Fagprøve i serviceelektronikk for

Trond Solberg

Fordypning: Data- og kontorsystemer
Prøveperiode: 3.-7. februar 2003-02-03
Universitetet i Bergen, Institutt for geovitenskap

Innhold

Planleggingsdel

Tidsramme for planleggingsdelen: Mandag, kl. 10.00 – 15.00

Utstyr jeg trenger:

- Jeg bruker min private PC med installert Windows XP SP1 operativsystem, MS Office 2000 og Acrobat Destiller til å skrive planleggingsdelen.

Oppgave 1

1.1 Installasjon av Linux RedHat 8.0 og "Realtime Linux"

Tidsramme: Mandag kl. 15.00 – 16.00, tirsdag, kl. 08.00 – 09.00

Fremdriftsplan:

- Installere ny harddisk i PC-en min
- Installere RedHat 8.0 fra CD-er
- Laste ned Realtime Linux 3.1 fra http://www.fsmlabs.com
- Laste ned Linux kernel-2.4.18 fra http://www.kernel.org
- Laste ned patch for Linux og Realtime Linux fra http://www.fsmlabs.com
- Følge vanlig fremgangsmåte for installasjon av Realtime Linux (det vil si, pakk ut Linux kjerne (heretter kalt kernel), oppdatere kernelen med Realtime Linux patch, pakke ut Realtime Linux, oppdatere (utdatert) versjon 3.1 med patch for kernel 2.4.18, kompilere kernelen, restarte maskinen med den nye kernelen, og til slutt kompilere Realtime Linux. Deretter kjøre et testprogram for å se at det hele var vellykket. Dette vil bli mer utførlig beskrevet senere.)
- Under installasjonen, vil jeg benytte tiden til å skrive litt om "sanntidsoperativsystemer".

Utstyr jeg trenger:

- PC – helst en rimelig oppdatert en, i og med at den skal brukes til utviklingsmaskin. Jo kraftigere maskinen er, jo raskere går kompileringen av programmene og en sløser vekk mindre tid på venting. (For denne oppgaven kommer jeg til å bruke PC-en jeg har brukt gjennom læretiden, basert på et MSI KT2-266 hovedkort med en AMD Athlon 1400MHz CPU og 512MB RAM.)
- Harddisk med stor nok kapasitet til å kunne romme en full installasjon av RedHat 8.0 + plass igjen til å arbeide på (10GB skulle holde, men den disken jeg har tilgjengelig og kommer til å bruke er på 40GB). Eventuelt en ekstra harddisk dersom den første skulle være defekt.
- Skrujern, type PH 1x75 (passer til de fleste stjerneskruer du finner i en PC)
- Antistatisk armbånd/lenke + matte. Eventuelt kan jeg jorde meg via lenken til chassiset på PC-en, hvis det ikke er mulighet/plass til å bruke matte.
- RedHat 8.0 installasjons CD-er.
- Internett-forbindelse

1.2 Program som kjører under "Realtime Linux"

Tidsramme: Tirsdag, kl. 08.00 – 15.00

Fremdriftsplan:

- Jeg kan tenkte meg å dele programmet opp i to deler – en Realtime modul som håndterer PPS signalet fra GPS-en, samt sender ut pulser på parallellporten, og et Linux program (monitor) som kommuniserer med denne modulen, leser NMEA setninger fra GPS-en, viser status- og klokkeinformasjon på skjermen, og synkroniserer den interne sekundtelleren (og dermed utpulsen på parallellporten) med GPS klokken.
- Disse blir utviklet parallelt, så tidsrammen er den samme for begge.
- GPS-en må på forhånd konfigureres vha Windows programvare. Den må settes opp til å sende data med 4800bps, 8 databit, ingen paritet, 1 stopbit. Den skal sende ut *kun* GPRMC-setningen, og PPS skal være aktivert.
- Når programmet et ferdig utviklet og testet, bør jeg ta noen "screenshots" under drift som kan brukes i dokumentasjonen.

Utstyr jeg trenger:

- PC med installert Realtime Linux operativsystem
- Teksteditor (Emacs)
- C kompilator (gnu gcc)
- Realtime Linux dokumentasjon
- PC med installert Windows 9x/NT/XP operativsystem
- GPS Garmin 35-HVS
- Strømforsyning til GPS-en. HVS-en er ikke så nøye på driftspenningen, alt fra 6 – 40 VDC er OK. Strømforbruk er ca 80mA.
- 2 stk 9pins DSUB hunn (DB9F) (én i reserve) for tilkobling av seriedata fra GPS-en til PC-ens serieport
- 2 stk 25pins DSUB han (DB25M) (én i reserve) for tilkobling av PPS-signal fra GPS-en til PC-ens parallellport.
- Da det er litt upraktisk å måtte lodde kontakter ute i felten, bruker jeg loddefrie kontakter, der ledningen presses ned mellom to kniver og låses vha en spesiell tang.
- Tang for montering av kabel på kontaktene
- Avbiter
- Multimeter for å teste koblingen dersom det ikke virker
- Koblingsskjema for kontaktene (vedlagt hovedoppgaven).
- Krympestrømper
- Varmluftspistol
- Kabel, 8-leder
- Oscilloskop – for å sjekke at det kommer seriedata, at PPS-pulsen blir sendt, og at pulsene som kommer ut fra parallellporten har korrekt form.
- GPS konfigurerings-programvare for Windows
- Kniv

Oppgave 2

2.1 Tidssynkronisering av datamaskiner i nettverk: NTP

Tidsramme: Tirsdag kl. 15.00 – 16.00, onsdag kl. 0800 – 11.00

Fremdriftsplan:

- Laste ned nødvendig software fra internett; NTP-4.1.1 fra http://www.eecis.udel.edu/~ntp/download.html, PPSkit-2.0.2 og PPSkit-2.0.2-fix2 fra http://www.kernel.org, og Linux kernel-2.4.19 fra samme server.
- Pakke ut NTP, Linux kernel og PPSkit m/tilhørende fix, patche PPSkit med PPSkit-fix2, patche kernelen med (tidligere patchet) PPSkit, deretter patche kernelen med PPSkit-fix2, konfigurere kernelen, kompilere den og restarte maskinen med denne kernelen.
- Kompilere og installere NTP
- Konfigurere NTP for å ta imot NMEA setninger fra GPS, samt bruke PPS tilkoblet serieport.
- Laste ned og installere NTP på eventuelle klienter (NTP er et multiplattform-program, som vil si at det kjører på mange arkitekturer og operativsystem..)
- Husk å ta notater etter hvert, til bruk i dokumentasjonen.
- NTP-serveren må kjøre en stund før den godtar GPS-en som såkalt ”syncronization peer”, dvs at PC-en brukes denne klokken til å synkronisere tiden mot. Ventetiden kan jeg bruke til å skrive litt om hvordan NTP virker.

Utstyr jeg trenger:

- PC med installert Linux operativsystem
- PC med installert Windows 9x/NT/XP operativsystem
- GPS Garmin 35-HVS
- Strømforsyning til GPS-en
- TTL->RS232 konverter for konvertering av PPS-pulsen (som er TTL-pulser) til RS232-pulser. Til dette bruker jeg interface-boks jeg tidligere har laget.
- 2 stk 9pin DSUB han (DB9M)
- 3 stk 9pin DSUB hun (DB9F)
- Kabel, 8-leder
- Koblingsskjema for kontaktene (vedlagt hovedoppgaven).
- GPS konfigurerings-programvare for Windows
- Avbiter
- Kniv
- Multimeter
- Oscilloskop
- Tang for montering av kabel på kontaktene
- Krympestrømper
- Varmluftspistol
- NTP dokumentasjon

Oppgave 3

3.1 Dokumentasjon og kalkulasjon

Tidsramme: Onsdag kl. 11.00 – 16.00

Fremdriftsplan:

- Skrive dokumentasjon for Realtime Linux program
- Dokumentasjonen må inneholde:
    - Programbeskrivelse
    - Installasjonsveiledning (både Realtime Linux + min programvare)
    - Brukerveiledning m/"screenshots"
    - Tilkoblingsdiagram for GPS signaler til PC-en
    - Flytdiagram
    - Programlisting
- Skrive dokumentasjon for installering og bruk av NTP (både server og klient) – her blir notatene fra installeringen en stor hjelp.
- Dokumentasjonen må inneholde:
    - Kort forklaring av NTP
    - Veiledning for hvordan anskaffe, kompilere og installere NTP
    - Konfigurasjon for server
    - Konfigurasjon for klient
    - Tilkoblingsdiagram for GPS signaler til PC-en via interface-boks.
    - "Screenshots" som viser hvilken nøyaktighet som ble oppnådd under testing

Utstyr jeg trenger:

- PC med installert Windows operativsystem og MS Office 2000, samt Acrobat Distiller for å konvertere dokumentene til PDF-format. Trenger også Adobe Photoshop for å behandle "screenshots"-ene (fjerne alle andre vinduer enn de som er relevante, osv…)
- Skriver – kommer til å skrive ut på skriverne som er tilkoblet nettverket.

Oppgave 4

4.1 Interne og Eksterne bus-systemer

Tidsramme: Torsdag kl. 08.00 – 12.00

Fremdriftsplan:

- Skrive om:
    - PCI
    - AGP
    - IDE
- Vil bruke internett som hjelpemiddel her, i tillegg til de kunnskaper jeg allerede har.

Oppgave 5

5.1 Kopimaskiner

Tidsramme: Torsdag kl. 12.00 – 16.00

Fremdriftsplan:

- Skrive om oppbyggingen og virkemåten til en analog s/h kopimaskin
- Som hjelpemidler vil jeg bruke:
    - Oppgave jeg selv skrev da jeg gikk på VKII
    - Annet skolemateriell
    - Internett

Annet

Gjennom hele oppgaven vil jeg bruke min private PC til å skrive notater, og lage dokumentasjon. Her har jeg installert alt jeg trenger av software, som blant annet omfatter:

- MS Office 2000
- Acrobat Distiller
- SSH Secure Shell for overføring av .HTML sider til webserver
- Adobe Photoshop for billedbehandling

Gjennomføring

Oppgave 1

1.1 Installasjon av Linux RedHat 8.0 og "Realtime Linux"

1.1.1 Montering av harddisk

Jeg hadde en IBM Deskstar 40GB harddisk liggende, som passet fint å bruke til denne oppgaven. Fremgangsmåten for å få denne installert i PC-en min (som nevnt i forberedelsesdelen bruker jeg PC-en jeg har hatt som arbeidsmaskin gjennom læretiden) var som følger:

- Plassere kabinettet på en antistatisk matte (som var klargjort og jordet på forhånd) – her er det viktig å merke seg at mange PC-er har gummiføtter under, som isolerer dem ifra underlaget – men hele vitsen med å plassere den på matten er jo å få kontakt med den, så her brukte jeg en ledning tilkoblet et kontaktpunkt på matten i ene enden, og en krokodilleklemme tilkoblet kabinettet i andre enden.
- Ta av sidedekselet på kabinettet, ta ut harddiskholderen, montere harddisken i denne, koble til strøm- og IDE-kabel, og sette harddiskholderen tilbake i kabinettet (kabinettet er av type ChiefTech Dragon, som er bygget opp av løse moduler kun festet med klips). Jumperne på den nye harddisken var på forhånd innstilt slik at denne disken ble "master". Jeg beholdt også den gamle disken i maskinen, og denne ble satt som "slave".
- Koblet til alt utstyr på maskinen, som skjerm, tastatur, mus og strømkabel, og slo den på for å sjekke at alt var i orden før jeg satt på igjen sidedekselet. BIOS-en i PC-en var allerede satt til auto-deteksjon av disker på både "primary master/slave" og "secondary master/slave", så her trengte jeg ikke gjøre noe. Den nye disken ble detektert, og jeg kunne begynne installasjonen.

1.1.2 Installering av RedHat 8.0

RedHat 8.0 distribusjonen kommer på 5 CD-er, hvorav kun de 3 første er nødvendig for installeringen. CD 4 inneholder kildekode til alle programmene, og CD 5 inneholder mye dokumentasjon. CD1 kan bootes fra, noe som eliminerer behovet for oppstartsdiskett. Installasjonen er rimelig enkel og grei, du får spørsmål om hvilke(t) språk du vil benytte, hvilken tastatur og hvilken mus du har og hvilken tidssone du befinner deg i. Deretter må harddisken(e) partisjoneres og formateres, men dette kan du velge å la installasjonsprogrammet gjøre automatisk for deg. Nettverket må konfigureres, og du må velge hvilke pakker – eller hvilken programvare – du vil installere. RedHat kommer med en hel del programmer, og det kan virke unødvendig å installere alt dette – men jeg valgte å kjøre en full installasjon siden jeg allikevel har harddiskplass til det. En fordel med det er at du slipper å måtte finne frem igjen CD-ene (eller lete på internett) for å installere programvare det viser seg at du får bruk for senere.

1.1.3 Installering av Realtime Linux 3.1

Realtime Linux kan lastes ned fra www.fsmlabs.com, men jeg benyttet meg av en lokal kopi fra vår egen server: http://www2.ifjf.uib.no/gunstamp/rtlinux-3.1.tar.gz og http://www2.ifjf.uib.no/gunstamp/rt-patch-2.4.18.tar.bz2
Linux kernel-2.4.18 lastet jeg ned fra: http://www.kernel.org/pub/linux/kernel/v2.4/linux-2.4.18.tar.bz2
Filene ble plassert i `/home/trond/`

Installasjonsprosedyren er som følger:

Du må være *root*, eller såkalt *superuser*, for å installere en ny kernel. Root-privilegier skaffer du deg med kommandoen:

```
# su -
Password:
```

Den nye kernelen pakkes ut under `/usr/src/rtlinux`. Den vil bli pakket ut i en undermappe kalt `linux`, så den må omdøpes til `linux-2.4.18-rtl31`, før det opprettes en symbolsk link kalt `linux`, som peker til `linux-2.4.18-rtl`. Når det er gjort, må realtime-patchen pakkes ut, og kernelen patches:

```
# cd /usr/src
# mkdir rtlinux
# cd rtlinux
# tar -xjvf /home/trond/linux-2.4.18.tar.bz2
  …
# mv linux linux-2.4.18-rtl31
# ln -s linux-2.4.18-rtl31 linux
# tar -xjvf /home/trond/rt-patch-2.4.18.tar.bz2
# cd rt-patch-2.4.18
# gunzip patch_rtlinux-3.1.gz
# gunzip patch_rtl-2.4.18.gz
# cd ../linux
# patch -p1 < ../rt-path-2.4.18/patch_rtl_2.4.18
  …
```

Nå må den nye kernelen konfigureres, og for å gjøre det enkelt, bruker vi konfigurasjonen som kom med RedHat. Deretter kompileres kernelen:

```
# cp /boot/config-2.4.18-15 .config
# make oldconfig
# make dep; make bzImage; make modules; make modules_install; make
install;
```

Etter en vellykket kompilering, må du legge til den nye kernelen i `/etc/lilo.conf`, som er konfigurasjonsfilen til *lilo* (Linux Loader), kjøre `/sbin/lilo` for å oppdatere konfigurasjonen, og restarte maskinen. Husk å velge ”rtlinux” som kernel:

```
# pico /etc/lilo.conf

< -- legg til disse linjene på slutten av filen: -- >

image=/boot/vmlinuz-2.4.18-rtl31
        label=rtlinux
        initrd=/boot/initrd-2.4.18-rtl31.img
        read-only
        append=”root=LABEL=/1”

< -- her til -- >

# /sbin/lilo
Added linux *
Added rtlinux
# reboot
```

Når maskinen så har kommet opp igjen, kan Realtime Linux kompileres og installeres. Først må Realtime Linux 3.1 pakkes ut, og rt-patch-2.4.18 tilføyes:

```
# su -
Password:
# cd /usr/src/rtlinux
# tar -xzvf /home/trond/rtlinux-3.1.tar.gz
  …
# cd rtlinux-3.1
# patch -p1 < ../rt-patch-2.4.18/patch_rtlinux-3.1
  …
# ln -sf /usr/src/rtlinux/linux-2.4.18-rtl31 linux
```

Deretter må Realtime Linux konfigureres, før det blir kompilert og installert. Konfigureringen blir gjort enkelt vha kommandoen `make menuconfig`, men standardoppsettet passer fint til denne oppgaven, så det er ikke nødvendig å gjøre noen forandringer:

```
# make menuconfig
# make dep; make; make devices; make install
```

Nå kan Realtime Linux startes og stoppes vha kommandoen `rtlinux start|stop`. For å teste installasjonen, kan en kjøre følgende testprogram:

```
# cd examples/frank
# make test
FIFO 1: Frank
FIFO 2: Zappa
FIFO 1: Frank
FIFO 2: Zappa
…
```

Realtime Linux 3.1 er nå installert.

