Håndtering af billeddata fra NOAA-vejrsatellitter

English title:
Handling Image Data from NOAA Weather Satellites

Eskil Binzer
Søren Sølling Christiansen

Håndtering af billeddata fra NOAA-vejrsatellitter
Vejledere: Jens Thyge Kristensen, IMM DTU og Gøsta Thuesen, Ørsted DTU.

1 Forord

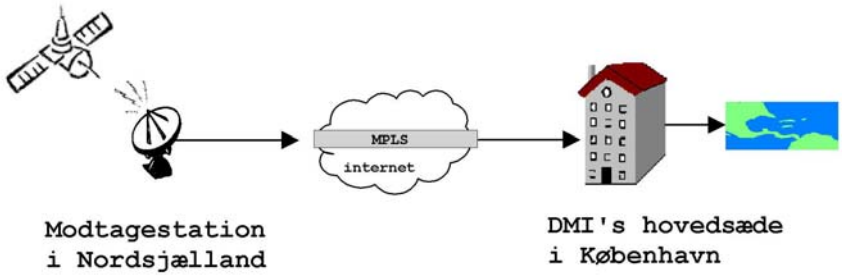
Denne rapport dokumenterer et civilingeniør afgangsprojekt ved DTU, udarbejdet i samarbejde med Danmarks Meteorologiske Institut. Projektet er påbegyndt den 13. februar 2006, og afleveret den 14 august 2006. Fra DMI skal der rettes tak til vores vejleder, Lars Erlind Bøthun, samt Ingolf Beck og Henrik Hartmann, for deres imødekommenhed og gode råd.

København, 14 august 2006

Eskil Binzer Søren Sølling Christiansen

2 Indledning

Danmarks Meteorologiske Institut (DMI) modtager vejrdata fra de amerikanske NOAA-vejrsatellitter. Der er for øjeblikket fire operationelle NOAA-satelitter (National Oceanic Atmosspheric Administraion) i kredsløb omkring jorden.
Modtagelsen foregår ved hjælp af en modtagestation som er placeret i Nordsjælland (Smidsbjerg), og data videresendes via en 2 Mbit MPLS-forbindelse (internetforbindelse med garanteret båndbredde og sikkerhed) til DMI's hovedsæde i København (Lyngbyvej) hvor billedbehandlingen foregår. Figur 1 illustrerer ovenstående.



Figur 1 Modtagelse af vejrdata i Nordsjælland og transmission til DMI's hovedsæde i København

Internettransmissionen varer ca. 5 minutter, mens en satellitpassage varer ca. 10 minutter. Da internettransmissionen først begynder når satellitpassagen er forbi, er satellitdata op til 15 minutter gammelt før DMI kan begynde at anvende det – denne tid vil DMI gerne have forkortet ved at transmittere løbende mens satellitten sender data til stationen i Nordsjælland.

Endvidere er modtageudstyret i Nordsjælland af ældre dato, hvilket betyder at det fylder meget. Det gør det besværligt at udskifte i tilfælde af nedbrud. DMI vil gerne have modtageudstyr der er lettere at håndtere / udskifte.

3 Kravspecifikation

3.1 Funktionalitet

- Konstruktionen på Smidbjerg skal visuelt indikere når der modtages brugbar NOAA-data.

Filer og information der skal produceres

- Først og fremmest skal den rå data lægges i NOAA-filer med formatet beskrevet i appendix a1. Disse filer skal navngives:
- NOAA-rammerne indeholder data fra 5 forskellige frekvensområder. Billeddata skal deles op i disse 5 kanaler og lægges i filer med formatet beskrevet i appendiks.
- Quickview billeder – filer hvor i man let kan se hvilke områder NOAA-filernes data dækker og hvilken vejrtype der er tale om. Det præcise format og udseende af filerne er ikke bestemt.
- Information om NOAA-data. DMI skal kunne læse oplysninger om af hvor mange bitfejl det er per Mbyte i hver NOAA-fil, om hvor mange rammer hver satellitpassage indeholder og om hvornår data stammer fra (første pixel i første ramme)

3.2 Hastighed

Rådatafil og kanalfilen skal være til rådighed "hurtigt" efter endt satellitpassage.
Hovedformålet med projektet er at spare den tid det tager at sende en NOAA-fil gennem et 2 Mbit netværk. Dvs. man sparer 5 min – derfor er det et krav at man ikke spilder den vundne tid et andet sted i forløbet.

3.3 Ydre rammer

3.3.1 Smidsbjerg

Konstruktionen skal monteres i et 19" rack chassis, med en højde på 1U.
DMI leverer en passende strømforsyning til montage i rack-chassiset.

Der er adgang til MPLS-netværket med en fast offentlig IP-addresse.
Der er ikke mulighed for fri adgang til internettet udenom MPLS – dvs. man kan kun regne med at kunne kontakte Lyngbyvej. Det fysiske interface er twisted pair med RJ45-konnektor.

NOAA-data tilgængelig via to signaltyper; et clock- og data-signal samt et BPSK-moduleret signal.
Clock-signalet er et firkantsignal med en frekvens på 665,4 kHz, og en duty cycle på 50%. Data-signalet er validt på både den faldende- og stigende flanke af clock-signalet. Begge signaler er TTL-kompatible.

BPSK-signalet har en frekvens på 665,4 kHz, og en amplitude på ±5V, symmetrisk omkring 0V.

Alle tre signaler fremføres i coax-kabler, med en BNC-konnektor (han) for enden. Udgangsimpedansen er på 75Ω.

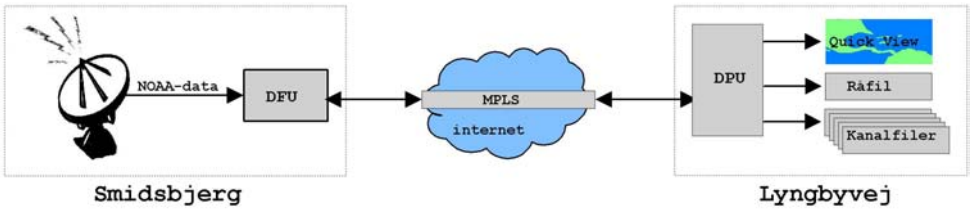
3.3.2 Lyngbyvej

Der bliver sat en server til rådighed hvorpå eventuel software kan installeres. Serveren kører Linux SUSE version 9.1 og DMI sørger for serverens vedligeholdelse.
Linuxserveren har adgang til stationen på Smidsbjerg gennem intenettet og føromtalte MPLS-netværk.
MPLS-netværket tilgås via en fast offentlig IP-addresse. Denne adresse hører ikke nødvendigvis eksklusivt til den benyttede Linux server, men det kan arrangeres at alt trafik på en aftalt port routes til den.
DMI sørger for adgang til opdaterede baneelementfiler for alle operationelle NOAA-satellitter samt satellitidentifikationsfil på lokalnetværket.

4 Forslag til overordnet løsning

4.1 Opdeling af opgaver mellem Smidsbjerg og Lyngbyvej

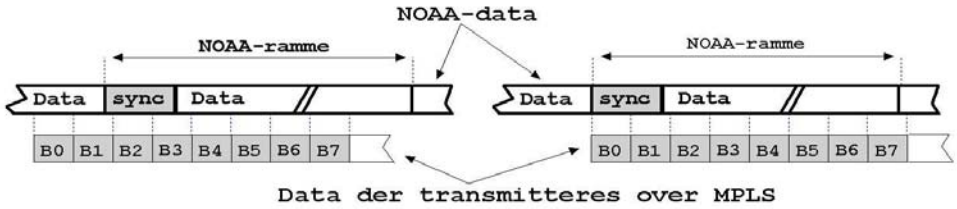
DMI ønsker hurtigst mulig adgang til deres vejrdata på Lyngbyvej, og man har adgang til et digitalt signal på en station på Smidsbjerg, samt en 2 Mbit MPLS-forbindelse mellem Smidsbjerg og Lyngbyvej.
Dvs. der skal konstrueres en enhed på Smidsbjerg og en enhed på Lyngbyvej, som skal kommunikere indbyrdes via et MPLS netværk.
Enheden på Smidsbjerg kan kaldes en *Data Forwarding Unit* (DFU), og på Lyngbyvej, en *Data Processing Unit* (DPU). Figur 2 illustrerer situationen på Smidsbjerg og Lyngbyvej, samt deres forbindelse.



Figur 2 Situationen fra Smidsbjerg til Lyngbyvej

Meteorologerne ønsker adgang både til helt rå data lagt i filer, og til behandlede data, f.eks. kanalfilerne og quick view'et. De behandlede data fylder ca. dobbelt så meget som de rå data – dvs. behandlingen medfører ca. en fordobling af den samlede datamængde, som skal transmitteres. Transmissionstiden har stor betydning, så det kan altså ikke betale sig at foretage databehandlingen på Smidsbjerg.

Data fra satellitterne bør dog ikke sendes helt ubehandlet over internettet; Af flere grunde vil det være en fordel at strukturere data i Smidsbjerg før internettransmission.
Data fra satellitterne modtages i rammer (NOAA-rammer) som har en størrelse på 110,9 kbit. Hver ramme starter med 60 bit kendt data, kaldet rammesynkronisering ('sync' på Figur 3). Disse 60 bit er ens for alle rammer. Sandsynligheden for at netop denne bit-rækkefølge findes uden for en rammesynkronisering er ganske lille, 1 til 2^{60} per bit. Det svarer groft regnet til at der skulle opstå en falsk rammesynkronisering for hver 140 milliarder megabyte eller én falsk NOAA-ramme for hver 2 milliarder satellitpassager (da en satellitpassage genererer ca. 70 Mbyte data). Det er altså ikke et problem man behøver at tage hensyn til.



Figur 3 Transmission af NOAA-data over MPLS-netværk. Usynkroniseret t.v. - og synkroniseret t.h.

Figur 3 t.v. viser NOAA-data der transmitteres usynkroniseret over internettet, hvor B0 til B7 symboliserer de afsendte bytes, med B0 som den

første. Bemærk at NOAA-rammen i dette eksempel starter midt i en byte. En NOAA-ramme er defineret sådan det er datas placering i rammen som definerer dens betydning. Det er altså nødvendigt at synkronisere bytes med rammesynkronisering således at man, på Lyngbyvej, kan tælle bytes i stedet for at skulle tælle bit fra NOAA-rammens start (illustreret på Figur 3 t.h.).

To andre grunde til at udføre synkronisering før transmission:

- DMI ønsker at man på Smidsbjerg kan se om der modtages brugbar data. Den bedste måde at afgøre om der modtages brugbar data er ved detektere rammesynkroniseringen. Hvis man kan se at der detekteres rammesynkronisering seks gange i sekundet kan man med god sikkerhed sige at den modtagne data er korrekt. Dvs. det er nærliggende at synkronisere på Smidsbjerg, da der under alle omstændigheder skal detekteres rammesynkronisering her.

- På Smidsbjerg kan et kort tidsrum uden valid NOAA-data, tolkes som at satellitten er passeret ned under horisonten, og at man derfor kan lukke datafilen på Lyngbyvej og påbegynde databehandlingen. Hvis man der imod vælger at detektere data slut på Lyngbyvej risikerer man at et kortvarigt brud i internetforbindelsen fejlagtigt får samme effekt, dvs. at en ufuldstændig datafil lukkes og behandles.

For at løse opgaven tilfredsstillende må man stille nogle yderligere krav:

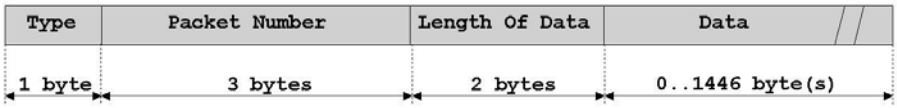
4.2 Krav til datatransmition

- Der må ikke opstå bitfejl i forbindelse med internettransmissionen – dvs. overførselsprotokollene skal have checksum osv.
- NOAA-protokollen er helt afhængig af at hver bit's placering i NOAA-rammen er korrekt så der må ikke være tab af datapakker eller ombytning – dvs. man er nød til at bruge TCP frem for UDP
- Håndtering af brud på internetforbindelse. Løsningen skal kunne håndtere at der opstår tab af internetforbindelse i kortere tidsrum, uden det giver anledning til tab af data. Ved længerevarende nedbrud (flere timer) må man forvente datatab.
- Transmissionshastighed skal være tilstrækkelig både til at følge med dataraten fra satellitten, og til samtidig at transmittere gammel data som pga. internetnedbrud ikke er blevet afsendt med det samme. En transmissionshastighed på ca. dobbelt NOAA-rate vil være acceptabel. Hastigheder på over 2 Mbit vil være unyttig idet den maksimale båndbredde DMI har til rådighed er 2 Mbit.

5 Kommunikation mellem DFU og DPU

Kommunikationen mellem DFU'en og DPU'en skal fastlægges.
Som nævnt i forrige afsnit skal det være muligt at indikere om NOAA-Data er synkroniseret efter framesync, samt om satellitpassagen er forbi.

Endvidere kunne det forestilles at DMI havde mere end én DFU placeret på et netværk. I så fald er det nødvendigt at kunne identificere DFU'en der afsender data. Figur 4 viser strukturen for DFU-DPU-protokollen, som den skal implementeres.



Figur 4 DFU-DPU-protokol.

Det er besluttet at reservere tre bytes til et evt. pakkenummer (eller andet nyttig information).

Type-feltet værdier og betydninger kan ses i nedenstående tabel.

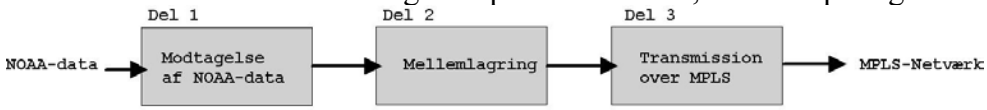
Værdi	Betydning
0x02	DFU-Identifikation(1)
0x05	NOAA-data
0x10	Rammesynkronisering og NOAA-data(2)
0x20	Sidste pakke med NOAA-data (passagen slut)

Tabel 1 Type-feltet s værdier og betydninger.

Note (1): Identifikationen består altid af seks bytes. Dvs. `Length Of Data` er altid seks.
`Length of data` angiver som navnet antyder, antal bytes i datafeltet (dvs. uden den resterende DFU-DPU-header).

6 Data Forwarding Unit (DFU)

I dette kapitel skal der udarbejdes en løsning til, hvordan DFU'en bedst implementeres.
Funktionaliteten i DFU'en kan groft opdeles i tre dele, som vist på Figur 5.



Figur 5 Opdeling af DFU'ens funktionalitet.

Del 1 skal kunne modtage NOAA-data, detektere og indikere rammesynkronisering. Endvidere skal data kunne skrives til mellemlageret (del 2), i tilfælde af at internetforbindelsen er afbrudt.
Del 3 skal transmittere data over internettet (MPLS-forbindelsen). Enten skal data læses fra mellemlageret (del 2), eller direkte fra del 1.

6.1 Vurdering af overordnede løsningsmuligheder

DMI kræver at DFU'en ikke fylder mere end et 1U 19" rackchassis. Med det in mente ses to typer løsninger:

1. Serverløsning: En rack-server i kombination med et interface til NOAA-data.
2. Stand Alone løsning: En løsning hvor der fremstilles ét PCB (Printed Circuit Board).

6.1.1 Serverløsning

Flere Computer-producenter, f.eks. Dell og IBM, fremstiller servere til montage i 19" Rack Chassiser, med en højde på 1U – netop som DMI kræver. En sådan server indeholder allerede mellemlagringsfunktionaliteten (ved brug af RAM eller harddisk), samt NOAA-data transmissionsfunktionaliteten over et MPLS-netværk (standart netværkskort). Dvs. de fysiske rammer for del 2 og del 3 på Figur 5 er til stede.
Dog mangler der et interface til modtagelse af NOAA-data fra satellitten (del 1 på Figur 5). Ifølge Dell's og IBM's websites leverer de kun 1U Rack-servere med udvidelseskonnektorer af typen PCI Express (PCIe) og/eller PCI-Extended (PCI-X). Dvs. interfacet skal være et PCIe- eller PCI-X-indstikskort. Sidst men ikke mindst skal der udvikles software til serveren, der kan læse data fra indstikskortet, og transmittere dette over internettet.

6.1.2 Stand Alone løsning

Ved en stand alone løsning forstås en konstruktion hvor de fornødne komponenter er placeret på ét PCB, til montage i et 1U 19" Rackchassis. Eksempelvis kan en mikrokontroller (MCU) forbundet med et lager (f.eks. RAM eller Flash RAM) udgøre NOAA-data modtagelsen og mellemlagringsdelen. Til datatransmission over MPLS-forbindelsen, kan der f.eks. vælges en netværkscontroller-IC, af samme slags som benyttes på et standart netværkskort til PC'ere.

6.1.3 Valg af løsning

Hvis serverløsningen vælges skal der udvikles:

- Modtagedelen til NOAA-data.
- PCIe / PCI-X interface.
- Software til serveren.

og hvis stand alone løsningen vælges, skal der udvikles hardwaredele:

- Modtagedelen til NOAA-data.
- Mellemlagringsdelen.
- Transmissionsdelen til internettet.

Stand alone løsningen ser, ud fra personlige erfaringer med udvikling af hardware, mest overkommelig ud. Hvis der ses bort fra modtagedelen til NOAA-data (som skal fremstilles uanset valg af løsning), så ser mellemlagringsdelen og transmissionsdelen begge overkommelige ud, og der er umiddelbart flere forskellige muligheder for valg af det endelige design.

Mht. serverløsningen, så er udvikling af et PCIe eller et PCI-X interface er en kompliceret affære. Det kræver stor præcision til fremstilling af PCB, da der er tale om busfrekvens på minimum 100MHz. De 100 MHz muliggør naturligvis stor overførselshastighed på bussen, men da NOAA-dataraten kun er 665,4 kbit per sekund, gavner det intet.

Igen, hvis stand alone løsningen vælges, vil DFU kunne fremstilles af relativ små, lette og billige komponenter, hvorimod serverløsningen vil blive dyrere og mere uhåndterbar (i kraft af serverens høje vægt).

Det vurderes at udviklingstiden vil være ens for de to løsninger, men også at stand alone løsningen vil være billigere og mere håndterbar. Derfor vælges det at DFU'en skal udvikles som en stand alone løsning.

I de efterfølgende afsnit vil løsningsmulighederne for hver enkelt del på Figur 5 blive diskuteret, og relevant teori blive belyst.

6.2 Løsningsmuligheder for modtagelse af NOAA-data

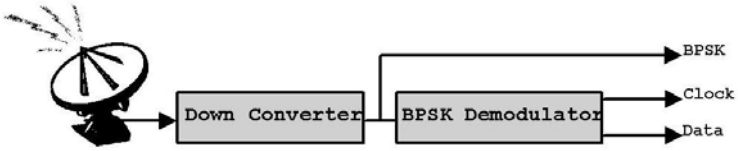
Der er to signaltyper med NOAA-data til rådighed for DFU'en:

1. To digitale signaler; Clock- og datasignal, med en frekvens på 665,4kHz. Spændingsniveauet på disser er 0V til 5V (TTL-niveau), og udgangsimpedansen er 75 Ω.
2. Et BPSK-moduleret signal, med en frekvens på 665,4 kHz – og en amplitude på mellem -5V og 5V, placeret symmetrisk omkring 0V.

Ifølge kravspecifikationen skal DFU'en kunne modtage NOAA-data vha. signaltype et. DMI er ved at udfase brugen af signaltype to (BPSK-signal), og derfor er det ikke påkrævet at kunne modtage dette.
DFU'en skal visuelt indikere når der modtages brugbar NOAA-data.

6.2.1 NOAA-data

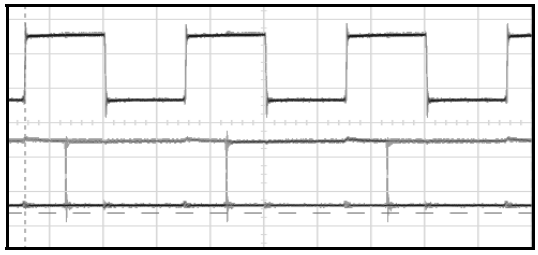
DMI modtager NOAA-data direkte fra satellitterne og nedkonverterer signalet til Binary Phase Shift Keying (BPSK), også kaldet et Manchester signal. Efterfølgende bitsynkroniseres (BPSK-demoduleres) dette signal, hvorved Clock og Data adskilles. Figur 6 viser signalvejen fra antenne til de tre signaler, der er til rådighed for DFU'en.



Figur 6 Signalvejen for NOAA-data

6.2.1.1 Clock og data

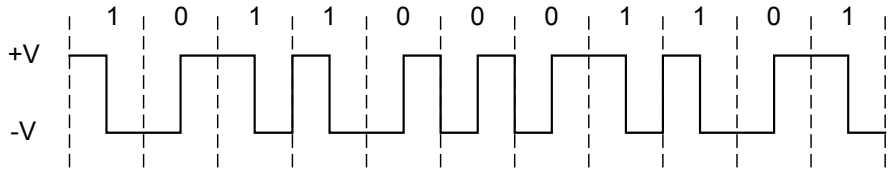
Figur 7 viser clock- og datasignalet (hhv. øverst og nederst), målt med et oscilloscope på DMI's testgenerator. Bemærk at data både kan detekteres på stigende- og falsende flanke.



Figur 7 Clock og data.

6.2.1.2 BPSK-signal

Manchester signalet, også kaldet Bi-ϕL (Bi-phase-Level), er et BPSK (Binary Phase Shift Keying) signal. Der skelnes mellem logisk ”nul” og logisk ”et” ved at se på niveauet i første (eller anden) halvdel af et bit.



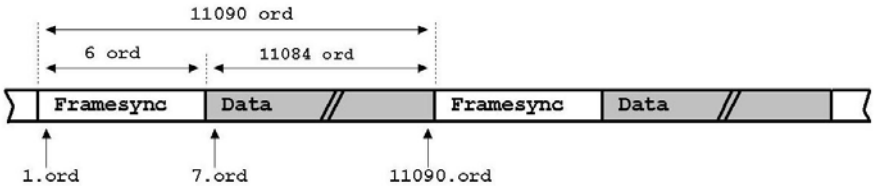
Figur 8 BPSK-signal.

Det har ikke været muligt at måle på DMI's testgenerator, da BPSK-udgangen er defekt.

6.2.2 Indikering af brugbar NOAA-data

DMI kræver en visuel indikation af , om der modtages brugbar NOAA-data (data som ikke er invalideret pga. støj). Grundlæggende betyder det, som forklaret i afsnit 4.1 (side 9), at DFU'en er nødsaget til at læse og behandle data, før det sendes videre, i modsætning til, bare at læse og sende det videre.

NOAA-data fra satellitten, sendes i "rammer" af 11090 ord, bestående af 10 bit hver, dvs. 110900 bit per ramme. En typisk satellitpassage (fra satellitten bliver synlig over horisonten, til den forsvinden under horisonten) indeholder typisk 5000 rammer.
I begyndelsen af hver ramme, er der 6 ord med kendt data (se Figur 9), kaldet rammesynkronisering (Framesync). Dvs. hvis DFU'en detekterer disse 60 bit skal det visuelt indikeres, da det er tegn på at der modtages brugbar data.



Figur 9 To NOAA-rammer.

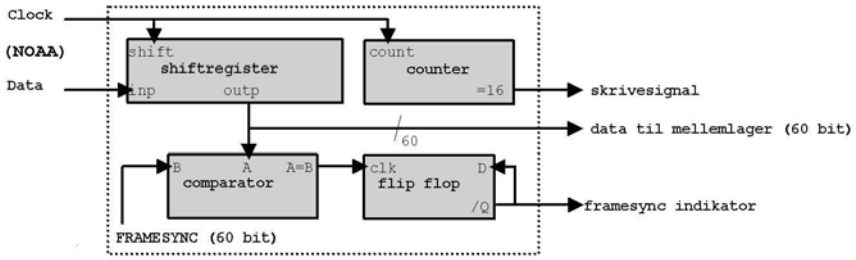
Visualiseringen kan enten ske med et LC-display (LCD), eller en lysdiode (LED). Spørgsmålet er hvilken form der er bedst. Ved brug af LCD er det muligt at præsentere en statistik over hvor mange framesync der er modtaget, samt hvor mange der er mistet, i en satellitpassage. Det kræver naturligvis at DFU'en holder rede på, præcis hvornår der forventes en framesync. Da antal bit i én ramme (110900 bit) kendes, kan en tæller der inkrementeres på clock-signalet, og nulstilles når der detekteres framesync, angive dette.
Benyttes en LED kan det kun indikeres når en framesync detekteres.

Det er besluttet at implementere indikationen vha. en LED, da det vurderes at antallet af hhv. framesync og manglende framesync, ikke er en interessant information på modtagestationen. Det vigtigste er at vide om der brugbar NOAA-data eller ej, indikeret på en tydelig måde.
Da NOAA-bitraten er 665,4 kbit per sekund, og der er 110900 bit per ramme (dvs. mellem hver framesync) forventes der framesync seks gange per sekund

$\left(665400bit \cdot s^{-1} / 110900bit\right)$. Et forsøg med en firkant-signalgenerator (med seks hertz) forbundet med en LED, viste at det var svært at se den blinke. Ved tre hertz var den tydelig. Derfor skal frekvensen halveres, som vist i næste afsnit.

6.2.3 Løsning 1:

Modtagelse af clock- og data-signalerne, samt detektering af framesync, kan realiseres med et 60 bit skifteregister, og en 60 bit digital comparator. For hver clock skifter der et databit ind i registeret. Udgangen (60 bit) sammenlignes med de 60 framesync bit – og comparatorens udgang aktiveres, hvis der er lighed. Hvis der bliver tilføjet en counter, kan denne generere et skrivesignal til mellemlagringsdelen. Hvis databredden f.eks. er 16 bit på mellemlagret, så skal der skrives for hver
16. NOAA-clock. Figur 10 illustrerer princippet. Bemærk flip flop'en. Dens funktion er at halvere LED'ens blinkefrekvens. Dvs. tre blink per sekund indikerer at alle seks framesync er detekteret.



Figur 10 Forslag til detektering af framesync

Denne løsningsmetode kan realiseres i en CPLD[1] eller en FPGA[2], vha. et Hardware Description Language (HDL), f.eks. VHDL. Brug af diskrete komponenter hører vist fortiden til, det vil i hvert fald være en ufleksibel løsning. Forskellen på en CPLD og en FPGA er i praksis kun den at FPGA'en skal bruge en ekstern komponent (f.eks. en EEPROM) til lagring af sit program, da den nulstilles hvis strømforsyningen fjernes. En CPLD et internt Nonvolantile lager.
Flere IC-producenter tilbyder CPLD'er og FPGA'er, f.eks. Altere, Atmel og Xilinx.

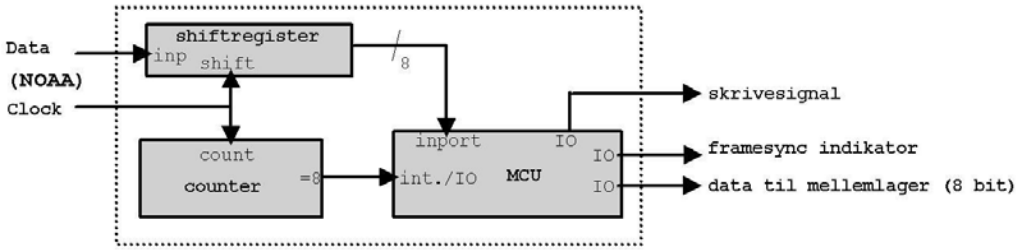
6.2.4 Løsning 2

Til modtagelse af clock og data, kan der benyttes en mikrokontroller (MCU). Der er umiddelbart to muligheder for indlæsning af data:

1. Hardwareinterrupt. Hvis clock-signalet forbindes til et eksternt interrupt ben, og data-signalet på et input-ben, kan en interrupt-rutine indlæse hver data-bit.
2. MCU'en læser kontinuert på clock-signalet (poller), og hvis der detekteres logisk 1 efterfulgt af logisk 0, er det udtryk for en faldende flanke – dvs. et data-bit på inputbenet kan indlæses.

[1] Complex Programable Logic Device
[2] Field Programable Gate Array

En uhensigtsmæssighed fælles for de to metoder er at det kun er ét bit der skal indlæses ad gangen (data-signalet). En MCU indlæser typisk otte per indlæsning, afhængig af databusbredden. M.a.o. så udnyttes kun ét bit ud af otte. Dette kan løses ved at benytte et eksternt 8bit-shiftregister, samt en counter, der for hver ottende bit generere signal til MCU'en (enten interrupt eller IO). Figur 11 illustrerer princippet. Naturligvis kan MCU'en også varetage skrivning til mellemlageret (se evt. Figur 5, del 2).



Figur 11 Modtagelse af NOAA-data vha. mikrokontroller, shiftregister og counter

En anden uhensigtsmæssighed ved brug af MCU'en opstår idet den skal detektere framesync. Hver byte der indlæses skal sammenlignes med den første byte i framesync (60 bit). Hvis der ikke er lighed skal den indlæste værdi shiftet op til syv gange, eller indtil der detekteres framesync. Dette skal ske for hver byte der indlæses. Med løsning 1 foregår sammenligningen kombinatorisk (den digitale comparator).

6.2.5 Valg af løsningsmetode

Brug af en CPLD eller en FPGA til modtagelse af NOAA-data, samt detektering af framesync er mere effektivt, i forhold til brug af en MCU. Derfor vælges løsning 1. Valg af enten CPLD eller FPGA, antal IO-forbindelser, samt størrelse (hhv. antal macrocell's og antal gates) må vente til implementationen (se afsnit 9), da det er sandsynligt at den også skal interface til mellemlagringsdelen. Personlige erfaringer med produkter fra Xilinx, gør at IC'en med fordel kan vælges hos denne producent.

7 Løsningsmuligheder for mellemlagring

Mellemlagringen skal bruges til lagring af NOAA-data, i tilfælde af at DFU'en ikke kan transmittere det over internettet (MPLS-forbindelsen), som følge af ustabil / afbrudet forbindelse. Til trods for at TDC garanterer en høj oppetid, samt transmissionshastighed på 2 Mbit per sekund, har DMI enkelte gange oplevet udfald på forbindelsen. Derfor er det nødvendigt at kunne mellemlagre i DFU'en.

7.1 Specifikationer

7.1.1 Læse- og skrivehastighed

Den laveste skrivehastighed som er acceptabel til mellemlagret afgøres af NOAA-dataraten, dvs. 665,4kbit per sekund. Laveste acceptable

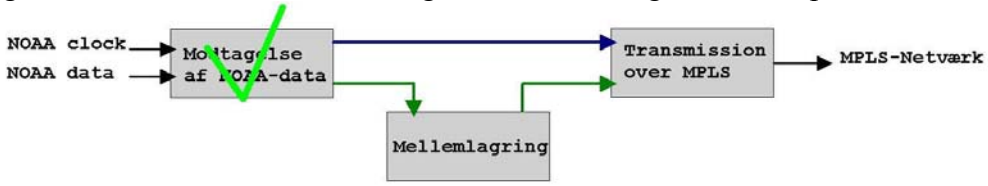
læsehastighed er givet ved den hastighed som DFU'en som minimum skal kunne transmittere med (se evt. afsnit 4.2 på side 10), dvs. to gange NOAA-dataraten (1308 kbit per sekund). Naturligvis vil det være at foretrække at kunne udnytte hele MPLS-forbindelsens båndbredde på 2048 kbit per sekund.

7.1.2 Lagerkapacitet

Det vurderes at mellemlagring af én satellitpassage er nok til DFU'en, da det kun er sket et par gange, at DMI har oplevet nedbrud i kortere tid.

7.1.3 Interaktion med modtagedelen og transmissionsdelen

Det er det kun nødvendigt at skrive til mellemlagret hvis der modtages NOAA-data OG MPLS-forbindelsen er afbrudt. Dvs. under normale omstændigheder skal NOAA-modtagedelen overføre sit data til MPLS-transmissionsdelen (den blå pil på Figur 12). Men hvis MPLS-forbindelsen afbrydes, skal NOAA-modtagedelen i stedet, overføre sit data til mellemlagringen. Så snart forbindelsen gendannes, skal data i mellemlagret overføres til transmissionsdelen (grønne pile på Figur 12). Umiddelbart en avanceret datarute. Et andet forslag er, konsekvent at lade modtagedelen skrive til mellemlagret. Så snart der er data heri, kan transmissionsdelen læse, og transmittere det. Denne løsning vil være mere ligefrem at implementere.



Figur 12 Interaktion med mellemlagret

Det er valgt at arbejde videre med sidst nævnte løsning, dvs. at lade modtagedelen skrive konsekvent til mellemlagret.

7.1.4 Lagertype

Anvendelse af en af følgende lagringstyper overvejes:

1. FlashRAM
2. RAM

7.1.4.1 FlashRAM

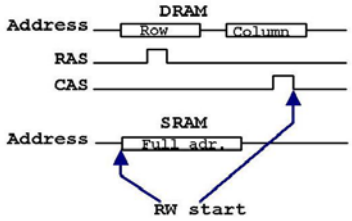
FlashRAM er Nonvoloantile RAM, dvs. informationen lagret gemmes selvom strømforsyningen fjernes. Til mellemlagret overvejes det at benytte et CompactFlash-kort (CF-kort) (ofte brugt i digitalkameraer). Et 128 Mbyte CF-kort fra chip producenten Kingston[3], har en skrive- og læsehastighed på hhv. 1,5 og 3,5 Mbytes per sekund, hvilket er langt hurtigere end mellemlagret behøver. Problemet er at FlashRAM typisk kun holder til 10000 skriveoperationer.
En typisk NOAA-passage fylder ca. 70 Mbytes. Dvs. et 128 Mbyte CF-kort, udsættes for én skriveoperation (på alle lagerpladser) efter 1,8 NOAA-passage. Hvis der modtages 20 passager per dag (DMI modtager mellem 20

[3] www.kingston.com

og 40 passager), skrives der 11,1 gang på alle lagerpladser per dag. Det giver CF-kortet en levetid på ca. 900 dage (i bedste fald).

7.1.4.2 RAM

Mht. RAM (Random Acces Memory), så er grundlæggende to typer tilgængelige: Statisk RAM (SRAM) og Dynamisk RAM (DRAM). SRAM lagrer hvert bit i en flip flop, hvorimod DRAM gemmer hvert bit som ladning på en kondensator. Det betyder i praksis at en SRAM med samme lagerkapacitet som en DRAM optager mere areal.



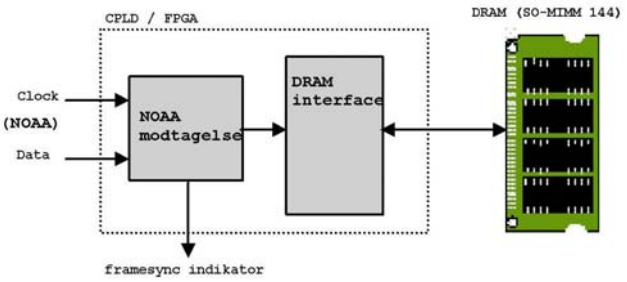
Figur 13 Adressering af DRAM (øverst) og SRAM (nederst)

Mellemlagret i DFU'en skal bestå af DRAM. Det mest fornuftige er at bruge et RAM-modul, af samme type som er monteret i en PC, dvs. i en sokkel. Dette giver mulighed for let udskiftning / opgradering. Det ville ikke være fordelagtigt at montere (pålodde) hver DRAM-IC på DFU'ens PCB. Der er flere modultyper tilgængelige, f.eks.SIMM[4] og DIMM[5], begge med forskellige antal benforbindelser og overførselshastighed. SIMM anvendes næsten ikke mere (blev brugt i de tidlige PC-modeller indtil Pentium-processoren).

7.1.4.3 Valg af løsning

Valget falder på et SO-DIMM modul med 144 benforbindelser. De fleste bærbare PC'ere benytter netop denne type, og er er derfor let tilgængeligt. Valget betyder imidlertid at der skal fremstilles et interface til DRAM-modulet, som kan varetage refresh, samt læse- og skrive operationer. Interfacet er derfor mere komplekst end ved S- og PSRAM. Det er naturligt at realisere interfacet i samme CPLD / FPGA som modtagedelen til NOAA-data. Med Figur 10 (side 6) in mente, kommer systemet til at se ud som vist på Figur 14.

[4] Single Inline Memory Module (modul med benforbindelser på én side)
[5] Dual Inline Memory Module (modul med benforbindelser på begge sider)



Figur 14 Modtagelse af NOAA-data og DRAM-nterface i en CPLD eller FPGA

Organisationen JEDEC har defineret specifikationerne for 144 pins DRAM SO-DIMM moduler [*JEDEC 21-C*]. Heraf fremgår det, at der er flere aspekter der skal tages højde for, når der implementeres et sådan RAM-interface: Spændingsforsyningen kan enten være 3,3V, 5V, eller brugerdefineret. For at forhindre at et ikke kompatibelt RAM.modul isættes i en sokkel, er der placeret en mekanisk stopklods i selve soklen. Dvs. udskæringen i RAM-modulet skal passe til placeringen af stopklodsen. Da der kan være forskel på lagerkapaciteten (og derved adressebusbredden) for hvert modul, er det nødvendigt at kende konstellationen af modulet.
F.eks. så har to 128 Mbytes RAM-moduler fra samme producent forskellige antal række- og kolonne-adresser, som vist i Tabel 2.

Type	Rows [bit]	Colums [bit]	Total Size [Mbytes]
1	12	9	128
2	13	8	128

Tabel 2 To forskellige opbygninger af 128 Mb RAM-modul

Da DRAM skal adresseres á to omgange (se Figur 13), er det nødvendigt at kende antal tilgængelige Row- og Column-adresser. Denne information bl.a., kan læses på en seriel EPROM monteret på RAM-modulet, vha. et I2C-interface[6]. Hvis DFU'en skal være i stand til at benytte vilkårlige RAM-moduler, behøves der derfor en I2C-controller. Et alternativ er at definere én kompatibel RAM-konstellation til brug i DFU'en. Disse aspekter behandles i dybden i afsnit 229 (Implementering af DFU).

8 Løsningsmuligheder til transmissionsdel

Transmissionsdelen skal overføre NOAA-data over internettet (via. en MPLS-forbindelsen). Følgende krav skal overholdes:

- Transmissionsrate ≥ 1330,8 kbit per sekund (to gange NOAA-dataraten).
- Fysisk interface som twisted pair (RJ45 konnektor), som defineret i kravspecifikationen.

[6] Inter Integrated Circuit Interface. Se http://www.semiconductors.philips.com/

8.1.1 Diskret Ethernet Controller

En klassiker når det kommer til diskrete Ethernet Controllere, er *CS8900A* fra producenten *Cirrus Logic*[7]. Den har et 10BASE-T-interface, dvs. overførselshastighed på 10 Mbit per sekund. Periferiudstyr kommunikerer med *CS9800A*, vha. et ISA[8]-interface. Denne kontroller kan bruges i DFU'en, da den lever op til kravene om transmissionshastighed og fysisk interface. Det kræver at der implementeres et ISA-interfacet, i den CPLD eller FPGA, der er vælges til RAM-controlleren og NOAA-modtagedelen.

8.1.2 Mikrokontroller

En anden mulighed er at vælge en mikrokontroller (MCU) med IP-stack. Producenten Beck IPC[9] har udviklet en Embedded Web Controller-serie. De er alle baseret på Intels 186 arkitektur, og kræver ikke brug af periferikomponenter, da både krystal til generering af clock-frekvens, og netværksinterfacet er indlejret. Et udpluk af specifikationerne for den mindste controller i serien (model SC12) ses nedenfor:

- 512 KB RAM, 512 KB Flash
- RTOS[10] with Flash File System (FAT16)
- TCP/IP, PPP, HTTP, FTP, Telnet, POP3, SMTP and DHCP
- I²C Bus

Som det kan ses, har *SC12* allerede et RTOS installeret, med et FAT16 filsystem. Alle brugerspecifikke programmer (f.eks programmet der skal transmittere NOAA-data), overføres som DOS eksekverbare filer, til *SC12's* interne Flash, vha. en ftp-klient. Manuel eksekvering af filerne, sker over *SC12's* serielport, vha. et terminalprogram (f.eks. Hyperterminal). Idet der tilsluttes, præsenteres brugeren for en kommandoprompt, hvor der kan eksekvere programmer, præcist som i DOS. Ønskes der automatisk eksekvering af programmer efter BootUp (som i dette projekt), kan de angives i en autoexec.bat-fil. Konfigurering af f.eks. *SC12's* statiske IP-adresse, eller om DHCP ønskes, sker i en initialiseringsfil. Denne læses automatisk af operativsystemet under opstart. Envidere har SC12 I2C-bus, hvilket muliggør læsning af information fra RAM-modulets EPROM.
Disse brugervenlige funktioner giver en stor fordel i forhold til *CS9800A*-chippen: DMI kan selv konfigurere DFU'ens egen IP-adresse, samt destinations IP-adressen (Lyngbyvej). Med *CS9800A* bliver konfigureringen kompliceret, da det vil kræve at der implementeres et brugerinterface i CPLD'en / FPGA'en, samt en ekstern hukommelse til lagring af f.eks. IP-adresser.

[7] http://www.cirrus.com/en/products/pro/detail/P46.html
[8] Industry Standard Architecture.
[9] http://www.beck-ipc.com/
[10] Real Time Operation System

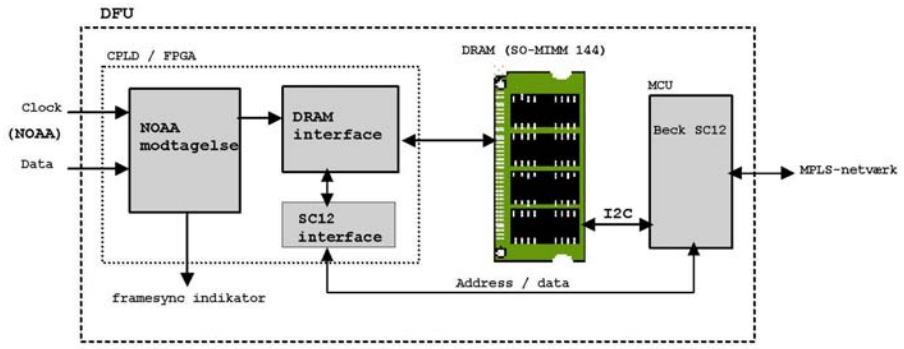
8.1.3 Valg af løsning

Brug af en *SC12* til transmissionsdelen er åbenlyst det bedste valg, frem for en diskret kreds, da det vil lette implementationen, og derved mindske udviklingstiden. Dens 10BASE-T-interface lever både op til kravene til hastighed og fysisk interface. Endvidere kan den let interfaces til CPLD'en / FPGA'en med sin standardiserede multiplex'ede adresse- og databus. Derfor vælges *SC12* til brug i DFU'en.

9 Implementering af DFU

9.1.1 Indledning

I de forrige afsnit er det fastlagt hvordan hver enkelt del i DFU'en skal implementeres i praksis. En oversigt over hele DFU'en kan ses på Figur 15 nedenfor. Der skal fremstilles flere VHDL-moduler til implementation i enten en CPLD eller en FPGA, fra producenten Xilinx. Om det skal være en CPLD eller FPGA kommer dels an på hvor meget VHDL-modulerne kommer til at fylde, og dels hvor mange IO-benforbindelser der påkræves. Afgørelsen må træffes senere i dette afsnit.



Figur 15 Blokdiagram over alle dele i DFU'en

Til hhv. programmering og simulering af VHDL-modulerne benyttes udviklingsværktøjet Project Navigator[11] (version 8,1i) og ModelSim XE III (Starter Version 6,0d). ModelSim er inkluderet i ProjectNavigator.
Som det fremgår af Figur 15 skal der udvikles et DRAM-interface, et interface til MCU'en (*SC12*), samt modtage- og synkroniseringsdelen til NOAA-data. Programmet til MCU'en fremstilles i en C-Compiler (Borland 5.02).

9.1.2 Modtagelse af NOAA-data

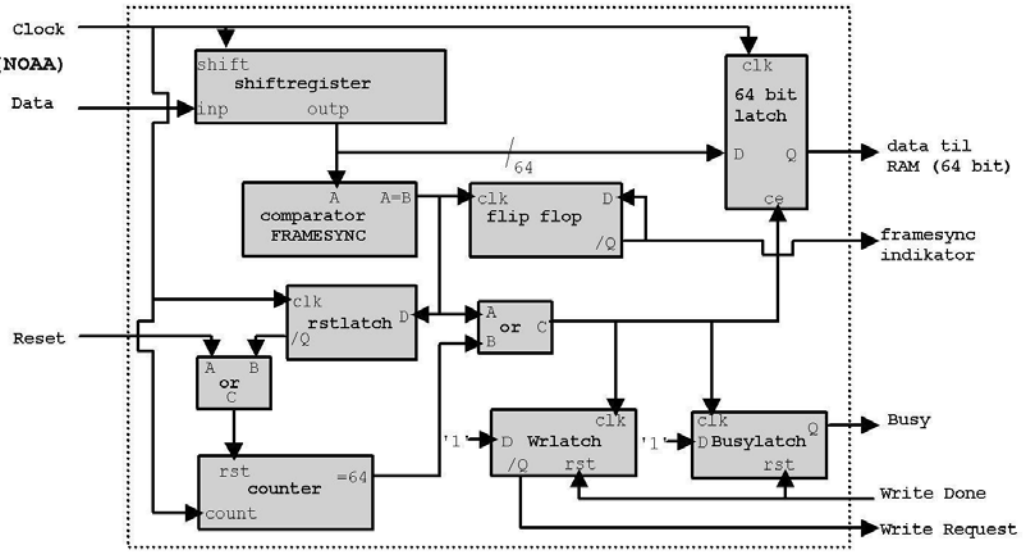
Der skal modtages NOAA-data med en frekvens på 665,4 kbit per sekund, i form at et clock- og et datasignal. Det er besluttet ikke at implementere modtagelse af NOAA-data i form af BPSK-signalet, da det ikke er krævet af DMI.

Både clock og data er TTL-niveau, med en udgangsimpedans på 75 Ω.

[11] Evalueringsversion af Project Navigator kan downloades på www.xilinx.com

Begge signaler kan derfor føres direkte ind i CPLD'en eller FPGA'en, dog med en impedanstilpasning på hver indgang. Da 665,4 kHz er en relativ lav frekvens, kan der blot placeret en 75 Ω modstand i parallel med indgangen.

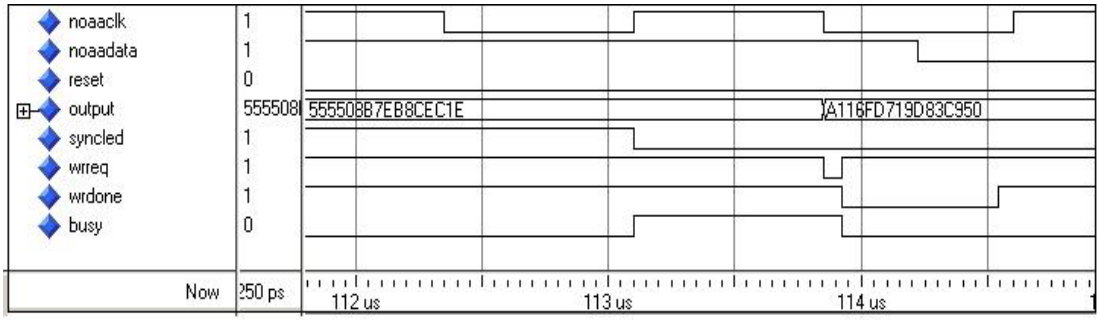
Modtagelsen er implementeret efter samme princip som vist på Figur 10, dvs. vha. et shiftregister og en comparator til detektering af framesync. Modtagelsen er imidlertid forfinet, da den også skal genere skrivesignal til RAM-controlleren. I den forbindelse skal der tages højde for, at både modtagedelen og transmissionsdelen samtidig kan foretage hhv. en skrive- og en læseoperation til/fra RAM-modulet, hvilket ikke må ske.
Den udviklede RAM-controller tager ikke højde for dette scenario (se evt. afsnit 9.1.3). Problemet er løst ved at tilføje en Busy-signal (se Figur 16) til modtagedelen, som aktiveres inden den skriver til ram. Dette signal er tilgængeligt for mikrokontrolleren, og gør at den kan læse dette signal, og sikre sig at RAM-controlleren er ledig, inden den påbegynder en læsning.
Figur 16 viser implementationen af modtagedelen. Øverst til venstre ses skifteregisteret (64 bit bredde), og comparatoren der konstant undersøger om bitsekvensen er lig med en NOAA-framesync. I tilfælde af, at der ikke detekteres framesync, vil tællerens (nederst t.v.) udgang blive aktiveret for hver 64ende bit, og gennem or-gaten, aktivere `ce` (chip enable) på latchen (øverst t.v). På næste clockflanke latches værdien, og er derved tilgængelig for RAM-controlleren. Samtidig latches der et write request (`WrLatch`) til RAM-controlleren.
Hvis der detekteres framesync bliver tælleren nulstillet, således at der ventes på 64 bit, før den latcher data til RAM-controlleren igen. Dvs. for hver framesync der detekteres, synkroniseres data til RAM.



Figur 16 Opbygning af NOAA-modtagerdelen.

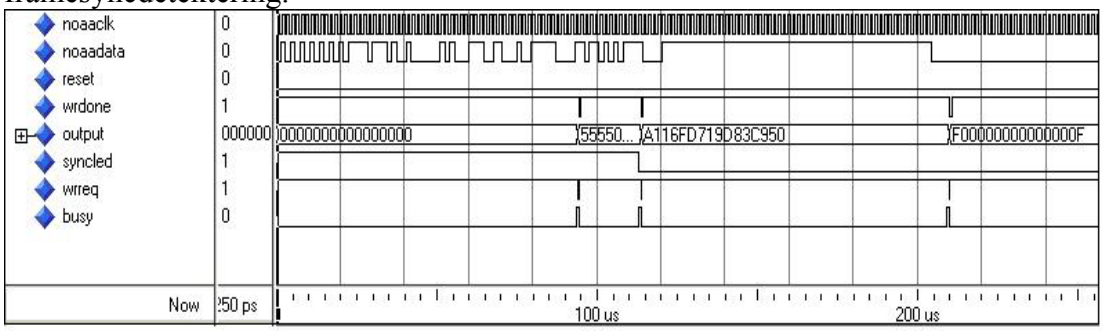
Modtagedelen er simuleret vha. en testbench, der først genererer tilfældigt data, dernæst framesync, og igen tilfældigt data. Testbenchen kan ses i bilag. Figur 17 viser signalerne idet der detekteres en framesync. Data skiftes ind i

på den `noaaclk`'s positive flanke. I det samme aktiveres `busy` (ca, 113 us), for at indikere til mikrokontrolleren at en skriveoperation er nær forestående. På den efterfølgende negative flanke af `noaaclk`, latches data til `output` (til RAM-controlleren), og `wrreq` aktiveres (aktiv lav), hvilket initiere skriveoperationen i RAM-controlleren. Signalet `wrdone` fra RAM-controlleren aktiveres (aktiv lav) idet den har accepteret skriveoperationen, og deaktiveres igen når den er fuldført.
Bemærk at `syncled` skifter efter detektering af framesync.



