CODEN: LUTFD2/(TFRT-5426)/1–103/(1990)

Snabb mätvärdesinsamling med en IBM AT

Patrik Nilsson
Leif Nilsson

Institutionen för Reglerteknik
Lunds Tekniska Högskola
Oktober 1990

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
MASTER THESIS

Date of issue
October 1990

Document Number
CODEN: LUTFD2/(TFRT-5426)/1–103/(1990)

Author(s)
Patrik Nilsson
Leif Nilsson

Supervisor
Rolf Johansson, Dag Brück

Sponsoring organisation

Title and subtitle
Snabb mätvärdesinsamling med en IBM AT
(Fast Sampling Using the IBM AT)

Abstract

The purpose of our master thesis was to write a fast and flexible program for data sampling with an IBM AT personal computer.

A special scheduler developed at the Department of Automatic Control constituted the base for our work. This foreground-background scheduler is a simple real-time kernel which can handle only two processes. In order to meet our demands on frequency range and exactness of the sampling frequency we had to make some changes in the scheduler.

The scheduler, written in assembler, is designed for foreground routines written in C, which is a fast and efficient language. Our C program is directed with the help of seventeen different user commands to sampling or supplementary work on data. Except for data storage the supplementary work consists of FFT and plotting. The user is offered 24 different sampling alternatives. The maximum sampling rate is between 3 and 6 kHz when data is stored internally. When data is stored on an external file the maximum sampling rate is 33 Hz regardless of the sampling alternative.

For some reason the sampling frequency does not remain constant if Matlab is used between two executions of the program. We have been unable to solve this problem. Further it would be desirable to develope the program to include filtering and a PID-controller.

Key words

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
Swedish

Number of pages
103

Recipient's notes

Security classification

Innehållsförteckning

Förord

Denna rapport är resultatet av teknologerna Leif Nilssons och Patrik Nilssons examensarbete. Våren 1989 bestämde vi oss för att examensarbetet skulle utföras sommaren 1989. Efter diverse samtal med olika företag samt besök på olika institutioner bestämdes att arbetet skulle utföras på Institutionen för Reglerteknik vid Lunds Tekniska Högskola. I samråd med högskolelektor Rolf Johansson, beslutades att examensarbetet skulle ligga inom ämnesområdet snabb mätvärdesinsamling. Bättre förklarat innebar detta att vi skulle skriva ett program som via A/D - ingångar läste in värden till en persondator, i vårt fall en Tandon AT. Programmet skulle dessutom innehålla diverse "finesser" typ FFT (snabb Fourier transform), grafisk presentation av insamlade data etc. Eftersom programmet använder sig av realtid behövdes en realtidskärna och en sådan tillhandahölls av Dag Brück. Handledare för vårt examensarbete var högskolelektor Rolf Johansson och forskningsassistent Dag Brück. Arbetet slutfördes sommaren 1990.

Erkännande

Vägen till färdigt examensarbete har varit både lång och snårig, och utan hjälp från en mängd personer hade vi aldrig nått fram. Därför vill vi nu passa på att tacka dessa.

Först och främst vill vi tacka våra bägge handledare, högskolelektor Rolf Johansson och forskningsassistent Dag Brück. Rolf Johansson har gett oss många goda råd och hjälp i allmänhet. Dag Brück har hjälpt oss i synnerhet med programmeringsarbetet, dessutom är realtidskärnan konstruerad av Dag Brück.

Vidare vill vi tacka Rolf Braun för hjälp med, samt utlåning av, diverse utrustning. Andra personer vi vill tacka är Thomas Schönthal för hans goda råd angående assembler, samt övriga personer på institutionen som känner sig träffade. I övrigt är vi mycket tacksamma för våra föräldrars fina markservice.

För att ha tackat så många som möjligt vill vi slutligen rikta ett sista tack till Perikles & CO som hjälpt (och ibland stjälpt!) oss igenom många långa nätter.

1. Introduktion

1.1 Snabb mätvärdesinsamling och -filtrering

Bakgrund

På Institutionen för Reglerteknik görs idag nästan all datainsamling med IBM AT maskiner. Med ordinarie realtidskärna kan man uppnå en samplingsfrekvens på cirka 100 Hz. Många intressanta tillämpningar av digital signalbehandling, reglering och identifiering kräver betydligt högre samplingsfrekvens. För att uppnå högre samplingsfrekvenser har därför en *foreground – background scheduler*, dvs ett realtidssystem med endast två processer, utvecklats på institutionen. För att uppnå högre samplingsfrekvens räcker det inte med att kärnan i sig är snabb utan dessutom måste förgrundsprocessen vara snabb. Av detta skäl är kärnan konstruerad för program skrivna i C.

Beskrivning av examensarbetet

Målet med vårt examensarbete var att skriva ett snabbt och flexibelt program för mätvärdesinsamling. Följande önskvärda prestanda och egenskaper för programmet sattes upp som mål

* Flexibelt och lättanvänt
* Datakanaler, samplingsalternativ och filtreringsalternativ skulle kunna väljas interaktivt före insamlingen
* En högsta samplingsfrekvens mellan 2000 och 5000 Hz
* Ett smidigt gränssnitt till program för mätvärdesbehandling, tex Matlab
* Möjligen en enkel grafisk presentation

Dessa mål har till stor del blivit uppfyllda. Någon filtrering har dock inte implementerats och gränssnittet fick vi förkasta eftersom det visade sig att samplingsfrekvensen ej förblev konstant om Matlab användes mellan två körningar av samplingsprogrammet. Överföring via datafil är dock möjlig.

Senare byggdes dessa mål på med PRBS-generator och FFT, som också har förverkligats.

1.2 Rapportens uppläggning

Kapitlet som följer ger först en översiktlig inblick i programmets olika delar för att sedan mera i detalj beskriva dessa delars funktion. En presentation av realtidskärnan och de förändringar vi gjort här följer sedan i kapitel 3. De två hittills nämnda kapitlen (kapitel 2 och 3) avslutas båda med en kort beskrivning av de uppnådda resultaten.

Användarhandledningen i kapitel 4 behandlar dels diskettens innehåll och dels de kommandon med vilka programmet styrs. Till sist ges ett detaljerat

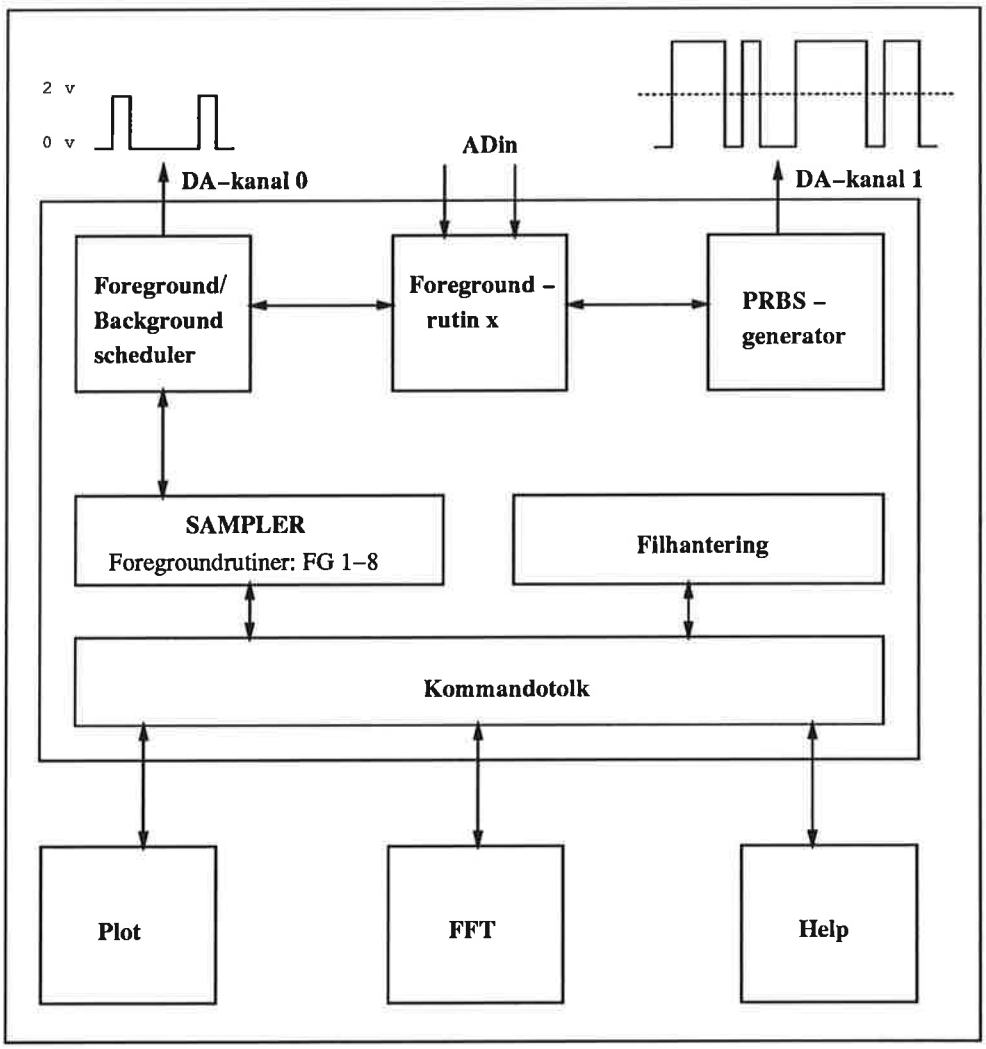
användarexempel. I utvärderingskapitlet (kapitel 5) tar vi upp erfarenheter och problem, dessutom ger vi några synpunkter på hur programmet skulle kunna utvecklas.

Efter referenserna i kapitel 6 följer sju appendix i vilka källkoden till programmets olika delar listats.

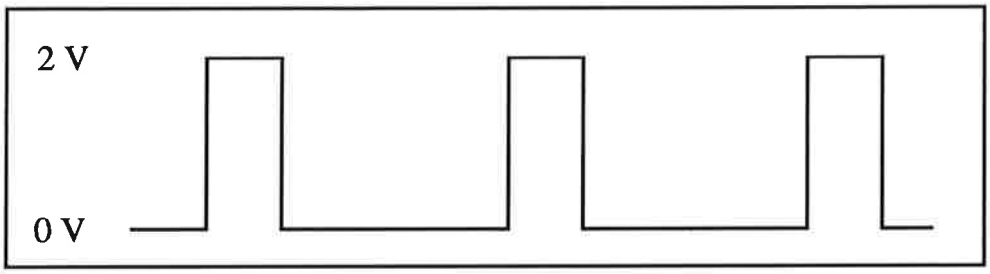
2. Programmets uppbyggnad

2.1 Blockschema över programmet

Programmet består av fyra separata delprogram som alla motsvarar ett block i Figur 2.1. Källkoden till plotprogrammet är skriven i Modula-2, medan de tre övriga exekverbara programmen har skrivits i Microsoft C. Samplingsprogrammet anropar de tre övriga programmen och kommunicerar med dem med hjäälp av filerna `helpfile.dat` och `plotfile.dat`. Plotprogrammet skrevs i Modula-2 eftersom ett färdigt grafikpaket i Modula-2 fanns att tillgå. Anledningen till de externa C programmen Help och FFT är att *schedulern* är skriven för C program som utnyttjar den lilla minnesmodellen, se vidare Appendix D.



Figur 2.1 Blockschema över programmet.



Figur 2.2 Under den tid som förgrundsprocessen är aktiv är spänningen ut från DA-kanal 0 = 2 V. I övrigt ligger denna spänning alltid på noll. Detta kan nyttjas till att verifiera samplingsfrekvensen.

2.2 Programdelarnas funktion

Kommandotolken

Denna del står för operatörskommunikationen och är utformad som ett enkelt kommandospråk. Här läses ett kommando in från terminalen och om kommandot är korrekt utförs önskad åtgärd, ibland med hjälp av övriga programdelar. Med några av de totalt 17 tillgängliga kommandona, som beskrivs närmre i användarhandledningen, kan önskat samplingsalternativ ställas in. Beroende på hur samplingen skall ske väljer kommandotolken ut den av samplerns åtta olika förgrundsprocesser som skall användas vid samplingen.

Samplern

Då önskat samplingsalternativ bestämts anropas samplern genom att ge kommandot RUN. Om allt är som det ska, anropas samplern som gör lämpliga initieringar och därefter startar kärnan med den i kommandodelen utvalda förgrundsprocessen. Under datainsamlingen är programmet låst till samplern. Varje gång förgrundsprocessen aktiveras läggs en spänning på två volt ut på DA-kanal 0. Detta kan utnyttjas som synksignal eller för att kontrollera den verkliga samplingsfrekvensen. Synksignalen visas i Figur 2.2.

PRBS-generatorn

Denna procedur är kompilerad enligt samma regler som samplerns förgrundsprocesser, dvs utan "stack-checking" (se vidare i implementeringen av kärnan i Appendix D). Följdaktligen kan denna procedur anropas från aktuell förgrundsprocess varvid en PRBS-signal erhålles på DA-kanal 1. För vidare information se användarhandledningen (kapitel 4).

Filhantering

I detta block finns procedurer som läser in filnamn, kontrollerar att längden på detsamma understiger åtta bokstäver, tar bort felaktiga extensioner, lägger till rätt extensioner, initierar informationsfilen med bland annat tid och datum, överför värden från samplings- och buffertvektorerna till aktuell "smp-fil" mm.

FFT

FFT-algoritmen ligger i ett separat program, kallat FFT. Algoritmen är hämtad från Richard C. Singeltons artikel (1967). Programmet hämtar indata från aktuell "smp-fil". Den läser i tur och ordning in hälften av värdena och lagrar

dessa, samt på varannan plats talet 0.0 (imaginära delen), i en vektor. Därefter lagras på motsvarande sätt resten av värdena i en annan vektor. Är antalet värden inte lika med en jämn potens av två fyller man på med nollor så att det totala antalet komplexa tal blir en jämn potens av två (2^m). Största antalet värden som programmet klarar ges av $M/2$, där M är en konstant given i programmets källkod[1]. M är för tillfället satt till $2^{13} = 8192$ vilket alltså innebär att programmet maximalt klarar att transformera $2^{12} = 4096$ värden.

Formeln för FFT-n använder sig av totalt fyra vektorer, två invektorer, v och w, samt två utvektorer, u och o. Man läser två komplexa tal, ett från varje invektor. Att läsa ett komplext tal innebär att man läser två tal, det första reellt och det andra imaginärt.

En "två-till-två"-transform beräknas och de två resulterande komplexa talen lagras i den första utvektorn. Efter det att $n/2$ komplexa tal har lagrats i den första utvektorn byter man till den andra utvektorn och fortsätter att skriva i denna.

Transformen vid den k:te beräkningsomgången, där $k = 1, 2, ..., m$ har följande utseende:

$$u_{2j} = v_j + w_j \tag{2.1}$$

$$u_{2j+1} = (v_j - w_j) * \exp(i\pi(j \div 2^{k-1})/2^{m-k}) \tag{2.2}$$

för $j = 0, 1, ..., (\frac{n}{2-1})$

där $\div$ representerar heltalsdivision utan rest. De succesiva beräkningsomgångarna är identiska, förutom att man mellan varje beräkningsomgång byter plats på invektorer och utvektorer.

För att snabba upp beräkningen så beräknas cosinus- och sinusfunktionerna enligt följande formler:

$$\cos((k+1)\theta) = \cos(k\theta) + C_{k+1} \tag{2.3}$$

$$C_{k+1} = R * \cos(k\theta) + C_k \tag{2.4}$$

$$\sin((k+1)\theta) = \sin(k\theta) + S_{k+1} \tag{2.5}$$

$$S_{k+1} = R * \sin(k\theta) + S_k \tag{2.6}$$

där $R = -4 * \sin^2(\frac{\theta}{2})$

och begynnelsevärdena är

$$C_0 = 2 * \sin^2(\tfrac{\theta}{2})$$

$$S_0 = \sin(\theta)$$

$$\cos(0) = 1$$

$$\sin(0) = 0$$

Efter m stycken beräkningsomgångar är man färdig. De komplexa talen ligger då i omvänd binär ordning. För att få rätt ordning på dem krävs $m - 1$ sorteringssteg. Vid första steget läses komplexa tal omväxlande från vektorerna v och w samt skrivs ut i vektorn u tills denna är fylld med $n/2$ komplexa tal. Därefter fortsätter man med vektorn o. Efter detta låter man utvektorerna vara invektorer och vice versa. Vid det andra steget läses två komplexa tal från vektorn v, sedan två från vektorn w, och så vidare tills vektorn u är fylld. Därefter skiftar man till vektorn o. Vid det k:te steget läses alltså 2^{k-1}

[1] Se appendix E

komplexa tal från v, sedan 2^{k-1} från vektorn w etc. Vid det sista steget skrivs värdena ut på den aktuella "fft-filen" och den färdiga Fouriertransformen finns alltså i denna fil.

Help

Detta program har hand om alla hjälputskrifter på skärmen. Förutom de procedurer som ger vidare information om kommandona omfattar programmet även de procedurer som används vid listning och information om de filer av typen "inf","smp" och "fft" som redan existerar.

Eftersom utskrifterna är många och omfattande blev vi tvungna att skriva detta program, ty utskrifterna upptar för stor del av den 64 kilobyte stora dataarea som är tillgängliga i Microsofts lilla minnesmodell.

Problemet hade naturligtvis också kunnat kringgås med hjälp av en extern fil innehållande alla hjälptexter, ifrån vilken programmet sedan hade kunnat hämta rätt textavsnitt. Med denna lösning hade vi dock inte kunnat avlasta programmet från informations- och listningsprocedurerna.

Plot

Den grafiska presentationen görs med hjälp av ett separat program som skrivits i Modula-2. Detta program startas inifrån samplingsprogrammet med anropet *system*("*plot*"). Plotprogrammet utnyttjar de grafikrutiner som utvecklats på institutionen av Leif Andersson.

Kommunikationen med plotprogrammet sker med hjälp av filen `plotfile.dat`. Plotprogrammet börjar med att hämta det heltal som ligger i denna fil. Den mest signifikanta ettan i detta tal, som alltid är större än 10000, ignoreras medan de två nästföljande talen används för att skala den vertikala axeln. Denna skalning skiljer sig beroende på om det är en "fft-fil" eller en "smp-fil". Om en "fft-fil" skall plottas skall talet i `plotfile.dat` vara 1xx01 annars 1xx00. Att en fil är en "fft-fil" innebär att filen har extensionen fft och att värdena i filen härstammar från en Fouriertransformering av insamlade värden. En "smp-fil" har på samma sätt extensionen smp och innehåller insamlade data.

Programmet klarar endast filer med en eller två signaler, där signalerna motsvarar varsin "kolonn" i filen. Observera att även en "fft-fil" innehåller en eller två kolonner, eftersom Fouriertransformens reella och imaginära delar placeras efter varandra. En "fft-fil" är alltså precis dubbelt så stor som motsvarande "smp-fil".

Om två signaler lagrats i en fil kommer den första kolonnen att plottas blå och den andra kommer att plottas röd. Efter upplottning kan man med hjälp av musen bestämma olika punkters värden genom att trycka på en musknapp då markören befinner sig på önskad plats.

För att lämna programmet måste en musknapp tryckas ned då markören befinner sig i plottens röda område.

2.3 Resultat

Programmet styrs med hjälp av 17 olika kommandon till sampling eller efterbehandling av data. Förutom lagring av data består efterbehandlingen av grafisk presentation och snabb Fouriertransform (FFT). Användaren erbjuds 24 olika samplingsalternativ med varierande maximal samplingsfrekvens.

	Vektorsampling				Filsampling
	En signal		Två signaler		
	Ej PRBS	PRBS	Ej PRBS	PRBS	
Utan triggfunktion	0.170	0.170	0.190	0.240	30
Nivåtriggning utan buffring	0.170	0.175	0.200	0.270	30
Nivåtriggning med buffring	0.190	0.230	0.245	0.290	30

Tabell 2.1 Minimala samplingsperioden varierar med samplingsalternativet. Tiderna avser en Tandon AT och anges i ms.

