1 Forord

Dette er sluttrapporten for TDT4295 Datamaskiner Prosjekt ved IDI, NTNU 2003. Prosjektet utgjør 15 studiepoeng i høstsemesteret i 4. årskurs og varte fra 11. august til 28. november.

Hensikten med prosjektet er å gi studentene praktisk og teoretisk erfaring med hardwareutvikling slik det foregår i større prosjekter. Prosjektet tar for seg hele designsyklusen fra kravspesifikasjon til ferdig prototype.

Vi vil gjerne takke våre faglærere, Pauline Haddow og Gunnar Tufte for god oppfølging. I tillegg vil vi takke Morten Hartmann og Frode Eskelund for hjelpen vi fikk med å finne feilen på PCBen vår. Karstein Kristiansen har også hjulpet oss mye når vi har hatt problemer. Vi vil også takke Atmel for utlån av utstyr og for å ha gitt oss mikrokontrollere.

> "Those parts of the system that you can hit with a hammer (not advised) are called hardware; those program instructions that you can only curse at are called software."
>
> – Levitating Trains and Kamikaze Genes: Technological Literacy for the 1990's.

2 Innledning

> "To undertake a project, as the word's derivation indicates, means to cast an idea out ahead of oneself so that it gains autonomy and is fulfilled not only by the efforts of its originator but, indeed, independently of him as well."
>
> – Czeslaw Milosz

Prosjektet er kalt Værstasjon 2003 og har blitt utført i faget TDT4295 Datamaskiner Prosjekt. Oppgaven vi fikk tildelt gikk ut på å designe, implementere og teste et system bestående av hardware og software. Systemet skulle realisere funksjonalitet for måling, lagring, bearbeiding og presentasjon av data knyttet til ulike aspekter ved klimaet. Rammebetingelsene vi fikk var ganske åpne, men det var et krav at vi benyttet FPGA og AVR i designet vårt. I tillegg skulle systemet bestå av to hoveddeler som etter hvert ble kalt sensorkortet og displaykortet. Dette kravet kom som en følge av at vi var relativt mange som tok faget i år. Vi fikk dermed to mindre grupper som kunne jobbe mer individuelt. Disse to kortene skulle også kunne kommunisere med hverandre over USB. Etter som prosjeket skred frem viste det seg imidlertid at samarbeidet mellom gruppene ble tettere og tettere, og vi har derfor i mange sammenhenger fungert som en stor gruppe.

Oppgaven har blitt utført av en gruppe på 11 studenter.

Sensorgruppe	**Displaygruppe**
Tord Andreas Fredriksen	Guri Kristine Birkeland
Stian Dalene Gjedrem	Peder Dagestad Rand
Joacim Thomassen	Morten Hauseggen
Kristian Skogstrøm	Lars Eirik Eilertsen
Marius Grannæs	Olav Gulling
Kjetil Aamodt	

Innhold

Tabeller

Figurer

3 Sammendrag

Denne rapporten beskriver systemet som har blitt laget i faget TDT4295 Datamaskiner Prosjekt høsten 2003. Systemet er en værstasjon for innsamlig og visning av værdata.

Oppgaven som ble gitt gikk ut på at vi skulle lage en værstasjon. Prosjektet skulle gå gjennom alle fasene i et maskinvareutviklingsprosjekt. Fra kravspesifikasjon, kretsdesign, PCB [1] -produksjon, programmering og til slutt test og ferdigstilling av produktet. Vi hadde hele høstsemesteret på å gjennomføre oppgaven. Gruppen besto opprinnelig av 13 personer, men ettersom tiden gikk ble vi av ulike årsaker redusert til 11. Vi ble ved oppstart delt inn i to grupper som skulle arbeide hver for seg med hvert sitt delsystem.

Oppgaveteksten påla oss enkelte designvalg. Blant annet var det bestemt at systemet, for begge delsystemer, måtte benytte seg av en FPGA [2] og en AVR [3] . Budsjettet vårt skulle ikke overskride 15 000 NOK. Utenom dette budsjettet kom FPGA, AVR og PCB-produksjon. Alt i alt hadde vi rimelig stor valgfrihet og et budsjett som ga oss muligheter for å teste ulike løsninger og komponenter.

Systemet er delt opp i to enheter. Den ene enheten skal tilkobles sensorer, kalt sensorkort, og den andre enheten er et grensesnitt for brukeren, kalt displaykort. Disse to enhetene kommuniserer via USB [4] . Sensorkortet kan også tilkobles en datamaskin. Kravene til systemet har vi stort sett bestemt selv. Vi ble fort enige om at grafisk display var en forutsetning dersom vi skulle kunne vise informasjonen slik vi ønsket. Brukeren måtte også ha en måte å gi kommandoer til systemet og her valgt vi å benytte oss av et 16-knappers tastatur. Dette er de to eneste delene av systemet som fungerer som grensesnitt mot brukeren.

Vi fant for ut at det var hensiktsmessig å lagre data på bare et av kortene. Sensorkortet ble valgt for denne oppgaven fordi dette må ha en måte å lagre avlest data på. Kortet ble dermed utstyrt med et flashminne. Målingene måtte også kunne tidsbestemmes og dette førte til at vi integrerte en sanntidsklokke i systemet. Displaykortet fikk dermed ikke noe ektsternt minne for lagring av data og vil ikke ha noen funksjon med mindre det er tilkoblet sensorkortet. All konfigurasjon av sensorkortet foregår via displaykortet. Siden displaykortet har relativt mange funksjoner har vi valgt å la dette systemet være menystyrt.

Strømtilførselen til sensor- og displaykortet er implementert som eget kort, heretter kalt strømforsyningskort. Dette kortet har i tillegg til spenningsstabiliseringskretser for alle nødvendige spenninger blitt utstyrt med en batterilader. Batteripakken som skal tilkobles systemet vil dermed på en enkel måte kunne lades opp og systemet blir uavhengig av ekstern strømforsyning i lange perioder.

Vi har benyttet mange ulike verktøy for design og utvikling. De aller fleste av disse har vært tilgjengelig på DM-gruppens laboratorier.

[1] Printed Circuit Board
[2] Field Programmable Gate Array
[3] mikrokontroller produsert av Atmel
[4] Universal Serial Bus

Kommunikasjon mellom kortene skulle utføres via USB. Men siden USB bare definerer hvordan data skal sendes, ikke hva, har vi utviklet vår egen protokoll for dataoverføring. Denne protokollen gjør det mulig for mikrokontrollerne å prate sammen.

Designvalgene vi har tatt er for det meste forklart og diskutert gjennom hele rapporten. Men vi har også valgt å legge inn et ekstra kapittel med avgjørelser som er viktige for designet men som ikke passet inn andre steder i rapporten.

Til slutt i prosessen har vi testet systemet. All testing er dokumenter i slutten av rapporten. Vi fant her ut at enkelte valg vi hadde tatt ikke var hensiktsmessige, mens andre viste seg å fungere som forventet. Det var under denne fasen vi så resultatene av det vi hadde arbeidet med gjennom flere måneder tydeligst. Det var også her vi oppdaget utfordrende feil, som ikke alltid var like enkle å takle, men etter litt tid kom vi oss videre og fikk rettet opp det meste.

Alt i alt har dette vært en spennende og lærerikt prosjekt for alle prosjektets deltagere. Vi har hatt litt problemer med samkjøring og fremdrift underveis. Størrelsen på gruppen kan kanskje tillegges mange av årsakene her. Når vi var så mange ble det mulig for noen å trekke seg ut uten at dette ble avgjørende i en tidlig fase i prosjektet, derimot ble det avgjørende på slutten når ting skulle settes sammen. Dette er årsaken til at gruppen gikk fra 13 personer ved oppstart til 11 ved avslutning.

Vi har laget et system med mange fungerende enkeltdeler, allikevel viste det seg at vi ikke hadde tid og ressurser til å fullføre alle delsystemene. Alt i alt er vi fornøyd med resultatet og veien frem. Dette har gitt oss mange erfaringer å bygge videre på både i studiesituasjonen og i arbeidslivet i fremtiden.

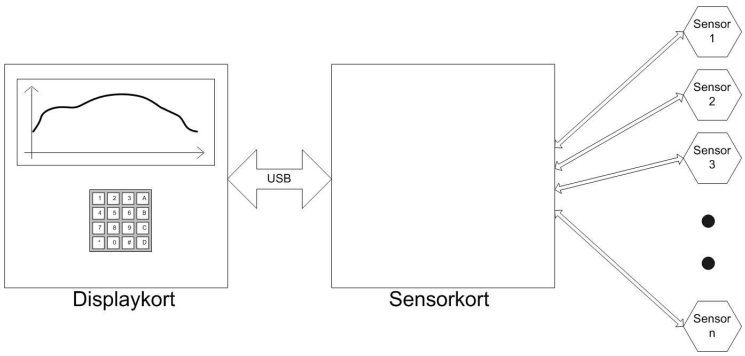
4 Oppgaveteksten

> "Isn't it interesting that the same people who laugh at science fiction listen to weather forecasts and economists?"
>
> – Kelvin Throop III

Vi fikk en veldig åpen oppgave. Kort fortalt var oppgaveteksten"Lag en værstasjon". Dette gav oss mange muligheter for hvordan vi kunne løse oppgaven. Vi fikk tildelt et budsjett på 15000 NOK som skulle brukes på prosjektet. Budsjettet inkluderte ikke FPGA, AVR eller PCB-produksjon. Det ble satt som et krav fra fagstaben at systemet skulle støtte USB, og at det skulle være mulig å koble til en PC. I tillegg var det et krav at systemet skulle bestå av 2 forskjellige kort som skulle kommunisere med hverandre.

5 Vår løsning

Ut fra oppgaveteksten skulle vi lage et system for måling og loggføring av ulike aspekter ved været. Utenom denne bestemmelsen sto vi veldig fritt til å utforme systemet som vi selv ønsket. Det var dog gitt enkelte restriksjoner for designet som vi måtte overholde. Som oppgaveteksten sa så skulle vi ha minst to delsystemer som var så adskilt at to grupper skulle kunne arbeide på systemene mest mulig uavhengig.



Figur 1: Sensorkort tilkoblet displaykort

Systemet består av to hovedenheter: Displaykortet, som er enheten som fungerer som grensesnitt mot brukeren, og sensorkortet som er enheten som står for innsamling og lagring av værdata. (Se figur 1 .)

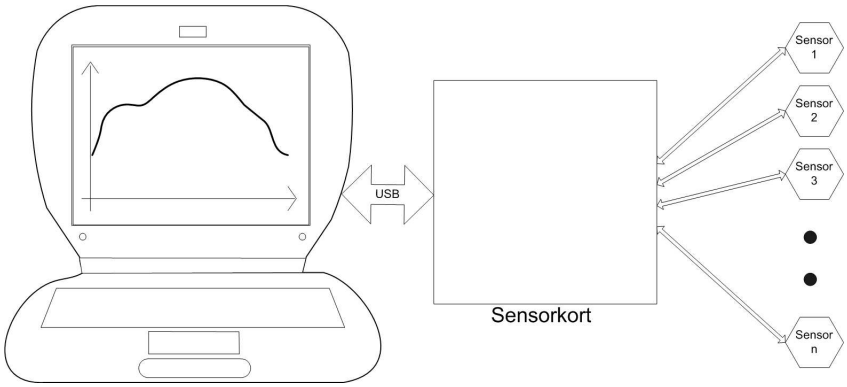
Vi bestemte at systemet skulle ha mulighet for grafisk visning av data og at det måtte ha et enkelt grensesnitt mot brukeren. Mye av designet er laget ut fra disse to kravene. Systemet er utstyrt med et LCD[5] display og tastatur med 16 knapper. All bruk av systemet kan gjøres ved hjelp av disse to komponentene som er plassert på displaykortet.

Systemet kan foreta målinger av mange ulike aspekter ved været. Avhengig av hvilke sensorer som kobles til kan det for eksempel måle temperatur, nedbør, vindstyrke, vindretning, lufttrykk, luftfuktighet og lysintensitet. Vi har lagt inn støtte for flere digitale sensorer. Disse kommuniserer over en buss kalt 1-Wire. Denne bussen gjør at sensorer kan kobles til eller fra uten at mye konfigurasjon er nødvendig. I tillegg har vi lagt inn støtte for at det kan benyttes en analog sensor. Denne kobles direkte på en AD-konverter på mikrokontrolleren på sensorkortet.

Til lagring av værdata benyttes det et flashminne. Dette er en minnetype som beholder skrevet data til tross for eventuelle strømbrudd. Det er benyttet en minnebrikke på 64Mbit, noe som gjør det mulig å la sensorenheten foreta måliger av været i flere år, gitt at avlesningene begrenses til en gang i timen.

Siden systemet skal lagre data, og det må finnes en måte å tidsbestemme målingene i ettertid, er sensorkortet utstyrt med en sanntidsklokke. Denne klokken vil alltid leses av før en måling skal lagres i minne, slik at måledataene kan lagres sammen med tidspunktet målingen er gjort.

[5]Liquid Crystal Display



Figur 2: Datamaskin tilkoblet sensorkortet

Siden sensorenheten ikke har noe grensesnitt direkte mot brukeren er det implementert kommunikasjon via USB mot displaykortet. USB-implementasjonen åpner også for mulighet for å koble til en datamaskin som grensesnitt. (Se figur 2 .) Datamaskinen kan da utføre de samme oppgavene som displayenheten og det vil i tillegg være mulig å oppnå langt bedre grafer og mer detaljerte visninger.

Begge delsystemene kan drives på batteri fra strømforsyningskortene eller det kan kobles til en batterieliminator. Benyttes det en batterieliminatoren vil denne også fungere som lader for batteriet.

6 Krav

> "The public demands certainties; it must be told definitely and a bit politics-raucously that this is true and that is false. But there are no certainties."
>
> – H.L. Mencken, "Prejudice"

Dette er kravene vi hadde til det ferdige systemet ved prosjektstart.

6.1 Overordnede krav fra veilederne

1. Det skal lages to forskjellige, atskilte kort med forskjellig funksjonalitet.
2. Kortene skal kunne kommunisere.
3. Det skal kunne kobles en datamaskin til et av kortene for flere og mer nøyaktige oversikter over data.

6.2 Våre overordnede krav til systemet

Dette er krav som ikke eksplisitt ble gitt i oppgaven, men som vi fant ut at var fornuftige for å få et velfungerende system.

1. Alle komponentene i systemet skal velges med tanke på at de må trekke minst mulig strøm for å spare batteri.
2. Data skal kunne lagres over lengre tid på et av kortene, nemlig sensorkortet.
3. Systemet skal ha to forskjellige konfigurasjoner:
   (a) Displaykortet kobles til sensorkortet for visning av data.
   (b) En datamaskin kobles til sensorkortet for visning av potensielt mer detaljert data.
4. Det skal være en batteripakke på hvert kort for drift.
5. Sensorkortet skal ha følgende sensorer tilkoblet:
   (a) Temperatur
   (b) Nedbør ved plussgrader
   (c) Vindstyrke
   (d) Vindretning
   (e) Lufttrykk

(f) Luftfuktighet

(g) Lysintensitet

6. Sensorkortet skal være en egen komponent som kan fungere uavhengig av andre kort.
7. Sensorkortet skal ha en realtimeklokke.
8. Sensorkortet skal kunne foreta målinger ved gitte tidsintervaller.
9. Displaykortet skal ikke kunne gi informasjon uten tilkobling til sensorkortet. Alle data lagres på sensorkortet og hentes kun ut for visning på displaykortet.
10. Displaykortet skal ha tastatur og display.
11. Brukergrensesnittet på displaykortet skal være enkelt og lett forståelig. Brukeren skal ikke måtte ha noen forkunnskaper for å få presentert ønsket infomasjon.
12. Strømforsyningen skal kunne drive kortene i

    (a) 6 mnd for sensorkort og

    (b) 2 mnd for displaykort

6.3 Mer detaljerte krav fra veilederne

1. Følgende komponenter må være på hvert kort: AVR og FPGA.
2. USB skal implementeres på FPGA.
3. FPGAen skal være av type Spartan IIE.
4. Mikrokontrolleren skal leveres av Atmel (AVR).
5. PCBen skal ha maksimalt fire lag.

6.4 Våre mer detaljerte krav til systemet

1. Komponentene til displaykortet må velges med tanke på at de skal kunne kommunisere med display og tastatur.
2. Komponentene til sensorkortet må velges med tanke på at de skal kunne kommunisere med sensorene.
3. Displaykortet skal fungere som host over USB, mens sensorkortet kun har som oppgave å sende data den blir forespurt om å sende.

4. Ved tilkobling av datamaskin i stede for displaykort vil datamaskinen fungere som host på USB.
5. Klokka til sensorkortet skal kunne bli stilt fra displaykortet eller fra en datamaskin.
6. Displaykortet styres av et tastatur.
7. Displaykortet kan slås av og på ved hjelp av bryter på batterikortet.
8. Sensorkortet skal kunne konfigureres for målinger med intervaller på ett minutt, en time, en dag, en uke, en måned eller et år. Konfigureringen skal kunne gjøres fra displaykortet eller en tilkoblet PC.
9. For å spare strøm skal sensorkortet for det meste være av. Når det skal på må det kunne "vekke" seg selv og "sovne" igjen etter at oppgaven er utført. Det er to tilfeller hvor sensorkortet er på:
   (a) Ved måling av data.
   (b) Ved aktivitet på USB-bussen, det vil si når displaykortet spør om informasjon.
10. Det skal lages egne strømforsyningskort for begge kort.

7 Systembeskrivelse

> "Computers are useless. They can only give you answers."
>
> – Pablo Picasso

Systemet er laget for å overvåke og å gi brukeren opplysninger om hvordan været er eller har vært den sist tiden. For at systemet skal kunne utføre disse oppgavene trenger vi flere undersystemer. Vi har valgt å dele systemet i to hovedkomponenter. Den ene hoveddelen, sensorkortet, står for innsamling og lagring av data, og den andre hoveddelen, displaykortet, fungerer som grensesnitt for brukeren. I tillegg består systemet av to strømforsyningskort som forsyner kortene med strøm.

Til sensorkortet er det koblet en mengde sensorer som sensorkortet leser av for å beregne klimatiske forhold. Forhold systemet lagrer informasjon om er temperatur, trykk, vindretning, vindhastighet, luftfuktighet, nedbør (regn) og lysstyrke. Sensorkortet har funksjonalitet for å regelmessig kunne foreta målinger uavhengig av tilkobling til displaykort. Dette gjør at sensorkortet kan fungere som en autonom enhet og samle inn værdata over lengre perioder uten at brukeren er tilstede. Brukeren kan koble til displaykortet senere for å lese ut dataene som har blitt lagret i minnet på sensorkortet i løpet av perioden. Ved hjelp av displaykortet er det mulig å justere inn hvor lange intervallene mellom målingene skal være. Brukeren kan også lese av en eller flere sensorer ved et hvilket som helst tidspunkt. Displaykortet og sensorkortet kommuniserer via USB. Dette er en kommunikasjonsprotokoll som er innført som standard for de fleste datamaskiner av nyere modell. En av fordelene med å bruke denne protokollen er at det fører til at sensorkortet kan kobles til en hvilken som helst datamaskin. Brukeren kan benytte seg av medfølgende programvare og utføre de samme operasjoner som med displaykortet. I tillegg gis det mulighet for lagring av data og noe mer detaljerte opplysninger og statistikker beregnet ut fra data som ligger lagret på sensorkortet.

For å gi en kort oppsummering av de ulike enhetenes oppgaver i systemet:

- Sensorkortet leser av og lagrer sensordata.
- Displaykortet fungerer som brukerens grensesnitt mot systemet.
- Strømforsyningskortene forsyner sensorkortet og displaykortet med strøm.

Sensorkortet har som nevnt, tilknyttet en mengde sensorer. Sensorene måler følgende forhold:

- Temperatur - Måler antall grader Celsius.
- Trykk - Måler lufttrykket i Pascal.
- Vindretning - Gir vindretning ut fra himmelretningene.
- Vindhastighet - Måler vindhastigheten i meter pr. sekund.

- Luftfuktighet - Gir luftfuktighet i prosent.
- Nedbør - Antall millimeter pr. døgn.
- Lys (natt / dag) - Gir om det er lyst eller mørkt ute. Det er ingen klart definert grense mellom dette.

De fleste sensorene er tilkoblet systemet via en busstandard som heter 1-Wire-Interface (OWI), mens lyssensoren er en analog enhet som er tilkoblet en AD-konverter på mikrokontrolleren.

Systemet tilbyr en rekke visninger av data, både som verdier og som grafer. Systemet gir brukeren mulighet til å be om avleste verdier for hver og en av senorene til bestemte tidpunkter eller man kan oppgi start- og slutt- tidspunkt om man ønsker å se utviklingen. Angir man et intervall kan man, foruten målt verdi, velge mellom å vise minimums-, middel- eller maksimalverdier. Ønsker man utviklingen over tid vil dette vises som en graf med for eksempel ett innslag pr. dag. Hvordan denne grafen tegnes opp bestemmes av bredden på intervallet og verdiene som skal vises.

8 Sensorer

> "Ummm, well, OK. The network's the network, the computer's the computer. Sorry for the confusion."
>
> – Sun Microsystems

8.1 Generelt

En værstasjon trenger sensorer. Vi brukte lang tid på å diskutere hvilke av disse vi ville bruke, ettersom vi visste at dette ville styre store deler av designet på sensorkortet.

8.2 Alternativene

8.2.1 Analogt

Våre første tanker gikk til primitive analoge måleinstrumenter. Vi visste at det fantes en god del av disse på markedet, og at innlesning ville være enkelt gjennom A/D konvertere. Ettersom en standard Atmel AVR mikrokontroller har opptil 8 AD-konvertere inkludert var dette en svært lovende mulighet. Ulempen er selvsagt at vi måtte ha brukt multipleksere for å ha flere enn disse 8. En annen ulempe er at en må trekke mange ledninger fordi hver enkelt sensor trenger en ny signalleder.

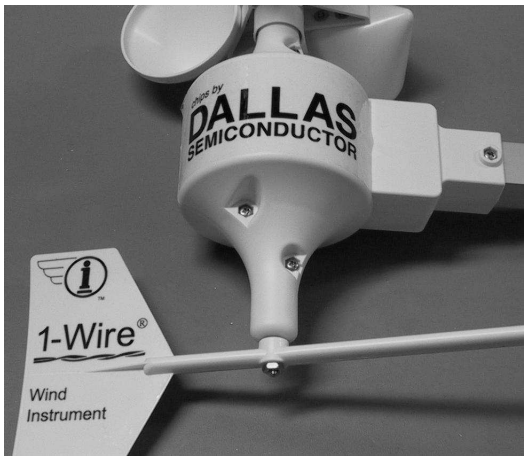
8.2.2 Digitalt : I2C

Det første digitale alternativet vi så på var I2C[6] basert. Denne løsningen brukes blant annet på hovedkort til PCer. Dette er en veldokumentert buss, og vi synes dette så ut til å være en god løsning. Problemet var at vi ikke fant noen sensorer for måling av vind og nedbør.

8.2.3 Digitalt : 1-Wire

Til slutt så vi på en løsning fra Aag Electronica kalt 1-Wire. Dette er et heldigitalt nettverksbasert system spesielt laget for værsystemer. Det betyr at vi slipper hele problematikken med mange ledninger som vi ville hatt ved analoge sensorer, samtidig som vi får tak i alle de sensorene vi ønsker oss. Dette er en bedre løsning, både estetisk og praktisk. Selve systemet kan gjøres selvkonfigurerende til en viss grad, ettersom hver sensor har en unik sensorid. Hvilket betyr at sensorer kan kobles til/fra uten mye konfigurering. Figur3 viser et eksempel på en vindmåler basert på 1-Wire systemet.

[6] Inter Integrated Circuit



Figur 3: Vindmåler til 1-Wiresystemet

8.2.4 Lyssensoren

Siden en lyssensor ikke var mulig å oppdrive, bestemte vi oss for å lage vår egen sensor. Dette ble gjort ved å lage et egetkretskort som kan observere hvor mye lys det blir belyst med. Les mer om lyssensoren i kapittel 9.

8.3 Konklusjon

Vi bestemte oss for en digital løsning basert på 1-Wire systemet. Dette fordi vi ønsket en nettverksbasert løsning, slik at vi slapp altfor mange ledninger. Det vil også gjøre det langt lettere for en sluttbruker som slipper å konfigurere systemet selv. Ettersom dette var en anerkjent løsning for værmåling hadde vi ingen problemer med å skaffe til veie de nødvendige komponentene. Det fantes dog ingen ferdige løsninger for målinger av lysstyrken. Dette løste vi ved å lage et eget kort for dette formålet.

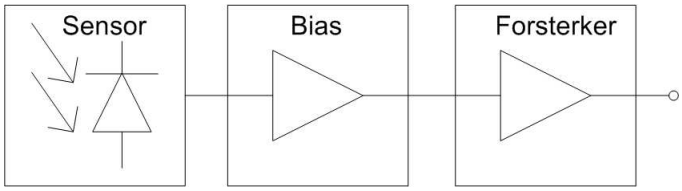
9 Lyssensor

> "Ar ilye tier undulaave lumbule.." ("And all roads are covered in darkness")
>
> – J.R.R Tolkien, The Lord of the Rings

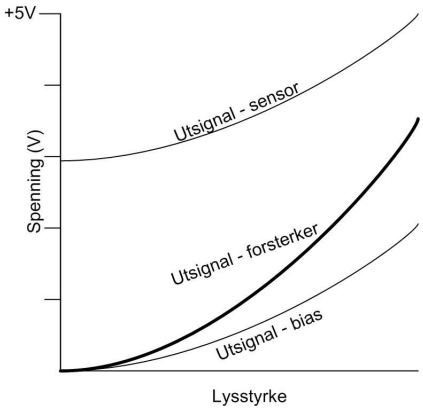
Lyssensoren er bygget rundt komponenten TSL250R[14]. Dette er en komponent som kun krever spenning og jord, og gir så ut en spenning i forhold til lysstyrken den belyses med. Fordi sensoren ikke er lineær benyttes det en tilpassningskrets bestående av to operasjonsforsterkere og et motstandsnettverk.

9.1 Virkemåte

Kortet har tre hovedkomponenter, se figur4.



Figur 4: Blokkskjema lyssensor



Figur 5: Eksempel på signal

Sensoren gir en utspenning avhengig av lysstyrken. Denne spenningen varierer mellom 0 og +5V. For å kunne regulere utspenningen har vi koblet inn en operasjonsforsterker som er koblet som en summerer. Operasjonsforsterkeren vi benytter er av typen LM258[13]. Kretsen er konstruert slik at den trekker fra en justerbar spenning fra utsignalet som kommer fra sensoren. På denne måten kan vi flytte signalet i spenningsnivå. Se figur5. Biasen justeres med P2 (se figur 107

i vedlegget). Denne blokken forsterker opp signalet slik at vi utnytter hele området til A/D-konverteren, i AVR'en, som kretsen skal kobles til. Igjen se figur5. Forsterkningen justeres med P1 (se figur 107 i vedlegget).

10 Bruksanvisning

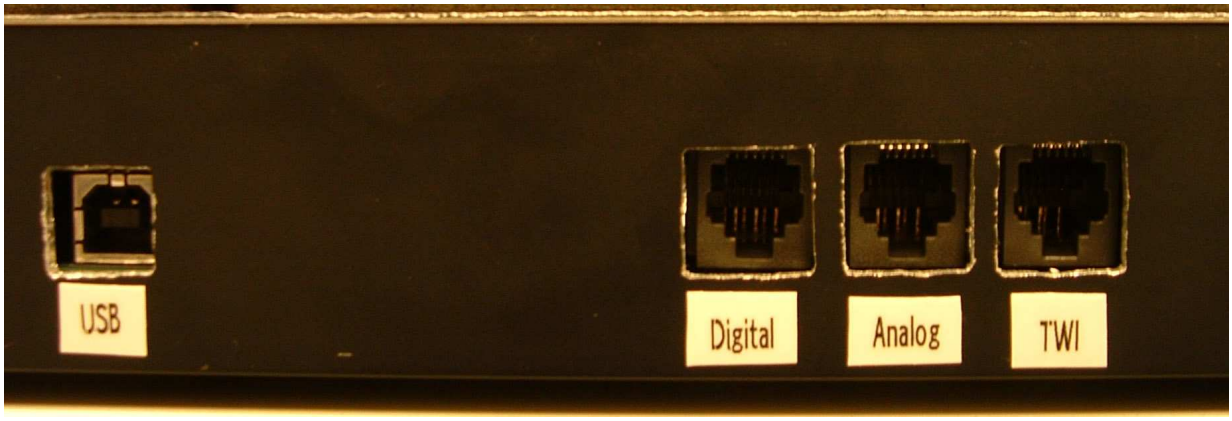
10.1 Oppkobling

Begge kortene er utstyrt med batteripakker, og kan dermed benyttes uten å tilknyttes strømnettet. Hvis det er ønskelig kan imidlertidig kortene kobles til nettet via en kontakt bak på kabinettet. På baksiden finnes også en på/av-knapp . Når sensorkortet skal stå å samle inn værdata må denne være satt 'på'. Da styrer kortet seg selv, og du trenger bare å koble til displaykortet for å lese av data, styre sensorkortet o.l. Se figur 6.

Sensorene kobles til kortet via plugger bak på kabinettet. Det er to forskjellige kontakter, en for de digitale sensorene, og en for de analoge (Se figur 7).

Kortene kobles sammen med en USB-kabel. Den samme kabelen kan benyttes når sensorkortet kobles sammen med andre enheter enn displaykortet, som for eksempel en PC.

Figur 6: Displaykortets bakside



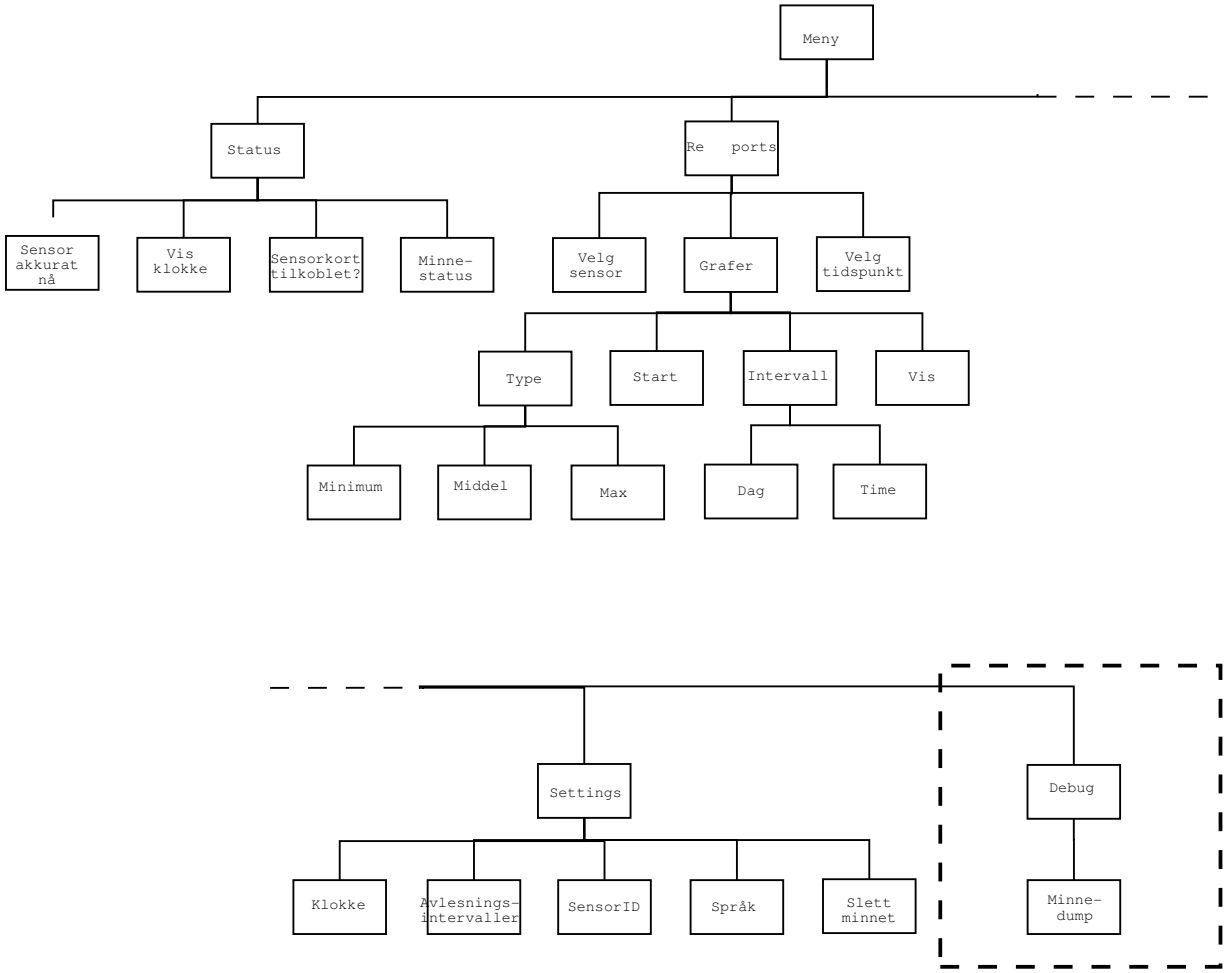
Figur 7: Sensorkortets bakside

10.2 Bruk

Displaykortet, og dermed hele systemet, styres ved hjelp av tastaturet og displayet som er på dette kortet. Når kortet skrus på vil displayet vise hovedmenyen. Menyene er tekstlige, og ønsket valg gjøres ved å trykke på det tilsvarende tall på tastaturet.

- Tasten **A** fører deg tilbake til hovedmenyen.
- Tasten **B** fører deg ett steg tilbake i menyhierarkiet.

En oversikt over menyen vises i figur 8. De menyvalgene som er inne i den stiplede boksen er debuggingsmuligheter.



Figur 8: Menysystemet

- Under statusmenyen kan man hente ut forskjellig informasjon om statusen til systemet. Man kan velge å avlese en sensor utenfor de faste oppgitte intervallene, for eksempel for å

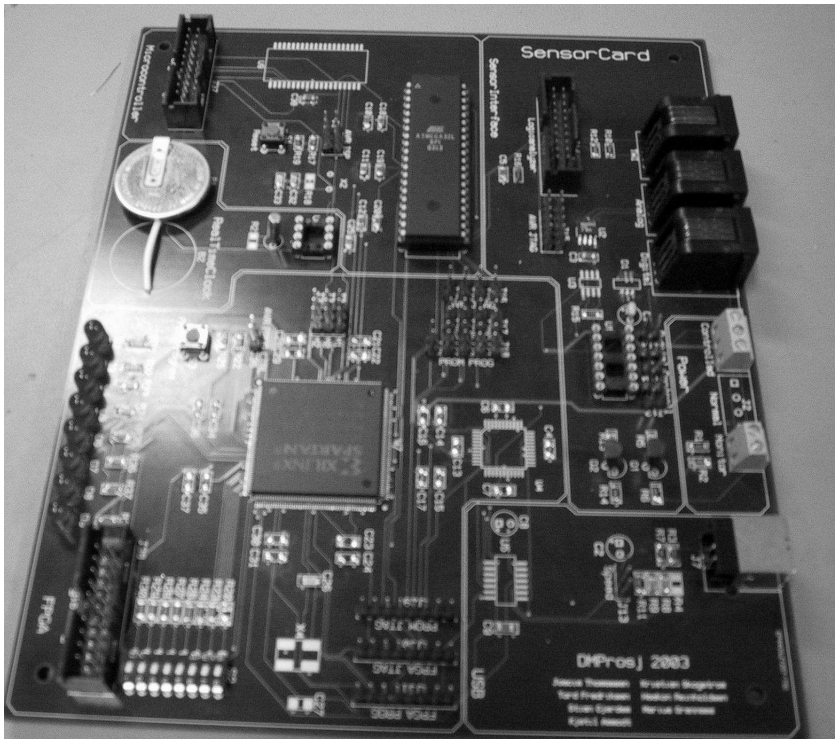
se temperaturen akkurat der og da. Videre kan man se hva klokken systemer opererer med er innstilt på. Siden det er denne klokken som knyttes opp mot de målingene som gjøres, er det lurt å passe på at denne er riktig. Neste valg gir en mulighet til å se om kortene er koblet sammen, altså om displaykortet har opprettet kommunikasjon med sensorkortet. Det siste valget gir brukeren mulighet til å se status på minnet på sensorkortet, hvor stort det er og hvor mye ledig plass det er igjen.

- Denne seksjonen gir brukeren muligheter til å se på de dataene som er registrert og lagret av systemet. Det første valget setter hvilken sensor som skal være den aktive. Det er denne man kan avlese under statusmenyen, og data fra denne som blir grunnlaget for grafene som kan tegnes. Når en graf skal vises kan brukeren velge mellom flere innstillinger. Grafene tegnes som søylediagrammer, og brukeren kan velge mellom tre forskjellige visninger. Ønsket starttidspunkt og intervall på visningen settes også her. Det siste valget i denne menyen viser den grafen brukeren har valgt.

- Under 'Settings' kan brukeren konfigurere systemet slik han/hun øsnker. Første gang systemet skrus på må klokken stilles, slik at de målingene som utføres blir knyttet opp mot riktig dato og klokkeslett. Brukeren kan så sette hvor ofte han vil at målinger skal utføres. Dette valget vil påvirke både hvor nøyaktig grafene vil bli, og hvor fort minnet fylles med data. På det neste valget knyttes en distinkt id for hver sensor opp mot gitte sensortyper. Det siste valget kan brukeren benytte hvis han ønsker å slette de dataene som er lagret i minnet på sensorkortet. Minnet vil tømmes helt, og målingene starter på nytt.

- Siden dette er en prototype, har menyen fremdeles et valg for feilsøking og testing. Denne biten er merket med en stiplet firkant i figuren. Med en mulighet for å dumpe minnet kan man finne feil lettere, og utviklingen blir enklere. Dette er selvsagt noe som vil fjernes fra en ferdigutviklet versjon.

11 Sensorkortet



11.1 Hensikt

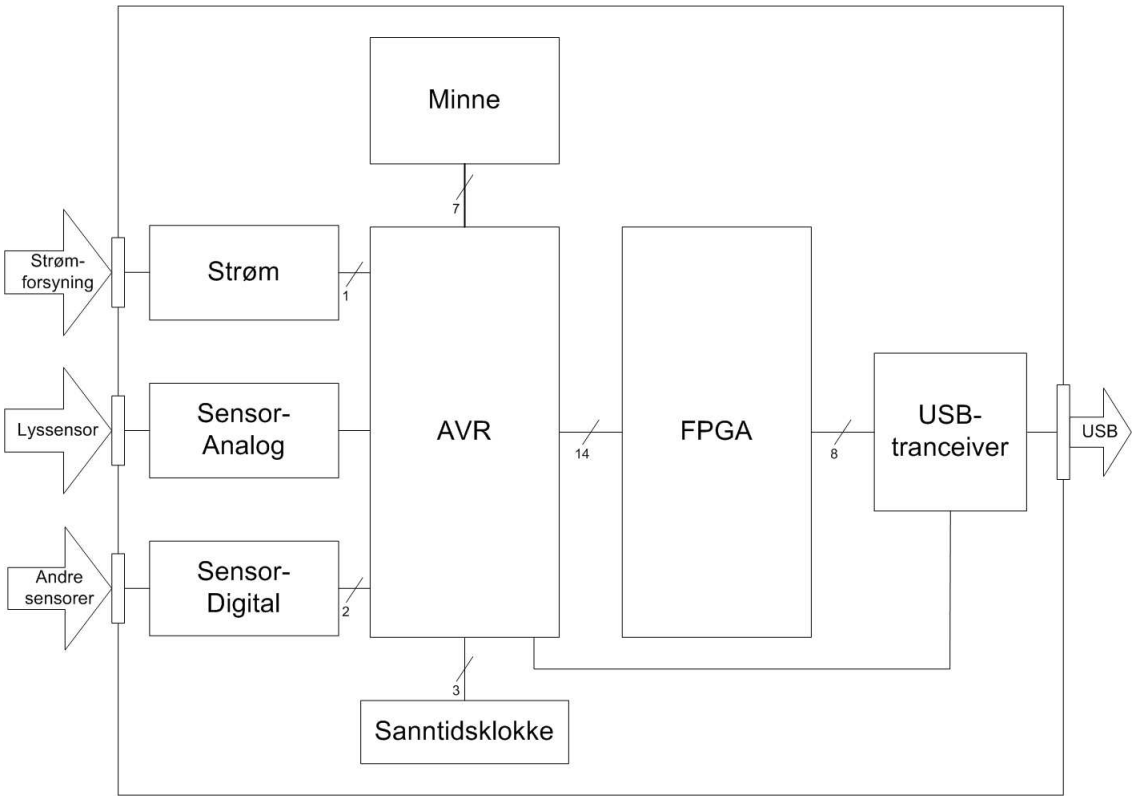
Hensikten med sensorkortet er tredelt. Hovedmålene er:

1. Lese inn sensorinformasjon, både fra analoge og digitale sensorer.
2. Lagre disse på et ikke-flyktig medium, sammen med et presist klokkeslett.
3. Kommunisere med omverdenen via et USB grensesnitt for
   (a) Konfigurasjon
   (b) Lesing av registrerte data

Sensorkortet prosesserer ikke noe av dataene (min, max, gjennomsnitt osv). Denne oppgaven er forbeholdt displaykortet, eller andre enheter (f.eks PC). Her finnes det ingen begrensninger, så lenge den andre enheten kan snakke vår egendefinerte protokoll (Se kapittel15).

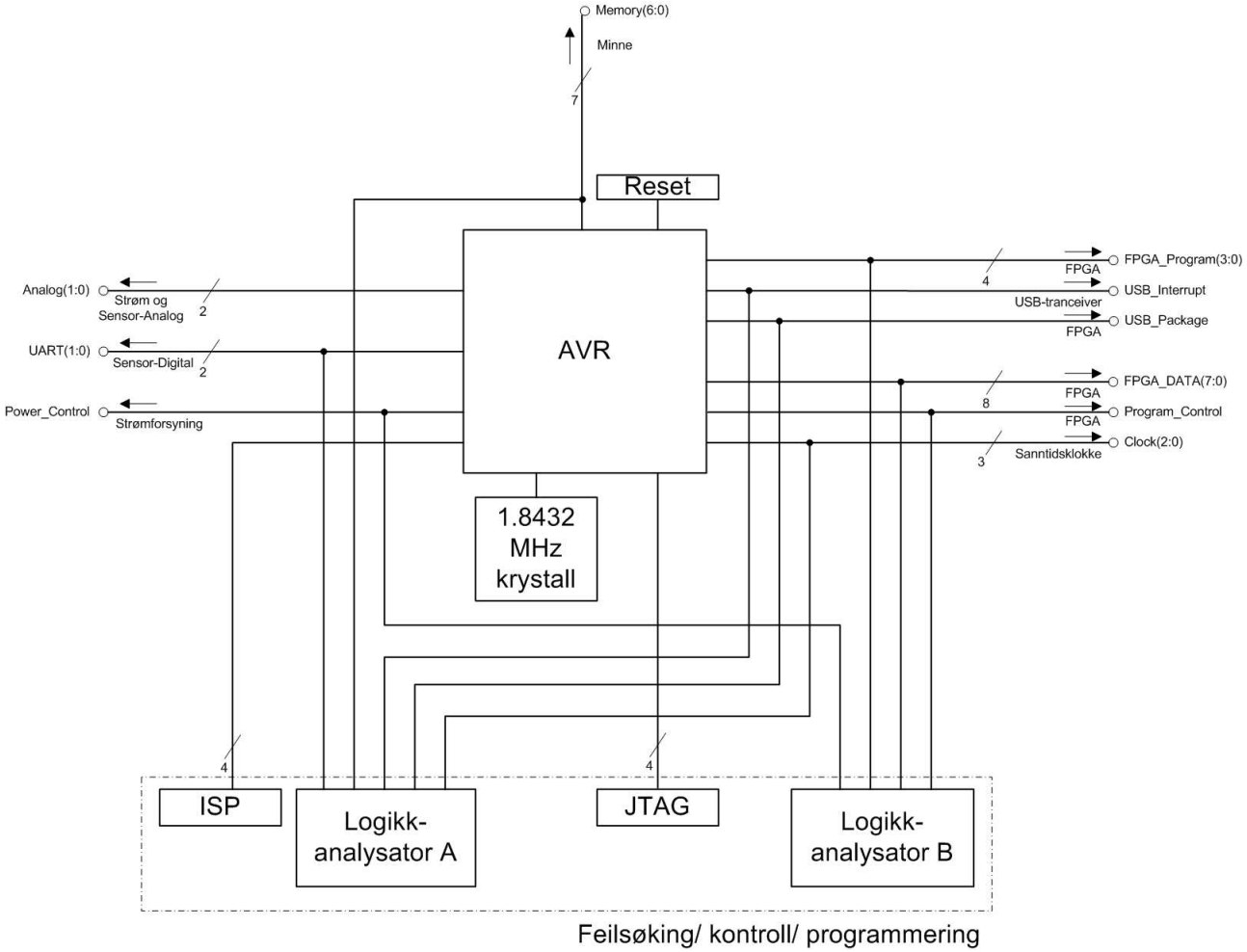
11.2 Hardware

Sensorkortet består av flere systemer. Se figur9. Mikrokontrolleren tar seg av innlesing av sensordata og lagrer den i minnet. Sanntidsklokken sørger for at alle data får et presist tidsstempel. Mikrokontrolleren styrer også strømtilførselen til FPGA'en. FPGA'en igjen tar seg av kommunikasjon med USB. Hensikten med å styre strømmen er å kunne kontrollere strømforbruket, for så å kunne øke batterilevetiden. Resonnementet her baserer seg på at USB bussen stort sett ikke er i bruk, og kan da med fordel skrus helt av. Vi kunne valgt å stoppe klokken i stedet, men valgte å prøve denne løsningen fordi vi mente at vi kunne få enda mindre strømforbruk slik.



Figur 9: Blokkdiagram over sensorkortet

11.2.1 Mikrokontrolleren



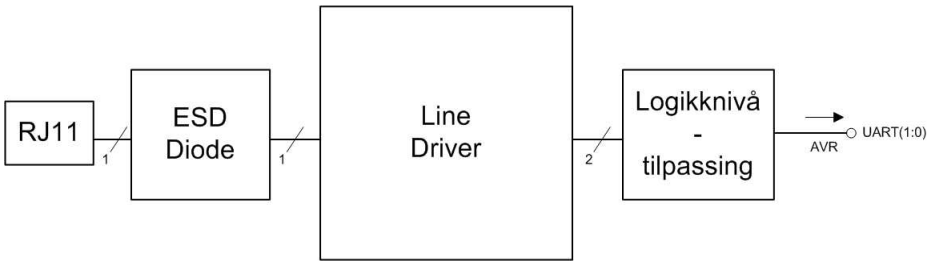
Figur 10: Blokkdiagram for mikrokontroller

Mikrokontrolleren har tre distinkte oppgaver. For det første leser den av sensorene, både analoge og digitale. Deretter leser den av sanntidsklokken for å sette et nøyaktig tidsstempel på dataene. Denne informasjonen lagres så i en flashbrikke. Så sover den i et brukerdefinert tidsintervall for deretter å gjenta hele prosessen. Programflyten blir som følger:

1. Les av sensorene. (se avsnitt11.2.2 og11.2.3)
2. Les av sanntidsklokken. (se avsnitt11.2.4)
3. Skriv dataene til minne. (see avsnitt11.2.5)
4. Sov i et brukerdefinert intervall, for deretter å gjenta prosessen fra trinn 1.

Til dette formålet brukte vi en ATMega32L[2]. For å lette debugging av kortet har vi lagt inn en rekke bryterere og tilkoblingspunkter. Vi har to forskjellige måter å programmere AVR'en på. Vi har trukket opp linjene for ISP[7]. Tilkoblingspunktet er pinnekompatibelt med STK500 kortet fra Atmel. Denne tilkoblingen er merket"AVR ISP". Den andre metoden er mer primitiv. Vi bruker rett og slett en sokkel for å kunne ta ut og inn mikrokontrolleren. Dette tjener to hensikter. Den viktigste er at vi har muligheten til å sette inn en emulator, og den andre er at vi kan programmere mikrokontrolleren med f.eks en STK500. For å resette kontrolleren har vi en knapp merket"reset", om denne trykkes inn vil man få en hard reset av mikrokontrolleren. Videre har vi trukket ut JTAG tilkoblinger til debugging. Denne er standard og pinnekompatibel med Atmels hardware. For å kunne overvåke batterinivået er spenningen på batteriet koblet til en av A/D konverterene via en spenningsdeler.

11.2.2 Digitale sensorer



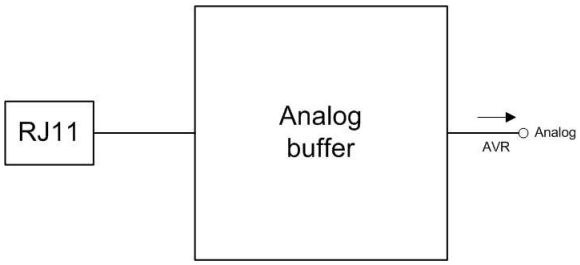
Figur 11: Blokkdiagram for digitale sensorer

De digitale sensorene er koblet til sensorkortet gjennom en RJ11 plugg (Se figur11). Dette signalet går først gjennom en ESD-diode. Hensikten med denne dioden er å beskytte de bakenforliggende kretsene mot elektrostatisk utladning. De digitale sensorene benytter seg av en buss som på engelsk kalles"1-wire". For å oversette fra denne bussen til seriell kommunikasjon benytter vi oss av en linedriver kalt DS2480B [4]. Kommunikasjon mot denne brikken foregår serielt. Linedriveren har en driftsspenning på 5V. Resten av kortet har en driftsspenning på 3.3V. Dette innbar at vi trengte en måte å oversette logikknivåer på. Dette løste vi ved å bruke to transistorer og inverterende Schmitt-triggere. De digitale sensorene er koblet til AVR'ens serielle interface på port d.

11.2.3 Analoge sensorer

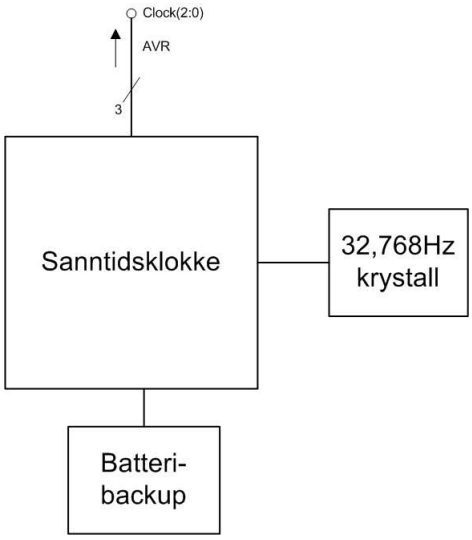
Lyssensoren er koblet til sensorkortet gjennom en RJ11 plugg (Se figur12). For å kompensere for linjestøy har vi valgt å bruke et analogt buffer av typen MAX4202[9]. Dette signalet går så videre til A/D konverterene på mikrokontrolleren. Dette gir oss en 10 bits oppløsning på avlesningen.

[7] In System Programming



Figur 12: Blokkskjema for analoge sensorer

11.2.4 Sanntidsklokke

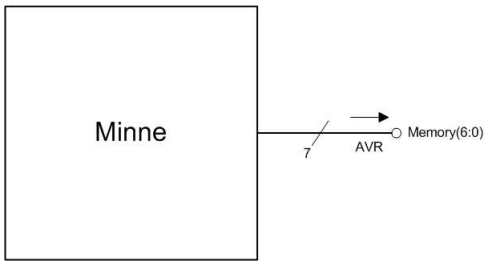


Figur 13: Blokkdiagram for sanntidsklokke

Sanntidsklokken brukes til å tidsstemple sensoravlesningene. Vi valgte å bruke en krets av typen DS1302[10] på grunn av dets enkel serielle grensesnitt. Denne benytter en standard 32.768kHz klokkekrystall (Se figur13). Kommunikasjon med denne kretsen skjer ved hjelp av en proprietær seriellbuss. For å ikke miste kalenderdata om strømmen skrus av har den batteribackup i form av et lithiumsbatteri. Sanntidsklokken er koblet til AVR'en på port d.

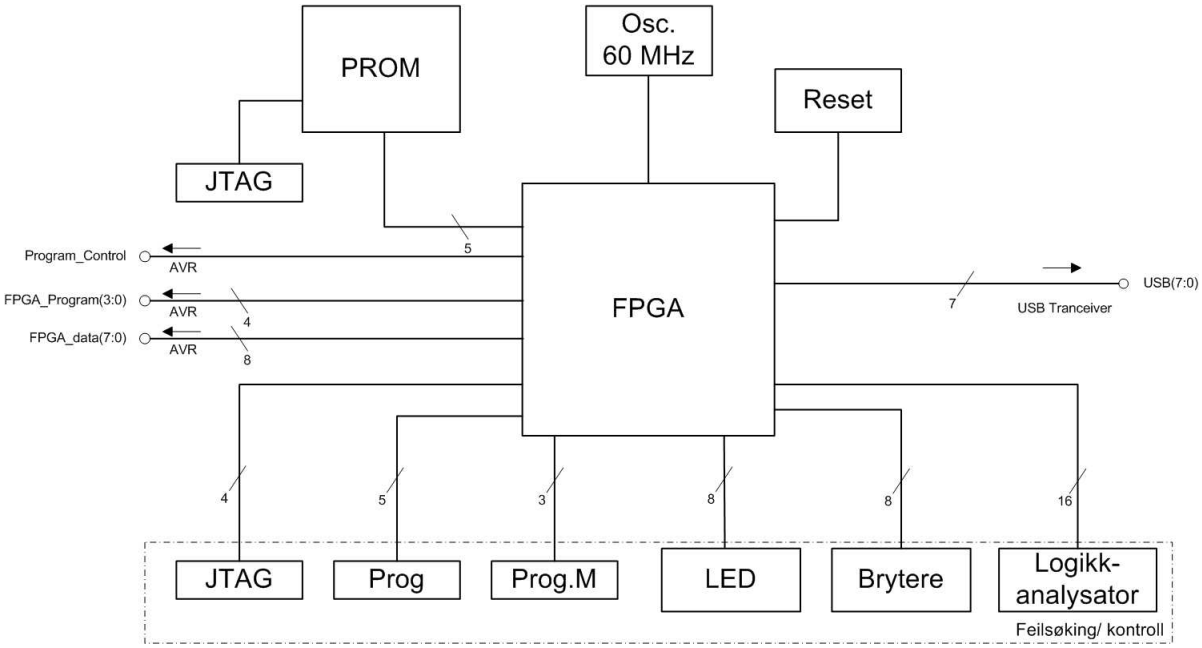
11.2.5 Minne

Vi valgte å bruke Atmel sin dataflash for lagring av sensordata. Dette valget baseres på to argumenter; for det første er den veldokumentert sammen med AVR mikrokontrollere, og for det andre er grensesnittet mot den veldefinert og gjennomtenkt. Dette er en buss på 7 ledere (se figur14). I tillegg finnes det masse eksempelkode for minnebrikken. Brikken vi endte opp med het AT45DB642[11]. Denne brikken er 64Mbit stor, og burde holde til minst 4 år sammenhengende registrering av sensordata med 1 times mellomrom.



Figur 14: Blokkdiagram for minnet

11.2.6 FPGA



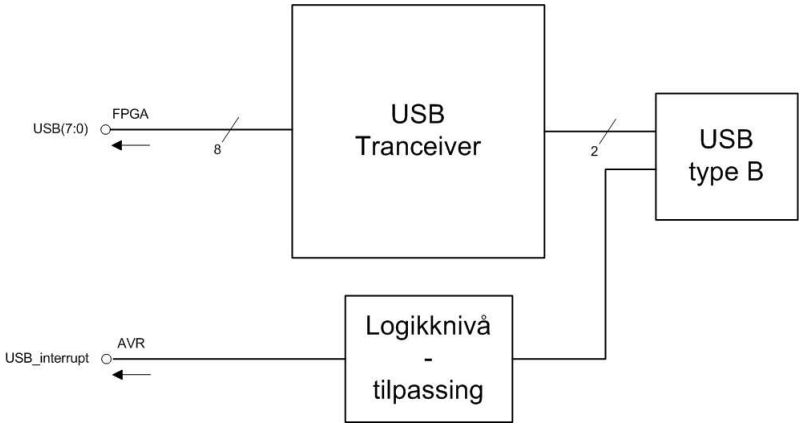
Figur 15: Blokkskjema for FPGA

FPGA'en vi skulle bruke var av typen Spartan IIE[3]. FPGAen har ansvar for USB systemet. Dette gjør den gjennom USB-transcieveren (se avsnitt11.2.7). I andre enden skal den snakke med mikrokontrolleren (Se figur41). Vi har trukket opp 8 linjer til dette formålet. Dette inkluderer TWI (Atmels implementasjon av I_2C). Dette må til for at det skal være mulig å implementere flere forskjellige protokoller, alt etter hva som viser seg å være fornutig. FPGAen blir klokket gjennom en dedikert oscillator på 60Mhz.

Programmering kan foregå på tre forskjellige måter. Til debugging vil det være mest vanlig å koble en PC med en FPGA programmerer på direkte til kortet. Etter at utvikling og debugging er ferdig kan man legge programmet på to forskjellige steder, alt etter hva som er mest optimalt. Enten kan man legge det inn i en PROM, eller så kan man la AVR'en programmere den direkte. Tanken her er å kunne legge FPGA programmet direkte i minnet til mikrokontrolleren.

For å lette debuggingen av FPGA'en er det lagt inn en mange muligheter. Vi har trukket opp JTAG interfacet på FPGA'en. Denne er videre dokumentert i databladet[3]. Ellers er det koblet opp 8 lysdioder, en rekke med 8 brytere og tilkobling for logikkanalysator.

11.2.7 USB transceiver



Figur 16: Blokkdiagram for USB-transceiveren

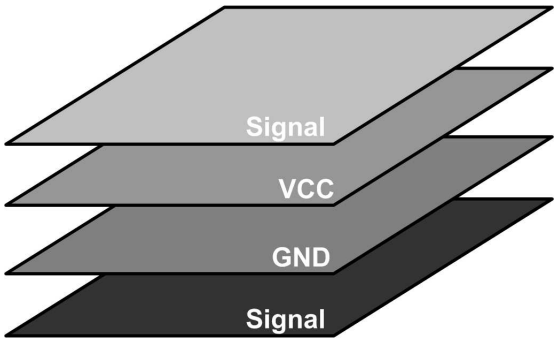
Hensikten med USB transceiveren er å tilpasse de logiske signalene fra FPGA'en til de elektriske karakteristikkene til USB-bussen. Oppbygningen er som i figur16. Transceiveren støtter både overføringer på 12Mbps og 1.5Mbps. Disse har forskjellige elektriske karakteristika. For å velge mellom de to elektriske karakteristikkene brukes jumpere. For at AVR'en skal oppdage at USB-bussen er i bruk, og dermed kunne sette i gang USB-systemet, er USB-bussen koblet på en avbruddslinje på AVR'en. Når noen kobler seg til sensorkortet via USB-bussen vil denne enheten legge 5V på avbruddslinjen. Denne spenningen legges på AVR'en gjennom en spenningsdeler. Denne linjen kalles"USB_interrupt" på skjemategningen.

11.2.8 Strømtilkobling

For å kunne spare strøm, er det mulig å skru av strømmen til hele USB systemet. AVR'en må legge en pinne høy for at FPGA'en, klokken og USB-transceiveren skal få strøm.

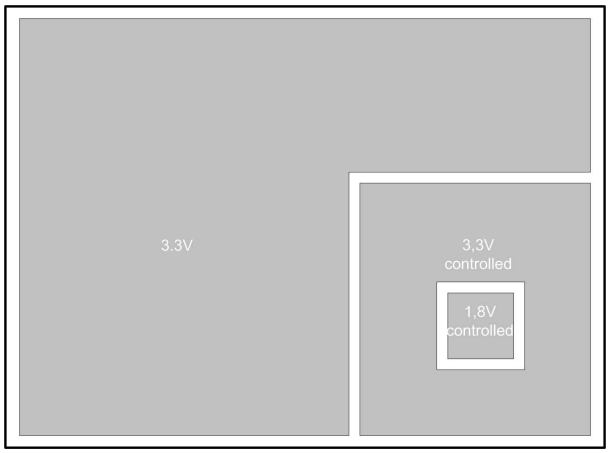
11.3 PCB

PCB utlegget til sensorkortet er gjort så enkelt som mulig. Vi har grupert de forskjellige funksjonelle delene, slik at kortet skulle bli oversiktlig. Kortet har fire lag (se figur17). To av lagene er signal-lag, mens ett er jordlag, og ett er powerlag. For å slippe å rute strøm rundt FPGAen er powerlaget splittet opp i tre biter (Se figur18) . Vi har +3.3V på delen som leverer strøm til AVR'en. Delen med FPGA'en har +3.3V (der vi har mulighet til å skru av strømmen) og +1.8V (der vi også



Figur 17: Lagdelingen av PCB'en

kan skru av strømmen). Fysisk sett ser det ut som på figur18. Signalene ble stort sett autorutet av PCB-verktøyet, med noen få unntak. Blant annet er powerlinjene til de kontrollerte delene av kortet gjort litt bredere for å kunne gi mer strøm om nødvendig. I tillegg har vi prøvd å gjøre klokkelinjer/høyhastighetslinjer så korte som mulig.



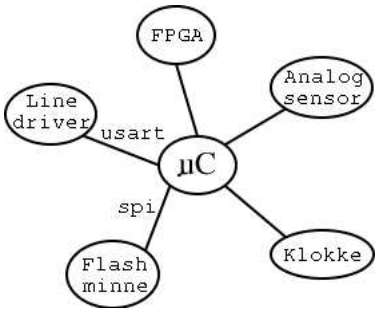
Figur 18: Powerlaget på sensorkortet

11.4 Mikroprosessor Software

> "No hardware designer should be allowed to produce any piece of hardware until three software guys have signed off for it."
>
> – Andrew S. Tanenbaum

11.4.1 Overordnet

Mikrokontrolleren(uC) skal i hovedsak kommunisere med fem enheter som vist i figur19. Protokollen som benyttes mellom enhetene er:



Figur 19: Enhetene mikrokontrolleren (uC) kommuniserer med

- FPGA : Egendefinert protokoll
- Sanntidsklokke : IC[8] spesifikk protokoll
- Linedriver : USART[9]
- Flashminnet : SPI[10]
- Analogsensor : mikrokontrollerens innebygde a/d konverter

Den kan selv igangsette kommuniksjon med alle enhetene, men kun FPGA enheten kan varsle mikrokontrolleren og be om kommunikasjon. Det gjør den via den eksterne interrupt inngangen. Under programutførelsen vil det være displaykortet og pc programvaren som via Usb porten, og Fpga enheten vil kunne endre programrutinen og hente ut data fra mikrokontrolleren.

Hovedrutinen

Når mikrokontrolleren starter opp vil programmet først gå inn i en initialiseringsfase hvor det starter med å enable interrupts, slik at mikrokontrolleren kan ta imot data fra Displaykortet/PC'en, se figur20. Ved oppstart vil sanntidsklokken være nullstilt, det er derfor viktig at ingen sensoravlesninger skjer før brukeren har stilt klokka. Dette tar Displaykortet seg av. Nå er programmet klar til å motta forespørsler fra USB porten, og kan dermed starte den rutinemessige vekslingen mellom å sove og avlese sensorene. Sensorene skal avleses etter et bestemt intervall som kan justeres fra displaykortet / pc'en. Den setter i gang timeren i henhold til samplingsintervallet og går inn i idle[11] mode.

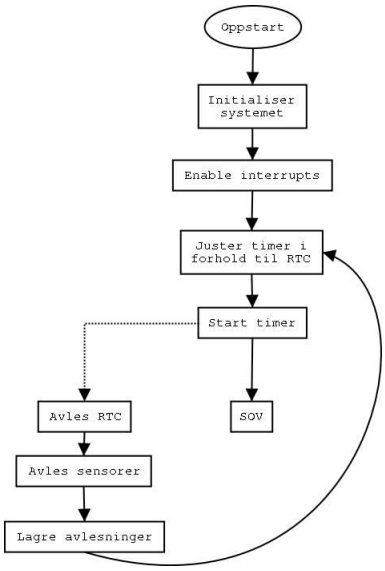
Ved sensoravlesning trigget av timer, leses først real time klokka av, deretter samtlige sensorer. Det taes ikke hensyn til tidsforsinkelsen i avlesningene i forhold til rtc, siden sensoroppsettet og

[8] Integrated Circuit

[9] Universal Synchronous Asynchronous Receiver / Transmitter

[10] Serial Peripheral Interface

[11] Helst skulle vi brukt power-down mode, men det er ikke mulig siden sanntidsklokka ikke har støtte for interrupts.



Figur 20: Sekvensdiagram over hovedrutinen på mikrokontrolleren

dermed forsinkelsen vil være omtrent den samme for hver avlesning. Alle data lagres så med sensorid, timestamp, og postid i flashminnet.

Interrupt rutinen

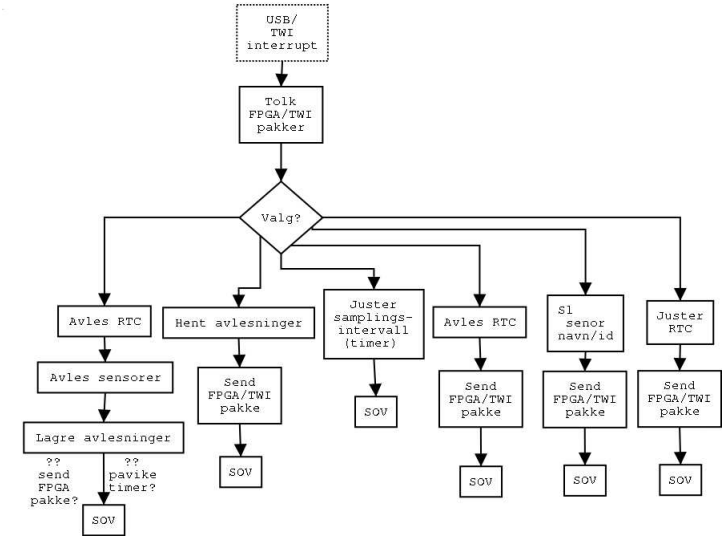
Nå og da vil det komme et interrupt fra Fpga'en. Programvaren må da tolke inndata og avgjøre hva slags forespørsel det er, se figur21.

- Juster samplingsintervallet
- Juster sanntidsklokken
- Sjekk tida (sanntidsklokken)
- Sjekk sensortypen
- Avles sensor(er)
- Hent avlesninger fra minnet

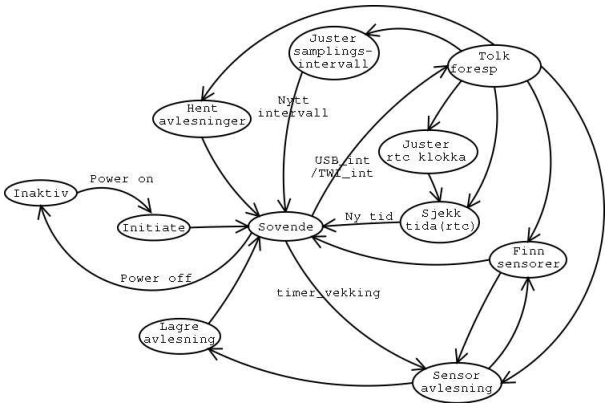
Oversikt over programmets tilstander

11.4.2 Digitale sensorer

De digitale sensorene er koblet sammen på en seriell buss som kalles 1-wire bussen. Dette er en enkel bus for to-veis kommunikasjon mellom master, linedriveren, og sensorehetene, se



Figur 21: Sekvensdiagram for interrupt håndteringen

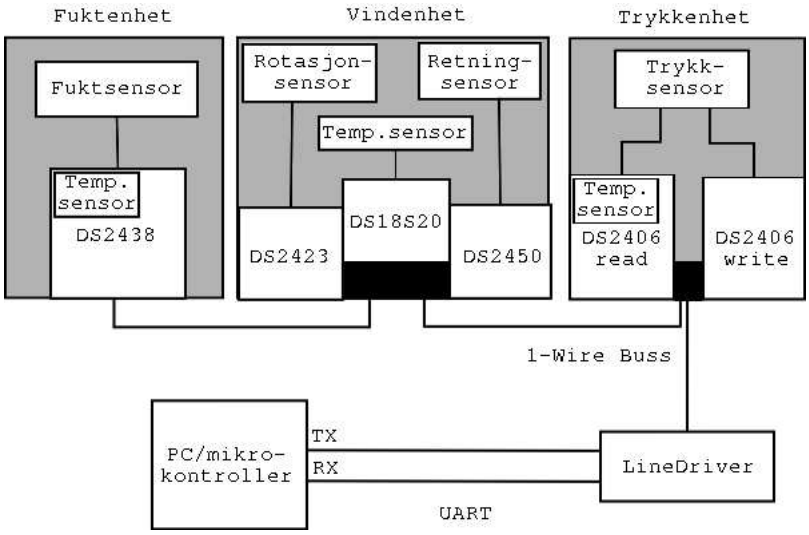


Figur 22: Tilstandsdiagram for mikrokontrolleren

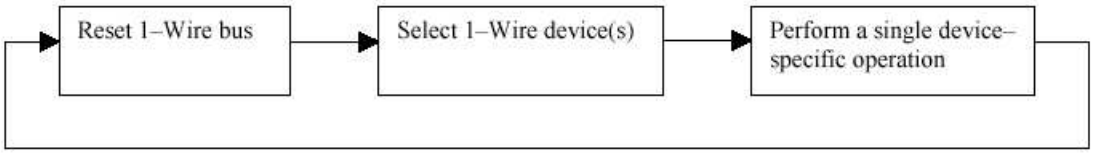
figur23. Hver sensorenhet har en eller flere integrerte kretser (IC'er) som kommuiserer over 1-wire bussen. Disse ic'ene har alle en unik id lagret i rom minne. Dermed kan vi adressere hver ic/sensor for kommunikasjon selv om det er mange enheter tilkoblet.

Som vi ser av figur24 starter all kommuniksjon med at linedriveren sender ut en reset kommando som synkroniserer bussen. Siden mikrokontrolleren styrer linedriverenheten, er det den som kontrollerer selve kommunikasjonsflyten. Deretter velges en sensorenhet for videre kommunikasjon. Dette gjøres ved å sende ut en"match_rom" kommando, etterfulgt av romkoden. Da vil alle sensorer/ic'er som ikke har tilsvarende rom-id ignorere videre kommunikasjon til neste reset kommando sendes ut. Når sensoren/ic'en har blitt valgt, kan master sende kommandoer som er definert for ic'en. Samt lese data fra ic'en, og skrive data til ic'en. Hver ic har en unik protokoll, men de følger alle den samme utvelgelses sekvensen, se figur .

Sensorprogramvaren er inndelt i to hovedseksjoner:



Figur 23: 1-Wire bussen og enhetene



Figur 24: Typisk 1-wire kommunikasjonsflyt

1-Wire kontroll

Sensoravlesning

1-Wire kontroll delen tar seg av all den generelle 1-Wire kommunikasjonen som ikke er sensor/ic spesifikk. Det er i denne delen grensesnittet mot hovedprogrammet ligger.

Sensoravlesningsdelen inneholder de spesifikke protokoll metodene for hver sensor/ic som skal avleses.

11.4.3 Sanntidsklokke

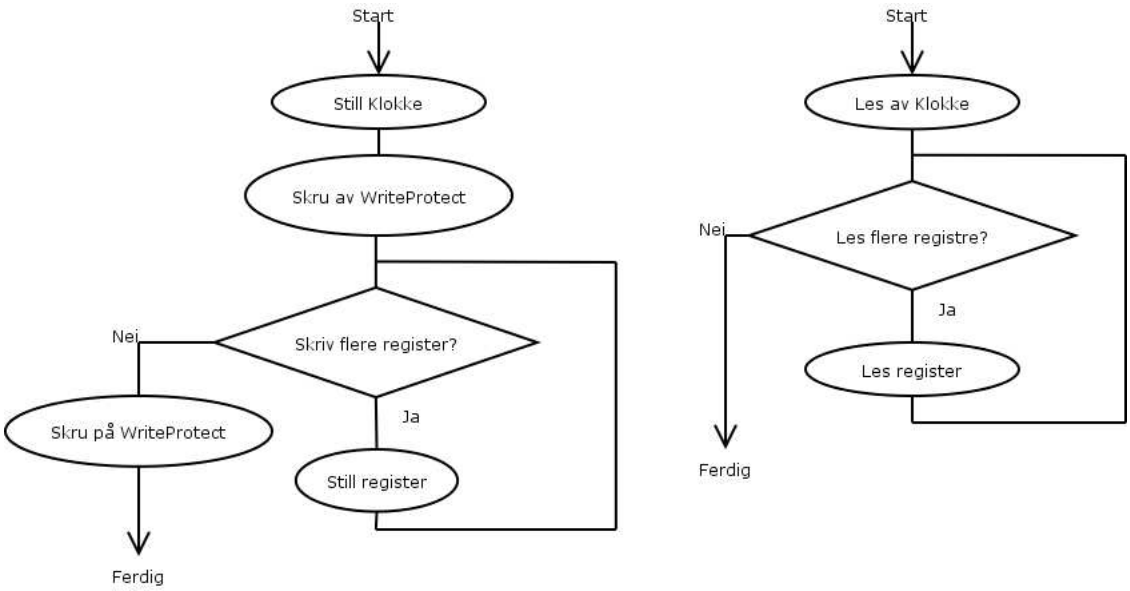
Tilstandene til klokkekode er som følger:

Still Klokke : Benyttes for å stille klokken

Les av Klokke : Benyttes for å lese av klokken.

Deltilstandene for hovedfunksjonene er som følger

Still Klokke : Tar sekund, minutt, time, dato, måned og år som argument.



Figur 25: Tilstands diagram for sanntidsklokken

Skru av WriteProtect : Disabler en sperre som hindrer skriving til klokkeregisterene.

Skriv flere register? : Sørger for at hvert av argumentene blir gjennomgått.

Still register : Stiller det akutelle registret

Skru på WriteProtect : Enabler en sperre som hindrer skriving til klokkeregisterene.

Merk, sekundregistret i klokken blir stilt sist, da dette starter klokken.

Les av Klokke : Skal lese av de forskjellige registerene på klokken, og returnere dem i et buffer spesifisert ved funk.kall.

Les flere register? : Sørger for at hvert register blir avlest.

Les register : Leser av det aktuelle registret og lagrer verdien i returbuffer.

Merk. Det er mulig å stille/lese ett og ett register manuelt, men dette er ikke anbefalt, da det krever manuell konvertering av in-/outputverdier, da brikken lagrer tallene på en litt spes. måte.

11.4.4 Minne

Hovedtilstandene i minnekoden:

Avhengig av hvilke funksjon som blir kalt, går den inn i en av disse tilstandene. Tilstander uten veier ut er også slutttilstander.

Les direkte : Dennen benyttes til å lese av en del av minnet med direkte addressering.

Les Post : Benyttes for å lese av en post. 3 varianter

- *Siste post* : Henter ut den siste posten som ble lagret.

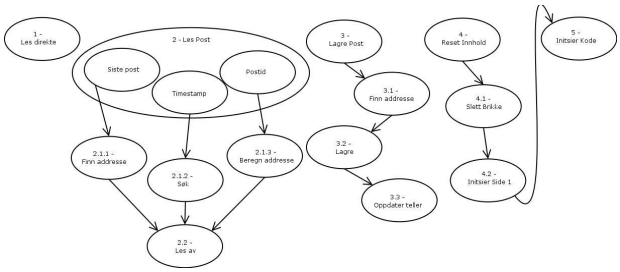
- *Timestamp* : Henter ut en post lagret ved tidspunkt spesifisert.

- *Postid* : Henter ut en post spesisfisert av en id som lett lar seg oversette til en minneadresse.

Lagre Post : Lagrer en ny post på brikken.

Reset Innhold : Sletter alt innholder på brikken

Initsier Kode : Setter opp starttilstanden



Figur 26: Tilstandsdiagram for minnedriveren

Deltilstandene er som følger

1 - Les direkte : Leser direkte fra brikken. Addresse, returbuffer og lengde blir spesifisert ved kall. Benytter en enkel kommando mot brikken (Continuous Array Read). Funksjonen som denne tilstanden implementerer blir ofte benyttet i de andre tilstandene

2 - Les Post : Som nevnt over. Finnes 3 varianter av denne.

2.1.1 - Finn addresse : Benytter tellervariabler for å finne ut hvor neste post skal lagres, og bruker denne addressen til å beregne hvor forige post ble lagret.

2.1.2 - Søk : Benytter tellervariablerfor å finne ut hvor mye av brikken som benyttes. Benytter derretter et binærsøk for å finne posten som tidspunktet stemmer best overens med.

2.1.3 - Beregn addresse : Siden hver side kan lagre like mange poster, og vi lagrer sekvensiet utover fra starten av brikken, med sekvensiet økende postid, så kan vi enkelt finne ut hvor en enkel postid ligger lagret.

2.2 - Les Av : Utfra enten*2.1.1*, *2.1.2* eller *2.1.3* vet vi nå addressen til posten som vi skal hente. Vi benytter*Les direkte* og lagrer resultatet i et globalt definert buffer.

3 - Lagre Post : Lagrer en post til flashbrikken. Hva som skal lagres ligger i et globalt definert buffer.

3.1 - Finn addresse : Benytter tellervariabler for å finne addressen som neste post skal lagres på

3.2 - Lagre : Skriver timestamp til post, og lagrer på den aktuelle addressen.

3.3 - Oppdater teller : Oppdaterer tellervariablene og skriver de tilbake til første side.

4 - Reset Innhold : Sletter hele innholder på brikken.

4.1 - Slett Brikke : Sletter hele minnet.

4.2 - Initsier Side 1 : Skriver til side 1, og spesifiserer at ingen poster er skrevet til brikken, og skriving skal starte på andre side. Går automatisk over til tilstand*5 - Initsier Kode*.

5 - Initsier Kode : leser av første side og setter tellervariablene deretter.

I enkelte tilfeller vil ting ikke fungere som forventet.

- *1 - Les direkte* : Side og/eller addresse er ugyldig (for stor). Ukjent. Driver skal ikke havne i denne situasjon.
- *2.1 - Les Siste Post* : Ingen poster er lagret. Software ignorerer dette, og leser av fra starten av andre side.
- *2.1.2 - Søk* : Timestamp er før første avlesning. Software returnerer første avlesning.
- *2.1.2 - Søk* : Timestamp er etter siste avlesning. Software returnerer siste avlesning.
- *2.1.2 - Søk* : Timestamp er mellom to avlesninger. Software returnerer avlesning før timestamp.
- *2.1.3 - Beregn addresse* : Postid er større en anntall poster lagret. Software returnerer 0
- *3 - Lagre Post :*Lagring av post ved fullt minne. Software fanger dette opp, og returnerer 0 uten å lagre posten.

Må være obs på at data ligger lagret sekvensielt etter timestamp. Stilling av klokke (bakover) uten først å lese av samtlige poster, og resetting av minne, må ikke forekomme da det kan resultere i at data ikke lengre ligger sortert på minnet.