Figur 17 Simulering af NOAA-modtagedelen, idet der detekteres framesync.

Simuleringen på Figur 17 er som ovenstående, hvor der blot er zoom'et ud. Her kan der ses tre skriveoperationer, hvoraf den ene er initieret af en framesyncdetektering.



Figur 18 Simulering af NOAA-modtagedelen. Tre efterfølgende skriveoperationer.

Simuleringen viser at modtagedelen virker som forventet.

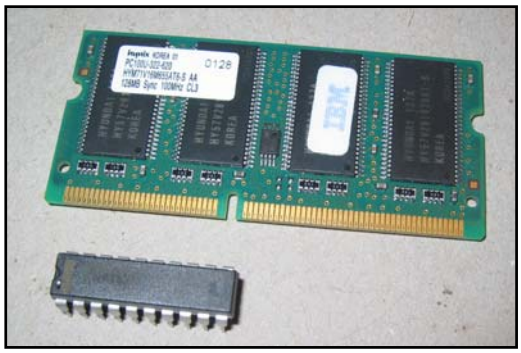
9.1.3 Mellemlagring

I følgende afsnit beskrives det hvordan implementationen af SDRAM-interfacet er udført. Som det første beskrives interfacet til et SO-DIMM 144 RAM-modul. Derefter beskrives funktionaliteten af det udviklede VHDL-modul; RAM-controlleren.

9.1.3.1 SO-DIMM 144 interface

Nogle RAM-producenter tilbyder en HDL timingsmodel af deres produkter. Det er lykkes at fremskaffe et 128 Mbytes SO-DIMM SDRAM-modul (se Figur 19), samt den tilhørende VHDL-timingsmodel, fra firmaet Hynex. SO-DIMM modulet er af typen HYM71V16M655 med følgende grundspecifikationer:

- Maksimal clock frekvens på 100MHz
- Driftspænding på 3,3V±0,3V
- LVTTL[12]-kompatible IO's
- 4 interne Banks, og 2 fysiske med en densitet på hver 64 Mbytes
- 4096 Refresh Cycles / 64 ms
- Programmerbar CAS-Latency på 2 eller 3 Clock Cycles



Figur 19 Det valgte SDRAM-modul.

Som tidligere nævnt er der i alt 144 benforbindelser på modulet. Ud af disse, er der hhv. 18 GND- og 17 Vcc forbindelser, der er jævnt fordelt, for at eliminere støj. De resterende signaltyper er opstillet i Tabel 3.

PIN	Funktion
CK0, CK1	Clock Input
CKE0, CKE1	Clock Enable
/S0, /S1	Chis Select
BA0, BA1	Bank Select
A0 – A11	Adresse bus
/RAS, /CAS, /WE	Row Address Strope Column Address Strope Write Enable
DQ0 – DQ63	Data I/O
DQM0 – DQM7	Data I/O Mask
SCL	SPD Clock
SCA	SPD Data

Tabel 3 Signaloversigt på SO-DIMM modul

Forklaring af signaler

CK0, CK1:

Der er to identiske clock-signaler til RAM-modulet, da et RAM-modul typisk består at mellem fire og otte IC'ere. Hvis ét clock-signal skal drive alle disse,

[12] Low Voltage TTL (3,3V)

vil Fan Out være for stor. Alle input-signaler samples på den stigende flanke af CK0 og CK1.

CKE0, CKE1:

Disse signaler kan deaktivere hhv. CK0 og CK1, hvilket betyder at modulet indtræder i Power Suspend Mode.

/S0, /S1:

Chip Select signaler som hver især kan aktivere et halvt RAM-modul. Ofte aktiverer /S0 den ene side af modulet, og /S1 den anden side.

BA0 – BA1:

Hver IC på RAM-modulet indeholder op til fire interne Banks, som hver især kan aktiveres med BA0 – BA1.

A0 – A11:

Adressebus som både benyttes til adressering af både Row- og Column address. Endvidere benyttes disse signaler til at konfigurere RAM-modulte (f.eks. hvilken CAS-latency der ønskes).

/RAS, /CAS, /WE:

Disse tre signaler afgør hvilken kommando RAM-modulet skal eksekvere, f.eks. skrive- eller læseoperation.

DQ0 – DQ 63:

Data-signaler som benyttes ved skrive- og læseoperationer. Alle otte bytes overføres på én gang.

DQMB0 – DQMB7:

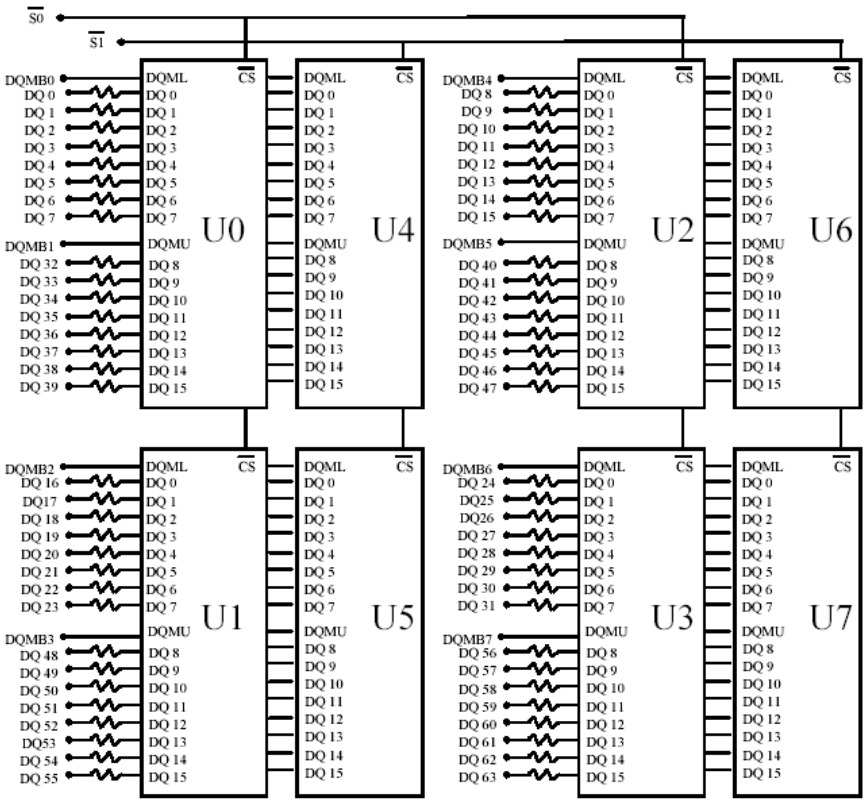
Maskering af hver byte på data-signalerne, ved læse- eller skriveoperation. Dvs. hvis der kun ønskes skrevet én byte til f.eks. DQ0 – DQ7, kan DQMB1 – DQMB7 deaktiveres. Det vil deaktivere / ignorere skrivning til de øverste syv bytes.

SCL og SCA:

Er hhv. clock- og data-signalet til SPD[13]-EPROM'en (dvs. clock og data på I2C-bussen).

Modulet består af otte SDRAM IC'ere, U0 – U7 med en databredde på 16 bit. Fordelingen af data-, DQMB- og Chip Select (S0 og S1) kan ses på figuren nedenfor. Hver IC adresseres med 9 kolonner (Columns) og 12 rækker (Rows), vha. A0 – A11. Endvidere er der 4 interne banker (Banks), som vælges med BA0 – BA1. Det giver et adresseområde på 2^{23}-1 (8 M). Dvs. lagerkapaciteten for hver IC er 16 Mbytes (16 bit data * 8 M), og derfor 128 Mbytes i alt på modulet.

[13] Serial Precense Detect (EPROM som indeholder information om RAM-modulets konstellation)



Figur 20 Opbygning af det valgte Hynix SO-DIMM modul.

Clock-signalerne `CK0` og `CK1` er hhv. tilsluttet U0 – U3 og U4 – U7.

9.1.3.2 Seriel Presence Detect (SPD)

Som nævnt i afsnit 7.1.4.3 er der monteret en seriel EPROM på RAM-modulet, som indeholder specifikationerne for opbygning, f.eks. antal rækker og kolonner. Hvis det skal være muligt at isætte RAM-moduler med varierende lagerkapacitet er det altså nødvendigt at kunne læse fra EPROM'en.

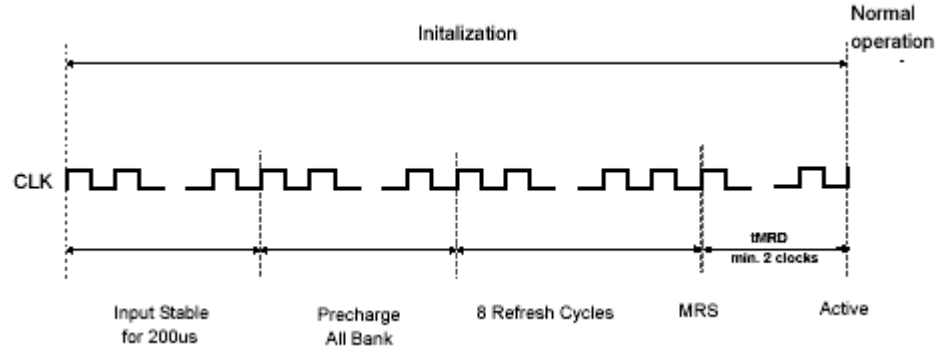
9.1.3.3 Beskrivelse af virkemåde

Som det ses i Tabel 3 er det signalerne `/RAS`, `/CAS`, og `/WE`, der afgør hvilken kommando RAM-modulet skal eksekvere. I dette afsnit vil de kommandoer der er implementerer i RAM-Controlleren, blive beskrevet.

Før RAM-modulet accepterer læse- og skriveoperationer, skal det initialiseres vha. kommandoen mode register set (MRS). Det indebærer følgende:

- **CAS-latenc:**. CAS-latency kan enten være to eller tre clock cycles, og afgører hvor mange clock cycles der går, fra en læseoperation eksekveres, til data er tilgængelig.
- **Burst Read / write:** Det er muligt at læse / skrive op til otte efterfølgende adresser, med én læse- / skriveoperation.

Følgende timingsdiagram viser den påkrævede sekvens af kommandoer efter at strømforsyningen er tilsluttet:

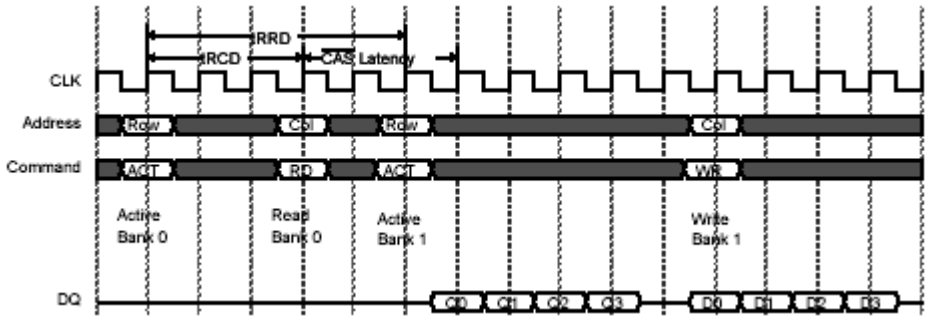


Figur 21 Initialisering af RAM-modul efter at strømforsyningen er tilsluttet.

Efter at strømforsyningen er tilsluttet skal der minimum ventes 200 us før der må eksekveres kommandoer. Dvs. RAM-modulet må kun udføre NOP[14] "operationer". Efterfølgende skal alle banker oplades (precharge kommando), hvorefter der som minimum skal eksekveres otte refresh-kommandoer.
Nu er RAM-modulet klar til at blive konfigureret vha. MRS-kommandoen. Efterfølgende bliver de operationer som skal implementeres beskrevet.

9.1.3.4 Læse- skriveoperation

Figur 22 illustrerer en burst read- (fire bursts) og en burst write operation (også fire bursts).
Læse operationen kan opdeles i tre trin:
1) Row address placeres på adressebussen (`A0 – A11`), Bank address placeres på `BA0 – BA1`, og aktiveres med en Activate kommando (ACT).
2) Column address placeres på adressebussen, og læseoperationen initieres (RD kommando).
3) Læseoperationen er i gang, og data er tilgængelig på `DQ0-DQ63` efter tre clock cycles (hvis CAS-latency konfigureret til tre).



Figur 22 Burst Read og Burst Write.

Skriveoperationen kan opdeles i to trin:

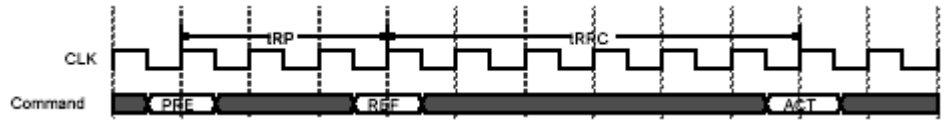
1) Row address placeres på adressebussen (`A0-A11`), Bank address placeres på `BA0 – BA1`, og aktiveres med en Activate kommando (ACT).

[14] No Operation

2) Column address placeres på adressebussen, data placeres på DQ0-DQ63, og skriveoperationen initieres (WR kommando).

9.1.3.5 Refresh og Precharge operation

For at undgå tab af data skal alle rækker "opdateres" med en auto refresh-operation (REF), med et interval på minimum 64 ms. Én REF-operation opdaterer Rækkerne der skal opdateres adresseres med en intern refresh counter, som inkrementerer efter hver REF-operation. Efter initiering af REF, skal der ventes tREC (70 ns) inden der igen må eksekveres kommandoer. Kun NOP-operationer er tilladt.



I alt skal der eksekveres 4096 ($2^{12\text{rækker}}$) REF-operationer inden for 64 ms.

9.1.3.6 Precharge operation

Når en bank aktiveres, f.eks. før en læseoperation, skal den efterfølgende precharges. Dette er der taget højde for i læse- og skriveoperationerne, ved at der internt precharges automatisk. Men som det ses på Figur 21 skal alle banker precharges idet RAM-modulet initialiseres. Derfor er det nødvendigt at implementere denne kommando.

9.1.3.7 Mode Register Set

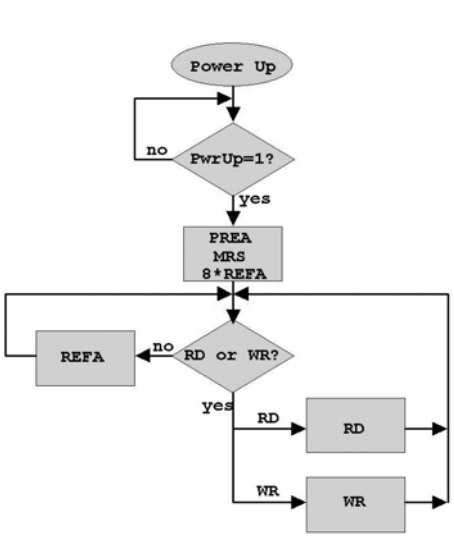
Som nævnt benyttes mode register set (MRS) kun til konfigurering af RAM-modulet. Det er værdien på A0-A11 der afgør konfigurationen:

Signal	Funktion
A0	Burst Length (LSB)
A1	Burst Length
A2	Burst Length (MSB)
A3	Burst Type
A4	CAS-latency (LSB)
A5	CAS-latency
A6	CAS-latency (MSB)
A6-A11	Reserved

9.1.3.8 Arkitektur af VHDL DRAM-controller

RAM-controlleren er designet i VHDL. Der er taget udgangspunkt i flow-diagrammet vist på figur 16.

Efter at spændingsforsyningen er tilsluttet ventes der på at signalet *PwrUp* aktiveres. Indtil da eksekveres der kun NOP-operationer. Det er for at sikre at der ventes minimum 200 us, som angivet på Figur 21. Idet *PwrUp* aktiveres eksekveres der en precharge (PREA)-, og otte refresh operationer (REFA), hvorefter mode registeret sætte (MRS). Nu er RAM-modulet initialiseret som foreskrevet. Nu er RAM-controlleren i *idle mode*, set fra brugerens synspunkt. Dvs. der ventes på enten en skrive- eller læseoperation, initieret fra brugeren. For at sørge for at der ikke mistes data, eksekveres der konsekvent refresh operationer (REFA) i denne tid.



Figur 23 Flow over RAM-controllerens virkemåde.

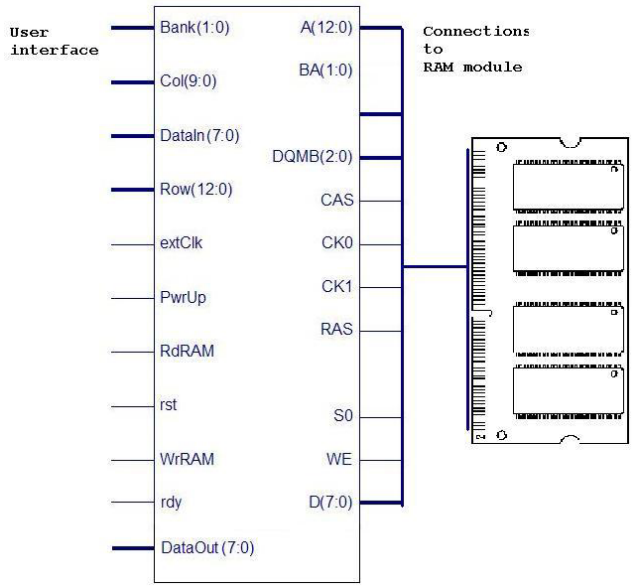
VHDL RAM controlleren er opbygget omkring to State Machines; MainStates, og ExecutionStates.
MainStates indeholder ét State for hver operation som controlleren skal udføre, f.eks. PowerUp-, Read-, og Write. Hver af disse operationer benytter mere end én clock cycle, for at eksekvere, og derfor er der de fornødne ExecutionStates (maks ti) i hver MainState. Nedenstående VHDL-kode viser princippet for PowerUp-mainstate'et:

```
case COMMAND is
  when PowerUp =>
    rdy <= '0';
    case EXE_STEP is
      when Step1 =>
        PREA;
        EXE_STEP <= Step2;
      when Step2 => .....
      COMMAND <= SetModeReg;
      end case; --End of PowerUp State

  when SetModeReg =>
    rdy <= '0';
    case EXE_STEP is
      when Step1 .....
```

RAM-modulet' entity er illustreret på Figur 24. Forbindelserne på venstre side "brugerens" interface, dvs. de signaler der kontrollerer RAM-

controllerens funktion. Højre side er udelukkende tilslutninger til RAM-modulet.



Figur 24 RAM-controllerens Entity. Venstre side er bruger interface, og højre side er tilslutninger til RAM-modulet.

Bemærk at der, på nuværende tidspunkt, er visse begrænsninger i VHDL-modellen. Det er fordi at timingsmodellen af det valgte SO-DIMM modul kun beskriver otte datalinier (`DQ0-DQ7`, ét DQM-signal, samt ét chip select signal (S_0, og ikke S_1). Derfor er den fremstillede VHDL-model også begrænset til det. I den endelige model skal alle signaler naturligvis implementeres.

9.1.3.9 Specifikation af brugerinterface

Signalerne til brugerinterfacet kræver en forklaring:

- `extClk`, er clock-signalet til RAM-controlleren og DRAM-modulet. Alle operationer sker på den stigende flanke af dette signal. Som tidligere nævnt, sampler RAM-modulet alle sine input på den stigende flanke af sit clock-signal. Derfor er RAM-controlleren designet til at operere på den faldende flanke.

- `RST`, er et asynkront reset-signal (dvs. uafhængigt af `exeClk`) , som er aktiv lav. Efter aktivering nulstilles alle registre i RAM-controlleren, og den vil efterfølgende vente på `PwrUp` (aktivt høj) signalet, som illustreret på Figur 24.

- `rdy` indikerer om RAM-controlleren er klar til at modtage nye kommandoer, eller om den er optaget med den forrige kommando.

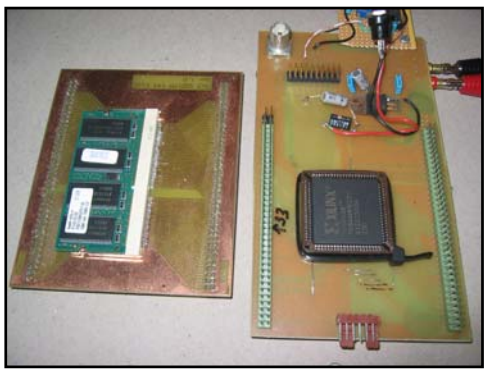
- `RdRAM` (aktiv høj) initiere en læseoperation fra RAM-modulet. Inden `RdRAM` aktiveres skal signalerne `Col`, `Row`, `Bank`, og S_0,

angive lokationen hvorfra der ønskes læst. Så snart RAM-controlleren har modtaget ønsket om læse-operationen (det sker på den faldende flanke af `extClk`), deaktiveres `rdy`-signalet. Dvs. at der ikke accepteres nye kommandoer før operationen er afsluttet, og `rdy` igen aktiveres. Den læste data placeres på `DataOut`.

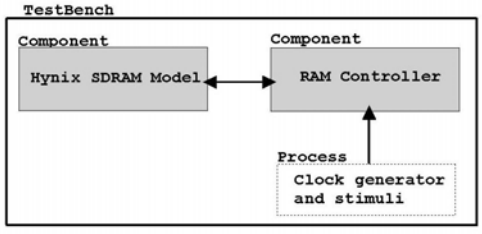
- `WrRAM` (aktiv høj) initierer en skriveoiperation til RAM-modulet. Ligesom ved en læseoperation, angiver `Col`, `Row`, `Bank` og S_0, lokationen hvor der ønskes skrevet. Data placeres på `DataIn`. `rdy` indikerer på samme måde, som ved en læseoperation, om kommandoen er modtaget, og/eller færdig.

9.1.3.10 Simulering og implementeringaf 1. RAM-controller

Tidligt i projektforløbet blev der fremstillet to test PCB'er. Det første indeholder en 108 macrocell's CPLD (XC108) fra Xilinx. Det andet PCB indeholder en SO-DIMM 144 RAM-sokkel. De to PCB'er kan forbindes med konnektorer, sådan at de fornødne benforbindelser på RAM-modulet kan forbindes til CPLD'ens IO-forbindelser. Billedet til højre viser de to PCB'er. Diagrammer og PCB-layout kan ses i bilag. Idéen med de to PCB'er var at hurtigt at blive fortrolig med RAM-controlleren, inden det endelige skulle fremstilles. Grunden til at RAM-modulet blev placeret særskilt, var for at evt. kunne genbruge dette, til den endelige konstruktion.



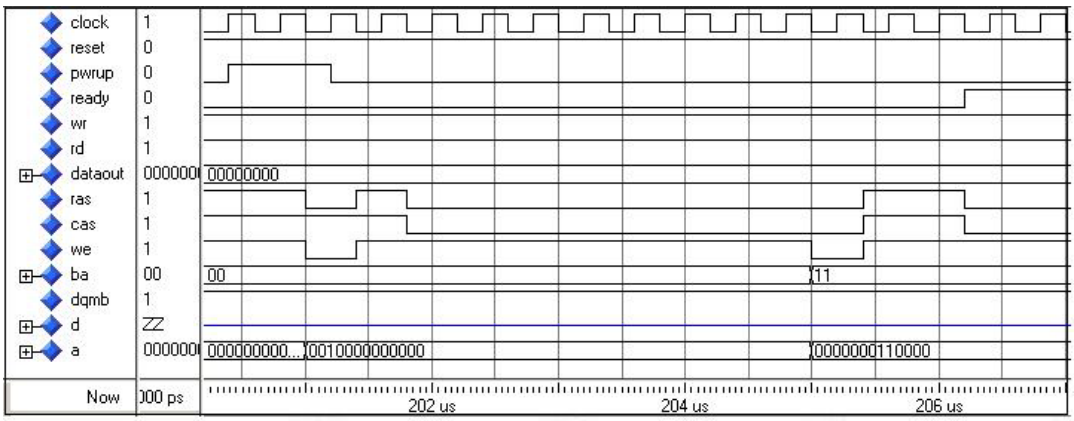
Før implementationen i CLD'en blev RAM-controlleren simuleret op imod leverandørens (Hynix) RAM-model. Der er fremstillet en VHDL-testbench med en struktur som vist på Figur 25. RAM-modellen er en component ligesom RAM-controlleren, så de er blevet forbundet med portmap funktionen. Udover disse to komponenter, er der én process til generering af clock til både RAM og RAM-controller. Endvidere indeholder denne process også stimuli, dvs. aktivering af `reset`, `pwrup`, `read` og `write`.



Figur 25 Testbench til simulering af RAM-controlleren forbundet med RAM-modellen fra Hynix.

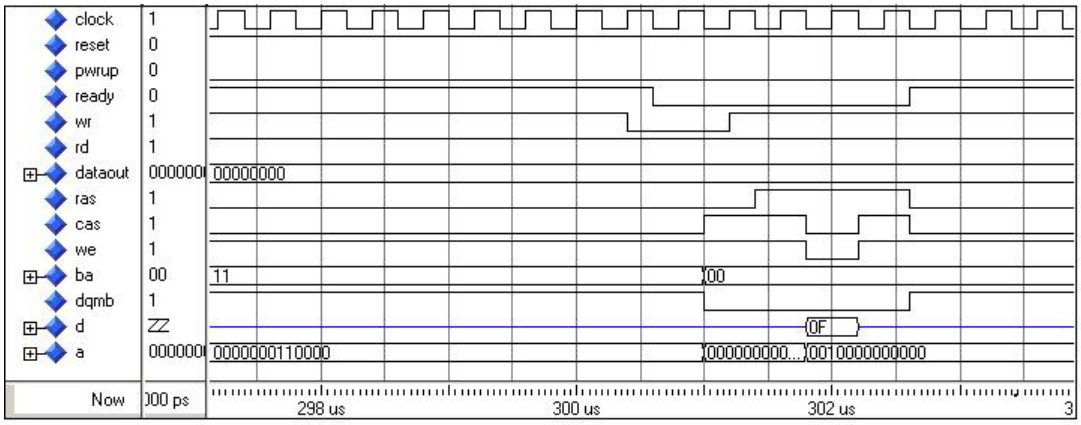
Simuleringen er begrænset til én skriveoperation og én læseoperation på én adresse i RAM, samt naturligvis initialiseringen af RAM (PowerUp og Mode Register Set).VHDL-koden for RAM-controlleren og testbenchen kan ses i bilag.

Figur 26 viser simuleringen af RAM-initialiseringen, dvs. efter opstart af RAM-controlleren venter denne på at signalet `pwrup` aktiveres (ca. 200 us), hvorefter der eksekveres en precharge-, en NOP, og otte refresh-kommandoer. Herefter eksekveres mode register set (ca. 205 us). bemørk at adressebussen (A) sættes til "000000110000" binært, hvilket ensbetydende med CAS latency på 3 clock cycles, samt burst rate på 1 (dvs. der skal kun læses / skrives én gang per læse- skriveoperation). Bemærk at `ready` signalet er deaktiveret indtil initialiseringen er forbi (ca. 206 us).



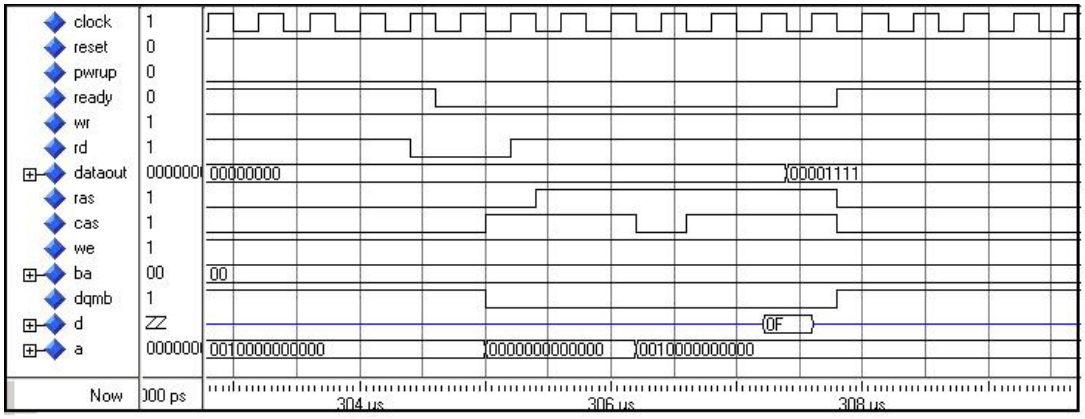
Figur 26 Simulering af PowerUp og Mode Register Set.

På Figur 27 ses skriveoperationen, der lagrer værdien 0x0F i RAM. Den initieres med signalet `wr` (ca. 300 us). Efterfølgende deaktiveres ready signalet, for at indikere at skriveoperationen igangsat. Den første kommando der eksekveres er aktivering af række og bank, derefter en NOP efterfulgt af skrive kommandoen. Skrive kommandoen placerer kolonne adressen på adressebussen (bemærk at bit ti er sat, hvilket indikerer at der udføres en intern precharge kommando), og værdien (0x0F) der skal skrives, på databussen.



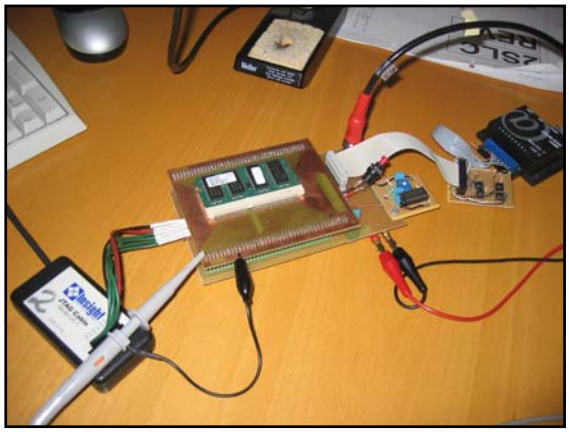
Figur 27 Simulering af skriveoperation til RAM.

Figur 28 viser sidste del i simuleringen, nemlig læseoperationen fra RAM. Den initieres med `rd` signalet (ca. 304 us) aktiveres, hvorefter `ready` deaktiveres. Rækken og banken aktiveres, og to NOP kommandoer eksekveres, hvorefter læseoperationen eksekveres. Som det ses, placerer RAM (på tredje clock) data med en værdi på 0x0F på databussen. Denne værdi latch'es ud på signalet `dataout`. Det var netop værdien som før blev skrevet, dvs. RAM-controlleren kan skrive og læse fra RAM.



Figur 28 Simulering af læseoperation fra RAM.

Timingssimuleringen viser at RAM-controlleren fungerer korrekt. Derfor er den implementeret i test-PCB'et med CPLD'en. Til aktivering af hhv. `reset`, `pwrup`, `read`, og `write`, blev der fremstillet et interimistisk hulprint med fire kontakter. Efter tilslutning af strømforsyningen blev reset aktiveret. Med et oscilloscope blev det verificeret af `ready` signalet var lavt, indtil `pwrup` blev aktiveret, netop som Figur 26 viser. For at verificere at skrive- og læseoperationen forløb



som hhv. Figur 27 og Figur 28, blev der med et oscilloscope målt på databussen. Efter at først `write` signalet blev aktiveret, og dernæst `read` signalet, kunne hver bit, ét ad gangen, verificeres med oscilloscopet. Resultatet var som simuleringen, at RAM-controlleren virkede. RAM-controlleren okkuperede, ifølge udviklingsværktøjet fra Xilinx, 67% af CPLD'ens lagerkapacitet.

9.1.3.11 Delkonklusion 1. RAM-controller

Første udkast til RAM-controlleren er fremstillet, simuleret, og implementeret i det fremstillede test-PCB. Både simuleringen og testen i PCB'et afslørede at den fungerede som forventet. Da der nu er vished for at konceptet er gangbart, kan der fortsættes med at forfine RAM-controlleren yderligere, med henblik på den endelige implementation. Det er tydeligt at CPLD'en på test-PCB'et ikke har kapacitet nok til DFU'ens komponenter, da bare RAM-controlleren okkuperede 67%.

9.1.3.12 Endelige RAM-controller

Med udgangspunkt i den 1. RAM-controller, er den endelige version udviklet. Strukturen er den samme, men datasignalerne er udvidet fra otte til 64 bit, og alle otte DQMB-signaler er implementeret. Til erstatning for test-PCB'et og CPLD'en, er det besluttet at anskaffe en FPGA, som med sikkerhed har rigelig plads til både RAM-controller, modtagelse af NOAA-data, samt interfacet til transmissionsdelen. Der er valgt en Xilinx FPGA (Spartan XC2S100-5), med 208 benforbindelser (i en PQ208 pakning). Som nævnt tidligere, kræver en FPGA en ekstern EPROM til lagring af sit program, medmindre at FPGA'en programmeres hver gang strømforsyningen tilsluttes. Det har ikke været muligt at fremskaffe den fornødne EPROM, da den var i restordre hos leverandørerne. Derfor er der gjort plads til EPROM'en på det endelige PCB. Under udviklingsfasen af VHDL-koden har det ikke været et problem at programmere FPGA'en uden EPROM. Det er naturligvis nødvendigt at den monteres, hvis DFU'en skal være operationel hos DMI. Diagram og layout af det endelige PCB kan ses i bilag.

9.1.3.13 Adressering af RAM-modulet

Til adressering af RAM-controlleren, er det besluttet at udvikle to tællere, til hhv. skrive- og læseadresse. For at adressere alle 128 Mbytes, i det valgte RAM-modul, kræver det tællere på 24 bit, da hver adresse i RAM indeholder otte bytes. Hvis det skulle være muligt at isætte et RAM-modul med vilkårlig lagerkapacitet, ville det kræve at de to tællere kunne "omprogrammeres", delt til større eller mindre bitbredde, og dels omfordeling af række, kolonne, og bank bit (da der ikke nødvendigvis er lige mange rækker / kollonner i hvert RAM-modul). Det blev besluttet, kun at implementere tællere med fast bredde, dvs. lavet specifikt til det valgte RAM-modul. De to tællere er implementeret som en FIFO-struktur, med status signalerne RAM Full og RAM Empty. RAM Full bruges til at forhindre at data i RAM bliver overskrevet, i tilfælde af at de bliver fyldt.

Det gøres ved at deaktivere modulet til modtagelse af NOAA-data, og derved skriveoperationen til RAM. Dvs. idet RAM Full detekteres, deaktiveres NOAA modtagelsen, hvilket betyder at data preller af.

Det er udelukkende NOAA modtageren der inkrementerer skrivetælleren, og skriver til RAM. Inkrementering af læsetælleren, sker ved at skrive en vilkårlig værdi til adresse 0x102 (fra *SC13*). Den oprindelige idé var at lade læseoperationen fra *SC13* inkrementere tælleren (ligesom i en FIFO-buffer), men det viste sig nyttigt selv at kunne kontrollere den i udviklingsfasen, da det gav mulighed for at læse gentagne gange fra samme adresse.

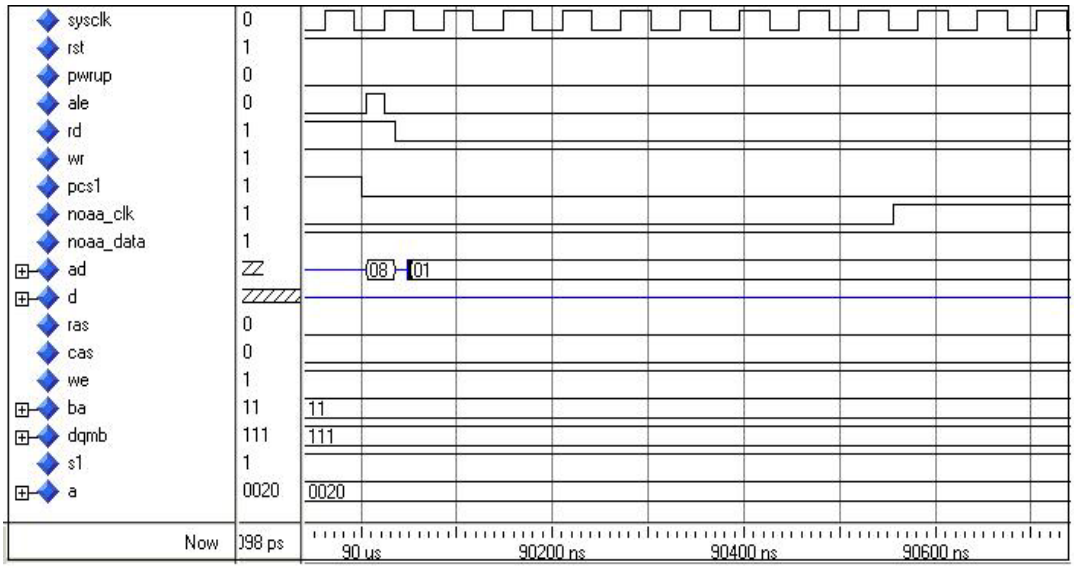
9.1.3.14 Clock frekvens til RAM og RAM-controller

Der valgt en clockfrekvens

9.1.3.15 Simulering af endelige RAM-controller

Den endelige RAM-controller med læse- og skriveadressering, samt interface til mikrokontrolleren, simuleret med den valgte FPGA. Testbenchen kan ses i bilag.

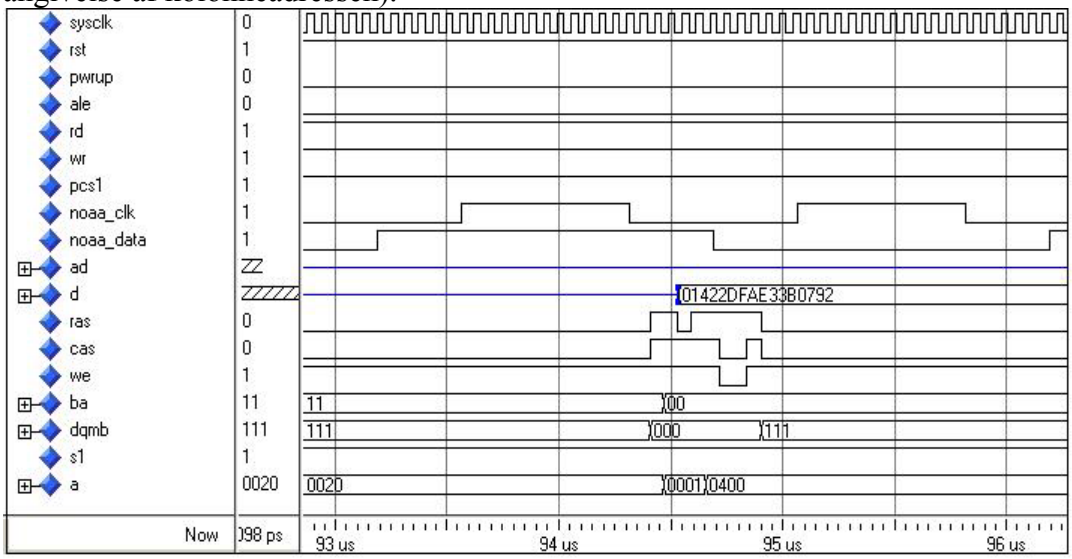
Figur 29 viser en læseoperation initieret af mikrokontrolleren. Den læser på adresse 0x108. På denne adresse er der placeret tre statussignaler fra RAM-controlleren, hhv. `RAMBusy`, `RAMFull`, og `RAMEmpty`, hvor sidstnævnte er placeret på LSB. På signalet `ad`, placeres først adressen (08), samtidig med at `ale` aktiveres. Efterfølgende placeres resultatet af læseoperationen på `ad` (0x01). Det ses at RAMEmpty er aktiveret, dvs. der er ikke skrevet til RAM endnu.



Figur 29 Simulering af den endelige RAM-controller. MCU læser på adresse 0x108 (RAM Empty).

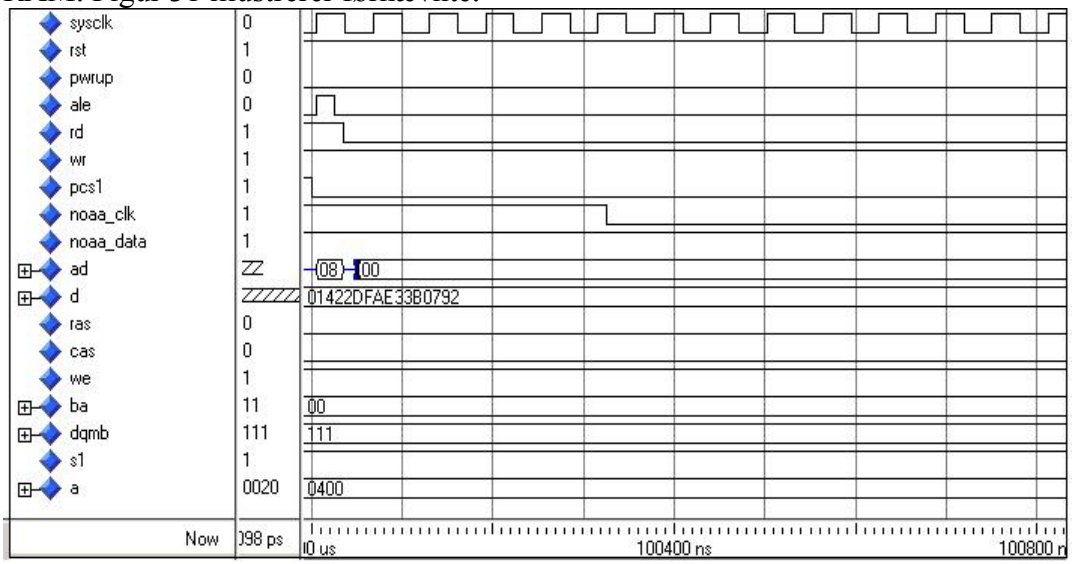
Figur 30 viser NOAA-modtagedelen der initiere en skriveoperation i RAM-controlleren. Der skrives 64 bit, hvor værdien er placeret på `d`. Der

skrives på række 1, kolonne 0, og bank 0. Bemærk at kolonneadressen bit 10 på kolonneadressen er sat (0x400). Dette indikerer at der er tale om en skriveoperation med autoprecharge (kun de nederste otte bit bruges til angivelse af kolonneadressen).



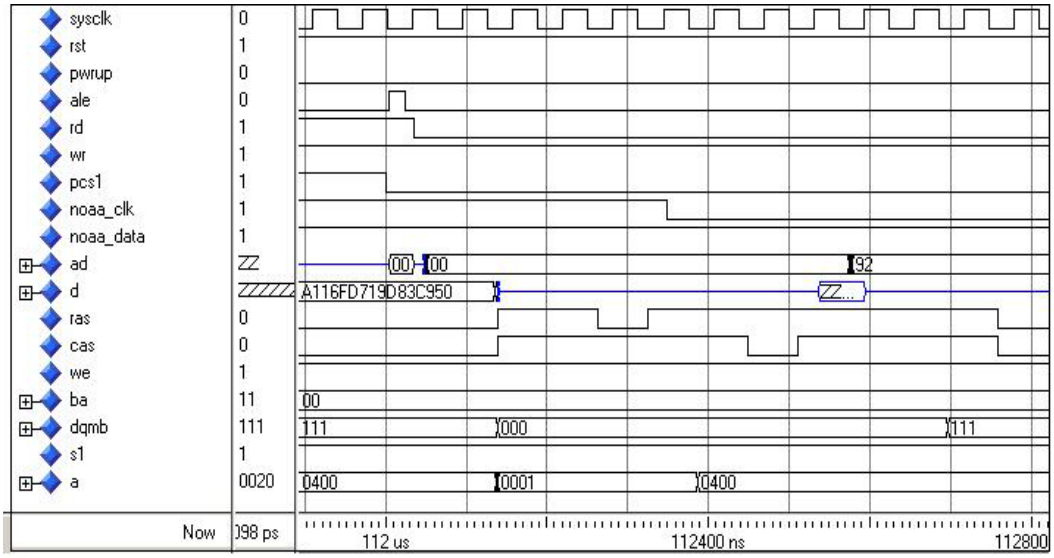
Figur 30 Simulering af den endelige RAM-controller. 64 bit NOAA-data skrives til RAM.

Der er nu skrevet NOAA-data til RAM. Nu initieres der atter en læseoperation af mikrokontrolleren, igen på adresse 0x108. Værdien er nu 0x00. Dvs. `RAMEmpty` er ikke længere aktiveret, og der er altså data i RAM. Figur 31 illustrerer førnævnte.



Figur 31 Simulering af den endelige RAM-controller. MCU læser på adresse 0x108 (RAM Not Empty).

Figur 32 viser en læseoperation fra RAM, initieret af mikrokontrolleren. Der læses på adresse 0x100, og resultatet er 0x92. Det var netop værdien som NOAA-modtagerdelen skrev før (vist på Figur 30).



Figur 32 Simulering af den endelige RAM-controller. MCU læser på adresse 0x100. Der initieres en læseoperation fra RAM.

9.1.4 Transmissionsdel

9.1.4.1 Indledning

For at lette implementationen af transmissionsdelen, er der anskaffet et evalueringskit af typen *DK40*, med den valgte mikrokontroller *SC12*.
Umiddelbart kan transmissionsdelen opdeles i to dele:

1. Kommunikation over internettet (MPLS-netværk)
2. Interface til CPLD / FPGA (*SC12 interface* på Figur 15).

9.1.4.2 Kommunikation over internettet

Den første test af transmissionshastigheden for SC12 viste at den havde problemer med at følge med NOAA-dataraten (ca. 83 kbytes per sekund). Testprogrammet, som var skrevet i C, havde ingen langsomme rutiner, som f.eks. udskrift til skærmen (f.eks. med *printf*), eller indlæsning af data fra I/O.
Alligevel var det kun muligt at komme op på ca. 96 Kbytes per sekund. (se testprogram i bilag). Efter at have nærstuderet dokumentationen af *SC12* viste det sig at dens maksimale transmissionshastighed var for lav, ifølge leverandøren. Det må konkluderes at *SC12* var et fejlkøb, som primært skyldes manglende forberedelse.

Erstatningen blev fundet hos samme leverandør. Modellen *SC13* (modellen større end *SC12*) har et 10/100BASE-T-interface (i modsætning til *SC12's* 10BASE-T), Een er den pin-kompatibel med *SC12*.

Det udviklede C-program kan ses i bilag, eller på den medfølgende CD-ROM.

9.1.4.3 Interface til FPGA

Interfacet mellem mikrokontrolleren og FPGAén skal muliggøre at NOAA-data kan hentes fra RAM-modulet og over i *SC13*, til transmission over internettet. I RAM-modulet skrives og læses der 64 databit ad gangen (for hver adresse), mens der i *SC13* kun kan skrives og læses otte bit ad gangen, for hver adresse. Derfor allokeres der otte forskellige I/O-adresser på *SC13* til indlæsning af data jf.

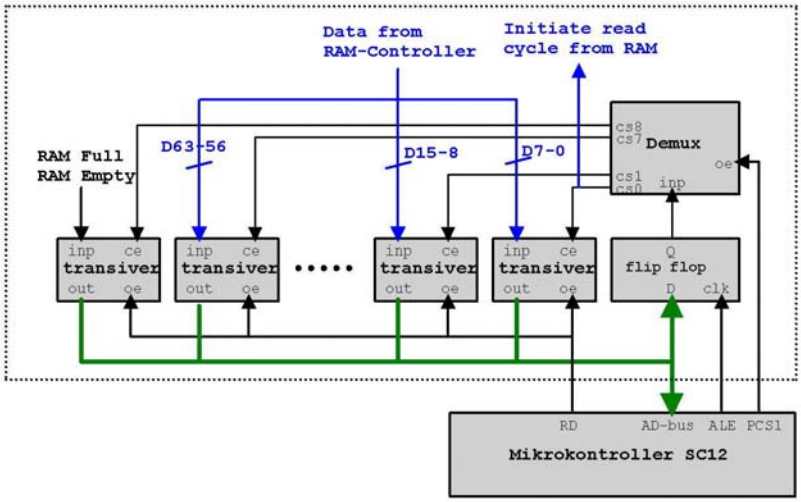
I/O-adresse	Internet chip select	Funktion
0x100	cs0	Databit 7-0 + RAM Read cycle
0x101	cs1	Databit 15-8
0x102	cs2	Databit 23-16
0x103	cs3	Databit 31-24
0x104	cs4	Databit 39-32
0x105	cs5	Databit 47-40
0x106	cs6	Databit 55-48
0x107	cs7	Databit 63-56
0x108	cs8	Bit 0= RAM Empty Bit 1= RAM Full

Tabel 4. Bemærk at adresse 0x108 indeholder information om mængden af data i RAM. Kun de to mindst betydende bit benyttes. Hvis der f.eks. læses en værdi på 0x02 på adressen, betyder det at RAM er fyldt.

I/O-adresse	Internet chip select	Funktion
0x100	cs0	Databit 7-0 + RAM Read cycle
0x101	cs1	Databit 15-8
0x102	cs2	Databit 23-16
0x103	cs3	Databit 31-24
0x104	cs4	Databit 39-32
0x105	cs5	Databit 47-40
0x106	cs6	Databit 55-48
0x107	cs7	Databit 63-56
0x108	cs8	Bit 0= RAM Empty Bit 1= RAM Full

Tabel 4 Allokering af I/O-adresser på *SC13* til indlæsning af NOAA-data fra RAM.

Signalerne fra *SC13* der benyttes til interfacet er AD0-AD7 (adresse-databussen), adresse latch signalet (ALE), read cycle signalet (RD), og et chip select signal (PCS1). PCS1 aktiveres så snart en læse- eller skriveoperation fra *SC13*, i adresseområdet 0x100 – 0x1FF, initieres. Timingsdiagrammet for signalerne kan ses i databladet for *SC13* (se bilag).
Interfacet er implementeret vha. otte bustransivere, en demultiplekser, samt en flipflop, som illustreret på Figur 33.



