Fagprøve i serviceelektronikk for

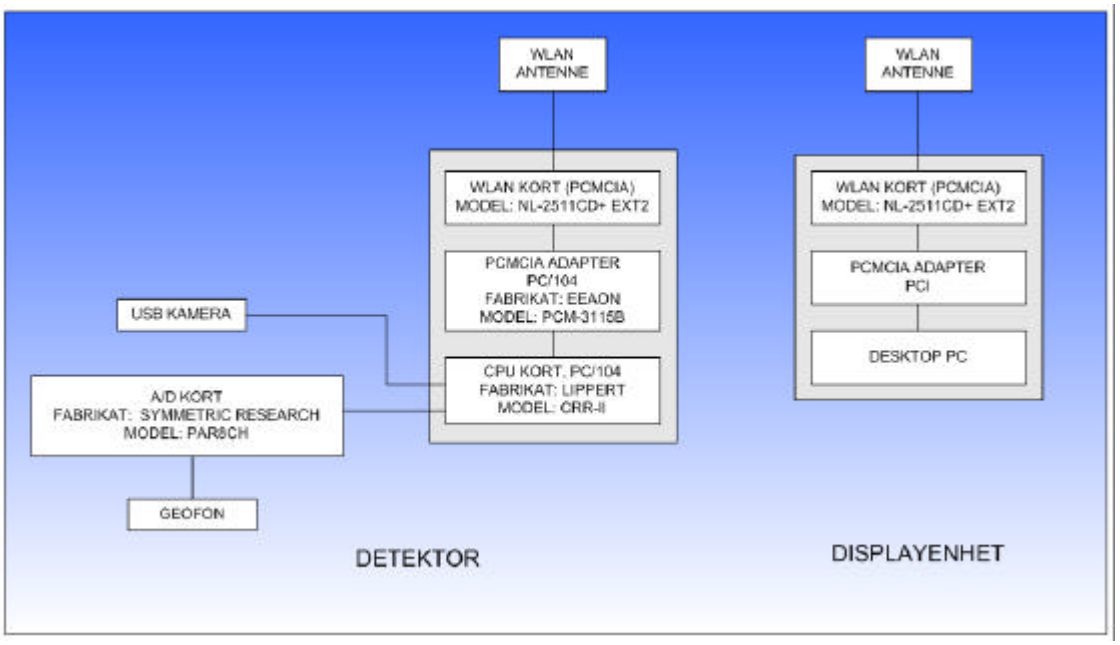
Kristian Hiim

Fordypning: Data- og kontorsystemer
Prøveperiode: 2.-6. februar 2004
Universitetet i Bergen, Institutt for geovitenskap

Innhold

Oppgave 1

1.1 Blokkdiagram



1.2 Detektor

For å få detektoren til å fungere må jeg installere et operativsystem. Siden det er lite ressurser tilgjengelig på maskinen, så skal jeg lage en embedded Linux distribusjon. Detektoren skal også ha et styreprogram, som skal ta bilder når en geofon registrerer bevegelse. Dette programmet skal kommunisere med et displayprogram som skal vise bildene som blir tatt. Displayprogrammet skal kjøre på en annen maskin, og kommunikasjonen skal skje via trådløst nettverk.

Embedded Linux

En vanlig Linux distribusjon i dag kan fort ta flere gigabytes. Embedded Linux er en betegnelse på Linux distribusjoner som er strippet ned, og optimalisert, med tanke på systemer med lite ressurser. Jeg skal sette opp en embedded Linux distribusjon. For å kunne gjøre dette trenger jeg en GNU toolchain.

Sette opp en GNU toolchain

En GNU toolchain er et ”verktøy” som kan brukes til å kompilere programvare som skal kjøres på en annen maskin. Toolchainen består av gcc (en C kompilator), glibc (en rekke grunnleggende funksjoner som de fleste programmer trenger), og binutils (programmer for å håndtere binærfiler). For å kunne kompilere glibc trenger jeg headerfiler fra Linux kjernen jeg skal bruke også.

Jeg har satt sammen en pakke som inneholder programvaren som trengs, og script for å installere dette. Denne pakken heter embedded.tar.gz og kan lastes ned fra http://www2.geo.uib.no/embedded/. Her vil du finne andre pakker relatert til fagprøven. En liste over filer i pakken, og en utskrift av scriptene er lagt ved besvarelsen. Scriptene vil i tillegg til å installere toolchainen, sette opp en katalogstruktur som vil brukes for å utvikle et rootfilsystem for detektoren.

I lister over kommandoer betyr $ at kommandoen skal kjøres som en vanlig bruker. # betyr at kommandoen kjøres som root. Det anbefales ikke å kjøre som root når en installerer en toolchain. Grunnen til dette er at det allerede finnes en lokal toolchain på maskinen. Dersom du installerer som root, og gjør en feil, kan du risikere å skrive over viktige systemfiler på maskinen. I listen over kommandoer nedenfor har jeg derfor endret rettigheter til filene, slik at brukeren min, kristian, eier katalogen, og filene. Da kan jeg jobbe som bruker kristian når jeg skal kompilere og sette opp systemet mitt.

Jeg har laget en del script, som jeg bruker i denne besvarelsen. Scriptene jeg har laget er basert på at toolchainen installeres i /opt katalogen på harddisken. Alle script, programmer og konfigurasjonsfiler jeg har laget er lagt ved i besvarelsen under Vedlegg. Her er listen over kommandoer for å starte et prosjekt:

```
$ su -
# cd /opt
# wget http://www2.geo.uib.no/embedded/embedded.tar.gz
# tar –zxvf embedded.tar.gz
# chown –R kristian.kristian embedded
# exit
$ cd embedded
$ mkdir projects
$ scripts/make-dirs detektor /opt/embedded/projects i586-linux
$ scripts/make-toolchain /opt/embedded/projects/detector i386
```

Toolchainen er nå installert i /opt/embedded/projects/detector. Detektor er navnet på prosjektet. Dette skrev jeg inn når jeg kjørte make-dirs scriptet.

Linux kjernen

Linux kjernen er hovedprogrammet i Linux som operativsystem. I planleggingsdelen hadde jeg tenkt å kompilere kjernen etter å ha gjort ferdig rootfilsystemet, men da en del av programmene er avhengig av at kjernen er ferdig konfigurert valgte jeg å gjøre dette nå. Kjernen ble lastet ned og pakket ut da jeg kompilerte glibc, fordi glibc var avhengig av headerfiler fra kjernen. Men kjernen må fremdeles konfigureres og kompileres for å kunne brukes. Det en gjør når en konfigurerer kjernen er å velge hvilke hardware det skal bygges inn støtte for, og hvilke funksjoner kjernen skal støtte. Jeg har lastet konfigurasjonsfilen som jeg laget opp på web. Her er kommandoene for å laste ned min konfigurasjon, og kompilere den, slik at den er klar til bruk.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd kernel/linux-2.4.24/
$ make mrproper
$ wget http://www2.geo.uib.no/embedded/kernel.config
$ mv kernel.config .config
$ make ARCH=i386 CROSS_COMPILER=i586-linux- menuconfig
$ make ARCH=i386 CROSS_COMPILER=i586-linux- dep
$ make ARCH=i386 CROSS_COMPILER=i586-linux- bzImage
$ make ARCH=i386 CROSS_COMPILER=i586-linux- modules
```

Etter at kjernen er kompilert kopierer jeg den til images/ katalogen i prosjektkatalogen. Jeg endrer også navn på kjernen, slik at det er lettere å se hvilke versjon av kjernen det er jeg bruker. Her er kommandoene jeg brukte for å kopiere kjernen til images/. ${PRJROOT}er en variabel som blir satt til prosjektkatalogen av . loadenv kommandoen ovenfor.

```
$ cp arch/i386/boot/bzImage ${PRJROOT}/images/bzImages-2.4.24
$ cp System.map ${PRJROOT}/images/System.map-2.4.24
$ cp .config ${PRJROOT}/images/2.4.24.config
$ make ARCH=i386 CROSS_COMPILER=i586-linux- \
INSTALL_MOD_PATH=${PRJROOT}/images/modules-2.4.24 \
modules_install
```

Rootfilsystem

For å sette sammen en embedded Linux distribusjon må man lage et rootfilsystem. Dette er en kopi av rootfilsystemet en finner på et vanlig Linux system, bare veldig nedstrippet. Alle unødvendige og overflødige funksjoner fjernes, for å få systemet så lite som mulig.

Rootfilsystemet inneholder en del standard kataloger, som har sine spesifikke funksjoner. Jeg har laget et script som lager katalogene i rootfilsystemet. Kommandoen for å generere rootfilsystemet er:

```
$ cd /opt/embedded
$ scripts/make-rootfs /opt/embedded/projects/detektor
```

Nå er katalogstrukturen i rootfilsystemet generert, og jeg kan begynne å legge inn filer. Rootfilsystemet ligger i prosjektkatalogen, under rootfs/

Legge inn devices i rootfilsystemet

Det første jeg har valgt å legge inn i rootfilsystemet mitt er devicer. Devicer er spesielle filer som brukes til å kommunisere med kjernen, og drivere for hardware. Jeg har laget et script som genererer de nødvendige devicene i rootfs/dev. Når en skal lage devicer må en være root. Vi blir derfor root når vi installerer devicene.

```
$ su –
# cd /opt/embedded
# scripts/make-devices /opt/embedded/projects/detektor
# exit
```

BusyBox

Det første programmet jeg installerer i rootfilsystemet mitt er BusyBox. I Linux finner vi vanligvis mange små grunnleggende programmer. BusyBox har samlet en nedstrippet versjon av alle disse programmene i en programfil. Ved å installere BusyBox vil en altså få de fleste grunnleggende funksjonene i Linux. Funksjonene som er tatt bort er som oftest ikke nødvendige i et embedded system, og en sparer derfor mye plass, uten at det går utover funksjonaliteten til systemet. En annen fordel med er at BusyBox kan konfigureres til å ta med bare de programmene en absolutt trenger. På denne måten kan en spare enda mer plass.

Jeg har laget en konfigurasjonfil som kan lastes ned, som slår på de nødvendige funksjonene for at Detektoren skal fungere. Her er kommandoene jeg bruker for å laste ned og konfigurere BusyBox i rootfilsystemet. Merk at jeg bruker en pre6 versjon, som er den siste tilgjengelige versjonen. Pre6 står for prerelease 6, som betyr at dette er en slags betaversjon av versjon 1.0. Jeg vil anbefale å bruke en nyere versjon av BusyBox, når det kommer ut. Men selv om det er en pre6 versjon har jeg ikke hatt noen problemer med den til nå.

