Tittel:	Videointerface to rocket PCM encoder	NR. :
		Dato :
Deltaker(e):	Morgan Trøvolt, 98HINED	
	Frode Nysted, 97HINED	
Veileder(e):	Vegar Johansen	
Oppdragsgiver:	Forsvarets Forskningsinstitutt	
Kontaktperson:	Terje Angeltveit	
Stikkord: (4 stk.)	MPEG komprimering, JPEG komprimering, dataflyt, Komprimeringsteknikker	
Antall sider: 100	Antall bilag: 5	Tilgjengelighet(åpen/konfidensiell): Åpen

Forsvarets forskningsinstitutt har i flere år jobbet med ubemannede overvåkningsfly. Det siste prosjektet er en rakett som skal kunne sende levende bilder i til bakken.

Oppgaven bestod i å lage en krets som mottar standard PAL signal fra et kamera, for så å digitalisere og komprimere den datastrømmen. Deretter skulle den komprimerte datastrømmen sendes videre til FFI sin nyutviklede PCM encoder, som sender data til bakken. Enheten skulle kunne komprimere både MPEG video format og JPEG bildeformat med varierende datarater. Det var også ønskelig at kretsen skulle tåle minst 70°C og bruke lite strøm. Det var også ønskelig å legge til rette for eventuell utvidelse av PCM encoderen slik at man kunne motta kontrollsignaler fra kontrollstasjon på bakken, slik at man kunne forandre datarater og innstillinger i luften.

Rapporten beskriver en nesten fullstendig løsning av oppgaven. Vi fikk aldri tak i brikken vi valgte å bygge vår løsning rundt, så vi har ikke utviklet et fullstendig kretskortutlegg. Vi har heller fokusert på å presentere forskjellig løsninger på forskjellige problemer, og foretatt et eget valg på de forskjellige stedene. Vi har så utredet hvorfor vi gjorde de valgene vi gjorde, og gjennom hele prosessen så forklarer vi detaljert hvordan vi valgte å løse de forskjellige problemstillingene.

Den største delen av oppgaven går på beskrivelse av hvordan signalene vil bevege seg i kretsen, og hvordan vi bruker en mikrokontroller til å sørge for at mottakeren på bakken både mottar feilmeldinger og data på ett lett leselig format.

Vi bemerker at vi ikke har hatt muligheten til å teste systemet, men vi mener at det vi har gjort vil fungere dersom det lages og kobles opp slik koblingsskjema viser.

Forord

Denne rapporten er et resultat av vårt arbeid med hovedprosjekt ved linjen Elektronikk/Mikrodata ved Høgskolen i Gjøvik.

Hovedprosjekt oppgaven er gitt av Forsvarets Forskningsinstitutt på Kjeller. Oppgaven er ment å være en nyutvikling av et lite kretskort som skal kunne motta et videosignal og sende det serielt videre.

Vi retter en takk til veilederen vår ved Avdeling Teknologi på Høgskolen i Gjøvik, Vegar Johansen, og til kontaktperson ved FFI, Terje Angeltveit, for å alltid ha tid til å svare på de spørsmål vi har hatt.

Jeg vil også benytte anledningen til å takke Øystein Skar ved Winbond sin avdeling i Norge. Øystein Skar jobbet hardt for å skaffe oss den hardwaren vi trengte, og klarte mot alle odds å skaffe oss et datablad over kretsen vi hadde valgt å bruke. Dessverre er dette databladet merket ”CONFIDENTIAL”, så vi har ikke lov til å legge det ved denne rapporten. Vi henviser til Winbond sin avdeling i Norge for å få tak i dette.

Gjøvik, 22.Mai 2001

---	---
Morgan Tørvolt	Frode Nysted

1.0 INNLEDNING **4**

2.0 MPEG OG JPEG. **5**

2.0.1 Hva er MPEG. *5*
2.0.2 Oversikt over de forskjellige MPEG standardene. *5*
MPEG-1 5
MPEG-2 5
MPEG-3 5
MPEG-5 & MPEG-6 6
MPEG-7 & MPEG-21 6
2.0.3 Hva er JPEG *7*

2.1 MPEG VIDEO KOMPRIMERING **8**

2.1.1 Konvertering RGB til YCbCr *8*
2.1.2 Frame og Field *9*
2.1.3 Block og Macroblock *10*
2.1.4 Slice og Picture *11*
2.1.5 GOP og Sequence *12*
2.1.6 Motion Estimation & Compensation. *13*
2.1.7 Diskret Cosinus Transform (DCT) *14*
2.1.8 Video Buffer Verifier (VBV) *14*

3.0 FORSKJELLIGE IC KRETSER OG LØSNINGER. **15**

3.1 AKTUELLE KRETSER FOR MPEG1/2 OG JPEG 16
3.1.1 IBM *17*
3.1.2 Texas Instruments *18*
3.2.3 VisionTech *18*
3.2 VALGT MPEG KRETS 19
3.3 Resterende komponenter *21*

4.0 SYSTEMBESKRIVELSE **22**

4.1 DATAFLYT 23
4.1.1 Mulige dataflytmodeller. *23*
4.1.1.1 Parallell til seriell bro 23
4.1.1.2 Parallell til Parallell (MUX) bro 24
4.1.1.3 Styring av datastrøm 24
4.1.2 De viktigste signalegenskapene til brikkene *25*
4.1.2.1 Dataflyt fra MPEG encoderen. 25
4.1.2.2 Dataflyt fra og til MUX 25
4.1.2.3 Dataflyt til PCM encoderen 25
4.1.2.4 I2C-bus (mellom video dekoder og MPEG encoder) 25
Beskrivelse av I2C busen 25
Dataformat mellom video dekoder og MPEG encoder 28
4.2 TEKNISK LØSNING 29
4.2.1 Signalgang *29*
4.2.2 Kobingssjema *32*

4.3 KONFIGURERING 33
4.3.1 Bestemme datarate *34*
4.3.2 Metoder for å forandre datarate fra MPEG encoder *37*
Forandre oppløsning 37
Forandre bildemønsteret 37
Forandre bildekvalitet 38
4.3.3 Konfigurering av video dekoder *39*
4.3.4 Konfigurering av MPEG encoder *40*
4.3.4.1 Moduser 40
4.3.4.2 Start prosedyre 42
4.3.4.3 Live video encoding 42
4.3.4.4 Single frame grab 45
4.3.4.4 Single frame encoding 46
4.3.4.5 Register-innstilling 47
Vdata_out (0x00) 48
I2c_data (0x02) 48
Video_reset (0x04) 49
Vstart (0x05) 49
Vstop (0x06) 49
I2c_start (0x07) 49
I2c_stop (0x08) 49
Vint_enable (0x0C) 49
Vint_source (0x0D) 50
Vint_clear (0x0E) 51
Vint_rd_clear (0x0F) 51
Vtreshold (0x10) 51
Vwork_mode(0x11) 52
Video_format (0x12) 53
Venc_cntl (0x13) 54
Vframe_pattern (0x14) 55
Vbit_rate_m (0x15), Vbit_rate(0x16) 56
Vbv_size (0x17) 57
Vbv_initial (0x18) 57
Vquality (0x19) 58
Vslice_header (0x1A) 59
Vgob_header (0x1B) 60
Vit_hour (0x1C) 61
Vit_minute (0x1D) 61
Vit_second (0x1E) 62
Vit_frame (0x1F) 62
Vin_offset (0x20) 63
Vin_cntl (0x21) 63
Vin_picture (0x22) 64
Vmem_select (0x23) 64
Vsize_h (0x24) 65
Vsize_v (0x25) 65

Vfifo_status (0x30) 65
I2c_status (0x31) 66
PCR (Processor Control Register) (0x66) 66
4.4 PROGRAMTEKNISKE LØSNINGER 68
4.4.1 Pseudokode for oppstart *68*
4.4.2 Pseudokode for live video overføring *69*
4.4.3 Pseudokode for live video snapshot og single frame encoding *70*
4.4.4 Pseudokode for I2C bus overføring *71*
4.4.5 Oversikt over hvordan kildekoden er lagt opp *72*
4.4.6 Funksjonsbeskrivelse av kildekode *73*
4.4.6.1 Hovedoversikt 73
4.4.6.2 Boot sekvens 75
4.4.6.3 Get mode sekvensen 76
4.4.6.4 Behandling av datastrøm 76
4.4.6.5 Send feilmelding 78
5.0 KONKLUSJON 79
6.0 VEDLEGG 81

1.0 Innledning

Kongsberg Defence & Aerospace har utviklet et nytt sjømissil som har gjenbrukbar motor. Forsvarets Forskningsinstitutt har i den anledning hatt ett prosjekt gående, med må å bruke denne missilen som et ubemmanet overvåkningsfartøy. Ved å ta ut spregstoff og sette inn et kamera og en radiosignal sender håper de at dette skal kunne brukes til overvåkning av store områder fra en kontrollsentral. Denne typen farkost kalles for UAV, eller Unmanned Aerial Vehicle.

Siden ukomprimert video tar veldig stor båndbredde, ønsker FFI å benytte seg av noen av dagens kompressjonsmetoder. Vi har derfor fått i oppdrag å lage en krets som komprimerer ett innsignal med både MPEG og JPEG og sende det videre serielt. Man skal i tillegg kunne forandre datarate, og gjene ha mulighet til å kunne sende kontrollsignaler fra bakken for å variere innstillinger og datarater.

Betingelsene vi fikk fra FFI var:

- Video signal: Standard PAL signal. Det samme signalet som brukes i vanlige TV apparater.
- Overføring: Man skal kunne overføre opp imot 10Mbit/s, men også takle lavere datarater.
- Ytre påkjenning: Kretsen bør tåle minst 70°C da det kan bli varmt i tuppen av raketten.
- Mål: Hele kretsen burde få plass på et kretskort som er 10cm*32mm stort.
- Generelt: Kretsen bør ha lite strømforbruk.

Først udtypes de forekjellige komprimeringsteknikkene i kapittel 2. Dete er viktig for å få en forståelse av hvordan man skal velge innstillinger.

I kapittel 3 gjøres det rede for hvilke brikker vi har vurdert, og hvorfor vil gjorde det valget vi gjorde. I tillegg forteller vi hvilke andre ICer vi har benyttet.

I kapittel 4 går vi gjennom hele systemet, og beskriver alt fra koblingssjema og dataflyt, til mikroprosessorkode og konfigurering av kretsene. Vedlegg 6.2 inneholder programlisting.

Vedlegg 6.3 inneholder en liste over henvisninger og web linker.

Vedlegg 6.4 er ei liste over datablad. Vi har lagt alle datablad og produktbeskrivelser under en egen katalog på CDen.

2.0 MPEG og JPEG.

2.0.1 Hva er MPEG.

MPEG (*Moving Pictures Expert Group*), er en prosjektgruppe opprettet av ISO (*the International Standards Organization*) og IEC (*International Electro-technical Commission*), i samarbeid med over 150 forskjellige selskaper og forsknings institutter.

MPEG har som oppgave å utarbeide standarder for komprimering av digitale video og audio data. Hensikt til MPEG er ikke å utarbeide fullstendige løsninger for enkoding og dekoding, men å lage en standard for de komprimerte dataene, og beskrive hvordan disse dataene kan brukes i system applikasjoner.

Arbeidet med den første MPEG standarden, kalt MPEG-1, startet i 1988. De arbeider nå med å standardisere MPEG-7 og MPEG-21.

2.0.2 Oversikt over de forskjellige MPEG standardene.

MPEG-1

- Standardisert i 1992.
- For progressiv video (ikke interlaced).
- Designet for medium båndbredde.
- Komprimert video/audio opp til 4 Mbit/s.
- Optimalisert for CD-ROM, Video-CD og CD-i applikasjoner.

MPEG-2

- Standardisert i 1996.
- Støtter både interlaced og progressiv video.
- Skalerbar koding
- Designet for høyere båndbredde.
- Komprimert video/audio opp til 80 Mbit/s.
- Brukes i digital TV/Satellitt/Kabel, HDTV, og DVD.

MPEG-3

- Tenkt brukt i HDTV.
- Ble droppet, og MPEG-2 ble utvidet til inkludere HDTV.

MPEG-4

- Standardisert i 1999.
- Lav til høy båndbredde.
- Komprimert video/audio fra 10 kbit/s til 10Mbit/s.
- Designet for multimedia applikasjoner, og video.
- Mest kjent for sin ufrivillige? bruk i DivX, som ikke er et godkjent format.

MPEG-5 & MPEG-6

Disse er ikke definert, og MPEG gikk direkte til MPEG-7.

MPEG-7 & MPEG-21

Under utvikling

MPEG-1 til MPEG-2 tar for seg komprimering av video/audio data for lagring på lagrings media og for bruk i digital TV/Satellitt/kabel, MPEG-4 beskriver bruk av MPEG i enkle multimedia applikasjoner.

MPEG-7 og MPEG-21 går videre, og beskriver bruk av MPEG i omfattende interaktive multimedia applikasjoner og bruk av multimedia i infrastruktur.

Vi vil beskrive MPEG-1 og delvis MPEG-2 nærmere, men MPEG-4, MPEG-7 og MPEG-21 er ikke relevante for denne oppgaven.

For de interesserte kan mer info om disse standardene hentes hos:

Hjemmeside for MPEG:
http://www.cselt.it/mpeg/

Hjemmeside for MPEG-7:
http://www.mpeg-7.com/

2.0.3 Hva er JPEG

JPEG (*Joint Photographic Experts Group*) er et samarbeidsprosjekt mellom ITU (*International Telecommunication Union*) og ISO.

Målet til JPEG er å utvikle standarder for komprimering av stillbilder.

JPEG formatet er et av mest brukte formatene, fordi man kan oppnå en meget god komprimering, uten at bildekvaliteten forringes for mye.

JPEG arbeider for tiden med å standardisere et nytt JPEG format kalt JPEG2000.

Målet til JPEG er ikke å erstatte JPEG med JPEG2000, men å komplementere det eksisterende formatet med fremtidsrettede muligheter. Med bl.a bruk av wavelets, og raskere/bedre kompresjon.

For mer informasjon om JPEG:
http://www.jpeg.org/

2.1 MPEG video komprimering

Vi vil i denne delen av rapporten forklare enkelt hvordan MPEG video komprimering foregår, for å gi en bedre forståelse av virkemåten til MPEG hardware enkodere. Vi vil ikke ta for oss MPEG audio komprimering, siden dette faller utenfor området til denne rapporten.

2.1.1 Konvertering RGB til YCbCr

Før videodata kan komprimeres, konverteres de vanlige RGB (Rød, Grønn, Blå) dataene, som de fleste videokilder leverer, til et mer komprimeringsvennlig format.

Dette formatet, kalt *YCbCr*, deler videodata i 3 deler. En luminans (lysstyrke) del, og to krominans (fargemetning) deler. Henholdsvis *Y* for luminans, *Cb* for blå krominans, og *Cr* for rød krominans.

Konverteringsmatrisen for å gå fra RGB til *YCbCr* format, kan skrives som.

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.1687 & -0.3313 & 0.5 \\ 0.5 & -0.4187 & -0.0813 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

figur 2.1 Konverteringsmatrise RGB til YCbCr.

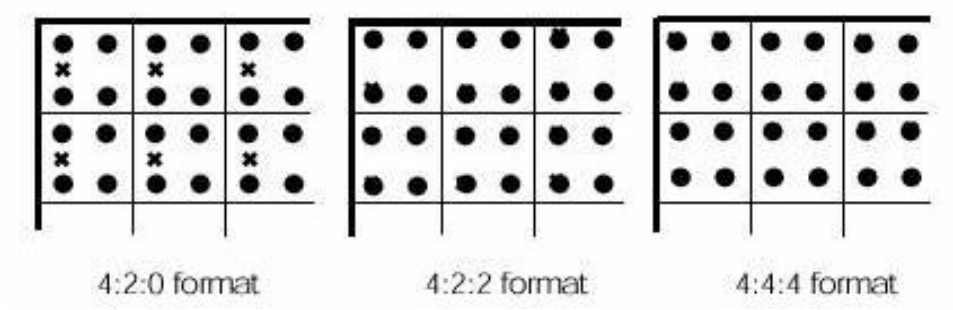
Hovedgrunnen til å bruke dette formatet er at forskning ved institutter for grafisk bildebehandling, har vist menneskeøyet er mer følsomt for lysintensiteten enn selve fargen. Ved å bruke *YCbCr* formatet kan man eliminere en del bildeinformasjon, ved å fjerne fargeinformasjon, uten at øyet vil merke noe særlig forskjell.

Et mye brukt format, er 4:2:2 formatet, som har 4 samplinger *Y*, og henholdsvis 2 samplinger med *Cb* og *Cr* som sendes med hver andre pixel.

En blokk med 4:2:2 samplinger kalles for *macroblock*, og vil forklares nærmere senere i dokumentet.

I MPEG komprimering benyttes vanligvis ikke begrepet pixel, men PEL som står for Picture ELement, og som er sample verdiene til et bestemt punkt i bildet.

Figur 2.2 viser forskjellige formater for YCbCr, luminans samplinger er • og krominans samplinger er **x**.



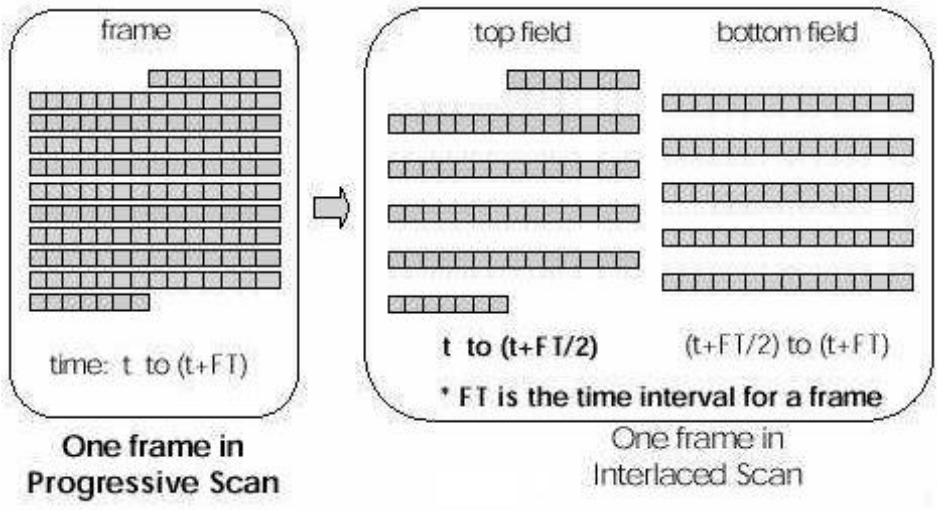
figur 2.2 formater for YCbCr koding.

2.1.2 Frame og Field

I videostandarder som PAL, NTSC, og SECAM, er hvert bilde (*frame*) delt opp i 2 delbilder (*field*), i en teknikk kalt interlaced. Grunnen til dette er at bilderørene som ble benyttet når standardene ble laget, ikke var raske nok til at de kunne tegne opp hele bildet på en gang. Man delte derfor bildet opp i 2 deler, og tegnet hvert delbilde for seg selv. Delbildene tegnes så raskt at øyet ikke oppfatter at det dreier seg om 2 forskjellige deler.

I Europa som stort sett benytter PAL, brukes det 25 bilder i sekundet, eller 50 delbilder.

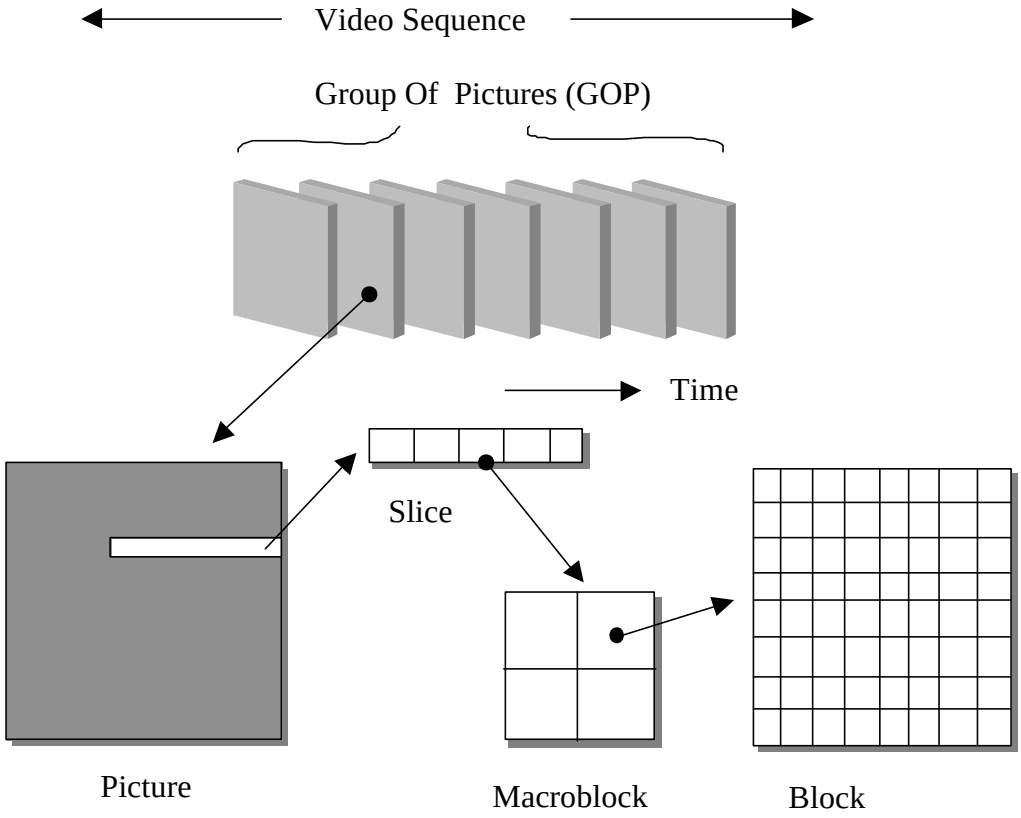
MPEG-1 benytter progressiv scan, mens MPEG-2 kan benytte begge deler, og egner seg derfor bedre til å komprimere videodata beregnet på bruk til TV formål.



figur 2.3 Sammenheng mellom frame og field

2.1.3 Block og Macroblock

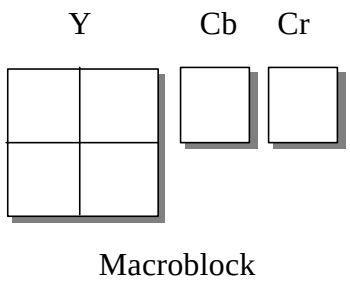
En *block* er en 8x8 blokk med PEL, se fig 2.4. Det er på block nivå at komprimeringen av MPEG data utføres med DCT (*Discrete Cosinus Transform*) som er hjertet i MPEG kompresjon.



figur 2.4 Sammenhengen i oppbygningen av MPEG

En macroblock er en 16x16 blokk, oppbygd av 4 blocks med luminans Y, og 2 blocks med krominans Cb og Cr, se fig 2.5. På macroblock nivå utføres ”*motion compensation/estimation*” og det er mulig å forandre kvantifiseringen i DCT.