Figur 33 Implementation af interface til mikrokontroller

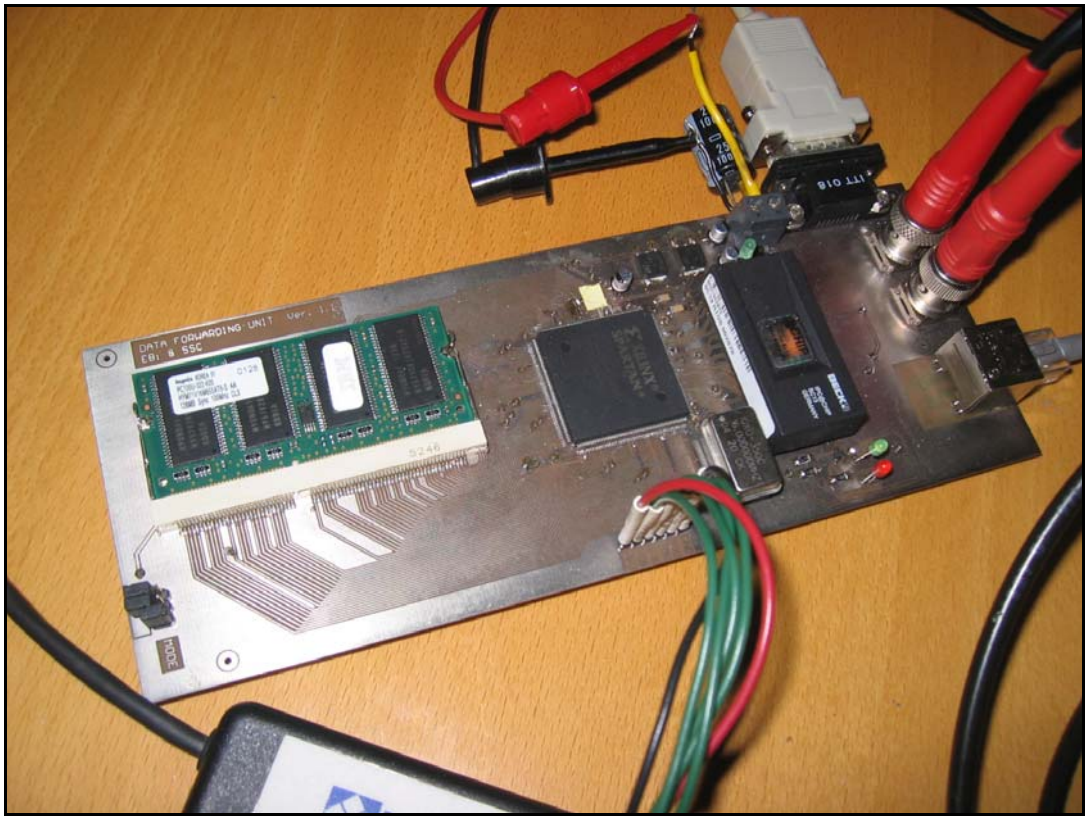
Flipflop'en har til opgave at gemme adressen på adresse- databussen, idet der læses eller fra *SC13*. Der er en valid adresse tilstede på den faldende flanke af ALE-signalet. Derfor er ALE forbundet til clock-signalet på flipflop'en (som latcher på den faldende flanke). Udgangen af flipflop'en (Q) er forbundet til en demultiplekser, der styrer signalerne cs0 – cs8, jf. Tabel 4. Dvs. kun adresser fra 0x100 til 0x108 aktiverer ét chipselect signal i demultiplekseren. De otte bustransiver's ce-signaler (chip enable) er hver forbundet til et af chip select signalerne fra demultiplekseren. Dvs. en læse- eller skriveoperation fra *SC13*, på adresse 0x100 aktiverer den første bustransiver, på adresse 0x101 den anden, osv. Bemærk at ikke alle otte bustransivere er skitseret på Figur 33). For at undgå at en skriveoperation aktiverer en bustransiver (og muligvis kortslutter databussen), er RD-signalet fra *SC13* forbundet til transivernes oe-signal (output enable).
For at læse data fra RAM-interfacet (se Figur 15) er alle 64 databit herfra forbundet til hver sin bustransiver (et til syv). Den ottende er kun forbundet til to signaler, RAM Full og RAM Empty.
For at læse data fra RAM, er det nødvendigt først at initiere en RAM læseoperation, hvilket vil opdatere de 64 databit til bustransiverene. Derfor er cs0 (chip select signalet associeret med adresse 0x100) også forbundet til RAM-interfacet. Dvs. en læseoperation på adresse 0x100 initierer en RAM-læseoperation, og aktiverer den første bustransiver, hvilket betyder at data fra RAM placeres på adresse- databussen på *SC13*.

Transmission over internettet

9.2 Samlede DFU

PCB-et af den samlede DFU kan ses på figur 34.
Diagrammer og PCB-layout kan findes på den vedlagte CD-ROM. Endvidere kan den fulde VHDL-kildekode også findes her.

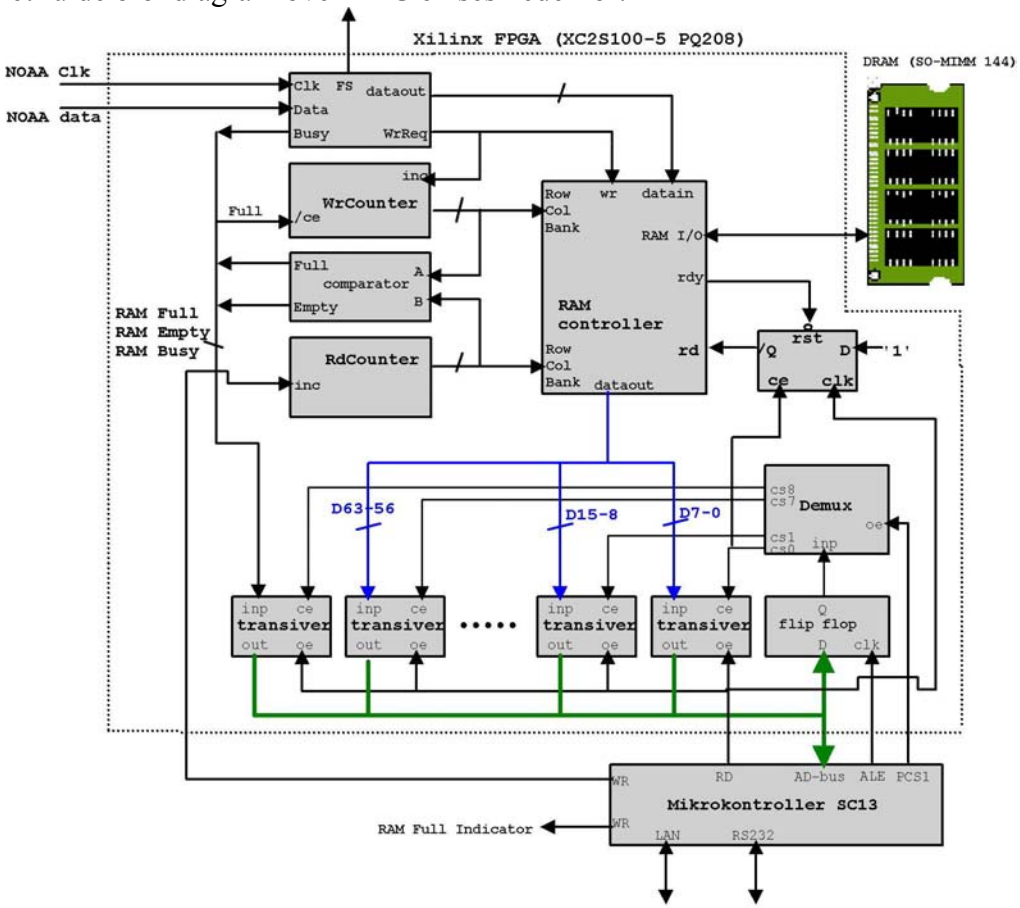
DFU'en fungerer som forventet. Det er muligt at modtage NOAA-data, og gemme dette i RAM-modulet. Endvidere er det muligt at udlæse data igen til mikrokontrolleren, og derfra transmittere over internettet.



Figur 34 DFU-PCB

PCB layout og diagrammer kan ses i bilag, samt på den medfølgende CD-ROM.

Det fulde blokdiagram over DFU'en ses nedenfor.



10 Data Processing Unit (DPU)

10.1 Funktionalitet

Installationen på Lyngbyvej's har to overordnede opgaver den skal udføre. For det første at modtage data fra DFU, og for det andet at behandle data så DMI får de filformater de har brug for.

Der stilles følgende krav til datamodtagelsen:

- DPUen skal agere server. Dvs det er DFU der tager initiativ til at oprette en forbindelse, og derfor skal DPU til enhver tid være klar til at modtage en sådan forbindelse fra DFU.
- Data skal kunne modtages fra internettet gennem en fast IP-addresse og en port. Der skal anvendes TCP og IP.
- DFU/DPU-protokollen skal implementeres.
    - Internetforbindelse skal kunne afbrydes og genoptages mit under overførsel af data uden det fører til beskadigelse af de producerede datafiler.
    - Data fra uidentificerede klienter skal sorteres fra – dvs først når en klient har sent en pakke med typen DFU_ID og en valid Identitet, skal andre pakketyper behandles.
    - Satellitdata består af NOAA-frames som hver starter 60 synkroniseringsbit. DFU sender altid disse 60 bit i begyndelsen af en datapakke, og markerer pakken som DFU_SYNC type. Alt data der modtages fra en satellitpassage inden første DFU_SYNC-pakke skal ignoreres, og altså ikke gemmes sammen med resten af data. Når første SYNC-pakke modtages skal der oprettes en ny fil, mens senere sync-pakker skal behandles fuldstændig som almindelig data så længe man kan bedømme passagen til at være ufuldstændig
    - En satellitpassage anses for at være fuldstændig så snart der er modtaget en DATAEND pakke. Så snart en sådan pakke modtages lukkes den opsamlede datafil og der ventes på næste sync-pakke.
- Datamodtagelsen skal mindst have en maxhastighed på 2 gange NOAAraten – dvs. ca. 1 Mbyte på 6 sekunder.

DPU's anden store opgave er at producere behandlet data. Der skal leveres filer af følgende typer for hver eneste satellitpassage:

- En rå NOAA-fil med format som angivet i appendix

- 5 kanal-filer hvor data fra NOAA-råfilen er separeret i de respektive kanaler.

- Et geometrisk oprettet quickview billede som klart afbilleder satellitdata så man kan se hvor i verden billedet er taget, og nogenlunde hvad vejret er. Samt 5 konturfiler af de geometrisk oprettede data uden billedheader.

- Information om hver satellitpassage: angivelse af Satellit navn, tidstempel, antal modtagne NOAA-Frames samt en indikation af billedkvalitet.

Der stilles følgende yderligere krav til DPU:

- Kanalfiler og quickview billeder skal være færdige og tilgængelige under et minut efter at datatransmissionen er overstået.

- Al software skal kunne køre under Linux SUSE version 9.1

10.2 Løsningsmuligheder og rammer

10.2.1 Valg af programmeringssprog

Flere sprog blev overvejet men valget faldt ret hurtigt på almindelig c. Java er platform uafhængigt, og derfor var det med i overvejelserne, men umiddelbart viste det sig at eksekvere langsommere end c. Men vigtigste argument for at anvende c som programmeringssprog var at DMI i forvejen anvender c, og der er vigtigt at de let kan ændre i kildekoden.

10.2.2 Opdeling af software i funktioner

Overordnet skal softwaren have en modtagelsesdel, en behandlingsdel og en præsentationsdel. Præsentationsdelen skal være meget begrænset for DMI har i forvejen deres egne programmer som skal behandle de filer der skal leveres, og det er derfor vigtigt at det nye program kræver så lidt opmærksomhed som muligt. DMI skal blot kunne se hvilke filer der er til rådighed, og så skal de have mulighed for at ændre nogle få parametre i programmet en sjælden gang imellem. Af denne grund bliver programmets præsentationsdel begrænset til de filer programmet producerer, og brugeren skal kontrollere DPU alene gennem programkaldet og en ini-fil hvor nogle få parametre kan ændres.

10.2.2.1 En-programsløsning – kontra flere programmer som kører samtidigt.

Separation af kanaler kunne godt foregå allerede under datamodtagelse, og inden en hel fil var modtaget, men data kan under alle omstændigheder ikke anvendes før filerne er færdigskrevet, og separationen tager kun ganske få sekunder. Det tager lidt længere at lave quickview billeder (20-30 sekunder), men det er til gengæld mindre tidskritisk for meteorologerne. Derfor er det muligt at adskille den del af softwaren som behandler data og den del som modtager, således at behandlingsdelen først kaldes når modtagelsesdelen ved

at den har modtaget en komplet satellitpassage. På den måde undgår man at have mange filer åbne i længere tid, og man får en let overskuelig snitflade mellem de forskellige programdele.
En mulighed som var under overvejelse var at lade de forskellige funktioner køre autonomt i hver sit program, lade dem kommunikere indbyrdes gennem flade filer. Dette ville have den fordel at man fra computerens styresystem kunne holde øje med hvis enkelte funktioner fejlede, og kunne genstarte hver funktion uden at påvirke de andre. Til gengæld ville man få en tungere kommunikation mellem funktionerne, og alle funktioner ville være tvunget til at køre konstant og bruge processorkraft på at undersøge om der var ny kommunikation til dem.
Valget faldt på at lade hele konstruktionen køre i et program således at datamodtageren starter databehandlingsdelene når en komplet fil er blevet samlet.

10.3 Datamodtagelse

Softwaren som udgør DPU skal agere server. Dvs. det er DFU som tager kontakt og altså kontrollere hvornår kommunikation påbegyndes, og DPU skal derfor sørge for hele tiden at være tilgængelig for kontakt.

10.3.1 Opsætning på Internettet

For at være klar til at modtage data fra internettet skal man have reserveret en adresse både på internettet (en IP-adresse) og i computeren (en port), som klienter kan kontakte. Tilsammen udgør de en Socket. IP-adressen skal være kendt af klienten så den skal være offentlig og konstant.

Alternativt kan DMI vælge at lad serveren stå på et lokalnet, og altså at den ikke har en selvstændig IP-adresse, så kan man bruge en offentlig IP-adresse som lokalnettet deler ved at route al data som bliver sent til denne adresse med DFU/DPU-protokollens port (6796) ind til den server DFUserveren er installeret på. Dette forudsætter at der ikke er flere applikationer på lokalnettet der bruger samme port og selvfølgelig at der ikke skal kære flere version af DFUserveren på netværket.

10.3.1.1 Brug af velkomstsocket og forbindelsessockets

Den kendte port (6796) anvendes kun til oprettelse af forbindelsen mellem klient og server (velkomstsocket) – når først forbindelsen ér oprettet kan kommunikationen fortsætte på en anden port (forbindelsessocket) således at den kendte port frigives til at andre klienter kan kontakte. Således er det muligt at servicere mange klienter samtidigt, selv om man kun har én offentlig adresse. På nuværende tidspunkt har DMI kun brug for en enkelt DFU, og således ikke brug for at være i forbindelse med flere klienter samtidigt, men dels kan DMI senere få brug for at kunne modtage fra flere stationer, og dels giver brug af en særlig velkomstsocket den fordel at den kendte port bliver beskyttet mod fejl i dataoverførsel. Hvis der opstår fejl under overførsel af data således at klienten mister sin forbindelse til serveren,

så vil den brugte port være optaget i serveren ind til serveren detekterer at noget er gået galt. Ved brug af en velkomstsocket undgår man denne ventetid.

10.3.2 Tråde i Datamodtagelse

Brug af adskilt velkomstsocket og forbindelsessockets betyder ikke umiddelbart at man kan servicere flere samtidige klienter. For at kunne dette skal forbindelsessocketen åbnes i en ny tråd. Dette vil ikke give nogen fordel så længe der kun er tale om en enkelt mulig klient – først hvis der sættes endnu en klient op vil der være en gevinst. En alternativ måde at servicere to potentielt samtidige klienter vil være blot at lade de to klienter kontakte forskellige velkomstporte sådan at der kan køre to samtidige servere – dette vil selvfølgelig kræve at de to serverprogrammer er konfiguret forskelligt, så flertrådsløsningen er den bedste måde at gøre det på.

10.4 Databehandling

10.4.1 Separation i kanalfiler

NOAA satellitterne optager billeder af jorden i 5 forskellige frekvensområder kaldet kanaler. Disse fem kanaler ønsker DMI adskilt og lagt i hver sin kanal fil (appendiks).
Data fra satellitterne ligger i pakker kaldet frames. Hver frame indeholder en header og et datafelt, hvor headeren indeholder rammesynkroniseringen, et tidsstempel, Satellit ID og Datafeltet er delt op i pixels repræsenteret ved 10 bit for hver kanal. Dvs. de første 10 bit i datafeltet er første pixel i kanal et, de næste 10 bit i kanal 2 osv. Bit 51-60 er så anden pixel i kanal et og så videre.

Den rå satellitdata bliver sendt fra satellitterne kontinuert, og derfor er det nødvendigt at have en header med forholdsvist korte mellemrum (6 Hz) for man ved jo ikke hvornår brugeren begynder at lytte og han skal straks have oplysninger om præcist klokkeslæt og satellit ID.
Når man behandler data har man i princippet kun brug for første header. F.eks; hvis man har tidsstemplet fra første header kan man regne sig til tidspunktet for optagelse af hver pixel i resten af passagen. Derfor skal er der kun en enkelt header i hver kanalfil.

I NOAA-headerens tidsstempel er der kun angivet tid på året, men ikke et årstal. I de gemte filer er der brug for et årstal idet filerne jo kan gemmes fra år til år.
Man kan som regel bruge behandlingstidspunktets årstal – kun omkring nytår skal man korrigere for at årstallet kan have skiftet siden første header blev skrevet i satellitten.
Fejlen kan kun opstå for de satellitpassager som bliver behandlet et andet år end de bliver påbegyndt
Dvs. det kun ske ved årsskiftet, og kun for årets sidste passage, plus de passager som på det tidspunkt ligger gemt i DFU'ens hukommelse.

I kanalfilerne er der kun afsat 8 bit til hver pixel, hvor der i rå-data er 10 bit. Det vil sige at informationen fra rå-filen på en eller anden måde skal komprimeres fra 10 til 8 bit.
Man kunne bruge mange kompressionsmetoder i forsøg på at mindske datatabet, men kanalfilerne skal kunne læses af meteorologernes programmer og disse forventer at de to mindst betydende bit er skåret bort.

10.4.2 Kvalitetsberegning og information om passagen

I NOAA headeren er der tre felter med data som man kender i forvejen (frame sync, spare words, og auxiliary sync) – disse felter kan bruges til at beregne kvaliteten af den data man modtager fra satellitten. Man sammenligner simpelthen den modtagne data med det forventede, og hver gang mad modtager et andet bit end det man forventede så har man fundet en bitfejl.
Vi har valgt ikke at benytte framesync i kvalitetsberegningen idet, en fejl i framesync ville betyde at den ikke blev genkendt som en framesync. En framesync som ikke bliver genkendt i DFU, kan betyde at hele framen bliver smidt væk, og altså må man forvente en smule højere datakvalitet i framesync end i resten af filen idet nogle fejlbehæftede framesync bliver sorteret fra. Desuden er de 100 ord i auxiliary sync nok til at lave en god beregning.

Bittene i auxiliary sync bliver bestemt ved hjælp af polinomiet x^10 + x^5 + x^2 + x +1
Generatorbittene er alle 1 når bitgenereringen startes.

10.4.3 Quick-view (Geometrisk opretning)

For at gøre det let for brugeren at overskue hvad de forskellige NOAA-råfiler og kanalfiler indeholder, skal der laves billeder af data, som gør det let og hurtigt at se hvilken data man kan forvente at finde i hver fil. Tidspunkt og Satellit ID kan man se i filernes navne, men ting som om Danmark er synlig, er det nat eller dag eller er der mulighed for at se isbevægelser omkring Grønland, skal man kunne se på et quickview billede. Dvs. det er vigtigt at få placeret data på et verdenskort, og at landområder, skyer og hav let kan skilles fra hinanden. At placere af data på et landkort kaldes geometrisk opretning og det er en omfattende operation som kræver beregning af satellitposition og projektion af jordkloden på en plan.

10.4.3.1 Hvorfor Geometrisk Opretning?

Data i kanalfilerne kan let lægges ind i en billedfil dermed vises som et billede. Et sådan billede vil vise et forvredet billede af jorden set fra satellittens synsvinkel. Billedet bliver vredet dels fordi når satellitten skanner jordoverfladen fra side til side så vil punktet lige under satellitten være nærmere satellitten end alle andre – dermed vil dette punkt se størst ud på billedet, og dels fordi satellitten bevæger sig i en ellipse (tilnærmet) og dermed er dens afstand fra jorden ikke konstant gennem hele billedet.

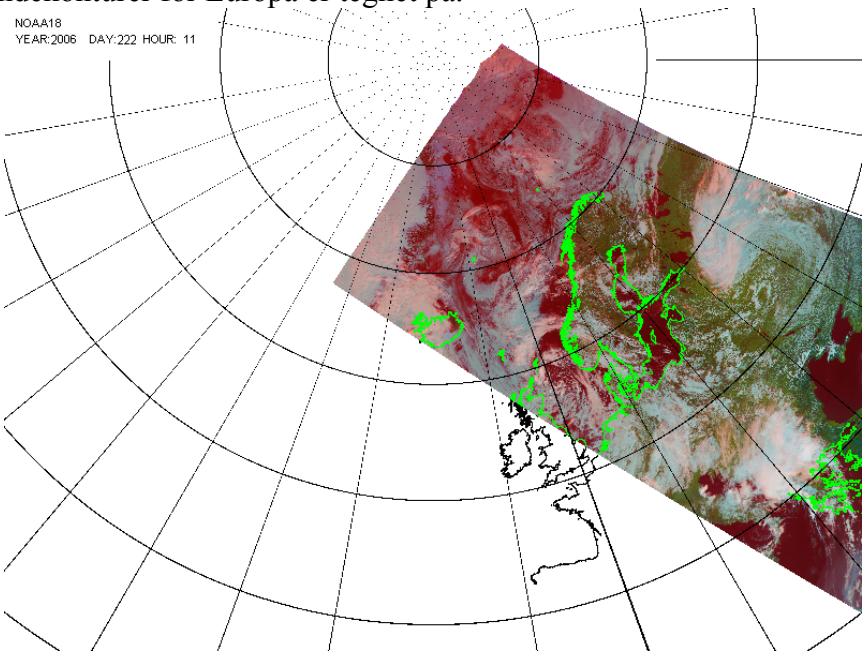
Kanalfil-billedet vil vende sådan at den øverste del af billedet er den del satellitten passerer sidst. En satellit i en bane som passerer over polerne vil bevæge sig mod nord i halvdelen af banen og mod syd den anden halvdel – da jorden drejer om sin akse uafhængigt af satellittens bevægelse vil det altså skifte om et billede af Danmark vil have nord opad eller nedad. NOAA-satellitterne ligger som regel ikke i baner som passerer direkte over polerne og bevæger sid dermed ikke præcist Nord-Syd – dvs. opad kan være en hvilken som helst retning på jordkloden.

Forskellige satellitpassager vil vende forskelligt og være vredet forskellig alt efter satellittens bane og bevægelsesretning, hvilket vil besværliggøre en umiddelbar identifikation af det afbillede område.

Nedenfor ses et billede dannet af de uoprettede kanalfiler (kanal1,2 og 4)

For at vende og vride billedet så alle passager kan lægges ind på samme let genkendelige kortudsnit udfører man geometrisk opretning. Dvs. man tager hvert enkelt pixel i kanalfilen og regner ud hvor på jordkloden det hører til – regner deres længde og bredde grader ud. Derefter projekterer man jordkloden ind på en flade (et kort) sådan at man af kanalfilerne laver billeder som passer på landkort med den valgte projektion.

Nedenfor vises en geometrisk oprette version af satellitdata. Det er samme satellitpassage der afbilledes som i billedet ovenfor. For at øje overskueligheden er her desuden lagt længde,breddegradslinjer ind, og landekonturer for Europa er tegnet på.



10.4.3.2 Beregning af pixels position i længde og bredde

Det der er målet med geometrisk opretning er at kunne placere et pixel fra en NOAAråfil på et landkort. Hvert pixel er udtryk for et punkt som satellitten ser i en bestemt retning.
For at komme frem til et pixels præcise position skal man altså først beregne satellittens position i det øjeblik pixlet er blevet "optaget". Derefter skal man beregne hvor satellitten ser hen, og til sidst projektere positionen fra dens position på jordkloden ned på et landkort.

10.4.3.3 Himmelkoordinater

Længde og breddegrader er koordinater som angiver en position i forhold til jordens drejning – men jorden drejer rundt om sig selv, og satellitter følger normalt ikke denne drejning. Derfor har man brug for at beskrive satellitternes bevægelse i forhold til stjernerne.
Man forestiller sig at himlen er en uendelig stor kugle med os i centrum, og så bruger man to vinkler på samme måde som jordens længde og bredde. Man har syd og nordpol samt ækvator for denne himmelkugle. Nordstjernen markerer ret præcist nordpolen. Deklination er hvad der i himmelkoordinater svarer til bredde (positiv nord for ækvator og negativ syd for).
Rectascension svarer til længde – hvor man har defineret et punkt på himlen til at være nulpunkt ligesom greenich er det for længdegrader. Dette punkt kalder man vædderpunktet.

10.4.3.4 Tilnærmet beregning af satellitbaner og forklaring af baneelementer

En ellipse er en udmærket tilnærmelse til en satellitbane. En satellits bane om en punktformet masse vil være en ellipse – en satellitbane omkring jorden er så tæt på en ellipse at man kan beskrive satellitbaner som ellipser der drejer. Så længe der ikke er træk i satellitten – dvs. man når man ser bort tab i bevægelsen fra f.eks. luftmodstand så vil ellipsebanen beholde samme størrelse og excentricitet.

For at tillade brugere af satellitterne at beregne satellitterne position er NOAA-satellitternes baneelementer frit tilgængelige på Internettet. Baneelementerne beskriver satellittens bane ved at definerer den ellipse som satellitten befinder sig i, på et givent tidspunkt.

Ellipsens ene brændpunkt er bestemt af jordens tyngdepunkt. Det plan ellipsen ligger i bestemmes ved to vinkler:

10.4.3.4.1 Inklination

Det punkt hvor satellitbanen krydser ækvator i nordlig retning kaldes Ascending Node.
Inklinationen er den vinkel satellitbanen danner med jordens ækvator i Ascending (vinklen nord for Ækvator og til højre for bevægelsesretningen). Man kunne også kalde inklinationen for vinklen mellem ellipsens plan og Ækvatorplanen, så inklinationen er den samme der hvor banen krydser ækvator i sydlig retning

10.4.3.4.2 Right Ascension of Ascending Node (RAAN)

Den anden vinkel som definerer bane-ellipsen er længdegraden af Ascending Node.

Hvordan ellipsen vender i planen bestemmes også af en vinkel:

10.4.3.4.3 Perigee's argument

Med mindre ellipsen er en cirkel, så har den to endepunkter. Det ene endepunkt er der hvor afstanden til jorden er størst. Det kaldes Apogæ. Det andet er der hvor afstanden til jorden er mindst og det kaldes Perigæ. Perigæ's argument er vinklen mellem Perigæ og ascending node, som set fra jordens centrum. Dvs. vinklen mellem de to linjer som går igennem henholdsvist centrum og Perigæ, og centrum og ascending node.

10.4.3.4.4 Excentricitet

Definér Excentricitet – med ligninger.

Excentriciteten bestemmer ellipsens form. En excentricitet på 0 indikerer en cirkel mens en excentricitet på 1 indikerer en streg.

10.4.3.4.5 Mean Motion (Revs per day)

Sidst skal ellipsens størrelse beskrives – den hastighed satellitten har når den passere jordens ækvator i nordlig retning afgør hvor stor ellipsen bliver. Mean motion angiver satellittens gennemsnitshastighed i antal omgange per dag.

10.4.3.4.6 Epoch

Eftersom satellitten ikke bevæger sig i en perfekt ellipse, vil satellitbanen ikke vedblive at kunne beskrives med den samme ellipse og derfor er det præcise tidspunkt banen lå på den beskrevne ellipse. Epoken giver dette tidspunkt i år

10.4.3.4.7 Middel Anomali

I det øjeblik den angivne ellipse passer på satellittens bane befinder ellipsen sig et sted på ellipsen. Middel anomalien angiver dette sted i grader fra Perigee (det sted på ellipsen som er nærmest jorden).

10.4.3.5 Bevægelse i banen

Vi har ikke blot brug for at kende satellitbanen / ellipsen som satellitten bevæger sig i, men for at vide hvor på ellipsen satellitten befinder sig til tidspunktet T. Dette sted angives entydigt ved vinklen V, kaldet satellittens anomali, som er satellittens øjeblikkelige vinkel på sin bane i forhold til perigæ.
Satellittens anomali kan findes ved at løse Keplers ligning:

Keplers ligning :
(T-TEPOCH)*GMY+GMA = GMM = E-EXC*sin(E)

Her er T det tidspunkt man er interesseret i mens Tepoch er det tidspunkt baneeelemnterne gælder. GMY er er satellittens gennemsnitlige daglige bevægelse, GMA er er middelanomalien til tiden Tepoch og GMM er middelanomalien.
V indgår kun i ligningen gennem E – en vinkel kaldet den excentriske anomali.

Den excentriske anomali relaterer sig til V (satellittens "sande anomali" – satellittens øjeblikkelige vinkel på sin bane i forhold til perigee) _

tan(V/2) = kvadratrod((1+exc)/(1-Exc)) * tan(E/2)

eller

tan(V/2) = XI * tan(E/2)

Keplers ligning kan ikke løses direkte, men skal løses ved successive approksimationer.

10.4.3.6 Bevægelse i rummet (ellipsens drejning)

Banens inklination holder sig ret præcist konstant, men RAAN og Perigee's argument flytter sig bl.a. pga. jordens form og uregelmæssigheder i dens massefordeling.

10.4.3.7 Beregning af pixel's koordinater ud fra satellittens position samt synsvinkel

Når man har satellittens præcise position kan man beregne hvor den ser hen i det øjeblik den optager data til et pixel i billeddata. Satellittens scanner frem og til bage på tværs af dens bevægelsesretning, og den vinkel den har i forhold til lodret ændrer sig helt konstant med tiden, så kender man tidspunk kan man beregne vinklen og der fra beregne hvilket sted på jorden der bliver betraktet

10.4.3.7.1 Datum (hensyntagen til jordens fladtrykthed)

Jorden er ikke helt kugleformet, men nærmere elipsoid – lidt fladtrykt, sådan at jorradius til ekvator er lidt større end til polerne. Dette betyder at man får en lille fejl i den beregnede billedpixlernes position når man antager at jorden er kugleformet. Datum er det den aproximation til jordens form man anvender, og der er forskellige alt efter hvor på kloden man ønsker den skal være præcis.

- **Projektion**

Billeddata skal ikke tegnes på en rund klode, men på et plan, så der kloden skal projekteres ind på en flade. Der er mange måder at gøre det, men vi har valgt at bruge en Polær Stereografisk Projektion. Den har vi valgt dels fordi den giver gode en god afbilleding af koordinater i nærheden af nordpolen, og fordi DMI er vandt til at se deres billeder i denne projektion.

For at forklare Polær Stereografisk Projektion skal man forestille sig et plan som tangerer jorden på nordpolen. Lader man så en linje gå gennem sydpolen og det punkt man ønsker at projekterer. Punktet bliver så projekteret op på planen der hvor linjen krydser planen.

Første skridt i implementering af softwaresystemet er identifikation og opdeling af opgaver.

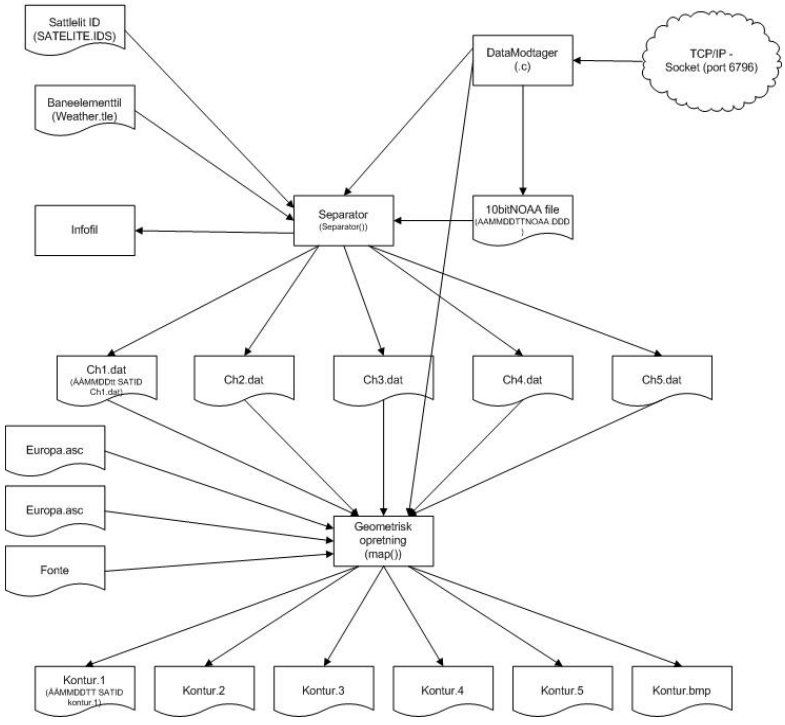
10.5 Opdeling af opgaver i funktioner

I implementeringen af DPU deles software grundlæggende delt op i tre hovedfunktioner.

1. En Datamodtager som sørger for kommunikationen med internet og DFU og som skriver NOAA-rådatafilen. Det er også datamodtageren som starter de to andre hovedfunktioner når den har afsluttet modtagelsen af en satellitpassage fra DFU
2. En kanal-separator (`seperator()`) som læser de råfiler datamodtager skriver og deler data op i 5 kanaler og skriver de 5 kanalfiler.
3. og Map() som laver quickview billeder at satellitdata ved at udføre geometrisk opretning på kanalfilerne. Dvs. `map()` beregner de korrekte længde- og breddegrader satellitbilledernes punkter afbilleder. Dette gøres på grundlag af beregninger af satelliternes baner og hvilken retning satellitten "ser" i på det tidspunkt det pågældene punkt blev optaget.

Programmets primære kontakt til brugeren går gennem filer, som det forklares i næste afsnit. Følgende figur viser de tre hovedfunktioner og hvilke filer de skriver eller læser.

Overordnet Programstruktur af DFU Receiver



10.6 Valg af Brugerflade og Programmeringsprog

Brugerflade

DMI ønsker ikke noget grafisk brugerflade, for programmet skal kunne køre automatisk, mest muligt uden menneskelig indblanding. Dvs. de sjældne gange der skal ændres på parametre eller at programmet bare skal genstartes er de godt tilfredse med at bruge en kommandolinje.

Brugerne er først og fremmest interesserede i de filer programmet producerer. Det vil sige filerne skal navngives og lægges så brugerne let kan finde dem og se hvad de indeholder.

Filstruktur

Der skal mindst produceres syv filer hvor hver satellitpassage: en råfil, fem kanalfiler, fem konturfiler, og et quickviewbillede. Desuden ønskes der præsenteret oplysninger om datakvalitet, modtagne antal frames, satellitidentitet og tidstempel – til dette laves endnu en fil: infofilen.

Råfilen er den eneste af filerne som er pålagt en bestem navnekonvention fra DMIs side. Den er ÅÅMMddttmm.SATID.DDD, hvor:

ÅÅ:	De sidste to cifre i årstallet	00-99
MM:	Måned i året	00-12
dd:	dato	00-31
tt:	time på dagen	01-24
SATID	Satellittens ID hentet fra baneelementfilen. F.eks.: NOAA17	
DDD	Dag på året i tre cifre	001-366

De andre fire filtyper navngives derfor på lignende vis:

Kanalfiler:	ÅÅMMddttmm.SATID.DDDCHN.DAT
Konturfiler:	ÅÅMMddttmm.SATID.DDDkontur.N
Quickview:	ÅÅMMddttmm.SATID.DDD.kontur.bmp
Infofil:	ÅÅMMddttmm.SATID.DDD.INF

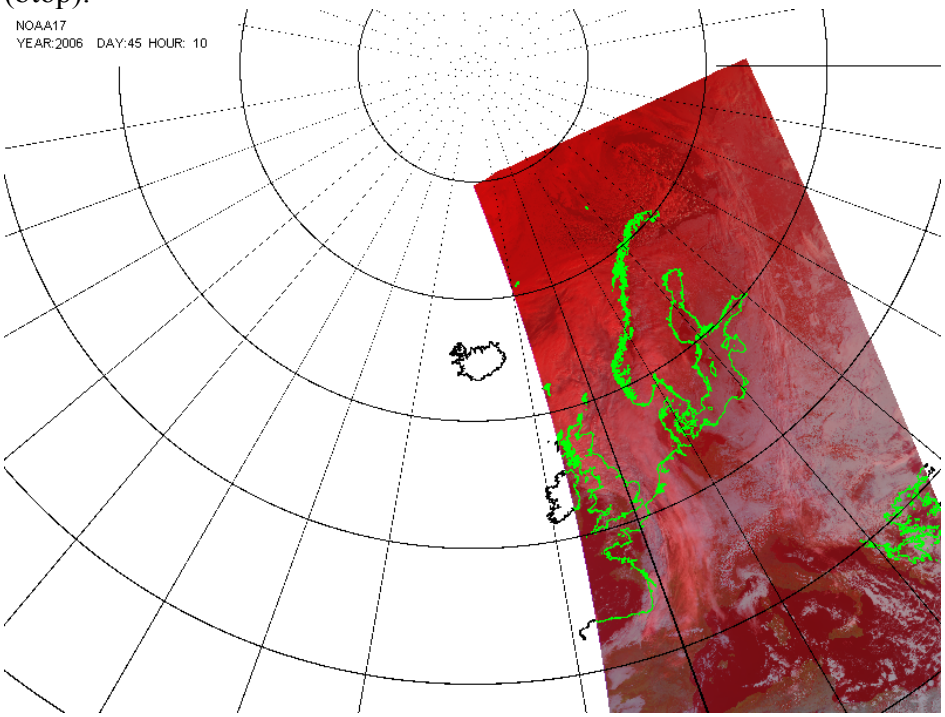
Programeringssprog

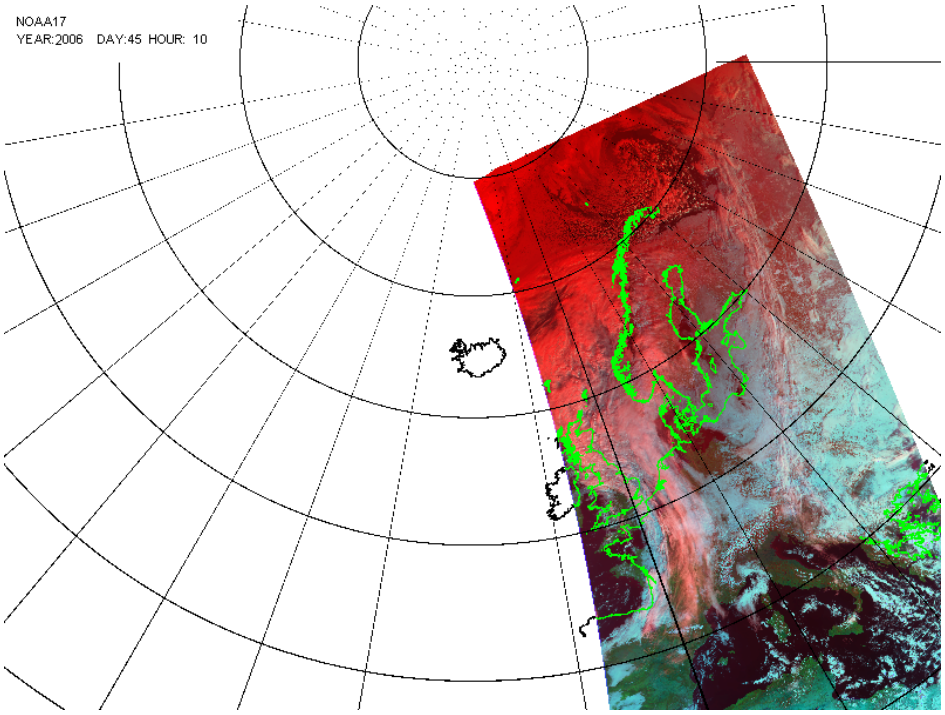
Vi har valgt at programmere i C hovedsageligt fordi det er det programeringssprog DMI's programører foretrækker. Desuden lavede vi tidligt en test-datamodtager i Java, og den kørte betydeligt langsommere end den tilsvarende programeret i C.

Quickviewbillederne skal bruges til at give brugerne overblik over filerne. Brugeren skal let og hurtigt kunne finde data som dækker de ønsker han har til tidspunkt, område og vejrtype.

Hvordan vælges billedudsnit
Når NOAA-data skal placere på et kort, så skal man vælge hvilken del af verden man ønsker at se – og hvilket zoomnivo. Man kunne vælge at lade Det modtagne data afgøre billedudsnittet, sådan at man altid så et område af verden som indeholdt al den modtagne data, i en zoomnivo, så der ikke var for meget spilplads. Ulempen ved dette valg ville være at mange billeder af denne type ville være besværlige at få et overblik over da man ville være nød til at genkende landområder for at finde ud af hvilken del af verden billedet afbillede.
Alternativet, at vælge et fast billedudsnit for alle satellitpassager er at enten viser man hele verden, og derved kommer satellit-data kun til at fylde en ganske lille del af billedet, eller også vælger man en del af verden og risikerer at satellitdata hører til uden for billedudsnittet, og derved måske slet ikke bliver afbilledet. MEN vi ved at data er hentet i en jordstation i Nordsjælland som kun kan se satellitter i en vis omkreds, hvilket igen begrænser det område satellitten kan se mens den transmitterer data til stationen – så det område billeddata kan afbillede begrænset til en afstand fra stationen. Desuden er DMI hovedsageligt interesseret i det danske og grønlandske vejr foruden is-situationen omkring Grønland. Vejret i Dk kommer fra vest – dvs. et område vest for Danmark og over Grønland vil være det relevante område at se.

Opløsningen af billedet er arbitrært valgt til 800pixel X 600pixel. Dette giver et billede som på den ene side er lille nok til ikke at optage væsentligplads når man sammenligner med rå-filerne (som typisk fylder i størrelsesordenen 70 Mb) og samtidig er stort nok til man kan genkende landmasser og få et indtryk af vejret på satellitdata.
Billedudsnittet beskrives ved længde og breddegrader af bitmapbilledets centrum (lmid og bmid) samt breddegraden i toppen af billedet centrum. (btop).





Farver i quickview-billedet

NOAA-data's 5 kanaler repræsenterer forskellige frekvensspektre, så det er oplagt at benytte tre forskellige kanaler til at repræsentere de tre farver i et bitmap-billede. Den kompination som har vist sig at give det mest overskuelige resultat er 1,2 og 4. Dette valg giver også "naturlige" farver, sådan at skyer og is er nogenlunde hvide, land er grønt og vand blåt.

Inifil

Med et serverprogram som dette vil der altid være en del parametre som det kunne være ønskeligt for brugerne at kunne ændre. Feks. kunne det være en god idé at lade brugerne kunne ændre hvor outputfilerne bliver lagt på serveren.

En mulighed ville være at brugeren indtastede sine valg når han startede programmet som parametre. Dette ville kræve at brugeren enten kunne huske sine programsyntaxen eller brugte en hjælpefunktion eller manual.

En letter måde at løse problemet er ved at bruge en ini-fil. Her skal brugeren kun huske inifilens placering og så kan han læse i selve filen hvilke muligheder han har for at ændre parametre.

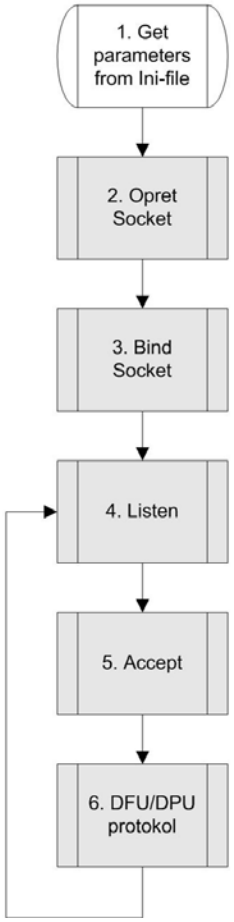
10.7 Gennemgang af software og flowdiagrammer

Herefter følger gennemgang af de tre hovedprogrammers virkemåde sammen med deres flowdiagrammer, og en kort beskrivelse af test som er udført på dem.

10.7.1 Datamodtager

Datamodtager består essentielt af to dele. Den ene del lytter på en velkomstsocket og opretter en forbindelsessocket så snart der er oprettet forbindelse til en klient.. Den anden del implementerer DFU/DPU-protokollen

Velkomstsocket (main program)



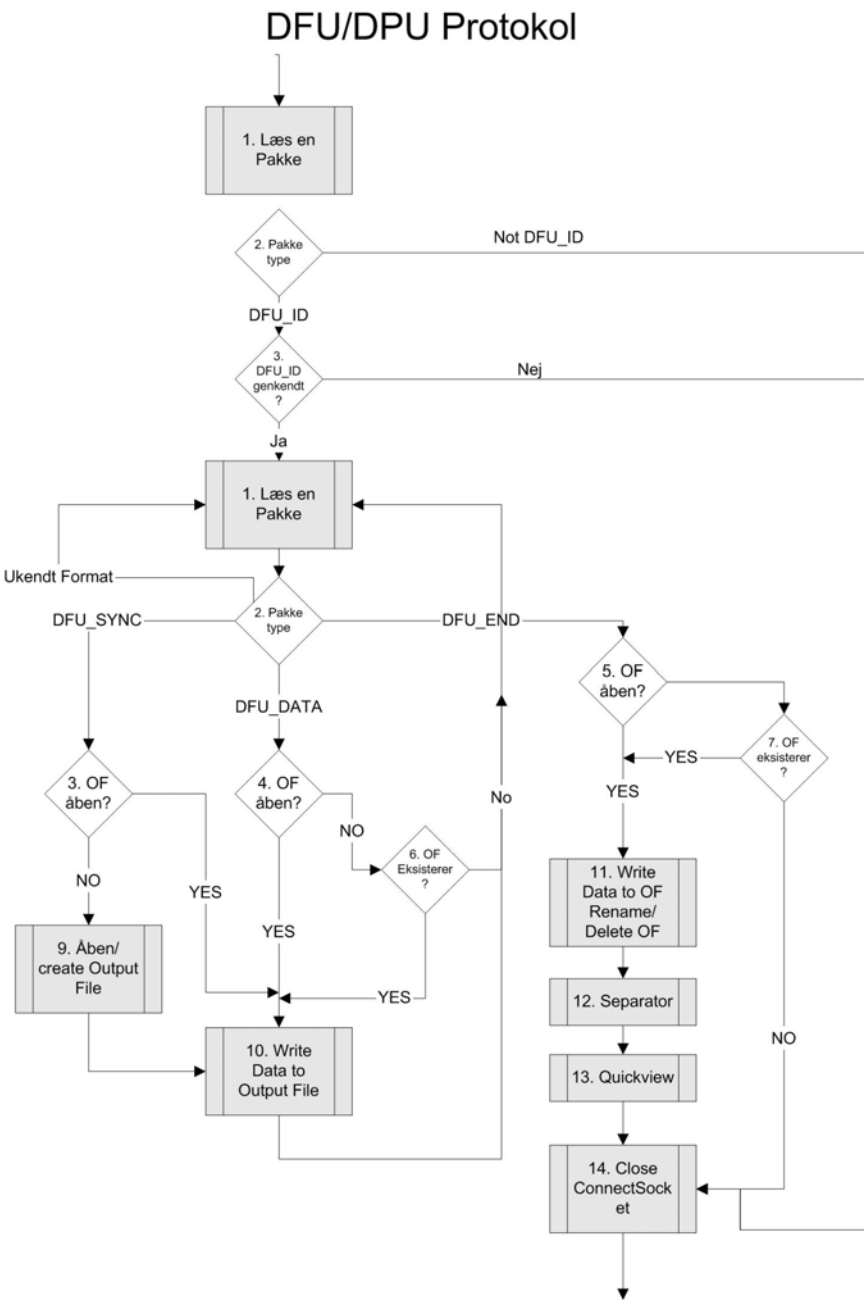
10.7.2

1. Det første der sker når programmet startes er at ini-filen åbnes og læses, og de læste variable sættes ind i en struktur (ini). Det drejer sig om variablerne:
   DFUID: DFU'ens serienummer i 12 cifre
   outdir: output directory)
   helpdir: help direktory hvor hjælpefilerne til at skrive labels og landekonture ligger

satdir: sat directory, hvor satellit indentificationfilen og baneelementfilen skal ligge)

tmidx: en halvquickview billedbredde i x-retningen, som SKAL være 400)
tmidy: En halv billedbredde i y-retningen – SKAL være 300
tlmid: Længdegrad af quickviewbilledets centrum. Standart 20
tbmid: Breddegrad af quickviewbilledets centrum. Standart 65
tbtop: Breddegrad af quickviewbilledets top i centrum. Standart 95
port: DFU/DPU-protokollens velkomstsokkel – værdien her skal selvfølgelig matches i DFU

2. Derefter Oprettes velkomstsokkelen
3. Sokkel bindes til den rette port (læst i ini-fil) og IP-adresse.
4. Så sættes programmet til at lytte på sockelen
5. og vente på at kunne acceptere en klientforbindelse på sokkelen.
6. Så snart der er oprettet en forbindelse til en klient behandles modtaget data efter vores vedtagne protokol (DFU/DPU protokollen)



10.7.3 DFU/DPU Protokol

1. Første Pakke i protokollen indlæses! Det foregår ved at programmet læser fem byte som header. Første byte fortolkes som pakketypen, de to næste som pakkenummeret, og de sidste to angiver datalængden i byte. Derefter forsøger programmet at læse datalængde bytes fra sokkelen. Hvis ikke sokkelens buffer indeholder returnerer sokkelen blot det antal byte der er til rådighed, så programmet fortsætter med at bede om data til det har modtaget hele datapakken.
2. Første Pakke skal altid være af typen DFU_ID. Dvs. første modtagne byte efter oprettet forbindelse skal være 0x02. Hvis typen er forkert, er protokollen forkert, og forbindelsen lukkes.
3. Hvis typen er OK checkes FDU'ens ID, og den sammenlignes med den ID der blev læst i ini-filen. Hvis der bliver brug for at kende mere end enkelt FDU skal

der ændres i læsning af inifilen som for øjeblikket læse FDU_ID ind i en variabel allerede når programmet startes. Med flere DFU'er ville det nok være mere logisk at finde ID på en liste som programmet løb igennem når en ny klient forsøgte at oprette forbindelse.

