HØGSKOLEN I BERGEN

Avdeling for Ingeniørutdanning
Studieretning Elektronikk

REFERANSESIDE FOR HOVEDPROSJEKT

Rapportens tittel: Overvåking av luftkanon fra Bolt Technology Corporation	*Dato:* 4. Mai 1998
	Rapportnummer: *1*
Forfatter(e): Erik (...) & Eirik (...)	*Antall sider u/vedlegg:* 62
	Antall sider vedlegg: 18
Avdeling/linje: Elektronikk	*Antall CD - ROM:* 1
Vegleder ved avdeling: Saba Mylvaganam	*Gradering:* Ingen
Merknader:	

Oppdragsgiver: Universitetet i Bergen, Institutt for Fast Jords Fysikk	*Oppdragsgivers referanse:*
Oppdragsgivers kontaktperson: Ole Meyer	*Telefon:* 55583421

Sammendrag:

Oppgaven tar for seg hvordan en kan overvåke stabiliteten til 8 luftkanoner ved å undersøke strømpulsen som avfyrer dem. Strømpulsene kalles solenoidstrømmer. Ved å sample spenningen over en drop-motstand er det mulig å kunne lese dataene inn i PC`en, og vise disse på en PC-monitor. Til hardwaredelen er det benyttet et prototypekort. Dette kortet utfører sampling av 8 kanaler ved hjelp av multipleksing. Det inneholder også en minnekrets for mellomlagring av data.

Programmet som utfører innlesning og visning av data, er utviklet i Visual Basic 5.0. Innlesningen er styrt av et interrupt. Dette gjøres ved hjelp av en DriverX-kontroll utviklet av Tetradyne.co. Plotting av data er gjort med Olectra Chart.

Stikkord:

Luftkanoner	Solenoidstrøm	Prototypekort

Høgskolen i Bergen, Tlf. 55 58 75 00 Fax 55 58 77 90 E-post: post@hib.no
Postadresse: Postboks 6030, 5020 BERGEN Besøksadresse: Nygårdsgaten 112, Bergen

Forord

Denne rapporten beskriver en hovedoppgave gitt ved Høgskolen i Bergen våren 1998. Høsten 97 ble hovedoppgavene lagt frem. Vi hadde mange å velge mellom, og vi valgte tilslutt en oppgave gitt av Universitetet i Bergen. Den går ut på å overvåke luftkanoner som blir brukt til seismiske undersøkelser, ved å se på strømpulsen som avfyrer kanonene.

Grunnen til at vi valgte denne oppgaven var at vi høsten 1997 hadde en prosjektoppgave i faget sensorikk, der vi skulle finne ut hvordan hydrofoner og geofoner ble brukt til seismiske undersøkelser. Gjennom denne prosjektoppgaven kom vi i kontakt med Ole Meyer ved Universitetet i Bergen, Institutt for den Faste Jords Fysikk. Han hadde flere interessante alternativer til hovedoppgave, og var også villig til å lytte til våre ønsker. Oppgavene var rene software oppgaver. Da vi kunne tenke oss en oppgave som også inneholdt hardware, definerte Meyer en annen oppgave som var en kombinasjon av begge deler.

Ole Meyer hadde lest en artikkel om at en kunne finne ut mye om en lufkanons tilstand ved å studere strømpulsen som avfyrer kanonen. Dette var grunnen til at han lagde oppgaven. Vi fant denne oppgaven svært interessant da den inkluderer både software og hardware.

Rapporten er laget på en slik måte at hovedkapitlene gir en inngående beskrivelse av prosjektet. Denne delen egner seg for alle interesserte med en viss kjennskap til elektronikk og data. For dem som ønsker en grundigere beskrivelse om de forskjellige emner henvises det til appendiks bak i rapporten. All programkode, kretstegninger, flytskjema samt skisse over eksisterende system finnes i vedlegg. I tillegg til rapporten er det også laget en egen brukermanual som er beregnet for dem som skal bruke produktet.

Vi har gjennom denne oppgaven fått verdifull kunnskap om hvordan en PC fungerer og hvilke muligheter som finnes når det gjelder å kombinere hardware og software. Vi vil derfor rette en stor takk til vår eksterne veileder Ole Meyer. Han har vist stor interesse for vårt prosjekt, og har hatt evne til å veilede oss på en fremragende måte. Vi vil derfor anbefale andre studenter til å velge hovedoppgave fra Universitetet i Bergen. Vi vil også takke vår interne veileder fra Høgskolen i Bergen Saba Mylvaganam, samt Anders (...) og Christian (...) ved Institutt for den Faste Jords Fysikk ved Universitetet i Bergen.

_____ _____

Eirik (...) Erik (...)

INNHOLDSFORTEGNELSE

1 Sammendrag

Under marin-seismiske målinger benyttes luftkanoner til å generere trykkbølger i vann. Luftkanonene har etter tid en tendens til å bli ustabile. Rapporten inneholder en løsning som kan brukes til å avdekke ustabilitet i kanonen på et tidlig tidspunkt. Dette gjøres ved å undersøke strømpulsen som avfyrer kanonen. Strømpulsen vil ha et "hakk". Plasseringen av dette gir informasjon om kanonens stabilitet.

Det ble bestemt at informasjonen skulle vises på en PC-monitor. Denne PC`en skal benyttes sammen med et eksisterende system for styring av luftkanoner (se vedlegg 1). Måten en PC knyttes sammen med perifere innretninger, er gjennom ekspansjonskort.

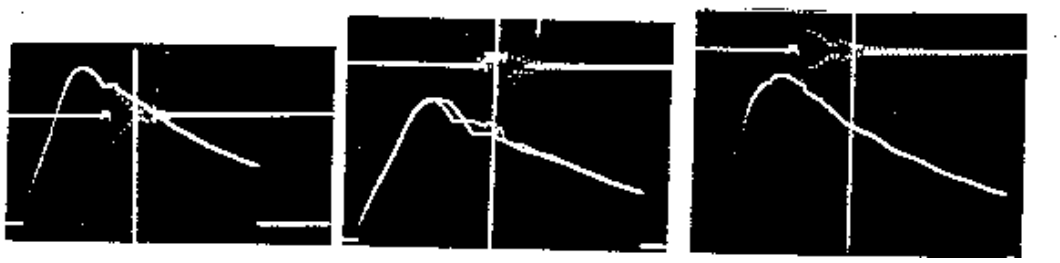
Vi har utviklet hardware på et prototypekort til PC, som gjør det mulig hente inn informasjon om luftkanonene og overføre disse til PC. Når prototypekortet plasseres i en PC, kommer det i kontakt med CPU gjennom I/O-busser (Input/Output). Prototypkortet er forhåndsdefinert til å ha et adresseråde fra 300H (heksadesimalt) til 31FH.

For å hente data inn i PC, må det analoge signalet digitaliseres. Dette gjøres ved hjelp av en 12 bits A/D-omformer. Samplingshastiget og samplingsintervall styres av en programmerbar timer. Instillingene til timeren bestemmes av brukeren, og kan endres når som helst. Ved hjelp av en 8 kanals mutiplekser kan vi innhente informasjon om 8 kanoner samtidig, og sample disse med èn A/D-omformer. Innlesning av data til PC skjer med en annen hastighet enn samplingen. Dette medfører at dataene må lagres midlertidig i et buffer. Som buffer benytter vi en FIFO (First In First Out). Dette er et minne som gjør at vi kan lese og skrive asynkront.

For å kunne vite når PC`en skal starte innlesning benytter vi et interrupt. Dette vil bli generert når FIFO er halvfull. Etter at samtlige data er innlest vil alle strømforløpene bli vist i forskjellige diagrammer. Brukeren har og mulighet til å undersøke historien til èn enkelt kanon, for å se om det er endringer i stabiliteten. I tillegg til dette er det laget en funksjon som kan finne "hakket" i kurven.

2 Utdypning og avgrensning av problemet

Ved marinseismiske undersøkelser benyttes luftkanoner som kilde for å generere trykkbølger i vann (se appendiks A). Bakgrunnen for dette prosjektet er et ønske om unngå kostbare avbrytelser av undersøkelsen på grunn av at luftkanonene ikke lenger er stabile. I en artikkel gitt av oppdragsgiver er det beskrevet hvordan en kan bestemme en luftkanon sin stabilitet ved å analysere strømforløpet til solenoidstrømmen. Det er denne som avfyrer kanonen. Det viser seg at strømpulsen har et hakk (back notch) som kan være med på å si noe om stabiliteten til kanonen (se figur 1.1). Hakket oppstår fordi den bevegelige delen i solenoiden (”plunger”) induserer en motspenning i bevegelsesøyeblikket. I en normal kanon skal ”hakket” ligge på stigende flanke på strømpulsen. Hvis hakket befinner seg på toppen av kurven, begynner kanonen å bli ustabil. Dersom hakket er på den negative flanken, vil kanonen være ustabil, og det er vansklig å forutse avfyringstidspunkt. Det er derfor et ønske om å kunne se på denne strømpulsen mens en utfører seismiske undersøkelser. Kanonene avfyres med ca 10 sekunders mellomrom så lenge undersøkelsen pågår.



Figur 1.1 Figur som viser «hakket» i solenoidstrømmen. (Kilde: Geophysical Company of Norway)

Problemet blir derfor å vise solenoidstrømmen for hvert skudd fortløpende på en oversiktlig måte. Målet er derfor å presentere denne strømmen på en PC. For å få dette til må en først konvertere strømpulsen til en spenning. Denne spenningen må A/D - konverteres før den kan leses inn av PC. Det må utvikles et program som leser inn samplingene fra A/D - omformeren og som kan presentere dataene grafisk på skjermen. Dette må skje mellom hvert skudd.

Vi så derfor på forskjellige måter å gjøre dette på:

1) Lage et kort med A/D omformer på som skulle sample kanonene og lagre disse i en FIFO (minnekrets), for deretter å lese disse inn via parallellporten på PC`en.

2) Lage et kort med en microcontroller som skulle styre samplingen av kanonene, og lagre dataene i en RAM-brikke. Når samplingen var ferdig skulle vi deretter lese inn dataene via seriellporten.

3) Bruke et prototypkort som stikkes inn i PC`en. På dette monteres all hardware som skal brukes. Samplingene skal også her mellomlagres i en FIFO.

I samråd med oppdragsgiver gikk vi inn for alternativ 3. Grunnen til at dette alternativet ble valgt er at en slik løsning er svært fleksibel og lar seg lett utvide eller modifisere senere. Det ansees også som en fordel å flytte hardwaren inn i PC'en. Alternativ 1 innebærer en komplisert innlesning gjennom parallellport og er vanskelig å forandre senere. Alternativ 2 innebærer programmering av microcontrollere, trolig i assembly-kode, noe som vanskeliggjør videreutvikling av prosjektet av andre. På denne bakgrunn ble alternativ 2 forkastet.

Vi måtte nå finne ut hvordan et prototypekort fungerte. Vi måtte også finne ut hvilken hardware som trengtes til prototypekortet. Det ble bestemt at strøm til spenningsomformingen skulle skje ved hjelp av en ekstern hardware som UiB (Universitetet i Bergen) kunne ta seg av. Denne hardwaren omfattet i første omgang en drop-motstand og et galvanisk skille. Etter at vi kom igang med oppgaven kom det frem ønske om å se på muligheten for å overvåke 8 kanoner samtidig. Dette lot seg gjøre ved hjelp av multipleksing og vi gikk inn for en slik løsning. Dette førte imidlertid til visse problemer med filtrering og forsterkning av signal. Det ble derfor bestemt at dette også skal inngå i den eksterne hardwaren. Til utvikling av hardware skulle det benyttes en DOS basert PC med Turbopascal som programmeringsspråk.

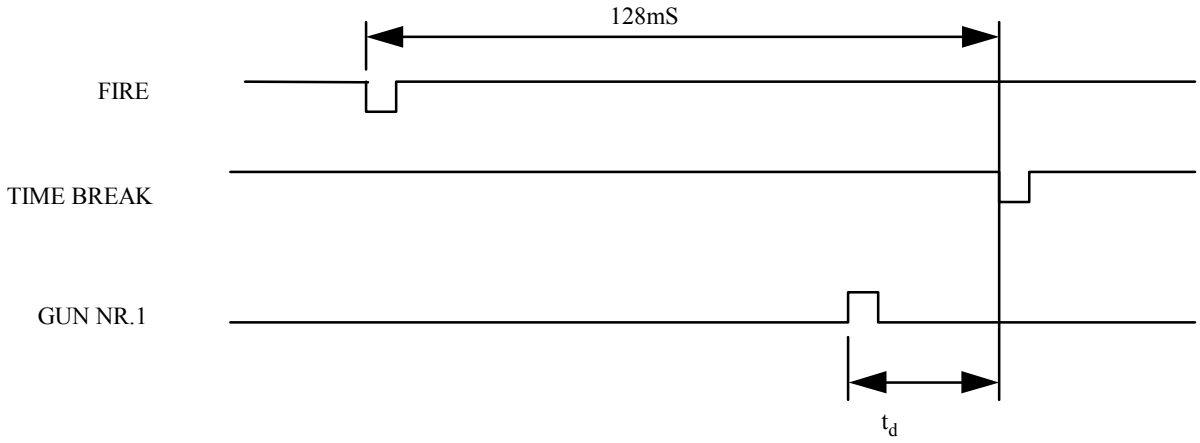
Programmet med Windows brukergrensesnitt ble utviklet på en annen PC. Vi såg på forskjellige programmeringsspråk, og bestemte oss for å benytte Visual Basic 5 siden vi hadde en del erfaringer med dette språket fra før. Innlesing av data skjer ved hjelp av interrupt. UiB kjøpte derfor inn et program som ruter et interrupt inn til programmet og kaller opp en bestemt interrupt funksjon. Dette programmet gjør det at blir mulig i Visual Basic å programmere funksjoner for å lese og skrive til I/O området. Når PC'en får et interrupt kan en starte innlesing av data. Når alle data er lest inn må det lages en programkode for å vise dataene på skjerm. Det benyttes her et program som heter Olectra 2D Chart.

3 Hovedkapitler

3.1 Systembeskrivelse

Systemet vårt består av hardware og software som skal brukes til overvåking av luftkanoner. Luftkanoner brukes for å generere trykkbølger som blir reflekterte i havbunnen. Refleksjonen av trykkbølgen fanges opp i hydrofoner. Denne informasjonen brukes til seismiske undersøkelser. Systemet skal integreres i en eksisterende enhet (se vedlegg 1) som brukes for å styre og kontrollere innretninger brukt til å utføre seismiske undersøkelser. "Gun controller" enheten mottar et FIRE signal, og den sender ut et strømsignal som går igjennom solenoiden (magnetventil) i luftkanonen, og avfyrer den. I vedlegg 1 er dette signalet også kalt FIRE. Dette signalet vil heretter bli omtalt som solenoidstrøm eller strømpuls for å unngå forveksling. Vi bruker FIRE-signalet til å starte samplingen (jfr figur 3.1).

Tidsdiagram**:**

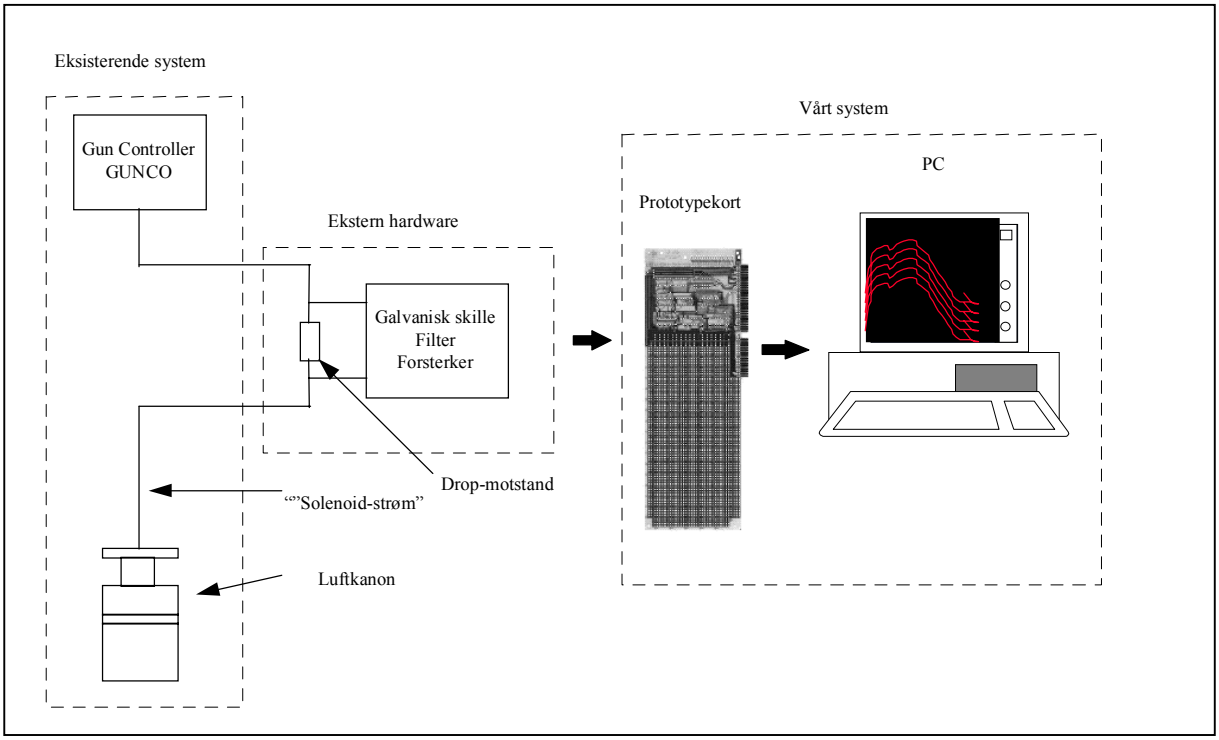


t_d = variabel tidsforsinkelse, avhengig av kanonens tilstand

Figur 3.1. Timingdiagram for eksisterende system. *(Kilde: Ole Meyer, IFJF UiB)*

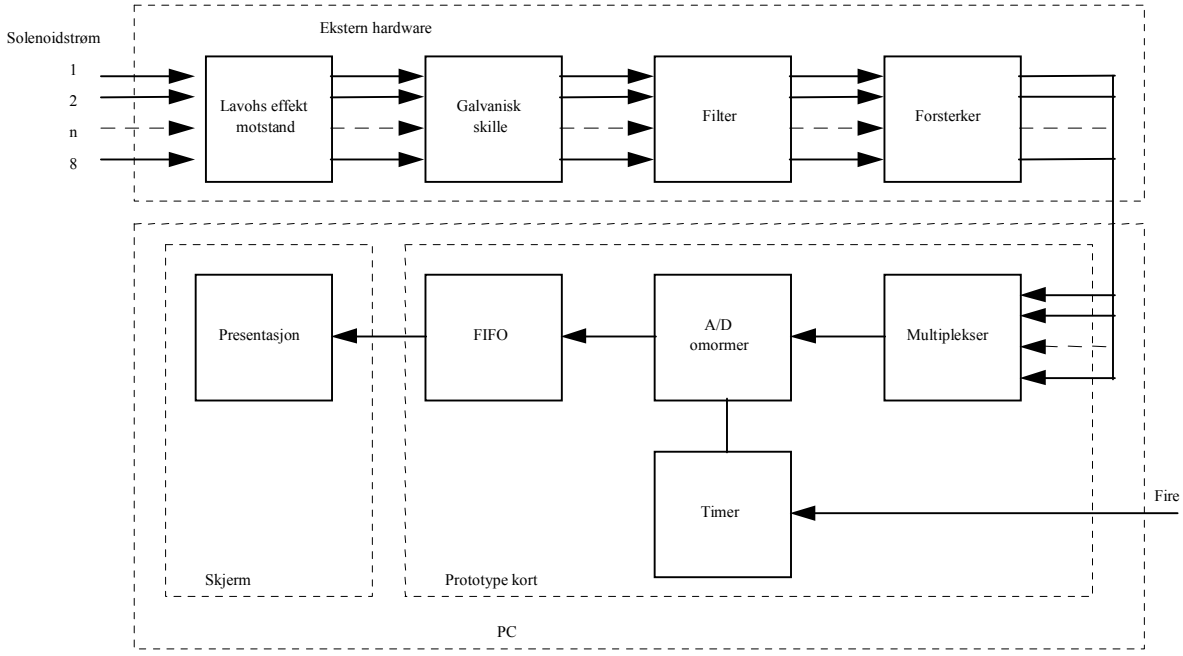
Ved å koble inn en lavohms drop-motstand kan dette signalet gjøres om til en spenning. Spenningen over motstanden A/D konverteres og leses inn i en PC. PC'en presenterer dataene grafisk i to dimensjoner og lagrer historien til de 4 forrige skuddene.