11.4.5 Grensesnittet mellom AVR og FPGA.

Kodefiler:

- usb_driver.c
- usb_driver.h
- avr_interface.c

USB kontroller grensesnittet mellom FPGA og Mikrokontroller benytter en master/slave arkitektur. Mikrokontrolleren er master og sender kommandoer, mens USB-kontrolleren svarer på disse forespørselene. Et unntak i denne master/slave arkitekturen er en interrupt linje som går fra USB-kontrolleren til mikrokontrolleren. Denne bruker USB-kontrolleren til å signalisere 2 hendelser:

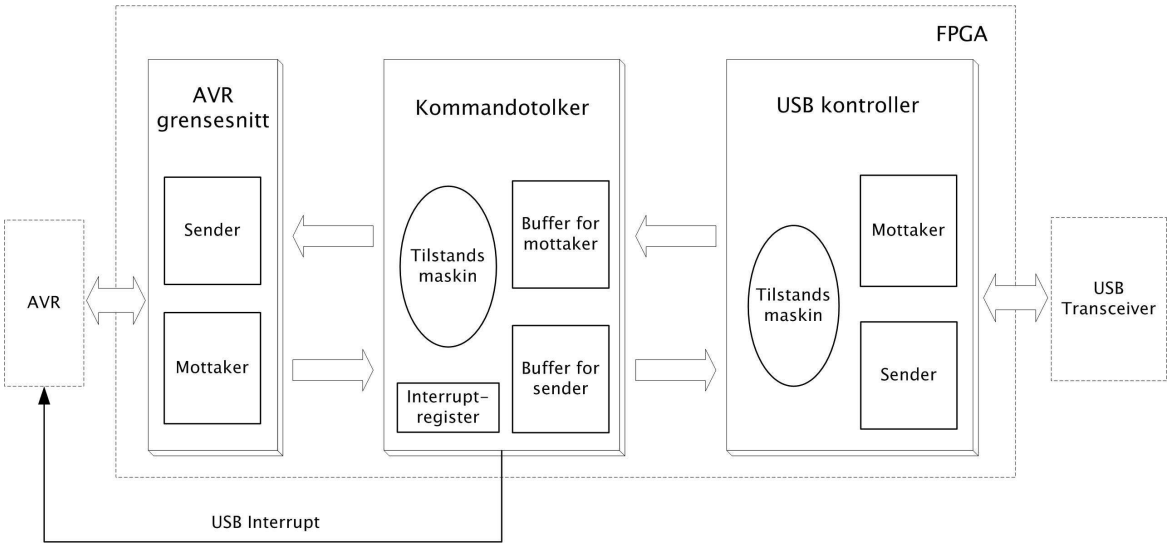
1. En USB pakke er mottatt og kan leses fra mottaks-bufferet.
2. USB pakken som ble lagt inn i send-bufferet har blitt sendt.

Etter å ha mottatt interrupt'et vil mikrokontrolleren lese interrupt registeret og ut i fra dette bestemme hva som skal skje videre. Når mikrokontrolleren leser interrupt registeret blir interrupt linja til mikrokontrolleren satt lav.

Mikrokontrolleren gir instruksjoner til USB-kontrolleren ved først å sende en kommando-byte. Etter denne kommando-byten vil det komme en eller flere lese- eller skrive-operasjoner avhengig av hvilken kommando som har blir sendt.

11.5 FPGA Design

Et overordnet blokkskjema for FPGA designet er vist i figur27. Under følger en kort forklaring av hver hoveddel i implementasjonen.



Figur 27: Toppnivå blokkdiagram for FPGA implementasjonen

AVR grensesnitt

Denne enheten tar seg av kommunikasjon med mikrokontrolleren. Den implementerer en enkel asynkron seriell bus kontroller.

Kommandotolker

Denne enheten lytter etter kommandoer fra mikrokontrolleren og utfører dem etter hvert som de kommer inn.

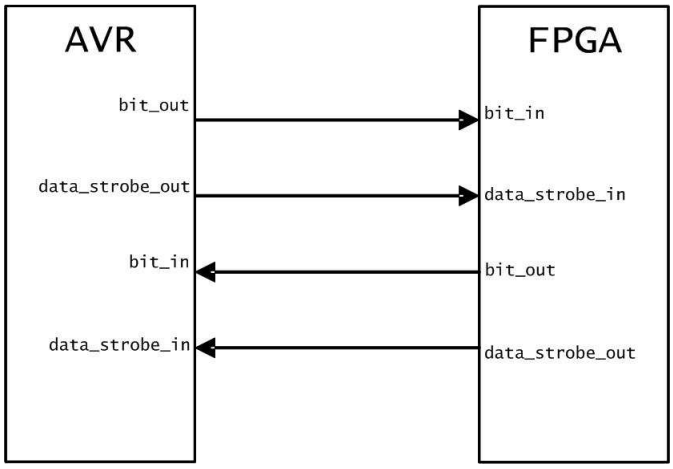
USB kontroller

Her foregår mottak og sending av USB pakker. Denne enheten tar seg blant annet av sending og mottak av bytes/bits på USB bussen og regner ut CRC.

11.5.1 AVR Grensesnitt

Kodefiler:

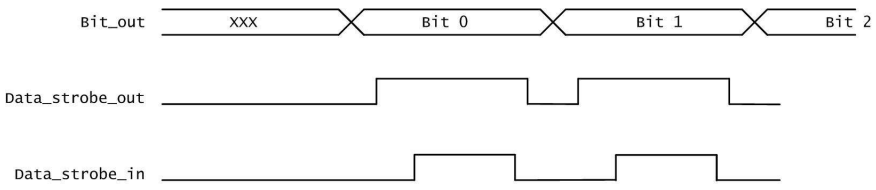
- avr_interface.vhd
- byte_receive.vhd
- byte_transmit.vhd



Figur 28: Blokkskjema over kommunikasjonen mellom FPGA og AVR

Dette er en beskrivelse av protokollen for kommunikasjon mellom AVR og FPGAen. Se figur28. Se forøvrig figur27 for et overordnet blokkskjema for FPGA designet.

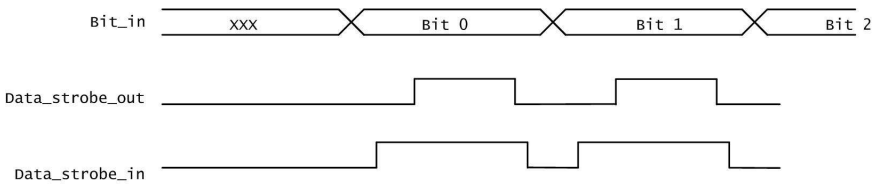
Protokollen benytter fire linjer; to for data (bit) i hver retning og to for gyldig data (data strobe) i hver retning. Protokollen baserer seg på handshaking og er uavhengig av en felles synkronisert klokke. Data sendes i sekvenser av hele bytes. Sending foregår slik (se figur29):



Figur 29: Sending av data

1. Sender legger ut data (1 bit) på bit_out.
2. Sender setter data_strobe_out høyt (gyldig data).
3. Mottaker setter data_strobe_in høyt så lenge han trenger for å lese bit'et.
4. Mottaker setter så data_strobe_in lavt.
5. Til slutt settes data_strobe_out lavt (data ikke lenger gyldig).

Denne sekvensen gjentar seg for hvert av de 8 bit'ene. Mottak foregår slik (Se figur30):



Figur 30: Mottak av data

1. Mottaker venter til data_strobe_in er satt høyt (gyldig data).
2. Mottaker setter så data_strobe_out høyt og holder denne høyt så lenge han trenger for å lese av bit'et.
3. Mottaker setter data_strobe_out lavt.
4. Sender svarer med å sette data_strobe_in lavt (data ikke lenger gyldig).
5. Sekvensen gjentar seg for hvert av de 8 bit'ene.

11.5.2 Kommandotolker

Kodefiler:

- endp_ctrl.vhd

- packet_buffer.vhd

Kommandotolkeren består i grove trekk av en tilstandsmaskin, et mottaksbuffer, et sendebuffer og et interrupt register. Oppførselen til tilstandsmaskinen er gjengitt i figur31.

Denne tilstandsmaskinen styrer kommunikasjonen med AVR Grensesnittet. En detaljert beskrivelse av hva som skjer i hver tilstand følger nedenfor. For øvrig vises det igjen til figur27 for et overordnet blokkskjema for FPGA designet.

avr_idle

I denne tilstanden venter man på at int skal gå høy, altså at man får et interrupt. Dette interruptet indikerer at byten inn er gyldig og inneholder kommandoen, cmd.

avr_wait_receive

Her skjer akkurat det samme som i avr_idle, man venter på at det skal komme inn en gyldig byte.

avr_byte_received

I denne tilstanden har man mottatt en byte (fra mikrokontrolleren).

avr_byte_written

Når man er i denne tilstanden har byten man mottok blitt skrevet til riktig adresse i bufferet og adressen kan økes med 1.

avr_address_set

Når man har kommet i denne tilstanden er adressen til mottaks-bufferet satt og byten denne adressen peker på ligger på utgangen til bufferet. Den kan dermed leses av.

avr_wait_send

Her venter kontrolleren på at enheten som kommuniserer med mikrokontrolleren, serializer/deserializer enheten, skal bli klar til motta en byte, dvs at busy linja settes lav. Hvis busy er lav legges byten som skal sendes ut slik at serializer/deserializer enheten kan motta den. En valid linje settes også høy slik at enheten på den andre siden kan detektere at byten er klar.

avr_wait_busy

I denne tilstanden venter kontrolleren på at serializer/deserializer enheten skal sette busy høy, som viser at den har mottatt dataene og begynt å sende bits til mikrokontrolleren.

avr_sending

Kontrolleren vil være i denne tilstanden inntil serializer/deserializer enheten er ferdig med å sende. Dette indikerer den med på sette busy lav.

avr_byte_sent

I denne tilstanden har byten man sender blitt levert til mikrokontrolleren.

Mottaksbufferet er et minne-element som kan lagre opptil 10 bytes. Den første byten angir størrelsen på dataene som følger, dvs. Flags og Byte 0, 1,..,n. Figur32 viser hvordan innholdet i mottaksbufferet er organisert.

Sendebufferet er organisert på tilsvarende måte. Interrupt registeret er på 1 byte benytter to bit, R og S. R bit'et er satt dersom en pakke har blitt mottatt og S bit'et er satt dersom en pakke har blitt sendt. Se figur33.

11.5.3 USB Kontroller

Kodefiler:

- controller.vhd
- ctrl_test.vhd

USB kontrolleren implementerer de laveste lagene i USB protokollen. For en bedre forståelse av hvordan USB kontrolleren fungerer sammen med de andre delene i FPGA implementasjonen, se figur27.

Kontrolleren er delt i tre hoveddeler:

- Tilstandsmaskin
- Sender
- Mottaker

Tilstandsmaskin

Tilstandsdiagrammet til tilstandsmaskinen er vist i figur34. Tilstandsmaskinen har ansvaret for å sette styresignal til og lese statussignal fra Mottaker- og Senderenheten.

En overføring på USB-bussen består av en TOKEN-pakke fulgt av en DATA-pakke og avsluttes med en HANDSHAKE-pakke. En slik sekvens av pakker finner vi igjen i tilstandsdiagrammet. For en nærmere beskrivelse se avnsittB.

Tilstand 1 til 5 - Token tilstander

Etter tilstand 5 skal den ha avklart hva slags Token-pakke som er mottatt. Oppdages det en feil før denne tilstanden avbrytes sekvensen og tilstandsmaskinen sender en NAK (Not Acknowledged) pakke til hosten. Er Token-pakken feilfri fortsetter tilstandsmaskinen basert på type Token.

Tilstand 6a til 11a - OUT og SETUP tilstander

Er mottatt Token av type OUT eller SETUP betyr det at hosten skal til å sende en DATA pakke. Etter tilstand 8a skal tilstandsmaskinen ha klarlagt om pakken er feilfri basert på statussignal fra Mottakerenheten. Er alt i orden sendes en ACK-pakke til hosten som bekreftelse på at pakken er mottatt.

Tilstand 6b til 14b - IN tilstander

Er mottatt Token derimot av type IN, forventer hosten å motta en DATA pakke fra devicen. Frem til og med tilstand 11b sendes så en DATA pakke. Til slutt, i løpet av de siste 3 tilstandene, venter devicen på å motta en ACK fra hosten som indikasjon på at pakken kom vel fram.

11.5.4 Sender

Kodefil:

- transmitter.vhd

Senderen tar i mot data fra tilstandsmaskinen og setter korrekte styresignal til USB Transceiveren. Senderen består av flere deler (se figur35):

- Parallell-til-seriell konverterer
- Bitstuffer

- NRZI koder
- CRC beregner

Det første som skjer med dataene mottatt fra tilstandsmaskinen er en serialisering. Som forklart innledningsvis i kapitlet om USB protokollen, benyttes bitstuffing for å sikre at en device på USB bussen klarer å synkronisere sin egen klokke med hosten. Det vil si at dersom det oppdages 6 etterfølgende bit av verdien '1' "stuffes" dataene med en ekstra '0'. Dermed får man en transisjon på bussen og unngår at det går for lang tid mellom hver slik endring.

For DATA-pakker må CRC[12] regnes ut. Dette er en måte å sikre at pakken kommer feilfritt frem til mottaker. Det siste som skjer før en bitstreng sendes til Transceiveren er å kode bitene i NRZI-formatet. Dette formatet er nærmere forklart innledningsvis.

11.5.5 Mottaker

Kodefil:

- receiver.vhd

Mottakeren leser signaler fra USB Transceiveren. Den består av følgende deler (se figur36):

- Seriell-til-parallell konverterer
- Bitstuff-fjerner
- NRZI dekoder
- Pakkestart detektor
- PLL (Phase-Locked-Loop)
- CRC Feilsjekk

Stort sett fungerer Mottakeren som Senderen, bare motsatt. Det første Mottakeren gjør med en bitstreng som mottas fra Transceiveren er å se etter SOP (Start-Of-Packet) eller pakkestart-sekvensen. Dette er strengen '01010101' kodet i NRZI. Med en gang denne er oppdaget settes et signal til Tilstandsmaskinen. Resten av bitstrengen oversettes fra NRZI koding til vanlig bit-mønster, se innføringen i USB protokollen for nærmere beskrivelse av hvordan denne kodingen fungerer.

Mens Sender kun trenger å tenke på feilsjekk av Data pakker, må Mottaker i tillegg kunne beregne og kontrollere at 5-bits CRC for Token-pakker er riktig.

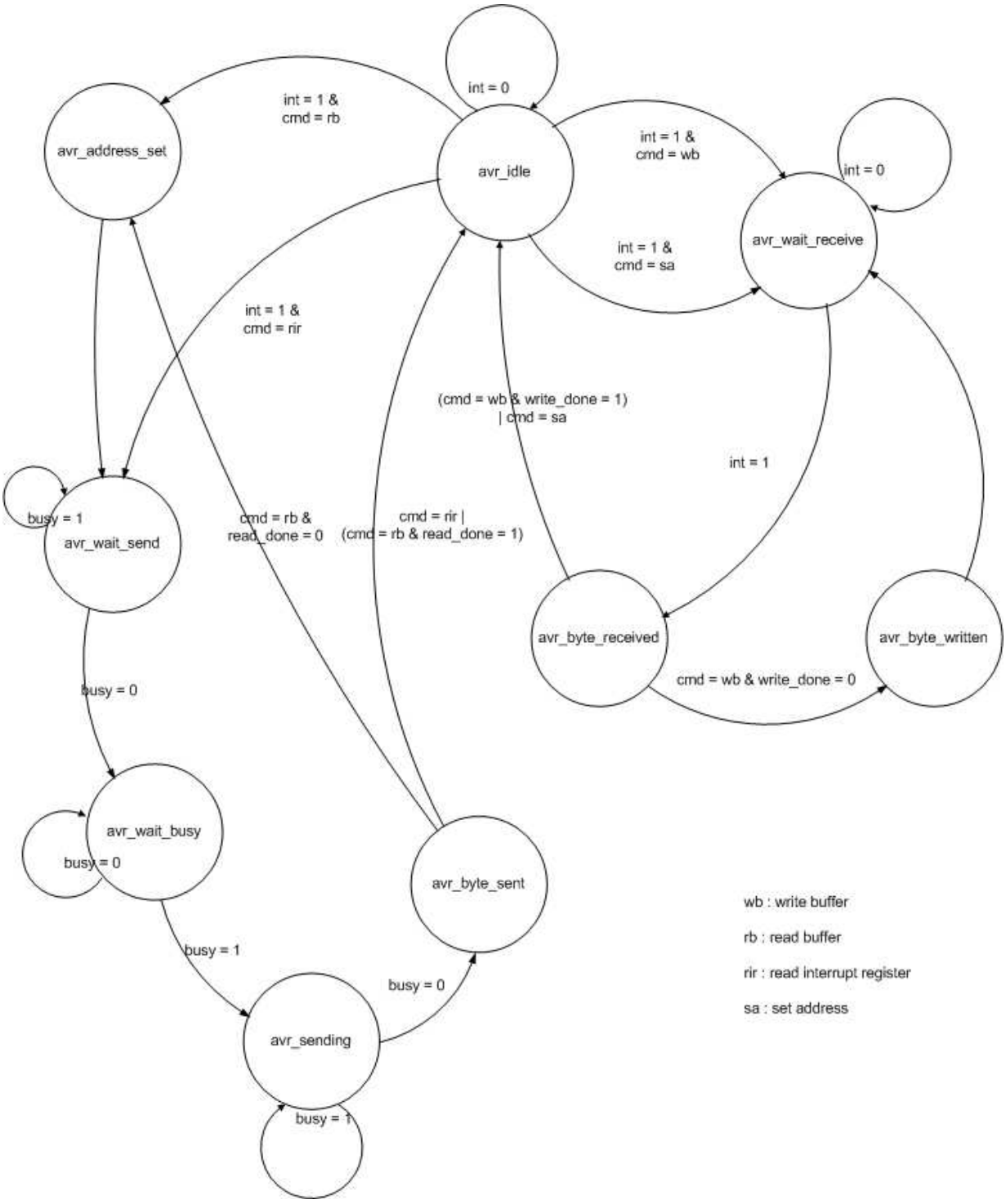
[12]Cyclic Redundance Check

PLL (Phased Locked Loop)

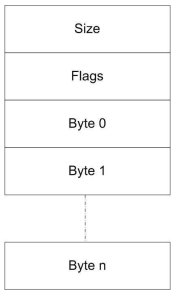
Dette er en svært viktig del av mottakeren. Internt i mottakeren befinner det seg en klokke som bestemmer når signalene fra Transceiveren er gyldige. PLL sørger for å holde denne klokka synkronisert med hosten sin klokke. Måten PLL fungerer på er at den sampler Transceiver 4 ganger for hvert bit (altså 1.5 MHz x 4 = 6 MHz). For hver 4. sample forventer den å få en transisjon. Oppdager den ikke en transisjon er det en forskyving mellom klokkene og PLL justerer sin interne klokke slik at host og device igjen er synkroniserte.

11.5.6 ProduktID og VendorID

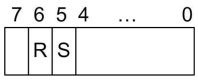
Et system som skal tilkobles USB-bussen trenger en VendorID og en ProduktID. For å få en slik av USB-gruppen må man betale lisens. Det sier seg selv at vi ikke har tenkt til å betale 15000kr for et testkort. Vi har kontrollert mot tabeller fra USB gruppen og funnet ut at VendorID 0x1337 ikke er i bruk. Deretter kan vi velge ProduktId valgfritt. Vi har her valgt 0xbabe.



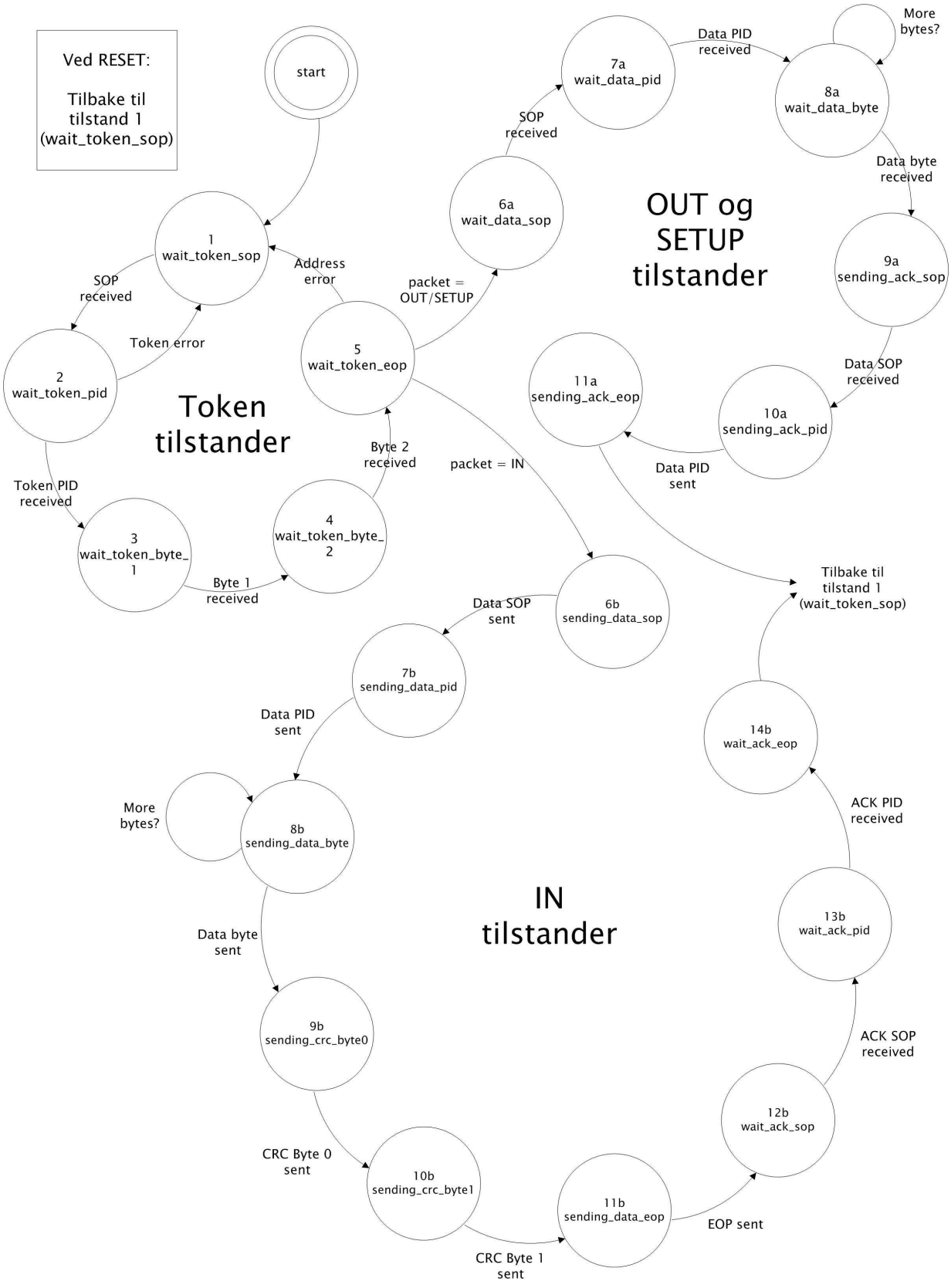
Figur 31: Tilstandsdiagram for kommandotolkeren



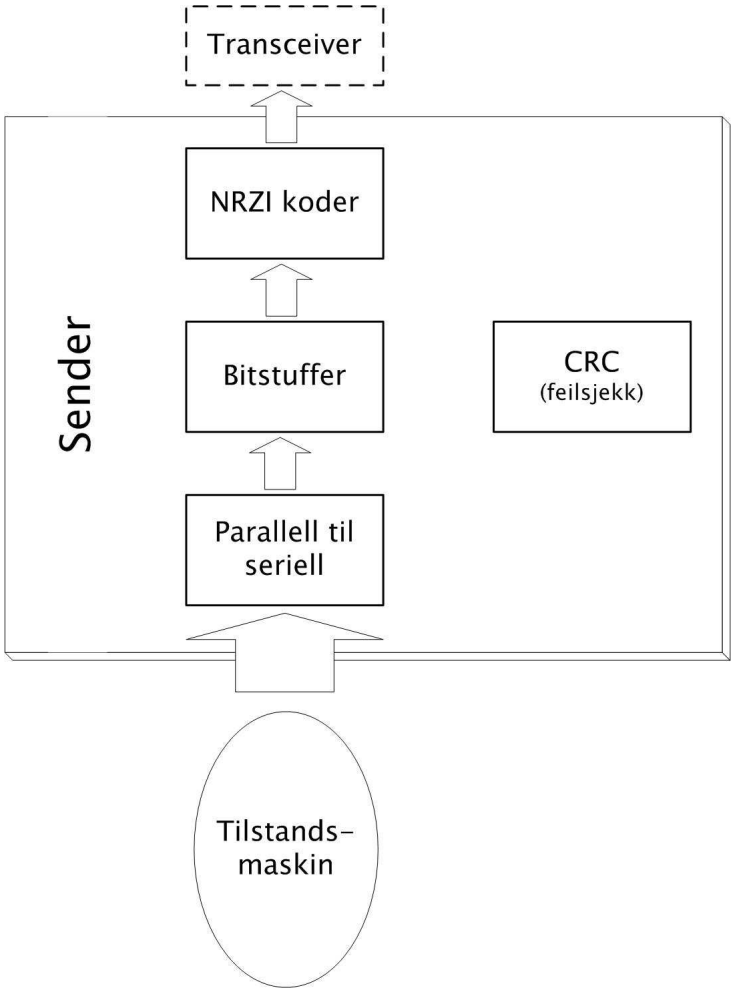
Figur 32: Recivebufferet til kommandotolkeren



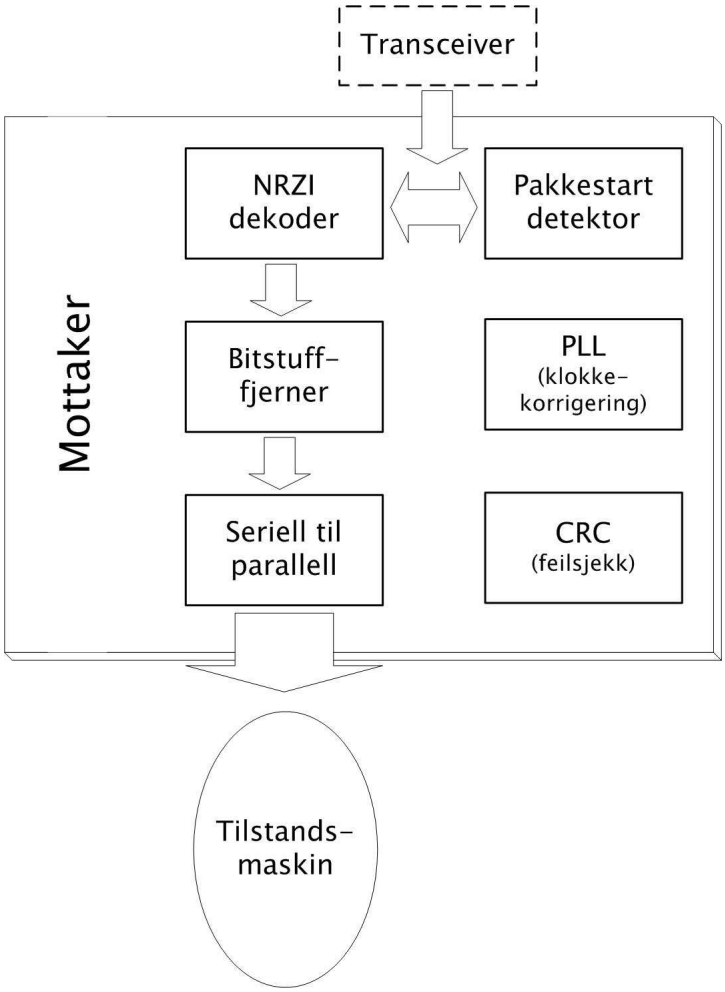
Figur 33: Interruptregisteret



Figur 34: Tilstandsmaskin for USB-kontrolleren

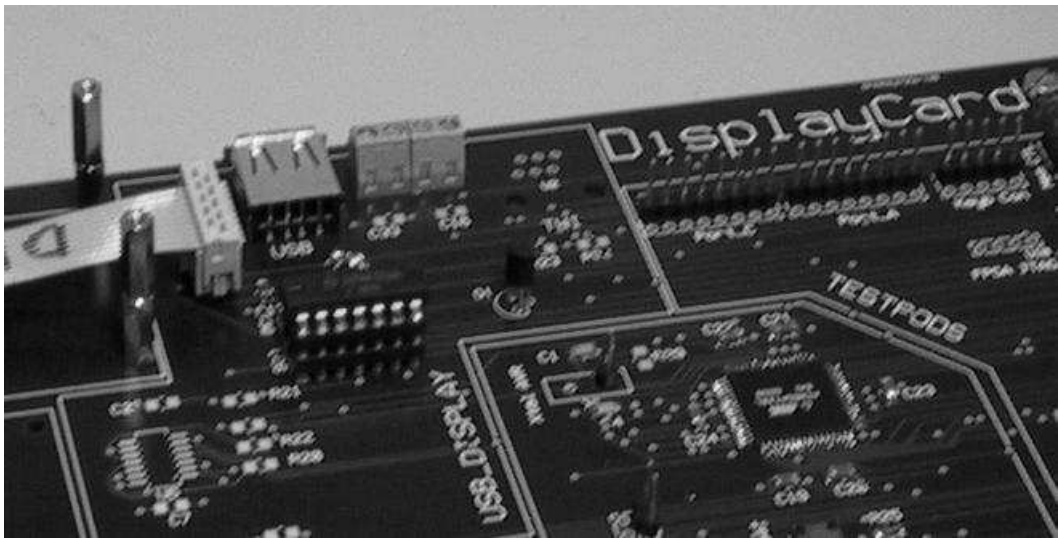


Figur 35: Blokkdiagram for USB-senderen



Figur 36: Blokkdiagram for mottakeren

12 Displaykort



12.1 Hensikt

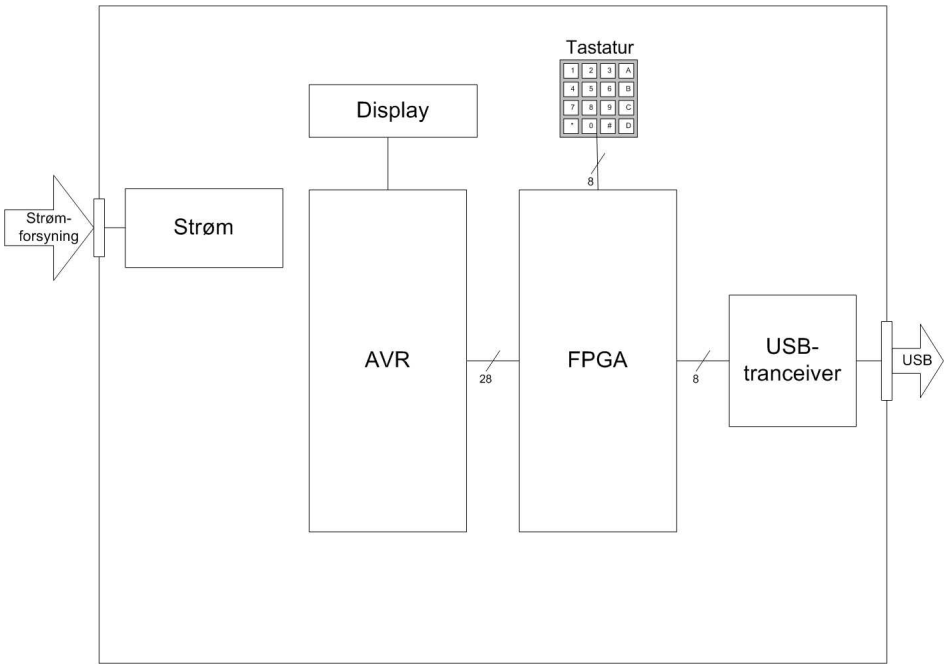
Hensikten med displaykortet er å gi brukeren av systemet et lettfattelig brukergrensesnitt mot sensorkortet. Kortet skal kunne kobles til sensorkortet over en USB-buss. Fra displaykortet skal brukeren kunne stille inn alle nødvendige innstillinger på sensorkortet, samt lese ut data fra denne. Dataene skal både kunne vises alfanumerisk og grafisk. Det skal også kunne utføre enkle statistiske funksjoner.

12.2 Hardware

Displaykortet ser overordna ut som vist i figur37. Den har en ATmega128L [1] og en FPGA av typen Xilinx Spartan IIE [3]. Mikrokontrolleren tar seg av grensesnittet mot brukeren via et enkelt display. Dersom brukeren ønsker det, kan hun også foreta enkle statistikkberegninger. I likhet med sensorkortet tar FPGAen seg av all kommunikasjon over USB. I tillegg tar den seg av dekodingen av tastaturet, og sender disse dataene videre til mikrokontrolleren.

12.2.1 Mikrokontroller

Mikrokontrolleren er kjernen i systemet. Den tar seg av all kommunikasjon med brukeren gjennom tastaturet og displayet. Displayet er koblet serielt til AVRen (Se figur38). For at den serielle kommunikasjonen skal gå så smertefritt som overhodet mulig har vi valgt å bruke en ekstern krystall. Menystruktur på displayet, visning av data, tegning av kurver og så videre er oppgaver som mikrokontrolleren tar seg av. Brukerinput skjer gjennom et 16 tasters tastatur som



Figur 37: Blokkdiagram over displaykortet

er koblet via FPGAen (se avsnitt12.2.3 for mer detaljer). Mikrokontrolleren er satt opp til å både kunne programmeres gjennom ISP [13] og med JTAG grensesnittet. Se [1] for hele databladet til mikrokontrolleren.

12.2.2 Display

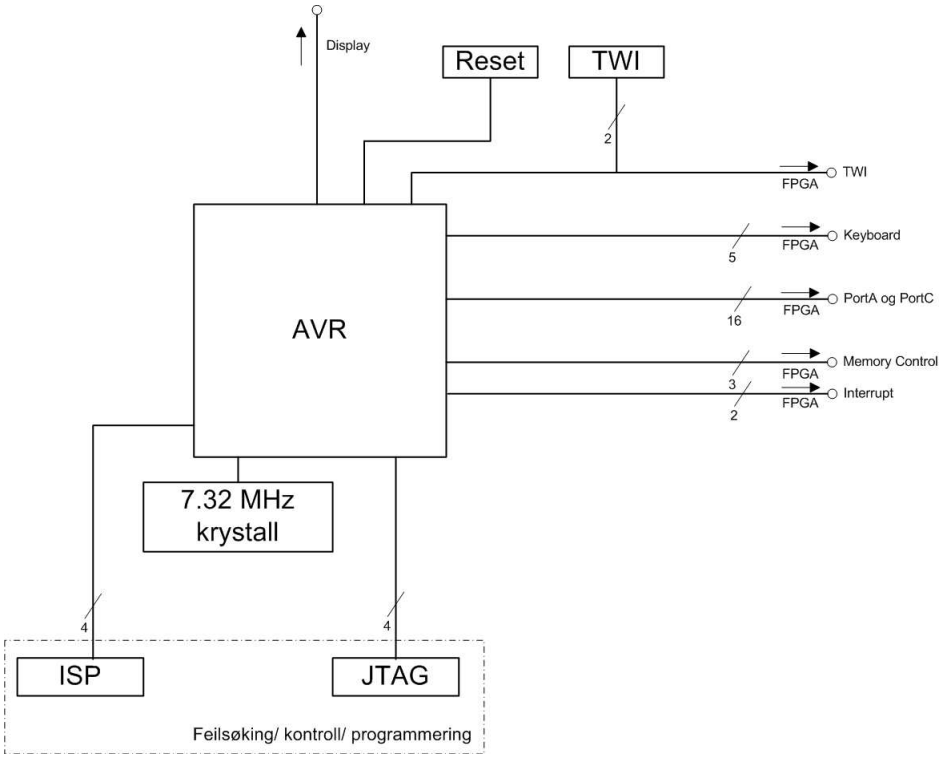
Displayet er en del av grensesnittet mot brukeren. Vi valgte et liquid crystal display (LCD) av typen"G12032 Serial Graphics LCD" [6] (For detaljer rundt denne beslutningen se avsnitt16.2.6). Dette har et enkelt serielt grensesnitt som man kan koble direkte på UARTen til AVRen. Displayet er enkelt å bruke siden det forstår ascii kodesekvenser, inkludert kontrollsekvenser som ctrl-L ("clear screen"). På denne måten kan informasjon lett genereres ved hjelp av en mikrokontroller. For å få grafikk ut på displayet innebærer det en serie med kontrollsekvenser.

Ettersom dette displayet har en driftsspenning på 5V og mikrokontrolleren har en driftsspenning på 3.3V, måtte vi oversette mellom logikknivåer (Se figur39). Dette er den samme løsningen som på sensorkortet.

12.2.3 Tastatur

Tastaturet vi benytter er et enkelt 16 tasters tastatur uten innebygd logikk. Det har kun 4 horisontale linjer og 4 vertikale linjer (Se figur 40). Om man trykker inn en tast kommer to av disse

[13] In System Programming

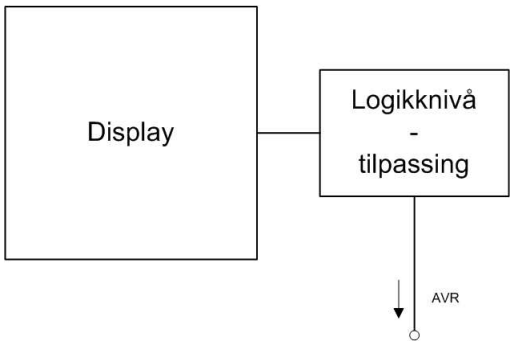


Figur 38: Blokkskjema for mikrokontrolleren

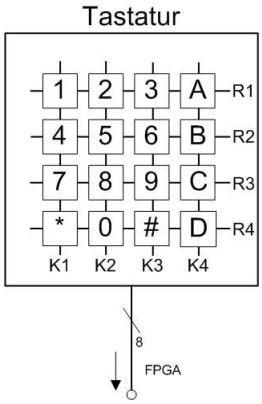
linjene i kontakt. For å finne ut hvilken tast brukeren har trykket på er man nødt til å implementere enkel logikk. Denne logikken er plassert på FPGAen. Resultatet av dekodingen sendes så videre til AVRen.

12.2.4 FPGA

FPGAen vi fikk utdelt var av typen Xilinx Spartan IIE FPGA [3]. Det er FPGAen som tar seg av all kommunikasjon med sensorkortet over USB og med tastaturet. FPGAen implementerer USB host på bussen mellom kortene og dekoder tastaturet for AVRen (se avsnitt12.2.3). For å klokke FPGAen har vi satt opp en ekstern klokke på 60 MHz (Se figur 41). Denne frekvensen er valgt for at vi lett skal kunne dele den ned ettersom USB er avhengig av riktig klokkefrekvens (6 MHz for low-speed USB). For å programmere FPGAen har vi i likhet med sensorkortet valgt å holde alle muligheter åpne. Man kan velge å programmere FPGAen enten gjennom et eget grensesnitt som er definert av Xilinx (se databladet [3]), JTAG tilkoblingen, eller bruke en PROM. PROMen vi benytter har modellbetegnelsen VX18V02.



Figur 39: Blokkskjema for displayet



Figur 40: Blokkskjema for tastaturet

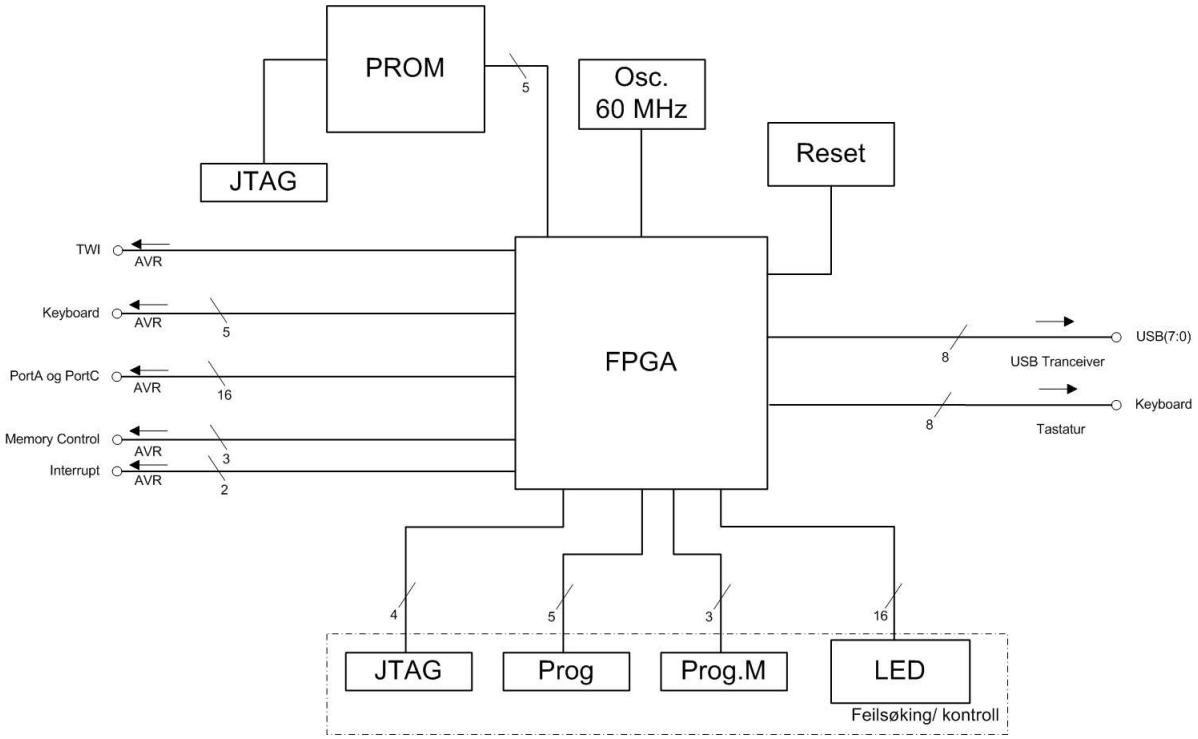
12.2.5 USB-transceiver

USB-transceiveren tar seg av alt som har med elektrisk karakteristikk på USB-bussen. Vi benytter oss av en krets som heter PDIUSB11AD [5] for å utføre denne jobben. I tillegg har vi en USB-kontakt type A i samsvar med USB-spesifikasjonen (Se figur 42). Transceiveren er koblet direkte til FPGAen.

12.3 PCB

Ved utarbeidelsen av PCB prøvde vi å gruppere alle komponenter slik at det skal være lett å se hvilke komponenter som tilhører hvilken logisk enhet i blokkskjemaet. Videre har vi lagt vekt på at det skal være mulig å montere tastaturet og displayet direkte på kortet, slik at man slipper å ha en egen boks dersom man ønsker å teste kortet.

Kortet består av 4 lag (Se figur 43). To av dem benyttes til signalruting, de to andre til jordingslag og til strøm. Alle komponentene er plassert på oversiden av kortet, altså på lag 1. Rutingen på dette laget er horisontal. Den vertikale rutingen foregår på lag 4. Lag 2, som altså er spenningslaget (se figur 44), er i tillegg til å føre hovedspenningen på 3,3V inndelt i to ekstra



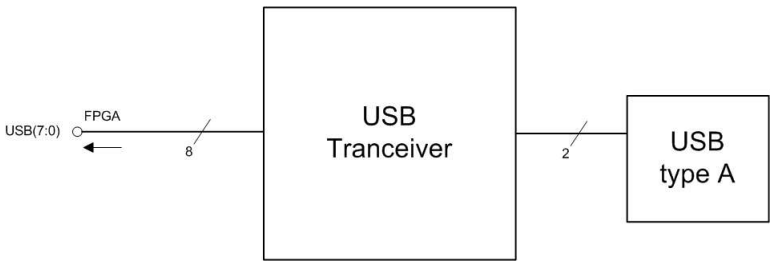
Figur 41: Blokkskjema for FPGA på displaykortet

områder med en annen spenning. Et lite område er gitt spenningen 5V, dette for å følge kravene til displayet vårt og til USB. Det andre ekstrafeltet, på 1,8V, er plassert rett under FPGAen, og er der for å gi driftspenning til den. Det er plassert slik for å gjøre det enklest mulig, med tanke på at all kommunikasjon inn og ut fra enheten foregår med 3,3V. Lag 3 er jordingslaget.

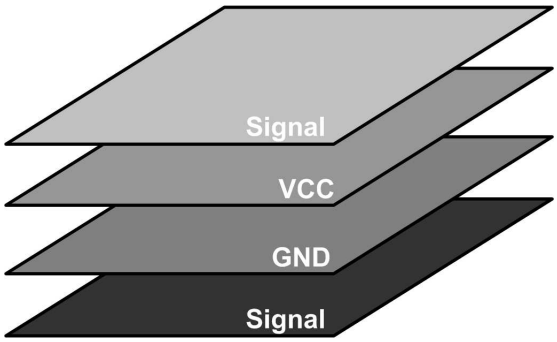
12.4 Mikroprosessor Software

12.4.1 Overordnet

Mikrokontrolleren skal kommunisere med følgende enheter:



Figur 42: Blokkskjema for USB-transceiveren



Figur 43: De fire lagene til PCBen

- USB på FPGA: Egendefinert buss
- Tastaturdekoder på FPGA: Egendefinert buss
- Display: UART

Koden er forsøkt gjort modulær, slik at enkeltkomponenter kan byttes ut uten vidtgripene endringer i resten. I designet har vi lagt vekt på at displaykortet skal være tilstandsløst, det lagrer med andre ord ikke noe som helst av sensordata, hverken avlesninger eller konfigurasjon. Det impliserer at samme displaykort kan brukes mot ulike sensorkort, og omvendt, uten at brukeren må tilpasse konfigurasjonen ved tilkobling.

12.4.2 Menysystemet

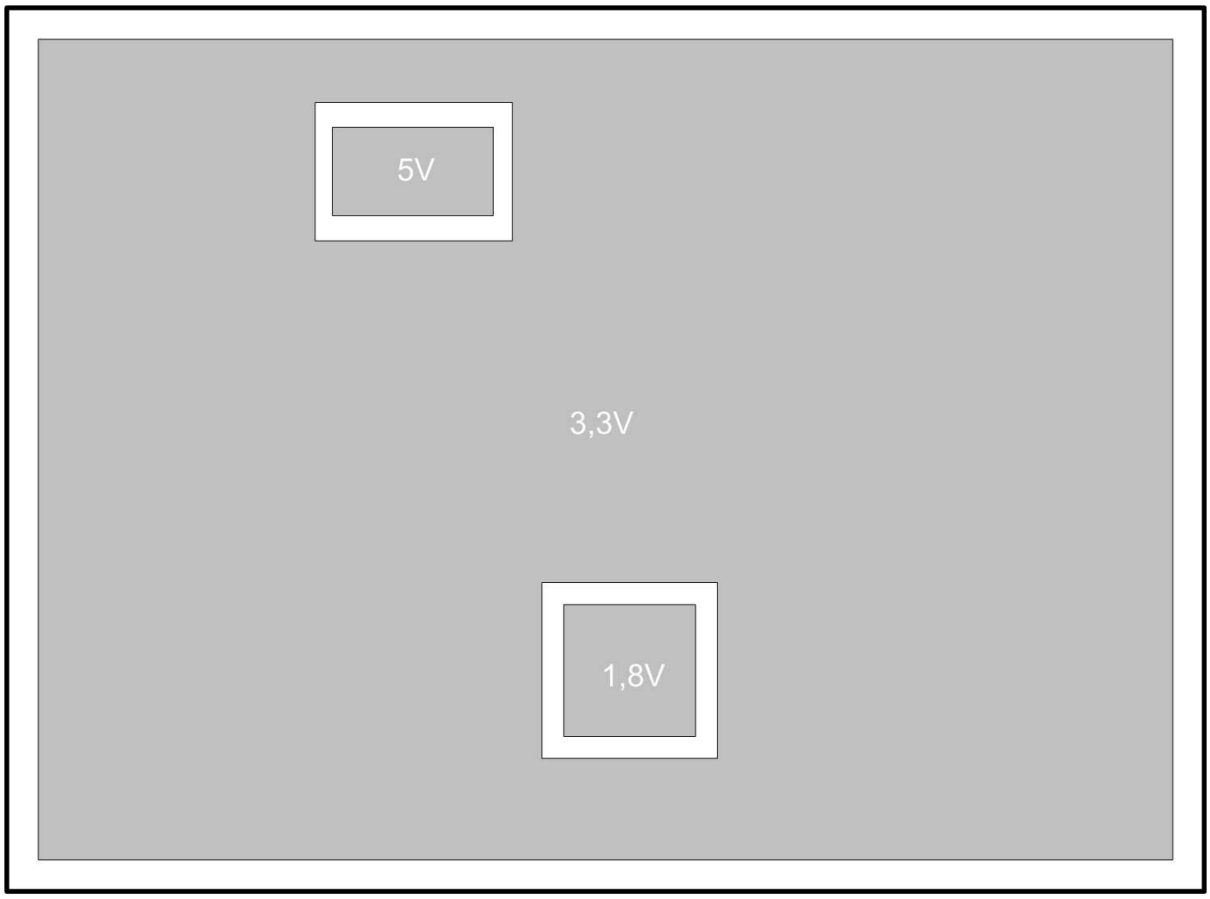
"Limet" som binder alt sammen på displaykortets AVR er et hierarkisk menysystem. Det er designet med tanke på at det skal være enkelt å legge til nye funksjoner og undermenyer. Displayet er i stand til å vise inntil 6 menyvalg av gangen. Sammen med undermenyer gjør dette det mulig å konstruere et oversiktlig brukergrensesnitt (Se figur 45).

12.4.3 Displaydriver

AVRen kommuniserer med displayet via en serieport. Tekstlig output er ganske rett frem, det er bare å sende ascii rett til displayet. Mer avansert funksjonalitet som grafikk, gjøres ved å sende kontroll-sekvenser. For å forenkle generering av grafikk er det allokert et globalt buffer som representerer hele skjermbildet til displayet. Hele bufferet kan så dumpes til displayet.

12.4.4 Tastaturdriver

FPGAen sender et interrupt til mikrokontrolleren når brukeren har trykket på en tast. Fire ledere brukes til å angi hvilken tast som er trykket. Når mikrokontrolleren mottar interruptet leses ver-



Figur 44: Fordelingen av powerlaget

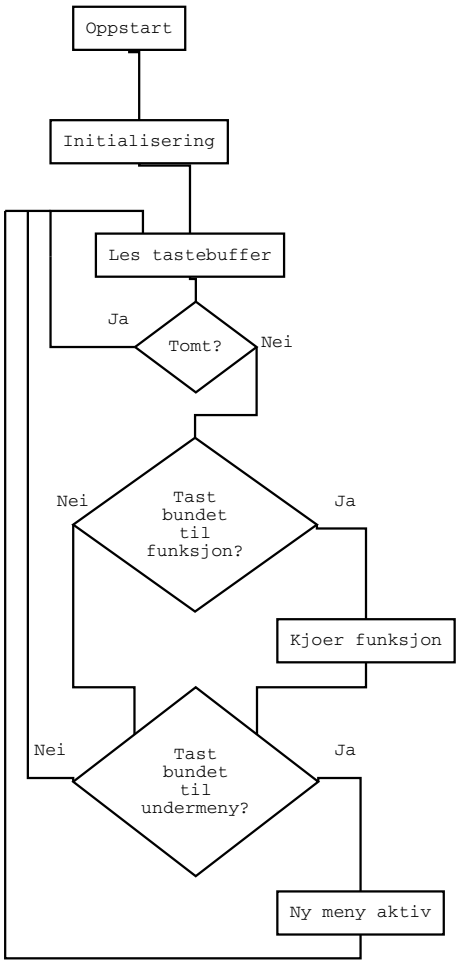
dien av disse av, og plasseres i et FIFO-buffer. Det gjør at tastetrykk ikke går tapt om brukeren gir inndata raskere enn programvaren får brukt den. Se figur 46.

12.4.5 Kommunikasjon mot USB-kontroller på FPGA

Slik vi har strukturert programvaren vil kommunikasjon kun bli initiert av displaykortet, samtidig som programvaren på displaykortet ikke vil være i stand til å fortsette før tilbakemeldingen fra sensorkortet foreligger. Enten fordi den inneholder data som er nødvendige for en beregning, eller fordi en ønsker å rapportere til brukeren om operasjonen gikk igjennom. Interrupt-linjen som var trukket opp fra FPGAen til AVRen forblir dermed ubrukt, og kan fjernes i neste versjon av kortet.

Vi forsøker ikke på å gjøre noen form for caching av data ut over skopet av et funksjonskall. Med 4KB SRAM til rådighet totalt ville en slik cache uansett ikke vært særlig effektiv.

For kommunikasjon mot sensorkortet er det definert en egen protokoll over USB (Se avsnitt15).



Figur 45: Flytdiagram for mikrokontrolleren

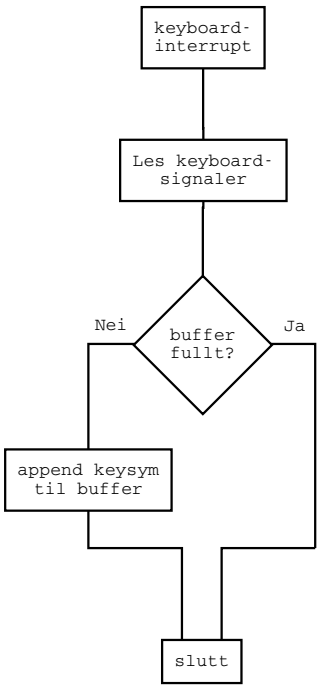
Kommunikasjonen er forsøkt isolert i egne funksjoner, slik at programvaren i hovedsak er likegyldig til om dataene ligger lokalt eller ikke. Dette muliggjør også at man kan bytte ut USB med f.eks TWI i en senere utvidelse.

12.5 FPGA Design

12.5.1 Deling av arbeid mellom FPGA og AVR

Vi har valgt en implementasjon der AVRen bestemmer når det skal sendes eller mottas data, og sørger for å legge disse ut eller motta de en byte om gangen. Dette innebærer at FPGAen bare bufrer en byte om gangen. USB-hastigheten er low speed, dvs 1.5mbit/s. AVRen kjører på omtrent 7.3MHz, som betyr at den har 39 klokkesykler på seg til å legge ut neste byte på bussen.

FPGAen står for kodingen og dekodingen i tillegg til sammensettingen av de forskjellige delene av en pakke. I tillegg grenser FPGAen mot transceiveren som implementerer det fysiske laget av



Figur 46: Flytdiagram for Tastaturavbrudd

USB-protokollen.

For å overføre data mellom USB-kontrolleren på FPGAen og mikrokontrolleren har vi valgt å definere en egen buss. Den består av 8 bidireksjonelle ledere for å overføre data i parallell, og 4 unidireksjonelle kontrolledere. To av kontrollederene brukes til å indikere et ønske om å overføre data, og to drives av FPGAen for å klokke inn data. Det hadde ikke vært noe problem å gjenbruke klokkelederen, men for å få enklest mulig kode valgte vi å ikke implementere det på prototypen.

Når AVRen ønsker å sende en pakke på USB-bussen trekker den data_out-lederen høy, og legger første byte tilgjengelig på datalederene. AVRen legger påfølgende bytes ut på synkende flanke på klokkelederen. Når FPGAen har inkommende USB-data til AVRen drar den data_in-lederenhøy, og legger første byte tilgjengelig på datalederene. Også her er det FPGAen som klokker inn lederene. AVRen leser data på stigende klokkeflanke.

12.5.2 USB Funksjonskontroller

USB funksjonskontrolleren implementerer den ikke-fysiske delen av USB-funksjonaliteten. Hovedelementene er:

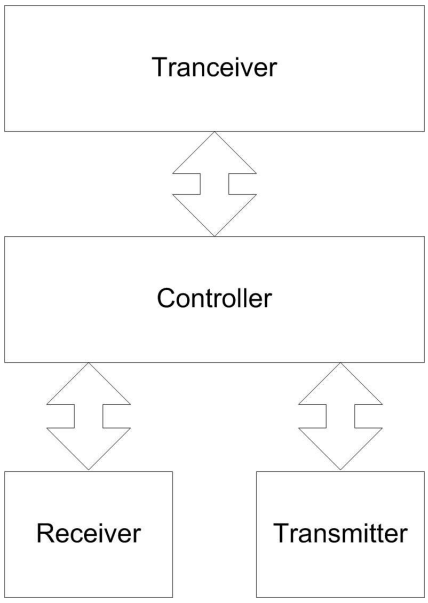
- Klokkesynkronisering
- NRZI koding og dekoding

- CRC koding og dekoding
- Bitstuffing
- PID tolking

Vår implementasjon av er laget i VHDL for en Xilinx Spartan IIE FPGA . For den fysiske delen av USB-implementasjonen brukte vi en Phillips PDIUSBP11AD transceiver.

12.5.3 Toppnivå design

Den overordnete designen av USB-implementasjonen består av komponentene controller, transmitter og receiver. Se figur 47.



Figur 47: Blokkskjema av USB-hosten

12.5.4 Controller

Kontrolleren har to hovedfunksjoner:

- Klokkegenerering
- Styring av sekvens og utsignaler

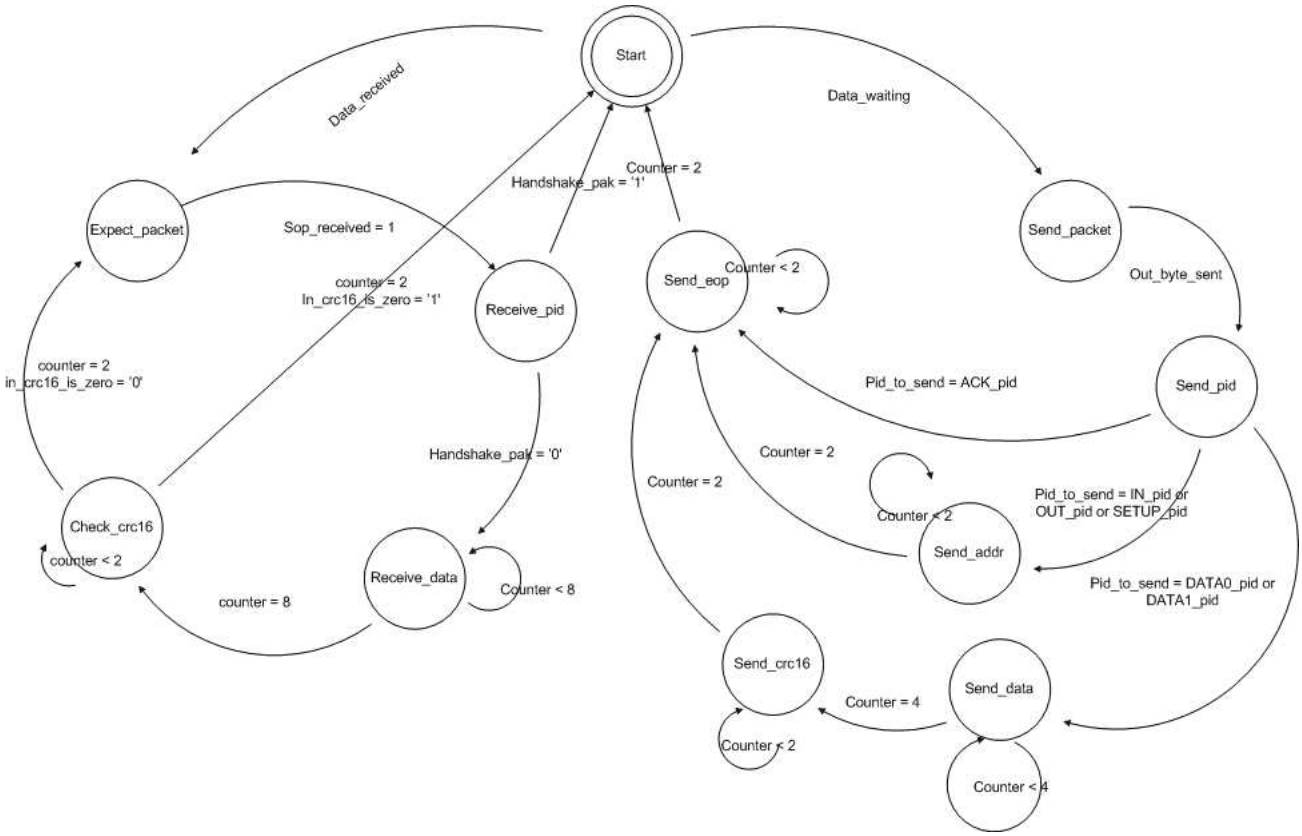
Klokkegenerering og synkronisering

Klokken som resten av prosessene kjører på, genereres av prosessen PLL. Den deler opp en ekstern klokke på 24MHz i 16. Dette gir en intern klokke på 1,5 MHz. Samtidig synkroniseres den interne klokken etter signaler som mottas over USB-bussen for å sørge for at vi er synkronisert med det devicet som sender på USB-bussen.

Styring av sekvens og utsignaler

Styringen av sekvens og utsignaler utføres av en finite state-machine (FSM). Mer spesifikt en synkron Mealy machine. Det betyr at det er en state-machine som forandrer state hver klokkesykel, og har outputs som både avhenger av hvilken state vi er i, og hvilke inputs vi har. Tilstandsdiagrammet finnes i figur 48.

Vi har også implementert en counter som brukes til å holde styr på hvor mange bytes som er sendt eller mottatt i de forskjellige statene.



Figur 48: Tilstandsdiagram for USB-kontrolleren

start_state: Denne staten resetter alle signaler. Her venter vi på signal fra AVRen om å sende eller motta en pakke, i så tilfelle går vi til respektivt send_packet eller expect_packet.

expect_packet: Her venter vi på å motta SOP (start of packet sekvens). Når den er mottatt går vi videre til receive_pid.

receive_pid: Her venter vi på å motta PID (packet ID). Dette er en kode på én byte som forteller hva slags pakke vi har fått. Når receiveren signaliserer at den har mottatt en hel byte, er veien videre avhengig av hvilken PID vi har fått: hvis det er en data PID, går vi videre til receive_data, ellers går vi tilbake til start_state.

receive_data: Her mottar vi data. Vi bruker counteren for å holde rede på hvor mange bytes vi har mottatt. For hver nye byte vi tar imot, legges de ut på datalinjene mot AVRen, og vi signaliserer at den skal lese det av ved å sette et kontrollsignal høyt. Vi forblir i samme state til vi har tatt imot 8 bytes, og går da videre til check_crc16.

check_crc16: Her venter vi til vi har mottatt de to CRC bytene, og sjekker om receiveren signaliserer at pakken er korrekt. Hvis den ikke er korrekt, signaliserer vi til AVRen at den var feil, og venter på at den skal bli sendt på nytt. En USB host sender ikke NAK signal. Hvis CRC koden stemte, går vi tilbake til start_state.

send_packet: Vi starter med å sende SOP byten, og ber i mens AVRen legge ut PID byten på datalinjene. Deretter går vi videre til send_pid.

send_pid: Når SOP byten er sendt, sender vi PID byten. Avhengig av hvilken PID vi fikk fra AVRen, går vi videre til å sende innholdet i pakken. Var det en SETUP eller TOKEN pakke, går vi videre til send_addr, var det en DATA pakke går vi til send_data, og var det en ACK pakke, går vi simpelthen tilbake til start.

send_addr: Her sender vi addressedata. Addressefeltet er hele første byten og de tre første bit i den andre byten, mens resten er en 5 bits CRC kode. Dette mottas som to bytes fra AVRen, og enkodes av transmitteren. For hver byte vi leser av, signaliserer vi til AVRen med et kontrollsignal at den skal legge ut neste byte på datalinjene. Herifra går vi til send_eop.

send_data: Her sender vi totalt 8 bytes med data. For hver byte vi leser av, signaliserer vi til AVRen med et kontrollsignal at den skal legge neste byte ut på datalinjene. Når har lest av 8 bytes, går vi videre til send_crc16.

send_crc16: Her sendes to CRC bytes basert på dataene som ble sendt i send_data. De ligger klare i transmitteren, og vi signaliserer til den at de skal sendes.

send_eop: Her sender vi et EOP signal som består av en såkalt se0 (single ended zero). Denne genereres av transceiver interfacet i transmitteren og signaliserer slutten på pakken.

For å forenkle FSM, har vi implementert kombinatorisk generering av enkelte signaler basert på outputs fra FSM.

12.5.5 Transmitter

Transmitteren sørger for å sende riktig data ut på USB-bussen, og kontrollerer signalene som går til USB-transceiveren.

Hovedprosedyre

Her skjer fire ting:

- Hver gang man begynner sendingen av en ny byte, buffres innholdet som er bestemt av multiplexeren i en variabel.
- Bitstuffing hvis det skal sendes mer enn 5 '1' verdier i sekvens, settes det inn en '0' ekstra for å sørge for at klokken hos mottaker skal ha noe å synkronisere etter.
- NRZI (non return to zero inverted) enkoding - man mapper en '0' til en transisjon i forhold til det forrige signalet som sendtes, og '1' til ikke transisjon.
- Man sjekker om man har sendt 8 ordinære bits (ikke inkludert eventuelle bit stuffed bits), og setter signalet byte_sent høyt og nullstiller telleren hvis det er tilfelle.

Transceiver Interface

Mot USB-transceiveren oversettes hver bit som skal sendes til vmo og vpo, eller se0 (single ended zero) for å signalisere eop (end of packet). Vmo og vpo er signalene transceiveren bruker for å sende data.

CRC

CRC[14] kode blir beregnet av prosessene crc_5 og crc_16. USB bruker en 5 bits CRC for setup- og token pakker, og en 16 bits CRC for data pakker. Disse genereres kontinuerlig, og blir sendt som de siste bytene i hver pakke.

12.5.6 Receiver

Receiveren sørger for dekoding av pakker som kommer inn fra USB-transceiveren.

[14]Cyclic Redundancy Check

SOP detector

Denne prosessen er en FSM som detekterer en SOP (start of packet). Dette er et synkroniseringsfelt som kommer i som første byte i hver pakke. Når prosessen har mottatt et kompelett synkroniseringsfelt, signaliserer den at man kan begynne å motta meningsfull data.

Hovedprosedyre

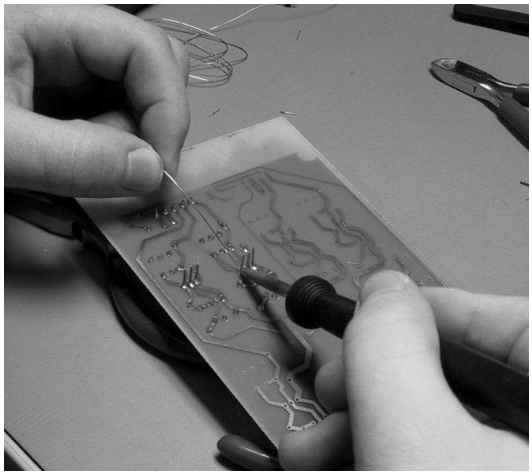
Her skjer det omvendte av hovedprosedyren i transmitteren:

- NRZI dekoding
- Bitunstuffer - fjerner bits lagt til ved bitstuffing
- De mottatte bits buffres i en vaiabel
- Hver gang man har mottatt 8 bits, settes signalet byte_received høyt, og man resetter counteren

CRC

Denne prosessen genererer kontinuerlig en 16 bits CRC (en usb host vil aldri motta setup eller token pakker, så en 5 bits CRC er ikke aktuelt). CRC har egenskapen at hvis man genererer en CRC av dataene den er generert av etterfulgt av den mottatte CRC koden selv, vil man hvis den mottatte CRC koden stemmer, få en generert CRC kode på 0.

13 Strømforsyning



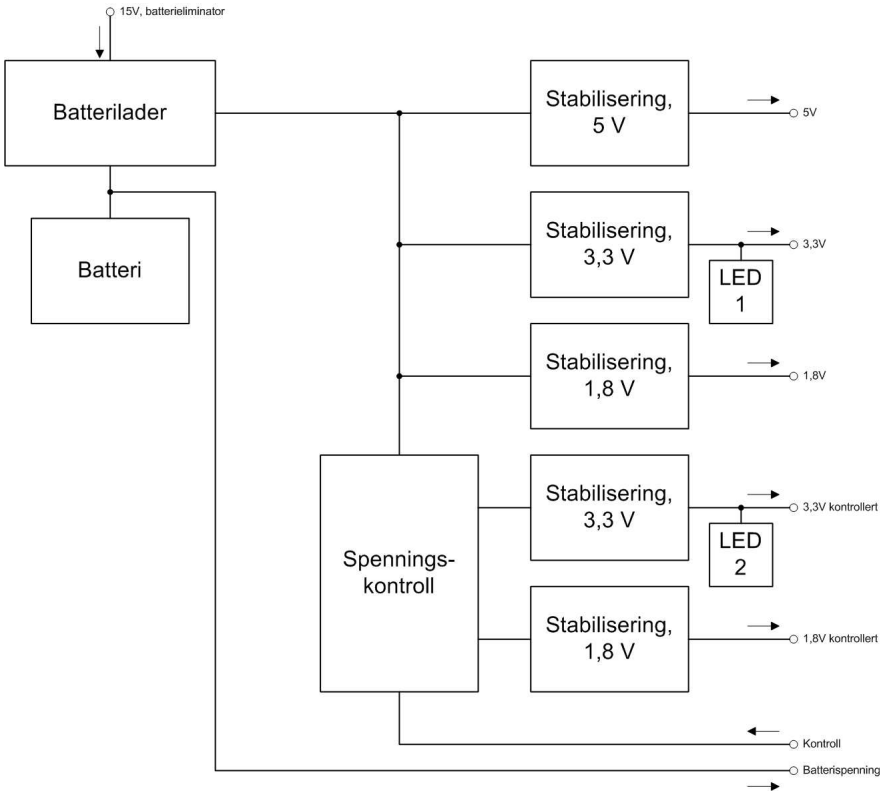
> "And, Lord, we're especially thankful for nuclear power, the cleanest, safest energy source there is, except for solar, which is just a pipe dream."
>
> – Homer Simpson
>
> Bart vs. Thanksgiving

Strømforsyningens hovedoppgave er å forsyne hovedkortene med strøm. Den skal også kunne lade batteriet som er tilkoblet.

Spenningsstabiliseringen er designet for å kunne drives både av batterieliminator og batteri. Dersom batterieliminatoren er tilkoblet skal batteriet lades samtidig som kretsene forsynes med spenning. Sensorkortet og displaykortet har noe ulike krav til spenninger og strømtrekk. Vi vil allikvel utstyre begge kortene med den samme strømforsyningen, det vil si at ikke begge kortene utnytter alle muligheter som ligger i strømforsyningsdesignet.

13.1 Virkemåte

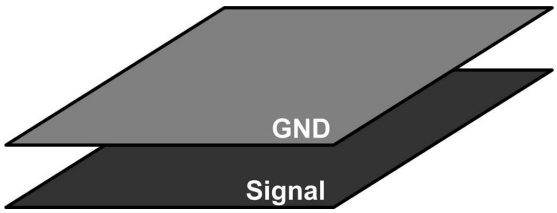
For å tilpasse spenningene til kretsene som skal drives benyttes det spenningsregulatorer av typen LM317[49]. Dette er en regulator som kan gi ut e hvilken som helst spenning. Spenningen bestemmes ved hjelp av motstander. Strømforsyningen er laget for å kunne gi ut 3 forskjellige spenningsnivåer, der to av disse skal være tilgjelige både som kontrollert (dvs. de skal kunne slås av og på) og som faste. Se figur49 for blokkskjema. Dette styrte vi ved hjelp av to halvlederreleer av typen PVA3354N[15]. For å kunne lade batteriet har vi integrert en batterilader på strømforsyningskortet. Denne er bygget opp rundt kretsen Maxim MAX713. Dette er en ladekrets for NiMH batterier. Det skal tilkobles en batteripakke på 7 celler, dvs. 8,4V, av typen NiMH. Denne kretsen har flere programmerbare funksjoner. Ladekretsen er programmert for 7 battericeller og en ladetid på 90 minutter. Se databladet [7]. For å hindre skade på tilkoblede komponenter



Figur 49: Blokkskjema for strømforsyningen

i tilfelle kortslutning er det benyttet en sikring. Sikringen ble valgt til å være på 3A da dette skal være tilstrekkelig for å forsyne alle komponenter. Kortet har to lysdioder for å vise hvilke regulatorer som er aktive, se tabellen under. Disse lysdiodene er koblet til henholdsvis +3.3V og +3.3V_kontrollert, men ettersom regulatoren har samme forsyningspenning og det benyttes felles kontrollsignal, regner vi LED'ene som indikatorer for hver gruppe.

13.2 PCB



Figur 50: PCB for strømforsyningen

PCB'en har vi laget selv på datamaskinlabben. Det har to lag (se figur50). Etter testing fant vi ut

at vi trengte ett eget lag til jord for å forsikre oss om at vi ikke fikk tap i jordbanene.[15]. Det andre laget er signallaget. De banene som skal føre mye strøm har her blitt gjort ekstra tykke.

[15]Nettopp dette var et problem med det første kortet vi lagde og vi besluttet derfor å ha tosidig kort.

14 Verktøy

> "In India, "cold weather" is merely a conventional phrase and has come into use through the necessity of having some way to distinguish between weather which will melt a brass door-knob and weather which will only make it mushy."
>
> – Mark Twain

14.1 Software

14.1.1 Mentor Design Capture/Expedition PCB

Alle våre skjemategninger ble laget i Design Capture. Dette tillater oss å designe modulært, dvs vi kan lage oss små sorte bokser der innholdet blir skjult for brukeren inntil man klikker på denne for å få tak i innholdet. Dette verktøyet støtter såkalte bibliotek med kompoenter. Vi fant fort ut at vi var nødt til å definere mesteparten av komponentene selv for å få riktig footprint osv på komponentene. Ut fra skjemategningen kunne man så generere en nettliste. Denne puttet vi så inn i Expedition PCB for pcbutlegg. Dette verktøyet tar inn nettlisten og lar deg plassere ut komponenter fysisk på en pcb. Verktøyet har en autoroutingsfunksjon som er et must med et så stort prosjekt. Hadde vi gjort dette for hånd ville det tatt flere dager. Vi fant det likevel nødvendig å route en del signaler selv, både fordi autorouteren gjorde en del"stygge" avgjørelser, men også fordi vi ønsket at et par linjer skullle være tykkere (f.eks) strømlinjene til fpga osv. Expedition PCB kan så generere Gerberfiler som kan sendes til produksjon direkte.

14.1.2 Xilinx Foundation 5.203i

Dette var hovedverktøyet for utvikling av FPGA softwaren. Den lille erfaringen vi hadde med vhdl tilsa at dette ville være det optimale valget når det gjaldt utvikling av vhdlkode. Det har direkte støtte for mange av de funksjonene vi krevde; blandt annet simulering og direkte programmering av FPGA'en. Vi hadde dog en del kryptiske problemer med dette programmet. Det hang seg en del ganger mens vi jobbet med det, men vi hadde ikke på langt nær like mange problemer som fjorårets gruppe (som brukte versjon 3.1i). Derfor beholdt vi dette verktøyet.

14.1.3 Atmel AVR Studio

Atmel AVR studio er et komplett IDE[16] for utvikling og debugging av software utviklet for Atmels linje med AVR mikrokontrollere. Det støtter hovedsaklig assemblyprogrammering, da C støtten er tredjeparts. Vi fant dette verktøyet spesielt nyttig iforbindelse med debugging av softwaren. I tillegg er det langt lettere å sette instillinger på AVR'en (f.eks om kontrolleren skal bruke intern RC til klokking eller om den skal bruke ekstern klokke) med et grafisk grensesnitt.

[16] Integrated Development Enviroment

De opensource variantene vi fant (uisp og avrdude) støttet setting av fuses, men hadde et dårligere grensesnitt.

14.1.4 GNU AVR tools (AVR GCC, AVR libc, AVR binutils, make)

Da vi skulle velge C kompilator til prosjektet falt valget fort på gcc. Dette er GNU[17] sin gratis C kompilator for AVR. Den har support for de aller fleste AVR'ene. Sammen med binutils og en liten implementasjon av standardbiblioteket (libc) utgjør dette et komplett utviklingsmiljø. For å automatisere byggeprosessen benyttet vi oss av GNU make.

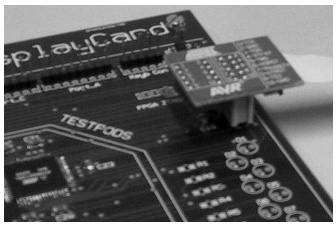
14.2 Hardware

14.2.1 Atmel STK500

Atmels STK500 kort er de facto standard når det gjelder utvikling av software til AVR. Den hadde de funksjonene vi trengte for å kunne utvikle softwaren effektivt uten tilgang til den ferdige PCB'en. På PCB'en hadde vi lagt ut pinner både for ISP og JTAG. Disse kunne kobles direkte til STK500-kortet for programmering.

14.2.2 Atmel JTAG ICE

Atmels JTAG-ICE er en løsning for å debugge AVR software. Den kobles på en egen dedikert port. Hovedformålet er å kunne debugge programmer ved å kunne inspisere AVR'ens interne registre. Den har også mulighet til å kunne programmere mikrokontrolleren.



Figur 51: JTAG i bruk på displaykortet

14.2.3 Atmel ICE 200

Atmel ICE 200 er en såkalt"in circuit emulator". Dvs det er noe vi kan putte inn i stedet for selve mikrokontrolleren. Denne vil så oppføre seg akkurat som om det var en ordentlig mikrokontroller, men tillater brukeren å inspisere det aller meste ved mikrokontrolleren under kjøring. Vi

[17] GNU's Not Unix

brukte ikke denne så altfor mye, ettersom de andre verktøyene viste seg å være tilstrekkelig for de fleste formål.

14.2.4 Memec Design Virtex2 System board

Dette kortet var arbeidshesten i vår utvikling av FPGA softwaren. Det har en Xilinx Virtex 2 FPGA innebygd og er temmelig enkelt bygd opp. Stort sett er det kun en FPGA, strømforsyning, et par knapper som man kan route til, en seriellport og to 7-segment display. Vi brukte dette kortet til all testing og utvikling av softwaren.

14.2.5 Memec Design P-160 Comm extension board

Dette kortet er et utvidelseskort til det første kortet. Det kan plugges rett inn på kortet uten problemer. På dette kortet finnes det en usbtransciever med tilhørende USB-port. Dette viste seg å være veldig nyttig under utviklingen av USB-softwaren. Problemet var at transceiveren var satt opp som en USB-device og det var ikke mulig å endre på dette til å være en USB-host (De to har forskjellige elektriske karakteristikk). Den støtter også begge hastigheter (low speed og high speed).

Figur 52: Virtex2 testkort med USBtransciever og STK500

14.2.6 On chip scope

I Xilinx ISE pakken fulgte det med et on chip scope som kunne syntetiseres inn på FPGA'en sammen med vår implementasjon, og sende data om valgte signaler over JTAG tilbake til PC'en. Dette var forholdsvis enkelt verktøy å bruke, og hadde mye av den samme funsjonaliteten som en logikkanalysator. Med det medfølgende programmet kunne vi da få opp grafer over signalene som vi hadde valgt.