4. Så snart DFU_ID er ok kan dataoverførslen starte. En pakke læses ind fra sokkelen, hvilket igen vil sige at der læses en header på 5 byte og derefter ventes på at der er modtaget datalængden data.
5. Data håndteres efter type. Kan typen ikke genkendes springer man pakken over og læser næste pakke.
6. DFU_SYNC. Når der modtages en DFU_SYNC kan outputfilen være i tre tilstande. Åben, lukket eller endnu ikke skabt. Er den Åben så skal data bare skrives til den, men er den åben eller slet ikke lavet så skal den åbnes.
7. DFU_DATA: Igen kan output-filen være i en af tre tilstande. Er den allerede åben skal der igen blot skrives til den,
8. og er den lukket, så skal den åbnes før der skrives. Men filen ikke er lavet endnu, så skal den modtagne data ignoreres, idet en ny fil skal starte med en frame sync.
9. DFU_END: Behandlingen afhænger som altid af outputfilens tilstand.
10. Hvis den ikke er lavet endnu, så er der sket en fejl og forbindelsen lukkes. Er den lukket åbnes den før afslutning af filen påbegyndes.
11. Åben/create outputfile
12. Skriv Data til Outputfile
13. Skriv data til Outputfile and rename OutputFile til det korrekte navn.
14. Nu er dataoverførslen overstået, så kan databehandlingen gå i gang. Første skridt er at starte Separator-funktionen som skriver kanalfilerne og info-filen
15. Derefter startes Map som laver et quickview-billede
16. Til slut lukkes forbindelsen til klienten.

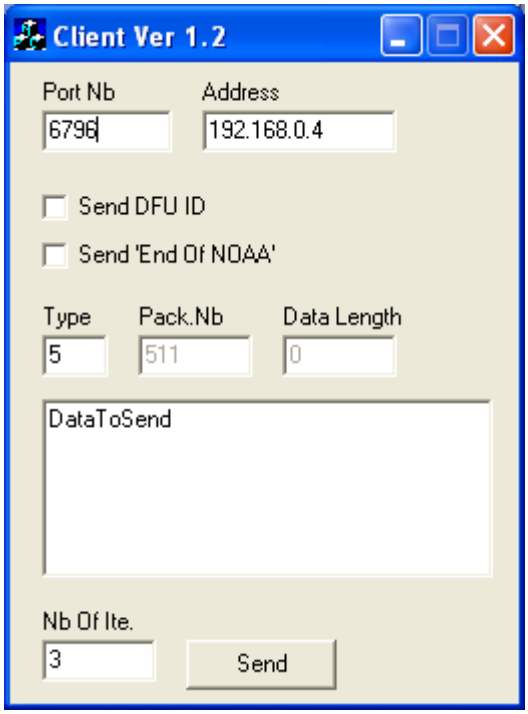
10.7.4 Test af Modtager

For at kunne teste at Modtagerprogrammet fungerede som det skulle blev der skrevet et program i Delphi som benytter IP, TCP og sockets og implementerer en del af DFU/DPU-protokollen (se appendix).

Programmet kan indstilles til at sende til en valgt ip-adresse og port, og kan sende et antal DFU/DPU-protokol datapakker med den type og det indhold man indtaster. Datalængden passer altid med indtastet data.

Afkrydsning af checkboxen Send DFU ID bevirker at klienten starter med at sende Type = DFU_ID i første pakke efter oprettet forbindelse, og i datafeltet som for denne pakke er 12 byte lang skrives: "0030556802b7" (her burde 'b' have været B for at svare til det ID DFU sender men under testen ændres denne karakter blot i ini-filen.

Afkrydsning af anden checkbox bevirker at data sluttes af med en TYPE = DFU_DATAEND



Datamodtageren blev testet ved at bruge klientprogrammet til at test at der var oprettet en velkomst sokkel, og at en forbindelsessokkel oprettedes så snart klienten fik forbindelse (portnummer blev skrevet til skærmen)

- Det blev testet at hvis første pakke ikke være en DFU_ID eller hvis ID'et var forkert, så blev forbindelsen blev nedlagt og velkomst sokkelen var straks klar til modtagelse igen.

- Test af at i den situation hvor der endnu ikke var lavet en output fil havde modtagele af hver datatype den forventede effekt:
    - DFU_DATA: data ignoreres
    - DFU_END: forbindelse lukket
    - DFU_SYNC: fil laves og data skrives til fil.
    - Ukendt type data ignoreres
- Tilsvarende når Outputfil er åben
    - DFU_DATA data skrives til fil
    - DFU_SYNC data skrives til fil
    - DFU_END data skrives til fil, og fil renamed (lukkes)
    - ukendt type data ignoreres
- Tilsvarende når outputfile er lukket
    - DFU_DATA: fil åbnes og data appendes til fil
    - DFU_SYNC: fil åbnes og data appendes
    - DFU_END: fil åbnes, data appendes, fil renamed
    - Ukendt type ignoreres
- Når forbindelse afbrydes fra klientens sideskal serveren straks være klar til at acceptere en ny.

Mangler: det viste sig at være et problem at det ikke i første omgang blev testet at modtageren kunne klare at der blev sendt DFU_DATA umiddelbart efter DFU_SYNC – som jo burde være det almindelige. Man burde have lavet en tilstandsmaskine hvor tilstanden kunne testes.

10.7.5 Kanalseparator

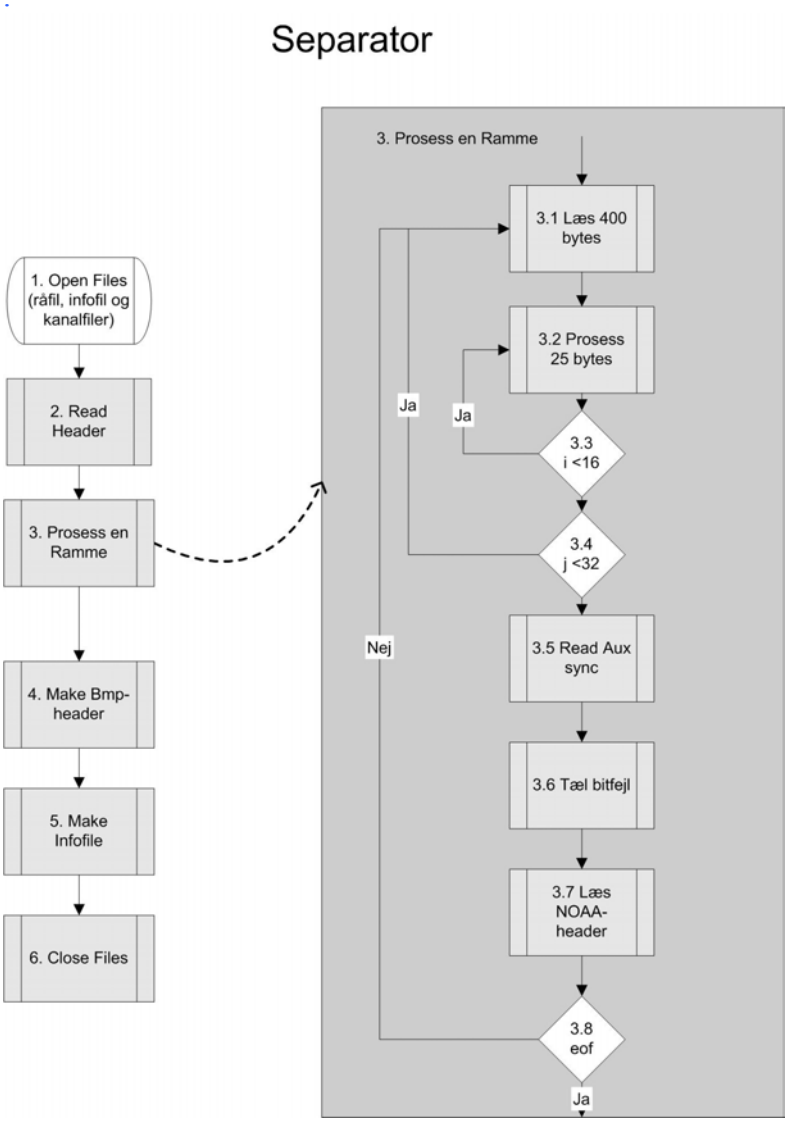
Kanalseparator-funktionens opgave er at læse NOAA-10bit-råfilen som DFU og Datamodtageren har samlet fra satellitterne, og skille data ud i de fem frekvenskanaler rådata indeholder. Kanalseparatorens output er fem 8 bit kanalfiler som hver indeholder billeddata fra kun en frekvens kanal.

Desuden skriver Separator en kort infofil med et par oplysninger om satellitpassagen – blandt andet datakvaliten af den modtagne data. Separator bruger 125 byte kendt data i NOAA-headeren til at beregne datakvaliteten (bitfejl / antal checkede bit)
Som input bruger separator altså den modtagne råfil, og som parametre råfilens navn/direktorie sti samt en satellittens baneelementer (de skal kun bruges for at kunne skrive infofilen)

De rå data fra NOAA-satellitter er inddelt i 10 bit "ord" i stedet for 8 bit bytes.
En hel NOAA ramme er 11090 ord lang og indeholder altså 13862.5 byte
For at kunne transmittere et helt antal byte over lægger DFUen 4 bit for enden af hver ramme som altså bliver 13864 byte lang.
På denne måde kommer hver NOAA-ramme starte i begyndelsen af en byte og rammerne kan derfor behandles uafhængigt af hinanden.

Udfordringen ved at separere kanalerne i NOAArammen består i at hver pixel i hver kanal er et 10 bit ord – sådan at hver femte ord i billeddatafeltet skal fortolkes som et pixel i samme kanal – men ikke fordelt på samme måde ud over de underliggende byte.
Når data skal præsenteres i adskilte kanaler ønskes hver pixel beskåret til 8 bit data ved at de to mindst betydende cifre smides væk.
Dette gøres med and, or og shift operatorer

Hver NoAA billeddatafelt består af 10240 ord (2048 per kanal) som altså alle sammen skal lægges ind som bytes i deres respektive kanalfiler. For at alle operationer skal passe i hvert omløb af den løkke som behandler data skal løkken både behandle et helt antal bytes og et helt antal pixel.
Der er 8 bit i en byte og (5*10)bit = 50bit i et pixel. Dvs. hver omløb i løkken skal behandle mindst 200 bit (mindste tal som både 8 og 50 går op i). Altså 25 byte.



10.7.6 gennemgang af Separator's flowdiagram

1. Open Files

Råfilen åbnes til læsning og de fem kanalfiler samt infofilen åbnes til skrivning. Der åbnes også en fil til at skrive et bmp-billede. Dette billede er strengt taget ikke noget DMI har brug for – det er her kun for hurtigt at kunne se om kanalfilerne er ok.

2. Read Header

Den Første header i råfilen læses og de nødvendige data gemmes.

3. Behandling af én Ramme

Tredje skridt er en løkke som prosesserer en NOAA-ramme per omgang. De bytes som skal skrives som billedpixel i kanalfilerne hentes direkte fra satellitdata's 10bit ord. Det giver en del beregning med and og or operatorer. Et alternativ til denne metode ville være først at lægge 10 bit data ind i 16bit ord. Det ville gøre at alle ord startede i begyndelsen af en byte. Transformationen fra ord til byte kunne så foregå ens for alle ord – men da der kun er et sted i Separator() man har brug for at transformere ord til byte ville det ikke give færre beregninger – man ville blot flytte problemet til den funktion som lagde ord i 16 bit arrays.

a. Behandlingen startes ved at programmet læser 400 byte ind i et array. Denne størrelse er valgt for både at begrænse læsetiden og brug af hukommelse. Data skal behandles i klumper af 25 byte (fordi det er mindste antal byte som indeholder et helt antal pixel (50 bit)). 25 går op i 400, og 400 går op i 12800 som er den mængde date der er i en hel NOAA-ramme.
b. Prosess 25 byte: Billeddata er som tidligere forklaret inddelt i 10 bits ord som hver repræsenterer en pixel i en kanal. I kanalfilerne skal hver pixel skrives som en 8bit byte. Dette betyder at hver ord skal hentes ud af to byte ved hjælp af "AND"-operationer og shift-operationer, og samles i en byte med en "OR"-operation. Ordene forskydes to bit i forhold til bytene for hver ord. Således at hvis første ord starter fire bit inde i en byte, så starter andet ord seks bit inde i en byte. Første ord starter samme sted i en byte som 16. ord, og samtidig tilhører både 1. og 16. ord første kanal, så klumper af 15 ord kan behandles på nøjagtig samme måde.
c. Read Aux sync: Efter datafeltet ligger der 100 kendte ord. Disse læses ind i et array, så eventuelle bitfejl kan tælles.
d. Det læste Auxiliary sync ord sammenlignes med de bit som skabes med polynomiet x^10 + x^5 + x^2 + x + 1, hvor generatorbittene alle er 1 lige før første auxiliary bit. De første fire og sidste fire bit i auxiliary sync benyttes ikke i udregningen fordi auxiliary sync starter og slutter midt i en byte. Det første 4 bit af auxiliary sync læses ind sammen med billeddata, og de sidste fire ligger i en byte sammen med 4 bit padding-bit som DFU'en sætter ind for at få en hel NOAA-ramme til at fylde et helt antal byte.
e. Læs NOAA-header. Kun den NOAA-header som ligger først i filen bliver brugt til noget. Her læses headeren kun for at komme frem til næste datafelt.
f. eof: Prosessen fortsættes ind til man filen er slut.

4. Make Bmp-header

For at kunne se kanalfilerne som et billede skrives der her en bmp-header til den sidste fil.
5. Make Infofile.
Der skrives en Infofil (samme navn som råfilen, men med endelsen .INFO) I den skrives Satellittens ID, tidstempel, hvor mange rammer der blev talt i råfilen og hvor mange bitfejl der var på hvor mange auxiliary sync bit.
6. Close Files

10.7.7 Test af Separator

Seperator funktionen blev testet på råfiler som DMI har modtaget på deres eksisterende udstyr. Det gav en del problemer at det viste sig at DMI's råfiler er tre byte kortere end dem man ville forvente at modtage fra DFU – fordi DMI af historiske årsager skærer auxiliary sync bort og erstatter den med 0'er – dog kun 122 byte i stedet for de 125 byte NOAA-protokollen foreskriver.

Først efter hensyntagen til de tre manglende byte lykkedes det at få lave et genkendeligt billede ud af kanalfilerne:

0602141001.NOAA17.45kanal.bmp

10.8 Map

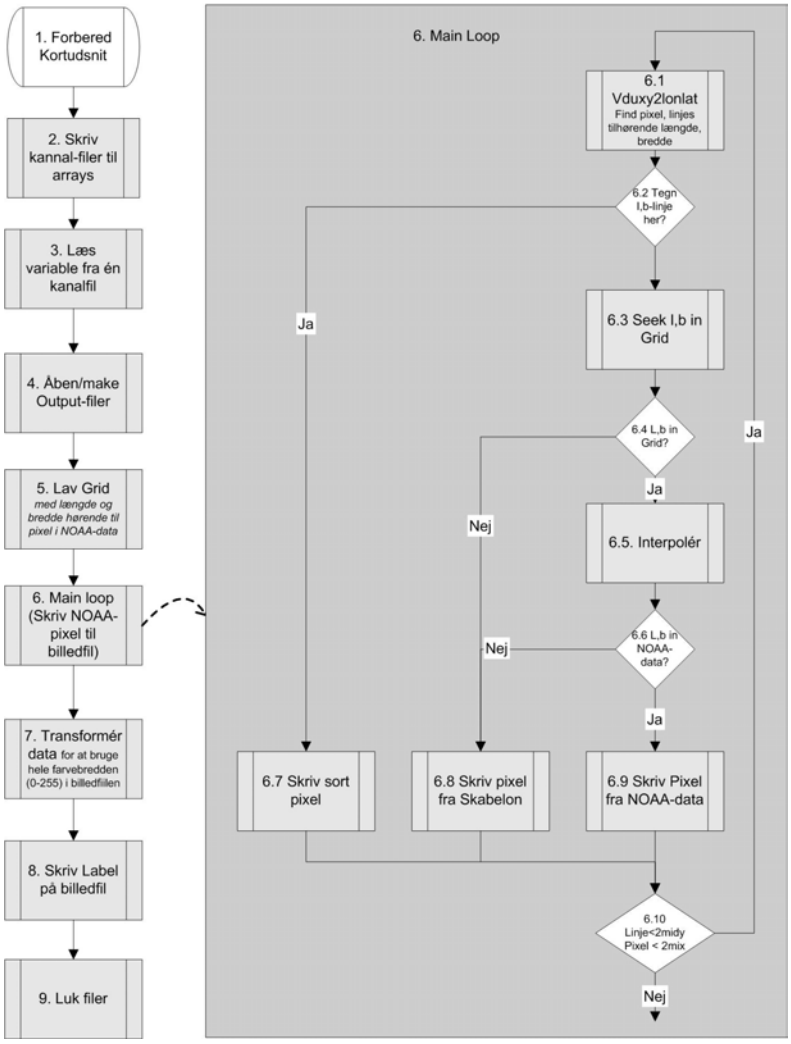
Funktionen map()'s hovedformål er at lave geometrisk opretning på NOAA-data. Satellitternes baner og øjeblikkelige position beregnes ud fra deres baneelementer og det præsise tidspunkt hver enkelt billedpixel blev optaget i satellitten. Baneberegningerne baserer sig på løsning af Kepler's ligning, men for at få tilstrækkeligt præcise resultater benyttet NASA's formler og metode og ikke vores egen. NASA's beregninger finder man i underfunktionen sgpsat() som også beregner hvilken retning satellitten ser i på netop det tidpunkt ved optagelse af et pixel.
Det nøjagtige tidspunkt findes ved at tælle frem pixel fra første pixel i første frame i passagen hvorfra man har det nøjagtigte tidstempel læst i framens header.

Der skrives 7 filer på grundlag af den geometriske opretning. Fem konturfiler og 2 bitmap billeder. Alle fem filer afbilleder samme udsnit af jordkloden, og har samme opløsning (800 pixel X 600 pixel).

Hver pixel repræsenterer en position på jordkloden, ligesom hver byte i kanalfilerne repræsenterer en position på jordkloden. De pixel i 800x600 billedet som repræsenterer en position der også kan findes i kanalfilerne bliver der skrevet satellitdata i, mens de andre bliver fyldt med data fra en skabelon. De fem konturfiler indeholder kun 1 byte per pixel og har ingen header – de er præsentationer af hver deres kanal
bitmapbillederne indeholer 24 byte per pixel – 8 byte fra tre kanaler (1,2 og 4) og har fået sat en bmp-header på så de umiddelbart kan vises som et billede. I den ene af bitmap billederne er der blevet manipuleret med farverne sådan at databredden på 0-255 er bedre udnyttet for hver kanal – i det andet er bytes fra kanalfilerne sat direkte ind i billedet.

For at kunne foretage den geometriske opretning skal map() bruge satellittens baneelementer som blev hentet i "modtager" foruden selvfølgelig de 5 kanalfiler.

Map



1. Forbered Kortudsnit

Der foretages omregning fra grader til radianer, og centrale parametre for projektionen beregnes. Zoomnivo er bestemt af afstanden mellem *bmid* og *btop*, og dette sammen med billeopløsningen (antal pixel per linje) bruges til at beregne jordradius, *r*, i pixel.

Herefter kan y koordinatens centrum (i pixel fra nordpolen), *ycntr*, beregnes (*ycntr = 2r*tan(pi/4 – bmid/2*)

2. Skriv kanal-filer til arrays

De fem kanalfiler åbnes og læses ind i hver deres byte-array til senere brug

3. Læs variable fra headeren af kanalfil

Så læses satellitadresse og tidstempel fra en af kanalfilernes header – de skal kun bruges til at skrive som oplysninger i kanalfilernes første linje – Nogle af DMI's programmer bruger disse oplysninger til finde ud af hvilken satellitpassage der er tale om – og til at skrive den label der sættes på quickview billedet.

4.Outputfilerne åbnes.
Der åbnes konturfiler til geometrisk oprettet version af hver kanal, og to billedfiler til oprettede quickviewbilleder.
To quickview billeder fordi i et af dem bliver farverne gjort klarere ved at sprede data ud så hele bitbredden udnyttes – der er jo 8 bit for hver pixel, kanal så værdierne skulle helst ligge godt fordelt mellem 0 og 255.

5. GRID.
De beregninger som skal til for at løse Keplers ligning og for at beregne et NOAA-pixels position på jordkloden er ret beregningstunge, så det kan betale sig at begrænse hvor mange gange de skal udføres.
I en gennemsnitlig satellitpassage er der ca. 5000 frames, og i en frame er der 2048 byte. Til sammenligning er der kun 800X600 pixel i et quick-view billede at placere date i. Man må altså gå ud fra at hver pixel i quickview billedet gennemsnitligt dækker over mindst 2.5*8 pixel i kanalfilerne. Derfor mistes der ingen præcision ved kun at foretage de præcise beregninger for et gitterværk af pixel i satellitdata og så interpolere når der er brug for positioner imellem de beregnede.

Man beregner længde og bredde for hver 128. pixel i hver 128.frame (linje i kanalfilbilledet).
Det præcise antal frames i en passage er ikke kendt på forhånd, men programmet beregner positioner i 18X46 punkter, så der dækkes et område på (18-1)*128pixel X (46-1)*128frames = 2176pixel X 5760 frames.

Gitterværkets første punkt ligger 128 frames før første frame i data, og 64 pixel før første pixel i denne fiktive frame. Tilsvarende er sidste punkt i hver linje 64 pixel efter sidste datapixel.
Grunden til man beregner punkter som ligger uden for satellitdata, og ikke blot nøjes med at gå til kanten af data er for at være sikker på, at der for ethvert punkt i satellitdata kan findes fire beregnede punkter som omgiver datapunktet. Dvs. de definerer hjørnerne i en firkant der indeholder datapunktet. Det ville man ikke kunne sikre hvis de beregnede punkter kun gik til kanten af data.

6. Main loop.
Når data skal lægges ind på et kort kan man vælge en af to veje. Første mulighed er at tage udgangspunkt i satellitdata, og byte for byte placere data på kortet.
Anden mulighed er at tage udgangspunkt i quickviewbilledet og for hver pixel i billedet finde den byte i satellitdata som bedst repræsenterer den position..
I en enkelt satellitpassage er der typisk ca. 10 millioner pixel, mens der kun er 480.000 pixel i et quickview billede, så at tage udgangspunkt i quickviewbilledet giver langt færre beregninger.

Main loop er en dobbelt løkke som løber gennem alle pixel i en linje og over alle linjer. For hver pixel beregnes den tilsvarende længde og bredde, og hvis positionen ligger inden for det område der er dækket af data hentes data fra kanalfilerne – hvis positionern ligger uden for dataområdet hentes data fra en skabelon.

6.1 vduxy2lonlat

Længde og bredde af den pågældende pixel beregnes

6.2 Tegn l,b-linje

For at give et overskueligt billede lægges der linjer ind, som markerer hvor breddegrader og længdegrader har hele multiplum af 10. (altså 0.,-10.,10.,-20.,20., ... bredde og længdegrad)

6.3 Seek

Det undersøges om den fundne (længde, bredde) ligger inden for gitteret af beregnede positioner. Dvs. om der kan findes 4 punkter som definerer hjørnerne i en firkant som indeholder (længde, bredde). Hvis sådanne 4 punkter findes, returneres det første af disse punkter. Det skal bruges som udgangspunkt for interpolertion.

6.4 l,b in grid?

Hvis (længde, bredde) ikke lå inden for griddet er det ingen grund til at fortsætte beregningerne, så skal data hentes fra skabelonen.

6.5 Interpoler

(længde,bredde) ligger inden for grid og der er fundet fire punkter i griddet som "indeholder" den ønskede position. Interpoler itererer sig frem til at finde (længde,bredde)'s relative position i forhold til de fire punkter og finder så den rette pixel ved at antage at de 127 pixel der er i hver retning mellem de beregnede pixel fordeler sig lineært.

6.6 l,b in NOAA-data?

Selv om det ønskede pixel er fundet i grid, er det ikke sikkert at det ligger inden for data. er pixelnummer under nul eller over 2048 så ligger det uden for og tilsvarende er linjen ikke mellem nul og antal modtagne frames så er det heller ikke i data, og i så fald skal data hentes fra skabelonen. Ligger pixlet inden for data hentes data i de arrays der er lavet af kanalfilerne.

6.7 Skriv sort pixel

Hvis det i 6.2 blev besluttet at denne en pixel var en del af en længde/breddegradslinje så bliver data her overskrevet med et sort pixel

6.8 Skriv pixel fra skabelon

6.9 Skriv pixel fra NOAA-data

6.10 En tæller tæller pixelnummer op fra 0 til 800 og linjer fra 1 til 600.

7. Transformér:
Hvis bytes fra kanalfilerne skrives helt ubehandlet til et quickview-billede vil billedet ofte blive meget blegt og svært at tolke. For at gøre farverne klarere forsøger programmet at udnytte databredden på 8 bit i hver kanal bedre.
Dette gøres ved hjælp af variablerne chamax og chamin. Hver gang der under den geometriske opretning skrives et pixel som har en værdi der er mindre end chamin eller større end flyttes chamin eller chamax en tiendedel af afstanden i retning af pixlens værdi. Således er chamin og chamax ved endt opretning udtryk for hvilken del at databredden der er udnyttet, og antager man at alle pixelværdier findes mellem disse to yderpunkter kan databredden udnyttes bedre ved at løbe igennem alle byte i billedet (skatemp[][]) og sætte:
skatemp[i][j]= 1 + (skatemp[i][j]-cha1min)*254/(cha1max-cha1min);
på denne måde kommer værdier der er fordelt mellem chamin og chamax til at ligge fordelt mellem 1 og 254 i stedet.
De værdier som ligger uden for chamin til chamax, bliver undtaget af behandlingen, og kommer til at se forkerte ud, men det vil dreje sig om et fåtal.

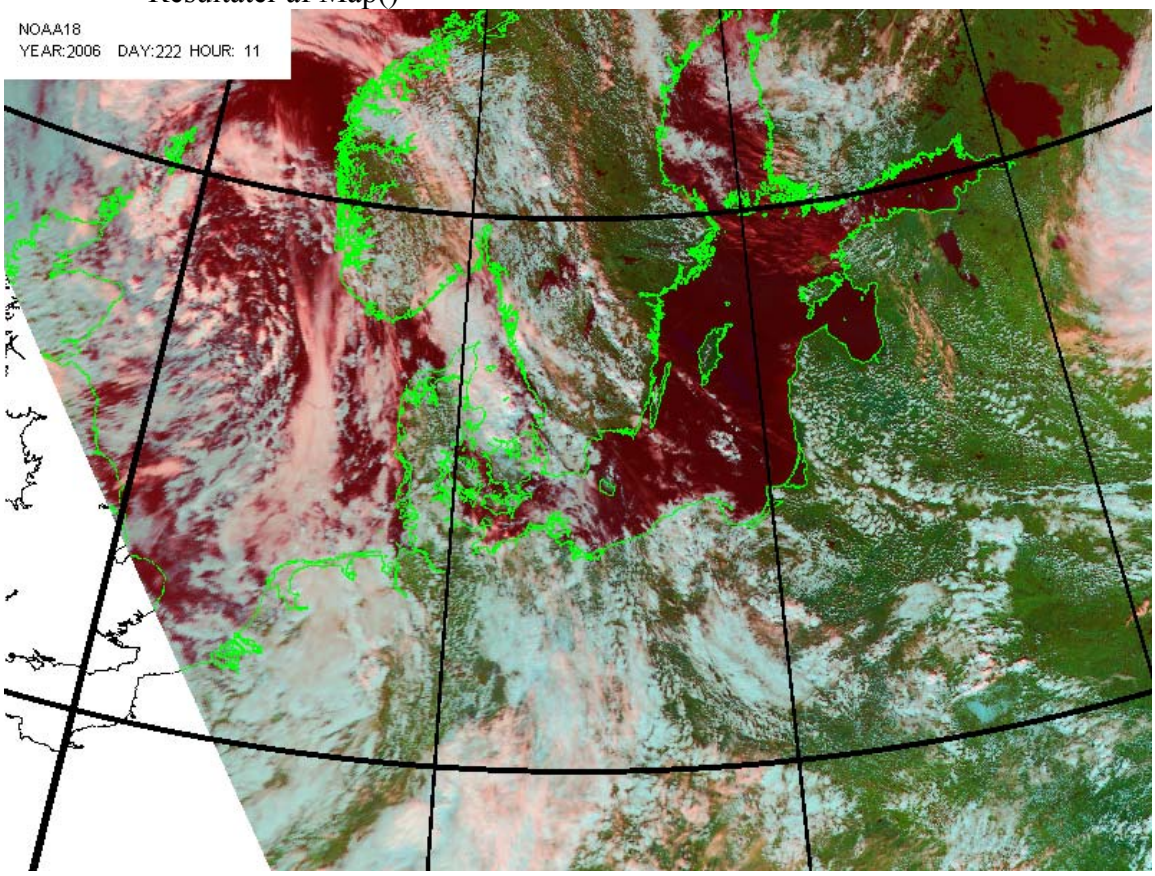
Denne transformation er grunden til at der bliver lavet to quickviewbilleder – et med de blege farver, og et hvor nogle bit falder ved siden af.
Det blev også forsøgt beregne chamin og chamax for hver linje i stedet for, for hele billedet, men selvom det giver et skarpere billede giver det også nogle generende linjer i billedet fordi hver linje fortolkes forskelligt.

Det skal nævnes at kun pixel med satellitdata medtages i denne behandling. De pixel som henter deres data fra skabelonen bliver ikke påvirket, og tæller heller ikke med i beregningen af chamin og chamax.

8. Skriv Label på billedfil
Quickview skal have en label i øverste venstre hjørne hvor der står hvilken satellit der har lavet billeddata og hvilket årstal, dag og time billedet er lavet.
Dette gøres ved først at skrive oplysningerne i en streng. F.eks.: "NOAA17 \n YEAR:2006 DAY:222 HOUR:11"
Så kaldes make_label() der konverterer strengen til pixel i et array af byte (50X600 byte) som så kan skrives til billedfilerne. Arrayet skrives lige efter bmp-headeren 54 byte inde i filen
make_label konverterer charakterer i strengen til pixels i bytearrayet ved hjælp af en skabelonfil, "fonte" som ligger i hjælpedirektoriet (defineret i inifilen). "fonte"-filen indeholder skabeloner til cifrene 0-9, de store bogstaver og de mest almindelige tegn. (; : , - _).
'\n' fortolkes som et linjeskift – men der er i virkeligheden kun plads til 2 linjer på de 50X600 pixel bytearrayet indeholder.
Der er ikke plads til et fast antal tegn i hver linje da tegnene ikke er lige brede, men de 600 byte repræsenterer 200 pixel idet der bruges 24 bit farver i quickviewbillederne.

9. Luk filer.
De fem kanalfiler og 2 billedfiler lukkes inden map() afsluttes.

Resultater af Map()



11 Afsluttende Test

Ind til nu har systemet (DFU/DPU) ikke være testet med live data. En live test kunne kun foretages på DMI's ubemandede station på Smidsbjerg. Testen skulle laves på lokalnettet på stationen idet stationens firewall ikke umidelbart kunne omkonfigureres.
Både hardware og en server med DPU installeret blev bragt til Smidsbjerg og forbundet gennem stationens firewall.

11.1.1 Testplan:

1. Første test skal blot teste dataoverførsel på stationens lokal net. Testgeneratoren genererer nogle Mb data som modtages man checker at den modtagne fil har framesync s med 13863 byte mellemrum.
2. I anden test skal det testes at live data kan akkumuleres i ram før det bliver sent. Her skal hastighed og datakvalitet testes. Hastigheden skulle gerne være mindst den dobbelte NOAA-rate (1330,8 kbit per sekund eller ca. seks sekunder per Mbytes). Datakvaliteten kan verificeres dels ved at se på billederne (kanalfil-billedet og quickview) og dels i info-filen hvor bitfejl angives.
3. Tredje test er den fulde test, hvor live data lægges i RAM, samtidig med at den videre sendes. Her er det først og fremmest datakvaliteten der skal checkes. Transmissionshastigheden skal være det samme som NOAA-raten.

11.1.2 Test forløb – Problemer og Løsninger

1. Test: Første test viste at der kunne oprettes forbindelse mellem DFU'en og DPU'en på lokalnettet, og overføres data. I DFU'en blev der detekteret framesync fra testgeneratoren, med den rette frekvens. Der blev akkumuleret ca. fem Mbytes, hvorefter testgeneratoren blev frakoblet. Efterfølgende blev klientprogrammet i DFU'en startet, og det opsamlede data blev transmitteret til DPU'en.
DPU'en opsamlede filen på ca. fem Mbytes, og fremesync var placeret for hver 13684. byte, hvilket er én byte for meget.
Denne fejl blev ikke rettet inden første sattelitpassage var tilgængelig, og vi vurderede at live data ville kunne bruges til testformål på trods af den forkerte framelængde.

2. Test: I anden test modtog DFU'en live data fra en satellit (ifølge tidsplanen skulle det være NOAA16) og der blev igen detekteret framesync i DFU med korrekt frekvens. Der blev opsamlet ca. 50 Mbyte data i ram, men det viste sig at kun de sidste 15 Mbyte var brugbar data – resten blev opsamlet som følge af at DMI's modtageudstyr konstant sender clock-signal selv når det ikke er data fra en satellit. DFU'en var konfigureret i forventning af at

clock-signalet kun ville være aktivt når der var satellitdata. Det betød at der blev fyldt data i ram så snart DFU'en blev tilsluttet modtageudstyret. Transmissionen af de 50 Mbytes tog 290 sekunder, hvilket svarer til ca. 170 kBytes per sekund.

De 15 Mbyte blev opsamlet af DPU i en fil hvor det med det samme viste sig at satellit ID og tidstempel blev læst forkert. Ved undersøgelse af filen kunne man se at alt data mellem framesync's var uigenkendeligt. Hver framesync var som forventet stadig en byte for lang.

Framelængdefejlen viste sig at skyldes mekanismen som detekterer RAM-syncronisering i DFU'ens klientprogram. Der blev konsekvent indskudt en ekstra byte før hver framesync.
Fejlen blev rettet og endnu en satellitpassage blev forsøgt optaget.
Denne gang var afstanden mellem framesync korrekt, men den mellemliggende data var stadig uforståelig.

Fejlen med den ødelagte data blev igen fundet i DFU'ens klientprogram og kunne rettes ved at ombytte læserækkefølgen fra RAM-Controleren.

3. Test: Fejlene havde desværre på dette tidspunkt taget næsten hele den testtid der var til rådighed, og derfor blev det besluttet at gennemføre 3. test, på den sidste satellitpassage der var til rådighed. Derfor blev den sidste passage opsamlet af DFU'en, og transmitteret samtidig. Kort tid efter start af klientprogrammet i DFU'en, opstod der en exception i mikrokontrolleren, som medførte at transmissionen blev afbrudt. DFU'en blev genstartet, og opsamlingen påbegyndt på ny. Passagen blev opsamlet indtil satellitten forsvandt under horisonten.
og der var blev derfor kun testet på slutningen af en satellitpassage.
Nu kunne satellitens identitet og tidstemplet læses korrekt, og et quickview-billede fremstilles.

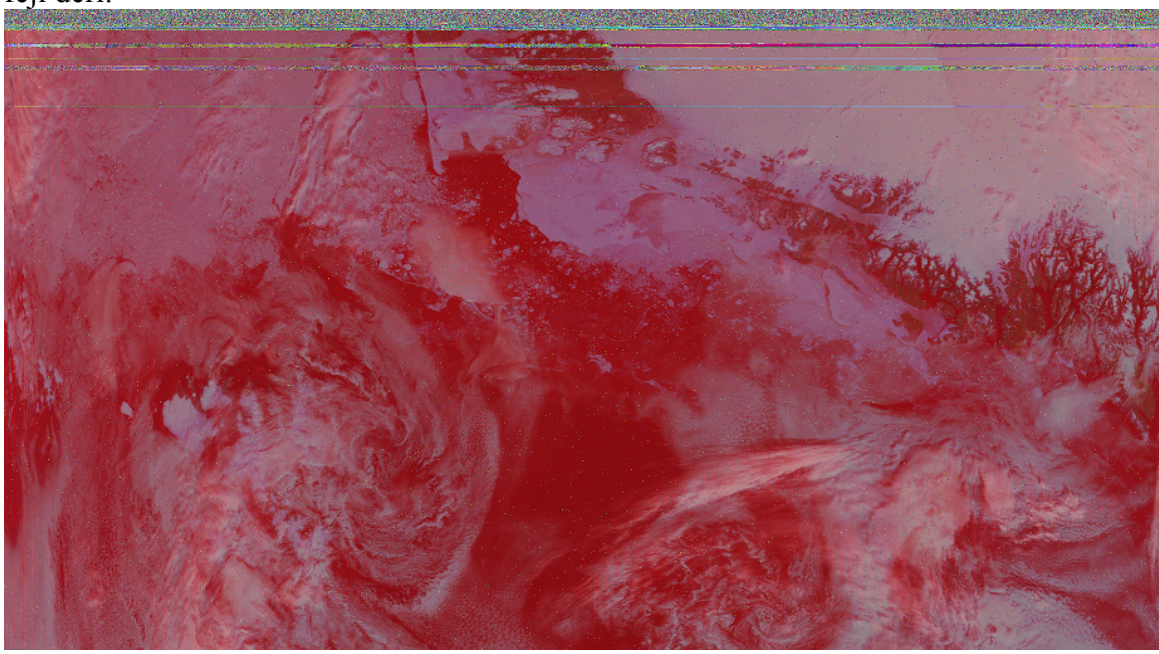
11.1.3 Konklusioner

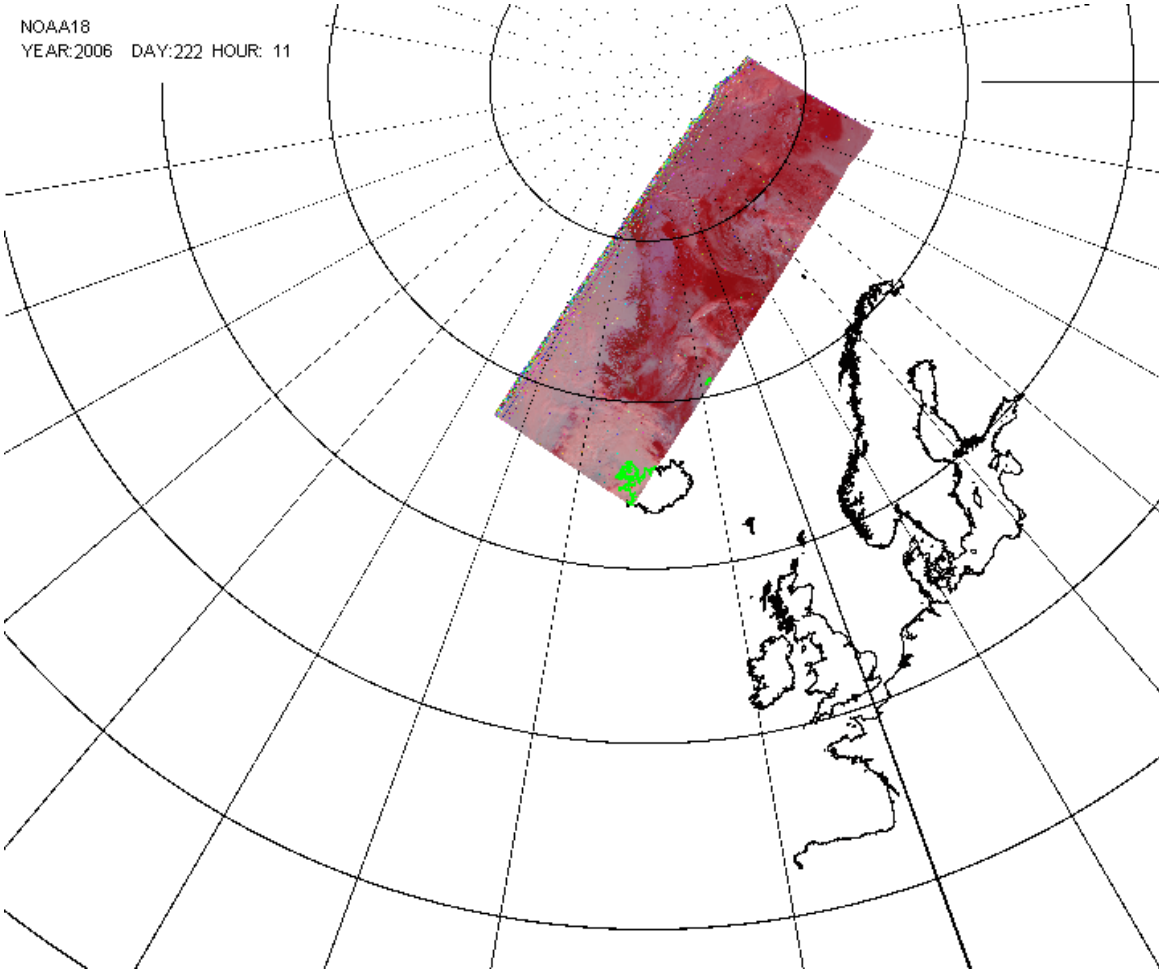
DFU'en modtager clock- og datasignalerne korrekt, og framesync detekteres med rette den frekvens.
Samtidig er systemet er i stand til at overføre oplagret data med op til den dobbelte NOAA-datarate, og også sende data løbende, samtidig med der modtages ny data fra satellitten.

Selvom testen kun nåede at blive gennemført på den sidste del af en passage, blev der modtaget en udmærket råfil på 13.5 Mb.
Navnet som DPU har givet den rå fil (0608101117.NOAA18.222) indeholder korrekt tidsstempel og satellit ID - dvs. programmet har læst første modtagne header. Både tidspunkt (8. måned, 10.dag (222. dag i året) kl. 11:17) og satellit ID passer med fakta, så i hvert fald begyndelsen af filen er korrekt.

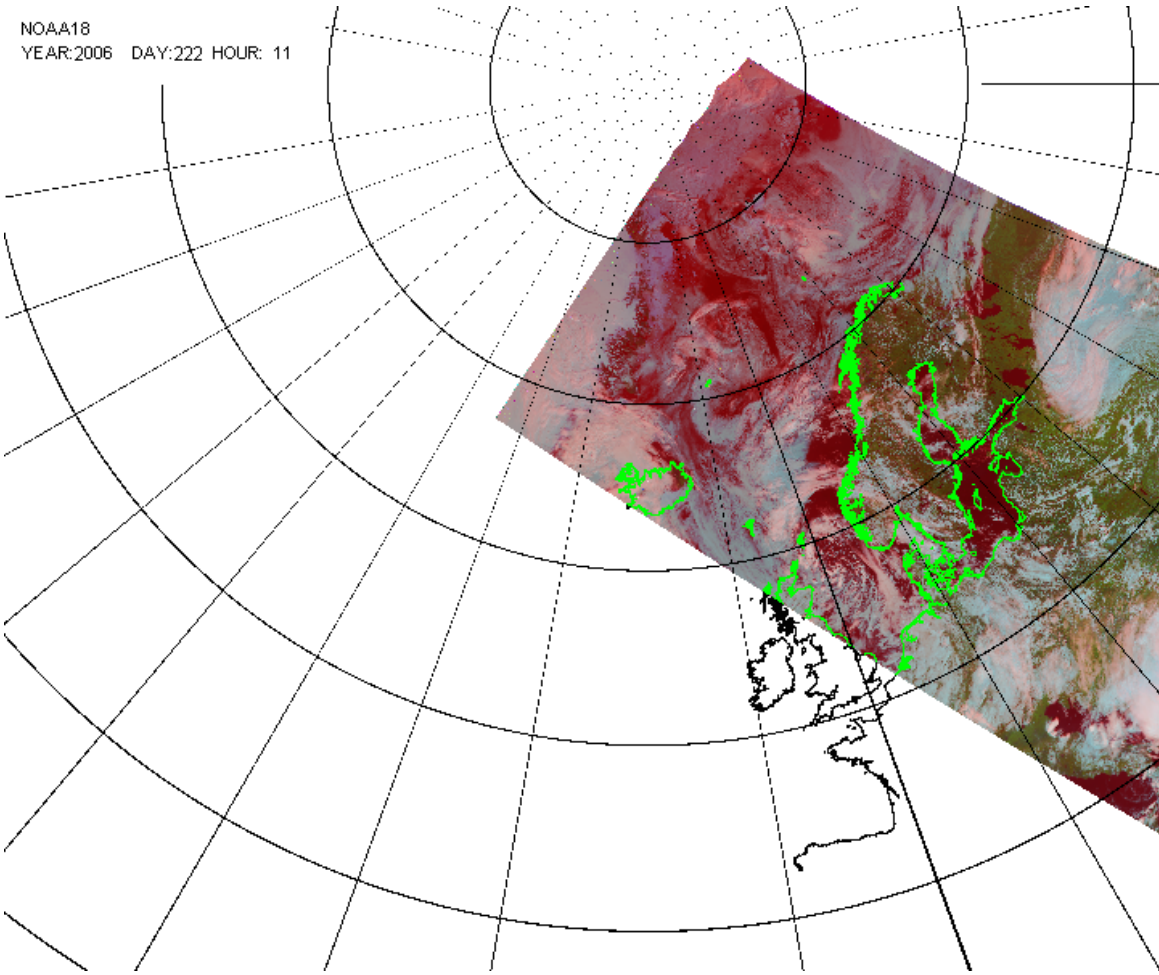
Man kan desuden se at framesync og aux sync, ligger med de korrekte mellemrum.

Info filen siger at der er 0.025707 bit fejl per bit – altså to fejl ud af hundrede. De fleste af disse kommer sandsynligvis fra slutningen af passagen, hvor en hel del af rammerne er forskubbet og helt ulæselige. Disse rammer vil jo gennemsnitligt have 50% bitfejl.
Bitfejlene kan også ses på både kanalfils billedet og på quickview billedet. Det vides ikke med sikkerhed hvordan bitfejlene opstår, men mellemlagringsdelen i DFU'en er under mistanke, da tidligere tests har vist fejl deri.





Den passage som det lykkedes at hente en del af med DFU'en, blev også optaget af DMI's eksisterende udstyr. DMI's råfil blev kørt igennem DPU programmet for at genererer sammenlignelig data – bl.a. et quickview billede. I det område som begge billeder dækker kan man se Grønlands østkyst og sky- og is–formationer, og i det område er billederne identiske. På billedet fra DMI's råfil kan man se at data passer med Europas landekonturer. Man kan altså med ret god sikkerhed konkludere at en komplet passage ville have givet et tilfredsstillende resultat.


NOAA18
YEAR:2006 DAY:222 HOUR: 11

12 Konklusion

Hovedformålet med projektet er at designe og konstruere en prototype på et system, som kan hjælpe DMI til at spare tid i modtagelsen af vejrdata fra NOAA-satellitter, således at meteorologerne får friskere data at arbejde med. Ideen er at spare den tid det tager at overføre data mellem DMI's ubemandede satellitmodtagestation og deres hovedsæde. Dette skal ske ved at transmittere data løbende mens man modtager den på satellitmodtagestationen, i stedet for som det sker i dag, at vente med at videresende til man har modtaget en komplet passage.
Løsningen skal desuden leve op til de rammer DMI stiller til rådighed. Bl.a er internetforbindelsen mellem station og hovedsæde en 2 Mbit MPLS-forbindelse og DMI's servere kører linux som operativsystem.

Vi har vurderet at den bedste løsning på opgaven er at udvikle hardware til placering på modtagestationen, og software til installation på en af DMI's servere i hovedsædet.
Hardwaredelen skal modtage data fra NOAA-satellitterne, og videresende den løbende over MPLS-forbindelsen.
Softwaredelen modtager og behandler data på hovedsædet.

På baggrund af disse valg det rimeligt at opstille yderligere krav, som skal opfyldes, før løsningen vil være tilfredsstillende.

- Hardwaredelen skal være i stand til at mellemlagre data i tilfælde af internetafbrydelse. Der skal være plads til mindst en hel satellitpassage
- Dataoverførslen over MPLSforbindelsen skal mindst svare til den ca. dobbelte NOAA-datarate således at akkumuleret data som følge af internetafbrydelse hurtigt kan indhentes.
- Ideen med at sende data løbende sparer ca. 5 minutter for DMI hvis man regner med en satellitpassage på 70 Mbyte og at den fulde båndbredde på 2Mbit er til rådighed. Derfor vurderes det ikke må tage mere end 1 minut at behandle data.

Det skal nu vurderes, om den designede løsning lever op til de opstillede krav, i tilstrækkelig grad til at løse opgaven tilfredsstillende:

Den designede stand alone enhed (DFUen) kan modtage og videresende NOAA-satellitdata, og det er testet at den kan fungere under de forhold DMI stiller til rådighed i deres station på Smidsbjerg.
Dvs. der kan modtages NOAA-data i form af clock- og data-signal med en frekvens på 665,4 kHz. Enheden er udformet så den kan monteres i et 19" rack chassis med en højde på en U, og den tilbyder visuel indikation af framesync i form en LED, samt indikation af datatab hvis mellemlagringskapaciteten overskrides – også en LED.

Tilsvarende kan den designede softwareenhed (DPUen) modtage og behandle data fra DFUen under de forhold DMI stiller til rådighed på deres hovedsæde på Lyngbyvej. Dvs. den er installeret på en Linuxserver som de har stillet til

rådighed, og leverer NOAA-råfiler, kanalfiler, samt quickview billeder og information om satellitpassagen inden for et acceptabelt tidsrum (under et minut).

Overførsel af data mellem DFU og DPU er testet på et IP-netværk og hastigheden er i test målt til ca. 170 kbyte/s hvilket er over det stillede krav på 2 gange NOAA-raten.

Alt i alt er der designet et system som kan modtage data under de forhold DMI har til rådighed både i deres station på Smidsbjerg og på Lyngbyej. Systemet kan transmittere data mellem stationen og DMI's hovedsæde, mens data stadig modtages fra satellitten.
Kravet om at spare tid i overførslen er helt klart opfyldt ved at transmittere løbende, og både transmissionshastighed og databehandling er hurtig nok til at den vundne tid kan udnyttes.

Til gengæld er systemet ikke drift-stabilt. Det er under den afsluttende test konstateret at DFU'en risikerer at lave exceptions og stoppe drift, og på softwaresiden på Lyngbyvej er der ikke lavet fejlhåndtering, så programmet risikerer stoppe hvis det støder på en uventet situation.

Systemet er altså ikke drift klart på nuværende tidspunkt, men det vurderes at det forholdsvist hurtigt ville kunne blive det hvis både DFU og DPU underkastes fejlhåndtering. Derfor bedømmes det at designet er i orden og at det med finpudsning af implementeringen tilfredsstiller de stillede krav.

