Innholdsfortegnelse

Figurfortegnelse

Tabellfortegnelse

1. INNLEDNING

I det vi går mot tusenårsskiftet med ekspressfart øker kravene til troverdig gjengivelse av bevegelige bilder med minst mulig sløsing av båndbredde. Med det som utgangspunkt utviklet den internasjonale standardiseringsorganisasjonen en standard for "koding av bevegelige bilder og tilhørende lyd for digital lagring ved opp til 1,5 Mbit/s" [ISO 11172].

Institutt for datateknikk og informasjonsvitenskap har i flere år bedrevet forskning på JPEG video som er basert på koding av enkeltbilder etter JPEG-standarden. Denne standarden benytter ikke båndbredden like effektivt som den nye standarden (ISO 11172), som også kalles MPEG-1.

I motsetning til JPEG-video kodes bildene (også kalt rammer) i MPEG-1 ut fra forskjellen mellom nærliggende bilder. På denne måten spares mye plass på mediet og båndbredden kan reduseres betraktelig (typisk ca. 1/20 sammenlignet med ukomprimert data) uten visuelt tap. Ytterligere reduksjon kan oppnåes, men man må da regne med en visuell degradering av signalet.

MPEG-1 signalstrømmer er ikke like enkle å redigere som JPEG-video. MPEG-1 er som nevnt, kodet ut fra forskjellen mellom nærliggende rammer i begge retninger, slik at et rått kutt i strømmen ville føre til uventede effekter p.g.a. at minst en av referanserammene til rammene det kuttes ved er blitt fjernet.

1.1 Oppgaven

I samråd med faglærer Roger Midtstraum og veileder Olav Sandstå er det kommet frem til at denne oppgaven skal videreføre og komplettere det tidligere arbeidet utført i vårprosjektet 1996: "Editering av MPEG-video" [EDITER96] som tok for seg redigering av videodelen av MPEG-1-standarden.

Denne oppgaven skal presentere en løsning som integrerer det tidligere utførte arbeidet med det som i denne oppgaven må kalles hovedproblemet: redigering av MPEG-audio.

Oppgaveløsningen skal komme fram til en metode for redigering av MPEG-audio ved først å presentere MPEG-1-standarden og deretter en evaluering av noen løsningsalternativer med én av disse som den valgte løsningen.

Problemet går ut på å redigere MPEG-1-audio (ISO 11172-3) på vilkårlig plass. I likhet med det tidligere løste problemet for MPEG-1-video (ISO 11172-2), er det ikke mulig å bare gå inn i en audiostrøm og klippe den på grunn av temporær avhengighet i begge retninger i rammesekvensen. I likhet med MPEG-1 video har også audiodelen avhengighet i begge retninger.

Andre del av problemet er en mulighet til å redigere MPEG-1 systemstrømmer på vilkårlig plass, men ikke nødvendigvis samme plass for audio- og videodelen. Problemet blir satt sammen til ett problem: konstruere et redigeringssystem for MPEG-1 system (ISO 11172-1).

1.2 Organisering av denne rapporten

De delene i denne rapporten som omhandler video er basert på arbeid gjort våren 1996 i [EDITER96].

Kapittel 2: MPEG-1 Video

Dette kapittelet inneholder en kort introduksjon til digital video. Deretter kommer et forstudium som tar for seg dagens situasjon. ISO 11172 beskrives. Kapittelet har et underkapittel om MPEG-1 System (ISO 11172-1), et om MPEG-1 Video (ISO 11172-2) og et om MPEG-1 Audio (ISO 11172-3). I dette kapittelet nevnes også MPEG-2.

Kapittel 3: Problembeskrivelse

Et kapittel med beskrivelse av problemet.

Kapittel 4: Alternative løsningsmodeller

Neste kapittel inneholder tenkte løsninger. Dette kapittelet ender med en valgt løsning med tilhørende begrunnelse for valget.

Kapittel 5: Overordnet design og konstruksjon:

Deretter følger en overordnet beskrivelse av valgt løsning og gjennomføringen av denne.

Kapittel 6: Implementasjonsbeskrivelse av systemdelen

Detaljert beskrivelse av systemdelen. Inneholder designbeskrivelse av hver funksjon.

Kapittel 7: Implementasjonsbeskrivelse av videodelen

Detaljert beskrivelse av videodelen. Inneholder designbeskrivelse av hver modul, klasse og funksjon.

Kapittel 8: Implementasjonsbeskrivelse av audiodelen

Detaljert beskrivelse av audiodelen. Inneholder designbeskrivelse av hver modul, klasse og funksjon.

Kapittel 9: Evaluering

Deretter kommer et kapittel om evaluering. Dette kapittelet evaluerer resultatet som kommer ut i andre enden av programmet (den ferdige system- og audiostrømmen). Det inneholder også en evaluering av tids- og ressurskrav programmet har for gjennomføring av en editering. Audio-editoren og systemoverbygningen (audio + video) blir evaluert hver for seg. En kort gjengivelse av audioeditoren fra [DYBVIK96] er også tatt med.

Kapittel 10: Oppsummering og konklusjon

Helt til slutt kommer et kapittel med oppsummering og konklusjon. Dette kapittelet inneholder et avsnitt med forslag til videre arbeid som kan forbedre/utvide produktet.

2. MPEG-1-STANDARDEN

2.1 Innledning

Digital video ble først innført for å redusere støyen som vanligvis blir introdusert ved redigering. Ved studioredigering benyttes det ingen komprimering, eller tapsløs («lossless») komprimering, dvs. at nøyaktig det samme signalet (video og/eller audio) blir rekonstruert ved dekoding.

Den tapsløse komprimeringen tar svært stor plass. Som et botemiddel mot dette benyttes ikke-tapsløse («lossy») komprimeringer for det meste. Det finnes mange varianter, f.eks. QuickTime™ fra Apple Computers® og MPEG.

MPEG finnes i tre utgaver. MPEG-1 er hovedsakelig utviklet for CD-ROM, men er også godt egnet for overføring i nett med lav båndbredde. Denne standarden gir VHS-kvalitet og har en båndbredde på opp til 1,5 Mbit/s. MPEG-2 er utviklet for digital kringkasting med muligheter til HDTV-kvalitet. MPEG-2 er best egnet til overføring i nett med høy båndbredde. Denne standarden kan gi meget god kvalitet med en båndbredde mellom 2 og 15 Mbit/s. MPEG-4 er beregnet for svært lave båndbredder. Den kan brukes til f.eks. telekonferanser over ISDN.

Støy under formidlingen av digital video er så godt som ikke eksisterende p.g.a. forskjellige feilkorrigeringsmekanismer.

2.2 Hva er MPEG

MPEG er en forkortelse for Motion Picture Experts Group [ISO 11172], en arbeidsgruppe under International Standards Organization (ISO), som er ansvarlig for standarder innen video- og audio-komprimering.

MPEG inneholder spesifikasjoner for komprimering og integrering av både audio og video. MPEG-1 ble utviklet spesifikt for lagring på CD-ROM og DAT, men brukes i dag til mye mer.

MPEG benyttes for å kode en serie bilder (film) hvor bildene er kodet etter et gitt mønster. Hvert bilde er kodet internt (spatialt), men også i forhold til kommende og/eller foregående bilder (temporært). Det finnes fire forskjellige bildetyper i en MPEG bildestrøm: I-, B-, P- og D-rammer. D-rammer benyttes nesten ikke, og vil derfor ikke bli gått inn på i dette prosjektet.

MPEG audio er den tilhørende lydstandarden. Den kan kodes i inntil tre nivåer (layers) med økende kompleksitet og komprimeringsgrad. I nivå 1 og 2 blir ikke (lyd)rammene kodet i forhold til hverandre. Det vil si at rammene er selvstendige moduler. I nivå 3 blir intrarammekoding benyttet med økt komprimeringsrate som resultat. Intrarammekoding øker kompleksiteten på kodingen vesentlig.

2.3 MPEG-1 System

Systemdelen (ISO 11172-1) er overbygningen som benyttes for å synkronisere og koordinere (samordne) strømmer med forskjellig innhold. Disse strømmene kan inneholde MPEG-1 video (ISO 11172-2), MPEG-1 audio (ISO 11172-3), en strøm med lyd som følger en annen standard, f.eks. Dolby AC-3 eller DTS eller hvilken som helst annen "privat" strøm (f.eks. underteksting). MPEG-1-strømmen kan inneholde flere "kanaler" med audio, video eller "private" strømmer.

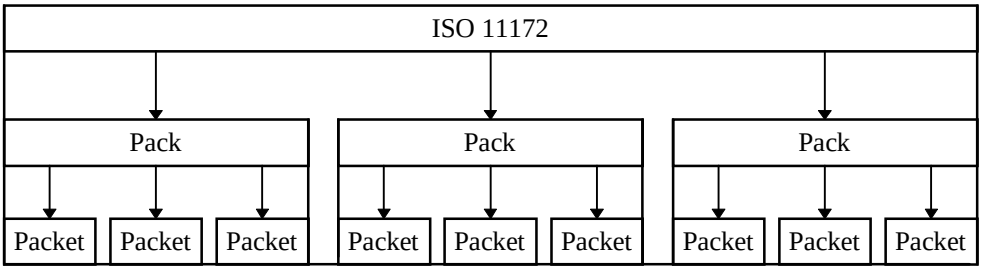
2.3.1 System datahierarki

Overordnet oppbygging av datastrømmen kan sees i Figur 1. Denne figuren er en sekvensiell oversikt over hvordan en MPEG-1 systemstrøm kan være oppbygd. En systemstrøm starter med en systemheader som definerer en del dekodingsparametre.

System-header	Datapakke strøm 1	Datapakke strøm 2	Datapakke strøm 1

Figur 1: Overordnet oppbygging av systemstrømmen (sekvensiell).

Hierarkisk oppbygging av strømmen er illustrert i Figur 2. Helt på topp ligger ISO 11172-laget. Dette laget består av ett eller flere sideordnede Pack-lag. Pack-laget består av ett eller flere sideordnede Packet-lag som igjen består av et pakkedatalag. Pakkedatalaget er de individuelle strømmene som er multiplekset inn i systemstrømmen.



Figur 2: MPEG-1 system datahierarki.

Hvert lag har sin egen startkode i bitstrømmen slik at det blir enkelt å gjenkjenne de enkelte lagene.

Et fellestrekk for alle start-/stopp-kodene som ligger i en MPEG-1-strøm er at de må starte på hele byteposisjoner i strømmen. Får å få til dette kan det være nødvendig å drive «bitstuffing», dvs. legge inn flere biter som har verdien '0' for å nå en hel byteposisjon før startkodene kan legges inn i strømmen.

2.3.2 Pack-laget

Pack-laget ligger rett innenfor ISO 11172-laget. En systemstrøm kan inneholde flere sekvensielle (sidestilte) Pack-lag.

Pack-laget inneholder "System Clock Reference", som er antatt ankomsttid for den siste byte av "System Clock Reference" til dekoderen. Denne er nærmere forklart i 2.3.5.4.

Etter at "System Clock Reference" er lest inn, leses en eventuell systemheader inn (se 2.3.3). Den første Pack i strømmen *må* inneholde en systemheader. Til slutt leses

gjentatte Packets (se 2.3.4) inn så lenge dekoderen møter på “Packet Start Code Prefix”.

2.3.3 Systemheaderen

Systemheaderen ligger tidlig i systemstrømmen bare med Pack-lagets definisjoner foran.

Systemheaderen inneholder en del definisjoner som benyttes av dekoderen. Den setter et maksimumstall for antallet ISO 11172 audiostrømmer og videostrømmer, hvorvidt strømmene følger de begrensende parametre, om bitraten er fast eller variabel og hvor stor buffer som trengs for hver av de multipleksede strømmene.

Som nevnt i avsnittet om Pack-laget (se 2.3.2), kan en systemstrøm inneholde flere sidestilte Pack-lag. Den kan derfor med andre ord også inneholde flere systemheadere med nye dekodingsdefinisjoner.

2.3.4 Packet-laget

Under Pack-laget ligger Packet-laget. I dette laget ligger datapakkene til de individuelle strømmene. Dette laget skiller ut disse pakkene og sender de til sine respektive dekodere (audiopakker til audiodekoderen, videopakker til video-dekoderen osv.).

I begynnelsen av pakken ligger informasjon om hvilken type strøm det er (f.eks. audio R (høyre), audio L (venstre) eller video), lengden på pakken (i antall byte), størrelsen på bufferen, presentasjonstidspunktet til første elementet i datapakken (“Presentation Time Stamp”; PTS) og dekodingstidspunktet for det første elementet i datapakken (“Decoding Time Stamp”; DTS).

Etter disse definisjonene ligger pakkedata inntil en ny startkode kommer. Lengden på pakken kan også regnes ut fra lengdedefinisjonen.

2.3.5 Tidskoder

I MPEG-1 system benyttes flere 33-bits tall for synkronisering av pakkene fra de forskjellige strømmene. Disse tallene er kodet i tre separate felt á 11 bit. Tidsinformasjonen ligger forut for alle datapakker og i begynnelsen av Pack-headeren.

2.3.5.1 Systemklokkefrekvens

Systemklokkefrekvensen må tilfredsstille følgende betingelser [ISO 11172]:

$$90\,000 - 4{,}5 \leq \text{systemklokkefrekvens} \leq 90\,000 + 4{,}5$$

$$\text{avviksrate for systemklokkefrekvensen} \leq 250 * 10^{-6}\ \text{Hz/s}$$

Systemklokkefrekvensen benyttes for både System Clock Reference (SCR), Presentation Time Stamp (PTS) og Decoding Time Stamp (DTS). Disse verdiene er nøyaktig ned til 2^{33}/systemklokkefrekvens [ISO 11172]. Årsaken til dette er at verdiene kodes med 33 biter.

Det antas for alle verdier at en perfekt dekoder/koder benyttes slik at ingen ledd introduserer noen som helst forsinkelse.

2.3.5.2 Presentation Time Stamp

Presentation Time Stamp (PTS) forekommer i Packet-laget. Det representerer *presentasjonstidspunktet* for den første presentasjonsenheten i pakken. PTS blir sendt til klokka i dekoderen for denne pakken (f.eks. audio eller video). Denne dekoderen (audio- eller videodekoderen) sørger for at pakkens første presentasjonsenhet blir presentert når systemklokka er det samme som PTS.

2.3.5.3 Decoding Time Stamp

Decoding Time Stamp (DTS) forekommer i Packet-laget. Det representerer antatt *dekodingstidspunkt* for den første presentasjonsenheten i pakken.