```
$ cd /opt/embedded/projects/detektor
$ . loadenv
$ cd sysapps
$ wget http://busybox.net/downloads/busybox-1.00-pre6.tar.gz
$ tar –zxvf busybox-1.00-pre6.tar.gz
$ cd busybox-1.00-pre6
$ wget http://www2.geo.uib.no/embedded/busybox.config
$ mv busybox.config .config
$ make menuconfig
```

Når en kjører make menuconfig kommer det opp en meny, der en kan velge funksjoner og innstillinger for BusyBox. Ved å laste ned innstillingene mine, så trenger en ikke konfigurere gjøre så mange endringer. Men for å installere BusyBox må en velge at en skal bruke en krysskompilator. Dette gjør en under Build Options når en kjører make menuconfig. Merk av på at du vil bruke en Cross Compiler, og skriv inn katalogen der krysskompilatoren ligger. Krysskompilatoren er den versjonen av gcc som ble installert når GNU toolchainen ble installert. I mitt tilfelle er dette

/opt/embedded/projects/detektor/tools/bin/i586-linux-

Merk at det står i586-linux- etter katalogen også. Dette er prefixen til krysskompilatoren din. Det vil si at BusyBox vil legge til gcc og andre programmer slik at den kjører for eksempel i586-linux-gcc, for å kompilere kode. På den måten bruker BusyBox korrekt kompilator.

Etter at en har valgt krysskompilator må en også velge hvor en skal installere BusyBox. Det gjøres under Installation Options. Skriv inn katalogen til rootfs. I mitt tilfelle:

/opt/embeddded/projects/detektor/rootfs

Etter at konfigureringen er ferdig kan en fortsette installasjonen:

```
$ make dep
$ make
$ make install
```

Pcmcia-cs

For å kunne bruke pcmcia-kort på Detektoren, må jeg installere pcmcia-cs. Dette er en cardmanager, som håndterer automatisk lasting av drivere når pcmcia-kort blir satt inn. Den fjerner også driverne når kortene tas ut. For å laste ned og konfigurere pcmcia-cs gjorde jeg følgende:

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd sysapps
$ wget http://pcmcia-cs.sourceforge.net/ftp/pcmcia-cs-3.2.7.tar.gz
$ tar –zxvf pcmcia-cs-3.2.7.tar.gz
$ cd pcmcia-cs-3.2.7
$ ./Configure
```

Når en kjører Configure scriptet må en skrive inn katalogen til kildekoden til Linux kjernen. En må da skrive inn katalogen til kilden en brukte til å kompilere kjernen som skal brukes på Detektoren. I mitt tilfelle skriver jeg inn:

/opt/embedded/projects/detektor/kernel/linux-2.4.24

Velg så det som er standard innstillinger til du kommer til valget om hvor du skal installere kjernemoduler. Skriv da inn katalogen der du installerte kjernemodulene dine, etter at du kompilerte kjernen, og legg til /lib/modules/2.4.24. I mitt tilfelle blir det:

/opt/embedded/projects/detektor/images/modules-2.4.24/lib/modules/2.4.24

For å få kompilert pcmcia-cs med vår krysskompilator måtte jeg endre en fil der innstillingene til Makefilen lå. Jeg åpnet config.mk i en teksteditor og endret følgende linjer fra:

```
KCC=cc
UCC=cc
```

Til:

```
KCC=i586-linux-gcc
UCC=i586-linux-gcc
```

Så fortsatte jeg installasjonen:

```
$ make all
$ make PREFIX=${PRJROOT}/rootfs
$ cd ${PRJROOT}/rootfs/usr
$ rm –rf share
```

Etter at installasjonen er ferdig er det en liten endring som må til for å få pcmcia adapteret som jeg bruker på detektoren til å fungere korrekt. Åpne ${PRJROOT}/rootfs/etc/pcmcia/config og endre:

```
device "orinoco_cs"
  #class "network" module "hermes", "orinoco", "orinoco_cs"
  class "network" module "orinoco_cs"
```

Til:

```
device "orinoco_cs"
  #class "network" module "hermes", "orinoco", "orinoco_cs"
  class "network" module "orinoco_cs" opts "ignore_cis_vcc=1"
```

OpenSSH

SSH (Secure Shell) er en protokoll som brukes for å etablere en sikker tilkobling mellom maskiner. En kan logge inn på en annen maskin via ssh, og fjernstyre den, eller en kan kopiere filer med kommandoen scp. Overføringen skjer kryptert, og er derfor sikrere enn for eksempel telnet, der alt sendes ukryptert. Jeg skal installere OpenSSH som er en åpen kildekode-versjon av ssh. Jeg vil installere enkeltprogrammene sshd, ssh, sftp-server og scp, slik at jeg kan både logge meg inn i detektoren utenfra, og overføre filer til og fra.

Før en kan installere OpenSSH må en først installere to nye biblioteker som trengs for å få OpenSSH til å fungere. Dette er zlib og OpenSSL.

Her er instruksjonene for å installere zlib:

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd build-tools
$ wget http://www.gzip.org/zlib/zlib-1.2.1.tar.gz
$ tar –zxvf zlib-1.2.1.tar.gz
$ cd zlib-1.2.1.tar.gz
$ CC=i586-linux-gcc ./configure –-prefix=${TARGET_PREFIX}
$ make
$ make install
```

Her er instruksjonene for å installere OpenSSL:

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd build-tools
$ wget http://www.openssl.org/source/openssl-0.9.7c.tar.gz
$ tar –zxvf openssl-0.9.7c.tar.gz
$ cd openssl-0.9.7c
$ ./config –-prefix=${TARGET_PREFIX} compiler:i586-linux-gcc
$ make
$ make install
```

Når zlib og OpenSSL er installert, kan vi fortsette med installasjonen av OpenSSH.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd sysapps
$ wget ftp://ftp.openbsd.org/pub/OpenBSD/OpenSSH/portable/openssh-
3.7.1p2.tar.gz
$ tar –zxvf openssh-3.7.1p2.tar.gz
$ cd openssh-3.7.1p2
$ CC=i586-linux-gcc ./configure \
--host=${TARGET} –-prefix=”/usr”
$ make
$ cp sshd ${PRJROOT}/rootfs/usr/sbin
$ cp sftp-server ${PRJROOT}/rootfs/usr/sbin
$ cp scp ${PRJROOT}/rootfs/usr/bin
$ cp ssh ${PRJROOT}/rootfs/usr/bin
$ mkdir –p ${PRJROOT}/rootfs/usr/etc
$ cat > ${PRJROOT}/rootfs/usr/etc/sshd_config << ”EOF”
Subsystem   sftp   /usr/sbin/sftp-server
EOF
$ ssh-keygen –t rsa1 –f ${PRJROOT}/rootfs/usr/etc/ssh_host_key
$ ssh-keygen –t rsa –f ${PRJROOT}/rootfs/usr/etc/ssh_host_rsa_key
$ ssh-keygen –t dsa –f ${PRJROOT}/rootfs/usr/etc/ssh_host_dsa_key
$ mkdir –p ${PRJROOT}/rootfs/var/run ${PRJROOT}/rootfs/var/empty
$ chmod 755 ${PRJROOT}/rootfs/var/empty
```

OpenSSH er nå installert i rootfilsystemet.

Wireless tools

For å kunne konfigurere det trådløse nettverkskortet trenger en iwconfig. Dette programmet finner en i wireless_tools pakken. Her er kommandoene for å laste ned og installere wireless tools.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd sysapps
$ wget \
http://pcmcia-cs.sourceforge.net/ftp/contrib/wireless_tools.26.tar.gz
$ cd wireless_tools.26.tar.gz
$ make CC=i586-linux-gcc
$ cp iwconfig ${PRJROOT}/rootfs/sbin/
```

De andre programmene i pakken er ikke nødvendige, så derfor kopierer vi bare inn iwconfig.

Xawtv – ta bilder med USB kameraet

For å kunne ta bilder med USB kameraet trenger en et program som kan hente bilder fra kameraet, og lagre bildene lokalt. Jeg har valgt å bruke et program som heter streamer, som er en del av en større pakke, xawtv. Jeg laster ned hele pakken, kompilerer den, og kopierer over streamer programmet og noen filter som programmet trenger. Men før jeg kan installere xawtv må jeg installere libjpeg, som trengs for å kunne lagre jpeg-bilder. For at xawtv skal kompilere trenger jeg også biblioteket ncurses.

```
$ cd /opt/ embedded/projects/detektor/
$ . loadenv
$ cd build-tools
$ wget http://www.ijg.org/files/jpegsrc.v6b.tar.gz
$ tar –zxvf jpegsrc.v6b.tar.gz
$ cd jpeg-6b
$ CC=i586-linux-gcc ./configure \
–-target=${TARGET} –-prefix=${TARGET_PREFIX} -–enable-shared
$ cd /opt/embedded/projects/detektor/
$ make
$ make install
```

Etter at libjpeg er installert kan jeg installere ncurses.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd build-tools
$ wget ftp://ftp.gnu.org/gnu/ncurses/ncurses-5.3.tar.gz
$ tar –zxvf ncurses-5.3.tar.gz
$ cd ncurses-5.3
$ CC=i586-linux-gcc CXX=i586-linux-g++ ./configure --target=${TARGET} \
–-prefix=${TARGET_PREFIX} –-without-ada –-without-progs
$ make
$ make install
$ cd ${TARGET_PREFIX}/include
$ cp –a ncurses/* .
```

Når ncurses og libjpeg er ferdig installert kan installasjonen av xawtv begynne.

```
$ cd /opt/ embedded/projects/detektor/
$ . loadenv
$ cd sysapps
$ wget http://bytesex.org/xawtv_3.91.tar.gz
$ tar –zxvf xawtv_3.91.tar.gz
$ cd xawtv-3.91
$ CC=i586-linux-gcc CXX=i586-linux-g++ ./configure \
–-without-x –-prefix=”/usr”
$ make
```

Siden vi bare skal ha tak i streamer programmet og noen filter, så kopierer vi filene vi skal ha direkte i stedet for å kjøre make install.