14.2.7 Logikkanalysator

For å teste signalene som kom ut av transceiveren og ut på bussen, brukte vi IDI's Hewlett Packard logikkanalysator. Bruken av denne enheten er meget enkel; kabler kobles på de pinnene man ønsker å lese av signalet fra. Deretter bruker man programvaren som ligger på enheten til å lese av disse signalene og få ut grafer på lik linje med de vi fikk under simulering og med ChipScopet.

15 Applikasjons protokollen

handshaking protocol, n:

A process employed by hostile hardware devices to initate a terse but civil dialogue, which, in turn, is characterized by occasional misunderstanding, sulking, and name-calling.

– unknown

15.1 Formål

Dette er protokollen som benyttes mellom sensorkortet og displaykortet på applikasjonslaget. Den er primært ment for å kunne brukes oppå USB, men er i god stil laget for å kunne brukes oppå alternative kommunikasjonsformer, som f.eks TWI.

15.2 Dataformat

Ved å benytte fragmentering har protokollen støtte for å sende lengre meldinger enn hva underliggende lag støtter. Maksimal nyttelast er 127bytes.

15.2.1 Pakkeformat

Første pakke i en serie

Bitnr	0	1-7	8	9-15	16-63
Antall Bit	1	7	1	7	48
Navn	FørstePakke	Lengde	Svar	Type	Nyttedata

Etterfølgende pakker:

Bitnummer	0	1-7	8-63
Antall Bit	1	7	56
Navn	FørstePakke	Offset	Nyttedata

- FørstePakke - Satt hvis dette ikke er første pakke i en forespørsel
- Lengde - Dersom dette er første pakke angir dette feltet den samlede lengden på data som skal overføres.
- Svar - request/reply (request = 0)
- Type - forespørsel/svar-type (se tabell 1)
- Nyttedata - nyttedata

Merk at ikke alle pakker har et nyttelast-felt.

15.2.2 Nyttedata definisjoner

Definisjonene for de forskjellige pakketypene finnes i tabell 1.

15.2.3 Forklaring av parametre

- Bytesfree: Antall ledige bytes i flashminnet på sensorkortet.
- Flashsize: Total størrelse paa flashminnet på sensorkortet.
- SampleID: Et unikt nummer for hver vektor av samples som er lagret på sensorkortet.
- SensorID: Id-nummeret som sensorkortet har for en instans av en sensor. Dette gjør at det ikke er noe i veien for å ha flere ulike sensorer av samme type tilkoblet samme kort.
- Sensortype: Et forhåndsdefinert nummer for hver type sensor som eksisterer.
- Verdi: Verdien til en sensor på et gitt tidspunkt.
- Sekund, minutt, time, dag, måned, år beskriver tilsammen et tidspunkt. Hvert felt er på 8bits, men verdier som tilsammen ikke gir en gyldig dato gir ingen mening.

Type	Funksjon	Forespørselsparametere	Svarparametere
1	GetSampleValue	sampleid:32	verdi:16
		sensorid:8	
2	GetSampleID	sensorid:8	sampleid:32
		år:8	
		måned:8	
		dag:8	
		time:8	
		minutt:8	
		sekund:8	
3	GetSensorType	sensorid:8	sensortype:8
4	SetInterval	sekunder:32	-
5	SensorCurrentValue	sensorid:8	verid:16
6	GetTime	-	år:8
			måned:8
			dag:8
			time:8
			minutt:8
			sekund:8
7	FlushFlash	-	-
8	SetTime	år:8	-
		måned:8	
		dag:8	
		time:8	
		minutt:8	
		sekund:8	
9	MapType	sensorid:8	-
		type:8	
10	SetInterval	sekunder:32	-
11	GetInterval	-	sekunder:32
12	GetFlashFree	-	bytesfree:32
13	GetFlashSize	-	flashsize:32

Tabell 1: Nyttedata definisjoner

16 Designvalg

> "... all the good computer designs are bootlegged; the formally planned products, if they are built at all, are dogs!"
>
> – David E. Lundstrom, "A Few Good Men From Univac", MIT Press, 1987

En del av de valgene vi har gjort krever bedre forklaring. Vi vil her prøve å overbevise leseren om disse valgene.

16.1 Sensorkortet

16.1.1 Fordeling av oppgaver

Et av de tøffeste designvalget vi møtte var hvilke oppgaver FPGA'en og mikroprosessoren skulle ha. Det ble tidlig klart at det ville være fordelaktig å bruke mikroprosessoren til å håndtere minnet ettersom AVRen allerede har temmelig god støtte for dette. For å kunne spare strøm ble det da bestemt at de komponentene som ville være mest aktive måtte kobles til AVR'en. FPGA'en fikk da ansvaret for USB-tilkoblingen, fordi vi hadde masse plass ledig, og vi kunne klokke denne temmelig høyt, slik at vi kunne velge mellom high-speed og low-speed USB, etter ønske.

16.1.2 Minnebrikke

Vi trengte en eller annen måte å lagre data på, slik at man ikke mistet dem dersom man skrudde av strømmen. Valget falt da på dataflash fra Atmel. Dette valget var enkelt. Det har et godt definert interface (SPI) som kan brukes av de fleste modeller av Atmel AVR. Det er godt dokumentert, og det finnes masse eksempelkode. Alternativet hadde vært flash fra en annen fabrikant, Compact flash eller SRAM[18] med batteribackup. Compact flash ble forkastet fordi det innebar unødvendig mye kompleksistet. SRAM med batteribackup ble forkastet fordi vi ikke ville ha enda et backup-batteri, og det økte strømforbruket det ville ha innebåret. Når det gjaldt størrelse gikk vi for den største på 64Mbit, slik at vi ville ha minne nok til å drive kortet i flere år sammenhengende uten å måtte slette gammel data.

16.1.3 Strømsparing

Ettersom systemet skulle gå på batteri ble strømsparing raskt et tema. For å kunne få minst mulig strømforbruk gjorde vi en rekke designvalg. Det viktigste er at vi designet strømforsyningen slik at vi kunne skru av strømmen fysisk til FPGA'en. Dette gjør vi ved å splitte opp powerlaget i PCB'en i tre. Hovedlaget har alltid +3.3V, mens de to andre har henholdsvis +3.3V og +1.8V,

[18] Static Random Access Memory

men kun dersom det er skrudd på ved at AVR'en legger en linje høy. Tanken her er at USB-systemet vanligvis ikke er i bruk og derfor er det heller ikke noe vits i å kaste bort strøm på det. Like viktig er det at AVR'en har diverse strømsparingsmoduser. Det gjør at vi slipper å ha den aktiv når det ikke skjer noe. Vi kan da ta en måling, sove i en time, for så å ta en ny måling. Til slutt har vi brukt litt tid på å gjøre alle pullups og andre motstander så høye som forsvarlig mulig for å slippe tap i de kretsene.

16.1.4 AVR

Vi kunne velge fritt fra Atmels sortiment av AVR'er. Kravene våre var temmelig enkle. Som et minimum estimerte vi at i kom til å trenge rundt 15kbyte med programminne. Videre trengte vi SPI for å kunne snakke med minnebrikken. Denne måtte så kunne tåle å kjøre på 3.3V slik at vi slapp mange konverteringskretser for forskjellige logikknivåer. Hastighet var av mindre betydning, ettersom vi ikke hadde noen spesielt beregningsmessig intensive oppgaver å utføre med den. Vi ville dog kjøre den på ekstern krystall for å kunne garantere god seriell-kommunikasjon. Vi endte da opp med en ATmega32L. Den hadde nok minne (+ en liten sikkerhetsmargin) og kjørte på 3.3V. Grunnen til at vi ikke valgte ATmega128L var at vi nå kunne bruke In-Circuit Emulator for å gi oss enda flere avlusingsmuligheter.

16.1.5 Sanntidsklokke

For å kunne tidsstemple sensoravlesningene nøyaktig trengte vi en ekstern sanntidsklokke. I tillegg til å kunne angi tiden nøyaktig måtte brikken kunne benytte seg av batteribackup slik at en bruker slapp å stille klokken dersom strømmen ble brutt, f.eks dersom man ville bytte batteri. Valget falt på en seriell krets. Dette for å minimalisere antall pinner vi trengte på AVR'en. Kretsen vi endte opp med å velge har en avansert kalenderfunksjon som gjør at man slipper å regne ut f.eks skuddår.

16.1.6 Debugging

Ettersom dette var et prototypekort, valgte vi å gi oss selv så mange debuggingsmuligheter som overhodet mulig. På AVR'en trakk vi ut kontakt for logikkanalysator på alle de 4 portene. I tillegg la vi ut kontaktpunkt for JTAG, som nevnt i avsnitt 16.1.4 valgte vi ATMega32L slik at vi lettere kunne benytte oss av In-Circuit Emulatorer. På FPGA siden trakk vi opp linjer for JTAG her også. Dette var for å kunne debugge lettere. Ettersom de fleste av oss var temmelig uerfarne med bruk av FPGA valgte vi å ha så mange muligheter som overhodet mulig åpne. Vi la derfor ut linjer for lysdioder, brytere og logikkanalysator.

16.1.7 Konvertering mellom logikknivåer

Kommunikasjonen mellom AVRen og linedriveren til 1-wire systemet fikk vi problemer med. Dette skyldes at AVRen vår kjørte på 3.3V, mens linedriveren kjørte på 5V. Ettersom hastigheten

var såpass lav fant vi ut at det burde kunne gå med transistorer. Dette testet vi i labben og fant ut at det gikk tilfredsstillende på hastigheter opp mot 30khz. Ettersom hastigheten maks ville bli 9600kbps mot linjedriveren fant vi ut at dette lå godt innenfor trygge marginer.

16.1.8 Bussen mellom AVRen og FPGA

Da vi startet prosjektet var vi usikre på hva som ville bli mest effektivt av en I2C-kompatibel buss, eller en egendesignet buss. For å kunne ha muligheten til begge trakk vi opp linjer fra port D på mikrokontrolleren, der vi hadde valget mellom å enten bruke pinnene direkte eller bruke TWI-kjernen i AVR'en. I tillegg trakk vi opp en interruptlinje fra FPGAen i tilfelle vi kom til å trenge en slik. Vi endte til slutt opp med en egenlaget buss, mest fordi I2C kjernen var temmelig kompleks. Det finnes en del ferdigskrevet kode på f.eks opencores som vi kunne ha brukt, men vi endte opp med å la være.

16.2 Displaykortet

16.2.1 Fordeling av oppgaver

Displaykortet følger noe av det samme resonnementet som med sensorkortet. Vi lot AVR'en ta seg av alle statistikkfunksjoner ettersom dette er veldig lett å implementere på en mikroprosessor i f.eks C. I tillegg finnes det her temmelig mye eksempelkode. Videre lot vi den ta seg av kommunikasjonen med displayet, ettersom det har et veldig enkelt serielt interface klart på brikken. FPGA'en fikk også her oppgaven med å ta seg av USB'en, dette kom av et ønske om å kunne dele mest mulig kode mellom de to kortene. Dette til tross for at displaykortet implementerer en USB-host, mens sensorkortet implementerer en USB-device. I tillegg tar FPGA'en seg her av dekodingen av tastaturet, slik at det skal bli lett å bruke for AVR'en.

16.2.2 Strømforbruk

På displaykortet var ikke minimalisering av strømforbruket en stor bekymring. Dette kommer hovedsaklig av at displaykortet er ment til å skrus av og på ettersom man har bruk for det. Derfor vil det heller ikke være noen krise å måtte bytte batteri nå og da.

16.2.3 Komponentvalg

Ved valg av komponenter har flere hensyn spilt inn. Strømforbruk, hvor enkle de er å bruke, leveringstid og kompatibiltet med resten av komponentene var faktorer vi tok hensyn til i valget. Navn og nummer på FPGA var bestemt av typen vi fikk utdelt, og dermed ikke et valg vi hadde noe med. Vi fikk også mikrokontrolleren gratis, men her var valget vårt om hvilken type vi skulle bruke. De andre enhetene måtte kjøpes inn, og dermed kom pris og budsjett også med i vurderingen av hva og hvor mange vi skulle benytte.

16.2.4 Plassering av komponenter

Kortet som er laget er ment å være en prototype, og plassering av komponenter bærer også preg av dette. Kortet er designet med mange alternative løsninger for både kommunikasjon, programmering og debugging. Lederene mellom våre to hovedenheter, mikrokontrolleren og FPGAen, går en stor omvei innom testområdet vårt. Disse kunne selvsagt ha vært plassert mer midt i mellom, men vi valgte i stedet å samle alle komponentene som kun har en funksjon under testing på samme sted. Ellers er enheter som fysisk er nært knyttet sammen prøvd samlet på kortet.

16.2.5 Valg av mikrokontroller

Ettersom displaykortet skulle ta seg av mesteparten av utregningene, sammen med grafikk og menyer, regnet vi med at vi kom til å bruke temmelig mye minne på dette. For å kunne f.eks tilby forskjellige språk i menyene var det nødvendig å ha plass nok til å ha tekststrenger på flere forskjellige språk inne i flashminnet. Dette kravet om mye minne drev vår beslutning mot Atmega128L. Vi var klar over at 128kbyte var urealistisk mye, men siden dette er en prototype valgte vi å gå for variant der vi i alle fall ikke fikk problemer med med å ha for lite minne. Siden C-kode skrevet for en AVR-modell kan overføres til en annen med bare mindre justeringer kan man enkelt nedskalere designet til en billigere brikke dersom prototypen skal videreutvikles til et kommersielt produkt. En nedskalering til ATmega64L vil være ganske triviell, mens en nedskalering til ATmega32L vil kreve en del optimalisering av koden til å bruke mindre SRAM. FPGA'en har i normal modus et logikknivå på 3.3V. Den har et modus for å bruke 5V, men dette må sannsynligvis skrus på på alle pinnene om den skal brukes. Vi vurderte det som enklere å velge en 'L'-versjon av AVRen, og kjøre denne på 3.3V, enn å kjøre AVRen på 5.0V og konvertere logikknivået på bussene imellom.

16.2.6 Display

Vi valgte et serielt display fra Scott Edwards Electronics kalt 'G12032'. Vi valgte dette displayet fordi det hadde støtte for både tekst og grafikk. Programmeringsinterfacet er veldig enkelt og kan kobles direkte til AVR'en, forutsatt at man bruker en inverter imellom. Deretter er det mer eller mindre å sende ascii til det for å få ut tekst, mens grafikk er noe vanskeligere med escape-sekvenser og kommandoord. Flere ulike display med seriell kommunikasjon, tekst og grafikkinstruksjoner ble vurdert, deriblant flere med større oppløsning. Disse ble imidlertid forkastet da de fleste krevde enten ganske høye inn-spenninger for drift, eller et sett ulike spenninger, gjerne med ulik polaritet i forhold til jord. Da vi allerede hadde komponenter som krevde flere ulike spenninger (1.8V, 3.3V og 5V) var det avgjørende at displayet nøyde seg med kun +5V inn.

16.2.7 Klokkefrekvensen på AVRen

Baudrate på UARTene på AVR bestemmes ved at man foretar en relativt grov ned-deling av hovedklokken på AVRen ved å sette en verdi i et register. For at to UARTer skal være i stand til

å kommunisere må avviket i faktisk baud-rate mellom kommunikasjonspartnerene være lite nok til at man klokker inn siste databit i en byte korrekt. Da mulighetene for å justere baudraten til displayet er svært begrenset, valgte vi en klokkefrekvensfor AVRen som kan gi eksakt 9600bps. Valget falt på 7.3728Mhz.

16.2.8 Brukergrensesnittet

Vi valgte å implementere et tekstlig brukergrensesnitt på displayet. Det ble foreslått å bruke et grafisk grensesnitt, men gitt størrelsen på displayet, begrenset minne til å lagre grafikk, og tid til rådighet, fant vi det best å la være. Det kan imidlertid være en mulig utvidelse i en senere utgave.

16.2.9 Tastatur

Tastaturet vi valgte er noe av det mest primitive man kunne komme unna med. For å håndtere dette må man enten scanne aktivt fra AVRen, eller implementere kretslogikk som finner krysningspunktet for tastetrykk, og sender dette videre til AVRen sammen med et interrupt. Da den siste metoden både vil gi langt enklere kode på AVRen, og raskere respons på tastetrykk, valgte vi å implementere den mellomliggende logikken på FPGA'en. Alternativene ville i praksis enten være å scanne for tasteinput mange steder i koden, eller implementere tråding.

16.2.10 Bussen mellom AVR og FPGA

Ettersom displaykortet endte opp med å bruke en ATmega128L, så vi ikke noe poeng i å rasjonere på hvor mange ledere som var trukket opp mellom AVR og FPGA. Muligheten for flere 3 ulike kommunikasjonsmåter ble holdt åpne ved at minnebussen og TWI, samt interruptlinje, ble trukket opp mellom AVR og FPGA. Vi ønsket primært å forsøke å implementere en USB-kontroller som ovenfor AVRen oppførte seg som en minnebrikke (memory mapped I/O). Dette ville gjort grensesnittet fra AVR-siden meget enkelt. Implikasjonen var imidlertid at implementasjonen på FPGAen ville bli desto mer kompleks, og vi fant det derfor greiest å bruke portene på minnebussen som generelle IO-pinner.

16.2.11 Protokollen mellom kortene

Det ble sett på muligheten for at sensorkortet kunne oppføre seg som en USB memory stick, med et enkelt filsystem. Dette ville gjort det enklere å hente ut data fra sensorkortet til en PC. Dersom dette ble brukt som kommunikasjonsprotokoll mellom kortene ville det gjort det enklere å teste begge kortene enkeltvis, og støtte for at displaykortet kunne ta backup av data fra et sensorkort til en memory-stick ville vært enkelt å implementere. Vi kom imidlertid til at dette medførte en unødvendig stor overhead, og ville ta såpass mye tid å implementere at gevinsten ved økte testmuligheter sannsynligvis var spist opp.

16.3 Strømforsyningen

Strømforsyningen ble laget som en separat PCB. Dette skyldes delvis at vi ikke var ferdig med designet når vi hovedkortene skulle produseres og delvis at vi ønsket å kunne teste mer før vi bestemte oss for designet. Alle tilkoblinger skal være for kabling på 1,5mm2. Dette for å garantere liten resistans i kablingen mellom kortene. Sensorkortet og displaykortet har noe ulike krav til spenninger og strømtrekk. Vi har allikevel bestemt at strømforsyningskortet skal ha samme PCB utlegg. Dette for å forenkle design og produksjon.

16.4 Lyssensor

Vi valgte komponenten TSL250R fremfor en enkle LDR[19] fordi vi på den måten fikk en enklere styring av kretsen. Prisen på komponenten forsvarer også dette valget. Grunnen til at vi valgte å benytte opamper for tilpassing fremfor å la TSL250R styre utgangen direkte skyldes delvis at vi ønsket impedanstilpassing for å unngå støy og delvis at vi ønsket å tilpasse spenningsnivået slik at vi kunne utnytte AD-konverterens konverteringsområde bedre. Vi valgte å legge opampene på kortet sammen med TSL250R, ikke på hovedkortet, utelukkende for å begrense støy på kabelen. For å oppnå riktig tilpassning har vi montert et analogt buffer på inngangen på hovedkortet.

[19] Light Dependent Resistor

17 Testing

Vi har lagt vekt på å gjøre systemet testbart i hele prosessen, helt fra komponentvalg og design til implementasjon. Grunnet tidspress fra vi fikk PCB'ene tilbake fra produksjon frem til avsluttning måtte vi kutte noe ned på hvilke tester som kunne utføres og hvilke av disse som skulle dokumenteres.

Det har også foregått en kontinuerlig prosess med testing parallelt med utviklingen av så vel maskinvare som programvare. Mye av den jobben som ble lagt ned her er ikke dokumentert, men har ført til at vi har fått en mengde subkomponenter til å fungere som ønsket uavhengig av status for hele systemet. I denne prosessen ble det også utviklet flere PCB'er og testkoblinger, disse er nærmere dokumentert i kap. 17.9.

Grunnet feil i PCB utlegget har vi ingen FPGA som fungerer på kortene vi skulle teste. Vi har derfor besluttet og teste VHDL-koden på testkort fra Memec (Nærmere beskrevet i kap 14.2.4). Dermed vil ikke tester som omfatter FPGA kunne foretas på PCBen som er en del av systemet, siden testene er tilpasset det aktuelle testkortet.

17.1 Test av systemet

For hver test som er gjennomført står det oppfør en rubrikk for "Resultat/ feilnr.". Dersom testen feiler vil det her stå ett "feilnr." som er videre forklart i kap. 17.8. Her finnes det en beskrivelse av feilen og hva som er gjort for å utbedre denne dersom dette har vært mulig.

For tester som krever spesielle testprogram skal disse programmene skrives før testen starter. Vi regner tester av subsystemer kun som tester for intern bruk og det er dermed ikke lagt vekt på dokumentasjonene av disse. Det er ingen krav til at disse programmene er tilgjengelig når testen er avsluttet.

17.1.1 Nummerering av tester

Testene er nummerert etter følgende format X-n. Der n er et løpenummer fra 1 og oppover og X har betydning gitt i tabell 2.

X	Delsystem
0	Hele systemet
1	Sensorkort
2	Displaykort
3	Strømforsyningskort
4	Lyssensor
5	PCB

Tabell 2: Delsystemer

17.1.2 Nummering av feil

Feilnummer er laget ut fra følgende system XF-n hvor X er som listet i tabell 20 og n er et løpenummer.

17.2 PCB

Systemet består av flere enkeltstående PCBer. I tabell 4 er det beskrevet hvordan vi testet og utfallet av disse testene for hver enkelt av PCB'ene.

Test nr.	Beskrivelse av test	Forventet resultat	Resultat/ feil nr.		
			Sensorkort	Displaykort	Strømfo
5-1	Kontroller kortet for åpenbare feil og mangler	Alle silketrykk og footprint er som forventet	OK	OK	OK
5-3	Kontroller at alle komponenter er med på PCB	Alle komponenter som er tegnet på skjemategninger er også på PCB	OK	OK	OK
5-4	Mål for kortsluttning mellom VCC og GND, mål også mellom alle ulike VCC	Ingen kortsluttning	OK	5F-1	OK
5-5	Monter alle komponenter og mål for kortsluttning etter hver montering	Ingen kortsluttning	5F-2	5F-3	OK
5-6	Koble til strømforsyningskort (test av strømforsyningskort må gjøres først) og mål driftsspenninger på alle komponenter som skal ha dette	Spenninger stemmer med datablader og skjemategninger	OK	5F-4	OK

Tabell 4: Test av PCB

17.3 Sensorkort

Alle tester i dette delkapitellet skal utføres med spenning på systemet.

17.3.1 Mikrokontroller

I tabell 6 er våre tester av mikrokontrolleren og dens undersystemer. Test av mikrokontroller mot FPGA er listet i egen tabell (Se kap. 17.3.3).

17.3.2 FPGA

I tabell 8 er det listet opp tester som kun gjelder FPGA-designet for sensorkortet. Det er verdt å merke seg at FPGA'en kun implementerer kommunikasjon og dermed ikke har noen funksjonalitet uten å være tilkoblet mikrokontrolleren. Vi har allikevel valgt å teste denne som en frittstående enhet. Kommunikasjonsdelen sammen med AVR er dokumenter i kap. 17.3.3.

17.3.3 Mikrokontroller og FPGA

I tabell 9er det listet tester for kommunikasjon mellom mikrokontroller og FPGA på sensorkortet.

17.4 Displaykort

Alle tester i dette kapitelet skal utføres med spenning på systemet.

17.4.1 Mikrokontroller

I tabell 11tester av mikrokontrolleren og dens undersystemer. Test av mikrokontroller mot FPGA er listet i egne tabell (Se kap. 17.4.3)

17.4.2 FPGA

I tabell 13er det listet opp tester som kun gjelder FPGAen på displaykortet. Kommunikasjonsdelen sammen med AVR er testet i kap. 17.4.3.

17.4.3 Mikrokontroller og FPGA

I tabell 53 er det listet tester som tester kommunikasjon mellom mikrokontroller og FPGA.

17.5 Strømforsyning

Strømforsyningen er testet før alle andre delsystmer. Dette gjør at vi kan utelukke feil på strømforsyningen dersom andre enheter ikke skulle gi ønsket respons. Testene er beskreevet i tabell 15.

17.6 Lyssensor

For at vi skulle kunne sikre at evt. problemer med anlalogsensoren ikke hadde med dette kortet å gjøre ble denne testen utført før vi testet kortet sammen med sensorkortet. Se tabell 16.

Test nr.	Beskrivelse av test	Forventet resultat	Resultat/ Feil nr.
6-1	Teste sending av data-pakke vha. testbench på AVR og ChipScop	Korrekt utformet waveform	
6-2	Teste sending av setup-pakke vha. testbench på AVR og ChipScope	Korrekt utformet waveform	
6-3	Teste mottak av ACK/NAK-pakke vha. sensorkort USB og ChipScope	Korrekt utformet waveform	Ikke gjennomført
6-4	Teste mottak av data-pakke vha. sensorkort USB og ChipScope	Korrekt utformet waveform	Ikke gjennomført
6-5	Teste sending av data-pakke vha. testbench på AVR og logikkanalysator	Korrekt utformet waveform	OK
6-6	Teste sending av setup-pakke vha. testbench på AVR og logikkanalysato	Korrekt utformet waverform	OK
6-7	Teste mottak av data-pakke på AVR vha. sensorkort USB	Korrekte verdier på display	Ikke gjennomført
6-8	Teste mottak av ACK/NAK på AVR vha. sensorkort USB og logikkanalysator	Ingen output på display og korrekt utformet waveform på logikkanalysatoren	Ikke gjennomført
6-9	Trykk en tast på tastaturet og register hvor vidt denne tasten funger	Forventet reaksjon	2F-1

Figur 53: Kommunikasjon mellom FPGA og AVR

17.7 Systemtest

Det er utarbeidet tester for hele systemet. Siden vi ikke fikk tid til å implementere kommunikasjon ble ikke disse testene utført, men de er alikevel listet opp i etterfølgende tabell 18.

17.8 Tester som feilet

I tabell 20 alle tester som feilet med beskrivelse av feil og hva som er/ kunne vært gjort for å rette opp feilen samt konsekvenser det fikk for videre test av systemet.

17.9 Testkort

Vi har utviklet flere testkort for kunne verifisere kode og design underveis i utviklingen. Disse kortene har hver for seg blitt brukt til å teste kodefragmenter og designbeslutninger. Kortene har ikke vært benyttet for å teste mer sammensatte systemer. Testingen av sammensatte systemer er gjort vha. testene som er beskrevet i de foregående kapitlene.

Kortene vi har laget har vært for testing av:

- Sanntidsklokke
- Flashminne (1Mbit versjon)[20]

17.9.1 Sanntidsklokke

Dette kortet ble laget for å kunne teste ut programvare skrevet for å kommunisere med RTC (Real Time Clock). Kortet består av kretsen DS1302[10] som skulle testes. For at denne komponenten skal fungere må den han en oscillator på riktig frekvens, denne har vi også inkludert på kortet. RTC kretsen har også blitt utstyrt med en avkoblingskondensator. For at kortet skal være enkelt å bruke, og slik at det skal oppføre seg mest mulig likt kretsen på sensorkortet, har vi benyttet samme kobling mot AVR. Dvs. at vi har lagt alle pinner som skal brukes på RTC ut til en 2 ganger 6 pinners koplingslist. Det er her satt av 8 pinner til data og 2 til VCC og GND som forsynes fra STK500 under test. Kortet kan på denne måten kobles til hvilken som helst port på STK500 via en standard flatkabel. Kortet har også to tilkoblinger for backupbatteri, dette er en funksjon som gjøre at klokken ikke resettes selv om strømmen blir borte. Skjema og bilde av kortet finnes i figur 108 ivedlegget. Kortet har vært helt avgjørende i utviklingen av programvare for denne modulen.

17.9.2 Minnebrikke

Kortet for å teste minnebrikken består kun av minnebrikken og en tilkoblingslist tilsvarende den som er benyttet på sanntidsklokken. Dette gjør at vi kan benytte en flatkabel mellom STK500 og testkortet, gjennom denne forsynes også testkortet med strøm. Vi har også utstyrt kortet med mulighet for å lodde på termineringsblokk slik at vi skulle kunne koble til en annen strømforsyning om det skulle bli nødvendig. Denne muligheten ønsket vi å ha åpen slik at vi kunne koble testkortet til sensorkortet om det skulle vise seg at vi ikke fikk riktig minnebrikke i tide. Skjema og bilde av dette kortet finnes i figur 110. Dette kortet ble brukt til å debugge og tilpasse koden som var skrevet for minnehåndtering. Testkortet gjorde at vi kunne garantere at minneprogramvaren var korrekt før vi satte delen sammen

17.9.3 Linedriver

Vi laget også et kort får å teste linedriveren som brukes sammen med OneWire systemet. Kortet består av linedriveren, tilkoblingsrekke mot STK500, RJ11-plugg og avkoblingskondensator. Pinnen som brukes til å koble mot STK500 er laget slik at kobler mot UART porten på mikrokontrolleren. RJ11 pluggen har samme konfigurasjon som tilsvarende plugg på sensorkortet. På dette testkortet har vi ikke samme problem som på sensorkortet mtp. konvertering fra 3,3V logikk til 5V, dette skyldes at kortet er benyttet sammen med AVR med driftspenning på 5V. Testkortet har ikke montert ESD-diode slik som sensorkortet har, dette skyldes først og fremst at vi ikke hadde komponenten tilgjengelig når testkortet ble laget, men også at denne ikke er strengt nødvendig for å teste funksjonalitet. Linedrivertestkortet gjorde at vi fikk, kanskje det mest kritiske, delsystemet som skulle kommunisere med sensorene riktig på et langt tidliger stadige i prosjektet enn vi ville hatt dersom vi måtte vente på sensorkortet for teste/ feilsøke. Skjemategning og bilde for dette kortet finnes i figur 112 i vedlegget.

17.9.4 USB tranceiver

Da vi innså at vi ikke ville kunne beholde FPGA funksjonaliteten på displaykortet pga. gal footprint for FPGAen, ble løsningen å bruke et FPGA testkort. Testkortet vi brukte var et Memec Virtex II System Board. Til dette fulgte det med et kommunikasjonskort med usb transceiver og plugg. Dette kortet var imidlertid konfigurert for å fungere som USB-device, og siden vi på displaykortet skulle implementere en USB host, så vi oss nødt til å lage et eget kort for det fysiske interfacet mot USB-bussen. Kortet er tilpasset bruk som host ved at det har en USB-host plugg, to pulldown motstander på datalinjene, og mulighet for å levere +5 volt til devicene på bussen. Skjemategning for dette kortet finnes i vedlegget (figur 113).

17.9.5 Logikkknivåkonvertering

Siden vi på to steder i designet måtte konvertere logiskesignaler fra 3,3V til 5V og motsatt måtte vi finne en fornuftig måte å gjøre dette på. Det finnes en mengde kretser på markede for å gjøre dette, men siden vi hadde signaler med lav frekvens og kravene til nøyaktighet var relativt små prøvde vi å finne en annen måte for dette. Grunnen til at vi ikke ville bruke en ferdig krets var først og fremst at vi var usikre på levering og pris. Vi valgte å bentte en standard NPN transistor av typen BC547b. Koblingen vi brukte gjorde signalet invertert og defor benyttet vi en inverter i koblingen også. For å få et så stabilt og pent signal som mulig valgte vi å bruke en inverter med Scmitt-trigger inngang. Dette er en komponent med en hysteresefunksjon for overgangen mellom logisk en og logisk null. For å verifesere at denne koblingen ville fungere ved de hastighetenen vi var avhengig av ble det hele koblet opp og testet. Spenninger og tidsforsinkelser ble avlest på oscilloskop.

17.9.6 Spenningsstabilisering

For å kunne garantere riktige spenninger fra strømforsyningskortet ble LM317 koblet opp og testet. Hovedgrunnen til at vi gjorde dette var for å verifersere motstandsverdiene som avgjør utspenningen fra LM317 kretsen. Vi benyttet også testkoblinger for å forsikre oss om at kretsen kunne gi tilstrekkelig strøm over lengre tid.

17.9.7 Konklusjon

Alt i alt har disse kortene og testkoblingene vært avgjørende for utviklingen av programvare og design for sensorkortet. Vi har vha. disse redusert feilsøkingen til små undersystemer. Dette har gjort at feil har vært betraktelig enklere å lokalisere og rette opp i allerede på et tidlig stadie i utvikingen. Spesielt har utviklingen av kortene gitt oss trening i bruk av programvare for skjemategninger og kortutlegg, ved siden av erfaring med lodding som har gjort jobben med montering på de store kortene enklere. Det har ved utvikling av testkortene hele tiden vært et mål å gjøre dem så like kortene de skulle erstatte som mulig. Dette for vi skulle kunne garantere at

koden som kjører på et testkort også kjører på de virkelige kortene, dette har vi lykkes med i stor grad. Testkoblingene våre har også vist seg å fungere i det virkelige designet.

17.10 Endringer som ble gjort på kortene

Når vi fikk sensorkortet og displaykortet tilbake fra produksjon oppdaget vi noen feil og mangler som vi utbedret. Under følger en forklaring av hvordan vi løste de problemene som ble oppdaget

Display hadde ikke spenning

Grunnet feil i skjematgninger og PCB utlegg hadde ikke displayet driftspenning. Dette ble løst ved å lodde på ledninger på undersidenn av kortet. Riktig spenning var lett tilgjengelig rett ved displayet.

Kortslutning på displaykortet

Displaykortet hadde kortslutning mellom planene for GND og 3,3V. Dette ble løst ved å borre opp de viaene som var kortsluttet.

Tastatur og mikrokontroller

I utgangspunktet var det meningen å koble tastaturet til mikrokonrolleren gjennom FPGAen. Men siden FPGA ikke ble montert på kortet besluttet vi å omgås denne beslutningen. Måten vi løste dette på var å designe et kort bestående av en integrert brikke for design dekoding. Denne heter 74x922. Kretsen trengte kun to kondensatorer for å fungere som ønsket. Driftspenning til kretsen tok vi fra nærmest mulig sted på undersiden av kortet. I tillegg til dette måtte vi lodde på noen ekstra ledninger på undersiden av kortet. Denne løsningen viste seg å fungere utmerket.

Bytte av TXD og RXD

På sensorkortet var elektronikken som konverter fra 3,3V til 5V byttet om på TXD og RXD. Dvs. retningen på koblingene var byttet om. Måten vi løste dette på var å bytte om hvordan pinnene på AVR'en gikk ned i sokkelen i tillegg til å krysskoble to jumpere på kortet.

Test nr.	Subkomponent	Beskrivelse av test	Forventet utfall	Resultat/ Feilnr.
1-1	ingen	Koble til STK500 via ISP pluggen og programmer brikken	Brikken lar seg programmere	OK
1-2	ingen	Skriv et testporgram som skriver noe til en tilgjengelig utgang. Koble til logikkanalyasator på utgang på AVR.	Logikkanlaysatoren skal gi respons i henhold til testprogram	OK
1-3	ingen	Koble til JTAG ICE og les av registerinnhold.	Registere lar seg lese av	OK
1-4	Minne	Lag et testprogram som skriver til minne og leser dette ut igjen. Det skal skrives en indikator til en av portene på kontrolleren for resultat	Forventet resultat avlest på port	OK
1-5	Minne	Fortsett fra testen over. Koble fra spenningen til kortet. Koble til spenningen igjen. Avles respons på utgang.	Forventet resultat avlest på port	OK
1-6	Sanntidsklokke	Lag et testprogram for å stille klokke. Programmet skal videre lese av klokken og vise dette på en måte som gjør at responens fra klokken lar seg lese av.	Klokken viser tegn til å endre tidspunkt	OK
1-7	Sanntidsklokke	Fortsett testen over, men koble fra spenningen til kortet. Testprogrammet skal fortsette å vise tiden når spenningen kobles til igjen.	Klokken viser tidspunkt i henhold til den tiden spennigen har vært borte	OK
1-8	Analog sensor	Lag et testprogram som leser av analogsensoren og skriver dette til en port (Lyssensoren må testets først).	Porten gir respons i henhold til testprogram og lysforhold	OK
1-9	Digital sensor	Lag testprogram for å detektere linedriver	Linedriver detekteres	1F-1
1-10	Digital sensor	Lag testprogram for å detektere sensorer på One-Wire bussen	Sensorer detekteres	OK
1-11	Digital sensor	Lag testprogram for å teste alle sensorer	Sensorene detekteres	1F-2
1-12	Digital sensor	Lag testprogram for å kommunisere med sensorer.	Data avleses som forventet	OK

Tabell 6: Test av mikrokontrolleren.

Test nr.	Funksjon som testes	Beskrivelse av test	Forventet resultat	Resultat/ Feilnr.
2-1	JTAG grensesnitt	FPGA koples til en PC via JTAG	PC programvare detekterer FPGA	Ok
2-2	Programmering	Testprogram overføres til FPGA via JTAG	"Done"-pinne på FPGA indikerer at programmering er fullført	Ok
2-3	Programmering	Testprogram overføres til FPGA fra PRO	"Done"-pinne på FPGA indikerer at programmering er fullført	Ok
2-4	FPGA	FPGAen resettes	FPGA resetter seg selv og inntar starttilstand	Ok
2-5	FPGA	Testprogram (med blinkende lysdioder) overføres til FPGA	Lysdioder blinker	Ok
2-6	AVR grensesnitt	En testpakke (1 byte) sendes til AVR	AVR rapporterer at pakke er mottatt	OK
2-7	AVR grensesnitt	En testpakke (1 byte) sendes fra AVR	FPGA rapporterer at pakke er motratt	OK
2-8	USB kommandotolker	Kommando "skriv pakke" med påfølgende Byte sendes inn til enheten	Pakke lagret i buffer	Ok i simulering
2-9	USB Kommandotolker	Pakke leses ut for videresending til USB kontroller	Pakke hentes Byte for Byte og legges ut på datapinner	OK i simulering
2-10	USB Kommandotolker	Kommando for lesing av interruptregister sendes inn	Pakke hentes Byte for Byte og legges ut på datapinner	OK i simulering
2-11	USB Kommandotolker	Kommando sendes inn fulgt av adressen	Korrekt adresse settes og legges ut på pinner for bruk i USB Kontroller	OK i simulering
2-12	USB Kommandotolker	Kommando sendes inn	Pakke leses ut fra buffer og legges ut på datapinner for videresending til AVR Grensesnitt	OK i simulering
2-13	USB Kontroller	En SETUP-token sendes inn	Tilstandsmaskinen kommer i riktig tilstand og venter på en innkommende DATA-pakke	OK i simulering
2-14	USB Kontroller	En OUT-token sendes inn	Tilstandsmaskinen kommer i riktig tilstand og venter på en innkommende DATA-pakke	OK i simulering
2-15	USB Kontroller	En IN-token sendes inn	Tilstandsmaskinen kommer i riktig tilstand og sender en	OK i simuler-

Test nr.	Testnavn	Beskrivelse av test	Forventet resultat	Resultat/ Feil nr.
3-1	Sending til AVR	AVR grensesnitt	En testpakke (str. 1 Byte) sendes til AVR	AVR rapporterer at pakke er mottatt
3-2	Sending fra AVR	AVR grensesnitt	En testpakke (str. 1 Byte) sendes fra AVR	FPGA rapporterer at pakke er mottatt

Tabell 9: Test av FPGA kommunikasjon mot AVR

Test nr.	Subkomponent	Beskrivelse av test	Forventet utfall	Resultat/ Feilnr.
4-1	ingen	Koble til STK500 via ISP pluggen og programmer brikken	Brikken lar seg programmere	
4-2	ingen	Koble til logikkanalyasator på utgang på AVR, benytt en tilfeldig utgang som er lett tilgjengelig.	Logikkanlaysatoren skal gi respons i henhold til testprogram	
4-3	ingen	Koble til JTAG ICE og les av registerinnhold.	Registere lar seg lese av	
4-4	Display	Lag testprogram som skriver ut tekst til display, kjør programmet	Tekst vises som ønsket	OK
4-5	Display	Lag testprogram som tegner grafikk på display, kjør programmet	Grafikk vises som ønsket	OK

Tabell 11: Test av mikrokontroller på displaykort

Test nr.	Funksjon som testes	Beskrivelse av test	Forventet resultat	Resultat/ Feil nr.
5-1	topp	Koble til aktuelt utstuyr for å programmere enheten	Program leses inn	OK
5-2	topp	Koble til JTAG og les ut informasjon	Informasjon er tilgjengelig	
5-3	Testkort	Programmere FPGA'en	Display skal vise 1	OK
5-4	Transmitter /toppnivå	Teste sending av en setup-pakke vha. Modelsim og VHDL testbench	Korrekt utformet waveform	OK
5-5	Transmitter /toppniv	Teste sending av data-pakke vha. Modelsim og VHDL testbenc	Korrekt utformet waveform	OK
5-6	Transmitter /toppnivå	Teste sending av token-pakke vha. Modelsim og VHDL testbench	Korrekt utformet waveform	OK
5-7	Receiver /toppnivå	Teste mottak av data-pakke vha. Modelsim og VHDL testbenc	Korrekt utformet waveform	OK
5-8	Receiver /toppnivå	Teste mottak av ACK/NAK-pakke vha. Modelsim og VHDL testbench	Korrekt utformet waveform	OK
5-9	Tastaturmodul	Test polling av tastatur og utsendig av data i henhold til trykket tast	Korrekt dataverdi på utgangen på FPGA, pinne for interrupt settes også	Ikke testet

Tabell 13: Test av FPGA på displaykortet

Test nr.	Beskrivelse av test	Forventet resultat	Resultat/ Feil nr.
7-1	Koble til belastning som trekker 100mA på hver av de ikke-kontrollerte strømutgangene. Mål levert spenning.Kontroller at korrekt LED lyser.	Spenninger som tilsvarer spesifisert spenning +/- 10%. LED lyser	OK
7-2	Kortslutt inngangen for kontroll mot 3,3V ikke-kontrollert. Belast utgangene med 100mA. Mål leverte spenninger. Kontroller at korrekt LED lyser.	Spenninger som tilsvarer spesifisert spenning +/- 10%. LED lyser	OK
7-3	Koble inn batterilader. Koble på batterieliminator. Lad batteriet i 90 minutter. Koble fra batterieliminator.	Utganger skal ha korrekt spenning	3-F1

Tabell 15: Test av Strømforsyning

Test nr.	Beskrivelse av test	Forventet resultet	Resultat/ Feilnr.
8-1	Koble til 5 volt og jord på korrekte pinner. Mål spenning på utgangen.	Spenningen skal endre seg avhengig lysstyrken sensoren belyses med.	OK

Tabell 16: Testing av lyssensor

Test nr.	Beskrivelse av test	Forventet resultet	Resultat/ Feil nr.
0-1	Koble sensorkort og displaykort sammen. Prøv å stille klokka fra displaykortet.	Klokke stilles på sensorkortet	Ikke gjort
0-2	Koble sensorkort og displaykort sammen. Prøv å sette samplingsintervallet.	Samplingsintervallstilles.	Ikke gjort
0-3	Koble sammen kortene. Skru på displaykortet og se at sensorkortet "våkner" ved USB-aktiviteten.	At displaykortet skal være på ved påknappen og sensorkortet på ved USB-aktivitet.	Ikke gjort
0-4	Hele systemet testes, dvs sensorkort og displaykkort kobles sammen, noen målinger tas og noen data finnes fram på displayet. Systemet evalueres.	At alt fungerer	Ikke gjort
0-5	Koble på en datamaskin med AVR Studio på sensorkortet og se at det foretas målinger med gitte mellomrom. Koble på logikkanalysatoren og se til at det ikke er noe aktivitet på kortet mellom målingene.	At sensorkortet "vekker" og "sovner" selv.	Ikke gjort
0-6	Gjør noen målinger med sensorkortet ikke tilkoblet displaykortet. Sjekk vha logikkanalysatoren at det ikke er aktivitet på FPGAen mens sensorer avleses.	At FPGAen ikke er aktiv ved sensoravlesning.	Ikke gjort
0-7	Få en utenforstående person til å teste systemets brukervennlighet.	At brukeren er fornøyd med systemet og synes det er lett å bruker uten for mye knot.	Ikke gjort

Tabell 18: Systemtester

Feil nr	Feil	Utbedring (hva)	Konsekvenser
2F-1	Det er ikke implementert støtte for tastatur i FPGA	Det ble laget eget kort for denne funksjonaliteten	Ingen
5F-1	Kortsluttning mellom VCC 3,3V og GND	Vias borret opp	Ingen, en bane som ikke er i bruk ble brutt.
5F-2, 5F-3	FPGA har feil footprint	Ikke mulig å rette opp	All funksjonalitet på FPGA ble flyttet ut på testkort
5F-4	VCC til display er ikke tilstede.	Loddet på ledninger	Ingen
1F-1	Linedriver ble ikke detektert. Nivåtilpassingen var tegnet feil vei på skjemategninger	Byttet om TX og RX på AVR i tillegg til krysskobling av to jumpere	Ingen
3F-1	Batteriladertest er ikke gjennomført. Dette skyldes at batteripakken som var bestillt aldri kom	-	Få, systemet kan alikevel benytte batterieliminator som spenningskilde
1F-2	Alle sensorenen som var tilkoblet ble detektert, men fordi vi ikke mottok alle sensorene vi hadde bestilt er ikke alle kravene oppfylt mtp. hvilke forhold som skulle kunne avleses.	-	Ikke alle værforhold som er nevnt i kravspekken kan ikke avleses
10F-4	Toppnivå test ikke bestått fordi enheter ikke er satt sammen	-	USB funksjon ute av drift
10F-6	Test ikke gjennomført fordi denne funksjonaliteten ikke er implementert	-	Feilsjekk av USB-pakker ute av drift

Tabell 20: Tester som feilet

18 Aktuelle utvidelser

FORD: Six pints of bitter. And quickly please, the world's about to end.
BARMAN: Oh yes, sir? Nice weather for it.
– Douglas Adams, The Hitchhikers Guide To The Galaxy

18.1 Hardware

18.1.1 (Korrekt) FPGA

Testkortet vårt hadde feil footprint på FPGA'en (OBS! Gerberoutputen i appendixet er PCB'en som ble produsert, og ikke rettet opp). Skjemategningene er alikevel korrekte. Problemet oppstod da vi skulle mappe pinne 1 til pinne 1 på pakken. Vi benyttet feil standardpakke til PQFP208, der pinne 1 sto midt på en av sidene, i stedet for i hjørnet. Dette gjorde hele PCB'en ubrukelig til testformål i forbindelse med FPGA software. Dette problemet ble løst ved at vi benyttet testkort til prototyping.

18.1.2 Testpod mellom FPGA og USBtransceiveren

Etter en del testing oppdaget vi at det kunne være greit med en pod å koble seg til. Slik at man kan verifisere at dataene ut fra FPGA'en er riktig og man slipper å undre seg over om det er feil på transceiveren. Spesielt når footprinten vår på FPGA'en ikke ble riktig var det viktig. Da kunne vi ha brukt poden til å koble oss direkte på.

18.1.3 TWI

Det ble lagt opp linjer for TWI (Two Wire Interface, eller I2C) på begge kortene. Tanken var å ha et alternativ til USB. Spesielt i en nedstrippet versjon av begge kortene, for å gjøre dem billigere eller dersom man ikke ønsker å bruke USB.

18.1.4 Alternative USB løsninger

Etterhvert som vi jobbet kom vi over en del alternativer til vår USB-løsning. Det finnes en rekke ASICs og andre ferdige produkter til dette formålet. Man kan f.eks bruke PDIUSBD11D. Det er en integrert krets som oversetter fra I2C til USB. Men slik oppgaven var gitt var litt av poenget å implementere USB på en FPGA, derfor benyttet vi ikke denne løsningen.

18.1.5 RS232 kommunikasjon

Å kunne koble til en gammeldags terminal (type vt330) kan være nyttig, og er heller ikke vanskelig. Hovedproblemet vil ligge i eventuell plassmangel på sensorkortets AVR, i og med at denne nå må ta vare på hele menystrukturen som ellers har blitt forvist til displaykortet eller en datamaskin. Dette kan også være aktuelt for å øke antall debuggingsmuligheter. Dette ble nedprioritert pga tidsmangel.

18.1.6 Større display

Det lille displayet egnet seg ikke så godt for grafer. Det hadde vært interessant å prøve et større display for dette formålet. Spesielt vanskelig var det å få inn benevning på aksene. Skal vi ha et større display krever det imidlertid en ekstern minnebrikke, dersom vi skal beholde dagens programvareløsning, der hele skjermbildet bufres i minnet på AVR'en og så tegnes i sin helhet.

18.1.7 Underklokke mikrokontrolleren

For å kunne spare enda mer strøm kan det være interessant å prøve å klokke AVR'en enda lavere. Som regel betyr det enda mindre effektforbruk.

18.2 Software

18.2.1 Utvidet PC software

Skikkelig PC software med støtte for komplisert statistikkføring. I tillegg bør det være mulig å legge inn dataene inn i en felles database. Det en kunne tenke seg her er f.eks en felles database med flere værstasjoner.

18.2.2 Støtte for USB memory-stick

Et av alternativene vi vurderte for designet av displaykortet var å støtte USB memory-stick. Dersom vi integrerer en USB hub sammen med USB hosten på displaykortet kan vi la brukeren koble inn en memory-stick og kopiere ut data fra sensorkortet. Disse kan så enten brukes til å generere statistikk på displaykortet, ved et senere tidspunkt, eller på en PC for nærmere analyse.

18.2.3 Utviding av minnedriver

Måten vi lagerer data på minnebrikken gjør det problematisk å stille klokken. Det kan derfor være ønskelig å forandre minnedriveren til å benytte en annen struktur som tillater relativt raske søk, selv om data ikke ligger sortert på brikken.

19 Diskusjon

> "There was, it appeared, a mysterious rite of initiation through which, in one way or another, almost every member of the team passed. The term that the old hands used for this rite – West invented the term, not the practice – was 'signing up.' By signing up for the project you agreed to do whatever was necessary for success. You agreed to forsake, if necessary, family, hobbies, and friends – if you had any of these left (and you might not, if you had signed up too many times before). "
>
> – Tracy Kidder, "The Soul of a New Machine"

19.1 Arbeidsflyt

Ettersom vi var temmelig mange som skulle ta faget i år (11 stykk), ble vi splittet opp i to undergrupper. Det ble da bestemt fra fagstabens side at vi skulle lage to separate PCB'er med to distinkte funksjoner. Det ene kortet skulle skulle stå for innsamling av data (sensorkortet), mens det andre skulle kunne vise frem informasjonen (displaykortet). 6 studenter skulle jobbe med sensorkortet, mens 5 stykker skulle jobbe med displaykortet. Inndelingen av gruppene sto fagstaben for. Vi prøvde så langt det var å la alle få prøve seg på de forskjellige aspektene ved prosjektet (PCB design, C og VHDL).

Det ble tidlig i prosjektet valgt en leder som skulle stå ansvarlig for møter og framgangen i prosjektet. I tillegg fant vi fort ut at de to undergruppene trengte en leder, så vi valgte en der også.

19.2 Erfaringer

19.2.1 Teamarbeid

Vi fokuserte vi ekstra mye på kommunikasjonsflyt og samarbeid. Vi opprettet et fellesområde med et CVS tre der alle kunne commite kode og dokumentasjon ettersom man ble ferdig med det. I tillegg hadde vi en mailingliste der vi kunne diskutere fritt, med faglærere som fulgte med på listen. Vi hadde også en webside, men den ble temmelig lite brukt ettersom alle hadde tilgang til filområdet.

19.2.2 Tutorials

Mesteparten av opplæringen i faget skjedde ved hjelp av tutorials. Disse dekket en rekke temaer, inkludert opplæring i bruk av elektroniske verktøy for design av PCB'er (Expedition PCB fra Mentor), Etsing av selvlagde PCB'er, innføring i VHDL, logikkanalysator og programmering for AVR.

19.2.3 Softwareutvikling på AVR.

De fleste på gruppen hadde allerede temmelig mye erfaring med programmering fra før. Vi bestemte oss tidlig for å bruke GCC som kompilator i stedet for en komersiell løsning, til tross for at IDI har lisenser på f.eks IAR kompilatoren. Dette var fordi vi ønsket å kunne benytte våre eksisterende kunnskaper om GCC kompilatoren i prosjektet. At kompilatoren er gratis gjør det også mulig for gruppemeldemmene å jobbe hjemme med prosjektet. Mesteparten av testing og utviklingen av koden skjedde mot STK500 kortet som er utmerket for dette formålet. Dette gjorde oss i stand til å teste enkeltkomponenter uten å ha en skikkelig PCB å prøve mot, samt vi kunne distribuere arbeidet bedre.

19.2.4 Softwareutikling på FPGA

Gruppen hadde ingen erfaring med VHDL før prosjektet. Her var det mye læring mens vi holdt på med prosjektet. En god del erfaring ble trukket fra datamaskinkonstruksjonsfaget (som gikk paralelt). Her fikk vi en oppgave der vi skulle implementere en mikroprosessor fra grunnen av. Dette gav oss en del erfaring før vi skulle implementere USB-host og USB-device. Vi fikk også her tak i et utviklingskort kalt "Virtex2 System board", laget av Memec design. Med tilleggskortet "P-160 Comm extension board", fikk vi muligheten til å teste USB direkte på et utviklingskort.

19.2.5 Forelesninger

Faget hadde et par forelesninger felles med datamaskinkonstruksjonsfaget. Dette hjalp litt på forståelsen av f.eks VHDL og FPGA-teknologi. Vi hadde dog foretrukket litt flere forelesninger om VHDL før vi satt i gang.

19.3 Ting som burde ha blitt gjort anderledes

19.3.1 Lese tidligere gruppers dokumentasjon

Vi oppdaget for sent at tidligere grupper også hadde noe å fare med, spesielt med tanke på organisering av arbeidet. Som et eksempel kan det nevnes at fjorårets gruppe valgte en dedikert regnskapsfører og en leder for dokumentasjonsarbeidet. I ettertid viste dette seg å være lurt, ettersom dokumentasjonsarbeidet vårt ble nedprioritert i forhold til selve oppgaven. Her fantes det også en god del andre nyttige tips, både tekniske og administrative. F.eks kunne vi unngått mye arbeid ved å gjenbruke PCB-biblioteket til fjorårets oppgave, slik at vi hadde sluppet å gjøre dette dobbeltarbeidet.

19.3.2 Starte programvareutvikling tidligere

Ettersom vi ønsket at flest mulig skulle få erfaring med flest mulig aspekter av designprosessen, designet vi først systemet på et konseptuellt nivå, deretter laget vi en PCB for så å utvikle programvaren/vhdl-koden. Dette førte til at vi kom igang med utviklingen av programvare og vhdl-kode alt for sent. Vi kunne fått til mye større parallellitet uten at det nødvendigvis skulle gå på bekostning av læringen.

19.3.3 Statusrapportering

Etterhvert som prosjeketet skred frem ble det klart at vi trengte mye større kontroll over hva som foregikk. Vi startet da med ukentlige statusrapporter der hvert medlem av gruppen fortalte om status på sitt egen oppgave. Dette gjorde at vi kunne sette realistiske tidsfrister. Uheldigvis startet vi med dette alt for sent.

19.3.4 Sette klare rettningslinjer for dokumentasjon

Etterhvert som gruppens medlemmer skrev dokumentasjon ble det fort klart at de var skrevet på høyst forskjellige måter. Struktur og detaljnviå varierte mye og mye tid gikk tapt i å flette dokumentasjonen slik at den fremsto som helhetlig. Dette kunne vært unngått om en hadde laget retningslinjer for skriving av dokumentasjon så tidlig som overhodet mulig.

19.3.5 Intern opplæring i CVS og TEX

En god del av gruppen hadde ikke noen erfaring med CVS[21] og TEX tidligere. Her burde vi ha satt i gang en liten opplæringsrunde på rundt en time eller to for å lære opp disse. I stedet tok vi opplæringen en for en og det tok unødvendig mye tid.

19.4 Sluttstatus

Det gjennstår enn noe arbeid før hele systemet fungere slik vi hadde ønsket. Det viktigste som gjennstår er å få USB implementasjonen til å fungere. Store deler her har vist seg å virke ved simmulering, men når systemet settes sammen er det noe som går galt. Nøyaktig hva er det vanskelig å si noe konkret om. Det er også vanskelig å si hvor mange timer en måtte regnet med å bruke for å få denne delen av systemet til å fungere tilfredstillende. AVR implementasjonen er derimot tilnærmet ferdig, selfølgelig med unntak av det som må kodes for at ting skal fungere mot FPGA og sammen med USB. Denne jobben er påbegynt, men ble ikke prioritert da det viste seg at vi alikevel ikke kom i havn. Med tanke på brukervennliget og MMI er det helt klart mye som kan gjøres bedre. Vi har prioritert å få systemet til å fungere, pynting på menyer og andre ting har vi ikke regnet som viktig i denne prosessen.

[21] Concurrent Versions System

Konklusjon

På de første møtene etter at vi hadde fått presentert vår forholdsvis åpent definerte oppgave, formelig haglet det av høytflyvende ideer om "state of the art" funksjonalitet som skulle implementeres på værstasjonen vår; GSM modul, webserver, osv. Mye av prosessen har bestått i å lære litt mer om hva hardware design egentlig går ut på, og utivkle vår ydmykhet i forhold til den prosessen som ligger bak et ferdig hardware produkt.

Testingen tok forholdvis mye mer tid og tanke enn vi var forberedt på, og vi lærte fort at det å debugge og teste en hardware implementasjon kan by på ganske andre utfordringer enn de tilsvarende prosessene for software.

Stemningen i gruppa var laber da vi fant ut at en uheldig misforståelse rundt footprinten til FPGAene gjorde at vi ikke fikk implementert all funksjonaliteten på våre egene kretskort. Heldigvis klarte vi å ta det for den utfordringen det var, og selv om det krevde litt ekstra innsats, klarte vi ved hjelp av tilgjengelige teskort og kort vi laget selv å legge til rette for å kunne implementere den planlagte funksjonaliteten allikevel.

Vi kan dessverre ikke presentere et ferdig virkende prototyp i skrivende stund, men vi har en konseptuell prototyp som vi mener representerer godt de designvalgene og timene vi har lagt ned foran skjermene. Her kan nenvnes at systemet kan lese av sensorer og lagre disse på minnebrikken, det kan ta i mot instuksjoner fra brukeren via tastaturet, det kan vise menyer og tegne grafer på displayet. Vi hadde måttet legge ned mange flere arbeidstimer får å få til en komplett USB implementasjon, og tatt i betraktning den utfordringen det er å implementere en såpass komplisert protokoll, er vi fornøyd med den funksjonaliteten vi klarte å få til.

Mange av prosjektmedlemmene hadde ingen erfaring med design av hardware før prosjektet startet, men dette prosjektet har gitt oss en positiv start, og inspirert til videre læring på området. Selv om ikke alle hadde tid til å la lære om alle aspeketer av hardwaredesign, har samtlige gjort seg nye erfaringer når det gjelder gruppearbeid og sammarbeid som kan være nyttig å ha med seg videre i både studiesituasjonen og arbeidslivet. Hvordan vi har manøvrert oss frem i ukjent terreng får bli opp til andre å bedømme. Vi føler selv at vi har fått til mye. Og det har vært spennende og utfordrende å bli kastet ut på dypt vann for å lære å svømme i syrebad.

Referanser

[1] ATMega128(L) Preliminary Complete, http://www.atmel.com/dyn/resources/prod_documents/doc2467.pdf

[2] ATmega32(L) Preliminary Complete, http://www.atmel.com/dyn/resources/prod_documents/doc2503.pdf

[3] Spartan-IIE Complete Data Sheet (All four modules), http://direct.xilinx.com/bvdocs/publications/ds077.pdf

[4] DS2480B, Complete Datasheet, http://pdfserv.maxim-ic.com/en/ds/DS2480B.pdf

[5] PDIUSBP11A; Universal Serial Bus Transceiver , http://www.semiconductors.philips.com/acrobat/datasheets/PDIUSBP11A_3.pdf

[6] G12032 Serial Graphics LCD Manual, http://www.seetron.com/pdf/sgx120.pdf

[7] MAX712, Complete Datasheet, http://pdfserv.maxim-ic.com/en/ds/MAX712-MAX713.pdf

[8] Universal Serial Bus Revision 2.0 specification, http://www.usb.org/developers/docs/usb_20.zip

[9] MAX4202, Complete Datasheet, http://pdfserv.maxim-ic.com/en/ds/MAX4200-MAX4205.pdf

[10] DS1302, Complete Datasheet, http://pdfserv.maxim-ic.com/en/ds/DS1302.pdf

[11] AT45DB642 Datasheet, http://www.atmel.com/dyn/resources/prod_documents/DOC1638.PDF

[12] Overview of 1-Wire Technology and Its Use, http://pdfserv.maxim-ic.com/en/an/AN1796.pdf

[13] LM258, Datasheet, http://www.national.com/ds.cgi/LM/LM158.pdf

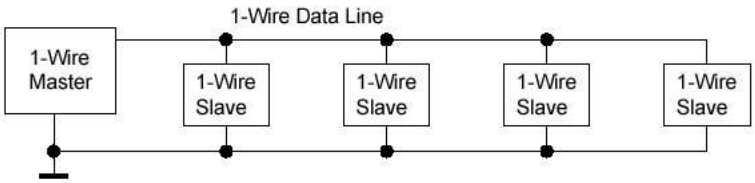
[14] TSL250R, Datasheet, http://www.taosinc.com/images/product/document/TSL250R.pdf

[15] PVA3354N, Datasheet, http://www.promelec.ru/pdf/pva33n.pdf

[16] LM317, Datasheet, http://www.national.com/ds/LM/LM117.pdf

A 1-Wire

A.1 Oversikt

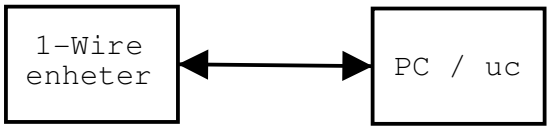


Figur 54: Enkelt 1-Wire Nett

1-Wire er en bussteknologi som kun bruker en leder (i tillegg til jord) for kommunikasjon og strøm overføring. En enkelt bus-master kan snakke med flere slaveenheter over en felles twisted-pair[22] kabel, se figur 54. Et viktig aspekt ved denne teknologien er at hver slaveenhet har en unik global digital adresse. 1-Wire teknologien leveres av Dallas Semiconductor/Maxim, for mer informasjon se 12.

A.2 1-Wire Nettverk

A.2.1 Hovedkomponenter



Figur 55: Hovedkomponentene i et 1-Wire Nett

Som vi ser av figur 55 består et 1-Wire nettverk av tre hovedkomponenter:

Bus-master (PC/mikrokontroller)

Kabler og koblinger

1-Wire enheter

Alle 1-Wire enhetene kommuniserer gjennom bus-master, og det er bus-master som bestemmer hvilke slaveenheter som får kommunisere.

[22] tvinna telefonkabel eller uisolert "twisted-pair" (UTP) CAT-5, avhengig av kabelstrekket

A.2.2 1-Wire Protokollen

1-Wire protokollen bruker standard TTL/CMOS logikk nivå og opererer innenfor 2.8V til 6V området. Det går en sekvensiell datastrøm mellom 1-Wire enheten og bus-master. Datastrømmen går kun en vei om gangen, og all lesing/skriving starter på minst signifikante bit først. For å slippe unna signalgenerering og timing kan man benytte en linedriver som tar seg av dette mot bus-master. Linedriveren, DS2480B, fungerer som en bro mellom 1-Wire bussen og et uart grensesnitt. Den tar seg av timeslot generering og alt som har med strømtilførsel av 1-Wire enhenten å gjøre.

Det er ikke nødvendig med system klokke over bussen fordi hver 1-Wire enhet har sin egen interne oscillator. Den interne oscillatoren synkroniseres med fallende klokkeflanke fra bus-master.

1-Wire enhetene får strøm ved å benytte "parasite-power"[23] prinsippet over bussen.

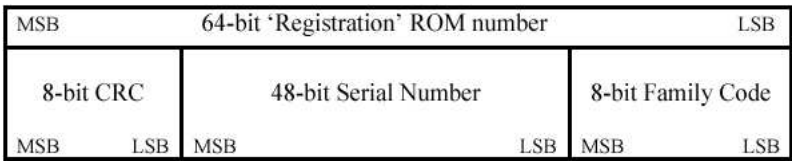
Kommunikasjonen er inndelt i:

Handshake

Data overføring

Handshaken består av en reset av 1-Wire bussen fra bus-master. Det vil si at bus-master eller linedriver holder 1-Wire bussen lav i en bestemt periode. Deretter svarer alle 1-Wire enhetene på bussen med en "prescence pulse". Bus-master kan da adressere ønsket enhet og starte enhetspesifikk kommunikasjon med denne for videre dataoverføring.

A.2.3 Unike adresser på 1-Wire enhetene



Figur 56: Unik 64-bits ROM 'registrerings' nummer

Hver 1-Wire slave har en unik 64-bits ROM kode. Koden er som vi ser av figuren 56 delt inn i 3 deler. Den første byten (MSB) inneholder en CRC kode, de neste 48-bit'ene inneholder ic'ens unike Serienummer, mens den siste byten (LSB) inneholder familiekoden til ic'en.

1-Wire systemet har gode muligheter med tanke på datasikkerhet og nettet kan bygges ut fra å være et ordinært kjeda nettverk til store hirarkiske nett ved å dele det inn i seksjoner via koblingspunkt ic'er.

[23] Hver 1-Wire enhet har en enveislikeretter. Dermed kan kondensatoren på hver enhet lades for hver gang datalinja går høy (5V) og fungerer som batteri for enheten når datalinja går lav igjen.

A.2.4 Godt utvalg av 1-Wire enheter

1-Wire systemet har et godt utvalg av 1-wire ic'er, både kontroll ic'er for 1-Wire bussen samt mange forskjellig registreringsenheter som temperaturmålere og a/d konvertere. Alle kommuniserer sammen over 1-Wire bussen.

B En kort innføring i USB protokollen

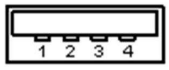
I dette kapitlet gir vi en kort innføring i USB protokollen. Det er lagt mest vekt på de delene av protokollen som er relevante og nødvendige for å kunne å kunne implementere USB på en FPGA. Ønskes mer detaljert informasjon om USB henvises det til USB 1.1/2.0 spesifikasjonen[8].

USB er en seriell buss som gjør det enkelt å kople til eksterne enheter til en PC. USB tilbyr hot-plugging, enheter kan plugges inn og ut uten at man trenger å være redd for kortslutning. I tillegg til at den er enkel å bruke tilbyr USB også hastigheter på opptil 480 Mbps.

Vi har valgt å bruke USB protokollen fordi den, i tillegg til å oppfylle de krav vi har stilt til kommunikasjon, er enkel å bruke for brukeren. For å overføre data fra værstasjonen er det ikke behov for de høyeste hastighetene, vi har derfor valgt å operere på 1.5 Mbps.

B.1 Kontakter

Pinne 1 er + 5V, pinne 4 er jord, mens pinne 2 ("D-") og 3 ("D+") brukes til overføring av selve dataene. Vi finner type A kontakter på en host(Se figur 57) (f.eks. en PC eller en hub), type B (se figur 58) finner vi på en USB device (som værstasjonen vår).



Figur 57: USB-plugg type A

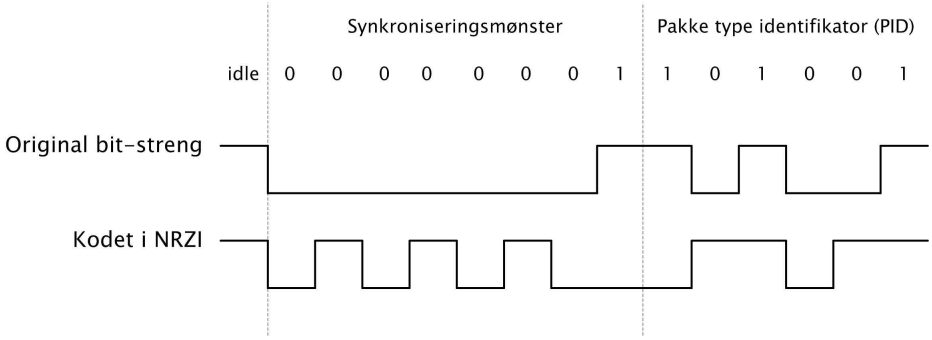
Figur 58: USB-plugg type B

B.2 Elektriske egenskaper

USB benytter seg av differensiell signalering. Det vil si at en logisk ’1’ sendes ved å la D+ få en spenning på over 2.8 V og D- en spenning under 0.3 V. En logisk ’0’ er motsatt, D+ med spenning under 0.3 V og D- med spenning over 2.8 V. I tillegg brukes også signalet "singel-ended-zero" ved å både D+ og D- få en spenning på under 0.3 V. Dette signalet brukes bl.a. til å indikere slutten på en pakke, dette forklares nærmere senere i dette kapitlet.

B.3 NRZI koding og bitstuffing

USB protokollen koder en bitsekvens med NRZI[24] før sending. Som vist i figuren under sendes en logisk 0 som en transisjon på linjen mens en logisk 1 sendes som ingen transisjon. I tillegg benyttes bitstuffing, hvor en ekstra 0 "stuffes" inn etter seks etterfølgende 1'ere Se figur 59. Dette for å sikre at det ikke går for lang tid mellom hver gang det er transisjon på bussen. Transisjonene brukes for å holde klokka til devicen synkronisert med klokka til hosten.



Figur 59: Eksempel på NRZI-enkodet signal

B.4 USB pakketyper

Det finnes mange forskjellige pakketyper, vi har her tatt med de pakketypene som er relevante for vårt formål.

Forklaring til felter i pakkene:

- Sync - brukes til synkronisering av klokka i devicen
- PID - pakkeidentifikator, de forskjellige pakketypene er vist i tabell 60
- Addr - adresse til en device
- Endp - hvilket endepunkt på devicen pakken skal til
- Crc5 / Crc16[25] - hhv. 5 og 16 bits felt å sikre at pakkene er uten feil
- Eop - End Of Packet, angir slutten på en pakke.

[24] Non-Return-to-Zero-Inverted
[25] Cyclic Redundancy Check[sum]

Type pakke	PID verdi	Pakkeidentifikator
Token	0001	OUT
	1001	IN
	1101	SETUP
Data	0011	DATA0
	1011	DATA1
Handshake	0010	ACK
	1010	NAK
	0110	STALL

Figur 60: Ulike pakketyper i USB protokollen

B.4.1 TOKEN pakke

Sync	PID	Addr	Endp	Crc5	Eop

Det er alltid hosten som tar initiativet til alle overføringer på USB bussen. En Token pakke brukes av hosten for å tildele bussen til en bestemt device. Skal en device konfigureres (f.eks. sette adressen) brukes SETUP, for overføringer til en device brukes OUT og for overføringer til hosten brukes en IN-pakke.

B.4.2 DATA0 / DATA1 pakke

Sync	PID	Data	Crc16	Eop

DATA pakker brukes for overføring av data. For overføring med lav hastighet kan dette feltet ikke være større enn 8 Byte. Det benyttes to typer datapakker; DATA0 og DATA1. Formålet med disse er å sikre at pakker ikke forsvinner på bussen uten at noen oppdager det.

B.4.3 HANDSHAKE pakke

Sync	PID	Eop

Handshake pakker brukes som bekreftelse på at en pakke er mottatt (ACK), at en pakke inneholdt feil eller ikke kom fram (NACK). Pakketypen brukes også for å indikere at devicen ikke har kapasitet til å motta flere pakker for øyeblikket (STALL).

B.5 Overføring av data

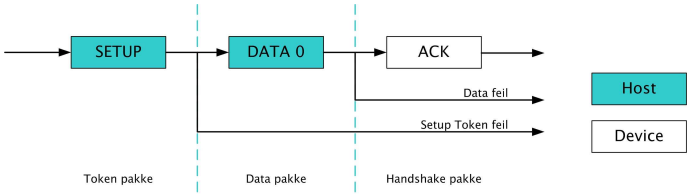
På USB-bussen er det som nevnt tidligere alltid hosten som tar initiativet til overføringer. Hver overføring starter derfor med at hosten genererer en TOKEN-pakke som forteller hvilken device det gjelder og hvilken retning dataene skal gå.

Det finnes flere forskjellige former for overføring av data på USB-bussen, bl.a. er det mulig å tildele en viss del av bussen til en bestemt device. Vi har imidlertid valgt å bruke "Control"-overføringer. Control- overføringer brukes først og fremst til å konfigurere en USB device, men egner seg også godt til overføring av data.

En Control-overføring består av tre deler:

1. Setup - her kunngjøres det hva slags forespørsel det gjelder
2. Data - her overføres data til og/eller fra en device
3. Status - mottaker av data rapporterer tilbake

B.5.1 Del 1: Setup



Figur 61: Sekvensdiagram for setup av kontrolloverføringer

Som vist i figur 61 består denne delen av en SETUP-pakke fulgt av en DATA pakke. Til slutt sender devicen en ACK-pakke som bekreftelse på mottak av DATA-pakken.

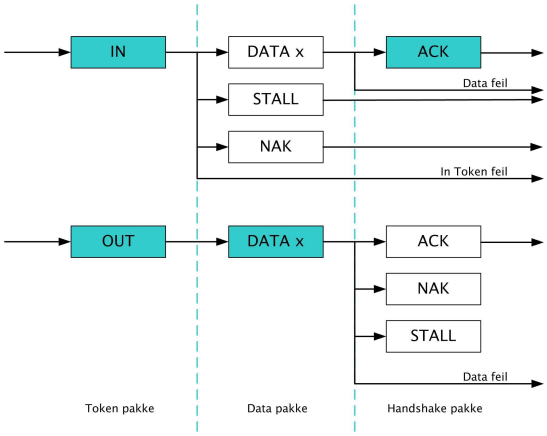
Innholdet i DATA 0-pakken er avhengig av type forespørsel. I USB standarden er det definert en rekke faste forespørsler; f.eks. etter Device-ID eller Vendor-ID, altså type enhet eller hvem som har produsert enheten. I tillegg har hver produsent mulighet for å legge til egne forespørsler. Slike forespørsler kan f.eks. være etter værdata for en bestemt dag.

B.5.2 Del 2: Data

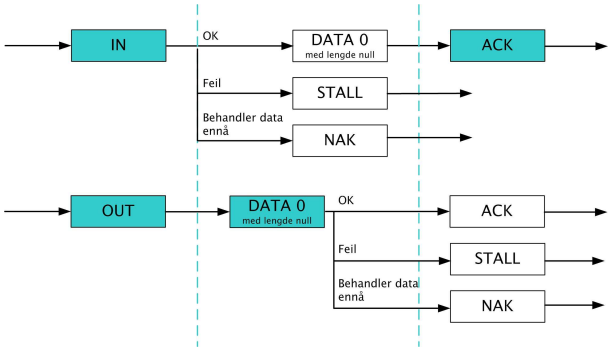
Data-delen består av en eller flere IN eller OUT overføringer (Se figur 62).

B.5.3 Del 3: Status

I forrige del, Datadelen, sendte mottaker en ACK-pakke for hver DATA-pakke for å indikere at akkurat den pakken var feilfri. Men det er også nødvendig å verifisere selve innholdet i DATA-pakkene, ikke bare at alle bit'ene ble korrekt sendt. Dette gjør mottaker i Statusdelen: Bestod Datadelen av evt. data til en device, rapporterer devicen tilbake med en DATA0-pakke med lengde null for å indikere at alle data var i orden (Se figur 63). Bestod Datadelen evt. i tillegg av data fra en device, rapporterer hosten tilbake til devicen med samme type DATA0-pakke.



Figur 62: Generell USB overføring



Figur 63: USB Status pakke, sekvensdiagram

C Kabinett

Kortene er montert i kabinett for å være lettere å håndtere. Vi hadde valget mellom å kjøpe bokser eller selv brekke til plater for å få mer nøyaktige størrelser. Vi valgte å kjøpe kabinett da både pris og utforming/ størrelse var innenfor våre krav. Utstyret er montert i to bokser.

Sensorkabinett:

- Sensorkort
- Strømforsyningskort
- Batteripakke

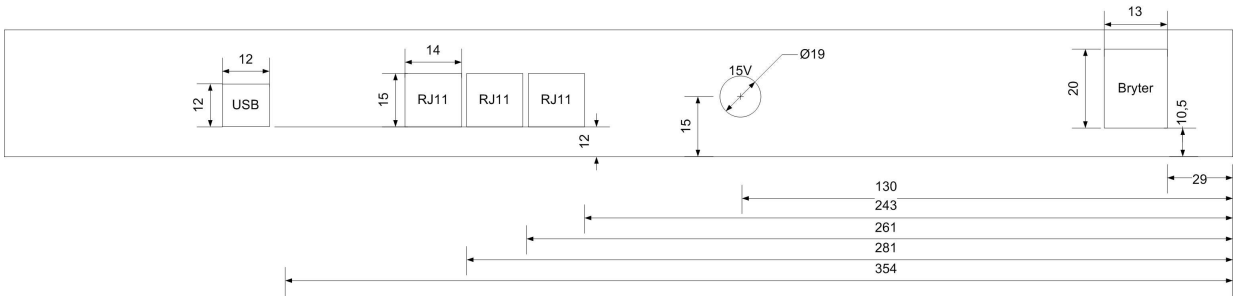
Displaykabinett:

- Sensorkort
- Strømforsyningskort
- Batteripakke

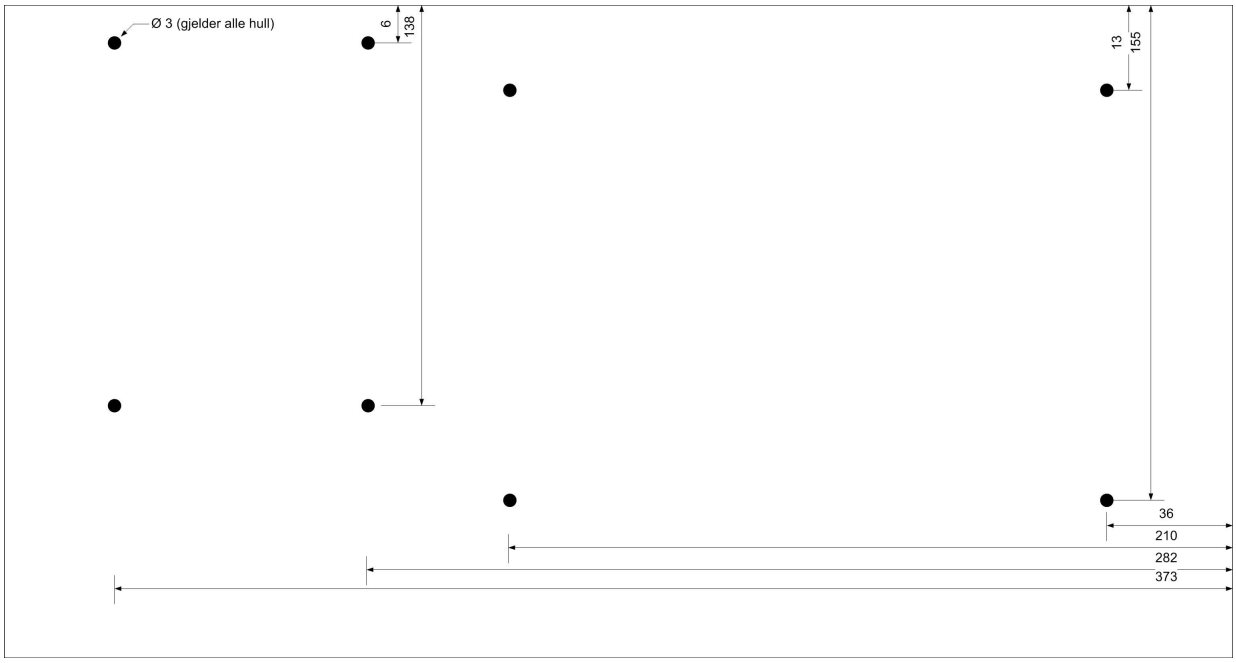
Vi har gjort visse modifikasjoner på kabinettene for at tilkoblinger og montering skulle fungere. Montering av kortene gjøres vha. buffere på 5 mm. slik at kortene ikke blir liggende ned mot chassi.