Hur maximala samplingsfrekvensen varierar beroende på samplingsalternativ syns i Tabell 2.1. Ur tabellen kan utläsas att maximala samplingsfrekvensen varierar från 3 – 6 kHz vid vektorsampling. Notera att lagring av data i fil är så krävande att maximala samplingsfrekvensen är 33 Hz oberoende av samplingsalternativ.

3. Realtidskärnan

3.1 FB - schedulern

Realtidskärnan är skriven av Dag Brück. Kärnan är implementerad som en *foreground/background scheduler*, dvs kärnan klarar endast två processer. Nackdelen med en så pass enkel kärna är dess begränsade flexibilitet. Denna begränsning leder också till enkel *scheduling*, vilket tillåter högre samplingsfrekvens och därmed passar kärnan bra till vårt examensarbete. Vidare finns rutiner för AD- och DA-omvandlare tillgängliga i kärnan.

Kärnan är skriven i Microsoft Macro Assembler (MASM) och är avsedd för program skrivna i Microsoft C.

3.2 Förbättrad scheduler

För att bättre klara våra krav på frekvensområdets storlek och exakthet i samplingsfrekvens blev vi tvungna att göra en del förändringar i kärnan. Samplingsintervallet bestämdes från början i antal TICKS, där en TICK = 0.5 ms. För att förbättra upplösningen gjorde vi om konstanten TIMER_PER, som bestämmer storleken på en TICK, till en variabel som sätts vid igångsättningen av *schedulern*. Förgrundsprocessen exekverades nu varje TICK och vi fick en klart förbättrad frekvensupplösning. Tyvärr innebar denna metod att maximala samplingsintervallet blev mindre än 55 ms, dvs en minimal frekvens på ca 18 Hz. Bättre frekvensområde erhölls nu genom att återinföra variabeln PERIOD, som bestämmer hur många TICKS det skall gå mellan anrop av förgrundsprocessen. PERIOD realiserades dock som ett långt heltal utan tecken för att få ännu större frekvensområde.

Som användare av kärnan märks skillnaden endast genom anropet av *schedule*, som nu har tre parametrar istället för två. Anropet ser nu ut på följande sätt:

$$schedule(foreground, Timer\_Per, Period);$$

där *foreground* är adressen till den rutin som skall exekveras vid samplingen, *Timer_Per* bestämmer storleken på en TICK och *Period* anger hur många TICK det skall gå mellan exekveringen av *foregroundrutinen*. Det resulterande samplingsintervallet i sekunder ges av

$$\frac{Timer\_Per * Period}{1193180} \tag{3.1}$$

där 1 193 180 är hårdvaruklockans frekvens i Hz.

EXEMPEL 3.1
Anropet $schedule(f, 11932, 3)$ innebär att avbrott kommer att genereras med $\frac{11932}{1193180}$ s = 10 ms mellanrum, men f exekveras med ett samplingsintervall på 3*10 = 30 ms. □

3.3 Resultat

Omskrivningen av kärnan gav alltså två fördelar. Bättre upplösning av samplingsfrekvensen genom att kärnans TICK nu bestäms av en variabel istället för av en konstant, och större frekvensområde genom att den variabel som bestämmer antalet TICKS mellan exekvering av förgrundsprocessen nu upptar två ord. Maximala värdet på denna variabel är alltså 2^{32}. Detta ger ett maximalt samplingsintervall på $\frac{2^{16}*2^{32}}{1193180} \approx 2730$ dygn. (Med ett vanligt heltal utan tecken blir maximala samplingsintervallet ungefär en timme.)

4. Användarhandledning

4.1 Diskettinnehåll och installation

Trädstrukturen för disketten ser ut som följer:

```
A:\MODULA       \PLOT MOD
                \PLOT OBJ
                \PLOT REF
A:\ASM          \FB   ASM
                \FB   H
                \FB   OBJ
A:\C            \SRC        \SAMPLER  C
                            \SAMPLER  H
                            \FILESAVE C
                            \FILESAVE H
                            \COMM     C
                            \COMM     H
                            \FFT      C
                            \HELP     C
A:\C            \OBJ        \SAMPLER  OBJ
                            \FILESAVE OBJ
                            \COMM     OBJ
                            \FFT      OBJ
                            \HELP     OBJ
A:\EXE          \SAMPLER EXE
                \FFT     EXE
                \HELP    EXE
                \PLOT    EXE
A:\COMPSAMP BAT
A:\COMPALL  BAT
A:\LSAMP    BAT
A:\LALL     BAT
```

För att köra igång räcker det att ge följande DOS-kommandon:

```
COPY A:\EXE\*.*
SAMPLER
```

Hur kompilering och länkning av samplingsprogrammet skall gå till ges av COMPSAMP.BAT och LSAMP.BAT. COMPALL.BAT och LALL.BAT kompilerar respektive länkar alla fyra programmen.

4.2 Kommandospråket

Programmet startas genom att i DOS ge kommandot SAMPLER. Väl inne i programmets kommandodel syns *prompten* >. Genom att nu som första kommando ge HELP erhålls följande lista över tillgängliga kommando.

```
channel     ADchannel1 [ADchannel2]
delete      FileName
FFT         FileName
help        [command]
info        FileName
list        smp|inf|fft
NOPRBS
notrig
period      Sampleperiod
plot        [amplitude] [FFT]
PRBS        period meanvalue amplitude
quit
run
sample      NumberOfSamples [file]
save        [old]
show
trig        level [+|-] [buffer]
```

Kommandotolken klarar förkortade kommandon och är ej känslig för stora och små bokstäver , dvs quit, QUIT, QuIt, Q, qu ... är alla exempel på samma kommando. Alla alternativ inom klammer är valfria. Exempelvis är "help channel" ett kommando som ger vidare information om kommandot channel. På motsvarande sätt ger "help sample", "help PRBS" etc. ytterligare information om respektive kommando.

channel ADchannel1 [ADchannel2]

Med detta kommando bestämmer man vilka AD-kanaler som kommer att användas vid datainsamlingen. Om en eller två signaler skall samplas bestäms av antalet kanaler som ges. Observera att om två signaler skall samplas kan endast den ena av dessa triggas och den som skall triggas måste kopplas till den först givna AD-kanalen. De sex AD-kanalerna numreras 0 – 5.

EXEMPEL 4.1
CHANNEL 0 innebär att en signal samplas på AD-kanal 0 och denna signal kan, om så önskas, triggas. □

EXEMPEL 4.2
CHANNEL 4 1 innebär att två signaler samplas på AD-kanalerna 4 och 1. Den signal som eventuellt skall triggas måste anslutas till AD-kanal 4. □

delete FileName

FileName.smp, *FileName.inf* och *FileName.fft* raderas om de existerar.

FFT FileName

Om *FileName.inf* och *FileName.smp* existerar utförs FFT på de insamplade värden som lagrats i *FileName.smp*. Maximalt klaras 4096 värden.

	Vektorsampling				Filsampling
	En signal		Två signaler		
	Ej PRBS	PRBS	Ej PRBS	PRBS	
Utan triggfunktion	0.170	0.170	0.190	0.240	30
Nivåtriggning utan buffring	0.170	0.175	0.200	0.270	30
Nivåtriggning med buffring	0.190	0.230	0.245	0.290	30

Tabell 4.1 Minimala samplingsintervallet beror av fyra faktorer. Tiderna avser en Tandon AT och anges i ms.

help [command]

Ges kommandot utan alternativ listas alla tillgängliga kommandon. Dessa är sjutton till antalet och en närmre förklaring erhålls om respektive kommando ges som alternativ till helpkommandot.

info FileName

Detta kommando talar om ifall *FileName.smp*, *FileName.inf* respektive *FileName.fft* existerar. Om *FileName.inf* existerar så kommer den information som finns lagrad i denna fil att visas på skärmen.

list smp|inf|fft

Alla tillgängliga filer med respektive extension listas.

NoPRBS

PRBS-generatorn stängs av.

notrig

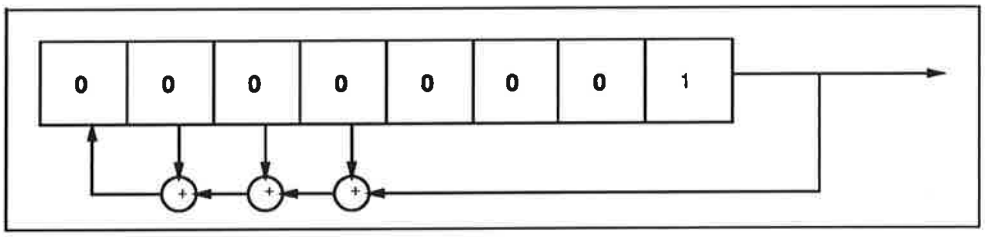
Ingen signal kommer att triggas.

Period SamplePeriod

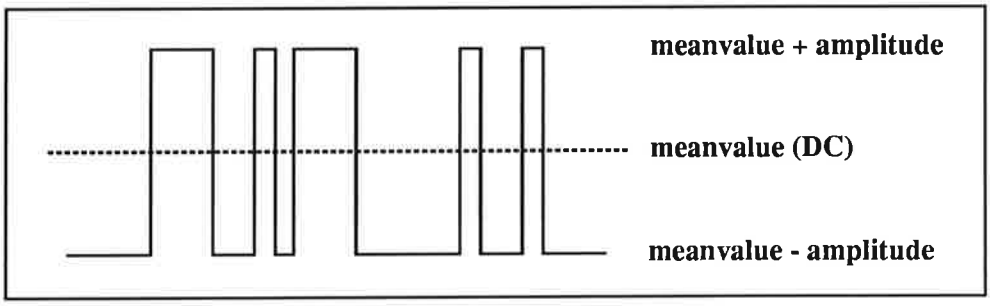
Med detta kommando bestäms samplingsintervallet i millisekunder. Minimal samplingsperiod varierar beroende på om:

- en eller två signaler skall samplas
- PRBS-generatorn är aktiv eller ej
- värdena lagras i en fil eller en vektor vid samplingen
- nivåtriggning med eller utan buffring har valts eller om triggning helt utelämnats

Hur minimala samplingsperioden mätt i ms varierar med dessa fyra faktorer syns i Tabell 4.1.



Figur 4.1 PRBS generatorn är realiserad som ett åtta bitars skiftregister.



Figur 4.2 PRBS generatorns utsignal.

plot [amplitude][FFT]

Plottar värden som lagrats i en fil. Den vertikala axeln skalas till $\pm amplitude$ om FFT alternativet utelämnas, annars skalas axeln från 0.01 till $10^{amplitude}$. *Amplitude* kan anta värdena 1 – 10 och om alternativet utelämnas sätts *amplitude* till 10. En fft fil plottas i ett loglog-diagram till skillnad från en sampelfil som plottas i ett linlin-diagram.

Om någon form av triggning har utförts är det alltid den röda signalplotten som motsvarar den triggade signalen.

Då själva plottningen är klar, kan man avläsa kurvorna genom att trycka på en musknapp då markören befinner sig på en kurva. Musens position visas då i plottens blå del.

För att lämna plotten och återgå till sampelprogrammet måste en musknapp tryckas in då markören befinner sig i plottens röda del. Om man av någon anledning vill avbryta plottningen innan själva plottningen är klar, tex. om ingen plottningsbar fil existerar, kan programmet lämnas med hjälp av CTRL-C.

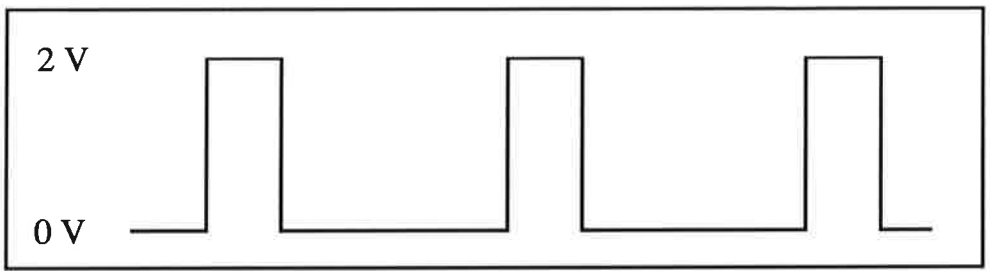
PRBS period meanvalue amplitude

Kommandot aktiverar PRBS-generatorn på DA-kanal 1. PRBS (Pseudo Random Binary Sequence) generatorn är realiserad som ett åttabitars skiftregister enligt Figur 4.1.

Initialt får skiftregistret det värde som syns i figuren. Utsignalen börjar om från början efter 255 högerskift. Efter $period * 255 * Ts$ sekunder börjar alltså signalen om från början igen, där *period* ges i kommandot och Ts = samplingsperioden i sekunder. Övriga parametrar i kommandot bestämmer signalens likspänning ut och den amplitud med vilken signalen varierar kring denna likspänning. Utsignalens utseende ges av Figur 4.2.

Observera att maximala amplituden, som ytterst begränsas av DA-omvandlarnas arbetsområde –10 V till 10 V, minskar med absolutbeloppet av likspänningen ut enligt följande formel.

$$maxamplitud = 10 - abs(meanvalue) \tag{4.1}$$



Figur 4.3 Under den tid som *foregroundrutinen* är aktiv är spänningen ut från DA-kanal 0 = 2 V. I övrigt ligger denna spänning alltid på noll. Detta kan nyttjas till att verifiera samplingsfrekvensen.

quit

Programmet avslutas.

run

Först kontrolleras att aktuellt samplingsintervall är korrekt jämfört med Tabell 4.1, och om så är fallet börjar programmet sampla in enligt användarens specifikationer.

OBS! Under sampling är programmet låst och nödstopp görs med hjälp av CTRL-BREAK ej CTRL-C.

OBS! Varje gång ett sampel tas läggs en spänning på två volt ut på DA-kanal noll. Se Figur 4.3

sample NumberOfSamples [file]

Antalet sampel som tas bestäms med hjälp av kommandot *sample*. Detta antal är begränsat till 32767 om de samplade värdena skall lagras i en vektor. Genom att ge kommandots andra alternativ kan fler värden samplas in, men eftersom värdena då lagras direkt i en fil blir maximala samplingsfrekvensen betydligt lägre.

save [old]

Om kommandot ges utan alternativ kontrolleras först att sampling har skett sedan programmet startades och om så är fallet tillåts man lagra insamplade värden i en fil, som man själv namnger. Med alternativ kan man välja ut och lagra en del av en gammal fil i en ny fil, tex. en speciellt intressant del som man vill titta närmre på vid plottning. Observera att detta inte fungerar för fft-filer.

show

Visar nuvarande parametrar och talar därmed om hur samplingen kommer att utföras om samplingen skulle göras nu.

trig level [+| −][buffer]

Nivåtriggning av signalen kopplad till "ADchannel1", enligt *channel* kommandot, erhålls med hjälp av detta kommando. Om triggning skall ske på positiv eller negativ flank bestäms genom att ge "+" eller "−". Utelämnas flankalternativet [+| −]kommer positiv flank att väljas. Ges *buffer*-alternativet buffras värden före triggpunkten.

4.3 Användarexempel
Insampling av värden från en tongenerator

Med risk att bli övertydliga följer nu ett detaljerat exempel, som steg för steg visar insampling av värden från en tongenerator. Alla kommandon vi ger avslutas med CR och står med kursiverad stil.

1) Starta datorn.
2) *login exempel*
3) Stoppa in disketten i diskettstationen.
4) *copy a : \exe\ * .**
5) Koppla in tongeneratorn på AD-kanal noll på baksidan av datorn.

   AD-kanalerna är de gröna kontakterna.

   Tongeneratorn jordas i den svarta kontakten längst till vänster sett bakifrån.
6) Ge följande kommandosvit

   sampler
   channel 0
   trig 6 *buffer*
   period 0.2
   sample 30000
7) För att kontrollera hur samplingen kommer att gå till ger vi nu kommandot *show*.
8) Ställ in tongeneratorn så att den genererar en fyrkantsvåg med grundtonen 10 Hz och amplituden 3 volt. Förvissa dig om hur mycket du behöver vrida upp amplituden för att den skall överstiga sex volt, dvs triggpunkten.
9) Koppla gärna ett oscilloskop till DA-kanal noll.

   DA-kanalerna är de röda kontakterna.
10) *run*, samplingen startas.
11) Vrid upp amplituden efter fem sekunder så att den säkert överstiger sex volt men understiger 10 volt.
12) När samplingen är klar får du svara på ett antal frågor.

    Ge följande svar:

    y
    exempel
    exempel
    4096
    50
13) Nu har drygt 2000 värden lagrats på vardera sidan om triggpunkten i en fil med namnet exempel.smp. Hur samplingen har gått till har lagrats i en fil med namnet exempel.inf. Kontrollera att filerna finns med kommandona *list smp* och *list inf*. För att få information om hur samplingen gått till ges nu kommandot *info exempel*.
14) För att plotta filen exempel.smp ges nu kommandot *Plot* med följande svar på frågorna som följer:

    exempel
    exempel

15) Markörens position visas nu vid tryckning på en musknapp.

16) Triggpunkten motsvarar tiden noll.

17) Studera plotten.

18) Lämna plotten genom att trycka på en musknapp då markören befinner sig det röda området.

19) Genom att nu ge kommandot *fft exempel* tillkommer ytterligare en fil med namnet exempel.fft. I denna fil ligger Fouriertransformen av "smp-filen".

20) För att plotta filen exempel.fft ges nu kommandot *Plot fft* med följande svar på frågorna som följer:
    exempel
    exempel

21) Studera plotten enligt punkterna 15, 17 och 18.

22) Avsluta genom att ge kommandot *quit*.

5. Utvärdering

5.1 Programmeringsspråket C

Eftersom vi aldrig förut programmerat i C blev vi först tvungna att bekanta oss med detta språk som, med Pascal som bakgrund, var ganska lätt att lära sig. Därmed inte sagt att vi till fullo behärskar alla de finesser och fällor som C "erbjuder". Vad fällor beträffar är de inte så få, men med guider som "C Traps and Pitfalls" (Koenig , 1989) kan de undvikas.

Själva programmeringen i C har gett både positiva och negativa erfarenheter.

+ Filhanteringen är mycket smidig.
+ Enkelt att typkonvertera.
- Kompilatorn släpper igenom alltför mycket. Detta leder till att väldigt enkla fel, som tex. felaktigt antal aktuella parametrar, blir svårfunna.
- Pascals "var" parametrar saknar vi.
- Pekare som visserligen har stora möjligheter men som är mer svårbegripliga än Pascals pekare.

