Førsteamanuensis Bernt Ribbum
Inst. for elektroteknikk og databehandling
Postboks 2557 Ullandhaug, 4004 Stavanger
Tlf 51 83 20 16 / Fax 51 83 17 50
email: bernt@hsr.no

Ingeniørstudenter i teleteknikk

Stavanger, 23. oktober 1996

TE509 Digitale og analoge filtre, DSP-lab høsten 1996

Faglærer:	Bernt Ribbum Institutt for elektroteknikk og databehandling Rom E-423b
Laboratorium:	Rom E-451b
Grupper:	Fordel dere i grupper på 2-4 studenter.
Øvinger:	Dere skal utføre to temmelig enkle forsøk i denne laben. Siden det er relativt mye nytt stoff dere må tilegne dere for å utføre disse forsøkene, er oppgavene mye i form av en “kokebok”. Dette betyr også at det er mye opp til dere selv hvor mye dere ønsker å lære. Laben kan gjennomføres i blinde dersom man ønsker — dette er dog ikke å anbefale!
Forarbeid:	Les igjennom oppgavesettet og design filtre i Matlab før dere begynner laben.
Dokumentasjon:	*Igjen, før labjournal.* Her skal all informasjon av betydning noteres (så som instrumentbruk, programløsninger, måleresultater etc). Fra denne laben krever vi også en kort rapport.
Tider:	Laben er stort sett ledig hver dag i uken. Merk at **laboratoriet skal være låst** når ingen er der. Untertegnede samt Tore Eike (E-455) vil hjelpe til med å åpne og låse. *Frist for innlevert labrapport: mandag 11. november kl. 13.15.*

Om verktøyet

Følgende programmer/utstyr skal brukes til gjennomføringen av oppgavene:

- **Matlab.** Matlab brukes til filterdesign, og er tilgjengelig på PC'ene på laben (under MS-Windows), samt på alle Unix-stasjoner på instituttet.
- **ADSP-21020 utviklingskort.** Kortet inneholder en Analog Devices flyttallsprosessor (21020) med klokkehastighet 25 MHz. Det er separat program- og datahukommelse. På grunn av to eksterne busser samt separat multiplikator og ALU, kan prosessoren innen én klokkesyklus utføre en multiplikasjon, en addisjon, en subtraksjon, samt to flytt-operasjoner. Dette gir et maksimum (*peak rate*) på 75 MFLOPS (millioner flyttallsoperasjoner pr. sekund). Analoge kanaler for inn- og utlesning av signaler kan kobles til mikrofon og øretelefoner (mini-jack), samt signalgenerator og oscilloskop (phono-plugger). Eksterne spenninger på $+5$V samt ± 15V er tilkoblet.
- **EZ-ICE emulator.** Dette er koblingen fra utviklingskortet til PC'en (flatkabel og instikk-kort). Via denne kan vi kjøre programmer på DSP-kortet fra PC'en, samt se på registerinnhold, minne etc. etc.
- **GNU C-kompilator (g21k).** Dette er Analog Devices' versjon av GNU CC, spesialsydd for å kompilere ADSP-21020-programmer på PC'en. Vi skal altså lage programmer i C som så kan utføres i sann tid på signalprosessoren.
- **EZ-ICE emulatorprogram (ice21k).** Dette er programmet vi kjører på PC'en som gir oss kontroll over utviklingskortet.

All dokumentasjon om DSP-kortene finnes på laboratoriet; se forøvrig teksten til oppgavene samt vedleggene for ytterligere informasjon.

Oppgave 1 - "Talk"

Denne oppgaven skal kjøres utelukkende på ADSP-21020. Programmet er en meget enkel sak som gir oss mulighet til å teste DSP-kortet i sann tid: analoge signaler leses fra en port på kortet ved valgt samplingsrate, og spilles av igjen uforstyrret. Vi finner ut hvordan vi kan endre samplingsraten samt hvordan vi kan velge mellom mikrofon og line-inngang.

