Jan Inge Sande
Institutt for geovitenskap, Elab

Fagprøve i serviceelektronikk

30. januar – 3. februar 2006

30 sider ink. forsiden

Innhold

Del 1

Fremdriftsplan

Tidsramme

Mandag, kl. 09:30 – 11:30

Hjelpemidler

- PC med SuSE og OpenOffice.org Writer 2.0

Oppgave 1, SBP-logger

Planlegging av arbeidet

Fremdriftsplan

Mandag, kl. 12:00 – 16:00

- Koble opp signalgenerator til lydkort på PC som skal brukes til skriving av program for datainnsamling. Setter opp denne til å generere ett passende signal som jeg kan bruke ved feilsøking under programutvikling.
- Eventuelt installere enda ett lydkort i denne maskinen dersom kort som er bestilt kommer i løpet av dagen, for så å flytte signalgenerator over på dette og fortsette utvikling av program med dette kortet til hjelp.
- Skrive program i Python for lesing av data fra lydkortet, mottak av UDP telegram og sending av trigger signal. Satser på å lese rå data fra Linux driverne til kortet. Mulig det er nødvendig å skrive dette i C pga. Python muligens ikke blir raskt nok om en ønsker real-time prosessering av data fra lydkortet. Men slik som oppgaven er beskrevet er dette ikke nødvendig nå, men kunne vært behov for ved en mulig senere oppgradering av funksjonene i programmet. Vil derfor kanskje skrive om dette programmet om det blir tid senere til å legge til slike funksjoner (automatisk trigger, direkte visning av data i GUI, osv.).

Tirsdag, kl. 08:00 – 16:00

- Vil skrive enda ett program i Python for styring av hovedprogrammet. Disse vil kommunisere seg imellom ved bruk av TCP. Til brukergrensesnitt vil jeg prøve å få laget ett vindusgrensesnitt, enten ved bruk av TkInter eller PyQT. Disse to verktøyene har jeg ikke jobbet så mye med den siste tiden, så er usikker på hvor lang tid jeg vil bruke på å få ett av disse til å gjøre jobben. Mulig jeg derfor må gjøre endringer i planen her. Om det viser seg at dette tar mye lenger tid enn ventet vil jeg isteden lage programmet basert på kommandoer som en skriver inn.
- Teste kommunikasjon mellom programmene og feilsøke andre feil som dukker opp under veis.

Onsdag, kl. 08:00 – 14:00

- Se til at alt fungerer som det skal ved å foreta tester av alle funksjoner.
- Beskrive hvordan plotting av måledata kan gjøres, eller eventuelt også bygge ut systemet med muligheter for plotting.

Nødvendig materiell

- PC med Linux, OSS lydsystem, Python og IDE (f.eks. Eric3). Windows og/eller Mac OS kan også brukes til utvikling av programmet som presenterer grafisk brukergrensesnitt.
- Signalgenerator
- Oscilloskop for å sjekke triggesignal, signalnivå etc.
- Diverse kabler og plugger; BNC og 3,5 mm 3-pols minijack for tilkobling til lydkort

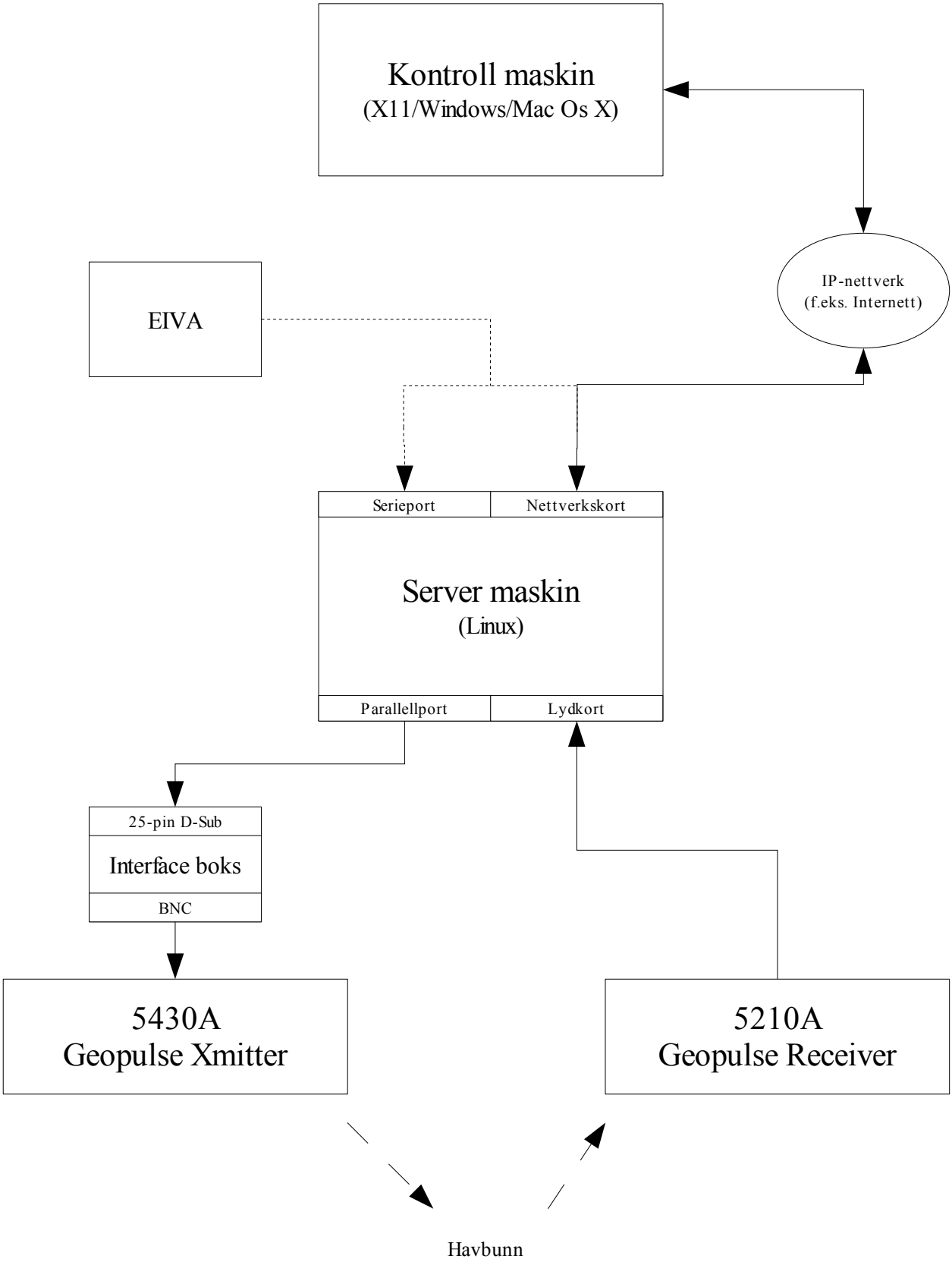
- Antistatisk armlenke og matte ved installasjon av lydkort i PC

Nødvendig dokumentasjon

- Oversikt over programbiblioteker
  - «Python Library Reference» http://docs.python.org/
  - «Qt Reference Documentation» http://doc.trolltech.com/
  - «OSS Programmer's guide» http://www.opensound.com/pguide/
  - «Linux man-pages»
- Bruksanvisning for signalgenerator, for generering av «egenproduserte» signalformer