5.2 Speciella problem

Plottning

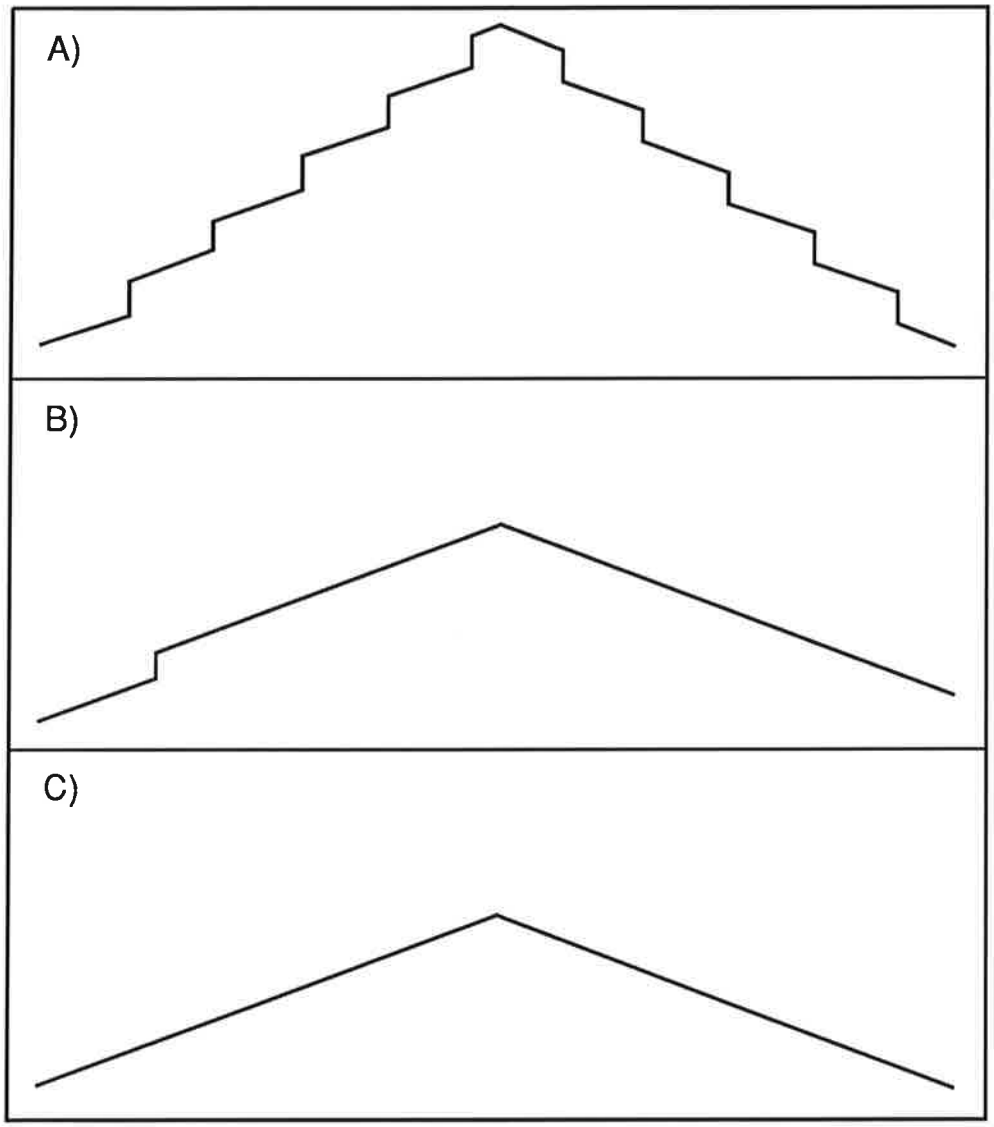
Från början hade vi tänkt att programmet skulle kunna köras inifrån Matlab och sedan skulle grafisk presentation ske med hjälp av egna funktioner skrivna för Matlab.

Snart visade sig programmet dock bli för stort för att kunna köras inifrån Matlab. Trots detta fortsatte vi att utnyttja Matlab för den grafiska presentationen, eftersom vi inte hade någon annan metod för att kontrollera de insamlade värdena. Förutom att plottningen nu blev en omständig process visade det sig att Matlab inte klarade att plotta tillräckligt stora filer. När vi sedan dessutom kom på att samplingsfrekvensen blev instabil efter ett besök i Matlab blev vi tvungna att helt överge denna metod.

Vi fick nu skriva ett separat plottningsprogram i Modula-2. Detta innebar att vi fick bekanta oss med ännu ett nytt språk och ännu en realtidskärna innan vi kunde fortsätta.

Instabil samplingsfrekvens

Genom att utnyttja synksignalen på DA-kanal 0 kunde vi med hjälp av ett oscilloskop med frekvensräknare bestämma maximala samplingsfrekvensen vid de olika samplingsalternativen. Vid bestämmandet av dessa frekvenser kom vi på att samplingsfrekvensen blev något mer instabil efter ett besök i Matlab. Då vi försökte att finna någon förklaring till detta fenomen upptäckte vi att samma sak inträffade om vi istället använde oss av Simnon mellan två starter



Figur 5.1 Filsampling av triangelvåg. A) Med buffert på 512 Byte. B) Utan buffert. C) Utan buffert och med tömmning av bufferten en gång.

av samplingsprogrammet, men vad det är som orsakar denna instabilitet har vi inte lyckats komma fram till. För att komma ifrån instabiliteten måste man starta om maskinen.

Ett annat fenomen är att samplingsfrekvensens stabilitet varierar med olika maskiner. Det bör observeras att variationerna dock är väldigt små.

Filsampling

Som framgår av Figur 5.1 hade vi problem med samplingsfrekvensens stabilitet även vid filsampling. Med filsampling menar vi att de insamlade värdena direkt lagras i en temporär fil, *tempfile.tmp*, istället för i en vektor. Genom att studera plotten av en samplad triangelvåg kunde vi se en klar periodicitet hos samplingsfelet. Redigering av *tempfile.tmp* visade distinkta "hopp" i de insamlade värdena med 512 bytes mellanrum. Eftersom utskriftsrutinen *fprintf* normalt arbetar med en buffert på 512 byte var det inte svårt att misstänka denna rutin. Nästa steg blev nu att nollställa denna buffert med anropet $setbuf(fpek, NULL)$, så att ett insamlat värde direkt skrevs ut på

filen. Tyvärr visade det sig att frekvensen blev stabil först efter insampling av 512 byte.

På något sätt verkar det som att *setbuf(fpek, NULL)* ej gör sig gällande förrän efter första tömningen av den ursprungliga bufferten på 512 byte. För att även kringgå detta problem blev vi tvungna att först skriva ut drygt 512 byte på *tempfile.tmp* och sedan återföra filpekaren till filens början, innan samplingen startas. För själva implementeringen hänvisas till proceduren *sampler* i Appendix A.

Från början var tanken att filsampling skulle utnyttjas vid insampling av "obegränsat" många värden. Med tanke på att vektorsamplingen tillåter insampling av drygt 32000 värden, och att man med filer på 30000 eller mer snabbt fyller hårddisken, får man ändå en begränsning av de "obegränsade" antalet värden som kan samplas in. Om man dessutom påminner sig att maximala samplingsfrekvensen vid filsampling är 33 Hz kan man fråga sig, om implementeringen av filsampling egentligen lönat sig.

Separata C-program

När vi kommit ganska långt med implementeringen av samplingsprogrammet uppstod plötsligt "oförklarliga" fel. De parametrar som matats in stämde inte längre efter insampling. Felet uppstod efter smärre tillägg av den text som används för att kommunicera med användaren. Genom att ta bort den tillagda texten fungerade programmet igen. Eftersom text som skrivs ut med *fprintf* lägger sig som konstanta data kom vi fram till att dataarean möjligen inte räckte till. För att kunna behålla den dataarea på 64 KB, som den lilla minnesmodellen av Microsoft C erbjuder, flyttade vi ut alla hjälptexter till ett separat program Help.C. På detta sätt slapp vi ytterligare ändringar av schedulern som är skriven för den lilla minnesmodellen.

När vi sedan skulle tillföra FFT fick av samma anledning även detta göras med hjälp av ett separat program.

Programmet låser sig

Det har visat sig att programmet låser sig efter ett slumpartat antal samplingar. Den enda förklaring vi kunnat se skulle vara att det beror på den för tillfället aktuella stackens storlek. Problemet med vilken stack som är aktiv omnämns även i implementeringen av *schedulern*[1]. Om *schedulern* skrevs om så att man med säkerhet visste vilken stack som nyttjas vid sampling skulle antagligen detta problem lösas.

5.3 Önskvärda förändringar och utvidgningar

Eftersom antalet värden som kan samplas in vid vektorsampling är tillräckligt stort i de flesta fall, och maximala samplingsfrekvensen vid filsampling är påfallande låg, kan man överväga att helt slopa filsamplingen.

Det vore trevligt om programmet utökades med PID-reglering samt möjlighet att sampla flera signaler. PID-regleringen skulle kunna tillföras enligt samma mönster som PRBS-generatorn[2]. Om sampling av fler signaler än två skall möjliggöras bör nog vektorernas storlek minskas till åtminstone hälften.

---

[1] Se Appendix D

[2] Se Appendix A

Den enklaste utvidgningen vore att tillföra programmet någon form av filtrering som efterbehandling enligt samma princip som vi har tillfört programmet FFT.

6. Referenser

ERIC ASTOR (1986): *Från Pascal till Modula-2*, Studentlitteratur, Lund.

KRIS A. JAMSA (1988): *Programmera i C*, Columna Förlags AB, Stockholm.

ANDREW KOENIG (1989): *C Traps and Pitfalls*, Addison-Wesley Publishing Company, USA.

LENNART OLBJER (1985): *Tidsserieanalys*, Avdelningen för matematisk statistik, Lunds Universitet och Lunds Tekniska Högskola, Lund.

RICHARD C. SINGLETON (1967): "A Method for Computing the Fast Fourier Transform with Auxiliary Memory and Limited High-Speed Storage," *IEEE Transactions on Audio and Electroacoustics*, **15**, 91–97.

Appendix A
Programlista för sampler.c

```
#include <stdio.h>
#include "fb.h"
#include "comm.h"


#define HARDWARE_CLOCK 1193180  /* Frequency of the hardware
                                   clock in Hz. */
#define MAX_CHANNEL_NUMBER 5
#define MAX_NO_OF_SAMPLES 32767 /* 2^15 -1 */
#define MAXVALUE 2147483647     /* 2^31 -1 */
#define FALSE 0
#define TRUE 1


long Read_Index,Buff_Index;
long No_Of_Buff,No_Of_Samples;

long Period;   /* See also PEERIOD in the FB scheduler   */
int Timer_Per; /* See also TIMER_PER in the FB scheduler */

int ADch1,ADch2,Trigchoice;
int Triglevel,temp1,temp2,Oldvalue;
int Filesample,Trigpoint_not_reached,TwoSignals;/* Used as booleans */

int huge Sampvec1[MAX_NO_OF_SAMPLES];
int huge Sampvec2[MAX_NO_OF_SAMPLES];
int huge Buffvec1[MAX_NO_OF_SAMPLES];
int huge Buffvec2[MAX_NO_OF_SAMPLES];

int count,minperiod,Imean,Iamp,shift,u;
int PRBS; /* Used as boolean */
float mean,amp;

float Td,Ts;  /* Desired sample period and actual sample period */

float Tdmin;

void (*foreground)();

FILE *fpek,*fopen();
```

```
/*************************************************************************
 ****************************** PRBS *************************************

 This Pseudo Random Binary Sequence generator can be called by all the
 foreground routines below. The generated signal is either Imean + Iamp
 or Imean - Iamp volts high. The generator is nothing but an eight bit
 shift register with the period 255. This 255 - bitperiod is manipulated
 so that the actual period is n * 255, where n = minperiod.

 The signal is available at D/A - channel 1.

 *************************************************************************/


#pragma check_stack-

void PRBS_Gen()
  {
   register int x;
   register int Tshift=shift;

   if (count++>=minperiod)
     {
      u=((Tshift&1) > 0) ? Imean+Iamp:Imean-Iamp;
      x=((Tshift&64) > 0) ? 128:0;
      if ((Tshift&32) > 0) x^=128;
      if ((Tshift&16) > 0) x^=128;
      if ((Tshift&1) > 0) x^=128;
      Tshift>>=1;
      Tshift^=x;
      shift=Tshift;
      count=1;
     }
   DAout(1,u);
  }
#pragma check_stack+


/*************************************************************************/
/******************** DIFFERENT FOREGROUND ROUTINES **********************/
/********************     USED BY SAMPLE BLOCK      **********************/
/*************************************************************************/


/*************************************************************************
 ********************** NotrigVector_foreground() ************************

 The sampled values are stored in a vector, this means that the number
 of samples is maximiced to 32767. On the other hand it allows higher
 sampling frequency compared to filesampling.

 This is the simplest foreground.

 If PRBS and two signals are chosen then Tdmin = 0.232 ms.
 Else if two signals are chosen then     Tdmin = 0.181 ms.
 Else if PRBS is chosen then             Tdmin = 0.170 ms.
 Else                                    Tdmin = 0.160 ms.

 *************************************************************************/
```

```
#pragma check_stack-

void NotrigVector_foreground()

  {
   if (Read_Index<No_Of_Samples)
    {
     if (PRBS) PRBS_Gen();
     if (TwoSignals) Sampvec2[Read_Index]=ADin(ADch2);
     Sampvec1[Read_Index++]=ADin(ADch1);
    }
   else
     schedule(foreground,0,0);
  }
#pragma check_stack+




/*************************************************************************
 ********************** NotrigFiler_foreground() ************************

 The sampled values are stored in a file, this means that the number
 of samples isn't maximiced but max sampling frequency is lower than
 in NotrigVector_foreground() above.

 This is the simplest foreground where the sampled values are stored in
 a file.

 Tdmin = 30 ms.

 *************************************************************************/


#pragma check_stack-

void NotrigFile_foreground()

  {
   if (Read_Index++<No_Of_Samples)
    {
     if (PRBS) PRBS_Gen();
     if (TwoSignals) fprintf(fpek,"%d    ",ADin(ADch2));
     fprintf(fpek,"%d\n",ADin(ADch1));
    }
   else
     schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** PosLevVector_foreground() ************************

 The sampled values are stored in a vector.

 This foreground is somewhat slower than NotrigVector_foreground()
 since the sampling is triggered by a level on positive slope.

 If PRBS and two signals are chosen then Tdmin = 0.24  ms.
 Else if two signals are chosen then     Tdmin = 0.19  ms.
 Else if PRBS is chosen then             Tdmin = 0.175 ms.
 Else                                    Tdmin = 0.170 ms.

 *************************************************************************/

#pragma check_stack-

void PosLevVector_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) Sampvec2[Read_Index]=ADin(ADch2);
      Sampvec1[Read_Index]=temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
          if (temp1>Triglevel && Oldvalue<Triglevel)
            {
             Trigpoint_not_reached=0; /* not true, the
                                         trigpoint is reached */
             Read_Index++;
            }

          Oldvalue=temp1;
        }
      else
         Read_Index++;
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** NegLevVector_foreground() ***********************

 The sampled values are stored in a vector.

 This foreground is somewhat slower than NotrigVector_foreground()
 since the sampling is triggered by a level on negative slope.

 If PRBS and two signals are chosen then Tdmin = 0.24  ms.
 Else if two signals are chosen then     Tdmin = 0.19  ms.
 Else if PRBS is chosen then             Tdmin = 0.175 ms.
 Else                                    Tdmin = 0.170 ms.

 *************************************************************************/


#pragma check_stack-

void NegLevVector_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) Sampvec2[Read_Index]=ADin(ADch2);
      Sampvec1[Read_Index]=temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
          if (temp1<Triglevel && Oldvalue>Triglevel)
            {
             Trigpoint_not_reached=0; /* not true, the
                                         trigpoint is reached */
             Read_Index++;
            }

          Oldvalue=temp1;
        }
      else
         Read_Index++;
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** PosLevBuffer_foreground() ***********************

 The sampled values are stored in a vector.

 This foreground is somewhat slower than PosLevVector_foreground() since
 those values that are sampled before the trigpoint are bufferred in
 another vector.

 If PRBS and two signals are chosen then Tdmin = 0.290 ms.
 Else if two signals are chosen then     Tdmin = 0.215 ms.
 Else if PRBS is chosen then             Tdmin = 0.191 ms.
 Else                                    Tdmin = 0.180 ms.

 *************************************************************************/


#pragma check_stack-

void PosLevBuffer_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) temp2=ADin(ADch2);
      temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
         if(temp1>Triglevel && Oldvalue<Triglevel)
           {
            Trigpoint_not_reached=0; /* not true, the
                                        trigpoint is reached */
            if (TwoSignals) Sampvec2[Read_Index]=temp2;
            Sampvec1[Read_Index++]=temp1;
           }
         else
           {
            if (TwoSignals) Buffvec2[Buff_Index]=temp2;
            Buffvec1[Buff_Index++]=temp1;
            No_Of_Buff++;
           }

         if (Buff_Index == MAX_NO_OF_SAMPLES)
            Buff_Index=0;

         Oldvalue=temp1;
        }
      else
        {
         if (TwoSignals) Sampvec2[Read_Index]=temp2;
         Sampvec1[Read_Index++]=temp1;
        }
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** NegLevBuffer_foreground() ***********************

 The sampled values are stored in a vector.

 This foreground is somewhat slower than NegLevVector_foreground() since
 those values that are sampled before the trigpoint are bufferred in
 another vector.

 If PRBS and two signals are chosen then Tdmin = 0.290 ms.
 Else if two signals are chosen then     Tdmin = 0.215 ms.
 Else if PRBS is chosen then             Tdmin = 0.191 ms.
 Else                                    Tdmin = 0.180 ms.

 *************************************************************************/


#pragma check_stack-

void NegLevBuffer_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) temp2=ADin(ADch2);
      temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
         if(temp1<Triglevel && Oldvalue>Triglevel)
           {
            Trigpoint_not_reached=0; /* not true, the
                                        trigpoint is reached */
            if (TwoSignals) Sampvec2[Read_Index]=temp2;
            Sampvec1[Read_Index++]=temp1;
           }
         else
           {
            if (TwoSignals) Buffvec2[Buff_Index]=temp2;
            Buffvec1[Buff_Index++]=temp1;
            No_Of_Buff++;
           }

         if (Buff_Index == MAX_NO_OF_SAMPLES)
            Buff_Index=0;

         Oldvalue=temp1;
        }
      else
        {
         if (TwoSignals) Sampvec2[Read_Index]=temp2;
         Sampvec1[Read_Index++]=temp1;
        }
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** PosLevFile _foreground() ************************

 The sampled values are stored in a file.

 If you have chosen to buffer values ( <=> Trigchoice > 2 ) then
 those values that are sampled before the trigpoint are bufferred in
 the same file. No_Of_Buff (= number of bufferred values) makes it
 possible to keep track of the trigpoint.

 Tdmin = 30 ms.

 *************************************************************************/


#pragma check_stack-

void PosLevFile_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) temp2 = ADin(ADch2);
      temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
          if (temp1>Triglevel && Oldvalue<Triglevel)
            {
             Trigpoint_not_reached=0; /* not true,
                                         the trigpoint is reached */
             if (TwoSignals) fprintf(fpek,"%d    ",temp2);
             fprintf(fpek,"%d\n",temp1);
             Read_Index++;
            }
          else if (Trigchoice > 2)
            {
             if (TwoSignals) fprintf(fpek,"%d    ",temp2);
             fprintf(fpek,"%d\n",temp1);
             No_Of_Buff++;
            }

          Oldvalue=temp1;
        }
      else
        {
         if (TwoSignals) fprintf(fpek,"%d    ",temp2);
         fprintf(fpek,"%d\n",temp1);
         Read_Index++;
        }
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************
 ********************** NegLevFile _foreground() ************************

 The sampled values are stored in a file.

 If you have chosen to buffer values ( <=> Trigchoice > 2 ) then
 those values that are sampled before the trigpoint are bufferred in
 the same file. No_Of_Buff (= number of bufferred values) makes it
 possible to keep track of the trigpoint.

 Tdmin = 30 ms.

 *************************************************************************/

#pragma check_stack-

void NegLevFile_foreground()

  {
   if (Read_Index<No_Of_Samples)
     {
      if (PRBS) PRBS_Gen();
      if (TwoSignals) temp2 = ADin(ADch2);
      temp1=ADin(ADch1);

      if (Trigpoint_not_reached)
        {
          if (temp1<Triglevel && Oldvalue>Triglevel)
            {
             Trigpoint_not_reached=0; /* not true, the
                                         trigpoint is reached */
             if (TwoSignals) fprintf(fpek,"%d    ",temp2);
             fprintf(fpek,"%d\n",temp1);
             Read_Index++;
            }
          else if (Trigchoice > 2)
            {
             if (TwoSignals) fprintf(fpek,"%d    ",temp2);
             fprintf(fpek,"%d\n",temp1);
             No_Of_Buff++;
            }

          Oldvalue=temp1;
        }
      else
        {
         if (TwoSignals) fprintf(fpek,"%d    ",temp2);
         fprintf(fpek,"%d\n",temp1);
         Read_Index++;
        }
     }
   else
      schedule(foreground,0,0);
  }
#pragma check_stack+
```

```
/*************************************************************************/
/************************ SAMPLE BLOCK ***********************************/
/*************************************************************************/

/*************************************************************************
 ******************************** Init ***********************************

 This procedure is used to initialize states before sampling.

 Oldvalue = Triglevel => If we choose to let the sampling be triggered
 by signal level, the trigpoint can't be reached the first time the
 foreground is invoked.

 *************************************************************************/

void Init()

  {
   Read_Index=Buff_Index=0;
   No_Of_Buff=0;

   Trigpoint_not_reached=TRUE; /* Used as boolean */

   shift=1;
   count=minperiod;

   Oldvalue=Triglevel;
  }
```

```
/*************************************************************************
 ***************************** sampler() *********************************

 This procedure uses Init before starting the foreground/background
 scheduler.

 Furthermore it uses schedule and reset from the foreground/background
 scheduler, and therefore the following file schould be inluded:
    - fblp.h

 *************************************************************************/

void sampler()

  {
   int i;

   /* Prepare for sampling */


   if (Td != 0)
     {
      if (Filesample)
        {
         fpek=fopen("tempfile.tmp","w");
         setbuf(fpek,NULL);
         for(i=1;i<8;i++)
          fprintf(fpek,"****************************** FILL BUFFER
                        ******************************\n");
         rewind(fpek);
        }

      Init();

      /* Start and lock the program to the simpel whileroutine */
      /* below until we have sampled enough.     */

      schedule(foreground,Timer_Per,Period);
      while (Read_Index<No_Of_Samples)
        ;
      reset();

      if (Filesample)
         fclose(fpek);
     }
  }



/*************************************************************************/
/************************** MAIN BLOCK ***********************************/
/*************************************************************************/


main()
{
 system("cls");
 Start_Up_Values();
 command();
}
```