```
$ cp console/streamer ${PRJROOT}/rootfs/usr/bin
$ mkdir –p ${PRJROOT}/rootfs/usr/lib/xawtv
$ cp –a libng/plugins/*.so ${PRJROOT}/rootfs/usr/lib/xawtv
```

Biblioteker

Jeg har nå installert mange programmer i rootfilsystemet mitt. Men programmene vil ikke fungere uten bibliotekene som de er koblet mot. Biblioteker er programkode som flere programmer deler, og laster inn når de trenger. På denne måten sparer en plass. I Linux heter bibliotekfilene .so. Disse filene fungerer i prinsippet på samme måte som Windows sine .dll filer. For å finne ut hva .so filer jeg trenger kan jeg kjøre en kommando som heter ldd. Den kjører jeg med programmet som parameter, for eksempel ldd rootfs/bin/busybox. Jeg vil da se hvilke filer som kreves for å kjøre programmet. Filene finnes i ${TARGET_PREFIX}/lib .

- ld-2.3.1.so
- libc-2.3.1.so
- libcrypt-2.3.1.so
- libdl-2.3.1.so
- libjpeg.so.62.0.0
- libm-2.3.1.so
- libnsl-2.3.1.so
- libnss_dns-2.3.1.so
- libnss_files-2.3.1.so
- libpthread-0.10.so
- libresolv-2.3.1.so
- libutil-2.3.1.so

Når en kopierer bibliotekene, må en også kopiere med linkene til bibliotekene, altså filene som peker på filene over. Det kan være lurt å ha med –a opsjonen når en kopierer, for at linkene skal fungere når en kopierer de. Jeg har kopiert filene over til ${PRJROOT}/rootfs/lib.

Kjernemoduler

Noen av driverne i Linux kjernen ble kompilert som kjernemoduler. Dette vil si at koden til driverne ligger i egne filer, som kan lastes inn i kjernen når det er behov for driveren. Disse filene må legges inn i /lib/modules i rootfilsystemet. Her er kommandoene jeg skrev for å legge modulene inn.

```
$ cp –a ${PRJROOT}/images/modules-2.4.24/lib/modules/ .
$ rm modules/2.4.24/build
```

Innstillinger i /etc

I /etc katalogen ligger innstillingene til systemet. Jeg har laget en fil som inneholder konfigurasjonsfilene i etc. Denne lastes ned og installeres slik:

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd rootfs/etc
$ wget http://www2.geo.uib.no/embedded/etc.tar.gz
$ tar –zxvf etc.tar.gz
$ rm etc.tar.gz
```

Oppstartsscript

Jeg har også lastet opp to oppstartscript. Det første scriptet brukes til å pakke ut et komprimert rootfilsystem til minne, og det andre scriptet mounter filsystemet og starter nettverk og sshd.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd rootfs
$ wget http://www2.geo.uib.no/embedded/linuxrc
$ chmod +x linuxrc
$ cd etc
$ mkdir init.d
$ cd init.d
$ wget http://www2.geo.uib.no/embedded/rcS
$ chmod +x rcS
```

Sette rootpassordet

For å kunne logge inn på det nye systemet må man sette et root passord.

```
$ su –
# cd /opt/embedded/projects/detektor/
# chroot rootfs/ /bin/sh
# passwd
# exit
```

Når en kjører chroot skal en få opp en melding det står BusyBox i. Kommandoer som kjøres etter dette vil kjøre som om det nye rootfilsystemet var det lokale rootfilsystemet. Når en kjører passwd for å sette passordet, vil altså passordfilene i det nye rootfilsystemet oppdateres.

Installere detektorprogrammet

Detektorprogrammet er et program som jeg laget som en del av fagprøven. Programmet registrerer signaler fra en geofon, via en A/D konverter. Når signalene går over et gitt nivå, trigger programmet og tar et bilde. Bilde sendes til et displayprogram over trådløst nettverk. Displayprogrammet vil så vise bildet.

For å bruke programmet må en først installere en driver for A/D konverteren. Jeg har modifisert driverpakken litt for at den skal virke på min embedded distribusjon.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd sysapps
$ wget http://www2.geo.uib.no/embedded/par8ch-fagprove.tgz
$ tar –zxvf par8ch-fagprove.tgz
$ cd par8ch
$ ./makeall
$ cd driver
$ mkdir ${PRJROOT}/rootfs/usr/par8ch
$ cp indriver ${PRJROOT}/rootfs/usr/par8ch
$ cp indriver.h ${PRJROOT}/rootfs/usr/par8ch
$ cp mkdevice ${PRJROOT}/rootfs/usr/par8ch
$ cp rmdriver ${PRJROOT}/rootfs/usr/par8ch
$ cp par8chkd.o ${PRJROOT}/rootfs/usr/par8ch
```

Etter at driveren er installert, kan jeg laste ned og installere detektorprogrammet.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd project
$ wget http://www2.geo.uib.no/embedded/detektor.tar.gz
$ tar –zxvf detektor.tar.gz
$ cd detektor
$ make
$ i586-linux-strip detektor
$ mkdir ${PRJROOT}/rootfs/usr/detektor
$ cp detektor ${PRJROOT}/rootfs/usr/detektor
$ cp grab.sh ${PRJROOT}/rootfs/usr/detektor
```

Da jeg testet detektorprogrammet fant jeg ut at scp var for treg til å kopiere bilder. Jeg gjorde derfor en endring i programmet, og bruker nå ftp til å overføre bildene til displayenheten.

Spare plass

Rootfilsystemet tar nå ganske stor plass. En måte å redusere plassbruken er å fjerne debugkode fra binærfiler, altså programfiler og bibliotekfiler.

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
$ cd rootfs
$ i586-linux-strip bin/*
$ i586-linux-strip sbin/*
$ i586-linux-strip usr/bin/*
$ i586-linux-strip usr/sbin/*
$ i586-linux-strip –-strip-unneeded lib/*
$ i586-linux-strip –-strip-unneeded usr/lib/xawtv/*
```

Når jeg gjorde dette reduserte jeg plassbruken fra ca 24 MB, til ca 5,2 MB.

Komprimere rootfilsystemet

For å spare enda mer plass har jeg valgt å komprimere rootfilsystemet til en fil. Denne filen vil så pakkes opp i minne, når detektoren starter opp. Når jeg skal gjøre dette blir jeg root, fordi jeg må sette rettighetene på filene til root, før de legges inn på detektoren. En må alltid være mer forsiktig når en arbeider som root, fordi en risikerer å ødelegge systemfiler viss en skulle gjøre feiltagelser. Disse instruksjonene må gjøres om igjen dersom det gjøres endringer i rootfilsystemet.

```
$ su -
# cd /opt/embedded/projects/detektor/
# mkdir loopback
# dd if=/dev/zero of=tmp/rootfs.img bs=1024 count=10000
# mke2fs –F tmp/rootfs.img
# mount –o loop tmp/rootfs.img loopback
# cp –a rootfs/* loopback/
# chown –R root.root loopback/
# umount loopback
# rm –rf loopback
# gzip tmp/rootfs.img
```

Rootfilsystemet er nå komprimert, og klart for å legges inn på Compact Flash kortet, sammen med Linux kjernen.

Compact Flash kortet

Detektoren skal bruke et Compact Flash kort. For å kunne legge filene som trengs over på kortet bruker jeg et CF til pcmcia adapter. Jeg setter det inn i pcmcia-adapteret som jeg har på PC-en min, og kortet dukker opp som /dev/hde

Først må kortet partisjoneres riktig. Jeg har valgt å lage to partisjoner på kortet, en liten som vil inneholde Linux kjernen og det komprimerte rootfilsystemet, og en som skal kunne inneholde informasjon som skal kunne endres, og ivaretas når maskinen starter på nytt igjen.

For å partisjonere kortet bruker jeg Linux sin versjon av fdisk. Etter at jeg har partisjonert kortet gjør jeg følgende for å legge inn filene som trengs for å starte opp: Merk at jeg dette gjøres som root.

```
# mke2fs /dev/hde1
# mke2fs /dev/hde2
# mkdir /mnt/flash
# mount /dev/hde1 flash
# cd /opt/embedded/projects/detektor/images
# cp bzImage-2.4.24 /mnt/flash
# cp System.map-2.4.24 /mnt/flash
# cp /boot/boot.b /mnt/flash
# cd ../tmp
# mv rootfs.img.gz /mnt/flash
```

Etter at filene er kopiert inn på CF kortet, må en legge inn en bootloader. Jeg har valgt å bruke lilo, da denne er enkel og liten. Jeg har laget en lilo-konfigurasjonsfil som installerer lilo på kortet. Her viser jeg hvordan jeg laster ned konfigurasjonen og installerer den.

```
# cd /opt/embedded/projects/detektor/bootldr
# wget http://www2.geo.uib.no/embedded/lilo.embedded.conf
# lilo –C lilo.embedded.conf
# umount /mnt/flash
```

Vær veldig forsiktig når du kjører lilo. Hvis du ikke bruker denrette konfigurasjonsfilen, kan du risikere å skrive over din egen bootloader.

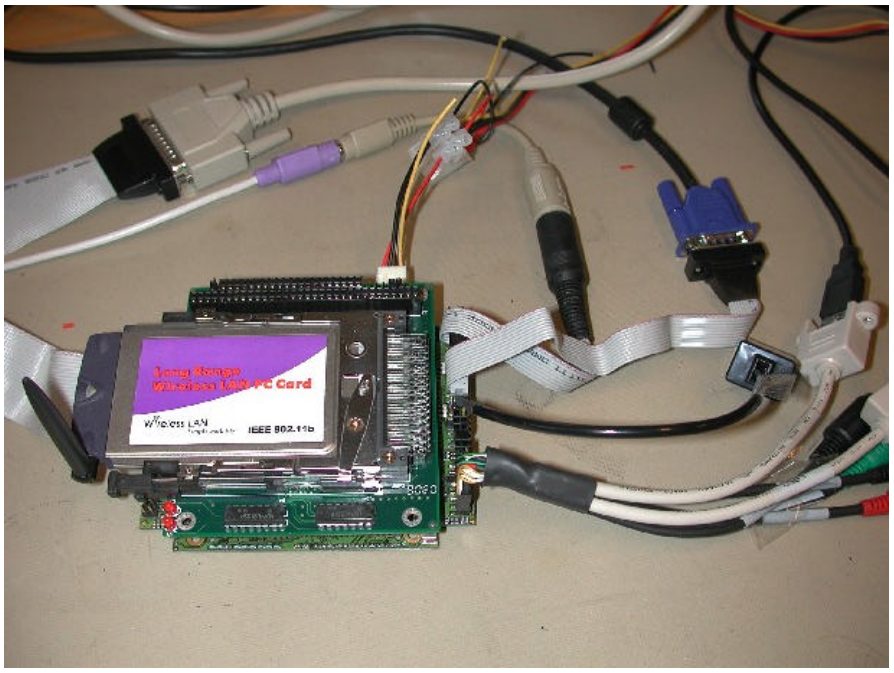
Koble opp detektoren

Detektoren er består av et PC/104 CPU kort, et PC/104 PCMCIA adapter, A/D koverter, geofon, USB kamera, trådløst nettverkskort og en strømforsyning. Strømforsyningen jeg brukte var strømforsyningen fra en gammel PC.