C.1 Sensorkabinett

Det er laget hull for at tilkoblingene skulle være tilgjengelig gjennom bakveggen i kabinett (Se figur 64). Kortene skrues fast i hullene vist i figur 65.



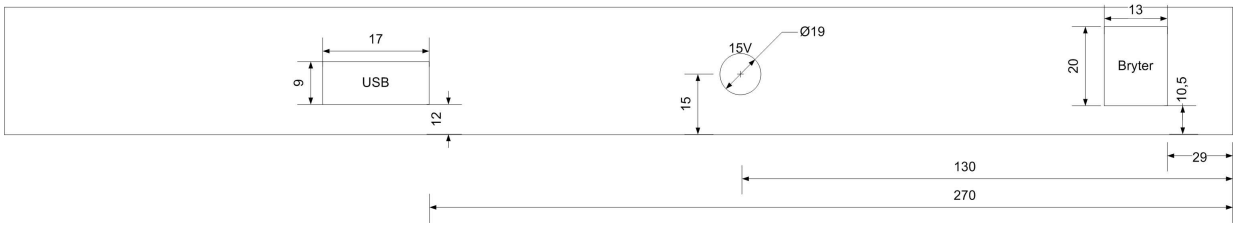
Figur 64: Sensorboksens bakside



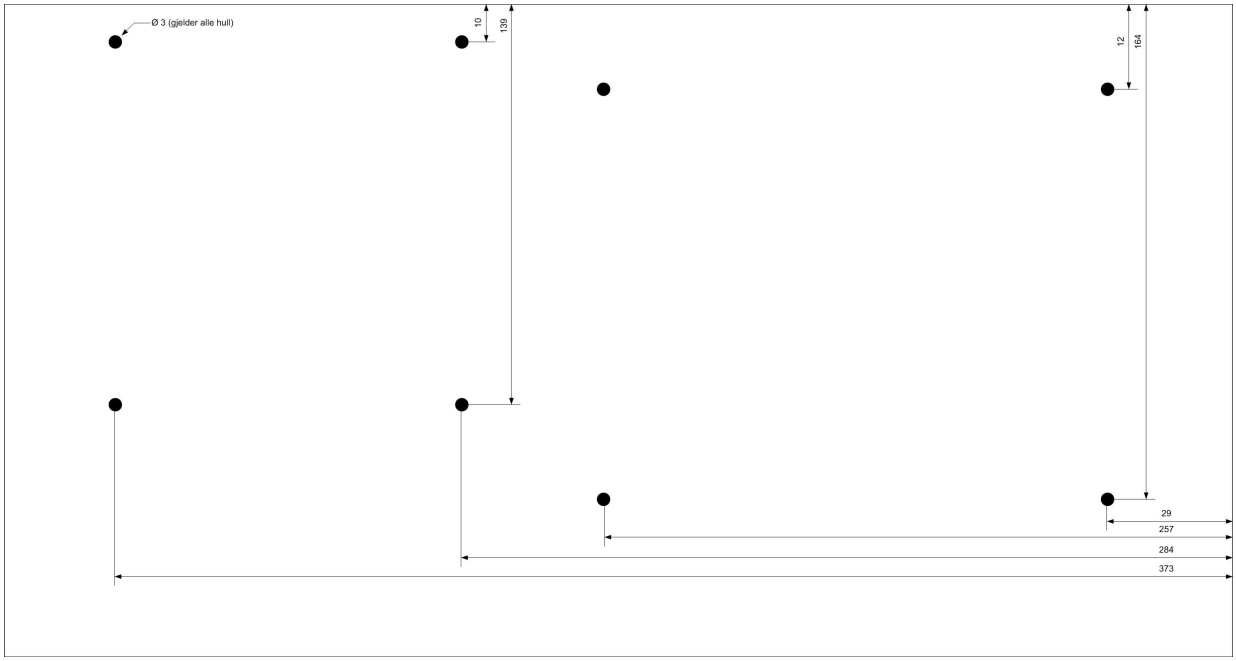
Figur 65: Sensorboksens underside

C.2 Displaykabinett

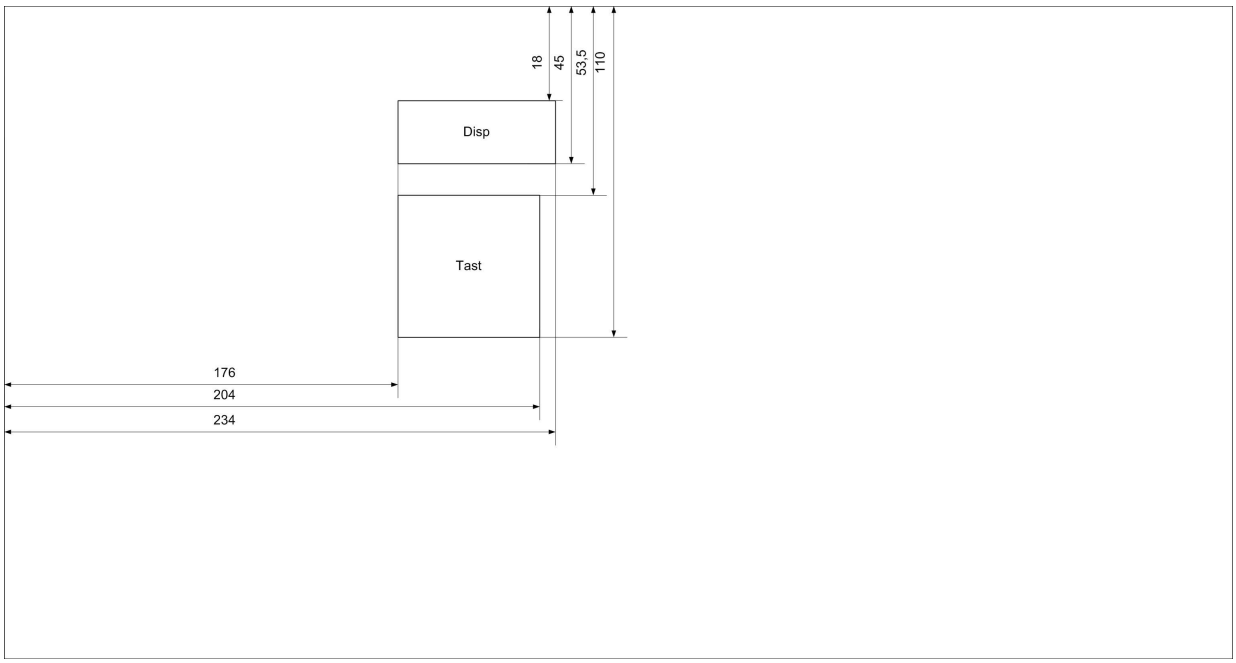
Også her er det laget hull i bakveggen for tilkoblinger, se figur 66. Monteringshull for kortene er vist i figur 67. Vi har valgt å montere display og tastatur direkte på displaykortet fremfor å montere disse til kabinettet. Dette er gjort for å forenkle håndtering av kort ved test og utvikling. På et evt. fremtidig produkt må det legges mer vekt på ergonomi og brukervennlighet. For at display og tastatur alikevel skal være tilgjengelig er det freset ut hull i lokket på kabinettet, disse er vist i figur 68.



Figur 66: Displayboksens bakside



Figur 67: Displayboksens underside



Figur 68: Displayboksen med utskåringer til tastatur og display

D Regnskap

Post nr.	Artikkel nr.	Beskrivelse	Antall	Pris pr. stk.	Sub total
1	AT45DB642	IC-SM Dataflash serial 64 Mb	2	324.7064	649.4128
2		Frakt	1	62	62
3	DS2480B	Maxim Linedriver	5	34.35	171.75
4		Frakt	1	100	100
5	TAI8515	1-Wire Weather Instrument Kit V3.0	1	576.7	576.7
6	TAI8540B	1-Wire Humidity Sensor	1	218.635	218.635
7	TAI8570	1-Wire Pressure Sensor	1	474.5	474.5
8		Frakt	1	305.14	305.14
9	64-347-65	Trimpot 64W 5kohm	12	27	324
10	64-348-07	Trimpot 64W 100kohm	2	29.7	59.4
11	33-154-54	Sikringsklips 5x20	2	1.59	3.18
12	70-003-67	1N4001 diode 1 50V DO41	25	0.446	11.15
13	71-039-22	BC547B trans	25	0.736	18.4
14	48-383-14	Koppl. Plint 2-pol 3,81 mm	4	7.35	29.4
15	75-576-22	Grafisk LCD seriell 120x3	2	1050	2100
16	35-678-31	Telefontastatur 4x4	10	59.6	596
17	42-707-08	USB type A	2	15.2	30.4
18	43-702-19	Stiftlist rak 1x36-pol	10	8.66	86.6
19	43-832-20	AMPMODU II Hylshus 1x6pol	2	3.32	6.64
20	43-832-38	AMPMODU II Hylshus 1x8pol	2	4.15	8.3
21	43-653-18	Kabelhylsdon 6-pol	4	16.4	65.6
22	43-108-01	Hylsdon 10-polig	2	13.7	27.4
23	43-116-35	Header lågprofil 20p	30	13.9	417
24	43-116-01	Header lågprofil 10p	2	14.8	29.6
25	43-653-18	Kabelhylsdon 6-pol	2	16.4	32.8
26	74-566-19	80,0 MHz ACMOS/TTL osc.	4	177	708
27	43-704-25	Stiftlist rak 2x36 forhøyd	10	30.5	305
28	43-709-87	Kortslutn. Bygel m. handtag	100	1.21	121
29	37-433-90	Mosfet-rele PVA3354N	2	74.3	148.6
30	73-221-26	TSL250R ljus till spänning	4	33.1	132.4
31	73-772-52	DS9503P ESD-skydd TSOC6	4	17.8	71.2
32	73-361-26	MAX4202ESA Buffer	4	40	160
33	42-695-85	Modularplugg 6/6 rund kabel	25	5.86	146.5
34	42-696-50	Modularjack PC 6/6 vinkl.	25	10.8	270
35	55-784-30	Mångledare 3x2x0,36mm2	10	17.7	177
36	42-707-40	USB type B	10	10.7	107
37	25-576-19	USB 2.0-kabel A-B 1,0m	2	45.4	90.8

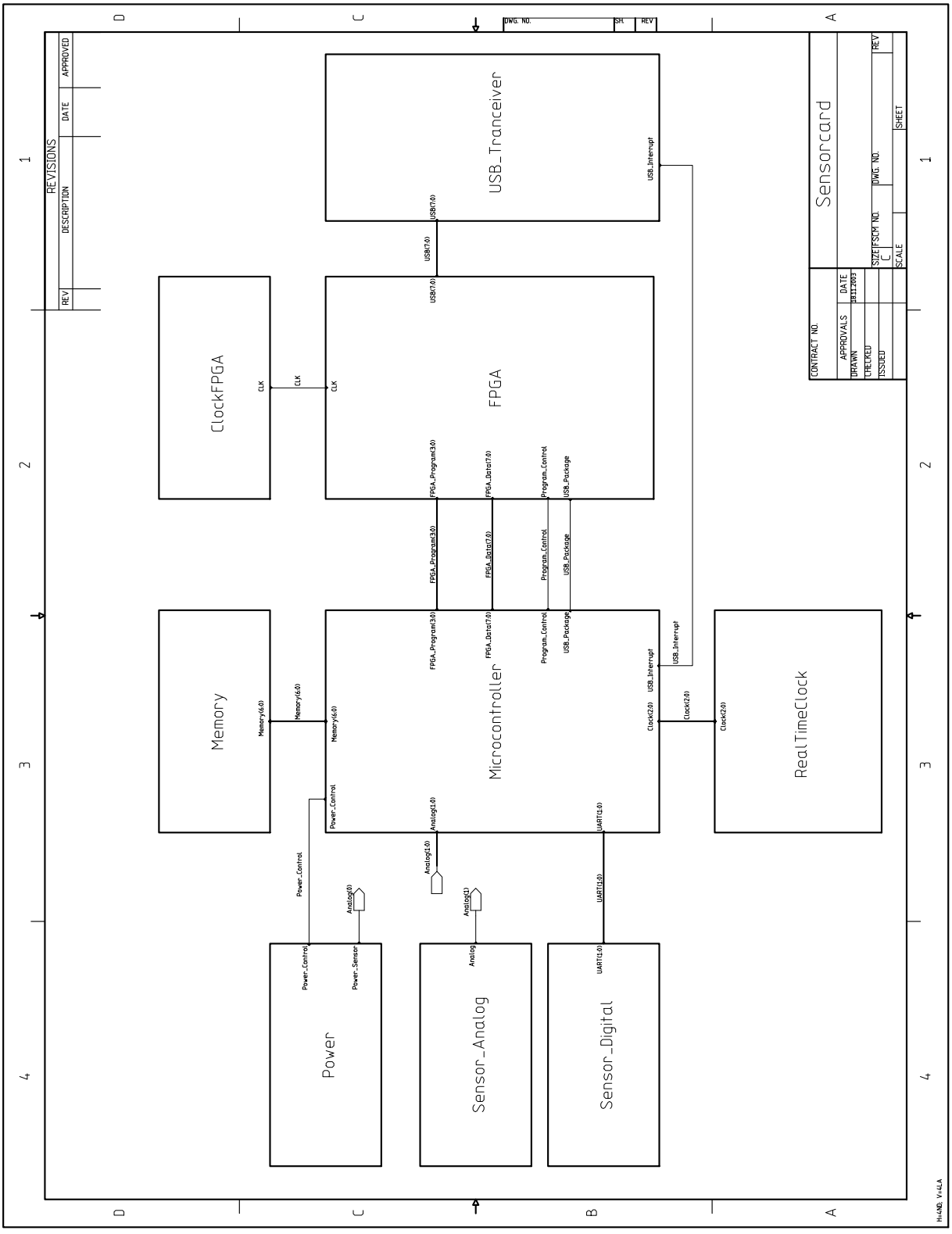
Post nr.	Artikkel nr.	Beskrivelse	Antall	Pris pr. stk.	Sub total
38	73-321-25	LM258N dubbel op-amp DIL8	10	4.46	44.6
39	74-501-82	4,0000 MHz krystall	5	10.9	54.5
40	74-530-04	32,768 kHz klokkekrystall	2	7.61	15.22
41	73-769-81	DS1302 ser. Tidräkn. DIL8	2	43.3	86.6
42	69-273-96	Li. Batt. CR2430THA	4	37.3	149.2
43	44-055-51	Stiftdon FH-DS-09P	2	7.52	15.04
44	35-038-02	Strömst. 1-pol H8600VBBB01	4	11.4	45.6
45	35-656-11	Tang bordströmst. SKHHAQ	10	5.69	56.9

E Skjemategninger

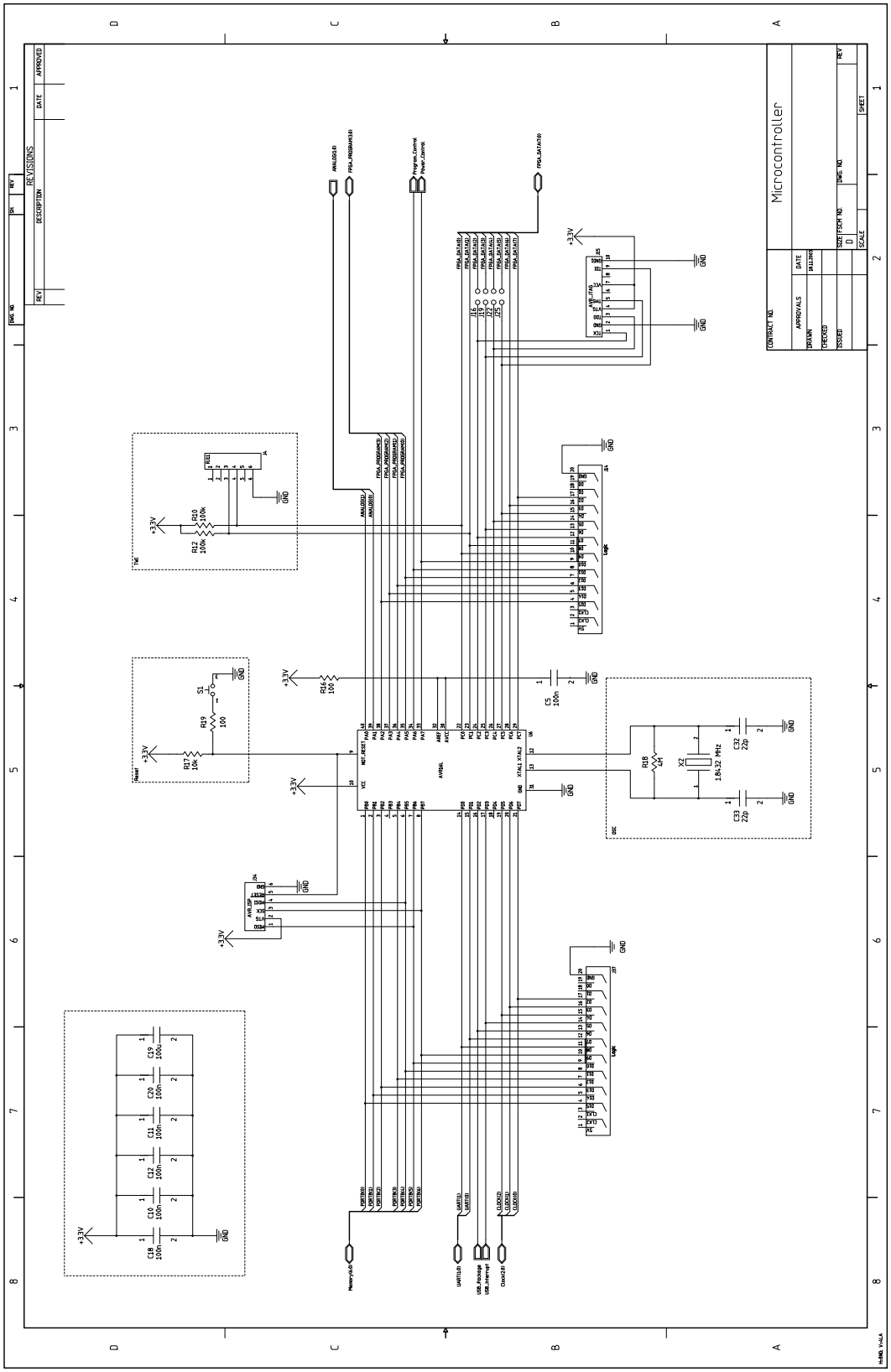
E.1 Sensorkort

item	qty	part number	description	references
1	1	AT45DB642		U9
2	1	AVR_ISP		J34
3	1	AVR_JTAG		J15
4	1	AVR16L		J15
5	2	BC547B	Transistor, NPN BJT	Q1-Q2
6	2	Battericelle		B1-B2
7	3	Cap_lyt_liten	Electrolytic Capacitor	C1-C2,C8
8	35	Cap_smd	Capacitor	C3-C7,C9-C38
9	1	DS1302		U8
10	1	DS2480B		U3
11	1	DS9503		D1
12	1	FPGA_Clock_smd		X4
13	2	FPGA_JTAG		J29-J30
14	1	FPGA_Prog		J31
15	19	Jumper	Jumper	J8-J12,J16-J28,J36
16	1	Krystall_AVR	Crystal	X2
17	8	LED	LED, Photoemissive Diode	D2-D9
18	3	Logic		J14,J37-J38
19	1	MAX4202	Buffer	U2
20	1	PDIUSBP11A		U5
21	3	RJ11		J4-J6
22	38	Res_smd	Resistor	R1-R38
23	1	Spartan2E		U7
24	1	USB_kvadrat		J7
25	1	XC18V02		U4
26	2	push_button		S1,S3
27	1	2-pin		J3
28	1	3-pin		J1
29	4	3-pin-korts		J13,J32-J33,J35
30	1	4-pin		J2
31	1	8_switch		S2
32	1	32.768kHz	Crystal	X3
33	1	74x14	Inverter	U1

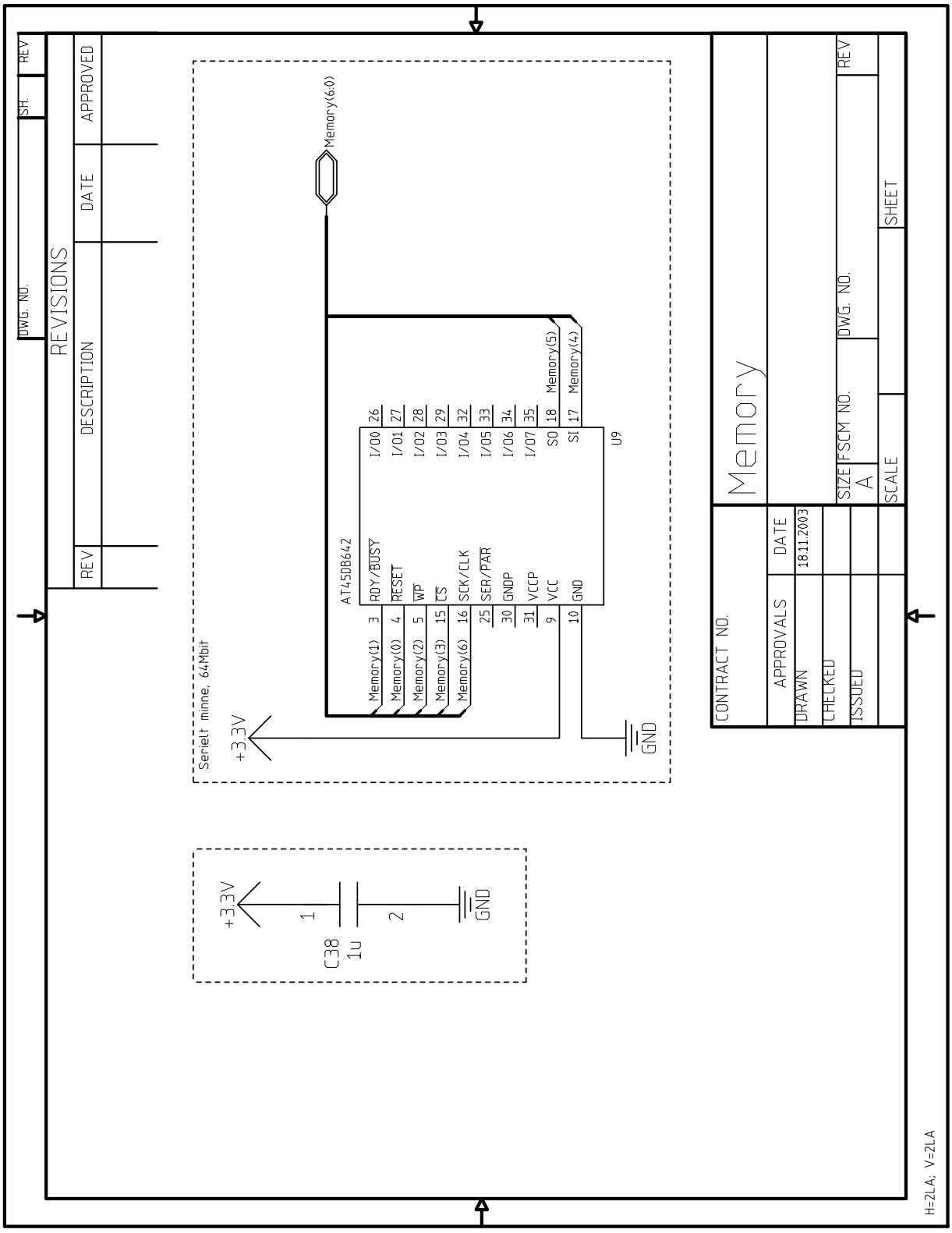
Tabell 21: Sensorkort - Materialer



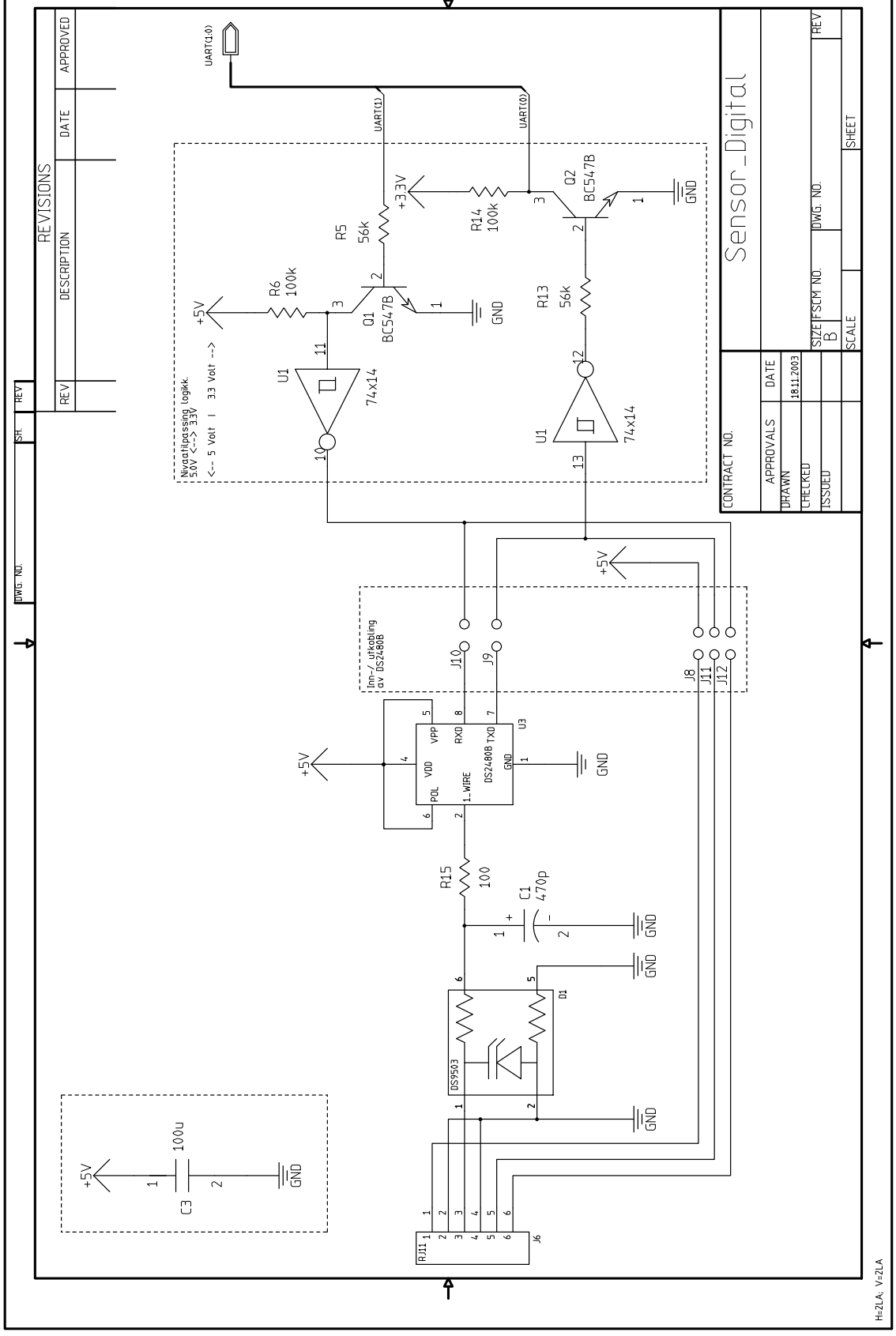
Figur 69: Sensorkort - Toppnivå



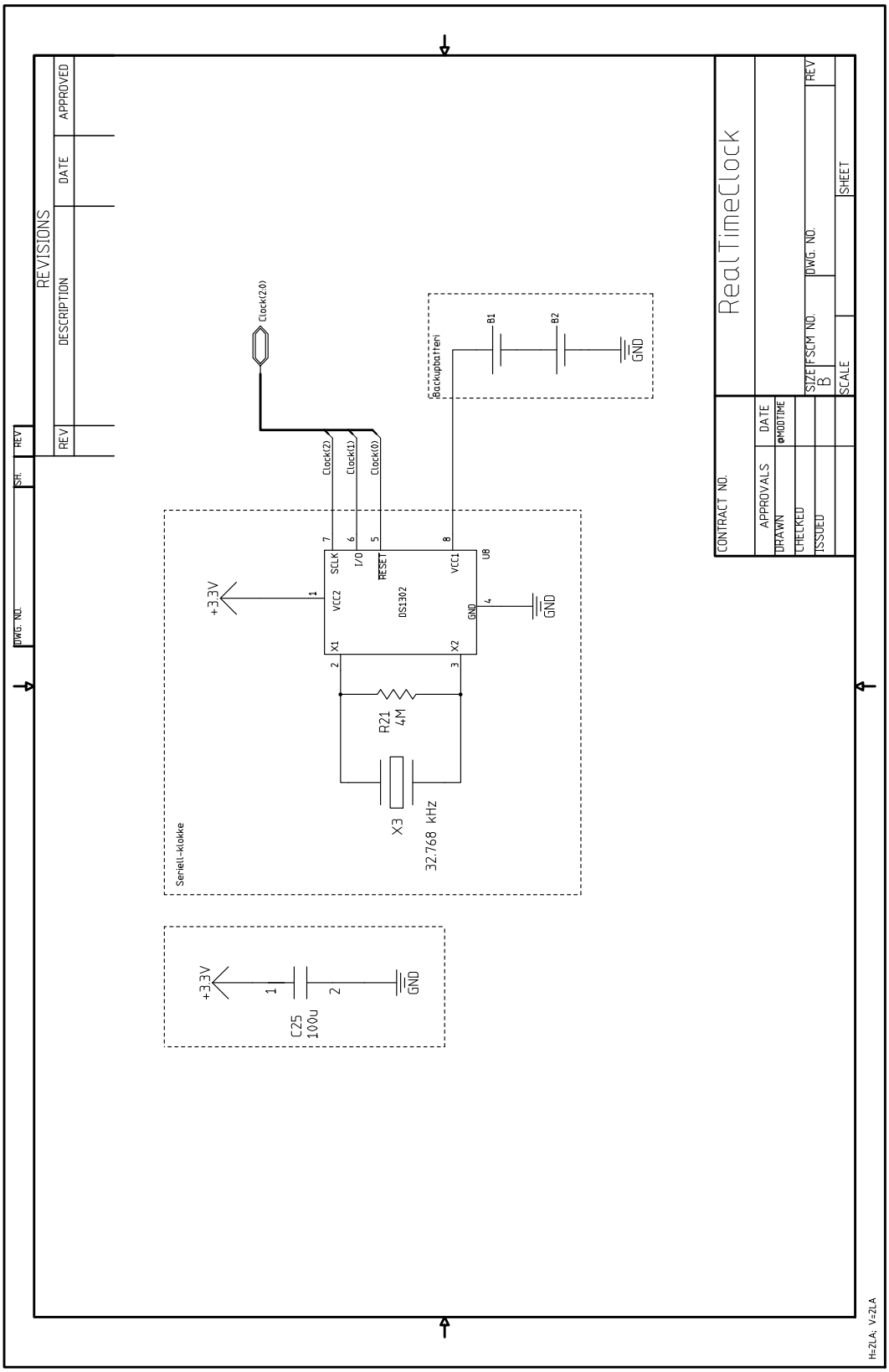
Figur 70: Sensorkort - AVR



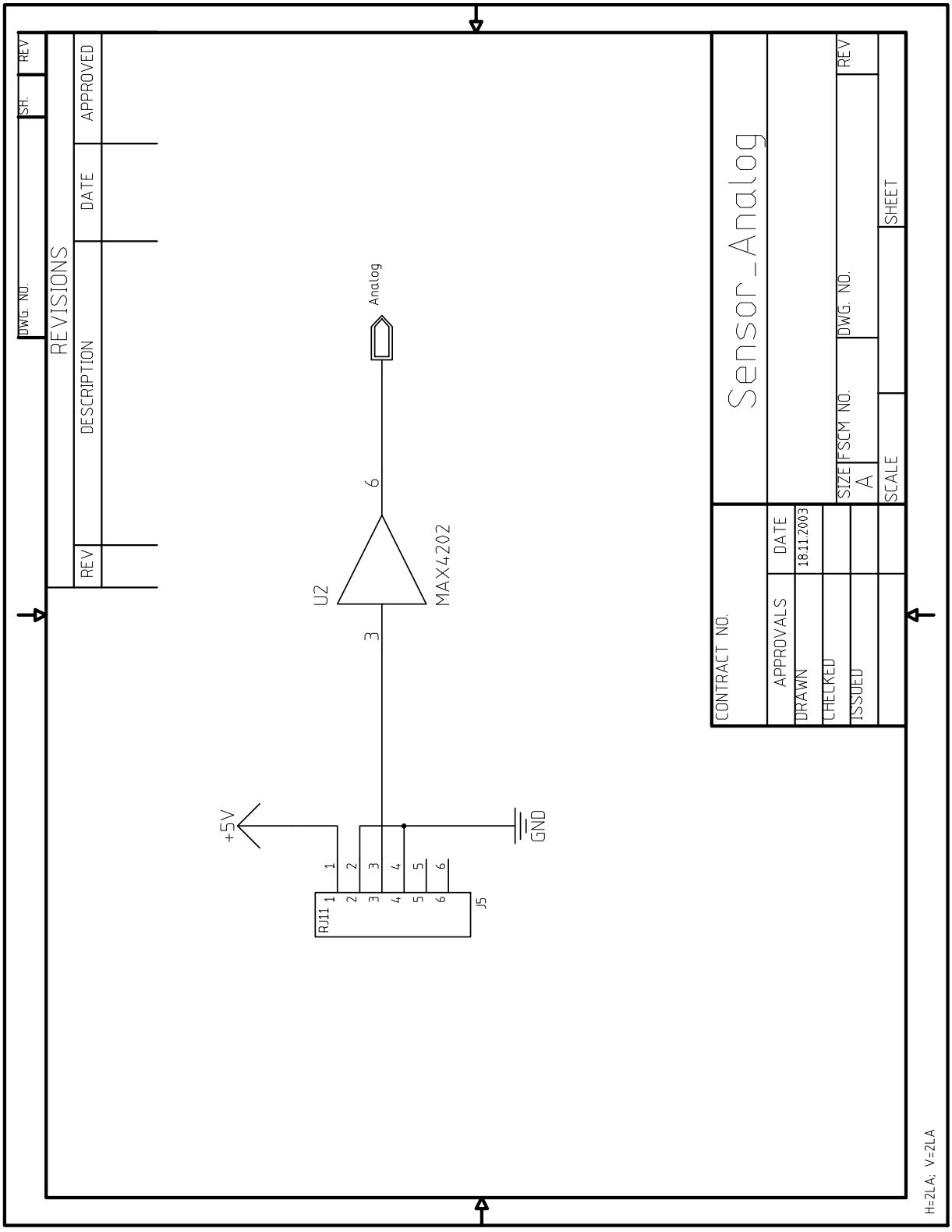
Figur 71: Sensorkort - Minne



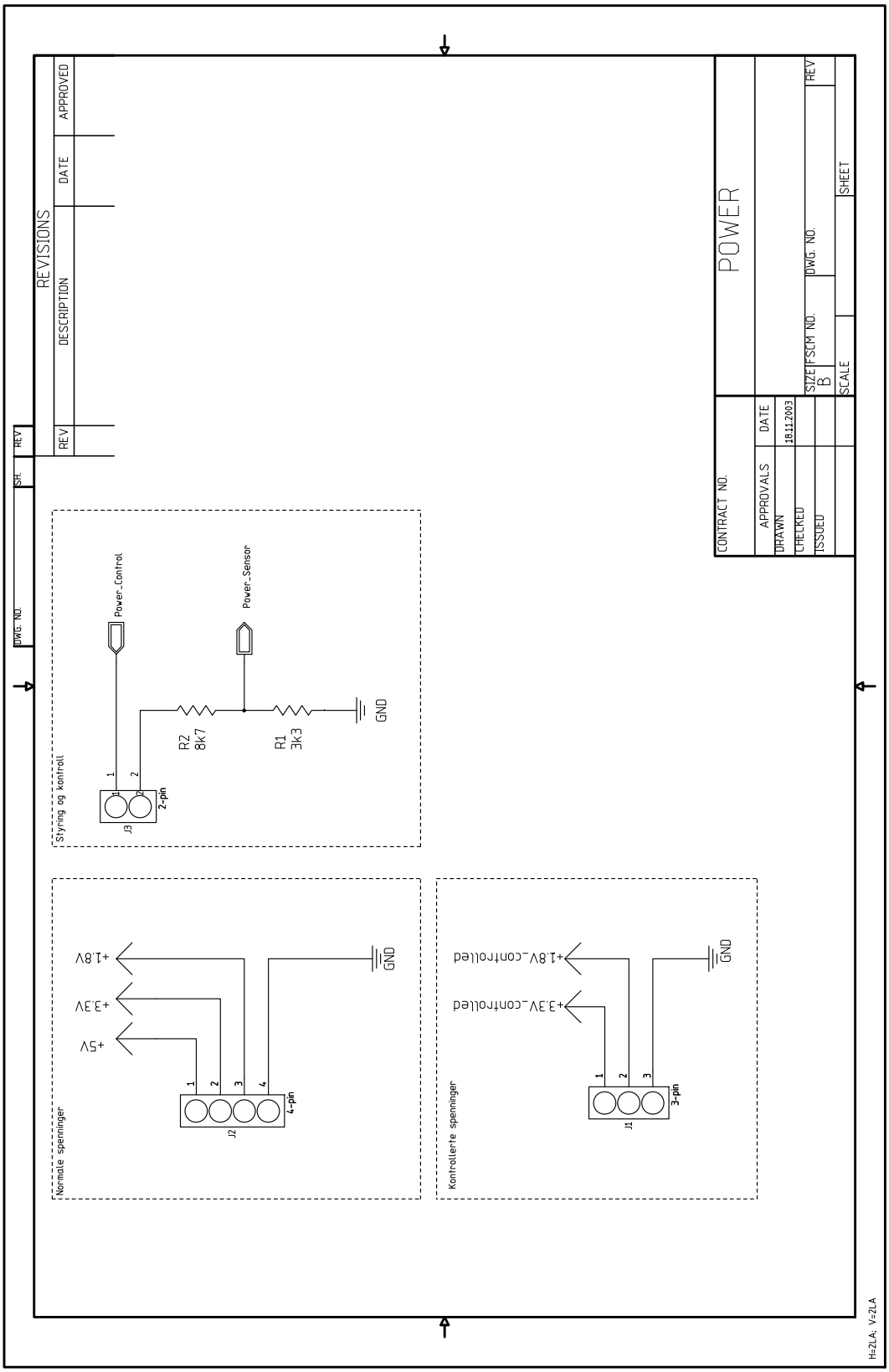
Figur 72: Sensorkort - Linedriver



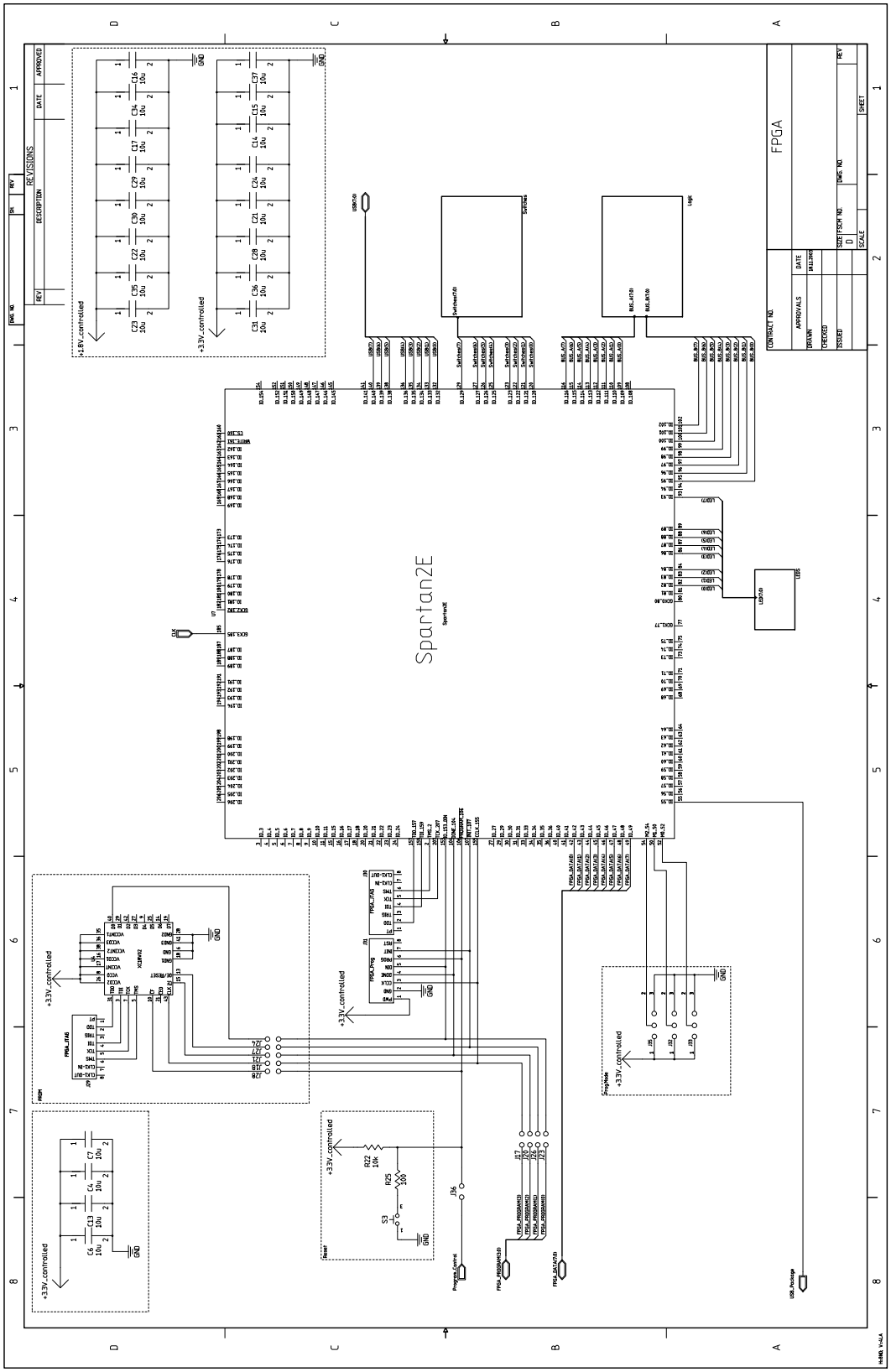
Figur 73: Sensorkort - Sanntidsklokke



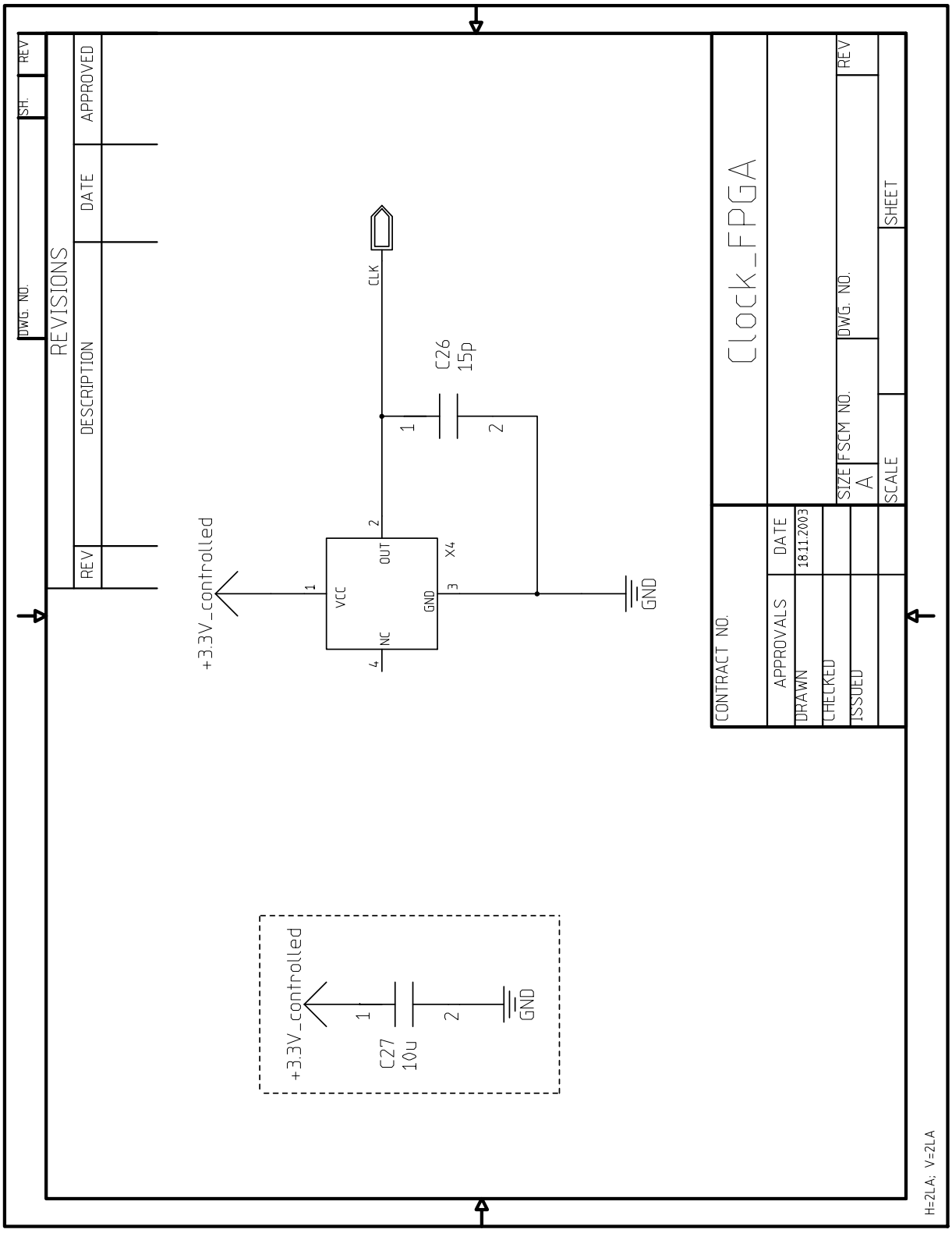
Figur 74: Sensorkort - Analoge Sensorer



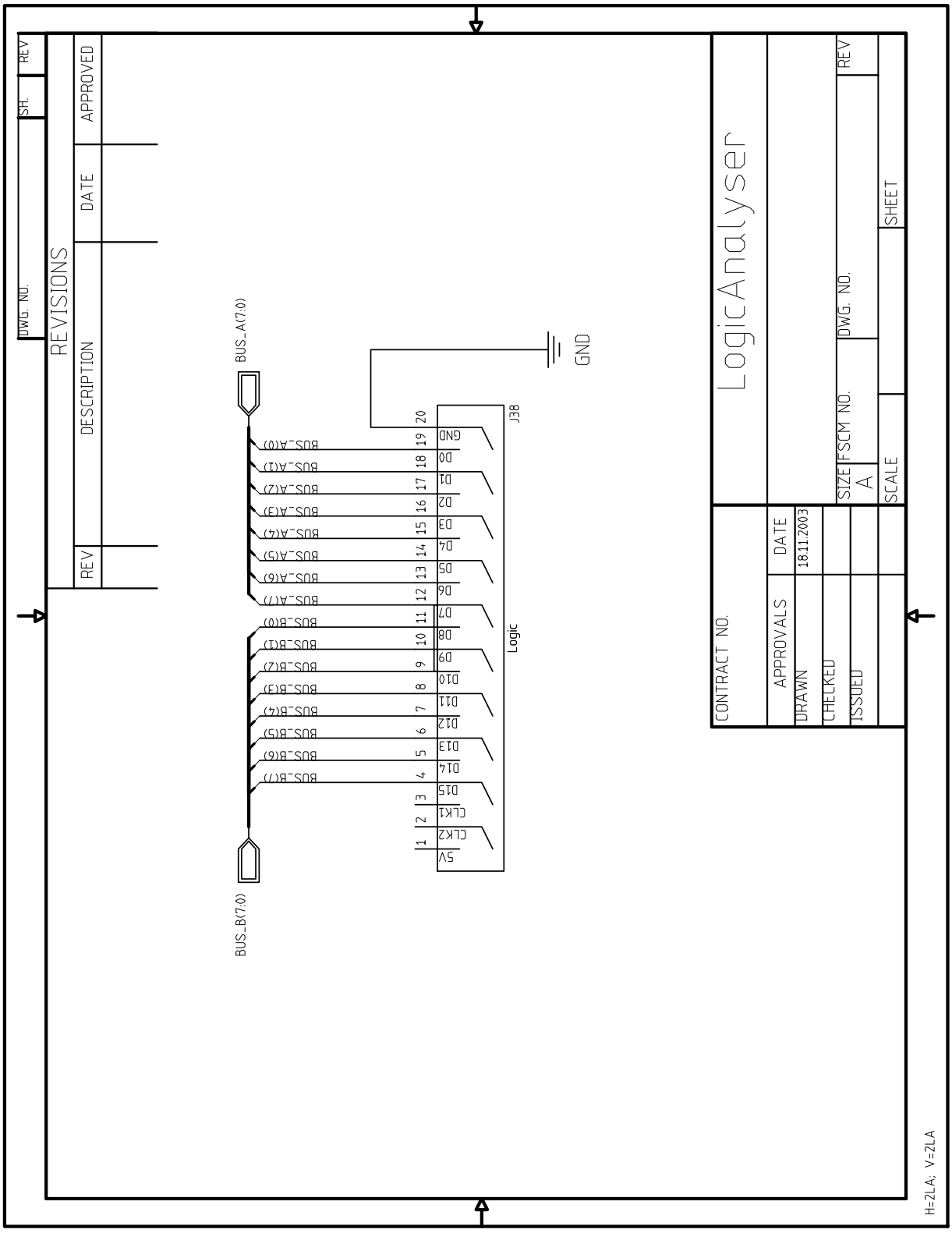
Figur 75: Sensorkort - Powertilkobling



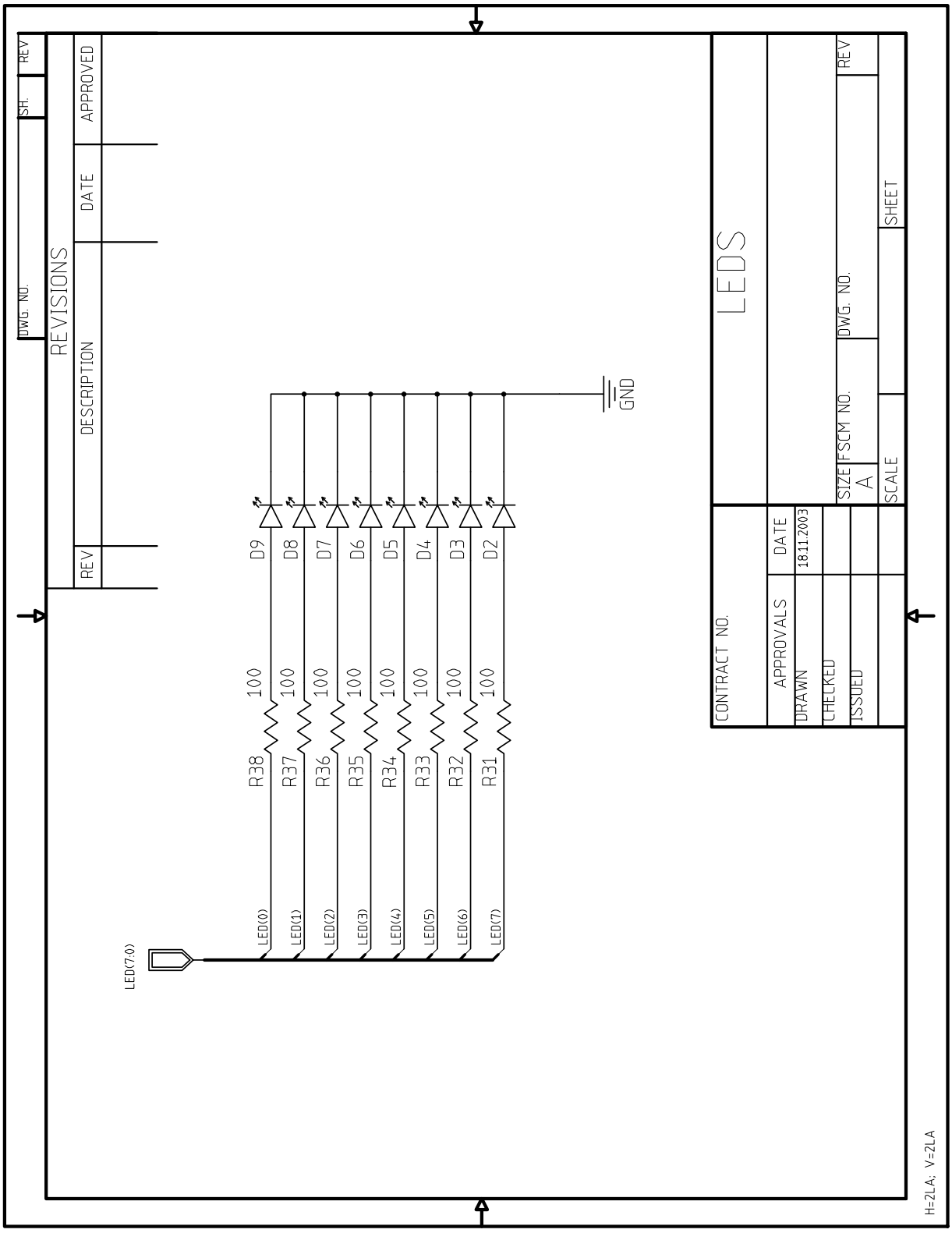
Figur 76: Sensorkort FPGA



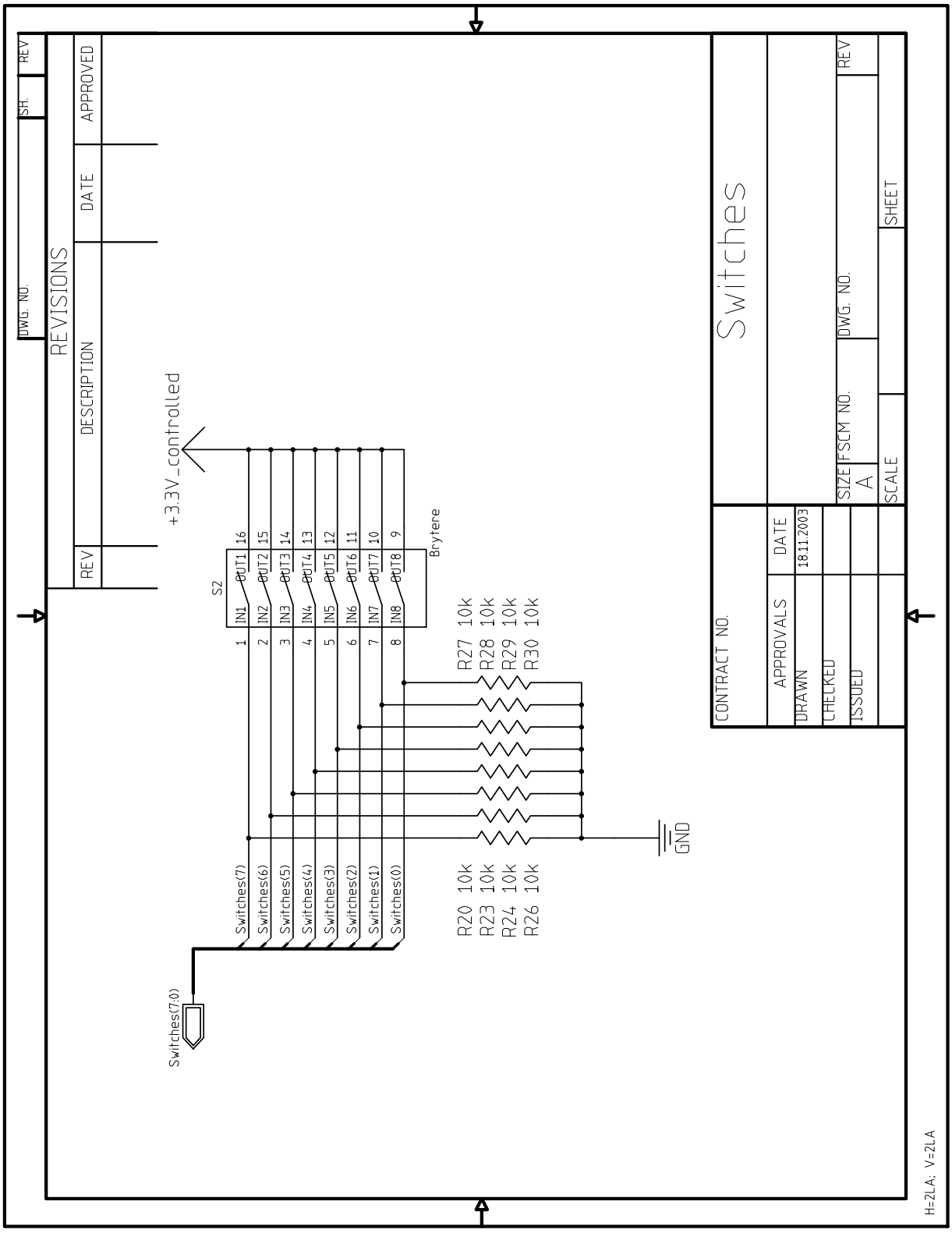
Figur 77: Sensorkort - FPGA klokke



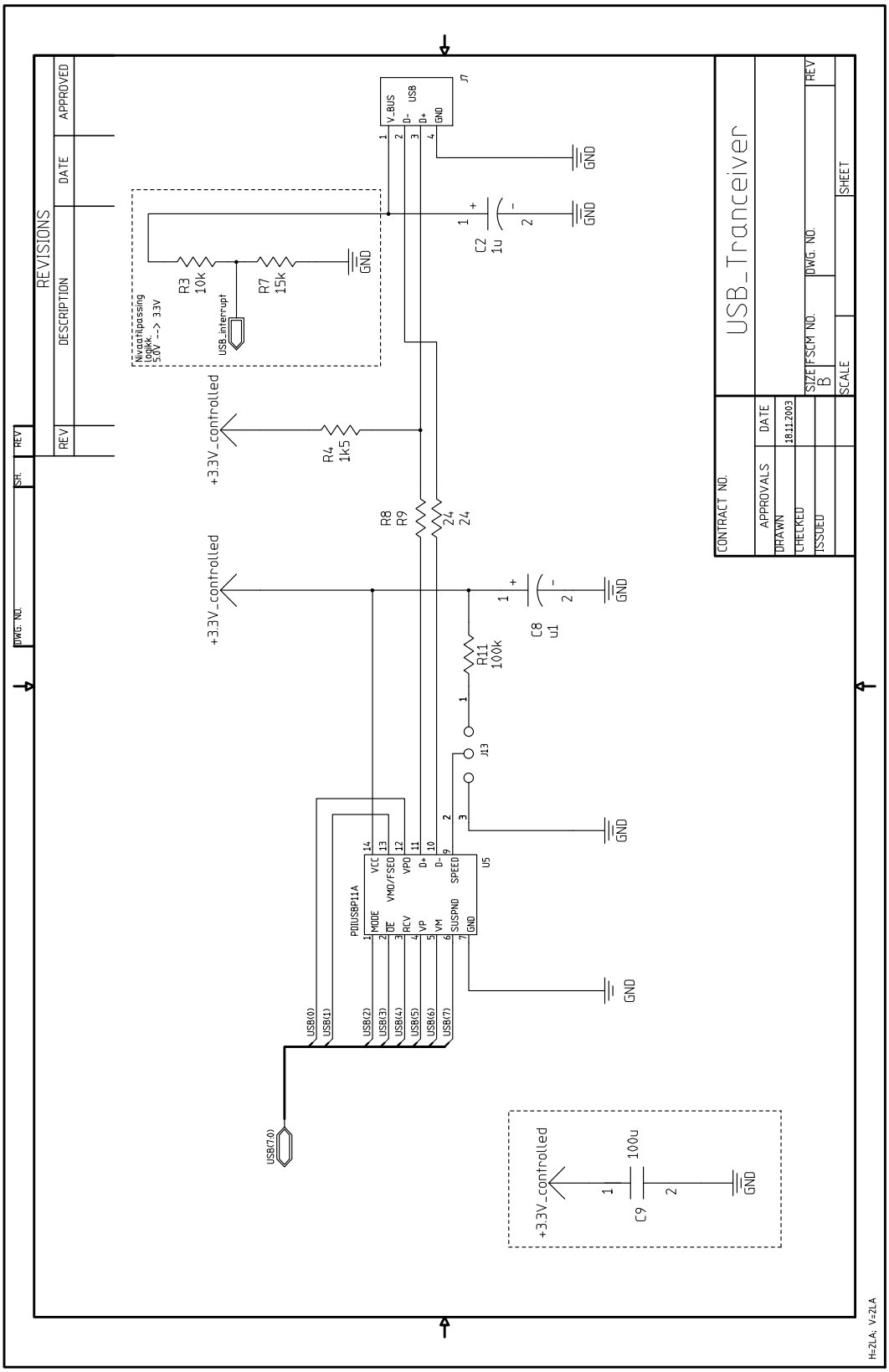
Figur 78: Sensorkort - Logikkanalysator FPGA



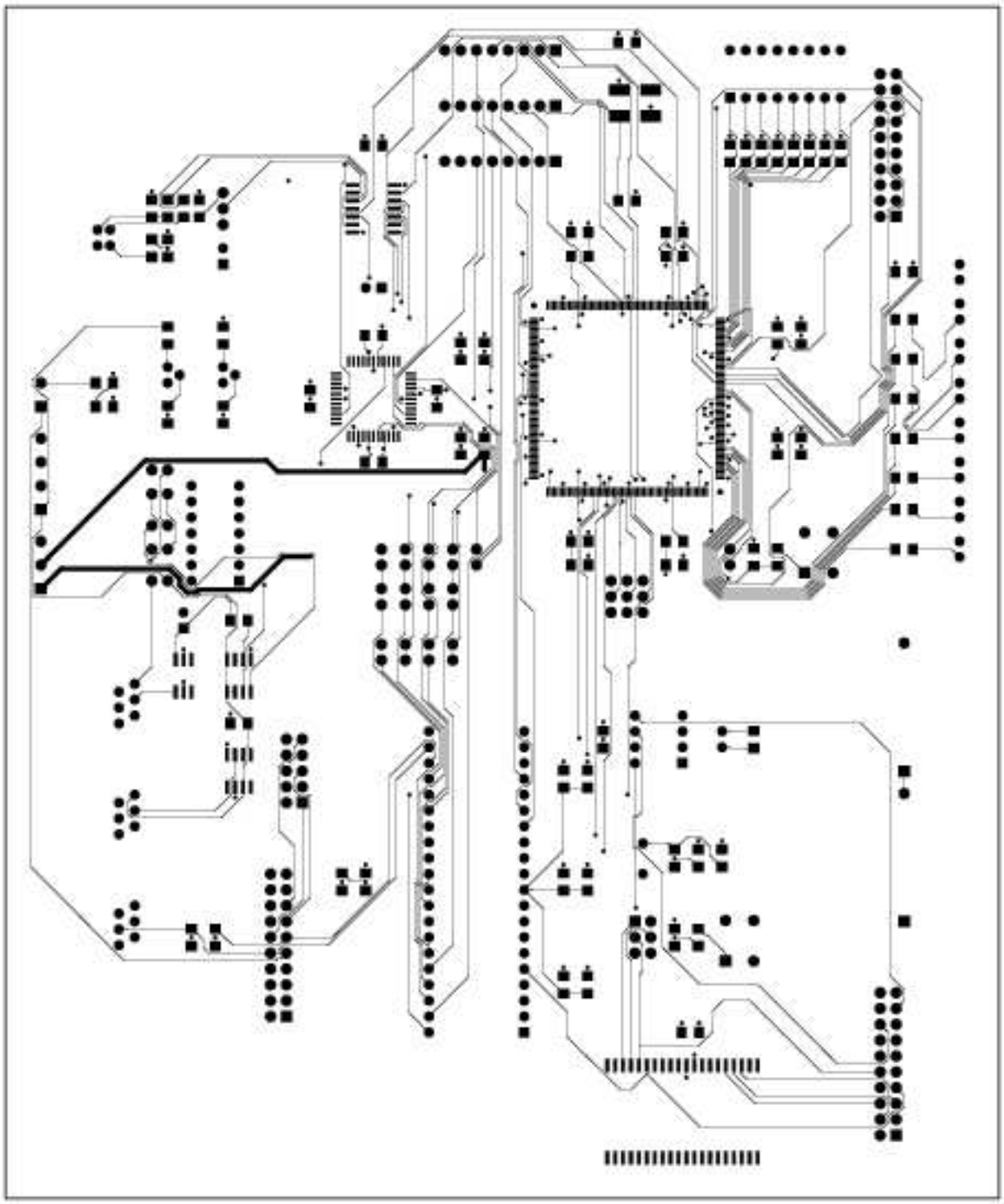
Figur 79: Sensorkort - Lysdioder



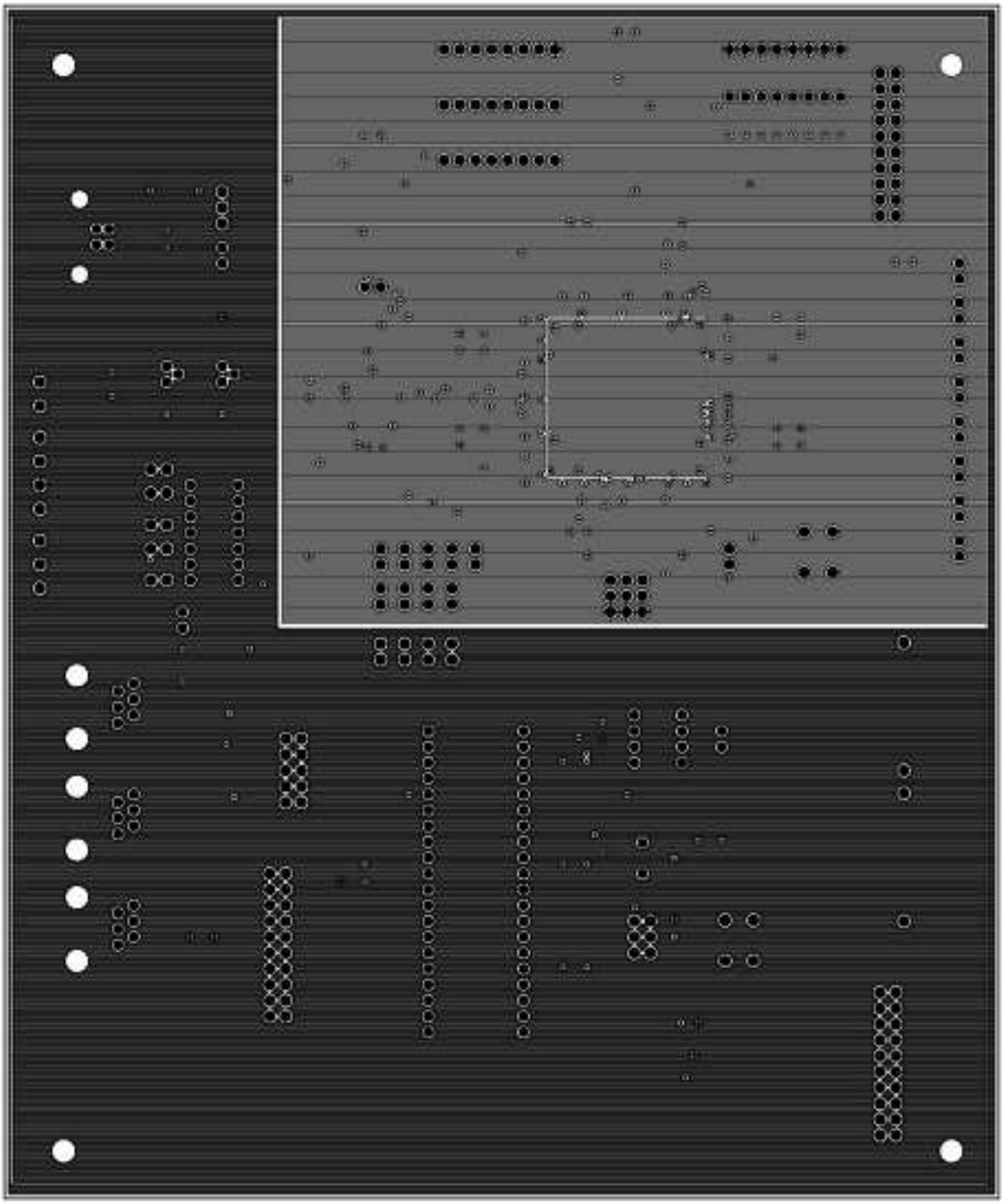
Figur 80: Sensorkort - Brytere



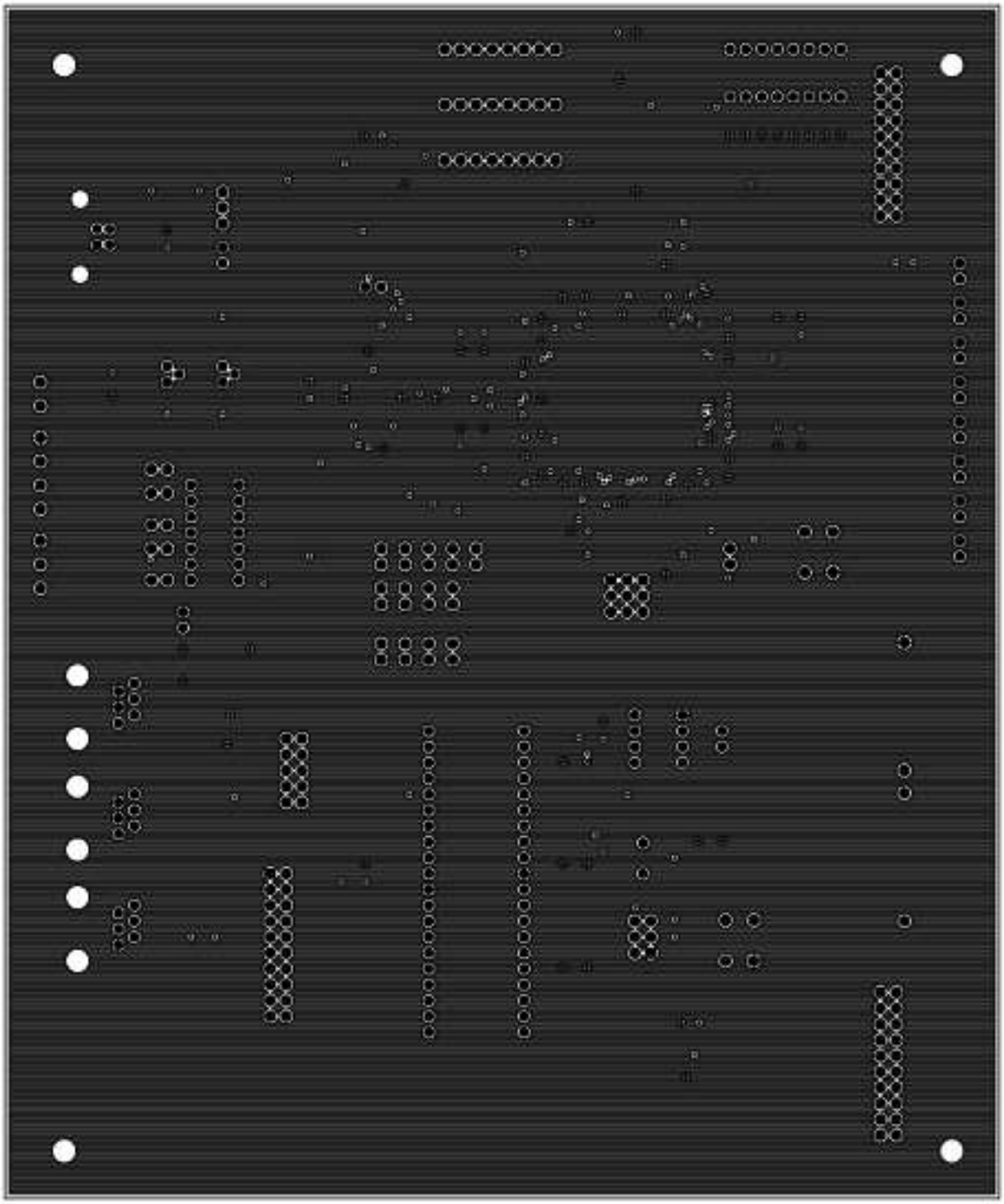
Figur 81: Sensorkort - USB-transceiver



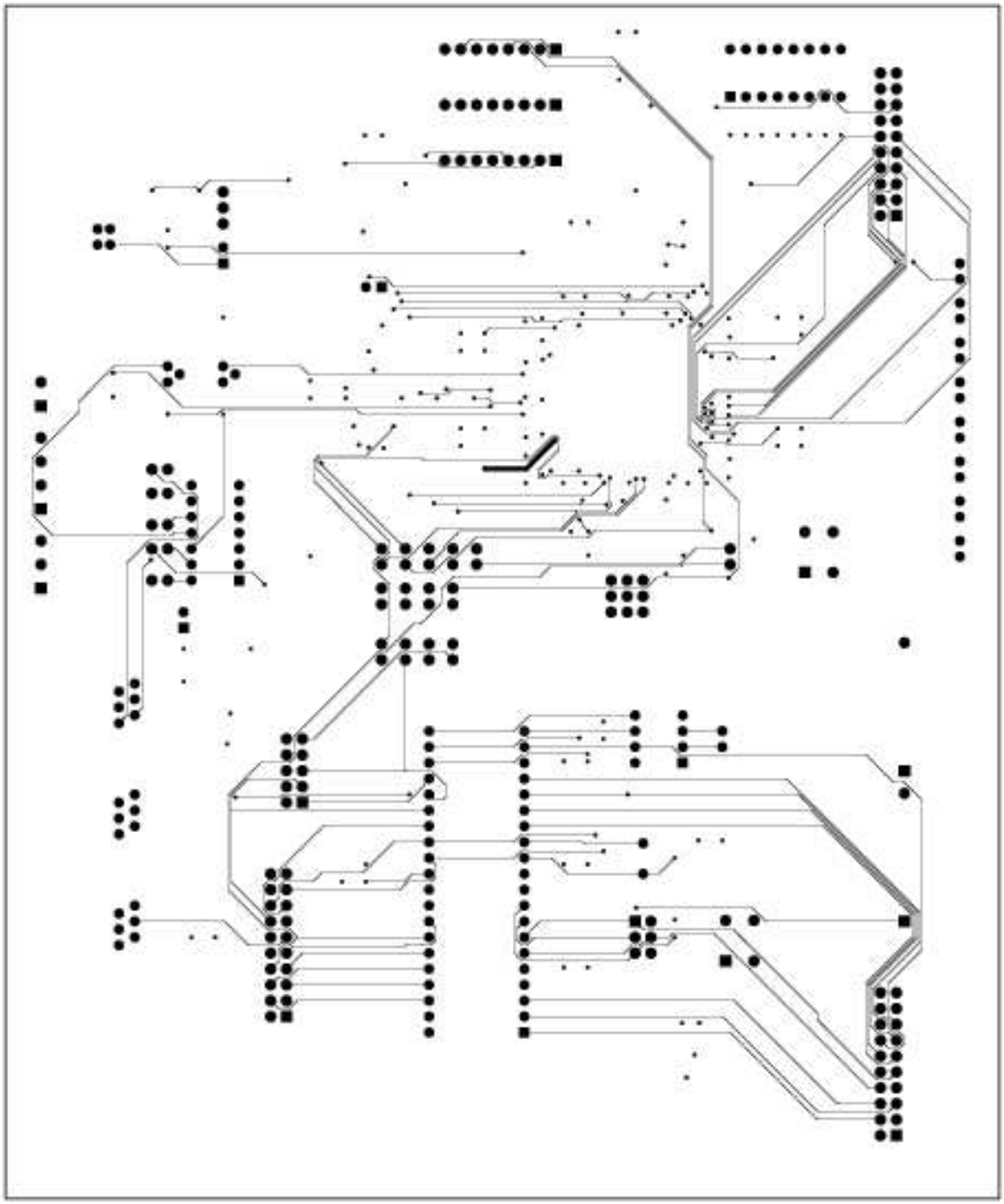
Figur 82: Sensorkort - PCB lag 1



Figur 83: Sensorkort - PCB lag 2



Figur 84: Sensorkort - PCB lag 3

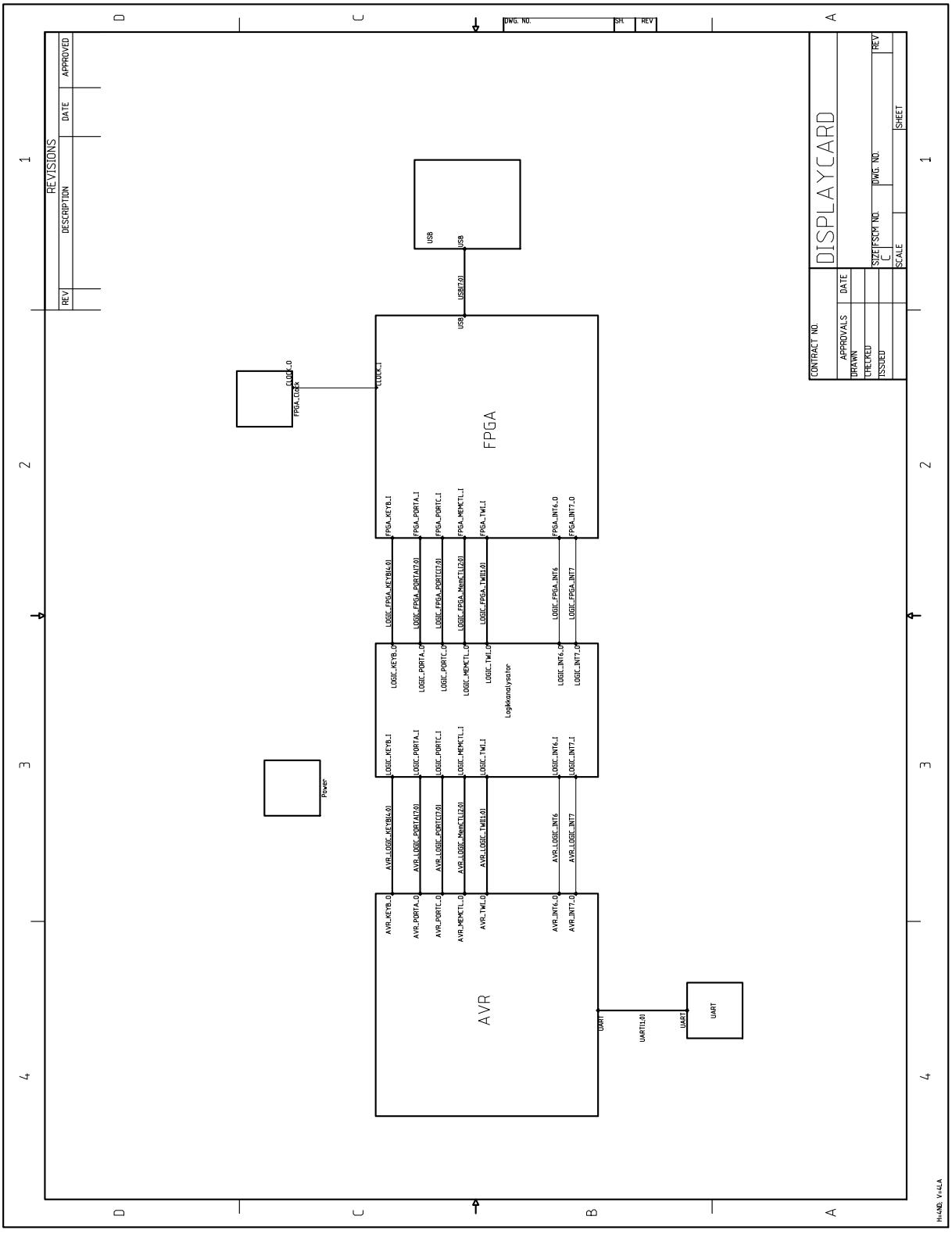


Figur 85: Sensorkort - PCB lag 4

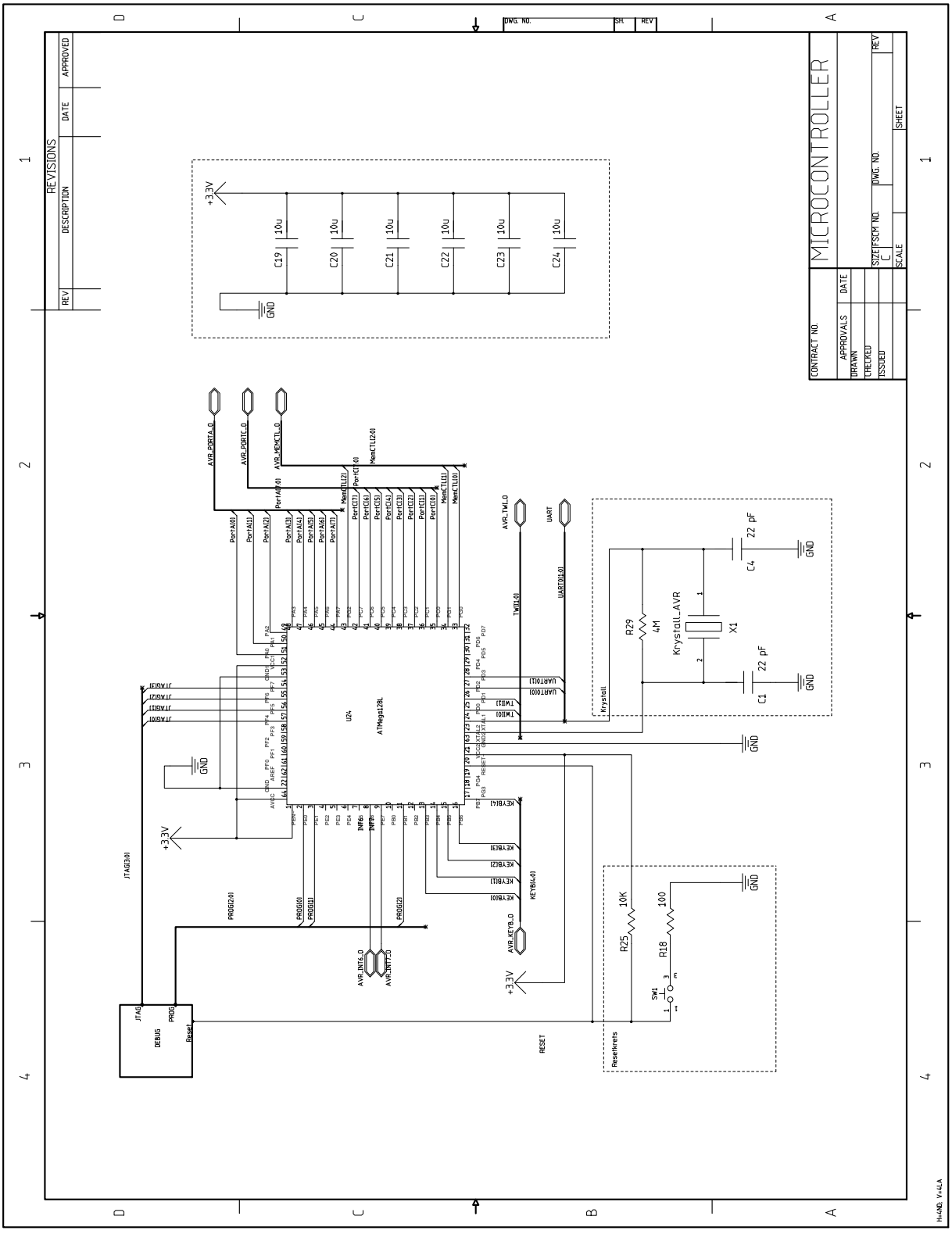
Item	Qty	Part number	Description	References
1	1	ATMega128L		U24
2	1	AVR_JTAG		J1
3	1	BC547B	Transistor, NPN BJT	Q1
4	41	Cap_smd		C1-C41
5	1	FPGA_Clock_smd		X2
6	1	FPGA_JTAG		U10
7	1	Keyb_Conn		U13
8	1	Keyboard		U17
9	1	Krystall_AVR	Crystal	X1
10	16	LED	LED, Photoemissive Diode	D1-D16
11	2	MEM_ADDR_DATA_CONN		U14-U15
12	1	MEM_CTL_CONN		U16
13	1	Mode_Con		U22
14	1	PDIUSBP11A		U6
15	1	Prog_Con		U18
16	1	RJ11		U2
17	30	Res_smd	Resistor	R1-R30
18	1	Spartan2E		U21
19	2	TWI_CON		U5,U23
20	1	UART_CONN		U12
21	1	XC18V02		U25
22	2	push_button		SW1-SW2
23	1	usb_a		U7
24	1	4-pin		J2
25	1	74x14	Inverter	U26