2.3.5.4 System Clock Reference

System Clock Reference (SCR) forekommer i Pack-laget. Det representerer antatt ankomsttid for siste byte i SCR til systemdekoderen. Antatt ankomsttid for de øvrige bytes følger av ligning (1) [ISO 11172]:

$$tm(i) = \frac{SCR(i')}{system\_clock\_frequency} + \frac{i - i'}{(mux\_rate \cdot 50)}$$

(1)

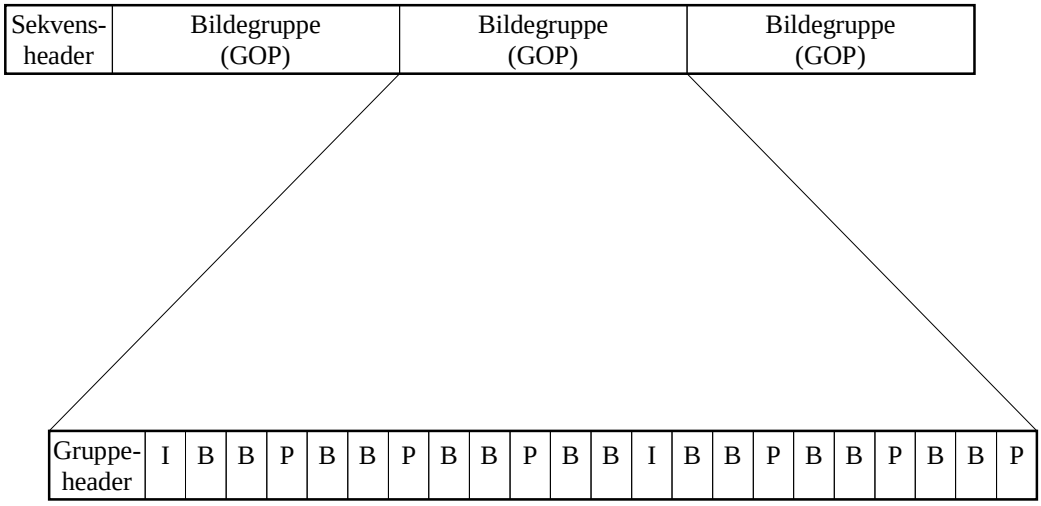
Denne ligningen sier når byte *i* (telt fra og med pack-headeren) ankommer systemdekoderen når *i'* er siste byte i SCR, *SCR(i')* er verdien i SCR som ligger i Pack-headeren, *system_clock_frequency* er systemklokkefrekvensen og *mux_rate* er hastigheten i antall 50 bytes/sekund dataene ankommer dekoderen (ligger i pack-headeren).

2.4 MPEG-1 Video

Videodelen er her beskrevet som definert av ISO 11172-2. Beskrivelsen er basert på eget arbeid utført våren 1996 [DYBVIK96] og er tatt med for kompletthet.

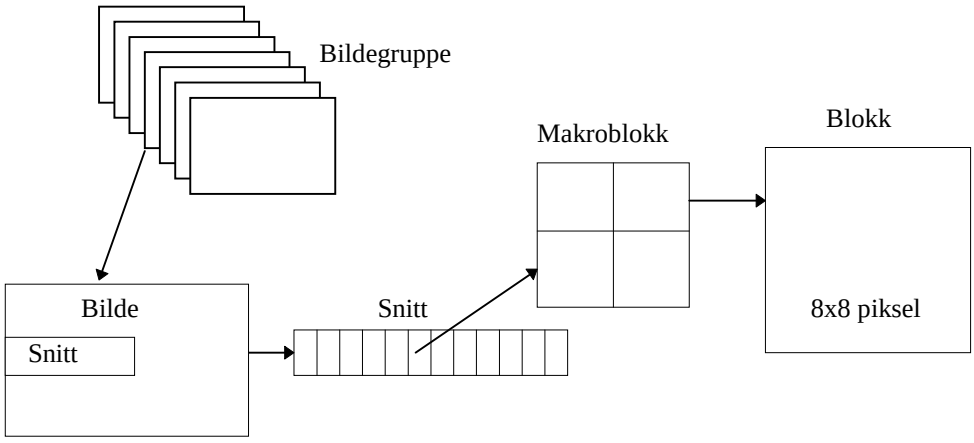
2.4.1 Video datahierarki

Overordnet oppbygging av datastrømmen kan sees i Figur 3. Sekvensheaderen ligger fremst i datastrømmen, deretter kommer det en eller flere grupper med bilder (GOP). Bildegruppene er bygd opp av bildetyper som vist i figuren. En videostrøm kan ha flere sekvensheadere [ISO 11172], men disse må ligge rett foran en gruppe. Helt til slutt kommer en sekvens-stoppkode.



Figur 3: Overordnet oppbygging av videostrømmen.

Datahierarkiet er beskrevet i Figur 4 [HODGE95]. Toppnivået er hele videostrømmen. Deretter kommer bildegruppene som består av en eller flere rammer. Hvert bilde (ramme) består av flere snitt («slice»). Hvert av disse snittene er igjen delt opp i makroblokker som igjen er delt opp i blokker bestående av 8x8 piksler.



Figur 4: MPEG-1 video datahierarki (hentet fra [HODGE95]).

Videosekvensen består av en sekvensheader, en eller flere bildegrupper og en sekvens-stoppkode. En bildegruppe er definert som en serie av ett eller flere bilder med mulighet for direkte aksess, for å muliggjøre editering eller direkte aksess i videostrømmen.

Et bilde består av tre rektangulære matriser; en for luminans (Y – gråtoner) og to for krominans (CrCb – farge). Y-matrisen er dobbelt så stor som hver av de to C-

matrisene. Cr er matrisen for røde komponenter, og Cb er matrisen for blå komponenter i bildet.

Et snitt er en samling med sammenkoblede makroblokker. Rekkefølgen på makroblokker i snittet er fra venstre mot høyre og fra topp til bunn.

En makroblokk er en 16x16 pixel matrise med Y-komponenter, og to 8x8 pixel matriser med henholdsvis Cb- og Cr-komponenter. En makroblokk består av fire Y-blokker, en Cr- og en Cb-blokk. Se Figur 4 for illustrasjon.

En blokk er en 8x8 pixel samling av luminans- og krominanskomponenter. MPEG-standarden [ISO 11172] tar hensyn til at det er mye informasjon som er felles fra bilde til bilde innen hver blokk. Dette gjøres ved å kode enkelte bilder med bare differansen til andre bilder.

2.4.2 Sekvensheaderen

Sekvensheaderen ligger helt i begynnelsen av videostrømmen, og når som helst i strømmen foran en gruppeheader dersom noen dekodingsparametre skal endres. Sekvensheaderen forteller dekoderen hvilke metoder som er benyttet for kodingen av den påfølgende strømmen. Disse parametrene er gyldige inntil en ny sekvensheader kommer. MPEG-1-standarden [ISO 11172] tillater flere sekvensheadere i en strøm, forutsatt at de kommer rett før en gruppeheader.

Sekvensheaderen setter størrelsen på rammene (høyde x bredde), aspektrate, bildehastighet, bitrate (kan være variabel), om «begrensede parametre» fra standarden [ISO 11172] følges (se Tabell 4) og om standard intra og/eller nonintra kvantiseringsmatrise er benyttet ved koding (hvis ikke skal matrisene ligge definert i strømmen).

2.4.3 Sekvens-stoppkode

Sekvens-stoppkoden ligger på slutten av videostrømmen. En del kodere legger gjerne inn en del 0-biter før denne koden. Når en dekoder møter på denne koden, stopper all dekoding opp, og dekoderen antar at strømmen er ferdig dekodet. Stoppkoden kan med andre ord stoppe avspillingen av en MPEG-1 video selv om den inneholder flere rammer.

2.4.4 Bildegrupper

Bildestrømmen deles gjerne opp i flere grupper med bilder (Group of Pictures, GOP). Disse kan defineres som åpen eller lukket. Disse gruppene er optimalisert hver for seg og representerer selvstendige enheter (forutsatt at gruppen er definert som lukket). En gruppe starter gjerne med en I-ramme som referanseramme for de følgende rammene. En gruppe kan bestå av en eller flere I-rammer og ingen eller flere P- og B-rammer. Hele strømmen kan også være definert som én gruppe, men dette blir svært upraktisk når vi jobber med lange videostrømmer.

Gruppeheaderen inneholder informasjon om når første rammen i gruppen skal vises og om gruppen er en lukket eller åpen gruppe.

2.4.4.1 Lukket gruppe

Lukket gruppe er den typen som er standard [ISO 11172]. Det vil si at hver gruppe er uavhengig av andre grupper. En gruppe kan plukkes ut av strømmen og dekodes separat. Dekodingsparametre hentes likevel fra sekvensheaderen.

2.4.4.2 Åpen gruppe

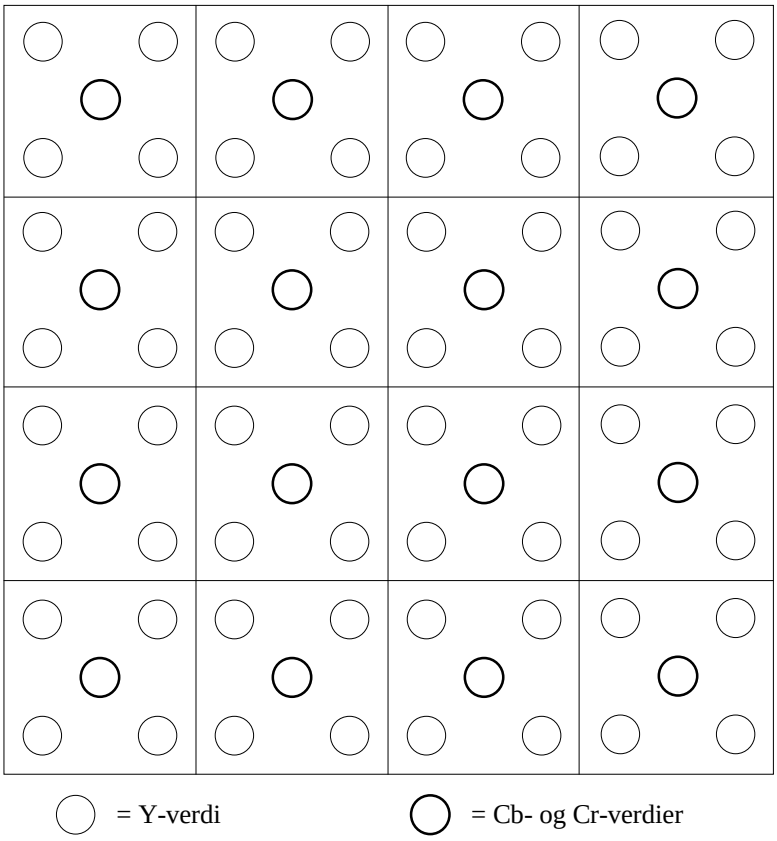
Åpen gruppe benyttes når rammer er kodet utfra rammer i andre grupper. Dette kan spare mye plass, men reduserer fleksibilitet fordi en gruppe ikke kan plukkes ut av strømmen og dekodes separat.

2.4.5 I-rammer

I-rammer (Intrakodet) representerer starten på bildestrømmen og ethvert vilkårlig aksesspunkt. I-rammene inneholder all informasjon som skal til for å bygge opp et bilde, dvs. at de ikke er kodet i forhold til noen andre bilder. Normalt opptrer I-rammene omtrent to ganger i sekundet. Bildet kodes etter en algoritme som er svært lik JPEG-koding [FLUCK95] av enkeltbilder.

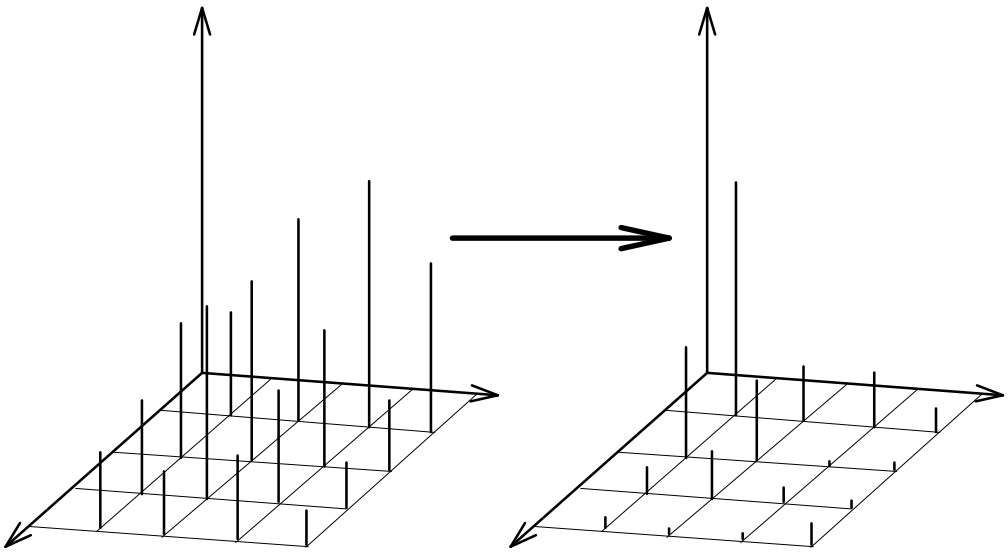
Kodingen av rammeinnholdet kan oppsummeres på denne måten [FLUCK95]:

- Luminans (gråtoner) og de to fargedifferanseplanene deles opp i blokker med 8×8 pixel.



Figur 5: Sammenhengen mellom Y-, Cr- og Cb-verdier (hentet fra [HODGE95]).

- Hver blokk transformeres til frekvensplanet ved hjelp av DCT (Discrete Cosine Transform) som vist i Figur 6 og Figur 7.



Figur 6: Amplitudene i en blokk

Figur 7: DCT-koeffisienter av en blokk.

Begge figurene: [FLUCK95].

- Bildet kvantifiseres ved hjelp av en kvantifiseringstabell. Som resultat blir enkelte koeffisienter neglisjert. Eksempel kan sees i Tabell 1, Tabell 2 og Tabell 3. Formel for utregning av $q_c(i,j)$ [FLUCK95]*:*

$$q_c = \frac{c(i,j)}{Q(i,j)}$$

168	45	7	3	2	2	1	1
67	32	3	3	2	1	1	1
12	5	5	5	2	1	2	1
5	5	2	2	1	1	1	1
3	2	2	1	1	1	1	1
2	2	2	1	1	1	1	0
1	1	1	1	2	1	1	0
1	2	1	1	1	1	0	0

Tabell 1: DCT koeffisienter, c(i,j) (hentet fra [FLUCK95]).

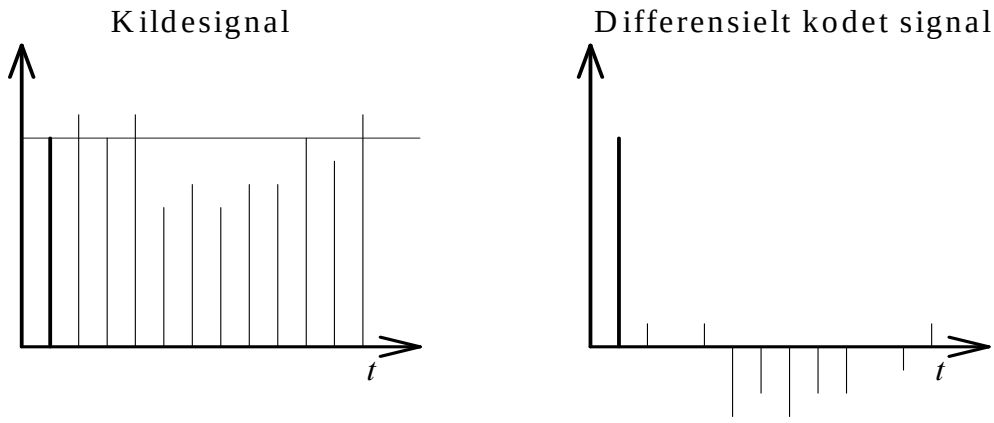
1	1	1	1	1	4	8	16
1	1	1	1	4	4	8	16
1	1	1	2	4	4	8	16
2	8	8	8	8	16	16	16
4	8	8	8	8	16	16	32
4	8	8	8	16	16	16	32
4	8	8	8	16	16	32	32
8	8	8	16	16	32	32	64

Tabell 2: Kvantiseringsmatrise, Q(i,j) (hentet fra [FLUCK95]).