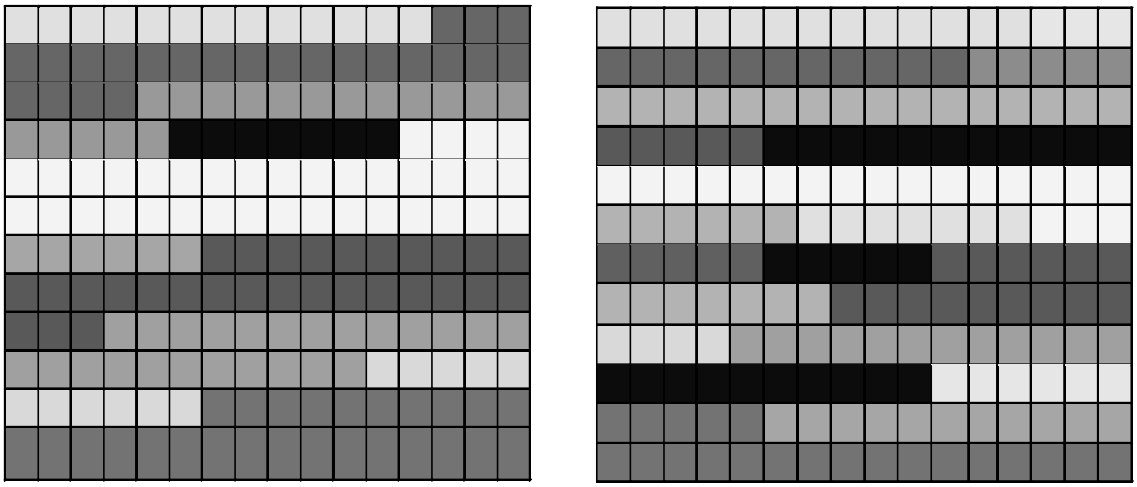
DCT og motion compensation forklares nærmere senere i rapporten.



figur 2.5 oppbygningen av en macroblock

2.1.4 Slice og Picture

Slice er en sekvens med sammenhengende macroblocks, se fig 2.4, som har samme luminans. En frame kan i teorien bestå av en lang slice i MPEG-1, men vanligvis så er det flere slices i en frame. I MPEG-1 kan en slice begynne i en rad, og fortsette til neste rad, men i MPEG-2, må hver slice avsluttes i samme rad i bildet, se fig 1.6.



figur 2.6 Eksempel på slice i MPEG-1 og MPEG-2, MPEG-1 til venstre.

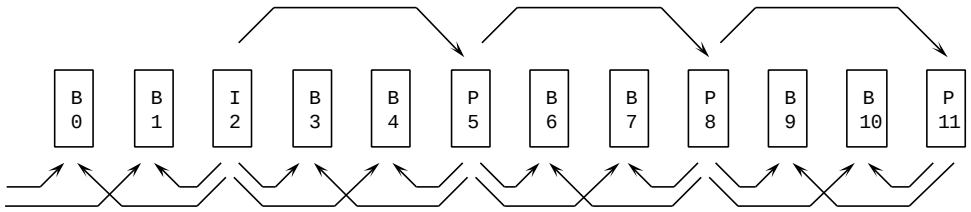
Hver slice kodes separat, og er med å bestemme kvaliteten i forskjellige bitrater, og kontrollere raten. Hvis det oppstår feil i en slice i datastrømmen, kan dekodingen fortsette med neste slice. Man mister derfor ikke et helt bilde ved feil, men kun slice som det er feil på.

Picture (bilde) er en frame eller field, MPEG bruker 3 forkjellige bildetyper som alle har forskjellig bruk, se fig 2.7.

Intra coded pictures (I-picture):
Er kodet uten referanse til andre bilder, brukes som start bilde i sekvenser og som referanse bilder. Bruker moderat kompresjon.

Predictive coded pictures (P-pictures):
Er kodet med referanse til tidligere ”I” og ”P” bilder, ved bruk av ”motion compensation”. Brukes som referanse for ”P” og ”B” bilder. Blir komprimert bedre enn ”I” bilder.

Bidirectionally-predictive coded pictures (B-pictures):
Er kodet med referanse til både tidligere og fremtidige ”I” og ”P” bilder, ved bruk av ”motion compensation”. Brukes ikke som referanse bilder. Blir komprimert best.



figur 2.7 Eksempel på bruk av ”I”,” P” og ”B” bilder.

MPEG trenger ikke andre bilder en ”I” bilder, men man vil da få en dårlig kompresjon.

2.1.5 GOP og Sequence

GOP (Group Of Pictures) er en serie med en eller flere kodede bilder, som gjør at en kan begynne å dekode MPEG data fra en tilfeldig posisjon. Det første bildet i en GOP er et ”I” bilde. Siden et ”I” bilde ikke har referanser til andre bilder.

Rekkefølgen av ”I”, ”P” og ”B” bilder er forskjellig i ukodet og kodet form, se fig 2.8.

Rekkefølge bilder i ukodet og dekodet form.

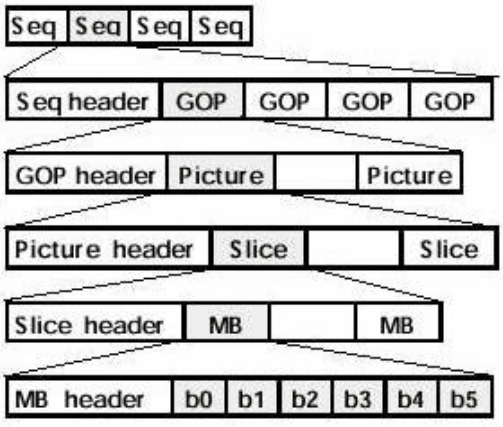
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28
I	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	I

Rekkefølge i kodet form.

1	4	2	3	7	5	6	10	8	9	13	11	12	16	14	15	19	17	18	22	20	21	25	23	24	28	26	27
I	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	I	B	B	P	B	B	P	B	B	I	B	B

figur 2.8 Eksempel på”I”, ”P” og ”B” bilder i ukodet, kodet og dekodet form

Figur 2.9 viser hvordan en MPEG video sekvens er oppbygd, med de forskjellige lagene, fra block til sequence.

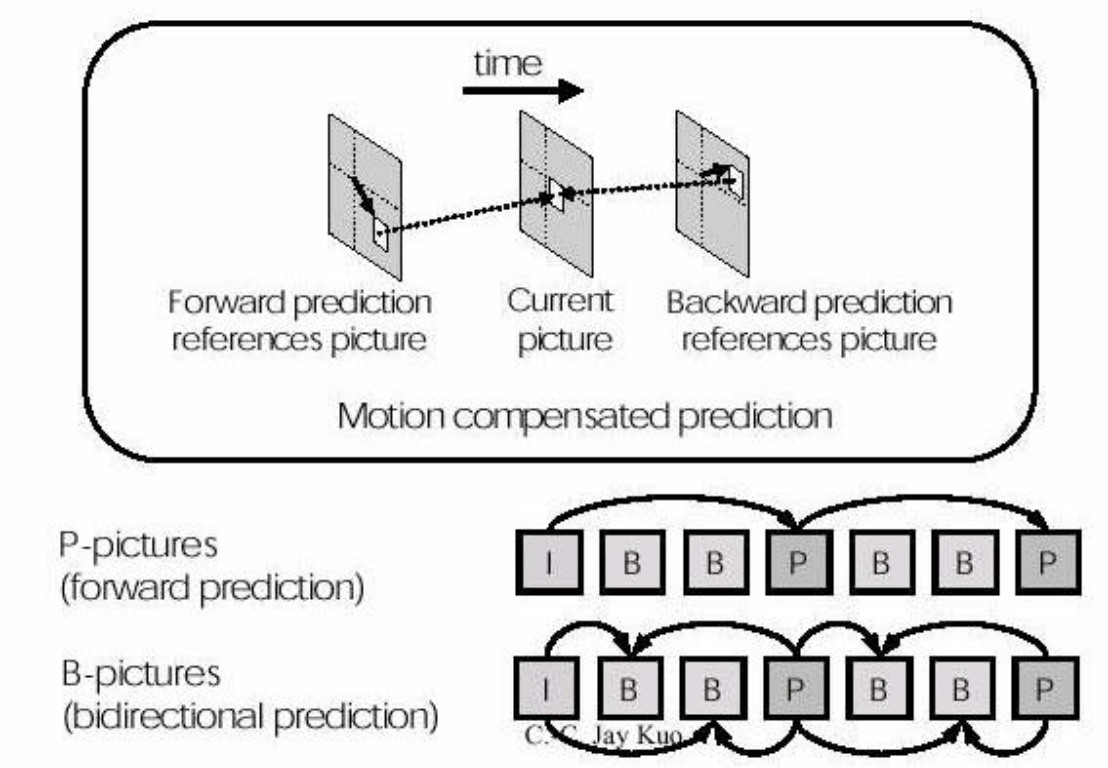


figur 2.9 oppbygningen av datastrømmen i MPEG

2.1.6 Motion Estimation & Compensation.

Ved Motion Estimation prøver man å forutsi verdiene til en block PEL i neste bilde ved å bruke en block i nåværende bilde. Forskjellen mellom block plasseringene, kalles Motion Vector. Det er enkoderens oppgave å beregne motion vector.

Dekoderen kan bruke denne informasjonen sammen med nåværende bilde, til å rekonstruere neste bilde. Denne prosessen kalles motion compensation. Generelt kan man si at motion compensation, er motsatt av motion estimation.

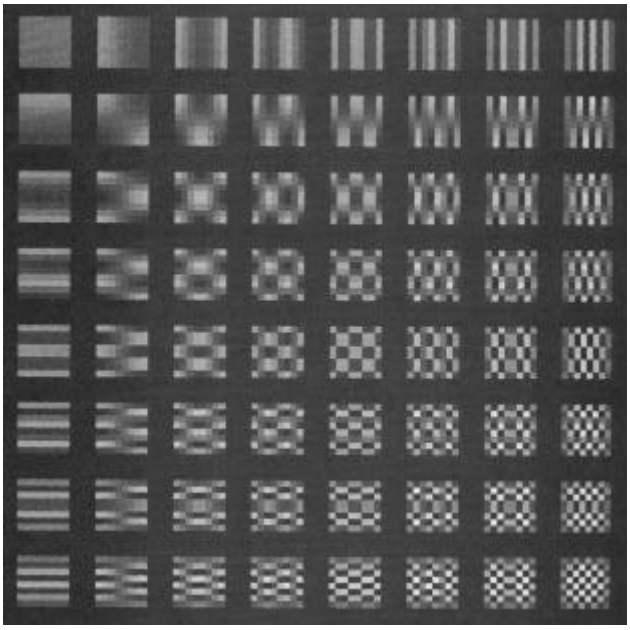


figur 2.10 Motion Estimation/Compensation

2.1.7 Diskret Cosinus Transform (DCT)

Diskret cosinus transform er selve hjertet av MPEG encodingen. Den har visse fordeler som forenkler kode modeller og gjør koden effektiv i forhold til hvordan øyet oppfatter bildet. Det er for det meste de delene av bildet som øyet ikke fanger opp som går tapt.

Det som skjer et at man ved hjelp av en vektet sum av frekvenser kan beskrive innholdet i en blokk. Man kan på den måte bare velge hvor mange frekvenser man skal bruke på hver blokk til å bestemme kvaliteten og bitraten.



Vi ser her en liten demonstrasjon på hvordan det hele virker. I retning nedover fra toppen, så øker verdien på den vertikale diskrete cosinus transformen. I horisontalretning fra venstre mot høyre, skjer det samme. Nede til høyre er et bilde der det er høy frekvens i begge retninger.

Der som er poenget her er at man kan sette sammen disse i forskjellige mønstre, og med større nøyaktighet i frekvensene, slik at de ved hjelp av veldig lite data kan gjengi en del av et bilde.

figur 2.11 Discret cosinus transform

2.1.8 Video Buffer Verifier (VBV)

For fast datarate. VBV verdien blir brukt til å sakke ned på eller øke bitraten slik at mottaker i andre enden ikke hverken går tom for data eller går tom for buffer. Dette kan brukes til å sørge for en datarate uten store bitrate ”spiker”.

3.0 Forskjellige IC kretser og løsninger.

Det er en lang og tidvis vanskelig prosess. Met datnke på at det ikke er veldig mange som faktisk har kompetanse på området rundt videokomprimering, og det er tilsvarende få bedrifter som leverer løsninger for formålet, så sier det seg selv at det er en møysommelig prosess å lete frem leverandører. Vi har de viktigste data fra de forskjellige reelle mulighetene vi hadde samlet i ett kapittel. Vi forsøker her å forklare hvorfor vi valgte den leverandøren vi valgte. Dette gjøres gjennom en liten kortfattet forklaring på brikkene de forskjellige leverandørene tilbyr.

3.1 Aktuelle kretser for MPEG1/2 og JPEG

Da vi valgte dette hovedprosjektet begynte vi straks å lete etter forskjellige aktuelle kretser vi kunne bruke til å komprimere video. Den første kretsen vi fant, ble tilfeldigvis den vi valgte å lage en fullstendig løsning rundt.

Vi går nå gjennom de kretsene og løsningene vi så på, og kommentere fordeler og ulemper ved de forskjellige. De fleste kretsene vi har sett på har vært MPEG2 kretser, for vi fant egentlig kun en MPEG1 krets som var aktuell. Av de MPEG2 kretsene vi så på så var det egentlig bare IBM sin som var noe som helst i nærheten av noe vi kunne bruke. De andre vi fant krevde en PC prosessor eller noe slikt, og var derfor automatisk utelukket. Andre kom kun fastmontert på ett eller annet kort, uten at de kunne kjøpes separat.

Grunnen til at det er så mye mer MPEG2 kretser enn MPEG1 på markedet er antakeligvis fordi både DVD og de nye digitale TV sendingene bruker MPEG2. Dette står det mer om i kapittel 1.

Vi fikk ganske strenge krav til temperatur av FFI. Vi fikk oppgitt at det i tuppen av raketten ville være rund 70°C, noe som egentlig er en standard temperatur. Det viste seg ganske fort at dette ikke ble så lett allikevel, da de fleste kretser med nok prosessorkraft ikke takler mer enn 40°C.

Vi hadde også ganske strenge krav til størrelse. Det var ønskelig at kretskortet ikke skulle være breiere enn 32mm, og opptil ca 10cm langt. Det å finne en krets som ikke er breiere enn 32mm var også en utfordring. Den valgte kretsen er faktisk så nærme som 31,5mm i bredde. Dette gir store utfordringer i kretskort utlegg.

Vi brukte veldig mye tid (og penger) på å ringe rundt til forskjellige leverandører i hele verden, fra Taiwan, gjennom USA til Sør Amerika og Asia. Det var ingen som kunne supplere oss med de kretsene vi ønsket.

I de tilfeller der vi ikke ville ha fått en brikke som både encoder JPEG og MPEG så ville vi ha benyttet en eksisterende løsning for JPEG som FFI har utviklet. Vi vil derfor ikke gå inn på de forskjellige leverandørene av JPEG brikker.

3.1.1 IBM

IBM har utviklet endel forskjellig kretser til å komprimere MPEG2 med. Vi hadde lenge en konversasjon med IBM Norge for å få tak i mer data for disse brikkene. Det hele så veldig lovende ut da vår kontaktperson ved IBM sa at vi skulle få tilsendt både brikker og datablad. Dette skjedde aldri.

Fra en tidlig mail:
"Fikk sendt over forespørsel om prøve til IBM på fredag. (S-series 4.2.0 Encoder) Har også bedt dem om å skaffe mer info på S-serien, som jeg kan sende deg. Dette tar gjerne noen dager, og vil gi deg en tilbakemelding så snart som mulig. "

Noe senere, etter purring:
"Det ser ut til at det tar noe lengre tid enn forventet, med å få en tilbakemelding i fra IBM. Jeg fortsetter å purre på dem, og gir deg en tilbakemelding så snart jeg hører noe."

Og enda senere:
"Dette har tatt veldig lang tid, og det beklager jeg på det sterkeste. Det har tydeligvis vært en del diskusjon frem og tilbake, men avgjørelsen har blitt, at IBM ikke ønsker å tilby produkter til militære applikasjoner. (Dette er tydeligvis deres policy) Jeg har forsøkt å få til noe, uten at jeg har kommet noe videre. Jeg beklager dette, og håper at dette ikke skaper for mye problemer for deg."

Vi skjønner ikke at ikke samarbeidpartnere er inneforstått med selskapets policy, men tar til takke med det vi fikk ut av dem. Vi fikk nemlig tak i noen små produktbeskrivelser.

IBM leverer to forskjellige løsninger. Den ene er til HDTV, noe som straks blir uaktuelt for oss. Den andre løsningen kalles S-serien. Den inneholder tre forskjellige brikker. Det er en for studio, en for , og en krets for alle andre formål. Disse kan forøvrig levere opp til 50Mb/s. Det står også at der er minstekrav om 0.5 m/s luftgjennomstrømning ved temperatur over 40°C. Dette gjør straks kretsen uaktuell for oss uansett. Når disse brikkene i tillegg har en størrelse på 35mm så ser vi ingen mulighet til å bruke IBM sin løsning.

3.1.2 Texas Instruments

Texas Instrumens leverer så mangt. Nå har vi ikke funnet noen MPEG encoder fra Texas, men vi har sett på muligheten med å implementere en MPEG encoder i en DSP. Dette var en løsning vi raskt utelukket. Dette ville tatt altfor lang tid å utvikle, selv med hjelp fra Texas sine egne video rutiner. Dette innebærer både omfattende lesing og forståelse av DSP programmering. Vi måtte i tillegg ha funnet en måte å lese den digitale datastrømmen som kommer fra video dekoderen, og det krever mye synkronisering av klokkefrekvenser og lignende.

Et annet problem med DSP ligner på det vi har med dagens prosessorer. De har en tendens til å kreve god kjøling dersom de går på så høy hastighet som vi ville måtte kreve.

Til gjengjeld åpner dette for mulighet til en enkel oppgradering til nye codecs for andre videoformat. Man ville i tillegg kunne bruke denne brikken til JPEG encoding, men det ville desidert bli enklere å bruke en annen krets til det formålet.

3.2.3 VisionTech

VisionTech leverer MPEG-2 encoder. Dessverre har vi ikke funnet ut om de leverer bare brikken, men etter bildene av kortet de leverer med MPEG-2 brikken sin på kan vi anta at den er for svær.

Det hadde vært moro å prøve å finne en løsning med MPEG-2, men det kan se ut som om teknologien ennå ikke tillater oss det. De tåler ikke nok varme, og er for store, ganske enkelt.

3.2 Valgt MPEG krets

Den første brikken vi fant var fra Winbond. Winbond er et Taiwansk selskap. Selskapet er en stor leverandør av videoprodukter, og det er en stor sjanse for at det står winbond på enkelte av kretsene dersom du kjøper deg et TV-kort til PCen din.

I Asia er markedet for VCD komponenter stort, i og med at VCD virkelig slo an i store deler av Asia. Dette er jo i stor kontrast med hvordan VCD ble mottatt i Norge og resten av Europa. VCD benytter som kjent MPEG1 standarden på video.

Kjernen i Winbond kretsen heter Z1011C, og er utviklet av Zapex. Zapex er et stor selskap innen alle former for bilde og videobehandling, og leverer produkter og teknologier til DVD, digitale kameraer, SVCD, digital lyd, og som sagt også kjerner til andre produsenter.

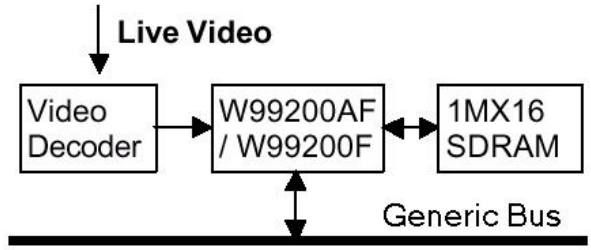
Oppgaven vi fikk fra FFI var som følger : ”Denne modulen skal komprimere video med MPEG/JPEG med varierende rater. Videodata skal sendes serielt ut i PCM-datastrømmen”.

Kjernen fra Zapex gjør akkurat det. Den komprimerer MPEG og JPEG med varierende rater. Det som skulle vise seg å bli den store utfordringen var å transportere data fra parallell utgang på MPEG encoderen til den serielle inngangen på PCM encoderen. Dette kommer vi nærmere inn på i kapittel 4.

Winbond kretsen vi har valgt å bruke har navnet W99200F. Det er en overflatemontert IC, med dimensjoner på 31,5mm * 31,5mm målt fra tuppen på benet. Dette holder så vidt. Vi ser vanskeligheten med å lage kretskort til denne, i og med at det må mange baner inn under kretsen, men vi har tro på at et flerlagskort vil kunne gjøre susen.

MPEG encoderen har 120 ben. Mange av disse skal være til databus på 32 bit (vi bruker bare 16 av disse) og endel til kommunikasjon og synkronisering med lyd og en eventuell MPEG dekoder. Brikken er hovedsakelig beregnet for PC. Den har innebygd støtte for PCI busen, men har også støtte for Generic bus, som vi benytter, og en noe usikker støtte for parallellport. Parallellport støtten fraråder datablad oss faktisk i å bruke.

