFreq;          //holding word name, and the matrix holding time-
}                                              //and frequency-domain data. This needs to be
                                               //deallocated in order to avoid memoryleaks.
```

```
  //--------------------//                    /*********************************************/
 // Get...() Functions //                     /* Need the time-/frequency data or wordname?    */
//--------------------//                      /*********************************************/

char* ComplexDataSet::GetWordName()           //Simply returns the word name. Will often come in
{ return cWordName;                           //handy.
}
ComplexMatrix* ComplexDataSet::GetTime()      //Want the time-domain data? Call this beauty!
{ return matTime;
}
ComplexMatrix* ComplexDataSet::GetFreq()      //Frequency-domain data? Yes please!
{ return matFreq;
}

  //-------------------------//                /*********************************************/
 // Stream Access Functions //                 /* For output to screen or file.                 */
//-------------------------//                  /*********************************************/

void ComplexDataSet::WriteFreqToFile()        //Writes the frequency domain data to the file
{                                             //'freqresponse'.
  ofstream OutFile("freqresponse.txt");

  for(int i = 1; i <= matFreq->NrOfRows(); i++) //It's simple enough. Just output anything in the
    for(int j = 1; j <= matFreq->NrOfColumns(); j++) //matFreq to the file, equally spaced with a
      OutFile << "\t" << (*matFreq)(i,j)->Mod();//tab. If you use the function in ComplexMatrix
                                              //to do the same you'll end up with the same data
  OutFile.close();                            //only on "matrix form", that is with rows/columns.
}
void ComplexDataSet::WriteTimeToFile()        //Writes the time domain data to the file
{                                             //'timeresponse.txt'
  ofstream OutFile("timeresponse.txt");

  for(int i = 1; i <= matTime->NrOfRows(); i++) //It's simple enough. Just output anything in the
    for(int j = 1; j <= matTime->NrOfColumns(); j++)//matTime to the file, equally spaced with a
      OutFile << "\t" << (*matTime)(i,j)->Re(); //tab. If you use the function in ComplexMatrix
                                              //to do the same you'll end up with the same data
  OutFile.close();                            //only on "matrix form", that is with rows/columns.
}
void ComplexDataSet::WriteToFile(ofstream *OutFile)//Outputs the whole ComplexDataSet to a file
{                                             //of users choice. For structure see appendix .
  int iNrOfRows    = matTime->NrOfRows(),     //First we will output the timedomain. We want a
      iNrOfColumns = matTime->NrOfColumns();  //nice structure on the file, therefore we will
                                              //to know the number of rows/columns of the matrix.
  *OutFile << cWordName << '\n'               //First of we output the word name, then on a new
           << iNrOfRows << '\t' << iNrOfColumns << '\n'; //line the number of rows / columns.

  for(int i = 1; i <= iNrOfRows; i++)         //For every row there is the same set of columns.
  {                                           //We run through each of these and output the
    for(int j = 1; j < iNrOfColumns; j++)     //values. Since these are complex and they are
      *OutFile << (*matTime)(i,j)->dRe << '\t' << (*matTime)(i,j)->dIm << '\t'; //represented on
                                              //rectangular form we have a real and an imaginary
    *OutFile << (*matTime)(i,iNrOfColumns)->dRe << '\t' //part. The end of every row has a
             << (*matTime)(i,iNrOfColumns)->dIm << '\n';//lineshift.
  }

  if(matFreq)                                 //The frequency domain might or might not be
  {                                           //present. If it is we will need to know the size
    iNrOfRows    = matFreq->NrOfRows(),       //of the matrix holding it as before for proper
    iNrOfColumns = matFreq->NrOfColumns();    //structure on the file.

    *OutFile << iNrOfRows << '\t' << iNrOfColumns << '\n'; //We start with outputting the matrix
                                              //size.
    for(int i = 1; i <= iNrOfRows; i++)       //For every row there is the same set of columns.
    {                                         //We run through each of these and output the
      for(int j = 1; j < iNrOfColumns; j++)   //values. Since these are complex and they are
        *OutFile << (*matFreq)(i,j)->dRe << '\t' << (*matFreq)(i,j)->dIm << '\t'; //represented on
                                              //rectangular form we have a real and an imaginary
      *OutFile << (*matFreq)(i,iNrOfColumns)->dRe << '\t' //part. The end of every row has a
               << (*matFreq)(i,iNrOfColumns)->dIm << '\n';//lineshift.
    }
  }
  else                                        //If the frequency domain was not present we simply
    *OutFile << 0 << '\t' << 0 << '\n';       //output 0 rows and 0 columns.
}
```

```
  //---------------------------------//         /*********************************************/
 // Functions that call the Toolbox //          /* When you need arithmetics done on the data.   */
//---------------------------------//           /*********************************************/

void ComplexDataSet::PreEmphasis(double dAlpha)                         //Frequency boost?
{ matTime = Tools.PreEmphasis(matTime, dAlpha);
}
void ComplexDataSet::WindowedSinc(double Fc, int M)                     //Runs a windowed-sinc
{ matTime = Tools.WindowedSinc(matTime, Fc, M);                         //filter on timedomain data.
}
void ComplexDataSet::FFTAnalysis()                                      //Whenever you want to enter
{ matFreq = Tools.FFTAnalysis(matTime);                                 //the frequency domain.
}
void ComplexDataSet::MFCC(int KmelTotal, double dSampleFreq, double dBaseBandwidth) //MEL Frequency
{ matFreq = Tools.MFCC(matFreq, KmelTotal, dSampleFreq, dBaseBandwidth);//Cepstral Coefficients
}
void ComplexDataSet::DCT()                                              //Discrete Cosinus
{ matTime = Tools.DCT(matFreq);                                         //Transform.
}
```

A.3 Filen Toolbox.h

```
   ////////////////////////////////////////////////////////////////////////////////////////////
 /// Include                                                ///
//////////////////////////////////////////////////

#include "Matrix.h"   //A tool to deal with sets of data.

  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Constants                                              ///
//////////////////////////////////////////////////


const double  PI      = 3.14159265358979323846;
const int     STRLEN  = 100;


  ////////////////////////////////////////////////////////////////////////////////////////////
 /// Global Class Declarations                              ///
//////////////////////////////////////////////////


                                                /***********************************************/
       ////////////////////                     /* This is a container class which holds a few   */
      /// class Toolbox ///                     /* tools for feature extraction for Automated    */
     ////////////////////                       /* Speech Recognition (ASR).                     */
                                                /***********************************************/

class Toolbox
{
private:
//You may wish to put precomputed stuff here.

public:
  Toolbox();                                    //Constructor.
  ~Toolbox();                                   //Destructor.
  ComplexMatrix* PreEmphasis(ComplexMatrix *matTime, double dAlpha);    //Frequency boosting.
  ComplexMatrix* WindowedSinc(ComplexMatrix *matTime, double Fc, int M);//Versatile filter.
  ComplexMatrix* FFTAnalysis(ComplexMatrix *matTime);                   //Time -> frequency domain.
  ComplexMatrix* MFCC(ComplexMatrix *matFreq, int KmelTotal,            //Mel Frequency Cepstral
                      double dSampleFreq, double dBaseBandwidth);       //Coefficient calculation.
  ComplexMatrix* DCT(ComplexMatrix *matFreq);                           //Discrete Cosinus Transform
} Tools;

                                                /***********************************************/
       /////////////////                        /* This is the container class which holds all   */
      /// class Fft ///                         /* kinds of data the fft will need, plus the     */
     /////////////////                          /* required functions to make the domain-switch. */
                                                /***********************************************/
class Fft
{
private:
  int iN,                                       //Number of samples.
      iLOGN,                                    //How many powers of two?
      iLevel,                                   //iLevel and iStep is variables holding powers of 2,
      iStep,                                    //representing different stages in a tree-structure.
      i, j, k,                                  //Some indexing variables.
      iKplusStep,                               //Frequently used when butterflying,
      i2xStep,                                  //so is this.
      *iBitRev;                                 //LUT for bit reversed pattern.

  Complex zTmp, zT,                             //Some temporarily variables.
          **zTwiddle;                           //A twodimensional dynamic array with all the
                                                //exponentials in it.
public:
  Fft();                                        //Constructor.
  ~Fft();                                       //Destructor.
  void SetFftData(int iNrOfSamples);            //Precalculates data depending on size of sampleset.
  void FftCalc(Complex *pzTime, Complex *pzFreq);//Calculates the frequency response for the given
  int NrOfSamples()                             //ComplexDataSet.
  { return iN; }
} FftData;
```

```
  ////////////////////////////////////////////////////////////////////////////////////////////////
 /// Class Function Definitions                  ///
////////////////////////////////////////////////////


  //////////////////////
 /// class Toolbox ///
//////////////////////

    //---------------------------//             /*********************************************/
   // Contructors / destructors //              /* Same old functions for creation/destruction.  */
  //---------------------------//               /*********************************************/

Toolbox::Toolbox()                              //Atm. these are both empty.
{}
Toolbox::~Toolbox()
{}

    //---------------------------//             /*********************************************/
   // The Preemphasis Function //               /* Whenever you feel like boosting frequencies.  */
  //---------------------------//               /*********************************************/

ComplexMatrix* Toolbox::PreEmphasis(ComplexMatrix *matTime, double dAlpha)
{                                               //The pre-emphasis work the following way. First we
  ComplexMatrix *matTimeShifted = *matTime->C >> 1; //need a delayed version of the timedomain,
                                                //doing this simply by shifting the matrix one step.
  *matTimeShifted *= dAlpha;                    //The shifted matrix are then weighted by a factor
                                                //assigned the symbol alpha (usually 0.95 - 0.99).
  ComplexMatrix *matResult = *matTime - *matTimeShifted; //Then we substract the delayed (and weighted)
                                                //version of the time-domain data and the process
  delete matTimeShifted;                        //is finished!
  delete matTime;                               //At the end we do a cleanup.
  return matResult;                             //And return the result.
}

    //----------------------------//            /*********************************************/
   // The Windowed-Sinc Function //              /* One of the best filters available.            */
  //----------------------------//               /*********************************************/

ComplexMatrix* Toolbox::WindowedSinc(ComplexMatrix *matTime, double Fc, int M)
/**********************************************************************************************/
/* Notice: Frequencies passed on to this function need to be in accordance with SI unit [Hz].    */
/*         You MUST catch the returned matrix, in order to delete it when no longer in use.      */
/**********************************************************************************************/
{
  double K   = 0.0;                             //K:  A normalization constant, calculated later.
  int    Md2 = M/2,                             //Fc: Cut-off frequency.
         i, j;                                  //M:  Windowed-sinc resolution.
  ComplexMatrix *matWSKernel = new ComplexMatrix(1, M+1), //We create ourselves a matrix holding
                *matHamming,                    //the values for the Window-sinc kernel, a
                *matResult;                     //correctingHamming window and finally something
                                                //to store the result in.
  //Calculate the windows kernel.
  for(i = 0; i <= M; i++)                       //There are a total of M+1 points to run through.
  { if (i == Md2)                               //For i=M/2 there will be a divide-by-zero error
      *(*matWSKernel)(1,i+1) = 2*PI*Fc;         //unless we deal with this special case with other
    else                                        //means. For the rest of the points we simply use
      *(*matWSKernel)(1,i+1) = ( sin(2*PI*Fc*(i-Md2)) / (i-Md2));//the standard windowed-sinc
  }                                             //kernel.

  //Calculate the normalization constant:
  for(i = 0; i <= M; i++)                       //The Window-sinc need to be powerconservative,
  { K += (*matWSKernel)(1,i+1)->dRe;            //this we fix with simply summing up every
  }                                             //value in the windowed-sinc matrix,

  for(i = 0; i <= M; i++)                       //and set every value as relative to this
  { *(*matWSKernel)(1,i+1) /= K;                //afterwards.
  }

  //Hamming Window:
  matHamming = new ComplexMatrix(1, M+1);       //Since we can't operate with negative indexes, we
                                                //don't obtain the perfect symmetry the windowed-
  for(j = 0; j <= M; j++)                       //sinc kernel needs in other to perform perfectly.
    *(*matHamming)(1,j+1) = 0.54 - 0.46*cos( (2*PI*j)/M );//We therefore calculate a correcting
   matHamming->S->WriteToFile();                                        //Hamming window. This is
just one of many possible
   //Kernel+Window Fusion:                      //windows to implement, though it's one of the best
  *matHamming->E *= *matWSKernel;               //when dealing with voice recognition algorithms.
  delete matWSKernel;                           //The Hamming window and windows-sinc kernel are
                                                //first simply multiplied with each other.
  //Filter and timedomain convolution:          //This corresponds to a convolution in the time-
  matResult = *matTime->S * *matHamming;        //domain and so this will need to be done aswell.

  delete matTime;                               //Finally, cleanup and return of the filtered
  delete matHamming;                            //data.
  return matResult;
}
```

```
    //---------------------------//             /*********************************************/
   // The FFT Analysis Function //              /* The path from one domain to another.      */
  //---------------------------//               /*********************************************/

ComplexMatrix* Toolbox::FFTAnalysis(ComplexMatrix *matTime)
{
  bool bPowerOfTwo = false;                    //Before we even think about running the Fft we need
  int  iC          = matTime->NrOfColumns();   //to make sure the recieved dataset has a power-of-
                                               //two numer of columns.
  ComplexMatrix *matFreq;                      //Also, since this is the path from the time- to
                                               //frequency-domain we need to "create" it...
  while( bPowerOfTwo == false && (iC % 2) == 0.0 )//Power of two? If a number is a power of two
  {                                            //it can be divided with any power of 2 lesser than
    iC >>= 1;                                  //its own, and there will not be a residual.
    if(iC == 1)                                //When we have made sure time domain is of valid
    {                                          //size, we create the frequency domain and are
      matFreq = new ComplexMatrix(matTime->NrOfRows(), matTime->NrOfColumns()); //ready to crank
                                               //up the business. First of all, if samplenumber has
      if(FftData.NrOfSamples() != matTime->NrOfColumns())//changed since last time we ran the Fft
        FftData.SetFftData(matTime->NrOfColumns());//we need to calculate new twiddlefactors etc.
                                               //We call the Fft simply by passing both time- and
      for(int i = 1; i <= matTime->NrOfRows(); i++)//frequency domain data to it, and let nature
        FftData.FftCalc( (*matTime)(i,1), (*matFreq)(i,1) );//take care of the rest.
                                               //Finally, when the whole process is done we
      bPowerOfTwo = true;                      //definitely had ourselves a power-of-two case.
    }
  }
  if(!bPowerOfTwo)                             //Not power-of-two? Well then, let user know!
    cout << "\n(ComplexDataSet 'FFTAnalysis()') TimeDomain matrix is not power of two!";

  return matFreq;                              //The fresh and steaming frequency spectra returned.
}

    //--------------------//                   /*********************************************/
   // The MFCC Function //                      /* Calculates a Mel-Frequency Cepstral Spectra.  */
  //--------------------//                     /*********************************************/

ComplexMatrix *Toolbox::MFCC
(ComplexMatrix *matFreq, int KmelTotal, double dSampleFreq, double dBaseBandwidth)
/*****************************************************************************************/
/* Notice: KmelTotal is the number of Mel-Frequency Cepstral Coefficients we want calculated.    */
/*         Frequencies passed on to this function need to be in accordance with SI unit [Hz].    */
/*****************************************************************************************/
{
  int     iNrOfSamples = matFreq->NrOfColumns(),          //We note the number of samples, and how
          KmelStep     = (int)(dBaseBandwidth*iNrOfSamples/dSampleFreq),//many samples between each
          Kmel, j, i;                                     //Mel centerfrequency.

  double  Kc     = 700.0*iNrOfSamples/dSampleFreq,         //The frequency at which the Mel-sprectra
          Kp     = 2595.0*iNrOfSamples/dSampleFreq,        //becomes logarithmic, and a normalization
          BWinit = iNrOfSamples*dBaseBandwidth/dSampleFreq,//constant. Also, we will need to know the
          K, Klast, Knext, BW, BWlow, BWhigh, h,           //initial bandwidth between the Mel-
          *ConvertedFreq = new double[KmelTotal+2];        //coefficients, and create an array to
                                                           //hold the transformed frequency spectra.
  Complex zTemp;
                                                           //We make a transformation matrix that
  ComplexMatrix *matMel = new ComplexMatrix(iNrOfSamples, KmelTotal);//transforms the frequency
                                                           //domain into a domain of Mel-
                                                           //coefficients.
  //Is it even possible to calculate all the Mel-coefficients the user want to?
  if( (Kc*(pow(10.0, (double)(KmelTotal/Kp)) - 1)) <= (iNrOfSamples/2) )
  {
    matMel->Clear();

    //Calculate |E|^2, where E is a element in the matrix.
    matFreq->S->MakeAbsolute();                  //First of all we replace the complex numbers with
    *matFreq->E ^= 2;                            //absolute numbers. Then we raise then by 2.