Systembeskrivelse



Figur 3.2 Systembeskrivelse

Blokkskjema av systemet



Figur 3.3 Blokkskjema av system

For å beskytte PC'en mot de høye spenningsnivåene på "fire" signalet (ca 75 V), og for å hindre støy i å forplante seg inn til PC'en er det nødvendig å bruke et galvanisk skille. Vi ble enige om at dette lages i en ekstern hardware som UiB tar seg av. I denne eksterne hardwaren inngår også filtrering og forsterkning. En analog optokobler brukes til å skille "fire" signalet galvanisk fra PC'en. En optokobler består av en lyssender (fotodiode) og en lysdetektor (foto transistor).

Hensikten med å bruke et antialiaseringsfilter er å få ned båndbredden på signalet slik at en unngår aliaseringsfeil. Disse feilene oppstår dersom en sampler et signal med frekvenskomponenter som er høyere enn halve samplinsfrekvensen $f_s/2$ også kalt nyquistfrekvensen. Dette filteret må komme før en sampler og digitaliserer signalet. For mer detaljer om antialiarerings problem refereres det til appendiks F

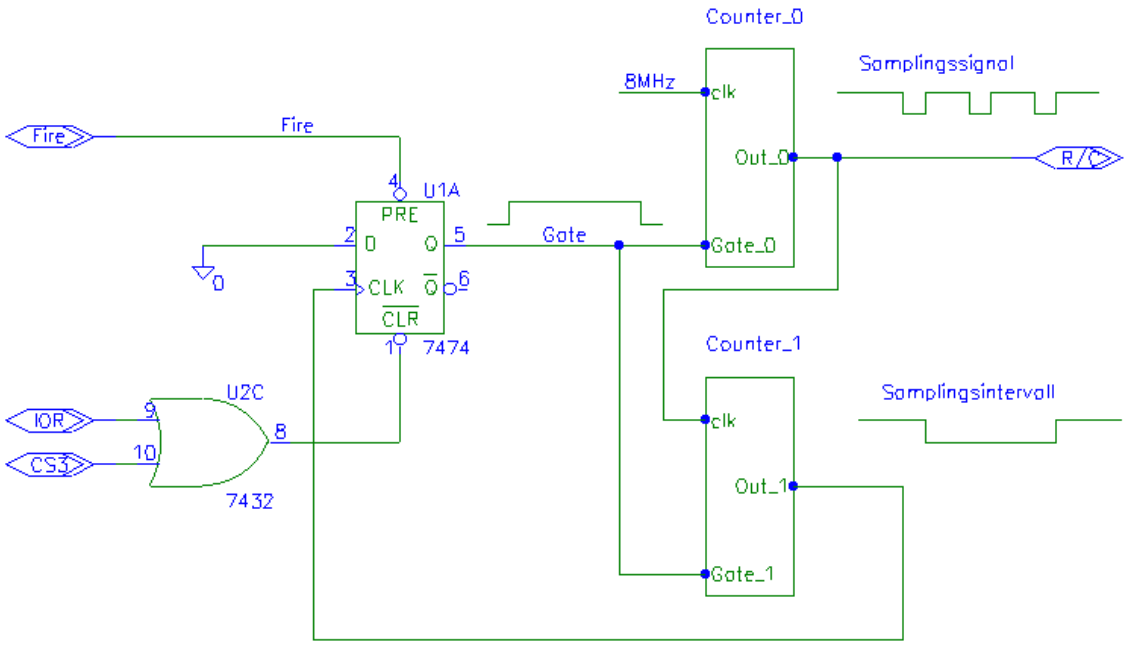
Forsterkeren har som funksjon å forsterke opp signalet før A/D – omforming slik at en kan utnytte A/D – omformerens oppløsning fullt ut. Dette er meget viktig med hensyn til nøyaktigheten på det gjengitte signalet. En A/D – omformer har en dårligere nøyaktighet på små signaler enn på høye. Dersom vi har en 12 bits A/D – omformer med fullskala utslag på 10 V ser en at oppløsningen blir:

$$\text{Oppløsning} = 10V/2^{12} = 2{,}44 \text{ mV /bit}$$

Et signal på 3,66 mV vil dermed ligge mellom det minst signifikante bit og det nest minste. Det vil si at dette enten må representere 2,44 mv eller 4,88 mv. I begge tilfeller blir den relative usikkerheten i signalet på ca 33 % . Dersom vi ser på et signal som ligger omtrent midt på A/D - omformerens arbeidsområde f.eks. 5001,22 mV, som vil ligge der det mest signifikante bit skifter fra 0 til 1. Her kan vi regne ut den relative usikkerheten til ca 0,00024 %.

For å kunne sample 8 kanoner samtidig med kun èn A/D - omformer bruker vi multipleksing. En multiplekser kan ses på som en roterende bryter med kontakter som slipper i gjennom signalet til èn og èn kanon etter tur. Fra multiplekseren går så signalet videre til A/D - omformeren. Denne sampler så en og en kanon. På grunn av hastigheten til A/D - omformeren virker det som om den sampler alle kanonene samtidig.

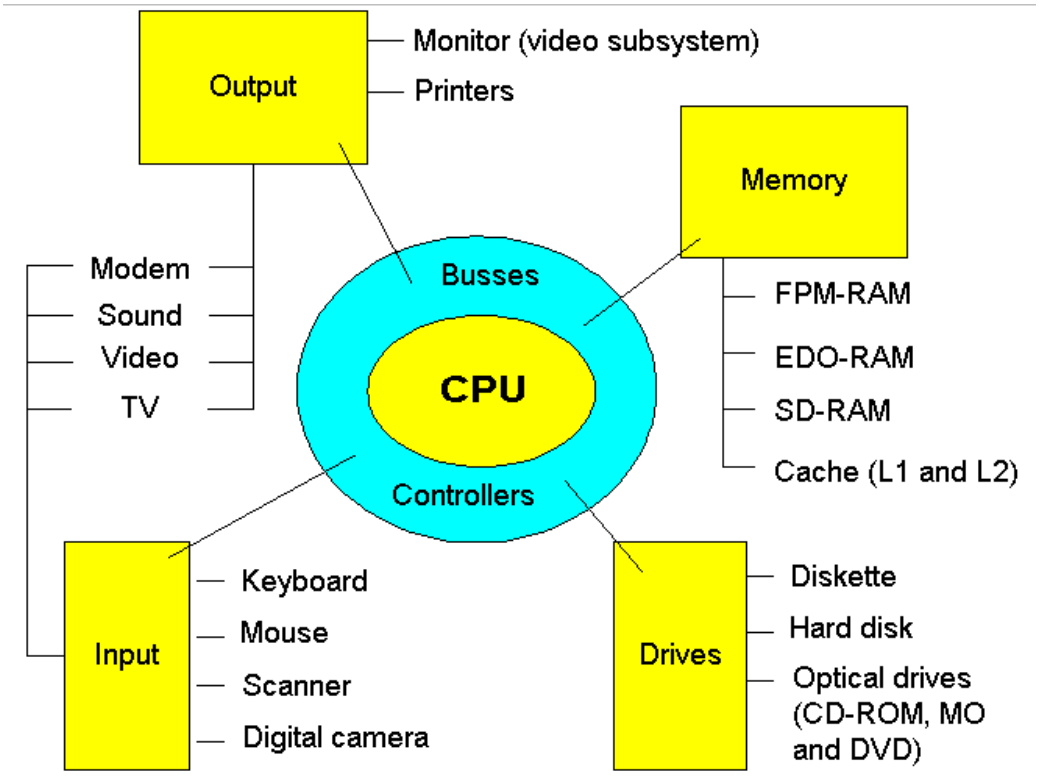
A/D-konverteringen starter når timeren blir enablet av FIRE-signalet (se figur 3.4). Timeren bestemmer samplingsfrekvens og samplingsintervall. Etter at timeren er programmert vil ”FIRE” signalet legge Gate høy. Tellerne blir dermed enablet, og Counter_0 som er programmert i modus 2 vil begynne å generere samplings signalet som går til A/D – omformeren. Counter_1 er programmert i modus_0, som er en hendelseteller. Ved å la utgangen til Counter_0 være klokken til Counter_1 vil utgangen på Counter_1 legge Gate lav, og dermed avslutte samplingen når den har talt ned det øskede antall samples. Dataene som blir lest inn lagres midlertidig i et buffer (FIFO), som gir PC'en tilstrekkelig tid til å starte innlesing. Når dataene er lest inn til PC'en blir de presentert på skjermen.



Figur 3.4. Kretstegning som viser oppkobling av timer 82C54.

3.2 Hardware

For å bruke prototypkortet må en ha en del grunnleggende kunnskaper om hvordan PC`en er oppbygd og hvordan den fungerer. PC'ens viktigste del er CPU (central prosessing unit), også kalt prosessoren. Dette er en chip som befinner seg på hovedkortet. CPU`ens oppgave er å styre dataflyten i PC`en, samt utføre operasjoner på dataene. En kan se på CPU`en som PC`ens "hjerne". Den kommuniserer med andre deler av maskinen gjennom kontrollere og busser. Disse finnes på hovedkortet. Hovedkortet inneholder en rekke viktige komponenter. Disse vil bli nærmere beskrevet, men først vil vi se på en modell som viser hvordan de forskjellige delene på hovedkortet er knyttet sammen.



Figur 3.5 Oversikt over PC oppbygning *(Kilde: MK-data)*

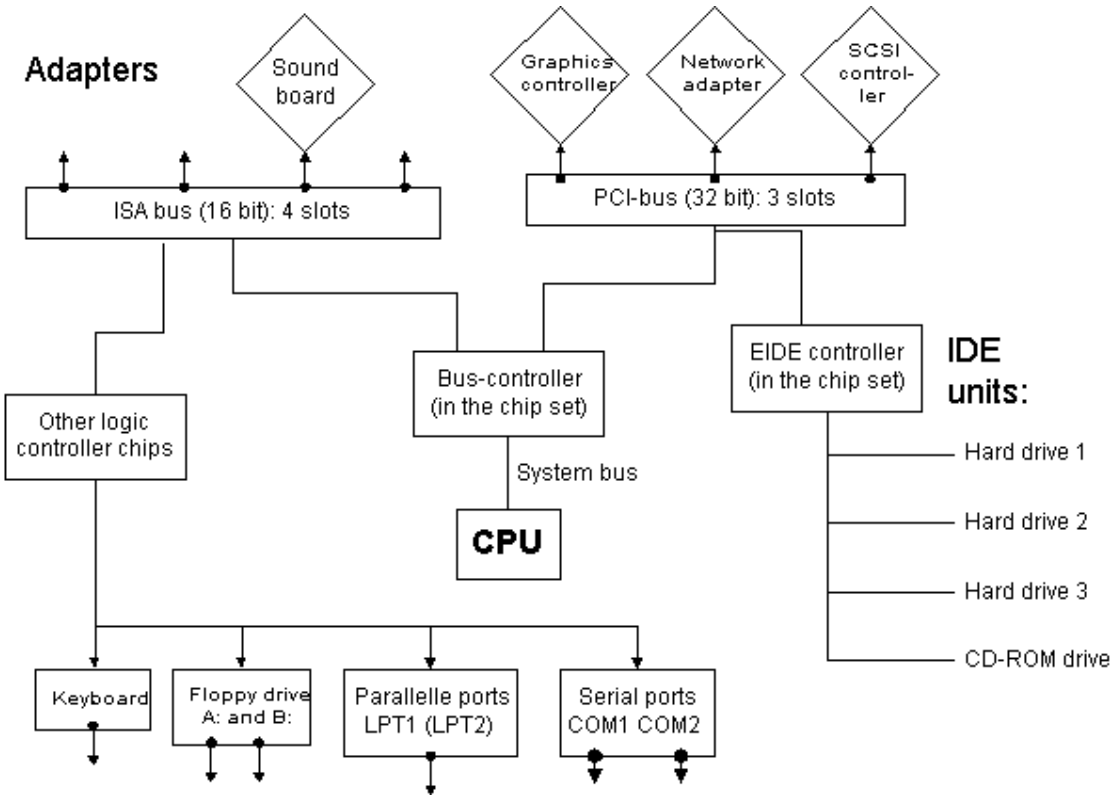
3.2.1 Hovedkort

Hele PC'en er bygget opp omkring hovedkortet. Dette kortet er viktig fordi det inneholder CPU med alle dens tilknytninger. Hovedkortet inneholder følgende deler:

•ROM-chips med BIOS og andre programmer
•CMOS, lagrer ”system setup” data
•CPU
•L2-cache
•Chipsets med I/O kontrollere
•RAM (Random Access Memory)
•Kort til å tilknytte tastatur og mus.
•Seriell og parallell porter
•Kontakter til harddisk, CD-ROM, og floppy disk.
•Slots for ekspansjonskort
•Jumpere for å justere spenning, systembusshastighet, klokke, etc.

3.2.2 Busser

CPU`en kommuniserer med prototypekortet er gjennom busser og kontrollere som vist på figur 3.6. Moderne PC`er har to hovedtyper av busser. Det er systembuss og I/O-buss.



Figur 3.6 Oppbygning av hovedkort *(Kilde: DK-data)*

3.2.2.1 System buss

Systembussen knytter sammen CPU med RAM (Random Access Memory), og på nyere PC'er også et buffer minne kalt Cache. Størrelsen på systembussen varierer fra PC til PC. De første PC'ene hadde 8 bits system buss, mens de mest moderne idag har 64 bits busser. Hastighetene på systembussene har også endret seg fra 4,77 Mhz i en 8088 maskin, til idag der en kan ha hastigheter på 100 Mhz.

3.2.2.2 I/O busser

Den høye hastigheten på systembussen medfører endel støyproblemer. Det er derfor nødvendig å redusere farten når en skal ut til ekspansjonskortene og annen hardware. Av denne grunn er det laget en eller flere I/O busser som kan operere med mindre hastigheter. Det er skjelden at ekspansjonskort opererer med hastigheter over 40 Mhz. I/O bussene knytter sammen CPU med alle innretninger untatt RAM. En PC kan ha forskjellige I/O busser. Her er de tre mest vanlige:

- ISA bussen, som er en eldre lavhastighetsbuss.
- PCI bussen, som er en ny høyhastighetsbuss.
- USB bussen (Universal Serial Bus), som er en ny lavhastighetsbuss.

ISA bussen:
ISA (Industry Standard Architecture) bussen har vært standard I/O buss. Den blir fortsatt brukt for å sikre kompabilitet med kort som er basert på denne bussen. ISA bussen er16 bits bred og har en maximum klokkefrekvens på 8 Mhz. Teoretisk er den i stand til å overføre med en hastighet på 8Mbps (Megabits/sekund), men i praksis er hastigheten under 2Mbps. ISA bussen brukes til 2 formål:

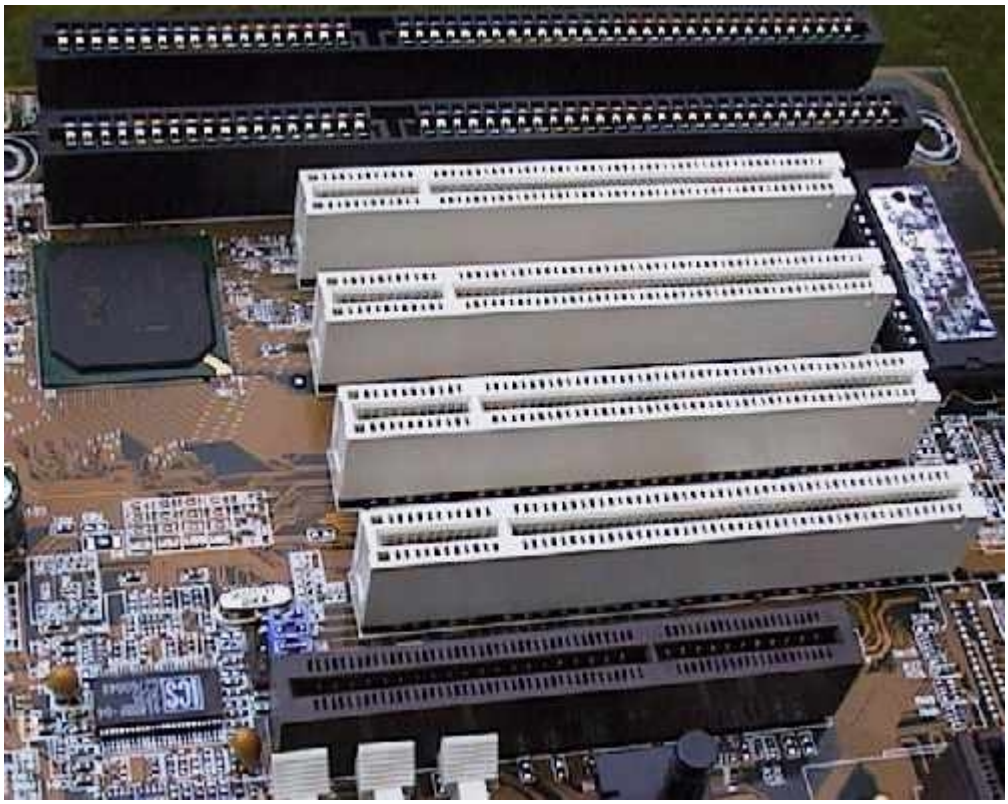
- Intern ISA buss, brukes til enkle porter slik som tastatur, seriell og paralell port
- Ekstern ekspansjons buss, kan knyttes til 16 bits ISA kontakter.

PCI bussen:
PCI (Peripheral Component Interconnect) er 90 åras høyhastighetsbuss. Den blir brukt i nesten alle PC'er idag. Bussen er 32 bit bred, men fungerer i praksis som en 64 bit buss. Dette gjøres ved å overføre 2*32 bit. Maksimum overføringshastighet er på 132 Mbps. Den kan brukes sammen med alle 32 bits og 64 bits processorer, og er kompatibel med ISA bussen, slik at den kan reagere på ISA buss signaler og lage de samme interruptene (IRQ).

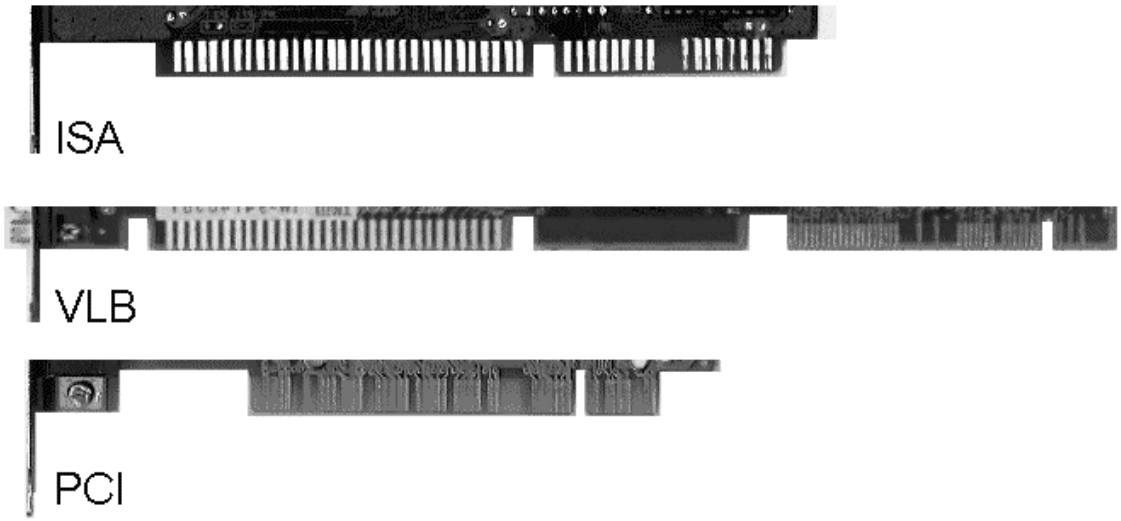
3.2.3 I/O enheter, ekspansjonskort og adaptere

3.2.3.1 Adaptere

Adapterene er kretskort, ofte kalt "expansion board" eller "expansion card". Disse kortene kan knytte sammen PC'en med perifere komponenter. Et eksempel på et slikt kort er prototypekortet. Kortet knytter sammen PC`en med det eksisterende systemet (se vedlegg 1). Dette blir kalt en åpen arkitektur. Adapterne har kontakter som gjør at de passer inn i "ekspansjons slot'ene" på hovedkortet. Siden et pentium hovedkort har to I/O busser (ISA og PCI) har det også to typer "ekspansjons slot'er". Figur 3.6 viser hvordan I/O slot'ene ser ut. De to øverste er ISA slots og de fire neste er PCI slots. På neste figur vises hvordan kontaktene på ekspansjonskortene ser ut.