Brikken bruker maksimalt 0.9W, og går på en 3.3V spenning. Intern klokkefrekvens på 54MHz, denne lages av en ekstern 27MHz klokkefrekvens. Det lave strømforbruket gjør slik at den tåler langt høyere varme enn de andre kretsene vi har sett på, faktisk hele 70°C. Dette gjør at brikken også takler kravene for omgivelsestemperatur.

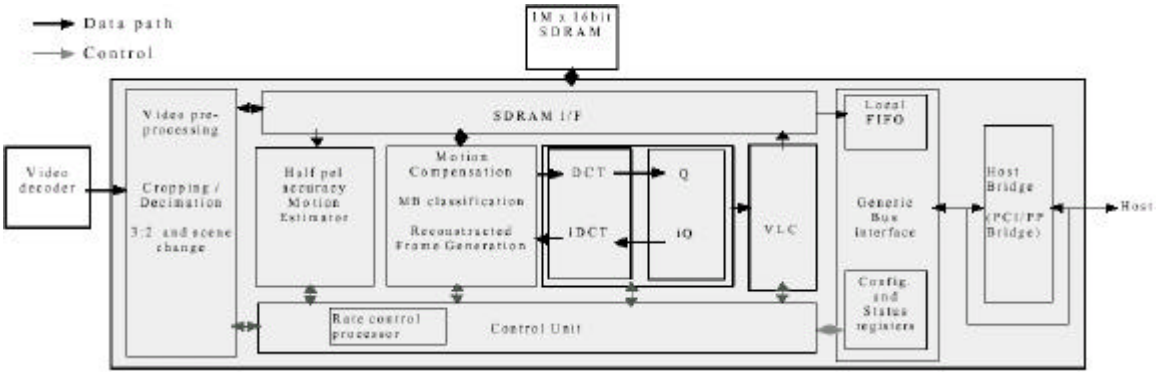


figur 3.1 Oversiktssjema for MPEG encoder tilkoblinger

Den inneholder direkte støtte for de fleste populære video dekoderene på markedet, uten å kreve noen form for signalomforming.

Den krever en 4MB SDRAM brikke, som man kobler direkte til brikken på egene pins uten at det er nødvendig med andre kretser.

Blokkskjemaet for brikken er veldig oversiktlig. Vi ser at den har en innebygd FIFO liste. Den gjør slik at vi slipper å ha noen form for buffer før PCM encoderen til forsvaret. PCM encoderen går med fast hastighet, så vi trenger bare å lese ut fra FIFO listen når det passer for oss. Meget praktisk



figur 3.2 Oversiktssjema over MPEGencoder chip

Vi ser på vårt valg av MPEG encoder som et bra valg. Det er ingen kretser som kommer i nærheten av hvor bra denne kretsen passer til vårt formål.

3.3 Resterende komponenter

Dette var en enkel del av prosessen. Siden vi hadde fått noen eksempler på kretskortutlegg fra Winbond, så valgte vi å gå for de samme kretsene som de har brukt. Det vil si at det ble enklere å konfigurere video dekoderen også, siden vi bare kunne ta det Winbond hadde gjort. For SDRAM brikken har det ikke så mye å si hvilken leverandør vi velger, men vi gikk for den samme som Winbond hadde valgt her også.

Som mikroprosessor valgte vi å gå for en prosessor fra Microchip. Vi har benyttet disse under våre forrige prosjekter, og har vært veldig fornøyde med måten man koder til disse. Det har dessuten vært greitt å ha muligheten til å kode i C kode. Vi måtte velge en prosessor med nok ben til å kunne behandle alle signalene vi ønsket.

For å få signalene fra MPEG encoderen til å bli serielle så bruker vi en Altera krets utviklet hos FFI. Vi har ikke noe stor beskrivelse av denne, men vi fikk nok info til å sette denne inn i kretsen vår.

I tillegg til dette bruker vi noen standard porter. Det er snakk om to AND porter og en OR port. Dette vil etter all sannsynlighet kunne implementeres i Altera kretsen, men det fikk vi ikke tid til, så derfor kjører vi nå med de som egne ICer.

MPEG/JPEG encoder :	W99200F (Winbond)
Video dekoder:	SA7113V1 (Philips)
SDRAM:	KM416S1020C (Samsung)
MUX:	Altera krets (FFI)
PCM encoder:	PCM encoder fra FFI (FFI)
AND port:	7432 (hvem som helst)
OR port:	7408 (hvem som helst)

4.0 Systembeskrivelse

Her kommer det en fullstendig beskrivelse av hele systemet vi har utviklet. Vi går gjennom endel muligheter og valg vi foretok. Siden vi aldri fikk tak i hardwaren vi ville ha, så ser vi det som en logisk reaksjon at vi prøver å presentere de forskjellige mulighetene som eksisterer i håp om å gi kommende prosjekter en idé eller to.

4.1 Dataflyt

Vi forklarer her enkelt hvilke muligheter vi har hatt, deretter presenterer vi det vi valgte å bruke.

4.1.1 Mulige dataflytmodeller.

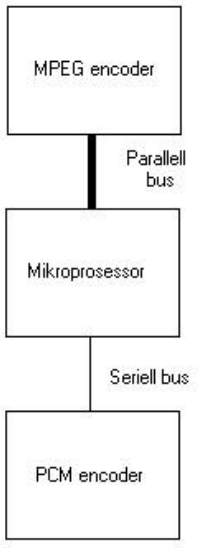
Den vanskeligste delen av oppgaven har nok vært å få dataflyten til å gå som ønsket. Det er mange mulig måter å få data til å flyte fra MPEG encoderen til PCM encoderen.

Først kommenteres hvorfor bruk av mikroprosessor som en databro ble valgt bort, deretter forklares hvilke signaler som er vesentlige for dataflyten, og deretter hvordan løsningen fungerer.

Vi så for oss tre mulige dataflytmodeller. Først så vi for oss å bruke Mikroprosessoren som en parallell til seriell bro. Deretter så vi på muligheten for å bruke Mikrokontrolleren som en parallell til parallell bro mellom MPEG encoderen og en MUX.

4.1.1.1 Parallell til seriell bro

Dersom vi skal sende data serielt ut til PCM encoderen, så må vi sørge for at dataflyten er synkronisert med PCM encoderens seriell data klokke (SCLK). Vi måtte derfor ha hatt tid til å polle SCLK, slik at vi ser at den går høy, deretter måtte vi ha lagt ut data manuelt på benet på mikroprosessoren, og igjen pollet for å finne neste stigende flake. Dette ville føre til at mikroprosessoren måtte ha minst fire ganger så høy hastighet som dataflyten. En klokkeperiode til å sjekke at SCLK er lav, en periode til å se at den er høy, en periode til å legge ut data, en periode til loop. Realistisk sett så må prosessoren ha en langt høyere hastighet for å både motta data og sende samtidig, og man bør i alle fatt få tid til å polle noen ganger i hver SCLK periode. Tatt i betraktning at PCM encoderen vil kunne komme til å kjøre på 10 Mbit, så skjønner vi at prosessoren vil måtte takle en meget høy klokkefrekvens, noe som fort blir dyrt og vanskelig å få til å tåle arbeidstemperatur på opptil 70°C.

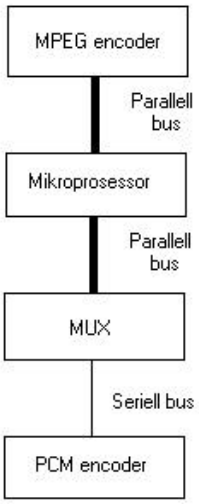


figur 4.1 Parallell til seriell bro

4.1.1.2 Parallell til Parallell (MUX) bro

En annen måte å bruke mikroprosessoren på ville være å ta imot data, legge det i en intern FIFO, for så å sende data til en MUX som kunne kommunisere med PCM encoderen. Da ville man kunne bruke en prosessor med langt lavere hastighet enn ved direkte seriell overføring til PCM encoderen. Vi fant ut at dette kunne være en gjennomførbar løsning, men hvis man likevel skal sende data parallelt videre så ville det være en vanskeliggjøring av dataflyten siden man bare sender data videre uten noe særlig behandling.

Prosessoren måtte også likevel kunne behandle inngående data fra MPEG encoderen, utgående data til PCM encoderen, eventuelt kunne ta imot data fra PCM encodern når det engang skulle bli implementert. Det ville jo gjøre sitt til at prosessoren måtte være ganske rask i dette tilfellet også.

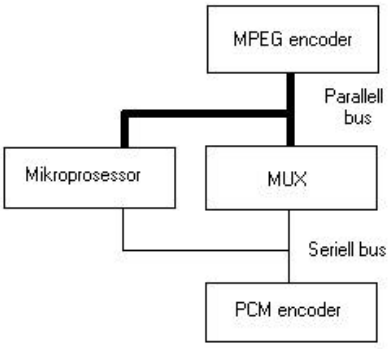


figur 4.2 Parallell til parallell bor

4.1.1.3 Styring av datastrøm

Den siste muligheten vi så på var muligheten til å sørge for at PCM encoderen en MUX og MPEG encoderen sørger for dataoverføringen selv, og bruke en mikrokontroller som et "overvåkende" element som sørger for at overføringen går smertefritt. Mikroprosessoren må også sørge for å forenkle mottaksprosessen for de som skal motta data fra PCM encoderen, slik at mottaker skjønner når neste dose med data kommer.

Valget falt på denne modellen, fordi den så ut til å være den billigste og mest oppgraderbare modellen. Den så også ut til å gi den desidert enkleste programmeringen.



figur 4.3 Styring av datastrøm

4.1.2 De viktigste signalegenskapene til brikkene

For at alle lesere skal kunne skjønne hvilke tilkoblinger som er gjort, og hvorfor, så presenterer vi kortfattet signalene fra de forskjellige brikkene. Vi har i tillegg skrevet en forklaring på I2C busen, da ganske få vet hvordan den fungerer.

4.1.2.1 Dataflyt fra MPEG encoderen.

Data som sendes ut ifra MPEG encoderen sendes parallelt, men med omvendt rekkefølge på bytene. (high byte på parallell utgangene 0 - 7, mens low byte kommer på 8 – 15) Data som sendes ut er asynkront med den interne klokkepulsen i brikken, og klokkes ut på databusen gjennom FIFO_RD_CLK pinen på MPEG encoderen. MPEG encoderen sender ikke ut data hvis ikke FIFO_RD pinen er aktiv (aktiv lav).

4.1.2.2 Dataflyt fra og til MUX

MUXen er veldig enkel. Den har en parallell inngang, en seriell utgang, en klokkepuls inn, og en LOAD_WA.

Når LOAD_WA settes aktiv, så låses den parallelle dataen inn på MUXen. Nå kan det legges ny data på innporten uten å forstyrre utsignalet. Deretter klokkes det ut data på loadut-pinen ved hjelp av klokkepulsen inn.

4.1.2.3 Dataflyt til PCM encoderen

Denne har flere signaler enn vi har bruk for. Vi bruker bare tre.
Klokkesignal, som kobles til MUXen for å klokke ut seriell data.
Data, som den serielle dataen fra MUXen klokkes inn på.
Gate, som kobles til LOAD_WA på MUXen. Dette signalet har en syklus for hver word, og derfor kobles dette signalet også til FIFO_RD_CLK på MPEG encoderen for å klokke ut data.

4.1.2.4 I2C-bus (mellom video dekoder og MPEG encoder)

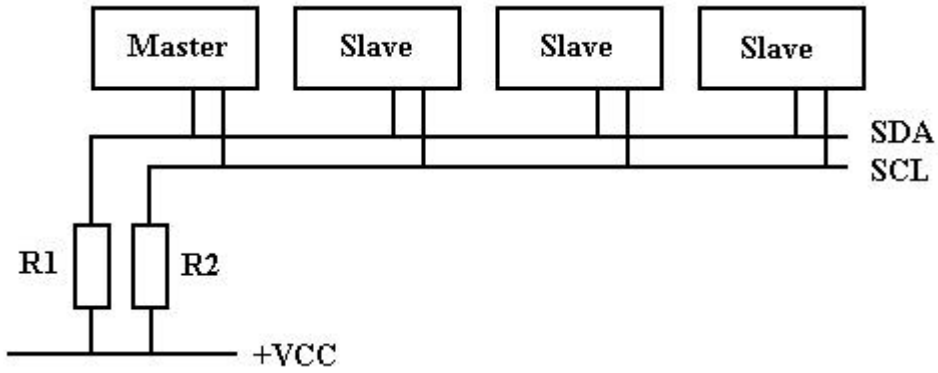
Beskrivelse av I2C busen

I2C busen er utviklet tidlig på 1980 tallet av Philips semiconductors. Formålet med denne bus-typen av at man skulle kunne overføre data mellom mange parter, og med opptil flere hoster på samme bus. Dette skulle brukes i TV apparater for å ha en sentral prosessor som styrer alt. Navnet står for Inter-IC, og skrives egentlig I^2C (I*I*C).

En stor fordel med busen er at det er en seriell bus, samtidig som den ikke er avhengig av tri-state utporter.

Det er to typer enheter på en I2C bus. Master er den som tar initiativ på busen, og ber om data, eller sender data til de kretsene det skal til. Slave er en passiv krets som bare mottar og sender data på kommando. Vi skal ikke gå inn på detaljer rundt hvordan I2C bus med flere mastere på samme bus fungerer, men vi kan kanskje nevne at det fungerer ganske likt som Interrupt Controller Communication i et Intel multiprosessor-system.

I2C busen er koblet opp slik:

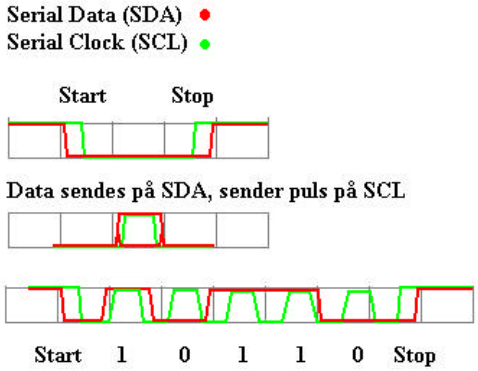


figur 4.4 Tilkobling til I2C bus

Der R1 og R2 er to store motstander. Det som skjer er at når master ønsker å kommunisere med en av slavene så sender den et start signal på busen. Fra det punktet så er busen opptatt, og master holder SDA og SCL nede (se neste figur). Deretter er der bare for Master å slippe taket på en av lederne for at de skal bli høye, VCC og motstanden sørger for at lederen får +VCC, eller høyt signal om du vil.

Det vil aldri forekomme at to brikker sender motsatt signal ut, og kortsluttes. Det er kun mulig å legge busen lav, og da vil det gå en svak strøm gjennom motstanden og inn i brikken som legger signalet lavt. Ellers er alle innganger sperret, og der går ingen strøm.

Etter at det er sendt et start signal, så kan man sende data, for så å avslutte med et stop signal. Se neste figur for å se hvordan signalene ser ut.



figur 4.5 Signaler på I2C bus

Fra en enhet på busen sender ett start signal, frem til den samme brikken sender ett stop signal, vil busen være opptatt. Etter stopp signalet er busen igjen ledig for overføring, og hvilken som helst brikke kan sende det den ønsker. Legg merke til at start og stop signalene er unike fordi der forandrer SDA mens SCL er høy. Alle andre overføringer krever at SDA er stabil mens SCL er høy. Det er ikke nødvendig med en fast frekvens, og man kan gjerne forandre hastigheten på klokkepulsen midt i en overføring.

Overføringen skjer som regel i steg. Det som følger er en beskrivelse av hvordan signalgangen vi bruker er. Det finnes mange metoder å gjøre dette på, og mange brikker har mulighet for sekvensiell lesing og skriving. Vi benytter oss ikke av de sistnevnte.

Signalene kommer i denne rekkefølgen.

- Alle brikker stenger sine utporter idet det kommer ett start signal.
- Deretter leser de første byten som blir sendt på busen. Den inneholder data om hvilken brikke som adresseres.
- Den brikken som har adressen som tilsvarer den første byten aktiveres, mens de andre venter på et stop signal.
- Masteren overlater deretter busen til den brikken som har adressen, som i sin tur sender en ’1’er tilbake, for å si at alt er i orden, og at den er blitt adressert.
- Deretter overtar Master busen igjen, og sender ny data. Dette er ofte en intern adresse, eller subadresse på den valgte slavebrikken.
- Slavebrikken overtar så busen igjen, og sender en ny ’1’er. (disse enerne kalles ACKNOWLEDGE).
- Nå kan master velge mellom to muligheter. Dersom master nå sender ny data på vanlig format, skrives det til slavebrikken. Dersom master sender et nytt start signal, etterfulgt av slaveadressen, så overtar slaven busen og sender data fra den valgte subadressen til master.
- Master eller slave, etter hvem som sendte mottok data, sender så ett acknowledge signal.
- Det sendes ett stopp signal, og alle slavene venter nå på et nytt start signal.

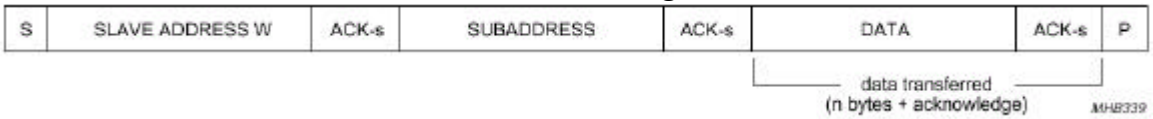
All data som sendes er 8 bit, etterfulgt av en acknowledge. Dersom kommando/adresse ikke blir godtatt, så er acknowledge en ’0’, hvis ikke er ’1’.

Dersom det ønskes mer informasjon om I2C, henviser vi til Phillips sine hjemmesider, eller http://perso.club-internet.fr/mbouget/i2c-faq.html som inneholder en kortfattet beskrivelse av I2C busen. Beskrivelse av hvordan vi har kodet dette finner du under 4.4.4

Dataformat mellom video dekoder og MPEG encoder

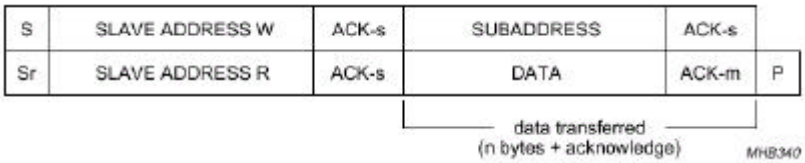
Dataformatet mellom video dekoderen og MPEG encoderen er ganske nøyaktig definert i databladet til video dekoderen. Vi antar at det kan forekomme forskjellig dataformat til forskjellige video dekodere, og det er derfor MPEG brikken ikke har noe innebygd funksjon for hele I2C transaksjonen.

Dataformatet i med vår video dekoder er for skriving til en slave brikke



figur 4.6 I2C signaler ved lesing

Og for lesing fra slave brikke:



figur 4.7 I2C signaler ved skriving

Dette passer bra inn i den beskrivelsen av I2C busen vi gikk gjennom i forrige avsnitt.

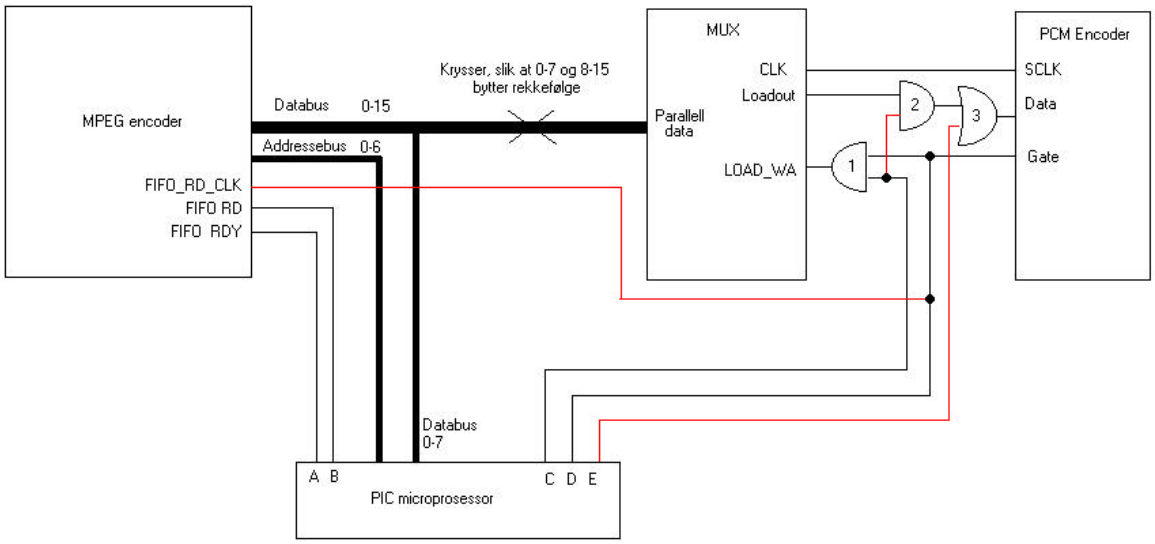
4.2 Teknisk løsning

Her kommer en forklaring på hvordan vi så for oss at brikkene skulle kobles sammen, deretter noen ord rundt kretsskjemaet.

4.2.1 Signalgang

MPEG encoderen må levere data litt tregere enn PCM encoderen mottar. Hvis datastrømmen fra MPEG encoderen er raskere enn PCM encoderen klarer å motta, så vil data gå tapt. Det må selvfølgelig unngås for at mottakeren skal kunne dekomprimere videosignalet. Dette medfører at det blir enkelte tomme hull i datastrømmen. For at mottaker skal kunne vite hva som er brukelig informasjon, og hva som ikke skal brukes, sørger vi for å ha et fast mønster.