1.1.4 Litt om ”sanntidsoperativsystem”

Sanntidsoperativsystem er en type operativsystem som brukes der man har spesielle tidskrav til når en oppgave må være utført. Vanligvis er maskinen operativsystemet er installert på, tilkoblet et eksternt system som den mottar signaler og data fra. Maskinen kan og bli brukt til å styre det eksterne systemet.

Aktiviteter innenfor et sanntidsbaset system kan deles inn i to grupper: periodiske (kjører med et bestemt intervall) og aperiodiske (for eksempel avbrudd).

Ved periodiske oppgaver stilles det gjerne strenge krav til at oppgaven må startes på riktig tidspunkt og være ferdig innen en viss tidsfrist. Eksempel på det kan være ved sampling av et målesignal. Ved en frekvens på 1kHz betyr det at oppgaven må kjøres en gang hvert millisekund. Et ikke-sanntidsoperativsystem som for eksempel Windows eller Linux kan ikke garantere at oppgaven blir kjørt med nøyaktig 1ms intervall. Dette er på grunn av at systemet er utviklet for å kunne gjøre mange oppgaver samtidig. Mens i et sanntidsoperativsystem, vil sanntidsoppgaven alltid ha høyest prioritet, noe som vil si at alle andre oppgaver blir avbrutt når sanntidsoppgaven trenger prosessoren. Så snart den er ferdig å kjøre, blir kontrollen returnert til prosessen som ble avbrutt.

Aperiodiske oppgaver er oppgaver som blir styrt av hendelser fra en ekstern prosess. PC-en blir varslet om hendelsen via avbrudd (eng; interrupts). Her er det kritisk at PC-en klarer gi respons på hendelsen så raskt som mulig. Sanntidsoperativsystemet Realtime Linux har en ”worst case latency”, det vil si en minimum garantert responstid, på 30us med relativt gammel hardware. I en godt konfigurert PC kan responstiden være helt nede i 10us. Med et ikke-sanntidsoperativsystem kan responstiden variere fra brøkdeler av et millisekund til flere hundre millisekunder, avhengig av hvilke prosesser som kjører og hvor hardt belastet prosessoren er. Et sanntidsoperativsystem har den fordelen at sanntidsoppgaver alltid får prosessoren når de trenger den – uansett hvor stor belastning den er under.

1.2 Program som kjører under Realtime Linux – OBSMCS

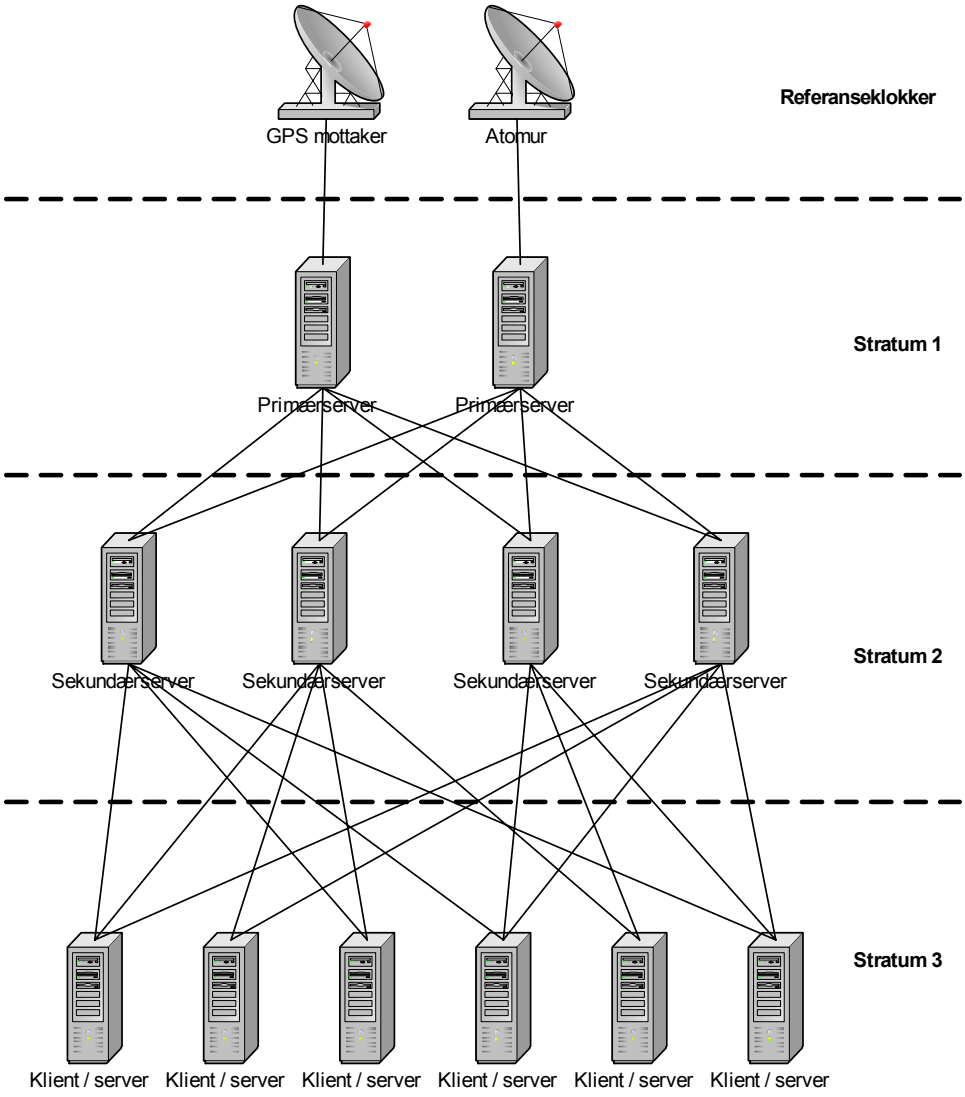
Programmet er dokumentert under oppgave 3.

Oppgave 2

Tidssynkronisering av datamaskiner i nettverk: NTP

NTP står for Network Time Protocol og er et system for synkronisering av klokkene til datamaskiner i et nettverk. Det brukes for eksempel i stor grad på internett for å distribuere riktig tid. Alle klokker driver, det vil si at de går enten litt fortere eller litt saktere enn det de skal, og klokken i PC-er er ikke noe unntak. Hvor nøyaktig klokken i en PC-er, varierer fra maskin til maskin. Enkelte kan drive med mange minutter for dagen, mens andre igjen bare driver med noen millisekunder.

NTP-systemet er organisert i en trestruktur sortert etter stratum, dvs hvor langt borte maskinen er ifra referanseklokken på toppen. Primærserveren (serveren som blir synkronisert av referanseklokken) har stratum 1. En sekundærserver har stratum 2, siden den er to enheter borte fra referanseklokken, osv.



Figur 1 - Eksempel på NTP nettverk

Som en ser av figuren på forrige side, vil en server som synkroniseres mot en stratum *n* server, kjøre i stratum *n+1*. Det er satt en øvre grense på stratum 15. Grunnen til at NTP er organisert på denne måten, er å sikre at tiden blir sendt over *nedover* stratumene (en stratum 3 server kan for eksempel ikke brukes til å synkronisere en stratum 2 server).

Referanseklokkene har alltid stratum 0. Det er mange typer referanseklokker som kan brukes, men det som er viktig er *nøyaktigheten* til disse. De beste klokkene som er tilgjengelig er atomur, men de er også meget dyre.

Et godt (men selvfølgelig ikke like nøyaktig) alternativ til dette er å bruke en GPS med PPS (Pulse-Per-Second) utgang. GPS-er mottar tiden fra satellitter i bane rundt jorden, og disse satellittene inneholder et atomur som blir stilt med jevne mellomrom. GPS-er uten PPS utgang kan også brukes, men det vil gjøre mer skade enn nytte, siden synkroniseringen da baseres kun på NMEA setninger som blir lest som seriedata, og tidspunktet setningen blir sendt ut på kan variere med opptil et par hundre millisekunder.

Et tredje alternativ som også er mye brukt, er å bruke radiour. Disse klokkene synkroniseres ved hjelp av radiobølger mot et offentlig atomur.

Referanseklokkene synkroniserer altså stratum 1 servere, og disse blir brukt til å synkronisere stratum 2 servere, som igjen blir brukt til å synkronisere stratum 3 servere, osv, osv. Dette kan foregå på tre forskjellige måter:

- *Polling (spørring)* – navn / IP til serveren konfigureres manuelt, og klienten spør serveren om tiden med jevne mellomrom
- *Broadcast (kringkasting)* – serveren sender tiden ut på nettet på en kringkastingsadresse, hvor klienter kan snappe den opp
- *Peer-to-peer (punkt til punkt)* – serveren sender tiden direkte til en klient

Klient		Server
Sender en forespørsel om tid til serveren. Pakken blir stemplet med sendingstidspunktet.	⇨	
	⇦	Serveren mottar pakken, og lagrer klokkeslettet i den. Når pakken blir sendt tilbake, blir den stemplet med sendingstidspunkt.
Klienten mottar pakken med tid fra serveren, og setter igjen et tidsstempel på den.		

Tabell 1 - Hendelsesforløp når en klient sender forespørsel om korrekt tid til server (polling)

Tiden det tar å sende pakken én vei blir anslått til å være *lik halvparten av den totale forsinkelsen minus tiden det tar for serveren å prosessere forespørselen.* Altså går NTP ut fra at det tar like lang tid å sende pakken begge veier. Det er selvfølgelig ikke alltid tilfelle, men over tid vil det bli riktig. NTP bruker mange forskjellige filter og algoritmer for å kunne anslå mest mulig korrekt tid. Tiden klienten mottar fra serveren vil heller ikke bli godtatt før flere forespørsler har blitt sendt. Typisk går det fem minutter (og fem forespørsler) fra du starter

NTP, til en server vil bli godtatt som synkroniseringskilde. Synkroniserer du mot flere servere, vil NTP plukke ut de av serverne som antas å være mest pålitelig. De dårligste serverne vil ikke bli brukt i synkroniseringen. Bruker du for eksempel fire servere, der to av dem gir tilnærmet lik tid, og de to andre gir helt forskjellige tider, vil kun de to serverne som er enige om tiden bli brukt.

Nøyaktigheten som er mulig å oppnå med NTP kommer selvfølgelig an på hvor langt nede i hierarkiet du befinner deg, og hvor mye trafikk det er på nettverket. Nøyaktigheten over internett kan variere fra 5ms til 100ms, alt etter kvaliteten på linjen, mengden trafikk, osv.

Med PPS synkronisering (kun stratum 1 servere) får du lett en nøyaktighet på mindre enn 50us, gjerne helt ned i 10us, men det kommer an på hvor rask PC-en er og hvor mye annet den har å gjøre.

Over et lokalt nettverk der pakkene ikke må gå igjennom altfor mange rutere, og med liten trafikk, er vanlig nøyaktighet mindre enn 1ms. Bedre tider kan oppnås med et lite nettverk, men allikevel gjelder det bare så lenge det er liten trafikk.

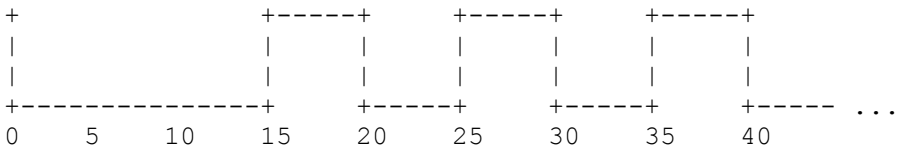
Beskrivelse av installasjon, konfigurasjon og test blir gjennomgått i oppgave 3b!

Oppgave 3 – Dokumentasjon

3a OBSMCS – Ocean Bottom Seismograph Master Clock Syncronization

Om obsmcs

Programmet sender ut OBS klokkesynkroniseringspulser, basert på PPS (Pulse-Per-Second) pulser og NMEA setninger som blir mottatt fra en GPS. Utpulsen har følgende mønster:



Manglende puls markerer starten på et nytt minutt.

PPS pulsen fra GPS-en, mottas via pinne acknowledge interruptet på parallellporten. Pulsen som tilkobles må være aktiv høy, 0 volt i lav tilstand, og 5-15 volt i høy tilstand.
NMEA setningene mottas som seriedata via serieport 1.

Programmet er delt i to; en kernelmodul som kjører i realtime og tar seg av synkronisering av utpulsen mot PPS, og et brukerprogram (monitor) som kommuniserer med realtime modulen, initialiserer den, tar imot data fra den og GPS-en, og presenterer alt for brukeren på skjermen.

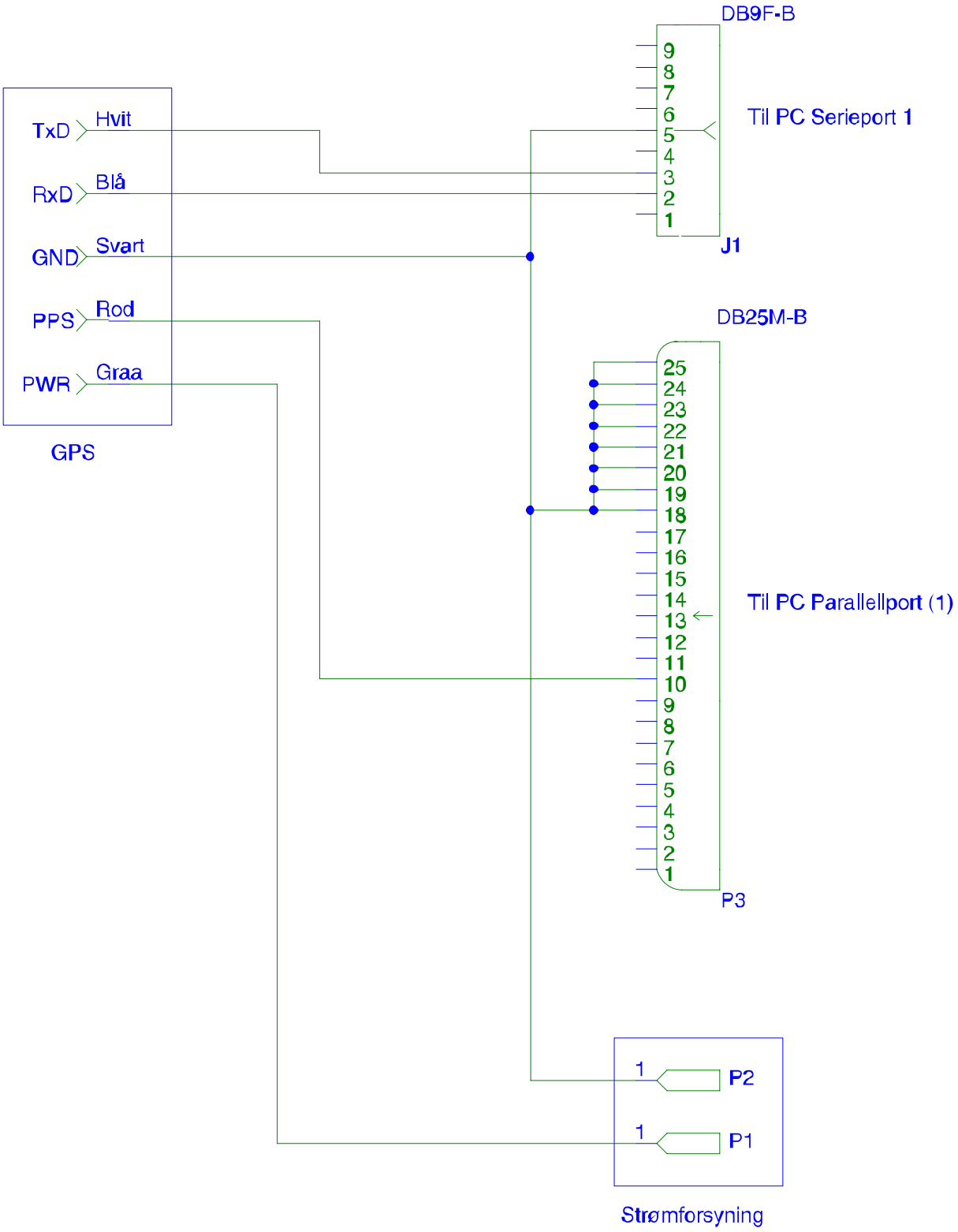
Installasjonsveiledning

Programmet lastes ned ifra følgende adresse:
http://www2.ifjf.uib.no/trond/fagprove/obsmcs.tar.gz

Deretter må programmet pakkes ut og kompileres før det kan kjøres. Det er en forutsetning at Realtime Linux 3.1 er installert på forhånd (se avsnitt 1.1.3 for instruksjoner).

```
# tar -xzvf obsmcs.tar.gz
# cd obsmcs
# make
```

Før programmet kan taes i bruk, må også GPS-en kobles til i henhold til koblingsskjema på neste side.