Arbeidsrapport

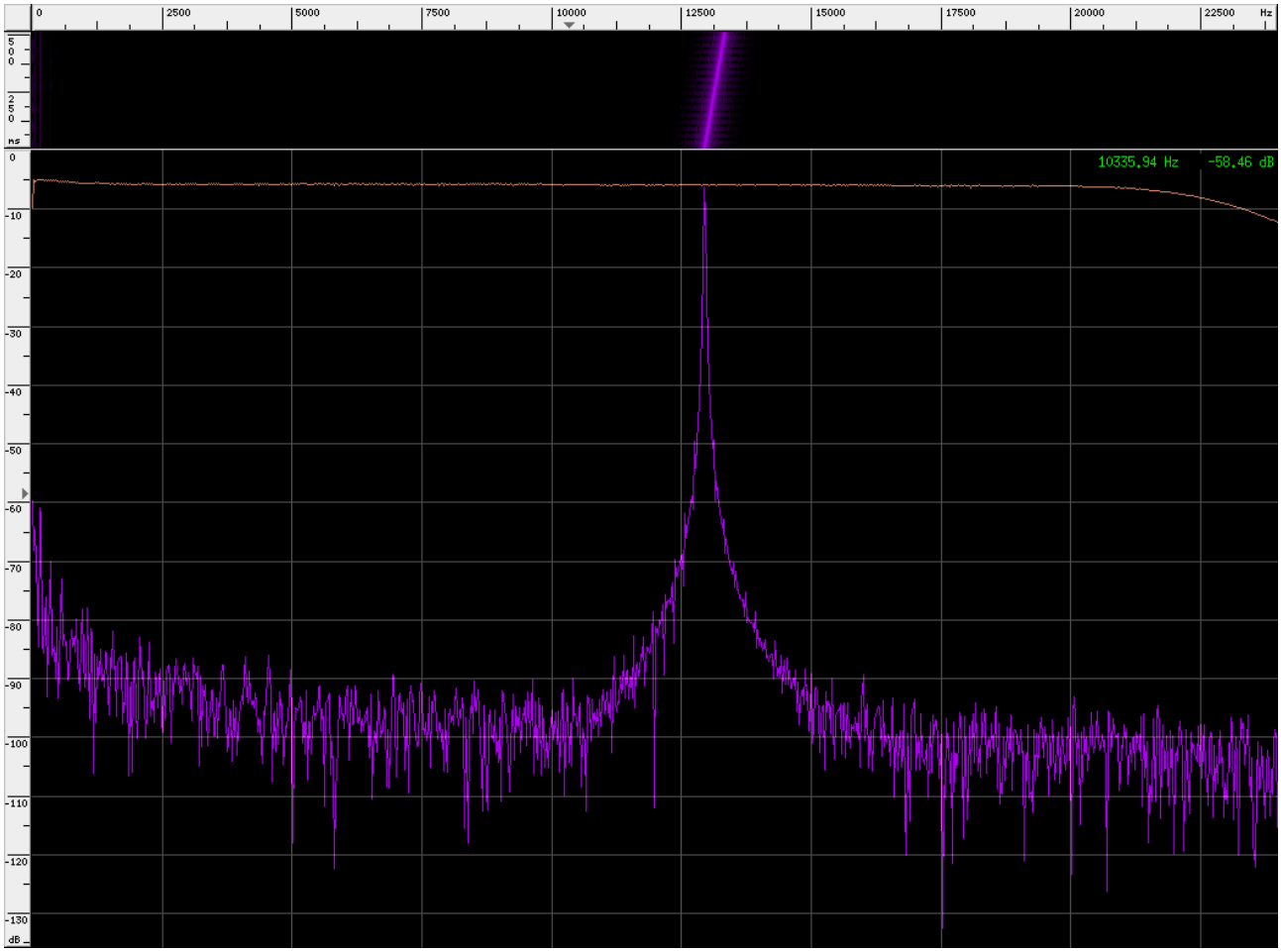
Blokkskjema



Test av lydkort

Det første jeg begynte med var å se om jeg klarte å få signaler inn på PC-en via lydkortet, fra en digital signalgenerator. Dette gikk greit og var fort gjort. Videre måtte jeg finne ut om lydkortet hadde de kvaliteter som skulle til for å kunne brukes til å digitalisere data fra en slik «sub-bottom-profiler». Signalene som kommer opp igjen fra havbunnen har visstnok ganske lav frekvens, lavere enn det som er nødvendig for å få grei nok lyd fra en PC. De fleste gamle lydkort hadde ganske dårlig frekvensrespons, spesielt på line og mikrofon inngangene. Gjerne ikke mer enn det som var nødvendig for en telefonsamtale. Så da var det å finne ut hva kortet jeg hadde til rådighet duger til, ett kort av typen som er bygget inn på hovedkortet i PC-en.

Til å finne ut dette brukte jeg programmet Baudline for Linux. Dette er ett gratis (men ikke åpen kildekode) program for sanntids signal analysering,. Jeg satt opp signalgeneratoren til å generere sweep med sinus signal og lik amplitude over frekvensbåndet 0 – 24 000 Hz, og satt Baudline til 48 kHz samplerate og tegning av en linje ved maks nivå. Her er ett skjermbilde av resultatet:



Som en ser her ble det en ganske flat kurve opp til ca. 20 kHz. Det er litt vanskelig å se begynnelsen på spektrumet på bildet her, men ved <= 12 Hz ble det en brå dempning av signalet.

Ett bra dynamikkområde er også viktig for slike data som lydkortet skal få bryne seg på her. Derfor ble det bestilt inn ett nytt semi-profesjonelt lydkort med oppgitt signal/støy-forhold på 100 dB for line-in inngangen. Dessverre var det bare tilgjengelig en modul for håndtering av det gamle OSS

(Open Sound System) lydsystemet for Linux, i programmeringsspråket Python som jeg har brukt. Dette støtter ikke samplinger større enn 16 bit, noe som begrenser dynamikkområdet til 96 dB.

Skriving av program

Som beskrevet i oppgaveteksten skulle jeg skrive ett program som triggrer en «5430A Geopulse Xmitter», enten via parallellporten eller signal ut fra line-out fra lydkortet. Og jeg skulle helst bruke programmeringsspråket Python. Problemet da med å velge parallellporten var (igjen) mangel på standard modul for å styre denne fra Python. Det fantes dog en tredjeparts modul med navn pyParallel, men på websidene til de som produserte denne sto det at denne var «under utvikling». Det var derimot mulig å laste ned det de hadde, og dette skulle fungere bra. Men jeg hadde testet denne før fagprøven, på flere forskjellige maskiner, uten å få noe til å fungere.

Det andre alternativet var å bruke utgangen på lydkortet, problemet med denne løsningen var at en ikke fikk mer enn ett signal på minimum 1 V ut. Og denne «Xmitter»-en skulle vel ha ett signal på 5 V. Og signalgeneratoren jeg brukte skulle ha minimum 1,5 V triggesignal. Kunne ha løst dette med noe enkel ekstern elektronikk, f.eks. ekstern strømforsyning (batteri, spenning fra USB-, serie-, eller parallellport etc.) og transistor, eller spenningsdobler med dioder, kondensator og AC fra lydkortet. En annen ting er at en ikke kan spytte ut DC fra ett lydkort, men de fleste kort burde klare å få ut ett signal på ca. 100 Hz, og dermed burde også en puls med lengde på 5 ms gå greit. En tredje ting som kan være problematisk er at de fleste (billige) lydkort ikke er «full duplex», de kan ikke ta opp samtidig som de spiller av. Altså får en ikke startet samplingen akkurat i det en sender triggesignalet. Men det får en ikke uansett så lenge en bruker Python, og ikke ett real-time operativsystem.

Jeg valgte å bruke parallellporten, og skrev da ett program i C for å sende triggesignalet. Dette programmet, pp.c, blir så kjørt fra Python og sender bokstaven «T» til stdin på programmet når det vil sende signalet.

Videre valgte jeg å dele opp programmet i to deler, en klient med grafisk brukergrensesnitt som kobler til en server som tar seg av selve jobben med innsamling av data. Disse kommuniserer så med hverandre via IP og TCP. IP står for «Internet Protocol», og er, som navnet sier, den protokollen som får Internett til å fungere. En kan dermed i teorien sitte hvor som helst hvor en har en Internett-tilkobling og styre datainnsamlingen. Men det var nå egentlig ikke dette jeg tenkte på når jeg fant på dette, men snarere at det kan være praktisk å sitte en annen plass å kontrollere innsamlingen enn der signalet kommer «om bord». En kortest mulig kabel med analogt signal fra mottakeren er ønskelig for å minske mengden støy som blander seg med signalet.

Etter lunsj på mandagen begynte jeg å skrive server programmet. Til dette brukte jeg Eric3, som er ett IDE (Integrated Development Environment) som også er skrevet i Python (og bruker PyQT). Python er ett ganske kjekt språk som jeg begynte først å bruke for ett par måneder siden, men det er utrolig kompakt og det går kjapt å skrive programmer i det. Med kompakt mener jeg at det tar liten plass oppe i hodet, og det er lett å huske hvordan de fleste funksjoner og slikt fungerer uten å måtte slå de opp. I tillegg trenger en ikke skrive så mye unødvendig dilldall for å få ting til å fungere. Ett Python program blir gjerne 10 ganger kortere enn ett tilsvarende program i C++ eller Java. Og så bruker en også ett par ganger kortere tid på å skrive det. Det ligner litt på Java siden en må bruke en «interpretter» for å kjøre det, og en kan ikke kompilere det til kode som kjører direkte på prosessoren. I likhet med Java blir programmet på forhånd kompilert til en slags «bytecode», men dette skjer automatisk og er ikke noe en trenger å tenke på som med Java. Men det er ikke helt påkrevd at en må ha en slik «runtime environment» som i Java. Om en bruker programmet py2exe kan en lage en pakke av programmet som kjører uten at en trenger å ha Python installert på maskinen fra før. Dette gjør den ved å sjekke hvilke moduler en har brukt i Python programmet, og

pakker kun disse ned. Dermed vil ikke pakken ta så stor plass (greit om en skal distribuere den på web etc.). Python er også støttet på mange plattformer; f.eks. Linux, Windows og Mac OS. Nokia har også laget en utgave til mobiltelefonene sine. Og programmene en lager kan kjøre på alle plattformene uten endringer i koden.

Å skrive serveren på mandagen gikk ganske greit, ingen problemer dukket opp. Men jeg brukte mer tid enn jeg hadde planlagt på forhånd, og ble ikke ferdig i løpet av dagen. Den delen som skulle snakke med GUI klienten ble klar, så jeg begynte på klient delen som planlagt på tirsdagen.