Appendix B
Programlista för comm.c

```
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include "sampler.h"
#include "filesave.h"



/*************************************************************************
 ********************** Option_Error(Errortext) ************************

 This procedure is used to produce an errormessage whenever a false
 command has been given.

 *************************************************************************/

void Option_Error(Errortext)
   char *Errortext;
  {
   printf("\n%s\n",Errortext);
  }


/*************************************************************************
 ************************** index(strng,del) *************************

 If the string strng contains del then the startposition of del in strng
 is returned, else -1 is returned.

 *************************************************************************/

int index(strng,del)
  char strng[];
  char del[];
{
  int aktuell = 0;
  int j,k;

  while (strng[aktuell])
   {
    j = aktuell;
    k = 0;

    while (strng[j++] == del[k])
      if (del[++k] == '\0') return(aktuell);
    aktuell++;
   }
  return(-1);
}
```

```
/*************************************************************************
 ************** check_range(prompt,value,min,max) ********************

 Checks if value is in the range min - max, if not then an errormessage
 is given. Returns 0 (FALSE) if value is out of range, else 1 (TRUE).

 *************************************************************************/

int check_range(prompt,value,min,max)
   char *prompt;
   float value,min,max;
  {
   if (value < min || value > max)
    {
     printf("\n%s%g isn't in the range (%g .. %g)\n",
                                  prompt,value,min,max);
     return(FALSE);
    }
   else return(TRUE);
  }


/*************************************************************************
 ********************** check_command(s) *********************************

 If s is a valid command then the commandnumber CNO is returned, else
 -1 is returned.

 *************************************************************************/


int check_command(s)
char *s;
  {
   int CNO = -1;

   if      (index(s,"nop") != -1 && index("noprbs",s) != -1) CNO = 1;
   else if (index(s,"not") != -1 && index("notrig",s) != -1) CNO = 2;
   else if (index(s,"q")   != -1 && index("quit",s)   != -1) CNO = 3;
   else if (index(s,"r")   != -1 && index("run",s)    != -1) CNO = 4;

   else if (index(s,"sh")  != -1 && index("show",s)   != -1) CNO = 5;
   else if (index(s,"h")   != -1 && index("help",s)   != -1) CNO = 6;
   else if (index(s,"sav") != -1 && index("save",s)   != -1) CNO = 7;
   else if (index(s,"pl")  != -1 && index("plot",s)   != -1) CNO = 8;

   else if (index(s,"f")   != -1 && index("fft",s)    != -1) CNO = 9;
   else if (index(s,"i")   != -1 && index("info",s)   != -1) CNO = 10;
   else if (index(s,"l")   != -1 && index("list",s)   != -1) CNO = 11;
   else if (index(s,"pe")  != -1 && index("period",s) != -1) CNO = 12;

   else if (index(s,"c")   != -1 && index("channel",s)!= -1) CNO = 13;
   else if (index(s,"sam") != -1 && index("sample",s) != -1) CNO = 14;
   else if (index(s,"t")   != -1 && index("trig",s)   != -1) CNO = 15;
   else if (index(s,"pr")  != -1 && index("prbs",s)   != -1) CNO = 16;

   else if (index(s,"d")   != -1 && index("delete",s) != -1) CNO = 17;

   return(CNO);
  }
```

```
/*************************************************************************
 ********************** show_parameters() ****************************

 Used by the command procedure when the show command has been given
 with correct number of options.

 *************************************************************************/

void show_parameters()

  {
   if (TwoSignals)
     {
      printf("\nTwo signals are sampled.");
      printf("\n   Triggable A/D channel = %d\n   Not
                  triggable A/D channel = %d",ADch1,ADch2);
     }
   else
     {
      printf("\nOne signal is sampled.");
      printf("\n   Triggable A/D channel = %d",ADch1);
     }

   printf("\n\nNumber of samples = %ld",No_Of_Samples);
   if (Filesample)
     printf("\n\nThe sampled values are stored in a file.");
   else
     printf("\n\nThe sampled values are stored in a vector.");

   if (PRBS)
     {
      printf("\n\nThe PRBS generator is active on D/A channel number one.");
      printf("\n    period = %d\n    meanvalue = %2.2f\n    amplitude =
   %2.2f",minperiod,mean,amp);
     }

   if (Trigchoice == 0)
      printf("\n\nNo trigfunction is activated.");
   else if (Trigchoice == 1 || Trigchoice == 3)
      printf("\n\nThe signal connected to Triggable A/D channel will
                  be triggered on positive\nslope.");
   else if (Trigchoice == 2 || Trigchoice == 4)
      printf("\n\nThe signal connected to Triggable A/D channel will
                  be triggered on negative\nslope.");
   if (Trigchoice == 3 || Trigchoice == 4)
      printf(" Values sampled before the trigpoint will be buffered.");
   if (Trigchoice > 0)
      printf("\n\n   Triglevel = %2.2f",10.0*(float)(Triglevel)/2047.0);

   printf("\n\nSample period = %f s",Ts);
   if (Ts == 0) printf(", this means that no sampling is done.");
   else printf(".\n\nActual sample frequency = %5.2f Hz.",1/Ts);
   printf("\n");
  }
```

```
/*************************************************************************
 ********************** PutHelpFile(x,strng) ****************************

 Writes the integer value x and the string strng into the file
 HelpFile.dat. If x < 20 then the external program Help is started else
 the external program FFT is started. These programmes are told what to
 do through HelpFile.dat.

 *************************************************************************/


void PutHelpFile(x,strng)
int x;
char *strng;
{
  FILE *p,*fopen();

  p = fopen("HelpFile.dat","w");
    fprintf(p,"%d\n",x);
    fprintf(p,"%s\n",strng);
  fclose(p);
  if (x < 20) system("Help");
  else system("fft");
}

/*************************************************************************
 ***************************** help(s,n) ********************************

 Used by the command procedure when the help command has been given
 with correct number of options.
 If no option is given then all available commands will be displayed,
 else the specified command will be explained. In order to give this
 information help uses PutHelpFile above.

 *************************************************************************/

void help(s,n)
   int  n;
   char *s;
  {
   int CNO;

   if (n == 0)
     PutHelpFile(0," ");
   else
     { CNO = check_command(strlwr(strdup(s)));
       if (CNO == -1) Option_Error("Invalid option.");
       else           PutHelpFile(CNO," ");
     }
  }
```

```
/*************************************************************************
 ********************** plot(s1,s2,n) ********************************

 Used by the command procedure when the plot command has been given with
 correct number of options. If the options are correct then this pro-
 cedure instructs the external program plot what to do through the help
 of PlotFile.dat. The procedure uses the system call to start the plot
 program.

 *************************************************************************/

void plot(s1,s2,n)
   char *s1,*s2;
   int n;
  {
   int Ok = TRUE;
   int PutVal,x,F;
   FILE *p,*fopen();

   x = atoi(s1);
   if (n == 1) F = strcmpi(s1,"FFT");
   else        F = strcmpi(s2,"FFT");

   if (n == 0 || (n == 1 && strcmpi(s1,"FFT") == 0))
     x = 10;
   else if (n == 1 || (n == 2 && strcmpi(s2,"FFT") == 0))
     Ok = check_range("Amplitude = ",(float)(x),1.0,10.0);
   else
     Ok = FALSE;

   if (Ok == TRUE)
     {
      PutVal = 10000 + x*100;
      if (F == 0) PutVal = PutVal + 1;
      p = fopen("PlotFile.DAT","w");
      fprintf(p,"%d",PutVal);
      fclose(p);
      system("plot");
     }
   else Option_Error("Invalid option");
  }
```

```
/*************************************************************************
 ********************** save(s,HaveSampled,n) ***************************

 *************************************************************************/

void save(s,HaveSampled,n)
   char  *s;
   int   HaveSampled,n;
  {
   if (n == 0 && HaveSampled)
     FileChoice(No_Of_Buff,No_Of_Samples,Buff_Index,TwoSignals,
                Trigchoice,Filesample,Ts);
   else if (n == 0 && (HaveSampled == FALSE))
     Option_Error("You haven't sampled anything");
   else if (n == 1 && (strcmpi(s,"old") == 0))
     OldChoice();
   else if (n == 1)
     Option_Error("Invalid option");
  }


/*************************************************************************
 ************************** Start_Up_Values() ***************************

 Used by the main program to initiate parameters.

*************************************************************************/

void Start_Up_Values()
  {
   PRBS = Trigchoice = ADch1 = TwoSignals = Filesample = Td = Ts = 0;
   No_Of_Samples = 0;
   ADch2 = 1;
   Tdmin = 0.16;
  }



/*************************************************************************
 *********** FVforeground(File_foreground,Vector_foreground) ************

 Different foregrounds should be invoked depending on wether the sampled
 values should be stored in a file or a vector.

 *************************************************************************/

void FVforeground(File_foreground,Vector_foreground)

   void (*File_foreground)(),(*Vector_foreground)();
  {
   if (Filesample)
     foreground=File_foreground;
   else
     foreground=Vector_foreground;
  }
```

```
/*************************************************************************
 ******************** Minper(Filemin,Vectormin) ************************

 Different foreground routines will be invoked depending on wether
 the sampled values should be stored in a vector or a file.

 Different foregrounds => different Tdmin.

 Tdmin also varies depending on choice of TwoSignals,PRBS or both.

 ( Tdmin = minimal sampling period )

 *************************************************************************/


float Minper(Filemin,Vectormin,PRBSmin,TWOmin,TWOPRBSmin)

   float Filemin,Vectormin,PRBSmin,TWOmin,TWOPRBSmin;

  {
   if (Filesample)
     return(Filemin);
   else if (PRBS && TwoSignals)
     return(TWOPRBSmin);
   else if (TwoSignals)
     return(TWOmin);
   else if (PRBS)
     return(PRBSmin);
   else
     return(Vectormin);
  }
```

```
/*************************************************************************
 ********************************* Trig()  ******************************

 The procedure uses Minper to determine minimal sampling period (Tdmin).
 Different foregrounds should be invoked depending on Trigchoice =>
   The parameters sent to Minper varies with Trigchoice.

 Furthermore it uses FVforeground so that the sampler will invoke the
 right foreground routine.

 *************************************************************************/

void Trig(choice)
   int choice;
  {
   switch(choice)
    {
     case 0:{Tdmin=Minper(30.0,0.17,0.17,0.19,0.24);
             FVforeground(NotrigFile_foreground,NotrigVector_foreground);
             break;
            }
     case 1:{Tdmin=Minper(30.0,0.17,0.175,0.2,0.27);
             FVforeground(PosLevFile_foreground,PosLevVector_foreground);
             break;
            }
     case 2:{Tdmin=Minper(30.0,0.17,0.175,0.2,0.27);
             FVforeground(NegLevFile_foreground,NegLevVector_foreground);
             break;
            }
     case 3:{Tdmin=Minper(30.0,0.19,0.23,0.245,0.29);
             FVforeground(PosLevFile_foreground,PosLevBuffer_foreground);
             break;
            }
     case 4:{Tdmin=Minper(30.0,0.19,0.23,0.245,0.29);
             FVforeground(NegLevFile_foreground,NegLevBuffer_foreground);
             break;
            }
    }
  }


/*************************************************************************
 ********************** read_channel(c1,c2,n) ****************************

 Used by the command procedure when the channel command has been given
 with correct number of options.
 Gives the triggable A/D channel (ADch1) the value of c1 and the other
 A/D channel (ADch2) the value of c2. If two channels are given then two
 signals will be sampled.

 *************************************************************************/
```

```
void read_channel(c1,c2,n)
   int c1,c2,n;
  {
   int Ok;
   Ok = check_range("ADch1 = ",(float)(c1),0.0,
                       (float)(MAX_CHANNEL_NUMBER));

   if (n == 2 && c2 != c1)
     Ok = Ok*check_range("ADch2 = ",(float)(c2),0.0,
                           (float)(MAX_CHANNEL_NUMBER));
   else if (n == 2)
     { Option_Error("Error: ADch2 = ADch1");
       Ok = FALSE;
     }

   if (Ok) { ADch1 = c1; TwoSignals = FALSE;
             if (n == 2) { TwoSignals = TRUE; ADch2 = c2; }
             Trig(Trigchoice);
           }
  }


/*************************************************************************
 ********************** read_sample(x,f,n) ****************************

 Used by the command procedure when the sample command has been given
 with correct number of options.

 *************************************************************************/

void read_sample(x,f,n)
   long x;
   int f,n;
  {
   int Ok;

   if (n == 2 && f == 0)
     Ok = check_range("Number of samples = ",(float)(x),0.0,
                         (float)(MAXVALUE));
   else if (n == 2)
     { Option_Error("Invalid option.");
       Ok = FALSE;
     }
   else
      Ok = check_range("Number of samples = ",(float)(x),0.0,
                         (float)(MAX_NO_OF_SAMPLES));

   if (Ok) { No_Of_Samples = x; Filesample = FALSE;
             if (n == 2) Filesample = TRUE;
             Trig(Trigchoice);
           }
  }
```

```
/*************************************************************************
 ************************ read_PRBS(x,y,z) ****************************

 Used by the command procedure when the PRBS command has been given
 with correct number of options.

 *************************************************************************/


void read_PRBS(x,y,z)
   int x;
   float y,z;
  {
   int Ok;

   Ok = check_range("PRBS period = ",(float)(x),1.0,32767.0);
   Ok = Ok * check_range("PRBS meanvalue = ",y,-10.0,10.0);
   if (mean < 0)
     Ok = Ok * check_range("PRBS amplitude = ",z,0.0,10.0+mean);
   else
     Ok = Ok * check_range("PRBS amplitude = ",z,0.0,10.0-mean);

   if (Ok)
     { PRBS      = TRUE;
       minperiod = x;
       mean      = y;
       Imean     = (int)(2047*mean/10 + 0.5);
       amp       = z;
       Iamp      = (int)(2047*amp/10 + 0.5);
       Trig(Trigchoice);
     }
  }
```

```
/*************************************************************************
 ********************* read_trig(s1,s2,s3,n) ***************************

 Used by the command procedure when the trig command has been given
 with correct number of options.

 *************************************************************************/

read_trig(s1,s2,s3,n)
   char *s1,*s2,*s3;
   int n;
  {
   int TC = 0;
   int b;
   char *ch = "+";
   float x;

   x = atof(s1);
   b = strcmpi(s3,"buffer");

   if (check_range("Triglevel = ",x,-10.0,10.0))
    {
     if (n == 1 || (n == 2 && strcmpi(s2,"+") == 0))    TC = 1;
     else if (n == 2 && strcmpi(s2,"-") == 0)           TC = 2;
     else if (n == 2 && strcmpi(s2,"buffer") == 0) TC = 3;
     else if (n == 3 && strcmpi(s2,"+") == 0 && b == 0) TC = 3;
     else if (n == 3 && strcmpi(s2,"-") == 0 && b == 0) TC = 4;

     if (TC > 0) { Triglevel = (int)((2047*x/10)+0.5);
                   Trig(TC);
                   return(TC);
                 }
     else Option_Error("Invalid option.");
    }
   if (TC == 0) { TC = Trigchoice; return(TC); }
  }



/*************************************************************************
 ************************* read_period(T) **********************************

 Used by the command procedure when the period command has been given
 with correct number of options.

 *************************************************************************/
```

```
read_period(T)
   float T;
  {
   double x,div;

   Td = T;
   if (Td == 0 || Td > 5.8e10)
      Ts=Timer_Per=Period=0;
   else if (Td < 27)
     {
      Period    = 1;
      Timer_Per = (int)(Td*HARDWARE_CLOCK/1000+0.5);
      Ts        = (float)Timer_Per/HARDWARE_CLOCK;
     }
   else
     {
      x = log10(Td);
      if ((int)x < 3)
        div = 1;
      else if ((int)x < 5)
        div = 10;
      else
        div = 27;

      Period    = (long)(Td/div + 0.5);
      Timer_Per = (int)(div*HARDWARE_CLOCK/1000+0.5);
      Ts        = (float)Period*Timer_Per/HARDWARE_CLOCK;
     }
  }


/*************************************************************************
 ******************************** run() ********************************

 Used by the command procedure when the run command has been given
 with correct number of options.

 ************************************************************************/

run()
{
 int  save;
 char ch;

 if (Td >= Tdmin && No_Of_Samples > 0)
   {
    sampler();
    save = chinput("\nDo you wish to save your sampled values?");
    if (save)
      FileChoice(No_Of_Buff,No_Of_Samples,Buff_Index,
                 TwoSignals,Trigchoice,Filesample,Ts);
    else
      scanf("%c",&ch);
    return(TRUE);
   }
 else if (Td != 0 && No_Of_Samples != 0)
   { Option_Error("To small samplingperiod.");
     printf("\nMinimal sampling period = %f\n",Tdmin);
   }
}
```

```
/*************************************************************************
 ************************** void delete() ******************************

 Used by the command procedure when the delete command has been given
 correct number of options. If any of the files given by FileName.smp,
 FileName.fft or FileName.inf exists then they are erased.

 *************************************************************************/

void delete(FileName)

   char FileName[40];

  {
   char SmpName[20],InfName[20],FftName[20];
   char DelSmp[20],DelInf[20],DelFft[20];

   if ((rindex(FileName,'.')) != -1)
     printf("Filename should be given without extension.\n");
   else
   {
     strupr(FileName);
     strcpy(SmpName,FileName);
     strcpy(FftName,FileName);
     strcpy(InfName,FileName);
     strcat(SmpName,".SMP");
     strcat(FftName,".FFT");
     strcat(InfName,".INF");
     strcpy(DelSmp,"del ");
     strcpy(DelFft,"del ");
     strcpy(DelInf,"del ");
     strcat(DelSmp,SmpName);
     strcat(DelFft,FftName);
     strcat(DelInf,InfName);

     if (access(InfName,0) != -1)
     {
       printf("%s erased\n",InfName);
       system(DelInf);
     }
     if (access(SmpName,0) != -1)
     {
       printf("%s erased\n",SmpName);
       system(DelSmp);
     }
     if (access(FftName,0) != -1)
     {
       printf("%s erased\n",FftName);
       system(DelFft);
     }
   }
  }
```

```
/*************************************************************************
 ************************ void fftcheck() ******************************

 Checks if information file (.INF) and sample file (.SMP) exist. If
 they exist then fftcheck calls PutHelpFile with the given filename
 and the FFT transform is calculated.

 The procedure uses functions from the run-time library and therefore
 the following files should be included:
        -stdio.h
        -string.h

 *************************************************************************/

void fftcheck(ComNo,FileName)

   int ComNo;
   char FileName[40];

  {
   char SmpName[40],InfName[40];
   int pos;

   if ((pos = rindex(FileName,'.')) != -1)
     remove_tail_of_string(pos,FileName);
   strupr(FileName);
   strcpy(SmpName,FileName);
   strcpy(InfName,FileName);
   strcat(SmpName,".smp");
   strcat(InfName,".inf");

   if ((access(InfName,0) != -1) && (access(SmpName,0) != -1))
     PutHelpFile(ComNo,FileName);
   else
   {
     if (access(InfName,0) == -1)
       printf("No information file (%s.INF) exists!\n",FileName);
     if (access(SmpName,0) == -1)
       printf("No sample file (%s.SMP) exists!\n",FileName);
     printf("FFT NOT CALCULATED!!\n");
   }
  }



/*************************************************************************
 ***************************** command() *******************************

 Reads a command line from the terminal and uses the procedures and
 functions above and those you find in sampler.h and filesave.h to
 perform the given command.

 *************************************************************************/

void command()
{
 int   i,j,iref,not_End,HaveSampled,CNO;
 char  line[100],c[4][10];
```

```
not_End     = TRUE;
HaveSampled = FALSE;

PutHelpFile(-2," ");

do{
   for (i=0; i<4; i++) c[i][0] = NULL;
   i=j=0;
   printf(">");
   gets(line);
   while (line[i] != NULL)
     {
      while (line[i] == ' ') i++;

      iref=i;

      while ((line[i] != ' ')&&(line[i] != NULL))
        {
         c[j][i-iref] = line[i];
         i++;
        }
      c[j][i-iref] = NULL;

      if (i>iref) j++;
     }

   j--;

   if (i != 0)
   {
    CNO = check_command(strlwr(strdup(c[0])));

    if (CNO == -1 || CNO < 9 && j == 0 || CNO > 5 && CNO < 16 && j == 1
        || (CNO == 8 || CNO > 12 && CNO < 16) && j == 2 ||
            CNO > 14 && j == 3 || CNO == 17 && j == 1)
     {
      switch(CNO)
       {case -1:{Option_Error("Invalid command");                break;}
        case  1:{PRBS = FALSE; Trig(Trigchoice);                 break;}
        case  2:{Trig(0); Trigchoice = 0;                        break;}
        case  3:{not_End=FALSE;                                  break;}
        case  4:{HaveSampled = run();                            break;}
        case  5:{show_parameters();                              break;}
        case  6:{help(c[1],j);                                   break;}
        case  7:{save(c[1],HaveSampled,j);                       break;}
        case  8:{plot(c[1],c[2],j);                              break;}
        case  9:{fftcheck(20,c[1]);                              break;}
        case 10:{PutHelpFile(18,c[1]);                           break;}
        case 11:{PutHelpFile(19,c[1]);                           break;}
        case 12:{read_period(atof(c[1]));                        break;}
        case 13:{read_channel(atoi(c[1]),atoi(c[2]),j);          break;}
        case 14:{read_sample(atol(c[1]),strcmpi(c[2],"file"),j); break;}
        case 15:{Trigchoice = read_trig(c[1],c[2],c[3],j);       break;}
        case 16:{read_PRBS(atoi(c[1]),atof(c[2]),atof(c[3]));    break;}
        case 17:{delete(c[1]);                                   break;}
       }
     }
    else Option_Error("Invalid number of parameters");
   }
  }
 while(not_End);
}
```