Tabell 22: Displaykort - Materialer

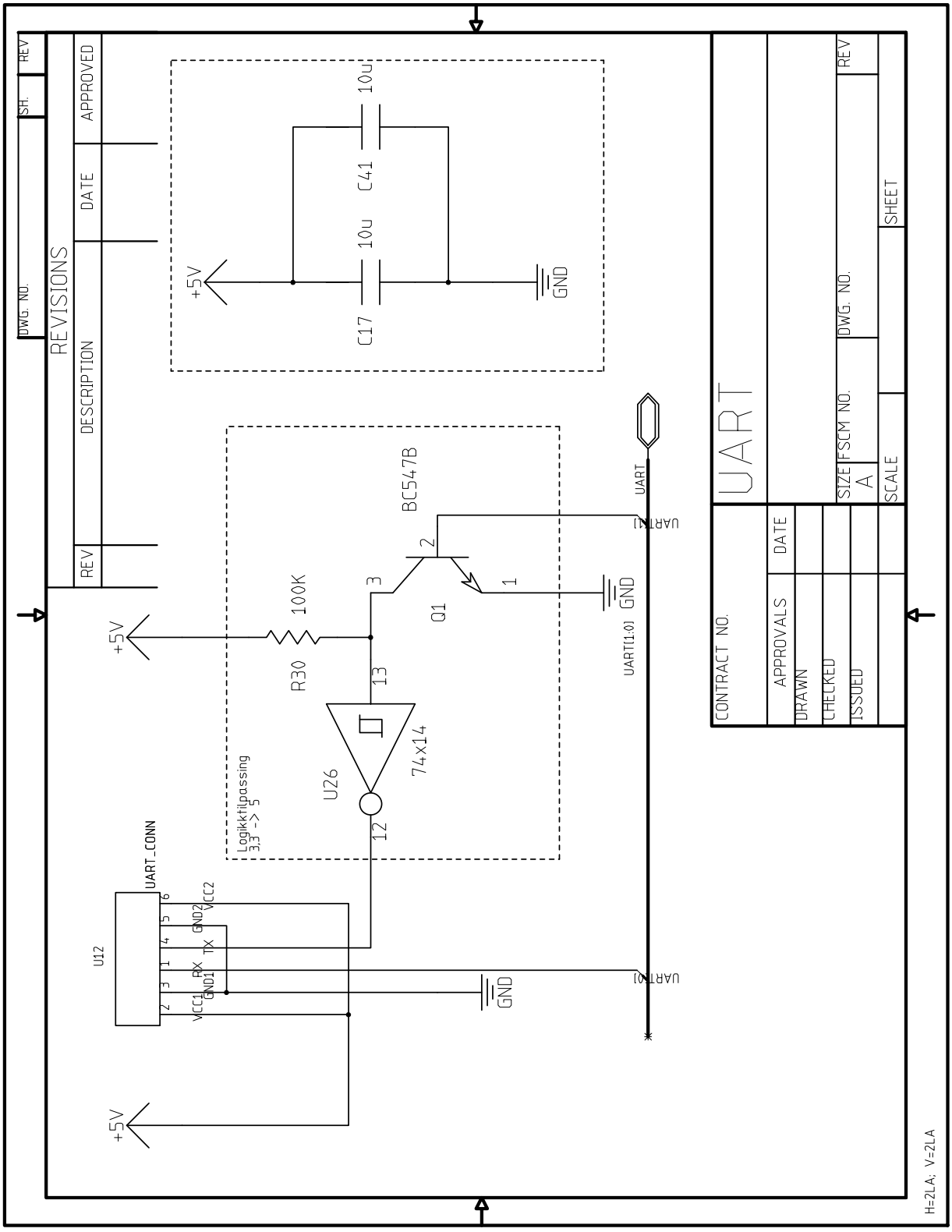
E.2 Displaykort



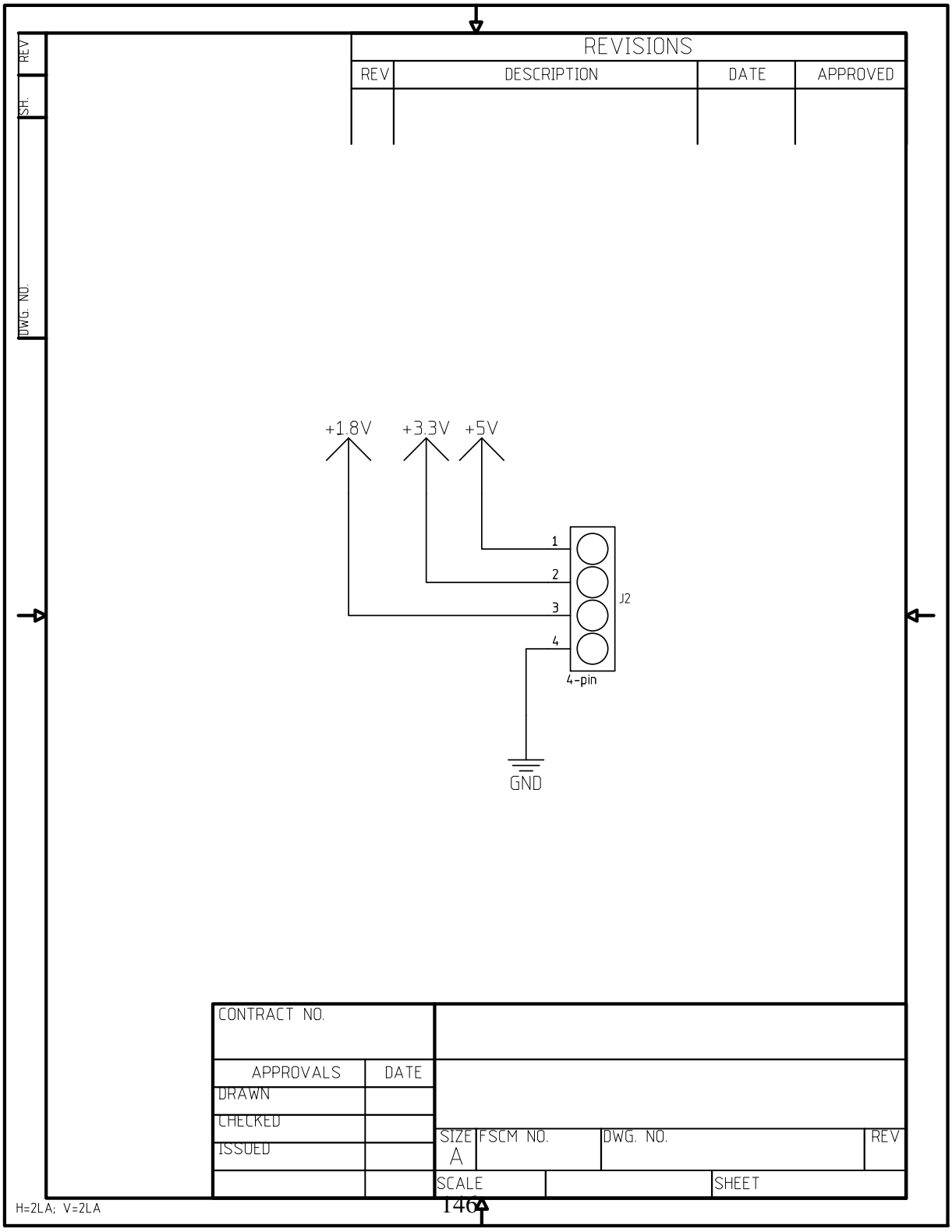
Figur 86: Displaykort - Toppnivå



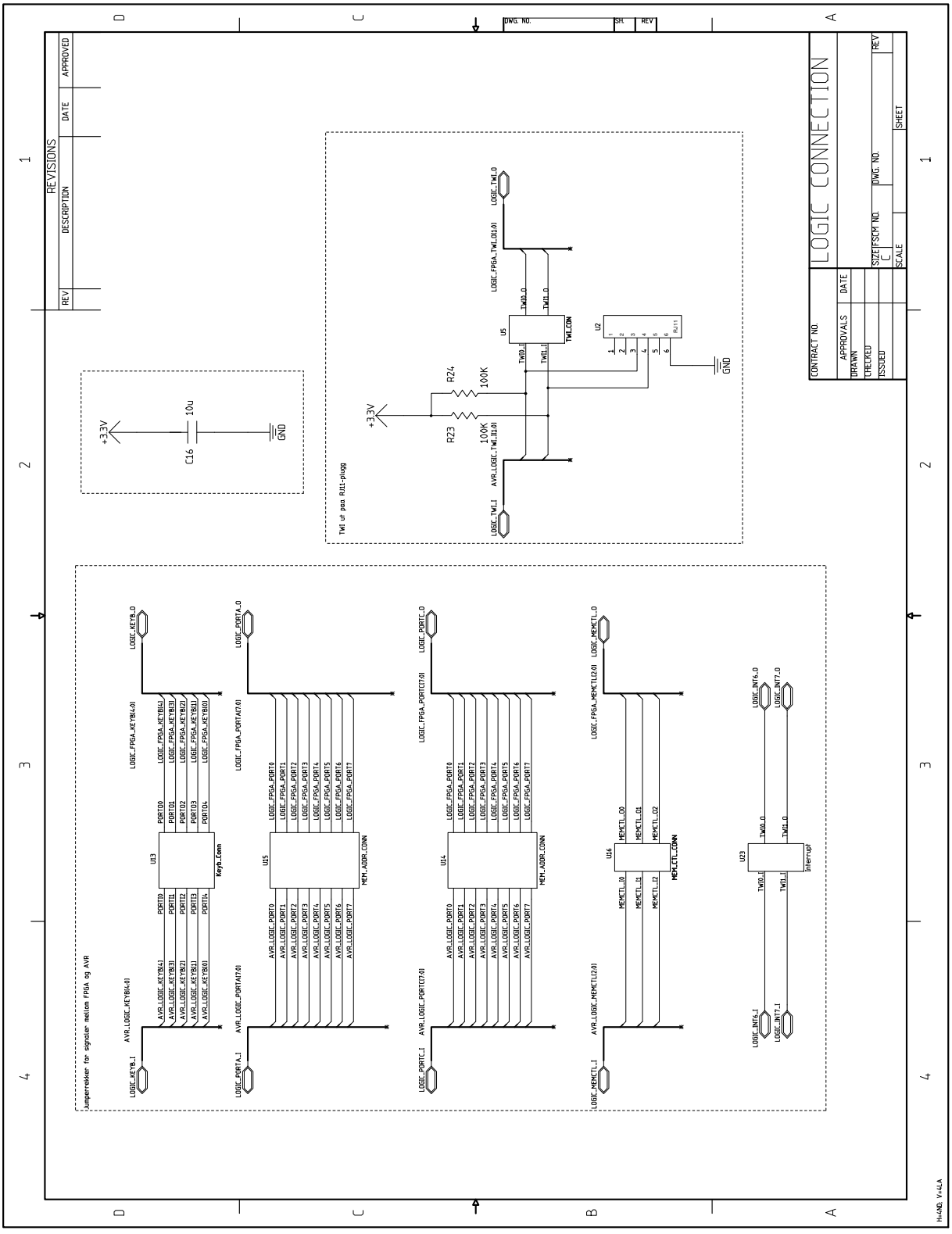
Figur 87: Displaykort - AVR



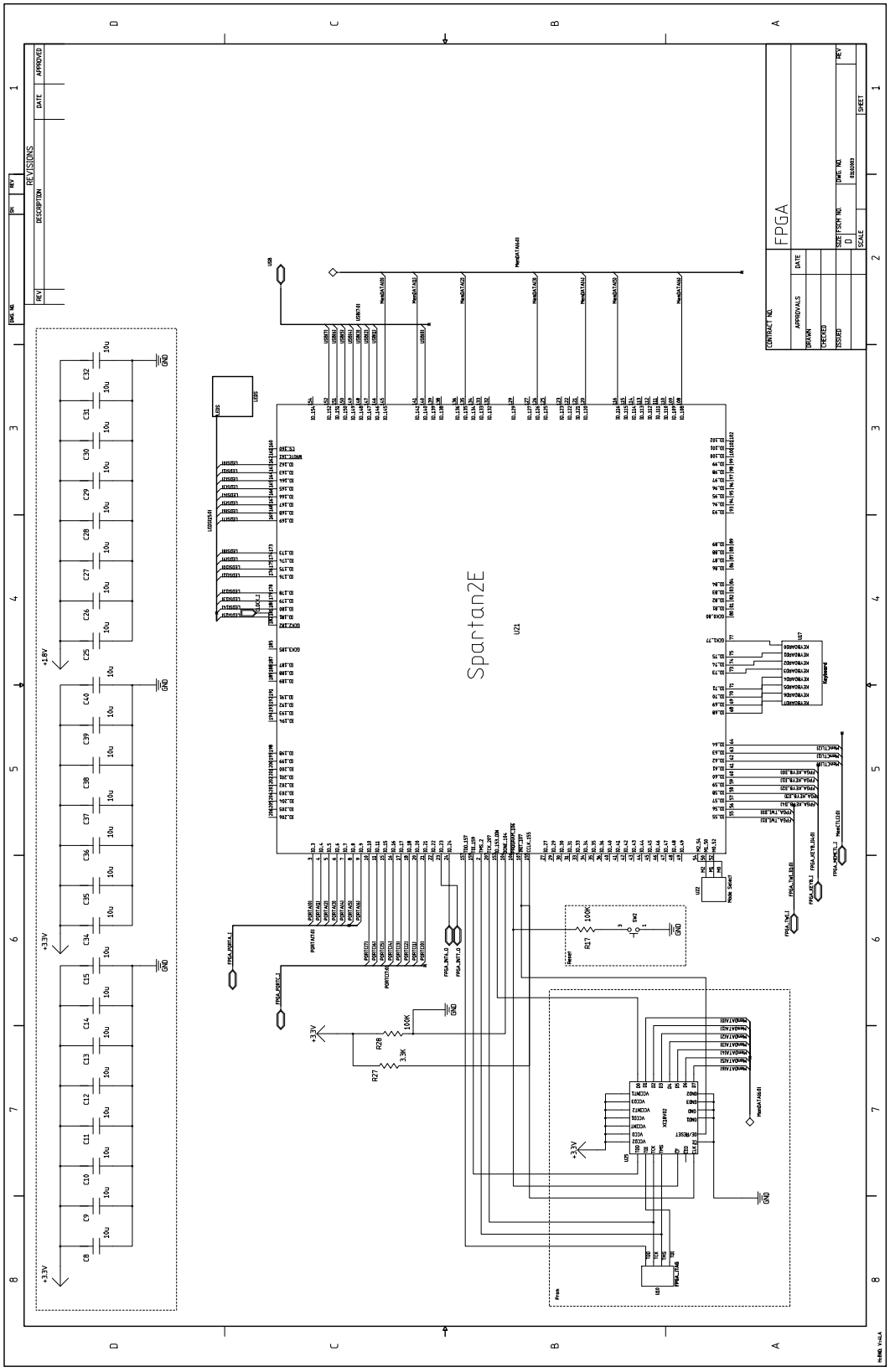
Figur 88: Displaykort - UART



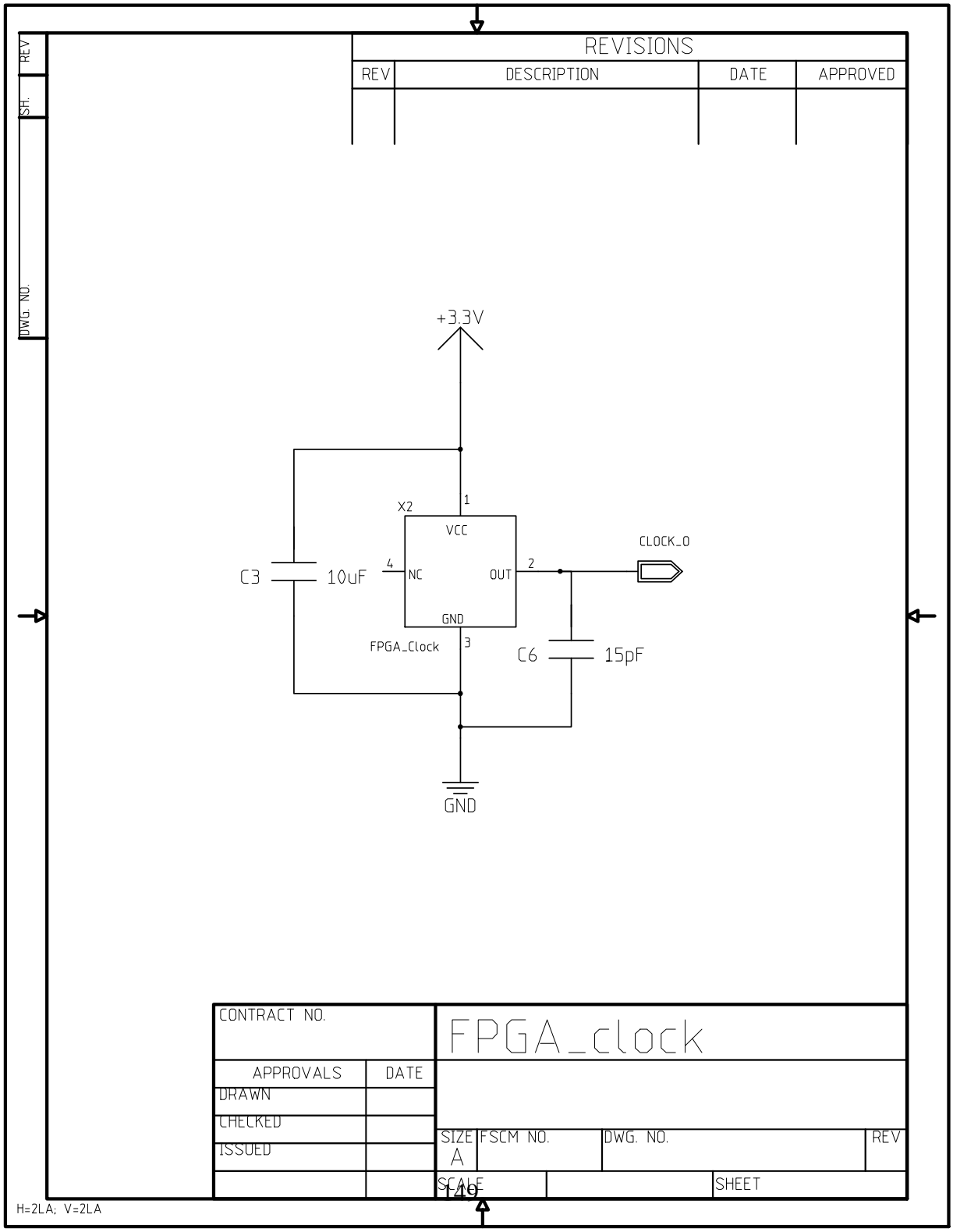
Figur 89: Displaykort - Powerkoblinger



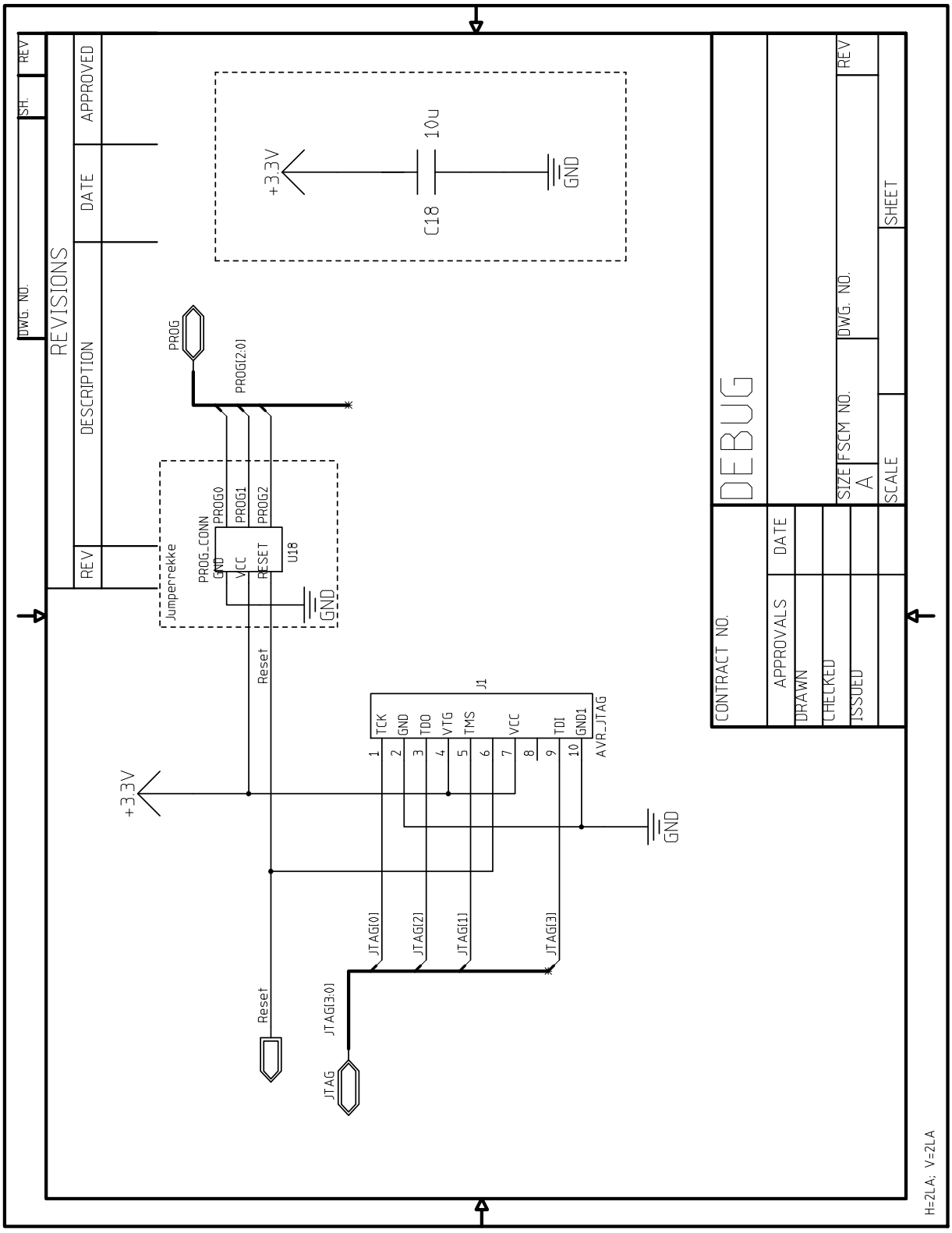
Figur 90: Displaykort - Logikkanalysatorporter



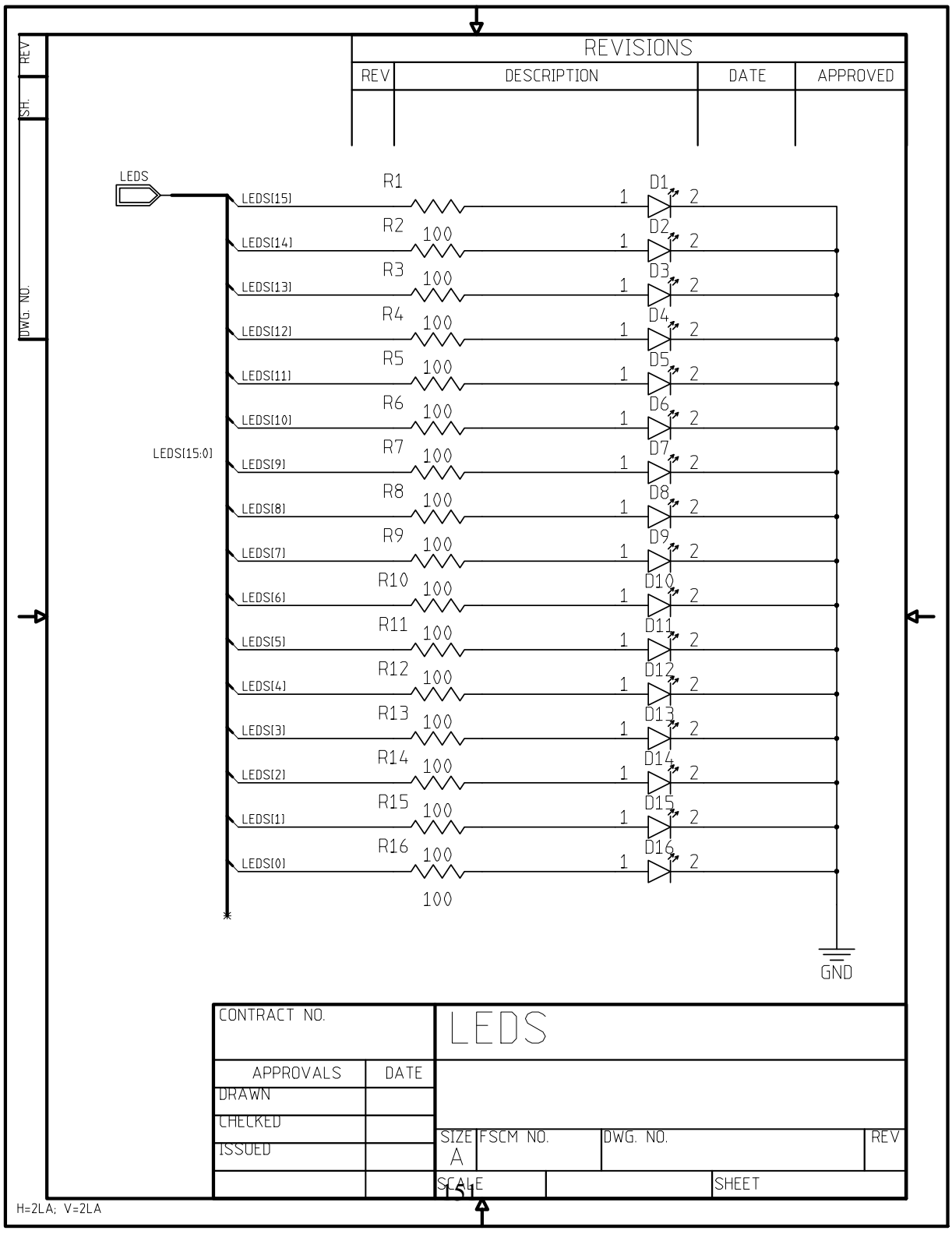
Figur 91: Displaykort - FPGA



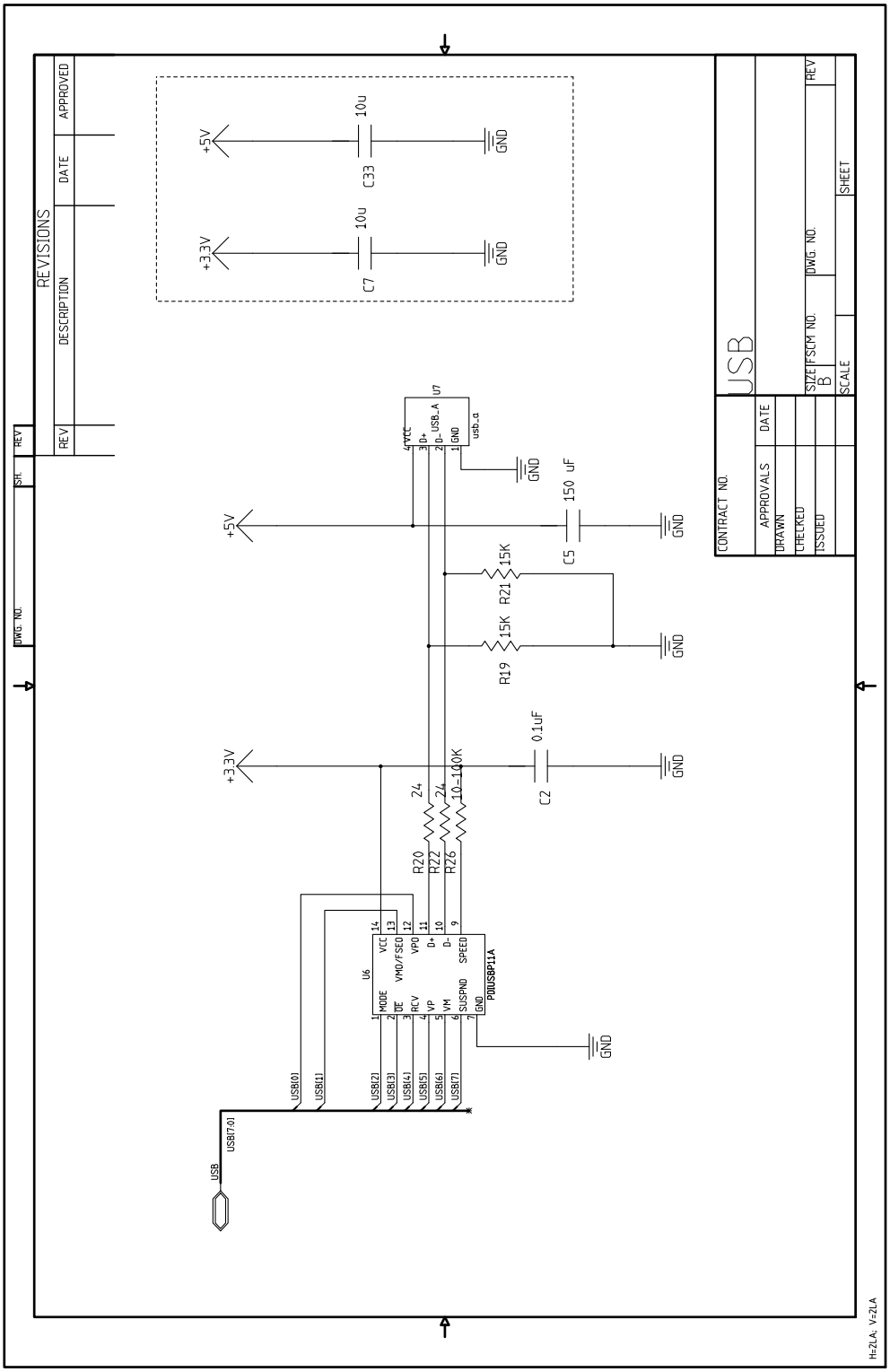
Figur 92: Displaykort - FPGA klokke



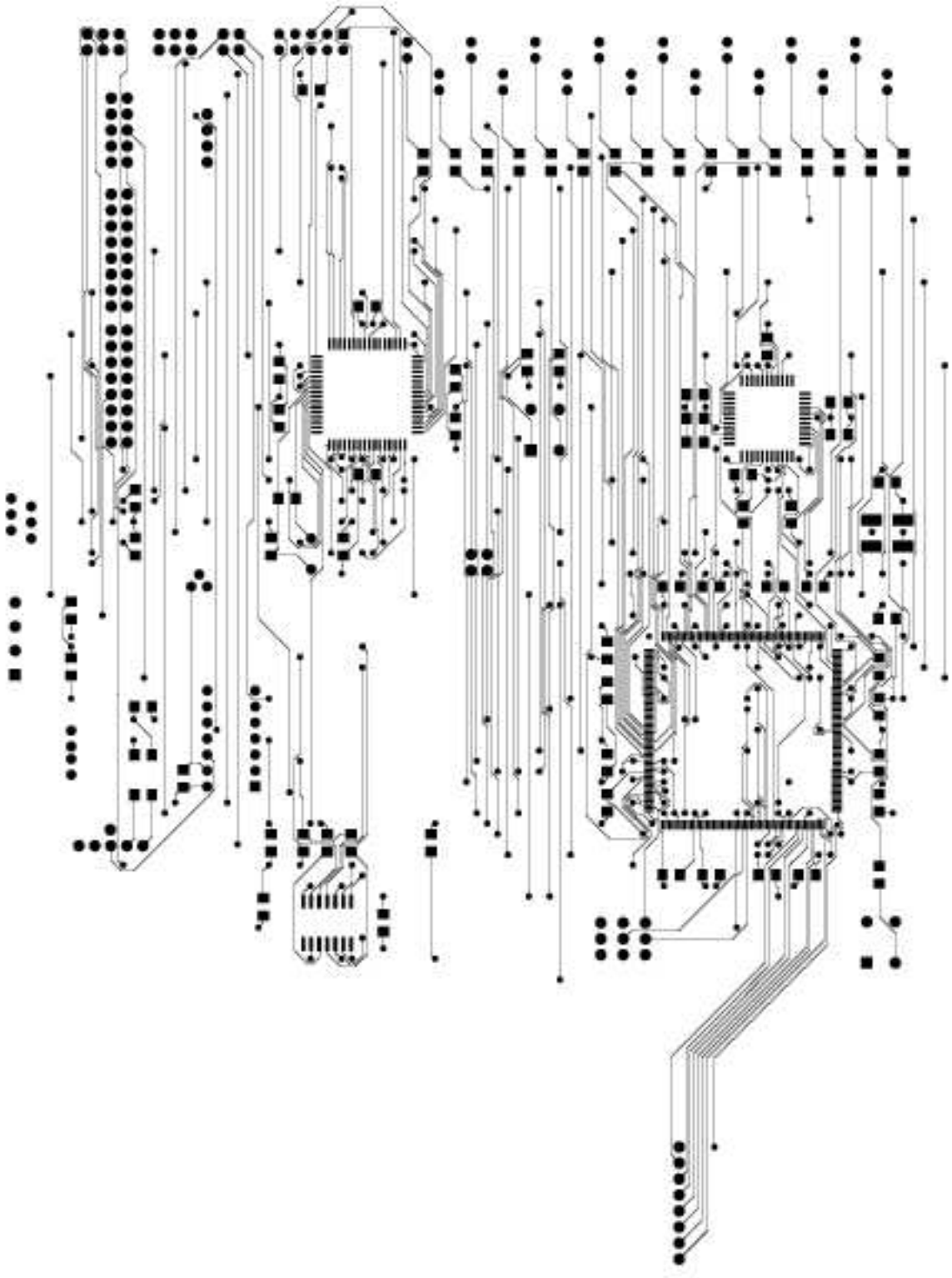
Figur 93: Displaykort - Debugporter



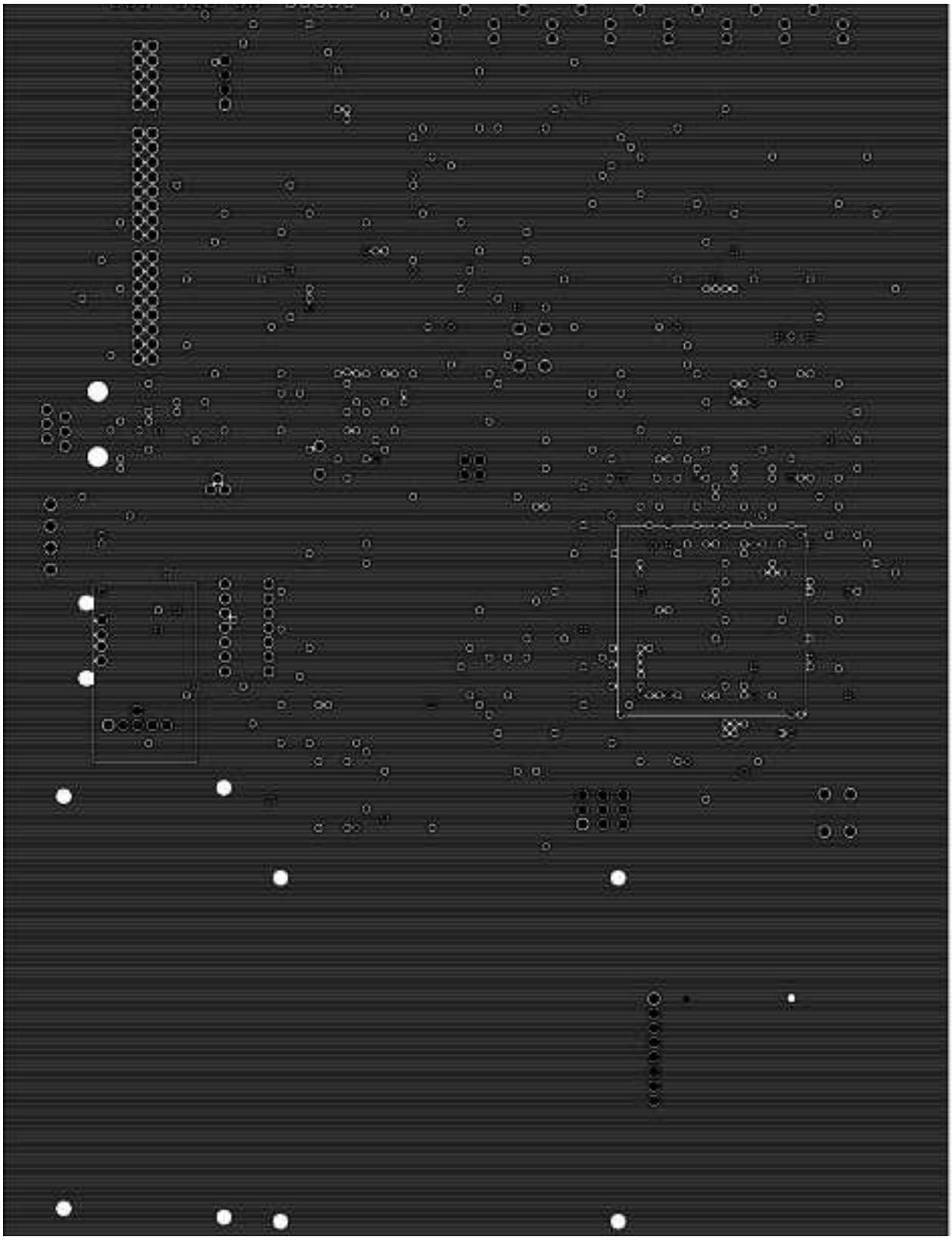
Figur 94: Displaykort - Lysdioder



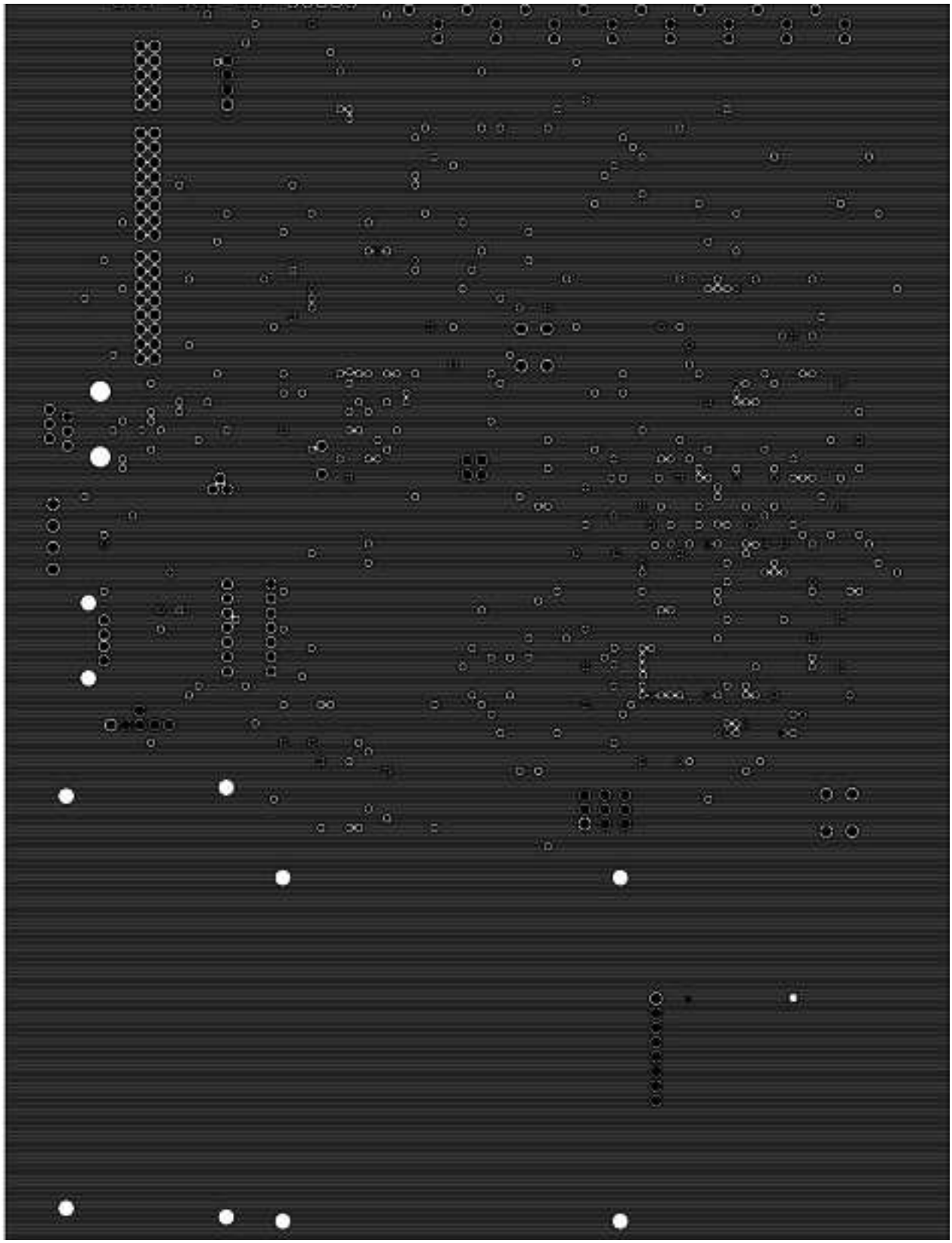
Figur 95: Displaykort - USB transciever



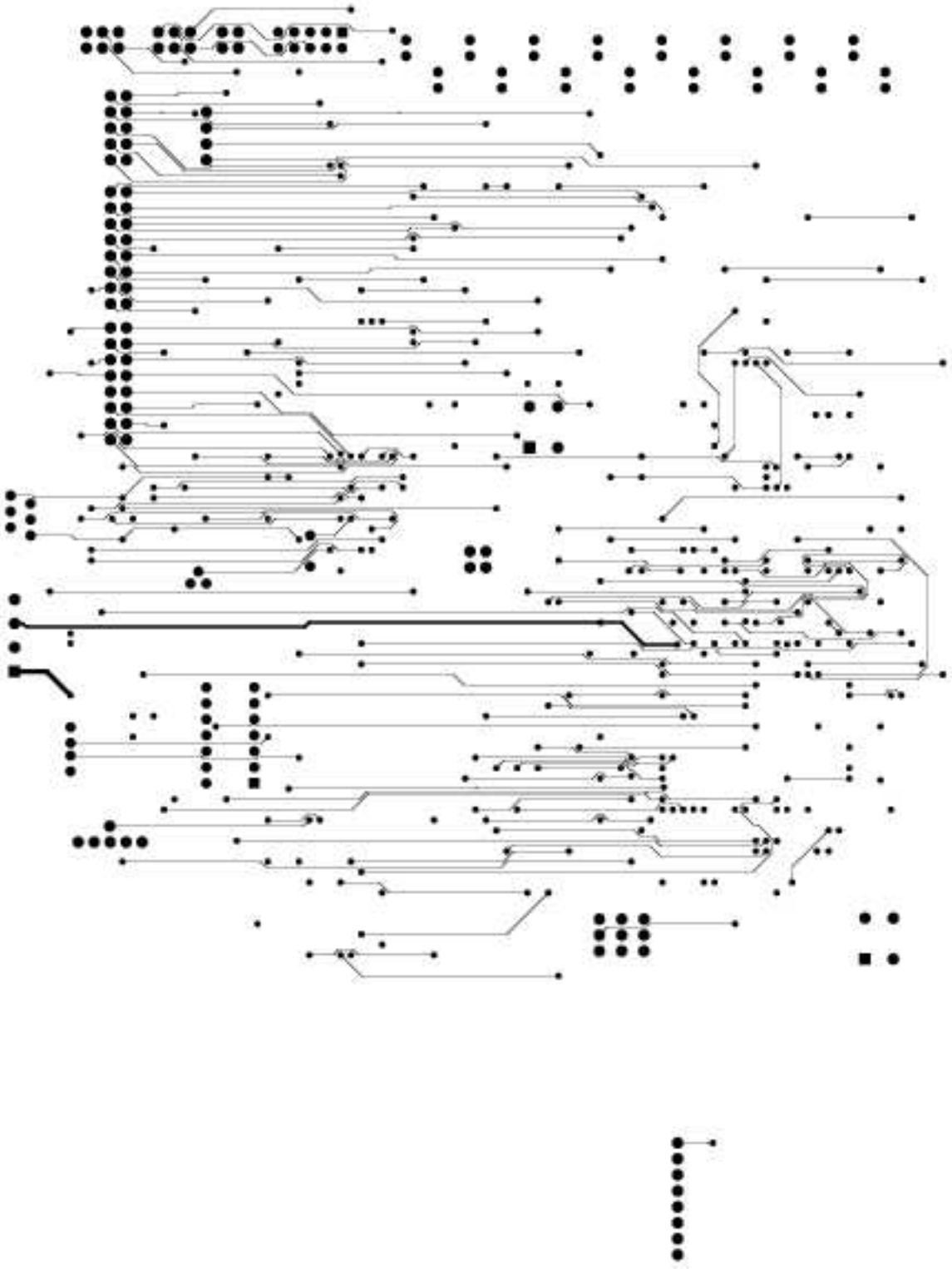
Figur 96: Displaykort - PCB lag 1



Figur 97: Displaykort - PCB lag 2



Figur 98: Displaykort - PCB lag 3

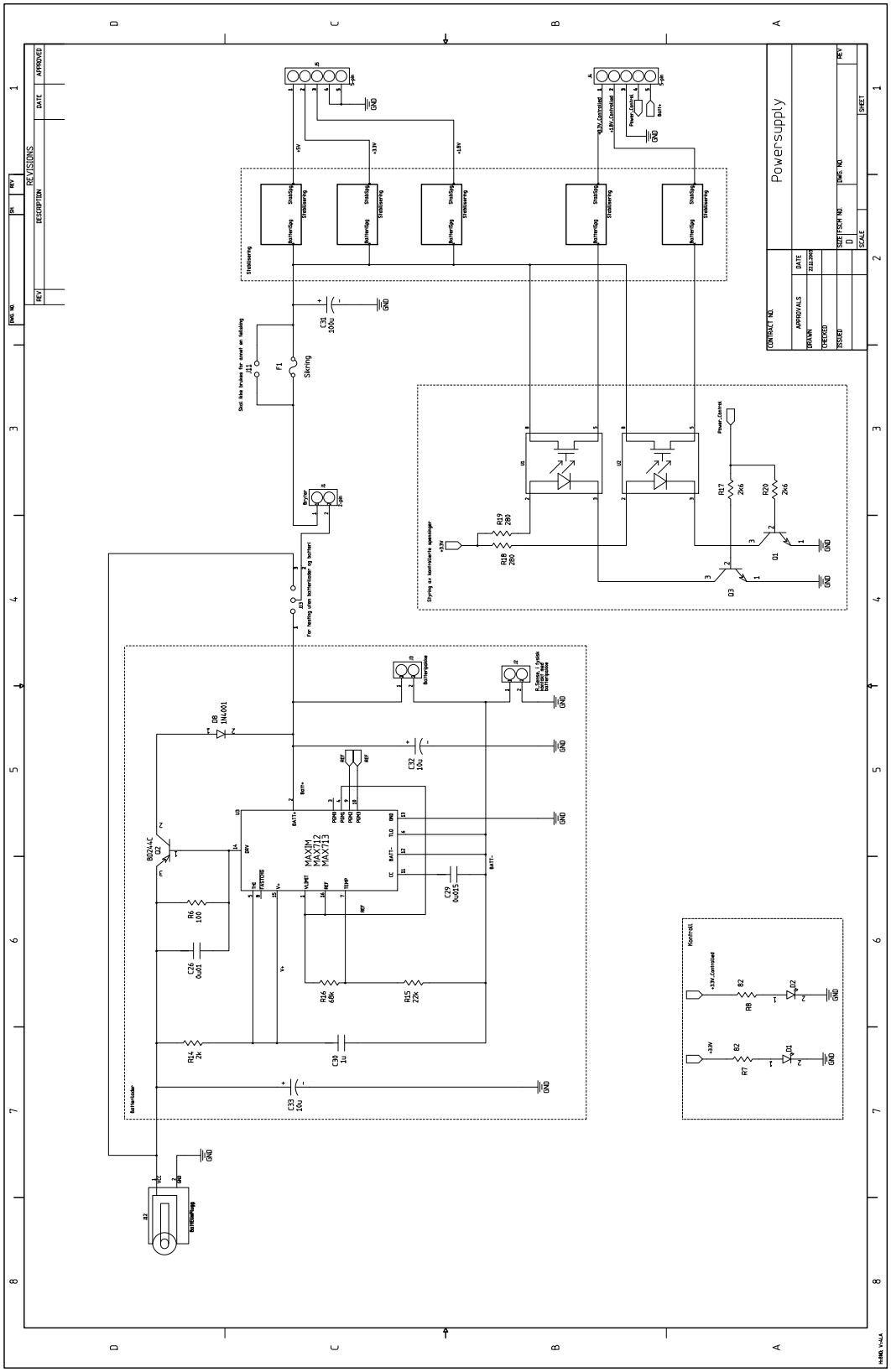


Figur 99: Displaykort - PCB lag 4

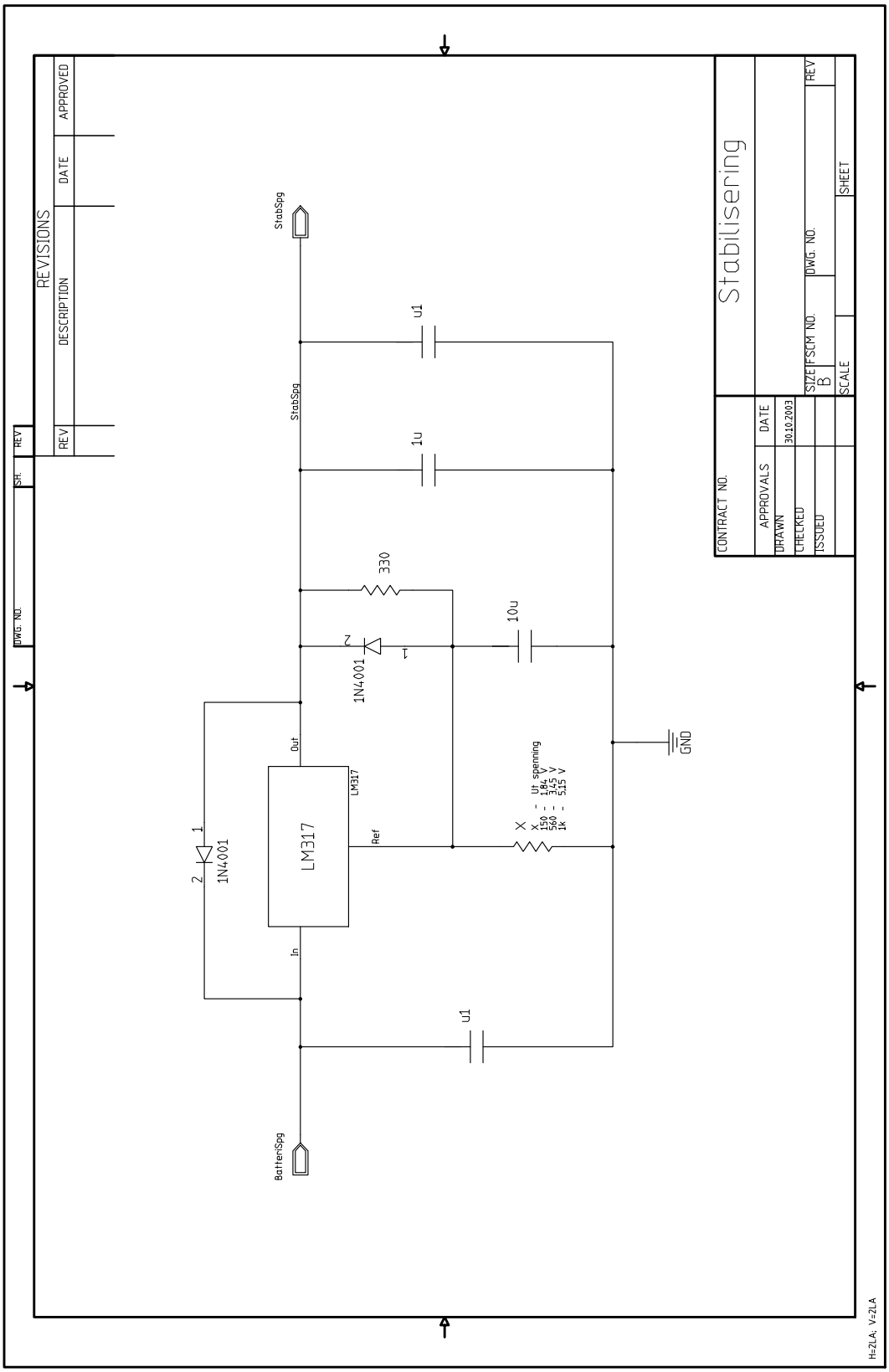
Item	Qty	Part number	Description	References
1	1	LM258N	Opamp 5-pin	U1
2	1	RJ11		J1
3	8	Res_thru		R1-R8
4	1	Sensor		U3
5	2	Var-res		P1-P2

Tabell 23: Lyssensor - Materialforbruk

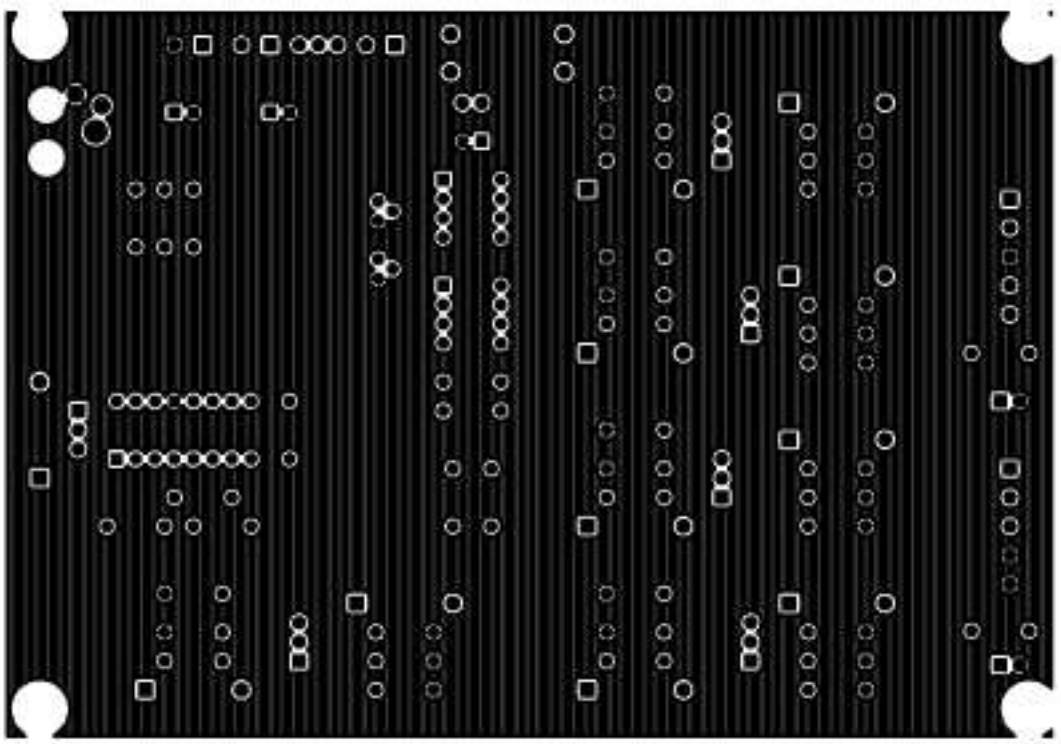
E.3 Powerkort og lyssensor



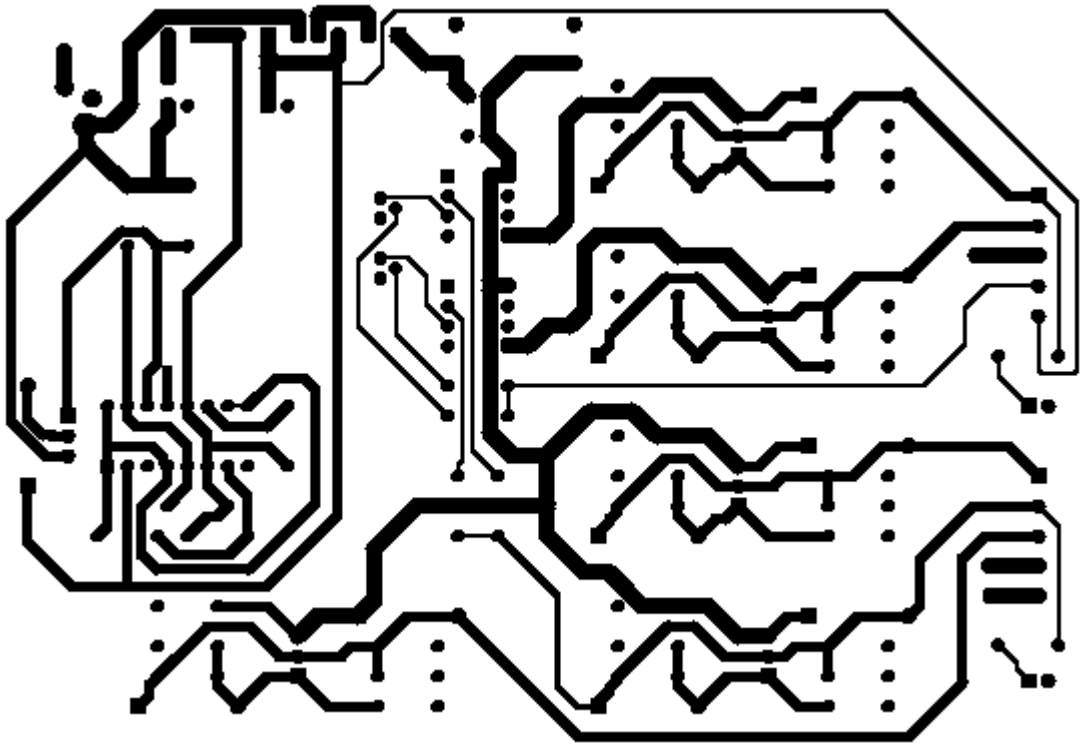
Figur 100: Powerkort - Toppnivå



Figur 101: Powerkort - Spenningsregulatorer



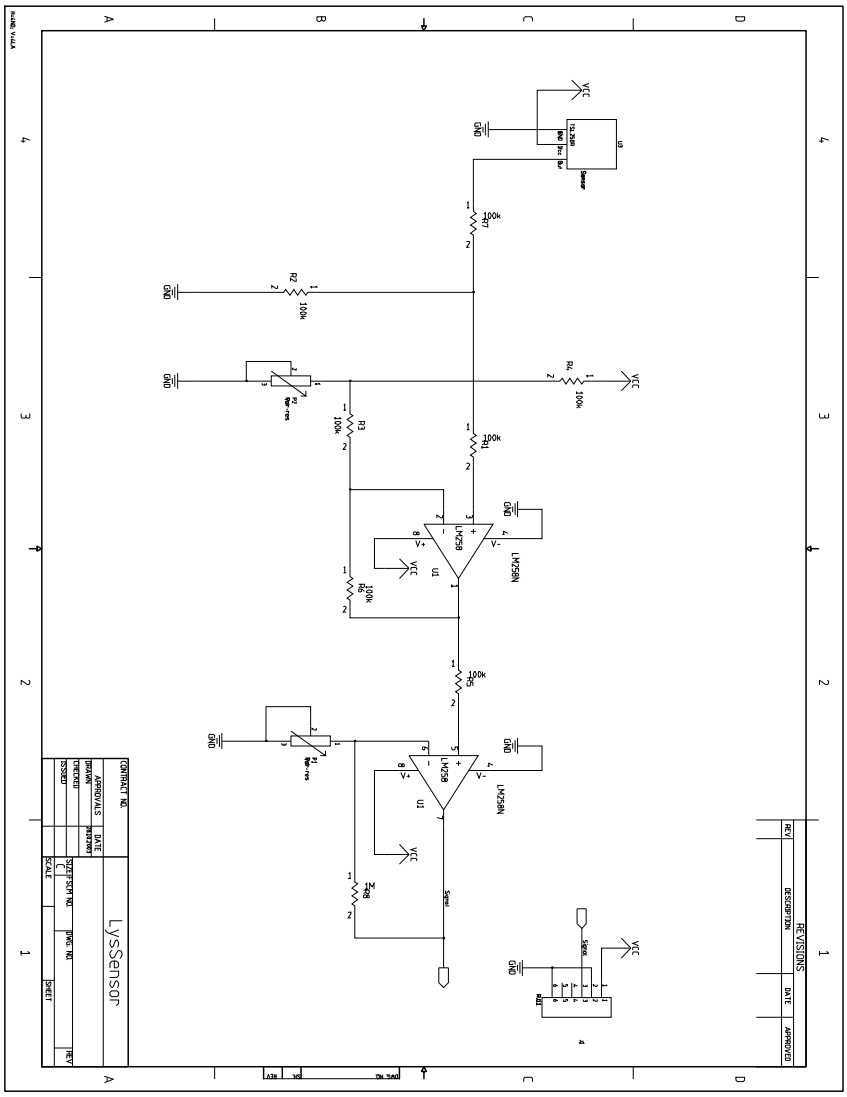
Figur 102: Powerkort - PCB lag 1



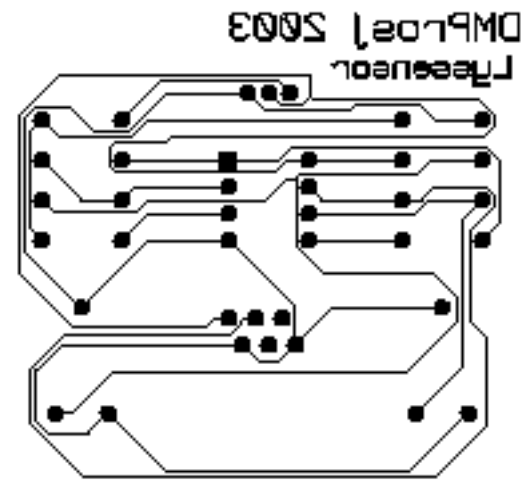
Figur 103: Powerkort - PCB lag 2

Item	Qty	Part number	Description	References
1	2	BC547B	Transistor, NPN BJT	Q1,Q3
2	1	BD244C	Transistor, PNP BJT	Q2
3	1	BattElimPlugg		J12
4	3	CCR75CG101FS	Capacitor	C26,C29-C30
5	20	CCR75CG151FS	Capacitor	C6-C25
6	3	Cap_lyt_liten	Electrolytic Capacitor	C31-C33
7	2	Halvlederele		U1-U2
8	1	Jumper	Jumper	J11
9	2	LED	LED, Photoemissive Diode	D1-D2
10	5	LM317		U4-U8
11	1	MAX713		U3
12	16	Res_thru	Resistor	R1-R16
13	4	Res_thru		R17-R20
14	1	Sikring	Fuse	F1
15	11	1N4001	Diode	D3-D13
16	3	2-pin		J1-J3
17	1	3-pin-korts		J13
18	2	5-pin		J4-J5

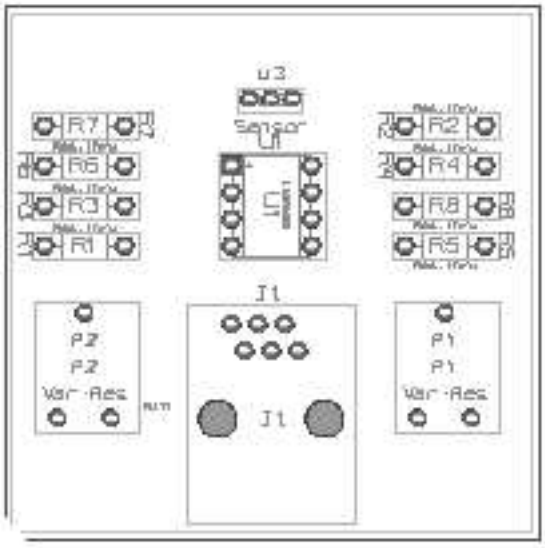
Figur 104: Powerkort - Materialer



Figur 105: Lyssensor - Skjemategning

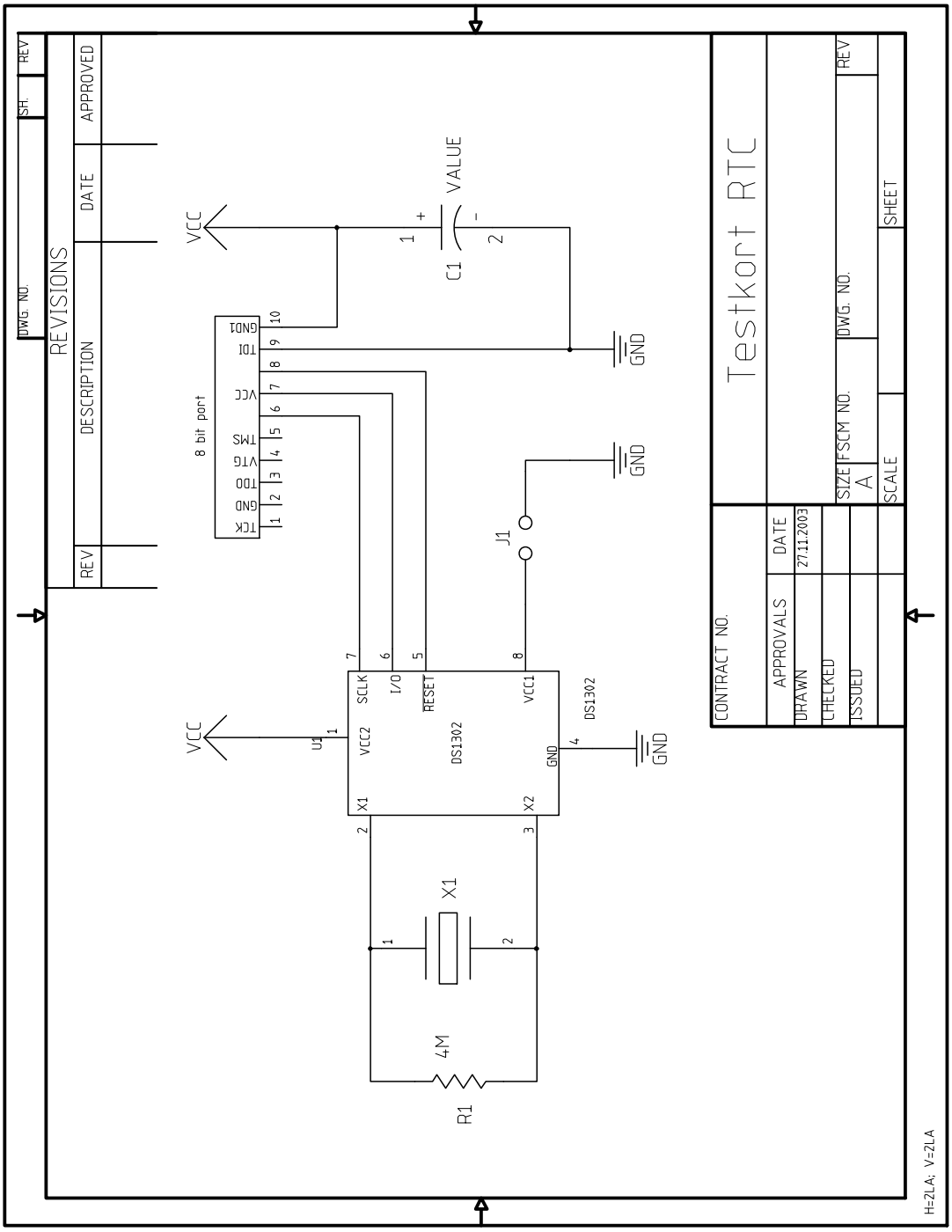


Figur 106: Lyssensor - PCB

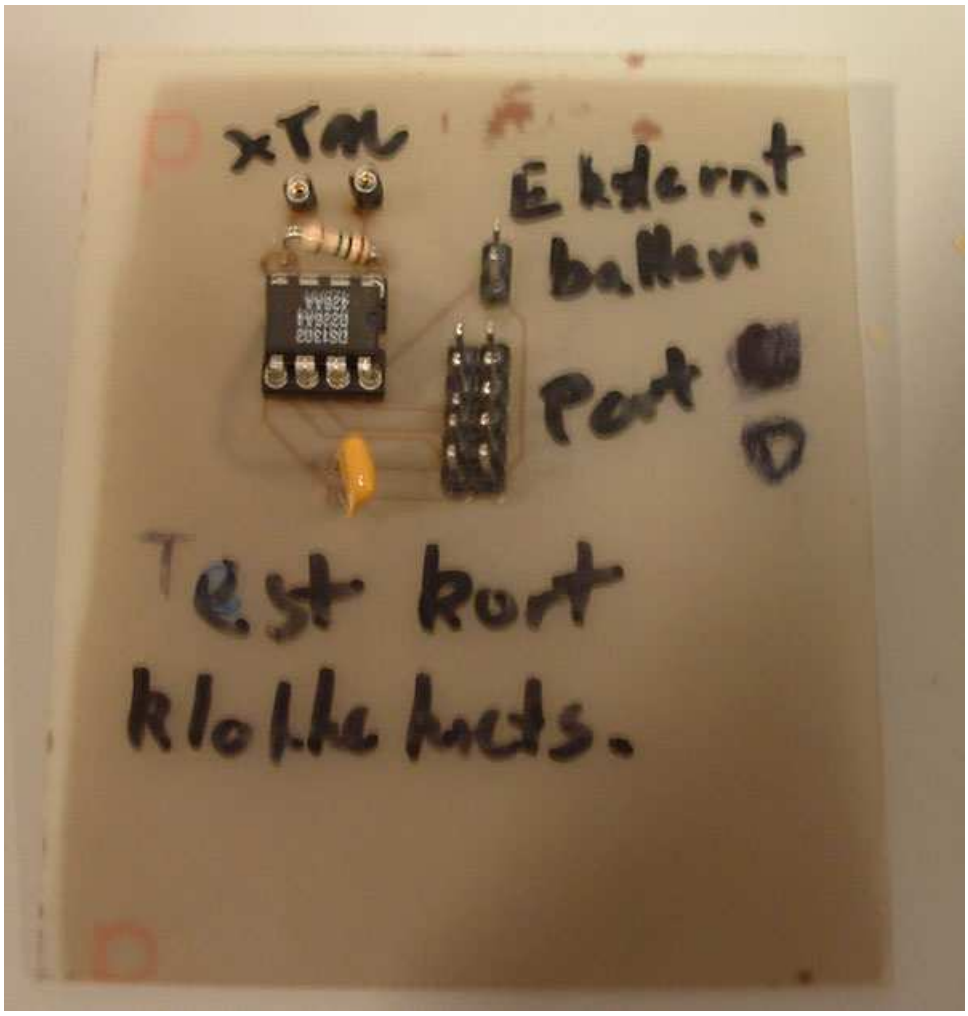


Figur 107: Lyssensor - Silkeprint

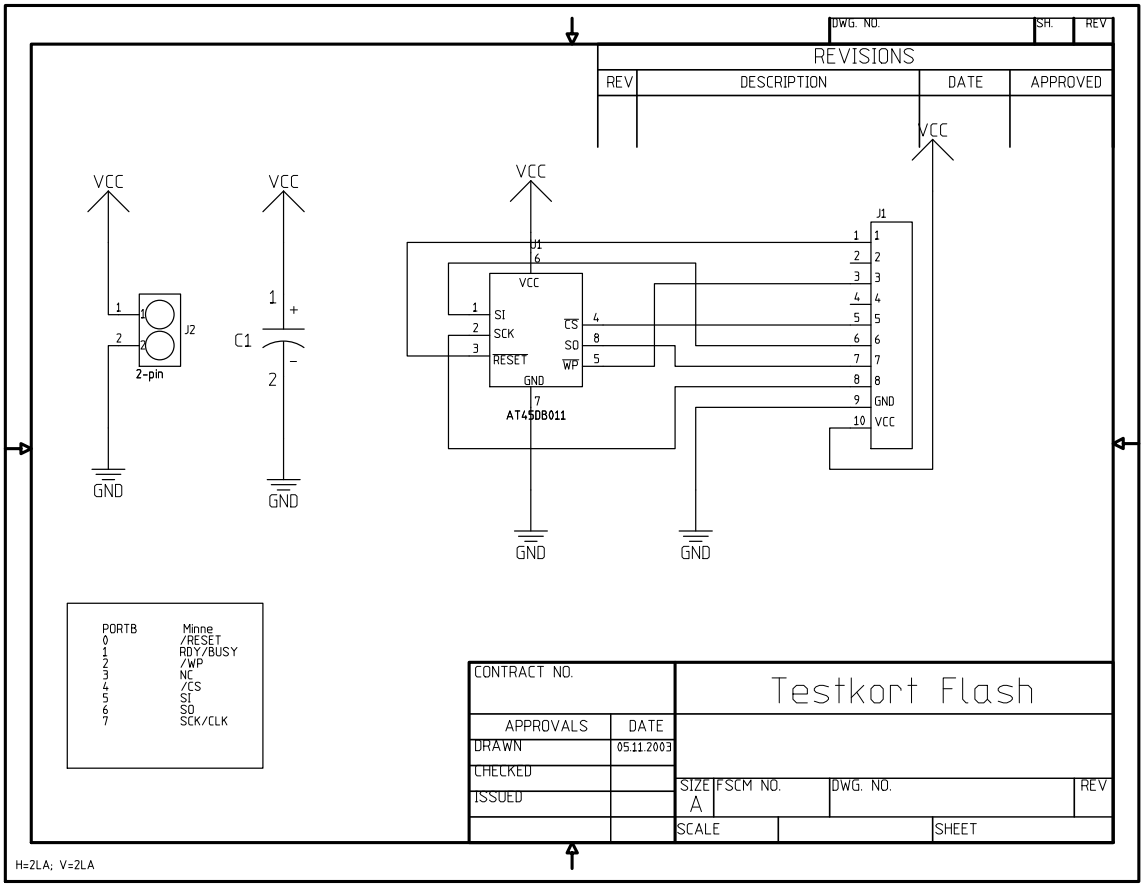
E.4 Diverse testkort



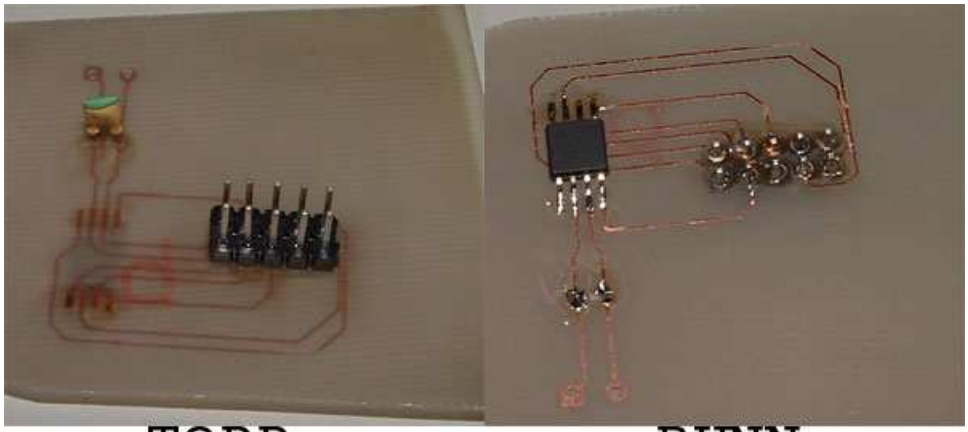
Figur 108: Skjema for sanntidsklokketestkortet