Jeg koblet PC/104 CPU kortet etter dokumentasjonen som følger med kortet. Jeg koblet også til skjerm og tastatur for å kunne teste at enheten fungerer.

Enheten kan senere endres til å kunne automatiseres ved å endre oppstartsscriptet, slik at displayprogrammet kjører automatisk ved oppstart. En vil da ikke trenge tastatur eller skjerm.

Etter at PC/104 CPU kortet var koblet opp, satte jeg på PC/104 PCMCIA adapteret, og satte det trådløse nettverkskortet i.



PC/104 CPU kort med kabler, pcmcia adapter og trådløst nettverk.

Videre koblet jeg til A/D konverteren og USB kameraet, og koblet geofonen til A/D konverteren.


A/D konverter, USB kamera og geofon

Så var det bare å koble strøm til PC/104 CPU kortet, og til A/D konverteren.

Starte opp fra kortet

Etter at en har installert all programvaren på CF kortet, er det bare å sette kortet i Lippert CPU-kortet, og starte det opp. Compact Flash kortet settes inn i adapteret som er montert på undersiden av CPU-kortet.

Når en kobler til strøm vil maskinen starte som en normal Linux box, og vil gi en login. Der kan en logge inn som root, med det passordet en satte tidligere.

Alle endringer som gjøres på rootfilsystemet vil forsvinne når en starter på nytt igjen, fordi rootfilsystemet er pakket ut til en ramdisk som forsvinner når en slår av maskinen.

Starte detektorprogrammet

Etter at en har logget inn på detektorenheten, er det bare å starte detektorprogrammet.

```
# cd /usr/detektor
# ./detektor
```

Programmet vil nå starte, og vente på at displayprogrammet skal koble seg til.

1.3 Displayenhet

Jeg vil sette opp utviklingsmaskinen min som displayenhet. For å få den til å fungere vil jeg vil installere et trådløst nettverkskort, starte en ftp-server, lage en bruker for displayenheten, og installere displayprogrammet.

Trådløst nettverkskort

Det trådløse nettverkskortet satte jeg inn i pcmcia-adaptere som jeg har på min maskin. Deretter konfigurerte jeg det med disse kommandoene:

```
$ su –
# iwconfig eth1 mode ad-hoc
# iwconfig eth1 channel 3
# iwconfig essid ELAB nick Prism2
# iwconfig rate 11Mb
# ifconfig eth1 10.0.0.20 netmask 255.0.0.0 broadcast 10.255.255.255
```

Disse innstillingene vil bare vare så til maskinen starter på nytt igjen. Siden jeg ikke har trådløst nettverk på min maskin til vanlig, har jeg valgt å ikke legge innstillingene inn i oppstarten av maskinen. Dette har jeg derimot gjort på detektorenheten. Innstillingene står i /etc/init.d/rcS i rootfilsystemet til detektoren.

FTP-server

Jeg hadde egentlig ikke tenkt å bruke ftp til overføring av filer, men da scp viste seg å være tregt på detektoren, valgte jeg å gå over til ftp.
I min standard Red Hat 9 distribusjon av Linux var det et installert et ftp-program som heter vsftpd. Jeg oppgraderte dette til siste versjon, tilgjengelig ved å søke på http://rpmfind.com/. Etterpå startet jeg den med kommandoen (kjøres som root):

```
# service vsftpd start
```

Standardinnstillingene i programmet var ok, så jeg gjorde ingen endringer der.

Installere og kjøre displayprogrammet

Jeg valgte å lage en ny bruker for displayenheten, som detektoren kan bruke når den skal overføre filer til displayenheten. Deretter installerte jeg displayprogrammet. Detektoren trenger denne brukeren for å fungere, fordi den er programmert til å koble seg til her. (Se grab.sh scriptet i detektoren).

```
$ su –
# useradd fagprove
# passwd fagprove
```

Som passord valgte jeg fagprove. Dette er ikke et sikkert passord, men siden dette kun skal brukes når en tester, så valgte jeg et enkelt passord.

```
# su – fagprove
$ wget http://www2.geo.uib.no/embedded/display.tar.gz
$ tar –zxvf display.tar.gz
$ cd display
$ ./build.sh
```

Etter at displayprogrammet er installert kan en starte displayprogrammet. Detektorprogrammet må kjøre for at displayprogrammet skal virke.

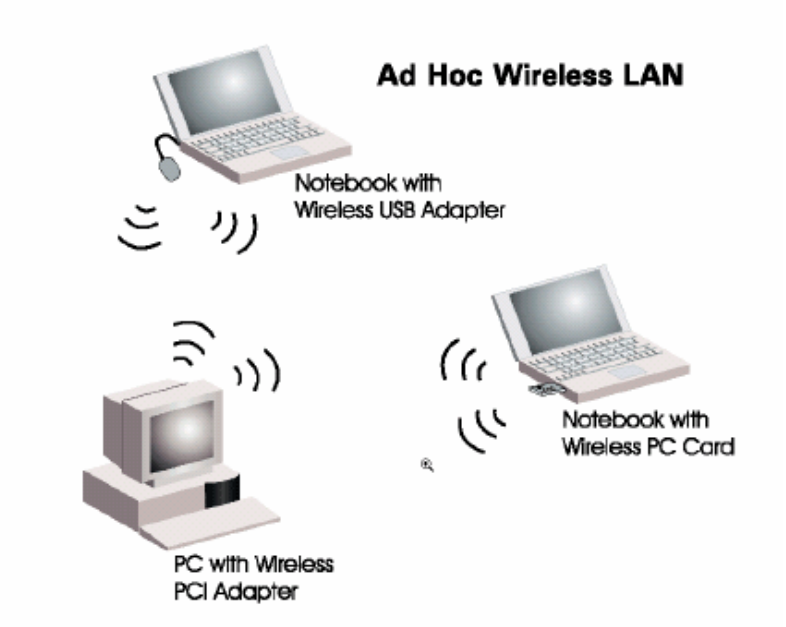
```
$ ./display
```

Når begge programmene kjører, vil detektorprogrammet ta bilder, og sende til displayenheten. Displayprogrammet vil vise det siste bilde som er tatt.

1.4 WLAN

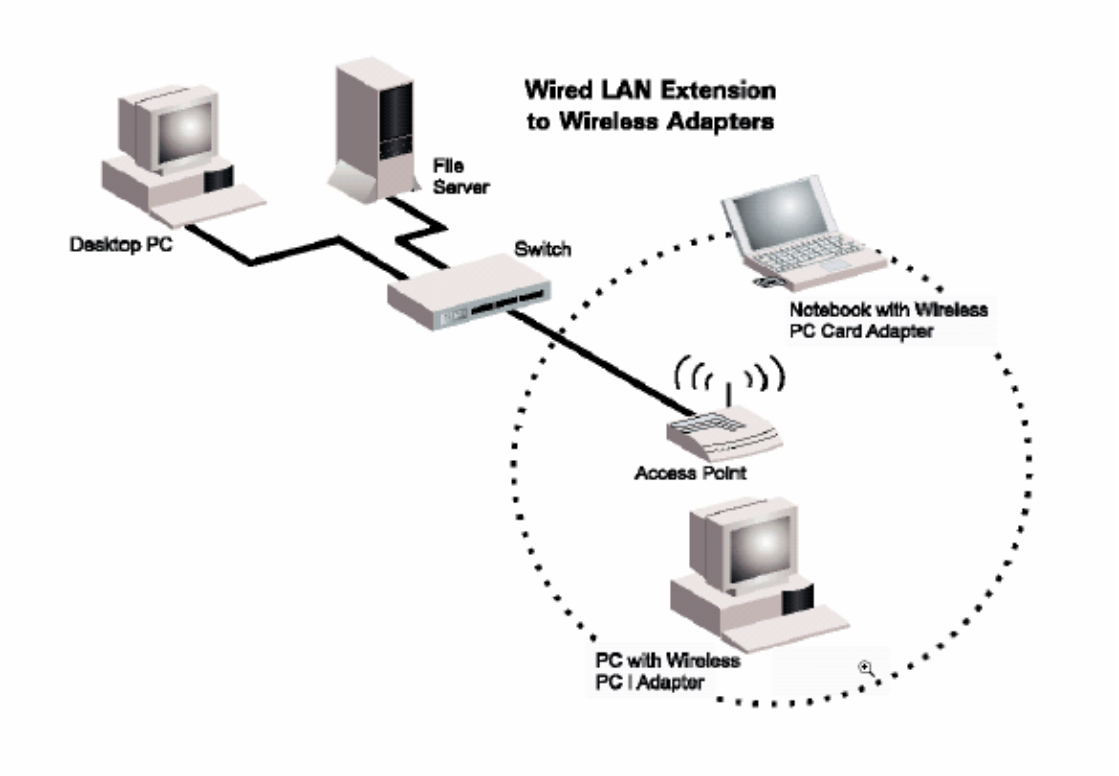
Infrastruktur og ad-hoc modus

Når det trådløse nettverkskortet fungerer i ad-hoc modus, kommuniserer det direkte med de enhetene det finner innenfor sin rekkevidde. På denne måten får vi en enkel løs kobling som kan egne seg for små lokale nettverk. Kommunikasjonstypen blir det vil kaller peer to peer, altså alle maskinene snakker direkte med andre maskiner.



Eksempel på ad-hoc nettverk

Infrastruktur modus trenger enten et aksesspunkt eller en basestasjon for å kommunisere mellom maskinene. Alle maskiner kommuniserer via aksesspunktet / basestasjonen, som i et vanlig nettverk med svitsj. De ulike aksesspunktene og basestasjonene kommuniserer seg imellom, og en kan på den måten nå maskiner som er utenfor den fysiske rekkevidden til det trådløse kortet. Aksesspunkt kan også ofte fungerer som gateway / ruter, slik at maskiner med trådløst nettverkskort kan koble seg til en internettforbindelse via den.



Eksempel på infrastruktur med aksesspunkt

Sikkerhet i trådløse nettverk

Trådløse nettverk har den fordelen at en er mobil, og slipper kabler. Men lufta er fri for alle, og signalene som går via lufta kan snappes opp av andre. Det er også mulig at andre kan benytte internett som du har delt via trådløst nettverk.

En måte å hindre at uvedkommende snoker i nettverket er å kryptere nettverket. En installerer da en nøkkel på hver av maskinene, som trengs for å forstå de krypterte meldingene som går over nettverket. Uten denne nøkkelen vil en verken kunne forstå eller kommunisere med maskiner på det trådløse nettverket.