Figur 3.7 I/O slotter *(Kilde: DK-data)*



Figur 3.8 Leppekontakter *(Kilde: DK-data)*

Prototypekortet er laget for 16 bits kommunikasjon, og er derfor beregnet for tilkobling til ISA-bussen. Dette skjer ved at kortet settes ned i ISA-kontakten som er avbildet på figur 3.7. Kortet kan nå komme i kontakt med CPU. For at CPU skal kunne kommunisere med prototypekortet, må dette tildeles et eget adresseområde. Dette området er på forhånd definert til å være fra 300H (hexadesimalt) til 31FH. Se appendiks B, tabell 2. Når CPU skal ha kontakt med en ekstern enhet f.eks

prototypkortet, benytter den seg av adresseringen. Den aktuelle I/O-enheten dekoder adressen for å sjekke om det er denne prosessoren vil kommunisere med.

3.2.4 IRQ

Når prototypekortet er satt ned i ISA-slotten, kommer det i kontakt med I/O bussen. Kortet kan nå sende og motta data. IRQ (Interrupt ReQuest) er konstruert for å kontrollere datatrafikken på I/O bussen. Det er to typer interrupt.

- Software interrupt, brukes for å kalle opp BIOS (Basic Input Output System) rutiner.
- Hardware interrupt, som beskrives nærmere under (se også appendiks E)

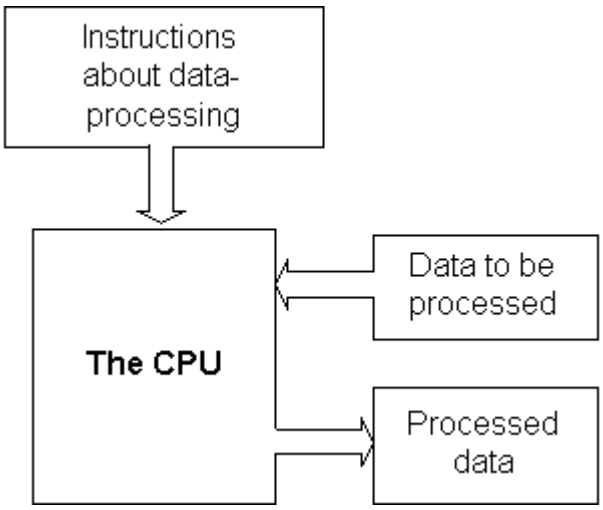
3.2.4.1 Hardware interrupt

Prototypekortet og andre innretningner på I/O bussen bruker interrupt til å signalisere at de ønsker å sende eller motta data. Dette interruptet detekteres av CPU, og den har en rutine for behandling av interrupt. En av fordelene med IRQ er at CPU kan jobbe med andre oppgaver mens kortet sender sine data. IRQ er en fysisk wire på bussen. Alle IRQ linjene finnes på alle I/O slot'ene. På denne måten spiller det ingen rolle hvilken slot en plasserer kortet i. PC'en har 16 IRQ'er (0-15), men 5 av dem er interne og kan ikke benyttes av I/O bussen. En ny innretning må gis et IRQ nummer som ikke er i bruk. Nummeret på IRQ bestemmer også prioriteringen av delen. Lavt nummer betyr høy prioritet. Vi har valgt IRQ 15.

3.2.5 CPU

CPU mottar minst to typer data.

- Instruksjoner som forteller hvordan den skal behandle andre data.
- Data, som må behandles i henhold til instruksjonene.



Figur 3.9 CPU`ens arbeidsmetoder. *(Kilde: DK-data)*

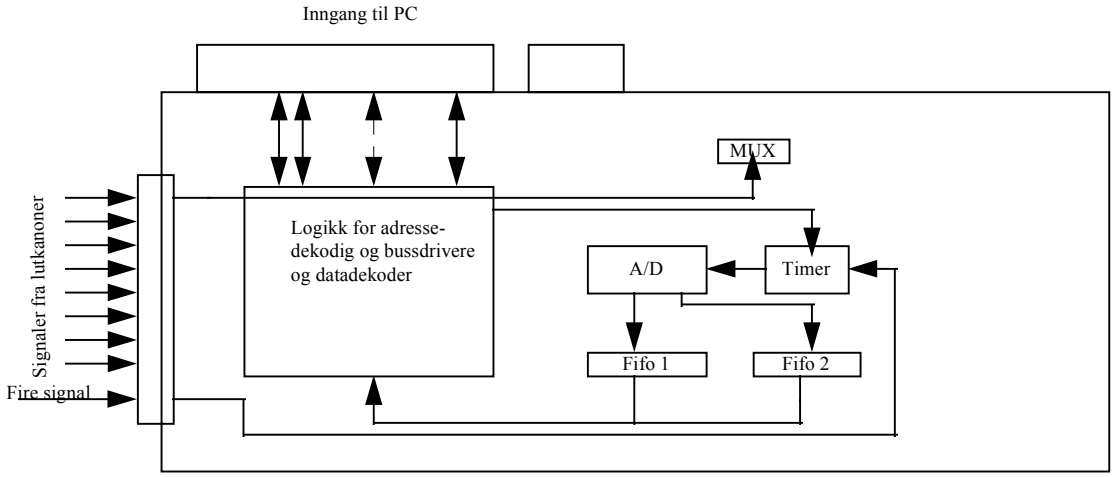
Instruksjonene kalles programkode. Hovedjobben til CPU består i å dekode instruksjonene og lokalisere data.

3.3 Oppbygging av hardware på prototypekort

Hardware delen av oppgaven er laget på et prototypekort beregnet på IBM AT kompatible system. Hensikten med et slikt kort er at en bruke PC'ens buss og adresserings metoder til å hente data fra hardware eller for å skrive til hardware. Vi skal som kjent A/D- konvertere spenningen som solenoidstrømmen danner over drop-motstanden. Hardware inneholder derfor følgene hovedelementer:

- PC kort med driverkretser for data og styringssignaler og adressedekodere
- Programmerbar timer (styrer samplingssignal og samplingsintervall samt cutoff- frekvens på antialiaseringsfilter).
- 12-bits A/D- konverter
- Multiplekser
- Antialiaseringsfilter
- Forsterker
- FIFO

Prinsippskisse av hardware:

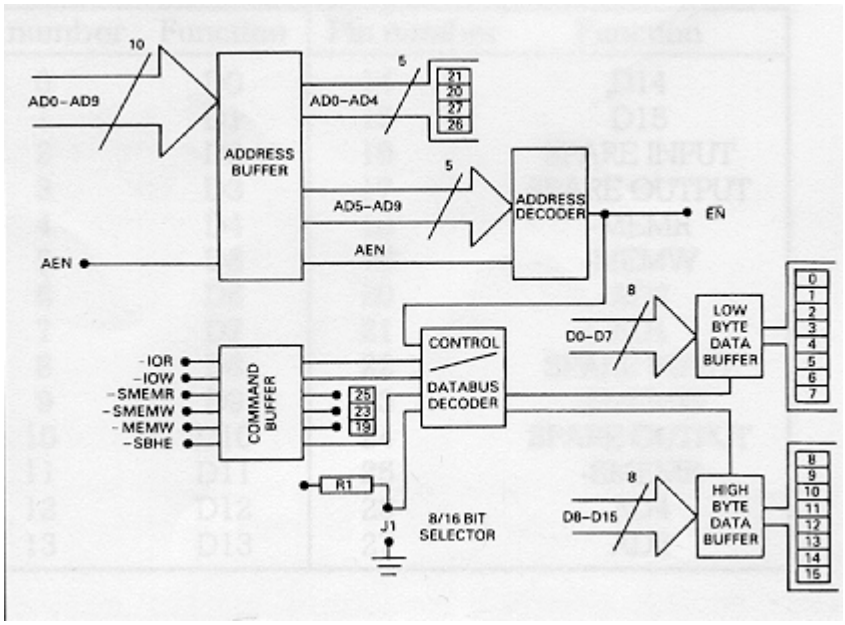


Figur 3.10. Blokkskjema over Prototypekort med hovedelementer.

3.3.1 Prototypekort

Prototypekortet er som tidligere nevnt et IBM AT kompatibelt kort. Dette betyr at det passer i de fleste PC'er på markedet og at det støtter 16-bit 's databuss. En kan også velge å bruke 8-bit's databuss. Dette gjøres ved å fjerne jumper J1 på figur 3.11. Siden vi trenger en A/D - omformer med med 12 bits oppløsning oppløsning, bruker vi 16 bits

databuss. Inngangene der prototypekortet kobles til PC'en kalles I/O porter (input/output) og er nærmere beskrevet i appendiks B. Måten PC'en kommuniserer med hardware på er å gi hardware forskjellige adresser. Disse adressene er angitt i appendiks B, tabell 2. En kan ut fra denne tabellen lese at adresse 300h til 31Fh er forbeholdt prototype kort. I softwaren kan det bestemmes hvilken adresse det skrives til eller leses data fra. Skriver man en I/O-kommando fra software, vil Read($\overline{IOR}$) eller Write($\overline{IOW}$) gå lavt i samme tidsrom som den aktuelle chip select linjen. Da vil data enten bli lest inn i PC'en eller bli skrevet til hardwaren. Figuren under viser blokkskjematisk hvordan driverkretsen for adressering og databussen er oppbygd. For nærmere detaljer refereres til vedlegg 2.



Figur 3.11 Blokkskjema over prototypkortets grunnutrustning (Kilde: Datablad)

$\overline{EN}$ signalet på blokkskjemaet gir ut logisk 0 når PC'en skriver til eller leser fra en adresse i adresseområde 300h til 31Fh. For å kunne skrive til forskjellige komponenter på prototypekortet er det nødvendig med en finere inndeling av adressene på kortet. Til dette formål benyttes en adressedekoder 74F138

Siden A0 blir brukt i forbindelse med enabling av data buffere og A5-A9 er brukt til å enable signalet $\overline{EN}$, har en tilgjengelig A1 til A4. For å programmere timeren må vi ha 2 adresselinjer, noe som blir beskrevet senere. For å finne ut hvordan en skal dekode adressene er det hensiktsmessig å sette opp en tabell.

Tabell 3.1 Oversikt over adresseområder:

Adresse	A4	A3	A2	A1
300h	0	0	0	0
302h	0	0	0	1
304h	0	0	1	0
306h	0	0	1	1
308h	0	1	0	0
30Ah	0	1	0	1
30Ch	0	1	1	0
30Eh	0	1	1	1
310h	1	0	0	0
312h	1	0	0	1
314h	1	0	1	0
316h	1	0	1	1
318h	1	1	0	0
31Ah	1	1	0	1
31Ch	1	1	1	0
31Eh	1	1	1	1

300h - 306h er chip select linje 1 og brukes til å adressere timeren. A2 og A1 brukes ved programmering

308h - 30Eh er chip select linje 2 og brukes til å adressere FIFO'ene.

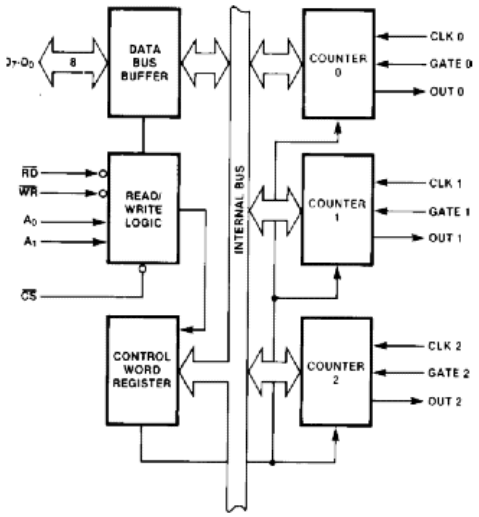
310h - 316h er chip select linje 3 og brukes til å resette alle komponentene.

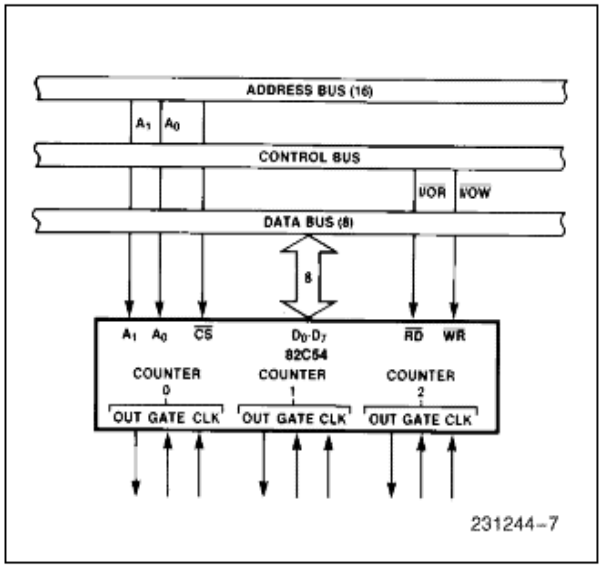
318h - 31Eh er chip select linje 4 og er blir brukt til å enable interrupt.

Siden A0 ikke er tilgjengelig ser en at adressen går i step på 2. A1 og A2 brukes til å programmere timeren. Dette medfører at vi har kun tilbake to adresselinjer som vi kan bruke til å dekode egne chip select signaler med. Vi får dermed fire adresseområder som vi kan spesifisere. Disse er angitt i tabellen over.

3.3.2 Timer 82C54

Denne timeren er i prinsippet den samme som benyttes i PC'en. Det er en programmerbar timer med 3 individuelle 16 bits tellere, og hver av tellerne kan operere i 6 forskjellige modi. En programmerer timeren ved at en først skriver et kontrollord som inneholder opplysninger om hvilken teller som skal programmeres og hvilken modus den skal operere i. Etterpå skriver man inn en tellerverdi enten som èn eller to byter etter hverandre. Vi har gitt timeren chip select linje 1, som betyr at den har adressene 300h - 306h. Hvilken av disse adressene en skal bruke avhenger av hvilken teller en skriver til og hva som skrives til den. For nærmere detaljer om de forskjellige modi, samt hvordan en programmerer timeren refereres til appendiks C. Figurene 3.12 og 3.13 viser hvordan timeren er oppbygd og hvordan den er knyttet til PC'en.



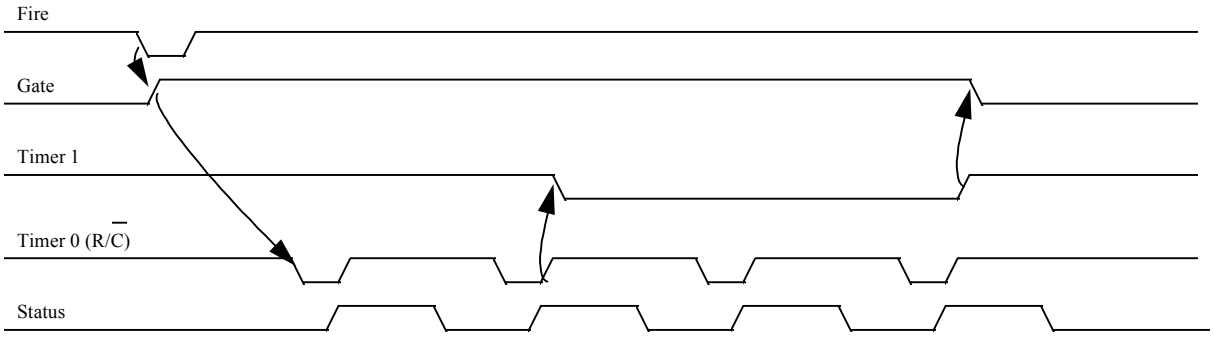


Figur 3.12. 82C54 Block diagram (Kilde: Datablad)

Figur 3.13. 82C54 System interface (Kilde: Datablad)

På figur 3.12 ser vi at hver timer har en utgang og to innganger ("Gate" og klokke). Timerne kan ha klokkefrekvenser inntil 10 Mhz. Vi bruker en 8MHz krystallosscillator som klokke på timer 0 og timer 2, mens timer 1 benytter utgangen på timer 0 som klokke. "Gate" signalet benyttes blant annet til å enable tellerne. Vi skal bruke timeren til å styre A/D - omformeren. Figur 3.14 viser opkoblingen til timeren, og timing – diagrammet under viser hvordan signalene blir styrt.

Timing – diagram



Figur 3.14. Timing diagram for styring av A/D-omforming

Timer 0 brukes til å generere samplingssignal. Vi programmerer den slik at den opererer i mode 2 som er en såkalt "Rate generator". Denne fungerer slik at vi skriver inn en tellerverdi. Timeren settes igang ved at "Gate" signalet settes høyt. Telleren teller ned til 0 og starter ny nedtelling fra den verdien som er lagt inn i teller-minnet. Dette skjer kontinuerlig så lenge "Gate" ligger høy. Hver gang telleren kommer til 0 går utgangen lav i en klokkepuls. Denne pulsen benyttes som samplingssignal.

Tellerverdien som legges inn bestemmer avstanden mellom hvert samplingssignal og sammenhengen er gitt ved følgende formel:

$$Tellerverdi = \frac{Klokkefrekvens}{Samplingsfrekvens}$$

Klokkefekvensen er 8MHz og max samplingsfrekvens for A/D - omformeren er 117 kHz slik at tellerverdien må være større enn 68 desimalt.

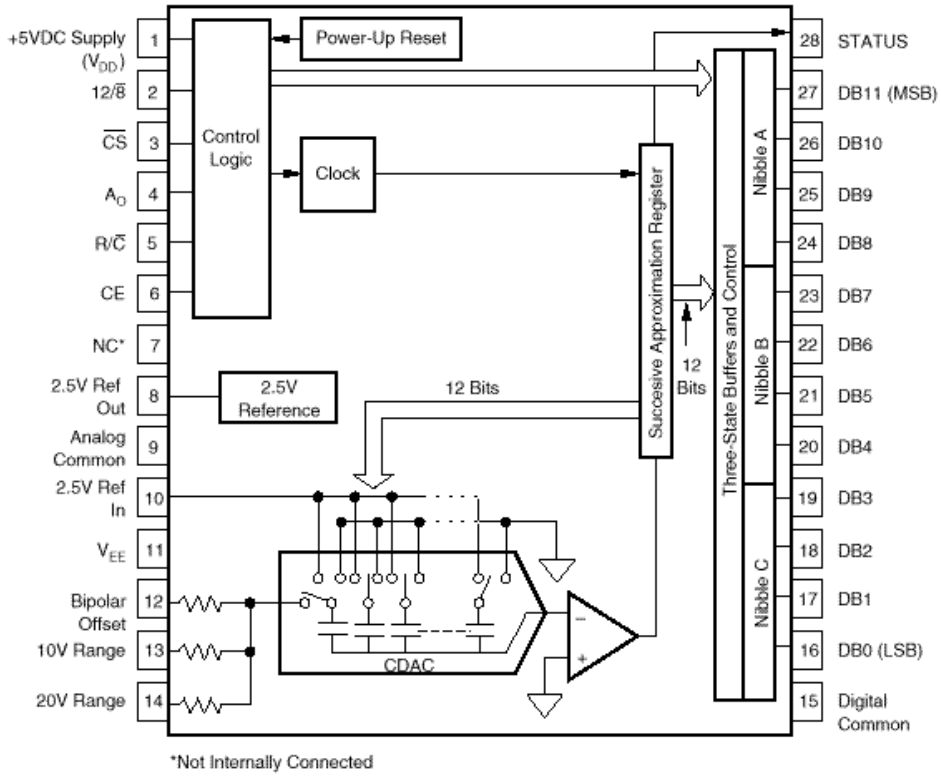
For å kunne bestemme hvor mange samplinger vi ønsker benyttes timer 1 i mode 0, som betyr at den fungerer som hendelses teller. Utgangen legges lavt ved initialisering. Dette skjer ved positiv flanke på "Gate". Utgangen ligger så lav til telleren er kommet til 0, og går så høy igjen. Ved å la utgangen på timer 0 være klokke på timer 1 vil tellerverdien som legges inn i timer 1 være lik antall samplinger. Ved hjelp av en D - vippe bruker en utgangen på teller 1 til å legge alle "Gate" signalene lav etter at en er ferdig med å sample det ønskede antall samplinger. Den samme D - vippen starter også samplingen ved at et "trigge" signal "setter" utgangen på D - vippen slik "Gate" signalet går høyt.