                                                 //This transforms the frequency-axes according to:
    //Mel-frequency to normal frequency:         //K = Kc( 10^(Kmel/Kp) - 1 ). Based on the base-
    for(Kmel = KmelStep, i = 0; i <= KmelTotal+1; Kmel += KmelStep, i++)//bandwidth we previously
    { ConvertedFreq[i] = Kc*pow(10.0, (double)(Kmel/Kp));//calculated the proper step for the Mel
      ConvertedFreq[i] -= Kc;                    //"frequencies", which is linear. Only the normal
    }                                            // frequency domain contains unlinearities.
    //Calculate Mel-sprectrum:
    for(i = 1; i <= KmelTotal; i++)              //There are a total of KmelTotal Mel-Coefficients.
    { K     = ConvertedFreq[i];                  //We calculate the Mel-Frequency spectra with Kmel
      Klast = ConvertedFreq[i-1];                //a the center frequency in the Mel-Frequency band,
      Knext = ConvertedFreq[i+1];                //and Kmel-1 as lower limit and Kmel+1 as upper.

      BWlow = K-Klast;                           //Since linearity in Mel-Frequency spectra means
      BWhigh = Knext-K;                          //unlinearity in the normal frequency spectra we
      BW = Knext - Klast;                        //calculate the bandwidth of the lower and upper
                                                 //half of the Mel-band.
      h = BWinit/BW;                             //The "height" of the Mel-Centerfrequencycomponent
                                                 //is inversely proportional with the bandwidth.
```

```
      //Calculate lower half band:
      for(j = (int)Klast+1; j <= (int)K; j++)     //For every frequency component in between the
      {  zTemp = h * (j - Klast) / BWlow;         //lower and center Mel-frequency we calculate
        *(*matMel)(j, i) = zTemp;                 //the linear curve from 0 til h.
      }

      //Calculate upper half band:
      for(j = (int)K+1; j <= (int)Knext; j++)     //For every frequency component in between the
      { zTemp =  h * (Knext - j) / BWhigh;        //center and uppper Mel-frequency we calculate
        *(*matMel)(j, i) = zTemp;                 //the linear curve from h to 0.
      }
    }
    *matFreq *= *matMel;                          //Finally the Mel transformation matrix are
  }                                               //multiplied with the frequency domain.
  else
    cout << "\n(Toolbox - MFCC(...) ) Not possible to calculate that many Mel-frequency"
         << "\n                       coefficients. Reduce the number, the 'basebandwith'"
         << "\n                       or increase the samplefrequency.";

  delete ConvertedFreq;

  return matFreq;
}

    //----------------------------//            /***********************************************/
   // Discrete Cosinus Transform //             /* Weight freq. with cosinus, convolve and woila!*/
  //----------------------------//              /***********************************************/

ComplexMatrix* Toolbox::DCT(ComplexMatrix *matFreq)
{
  int M  = matFreq->NrOfColumns(),              //The samplenumber is as usual the same as the
      Ki = matFreq->NrOfRows(),                 //number of columns in the frequencymatrix.
      i, j;

  ComplexMatrix *matDCT = new ComplexMatrix(M, Ki); //The DCT are calculated and put into this
                                                //matrix.
  for(i = 0; i < Ki; i++)                       //The transform itself is very simple. One just
    for(j = 0; j < M; j++)                      //weights every frequency spectra with a cosinus-
      *(*matDCT)(j+1,i+1) = cos(PI*i*(j+0.5)/M);//function, and since weighting in the frequency
                                                //domain corresponds to convolution in the time-
  return (*matFreq * *matDCT);                  //domain this is what we return.
}


  /////////////////                             /***********************************************/
 /// class Fft ///                             /* Frequency-domain here we come!              */
/////////////////                               /***********************************************/

    //----------------------------//            /***********************************************/
   // Contructors / destructors //              /* Same old functions for creation/destruction.  */
  //----------------------------//              /***********************************************/

Fft::Fft()
{}
Fft::~Fft()
{
  for(i = 0; i < iLOGN; i++)                    //It is always good practice to do a cleanup when
  { delete [] zTwiddle[i];                      //session is terminated. First we kill the zTwiddle
  }                                             //columns then the rows. iBitRev has dynamically
  delete [] zTwiddle;                           //allocated memory space aswell, thus it is next on
  delete [] iBitRev;                            //our hitlist.
}

    //-------------------------//                /***********************************************/
   // The SetFftData Function //                /* Precalc's only dependent of the samplenumber. */
  //-------------------------//                 /***********************************************/

void Fft::SetFftData(int iNrOfSamples)
/*********************************************************************************************/
/*Holds every data the fft-routine will use that can be precalculated when samplenumber is known.*/
/*Uses column size of the matrix recieved as samplenumber. What mainly is calculated is a LUT of */
/*indexes that are bitreversed and a twiddlefactor-array to use when butterflying.             */
/*********************************************************************************************/
{
  iN = iNrOfSamples;                            //How many samples are we working with?
  iLOGN = (int)(log((float)iN)/log(2.0));       //Calc's the number of "stages" in the decomposition.
  iStep = iN>>1;                                //The MSB is worth N/2.
  iLevel = 1;                                   //The LSB is allways worth 1.
  iBitRev = new int[iN];

  for(i = 0; i < iN; i++)                       //Cleans out the array we will use as LUT for the
    iBitRev[i] = 0;                             //bit reversed pattern.
```

```
  //Calculate indexes for bitreversed pattern:
  for(i = 0; i < iLOGN; i++)                    //Runs through every bit that is to be reversed.
  { for(j = iN-1; j >= 0; j -= 2*iLevel )       //Then we run through the N numbers available and
      for(k = 0; k < iLevel; k++)               //add the opposite value of every bit to the
        iBitRev[j-k] += iStep;                  //corresponding place in the array.

    iLevel <<= 1;                               //With every new bit the pattern change, this must
    iStep >>= 1;                                //be accounted for.
  }

  //Precompute complex exponentials:
  iStep = 1;                                    //The factor that scale the angle.
  zTwiddle = new Complex*[iLOGN];               //Make a twodimensional dynamic array with all the
                                                //exponentials in it.
  for(i = 0; i < iLOGN; i++)                    //There are iLOGN "stages" of exponentials,
  {
    zTwiddle[i] = new Complex[iN];              //(allocate memory for the exponentials needed)
    for(int j = 0; j < iN; j++)                 //and iN exponentials in all of them.
    {
      zTwiddle[i][j].dRe = cos(PI*j / iStep);   //For the first stage the angle is PI*J / 1, for the
      zTwiddle[i][j].dIm = -sin(PI*j / iStep);  //second it's PI*j / 2, then PI*J / 4 asf.
    }                                           //The exponentials is often referred to as twiddle-
  iStep <<= 1;                                  //factors.
  }
}

    //----------------------//                  /*********************************************/
   // The FftCalc Function //                   /* Runs a 2 radix FFT, decimation-in-time.   */
  //----------------------//                    /*********************************************/

void Fft::FftCalc(Complex *pzTime, Complex *pzFreq)
/*****************************************************************************************/
/*Runs a 2 radix FFT. Bitreversed indexes is found in iBitRev[] and twiddlefactors in     */
/*zTwiddle[][]. Works directly on the stored data pzTime[][], first copying the timedomain data  */
/*in bitreversed order to the frequency-domain pzFreq[][], and then the butterfly calculations   */
/*are carried out. Demands that SetFftData() are called whenever the samplenumber changes.       */
/*****************************************************************************************/
{
  //Load data into frequency domain:
  for(i = 0; i < iN; i++)                       //Loads data from timedomain into the frequency-
    pzFreq[i] = pzTime[iBitRev[i]];             //domain in a bitreversed order. The frequency-
                                                //domain is now ready for butterflying.
  //From time to frequency:
  iStep = 1;                                    //Start off with a step of 1, this will double
                                                //every stage of the transformation.
  for(i = 0; i < iLOGN; i++)                    //There are log2(N) such stages.
  {
    i2xStep = iStep<<1;                         //Since the value iStep/2 will be frequently
                                                //used we allocate an own variable for it.
    for (j = 0; j < iStep; j++)                 //Step indicates how many set of SubDft's
    {                                           //is to be carried out.
      zTmp = zTwiddle[i][j];                    //We pick out the appropriate twiddlefactor.

      for(k = j; k < iN; k += i2xStep)          //For every SubDft there is a given set of
      {                                         //butterflies to be carried out.
        iKplusStep = k + iStep;                 //Each butterfly operate on data with index
                                                //'k' and 'k+iStep'.
        zT = zTmp;                              //The twiddlefactor is the same for all butterflies
        zT *= pzFreq[iKplusStep];               //in a SubDft. The next lines are the actual
        pzFreq[iKplusStep] = pzFreq[k];         //butterfly calculation. Odd indexed-component are
        pzFreq[iKplusStep] -= zT;               //multiplied with twiddle then added to evenindexed.
        pzFreq[k] += zT;                        //The the sign on the odd-indexed component is
      }                                         //inverted and the even indexed component is added
    }                                           //to form the new odd-indexed component.
    iStep <<= 1;                                //The procedure follows a pattern with powers of 2.
  }
  //Normalize frequency domain:
  for(i = 0; i < iN; i++)                       //The frequency spectra will need to be normalized
  {                                             //in order to be prepared for direct transform back
    pzFreq[i].dRe /= iN;                        //to the time-domain later. We simply divide with
    pzFreq[i].dIm /= iN;                        //the samplenumber to bring the values down to a
  }                                             //level in accordance with the complex theory.
}
```

A.4 Filen Matrix

```
#include "Complex.h"
#include <iostream>
#include <iomanip>   //Setprecision

using namespace std;

                                          /***********************************************/
  ////////////////////                    /* Basic class that defines the basic          */
 /// class Matrix ///                     /* properties of a matrix, and contains operator */
////////////////////                      /* overloads. Types are defined in Enum Types.   */
                                          /***********************************************/
class Matrix
{
protected:
  int iNrOfRows, iNrOfColumns;            //Number of rows and columns in the matrix.

public:
  Matrix(int iRows, int iColumns)         //All matrices has fundamental info about number
  { iNrOfRows = iRows;                    //of row and columns in common. This baseclass
    iNrOfColumns = iColumns;              //holds this info.
  }
  int NrOfRows()
  { return iNrOfRows;                     //Returns number of rows.
  }
  int NrOfColumns()
  { return iNrOfColumns;                  //Returns number of columns.
  }
  ~Matrix()
  { }
};

                                          /***********************************************/
  ////////////////////////////            /* Basic class that defines the basic          */
 /// class ComplexMatrix ///              /* properties of a matrix, and contains operator */
////////////////////////////              /* overloads. Types are defined in Enum Types.   */
                                          /***********************************************/
class ComplexMatrix: public Matrix
{
private:
  Complex **zTable;

public:
  ComplexMatrix(int iRows, int iColumns): Matrix(iRows, iColumns) //Construct Matrix.
  { E = new ElementOperations(this);      //Create containers for operations
    R = new RowOperations(this);          //related to elements, rows, columns,
    C = new ColumnOperations(this);       //and special manipulations like the convolve-
    S = new SpecialOperations(this);      //alogorithm.
    zTable = new Complex*[iNrOfRows];     //Size are inherited
    for(int i = 0; i < iNrOfRows; i++)    //from Matrix and so are sent to Matrix().
      zTable[i] = new Complex[iNrOfColumns];  //Memory are allocated to the table.
  }
  ~ComplexMatrix()                        //Destructor that cleans up memory space.
  { for(int i = 0; i < iNrOfRows; i++)    //Table aswell as all the containers has to go.
      delete [] zTable[i];
    delete [] zTable;
    delete E, R, C, S;
  }
                                          /***********************************************/
    //==========================//        /* This is a collection of functions that      */
   // struct ElementOperations //         /* operates on each specific element, similar  */
  //==========================//          /* to the "dot"-operator in MATLAB.            */
                                          /***********************************************/
  struct ElementOperations
  { ComplexMatrix *pBase;                 //Holds the adress to base class scope.

    ElementOperations(ComplexMatrix *BasePtr)  //To gain the scope of the base class we store the
    { pBase = BasePtr; }                  //baseclass adress in the pBase pointer.
```

```
  //-------------------------------------// /*********************************************/
 // The Element Multiplication Operator //  /* Multiplies elementwise.                     */
//-------------------------------------//   /*********************************************/

ComplexMatrix* operator *  (ComplexMatrix &matIn)
//*****************************************************************************************//
// Notice: This type of multiplication require that the matrices are of same size (and type).//
//         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*****************************************************************************************//
{
  //Check if matrix sizes match:
  if( (pBase->iNrOfRows == matIn.NrOfRows()) && (pBase->iNrOfColumns == matIn.NrOfColumns()) )
  {
    //Create new matrix to store result in:
    ComplexMatrix *matResult = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);
    matResult->Clear();

    //Elementwise multiplication:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)
        matResult->zTable[i][j] = *pBase->zTable[i,j] * matIn.zTable[i][j];

    //Result return:
    return matResult;
  }
  else
    cout << "\n(ComplexMatrix.E  *= ) Dimension mismatch.";
    return NULL;
}
bool           operator *= (ComplexMatrix &matIn)
//*****************************************************************************************//
// Notice: This type of multiplication require that the matrices are of same size (and type).//
//         Catch the returned boolean to determine whether the operation was successful.     //
//*****************************************************************************************//
{
  //Check if matrix sizes match:
  if( (pBase->iNrOfRows == matIn.NrOfRows()) && (pBase->iNrOfColumns == matIn.NrOfColumns()) )
  {
    //Elementwise multiplication:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)
        pBase->zTable[i][j] *= *matIn(i+1,j+1);

    //Success?
    return true;
  }
  else
  { return false;
    cout << "\n(ComplexMatrix.E  *= ) Dimension mismatch.";
  }
}

  //----------------------------//          /*********************************************/
 // The Element Raise Operator //           /* Raises each element into a power of itself.   */
//----------------------------//            /*********************************************/

ComplexMatrix* operator ^   (int iPower)
//*****************************************************************************************//
// Notice: You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*****************************************************************************************//
{
  ComplexMatrix *matResult = new ComplexMatrix(pBase->NrOfRows(), pBase->NrOfColumns());
  Complex zOne; zOne = 1.0;
  int iCase;

  if(iPower >  0) iCase = 1;                              //We use a integer to choose
  if(iPower == 0) iCase = 0;                              //algorithms accordin to whether the
  if(iPower <  0) iCase = -1;                             //power is positive, negative or 0.

  switch (iCase)
  {
  //Positive power:
  case (1):
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)
      { matResult->zTable[i][j] = pBase->zTable[i][j];    //Here we simply notice the base
        for(int k = 1; k < iPower; k++)                   //value and multiply with it the
          matResult->zTable[i][j] *= pBase->zTable[i][j]; //given amount of times.
      }
    break;

  //Power is zero:
  case (0):
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)        //Anything rasied 0 times is per
        matResult->zTable[i][j] = 1.0;                    //defintion 1.
```

```
      break;

    //Power is negative:
    case (-1):
      for(int i = 0; i < pBase->iNrOfRows; i++)
        for(int j = 0; j < pBase->iNrOfColumns; j++)
        { matResult->zTable[i][j] = pBase->zTable[i][j];    //Only difference between the
          for(int k = 1; k < iPower; k++)                   //operation with negative powers and
            matResult->zTable[i][j] *= pBase->zTable[i][j]; //positive is that one once need to
          matResult->zTable[i][j] = zOne/matResult->zTable[i][j]; //invert the base.
        }
      break;
    }
    return matResult;
  }
  void          operator ^=  (int iPower)
  //*********************************************************************************************//
  // Notice: Operates on the matrix directly, thus overwriting its old data.                     //
  //*********************************************************************************************//
  {
    Complex zBase,
            zOne; zOne = 1.0;
    int iCase;

    if(iPower >  0) iCase = 1;                              //We use a integer to choose
    if(iPower == 0) iCase = 0;                              //algorithms accordin to whether the
    if(iPower <  0) iCase = -1;                             //power is positive, negative or 0.

    switch (iCase)
    {
    //Positive power:
    case (1):
      for(int i = 0; i < pBase->iNrOfRows; i++)
        for(int j = 0; j < pBase->iNrOfColumns; j++)
        { zBase = pBase->zTable[i][j];                      //Here we simply notice the base
          for(int k = 1; k < iPower; k++)                   //value and multiply with it the
            pBase->zTable[i][j] *= zBase;                   //given amount of times.
        }
      break;

    //Power is zero:
    case (0):
      for(int i = 0; i < pBase->iNrOfRows; i++)
        for(int j = 0; j < pBase->iNrOfColumns; j++)        //Anything rasied 0 times is per
          pBase->zTable[i][j] = 1.0;                        //defintion 1.
      break;

    //Power is negative:
    case (-1):
      for(int i = 0; i < pBase->iNrOfRows; i++)
        for(int j = 0; j < pBase->iNrOfColumns; j++)
        { zBase = pBase->zTable[i][j];                      //Only difference between the
          for(int k = 1; k < iPower; k++)                   //operation with negative powers and
            pBase->zTable[i][j] *= zBase;                   //positive is that one once need to
          pBase->zTable[i][j] = zOne/pBase->zTable[i][j];   //invert the base.
        }
      break;
    }
  }

}
*E;
                                        /***********************************************/
  //=========================//           /* This is a collection of functions that      */
 // struct SpecialOperations //           /* operates on the matrix data in special ways.*/
//=========================//            /* Eg. convolution, resize, stream access etc. */
                                        /***********************************************/
struct SpecialOperations
{ ComplexMatrix *pBase;                 //Holds the adress to base class scope.