Oppgave 2

2.1 Reléstasjon

Jeg skal sette opp en reléstasjon, som skal rute trafikk mellom detektoren og displayenheten. Dette skal jeg gjøre ved hjelp av å installere uOLSRd på alle tre maskinene. Dette er et program som håndterer ruting av IP pakker mellom maskinene automatisk, slik at detektoren og displayenheten kan kommunisere selv om de ikke er innen fysisk rekkevidde, så lenge begge har kontakt med reléstasjonen

Trådløst nettverk på reléstasjonen

For å sette opp trådløst nettverk på reléstasjonen, som er en bærbar PC med Linux installert, vil jeg gå fram på samme måte som jeg gjorde i forrige oppgave da jeg satte opp trådløst nettverk på min utviklingsmaskin. Den eneste forandringen er at jeg vil bruke IP-nummer 10.0.0.15 på den bærbare PC-en.

Installere uOLSRd på reléstasjonen

Installasjonen av uOLSRd er lik både på reléstasjonen og displayenheten, da begge disse maskinene er satt opp ganske likt. Det jeg gjør her for å installere uOLSRd gjør jeg også på utviklingsmaskinen. uOLSRd kan lastes ned fra http://www.uolsr.org/. Installasjonen er rett frem:

```
$ tar –jxvf uolsrd-0.3.6.tar.bz2
$ cd unik-olsrd-0.3.6
$ make
$ su –m
# make install
```

Deretter redigerte jeg /etc/olsrd.conf og forandret følgende innstilling til:

```
DEBUG 0
INTEFACES eth1
```

De andre innstillingene lot jeg stå som de var. Etter dette er det bare å starte uOLSRd med kommandoen olsrd som root. uOLSRd må kjøre på alle maskinene som skal kommunisere via den dynamiske rutingen.

Installere uOLSRd på detektoren

For å installere uOLSRd på detektoren må jeg kompilere den for detektoren. Jeg flytter uolsrd-pakken jeg lastet ned til /opt/embedded/projects/detektor/sysapps og gjør som følger:

```
$ cd /opt/embedded/projects/detektor/
$ . loadenv
```

```
$ cd sysapps
$ tar –jxvf uolsrd-0.3.6.tar.bz2
$ cd unik-olsrd-0.3.6
```

Så redigerer jeg Makefilen og setter CC til i586-linux-gcc, som er min krysskompilator. Etter det fortsetter jeg installasjonen.

```
$ make
$ i586-linux-strip bin/olsrd
$ cp bin/olsrd ${PRJROOT}/rootfs/usr/sbin
$ cp files/olsrd.conf.default ${PRJROOT}/rootfs/etc/olsrd.conf
```

Etter å ha installert filene gjør jeg samme forandringen i olsrd.conf i rootfs/etc som jeg gjorde i den lokale olsrd.conf.

For at olsrd skal starte automatisk, må en også legge den til i oppstartsscriptet. Følgende kommando starter olsrd, og kan skrives inn nederst i rootfs/etc/init.d/rcS:

```
/usr/sbin/olsrd
```

Kommandoen starter olsrd i bakgrunnen, og sender output til terminal 2.

Etter at olsrd er lagt inn i rootfilsystemet er det bare å følge instruksjonene i forrige oppgave for å komprimere og installere rootfilsystemet på nytt på CF kortet, med noen unntak: Du trenger ikke formatere CF kortet på nytt igjen, eller legge inn kjernen på nytt. Det holder å kopiere inn det nye rootfilsystemet (rootfs.img.gz), og kjøre lilo på nytt, slik at endringene blir registrert.

2.2 Forsterker/filter for geofon

S/N – Signal to noise

S/N betyr "Signal to noise" forhold angitt i dB. Vi kan se på det som forholdet mellom den største amplitydeverdi forsterkeren kan gi ut (før forvrengning inntreffer) og støyen som forsterkeren selv lager (dynamikkområdet).

Støymåling av forsterkere er et omfattende emne og det er her bare tid til en forenklet tilnærming.

Ved å måle utgangssignalet når inngangen er kortsluttet kan man se hvilken støy forsterkeren lager.

Avlesning med et Fluke model 87 "True RMS Multimeter" (på AC innstilling) gir 1.0 mVrms. I henhold til datablad på forsterkeren, under avsnittet "Output Characteristics" er Maximum voltage = +/- 7 V peak. Omregnet til rms (forutsatt sinussignal) blir dette 7/SQR(2) = 4.95 Vrms.

Dette gir:
S/N = 20 log (4.95 Vrms / 1.5 mVrms) = 73.9 dB.

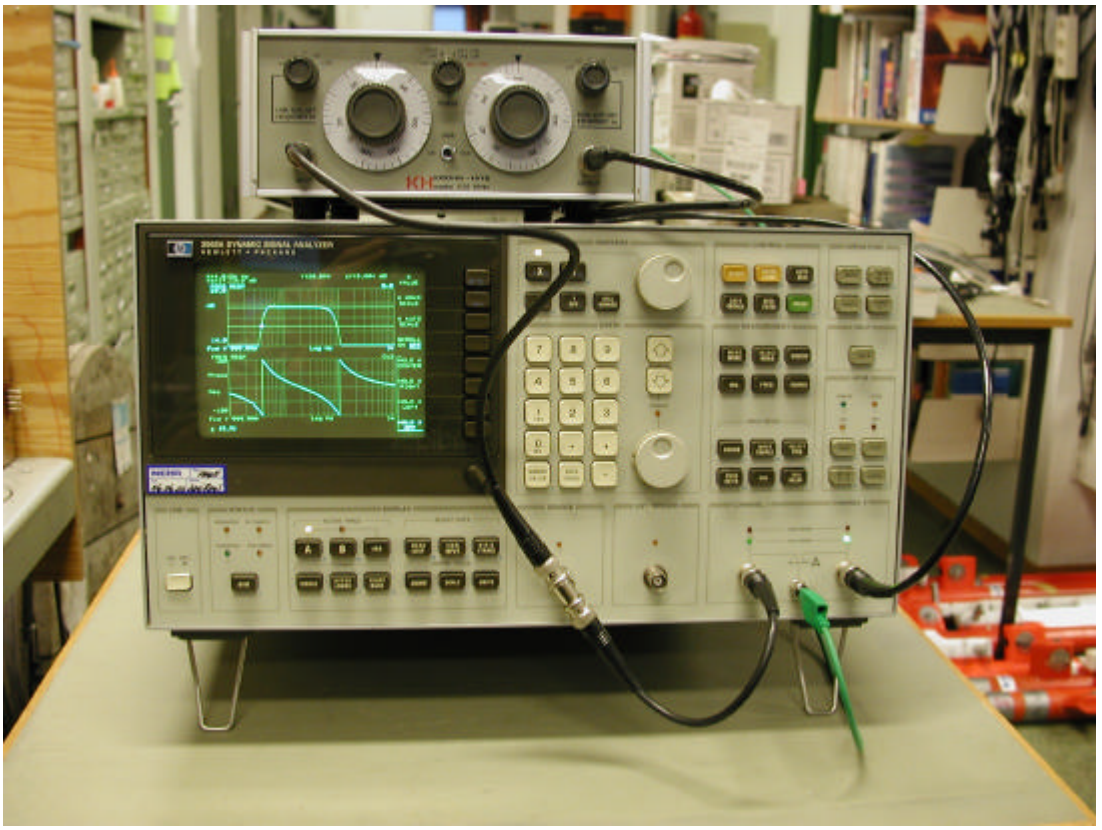
"Hum and Noise" er under "Output Characteristics" oppgitt til < 200 uV.
Tolkes dette som rms støynivå får vi:
S/N = 20 log (4.95 Vrms / 200 uVrms) = 87.9 dB

Opptak av støymålinger bør være relatert til båndbredden til forsterkeren.
Fluke multimeteret måler rms støyen innenfor et bredt frekvensområde.
Dette kan forklare noe av forskjellen.

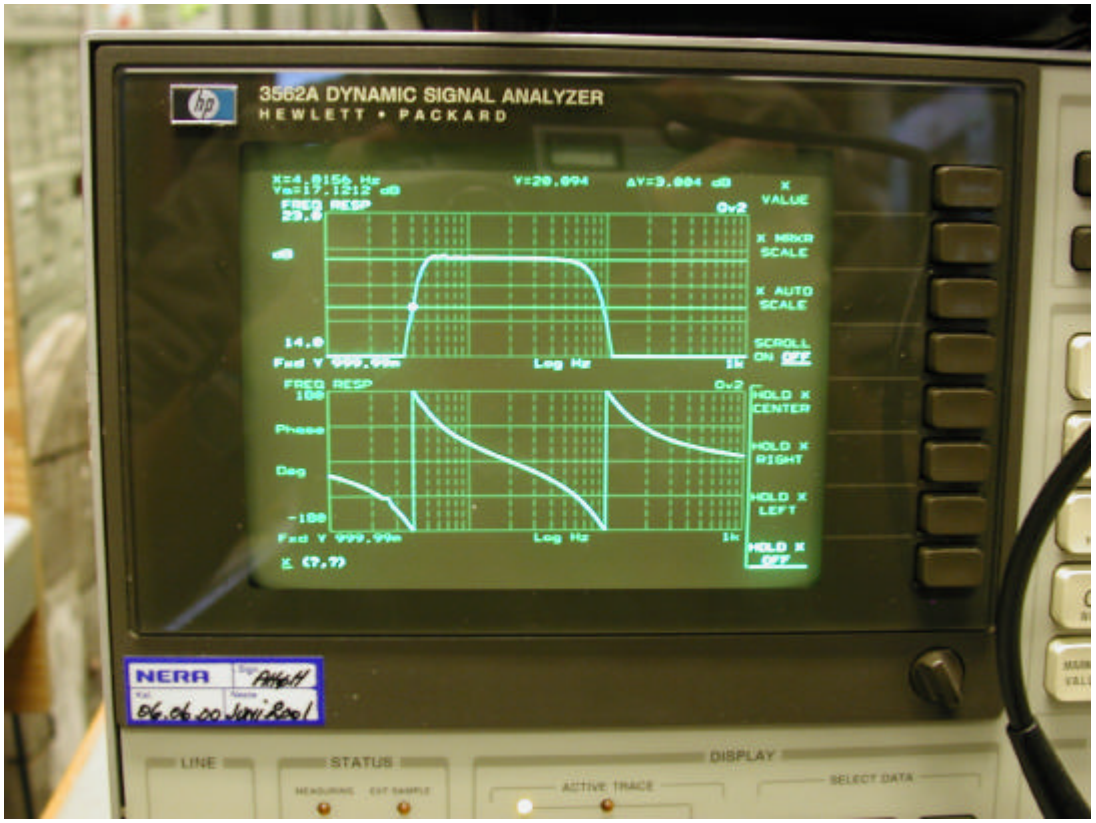
En 24 bits A/D konverter har et teoretisk dynamikkområde på
20 log(2exp24) = 144.5 dB. Når samplingsraten stiger synker dette.
I databladet for PAR8CH A/D-konverteren er det angitt typisk 19 bits oppløsning
ved 1 kHz samplingsrate, som gir dynamikkområde på 20 log(2exp19) = 114.4 dB.