13 Appendiks

13.1 VHDL-kode for 1. RAM-controller

```
1  ----------------------------------------------------------------------
2  --                                                                  --
3  -- Handling Image Data from NOAA Weather Satellites                 --
4  --                                                                  --
5  -- Master Thesis DTU (IMM), 2006                                    --
6  ----------------------------------------------------------------------
7  -- Authors:                                                         --
8  --       Soeren Soelling Christiansen                               --
9  --       Eskil Binzer                                               --
10 ----------------------------------------------------------------------
11 -- Description of Code:                                            --
12 --                                                                  --
13 --    Top module holding of SDRAM Controller.                       --
14 --    Implementation model for Xilinx XC108 CPLD                    --
15 --    This model have to be Simulated with the Hynix hy57v28820     --
16 -- Version:                                                         --
17 --    1.1   (03-05-06)                                             --
18 -- Changes since last version:                                      --
19 --    None! - This is the 1. approach-model!                        --
20 --                                                                  --
21 ----------------------------------------------------------------------
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity FirstTest is Port (
28    wr : in std_logic;
29    DataOut : out std_logic_vector(7 downto 0);
30    rd : in std_logic;
31    clock : in std_logic;
32    reset : in std_logic;
33    pwrup : in std_logic;
34    ready : out std_logic;
35    CK0 : out std_logic;
36    CKE0 : out std_logic;
37    CK1 : out std_logic;
38    CKE1 : out std_logic;
39    RAS : out std_logic;
40    CAS : out std_logic;
41    WE : out std_logic;
42    BA : out std_logic_vector(1 downto 0);
43    DQMB : out std_logic;
44    S0 : out std_logic;
45    S1 : out std_logic;
46    D : inout std_logic_vector(7 downto 0);
47    A : out std_logic_vector(12 downto 0));
48 end FirstTest;
49
50 architecture Behavioral of FirstTest is
51
52 component Controller is
53 Port(
54 --Signals to the DRAM-Controller
55    extClk : in std_logic;
56    rst : in std_logic;
57    PwrUp : in std_logic;
58    WrRAM : in std_logic;
59    DataIn : in std_logic_vector(7 downto 0);
60    RdRAM : in std_logic;
61    DataOut : out std_logic_vector(7 downto 0);
62    rdy : out std_logic;
63 --SDRAM Address Signals
64    Row : in std_logic_vector(12 downto 0);
65    Col : in std_logic_vector(9 downto 0);
```

```
66            Bank : in std_logic_vector(1 downto 0);
67            ChipSel : in std_logic_vector(1 downto 0);
68          --Signals to the SDRAM
69            CKE0 : out std_logic;
70            CK0 : out std_logic;
71            RAS : out std_logic;
72            CAS : out std_logic;
73            WE : out std_logic;
74            BA : out std_logic_vector(1 downto 0);
75            DQMB : out std_logic;
76            S0 : out std_logic;
77            D : inout std_logic_vector(7 downto 0);
78            A : out std_logic_vector(12 downto 0));
79        end component;
80
81        begin
82        Ctr: Controller port map(
83            extClk => clock,
84            rst => reset,
85            PwrUp => pwrup,
86            WrRAM => wr,
87            DataIn => X"0F",
88            RdRAM => rd,
89            DataOut => DataOut,
90            rdy => ready,
91        --SDRAM Address Signals
92            Row => "0000000000000",
93            Col => "0000000000",
94            Bank => "00",
95            ChipSel => "00",
96            CKE0 => open,
97            CK0 => open,
98            RAS => RAS,
99            CAS => CAS,
100           WE => WE,
101           BA => BA,
102           DQMB => DQMB,
103           S0 => open,
104           D => D,
105           A => A );
106           CK0 <= clock;
107           CKE0 <= '1';
108           CK1 <= clock;
109           CKE1 <= '1';
110           S0 <= '0';
111           S1 <= '1'; -- disable S1
112       end Behavioral;
113
```

```
1  ----------------------------------------------------------------------
2  --                                                                  --
3  -- Handling Image Data from NOAA Weather Satellites                 --
4  --                                                                  --
5  -- Master Thesis DTU (IMM), 2006                                    --
6  ----------------------------------------------------------------------
7  -- Authors:                                                         --
8  --       Soeren Soelling Christiansen                               --
9  --       Eskil Binzer                                               --
10 ----------------------------------------------------------------------
11 -- Description of Code:                                            --
12 --                                                                  --
13 --       Beh. model of DRAM Controller.                             --
14 --    This model have to be Simulated with the Hynix hy57v28820     --
15 -- Version:                                                         --
16 --    1.1   (03-05-06)                                            --
17 -- Changes since last version:                                     --
18 --    None! - This is the 1. approach-model!                      --
19 --                                                                  --
20 ----------------------------------------------------------------------
21
22 library IEEE;
23 use IEEE.STD_LOGIC_1164.ALL;
24 use IEEE.STD_LOGIC_ARITH.ALL;
25 use IEEE.STD_LOGIC_UNSIGNED.ALL;
26
27 entity Controller is
28 Port(
29 --Signals to the SDRAM-Controller
30    extClk : in std_logic;
31    rst : in std_logic;
32    PwrUp : in std_logic;
33    WrRAM : in std_logic;
34    DataIn : in std_logic_vector(7 downto 0);
35    RdRAM : in std_logic;
36    DataOut : out std_logic_vector(7 downto 0);
37    rdy : out std_logic;
38 --SDRAM Address Signals
39    Row : in std_logic_vector(12 downto 0);
40    Col : in std_logic_vector(9 downto 0);
41    Bank : in std_logic_vector(1 downto 0);
42    ChipSel : in std_logic_vector(1 downto 0);
43  --Signals to the DRAM
44    CKE0 : out std_logic;
45    CK0 : out std_logic;
46    RAS : out std_logic;
47    CAS : out std_logic;
48    WE : out std_logic;
49    BA : out std_logic_vector(1 downto 0);
50    DQMB : out std_logic;
51    S0 : out std_logic;
52    D : inout std_logic_vector(7 downto 0);
53    A : out std_logic_vector(12 downto 0));
54 end Controller;
55
56 architecture Beh of Controller is
57 --Main States in this SDRAM Controller.
58 --Idle: Waits for User Input and performs Refresh
59 --Locked: After reset this State kept until PwrUp is activated
60 --(To Satisfy the 200us Wait after Power On)
61 --Only NOP's to SDRAM are issued in this State.
62 type INTERNAL_COMMAND_STATE is (PowerUp,SetModeReg,Read,Write,Idle,Locked);
63 --Execution States (inside the Main States). SDRAM Commands takes Max 10 Clocks
64 type INTERNAL_EXE_STATE is (Step1,Step2,Step3,Step4,Step5,Step6,Step7,Step8,Step9,Step10
65 signal COMMAND :  INTERNAL_COMMAND_STATE;
```

```
66          signal EXE_STEP : INTERNAL_EXE_STATE;
67          --Latch User Data until RAM is ready to write it
68          signal DataToRAM : std_logic_vector(7 downto 0);
69
70          begin
71          process(extClk,rst)
72
73          procedure RD ( BankAdr:in std_logic_vector(1 downto 0);
74                         ColAdr:in std_logic_vector(9 downto 0) ) is
75          begin
76             RAS   <='1';
77             CAS   <='0';
78             WE <='1';
79             BA <= BankAdr;
80             A   <= "001" & ColAdr;--Note that A10/AP is set to 1
81          end procedure RD;
82
83          procedure WR ( BankAdr:in std_logic_vector(1 downto 0);
84                         ColAdr:in std_logic_vector(9 downto 0);
85                         Data:in std_logic_vector(7 downto 0) ) is
86          begin
87             RAS   <='1';
88             CAS   <='0';
89             WE <='0';
90             BA <= BankAdr;
91             A   <= "001" & ColAdr;
92             D <= DataToRAM;
93          end procedure WR;
94
95          procedure ACT ( BankAdr:in std_logic_vector(1 downto 0);
96                         RowAdr:in std_logic_vector(12 downto 0) ) is
97          begin
98             RAS   <='0';
99             CAS   <='1';
100            WE <='1';
101            BA <= BankAdr;
102            A   <= RowAdr;
103         end ACT;
104
105         procedure MRS ( BankAdr:in std_logic_vector(1 downto 0);
106                        Adr:in std_logic_vector(12 downto 0) ) is
107         begin
108            RAS   <='0';
109            CAS   <='0';
110            WE <='0';
111            BA <= BankAdr;
112            A   <= Adr;
113         end MRS;
114
115         procedure NOP is
116         begin
117            RAS   <='1';
118            CAS   <='1';
119            WE <='1';
120            D <= "ZZZZZZZZ";
121         end procedure NOP;
122
123         procedure PREA is
124         begin
125            RAS   <='0';
126            CAS   <='1';
127            WE <='0';
128            A(10) <= '1'; --set AP (auto precharge)
129         end procedure PREA;
130
```

```
131         procedure REFA is
132         begin
133            RAS   <='0';
134            CAS   <='0';
135            WE <='1';
136         end procedure REFA;
137
138
139         begin
140         if (rst ='0') then
141            COMMAND  <= Locked;--After reset we wait for PwrUp Pin (In Locked State)
142            EXE_STEP<= Step1;
143            rdy <= '0';--We are not rdy for Commands
144            NOP;--Init Cas,Ras etc.
145            BA <= "00";
146            A <= "0000000000000";
147            S0 <= '0';
148            DQMB <= '1';--Disable Output of all RAM-Cells
149            CKE0 <= '1';--Enable SDRAM Clock
150
151         elsif (extClk'event and extClk = '0') then
152         --extClk is the same signal as the SDRAM Clock
153            case COMMAND is
154               when PowerUp =>
155                  rdy <= '0';
156                  case EXE_STEP is
157                  when Step1 =>
158                     PREA;
159                     EXE_STEP <= Step2;
160                  when Step2 =>
161                     NOP; --Be sure to wait for Trp
162                     EXE_STEP <= Step3;
163                  when Step3 =>
164                     REFA;--1st Refresh
165                     EXE_STEP <= Step4;
166                  when Step4 =>
167                     REFA;--2nd Refresh
168                     EXE_STEP <= Step5;
169                  when Step5 =>
170                     REFA;--3rd
171                     EXE_STEP <= Step6;
172                  when Step6 =>
173                     REFA;--4th
174                     EXE_STEP <= Step7;
175                  when Step7 =>
176                     REFA;--5th
177                     EXE_STEP <= Step8;
178                  when Step8 =>
179                     REFA;--6th
180                     EXE_STEP <= Step9;
181                  when Step9 =>
182                     REFA;--7th
183                     EXE_STEP <= Step10;
184                  when Step10 =>
185                     REFA;--8th
186                     COMMAND <= SetModeReg;--After PowerUp we have to configure the ModeRegister
187                     EXE_STEP <= Step1;--Reset state variable
188                  end case; --End of PowerUp State
189
190               when SetModeReg =>
191                     rdy <= '0';
192                  case EXE_STEP is
193                  when Step1 =>
194                     MRS("11","0000000110000");--Set CAS latency = 3
195                     EXE_STEP <= Step2;
```

```
196                when Step2 =>
197                   NOP;
198                   EXE_STEP <= Step3;
199                when Step3 =>
200                   NOP;
201                   COMMAND <= Idle;--Command Finished, next state is Idle
202                   EXE_STEP <= Step1;--Reset state variable
203                when others => null;--becarse we don't use all 10 step
204                end case;
205
206             when Read =>
207                rdy <= '0';
208                DQMB <= '0';--Enable RAM-Cells
209                case EXE_STEP is
210                when Step1 =>
211                   ACT("00",Row);--Activate Bank & Column
212                   EXE_STEP <= Step2;
213                when Step2 =>
214                   NOP;--Wait Trcd
215                   EXE_STEP <= Step3;
216                when Step3 =>
217                   NOP;--Wait Trcd
218                   EXE_STEP <= Step4;
219                when Step4 =>
220                   RD("00",Col);--Read Col and do autoprecharge
221                   EXE_STEP <= Step5;
222                when Step5 =>
223                   NOP;--Wait for CAS latency (1st)
224                   EXE_STEP <= Step6;
225                when Step6 =>
226                   NOP;--Wait for CAS latency (2nd)
227                   EXE_STEP <= Step7;
228                when Step7 =>
229          --Data is ready here.
230                   DataOut <= D;
231                   COMMAND <= Idle;--Command Finished, next state is Idle
232                   EXE_STEP <= Step1;
233                when others => null;--becarse we don't use all 10 step
234                end case;
235
236             when Write =>
237                rdy <= '0';
238                DQMB <= '0';--Enable RAM-Cells
239                case EXE_STEP is
240                when Step1 =>
241                   ACT(Bank,Row);--activate Bank & Column
242                   EXE_STEP <= Step2;
243                when Step2 =>
244                   NOP;
245                   EXE_STEP <= Step3;
246                when Step3 =>
247                   WR(Bank,Col,DataToRAM);--write + autoprecharge
248                   EXE_STEP <= Step4;
249                when Step4 =>
250                   NOP;
251                   COMMAND <= Idle;
252                   EXE_STEP <= Step1;
253                when others => null;
254                end case;
255
256             when Idle =>
257                rdy <= '1';--We are rdy for Commands
258                DQMB <= '1';--Disable RAM-Cells
259                if (WrRAM = '0') then
260                   rdy <= '0';
```

```
261                  DataToRAM <= DataIn; --Latch Data to be written to RAM
262                  COMMAND <= Write;
263                  EXE_STEP <= Step1;
264               elsif (RdRAM = '0') then
265                  rdy <= '0';
266                  COMMAND <= Read;
267                  EXE_STEP <= Step1;
268               else
269                  REFA; --No RD or WR requeste - do the refresh
270               end if;
271
272            when Locked =>
273               if (PwrUp = '1') then
274                  rdy <= '0';
275                  COMMAND <= PowerUp;
276                  EXE_STEP <= Step1;
277               else
278                  NOP;
279               end if;
280         end case;
281      end if;
282      end process;
283         CK0 <= extClk;--Transfer clock to RAM
284      end Beh;
285
```

```
--------------------------------------------------------------------
--                                                                --
-- Handling Image Data from NOAA Weather Satellites               --
--                                                                --
-- Master Thesis DTU (IMM), 2006                                  --
--------------------------------------------------------------------
-- Authors:                                                       --
--       Soeren Soelling Christiansen                             --
--       Eskil Binzer                                             --
--------------------------------------------------------------------
-- Description of Code:                                           --
--       Simple Testbench for RAM power up and a single           --
--       write- and read cycle                                    --
--       Beh. model of DRAM Controller.                           --
--    This model have to be Simulated with the Hynix hy57v28820   --
-- Version:                                                       --
--    1.1   (03-05-06)                                           --
-- Changes since last version:                                    --
--    None! - This is the 1. approach-model!                      --
--                                                                --
--------------------------------------------------------------------

LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY firsttest_tb5_vhd_tb IS
END firsttest_tb5_vhd_tb;

ARCHITECTURE behavior OF firsttest_tb5_vhd_tb IS
   COMPONENT hy57v28820hct
   PORT(
         VDD1 : IN std_logic;
         VDDQ3 : IN std_logic;
         NC4 : IN std_logic;
         VSSQ6 : IN std_logic;
         NC7 : IN std_logic;
         VDDQ9 : IN std_logic;
         NC10 : IN std_logic;
         VSSQ12 : IN std_logic;
         NC13 : IN std_logic;
         VDD14 : IN std_logic;
         NC15 : IN std_logic;
         WEB : IN std_logic;
         CASB : IN std_logic;
         RASB : IN std_logic;
         CSB : IN std_logic;
         BA0 : IN std_logic;
         BA1 : IN std_logic;
         A10 : IN std_logic;
         A0 : IN std_logic;
         A1 : IN std_logic;
         A2 : IN std_logic;
         A3 : IN std_logic;
         VDD27 : IN std_logic;
         VSS28 : IN std_logic;
         A4 : IN std_logic;
         A5 : IN std_logic;
         A6 : IN std_logic;
         A7 : IN std_logic;
         A8 : IN std_logic;
         A9 : IN std_logic;
         A11 : IN std_logic;
         NC36 : IN std_logic;
         CKE : IN std_logic;
```

```
66                CLK : IN std_logic;
67                DQM : IN std_logic;
68                NC40 : IN std_logic;
69                VSS41 : IN std_logic;
70                NC42 : IN std_logic;
71                VDDQ43 : IN std_logic;
72                NC45 : IN std_logic;
73                VSSQ46 : IN std_logic;
74                NC48 : IN std_logic;
75                VDDQ49 : IN std_logic;
76                NC51 : IN std_logic;
77                VSSQ52 : IN std_logic;
78                VSS54 : IN std_logic;
79                DQ0 : INOUT std_logic;
80                DQ1 : INOUT std_logic;
81                DQ2 : INOUT std_logic;
82                DQ3 : INOUT std_logic;
83                DQ4 : INOUT std_logic;
84                DQ5 : INOUT std_logic;
85                DQ6 : INOUT std_logic;
86                DQ7 : INOUT std_logic);
87          END COMPONENT;
88
89          COMPONENT firsttest
90          PORT(
91             wr : IN std_logic;
92             rd : IN std_logic;
93             clock : IN std_logic;
94             reset : IN std_logic;
95             pwrup : IN std_logic;
96             D : INOUT std_logic_vector(7 downto 0);
97             DataOut : OUT std_logic_vector(7 downto 0);
98             ready : OUT std_logic;
99             CK0 : OUT std_logic;
100            CKE0 : OUT std_logic;
101            CK1 : OUT std_logic;
102            CKE1 : OUT std_logic;
103            RAS : OUT std_logic;
104            CAS : OUT std_logic;
105            WE : OUT std_logic;
106            BA : OUT std_logic_vector(1 downto 0);
107            DQMB : OUT std_logic;
108            S0 : OUT std_logic;
109            S1 : OUT std_logic;
110            A : OUT std_logic_vector(12 downto 0)
111            );
112         END COMPONENT;
113
114         SIGNAL wr :  std_logic;
115         SIGNAL DataOut :  std_logic_vector(7 downto 0);
116         SIGNAL rd :  std_logic;
117         SIGNAL clock :  std_logic;
118         SIGNAL reset :  std_logic;
119         SIGNAL pwrup :  std_logic;
120         SIGNAL ready :  std_logic;
121         SIGNAL CK0 :  std_logic;
122         SIGNAL CKE0 :  std_logic;
123         SIGNAL CK1 :  std_logic;
124         SIGNAL CKE1 :  std_logic;
125         SIGNAL RAS :  std_logic;
126         SIGNAL CAS :  std_logic;
127         SIGNAL WE :  std_logic;
128         SIGNAL BA :  std_logic_vector(1 downto 0);
129         SIGNAL DQMB :  std_logic;
130         SIGNAL S0 :  std_logic;
```

```
131          SIGNAL S1 :  std_logic;
132          SIGNAL D :  std_logic_vector(7 downto 0);
133          SIGNAL A :  std_logic_vector(12 downto 0);
134
135       BEGIN
136          RAM: hy57v28820hct PORT MAP(
137                VDD1 => '1',
138                DQ0 => D(0),
139                VDDQ3 => '1',
140                NC4 => 'L',
141                DQ1 => D(1),
142                VSSQ6 => '0',
143                NC7 => 'L',
144                DQ2 => D(2),
145                VDDQ9 => '1',
146                NC10 => 'L',
147                DQ3 => D(3),
148                VSSQ12 => '0',
149                NC13 => 'L',
150                VDD14 => '1',
151                NC15 => 'L',
152                WEB => WE,
153                CASB => CAS,
154                RASB => RAS,
155                CSB => S0,
156                BA0 => BA(0),
157                BA1 => BA(1),
158                A10 => A(10),
159                A0 => A(0),
160                A1 => A(1),
161                A2 => A(2),
162                A3 => A(3),
163                VDD27 => '1',
164                VSS28 => '0',
165                A4 => A(4),
166                A5 => A(5),
167                A6 => A(6),
168                A7 => A(7),
169                A8 => A(8),
170                A9 => A(9),
171                A11 => A(11),
172                NC36 => 'L',
173                CKE => '1',
174                CLK => CK0,
175                DQM => DQMB,
176                NC40 => 'L',
177                VSS41 => '0',
178                NC42 => 'L',
179                VDDQ43 => '1',
180                DQ4 => D(4),
181                NC45 => 'L',
182                VSSQ46 => '0',
183                DQ5 => D(5),
184                NC48 => 'L',
185                VDDQ49 => '1',
186                DQ6 => D(6),
187                NC51 => 'L',
188                VSSQ52 => '0',
189                DQ7 => D(7),
190                VSS54 => '0');
191
192          uut: firsttest PORT MAP(
193             wr => wr,
194             DataOut => DataOut,
195             rd => rd,
```

```
196             clock => clock,
197             reset => reset,
198             pwrup => pwrup,
199             ready => ready,
200             CK0 => CK0,
201             CKE0 => CKE0,
202             CK1 => CK1,
203             CKE1 => CKE1,
204             RAS => RAS,
205             CAS => CAS,
206             WE => WE,
207             BA => BA,
208             DQMB => DQMB,
209             S0 => S0,
210             S1 => S1,
211             D => D,
212             A => A
213          );
214
215          tb : PROCESS BEGIN
216
217          if (now < 2 ns) then
218             reset <= '0';
219             Wr <= '1';  --disable
220             Rd <= '1';
221             PwrUp <= '0'; --disable
222          else reset <= '1';
223          end if;
224
225          if (now > 200 us) and (now < 201 us) then
226       --Wait for power stable
227             PwrUp <= '1';
228          else PwrUp <= '0';
229          end if;
230
231       --Write to RAM
232          if (now > 300 us) and (now < 301 us) then
233             Report "We try to write now...";
234             Wr <= '0';
235          else
236             Wr <= '1';
237          end if;
238       --Read from RAM
239          if (now > 304 us) and (now < 305 us) then
240             Report "We try to read now...";
241             Rd <= '0';
242          else
243             Rd <= '1';
244          end if;
245       --Generate clock to RAM-Controller and RAM
246          Clock <= '1';
247          wait for 200 ns;
248          Clock <= '0';
249          wait for 200 ns;
250
251          if (now > 400 us) then wait; -- will wait forever
252          end if;
253          END PROCESS;
254       END;
255
```

13.2 VHDL-kode for endelige DFU

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6
7   entity main is Port (
8      SYSCLK : in std_logic;
9      RST : in  STD_LOGIC;
10     PWRUP : in  STD_LOGIC;
11     ALE : in  STD_LOGIC;
12     RD : in  STD_LOGIC;
13     WR : in  STD_LOGIC;
14     AD : inout  STD_LOGIC_VECTOR (7 downto 0);
15     PCS1 : in  STD_LOGIC;
16     BUSYLED : out  STD_LOGIC;
17     FULLLED : out  STD_LOGIC;
18     TEST : out  STD_LOGIC;
19     LED : out  STD_LOGIC;
20  --Below we have connections to SDRAM Module
21     CK0 : out std_logic;
22     CK1 : out std_logic;
23     RAS : out std_logic;
24     CAS : out std_logic;
25     WE : out std_logic;
26     BA : out std_logic_vector(1 downto 0);
27     DQMB : out std_logic_vector(2 downto 0);
28     S0 : out std_logic;
29     S1 : out std_logic;
30     D : inout std_logic_vector(63 downto 0);
31     A : out std_logic_vector(12 downto 0);
32  --NOAA pins
33     NOAA_CLK : in std_logic;
34     NOAA_DATA : in std_logic);
35  end main;
36
37  architecture RTL of main is
38
39  component latch8P is port(
40     clk   : in  std_logic;
41     rst   : in  std_logic;
42     ce    : in  std_logic;
43     d     : in  std_logic_vector(7 downto 0);
44     q     : out std_logic_vector(7 downto 0));
45  end component;
46
47  component latch8N is port(
48     clk   : in  std_logic;
49     rst   : in  std_logic;
50     ce    : in  std_logic;
51     oe    : in  std_logic;
52     d     : in  std_logic_vector(7 downto 0);
53     q     : out std_logic_vector(7 downto 0));
54  end component;
55
56  component bustrans8 is port(
57     oe    : in  std_logic;
58     ce    : in  std_logic;
59     input : in  std_logic_vector(7 downto 0);
60     output: out std_logic_vector(7 downto 0));
61  end component;
62
63  component latchN is port(
64     clk   : in  std_logic;
65     rst   : in  std_logic;
```

```
66          ce    : in  std_logic;
67          d     : in  std_logic;
68          not_q : out std_logic;
69          q     : out std_logic);
70       end component;
71
72       component latchP is port(
73          clk   : in  std_logic;
74          rst   : in  std_logic;
75          ce    : in  std_logic;
76          d     : in  std_logic;
77          not_q : out std_logic;
78          q     : out std_logic);
79       end component;
80
81       component Controller is Port(
82       --Signals to the DRAM-Controller
83          extClk : in std_logic;
84          rst : in std_logic;
85          PwrUp : in std_logic;
86          WrRAM : in std_logic;
87          DataIn : in std_logic_vector(63 downto 0);
88          RdRAM : in std_logic;
89          DataOut : out std_logic_vector(63 downto 0);
90          rdy : out std_logic;
91       --SDRAM Address Signals
92          RowRd : in std_logic_vector(11 downto 0);
93          ColRd : in std_logic_vector(8 downto 0);
94          BankRd : in std_logic_vector(1 downto 0);
95          RowWr : in std_logic_vector(11 downto 0);
96          ColWr : in std_logic_vector(8 downto 0);
97          BankWr : in std_logic_vector(1 downto 0);
98       --ChipSel : in std_logic_vector(1 downto 0);
99          CK0 : out std_logic;
100         CK1 : out std_logic;
101         RAS : out std_logic;
102         CAS : out std_logic;
103         WE : out std_logic;
104         BA : out std_logic_vector(1 downto 0);
105         DQMB : out std_logic_vector(2 downto 0);
106         S0 : out std_logic;
107         D : inout std_logic_vector(63 downto 0);
108         A : out std_logic_vector(12 downto 0));
109      end component;
110
111      component counter8 is Port (
112         count : in std_logic;
113         rst : in std_logic;
114         result : out std_logic);
115      end component;
116
117      component WrCounter is Port (
118         count : in std_logic;
119         rst : in std_logic;
120         ce : in std_logic;
121         result : out std_logic_vector(22 downto 0));
122      end component;
123
124      component RdCounter is Port (
125         count : in std_logic;
126         rst : in std_logic;
127         ce : in std_logic;
128         result : out std_logic_vector(22 downto 0));
129      end component;
130
```

```
131         component demux is Port (
132            oe: in std_logic;
133            sel: in std_logic_vector(7 downto 0);
134            d: out std_logic_vector(8 downto 0));
135         end component;
136
137         component comparator8 is Port(
138            empty : out  std_logic;
139            full : out std_logic;
140            wradr : in  std_logic_vector(22 downto 0);
141            rdadr: in std_logic_vector(22 downto 0));
142         end component;
143
144         component adr_transiver is Port(
145            oe : in  std_logic;
146            input : in  std_logic_vector(20 downto 0);
147            output : out std_logic_vector(20 downto 0));
148         end component;
149
150         component INVgate is Port(    a: in  std_logic;
151            b: out std_logic);
152         end component;
153
154         component FrameSyncronizer is port(
155            NoaaClk  : in  std_logic;
156            NoaaData : in  std_logic;
157            Reset    : in  std_logic;
158            Output   : out std_logic_vector(63 downto 0);
159            SyncLed  : out std_logic;
160            WrReq    : out std_logic;
161            WrDone   : in std_logic;
162            Busy     : out std_logic);
163         end component;
164
165         signal NET12,NET13 : std_logic_vector(22 downto 0);
166         signal NET9, NET2:std_logic_vector(7 downto 0);
167         signal NET1,NET3, NET5, NET6, NET7 : std_logic;
168         signal NET10: std_logic_vector(8 downto 0);
169         signal RamFull, RamEmpty, RamBusy : std_logic;
170         signal DataToRAM : std_logic_vector(63 downto 0);
171         signal SyncLed: std_logic;
172         signal DataFromRAM : std_logic_vector(63 downto 0);
173
174         begin
175         datalatch: latch8P port map(
176            clk => WR,
177            rst => '1',
178            ce => NET10(1),
179            d => AD,
180            q => NET2);
181
182         adrlatch: latch8N port map(
183            clk => ALE,
184            rst => '1',
185            ce => '0',--PCS1,--if static 0, we are sure to latch address
186            oe => '0',
187            d => AD,
188            q => NET9);
189
190         trans0: bustrans8 port map(
191            oe =>RD,
192            ce =>NET10(0),
193            input =>dataFromRam(7 downto 0),
194            output =>AD );
195
```

```
196         trans1: bustrans8 port map(
197            oe =>RD,
198            ce =>NET10(1),
199            input =>dataFromRam(15 downto 8),
200            output =>AD );
201
202         trans2: bustrans8 port map(
203            oe =>RD,
204            ce =>NET10(2),
205            input => dataFromRam(23 downto 16),
206            output =>AD );
207
208         trans3: bustrans8 port map(
209            oe =>RD,
210            ce =>NET10(3),
211            input => dataFromRam(31 downto 24),
212            output =>AD );
213
214         trans4: bustrans8 port map(
215            oe =>RD,
216            ce =>NET10(4),
217            input => dataFromRam(39 downto 32),
218            output =>AD );
219
220         trans5: bustrans8 port map(
221            oe =>RD,
222            ce =>NET10(5),
223            input(7 downto 0) => dataFromRam(47 downto 40),
224            output =>AD );
225
226         trans6: bustrans8 port map(
227            oe =>RD,
228            ce =>NET10(6),
229            input => dataFromRam(55 downto 48),
230            output =>AD );
231
232         trans7: bustrans8 port map(
233            oe =>RD,
234            ce =>NET10(7),
235            input(7 downto 0) => dataFromRam(63 downto 56),
236            output =>AD );
237
238         trans8: bustrans8 port map(
239            oe =>RD,
240            ce =>NET10(8),
241            input(0) =>RamEmpty,
242            input(1) =>RamFull,
243            input(2) => RamBusy,
244            input(7 downto 3) =>"00000",
245            output =>AD );
246
247         frsync: FrameSyncronizer port map(
248            NoaaClk => NOAA_CLK,
249            NoaaData => NOAA_DATA,
250            Reset => '0',
251            Output => DataToRAM,
252            SyncLed =>SyncLed,
253            WrReq => net6,
254            WrDone => net5,
255            Busy => RamBusy);
256
257         invRdLatch: INVgate PORT MAP (
258            a => NET5,
259            b => NET1);
260
```

```
261         RDlatch:latchN port map(
262            clk => RD,
263            rst => NET1,
264            ce => NET10(0),
265            d => '1',
266            not_q => NET7,
267            q => OPEN );
268
269         RAMCtrl: Controller PORT MAP(
270            extClk => SYSCLK,
271            rst => RST,
272            PwrUp => PWRUP,
273            WrRAM => NET6,
274            DataIn => DataToRAM,
275            RdRAM => NET7,
276            DataOut => dataFromRam,
277            rdy => NET5,
278            RowRd => NET13(11 downto 0),
279            ColRd => NET13(20 downto 12),
280            BankRd => NET13(22 downto 21),
281            RowWr => NET12(11 downto 0),
282            ColWr => NET12(20 downto 12),
283            BankWr => NET12(22 downto 21),
284            CK0 => CK0,
285            CK1 => CK1,
286            RAS => RAS,
287            CAS => CAS,
288            WE => WE,
289            BA => BA,
290            DQMB => DQMB,
291            S0 => open,
292            D => D,
293            A => A);
294
295         adr: WrCounter PORT MAP (
296            count=> NET6,
297            rst=> RST,
298            ce => RamFull,
299            result=> NET12);
300
301         Rdadr: RdCounter PORT MAP (
302            count=> WR,
303            rst=> RST,
304            ce => NET10(2),
305            result=> NET13);
306
307         ChipSel: demux PORT MAP (
308            oe => PCS1,
309            sel=> NET9,
310            d=> NET10);
311
312         memStatus: comparator8 PORT MAP(
313            empty => RamEmpty,
314            full => RamFull,
315            wradr => NET12,
316            rdadr =>NET13);
317
318         S0 <= '0';
319         S1 <= '1';
320
321         LED <= SyncLed;
322         FULLLED <= RamFull;
323         BUSYLED <= not NET5;
324         TEST <= '0'; --be sure to have input on all pins connected to Beck
325         end RTL;
```

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity Controller is
7   Port(
8   --Signals to the DRAM-Controller
9            extClk : in std_logic;
10           rst : in std_logic;
11           PwrUp : in std_logic;
12           WrRAM : in std_logic;
13           DataIn : in std_logic_vector(63 downto 0);
14           RdRAM : in std_logic;
15           DataOut : out std_logic_vector(63 downto 0);
16           rdy : out std_logic;
17  --SDRAM Address Signals
18           RowRd : in std_logic_vector(11 downto 0);
19           ColRd : in std_logic_vector(8 downto 0);
20           BankRd : in std_logic_vector(1 downto 0);
21
22           RowWr : in std_logic_vector(11 downto 0);
23           ColWr : in std_logic_vector(8 downto 0);
24           BankWr : in std_logic_vector(1 downto 0);
25           --ChipSel : in std_logic_vector(1 downto 0);
26    --Signals to the SDRAM
27           CK0 : out std_logic;
28           CK1 : out std_logic;
29           RAS : out std_logic;
30           CAS : out std_logic;
31           WE : out std_logic;
32           BA : out std_logic_vector(1 downto 0);
33           DQMB : out std_logic_vector(2 downto 0);
34           S0 : out std_logic;
35           D : inout std_logic_vector(63 downto 0);
36           A : out std_logic_vector(12 downto 0));
37  end Controller;
38
39
40  architecture Beh of Controller is
41
42  --Main States in this SDRAM Controller.
43  --Idle: Waits for User Read/Write Requist and performs Refresh
44  --Locked: After reset this State kept until PwrUp is activated
45  --(To Satisfy the 200us Wait after Power On)
46  --Only NOP's to SDRAM are issued in this State.
47  type INTERNAL_COMMAND_STATE is (PowerUp,SetModeReg,Read,Write,Idle,Locked);
48  --Execution States (inside the Main States). SDRAM Commands takes Max 10 Clocks
49  type INTERNAL_EXE_STATE is (Step1,Step2,Step3,Step4,Step5,Step6,Step7,Step8,Step9,Step10
50
51  signal COMMAND :  INTERNAL_COMMAND_STATE;
52  signal EXE_STEP : INTERNAL_EXE_STATE;
53
54  begin
55  process(extClk,rst)
56
57  procedure ACT ( BankAdr:in std_logic_vector(1 downto 0);
58                  RowAdr:in std_logic_vector(12 downto 0) ) is
59  begin
60     RAS   <='0';
61     CAS   <='1';
62     WE <='1';
63     BA <= BankAdr;
64     A   <= RowAdr;
65  end ACT;
```

```
66
67          procedure MRS ( BankAdr:in std_logic_vector(1 downto 0);
68                          Adr:in std_logic_vector(12 downto 0) ) is
69          begin
70             RAS   <='0';
71             CAS   <='0';
72             WE <='0';
73             BA <= BankAdr;
74             A   <= Adr;
75          end MRS;
76
77
78          procedure NOP is
79          begin
80             RAS   <='1';
81             CAS   <='1';
82             WE <='1';
83             D <= "ZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZZ";
84          end procedure NOP;
85
86          procedure PREA is
87          begin
88             RAS   <='0';
89             CAS   <='1';
90             WE <='0';
91             A(10) <= '1'; --set AP (auto precharge)
92          end procedure PREA;
93
94          procedure REFA is
95          begin
96             RAS   <='0';
97             CAS   <='0';
98             WE <='1';
99          end procedure REFA;
100
101
102
103         begin
104         if (rst ='0') then
105            PREA;
106            COMMAND  <= PowerUp;--Locked;--After reset we wait for PwrUp Pin (In Locked State)
107            EXE_STEP<= Step1;
108            rdy <= '0';--We are not rdy for Commands
109            NOP;--Init Cas,Ras etc.
110            BA <= "00";
111            --A <= "0000000000000";
112            S0 <= '0';
113            DQMB <= "111";--Disable Output of all RAM-Cells
114
115         elsif (extClk'event and extClk = '0') then
116            case COMMAND is
117               when PowerUp =>
118                  rdy <= '0';
119                  case EXE_STEP is
120                  when Step1 =>
121                     PREA;
122                     EXE_STEP <= Step2;
123                  when Step2 =>
124                     NOP; --Be sure to wait for Trp
125                     EXE_STEP <= Step3;
126                  when Step3 =>
127                     REFA;--1st Refresh
128                     EXE_STEP <= Step4;
129                  when Step4 =>
130                     REFA;--2nd Refresh
```

```
131                     EXE_STEP <= Step5;
132                  when Step5 =>
133                     REFA;--3rd
134                     EXE_STEP <= Step6;
135                  when Step6 =>
136                     REFA;--4th
137                     EXE_STEP <= Step7;
138                  when Step7 =>
139                     REFA;--5th
140                     EXE_STEP <= Step8;
141                  when Step8 =>
142                     REFA;--6th
143                     EXE_STEP <= Step9;
144                  when Step9 =>
145                     REFA;--7th
146
147                     EXE_STEP <= Step10;
148                  when Step10 =>
149                     REFA;--8th
150                     COMMAND <= SetModeReg;--After PowerUp we have to configure the ModeRegister
151                     EXE_STEP <= Step1;--Reset state variable
152                  end case; --End of PowerUp State
153
154               when SetModeReg =>
155                     --rdy <= '0';
156                  case EXE_STEP is
157                  when Step1 =>
158                     --MRS("11","0000000110000");--CAS LATENCY = 3
159                     MRS("11","0000000100000");--CAS LATENCY = 2
160                     EXE_STEP <= Step2;
161                  when Step2 =>
162                     NOP;
163                     EXE_STEP <= Step3;
164                  when Step3 =>
165                     NOP;
166                     COMMAND <= Idle;--Command Finished, next state is Idle
167                     EXE_STEP <= Step1;--Reset state variable
168                  when others => null;--becoarse we don't use all 10 step
169                  end case;
170
171               when Read =>
172
173                  --DQMB <= not DQMB;
174                  DQMB <= "000";--Enable RAM-Cells
175                  case EXE_STEP is
176                  when Step1 =>
177                     NOP;
178                     A <= '0'&RowRd;
179                     BA <= BankRd;
180
181                     EXE_STEP <= Step2;
182                  when Step2 =>
183                     RAS   <='1';--NOP
184                     CAS   <='1';
185                     WE <='1';
186
187                     EXE_STEP <= Step3;
188                  when Step3 =>
189                     RAS   <='0';--ACT
190                     CAS   <='1';
191                     WE <='1';
192
193                     EXE_STEP <= Step4;
194                  when Step4 =>
195                     RAS   <='1';--NOP
```

```
196                  CAS   <='1';
197                  WE <='1';
198                  EXE_STEP <= Step5;
199               when Step5 =>
200                  RAS   <='1';--NOP
201                  CAS   <='1';
202                  WE <='1';
203                  A <= "0010" & ColRd;
204                  EXE_STEP <= Step6;
205               when Step6 =>
206                  RAS   <='1';--RD
207                  CAS   <='0';
208                  WE <='1';
209                  EXE_STEP <= Step7;
210               when Step7 =>
211                  RAS   <='1';--NOP
212                  CAS   <='1';
213                  WE <='1';
214                  EXE_STEP <= Step8;
215               when Step8 =>
216                  RAS   <='1';--NOP
217                  CAS   <='1';
218                  WE <='1';
219                  --when CL=2 then data is ready here
220                  DataOut <= D;
221                  EXE_STEP <= Step9;
222               when Step9 =>
223                  RAS   <='1';--NOP
224                  CAS   <='1';
225                  WE <='1';
226                  --when CL=3 then data is ready here
227                  EXE_STEP <= Step10;
228               when Step10 =>
229                  RAS   <='1';--NOP
230                  CAS   <='1';
231                  WE <='1';
232                  DQMB <= "111";
233                  rdy <= '0';
234                  COMMAND <= Idle;--Command Finished, next state is Idle
235                  EXE_STEP <= Step1;
236               when others => null;--becarse we don't use all 10 step
237               end case;
238
239
240            when Write =>
241               case EXE_STEP is
242               when Step1 =>
243
244                  RAS   <='1';--NOP
245                  CAS   <='1';
246                  WE <='1';
247                  --A <= Row;
248                  --BA <= Bank;
249                  DQMB <= "000";--Enable RAM-Cells
250                  EXE_STEP <= Step2;
251               when Step2 =>
252                  --ACT(Bank,Row);--Bank & Column
253                  RAS   <='1';--NOP
254                  CAS   <='1';
255                  WE <='1';
256                  A <= '0'&RowWr;
257                  BA <= BankWr;
258
259                  EXE_STEP <= Step3;
260               when Step3 =>
```

```
261                  RAS   <='0';--ACT
262                  CAS   <='1';
263                  WE <='1';
264                  D <= DataIn;
265
266                  EXE_STEP <= Step4;
267               when Step4 =>
268                  --WR(Bank,Col,DataToRAM);--write + autoprecharge
269                  RAS   <='1';--NOP
270                  CAS   <='1';
271                  WE <='1';
272                  rdy <= '0';
273                  EXE_STEP <= Step5;
274               when Step5 =>
275                  RAS   <='1';--NOP
276                  CAS   <='1';
277                  WE <='1';
278                  A <= "0010" & ColWr;
279                  EXE_STEP <= Step6;
280               when Step6 =>
281                  RAS   <='1';--WR
282                  CAS   <='0';
283                  WE <='0';
284                  EXE_STEP <= Step7;
285               when Step7 =>
286                  --RAS <='1';--NOP
287                  --CAS <='1';
288                  --WE  <='1';
289                  RAS   <='1';--WR
290                  CAS   <='0';
291                  WE <='0';
292       --           COMMAND <= Idle;
293       --           EXE_STEP <= Step1;
294                  EXE_STEP <= Step8;--test
295               when Step8 =>--test state 19-7-06
296                  RAS   <='1';--NOP
297                  CAS   <='1';
298                  WE <='1';
299                  COMMAND <= Idle;
300                  EXE_STEP <= Step1;
301
302
303               when others => null;
304               end case;
305
306
307            when Idle =>
308               --NOP; --test 17-7
309               rdy <= '1';--We are rdy for Commands
310               DQMB <= "111";--Disable RAM-Cells
311               if (WrRAM = '0') then
312                  COMMAND <= Write;
313                  EXE_STEP <= Step1;
314               elsif (RdRAM = '0') then
315                  --rdy <= '0';
316                  COMMAND <= Read;
317                  EXE_STEP <= Step1;
318               else
319                  REFA;
320               end if;
321
322            when Locked =>
323               if (PwrUp = '1') then
324                  --rdy <= '0';
325                  COMMAND <= PowerUp;
```

```
326                   EXE_STEP <= Step1;
327                else
328                   NOP;
329                end if;
330
331             end case;
332
333          end if;
334          end process;
335
336             CK0 <= extClk;
337             CK1 <= extClk;
338
339          end Beh;
340
```

```
----------------------------------------------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity WrCounter is
Port (
          count : in std_logic;
          rst : in std_logic;
          --oe : in std_logic;
          ce : in std_logic;
          result : out std_logic_vector(22 downto 0));
end WrCounter;

architecture Beh of WrCounter is

signal value: std_logic_vector(22 downto 0);

begin
process(rst,count) begin

if (rst = '0') then
   value <= "00000000000000000000000";
elsif (count'event and count = '0' and ce = '0') then
   value <= value + "00000000000000000000001";
end if;
end process;
   result <= value;
end Beh;

----------------------------------------------------------------------------------------------------
------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity RdCounter is
Port (
          count : in std_logic;
          rst : in std_logic;
          ce : in std_logic;
          result : out std_logic_vector(22 downto 0));
end RdCounter;

architecture Beh of RdCounter is

signal value: std_logic_vector(22 downto 0);

begin
process(rst,count) begin

if (rst = '0') then
   value <= "00000000000000000000000";
elsif (count'event and count = '1' and ce = '1') then
   value <= value + "00000000000000000000001";
end if;
end process;
   result <= value;
end Beh;
```

```
65
66      ------------------------------------------------------------------------------------------------
        ------------
67
68
69
```

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3   use IEEE.STD_LOGIC_ARITH.ALL;
4   use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6   entity FrameSyncronizer is port(
7           NoaaClk  : in  std_logic;
8           NoaaData : in  std_logic;
9           Reset    : in  std_logic;
10          Output   : out std_logic_vector(63 downto 0);
11          SyncLed  : out std_logic;
12          WrReq    : out std_logic;
13          WrDone   : in std_logic;
14          Busy     : out std_logic);
15  end FrameSyncronizer;
16
17  architecture RTL of FrameSyncronizer is
18
19  component shiftregister is port(
20          clk   : in  std_logic;
21          input : in  std_logic;
22          output: out std_logic_vector(63 downto 0));
23  end component;
24
25  component comparator is port(
26          input : in  std_logic_vector(59 downto 0);
27          match : out std_logic);
28  end component;
29
30  component latch64 is port(
31          clk   : in  std_logic;
32          ce    : in std_logic;
33          d     : in  std_logic_vector(63 downto 0);
34          q     : out std_logic_vector(63 downto 0));
35  end component;
36
37  component latchN is port(
38          clk      : in  std_logic;
39          rst      : in  std_logic;
40          ce       : in std_logic;
41          d        : in  std_logic;
42          q        : out std_logic;
43          not_q    : out std_logic);
44  end component;
45
46  component latchP is port(
47          clk      : in  std_logic;
48          rst      : in  std_logic;
49          d        : in  std_logic;
50          q        : out std_logic;
51          not_q    : out std_logic);
52  end component;
53
54  component ORgate is port(
55          a        : in  std_logic;
56          b        : in std_logic;
57          c        : out std_logic);
58  end component;
59
60  component counter is port(
61          clk      : in  std_logic;
62          rst      : in  std_logic;
63          bit64    : out  std_logic);
64  end component;
65
```

```
66        component INVgate is port(
67                a        : in  std_logic;
68                b        : out std_logic);
69        end component;
70
71        signal NET1 : std_logic_vector(63 downto 0);
72        signal NET2,NET3,NET4,NET5,NET6,NET7,NET8 :std_logic;
73
74        begin
75
76        shiftreg: shiftregister port map(
77                clk      => NoaaClk,
78                input    => NoaaData,
79                output => NET1);
80
81        comp : comparator port map(
82                input    => NET1(63 downto 4),
83                match    => NET2);
84
85        or2: ORgate port map(
86                a => NET5,
87                b => Reset,
88                c => NET7);
89
90        cnt: counter port map(
91                clk =>NoaaClk,
92                rst => NET7,
93                bit64 => NET3);
94
95        rstlatch: latchN port map(
96                clk => NoaaClk,
97                rst => '0',
98                ce => '1',
99                d => NET2,
100               q => NET5,
101               not_q => open);
102
103       outp: latch64 port map(
104               clk => NoaaClk,
105               ce => NET4,
106               d => NET1,
107               q => Output);
108
109       or1: ORgate port map(
110               a => NET2,
111               b => NET3,
112               c => NET4);
113
114       Wrlatch: latchN port map(
115               clk => NoaaClk,--NET4,
116               rst => NET8,
117               ce => NET4,
118               d => '1',
119               q => open,
120               not_q => WrReq);
121
122       INV1: INVgate port map(
123               a => WrDone,
124               b => NET8);
125
126       Busylatch: latchP port map(
127               clk => NET4,
128               rst => NET8,
129               d => '1',
130               q => Busy,
```

```
131             not_q => open);
132
133     Synclatch: latchP port map(
134             clk => NET2,
135             rst => '0',
136             d => NET6,
137             q => open,
138             not_q => NET6);
139
140     SyncLed <= NET6;
141     end RTL;
142
143     ----------------------------------------------------------------------------------
144     ----------------------------------------------------------------------------------
145
146
147     library IEEE;
148     use IEEE.STD_LOGIC_1164.ALL;
149     use IEEE.STD_LOGIC_ARITH.ALL;
150     use IEEE.STD_LOGIC_UNSIGNED.ALL;
151
152     entity shiftregister is
153        port( clk         : in  std_logic;
154              input    : in  std_logic;
155              output   : out std_logic_vector(63 downto 0));
156     end shiftregister;
157
158     architecture RTL of shiftregister is
159        signal reg: std_logic_vector(63 downto 0);
160     begin
161        process(clk)
162        begin
163           if clk'event and clk = '1' then
164              reg(63 downto 1) <= reg(62 downto 0);
165              reg(0) <=  not input;
166           end if;
167        end process;
168        output <= reg;
169     end RTL;
170
171     ----------------------------------------------------------------------------------
172     --- comparator. 60 bit comparator. When the six Noaa Frame sync words are found
173     --- output goes high.
174     ----------------------------------------------------------------------------------
175     library IEEE;
176     use IEEE.STD_LOGIC_1164.ALL;
177     use IEEE.STD_LOGIC_ARITH.ALL;
178     use IEEE.STD_LOGIC_UNSIGNED.ALL;
179
180     entity comparator is
181        port( input       : in  std_logic_vector(59 downto 0);
182              match       : out std_logic );
183     end comparator;
184
185     architecture RTL of comparator is
186        CONSTANT SYNC1    : std_logic_vector(9 downto 0):=  "1010000100"; --FramesyncWord 1
187        CONSTANT SYNC2    : std_logic_vector(9 downto 0):=  "0101101111"; --FramesyncWord 2
188        CONSTANT SYNC3    : std_logic_vector(9 downto 0):=  "1101011100"; --FramesyncWord 3
189        CONSTANT SYNC4    : std_logic_vector(9 downto 0):=  "0110011101"; --FramesyncWord 4
190        CONSTANT SYNC5    : std_logic_vector(9 downto 0):=  "1000001111"; --FramesyncWord 5
191        CONSTANT SYNC6    : std_logic_vector(9 downto 0):=  "0010010101"; --FramesyncWord 6
192     begin
193        process(input)
194        begin
195           if input = SYNC1&SYNC2&SYNC3&SYNC4&SYNC5&SYNC6 then
```

```
            match <= '1';
         else
            match <= '0';
         end if;
      end process;
   end RTL;

   ----------------------------------------------------------------------------
   --- latch64. 64 bit latch (trigged at negative edge of clock)
   ----------------------------------------------------------------------------
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
   use IEEE.STD_LOGIC_UNSIGNED.ALL;

   entity latch64 is
      port( clk   : in  std_logic;
            ce    : in std_logic;
            d     : in  std_logic_vector(63 downto 0);
            q     : out std_logic_vector(63 downto 0));
   end latch64;

   architecture RTL of latch64 is
   signal reg : std_logic_vector(63 downto 0);
   begin
      process(clk)
      begin
         if clk'event and clk = '0' and ce = '1' then
            reg <= d;
         end if;
      end process;
      q <= reg;
   end RTL;