168	45	7	3	2	0	0	0
67	20	3	3	2	0	0	0
12	5	5	2	0	0	0	0
2	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

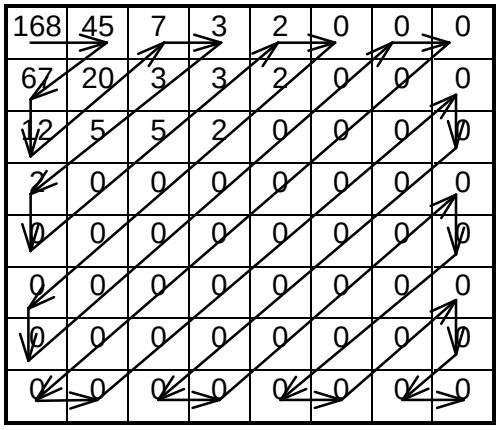
Tabell 3: Kvantifiserte DCT koeffisienter, $q_c(i,j)$ (hentet fra [FLUCK95]).

- De mest signifikante koeffisientene i hver blokk, DC koeffisientene, kodes ved hjelp av en DPCM-teknikk – bare forskjellen mellom to nærliggende DC-verdier kodes. Eksempel på en enkel DPCM-teknikk er vist i Figur 8.



Figur 8: Eksempel på simpel DPCM-teknikk ([FLUCK95]).

- Koeffisientene i hver blokk sikksakkordnes som vist i Figur 9, og blir deretter «run-length»-kodet.

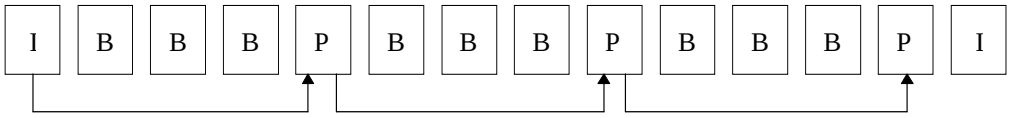


Figur 9: Sikksakkordning (hentet fra [FLUCK95]).

- Til slutt blir en Huffman-lignende koding brukt.

2.4.6 P-rammer

P-rammer (Predictive) kodes etter hvordan man forutser rammen skal se ut. P-rammer kodes bare etter forutgående rammer. Metoden er illustrert i Figur 10.



Figur 10: Referansekoding av P-rammer.

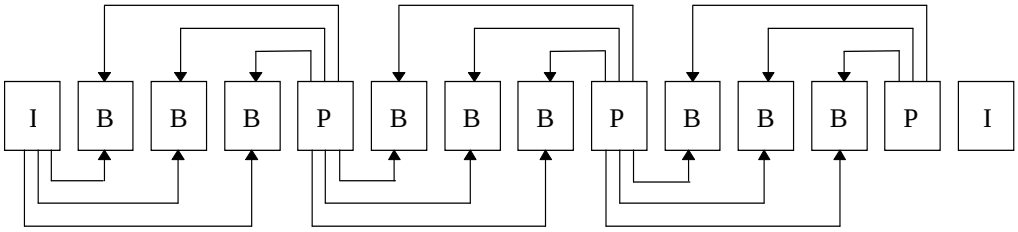
Kodingen av P-rammer kan oppsummeres på denne måten [FLUCK95]:

- For hver makroblokk søkes det i referanserammen etter den makroblokken som passer best.
- Differansen mellom den faktiske makroblokken og den som passer best blir beregnet. I tillegg blir bevegelsesvektoren[1] (motion vector) beregnet.
- Feiltermen, som også er en makroblokk, transformeres ved hjelp av DCT.
- Et kvantifiseringssteg, «sikksakk ordning», "run-length" og Huffman koding blir utført til slutt. Kvantifiseringstabellen er forskjellig fra den som brukes for I-rammer. I motsetning til I-rammer og JPEG-bilder kodes DC-koeffisientene på samme måte som de andre koeffisientene.
- Bevegelsesvektoren for hver blokk kodes ved hjelp av en DPCM-teknikk, fordi nærliggende bevegelsesvektorer har som oftest bare små variasjoner i verdi. Det resulterende datasettet gjennomgår til slutt en Huffman-koding.

2.4.7 B-rammer

B-rammene (Bi-directional) kodes i forhold til andre bilder i datastrømmen. Det er tre forskjellige måter å kode disse på (Figur 11):

[1] I visse tilfeller i en B-ramme, kan to bevegelsesvektorer knyttes opp mot en makroblokk, en i hver retning.



Figur 11: Referansekoding av B-rammer.

1. Reverse: Bildet kodes etter et bilde som kommer før dette bildet i datastrømmen.
2. Forward: Bildet er kodet i forhold til et bilde som kommer etter dette bildet i datastrømmen.
3. Bi-directional: Bildet er kodet i forhold til et bilde som kommer før og et som kommer etter dette bildet i datastrømmen.

Hvilken av disse metodene som benyttes er avhengig av hvilken som genererer minst datamengde. Kodingen foregår i forhold til referanserammer som for B-rammer er P- eller I-rammer.

Metoden kodingen blir utført etter er forøvrig den samme som for P-rammer.

2.4.8 D-rammer

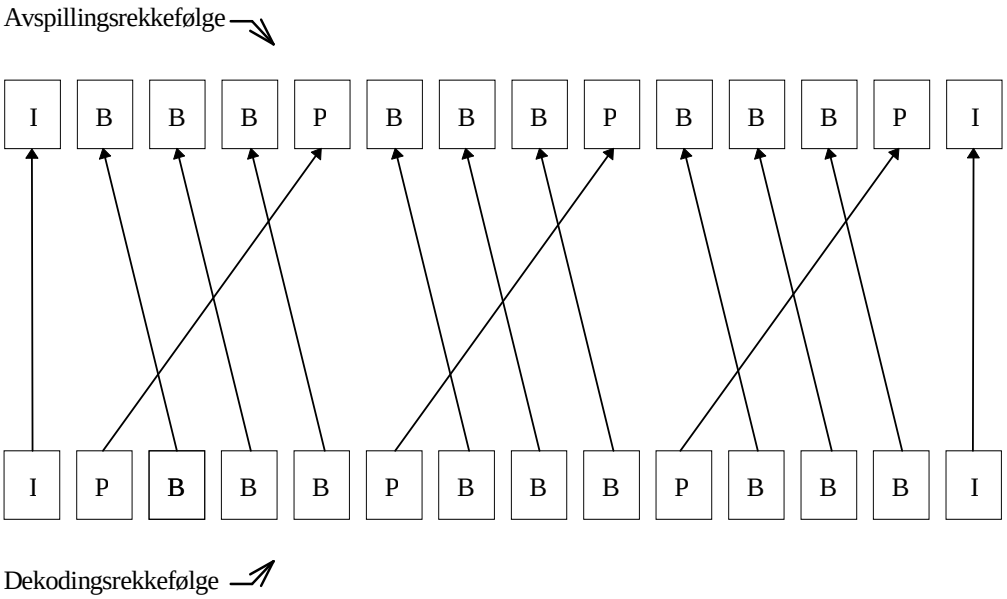
D-rammer (DC-Intrakodet) er referanserammer som legges inn først og fremst for å lette avspilling av en videostrøm i valgfri hastighet. De er kodet utfra informasjon fra seg selv. D-rammer blir *svært* sjelden brukt. D-rammer må aldri kombineres med andre rammetyper i samme videosekvens (kfr. [ISO 11172] pt.2, side 16).

2.4.9 Video datastrøm-komposisjon

Figur 10 viser hvordan P-rammer kodes i forhold til andre rammer. P-rammer benytter I-rammer som referanserammer for kodingen. P-rammer benytter bare reverskoding, dvs. at et bilde bare kodes etter bilder som har vært vist.

Figur 11 viser hvordan B-rammer kodes i forhold til andre rammer. For B-rammene fungerer P-rammene som referanserammer i tillegg til I-rammene. B-rammene kan, som nevnt, kodes utfra referanserammer i begge retninger.

Fra Figur 11 ser vi at vi har et problem når bildene skal sendes ut; B-rammene er kodet i forhold til bilder som ennå ikke har blitt vist. Dette problemet løses ved at bildene sendes ut i en annen rekkefølge enn den de blir vist i. Referanserammen blir sendt ut før de B-rammene som er kodet i forhold til den. Scenarioet er illustrert i Figur 12.



Figur 12: Dekodingsrekkefølge og avspillingsrekkefølge.

2.4.10 Begrensede parametre

Fordi det er mulig å kode en videosekvens på så mange måter er det definert begrensede parametre («Constrained Parameters») i MPEG-1-standarden [ISO 11172]. MPEG-1 kan benyttes for å kode videosekvenser med en bildestørrelse på opptil 4095 linjer x 4095 pels[3]. Noen av disse parametrene er vist i Tabell 4.

Horisontal bildestørrelse	Mindre eller lik 768 pels[2]
Vertikal bildestørrelse	Mindre eller lik 576 linjer
Bildeområde	Mindre eller lik 396 makroblokker
Pel-hastighet	Mindre eller lik 396x25 makroblokker/s
Bildehastighet	Mindre eller lik 30 Hz
Bitrate	Mindre eller lik 1 856 000 bit/s

Tabell 4: Et utvalg av «Constrained Parameters» fra MPEG-1-standarden (hentet fra [ISO 11172]).

2.5 MPEG-1 Audio

I dette kapittelet beskrives audiodelen utfra dokumentasjonen i ISO 11172-3.

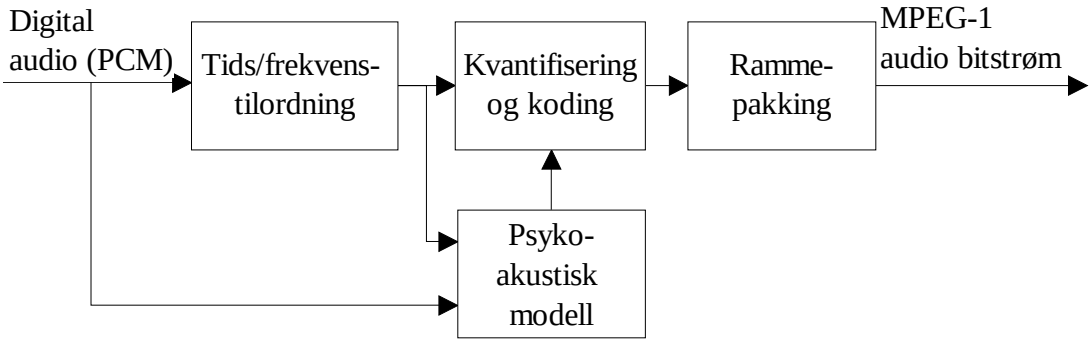
2.5.1 Oppbyggingen av en MPEG-1 audiostrøm

Toppnivået i en MPEG-1 audiostrøm består av rammer som starter med et synkord ("syncword"). Hver ramme er inndelt i "slots". Disse er igjen inndelt i subbånd. Hvert bånd er kodet etter en psykoakustisk modell (se 2.5.12) som ikke er dekket av standarden, men kan velges av enhver som implementerer en koder. Eneste kravet er

[2] Pel = Picture Element. Et bilde-element (det samme som piksel).

at den resulterende strømmen kan dekodes av en hvilken som helst dekoder såfremt denne følger standarden [ISO 11172].

2.5.1.1 Kodingsforløp



Figur 13: Kodingsforløp for MPEG-1 audio [BRASTO94].

Forløpet for kodingen er illustrert i Figur 13 [BRASTO94]. Den innkommende strømmen må allerede være PCM-kodet (rå sampling).

1. PCM-strømmen går så til en tids/frekvens-tilordning (filterbank) hvor frekvens-spekteret blir dekomponert i subbånd (bånd med hvert sitt frekvensområde).
2. Resultatet fra denne filterbanken eller resultatet fra en parallell transformasjon blir brukt til å beregne en tidsavhengig maskeringsterskel ved hjelp av regler fra psykoakustikk.
3. Subbåndene blir kvantifisert og kodet med det mål å holde støyen som blir introdusert ved kvantifisering, under en maskeringsterskel. Algoritmen for dette varierer sterk mellom de forskjellige valgte løsninger.
4. Til slutt pakkes bitstrømmen inn i rammer.

2.5.1.2 Dekodingsforløp

Dekodingsforløpet blir i grove trekk det motsatte av kodingsforløpet. Dekodingsforløpet er illustrert i Figur 14.

Figur 14: Dekodingsforløp for MPEG-1 audio [BRASTO94].

2.5.2 Lagdeling

MPEG-1 tillater tre forskjellige nivå av koding avhengig av kompleksitet og komprimeringsrate [ISO 11172]. Disse nivåene kalles enkelt og greit: Layer I, layer II og layer III. Her har vi valgt å kalle disse for lag 1, lag 2 og lag 3 henholdsvis. En dekoder som kan dekode lag N kan også dekode alle lag under N.

En nærmere beskrivelse av lagene følger i avsnitt 2.5.9, 2.5.10 og 2.5.11.

2.5.3 Audiorammer

For synkronisering av bilde og lyd deles audiostrømmen inn i rammer på 384 samplinger (lag 1) eller 1152 samplinger (lag 2 og lag 3). Hver ramme starter med fire bytes lang header (se 2.5.4) og feilsjekkdata (se 2.5.7). Antall samplinger i en ramme kan *ikke* varieres [ISO 11172].

2.5.4 Audioheaderen

Audioheaderen ligger i starten av hver ramme. Den er fire bytes lang og inneholder informasjon som er nødvendig for å dekode lyden som ligger kodet i rammen.

Synkordet er første del av headeren og dermed også første del av audiorammen. Videre ligger det informasjon om audiostrømmen følger ISO 11172-3, og i såfall hvilket lag (1, 2, 3 eller reservert). Dersom lagnummeret er forskjellig fra foregående ramme, kan det være nødvendig å restarte dekoderen.

Videre ligger det informasjon om hvilken bitrate som er benyttet (se 2.5.8), hvilken samplingsfrekvens som er benyttet (se 2.5.5), hvilken modus som er benyttet (se 2.5.6), om materiellet er copyrightet, om det er originalt eller en kopi og hvilken betoning som er benyttet (ingen, 50/15 μs, reservert eller CCITT J.17).