Til verktøy for å lage grafisk brukergrensesnitt sto valget mellom TkInter og PyQt. Ved å velge TkInter, som det sto i oppgaveteksten at jeg burde velge, måtte jeg ha skrevet kode for å tegne opp grafikken. Med PyQt er det mulig å bruke Qt Designer, hvor en setter opp grafiske elementer ved å bruke «pek og klikk»-metode og en får se hvordan ting vil bli seende ut mens en holder på. Med TkInter ville en ha måttet visualisere hvordan ting ville blitt seende ut oppe i hodet, noe som nok er lettere for mindre programmer. Men blir det en viss størrelse på programmet, vil jeg tro PyQt er lettere å arbeide med, og resultatet blir gjerne litt penere.

PyQt er en binding til biblioteket Qt fra det norske firmaet Trolltech. Egentlig er det ett bibliotek for C++, men det finnes bindinger til mange andre språk, slik at det kan brukes fra disse også. Trolltech har to typer lisensvilkår for dette verktøyet. Om en lager ett program med åpen kildekode, kan en bruke biblioteket gratis og må forholde seg til vilkårene i GPL (GNU General Public License, samme som gjelder for f.eks. Linux). Skriver en lukket programvare må en betale litt lisenskroner, og får andre vilkår. Akkurat som Python, fungerer Qt på mange plattformer; systemer som X11 fungerer på (blant annet Linux), Windows og Mac OS.

Når en bruker Qt med Python istedenfor C++, bruker en programmet «pyuic» istedenfor «uic». Dette genererer så Python kode, akkurat som uic genererer C++ kode. Disse filene importeres så i Python programmet hvor en vil bruke de.

Verken PyQt eller TkInter har jeg jobbet noe særlig med tidligere, så var litt spent på om jeg ville få ting til å fungere og hvor lang tid jeg ville bruke. Hadde satt av hele tirsdagen til å lage klient programmet, og ting gikk veldig greit så jeg holdt tidsskjemaet for denne delen.

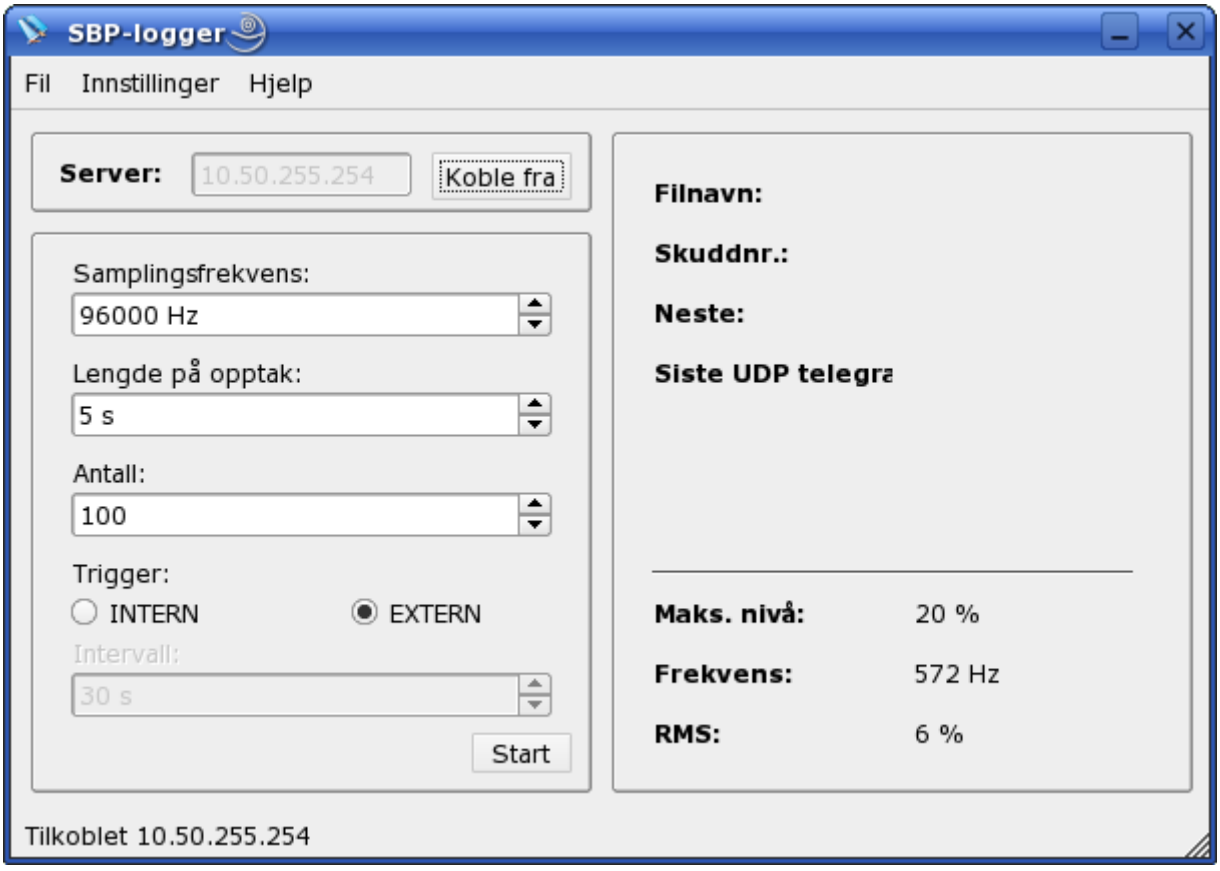
Onsdagen gikk med til å gjøre ferdig det jeg ikke fikk gjort i tide på serveren, og litt «debugging» av programmene. Dermed fikk jeg ikke begynt på oppgave 2 før på torsdagen. Ble ikke så mye tid til å teste programmene, så kan dessverre ikke garantere at det ikke dukker opp grusomme bivirkninger ved bruk.

Programmene ligger vedlagt dette dokumentet, «klient.py» er klient programmet. Og «server.py» er server programmet, som må kjøre på Linux eller andre systemer hvor en har OSS (f.eks. FreeBSD). I de fleste Linux distribusjoner er nå OSS blitt byttet ut med det mer moderne ALSA (Advanced Linux Sound Architecture), men dette systemet støtter emulering av OSS.

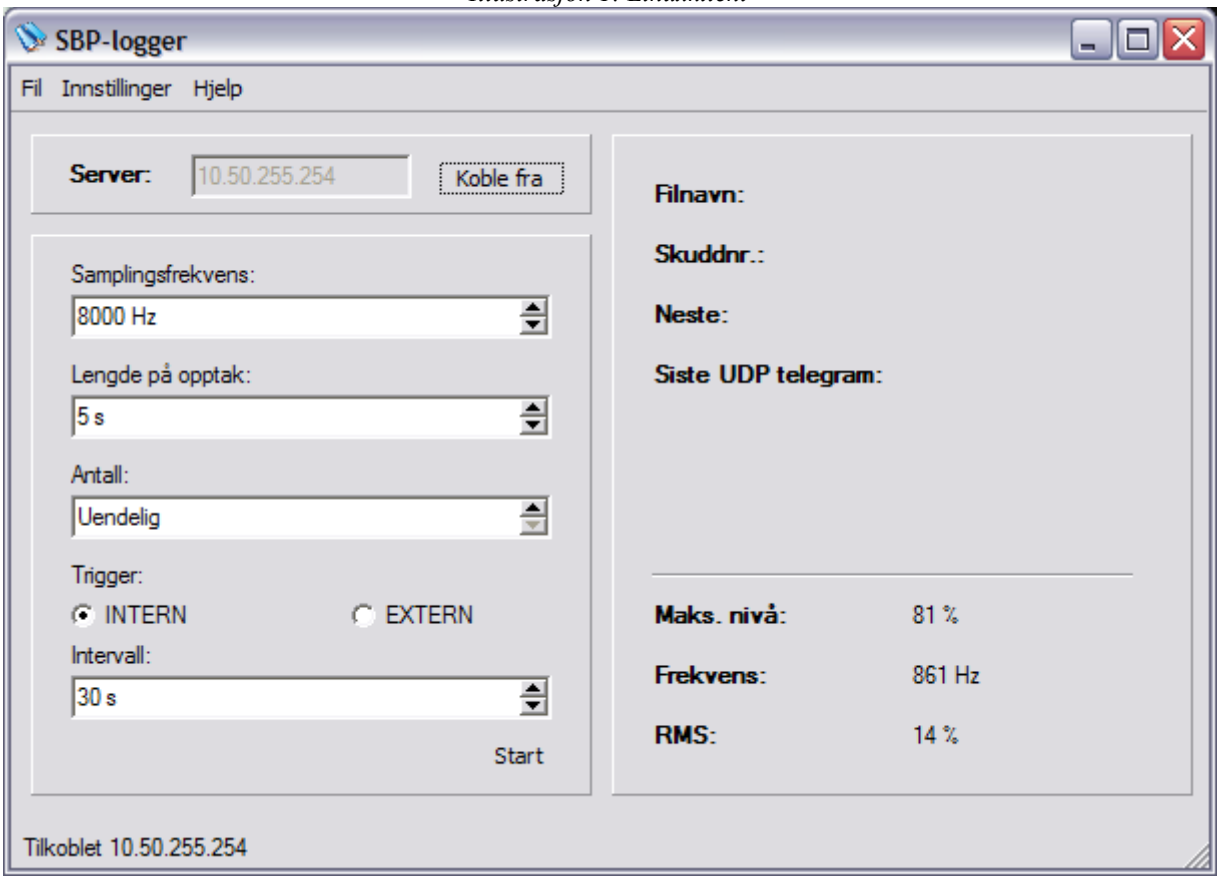
Har ikke lagt ved filer generert av pyuic e.l., grunnet størrelsen på disse. Kun filer skrevet for hånd.