Figur 2 - Koblingsdiagram for tilkobling av GPS til PC

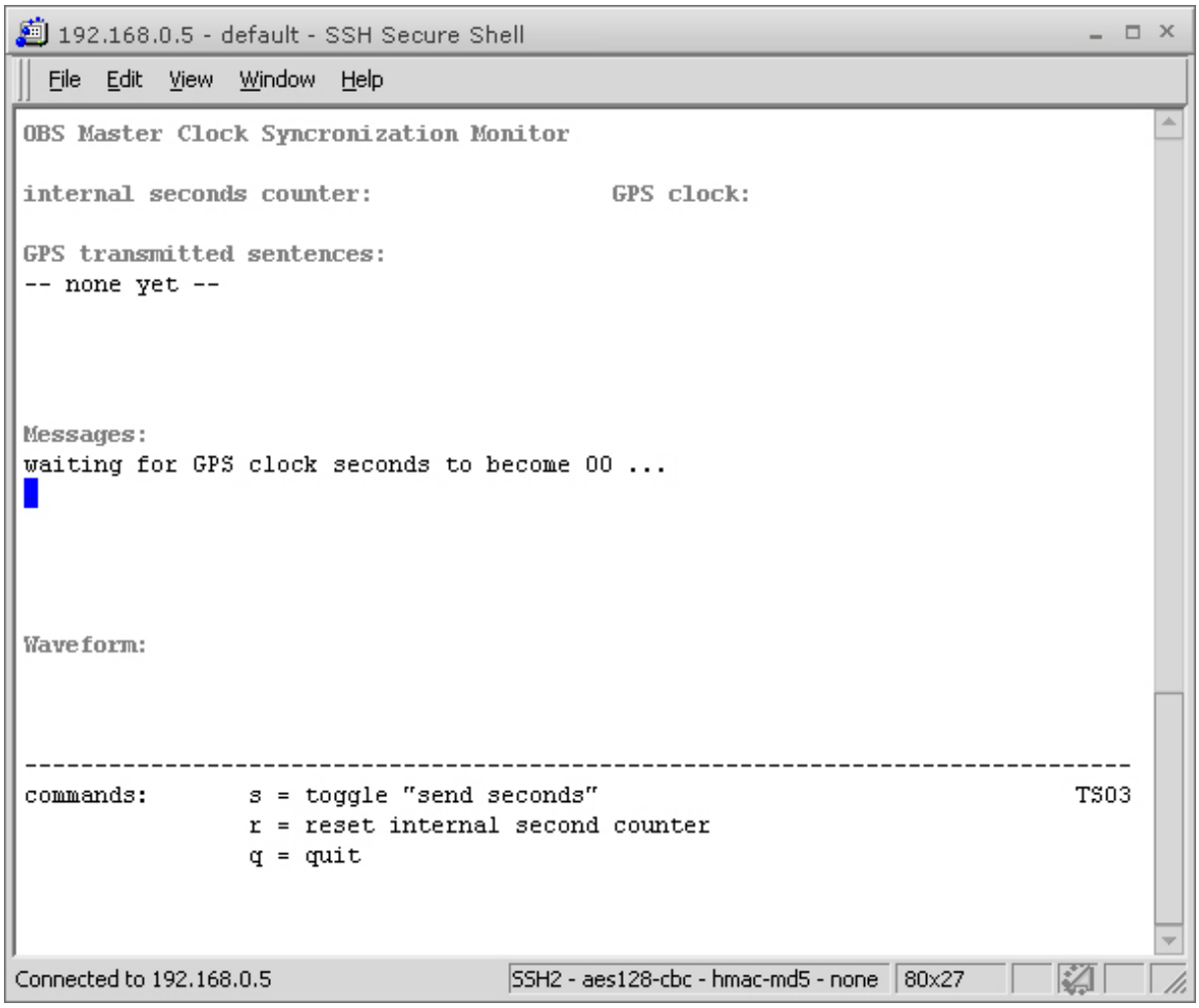
Brukerveiledning

Først må realtime modulen aktiveres. For å kunne gjøre dette, må du være superuser.

```
# su -
Password:
# insmod obsmcs.o
```

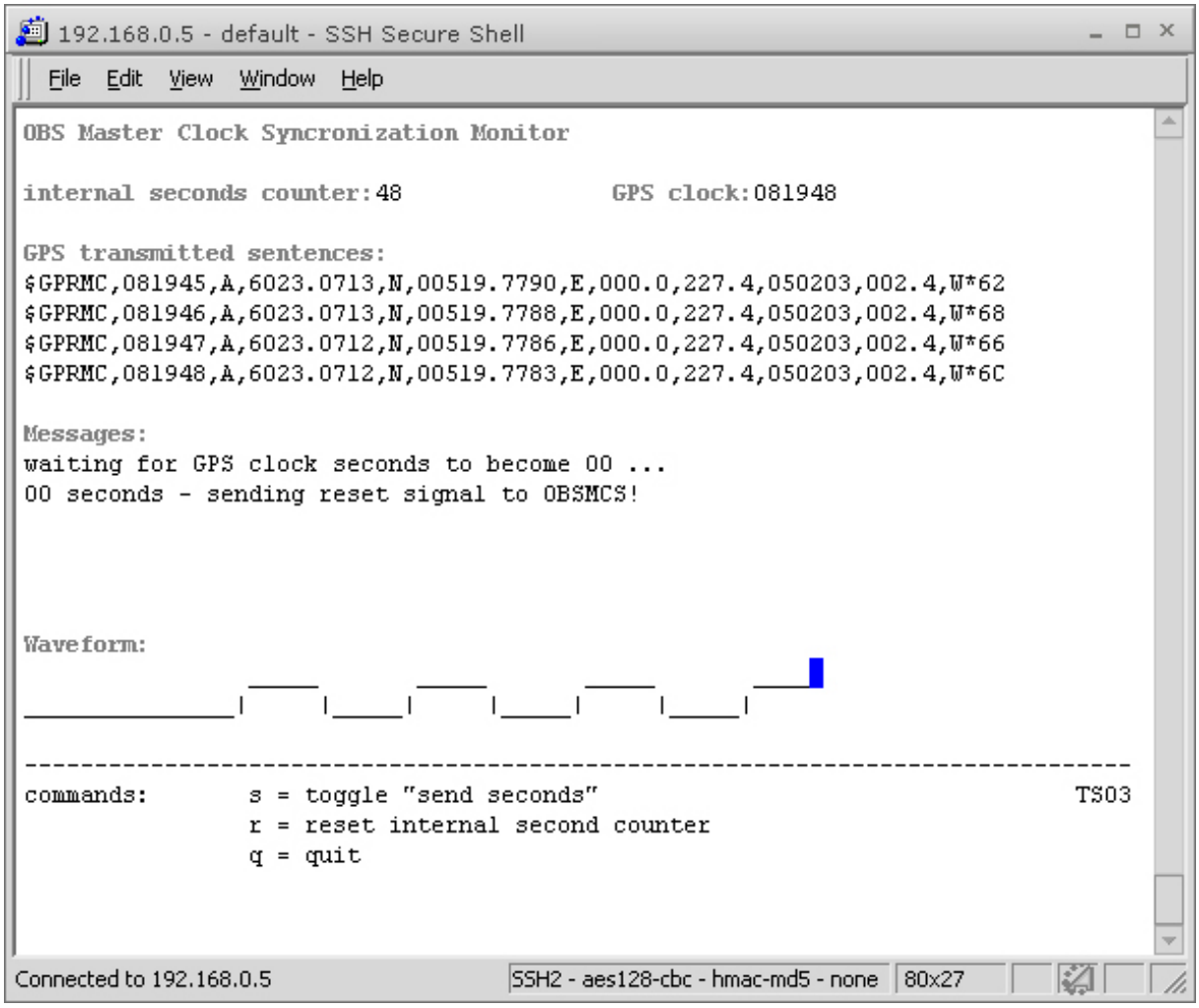
Fremdeles som superuser, starter du monitorprogrammet med kommandoen:

```
# ./monitor
```



Figur 3 - obsmcs; oppstartsbilde (kjørt gjennom SSH Secure Shell)

Så snart programmet begynner å motta data fra GPS-en, vil NMEA setningene som kommer inn bli vist under ”GPS transmitted sentences”. GPS klokken vil bli ekstrahert fra disse setningene, og blir vist etter ”GPS clock”. Det er denne klokken som brukes til å synkronisere den interne sekundtelleren som igjen brukes for å generere riktig puls ut. *NB! Pulsene vil ikke være riktig før første gang klokken har rundet ett minutt!*



Figur 4 - obsmcs etter å ha kjørt en stund

Legg merke til at under ”Waveform:” blir utpulsen tegnet opp – en strek for hvert sekund i ett minutt. Ved begynnelsen på et nytt minutt, blir ”wavform”-vinduet blanket ut, og det hele begynner på nytt igjen.

Kommandotaster:

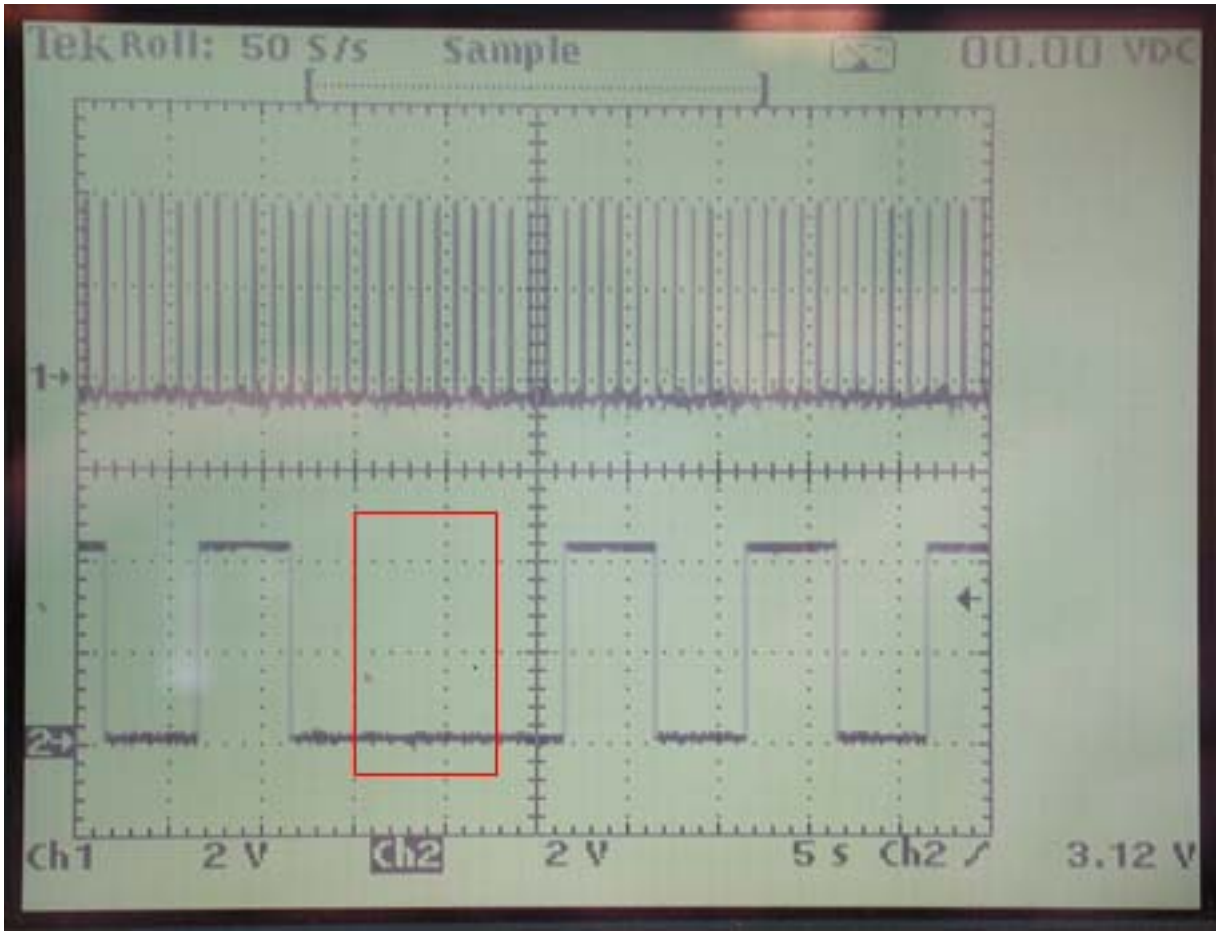
S Aktiverer/deaktiverer tilbakemelding fra kernelmodulen (dvs den vil sende/ikke sende tilbake den interne sekundtelleren).

R Resetter den interne sekundtelleren i kernelmodulen neste gang klokken runder et minutt.

Q Avslutter monitorprogrammet

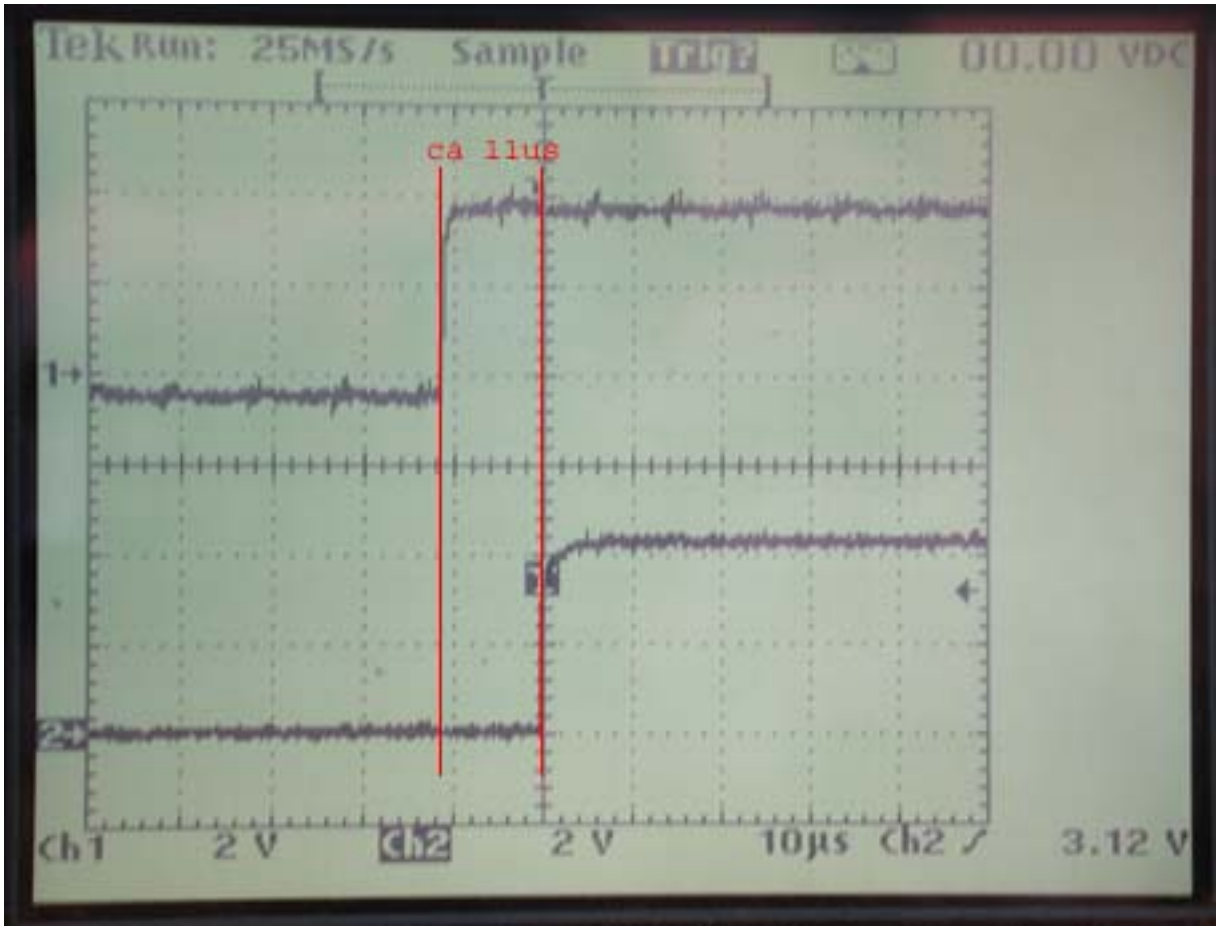
For å fjerne kernelmodulen igjen, bruker du kommandoen:

```
# rmmod obsmcs
```