2.5.5 Samplingsfrekvens

MPEG-1 audio tillater samplingsfrekvenser i 44,1 kHz (CD), 48 kHz (DAT), 32 kHz eller reservert (iflg standarden [ISO 11172]; antagelig ment for senere bruk).

2.5.6 Lydmodus

Med lydmodus forstås forskjellige måter å kode en eller flere kanaler på.

Én kanal kan bare kodes på én måte: som én kanal. To kanaler kan kodes på tre måter: som stereo, intensitetstereo/MS stereo eller som to separate kanaler. MS stereo blir bare benyttet i lag 3. De øvrige blir benyttet i alle lag.

Stereokoding er bare en måte å fortelle dekoderen at kanalene skal avspilles i lag. Intensitetstereo er det samme som “Joint Stereo”. Dvs. at kanalene blir kodet som to kanaler, men likheten mellom dem, som kan være ganske stor for et stereospor, blir utnyttet slik at lydkvaliteten kan økes. Dette gjelder især ved lav bitrate.

Koding som to separate kanaler kan forståes som koding av to monokanaler, dvs. at kanalene ikke hører sammen. Det kan f.eks. dreie seg om en film som er dubbet, og originallyden går på én kanal mens dubbingen går på en annen.

Ikke alle kombinasjoner av lydmodus og bitrate er tillatt.

2.5.7 Feilsjekk

Feilsjekken som benyttes i MPEG-1 er CRC-16 [ISO 11172].

2.5.8 Bitrate

Lydstrømmen kan enten ha fri bitrate, dvs. variabel, eller den kan være fast. For de faste valgene kan det velges fra 32 til 448 kbit/s. Hvilke som kan velges er avhengig av lagnivå. De forskjellige tillatte valgene er opplistet i Tabell 5.

Bitrate (kbit/s)		
Lag 1	**Lag 2**	**Lag 3**
fri	fri	fri
32	32	32
64	48	40
96	56	48
128	64	56
160	80	64
192	96	80
224	112	96
256	128	112
288	160	128
320	192	160
352	224	192
384	256	224
416	320	256
448	384	320

Tabell 5: Forskjellige bitrater i MPEG-1 audio (hentet fra [ISO 11172]).

Bit rate (kbit/s)	Tillatt lydmodus
fri	alle
32	mono
48	mono
56	mono
64	alle
80	mono
96	alle
112	alle
128	alle
160	alle
192	alle
224	stereo, intensitetstereo, to kanaler
256	stereo, intensitetstereo, to kanaler
320	stereo, intensitetstereo, to kanaler
384	stereo, intensitetstereo, to kanaler

Tabell 6: Tillatte kombinasjoner av bitrate og lydmodus i MPEG-1 audio lag 2 (hentet fra [ISO 11172]).

Som tidligere nevnt kan ikke alle kombinasjoner av bitrate og lydmodus kombineres. Dette gjelder bare lag 2. De tillatte valgene er opplistet i Tabell 6.

2.5.9 Audiokoding generelt

Alle lagene konverterer først PCM rådata til frekvensdomenet. Dette gjøres med en filterbank bestående av 32 subbånd [BRASTO94]. Hvert av disse subbåndene representerer hvert sitt frekvensområde. Denne inndelingen gjør det mulig å kode hvert subbånd for seg.

Lag 1 er en forenklet utgave av MUSICAM-kodingen [BRASTO94], mest egnet for forbrukerapplikasjoner som magnetiske bånd og optiske disker. Dvs. lav båndbredde er ikke påkrevd.

Lag 2 er nesten identisk med MUSICAM-kodingen, og benyttes til både forbruker-applikasjoner og profesjonelle formål som TV, telekommunikasjon og multimedia.

Lag 3 er en kombinasjon av de mest effektive modulene fra MUSICAM- og ASPEC-kodingen. Lag 3 benyttes først og fremst til telekommunikasjon over linjer med lav båndbredde slik som ISDN.

I lag 1 og 2 lages 32 subbånd som deretter kvantifiseres og kodes under kontroll av en psykoakustisk modell.

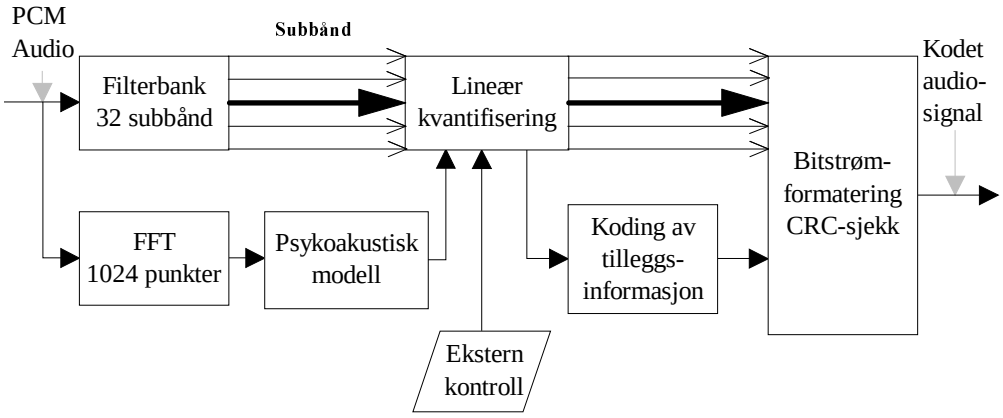
I lag 3 lages det også 32 subbånd, men hvert av disse båndene deles igjen opp i 18 frekvenslinjer. Dette fører til høyere frekvensoppløsning enn de øvrige lagene.

Felles (Joint) stereokoding kan legges til alle lagene. Se 2.5.6 for en forklaring av de forskjellige lydmodi.

Lag 3 er ikke en del av denne oppgaven. Årsaken til dette er at lag 3 er ganske komplisert fordi kodingen ikke bare skjer internt i hver ramme, men likheter mellom rammene utnyttes. Dette laget er ikke mye brukt bortsett fra i profesjonelle applikasjoner.

2.5.10 Audiodata og koding lag 1 og lag 2

Blokkdiagram av kodingen i lag 1 og 2 er illustrert i Figur 15. Kodeteknikken for disse lagene er basert på subbåndoppsplitting av PCM rådata ved hjelp av en filterbank, til 32 subbånd. Deretter følger en dynamisk bittilordning avledet fra en psykoakustisk modell og til slutt bitstrømformatering [BRASTO94].



Figur 15: Blokkdiagram for MPEG-1 audio koder lag 1 og 2 (hentet fra [BRASTO94]).

2.5.10.1 Filterbank

En filterbank er et slags båndpassfilter, dvs. at bare frekvenskomponenter mellom gitte verdier slipper gjennom. Inndelingen av disse verdiene varierer mellom de forskjellige typene filterbanker. Bakgrunnen til inndelingen av frekvensspekteret i subbånd ved hjelp av en filterbank er økt tilpassing av kodingen. F.eks. dersom det er ønsket med høyere oppløsning i midtfrekvensområdet, men lav oppløsning i høye og lave frekvenser (typisk egnet for menneskets stemme)

Det finnes mange forskjellige filterbanker, men de mest brukte i disse lagene er Polyphase og QMF-tree [BRASTO94].

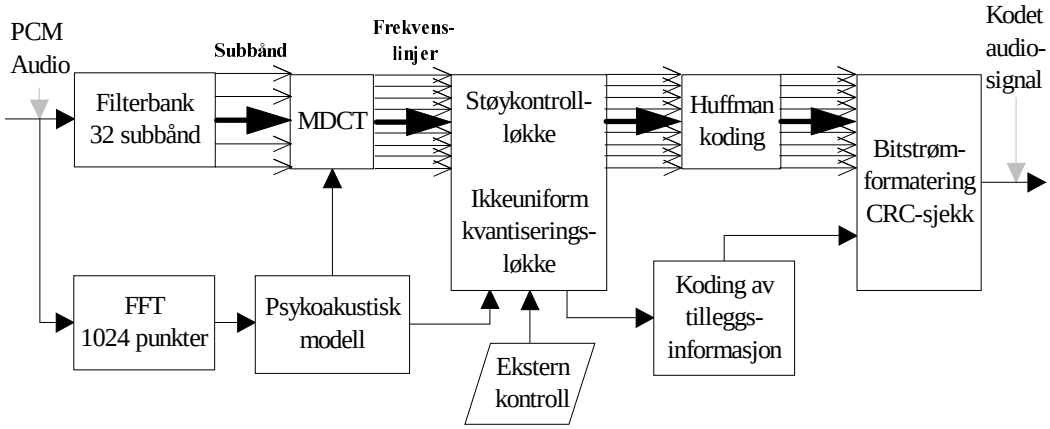
QMF-tree tillater subbånd med forskjellig avstand mellom båndene. Det vil si at det er mulig med forskjellig oppløsning ved forskjellige frekvenser. Kompleksiteten er høy. QMF bruker mellom 4 og 24 subbånd.

Polyphase tillater ikke forskjellig avstand mellom båndene, men må ha lik avstand mellom de. Polyphase bruker 32 subbånd.

2.5.10.2 Lineær kvantifisering

Resultatet fra filterbanken skaleres etter skaleringsfaktoren og kvantifiseres. I lag 1 kvantifiseres hvert subbånd for seg. I lag 2 kan det velges forskjellig kvantifiseringsnivå (kvantifiseringsoppløsning) for de enkelte subbånd. Dette kan gi en gevinst på opptil 37,5% mindre størrelse på den ferdigkodede strømmen [BRASTO94].

2.5.11 Audiodata og koding lag 3



Figur 16: Blokkdiagram for MPEG-1 audio koder lag 3 (hentet fra [BRASTO94]).

Et blokkdiagram av kodingen i lag 3 er vist i Figur 16. Filterbanken som blir brukt i lag 3 er en hybrid Polyphase/MDCT filterbank. Ikkeuniform kvantifisering og Huffman-koding blir benyttet for øke kodingseffektiviteten.

2.5.11.1 Filterbank

Polyphase filterbank er den samme som i lag 2. De 32 subbåndene fra denne filterbanken blir matet inn i en 18-kanals MDCT (Modified Discrete Cosine Transform) filterbank. Maksimum antall utgående kanaler er 32*18=576.

MDCT filterbanken kan settes til forskjellig frekvensoppløsning på de forskjellige kanaler. På denne måten kan frekvensresponsen skaleres etter ørets ømfintlighet. Med 576 kanaler er skalerbarheten i de forskjellige frekvensområdene svært fleksibel.

2.5.12 Psykoakustisk koding

Psykoakustisk koding benyttes for å maskere bort informasjon øret vårt ikke er i stand til å høre (teoretisk). Små styrkevariasjoner ved allerede høy lyd kan ikke høres. Lydkomponenter med lavt nivå er heller ikke mulig å høre dersom det er en annen lydkomponent som totalt dominerer lydbildet.

Definering av disse grensene og eventuelle andre metoder gjøres av koderen, og er således ikke en del av standarden. Standarden gir likevel to eksempler på hvordan dette kan løses. Disse kan i teorien benyttes i alle lag, men i praksis benyttes modell 1 for lag 1 og lag 2, og modell 2 for lag 3 [BRASTO94].

2.5.12.1 Psykoakustisk modell 1

Høy frekvensoppløsning, dvs. små subbånd i lavfrekvensområdet, og lav frekvensoppløsning, dvs. større subbånd i høyfrekvensområdet.

Denne modellen tar utgangspunkt i at vi har større følsomhet for små frekvensvariasjoner i det lavfrekvente området enn i det høyfrekvente, og bruker derfor mer av båndbredden på koding av lavfrekvente lyder. Dette skjer naturligvis på bekostning av de høyfrekvente lydene som etter denne modellen ikke er så viktige.

Denne modellen benytter også FFT for å få verdiene over til frekvensdomenet. Deretter beregnes en maskeringsterskel som verdiene kjøres gjennom, slik at små variasjoner ikke plukkes opp.

2.5.12.2 Psykoakustisk modell 2

Denne metoden deler opp lyden i vinduer på 1024 samplinger. Et nytt vindu begynner hver 576. sampling. På dette vinduet benyttes FFT for å få verdiene over til frekvensdomenet.

Denne metoden baserer seg på estimering, dvs. at forutgående verdier benyttes til å beregne antatt nye verdier. Deretter benyttes verdier mellom 0 og 1 for hvor godt den beregnede verdien traff. 0 vil si treff, og 1 vil si at den beregnede verdien ligger lengst fra den virkelige.

Til slutt beregnes en maskeringsterskel som verdiene kjøres gjennom, slik at små variasjoner ikke plukkes opp.

2.6 MPEG-2

I de siste årene har kravene til bilde- og audiokvalitet gått opp. I den forbindelse startet utformingen av en ny og oppdatert standard for «bevegelige bilder og tilhørende lyd». For å markere at dette er en videreføring av MPEG (nå kjent som MPEG-1) kalte de den nye versjonen MPEG-2. MPEG-2 er beregnet for kringkasting, mens MPEG-1 er beregnet for CD-ROM.

MPEG-2 tillater mye større bitrate og rammestørrelse [ISO 13818]. Den har implementert interlacing[3], en funksjon som benyttes i kringkasting av video.

MPEG-2 er godt tilrettelagt for overføring av kinofilm til video. Det finnes spesifikke koder som forteller dekoderen at bestemte rammer skal gjentas (vises en gang til). Siden film er på 24 bilder/sekund kan dette spare mye plass i forhold til amerikansk video (NTSC), som er på 30 bilder/sekund (60 halvrammer ved interlacing) og europeisk (PAL/SECAM), som er på 25 bilder/sekund (50 halvrammer ved interlacing).

Bildegrupper eksisterer ikke i MPEG-2 [ISO 13818], men kan benyttes dersom dekoderen trenger informasjon om at enkelte rammer ikke skal vises. D-rammer eksisterer heller ikke i MPEG-2-standarden.

MPEG-2 er ikke en del av denne oppgaven.

[3] Ved interlacing sendes først scanlinje 1, 3, 5, 7 .... og deretter 2, 4, 6, 8, ...

3. PROBLEMBESKRIVELSE

Det er ytret ønske fra IDI om å få utviklet en editor for MPEG-1 System. Denne editoren skal være i stand til å sette sammen flere uavhengige systemstrømmer på vilkårlige plasser i hver av strømmene.

Dette kapittelet er oppdelt i en seksjon for system, en for video og en for audio. Dette er gjort fordi problemet egentlig ikke kan løses på annen måte enn å atskille strømmene, redigere og sette de sammen til systemstrøm igjen.

3.1 System

System er egentlig ikke noe problem. Dette nivået kommer man borti bare når strømmen skal separeres i forskjellige understrømmer (video og audio) og når disse skal settes sammen igjen. Dette er forholdsvis enkelt siden det bare er å gjennomløpe systemstrømmen og skille ut de forskjellige pakkene etter identifikator (strømnummer). Se forøvrig 2.3 for beskrivelse av MPEG-1 system.

3.2 Video

Dette underkapittelet er basert på eget arbeid utført våren 1996 [DYBVIK96] og er tatt med for kompletthet.