Om en har flere lydkort i maskinen en skal bruke serveren på, må en endre på «SND_DEVICE» definisjonen i «server.py» til å inneholde riktig navn i forhold til kortet en vil bruke.

Resultat



Illustrasjon 1: Linuxklient



Illustrasjon 2: Windowsklient

Oppgave 2, GPS-logger

Planlegging av arbeidet

Fremdriftsplan

Onsdag 14:00 – 16:00

- Setter opp lokalt WDS nettverk med WPA2 (AES-kryptering) mellom to stk. Linksys WRT54GS.
- Legger inn uoriginal firmware fra Sveasoft e.l. på det ene aksesspunktet, og kobler det andre til PC/lokalt nettverk.
- Ser litt på oppsett av «link budget» og valg av antenner etc.
- Begynner på program for mottak av data på serieporten (fra GPS), og sending via nettverk.

Torsdag 08:00 – 16:00

- Legger programmet inn på «modifisert» Linksys aksesspunkt, og gjør feilsøking
- Skriver ferdig programmet for videresending av data, eventuelt legger til mulighet for buffring av data i FIFO buffer i RAM

Nødvendig materiell

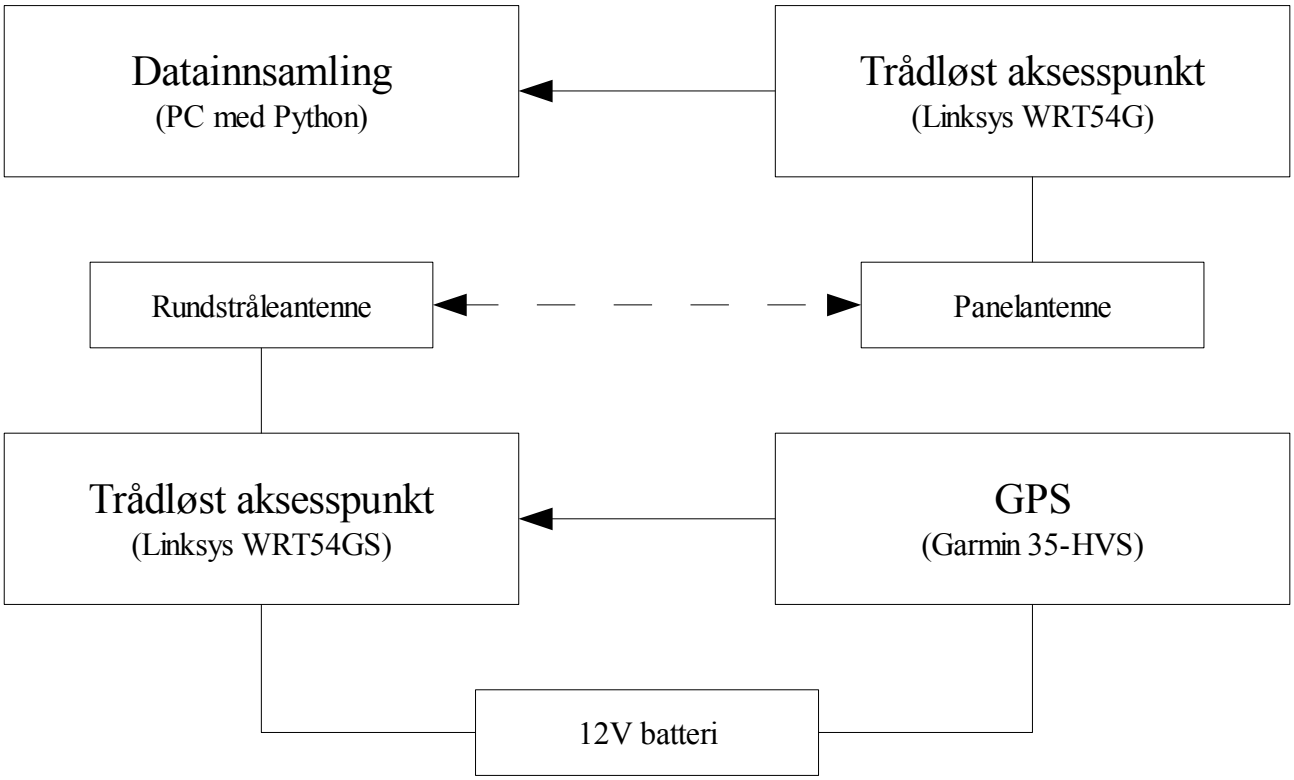
- PC med Linux, IDE og krysskompilator for ARM prosessor.
- 2 stk. WRT54GS og det som hører med her; nettverkskabler, strømforsyning og rubberduck antenner
- Windows programmet Radio Mobile for grafisk fremstilling av radiolink (ikke nødvendig, men ett greit supplement)
- Eventuelt annet tilgjengelig utstyr som bedre antenner, low-loss koaksialkabel, plugger for tilkobling til aksesspunkt (RP-TNC). Men dette er egentlig ikke nødvendig for denne prototypen, om en ikke skal teste systemet skikkelig ved utplassering.

Nødvendig dokumentasjon

- Gjeldende lover og regler om radionettverk, fra PT og/eller Lovdata.
- «Linux man pages»
- Dokumentasjon på firmwaren fra Sveasoft

Arbeidsrapport

Blokkskjema



Oppsett av bro mellom trådløse aksesspunkt

Ettersom aksesspunktene jeg hadde tilgjengelig av modell WRT54GS trolig hadde vært i bruk før, var det første jeg gjorde å ta en tilbakestilling til fabrikkinnstilling på hver av boksene. Dette gjøres ved å sette på strøm til boksene, vente til de er startet opp (når «Power» lysdioden på frontpanelet slutter å blinke), for så å holde «Reset» knappen på baksiden av boksen inne i minimum 5 sekunder.

Når aksesspunktet så starter på nytt igjen er innstillingene nullstilt. Og standardoppsett på de fleste trådløse bokser fra Linksys er følgende:

- Boksens IP adresse: 192.168.1.1
- Passord for konfigurasjon: admin
- Trådløst nettverksnavn (SSID): linksys
- DHCP-server aktivert

Nå er radio delen av aksesspunktet slått på, og ingen autentisering blir foretatt om noen naboer etc. prøver å koble seg til. Derfor kan det være en fordel om en bare kobler boksen direkte i nettverkskortet på PC-en en bruker, slik at ingen får urettmessig tilgang til lokalt nettverk/Internett. Alle Ethernet portene på baksiden av boksen støtter automatisk MDI-X, slik at en ikke trenger å tenke på å bruke krysset nettverkskabel for dette.

Men før en gjør det kan det være lurt å laste ned ny uoriginal firmware fra Sveasofts websider. De

har flere utgaver av denne avhengig av hvilken versjon av WRT54G(S) en har. Disse modellene fra Linksys har vært på markedet i noen år, men har blitt oppgradert med kraftigere prosessor, ny Ethernet switchmodul og trådløst chipset i nyere utgaver. Så for at en ikke skal f.eks. få feil driver i forhold til det som er i boksen er det viktig å laste ned rett utgave. Versjonsnummeret på boksen er å finne på undersiden, over serienummeret.

Ettersom DHCP-serveren i boksen er aktivert, trenger en bare få DHCP-klienten på PC-en til å spørre om ny adresse for å få dette riktig satt opp. Dette gjøres enkelt i SuSE Linux ved å skrive følgende kommando som «root»-bruker:

```
/etc/init.d/network restart
```

Dette forutsetter at en bruker DHCP til konfigurering av IP-adresser fra før. Om en bruker Windows kan en velge «Kjør» på Startmenyen og skrive inn følgende:

```
ipconfig /renew *
```

Nå skal det være mulig logge inn på webserveren til aksesspunktet ved å skrive inn adressen http://192.168.1.1/ i en nettleser. En boks hvor en skal skrive inn brukernavn og passord spretter opp, her skriver en kun inn «admin» som passord – ingen brukernavn.

For å legge inn ny firmware på boksen velger en her fanen «Administration» og så «Firmware Upgrade». Filen en har lastet ned fra Sveasoft er en ZIP-fil, denne må pakkes ut før en kan laste opp innholdet. Dette kan f.eks. gjøres med kommandoen unzip i Linux, eller programmet WinRar til Windows. Om en har Windows XP, har en også innbygget ZIP støtte i Explorer.

Nå skal en ha en fil med navn som slutter på .bin, denne lastes opp gjennom webgrensesnittet. Oppgraderingen kan ta noen minutter, mens denne prosessen er i gang er det viktig at en ikke kobler fra strømmen til boksen. Om en gjør dette vil bare deler av firmwaren bli skrevet til FLASH-minnet, og en skal da være heldig om den starter opp igjen som den skal. Det er derimot ikke noe problem å rette dette opp igjen uten å måtte åpne boksen, ettersom det i oppstartslasteren ligger en liten TFTP-server. Og denne delen av minnet blir ikke overskrevet med oppgraderingsmetoden beskrevet her. Om en tar av lokket på boksen finner en også ett JTAG grensesnitt, som gjør det mulig å skrive data direkte til minnet. Men skal ikke gå nærmere inn på hvordan en foretar seg slik gjenoppliving nå.