(a) Lag deres eget directory på PC'en under `c:\student`. Begrens dere til dette directoriet for filene deres.

(b) Les og forstå kildefilen `talk.c` som er gjengitt i appendix A. Jeg håper den kommenterte kildefilen gir nok informasjon til å forstå programmet. Filen kan kopieres fra directory `c:\adi_21\21k\dsplab\` til deres eget directory. Dere må også kopiere filen `c_ezlab.ach` herfra. Sistnevnte inneholder en beskrivelse av oppsettet til selve kortet, og er nødvendig både for kompilator og emulator.

(c) Velg en samplingsrate, og sørg for at programmet leser fra mikrofonen. Så kan programmet kompileres. (*Merk: kompilator og emulator er ikke kompatible med Windows!*) Kompileringen gjøres som følger fra DOS:

```
> g21k -a c_ezlab.ach -o talk.ldr talk.c
```

Viktig informasjon om kompilatoren finnes i Appendix C.

(d) Hvis kompileringen gikk som den skulle (ingen feilmeldinger), kan vi nå starte emulatoren og laste opp programmet vårt:

```
> ice21k -a c_ezlab.ach -e talk.ldr
```

Se Appendix D for informasjon om emulatoren. Forviss dere om at programmet kjører som det skal.

(e) Editer programmet (bruk DOS-editoren `edit` eller en bedre editor som heter `mel`) slik at det velger signaler fra line-inngangen, kompiler på nytt og prøv systemet med sinus- og firkantsignaler fra signalgeneratoren. Sjekk at Nyquist hadde rett (dvs. at systemet slipper igjennom signaler opp til $f_s/2$. Prøv med forskjellige samplingsrater. *Merk:* utviklingskortet inneholder analoge lavpassfiltre som kutter temmelig skarpt, og som følger samplingsfrekvensen automatisk.

Oppgave 2: Filtrering

Denne oppgaven inneholder en Matlab-komponent for filterdesign samt en DSP-del som prøver ut disse filtrene i sann tid.

(a) Bruk Matlab til å generere minst fire filtre av forskjellig kaliber. Forslag: Lavpass FIR av orden 30 med $\omega_c = 0.3\pi$, båndpass FIR av orden 500 med passbånd $0.3\pi \leq \omega \leq 0.6\pi$, lavpass Butterworth IIR av orden 10 med $\omega_c = 0.3\pi$ og høypass Chebyshev IIR av orden 13 med $\omega_c = 0.5\pi$. Lag plott av forventet amplituderespons for disse filtrene.[1]

(b) Bruk Matlab-funksjonen `SAVEADI` til å lagre filterkoeffisienter til `.h`-filer. Les om `SAVEADI` i Appendix E.

(c) Implementer disse filtrene i programmet `filter.c` som finnes på `c:\adi_21\21k\dsplab\`. Implementer de to lavpassfiltrene i samme program slik at man kan sammenligne dem ved å holde bryteren `FLAG0` nede eller ikke. Undersøk amplituderesponsen ved forskjellige sinussignaler, og verifiser at filtrene virker som de skal.

(d) Bruk *firkantpulser* som inn-signal, og observer ut-signalet fra de forskjellige filtrene. Forklar utseendet på signalet, ut fra kjennskapen til hvilke harmoniske som er tilstede i firkant-toget, amplitude- og faserespons til filtrene. Spesielt skal man se hvordan lineær fase (FIR-filter) og ikke-lineær fase (IIR) genererer forskjellige bølgeformer.

[1] Det er flere grunner til ikke å interessere seg for mye om faseresponsen her – den viktigste er at "analog"-filtrene på inngang/utgang til kortet har uspesifisert men lineær fase. Vi kan derfor ikke måle absolutt faserespons til kretsen.