Appendix C
Programlista för filesave.c

```
#include <stdio.h>
#include <time.h>
#include <string.h>
#include <math.h>
#include <stdlib.h>
#include <io.h>
#include <process.h>

#define MAX_NO_OF_SAMPLES 32767
#define FALSE 0
#define TRUE 1
#define MAXNAME 40  /* Max namelength for .inf and .smp files. */

extern int huge Sampvec1[MAX_NO_OF_SAMPLES];
extern int huge Sampvec2[MAX_NO_OF_SAMPLES];
extern int huge Buffvec1[MAX_NO_OF_SAMPLES];
extern int huge Buffvec2[MAX_NO_OF_SAMPLES];

/* (Present filename).inf=File where information are stored */
char InfFileName[MAXNAME];

/* (Present filename).smp=File where the samples are stored */
char SmpFileName[MAXNAME];

char OldSmpFile[MAXNAME];
char OldInfFile[MAXNAME];


/*************************************************************************
 ************************** rindex ***************************************

 A function that checks if a character is present in the string that
 strng points to.If the character is present the function returns the
 leftmost position of the given character.
 If character is not present the function returns -1.

 *************************************************************************/

rindex(strng,character)
   char *strng;
   char character;

  {
   int position=-1; /*Letter nonexcistent*/
   int act_position=0;

   while ((*strng != NULL) && (position==-1))
   {
     act_position++;
     if (*strng++==character)
       position=act_position;
   }
   return(position);
  }
```

```
/*************************************************************************
 ************************** chinput ***************************************

 A function that prints the string that prompt points to and adds (y/n).
 The function then reads a character.If the character is not y,Y,n or N
 the function prints an error message and asks for the character again.
 If the character is y or Y the function returns 1 and if the character
 is n or N the function returns 0.

 The function uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h


 The function also uses rindex.

 *************************************************************************/



chinput(prompt)

   char *prompt;

  {
   int ans,Not_Allowed;
   char ch;

   Not_Allowed=TRUE;
   while (Not_Allowed)
   {
     printf("%s (y/n)\n",prompt);
     ch=getchar();
     while (rindex("\n",ch) != -1)
       ch=getchar();
     if (rindex("ynYN",ch)==-1)
       printf("Write y (or Y) for yes or n (or N) for no!\n");
     else
       Not_Allowed=FALSE;
   }
   if (rindex("yY",ch) != -1)
     ans=1;
   else
     ans=0;
   return(ans);
  }
```

```
/*************************************************************************
 ************************** Input() ***************************************

 Read a value with prompt and range checking.

 The procedure uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h

 *************************************************************************/

float Input(prompt,min,max)

   char *prompt;
   float min,max;

  {
   float x;

   do{
      printf("%s: ",prompt);
      scanf("%f",&x);
      if ( x<min || x>max )
        printf("\nValue out of range (%g .. %g)\n",min,max);
     }
   while( x<min || x>max );

   return(x);
  }



/*************************************************************************
 ************************ Get_Time_And_Date ******************************

 A procedure that is used to get the time and date that is to be written
 into the inf file.


 The procedure loads a character string containing the time into the
 vector that timestrng points to.Further it loads a character string
 containing the date into the vector that datestrng points to.
 The procedure uses functions from the run-time library and therefore
 the following files should be included:
     - time.h
     - stdio.h
     - string.h

 The run-time library functions  from the file time.h performs the
 following:
   time  - Gets current time as long integer.
   ctime - Converts time from long integer to character string that has
           the form of the following example:
           Mon Jan 02 02:04:55 1989\n\0

 The run-time library functions from the file string.h performs the
 following:
   strcat(string1,string2) - Appends string2 to string1.
   strdup(string)          - Duplicates string.

 *************************************************************************/
```

```
Get_Time_And_Date(timestrng,datestrng)

   char *timestrng;
   char *datestrng;

  {
   long ltime;
   char *token2,*token;
   char strngvect[30],datevect[20],timevect[15];

   time(&ltime);
   strcpy(strngvect,ctime(&ltime));
   token=strtok(strngvect," ");
   strcpy(datevect,token);
   token2=strtok(NULL," ");
   token=strtok(NULL," ");
   strcat(datevect," ");
   strcat(datevect,token);
   strcat(datevect," ");
   strcat(datevect,token2);
   token=strtok(NULL," ");
   strcpy(timevect,token);
   token=strtok(NULL,"\n");
   strcat(datevect," ");
   strcat(datevect,token);
   strcpy(timestrng,timevect);
   strcpy(datestrng,datevect);

  }
```

```
/*************************************************************************
 ******************** remove_tail_of_string ****************************

 A function that removes the tail (from the position index) of a given
 string.

 The function also checks if the index is correct.If the index is not
 correct the function returns -1.If the index is correct the function
 removes the tail and returns 1.

 The function uses a function from the run-time library and therefore
 the following file should be included:
     - string.h

 The run-time library function from the file string.h performs the
 following:
   strlen(string)      - Finds lenght of string.

 *************************************************************************/
```

```
remove_tail_of_string(index, strng)

   int index;  /*Startposition for the string that shall be erased.*/

   char *strng;  /*String in which a string shall be erased.*/

  {
   int No_Of_Signs;  /*No of signs in the string.*/
   int x;

   /*Check if index is correct.*/
   if (index<1)
     return(-1);  /*Index wrong.*/

   /*Get the stringlength*/
   No_Of_Signs=strlen(strng);

   /*If startposition is greater than the
           number of signs then return -1.*/
   if (index>No_Of_Signs)
     return(-1);

   for (x=1;x<index;x++)
     strng++;
   *strng=NULL;
   return(1);
  }
```

```
/*************************************************************************
 ************************** Check_Filename ******************************

 A function that recieves a pointer (Filename) that points to a string
 containing a filename and a pointer (ext) that points to a string
 containing a file extension (without the '.').
 The function then checks if the filename is written with extension and
 if it is the function removes the extension.The function then checks if
 the filename length is more than 8 positions.If so the function returns
 1.If the length is less than 9 positions the function appends the file
 extension (including the '.') to the filename and returns 0.

 The function uses functions from the run-time library and therefore
 the following file should be included:
     - string.h

 The run-time library functions from the file string.h performs the
 following:
   strcat(string1,string2) - Appends string2 to string1.
   strlen(string)       - Finds lenght of string.


 The function also uses rindex and remove_tail_of_string.

 ************************************************************************
```

```
Check_Filename(Filename,ext)

   char *Filename;
   char *ext;

  {
   int strnglength,pos;
   int NotOkValue=FALSE;
   char *point,*p2,*p3,*p4;
   char fil[40],fe[10];

   p2=fil;
   p3=Filename;
   while ((*p2++=*p3++) != NULL)
     ;
   if ((pos=rindex(Filename,'.')) != -1)
   {
     remove_tail_of_string(pos,Filename);
   }
   strnglength=strlen(Filename);
   if (strnglength<9)
   {
     p3=fe;
     *p3='.';
     p3++;
     *p3=NULL;
     strcat(fe,ext);
     strcat(Filename,fe);
   }
   else
     NotOkValue=TRUE;
   return(NotOkValue);
  }
```

```
/*************************************************************************
 ***************************** Read_Strng ********************************

 A procedure that prints the string that StrPrompt points to.It then
 reads a string and lets AnsStrng point to it.

 The procedure uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h

 *************************************************************************
```

```
Read_Strng(AnsStrng,StrPrompt)

   char *AnsStrng;
   char *StrPrompt;

  {
   printf("%s\n",StrPrompt);
   scanf("%s",AnsStrng);
  }
```

```
/*************************************************************************
 ************************** Init_Inf_File ********************************

 A procedure that initiates the inf files.

 The procedure reads the title.It also recieves the time and date from
 the Get_Time_And_Date procedure.
 All of this are written into the inf file like this:

                          TITLE
                          TIME
                          DATE


 The function uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h

 The procedure also uses Read_Strng and Get_Time_And_Date.

 *************************************************************************/


Init_Inf_File ()


  {
   #define LENGTH 40

   char Text[LENGTH],FileTime[LENGTH],FileDate[LENGTH];

   FILE *fipek,*fopen();

   fipek=fopen(InfFileName,"w");
   Read_Strng(Text,"\nWrite Title");
   fprintf(fipek,"%s\n",Text);
   Get_Time_And_Date (FileTime,FileDate);
   fprintf(fipek,"%s\n",FileTime);
   fprintf(fipek,"%s\n",FileDate);
   fclose(fipek);

  }
```

```
/*************************************************************************
 ************************** New_Sample ***********************************

 A procedure that reads desired filename and intiates the global vectors
 InfFileName and SmpFileName.If the filename lenght is more than 8
 positions the procedure prints an errror message and asks for the file
 name again.
 Example:If you write fname when the procedure asks for the desired file
 name InfFileName will contain fname.inf and SmpFileName will contain
 fname.smp

 The procedure uses functions from the run-time library and therefore
 the following files should be included:
     - stdio.h
     - string.h

 The run-time library functions from the file string.h performs the
 following:
   strcpy(string1,string2) - Copies string2 to string1


 The procedure also uses Read_Strng and Check_Filename

 *************************************************************************/




New_Sample()

  {
   int NotOkName=TRUE;
   int InfFileExist=TRUE;
   int SmpFileExist=TRUE;
   char Act_File[40];

   while (InfFileExist || SmpFileExist)
   {
     printf("\n%s\n","Write filename (max 8 letters)");
     Read_Strng(Act_File,"Fileextension is given automatically.");
     strcpy(InfFileName,Act_File);
     NotOkName=Check_Filename(InfFileName,"inf");
     if (NotOkName)
       printf("Maximum 8 letters in filename\n");
     else
     {
       strcpy(SmpFileName,Act_File);
       Check_FileName(SmpFileName,"smp");
       InfFileExist=(access(InfFileName,0))+1;
       if (InfFileExist)
         printf("Already existing filename (.INF) !\n");
       SmpFileExist=(access(SmpFileName,0))+1;
       if (SmpFileExist)
         printf("Already existing filename (.SMP) !\n");

       if (SmpFileExist || InfFileExist)
          SmpFileExist = InfFileExist =
                1 - chinput("\nDo You wish to overwrite?");
     }
   }
  }
```

```
/*************************************************************************
 ************************** Old_Sample ***********************************

 This function is very similar to New_Sample. This procedure asks for
 an old file, that is, a file in which samplevalues already have been
 stored. If the given file exists then 1 is returned else 0 is returned.
 The global vectors OldSmpFile and OldInfFile will contain the name of
 the old file.

 *************************************************************************/

int Old_Sample()

  {
   int NotOkName=TRUE;
   int InfFileExist=TRUE;
   int SmpFileExist=TRUE;
   char Act_File[40];

   printf("\n%s\n","Write  old filename (max 8 letters)");
   Read_Strng(Act_File,"Fileextension is given automatically.");
   strcpy(OldInfFile,Act_File);
   NotOkName=Check_Filename(OldInfFile,"inf");
   if (NotOkName)
     printf("Maximum 8 letters in filename\n");
   else
   {
     strcpy(OldSmpFile,Act_File);
     Check_FileName(OldSmpFile,"smp");
     InfFileExist=(access(OldInfFile,0));
     if (InfFileExist == -1)
       printf("Filename (.INF) doesn't exist !\n");
     SmpFileExist=(access(OldSmpFile,0));
     if (SmpFileExist == -1)
       printf("Filename (.SMP) doesn't exist !\n");
     if (InfFileExist == -1 || SmpFileExist == -1)
       return(0);
     else return(1);
   }
  }



/*************************************************************************
 ************************** Vect_To_File *********************************

 A procedure that writes the sample values from the samplevector and/or
 the buffervector into the file specified by SmpFileName.

 The procedure requires:
   PreSamples    - How many values are to be read from the buffervector
   FileSamples   - How many samples are to be stored in the file

 The procedure uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h

 *************************************************************************/
```

```
Vect_To_File(PreSamples,FileSamples,Buff_Index,TwoSignals)

   long PreSamples,FileSamples,Buff_Index;
   int  TwoSignals;

  {
   FILE *point,*fopen();
   long Index,SampVect_Samples,Present_Index,Co_Index;


   SampVect_Samples=FileSamples;
   point=fopen(SmpFileName,"w");
   if (PreSamples>0)
   {
     Present_Index=Buff_Index-PreSamples;
     if (Present_Index<0)
       Present_Index=Present_Index+MAX_NO_OF_SAMPLES;
     for (Co_Index=0;Co_Index<PreSamples;Co_Index++)
     {
       if (TwoSignals)
         fprintf(point,"%f    ",(float)Buffvec2[Present_Index]*10/2047);
       fprintf(point,"%f\n",(float)Buffvec1[Present_Index]*10/2047);
       if (Present_Index==(MAX_NO_OF_SAMPLES-1))
         Present_Index=0;
       else
         Present_Index++;
     }
     SampVect_Samples=FileSamples-PreSamples;
   }
   for (Index=0; Index<SampVect_Samples; Index++)
     {
      if (TwoSignals)
        fprintf(point,"%f    ",(float)Sampvec2[Index]*10/2047);
      fprintf(point,"%f\n",(float)Sampvec1[Index]*10/2047);
     }
   fclose(point);
  }
```

```
/*************************************************************************
 ************************** File_To_File ********************************

 A procedure that writes the sample values from the tempfile.tmp into
 the file specified by SmpFileName.

 The procedure requires:
   PreSamples  - How many values are to be read before the trigpoint
   FileSamples - How many samples are to be stored in the file specified
                 by SmpFileName

 The procedure uses functions from the run-time library and therefore
 the following file should be included:
     - stdio.h

 *************************************************************************/
```

```
File_To_File(PreSamples,FileSamples,No_Of_Buff,TwoSignals)

   long PreSamples,FileSamples,No_Of_Buff;
   int  TwoSignals;

  {
   FILE *smp_pointer,*tmp_pointer,*fopen();
   int value;
   long index;


   tmp_pointer=fopen("tempfile.tmp","r");
   smp_pointer=fopen(SmpFileName,"w");

   for (index=0; index<(No_Of_Buff-PreSamples); index++)
     {
      if (TwoSignals)
        fscanf(tmp_pointer,"%d",&value);/*Read to move the filepointer.*/
      fscanf(tmp_pointer,"%d",&value);  /*Read to move the filepointer.*/
     }

   for (index=0; index<FileSamples; index++)
   {
     if (TwoSignals)
       {
        fscanf(tmp_pointer,"%d",&value);
        fprintf(smp_pointer,"%f     ",(float)value*10/2047);
       }
     fscanf(tmp_pointer,"%d",&value);
     fprintf(smp_pointer,"%f\n",(float)value*10/2047);
   }

   fclose(tmp_pointer);
   fclose(smp_pointer);

  }
```

```
/*************************************************************************
 ************************** Load_InfFile *********************************

 A procedure that is used to store characteristic values about how the
 sampling has been carried out in the information file.

 *************************************************************************/


  Load_InfFile(LowerTime,HigherTime,Ts,TotSamp,TwoSignals)

   float LowerTime;
   float HigherTime;
   float Ts;
   long  TotSamp;
   int   TwoSignals;

  {

   #define LEN 40

   FILE *infpoint,*fopen();
   char Inform[LEN];
   int TS;

   TS = (int)(TotSamp);

   infpoint=fopen(InfFileName,"a");
   fprintf(infpoint,"%f\n",LowerTime);
   fprintf(infpoint,"%f\n",HigherTime);
   fprintf(infpoint,"%f\n",Ts);
   fprintf(infpoint,"%ld\n",TotSamp);
   fprintf(infpoint,"%d\n",TS);
   fprintf(infpoint,"%d",TwoSignals);
   fclose(infpoint);
  }




/*************************************************************************
 ************************** FileChoice ***********************************

 A procedure that asks for how many samples that are to be stored in the
 final file specified by SmpFileName. The procedure also asks for where
 on the screen the trigpoint is to be placed. The procedure checks if
 the choice is possible due to the number of samples before and after the
 trigpoint and the total number of samples chosen to be stored in the
 file specified by SmpFileName. When the choice is found to be possible
 the procedure transfers the chosen samples to the file specified by the
 vector SmpFileName. It also writes to the inf file specified by
 InfFileName.

 The procedure uses  functions from the run-time library and therefore
 the following files should be included:
     - stdio.h
     - math.h

 *************************************************************************/
```

```
FileChoice(No_Of_Buff,No_Of_Samples,Buff_Index,TwoSignals,Trigchoice,
           Filesample,Ts)

   long  No_Of_Buff,No_Of_Samples,Buff_Index;
   int   TwoSignals,Trigchoice,Filesample;
   float Ts;

  {
   char ch;
   long TotSamp,PreSamp,Low_Lim,High_Lim,Total;
   int Lowest_Trig,Highest_Trig,Trigpos;
   float LowTime,HighTime,per;

   New_Sample();
   Init_Inf_File();

   if ((Filesample == 0) && (No_Of_Buff > MAX_NO_OF_SAMPLES))
     No_Of_Buff=MAX_NO_OF_SAMPLES;
   Total=No_Of_Samples+No_Of_Buff;
   printf("\nNumber of samples before trigpoint: %ld",No_Of_Buff);
   printf("\nNumber of samples after trigpoint : %ld",(No_Of_Samples-1));
   printf("\nTotal number of samples           : %ld",Total);
   TotSamp=(long)(Input("\n\nHow many samples are to be stored?",
                                                  0.0,(float)(Total)));
   if (Trigchoice<3)
     PreSamp=0;
   else
     {
      Low_Lim=0-TotSamp+1;
      if (TotSamp>No_Of_Buff+1)
        Low_Lim=0-No_Of_Buff;
      High_Lim=TotSamp;
      if (TotSamp>No_Of_Samples)
        High_Lim=No_Of_Samples;
      Lowest_Trig=(int)(ceil(((float)(TotSamp)-
                        (float)(High_Lim))/((float)(TotSamp)-1)*100));
      Highest_Trig=(int)((0-(float)(Low_Lim))/
                                          ((float)(TotSamp)-1)*100);
      if (Highest_Trig>Lowest_Trig)
      {
        printf("Where is the trigpoint to be placed ? (0..100)\n");
        printf("(0=leftmost on screen, 100=rightmost on screen)\n");
        if ((Lowest_Trig>0) || (Highest_Trig<100))
        {
           printf("Because of present choices the trigpoint can only
                                                          be placed\n");
           printf("in the interval %d .. %d\n",Lowest_Trig,Highest_Trig);
        }
        Trigpos=(int)(Input(" ",(float)Lowest_Trig,(float)Highest_Trig));
        PreSamp=(long)((float)(Trigpos)/100*((float)(TotSamp)-1));
      }
      else
        PreSamp=No_Of_Buff;
     }
   LowTime=(float)(-PreSamp*Ts);
   HighTime=(float)((TotSamp-PreSamp-1)*Ts);
   Load_InfFile(LowTime,HighTime,Ts,TotSamp,TwoSignals);
   if (Filesample)
     File_To_File(PreSamp,TotSamp,No_Of_Buff,TwoSignals);
   else
     Vect_To_File(PreSamp,TotSamp,Buff_Index,TwoSignals);
   scanf("%c",&ch);
  }
```

```
/*************************************************************************
 ************************** Oldchoice ***********************************

 This procedure makes it possible to store a part of an old file in a
 new file. Oldchoice starts by asking for the name of the old file. If
 this file exists oldchoice proceeds by asking for the name of the old
 file. Values within a new smaller timeinterval, specified by the user,
 will be saved in the new file.

 *************************************************************************/

OldChoice()
  {
   FILE  *Newpek,*Oldpek,*fopen();
   char  line[30],ch;
   float LowTime,HighTime,period,NewLow,NewHigh,T,value;
   int   TwoSignals;
   long  TotSamp = 0;
   long  TS;

   if (Old_Sample() == 1)
   {
    New_Sample();
    Init_Inf_File();

    Oldpek = fopen(OldInfFile,"r");
    fgets(line,30,Oldpek);
    fgets(line,30,Oldpek);
    fgets(line,30,Oldpek);
    fscanf(Oldpek,"%f",&LowTime);
    fscanf(Oldpek,"%f",&HighTime);
    fscanf(Oldpek,"%f",&period);
    fscanf(Oldpek,"%ld",&TS);
    fscanf(Oldpek,"%d",&TwoSignals);
    fscanf(Oldpek,"%d",&TwoSignals);
    fclose(Oldpek);
    printf("\nLow time  : %f",LowTime);
    printf("\nHigh time : %f",HighTime);
    NewLow = Input("\nChoose new low time",LowTime,HighTime);
    NewHigh = Input("\nChoose new High time",NewLow,HighTime);

    Oldpek = fopen(OldSmpFile,"r");
    Newpek = fopen(SmpFileName,"w");
    T = LowTime;

    while (T < NewLow)
     {
      if (TwoSignals) fscanf(Oldpek,"%f",&value);
      fscanf(Oldpek,"%f",&value);
      T = T + period;
     }

    while (T <= NewHigh)
     {
      if (TwoSignals)
       {
        fscanf(Oldpek,"%f",&value);
        fprintf(Newpek,"%f    ",value);
       }
      fscanf(Oldpek,"%f",&value);
      fprintf(Newpek,"%f\n",value);
```

```
    T = T + period;
    TotSamp++;
   }
  Load_InfFile(NewLow,NewHigh,period,TotSamp,TwoSignals);
 }
 scanf("%c",&ch);
}
```