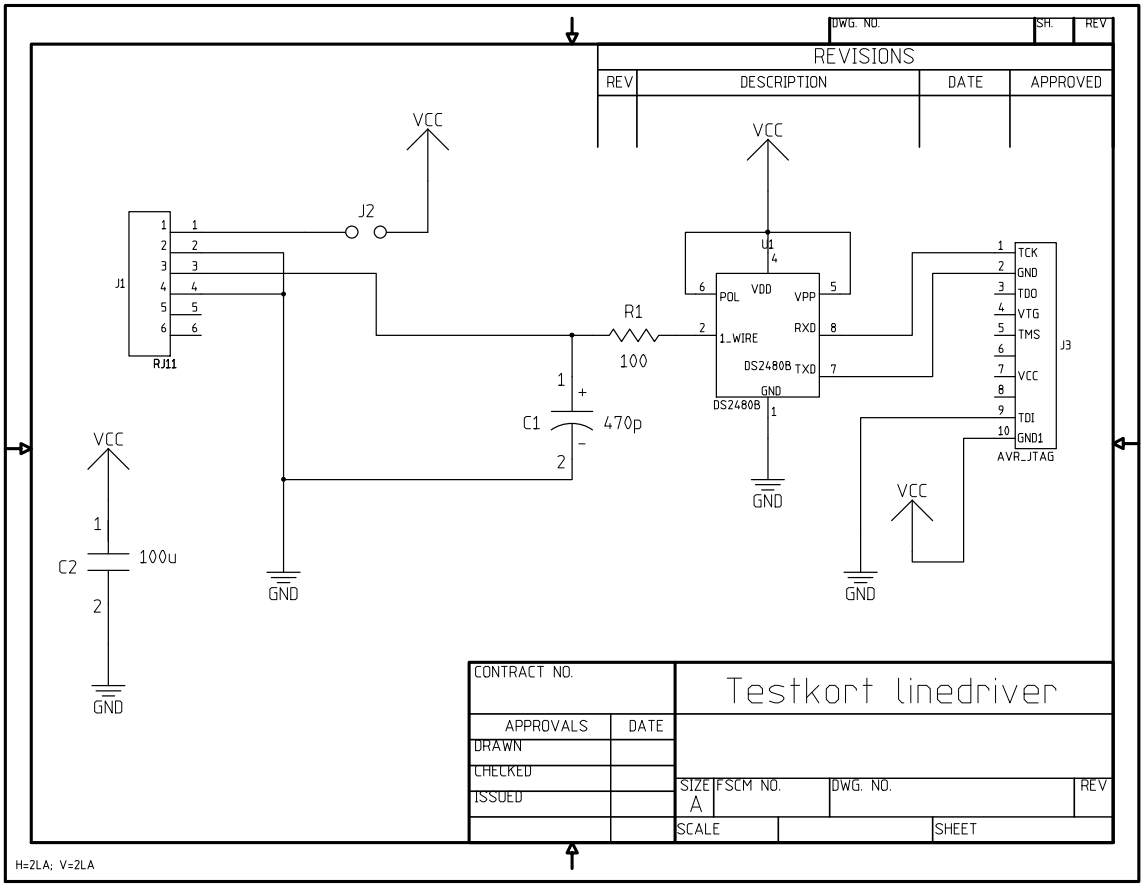
Figur 109: Slik ser testkortet for sanntidsklokken ut



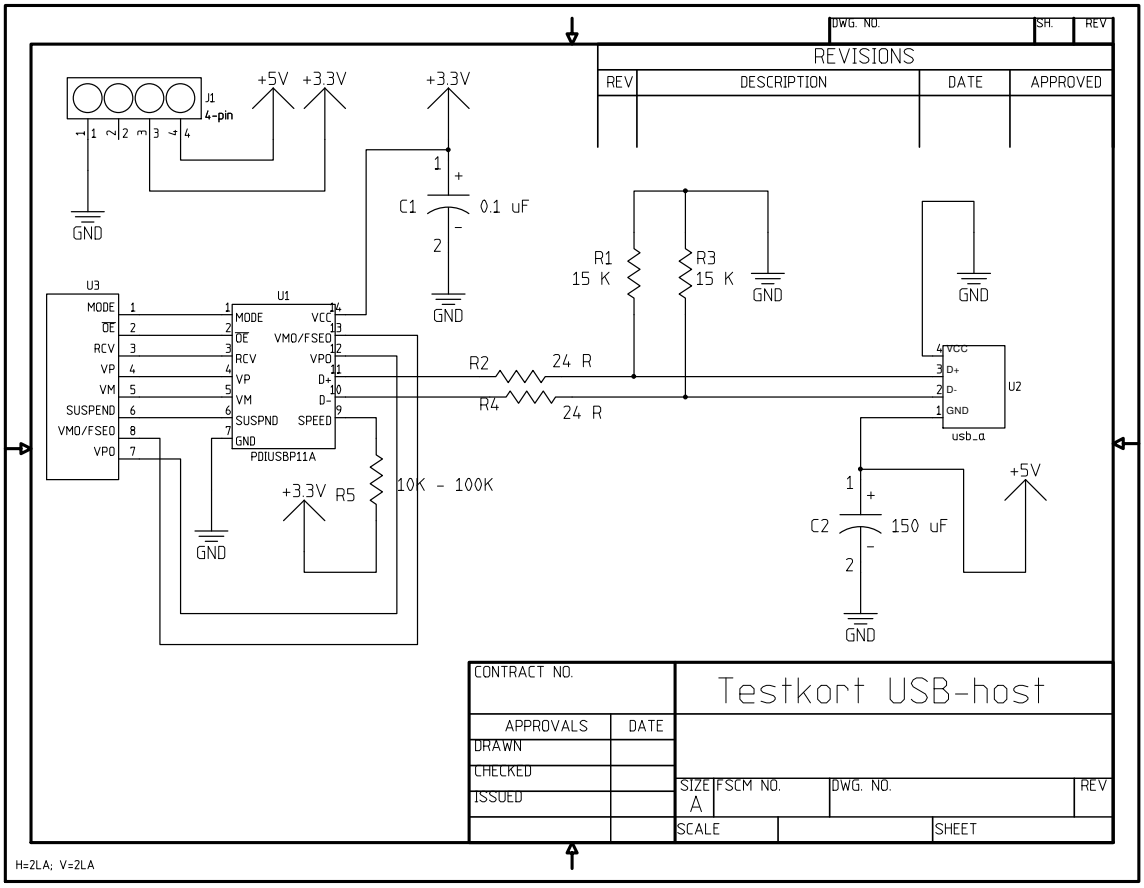
Figur 110: Skjema for minnetestkortet



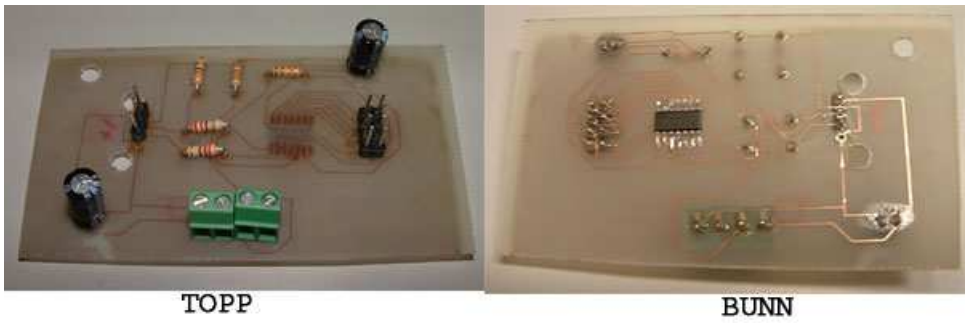
Figur 111: Slik ser testkortet for minnebrikken ut



Figur 112: Skjema for linedrivertestkortet



Figur 113: Skjema for USBtranceivertestkortet



Figur 114: Slik ser testkortet for USBtranceiver ut

F Kildekode

Listings

F.1 Sensorkort - Mikrokontroller

Listing 1: Sensorkort/Makefile

```
# TDT4255 - Konstruksjon av Datamaskiner
# Very simple, heavily commented makefile for AVR compilation


#--------------------Definition of variables to be used-----------------------------------------------------------

# Use avr-gcc for compilation and linking
CC      = /usr/local/avr/bin/avr-gcc
LD      = /usr/local/avr/bin/avr-gcc

# Use avr-objcopy for copying object file
OBJCOPY = /usr/local/avr/bin/avr-objcopy

# Use avr-size to determine size of code
SIZE    = /usr/local/avr/bin/avr-size

# Use avr-upload for uploading to microcontroller
UPLOAD  = avrdude

# Use avr-objdump to mix C source and assembler code
DUMP    = avr-objdump

# AVR microcontroller model
PART    = atmega32

# Programmer to use
PROG    = stk500

# Flags for compilation
# -Wall: Display all warnings
# -Os  : Optimize for size
# -mmcu: Type of CPU
CFLAGS = -Wall -mmcu=$(PART) -g -Os

# Flags for linking (same flags as for compilation)
```

```
LDFLAGS = -Wall -mmcu=$(PART) -g

# List of object files
OBJECTS = ow.o sensorer.o teller.o usart.o main.o time.o analogsensor.o memory.o memutils.o twi.o


#---------------------Rules for compilation-----------------------------------------------------------------------------

# The different processes required to generate the final machine code are defined by rules
# Currently, this file specifies three rules to generate a hex-file with the final machine code
#
# First, the source file is compiled into the object file myprogram.o
# Secondly, the object files (only one here) are linked, producing myprogram.out
# Finally, the output must be converted into a format (ihex) readable by the uploader program (avrdude)
#
# The left side of a rule is the file to be generated
# The right side defines what the generation of the left side file depends on
# Make starts checking the first rule by default, checking if the right side elements exist and are up to date
# Thus, the three processes are listed in reverse order, and make will track its way down to the first process (COMPILE)


#---CONVERT---
# Create the hex-file by making a copy of and converting myprogram.out
# Necessary to convert from ELF (output from GCC linker) to ihex (raw format for AVR)
#-j .text : Copy only the .text-section of myprogram.out
#-O ihex  : Convert format to Intel HEX
# $< : Special variable, refers to the first right side file (whack_a_rat.out)
# $@ : Special variable, refers to left side of rule (flash.hex)
main.hex : main.out
        $(OBJCOPY) -j .text -O ihex $< $@

#---LINK---
# Link object-files
# Makefile is specified on the right side so that linking will occur if the Makefile has a timestamp newer than myprogram.out
#-o : output file
# Also, report the size of the output file
main.out : $(OBJECTS) Makefile
        $(LD) $(LDFLAGS) -o $@ $(OBJECTS)
        $(SIZE) $@

#---COMPILE---
# Compile the program into an object file
#-c : only compile (do not link)
main.o : main.c usart.h sensorer.h ow.h teller.h time.h analogsensor.h memory.h twi.h Makefile
        $(CC) $(CFLAGS) -c $<


#---------------------Extra functions-----------------------------------------------------------------------------------

# Make will by default start with the first compilation rule in the makefile (generate flash.hex)
# You can, however, call make with the first argument being a spesific rule. Make will start with that rule.
# You can use the following three rules as follows:
#
# make upload    - Upload flash.hex to a programmer connected to a com-port (such as the stk500 developers board)
# make clean     - Deletes all non-source files (i.e. files produced by make)
# make dump      - Dumps disassembled code and intermixes it with source C code in the file myprogram.asm

#---UPLOAD---
# Upload uses avrdude to load the hex-file into a connected programmer
# Using .PHONY generates an alias that can be used on the left side of a rule without being a file
#-p     : Type of AVR microcontroller
#-c     : Programmer attached to the com-port
#-e     : Erase former program from the AVR
#-i     : Input file (flash.hex)
#-v     : Verbose mode (rich text feedback)
.PHONY : upload
upload : flash.hex
        $(UPLOAD) -p $(PART) -c $(PROG) -e -i $< -v

#---CLEAN---
# Clean simply deletes the files generated by make
.PHONY : clean
clean :
        rm -f *.hex *.out *.o *.map *.asm
```

Listing 2: Sensorkort/main.c

```
#include "main.h"
// include intrinsic commands like _SLEEP
#include <avr/ina90.h>

// Dette holder for samplingsintervaller åp maks 27 ødgn
volatile uint16_t ant_sleep;
volatile uint16_t beregna_sleep;
// sek, min, time, dato, mndå, r
uint8_t buf[6];
```

```
uint8_t buff[6];
uint8_t twi_rx[30];
uint8_t twi_tx[60];

void set_rtc( uint8_t *buff);

SIGNAL(SIG_2WIRE_SERIAL) {
  uint16_t tsens;
  PORTC |= ~0x1f;
  twi_les(twi_rx);
  TWCR &= ~(1<<TWIE);
  sei();
  switch(twi_rx[1]) {
    case 0x06 :
      // Les klokke
      PORTC |= ~0x2f;
      twi_tx[0] = 6;
      twi_tx[1] = 0x86;
      rtc_getClock(twi_tx+2);
      //twi_tx[7] = gen_crc(twi_tx, 7);
      twi_send(0x01, twi_tx);
      break;
    case 0x08 :
      // Still klokke
      PORTC |= ~0x3f;
      set_rtc(twi_rx+2);
      twi_tx[0] = 0;
      twi_tx[1] = 0x88;
      twi_send(0x01, twi_tx);
      break;
    case 0x05 :
      // Les sensor X
      PORTC |= ~0x4f;
      ow_initialization();
      tsens = avles_sensor(twi_rx[2]);
      twi_tx[0] = 2;
      twi_tx[1] = 0x85;
      twi_tx[2] = (uint8_t)(tsens >> 8);
      twi_tx[3] = (uint8_t)tsens;
      twi_send(0x01, twi_tx);
      break;
    default :
      // tja...
      PORTC |= ~0x5f;

      break;
  }
  cli();
  TWCR |= (1<<TWIE);

}


SIGNAL(SIG_OUTPUT_COMPARE1A) {
  ant_sleep++;
}

void sov( void ) {
  uint8_t i;

  // start 16-bits counter1
  while (1) {
    start_teller();
    // enable sleep
    MCUCR |= (1 << SE);
    _SLEEP();
    // disable sleep
    MCUCR &= ~(1 << SE);
    if (ant_sleep > beregna_sleep) {
      cli();
      stopp_teller();
      //uint8_t pre_min, post_min;
      //rtc_getClock(buf);
      //pre_min = buf[0];
      //ow_initialization();
      /*
      // LEGGER INN EN POST I FLASH MINNET
      sensor_avlesning avlest = avles_alle_sensorer();
      for (i = 1; i < ANT_SENSORER +1; i++) {
        // time-stamp
        for (i = 4; i < 8; i++) {
          df_global_post[i] = 0x00;
        }
        // ant-sensorer
        df_global_post[8] = ANT_SENSORER;
        // sensorid & sensordata
        for (i = 1; i < ANT_SENSORER + 1; i++) {
          // legger inn sensorid for alle sensorene
```

```
            df_global_post[6 + (3 * i)] = i;
            // legger inn sensordata som 2 byte
            df_global_post[avlest.sensordata[i] >> 8];
            df_global_post[avlest.sensordata[i]];
          }
        }*/
        /*
        // time-stamp
        for (i = 4; i < 8; i++){
            df_global_post[i] = 0x00;
        }
        // ant-sensorer
        df_global_post[8] = 0x04;
        // sensorid & sensordata
        df_global_post[9] = 0x03;
        */
        //uint16_t temples = avles_sensor(0x03);
        //df_global_post[10] = (uint8_t)(temples >> 8);
        //df_global_post[11] = (uint8_t)(temples);
        //PORTC = ~((uint8_t)avles_sensor(0x02)); // vindhast
        //delay_ms(5000);
        //PORTC = ~(uint8_t)temples;
        //PORTC = ~((uint8_t) avles_sensor(0x01)); // temp
        //delay_ms(5000);
        ant_sleep = 0;
        sei();
      }
    }
}

void avr_initialization( void ) {
  //DDRC = 0xFF;
  //PORTC = ~0xCC;
  ant_sleep = 0;
  // idle mode
  MCUCR &= ~((1 << SM2) | (1 << SM1) | (1 << SM0));
  OCR1A = 0x2327;        // ca. 5 s m/prescaler 1024
  // compare match interrupt enabled
  TIMSK |= (1 << OCIE1A);
  // enable global interrupts
  sei();
}

void set_rtc( uint8_t *buff ) {
  RTC_WPDisable();
  rtc_setClock(buff);
  RTC_WPEnable();
}

// teller oss fram til det er ~5sekunder igjen til avlesning
void beregn_sleep( int sekunder ) {
  float temp = (float)sekunder;
  beregna_sleep = (temp * 0.2) - 1;
}


void main( void ) {
  DDRC |= 0xfc;
  beregn_sleep(60);
  uint8_t i;
  for (i = 0; i < 6; i++) {
    buf[i] = 0;
  }
  buff[0] = 0;
  buff[1] = 0;
  buff[2] = 0;
  buff[3] = 21;
  buff[4] = 11;
  buff[5] = 3;
  avr_initialization();
  twi_init(2);
  twi_enableinterrupt();
  rtc_init();
  //ow_initialization();
  //set_rtc();
  sov();
}
```

Listing 3: Sensorkort/main.h

```
#ifndef main_program
#define main_program

#include "ow.h"
#include "time.h"
#include "memory.h"
```

```
#include "twi.h"

#endif
```

Listing 4: Sensorkort/ow.c

```
/* ow.c
 *
 * Dette programmet kommuniserer med uart/1-wire broen DS2480B. Det
 * er vha denne broen de digitale sensorene leses av ATmega16.
 *
 * Det er øforelpig ikke lagt inn delay, siden DS2480B simuleres.
 * Det vil antagelig bli øndvendig med delay etter hver øforesprsel til
 * DS2480B.
 *
 * Hvor skal mapping av SerialNmbr/ID og sensornavn åforeg, hvor skal
 * det lagres?
 *
 */

#include "ow.h"

/*
typedef struct{
  uint16_t sensordata[5];
} sensor_avlesning;
*/

uint8_t respons;          // *DEBUGGING*


// globale metoder
uint16_t avles_sensor(uint8_t sensorid);
uint8_t sjekk_sensortype(uint8_t sensorid);
sensor_avlesning avles_alle_sensorer( void );
void delay_ms(uint16_t msek);

// lokale metoder
int reset_ow( void );
int DS2480Detect(void);
void software_masterReset(void);

uint8_t rx_buffer[30];
uint8_t tx_buffer[30];


/* avles_sensor
 *
 * Innparametere: sensorid, operasjon
 *
 * sensorid = id'en som benyttes i minnet og åp displaykortet
 * operasjon =  avles sensoren med aktuell sensorid
 *
 * Returnerer: sensorverdien uint16_t :
 */
uint16_t avles_sensor(uint8_t sensorid) {

  switch (sjekk_sensortype(sensorid)) {
    case 0x00:
      return avles_ingen();
      break;
    case 0x01:
      return avles_generell();
      break;
    case 0x02:
      return avles_temperatur(sensorid);
      break;
    case 0x03:
      return avles_fuktighet(sensorid);
      break;
    case 0x04:
      return avles_trykk(sensorid);
      break;
    case 0x05:
      return avles_vindhastighet(sensorid);
      break;
    case 0x06:
      return avles_vindretning();
      break;
    case 0x07:
      return avles_nedbor();
      break;
    case 0x08:
      return avles_lys();
      break;
    default:
```

```
        return 0;
        break;
    }
  }

uint8_t sjekk_sensortype(uint8_t sensorid) {

  switch (sensorid) {
    case 0x01:
      return sensortab[1].type;
      break;
    case 0x02:
      return sensortab[2].type;
      break;
    case 0x03:
      return sensortab[3].type;
      break;
    case 0x04:
      return sensortab[4].type;
      break;
    case 0x05:
      return sensortab[5].type;
      break;
    case 0x06:
      return sensortab[6].type;
      break;
    case 0x07:
      return sensortab[7].type;
      break;
  }
  return(0);
}

sensor_avlesning avles_alle_sensorer( void ) {
  uint8_t i;
  sensor_avlesning temp;
  for (i = 1; i < ANT_SENSORER + 1; i++) {
    temp.sensordata[i] = avles_sensor(i);
  }
  return temp;
}

/* STK500 *DEBUGGING* */
void debug_initialization( void ) {
  //DDRC  = 0xFF;      // Set //PORTC as output

}


/* Har ikke klartå finne ut om USART åp AVR'en har en åmteå
   * sende ut BREAK åp. åS istedet simulerer jeg en break
   * vedå justere character østrrelsen fra 8-bit til 9-bit
   * og sender en char med 0 i MSB-->f.eks 0x00
   */
void software_masterReset(void) {

  UCSRB = 0x9C;          // 1001 1100 UCSZ2 enabled-->9-bit
                         // Receiver enabled, Transmitter enabled, 9-bit enabled
  usart_putc(0x00);      // Sender NULL tegnet ('\0')

  // delay ??

  UCSRB = 0x98;          // 1001 1000
                         // Receiver enabled, Transmitter enabled, back to 8-bit

  /* OBS! VIKTIGÅ MERKE SEG VED MASTER_RESET
   *
   * øflgende 2 kodelinjer eller tilsvarende åm ævre med for at DS2480B
   * skal gi forventa reset respons
   *
   * 1 byte med gjenkjennbar kode
   * 1 getc() øforesprsel forå æ fjrne øfrste respons (ukjent/udokumentert)
   * MERKELIG!
   */
  usart_putc(0xC1);
  usart_getc();

}


/* reset kommando til DS2480B forå nullstille 1-wire bussen.
 * Forutsetter at DS2480B allerede er i COMMAND_MODE
 *
 * Returns: TRUE/FALSE
```

```
 */
int reset_ow( void ) {
  uchar respons;
  // Sender RESET til DS2480B
  usart_putc(0xC1);      // 110X0001 reg.speed
  // Venter åp svar fra DS2480B
  respons = usart_getc();
  // Respons : DS2480B er klar
  if (respons == 0xCD || 0xED)
    return TRUE;
  else
    return FALSE;
}


int DS2480Detect( void ) {
  uchar respons;
  int DS2480B_detected = FALSE;

  // OBS! DS2480B åm ha en gjenkjennbar byte rett etter pwr-on
  //usart_putc(0xC1);

  // Sender RESET til DS2480B
  usart_putc(0xC1);      // 110X0001 reg.speed

  ////PORTC = ~0x88;
  // Venter åp svar fra DS2480B
  respons = usart_getc();

  ////PORTC = ~respons; // *DEBUGGING*

  //delay_ms(4000);

  switch (respons) {

  // Respons : DS2480B er klar
  case 0xCD: case 0xED:          // 2 muligheter pga don't care bit5
    DS2480B_detected = TRUE; // kan åogs forenkle v/ return true fra metoden
    ////PORTC = ~0xAA;
    break;

  // Respons : 1-Wire kortslutning
  case 0xCC: case 0xEC:
    //Error "DS2480B funnet, men 1-wire linja er kortslutta"
    ////PORTC = ~0x01; // STK500 *DEBUGGING*
    break;

  // Respons : Alarming
  case 0xCE: case 0xEE:
    //Error "DS2480B funnet, men 1-wire enhet(er) er i tilstand ALARMING"
    //PORTC = ~0x02; // STK500 *DEBUGGING*
    break;

  // Respons : Ingen 1-wire enheter tilkobla DS2480B
  case 0xCF: case 0xEF:
    //Error "DS2480B funnet, men ingen enheter åp 1-wire"
    //PORTC = ~0x03; // STK500 *DEBUGGING*
    break;

  // Respons : ingen
  default:
    //Error "DS2480B ikke funnet!"
    //PORTC = ~0x00; // STK500 *DEBUGGING*
    break;
  }

  if (DS2480B_detected == TRUE)
    return TRUE;
  else
    return FALSE;
}

/* 1-wire initialization rutine som økjres ved power-on
 */
void ow_initialization( void ) {

  //debug_initialization();

  // UBRRL = 11 for 1.843MHz clock, U2X = 0, baud = 9600
  usart_init(11);

  // OBS! DS2480B åm ha en byte med gjenkjennbar kode rett etter power-on
  usart_putc(0xC1);

  if (DS2480Detect()) {

    sensortab[1].type = 0x03; // fuktsensor (DS2438)
```

```
        sensortab[1].romkode = 0xB00000001536E326;

        sensortab[2].type = 0x05; // vindhastighet DS2423
        sensortab[2].romkode = 0x5A000000018A7B1D;

        sensortab[3].type = 0x02; // temperatur (fuktsensor DS2438)
        sensortab[3].romkode = 0xB00000001536E326;

        sensortab[4].type = 0x04; // trykk (DS2406 read)
        sensortab[4].romkode = 0x110000002466ED12;

        sensortab[7].type = 0x08; // analog lyssensor


        //sensortab[5].type = 0x02; // trykk (DS2406 write)
        //sensortab[5].romkode = 0xB30000024655812;

        //sensortab[6].type = 0x02; // temperatur (DS18S20)
        //sensortab[6].romkode = 0x9600080014269B10;

        ////PORTC = ~(uint8_t)avles_sensor(0x01);
        //delay_ms(10000);

        ////PORTC = ~0xA0;
        //delay_ms(4000);

    }

    /* ****** VIKTIG *******
     * power-on RESET av DS2480B og
     * software_masterReset() øoppfrer seg IKKE likt!
     * software_masterReset trenger i tillegg til gjennkjennbarkode byte
     * åogs at en usart_getc() øutfres!
     *
     * Se eks. kode under
     *

    usart_putc(0xC1);
    respons = usart_getc();
    //PORTC = ~respons;

    software_masterReset();

    usart_putc(0xC1);
    respons = usart_getc();
    //PORTC = ~respons;

    */

}
```

Listing 5: Sensorkort/ow.h

```
#ifndef ow
#define ow

#include <avr/io.h>
#include "usart.h"
#include "sensorer.h"
#include "teller.h"


#define ANT_SENSORER 4  // *FIX! sjekk heller østrrelsen åp sensortab
#define ANT_SENSOR_ICer 5
#define ANT_SENSORTYPER 9

#define TRUE 1
#define FALSE 0

typedef unsigned char uchar;

/* oppslagstabell for sensorene
 *
 * sensortypene er definert slik:
 *
 * 0x00 = ikke tilstede
 * 0x01 = generell sensor
 * 0x02 = temperatursensor (DS2438/DS2406/DS18S20)
 * 0x03 = fuktsensor (DS2438)
 * 0x04 = trykksensor (DS2406)
 * 0x05 = vindhastighetsensor (DS2423 vindenhet)
 * 0x06 = vindretningsensor (DS2450)
 * 0x07 = ønedbrsensor (DS2423 ønedbrkit)
 * 0x08 = lyssensor (analog)
 *
 * romkoden er definert slik:
```

```
 *
 * 8-bit (MSB) : CRC kode
 * 48-bit      : Serie nummer
 * 8-bit (LSB) : Familie kode (IC typen)
 *
 */
struct sensor {
  uint8_t type;
  uint64_t romkode;
};

struct sensor sensortab[ANT_SENSOR_ICer];


typedef struct{
    uint16_t sensordata[5];
} sensor_avlesning;


/* request_sensor
 *
 * Innparametere: sensorid, operasjon
 *
 * sensorid = id'en som benyttes i minnet og åp displaykortet
 * operasjon =  0x01 å sl_opp_sensortypen
 *              0x02  avlesning
 *
 * Returnerer: ingen
 */
uint16_t avles_sensor(uint8_t sensorid);

uint16_t sjekk_sensor(uint8_t sensorid);

sensor_avlesning avles_alle_sensorer( void );

int DS2480Detect(void);

void delay_ms(uint16_t msek);

/* STK500 *DEBUGGING* */
void debug_initialization( void );


/* reset kommando til DS2480B forå  nullstille 1-wire bussen.
 * Forutsetter at DS2480B allerede er i COMMAND_MODE
 *
 * Returns: TRUE/FALSE
 */
int reset_ow( void );

/* 1-wire initialization rutine som økjres ved power-on
 */
void ow_initialization( void );

#endif
```

Listing 6: Sensorkort/sensorer.c

```
#include "sensorer.h"
#include "analogsensor.h"

// globale variable

/* temperatursensoren aksesseres via DS2406(trykk)
 * eller DS2438(fukt) eller DS18S20 (vind)
 */
uint16_t avles_temperatur( uint8_t sensorid ) {

  int16_t temperatur;

  uint8_t rx[9];
  uint8_t i;

  // snakker med DS2438: avles temp, hent verdi
  if (reset_ow()) {
    usart_putc(0xE1);

    // MatchROM
    usart_putc(0x55);
    for (i = 0; i < 8; i++) {
      usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
      // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
      if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
        usart_putc(0xE3);
    }
    for (i = 0; i < 9; i++) {
```

```
      usart_getc();
    }

    usart_putc(0x44);             // start temperaturavlesning
    usart_getc();

    usart_putc(0xE3);

    if (reset_ow()) {
      usart_putc(0xE1);

      // MatchROM
      usart_putc(0x55);
      for (i = 0; i < 8; i++) {
        usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
        // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
        if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
          usart_putc(0xE3);
      }
      for (i = 0; i < 9; i++) {
        usart_getc();
      }

      usart_putc(0xB8);           // legg resultatene i "scratch pad" minnet
      usart_getc();
      usart_putc(0x00);           // scratch pad : side 0
      usart_getc();

      usart_putc(0xE3);

      if (reset_ow()) {
        usart_putc(0xE1);

        // MatchROM
        usart_putc(0x55);
        for (i = 0; i < 8; i++) {
          usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
          // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
          if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
            usart_putc(0xE3);
        }
        for (i = 0; i < 9; i++) {
          usart_getc();
        }

        usart_putc(0xBE);                 // les scratch pad
        usart_getc();
        usart_putc(0x00);                 // side 0
        usart_getc();
        for (i = 0; i < 9; i++)
          usart_putc(0xFF);
        for (i = 0; i < 9; i++) {
          rx[i] = usart_getc();
        }

        usart_putc(0xE3);

        if (reset_ow()) {
          temperatur = (int16_t) rx[1];
          temperatur |= (int16_t) (rx[2] << 8);
          ///PORTC = ~(int8_t) (temperatur >> 8);       // *DEBUGGING*
          //delay_ms(5000);                             // *DEBUGGING*
        }
      }
    }
  }

  return rx[2];
}


/* vindhastighetssensoren aksesseres via DS2423 (counter)
 * ic'en åp vindsensoren.
 *
 * DS2423 har familiekode 0x1D
 */
uint16_t avles_vindhastighet( uint8_t sensorid ) {

  uint16_t ant_rotasjoner;
  uint16_t vindhastighet;
  uchar i;
  int address;
  int counterPage = 15;

  uint16_t start_telling, slutt_telling, telle_periode;

  uint8_t rx[20];
```

```
    if (reset_ow()) {
    usart_putc(0xE1);

120 // MatchROM
    usart_putc(0x55);
    for (i = 0; i < 8; i++) {
      usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
      // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
125   if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
        usart_putc(0xE3);
    }
    for (i = 0; i < 9; i++) {
      usart_getc();
130 }

    usart_putc(0xA5);       // Read memory + counter
    usart_getc();
    // address of last data byte before counter
135 address = (counterPage << 5) + 31;
    usart_putc((uchar)(address & 0xFF));
    usart_getc();
    usart_putc((uchar)(address >> 8));
    usart_getc();
140
    start_teller();
    delay_ms(2000);         // DETTE ER SAMPLINGS INTERVALLET FOR VINDSENSOREN!!


145 ////PORTC = ~(uint8_t) TIM16_ReadTCNT1();
    //delay_ms(1000);

    // now add the read bytes for data byte,counter,zero bits, crc16
    for (i = 0; i < 11; i++)
150   usart_putc(0xFF);
    for (i = 0; i < 11; i++) {
      rx[i] = usart_getc();
    }
    start_telling = rx[1] | rx[2] << 8;
155 ////PORTC = ~(uint8_t) start_telling;
    //delay_ms(2000);

    usart_putc(0xE3);
    if (reset_ow()) {
160  usart_putc(0xE1);

     // MatchROM
     usart_putc(0x55);
     for (i = 0; i < 8; i++) {
165    usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
       // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
       if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
         usart_putc(0xE3);
     }
170  for (i = 0; i < 9; i++) {
       usart_getc();
     }

     usart_putc(0xA5);      // Read memory + counter
175  usart_getc();
     // address of last data byte before counter
     address = (counterPage << 5) + 31;
     usart_putc((uchar)(address & 0xFF));
     usart_getc();
180  usart_putc((uchar)(address >> 8));
     usart_getc();

     telle_periode = TIM16_ReadTCNT1();
     ////PORTC = ~(uint8_t) telle_periode;
185  //delay_ms(2000);

     // now add the read bytes for data byte,counter,zero bits, crc16
     for (i = 0; i < 11; i++)
       usart_putc(0xFF);
190  for (i = 0; i < 11; i++) {
       rx[i] = usart_getc();
     }
     slutt_telling = rx[1] | rx[2] << 8;
     ////PORTC = ~(uint8_t) slutt_telling;
195  //delay_ms(2000);


     usart_putc(0xE3);
     if (reset_ow()) {
200    usart_putc(0xE1);
     }
    }
    }
```

```
205   // beregn vindhastigheten i m/s

      ant_rotasjoner = (slutt_telling - start_telling);
      //ant_rotasjoner = ant_rotasjoner >> 1;// (/2.0) TELLER FOR HVER HALVE ROTASJON!!

210   vindhastighet = ant_rotasjoner;

      ////PORTC = ~(uint8_t) ant_rotasjoner;
      //delay_ms(1000);
      /*
215   uint16_t rotasj_clk = ant_rotasjoner / telle_periode;

      uint32_t clock_cyclz = telle_periode * 1024.0;

      uint16_t ms = (uint16_t) (clock_cyclz / 2000000.0) * 10000.0;
220
      //PORTC = ~(uint8_t) ms >> 8;
      delay_ms(2000);
      //PORTC = ~(uint8_t) ms;
      delay_ms(2000);
225

      // calculate the wind speed based on the revolutions per second
      *revol = (((end_count - start_count) * 1000.0) /
                (end_time - start_time)) / 2.0;
230
      */

      return ant_rotasjoner; // åm beregnes til m/s !
    }
235

    /* vindretningssensoren aksesseres via DS2450 (4x a/d)
     * ic'en åp vindsensoren.
     *
240  * DS2450 har familiekode 0x20
     */
    uint16_t avles_vindretning( void ) {

      return(0);
245 }


    /* fuktsensoren aksesseres via DS2438 (temperatur, a/d)
     * ic'en åp fuktsensoren.
250  *
     * DS2438 har familiekoden 0x26
     *
     * metoden inneholder en rutine som snakker med DS2438 ic'en
     * 1. sender 1-wire skip romkode *FIX*
255  * 2. ber DS2438 lese av temperaturen
     * 3. ber DS2438 A/D konvertere volten fra fuktsensoren
     * 4. ber DS2438 dumpe disse verdiene til scratch-pad'en
     * 4. avleser side 0, byte 1,2,3,4 i scratchpaden
     *      byte1 : temperatur LSB
260  *      byte2 : temperatur MSB
     *      byte3 : volt LSB
     *      byte4 : volt MSB
     *
     * Deretter regnes fuktigheten ut og returneres
265  *
     */
    uint16_t avles_fuktighet( uint8_t sensorid ) {

      int16_t temperatur;
270   uint16_t adc_volt;     // fuktsensorvolt
      uint16_t ic_volt;      // DS2438 volt

      uint8_t rx[9];
      uint8_t i;
275
      // snakker med DS2438: avles temp, a/d volt, hent verdier
      if (reset_ow()) {
        usart_putc(0xE1);

280     // MatchROM
        usart_putc(0x55);
        for (i = 0; i < 8; i++) {
          usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
          // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
285       if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
            usart_putc(0xE3);
        }
        for (i = 0; i < 9; i++) {
          usart_getc();
290     }

        usart_putc(0x44);             // start temperaturavlesning
```

```
      usart_getc();

295   usart_putc(0xE3);

      if (reset_ow()) {
        usart_putc(0xE1);

300     // MatchROM
        usart_putc(0x55);
        for (i = 0; i < 8; i++) {
          usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
          // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
305       if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
            usart_putc(0xE3);
        }
        for (i = 0; i < 9; i++) {
          usart_getc();
310     }

        usart_putc(0xB4);           // start a/d konvertering av fuktspenningen
        usart_getc();

315     usart_putc(0xE3);
        if (reset_ow()) {
          usart_putc(0xE1);

          // MatchROM
320       usart_putc(0x55);
          for (i = 0; i < 8; i++) {
            usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
            // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
            if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
325           usart_putc(0xE3);
          }
          for (i = 0; i < 9; i++) {
            usart_getc();
          }
330
          usart_putc(0xB8);                 // legg resultatene i "scratch pad" minnet
          usart_getc();
          usart_putc(0x00);                 // scratch pad : side 0
          usart_getc();
335
          usart_putc(0xE3);

          if (reset_ow()) {
            usart_putc(0xE1);
340
            // MatchROM
            usart_putc(0x55);
            for (i = 0; i < 8; i++) {
              usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
345           // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
              if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
                usart_putc(0xE3);
            }
            for (i = 0; i < 9; i++) {
350           usart_getc();
            }

            usart_putc(0xBE);               // les scratch pad
            usart_getc();
355         usart_putc(0x00);               // side 0
            usart_getc();
            for (i = 0; i < 9; i++)
              usart_putc(0xFF);
            for (i = 0; i < 9; i++) {
360           rx[i] = usart_getc();
            }

            usart_putc(0xE3);

365         if (reset_ow()) {
              temperatur = (int16_t) rx[1];
              temperatur |= (int16_t) (rx[2] << 8);
              adc_volt = (uint16_t) rx[3];
              adc_volt |= (uint16_t) (rx[4] << 8);
370           ////PORTC = ~(int8_t) (temperatur >> 8);     // *DEBUGGING*
              ////PORTC = ~(uint8_t) (adc_volt >> 2);      // *DEBUGGING*
              //delay_ms(2000);                            // *DEBUGGING*
            }
          }
375     }
      }
    }


380 //ø nskerå  lese av vdd åp DS2438 åogs for mer ønyaktig
```

```
      // beregning av fuktigheten (ca. 5V)
      if (reset_ow()) {
        usart_putc(0xE1);

385     // MatchROM
        usart_putc(0x55);
        for (i = 0; i < 8; i++) {
          usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
          // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
390       if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
            usart_putc(0xE3);
        }
        for (i = 0; i < 9; i++) {
          usart_getc();
395     }

        usart_putc(0x4E);    // skriv til scratch pad
        usart_getc();
        usart_putc(0x00);    // skriv til side 0
400     usart_getc();
        usart_putc(rx[0] | 0x08);    // vdd istedet for adc inn åp a/d
        usart_getc();
        usart_putc(0xE3);    // command_mode
        if (reset_ow()) {
405       usart_putc(0xE1);

          // MatchROM
          usart_putc(0x55);
          for (i = 0; i < 8; i++) {
410         usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
            // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
            if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
              usart_putc(0xE3);
          }
415       for (i = 0; i < 9; i++) {
            usart_getc();
          }

          usart_putc(0xB4);          // koverter a/d
420       usart_getc();
          usart_putc(0xE3);
          if (reset_ow()) {
            usart_putc(0xE1);

425         // MatchROM
            usart_putc(0x55);
            for (i = 0; i < 8; i++) {
              usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
              // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
430           if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
                usart_putc(0xE3);
            }
            for (i = 0; i < 9; i++) {
              usart_getc();
435         }

            usart_putc(0xB8);
            usart_getc();
            usart_putc(0x00);
440         usart_getc();
            usart_putc(0xE3);
            if (reset_ow()) {
              usart_putc(0xE1);

445           // MatchROM
              usart_putc(0x55);
              for (i = 0; i < 8; i++) {
                usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
                // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
450             if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
                  usart_putc(0xE3);
              }
              for (i = 0; i < 9; i++) {
                usart_getc();
455           }

              usart_putc(0xBE);
              usart_getc();
              usart_putc(0x00);
460           usart_getc();
              for (i = 0; i < 9; i++)
                usart_putc(0xFF);
              for (i = 0; i < 9; i++) {
                rx[i] = usart_getc();
465           }
              usart_putc(0xE3);
              if (reset_ow()) {
                ic_volt = (uint16_t) rx[3];
```

```
                ic_volt |= (uint16_t) (rx[4] << 8);
470             ////PORTC = ~(uint8_t)(ic_volt >> 2);              // *DEBUGGING*
                //delay_ms(2000);                                 // *DEBUGGING*
              }
            }
          }
475     }
    }

    ////PORTC = ~0x88;
    //delay_ms(2000);
480 // OBS ! DISSE BEREGNINGENEØ KTE KODE ØSTRRELSEN MED 3000B !

  // se gcctest6 for fixed point, float/double -> int

    // temp åm her ævre negative og positive verdier!
485 /*
    // OVERGANGEN INT > FLOAT/DOUBLE, hvilke bit er desimal? hva er sign/magnitude
    int8_t temp;
    float vdd, vad;
    double fukt;
490 // beregn fuktigheten

    vdd = (float) ic_volt;
    vad = (float) adc_volt;
    temp = (int8_t) (temperatur >> 8);
495
    fukt = (((vad/vdd) - 0.16)/0.0062)/(1.0546 - (0.00216 * temp));

    fuktighet = (uint16_t) (fukt);
    //PORTC = ~(uint8_t) (fuktighet >> 8);
500 delay_ms(2000);
    //PORTC = ~(uint8_t) (fuktighet);
    delay_ms(2000);
  */
    return adc_volt;
505
  }


  /* trykksensoren aksesseres via 2 DS2406,
510  * en for lesing og en for skriving
   */
  uint16_t avles_trykk( uint8_t sensorid ) {

    uchar i;
515 int address;
    int counterPage = 15;
    uint16_t start_telling, slutt_telling, telle_periode;
    uint8_t rx[20];

520
    if (reset_ow()) {
     usart_putc(0xE1);

     // MatchROM
525  usart_putc(0x55);
     for (i = 0; i < 8; i++) {
       usart_putc((uint8_t) (sensortab[sensorid].romkode >> (8*i)));
       // åUnng kommando_modus (0xE3) vedå  sende 0xE3 to ganger
       if ((uint8_t) (sensortab[sensorid].romkode >> (8*i)) == 0xE3)
530      usart_putc(0xE3);
     }
     for (i = 0; i < 9; i++) {
       usart_getc();
     }
535
     usart_putc(0xF5);     // Channel access
     usart_getc();
     usart_putc(0x48);     // Read ch.B (PIOB) (ut fra trykksensoric'en)
     usart_getc();
540  usart_putc(0xFF);     // Reservert byte
     usart_getc();

     usart_putc(0xFF);     // Sende foråmotta info byte
     //PORTC = ~usart_getc();
545  delay_ms(2000);

     usart_putc(0xFF);
     //PORTC = ~usart_getc();
     delay_ms(2000);
550
    }

    return(0);
  }
555
```

```
/* ønedbrsmleren aksesseres via DS2423
 */
uint16_t avles_nedbor( void ) {

  return(0);
}


/* lyssensoren sitter åp et egentprodusert
 * lyssensorkort, og er en analog enhet
 */
uint16_t avles_lys( void ) {
  return analog_sensor();
}


/* østtte for en generell ekstra sensor
 */
uint16_t avles_generell( void ) {

  return(0);
}


/* sensorer som er kobla fra 1-wire bussen
 * skal enten legges i minnet med en NULL
 * post eller ikke lagres i det hele tatt
 */
uint16_t avles_ingen( void ) {

  return(0);
}
```

Listing 7: Sensorkort/sensorer.h

```
#ifndef sensor_avlesn
#define sensor_avlesn

#include <inttypes.h>
#include "ow.h"

//extern int16_t temperatur;
//extern uint16_t fuktighet;
//extern uint16_t vindhastighet;

/* temperatursensoren aksesseres via DS2406(trykk)
 * eller DS2438(fukt)
 */
uint16_t avles_temperatur( uint8_t sensorid );

/* vindhastighetssensoren aksesseres via DS2450
 */
uint16_t avles_vindhastighet( uint8_t sensorid );

/* vindretningssensoren aksesseres via DS2450
 */
uint16_t avles_vindretning( void );

/* fuktsensoren aksesseres via DS2438
 */
uint16_t avles_fuktighet( uint8_t sensorid );

/* trykksensoren aksesseres via 2 DS2406,
 * en for lesing og en for skriving
 */
uint16_t avles_trykk( uint8_t sensorid );

/* ønedbrsmleren aksesseres via DS2423
 */
uint16_t avles_nedbor( void );

uint16_t avles_lys( void );

uint16_t avles_generell( void );

uint16_t avles_ingen( void );

#endif
```

Listing 8: Sensorkort/teller.c

```
#include "teller.h"
```

```
void delay_ms(uint16_t msek) {
  uint16_t teller,stop;
  stop = msek * 7; // overflower 7 ganger åp 0.996 ms. ved 1.8432 MHz
  TCCR0 = 0x01;
  for (teller = 0; teller < stop; teller++) {
    while(!(TIFR & 0x02)); TIFR = 0x02;
  }
  TCCR0 = 0x00;
}


unsigned int TIM16_ReadTCNT1( void ) {
  unsigned char sreg;
  unsigned int i;
  /* Save global interrupt flag */
  sreg = SREG;
  /* Disable interrupts */
  //_CLI();
  cli();
  /* Read TCNT1 into i */
  i = TCNT1;
  /* Restore global interrupt flag */
  SREG = sreg;
  return i;
}


void TIM16_WriteTCNT1 ( uint16_t i ) {
  uchar sreg;
  uint16_t i;
  sreg = SREG;
  cli();
  TCNT1 = i;
  SREG = sreg;
}

void start_teller( void ) {
  // nullstill timer/counter1
  TIM16_WriteTCNT1(0);
  // setter klokkekilden til prescaleren til clk(i/o)/1024
  TCCR1B = (1 << CS12) | (1 << CS10);
}

void stopp_teller( void ) {
  // åslr av telleren
  TCCR1B &= ~(1 << CS12) | (1 << CS10);
  // nullstill timer/counter1
  TIM16_WriteTCNT1(0);
}
```

Listing 9: Sensorkort/teller.h

```
#include "ow.h"
#include <avr/io.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

void delay_ms(uint16_t ms);

uint16_t TIM16_ReadTCNT1( void );

void start_teller( void );

void stopp_teller( void );
```

Listing 10: Sensorkort/time.c

```
#include "time.h"


// Enkle makroer forå  initsiere/stoppe en øoverfring
#define RTC_Enable() RTC_PORT &= ~RTC_Clk; RTC_PORT |= RTC_nReset
#define RTC_Disable() RTC_PORT &= ~RTC_nReset



// Disse klokker 8 bit inn eller ut fra realtime-klokken.
// Initsialiserer ikke chipen øfrst.
// Til vanlig bruk, benytt rtc_write/rtc_read istede
void rtc_send(uint8_t _arg);
void rtc_get(uint8_t *_arg);

```

```
void rtc_init(void) {
  RTC_DDR |= (RTC_nReset | RTC_Clk); // DDRD<=1?1????? Pin nr 7 og 5 blir output.
  RTC_PORT &= ~RTC_IO; // Sett nReset lav.
}

void rtc_send(uint8_t _arg) {
  uint8_t teller,arg;
  arg = _arg;
  RTC_DDR |= RTC_IO;   // Sett RTC_IO til output
  for (teller = 0; teller < 8; teller++) {
    delay_ms(1);
    RTC_PORT &= ~RTC_Clk; // Sett klokke lav
    RTC_PORT &= ~RTC_IO; // Sett IOpin lav, øfr en evnt setter den øhy
    RTC_PORT |= ((arg & 0x01) << RTC_IO_Pin); // Send bit 0 av arg til IO pin
    delay_ms(1);
    arg >>= 1;
    RTC_PORT |= RTC_Clk; // Sett klokke øhy
  }
}

// FIXME : denne tror den leser msb. fiks det til at den øskjnner at den skal ha lsb
void rtc_get(uint8_t *_arg) {
  // Antar øhy klokke
  uint8_t teller;
  RTC_PORT &= ~RTC_IO; // Sett IO-bit i portregister lik 0 foråå unng tristate
  RTC_DDR &= ~RTC_IO; // Setter RTC_IO til input
  for (teller = 0; teller < 8; teller++) {
    delay_ms(1);
    RTC_PORT &= ~RTC_Clk; // Sett klokke lav
    *_arg >>= 1;
    delay_ms(1); // Vent 1 ms
    // Trenger ikke initsiere bit til 0, siden leftshift øgjr det automatisk.

    *_arg |= ((RTC_PIN & RTC_IO) << (7-RTC_IO_Pin));
    RTC_PORT |= RTC_Clk; // Sett klokke øhy
  }
}


void rtc_write(uint8_t _com, uint8_t _arg) {
  uint8_t com = _com & ~RTC_LesAv;

  RTC_Enable();
  rtc_send(com);
  rtc_send(_arg);
  RTC_Disable();
}

void rtc_read(uint8_t _com, uint8_t *_arg) {
  uint8_t com = _com | RTC_LesAv;

  RTC_Enable();
  rtc_send(com);
  rtc_get(_arg);
  RTC_Disable();
}


void rtc_setClock(uint8_t *_time) {
  // Skriver sekundregisteret sist, siden dette vil starte klokken.
  // Kan ikke settes direkte siden klokken lagrer verdiene åp en rar åmte.
  uint8_t tmp;
  // Min
  RTC_Stop_Klokke();
  tmp = (_time[1] / 10) << 4;  tmp |= (_time[1] % 10);
  rtc_write(RTC_Minutt, tmp);
  // Time
  tmp = (_time[2] / 10) << 4;  tmp |= (_time[2] % 10);
  rtc_write(RTC_Time, tmp);
  // Dato
  tmp = (_time[3] / 10) << 4;  tmp |= (_time[3] % 10);
  rtc_write(RTC_Dato, tmp);
  // åMned
  tmp = (_time[4] / 10) << 4;  tmp |= (_time[4] % 10);
  rtc_write(RTC_Maaned, tmp);
  //Å r
  tmp = (_time[5] / 10) << 4;  tmp |= (_time[5] % 10);
  rtc_write(RTC_Aar, tmp);
  // Sekund
  tmp = (_time[0] / 10) << 4;  tmp |= (_time[0] % 10);
  rtc_write(RTC_Sekund, tmp);
}

void rtc_getClock(uint8_t *_time) {
  uint8_t tmp, debug;
  rtc_read(RTC_Sekund, &tmp);
```

```
    _time[0] = (tmp >> 4)*0x0a + (tmp % 0x10); debug = _time[0];
    rtc_read(RTC_Minutt, &tmp);
    _time[1] = (tmp >> 4)*10 + (tmp % 0x10); debug = _time[1];
    rtc_read(RTC_Time, &tmp);
    _time[2] = (tmp >> 4)*10 + (tmp % 0x10); debug = _time[2];
    rtc_read(RTC_Dato, &tmp);
    _time[3] = (tmp >> 4)*10 + (tmp % 0x10); debug = _time[3];
    rtc_read(RTC_Maaned, &tmp);
    _time[4] = (tmp >> 4)*10 + (tmp % 0x10); debug = _time[4];
    rtc_read(RTC_Aar, &tmp);
    _time[5] = (tmp >> 4)*10 + (tmp % 0x10); debug = _time[5];
  }

  //år-6 måned-4 dato-5 time-5 min-6 sek-6   totalt 32 bit
  // bruker bare 6 bit til årstall, dvs 0-63
  #define tid_aar     6
  #define tid_maaned  4
  #define tid_dato    5
  #define tid_time    5
  #define tid_min     6
  #define tid_sek     6
  #define tid_aar_offset     26
  #define tid_maaned_offset  22
  #define tid_dato_offset    17
  #define tid_time_offset    12
  #define tid_min_offset      6
  #define tid_sek_offset      0
  #define tid_aar_map     0xfc000000
  #define tid_maaned_map  0x03c00000
  #define tid_dato_map    0x003e0000
  #define tid_time_map    0x0001f000
  #define tid_min_map     0x00000fc0
  #define tid_sek_map     0x0000003f

  void rtc_convert_to_32bit(uint8_t *_time, uint32_t *_time32) {
    *(_time32) =
      (((uint32_t)_time[0]) << tid_sek_offset) |
      (((uint32_t)_time[1]) << tid_min_offset) |
      (((uint32_t)_time[2]) << tid_time_offset) |
      (((uint32_t)_time[2]) << tid_dato_offset) |
      (((uint32_t)_time[4]) << tid_maaned_offset) |
      (((uint32_t)_time[3]) << tid_aar_offset);
  }

  void rtc_convert_from_32bit(uint8_t *_time, uint32_t _time32) {
    _time[0] = (_time32 & tid_sek_map) >> tid_sek_offset;
    _time[1] = (_time32 & tid_min_map) >> tid_min_offset;
    _time[2] = (_time32 & tid_time_map) >> tid_time_offset;
    _time[3] = (_time32 & tid_dato_map) >> tid_dato_offset;
    _time[4] = (_time32 & tid_maaned_map) >> tid_maaned_offset;
    _time[5] = (_time32 & tid_aar_map) >> tid_aar_offset;
  }
```

Listing 11: Sensorkort/time.h

```
  #ifndef _Real_Time_Clock_
  #define _Real_Time_Clock_

  #include <inttypes.h>
  #include <avr/io.h>

  // Diverse kommando/register-bits som kan settes
  #define RTC_LesAv 0x01
  #define RTC_Write_Protect 0x80
  #define RTC_nWrite_Protect 0x7f // Logisk negasjon av RTC_Write_Protect
                                  // Egentlig helt øundvendig, men slapp unna med en warning mindre
                                  // Ferdig preprossesert kode blir identisk, om ikke enklere
  #define RTC_Klokke_Stop_bit 0x80

  // De forskjellige kommando-bytene som klokken åforstr.
  #define RTC_Sekund 0x80
  #define RTC_Minutt 0x82
  #define RTC_Time 0x84
  #define RTC_Dato 0x86
  #define RTC_Maaned 0x88
  #define RTC_Dag 0x8A
  #define RTC_Aar 0x8C
  #define RTC_Kontroll 0x8E
  #define RTC_Trickle_Charge 0x90
  #define RTC_Clock_Burst 0xBE

  // Definerer hvilken port som skal brukes.
  #define RTC_PORT PORTD
  #define RTC_DDR DDRD
  #define RTC_PIN PIND
```

```
// Definerer hvilke pin de forskjellige signalene skal åg åp
// og hvilke øbitmnster som kan brukes forå sette dem.
#define RTC_nReset      0x80    // øBitmnster for nReset
#define RTC_nReset_Pin    7     // Pin nr for nReset
#define RTC_IO          0x40    // øBitmnster for IO
#define RTC_IO_Pin        6     // Pin nr for IO
#define RTC_Clk         0x20    // øBitmnster for Clk
#define RTC_Clk_Pin       5     // Pin nr for Clk


// Enkel makroer forå enable/disable skrivebeskyttelse for
// klokkechip.
#define RTC_WPEnable() rtc_write(RTC_Kontroll, RTC_Write_Protect)
#define RTC_WPDisable() rtc_write(RTC_Kontroll, RTC_nWrite_Protect)

// Vil starte klokketelleren, men vil åogs nullstille sekundregisteret.
// Egentlig ikke øndvendig, da man oftest vil starte klokken samtidig
// som en stiller sekundregisteret
#define RTC_Start_Klokke() rtc_write(RTC_Sekund, 0x00)

// Vil stoppe klokketelleren, men vil åogs nullstille sekundregisteret.
#define RTC_Stop_Klokke() rtc_write(RTC_Sekund, RTC_Klokke_Stop_bit)

// Initsialiserer Porten som skal benyttes
void rtc_init(void);

// Bruk disse forå lese/skrive til klokken, siden disse automatisk setter nReset øhy.
// Kan åogs bruke de over, men det kreves litt mer manuell fikling.
void rtc_write(uint8_t _com, uint8_t _arg);
void rtc_read(uint8_t _com, uint8_t *_arg);


// Stiller klokken _time er her et 6-bytes buffer, hvor hver byte inneholder en innstilning for hvert av registerene (sek, min, time, data, åmnedå
void rtc_setClock(uint8_t *_time);

// Leser av klokken til _time, som er et 6-bytes buffer. (se rtc_setClock(..))
void rtc_getClock(uint8_t *_time);

void rtc_convert_to_32bit(uint8_t *_time, uint32_t *_time32);
void rtc_convert_from_32bit(uint8_t *_time, uint32_t _time32);
#endif
```

Listing 12: Sensorkort/usart.c

```
/*****************************************************************************
 * Atmel AVR USART Library for GCC
 * Version: 1.0
 *
 * Works with AVR MCUs equiped with USART hardware (ATmega series).
 * Does not work with older UART's or USI of ATtiny series.
 * Tested with ATmega8.
 *
 * Uses USART Receive Complete Interrupt. Disabling Global Interrupts
 * after usart initialization disables data receive.
 *
 * Jaakko Ala-Paavola 2003/06/28
 * http://www.iki.fi/jap email:jap@iki.fi
 */

#include <avr/io.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include <string.h>
#include "usart.h"

char usart_buffer[USART_BUFFER_SIZE];
volatile unsigned char usart_buffer_pos_first = 0, usart_buffer_pos_last = 0;
volatile unsigned char usart_buffer_overflow = 0;

void usart_init(unsigned char baud_divider) {

  // Baud rate selection
  UBRRH = 0x00;
  UBRRL = baud_divider;
  //UBRRL = 12;   // 2MHz
  //UBRRL = 25;  // 4MHz

  // USART setup
  UCSRC = 0x86;         // 1000 0110
                        // Access UCSRC, Asyncronous 8N1
  UCSRB = 0x98;         // 1001 1000
                        // Receiver enabled, Transmitter enabled
                        // RX Complete interrupt enabled
  sei();                // Enable interrupts globally
```

```
}

void usart_putc(char data) {
    while (!(UCSRA & 0x20)); // Wait untill USART data register is empty
    // Transmit data
    UDR = data;
}

void usart_puts(char *data) {
  int len, count;

  len = strlen(data);
  for (count = 0; count < len; count++)
    usart_putc(*(data+count));
}

char usart_getc(void) {
  // Wait untill unread data in ring buffer
  if (!usart_buffer_overflow) {
    while(usart_buffer_pos_first == usart_buffer_pos_last) {
                }

        }
  usart_buffer_overflow = 0;
  // Increase first pointer
  if (++usart_buffer_pos_first >= USART_BUFFER_SIZE)
    usart_buffer_pos_first = 0;
  // Get data from the buffer
  return usart_buffer[usart_buffer_pos_first];
}

unsigned char usart_unread_data(void) {
  if (usart_buffer_overflow)
    return USART_BUFFER_SIZE;
  if (usart_buffer_pos_last > usart_buffer_pos_first)
    return usart_buffer_pos_last - usart_buffer_pos_first;
  if (usart_buffer_pos_last < usart_buffer_pos_first)
    return USART_BUFFER_SIZE-usart_buffer_pos_first
      + usart_buffer_pos_last;
  return 0;
}

SIGNAL(SIG_UART_RECV) {
  // Increase last buffer
  if (++usart_buffer_pos_last >= USART_BUFFER_SIZE)
    usart_buffer_pos_last = 0;
  if (usart_buffer_pos_first == usart_buffer_pos_last)
    usart_buffer_overflow++;
  // Put data to the buffer
  usart_buffer[usart_buffer_pos_last] = UDR;
}
```

Listing 13: Sensorkort/usart.h

```
/************************************************************************
 * Atmel AVR USART Library for GCC
 * Version: 1.0
 *
 * Works with AVR MCUs equiped with USART hardware (ATmega series).
 * Does not work with older UART's or USI of ATtiny series.
 * Tested with ATmega8.
 *
 * Uses USART Receive Complete Interrupt. Disabling Global Interrupts
 * after usart initialization disables data receive.
 *
 * Jaakko Ala-Paavola 2003/06/28
 * http://www.iki.fi/jap email:jap@iki.fi
 */

// Size of receive ring buffer. Must be at least 2.
#define USART_BUFFER_SIZE 100

/* Baudrate settings. Refer to datasheet for baud rate error.
   Note also maximun baud rate.
   br = baudrate, fosc = clock frequency in megahertzs */
#define USART_BAUDRATE(br, fosc) (fosc*125000/br-1)

/* Initializes USART device. Use USART_BAUDRATE macro for argument or
   consult datasheet for correct value. */
void usart_init(unsigned char baud_divider);

/* Transmit one character. No buffering, waits until previous character
   transmitted. */
void usart_putc(char a);

/* Transmit string. Returns after whole string transmitted. */
```

```
void usart_puts(char *data);

/* Receive one character. Blocking operation, if no new data in buffer. */
char usart_getc(void);

/* Returns number of unread character in ring buffer. */
unsigned char usart_unread_data(void);
```

Listing 14: Sensorkort/memory.c

```
#include "memory.h"
#include "memutils.h"


uint8_t df_global_post[postsize];
uint16_t df_current_page;
uint16_t df_current_post;
uint32_t df_current_postid;

void df_readDirectly(uint16_t _page, uint16_t _addr, uint8_t *_buffer, uint16_t _length) {
  uint16_t count;
  df_EnableChip();
  df_command(DF_read_from_page, _page, _addr);
  DF_send8(0x00);
  DF_send8(0x00);
  DF_send8(0x00);
  DF_send8(0x00);
  for (count = 0; count < _length; count ++) {
          //PORTC=~count;
    _buffer[count] = DF_get8();
  }
  df_DisableChip();
}

uint8_t df_store_post(uint8_t *_post) {
  uint8_t count;
  if (_post == 0x0000) return 0x00;
  else {
    for (count = 0; count < postsize; count++) {
      df_global_post[count] = _post[count];
    }
    return df_store_gpost();
  }
}



uint8_t df_store_gpost(void) {
  EnableSPI();
  uint8_t buffer[8];

  // Skjekk om fullt minne
  if ((df_current_page == 1) && (df_current_post == 0) && (df_current_postid != 0)) return 0;
  else {

    // hent inn rett page til buf1
    df_readPagetoBuffer1(df_current_page);

    // Sett postid
    df_global_post[0] = df_current_postid >> 24;
    df_global_post[1] = (df_current_postid >> 16) % 0x100;
    df_global_post[2] = (df_current_postid >> 8) % 0x100;
    df_global_post[3] = df_current_postid % 0x100;

    // skriv til buf1
    df_writetoBuffer1(df_current_post * postsize, df_global_post, postsize);

    // send buf1 tilbake til page
    df_writeBuffer1toPage(df_current_page, 1);

    // oppdater page0
    // les inn  page 0
    df_readPagetoBuffer1(0x0000);

    // Beregn nye verdier
    df_current_post += 1;
    df_current_post %= post_per_page;
    if (df_current_post == 0) df_current_page++;
    df_current_page %= total_pages;
    if (df_current_page == 0) df_current_page++;
    df_current_postid += 1;
    buffer[0] = df_current_page >> 8;
    buffer[1] = df_current_page % 0x100;
```

```
      buffer[2] = df_current_post >> 8;
      buffer[3] = df_current_post % 0x100;
      buffer[4] = df_current_postid >> 24;
      buffer[5] = (df_current_postid >> 16) % 0x100;
      buffer[6] = (df_current_postid >>  8) % 0x100;
      buffer[7] = df_current_postid % 0x100;

      // skriv til buffer
      df_writetoBuffer1(0x0000, buffer, 8);

      // send tilbake til page 0
      df_writeBuffer1toPage(0,1);

      DisableSPI();
      return 1;
    }
  }

void df_get_glast(void) {
  //  uint32_t post_addr;
  uint16_t current_page, current_post;
  EnableSPI();

  // Finn riktig adresse

  // Hvis currentpost = 0 da skal vi hente siste post åp forige side
  // Hvis ikke, hent currentpage:(currenpost-1)
  if (df_current_post == 0) {
    current_post = post_per_page -1;
    current_page = df_current_page -1;
  } else
      current_page = df_current_page;
    current_post = df_current_post -1;

  // Hvis currentpage = 0 (skjer hvis get_last kalles øfr øfrste innskriving)
  // ås les av fra side 1 istede. (bare nullere)
  if (current_page == 0) current_page = 1;

  // les rett fra page
  df_readDirectly(current_page, current_post*postsize, df_global_post, postsize);

  DisableSPI();
}

uint8_t df_get_post(uint32_t _time, uint8_t *_buffer) {
  uint8_t returnvalue, count;
  if (_buffer == 0) returnvalue = 0;
  else {
    df_get_gpost(_time);
    for (count = 0; count < postsize; count++) {
      _buffer[count] = df_global_post[count];
    }
    returnvalue = 1;
  }
  return returnvalue;
}

uint8_t df_get_gpost(uint32_t _time) {
  uint32_t time_mid, time_end, found_time;
  uint16_t Bpage, Mpage, Epage;
  uint16_t Bpost, Mpost, Epost;
  uint8_t returnvalue;

  uint8_t mtbuffer[4];
  uint8_t etbuffer[4];

  EnableSPI();


  Bpage = 1;
  Epage = df_current_page; if ((df_current_post == 0) && (df_current_page != 1)) Epage--;

  while (Bpage < Epage) {
    Mpage = (Bpage + Epage) / 2;

    /* finn time_end,time_mid */
    df_readDirectly(Mpage,0x0004, mtbuffer, 4);
    time_mid =
      ((uint32_t)mtbuffer[0] << 24) + ((uint32_t)mtbuffer[1] << 16) +
      ((uint32_t)mtbuffer[2] << 8) + mtbuffer[3];
    df_readDirectly(Epage,0x0004, etbuffer, 4);
    time_end =
      ((uint32_t)etbuffer[0] << 24) + ((uint32_t)etbuffer[1] << 16) +
      ((uint32_t)etbuffer[2] << 8) + etbuffer[3];

    if      (time_end <= _time)
      Bpage = Epage;
    else if (Mpage == Bpage)
```

```
165       Epage = Mpage;
        else if (time_mid <  _time)
          Bpage = Mpage;
        else  /* time_mid>=_time */
          Epage = Mpage;
170     }
        if (Bpage == Epage) {
          Bpost = 0;


175       if (Epage == df_current_page)
            Epost = df_current_post -1;
          else
            Epost = post_per_page -1;

180
    /*        Epost = df_current_post;
              if (Epost == 0)
                Epost = post_per_page-1;
              else
185             Epost--;
    */
          while (Bpost < Epost) {
            Mpost = (Bpost + Epost) / 2;

190         /* finn time_end og time_mid */
            df_readDirectly(Bpage,(Mpost*postsize)+0x0004,mtbuffer,4);
            time_mid =
              ((uint32_t)mtbuffer[0] << 24) + ((uint32_t)mtbuffer[1] << 16) +
              ((uint32_t)mtbuffer[2] << 8) + mtbuffer[3];
195         df_readDirectly(Bpage,(Epost*postsize)+0x0004,etbuffer,4);
            time_end =
              ((uint32_t)etbuffer[0] << 24) + ((uint32_t)etbuffer[1] << 16) +
              ((uint32_t)etbuffer[2] << 8) + etbuffer[3];

200         if       (time_end <= _time)
              Bpost = Epost;
            else if (Mpost == Bpost)
              Epost = Bpost;
            else if (time_mid <= _time)
205           Bpost = Mpost;
            else  /* time_mid>_time) */
              Epost = Mpost;
          }

210       /* Les post inn i global_post */
          df_readDirectly(Bpage,Bpost*postsize,df_global_post,postsize);

          /*
           * Les av found_time fra df_global_post
215        * Dette fordi det ikke er sikkert at time_mid er initisert ved slutt
           * åM ogsa gjores hvis _time > end_time
           */
            found_time =
              ((uint32_t)df_global_post[4] << 24) + ((uint32_t)df_global_post[5] << 16) +
220           ((uint32_t)df_global_post[6] << 8) + df_global_post[7];

          if (found_time == _time)
            returnvalue = 2;
          else
225         returnvalue = 1;

        } else { returnvalue = 0; }
        DisableSPI();
        return returnvalue;
230   }

      uint8_t df_get_id(uint32_t _postid, uint8_t *_buffer) {
        uint8_t count,retval;
        if ((_buffer == 0) || (_postid >= df_current_postid)) retval = 0;
235     else {
          df_get_gid(_postid);
          for (count = 0; count < postsize; count++)
            _buffer[count] = df_global_post[count];
          retval = 1;
240     }
        return retval;
      }

      uint8_t df_get_gid(uint32_t _postid) {
245     if (_postid >= df_current_postid) return 0;
        else {
              EnableSPI();
          df_readDirectly((_postid / post_per_page)+1,          // page nr
                          (_postid % post_per_page) * postsize,  // byte nr
250                       df_global_post,                        // buffer
                          postsize);                             // lengde
          DisableSPI();
```

```
    return 1;
  }
}



void df_reset_chip(void) {
  uint16_t count;
  uint8_t tmp_page_pointer[8] = {0x00, 0x01,             // side nr.
                                 0x00, 0x00,             // post nr.
                                 0x00, 0x00, 0x00, 0x00 // postid
                                 };

  EnableSPI();

  // Erase content on chip
  for (count = 0; count < mem_blocks; count++) {
        //PORTC=~count;
    df_EnableChip();
        df_command(DF_eraseblock, count<<3, 0x00);
    df_DisableChip();
  }

  // Initsiere data åp page 0
  // Hent inn page 0 til buffer 1
  df_readPagetoBuffer1(0x0000);

  // Skriv sett active pagenr til 1
  df_writetoBuffer1(0x0000, tmp_page_pointer, 8);

  // Skriv den oppdaterte page 0 tilbake
  df_writeBuffer1toPage(0x0000, 0x01);

  DisableSPI();

  df_init();
}

void df_init(void) {
  uint8_t buffer[8];
  EnableSPI();

  df_readDirectly(0x0000,0x0000,buffer,8);
  df_current_page = ((uint16_t)buffer[0] << 8) + buffer[1];
  df_current_post = ((uint16_t)buffer[2] << 8) + buffer[3];;
  df_current_postid =
    ((uint32_t)buffer[4] << 24) + ((uint32_t)buffer[5] << 16) +
    ((uint32_t)buffer[6] << 8) + buffer[7];

  DisableSPI();
}

uint8_t df_fill(void) {
  return (uint8_t) ( ((float)df_current_postid*100) / ((float) (post_per_page * (total_pages-1))));
}
uint32_t df_leftspace(void) {
  return ((post_per_page - (uint32_t)df_current_post) + ((total_pages - ((uint32_t)df_current_page+1))*post_per_page)) * postsize;
}
```

Listing 15: Sensorkort/memory.h

```
#ifndef avr_memory
#define avr_memory


#include <inttypes.h>
#include <avr/io.h>

#define postsize 57
#define max_sensors 16

// Inkluder fil med brikkespesifikke spesifikasjoner
//#include "at45db642.h"
#include "at45db011.h"


#define SPSR_ready 0x80

#define DF_nRESET 0x01
#define DF_RDY    0x02
#define DF_nWP    0x04
#define DF_nCE    0x10
#define DF_MOSI   0x20
```

```
#define DF_MISO    0x40
#define DF_SCK     0x80

/**
 * Side 0 av brikken benyttes tilå lagre informasjon
 * om "filsystemet" åp brikken. Dvs hvor langt åp brikken er
 * skrivingen kommet, og hvor mange poster er lagret
 * Denne informasjonen ligger lagret øfrst med
 * Sidenr, derretter
 * Postnr, og til slutt
 * Postid. (unik verdi for hver post.)
 **/

/**
 * Post åbestr av :
 * ID(32 bit) Timestamp(32 bit) Metadata(8bit) Sensorinfo(384 bit)
 * - ID er unik id. Settes av memorymodulen
 * - Timestamp åbestr av
 * å  r(7 bit) åmned (4bit) dato(5bit) time(6bit) min(6) reserved(4)
 * - Metadata åbestr ikke av :
 *   ant sensorer(4 bit)  reservert (4 bit)
 * - Sensorinfo åbestr av
 *   16 stk (Metadata(8 bit) Sensordata (16 bit)
 *   - Metadata åbestr av
 *     Sensorid(4 bit) reserved(4 bit)
 *   - Sensorinfo (16 bit årdata)
 */

extern uint8_t df_global_post[postsize];
extern uint16_t df_current_page;
extern uint16_t df_current_post;
extern uint32_t df_current_postid;
/**
 * Leser direkte fra dataflash inn i en buffer angitt av _buffer.
 * Leser inn _lenght anntall bytes.
 **/
void df_readDirectly(uint16_t _page, uint16_t _addr, uint8_t *_buffer, uint16_t _length);


/**
 * Stores a post to flash-memory
 * uint8_t *_post points to a 53byte array containg the post.
 * The post is assumed to be 53 byte in the correct order.
 * Does this by storing *_post in global_post and calling store_gpost()
 * Return 1 on success, or 0 on fail (_post = NULL or memory full)
 **/
uint8_t df_store_post(uint8_t *_post);

/**
 * Stores a post to flash-memory
 * Uses global_post to read from
 * Returns 1 on success, and 0 on fail (memory full)
 **/
uint8_t df_store_gpost(void);

/**
 * Retrieves a post from flash-memory
 * Uses _time as reference, and stores to _buffer
 * Uses get_gpost, and copies global_post to buffer
 **/
uint8_t df_get_post(uint32_t _time, uint8_t *_buffer);

/**
 * Retrieves a post from flash-memory
 * Uses _time as reference, and stores to global_post
 **/
uint8_t df_get_gpost(uint32_t _time);

/**
 * Henter inn den sist lagrede posten inn i global_post
 **/
void df_get_glast(void);

/**
 * Henter inn en post fra minnebrikken basert åp _postid og ikke timestamp.
 * Lagrer posten i df_global_post
 **/
uint8_t df_get_gid(uint32_t _postid);

/**
```

```
     * Samme som over, men kopierer df_global_post over i _buffer
     **/
    uint8_t df_get_id(uint32_t _postid, uint8_t *_buffer);

115 /**
     * Sletter innholdet åp en chip
     **/
    void df_reset_chip(void);

120 /**
     * Henter inn current_page og current_post fra side 1 av brikken
     **/
    void df_init(void);

125
    uint8_t df_fill(void);

    uint32_t df_leftspace(void);

130

    #endif
```

Listing 16: Sensorkort/memutils.c

```
    #include "memutils.h"
    #include "memory.h"

    void delay_ms(uint16_t msek) {
 5    uint16_t teller,stop;
      stop = msek * 8;
      TCCR0 = 0x01;
      for (teller = 0; teller < stop; teller++) {
        while(!(TIFR & 0x02));
10      TIFR = 0x02;
      }
      TCCR0 = 0x00;
    }

15  void df_EnableChip(void) {
      delay_ms(30);
      //while (!(PINB & 0x02));
      PORTB &= ~DF_nCE;
    }
20
    void df_DisableChip(void) {
      delay_ms(300);
      PORTB |= DF_nCE;
    }
25
    void DF_send8(uint8_t value) {
      SPDR = value;
      while(!(SPSR&SPSR_ready)); }
    void DF_send16(uint16_t value) {
30    DF_send8(value >> 8);
      DF_send8(value % 0x100); }
    void DF_send24(uint32_t value) {
      DF_send8((value >> 16) % 0x100);
      DF_send8((value >>  8) % 0x100);
35    DF_send8(value % 0x100); }
    void DF_send32(uint32_t value) {
      DF_send8( value >> 24);
      DF_send8((value >> 16) % 0x100);
      DF_send8((value >>  8) % 0x100);
40    DF_send8(value % 0x100); }

    // 0xff er en dummy-verdi. Kan ævre hva som helst
    // Skriving til SPDR starter SPI-øoverfringen automatisk
    // åNr skriving er ferdig, ås kan øoverfrt verdi leses rett av SPDR
45  uint8_t DF_get8(void) {
      SPDR = 0xff;
      while(!(SPSR&SPSR_ready));
      return SPDR; }
    uint16_t DF_get16(void) {
50    uint16_t tmp;
      SPDR = 0xff;
      while(!(SPSR&SPSR_ready));
      tmp =  SPDR; tmp <<= 8;
      SPDR = 0xff;
55    while(!(SPSR&SPSR_ready));
      tmp += SPDR;
      return tmp; }
    uint32_t DF_get24(void) {
      uint32_t tmp;
60    SPDR = 0xff;
      while(!(SPSR&SPSR_ready));
```

```
  tmp =  SPDR; tmp <<= 8;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp += SPDR; tmp <<= 8;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp += SPDR;
  return tmp; }
uint32_t DF_get32(void) {
  uint32_t tmp;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp =  SPDR; tmp <<= 8;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp += SPDR; tmp <<= 8;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp += SPDR; tmp <<= 8;
  SPDR = 0xff;
  while(!(SPSR&SPSR_ready));
  tmp += SPDR;
  return tmp; }

void df_readPagetoBuffer1(uint16_t _page) {
  df_EnableChip();
  df_command(DF_read_page_to_buffer_1, _page, 0x00);
  df_DisableChip();
}
void df_writeBuffer1toPage(uint16_t _page, uint8_t _erase) {
  df_EnableChip();
  if (_erase)
    df_command(DF_write_buffer_1_erase, _page, 0x00);
  else
    df_command(DF_write_buffer_1_no_erase, _page, 0x00);
  df_DisableChip();
}
void df_writetoBuffer1(uint16_t _addr, uint8_t *_buffer, uint16_t _length) {
  uint16_t count;
  df_EnableChip();
  df_command(DF_write_to_buffer_1, 0x00, _addr);
  for (count = 0; count < _length; count++) {
    DF_send8(_buffer[count]);
    //PORTC=~count;
  }
  df_DisableChip();
}

void df_command(uint8_t _opcode, uint16_t _page, uint16_t _byte) {
        DF_send8(_opcode);
        DF_send8(_page >> 7);
        DF_send8(((_page << 1) % 0x100) | _byte >> 8);
        DF_send8(_byte % 0x100);
}
```