Appendix A: talk.c

```
/*******************************************************************
* TALKTHRU.C          Written by: Jerry Lenaz,  12/21/92           *
*                     Revised by: Matt Begg,    1/5/93             *
********************************************************************
  Ny versjon TALK.C med norske kommentarer mars 95. BR

  Dette programmet setter opp ADSP-21020 til å kopiere analog inn til
  ut. Først settes samplingsrate opp, dernest velges riktig inn-kanal
  (line-inngang eller mikrofon). For hvert innkommende sample
  genereres et interrupt, og rutinen process_input kalles opp.
  Siden all prosessering foregår i denne rutinen, består hovedprogrammet
  kun av en uendelig løkke som gjør ingenting (idle()).
********************************************************************/

#include <21020.h>    /* For the idle() command */
#include <signal.h>   /* For the interrupt command */
#include <macros.h>   /* For the DEF_PORT function */

/*
   Her settes de forskjellige portene opp (en port er koblet opp mot
   prosessoren som vanlig RAM) slik at 21020 kan aksessere dem.
   (Se filen c_ezlab.ach!)
*/

DEF_PORT(in_port,int,hip_reg0,dm);   /* analog port inn */
DEF_PORT(out_port,int,hip_reg2,dm);  /* analog port ut */
DEF_PORT(control_0,int,hip_reg4,dm); /* port for å velge signalkilde */
DEF_PORT(control_1,int,hip_reg5,dm); /* port for å sette samplingsrate */

void process_input(int);  /* prototype for interrupt-funksjonen */

/* Hovedprogram: */

void main(void)
{
  /* Velger samplingsrate. Først "vekkes" kretsen opp */
  control_1 = 0x00000000;
  while ((control_1&0x80000000) == 0x0);

  /* Dernest kan vi sette samplingsrate. Lovlige verdier er bl.a.
     0x00000000 : 8kHz
     0x00010000 : 16kHz
     0x00030000 : 32kHz
     0x000d0000 : 44.1kHz
     0x00060000 : 48kHz

     Kretsen virker slik at vi ikke bør fortsette før bit 31 (helt til
     venstre i tallet) er blitt satt. Da har vi riktig samplingsrate. */

  control_1 = 0x00010000;              /* velger 16kHz */
  while ((control_1&0x80000000) == 0x0);
```

```
  /* Velger mikrofoninngang. For line-inngang bruker vi 0x24000000. */
  control_0 = 0x64000000;

  /* Nå setter vi opp interrupt-rutinen og starter en uendelig løkke. */
  /* All prosessering foregår i process_input. */

  interrupt(SIG_IRQ3, process_input);

  while (1)
  {
    idle();
  }
}

/*
  Interrupt-rutine. Leser inn et 16-bit heltall fra analogporten,
  konverterer dette til et flyttall (i en intern variabel), og konverterer
  tilbake for utlesning. Merk at analogporten er koblet til de 16 øverste
  bitene av databussen (en int er 32 bit). Derfor må vi skifte tallet
  16 bit til høyre (operatoren >>).
*/

void process_input(int sig_number)
{
  float sample;
  sample = (float)(in_port>>16);
  out_port = ((int)sample)<<16;
}
```