  SpecialOperations(ComplexMatrix *BasePtr)   //To gain the scope of the base class we store the
  { pBase = BasePtr;                    //baseclass adress in the pBase pointer.
  }
```

```
  //--------------------------//            /*********************************************/
 // The Convolution Operator //             /* Returns the the convolution of two matrices.  */
//--------------------------//              /*********************************************/

ComplexMatrix* operator * (ComplexMatrix &matIn)
//*****************************************************************************************//
// Notice: Convolves two matrices. It reads a matrix as you would read a book, that is first //
//         element in the top left corner, last in the bottom right corner. Result returned. //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*****************************************************************************************//
{
  int iSizeA  = pBase->iNrOfRows*pBase->iNrOfColumns, //The size of the matrices will need to
      iSizeB  = matIn.NrOfRows()*matIn.NrOfColumns(), //be known.

      matInRows = matIn.NrOfRows(),         //Might aswell make a notice of the size
      matInColumns = matIn.NrOfColumns(),   //of the incoming matrix for future use.

      iLength = iSizeA+iSizeB-1,            //The length of the matrix after the
                                            //convolution.
      ImJ,                                  // Element (i-j)
      iRowImJ,                              // The rowindex of element (i-j)
      iColumnImJ,                           // The columnindex of element (i-j)
      iRowI,                                // The rowindex of element (i)
      iColumnI,                             // The columnindex of element (i)
      iRowJ,                                // The Row (j)
      iColumnJ;                             // The Column (j)

      Complex zTemp;                        //A temporarily variable for complex numbers storage.

  ComplexMatrix                             //The matrix to store the convolved result in.
  *matConvolved = new ComplexMatrix( iLength/pBase->iNrOfColumns+1, pBase->iNrOfColumns );

  matConvolved->Clear();                    //Make sure there's no random numbers in it.

  //Convolving:
  for(int i = 0; i < iLength; i++)          //We adress the matrix that holds the convolved
    for(int j = 0; j < iSizeA; j++)         //result, for the rand values - that is when one
    { ImJ = i-j;                            //of the matrices are indexed with <1 or out of
      if( (ImJ >= 0) && (ImJ < iSizeB) )    //its range we are not to calculate anything cuz
      {                                     //there's simply no data in those areas.
        iRowI      = i/pBase->iNrOfColumns; //Element nr. (i)   has this rowindex.
        iColumnI   = i%pBase->iNrOfColumns; //Element nr. (i)   has this columnindex.
        iRowJ      = j/pBase->iNrOfColumns; //Element nr. (j)   has this rowindex.
        iColumnJ   = j%pBase->iNrOfColumns; //Element nr. (j)   has this columnindex.
        iRowImJ    = ImJ/matInColumns;      //Element nr. (i-j) has this rowindex.
        iColumnImJ = ImJ%matInColumns;      //Element nr. (i-j) has this columnindex.
                                            //The actuall convolution: sum(A[i]*B[j-i],i=0..N-1)
        zTemp = pBase->zTable[iRowJ][iColumnJ] * matIn.zTable[iRowImJ][iColumnImJ];
        matConvolved->zTable[iRowI][iColumnI] += zTemp;
      }
    }
  return matConvolved;                      //When we're all done we return the return.
}

  //----------------------//                /*********************************************/
 // The Resize Function //                 /* Resizes the matrix by cropping or zeropadding.*/
//----------------------//                  /*********************************************/

void ResizeMatrix(int iNewNrOfRows, int iNewNrOfColumns)
//*****************************************************************************************//
// Notice: Transform to lesser rows/columns will crop matrix, more of them results in        //
//         zeropadding. Operates on the matrix directly, thus overwriting its old data.      //
//*****************************************************************************************//
{
  //First of all, if new matrix is same size as old matrix there's no need to change anything.
  if( (iNewNrOfRows != pBase->iNrOfRows) || (iNewNrOfColumns != pBase->iNrOfColumns) )
  {
    int  i, j;
    bool bNotLessRows    = (iNewNrOfRows >= pBase->iNrOfRows),
         bNotLessColumns = (iNewNrOfColumns >= pBase->iNrOfColumns);

    //When different size we construct a matrix with the "new size":
    Complex **zPadded = new Complex*[iNewNrOfRows];
    for(i = 0; i < iNewNrOfRows; i++)
      zPadded[i] = new Complex[iNewNrOfColumns];
```

```
    //There are 4 possibilities:
    //1. New matrix has same or more rows aswell as same or more columns:
    if(bNotLessRows && bNotLessColumns)
    {
      for(i = 0; i < pBase->iNrOfRows; i++)                  //In this case we may first of
        for(j = 0; j < pBase->iNrOfColumns; j++)             //copy all data from the old
          zPadded[i][j] = pBase->zTable[i][j];               //matrix into the new one.

      for(i = 0; i < pBase->iNrOfRows; i++)                  //Then we need to "blank" out the
        for(j = pBase->iNrOfColumns; j < iNewNrOfColumns; j++)//part of the new matrix not in
          zPadded[i][j] = 0.0;                               //use. This is done in 2 steps,
                                                             //first we blank out the extra
      for(i = pBase->iNrOfRows; i < iNewNrOfRows; i++)       //columns in row 0 to NewNrOfRows,
        for(j = 0; j < iNewNrOfColumns; j++)                 //then we reset all columns from
          zPadded[i][j] = 0.0;                               //OldNrOfRows to NewNrOfRows.
    }
    //2. New matrix has same or more rows but fewer columns:
    else if(bNotLessRows && !bNotLessColumns)
    {
      for(i = 0; i < pBase->iNrOfRows; i++)                  //First we copy all the data
        for(j = 0; j < iNewNrOfColumns; j++)                 //possible into the new matrix.
          zPadded[i][j] = pBase->zTable[i][j];               //Data in the old "extra" columns
                                                             //will be lost.
      for(i = pBase->iNrOfRows; i < iNewNrOfRows; i++)       //All the new rows, that is from
        for(j = 0; j < iNewNrOfColumns; j++)                 //OldNrOfRows to NewNrOfRows
          zPadded[i][j] = 0.0;                               //will be reset.
    }
    //3. New matrix has fewer rows but same or more rows:
    else if(!bNotLessRows && bNotLessColumns)
    {
      for(i = 0; i < iNewNrOfRows; i++)                      //First we copy all the data
        for(j = 0; j < pBase->iNrOfColumns; j++)             //possible into the new matrix.
          zPadded[i][j] = pBase->zTable[i][j];               //Data in the old "extra" rows
                                                             //will be lost.
      for(i = 0; i < iNewNrOfRows; i++)                      //All the new columns, that is
        for(j = pBase->iNrOfColumns; j < iNewNrOfColumns; j++)//from OldNrOfColumns to
          zPadded[i][j] = 0.0;                               //NewNrOfColumns will be reset.
    }
    //4. New matrix has fewer rows aswell as fewer columns:
    else
    {
      for(i = 0; i < iNewNrOfRows; i++)                      //In this case we only need to
        for(j = 0; j < iNewNrOfColumns; j++)                 //copy the data possible, rest
          zPadded[i][j] = pBase->zTable[i][j];               //will be cropped.
    }

    Complex **zRemove = pBase->zTable;                       //Mark the old table for deletion.
    pBase->zTable = zPadded;                                 //Replace table.

    for(int i = 0; i < pBase->iNrOfRows; i++)                //Delete old matrix. First we
      delete [] zRemove[i];                                  //delete the columns, then
    delete [] zRemove;                                       //the rows.

    pBase->iNrOfRows    = iNewNrOfRows;                      //iNrOfRows and iNrOfColumns also
    pBase->iNrOfColumns = iNewNrOfColumns;                   //need update according to new
                                                             //table size.
  }
}

  //-----------------------------//         /*********************************************/
 // The Absolutevalue Functions //          /* Converts all matrix values to absolutevalues. */
//-----------------------------//           /*********************************************/

ComplexMatrix* ReturnAbsolute()
//*****************************************************************************************//
// Notice: You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*****************************************************************************************//
{
  ComplexMatrix *matAbsolute = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);

  for(int i = 0; i < pBase->iNrOfRows; i++) //Fill matrix with its absolutevalues.
    for(int j = 0; j < pBase->iNrOfColumns; j++)
      matAbsolute->zTable[i][j] = pBase->zTable[i][j].Mod();

  return matAbsolute;                       //Return it when all filled up.
}
```

```
  void MakeAbsolute()
  //*********************************************************************************************//
  // Notice: Operates on the matrix directly, thus overwriting its old data.                     //
  //*********************************************************************************************//
  {
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)       //Replace all values with the
        pBase->zTable[i][j] = pBase->zTable[i][j].Mod(); //corresponding absolute values.
  }

    //----------------------//               /***********************************************/
   // The Rotate Functions //                /* Need a matrix rotated? This is your solution. */
  //----------------------//                 /***********************************************/

  void Rotate180()
  //*********************************************************************************************//
  // Notice: Operates on the matrix directly, thus overwriting its old data.                     //
  //*********************************************************************************************//
  {
    int i,j;                                                 //First we will need a new table
    Complex **zRotated = new Complex*[pBase->iNrOfRows];     //to hold the rotated data. First
                                                             //is to create the rows, the next
    for(i = 0; i < pBase->iNrOfRows; i++)                    //is to create the columns.
      zRotated[i] = new Complex[pBase->iNrOfColumns];

    for(i = 0; i < pBase->iNrOfRows; i++)                    //The rotation itself is pretty
      for(j = 0; j < pBase->iNrOfColumns; j++)               //straightforward, one simply
        zRotated[i][j] = pBase->zTable[pBase->iNrOfRows-i-1][pBase->iNrOfColumns-j-1];//move the
                                                             //data into the new location.
    Complex **zRemove = pBase->zTable;                       //Mark the old table for deletion.
    pBase->zTable = zRotated;                                //Replace table.

    for(i = 0; i < pBase->iNrOfRows; i++)                    //Delete old matrix. First we
      delete [] zRemove[i];                                  //delete the columns, then
    delete [] zRemove;                                       //the rows.
  }

    //--------------------------------//     /***********************************************/
   // The Output to Stream Functions //      /* Outputs the matrix data to file / screen.     */
  //--------------------------------//       /***********************************************/

  void WriteToFile()
  {
    ofstream OutFile("matrix.txt");                 //This function simply outputs the whole
    for(int i = 0; i < pBase->iNrOfRows; i++)       //matrix to the file 'matrix.txt', in the
    { for(int j = 0; j < pBase->iNrOfColumns; j++)  //exact same format as the matrix itself.
        OutFile << setprecision(3) << pBase->zTable[i][j].Mod() << "\t";
      OutFile << "\n";
    }
    OutFile.close();
  }
  void WriteToScreen()
  {
    for(int i = 0; i < pBase->iNrOfRows; i++)       //This function simply outputs the whole
    { for(int j = 0; j < pBase->iNrOfColumns; j++)  //matrix to screen, in the exact same format
        cout << pBase->zTable[i][j].Mod() << "\t";  //as the matrix itself.
      cout << "\n";
    }
  }
}
*S;
                                        /***********************************************/
  //======================//              /* This is a collection of functions that        */
 // struct RowOperations //               /* operates on the matrix rows.                  */
//======================//                /*                                               */
                                        /***********************************************/
struct RowOperations
{ ComplexMatrix *pBase;                 //Holds the adress to base class scope.

  RowOperations(ComplexMatrix *BasePtr) //To gain the scope of the base class we store the
  { pBase = BasePtr;                    //baseclass adress in the pBase pointer.
  }
```

```
  //---------------------//               /*********************************************/
 // The Shift Operators //                /* These operators shifts the matrix up/down.    */
//---------------------//                 /*********************************************/

void operator <<= (int iNrOfShifts)
//*************************************************************************************//
// Notice: When used on rows <<= shifts the rows in the left operand (matrix) up the desired //
//         amount of shifts. The remainding rows from the bottom are filled with zeros.      //
//         Operates on the matrix directly, thus overwriting its old data.                   //
//*************************************************************************************//
{
  //Run through all columns and shift the rows:
  for(int i = 0; i < pBase->iNrOfColumns; i++)
    for(int j = 0; j < pBase->iNrOfRows; j++)
    {
      if((j+iNrOfShifts) < pBase->iNrOfRows)//Rows are shifted
        pBase->zTable[j][i] = pBase->zTable[j+iNrOfShifts][i];
      else                              //and empty spaces filled with zeros.
        pBase->zTable[j][i] = 0.0;
    }
}
void operator >>= (int iNrOfShifts)
//*************************************************************************************//
// Notice: When used on rows >>= shifts the rows in the left operand (matrix) down the       //
//         desired amount of shifts. The rows from the bottom are filled with zeros.         //
//         Operates on the matrix directly, thus overwriting its old data.                   //
//*************************************************************************************//
{
  //Run through all columns and shift the rows:
  for(int i = 0; i < pBase->iNrOfColumns; i++)
    for(int j = pBase->iNrOfRows-1; j >= 0; j--)
    {
      if((j-iNrOfShifts) >= 0)          //Rows are shifted
        pBase->zTable[j][i] = pBase->zTable[j-iNrOfShifts][i];
      else                              //and empty spaces filled with zeros.
        pBase->zTable[j][i] = 0.0;
    }
}
ComplexMatrix* operator << (int iNrOfShifts)
//*************************************************************************************//
// Notice: When used on rows << returns the shifted version of the left operand (matrix).    //
//         It shifts upwards, and the remaining rows from the bottom are filled with zeros.  //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*************************************************************************************//
{
  //Create matrix to store result in:
  ComplexMatrix *matResult = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);

  //Run through all columns and shift the rows:
  for(int i = 0; i < pBase->iNrOfColumns; i++)
    for(int j = 0; j < pBase->iNrOfRows; j++)
    {
      if((j+iNrOfShifts) < pBase->iNrOfRows)//Rows are shifted
        matResult->zTable[j][i] = pBase->zTable[j+iNrOfShifts][i];
      else                              //and empty spaces filled with zeros.
        matResult->zTable[j][i] = 0.0;
    }
  //Return result:
  return matResult;
}
ComplexMatrix* operator >> (int iNrOfShifts)
//*************************************************************************************//
// Notice: When used on rows >> returns the shifted version of the left operand (matrix).    //
//         It shifts downwards, and the remaining rows from the top are filled with zeros.   //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
//*************************************************************************************//
{
  //Create matrix to store result in:
  ComplexMatrix *matResult = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);

  //Run through all columns and shift the rows:
  for(int i = 0; i < pBase->iNrOfColumns; i++)
    for(int j = pBase->iNrOfRows-1; j >= 0; j--)
    {
      if((j-iNrOfShifts) >= 0)          //Rows are shifted
        matResult->zTable[j][i] = pBase->zTable[j-iNrOfShifts][i];
      else                              //and empty spaces filled with zeros.
        matResult->zTable[j][i] = 0.0;
    }
  //Return result:
  return matResult;
}
*R;
                                        /*********************************************/
```

```
  //=========================//              /* This is a collection of functions that        */
 // struct ColumnOperations //               /* operates on the matrix columns.               */
//=========================//                /*                                               */
                                            /***********************************************/
struct ColumnOperations
{ ComplexMatrix *pBase;                     //Holds the adress to base class scope.