Konklusjonen er at forsterkeren nok kunne vært bedre tilpasset A/D konverteren.
Selv med det S/N nivå som databladet antyder (ca. 88 dB) mister man ganske
mye dynamikkområde.

Måling av amplityde- og faserespons



Bildet viser oppkoblingen for å måle amplityde- og faserespons



Bildet viser resultatet av målingen

Anti-alias filter

Samplingsteoremet sier at man minst må ha en dobbelt så stor samplingsrate i forhold til frekvensinnholdet av det signalet man vil digitalisere. Frekvenskomponenter som ligger høyere enn halve samplingsraten vil bli "speilet" slik at de blir blandet med komponenter som ligger under halve samplingsraten - vi får et alias.

Anti-alias filterets rolle er altså å forhindre at det er frekvenskomponenter som er over halv samplingsfrekvens. Men siden filtrene ikke har en absolutt stopp-frekvens - de har en helning i amplitydekarakteristikken gitt ved filterets orden - må man heller si at de må dempe signaler over halv samplingsrate med en bestemt dB. Antall dB er relatert til A/D-konverteren som følger etter anti-alias filteret.

I et ideelt tilfelle bruker vi den oppgitte verdi på 19 bits oppløsning ved 1 kHz, 114.4 dB. Altså burde anti-alias filteret dempe signaler med frekvens på halve samplingsraten med minst denne verdien.

I praksis fant vi at anti-alias filteret hadde dårligere dynamikkområde, så la oss si at signalet må være dempet med 80 dB ved halve samplingsraten.

Ved 100 Hz er signalet dempet med 3 dB. Vi trenger i tillegg 80 - 3 = 77 dB dempning. Et 4. ordens filter demper med 80 dB/dekade. Dermed trenger vi 77/80 = 0.96 dekader høyere enn 100 Hz, noe som blir 960 Hz. Samplingsraten må være den doble av denne, altså 1920 Hz.

Oppgave 3

Arbeidsgiveren

Arbeidsgiveren har i følge Arbeidsmiljøloven (AML) hovedansvaret for arbeidsmiljøet. Arbeidsgiveren skal sørge for at arbeid blir planlagt, organisert og utført i samsvar med bestemmelsene i AML. Arbeidsgiveren skal ivareta arbeidstakernes sikkerhet, helse og velferd.

Med dette som utgangspunkt skal arbeidsgiver sette i verk tiltak som å

- utrede arbeidsskader
- opprette handlingsplaner sammen med arbeidstakere og verneombud
- organisere arbeidstilpassings- og rehabiliteringsarbeidet
- passe på at arbeidsmiljøutvalget deltar i arbeidsmiljøarbeidet

Arbeidsgiver plikter å melde fra til arbeidstilsynet og nærmeste politimyndighet dersom en arbeidstaker rammes av en arbeidsulykke med død eller alvorlig skade som følger.

Arbeidstakeren

Arbeidstakeren skal

- ta del i arbeidsmiljøarbeidet
- delta i gjennomføring av tiltak
- følge gitte forskrifter
- være forsiktig slik at en selv og andre ikke blir skadd
- underrette arbeidsgiveren om det oppstår en alvorlig fare

Arbeidstakeren har rett til

- permisjon fra arbeid i situasjoner som arbeidsmiljøloven beskriver, som i forbindelse med svangerskap og fødsel, ved barns sykdom, ved utføring av offentlige verv, og ved militærtjeneste
- redusert arbeidstid dersom arbeidstakeren har helsemessige, sosiale eller andre vektige velferdsgrunner behov for dette
- å bli fritatt fra overtidsarbeid og merarbeid når arbeidsgiver av helsemessige eller vektige sosiale grunner ber om det
- minst en hvilepause dersom arbeidstida er over 5 ½ time i døgnet
- tillegg i lønn ved overtidsarbeid
- vern mot usaklig oppsigelse
- skriftlig attest når arbeidstaker fratrer etter lovlig oppsigelse

Vedlegg

Her er installasjonsscript, oppstartsscript, konfigurasjonsfiler og programkode som jeg har laget lagt ved.

Installasjonsscript

Installasjonsscriptene blir brukt til å sette opp en Linux embedded distribusjon.

make-dirs

```
#!/bin/sh

# Check if we supplied some arguments
if [ -n "$1" ] && [ -n "$2" ] && [ -n "$3" ]
then
  PROJECT=$1
  EMBEDDEDROOT=$2
  TARGET=$3
  if [ -d "${EMBEDDEDROOT}" ]
  then
    PRJROOT=${EMBEDDEDROOT}/${PROJECT}
    echo "Making project ${PROJECT} directories in ${PRJROOT} for
target ${TARGET}"
  else
    echo "Error: ${EMBEDDEDROOT} does not exist!"
    exit 1
  fi
else
  echo "Usage: make-dirs project-name embedded-root target-plattform"
  echo "Example:"
  echo "  make-dirs pc104-project /opt/embedded i386-linux"
  exit 1
fi

echo -n "Creating main project directory... "
mkdir ${PRJROOT}
if [ -e "${PRJROOT}" ]
then
  echo "done."
  cd ${PRJROOT}
  PRJROOT=`pwd`
else
  echo "Error: Could not create directory!"
  exit 1
fi

echo -n "Creating project subdirectories... "
mkdir bootldr
mkdir build-tools
mkdir debug
mkdir doc
mkdir images
```

```
mkdir kernel
mkdir project
mkdir rootfs
mkdir sysapps
mkdir tmp
mkdir tools
echo "done."

echo -n "Creating workspace variable script... "
cat > ${PRJROOT}/loadenv <<EOF
export PROJECT=${PROJECT}
export PRJROOT=${PRJROOT}
export TARGET=${TARGET}
export PREFIX=\${PRJROOT}/tools
export TARGET_PREFIX=\${PREFIX}/\${TARGET}
export PATH=\${PREFIX}/bin:\${PATH}
EOF
echo "done."
```

make-toolchain

```
#!/bin/sh

# Source directory and source files used to make toolchain
SOURCEDIR=/opt/embedded/sources
KERNELVERSION=2.4.24
KERNELSOURCE=${SOURCEDIR}/linux-${KERNELVERSION}.tar.bz2
KERNELDEFAULTCONFIG=${SOURCEDIR}/linux.config
BINUTILSVERSION=2.14
BINUTILSSOURCE=${SOURCEDIR}/binutils-${BINUTILSVERSION}.tar.bz2
GCCVERSION=3.2.2
GCCSOURCE=${SOURCEDIR}/gcc-${GCCVERSION}.tar.bz2
GLIBCVERSION=2.3.1
GLIBCSOURCE=${SOURCEDIR}/glibc-${GLIBCVERSION}.tar.bz2
GLIBCTHREADSOURCE=${SOURCEDIR}/glibc-linuxthreads-
${GLIBCVERSION}.tar.bz2

# Check if we supplied some arguments
if [ -n "$1" ] && [ -n "$2" ]
then
  cd $1
  if [ -e loadenv ]
  then
    . loadenv
  else
    echo "Error: Unable to load enviroment variables"
    exit 1
  fi
# ARCH (needed for installing kernel headers) (this should probably be
a parameter)
  TARGETARCH=$2
else
  echo "Usage: make-toolchain project-root kernel-arch"
  echo "Example:"
  echo "  make-toolchain /opt/embedded/projects/test-project i386"
  echo " "
```

```
  echo "Use kernel-arch i386 for x86"
  exit 1
fi

echo Making toolchain in ${PRJROOT} for ${TARGET}
cd ${PRJROOT}

echo -n "Extracting kernel source... "
cd kernel
tar -jxvf $KERNELSOURCE > /dev/null
cd ..
echo "done."

cd build-tools

echo -n "Extracting binutils source... "
tar -jxvf $BINUTILSSOURCE > /dev/null
echo "done."

echo -n "Extracting gcc source... "
tar -jxvf $GCCSOURCE > /dev/null
echo "done."

echo -n "Extracting glibc source... "
tar -jxvf $GLIBCSOURCE > /dev/null
cd glibc-${GLIBCVERSION} > /dev/null
tar -jxvf $GLIBCTHREADSOURCE > /dev/null
cd ..
echo "done."

cd ${PRJROOT}
echo "Installing kernel headers..."
cd kernel/linux-${KERNELVERSION}
make mrproper
cp ${KERNELDEFAULTCONFIG} .config
make ARCH=${TARGETARCH} CROSS_COMPILE=${TARGET}- oldconfig
make ARCH=${TARGETARCH} CROSS_COMPILE=${TARGET}- clean dep
mkdir -p ${TARGET_PREFIX}/include
cp -r include/linux/ ${TARGET_PREFIX}/include
cp -r include/asm/ ${TARGET_PREFIX}/include/asm
cp -r include/asm-generic/ ${TARGET_PREFIX}/include
echo "Done installing kernel headers."

echo "Building toolchain"
cd ${PRJROOT}/build-tools

echo "Building binutils..."
mkdir -p build-binutils
cd build-binutils
../binutils-${BINUTILSVERSION}/configure --target=${TARGET} --
prefix=${PREFIX}
make
make install
cd ..
echo "Done building binutils."

echo "Installing glibc headers..."
```

```
mkdir -p build-glibc-headers
cd build-glibc-headers
../glibc-${GLIBCVERSION}/configure --host=$TARGET --prefix="/usr" --
enable-add-ons --with-headers=${TARGET_PREFIX}/include
make cross-compiling=yes install_root=${TARGET_PREFIX} prefix=""
install-headers

mkdir -p ${TARGET_PREFIX}/include/gnu
touch ${TARGET_PREFIX}/include/gnu/stubs.h

echo "Done installing glibc headers."

echo "Building and installing bootstrap gcc"
cd ${PRJROOT}/build-tools/
mkdir -p build-boot-gcc
cd build-boot-gcc
../gcc-${GCCVERSION}/configure --target=${TARGET} --prefix=${PREFIX} --
disable-shared --with-headers=${TARGET_PREFIX}/include --with-newlib --
enable-languages=c
make all-gcc
make install-gcc
echo "Done installing bootstrap gcc"

echo "Building and installing glibc"
cd ${PRJROOT}/build-tools
mkdir -p build-glibc
cd build-glibc
CC=${TARGET}-gcc ../glibc-${GLIBCVERSION}/configure --host=$TARGET --
prefix="/usr" --enable-add-ons --with-headers=${TARGET_PREFIX}/include
make
make install_root=${TARGET_PREFIX} prefix="" install
cd ${TARGET_PREFIX}/lib
mv libc.so libc.so.orig
cp ${SOURCEDIR}/libc.so .
echo "Done installing glibc"

echo "Building and installing gcc"
cd ${PRJROOT}/build-tools
mkdir -p build-gcc
cd build-gcc
../gcc-${GCCVERSION}/configure --target=$TARGET --prefix=${PREFIX} --
enable-languages=c,c++
make all
make install
cd ${PREFIX}/${TARGET}/bin
mkdir -p ${PREFIX}/lib/gcc-lib/${TARGET}/${GCCVERSION}
mv as ar gcc ld nm ranlib strip ${PREFIX}/lib/gcc-
lib/${TARGET}/${GCCVERSION}
for file in as ar gcc ld nm ranlib strip
do
ln -s ${PREFIX}/lib/gcc-lib/${TARGETARCH}-linux/${GCCVERSION}/$file .
done
echo "Done installing gcc"

echo "Done installing toolchain"
```