Figur 5 - Bilde av oscilloscope som viser PPS og obs klokkepuls

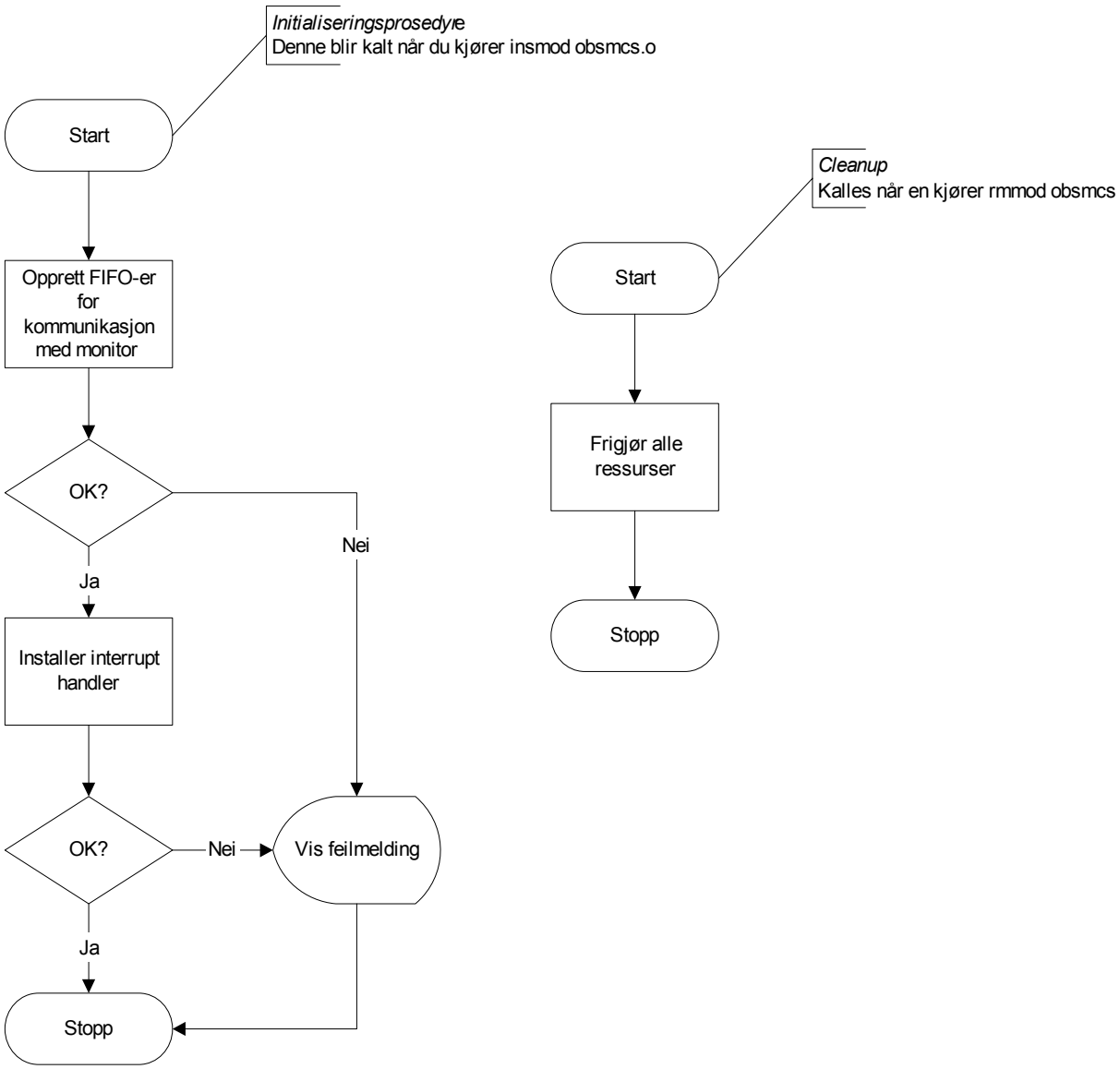
Under testkjøring av programmet ble det koblet til et oscilloskop for å kontrollere at pulsen ut (obs klokkepuls) stemte overens med spesifikasjonene.
Øverst, på kanal 1, ser du PPS pulsene som kommer fra GPS-en.
Nederst, på kanal 2, ser du obs klokkepulsene – fem sekunder høy og fem sekunder lav. Den manglende pulsen (markert med rød firkant) indikerer starten på et nytt minutt.



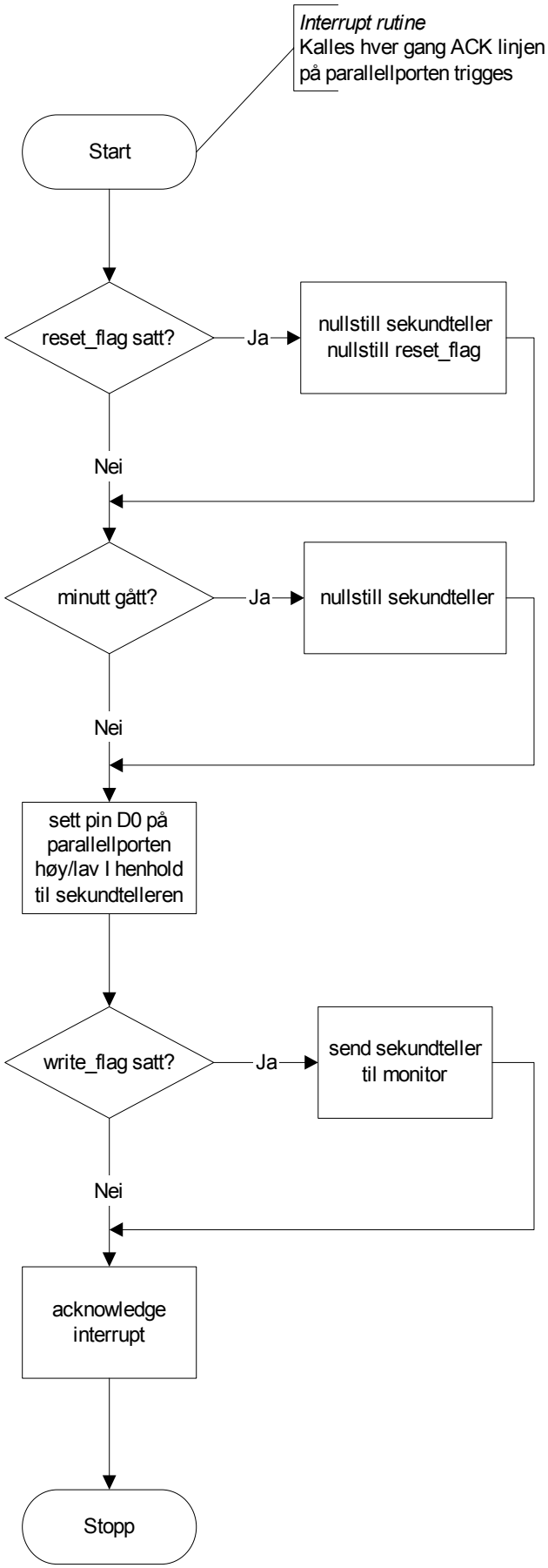
Figur 6 - Bilde av oscilloscope som viser tidsforsinkelse av obs klokkepuls

Nytt bilde fra oscilloskop, som viser forskyvningen av obs klokkepulsen i forhold til PPS pulsen. Forsinkelsen holder seg innenfor 10-12us, noe som er meget bra når en tenker på at pulsen er generert av en PC.

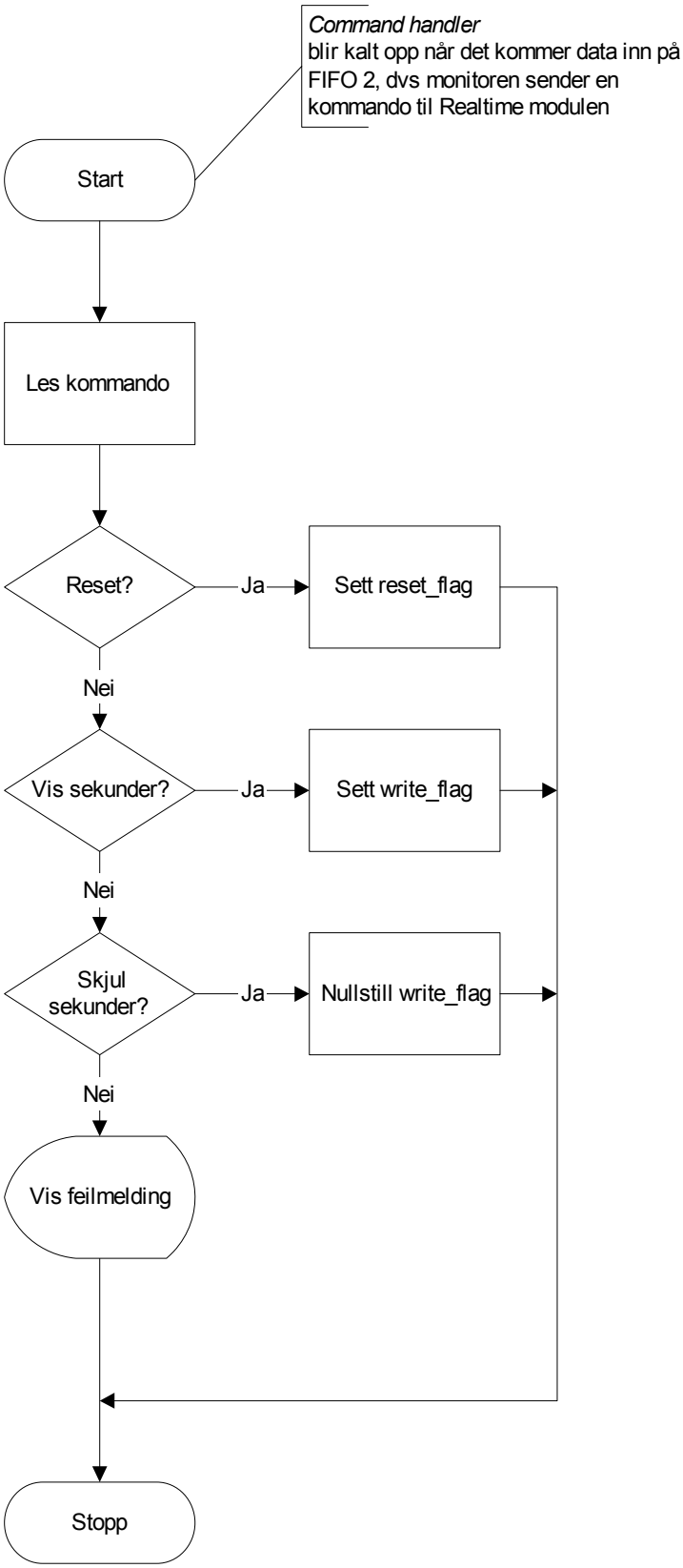
Flytdiagram for obsmcs



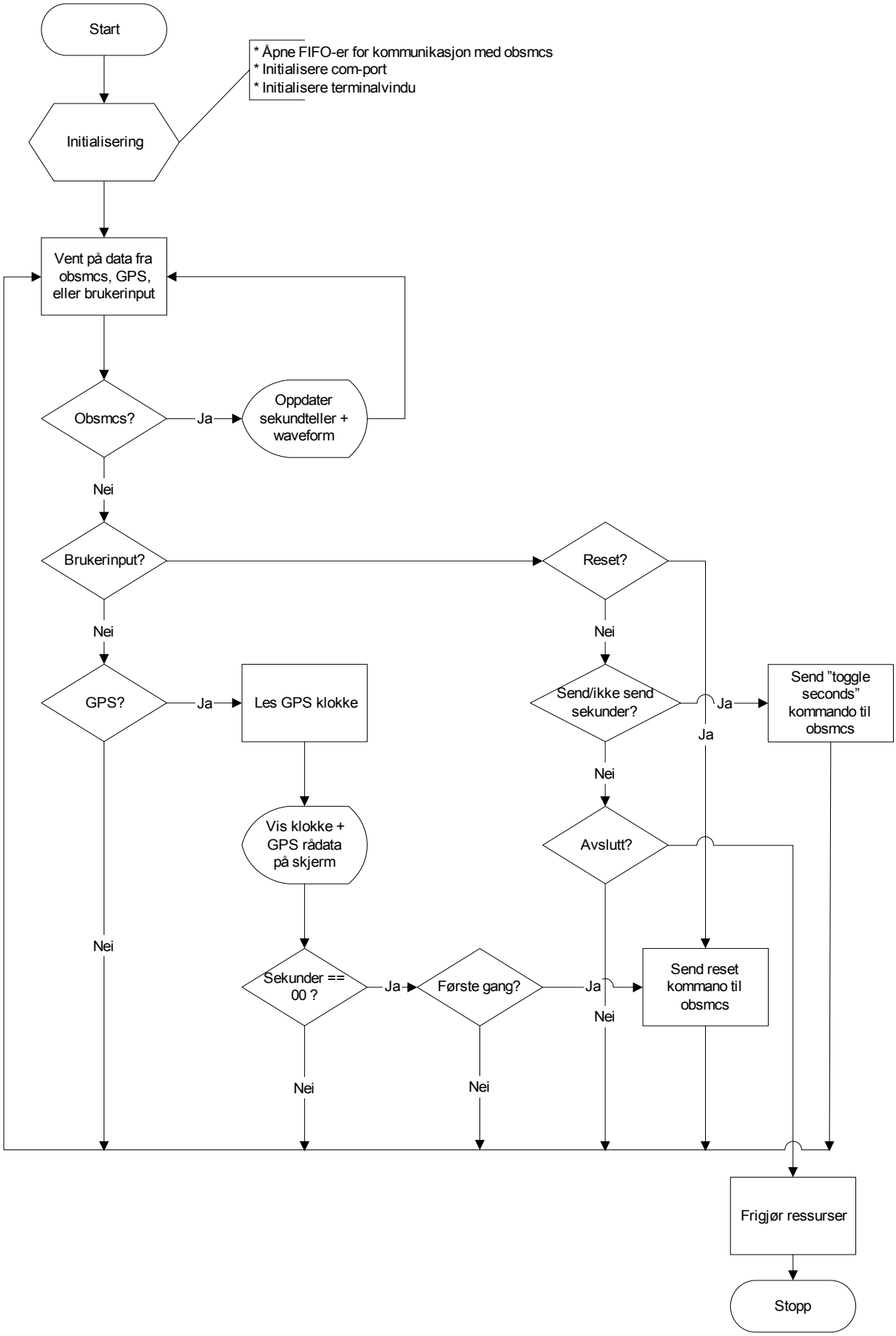
Figur 7 - Flytdiagram for obsmcs; init_module og cleanup_module