  ColumnOperations(ComplexMatrix *BasePtr)  //To gain the scope of the base class we store the
  { pBase = BasePtr;                        //baseclass adress in the pBase pointer.
  }

    //---------------------//                /***********************************************/
   // The Shift Operators //                 /* These operators shifts the matrix left/right. */
  //---------------------//                  /***********************************************/

  void operator <<= (int iNrOfShifts)
  //*********************************************************************************************//
  // Notice: When used on columns <<= shifts the columns in the left operand (matrix) left the //
  //         desired amount of shifts. The columns to the right are filled with zero.          //
  //         Operates on the matrix directly, thus overwriting its old data.                   //
  //*********************************************************************************************//
  {
    //Run through all rows and shift the columns:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)
      {
        if((j+iNrOfShifts) < pBase->iNrOfColumns)//Columns are shifted
          pBase->zTable[i][j] = pBase->zTable[i][j+iNrOfShifts];
        else                                     //and empty spaces filled with zeros.
          pBase->zTable[i][j] = 0.0;
      }
  }
  void operator >>= (int iNrOfShifts)
  //*********************************************************************************************//
  // Notice: When used on columns >>= shifts the columns in the left operand (matrix) right the//
  //         desired amount of shifts. The columns to the left are filled with zeros.          //
  //         Operates on the matrix directly, thus overwriting its old data.                   //
  //*********************************************************************************************//
  {
    //Run through all rows and shift the columns:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = pBase->iNrOfColumns-1; j >= 0; j--)
      {
        if((j-iNrOfShifts) >= 0)                 //Columns are shifted
          pBase->zTable[i][j] = pBase->zTable[i][j-iNrOfShifts];
        else                                     //and empty spaces filled with zeros.
          pBase->zTable[i][j] = 0.0;
      }
  }
  ComplexMatrix* operator << (int iNrOfShifts)
  //*********************************************************************************************//
  // Notice: When used on columns << returns the left-shifted version of the left operand      //
  //         (matrix). The remaining columns from the right are filled with zeros.             //
  //         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
  //*********************************************************************************************//
  {
    //Create matrix to store result in:
    ComplexMatrix *matResult = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);

    //Run through all rows and shift the columns:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = 0; j < pBase->iNrOfColumns; j++)
      {
        if((j+iNrOfShifts) < pBase->iNrOfColumns)//Columns are shifted
          matResult->zTable[i][j] = pBase->zTable[i][j+iNrOfShifts];
        else                                     //and empty spaces filled with zeros.
          matResult->zTable[i][j] = 0.0;
      }
    //Return result:
    return matResult;
  }
```

```
  ComplexMatrix* operator >> (int iNrOfShifts)
  //*****************************************************************************************//
  // Notice: When used on columns << returns the right-shifted version of the left operand     //
  //         (matrix). The remaining columns from the left are filled with zeros.              //
  //         You MUST catch the returned matrix, in order to delete it when no longer in use.  //
  //*****************************************************************************************//
  {
    //Create matrix to store result in:
    ComplexMatrix *matResult = new ComplexMatrix(pBase->iNrOfRows, pBase->iNrOfColumns);

    //Run through all rows and shift the columns:
    for(int i = 0; i < pBase->iNrOfRows; i++)
      for(int j = pBase->iNrOfColumns-1; j >= 0; j--)
      {
        if((j-iNrOfShifts) >= 0)               //Columns are shifted
          matResult->zTable[i][j] = pBase->zTable[i][j-iNrOfShifts];
        else                                   //and empty spaces filled with zeros.
          matResult->zTable[i][j] = 0.0;
      }
    //Return result:
    return matResult;
  }
}
*C;

  //--------------------//                   /***********************************************/
 // The Clear-function //                    /* Fills matrix with zeros.                    */
//--------------------//                     /***********************************************/

void Clear()                                 //Basic function that simply resets the
{ for(int i = 0; i < iNrOfRows; i++)         //matrix. All elements are zero after this.
  for(int j = 0; j < iNrOfColumns; j++)
    zTable[i][j] = 0.0;
}

  //-------------------------//               /***********************************************/
 // The Transponse-function //               /* Returns the transponsematrix.               */
//-------------------------//                /***********************************************/

ComplexMatrix* T()
//*****************************************************************************************//
// Notice: You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*****************************************************************************************//
{
  int i, j;
  ComplexMatrix *matTransponse = new ComplexMatrix(iNrOfColumns, iNrOfRows);

  for(i = 0; i < iNrOfRows; i++)             //Basics are as follows: Rows are swapped with
    for(j = 0; j < iNrOfColumns; j++)        //columns and columns are swapped with rows.
      matTransponse->zTable[j][i] = zTable[i][j];

  return matTransponse;                      //Return adress.
}

  //--------------------------//              /***********************************************/
 // The MakeDiagonal-function //              /* Makes a NxN diagonal matrix from a 1xN matrix.*/
//--------------------------//               /***********************************************/

ComplexMatrix* MakeDiagonal(ComplexMatrix &matIn)
//*****************************************************************************************//
// Notice: You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*****************************************************************************************//
{
  ComplexMatrix *matIdentity;
                                           //Check if dimension is correct.
  if( (matIn.NrOfRows() == 1) && matIn.NrOfColumns() > 0)
  {                                        //If it is we create a new matrix to hold the
    matIdentity = new ComplexMatrix(matIn.NrOfColumns(), matIn.NrOfColumns()); //diagonal matrix
    Clear();                               //and clear it.

    for(int i = 1; i <= matIn.NrOfColumns(); i++)
      *(*matIdentity)(i,i) = matIn(1,i);   //Fill the indentitymatrix with data.
  }
  else                                     //Dimension mismatch? Let user know.
    cout << "\n(ComplexMatrix MakeDiag)Transform only possible with 1:N or N:1 matrices";

  return matIdentity;
}
```

```
  //-------------------------//             /*********************************************/
 // The Data Access Operator //             /* Provides direct access to data in matrix.     */
//-------------------------//              /*********************************************/

Complex* operator () (int iRow, int iColumn)
//*****************************************************************************************//
// When 2 arguments are recieved the adress to the value from the corresponding row and column //
// are returned, if element doesn't exist NULL will be returned.                           //
//*****************************************************************************************//
{ int iRowM1    = iRow-1,
      iColumnM1 = iColumn-1;

  //Return value if element exists:
  if ( (iRowM1 < iNrOfRows) && (iColumnM1 < iNrOfColumns) )
    return &zTable[iRowM1][iColumnM1];
  else
  { cout << "\n(ComplexMatrix '()' operator) Illegal adressing!";
    return NULL;
  }
}
Complex* operator () (int iElementNr)
//*****************************************************************************************//
// When 1 arguments are recieved the adress to the corresponding element in the matrix are     //
// returned (treats matrix as one-dimensional). Faulty adressing means NULL will be returned.  //
//*****************************************************************************************//
{ int iRowM1    = (iElementNr-1)/iNrOfColumns,
      iColumnM1 = (iElementNr-1)%iNrOfColumns;

  //Return value if element exists:
  if( (iRowM1 < iNrOfRows) && (iColumnM1 < iNrOfColumns) )
    return &ComplexMatrix::zTable[iRowM1][iColumnM1];
  else
  { cout << "\n(ComplexMatrix '()' operator) Illegal adressing!";
    return NULL;
  }
}

  //-------------------------//             /*********************************************/
 // The Data Storage Operator //             /* Copies right side matrix into left side matrix*/
//-------------------------//              /*********************************************/

bool operator = (ComplexMatrix &matIn)
/*****************************************************************************************/
/* Notice: This type operation require that the matrices are of same size (and type).          */
/*         Catch the returned boolean to determine whether the operation was successful.       */
/*****************************************************************************************/
{
  //Check if matrix sizes match:
  if( (iNrOfRows == matIn.NrOfRows()) && (iNrOfColumns == matIn.NrOfColumns()) )
  {
    //Copy values from right side matrix to left side matrix:
    for(int i = 0; i < iNrOfRows; i++)
      for(int j = 0; j < iNrOfColumns; j++)
        zTable[i][j] = matIn.zTable[i][j];

    //Success?
    return true;
  }
  else
  { cout << "\n(ComplexMatrix '=' operator) Dimension mismatch.";
    return false;
  }
}

  //-------------------------//             /*********************************************/
 // The Arithmetic Operators //             /* Performs matrix arithmetics.                */
//-------------------------//              /*********************************************/

ComplexMatrix* operator +  (ComplexMatrix &matIn)
//*****************************************************************************************//
// Notice: Overloads the + operator such that it can be used to compute the sum of two matrices//
//         You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*****************************************************************************************//
{                                       //First we check that the matrices are alike,
  if( (iNrOfRows == matIn.NrOfRows()) && (iNrOfColumns == matIn.NrOfColumns()) )
  {                                     //then we create a variable to store result in.
    ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, iNrOfColumns);

    for(int i = 0; i < iNrOfRows; i++)        //The sum of two matrices is simply the sum of
      for(int j = 0; j < iNrOfColumns; j++)   //every element.
        matResult->zTable[i][j] = zTable[i][j] + matIn.zTable[i][j];
    return matResult;                   //If success we return the result, else we
  }                                     //return NULL.
  else
  { cout << "\n(ComplexMatrix '+' operator) Dimension mismatch.";
    return NULL;
  }
```

```
}
bool          operator += (ComplexMatrix &matIn)
//*********************************************************************************************//
// Notice: Overloads the += operator such that it can be used to add the right side matrix to  //
//         the one to the left.                                                                //
//*********************************************************************************************//
{
  if( (iNrOfRows == matIn.NrOfRows()) && (iNrOfColumns == matIn.NrOfColumns()) ) // If
  {                                          //both matrices are alike, we may add them together
    for(int i = 0; i < iNrOfRows; i++)       //and update the table of data accordingly.
      for(int j = 0; j < iNrOfColumns; j++)
        zTable[i][j] += matIn.zTable[i][j];
                                             //If success we return true else we return false.
    return true;
  }
  else
  { cout << "\n(ComplexMatrix '+=' operator) Dimension mismatch.";
    return false;
  }
}
ComplexMatrix* operator -  (ComplexMatrix &matIn)
//*********************************************************************************************//
// Notice: Overloads the - operator such that it can be used to compute the difference         //
//         between two matrices.                                                               //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*********************************************************************************************//
{
  if( (iNrOfRows == matIn.NrOfRows()) && (iNrOfColumns == matIn.NrOfColumns()) )//First we check
  {                                          //that the matrices are alike, then create a
    ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, iNrOfColumns);// variable to store the
                                             //difference in.
    for(int i = 0; i < iNrOfRows; i++)       //The difference between of two matrices is simply
      for(int j = 0; j < iNrOfColumns; j++)  //the difference between each element.
        matResult->zTable[i][j] = zTable[i][j] - matIn.zTable[i][j];
    return matResult;                        //If success we return the result, else we
  }                                          //return NULL.
  else
  { cout << "\n(ComplexMatrix '-' operator) Dimension mismatch.";
    return NULL;
  }
}
bool          operator -= (ComplexMatrix &matIn)
//*********************************************************************************************//
// Notice: Overloads the += operator such that it can be used to subtract the right side       //
//         matrix from the one to the left.                                                    //
//*********************************************************************************************//
{
  if( (iNrOfRows == matIn.NrOfRows()) && (iNrOfColumns == matIn.NrOfColumns()) )//If
  {                                          //both matrices are alike, we may subtract one from
    for(int i = 0; i < iNrOfRows; i++)       //the other by doing so element by element. Result
      for(int j = 0; j < iNrOfColumns; j++)  //is stored in the left side matrix.
        zTable[i][j] -= matIn.zTable[i][j];
                           //If success we return true else we return false.
    return true;
  }
  else
  { cout << "\n(ComplexMatrix '-=' operator) Dimension mismatch.";
    return false;
  }
}
ComplexMatrix* operator *  (int iScalar)
//*********************************************************************************************//
// Notice: Multiplies the whole matrix with a scalar. Result are returned.                     //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*********************************************************************************************//
{                                            //First we create a matrix
  ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, iNrOfColumns);
                                             //to hold the result.
  for(int i = 0; i < iNrOfRows; i++)         //Then we run trough each row
    for(int j = 0; j < iNrOfColumns; j++)    //and column and scale each element.
      matResult->zTable[i][j] = zTable[i][j] * iScalar;
  return matResult;                          //Result are returned.
}
ComplexMatrix* operator *  (double dScalar)
//*********************************************************************************************//
// Notice: Multiplies the whole matrix with a scalar. Result are returned.                     //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*********************************************************************************************//
{                                            //First we create a matrix
  ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, iNrOfColumns);
                                             //to hold the result.
  for(int i = 0; i < iNrOfRows; i++)         //Then we run trough each row
    for(int j = 0; j < iNrOfColumns; j++)    //and coulmn and scale each element.
      matResult->zTable[i][j] = zTable[i][j] * dScalar;
  return matResult;                          //Result are returned.
}
```

```
void          operator *= (int iScalar)
//*********************************************************************************************//
// Notice: Multiplies the whole matrix with a scalar. Result are stored in left side matrix.   //
//*********************************************************************************************//
{
  for(int i = 0; i < iNrOfRows; i++)          //To compute we run through each row
    for(int j = 0; j < iNrOfColumns; j++)     //and column and
      zTable[i][j] *= iScalar;                //scale every element with the same value.
}
void          operator *= (double dScalar)
//*********************************************************************************************//
// Notice: Multiplies the whole matrix with a scalar. Result are stored in left side matrix.   //
//*********************************************************************************************//
{
  for(int i = 0; i < iNrOfRows; i++)          //To compute we run through each row
    for(int j = 0; j < iNrOfColumns; j++)     //and column and
      zTable[i][j] *= dScalar;                //scale every element with the same value.
}
ComplexMatrix* operator *  (ComplexMatrix &matIn)
//*********************************************************************************************//
// Notice: Compute the product of two matrices. Result are returned.                           //
//         You MUST catch the returned matrix, in order to delete it when no longer in use.    //
//*********************************************************************************************//
{
  if(iNrOfColumns == matIn.NrOfRows())        //First we check that the matrices may be
  {                                           //multiplied. If that's the case we'll need
    ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, matIn.NrOfColumns());//a variable to
                                              //store the product in.
    matResult->Clear();                       //First of we fill it with zeros.

    int iRow = 0,                             //We'll also need some new indexes since
        iColumn = 0;                          //resulting matrix will probably have a
                                              //Different shape.
    while(iRow < iNrOfRows)                   //The computation: We'll need to run through each
    {                                         //row and column in the new matrix. The element
      for(int j = 0; j < iNrOfColumns; j++)   //(1,1) will contain the sum of the elements in the
        matResult->zTable[iRow][iColumn] += zTable[iRow][j] * matIn.zTable[j][iColumn];//left side
                                              //matrix 1st row and right side matrix 1st column
      if(++iColumn >= matResult->NrOfColumns())//multiplied elementwise. The calculation needs to
      { iColumn = 0;                          //be repeated for all the elements in the resulting
        iRow++;                               //matrix.
      }
    }
    return matResult;                         //Finally the result are returned, that is if there
  }                                           //was even possible to calculate it in the first
  else                                        //place.
  { cout << "\n(ComplexMatrix '*' operator) Dimension mismatch.";
    return NULL;
  }
}
```

```
  bool          operator *= (ComplexMatrix &matIn)
  //*****************************************************************************************//
  // Notice: Compute the product of two matrices. Result are stored in left side matrix.     //
  //*****************************************************************************************//
  {
    if(iNrOfColumns == matIn.NrOfRows())        //First we check that the matrices may be
    {                                           //multiplied. If that's the case we'll need
      ComplexMatrix *matResult = new ComplexMatrix(iNrOfRows, matIn.NrOfColumns());//a variable to
                                                //store the product in.
      matResult->Clear();                       //First of we fill it with zeros.

      int iRow = 0,                             //We'll also need some new indexes since
          iColumn = 0;                          //resulting matrix will probably have a
                                                //Different shape.
      while(iRow < iNrOfRows)                   //The computation: We'll need to run through each
      {                                         //row and column in the new matrix. The element
        for(int j = 0; j < iNrOfColumns; j++)   //(1,1) will contain the sum of the elements in the
        {
          matResult->zTable[iRow][iColumn] += zTable[iRow][j] * matIn.zTable[j][iColumn];//left side
        }
                                                //matrix 1st row and right side matrix 1st column
        if(++iColumn >= matResult->NrOfColumns())//multiplied elementwise. The calculation needs to
        { iColumn = 0;                          //be repeated for all the elements in the resulting
          iRow++;                               //matrix.
        }
      }
      Complex **zRemove = zTable;               //Mark the address of the old table so we can delete
      zTable = matResult->zTable;               //it, and replace the table.