Valget falt på en meget enkel modell. Encoderen har en innebygd FIFO liste. Denne listen må aldri bli full, for da taper vi data. Derfor er det et signal (FIFO_RDY) fra MPEG encoderen som sier fra når FIFO listen når et visst nivå. Dette nivået bestemmes av et treshold register. Etter at FIFO_RDY er satt, må vi lese treshold antall bytes før FIFO_RDY signalet skal kunne settes igjen. Vi vet derfor at lengden på datapakkene er treshold lange.



figur 4.8 Blokkskjema over oppkoblig for dataflyt

I og med at vi vet størrelsen på datablokkene, trenger vi nå bare si ifra når en ny datablokk kommer. Vi sørger for at all data som er utenfor datapakkene er 0. Dette gjøres gjennom å legge AND (1, 2) portene som er mellom Loadout på MUXen og Data på

PCM encoderen lav fra mikroprosessoren. Når vi da får et FIFO_RDY signal, venter vi en klokke, så setter vi FIFO_RD signalet aktivt.

Deretter setter vi data inn på PCM encoderen til 1 ved hjelp av OR (3) porten. Denne holdes på 1 til det er sendt et word (det vil si en puls på gate). Samtidig setter vi C høy, og åpner begge AND portene. Da sendes første word ut fra MUXen. Det første ordet vil bare inneholde enere for å si at det nå kommer en datapakke, for å forenkle mottaksprosessen.

Vi venter så en klokke. Data ligger nemlig ikke på datautgangene på MPEG encoderen på klokke forsinket. Etter det er gått en klokkepuls legger vi så OR(3) porten lav fra encoderen (E) og åpner AND portene, for utklokking av data.

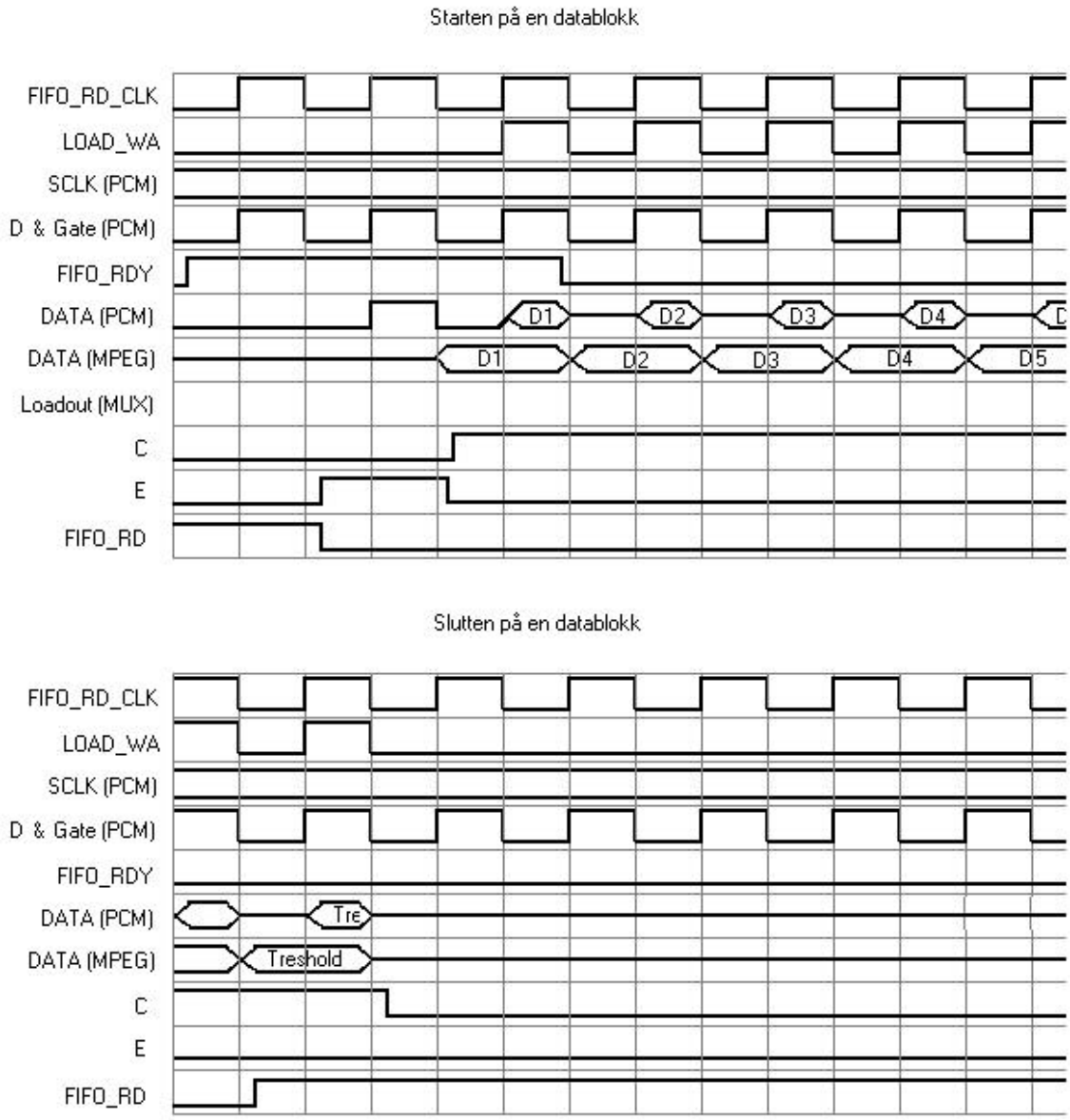
Ved den neste klokken (dvs den andre pulsen etter at FIFO_RD signalet er satt) ligger det data på databusen, og PCM encoderen klokker da inn data slik den skal.

Syklusene vil nå følge Gate på PCM encoderen. For hver gang det leses ut ett word serielt fra MUXen, sendes ny data til MUXen fra encoderen. Vi trenger da bare å telle antall høye pulser fra gate på PCM encoderen, og stenge datastrømmen med AND portene når treshold antall word er sendt. Deretter venter vi på nytt FIFO_RDY signal, og gjentar prosessen.

Det er ikke nødvendig å lese fra FIFO med en gang FIFO_RDY blir satt. Derfor får vi litt bedre tid til å sende krontrollsignaler mellom hver sendesyklus. Vi kan altså gi kontrollord, og forandre de innstillinger som er mulig å forandre, mellom sendesyklusene.

Signalgangen blir da seende ut som på figuren på neste side.

Denne enkle modellen gjør at mottaker kan identifisere begynnelsen på en datapakke gjennom å vente på en byte med bare enere. Når denne kommer, så leses treshold antall bytes, for så å vente på enere igjen. Dette forenkler mottaksprosessen stort i forhold til en modell som bare inneholder tomrom. Det ville ført til at mottaker måtte analysert signalet grundig. Man har uansett ikke mulighet til å vite hva slags signal man skal vente seg, eller hvor mange bytes som skal kuttes vekk, så vi så på løsningen som nødvendig og forenklende for mottaker.



figur 4.9 Signalgang mellom PCM encoder, MUX, microprosessor og MPEG encoder

4.2.2 Kobingssjema

Vi har ikke laget noe ferdig kretskortløsning. Dette kan til dels begrunnet med at vi ikke så noen grunn til det da vi ikke får tak i brikken, og til dels at FFI ønsker det i OrCad. Siden FFI ønsket det hele i OrCad, og sannsynligvis vil bruke OrCad hvis de ønsker å utvikle kretsen videre, så vi ikke noen grunn til at kretskortutlegget skulle lages to ganger. Vi har derfor bare ett koblingsskjema for hele kretsen. Det er etter vår mening blitt både oversiktlig og lettlest.

Det er viktig at man legger merke til at MAST_EN pinen skal være aktiv. Det er for å enable FIFO_RDY signalet, som vi benytter flittig. Det er også vesentlig at HTS[0:1] pinene blir lagt riktig. HTS0 skal være lav, og HTS1 skal være høy for at brikken skal være i Generic bus mode.

Ellers så er det meste ganske enkelt. Oppkoblingen av AND portene og OR porten har vi vist i kapittel 4.2.1, og det samme med PCM encoderen og MUXen.

SDRAM brikken og Video dekoderen er koblet rett til encoderen slik som Winbond har gjordt i sine kretskortutlegg.

Det er også viktig å merke seg at vi bruker pull-up motstander på alle portene på mikrokontrolleren. Dette er fordi mikrokontrolleren ikke har disse innebygd. Vi antar at det også kan være tilfelle for andre signaler også. Siden alle brikkene kan motta mye mer strøm en de kan avgi, går vi ut ifra at det vil funke greitt. Dersom man ikke skal ha pull-up motstander på alle signalene, kan man bare lodde de ut igjen. Dette er noe som må gjøres i en testfase.

4.3 Konfigurering

MPEG encoderen konfigureres gjennom å sette de interne registrene til ønskede verdier. All datastrøm inn til encoderen sendes gjennom samme databus som datastrømen med komprimert bilde og video. Dette fører til at vi har veldig små tidsluker til å konfigurere encoderen mens det sendes data. Det må nemlig gjennomføres mellom hver ”sendesyklus”. Vi vil gå igjennom alle stegene som må til for å konfigurere encoderen, slik at den både starter riktig, og komprimerer riktig.

For å kunne velge riktige innstillinger, så trenger vi noe kunnskap angående periferi-enheter, og hvordan disse er konfigurert. Først så vil vi gå igjennom konfigureringen av video dekoderen, deretter hvordan vi bestemmer hva slags datarate vi skal kjøre på.

Nå har vi litt bakgrunnskunnskap, så nå går vi gjennom hvilke registre som må settes for å starte riktig. Vi vil fortsette med hvilke registre som må settes for de forskjellige encoding metodene som er aktuelle.

Etter det forrige steget vet vi hvilke registre som er aktuelle for oss, så vi vil så forklare hvert enkelt register grundig, og gå gjennom hvordan man kan sette disse registrene. Denne delen er viktig å sette seg inn i dersom man ønsker å få god kvalitet samtidig som man ønsker er lav datarate.

4.3.1 Bestemme datarate

Det er mange faktorer det kommer an på når man skal bestemme dataraten. Det vi her skal prøve å gi begrunnelse for er den maksimale dataraten ut ifra MPEG encoderen, sett i forhold til dataraten fra PCM encoderen.

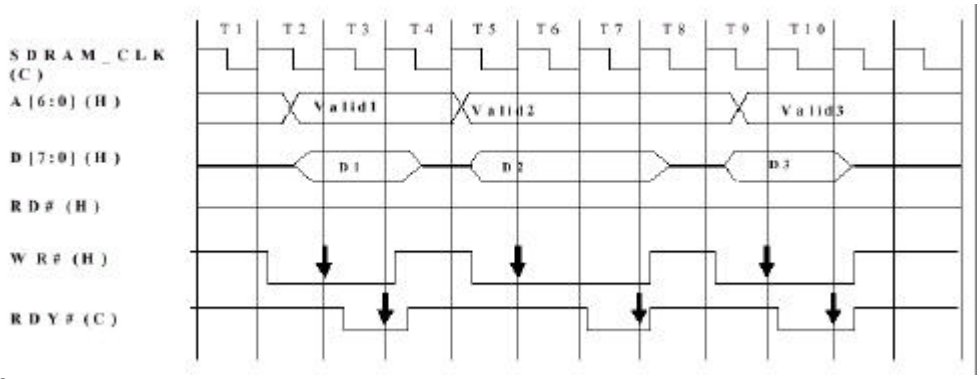
Encoderen har en innebygd FIFO liste som kan inneholde opp til 128 double word (32 bit), som gir oss 256*16bits FIFO. Det er viktig at denne listen aldri blir full, for da kan man tape data.

For å få til en fungerende dataflyt, så må dataraten fra MPEG encoderen være lavere enn dataraten ut ifra PCM encoderen. Dette kan jo sammenlignes med en veistrekning. Man må få minst like mange biler ut i den ene enden, som man sender inn i den andre enden, hvis ikke blir det bare fler og fler biler, og til slutt så er det ikke plass til flere, og noen må kjøre av vegen (les: avslutte sin ferd uten å komme frem).

For å få valgt datarate til MPEG encoderen så må man vite dataraten til PCM encoderen. Deretter må man gjennom konfigurering av MPEG encoderen sette en maksimal datarate som ligger noe under nivået til PCM encoderen. I teorien så går det jo hvis begge dataratene er like, men da har vi ikke mulighet til å sende kontrollord til MPEG encoderen, og da heller ikke stoppe datastrømmen. Vi har gjennom et enkelt lite regnestykke funnet ut omtrentlig hva slags forhold mellom dataratene fra PCM encoder og MPEG encoder det minst bør være for at vi i tillegg til å overføre data også skal kunne sende kontrollord og forandre på registerinnstillinger.

For å kunne sende data til et register trenger vi antall klokkesykluser:

1. Sette databus-porter til utporter
2. Sette addresse på addressebus
3. Sette data på databus
4. Sette WR (write signal) aktivt
5. Polle RDY signalet (kan ta mer enn enn klokke, så vi teller denne som to)
6. Sette WR inaktiv
7. Sette databus-porter til inn-porter



figur 4.10 Signalgang ved skriving til registre

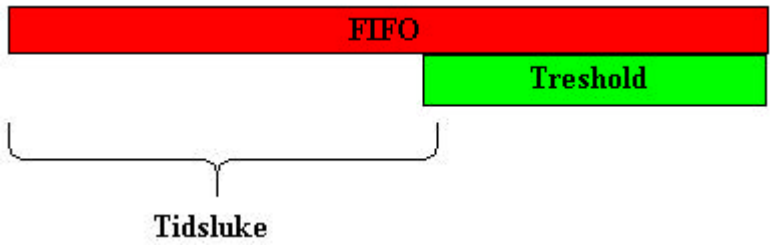
Det vil si 8 klokkesykluser

Siden prosessoren vi bruker går på 1 MHz intern klokkefrekvens, så vil dette da ta:

$$\frac{8cycl}{1000000cycl/sek} = 0{,}000008sek$$

Dette gjelder uansett hvor høy datarate man får ut fra encoderen. I tillegg så bør vi egentlig ha litt ekstra tid til eventuelle funksjonskall og returnering fra funksjoner. Vi setter da verdien til å være 0,000012 sek, slik at vi har god tid.

Hvis man kjører på maksimal datarate (vi sier 10Mbit, for å forenkle regningen litt) og maksimal størrelse på treshold, samtidig som vi forventer at en dataoverføringssyklus er ett word raskere ut ifra PCM encoderen enn dataoutputen fra MPEG encoderen, vil vi få en tidsluke som ser noe slik ut:



figur 4.11 FIFO og Treshold

FIFO er 256 lang, og Treshold er 60 lang. Da blir tidsluka lik FIFO minus Treshold pluss 1 (fordi dataoverføringa er en word raskere enn dataoutputen fra encoderen). Det gir oss

$$\lfloor 256 - 60 + 1 = \underline{\underline{197}} \rfloor words$$

hvilket er

$$197words * 16bits = \underline{\underline{3152bits}}$$

Det gir oss en tidsluke på

$$\frac{3152}{10000000} = 0{,}0003152sek$$

Vi ser at vi uansett har mer enn nok tid til å fullføre opp til flere registerskrivninger. Datarateforholdet vårt er derfor ikke begrenset av verken vår skrivning til registre, eller vår lesning fra registre (som er omtrent det samme). Dersom det blir mye skrivning til registre, noe vi antar at det ikke blir, så må dette kanskje tas med i betraktningen for hvor

stor datarate man skal velge fra MPEG encoderen. Vi skal bare sende et Vstop kontrollord for å stoppe encodingen, og trenger derfor ikke ta hensyn til dette.

Av det vi har regnet oss frem til nå, ser vi at det eneste kravet er at dataraten ut fra PCM encoderen må klare å sende litt mer enn det MPEG encoderen genererer. Vi overlater det derfor til de som kan teste denne kretsen å finne ut hvor stort forholdet må minst være. Vi kan jo bemerke at dersom man overfører MPEG1 med samme datarate på MPEG encoderen som på PCM encoderen, så vil dette antakeligvis ikke være noe problem da den gjennomsnittlige dataraten vil som regel være vesentlig lavere enn den dataraten vi har valgt (sett bort ifra at vi har valgt data stuffing da, noe som vi ikke ser nytten av i denne oppgaven). Vi kan dermed anta at vi kan bruke en datarate som ligger veldig tett opptil det PCM encoderen har.

Vi bemerker også et annet fenomen, som bør løses gjennom å teste kretsen. Det blir forklart under register innstilling på registrene Vbit_rate_m og Vbit_rate_l, at brikken kan komme til å ha en høyere bit-rate enn det vi har valgt. Dette vil kunne føre til at mottakeren ikke mottar hele det neste bildet i datastrømmen før det skal dekodes. Vi antar at dette ikke vil være noe problem så lenge man har stor nok buffer i mottakerdelen, men dette er noe som må testes. Det kan hende bitraten fra MPEG encoderen må ligge lavere enn det man egentlig ønsker, for at kretsen skal tåle å få enkelte datarate-topper.

4.3.2 Metoder for å forandre datarate fra MPEG encoder

Forandre oppløsning

Det er flere måter å styre dataraten fra PCM encoderen på. Den enkleste måten vil være å senke oppløsningen på bildet. Dette vil være nødvendig for å oppnå veldig lave datarater. Det kan jo legges til at det ofte gir bedre bilde med lav oppløsning på lave datarater enn med høy oppløsning på den same dataraten. Vi skal se på kvaliteten på videobildene senere.

Frame size	MPEG1 'I' frame Live video encoding	MPEG1 'IBP' Live video encoding
	Bit rate (Kbps)	Bit rate (Kbps)
SIF	1536 - 9216	192 - 6144
QSIF	512 - 3072	64 - 2048

figur 4.12 Oversikt over mulige bitrater ved forskjellige oppløsninger

Vi ser her hvilke bitrater som er tilgjengelige ved de forskjellige oppløsningene. Vi ser liten nødvendighet i å bruke MPEG1 'I' frame encoding, hvilket vil si at hvert enkelt bilde er fullt komprimert, og er ikke avhengig av forrige og neste bilde for å dekomprimeres. Dersom vi bruker MPEG1 'IBP' encoding, vil alle I bildene være fullt komprimert, mens B og P bildene vil være avhengige av forrige og neste bilder. For mer informasjon på dette les definering av IPB bilder for mer info.

Forandre bildemønsteret

Ettersom alle I bilder er fullstendige bilder, uavhengige av andre, så vil disse naturlig nok kreve mest båndbredde. P bilder er avhengige av foregående I og P bilder, og inneholder kun forandringer. Disse vil derfor ta langt mindre plass enn I bilder. Dersom vi velger å bruke mange P bilder vil derfor datastrømmen kreve mindre båndbredde. Sist, men ikke minst, må det nevnes B bildene, som er avhengige av både tidligere og kommende bilder. Disse bildene vil ta mindre båndbredde enn P bildene igjen, og dermed sørge for lavere datarate. Det man gir fra seg ved å velge P og B bilder er bevegelse i bildet. Man vil få deler av bildet som er statiske, eller veldig lite bevegelse i. Til gjengjeld får man lavere båndbreddekrav eller bedre kvalitet på bildet. I vårt bruk vil statiske deler av bildet ha lite å si, og vi velger å bruke ende B og P bilder for å få bedre bildekvalitet. Dersom kameraet bare får inn himmel, eller en lineært bevegende bakke, så vil komprimeringer bli meget bra, og båndbreddekravet lavt.

Forandre bildekvalitet

Vi har metoder for å velge kvaliteten på bildene som overføres. Dette gjøres gjennom konfigurering av MPEG encodere. Bildekvaliteten vil vanligvis bli valgt av encoderen, slik at bitraten stemmer med det vi ønsker. Det er også mulig å velge variabel bitrate og konstant kvalitet. Problemet med det sistnevnte er at vi ikke har en særlig stor buffer i MPEG encoderen, og derfor vil den metoden til tider kunne gi store krav til datarate.

For å se hva slags kvalitet MPEG video gir ved forskjellige datarater heviser vi til vedlegg 6.1. Der er det en rekke bilder som viser kvaliteten ved forskjellige datarater. Ellers vil vi vise til filmene som ligger på CDen. Filmene ligger i en egen katalog. Der er seks filmer som bare er forskjellige datarater og oppløsninger av samme filmen, samt originalfilmen.

Det er også to filmer som er komprimert med en Winbond W99200F brikke. Disse er funnet på Internet.

4.3.3 Konfigurering av video dekoder

Konfigureringen av video dekoderen er en omfattende prosess. Datablad er på hele 80 sider, og den har hele 65 kontrollord som må settes.

Kort fortalt inneholder chipen alle valg som et moderne TV har, og endel ekstra. Her kan det forandres både luminans, crominans, brightness, contrast, crominans-saturation, -hue, -gain og så videre. Det er også endel registre som går på valg av format (f.eks PAL eller NTSC). I tillegg kan man koble til fire forskjellige kameraer til denne brikken, for så å velge mellom de analoge inngangene. Dette syntes vi var veldig praktisk, men det har jo lite nytte ved det formålet vi utvikler for.

I og med at det bare finnes en løsning på hvodan denne MPEG encoderen skal konfigureres, så så vi det som lite nødvendig for oss å sette oss inn i noe så omfattende som denne video dekoderen. Vi prioriterte derfor å beskrive MPEG encoderen grundig, og tok alle innstillingsverdier fra ei fil vi fikk fra Winbond. I kildekoden i Boot() funskjonen kan dere se hvilke kontrollord som sendes hvor.

Vi gjør dette avsnittet ganske kort ved å henvise dere til datablad for brikken. Datablad ligger i sin helhet på Internet eller på den leverte CDen.

Internett adresse for datablad :
http://www.semiconductors.philips.com/pip/SAA7113H/V1

4.3.4 Konfigurering av MPEG encoder

Dette er vel utvilsomt den mest omfattende delen av rapporten. Den ble så omfattende på grunn av at MPEG encoderen vi valgte har et datablad som vi ikke har lov til å sende med. Vi valgte derfor å legge den nødvendige informasjonen i rapporten.

Først så beskriver vi hvilke forskjellige moduser man kan kjøre MPEG encoderen i. Deretter forklarer vi hva slags prosedyrer go hvilke registre som må settes for de modusene vi skal bruke. Deretter har vi beskrevet alle relevante registrene gurndig.