Figur 8 - Flytdiagram for obsmcs; ack_isr



Figur 9 - Flytdiagram for obsmcs; cmd_handler



Figur 10 - Flytdiagram for obsmcs monitor

Programlisting

obsmcs.c *(Realtime Linux kernelmodul)*

```
/*
 * obsmcs  --  OBS Master Clock Syncronization
 *

*************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*

*************************************************************************
 *
 *
 * Description : Based on GPS Pulse-per-second (PPS) and NMEA formatted
 *               serial input (GPRMC sentence), the program issues OBS
 *               Master Clock synchronizing pulses. The pattern is as
 *               follows:
 *
 *
 *
 *                                  _____       _____
 *               |                  |     |     |     |
 *               |__________________|     |_____|     |_____
 *               0     5    10    15    20    25    30   [second of minute]
 *
 *
 *
 *
 *               GPS PPS pulse assumed to be active high, of TTL level,

 *               and of min 1 ms duration. Leading edge marks second
 *               boundary.
 *
 *                     ___
 *                    |   |
 *               _____|   |_____
 *
 *                    ^
 *                    |___ Second boundary
 *
 *
 *
 *               ( Program is tested with Garmin GPS mod 35-HVS. )
 *
 *
 * PC ports:
```

```
 *              LPT1 :    Pin 2 (D0)    = Output, OBS Master Clock
calibration
 *                                        signal.
 *                        Pin 10 (ACK)  = Acknowledge interrupt, PPS from
GPS
 *                                        The positive edge of a signal on
 *                                        this pin can be used to generate
 *                                        an interrupt (assumed INT7)
 *                        Pin 18 (GND)  = GND
 *
 *              COM1 :    Pin 2 (RXD)   = NMEA sentences from GPS
 *                        Pin 5 (GND)   = GND
 *
 *
 *
 * NMEA sentence used: GPRMC - Recommended Minimum Specific GPS/TRANSIT
Data
 *
 *
$GPRMC,<1>,<2>,<3>,<4>,<5>,<6>,<7>,<8>,<9>,<10>,<11>,<12>*hh<CR><LF>
 *      <1>  UTC time of position fix, hhmmss format
 *      <2>  Statuc, A = Valid position, V = NAV receiver warning
 *      <3>  Latitude, ddmm.mmmm format (leading zeros will be transmitted)
 *      <4>  Latitude hemisphere, N or S
 *      <5>  Longitude, dddmm.mmmm format (leading zeros will be
transmitted)
 *      <6>  Longitude hemisphere, E or W
 *      <7>  Speed over ground, 0000.0 to 1851.8 knots (leading zeros will
be transmitted)
 *      <8>  Course over ground, 000.0 to 359.9 degrees, true (leading
zeros will be transmitted)
 *      <9>  UTC date of position fix, ddmmyy format
 *      <10> Magnetic variation, 000.0 to 180.0 degrees (leading zeros will
be transmitted)
 *      <11> Magnetic variation direction, E or W (westerly variation adds
to true course)
 *      <12> Mode indicator (only output if NMEA 2.30 active), A =
Autonomous,
 *           D = Differential, E = Estimated, N = Data not valid
 *
 */



#ifndef MODULE
#define MODULE
#endif
#define _GNU_SOURCE
#include <linux/config.h>
#include <linux/errno.h>
#include <linux/ioport.h>
#include <linux/module.h>
#include <linux/kernel.h>
#include <linux/version.h>
#include <asm/system.h>
#include <asm/io.h>
#include <rtl.h>
#include <rtl_sync.h>
#include <rtl_sched.h>
#include <rtl_core.h>
#include <rtl_fifo.h>
#include "serp.h"
```

```
#include "common.h"


void cleanup_module (void);
unsigned int ack_isr(unsigned int, struct pt_regs*);
unsigned int ser_isr(unsigned int, struct pt_regs*);


static unsigned short seconds=0;
static unsigned int lpt1_data=0;
static unsigned int reset_flag=0;
static unsigned int write_flag=0;


/*
 *  interrupt handler for parallell port
 */
unsigned int ack_isr(unsigned int irq, struct pt_regs *regs)
{
  if(reset_flag) {        // reset if reset_flag==1
    seconds = 0;
    reset_flag = 0;
  }
  if(seconds >= 60)       // no more than 60 seconds per minute...
    seconds = 0;          // (below) output 5 second pulses
  if(((seconds % 5) == 0) && !(seconds == 5) && !(seconds==10)) {
    lpt1_data ^= 1;
    outb(lpt1_data, LPT_PORT);
  }
  if(seconds == 0) {
    lpt1_data = 0;
    outb(lpt1_data, LPT_PORT);
  }
  if(write_flag)         // send seconds counter back to monitor
    rtf_put(3, &seconds, sizeof(seconds));

  seconds++;             // increase second counter

  /* acknowledge interrupt and return */
  rtl_hard_enable_irq(LPT_IRQ);
  return(0);
}


/*
 *  command handler
 *  takes care of commands sent from the monitor
 */
int cmd_handler(unsigned int fifo)
{
  unsigned int cmd;

  /* read command (unsigned integer) from rtf2 */
  rtf_get(2, &cmd, sizeof(int));

  switch(cmd) {
  case 0: // STOP
    break;
  case 1: // RESET
    reset_flag = 1;
    break;
  case 2: // OUTPUT SECONDS ON
```

```
    write_flag = 1;
    break;
  case 3: // OUTPUT SECONDS OFF
    write_flag = 0;
    break;
  default: // UNKNOWN COMMAND
    rtl_printf("obsmcs.o: ERR: COMMAND %d UNKNOWN\n", cmd);
    break;
  }
  return 0;
}


////////////////////////////////////////////////////////////////////////
/

/*
 *  initialize resources
 */
int init_module (void)
{
  int i;

  /* create /dev/rtf2  (for commands) */
  i = rtf_create(2, 4096);
  if(i != 0) {
    rtl_printf("obsmcs.o: ERR: can not create fifo 2 (err %d)\n", i);
    return 1;
  }

  /* create /dev/rtf3  (for sending second counter to user space) */
  i = rtf_create(3, 4096);
  if(i != 0) {
    rtl_printf("obsmcs.o: ERR: can not create fifo 3 (err %d)\n", i);
    return 1;
  }

  /* initialize the command handler for rtf2 */
  i = rtf_create_handler(2, &cmd_handler);
  if(i != 0) {
    rtl_printf("obsmcs.o: ERR: can not create handler 2 (err %d)\n", i);
    return 1;
  }

  /* parallel port setup */
  rtl_hard_disable_irq(LPT_IRQ);
  if(rtl_request_irq(LPT_IRQ, ack_isr) != 0) {
    rtl_printf("obsmcs.o: Couldn't create interrupt handler for parallel
port\n");
    return 1;
  }
  rtl_hard_enable_irq(LPT_IRQ);  // enable ACK interrupt
  outb(0x10, LPT_PORT+2);

  /* Let the user know the module got inserted properly */
  rtl_printf("obsmcs.o loaded successfully\n");
  return 0;
}


/*
 *  free all used resources
```

```
 */
void cleanup_module (void)
{
  rtf_destroy(2);  // destroy FIFO's
  rtf_destroy(3);

  /* Free parallel port IRQ (7) */
  outb(0x00, LPT_PORT+2);
  rtl_hard_disable_irq(LPT_IRQ);
  rtl_free_irq(LPT_IRQ);

  /* Say we're done */
  rtl_printf("obsmcs.o unloaded.\n");
}
////////////////////////////////////////////////////////////////////////////
//
```

common.h :

```
/* common header file for obsmcs.c and monitor.c
 */

#define LPT_PORT 0x378
#define LPT_IRQ 7
#define RTC_IRQ 8
```

serp.h :

```
/* header file for obsmcs.c
*/
void rt_com_exit( void );
void rt_com_init( void );
int  rt_com_read( int, char *, int );
void rt_com_setup( unsigned, unsigned, unsigned, unsigned );
void rt_com_write( int, char *, int );

/*--------------------------------------------------------------------------*/
#define TRIGGER 4
#define FIFO 4

/*--------------------------------------------------------------------------*/
/* status masks, several bits may be set simultaneously     */
/* status information is returned by LineStatus and as      */
/* the high byte of COMData                                 */

#define DATA_READY   0x01     /* not an error               */
#define OVERRUN      0x02     /* error detected by hardware */
#define PARITY       0x04     /* error detected by hardware */
#define FRAME        0x08     /* error detected by hardware */
#define BREAK        0x10     /* not an error               */
```

```
#define BUFFER_FULL  0x80     /* error detected by software */
#define TXB_EMPTY    0x20     /* not an error               */
#define HARD_ERROR   (OVERRUN | PARITY | FRAME | BREAK)

/* port offsets */
#define RXB  0x00 /* Receive register  (READ)  */
#define TXB  0x00 /* Transmit register (WRITE) */
#define IER  0x01 /* Interrupt Enable          */
#define IIR  0x02 /* Interrupt ID              */
#define FCR  0x02
#define LCR  0x03 /* Line control              */
#define MCR  0x04 /* Modem control             */
#define LSR  0x05 /* Line Status               */
#define MSR  0x06 /* Modem Status              */
#define DLL  0x00 /* Divisor Latch Low         */
#define DLM  0x01 /* Divisor Latch High        */

#define PARITY_EVEN  0        /* for port initialisation    */
#define PARITY_ODD   1
#define PARITY_NONE  2

/* bit masks which may be written to the MCR using ModemControl */
#define DTR          0x01     /* data Terminal Ready        */
#define RTS          0x02     /* Request To Send            */
#define Out1         0x04
#define Out2         0x08
#define LoopBack     0x10

#define RT_BUF_SIZ 64

struct rt_buf_struct{
  int  head;
  int  tail;
  char buf[ RT_BUF_SIZ ];
};

struct rt_com_struct{
  int magic;
  int baud_base;
  int port;
  int irq;
  int flag;
  void (*isr)(void);
  int type;

  int ier;  /* copy of chip register */

  struct rt_buf_struct ibuf;
  struct rt_buf_struct obuf;
};

struct rt_trans_buf{
  char odat[512];
  int ion;
  int iox;
};

/*-----------------------------------------------------------------------*/

#define BASE_BAUD 115200
#define STD_COM_FLAG 0
#define RT_COM_CNT 3
```

monitor.c *(Monitor program)*

```
/*
 *  OBS Master Clock Syncronization MONITOR
 *
*************************************************************************
 *
*
 *   This program is free software; you can redistribute it and/or modify
*
 *   it under the terms of the GNU General Public License as published by
*
 *   the Free Software Foundation; either version 2 of the License, or
*
 *   (at your option) any later version.
*
 *
*
*************************************************************************
 */

#define _GNU_SOURCE
#include <stdio.h>
#include <fcntl.h>
#include <string.h>
#include <stdlib.h>
#include <unistd.h>
#include <ncurses.h>
#include <termios.h>


char buf[256];
char *bufptr=buf;
WINDOW *gps_window;
WINDOW *error_window;
WINDOW *wave_window;
static int seconds = 0;
static int first_reset = 0;
int ctl;     // fd for command fifo
int sec;     // fd for seconds fifo
int gps;     // fd for gps


/*
 *  allow program to clean up after itself :)
 */
void exitnice(void)
{
  beep();
  sleep(3);  // sleep in case there were any errormessages
  endwin();
  exit(1);
```

```
}


/*
 *  on-screen oscilloscope ...
 */
void update_waveform() {
  static int x = 0;
  static int y = 1;

  /* generate the correct waveform .. */
  if (((seconds % 5) == 0) && !(seconds == 5) && !(seconds == 10)) {
    if (y==0) {
      y=1;
      mvwprintw( wave_window, y, x, "%s", "|" );
      x++;
    } else {
      mvwprintw( wave_window, y, x, "%s", "|" );
      x++;
      y=0;
    }
  }

  /* if a minute has passed, clear window */
  if ( seconds == 0 ) {
    x=0;
    y=1;
    wclear( wave_window );
  }

  /* draw trace */
  mvwprintw( wave_window, y, x, "%s", "_" );
  x++;  // increase x position

  /* refresh window */
  wrefresh( wave_window );
}


/*
 *  writes a message in the "messages" window
 */
void write_info(char *streng) {
  wprintw(error_window, "%s", streng);
  wrefresh(error_window);
}


/*
 *  "Parse" NMEA sentences
 *  NMEA_code     : NMEA sentence to look for. Returns NULL if not equal
 *  NMEA_sentence : the string to parse (will NOT be modified)
 *  NMEA_field_no : specifies which field in the sentence to return
 *  s             : if not NULL, the returned string will also be copied
into this location
 */
char *NMEA_parse(char *NMEA_code, char *NMEA_sentence, int NMEA_field_no,
char *s)
{
  int i;
  static char *NMEA;
  static char code[255];
```

```
  static char *codeptr = code;
  static char sent[255];
  static char *sentptr;

  /* make a copy of all strings, so we don't mess anything up */
  strcpy(code, (char *)NMEA_code);
  codeptr = code;
  strcpy(sent, (char *)NMEA_sentence);
  sentptr = sent;

  /* check to see if the sentence is the one we're looking for ..
   * return NULL if false. */
  NMEA = (char *)strsep(&sentptr, ",");
  if(strcmp(NMEA, codeptr) != 0)
    return(NULL);

  /* search the sentence for the correct fieldno..
   * if field doesn't exist, NULL will be returned */
  sentptr = (char *)strsep(&sentptr, "*");
  for(i = 1; i < (NMEA_field_no + 1); i++)
    NMEA = (char *)strsep(&sentptr, ",");

  // printf("NMEA: %s\n", NMEA);
  /* copy field into location of s if s is not NULL */
  if(s)
    strcpy((char *)s, (char *)NMEA);

  /* return */
  return(NMEA);
}


/*
 *  oppdater klokke og sekundteller
 */
void check_NMEA(char *nmeastr) {
  //  int gps_valid = 1;
  char *tid;

  /* output the GPRMC sentence */
  if ( NMEA_parse( "$GPRMC", buf, 1, NULL ) ) {
    /* get GPS time (first field in GPRMC) */
    tid = NMEA_parse("$GPRMC", buf, 1, NULL);
    wprintw(gps_window, "%s\n", nmeastr);     // output "raw" NMEA sentence
    mvprintw(2, 52, "%s", tid);               // output GPS time
    /* first time we round a minute, syncronize the seconds counter to the
clock */
    if(!first_reset) {
      if((*(tid+4) == '0') && (*(tid+5) == '0')) {
      write_info("00 seconds - sending reset signal to OBSMCS!\n");
      first_reset = 1;
      write(ctl, &first_reset, sizeof(first_reset));
      }
    }
    /* refresh window */
    wrefresh(gps_window);
    refresh();
  }
}


/*
```

```
 *  main program
 */
int main() {
  int msg=0, retval, n;
  char ch;
  fd_set rfds;
  struct timeval tv;
  static int send_seconds_flag=1;
  struct termios attr;

  /* initialize screen */
  initscr();

  /* open /dev/rtf2  (command fifo) */
  if((ctl = open("/dev/rtf2", O_WRONLY)) < 0) {
    printf("err: could not open /dev/rtf2.\n\fIs obsmcs.o loaded?\n\f");
    exitnice();
  }

  /* open /dev/rtf3  (seconds fifo) */
  if((sec = open("/dev/rtf3", O_RDONLY)) < 0) {
    printf("err: could not open /dev/rtf3.\nIs obsmcs.o loaded?\n");
    close(ctl);
    exitnice();
  }

  /* open /dev/ttyS0 (serial port 1) */
  if((gps = open("/dev/ttyS0", O_RDONLY)) < 0) {
    printf("err: could not open /dev/ttyS0\n");
    close(ctl);
    close(sec);
    exitnice();
  }

  /* configure /dev/ttyS0 */
  tcgetattr(gps, &attr);
  cfsetispeed(&attr, B4800);
  cfsetospeed(&attr, B4800);
  tcsetattr(gps, TCSANOW, &attr);

  /* write "send seconds" command to obsmcs.o */
  msg = 2;
  if(write(ctl, &msg, sizeof(msg)) < 0) {
    printf("err: could not write to /dev/rtf2\n");
    exitnice();
  }

  attron(A_BOLD);

  move(0, 0);
  printw("%s", "OBS Master Clock Syncronization Monitor");

  move(2, 0);
  printw("%s", "internal seconds counter:                GPS clock:");

  move(4, 0);
  printw("%s", "GPS transmitted sentences:");

  move(10, 0);
  printw("%s", "Messages:");

  move(17, 0);
```

```
  printw("%s", "Waveform:");

  attroff(A_BOLD);

  move(21, 0);
  printw("%s", "------------------------------------------------------------
--------------------\n");
  printw("%s", "commands:\t");
  printw("%s", "s = toggle \"send seconds\"\n");
  printw("%s", "\t\tr = reset internal second counter\n");
  printw("%s", "\t\tq = quit\n");
  move(22, 75);
  printw("%s", "TS03");
  refresh();

  gps_window = newwin(5, 80, 5, 0);
  scrollok(gps_window, TRUE);
  error_window = newwin(5, 80, 11, 0);
  scrollok(error_window, TRUE);
  wave_window = newwin(3, 80, 18, 0);
  scrollok(wave_window, FALSE);
  mvwprintw(gps_window, 0, 0, "%s", "-- none yet --\n");
  mvwprintw(error_window, 0, 0, "%s", "waiting for GPS clock seconds to
become 00 ...\n");
  wrefresh(gps_window);
  wrefresh(error_window);

  /*
   *  main loop
   */
  while(1) {
    FD_ZERO(&rfds);
    FD_SET(sec, &rfds);
    FD_SET(0, &rfds);
    FD_SET(gps, &rfds);
    tv.tv_sec = 1;
    tv.tv_usec = 0;

    /* check for data from stdin, gps and obsmcs
     * timeout after one second */
    retval = select(FD_SETSIZE, &rfds, NULL, NULL, &tv);
    if(retval > 0) {  // did we get some data??

      if(FD_ISSET(sec, &rfds)) {       // got some new seconds
      /* read second counter from obsmcs */
      n = read(sec, &msg, sizeof(msg));
      if(n < 0) {
        printw("%s", "err: could not read from /dev/rtf3\n");
        close(ctl);
        close(sec);
        exit(1);
      }
      /* print second counter to screen and update waveform */
      move(2, 25);
      seconds = msg;
      printw("%02d", seconds);
      refresh();
      update_waveform();
      }

      if(FD_ISSET(0, &rfds)) {        // user pressed some keys...
      n = read(0, &ch, sizeof(ch));
```

```
      switch(ch) {
      case 'q':
        /* write "stop sending seconds" command */
        msg = 3;
        write(ctl, &msg, sizeof(msg));
        close(ctl);
        close(sec);
        exitnice();
        break;
      case 's':
        /* toggle "send seconds" */
        send_seconds_flag ^= 1;
        if(send_seconds_flag)
          msg = 2;
        else
          msg = 3;
        write(ctl, &msg, sizeof(msg));
        break;
      case 'r':
        /* reset internal second counter */
        write_info("waiting for GPS clock seconds to become 00 ...\n");
        first_reset = 0;
        break;
      default:
        break;
      }
      }

      if(FD_ISSET(gps, &rfds)) {   // gps data ready
      n = read(gps, &ch, sizeof(ch));
      *bufptr = ch;
      bufptr++;
      if(ch == '\n') {
        *(--bufptr) = '\0';
        bufptr = buf;
        check_NMEA(bufptr);
        memset(&buf, '\0', sizeof(buf));
        bufptr = buf;
      }
      }
    }
  }
}
```