   ----------------------------------------------------------------------------
   --- counter. at the 63th clock (rising edge) output goes high
   ----------------------------------------------------------------------------
   library IEEE;
   use IEEE.STD_LOGIC_1164.ALL;
   use IEEE.STD_LOGIC_ARITH.ALL;
   use IEEE.STD_LOGIC_UNSIGNED.ALL;

   entity counter is
      port( clk         : in  std_logic;
            rst         : in  std_logic;
            bit64       : out  std_logic);
   end counter;

   architecture RTL of counter is
      signal reg: std_logic_vector(5 downto 0);
   begin
      process(clk,rst)
      begin
         if rst = '1' then
            reg <= "000000";
         elsif clk'event and clk = '1' then
            reg <= reg + "000001";
         end if;
      end process;
      bit64 <= (reg(5) and reg(4)and reg(3) and reg(2) and reg(1) and reg(0));
   end RTL;

   ----------------------------------------------------------------------------
   --- END OF FILE: framesyncronizer.vhd
   ----------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
--- latchP. 1 bit latch (trigged at rising edge of clock)
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity latchP is
   port( clk      : in  std_logic;
         rst      : in  std_logic;
         d        : in  std_logic;
         q        : out std_logic;
         not_q    : out std_logic);
end latchP;

architecture RTL of latchP is
   signal reg: std_logic;
begin
   process(clk,rst)
   begin
      if rst = '1' then
         reg <= '0';
      elsif clk'event and clk = '1' then
         reg <= d;
      end if;
   end process;
   q <= reg;
   not_q <= not reg;
end RTL;

--------------------------------------------------------------------------------
--- ORgate. 2 input or gate
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity ORgate is
   port( a           : in  std_logic;
         b           : in std_logic;
         c           : out std_logic);
end ORgate;

architecture RTL of ORgate is
begin
   c <= a or b;
end RTL;

--------------------------------------------------------------------------------
--- INVgate. 1 input inverter
--------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity INVgate is
   port( a           : in  std_logic;
         b           : out std_logic);
end INVgate;

architecture RTL of INVgate is
begin
```

```
66          b <= not a;
67       end RTL;
68
69       ---------------------------------------------------------------------------------
70       --- latchN. 1 bit latch (trigged at falling edge of clock)
71       ---------------------------------------------------------------------------------
72       library IEEE;
73       use IEEE.STD_LOGIC_1164.ALL;
74       use IEEE.STD_LOGIC_ARITH.ALL;
75       use IEEE.STD_LOGIC_UNSIGNED.ALL;
76
77       entity latchN is
78          port( clk      : in  std_logic;
79                rst      : in  std_logic;
80                ce       : in std_logic;
81                d        : in  std_logic;
82                q        : out std_logic;
83                not_q    : out std_logic);
84       end latchN;
85
86       architecture RTL of latchN is
87          signal reg: std_logic;
88       begin
89          process(clk,rst,ce)
90          begin
91             if rst = '1' then
92                reg <= '0';
93             elsif clk'event and clk = '0'and ce = '1' then
94                reg <= d;
95             end if;
96          end process;
97          q <= reg;
98          not_q <= not reg;
99       end RTL;
100      -----
101
102
103      library IEEE;
104      use IEEE.STD_LOGIC_1164.ALL;
105      use IEEE.STD_LOGIC_ARITH.ALL;
106      use IEEE.STD_LOGIC_UNSIGNED.ALL;
107
108      entity comparator8 is
109         port( empty          : out  std_logic;
110               full           : out std_logic;
111               wradr       : in  std_logic_vector(22 downto 0);
112               rdadr    : in std_logic_vector(22 downto 0));
113      end comparator8;
114
115      architecture RTL of comparator8 is
116      begin
117         process(wradr,rdadr)
118         begin
119               if (wradr = rdadr) then
120                  empty <= '1';
121               else
122                  empty <= '0';
123               end if;
124
125               if (rdadr -1 = wradr) then
126                  full <= '1';
127               else
128                  full <= '0';
129               end if;
130         end process;
```

```
131         end RTL;
132         --------------------------------------------------------------------------------
133
134
135         library IEEE;
136         use IEEE.STD_LOGIC_1164.ALL;
137         use IEEE.STD_LOGIC_ARITH.ALL;
138         use IEEE.STD_LOGIC_UNSIGNED.ALL;
139
140         entity demux is
141           port (oe: in std_logic;
142            sel: in std_logic_vector(7 downto 0);
143            d: out std_logic_vector(8 downto 0));
144         end demux;
145
146         architecture rtl of demux is
147           signal t : std_logic_vector(8 downto 0);
148         begin
149         process(sel) begin
150            case sel is
151            when X"00" =>
152               t <= "000000001";
153            when X"01" =>
154               t <= "000000010";
155            when X"02" =>
156               t <= "000000100";
157            when X"03" =>
158               t <= "000001000";
159            when X"04" =>
160               t <= "000010000";
161            when X"05" =>
162               t <= "000100000";
163            when X"06" =>
164               t <= "001000000";
165            when X"07" =>
166               t <= "010000000";
167            when X"08" =>
168               t <= "100000000";
169            when others =>
170               t <= "000000000";
171            end case;
172         end process;
173           --below we activate output if 'oe' is low
174           d <= t when oe = '0' else "000000000";
175         end rtl;
176
177         --------------------------------------------------------------------------------
178
179
180
181
182
183         library IEEE;
184         use IEEE.STD_LOGIC_1164.ALL;
185         use IEEE.STD_LOGIC_ARITH.ALL;
186         use IEEE.STD_LOGIC_UNSIGNED.ALL;
187
188         entity bustrans8 is
189            port( oe          : in  std_logic;
190                  ce          : in  std_logic;
191                  input       : in  std_logic_vector(7 downto 0);
192                  output      : out std_logic_vector(7 downto 0));
193         end bustrans8;
194
195         architecture RTL of bustrans8 is
```

```
196         begin
197            process(oe,ce,input)
198            begin
199               if oe = '0' and ce = '1' then
200                  output <= input;
201               else output <= "ZZZZZZZZ";
202               end if;
203            end process;
204         --output <= input when oe = '0' and ce = '0' else "ZZZZZZZZ";
205
206         end RTL;
207
208
209
210         library IEEE;
211         use IEEE.STD_LOGIC_1164.ALL;
212         use IEEE.STD_LOGIC_ARITH.ALL;
213         use IEEE.STD_LOGIC_UNSIGNED.ALL;
214
215         entity latch8P is
216            port( clk         : in  std_logic;
217                  rst         : in  std_logic;
218                  ce          : in  std_logic;
219                  d           : in  std_logic_vector(7 downto 0);
220                  q           : out std_logic_vector(7 downto 0));
221         end latch8P;
222
223         architecture RTL of latch8P is
224         begin
225            process(clk,rst)
226            begin
227               if rst = '0' then
228                  q <= X"00";
229
230               elsif clk'event and clk = '1' and ce = '1' then
231                  q <= d;
232               end if;
233            end process;
234         end RTL;
235
236         --------------------------------------------------------------------------------
237         --------------------------------------------------------------------------------
238
239         --------------------------------------------------------------------------------
240         --------------------------------------------------------------------------------
241
242
243
244
245
246         library IEEE;
247         use IEEE.STD_LOGIC_1164.ALL;
248         use IEEE.STD_LOGIC_ARITH.ALL;
249         use IEEE.STD_LOGIC_UNSIGNED.ALL;
250
251         entity latch8N is
252            port( clk         : in  std_logic;
253                  rst         : in  std_logic;
254                  ce          : in  std_logic;
255                  oe          : in  std_logic;
256                  d           : in  std_logic_vector(7 downto 0);
257                  q           : out std_logic_vector(7 downto 0));
258         end latch8N;
259
260         architecture RTL of latch8N is
```

```
261         signal val: std_logic_vector(7 downto 0);
262         begin
263            process(clk,rst)
264            begin
265               if rst = '0' then
266                  val <= X"00";
267
268               elsif clk'event and clk = '0' and ce = '0' then
269                  val <= d;
270               end if;
271            end process;
272            q <= val when oe = '0' else "ZZZZZZZZ";
273         end RTL;
274
275         --------------------------------------------------------------------------------
276         --------------------------------------------------------------------------------
277
278         --------------------------------------------------------------------------------
279         ---------------------------
280
281
```

```
1         LIBRARY ieee;
2         USE ieee.std_logic_1164.ALL;
3         USE ieee.std_logic_unsigned.all;
4         USE ieee.numeric_std.ALL;
5
6         ENTITY tb1_vhd IS
7         END tb1_vhd;
8
9         ARCHITECTURE behavior OF tb1_vhd IS
10
11           -- Component Declaration for the Unit Under Test (UUT)
12           COMPONENT main
13           PORT(
14              SYSCLK : IN std_logic;
15              RST : IN std_logic;
16              PWRUP : IN std_logic;
17              ALE : IN std_logic;
18              RD : IN std_logic;
19              WR : IN std_logic;
20              PCS1 : IN std_logic;
21              AD : INOUT std_logic_vector(7 downto 0);
22              D : INOUT std_logic_vector(63 downto 0);
23              TEST : OUT std_logic;
24              LED : OUT std_logic;
25              CK0 : OUT std_logic;
26              CK1 : OUT std_logic;
27              RAS : OUT std_logic;
28              CAS : OUT std_logic;
29              WE : OUT std_logic;
30              BA : OUT std_logic_vector(1 downto 0);
31              DQMB : OUT std_logic_vector(2 downto 0);
32              S0 : OUT std_logic;
33              S1 : OUT std_logic;
34              A : OUT std_logic_vector(12 downto 0);
35              NOAA_CLK : IN std_logic;
36              NOAA_DATA : IN std_logic);
37           END COMPONENT;
38
39        COMPONENT hy57v28820hct
40        PORT(
41                 VDD1 : IN std_logic;
42                 VDDQ3 : IN std_logic;
43                 NC4 : IN std_logic;
44                 VSSQ6 : IN std_logic;
45                 NC7 : IN std_logic;
46                 VDDQ9 : IN std_logic;
47                 NC10 : IN std_logic;
48                 VSSQ12 : IN std_logic;
49                 NC13 : IN std_logic;
50                 VDD14 : IN std_logic;
51                 NC15 : IN std_logic;
52                 WEB : IN std_logic;
53                 CASB : IN std_logic;
54                 RASB : IN std_logic;
55                 CSB : IN std_logic;
56                 BA0 : IN std_logic;
57                 BA1 : IN std_logic;
58                 A10 : IN std_logic;
59                 A0 : IN std_logic;
60                 A1 : IN std_logic;
61                 A2 : IN std_logic;
62                 A3 : IN std_logic;
63                 VDD27 : IN std_logic;
64                 VSS28 : IN std_logic;
65                 A4 : IN std_logic;
```

```
66                A5 : IN std_logic;
67                A6 : IN std_logic;
68                A7 : IN std_logic;
69                A8 : IN std_logic;
70                A9 : IN std_logic;
71                A11 : IN std_logic;
72                NC36 : IN std_logic;
73                CKE : IN std_logic;
74                CLK : IN std_logic;
75                DQM : IN std_logic;
76                NC40 : IN std_logic;
77                VSS41 : IN std_logic;
78                NC42 : IN std_logic;
79                VDDQ43 : IN std_logic;
80                NC45 : IN std_logic;
81                VSSQ46 : IN std_logic;
82                NC48 : IN std_logic;
83                VDDQ49 : IN std_logic;
84                NC51 : IN std_logic;
85                VSSQ52 : IN std_logic;
86                VSS54 : IN std_logic;
87                DQ0 : INOUT std_logic;
88                DQ1 : INOUT std_logic;
89                DQ2 : INOUT std_logic;
90                DQ3 : INOUT std_logic;
91                DQ4 : INOUT std_logic;
92                DQ5 : INOUT std_logic;
93                DQ6 : INOUT std_logic;
94                DQ7 : INOUT std_logic);
95        END COMPONENT;
96           --Inputs
97           SIGNAL SYSCLK :  std_logic := '0';
98           SIGNAL RST :  std_logic := '1';
99           SIGNAL PWRUP :  std_logic := '0';
100          SIGNAL ALE :  std_logic := '0';
101          SIGNAL RD :  std_logic := '1';
102          SIGNAL WR :  std_logic := '1';
103          SIGNAL PCS1 :  std_logic := '1';
104          SIGNAL PCS5 :  std_logic := '0';
105          SIGNAL PCS6 :  std_logic := '0';
106          SIGNAL NOAA_CLK:std_logic:= '0';
107          SIGNAL NOAA_DATA:std_logic:= '0';
108          --BiDirs
109          SIGNAL AD :  std_logic_vector(7 downto 0):="ZZZZZZZZ";
110          SIGNAL D :  std_logic_vector(63 downto 0);
111
112          --Outputs
113          SIGNAL TEST :  std_logic;
114          SIGNAL LED :  std_logic;
115          SIGNAL CK0 :  std_logic;
116          SIGNAL CK1 :  std_logic;
117          SIGNAL RAS :  std_logic;
118          SIGNAL CAS :  std_logic;
119          SIGNAL WE :  std_logic;
120          SIGNAL BA :  std_logic_vector(1 downto 0);
121          SIGNAL DQMB :  std_logic_vector(2 downto 0);
122          SIGNAL S0 :  std_logic;
123          SIGNAL S1 :  std_logic;
124          SIGNAL A :  std_logic_vector(12 downto 0);
125
126
127          CONSTANT Tclock : time := 114 ns;
128          CONSTANT Tclk : time := 1503 ns;--Noaa Frequency is 665,4 kHz
129          --Stimuli defined as constants
130          CONSTANT FIRST : std_logic_vector(0 to 5):= "000000";
```

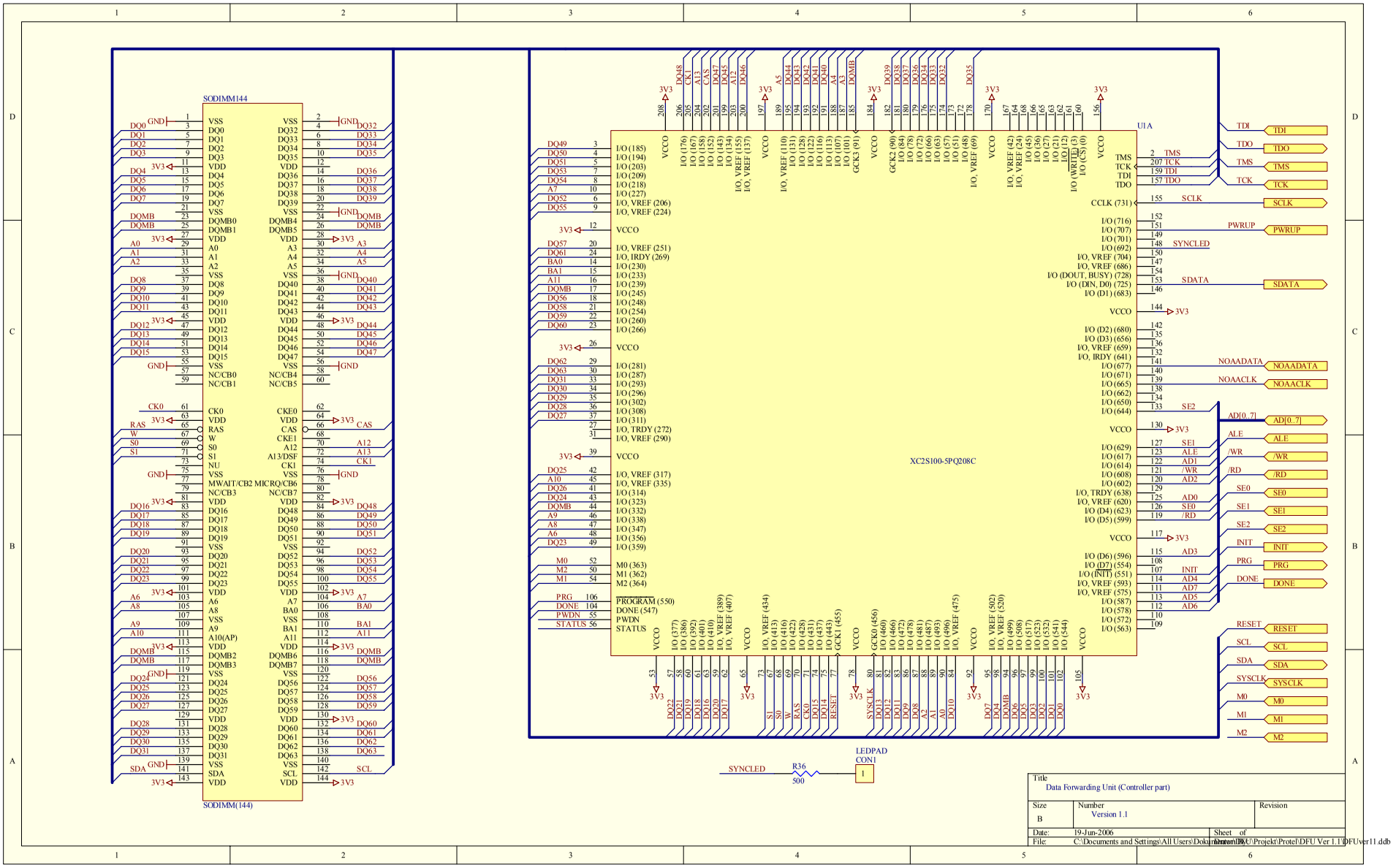
```
131           CONSTANT SYNC1 : std_logic_vector(0 to 9):=  "1010000100";  --FramesyncWord 1
132           CONSTANT SYNC2 : std_logic_vector(0 to 9):=  "0101101111";  --FramesyncWord 2
133           CONSTANT SYNC3 : std_logic_vector(0 to 9):=  "1101011100";  --FramesyncWord 3
134           CONSTANT SYNC4 : std_logic_vector(0 to 9):=  "0110011101";  --FramesyncWord 4
135           CONSTANT SYNC5 : std_logic_vector(0 to 9):=  "1000001111";  --FramesyncWord 5
136           CONSTANT SYNC6 : std_logic_vector(0 to 9):=  "0010010101";  --FramesyncWord 6
137           CONSTANT DATA  : std_logic_vector(0 to 63):= X"0102030405060708";
138           CONSTANT LAST  : std_logic_vector(0 to 3):= "1111";
139
140           SHARED VARIABLE clkCount : integer :=0;
141           CONSTANT Stimuli : std_logic_vector(0 to 133) :=FIRST&
142                                                           SYNC1&
143                                                           SYNC2&
144                                                           SYNC3&
145                                                           SYNC4&
146                                                           SYNC5&
147                                                           SYNC6&
148                                                           DATA&LAST;
149
150
151       BEGIN
152
153           -- Instantiate the Unit Under Test (UUT)
154           uut: main PORT MAP(
155              SYSCLK => SYSCLK,
156              RST => RST,
157              PWRUP => PWRUP,
158              ALE => ALE,
159              RD => RD,
160              WR => WR,
161              AD => AD,
162              PCS1 => PCS1,
163              TEST => TEST,
164              LED => LED,
165              CK0 => CK0,
166              CK1 => CK1,
167              RAS => RAS,
168              CAS => CAS,
169              WE => WE,
170              BA => BA,
171              DQMB => DQMB,
172              S0 => S0,
173              S1 => S1,
174              D => D,
175              A => A,
176              NOAA_CLK => NOAA_CLK,
177              NOAA_DATA => NOAA_DATA);
178
179       RAM: hy57v28820hct PORT MAP(
180                VDD1 => '1',
181                DQ0 => D(0),
182                VDDQ3 => '1',
183                NC4 => 'L',
184                DQ1 => d(1),
185                VSSQ6 => '0',
186                NC7 => 'L',
187                DQ2 => D(2),
188                VDDQ9 => '1',
189                NC10 => 'L',
190                DQ3 => D(3),
191                VSSQ12 => '0',
192                NC13 => 'L',
193                VDD14 => '1',
194                NC15 => 'L',
195                WEB => WE,
```

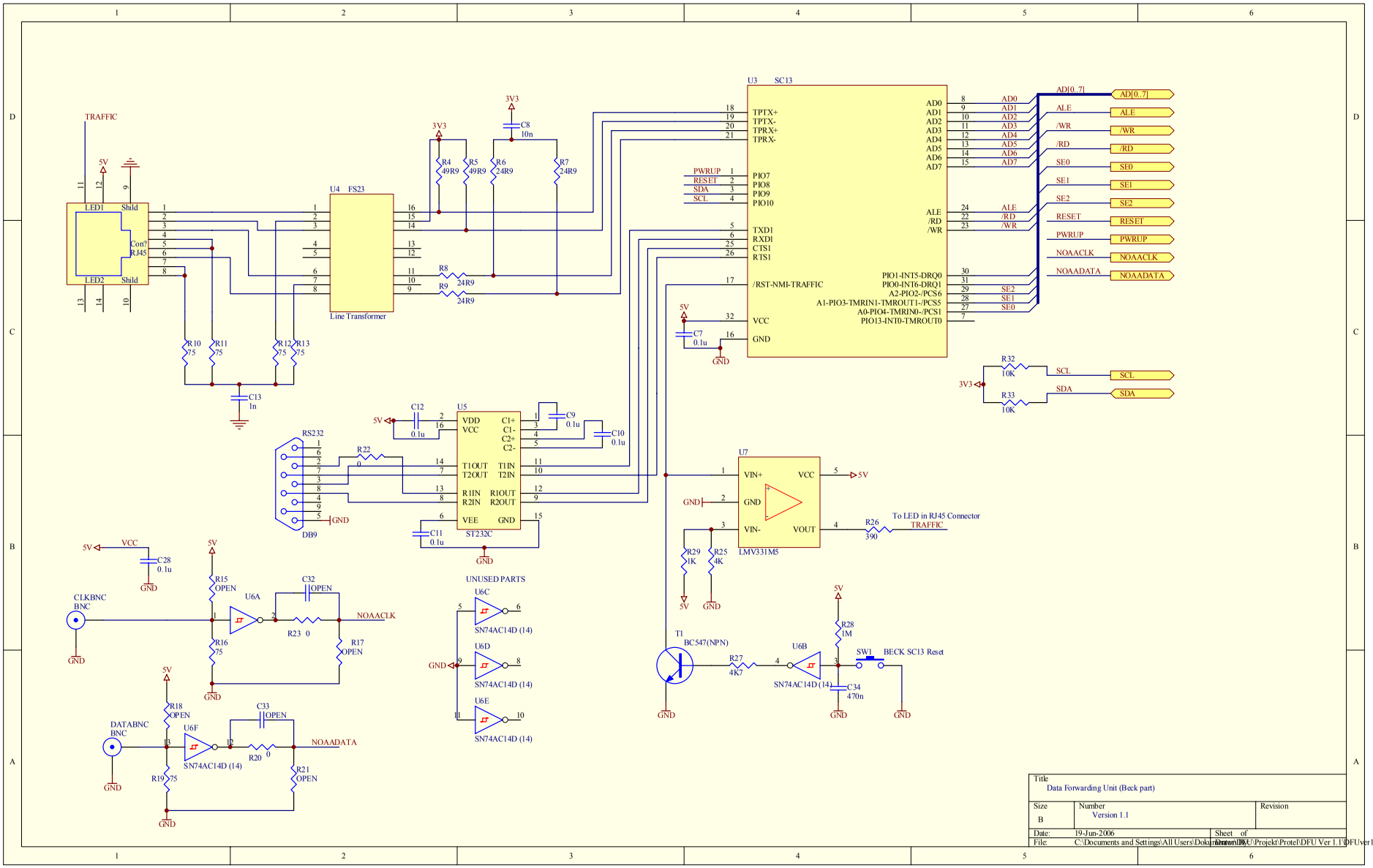
```
196                CASB => CAS,
197                RASB => RAS,
198                CSB => '0',
199                BA0 => BA(0),
200                BA1 => BA(1),
201                A10 => A(10),
202                A0 => A(0),
203                A1 => A(1),
204                A2 => A(2),
205                A3 => A(3),
206                VDD27 => '1',
207                VSS28 => '0',
208                A4 => A(4),
209                A5 => A(5),
210                A6 => A(6),
211                A7 => A(7),
212                A8 => A(8),
213                A9 => A(9),
214                A11 => A(11),
215                NC36 => 'L',
216                CKE => '1',
217                CLK => SYSCLK,
218                DQM => DQMB(0),
219                NC40 => 'L',
220                VSS41 => '0',
221                NC42 => 'L',
222                VDDQ43 => '1',
223                DQ4 => D(4),
224                NC45 => 'L',
225                VSSQ46 => '0',
226                DQ5 => D(5),
227                NC48 => 'L',
228                VDDQ49 => '1',
229                DQ6 => D(6),
230                NC51 => 'L',
231                VSSQ52 => '0',
232                DQ7 => D(7),
233                VSS54 => '0');
234          CLK : PROCESS
235          BEGIN
236             SYSCLK <= '1';
237             WAIT FOR 31 NS;
238             SYSCLK <= '0';
239             WAIT FOR 31 NS;
240          END PROCESS;
241
242          NoaaClock : PROCESS
243          BEGIN
244             --Place Noaa Data
245             if ClkCount < 134 then
246                NOAA_DATA <= not Stimuli(ClkCount);
247             else
248                NOAA_DATA <= '0';
249             end if;
250             wait for Tclk/4;
251             NOAA_CLK <= '1';
252             wait for Tclk/2;
253             NOAA_CLK <= '0';
254             wait for Tclk /4;
255
256             ClkCount := ClkCount + 1;
257          END PROCESS;
258
259
260          tb : PROCESS
```

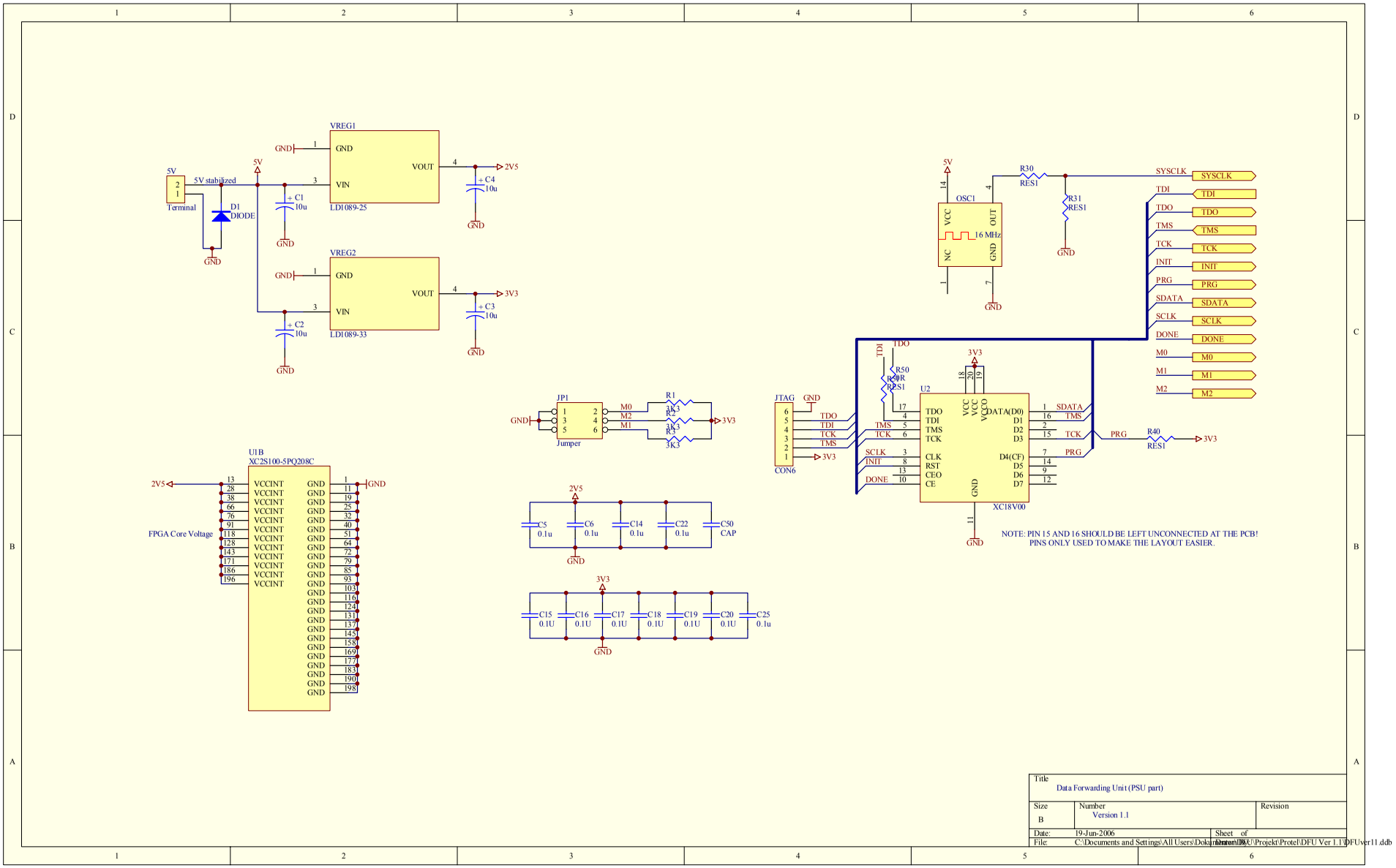
```
261         BEGIN
262
263            -- Wait 100 ns for global reset to finish
264            IF NOW > 0 ns  AND NOW < 201 NS THEN
265               RST <= '0';
266            ELSE
267               RST <= '1';
268            END IF;
269
270            --ACTIVATE POWER UP PULSE
271            IF NOW > 2 US AND NOW < 4 US THEN
272               PWRUP <= '1';
273            ELSE
274               PWRUP <= '0';
275            END IF;
276
277            if now = 90 us then
278            --BECK READ CYCLE
279               PCS1 <= '0';
280               WAIT FOR 5 NS;
281               ALE <= '1';
282               AD <= X"08";
283               WAIT FOR 20 NS;
284               ALE <= '0';
285
286               WAIT FOR 10 NS;
287               AD <= "ZZZZZZZZ";
288               RD <= '0';
289               WAIT FOR 900 NS;
290               RD <= '1';
291               WAIT FOR 20 NS;
292               PCS1 <= '1';
293            end if;
294
295            if now = 100 us then
296            --BECK READ CYCLE
297               PCS1 <= '0';
298               WAIT FOR 5 NS;
299               ALE <= '1';
300               AD <= X"08";
301               WAIT FOR 20 NS;
302               ALE <= '0';
303
304               WAIT FOR 10 NS;
305               AD <= "ZZZZZZZZ";
306               RD <= '0';
307               WAIT FOR 900 NS;
308               RD <= '1';
309               WAIT FOR 20 NS;
310               PCS1 <= '1';
311            end if;
312
313               IF NOW = 110 US THEN
314            --BECK WRITE CYCLE
315               PCS1 <= '0';
316               WAIT FOR 5 NS;
317               ALE <= '1';
318               AD <= X"02";
319               WAIT FOR 20 NS;
320               ALE <= '0';
321
322               WAIT FOR 10 NS;
323               AD <= X"01";--increment RdPointer
324               WR <= '0';
325               WAIT FOR 900 NS;
```

```
326                WR <= '1';
327                WAIT FOR 20 NS;
328                PCS1 <= '1';
329                AD <= "ZZZZZZZZ";
330             END IF;
331
332             if now = 112 us then
333             --BECK READ CYCLE
334                PCS1 <= '0';
335                WAIT FOR 5 NS;
336                ALE <= '1';
337                AD <= X"00";
338                WAIT FOR 20 NS;
339                ALE <= '0';
340
341                WAIT FOR 10 NS;
342                AD <= "ZZZZZZZZ";
343                RD <= '0';
344                WAIT FOR 900 NS;
345                RD <= '1';
346                WAIT FOR 20 NS;
347                PCS1 <= '1';
348             end if;
349
350             WAIT FOR 5 NS;
351
352       -- STOP TEST IF 1 US REACHED
353             IF NOW > 700 US THEN
354                wait; -- will wait forever
355             END IF;
356
357          END PROCESS;
358
359       END;
360
```

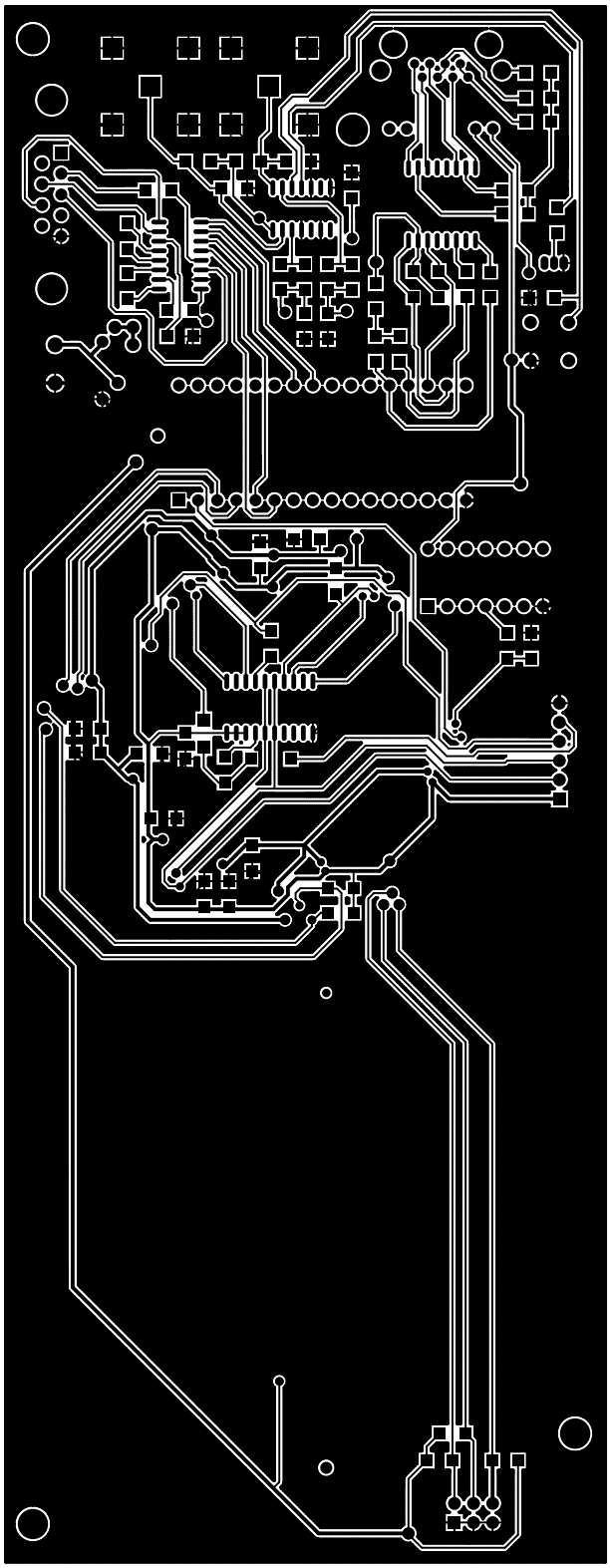
13.3 Diagrammer og PCB'er


SODIMM144
SODIMM(144)
U1A
XC2S100-5PQ208C
LEDPAD
CON1
SYNCLED
R36
500
Title
Data Forwarding Unit (Controller part)
Size
B
Number
Version 1.1
Revision
Date: 19-Jun-2006
Sheet of
File: C:\Documents and Settings\All Users\Dokumente\DFU\Projekt\Protel\DFU Ver 1.1\DFUver11.ddb


TRAFFIC
LED1
LED2
Shild
Con?
RJ45
U4 FS23
Line Transformer
U3 SC13
TPTX+
TPTX-
TPRX+
TPRX-
PIO7
PIO8
PIO9
PIO10
TXD1
RXD1
CTS1
RTS1
/RST-NMI-TRAFFIC
VCC
GND
AD0
AD1
AD2
AD3
AD4
AD5
AD6
AD7
ALE
/RD
/WR
PIO1-INT5-DRQ0
PIO0-INT6-DRQ1
A2-PIO2-/PCS6
A1-PIO3-TMRIN1-TMROUT1-/PCS5
A0-PIO4-TMRIN0-/PCS1
PIO13-INT0-TMROUT0
AD[0..7]
SE0
SE1
SE2
RESET
PWRUP
NOAACLK
NOAADATA
SCL
SDA
U5
ST232C
VDD
VCC
VEE
GND
C1+
C1-
C2+
C2-
T1OUT
T2OUT
R1IN
R2IN
T1IN
T2IN
R1OUT
R2OUT
RS232
DB9
U7
LMV331M5
VIN+
VIN-
VOUT
To LED in RJ45 Connector
TRAFFIC
CLKBNC
DATABNC
BNC
U6A
U6B
U6C
U6D
U6E
U6F
SN74AC14D (14)
UNUSED PARTS
T1
BC547(NPN)
SW1
BECK SC13 Reset
R4 49R9
R5 49R9
R6 24R9
R7 24R9
R8 24R9
R9 24R9
R10 75
R11 75
R12 75
R13 75
R15 OPEN
R16 75
R17 OPEN
R18 OPEN
R19 75
R20 0
R21 OPEN
R22 0
R23 0
R25 4K
R26 390
R27 4K7
R28 1M
R29 1K
R32 10K
R33 10K
C7 0.1u
C8 10n
C9 0.1u
C10 0.1u
C11 0.1u
C12 0.1u
C13 1n
C28 0.1u
C32 OPEN
C33 OPEN
C34 470n
3V3
5V
Title
Data Forwarding Unit (Beck part)
Size
B
Number
Version 1.1
Revision
Date: 19-Jun-2006
Sheet of
File: C:\Documents and Settings\All Users\Dokumente\DFU\Projekt\Protel\DFU Ver 1.1\DFUver11.ddb



Title: Data Forwarding Unit (PSU part)		
Size: B	Number: Version 1.1	Revision:
Date: 19-Jun-2006	Sheet of	
File: C:\Documents and Settings\All Users\Dokumente\DFU\Projekt\Protel\DFU Ver 1.1\DFUver11.ddb		



DATA FORWARDING UNIT Ver. 1.1
EB: 8 SSC
MODE
GND
TDO
TMS
TCK
TMS
VCC
RST
CLK
DATA

13.4 C-program til SC13

```
1   /*****************************************************************************
2   * Includes
3   *****************************************************************************/
4   #include <stdio.h>
5   #include <string.h>
6   #include "clib.h"
7   #include <dos.h>
8
9   /*****************************************************************************
10  * Constants
11  *****************************************************************************/
12  #define TM_PORT_ECHO                    6796
13  #define CLIENT_PACKETSIZE               3000   //1400 is maximum in DPU
14  #define HEADER_LENGTH                   5
15  #define TM_TCPECHOBUF_CLIENT_RECVSIZE   20
16  #define PRINT_DATA
17
18  #define RESET_PIN  8
19  #define PWRUP_PIN  7
20  #define LOW  5
21  #define HIGH  4
22
23
24  /*****************************************************************************
25  * Global variables
26  *****************************************************************************/
27  unsigned char sendbuf[CLIENT_PACKETSIZE  + HEADER_LENGTH ];
28  char recvbuf[TM_TCPECHOBUF_CLIENT_RECVSIZE   ];
29  char destIPStr[17];
30  unsigned long destIPAddr;
31  struct sockaddr_in  addr;
32     int sd;
33     int i;
34     int error;
35     int retval;
36
37
38  long int expFrameSyncIndex  = 1;
39  long int errorcount=0;
40  unsigned int BytesInBuffer=0;
41  unsigned char fromRam[9];
42  long int last=0;
43  int N; //loop variable
44  unsigned char type;
45  /*****************************************************************************
46  * Prototyping
47  *****************************************************************************/
48  int ConnectDPU();
49  int SendData( char far * bufptr, unsigned int bufLen,unsigned char typeOfData) ;
50
51
52
53  /*****************************************************************************
54  * Main
55  *****************************************************************************/
56  int main(int argc, char *argv[])
57  {
58  //int M;
59
60
61     BIOS_Set_Focus (FOCUS_APPLICATION );
62  //below we reset read and write pointer
63     pfe_enable_pio (RESET_PIN,HIGH);
64     pfe_enable_pio (PWRUP_PIN,LOW);
65     pfe_enable_bus (0xFFFF,1);//ENABLE ALE
66     pfe_enable_pcs (1);//ENABLE PCS 1
67
68
69
70  //convert IP address from argument to proper format
71     strcpy(destIPStr,"1.1.1.1");
72     if(argc==2)
73     {
```

```
74          if(strlen(argv[1]) < 17)
75          strcpy(destIPStr,argv[1]);
76       }
77       else
78       {
79         printf("\r\nMissing or invalid argument\r\n"  );
80       }
81       printf("\r\nDFU:  Dest. IP: %s ,  Port %d\r\n"  ,destIPStr,TM_PORT_ECHO );
82
83
84       //Create out ID-string
85
86       //Send ID-string (length = 9, type = 0x02)
87    //   SendData((char *) sendbuf ,9,0x02);
88
89
90       ConnectDPU ();
91    //if ( (unsigned char) inportb ( 0x108) !=1)
92       SendData("0030556802B7"  ,12,0x02);
93
94       //outportb(0x102,2);//increment Read Pointer
95       // to match testgenerator delay
96
97       for(;;)
98       {
99          if ( (unsigned char) inportb ( 0x108) !=1)
100         {
101            for(;;)
102            {
103               if ( (unsigned char) (inportb ( 0x108)& 0x04) !=4)
104               {//Not Busy! - we can read from RAM!
105                  fromRam[0]=(unsigned char) inportb(0x100);//,0xFFFF,0x0000);
106                  fromRam[1]=(unsigned char) inportb(0x101);//,0xFFFF,0x0000);
107                  fromRam[2]=(unsigned char) inportb(0x102);//,0xFFFF,0x0000);
108                  fromRam[3]=(unsigned char) inportb(0x103);//,0xFFFF,0x0000);
109                  fromRam[4]=(unsigned char) inportb(0x104);//,0xFFFF,0x0000);
110                  fromRam[5]=(unsigned char) inportb(0x105);//,0xFFFF,0x0000);
111                  fromRam[6]=(unsigned char) inportb(0x106);//,0xFFFF,0x0000);
112                  fromRam[7]=(unsigned char) inportb(0x107);//,0xFFFF,0x0000);
113                  outportb(0x102,2);//increment Read Pointer
114                  if(
115                     ((fromRam[0] & 0xF0) == 0x50) &&
116                     (fromRam[1] == 0xC9) &&
117                     (fromRam[2] == 0x83) &&
118                     (fromRam[3] == 0x9D) &&
119                     (fromRam[4] == 0x71) &&
120                     (fromRam[5] == 0xFD) &&
121                     (fromRam[6] == 0x16) &&
122                     (fromRam[7] == 0xA1))
123                  {
124                  //printf("\n\rframesync found!");
125                  SendData(( char *) sendbuf ,BytesInBuffer-1,0x05);
126                  BytesInBuffer  = 0;
127                  for(N=7;N>=0;N--) sendbuf[BytesInBuffer++] = fromRam[N];
128                  SendData((  char *) sendbuf ,BytesInBuffer,0x10);
129                  BytesInBuffer  = 0;
130
131                  }
132                  else
133                  {
134                     //type = 0x05;
135                     for(N=7;N>=0;N--) sendbuf[BytesInBuffer++] = fromRam[N];
136                     if (BytesInBuffer  > (CLIENT_PACKETSIZE -12))
137                     {
138                        SendData(( char *) sendbuf ,BytesInBuffer,0x05);
139                        BytesInBuffer  = 0;
140                        //break;
141                        //printf("\r\nBytes in byffer > client_sendsize!");
142                     }
143                  }
144               }//end if not busy
145               else
146               {
```

```
147                 //printf("\n\rBusy!");
148              }
149              break;
150           }//end inner for(;;)
151
152           //outportb(0x102,2);//increment Read Pointer
153
154        }//end if DataInRAM
155        else
156        {
157          printf ("\r\nRam empty" );
158          api_sleep (1000);
159          SendData ("0000" ,2,0x20);
160          break;
161        }
162
163
164     }
165
166
167
168
169
170
171
172
173   //  SendData((char *) "hejsa fra Beck" ,13,0x05);
174
175
176     if(retval == API_ERROR )
177     {
178        printf ("\r\nTCPClient: Senderror %d" ,error);
179        goto TCP_CLIENT_DONE ;
180     }
181     api_sleep (100);
182     //end while(1)
183
184  //   SendData((char *) sendbuf ,9,36);
185  //   printf("\r\nTerminating client\r\n");
186
187  TCP_CLIENT_DONE :
188     printf ("\r\nClosing connection\r\n" );
189     retval = closesocket ( sd, &error );
190     if(retval==API_ERROR )
191     {
192         printf ("\r\nTCPClient: Socket close failed: %d" ,error);
193     }
194
195
196     BIOS_Set_Focus ( FOCUS_BOTH );
197     return 0;
198  }
199
200  /*****************************************************************************/
201  //********** END OF MAIN ******************************************************
202  /*****************************************************************************/
203
204  int DataInRam ()
205  {
206     if( (unsigned char) inportb (0x108) ==1) return 0; //RAM empty
207     else return 1; //data in RAM!
208  }
209
210
211
212
213  int ConnectDPU ()      ////////////////////////////////////////////////// ConnectDPU ////
214  {
215
216
217     //API call open socket
218     retval = opensocket ( SOCK_STREAM , &error );
219     if(retval == API_ERROR )
```

```
220     {
221        printf("\r\nTCPClient: Socket open failed: %d"  , error);
222        //goto TCP_CLIENT_DONE;
223     }
224     else sd = retval;
225
226     //convert server  port to correct byte order
227     addr.sin_family =  PF_INET;
228     retval = htons( TM_PORT_ECHO );
229     addr.sin_port =  retval;
230     //convert DestIPString
231     retval = inet_addr( (char *)destIPStr, &destIPAddr );
232
233     if(retval == API_ERROR) ;  //goto TCP_CLIENT_DONE;
234     else addr.sin_addr.s_addr =  destIPAddr;
235
236     //Connect to Socket
237     retval = connect( sd, (const struct sockaddr *)&addr, &error );
238
239     if(retval == API_ERROR)
240     {
241       if(error==261)  //connection refused from server host
242       {
243        printf("\r\nTCPClient: Connection refused from %s"  ,destIPStr);
244       }
245       else
246       {
247          printf("\r\nTCPClient:Socket connect failed: %d"  ,error);
248       }
249        //goto TCP_CLIENT_DONE;
250     }
251
252     printf("\r\nTCPClient connected with %s\r\n"  ,destIPStr);
253
254
255     return 0;
256  } // End of ConnectDPU ////////////////////////////////////////////////////////////////////
   //////////
257
258
259
260
261
262
263  int SendData(char far * bufptr, unsigned int bufLen, unsigned char typeOfData)
264  {
265     unsigned char tmpBuf[CLIENT_PACKETSIZE  + HEADER_LENGTH];
266     unsigned int m;
267
268     for(m=0;m<bufLen;m++)
269     {
270        tmpBuf[m+HEADER_LENGTH] = bufptr[m];
271     }
272
273    // bufptr[bufLen-1] = 's';  //byte nr. 100
274   // bufptr[bufLen] = 't';    //101
275  //
276   // tmpBuf[5]=(char) bufLen;
277     tmpBuf[0] = (unsigned char) typeOfData;
278     tmpBuf[1] = 0;
279     tmpBuf[2] = 255;
280     tmpBuf[3] = (unsigned char) ((bufLen & 0xFF00) >> 8);
281     tmpBuf[4] = (unsigned char) bufLen;
282
283   //  if (tmpBuf[3] != 1) printf(" tmpbuf[3] %u",tmpBuf[3]);
284   //  if (tmpBuf[4] != 64) printf(" tmpbuf[4] %u",tmpBuf[4]);
285    //printf(" tmpbuf[4] %d",tmpBuf[4]);
286
287     retval = send( sd,( char *) tmpBuf,bufLen + HEADER_LENGTH,0,&error );
288     if(retval == API_ERROR)
289     {
290        printf("\r\nTCPClient: error from SendData %d"  ,error);
291        return 1;
```

```
292     }
293   return 0;
294  }
295
296
297  // End of file
```

13.5 Kildekode til DPU

```
/*
-------------------------------------------------------------------------------
-- Master Thesis at DTU 2006                    --
--                          --
-- Project Title:                       --
-- Handling Image Data from NOAA Weather Satellites          --
--                          --
-- Authors:                         --
-- Eskil Binzer                             --
-- Soeren Soelling Christiansen                     --
-------------------------------------------------------------------------------
-- File Description:                        --
--                          --
-- Recieval of 10bit NOAA-data and splitting data into 5 8bit files.    --
-- Receiving port is 6789               --
-- Only a test program - thus no exception handling          --
-- Compiled with g++                    --
--                          --
-- Version:                         --
-- 6.6                              --
-- Date:                            --
-- 09-08-2006                           --
-------------------------------------------------------------------------------
*/

/*****************************************************************************
DPU i denne version aabner en modtagelses-socket og venter paa at blive
kontaktet af en klient.
Det er et enkelttraadet program, saa det kan ikke klare flere forbindelser af
gangen
De foerste 5 byte af data bliver laest som DFU-header. Foerste byte er typen
og den kan have 4 vaerdier. DFU_ID, DFU_DATASYNC, DFU_DATA, DFU_DATAEND.

Foerste pakke efter en sockel er blevet oprettet skal altid vaere DFU_ID -
ellers lukker programmet socklen og lytter igen paa ny forbindelse
Hvis der ikke allerede findes en Pending fil fra samme DFU_ID saa skal der
startes en ny fil, og det goeres kun ved en DFU_DATASYNC
Efter modtagelse af en DATAEND lukkes socklen, og pendingfilen renames saa
den ikke laengere er pending, og databehandlingsprogrammerne kan gaa igang.