B- og P-rammer baserer seg på innholdet i andre rammer ved hjelp av såkalte bevegelsesvektorer. Disse koder bare hvor mye bildet har endret seg, og i hvilken retning en eventuell bevegelse har foregått.

Den enkleste løsningen vil være å bare kutte rett etter den siste rammen som skal være med fra den ene strømmen, og sette sammen med første rammen fra den andre.

Dersom det kommer en ramme med en bevegelsesvektor som baserer seg på en ramme som er erstattet med en annen, blir resultatet noe annet enn ventet. Et eksempel på en slik editering er illustrert i Figur 17, Figur 18 og Figur 19. Dette består av to klipp tatt fra Nordnytts sendinger på NRK 1, 15. april 1996. Noen bilder fra disse klippene er vist i Figur 17 og Figur 18. Ramme 0 er siste ramme før kuttet for Figur 17, og første ramme etter klippet for Figur 18.

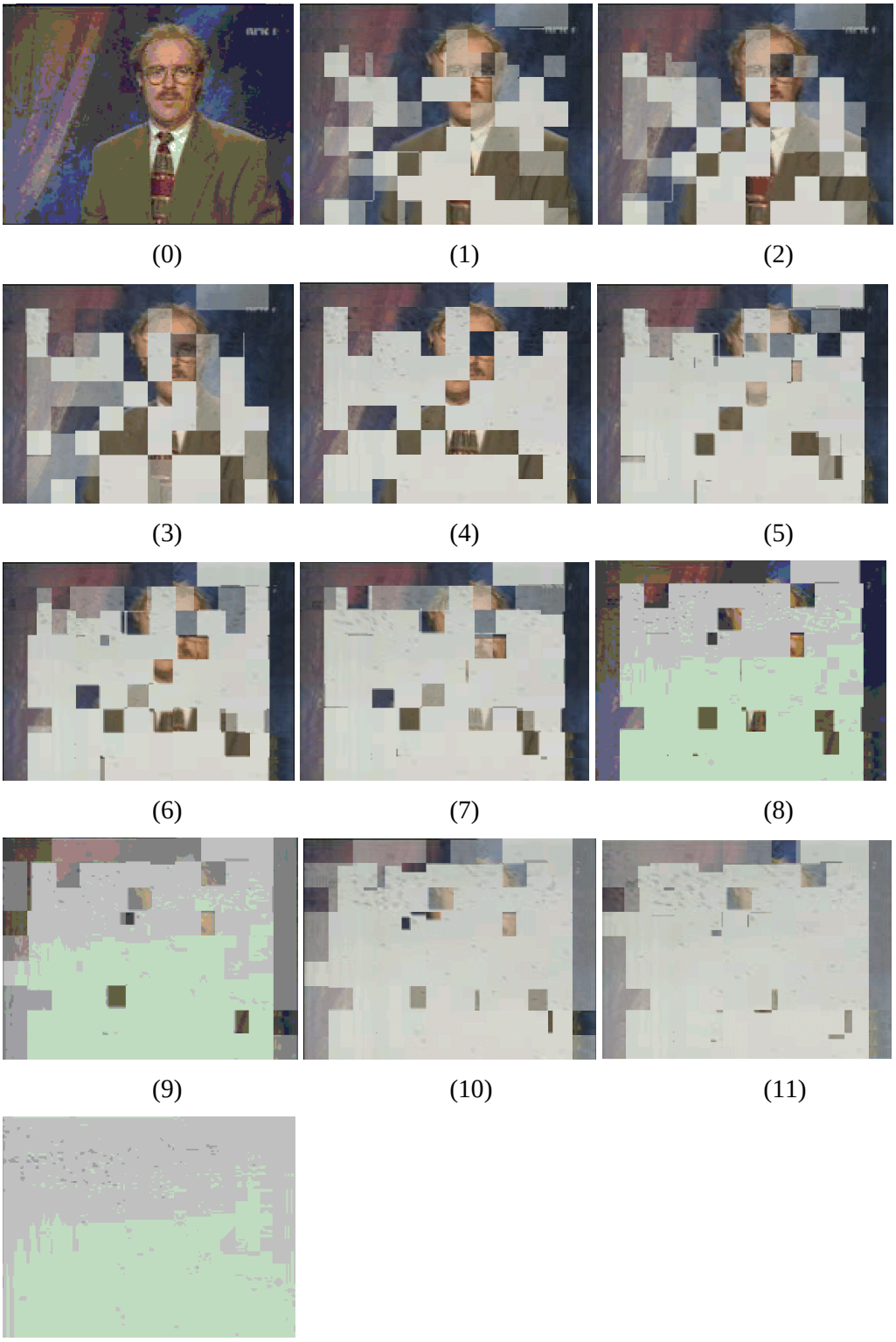
I Figur 20 finner vi resultatet av denne editeringen. Ramme 0 hører til klipp 1, mens ramme 1 egentlig er første rammen i klipp 2, som er klippet inn. Denne rammen er en P-ramme med en del nye makroblokker. Ramme 1 skulle egentlig ha sett ut som ramme 0 i Figur 18 (en flokk med måker som flyger over himmelen).

(-1) (0) (+1)

Figur 17: Klipp 1.

(-1) (0) (+1)

Figur 18: Klipp 2.



Figur 19: Rammesekvens ved editering.

3.3 Audio

Audio er inndelt i rammer som video, men er ikke avhengig av andre rammers informasjon bortsett fra i lag 3, som bruker vinduer som går på tvers av rammene. Siden lag 3 ikke er en del av oppgaven, er ikke dette problemet relevant.

Rammestørrelsen i lag 1 er på 384 samplinger og i lag 2 på 1152 samplinger. Denne størrelsen er den samme uansett samplingshastighet. Dette innebærer at en ramme inneholder lyd av forskjellig lengde avhengig av samplingshastighet og lag. Relasjonen kan sees i Tabell 7. Ved 25 videorammer/sekund (PAL og SECAM) varer én videoramme 1/25 = 40 ms. Som vi ser av tabellen går dette opp kun for 48 kHz i lag 1: 40 ms/8 ms = 5 rammer. Dvs. fem audiorammer for hver videoramme. Ved 24 videorammer/sekund (film på kino) varer én ramme 1/24 = 41,67 ms. Ved denne hastigheten går ingen av kombinasjonene lag/frekvenshastighet opp. Heller ikke ved 30 videorammer/sekund (NTSC) går regnskapet opp. Én videoramme varer her i 1/30 = 33,3 ms.

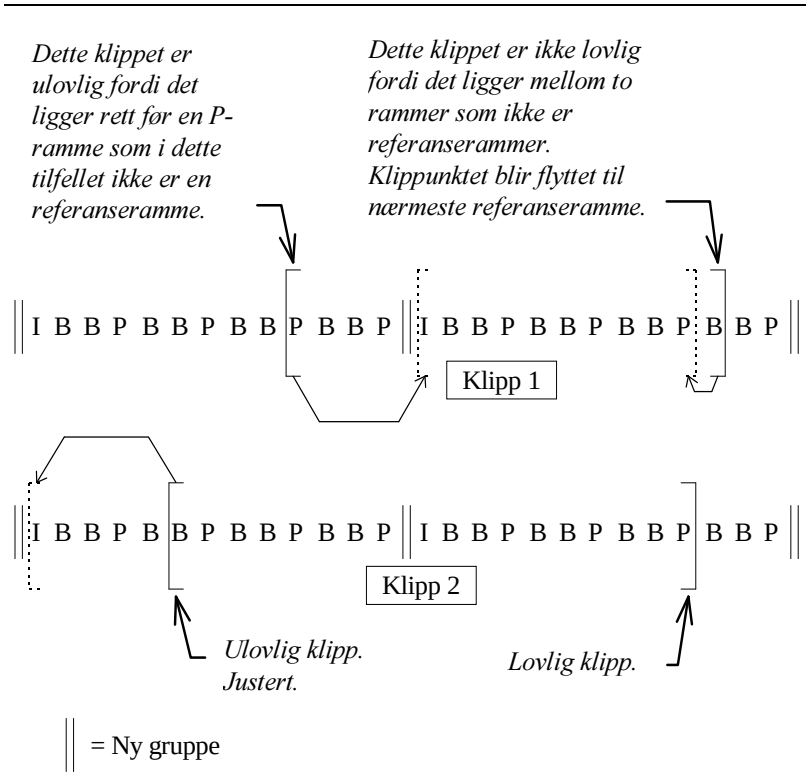
Samplingshastighet (kHz)	Lag	Tid (ms)
32	1	12,0
32	2	36,0
44,1	1	8,71
44,1	2	26,1
48	1	8,00
48	2	24,0

Tabell 7: Rammelengde ved forskjellige samplingshastigheter og lag.

3.4 Dagens «State of the Art»-løsning

Det finnes noen editorer for MPEG-1 video- og systemstrømmer tilgjengelig i dag. Felles for de fleste er at de ikke tillater editering på vilkårlig plass i bildestrømmen. En felles begrensning er at klipp må foretaes ved ny referanseramme/gruppestart (GOP) i videostrømmen. Denne løsningen medfører liten fleksibilitet ved redigering, fordi bildegrupper kan være svært store (ubegrenset i følge standarden [ISO 11172]).

«mpegUtil» er et slikt program for video. Dette programmet tillater bare redigering ved hver referanseramme. Dersom spesifisert klippunkt ligger på ulovlig sted, flyttes bare klippunktet til nærmeste lovlige sted. Dette medfører en svært upresis redigering. Et eksempel på denne metoden er illustrert i Figur 20.



Figur 20: Redigering ved hjelp av referanserammer i "mpegUtil".

Jeg har ikke vært i stand til å finne noen programmer som implementerer editering av MPEG-1 audio. Metoden som benyttes er antakelig at hele strømmen dekodes først og redigeres med et PCM redigeringsprogram for så å bli satt sammen igjen. Dette kan medføre et kvalitetstap. Størrelsen på dette tapet bestemmes av hvor stor komprimering den nye strømmen får. Den generelle regelen er at desto større kompresjon, desto større kvalitetstap.

3.5 Problemer med den ønskede løsningen

Den ønskede løsningen introduserer mange problemer som må løses. Den ønskede løsningen skal være i stand til å sette sammen videostrømmer også utenfor referanserammer. Dette er ikke helt uproblematisk siden rammene i en gruppe er kodet i forhold til hverandre.

For audio skal det være mulig å kutte på samme plass som for video. Audiorammene er ikke like store som videorammer, dvs. at det kan bli problemer med synkronisering av audio og video dersom audiostrømmen klippes ved hver (audio) rammeslutt/start.

MPEG-1 ble i utgangspunktet ikke utviklet med tanke på muligheter for editering, men for å muliggjøre lagring av større mengder video og/eller lyd på begrenset plass, samt å redusere båndbredden på denne ved overføring. Selv om MPEG-1 ikke ble utviklet for å muliggjøre editering er det likevel mulig å editere dersom vi er villige til å inngå visse kompromisser. Blant disse er redusert lyd- og bildekvalitet.

4. ALTERNATIVE LØSNINGSMODELLER

Det er mange forskjellige måter å løse dette problemet på. Alle har forskjellig vanskelighetsgrad. Et fellestrekk er at det som er enkelt å implementere medfører at brukeren får en dårlig løsning og/eller at brukeren må bruke mye datakraft for å få løst oppgaven.

I dette kapittelet blir flere løsningesalternativer gjennomgått og vurdert. System, video og audio blir også her behandlet hver for seg.

4.1 System

Systemlaget kan ikke redigeres direkte p.g.a. den komplekse multipleksingen av forskjellige strømmer. Derfor må strømmen deles opp i separate strømmer (demultiplekses) før redigeringen. Etter demultipleksingen startes audioeditoren og videoeditoren på de demultipleksede strømmene.

Etter at de forskjellige strømmene er ferdig redigert, settes de sammen igjen (multiplekses). Disse operasjonene (multipleksing og demultipleksing) er tapsløse siden det bare er utpakking og innpakking av datapakker. Det burde heller ikke være spesiellt ressurskrevende å demultiplekse og multiplekse disse strømmene. Det eneste som gjøres er å flytte data fra en fil til en annen med diverse tilleggsinformasjon lagt inn mellom datapakkene.

4.2 Video

Dette underkapittelet er basert på eget arbeid utført våren 1996 [DYBVIK96] og er tatt med for kompletthet.

4.2.1 Redigering med hele grupper

Et av problemene man unngår ved å sette som kriterium at editering kun kan foretaes ved hver gruppestart, er dekoding og koding av rammer. Med dette alternativet er det bare å kopiere hele grupper fra kildefilene til resultatfila. Denne metoden vil heller ikke føre til noen kvalitetsreduksjon.

Ulempen med denne metoden er at det ikke er mulig å foreta en ramme-nøyaktig editering. Redigeringspunktet vil bli forskjøvet til nærmeste etterfølgende eller foregående gruppestart.

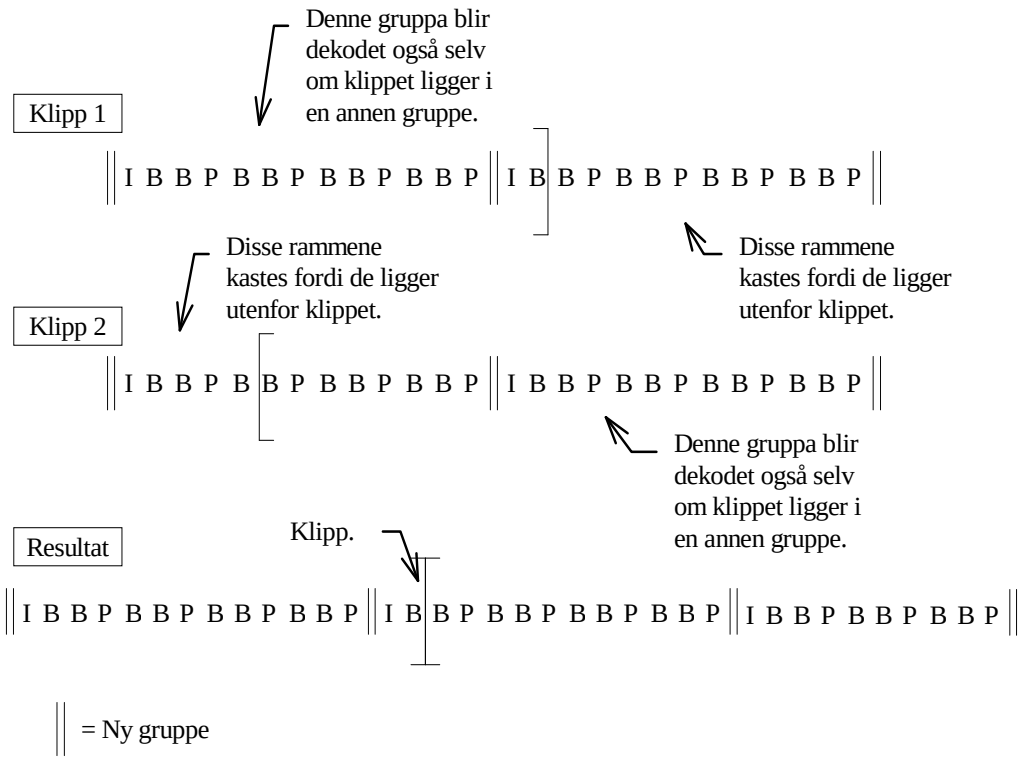
Ved editering inne i en gruppe ødelegges avhengigheten mellom rammene. Vi kan da sitte igjen med rammer som er kodet utfra referanserammer som er klippet bort.