      for(int i = 0; i < iNrOfRows; i++)        //When deleting a table we first need to delete all
        delete [] zRemove[i];                   //columns, then the rows.
      delete [] zRemove;

      iNrOfRows    = matResult->NrOfRows();     //We also need to update iNrOfRows and iNrOfColumns
      iNrOfColumns = matResult->NrOfColumns();  //according to size of the new table.

      return true;                              //If operation was successful we return true,
    }                                           //else we return false.
    else
    { cout << "\n(ComplexMatrix '*' operator) Dimension mismatch.";
      return false;
    }
  }
};
```

A.5 Filen Complex.h

```
                                                /***********************************************/
       ///////////////////////                  /* Basic struct that defines the datatype that   */
      /// struct Complex ///                   /* is complex numbers. Also contain operator     */
     ///////////////////////                    /* overloads.                                    */
                                                /***********************************************/
struct Complex
{ double dRe, dIm;                              //Stores the information about a complex number
                                                //in rectangular form.
  Complex()
  { dRe = dIm = 0.0;                            //At creation we reset the number.
  }
  double Mod()
  { return(sqrt(dRe * dRe + dIm * dIm));        //It is often practical knowing the absolute value.
  }
  double Re()
  { return dRe;                                 //We might also wish to know the real part
  }
  double Im()
  { return dIm;                                 //or the imaginary part.
  }

    //-------------------//
   // Storing operators //
  //-------------------//

  Complex* operator =  (Complex *zIn)
  { dRe = zIn->dRe;                             //Set the complex number alike the recieved
    dIm = zIn->dIm;                             //complex number.
    return zIn;
  }
  void operator =  (double dInitialValue)       //In case we recieve a double we assume it s a real
  { dRe = dInitialValue; dIm = 0.0;             //value and reset the imaginary part.
  }

    //----------------------//
   // Arithmetic operators //
  //----------------------//

  Complex operator +  (Complex zIn)             //Returns the sum of two complex numbers.
  { Complex zRes;
    zRes.dRe = dRe + zIn.dRe;
    zRes.dIm = dIm + zIn.dIm;
    return zRes;
  }
  void    operator += (Complex zIn)             //Adds the complex number on both sides of the
  { dRe += zIn.dRe;                             //operator together and stores the result in the
    dIm += zIn.dIm;                             //complex number on the left side.
  }
  Complex operator -  (Complex zIn)             //Returns the difference between two complex
  { Complex zRes;                               //numbers.
    zRes.dRe = dRe - zIn.dRe;
    zRes.dIm = dIm - zIn.dIm;
    return zRes;
  }
  void    operator -= (Complex zIn)             //Subtracts the right side complex number from the
  { dRe -= zIn.dRe;                             //left side complex number, and stores result in
    dIm -= zIn.dIm;                             //the left side complex number.
  }
  Complex operator *  (Complex zIn)             //Returns the product of two complex numbers.
  { Complex zRes;
    zRes.dRe = dRe*zIn.dRe - dIm*zIn.dIm;
    zRes.dIm = dIm*zIn.dRe + dRe*zIn.dIm;
    return zRes;
  }
  void    operator *= (Complex zIn)             //Multiplies both complex numbers together and
  { double fReCopy = dRe;
    dRe = dRe*zIn.dRe - dIm*zIn.dIm;            //stores product in the left side complex number.
    dIm = dIm*zIn.dRe + fReCopy*zIn.dIm;
  }
  Complex operator *  (int iScalar)             //Returns the scaled complex number.
  { Complex zRes;
    zRes.dRe = dRe*iScalar;
    zRes.dIm = dIm*iScalar;
    return zRes;
  }
```

```
  Complex operator *  (double dScalar)          //Returns the scaled complex number.
  { Complex zRes;
    zRes.dRe = dRe*dScalar;
    zRes.dIm = dIm*dScalar;
    return zRes;
  }
  void    operator *= (int iScalar)             //Scales the left side complex number by the factor
  { dRe *= iScalar;                             //on the right side, and stores the product in the
    dIm *= iScalar;                             //left side complex number.
  }
  void    operator *= (double dScalar)          //Scales the left side complex number by the factor
  { dRe *= dScalar;                             //on the right side, and stores the product in the
    dIm *= dScalar;                             //left side complex number.
  }
  Complex operator /  (Complex zIn)             //Left side complex number is the dividend and right
  { Complex zRes;                               //side divisor. Quotient is returned.
    double fTmp = zIn.dRe*zIn.dRe + zIn.dIm*zIn.dIm;
    zRes.dRe = (dRe*zIn.dRe + dIm*zIn.dIm)/fTmp;
    zRes.dIm = (dIm*zIn.dRe - dRe*zIn.dIm)/fTmp;
    return zRes;
  }
  void    operator /= (Complex zIn)             //Left side complex number is the dividend and right
  { double fTmp = zIn.dRe*zIn.dRe + zIn.dIm*zIn.dIm; //side divisor. Quotient is stored in the right side
    double fReCopy = dRe;
    dRe = (dRe*zIn.dRe + dIm*zIn.dIm)/fTmp;     //complex number.
    dIm = (dIm*zIn.dRe - fReCopy*zIn.dIm)/fTmp;
  }
  Complex operator /  (int iScalar)             //Left side complex number is the dividend which
  { Complex zRes;                               //is inversely scaled by the scalar divisor.
    zRes.dRe = dRe/iScalar;                     //Quotient is returned.
    zRes.dIm = dIm/iScalar;
    return zRes;
  }
  Complex operator /  (double dScalar)          //Left side complex number is the dividend which
  { Complex zRes;                               //is inversely scaled by the scalar divisor.
    zRes.dRe = dRe/dScalar;                     //Quotient is returned.
    zRes.dIm = dIm/dScalar;
    return zRes;
  }
  void    operator /= (int iScalar)             //Left side complex number is the dividend which
  { dRe /= iScalar;                             //is inversely scaled by the scalar divisor.
    dIm /= iScalar;                             //Quotient is stored in left side complex number.
  }
  void    operator /= (double dScalar)          //Left side complex number is the dividend which
  { dRe /= dScalar;                             //is inversely scaled by the scalar divisor.
    dIm /= dScalar;                             //Quotient is stored in left side complex number.
  }

    //-------------------//
   // Logical operators //
  //-------------------//

  bool    operator == (Complex zIn)             //Compares two complex numbers, returns true if
  { return ( (dRe == zIn.dRe) && (dIm == zIn.dIm) );//they're alike.
  }
  bool    operator != (Complex zIn)             //Compares two complex numbers, returns false if
  { return ( (dRe != zIn.dRe) || (dIm != zIn.dIm) );//they're alike.
  }
};
```

A.6 Filen Lists.h

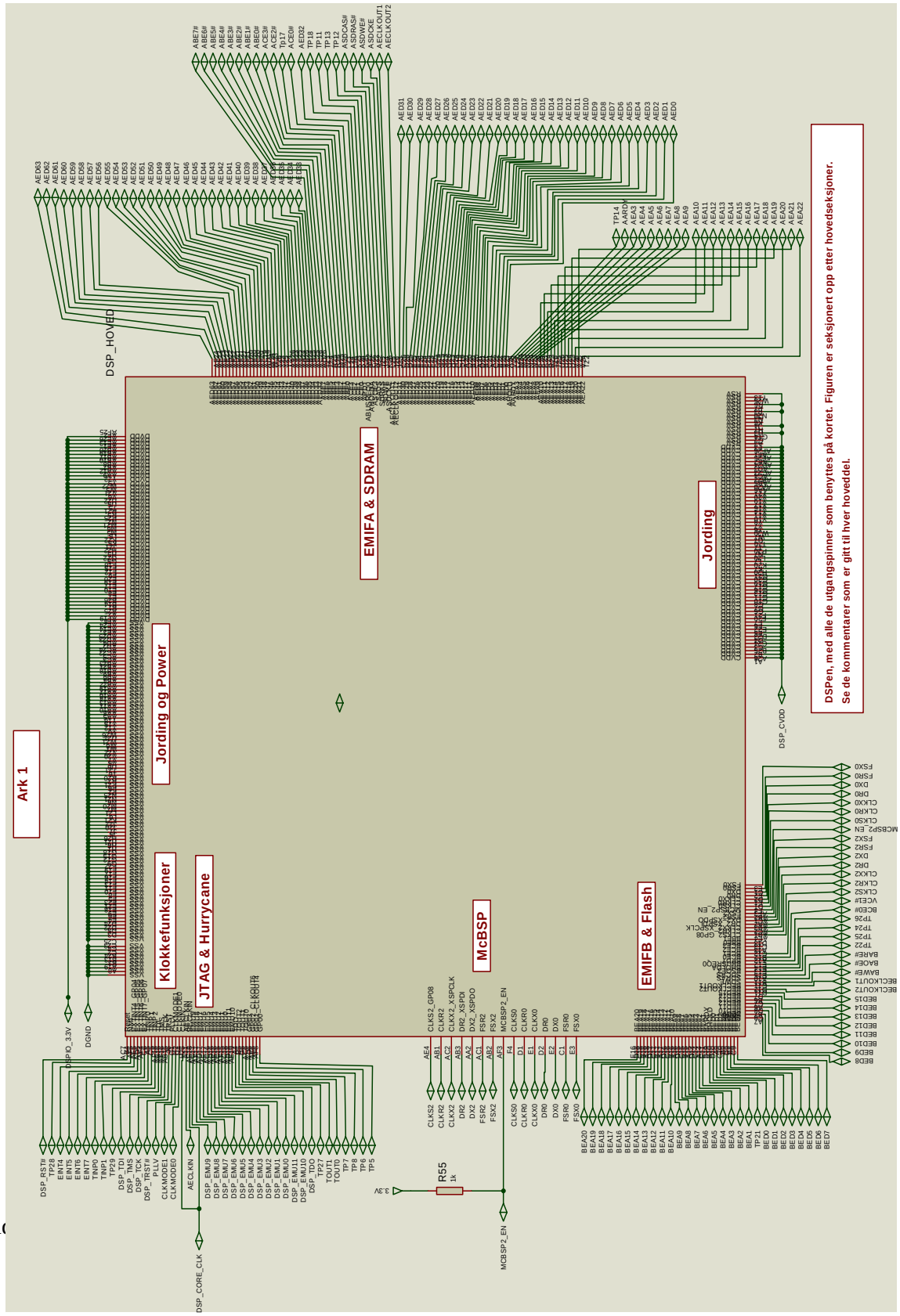
```
#include <iostream>
#include <cstring>

using namespace std;

//Predeclare:
class NumElement;
class CharElement;

//Class defines:
class Element                                   //Basic class to handle.
{
private:
  char cElementType;                            //Defines how this element is sorted.
                                                // 'B'(ase)  -  'N'(umbers)  -  'C'(haracters)
  friend class NumElement;                      //Only these two classes may
  friend class TextElement;                     //manipulate cElementType

public:
  Element()
    { cElementType = 'B'; }                     //Defines element type.
  virtual ~Element()                            //Empty virtual destructor
    {}                                          //(to prevent memory leaks).
  char GetType()
    { return cElementType; }
  virtual int Compare(Element *RecievedElement)
    { return 0; }                               //Dummy.
};


class NumElement: public Element
{
protected:
  int iNumber;

public:
  NumElement()                                  //Constructs itself with data from user.
    { cout << "\nNumber to sort on: "; cin >> iNumber;
      cElementType = 'N';
    }

  NumElement(int iRecievedNr)                   //Constructs itself with recieved data.
    { iNumber = iRecievedNr; cElementType = 'N'; }

  virtual int Compare(Element *RecievedElement)               //Compares own ID-number with the
    { NumElement *tmpNumElement = (NumElement*)RecievedElement;//recieved element ID-number.
      int iDiff = (iNumber - tmpNumElement->iNumber);
      if(iDiff > 0)        return 1;            //In case own ID is larger return 1,
      else if(iDiff == 0)  return 0;            //same return 0,
      else                 return -1;           //and if smaller return -1.
    }
};


class TextElement: public Element
{
protected:
  char *cText;

public:
  TextElement()                                 //Constructs itself with data from user.
    { char ID[80];
      cout << "\nID to sort on: ";        cin.getline(ID, 80);
      cText = new char[strlen(ID)+1];     strcpy(cText, ID);
      cElementType = 'C';
    }
  TextElement(const char *cRecievedText)        //Constructs itself with recieved data.
    { cText = new char[strlen(cRecievedText)+1];
      strcpy(cText, cRecievedText);
      cElementType = 'C';
    }
  ~TextElement()                                //Destructor.
    { delete[] cText;
    }
  virtual int Compare(Element *RecievedElement)
    { TextElement *temp_text = (TextElement*)RecievedElement;
      return (strcmp(cText, temp_text->cText)); //Returns :
                                                //  < 0 if own text is "less" than  the one recieved.
                                                //    0 if own text     equals      the one recieved.
                                                //  > 0 if own text is greater than the one recieved.
    }
```

```
};
class List
{
private:
  struct Node
    { Element *ListElement;
      Node *NextNode;
    };

  Node *FirstNode;
  Node *LastNode;

  int iNrInList;

  Node* FindPosition(Element *RecievedElement);

public:
  List();
  ~List();
  int  NrOfElements();
  bool Add(Element *RecievedElement);
  Element* Remove(int iRecievedNr);
  Element* Remove(const char *cRecievedText);
  Element* RemoveNr(int iRecievedNr);
  bool Destroy(int iRecievedNr);
};

List::List()
{ FirstNode = LastNode = NULL; iNrInList = 0; }
List::~List()
{
  Node *tmpNode;

  if(FirstNode)   //If there's something in the list:
  { do
    { tmpNode = FirstNode;
      FirstNode = FirstNode->NextNode;
      delete tmpNode->ListElement;
      delete tmpNode;
    }while(FirstNode);

    FirstNode = LastNode = NULL;
  }
}
List::Node* List::FindPosition(Element *RecievedElement)//We wish to find the RecievedElement position.
{
  Node *CurrentNode = FirstNode;               //Start with the first node.

  while(    (RecievedElement->Compare(CurrentNode->NextNode->ListElement) > 0) //If RecievedElement has ID
        && (CurrentNode->NextNode != LastNode) )   //greater than our CurrentNode and we're not at end of
list:
  { CurrentNode = CurrentNode->NextNode; }      //We climb on step in list until we find the element that
                                                //is in the position just before where the new node is to
    return CurrentNode;                         //added. We return the adress of this element.
}


int List::NrOfElements()
{ return iNrInList; }
bool List::Add(Element *RecievedElement)
{ Node *NewNode;                                //Pointer to the node containing the new element.
  Node *Position;                               //Points to the node in the position just before
  bool bAdded = false;                          //where the new node is to be inserted.

  if(RecievedElement)                           //If pointer really points to something.
  {
    char cElementType = RecievedElement->GetType();

    if(cElementType == 'N' || cElementType == 'C')//Legal type ('N'umbers or 'C'haracters)?
    {
      //If empty list we make a head and tail in the list:
      if(FirstNode == NULL)                     //Empty list?
      { FirstNode = new Node;                   //Creates the nodes pointing to the first
        LastNode = new Node;                    //and last node in the list.

        FirstNode->NextNode = LastNode;         //FirstNode points to LastNode, and LastNode
        LastNode->NextNode = NULL;              //points to NULL thus terminating it.

        if(cElementType == 'N')                 //Numbered list?
        { FirstNode->ListElement = new NumElement(0);   //Allocates some space for the
          LastNode->ListElement = new NumElement(0);    //elements.
        }
        else                                    //Sorted on characters:
        { FirstNode->ListElement = new TextElement(""); //Allocates some space for the
          LastNode->ListElement = new TextElement("");  //elements.
        }
      }
```

```
      if(cElementType == FirstNode->ListElement->GetType()) //If element is of same type
      { //as rest of list:
        NewNode = new Node;                    //Make the new node.
        NewNode->ListElement = RecievedElement; //Fill it with data.