Saa snart data i en DFU_END pakke er blevet skrevet til output filen, renames
denne til ikke laengere at vaere pending, og funktionen separate kaldes
Seperate skaerer de 2 LSB i hver ord fra, og fordeler data fra de fem kanaler
ud paa 5 forskellige filer Chx.dat
*****************************************************************************/

#include <fstream>
#include <arpa/inet.h>
#include <netdb.h>
#include <netinet/in.h>
#include <unistd.h>
#include <stdio.h>
#include "lib.h"
#include <stdlib.h>

#define OUTPUT_DIR "/home/radman/NOAAFiles/27jun"
#define MAX_MSG 100
#define PAKKE_ARRAY_SIZE (MAX_MSG+1)
#define TRUE 1
#define FALSE 0
#define BYTE unsigned char
#define DFU_DATA 0x05
#define DFU_DATASYNC 0x10
#define DFU_DATAEND 0x20
#define DFU_ID 0x02
#define DFU_HEADERLNG 5
#define welcomePort 6788
#define TESTMODE 0
//-------------------------------------------------------
// For map:
#include <stdlib.h>
#include <string.h>
#include  "math.h"
```

```
74
75  #define J2  0.00108248
76  #define MY 11467.874226
77  #define JORDRADIUS 6378.145
78  #define EOL printf("\n")
79  #define midy 300
80  #define midx 400
81  #define j3oj2 -2.3667873E-03
82  #define mv 398601.2
83  //in out test-files AUXSYNC damaged, and 3 byte shorter
84  //#define AUXSYNC_LNTH 122
85  #define AUXSYNC_LNTH 125
86  //egentlig er AUXSYNC 125 byte lang, men foerste halve byte laeses med i data, saa sidste halv
    e byte vil vaere padding som DFUen har lagt til for at naeste NOAAramme kan starte paa starten
     af en byte
87  //-------------------------------------------------------------
88   struct banel //Struct that contains sattelite elements
89   {
90     double tstart,epo,dr,ink,cosi,sini,knu,exc,xi,per,mid,rot,axe,dk,dp,wm;
91     char navn[16];
92   };
93
94   struct kort   //Struct that describes the part of the worldmap which
95   //the user has requested
96   {
97     float r,ycntr,latlimit,midlong;
98   };
99
100  //Struct that contains the variables that are read in the ini-file. Such as
101  //directories, picture width and height and the map-area covered by the quickview picture:
102  struct ini_info
103  {
104    int tmidx,tmidy,port;
105    double tlmid,tbmid,tbtop;
106    char outdir[100],helpdir[100],satdir[100];
107    BYTE DFUID[12];
108  };
109
110  static double GR=180.0/M_PI, TOPI=2.0*M_PI, retvinkel=M_PI/2.0; // global !
111
112  int make_label(BYTE labelArray[50][600], struct ini_info *ini,char *labelstring);
113  int separate(char inputfile[200],struct banel *Pb);
114  int map(int argc, char argv[],char inputfile[200],struct banel *Pb, struct ini_info *ini);
115  int get_ini(struct ini_info *ini);
116  int get_elements(struct banel *Pb, int scadr,struct ini_info *ini); // Function that opens
117  //the "weather"file, and fills the stuct Pb with element-data
118  void insert_space(char* l, int nr); // inserts space in text string,l, at position, nr.
119  long filesize(FILE *stream);
120  void sgpsat(float line, float pixel, float *longi, float *lati,
121    struct banel *Pb);
122
123  // Function that finds the longitudes and lattitudes that belongs to a given
124  // x,y on the projection
125  void vduxy2lonlat(int elem, int scan, float *longi, float *lati,
126    struct kort *Pg);
127  void lonlat2vduxy(float longi, float lati, int *elem, int *scan, struct kort *Pg);
128  void MinMax(int a, int b, int c, int d, int *minimum, int *maximum);
129  int interpoler(int i0, int j0, float bp,float lp,float grid[18][46][2],
130            float *pixelnummer, float *linjenummer);
131
132  int indenfor(float bp, float lp, float p[4][2]);
133  int seek(float la, float lo, float pt[19][46][2], int m ,int *p, int *l);
134  int drawkonturs(BYTE skabelon[midy*2][midx*2], struct kort *Pg, struct ini_info *ini);
135  int synccheck(BYTE Auxsync[AUXSYNC_LNTH]);
136  //int synccheck(void);
137
138 int main()
139 {
140    struct banel orbital_elements,*Pb;
141    struct ini_info info_konstants, *ini;
142    //int separate(FILE *raw10bitfile);
143    struct sockaddr_in serverAddress, clientAddress;
144    int welcomeSocket, connectionSocket, i,readint,pakkenr,pakkelng,status,aar,maan,dag,tim,min
```

```
   ,scadr,yearday,ms,k,pakkeindex,modtagby,wantby;
145    socklen_t  clientAddressLength ,framecounter ;
146    FILE *raw10bitfile , *test,*fpind;
147    BYTE header[DFU_HEADERLNG ],pakke[3100],hjlppakke[3100],type,DFUID[12],FileIsOpen ;
148    char  filename[200],donefilename[200],teststring[10],yeardaystr[3],receivetime[10],maparg[3
   ];
149    div_t divresult ;
150    double time;
151    float sek;
152
153   FileIsOpen  = FALSE;
154   Pb=&orbital_elements ;
155   ini=&info_konstants ;
156
157   if (get_ini(ini) != TRUE)
158   {
159    printf("Get_initial error\n" );
160    return(TRUE);
161   }
162
163  /*************************************************************************
164             VELKOMMENSOCKET!
165  *************************************************************************/
166  //lav en adgangssocket:
167    welcomeSocket  = socket(AF_INET, SOCK_STREAM,0);
168    if (welcomeSocket  < 0) {
169       printf("cannot create listen socket"  );
170    }
171
172
173  //bind adgangsocklen til port 6789, brug INET FORMAT (port, ip) og accepter
174  //alle ip-adresser
175    serverAddress.sin_family  = AF_INET;
176    serverAddress.sin_addr.s_addr  = htonl(INADDR_ANY);
177    serverAddress.sin_port  = htons(ini->port);
178
179    if(bind(welcomeSocket,(struct sockaddr *) &serverAddress,sizeof(serverAddress) )<0)
180    {
181       printf("cannot bind socket i%" ,welcomeSocket );
182       exit(1);
183    }
184
185  //-------------------------------------------------------------------------
186  //       CONNECTIONSOCKET!
187    //Nu er vi klar til at modtage en opringning:
188  //-------------------------------------------------------------------------
189    while(1)
190    {
191          listen(welcomeSocket,5);
192          printf("Waiting for TCP connection on port %i ...\n"  ,ini->port);
193          clientAddressLength  = sizeof(clientAddress);
194          connectionSocket  = accept(welcomeSocket, (struct sockaddr *) &clientAddress, &clientA
   ddressLength);
195
196          if(connectionSocket  < 0 )
197          {
198             printf("cannot accept connection" );
199             exit(1);
200          }
201          printf(" connected to %i" ,clientAddress.sin_addr);
202          printf(":%i\n",clientAddress.sin_port);
203          memset (pakke,0x0, PAKKE_ARRAY_SIZE );
204          memset (header,0x0, 7);
205
206  //-------------------------------------------------------------------------
207  //    Nu har vi oprettet forbindelse og skal til at li¿½e hvad det er vi modtager
208  //-------------------------------------------------------------------------
209
210  //-------------------------------------------------------------------------
211  //    IDENTIFICER PASSAGE
212  // Er det foerste pakke? - hvis det er, hvad hedder den gamle fil si¿½
213  // Hvis det ikke er si¿½lav en ny fil
214  //-------------------------------------------------------------------------
```

```
215
216     readint  = recv(connectionSocket , header, DFU_HEADERLNG ,0);
217     type  = header[0];
218     pakkenr  = ((header[1]*256)+header[2]);
219     pakkelng  = header[3]*256  + header[4];
220     if (type  == DFU_ID)
221     {
222        readint  = recv(connectionSocket , DFUID, pakkelng ,0);
223
224        if(strncmp((char*)DFUID,(char*)ini->DFUID,12)==0) //check om DFUID er korrekt!
225        {
226        strcpy(filename ,ini->outdir);
227        (void) strncat(filename ,(char  *) ini->DFUID,12);
228        (void) strncat(filename ,"PendingNOAA.dat" ,15);
229        printf(filename );
230        framecounter =0;
231              while (readint >0) // bliv ved saaï¿½laenge der er mere at laese
232              {
233           readint  = recv(connectionSocket , header, DFU_HEADERLNG ,0);
234           type  = header[0];
235           pakkenr  = ((header[1]*256)+header[2]);
236           pakkelng  = header[3]*256  + header[4];
237                    //printf("ID: %i %i,",type,pakkelng);
238
239           pakkeindex =0;
240           wantby=pakkelng ;
241                 modtagby  = recv(connectionSocket , pakke, wantby,0);
242           while(modtagby <wantby){
243              pakkeindex =pakkeindex +modtagby ;
244              wantby=wantby-modtagby ;
245              modtagby  = recv(connectionSocket , hjlppakke, wantby,0);
246              for(k=0;k<modtagby ;k++)
247              {
248                 pakke[pakkeindex +k]=hjlppakke[k];
249              }
250           }
251
252           if (type  == DFU_DATASYNC )
253           {
254              framecounter ++;
255              //printf("DATASYNC%i ",framecounter);
256              if (!FileIsOpen )
257              {
258                 if ((raw10bitfile =fopen(filename ,"ab"))== NULL)
259                 {
260                    printf("Unable to open datafile" );
261                    return (TRUE);
262                 }
263                 else
264                 {
265                    FileIsOpen  = TRUE;
266                    printf("File opened " );
267                 }
268              }
269                    fwrite(pakke,pakkelng ,1,raw10bitfile );
270           }
271
272
273           if (type  == DFU_DATA )
274           {
275
276              if(FileIsOpen ) fwrite(pakke,pakkelng ,1,raw10bitfile );
277              //Hvis output file allerede er ï¿½en skal data bare skrives til den
278              else //Hvis output ikke er ï¿½en skal det checkes om filen eksisterer
279              {
280                 //raw10bitfile=fopen(filename,"rb");
281
282                 if ((raw10bitfile =fopen(filename ,"rb")) > 0)//ï¿½aaben til laesning for at chec
    ke om filen eksisterer
283                 {
284                    FileIsOpen  = TRUE;
285                    printf("Filen eksisterer " );
286                    if(FileIsOpen ) {//selvflgelig er den aaben! - men der checkes alligevel
```

```
287                     fclose(raw10bitfile); //filen skal lukkes - den er ï¿½en i skrivebeskytte
    t tilstand
288                     FileIsOpen=FALSE;
289                  }
290                  if ((raw10bitfile=fopen(filename,"ab"))==NULL) //filen aabnes for at appende
     binï¿½ data
291                  {
292                     printf("Unable to reopen datafile" );
293                     return (TRUE);
294                  }
295                  else
296                  {
297                     FileIsOpen = TRUE;
298                     printf("File opened for append" );
299                     fwrite(pakke,pakkelng,1,raw10bitfile);
300                  }
301               }
302               else
303               {
304                              //printf("ID: %i %i,",type,pakkelng);
305                  printf("Der er ikke sync og heller ikke en pending fil\n"  );
306               }
307            }
308         }
309
310         if (type == DFU_DATAEND)
311            {
312         if (FileIsOpen){
313                  printf(" -END OF RECEIVED DATA- \n" );
314                  fwrite(pakke,pakkelng,1,raw10bitfile);
315
316   //-------------------------------------------------------------------------------------------
    --//
317   //    find baneelementer nu, for satelitnavnet skal bruges til at lave filnavne  //
318   //-------------------------------------------------------------------------------------------
    --//
319               //spol fil tilbage for at lï¿½e tidstempel og satellit ID
320               //rewind(raw10bitfile);
321               fclose(raw10bitfile);
322               if ((raw10bitfile=fopen(filename,"rb"))==NULL) //filen aabnes for at laese bina
    erï¿½data
323               {
324                  printf("Unable to reopen datafile" );
325                  return (TRUE);
326               }
327   /*--------------------------------- FOR TESTING ! ---------------------------------------*/
328   //Luk nymodtagne outputfile og ï¿½en eksisterende NOAA-10bit fil i stedet. Denne vil sï¿½blive
     brug til navngivning
329   //Men det er stadig den nye fil som bliver renamed - den eksisterende fil forbliver uberrt
330            if (TESTMODE) {
331               if(FileIsOpen) {
332                  fclose(raw10bitfile);
333                  FileIsOpen = FALSE;
334               }
335
336               if ((raw10bitfile=fopen("/home/radman/UserRod/UserSource/0602141001.NOAA17.045"
    ,"rb"))==NULL)
337               {
338                  printf("Cannot open file 0602141001.NOAA17.045\n"  );
339                  return (TRUE);
340               }
341            }
342   /*--------------------------------- END "for testing"-----------------------------------*/
343               fread(pakke,100,1,raw10bitfile);
344               scadr=((pakke[7]&1)<<3)|(pakke[8]&224)>>5;       // laes sattelit ID
345
346               yearday = (pakke[10]<<1)|((pakke[11]&128)>>7);
347               ms = ((pakke[11]&7)<<24) | (pakke[12]<<16) | (pakke[13]<<8) | pakke[14] ;
348               printf("Dag: %i, ms: %i scadr: %i" ,yearday,ms,scadr);
349               find_tid(&aar,&maan,&dag,&tim,&min,&i);  //der er ikke noget ï¿½aarstal i NOAA-
    headeren, sï¿½man bruger ï¿½stallet for modtagelsen
350               Pb->tstart=julia(aar, 1, yearday) + ms/864E5; //finder tidspunkt i dage efter d
    en Julianske kalender
```

```
351                 j2d(Pb->tstart, &aar,&maan,&dag,&tim,&min,&sek);    //finder satellitdatas star
    t-tidspunkt i aar, maaned, dag, time, sekund
352
353
354                 //scadr=13; //Kun for at kunne teste med NOAA-generator
355                 if (get_elements (Pb, scadr, ini) != TRUE)
356                 {
357                    printf("Get_element error\n" );
358                    return (TRUE); //her stopper programmet, hvor det burde haandtere situatione
    n og gaa tilbage og vente paa en ny klientforbindelse.
359                 }
360                 Pb->wm=55.4/GR;
361                 //printf("Pb navn: %s\n",Pb->navn);
362 //-------------------------------------------------------------------------------------------
    --//
363
364                 //Laesï¿½aar, Maaned, Dag, Time og minut i en streng saaï¿½de kan bruges i filn
    avne
365                 divresult=div(aar,100);
366                 divresult=div(divresult.rem,10);
367                 receivetime[0]= divresult.quot + 48;
368                 receivetime[1]= divresult.rem + 48;
369                 divresult=div(maan,10);
370                 receivetime[2]= divresult.quot + 48;
371                 receivetime[3]= divresult.rem + 48;
372                 divresult=div(dag,10);
373                 receivetime[4]= divresult.quot + 48;
374                 receivetime[5]= divresult.rem + 48;
375                 divresult=div(tim,10);
376                 receivetime[6]= divresult.quot + 48;
377                 receivetime[7]= divresult.rem + 48;
378                 divresult=div(min,10);
379                 receivetime[8]= divresult.quot + 48;
380                 receivetime[9]= divresult.rem + 48;
381
382                 //Dag paaï¿½aaret skal ogsaaï¿½bruges i filnavn
383                 divresult=div(yearday,100);
384                 yeardaystr[0] = divresult.quot +48;
385                 divresult=div(divresult.rem,10);
386                 yeardaystr[1] = divresult.quot + 48;
387                 yeardaystr[2] = divresult.rem +48;
388                 strcpy(donefilename,ini->outdir);
389                 strncat(donefilename,receivetime,10);
390                 strncat(donefilename,".",1);
391                 (void) strncat(donefilename,Pb->navn,6);
392                 strncat(donefilename,".",1);
393                 strncat(donefilename,yeardaystr,3);
394
395                 status = rename(filename, donefilename);
396                 if (status != 0) printf("ERROR WHILE RENAMING FILE" );
397                 if(FileIsOpen) {
398                    fclose(raw10bitfile);
399                    FileIsOpen = FALSE;
400                 }
401 //-------------------------------- Kun til Test ! -----------------------------------------
    -------------
402                 if (TESTMODE) {
403                    strcpy(donefilename,"/home/radman/NOAAFiles/27jun/0602141001.NOAA17.045"   );//
    cheat for test
404                    printf("\nName of Rawdata file: " );
405                    printf(donefilename);
406                    printf("\nName of suposed direktory: "  );
407                    printf(ini->outdir);
408                 }
409 //-------------------------------- End Test -----------------------------------------------
    --------------
410                 status = separate(donefilename,Pb);
411                 if (status!=0)
412                 {
413                    printf("Error seperating rawfile!" );
414                 }
415                 status = map(5,maparg,donefilename,Pb,ini);
416                 if (status!=0)
```

```
417                 {
418                    printf("Error creating quickview!" );
419                 }
420              }
421              printf("closeconnection" );
422              close(connectionSocket );
423              readint=0;
424           }
425           }
426        if (FileIsOpen)
427        {
428           FileIsOpen  = FALSE;
429           fclose(raw10bitfile );
430        }
431           }
432     }
433     else
434     {
435        close(connectionSocket );
436     }
437    printf("\nend of whileloekke" );
438     }
439  }
440
441
442  /*************************************************************************
443  // Udpaknings funktion
444  Aabner den faerdige NOAA 10bit-fil skaerer de to mindst betydende bit af hvert
445  ord i datapakkerne, og skilder data op i de 5 kannaler, saa der bliver 5 ch.dat
446  filer.
447  *************************************************************************/
448
449   /*FUNCTION*/
450  int separate(char inputfile[200],struct banel *Pb )
451  {
452  // header[] is here 938 bytes long even though the NOAA header is 750 words = 937.5 byte long
453  // This means that the last half byte of header will contain Earth data that must be transfere
     d to ch1.dat.
454  // It also means that the five dat/files ch1-5.dat will have half a byte of stuffing between t
     heir NOAA-headers
455  // and their data.
456     BYTE sync[AUXSYNC_LNTH],bmpheader[54],rest,n[400], header[938],NOAAheader[2048], chan1[64],
     chan2[64],chan3[64],chan4[64],chan5[64],chanbmp[64*3];
457     unsigned int heigth,temp,size;
458     unsigned int framesync[2]= {2702638449,2642657280};
459     double tstart;
460     float errpbit;
461     register int i,j,blockcounter;
462     const int bredde = 2048;
463     int aar,dag,ms,status,scadr,scdag,p,biterrors,syncbits,filstr;
464     div_t divresult;
465     FILE *inff,*out1, *out2, *out3, *out4,*out5;//,*outbmp;
466     char filename[200];
467     heigth=0;
468     FILE *in;
469
470  //---------------------------------------------------------------------
471  // OPEN FILES (and create temp outputfile)
472  //---------------------------------------------------------------------
473     if ((in=fopen(inputfile,"rb"))== NULL)
474     {
475        printf("Unable to open datafile" );
476        return (TRUE);
477     }
478     strcpy(filename,inputfile);
479     (void) strncat(filename,(char *) ".INFO",5);
480     inff = fopen(filename,"w");
481     if(inff == NULL) {
482        printf("Cannot make file .INFO\n" );
483        return 1;
484     }
485
486     strcpy(filename,inputfile);
```

```
487     (void) strncat(filename,(char *) "CH1.DAT",7);
488     out1 = fopen(filename,"wb");
489     if(out1 == NULL) {
490        printf("Cannot open file CH1.DAT.\n" );
491        return 1;
492     }
493
494     strcpy(filename,inputfile);
495     (void) strncat(filename,(char *) "CH2.DAT",7);
496     out2 = fopen(filename,"wb");
497     if(out2 == NULL) {
498        printf("Cannot open file CH2.DAT.\n" );
499        return 1;
500     }
501     strcpy(filename,inputfile);
502     (void) strncat(filename,(char *) "CH3.DAT",7);
503     out3 = fopen(filename,"wb");
504     if(out3 == NULL) {
505        printf("Cannot open file CH3.DAT\n" );
506        return 1;
507     }
508     strcpy(filename,inputfile);
509     (void) strncat(filename,(char *) "CH4.DAT",7);
510     out4 = fopen(filename,"wb");
511     if(out4 == NULL) {
512        printf("Cannot open file CH4.DAT.\n" );
513        return 1;
514     }
515     strcpy(filename,inputfile);
516     (void) strncat(filename,(char *) "CH5.DAT",7);
517     out5 = fopen(filename,"wb");
518     if(out5 == NULL) {
519        printf("Cannot open file CH5.DAT.\n" );
520        return 1;
521     }
522     /*strcpy(filename,inputfile);
523     (void) strncat(filename,(char *) "CHbmp.BMP",9);
524     outbmp = fopen("/home/radman/NOAAFiles/27jun/CHbmp.BMP","wb");
525     if(outbmp == NULL) {
526        printf("Cannot open file CHbmp.BMP.\n");
527        return 1;
528     }*/
529  //---------------------------------------------------------------------
530  // READ HEADER
531  //---------------------------------------------------------------------
532
533     fread((char *)header,938,1,in);
534
535     scadr=((header[7]&1)<<3)|(header[8]&224)>>5;        // li¿½ sattelit ID
536     scdag = (header[10]<<1)|((header[11]&128)>>7);
537     p = ((header[11]&7)<<24) | (header[12]<<16) | (header[13]<<8) | header[14];
538
539
540     rest = header[938]&0x0f;
541
542     blockcounter=biterrors=syncbits=0;
543     while(!feof(in)){
544        heigth++;
545        for (j=0;j<32;j++){
546  //-------------------------------------------------------------------------
547  //      WRITE DATA TO FILES - Prosess en Ramme
548  //-------------------------------------------------------------------------
549
550           fread((char *) &n,sizeof n,1,in);
551              for(i=0; i<16; i++){
552              chan1[4*i+0]= (rest<<4)|(n[25*i]>>4);
553                    chan2[4*i+0]=   (n[25*i]<<6) | (n[25*i+1]>>2);
554                    chan3[4*i+0]= (n[25*i+2]); //| (n[25*i+3]>>6);
555                    chan4[4*i+0]=(n[25*i+3]<<2) | (n[25*i+4]>>6);
556                    chan5[4*i+0]= (n[25*i+4]<<4)|(n[25*i+5]>>4);
557
558                    chanbmp[12*i+0]= chan1[4*i+0];
559                    chanbmp[12*i+1]= chan1[4*i+0];
```

```
560                     chanbmp[12*i+2]= chan1[4*i+0];
561
562
563                     chan1[4*i+1]=    (n[25*i+5]<<6) | (n[25*i+6]>>2);
564                     chan2[4*i+1]= (n[25*i+7]);// | (n[25*i+8]>>8);
565                     chan3[4*i+1]=(n[25*i+8]<<2) | (n[25*i+9]>>6);
566                     chan4[4*i+1]= (n[25*i+9]<<4)|(n[25*i+10]>>4);
567
568                  chanbmp[12*i+3]= chan1[4*i+1];
569               chanbmp[12*i+4]= chan1[4*i+1];
570               chanbmp[12*i+5]= chan1[4*i+1];
571
572
573               chan5[4*i+1]=    (n[25*i+10]<<6)| (n[25*i+11]>>2);
574               chan1[4*i+2]= (n[25*i+12]);// | (n[25*i+13]>>8);
575               chan2[4*i+2]=(n[25*i+13]<<2) | (n[25*i+14]>>6);
576               chan3[4*i+2]= (n[25*i+14]<<4)|(n[25*i+15]>>4);
577
578               chanbmp[12*i+6]= chan1[4*i+2];
579               chanbmp[12*i+7]= chan1[4*i+2];
580               chanbmp[12*i+8]= chan1[4*i+2];
581
582
583               chan4[4*i+2]=    (n[25*i+15]<<6) | (n[25*i+16]>>2);
584               chan5[4*i+2]= (n[25*i+17]); //| (n[25*i+17]>>2);
585               chan1[4*i+3]=(n[25*i+18]<<2) | (n[25*i+19]>>6);
586               chan2[4*i+3]= (n[25*i+19]<<4)|(n[25*i+20]>>4);
587
588
589               chan3[4*i+3]=    (n[25*i+20]<<6) | (n[25*i+21]>>2);
590               chan4[4*i+3]= (n[25*i+22]);// | (n[25*i+23]>>6);
591               chan5[4*i+3]=(n[25*i+23]<<2) | (n[25*i+24]>>6);
592
593
594               chanbmp[12*i+9]= chan1[4*i+3];
595               chanbmp[12*i+10]= chan1[4*i+3];
596               chanbmp[12*i+11]= chan1[4*i+3];
597
598               rest= n[25*i+24];
599            }
600            if(blockcounter ==0)
601            {
602               aar = (inputfile[29]-48)*10+inputfile[30]-48;
603               //printf("\nAar: %i",aar);
604               //printf("\nin1: %i, in2: %i\n",inputfile[29],inputfile[30]);
605               chan1[0]=chan2[0]=chan3[0]=chan4[0]=chan5[0]=chanbmp[0]=scadr;
606               chan1[1]=chan2[1]=chan3[1]=chan4[1]=chan5[1]=chanbmp[1]=aar%100;
607
608               chan1[2]=chan2[2]=chan3[2]=chan4[2]=chan5[2]=chanbmp[2]=scdag/100;
609               chan1[3]=chan2[3]=chan3[3]=chan4[3]=chan5[3]=chanbmp[3]=scdag%100;
610               chan1[7]=chan2[7]=chan3[7]=chan4[7]=chan5[7]=chanbmp[7]=p%100L;
611               chan1[6]=chan2[6]=chan3[6]=chan4[6]=chan5[6]=chanbmp[6]=(p/100L)%100;
612               chan1[5]=chan2[5]=chan3[5]=chan4[5]=chan5[5]=chanbmp[5]=(p/10000L)%100;
613               chan1[4]=chan2[4]=chan3[4]=chan4[4]=chan5[4]=chanbmp[4]=(p/1000000L)%100;
614            }
615   //-------------------------------------------------------------------------
616   //       WRITE TO OUTPUT FILE
617   //-------------------------------------------------------------------------
618            fwrite(chan1,sizeof chan1,1,out1);
619            fwrite(chan2,sizeof chan2,1,out2);
620            fwrite(chan3,sizeof chan3,1,out3);
621            fwrite(chan4,sizeof chan4,1,out4);
622            fwrite(chan5,sizeof chan5,1,out5);
623            //fwrite(chanbmp,sizeof chanbmp,1,outbmp);
624
625            blockcounter ++;
626         }
627
628         fread((char *) &sync, AUXSYNC_LNTH,1,in);
629         //printf("biterrors: %i, syncbits: %i ",biterrors,syncbits);
630         biterrors = biterrors + synccheck(sync);
631         syncbits= syncbits + (AUXSYNC_LNTH-1)*8; //foerste og sidste halve byte af auxsync check
    es ikke
```

```
632
633         fread((char *) &header, sizeof header,1,in);
634         rest = header[938]&0x0f;
635       }
636
637
638   /*for (i=0;i<54;i++) bmpheader[i]=0;
639   bmpheader[0]='B';
640   bmpheader[1]='M';
641   size = heigth*bredde*3;
642
643   bmpheader[5]= (size&0xff000000)>>24;
644   bmpheader[4]= (size&0x00ff0000)>>16 ;
645   bmpheader[3]= (size&0x0000ff00)>>8  ;
646   bmpheader[2]= (size&0x000000ff)    ;
647
648   bmpheader[10]= 0x36;       //size of header;
649   bmpheader[14]= 0x28;       //size of infoheader;
650
651   bmpheader[18]=  bredde&0xff;           //pixel bredde
652   bmpheader[19]=  (bredde&0xff00)>>8;           //pixel bredde
653   bmpheader[20]=  (bredde&0xff0000)>>16;           //pixel bredde
654   bmpheader[21]=  (bredde&0xff000000)>>24;           //pixel bredde
655
656   bmpheader[22]= heigth&0xff;             //pixel height
657   bmpheader[23]=  ((heigth)&0xff00)>>8;
658   bmpheader[24]=  ((heigth)&0xff0000)>>16;
659   bmpheader[25]=  ((heigth)&0xff000000)>>24;
660
661
662   bmpheader[26]=1;        //planes
663   bmpheader[28]=24; //bit per pixel
664   bmpheader[30]=0; //biCompression*/
665
666   //fseek(outbmp,0,SEEK_SET);
667   //fwrite(bmpheader,54,1,outbmp);
668
669
670
671   filstr= (int) (filesize(out1)/2048L);
672
673   fprintf(inff,"Sattelite: %s, " ,Pb->navn);
674   fprintf(inff,"Year:%i, Day:%i, %i \n" ,aar,scdag,p);
675   fprintf(inff,"Lines in passage: %i \n" ,filstr);
676   printf("bits: %i   errors: %i " ,syncbits,biterrors);
677   //fprintf(inff,"bits: %i   errors: %i \n",syncbits,biterrors);
678   errpbit = (double)biterrors/(double)syncbits;
679   fprintf(inff,"biterrors per bit: %f " ,errpbit);
680
681      fclose(in);
682      fclose(out1);
683      fclose(out2);
684      fclose(out3);
685      fclose(out4);
686      fclose(out5);
687      //fclose(outbmp);
688      fclose(inff);
689
690      //maparg[0] = (char)30;
691      //maparg[1] = (char)30;
692      //maparg[2] = (char)50;
693      //status = map(5,maparg,Pb);
694      return 0;
695   }
696
697
698   /*****************************FUNKTIONER*************************************/
699   /*FUNCTION*/
700   int map(int argc, char argv[],char inputfile[200],struct banel *Pb,struct ini_info *ini)
701   {
702
703    struct kort geografi, *Pg;
704    BYTE indat1[6000][2048],indat2[6000][2048],indat3[6000][2048],
```

```
705         indat4[6000][2048],indat5[6000][2048],uddat1[midx*2],uddat2[midx*2],
706         uddat3[midx*2],uddat4[midx*2],uddat5[midx*2],uddat6[midx*6],labelArray[50][600],skabelon
    [midy*2][midx*2],skatemp[midy*2][midx*6],bmpheader[54],cha1min,cha1max,cha2min,cha2max,cha3min
    ,cha3max,cha4min,cha4max,cha5min,cha5max
707         ,chan1min,chan1max,chan2min,chan2max,chan3min,chan3max,chan4min,chan4max,chan5min,chan5m
    ax;
708   double lmid,bmid,btop,tid,ar,de,m,h;
709   float sek,l,b,l1,l2,l3,l4,p1,p2,p3,p4,grid[18][46][2],line,pixel,
710   lv,lh,pv,ph,tlav,thoj;
711   int kanal,aar,maan,dag,tim,min,scadr,filstr,i,j,x,y,z,zoom,scan,
712       i0,i1,i2,i3,j0,j1,j2,j3,pmin,pmax,lmin,lmax,ip,il,scdag,indx,cifres,dis1[13],dis2[13],dis
    4[13];
713   long int ms,offset,varighed,test;
714   FILE *fpind, *fpu1,*fpu2,*fpu3,*fpu4,*fpu5,*fpu6,*fpu7;
715   char filename[200], labelstring[200],intstr[20];
716
717   memset(dis1,0,13);
718   memset(dis2,0,13);
719   memset(dis4,0,13);
720   Pg=&geografi;
721
722  argc=3;
723
724
725
726   find_tid(&aar,&maan,&dag,&tim,&min,&i);  //hent tid for at maale hvor lang tid beregningerne
    tager
727   varighed=tim*3600 + min*60 + i;
728
729  //printf("lmid: %lf, bmid %lf, btop %lf \n",ini->tlmid,ini->tbmid,ini->tbtop);
730   // Omregn billedudsnitskoordinater fra grader til radianer og beregn derefter
731   // det oenskede billedudsnit.
732   ini->tlmid/=GR;
733   ini->tbmid/=GR;
734   ini->tbtop/=GR;
735   Pg->r=midy/2.0/(tan(M_PI/4.0 - ini->tbmid/2.0) - tan(M_PI/4.0 - ini->tbtop/2.0));
736   Pg->ycntr=2.0*Pg->r*tan(M_PI/4.0 - ini->tbmid/2.0);
737   Pg->latlimit=88/GR;
738   Pg->midlong=ini->tlmid;
739
740  //printf("lmid: %lf, bmid %lf, btop %lf \n",ini->tlmid,ini->tbmid,ini->tbtop);
741
742  /*************************************************************************/
743  // OPEN INPUT-FILES                                                      //
744  //De fem DAT-filer aabnes og laegges i hvert deres array: indat1-5          //
745  /*************************************************************************/
746
747  // SKABELON TIL VEJRKORT      //
748   strcpy(filename,ini->helpdir);
749   strncat(filename,"Template.DAT",12);
750   fpind=fopen(filename,"rb");
751
752   if (fpind == NULL)
753   {
754    printf("Template.DAT findes ikke!\n" );
755    return (TRUE);
756   }
757   filstr= (int) ((filesize(fpind)-54)/(midx*6));
758   fread(bmpheader,54,1,fpind);
759
760   if (drawkonturs(skabelon, Pg, ini)>0)
761     printf("konturer kan ikke tegnes!" );
762   fclose(fpind);
763
764   //   INPUTFIL FOR KANAL 1    //
765   strcpy(filename,inputfile);
766   (void) strncat(filename,(char *) "CH1.DAT",7);
767   fpind=fopen(filename,"rb");
768   if (fpind == NULL)
769   {
770    printf("CH1.DAT findes ikke!\n" );
771    return (TRUE);
772   }
```

```
773   filstr= (int) (filesize(fpind)/2048L);
774   for (j=0; j<filstr; j++) fread(indat1[j],2048,1,fpind);
775   fclose(fpind);
776
777   //   INPUTFIL FOR KANAL 2    //OUTPUT_DIR "/home/radman/NOAAFiles/"
778   strcpy(filename,inputfile);
779   (void) strncat(filename,(char *) "CH2.DAT",7);
780   fpind=fopen(filename,"rb");
781   if (fpind == NULL)
782   {
783    printf("CH2.DAT findes ikke!\n" );
784    return (TRUE);
785   }
786   filstr= (int) (filesize(fpind)/2048L);
787   for (j=0; j<filstr; j++) fread(indat2[j],2048,1,fpind);
788   fclose(fpind);
789
790   //   INPUTFIL FOR KANAL 3    //
791   strcpy(filename,inputfile);
792   (void) strncat(filename,(char *) "CH3.DAT",7);
793   fpind=fopen(filename,"rb");
794   if (fpind == NULL)
795   {
796    printf("CH3.DAT findes ikke!\n" );
797    return (TRUE);
798   }
799   filstr= (int) (filesize(fpind)/2048L);
800   for (j=0; j<filstr; j++) fread(indat3[j],2048,1,fpind);
801   fclose(fpind);
802
803   //   INPUTFIL FOR KANAL 4    //
804   strcpy(filename,inputfile);
805   (void) strncat(filename,(char *) "CH4.DAT",7);
806   fpind=fopen(filename,"rb");
807   if (fpind == NULL)
808   {
809    printf("CH4.DAT findes ikke!\n" );
810    return (TRUE);
811   }
812   filstr= (int) (filesize(fpind)/2048L);
813   for (j=0; j<filstr; j++) fread(indat4[j],2048,1,fpind);
814   fclose(fpind);
815
816
817   //   INPUTFIL FOR KANAL 5    //
818   strcpy(filename,inputfile);
819   (void) strncat(filename,(char *) "CH5.DAT",7);
820   fpind=fopen(filename,"rb");
821   if (fpind == NULL)
822   {
823    printf("CH5.DAT findes ikke!\n" );
824    return (TRUE);
825   }
826   filstr= (int) (filesize(fpind)/2048L);
827   for (j=0; j<filstr; j++) fread(indat5[j],2048,1,fpind);
828   fclose(fpind);
829
830
831  /*************************************************************************/
832  // Laes scan starttidspunkt med mere                                     //
833  /*************************************************************************/
834
835   scadr=indat1[0][0];        // laes sattelit ID
836   scadr=(scadr & 30)>>1;
837   aar=indat1[0][1];
838   if (aar < 95) aar+=2000; else aar+=1900;
839   dag=indat1[0][2]*100+indat1[0][3];
840   scdag=dag;
841   printf("aar: %i dag: %i \n" ,aar,dag);
842   ms=indat1[0][7]+100L*(indat1[0][6]+100L*(indat1[0][5]+100L*indat1[0][4]));
843   tlav=indat1[0][18] + 0.01*indat1[0][19];
844   thoj=indat1[0][20] + 0.01*indat1[0][21];
845
```

```
846  /*************************************************************************/
847  // OPEN OUTPUT-FILES                                                     //
848  //De fem billed-filer aabnes og                                          //
849  /*************************************************************************/
850
851   strcpy(filename,inputfile);
852   (void) strncat(filename,(char *) "kontur.1",8);
853   fpu1=fopen(filename,"wb");
854   strcpy(filename,inputfile);
855   (void) strncat(filename,(char *) "kontur.2",8);
856   fpu2=fopen(filename,"wb");
857   strcpy(filename,inputfile);
858   (void) strncat(filename,(char *) "kontur.3",8);
859   fpu3=fopen(filename,"wb");
860   strcpy(filename,inputfile);
861   (void) strncat(filename,(char *) "kontur.4",8);
862   fpu4=fopen(filename,"wb");
863   strcpy(filename,inputfile);
864   (void) strncat(filename,(char *) "kontur.5",8);
865   fpu5=fopen(filename,"wb");
866   strcpy(filename,inputfile);
867   (void) strncat(filename,(char *) "kontur.bmp",10);
868   fpu6=fopen(filename,"wb");
869   strcpy(filename,inputfile);
870   (void) strncat(filename,(char *) "konturfar.bmp",13);
871   fpu7=fopen(filename,"wb");
872
873
874    fwrite(bmpheader,54,1,fpu6);
875    fwrite(bmpheader,54,1,fpu7);
876
877  /*************************************************************************/
878  // LAV GRID                                                              //
879  //Der laves et grid af laengde og bredde-radianer, som fortaeller hvor hver    //
880  //128. pixel i NOAA-data hrer til paa jordkloden. Dette grid skal bruges til//
881  //at interpolerer ud fra, saaledes at man ikke behoever at lave den fulde      //
882  //udregning for alle punkter i NOAA-arrayet                             //
883  /*************************************************************************/
884  j2d(Pb->tstart, &aar,&maan,&dag,&tim,&min,&sek);
885  printf("aar: %i, maan %i, dag: %i, tim: %i, min %i, sek: %f\n"   ,aar,maan,dag,tim,min,sek);
886
887              printf("\n tstart: %f \n",Pb->tstart);
888   for (i=0; i<18; i++)
889   for (j=0; j<46; j++)
890   {
891    pixel=i*128.0-64.0;
892    line=j*128.0-128.0;
893    sgpsat(line, pixel, &l, &b, Pb);
894    grid[i][j][0]=b;
895    grid[i][j][1]=l;
896    //printf("%f ,%f : %f ,%f ",pixel,line,l,b);
897   }
898
899
900  /*************************************************************************/
901  // SKRIV DATA I OUTPUT FILER                                             //
902  //For hver pixel i outputbilledet (800X600 pixel) skrives NOAA-data hvis    //
903  //punktet findes inden for sattelittens synsfelt - altsi¿½ hvis NOAA-data   //
904  //indeholder data for den li¿½gde og bredde som pixlet representerer - ellers//
905  //skrives '0'                                                           //
906  //Man afgr om der er NOAA-data til det valgte pixel frst ved at se om    //
907  //punktet findes inden for det genererede grid, og hvis det er tilfi¿½det si¿½//
908  // interpolerer man og undersger om den ndvendige frame og byte begge er  //
909  //strre end 0 og mindre en hendholdsvis antal modtagne frames og 2048       //
910  //                                                                       //
911
912  /*************************************************************************/
913   cha1max=cha2max=cha3max=cha4max=cha5max=0;
914  cha1min=cha2min=cha3min=cha4min=cha5min=255;
915
916
917
918   for (i=0; i<(2*midy); i++)    //for hver linje i outputbilledet
```

```
919   {
920
921    chan1max =chan2max =chan3max =chan4max =chan5max =0;
922   chan1min =chan2min =chan3min =chan4min =chan5min =255;
923
924     if ((i%50)==0) printf("Linje %3d\n",i);
925     for (j=0; j<midx*2; j++)       // for hvert pixel i billedlinjen
926     {
927      x=j;
928      y=(midy*2 -i);
929      vduxy2lonlat (x,     y,     &l, &b, Pg);  // udregn laengde og bredde af det
930      //oenskede pixel i outputbilledet
931      //Lav en linje gennem breddegrader ...-20,-10,0,10, 20 ... og ligeledes for laengdegrader
932      if(((fmod((b*GR ),10)<0.1)&&(fmod((b*GR+2),10)>-0.1)) || ((fmod((l*GR),10)<0.1)&&(fmod((l*G
    R),10)>-0.1))) test = TRUE;
933      else test = false;
934      /*if(((b*GR)<33.1)&&((b*GR)>33)) test = TRUE;
935      if(((b*GR)<55.85)&&((b*GR)>55.75)) test = TRUE;*/
936      if (seek(b, l, grid, 45, &i0, &j0))     //undersoeg om l,b ligger inden for grid
937      {
938         interpoler(i0, j0, b, l, grid, &pixel, &line); //find det praesise byte hvor i man skal
    finde data
939         i0=pixel;
940         j0=line;
941         if (i0 >=0 && i0 < 2048 && j0 >=0 && j0 < filstr)     //hvis i0,j0 findes i NOAA-data
942         {
943      uddat1[j]=indat1[j0][i0];
944      uddat2[j]=indat2[j0][i0];
945      uddat3[j]=indat3[j0][i0];
946      uddat4[j]=indat4[j0][i0];
947      uddat5[j]=indat5[j0][i0];
948      if (skabelon[i][j]==2) {
949         uddat6[j*3]= 2;
950         uddat6[j*3+1]=255;
951         uddat6[j*3+2]=0;
952         skatemp[i][j*3]= 2;
953         skatemp[i][j*3+1]=255;
954         skatemp[i][j*3+2]=0;
955      }
956      else
957      {
958         uddat6[j*3]=indat1[j0][i0];
959         uddat6[j*3+1]=indat2[j0][i0];
960         uddat6[j*3+2]=indat4[j0][i0];
961         skatemp[i][j*3]=indat1[j0][i0];
962         skatemp[i][j*3+1]=indat2[j0][i0];
963         skatemp[i][j*3+2]=indat4[j0][i0];
964
965         //Find minimum og maximum for hver kanal, saa hele bitbredden kan udnyttes:
966         if((cha1min>indat1[j0][i0])&(indat1[j0][i0]>0)) cha1min = (cha1min*9 + indat1[j0][i0])/1
    0;
967         if(cha1max<indat1[j0][i0]) cha1max = (cha1max*9 + indat1[j0][i0])/10;
968         if((cha2min>indat2[j0][i0])&(indat1[j0][i0]>0)) cha2min = (cha2min*9 + indat2[j0][i0])/1
    0;
969         if(cha2max<indat2[j0][i0]) cha2max = (cha2max*9 + indat2[j0][i0])/10;
970         if((cha3min>indat3[j0][i0])&(indat1[j0][i0]>0)) cha3min = (cha3min*9 + indat3[j0][i0])/1
    0;
971         if(cha3max<indat3[j0][i0]) cha3max = (cha3max*9 + indat3[j0][i0])/10;
972         if((cha4min>indat4[j0][i0])&(indat1[j0][i0]>0)) cha4min = (cha4min*9 + indat4[j0][i0])/1
    0;
973         if(cha4max<indat4[j0][i0]) cha4max = (cha4max*9 + indat4[j0][i0])/10;
974         if((cha5min>indat5[j0][i0])&(indat1[j0][i0]>0)) cha5min = (cha5min*9 + indat5[j0][i0])/1
    0;
975         if(cha5max<indat5[j0][i0]) cha5max = (cha5min*9 + indat5[j0][i0])/10;
976
977         //find min og max i hver linje
978         if((chan1min>indat1[j0][i0])&(indat1[j0][i0]>0)) chan1min = (chan1min*1 + indat1[j0][i0]
    )/2;
979         if(chan1max<indat1[j0][i0]) chan1max = (chan1max*1 + indat1[j0][i0])/2;
980         if((chan2min>indat2[j0][i0])&(indat1[j0][i0]>0)) chan2min = (chan2min*1 + indat2[j0][i0]
    )/2;
981         if(chan2max<indat2[j0][i0]) chan2max = (chan2max*1 + indat2[j0][i0])/2;
982         if((chan3min>indat3[j0][i0])&(indat1[j0][i0]>0)) chan3min = (chan3min*1 + indat3[j0][i0]
```

```
    )/2;
983         if(chan3max<indat3[j0][i0]) chan3max  = (chan3max*1 + indat3[j0][i0])/2;
984         if((chan4min>indat4[j0][i0])&(indat1[j0][i0]>0)) chan4min  = (chan4min*1 + indat4[j0][i0]
    )/2;
985         if(chan4max<indat4[j0][i0]) chan4max  = (chan4max*1 + indat4[j0][i0])/2;
986         if((chan5min>indat5[j0][i0])&(indat1[j0][i0]>0)) chan5min  = (chan5min*1 + indat5[j0][i0]
    )/2;
987         if(chan5max<indat5[j0][i0]) chan5max  = (chan5min*1 + indat5[j0][i0])/2;
988      }
989         }
990         else
991         {  //if pixel is within the prepared grid, but still not within the are covered by data
992            uddat1[j]=skabelon[i][j];
993      uddat2[j]=skabelon[i][j];
994      uddat3[j]=skabelon[i][j];
995      uddat4[j]=skabelon[i][j];
996      uddat5[j]=skabelon[i][j];
997      uddat6[j*3]=uddat6[j*3+1]=uddat6[j*3+2]=skabelon[i][j];
998      skatemp[i][j*3]=skatemp[i][j*3+1]=skatemp[i][j*3+2]=skabelon[i][j];
999         }
1000     }
1001     else
1002     {  //The pixel is outside the prepared grid
1003        uddat1[j]=skabelon[i][j];
1004        uddat2[j]=skabelon[i][j];
1005        uddat3[j]=skabelon[i][j];
1006        uddat4[j]=skabelon[i][j];
1007        uddat5[j]=skabelon[i][j];
1008        uddat6[j*3]=uddat6[j*3+1]=uddat6[j*3+2]=skabelon[i][j];
1009        skatemp[i][j*3]=skatemp[i][j*3+1]=skatemp[i][j*3+2]=skabelon[i][j];
1010     }
1011     if (test) {
1012        uddat6[j*3]=uddat6[j*3+1]=uddat6[j*3+2]=0;
1013     skatemp[i][j*3]=skatemp[i][j*3+1]=skatemp[i][j*3+2]=0;
1014     }
1015
1016     if (!i)//(!y)
1017     {
1018        uddat1[0] = scadr;
1019        uddat1[1] = aar % 100;
1020        uddat1[2] = scdag/100;
1021        uddat1[3] = scdag%100;
1022        uddat1[7] = ms%100L;
1023        uddat1[6] = (ms/100L)%100L;
1024        uddat1[5] = (ms/10000L)%100L;
1025        uddat1[4] = (ms/1000000L)%100L;
1026        if (ini->tlmid < 0.0) uddat1[8]=1; else uddat1[8]=0;
1027        z= (int) fabs(ini->tlmid*GR*100);
1028        uddat1[9]=z/100;
1029        uddat1[10]=z%100;
1030        if (ini->tbmid < 0.0) uddat1[11]=1; else uddat1[11]=0;
1031        z= (int) fabs(ini->tbmid*GR*100);
1032        uddat1[12]=z/100;
1033        uddat1[13]=z%100;
1034        if (ini->tbtop < 0.0) uddat1[14]=1; else uddat1[14]=0;
1035        z= (int) fabs(ini->tbtop*GR*100);
1036        uddat1[15]=z/100;
1037        uddat1[16]=z%100;
1038        uddat1[17]=kanal;
1039        uddat1[18] = (int) (tlav);
1040        uddat1[19] = (int) fmod(tlav*100.0, 100.0);
1041        uddat1[20] = (int) (thoj);
1042        uddat1[21] = (int) fmod(thoj*100.0, 100.0);
1043
1044        uddat2[0] = scadr;
1045        uddat2[1] = aar % 100;
1046        uddat2[2] = scdag/100;
1047        uddat2[3] = scdag%100;
1048        uddat2[7] = ms%100L;
1049        uddat2[6] = (ms/100L)%100L;
1050        uddat2[5] = (ms/10000L)%100L;
1051        uddat2[4] = (ms/1000000L)%100L;
1052        if (ini->tlmid < 0.0) uddat2[8]=1; else uddat2[8]=0;
```

```
1053        z= (int) fabs(ini->tlmid*GR*100);
1054        uddat2[9]=z/100;
1055        uddat2[10]=z%100;
1056        if (ini->tbmid < 0.0) uddat2[11]=1; else uddat2[11]=0;
1057        z= (int) fabs(ini->tbmid*GR*100);
1058        uddat2[12]=z/100;
1059        uddat2[13]=z%100;
1060        if (ini->tbtop < 0.0) uddat2[14]=1; else uddat2[14]=0;
1061        z= (int) fabs(ini->tbtop*GR*100);
1062        uddat2[15]=z/100;
1063        uddat2[16]=z%100;
1064        uddat2[17]=kanal;
1065        uddat2[18] = (int) (tlav);
1066        uddat2[19] = (int) fmod(tlav*100.0, 100.0);
1067        uddat2[20] = (int) (thoj);
1068        uddat2[21] = (int) fmod(thoj*100.0, 100.0);
1069
1070        uddat3[0] = scadr;
1071        uddat3[1] = aar % 100;
1072        uddat3[2] = scdag/100;
1073        uddat3[3] = scdag%100;
1074        uddat3[7] = ms%100L;
1075        uddat3[6] = (ms/100L)%100L;
1076        uddat3[5] = (ms/10000L)%100L;
1077        uddat3[4] = (ms/1000000L)%100L;
1078        if (ini->tlmid < 0.0) uddat3[8]=1; else uddat3[8]=0;
1079        z= (int) fabs(ini->tlmid*GR*100);
1080        uddat3[9]=z/100;
1081        uddat3[10]=z%100;
1082        if (ini->tbmid < 0.0) uddat3[11]=1; else uddat3[11]=0;
1083        z= (int) fabs(ini->tbmid*GR*100);
1084        uddat3[12]=z/100;
1085        uddat3[13]=z%100;
1086        if (ini->tbtop < 0.0) uddat3[14]=1; else uddat3[14]=0;
1087        z= (int) fabs(ini->tbtop*GR*100);
1088        uddat3[15]=z/100;
1089        uddat3[16]=z%100;
1090        uddat3[17]=kanal;
1091        uddat3[18] = (int) (tlav);
1092        uddat3[19] = (int) fmod(tlav*100.0, 100.0);
1093        uddat3[20] = (int) (thoj);
1094        uddat3[21] = (int) fmod(thoj*100.0, 100.0);
1095
1096        uddat4[0] = scadr;
1097        uddat4[1] = aar % 100;
1098        uddat4[2] = scdag/100;
1099        uddat4[3] = scdag%100;
1100        uddat4[7] = ms%100L;
1101        uddat4[6] = (ms/100L)%100L;
1102        uddat4[5] = (ms/10000L)%100L;
1103        uddat4[4] = (ms/1000000L)%100L;
1104        if (ini->tlmid < 0.0) uddat4[8]=1; else uddat4[8]=0;
1105        z= (int) fabs(ini->tlmid*GR*100);
1106        uddat4[9]=z/100;
1107        uddat4[10]=z%100;
1108        if (ini->tbmid < 0.0) uddat4[11]=1; else uddat4[11]=0;
1109        z= (int) fabs(ini->tbmid*GR*100);
1110        uddat4[12]=z/100;
1111        uddat4[13]=z%100;
1112        if (ini->tbtop < 0.0) uddat4[14]=1; else uddat4[14]=0;
1113        z= (int) fabs(ini->tbtop*GR*100);
1114        uddat4[15]=z/100;
1115        uddat4[16]=z%100;
1116        uddat4[17]=kanal;
1117        uddat4[18] = (int) (tlav);
1118        uddat4[19] = (int) fmod(tlav*100.0, 100.0);
1119        uddat4[20] = (int) (thoj);
1120        uddat4[21] = (int) fmod(thoj*100.0, 100.0);
1121
1122        uddat5[0] = scadr;
1123        uddat5[1] = aar % 100;
1124        uddat5[2] = scdag/100;
1125        uddat5[3] = scdag%100;
```