4.2.2 Dekoding og koding av hele bildestrømmen

Et alternativ for å løse dette problemet er å kode hele bildestrømmen på nytt. Med dette alternativet dekodes først alle rammene og lagres på disk eller i arbeidsminnet (RAM). Deretter kodes alle rammene på nytt i den nye rekkefølgen.

Denne løsningen er definitivt den enkleste å implementere. Alle rammene kan dekodes ved hjelp av et eksternt program, og rammene kan kodes igjen ved hjelp av et eksternt program. Den eneste oppgaven som må løses, er å sortere rekkefølgen av rammene i den nye strømmen.

Denne dekodingen og kodingen av alle rammene vil redusere bildekvaliteten på *alle* bildene. Grunnen til at bildekvaliteten blir forringet ved en ny komprimering er at 0MPEG-koding ikke er tapsløs. Koding av rammer er en svært tidkrevende prosess, og bør såfremt det er mulig unngåes. Denne løsningen er illustrert i Figur 21. En naturlig optimalisering av denne metoden ville være å unngå dekoding/koding av rammene foran redigeringspunktet.



Figur 21: Dekoding og koding av alle rammer.

4.2.3 Dekoding og koding av bare de gjeldende gruppene

Et tredje alternativ er å dekode rammene i bare de gruppene som blir berørt av editeringen, sette sammen i følge den spesifiserte klipplisten, kode rammene i den nye rekkefølgen og til slutt sette sammen med delene som ligger utenfor de berørte gruppene.

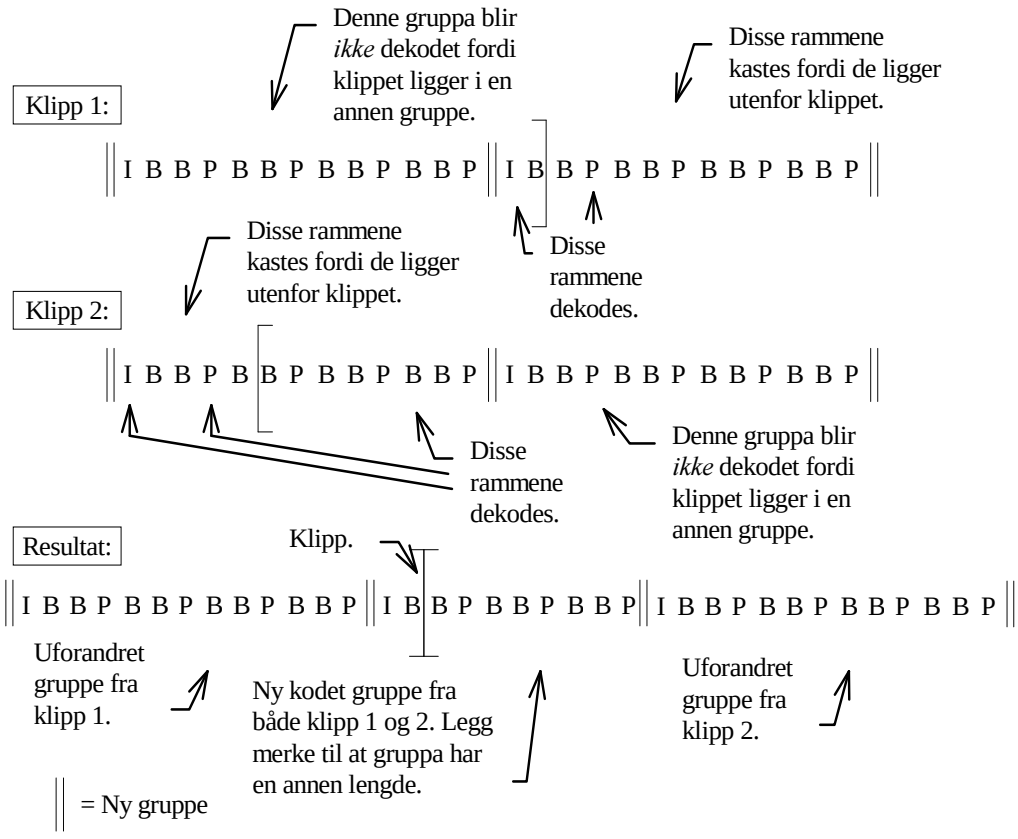
Det er to måter å kode de løse rammene på. De kan kodes som én gruppe slik at alle rammene fra begge strømmene ligger sammen. Dette kan føre til problemer akkurat i overgangen fordi den første rammen etter klippet er etter all sannsynlighet, ganske forskjellig fra siste ramme før klippet. En annen måte å gjøre det på er å kode rammene fra hver videostrøm hver for seg slik at vi får to mindre grupper. Dette vil føre til et lite hopp i bitrate fordi hyppigheten av I-rammer øker. Et tredje alternativ er å finne ut hvilke av de to metodene som skal brukes for hver gang. På den måten får vi hverken spesielt lange eller spesielt korte grupper.

Denne løsningen er ganske problematisk å implementere. For den fila som skal klippes ut forutsetter den at strømmen må gjennomløpes frem til den gruppestarten som ligger foran klippet. Deretter skrives alt frem til denne gruppestarten umodifisert til den nye fila. Resten av rammene frem til klippet må dekodes og ordnes. For den fila som skal klippes inn, må alle rammene frem til klippet gjennomløpes, og alle rammene etter den nærmeste gruppestarten *foran* klippet fram til neste gruppestart må dekodes og ordnes. Alt etter gruppestarten som ligger etter klippet, skrives til fil umodifisert. Ordningen av rammene i de gruppene som skal dekodes er ikke helt uproblematisk. Årsaken til dette er at rammene ikke ligger i visningsrekkefølge (se Figur 12) i filene.

Denne løsningen forringer ikke kvaliteten på andre rammer enn de som ligger i de gruppene som er berørt av klippet (det kan i enkelte tilfeller være ingen grupper dersom klippet i begge strømmene ligger ved gruppestart). Løsningen bruker litt tid på å kode de rammene som er dekodet, men i ekstreme tilfeller kan det likevel antas at det ikke er nødvendig å kode mer enn femti rammer (en gruppe overstiger sjelden ett sekund).

Dersom de to videostrømmene har forskjellige kvantifiseringsmatriser må den nye legges inn i strømmen på det punktet den blir gjeldende, dvs. når rammene bare blir flyttet over til den nye strømmen (rammer/grupper som *ikke* blir dekodet). Dette løses med å legge inn en ny sekvensheader (se 2.4.2).

Denne løsningen er illustrert i Figur 22.



Figur 22: Dekoding og koding av bare de gjeldende gruppene.

Som vi ser av figuren må referanserammene til de rammene som ligger innenfor klippet, også dekodes. Den nye gruppa som er satt inn som en «skjøt» mellom de to

strømmene, har en annen lengde enn de andre gruppene. Dette kommer av at de opprinnelige gruppene som ligger utenfor klippet er beholdt, og rammene i de berørte gruppene er kodet sammen som én ny gruppe. I dette tilfellet medførte det en kortere gruppe, men kan like gjerne medføre en lengre gruppe (inntil dobbel gruppelengde - 2). I slike tilfeller bør rammen kodes som to grupper.

4.2.4 Valgt løsning

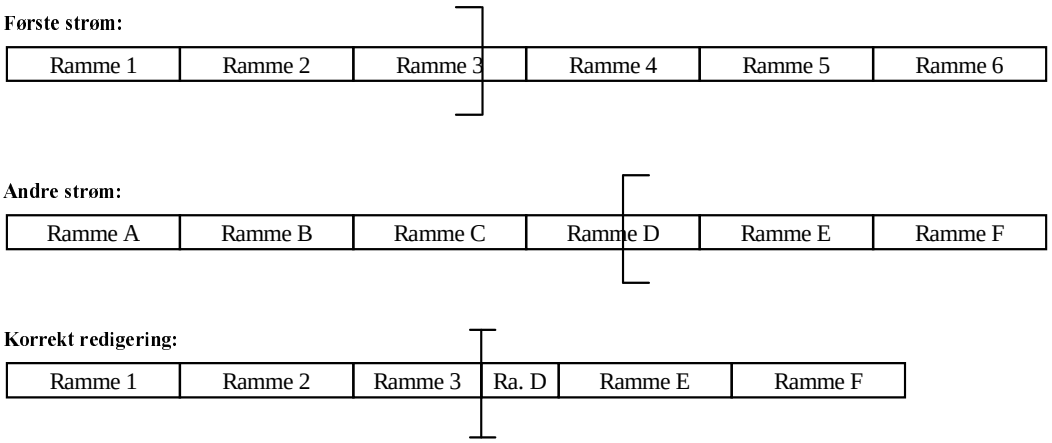
Alle alternativene er oppsummert i Tabell 8. Utfra disse, faller det mest naturlig å velge løsningen i avsnitt 4.2.3 selv om den er mest komplisert å implementere. Forskjellige nivåer av implementeringsgrad og gjenbruk av andre programmer kan velges. Det er utviklet både MPEG spillere (dekoder + grafisk visning) og kodere. Jeg har i dette prosjektet valgt å basere meg på koden fra University of California Berkeley (UCB). Årsaken til at jeg har valgt Berkeleys programvare er at den er mest portabel og definitivt den mest brukte. Den er også gratis.

Metode:	**Hele grupper**	**Koding/dekoding av hele strømmen**	**Dekoding av bare gjeldende grupper**
Kompleksitet:	Enkelt	Enkelt	Komplisert
Maskinressurser:	Lite	Mye	Middels
Fordeler:	Enkelt	Kan editere på vilkårlig punkt.	Kan editere på vilkårlig punkt.
Ulemper:	Må tilpasse editeringspunktet	Forringer bildekvaliteten	Komplisert å implementere

Tabell 8: Oppsummering av løsningsalternativene for video.

4.3 Audio

I Figur 23 er det tegnet opp en tenkt audiosekvens som skal redigeres. Den første strømmen skal klippes rett etter midten i ramme 3. Deretter skal den nye strømmen fortsette fra ca. midten av ramme D i den andre strømmen. Nederst i figuren er det tegnet opp hvordan en korrekt redigering skal være.



Figur 23: Tenkt audiosekvens for redigering.

I de følgende avsnittene illustreres det hvordan resultatet av de forskjellige løsningene blir sammenlignet med den korrekte redigeringen i Figur 23.

4.3.1 Redigering med hele rammer

En mulighet for å redigere audiostrømmen er å foreta klipp ved hver rammestart. Dette vil forenkle prosessen noe fordi i likhet med video (se 4.2.1) slipper man å dekode/kode rammer dersom klippet foretas ved hver rammestart. Denne løsningen er illustrert i Figur 24.

Denne løsningen innfører noen problemer. Rammestart i audiostrømmen tilsvarer svært sjelden rammestart i videostrømmen (se 3.3). Derfor vil en slik løsning skape problemer med synkroniseringen mellom audio og video. Hvor stort problem dette innebærer avhenger av hvor store tidsavvik vi kan akseptere mellom det vi ser og det vi hører.

Denne løsningen vil også enten føre til at deler av lyden mangler, eller for mye av lyden fra en strøm blir tatt med. Dette vil skje fordi kuttet oftest ligger inne i en ramme. Da må vi foreta et valg, om vi skal kutte før eller etter rammen kuttet ligger i.

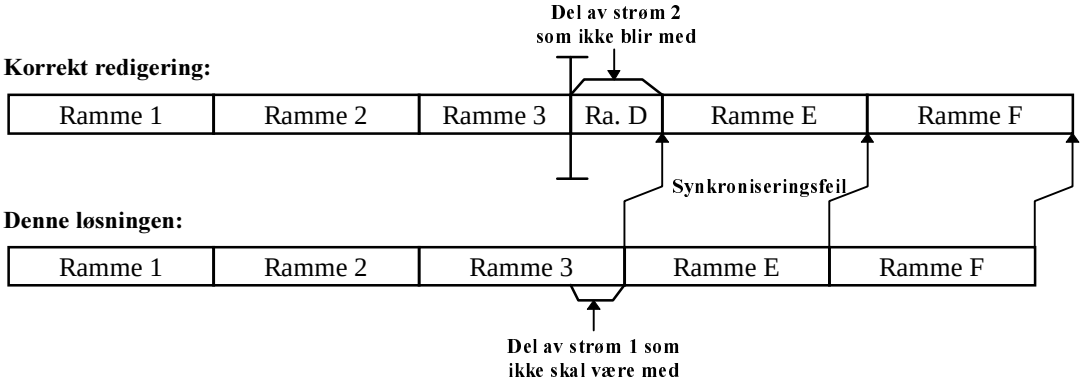
Med denne løsningen vil sannsynligvis lyden aldri komme synkront med bildet igjen (se 3.3 og Figur 24), men kommer forut for og/eller etter bildet (synkroniseringsfeil). Dersom en slik løsning implementeres rått kan audio i verste fall komme mange sekunder forskjellig fra video dersom det er en lang klippliste. Derfor må synkroniseringsfeilen fra siste klipp taes vare på slik at vi ved neste klipp kan avgjøre om det blir best resultat å klippe før eller etter rammen klippet ligger i. Med denne løsningen blir synkroniseringsfeilen maksimalt en halv rammelengde etter følgende formel:

$$t_{synkroniseringsfeil} \leq \frac{1}{2} t_{ramme}$$

Utregningene av denne formelen er listet opp i Tabell 9.

Samplingshastighet (kHz)	Lag	Synkroniseringsfeil (ms) ≤
32	1	6,0
32	2	18,0
44,1	1	4,35
44,1	2	13,0
48	1	4,00
48	2	12,0

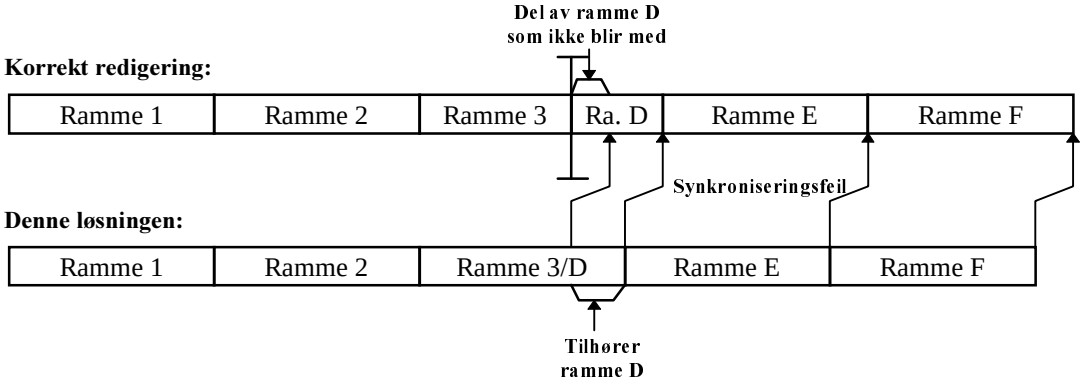
Tabell 9: Maksimale synkroniseringsfeil ved forskjellige samplingshastigheter og lag.