Listing 17: Sensorkort/memutils.h

```
#ifndef memutils
#define memutils

#include <inttypes.h>
#include <avr/io.h>

void df_command(uint8_t _opcode, uint16_t _page, uint16_t _byte);

void delay_ms(uint16_t msek);

#define EnableSPI() SPCR = 0x5c
#define DisableSPI() SPCR = 0x00

void df_EnableChip(void);

void df_DisableChip(void);

void DF_send8(uint8_t value);

void DF_send16(uint16_t value);

void DF_send24(uint32_t value);

void DF_send32(uint32_t value);

uint8_t DF_get8(void);
```

```
uint16_t DF_get16(void);

uint32_t DF_get24(void);

uint32_t DF_get32(void);

void df_readPagetoBuffer1(uint16_t _page);

void df_writeBuffer1toPage(uint16_t _page, uint8_t _erase);

void df_writetoBuffer1(uint16_t _addr, uint8_t *_buffer, uint16_t _length);

void df_command(uint8_t _opcode, uint16_t _page, uint16_t _byte);

#endif
```

Listing 18: Sensorkort/memory.h

```
#ifndef _at45db642_h_
#define _at45db642_h_

// Fysiske østrrelser
#define total_pages   8192
#define page_size     1056
#define post_per_page   19
#define mem_blocks        1024

// Kommandoer
#define DF_eraseblock 0x50
#define DF_read_page_to_buffer_1 0x53
#define DF_read_from_page 0x52
#define DF_write_to_buffer_1 0x84
#define DF_write_buffer_1_no_erase 0x88
#define DF_write_buffer_1_erase 0x83



#endif
```

Listing 19: Sensorkort/at45db011.h

```
#ifndef _at45db011_
#define _at45db011_


// Fysiske østrrelser
#define total_pages   512
#define page_size     264
#define post_per_page   4
#define mem_blocks     64

// Kommandoer
#define DF_eraseblock 0x50
#define DF_read_page_to_buffer_1 0x53
#define DF_read_from_page 0x52
#define DF_write_to_buffer_1 0x84
#define DF_write_buffer_1_no_erase 0x88
#define DF_write_buffer_1_erase 0x83



#endif
```

Listing 20: Sensorkort/usb_driver.c

```
//#include <inttypes.h>
//#include <avr/io.h>
//#include <avr/interrupt.h>
//#include <avr/signal.h>
#include <string.h>
#include <stdio.h>

#include "usb_driver.h"

unsigned char transfer_type;
unsigned char transfer_index;
unsigned char transfer_size;
unsigned char* transfer_data;

unsigned char send_zerolength_packet;
```

```
    unsigned char receive_zerolength_packet;

    unsigned char in_stream[256];
    int in_stream_size;
    unsigned char out_stream[256];
    int in_stream_index;

    device_descriptor_t dev_desc;
    configuration_descriptor_t config_desc;
    interface_descriptor_t interface_desc;
    endpoint_descriptor_t endp_desc[2];

    unsigned char config_data[256];

    unsigned char request_buffer[256]; // check size
    int request_buffer_size;
    unsigned char receiving_request;
    int request_size;

    unsigned char reply_buffer[256];
    int reply_size;


    unsigned char data01;

    // 09 02 22 00 01 01 00 A0 32 - Config
    // 09 04 00 00 01 03 01 02 00 - Interface
    // 09 21 00 01 00 01 22 48 00 - Ekstra
    // 07 05 81 03 04 00 0A       - Endpoint

    void usb_init() {        // Initialization of the AVR
            int index;
            int i2;
            setup_data_t setup;

            in_stream_index = 0;

            send_zerolength_packet = 0;
            receive_zerolength_packet = 0;

            receiving_request = 0;
            request_buffer_size = 0;

            reply_size = 0;

            //DDRC  = 0xFF;                 // Set PORTC as output
            //DDRD  = 0x00;                 // Set PORTD as input
            //TCCR0         = 0x05;                 // Count clock/1024.

            dev_desc.bLength = 0x12;
            dev_desc.bDescriptorType = 0x01;
            dev_desc.bcdUSB = 0x0110; // USB 1.1
            dev_desc.bDeviceClass = 0x00;
            dev_desc.bDeviceSubClass = 0x00;
            dev_desc.bDeviceProtocol = 0x00;
            dev_desc.bMaxPacketSize0 = 0x08;
            dev_desc.idVendor = 0x1337;
            dev_desc.idProduct = 0xBABE;
            dev_desc.bcdDevice = 0x00;
            dev_desc.iManufacturer = 0x00;
            dev_desc.iProduct = 0x00;
            dev_desc.iSerialNumber = 0x00;
            dev_desc.bNumConfigurations = 0x01;

            config_desc.bLength = 0x09;
            config_desc.bDescriptorType = 0x02;
            config_desc.wTotalLength = sizeof(configuration_descriptor_t) + sizeof(interface_descriptor_t);
                                                                    //sizeof(endpoint_descriptor_t)*2;
            config_desc.bNumInterfaces = 0x01;
            config_desc.bConfigurationValue = 0x01;
            config_desc.iConfiguration = 0x00;
            config_desc.bmAttributes = 0xC0; // bit 7 Reserved(set to 1), bit 6 Self-Powered
            config_desc.bMaxPower = 0x00; // in 2mA units

            interface_desc.bLength = 0x09;
            interface_desc.bDescriptorType = 0x04;
            interface_desc.bInterfaceNumber = 0x00;
            interface_desc.bAlternateSetting = 0x00;
            interface_desc.bNumEndpoints = 0x00;
            interface_desc.bInterfaceClass = 0x03; // should be FF
            interface_desc.bInterfaceSubClass = 0x01; // should be FF
            interface_desc.bInterfaceProtocol = 0xFF;
            interface_desc.iInterface = 0x00;
    /*
            endp_desc[0].bLength = 0x07;
            endp_desc[0].bDescriptorType = 0x05;
            endp_desc[0].bEndpointAddress = 0x81; // IN endpoint
```

```
        endp_desc[0].bmAttributes = 0x03; // Interrupt
105     endp_desc[0].wMaxPacketSize = 0x0008;
        endp_desc[0].bInterval = 0x0A; // Polling interval in ms

        endp_desc[1].bLength = 0x07;
        endp_desc[1].bDescriptorType = 0x05;
110     endp_desc[1].bEndpointAddress = 0x01; // OUT endpoint
        endp_desc[1].bmAttributes = 0x03; // Interrupt
        endp_desc[1].wMaxPacketSize = 0x0008;
        endp_desc[1].bInterval = 0x0A; // Polling interval in ms
    */
115     index = 0;
        memcpy(&config_data[index], &config_desc, sizeof(config_desc));
        index += sizeof(config_desc);

        memcpy(&config_data[index], &interface_desc, sizeof(interface_desc));
120     index += sizeof(interface_desc);
    /*
        memcpy(&config_data[index], &endp_desc[0], sizeof(endp_desc));
        index += sizeof(endp_desc);

125     memcpy(&config_data[index], &endp_desc[1], sizeof(endp_desc));
        index += sizeof(endp_desc);
    */

        i2 = 0;
130 /*
        // Set Address
        in_stream[i2++] = 0x09; // size
        in_stream[i2++] = 0x80; // Setup Packet
        setup.bmRequestType = 0x00;
135     setup.bRequest = SET_ADDRESS;
        setup.wValue = 0x000F;
        setup.wIndex = 0x0000;
        setup.wLength = 0x00;
        memcpy(&in_stream[i2], &setup, sizeof(setup));
140     i2 += sizeof(setup);

        // Device Descriptor
        in_stream[i2++] = 0x09; // size
        in_stream[i2++] = 0x80; // Setup Packet
145     setup.bmRequestType = 0x80;
        setup.bRequest = GET_DESCRIPTOR;
        setup.wValue = 0x0100; // Device
        setup.wIndex = 0x0000;
        setup.wLength = 0x12;
150     memcpy(&in_stream[i2], &setup, sizeof(setup));
        i2 += sizeof(setup);

        in_stream[i2++] = 0x01; // size
        in_stream[i2++] = 0x00; // Out Packet
155
        // Configuration Descriptor
        in_stream[i2++] = 0x09; // size
        in_stream[i2++] = 0x80; // Setup Packet
        setup.bmRequestType = 0x80;
160     setup.bRequest = GET_DESCRIPTOR;
        setup.wValue = 0x0200; // CONFIGURATION Descriptor Type
        setup.wIndex = 0x0000;
        setup.wLength = 0x09;
        memcpy(&in_stream[i2], &setup, sizeof(setup));
165     i2 += sizeof(setup);

        in_stream[i2++] = 0x01; // size
        in_stream[i2++] = 0x00; // Out Packet
    */
170
        in_stream[i2++] = 0x09; // size
        in_stream[i2++] = 0x80; // Setup Packet
        setup.bmRequestType = 0x40; // Vendor
        setup.bRequest = PREPARE_RESPONSE;
175     setup.wValue = 0x0100; // Device
        setup.wIndex = 0x0000;
        setup.wLength = 0x01;
        memcpy(&in_stream[i2], &setup, sizeof(setup));
        i2 += sizeof(setup);
180
        in_stream[i2++] = 0x02; // size
        in_stream[i2++] = 0x00;
        in_stream[i2++] = GET_TIME; // Request

185

        in_stream[i2++] = 0x09; // size
        in_stream[i2++] = 0x80; // Setup Packet
        setup.bmRequestType = 0xC0; // Device-to-Host, Vendor
        setup.bRequest = GET_RESPONSE;
190     setup.wValue = 0x0100; // Device
        setup.wIndex = 0x0000;
```

```
        setup.wLength = 0x08;
        memcpy(&in_stream[i2], &setup, sizeof(setup));
        i2 += sizeof(setup);

        in_stream[i2++] = 0x01; // size
        in_stream[i2++] = 0x00; // Out Packet

}

void delay(unsigned char multiples)     {       // Producing a delay in mult. of 65 ms at 4 MHz
        unsigned char i;

        for (i=0;i<multiples;i++) {
                //while (!(TIFR&0x02)); // Waiting for timer0 overflow flag to be set
                //TIFR = 0x02;                  // Clearing overflow flag
        }
}

void usb_send_byte(unsigned char byte) {
        //PORTC = ~byte;
        //delay(10);
        //PORTC = 0xFF;
        printf("sending: %X\n", byte);
}


unsigned char usb_receive_byte() {
        printf("receiving: %X\n", in_stream[in_stream_index]);
        return in_stream[in_stream_index++];
}


void interrupt_loop() {
        int last_type = transfer_type;
        while (1) {
                char my_string[10];
                printf("waiting for interrupt...");
                gets(my_string);
                if (my_string[0] == 'q') {
                        break;
                }
                SIG_INTERRUPT0();
                if (last_type == SEND_TRANSFER && transfer_type == NO_TRANSFER) {
                        printf("send transfer completed.\n");
                }
                last_type = transfer_type;
        }
}



int main (void) {
        int i;
        usb_init();
        transfer_type = NO_TRANSFER;
        in_stream_size = 0;

        /*
        GIMSK |= 0x40;
        MCUCR |= 0x03;
        SREG |= 0x80;

        PORTC = 0xFF;
        */
        //printf("%i\n", sizeof(configuration_descriptor_t));
        //printf("%i\n", sizeof(interface_descriptor_t));
        //printf("%i\n", sizeof(endpoint_descriptor_t));

        for (i = 0; i < 11; i++)
                out_stream[i] = i+1;


        //for (i = 0; i < 11; i++)
        //      hub_stream[i] = i+1;


        //usb_start_send_transfer(out_stream, 11);

        // get device descriptor
        interrupt_loop();
        // send device descriptor
        //interrupt_loop();


        //interrupt_loop();
```

```
280
        //PORTC = 0xF0;

        return 1;
}
285

// Sends data to the USB Controller. This function splits the data into 8 byte
// packets and sends each in turn.
// It is important to note that the host must expect this
290 void usb_start_send_transfer(unsigned char *data, unsigned char size) {
        data01 = 1;
        transfer_size = size;
        transfer_data = data;
        transfer_index = 0;
295     transfer_type = SEND_TRANSFER;
        if (size > 8) {
                usb_send_packet(transfer_data, 8, data01);
                transfer_index += 8;
        } else {
300             usb_send_packet(transfer_data, size, data01);
                transfer_index += size;
        }
}

305 void usb_continue_send_transfer() {
        unsigned char rest = transfer_size - transfer_index;
        // toggle data0/1
        if (data01) data01 = 0;
        else data01 = 1;
310
        if (rest > 0) {
                if (rest > 8) {
                        usb_send_packet(&transfer_data[transfer_index], 8, data01);
                        transfer_index += 8;
315             } else {
                        usb_send_packet(&transfer_data[transfer_index], rest, data01);
                        transfer_index += rest;
                }
        }
320     else {
                transfer_type = NO_TRANSFER; // we are done
                if (send_zerolength_packet) { // Zero-Length data packet sent
                        printf("Write or No-Data Control Transfer complete. (Zero Length Packet sent)\n");
                        send_zerolength_packet = 0;
325             }
        }
}


330 // Sends a data packet to the USB Controller
    // size can be max 8 bytes
    // data1 is 1 if it is a data1 packet
    void usb_send_packet(unsigned char *packet_data, unsigned char size, unsigned char data1) {
        unsigned char i;
335     usb_send_byte(WRITE_BUFFER);
        usb_send_byte(size | (data1 << 7));
        for (i = 0; i < size; i++) {
                usb_send_byte(packet_data[i]);
        }
340 }


    unsigned char usb_read_int_reg() {
        printf("Reading interrupt register\n");
345     //usb_send_byte(READ_INT_REG);
        return 0xF0; //usb_receive_byte();
    }

    void usb_set_address(unsigned char addr) {
350     printf("Setting address: %i\n", addr);
        usb_send_byte(SET_DEVICE_ADDRESS);
        usb_send_byte(addr);
    }

355
    void handle_request() {
        int i;
        printf("Request received\n");
        reply_size = 0;
360     switch (request_buffer[0]) {
                case GET_TIME:
                        for (i = 0; i < 8; i++) {
                                reply_buffer[i] = i;
                        }
365                     reply_size = 8;
                        break;
        }
```

```
}

void usb_get_packet() {
        unsigned char flags;
        unsigned char packet[8];
        int i;
        unsigned char size;

        usb_send_byte(READ_BUFFER);
        size = usb_receive_byte();
        flags = usb_receive_byte();

        if ((flags & 0x0F) == 0) { // control endpoint
                if (flags & 0x80) { // Setup Packet (always 8 bytes)
                        for (i = 0; i < 8; i++) {
                                packet[i] = usb_receive_byte();
                        }
                        parse_setup_data((setup_data_t*)packet);
                } else { // Out
                        if (receiving_request) {
                                for (i = 0; i < size-1; i++) {
                                        request_buffer[request_buffer_size] = usb_receive_byte();
                                        request_buffer_size++;
                                }

                                if (request_buffer_size == request_size) { // we have received the whole request
                                        handle_request();
                                        receiving_request = 0;
                                        // Send Zero-Length data packet
                                        usb_start_send_transfer(0, 0);
                                        send_zerolength_packet = 1;
                                }
                        }

                        if (receive_zerolength_packet) { // Status Stage of Control transfer
                                // Get the packet, it should be zero length
                                for (i = 0; i < size-1; i++) {
                                        packet[i] = usb_receive_byte();
                                }
                                receive_zerolength_packet = 0; // End of Control Transfer
                                printf("Read Control Transfer complete. (Zero Length Packet)\n");
                        }
                }
        }
}

void parse_setup_data(setup_data_t *setup) {
        if (setup->bmRequestType & 0x80) { // Device-to-host
                data01 = 1;
                if ((setup->bmRequestType & 0x60) == 0) { // Type = Standard
                        switch (setup->bmRequestType & 0x1F) { // Recipient
                                case RECIPIENT_DEVICE:
                                        if (setup->bRequest == GET_DESCRIPTOR) {
                                                int descriptor_type = setup->wValue >> 8;
                                                switch (descriptor_type) {
                                                        case DEVICE_DESCRIPTOR:
                                                                usb_start_send_transfer((unsigned char*)&dev_desc, setup->wLength); // should alwa
                                                                receive_zerolength_packet = 1;
                                                                break;

                                                        case CONFIGURATION_DESCRIPTOR:
                                                                usb_start_send_transfer(config_data, setup->wLength);
                                                                receive_zerolength_packet = 1;
                                                                break;

                                                        case STRING_DESCRIPTOR:
                                                                break;
                                                }
                                        }
                                        break;

                                case RECIPIENT_INTERFACE:
                                        break;

                                case RECIPIENT_ENDPOINT:
                                        break;
                        }
                } else if ((setup->bmRequestType & 0x60) == 0x40) { // Type = Vendor (Us)
                        switch (setup->bmRequestType & 0x1F) { // Recipient
                                case RECIPIENT_DEVICE:
                                        if (setup->bRequest == GET_RESPONSE) {
                                                usb_start_send_transfer(reply_buffer, reply_size);
                                                receive_zerolength_packet = 1;
                                        }
                                        break;
                        }
                }
        } else {
```

```
                // Host-to-device
                if ((setup->bmRequestType & 0x60) == 0) { // Type = Standard
                        switch (setup->bmRequestType & 0x1F) { // Recipient
                                case RECIPIENT_DEVICE:
                                        if (setup->bRequest == SET_ADDRESS) {
                                                usb_set_address((unsigned char)setup->wValue);
                                                // Send Zero-Length data packet
                                                usb_start_send_transfer(0, 0);
                                                send_zerolength_packet = 1;
                                        } else if (setup->bRequest == SET_CONFIGURATION) {
                                                // Should select the configuration here..we just reply that everyting is ok
                                                // Send Zero-Length data packet
                                                usb_start_send_transfer(0, 0);
                                                send_zerolength_packet = 1;
                                        }
                                        break;
                        }
                } else if ((setup->bmRequestType & 0x60) == 0x40) { // Type = Vendor (Us)
                        switch (setup->bmRequestType & 0x1F) { // Recipient
                                case RECIPIENT_DEVICE:
                                        if (setup->bRequest == PREPARE_RESPONSE) {
                                                receiving_request = 1;
                                                request_buffer_size = 0;
                                                request_size = setup->wLength;
                                        }
                                        break;
                        }
                }
        }

void SIG_INTERRUPT0() {
        /*
        PORTC = 0;
        delay(5);
        PORTC = 0xFF;
        */
        if (transfer_type == SEND_TRANSFER) {
                usb_continue_send_transfer();
        } else { //else if (transfer_type == RECEIVE_TRANSFER) {
                unsigned char int_reg = usb_read_int_reg();
                if (int_reg & PACKET_RECEIVED_MASK)
                        usb_get_packet();
        }

}
```

Listing 21: Sensorkort/usb_driver.h

```
#define SET_ADDRESS                     0x05
#define GET_DESCRIPTOR          0x06
#define GET_CONFIGURATION       0x08
#define SET_CONFIGURATION       0x09

// Vendor Specific
#define PREPARE_RESPONSE        0xF0
#define GET_RESPONSE            0xF1

#define GET_TIME                                0x06
#define SET_TIME                                0x08
#define SET_SAMPLE_INTERVAL             0x10
#define GET_SAMPLE_INTERVAL             0x11
#define FLUSH_ALL                               0x07
#define GET_SAMPLE                              0x05

#define DEVICE_DESCRIPTOR                       0x01
#define CONFIGURATION_DESCRIPTOR        0x02
#define STRING_DESCRIPTOR                       0x03
#define INTERFACE_DESCRIPTOR            0x04
#define ENDPOINT_DESCRIPTOR                     0x05

#define SET_DEVICE_ADDRESS      0x00
#define READ_INT_REG            0x01
#define WRITE_BUFFER            0x02
#define READ_BUFFER             0x03

#define NO_TRANSFER                     0x00
#define SEND_TRANSFER           0x01
#define RECEIVE_TRANSFER        0x02

#define RECIPIENT_DEVICE        0x00
#define RECIPIENT_INTERFACE     0x01
#define RECIPIENT_ENDPOINT      0x02

#define PACKET_RECEIVED_MASK    0x40
```

```
#define PACKET_SENT_MASK                0x20

#pragma pack(1) // should be removed

typedef unsigned short avr_word;

typedef struct {
        unsigned char bmRequestType;
        unsigned char bRequest;
        avr_word wValue;
        avr_word wIndex;
        avr_word wLength;
} setup_data_t;


typedef struct {
        unsigned char bLength;
        unsigned char bDescriptorType;
        avr_word bcdUSB;
        unsigned char bDeviceClass;
        unsigned char bDeviceSubClass;
        unsigned char bDeviceProtocol;
        unsigned char bMaxPacketSize0;
        avr_word idVendor;
        avr_word idProduct;
        unsigned char bcdDevice;
        unsigned char iManufacturer;
        unsigned char iProduct;
        unsigned char iSerialNumber;
        unsigned char bNumConfigurations;
} device_descriptor_t;


typedef struct {
        unsigned char bLength;
        unsigned char bDescriptorType;
        avr_word wTotalLength;
        unsigned char bNumInterfaces;
        unsigned char bConfigurationValue;
        unsigned char iConfiguration;
        unsigned char bmAttributes;
        unsigned char bMaxPower;
} configuration_descriptor_t;

typedef struct {
        unsigned char bLength;
        unsigned char bDescriptorType;
        unsigned char bInterfaceNumber;
        unsigned char bAlternateSetting;
        unsigned char bNumEndpoints;
        unsigned char bInterfaceClass;
        unsigned char bInterfaceSubClass;
        unsigned char bInterfaceProtocol;
        unsigned char iInterface;
} interface_descriptor_t;


typedef struct {
        unsigned char bLength;
        unsigned char bDescriptorType;
        unsigned char bEndpointAddress;
        unsigned char bmAttributes;
        avr_word wMaxPacketSize;
        unsigned char bInterval;

} endpoint_descriptor_t;

void SIG_INTERRUPT0(); // __attribute__ ((interrupt));

int usb_transfer_completed();
void usb_start_send_transfer(unsigned char *data, unsigned char size);
void usb_continue_transfer();
void usb_send_packet(unsigned char *packet_data, unsigned char size, unsigned char data1);
void send_byte(unsigned char byte);
void parse_setup_data(setup_data_t *setup);

/*
09 02 22 00 01 01 00 A0 32 - Configuration Descriptor
09 04 00 00 01 03 01 02 00 - Interface 0 Descriptor
09 21 00 01 00 01 22 48 00 - Class spcific descriptor
07 05 81 03 04 00 0A       - Endpoint Descriptor
*/
```

Listing 22: Sensorkort/avr_interface.c

```
/*

        avr_interface.c

        Funksjoner for kommunikasjon mellom AVR og FPGA

        Benytter 4 pinner:

                - bit_in
                - bit_out
                - ds_in
                - ds_out

        2003 (c) dmprosj

*/
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/signal.h>

#define BIT_IN   0x01
#define BIT_IN_POS 1
#define DS_IN    0x02

#define BIT_OUT 0x04
#define BIT_OUT_POS 2
#define DS_OUT   0x08

#define INPORT PIND
#define OUTPORT PORTD

void initialization(void)       // Initialization of the AVR
{
        DDRC    = 0xFF;                 // Set PORTC as output
        DDRD    = 0xFC;                 // Set PORTD as input & output

        TCCR0   = 0x05;                 // Count clock/1024.

}

void delay(unsigned char multiples)             // Producing a delay in mult. of 65 ms at 4 MHz
{
        unsigned char i;

        for (i=0;i<multiples;i++)
        {
                while (!(TIFR&0x02));   // Waiting for timer0 overflow flag to be set
                TIFR = 0x02;                    // Clearing overflow flag
        }
}

void send_byte(unsigned char byte) {

        OUTPORT = (0x00);

        unsigned char i;
        unsigned char bit;
        for (i=0; i<8; i++) {

                // mask out & shift LSB bit
                bit = (byte & 0x01) << BIT_OUT_POS;

                // output one bit & strobe data
                OUTPORT = (bit | DS_OUT);

                // wait for ack to be set (data strobe in)
                while ((INPORT & DS_IN) == 0x00) {
                }

                // wait for ack to be cleared (data strobe in)
                while ((INPORT & DS_IN) != 0x00) {
                }

                // clear data strobe out
                OUTPORT = (bit);
                delay(2);

                // shift byte
                byte >>= 1;
        }
        OUTPORT = (0x00);
}

unsigned char receive_byte() {
```

```
        PORTC = (0xFF);

        unsigned char i;
        unsigned char byte = 0x00;
        unsigned char bit = 0x00;

        for (i=0; i<8; i++) {

                // wait for ack to be set (data strobe in)
                while ((INPORT & DS_IN) == 0x00) {
                }

                // output ack
                OUTPORT = (DS_OUT);

                // shift byte
                byte >>= 1;

                // mask out & shift LSB bit
                bit = (INPORT & BIT_IN) << (8 - BIT_IN_POS);
                byte |= bit;

                delay(2);

                // clear ack
                OUTPORT = (0x00);

                // wait for ack to be cleared (data strobe in)
                while ((INPORT & DS_IN) != 0x00) {
                }

                delay(2);
                PORTC = ~byte;

        }
        OUTPORT = (0x00);

        return byte;
}


int main (void)
{

        initialization();               // Initialize Pheripherals

        while (1) {
                PORTC = 0x00;
                delay(5);
                PORTC = 0xFF;
                delay(5);

                send_byte(0x0F);
                PORTC = ~(receive_byte());
        }

        return 1;
}
```

F.2 Sensorkort - FPGA

Listing 23: Sensorkort/avr_interface.vhd

```
-------------------------------
--      avr_interface.vhd
-------------------------------
--
--      VHDL-kode for kommunikasjon mellom AVR og FPGA
--
--      Benytter 4 pinner til kommunikasjon med AVR:
--
--              - bit_in
--              - bit_out
--              - ds_in
--              - ds_out
--
-------------------------------
--      2003 (c) dmprosj
-------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
```

```
entity avr_interface is
    port (
                        byte_in                                 : in std_logic_vector(7 downto 0);
                        byte_valid_in    : in std_logic;

                        byte_out                                : out std_logic_vector(7 downto 0);
                        byte_valid_out   : out std_logic;

                        bit_in   : in std_logic;
                        ds_in    : in std_logic;

                        bit_out : out std_logic;
                        ds_out   : out std_logic;

                        busy : out std_logic;

                        reset : in std_logic;
                        clk      : in std_logic
                );
end avr_interface;

architecture Behavioral of avr_interface is

        COMPONENT byte_receive
        PORT(
                bit_in : IN std_logic;
                ds_in : IN std_logic;
                reset : IN std_logic;
                clk : IN std_logic;
                enabled : IN std_logic;
                byte_out : OUT std_logic_vector(7 downto 0);
                byte_valid : OUT std_logic;
                ds_out : OUT std_logic;
                busy : OUT std_logic
                );
        END COMPONENT;

        COMPONENT byte_transmit
        PORT(
                bit_out : out std_logic;
                ds_in : IN std_logic;
                reset : IN std_logic;
                clk : IN std_logic;
                enabled : IN std_logic;
                byte_in : in std_logic_vector(7 downto 0);
                byte_valid : in std_logic;
                ds_out : OUT std_logic;
                busy : OUT std_logic
                );
        END COMPONENT;

        signal tx_busy : std_logic;
        signal tx_enabled : std_logic;
        signal tx_ds_out : std_logic;

        signal rx_enabled : std_logic;
        signal rx_busy : std_logic;
        signal rx_ds_out : std_logic;

begin

        rx_unit: byte_receive PORT MAP(
                byte_out => byte_out,
                byte_valid => byte_valid_out,
                bit_in => bit_in,
                ds_in => ds_in,
                ds_out => rx_ds_out,
                reset => reset,
                clk => clk,
                enabled => rx_enabled,
                busy => rx_busy
        );

        tx_unit: byte_transmit PORT MAP(
                byte_in => byte_in,
                byte_valid => byte_valid_in,
                bit_out => bit_out,
                ds_in => ds_in,
                ds_out => tx_ds_out,
                busy => tx_busy,
                reset => reset,
                clk => clk,
                enabled => tx_enabled
        );

        rx_enabled <= not(tx_busy);
```

```
        tx_enabled <= '1';
        busy <= tx_busy;
        with tx_busy select
                ds_out <= tx_ds_out when '1',
                                                rx_ds_out when others;

end Behavioral;
```

Listing 24: Sensorkort/controller.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity usb_controller is
        port (
                rcv: in STD_LOGIC;
                vpo : out std_logic;
                vmo : out std_logic;

                vp : in std_logic;
                vm : in std_logic;
                eop : out std_logic;

                current_state : out std_logic_vector(4 downto 0); -- remove
                out_clk : out std_logic; --remove

                --data_received : out std_logic_vector(7 downto 0);
                data_to_send : in std_logic_vector(7 downto 0);

                device_address : in std_logic_vector(6 downto 0);

                transmit_data : out std_logic_vector(7 downto 0);

                -- IN BUFFER (write inncomming packets)
                in_w_address : out std_logic_vector(3 downto 0);
                in_w_data : out std_logic_vector(7 downto 0);
                in_w_enable : out std_logic;
                in_validate : out std_logic;

                -- OUT BUFFER (reads outgoing packets)
                --out_address : out std_logic_vector(3 downto 0);
                --out_data : in std_logic_vector(7 downto 0);
                --out_clear : out std_logic;
                --out_full : in std_logic;


                fast_clk: in STD_LOGIC;
                rst: in STD_LOGIC
        );
end usb_controller;

architecture usb_controller of usb_controller is

        component transmitter
                port (
                        rst: in STD_LOGIC;
                        clk: in STD_LOGIC;
                        input: in STD_LOGIC_VECTOR (7 downto 0);
                        input_select: in STD_LOGIC_VECTOR(2 downto 0);
                        pid_select: in STD_LOGIC_VECTOR(1 downto 0);
                        enable: in STD_LOGIC;
                        send_eop: in STD_LOGIC;
                        eop_sent : out std_logic;
                        vpo : out std_logic;
                        vmo : out std_logic;
                        out_byte_sent: out STD_LOGIC
                );
        end component;


        component receiver
                port (
                        input: in STD_LOGIC;
                        vp : in std_logic;
                        vm : in std_logic;
                        enable_rec: in STD_LOGIC;
                        quick_enable: in STD_LOGIC;
                        output: out STD_LOGIC_VECTOR(7 downto 0);
                        byte_received: out STD_LOGIC;
                        sop_detect: out std_logic;
                        eop_detect: out std_logic;
                        crc5_ok : out std_logic;
                        crc16_ok : out std_logic;
```

```
                calc_crc5 : in std_logic;
                calc_crc16 : in std_logic;
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                clk_out: out STD_LOGIC
        );
    end component;

    --Signals for quick_enable
    signal quick_enable: STD_LOGIC;
    signal mult_select: STD_LOGIC_VECTOR (2 downto 0);
    signal enable_sender: STD_LOGIC;
    signal enable_receiver: STD_LOGIC;

    signal pid_to_send: STD_LOGIC_VECTOR (1 downto 0);
    signal send_eop: STD_LOGIC;
    signal eop_sent : std_logic;
    signal out_byte_sent: STD_LOGIC;
    signal byte_received: STD_LOGIC;

    --constant out_pid : std_logic_vector(3 downto 0)       :=  "1000"; --"0001";
    --constant in_pid : std_logic_vector(3 downto 0)                :="1001";
    --constant setup_pid : std_logic_vector(3 downto 0) :="1101";
    --constant data0_pid : std_logic_vector(3 downto 0) :="0011";
    --constant data1_pid : std_logic_vector(3 downto 0) :="1011";
    --constant ack_pid : std_logic_vector(3 downto 0)       :="0010";
    --constant nack_pid : std_logic_vector(3 downto 0)      :="1010";
    --constant stall_pid : std_logic_vector(3 downto 0) :="1110";

    type fifo_type is array (0 to 1) of std_logic_vector(7 downto 0);
    signal fifo : fifo_type;

    type state_type is (
                -- Token states
                wait_token_sop, wait_token_pid, wait_token_byte_1, wait_token_byte_2, wait_token_eop,

                -- OUT and SETUP states
                wait_data_sop, wait_data_byte, wait_data_pid,
                sending_ack_sop, sending_ack_pid, sending_ack_eop,

                -- IN states
                sending_data_sop, sending_data_pid, sending_data_byte, sending_crc_byte0, sending_crc_byte1,
                sending_data_eop, wait_ack_sop, wait_ack_pid, wait_ack_eop
        );

    signal state : state_type;


    type transaction_type is (no_transaction, in_transaction, out_transaction, setup_transaction);
    signal current_transaction : transaction_type;

    ------------------------------------------------------
    -- Signals for controller that were originally ports.
    ------------------------------------------------------
    signal sop_detect: std_logic;
    signal eop_detect: std_logic;
    signal crc5_ok : std_logic;
    signal crc16_ok : std_logic;

    signal calc_crc5 : std_logic;
    signal calc_crc16 : std_logic;

    -- The clock from the PPL
    signal clk: STD_LOGIC;

    signal receiver_data : std_logic_vector(7 downto 0);
    signal transmitter_data : std_logic_vector(7 downto 0);

    signal sender_mult_select : std_logic_vector(2 downto 0);

    signal address_in_token : std_logic_vector(6 downto 0);
    signal endpoint_in_token : std_logic_vector(3 downto 0);

    constant SETUP_PID : std_logic_vector(3 downto 0) := "1011";
    constant OUT_PID : std_logic_vector(3 downto 0) := "1000";
    constant IN_PID : std_logic_vector(3 downto 0) := "1001";

    signal write_address : std_logic_vector(3 downto 0);

    begin

        in_w_address <= write_address;

        transmit_data <= transmitter_data;

        out_clk <= clk;
        eop <= eop_detect;
```

```
165             transmitter1 : transmitter
                        port map (rst,
                                                        clk,
                                                        transmitter_data,
                                                        sender_mult_select,
170                                                     pid_to_send,
                                                        enable_sender,
                                                        send_eop,
                                                        eop_sent,
                                                        vpo,
175                                                     vmo,
                                                        out_byte_sent);

                receiver1 : receiver
                        port map (rcv,
180                                                     vp,
                                                        vm,
                                                        enable_receiver,
                                                        quick_enable,
                                                        receiver_data,
185                                                     byte_received,
                                                        sop_detect,
                                                        eop_detect,
                                                        crc5_ok,
                                                        crc16_ok,
190                                                     calc_crc5,
                                                        calc_crc16,
                                                        rst,
                                                        fast_clk,
                                                        clk);
195
                state_machine :
                process (clk, rst)
                        variable bit_time_count : integer;
                        --variable last_data : std_logic; -- DATA0 or DATA1
200                     variable token_pid : STD_LOGIC_VECTOR(3 downto 0);
                        variable token_pid_inv : std_logic_vector(3 downto 0);
                        variable inc_addr : boolean;
                begin
                        if rst = '1' then
205                             state <= wait_token_sop;
                                enable_receiver <= '0';
                                enable_sender <= '0';
                                pid_to_send <= "00";
                                bit_time_count := 0;
210                             --last_data := '0';
                                send_eop <= '0';
                                current_transaction <= no_transaction;
                                write_address <= "0010";
                                in_w_data <= (others => '0');
215                             in_w_enable <= '0';
                                in_validate <= '0';
                                inc_addr := false;

                        elsif rising_edge(clk) then
220                                 case current_transaction is

                                        -- waiting for SETUP, OUT or IN token
                                        when no_transaction =>
                                                case state is
225                                                     when wait_token_sop =>
                                                                current_state <= "00000";
                                                                enable_receiver <= '0';
                                                                enable_sender <= '0';
                                                                send_eop <= '0';
230                                                             write_address <= "0010";

                                                                if sop_detect = '1' then
                                                                        state <= wait_token_pid;
                                                                        enable_receiver <= '1';
235                                                             end if;


                                                        when wait_token_pid =>
                                                                current_state <= "00001";
240                                                             if byte_received = '1' then
                                                                        token_pid := receiver_data(7 downto 4);
                                                                        token_pid_inv := receiver_data(3 downto 0);
                                                                        if token_pid = not(token_pid_inv) and
                                                                                (token_pid = SETUP_PID or token_pid = OUT_PID or token_pid
245                                                                             state <= wait_token_byte_1;
                                                                        else
                                                                                state <= wait_token_sop; -- error
                                                                        end if;
                                                                end if;
250
                                                        when wait_token_byte_1 =>
                                                                current_state <= "00010";
```

```
                        if byte_received = '1' then
                                address_in_token(0) <= receiver_data(7);
                                address_in_token(1) <= receiver_data(6);
                                address_in_token(2) <= receiver_data(5);
                                address_in_token(3) <= receiver_data(4);
                                address_in_token(4) <= receiver_data(3);
                                address_in_token(5) <= receiver_data(2);
                                address_in_token(6) <= receiver_data(1);
                                endpoint_in_token(0) <= receiver_data(0);
                                state <= wait_token_byte_2;
                        end if;

                when wait_token_byte_2 =>
                        current_state <= "00011";
                        if byte_received = '1' then
                                endpoint_in_token(1) <= receiver_data(7);
                                endpoint_in_token(2) <= receiver_data(6);
                                endpoint_in_token(3) <= receiver_data(5);
                                bit_time_count := 0;
                                state <= wait_token_eop;
                        end if;

                when wait_token_eop =>
                        current_state <= "00100";
                        if eop_detect = '1' then
                                --if crc5_ok = '1'
                                if address_in_token = device_address then
                                        case token_pid is
                                                when SETUP_PID =>
                                                        state <= wait_data_sop;
                                                        enable_receiver <= '0';
                                                        current_transaction <= setup_transaction;

                                                when OUT_PID =>
                                                        state <= wait_data_sop;
                                                        enable_receiver <= '0';
                                                        current_transaction <= out_transaction;

                                                when IN_PID =>
                                                        state <= sending_data_sop;
                                                        sender_mult_select <= "111";
                                                        enable_receiver <= '0';
                                                        enable_sender <= '1';
                                                        current_transaction <= in_transaction;

                                                when others =>
                                                        state <= wait_token_sop;

                                        end case;
                                else
                                        state <= wait_token_sop;
                                end if;

                                --else
                                --      state <= wait_token_sop; -- invalid packet!
                                --end if;
                        --else
                        --      bit_time_count := bit_time_count + 1;
                        --      if bit_time_count > 5 then -- maybe change this number
                        --              state <= wait_token_sop;        -- invalid packet!!
                        --      end if;
                        --end if;
                        end if;

                when others =>
                        null;

        end case;


        ------------------------------------------------------------------

        -- OUT token received..move on!
        when out_transaction | setup_transaction =>

                case state is
                        -- timeout ??
                        when wait_data_sop =>
                                current_state <= "00101";
                                if sop_detect = '1' then
                                        state <= wait_data_pid;
                                        enable_receiver <= '1';
                                end if;

                        when wait_data_pid =>
                                current_state <= "00110";
```

```
                        if byte_received = '1' then
                                -- check pid for data0 or data1
                                in_w_enable <= '1';
                                state <= wait_data_byte;
                        end if;

                -- timeout?
                when wait_data_byte =>
                        current_state <= "00111";
                        if eop_detect = '1' then
                                --if crc16_ok = '1' then
                                        state <= sending_ack_sop;
                                        enable_sender <= '1';
                                        enable_receiver <= '0';
                                        sender_mult_select <= "111"; -- sync
                                        in_w_enable <= '0';
                                --else
                                --      state <= wait_token_sop; -- oops .. clear the buffer too!
                                --      current_transaction <= no_transaction;
                                --end if;
                        end if;

                        if byte_received = '1' then
                                in_w_data(7) <= receiver_data(0);
                                in_w_data(6) <= receiver_data(1);
                                in_w_data(5) <= receiver_data(2);
                                in_w_data(4) <= receiver_data(3);
                                in_w_data(3) <= receiver_data(4);
                                in_w_data(2) <= receiver_data(5);
                                in_w_data(1) <= receiver_data(6);
                                in_w_data(0) <= receiver_data(7);
                                inc_addr := true;
                        else
                                if inc_addr = true and write_address < 9 then
                                        write_address <= write_address + 1;
                                        inc_addr := false;
                                end if;
                        end if;

                when sending_ack_sop =>
                        current_state <= "01000";
                        if out_byte_sent = '1' then -- sync sent
                                sender_mult_select <= "000"; -- pid
                                pid_to_send <= "00"; -- ack
                                state <= sending_ack_pid;
                        end if;


                when sending_ack_pid =>
                        current_state <= "01001";
                        if out_byte_sent = '1' then
                                send_eop <= '1';
                                state <= sending_ack_eop;
                        end if;

                when sending_ack_eop =>
                        current_state <= "01010";
                        if eop_sent = '1' then
                                send_eop <= '0';
                                enable_sender <= '0';
                                current_transaction <= no_transaction;
                                state <= wait_token_sop;
                        end if;


                when others =>
                        null;

        end case;

-------------------------------------------------------------------

when in_transaction =>
        case state is
                when sending_data_sop =>
                        current_state <= "01011";
                        if out_byte_sent = '1' then
                                state <= sending_data_pid;
                                sender_mult_select <= "000";
                                pid_to_send <= "11"; -- DATA0 .. must check here..
                        end if;

                when sending_data_pid =>
                        current_state <= "01100";
                        if out_byte_sent = '1' then
                                state <= sending_data_byte;
                                transmitter_data <= "00000000"; -- first byte to send if any!
```

```
                                                        sender_mult_select <= "011";
                                                end if;

                                        when sending_data_byte =>
                                                current_state <= "01101";
                                                if out_byte_sent = '1' then
                                                        sender_mult_select <= "100"; -- send first crc
                                                        state <= sending_crc_byte0;
                                                end if;

                                        when sending_crc_byte0 =>
                                                current_state <= "01110";
                                                if out_byte_sent = '1' then
                                                        sender_mult_select <= "101"; -- send second crc
                                                        state <= sending_crc_byte1;
                                                end if;

                                        when sending_crc_byte1 =>
                                                current_state <= "01111";
                                                if out_byte_sent = '1' then
                                                        send_eop <= '1';
                                                        state <= sending_data_eop;
                                                end if;

                                        when sending_data_eop =>
                                                current_state <= "10000";
                                                if eop_sent = '1' then
                                                        send_eop <= '0';
                                                        enable_receiver <= '0';
                                                        enable_sender <= '0';
                                                        state <= wait_ack_sop;
                                                end if;

                                        -- put in a timeout here..
                                        when wait_ack_sop =>
                                                current_state <= "10001";
                                                if sop_detect = '1' then
                                                        enable_receiver <= '1';
                                                        state <= wait_ack_pid;
                                                end if;

                                        when wait_ack_pid =>
                                                current_state <= "10010";
                                                if byte_received = '1' then
                                                        -- check ack
                                                        state <= wait_ack_eop;
                                                end if;

                                        when wait_ack_eop =>
                                                current_state <= "10011";
                                                if eop_detect = '1' then
                                                        current_transaction <= no_transaction;
                                                        state <= wait_token_sop;
                                                end if;

                                        when others =>
                                                null;

                                end case;

                        end case;
                end if;
        end process;


--mult_select_assignment:
--mult_select <=        "000" when (state = wait_token_pid) else
--                                                      "001" when (state = wait_token_byte_1) else
--                                                      "010" when (state = wait_token_byte_2) else
--                                                      "111" when (state = wait_data_sop) else
--                                                      "000" when (state = wait_data_pid) else
--                                                      "100" when (state = crc_data) else
--                                                      "000" when (state = sending_ack_pid) else
--                                                      "011" when (state = data) else
--                                                      "111" when (state = sending_ack_sop) else
--                                                      "000";


quick_enable_process:
process(sop_detect)
begin
        if sop_detect = '1' and (state = wait_token_sop or state = wait_data_sop or state = wait_ack_sop) then
                quick_enable <= '1';
        else
                quick_enable <= '0';
        end if;
```

```
end process;
end usb_controller;
```

Listing 25: Sensorkort/byte_receive.vhd

```
--------------------------------
--      byte_receive.vhd
--------------------------------
--
--      VHDL-kode for mottaker-delen av kommunikasjonen mellom AVR og FPGA
--
--------------------------------
--      2003 (c) dmprosj
--------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity byte_receive is
    port (
                        byte_out : out std_logic_vector(7 downto 0);
                        byte_valid : out std_logic;

                        bit_in  : in std_logic;
                        ds_in           : in std_logic;
                        ds_out  : out std_logic;

                        busy            : out std_logic;

                        reset           : in std_logic;
                        clk                     : in std_logic;
                        enabled : in std_logic
                );
end byte_receive;

architecture Behavioral of byte_receive  is

        type state_type is (idle, rcv, ack);

        signal state, next_state : state_type;
        signal reg : std_logic_vector(7 downto 0);
        signal shift_reg_full : std_logic;
        signal done : std_logic;

begin

        busy <= not(shift_reg_full);

        -- state machine
        process (clk, reset)
                variable bit_no : integer := 0;

        begin

                if reset = '1' then
                        state <= idle;

                        bit_no := 0;
                        shift_reg_full <= '0';
                        reg <= "00000000";
                        byte_out <= "00000000";
                        ds_out <= '0';
                        done <= '1';
                        byte_valid <= '0';

                elsif enabled = '1' and rising_edge(clk) then
                        if (state = idle) then

                                        if shift_reg_full = '1' then
                                                byte_valid <= '1';
                                                shift_reg_full <= '0';
                                                bit_no := 0;

                                        -- wait for data strobe to be set
                                        elsif ds_in = '1' and shift_reg_full = '0' then
                                                byte_valid <= '0';
                                                state <= rcv;
                                                done <= '0';
                                                ds_out <= '1';
                                        end if;

                                elsif (state = rcv) then

                        -- load shift register with bit
```

```
                                        reg <= bit_in & reg(7 downto 1);

                                        ds_out <= '0';
                                        state <= ack;

                                elsif (state = ack) then

                                        -- wait for data strobe to negate
                                        if ds_in = '0' then
                                                bit_no := bit_no + 1;
                                                if bit_no = 8 then
                                                        shift_reg_full <= '1';
                                                end if;

                                                byte_out <= reg;
                                                state <= idle;
                                        end if;

                        end if;

                end if;
        end process;

end Behavioral;
```

Listing 26: Sensorkort/byte_transmit.vhd

```
--------------------------------
--      byte_transmit.vhd
--------------------------------
--
--      VHDL-kode for sender-delen av kommunikasjonen mellom AVR og FPGA
--
--------------------------------
--      2003 (c) dmprosj
--------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity byte_transmit is
    port (
                        byte_in : in std_logic_vector(7 downto 0);
                        byte_valid : in std_logic;

                        bit_out         : out std_logic;
                        ds_in                   : in std_logic;
                        ds_out          : out std_logic;

                        busy            : out std_logic;
                        reset           : in std_logic;
                        clk                     : in std_logic;
                        enabled : in std_logic
                );
end byte_transmit;

architecture Behavioral of byte_transmit  is

        type state_type is (idle, trsmit, ack);

        signal state : state_type;
        signal reg : std_logic_vector(7 downto 0);
        signal done : std_logic;

begin

        busy <= not(done);

        -------------------
        -- state machine --
        -------------------

        process (clk, reset, ds_in)
                variable bit_no : integer := 0;

        begin

                if reset = '1' then
                        state <= idle;

                        bit_no := 0;
                        reg <= "00000000";
                        ds_out <= '0';
                        done <= '1';
```

```
60
                elsif enabled = '1' and rising_edge(clk) then
                        if (state = idle) then

                                -- wait for valid byte to transmit
65                              if byte_valid = '1' and done = '1' then
                                        reg <= byte_in;
                                        done <= '0';
                                        bit_no := 0;
                                        state <= trsmit;
70                              end if;

                        elsif (state = trsmit) then

                                -- output one bit
75                              bit_out <= reg(0);
                                -- bit is valid
                                ds_out <= '1';

                                -- wait for bit to be ack'ed
80                              if (ds_in = '1') then
                                        state <= ack;
                                else
                                        state <= trsmit;
                                end if;
85
                        elsif (state = ack) then

                                -- wait for bit to be cleared
                                if (ds_in = '0') then
90
                                        -- shift register & clear data out
                                        reg <= '0' & reg(7 downto 1);
                                        ds_out <= '0';
                                        bit_no := bit_no + 1;
95
                                        state <= trsmit;

                                        if bit_no = 8 then
                                                done <= '1';
100                                             state <= idle;
                                        end if;

                                else
                                        state <= ack;
105                             end if;
                        end if;

                end if;
        end process;
110
end Behavioral;
```

Listing 27: Sensorkort/ctrl_test.vhd

```
-- VHDL Test Bench Created from source file usb_controller.vhd -- 12:23:18 11/03/2003
--
-- Notes:
5 -- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
10 --
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

15 ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

20      COMPONENT usb_controller
        PORT(
                        rcv: in STD_LOGIC;
                        vpo : out std_logic;
                        vmo : out std_logic;
25
                        vp : in std_logic;
                        vm : in std_logic;
                        eop : out std_logic;

30                      current_state : out std_logic_vector(4 downto 0); -- remove
                        out_clk : out std_logic; --remove
```

```
                        data_to_send : in std_logic_vector(7 downto 0);

35                      device_address : in std_logic_vector(6 downto 0);

                        transmit_data : out std_logic_vector(7 downto 0);

                        in_w_address : out std_logic_vector(3 downto 0);
40                      in_w_data : out std_logic_vector(7 downto 0);
                        in_w_enable : out std_logic;
                        in_validate : out std_logic;

                        fast_clk: in STD_LOGIC;
45                      rst: in STD_LOGIC
                );
        END COMPONENT;

        SIGNAL rcv :  std_logic;
50      SIGNAL vpo :  std_logic;
        SIGNAL vmo :  std_logic;
        SIGNAL vp :  std_logic;
        SIGNAL vm :  std_logic;
        signal eop : std_logic;
55      SIGNAL data_to_send :  std_logic_vector(7 downto 0);
        SIGNAL fast_clk :  std_logic;
        SIGNAL rst :  std_logic;
        signal current_state : std_logic_vector(4 downto 0);
        signal out_clk : std_logic;
60      signal device_address : std_logic_vector(6 downto 0);
        signal transmit_data : std_logic_vector(7 downto 0);
        signal in_w_address : std_logic_vector(3 downto 0);
        signal in_w_data : std_logic_vector(7 downto 0);
        signal in_w_enable : std_logic;
65      signal in_validate : std_logic;


        signal END_SIM: BOOLEAN := FALSE;
        --signal rcv_reg: STD_LOGIC_VECTOR (95 downto 0) :=
70      --"01010100010011101010101010101011010101000010100011001111101010101010101010101010101010101010";

        -- addr 2 endp 1
        signal out_token : std_logic_vector(31 downto 0) := "01010100010100001101010010110010";

75      signal in_token : std_logic_vector(31 downto 0) := "01010100010011101101010010101010";



        -- 2 bytes 10 20
80      signal data_packet : std_logic_vector(47 downto 0) := "010101000010100011001010100110101010110010101110";

        signal ack_token : std_logic_vector(15 downto 0) := "0101010011011000";


85  BEGIN

        uut: usb_controller PORT MAP(
                rcv => rcv,
                vpo => vpo,
90              vmo => vmo,
                vp => vp,
                vm => vm,
                eop => eop,
                current_state => current_state,
95              out_clk => out_clk,
                data_to_send => data_to_send,
                device_address => device_address,
                transmit_data => transmit_data,
                in_w_address => in_w_address,
100             in_w_data => in_w_data,
                in_w_enable => in_w_enable,
                in_validate => in_validate,
                fast_clk => fast_clk,
                rst => rst
105     );

        CLOCK_fast_clk : process
        begin
                if END_SIM = FALSE then
110                     fast_clk <= '0';
                        wait for 50 ns; --0 ps
                else
                        wait;
                end if;
115             if END_SIM = FALSE then
                        fast_clk <= '1';
                        wait for 50 ns; --50 ns
                else
                        wait;
```

```
120                 end if;
            end process;

    -- *** Test Bench - User Defined Section ***
            tb : PROCESS
125                 variable count: integer := 0;
            BEGIN
                    rst <= '1';
                    device_address <= "0000010";
                    -- IDLE
130                 rcv <= '1';
                    vp <= '0';
                    vm <= '1';
                    wait for 400 ns;
                    rst <= '0';
135                 wait for 400 ns; --800 ns

                    -- OUT TOKEN
                    while (count < out_token'length) loop
                            rcv <= out_token(out_token'left);
140                         out_token <= (out_token((out_token'left - 1) downto 0) & '0');
                            count := count + 1;
                            wait for 400 ns;
                    end loop;

145                 -- EOP
                    vp <= '0';
                    vm <= '0';
                    wait for 800 ns;
                    --rcv <= '0';
150                 --wait for 400 ns;

                    -- IDLE
                    rcv <= '1';
                    vp <= '0';
155                 vm <= '1';
                    count := 0;
                    wait for 800*10 ns;

                    -- DATA PACKET
160                 while (count < data_packet'length) loop
                            rcv <= data_packet(data_packet'left);
                            data_packet <= (data_packet((data_packet'left - 1) downto 0) & '0');
                            count := count + 1;
                            wait for 400 ns;
165                 end loop;

                    -- EOP
                    vp <= '0';
                    vm <= '0';
170                 wait for 800 ns;
                    --rcv <= '0';
                    --wait for 400 ns;


175                 -- IDLE
                    rcv <= '1';
                    vp <= '0';
                    vm <= '1';
                    count := 0;
180                 wait for 400*30 ns;


                    -- IN TOKEN
                    while (count < in_token'length) loop
185                         rcv <= in_token(in_token'left);
                            in_token <= (in_token((in_token'left - 1) downto 0) & '0');
                            count := count + 1;
                            wait for 400 ns;
                    end loop;
190
                    -- EOP
                    vp <= '0';
                    vm <= '0';
                    wait for 800 ns;
195
                    -- IDLE
                    rcv <= '1';
                    vp <= '0';
                    vm <= '1';
200                 count := 0;
                    wait for 400*150 ns;


                    -- ACK TOKEN
205                 while (count < ack_token'length) loop
                            rcv <= ack_token(ack_token'left);
                            ack_token <= (ack_token((ack_token'left - 1) downto 0) & '0');
```

```
                        count := count + 1;
                        wait for 400 ns;
                end loop;

                -- EOP
                vp <= '0';
                vm <= '0';
                wait for 800 ns;

                -- IDLE
                rcv <= '1';
                vp <= '0';
                vm <= '1';
                count := 0;
                wait for 400*20 ns;



                END_SIM <= TRUE;
                wait;
        END PROCESS;
-- *** End Test Bench - User Defined Section ***

END;
```

Listing 28: Sensorkort/endp_ctrl.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity endp_ctrl is
        port (avr_data_in : in std_logic_vector(7 downto 0);
                              avr_in_int : in std_logic;
                              avr_data_out : out std_logic_vector(7 downto 0);
                              avr_busy : in std_logic;
                              avr_data_out_valid : out std_logic;

                              -- IN BUFFER (writes)
                              in_w_address : in std_logic_vector(3 downto 0);
                              in_w_data : in std_logic_vector(7 downto 0);
                              in_w_enable : in std_logic;
                              in_validate : in std_logic;

                              -- OUT BUFFER (reads)
                              out_address : in std_logic_vector(3 downto 0);
                              out_data : out std_logic_vector(7 downto 0);
                              out_clear : in std_logic;
                              out_full : out std_logic;

                              device_address : out std_logic_vector(6 downto 0);
                              interrupt : out std_logic;

                              rst : in std_logic;
                              clk : in std_logic);
end endp_ctrl;

architecture behavioral of endp_ctrl is

        component packet_buffer is
                port (
                        -- read port
            address   : in  std_logic_vector(3 downto 0);
            data      : out std_logic_vector(7 downto 0);

            -- write port
            w_address : in std_logic_vector(3 downto 0);
            w_data    : in std_logic_vector(7 downto 0);
            w_enable  : in std_logic;

                        validate : in std_logic;
                        clear : in std_logic;
                        full : out std_logic;

            clk : in std_logic;
                        rst : in std_logic);
        end component;

        -- The different states
        type avr_state_type is (avr_idle, avr_wait_receive, avr_byte_received, avr_wait_send, avr_wait_busy,
                                                                                avr_sending, avr_byte_sent, avr_byte_writt
        -- The state variables
        signal avr_state : avr_state_type;
```

```
        -- The last command received
        signal avr_command : std_logic_vector(7 downto 0);

        -- The current byte received or ready to be sent
        signal avr_byte : std_logic_vector(7 downto 0);

        -- Signals used to detect transition from 0 to 1 on avr_in_int
        signal last_avr_in_int : std_logic;
        signal current_avr_in_int : std_logic;

        -- READ PACKET
        signal in_address : std_logic_vector(3 downto 0);
        signal in_data : std_logic_vector(7 downto 0);
        signal in_full : std_logic;
        signal in_clear : std_logic;

        -- WRITE PACKET
        signal out_w_address : std_logic_vector(3 downto 0);
        signal out_w_data : std_logic_vector(7 downto 0);
        signal out_w_enable : std_logic;
        signal out_validate : std_logic;

        signal int_reg : std_logic_vector(7 downto 0);
        signal usb_interrupt : std_logic;

begin

        interrupt <= usb_interrupt;

        rcv_buf : packet_buffer
        port map( address => in_address,
                                        data => in_data ,
                                                w_address => in_w_address,
                                                w_data => in_w_data,
                                                w_enable => in_w_enable,
                                                validate => in_validate,
                                                clear => in_clear,
                                                full => in_full,
                                                clk => clk,
                                                rst => rst);

        send_buf : packet_buffer
        port map(       address => out_address,
                                        data => out_data,
                                                w_address => out_w_address,
                                                w_data => out_w_data,
                                                w_enable => out_w_enable,
                                                validate => out_validate,
                                                clear => out_clear,
                                                full => out_full,
                                                clk => clk,
                                                rst => rst);

        process (clk, rst)
                variable avr_byte_count : integer range 0 to 10;
                variable avr_packet_size : integer range 0 to 8;
                variable int_reg_read : boolean;
        begin
                if rst = '1' then
                        avr_state <= avr_idle;
                        current_avr_in_int <= avr_in_int;
                        in_address <= "0000";
                        avr_command <= (others => '0');
                        avr_data_out_valid <= '0';
                        avr_byte <= (others => '0');
                        avr_data_out <= (others => '0');
                        avr_byte_count := 0;
                        avr_packet_size := 0;
                        int_reg <= (others => '0');
                        usb_interrupt <= '0';
                        int_reg_read := false;
                        device_address <= (others => '0');
                elsif rising_edge(clk) then
                        last_avr_in_int <= current_avr_in_int;
                        current_avr_in_int <= avr_in_int;

                        if in_validate = '1' then
                                usb_interrupt <= '1';
                                int_reg(6) <= '1'; -- Packet Received
                        end if;

                        if out_clear = '1' then
                                usb_interrupt <= '1';
                                int_reg(5) <= '1'; -- Packet Sent
                        end if;

                        if usb_interrupt = '1' then
```

```
                if int_reg_read = true then
                        usb_interrupt <= '0';
                        int_reg <= (others => '0');
                        int_reg_read := false;
                end if;
        end if;

        case avr_state is
                when avr_idle =>
                        avr_byte_count := 0;
                        avr_packet_size := 0; --
                        out_validate <= '0';
                        in_clear <= '0';
                        avr_data_out_valid <= '0';
                        if last_avr_in_int = '0' and current_avr_in_int = '1' then
                                case avr_data_in is
                                        -- Set Address
                                        when X"00" =>
                                                avr_state <= avr_wait_receive;

                                        -- Read Interrupt Register
                                        when X"01" =>
                                                avr_byte <= int_reg;
                                                avr_state <= avr_wait_send;

                                        -- Write Buffer
                                        when X"02" =>
                                                avr_state <= avr_wait_receive;
                                                out_w_address <= "0000";

                                        -- Read Buffer
                                        when X"03" =>
                                                avr_state <= avr_address_set;
                                                in_address <= "0000";

                                        when others =>
                                                null;
                                end case;
                                avr_command <= avr_data_in; -- save the command
                        end if;

                when avr_address_set =>
                        avr_byte <= in_data;
                        avr_state <= avr_wait_send;

                when avr_wait_receive =>
                        if last_avr_in_int = '0' and current_avr_in_int = '1' then
                                avr_byte <= avr_data_in;
                                avr_state <= avr_byte_received;
                        end if;


                when avr_byte_received =>
                        avr_byte_count := avr_byte_count + 1;
                        case avr_command is
                                -- Set Address
                                when X"00" =>
                                        device_address <= avr_byte(6 downto 0);
                                        avr_state <= avr_idle;

                                -- Write Buffer
                                when X"02" =>
                                        out_w_data <= avr_byte;
                                        out_w_enable <= '1';
                                        if avr_byte_count = 1 then
                                                avr_packet_size := conv_integer(avr_byte(3 downto 0));
                                        end if;
                                        if avr_byte_count = (avr_packet_size + 1) then
                                                out_validate <= '1';
                                                avr_state <= avr_idle;
                                        else
                                                out_validate <= '0';
                                                avr_state <= avr_byte_written;
                                        end if;

                                when others =>
                                        null;
                        end case;


                when avr_byte_written =>
                        out_w_address <= out_w_address + 1;
                        out_w_enable <= '0';
                        avr_state <= avr_wait_receive;

                when avr_wait_send =>
                        if avr_busy = '0' then
```

```
                                        avr_data_out <= avr_byte;
                                        avr_data_out_valid <= '1';
                                        avr_state <= avr_wait_busy;
                                end if;

                        when avr_wait_busy =>
                                if avr_busy = '1' then
                                        avr_state <= avr_sending;
                                end if;

                        when avr_sending =>
                                if avr_busy = '0' then
                                        avr_data_out_valid <= '0';
                                        avr_state <= avr_byte_sent;
                                end if;


                        when avr_byte_sent =>
                                case avr_command is
                                        -- Read Interrupt Register
                                        when X"01" =>
                                                int_reg_read := true;
                                                avr_state <= avr_idle;

                                        -- Read Buffer
                                        when X"03" =>
                                                avr_byte_count := avr_byte_count + 1;
                                                if avr_byte_count = 1 then
                                                        avr_packet_size := conv_integer(avr_byte);
                                                end if;

                                                if avr_byte_count <= avr_packet_size then
                                                        avr_state <= avr_address_set;
                                                else
                                                        in_clear <= '1';
                                                        avr_state <= avr_idle;
                                                end if;
                                                in_address <= in_address + 1;

                                        when others =>
                                                null;
                                end case;


                        when others =>
                                null;

                end case;
        end if;
    end process;

end behavioral;
```

Listing 29: Sensorkort/packet_buffer.vhd

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;

-------------------------------------------------------------------------------

entity packet_buffer is

  port (
    -- read port
    address   : in  std_logic_vector(3 downto 0);
    data      : out std_logic_vector(7 downto 0);

    -- write port
    w_address : in std_logic_vector(3 downto 0);
    w_data    : in std_logic_vector(7 downto 0);
    w_enable  : in std_logic;

                validate : in std_logic;
                clear : in std_logic;
                full : out std_logic;

    clk : in std_logic;
                rst : in std_logic

                );

end packet_buffer;

-------------------------------------------------------------------------------
```

```
architecture arch of packet_buffer is

  type mem_type is array (0 to 15) of std_logic_vector(7 downto 0);
  signal mem_storage : mem_type := (others => (others => '0'));

begin

  process (rst, clk)
  begin
    if rst = '1' then
                        full <= '0';
                elsif rising_edge (clk) then
                        if w_enable = '1' then
        mem_storage(conv_integer(w_address)) <= w_data;
                        end if;

                        if validate = '1' then
                                full <= '1';
                        elsif clear = '1' then
                                full <= '0';
                        end if;
                end if;
        end process;

        process (address, mem_storage)
        begin
                data <= mem_storage(conv_integer(address));
        end process;

end arch;
```

Listing 30: Sensorkort/receiver.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity receiver is
        port (
                input: in STD_LOGIC;
                vp : in std_logic;
                vm : in std_logic;
                enable_rec : in STD_LOGIC;
                quick_enable : in STD_LOGIC;
                output : out STD_LOGIC_VECTOR(7 downto 0);
                byte_received: out STD_LOGIC;
                sop_detect : out std_logic;
                eop_detect : out std_logic;
                crc5_ok : out std_logic;
                crc16_ok : out std_logic;
                calc_crc5 : in std_logic;
                calc_crc16 : in std_logic;
                rst : in STD_LOGIC;
                clk : in STD_LOGIC;
                clk_out : out STD_LOGIC
        );
end receiver;

architecture receiver of receiver is
        signal last_input: STD_LOGIC;
        signal stuffed_data: STD_LOGIC;
        signal bit_count: INTEGER; --Keeps track of consecutive 1's.
        signal data: STD_LOGIC_VECTOR (7 downto 0);
        signal shift_count: INTEGER; --Keeps track of number of bits

        --shifted by serial to parallel converter.
        -- signals for clk.
        signal to_buffer: std_logic;

        ---------------------------------
        --Signals for PLL
        ---------------------------------
        signal pll_last_input: STD_LOGIC;
        signal my_clk: STD_LOGIC;
        signal last_output_sig: STD_LOGIC;
        signal pll_count: INTEGER;

        --Signals for multiplexor.
        signal out_byte: STD_LOGIC_VECTOR(7 downto 0);

        --Signals for pid checker.
        signal pid: STD_LOGIC_VECTOR(7 downto 0);

        --Signals for sop_detect of packet detector.
        type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
        signal state,next_state:state_type;
```

```
55
        type eop_state_type is (eop_idle, one_se0, two_se0);
        signal eop_state, eop_next_state : state_type;


60      --Signals for CRC calculators.
        signal crc_next_state: STD_LOGIC_VECTOR(15 downto 0);
        signal crc_state: STD_LOGIC_VECTOR(15 downto 0);
        signal out_crc16: STD_LOGIC_VECTOR(15 downto 0);

65      signal crc5_next_state: STD_LOGIC_VECTOR(4 downto 0);
        signal crc5_state: STD_LOGIC_VECTOR(4 downto 0);
        signal out_crc5: STD_LOGIC_VECTOR(4 downto 0);


70      begin

clk_driver:
process(my_clk)
begin
75      if my_clk = '1' then
                clk_out <= '1';
        else
                clk_out <= '0';
        end if;
80 end process;

-- this must be fixed ..
sop_detector:
process(state, input)
85 begin
        if enable_rec = '0' then
                case state is
                        when s0 =>if input = '0' then next_state <= s1;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
90                      when s1=>if input = '1' then next_state <= s2;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
                        when s2=>if input = '0' then next_state <= s3;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
                        when s3=>if input = '1' then next_state <= s4;sop_detect <= '0';
95                              else next_state <= s0;sop_detect <= '0';end if;
                        when s4=>if input = '0' then next_state <= s5;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
                        when s5=>if input = '1' then next_state <= s6;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
100                     when s6=>if input = '0' then next_state <= s7;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
                        when s7=>if input = '0' then next_state <= s8;sop_detect <= '0';
                                else next_state <= s0;sop_detect <= '0';end if;
                        when s8=>next_state <= s0;sop_detect <= '1';
105             end case;
        else
                next_state <= s0;
        end if;
end process;
110

-----------------------------------------
--State updater for sop_detect of packet detector.
-----------------------------------------
115 process(my_clk, rst)
begin
        if rst = '1' then
                state <= s0;
        elsif rising_edge(my_clk) then
120             state <= next_state;
        end if;
end process;


125
-----------------------------------
-- PLL for clock recovery from data
-----------------------------------
PLL:
130 process (clk)
        variable last_output: STD_LOGIC;
begin
        if rst = '1' then
                pll_last_input <= '0';
135             last_output_sig <= '0';
                pll_count <= 0;
                last_output := '0';
        elsif clk'event and clk='1' then
                last_output := last_output_sig;
140
                if not(pll_last_input = input) then
```

```
                        last_output := not (last_output);
                        pll_count <= 0;
                elsif pll_count = 1 then
                        last_output := not (last_output);
                        pll_count <= 0;
                else
                        pll_count <= pll_count + 1;
                end if;
        end if;

        pll_last_input <= input;
        last_output_sig <= last_output;
        my_clk <= last_output;
end process;



main_process:
process (my_clk, rst)
        variable tmp_stuffed_data: STD_LOGIC;
        variable tmp_data: STD_LOGIC_VECTOR(7 downto 0);
        variable tmp_bit_count: INTEGER;
        variable tmp_shift_count: INTEGER; --Keeps track of number of bits

        variable se0_count : integer;

begin
        if rst = '1' then
                last_input <= '0';
                stuffed_data <= '0';
                bit_count <= 0;
                data <= "00000000";
                shift_count <= 0;
                byte_received <= '0';
                out_byte <= "00000000";
                crc5_ok <= '1';
                crc16_ok <= '1';

                se0_count       := 0;
                eop_detect <= '0';

        ------------------
        -- NRZI Decoder --
        ------------------
        elsif my_clk'EVENT and my_clk = '1' then
                if vp = '0' and vm = '0' then
                        se0_count := se0_count + 1;
                        if se0_count = 2 then
                                eop_detect <= '1';
                                se0_count := 0;
                        else
                                eop_detect <= '0';
                        end if;
                else
                        eop_detect <= '0';
                        if enable_rec = '1' or quick_enable = '1' then
                                tmp_stuffed_data := stuffed_data;
                                tmp_data := data;
                                tmp_bit_count := bit_count;
                                tmp_shift_count := shift_count;
                                        if (input = last_input) then
                                                tmp_stuffed_data := '1';
                                        else
                                                tmp_stuffed_data := '0';
                                        end if;

                                ------------------------
                                -- Stuffed Bit Remover --
                                ------------------------
                                -- Keep count of of number of 1's.
                                -- This info will be used to remove stuffed bit.
                                if tmp_stuffed_data = '1' then
                                        tmp_bit_count := tmp_bit_count + 1;
                                else
                                        tmp_bit_count := 0;
                                end if;

                                --Only add input to the data register if it is not
                                --a stuffed bit.
                                if not(tmp_bit_count = 6) then
                                        tmp_data(7) := tmp_data(6);
                                        tmp_data(6) := tmp_data(5);
                                        tmp_data(5) := tmp_data(4);
                                        tmp_data(4) := tmp_data(3);
                                        tmp_data(3) := tmp_data(2);
                                        tmp_data(2) := tmp_data(1);
                                        tmp_data(1) := tmp_data(0);
                                        tmp_data(0) := tmp_stuffed_data;
```

```
230                                     tmp_shift_count := tmp_shift_count + 1;
                                    else
                                        --Set bit_count to 0 since stuffed bit has
                                        --been removed.
                                        tmp_bit_count := 0;
235                                 end if;

                                    -- Let the outside world know if a byte has been received.
                                    if tmp_shift_count = 8 then
                                        byte_received <= '1';
240                                     out_byte <= tmp_data;
                                        --Reset shift_count for next byte.
                                        tmp_shift_count := 0;
                                    else
                                        byte_received <= '0';
245                                 end if;

                                    stuffed_data <= tmp_stuffed_data;
                                    data <= tmp_data;
                                    output <= tmp_data; --
250                                 bit_count <= tmp_bit_count;
                                    shift_count <= tmp_shift_count;
                                    last_input <= input;
                                end if;
                            end if;
255         end if;
    end process;


    -- mye feil her.. åm ikke regne crc åp stuffed_data
260
    ----------------------
    --16-bit CRC calculator--
    ----------------------
    process(clk, rst, stuffed_data)
265 begin
        if clk'EVENT and clk = '1' then
            out_crc16 <= crc_next_state;
            crc_state <= crc_next_state;
        end if;
270     if rst = '1' then
            crc_state <= "0000000000000000";
            out_crc16 <= "0000000000000000";
            crc_next_state <= "0000000000000000";
        else
275         crc_next_state(15) <= crc_state(14) xor stuffed_data xor crc_state(15);
            crc_next_state(14) <= crc_state(13);
            crc_next_state(13) <= crc_state(12);
            crc_next_state(12) <= crc_state(11);
            crc_next_state(11) <= crc_state(10);
280         crc_next_state(10) <= crc_state(9);
            crc_next_state(9) <= crc_state(8);
            crc_next_state(8) <= crc_state(7);
            crc_next_state(7) <= crc_state(6);
            crc_next_state(6) <= crc_state(5);
285         crc_next_state(5) <= crc_state(4);
            crc_next_state(4) <= crc_state(3);
            crc_next_state(3) <= crc_state(2);
            crc_next_state(2) <= crc_state(1) xor stuffed_data xor crc_state(15);
            crc_next_state(1) <= crc_state(0);
290         crc_next_state(0) <= stuffed_data xor crc_state(15);
        end if;
    end process;

    ----------------------
295 --5-bit CRC calculator--
    ----------------------
    process(clk)
    begin
        if clk'EVENT and clk = '1' then
300         out_crc5 <= crc5_next_state;
            crc5_state <= crc5_next_state;
        end if;
        if rst = '1' then
            crc5_state <= "00000";
305         out_crc5 <= "00000";
            crc5_next_state <= "00000";
        else
            crc5_next_state(4) <= crc5_state(3);
            crc5_next_state(3) <= crc5_state(2);
310         crc5_next_state(2) <= crc5_state(1) xor stuffed_data xor crc5_state(4);
            crc5_next_state(1) <= crc5_state(0);
            crc5_next_state(0) <= stuffed_data xor crc5_state(4);
        end if;
    end process;
315
    end receiver;
```

Listing 31: Sensorkort/transmitter.vhd

```
library ieee;
use ieee.std_logic_1164.all;

entity transmitter is
        port (
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                input: in STD_LOGIC_VECTOR (7 downto 0);
                input_select: in STD_LOGIC_VECTOR(2 downto 0);
                pid_select: in STD_LOGIC_VECTOR(1 downto 0);
                enable: in STD_LOGIC;
                send_eop: in STD_LOGIC;
                eop_sent : out std_logic;
                vpo : out std_logic;
                vmo : out std_logic;
                out_byte_sent: out STD_LOGIC
        );
end transmitter;

architecture transmitter of transmitter is
        signal last_output: STD_LOGIC;
        signal stuffed_data: STD_LOGIC;
        signal bit_count: INTEGER RANGE 0 to 6;
        signal shift_count: INTEGER RANGE 0 to 8;
        signal Sreg: STD_LOGIC_VECTOR(7 downto 0);
        signal mult_output: STD_LOGIC_VECTOR(7 downto 0);
        signal byte_sent: STD_LOGIC;
        signal to_buffer: std_logic;
        signal output: std_logic;

        -- Signals for CRC 16 calculator.
        signal state, next_state, out_crc16: std_logic_vector(15 downto 0);

        -- Signal for PID generator.
        signal pid: STD_LOGIC_VECTOR(7 downto 0);

        signal send_se0 : std_logic;

begin

        transceiver_interface:
        process(output, send_se0)
        begin
                if send_se0 = '1' then
                        vpo <= '0';
                        vmo <= '0';
                else
                        if output = '1' then
                                vpo <= '1';
                                vmo <= '0';
                        else
                                vpo <= '0';
                                vmo <= '1';
                        end if;
                end if;
        end process;

        main:
        process(clk, rst)
                variable tmp_Sreg: STD_LOGIC_VECTOR(7 downto 0);
                variable tmp_stuffed_data: STD_LOGIC;
                variable tmp_shift_count: INTEGER RANGE 0 to 8;
                variable tmp_bit_count: INTEGER RANGE 0 to 6;
                variable bit_time_count : integer range 0 to 5;
        begin
                if rst = '1' then
                        output <= '1';
                        last_output <= '0';
                        tmp_stuffed_data := '1';
                        tmp_bit_count := 0;
                        tmp_shift_count := 0;
                        tmp_Sreg := "00000000";
                        byte_sent <= '0';
                        stuffed_data <= tmp_stuffed_data;
                        shift_count <= tmp_shift_count;
                        bit_count <= tmp_bit_count;

                        bit_time_count := 0;
                        send_se0 <= '0'; -- will start the process above..
                        eop_sent <= '0';

                elsif clk'EVENT and clk = '1' then
                        tmp_stuffed_data := stuffed_data;
                        tmp_shift_count := shift_count;
                        tmp_bit_count := bit_count;
                        tmp_Sreg := Sreg;
```

```
                        if tmp_shift_count = 0 then
                                tmp_Sreg := mult_output;
                        end if;
                        if enable = '1' then
                                if send_eop = '0' then
                                        bit_time_count := 0;
                                        send_se0 <= '0';
                                        eop_sent <= '0';

                                        if tmp_bit_count = 6 then
                                                tmp_stuffed_data := '0';
                                                tmp_bit_count := 0;
                                        else
                                                tmp_stuffed_data := tmp_Sreg(7);
                                                tmp_Sreg := tmp_Sreg(6 downto 0) & '1';

                                                -- Only increment counter when normal bit is sent.
                                                tmp_shift_count := tmp_shift_count + 1;
                                        end if;

                                        -- Keep count of of number of 1's.
                                        -- This info will be used to add stuffed bit.
                                        if tmp_stuffed_data = '1' then
                                                tmp_bit_count := tmp_bit_count + 1;
                                        else
                                                tmp_bit_count := 0;
                                        end if;
                                        if tmp_shift_count = 8 then
                                                out_byte_sent <= '1';
                                                tmp_shift_count := 0;
                                                tmp_Sreg := mult_output;
                                        else
                                                out_byte_sent <= '0';
                                        end if;
                                else
                                        bit_time_count := bit_time_count + 1;
                                        if bit_time_count = 3 then
                                                send_se0 <= '0';
                                                output <=       '0'; -- send a 'J', ie. 0
                                                eop_sent <= '1';
                                                bit_time_count := 0;
                                        else
                                                send_se0 <= '1';
                                        end if;
                                end if;

                        end if;

                        if tmp_stuffed_data = '0' then
                                last_output <= not(last_output);
                                output <= last_output;
                        end if;

                        -- Update signal values.
                        Sreg <= tmp_Sreg;
                        stuffed_data <= tmp_stuffed_data;
                        shift_count <= tmp_shift_count;
                        bit_count <= tmp_bit_count;
                end if;
        end process;

        ------------------------
        --5-bit CRC calculator--
        ------------------------
        process(clk, rst, stuffed_data)
        begin
                if clk'EVENT and clk = '1' then
                        out_crc16 <= next_state;
                        state <= next_state;
                end if;
                if rst = '1' then
                        state <= "0000000000000000";
                        out_crc16 <= "0000000000000000";
                        next_state <= "0000000000000000";
                else
                        next_state(15) <= state(14) xor stuffed_data xor state(15);
                        next_state(14) <= state(13);
                        next_state(13) <= state(12);
                        next_state(12) <= state(11);
                        next_state(11) <= state(10);
                        next_state(10) <= state(9);
                        next_state(9) <= state(8);
                        next_state(8) <= state(7);
                        next_state(7) <= state(6);
                        next_state(6) <= state(5);
                        next_state(5) <= state(4);
                        next_state(4) <= state(3);
                        next_state(3) <= state(2);
```

```
175                         next_state(2) <= state(1) xor stuffed_data xor state(15);
                            next_state(1) <= state(0);
                            next_state(0) <= stuffed_data xor state(15);
                    end if;
            end process;
180
            ------------------
            --PID Generator--
            ------------------
            process (pid_select)
185         begin
                    case pid_select is
                            --ACK
                            when "00" => pid <= "01001011";
                            --NAK
190                         when "01" => pid <= "01011010";
                            --Send DATA0 packet.
                            when others => pid <= "11000011";
                    end case;
            end process;
195
            ------------------
            -- Multiplexor--
            ------------------
            process (input_select, input, out_crc16)
200         begin
                    case input_select is
                            when "000" => mult_output <= pid;
                            when "100" => mult_output <= out_crc16(7 downto 0);
                            when "101" => mult_output <= out_crc16(15 downto 8);
205                         when "011" => mult_output <= input;
                            when "111" => mult_output <= "00000001";
                            when others => mult_output <= input;
                    end case;
            end process;
210
    end transmitter;
```