make-rootfs

```
#!/bin/sh

# Check if we supplied some arguments
if [ -n "$1" ]
then
  cd $1
  if [ -e loadenv ]
  then
    . loadenv
  else
    echo "Error: Unable to load enviroment variables"
    exit 1
  fi
else
  echo "Usage: make-rootfs project-root"
  echo "Example:"
  echo "  make-rootfs /opt/embedded/projects/test-project"
  exit 1
fi

echo -n "Creating root file system in ${PRJROOT}/rootfs ..."

cd ${PRJROOT}/rootfs
mkdir bin dev etc lib proc sbin tmp usr var root mnt
chmod 1777 tmp
chmod 700 root
mkdir usr/bin usr/lib usr/sbin
mkdir var/lib var/lock var/log var/run var/tmp
chmod 1777 var/tmp

echo "done."
```

make-devices

```
#!/bin/sh

# Check if we supplied some arguments
if [ -n "$1" ]
then
  cd $1
  if [ -e loadenv ]
  then
    . loadenv
  else
    echo "Error: Unable to load enviroment variables"
    exit 1
  fi
else
  echo "Usage: make-devices project-root"
  echo "Example:"
  echo "  make-devices /opt/embedded/projects/test-project"
  exit 1
fi
```

```
if [ -d "${PRJROOT}/rootfs/dev" ]
then
  echo -n "Making devices in ${PRJROOT}/rootfs/dev... "
else
  echo "Error: No rootfs/dev directory found."
  exit 1
fi

cd ${PRJROOT}/rootfs/dev

mknod -m 600 console c 5 1
mknod -m 660 hda b 3 0
mknod -m 660 hda1 b 3 1
mknod -m 660 hda2 b 3 2
mknod -m 660 initrd b 1 250
mknod -m 600 mem c 1 1
mknod -m 666 null c 1 3
mknod -m 666 ptmx c 5 2
mknod -m 660 ram0 b 1 0
mknod -m 660 ram1 b 1 1
mknod -m 644 random c 1 8
mknod -m 666 tty c 5 0
mknod -m 600 tty0 c 4 0
mknod -m 600 tty1 c 4 1
mknod -m 600 ttyS0 c 4 64
mknod -m 666 video0 c 81 0
mknod -m 666 zero c 1 5

mkdir pts

ln -s /proc/self/fd fd
ln -s fd/0 stdin
ln -s fd/1 stdout
ln -s fd/2 stderr

echo "done."
```

Oppstartsscript

Oppstartsscriptene håndterer oppstarten av Linux, og setter opp ramdisk, nettverk, drivere og andre innstillinger.

/linuxrc

```
#!/bin/sh

export PATH=/bin:/sbin:/usr/sbin:

sleep 2

echo Setting up ramdisk.

echo -n Mounting /proc...
mount /proc
echo done.

echo -n Mounting /dev/hda1 to /mnt...
mount /dev/hda1 /mnt
echo done.

echo -n Extracting rootfs to /dev/ram1...
gunzip -c /mnt/rootfs.img.gz > /dev/ram1
echo done.

echo -n Unmounting /dev/hda1...
umount /mnt
echo done.

echo Setting /dev/ram1 to /
echo 257 > /proc/sys/kernel/real-root-dev

echo -n Unmounting proc...
umount /proc
echo done.

echo Continuing normal boot from ramdisk.
```

/etc/init.d/rcS

```
#!/bin/sh

echo "Setting Norwegian keyboard"
/sbin/loadkmap < /etc/no.kmap

echo Mounting filesystem in read-write mode.
mount -n -o remount,rw /

echo Mounting /proc
mount /proc
echo Mounting /dev/pts
mount /dev/pts

echo "Setting hostname detektor"
/bin/hostname detektor

echo "Starting pcmcia"
/etc/rc.d/rc.pcmcia start
sleep 5

echo "Starting network"
/sbin/ifconfig lo 127.0.0.1 netmask 255.0.0.0 broadcast 127.255.255.255
/sbin/route add -net 127.0.0.0 netmask 255.0.0.0 lo

/sbin/ifconfig eth0 192.168.0.10 netmask 255.255.255.0
/sbin/route add default gw 192.168.0.1 eth0

/sbin/iwconfig eth1 mode ad-hoc
/sbin/iwconfig eth1 channel 3
/sbin/iwconfig eth1 essid ELAB nick Prism2
/sbin/iwconfig eth1 rate 11Mb

/sbin/ifconfig eth1 10.0.0.10 netmask 255.0.0.0 broadcast
10.255.255.255

echo "Starting sshd"
/usr/sbin/sshd

echo "Starting par8ch driver"
cd /usr/par8ch
./indriver SrPar8ch378
```

Konfigurasjonsfiler

Konfigurasjonsfilene ligger i /etc katalogen, og inneholder innstillinger for systemet. Jeg vil skrive litt om hva de ulike konfigurasjonsfilene inneholder i tillegg til å liste opp innholdet.

/etc/fstab

Denne filen inneholder konfigurasjonen om partisjonene på systemet, og hvordan de skal settes inn i rootfilsystemet.

```
/dev/ram1   /     ext2  defaults,errors=remount-ro 0 1
none        /proc proc  defaults 0 0
none        /dev/pts devpts gid=5,mode=620 0 0
```

/etc/passwd

Denne filen, sammen med /etc/group og /etc/shadow inneholder informasjon om brukerene på systemet. Jeg har bare en root-bruker, og en bruker som sshd kjører som.

```
root:x:0:0:root:/root:/bin/sh
sshd:*:501:255:sshd privsep:/var/empty:/bin/false
```

/etc/group

```
root:x:0:root
sshd:x:255:
```

/etc/shadow

```
root:$1$$xHe8p9GfUD/xY0.eudbAY/:12424:0:99999:7:::
sshd:*:11880:0:99999:7:-1:-1:0
```

/etc/hosts

Denne filen inneholder lokal dns-informasjon.

```
127.0.0.1               detektor localhost.localdomain localhost
```

/etc/inittab

Denne filen forteller init hvilke prosesser som skal kjøres ved oppstart. Init er den første prosessen som kjøres etter at kjernen er lastet inn. Det er inittab som forteller init at /etc/init.d/rcS skal kjøres som et oppstartsscript.

```
::sysinit:/etc/init.d/rcS
::ctrlaltdel:/sbin/reboot
tty1::respawn:/sbin/getty 38400 /dev/tty1
::restart:/sbin/init
::shutdown:/bin/umount -a -r
```

/etc/issue

Denne filen forteller hva som skal stå som velkomstmelding når login vises.

```
Linux Detektor box - by Kristian Hiim
Kernel \r on an \m
```

/etc/nsswitch.conf

Denne filen konfigurerer nss, som håndterer dns og innlogging.

```
passwd:     files
shadow:     files
group:      files

#hosts:     db files nisplus nis dns
hosts:      files dns

protocols:  files
services:   files
```

/etc/profile

Denne filen inneholder globale variabler, som lastes når brukere logger inn. Den eneste variabelen jeg setter er PATH, som forteller hvor en finner programmer som en ikke spesifiserer hvor er.

```
# Set path
PATH=/bin:/sbin:/usr/bin:/usr/sbin
export PATH
```

/etc/resolv.conf

Denne filen har informasjon om navnetjenere. A.b.c.d erstattes med ip-nummeret til navnetjeneren (DNS). En kan legge inn så mange navnetjenere som en måtte ønske.

```
nameserver a.b.c.d
```

/etc/protocols og /etc/services

Disse filene er kopiert direkte fra /etc på mitt lokale utviklingssystem. Filene er ikke nødvendige, men kan være greie å ha. De beskriver navn på protokoller og tjenester, og kan brukes av programmer, slik at en slipper å huske for eksempel port-nummer og lignende. Da jeg ikke har laget filene selv, lister jeg de ikke opp her, men viser til /etc på enhver standard Linux installasjon, hvor en kan finne filene.

Programkode

Jeg har laget to programmer i denne fagprøven. Hvert av programmene består av flere filer. Jeg har lagt ved de viktige filene her. Dersom du skal kompilere og installere programmene, vil jeg anbefale å laste de ned fra http://www2.geo.uib.no/embedded/

Detektorprogrammet

Detektorprogrammet består av detektor.c, som er selve programmet. I tillegg til detektor.c, har jeg laget en makefile som forteller kompilatoren hvordan programmet skal kompileres til binærfiler som kjøres, og et script som tar bildet, og sender det til displayenheten. Jeg har også kopiert følgende filer fra par8ch-fagprove.tgz til katalogen, for å kunne kompilere programmet:

- par8ch.c
- par8ch.h
- par8chkd.h
- srhelper.c
- srhelper.h

Disse filene er filene som fulgte med PAR8CH (A/D konverteren), og som programmet bruker for å kommunisere med enheten.