Timer 2 benyttes til å bestemme cut-off frekvensen til antialiaseringsfilteret. Filteret benytter et klokkesignal til å bestemme cut-off frekvensen i et forhold på 100:1. Nærmere beskrivelse av filteret finnes i appendiks F. Timeren programmeres i mode 3 som er "square wave generator". Utgangen initialiseres ved at "Gate" signalet går høyt. Utgangen vil ligge høy under første halvdel av nedtellingen og være lav under den siste halvdel. Telleren starter på nytt så lenge "Gate" ligge høy.

3.3.3 A/D - omformer

Når vi begynte på oppgaven var det meningen at vi skulle overvåke én eller kanskje to luftkanoner, og vi antok at en samplingshastighet på 20 kHz skulle være rikelig. Valget av A/D - omformer ble gjort på bakgrunn av dette, samt at den skulle ha en oppløsning på 12 bit. Videre måtte den være enkel å styre ved hjelp av timeren og den måtte ha styresignal som gjorde den enkel å koble til en FIFO. Vi valgte ADS774 som er en 12 bits "successive approximation" (se appendiks D) A/D -omformer med en maximum samplingsrate på 117 kHz.

Pin konfigurasjon og oppbyggning:



Figur 3.15 A/D-omformerens oppbygning *(Kilde: Datablad)*

Den har har flere styringsignaler ("Control Logic") som kan brukes til å styre samplingen. For detaljer om ADS774 vises det til Appendiks D.

Ut fra sannhetstabell (se appendiks D, figur 7) valgte vi å bruke en "stand alone" oppkobling der en kun bruker ett styresignal (samplingssignalet fra Timer 0) som kobles til pin 5 ($R/\overline{C}$). En fallende flanke på $R/\overline{C}$ starter konverteringen og STATUS (pin28) ligger høy under konverteringen. Disse to signalene er tilstrekkelig for å styre A/D - omformeren under 12 bits operasjon.

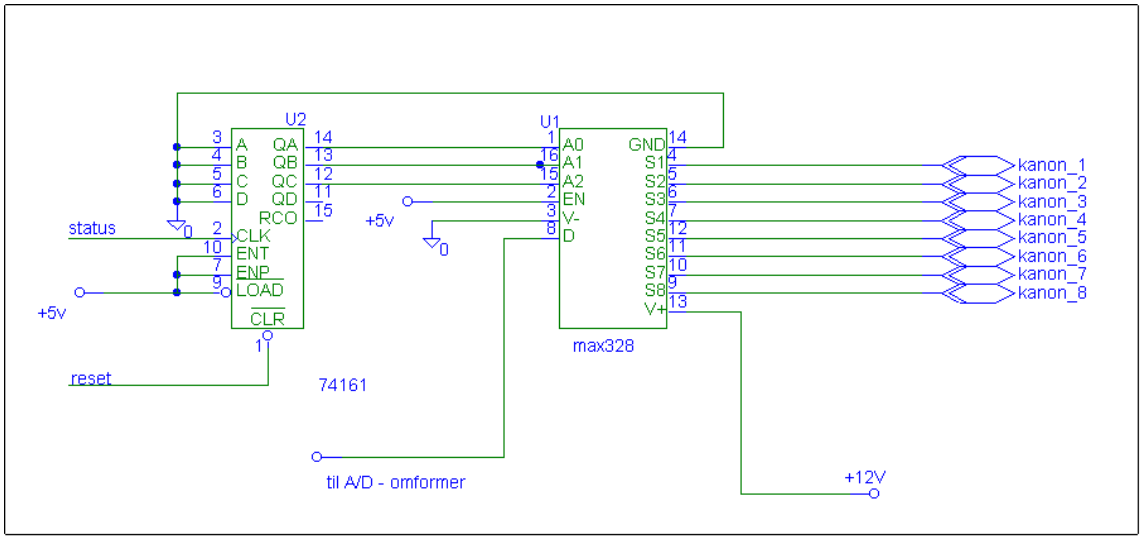
Etterhvert som vi kom igang med prosjektet kom det frem ønske fra oppdragsgiver om å undersøke muligheten for å kunne sample 8 luftkanoner ved hjelp av multipleksing. Siden vår A/D - omformer kan sample og få gjennom data med en hastighet på 117 kHz betyr dette at hver luftkanon kan sample med maksimum: 117 kHz / 8 = 14625 Hz . Dette vil sansynligvis være tilstrekkelig, men det kommer an på varigheten til det hakket vi skal ”jakte” på.

3.3.4 Multiplekser

For å kunne sample 8 luftkanoner på en gang med kun en A/D - omformer må en benytte en analog 8 kanals multiplekser i tilknytning til en teller. MAX328 passer bra til

dette formålet. Den er tilstrekkelig rask og er TTL (Transistor – Transistor Logic) kompatibel samtidig som den dekker ønsket spenningsnivå.

Oppkobling av multiplekser



Figur 3.16 Oppbygning av multiplekser

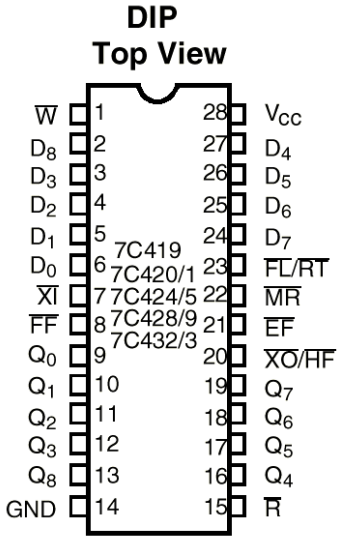
Inngangene A0 til A2 er de tre bit'ene som bestemmer hvilken linje på multiplekseren som slippes igjennom. A0 er minst signifikante bit. Disse signalene kommer fra telleren som er av type 74F161. Dette er en fire bits teller, men en bruker bare de tre minst signifikante bit'ene. Statussignalet brukes som klokke på telleren, og alle inngangene kobles til jord for å hindre at kretsen begynnner å oscillere, og på den måten forstyrrer virkemåten til telleren. Vi hadde først tenkt å la signalet gå gjennom et antialiasringsfilter og deretter til en forsterker før det går inn til A/D - omformeren. Etter at det ble bestemt at vi skulle bruke en multiplekser lot ikke dette seg gjøre lenger. Dette er fordi vi ikke har et kontinuerlig signal inn til filteret. Det samme problemet vil vi få med forsterkeren da denne ikke er rask nok til å følge de raske forandringene til multiplekserutgangen. Av denne grunn ble det bestemt at forsterker og filter ble flyttet til den eksterne hardwaren. Opplysninger om forsterker og filter finnes i appendiks F.

3.3.5 FIFO

FIFO er en minnekrets. Den blir brukt som et ”buffer” som fungerer på den måten at det som leses først inn i FIFO også leses først ut igjen. Hensikten med dette bufferet er å mellomlagre data. Dette er nødvendig for å kunne overføre asynkront. Den gir også PC`en tid til å starte innlesning, slik at en ikke mister de første dataene. De fleste FIFO'er har 9 bit's datalengde og variabel dybde. Da vi har 12 bit's datalengde, og dessuten ønsker å lagre informasjon om hvilken kanon som er samplet (3 bit), må vi koble to FIFO'er i paralell. Da vi bestemte dybden på FIFO'en var det meningen at vi skulle sample en eller to strømpulser. Lengden på samplingssignalet var ca. 35 ms og med 20 kHz samplingsfrekens kan en rekne ut antall samplinger slik:

$$\text{Antall samplinger} = 2 * \frac{20 * 10^{3}}{\frac{1}{35 * 10^{-3}}} = 1400$$

Vi trengte derfor en fifo som kunne lagre 2k*15 bit. FIFO`en har en bredde på 9 bit, noe som gjorde at vi måtte paralellkoble to FIFO`er. Til å laste data inn i FIFO'ene benytter vi STATUS signalet til A/D - omformeren (pin 28). Dette signalet går høyt når konverteringen starter og går lavt når den er ferdig. Ut fra timing diagram til ADS774 finner en at dataene på utgangen er gyldige i minst 75 ns før status går lav (jfr. t_{HS} i appendiks D tabell 3). FIFO leser inn data ved stigende flanke på $\overline{W}$ (pin 1). Dataene må være gyldige minst t_{SD} = 18 ns før positiv flanke (se appendiks D, tabell 4). Ved å invertere statussignalet fra A/D - omformeren kan en bruke dette til laste inn data.



Figur 3.17. Pinkonfigurasjon til FIFO *(Kilde: Datablad)*

Da det ble bestemt at vi skulle sample 8 luftkanoner ved hjelp av multipleksing var det ikke lenger mulig å lagre alle dataene i FIFO. Vi måtte derfor få PC'en til å starte innlesing før FIFO'en ble full. På FIFO finnes det tre flagg (hardware utganger) som indikerer hvor mange data den har motatt. Disse flaggene er:

- $\overline{EF}$, er lavt når FIFO er tom og går høyt etter første innlesing.
- $\overline{HF}$, er høyt når fifo er under halvfull og går lavt når den passerer halvfull.
- $\overline{FF}$, går lavt når fifo blir full og er høy ellers.

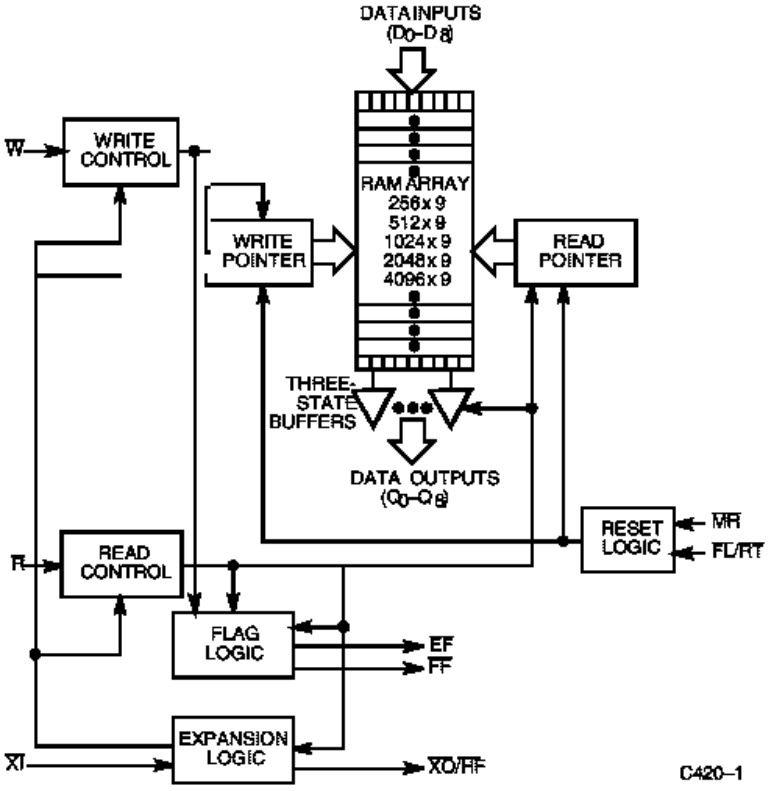
Vi fant ut at det var mest hensiktsmessig å benytte $\overline{HF}$ flagg til å generere et interrupt til PC'en. Inlesingen i PC måtte starte før FIFO ble full. Dette medførte at vi hadde 1024*8,5 µs = 8,7 ms tilgjengelig fra PC'en fikk et interrupt til innlesingen måtte være begynt. Når innlesingen starter vet vi at FIFO inneholder minst 1024 data. Ved å la

software lese inn 1024 data ved hvert interrupt unngår en å lese fra tom FIFO. For at det skal bli generert et nytt interrupt, samt hindre at FIFO blir full må også programmet lese inn data raskere enn 8,5 µs.

Når PC`en begynner å lese fra FIFO`en, vil den etter en stund lese FIFO`en under halvfull. Dersom det skulle komme et nytt sett med data før PC`en leste inn neste gang, ville dette generere et nytt interrupt. For å unngå dette er det nødvendig å bruke en D-vippe. Ved å la halvfullflagget gå inn på "Preset" , adresse 318H gå inn på klokken og la D-inngangen ligge lav kan vi styre dette ved hjelp av software.

Fifo'en har både en "write pointer" og en "read pointer" dette gjør at en kan lese og skrive til fifoen asynkront. For å sikre at fifoen er tom ved ved starten av hver serie av innlesninger har fifoene en $\overline{MR}$ (master reset) som tømmer fifo'en og nullstiller pekerne (figur 3.18). Ved oppstart av programmet og etter hver serie av samplinger resetter vi FIFO'ene ved hjelp av $\overline{MR}$. Denne resetter også resten av komponentene i kretsen. På denne måten sikrer man at luftkanon nummer 1 er alltid den første man leser inn.

Blokk skjema



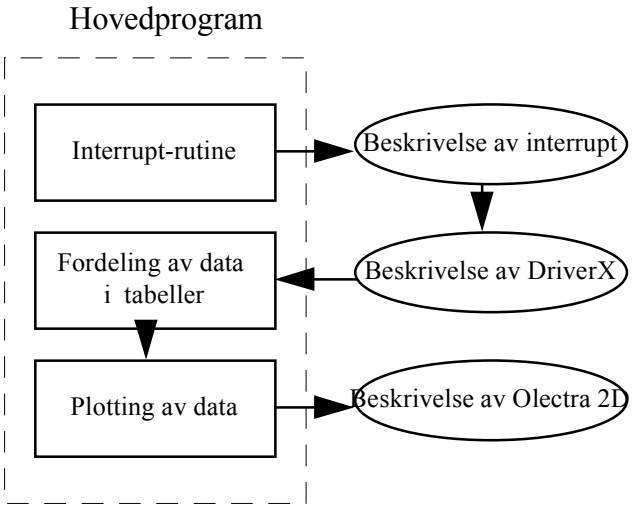
Figur 3.18 Fifoens oppbygning *(Kilde: Datablad)*

3.4 Software

3.4.1 Programoppbygning

Fordi programmutviklingen begynte før hardware var ferdig var det vanskelig å vite nøyaktig hvordan programmet skulle bli. Sluttproduktet ble derfor bygget opp av mange funksjoner. En funksjon er bygget opp slik at man ”sender” en del parametre til den. Når funksjonen er ferdig med å gjøre den oppgaven den er laget for, vil den returnere en verdi/parameter.Programmet er bygget opp slik at man først må programmere timer i hardwaren.

Timeren må ha informasjon om hvor mange ganger vi skal sample. Den må i tillegg ha informasjon om lavpassfilter-frekvensen og samplingshastigheten. Instillingene til timeren kan endres på ”Instillinger”-menyen og programmeringen gjøres ved å velge «Start innlesning» på «Fil»-menyen. Innlesningen blir startet ved et kall på «On_Interrupt()». For å få en oversikt, vil beskrivelsen av programmet bli som vist på figuren under.



Figur 3.19 Programmets oppbygning.

Når vi programmerer timeren, vil kretsen være klargjort til å kunne starte sampling. Denne vil begynne så snart kretsen har mottatt et FIRE-signal. Når FIFO er halvfull, vil denne gi et interrupt til PC. Etter at PC har fått et interrupt, vil den kalle opp funksjonen som styrer innlesningen. Når alle data er lest inn i programmet, kan behandlingen av data starte. Funksjonen som blir kallet ved interruptet er vist under.

Programkode for interrupt-funksjon:

```
Private Sub DxControl1_OnInterrupt()
Dim temp As Boolean
Dim i As Integer

Timer1.Enabled = False                    'Stopp softwaretimer
For i = 1 To 1024
```

```
   buffer(Antall_ganger) = vbInpw(Fifo) And 4095   'Les inn 1024 byte og mask vekk
   Antall_ganger = Antall_ganger + 1               '13.-16. bit
Next
Timer1.Enabled = True                              'Start softwaretimer
If (ant_sampl - Antall_ganger) > 1024 Then         'Forventes  flere interrupt ->
   Call vbOutw(Interrupt, 0)                       'Klargjør for nytt interrupt
End If
End Sub
```

3.4.2 DriverX- program

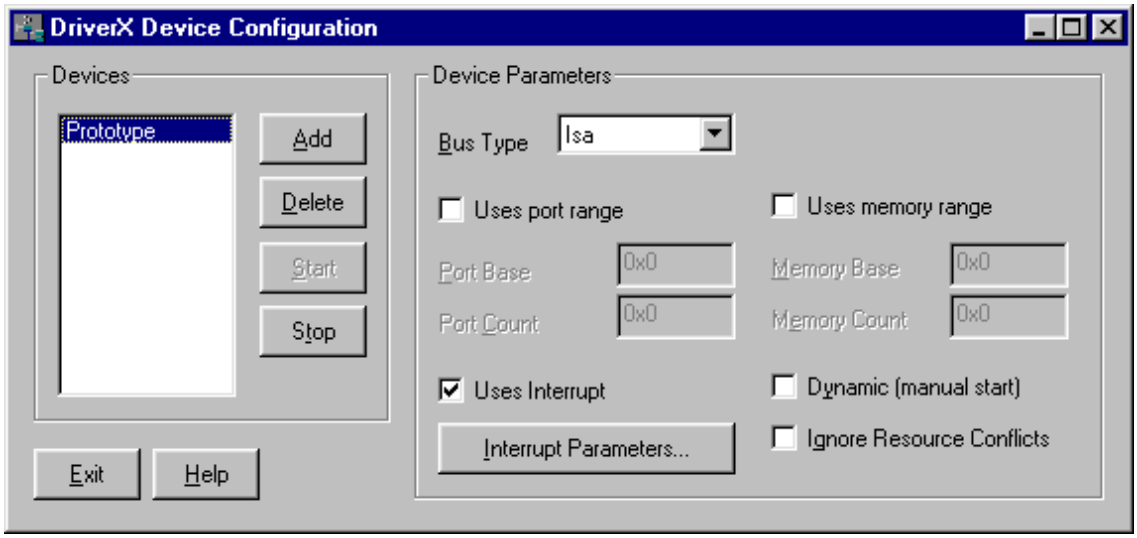
Når hardware vil la applikasjonene vite at de kan starte innlesningen, vil de signalisere dette ved hjelp av et interrupt. For å kunne oppfatte et interrupt i Windows95, er det behov for en spesiell filtype som kalles VxD`er (Virtual device Driver). Disse har en svært avansert oppbygning, noe som gjør dem vanskelig å lage.

På internett fant vi en link til hjemmesiden til Tetradyne. Dette er et firma fra USA, som spesialiserer seg på softwareutvikling. De hadde laget en ActiveX-kontroll som fungerer som en VxD. En ActiveX-kontroll et lite program som har en del metoder og egenskaper, og som kan implementeres i visuell programmering.

Tetradyne har utviklet en ActiveX kontroll som gjør det mulig å rute et interrupt fra hardware til VMM (Virtual Maschine Manager). Dette er en registreringsenhet som holder oversikt over alle VxD`er som er startet. VxD`er er programmer som gjør at flere applikasjoner kan dele på prosessortiden. Når det ikke er bruk for dem lenger, vil VMM også avslutte dem. Når VMM har fått informasjon om interruptet, vil den kalle opp funksjonen som er knyttet til dette interruptet. Mer utførlige forklaringer finnes i appendiks E. Halvfullflagget kobles til IRQ-linje 15. Funksjonen ”On_Interrupt” blir dermed aktivert når FIFO`en er halvfull. Ved hjelp av et medfølgende program kunne man tilpasse ActiveX kontrollen slik at den fungerte som ønsket.

På grunn av ActiveX-kontrollens lite egnede funksjoner til innlesning fra I/O området, ble det brukt en freeware DLL (Dynamic Link Library)-fil til innlesning av data. En forutsetning for at ikke FIFO`en skal fylles opp er at maskinen leser raskere enn A/D-konverteren klarer å lage nye data. A/D konverteren har en konverteringstid på 8.5µS. Ved hjelp av DLL-filen klarer vi å lese data på 3µS, mens vi med Tetradyne`s DriverX program bruker over 30µS på å lese inn et enkelt sett med data. Reultatet er derfor at vi bare bruker interruptet til å starte innlesningen fra FIFO`en, mens vi bruker DLL-filen til innlesning av data.

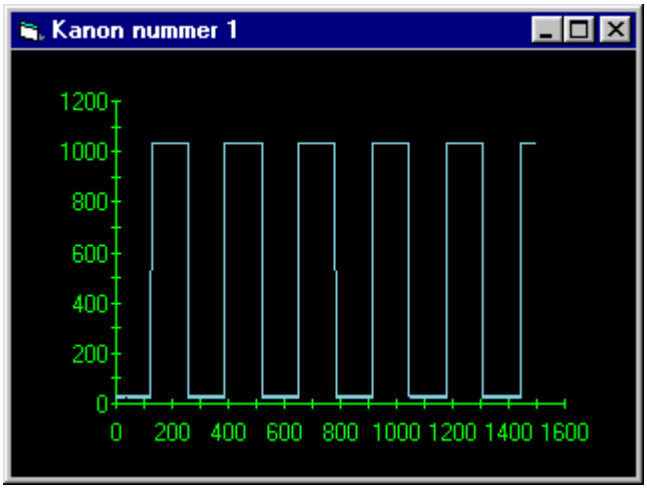
For å kunne bruke ActiveX kontrollen er det nødvendig å kalle opp en «ConnectDevice» funksjon. Man må i tillegg kalle en «EnableIsr» funksjon for å enable interruptet. Les appendiks E for mer informasjon.