```
1126        uddat5[7] = ms%100L;
1127        uddat5[6] = (ms/100L)%100L;
1128        uddat5[5] = (ms/10000L)%100L;
1129        uddat5[4] = (ms/1000000L)%100L;
1130        if (ini->tlmid < 0.0) uddat5[8]=1; else uddat5[8]=0;
1131        z= (int) fabs(ini->tlmid*GR*100);
1132        uddat5[9]=z/100;
1133        uddat5[10]=z%100;
1134        if (ini->tbmid < 0.0) uddat5[11]=1; else uddat5[11]=0;
1135        z= (int) fabs(ini->tbmid*GR*100);
1136        uddat5[12]=z/100;
1137        uddat5[13]=z%100;
1138        if (ini->tbtop < 0.0) uddat5[14]=1; else uddat5[14]=0;
1139        z= (int) fabs(ini->tbtop*GR*100);
1140        uddat5[15]=z/100;
1141        uddat5[16]=z%100;
1142        uddat5[17]=kanal;
1143        uddat5[18] = (int) (tlav);
1144        uddat5[19] = (int) fmod(tlav*100.0, 100.0);
1145        uddat5[20] = (int) (thoj);
1146        uddat5[21] = (int) fmod(thoj*100.0, 100.0);
1147     }
1148    }
1149
1150    //Srg for at data i hver RAMME (linje) for hver kanal bliver bredt ud over hele bitbredden s
     ï¿½edes at
1151    //chan1max bliver 255 og chan1min bliver 5
1152    /*for (j=0; j<midx*2; j++)      // for hvert pixel i billedlinjen
1153    {
1154     if ((uddat6[3*j]>2)&(uddat6[3*j]<255)) uddat6[3*j]= 5 + (uddat6[3*j]-chan1min)*250/(chan1ma
     x-chan1min);
1155     if ((uddat6[3*j+1]>2)&(uddat6[3*j+1]<255)) uddat6[3*j+1]= 5 + (uddat6[3*j+1]-chan2min)*250/
     (chan2max-chan2min);
1156     if ((uddat6[3*j+2]>2)&(uddat6[3*j+2]<255)) uddat6[3*j+2]= 5 + (uddat6[3*j+2]-chan4min)*250/
     (chan4max-chan4min);
1157    }*/
1158
1159    fwrite(uddat1,midx*2,1,fpu1);
1160    fwrite(uddat2,midx*2,1,fpu2);
1161    fwrite(uddat3,midx*2,1,fpu3);
1162    fwrite(uddat4,midx*2,1,fpu4);
1163    fwrite(uddat5,midx*2,1,fpu5);
1164    fwrite(uddat6,midx*6,1,fpu6);
1165
1166  }
1167
1168  //Srg for at data i hver kanalfil bliver bredt ud over hele bitbredden sï¿½edes at
1169  //cha1max bliver 255 og cha1min bliver 1. Dette skrives til quickviewbilledet konturfar.bmp -
     ikke til kontur.bmp
1170  for (i=0; i<(2*midy); i++)    //for hver linje i outputbilledet
1171  {
1172    for (j=0; j<midx*2; j++)       // for hvert pixel i billedlinjen
1173    {
1174       if ((skatemp[i][3*j]>cha1min)&(skatemp[i][3*j]<cha1max)) skatemp[i][3*j]= 1 + (skatemp[i
     ][3*j]-cha1min)*254/(cha1max-cha1min);
1175       if ((skatemp[i][3*j+1]>cha2min)&(skatemp[i][3*j+1]<cha2max)) skatemp[i][3*j+1]= 1 + (ska
     temp[i][3*j+1]-cha2min)*254/(cha2max-cha2min);
1176       if ((skatemp[i][3*j+2]>cha4min)&(skatemp[i][3*j+2]<cha4max)) skatemp[i][3*j+2]= 1 + (ska
     temp[i][3*j+2]-cha4min)*254/(cha4max-cha4min);
1177    }
1178    fwrite(skatemp[i],midx*6,1,fpu7);
1179  }
1180  //printf("\n%s\nYEAR:%i  DAY:%i HOUR: %i",Pb->navn,aar,scdag,tim);
1181  sprintf(labelstring,"%s\nYEAR:%i  DAY:%i HOUR: %i"  ,Pb->navn,aar,scdag,tim);
1182  make_label(labelArray,ini,labelstring);
1183  for (i=0;i<50;i++)
1184  {
1185    fseek(fpu6,54+(midx*6*midy*2)-midx*6*i,SEEK_SET);
1186    fwrite(labelArray[i],3*200,1,fpu6);
1187    fseek(fpu7,54+(midx*6*midy*2)-midx*6*i,SEEK_SET);
1188    fwrite(labelArray[i],3*200,1,fpu7);
1189  }
1190
```

```
1191  fclose(fpu1);
1192  fclose(fpu2);
1193  fclose(fpu3);
1194  fclose(fpu4);
1195  fclose(fpu5);
1196  fclose(fpu6);
1197  fclose(fpu7);
1198
1199 // printf("cha1min: %i, cha1max: %i, cha2min: %i, cha2max: %i\n ",cha1min,cha1max,cha2min,cha2
     max);
1200  find_tid(&aar,&maan,&dag,&tim,&min,&i);
1201  varighed=3600*tim + 60*min +i - varighed;
1202  printf("\nlinjer: %d, det tog %d sekunder\n"  ,filstr,varighed); //TEST
1203  return (FALSE);
1204 }
1205 /*****************************************************************/
1206
1207 /*FUNCTION*/
1208 int get_elements (struct banel *Pb, int scadr,struct ini_info *ini)
1209 {
1210  FILE *fpele,*fpids;
1211  char l[128],q[16],w[16],si[4],d3,sat_id[16],f,found=FALSE,filename[100];
1212  int aar,i,j,d1=0,d2;
1213  double a,b,c,d,e;
1214  long x;
1215  int tempint;
1216
1217  strcpy(filename,ini->satdir);
1218  strncat(filename,"SATELITE.IDS",12);
1219  fpids=fopen(filename,"rt");
1220  //fpids=fopen("/home/radman/Documents/SATELITE.IDS","rt");
1221  if (fpids == NULL) return (FALSE);
1222
1223  do
1224  {
1225   if (feof(fpids)) break;
1226   fgets(l,127,fpids);
1227   if (strstr(l,"#") == NULL)
1228   {
1229    sscanf(l,
1230    "name: %s indx: %s hd_id: %d year: %d launch: %d sat_id: %d active: %c frekv: %c"    ,
1231      &q,si,&d1,&aar,&d2,&j,&d3,&f);
1232    if (d1 == scadr)
1233    {
1234     sprintf(Pb->navn ,"%s%s",q,si);
1235     sprintf(sat_id,"1 %dU",j);
1236    }
1237    if (d1 == scadr) break; //hov hvorfor har jeg skrevet dette i en if-saetning for sig selv?
1238   }
1239  } while (!feof(fpids));
1240
1241  fclose(fpids);
1242  if (d1 != scadr) return (FALSE);
1243
1244  strcpy(filename,ini->satdir);
1245  strncat(filename,"WEATHER.TLE",11);
1246  fpele=fopen(filename,"rt");
1247   if (fpele == NULL) printf("Weather");
1248  if (fpele == NULL) return (FALSE);
1249
1250  do
1251  {
1252   fgets(l,127,fpele); //soeg efter navn
1253   if (feof(fpele)) break;
1254   if (strstr(l,sat_id) != NULL)
1255   {
1256    found=TRUE;
1257    insert_space(l, 20);
1258    sscanf(l,"%d %s %s %d %lf %lf" ,&j,q,w,&aar,&a,&b);
1259    aar+=1900;
1260    if (aar<1990) aar+=100; //Y2K !!!
1261    Pb->epo =julia(aar,1,0)+a;
1262    Pb->dr  =b;
```

```
1263    fgets(l,127,fpele);
1264    if (feof(fpele)) break;
1265    insert_space (l, 63);
1266    insert_space (l, 28);
1267    sscanf(l,"%d %s %lf %lf %i %li %lf %lf %lf"  ,&j,q,&a,&b,&tempint,&x,&c,&d,&e);
1268    //printf("j: %d\n q: %s\n a: %lf\nb: %lf\nx: %li\nc: %lf\n d: %lf\ne: %lf\n",j,q,a,b,x,c,d,
    e);
1269    Pb->ink =a/GR;
1270    Pb->sini =sin(a/GR);
1271    Pb->cosi =cos(a/GR);
1272    Pb->knu =b/GR;
1273    Pb->exc =x/1E07;
1274    printf("excentricity: %lf \n" ,Pb->exc);
1275    Pb->xi =sqrt((1.+Pb->exc )/(1.-Pb->exc ));
1276    Pb->per =c/GR;
1277    Pb->mid =d/GR;
1278    Pb->rot =e;
1279    e=Pb->rot *TOPI;
1280    a=pow(MY/(e*e), (1./3.));
1281    b=-1.5*J2*kvadrat(1./a)/pow((1-kvadrat(Pb->exc )), 1.5)*
1282    (1. - 1.5*kvadrat(Pb->sini ));
1283    Pb->axe =a*(1. + b/3. - kvadrat(b)/3.);
1284    a=1.5*J2*kvadrat(1./Pb->axe *(1-kvadrat(Pb->exc )))*e;
1285    Pb->dk =-a*cos(Pb->ink );
1286    Pb->dp = a*(2. - 2.5*kvadrat(Pb->sini ));
1287    Pb->axe *=6378.145;
1288    printf("Data fra %s scadr %d fundet\n"  ,Pb->navn,scadr);
1289    fclose(fpele);
1290     printf("tstart: %d \n epo: %d \n dr: %d\n ink: %d\n cosi: %d\n "   , Pb->tstart,Pb->epo,Pb->d
    r,Pb->ink,Pb->cosi);
1291    printf("4\n");
1292    return (found);
1293   } //if strstr...
1294   if (found) break;
1295  } while (!found);
1296  fclose(fpele);
1297  return (found);
1298 }
1299 /*FUNCTION*/
1300 void insert_space (char* l, int nr)
1301 {
1302  char m[128];
1303  int j;
1304
1305  strcpy(m,l);
1306  for (j=0; j<128; l[j++]='\0');
1307  for (j=0; j<nr; j++) l[j]=m[j];
1308  l[nr]=' ';
1309  for (j=nr; j<69; j++) l[j+1]=m[j];
1310 }
1311 /*FUNCTION*/
1312 void insert_point (char* l, int nr)
1313 {
1314  char m[128];
1315  int j;
1316
1317  strcpy(m,l);
1318  for (j=0; j<128; l[j++]='\0');
1319  for (j=0; j<nr; j++) l[j]=m[j];
1320  l[nr]='.';
1321  for (j=nr; j<69; j++) l[j+1]=m[j];
1322 }
1323 /*FUNCTION*/
1324 long filesize(FILE *stream)
1325 {
1326    long curpos, length;
1327
1328    curpos = ftell(stream);
1329    fseek(stream, 0L, SEEK_END);
1330    length = ftell(stream);
1331    fseek(stream, curpos, SEEK_SET);
1332    return length;
1333 }
```

```
1334 /*FUNCTION*/
1335 void sgpsat(float line, float pixel, float *longi, float *lati,
1336        struct banel *Pb)
1337 {
1338  double dt,lo,dm,lm,perm,knum,nm,ram,em,t,dpa,dka,
1339     templ,axnl,aynl,elong,perl,llong,rmida,e,rmidx,trueu,
1340     rmag,rvdt,rmgdt,temps,sin2u,cos2u,knus,sinux,
1341     cosux,sinu,cosu,ux,ra,de,arg/*,sinis,cosis,is*/ ;
1342
1343  t=Pb->tstart + (line + pixel/2048.0)/5184E02;
1344  dt=t - Pb->epo;
1345  lo=Pb->mid + Pb->knu + Pb->per;
1346  dm=TOPI*(Pb->rot + Pb->dr*dt)*dt;
1347  dpa=Pb->dp*dt;
1348  dka=Pb->dk*dt;
1349  lm=dm + dpa + dka + lo;
1350  lm=fmod(lm, TOPI);
1351  perm=fmod(Pb->per + dpa, TOPI);
1352  knum=fmod(Pb->knu+ dka, TOPI);
1353  nm=Pb->rot + 2*dt*Pb->dr;
1354  ram=Pb->axe*pow(Pb->rot/nm, 0.6666666666667 )/JORDRADIUS;
1355  em=1 - Pb->axe*(1 - Pb->exc)/(ram*JORDRADIUS);
1356  if (em < 1E-08) em=1E-08;
1357  templ=j3oj2*(1/ram)*Pb->sini/(1 - em*em);
1358  axnl=em*cos(perm);
1359  aynl=em*sin(perm) - 0.5*templ;
1360  elong=sqrt(axnl*axnl + aynl*aynl);
1361  perl=atan2(aynl,axnl);
1362  llong=lm-0.25*templ*axnl*(3 + 5*Pb->cosi)/(1 + Pb->cosi);
1363  rmida=llong-perl-knum;
1364  e=rmida + elong*sin(rmida);
1365  do
1366  {
1367   rmidx=e - elong*sin(e);
1368   e=e + (rmida - rmidx)/(1 - elong*cos(e));
1369  } while (fabs(rmida - rmidx) > 1E-08);
1370  trueu=2*atan( sqrt((1+elong)/(1-elong))*tan(e/2)) + perl;
1371  rmag=ram*(1 - elong*cos(e));
1372  temps=0.25*J2*kvadrat(1/(ram*(1 - kvadrat(elong))));
1373  sin2u=sin(2*trueu);
1374  cos2u=cos(2*trueu);
1375  rmag=( rmag + temps*kvadrat(Pb->sini)*cos2u*(ram*(1 - kvadrat(elong))) );
1376  trueu=trueu - 0.5*temps*(6 - 7*kvadrat(Pb->sini))*sin2u;
1377  knus=knum + 3*temps*Pb->cosi*sin2u;
1378 if (line+pixel==0.0) printf("dt %f lo %f llong %f\n" , dt,lo,llong);
1379  sinu=sin(trueu);
1380  cosu=cos(trueu);
1381  ux=(pixel-1024.5)/1024.5*Pb->wm;
1382 //printf("Rmag %lf %lf\n",rmag,rmag*sin(ux));
1383  arg=rmag*sin(ux);
1384  if (arg > 1.0) arg=1.0;
1385  if (arg <-1.0) arg=-1.0;
1386  ux=asin(arg) - ux;
1387  sinux=sin(ux);
1388  cosux=cos(ux);
1389 //printf("ux %lf\n",ux*GR);
1390  ra=atan2(Pb->cosi*cosux*sinu-Pb->sini*sinux, cosux*cosu)+knus;
1391  de=asin(Pb->sini*cosux*sinu+Pb->cosi*sinux);
1392  *longi=sideral_time(t,0.0)-ra;
1393  *longi=fmod(TOPI+*longi,TOPI);
1394  if (*longi>M_PI) *longi=*longi-TOPI;
1395  *lati=de + 3.3451E-3*sin(2*de)/rmag;
1396 //printf("lb %f %f\n",*longi*GR, *lati*GR);
1397 }
1398 /*FUNCTION*/
1399 void vduxy2lonlat(int elem, int scan, float *longi, float *lati,
1400         struct kort *Pg)
1401 {
1402 /*foelgende forudsaettes kendt: r  ("jordradius", afhaenger af maalestokken)
1403              ycntr ( = 2*r*tan(45- midlati/2) hvor
1404              midlati er bredde for billedcntre)
1405              midlong ( laengde i billedcntre)
1406              midx,midy (punkt,lin for billedcnter) */
```

```
1407 float x,y,rho;
1408
1409  x=elem - midx;
1410  y=scan + Pg->ycntr - midy;
1411  rho=sqrt(x*x + y*y);
1412  if (rho > 1E-05)
1413  {
1414   *longi=fmod(TOPI + Pg->midlong - atan2(x,y), TOPI);
1415   if (*longi > M_PI) *longi-=TOPI;
1416   *lati=retvinkel - 2*atan(rho/2/Pg->r);
1417  }
1418  else
1419  {
1420   *longi=Pg->midlong;
1421   *lati=retvinkel;
1422  }
1423 }
1424 /*FUNCTION*/
1425 void lonlat2vduxy (float longi, float lati, int *elem, int *scan, struct kort *Pg)
1426 {
1427
1428  float x,y,rho;
1429  float rlongi; //longitude relative to the used koordinate system - ie. angel from center of m
    ap
1430
1431  rlongi = longi-Pg->midlong;
1432  rho = 2*Pg->r*tan((retvinkel-lati)/2);
1433  x = -rho*sin(rlongi);
1434  y = rho*cos(rlongi);
1435  *elem = (int)(x+midx);
1436  *scan = (int)(y+midy-Pg->ycntr);
1437 }
1438
1439
1440 /*FUNCTION*/
1441 void MinMax(int a, int b, int c, int d, int *minimum, int *maximum)
1442 {
1443  int mini,maxi;
1444  mini=a;
1445  if (b<mini) mini=b;
1446  if (c<mini) mini=c;
1447  if (d<mini) mini=d;
1448  maxi=a;
1449  if (b>maxi) maxi=b;
1450  if (c>maxi) maxi=c;
1451  if (d>maxi) maxi=d;
1452  *minimum= mini;
1453  *maximum= maxi;
1454 }
1455 /*FUNCTION*/
1456 int interpoler(int i0, int j0, float bp, float lp, float grid[18][46][2],
1457           float *pixelnummer, float *linjenummer)
1458 {
1459  /*
1460     bp,lp: det soegte punkts bredde og laengde.
1461     p: de fire graensepunkter, p[0][]=b0  p[0][1]=l0 etc.
1462     Punktnummerering:
1463
1464        X (b0,l0)                        X (b1,l1)
1465
1466
1467       x (bp,lp)
1468
1469        X (b3,l3)                        X (b2,l2)
1470
1471     koo[0]: x    koo[1]: y.
1472
1473  */
1474
1475  float lv,lh,bv,bh,l,b,x,y,dldx,dldy,dbdx,dbdy,D,dx,dy,dl,db,p[4][2];
1476  int j;
1477
1478  if (i0 < -9000 || j0 < -9000)
```

```
1479  {
1480
1481  *pixelnummer =-9999.0;
1482  *linjenummer =-9999.0;
1483  return (FALSE);
1484  }
1485
1486  p[0][0]=grid[i0][j0][0];
1487  p[0][1]=grid[i0][j0][1];
1488  p[0][0]=grid[i0][j0][0];
1489  p[0][1]=grid[i0][j0][1];
1490  p[1][0]=grid[i0+1][j0][0];
1491  p[1][1]=grid[i0+1][j0][1];
1492  p[2][0]=grid[i0+1][j0+1][0];
1493  p[2][1]=grid[i0+1][j0+1][1];
1494  p[3][0]=grid[i0][j0+1][0];
1495  p[3][1]=grid[i0][j0+1][1];
1496
1497  x=0.5;
1498  y=0.5;
1499
1500 // printf("Inde %d\n",indenfor(bp,lp,p));
1501  for (j=0; j<10; j++)
1502  {
1503   lv=p[0][1] + y*(p[3][1]-p[0][1]);
1504   bv=p[0][0] + y*(p[3][0]-p[0][0]);
1505   lh=p[1][1] + y*(p[2][1]-p[1][1]);
1506   bh=p[1][0] + y*(p[2][0]-p[1][0]);
1507   l=lv + x*(lh-lv);
1508   b=bv + x*(bh-bv);
1509   db=bp-b;
1510   dl=lp-l;
1511
1512   dldx=p[1][1] + y*(p[2][1]-p[1][1]) - p[0][1] - y*(p[3][1]-p[0][1]);
1513   dldy=p[3][1] - p[0][1] + x*(p[2][1]+p[0][1] - p[1][1]-p[3][1]);
1514   dbdx=p[1][0] + y*(p[2][0]-p[1][0]) - p[0][0] - y*(p[3][0]-p[0][0]);
1515   dbdy=p[3][0] - p[0][0] + x*(p[2][0]+p[0][0] - p[1][0]-p[3][0]);
1516
1517   D=dldx*dbdy - dbdx*dldy;
1518   dx=(dl*dbdy - db*dldy)/D;
1519   dy=(dldx*db - dbdx*dl)/D;
1520   x+=dx;
1521   y+=dy;
1522  // printf("dx %10.4f  dy %10.4f\n",dx,dy);
1523   if (fabs(dx) < 0.01 && fabs(dy) < 0.01) break;
1524  } //for (j...
1525
1526  if (j>8) return (FALSE); /*Ingen konvergens*/
1527
1528  *pixelnummer =(i0 + x)*128.0-64.0;
1529  *linjenummer =(j0 + y)*128.0-128.0;
1530 //printf("x %f  y %f  it %d\n",x,y,j);
1531  return (TRUE);
1532 }
1533 /*FUNCTION*/
1534 int indenfor(float bp, float lp, float p[4][2])
1535 {
1536  float r,s,t,u,x,y;
1537
1538  x=lp-p[0][1];
1539  y=bp-p[0][0];
1540  r=-((p[3][1]-p[0][1])*y - (p[3][0]-p[0][0])*x);
1541  s=(p[1][1]-p[0][1])*y - (p[1][0]-p[0][0])*x;
1542  x=lp-p[2][1];
1543  y=bp-p[2][0];
1544  t=(p[3][1]-p[2][1])*y - (p[3][0]-p[2][0])*x;
1545  u=-((p[1][1]-p[2][1])*y - (p[1][0]-p[2][0])*x);
1546
1547  if (r<0.0 || s<0.0 || t<0.0 || u<0.0) return (0);
1548  return (1);
1549 }
1550 /*FUNCTION*/
1551 int seek(float la, float lo, float grid[18][46][2], int m, int *p, int *l)
```

```
1552 {
1553  int i,j;
1554  float d,pnt[4][2];
1555
1556  for (i=0; i<17; i++)
1557  for (j=0; j<m ; j++)
1558  {
1559   pnt[0][0]=grid[  i][  j][0];
1560   pnt[0][1]=grid[i  ][  j][1];
1561   pnt[1][0]=grid[i+1][  j][0];
1562   pnt[1][1]=grid[i+1][  j][1];
1563   pnt[2][0]=grid[i+1][j+1][0];
1564   pnt[2][1]=grid[i+1][j+1][1];
1565   pnt[3][0]=grid[i  ][j+1][0];
1566   pnt[3][1]=grid[i  ][j+1][1];
1567
1568   if (indenfor(la, lo, pnt))
1569   {
1570    *p=i;
1571    *l=j;
1572    return (TRUE);
1573   }
1574  }
1575  *p=-9999;
1576  *l=-9999;
1577  return (FALSE);;
1578 }
1579
1580 /*FUNCTION*/
1581 int drawkonturs(BYTE skabelon[midy*2][midx*2], struct kort *Pg,struct ini_info *ini)
1582 {
1583  char laes[25],filename[100];
1584  int size,i,j,x,y;
1585  float l,b;
1586  FILE *fpind2;
1587
1588 //Template array nulstilles
1589  for (i=0;i<midy*2 ;i++)
1590  for (j=0;j<midx*6 ;j++)
1591  {
1592    skabelon[i][j] = 255;
1593  }
1594  //open europa.asc
1595  strcpy(filename,ini->helpdir);
1596  strncat(filename,"europa.asc",10);
1597  fpind2=fopen(filename,"rb");
1598  if (fpind2 == NULL)
1599  {
1600    printf("europa.asc findes ikke!\n" );
1601    return (TRUE);
1602  }
1603  do
1604  {
1605    if (feof(fpind2)) break;
1606    fgets(laes,22,fpind2);
1607    sscanf(laes,"%f %f",&l,&b);
1608    if (l != -999.00)
1609    {
1610       l=(l+0.5)/GR;
1611       b=(b-1.7)/GR;
1612       lonlat2vduxy(l,b,&x,&y,Pg);
1613       if ((x<midx*2 )&(x>0 )&(y<midy*2 )&(y>0 ))
1614       {
1615          skabelon[midy*2-y-1][x]=2;
1616          //printf("D");
1617       }
1618    }
1619  } while (!feof(fpind2));
1620  fclose(fpind2);
1621 }
1622
1623
1624 /*FUNCTION*/
```

```
1625 int get_ini(struct ini_info *ini)
1626 {
1627  FILE *fpini;
1628  char reader[500];
1629  int l,b,top;
1630
1631  fpini=fopen("/home/radman/NOAAFiles/Kompi6.ini" ,"rt");
1632  if (fpini == NULL) return (FALSE);
1633  do
1634  {
1635    if (feof(fpini)) break;
1636    fgets(reader,500,fpini);
1637    if (strstr(reader,"#") == NULL)
1638    {
1639          sscanf(reader,"DFUID: %s outdir: %s helpdir: %s satdir: %s midx: %d midy: %d lmid: %i
     bmid: %i mtop: %i port: %d",&ini->DFUID,&ini->outdir,&ini->helpdir,&ini->satdir,&ini->tmidx,&
    ini->tmidy,&l,&b,&top,&ini->port);
1640       ini->tlmid = l;
1641       ini->tbmid = b;
1642       ini->tbtop = top;
1643    //printf("l: %i, b: %i, top: %i ",l,b,top);
1644          if (ini->tmidx==400 ) break;
1645    }
1646  } while (!feof(fpini));
1647
1648  fclose(fpini);
1649  return(TRUE);
1650 }
1651
1652 /*FUNCTION*/
1653 int make_label (BYTE labelArray[50][600], struct ini_info *ini,char labelstring[200])
1654 {
1655  FILE *fpfont;
1656  char filename[100];
1657  BYTE reader[3500];
1658  int found = FALSE;
1659  int bytecount,curser,lcurser,line,pixel,sign[10][6],signswritten,test,test2;
1660
1661
1662  for (line=0;line<50;line++)
1663  for (pixel=0;pixel<600;pixel++)
1664    labelArray[line][pixel] = 255;
1665  strcpy(filename,ini->helpdir);
1666  strncat(filename,"fonte",5);
1667  fpfont=fopen(filename,"rb");
1668  if (fpfont == NULL)
1669  {
1670    printf("Error opening fonte " );
1671    printf(filename);
1672    return (FALSE);
1673  }
1674    curser=lcurser= 10;
1675    signswritten = 0;
1676    line=pixel=bytecount=0;
1677
1678    fread(reader,3500,1,fpfont);
1679    fclose(fpfont);
1680    printf(labelstring);
1681  while (signswritten<strlen(labelstring))
1682  {
1683    if(labelstring[signswritten]=='\n')
1684       {
1685          lcurser+=15;
1686          curser=10;
1687          signswritten++;
1688       }
1689    found=FALSE;
1690    bytecount = 0;
1691    do
1692    {
1693       do
1694       {
1695          bytecount++;
```

```
1696        } while (reader[bytecount]!=2);
1697        bytecount++;
1698        if (reader[bytecount]==labelstring[signswritten]) found = TRUE;
1699        //printf("Sign: %c %i %i %i",labelstring[signswritten],bytecount,found,(!found)&(bytecou
    nt<sizeof(reader)));
1700     } while ((!found)&(bytecount<sizeof(reader)) );
1701     if (!found) return(TRUE);
1702     bytecount++;
1703     while ((reader[bytecount]==255)|(reader[bytecount]==00))
1704     {
1705        while ((reader[bytecount]==255)|(reader[bytecount]==00))
1706        {
1707           labelArray[lcurser+line][(curser+pixel)*3]=reader[bytecount];
1708           labelArray[lcurser+line][(curser+pixel)*3+1]=reader[bytecount];
1709           labelArray[lcurser+line][(curser+pixel)*3+2]=reader[bytecount];
1710           test=(curser+pixel)*3;
1711           test2=lcurser+line;
1712           bytecount++;
1713           pixel++;
1714        }
1715        if (reader[bytecount]== 13)
1716        {
1717           bytecount++;
1718           line++;
1719           pixel=0;
1720        }
1721     }
1722     if (reader[bytecount]== 12)
1723     {
1724        //printf("PIXEL: %i ",pixel);
1725        curser = curser+pixel;
1726        signswritten++;
1727        line=0;
1728        pixel=0;
1729     }
1730     else //error - sign should be deleted and not found returned!
1731     {
1732        found=FALSE;
1733     }
1734  }
1735  return(FALSE);
1736 }
1737
1738
1739  /*FUNCTION*/
1740  int synccheck(BYTE AUXSYNC[AUXSYNC_LNTH])
1741  {
1742    BYTE output[1001],bit[10];
1743    int i,errors;
1744    errors=0;
1745    for (i=0;i<10;i++) bit[i] = 1;
1746    for (i=0;i<1001;i++){
1747    output[i] = bit[9];
1748        bit[9]   =  bit[8];
1749        bit[8]   =  bit[7];
1750        bit[7]   =  bit[6];
1751        bit[6]   =  bit[5];
1752        bit[5]   =  bit[4]^output[i];
1753        bit[4]   =  bit[3];
1754        bit[3]   =  bit[2];
1755        bit[2]   =  bit[1]^output[i];
1756        bit[1]   =  bit[0]^output[i];
1757        bit[0]   =  output[i];
1758    }
1759    /* for (i=0;i<80;i++){
1760    output[i] =  bit[6];
1761        bit[6]   =  bit[5]^output[i];
1762        bit[5]   =  bit[4];
1763        bit[4]   =  bit[3];
1764        bit[3]   =  bit[2]^output[i];
1765        bit[2]   =  bit[1]^output[i];
1766    bit[1]   =  output[i];
1767    }*/
```

```
1768
1769    for (i=0;i<AUXSYNC_LNTH -1;i++) //tester kun til naestsidste byte idet sidste byte er halvt
     fyldt med garbage
1770    {
1771        if((AUXSYNC[i]&128)>>7 != output[i*8+4]) { errors++; } //starter med bit 4 og ikke bit 0
     fordi de foerste fire bit
1772        if((AUXSYNC[i]&64)>>6 != output[i*8+5]){ errors++; }  //ikke er med i AUXSYNC[]
1773        if((AUXSYNC[i]&32)>>5 != output[i*8+6]) { errors++; }
1774        if((AUXSYNC[i]&16)>>4 != output[i*8+7]) { errors++; }
1775        if((AUXSYNC[i]&8)>>3 != output[i*8+8]) { errors++; }
1776        if((AUXSYNC[i]&4)>>2 != output[i*8+9]) { errors++;}
1777        if((AUXSYNC[i]&2)>>1 != output[i*8+10]) { errors++;}
1778        if((AUXSYNC[i]&1) != output[i*8+11]) { errors++;}
1779    }
1780    return(errors);
1781  }
```

13.6 Delpfi testprogram af DPU

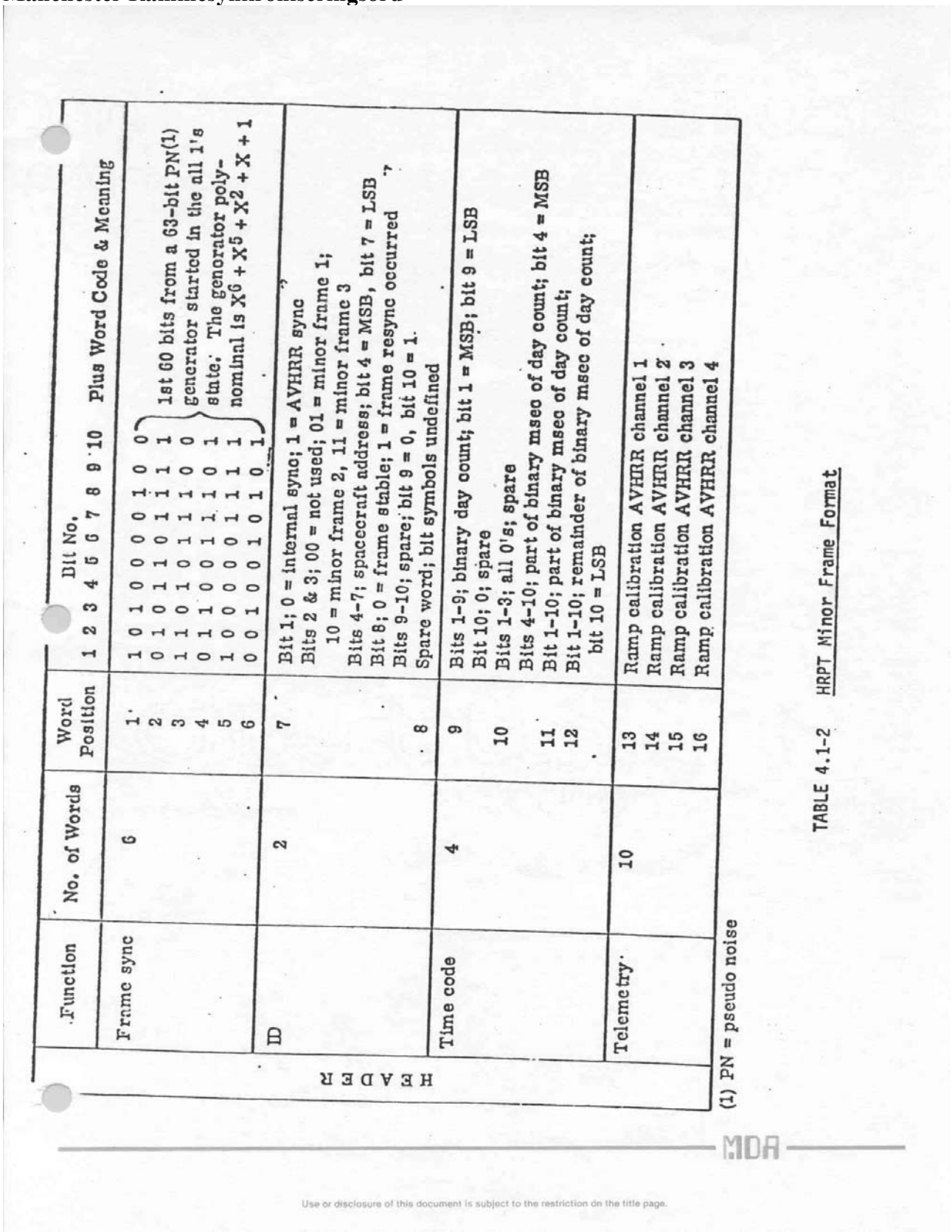
```
 1: unit main;
 2: interface
 3: uses
 4:   Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
 5:   StdCtrls, ScktComp;
 6: type
 7:   TMainForm = class(TForm)
 8:     myClientSocket: TClientSocket;
 9:     PortEdit: TEdit;
10:     SendButton: TButton;
11:     AddressEdit: TEdit;
12:     DataMemo: TMemo;
13:     TypeEdit: TEdit;
14:     PacketNbEdit: TEdit;
15:     LengthEdit: TEdit;
16:     LabelType: TLabel;
17:     LabelPacketNb: TLabel;
18:     LabelLength: TLabel;
19:     PackToSendEdit: TEdit;
20:     Label1: TLabel;
21:     Label2: TLabel;
22:     Label3: TLabel;
23:     CheckBoxDFUID: TCheckBox;
24:     CheckBoxEndNOAA: TCheckBox;
25:     procedure SendButtonClick(Sender: TObject);
26:     procedure myClientSocketConnect(Sender: TObject;
27:       Socket: TCustomWinSocket);
28:   private{ Private declarations }
29:   public{ Public declarations }
30:   end;
31: var
32:   MainForm: TMainForm;
33: implementation
34: {$R *.DFM}
35:
36: procedure TMainForm.SendButtonClick(Sender: TObject);
37: {********** this procedure is called when Send button is pressed *************}
38: begin
39:    myClientSocket.Active := False; {disconnect before setting properties}
40:    myClientSocket.Address := AddressEdit.Text;
41:    myClientSocket.Port := StrToIntDef(PortEdit.Text,6789);{default port:6789}
42:    myClientSocket.Active := True; {connect .. this will call procedure below}
43: end;
44:
45: {*****************************************************************************}
46: {********** Procudure below is fired when client connects to server ***********}
47: {*****************************************************************************}
48: procedure TMainForm.myClientSocketConnect(Sender: TObject;
49:   Socket: TCustomWinSocket);
50: var
51: TypeOfData : Char;
52: m,LengthOfData :Integer;
53: TextToSend : string;
54: begin
55:    LengthOfData :=0;
56: {convert 'Type of data' to integer}
57:    TypeOfData := Chr( StrToInt(TypeEdit.Text) );
58: {get number of chars in in memebox, thus 'Length Of Data'}
59:    For m:=0 TO DataMemo.Lines.Count do
60:       LengthOfData := LengthOfData +Length( DataMemo.Lines.Strings[m] );
61:
62: {update GUI with 'Length Of Data'}
63:    LengthEdit.Text := IntToStr(LengthOfData);
64:
65: { ****************Now we send Data to the server ***************************}
66: { Send presentation }
67:    if (CheckBoxDFUID.Checked = True) then begin
68:       myClientSocket.Socket.SendText(Chr(2)+  {type = 2}
69:                                      Chr(1) +  {upper byte of Packet Nb.}
70:                                      Chr(255)+ {lower byte of Packet Nb.}
71:                                      Chr(0)+   {upper byte of Length}
72:                                      Chr(16)+   {lower byte of length}
73:                                      '0030556802b7' );
74:       Sleep(50); {better give API some time ...}
75:    end; {end if}
76:
```

```
 77:
 78: { send datapackets, amount defined by 'PackToSend'-Edit }
 79:    TextToSend := '';
 80:    for m:=0 to DataMemo.Lines.Count - 1 do begin
 81:       TextToSend := TextToSend + DataMemo.Lines.Strings[m];
 82:    end; {end for}
 83:
 84:
 85:    For m:=1 TO StrToIntDef(PackToSendEdit.Text,1) do begin
 86:       myClientSocket.Socket.SendText(TypeOfData + {type (from Edit-field)}
 87:                 Chr(1)+                 {upper byte of Packet Nb.}
 88:                 Chr(255)+               {lower byte of Packet Nb.}
 89:                 Chr((LengthOfData AND $FF00) SHR 8)+ {upper byte of Length}
 90:                 Chr(LengthOfData AND $FF)+           {lower byte of Length}
 91:                 TextToSend);
 92:                 //DataMemo.Lines.Strings[DataMemo.Lines.Count - 1]); {data}
 93:      Sleep(5); {better give API some time ...}
 94:    end;{end For-loop}
 95:   // showmessage( TextToSend );
 96: { send the 'final' data packet (defined by value in type-field) }
 97:
 98:    if (CheckBoxEndNOAA.Checked = True) then begin
 99:       myClientSocket.Socket.SendText(Chr(32)+
100:                                      Chr(1) +
101:                                      Chr(255)+
102:                                      Chr(0) +
103:                                      Chr(9)+
104:                                      '012345678' );
105:    end;
106:    showmessage('All data send!');
107: end;
108:
109: end.
```

13.7 NOAA-frame

25.4. Bilag 5

Manchester Rammesynkroniseringsord



	Function	No. of Words	Word Position	Bit No. 1 2 3 4 5 6 7 8 9 10	Plus Word Code & Meaning
HEADER	Frame sync	6	1	1 0 1 0 0 0 0 1 0 0	1st 60 bits from a 63-bit PN(1) generator started in the all 1's state. The genorator polynominal is $X^6 + X^5 + X^2 + X + 1$
			2	0 1 0 1 1 0 1 1 1 1	
			3	1 1 0 1 0 1 1 1 0 0	
			4	0 1 1 0 0 1 1 1 0 1	
			5	1 0 0 0 0 0 1 1 1 1	
			6	0 0 1 0 0 1 0 1 0 1	
	ID	2	7		Bit 1; 0 = internal sync; 1 = AVHRR sync Bits 2 & 3; 00 = not used; 01 = minor frame 1; 10 = minor frame 2, 11 = minor frame 3 Bits 4-7; spacecraft address; bit 4 = MSB, bit 7 = LSB Bit 8; 0 = frame stable; 1 = frame resync occurred Bits 9-10; spare; bit 9 = 0, bit 10 = 1. Spare word; bit symbols undefined
			8		
	Time code	4	9		Bits 1-9; binary day count; bit 1 = MSB; bit 9 = LSB Bit 10; 0; spare
			10		Bits 1-3; all 0's; spare Bits 4-10; part of binary msec of day count; bit 4 = MSB
			11		Bit 1-10; part of binary msec of day count;
			12		Bit 1-10; remainder of binary msec of day count; bit 10 = LSB
	Telemetry	10	13		Ramp calibration AVHRR channel 1
			14		Ramp calibration AVHRR channel 2
			15		Ramp calibration AVHRR channel 3
			16		Ramp calibration AVHRR channel 4

(1) PN = pseudo noise

TABLE 4.1-2 HRPT Minor Frame Format

Function	No. of Words	Word Position	Bit No. 1 2 3 4 5 6 7 8 9 10 Plus Word Code & Meaning
Telemetry (cont.)	10	17	Ramp calibration AVHRR channel 5
		18	AVHRR channel 3 target temp.(2) } Each of these words is a 5-channel subcom -4 words of IR data plus subcom sync (10 0's)
		19	AVHRR channel 4 target temp.
		20	AVHRR channel 5 target temp.
		21	Channel-3 patch temp.
		22	0 0 0 0 0 0 0 0 0 1 spare
Back scan	30	23	10 words of back scan data from each AVHRR channel 3, 4, and 5. These data are time multiplexed as Chan 3 (word 1), chan 4 (word 1), chan 5 (word 1), chan 3 (word 2), chan 4 (word 2), chan 5 (word 2), etc.
		↓	
		52	
Space data	50	53	10 words of space-scan data from each AVHRR channel 1, 2, 3, 4, and 5. These data are time multiplexed as chan 1 (word 1), chan 2 (word 1), chan 3 (word 1), chan 4 (word 1), chan 5 (word 1), chan 1 (word 2), chan 2 (word 2), chan 3 (word 2), chan 4 (word 2), chan 5 (word 2), etc.
		↓	
		102	
Sync Δ	1	103	Bit 1; 0 = AVHRR sync early; 1 = AVHRR sync late Bits 2-10; 9-bit binary count of 0.9984-MHz periods; bit 2 = MSB, bit 10 = LSB

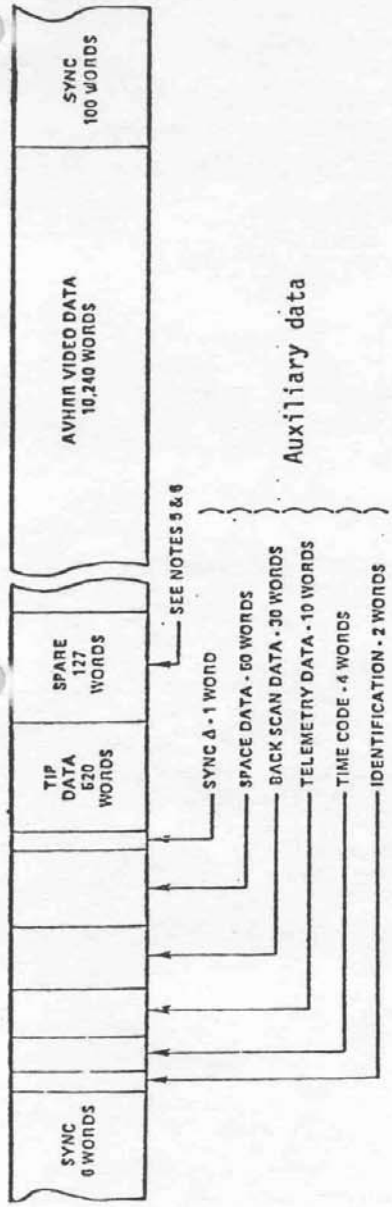
TABLE 4.1-2 Continued

Function	No. of Words	Word Position	Bit No. 1 2 3 4 5 6 7 8 9 10	Plus Word Code & Meaning
Tip data	520	104		The 520 words contain four frames of TIP data (104 TIP data words/frame)
				Bits 1-8: exact format as generated by TIP
				Bit 9: even parity check over bits 1-8
		623		Bit 10: - bit 1
Spare words	127	624	1 0 1 0 0 0 1 1 1 0	Derived by inverting the output of a 1023-bit PN sequence provided by a feedback shift register generating the polynominal: $X^{10} + X^5 + X^2 + X + 1$ The generator is started in the 1's state at the beginning of word 7 of each minor frame.
		625	1 1 1 0 0 0 1 0 1 1	
		626	0 0 0 0 1 0 1 1 1 1	
		627	1 0 1 1 0 0 0 1 1 1	
		628	1 1 0 1 0 1 0 0 1 0	
		748	1 0 0 1 0 1 1 0 1 0	
		749	1 1 0 0 1 0 0 0 1 0	
		750	1 0 0 0 1 0 0 0 0 0	

TABLE 4.1-2 Continued

Function	No. of Words	Word Position	Bit No. 1 2 3 4 5 6 7 8 9 10	Plus Word Code & Meaning
Earth data	10,240	751	Chan 1 - Sample 1	Each minor frame contains the data obtained during one earth scan of the AVHRR sensor. The data from the five sensor channels of the AVHRR are time multiplexed as indicated
		752	Chan 2 - Sample 1	
		753	Chan 3 - Sample 1	
		754	Chan 4 - Sample 1	
		755	Chan 5 - Sample 1	
		756	Chan 1 - Sample 2	
		↓	↓	
		10,985	Chan 5 - Sample 2047	
		10,986	Chan 1 - Sample 2048	
		10,987	Chan 2 - Sample 2048	
		10,988	Chan 3 - Sample 2048	
		10,989	Chan 4 - Sample 2048	
		10,990	Chan 5 - Sample 2048	
Auxiliary sync	100	10,991	1 1 1 1 1 0 0 0 1 0	Derived from the non-inverted output of a 1023-bit PN sequence provided by a feedback shift register generating the polynominal: $X^{10}+X^5+X^2+X+1$ The generator is started in the all 1's state at the beginning of word 10,991
		10,992	1 1 1 1 1 1 0 0 1 1	
		10,993	0 1 1 0 1 1 0 1 0 1	
		10,994	1 0 1 0 1 1 1 1 0 1	
		↓	↓	
		11,089	0 1 1 1 1 1 0 0 0 0	
		11,090	1 1 1 1 0 0 1 1 0 0	

TABLE 4.1-2 Continued



NOTES:
(1) MINOR FRAME LENGTH - 11,090 WORDS
(2) THREE MINOR FRAMES PER MAJOR FRAME
(3) MINOR FRAME RATE - 6 FRAMES/SECOND
(4) WORD LENGTH - 10 BITS/WORD

(5) HRPT OUTPUT - ALL SPARES ARE 10TH DEGREE P-N CODE (BAR).
(6) IF A FOURTH SOUNDING INSTRUMENT IS ADDED, THESE SPARE WORD SLOTS WILL MOST LIKELY BE USED FOR DATA FROM THIS INSTRUMENT.

TLM WORD ALLOCATIONS		ID WORD BIT ALLOCATIONS			
		1ST ID WORD		2ND ID WORD	
1-5	RAMP CALIBRATION	1	SYNC ID	(SPARE)	
6	CHANNEL-3 TARGET TEMP (5 PT SUBCOM)	2-3	FRAME ID		
7	CHANNEL-4 TARGET TEMP (5 PT SUBCOM)	4-7	SPACECRAFT ADDRESS		
8	CHANNEL-5 TARGET TEMP (5 PT SUBCOM)	8	RESYNC MARKER		
9	CHANNEL-3 PATCH TEMP	9	DATA 0	1-9	DATA 0
10	SPARE	10	DATA 1	10	DATA 1

FIGURE 4.1-2 TIROS-N HRPT Frame Format

13.8 DPUens inifil

```
#
# ini file for NOAA-DPU-vrs 6.7
#
# this is a one-line char format
#
# read this with scanf-format:
#   "DFUID: %s outdir: %s helpdir: %s satdir: %s pictw: %d picth %d lmid %d bmid
%d mtop %d port: %d\n"
#
#
#DFUID: 0030556802B7
#Aktivt Parametersaet (standart billedudsnit):
DFUID: 0030556802B7 outdir: /home/radman/NOAAFiles/29jun/ helpdir:
/home/radman/NOAAFiles/help/ satdir: /home/radman/NOAAFiles/sat/ midx: 400 midy:
300 lmid: 20 bmid: 65 mtop: 95 port: 6796 kanaler: 1 2 4
#udkommaterede parametersaet (andre billedudsnit):
#DFUID: 0030556802B7 outdir: /home/radman/NOAAFiles/29jun/ helpdir:
/home/radman/NOAAFiles/help/ satdir: /home/radman/NOAAFiles/sat/ midx: 400 midy:
300 lmid: 5 bmid: 52 mtop: 54 port: 6789 kanaler: 1 2 4
#DFUID: 0030556802b7 outdir: /home/radman/NOAAFiles/29jun/ helpdir:
/home/radman/NOAAFiles/help/ satdir: /home/radman/NOAAFiles/sat/ midx: 400 midy:
300 lmid: 35 bmid: 37 mtop: 39 port: 6789 kanaler: 1 2 4
#DFUID: 0030556802b7 outdir: /home/radman/NOAAFiles/29jun/ helpdir:
/home/radman/NOAAFiles/help/ satdir: /home/radman/NOAAFiles/sat/ midx: 400 midy:
300 lmid: -22 bmid: 36 mtop: 38 port: 6789 kanaler: 1 2 4
```

13.9 Eksempler på info-fil som output fra DPU

Sattelite: NOAA18, Year:6, Day:222, 40635078
Lines in passage: 1020
biterrors per bit: 0.025707

Sattelite: NOAA18, Year:6, Day:222, 39911578
Lines in passage: 5336
biterrors per bit: 0.488642