4.3.4.1 Moduser

Empeg encoderen har hele tolv forskjellige moduser. Disse er:

1. Real time live video encoding mode
   Her hentes videosignalet fra video decoderen, det encodes i MPEG eller M/JPEG og sendes så videre i komprimert form.
2. Real time live video pass trough mode
   Her hentes videosignal fra video decoderen og sendes direkte videre uten at det komprimeres på noen måte. Vi kommer ikke til å benytte denne formen da vi ikke har stor nok båndbredde til dette.
3. Live video snap shot mode
   Denne modusen henter ett og ett bilde fra video decoderen på reqest. Denne brukes i forbindelse med det neste punktet for å sende stillbilder.
4. Single frame encoding mode
   Denne modusen brukes for å encode ett bilde. Dette er en funksjon som kan være nytting når man skal encode video som man har liggende på f.eks en datamaskin. Vi vil bruke denne til å encode og sende et bilde som vi har hentet fra video decoderen ved hjelp av forrige modus.
5. Write frame mode
   Dette er en modus der du kan skrive ett bilde inn i SDRAM brikken slik at du kan encode det med encoderen
6. Read frame mode
   Denne brukes til å lese ett bilde fra minnet. Det kan for eksempel brukes til å lese ett ukompromert bilde som man har hentet fra video decoderen, eller hente et bilde man har lagt inn i SDRAMen selv ved hjelp av Write frame mode.
7. SDRAM write mode
8. SDRAM read mode
   Disse to brukes til å lese og skrive direkte til minnet. Denne funksjonene er egentlig bare til for å teste at encoderen kommunsierer riktig med minnebrikken. Vi ser ikke på det som et potensielt problem, og velger å overse denne funksjonen. Vi refererer til datablad dersom man ønsker å feilsøke ved hjelp av disse modusene.
9. Write internal memories mode
10. Read internal memories mode

Disse brukes til å skrive alle innstillingene direkte inn i brikken. Dette er en mye raskere metode for å raskt initialisere brikken. Vi velger å ikke bruke denne modusen fordi vi vil uansett bruke mange forskjellige innstillinger og moduser, samtiding som det blir langt mindre oversiktlig med ett langt sett med hexadecimale data. Dessuten er det ikke noen stor tidsbegrensning i dette prosjektet, så om vi bruker noen sekunder med i startup fasen spiller ingen rolle. Dette kan kanskje være en utfordring og optimalisering å få til dersom man allerede har en fungerende ordning som man vil forbedre.

11. Audio bitstreams input mode

    Dette er en mode som brukes til å få perfekt audio/video synkronisering. Siden vi ikke har noen lyd, så er denne uten nytte for oss.

12. VCD bitstreams pass trough.

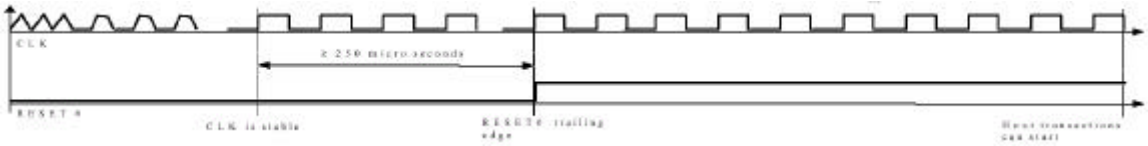
    Denne brukes i forbindelse med dekoding av VCD signal, og er helt uaktuell for oss

Vi kommer kun til å beskrive modusene 1, 3 og 4, da disse er de eneste vi kommet til å benytte oss av.

Nå følger prosedyrer som må følges for de forskjellige modusene. Start prosedyren er felles for alle.

4.3.4.2 Start prosedyre

Når man starter kretsen så er det essensielt at man følger denne signalgangen før man foretar seg noen som helst form for initialisering.



figur 4.13 Start sekvens for MPEG encoderen

Ifølge vår dataflytmodell, så må MAST_EN pinen på encoderen settes aktiv for at FIFO_RDY skal kunne settes. Dette går vi ut ifra når vi beskriver resten av register innstillingene.

Først må man velge rikting klokkekilde. Vi har en ekstern klokkekilde, og må derfor sette BY_PASS biten (bit 2) i Proscessor Control Registeret (0x66) til 0. Mens vi er i gang med å stille inn PCR registeret, kan vi ta med at resten av registeret også må settes. Fyldigere beskrivelse av dette under punktet registerinnstilling.

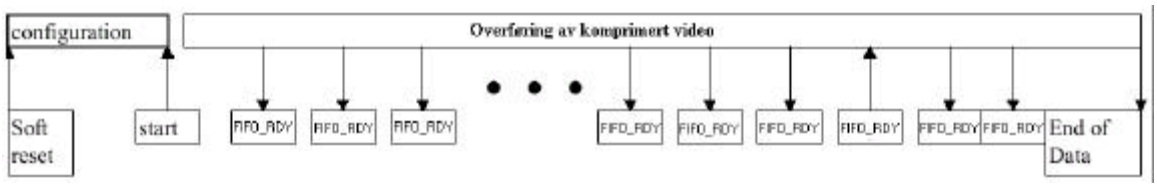
4.3.4.3 Live video encoding

Det er denne modusen som er den mest aktuelle i vårt tilfelle. Vi kommer derfor til å beskrive denne delen ekstra grundig.

Disse stegene må man gjennom for å stille inn encoderen for live MPEG1 encoding:

1. Video_reset
2. Skriv SDRAM bit map til minne. Dette er hvis man ønsker en alfa blendet logo lagt oppå bildet. Det ønsker ikke vi.
3. Video_reset
4. Skriv til internt minne (modus 9) – initialliser registre i brikken. (Dette gjør ikke vi, men det er her det eventuelt skal gjøres, dersom man velger å initialisere alle registrene med egne default verdier)
5. Konfigurering av encoderen
6. Vstart signal sendes for å starte video overføring
7. Dataflyten sørger for at alt som kommer i FIFO listen blir sendt til PCM encoder. (interrupt behandling)
8. Vstop signal sendes
9. Dataflyten sørger for at alt som kommer i FIFO listen blir sendt til PCM encoder frem til Video end of data interrupt kommer. (interrupt behandling)

Legg merke til at det fremdeles genereres data etter at Vstop signalet er gitt. Dette er for at data som kommer ut ikke skal bare slutte brått, men det skal få et riktig format ifølge MPEG standarden. Konfigurering, dataoverføring og start/stopp signal vil se ut som dette:



figur 4.14 Oversikt over signalgang fra MPEG encoder

Nå har vi en modell for når det forskjellige skal skje. Vi vil nå gå gjennom de registrene som må settes for live video encoding. Ingen registre som ikke er nevnt her er relevante for live video encoding. Vi bemerker at disse registrene, og hvordan man skal bestemme innholdet på de forskjellige er grundigere beskrevet i et senere avsnitt (registerinnstilling).

MPEG encoding:

Register	Komentarer
Vint_enable	Her velger man aktive interrupts
Vtreshold	Her velger mn hvor full FIFO listen skal være før man får beskjed.
Vwork_mode	Bitene 4:0 settes lik 0x0 for MPEG encoding
Video_format	Her velger man SIF, QSIF, og NTSC/PAL/SQUARE
Venc_cntl	Bit rate restriksjoner (variabel, fast, max), Halv pixel nøyaktighet, 'B' start, lukket GOB, Telecine, scene-bytte deteksjon
Vframe_pattern	Velger 'm' og 'n' verdier
Vbit_rate_m, Vbit_rate_l	Valg av data-rate.
Vbv_size	Velger VBV størrelse. Ignorert i variabel bitrate
Vbv_initial	Velger VBV start verdi. Ignorert i variabel bitrate
Vquality	Velger kvaliteten på video signalet. Ignorert i fast datarate
Vslice_header	Velger avstanden mellom slice header i videostrømmen
Vgop_header	Velger avstanden mellom GOB header i videostrømmen
Vit_hour	Start tid time
Vit_minute	Start tid minutt
Vit_second	Start tid sekund
Vit_frame	Start tid frame i det sekundet
Vin_offset	Etter hva video dekoderen er satt til
Vin_cntl	Etter hva video dekoderen er satt til
Vin_picture	Etter innsignal + av/på med innsetting av bitmap

tabell 4.1 oversikt over registre som må settes ved live video MPEG encoding

M/JPEG encoding:

Register	Komentarer
Vint_enable	Her velger man aktive interrupts
Vtreshold	Her velger mn hvor full FIFO listen skal være før man får beskjed.
Vwork_mode	Bitene 4:0 settes lik 0x10 for JPEG encoding
Video_format	Her velger man SIF, QSIF, og NTSC/PAL/SQUARE
Venc_cntl	Bit rate restriksjoner (variabel, fast, max), Halv pixel nøyaktighet, 'B' start, lukket GOB, Telecine, scene-bytte deteksjon
Vframe_pattern	Velger avstanden mellom to encodede bilder
Vquality	Velger kvaliteten på video signalet. Ignorert i fast datarate
Vin_offset	Etter hva video dekoderen er satt til
Vin_cntl	Etter hva video dekoderen er satt til
Vin_picture	Etter innsignal + av/på med innsetting av bitmap

tabell 4.2 oversikt over registre som må settes ved live video MJPEG encoding

Nå følger en liten innføring i hvordan dataflyten ut fra MPEG encoderen er i grove trekk.

- Dataoverføring starter når Vstart registeret blir skrevet til.
- Gjennom data overføringen vil FIFO_RDY settes hver gang det ligger treshold antall bytes i FIFO listen. Da må det leses treshold antall data-enheter (byte, word, double word) fra FIFO listen.
- Mens dataoverføringen foregår kan man fremdeles lese statusregistrene. Spesielt viktig er det kanskje å lese Vfifo_status registeret for å sjekke hvor mye data det ligger i FIFO listen.
- Encoderen slutter å hente et blide i disse tilfellene
    1. Vstop registeret blir skrevet til. I dette tilfellet fullføres encodingen av hentede bilder.
    2. FIFO overflow. I dette tilfellet slutter encodingen, og et interrupt-bit blir satt. Man kan fremdeles lese det resterende innholdet i FIFO listen.
- Når siste data-enhet er lest fra FIFO listen settes det et interrupt flagg. Denne må enten polles, eller settes som aktivt interrupt som sender #INT signal.

Når man sender data i MPEG format så består bildestrømmen av I, B og P bilder. I bildene er bilder der hele bildet er encodet, mens B og P bilder er avhengige av foregående og følgende bilder. Dette gjør at man for det meste bare sender de delene av bildet som forandrer seg. Dette senker dataraten betraktelig. Dersom man har valgt å bruke M/JPEG format, så vil datastrømmen bestå av en kontinuerlig strøm av JPEG bilder, uten header eller noen annen slags data mellom bildene. Man kan ikke velge en maksimal datarate for M/JPEG, så den blir litt uaktuell for oss, i og med at vi har strenge krav til maksimal datarate.

4.3.4.4 Single frame grab

Denne funksjonen tar ett enkelt bilde ut ifra den innkommende videostrømmen, og lagrer dette i SDRAM brikken. Dette kan da leses direkte ut ifra SDRAMen og sendes videre som et ukomprimert bilde. Vi kommer ikke til å bruke dette, for oppgaven forteller oss at vi skal kunne ”komprimere MPEG og JPEG med varierende rater”. Vi kommer derfor til å benytte denne fuksjonen i forbindelse med den neste vi har beskrevet, nemlig Single Frame Encoding. Vi har da mulighet til å komprimere bilder ”on demand”, eller med faste mellomrom. Vi vil da ta Single frame grab og Single frame encoding annehvert. Dette kan gi oss en meget lavere datarate enn det vi kan få med live video, samtidig som vi får høy bildekvalitet. Bakdelen er at man ikke får et kontinuerlig oppdatert bilde.

Det er også her endel punkter som man må igjenom for å få tatt et ”snap shot”. Dette er vesentlig enklere enn for live video.

1. Video_reset
2. Skriv SDRAM bit map til minne. Dette er hvis man ønsker en alfa blendet logo lagt oppå bildet. Det ønsker ikke vi.
3. Video_reset
10. Skriv til internt minne (modus 9) – initialliser registre i brikken. (Dette gjør ikke vi, men det er her det eventuelt skal gjøres, dersom man velger å initiallisere alle registrene med egene default verdier)
4. Konfigurering av encoderen
5. Vstart signal sendes for å starte skriving av det førstkommende bildet til minnet.
6. Vente til ”end of data” interrupt for å være sikker på at et helt bilde er skrevet til minnet.

Konfigureringen i denne modusen er ganske kort og enkel. Det er bare snakk om 8 registre som må settes. Alle registrene er bedre beskrevet i et sener avnsitt (Register-innsilling)

Register	Komentarer
Vint_enable	Her velger man aktive interrupts
Vtreshold	Her velger mn hvor full FIFO listen skal være før man får beskjed.
Vwork_mode	Bitene 4:0 = 0x2
Video_format	Her velger man FULL/SIF/QSIF, og NTSC/PAL/SQUARE
Vin_offset	Etter hva video dekoderen er satt til
Vin_cntl	Etter hva video dekoderen er satt til
Vin_picture	Etter innsignal + av/på med innsetting av bitmap
Vmem_select	Velger hvilken del av minnet som bildet skal lagres i. Man kan lagre mer enn ett bilde i minnet samtidig.

tabell 4.3 oversikt over registre som må settes ved single frame grab

4.3.4.4 Single frame encoding

Denne modusen brukes i forbindelsen med den forrige vi beskrev. Det denne modusen gjør er at den går til en bestemt del av minnet, og komprimerer det den finner der til enten JPEG eller I-MPEG format, etter ønske.

Punktene som man må gjennom i denne modusen er:

1. Video_reset
2. Konfigurering
3. Vstart-kommando
4. Dataflyten sørger for at alt som kommer i FIFO listen blir sendt til PCM encoder. (interrupt behandling)

Registerene som må settes i denne modusen er heller ikke så tallrike, men det er en viktig ting man må legge merke til. Vi ser nemlig her at Vmem_select ikke skal settes til noen verdi. Dette er antagelig fordi at den skal inneholde samme verdi som den hadde da man grabbet bildet, eventuelt at man la bildet inn i SDRAMen manuelt – noe vi ikke kommer til å gjøre – og med det valgte minneplassen. Dette medfører at man ikke kan legge mer enn ett bilde i minnet dersom man skal bruke denne funksjonen. Nå har ikke vi testet dette, og datablad sier ingenting om at Video_reset resetter dette registeret eller ikke. Dersom Video_reset

Register	Komentarer
Vint_enable	Her velger man aktive interrupts
Vtreshold	Her velger mn hvor full FIFO listen skal være før man får beskjed.
Vwork_mode	Bitene 3:0 = 0x3. Velger JPEG/MPEG encoding
Video_format	Her velger man SIF, QSIF, og NTSC/PAL/SQUARE
Vquality	Her velger man kvaliteten, og dermed også størrelsen på bildet.
Vsize_h	Her velger man dern horisontale størrelsen på det komprimerte bildet
Vsize_v	Den vertikale størrelsen
Vslice_header	Velger hvor ofte slice header skal forekomme i bitstrømmen
Vit_hour	Start tid time
Vit_minute	Start tid minutt
Vit_second	Start tid sekund
Vit_frame	Start tid frame i det sekundet

tabell 4.4 oversikt over registre som må settes ved single frame encoding

Dataoverføringen foregår på samme måte som ved live video, med unntak av at man bare venter på end og data.

4.3.4.5 Register-innstilling

MPEG encoderen har en stor samling av registre som kan settes. Vi har valgt å kun nevne de som er relevante for vårt bruk. Dersom det er ønskelig med beskrivelse av de resterende registrene, må man få tak i ett komplett datablad fra Winbond.

Nå følger ei oppramsing av alle registrene vi kommer til å benytte og beskrive, og hva slags type disse er. Vi nevner også adressen til alle registrene, og hva de kort fortalt gjør. Etter denne oppramsingen, kommer det en grundigere forklaring på hvordan hvert enkelt register opererer og fungerer, samt hvordan vi skal velge verdier for dem.

Addr.	Gruppe	Navn	Beskrivelse
0x00	Data	Vdata_out	I dataoverføring er dette utdata fra FIFO. Dette leses fra vha FIFO_RD signalet.
0x02	Data	I2c_data	Her legge dataen som skal overføres på I2C busen
0x04	Kommando	Video_reset	Beukes til å resette video delen av encoderen.
0x05	Kommando	Vstart	Starter encodingsprosessen og datastrøm.
0x06	Kommando	Vstop	Stopper encodingsprosessen ved neste bilde.
0x07	Kommando	I2c_start	Sender en I2c START signal på I2c busen
0x08	Kommando	I2c_stop	Sender en I2c STOP signal på I2c busen
0x0C	Interrupt	Vint_enable	Enable/disable interrupts
0x0D	Interrupt	Vint_source	Informasjon om hvilket interruptflagg som ble satt
0x0E	Interrupt	Vint_clear	Nullstiller aktive interrupt kilder og INT signalet
0x10	Konfigurering	Vtreshold	Velger FIFO treshold nivå
0x11	Konfigurering	Vwork_mode	Velger arbeidsmodus og encoding format
0x12	Konfigurering	Video_format	Velger input/output bildeformat
0x13	Konfigurering	Venc_cntrl	Velger bitrate restriksjons type, halv-pixel nøyaktighet, B-frame start, åpen/lukket GOB, Inv. Telecine, scene skifte detetktering.
0x14	Konfigurering	Vframe_pattern	Velger m og n verdier for bilde mønster
0x15	Konfigurering	Vbit_rate_m	Setter encoding-bit-rate. Mest signifikant byte
0x16	Konfigurering	Vbit_rate_l	Setter encoding-bit-rate. Minst signifikant byte
0x17	Konfigurering	Vbv_size	Setter størrelsen på VBV
0x18	Konfigurering	Vbv_initial	Setter VBV start-fullhet
0x19	Konfigurering	Vquality	Setter konstant kvalitet

0x1A	Konfigurering	Vslice_header	Setter slice header frekvens
0x1B	Konfigurering	Vgob_header	Setter GB header frekvens
0x1C	Konfigurering	Vit_hour	Start tid – timer
0x1D	Konfigurering	Vit_minute	Start tid – minutt
0x1E	Konfigurering	Vit_second	Start tid – sekunder
0x1F	Konfigurering	Vit_frame	Start tid – bilde nummer
0x20	Konfigurering	Vin_offset	Video dekoder sync. offset.
0x21	Konfigurering	Vin_cntl	Video dekoder control register
0x22	Konfigurering	Vin_picture	Settes etter innsignal + enable/disable logo.
0x23	Konfigurering	Vmem_select	Velger minneplass som blir skrivet/lest i skrive/lese SDRAM samt internal memory modusene.
0x24	Konfigurering	Vsize_h	I singel frame encoding velger horisontal størrelse
0x25	Konfigurering	Vsize_v	I singel frame encoding velger vertikal størrelse
0x30	Status	Vfifo_status	Forteller om hvor full FIFO listen er.
0x31	Status	I2c_status	Forteller om hvordan forrige I2C transaksjon gikk
0x66	Konfigurering	PCR	Processor Control Register

tabell 4.5 oversikt over samtlige registre som settes

Det er svært viktig at reserverte adresser ikke skrives til, da det kan gi uante følger. Det samme gjelder for reserverte bit i registrene. Disse skal ALLTID settes til ’0’.

Gå aldri utenfor verdiene som er satt som maksimale for registrene. Dette kan også gi udokumenterte problemer.

Det er også viktig at ingen registre blir skrevet til etter at man har sendt en Vstart kommando. Det er to unntak, Vquality og Vin_picture. Disse registrene kan forandres underveis. Vquality kan forandres ved M/JPEG for å senke/øke bitraten. Det skjer ikke øyeblikkelig. Vin_picture kan forandres for fjerne logo på bildet.

Vdata_out (0x00)

Dette er et register man kun kan lese fra. Her kommer all den komprimerte dataen ut. Utdata kommer i formatet 8, 16 eller 32 bit. I 16 og 32 bit utgaven vil databytene komme i omvendt rekkefølge, slik at den minst signifikante byten er den siste av de to/fire bytene i MPEG datastrømmen. Det er derfor vi bytter om bit 7-0 og 15-8 før de kommer til MUXen (se avsnitt om Teknisk Løsning).

I Generic Bus mode, som vi bruker, vil man kunne lese dette registeret ved hjelp av FIFO_RD signalet, og utdata forandrer seg synkront med FIFO_RD_CLK signalet.

Dersom man prøver å lese samme data to ganger (hvis man sender to pulser på FIFO_RD), så vil data forandre seg.

I2c_data (0x02)

Dette er registeret man legger data man skal sende over I2C busen i. Det er også her data man henter fra I2C busen havner. Dette registeret er altså både lesbart og skrivbart.

Video_reset (0x04)

Dette registeret kan man kun skrive til. Hvis man skriver til dette registeret vil man resette video delen av brikken. Det gjelder uansett hva slags data man sender.

Vstart (0x05)

Dette registeret kan man kun skrive til. Hvis man sender data til dette registeret, så starter man den valgte modusen.

Vstop (0x06)

Dette registeret gjør akkurat det motsatte av Vstart. Dersom man skriver til dette registeret, vil man stanse modusen. Dette registeret vil bli brukt til å stanse komprimering av live video.

I2c_start (0x07)

Dette registeret kan man bare skrive til. Uansett hva man sender til dette registeret setter ett startsignal på I2C busen. For nermere forklaring på I2C, se eget avsnitt.

I2c_stop (0x08)

Dette registeret kan man bare skrive til. Uansett hva man skriver til dette registert setter ett stoppsignal på I2C busen.

Vint_enable (0x0C)

Dette registeret brukes til å velge hva slags feil som skal gi interrupt. Det har en default start verdi på 0x00. Registeret kan både skrives til og leses fra.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	I2C ready	VBV underflow	Vin Out-Of-sync	Video data error	Video FIFO Overflow	Video FIFO Ready	Video End of Data
0	0	0	0	0	0	0	0

figur 4.15 Vint_enable registeret

- Bit 0 er til ”Video end of data”. Denne vil sette INT signalet aktivt når den siste databyten er lest. Dette brukes til å finne ut når man har lest slutten på MPEG videoen, eller slutten på ett enkelt enkodet bilde.
- Bit 1 er til ”Video FIFO ready”, hvilket vil si at den fungerer slik som FIFO_RDY signalet. Den settes når FIFO listen inneholder mer enn treshold data.