Figur 3.20 Instillingene til ActiveX controllen.

3.4.3 Olectra 2D-chart

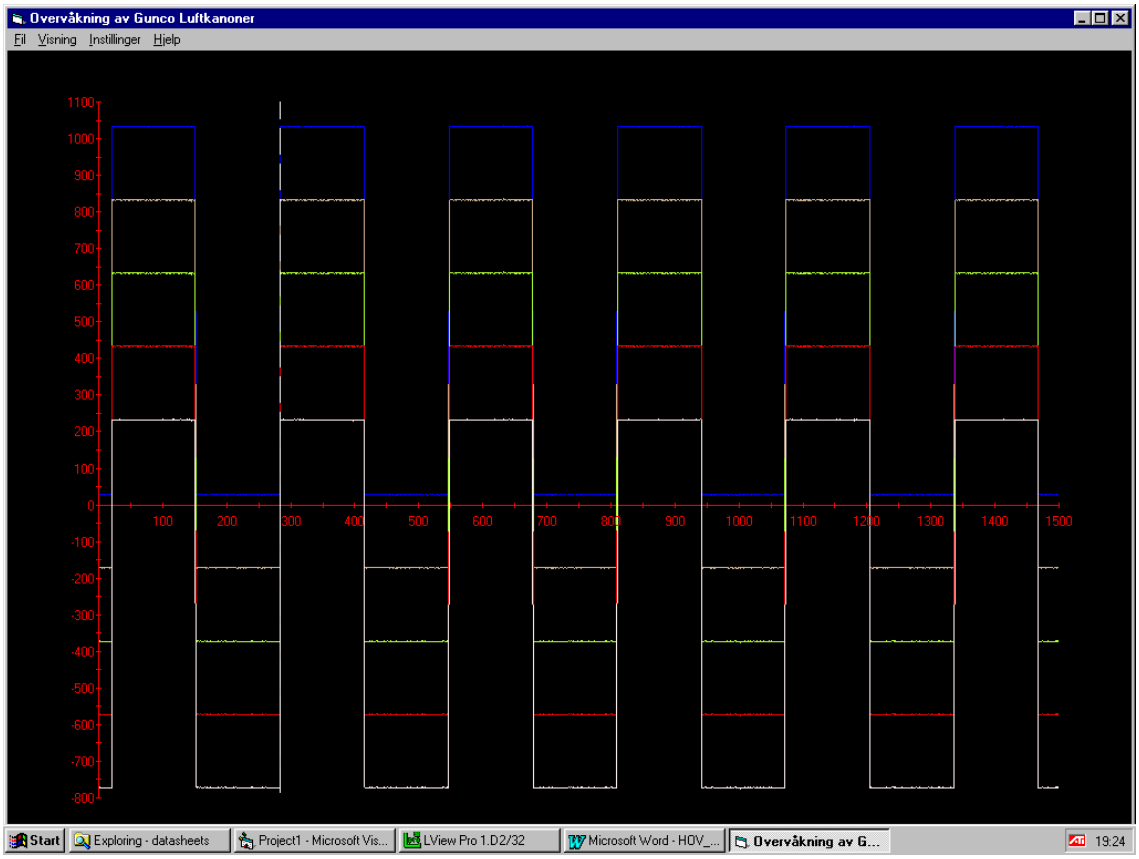
Olectra 2D plot er et program designet for visning av data i diagrammer. Vi bruker dette programmet til å fremstille de dataene vi leser inn i et linjediagram. For å gjøre visningen lettest mulig, leste vi dataene inn i et «array» (en matrise). Vi kan da ved hjelp av et enkelt funksjonskall kopiere dataene inn i diagrammet. Disse dataene blir da presentert i syv forskjellige diagram, ett for hver kanon. Figur 3.21 viser presentasjonen av et firkantsignal på 70Hz, sendt inn på en av inngangene. Dette signalet er samplet med en hastighet på 14kHz.



Figur 3.21 Sampling av 70 Hz firkantsignal.

I et 2D diagram kan man vise et stort antall «serier» med data. Dette gjøres ved å lage en todimensjonal matrise. Lengden på matrisen er «Antall samplinger», og bredden er «Antall serier». Historien til kanonene vises ved å kopierer de nyeste dataene inn som serie 1, og de eldste dataene som serie 5. (Forutsatt at antall serier er 5). Hver gang

gamle data blir kopiert over i en nye serie, reduserer vi y-verdien med 200. I tillegg til dette vises hver serie med forskjellig farge. Et eksempel er et signal på 70Hz samplet med en hastighet på 10kHz (figur 3.22)



Figur 3.22 Visning av gamle data.

Tabellen under er et utdrag av en 2 dimensjonal matrise. Her kommer dataene etterhverandre horisontalt, mens gamle skudd kommer i raden under. Tallverdien i hver enkelt celle bestemmer høydeutslaget til grafen.

Tabell 3.2: Utdrag av 2 dimensjonal matrise.

Serie 1	3200	3100	2900	2600	2300
Serie 2	3000	2900	2700	2400	2100
Serie 3	2800	2700	2500	2200	1900
Serie 4	2600	2500	2300	2000	1700
Serie 5	2400	2300	2100	1800	1500

3.4.4 Utdypning av programkode

Når det kommer et interrupt fra hardwaren, vil «On_Interrupt» funksjonen bli kalt opp. Siden interruptet er et signal på at FIFO`en er halvfull kan vi med en gang lese inn 1024 byte med data. Innlesningen til maskinen går imidlertid så raskt i forhold til A/D konverteringen (ca 7 ganger raskere), at vi ikke vil lese flere samplinger enn dette. For å kunne lese inn de neste dataene venter PC`en på et nytt interrupt.

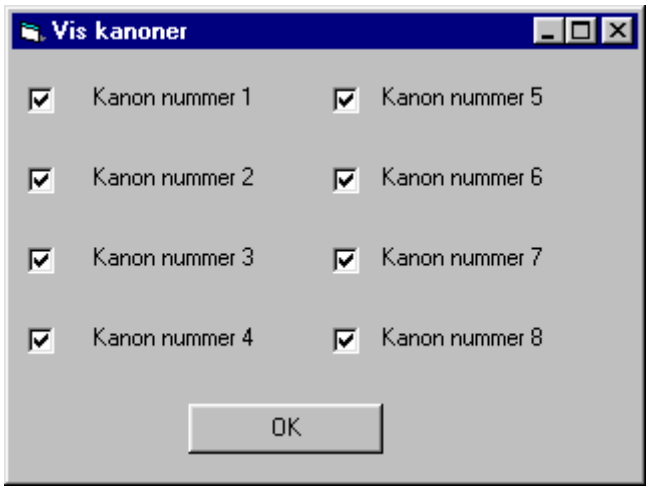
Hver innlesning OG`er vi med 4095. Dette er den desimale verdien til 000011111111111, som masker vekk alt bortsett fra de 12 bit`ene med informasjon. Bit nummer 13-15 inneholder informasjon om hvilken kanon som er samplet. Ved uttestingen av programmet fant vi ut at det var unødvendig å teste på disse bit`ene i programkoden for å bestemme hvilken kanon vi sampler. Under har vi vist hvordan ”On_Intrrupt”-funksjonen er bygget opp.

For å vite når samplingen er ferdig, har vi tatt i bruk en software-timer. Hver gang vi får et interrupt, vil timeren bli enablet etter innlesning. I oppgaven har vi programmert timeren til å telle i 100mS. Dersom denne tiden går ut, vil en bestemt funksjon ”Timer1_timer()” bli utført.

Denne funksjonen leser inn resten av dataene fra FIFO, og vi vil begynne visning av data ved hjelp av funksjonen ”Innlesning()”. Hvis det derimot kommer et nytt interrupt før tiden løper ut, vil timeren bli disablet. Det går derfor ikke an å sample seinere enn at 1024 samplinger blir samplet på 100mS.
Frekvens pr kanon * antall kanoner må derfor være større enn 1/100mS. (Fordi vi bruker tid på innlesningen, ville det i realiteten fungert med en lavere frekvens.)

Hvis det foretaes sampling av 8 kanoner, vil hvert 8. tall i innlesningstabellen(Buffer()) inneholde informasjon om èn kanon. Tilsvarende vil det være for 4 kanoner. Timerfunksjonen leser dataene til kanonene inn i respektive tabeller. Disse tabellene er globalt definert, og kan derfor brukes av alle funksjoner. Etter tallbehandlingen blir Innlesning() kalt opp. Her bestemmes hvilke data som skal vises, og hvilke funksjoner vi skal kalles. Vi har her tre funksjoner som kan kalles opp i flere kombinasjoner.

Plot2D(): Skriver tabellene inn i 8/4 grafer. Hver graf har sin egen form som kan velges bort eller vises. Programmet viser informasjon om alle kanonene default.

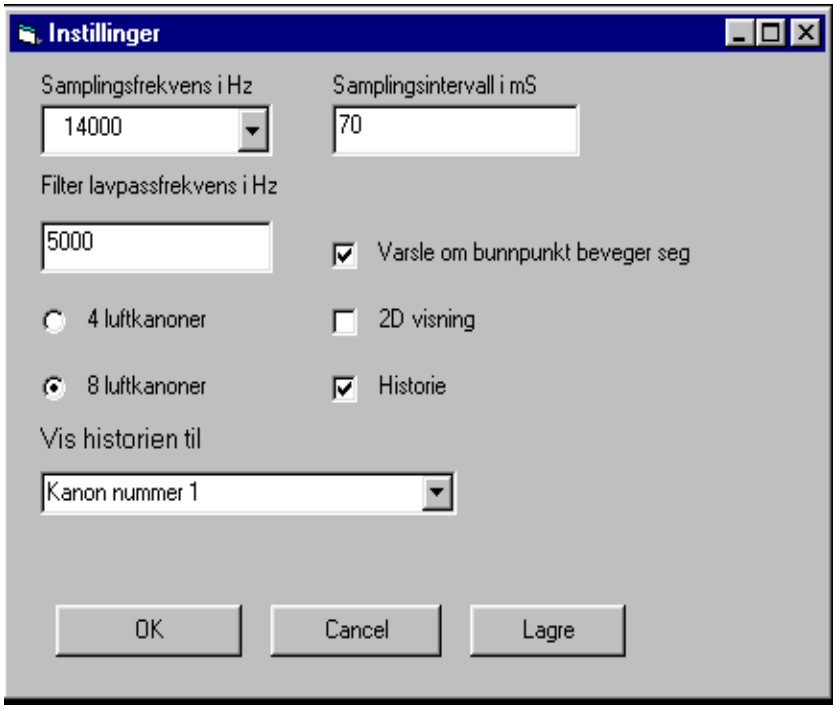


Figur 3.23 Kanoninnstillinger

Plot2D_til_3D(): Viser historien til hver kanon. Dette gjøres ved at 5 forskjellige grafer vises i samme koordinatsystem. Til å begynne med er det bare den ene grafen som er forskjellig fra 0. Når det neste skuddet går vil alle grafene bli kopiert fra grafen og til en matrise. Deretter vil alle tallene i rad 4 bli trukket fra verdien 200 og flyttet over til rad 5. Samme prosedyre vil bli fulgt for rad 3 til rad 4 osv. Tallene som opprinnelig er i rad 5 blir kastet. Vi få på denne måten "flyttet" gamle data nedover på grafen. (Dette kan vi gjøre fordi spenningsnivået over motstanden ikke er så interessant.)

Bunnpunkt(): Finner stigningstallet til grafen i alle punkter. Hvis stigningstallet er positivt, stiger grafen. Idet stigningstallet blir negativt er det muligheter for å finne et bunnpunkt. Når grafen så begynner å stige igjen, kan vi merke av et bunnpunkt. For at ikke støy på signalet skal virke inn på funksjonen, er det nødvendig med et stigningstall på mindre enn –5 for å detektere negativt stigningstall. Det samme gjelder for å detektere ny stigning.

For at vi skal kunne starte samplingen må timeren programmeres. Dette er derfor det første som gjøres etter at programmet er startet. Programmeringen er lagt til en funksjon som er kalt Timer_settings(). Her regnes det ut tellerverdier til alle timerene. For at samplingsfrekvenser, samplingsintervaller og antall kanoner skal blir registrert i programmet, må denne funksjonen kjøres. Det er med andre ord her alle instillingene blir satt.

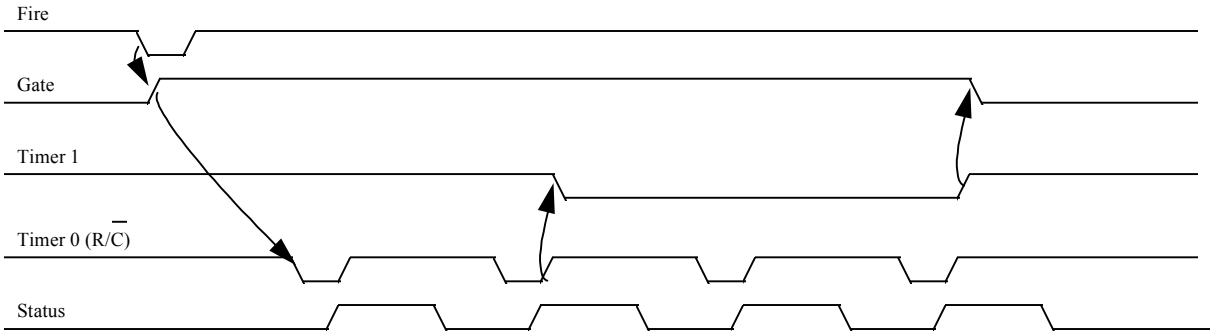


Figur 3.24 Samplings-imnstillinger

Alle globale deklarasjoner og DLL-filer blir deklarert i Module1. Dette gjør at de kan brukes av alle funksjoner i hele prosjektet.

4 Resultater og analysering

Ved uttesting av hardware benyttet vi en logikkanalysator (HP1651B). Denne gjør det mulig å lese raske endringer i TTL-logikk, ved å lagre signalene og vise dem på en monitor. Ved å feste prober til de aktuelle plasser i hardware er det mulig å skaffe seg informasjon om signaler som man ikke ville fanget opp med et oscilloscoop. En kan velge hvilket av signalene en vil trigge på. Alle signalendringer etter triggesignalet vil da bli lagret og vist. Ved uttesting av timeren ble logikkanalysatoren et svært viktig hjelpemiddel. Når vi programmerte timeren i TurboPascal fikk vi følgende resultat for utgangene på timeren:



Figur 4.1 Timingdiagram for innlesning av 3 samplinger.

Uttestingen viste at vi fikk 1 sampling mer enn programmert. Med det antall samplinger vi ønsker er ikke dette noe problem. Årsaken til dette problemet er at timeren som skal bestemme antall samplinger (timer1) drives av timeren som genererer samplingssignalet (timer0). Den første klokkepulsen laster inn tellerverdien og utgangen på timeren går ikke lav før neste klokkepuls.

For å laste dataene over i FIFO hadde vi to signaler som kunne brukes. Dette var $R/\overline{C}$ signalet (samplingssignalet) eller statussignalet. Ved å studere timingdiagrammer (appendiks D, figur 8 og figur 9) kom vi frem til at vi kunne benytte statussignalet. Når konverteringen av signalet er ferdig går statussignalet lavt. Skriving til FIFO skjer på stigende flanke. Det er derfor nødvendig å invertere statussignalet.

Til å styre multiplekseren trengte vi en 0-7 teller. For å finne ut hvilket siganal vi skulle bruke som klokke, samplet vi et visst antall samplinger og leste dem inn til PC`en. Deretter ble de skrevet ut på skjermen i 8 kolonner, en for hver kanal. Ved å bruke forskjellig spenning på inngangene til multiplekseren kunne vi se hvilken kanal vi samplet først. Vi kunne også se om telleren fungerte som den skulle. Det viste seg at det eneste signalet vi kunne bruke var det inverterte statussignalet. De andre signalalternativene førte til at vi leste kanal 2 først. Dette medfører at A/D-omformeren har minst 1µS før den skal starte neste sampling. Dette har vist seg å være tilstrekkelig.

Visning av data kan gjøres i 8 forskjellige vinduer som kommer opp på skjermen samtidig. Dette skjer like etter at innlesningen er ferdig, og i god tid før neste sampling. Dette alternativet til visning av data kan vise opptil 250mS ved full frekvens. Dette er rikelig i og med at hele sekvensen normalt tar 128mS. Figuren under viser sampling av en firkantpuls med frekvens på 100Hz, og en samplingsfrekvens på 14kHz.

Figur 4.2 Oversikt over 8 samplede kanaler.

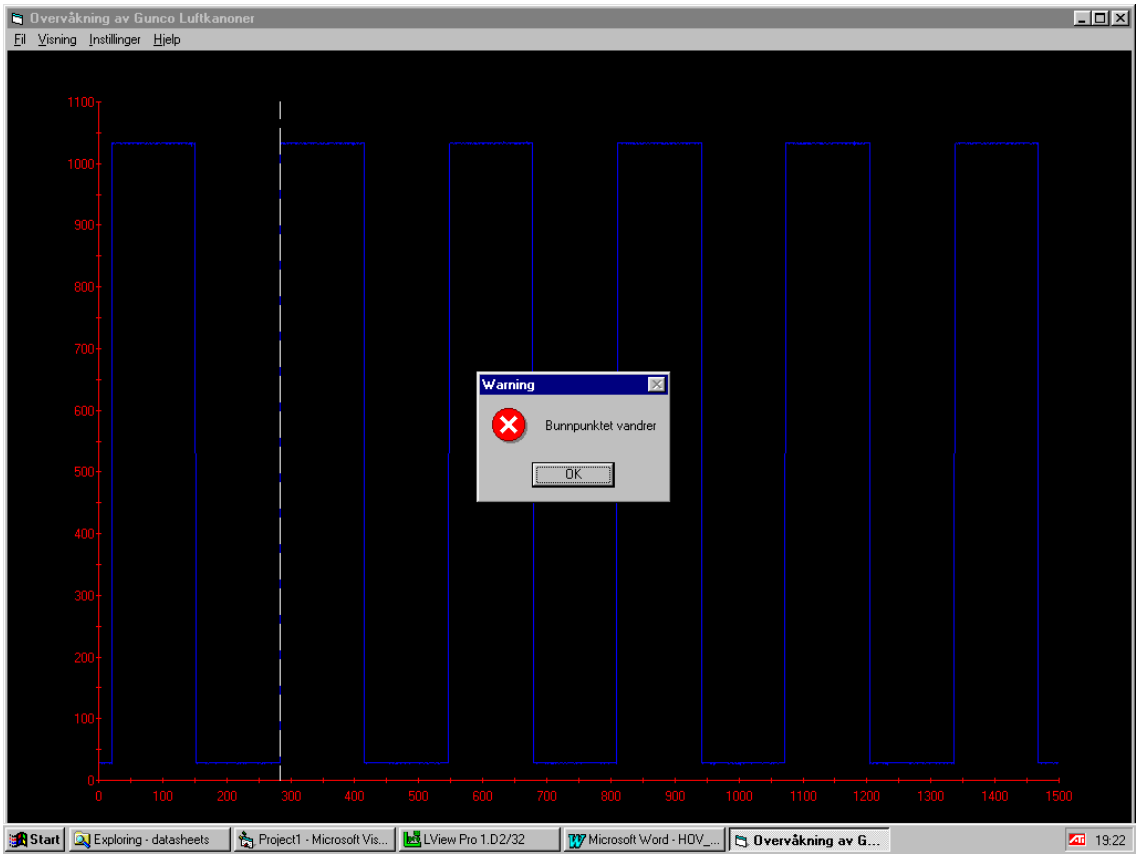
Når vi tester ut programmet i Visual Basic 5.0 er det mulig for programmereren å gå gjennom koden steg for steg. Det viste seg under uttesting at programmet har vanskeligheter for å kunne vise historiske data hvis det behandles et stort antall data. Ved å gjennomgå programmet steg for steg fant vi en begrensning. Etter at programmet har lest inn 4 strømpulser og vist disse historisk, vil ikke interruptet bli klargjort på nytt. Vi kan imidlertid se at programmet tilsynelatende skriver til den porten den skal. Ser vi på logikkanalysatoren, kan vi derimot ikke se at det skrives til porten.