Appendix B: filter.c

```
/* WHEN 950306 09:53:19**************************************************
*  FILTER.C                                         Bernt Ribbum 18/5/93 *
*************************************************************************
  FILTER.C - enkelt program som viser filtrering på ADSP-21020.
  Programmet sjekker bryteren FLAG0 for hvert sample, og velger
  enten et FIR- eller et IIR-filter.
*************************************************************************/

#include <21020.h>    /* For the idle() command */
#include <signal.h>   /* For the interrupt command */
#include <macros.h>   /* For the CIRCULAR_BUFFER and DEF_PORT functions */
#include <filters.h>  /* For the fir() and iir() routines */

DEF_PORT(in_port,int,hip_reg0,dm);   /* analog port inn */
DEF_PORT(out_port,int,hip_reg2,dm);  /* analog port ut */
DEF_PORT(control_0,int,hip_reg4,dm); /* port for å velge signalkilde */
DEF_PORT(control_1,int,hip_reg5,dm); /* port for å sette samplingsrate */

/*
  Filene fir.h og iir.h inneholder filterkoeffisienter. Disse må lages.
  Enkleste måten å gjøre dette på er å bruke rutinen SAVEADI fra Matlab.
  Da blir også konstantene FIR_ORDER og IIR_ORDER definert.
*/

#include "fir.h"
#include "iir.h"

void process_input(int); /* interrupt routine to sample */

float iirstate[IIR_ORDER+1];
float firstate[FIR_ORDER+1];

/* Merk: "volatile" er nødvendig hvis man kompilerer med opsjon -O2 */

volatile float sample;

void main(void)
{
  int i;

  for (i=0;i<=FIR_ORDER;i++) firstate[i]=0;
  for (i=0;i<=IIR_ORDER;i++) iirstate[i]=0;

  set_flag(SET_FLAG1, CLR_FLAG);
  set_flag(SET_FLAG2, CLR_FLAG);
  set_flag(SET_FLAG3, CLR_FLAG);

  control_1 = 0x00000000;
  while ((control_1&0x80000000) == 0x0);
  control_1 = 0x00010000;               /* velger 16kHz */
  while ((control_1&0x80000000) == 0x0);
```

```
  /* Velger lineinngang. */
  control_0 = 0x24000000;

  interrupt(SIG_IRQ3, process_input); /* sample interrupt */

  while (1)
  {
    idle();
    set_flag(SET_FLAG2, SET_FLAG);
    if (poll_flag_in(READ_FLAG0,RETURN_FLAG_STATE))
    {
      set_flag(SET_FLAG1, CLR_FLAG);
      sample = fir(sample,fir_b,firstate,FIR_ORDER);
    }
    else
    {
      set_flag(SET_FLAG1, SET_FLAG);
      sample = iir(sample,iir_a,iir_b,iirstate,IIR_ORDER);
    }
    set_flag(SET_FLAG2, CLR_FLAG);
  }
}

/* Interrupt-rutinen skriver her ut FORRIGE filtrerte verdi fra den
   globale variabelen sample, og henter deretter inn ny verdi som
   skal filtreres. Filtreringen foregår i hovedprogrammet. */

void process_input(int int_number)
{
  out_port = ((int)sample)<<16;
  sample = (float)(in_port>>16);
}
```

Appendix C: Kompilatoren g21k.exe

Kompilatoren er et generelt program som på Unix-vis kan kompilere C-filer og assembly-filer til objektfiler, samt lenke objekt- og biblioteksfiler til eksekverbare programmer. Kommandolinjen er generelt:

`g21k` [*opsjoner*] *innfil(er)*

Innfiler kan være av type `.c` for C-kildefiler, `.s` eller `.asm` for assemblyfiler eller `.o` for allerede kompilerte objektfiler.

De mest brukte opsjonene er (store og små bokstaver gir forskjellige opsjoner!):

- `-o` *utfil* for å navngi resultatfilen (oftest det eksekverbare programmet)
- `-c` for kun å kompilere (ikke lenke)
- `-S` for kun å generere assembly-fil (fra C-fil)
- `-g` for å lage CBUG-klart program
- `-a` *arkitekturfil* for å bruke en navngitt arkitekturfil
- `-O` for at kompilatoren skal gjøre endel optimalisering
- `-O2` for at kompilatoren skal gjøre full optimalisering
- `-l`*xxx* for å ta med biblioteket `lib`*xxx*`.o` ved lenking. Normalt sett klarer vi oss uten, men f.eks. matematikkrutiner ligger i `libm.a` og må spesifiseres eksplisitt ved bruk.
- `-h` for hjelp

Etternavnet (*extension*) på programfilen er helt fritt – vi har valgt å bruke `.ldr` for dette i eksemplene. Hvis vi skal kompilere filen `test.c` til det eksekverbare programmet `test.ldr`, med full optimalisering og bruk av arkitekturfilen `c_ezlab.ach`, skriver vi

```
> g21k -a c_ezlab.ach -o test.ldr -O2 test.c
```