- Bit 2 er ”Video FIFO overflow”. Det betyr at vi har vært sløve, og ikke lest data fra FIFO listen raskt nok. Hvis dette interruptet kommer, så vil ecodingen ha blitt stoppet umiddelbart.
- Bit 3 er ”Video data error”. Denne settes dersom man sender inn ett annet bildeformat enn det som er definert i Vsize_h og Vsize_v. Vi kommer ikke til å bruke denne funskjonen..
- Bit 4 er ”Vin Out-Of-Sync”. Denne forteller oss at videoen som kommer fra decoderen ikke er synkronisert med encoderen. Det vil si at klokkefrekvensene ikke passer helt sammen. I og med at vi tar klokkefrekvensen fra dekoderen og bruker denne til encoderen, så skal ikke dette være noe problem.
- Bit 5 er ”VBV underflow”. Dette er en teoretisk feil. Dersom mottaker har et buffer som definert i registeret VBV_size, vil dette bety at mottaker ikke har fått nok data til å dekomprimere hele det nåværende bildet, og vil dermed ha fått en feil. Denne gjelder kun i live video MPEG1 med konstant bitrate.
- Bit 6 er ”I2C ready”, og forteller oss om I2C busen er ferdig eller ikke. Denne kan brukes til å finne ut når forrige transaksjon ble ferdig, slik at man ikke sender ett stoppsignal før all data er klokket ut på busen.
- Bit 7 Reservert. Må være ’0’

Mer enn ett interrupt kan være aktivt samtidig. Det er software som prioriterer interruptene.
Hvis man forandrer på Vint_enable, så kan det resultere i at INT blir aktivt, i samsvar med Vint_source.

Vint_source (0x0D)

Dette er et register man bare kan lese fra. Dette registeret inneholder informasjon om hvilke interruptflagg som er aktive. Disse flaggene blir satt når hendelsene skjer, uavhengig av hvilke interrupt som er valgt i Vint_enable. Dersom de ikke aktivert i Vint_enable, så vil de ikke generere ett INT signal.

Reset verdien til dette registeret er 0x40.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	I2C ready	VBV underflow	Vin Out-Of-sync	Video data error	Video FIFO Overflow	Video FIFO Ready	Video End of Data
0	1	0	0	0	0	0	0

figur 4.16 Vint_source registeret

Dette registeret burde bli lest når man mottar ett INT signal, for å finne ut hvilket interrupt det er som har blitt sendt og hvilken interrupt rutine som skal kjøres. Man kan også polle dette registeret dersom man venter på ett event, som for eksempel at man venter på at I2C busen skal bli ferdig med å klokke ut data.

For å sliope å kopiere data, henviser vi til Vint_enable registerets dokumentasjon for funksjonsbeskrivelse av de forskjellige flaggene som kan settes.

Vint_clear (0x0E)

Dette registeret kan man bare skrive til. Dette registeret brukes til å nullstille valgte interrupt-flagg i Vint_source registeret. Man nullstiller for eksempel bit 2 ved å sende 0x04 til dette registeret. Man kan nullstille flere interrupt-flagg samtidig.

Vint_rd_clear (0x0F)

Dette registeret kan bare lese fra. Dette registeret fungerer omtrent slik som Vint_source. Det spesielle med dette registeret er at man samtidig som man leser hvilke interupt-flagg som er satt, nullstiller "Video FIFO ready" biten, slik at man sparer en transaksjon på det, sett iforhold til å først lese Vint_source, deretter skrive til Vint_clear. Vi bruker ikke Video FIFO ready interruptet, da vi bruker FIFO_RDY signalet istedenfor. Derfor er dette ikke veldig aktuelt for oss. Det kan kanskje gjøre jobben enklere i noen tilfeller, og vi nevner det derfor her.

Vtreshold (0x10)

Dette registeret brukes til å velge hvor full FIFO listen skal være før vi mottar ett FIFO_RDY signal eller "Video FIFO ready" i Vint_source blir satt. Det kan både leses fra, og skrives til. Det har verdien 0x04 når encoderen blir resatt.

Vi kan her velge om vi vil ha store eller små datapakker som blir sendt av gangen.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Threshold Level [5]	Threshold Level [4]	Threshold Level [3]	Threshold Level [2]	Threshold Level [1]	Threshold Level [0]
0	0	0	0	0	1	0	0

figur 4.17 Vtreshold registeret

Når FIFO listen er over Vtreshold, vil det følgende skje:

1. Vint_source "Video FIFO Ready" bitet vil bli satt
2. Dersom "Video FIFO Ready" bitet i Vint_enable er satt, sendes det ett INT signal.
3. Dersom MAST_EN signalet er aktivt, sendes det også ett FIFO_RDY signal.

FIFO listen kan maksimalt holde 256 16bits ord. Den maksimale verdien dette registeret kan ha ved 16bit er fra 0x1 til 0x1E, eller fra 1 til 30. Dette tallet ganget med 4, forteller oss hvor mye data som skal ligge i FIFO listen før ett FIFO_RDY signal sendes.

Eksempel:

I 16bit mode, og med Vtreshold = 0x0F, vil det sendes ett FIFO_RDY signal når det er 15*4 = 60 16bits ord i FIFO listen.

Vwork_mode(0x11)

Dette registeret velger hva slags modus vi skal være i, og hva slags dataformat utdata-strømmen skal være. Dette registeret er default 0x00, hvilket betyr live video MPEG1.

[7] ***	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Audio clock locking	Reserved	Reserved	Encoding format	Working mode [3]	Working mode [2]	Working mode [1]	Working mode [0]
0	0	0	0	0	0	0	0

*** Dette registeret der kun tilgjengelig i W99200F B-0 kjernen. I A-0/1 kjernene er dette et reservert bit.

figur 4.18 Vwork_mode registeret

Bit 3:0 –Arbeids-modus.
- 0 – Live video enkoding
- 1 – Live video pass trough
- 2 – Live video snap shot
- 3 – Single frame encoding mode
- 4 – Write frame mode
- 5 – Read frame mode
- 6 – SDRAM write mode
- 7 – SDRAM read mode
- 8 – Internal memories write mode
- 9 – Internal memories read mode
- A-F reserved

Bit 4 – Encoding format for live video eller Single frame encoding
- 0 – MPEG1
- 1 – JPEG

Bit 7 – Lyd-synkronisering. Ikke noe vi kommer til å bruke.

Video_format (0x12)

Vi kan både skrive til og lese fra dette registeret Det forteller oss hva slags video input og output format som brukes. Dette må samsvare med video dekoderen. Default verdien i dette registeret er 0x05, som er quare pixel NTSC format. Vi skal ha dette i standard PAL signal, og må derfor velge andre verdier.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Reserved	Picture Format [1]	Picture Format [0]	Picture Size [1]	Picture Size [0]
0	0	0	0	0	1	0	1

figur 4.19 Video_format registeret

Bit 1:0 – Bilde størrelse
- 0 – Full
- 1 – SIF
- 2 – QSIF
- 3 – Annet (definert i Vsize_h og Vsize_l)

Bit 3:2 – Bilde format
- 0 – CCIR601 NTSC
- 1 – Square pixel NTSC
- 2 – CCIR601 PAL
- 3 – Reservert

Hvis man bruker live video encoding så må man i bit 0:1 velge enten SIF eller QSIF. Oppløsningen i de forskjellige tilfellene er i følgende tabell

	Input Video Format	Input Video Resolution	Output Video Resolution (4:2:0)				
			FULL	SIF		QSIF	
			Cropping	Cropping	Decimation	Cropping	Decimation
1.	CCIR601 NTSC	720 x 480	704 x 480	704 x 480	352 x 240	704 x 448	176 x 112
2.	CCIR601 PAL	720 x 576	704 x 576	704 x 576	352 x 288	704 x 576	176 x 144
3.	SQUARE NTSC	640 x 480	640 x 480 (no cropping)	640 x 480 (no cropping)	320 x 240	640 x 448	160 x 112

figur 4.20 Forskjellige oppløsninger ved forskjellige formater

Venc_cntl (0x13)

Dette registeret kan man både skrive til og lese fra. Det er et register som styrer mye av komprimerings prosessen, og det er her man velger hva slags bitrate-restriksjoner vi skal bruke. Reset verdien er 0x0C.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Time Stamp Enable	Auto Scene Change	Auto Inverse Telecine	Closed GOP	'B' Start	Half Pixel ME search Enable	Bit Rate Policy [1]	Bit Rate Policy [0]
0	0	0	0	1	1	0	0

figur 4.21 Venc_cntl registeret

Bit 1:0 – Bit Rate Policy

0 - Konstant bitrate. Bitraten som er definert i Vbit_rate_m og Vbir_rate_l registrene er konstant. Det blir gjordt forsøk på å hindre VBV overflow og VBV underflow. (se definisjoner for informasjon på VBV). Det blir satt et VBV underflow interrupt flagg dersom dette forkommer. Denne modusen bruker bit stuffing, som putter inn tomme bit, slik at bitraten hverken skal være mer eller mindre enn det som er definert

1 - Maksimal bit rate. Dette er nesten det samme som den over. Forskjellen er at denne ikke har bit stuffing, som gjør at bitraten kan være lavere enn det som er definert, men ikke høyere. VBV underflow og overflow forsøkes ikke å forhindres.

2 - Variabel bitrate. Dette valget har ingen restriksjoner på bitraten, men det velges en konstant kvalitet. Bitraten er da avhengig av kompleksiteten på det innkommende bildet, og valget i Vquality registeret.

3 - Reservert

Bit 2 – Half pixel on. Her velger man om prosessoren skal regne med ”halve” pixler når det komprimeres. Dette gjør at bevegelse estimeringen blir bedre, og dermed også bedre komprimering.

0 - Halv pixel av

1 - Halv pixel på

Bit 3 – B-bilde som startbilde. Dersom denne er valgt, så vil det første bildet i datastrømmen være et B-bilde.

0 - 'B' bilde som start av

1 - 'B' bilde som start på

Bit 4 – Closed GOB

0 - Open GOB. Dette betyr at GOBen komprimeres med bevegelsesvektorer som peker til forrige GOB.

1 - Closed GOB betr at man ikke har bevegelsesvektorer som peker til forrige GOB.

NB. Denne blir oversett hvis m verdien er mindre enn 2. Se Vframe_pattern.

Bit 5 – Inverse Telecine on
0 - Automatisk inverse telecine av.
1 - Automatisk inverse telecine på. Dette fører til at MPEG strømmen komprimeres med varierende frame rate. Dette er bare for 29,75 fps, så det berører ikke oss.
NB. Denne blir oversett hvis innsignalet er PAL

Bit 6 – Auto scene change. Dette trenger ikke vi, da vi har bare ett kamera.
0 - Automatisk scenebytte deteksjon av.
1 - Automatisk scenebytte deteksjon på. Denne sørger for at det blir startet en ny GOB dersom det kommer en ny scene. Dette kan være nyttig i den forstand at man slipper dårlige bilder på begynnelsen av scener på grunn av at de egentlig skulle inneholde forskjellen fra det forrige bildet. På denne måten blir det ett nytt 'I' bilde hver gang det kommer et totalt annerledes bilde.

Bit 7 – Time Stamp. Dette er hovedsakelig for synkronisering av lyd og bilde. Når denne er valgt, så vil hvert bilde ha tiden i headeren.
0 - Time stamp av
1 - Time stamp på

Vi kommer her til å bruke 0x0D når vi har MPEG komprimering. Ingen timestamp, ikke automatisk scenebytte deteksjon, ikke inverse telecine, ikke lukket GOB, 'B' som startbilde, halv pixel nøyaktighet, og maksimal bitrate.
Dersom vi arbeider med JPEG, så må bit 0:1 være 0x2.

Vframe_pattern (0x14)

Dette har to funksjoner. I MPEG komprimering bestemmer registeren hva slags mønster de foskjellige bildetypene i MPEG formatet skal komme i. Det finnes tre typer bilder i MPEG video, 'I', 'B' og 'P'. Mer informasjon om dette under definering av GOB.

Dersom man bruker MJPEG, så bestemmer registeret hvor mange bilder det skal være mellom hvert bilde som komprimeres. Man kan for eksempel velge å sende bare hvert femte bilde dersom man ønsker det.

Dette registeret er både lesbart og skrivbart, og bestemmer hva slags mønster bildene skal forekomme i. Reset verdi er 0x04 (m = n = 1).

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
n/m [5]	n/m [4]	n/m [3]	n/m [2]	n/m [1]	n/m [0]	m-1 [1]	m-1 [0]
0	0	0	0	0	1	0	0

figur 4.22 Vframe_pattern registeret

I live video MPEG1 format velger registeret bildemønsteret.

Bit 1:0 – ”m-1” verdien. Dette velger hvor mange ’B’ bilder det skal være mellom to ankerbilder.

0 - Ingen ’B’ bilder. (m = 1)
1 - Ett ’B’ bilde. (m = 2)
2 - To ’B’ bilder. (m = 3)
3 - Tre ’B’ bilder. (m = 4)

Bit 7:2 – ”n/m” verdien. Denne verdien velger distansen mellom to ’I’ bilder dividert på avstanden mellom to ankerbilder.

For eksempel, dersom ”m-1” = 0x2, så er m = 3. Hvis ”n/m” da er satt til 0x5, så vil det hvert 3 * 5 = 15. bilde være ’I’ bilde. Bildesekvensen vil da se slik ut : IBBPBBPBBPBBPBBIBBPBBPBBPBBPBBI

NB. N og GOB header frekvens(GHF, Vgob_header registeret) må oppfylle følgende formel: n <= 1024 / GHF

I live video encoding i MJPEG, og i live video pass trough, vil registeret fungere slik:

Bit 1:0 – Reservert

Bit 7:2 – Definerer avstanden mellom bildene som skal komprimeres/sende videre. Gyldige verdier er 0x01 til 0x3F (= 63). Det er opp brukeren å sørge for at dataraten ut ifra MPEG encoderen ikke overstiger den fra PCM encoderen.

Vbit_rate_m (0x15), Vbit_rate(0x16)

Disse registrene er både skrivbare og lesbare. Reset verdien er 0x0A00, der Vbit_rate_m er den mest signifikante byten, og Vbit_rate_l er den minst signifikante byten.

I dette registeret så definerer man bitrate grensen til utdata fra MPEG encoderen. Dette gjøres i steg på 400bps. Gyldige verdier er som i følgende tabell.

Frame size	MPGE1 ‘I’ frame Live video encoding		MPEG1 ‘IBP’ Live video encoding	
	Bit rate (Kbps)	Registers values (hex)	Bit rate (Kbps)	Registers values (hex)
SIF	1536 - 9216	0F00 - 5A00	192 - 6144	01E0 - 3C00
QSIF	512 - 3072	0500 - 1E00	64 - 2048	00A0 - 1400

figur 4.23 Forskjellige bitrater ved forekjellige formater

Vi ser at vi har store utfoldelsesmuligheter. Det er dessverre enkelte ting vil må ta hensyn til.

W99200 kjernen kan gi ut en høyere bitrate enn den som er valgt. Dette vil føre til at det blir satt et VBV underflow flagg.
Den faktisk produserte bitraten kommer an på hva slags bitrate restriksjoner vi har valgt.
Dersom variabel bitrate er valgt, blir disse registrene oversett.
Dersom konstant eller maksimal bitrate er valgt, vil den produserte bitraten være avhengig av størrelsen på VBV og kan ikke være høyere enn VBV multiplisert med antall bilder pr. sekund.

Vbv_size (0x17)

Dette registeret kan man både skrive til og lese fra. Reset verdien i dette registeret er 0x14

Dette registeret skal inneholde en verdi som angir bufferet i mottakeren. Dette vil i de fleste tilfeller være en rent teoretisk verdi. Se under defineringen for mer informasjon rundt VBV.
Verdien i dette registeret angir hvor stor VBV bufferet må minst være i mottaker for å i kunne dekomprimere videostrømmen.
Bufferstørrelsen er definert som B = 16384 * Vbv_size.

Det kan være nyttig å legge merke til at man uansett ikke kan få en høyere bitrate enn størrelsen på VBV ganget med antall bilder pr sekund.

Registeret blir oversett hvis ikke alle de følgende utsagnene er sanne.
Arbeidsmodus er Live video encoding
Video format er MPEG1
Bitrate begrensningen er enten konstant bitrate eller maksimal bitrate

Vbv_initial (0x18)

Dette registeret fungerer på samme måte som Vbv_size, men det definerer hvor mye av VBV som må være fylt opp før mottaker begynner å dekomprimere data. Verdien her må altså være lik, eller mindre enn Vbv_size. Dette registeret blir også oversett med mindre det oppfyller de samme betingelsene som for Vbv_size.

Vquality (0x19)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x08

I dette registeret setter man den kvaliteten man vil ha på bildene. Det vil derfor ikke ha noen effekt dersom man har valgt noe annet enn variable bitrate, eller man komprimerer ett enkelt bilde.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Quality [4]	Quality [3]	Quality [2]	Quality [1]	Quality [0]
0	0	0	0	1	0	0	0

figur 4.24 Vquality registeret

Bit 4:0 – Bilde kvalitet
0x00 – Reservert
0x01 – Høyeste kvalitet
.....
.....
0x1F – Laveste kvalitet
Bit 7:5 – Reservert

Vquality registeret er i bruk dersom en av disse utsagnene er sanne
Arbeidsmodus er Single frame encoding
Arbeidsmodus er Live video encoding og encoding format er MJPEG
Arbeidsmodus er Live video encoding og encoding format er MPEG1 og variabel bitrate er valgt.

Når man arbeider med konstant kvalitet på bildene vil den produserte bitraten være avhengig av kompleksiteten til det innkommende bildet. Det blir derfor opp til mottaker å sørge for at bitraten ikke overstiger båndbredden.

Vquality (sammen med Vin_picture) registeret kan forandres mens man er midt i en komprimeringsprosess. Man bør derfor polle statusen til FIFO for å sjekke om denne begynner å bli full. Dersom det er tilfelle, så må man senke kvaliteten for å unngå en overflow.

Vslice_header (0x1A)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x01

Dette registeret bestemmer hvor ofte det skal komme en slice header. Slice header er en header som inneholder informasjon som hjelper dekodere å synkronisere. En slice forekommer mellom et antall macroblokker. Vanligvis er det en hel linje med makroblokker, men det kan være mer. Gyldige verdier er derfor fra 1 til antall rader med macroblokker i ett bilde, i henhold til bildets format og størrelse.

Det er viktig at det kommer headere i datastrømmen, da det vil sørge for at bitfeil blir raskt rettet opp av mottaker. Dersom det da blir en feil i overføringen, kan mottaker bare lese videre til det kommer en header, og fortsette dekodingen der.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Repeat 'B' frames	Reserved	Slice Header [5]	Slice Header [4]	Slice Header [3]	Slice Header [2]	Slice Header [1]	Slice Header [0]
0	0	0	0	0	0	0	1

figur 4.25 Vslice_header registeret

Vi ser at dette registeret også har en annen funksjon. Dersom Repeat 'B' frames er satt, så vil B bildene ikke encodes, men være de samme som foregående bilde. Det vil si at vi får eh høyere fps med mindre prosessorkraft. Samtidig får vi en lavere bitrate. Desverre vil det gå vesentlig ut over flyten i bildet, og det vil oppleves som hakking.

Bit 5:0 – Slice header frekvens

Velger hvor her hvor mange raders mellomrom det skal være mellom slice headerene. Gyldige verdier er fra 1 til antall rader med macroblokker i ett bilde, i henhold til bildets format og størrelse.

Bit 6 – Reservert

Bit 7 – Repeat 'B' pictures

Dersom denne er satt, så vil alle B bilder være kopier av forrige bilde.

Vgob_header (0x1B)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x01

Dette registeret beskriver hvor ofte to forskjellige headere vil forekomme. Sequence header og GOB header. Som beskrevet i forrige register, så er det viktig med headere i datastrømmen for å forhindre at bitfeil vil ødelegge hele dekomprimeringen.

Sequence headeren inneholder følgende informasjon

- Bildestørrelse
- Sideforhold
- Bilde-rate og bit-rate (PAL har 25 bilder per sekund)
- Kvantiserings matrise. Denne er ikke nødvendig. Beskrivelse av hva kvantiseringsmatrise er for noe er tung teori og utelates.
- Krav om decoder buffer størrelse
- Croma pixel struktur.
- Eventuell brukerdata

GOP headeren inneholder følgende:

- Tidskode informasjon
- Editerings informasjon
- Eventuell brukerdata

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Sequence Header [3]	Sequence Header [2]	Sequence Header [1]	Sequence Header [0]	GOP Header [3]	GOP Header [2]	GOP Header [1]	GOP Header [0]
0	0	0	0	0	0	0	1

figur 4.26 Vgob_header registeret

Bit 3:0 – GOP header frekvens.
Inneholder ett tall (la oss kalle det k) som forteller at det skal være GOP header i hvert k'te 'I' bilde. Gyldige verdier 0x1 til 0xF

Bit 7:4 – Sequence header frekvens
Inneholder ett tall (la oss kalle det s) som forteller oss at det skal være sequence header ved hver s'te GOP header. Verdien 0x0 sier at det skal kun være sequencce header på begynnelsen av dataoverføringen. Gyldige verdier er 0x0 til 0xF.

Dersom inverce telecine eller scene change detection er aktivert vil det kunne forekomme midlertidige forandringer i avstanden mellom GOP og sequence headerene i bitstrømmen.

NB. N(Vframe_pattern registeret) og GOP header frekvens(GHF) må oppfylle følgende formel: n <= 1024 / GHF

Vit_hour (0x1C)

Dette registeret kan man både skrive til og lese fra. Reset verdi er 0x00.