        Position = FindPosition(RecievedElement);

        NewNode->NextNode = Position->NextNode;
        Position->NextNode = NewNode;
        ++iNrInList;
        bAdded = true;
      }
      else
        cout << "\nYou're trying to add an element of wrong type!";
    }
    else
      cout << "\nElement is of invalid type!";
  }
  else
    cout << "\nPointer to element is corrupt!";

  return bAdded;
}
Element* List::Remove(int iRecievedNr)
{
  Element *tmpElement = NULL;
  NumElement *tmpNumElement;
  Node *RemoveNode;
  Node *Position;

  if(FirstNode)                                //List must have atleast head and tail.
  { if(FirstNode->ListElement->GetType() == 'N')//NumElement:
    { if(NrOfElements())                       //List can not be empty.
      {
        tmpNumElement = new NumElement(iRecievedNr);
        Position = FindPosition(tmpNumElement);

        if(tmpNumElement->Compare(Position->NextNode->ListElement) == 0)
        { RemoveNode = Position->NextNode;
          tmpElement = RemoveNode->ListElement;
          Position->NextNode = RemoveNode->NextNode;
          delete RemoveNode;
          --iNrInList;
        }
        delete tmpNumElement;
      }
      else cout << "\nNo elements in list!";
    }
    else cout << "\nList::Remove(int RecievedNr) only manipulates NumElement!";
  }
  else cout << "\nNo elements in list!";

  return tmpElement;
}
Element* List::Remove(const char *cRecievedText)
{
  Element *tmpElement = NULL;
  TextElement *tmpTextElement;
  Node *RemoveNode;
  Node *Position;

  if(FirstNode)                                //List must have atleast head and tail.
  { if(FirstNode->ListElement->GetType() == 'T')//NumElement:
    { if(NrOfElements())                       //List can not be empty.
      {
        tmpTextElement = new TextElement(cRecievedText);
        Position = FindPosition(tmpTextElement);

        if(tmpTextElement->Compare(Position->NextNode->ListElement) == 0)
        { RemoveNode = Position->NextNode;
          tmpElement = RemoveNode->ListElement;
          Position->NextNode = RemoveNode->NextNode;
          delete RemoveNode;
          --iNrInList;
        }
        delete tmpTextElement;
      }
      else cout << "\nNo elements in list!";
    }
    else cout << "\nList::Remove(const char *cRecievedText) only manipulates TextElement!";
  }
  else cout << "\nNo elements in list!";

  return tmpElement;  //Returns element or NULL.
}
```

```
Element* List::RemoveNr(int iRecievedNr)
{
  Element *tmpElement = NULL;
  Node *RemoveNode;
  Node *Position;

  if(FirstNode)                                 //List must have atleast head and tail.
  { if(NrOfElements())                          //List can not be empty.
    { if(iNrInList >= iRecievedNr)
      {
        Position = FirstNode;

        for(int i = 1; i < iRecievedNr; i++, Position = Position->NextNode) ;

        RemoveNode = Position->NextNode;
        tmpElement = RemoveNode->ListElement;
        Position->NextNode = RemoveNode->NextNode;
        delete RemoveNode;
        --iNrInList;
      }
    }
    else cout << "\nNo elements in list!";
  }
  else cout << "\nNo elements in list!";

  return tmpElement;  //Returns element or NULL.

}
bool List::Destroy(int iRecievedNr)
{
  Element *tmpElement;
  bool bDestroyed = false;