Appendix D
Programlista för fb.asm

```
NAME FB
TITLE Foreground/Background scheduler for C on IBM AT
PAGE 50,130

;------------------------------------------------------------------------
;
; Overview.
;
; This is a foreground/background scheduler for simple control
; applications. A major design objective is to reduce overhead and
; to determine the maximum practical sampling rate.
;
; The program is normally executing the background task, which in a
; typical case is responsible for operator communication. At regular
; time intervals defined by the user, the scheduler starts the
; foreground task.  Typically, the foreground task is the realization
; of a PID controller.
;
; The scheduler runs on Tandon AT and compatible machines (e.g., IBM AT).
; It has been designed for a user program written in Microsoft C,
; using the small memory model.  It can easily be re-written for other
; programming languages.
;
;
;
; Using the scheduler.
;
; The following declarations must be included in the user's program:
;
; extern void schedule();
; extern void reset();
;
; The code executed by the foreground task must be defined as a void
; function (called a procedure in Pascal).  This function, and all
; functions called by the foreground task, MUST be compiled with stack
; checking turned off.  The best way to do this is to use a pragma:
;
; #pragma check_stack -
; void foreground()
; {
;    /* foreground implementation */
; }
; #pragma check_stack +
;
; The scheduler is invoked at fixed intervals (ticks), determined by the
; variable TIMER_PER defined below.  A typical tick is 10 milliseconds.
; Timing is based on a hardware clock of 1.19318 MHz, the clock is scaled
; by a hardware timer which generates interrupts every TIMER_PER clock
; cycles.
```

```
; The user's program must install the foreground procedure by calling
; schedule().  The second parameter determines TIMER_PER and the third
; parameter determines PERIOD. The foreground is invoked every PERIOD
; interrupt. Example:
;
; schedule(foreground,11932,3);
;
; Here interrupts will be generated every 10 ms (11932/clock frequency),
; but the time between two consecutive foreground invokations will be
; 3 * 10 = 30 ms.
;
; The following should be noted:
;
;   1. The foreground task will always run to completion.  If it needs
; more than one tick (i.e., it should have been started again), the
; lag is registered, and the foregound task is immediately started
; again when it has completed.  Lag does not accumulate, so only
; one restart is remembered.
;
;   2. The foreground task can be re-scheduled at any time by calling
; schedule() again.  In this way a new controller algorithm can
; be installed.
;
;   3. A period of 0 ticks has a special meaning: the foreground task
; will not be executed, but the scheduler is still running.
; This is more efficient than calling reset(), see below.
;
; Before terminating, reset() MUST be called.  Otherwise the computer
; will crash.  Note that is is possible to restart the scheduler, even
; after a reset.  If the program crashes without calling reset(), all
; hope is lost.  For emergency stop, CTRL-BREAK is treated specially
; and will terminate the program more gracefully.  Do not terminate
; the program by pressing CTRL-C.
;
;
;
; Caveats.
;
; There are a number of problems, potential problems and future
; extensions to consider:
;
;   1. The registers of the floating-point coprocessor (Intel 80287)
; are not saved when the foreground task is started.  This
; requires a modification in FGstart.
;
;   2. The foreground task executes in the currently active stack,
; so the internal DOS stack is sometimes used.  We do not know
; the size of the DOS stack, so there is a risk of stack overflow.
;
;   3. The scheduler is tailored to the small memory model, which
; normally only handles 64 KB code and 64 KB data.  To ovecome
; this limitation, use FAR and HUGE keywords; see "Microsoft C
; Compiler User's Guide", section 8.3.
;
;   4. There is only one foreground task.
;
;   5. Many operations may not be possible to perform in the foreground
; task, e. g.,calling DOS routines.
;
; The scheduler has not been thoroughly tested, and no real application
; has been written using it.
```

```
; References.
;
; BURR-BROWN (1986): "The Handbook of Personal Computer Instrumentation",
; Burr-Brown Corporation, Tuscon, Arizona, USA.
;
; IBM (March 1984): "Technical Reference (AT)", IBM Corporation, USA.
;
; IBM (May 1984): "Disk Operating System Version 3.00 Technical Reference",
; IBM Corporation, USA.
;
; INTEL (1985): "iAPX 286 Programmer's Reference Manual", Intel Corporation,
; Santa Clara, California, USA.
;
; MATTSSON, SVEN ERIK (1978): "A Simple Real-Time Scheduler", TFRT-7156,
; Department of Automatic Control, Lund.
;
; MICROSOFT (1986): "Microsoft C Compiler User's Guide", Microsoft
; Corporation, USA.
;
;-----------------------------------------------------------------------
; History:
;
; 1988-01-28  DB  Created.
; 1988-02-03  DB  First working version, no BIOS clock.
; 1988-02-04  DB  BIOS clock updated, handles CTRL-BREAK gracefully.
; 1988-02-05  DB  Improved documentation.
; 1988-03-06  DB  Added A/D and D/A conversion routines for C.
; 1988-05-10  DB  Sends 0 to channel 1, in _reset().
; 1988-05-13  DB  Time with interrupts off reduced in ADin.


.286p


_TEXT SEGMENT  BYTE PUBLIC 'CODE'
_TEXT ENDS
_DATA SEGMENT  WORD PUBLIC 'DATA'
_DATA ENDS
CONST SEGMENT  WORD PUBLIC 'CONST'
CONST ENDS
_BSS SEGMENT  WORD PUBLIC 'BSS'
_BSS ENDS
DGROUP GROUP CONST, _BSS, _DATA
ASSUME  CS: _TEXT, DS: DGROUP, SS: DGROUP, ES: DGROUP
```

```
;------------------------------------------------------------------------
;
; Constant definitions.
;
; TSIGNAL The scheduler will normally send a test signal to
; D/A converter channel 0. This test signal is determined
;               by the constants INTOUT and FGOUT.
; Setting TSIGNAL zero will disable this feature.
;
; INTOUT        Every time the interrupt handler is invoked DAOUT INTOUT
;               will be performed.
;
; FGOUT         Every time the foreground is invoked DAOUT INTOUT will be
;               performed.
;
; CLOCK We should normally update the DOS clock at 18.2 Hz.
; A small speed-up is acheived by setting CLOCK zero.
;
; TIMER_INT This is the timer interrupt number, see BIOS manual.
;
; BIOS_CLOCK_INT The BIOS clock handler should be called at a
; rate of about 18.2 Hz.  Normally the clock interrupt
; does this, so we have to install the original clock
; interrupt handler in a new place.  BIOS_CLOCK_INT
; is a free interrupt number we use for this purpose.
;
; ADBASE Base address of A/D converter registers.  The other
; ADxxxx symbols are offsets from ADBASE.
;
; DAC Base address of D/A converter registers.
;
;

TSIGNAL EQU 1
; CLOCK EQU 1
CLOCK EQU 0
TIMER_INT EQU 8
BIOS_CLOCK_INT EQU 0E5H

;INTOUT         EQU     1
INTOUT          EQU     0
FGOUT           EQU     2


ADBASE EQU 0300H
ADSTATUS EQU ADBASE
ADCHAN EQU ADBASE+1
ADCONV EQU ADBASE+2
ADLOW EQU ADBASE+3
ADHIGH EQU ADBASE+4

DAC EQU 0310H
```

```
;--------------------------------------------------------------------------
;
; Scheduler data
;
; TIMER_PER Determines the period between clock interrupts,
; called a tick. TIMER_PER = ROUND(T * 1193.18),
;               where T is the desired period in milliseconds.
;
; PERIOD        Clock ticks between FG starts. If PERIOD = 1 then
;               the foreground will be started every tick.
;
; ICNT          Used as counter with initial value PERIOD. When
;               ICNT = 1 the foreground will be invoked.
;

_DATA SEGMENT

; TIMER_PER DW 11932 ; approximately 10 ms
; TIMER_PER DW  1193 ; approximately 1 ms
; TIMER_PER DW  597 ; approximately 0.5 ms
; TIMER_PER     DW      298     ; approximately 0.25 ms
TIMER_PER       DW      0       ; 0 = FG task

PERIOD          DW      0
DW      0

ICNT            DW      0
                DW 0

FG_ADDR DW 0 ; pointer to FG procedure
FG_ACTIVE DW 0 ; <>0 = FG active
LAG DW 0 ; >0 = FG task is lagging

INIT_FLAG DW 0 ; 1 = scheduler initialized
BIOS_TIMER DW 0 ; Bios clock int must be called at 18 Hz

BIOS_VEC DW 0 ; original interrupt vector for int 8
DW 0

_DATA ENDS



;--------------------------------------------------------------------------
;
; SENDEOI - Send End-Of-Interrupt to interrupt controller
;
; Note: AX is destroyed.
;
SENDEOI MACRO
MOV AL,20H
OUT 20H,AL
ENDM
```

```
;-------------------------------------------------------------------------
;
; DAOUT  -  output integer voltage on D/A-converter channel 0
;
; Note: AX is destroyed.
; VOLTAGE must be a constant expression.
;

DAOUT MACRO VOLTAGE
      IF TSIGNAL
PUSH DX
CLI
MOV DX,DAC
SUB AX,AX ;; channel 0
OUT DX,AL
INC DX
IF VOLTAGE NE 0
  MOV AX,205*VOLTAGE
  OUT DX,AL
  MOV AL,AH
ELSE
  OUT DX,AL
ENDIF
INC DX
OUT DX,AL
STI
POP DX
      ENDIF
ENDM


;-------------------------------------------------------------------------
;
; TIMER  -  Program timer
;
; Note: AX is destroyed.
;
TCC EQU 043H ; Timer/counter control word
TC0 EQU 040H ; Timer 0

TIMER MACRO PERIOD
CLI
MOV AL,036H
OUT TCC,AL
MOV AX,PERIOD
OUT TC0,AL
MOV AL,AH
NOP
OUT TC0,AL
STI
ENDM
```

```
_TEXT SEGMENT
;------------------------------------------------------------------------
;
; IntHandler - interrupt handler when scheduler is used.
;
; The interrupt handler must perform the following functions:
;
;   1.  Save registers and set up the DS register.
;   2.  Update the BIOS clock at 18.2 Hz, or send an EOI signal.
;   3. Determine if the FG task should be run, otherwise
; restore registers and return.
;
; See also: The Handbook of Personal Computer Instrumentation,
; Burr-Brown Corporation, USA (1986).  Pp 9-30.
;
; Note on timing:  The first and last instructions of the interrupt
; handler (before and after the calls to DAOUT) require 10 + 27
; clock cycles, i.e., 4.6 us at 8 MHz.
;

IntHandler:
PUSH AX
PUSH DS
MOV AX,DGROUP
MOV DS,AX

DAOUT INTOUT

; Increment the BIOS clock timer and check if we must
; call the original interrupt handler.

IF CLOCK
ADD BIOS_TIMER,TIMER_PER
JNC NoBios
INT BIOS_CLOCK_INT
JMP CheckFG  ; BIOS does a SENDEOI
ENDIF

NoBios:
SENDEOI

; Count all interrupts.  TIMER_PER = 0 means that FG task should'nt run.

CheckFG:
CMP TIMER_PER,0          ; if (timer_per == 0) return;
JE IntReturn
SUB     ICNT,1                          ; if (--icnt > 0) return;
        SBB     ICNT+2,0
        MOV     AX,ICNT
        OR      AX,ICNT+2
        JNE     IntReturn

; ICNT = 0, so we should start the FG task. If FG task is already
        ; running, special handling is required.

MOV     AX,PERIOD
        MOV     ICNT,AX
        MOV     AX,PERIOD+2
        MOV     ICNT+2,AX
        CMP FG_ACTIVE,0
JE FGstart
```

```
; There has been 'PERIOD' interrupts while FG task was active,
; so the FG task is lagging.  Flag this, and restart FG task
; as soon as it has finished.  NOTE: FG is only restarted once,
; even if many interrupts were missed.

MOV LAG,1

; Return from interrupt, FG task has not been started.

IntReturn:
DAOUT 0
POP DS
POP AX
IRET



;-------------------------------------------------------------------------
;
; FGstart - suspend BG task and start FG task
;
; Note that there are two potential problems:
;
; 1. The FP registers of the 80287 are not saved.
;
; 2. The currently active stack is used, so the FG task
; is sometimes run in the DOS stack which size
; we do not know.
;
FGstart:
MOV FG_ACTIVE,1

; Save status.  Flags, CS, IP, AX and DS are already on the
; stack in the interrupt frame or saved entering IntHandler.

PUSH ES
PUSHA

; Call the FG task procedure once, or as long as the FG task
; is lagging.  A short call is possible because CS was
; loaded when invoking the interrupt handler.

LagLoop:
        DAOUT FGOUT
MOV LAG,0 ; initially no lag
MOV BX,FG_ADDR
CALL BX
CMP LAG,0 ; call until no more lag
JNZ LagLoop

MOV FG_ACTIVE,0

; Restore registers of BG task and return.

POPA
POP ES

DAOUT 0

POP DS
POP AX
IRET
```

```
;--------------------------------------------------------------------------
;
; BREAK_int - CTRL-BREAK interrupt handler.
;
; This routine is invoked by BIOS through interrupt 1BH when
; CTRL-BREAK is pressed.  We must under all circumstances
; restore the original interrupt handler for the timer.
; In addition, the program is terminated.
;
; See also: IBM AT Technical Reference manual, page 5-7.
;     DOS Version 3.00 Technical Reference, page 5-129.
;

BREAK_int:
PUSHA
PUSH DS
MOV AX,DGROUP
MOV DS,AX

SENDEOI
SENDEOI
CALL _reset

MOV AH,4CH
MOV AL,1
INT 21H ; should not return...

POP DS
POPA
IRET
```

```
;-------------------------------------------------------------------------
;
; _schedule - register FG procedure and period
;
; C syntax:  extern void schedule(procedure FG,Timer_Per,Period);
;
; Note: Timer_Per is an unsigned integer and Period is a long integer.
;
; Note:  The C function MUST be compiled with stack checking turned off.
; The best way is put "#pragma check_stack-" immediately before the
; function, and "#pragma check_stack+" immediately after.  This also
; applies for functions called by the FG task.
;
; See also: DOS Version 3.00 Technical Reference Manual, pp 5-67 and 5-82.
;

PUBLIC _schedule
_schedule PROC NEAR

OFF1 EQU 4 ; offset of FG
OFF2 EQU 6 ; offset of timer_per
OFF3    EQU     8                       ; offset of period
OFF4    EQU    10                       ; offset of period+2

ENTER 0,0

; Setup scheduler: take address of procedure (void function),
; value of timer_per and value of period from the main program in C.

MOV AX,[BP+OFF1]
MOV FG_ADDR,AX
MOV AX,[BP+OFF2]
MOV TIMER_PER,AX
MOV AX,[BP+OFF3]
MOV PERIOD,AX
MOV ICNT,AX
        MOV     AX,[BP+OFF4]
        MOV     PERIOD+2,AX
        MOV     ICNT+2,AX
MOV FG_ACTIVE,0

; Check if scheduler has been initialized.

CMP INIT_FLAG,1
JE NoInit

; Get the original interrupt handler from DOS and save it.

MOV INIT_FLAG,1
MOV AH,35H
MOV AL,TIMER_INT
INT 21H
MOV BIOS_VEC,ES
MOV BIOS_VEC+2,BX

; Install it as handler for an unused interrupt, in order
; to update the BIOS clock.

PUSH DS
MOV AX,ES
MOV DS,AX
MOV DX,BX
```

```
MOV AH,25H
MOV AL,BIOS_CLOCK_INT
INT 21H

; Catch CTRL-BREAK to make sure we reset the interrupt

MOV     AX,CS
        MOV     DS,AX
        MOV     DX,OFFSET BREAK_int
        MOV     AH,25H
        MOV     AL,1BH
        INT     21H

; Install private interrupt handler for scheduler

MOV AX,CS
MOV DS,AX
MOV     DX,OFFSET IntHandler
MOV AH,25H
MOV AL,TIMER_INT
INT 21H
POP DS

; Program timer

TIMER TIMER_PER

NoInit:
LEAVE
RET
_schedule ENDP
```

```
;-------------------------------------------------------------------------
;
; _reset - reset interrupt handler and timer
;
; C syntax:  extern void reset();
;
PUBLIC _reset
_reset PROC NEAR
ENTER 0,0

; Reset some output signals...

PUSH 0
PUSH 1
CALL _DAout

; Restore the original interrupt handler

PUSH DS
MOV AX,BIOS_VEC
MOV DX,BIOS_VEC+2
MOV DS,AX
MOV AH,25H
MOV AL,TIMER_INT
INT 21H
POP DS

; Reset timer and final cleanup

TIMER 0

MOV TIMER_PER,0
MOV PERIOD,0
        MOV     PERIOD+2,0
MOV ICNT, 0
        MOV     ICNT+2,0
MOV FG_ADDR,0
MOV INIT_FLAG,0

LEAVE
RET
_reset ENDP
```

```
;-----------------------------------------------------------------------
;
; _ADin - Analog in for RTI-800.
;
; C syntax:  int ADin(channel)
;       int channel;
;

PUBLIC _ADin
_ADin PROC NEAR

ENTER 0,0
CLI ; interrupts off

MOV DX,ADCHAN
MOV AX,[BP+OFF1] ; channel
OUT DX,AL

MOV DX,ADCONV ; start conversion
SUB AX,AX
OUT DX,AL
STI

; Loop until conversion done

MOV DX,ADSTATUS
LOOP:
IN AL,DX
AND AX,0040H ; bit 6 busy bit
JZ LOOP

; Read result and return in AX

MOV DX,ADHIGH
IN AL,DX
MOV AH,AL
MOV DX,ADLOW
IN AL,DX

LEAVE
RET
_ADin ENDP
```

```
;-------------------------------------------------------------------------
;
; _DAout - Analog out for RTI-800.
;
; C syntax:  void DAout(channel, value)
;       int channel;
;       int value;
;

PUBLIC _DAout
_DAout PROC NEAR

ENTER 0,0
CLI

MOV DX,DAC
MOV AX,[BP+OFF1] ; channel
OUT DX,AL

INC DX
MOV AX,[BP+OFF2] ; value
OUT DX,AL

INC DX
MOV AL,AH
OUT DX,AL

STI
LEAVE
RET
_DAout ENDP


_TEXT ENDS
END
```