Når «Power» lysdioden slutter å blinke er oppgraderingen fullført, og boksen restartet. Nå kan en prøve å gå til websidene på aksesspunktet igjen. Ettersom boksen som skal plasseres i halebøyen må være mulig å nå fra det nettverket hvor maskinen som samler inn dataene står, må den få en IP-adresse på samme subnet som denne. Om det er DHCP-server tilgjengelig på dette nettverket kunne en satt opp DHCP-klienter på aksesspunktene, men det er ikke mulig å gjøre dette via webgrensesnittet. Det kan også være greit med faste adresser slik at en unngår problemer om en får nye forskjellige adresser. Om en har DHCP-server på nettverket må en velge ett adresseområde hvor denne ikke deler ut adresser, slik at ikke serveren deler ut de samme adressene til andre maskiner (eller en kan sette opp faste adresser i serveren).

Har en f.eks. adresseblokken 192.168.0.0/26, og DHCP serveren deler ut adresser mellom 192.168.0.10 og 192.168.0.62, kan en velge adressene 192.168.0.9 og 192.168.0.8 dersom disse ikke er i bruk. For ett slikt oppsett kan en f.eks. gjøre følgende endringer under «Basic Setup»:

- Internet Connection Type: Disable
- Router Name: Mosby
- Router IP: 192.168.0.8

- Subnet Mask: 255.255.255.192
- Time Zone: GMT +1

Trykk deretter «Save Settings».

Og her er eksempel andre innstillinger som må endres for å få opp en trådløs bro, trykk «Save Settings» etter endringer på hver enkelt fane:

- *Wireless*
    - *Basic Settings*
        - Wireless Network Name: Mosby
        - Wireless Channel: 1 – 2,412 Ghz
    - *Security*
        - Security Mode: WPA Pre-Shared Key
        - WPA Algorithms: AES
        - WPA Shared Key: l3o2ambO3Flkg4arjKW
    - *WDS*
        - LAN 00:12:17:16:00:75 GPS
- *Security*
    - *Firewall*
        - Firewall Protection: Disable
- *Applications & Gaming*
    - *DHCP*
        - DHCP Server: Disable
- *Administration*
    - *Management*
        - Router Password: fiskesuppe
        - HTTPS: Disable
        - Spanning Tree Protocol: Enable
        - 802.1x: Enable
        - NTP Client: Disable
        - RW Partition: Enable
        - SSHD: Enable
        - UpnP: Disable

På aksesspunktet som skal plasseres i bøyen blir de eneste endringene, i forhold til over, følgende punkter:

- Setup

- Basic Setup
    - Router Name: GPS
    - Router IP: 192.168.0.9
- Wireless
    - WDS
        - LAN 00:12:17:E7:83:9C Mosby

Under WDS angir en MAC adresse på det trådløse interfacet på aksesspunket en vil koble til. Om en bare skal bruke WDS kan en også aktivere MAC filtrering uten å angi adresser. Dette vil ikke ha noe å si for WDS oppsettet, men det vil være umulig å koble til på vanlig måte.

Andre tips er å finne en «ledig» kanal der hvor en skal sette opp utstyret, slik at en unngår mest mulig interferens. Kanal nr. 6 er standard på omtrent alle trådløse aksesspunkt, og dermed mest i bruk. Modulasjonsmetoden som brukes i 802.11b og 802.11g er «Direct Sequence Spread Spectrum» DSSS, og som navnet sier så sprer den signalet over flere frekvenser. Velger en kanal 6 vil en sende ut omtrent like høy effekt på «kanal5» og «kanal 7» også. En bør ikke velge kanal 14, ettersom denne bare er lovlig å bruke i Japan.

En kan også skru ned «Transmission rate» om en ikke har bruk for best mulig hastighet hele tiden, f.eks. ved overføring av GPS data. En vil da få en mer stabil link (mindre pakketap) enn med automatisk valg.

Installasjon av loggeprogram

Om en følger oppskriften over vil nå aksesspunktet være satt opp omtrent på samme måte som en vanlig Linux PC, og det vil være mulig å logge inn og overføre filer via SSH. Men før en kan overføre loggeprogrammet må det kompileres for prosessortypen som brukes i disse boksene fra Linksys, nemlig en MIPS prosessor. Denne arkitekturen brukes mye i embedded enheter, f.eks. i alt fra Ciscos routere til leketøyene Playstation fra Sony. Inkludert med kildekoden til firmwaren for aksesspunktene, har Linksys lagt ved en kompilator som genererer kode for andre arkitekturer enn det en bruker på utviklingsmaskinen, en såkalt krysskompilator. Så last ned en nyere utgave av koden til en WRT54GS fra GPL kode sidene til Linksys. For tiden er denne å finne på følgende adresse:

http://www1.linksys.com/support/gpl.asp

I pakken du laster ned skal det være en mappe med navn /tools/brcm, og i denne finner en kildekoden til krysskompilatoren. Så flytt f.eks. denne over i /opt, og kvitt deg med restene (kildekoden til firmwaren). Om du vil lage en egen utgave av firmwaren får du tak i kildekode med flere funksjoner fra Sveasoft.

Når du har flyttet filene må kompilatoren kompileres. Det følger med ett makefile script for å kompilere alle delene denne består av, så dette er ikke noe problem å få til. Men det kan ta en stund, avhengig av hvor rask maskin du har tilgjengelig.

Her er ett eksempel:

1. wget ftp://ftp.linksys.com/opensourcecode/wrt54gs/4.70.6/WRT54GS_v4.70.6.tgz
2. tar zxvf WRT54GS_v4.70.6.tgz
3. cd WRT54GS/tools

4. mv -r brcm /opt/
5. cd ../..
6. rm -rf WRT54GS
7. cd /opt/brcm
8. make

Når dette er gjort har du gcc, cpp, ld osv. i /opt/brcm/hndtools-mipsel-uclibc/bin/. En kan eventuelt legge denne til i PATH variabelen om en ønsker det (rediger /etc/profiles.local om du bruker SuSE 10.0 e.l.). En /opt/brcm/hndtools-mipsel-linux/ mappe er også laget, men biblioteket µClibc er det som brukes i Linksys boksene, og de fleste andre embedded enheter basert på Linux.

Så da var det bare igjen å kompilere loggeprogrammet og legge det inn på aksesspunktet. Dette kan gjøres som følger:

9. /opt/brcm/hndtools-mipsel-uclibc/bin/mipsel-uclibc-gcc -o serialtransfer serialtransfer.c
10. scp serialtransfer root@192.168.0.9:/usr/local/bin/

Når en får spørsmål om brukernavn og passord her, skriver en inn «root» som brukernavn og «fiskesuppe» som passord.

Under /usr/local er det montert ett FLASH filsystem ved bruk av nyere Sveasoft firmware utgaver, om det er aktivert i webgrensesnittet. Resten av mappestrukturen er filsystem i RAM, om en laster opp nye versjoner av programmer ofte kan det være lurt å laste de opp i RAM filsystemet i stedet, slik at en ikke sliter ut FLASH-minnet, men da forsvinner selvfølgelig filene om en restarter aksesspunktet eller strømmen blir slått av.

For å få programmet kjørt automatisk ved oppstart av aksesspunktet lager en ett rc script i /usr/local/etc/rc.d/, dette kan f.eks. hete «S01serialtransfer.sh» og innhold kan være følgende to linjer:

```
#!/bin/sh
/usr/local/bin/serialtransfer &
```

Av installerte teksteditor på boksen har en kun Vi. Denne fungerer på en litt annen måte enn f.eks. Notepad i Windows, her er en liten innføring:

11. vi /usr/local/etc/rc.d/S01serialtransfer.sh
12. Trykk på bokstaven «i» på tastaturet, for å sette inn tekst. Lim deretter inn linjene over, og trykk escape når dette er gjort.
13. Skriv inn «:wq» for å lagre filen, og avslutte programmet.

Nå skal alt være på plass, og programmet startes hver gang Linux starter opp. Vær oppmerksom på at alle nettverksinterface på aksesspunktet muligens ikke alltid er satt opp i det programmet starter.

Datainnsamling

Grunnet at det dukket opp en feil i UDP broadcast programmet jeg begynte å skrive på, som jeg ikke fant ut av, valgte jeg heller å ikke bruke mer tid på det og heller skrive om programmet serialtransfer.c til å fungere som en enkel TCP server (den støtter bare en klient). Ulempen med dette er at en da må sette opp en forbindelse direkte til aksesspunktet fra datainnsamlingsmaskinen, og derfor må angi en IP adresse manuelt. Og mister en forbindelsen må denne settes opp på nytt. Dette kan automatiseres, men det er vanskelig å detektere når en har mistet forbindelsen om en bare

mottar data, uten å sende noe. En fordel med TCP fremfor UDP er at en ikke mister pakker, men det er vanligvis ikke så nøye om en mister en linje eller to av og til med GPS data. I trådløse nettverk av typen 802.11b, g og n har en dessuten innbygget system for retransmisjon, og dette tar seg av det meste av «pakketap» (eller rammer) som oppstår på den trådløse linken.