I og med at denne programkoden er kjørt 3 ganger tidligere, samt at problemet ikke oppstår ved små datamengder, har vi ikke kapasitet/kunnskap nok til å finne løsningen på problemet. Vi tror at problemet kanskje skyldes for lite virtuelt minne.

For å være sikker på at ikke det er DLL-filen som svikter, har vi også testet ut Tetradyne`s funksjon for skriving til port. Resultatet var imidlertid det samme. En ting til som kan vitne om problemer med minnet, er at vi kan få utført små operasjoner som ikke krever minne. (F.eks. visning av ”MessageBox”). På den andre siden kan

funksjonen kjøres manuelt, dvs. man kan kopierer data bakover i matrisen. Funksjonen vil da kopiere inn data fra det forrige skuddet en gang til. Porten kan vi imidlertid ikke skrive manuelt til. Figur 3.22 er laget ved hjelp av manuell kopiering.

Vi har laget en funksjon som finner bunnpunktet til en kurve. Ved å lete gjennom tabellen til en valgt kanon kan denne funksjonen finne et bunnpunkt, og lage en vertikal stiplet linje gjennom punktet. Det kan også testes om bunnpunktet beveger seg mye, og det vises da en advarsel i form av en "Warning", som vist på figuren under.



Figur 4.3 Varsling om bunnpunkt som flytter seg.

5 Konklusjon

I begynnelsen av semesteret ble det fremlagt en målbeskrivelse. I denne rapporten kom det ikke direkte frem hvilken måte vi ønsket å løse oppgaven på. Det var imidlertid en klar formulering av problemet, og hva vi ønsket å oppnå. Fordi at vårt valg av løsningsmetode er noe annerledes enn som forespeiles i forrapporten, fokuserer vi på måloppnåelsen. Målet med oppgaven var:

- Å kunne vise forløpet til solenoidstrømmen på en PC monitor.
- Å kunne styre A/D-omformeren ved hjelp av PC.
- Å kunne fremstille historiske data i et diagram.
- Å finne bunnpunktet til strømpulsen.

Ser vi på det første målet, var dette hovedmålet med oppgaven. Ved hjelp av et prototypekort og et program har vi klart å vise hvordan strømpulsen til luftkanonen utarter seg. Vi har i tillegg til dette også gjort det mulig for brukeren å lese inn data fra 8 kanoner samtidig. Dette er en forbedring i forhold til oppgaveformuleringen. Til å styre A/D omformeren brukte vi en programmerbar timer. Denne kan når som helst omprogrammeres ved hjelp av softwaren. Dette gjør produktet fleksibelt, både med tanke på bruk og videre utvikling av systemet.

Innleste data blir behandlet og fremvist i et program utviklet i Visual Basic. Dette gir det et Windowsbasert grensesnitt, som er enkelt å bruke. Her har brukeren mulighet til å velge visning av 8 kanoner samtidig, samt visning av historien til en enkelt kanon.

Grunnen til at en ikke kan vise historien til alle kanonene samtidig, er at dette ville ta for lang tid. Ved oppgradering til en raskere maskin, sees det på som naturlig å utvide programmet til å vise historien til mer enn 1 valgfri kanon. Det er derfor laget en funksjon som gjør det mulig å oppdatere programmet på en svært enkel måte.
Det kan også synes aktuelt å ha en funksjon som finner bunnpunktet til alle kurvene, samt informerer brukeren om endringer.

Utvikling av en funksjon som kan detektere bunnpunktet til kurven, har vist seg å være vanskelig. Dette er fordi signalet som kommer inn inneholder en del støy. I og med at produktet ikke er prøvd ut i sitt rette element, er det ikke mulig å bedømme hvorvidt denne funksjonene oppfyller kravet.

Det som må gjøres for å kunne ta dette produktet i bruk er å lage ekstern hardware. Denne delen må inbefatte strøm/spenning-omforming ved hjelp av en lavohms-effektmotstand, filtrering og forsterkning. En mulig løsning på filtrering/forsterkning er skissert i appendiks F.

6 Litteraturliste

Bøker:

Andersen, Erling S.: Prosjektarbeid – en veiledning for studenter. NKI Forlaget, Bærum 1988

Norton, Peter.: Programmers Guide to the IBM PC. Microsoft Press, Washington 1995

Swan, Tom.: Mastering Turbo Pascal 5.5. Hayden Books, Indiana USA 1991

Oney, Walter.: Systems Programming for Windows95. Microsoft Press, Washington 1996

Gurewich, Nathan & Ori.: Visual Basic 5 in 21 days. Sams Publishing, Indiana USA 1997

Haugland, Kristen.: Innsamling av seismiske data. UiB, Bergen 1981

Datablader:

Advanced Micro Devices:	**CMOS FIFO Memory Product**
Burr-Brown:	**IC Data Book 1994**
Maxim:	**Databok Volume II 1993**

Internett adresser:

Cast inc.	http://www.cast-inc.com/cores/c8259a/c8259a.htm
DK – data	http://www.mkdata.dk/danish/start.htm
Intel	http://134.134.214.1/design/periphrl/datashts/231244.htm
The PC Guide	http://www.pcguide.com/ref/mbsys/res/irq/func.htm
McGill University	http://www.ee.mcgill.ca/~rudolph/imdac/isa.txt

7 Forkortelser

ActiveX kontroll:	Dette er et program som kan implementeres i alle visuelle programmeringsspråk til å utføre spesielle oppgaver. Programmet inneholder en del metoder og funksjoner.
BIOS (Basic Input Output System):	Dette er et program som er lagret i ROM (Read Only Memory). Programmet inneholder blant annet informasjon om hvordan PC`en skal kommunisere med hardwaren f.eks mus, tastatur, harddisk etc.
CPU (Central Prosessing Unit):	Dette er prosessoren. Denne styrer dataflyten på hovedkortet. Den tar også mot instruksjoner fra programmer og utfører alle beregninger.
DLL-fil (Dynamic Link Library):	Dette er navn på en fil som er felles for alle programmer som kjøres på en PC. En DLL inneholder programkode for en spesiell funksjon og kan benyttes av et hvilket som helst applikasjon.
DMA (Direct Memory Access):	Gir hardware mulighet til å kommunisere direkte med minnet uten å ta i bruk CPU`en.
FIFO (First In First Out) :	Minnekrets som lar de data som lagres først være de første som leses ut igjen.
IBM-AT:	Utvidelsen fra den opprinnlige 8 bits standarden (IBM-XT) til 16 bits dataoverføring til kortene (IBM-AT).
I/O (Input/Output)-busser:	Busser som overfører data mellom prosessoren (CPU) og ekspansjonskort som er tilknyttet ovedkortet.
IRQ (Interrupt ReQuest):	Interruptlinjer som kan brukes av hardware til å få CPU`ens oppmerksomhet. Det er totalt 16 IRQ-linjer. Disse har alle ulik prioritet.

ISA (Industry Standard Architecture) – bussen:	16 bits databuss som brukes av IBM-AT kompatible kort. Maksmum hastighet 8 Mbps.
PCI (Periferal Component Interconnect) – bussen:	En 32 bits databuss med maksimums overføringshastighet på 132Mbps.
PIC (Programmable Interrupt – Controller):	Inneholder tabeller som sier om et interrupt er gyldig. Programmeres av BIOS under oppstarten.
TTL (Transistor-Transistor Logic):	Kretser med to logiske nivåer 1 og 0. Disse nivåene er på henholdsvis 5 og 0 volt.
VxD (Virtual device (”x”) Driver):	Et program som blir kallet etter at PC`en har mottatt et interrupt. Det finnes en VxD for hver interruptlinje.
VMM (Virtual Machine Manager):	Dette er en VxD som holder kontroll over alle de andre VxD`ene. Den sørger for å opprette de VxD`ene som trengs, og terminerer de som ikke trengs mer.

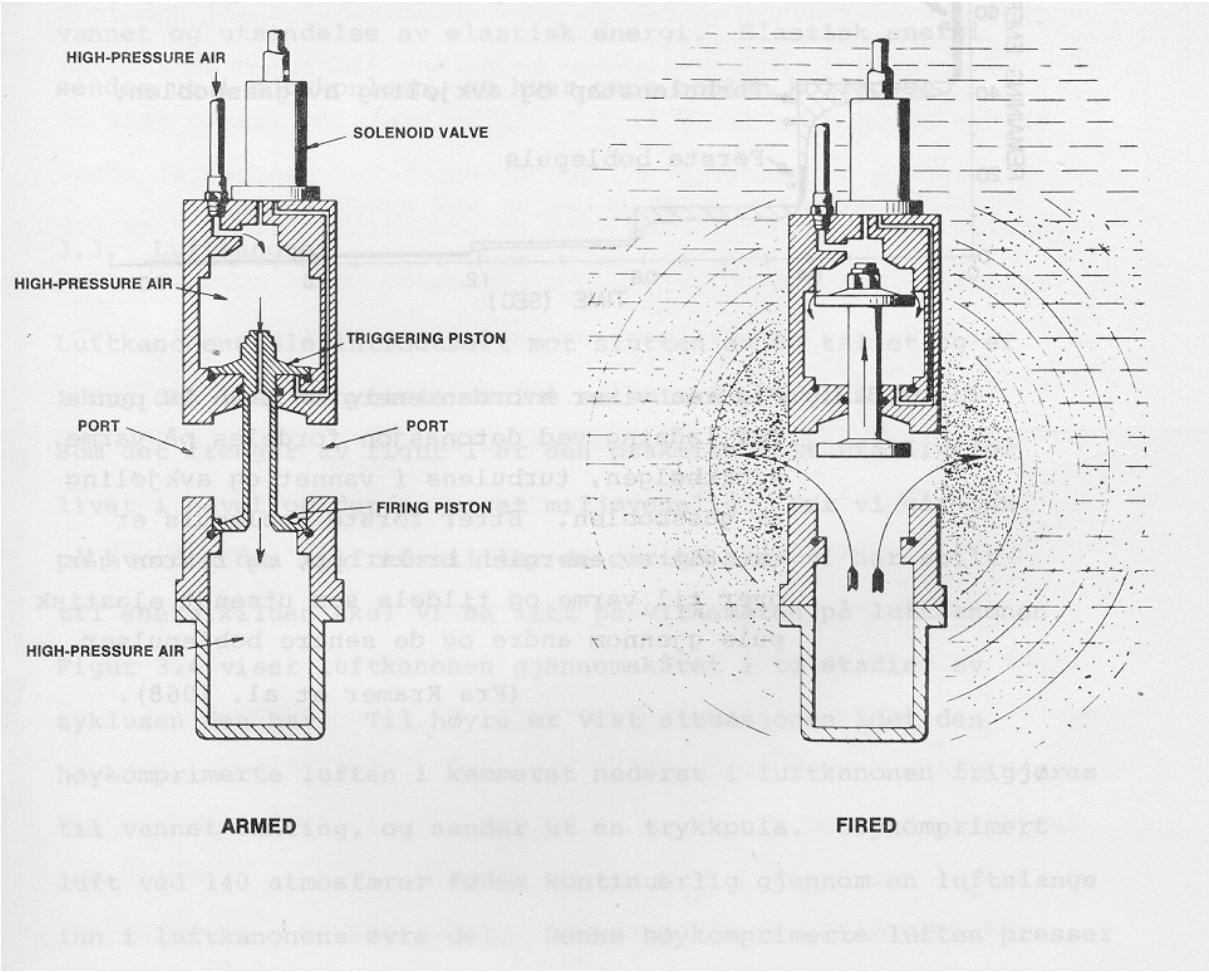
8 Stikkordregister

APPENDIKS

Appendiks A

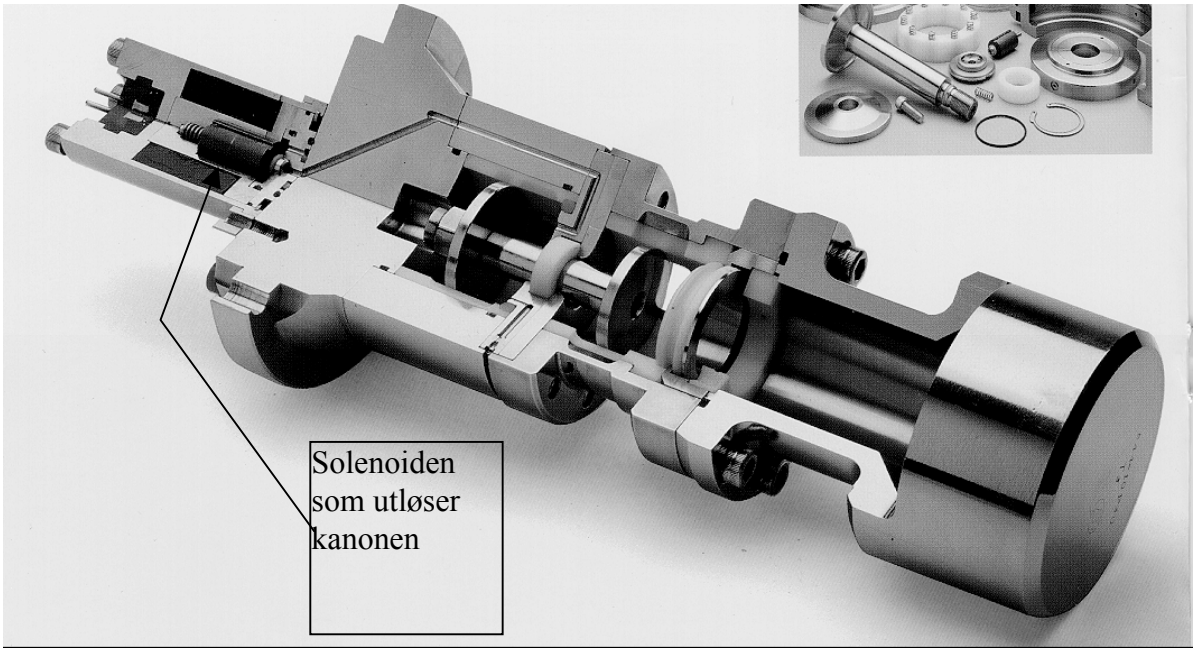
Luftkanon

Luftkanoner brukes til å lage trykkbølger i vann. Disse trykkbølgene brukes til å undersøke geologiske formasjoner i berggrunnen ved hjelp av refleksjonsseismikk. Kanonene lades opp ved at luft pumpes inn i dem under svært høyt trykk (ca 140 bar). Når luftkanonen avfyres frigjøres dette trykket, noe som kan sammenlignes med en eksplosjon. Trykkbølgen forplanter seg nedover mot havbunnen og den blir så reflektert av de ulike lagene i berggrunnen. Disse refleksjonene detekteres av hydrofoner, som er trykk sensorer. Det er disse signalene som er rådata for de seismiske undersøkelsene. Figuren under viser en slik kanon før og etter avfyring.



Figur 1 Avfyring av luftkanon *(Kilde: Innsamling av seismiske data. Haugland)*

På figur 2 vises også huset som inneholder solenoiden (kalt ”solenoid valve”). Dette er en magnetventil som åpnes ved et «fire» signal som kommer fra «gun controller» (se vedlegg 1). Når denne åpnes vil trykkendringen i øvre og nedre kammer føre til at stempelet beveger seg oppover. Kanonene som skal overvåkes blir laget av Bolt Technology Corporation. Under vises en figur av en slik kanon som er gjennomskåret, og en ser her tydelig solenoiden.



Figur 2 Gjennomskjært luftkanon *(Kilde: Bolt Technology Corporation)*

Appendiks B

Tabell 1: Beskrivelse av signallinjer i PC AT - buss system.

Signal	I/O	Beskrivelse
OSC	O	14.31818 Mhz oscillator
CLK	O	System clock 1/3 av oscilatorfrekvensen
RESET DRV	O	Dette signalet benyttes til å resette eller initialisere systemlogikk ved oppstart.
A0 - A19	O	Adressebit 0 til 19. Disse linjene blir brukt til å adresserememory og I/O deler innen systemet
D0 - D15	I/O	Databit 0 til 15. Disse linjene forsyner databuss bit 0 til 15 for prosessor, memory og I/O deler.
ALE	O	Adress Lach Enable. Denne linjen blir forsynt av Bus Controller'en og brukes av PC kortene til å for å låse gyldige adresser fra prosessoren.
$\overline{\text{I/O CHCK}}$	I	I/O kanal skjekk. Denne linjen forsyner prosessoren med paritets informasjon på memory eller deler på I/O kanalen.
I/O CH RDY	I	I/O channel ready. Denne linjen er normalt høy (ready), men kan trekkes lav (not ready) av en memory eller I/O del for å forlenge I/O eller memory syklusen. Dette tillater tregere deler å knytte seg på I/O kanalen med mindre vanskligheter.
IRQ2 - IRQ15	I	Interrupt Request 2 til 15. Disse linjene brukes til å gi signal til prosessoren om at en trenger dens oppmerksomhet. De er prioritert med 2 som høyeste og 15 som laveste prioritet. Et intrrupt blir generert ved en positiv flanke på IRQ linjen, og linjen holdes høy til interruptet er oppdaget.
$\overline{\text{IOR}}$	O	I/O Read Command. Dette signale forteller en komponent at den skal legge data ut på databussen.

$\overline{\text{IOW}}$	O	I/O Write Command. Dette signalet forteller at en komponent skal lese data fra datalinjen.
$\overline{\text{MEMR}}$	O	Memory Read Command. Forteller memory at den skal legge data ut på databussen.
$\overline{\text{MEMW}}$	O	Memory Write Command. Forteller memory at den skal lagre dataene som ligger på databussen.
$\overline{\text{DRQ1-DRQ3}}$	I	DMA Request 1 til 3. Disse linjene er "asynchronous channel request" som blir brukt av perifere komponenter for å oppnå DMA tjenester.
$\overline{\text{DACK0-}}$ DACK3	O	DMA Acknowledge 0 til 3. Disse linjene blir brukt til å gi "acknowledge" på DMA - Request 1 til 3.
AEN	O	Adress Enable. Denne linjen brukes til å disable (de-gate) prosessoren og andre deler fra I/O kanalene for å tillate DMA overføringer.
T/C	O	Terminal Count. Denne linjen gir ut en puls når "terminal count" er for noen av DMA kanalene er nådd.

Tabell 2: I/O Adress Map

000-00F	DMA chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer8253-5
060-063	PPI 8255A-5
080-083	DMAPage Registers
0Ax	NMI Mask Registers
0Cx	Reserved
0Ex	Reserved
100-1FF	Not Useable
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asyncronous Communications(Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk

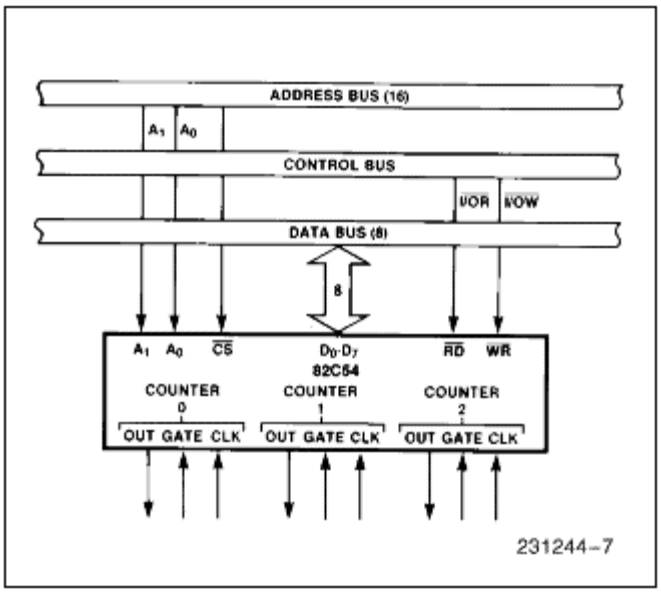
378-37F	Parallel Printer
380-38F	SDLC Communications
3A0-3AF	Reserved
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communications (Primary)

Appendiks C

Programmerbar timer 82C54

System Interface

Timeren blir behandlet av software som en tabell av perifere I/O porter. Tre av dem er tellere og den fjerde er control register for mode programmering.



Figur 3. Styring av timer *(Kilde: Datablad)*

Programmering av timer

For å programmere èn av tellerne må en først skrive et kontrollord til timeren og deretter en teller verdi. Timeren må programmeres hver gang en starter programmet og den kan omprogrammeres ved å skrive inn et nytt kontrollord og en ny tellerverdi hvor mange ganger en vil. Kontrollordet har følgende format:

the Control Word used.