detektor.c

```
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include <time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>


#include "par8ch.h"
#include "srhelper.h"

#define BUFFER_SIZE (256*PAR8CH_ANALOG_CHANNELS)

int stop_program=0;     // Setting this will cause the program to stop
int client_connected=0; // When this flag is 1, the program will take snapshots
int recording=0;        // This is automatically set to 1 when the program
                        // is recording.


long low_value  = 177200;   // The lowest value before motion event
long high_value = 177900;   // The highest value before motion event

pthread_mutex_t trigger_mutex;
pthread_cond_t trigger_cond;

DEVHANDLE handle;  // The handle for the A/D converter
```

```
int client_socket; // The client network socket
int server_socket; // The server network socket

/*
 * Monitor routine
 *
 * This routine reads from the A/D converter,
 * and activates the trigger routine when motion
 * is detected.
 */
void *monitor_routine(void *arg) {
  int ErrorCode;
  double ActualSps;
  long buffer[BUFFER_SIZE];
  int count;
  int i;
  long hi, lo;

  handle = Par8chOpen(

                        PAR8CH_0x378,
                        PAR8CH_PORT_MODE_EPP,

                        100.0, // Requested Sps
                        &ActualSps,

                        &ErrorCode
                        );

  if ( handle == BAD_DEVHANDLE ) {
    printf("Failed to open SrPar8ch! Stopping program.\n");
    stop_program=1;
    pthread_exit(NULL);
  }

  //  printf("Samples per second: %.1f\n", ActualSps);

  Par8chStart(handle);

  while(!stop_program) {
    sleep(1);
    count = Par8chReadData(handle, buffer, BUFFER_SIZE, &ErrorCode);

    if (ErrorCode != PAR8CH_ERROR_NONE) {
      Par8chStop( handle );
      Par8chClose( handle );
      printf("Error reading data - stopping\n");
      Par8chStop(handle);
      Par8chClose(handle);
      stop_program=1;
      pthread_exit(NULL);
    }

    if (client_connected && (!recording)) {
      hi=-1;
      lo=-1;
      for (i=0; i<count; i+=PAR8CH_ANALOG_CHANNELS) {
         //      printf("%d ", buffer[i]);
         if ((buffer[i] < low_value) || (buffer[i] > high_value)) {
           pthread_mutex_lock(&trigger_mutex);
           pthread_cond_signal(&trigger_cond);
           pthread_mutex_unlock(&trigger_mutex);
           break;
         } else {
           if (hi<0)
             hi=buffer[i];
           if (lo<0)
             lo=buffer[i];
```

```
            if (buffer[i]>hi)
              hi = buffer[i];
            if (buffer[i]<lo)
              lo = buffer[i];
          }
      }
      //      printf("High: %d\n Low: %d\n", hi, lo);
    }

  }
  Par8chStop(handle);
  Par8chClose(handle);
  pthread_exit(NULL);
}

/*
 * Trigger routine
 *
 * This routine calls a script when motion is detected
 * The script takes a picture with the USB camera, and
 * sends it to the display unit.
 */
void *trigger_routine(void *arg) {
  char dtbuf[20];
  char cmd[256];
  int z;
  time_t td;
  pthread_mutex_lock(&trigger_mutex);
  while (!stop_program) {
    pthread_cond_wait(&trigger_cond, &trigger_mutex);
    if (!stop_program) {
      recording = 1;
      Par8chUserLed(handle, 1);
      printf("Trigger - grabbing image!!\n");
      time(&td);
      z = strftime(dtbuf, sizeof dtbuf, "%Y%m%d%H%M%S", localtime(&td));
      snprintf(cmd, 256, "./grab.sh %s.jpeg", dtbuf);
      system(cmd);
      write(client_socket, dtbuf, z);
      write(client_socket, "\n", 1);
      sleep(1);
      Par8chUserLed(handle, 0);
      recording = 0;
    }
  }
  pthread_mutex_unlock(&trigger_mutex);
  pthread_exit(NULL);
}

/*
 * Server routine
 * This routine handles connecting and disconnecting a client.
 * The program will not record images unless there is a client
 * connected.
 */
void *server_routine(void *arg) {
  int srvr_port = 9096;
  struct sockaddr_in adr_srvr;
  struct sockaddr_in adr_clnt;
  int len_inet;
  int z;

  char buffer[50];

  server_socket = socket(PF_INET,SOCK_STREAM,0);
  if (server_socket == -1) {
    printf("Error: Could not open network socket!\n");
    stop_program=1;
    pthread_exit(NULL);
  }
```

```
  memset(&adr_srvr,0,sizeof adr_srvr);
  adr_srvr.sin_family = AF_INET;
  adr_srvr.sin_port = htons(srvr_port);
  adr_srvr.sin_addr.s_addr = INADDR_ANY;

  len_inet = sizeof adr_srvr;
  z = bind(server_socket,(struct sockaddr *)&adr_srvr, len_inet);
  if ( z == -1 ) {
    printf("Error: Could not bind network socket!\n");
    stop_program=1;
    pthread_exit(NULL);
  }

  z = listen(server_socket, 1);
  if (z == -1) {
    printf("Error: Could not listen to network socket!\n");
    stop_program=1;
    pthread_exit(NULL);
  }

  while (!stop_program) {
    len_inet = sizeof adr_clnt;
    printf("Waiting for a connection...\n");
    client_socket = accept(server_socket, (struct sockaddr *)&adr_clnt, &len_inet);

    if (client_socket == -1) {
      printf("Error: Client could not connect!\n");
      continue;
    }
    printf("Client connected...\n");
    client_connected=1;
    while ((z = read(client_socket, buffer, 50))>0) {

      // Commands from the client could be handled here, if needed

    }
    printf("Client disconnected...\n");
    client_connected=0;
    close(client_socket);

  }
  // close(s);
  pthread_exit(NULL);
}



int main(void) {
  pthread_t monitor_thread;
  pthread_t trigger_thread;
  pthread_t server_thread;
  pthread_attr_t attr;

  sleep(1);

  printf("Detektor program\n");
  printf("Press ENTER to stop program\n");

  // Initializing keyboard
  if ( SrKeyboardNonBlock() == -1 ) {
      printf( "ERROR: Can't set non-blocking read ... (%s)\n", PAR8CH_ERROR_MSG[1] );
    exit(1);
  }

  // Initializing threads
  pthread_mutex_init(&trigger_mutex, NULL);
  pthread_cond_init(&trigger_cond, NULL);

  pthread_attr_init(&attr);
  pthread_attr_setdetachstate(&attr, PTHREAD_CREATE_JOINABLE);
```

```
  pthread_create(&monitor_thread, &attr, monitor_routine, NULL);
  pthread_create(&trigger_thread, &attr, trigger_routine, NULL);
  pthread_create(&server_thread, &attr, server_routine, NULL);


  sleep(1);
  while ( !stop_program ) {
    if ( SrGetKeyCheck() != EOF ) {
      printf("Key pressed - stopping\n");
      if (client_connected) {
          printf("Could not stop.. client still connected\n");
      } else {
        stop_program=1;
      }
    }
  }


  pthread_detach(monitor_thread);

  pthread_mutex_lock(&trigger_mutex);
  pthread_cond_signal(&trigger_cond);
  pthread_mutex_unlock(&trigger_mutex);
  pthread_detach(trigger_thread);

  close(server_socket);
  pthread_detach(server_thread);

  pthread_mutex_destroy(&trigger_mutex);
  pthread_cond_destroy(&trigger_cond);

  exit(0);
}
```

grab.sh

```
#!/bin/sh

if [ -n "$1" ]
then
  streamer -o $1
  ftpput --user=fagprove --pass=fagprove 10.0.0.20 display/images/ $1
  rm -f $1
#  scp test.jpeg $1
else
  echo "This program should not be called directly."
fi
```

Makefile

```
CC=i586-linux-gcc

SROS=-DSROS_LINUX

all: detektor

par8ch.o:
         $(CC) $(SROS) -c par8ch.c

srhelper.o:
         $(CC) $(SROS) -c srhelper.c

detektor.o: detektor.c
         $(CC) $(SROS) -c detektor.c

detektor: detektor.o par8ch.o srhelper.o
         $(CC) -lpthread -o detektor detektor.o par8ch.o srhelper.o

clean:
         rm -f *.o
         rm -f detektor
```

Displayprogrammet

Displayprogrammet et Qt program. Qt er et utviklingsverktøy som gjør det enkelt å sette opp programmer med grafisk brukergrensesnitt.

Displayprogrammet består av tre filer:

- display.cpp, som er selve programmet
- display.pro, som er en prosjektfil
- build.sh, som er et script som kompilerer displayprogrammet og klargjør det for bruk.

display.cpp

```
#include <qapplication.h>
#include <qsocket.h>
#include <qpushbutton.h>
#include <qlabel.h>
#include <qvbox.h>
#include <qstring.h>
#include <qtextview.h>

class MainW : public QVBox
{
  Q_OBJECT
public:
  MainW(QWidget *parent=0, const char *name=0 ) : QVBox(parent, name)
  {
    QPixmap p(320,240);
    p.fill(Qt::black);
    setMinimumSize(320, 320);
    setMaximumSize(320, 320);
    image = new QLabel(this);
    image->setPixmap(p);
    infoText = new QTextView( this );
    QPushButton *quit = new QPushButton( "Quit", this, "quit" );
    connect( quit, SIGNAL(clicked()), this, SLOT(close()));
    is_connected=false;
    socket = new QSocket( this );
    connect( socket, SIGNAL(connected()),
             SLOT(socketConnected()) );
    connect( socket, SIGNAL(connectionClosed()),
             SLOT(socketConnectionClosed()) );
    connect( socket, SIGNAL(readyRead()),
             SLOT(socketReadReady()) );
    connect( socket, SIGNAL(error(int)),
             SLOT(socketError(int)) );
    connectToHost();
  }

  ~MainW() { }

  void connectToHost() {
    if (is_connected)
      return;
    socket->connectToHost(QString("10.0.0.10"), 9096);
  }

protected slots:
  void closeConnection() {
    socket->close();
    if (socket->state() == QSocket::Closing) {
      connect(socket, SIGNAL(delayedCloseFinished()),
               SLOT(socketClosed()) );
    } else {
```

```
      socketClosed();
    }
    is_connected=false;
  }

  void socketReadReady() {
    QString r;
    QString i;
    QPixmap p;
    while (socket->canReadLine()) {
      i = socket->readLine();
      r = tr("images/%1.jpeg").arg(i.stripWhiteSpace());
      p.load(r);
      image->setPixmap(p);
      infoText->append(r);
    }
  }

  void socketConnected() {
    is_connected=true;
    infoText->append(tr("Connected to server\n"));
  }

  void socketConnectionClosed() {
    is_connected=false;
    infoText->append(tr("Connection closed by the server\n"));
  }

  void socketClosed() {
    is_connected=false;
    infoText->append(tr("Connection closed\n"));
  }

  void socketError( int e) {
    infoText->append(tr("Error number %1 occured\n").arg(e));
  }

protected:
  QSocket *socket;
  QTextView *infoText;
  QLabel *image;
  bool is_connected;
};


int main( int argc, char **argv )
{
    QApplication a( argc, argv );
    MainW m;

    a.setMainWidget( &m );
    m.show();
    return a.exec();
}

#include "display.moc"
```

display.pro

```
CONFIG += qt warn_on
SOURCES = display.cpp
TARGET = display
```

build.sh

```
#!/bin/sh

qmake -o Makefile display.pro
make
mkdir images
```