Kom ikke så langt at jeg fikk skrevet ett program for å koble opp til aksesspunktet med GPS koblet til, automatisk. Men en kan lett logge data til fil ved f.eks. å skrive inn noe slikt i ett konsoll i Linux på datainnsamlingsmaskinen:

```
telnet 192.168.0.9 3000 > gps.log &
```

En kunne ha skrevet noen linjer i Python for å få gjort det samme, og få laget til automatisk oppkobling i det strøm ble koblet til aksesspunktet. Dog, vil tro en bedre løsning er å lage ett program som sender ut UDP pakker på broadcast adresse, f.eks. ved å ta utgangspunkt i serialtransfer.c. Programmet som skal motta dataene blir også en del enklere da.

Link budsjett

For å finne ut ca. hvor lang rekkevidde en vil få på ett trådløst nettverk kan en sette opp ett «link budsjett». Dette inneholder de forskjellige delene radiosignalet passerer gjennom, og dempning/forsterkning i hver av disse delene. Så legger en disse sammen, og finner så ut hvor stor margin en har før nettverket i teorien virker eller slutter å virke. Her er ett forslag:

WRT54GS, Mosby – sendereffekt: 9 mW	+	9,5	dBm
ARTP-1404, RP-TNC	-	0,2	dB
WBC400, 5 meter	-	1,1	dB
ANM-1406, N-Male	-	0,2	dB
HG2412P, panelantenne m/65° strålingsvinkel	+	12,0	dBi
Tap i luft, 3 km	-	109,7	dB
HG2412U, rundstråle	+	12,0	dBi
ANM-1406, N-Male	-	0,2	dB
WBC400, 1 meter	-	0,2	dB
ARTP-1404, RP-TNC	-	0,2	dB
Signaleffekt til radio i WRT54GS, GPS	-	78,6	dBm

Signaltap over en avstand finner en med følgende ligning:

Tap = 20 log(4πr/λ) = 20 log(4 * 3,1415 * 3000 / 0,124) = 109,67 dB

Altså er «r» avstand i meter, og «λ» bølgelengden til signalet – 12,4 cm ved 2,4 GHz.

Det var ikke så lett å finne noe oppgitt følsomhet fra Linksys på radioene i disse WRT54GS boksene. På ett gammelt dataark for en WRT54G (ca. samme boks som WRT54GS, men forskjellig firmware) sto det oppgitt -84 dBm for 11 Mbps. Da får en i så fall en margin på 5 dB. Men 11 Mbps er ikke nødvendig for overføring av litt GPS data (ca. 1,1 Kbps ved GPGGA og GPRMC). Fant ikke noe oppgitt for lavere hastigheter heller, men for en hastighet på 1 Mbps er det vanlig med en følsomhet på ca. 90 dBm, så da får en litt bedre margin – 11 dB.

I motsatt retning må en gå ned en mW eller to i sendereffekt for å ikke gå over maks tillatt utstrålt effekt, som er 100 mW e.i.r.p. (Effective Isotropic Radiated Power) i Norge. Uten om det blir regnestykket likt motsatt vei også, men marginene blir 0,5 – 1 dB lavere.

En kan få bedre rekkevidde ved å velge bedre antenner. Men grunnen til at jeg satt opp en panelantenne på båten, og ikke noe mer «retningsbestemt» var at jeg tenkte at bøyen ikke nødvendigvis vil holde seg på en rett linje bak båten (om den gjorde det hadde det jo ikke vært noen vits med GPS der). Og av omtrent samme grunn en rundstråleantenne i bøyen.

En annen ting å tenke på, spesielt når en skal bruke systemet ute på havet, er såkalt Fresnel-sone. Langt fra alt av utstrålt effekt fra antennen går i en rett linje til mottakerantennen, og signalet som tar andre veier kan reflekteres fra gjenstander for så å treffe mottakerantennen. Om disse gjenstandene (f.eks. havet), er innenfor en hvis avstand fra denne rette linjen, vil signalet som reflekteres komme frem med en faseforskjell og kan motvirke det opprinnelige signalet. En kan tenke seg at denne sonen er formet som en ellipse om står og ser 90° på linjen mellom sender og mottaker.

Vedlegg

klient.py

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

SRV_PORT = 3030
SRV_BUFFER = 1500
SND_SRATE = 8000
SND_RECTIME = 5
SND_COUNT = -1
SND_INTERVAL = 30

from qt import *
from mainform import *
from aboutform import *
from imagec import *
import sys
import webbrowser
import thread
import socket
import signal

if sys.platform == "win32":
        import winsound

state = {
'samplerate' : SND_SRATE,
'rectime' : SND_RECTIME,
'count' : SND_COUNT,
'interval' : SND_INTERVAL,
'run' : True,
'filename' : ''
}

global sd

def connect():
        qt.maindialog.statusBar().message("Kobler til...")
        # sd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

        try:
                sd.connect((str(qt.maindialog.serverEdit.text()), SRV_PORT))
        except socket.error:
                qt.maindialog.serverButton.setText("Koble til")
                qt.maindialog.filKoble_tilAction.setEnabled(True)
                qt.maindialog.filKoble_fraAction.setEnabled(False)
                qt.maindialog.serverEdit.setEnabled(True)
                qt.maindialog.statusBar().message("Kunne ikke koble til %s: %s" %
(str(qt.maindialog.serverEdit.text()), sys.exc_info()[1][1]))
                sys.exit()

        qt.maindialog.settingsGroup.setEnabled(True)
        qt.maindialog.statusBar().message("Tilkoblet %s" % str(qt.maindialog.serverEdit.text()))

        while state['run']:
                for line in sd.recv(SRV_BUFFER).splitlines():
                        words = line.split(" ")

                        # if words[0] != "STATS" and len(words) > 1:
                        #       print "%s - %s" % (words[0], words[1])

                        if words[0] == "SAMPLERATE":
                                state['samplerate'] = int(words[1])
                                qt.maindialog.samplerateBox.setValue(state['samplerate'])

                        elif words[0] == "RECTIME":
                                state['rectime'] = int(words[1])
                                qt.maindialog.lengthBox.setValue(state['rectime'])

                        elif words[0] == "COUNT":
```

```
                        state['count'] = int(words[1])
                        qt.maindialog.countBox.setValue(state['count'])

                elif words[0] == "TRIGGER":
                        if words[1] == "EXTERN":
                                qt.maindialog.intervalBox.setEnabled(False)
                                qt.maindialog.intervalLabel.setEnabled(False)
                                qt.maindialog.externButton.setChecked(True)
                        else:
                                qt.maindialog.intervalBox.setEnabled(True)
                                qt.maindialog.intervalLabel.setEnabled(True)
                                qt.maindialog.internButton.setChecked(True)

                elif words[0] == "INTERVAL":
                        state['interval'] = int(words[1])
                        qt.maindialog.intervalBox.setValue(state['interval'])

                elif words[0] == "FILENAME":
                        state['filename'] = words[1]
                        qt.maindialog.filenamevLabel.setText(state['filename'])

                elif words[0] == "RECNUM":
                        qt.maindialog.shotsvLabel.setText(words[1])

                elif words[0] == "TIMELEFT":
                        if words[1] == "S":
                                qt.maindialog.nextvLabel.setText("Sampler...")
                                if
qt.maindialog.innstillingerSpil_avl_lyd_ved_aktivering_av_triggerAction.isOn():
                                        winsound.Beep(1200, 500)
                        elif words[1] == "-":
                                qt.maindialog.nextvLabel.setText("")
                        else:
                                qt.maindialog.nextvLabel.setText("%s s" % words[1])

                elif words[0] == "RUNNING":
                        qt.maindialog.samplerateBox.setEnabled(False)
                        qt.maindialog.lengthBox.setEnabled(False)
                        qt.maindialog.countBox.setEnabled(False)
                        qt.maindialog.internButton.setEnabled(False)
                        qt.maindialog.externButton.setEnabled(False)
                        qt.maindialog.intervalBox.setEnabled(False)
                        qt.maindialog.runButton.setText("Stopp")
                        qt.maindialog.runButton.setEnabled(True)

                elif words[0] == "STOPPED":
                        qt.maindialog.samplerateBox.setEnabled(True)
                        qt.maindialog.lengthBox.setEnabled(True)
                        qt.maindialog.countBox.setEnabled(True)
                        qt.maindialog.internButton.setEnabled(True)
                        qt.maindialog.externButton.setEnabled(True)
                        qt.maindialog.runButton.setText("Start")
                        qt.maindialog.runButton.setEnabled(True)
                        if qt.maindialog.internButton.isOn():
                                qt.maindialog.intervalBox.setEnabled(True)

                if words[0] == "STATS":
                        if words[1] == "-":
                                qt.maindialog.rmsvLabel.setText("")
                                qt.maindialog.levelvLabel.setText("")
                                qt.maindialog.ratevLabel.setText("")
                        else:
                                qt.maindialog.rmsvLabel.setText("%s %%" % words[3])
                                qt.maindialog.levelvLabel.setText("%s %%" % words[2])
                                qt.maindialog.ratevLabel.setText("%s Hz" % words[1])

        state['run'] = False
        qt.maindialog.settingsGroup.setEnabled(False)
        sd.close()
        qt.maindialog.statusBar().message("Frakoblet %s" % str(qt.maindialog.serverEdit.text()))
        global sd
        sd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        qt.maindialog.serverButton.setText("Koble til")
        qt.maindialog.filKoble_tilAction.setEnabled(True)
        qt.maindialog.filKoble_fraAction.setEnabled(False)
```