Makefile:

```
# Makefile for obsmcs  --  version 0.1

# This should work.. if not, set the correct path to
# rtl.mk, or copy the file to this directory
include /usr/rtlinux/rtl.mk


all: obsmcs.o monitor

gunstamp.o:
	$(CC) $(INCLUDE) $(CFLAGS) -c gunstamp.c -o gunstamp.o

monitor: monitor.c
	$(CC) $(INCLUDE) $(USER_CFLAGS) -lncurses -Wall -O2 -o monitor
monitor.c

clean:
	rm -f *.o
	rm -f monitor

garbage_go_away:
	rm -f *~

rebuild: clean all
```

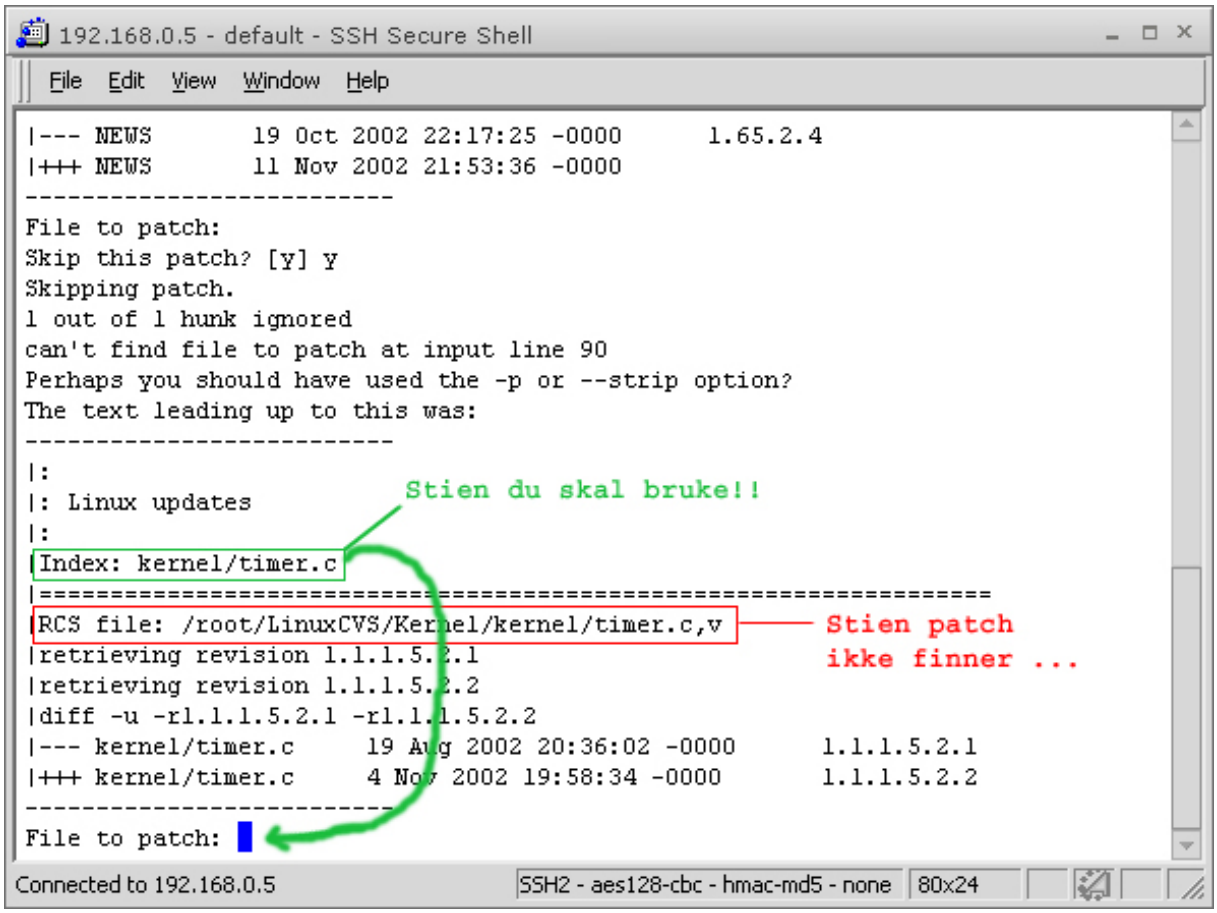
3b NTP (Network Time Protocol) – Installasjon, konfigurasjon og test

Installasjon og testing – steg for steg

Server:

- Det er en forutsetning at en versjon av Linux er installert på forhånd (se avsnitt 1.1.2).
- Følgende programvare må lastes ned (jeg plasserte alle filene i `/home/trond`):
  - Linux kernel-2.4.19 (www.kernel.org/pub/linux/kernel/v2.4/kernel-2.4.19.tar.bz2)
  - PPSkit-2.0.2 (http://www.kernel.org/pub/linux/daemons/ntp/pps/PPSkit-2.0.2.tar.bz2)
  - PPSkit-2.0.2-fix2 (http://www.kernel.org/pub/linux/daemons/ntp/pps/PPSkit-2.0.2-fix2.tar.bz2)
  - Ntp-4.1.1 (http://www.ntp.org/ntp_spool/ntp4/ntp-4.1.1.tar.gz)
- Den nye Linux kernelen må pakkes ut, patches med PPSkit, konfigureres og kompileres. Fremgangsmåten er som følger:

```
01: # su -
    Password:
02: # cd /usr/src
03: # tar -xjvf /home/trond/PPSkit-2.0.2.tar.bz2
04: # tar -xjvf /home/trond/PPSkit-2.0.2-fix2.tar.bz2
05: # cd PPSkit-2.0.2
06: # patch < ../PPSkit-2.0.2-fix2

(obs! Denne fix'en inneholder patch for bade PPSkit og Linux
kernelen. Etter å ha patchet tre filer (som hører til PPSkit), vil
programmet spørre deg hvor filen kernel/timer.c er plassert. Avbryt
med <CTRL-C> når det spørsmålet kommer)

07: # cd ..
08: # tar -xjvf /home/trond/linux-2.4.19.tar.bz2
09: # mv linux-2.4.19 linux-2.4.19-NANO
10: # cd linux-2.4.19-NANO
11: # patch -p1 < ../PPSkit-2.0.2/patch-2.4.19
12: # patch < ../PPSkit-2.0.2-fix2

(obs! Samme som tidligere - filen inneholder patch for både PPSkit og
Linux kernelen. Filene som patches før kernel/timer.c hopper du over
ved å trykke <ENTER> og <Y>. Deretter må stien til alle filene som
skal patches spesifiseres manuelt. For hver fil vil det stå en sti
etter "RCS file:", som er der den venter å finne filen. Men det står
også en sti etter "Index:", og det er den korrekte. Skriv enkelt og
greit inn denne, eller merk den med musen, og trykk på den midterste
museknappen (eller begge to samtidig om du bare har 2 knapper). Se
eksempel på neste side for bedre beskrivelse.)
```




```
13: # cp /boot/config-2.4.18-14 .config
14: # make oldconfig

(Du vil få fem spørsmål; fire av dem er relatert til NTP, og de må du
svare ja (<Y>) på. Det siste (eller egentlig det fjerde..) kan du
bare trykke <ENTER> på.)

15: # make menuconfig    (eventuelt config eller xconfig)

(Alle SCSI og ATM drivere må velges bort. Den generelle SCSI driveren
skal virke, men det fungerte ikke på min maskin.)

16: # make dep; make bzImage; make modules; make modules_install;
make install;
17: # pico /etc/lilo.conf

(Sjekk at den nye kernelen har blitt lagt til i listen. Hvis ikke,
skriv inn følgende linjer på slutten av filen: )

image=/boot/vmlinuz-2.4.19-NANO
        label=nano
        initrd=/boot/initrd-2.4.19-NANO.img
        read-only
        append="root=LABEL=/1"

18: # /sbin/lilo
19: # reboot

( Velg nano kernelen ved oppstart. )
```

Gjennomgang linje for linje:

Du må være superuser for å kunne installere en ny kernel. Du får root-privilegier med kommandoen på linje 1. I den nyeste versjonen av PPSkit (2.0.2) er det et par bugs som kan rettes med å patche den med PPSkit-2.0.2-fix2. PPSkit pakkes ut og patches på linjene 2-6. Deretter må Linux kernelen pakkes ut og patches, både med PPSkit-2.0.2 og PPSkit-2.0.2-fix2. Dette skjer på linje 7-12. For å lette konfigureringen av den nye kernelen, "stjeler" vi konfigurasjonen som RedHat bruker for 2.4.18 kernelen (linje 13-14). Litt manuell konfigurering må du allikevel gjøre selv, blant annet må du deaktivere driverne for SCSI og ATM (linje 15). Nå er det tid for å kompilere kernelen, og etter vellykket kompilering må den legges til i konfigurasjonen til bootloaderen (lilo) før du restarter maskinen (linje 16-19).

Når maskinen har bootet med den nye kernelen, kan en fortsette med å installere NTP:

```
20: # ln -s /usr/src/linux-2.4.19-NANO/include/linux/timepps.h
/usr/include/sys/timepps.h
21: # ln -s /usr/src/linux-2.4.19-NANO/include/linux/timex.h
/usr/include/sys/timex.h
22: # cd /usr/src
23: # tar -xzvf /home/trond/ntp-4.1.1.tar.gz
24: # cd ntp-4.1.1
25: # ./configure --prefix=/usr --bindir=/usr/sbin --enable-ATOM
26: # make
27: # make install
```

Etter en forhåpentligvis vellykket installering, må NTP konfigureres. Konfigurasjonsfilen heter `/etc/ntp.conf` . Her kan en sette alle innstillinger, som hvilke referanseklokker som brukes (hvis det er en stratum 1 server) eller hvilke servere som brukes (for alle andre stratum). Du kan konfigurere hvilke maskiner eller hvilke nettverk som skal kunne bruke din server som synkroniseringskilde, hvilken metode som skal brukes for distribusjon av tid, hvor mye som skal logges, autentisering, osv. Full dokumentasjon på dette finnes på følgende adresse: http://www.eecis.udel.edu/~ntp/documentation.html Det er ikke brukt noen autentisering i denne oppgaven. For et lokalt nettverk bak en firewall, der alle skal kunne ha tilgang til NTP serveren, er det heller ikke nødvendig. Autentisering blir brukt kun når du vil begrense tilgangen til serveren på store nettverk (for eksempel over internett).

Konfigurasjonsfilen jeg har brukt er gjengitt på neste side. Denne er laget for å ta imot NMEA setninger fra GPS tilkoblet serieport 1, samt PPS pulser tilkoblet pinne 1 på samme serieport. *NB! PPS pulsene har TTL nivå, som vil si at de må konverteres til RS232-nivå først! I denne oppgaven har jeg brukt en interface-boks som jeg laget tidligere, den har også innebygget strømforsyning slik at egen strømforsyning til GPS-en er unødvendig. Koblingsskjema for hvordan GPS-en kobles til interface-boksen, og kabel mellom interaface-boksen og PC-en finner du på side 47. Jeg har også lagt ved tegning for hvordan du kan lage din egen levelconverter på side 48.*

Konfigurasjonsfil for NTP server (/etc/ntp.conf):

```
### Miscellaneous section ###
# logconfig [+|-
|=][{sync|sys|peer|clock}{{,all}{info|statistics|events|status}}]...
logconfig =syncevents +peerevents +sysevents +allclock

pps /dev/refclock-1 assert hardpps  # PPS device

### Configuration section ###
server 127.127.20.0 mode 1 prefer   # Generic NMEA
fudge 127.127.20.0 time1 0.0042     # relative to PPS for my hardware

server 127.127.22.0                 # ATOM(PPS)
fudge 127.127.22.1 flag3 1          # enable PPS API

#broadcast 224.0.1.1 key 2 ttl 2    # broadcast the time

enable pps                    # needed for `pps_sync'

# enable auth monitor stats
enable monitor stats
driftfile /etc/ntp.drift      # remember the drift of the local clock

### Statistics section ###
statistics loopstats clockstats
statsdir /tmp/xntpd/
filegen peerstats file peers type day link enable
filegen loopstats file loops type day link enable
filegen clockstats file clocks type day link enable

### Authentication section ###
#keys /etc/ntp.keys
#trustedkey 1 2 15
#requestkey 15
#controlkey 15
#authdelay 0.000019     # DES, Pentium @ 100MHz
#authdelay 0.000030     # MD5, Pentium @ 100MHz

## Access control section ###
#restrict 0.0.0.0 mask 0.0.0.0 ignore
restrict 127.127.0.0 mask 255.255.0.0 noquery # allow referenc clocks
#restrict 127.0.0.1 nopeer    # don't synchronize on yourself
# allow clients from our local network
restrict 192.168.0.0 mask 255.255.255.0 notrust nomodify notrap
```

Det eneste som gjenstår nå, før en kan starte NTP, er å opprette to symbolske linker, lage et oppstartsskript for NTP og å koble til GPS-en. RedHat kommer med et ferdig skript for å starte NTP, men det er beregnet på klienter, og virker ikke sammen med en NTP server.

```
# ln -s /dev/ttyS0 /dev/gps0
# ln -s /dev/ttyS0 /dev/pps0
```

Disse linkene trengs på grunn av NTP klokkedriverne. De venter å finne en GPS tilkoblet `/dev/gps0`, og en PPS-kilde tilkoblet `/dev/pps0`. (Her linkes begge til serieport 1). Koble til GPS-en i henhold til skjema på side 47.

Oppstartsskriptet heter `/etc/rc.d/init.d/ntpd` og ser slik ut:

```
#!/bin/bash
#
# description: NTPD Network Time Protocol Daemon
#
# script author: Trond Solberg  <trond.solberg@broadpark.no>
#
# chkconfig: 2345 95 05

# source function library
. /etc/init.d/functions

PATH=$PATH:/usr/sbin
SERVER=ntpd
prog="$SERVER"

[ -f /usr/sbin/$SERVER ] || exit 0

start() {
   echo -n $"Starting $prog: "
   daemon $prog
   echo
}

stop() {
   echo -n $"Stopping $prog:"
   killproc $SERVER
   echo
}

case "$1" in
  start)
   start
   ;;
  stop)
   stop
   ;;
  status)
   status ntpd
   ;;
  restart|reload)
   stop
   start
   ;;
  *)
   echo $"Usage: $0 {start|stop|restart|status}"
```

```
   exit 1
esac

exit 0

# script done
```

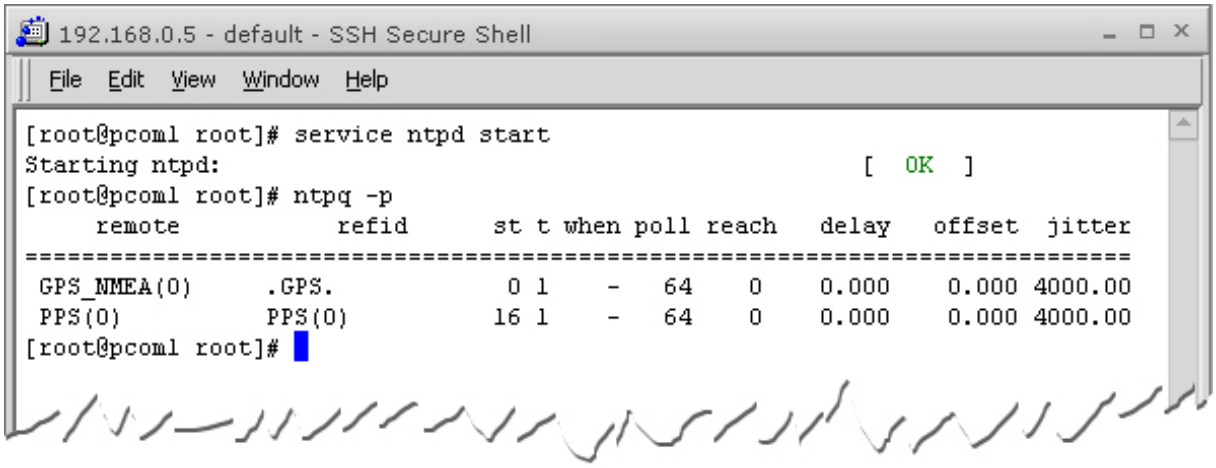
Du kan nå bruke kommandoene `service ntpd start` og `service ntpd stop` for henholdsvis å starte og stoppe NTP. Du kan også la NTP bli automatisk startet når du slår på maskinen. For å gjøre det, må en først kjøre `chkconfig --add ntpd` for å generere alle nødvendige skript, før en aktiverer automatisk oppstart av NTP med kommandoen `chkconfig --level 2345 ntpd on` . Senere kan NTP deaktiveres igjen med kommandoen `chkconfig --level 2345 ntpd off`