Control Word Format

$A_1, A_0 = 11$ $\overline{CS} = 0$ $\overline{RD} = 1$ $\overline{WR} = 0$

D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC — Select Counter:

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Read-Back Command (See Read Operations)

M — MODE:

M2	M1	M0	
0	0	0	Mode 0
0	0	1	Mode 1
X	1	0	Mode 2
X	1	1	Mode 3
1	0	0	Mode 4
1	0	1	Mode 5

RW — Read/Write:

RW1	RW0	
0	0	Counter Latch Command (see Read Operations)
0	1	Read/Write least significant byte only.
1	0	Read/Write most significant byte only.
1	1	Read/Write least significant byte first, then most significant byte.

BCD:

0	Binary Counter 16-bits
1	Binary Coded Decimal (BCD) Counter (4 Decades)

NOTE: Don't care bits (X) should be 0 to insure compatibility with future Intel products.

Figur 4 Kontrollordformat *(Kilde: Datablad)*

Timer 82C54 Moder

Mode 0: **Interrupt on terminal count**

- Teller hendelser
- Etter at kontrollordet er skrevet går utgangen lav og ligger lav inntil telleren når 0
- Når telleren blir 0 går utgangen høy og holdes høy inntil en ny tellerverdi eller kontrollord blir skrevet til den.

Mode 1: **Hardware retriggable one-shot**

- Utgangen vil ved initiering være høy. Den vil legges lav ved klokkepuls etter triggesignal for å starte "one shot pulse", og ligger lav til telleren har nådd 0.
- Utgangen går så høy, og ligger høy inntil klokkepulsen etter neste triggesignal.

Mode 2: **Rate generator**

- Fungerer som en "divide by N" teller og brukes til å generere "Real time clock interrupt"
- Utgangen vil ved initialisering være høy.

- Når tellereren har kommet til 1 vil utgangen legges lav i en klokkepuls.
- Utgangen går deretter høy og ny nedtelling starter.
- Mode 2 er periodisk. Samme sekvensen gjentas i det uendelige så lenge "Gate" ligger høy.

Mode 3: Square wave generator

- Brukes til å lage klokkesignal (baud rate generator).
- Utgangen vil ved initiering liggge høy.
- Når halve telleverdien er talt ned vil utgangen ligge lav ved resten av nedtellingen.
- Mode 3 er periodisk. Samme sekvens gjentas i det uendelige så lenge "Gate" ligger høy.

Mode 4: Software triggered strobe

- Utgangen vil ved initialisring være høy.
- Når tellerverdien går ut vil utgangen gå lavt for en klokkepuls og så gå høy igjen.
- Tellesekvensen "trigges" ved å legge inn tellerverdi.
- Telleren lastes på første klokkepuls etter kontrollord og tellervedi.

Mode 5: Hardware triggered strobe (retriggable)

- Utgangen vil ved initiering være høy.
- Tellingen "trigges" ved stigende flanke på "Gate".
- Når telleren går ut vil utgangen gå lav i en klokkepuls og så gå høy igjen.
- Forskjellen på Mode 4 og Mode 5 er at i Mode 5 vil ikke telleren bli lastet før klokkepulsen etter trigging.

Felles for alle moder:

- Når et kontrollord blir skrevet til en teller, blir all "Control logic" øyeblikkelig reset'et, og utgangen går til initieringstilstand. Det trengs ingen klokkepuls til dette.
- "Gate" inngangen blir alltid samplet på stigende flanke på klokken.
- I noen moder er "Gate" nivå sensitiv, og noen er sensitive for stigende flanke. Noen moder er begge deler.
- Nye tellerverdier blir lastet og tellerer dekrementeres ved falllende flanke å klokken.
- Den største tellerverdi er 0. Dette tilsvarer 2^16 binært.
  Telleren stopper ikke når den når 0. I Mode 0, 1, 4 og 5 vender telleren rundt den høyeste tellerverdi (FFFFh) og fortsetter å telle ned. I Mode 2 og 3 (som er periodiske) laster telleren seg selv med tellerverdien og fortsetter å telle fra der.

Tidligere har vi nevnt at vi brukte mode 3 til filteret, mode 2 til å generere samplingssignal og mode 0 til å telle antall samplinger.
Teller 0 brukes som samplingssignal til A/D-omformeren. Denne opererer i mode 2. Hvis vi vil skrive en verdi større enn 255 til timeren, gjøres dette ved å skrive to byte etterhverandre til timeren. Dette gir kontrollord:

00110100 = 52 desimalt.

Dette er kontrollord0 som gir telleren informasjon om hvordan teller0 skal oppføre seg. Tellerverdier kan lastes inn på et senere tidspunkt, men kontrollordene er faste.
For teller1 velger vi mode0. Dette er en hendelsesteller, som skal vite hvor mange samplinger som er samplet, og hvor mange som er igjen. Alle andre instillinger er lik det forrige kontrollordet. Dette gir oss:

01110010 = 114 desimalt

For teller2, skal vi ha mode3. Denne telleren skal være klokke for filteret.
Kontrollordet blir som følger:

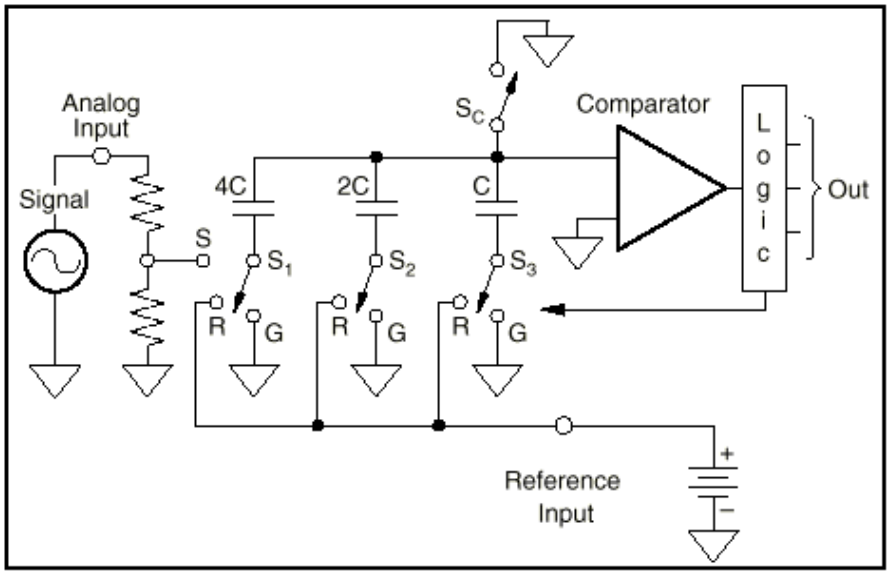
10110110 = 182 desimalt

Appendiks D

A/D omformer ADS774

Succsessiv approximation

For å digitalisere signalet benytter ADS774 en innebygd "successiv approximation" krets (figur 5). Dette er en krets som har en kondensator for hvert bit. Den utfører så gjentatte sammenligninger med det innkomne signal. Først tester den om spenningen inn er større enn den spenning som assosieres med MSB. Dersom den er større, settes MSB = "1" ellers settes det = "0". Deretter skjekker den hvert bit etter tur helt ned til LSB. Hvert bit lagres i et register, som også enabler utgangen etter at alle bit er kontrollert.



Figur 5. Skisse som viser 3 bit successive approksimation (Kilde: Datablad ADS774)

ADS774 kan styres ved hjelp av ulike kontrollsignaler. En beskrivelse av de forskjellige signalene er gitt i figur 6.

DESIGNATION	DEFINITION	FUNCTION
CE (Pin 6)	Chip Enable (active high)	Must be HIGH ("1") to either initiate a conversion or read output data. 0-1 edge may be used to initiate a conversion.
$\overline{CS}$ (Pin 3)	Chip Select (active low)	Must be LOW ("0") to either initiate a conversion or read output data. 1-0 edge may be used to initiate a conversion.
R/$\overline{C}$ (Pin 5)	Read/Convert ("1" = read) ("0" = convert)	Must be LOW ("0") to initiate either 8- or 12-bit conversions. 1-0 edge may be used to initiate a conversion. Must be HIGH ("1") to read output data. 0-1 edge may be used to initiate a read operation.
A_0 (Pin 4)	Byte Address Short Cycle	In the start-convert mode, A_0 selects 8-bit (A_0 = "1") or 12-bit (A_0 = "0") conversion mode. When reading output data in two 8-bit bytes, A_0 = "0" accesses 8 MSBs (high byte) and A_0 = "1" accesses 4 LSBs and trailing "0s" (low byte).
12/$\overline{8}$ (Pin 2)	Data Mode Select ("1" = 12 bits) ("0" = 8 bits)	When reading output data, 12/$\overline{8}$ = "1" enables all 12 output bits simultaneously. 12/$\overline{8}$ = "0" will enable the MSBs or LSBs as determined by the A_0 line.

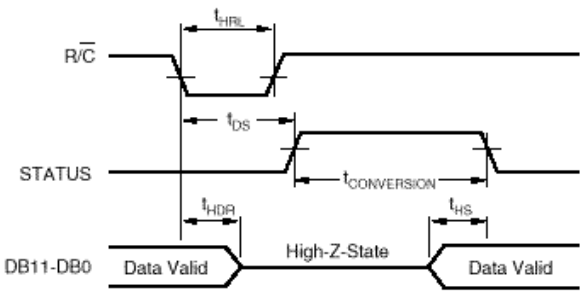
Figur 6. Beskrivelse av de forskjellige kontrollsignaler *(Kilde: Datablad)*

Virkningen av de ulike signalene er gitt i sannhetstabell i figur 7. I "stand alone" operasjon benytter en kun R/C signalet slik som vist over den stiplede linjen.

CE	$\overline{CS}$	R/$\overline{C}$	12/$\overline{8}$	A_0	OPERATION
0	X	X	X	X	None
X	1	X	X	X	None
↑	0	0	X	0	Initiate 12-bit conversion
↑	0	0	X	1	Initiate 8-bit conversion
1	↓	0	X	0	Initiate 12-bit conversion
1	↓	0	X	1	Initiate 8-bit conversion
1	0	↓	X	0	Initiate 12-bit conversion
1	0	↓	X	1	Initiate 8-bit conversion
1	0	1	1	X	Enable 12-bit output
1	0	1	0	0	Enable 8 MSBs only
1	0	1	0	1	Enable 4 LSBs plus 4 trailing zeroes

Figur 7. Sannhetstabell *(Kilde: Datablad)*

For å finne ut når dataene kunne lastes inn i FIFO måtte vi studere timing diagrammet til A/D – omformeren og FIFO'en. Figur 8 viser timing diagram til ADS774 og de ulike tidene er angitt i tabell 3
Timing Diagram

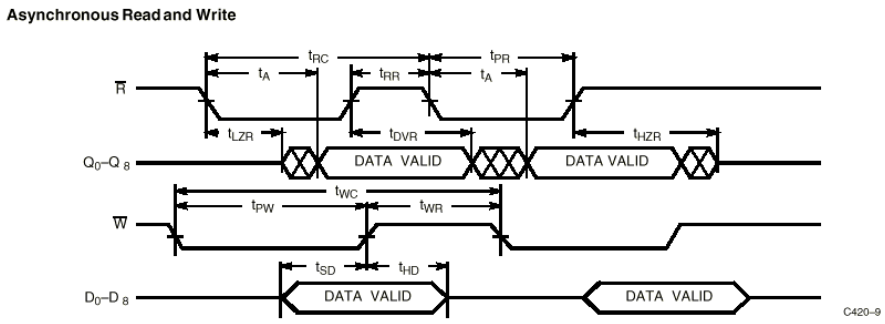


Figur 8. Timing diagram for ADS774 som viser når data på utgangen er stabile (Kilde: Datablad)

Tabell 3: Timing tabell for ADS774

SYMBOL	PARAMETER	MIN	TYP	MAX	UNITS
Convert Mode					
t_{DSC}	STS delay from CE		60	200	ns
t_{HEC}	CE Pulse width	50	30		ns
t_{SSC}	$\overline{CS}$ to CE setup	50	20		ns
t_{HSC}	$\overline{CS}$ low during CE high	50	20		ns
t_{SRC}	R/$\overline{C}$ to CE setup	50	0		ns
t_{HRC}	R/$\overline{C}$ low during CE high	50	20		ns
t_{SAC}	A_0 to CE setup	0			ns
t_{HAC}	A_0 valid during CE high	50	20		ns
Read Mode					
t_{DD}	Access time from CE		75	150	ns
t_{HD}	Data valid after CE low	25	35		ns
t_{HL}	Output float delay		100	150	ns
t_{SSR}	$\overline{CS}$ to CE setup	50	0		ns
t_{SRR}	R/$\overline{C}$ to CE setup	0			ns
t_{SAR}	A_0 to CE setup	50	25		ns
t_{HSR}	$\overline{CS}$ valid after CE low	0			ns
t_{HRR}	R/$\overline{C}$ high after CE low	0			ns
t_{HAR}	A_0 valid after CE low	50			ns
t_{HS}	STATUS delay after data valid	75	150	375	ns

Figur 9 viser timingdiagrammet til FIFO`en, og tabell 4 viser de aktuelle tidene.



Figur 9. Timingdiagram for FIFO. (Kilde: Datablad)

Tabell 4: Timing tabell for FIFO

Switching Characteristics Over the Operating Range[6, 7]

Parameter	Description	7C419–10 7C421–10 7C425–10 7C429–10 7C433–10		7C419–15 7C421–15 7C425–15 7C429–15 7C433–15		7C420–20 7C421–20 7C424–20 7C425–20 7C428–20 7C429–20 7C433–20		7C420–25 7C421–25 7C424–25 7C425–25 7C429–25 7C432–25 7C433–25		Unit
		Min.	Max.	Min.	Max.	Min.	Max.	Min.	Max.	
t_{RC}	Read Cycle Time	20		25		30		35		ns
t_A	Access Time		10		15		20		25	ns
t_{RR}	Read Recovery Time	10		10		10		10		ns
t_{PR}	Read Pulse Width	10		15		20		25		ns
t_{LZR}[5,8]	Read LOW to Low Z	3		3		3		3		ns
t_{DVR}[8,9]	Data Valid After Read HIGH	5		5		5		5		ns
t_{HZR}[5,8,9]	Read HIGH to High Z		15		15		15		18	ns
t_{WC}	Write Cycle Time	20		25		30		35		ns
t_{PW}	Write Pulse Width	10		15		20		25		ns
t_{HWZ}[5,8]	Write HIGH to Low Z	5		5		5		5		ns
t_{WR}	Write Recovery Time	10		10		10		10		ns
t_{SD}	Data Set-Up Time	6		8		12		15		ns
t_{HD}	Data Hold Time	0		0		0		0		ns
t_{MRSC}	$\overline{MR}$ Cycle Time	20		25		30		35		ns
t_{PMR}	$\overline{MR}$ Pulse Width	10		15		20		25		ns
t_{RMR}	$\overline{MR}$ Recovery Time	10		10		10		10		ns
t_{RPW}	Read HIGH to $\overline{MR}$ HIGH	10		15		20		25		ns
t_{WPW}	Write HIGH to $\overline{MR}$ HIGH	10		15		20		25		ns
t_{RTC}	Retransmit Cycle Time	20		25		30		35		ns
t_{PRT}	Retransmit Pulse Width	10		15		20		25		ns
t_{RTR}	Retransmit Recovery Time	10		10		10		10		ns

Notes:

6. Test conditions assume signal transition time of 3 ns or less, timing reference levels of 1.5V and output loading of the specified I_{OL}/I_{OH} and 30 pF load capacitance, as in part (a) of AC Test Load and Waveforms, unless otherwise specified.
7. See the last page of this specification for Group A subgroup testing information.
8. t_{HZR} transition is measured at +200 mV from V_{OL} and –200 mV from V_{OH}. t_{DVR} transition is measured at the 1.5V level. t_{HWZ} and t_{LZR} transition is measured at ±100 mV from the steady state.
9. t_{HZR} and t_{DVR} use capacitance loading as in part (b) of AC Test Load and Waveforms.

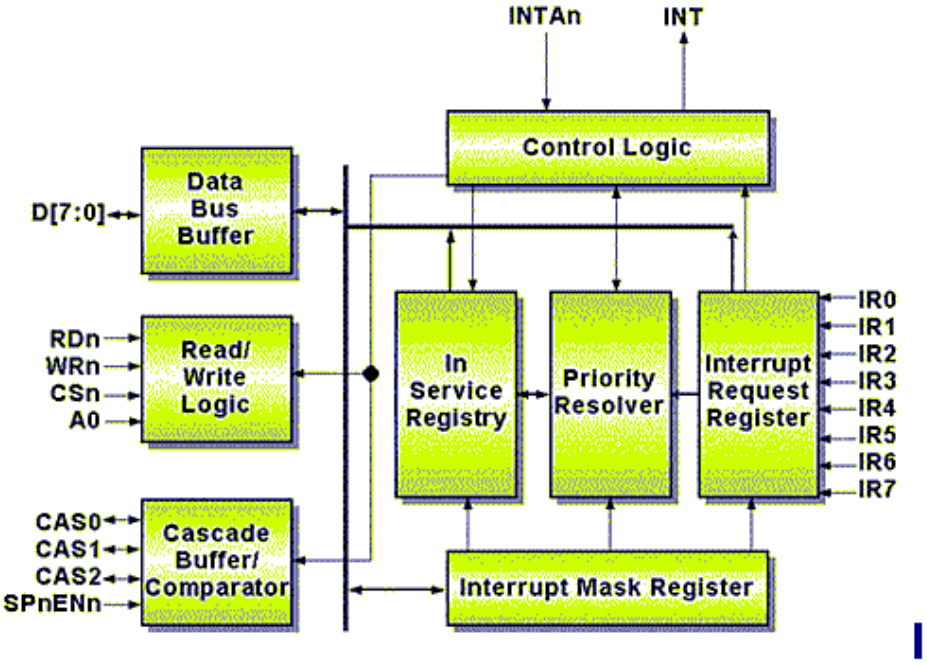
Appendiks E

Software

Dette avsnittet vil gi en grundig innføring i hvordan Windows95 behandler et interrupt. For å få en bedre innsikt i problemstillingen og helheten, vil vi referere til software-kapittelet. Vil også henvise til en interrupt tutorial på internettside:
http://www.pcwebopaedia.com/interrupt_request_line.htm

Interrupt i Windows95

I stedet for å polle en port for å finne ut om det er gyldige data på porten, er det i mange tilfeller ønskelig at PC`en skal reagere på endringer på en bestemt interruptlinje. Det er for bedre å kunne utnytte kapasiteten til PC`en at man derfor velger å bruke interrupt. For å kunne registrere interruptene som kommer til Windows95 , må man vite litt om operativsystemets oppbygning. Windows95 er bygget opp på den måten at alle hardware-interrupt blir registrert av en VMM(Virtual Machine Manager). VMM er en softwareoppbygget enhet av PC`en og kan finnes som VMM32.vxd i Windows/System-katalogen. Når interruptlinjene til PC`en aktiviseres, vil interruptet rutes til en PIC (Programmable Interrupt Controller). Denne kontrolleren har en del tabeller som må undersøkes før interruptet blir gyldig. Under vises et blokkskjema over PIC`en:



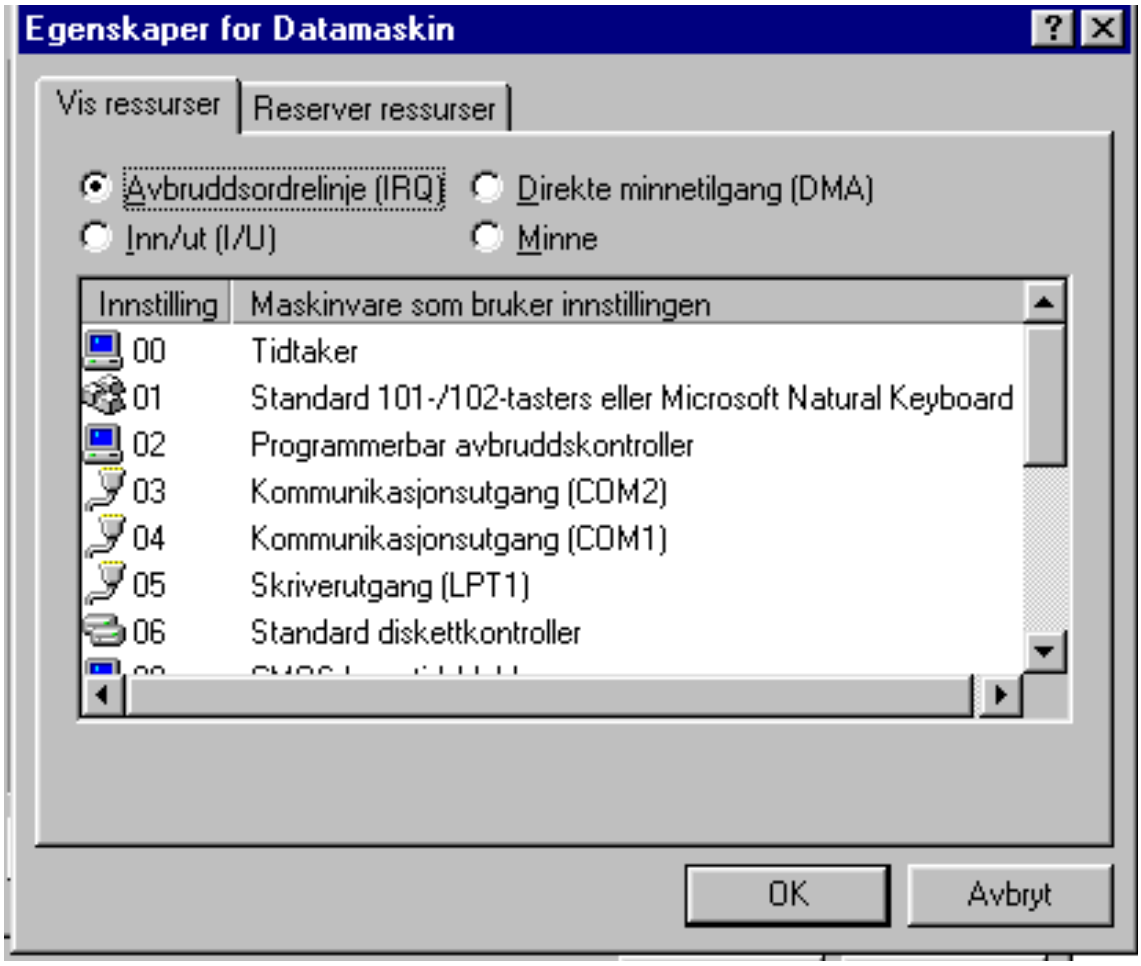
Figur 10 Programmable Interrupt Controller *(Kilde: Cast.inc)*

Hvis kontrolleren finner interruptet gyldig, vil den prøve å avbryte prosessoren. Den sender så en verdi til den. Prosessoren vil nå kalle opp VMM, og den vil ta kontrollen over prosessoren. VMM`en vil starte funksjonen som kan «svare» på hardwarekallet.