Appendix E
Programlista för fft.c

```
#include <math.h>
#include <string.h>
#include <stdio.h>

#define MAX 40
#define M 8192 /** Max number of samples = M/2 **/

float far Samplevector[M];
float far Buffervector[M];
float far Samplevector2[M];
float far Buffervector2[M];


fftVect(filename,ext,n,TwoSig)

   char *filename,*ext;
   int n,TwoSig;
  {
   int NoOfValues,m,m2,x,y,z,i1,i2,i3,i4,i5,i6,j,hj,tj,Lim1,Lim2;
   int JVal,Old_JVal,Buffer,end,k;
   FILE *point,*fopen();
   char SmpName[MAX],FFTname[MAX];
   double md,re,theta,s,c,r;
   float ReIn1,ReIn2,ImIn1,ImIn2,ReIn21,ReIn22,ImIn21,ImIn22,Kf;
   float reval,imval,SinVal,CosVal;


   strcpy(SmpName,filename);
   strcat(SmpName,ext);
   point=fopen(SmpName,"r");

   md=ceil(log10((double)(n))/log10(2.0));
   re=fmod(log10((double)(n)),log10(2.0));
   NoOfValues=(int)(pow(2.0,md));
   m=(int)(md);


   Buffer=FillVect(NoOfValues,n,TwoSig,point);

   fclose(point);

   end=NoOfValues-1;

   theta=3.141592654/(double)(NoOfValues);
   Kf=1;

   for (k=1; k<=m; k++)
   {
     theta=theta*2;
     s=sin(theta);
     c=2*pow(sin(theta/2.0),2.0);
     r=-2*c;
     Kf=Kf*2;
     SinVal=0.0;
```

```
CosVal=1.0;
Old_JVal=0;

if (Buffer)
{
  for (j=0; j<end; j=j+2)
  {
    hj=j+NoOfValues;
    tj=2*j;
    Samplevector[tj]  =(ReIn1=Buffervector[j])+
                              (ReIn2=Buffervector[hj]);
    Samplevector[tj+1]=(ImIn1=Buffervector[j+1])+
                            (ImIn2=Buffervector[hj+1]);
    if (TwoSig)
    {
      Samplevector2[tj]  =(ReIn21=Buffervector2[j])+
                              (ReIn22=Buffervector2[hj]);
      Samplevector2[tj+1]=(ImIn21=Buffervector2[j+1])+
                           (ImIn22=Buffervector2[hj+1]);
    }

    if ((JVal=(int)((float)(j)/Kf)) > Old_JVal)
    {
      Old_JVal=JVal;
      c=r*CosVal+c;
      CosVal=CosVal+c;
      s=r*SinVal+s;
      SinVal=SinVal+s;
    }

    reval=ReIn1-ReIn2;
    imval=ImIn1-ImIn2;
    Samplevector[tj+2]=reval*CosVal-imval*SinVal;
    Samplevector[tj+3]=reval*SinVal+imval*CosVal;
    if (TwoSig)
    {
      reval=ReIn21-ReIn22;
      imval=ImIn21-ImIn22;
      Samplevector2[tj+2]=reval*CosVal-imval*SinVal;
      Samplevector2[tj+3]=reval*SinVal+imval*CosVal;
    }
  }
  Buffer=0;
}
else
{
  for (j=0; j<end; j=j+2)
  {
    hj=j+NoOfValues;
    tj=2*j;
    Buffervector[tj]  =(ReIn1=Samplevector[j])+
                              (ReIn2=Samplevector[hj]);
    Buffervector[tj+1]=(ImIn1=Samplevector[j+1])+
                            (ImIn2=Samplevector[hj+1]);
    if (TwoSig)
    {
      Buffervector2[tj]  =(ReIn21=Samplevector2[j])+
                              (ReIn22=Samplevector2[hj]);
      Buffervector2[tj+1]=(ImIn21=Samplevector2[j+1])+
                           (ImIn22=Samplevector2[hj+1]);
    }

    if ((JVal=(int)((float)(j)/Kf)) > Old_JVal)
```

```
      {
        Old_JVal=JVal;
        c=r*CosVal+c;
        CosVal=CosVal+c;
        s=r*SinVal+s;
        SinVal=SinVal+s;
      }

      reval=ReIn1-ReIn2;
      imval=ImIn1-ImIn2;
      Buffervector[tj+2]=reval*CosVal-imval*SinVal;
      Buffervector[tj+3]=reval*SinVal+imval*CosVal;
      if (TwoSig)
      {
        reval=ReIn21-ReIn22;
        imval=ImIn21-ImIn22;
        Buffervector2[tj+2]=reval*CosVal-imval*SinVal;
        Buffervector2[tj+3]=reval*SinVal+imval*CosVal;
      }
    }
    Buffer=1;
  }
}

m2=m-2;

for (y=0; y<m2; y++)
{
  i3=i6=NoOfValues;
  i1=i2=i4=i5=0;

  if (Buffer)
  {
    Lim2=(int)(pow(2.0,(double)(y)));
    Lim1=(int)((double)(NoOfValues)/Lim2/2);

    for (x=0; x<Lim1; x++)
    {
      for (z=0; z<Lim2; z++)
      {
        Samplevector[i1++]=Buffervector[i2++];
        Samplevector[i1++]=Buffervector[i2++];
        if (TwoSig)
        {
          Samplevector2[i4++]=Buffervector2[i5++];
          Samplevector2[i4++]=Buffervector2[i5++];
        }
      }

      for (z=0; z<Lim2; z++)
      {
        Samplevector[i1++]=Buffervector[i3++];
        Samplevector[i1++]=Buffervector[i3++];
        if (TwoSig)
        {
          Samplevector2[i4++]=Buffervector2[i6++];
          Samplevector2[i4++]=Buffervector2[i6++];
        }
      }
    }
    Buffer=0;
  }
  else
```

```
    {
      Lim2=(int)(pow(2.0,(double)(y)));
      Lim1=(int)((double)(NoOfValues)/Lim2/2);

      for (x=0; x<Lim1; x++)
      {
        for (z=0; z<Lim2; z++)
        {
          Buffervector[i1++]=Samplevector[i2++];
          Buffervector[i1++]=Samplevector[i2++];
          if (TwoSig)
          {
            Buffervector2[i4++]=Samplevector2[i5++];
            Buffervector2[i4++]=Samplevector2[i5++];
          }
        }

        for (z=0; z<Lim2; z++)
        {
          Buffervector[i1++]=Samplevector[i3++];
          Buffervector[i1++]=Samplevector[i3++];
          if (TwoSig)
          {
            Buffervector2[i4++]=Samplevector2[i6++];
            Buffervector2[i4++]=Samplevector2[i6++];
          }
        }
      }
      Buffer=1;
    }
  }
  FillFile(TwoSig,NoOfValues,Buffer,m2,filename);
}
```

```
FillFile(TwoSig,NoOfValues,Buffer,y,filename)

   int TwoSig,NoOfValues,Buffer,y;
   char *filename;
  {
   int Limit1,Limit2,i1,i2,i3,i4,i5,i6,x,z;
   char FFTname[MAX];
   FILE *point,*fopen();

   strcpy(FFTname,filename);
   strcat(FFTname,".fft");
   point=fopen(FFTname,"w");
   i3=i6=NoOfValues;
   i1=i2=i4=i5=0;
   if (Buffer)
   {
     Limit2=(int)(pow(2.0,(double)(y)));
     Limit1=(int)((double)(NoOfValues)/Limit2/2);

     for (x=0; x<Limit1; x++)
     {
       for (z=0; z<Limit2; z++)
       {
         if (TwoSig)
         {
           fprintf(point,"%f  ",Buffervector2[i5++]);
         }
         fprintf(point,"%f\n",Buffervector[i2++]);
         if (TwoSig)
         {
           fprintf(point,"%f  ",Buffervector2[i5++]);
         }
         fprintf(point,"%f\n",Buffervector[i2++]);
       }

       for (z=0; z<Limit2; z++)
       {
         if (TwoSig)
         {
           fprintf(point,"%f  ",Buffervector2[i6++]);
         }
         fprintf(point,"%f\n",Buffervector[i3++]);
         if (TwoSig)
         {
           fprintf(point,"%f  ",Buffervector2[i6++]);
         }
         fprintf(point,"%f\n",Buffervector[i3++]);
       }
     }
   }
   else
   {
     Limit2=(int)(pow(2.0,(double)(y)));
     Limit1=(int)((double)(NoOfValues)/Limit2/2);

     for (x=0; x<Limit1; x++)
     {
       for (z=0; z<Limit2; z++)
       {
         if (TwoSig)
         {
           fprintf(point,"%f  ",Samplevector2[i5++]);
```

```
        }
        fprintf(point,"%f\n",Samplevector[i2++]);
        if (TwoSig)
        {
          fprintf(point,"%f  ",Samplevector2[i5++]);
        }
        fprintf(point,"%f\n",Samplevector[i2++]);
      }

      for (z=0; z<Limit2; z++)
      {
        if (TwoSig)
        {
          fprintf(point,"%f  ",Samplevector2[i6++]);
        }
        fprintf(point,"%f\n",Samplevector[i3++]);
        if (TwoSig)
        {
          fprintf(point,"%f  ",Samplevector2[i6++]);
        }
        fprintf(point,"%f\n",Samplevector[i3++]);
      }
    }
  }
  fclose(point);
}
```

```
FillVect(Val,en,TwoSig,pe)

   int Val,en,TwoSig;
   FILE *pe;
  {
   double md;
   float repart;
   int x,end,end2;

   end=2*en;

   for (x=0; x<end; x++)
   {
     if (TwoSig)
     {
       fscanf(pe,"%f",&repart);
       Buffervector2[x]=repart;
       Buffervector2[x+1]=0.0;
     }
     fscanf(pe,"%f",&repart);
     Buffervector[x++]=repart;
     Buffervector[x]=0.0;
   }

   end2=2*Val;

   if (end2<end || end2>end)
     printf("Filling with trailing zeros!!\n");

   for (x=end; x<end2; x++)
   {
     if (TwoSig)
     {
       Buffervector2[x]=0.0;
       Buffervector2[x+1]=0.0;
     }
     Buffervector[x++]=0.0;
     Buffervector[x]=0.0;
   }
   return(1);
  }
```

```
main()
{ FILE  *p,*fopen();
  int   NOFS,Two,x;
  long  TotSamp;
  float Th,Tl,Ts;
  char  InfName[MAX],strng[MAX],s[40];

  p = fopen("HelpFile.DAT","r");
    fscanf(p,"%d",&x);
    fscanf(p,"%s",strng);
  fclose(p);

  strcpy(InfName,strng);
  strcat(InfName,".inf");

  p = fopen(InfName,"r");
    for (x=1; x<7; x++) fscanf(p,"%s",s);
    fscanf(p,"%f",&Tl);
    fscanf(p,"%f",&Th);
    fscanf(p,"%f",&Ts);
    fscanf(p,"%ld",&TotSamp);
    fscanf(p,"%d",&NOFS);
    fscanf(p,"%d",&Two);
  fclose(p);
  if (NOFS <= M/2)
     fftvect(strng,".smp",NOFS,Two);
  else
     printf("\nMax number of samples = %d\n",M/2);
}
```

Appendix F
Programlista för help.c

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <io.h>
#include <process.h>

void start()
  {
   printf("+------------------------------------------------------------------------+\n");
   printf("|                                                                        |\n");
   printf("|                  WELCOME TO S_A_M_P_L_E_R                              |\n");
   printf("|                                                                        |\n");
   printf("|  Developed by Leif Nilsson and Patrik Nilsson as a master thesis       |\n");
   printf("|  work at the Department of Automatic Control at Lund Institute of      |\n");
   printf("|  Technology.                                                           |\n");
   printf("|                                                                        |\n");
   printf("|              Version 1.01        1-June-1990                           |\n");
   printf("|                      Serial Number 1                                   |\n");
   printf("|                                                                        |\n");
   printf("+------------------------------------------------------------------------+\n\n");
   printf("               Print HELP to list available commands!\n\n");
   printf("");
  }

void help_commands()
  {
   printf("\n\n   channel   ADchannel1 [ADchannel2]");
   printf("\n   delete    FileName");
   printf("\n   FFT       FileName");
   printf("\n   help      [command]");
   printf("\n   info      FileName");
   printf("\n   list      smp|inf|fft");
   printf("\n   NoPRBS");
   printf("\n   notrig");
   printf("\n   period    SamplePeriod");
   printf("\n   plot      [amplitude] [FFT]");
   printf("\n   PRBS      period meanvalue amplitude");
   printf("\n   quit");
   printf("\n   run");
   printf("\n   sample    NumberOfSamples [file]");
   printf("\n   save      [old]");
   printf("\n   show");
   printf("\n   trig     level [+|-] [buffer]\n\n");
  }



void help_channel()
  {
   printf("\n\n                   channel   ADchannel1 [ADchannel2]\n");
   printf("\n This command takes either one or two options. If two channels are given then");
   printf("\n two signals will be sampled. The signal connected to ADchannel1 is triggable.");
   printf("\n The values of the channels must be in the interval 0 - 5.\n");
  }
```

```
void help_sample()
  {
   printf("\n\n                    sample    NumberOfSamples [file]\n");
   printf("\n This command takes either one or two options. The first option determines the");
   printf("\n number of values that should be sampled and if the second option is given");
   printf("\n then the sampled values are stored in a file.\n");
  }

void help_PRBS()
  {
   printf("\n\n                    PRBS      period meanvalue amplitude\n");
   printf("\n This command takes three options. When given the PRBS generator will be ");
   printf("\n activated with it's period, meanvalue and amplitude determined by the ");
   printf("\n command options. The PRBS signal is available at D/A channel 1 and the period");
   printf("\n in ms before this signal starts repeating itself is given by: ");
   printf("\n     period*T*Td , where Td = sampling period in ms and T = 255\n");
  }

void help_NoPRBS()
  {
   printf("\n\n                    NoPRBS\n");
   printf("\n This command deactivates the PRBS generator.\n");
  }

void help_trig()
  {
   printf("\n\n                    trig     level [+|-] [buffer]\n");
   printf("\n This command takes either one, two or three options. The first option ");
   printf("\n determines the level on which the signal connected to ADchannel1 (see channel");
   printf("\n command) will be triggered. With the second option you determine if the ");
   printf("\n triggered signal should be triggered on positive or negative slope, if the");
   printf("\n option is left out then + will be chosen. Values sampled before the trigpoint");
   printf("\n will be buffered only if the buffer option is given.\n");
  }

void help_notrig()
  {
   printf("\n\n                    notrig\n");
   printf("\n If this command is given then no signal will be triggered.\n");
  }

void help_period()
  {
   printf("\n\n                    period    SamplePeriod\n");
   printf("\n With this command you can determine the sample period in milliseconds.\n");
  }

void help_show()
  {
   printf("\n\n                              show\n");
   printf("\n Displays current parameters, that is, this command tells you how the sampling ");
   printf("\n will be done if you start sampling now.\n");
  }

void help_run()
  {
   printf("\n\n                              run\n");
   printf("\n Checks if the current parameters are ok and if so starts sampling, else an");
   printf("\n errormessage will be displayed.\n");
  }
```

```
void help_plot()
  {
   printf("\n\n                    plot      [amplitude] [FFT]\n");
   printf("\n Plots values which you have saved in a file after sampling. The amplitude ");
   printf("\n option is used to scale the plots vertical axis to +/- amplitude, if not  ");
   printf("\n given then maximal amplitude (10.0) will be chosen. Only integer values in ");
   printf("\n the range 1 - 10 will be accepted. The red signal is the one that might be ");
   printf("\n triggered. If you press one of the mouse buttons then the position of the ");
   printf("\n cursor will be displayed in the blue field.");
   printf("\n If the FFT option is given then the values to be plotted will be taken from");
   printf("\n a file with the extension .FFT. The vertical axis will in this case be scaled");
   printf("\n from 0 to amplitude*1E6.");
   printf("\n In order to return from plotting you must press one of the mouse buttons when");
   printf("\n the cursor is in the red field.\n");
  }

void help_save()
  {
   printf("\n\n                      save      [old]\n");
   printf("\n If the option is given you will be able to save a part of an old sample file");
   printf("\n in a new file named by yourself. If you leave out the option this command will");
   printf("\n check if you have sampled anything and if you have you will be able to save");
   printf("\n those values in a file named by yourself.\n");
  }

void help_info()
  {
   printf("\n\n                      info      FileName\n");
   printf("\n This command tells you if FileName .smp,.inf and .fft exists. If FileName.inf");
   printf("\n exist then the information stored in this file will be displayed.\n");
  }

void help_list()
  {
   printf("\n\n                      list      smp|inf|fft\n");
   printf("\n List smp/inf/fft lists all available files with the given extension.\n");
  }

void help_FFT()
  {
   printf("\n\n                      FFT       FileName\n");
   printf("\n The command FFT will perform Fast Fourier Transform on the file named FileName");
   printf("\n if the file exists.\n");
  }

void help_delete()
  {
   printf("\n\n                      delete       FileName\n");
   printf("\n The command Delete will delete FileName.SMP/INF/FFT if existing.\n");
  }

void help_quit()
  {
   printf("\n\n                      quit\n");
   printf("\n Quits execution of this program and return to DOS.\n");
  }
```

```
/*************************************************************************
 ********************** Info(FileName) ********************************

  A procedure that recieves a filename and prints out the available
  information. That includes if .INF , .SMP and .FFT files are available.
  If .INF file is available the procedure also prints out the information
  in the .INF file. It also writes the number of samples before and after
  the trigpoint and the total number of samples.