Hvis vi kun vil se på assembly-koden (lage filen `test.s`), kan vi skrive

```
> g21k -a c_ezlab.ach -c -S -O2 test.c
```

Appendix D: Emulatoren ice21k.exe

Kommandolinjen for å starte programmet er:

`> ice21k -a` *arkitekturfil* `-e` *.ldr-fil*

Dette starter opp programmet på PC'en, som først gjør en test av utviklingskortet, deretter laster inn beskrivelsen av kortet som er gitt i arkitekturfilen, og tilslutt laster inn binærfilen (.ldr-filen). Brukeren presenteres deretter med et vindu som viser assemblykode for ADSP-21020. Vi skal prøve å holde oss unna assemblykoden i dette kurset, så for å starte prosessoren gjør vi følgende:

- Når programmet er lastet inn og det nederst på skjermen står "Processor halted" trykker man på RESET-knappen på utviklingskortet.
- Velg deretter **Execution/Chip Reset** med musa og vent til dette er gjort. (Merk: disse to reset-funksjonene er forskjellige, og må utføres i den nevnte orden.)
- Velg **Execution/Run** med musa (eller trykk funksjonsknappen F4). Da skal programmet kjøre, og nederst på skjermen skal "Running" blinke. Skjermen oppdateres IKKE mens prosessoren kjører.

Hvis man er interessert i å se hva som skjer i prosessoren, må den først stoppes. Deretter kan man f.eks. velge **Memory/Data** for å åpne et vindu til datahukommelsen. Her kan man kjenne igjen sine egne variabelnavn (med en understrek, "_" foran), og se verdien som enten heksadesimal, desimal eller flyttall (trykk control-t for å velge representasjon).

Man stopper emulatorprogrammet ved å velge **File/Quit** fra menyen.

Appendix E: SAVEADI og filterimplementasjon

SAVEADI er en kommando til Matlab som er laget kun for labkursene på DSP-laben. Derfor finnes filen kun på PC'ene der, og ikke under Unix. Bruken er som følger:

- For FIR-filtre (kun b-koeffisienter)

  ```
  >> saveadi('fir.h',fir_b,'fir_b')
  ```

  Parametre er filnavn, koeffisientvektor, variabelnavn. Sistnevnte bruker man siden som variabel i C-programmet f.eks. ved kall av rutinen `fir()`.

- For IIR-filtre (både a- og b-koeffisienter)

  ```
  >> saveadi('iir.h',iir_b,'iir_b', iir_a,'iir_a')
  ```

  Bruken er tilsvarende; man får her to tabeller med koeffisienter man kan bruke direkte i kall til rutinen `iir()`.

SAVEADI definerer i tillegg en konstant (enten `FIR_ORDER` eller `IIR_ORDER`) som man kan bruke direkte i programmet. (*Merk:* filterorden er én mindre enn antall koeffisienter.)

Implementering

Ved bruk av rutinene `fir` og `iir` må man selv sørge for at et buffer er tilgjengelig for filterhukommelsen. Dette må nullstilles før man begynner filtreringen, ellers trenger man ikke bekymre seg om innholdet. Bufferet må være av størrelse filterorden pluss 1, og i tillegg ligge i programhukommelsen. Dette ordner man f.eks. slik:

```
float pm iirstate[IIR_ORDER+1];
```

`pm` betyr her "program memory" – dette er Analog Devices' utvidelse av C. Når dette er satt opp, kan man filtrere et enkelt sample ved følgende kall:

```
sample = iir(sample,akoeff,bkoeff,iirstate,IIR_ORDER);
```

Legger man dette inn i en løkke i programmet sitt (som i `filter.c`), skal filtreringen gå riktig. For FIR-filtre blir kallet

```
sample = fir(sample,bkoeff,firstate,FIR_ORDER);
```