For å kunne drive med interrupt i Windows95 trenger vi en type programmer som kalles VxD`er. Når en hardware-innretning trenger oppmerksomhet fra prosessoren, vil den gi et interrupt på en av interruptlinjene. For at PC`en skal vite hvilken funksjon som skal kalles, må hver av interruptlinjene ”knyttes” til en bestemt funksjon. En VxD er egentlig et program som skaper en slags virtuell virkelighet. Den vil skape et eget «miljø» for applikasjonen som kaller den, og simulere at applikasjonen har maskinen for seg selv. I virkeligheten er dette en form for multipleksing, der hver applikasjon får litt prosessortid hver.

Bakgrunnen for at Microsoft begynte utviklingen av VxD`er var behovet for «multitasking» i PC`en. I MS-DOS måtte Windows-programmerere periodisk gi fra seg kontrollen over operativsystemet (MS-DOS) for at andre applikasjoner også skulle kunne bli utført. Virtuelle maskiner (VxD`er) i Windows95 er derfor løsningen på dette problemet. For hver applikasjon vil det se ut som om denne «eier» hele maskinen. Dette er fordi den Virtuelle Maskinen vil simulere at programmet/applikasjonen har kontroll over keyboard, mus, hardware, skjerm osv. Sannheten er imidlertid at alle applikasjonene deler på prosessortiden, og eier maskinen bare deler av tiden.

På grunn av forskjellig prioritetsnivå på interruptene må man ved installeringen av VxD`en angi et interruptnivå. Som tidligere forklart er det 15 interruptnivåer i en PC, i denne oppgaven bruker vi interrupt nummer 15.



Figur 11. Interruptinstillingene til PC, under Windows95.

DriverX

ActiveX kontrollen DriverX er laget av Tetradyne co. og kan kjøpes over Internettet. Dette er en kontroll som gir brukeren mulighet til å kunne benytte seg av interruptstyrt programmering. Ved hjelp av et eget medfølgende program kan man sette alle de innstillinger som man ønsker ActiveX kontrollen skal ha. Fordi VxD programmering er noe spesielt, må man for å lage slike filer anskaffe seg et eget ”Development Kit” som selges av Microsoft.

I DriverX instillingene kan man velge om man vil ha ”Direct Memory Access” (DMA) , I/O access, eller ingen access. Fordi vi ikke leser inn data ved hjelp av ActiveX-kontrollens funksjoner satte vi innstillingene til ”ingen access”. Hvis vi hadde brukt kontrollen ville vi valgt ”I/O access”. I tillegg til dette kunne vil velge å bruke interrupt. Vi måtte da sette alle interruptinstillingene til kretsen og ”starte kretsen” med en ”Start”-knapp. Den siste instillingsmuligheten var å bruke ”Dynamisk oppstart” eller ”Statisk oppstart”. ”Dynamisk oppstart” betyr manuell start av kretsen, mens ”Statisk start” betyr at kretsen starter under maskinens oppstart. Vi valgte ”Statisk start”, og gav kretsen navnet ”Prototype”.

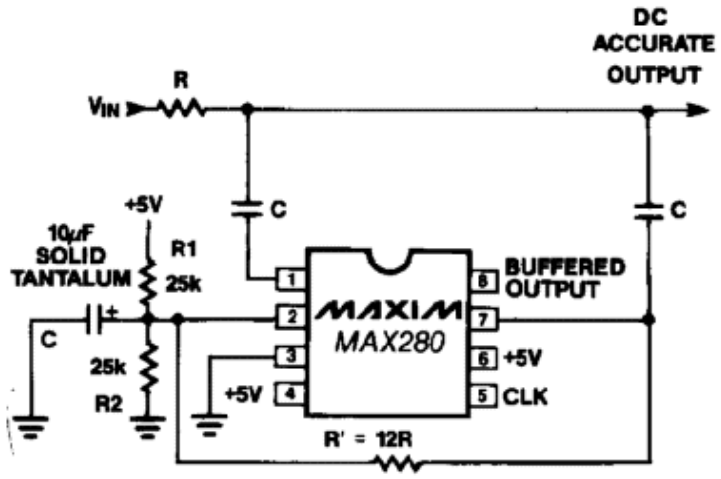
For å kunne bruke AxtiveX kontrollen må en kalle opp en funksjon ”ConnectDevice()”.
For å enable interruptet, må en kalle en funksjon ”EnableIrs()”.

Appendiks F

Antialiaseringsfilter og forsterker

Siden vi kan bestemme samplingsfrekvensen på A/D - omformeren ved hjelp av software er det også ønskelig å kunne gjøre det samme med antialiaseringsfilteret. Vi bruker et 5 ordens lavpassfilter fra Maxim, MAX280. Dersom en trenger høyere ordens filter går det an å kaskadekoble to filtre slik at en får et 10. ordens filter. Dette filteret kan ha cutoff frekvens fra DC til 20kHz. Cutoff frekvensen kan bestemmes av en ekstern klokke. Forholdet mellom klokke og cutoff frekvens er 100:1.

Oppkobling



Figur 12. Oppkobling av 5.ordens Butterworth-filter (Kilde: Datablad)

Verdiene for R og C velges ut fra følgende formel:

$$\frac{1}{2\pi RC} = \frac{f_c}{1{,}84} \qquad \text{,der } f_c \text{ er cutoff frekvens}$$

Videre er det angitt at vedien på R er i størrelsesorden 20 kΩ, og at minimumverdien er på 1 kΩ. Ved å velge 13,3 kΩ motstand finner en at verdien på C = 2,2nF og R' = 12*R = 158 kΩ.

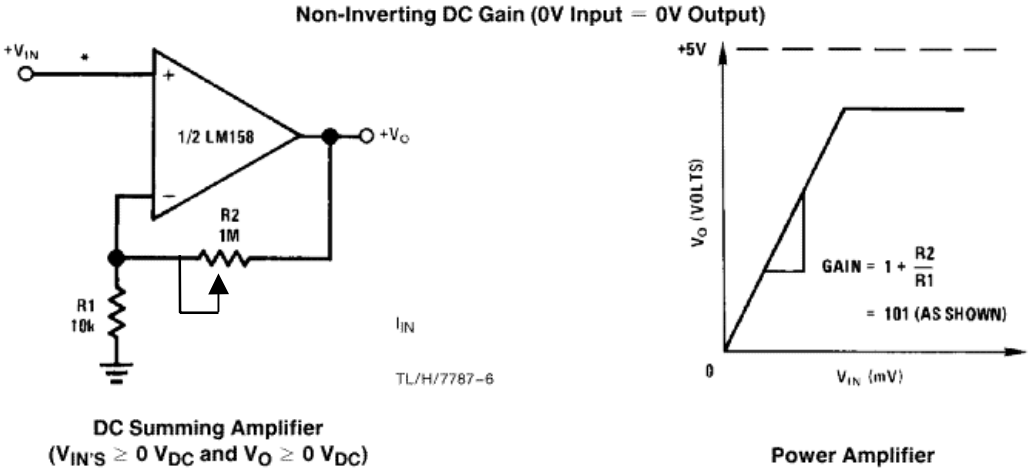
Ved å benytte timer 2 som "square wave generator" (mode 3) kan en ved hjelp av denne bestemme cutoff frekvensen. Dette betyr at den kan styres av software. Utgangen på timer 2 kobles til pin 5 (clk) på filteret, og må ha en frekvens som er hundre ganger større en cutoff frekvensen.

Forsterker

Etter at signalet har vært gjennom filteret forsterkes det opp slik at en får full nytte av oppløsningen til A/D-omformeren. Solenoidstrømmen som skal analyseres er i størrelsesorden 15 A. Ved å ta ut signalet over en 0.1 Ω høyeffektsmotstand vil signalet være opptil 1,5 V. Vi lar signalet gå inn på A/D - omformerens 10 V unipolar inngang. Av denne grunn bruker vi en ikke inverterende forsterkerkobling med variabel forsterkning fra 1 til 11. Vi bruker er LM 358 operasjonsforsteker fra National Semiconductor. Forsterkningen endres ved å justere R2 som er et 100 kΩ presisjons potensiometer. Ved å la R1 være lik 10 kΩ får vi følgende uttrykk for forsterkningen:

$$A = 1 + \frac{R2}{R1}$$

Oppkobling



Figur 13 Oppkobling av ikke inverterende forsterker *(Kilde: Datablad)*

Spenningen ut fra forsterkeren justeres slik at den blir maximum 10 V. Dette gjør at en får maksimal oppløsning når den kobles til pin 13 på A/D - omformeren, som er 10 V unipolar mode.

Filterteori:

For å forstå hvordan aliaseringsfeil oppstår må en kjenne til hva som skjer når vi sampler et signal. Vi kan tenke oss et generelt sinussignal som kan beskrives på følgende måte.

$x(t) = \sin(2\pi f_0 t)$, der f_o er frekvensen

Vi sampler dette signalet med f_s samplinger / s som gir en periodetid $t_s = 1/f_s$. Dersom vi starter samplingen ved 0 s vil vi få samplinger ved $0t_s$, $1t_s$, $2t_s$ o.s.v. som videre kan skrives som samplingsverdier på formen:

0'te sample: $x(0) = \sin(2\pi f_0 0 t_s)$

1'te sample:	$x(1) = \sin(2\pi f_0 1 t_s)$
2'te sample:	$x(2) = \sin(2\pi f_0 2 t_s)$
3'te sample:	$x(3) = \sin(2\pi f_0 3 t_s)$
...	...
...	...
n'te sample:	$x(n) = \sin(2\pi f_0 n t_s)$

Vi har fra geometrien følgende likhet: $\sin(\phi) = \sin(\phi + 2\pi m)$, der m er et heltall.

Vi kan nå skrive det generelle utrykket for der et sample:

$$x(n) = \sin(2\pi f n t_s) = \sin(2\pi f n t_s + 2\pi m) = \sin(\, 2\pi(\, f_0 + m/n t_s \,) n t_s)$$

Vi lar m være et heltalls multiplum av n, m = kn slik at:

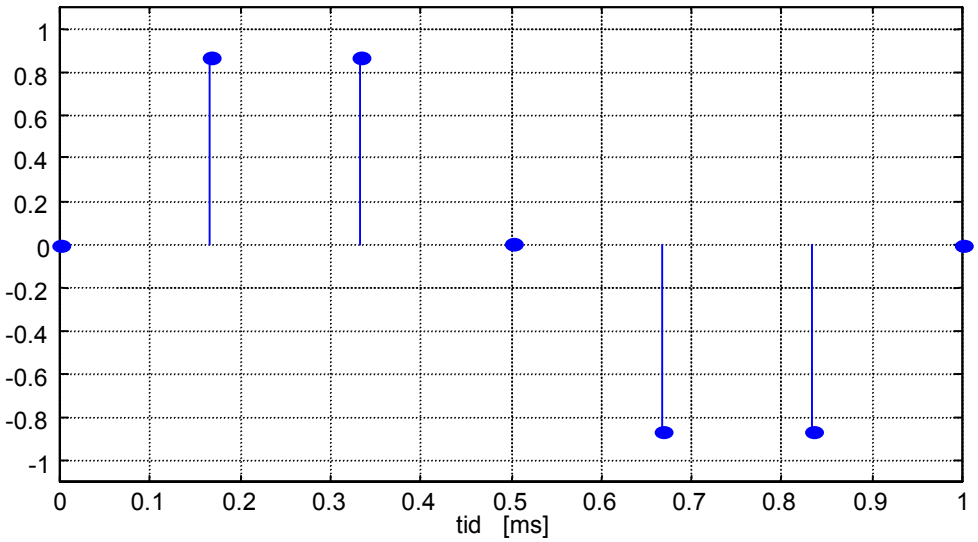
$$x(n) = \sin(\, 2\pi(\, f_0 + k/t_s \,) n t_s)$$

Siden $f_s = 1/t_s$ kan vi skrive:

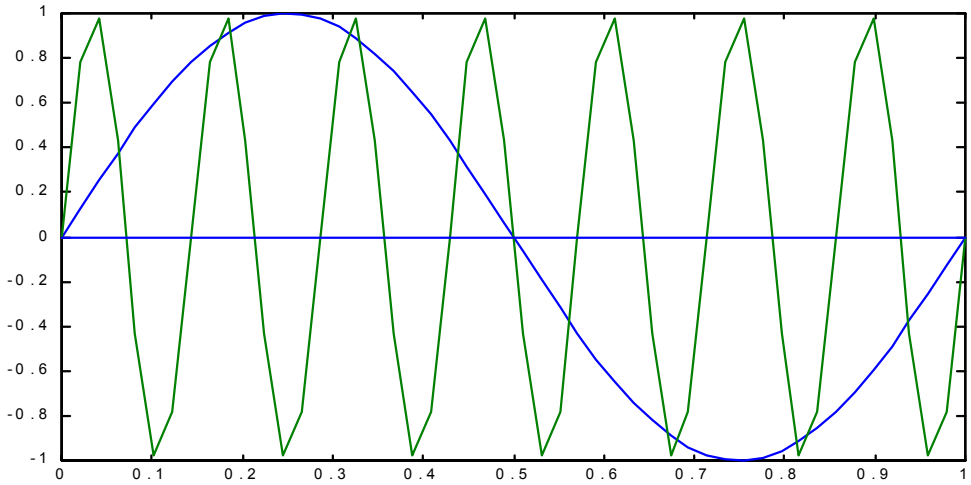
$$x(n) = \sin(2\pi f_0 n t_s) = \sin(\, 2\pi(\, f_0 + k f_s \,) n t_s)$$

Vi ser her at faktorene f_0 og $(\, f_0 + k f_s \,)$ svarer til hverandre. Dette betyr at digitalt samplede verdier som representerer en sinus med frekvens f_0 også kan representere nøyaktig en sinus med andre frekvenser, nærmere bestemt $f_0 + k f_s$.

Vi skal nå se på hva som skjer dersom den høyeste frekvensen til signalet ligger over $f_s/2$. Som eksempel ser vi på et signal på 7 kHz som vi sampler med 6 kHz.



Figur 14. Diskrete verdier av 7 kHz sinussignal som er samplet med en samplingsfrekvens på 6 kHz

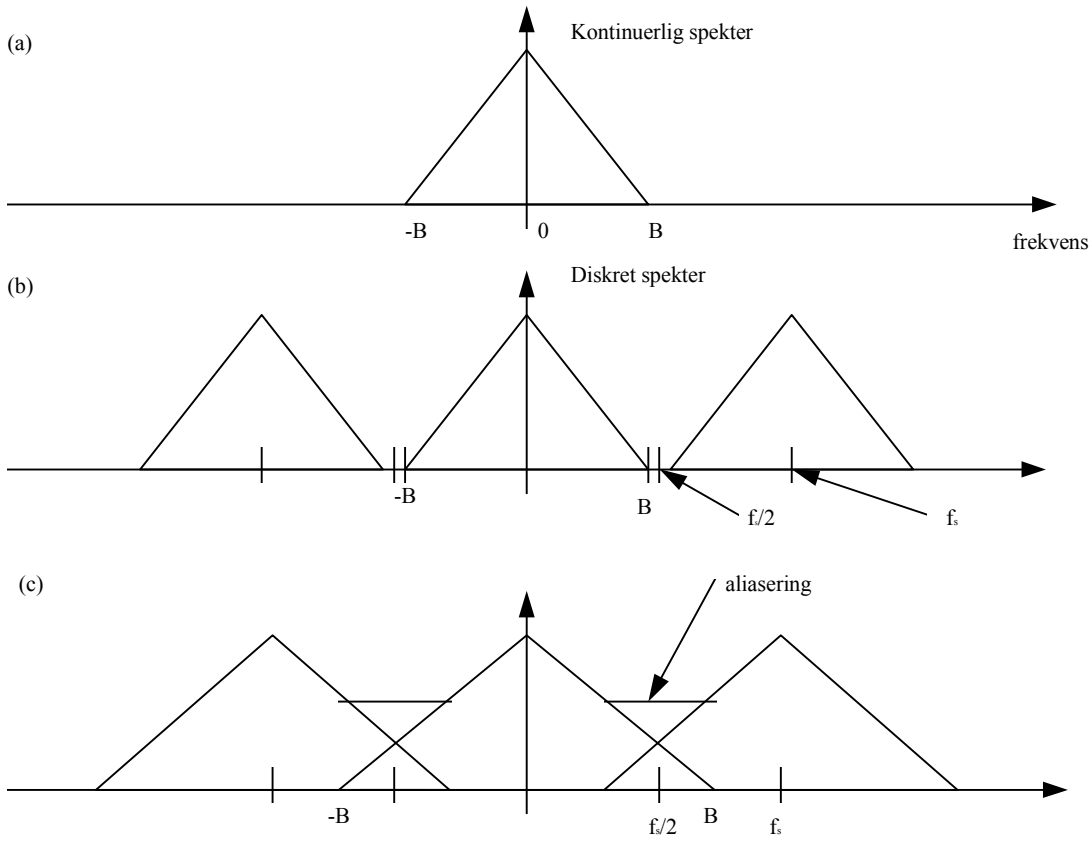


Figur 15. Sinussignaler på 7kHz og 1Khz

Dersom en ser på den øverste figuren som består av diskrete verdier av et 7kHz sinusignal samplet med en samplingsfrekvens på 6 kHz kan ikke med sikkerhet si hvilket signal det representerer. Dersom vi setter inn k =-1 i formelen som er utledet over får vi:

$$f_0 + kf_s = 7kHz + (-1*6kHz) = 1kHz$$

Dette betyr at vi får det samme resultat enten vi sampler et sinussignal på 1kHz eller et på 7kHz dersom samplingsfrekvensen er 6kHz. Vi kaller derfor 1kHz signalet for et alias av 7kHz signalet. En annen måte å beskrive aliasering på er å se på frekvensspekteret til signalet.



Figur 16. Frekvensspekter

Figur 16(a) viser frekvensspekteret til et kontinuerlig signal. Figur 16(b) viser frekvensspekteret til det samplede signalet når samplingsfrekvenser er > 2B, og figur 16(c) viser spekteret til det samplede signalet der samplingsfrekvensen er < 2B. Vi ser da at vi får overlappende frekvenskomponenter som ikke kan fjernes.

Vedlegg