Listing 32: Sensorkort/Packetbuilder.java

```
    public class PacketBuilder {

 5          public PacketBuilder(String[] input) {

                    int[] data = new int[100];
                    int size = 0;

10                  data[size++] = 0x01;

                    if (input[0].equals("ack")) {
                            data[size++] = 0x4B; // 0100 1011
                    }
15                  else if (input[0].equals("nack")) {
                            data[size++] = 0x5A; // 0101 1010
                    }
                    else if (input[0].equals("stall")) {
                            data[size++] = 0x78; // 0101 1010
20                  }
                    else if (input[0].equals("data0") || input[0].equals("data1")) {
                            if (input[0].equals("data0")) {
                                    data[size++] = 0xC3; // 1100 0011
                            } else {
25                                  data[size++] = 0xD2; // 1101 0010
                            }
                            for (int i = 1; i < input.length; i++) {

                                    int d = Integer.parseInt(input[i]);
30                                  int b = 0;
                                    for (int j = 0; j < 8; j++) {
                                            int bit;
                                            if ((d & (1 << j)) != 0)
                                                    bit = 1;
35                                          else
                                                    bit = 0;
                                            b |= bit << (7 - j);
                                    }
                                    data[size++] = b;
40                          }

                            // TODO
                            data[size++] = 5; // CRC
                            data[size++] = 6;
45                  }
                    else if (input[0].equals("in") || input[0].equals("out") || input[0].equals("setup")) {
```

```
                if (input[0].equals("in")) {
                        data[size++] = 0x96;
                }
                else if (input[0].equals("out")) {
                        data[size++] = 0x87;
                }
                else if (input[0].equals("setup")) {
                        data[size++] = 0xB4;
                }

                int addr = Integer.parseInt(input[1]);
                int endp = Integer.parseInt(input[2]);


                StringBuffer content = new StringBuffer();
                for (int i = 0; i < 7; i++) {
                        if ((addr & (1 << i)) == 0)
                                content.append(0);
                        else
                                content.append(1);
                }
                content.append(endp & 0x01);
                for (int i = 1; i < 4; i++) {
                        if ((endp & (1 << i)) == 0)
                                content.append(0);
                        else
                                content.append(1);
                }

                // TODO
                content.append("00000"); // CRC5

                int j = 0;
                for (int i = 7; i >= 0; i--){
                        int bit;
                        if (content.charAt(j) == '1')
                                bit = 1;
                        else
                                bit = 0;
                        data[size] |= bit << i;
                        j++;
                }

                for (int i = 7; i >= 0; i--){
                        int bit;
                        if (content.charAt(j) == '1')
                                bit = 1;
                        else
                                bit = 0;
                        data[size+1] |= bit << i;
                        j++;
                }
                size += 2;

                //System.out.println(content.length());
        }

        //for (int i = 0; i < size; i++){
        //      System.out.println(Integer.toBinaryString(data[i]));
        //}


        StringBuffer bits = new StringBuffer();
        int prev = 1;
        int ones = 0;
        for (int i = 0; i < size; i++) {
                int b = data[i];
                for (int j = 7; j >= 0; j--){
                        if ((b & (1 << j)) == 0) { // 0
                                if (prev == 0) {
                                        bits.append(1);
                                        prev = 1;
                                }
                                else {
                                        bits.append(0);
                                        prev = 0;
                                }
                                ones = 0;
                        }
                        else { // 1
                                bits.append(prev);
                                ones++;
                                // bit stuffing
                                if (ones == 6) {
                                        if (prev == 0) {
                                                bits.append(1);
                                                prev = 1;
                                        }
```

```
                                        else {
                                                bits.append(0);
                                                prev = 0;
                                        }
                                        ones = 0;
                                }
                        }
                }
                System.out.println(bits.toString());
        }

        public static void main(String[] args) {
                new PacketBuilder(args);
        }

}
```

F.3 Displaykort - AVR

Listing 33: Displaykort/menu.c

```
#include <stdio.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include <avr/signal.h>
#include <avr/interrupt.h>
#include "display.h"
#include "menustrings.h"

/* Dette er arrayet som deklarerer meny-hierarkiet.
 * Kanskje ikke helt enkel aa forstaa, men den blir antakeligvis klarere
 * om man leser definisjonen av struct Menu og struct Menuitem foerst.
 * Kort forklart er dette en initialisering av en relativt innviklet
 * datastruktur
 */
struct Menu menuarr[] = {
        { 0, 5, {                       /* item 0, main meny */
                { 0,    &menuarr[2],    2, KEYSYM_1 },
                { 0,    &menuarr[1],    3, KEYSYM_2 },
                { 0,    &menuarr[4],    4, KEYSYM_3 },
                { 0,    &menuarr[3],    1, KEYSYM_4 },
                { 0,    &menuarr[3],    1, KEYSYM_D }
                }
        },
        { 3, 5, {                       /* item 1, reports */
                {&sensor_prompt,        0,      5, KEYSYM_1 },
                { 0,    &menuarr[5],    6, KEYSYM_2 },
                {&sensor_show_time,     0,      14, KEYSYM_3 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[0],    0,      KEYSYM_B }
                }
        },
        { 2, 6, {                       /* item 2, status */
                {&time_show,            0,      7, KEYSYM_1 },
                {&sensor_show_now,      0,      8, KEYSYM_2 },
                {&connections_show,     0,      9, KEYSYM_3 },
                {&flash_status_show, 0, 10, KEYSYM_4 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[0],    0,      KEYSYM_B }
                }
        },
        { 1, 5, {                       /* item 3, debug */
                { 0,    &menuarr[0],    0, KEYSYM_1 },
                { 0,    &menuarr[0],    0, KEYSYM_2 },
                { 0,    &menuarr[0],    0, KEYSYM_3 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[0],    0,      KEYSYM_B }
                }
        },
        { 4, 6, {                       /* item 4, settings */
                {&sample_interval_prompt,0,11, KEYSYM_1 },
                {&time_prompt,          0,      7, KEYSYM_2 },
                { 0,    &menuarr[6],    12, KEYSYM_3 },
                {&sensor_mapping,       0,      13, KEYSYM_4 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[0],    0,      KEYSYM_B }
                }
        },
        { 6, 6, {                       /* item 5, graphs */
                { 0,    &menuarr[7],    15, KEYSYM_1 },
                {&setenv_gp_start,      0,      16, KEYSYM_2 },
```

```
                {&setenv_gp_stop,       0,      17,     KEYSYM_3 },
                {&graph_show,           0,      18,     KEYSYM_4 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[1],    0,      KEYSYM_B }
                }
        },
        { 12, 7, {                      /* item 6, language */
                {&language_english,     &menuarr[4],    19, KEYSYM_1 },
                {&language_german,      &menuarr[4],    20, KEYSYM_2 },
                {&language_norwegian,&menuarr[4],       21, KEYSYM_3 },
                {&language_french,      &menuarr[4],    22, KEYSYM_4 },
                {&language_quenya,      &menuarr[4],    23,     KEYSYM_5 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[4],    0,      KEYSYM_B }
                }
        },
        { 15, 5, {                      /* item 7, graphtype */
                {&setenv_gp_gt_hi,      &menuarr[5],    24,     KEYSYM_1 },
                {&setenv_gp_gt_med,     &menuarr[5],    25,     KEYSYM_2 },
                {&setenv_gp_gt_lo,      &menuarr[5],    26, KEYSYM_3 },
                { 0,    &menuarr[0],    0,      KEYSYM_A },
                { 0,    &menuarr[5],    0,      KEYSYM_B }
                }
        }
};

/* funksjon for aa lese fremste tegn i keyboardbufferet */
int8_t keyboard_getchar(void) {
        int8_t i = 0;

        if(globals.keybbufend == globals.keybbufstart)
                return -1;

        i = globals.keybbuf[globals.keybbufstart];

        if(globals.keybbufstart == KEYBOARD_BUFFER_SIZE)
                globals.keybbufstart = 0;
        else
                globals.keybbufstart++;
        return i;
}

/* funksjon for aa appende et tegn til keyboarbufferet */
void keyboard_ungetchar(int8_t i) {
        if((globals.keybbufend + 1) == globals.keybbufstart)
                return;
        globals.keybbuf[globals.keybbufend++] = i;
        if(globals.keybbufend == KEYBOARD_BUFFER_SIZE+1)
                globals.keybbufend = 0;
}

/* Interrupt-handler for keyboard */
INTERRUPT(SIG_INTERRUPT6) {
        int8_t i;
        cli();
        /* Jeg _tror_ dette er bitene vi trenger */;
        i = (int8_t) ((PINB & 0xF0) >> 4);
        PORTC = ~i;
        keyboard_ungetchar(i);
        sei();
}

#ifdef KEYBOARD_DIRECT
/*
INTERRUPT(SIG_INTERRUPT4) {
        uint8_t i;
        cli();
        EIMSK = 0;
        display_putc('4');

        for(i=0;i<4;i++){
                PORTE = ~(1 << i);
                if((PINE & 0xf0) != 0xf0) {
                        PORTC = ~(0+i);
                        break;
                }
        }
        PORTE = 0xf0;
        EIMSK = ((1 << INT7) | (1 << INT6) | (1 << INT5) | (1 << INT4));
        sei();
}

INTERRUPT(SIG_INTERRUPT5) {
        uint8_t i;
        cli();
        EIMSK = 0;
        display_putc('5');
```

```
150             for(i=0;i<4;i++){
                        PORTE = ~(1 << i);
                        if((PINE & 0xf0) != 0xf0) {
                                PORTC = ~(4+i);
                                break;
155                     }
                }
                PORTE = 0xf0;
                EIMSK = ((1 << INT7) | (1 << INT6) | (1 << INT5) | (1 << INT4));
                sei();
160     }

        INTERRUPT(SIG_INTERRUPT6) {
                uint8_t i;
                cli();
165             EIMSK = 0;
                display_putc('6');

                for(i=0;i<4;i++){
                        PORTE = ~(1 << i);
170                     if((PINE & 0xf0) != 0xf0) {
                                PORTC = ~(8+i);
                                break;
                        }
                }
175             PORTE = 0xf0;
                EIMSK = ((1 << INT7) | (1 << INT6) | (1 << INT5) | (1 << INT4));
                sei();
        }

180     INTERRUPT(SIG_INTERRUPT7) {
                uint8_t i;
                cli();
                EIMSK = 0;
                display_putc('7');
185
                for(i=0;i<4;i++){
                        PORTE = ~(1 << i);
                        if((PINE & 0xf0) != 0xf0) {
                                PORTC = ~(12+i);
190                             break;
                        }
                }
                PORTE = 0xf0;
                EIMSK = ((1 << INT7) | (1 << INT6) | (1 << INT5) | (1 << INT4));
195             sei();
        }

        */
        #endif
200
        /* Alternativ interrupt-handler for aa gi keyboard-input
         * paa uart1 */
        #ifdef SERIOUS_DEBUGGING
        SIGNAL(SIG_USART0_RECV) {
205             char c;
                c = UDR0;
                PORTC = ~4;
                switch(c) {
                        case '0': keyboard_ungetchar(KEYSYM_0); break;
210                     case '1': keyboard_ungetchar(KEYSYM_1); break;
                        case '2': keyboard_ungetchar(KEYSYM_2); break;
                        case '3': keyboard_ungetchar(KEYSYM_3); break;
                        case '4': keyboard_ungetchar(KEYSYM_4); break;
                        case '5': keyboard_ungetchar(KEYSYM_5); break;
215                     case '6': keyboard_ungetchar(KEYSYM_6); break;
                        case '7': keyboard_ungetchar(KEYSYM_7); break;
                        case '8': keyboard_ungetchar(KEYSYM_8); break;
                        case '9': keyboard_ungetchar(KEYSYM_9); break;
                        case 'a': keyboard_ungetchar(KEYSYM_A); break;
220                     case 'b': keyboard_ungetchar(KEYSYM_B); break;
                        case 'c': keyboard_ungetchar(KEYSYM_C); break;
                        case 'd': keyboard_ungetchar(KEYSYM_D); break;
                        case '#': keyboard_ungetchar(KEYSYM_HASH); break;
                        case '*': keyboard_ungetchar(KEYSYM_STAR); break;
225                     default: PORTC = ~5; break;
                }
                return;
        }
        #endif
230
        /* funksjon som viser en struct Menu * paa displayet */
        void menu_show(struct Menu *menu) {
                display_clear();
                display_puts_P(*(menustrings + (*menu).msgid));
235             display_position(0,1);
                display_puts_P(*(menustrings + (*menu).item[0].msgid));
```

```
        if(!((*menu).item[1].func || (*menu).item[1].submenu) || ((*menu).item[1].msgid == 0))
                return;
        display_position(0,2);
        display_puts_P(*(menustrings + (*menu).item[1].msgid));

        if(!((*menu).item[2].func || (*menu).item[2].submenu) || ((*menu).item[2].msgid == 0))
                return;
        display_position(0,3);
        display_puts_P(*(menustrings + (*menu).item[2].msgid));

        if(!((*menu).item[3].func || (*menu).item[3].submenu) || ((*menu).item[3].msgid == 0))
                return;
        display_position(10,1);
        display_puts_P(*(menustrings + (*menu).item[3].msgid));

        if(!((*menu).item[4].func || (*menu).item[4].submenu) || ((*menu).item[4].msgid == 0))
                return;
        display_position(10,2);
        display_puts_P(*(menustrings + (*menu).item[4].msgid));

        if(!((*menu).item[5].func || (*menu).item[5].submenu) || ((*menu).item[5].msgid == 0))
                return;
        display_position(10,3);
        display_puts_P(*(menustrings + (*menu).item[5].msgid));
}

void delay(uint8_t i) {
        uint8_t j;
        for(j=0;j<i;j++) {
                while(!(TIFR&0x02));
                TIFR = 0x02;
        }
}

int main(void) {
        struct Menu *curmenu;
        int8_t key;
        int i;
        uint8_t x,y;

        TCCR0 = 0x05;
        delay(200);

        curmenu = &menuarr[0];
        key = -1;
        i = 0;

        globals.refresh_menu=1;
        globals.keybbufstart=0;
        globals.keybbufend=0;

        language_english();

        DDRC = 0xff;
        PORTC = ~1;
        display_uart_init();
        display_configure();
        display_bl_on();
        EICRB = 0x30;
        EIMSK = (1 << INT6);

        PORTC = ~2;
        sei();
        PORTC = ~3;
/*
        for(x = 0; x < 120; x++)
                for(y = 0; y < x/4; y++)
                        displaybuf_setpixel(x,y);
        PORTC = ~4;
        display_show(globals.displaybuf);
        PORTC = ~5;

        while(1)
                delay(255);
*/

//      keyboard_ungetchar(KEYSYM_A);
// **** DEBUGSTUFF END ****
        while(1) {
                delay(20);
                if(globals.refresh_menu) {
                        menu_show(curmenu);
                        globals.refresh_menu = 0;
                }
                key = keyboard_getchar();
                if(key == -1)
                        continue;
                for(i=0; i < (*curmenu).itemc; i++) {
```

```
                        if(key == (*curmenu).item[i].key) {
                                if((*curmenu).item[i].func)
                                        ((*curmenu).item[i].func)();
                                if((*curmenu).item[i].submenu) {
                                        curmenu = (*curmenu).item[i].submenu;
                                        globals.refresh_menu = 1;
                                }
                        }
                }
        }
}

uint32_t prompt_integer(uint16_t textindex) {
        int8_t key;
        uint32_t work;

        key = -1;
        work = 0;
        globals.refresh_menu = 1;

        display_clear();
        display_position(0,0);
        display_puts_P(*(menustrings + textindex));
        display_position(0,2);
        while(( key = keyboard_getchar() )) {
                PORTC = ~key;
                switch(key) {
                        case KEYSYM_0:
                                display_putc('0');
                                work*=10;
                                break;
                        case KEYSYM_1:
                                display_putc('1');
                                work*=10;
                                work+=1;
                                break;
                        case KEYSYM_2:
                                display_putc('2');
                                work*=10;
                                work+=2;
                                break;
                        case KEYSYM_3:
                                display_putc('3');
                                work*=10;
                                work+=3;
                                break;
                        case KEYSYM_4:
                                display_putc('4');
                                work*=10;
                                work+=4;
                                break;
                        case KEYSYM_5:
                                display_putc('5');
                                work*=10;
                                work+=5;
                                break;
                        case KEYSYM_6:
                                display_putc('6');
                                work*=10;
                                work+=6;
                                break;
                        case KEYSYM_7:
                                display_putc('7');
                                work*=10;
                                work+=7;
                                break;
                        case KEYSYM_8:
                                display_putc('8');
                                work*=10;
                                work+=8;
                                break;
                        case KEYSYM_9:
                                display_putc('9');
                                work*=10;
                                work+=9;
                                break;
                        case KEYSYM_STAR:
                                work=0;
                                display_position(0,2);
                                display_puts_P(PSTR("          "));
                                display_position(0,2);
                                break;
                        case -1:
                                break;
                        case KEYSYM_HASH:
                                PORTC = (uint8_t) ~work;
                                return work;
                        break;
```

```
                }
        }
}

void prompt_uint8ptr(uint8_t *intptr, uint16_t textindex) {
        *intptr = (uint8_t) prompt_integer(textindex);
}

void prompt_uint16ptr(uint16_t *intptr, uint16_t textindex) {
        *intptr = (uint16_t) prompt_integer(textindex);
}

void prompt_uint32ptr(uint32_t *intptr, uint16_t textindex) {
        *intptr = (uint32_t) prompt_integer(textindex);
}

void language_english(void) {
        menustrings = menustrings_en;
}

void language_german(void) { }

void language_french(void) { }

void language_quenya(void) { }

void language_norwegian(void) {
        menustrings = menustrings_no;
}
```

Listing 34: Displaykort/display.c

```
#include <stdio.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "display.h"


/* Rimelig selvforklarende; funksjon som skriver en byte til displayet
*/
void display_putc(char c) {
        loop_until_bit_is_set(UCSR1A, UDRE1);
        UDR1=c;
        return;
}

void display_puts(char *s) {
        while(*s++)
                display_putc(*s);
}

void display_puts_P(const char *s) {
        char c;
        while((c=pgm_read_byte_near(s++)))
                display_putc(c);
}

/* Denne funksjonen skriver minneinnholdet f.o.m *buf og DISPLAY_BUFSIZE bytes
 * fremover ut til displayet som et bitmap. Kan brukes til debugging ved aa
 * kalle funksjonen med noe annet enn &globals.displaybuf som parameter.
 */
void display_show(char *buf) {
        uint16_t i = 0;
        globals.refresh_menu=1;
        display_putc(0x1b);
        display_putc('D');
        display_putc('G');

        while(i < DISPLAY_BUFSIZE) {
                display_putc(*(buf+i++));
        }
}

void display_clear() {
        display_putc(0x0c);
        display_putc(0x01);
        display_position(0,0);
}

void display_bl_on() {
        display_putc(0x0e);
}

void display_bl_off() {
        display_putc(0x0f);
```

```
}

void display_position(uint8_t x, uint8_t y) {
        char c = (64 + x + y*20);
        display_putc(0x10);
        display_putc(c);
};

void display_uart_init() {
        UBRR1L = 47; /* 9600bps ved 7.3728 Mhz */
/*      UBRR1L = 23; /* 9600bps ved 3.68 Mhz */
        UBRR1H = 0;
        UCSR1B = (1 << TXEN);
        UCSR1C = ((1 << UCSZ10) | (1 << UCSZ11)); /* 8 databits */
}

void display_configure() {
        display_putc(0x1b);
        display_putc('V');
        display_putc(0x41);
        display_clear();
}

void debug_uart_init() {
        UBRR0L = 23;
        UBRR0H = 0;
        UCSR0B = ((1 << RXEN0) | (1 << RXCIE0));
        UCSR0C = ((1 << UCSZ00) | (1 << UCSZ01));
}

void displaybuf_setpixel(uint8_t x, uint8_t y) {
        uint16_t byteoff;
        uint8_t bitoff;
        char *c;

        y = 31-y;
        byteoff = (uint16_t) (x + (y/8)*120);
        bitoff  = (uint8_t)  (y%8);

        c = (globals.displaybuf + byteoff);
        *c |= (1 << bitoff);
}

void displaybuf_unsetpixel(uint8_t x, uint8_t y) {
        uint16_t byteoff;
        uint8_t bitoff;
        char *c;

        y = 31-y;
        byteoff = (uint16_t) (x + (y/8)*120);
        bitoff  = (uint8_t)  (y%8);

        c = (globals.displaybuf + byteoff);
        *c &= (char) ~(1 << bitoff);
}

/* Lager en graf i globals.displaybuf av count samples fra start og utover
 * for sample-verdiene for sensor. graphtype bestemmer hvaslags graf det er
 * snakk om.
 */
void make_sample_graph(uint32_t start, uint16_t count, uint8_t sensor,
                uint8_t graphtype ) {
        uint16_t i,j,k,colw;
        uint8_t extra,sample,height;

        colw = count/DISPLAY_WIDTH;
        extra = (uint8_t) count/DISPLAY_WIDTH;

        /* Beregne soeylehoeyde */
        for (i=0;i<DISPLAY_WIDTH;i++) {
                if(graphtype == GT_LOW)
                        height = 255;
                else {
                        k = 0;
                        height = 0;
                }
                for(j=0;j<colw;j++) {
                        sample=sample_getvalue(sensor,start+i*colw+j);
                        switch(graphtype) {
                                case GT_LOW:
                                        if(sample<height)
                                                height = sample;
                                        break;
                                case GT_MED:
                                        k += (uint16_t) sample;
                                        break;
                                case GT_HIGH:
```

```
                                        if(sample>height)
                                                height = sample;
                                        break;
                                }
                        }
                        if (graphtype == GT_MED)
                                height = (uint8_t) k/colw;
                        for(j=0;j < (height/32);j++)
                                displaybuf_setpixel(i,j);

                }
                display_show(globals.displaybuf);
}

void time_show(void) {
        char *buf;

        while(keyboard_getchar() == -1) {
                display_clear();
                display_position(0,0);
                display_puts_P(*(menustrings + 29));
                display_position(8,1);
                buf = (char *) malloc(12);
                sprintf(buf,"␣%li",time_get());
                display_puts(buf);
                delay(50);
        }
        globals.refresh_menu = 1;
        free(buf);
        return;
}

uint32_t time_get(void) {
        return 31337;
}

void time_prompt(void) {
        time_set(prompt_integer(30));
}

void time_set(uint32_t new) {}

void sensor_prompt(void) {
        globals.graphparams.sensor = (uint8_t) prompt_integer(31);
}

void sensor_show_time(void) {
        uint32_t time;
        uint16_t data;
        char *buf;

        buf = (char *) malloc(12);
        time = prompt_integer(7);
        data = sample_getvalue(globals.graphparams.sensor, time);
        sprintf(buf,"␣%i",data);
        display_clear();
        display_position(0,1);
        display_puts(buf);
        free(buf);
        globals.refresh_menu = 1;
        while(keyboard_getchar() == -1);
}

void sensor_show_now(void) {
        uint16_t data;
        char *buf;

        buf = (char *) malloc(12);

        while(keyboard_getchar() == -1) {
                sprintf(buf,"␣%i", globals.graphparams.sensor);
                display_clear();
                display_position(0,0);
                display_puts_P(*(menustrings + 33));
                display_position(8,0);
                display_puts(buf);
                display_position(2,1);
                data = sample_getvalue(globals.graphparams.sensor, 0xffffffff);
                sprintf(buf,"␣%i",data);
                display_puts(buf);
                delay(50);
        }
        globals.refresh_menu = 1;
        free(buf);
        return;
}

int16_t sample_getvalue(uint8_t sensor, uint32_t sampleid) {
```

```
        usb_clearbuf_out();

        globals.usb.outbuf[0] = 5;
        globals.usb.outbuf[1] = 1;
        usb_outbuf_adduint32(sampleid, 2);
        globals.usb.outbuf[6] = sensor;

        usb_sendbuf();
        /* Vi forventer at neste pakke er svaret. Vi vil jo bare faa pakker
         * tilbake naar vi har spurt om noe, og vi er tross alt ikke multithreaded
         */
        usb_receivebuf();
        if(!((globals.usb.inbuf[1] & 0x7f) == 1)) {
                display_puts_P(PSTR("usb: unexpected packet\n"));
                return 0;
        }

        return usb_inbuf_getuint16(2);
}

uint32_t sample_getid(uint8_t sensor, uint32_t timestamp) {
        usb_clearbuf_out();

        globals.usb.outbuf[0] = 5;
        globals.usb.outbuf[1] = 2;
        usb_outbuf_adduint32(timestamp, 2);
        globals.usb.outbuf[6] = sensor;

        usb_sendbuf();
        /* Vi forventer at neste pakke er svaret. Vi vil jo bare faa pakker
         * tilbake naar vi har spurt om noe, og vi er tross alt ikke multithreaded
         */
        usb_receivebuf();
        if(!((globals.usb.inbuf[1] & 0x7f) == 2)) {
                display_puts_P(PSTR("usb: unexpected packet\n"));
                return 0;
        }

        return usb_inbuf_getuint32(2);
}


void connections_show(void) { }

void flash_status_show(void) {
        uint32_t data;
        char *buf;

        buf = (char *) malloc(12);
        display_clear();
        display_position(0,0);
        display_puts_P(*(menustrings + 34));

        display_position(0,1);
        display_puts_P(*(menustrings + 35));
        data = flash_free_get();
        sprintf(buf," %i",data);
        display_puts(buf);
        display_puts_P(*(menustrings + 37));

        display_position(0,2);
        display_puts_P(*(menustrings + 36));
        data*=100;
        data/=FLASH_SIZE;
        sprintf(buf," %i",data);
        display_puts(buf);
        display_puts_P(PSTR("%"));
        free(buf);
        globals.refresh_menu = 1;
        while(keyboard_getchar() == -1);
}

uint32_t flash_free_get() {
        return 800;
}

void sample_interval_prompt(void) {
        sample_interval_set(prompt_integer(32));
}

void sample_interval_set(uint32_t new) {
        usb_clearbuf_out();

        globals.usb.outbuf[0] = 4;
        globals.usb.outbuf[1] = 4;
        usb_outbuf_adduint32(new, 2);

        usb_sendbuf();
```

```
        usb_receievebuf();
        if(!((globals.usb.inbuf[1] & 0x7f) == 4)) {
                display_puts_P(PSTR("usb: unexpected packet\n"));
                return 0;
        }

        display_clear();
        display_position(0,0);
        display_puts_P(*(menustrings + 38));
        while(keyboard_getchar() == -1);
};

uint32_t sample_interval_get() {
        usb_clearbuf_out();

        globals.usb.outbuf[0] = 0;
        globals.usb.outbuf[1] = 11;

        usb_sendbuf();
        usb_receivebuf();
        if(!((globals.usb.inbuf[1] & 0x7f) == 4)) {
                display_puts_P(PSTR("usb: unexpected packet\n"));
                return 1;
        }

        return usb_inbuf_getuint32(2);
}


void sensor_mapping(void) { }

void sensor_maptype(uint8_t sensor, uint8_t type) {
        usb_clearbuf_out();

        globals.usb.outbuf[0] = 2;
        globals.usb.outbuf[1] = 9;
        globals.usb.outbuf[2] = sensor;
        globals.usb.outbuf[3] = type;

        usb_sendbuf();
        usb_receivebuf();
        if(!((globals.usb.inbuf[1] & 0x7f) == 9))
                display_puts_P(PSTR("usb: unexpected packet\n"));
        return;
}

void setenv_gp_start(void) {
        globals.graphparams.start = prompt_integer(27);
}

void setenv_gp_stop(void) {
        globals.graphparams.stop = prompt_integer(28);
}

void setenv_gp_gt_hi(void) {
        globals.graphparams.graphtype = GT_HIGH;
}

void setenv_gp_gt_med(void) {
        globals.graphparams.graphtype = GT_MED;
}

void setenv_gp_gt_lo(void) {
        globals.graphparams.graphtype = GT_LOW;
        return;
}

void graph_show(void) { }
```

Listing 35: Displaykort/display.h

```
#ifndef __DISPLAY_H
#define __DISPLAY_H

#define GT_LOW 0
#define GT_MED 1
#define GT_HIGH 2

#define DISPLAY_HEIGHT 32
#define DISPLAY_WIDTH 120
#define DISPLAY_BUFSIZE 480

#define MAX_MENU_ITEMS 12

#define KEYBOARD_BUFFER_SIZE 8

```

```
#define FLASH_SIZE 1000

#define KEYSYM_0 0x0
#define KEYSYM_1 0x1
#define KEYSYM_2 0x2
#define KEYSYM_3 0x3
#define KEYSYM_4 0x4
#define KEYSYM_5 0x5
#define KEYSYM_6 0x6
#define KEYSYM_7 0x7
#define KEYSYM_8 0x8
#define KEYSYM_9 0x9
#define KEYSYM_A 0xA
#define KEYSYM_B 0xB
#define KEYSYM_C 0xC
#define KEYSYM_D 0xD
#define KEYSYM_HASH 0xE
#define KEYSYM_STAR 0xF

#define min(x,y) (y < x ? y : x)

/* globale variable og typedefinisjoner */

struct Menuitem {
        void (* func)();
        struct Menu *submenu;
        uint16_t msgid;
        uint8_t key;
};

struct Menu {
        uint16_t msgid;
        uint8_t itemc;
        struct Menuitem item[MAX_MENU_ITEMS+1];
};

struct {
        struct {
                char inbuf[132];
                char outbuf[132];
        } usb;
        char displaybuf[DISPLAY_BUFSIZE+1];
        volatile char keybbuf[KEYBOARD_BUFFER_SIZE+1];
        volatile uint8_t keybbufstart;
        volatile uint8_t keybbufend;
        volatile int8_t refresh_menu;
        struct {
                uint8_t graphtype;
                uint8_t sensor;
                uint32_t start;
                uint32_t stop;
                uint16_t length;
        } graphparams;
} globals;

char **menustrings;

/* funksjonssignaturer */

int16_t sample_getvalue(uint8_t, uint32_t);
uint32_t sample_getid(uint8_t, uint32_t);

void display_putc(char);
void display_puts(char *);
void display_puts_P(const char *);
void display_show(char *);
void display_clear();
void display_bl_on();
void display_bl_off();
void display_position(uint8_t, uint8_t);
void display_uart_init();
void display_configure();
void debug_uart_init();
void displaybuf_setpixel(uint8_t, uint8_t);
void displaybuf_unsetpixel(uint8_t, uint8_t);

void make_sample_graph(uint32_t, uint16_t, uint8_t, uint8_t);

int8_t keyboard_getchar(void);
void keyboard_ungetchar(int8_t);

void menu_show(struct Menu *);

void prompt_uint8ptr(uint8_t *, uint16_t);
void prompt_uint16ptr(uint16_t *, uint16_t);
void prompt_uint32ptr(uint32_t *, uint16_t);

void sensor_prompt(void);
```

```
void sensor_show_time(void);
void time_show(void);
void time_prompt(void);
void time_set(uint32_t);
uint32_t time_get(void);
void sensor_show_now(void);
void connections_show(void);
void flash_status_show(void);
uint32_t flash_free_get(void);
void sample_interval_prompt(void);
void sample_interval_set(uint32_t);
void sensor_mapping(void);
void setenv_gp_start(void);
void setenv_gp_stop(void);
void setenv_gp_gt_hi(void);
void setenv_gp_gt_med(void);
void setenv_gp_gt_lo(void);
void graph_show(void);
void language_english(void);
void language_norwegian(void);
void language_german(void);
void language_french(void);
void language_quenya(void);
#endif
```

Listing 36: Displaykort/menustrings.h

```
#ifndef __MENUSTRINGS_H
#define __MENUSTRINGS_H

const char HH0000[] PROGMEM = "Main menu";
const char HH0001[] PROGMEM = "Debugging";
const char HH0002[] PROGMEM = "Status";
const char HH0003[] PROGMEM = "Reports";
const char HH0004[] PROGMEM = "Settings";
const char HH0005[] PROGMEM = "Select Sensor";
const char HH0006[] PROGMEM = "Graphs";
const char HH0007[] PROGMEM = "Time";
const char HH0008[] PROGMEM = "Read sensor";
const char HH0009[] PROGMEM = "Show connections";
const char HH0010[] PROGMEM = "Mem stat";
const char HH0011[] PROGMEM = "Sample interval";
const char HH0012[] PROGMEM = "Language";
const char HH0013[] PROGMEM = "Sensorconf";
const char HH0014[] PROGMEM = "Point in time";
const char HH0015[] PROGMEM = "Graphtype";
const char HH0016[] PROGMEM = "Start";
const char HH0017[] PROGMEM = "Stop";
const char HH0018[] PROGMEM = "Display";
const char HH0019[] PROGMEM = "English";
const char HH0020[] PROGMEM = "German";
const char HH0021[] PROGMEM = "Norwegian";
const char HH0022[] PROGMEM = "French";
const char HH0023[] PROGMEM = "Elvish";
const char HH0024[] PROGMEM = "Interval maximum";
const char HH0025[] PROGMEM = "Interval average";
const char HH0026[] PROGMEM = "Interval minimum";
const char HH0027[] PROGMEM = "Enter start time";
const char HH0028[] PROGMEM = "Enter stop time";
const char HH0029[] PROGMEM = "Current time is";
const char HH0030[] PROGMEM = "Enter new time";
const char HH0031[] PROGMEM = "Enter active sensor";
const char HH0032[] PROGMEM = "Enter new sampling interval";
const char HH0033[] PROGMEM = "Sensor:";
const char HH0034[] PROGMEM = "Memory status";
const char HH0035[] PROGMEM = "Free space:";
const char HH0036[] PROGMEM = "Percentage used:";
const char HH0037[] PROGMEM = "bytes";
const char HH0038[] PROGMEM = "Acknowledged";


const char *menustrings_en[] = {
        HH0000, HH0001, HH0002, HH0003, HH0004, HH0005,
        HH0006, HH0007, HH0008, HH0009, HH0010, HH0011,
        HH0012, HH0013, HH0014, HH0015, HH0016, HH0017,
        HH0018, HH0019, HH0020, HH0021, HH0022, HH0023,
        HH0024, HH0025, HH0026, HH0027, HH0028, HH0029,
        HH0030, HH0031, HH0032, HH0033, HH0034, HH0035,
        HH0036, HH0037, HH0038
};

const char NO0000[] PROGMEM = "Hovedmeny";
const char NO0001[] PROGMEM = "Debugging";
const char NO0002[] PROGMEM = "Status";
const char NO0003[] PROGMEM = "Rapporter";
const char NO0004[] PROGMEM = "Instillinger";
```

```
const char NO0005[] PROGMEM = "Velg Sensor";
const char NO0006[] PROGMEM = "Grafer";
const char NO0007[] PROGMEM = "Tid";
const char NO0008[] PROGMEM = "Les sensor";
const char NO0009[] PROGMEM = "Vis tilkoblinger";
const char NO0010[] PROGMEM = "Minnestat";
const char NO0011[] PROGMEM = "Samplingsinterval";
const char NO0012[] PROGMEM = "Spraak";
const char NO0013[] PROGMEM = "Sensorkonf";
const char NO0014[] PROGMEM = "Tidspunkt";
const char NO0015[] PROGMEM = "Graftype";
const char NO0016[] PROGMEM = "Start";
const char NO0017[] PROGMEM = "Stopp";
const char NO0018[] PROGMEM = "Vis";
const char NO0019[] PROGMEM = "Engelsk";
const char NO0020[] PROGMEM = "Tysk";
const char NO0021[] PROGMEM = "Norsk";
const char NO0022[] PROGMEM = "Fransk";
const char NO0023[] PROGMEM = "Elvish";
const char NO0024[] PROGMEM = "Interval maximum";
const char NO0025[] PROGMEM = "Interval average";
const char NO0026[] PROGMEM = "Interval minimum";
const char NO0027[] PROGMEM = "Enter start time";
const char NO0028[] PROGMEM = "Enter stop time";
const char NO0029[] PROGMEM = "Current time is";
const char NO0030[] PROGMEM = "Enter new time";
const char NO0031[] PROGMEM = "Enter active sensor";
const char NO0032[] PROGMEM = "Enter new sampling interval";
const char NO0033[] PROGMEM = "Sensor:";
const char NO0034[] PROGMEM = "Memory status";
const char NO0035[] PROGMEM = "Free space:";
const char NO0036[] PROGMEM = "Percentage used:";
const char NO0037[] PROGMEM = "bytes";
const char NO0038[] PROGMEM = "OK";

const char *menustrings_no[] = {
        NO0000, NO0001, NO0002, NO0003, NO0004, NO0005,
        NO0006, NO0007, NO0008, NO0009, NO0010, NO0011,
        NO0012, NO0013, NO0014, NO0015, NO0016, NO0017,
        NO0018, NO0019, NO0020, NO0021, NO0022, NO0023,
        NO0024, NO0025, NO0026, NO0027, NO0028, NO0029,
        NO0030, NO0031, NO0032, NO0033, NO0034, NO0035,
        NO0036, NO0037, NO0038
};
#endif
```

Listing 37: Displaykort/usb.c

```
#include <stdio.h>
#include <inttypes.h>
#include <avr/io.h>
#include <avr/pgmspace.h>
#include "display.h"

void usb_fragment_send(const char *packet) {

}

void usb_sendbuf() {
        uint8_t len,pos;
        char xor;
        char *buf;

        len=atoi(globals.usb.outbuf[0] & 0x7f);

        if(len>127) {
                display_puts_P(PSTR("usb: overflow"));
                return;
        }

        pos=0;
        globals.usb.outbuf[0] &= 0x7f;

        buf=(char *) malloc(9);

        while(pos < len+3) {
                memcpy(buf,&globals.usb.outbuf[pos],min(8,len-pos));
                if(pos!=0)
                        buf[0] = pos;
                /* Ja, dette betyr at vi i neste iterasjon vil ha et overlapp
                 * paa en byte. Dette er imidlertid med hensikt, da payload-
                 * offseten uansett vil overskrive dette.
                 */
                pos+=7;
                usb_fragment_send(buf);
        }
```

```
        free(buf);
}

uint8_t usb_fragment_reassemble(const char *packet) {
        uint8_t i,j;

        i = (uint8_t) packet[0];
        if(!(i & 80)) {
                memcpy(globals.usb.inbuf, packet, min(i,8));
                return 0;
        }

        i &= 0x7f;
        j = globals.usb.inbuf[0];

        memcpy((globals.usb.inbuf + i), packet+1, min(j,8));
        return 1;
}

void usb_receivebuf() {
        char *buf;
        uint8_t pos;

        buf = (char *) malloc(9);
        pos = 0;

        do {

                buf[i++] = PORTA;
                pos = 0;
        } while(usb_fragment_reassemble(buf))

        free(buf);
}

void usb_inbuf_clear() {
        memset(globals.usb.inbuf,0,132);
}

void usb_outbuf_clear() {
        memset(globals.usb.outbuf,0,132);
}

void usb_outbuf_adduint32(uint32_t i, uint16_t offset) {
        globals.usb.outbuf[offset]   = (char) ((i & 0xff000000) >> 24);
        globals.usb.outbuf[offset+1] = (char) ((i & 0x00ff0000) >> 16);
        globals.usb.outbuf[offset+2] = (char) ((i & 0x0000ff00) >> 8);
        globals.usb.outbuf[offset+3] = (char)  (i & 0x000000ff);
}

uint32_t usb_inbuf_getuint32(uint16_t offset) {
        uint32_t i;
        i = globals.usb.inbuf[offset]   << 24;
        i+= globals.usb.inbuf[offset+1] << 16;
        i+= globals.usb.inbuf[offset+2] << 8;
        i+= globals.usb.inbuf[offset+3];
        return i;
}


uint16_t usb_inbuf_getuint16(uint16_t offset) {
        uint16_t i;
        i = globals.usb.inbuf[offset+2] << 8;
        i+= globals.usb.inbuf[offset+3];
        return i;
}
```

F.4 Displaykort - FPGA

Listing 38: Displaykort/controller.vhd

```
-- USB function controller
-- DMPROSJ 2003

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity usb_controller is
        port (
                -- transmit
                out_clk : out STD_LOGIC;
```

```
                input: in STD_LOGIC_VECTOR (7 downto 0); -- data input from microcontroller
                input_waiting: in STD_LOGIC; -- raised by the microcontroller when there is data on input

                input_read : out STD_LOGIC; -- signals that the microcontroller can put other data on the input
                vmo: out STD_LOGIC;
                vpo: out STD_LOGIC;
                out_oe: out STD_LOGIC;  -- output enable

                -- transceiver control
                mode: out STD_LOGIC; -- mode select
                suspend: out STD_LOGIC; -- low power mode

                -- receive
                rcv: in STD_LOGIC; -- data input from transceiver
                output_out: out STD_LOGIC_VECTOR (7 downto 0); -- data out to microcontroller
                output_waiting: out STD_LOGIC;   -- raised to signal to microcontroller that we have received data on the usb bus
                receive_packet: in STD_LOGIC; -- raised by the microcontroller to go to the expect_packet state
                crc_error: out STD_LOGIC;
                vp: in STD_LOGIC; -- detects errors and se0
                vm: in STD_LOGIC; --  "

                -- debug
                error_code_out: out STD_LOGIC_VECTOR (5 downto 0);
                counter_out: out STD_LOGIC_VECTOR (31 downto 0);
                seg_led_1: out STD_LOGIC_VECTOR (6 downto 0);   -- least significant digit
                seg_led_2: out STD_LOGIC_VECTOR (6 downto 0);   -- most significant digit
                out_out_byte_sent: out STD_LOGIC;


                -- system
                fast_clk: in STD_LOGIC; -- external clock
                rst: in STD_LOGIC -- bus reset signal
        );
end usb_controller;

architecture usb_controller of usb_controller is
component transmitter
        port (
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                input: in STD_LOGIC_VECTOR (7 downto 0);
                input_select: in STD_LOGIC_VECTOR(2 downto 0);
                pid_select: in STD_LOGIC_VECTOR(3 downto 0);

                enable: in STD_LOGIC;
                send_eop: in STD_LOGIC;
                vpo: out STD_LOGIC;
                vmo: out STD_LOGIC;
                out_byte_sent: out STD_LOGIC
        );
end component;

component receiver
        port (
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                input: in STD_LOGIC;
                enable_rec: in STD_LOGIC;
                output: out STD_LOGIC_VECTOR(7 downto 0);
                byte_received: out STD_LOGIC;

                pid_out: out STD_LOGIC_VECTOR(3 downto 0);
                start: out std_logic;
                handshake_pck: out STD_LOGIC;
                setup_transaction: out STD_LOGIC;
                crc16_state_is_zero: out STD_LOGIC
        );
end component;

component segment_decoder
        port (
                HEX: in STD_LOGIC_VECTOR(5 downto 0);
                LED: out STD_LOGIC_VECTOR(6 downto 0)
        );
end component;

        -- Pid constants
        constant OUT_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0001";
        constant IN_pid         : STD_LOGIC_VECTOR(3 downto 0) := "1001";
        constant SOF_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0101";
        constant SETUP_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1101";
        constant DATA0_pid      : STD_LOGIC_VECTOR(3 downto 0) := "0011";
        constant DATA1_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1011";
        constant ACK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0010";
        constant NAK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1010";
        constant STALL_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1110";
        constant PRE_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1100";
```

```
        -- Enable signals
        signal enable_transmitter: STD_LOGIC;
        signal enable_receiver: STD_LOGIC;
105
        -- Reset signal

        signal oe: STD_LOGIC;
        --signal rst: STD_LOGIC;
110     signal error_code: STD_LOGIC_VECTOR (5 downto 0);

        signal pid_to_send: STD_LOGIC_VECTOR(3 downto 0);
        signal pid_in: STD_LOGIC_VECTOR(3 downto 0);
        signal send_eop: STD_LOGIC;
115     signal out_byte_sent: STD_LOGIC;
        signal byte_received: STD_LOGIC;

        -- signals to detect crc error
        signal in_crc16_is_zero: STD_LOGIC;
120     signal in_crc5_is_zero: STD_LOGIC;

        -- Div signals
        signal HEX: STD_LOGIC_VECTOR (5 downto 0);
        signal LED: STD_LOGIC_VECTOR (6 downto 0);
125     signal vp_sig: std_logic;
        signal vm_sig: std_logic;

        -- Signals for counter
        signal count,reset_counter: std_logic;
130     signal counter : integer range 0 to 16;

        ----------------------------------------
        -- Signals for controller
        ----------------------------------------
135     signal sender_mult_select: STD_LOGIC_VECTOR (2 downto 0);
        signal crc_bytes_received: STD_LOGIC_VECTOR (1 downto 0);
        type controller_state_type is (start, expect_packet, receive_pid, receive_data, check_crc16, send_packet, send_pid, send_addr, send_data,
        signal controller_state, next_controller_state: controller_state_type;
        signal sop_received: STD_LOGIC;
140     signal handshake_pak: STD_LOGIC;
        signal PID_type : STD_LOGIC_VECTOR(3 downto 0);
        signal setup_transaction: STD_LOGIC;
        signal clk: STD_LOGIC;
        signal output: STD_LOGIC_VECTOR(7 downto 0);
145
        ----------------------------------------
        --Signals for PLL
        ----------------------------------------
        signal pll_last_input: STD_LOGIC;
150     signal last_output_sig: STD_LOGIC;
        signal pll_count: INTEGER;

        begin
                transmitter1: transmitter port map(rst, clk, input, sender_mult_select, pid_to_send, enable_transmitter, send_eop,vpo,vmo, out_byt
155             receiver1: receiver port map(rst,clk,rcv,enable_receiver,output,byte_received,pid_in, sop_received,handshake_pak,setup_transaction
                segment_decoder1: segment_decoder port map(HEX, LED);
                -- general routing
                out_clk <= clk;
                pid_to_send <= input(3 downto 0);
160             counter_out <= conv_std_logic_vector(counter,32);
                error_code_out <= error_code;
                --rst <= not in_rst;
                HEX <= error_code;
                seg_led_1 <= LED;
165             seg_led_2 <= "0000000";
                mode <= '1'; -- force se0 mode
                suspend <= '0'; -- no suspend
                out_oe <= not oe; -- active low
                vp_sig <= vp;
170             vm_sig <= vm;
                out_out_byte_sent <= out_byte_sent;

        ----------------------------------------
        -- Controller
175     ----------------------------------------
        controller_state_machine: process (clk,controller_state)

        begin
                enable_receiver <= '0';
180             enable_transmitter <= '0';
                send_eop <= '0';
                enable_receiver <= '0';
                crc_error <= '0';
                output_waiting <= '0';
185             output_out <= output;
                reset_counter <= '0';
                count <= '0';
                input_read <= '0';
```

```
                error_code <= "111111";
190
                case controller_state is
                        ----------------
                        -- Start state
                        ----------------
195                     when start =>
                                error_code <= "000001"; -- debug
                                reset_counter <= '1';
                                -- detect if the microcontroller wants to send
                                if input_waiting = '1' then
200                                     next_controller_state <= send_packet;
                                elsif receive_packet = '1' then
                                        next_controller_state <= expect_packet;
                                else
                                        next_controller_state <= start;
205                             end if;

                        ----------------------------
                        -- Receive states
                        ----------------------------
210                     when expect_packet =>
                                error_code <= "001000";
                                if sop_received = '1' then
                                        next_controller_state <= receive_pid;
                                        enable_receiver <= '1';
215                             else
                                        next_controller_state <= expect_packet;
                                end if;

                        when receive_pid =>
220                             error_code <= "001001";
                                if byte_received = '1' then
                                        if handshake_pak = '1' then
                                                case pid_in is
                                                        when ACK_pid =>
225                                                             null;      -- transaction completed
                                                        when NAK_pid =>
                                                                crc_error <= '1';
                                                                 -- resend
                                                        when STALL_pid =>
230                                                             crc_error <= '1';
                                                                -- resend
                                                        when others =>
                                                                crc_error <= '0';
                                                end case;
235                                             next_controller_state <= start;
                                        else
                                                next_controller_state <= receive_data; -- neccesarily data
                                        end if;
                                end if;
240                             enable_receiver <= '1';

                        when receive_data =>
                                error_code <= "001010";
                                if byte_received = '1' then
245                                     output_waiting <= '1';
                                        if counter < 8 then
                                                -- add buffer functionality here?
                                                next_controller_state <= receive_data;
                                                count <= '1';
250                                     else
                                                next_controller_state <= check_crc16;
                                        end if;
                                else
                                        output_waiting <= '0';
255                                     next_controller_state <= receive_data;
                                end if;
                                enable_receiver <= '1';

                        when check_crc16 =>
260                             error_code <= "001011";
                                if counter > 2 then
                                        reset_counter <= '1';
                                else
                                        reset_counter <= '0';
265                             end if;
                                output_waiting <= '1';
                                enable_receiver <= '1';
                                if byte_received = '1' then
                                        output_waiting <= '0';
270                                     if counter < 2 then
                                                next_controller_state <= check_crc16;
                                                count <= '1';
                                        else
                                                enable_receiver <= '0'; -- stop receiving bytes
275                                             if in_crc16_is_zero = '1' then
                                                        next_controller_state <= start;-- crc is ok
```

```
                                        else
                                                crc_error <= '1'; -- tell microcontrloller to discard the package
                                                next_controller_state <= expect_packet; -- there was a crc error, wait for retransmit
                                        end if;
                                end if;
                        end if;

                ------------------
                -- Send states
                ------------------
                when send_packet =>
                        enable_transmitter <= '1';
                        error_code <= "000010"; -- debug
                        -- start the sending of the sync field (1 byte)
                        next_controller_state <= send_pid;


                when send_pid =>
                        enable_transmitter <= '1';
                        error_code <= "000011"; -- debug
                        input_read <= '1';
                        if out_byte_sent = '1' then  -- check to see that the sync field has been sent
                                input_read <= '1';
                                -- handshake packet
                                if pid_to_send = ACK_pid then
                                        next_controller_state <= send_eop_state; -- token packet
                                elsif pid_to_send = IN_pid or pid_to_send = OUT_pid or pid_to_send = SETUP_pid then
                                        next_controller_state <= send_addr;
                                elsif pid_to_send = DATA0_pid or pid_to_send = DATA1_pid then
                                        next_controller_state <= send_data;
                                else
                                        next_controller_state <= start; -- something is seriously wrong
                                end if;
                        else
                                next_controller_state <= send_pid;
                        end if;

                when send_addr =>
                        crc_error <= out_byte_sent; -- debug
                        enable_transmitter <= '1';
                        error_code <= "000100"; -- debug
                        if out_byte_sent = '1' then
                                count <= '1';
                        end if;
                        if counter = 2 then
                                next_controller_state <= send_eop_state;
                        end if;

                when send_data =>
                        enable_transmitter <= '1';
                        error_code <= "000101"; -- debug
                        if out_byte_sent = '1' then
                                input_read <= '1';
                                if counter = 4 then
                                        next_controller_state <= send_crc16;
                                else
                                        count <= '1';
                                        next_controller_state <= send_data;
                                end if;
                                input_read <= '0';
                        else
                                next_controller_state <= send_data;
                                input_read <= '1';
                        end if;


                when send_crc16 =>
                        enable_transmitter <= '1';
                        error_code <= "000110"; -- debug
                        if counter > 2 then
                                reset_counter <= '1';
                        elsif (out_byte_sent = '1') then
                                count <= '1';
                                next_controller_state <= send_crc16;
                        elsif counter = 2 then
                                next_controller_state <= send_eop_state;
                        else
                                next_controller_state <= send_crc16;
                        end if;

                when send_eop_state     =>
                        enable_transmitter <= '1';
                        error_code <= "000111"; -- debug
                        next_controller_state <= send_eop_state;
                        crc_error <= out_byte_sent;
                        if (out_byte_sent = '1') then
                                enable_transmitter <= '0';
```

```
365                                     send_eop <= '1';
                                        next_controller_state <= start;
                                else
                                        enable_transmitter <= '1';
                                        send_eop <= '0';
370                             end if;
                end case;
        end process;

        do_count : process(clk,rst)
375     begin
                if (reset_counter = '1' or rst = '1') then
                        counter <= 0;
                elsif (rising_edge(clk) and count = '1') then
                        counter <= counter + 1;
380             end if;
        end process;

        CONTROLLER_STATE_change: process (clk, rst)
        begin
385             if rst = '1' then
                        controller_state <= start;
                elsif rising_edge(clk) then
                        controller_state <= next_controller_state;
                end if;
390     end process;


        SENDER_MULT_SELECT_assignment:
        sender_mult_select <=
395     "000" when (controller_state = send_pid) else    -- pid
        "001" when (controller_state = send_addr and counter = 0) else
        "010" when (controller_state = send_addr and counter = 1) else
        "011" when (controller_state = send_data and counter = 1) else
        "100" when (controller_state = send_crc16 and counter = 0) else
400     "101" when (controller_state = send_crc16 and counter = 1) else
        "111" when (controller_state = send_packet) else -- send sync field
        "000";

        OE_assignment: -- output enable to transceiver
405     oe <=
        '1' when (controller_state = send_packet) else
        '1' when (controller_state = send_pid) else
        '1' when (controller_state = send_addr) else
        '1' when (controller_state = send_data) else
410     '1' when (controller_state = send_crc16) else
        '1' when (controller_state = send_eop_state) else
        '0';

        ------------------------------------
415     -- PLL and clock sync
        ------------------------------------
        PLL:
        process (fast_clk, rst, rcv)
                variable last_output: STD_LOGIC;
420     begin
                if rst = '1' then
                        pll_last_input <= '0';
                        last_output_sig <= '0';
                        pll_count <= 0;
425                     last_output := '0';
                elsif rising_edge(fast_clk) then
                        last_output := last_output_sig;

                        if not(pll_last_input = rcv) then
430                             last_output := not (last_output);
                                pll_count <= 0;
                        elsif pll_count = 7 then
                                last_output := not (last_output);
                                pll_count <= 0;
435                     else
                                pll_count <= pll_count + 1;
                        end if;
                end if;

440             pll_last_input <= rcv;
                last_output_sig <= last_output;
                clk <= last_output;
        end process;

445 end usb_controller;
```

Listing 39: Displaykort/receiver.vhd

```
-- USB function controller : receiver
-- DMPROSJ 2003
```

```
library IEEE;
use IEEE.std_logic_1164.all;
entity receiver is
        port (
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                input: in STD_LOGIC;
                enable_rec: in STD_LOGIC;

                output: out STD_LOGIC_VECTOR(7 downto 0);
                byte_received: out STD_LOGIC;
                pid_out: out STD_LOGIC_VECTOR(3 downto 0);
                start: out std_logic;
                handshake_pck: out STD_LOGIC;
                setup_transaction: out STD_LOGIC;
                crc16_state_is_zero: out STD_LOGIC
        );
end receiver;

architecture receiver of receiver is

-- Pid constants
        constant OUT_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0001";
        constant IN_pid         : STD_LOGIC_VECTOR(3 downto 0) := "1001";
        constant SOF_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0101";
        constant SETUP_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1101";
        constant DATA0_pid      : STD_LOGIC_VECTOR(3 downto 0) := "0011";
        constant DATA1_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1011";
        constant ACK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0010";
        constant NAK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1010";
        constant STALL_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1110";
        constant PRE_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1100";

        signal last_input: STD_LOGIC;
        signal stuffed_data: STD_LOGIC;
        signal bit_count: INTEGER; --Keeps track of consecutive 1's.
        signal data: STD_LOGIC_VECTOR (7 downto 0);
        signal shift_count: INTEGER; --Keeps track of number of bits

        --Signals for multiplexor.
        signal out_byte: STD_LOGIC_VECTOR(7 downto 0);

        --Signals for pid checker.
        signal pid: STD_LOGIC_VECTOR(7 downto 0);

        --Signals for start of packet detector.
        type state_type is (s0,s1,s2,s3,s4,s5,s6,s7,s8);
        signal state,next_state:state_type;

        --Signals for CRC calculators.
        signal crc16_next_state: STD_LOGIC_VECTOR(15 downto 0);
        signal crc16_state: STD_LOGIC_VECTOR(15 downto 0);
        signal crc5_next_state: STD_LOGIC_VECTOR(4 downto 0);
        signal crc5_state: STD_LOGIC_VECTOR(4 downto 0);

        begin

        pid <= out_byte;
        output <= out_byte;

        sop_detector:
        process(state, input)
        begin
                case state is
                        when s0=>if input = '0' then next_state <= s1;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s1=>if input = '1' then next_state <= s2;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s2=>if input = '0' then next_state <= s3;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s3=>if input = '1' then next_state <= s4;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s4=>if input = '0' then next_state <= s5;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s5=>if input = '1' then next_state <= s6;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s6=>if input = '0' then next_state <= s7;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s7=>if input = '0' then next_state <= s8;start <= '0';
                                else next_state <= s0;start <= '0';end if;
                        when s8=>next_state <= s0;start <= '1';
                end case;
        end process;

        ------------------------------------------
        --State updater for start of packet detector.
        ------------------------------------------
```

```
process(clk, rst, next_state)
begin
        if rst = '1' then
                state <= s0;
        elsif rising_edge(clk) then
                state <= next_state;
        end if;
end process;

main_process:
process (clk, rst, input)
        variable tmp_stuffed_data: STD_LOGIC;
        variable tmp_data: STD_LOGIC_VECTOR(7 downto 0);
        variable tmp_bit_count: INTEGER;
        variable tmp_shift_count: INTEGER; --Keeps track of number of bits
begin
        if rst = '1' then
                last_input <= '0';
                stuffed_data <= '0';
                bit_count <= 0;
                data <= "00000000";
                shift_count <= 0;
                byte_received <= '0';
                out_byte <= "00000000";

        elsif clk'EVENT and clk = '1' then
                if enable_rec = '1' then

                        -- push signals onto variables
                        tmp_stuffed_data := stuffed_data;
                        tmp_data := data;
                        tmp_bit_count := bit_count;
                        tmp_shift_count := shift_count;

                        ----------------
                        -- NRZI Decoder--
                        ----------------

                        if (input = last_input) then -- input is rcv from controller
                                tmp_stuffed_data := '1';          -- if no change in signal
                        else
                                tmp_stuffed_data := '0';          -- if change in signal
                        end if;

                        ------------------------
                        -- Stuffed Bit Remover--
                        ------------------------
                        -- Keep count of of number of 1's.
                        -- This info will be used to remove stuffed bit.
                        if tmp_stuffed_data = '1' then
                                tmp_bit_count := tmp_bit_count + 1;    -- tmp_bit_count is number of 1's encountered
                        else
                                tmp_bit_count := 0;
                        end if;

                        --Only add input to the data register if it is not
                        --a stuffed bit.
                        -- tmp_stuffed_data is shifted into the LSB of tmp_data
                        if not(tmp_bit_count = 6) then
                                tmp_data(7) := tmp_data(6);
                                tmp_data(6) := tmp_data(5);
                                tmp_data(5) := tmp_data(4);
                                tmp_data(4) := tmp_data(3);
                                tmp_data(3) := tmp_data(2);
                                tmp_data(2) := tmp_data(1);
                                tmp_data(1) := tmp_data(0);
                                tmp_data(0) := tmp_stuffed_data;

                                tmp_shift_count := tmp_shift_count + 1;
                        else
                                --Set bit_count to 0 since stuffed bit has
                                --been removed.
                                tmp_bit_count := 0;
                        end if;

                        -- Let the outside world know if a byte has been received.
                        if tmp_shift_count = 8 then
                                -- the out_byte signal to tmp_data, and rise byte_received signal
                                byte_received <= '1';
                                out_byte <= tmp_data;
                                --Reset shift_count for next byte.
                                tmp_shift_count := 0;
                        else
                                byte_received <= '0';
                        end if;

                        -- set signals from variables
```

```
                                stuffed_data <= tmp_stuffed_data;
                                data <= tmp_data;
                                bit_count <= tmp_bit_count;
                                shift_count <= tmp_shift_count;
                                last_input <= input;
                        end if;
                end if;
        end process;


        pid_checker:
        process (rst, pid)
                begin
                        if rst = '1' then
                                pid_out <= SOF_pid;
                                setup_transaction <= '0';
                                handshake_pck <= '0';
                        --SOF
                        elsif pid = "10100101" then
                                pid_out <= SOF_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        --SETUP
                        elsif pid = "10110100" then
                                pid_out <= SETUP_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '1';
                        --OUT
                        elsif pid = "10000111" then
                                pid_out <= OUT_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        --IN
                        elsif pid = "10010110" then
                                pid_out <= IN_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        --DATA0
                        elsif pid = "11000011" then
                                pid_out <= DATA0_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        --DATA1
                        elsif pid = "11010010" then
                                pid_out <= DATA1_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        --ACK
                        elsif pid = "01001011" then
                                pid_out <= ACK_pid;
                                handshake_pck <= '1';
                                setup_transaction <= '0';
                        --NAK
                        elsif pid = "01011010" then
                                pid_out <= NAK_pid;
                                handshake_pck <= '1';
                                setup_transaction <= '0';
                        --STALL
                        elsif pid = "01111000" then
                                pid_out <= STALL_pid;
                                handshake_pck <= '1';
                                setup_transaction <= '0';
                        --FEIL
                        else
                                pid_out <= SOF_pid;
                                handshake_pck <= '0';
                                setup_transaction <= '0';
                        end if;
        end process;

        ------------------------
        --16-bit CRC calculator--
        ------------------------
        process(clk, rst, stuffed_data)
        begin
                if clk'EVENT and clk = '1' then
                        crc16_state <= crc16_next_state;
                        if crc16_state = "0000000000000000" then
                                crc16_state_is_zero <= '1';
                        else
                                crc16_state_is_zero <= '0';
                        end if;
                end if;
                if rst = '1' then
                        crc16_state <= "0000000000000000";
                        crc16_next_state <= "0000000000000000";
                else
                        crc16_next_state(15) <= crc16_state(14) xor stuffed_data xor crc16_state(15);
```

```
                        crc16_next_state(14) <= crc16_state(13);
                        crc16_next_state(13) <= crc16_state(12);
                        crc16_next_state(12) <= crc16_state(11);
                        crc16_next_state(11) <= crc16_state(10);
                        crc16_next_state(10) <= crc16_state(9);
                        crc16_next_state(9) <= crc16_state(8);
                        crc16_next_state(8) <= crc16_state(7);
                        crc16_next_state(7) <= crc16_state(6);
                        crc16_next_state(6) <= crc16_state(5);
                        crc16_next_state(5) <= crc16_state(4);
                        crc16_next_state(4) <= crc16_state(3);
                        crc16_next_state(3) <= crc16_state(2);
                        crc16_next_state(2) <= crc16_state(1) xor stuffed_data xor crc16_state(15);
                        crc16_next_state(1) <= crc16_state(0);
                        crc16_next_state(0) <= stuffed_data xor crc16_state(15);
                end if;
        end process;


end receiver;
```

Listing 40: Displaykort/segment_decoder.vhd

```
-------------------------
-- segment_decoder.vhd
-- Maps 6 bits vector to 7 segment led digit
-------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity segment_decoder is
    port (

                        HEX:    in      STD_LOGIC_VECTOR (5 downto 0);
                        LED:    out     STD_LOGIC_VECTOR (6 downto 0)

                );
end segment_decoder;

architecture Behavioral of segment_decoder is

begin

--HEX-to-seven-segment decoder
--
-- segment encoding
--       0
--      ---
--   5 |   | 1
--      ---   <- 6
--   4 |   | 2
--      ---
--       3

    with HEX SELect
    LED <= "0000110" when "000001",   --1
           "1011011" when "000010",   --2
           "1001111" when "000011",   --3
           "1100110" when "000100",   --4
           "1101101" when "000101",   --5
           "1111101" when "000110",   --6
           "0000111" when "000111",   --7
           "1111111" when "001000",   --8
           "1101111" when "001001",   --9
           "1110111" when "001010",   --A
           "1111100" when "001011",   --b
           "0111001" when "001100",   --C
           "1011110" when "001101",   --d
           "1111001" when "001110",   --E
           "1110001" when "001111",   --F
           "0111111" when others;   --0


end Behavioral;
```

Listing 41: Displaykort/transmitter.vhd

```
-- USB function controller : transmitter
-- DMPROSJ 2003

library IEEE;
```

```
use IEEE.std_logic_1164.all;

entity transmitter is
        port (
                -- in
                rst: in STD_LOGIC;
                clk: in STD_LOGIC;
                input: in STD_LOGIC_VECTOR (7 downto 0);
                input_select: in STD_LOGIC_VECTOR(2 downto 0);
                pid_select: STD_LOGIC_VECTOR(3 downto 0);
                enable: in STD_LOGIC;
                send_eop: in STD_LOGIC;

                -- out
                vpo: out STD_LOGIC;
                vmo: out STD_LOGIC;
                out_byte_sent: out STD_LOGIC
        );
end transmitter;

architecture transmitter of transmitter is

        -- Pid constants
        constant OUT_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0001";
        constant IN_pid         : STD_LOGIC_VECTOR(3 downto 0) := "1001";
        constant SOF_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0101";
        constant SETUP_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1101";
        constant DATA0_pid      : STD_LOGIC_VECTOR(3 downto 0) := "0011";
        constant DATA1_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1011";
        constant ACK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "0010";
        constant NAK_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1010";
        constant STALL_pid      : STD_LOGIC_VECTOR(3 downto 0) := "1110";
        constant PRE_pid        : STD_LOGIC_VECTOR(3 downto 0) := "1100";

        signal last_output: STD_LOGIC;
        signal stuffed_data: STD_LOGIC;
        signal bit_count: INTEGER RANGE 0 to 6;
        signal shift_count: INTEGER RANGE 0 to 8;
        signal Sreg: STD_LOGIC_VECTOR(7 downto 0);
        signal mult_output: STD_LOGIC_VECTOR(7 downto 0);
        signal to_buffer: std_logic;
        signal output: std_logic;

        -- Signals for CRC 16 calculator.
        signal state, next_state: std_logic_vector(15 downto 0);

        -- Signals for CRC 5 calculator.
        signal crc5_next_state: STD_LOGIC_VECTOR(4 downto 0);
        signal crc5_state: STD_LOGIC_VECTOR(4 downto 0);


        -- Signal for PID generator.
        signal pid: STD_LOGIC_VECTOR(7 downto 0);

        begin

        transceiver_interface:
        process(output, send_eop)
        begin
                if send_eop = '1' then
                        vpo <= '0';
                        vmo <= '0';
                else
                        if output = '1' then
                                vpo <= '1';
                                vmo <= '0';
                        else
                                vpo <= '0';
                                vmo <= '1';
                        end if;
                end if;
        end process;

        main:
        process(clk, rst)
                variable tmp_Sreg: STD_LOGIC_VECTOR(7 downto 0);
                variable tmp_stuffed_data: STD_LOGIC;
                variable tmp_shift_count: INTEGER RANGE 0 to 8;
                variable tmp_bit_count: INTEGER RANGE 0 to 6;
        begin
                if rst = '1' then
                        output <= '1';
                        last_output <= '0';
                        tmp_stuffed_data := '1';
                        tmp_bit_count := 0;
                        tmp_shift_count := 0;
                        tmp_Sreg := "00000000";
                        out_byte_sent <= '0';
```

```
                        stuffed_data <= tmp_stuffed_data;
                        shift_count <= tmp_shift_count;
                        bit_count <= tmp_bit_count;
                elsif clk'EVENT and clk = '1' then
                        out_byte_sent <= '0';
                        tmp_stuffed_data := stuffed_data;
                        tmp_shift_count := shift_count;
                        tmp_bit_count := bit_count;
                        tmp_Sreg := Sreg;
                        if tmp_shift_count = 0 then -- this is the first byte we are sending
                                tmp_Sreg := mult_output;
                        end if;
                        if enable = '1' then
                                if tmp_bit_count = 6 then
                                        tmp_stuffed_data := '0';
                                        tmp_bit_count := 0;
                                else
                                        tmp_stuffed_data := tmp_Sreg(7);
                                        tmp_Sreg := tmp_Sreg(6 downto 0) & '1';

                                        -- Only increment counter when normal bit is sent.
                                        tmp_shift_count := tmp_shift_count + 1;
                                end if;

                                -- Keep count of of number of 1's.
                                -- This info will be used to add stuffed bit.
                                if tmp_stuffed_data = '1' then
                                        tmp_bit_count := tmp_bit_count + 1;
                                else
                                        tmp_bit_count := 0;
                                end if;

                                if tmp_shift_count = 8 then
                                        tmp_shift_count := 0;
                                        out_byte_sent <= '1';
                                else
                                        out_byte_sent <= '0';
                                end if;
                        end if;

                        -- NRZI encoding
                        if tmp_stuffed_data = '0' then
                                last_output <= not(last_output);
                                output <= last_output;
                        end if;

                        -- Update signal values.
                        Sreg <= tmp_Sreg;
                        stuffed_data <= tmp_stuffed_data;
                        shift_count <= tmp_shift_count;
                        bit_count <= tmp_bit_count;
                end if;
        end process;

        ----------------------
        --5-bit CRC calculator--
        ----------------------
        process(clk, rst, crc5_state)
        begin
                if clk'EVENT and clk = '1' then
                        crc5_state <= crc5_next_state;
                        crc5_state <= crc5_next_state;
                end if;
                if rst = '1' then
                        crc5_state <= "00000";
                        crc5_state <= "00000";
                        crc5_next_state <= "00000";
                else
                        crc5_next_state(4) <= crc5_state(3);
                        crc5_next_state(3) <= crc5_state(2);
                        crc5_next_state(2) <= crc5_state(1) xor stuffed_data xor crc5_state(4);
                        crc5_next_state(1) <= crc5_state(0);
                        crc5_next_state(0) <= stuffed_data xor crc5_state(4);
                end if;
        end process;

        ----------------------
        --16-bit CRC calculator--
        ----------------------
        process(clk, rst, state)
        begin
                if clk'EVENT and clk = '1' then
                        state <= next_state;
                end if;
                if rst = '1' then
                        state <= "0000000000000000";
                        next_state <= "0000000000000000";
                else
```

```
                        next_state(15) <= state(14) xor stuffed_data xor state(15);
                        next_state(14) <= state(13);
                        next_state(13) <= state(12);
                        next_state(12) <= state(11);
                        next_state(11) <= state(10);
                        next_state(10) <= state(9);
                        next_state(9) <= state(8);
                        next_state(8) <= state(7);
                        next_state(7) <= state(6);
                        next_state(6) <= state(5);
                        next_state(5) <= state(4);
                        next_state(4) <= state(3);
                        next_state(3) <= state(2);
                        next_state(2) <= state(1) xor stuffed_data xor state(15);
                        next_state(1) <= state(0);
                        next_state(0) <= stuffed_data xor state(15);
                end if;
        end process;

        ------------------
        --PID Generator--
        ------------------
        process (pid_select)
        begin
                case pid_select is
                        -- ACK
                        when ACK_pid =>
                                pid <= "01001011";
                        -- SETUP
                        when SETUP_pid =>
                                pid <= "10110100";
                        -- Send DATA0 packet.
                        when others =>
                                pid <= "11000011";
                end case;
        end process;

        ------------------
        -- Multiplexor--
        ------------------
        process (input_select)
        begin
                case input_select is
                        when "000" =>
                                mult_output <= pid;
                        when "001" => -- first addr byte
                                mult_output <= input; -- first addr byte
                        when "010" => -- second addr byte
                                mult_output(7 downto 5) <= input(2 downto 0);
                                mult_output(4 downto 0) <= crc5_state;
                        when "100" =>
                                mult_output <= state(7 downto 0);
                        when "101" =>
                                mult_output <= state(15 downto 8);
                        when "011" =>
                                mult_output <= input; -- data
                        when "111" =>
                                mult_output <= "00000001";
                        when others =>
                                mult_output <= input; -- data
                end case;
        end process;
end transmitter;
```

Listing 42: Displaykort/usb_controller_vhdl_tb.vhd

```
-- Testbenk for testing av sending og mottak for usb_controller
-- Sender en pakke øfrst, og mottar åetterp

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY testbench IS
END testbench;

ARCHITECTURE behavior OF testbench IS

        COMPONENT usb_controller
        PORT(
        -- transmit
                out_clk: out STD_LOGIC;
                input: in STD_LOGIC_VECTOR (7 downto 0); -- data input from microcontroller
                input_waiting: in STD_LOGIC; -- raised by the microcontroller when there is data on input
                input_read: out STD_LOGIC; -- signals that the microcontroller can put other data on the input
```

```
                vmo: out STD_LOGIC;
                vpo: out STD_LOGIC;
                out_oe: out STD_LOGIC;  -- output enable

                -- transceiver control
                mode: out STD_LOGIC; -- mode select
                suspend: out STD_LOGIC; -- low power mode

                -- receive
                rcv: in STD_LOGIC; -- data input from transceiver
                output_out: out STD_LOGIC_VECTOR (7 downto 0); -- data out to microcontroller
                output_waiting: out STD_LOGIC;  -- raised to signal to microcontroller that we have received data on the usb bus
                receive_packet: in STD_LOGIC; -- raised by the microcontroller to go to the expect_packet state
                crc_error: out STD_LOGIC;
                vp: in STD_LOGIC; -- detects errors and se0
                vm: in STD_LOGIC; --  "

                -- debug
                error_code_out: out STD_LOGIC_VECTOR (5 downto 0);
                counter_out: out STD_LOGIC_VECTOR (31 downto 0);
                seg_led_1: out STD_LOGIC_VECTOR (6 downto 0);  -- least significant digit
                seg_led_2: out STD_LOGIC_VECTOR (6 downto 0);  -- most significant digit
                out_out_byte_sent: out STD_LOGIC;


                -- system
                fast_clk: in STD_LOGIC; -- external clock
                rst: in STD_LOGIC -- bus reset signal
        );
        END COMPONENT;

        SIGNAL input :  std_logic_vector(7 downto 0);
        SIGNAL input_waiting :  std_logic;
        SIGNAL input_read :  std_logic;
        SIGNAL receive_packet :  std_logic;
        SIGNAL rcv :  std_logic;
        SIGNAL fast_clk :  std_logic;
        SIGNAL rst :  std_logic;
        SIGNAL output_out :  std_logic_vector(7 downto 0);
        SIGNAL output_waiting :  std_logic;
        SIGNAL crc_error :  std_logic;
        SIGNAL vmo :  std_logic;
        SIGNAL vpo :  std_logic;
        SIGNAL vm :  std_logic;
        SIGNAL vp :  std_logic;
        SIGNAL out_out_byte_sent: std_logic;
        SIGNAL out_oe : std_logic;
        SIGNAL out_clk :  std_logic;
        SIGNAL error_code_out: std_logic_vector(5 downto 0);
        signal counter_out: STD_LOGIC_VECTOR (31 downto 0);
        signal seg_led_1: STD_LOGIC_VECTOR (6 downto 0);
        signal seg_led_2: STD_LOGIC_VECTOR (6 downto 0);
        signal END_SIM: BOOLEAN:=FALSE;
        signal rcv_reg: STD_LOGIC_VECTOR (103 downto 0):=
--"01010100010100001010101010101011010101000010100011101011101010101010101010101010101010101010";
-- start...|.......|.......|.......|start..|.......|.......|.......|.......|.......|.......|.......
  "01010100010011101010101010101011010101000010100011100111101010101010101011101010101000101000101010100011";

BEGIN

        uut: usb_controller PORT MAP(
                input => input,
                input_waiting => input_waiting,
                input_read => input_read,
                receive_packet => receive_packet,
                rcv => rcv,
                fast_clk => fast_clk,
                rst => rst,
                output_out => output_out,
                output_waiting => output_waiting,
                crc_error => crc_error,
                vmo => vmo,
                vpo => vpo,
                vm => vm,
                vp => vp,
                out_oe => out_oe,
                out_clk => out_clk,
                error_code_out => error_code_out,
                counter_out => counter_out,
                seg_led_1 => seg_led_1,
                seg_led_2=> seg_led_2,
                out_out_byte_sent => out_out_byte_sent
        );


        -- *** Test Bench - User Defined Section ***
        CLOCK_fast_clk : process
        begin
```

```
                if END_SIM = FALSE then
                        fast_clk <= '0';
                        wait for 20 ns;
                else
                        wait;
                end if;
                if END_SIM = FALSE then
                        fast_clk <= '1';
                        wait for 20 ns;
                else
                        wait;
                end if;
        end process;

        RESET: process
        begin
                rst <= '0';
                input_waiting <= '0';
                wait for 400 ns; --0 ps
                rst <= '1';
                wait for 600 ns;
                rst <= '0';
                wait for 800 ns; -- wait for clk to start
                input_waiting <= '1';
                wait for 800 ns;
                input_waiting <= '0';
                wait;
        end process;

        TRANSMIT: process (input_read)
        variable count: Integer := 0;
        begin
                if input_read'event and input_read = '1' then
                        case count is
                                when 0 =>
                                        input <= "00000011"; -- data0_pid
                                when 1 =>
                                        input <= "00000001";    -- data
                                when 2 =>
                                        input <= "00000011";    -- data
                                when 3 =>
                                        input <= "00000010";    -- data
                                when 4 =>
                                        input <= "10101010";    -- data
                                when others =>
                                        input <= "11111111";
                        end case;
                        count := count + 1;
                end if;
        end process;

        RECEIVE: process
        variable count: Integer := 0;
        begin
                receive_packet <= '0';
                rcv <= '0';
                wait for 50 us;
                receive_packet <= '1';
                wait for 1000 ns;
                while (count < rcv_reg'length) loop
                        rcv <= rcv_reg(rcv_reg'left);
                        rcv_reg <= (rcv_reg((rcv_reg'left - 1) downto 0) & '0');
                        count := count + 1;
                        wait for 640 ns;
                        receive_packet <= '0';
                end loop;
                wait for 10 us;
                END_SIM <= TRUE;
                wait;
        end process;

END behavior;

configuration TESTBENCH_FOR_usb_controller of testbench is
        for behavior
                for UUT : usb_controller
                        use entity work.usb_controller(usb_controller);
                end for;
        end for;
END TESTBENCH_FOR_usb_controller;
```