Figur 24: Resultat etter redigering med hele rammer.

4.3.2 Dekoding og koding av bare de gjeldende rammene.

Dekoding og koding av bare de gjeldende rammene (rammene som klippet ligger i) vil avhjelpe situasjonen noe, fordi vi ikke får med for mye av lyden fra den strømmen som blir klippet ut. Men problemet blir ikke løst fordi lyden fortsatt ikke kommer synkront inn med resten av videostrømmen. Som illustrert i Figur 25, mister vi også litt av den delen av ramme D som egentlig skulle være med, eventuelt tilsvarende del av ramme 3 eller en mindre del av begge rammene. Årsaken til at det blir slik er at rammestørrelsen består av et konstant antall samplinger (se 2.5.3). Rammer i audio må ikke forveksles med grupper i video. Videogruppene kan ha forskjellig størrelse, men størrelsen på audiorammene kan *ikke* varieres [ISO 11172].



Figur 25: Resultat etter dekoding og koding av gjeldende rammer.

Denne løsningen er bedre enn redigering med hele rammer fordi vi ikke får med uønsket lyd. Problemet er at synkroniseringsfeilen fortsatt er til stede og at vi får en viss degradering av lydkvaliteten i den rammen som er kodet på nytt.

Synkroniseringsfeilen kan løses på samme måte som forklart i 4.3.2.

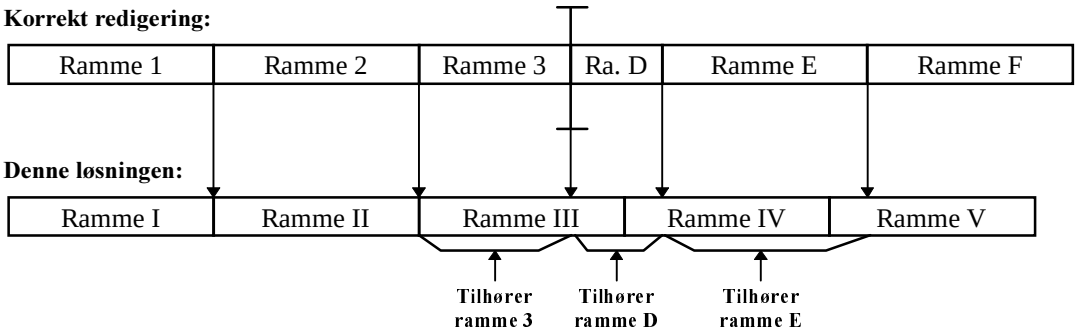
4.3.3 Dekoding og koding av hele audiostrømmen

Dekoding og koding av hele strømmen er den eneste løsningen som beholder synkroniseringen av lyd og bilde gjennom hele den nye strømmen. Denne løsningen har noen ganske store problemer som ikke finnes i de andre løsningsalternativene.

Dekoding og koding av hele audiostrømmen medfører et økt tidsforbruk sammenlignet med dekoding og koding av bare gjeldende rammer og redigering av hele rammer.

Denne løsningen medfører også en degradering av lydkvaliteten for hele strømmen. Siden MPEG-1 audio koding ikke er tapsløs vil det innføres tap hver gang signalet blir kodet. Det er derfor en fordel å begrense dette så mye som mulig for å opprettholde en akseptabel kvalitet på sluttproduktet.

Denne løsningen er illustrert i Figur 26. Det må gjøres oppmerksom på at rammene i den nye strømmen ikke er de samme som i de opprinnelige strømmene. Det er derfor tegnet inn piler for å vise korresponderende tidspunkt i audiostrømmene.



Figur 26: Resultat etter dekoding og koding av hele audiostrømmen.

4.3.4 Andre momenter

Et rått klipp i en audiostrøm kan medføre et godt hørbart "klikk" p.g.a. en plutselig endring i amplituden. Dette kan forhindres dersom det legges inn en effekt inn mot kuttet fra begge sidene. Effekten som må legges inn er "fading"; d.v.s. at volumet skrus ned inn mot kuttet, for deretter å øke igjen etter kuttet.

"Fading" foretaes med å finne en funksjon som er monotont avtagende fra 1 til 0 og multiplisere denne med amplituden til det opprinnelige signalet. Nullpunktet må ligge i klippunktet. Toppunktet må ligge et forutbestemt antall sekunder fra kuttet i hver retning fra kuttet.

Med denne ekstrafunksjonen må rammene rundt kuttet dekodes uansett hvilken løsning som velges. Årsaken til dette er fordi det er ganske komplisert å manipulere en MPEG-1 audiostrøm direkte, selv om dette er mulig og ikke vil introdusere noe tap av kvalitet noe dekoding og koding vil [BROWEN95].

4.3.4.1 Synkronisering av audio og video

For å sette den eventuelle synkroniseringsfeilen i perspektiv, kan en sammenligning med kino gjøres. I en kinosal av middels størrelse kan forskjellen mellom den som sitter fremst og den som sitter bakerst være mellom 10 og 20 meter. Med en lydfart på 340 m/s vil det gi en synkroniseringsforskjell på mellom *20 m/340 m/s = **58,9 ms*** og *10 m/340 m/s = **29,4 ms***. Dersom vi antar at en kino er satt opp slik at lyden kommer litt *før* bildet for den som sitter fremst og litt *etter* bildet for den som sitter bakerst, sitter vi fortsatt igjen med inntil 29,4 ms i forskjell. Ettersom disse tallene er flere ganger så store for store kinosaler (f.eks. Oslo Spectrum), må vi anta at en synkroniseringsfeil på inntil 36 ms ikke kan være merkbart.

Dette synspunktet støttes også av [BULLIE91] hvor det står at leppesynkronisering av audio og video må være mellom 10 og 100 ms (forsinkelse). En annen artikkel [LEYTEU91] mener synkroniseringen må ligge mellom -20 og +40 ms. Det vil si at lyden kan være inntil 40 ms *etter* video, men ikke mer 20 ms *før*. Denne siste artikkelens syn kan også tilfredsstilles i alle tilfeller dersom det legges inn forskjellige grenser avhengig av hvilken side av videostrømmen lyden kommer. Med maksimum 36 ms får vi, dersom vi fordeler med samme forhold som i [LEYTEU91], -12 ms og +24 ms.

Ved IBM European Network Centre ble det presentert data som indikerte at synkroniseringsfeilen ved leppesynkronisering av audio og video kan være opp til +/- 80 ms [STEMEY92].

4.3.5 Valgt løsning

Redigering med hele rammer fører til at audio og video blir usynkront. Det dreier seg ikke om lange perioder. Inntil en halv rammelengde, i verste fall inntil 18,0 ms. Se forøvrig Tabell 9: Maksimale synkroniseringsfeil ved forskjellige samplingshastigheter og lag.

Dekoding og koding av bare de gjeldende rammene utgjør ikke så stor forskjell fra redigering med hele rammer. Synkroniseringsfeilen vil være den samme p.g.a. den faste rammelengden. Denne løsningen vil bare medføre økt kompleksitet fordi noen rammer må dekodes og kodes. I de kodete rammene må det også telles samplinger fram til klippunktet.

Dekoding og koding av hele audiostrømmen ansees som utilfredsstillende av flere årsaker. Å dekode en audiostrøm tar ikke så lang tid, men koding tar *svært* lang tid. En audiostrøm vil i likhet med en videostrøm, bli degradert av dekoding og koding.

Med bakgrunn i ønsket om mulighet for "fading" må enten koding/dekoding av hele strømmen velges, eller noen rammer på hver side av kuttet kan dekodes slik at "fadingen" legges inn før disse kodes igjen. Problemet med den første er en (betydelig) reduksjon av lydkvaliteten og stort behov for maskinressurser/tid. Den andre metoden fører til synkroniseringsfeil.

Med bakgrunn i artiklene om tolerante grenser for synkroniseringsfeil og en oppsummering av løsningsalternativenes positive og negative elementer (se Tabell 10), velges løsningen basert på redigering av hele rammer med dekoding av tilstøtende rammer for å tillate fading. Det er mulig å komme godt innenfor synkroniseringskravene til den strengeste av disse artiklene (-20 ms +40 ms) med den valgte løsningen. I verste fall blir synkroniseringsfeilen -12 ms +24 ms (-1/3 +2/3 av rammelengden). Derfor velges denne løsning for implementasjon i oppgaven.

Den valgte løsningen blir derfor en blanding av "Hele rammer" og "Dekoding av bare gjeldende rammer" i Tabell 10.

Metode:	**Hele rammer**	**Koding/dekoding av hele strømmen**	**Dekoding av bare gjeldende rammer**
Kompleksitet:	Middels	Enkelt	Komplisert
Maskinressurser:	Lite	Mye	Middels
Fordeler:	Slipper dekoding og koding. Tillater *ikke* "fading".	Kan editere på vilkårlig punkt. Tillater "fading".	Kan editere på vilkårlig punkt. Tillater "fading".
Ulemper:	Må tilpasse editeringspunktet	Forringer lydkvaliteten	Komplisert å implementere

Tabell 10: Oppsummering av løsningsalternativene for audio.

5. OVERORDNET DESIGN OG KONSTRUKSJON

5.1 Innledning

I løsningen til denne oppgaven er det implementert et redigeringssystem for MPEG-1 systemstrømmer. Systemet er for det meste et overordnet system som benytter seg av allerede eksisterende programvares funksjoner. Ved manglende funksjonalitet hos denne programvaren er den endret. For å styre og kontrollere denne editoren er det utviklet egne programmer. Det er også utviklet egen programvare der eksisterende programvare ikke kunne utføre ønsket funksjon.

I systemdelen er det benyttet et program som følger med Berkeleys MPEG-1 distribusjon som heter "mpeg_demux". Dette programmet splitter opp en systemstrøm (systemfil) i separate strømmer (filer) for video og for audio.

I videodelen benyttes Berkeleys "mpeg_play" med biblioteksgrensenittet "mpeg_lib" til dekoding av enkeltrammer, og Berkeleys "mpeg_encode" blir brukt til koding av rammer. Koden som skal til for å sette sammen klippene har jeg implementert selv.

"mpeg_lib"s funksjonalitet er ikke helt som ønsket. Det er dessverre ikke mulig å gå til en vilkårlig ramme for å dekode denne, men hele videostrømmen må gjennomløpes til de rammene som skal dekodes er nådd. Dette medfører økt tidsforbruk, men siden dekoding av rammer innebærer mye programmering er denne løsningen likevel å foretrekke. Dette biblioteket måtte dessuten utvides noe i funksjonalitet for å tilfredsstille prosjektets behov.

"mpeg_encode" blir startet av systemet som et eksternt program. Systemet genererer nødvendige filer som koderen trenger, herunder de rammene som skal kodes i RGB-format og en parameterfil. Parameterfila inneholder alle opplysninger om hvilke metoder koderen skal benytte, hvor den skal hente rammefilene, rammemønsteret osv.

I audiodelen blir programmet "musicout" brukt til dekoding, og "musicin" til koding av MPEG-1 audio layer I og II. Alle andre elementer i denne delen er implementert selv.

5.2 Overordnet beskrivelse

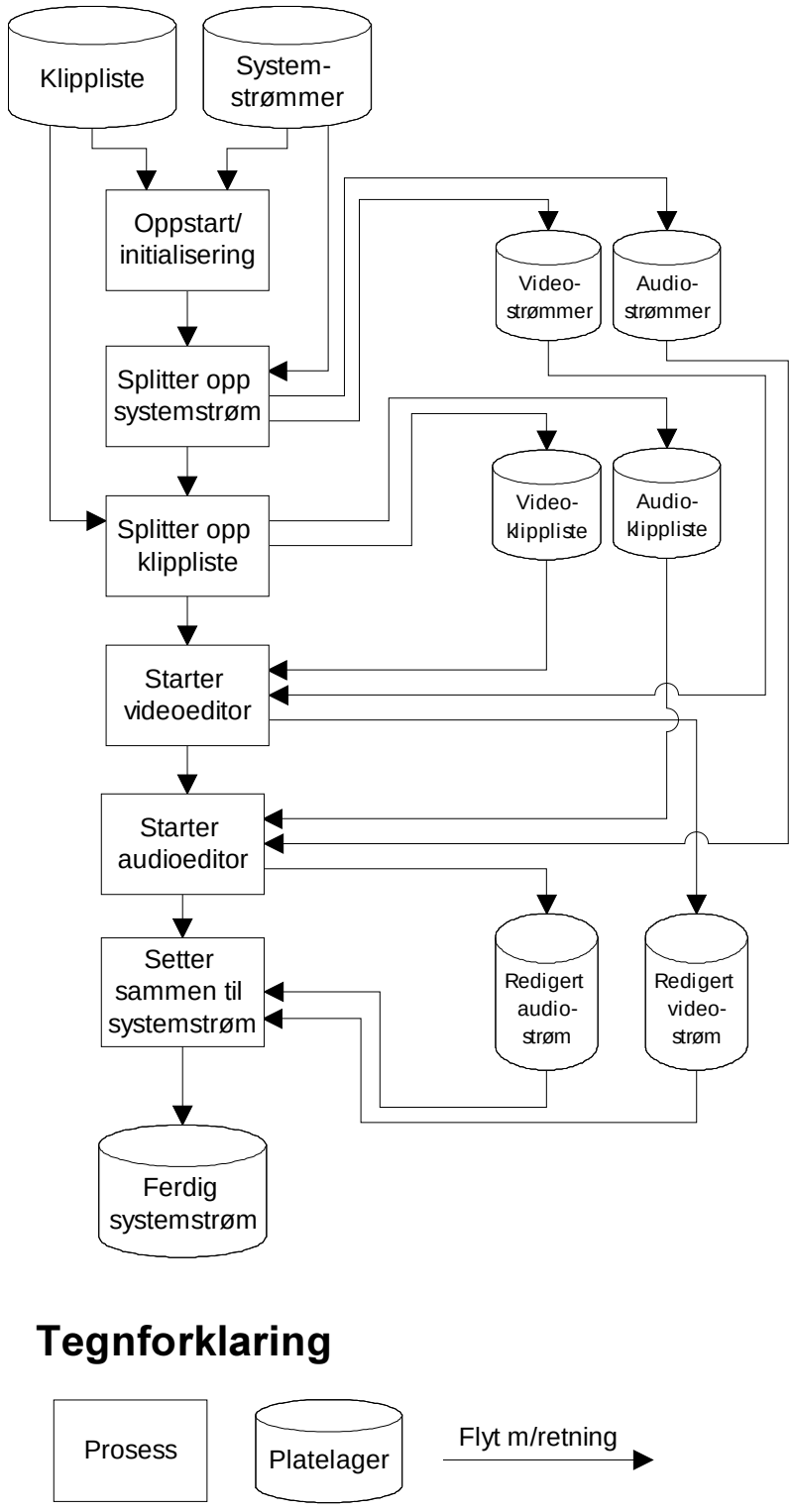
5.2.1 System

Systemdelen er en overbygning som ligger over en redigeringsdel for audio og en for video. Se Figur 27 for et prosess- og flytdiagram.