  if(FirstNode)
  {
    if(FirstNode->ListElement->GetType() == 'N')
    {
      if(NrOfElements())
      {
        tmpElement = Remove(iRecievedNr);
        if(tmpElement)
        {
          delete tmpElement;
          bDestroyed = true;
        }
      }
    }
  }
  return bDestroyed;
}
```

VEDLEGG B
SKJEMATEGNINGER ISIS



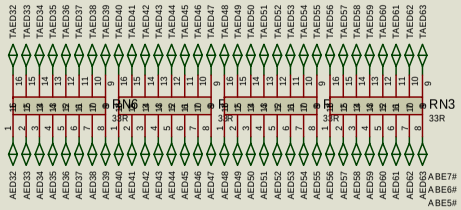
Ark 3

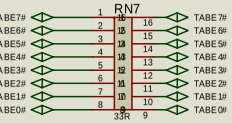
TAEA[22..3]

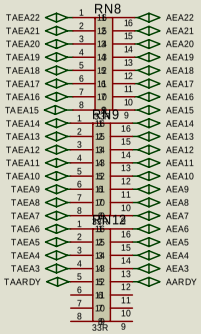
TAED[63..0]

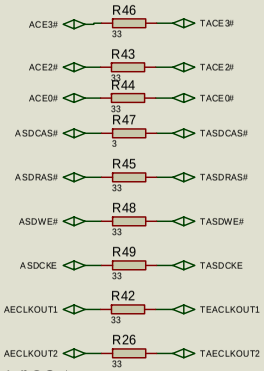
A[22..3]

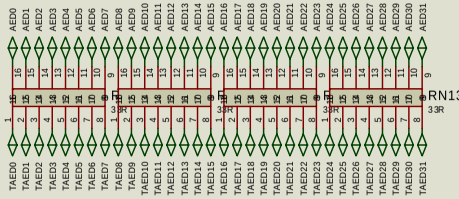
D[63..0]







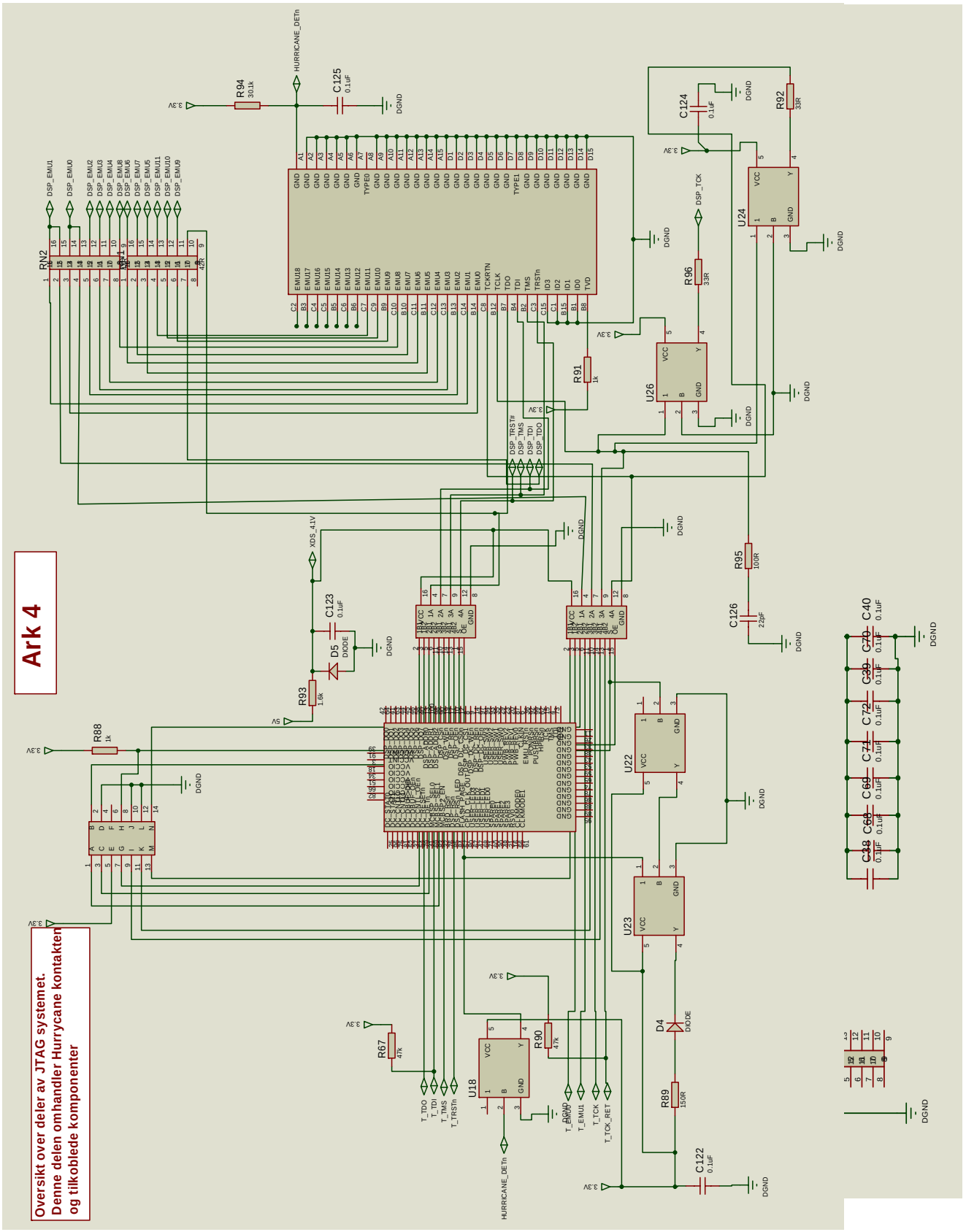


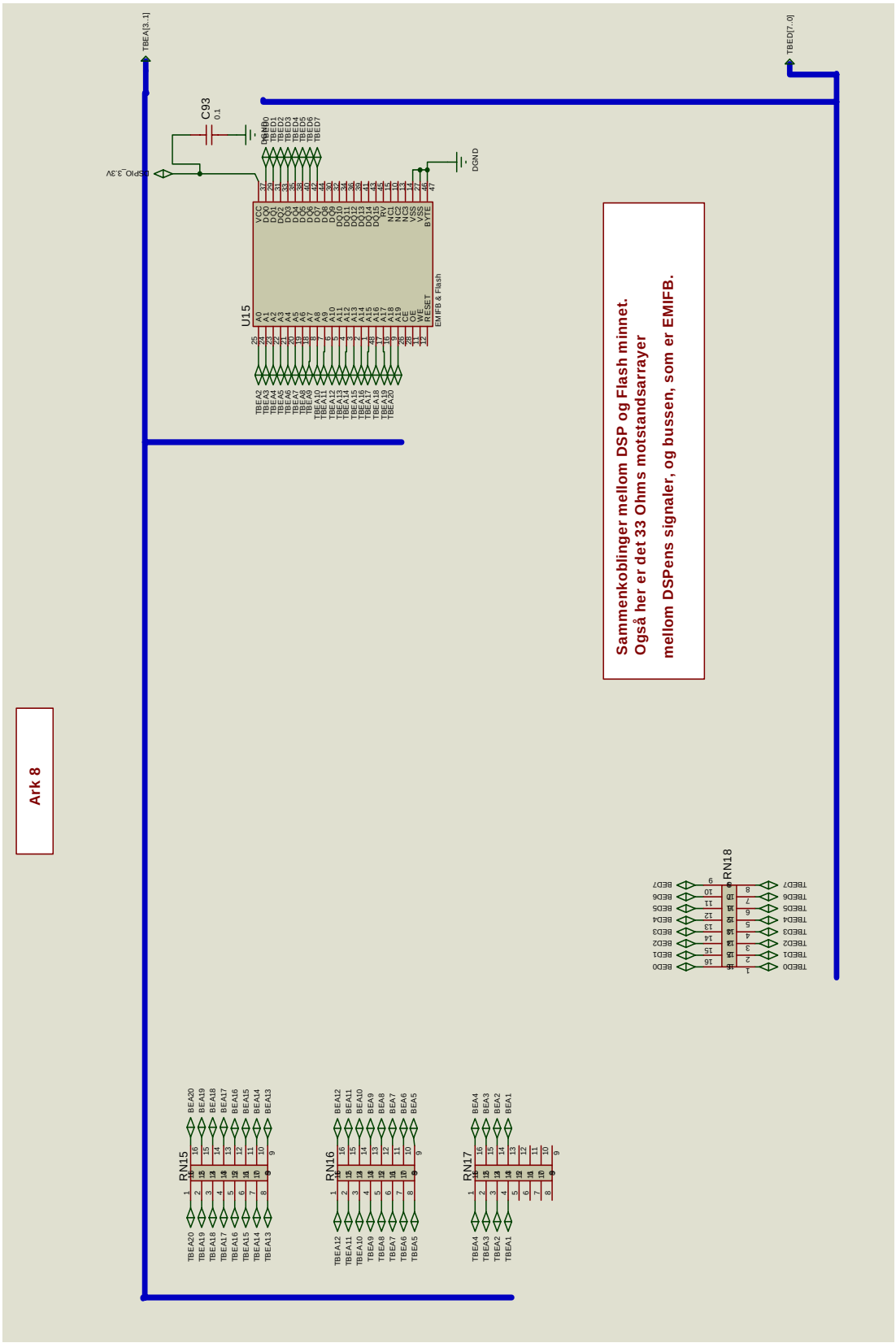


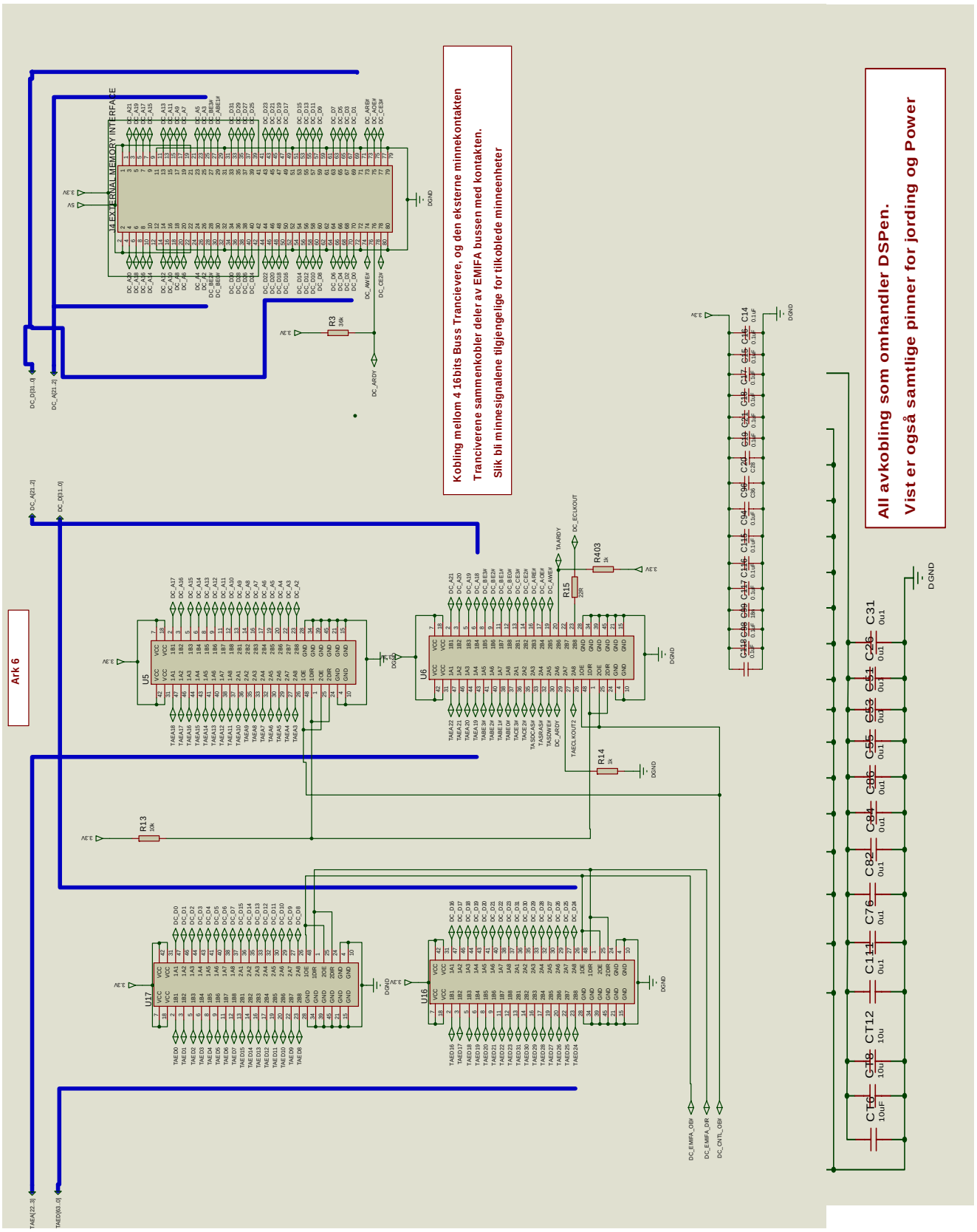
Koblingene mellom DSP og EMIFA bussen. Det er motstandsarrayer på 33kOhm mellom DSPens signaler og bussen DSPen er tegnet på Ark 1

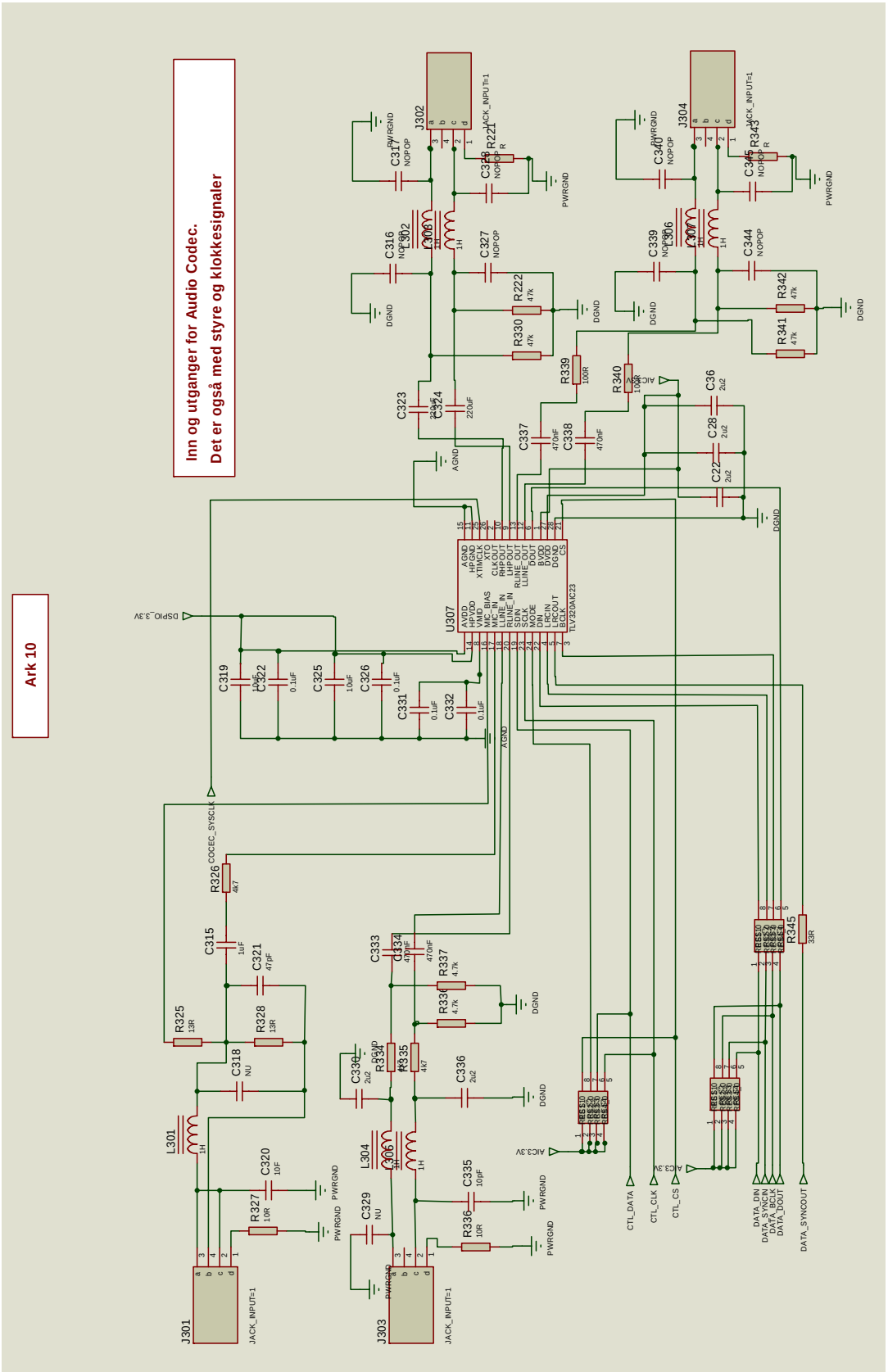
ark 1.

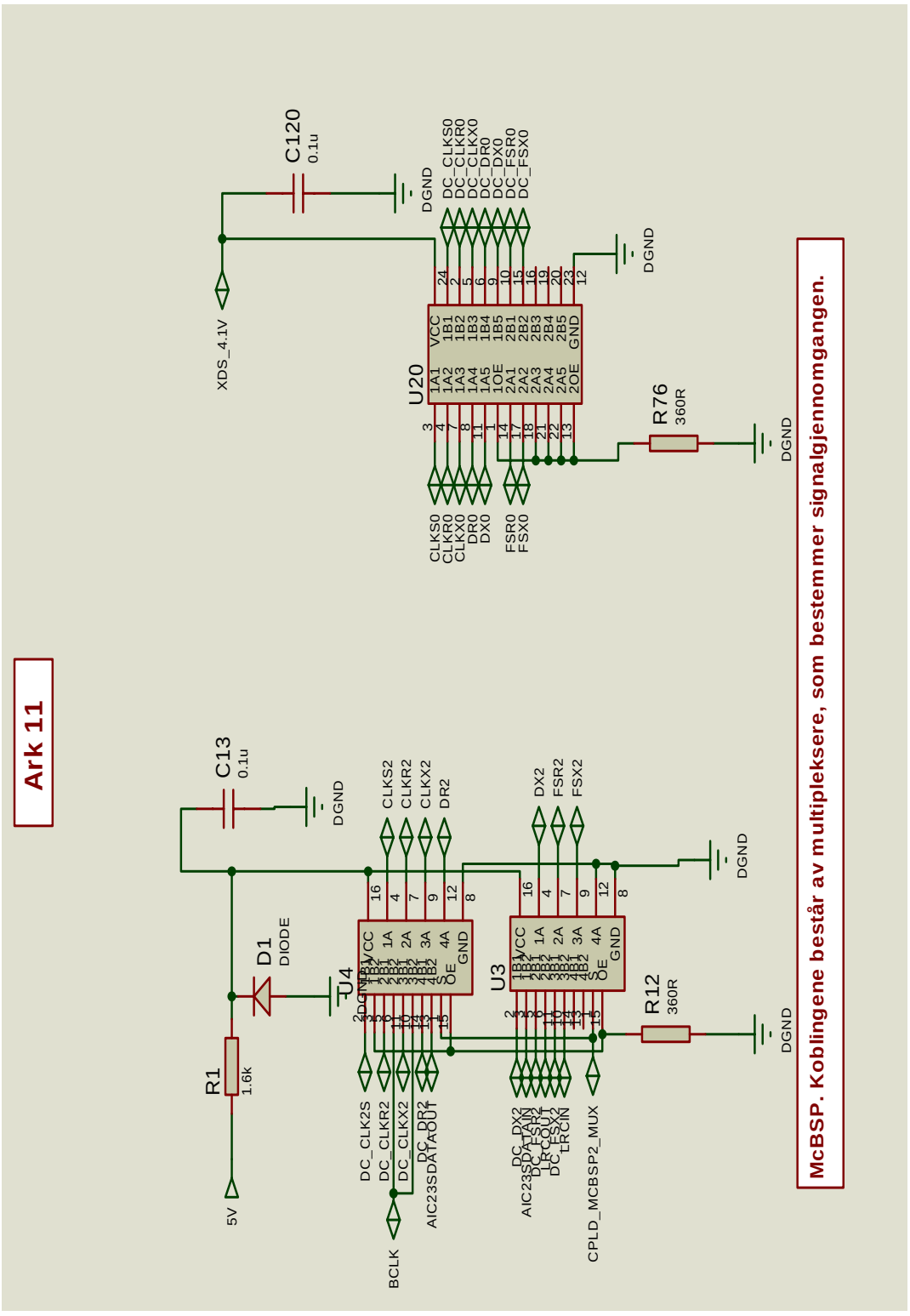
DGND

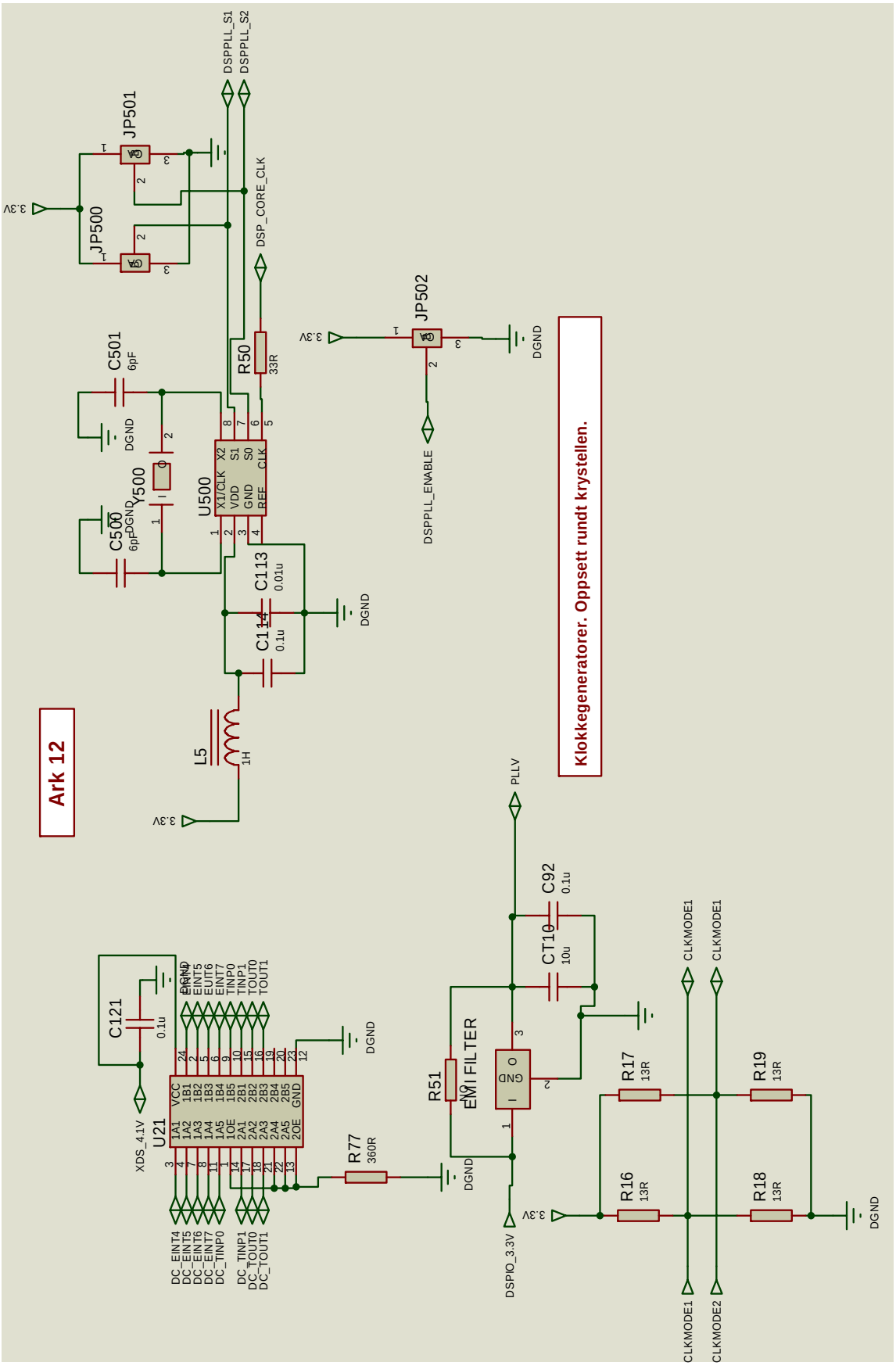

Ark 4
Oversikt over deler av JTAG systemet.
Denne delen omhandler Hurrycane kontakten
og tilkoblede komponenter

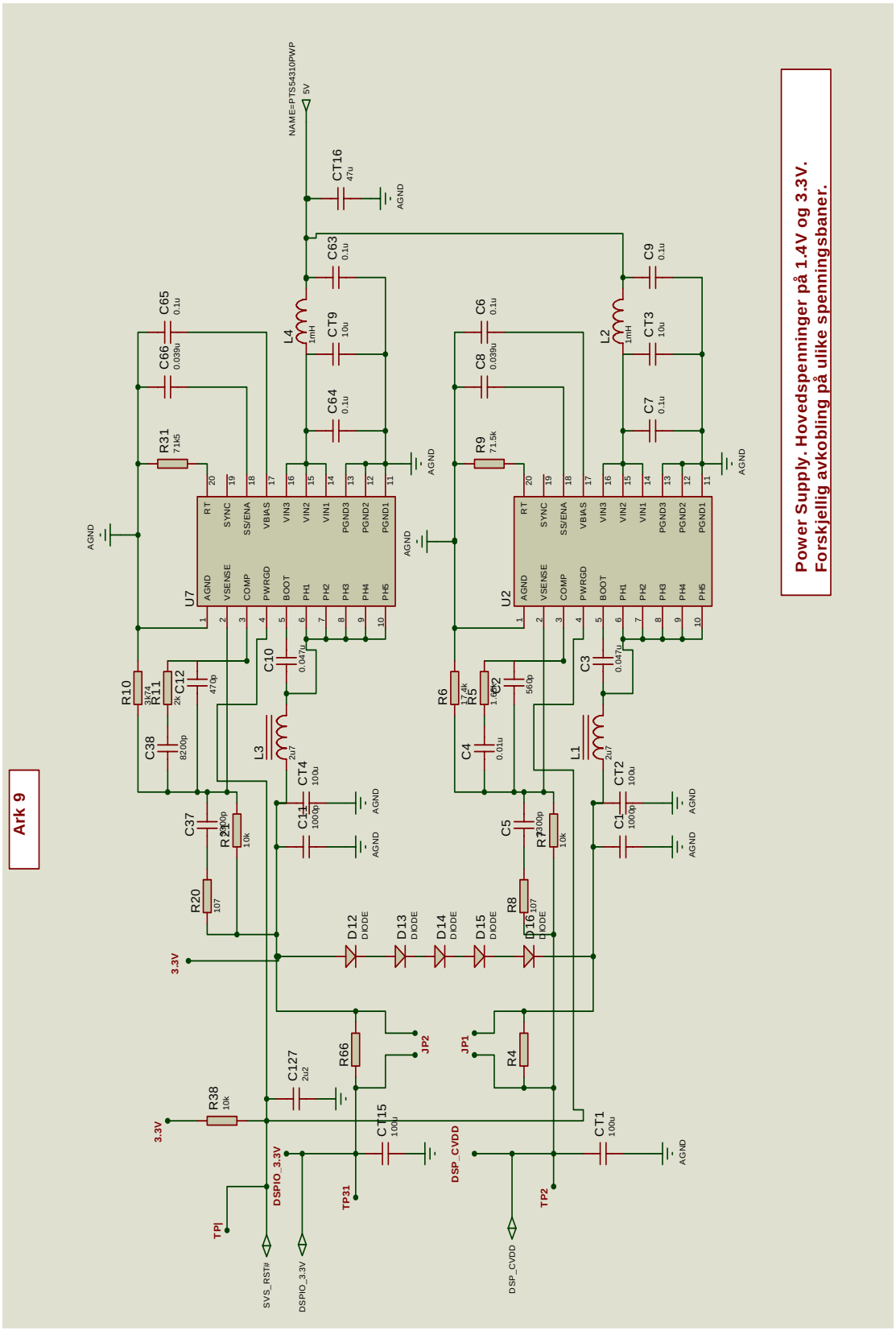

Ark 8
Sammenkoblinger mellom DSP og Flash minnet.
Også her er det 33 Ohms motstandsarrayer
mellom DSPens signaler, og bussen, som er EMIFB.
TBEA[3..1]
TBED[7..0]
C93
0.1
DSPIO_3.3V
DGND
U15
EMIFB & Flash
RN15
RN16
RN17
RN18


Ark 6
Kobling mellom 4 16bits Buss Trancievere, og den eksterne minnekontakten
Tranciverene sammenkobler deler av EMIFA bussen med kontakten.
Slik bli minnesignalene tilgjengelige for tilkoblede minneenheter
All avkobling som omhandler DSPen.
Vist er også samtlige pinner for jording og Power
J4 EXTERNAL MEMORY INTERFACE
U5
U6
U16
U17
R3
R13
R14
R15
R403
DGND

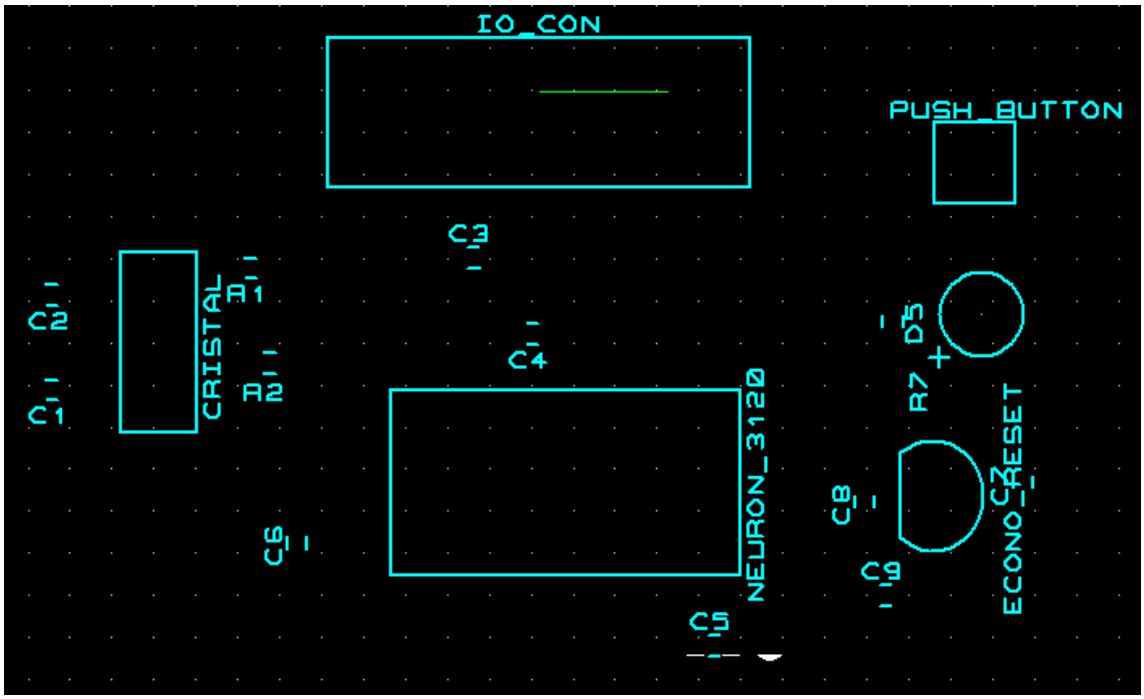

Ark 10
Inn og utganger for Audio Codec.
Det er også med styre og klokkesignaler
U307
TLV320AIC23
J301
J302
J303
J304
L301
L302
L303
L304
L305
L306
L307
L308
R326
R325
R328
R327
R330
R329
R331
R332
R333
R334
R335
R336
R337
R338
R339
R340
R341
R342
R343
R344
R345
R221
R222
C315
C316
C317
C318
C319
C320
C321
C322
C323
C324
C325
C326
C327
C328
C329
C330
C331
C332
C333
C334
C335
C336
C337
C338
C339
C340
C341
C344
C345
C36
C28
C22
DSPIO_3.3V
COCEC_SYSCLK
AGND
DGND
PWRGND
AIC3.3V
CTL_DATA
CTL_CLK
CTL_CS
DATA_DIN
DATA_BCLK
DATA_DOUT
DATA_SYNCOUT
JACK_INPUT=1


Ark 11
C120
0.1u
DGND
XDS_4.1V
U20
VCC
GND
R76
360R
C13
D1
DIODE
U4
U3
R1
1.6k
5V
R12
BCLK
CPLD_MCBSP2_MUX
McBSP. Koblingene består av multipleksere, som bestemmer signalgjennomgangen.

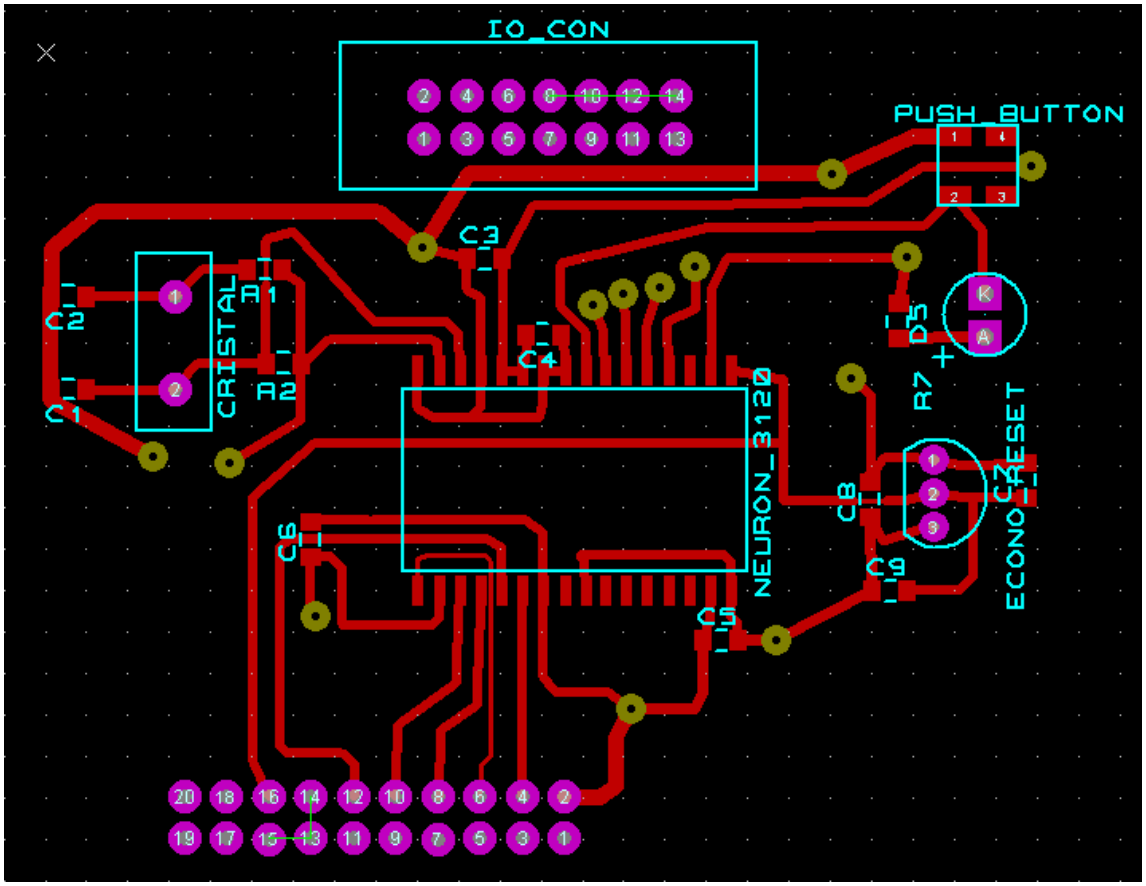

Ark 12
Klokkegeneratorer. Oppsett rundt krystellen.
DSPPLL_S1
DSPPLL_S2
DSP_CORE_CLK
DSPPLL_ENABLE
JP500
JP501
JP502
3.3V
DGND
C501
6pF
C500
6pF
Y500
U500
XJ/CLK
VDD
GND
REF
X2
S1
S0
CLK
R50
33R
C113
0.01u
C114
0.1u
L5
1H
PLLV
C92
0.1u
CT10
10u
EMI FILTER
R51
0
R77
360R
C121
0.1u
U21
XDS_4.1V
DC_EINT4
DC_EINT5
DC_EINT6
DC_EINT7
DC_TINP0
DC_TINP1
DC_TOUT0
DC_TOUT1
DSPIO_3.3V
R16
13R
R17
13R
R18
13R
R19
13R
CLKMODE1
CLKMODE2


Ark 9
Power Supply. Hovedspenninger på 1.4V og 3.3V.
Forskjellig avkobling på ulike spenningsbaner.
NAME=PTS54310PWP
5V
U7
U2
AGND
3.3V
DSP_CVDD
DSPIO_3.3V
SVS_RST#
TP1
TP2
TP31
JP1
JP2

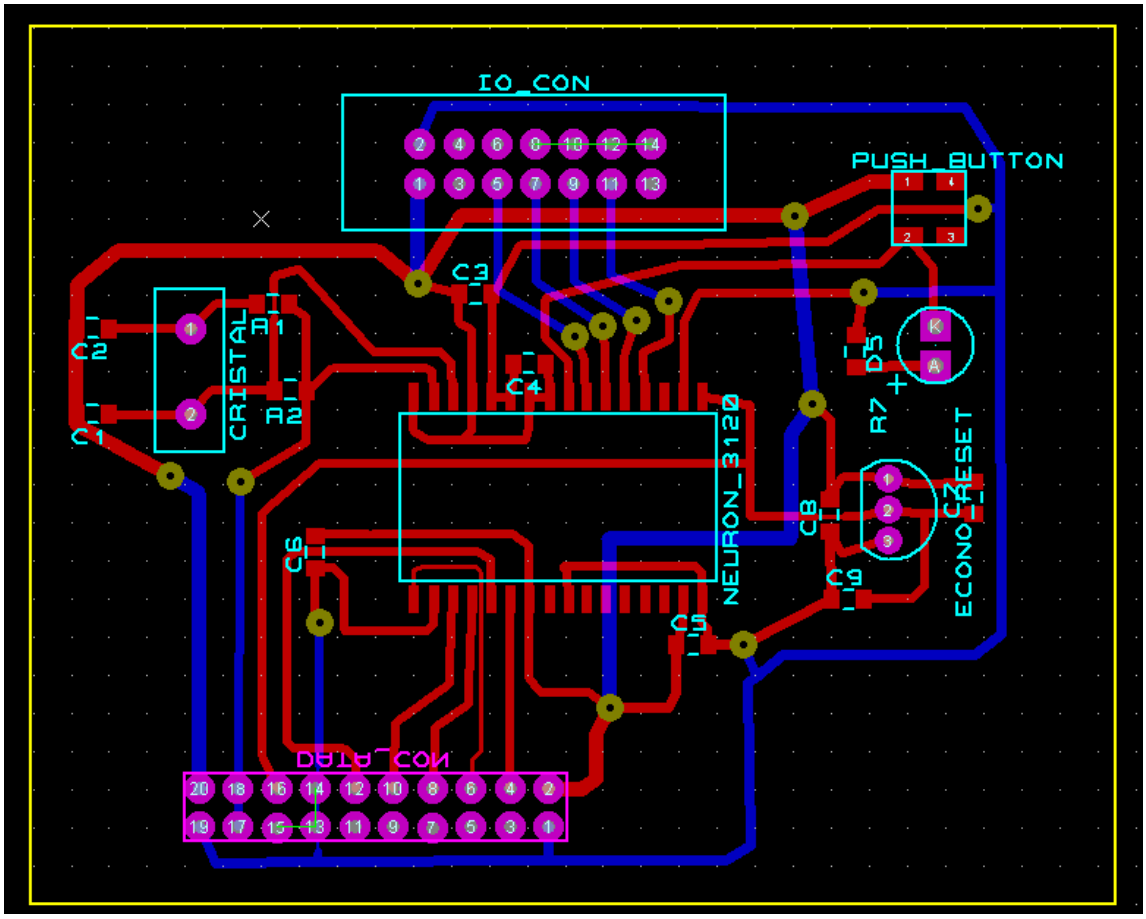
VEDLEGG C
PRINTSKJEMAER ARES



Top Silk for Binde kort



Top Silk og Top Copper for Binde kort



Fullstendig lagoversik for Binde Kort

VEDLEGG D
NEURON KODE