```
        qt.maindialog.serverEdit.setEnabled(True)

class qtgui(QApplication):
        def __init__(self, args):
                QApplication.__init__(self,args)

                self.connect(self, SIGNAL("lastWindowClosed()"), self, SLOT("quit()"))
                self.maindialog = mainform(None)

                self.setMainWidget(self.maindialog)

                self.maindialog.show()

class mainform(mainForm):
        def __init__(self,parent):
                mainForm.__init__(self,parent)
                self.aboutdialog = self.aboutform(None)
                self._connectSlots()
                self.statusBar().message("Ikke tilkoblet")
                if sys.platform == "win32":
                        self.innstillingerSpil_avl_lyd_ved_aktivering_av_triggerAction.setEnabled(Tr
ue)
                self.timer = QTimer(self)
                self.timer.start(500, False)


        def _connectSlots(self):
                self.connect(self.omOmAction, SIGNAL("activated()"), self._slotOmActivated)
                self.connect(self.hjelpInternettFagprveAction, SIGNAL("activated()"),
self._slotInternettFagprveActivated)
                self.connect(self.hjelpInternettElabAction, SIGNAL("activated()"),
self._slotInternettElabActivated)
                self.connect(self.filKoble_tilAction, SIGNAL("activated()"),
self._slotServerButtonClicked)
                self.connect(self.filKoble_fraAction, SIGNAL("activated()"),
self._slotServerButtonClicked)
                self.connect(self.serverButton, SIGNAL("clicked()"), self._slotServerButtonClicked)
                self.connect(self.samplerateBox, SIGNAL("valueChanged(int)"),
self._slotSamplerateBoxChanged)
                self.connect(self.lengthBox, SIGNAL("valueChanged(int)"),
self._slotLengthBoxChanged)
                self.connect(self.countBox, SIGNAL("valueChanged(int)"), self._slotCountBoxChanged)
                self.connect(self.intervalBox, SIGNAL("valueChanged(int)"),
self._slotIntervalBoxChanged)
                self.connect(self.externButton, SIGNAL("clicked()"), self._slotExternButtonClicked)
                self.connect(self.internButton, SIGNAL("clicked()"), self._slotInternButtonClicked)
                self.connect(self.runButton, SIGNAL("clicked()"), self._slotRunButtonClicked)

        def _slotOmActivated(self):
                self.aboutdialog.show()

        def _slotInternettFagprveActivated(self):
                webbrowser.open("http://www2.geo.uib.no/service-elektronikk/fagprover/JanInge/")

        def _slotInternettElabActivated(self):
                webbrowser.open("http://www2.geo.uib.no/")

        def _slotServerButtonClicked(self):
                if self.serverButton.text() == "Koble til":
                        self.serverButton.setText("Koble fra")
                        self.serverEdit.setEnabled(False)
                        self.filKoble_tilAction.setEnabled(False)
                        self.filKoble_fraAction.setEnabled(True)
                        state['run'] = True
                        thread.start_new_thread(connect, ())
                else:
                        state['run'] = False
                        self.serverButton.setText("Koble til")
                        self.serverEdit.setEnabled(True)
                        self.filKoble_tilAction.setEnabled(True)
                        self.filKoble_fraAction.setEnabled(False)

        def _slotRunButtonClicked(self):
                if self.runButton.text() == "Start":
                        self.samplerateBox.setEnabled(False)
```

```
                        self.lengthBox.setEnabled(False)
                        self.countBox.setEnabled(False)
                        self.internButton.setEnabled(False)
                        self.externButton.setEnabled(False)
                        self.intervalBox.setEnabled(False)
                        self.runButton.setEnabled(False)
                        self.runButton.setText("Stopp")
                        sd.send("RUN")

                else:
                        self.runButton.setEnabled(False)
                        sd.send("STOP")

        def _slotSamplerateBoxChanged(self):
                nrate = int(str(self.samplerateBox.cleanText()))
                if nrate != state['samplerate']:
                        sd.send("SAMPLERATE %i\n" % nrate)
                        state['samplerate'] = nrate

        def _slotLengthBoxChanged(self):
                ntime = int(str(self.lengthBox.cleanText()))
                if ntime != state['rectime']:
                        sd.send("RECTIME %i\n" % ntime)
                        state['rectime'] = ntime

        def _slotCountBoxChanged(self):
                try:
                        ncount = int(str(self.countBox.cleanText()))
                except:
                        ncount = -1
                if ncount != state['count']:
                        sd.send("COUNT %i\n" % ncount)
                        state['count'] = ncount

        def _slotIntervalBoxChanged(self):
                ninterval = int(str(self.intervalBox.cleanText()))
                if ninterval != state['interval']:
                        sd.send("INTERVAL %i\n" % ninterval)
                        state['interval'] = ninterval

        def _slotExternButtonClicked(self):
                self.intervalBox.setEnabled(False)
                self.intervalLabel.setEnabled(False)
                sd.send("TRIGGER EXTERN")

        def _slotInternButtonClicked(self):
                self.intervalBox.setEnabled(True)
                self.intervalLabel.setEnabled(True)
                sd.send("TRIGGER INTERN")

        class aboutform(aboutForm):
                def __init__(self,modal=1):
                        aboutForm.__init__(self,modal)

if __name__ == "__main__":
        sd = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        qt = qtgui(sys.argv)
        qt.exec_loop()
        state['run'] = not state['run']
```

server.py

```
#!/usr/bin/env python
#-*- coding: utf-8 -*-

import ossaudiodev
import audioop
import socket
import select
import thread
import time
import wave
import os

SRV_PORT = 3030
SRV_TRIGPORT = 3031
SRV_BUFFER = 1500
SND_DEVICE = "/dev/dsp"
SND_FMT = ossaudiodev.AFMT_S16_LE
SND_NCHAN = 1
SND_SRATE = 8000
SND_RECTIME = 5
SND_COUNT = -1
SND_INTERVAL = 30
FILE_DIR = "recordings"
FILE_DATE = "%j_%H%M%S_"
TRIGGER = "./pp"

state = {
'device' : SND_DEVICE,
'format' : SND_FMT,
'channels' : SND_NCHAN,
'samplerate' : SND_SRATE,
'rectime' : SND_RECTIME,
'count' : SND_COUNT,
'udptrig' : False,
'interval' : SND_INTERVAL,
'ready' : True,
'run' : True,
'filename' : '',
'fileno' : 0,
'shotno' : 0
}

def uclistate(cli):
        # Send current state to client
        cli.send("HELLO %s\n" % addr[0])
        cli.send("DEVICE %s\n" % state['device'])
        cli.send("SAMPLERATE %s\n" % state['samplerate'])
        cli.send("CHANNELS %s\n" % state['channels'])
        cli.send("RECTIME %i\n" % state['rectime'])
        cli.send("COUNT %i\n" % state['count'])
        cli.send("INTERVAL %i\n" % state['interval'])

        if state['udptrig']:
                cli.send("TRIGGER EXTERN\n")
        else:
                cli.send("TRIGGER INTERN\n")

        if state['ready']:
                cli.send("STOPPED\n")
        else:
                cli.send("RUNNING\n")

def broadcast(msg):
        for sfd in clients.keys():
                if sfd == srv.fileno():
                        continue

                try:
                        clients[sfd].send(msg)
                except socket.error:
                        poll.unregister(sfd)
                        clients[sfd].close()
```

```
                        del clients[sfd]

def cmdexec(cmd, cli):
        words = cmd.split(" ")

        if words[0] == "SAMPLERATE":
                state['samplerate'] = dsp.speed(int(words[1]))
                broadcast("SAMPLERATE %s\n" % state['samplerate'])
                # print "Changed samplerate to %s by request from %s " % (state['samplerate'],
cli.getpeername()[0])

        elif words[0] == "RECTIME":
                state['rectime'] = int(words[1])
                broadcast("RECTIME %s\n" % state['rectime'])

        elif words[0] == "COUNT":
                state['count'] = int(words[1])
                broadcast("COUNT %s\n" % state['count'])

        elif words[0] == "INTERVAL":
                state['interval'] = int(words[1])
                broadcast("INTERVAL %s\n" % state['interval'])

        elif words[0] == "TRIGGER":
                if words[1] == "EXTERN":
                        state['udptrig'] = True
                        broadcast("TRIGGER EXTERN\n")
                else:
                        state['udptrig'] = False
                        broadcast("TRIGGER INTERN\n")

        elif words[0] == "RUN":
                state['ready'] = False
                broadcast("RUNNING")
                thread.start_new_thread(recthread, ())

        elif words[0] == "STOP":
                state['ready'] = True

def statsthread():
        while state['ready']:
                samplebuffer = dsp.read(state['samplerate'])

                for sfd in clients.keys():
                        if sfd == srv.fileno():
                                continue