Test av server:

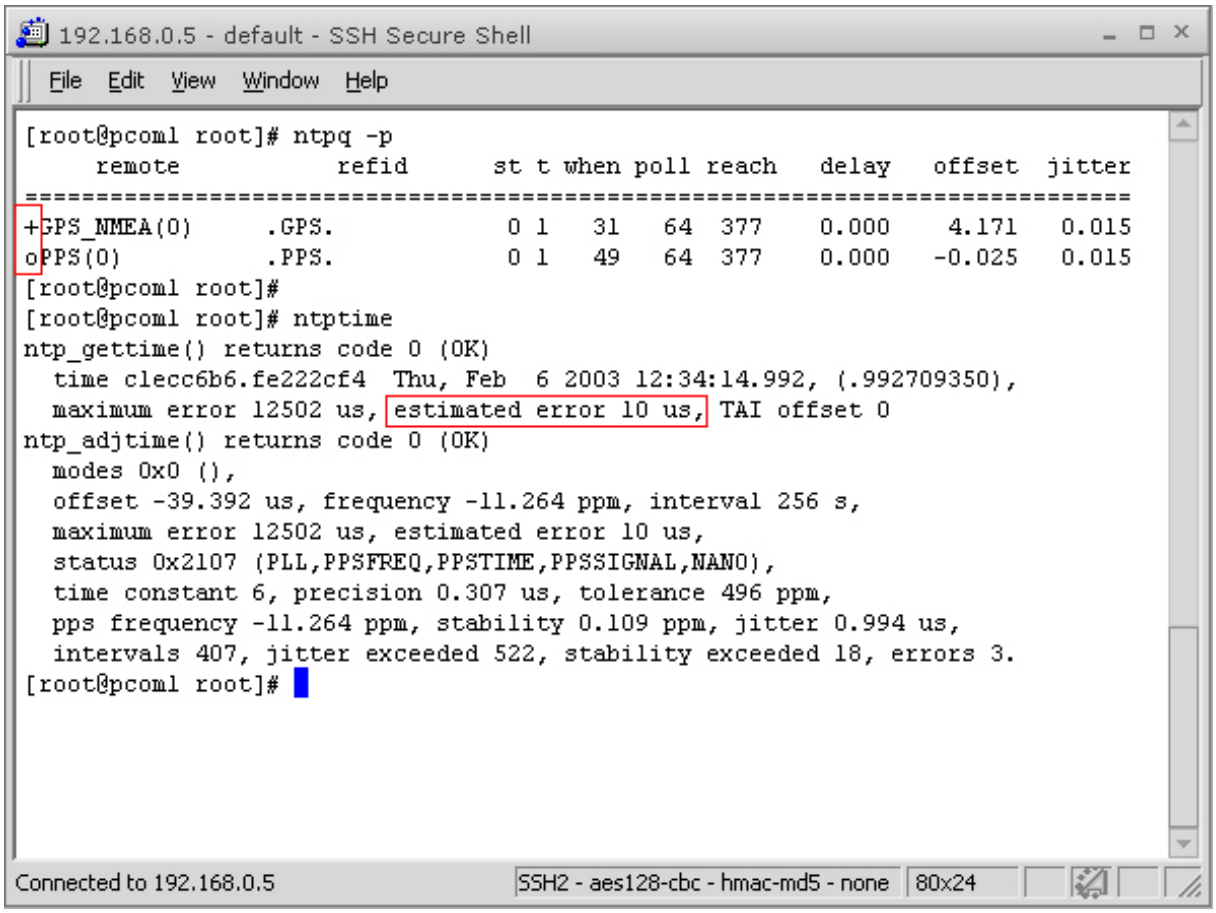
Start NTP (må gjøres som root):

service ntpd start

NTP har et diagnoseverktøy som hetter ntpq. Dette kan du kjøre for å få diverse statusinformasjon, men vi skal kun bruke det til å se hvilke servere/klokker som er konfigurert, og om NTP har kontakt med disse:



I starten vil statusen se ut slik som på bildet over – to klokker er konfigurert, men ingen av dem er i bruk. Det står ’0’ på ”reach” for begge klokkene, som betyr at NTP ikke har kontakt med dem. ”when” er hvor lang tid som er gått siden siste oppdatering, ”poll” er intervallet mellom hver oppdatering, ”st” er hvilket stratum klokken befinner seg i, og ”offset” er hvor langt unna PC klokken er fra referanseklokkene. Det tar litt tid fra du starter NTP til klokkene blir aktivert (de må gjennom en hel del tester først), men etter en halvtimes tid burde det se ut slik som på bildet på neste side.



Etter å ha kjørt `ntpq -p` ser vi at begge klokkene er i bruk, PPS pulsen blir brukt til å synkronisere mot (markert med 'o' foran PPS), og den får tiden fra NMEA setningene (markert med '+' foran GPS_NMEA). "delay" viser 0.000 som viser at det er ingen forsinkelse på linjen (logisk i og med at klokken er koblet direkte til PC-en).

Har og kjørt kommandoen `ntptime`, og det mest interessante vi får ut av det, er feltet "estimated error 10us" som forteller at anslått nøyaktighet er 10us. Meget bra!!

Installasjon på klient:

Samme prosedyre som ved installering av server, se side 40 for instruksjoner (men hopp over linje 20-21).

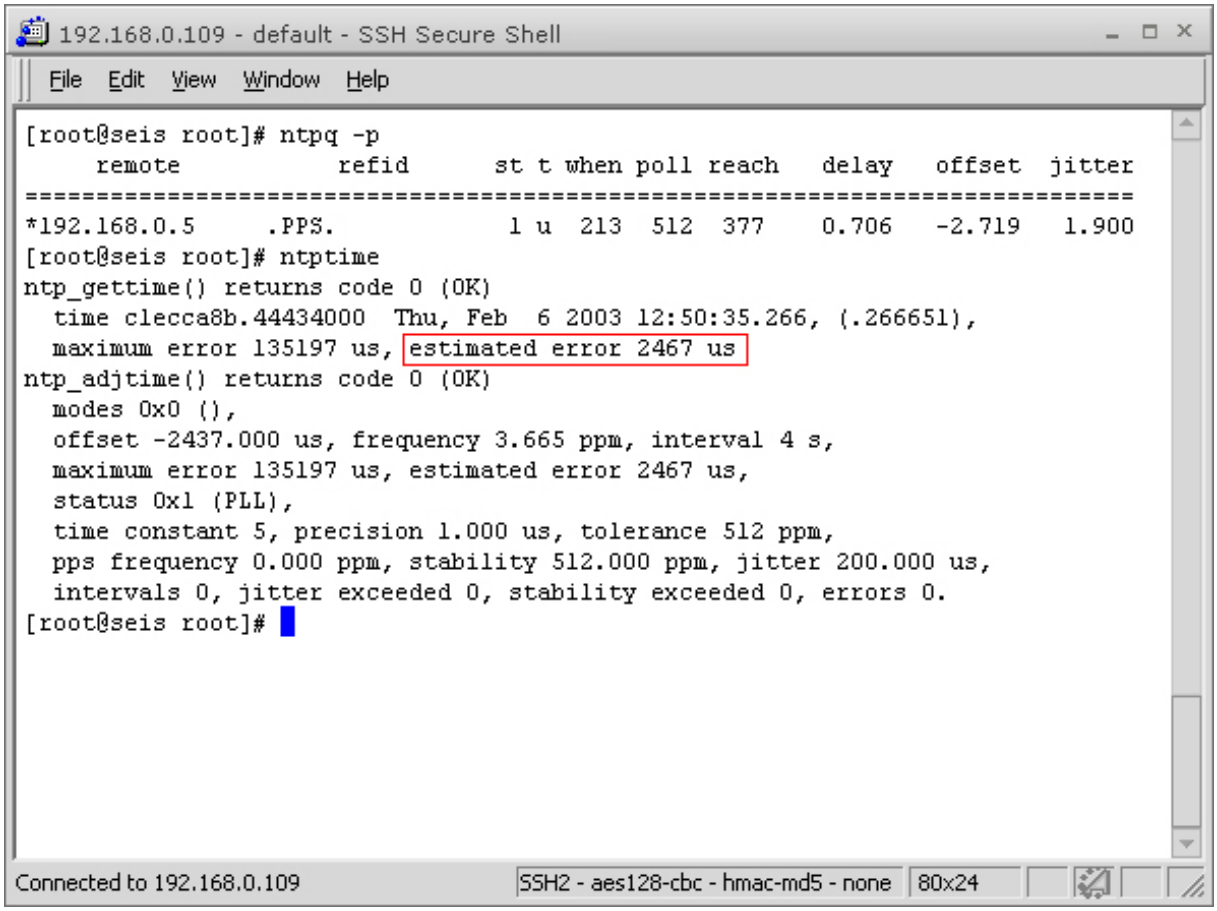
Etter installering må klienten konfigureres til å hente tidsdata fra min server. Konfigurasjonsfilen jeg har brukt er gjengitt på neste side.

```
# Prohibit general access to this service.
restrict default ignore

# Permit all access over the loopback interface.  This could
# be tightened as well, but to do so would effect some of
# the administrative functions.
restrict 127.0.0.1

# --- OUR TIMESERVERS -----
# Permit time synchronization with our time source, but do not
# permit the source to query or modify the service on this system.
restrict 192.168.0.5 mask 255.255.255.255 nomodify notrap noquery
server 192.168.0.5

#
# Drift file.  Put this in a directory which the daemon can write to.
# No symbolic links allowed, either, since the daemon updates the file
# by creating a temporary in the same directory and then rename()'ing
# it to the file.
#
driftfile /etc/ntp/drift
broadcastdelay    0.008

#
# Authentication delay.  If you use, or plan to use someday, the
# authentication facility you should make the programs in the auth_stuff
# directory and figure out what this number should be on your machine.
#
#authenticate yes

#
# Keys file.  If you want to diddle your server at run time, make a
# keys file (mode 600 for sure) and define the key number to be
# used for making requests.
#
# PLEASE DO NOT USE THE DEFAULT VALUES HERE. Pick your own, or remote
# systems might be able to reset your clock at will. Note also that
# ntpd is started with a -A flag, disabling authentication, that
# will have to be removed as well.
#
keys        /etc/ntp/keys
```

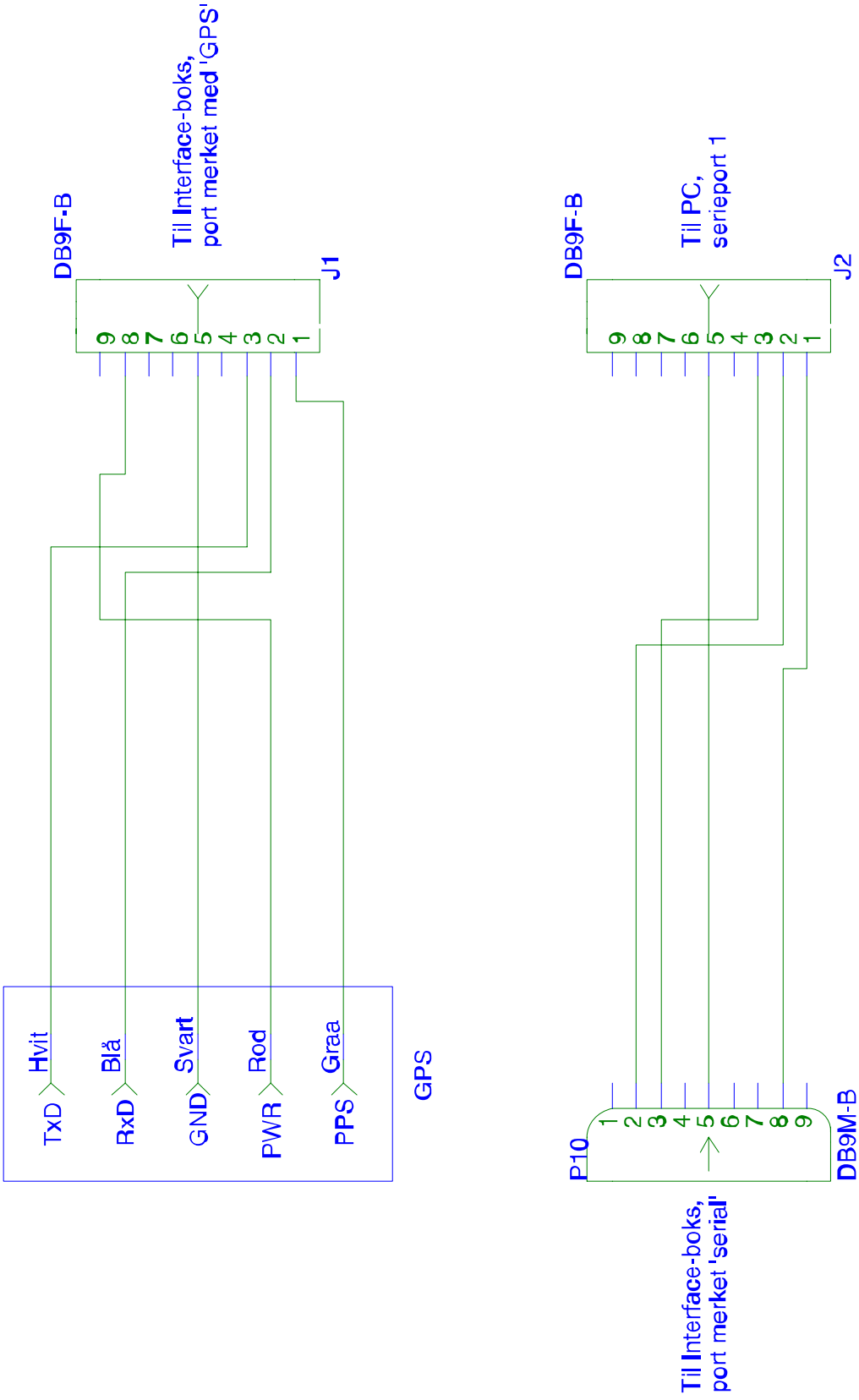
Test av klient:

Klienten startes på samme måte som på serveren, med kommandoen `service ntpd start`. (Merk at her trenger en ikke å gjøre noen forandringer i oppstartsskriptet slik en måtte på serveren). Ønsker du at NTP skal startes automatisk når du slår på maskinen, gjøres det med kommandoen `chkconfig --level 2345 ntpd on`

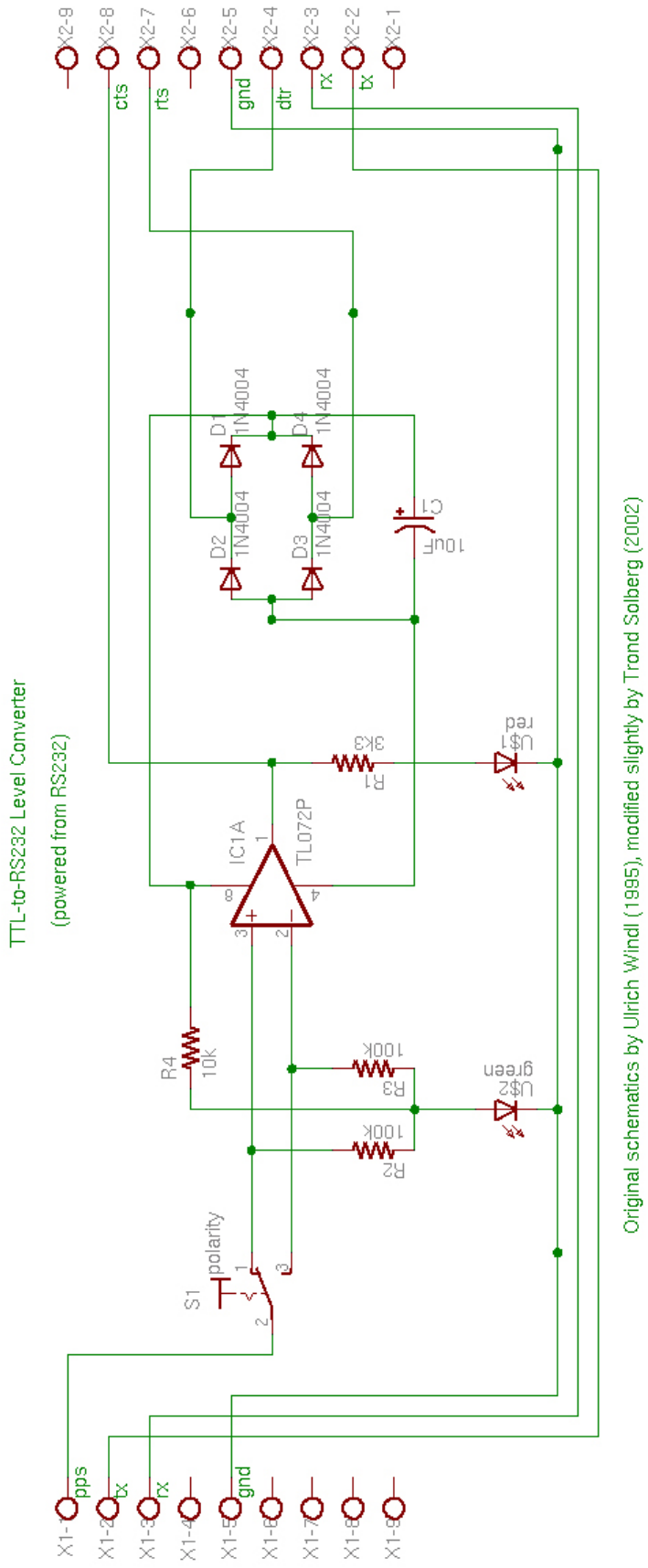
La klienten kjøre en stund, før du prøver kommandoene `ntpq -p` og `ntptime`.