```
//********************** Pragma Functions *************************

#pragma enable_io_pullups  //using the internal built inn pull-up resistors

//********************** Network Variables ************************
network output sync SNVT_switch nvo_lightstatus; //snvt for controlling light sequenceing.
//Compatible also for dimming funnctions. Standard definitions in the Master SNVT List

network output boolean nvo_bitarray[8];
//The paralell bit sequence given from the DSP are stored here. The bits are sent in 2 sequences, 4 bits in each sequence.
// Total integer value of the bit sequence in the array point out a unique SNVT command for the SNVT_switch.
// Each index in the array are given as a boolean, since we are only storing one bit for each index.
//Using integer would use more memory than we would need, for creating this array.

//************************  I/O Declarations  *********************

IO_0 input bit inputsignal_1;         //Using IO_0 - IO_3 for the signal inputs.
IO_1 input bit inputsignal_2;   //The bit sequence from the DSP are sent in two 4bits operations.
IO_2 input bit inputsignal_3;
IO_3 input bit inputsignal_4;


//************************ STRUCTURES *****************************

 typedef struct {
                                                unsigned                on;
                                                unsigned      off;
                                                signed       dimm_20;
                                                signed       dimm_50;
                                                } State;
//The State struct contains all the possible conditions a light can have, using the SNVT_switch.
// The struct names are corresponding with the desired function, but it is important to remember that the functions
//are only defined by us, and that the SNVT_switch in general can be used also in other ways.
//The thought behind using a struct, is that we can have many lights connected to the network. By using a struct,
//we do not need to define a new set of variables for each light we want to connect. This, in the end, allows for grater
//flexibility, and saves memory.

//************************ VARIABLES ******************************
State light;   //Light is derived from the struct State, containing all the states of the struct.
int state_sum; //testvariabel for valg av tilstand, ut fra bitarrayens totalsum
int weight_table[8]  = {1,2,4,8,16,32,64,128};

//A set of pre defined weighting integers, used to weight bits placed after the binary system, to the descimal system.
//By multiplying each bit with it`s corresponding weighting, we get an integer value, we can compare with our int variables
//inside tha state struct.

int n,m; //counter variables


mtimer reset_timer;                                     //Timer which initiates the reset sequence for the nvo_bitarray
mtimer aritmetic_timer;                                 //Initiates the weighting process of each bit
mtimer init_timer;                  //Time the DSP has to send out the value of the 4 MSBs


//********************** Main Program *****************************

when (reset){   //initializing variables when the chip is reseted
   nvo_lightstatus.value = 0; //initialising the standard states of the SNVT_switch
            nvo_lightstatus.state = 0;
   light.on = 131;                  //Integer values corresponding to the bit sequence in the bit array.
   light.off  = 133;        //The values are presented as the bit sequence, multiplied by it`s weighting factor,
   light.dimm_20 = 137;                     //according to each bits position in the array.
   light.dimm_50 = 145;
} //End of Reset sequence
```

```
when(io_changes(inputsignal_1) to 1)  //when the LSB changes state to high, we begin to fill in the first bit sequence
{
                nvo_bitarray[0] = io_in(inputsignal_1);  //Filling the indexes with correct bits
                        nvo_bitarray[1] = io_in(inputsignal_2);  //The 4 least significant bits in this process
                        nvo_bitarray[2] = io_in(inputsignal_3);
                        nvo_bitarray[3] = io_in(inputsignal_4);
              init_timer = 5000;   //Time window before the next sequence of bits must be sent
}

when(timer_expires(init_timer)){   //next 4 bits can be recieved

      nvo_bitarray[4] = io_in(inputsignal_1);
                        nvo_bitarray[5] = io_in(inputsignal_2);  //The 4 most significant bits
                nvo_bitarray[6] = io_in(inputsignal_3);
                nvo_bitarray[7] = io_in(inputsignal_4);

                        aritmetic_timer = 250;  //time until the wetighting sequnce begins
}

when(timer_expires(aritmetic_timer)){  //the weighting process may begin

  for(n=0; n<=7; n++)//looping through the array, containing a total of eight indexes (0-7)
   {
                  state_sum += nvo_bitarray[n]*weight_table[n];
      //This for loop weights each bit, according to it`s position in the array. The value returned is compared
                        // to the integer values of our SNVT_switch variables, executing the desired command if the correct match is found.

   }  //END of for loop



   if(state_sum == light.on)                    //When the integer value equals the desired integer value for the .on command
              { nvo_lightstatus.state = 1;          //Setting the .state variable of the SNVT to high, when desired value is
                                                                                                  //extracted from the bitarray. This will turn on the
connected light.

   }
              else if(state_sum == light.off) ////When the integer value equals the desired integer value for the .off command
   {
     nvo_lightstatus.state = 0;  //turning of the light, by setting the .state variable to low

              }
              else if(state_sum == light.dimm_20) //When the integer value corresponds the desired value for the .dimm_20 command
              {
                nvo_lightstatus.value = 50;  //Dimming the light down to 20% of full value

   }
   else if(state_sum == light.dimm_50)  //When the integer value equals the desired integer value for the .dim_50 command
   {
     nvo_lightstatus.value = 100;   //dimming the light down to 50% off full value.

              }

              reset_timer = 1000; //Time before the reset sequence is initiated

}

when(timer_expires(reset_timer)){
     if(nvo_bitarray[7]==1) //MSB set to 1 means that the whole array has been filled with the desired
                      //bit sequence. This ensures that reset only operates under thid condition.
                        {
                                        for(m=0; m<=7; m++)
        nvo_bitarray[m] = 0; //Setting all the indexe`s in the array to zero. The array is then ready for the
                        //next bit sequence.

                                        state_sum = 0; //setting the overall integer sum to zero.

      }// END of if sentence
}
```