                        try:
                                clients[sfd].send("STATS %i %i %i\n" % (audioop.cross(samplebuffer,
2), \
                                ((audioop.maxpp(samplebuffer, 2) * 100) / 32768), \
                                ((audioop.rms(samplebuffer, 2) * 100) / 32768)))
                        except socket.error:
                                poll.unregister(sfd)
                                clients[sfd].close()
                                del clients[sfd]

def recthread():
        state['shotno'] += 1
        state['filename'] = "%s%i.wav" % (time.strftime(FILE_DATE), state['shotno'])
        recnum = 0
        recfp = wave.open("%s/%s" % (FILE_DIR, state['filename']), "wb")

        broadcast("FILENAME %s\n" % state['filename'])

        recfp.setnchannels(state['channels'])
        recfp.setsampwidth(2)
        recfp.setframerate(state['samplerate'])

        # Open trigger pipe
        trig = os.popen(TRIGGER, "w")

        while True:
                if len(dsp.read(2)):
                        break
```

```
        broadcast("STATS - - -\n")

        if state['udptrig']:
                while not state['ready']:
                        ts = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
                        ts.bind(('', SRV_TRIGPORT))

                        tp = select.poll()
                        tp.register(ts)

                        while not state['ready']:
                                event = tp.poll(1000)

                                if event[0] == ts.fileno():
                                        eiva = ts.recvfrom(SRV_BUFFER)

                                        recnum += 1
                                        broadcast("RECNUM %i\nTIMELEFT S\n" % recnum)

                                        trig.write("T\n")
                                        trig.flush()

                                        samplebuffer = dsp.read(state['samplerate'] *
state['rectime'] * 2)
                                        recfp.writeframes(samplebuffer)

                                        broadcast("TIMELEFT -\n")

                                        if recnum == state['count']:
                                                state['ready'] = True

                        ts.close()
        else:
                while not state['ready']:
                        timeleft = state['interval']

                        while timeleft and not state['ready']:
                                timeleft -= 1
                                time.sleep(1)
                                broadcast("TIMELEFT %i\n" % timeleft)

                        if state['ready']:
                                break

                        recnum += 1
                        broadcast("RECNUM %i\nTIMELEFT S\n" % recnum)

                        trig.write("T\n")
                        trig.flush()

                        samplebuffer = dsp.read(state['samplerate'] * state['rectime'] * 2)
                        recfp.writeframes(samplebuffer)

                        if recnum == state['count']:
                                state['ready'] = True


        broadcast("TIMELEFT -\nSTOPPED\n")
        thread.start_new_thread(statsthread, ())

        recfp.close()

        trig.write("C\n")
        trig.flush()
        trig.close()

# Open soundcard and set initial format, samplerate and channels
print "Initializing audio device: %s" % state['device']

dsp = ossaudiodev.open(state['device'], 'r')
dsp.setparameters(state['format'], state['channels'], state['samplerate'])
state['samplerate'] = dsp.speed(state['samplerate'])

print "Initial setup: %i channel(s), %i Hz samplerate" % (state['channels'], state['samplerate'])
```

```
# Create listening socket
srv = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
srv.bind(("", SRV_PORT))
srv.listen(1)

print "Listening on port %i..." % SRV_PORT

# Set up select object and register listening socket
poll = select.poll()
poll.register(srv, select.POLLIN | select.POLLPRI)

# Dictionary with currently connected clients
clients = {}

# Monitor soundcard input and send statistics to connected clients
thread.start_new_thread(statsthread, ())

# Wait for connections or data
while state['run']:
        waiting = poll.poll(1000)

        for item in waiting:
                # Check if a new client is in the queue
                if item[0] == srv.fileno():
                        cli, addr = srv.accept()
                        clients[cli.fileno()] = cli
                        cli.setblocking(0)

                        poll.register(cli, select.POLLIN | select.POLLERR | select.POLLHUP |
select.POLLNVAL)

                        print "New connection from %s:%i" % (addr[0], addr[1])

                        # Enlighten client
                        uclistate(cli)

                elif item[1] == select.POLLIN:
                        cli = clients[item[0]]
                        data = cli.recv(SRV_BUFFER)

                        if len(data) == 0:
                                print "Closing connection to %s:%i" % (cli.getpeername()[0],
cli.getpeername()[1])
                                poll.unregister(cli)
                                del clients[item[0]]
                                cli.close()

                        else:
                                # print "Got data from %s: %s" % (cli.getpeername()[0],
data.strip())

                                if data.strip() == "SHUTDOWN":
                                        broadcast("SHUTDOWN")
                                        cli = clients[item[0]]
                                        del clients[item[0]]
                                        poll.unregister(cli)
                                        cli.close()
                                        state['run'] = False
                                        break;

                                cmdexec(data.strip(), clients[item[0]])

                else:
                        # Error on client socket, close connection
                        cli = clients[item[0]]
                        del clients[item[0]]
                        poll.unregister(cli)

                        cli.close()

print "Shutting down"

srv.close()
```

pp.c

```
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <sys/ioctl.h>
#include <linux/parport.h>
#include <linux/ppdev.h>

#include <stdio.h>
#include <unistd.h>

#define PORT "/dev/parport0"

int pp_open(char *port);
int pp_close(int pp);
int pp_setdir(int pp, int direction);
int pp_setdata(int pp, char byte);


int main(int argc, char *argv[])
{
        int pp, chr;

        pp = pp_open(PORT);

        for (;;)
        {
                chr = getc(stdin);
                if (chr == 'T')
                {
                        pp_setdata(pp, 0xFF);
                        usleep(1000);
                        pp_setdata(pp, 0x00);
                }
                else if (chr == 'C')
                {
                        return 0;
                }
        }

        pp_close(pp);
}

int pp_open(char *port)
{
        int pp; /* Parallel port file descriptor */
        int mode = IEEE1284_MODE_COMPAT; /* Set transfer mode; COMPAT, BYTE, NIBBLE, EPP or ECP */

        /* Open port */
        pp = open(port, O_RDWR);
        if (pp == -1)
                return pp;

        /* Claims access to the port, this fails if any other program have used PPEXCL. */
        if (ioctl(pp, PPCLAIM))
        {
                close(pp);
                return -1;
        }

        /* Set transfer mode, and do negotiation */
        if (ioctl(pp, PPNEGOT, &mode))
        {
                close(pp);
                return -1;
        }

        /* Return file descriptor, should be used when calling these functions later */
        return pp;
}

int pp_close(int pp)
{
```

```
        /* Release port and close file descriptor */
        ioctl(pp, PPRELEASE);
        return close(pp);
}

int pp_setdir(int pp, int direction)
{
        /* Set data direction forward/reverse */
        return ioctl(pp, PPDATADIR, direction);
}

int pp_setdata(int pp, char byte)
{
        /* Set data bits to byte */
        return ioctl(pp, PPWDATA, &byte);
}
```

serialtransfer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/stat.h>
#include <netinet/in.h>
#include <fcntl.h>
#include <strings.h>
#include <termios.h>

#define PORT 3000
#define TERM "/dev/tts/1"
#define BAUDRATE B9600
#define BUFFER 1408

int main(int argc, char *argv[]);
int dev_open(char *device, int baudrate);
int dev_close(int dev);

int main(int argc, char *argv[])
{
        int dev;
        struct sockaddr_in sin;
        struct sockaddr_in pin;
        int serverd;
        int clientd;
        int size;
        int ret;
        char buff[BUFFER];

        serverd = socket(AF_INET, SOCK_STREAM, 0);

        bzero(&sin, sizeof(sin));
        sin.sin_family = AF_INET;
        sin.sin_addr.s_addr = INADDR_ANY;
        sin.sin_port = htons(PORT);

        bind(serverd, (struct sockaddr *)&sin, sizeof(sin));

        listen(serverd, 1);

        for (;;)
        {
                clientd = accept(serverd, (struct sockaddr *)&pin, &size);

                dev = dev_open(TERM, BAUDRATE);

                for (;;)
                {
                        ret = read(dev, buff, BUFFER);

                        if (ret <= 0)
                                break;

                        ret = send(clientd, buff, ret, 0);

                        if (ret == -1)
                                break;
                }

                dev_close(dev);

                close(clientd);

        }

        return 0;
}

int dev_open(char *device, int baudrate)
{
```

```
        int dev; /* Serial port file descriptor */
        struct termios termopt;

        /* Open serial port */
        dev = open(device, O_RDONLY|O_NOCTTY);

        if (dev == -1)
                return dev;

        /* Clear terminal options structure */
        bzero(&termopt, sizeof(termopt));

        /* Set baudrate, 8n1, ignore modem control lines and activate receiver */
        termopt.c_cflag = baudrate | CS8 | CLOCAL | CREAD;

        /* Ignore parity errors */
        termopt.c_iflag = IGNPAR;

        termopt.c_cc[VMIN] = 1;
        termopt.c_cc[VTIME] = 0;

        /* Flush data received but not read and save settings */
        tcflush(dev, TCIFLUSH);
        tcsetattr(dev, TCSANOW, &termopt);

        return dev;
}

int dev_close(int dev)
{
        int ret;

        ret = close(dev);

        return ret;
}
```