Som vi ser har klienten her fått en nøyaktighet på 2467us, eller ca 2.5ms noe som ikke er så aller verst.



Figur 11 - Koblingsskjema for tilkobling av GPS til PC via interface-boks



Figur 12 - Koblingsskjema for TTL->RS232 levelconverter

Oppgave 4

4.1 Bus-systemer

4.1.1 PCI bus

Figur 13 - Et PCI kort (nettverkskort)

PCI står for Peripheral Component Interconnect og er en bus som blir brukt for kommunikasjon mellon prosessoren i en PC og diverse utvidelseskort, som f.eks. nettverkskort, interne modem, lydkort, skjermkort, mm, samt for direkte kommunikasjon mellom kortene. Opptil 5 enheter kan kobles til PCI bussen i en PC.

Opprinnelig hadde PCI bussen en klokkefrekvens på 33MHz og en 32 bits data/adressebuss. Senere har standarden blitt revidert, og PCI 2 fikk en klokkefrekvens på 66MHz og en 64 bits data/adressebuss. Den siste standarden som har komt, PCI-X, har en klokkefrekvens på 133MHz og 64 bits data/adressebuss, noe som setter den i stand til å overføre data med en hastighet på over 1GB per sekund.

Busbredde	Klokkefrekvens	MB/s
32 bits	33 MHz	132 MB/s
64 bits	33 MHz	264 MB/s
64 bits	66 MHz	512 MB/s
64 bits	133 MHz	1067 MB/s

4.1.2 AGP bus

Figur 14 - AGP skjermkort

AGP står for Accelerated Graphics Port og er en modifikasjon av PCI bussen laget for én ting: grafikk med høy ytelse. Selv om den ofte kalles for en bus, er det egentlig ikke det. Det er kun mulig å koble til én enhet, så det blir mer som en direkte forbindelse mellom prosessoren i PC-en og grafikkortet.

AGP har to fordeler framfor PCI; bedre ytelse og direkte tilgang til systemminnet.

Per i dag er det tre standarder for AGP: AGP 1.0, AGP 2.0 og AGP Pro. Alle er basert på en 32 bits databuss med en klokkefrekvens på 66MHz, men mens AGP 1.0 bare kan sende data én gang per klokkesyklus, kan AGP 2.0 sende data to ganger, og AGP Pro fire ganger. Det er også en ny standard under utarbeidelse, som heter AGP8x.

AGP versjon	MB/s
1.0	266 MB/s
2.0	533 MB/s
Pro	1066 MB/s
8x	2133 MB/s

4.1.3 IDE bus

Figur 15 - IDE harddisk

IDE, som står for Integrated Drive Electronics, er et standard interface for å koble lagringsenheter som harddisker, diskettstasjoner eller CD-spillere til en PC. IDE er egentlig ikke det riktige navnet for denne standarden, opprinnelig het den ATA (AT Attachment).

IDE ble opprinnelig utviklet for å ha en standard måte å koble til harddisker på. Tidligere var harddisk og kontroller separert, og ideen bak IDE er at kontroller og harddisk skal være sammen i én enhet.

Hovedkort har vanligvis to IDE interface, hvor hvert interface støtter opp til to enheter. For å kunne ha to enheter på samme kabel, bruker IDE en spesiell konfigurasjon kalt master og slave. Har du bare én enhet installert, er den alltid master. Setter du i en til, må den konfigureres som slave. Kun én enhet kan sende data over IDE interfacet om gangen, så for å unngå kollisjon, må slaven spørre masteren om å få sende data. Hvis masteren bruker bussen da, blir slaven bedt om å vente til masteren er ferdig med å overføre data.

IDE standarden har gjennomgått mange revisjoner opp gjennom tidene, og dataoverføringshastigheten har økt fra de opprinnelige 4.16 MB/s til 150 MB/s.

Oppgave 5

Oppbygging av og virkemåte til en analog s/h kopimaskin

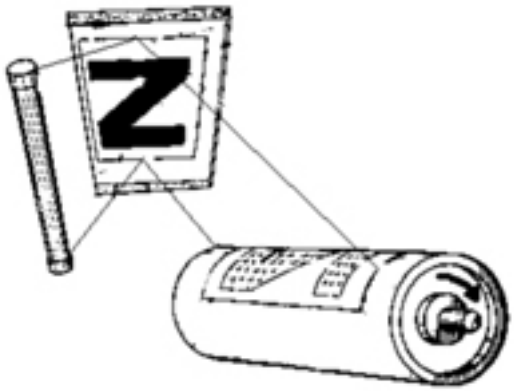
Kopiprosessen er delt opp i seks faser:

- Hovedcorona
- Billedeksponering
- Fremkalling
- Overføring
- Fiksering
- Trommelrensing

Hovedcorona

Hovedcoronaen er første fast i kopieringen. Den sørger for å lade opp overflaten av den fotofølsomme trommelen med enten + eller – ladninger. I hovedcoronaen ligger det en gullbelagt wolframtråd med en diameter på 0,06 diameter parallelt med trommelen. Denne kalles coronatråd.

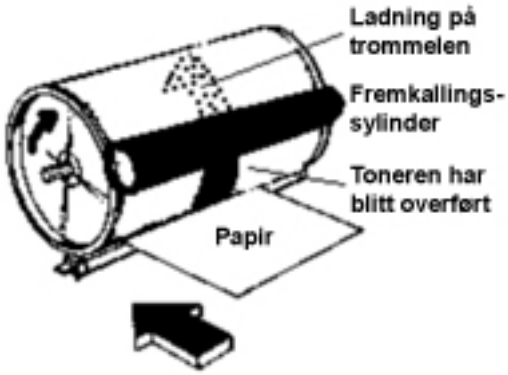
Billedeksponering



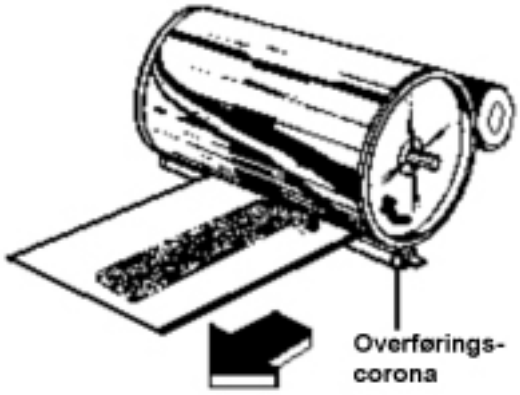
Originalen blir belyst nedenifra og opp av en lampe på undersiden av glassplaten. De hvite områdene på originalen vil reflektere dette lyset, og via en del optikk (speil og linser) blir lyset dirigert ned på den fotofølsomme trommelen. De områdene på trommelen som blir belyst, mister ladningen sin.

Fremkalling

Her blir et pulver kalt toner overført til trommelen. Toneren må ha motsatt ladning av trommelen for å kunne bli ”dradd” over. Har trommelen negativ ladning, må toneren ha positiv ladning. Prosessen foregår slik at det blir tilført toner til en fremkallingssylinder som roterer inntil den fotofølsomme trommelen. Det oppstår en positiv ladning i fremkallingssylinderen på grunn av friksjonen mellom denne og trommelen, og toneren blir ladet. De områdene på den fotofølsomme trommelen som har negativ ladning, vil da trekke til seg toneren.

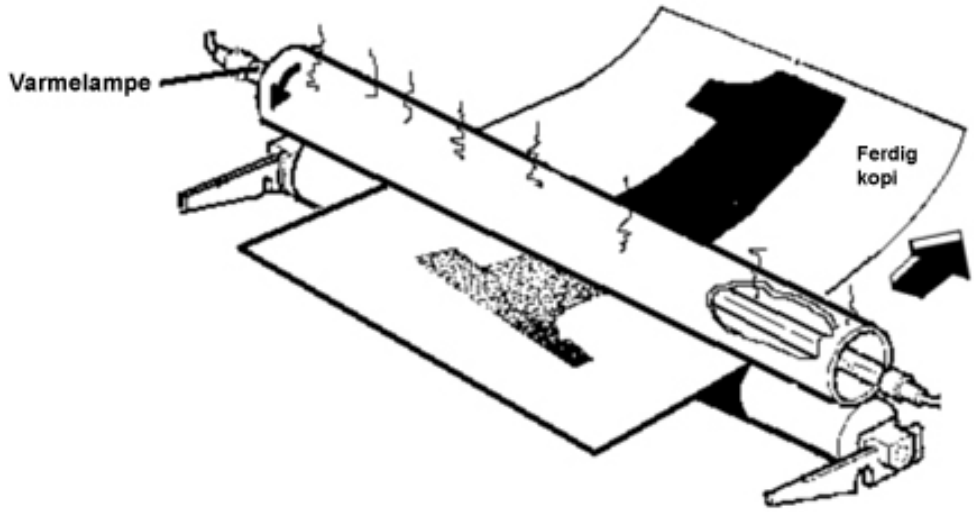


Overføring



Papiret blir negativt ladet av en coronatråd på baksiden av papiret, kalt overføringscorona. Dette gjør at toneren på trommelen blir overført til kopipapiret. For at dette skal fungere skikkelig er det viktig at papiret har nær kontakt med trommelen.

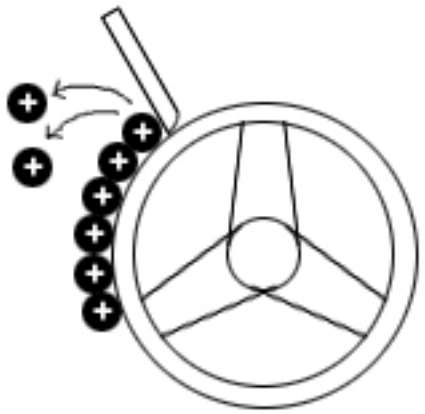
Fiksering



Selv om toneren nå er overført til papiret, er den fremdeles helt løs. Får å få den festet til papiret, blir det ført gjennom to valser. Den øverste er varm og ”smelter” toneren fast i papiret, men den nederste valsen er en pressvalse som presser papiret mot den varme valsen. Valsene er innsmurt med et tynt lag silikonolje for at toneren ikke skal henge fast i dem.

Trommelrensing

Denne prosessen fjerner tonerrestene som henger igjen på trommelen etter overføringsprosessen. Et gummiblad berører trommelen og renser overflaten.



Hjelpeprosesser

I tillegg til disse grunnleggende prosessene, finnes det flere spesialprosesser som brukes for å øke kopikvaliteten og sikkerheten i prosessen. Dette er prosesser som:

Forhåndseksponering
Etter at trommelen er renset vil det fremdeles være en restladning igjen. Under forhåndseksponering vil en lampe belyse hele trommelens overflate med et jevnt lys og fjerne restladningen for å hindre ujevn kvalitet på neste kopi.

Valseelektrode
En metallvalse er montert parallelt med trommelen og tilføres en spenning med samme polaritet som toneren. Dette tvinger eventuell løs toner tilbake til trommelen for å hindre at tonerstøv spres i maskinen.

Forladning
Denne prosessen gjør det enklere å overføre toner fra trommelen til kopipapiret. Siden papiret har en tendens til å klebe seg til trommelen på grunn av statisk elektrisitet, vil denne prosessen samtidig motvirke det slik at papiret lett separeres fra trommelen. Forladningscoronaen tilføres en spenning for å senke trommelens overflateladning. Det gjør at tiltrekningen mellom trommelen og toneren minskes, og toneroverføringen til kopipapiret blir enklere.

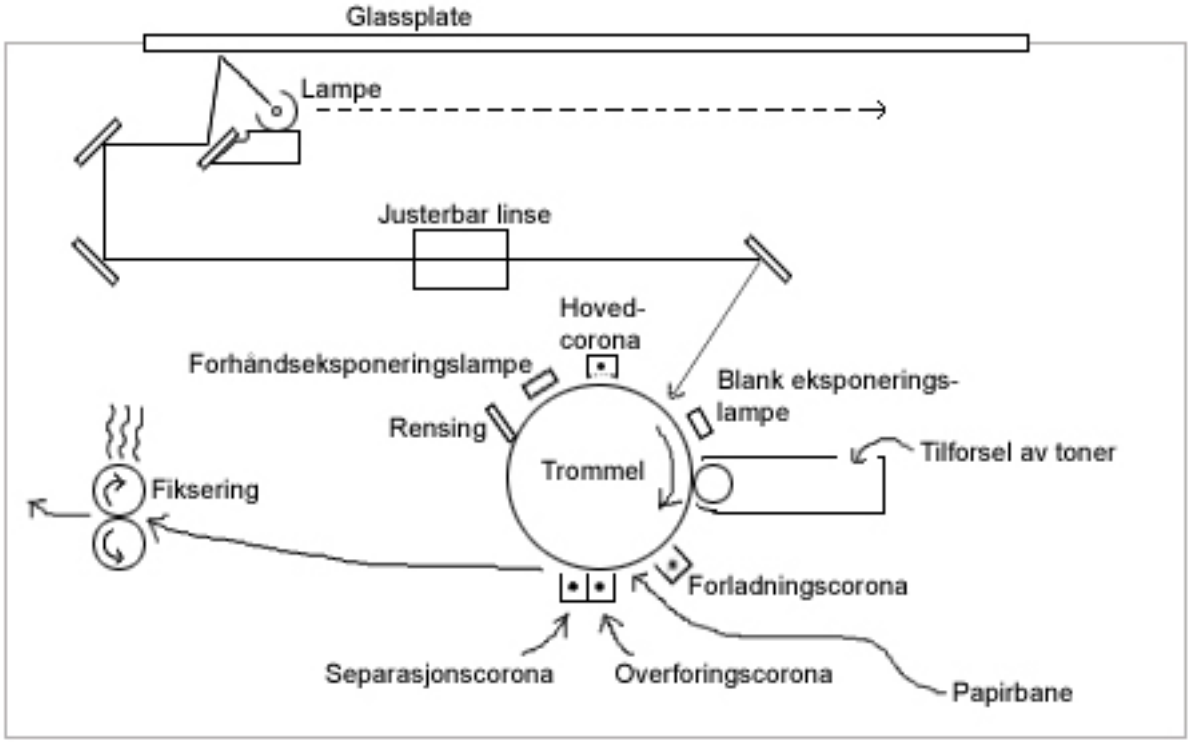
Separasjon
Etter overføringen separeres papiret fra trommelen. Det kan gjøres på tre forskjellige måter:

- Kurveseparasjon
  Kun papirets stivhet blir benyttet til separasjon. Ruller du et ark sammen og slipper det, vil det rette seg ut igjen til et flatt ark igjen. Jo mindre diameter vi ruller papiret sammen til, desto sterkere vil kraften til å rette seg ut igjen være. Derfor brukes denne metoden kun på maskiner som har liten trommeldiameter, og høy hastighet.

- Elektrostatisk separasjon
  Her blir en separasjonscorona plassert under trommelen. Denne tilføres en spenning slik at det dannes en utladning under papiret. Dermed blir ladningen i papiret opphevet, og tiltrekningen mellom trommel og papir blir så liten at papiret løsner ved hjelp av sin egen vekt.

- Separasjonsbelte
  Et separasjonsbelte blir plassert helt i ytterkanten på trommelen slik at det kommer mellom trommelen og kopipapiret. Siden separasjonsbeltet berører kopien kan det ikke kopieres på denne delen av arket.

Blank eksponering
Hvis en del av trommelen ikke skal brukes – for eksempel ved forminsking eller kopiering av mindre format – blir den ubrukte delen av trommelen belyst slik at det ikke legger seg toner her. Dette skjer ved hjelp av en rad med lys (LEDs) og foregår samtidig med billedeksponeringen.



Figur 16 - Oversikt over hele den analoge kopiprosessen