I dette registeret, og de tre følgende kan man legge inn tiden. Dette er nødvendig når man skal synkronisere med lyd, men kan sikkert være nyttig til andre ting også. For eksempel kan de fortelle om når en lagret videosnutt er filmet. Registrene må skrives til før encodingen starter.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Hours [4]	Hours [3]	Hours [2]	Hours [1]	Hours [0]
0	0	0	0	0	0	0	0

figur 4.27 Vit_hour registeret

Bit 4:0 – Tiden i timer. Lovlige verdier 0x00 til 0x17 (= 23).

Bit 7:5 – Reservert

Vit_minute (0x1D)

Dette registeret kan man både skrive til og lese fra. Reset verdi er 0x00.

Se Vit_hour (0x1C) for info.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Minutes [5]	Minutes [4]	Minutes [3]	Minutes [2]	Minutes [1]	Minutes [0]
0	0	0	0	0	0	0	0

figur 4.28 Vit_minute registeret

Bit 5:0 – Tiden i minutter. Lovlige verdier 0x00 til 0x3B (= 59).

Bit 7:6 – Reservert

Vit_second (0x1E)

Dette registeret kan man både skrive til og lese fra. Reset verdi er 0x00.

Se Vit_hour (0x1C) for info.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Seconds [5]	Seconds [4]	Seconds [3]	Seconds [2]	Seconds [1]	Seconds [0]
0	0	0	0	0	0	0	0

figur 4.29 Vit_second registeret

Bit 5:0 – Tiden i sekunder. Lovlige verdier 0x00 til 0x3B (= 59).

Bit 7:6 – Reservert

Vit_frame (0x1F)

Dette registeret kan man både skrive til og lese fra. Reset verdi er 0x00.

Se Vit_hour (0x1C) for info.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Frames [4]	Frames [3]	Frames [2]	Frames [1]	Frames [0]
0	0	0	0	0	0	0	0

figur 4.30 Vit_frame registeret

Dette registeret har grenser etter hva slags bilderate du har på datastrømmen din. Siden PAL er 25 bilder per sekund, så er gyldige verdier fra 0x00 til 0x18 (= 24). Dersom det benyttes bilderater som ikke er heltall, så vil det kompenseres for ved at man hopper over enkelte verdier. Siden det ikke er relevant for vår oppgave, henviser vi til komplett datablad som kan fås hos Winbond.

Bit 4:0 – Hvilket bildet det er innen sekundet.

Bit 7:5 – Reservert

Vin_offset (0x20)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x00.

Her kommer vår konfigurering av og for dekoderen inn. Dette registeret skal nemlig inneholde avviket til odd og even field i det innkommende bildet.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
EFO [3]	EFO [2]	EFO [1]	EFO [0]	OFO [3]	OFO [2]	OFO [1]	OFO [0]
0	0	0	0	0	0	0	0

figur 4.31 Vin_offset registeret

Bit 3:0 – Odd fiels offset. (OFO)
Signed integer (-8 .. 0) som indikerer offseten (i antall linjer) for ”odd field” signalet, i forhold til default linje plassering (default linjeplassering er linje 7 i PAL, linje 10 i NTSC. Minimum verdi for PAL er –6).

Bit 7:4 – Even field offset. (IFO)
Signed integer (-8 .. 0) for NTSC og (-7 .. 0) for PAL, som indikerer offseten (i antall linjer) for ”even field” signalet, i forhold til default linje plassering (default linjeplassering er linje 320 i PAL, linje 273 i NTSC).

Vin_cntl (0x21)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x0C.

Dette registeret har også forbindelse med video dekoderen. Det er forskjellige standarder på forskjellige video dekodere. Dette registeret forteller om de forskjellige signalene fra video dekoderen er aktivt lave eller aktivt høye.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	VIVS# input select	SCR generation clock	VIHS# sampled level	VIDV polarity	VIHACT polarity	VIVS# polarity	VIHS# polarity
0	0	0	0	1	1	0	0

figur 4.32 Vin_cntl registeret

Bit 0 – VIHS polaritet.
0 - Aktiv lav. Fallende flanke indikerer ny videolinje (standard)
1 - Aktiv høy. Stigende flanke indikerer ny videolinje

Bit 1 – VIVS polaritet. (Dette registeret har forkjellige funksjoner, og har sammenheng med bit 6.)
0 - Dersom bit 6 er ’0’ – Fallende flanke indikerer begynnelsen på en ”nytt field”.
Dersom bit 6 er ’1’ – Fallende flanke indikerer begynnelsen på en ”odd field”.
1 - Dersom bit 6 er ’0’ – Stigende flanke indikerer begynnelsen på en ”nytt field”.

Dersom bit 6 er ’1’ – Stigende flanke indikerer begynnelsen på en ”odd field”.

Bit 2 – VIHACT polaritet.
0 - Aktiv lav. Når lav indikeres aktiv region av linjen.
1 - Aktiv høy. Når høy indikeres aktiv region av linjen.

Bit 3 – VIDV polaritet
0 - Aktiv lav. Når lav indikeres godkjent pixel kan samples.
1 - Aktiv høy. Når høy indikeres godkjent pixel kan samples.

Bit 4 – VIHS sampel nivå
2. ”Odd field” er indikert med at VIHS pin høy samplet med VIVS blir lav

Vin_picture (0x22)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x00.

I dette registeret kan man velge om man ønsker å legge på en logo på bildet. Man har også muligheten til å kutte noen linjer av bildet. Det er for å slippe å få linjeg i bunnen av bildet som bare inneholder søppel. Endel kameraer har en tendens til å ikke bruke hele bildet, og i slike tilfeller kan det være praktisk.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Bitmap insertion	Blanked video lines
0	0	0	0	0	0	0	0

figur 4.33 Vin_picture registeret

Bit 0 – Blanked Video lines
0 - (default) Ingen linjer blir kuttet
1 - De siste 16 linjene i FULL oppløsning, de siste 8 i SIF og de 4 siste i QSIF blir blanke.

Bit 1 – Bit-map insertion
0 - (default) Det blir ike satt inn et bit-map bilde.
1 - Det blir satt inn et bit-map bilde på alle bilder i datastrømmen. Dette kan for eksempel være en firmalogo.

Dette registeret gjelder bare dersom modusen er Live video encoding, Live video pass trough eller Live video snap shot.

Bit 1 kan forandres midt i overføringsprosessen. Forandringen gjør seg ikke gjeldene før førstkommende bilde.

Vmem_select (0x23)

Dette registeret kan man både skrive til og lese fra.

Registeret har to funksjoner:
1 - I skriv til eller les fra SDRAM, velger dette registeret hvilken del av SDRAMen det skal skrives til eller leses fra. SDRAMen er delt opp i 256 sider av 4K * 16 bit hver. Gyldige verdier er 0x00 til 0xFF
2 - I skriv til eller les fra internt minne modus, så velger dette registeret det interne minnet, eller registrene som det skal skrives til i deler på 256 bytes.

Vsize_h (0x24)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x2C (= 44)

I Write frame, Read frame og Single frame encoding modusene angir dette registeret den horisontale størrelsen på bildet i enheter av 16 pixels. Default verdien er derfor $16 * 44 = 704$ linjer i horisontal retning. Lovlige verdier er 0x02 til 0x2D (2 til 45).

I alle andre arbeidsmoduser blir disse registrene oversett.

Vsize_v (0x25)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x1E (= 30)

Dette registeret fungerer akkurat som Vsize_h bare at det definerer den vertikale størrelsen på bildet. Default verdien er $16 * 30 = 480$ linjer i horisontal retning. Lovlige verdier er 0x02 til 0x24 (2 til 36).

Vfifo_status (0x30)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x00

Vi skjønner ikke helt hvorfor dette registeret er skrivbart. Registeret indikerer hvor full FIFO listen er. Man kan på den måten finne ut om man må senke dataraten (f.eks ved Vquality) for å ikke overstige båndbredden.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Reserved	Full [3]	Full [2]	Full [1]	Full [0]
0	0	0	0	0	0	0	0

figur 4.34 Vfifo_status registeret

Bit 3:0 – FIFO fullhets indikator
0 - FIFO er mindre enn 1/16 full
1 - FIFO er mellom 1/16 og 2/16 full
2 - FIFO er mellom 2/16 og 3/16 full
.....

.....
F - FIFO er over 15/16 full

Bit 7:4 – Reservert

I2c_status (0x31)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x00

Dette registeret indikerer om forrige I2C kommando ble fullført uten feil.

[7]	[6]	[5]	[4]	[3]	[2]	[1]	[0]
Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	Reserved	I2C Succeeded
0	0	0	0	0	0	0	0

figur 4.35 I2c_status registeret

Bit 0 – I2C Succeeded
- 0 - Forrige kommando mislykkes.
- 1 - Forrige kommando ble fullført korrekt. Dersom det var en lesekommando, så vil det nå ligge data i I2c_data registeret

Bit 7:1 – Reservert

PCR (Processor Control Register) (0x66)

Dette registeret kan man både skrive til og lese fra. Reset verdien er 0x64. Det kan kun resettes gjennom hardware reset.

Her kontrolleres de viktigste hardware funksjonene.

bit(7)	bit(6)	bit(5)	bit(4:3)	bit(2)	bit(1)	bit(0)
SDRAM_TRI	DE_TRI1	DE_TRI0	(not used)	BY_PASS	PD	SRST
0	1	1	0	1	0	0

figur 4.36 PCR registeret

Bit 0 – SRTS (Software Reset)
- 0 - Software reset av
- 1 - Software reset på

Man bruker denne biten til å resette hele chipen. Man må sette det til 1 og så 0 igjen.

Bit 1 – PD (Power Down)
- 0 - Chipen fungerer som normalt
- 1 - Klokkepulsen stenges, og chipen arbeider ikke. Strømsparing.

Bit 2 – BY_PASS (Valg av klokkekilde)

0 - Intern klokke lages av PLL output

1 - Intern klokke lages av direkte fra MCLK input pin

Det er en intern krest som fordobler MCLK og sender det videre som PLL. Dersom man har 27 MHz på MCLK, så må man her velge 0. Å velge 1, for så å senke MCLK kan være en idé dersom man ønsker å senke dataraten ved single frame encoding. Vi vet ikke om det fungerer, men det burde gjøre det.

Bit 4:3 – Reservert

Bit 6:5 – DE_TRI0, DE_TRI1 (Brukt til å sette tri-state på utgangene fra VCD dekoder)

Disse bitene blir brukt til å kontrollere en eventuell VCD dekodersom deler SDRAM med MPEG encoderen. Disse er ikke aktuelle for oss, og vi setter dem til hva vi vil.

Bit 7 – SDRAM_TRI (Brukt til å sette signalene til SDRAM i tri-state)

0 - Tri-state av

1 - Tri-state på

Dette har egentlig ingen funksjon dersom vi ikke deler minnet med en annen krets, for eksempel VCD dekoder. Vi setter derfor denne til '1'.

4.4 Programtekniske løsninger

I dette avsnittet skal vi beskrive hvordan vi har løst oppgaven rent programmessig. Vi benytter en PIC prosessor fra MicroChip, og koder i C. Alle programmene som er beskrevet her er kompilert, men ikke testet. Grunnen til det er at vi ikke har kretsene vi trenger for å gjennomføre testingen. Koden skal fungere slik den er, men vi kan ikke garantere det.

Vi kommer først til å gå gjennom hva programmet må gjøre i de forskjellige enkodingsmodellene vi ha, nemlig live video encoding, live snap shot og single frame encoding. Vi vil så gå gjennom en enkel pseudokode for de forskjellige programdelene slik at det skal bli enklere å forstå koden. Deretter vil vi forklare i detalj hvordan vi har skrevet kode. Dersom hele koden ønskes samlet, så henviser vi til vedlagt kildekode. Den koden er ikke så grundig beskrevet som det vi kommer til å gjøre her. Det er for å bedre oversikten i koden. Man kan lese under avsnitt 4.3.4 for mer informasjon på oppstartsprosedyrer, dataoverføringsprosedyrer, og hvilke kontrollord som må konfigureres.

4.4.1 Pseudokode for oppstart

- Vente til klokkepulsen har stabilisert seg. Vi har ikke hastverk, så vi venter ett halvt sekund.
- Setter PCR registeret
- Følger opp med konfigurering etter hva slags modus som skal kjøres.

4.4.2 Pseudokode for live video overføring

Vi henviser til figurer i avsnitt 4.2.1 for henvisninger på signaler og oppkobling.

- Overføring startes (sender start signal)
- Setter adressen til 0x00
- Går i løkke
    - Venter på FIFO_RDY
    - FIFO_RDY settes
    - Venter på førstkommende lave Gate fra PCM encoderen.
    - Setter aktiv FIFO_RD (C)
    - Setter høyt signal til OR (3) port fra mikroprosessor (E) inn på PCM encoderen for å markere start på datablokk.
    - Venter på neste lave Gate
    - Åpner AND porter
    - Setter lav signal til OR (3) port fra mikroprosessor (E).
    - Teller opp treshold-1 antall pulser på Gate.
    - Setter inaktiv FIFO_RD (C)
    - Venter på neste lave Gate
    - Stenger AND porter
- Stoppsignal mottas, sender stoppsignal til MPEG encoder
- Går i løkke
    - Venter på FIFO_RDY
    - FIFO_RDY settes
    - Venter på førstkommende lave Gate fra PCM encoderen.
    - Setter aktiv FIFO_RD (C)
    - Setter høyt signal til OR (3) port fra mikroprosessor (E) inn på PCM encoderen for å markere start på datablokk.
    - Venter på neste lave Gate
    - Åpner AND porter
    - Setter lav signal til OR (3) port fra mikroprosessor (E).
    - Teller opp treshold-1 antall pulser på Gate.
    - Setter inaktiv FIFO_RD (C)
    - Venter på neste lave Gate
    - Stenger AND porter
- Dersom ikke neste FIFO_RDY signal kommer innen ett halvt sekund, aktiveres Video end of data interrupt.
    - Venter på førstkommende lave Gate fra PCM encoderen.
    - Setter aktiv FIFO_RD (C)
    - Setter høyt signal til OR (3) port fra mikroprosessor (E) inn på PCM encoderen for å markere start på datablokk.
    - Venter på neste lave Gate
    - Åpner AND porter
    - Setter lav signal til OR (3) port fra mikroprosessor (E).
    - Mottar Video End Of Data interrupt.
    - AND porter stenges.
- Dataoverføring ferdig.

4.4.3 Pseudokode for live video snapshot og single frame encoding

- Setter arbeidsmodus til live video snap shot
- Velger hvor i minnet bildet skal lagres
- Sender Vstart kontrollord
- Setter arbsidsmodus til single frame encoding
- Sender Vstart kontrollord
- Går i løkke
    - Venter på FIFO_RDY
    - FIFO_RDY settes
    - Venter på førstkommende lave Gate fra PCM encoderen.
    - Setter aktiv FIFO_RD (C)
    - Setter høyt signal til OR (3) port fra mikroprosessor (E) inn på PCM encoderen for å markere start på datablokk.
    - Venter på neste lave Gate
    - Åpner AND porter
    - Setter lav signal til OR (3) port fra mikroprosessor (E).
    - Teller opp treshold-1 antall pulser på Gate.
    - Setter inaktiv FIFO_RD (C)
    - Venter på neste lave Gate
    - Stenger AND porter
    - Dersom ikke neste FIFO_RDY signal kommer innen ett halvt sekund, aktiveres Video end of data interrupt, og man hopper ut av løkken.
- Det ligger antageligvis femdeles litt data i FIFO. Nå skal det sendes.
    - Venter på førstkommende lave Gate fra PCM encoderen.
    - Setter aktiv FIFO_RD (C)
    - Setter høyt signal til OR (3) port fra mikroprosessor (E) inn på PCM encoderen for å markere start på datablokk.
    - Venter på neste lave Gate
    - Åpner AND porter
    - Setter lav signal til OR (3) port fra mikroprosessor (E).
    - Mottar Video End Of Data interrupt.
    - AND porter stenges.
- Dataoverføring ferdig.
- Gjentar prosessen. Dersom man har mulighet til å kontrollere prosessen fra bakken, vil man her kunne vente til det kommer en ”nytt bilde” kommando fra kontrollstasjon, før man fortsetter. På den måten vil man kunne sende bilde ”on demand”.

4.4.4 Pseudokode for I2C bus overføring

Vi går ut ifra at leseren er kjent med I2C busen. Det er beskrevet hvordan I2C busen fungerer under definisjonene.

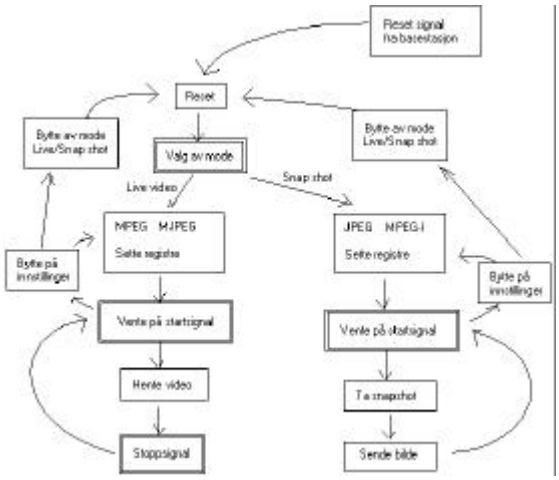
- Sender I2c_start kontrollord
  - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
  - Sjekker I2c_status registeret, for å se om overføringen var ok.
  - Dersom overføring ikke ok
    - Send I2c_stop kontrollord
    - Begynne overføringen på nytt
- Setter slave-addresse i I2c_data registeret
  - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
  - Sjekker I2c_status registeret, for å se om overføringen var ok.
  - Dersom overføring ikke ok
    - Send I2c_stop kontrollord
    - Begynne overføringen på nytt
- Setter sub-addresse i I2c_data registeret
  - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
  - Sjekker I2c_status registeret, for å se om overføringen var ok.
  - Dersom overføring ikke ok
    - Send I2c_stop kontrollord
    - Begynne overføringen på nytt
- Dersom man skal skrive data:
  - Setter data i I2c_data registeret
    - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
    - Sjekker I2c_status registeret, for å se om overføringen var ok.
    - Dersom overføring ikke ok
      - Send I2c_stop kontrollord
      - Begynne overføringen på nytt
- Dersom man skal lese data:
  - Setter I2c_start kontrollord
    - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
    - Sjekker I2c_status registeret, for å se om overføringen var ok.
    - Dersom overføring ikke ok
      - Send I2c_stop kontrollord
      - Begynne overføringen på nytt
  - Leser innhold i I2c_data registeret.
- Sender I2c_stop kontrollord
  - Poller I2c_ready i Vint_source registeret, for at data skal bli ferdig sendt.
  - Sjekker I2c_status registeret, for å se om overføringen var ok.
  - Dersom overføring ikke ok
    - Send I2c_stop kontrollord
    - Begynne overføringen på nytt

4.4.5 Oversikt over hvordan kildekoden er lagt opp

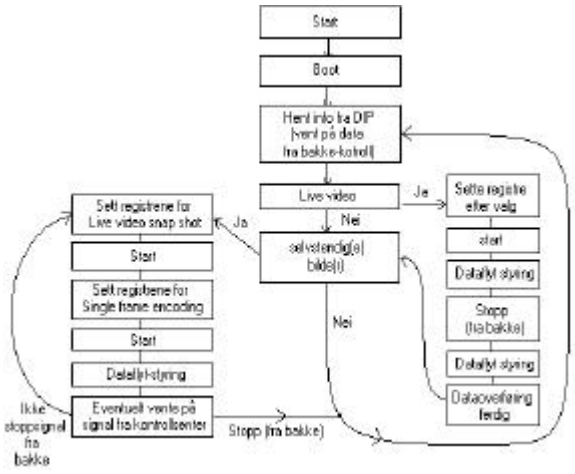
Kildekoden til programmet er relativt enkel. Det største problemet i prosjektet var ikke å få til et program som virker slik vil ville ha det. Problemet var å finne ut hvordan vi ville ha det.

Det er mange forskjellige måter å sette opp kildekoden på. En mulighet vi har til å legge opp kildekoden er å lage den som en slags tilstandsmaskin, der man hele tiden hopper mellom tilstander (se figur). Dette ville antakeligvis gjøre kildekoden både vanskelig å skrive og vanskelig å forandre på. Spesielt med tanke på de som eventuelt skal lage kretsen og må forandre kildekoden for å åpne for å motta kontrollord fra kontrollsenteret, eller andre ting. Fordelen ville være at man får et fast mønster, der det eksisterer et fast oppsett på hvilke kontrollord som blir forandret og overført til og fra de forskjellige delene. Dette høres veldig objektorientert ut, og passer ikke helt inn i C, som vi koder i.