  The procedure uses  functions from the run-time library and therefore
  the following files should be included:
      - stdio.h
      - string.h
      - process.h
      - io.h

 *************************************************************************/

void Info(FileName)

   char *FileName;

  {
   char SmpName[40],InfName[40],FftName[40],UpName[40];
   char FileText[100];
   char Wait;
   int InfExist,SmpExist,FftExist,TwoSignals,Text3,n;
   long TotSamp;
   float LowTime,HighTime,Per;
   FILE *point,*fopen();

   system("cls");
   strcpy(SmpName,FileName);
   strcpy(InfName,FileName);
   strcpy(FftName,FileName);
   strcat(SmpName,".smp");
   strcat(InfName,".inf");
   strcat(FftName,".fft");
   InfExist=(access(InfName,0))+1;
   SmpExist=(access(SmpName,0))+1;
   FftExist=(access(FftName,0))+1;

   strcpy(UpName,strupr(strdup(FileName)));

   if ((InfExist == 0) && (SmpExist == 0) && (FftExist == 0))
     printf("NO FILES AT ALL WITH NAME %s EXIST!\n\n",UpName);
   else
   {
     printf("INFORMATION ON %s:\n\n",UpName);
     printf("Information file (%s.INF) ",UpName);
     if (InfExist)
       printf("exists.\n");
     else
       printf("doesn't exist.\n");
```

```
    printf("Sample file (%s.SMP) ",UpName);
    if (SmpExist)
      printf("exists.\n");
    else
      printf("doesn't exist.\n");

    printf("FFT file (%s.FFT) ",UpName);
    if (FftExist)
      printf("exists.\n\n");
    else
      printf("doesn't exist.\n\n");

    if (InfExist)
    {
      point=fopen(InfName,"r");
      fscanf(point,"%s",FileText);
      printf("Title: %s\n",FileText);
      fscanf(point,"%s",FileText);
      printf("Time : %s\n",FileText);
      fscanf(point,"%s",FileText);
      printf("Date : %s",FileText);
      for (n=1; n<4; n++)
      {
        fscanf(point,"%s",FileText);
        printf(" %s",FileText);
      }
      printf("\n\n");
      fscanf(point,"%f",&LowTime);
      printf("Lower time limit  : %f seconds\n",LowTime);
      fscanf(point,"%f",&HighTime);
      printf("Higher time limit : %f seconds\n",HighTime);
      fscanf(point,"%f",&Per);
      printf("Sample period     : %f seconds\n\n",Per);

      fscanf(point,"%ld",&TotSamp);
      printf("Total number of samples : %ld\n\n",TotSamp);

      fscanf(point,"%d",&TwoSignals);
      fscanf(point,"%d",&TwoSignals);
      if (TwoSignals)
        printf("Sample file contains two signals.\n\n");
      else
        printf("Sample file contains one signal.\n\n");
    }
  }
  printf("Press <ENTER> when ready\n");
  scanf("%c",&Wait);
  system("cls");
}
```

```
/*****************************************************************************
 ************************** listfile ***************************************

  A procedure that prints the available files of a certain type. ftype is
  used for telling what type of files that are to be listed. If ftype=1
  all .INF files are printed out. If ftype=2 all .SMP files are printed
  out and if ftype=3 all .FFT files are printed out.

  The procedure uses functions from the run-time library and therefore the
  following files should be included:
      -process.h
      -stdio.h

 *****************************************************************************/


void listfile(ftype)

   int ftype;
  {
   char Wait;
   char st[50],st1[50],st2[50],st3[50];
   FILE *point,*fopen();
   int k,End_File,s1,s2,s3;

   system("cls");
   printf("AVAILABLE ");
   if (ftype == 1)
   {
     printf("INFORMATION FILES (.INF) ARE THE FOLLOWING:\n\n");
     system("dir *.inf >gsatreju.qws");
   }
   if (ftype == 2)
   {
     printf("SAMPLE FILES (.SMP) ARE THE FOLLOWING:\n\n");
     system("dir *.smp >gsatreju.qws");
   }
   if (ftype == 3)
   {
     printf("FFT FILES (.FFT) ARE THE FOLLOWING:\n\n");
     system("dir *.fft >gsatreju.qws");
   }
   point=fopen("gsatreju.qws","r");
   fscanf(point,"%[^\n]\n",st);
   fscanf(point,"%[^\n]\n",st);
   fscanf(point,"%[^\n]\n",st);

   fscanf(point,"%s",st1);
   fscanf(point,"%[^\n]",st);
   fscanf(point,"%s",st2);
   fscanf(point,"%[^\n]",st);

   End_File=feof(point);
   s1=1;
   s2=0;
   s3=0;
   k=0;
```

```
 while (End_File == 0)
 {
   k++;
   if (k == 20)
   {
     printf("\nPress <ENTER> to continue!\n");
     scanf("%c",Wait);
     k=0;
   }
   if (s1 == 1)
   {
     fscanf(point,"%s",st3);
     fscanf(point,"%[^\n]",st);
     s1=0;
     s2=1;
     printf("    %s\n",st1);
   }

   else if (s2 == 1)
   {
     fscanf(point,"%s",st1);
     fscanf(point,"%[^\n]",st);
     s2=0;
     s3=1;
     printf("    %s\n",st2);
   }

   else if (s3 == 1)
   {
     fscanf(point,"%s",st2);
     fscanf(point,"%[^\n]",st);
     s3=0;
     s1=1;
     printf("    %s\n",st3);
   }

   End_File=feof(point);
 }
 fclose(point);
 system("del gsatreju.qws");
 printf("\nPress <ENTER> when ready.\n");
 scanf("%c",&Wait);
 system("cls");
}
```

```
/*************************************************************************
 ***************************** list(s) ***********************************

 *************************************************************************/

void list(s)
   char *s;
  {
   if (strcmpi(s,"inf") == 0)
     listfile(1);
   else if (strcmpi(s,"smp") == 0)
     listfile(2);
   else if (strcmpi(s,"fft") == 0)
     listfile(3);
   else
     printf("Invalid option\n");
  }



/*************************************************************************/
/************************  Main ******************************************/
/*************************************************************************/

main()
{ FILE *p,*fopen();
  int x;
  char strng[10];

  p = fopen("HelpFile.DAT","r");
    fscanf(p,"%d",&x);
    fscanf(p,"%s",strng);
  fclose(p);

  switch(x)
  { case -2 : { start();         break; }
    case 0  : { help_commands(); break; }
    case 1  : { help_NoPRBS();   break; }
    case 2  : { help_notrig();   break; }
    case 3  : { help_quit();     break; }
    case 4  : { help_run();      break; }
    case 5  : { help_show();     break; }
    case 7  : { help_save();     break; }
    case 8  : { help_plot();     break; }
    case 9  : { help_FFT();      break; }
    case 10 : { help_info();     break; }
    case 11 : { help_list();     break; }
    case 12 : { help_period();   break; }
    case 13 : { help_channel();  break; }
    case 14 : { help_sample();   break; }
    case 15 : { help_trig();     break; }
    case 16 : { help_PRBS();     break; }
    case 17 : { help_delete();   break; }
    case 18 : { Info(strng);     break; }
    case 19 : { list(strng);     break; }
  }
}
```

Appendix G
Programlista för plot.mod

```
MODULE Plot;

FROM Graphics IMPORT
  handle, color, rectangle, point, buttonset,
  VirtualScreen, SetWindow, SetViewPort, Shutdown,
  SetLineColor, PolyLine, SetFillColor, FillRectangle,
  GetMouseRectangle, SetMouseRectangle, WriteString,
  ShowCursor, WaitForMouse;

FROM FileSystem IMPORT
  File, Response, Lookup, SetRead, ReadChar, Close;

FROM InOut IMPORT
  OpenInput, CloseInput, WriteLn, ReadInt, Write;

IMPORT InOut; (* WriteString *)

FROM RealInOut IMPORT
  ReadReal,Done;

FROM LongIO IMPORT ReadLongInt;

FROM RTMouse IMPORT Init;

FROM ConvReal IMPORT RealToString;

FROM NumberConversion IMPORT StringToInt;

FROM MathLib IMPORT ln, exp, sqrt;

VAR
  i,TwoSignals,FFT : INTEGER;
  T0,T1,TD,T,amp   : REAL;
  N,F1,FD,Ln10     : REAL;
  w,n              : CARDINAL;
  s                : ARRAY [0..12] OF CHAR;
  h1,h2,h3         : handle;
  win              : rectangle;
  p1,p2            : point;
  b                : buttonset;
```

```
PROCEDURE ReadInfo(VAR amp,T0,T1,TD,N,F1,FD : REAL;
                   VAR FFT,TwoSignals : INTEGER);

VAR f  : File;
    s  : ARRAY [1..2] OF CHAR;
    i  : INTEGER;
    Ok : BOOLEAN;
    LI : LONGINT;

BEGIN
  Lookup(f,"PLOTFILE.DAT",FALSE);
    SetRead(f);
    ReadChar(f,s[1]);
    ReadChar(f,s[1]);
    ReadChar(f,s[2]);
    StringToInt(s,i,Ok);
    amp := FLOAT(i);
    ReadChar(f,s[1]);
    ReadChar(f,s[2]);
    StringToInt(s,FFT,Ok);
  Close(f);

  WriteLn;
  InOut.WriteString("Filename (.INF) ");
  OpenInput('inf');

  FOR i := 1 TO 7 DO
    ReadReal(T0);
  END;
  ReadReal(T1);
  ReadReal(TD);
  ReadLongInt(LI);
  ReadInt(i);
  ReadInt(TwoSignals);
  CloseInput;
  N  := FLOAT(i);
  F1 := 1.0/(TD*2.0);
  FD := 1.0/(TD*N);
END ReadInfo;



PROCEDURE SetPlotWindow(VAR h : handle; amp,T0,T1 : REAL);
VAR  win,VP : rectangle;
BEGIN
  win.xlo := T0 - (T1 - T0)/100.0; win.xhi := T1 + (T1 - T0)/100.0;
  win.ylo := -amp - amp/10.0;      win.yhi := amp + amp/10.0;
  VP.xlo  := 0.0;                  VP.xhi  := 1.5;
  VP.ylo  := 0.1;                  VP.yhi  := 1.0;
  SetViewPort(h,VP);
  SetWindow(h,win);
  SetFillColor(h,black);
  FillRectangle(h,win);
END SetPlotWindow;
```

```
PROCEDURE SetFFTWindow(VAR h : handle; amp,F1,FD : REAL);
VAR  win,VP : rectangle;
     LgF1   : REAL;
BEGIN
  LgF1    := ln(F1)/ln(10.0);
  win.xlo := -LgF1/100.0;              win.xhi := LgF1 + LgF1/100.0;
  win.ylo := -2.0 - (amp + 2.0)/12.0;  win.yhi := amp + (amp + 2.0)/12.0;
  VP.xlo  := 0.0;                      VP.xhi  := 1.5;
  VP.ylo  := 0.1;                      VP.yhi  := 1.0;
  SetViewPort(h,VP);
  SetWindow(h,win);
  SetFillColor(h,black);
  FillRectangle(h,win);
END SetFFTWindow;


PROCEDURE SetInfWindow(VAR h : handle; VAR win : rectangle);

VAR VP : rectangle;

BEGIN
  win.xlo := 0.0; win.xhi := 10.0;
  win.ylo := 0.0; win.yhi := 10.0;
  VP.xlo  := 0.0; VP.xhi  := 1.2;
  VP.ylo  := 0.0; VP.yhi  := 0.1;
  SetViewPort(h,VP);
  SetWindow(h,win);
  SetFillColor(h,blue);
  FillRectangle(h,win);
END SetInfWindow;


PROCEDURE SetExitWindow(VAR h : handle);

VAR win,VP : rectangle;
    p      : point;

BEGIN
  win.xlo := 0.0; win.xhi := 10.0;
  win.ylo := 0.0; win.yhi := 10.0;
  VP.xlo  := 1.2; VP.xhi  := 1.5;
  VP.ylo  := 0.0; VP.yhi  := 0.1;
  SetViewPort(h,VP);
  SetWindow(h,win);
  SetMouseRectangle(h,win,3);
  SetFillColor(h,red);
  FillRectangle(h,win);
  p.v := 4.0; p.h := 1.0;
  WriteString(h,p,"Press To Exit");
END SetExitWindow;
```

```
PROCEDURE Axis(VAR h : handle; amp,T0,T1 : REAL);
VAR T,Horigo : REAL;
    parray   : ARRAY [0..1] OF point;
BEGIN
  SetLineColor(h1,white);

  (* Horisontal axis *)

  parray[0].h := T0;
  parray[1].h := T1;
  parray[0].v := 0.0;
  parray[1].v := 0.0;
  PolyLine(h,parray,2);
  parray[0].v := -amp/100.0;
  parray[1].v := amp/100.0;

  T:=((T1-T0)/10.0);

  IF (T0 <= 0.0) AND (T1 >= 0.0) THEN
    Horigo := 0.0;
  ELSIF (T0 > 0.0) THEN
    Horigo := T0;
  ELSE
    Horigo := T1;
  END;

  parray[0].h := Horigo;
  WHILE parray[0].h >= T0 DO
    parray[1].h := parray[0].h;
    PolyLine(h,parray,2);
    parray[0].h := parray[0].h - T;
  END;

  parray[0].h := Horigo;
  WHILE parray[0].h <= T1 DO
    parray[1].h := parray[0].h;
    PolyLine(h,parray,2);
    parray[0].h := parray[0].h + T;
  END;

  (* Vertical axis *)

  parray[0].h := Horigo;
  parray[1].h := Horigo;
  parray[0].v := -amp;
  parray[1].v := amp;
  PolyLine(h,parray,2);
  parray[0].h := Horigo - T/20.0;
  parray[1].h := Horigo + T/20.0;

  WHILE parray[0].v <= amp DO
    parray[1].v := parray[0].v;
    PolyLine(h,parray,2);
    parray[0].v := parray[0].v + amp/10.0;
  END;
END Axis;
```

```
PROCEDURE FFTAxis(VAR h : handle; amp,F1,FD : REAL);
VAR F,Horigo : REAL;
    parray   : ARRAY [0..1] OF point;
    LgF1     : REAL;
BEGIN
  SetLineColor(h1,white);
  LgF1 := ln(F1)/ln(10.0);

  (* Horisontal axis *)

  parray[0].h := 0.0;
  parray[1].h := LgF1;
  parray[0].v := -2.0;
  parray[1].v := -2.0;
  PolyLine(h,parray,2);
  parray[0].v := -2.0 - (amp + 2.0)/120.0;
  parray[1].v := -2.0 + (amp + 2.0)/120.0;

  F:=LgF1/10.0;

  parray[0].h := 0.0;
  WHILE parray[0].h <= LgF1 DO
    parray[1].h := parray[0].h;
    PolyLine(h,parray,2);
    parray[0].h := parray[0].h + F;
  END;

  (* Vertical axis *)

  parray[0].h := 0.0;
  parray[1].h := 0.0;
  parray[0].v := -2.0;
  parray[1].v := amp;
  PolyLine(h,parray,2);
  parray[0].h := -F/20.0;
  parray[1].h := F/20.0;

  parray[0].v := -2.0;
  WHILE parray[0].v <= amp DO
    parray[1].v := parray[0].v;
    PolyLine(h,parray,2);
    parray[0].v := parray[0].v + (amp + 2.0)/12.0;
  END;
END FFTAxis;
```

```
PROCEDURE ReadPlot(VAR h : handle; T0,T1,TD:REAL; TwoSignals : INTEGER);
VAR parray1,parray2 : ARRAY [0..1] OF point;
    i               : INTEGER;
BEGIN
  IF TwoSignals = 1 THEN
    ReadReal(parray2[0].v);
    parray2[0].h:=T0;
  END;

  ReadReal(parray1[0].v);
  parray1[0].h:=T0;
  parray1[1].h:=T0;

  (* Plot values between T0 and T1 *)

  WHILE Done DO

    IF TwoSignals = 1 THEN
      ReadReal(parray2[1].v);
      parray2[1].h := parray2[0].h + TD;
      SetLineColor(h,green);
      IF Done THEN PolyLine(h,parray2,2); END;
    END;

    ReadReal(parray1[1].v);
    parray1[1].h := parray1[0].h + TD;
    SetLineColor(h,red);
    IF Done THEN PolyLine(h,parray1,2); END;

    parray1[0] := parray1[1];
    parray2[0] := parray2[1];
  END;
END ReadPlot;
```

```
PROCEDURE ReadFFTPlot(VAR h : handle; F1,FD:REAL; TwoSignals : INTEGER);
VAR parray1,parray2 : ARRAY [0..1] OF point;
    i               : INTEGER;
    RE1,RE2,IM1,IM2 : REAL;
    Ln10,F          : REAL;
BEGIN
  Ln10 := ln(10.0);
  F    := FD;

  IF TwoSignals = 1 THEN ReadReal(RE2); END;
  ReadReal(RE1);
  IF TwoSignals = 1 THEN ReadReal(IM2); END;
  ReadReal(IM1);

  IF TwoSignals = 1 THEN
     parray2[0].v := ln(sqrt(RE2*RE2 + IM2*IM2))/Ln10;
     parray2[0].h := 0.0;
  END;

  parray1[0].v := ln(sqrt(RE1*RE1 + IM1*IM1))/Ln10;
  parray1[0].h := 0.0;
  parray1[1].h := 0.0;

  (* Plot values between 0 and F1 *)

  WHILE (F <= F1) DO

    IF TwoSignals = 1 THEN ReadReal(RE2); END;
    ReadReal(RE1);
    IF TwoSignals = 1 THEN ReadReal(IM2); END;
    ReadReal(IM1);
    F := F + FD;
    IF TwoSignals = 1 THEN
       parray2[1].v := ln(sqrt(RE2*RE2 + IM2*IM2))/Ln10;
       parray2[1].h := ln(F)/Ln10;
       SetLineColor(h,green);
       IF Done THEN PolyLine(h,parray2,2); END;
    END;

    parray1[1].v := ln(sqrt(RE1*RE1 + IM1*IM1))/Ln10;
    parray1[1].h := ln(F)/Ln10;
    SetLineColor(h,red);
    IF Done THEN PolyLine(h,parray1,2); END;

    parray1[0] := parray1[1];
    parray2[0] := parray2[1];
  END;
END ReadFFTPlot;
```

```
BEGIN
  Ln10 := ln(10.0);
  ReadInfo(amp,T0,T1,TD,N,F1,FD,FFT,TwoSignals);

  WriteLn;
  IF (FFT = 1) THEN InOut.WriteString("Filename (.FFT) "); OpenInput('fft');
  ELSE InOut.WriteString("Filename (.SMP) "); OpenInput('smp');          END;

  Init;
  VirtualScreen(h1);
  VirtualScreen(h2);
  VirtualScreen(h3);

  IF FFT = 1 THEN
     SetFFTWindow(h1,amp,F1,FD);
     FFTAxis(h1,amp,F1,FD);
     ReadFFTPlot(h1,F1,FD,TwoSignals);
  ELSE
     SetPlotWindow(h1,amp,T0,T1);
     Axis(h1,amp,T0,T1);
     ReadPlot(h1,T0,T1,TD,TwoSignals);
  END;
  CloseInput;

  ShowCursor;

  SetInfWindow(h2,win);
  SetExitWindow(h3);

  n := 0; w := 10; p2.v := 4.0;
  WHILE n<>3 DO
    WaitForMouse(h1,p1,b);
    n := GetMouseRectangle(h3);
    FillRectangle(h2,win);
    p2.h := 0.5;
    IF FFT = 1 THEN
       WriteString(h2,p2,"Frequency :");
       p2.h := 2.5;
       RealToString(exp(p1.h*Ln10),s,w);
       WriteString(h2,p2,s);
       p2.h := 4.2;
       WriteString(h2,p2,"Hz           Power :");
       p2.h := 7.5;
       RealToString(exp(p1.v*Ln10),s,w);
       WriteString(h2,p2,s);
       p2.h := 9.3;
       WriteString(h2,p2,"W");
    ELSE
       WriteString(h2,p2,"Time :");
       p2.h := 1.5;
       RealToString(p1.h,s,w);
       WriteString(h2,p2,s);
       p2.h := 3.2;
       WriteString(h2,p2,"s              Voltage :");
       p2.h := 7.0;
       RealToString(p1.v,s,w);
       WriteString(h2,p2,s);
       p2.h := 8.8;
       WriteString(h2,p2,"V");
    END;
  END;
```

```
  Shutdown;
END Plot.
```