Vi valgte å lage en kronologisk modell (se figur). Denne sørger for at alt som skjer kommer i riktig rekkefølge. Dersom man siden en gang får muligheten til å sende kontrollord fra bakken, vil man da kunne stoppe overføringen, hoppe tilbake til start, og begynne på nytt med de nye innstillingene. Siden dette programmet var forventet å bli ganske lite og veldig enkelt, så vil det ikke være noe problem for en mann å sette seg inn i kildekoden, og derfor ble dette oppsettet valgt.



figur 4.37 Tilstandsmaskin. Dobbeltfirkanter angir punkter der det kreves info fra DIP-switches eller basestasjon



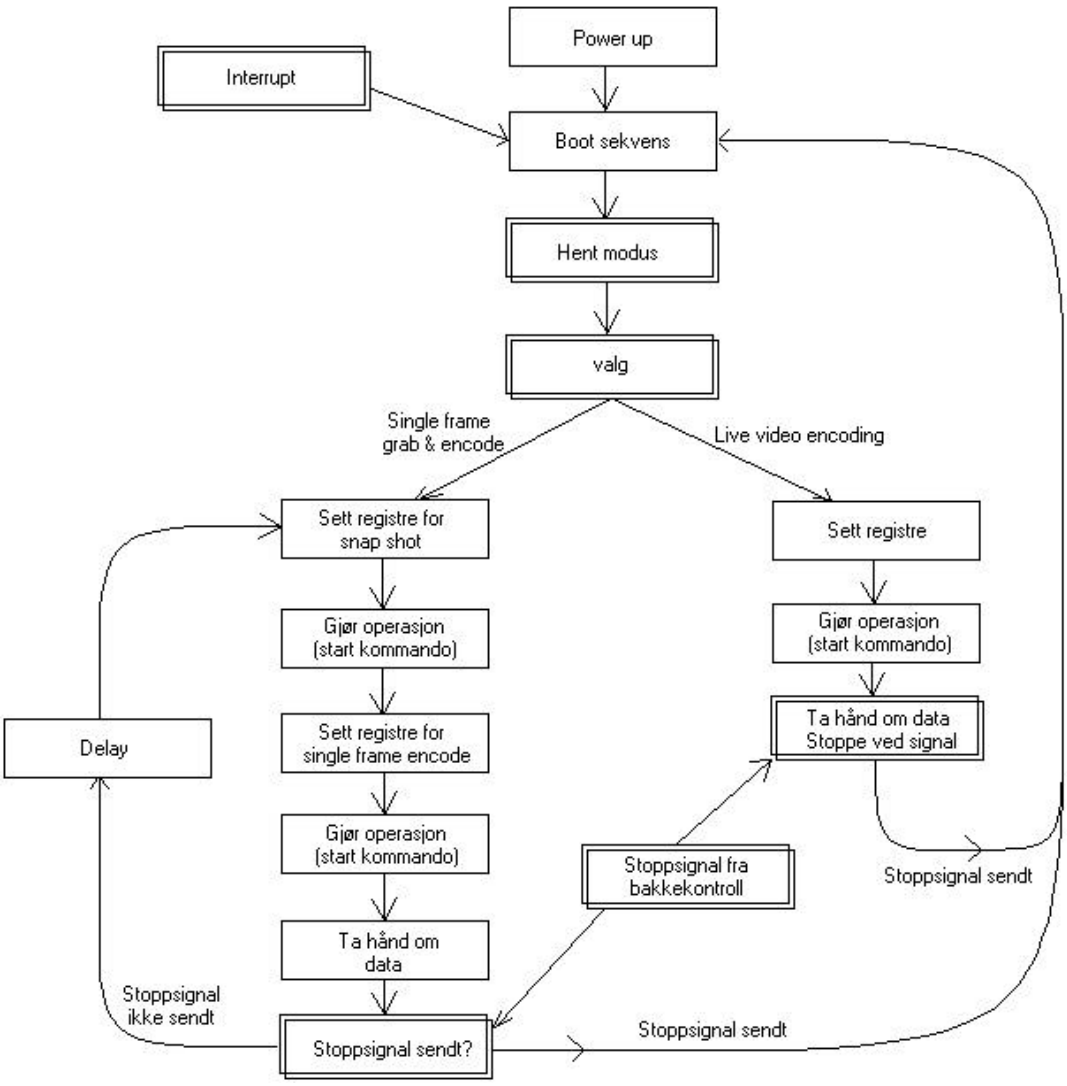
figur 4.38 Kronologisk modell. Her er det allerede lagt opp til bakkestyring.

4.4.6 Funksjonsbeskrivelse av kildekode

I og med at vi hadde skrevet pseudokode for det meste, så hadde vi ikke altfor store problemer med å skrive koden for PIC prosessoren. Av den grunn ble også koden fin og oversiktlig. Vi kommer likevel til å gå ganske grundig igjennom strukturen i koden vi har skrevet, slik at det ikke oppstår noen misforståelser. Selve kildekoden er også kommentert, men det har vi gjort på engelsk. Det som følger nå er en del blokkskjemaer som viser hva som skjer når, og i vanskelige partier også hvordan det skjer. Det er litt kildekode innimellom for å illustrere. Det kan kanskje være en ide å ha et bokmerke i kildekoden dersom oversikt over kildekoden er ønskelig.

4.4.6.1 Hovedoversikt

Alle firkanter som har dobbel firkant har enten ett valg å gjøre, og/eller er har med uregelmessige signaler å gjøre, for eksempel interrupt og styresignaler fra bakken.



figur 4.39 Hovedoversikt over programkoden

Det første som skjer er Boot sekvensen. Den skal beskrives litt nærmere senere. Deretter kommer Hent Modus. Denne skal også beskrives senere. Ut ifra hva som kommer frem i Hent Modus delen, så velger da programmet mellom de to retningene. En for live video encoding, og en for single frame encoding/grab.

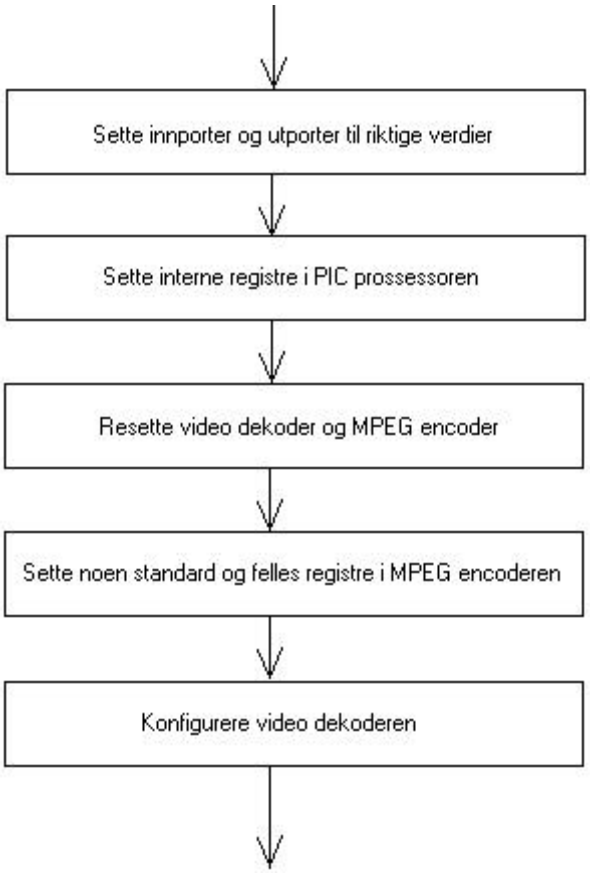
Deretter skjer ting sekvensielt. Registrene blir satt etter standardverdier eller etter Hent Modus sine innstillinger. For live video encodinga er det nå bare å starte encodinga, og fortsette til det eventuelt blir sendt ett stoppsignal.

Mottak av stoppsignal er ikke implementert, for det er per idag ikke mulig å motta signaler med PCM encodere. Vi har holdt mulighetene åpne for enkel forandring, slik at hvis det skulle komme en gang, så vil det være enkelt å implementere.

For single frame grab/encoding, må man først motta bilde, deretter encode det, og så håndtere datastrøm. Etter det, så vil man sjekke om det er kommet ett stopp-signal. Dersom det ikke er tilfelle, så vil man gå til en liten pause, for så å fortsette videre. Istedenfor den pausen, kan man sette inn en funksjon som venter på et ”ta bilde nå” signal fra bakkekontroll, dersom man ønsker å kunne ta bilder ”on demand”. Dette kan kanskje med fordel implementeres i stoppsignal sjekken istedenfor, for så å droppe hele pausen. Vitsen med en pause der er for å begrense kravet til båndbredde.

4.4.6.2 Boot sekvens

Boot sekvensen er relativt enkel. I hovedsak utfører den bare fem typer instruksjoner. Der er i og for seg helt nødvendige alle sammen.



Det første den gjør er å sette default verdiene til alle portene.

Det som så skjer, er at de interne registrene, og med det blant annet tellere, blir initialisert.

Deretter resettes video dekoderen og MPEG encoderen

For så å konfigurere MPEG encoderen

Den siste delen er relativt tung. Den består i å konfigurere video dekoderen. Den har bare en riktig innstilling, og på grunn av omfattende datablad valgte vi å bruke Winbond sine forhåndsvalgte verdier. Vi så liten vits i å sette seg grundig inn i den kretsen også, det er jo bare en måte som er riktig uansett.

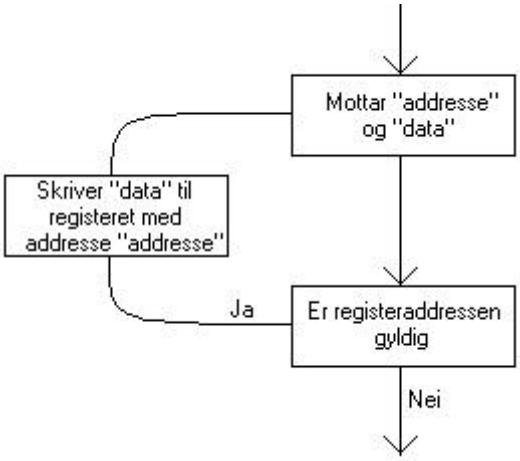
figur 4.40 Oversikt over Boot sekvensen

Som illustrert i oversiktsbildet, så går man gjennom Boot sekvensen hver gang man stopper en prosess. Det kan være at et eller annet register ikke er som det skal være, så vi velger å gå gjennom hele greia med initialisering hver gang. Da blir også alle kretsene resatt, slik at vi starter med nye blanke ark.

4.4.6.3 Get mode sekvensen

Dette er vel nesten den viktigste delen å legge merke til. Inne i denne funksjonen skjer det nemlig veldig lite. Vi tegner ikke engang opp et skjema på hva som skjer. Dersom det en gang blir mulig å sende kontrollord fra basestasjon, bakkekontroll, eller hvor det nå skal sendes fra, så skal de kontrollordene mottas her. Vi ser for oss at man da stiller inn alle registrene direkte fra bakken, og at man da ikke har noen få forskjellige kombinasjoner. Der er av den grunnen at vi har valgt å gå så grundig gjennom hvordan man konfigurerer MPEG encoderen.

Vi ser i all enkelhet for oss noe slikt som dette:



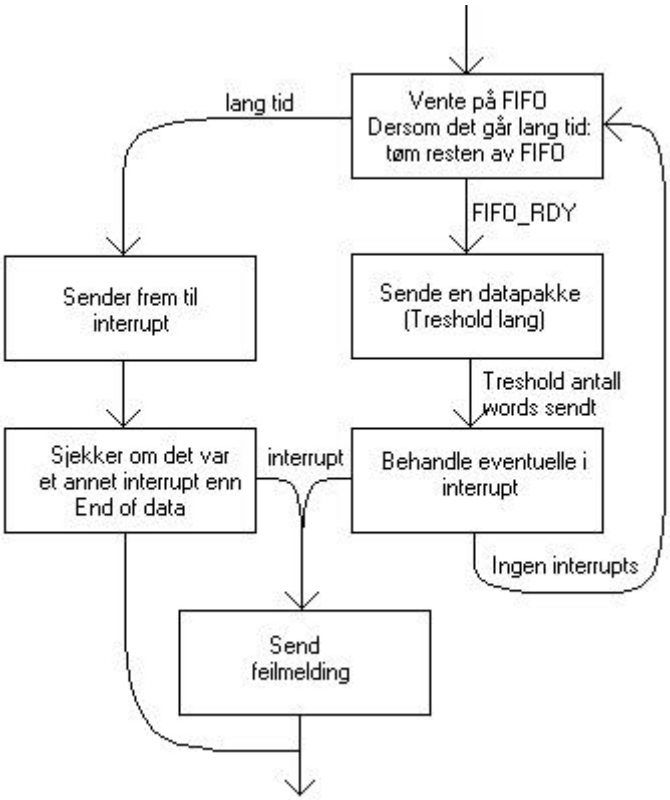
figur 4.41 Figur over hvordan vi tenker oss Get_info() funksjonen

Her mottar man først to bytes fra basestasjon. Deretter sjekker man om det er en gyldig adresse. Dersom man er ferdig med å konfigurere kan man da bare sende 0xFF for eksempel. Da vil man hoppe ut av løkken.

Dersom det er en gyldig adresse, så skriver man dataen man har mottatt til registeret med adressen man har mottatt. På den måten kan man for eksempel forandre datarate underveis.

4.4.6.4 Behandling av datastrøm

Det er litt forskjell på behandling av datasrømmen i de to modusen vi kjører. Vi beskriver først snap shot modellen.



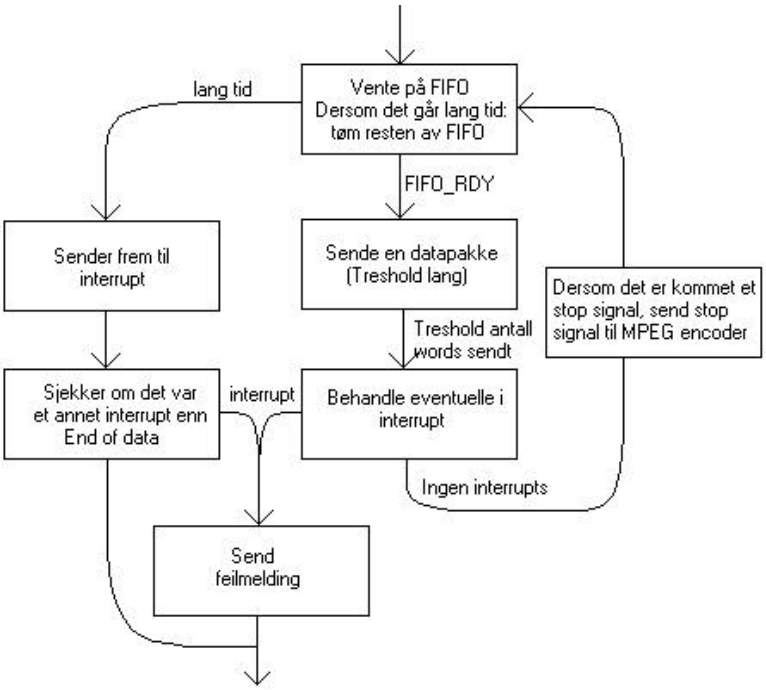
Dersom alt er som det skal, så sendes det et FIFO_RDY signal når det er på tide å lese fra FIFO listen. Når det signalet da kommer, så leses treshold antall bytes, for så å fortsette videre.

Det som skjer da, er at det sjekkes om det har kommet noen interrupts. Dersom det har skjedd, så er det kommet en eller annen feil, og feilen sendes til PCM encoderen på et definert format. (se gjennomgang av feilmelding i 4.4.5.5) Deretter hopper man ut av løkken, og returnerer en beskjed om at det har skjedd noe feil.

figur 4.42 Dette er hvodan vi forvalter data

Dersom man nærmer seg slutten på bildet/videoen, så vil kanskje ikke FIFO listen komme opp på et slikt nivå at det trigges en FIFO_RDY signal. Da må man bare gå ut ifra at hvis det ikke kommer et slikt signal innen en predefinert tidsluke, så må man klokke ut den gjenværende delen av dataen. For å si ifra når det ikke er mer data i FIFO listen, har MPEG encoderen et lite interrupt som sier Video end of data. Når dette interruptet kommer så bare stenger vi alle porter, sjekker om det er kommet noen andre interrupts, og hopper ut.

Ved live video, må man legge inn en sjekk om det er kommet et stop signal fra bakkekontrollen samtidig som man venter på FIFO_RDY signalet. Deretter må man sende ett stoppsignal til encoderen, og fortsette prosessen slik som for snap shot modusen.



Vi setter det inn på denne måten. Du vil se at vi har kommentert dette i kildekoden, men det er ikke implementert ennå, på grunn av at det ikke er støtte for å motta data i PCM encoderen ennå.

Dette er en enkel sak å implementere, men vi vet ikke med sikkerhet hvordan data vil kunne leses fra PCM encoderen, og vi velger derfor å overlate dette til våre etterfølgere.

figur 4.43 Slik ser vi for oss datastyring med bakkekontroll

4.4.6.5 Send feilmelding

Denne delen er ganske enkel. Det som gjøres er at man legger innporten på PCM encoderen høy og lav annenhver gang. Det vil si at mottaker mottar 0xFFFF og 0x0000 annethvert. Dersom man gjør dette 10 ganger, så kan man anta at det er et unikt mønster, og si at til 0xFFFF med 0x0000 imellom betyr at det har skjedd en feil.

For å gjøre det hele bedre så har vi valgt å legge data mellom 0xFFFF og 0x0000. Det vi legger inn er verdien på Vint_source registeret. På den måten kan mottaker selv lese av hva som gikk feil. Siden brikken kun er koblet til 8 av databusens 16 ledere, og lederne blir krysset, så vil mottaker kun få nyttig data i den høye byten.

Dataen som kommer til mottaker blir da seende noe slik ut (der X er søppel, og D er data fra mikroprosessoren):

...0x0000, 0x0000, 0x0000, 0xFFFF, 0xXXDD, 0x0000, 0xFFFF, 0xXXDD, 0x0000.....
..... 0xFFFF, 0xXXDD, 0x0000, 0x0000, 0x0000, 0x0000, ...

Det vil med andre ord komme ti stykker av sekvensen 0xFFFF, 0xXXDD, 0x0000. Lett å gjenkjenne i en tom datastrøm, og det blir vesentlig lettere å finne ut hvorfor videoen blir sendt riktig.

Denne funksjonen kan man gjøre mye mer ut av enn det vi har gjort. Blir programmet større, og mulighetene flere, så kan dette være en flott hjelp i debuging.

5.0 Konklusjon

Vi brukte endel tid på å se etter forskjellige kretser. Vi fant ikke så mye som kunne være aktuelt for oss. Det meste var både for stort, for dyrt eller tålte for lav temperatur.

Vi fant ut av at vi skulle bruke Winbond kretsen. Den både inneholdt det vi trengte av hardware, samtidig som den lå innenfor alle krav til fysiske påkjenninger og størrelser. Det at kretsen bruker lite strøm er bare en bonus.

Etter litt om og men, og mye snakk, fikk vi endelig tak i riktig person hos Winbond sin avdeling i Norge. Der fikk vi beskjed om at det var lang leveringstid, men at vi kunne kanskje få et testeksemplar eller to. Da åpnet vi alle muligheter, og tok kontakt med både IBM og leverandører i andre land, som for eksempel Nu Horizon og All American Semiconductors. Dette var resultatløst. Det samme var forsøket på å få en slik krets tilsendt fra en Winbond avdeling i et annet kontinent. Det hele var veldig utmattende og til tider svært frustrerende. Vi fikk tilbud om levering til Norge, men aldri med mindre enn 12 ukers leveringstid, noe som var langt fra akseptabelt.

Da vi endelig mottok datablad fra Winbond sin avdeling Norge, gikk vi ut ifra at vi skulle kunne få tak i et testeksemplar likevel, og satset alt. Vi kunne ikke bruke mer tid på å se etter muligheter. Vi gikk til verks, og leste datablad og kildekode som vi fikk på CD fra Winbond. Vi arbeidet jevnt og trutt, og hadde klar store deler av oppgaven. Vi fikk dat beskjed fra Winbond Norge at brikken ikke er tilgjengelig i Europa på grunn av manglende support. Det var ingen mulighet for å gå rundt dette. Vi valgte derfor å konsentrere oss om å få alt til å fungere i teorien. Vi mener at vi har fått det til på en tilfredsstillende måte. Dersom man lager et kretskort og lodder på brikkene, og kjører en mikrokontroller med vårt program på, så tror og håper vi at det vil fungere med en gang.

Vi bestemte oss også for at programmet skulle implementeres på en måte som gjør det enkelt å utvide programmet til å ta imot kontrollord fra PCM encoderen. Dette er ikke ennå implementert, men det ble hintet fra FFI om at det kunne komme til å komme. Siden vi ikke visste dataformat eller noe slikt så har vi lagt alt opp til å være klargjort for det meste.

Det var ønske fra FFI at vi skulle benytte OrCad i tegningen av kretskort. Etter litt prøving og feiling innså vi at de versjonene som lå tilgjengelig på nett ikke holdt mål. Vi måtte ha mulighet til å ha flere komponenter enn det den utgaven tillot. Vi valgte derfor å bruke noe vi var vant til. Valget ble da Proteus.

Dersom man skulle bestille kretsene og prøve vår løsning, er det endel ting man kan legge merke til. Vi har implementer en ”Send_error()” funksjon i mikrokontroller-koden vår. Denne kan helt sikkert bli betydelig bedre enn det den er. Det er enkelt å legge inn mulighet for en langt mer omfattende feilmelding, og det kan forenkle testeprosessen betraktelig. Noe som også ville øke brukervennligheten og fleksibiliteten formidabelt er å føye til mulighet for å sende kontrollord fra bakken. Hvis vi hadde hatt muligheten ville dette vært det neste vi skulle fått gjennomført.

Alt i alt så er vi fornøyd med resultatet. Det har vært vanskelig og utfordrende, spesielt med tanke på at vi ikke hadde noen måte å sjekke at det vi gjorde var riktig. Det ble veldig mye studering av signalgang og styresignaler, samt mye lesing av datablad og kontrollord.

Vi har lært mye, spesielt rundt komprimering av video. Vi har også fått endel erfaring med å lese seg frem til løsninger på problemer som har med signalgang å gjøre. Dette har absolutt vært noe av det mest lærerike som har skjedd på Høgskolen, og vi takker FFI for at vi fikk ta dette utfordrende og vanskelige oppdraget.

6.0 Vedlegg

6.1 Kvalitet ved forskjellige datarater og oppløsninger

Vi har tatt for oss endel bitrater og komprimert en videosnutt i disse. Vi her gjort dette med forskjellig oppløsning også, for å illustrere forskjellen ved de forskjellige oppløsningene man kan velge med brikken.

Vi har komprimert MPEG i 250Kbit/s, 500Kbit/s, 1Mbit/s, og 2Mbit/s. Oppløsningene har vært både 352*288 og 192*144. Vi presenterer her noen bilder. Det er også mulig å nyte filmklippene dersom du har tilgang på CDen. Vi viser i store bilder for å få frem svakhetene mer tydelig.

Vi bemerker at det i videoene viste seg at bildene på lav datarate ble bedre og bedre frem til det kom et nytt 'I' bilde. Det dere ser her er det første bildet i videoen. I gjennomsnitt så var videoen bedre enn det som vises her.

For at man skulle lettere kunne sette de forskjellige bitratene opp mot hverandre, så har vi satt to bilder pr. side. Det er fire bilder på hver oppløsning.