Denne delen starter med å splitte opp systemstrømmen i en audio- og en videodel. Disse oppsplittede strømmene blir deretter matet inn i hver sin redigeringsdel; en for video og en for audio. Disse redigeringsdelene blir startet som selvstendige program ved hjelp av systemkall.

Når disse programmene er ferdige med redigeringen (som kan ta *lang* tid), tar systemdelen over kontrollen igjen for å sette audio- og videodelen sammen til en ny systemstrøm.

Systemdelen splitter også opp klipplista i en audio- og en videoliste som blir matet inn i henholdsvis audio- og videodelen.



Figur 27: Overordnet prosess- og flytdiagram.

5.2.2 Video

Denne delen er basert på arbeid utført våren 1996. For kompletthet er en mer utfyllende beskrivelse tatt med i denne rapporten.

I Figur 28 er videodelens prosess- og flytdiagram illustrert. Hvert enkelt steg er også illustrert i en serie med figurer.

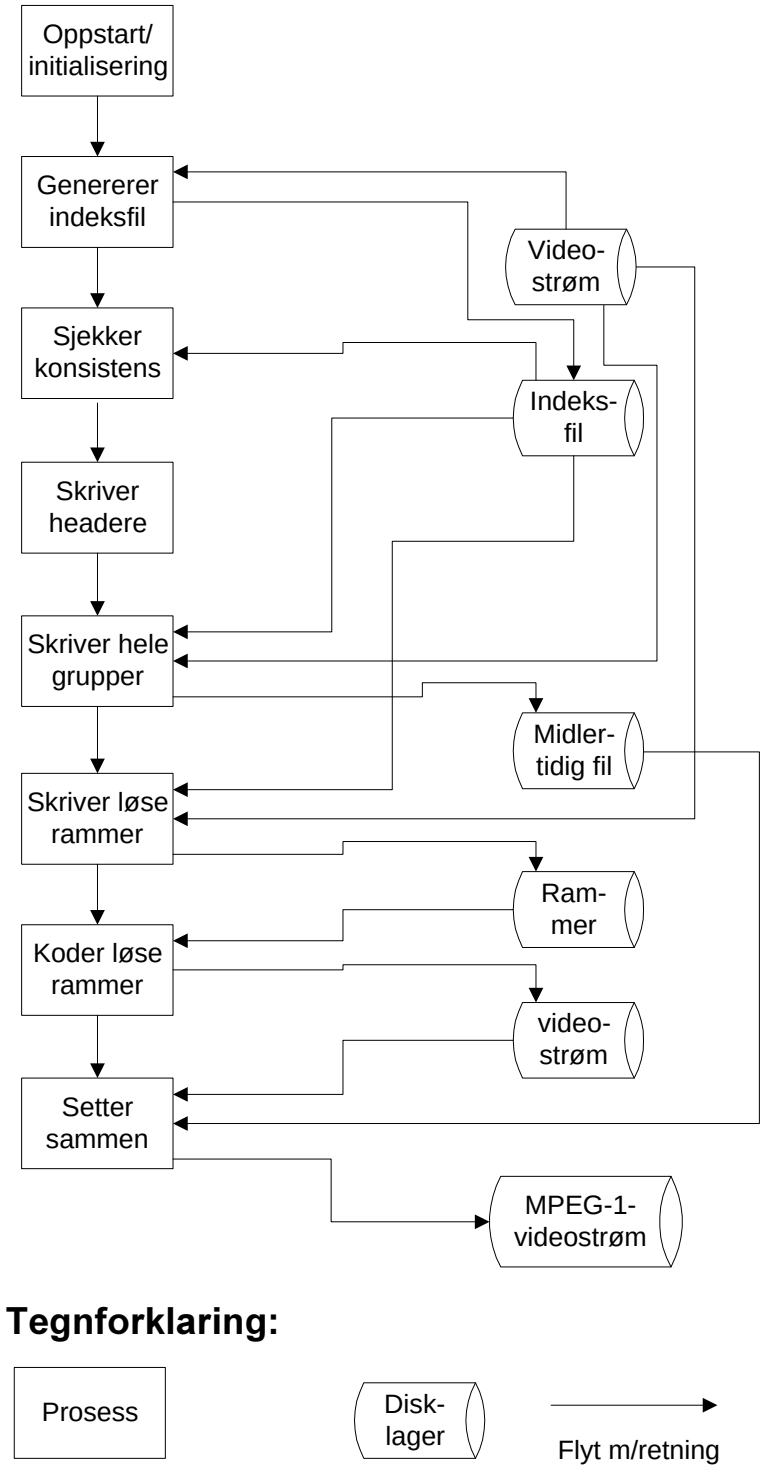
Videodelen starter med MPEG-1 videostrømmer (se Figur 29). Disse strømmen må tilfredsstille krav om lik bildehastighet, -størrelse og aspektrate.

Av disse blir det generert indeksfiler for hurtig oppslag til bestemte grupper i videostrømmen. Bildeparametrene blir lagt i toppen av disse indeksfilene for kontroll av konsistens. Det blir ikke generert indeksfil(er) dersom en eller flere allerede eksisterer. Dette blir gjort for å spare tid.

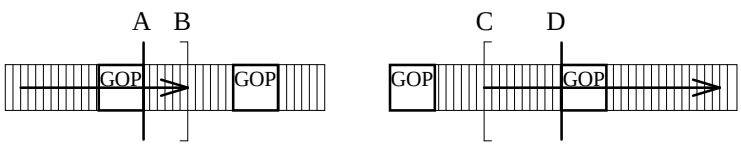
Deretter dekodes og lagres de rammene som ligger i gruppene som blir berørt av klippet, mens resten av gruppene forblir urørt (se Figur 30). Rammene lagres i PPM-formatet med filnavn som forteller om det er rammer som ligger før eller etter klippet.

De dekodede rammene settes sammen i riktig rekkefølge (se Figur 31) før de kodes på nytt til to nye grupper (se Figur 32), en for rammene til hver av de to opprinnelige strømmene. Til slutt blir de nye gruppene satt sammen med resten av de to opprinnelige strømmene (se Figur 33).

Alle gruppene i klippsonen har ny sekvensheader. Dette gjelder gruppa med rammer før klippet, gruppa med rammer etter klippet og resten av gruppene i den strømmen som ble klippet inn. Den sistnevnte sekvensheaderen hentes fra starten av samme strøm. Resultatet blir en ferdig editert strøm.

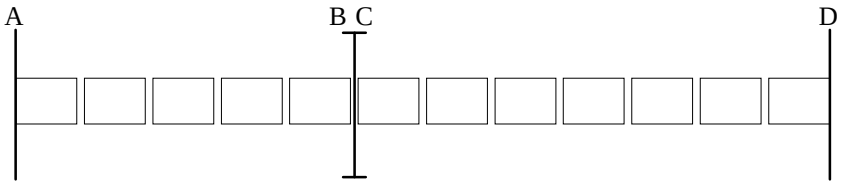


Figur 28: Prosess- og flytdiagram for videodelen.

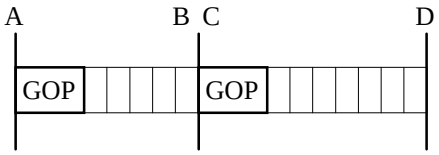


Figur 29: Klippunkt i to videostrømmer som skal settes sammen.

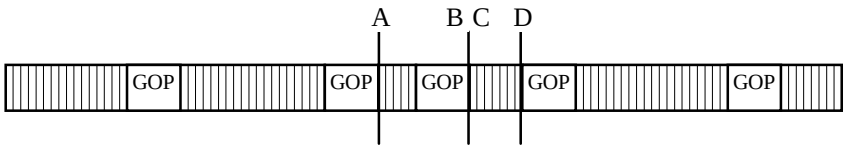
Figur 30: Oppsplitting i rammer og delfiler.



Figur 31: Sammensetting av rammene i klippsonen.



Figur 32: Koding av rammene i klippsonen.



Figur 33: Ferdig sammensatt sekvens.

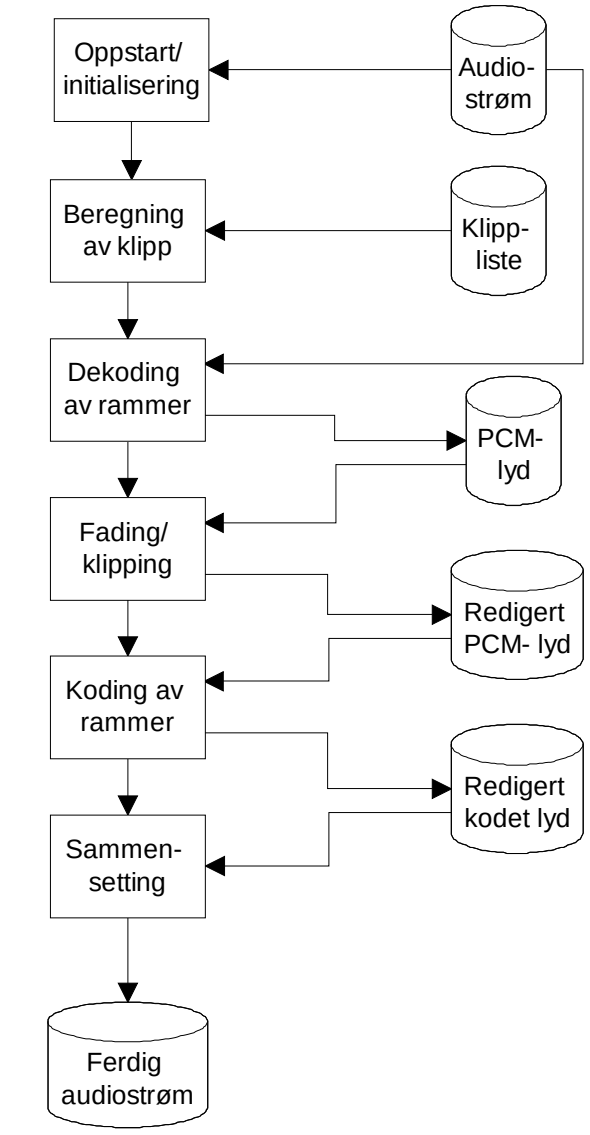
5.2.3 Audio

Prosess- og flytdiagram for audiodelen er illustrert i Figur 34.

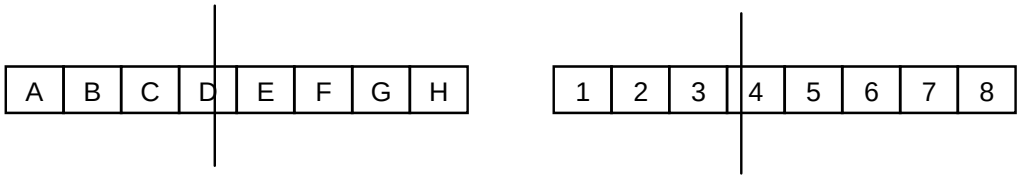
Audiodelen starter med å godkjenne MPEG-1 audiostrømmene. Den leser headerinformasjonen og sammenligner strømmene slik at de har like parametre. Samplingshastighet, bitrate, betoning (emphasis), lag og lydmodus må være like for at redigering skal tillates. Det blir også sjekket at det ikke er oppgitt en audiostrøm i klipplista som ikke eksisterer på kommandolinja.

Dersom strømmene blir godkjent, blir antall rammer som trengs for "fading" kalkulert. Antall rammer som blir dekodet avhenger av fadelengden som er oppgitt i klipplista. Dette blir gjort ved å dele fadelengden på rammelengde. Synkroniseringsberegningen blir akkumulert for hvert klipp, og tilstreber å holde seg innenfor de tidsrom som er oppgitt i 4.3.5 (-12 ms +24 ms). To strømmer som skal redigeres er illustrert i Figur 35.

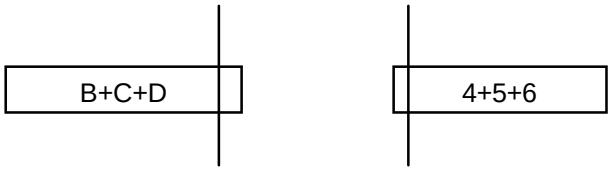
Deretter dekodes de rammene som ligger rundt kuttet (se Figur 36). En "fading" blir utført på de dekodede rammene (se Figur 37). Fade-funksjonen blir beregnet slik at amplituden når henholdsvis 0 og maks når det oppgitte tidspunkt er nådd. Hvordan dette blir når de to kuttene blir satt sammen er illustrert i Figur 38. Til slutt kodes kuttsekvensen (se Figur 39) og settes sammen med resten av de rammene som skal være med i den nye strømmen (se Figur 40). Disse rammen dekodes *ikke*.



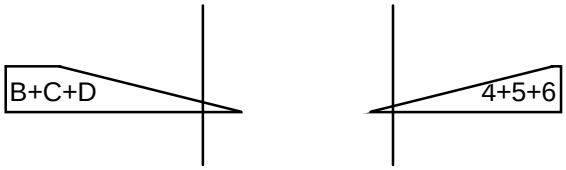
Figur 34: Prosess- og flytdiagram for audiodelen.



Figur 35: Klippunkt i to audiostrømmer som skal settes sammen.

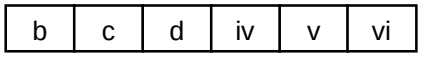


Figur 36: PCM-data fra de dekodede rammene fra begge audiostrømmene.

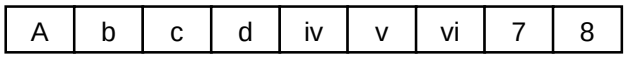


Figur 37: Etter "fading" av begge sider av kuttet.

Figur 38: Sammensatt audio med fading inn mot klippunktet.



Figur 39: Kodet klippsone.



Figur 40: Ny audiostrøm.

6. IMPLEMENTASJONSBESKRIVELSE AV SYSTEMDELEN

I 5.2.1 ble design og konstruksjon av systemdelen presentert. Dette kapittelet inneholder en detaljert beskrivelse av hvordan systemdelen er implementert.

6.1 Demultipleksing og multipleksing av MPEG-1 system

For å demultiplekse (splitte opp) en MPEG-1 systemstrøm i MPEG-1 audio- og videostrømmer benyttes et lite tilleggsprogram som følger med pakken "mpeg_encode" (se 7.2). Dette tilleggsprogrammet heter "mpeg_demux".

For å sette sammen MPEG-1 audio og video til en MPEG-1 systemstrøm benyttes programmet «system_encode» som er utviklet ved *The Multimedia Communications Lab, Boston University* i 1994.

6.1.1 Input til «mpeg_demux»

Demultiplekseren tar en MPEG-1 systemstrøm og splitter den opp i en MPEG-1 videostrøm og en MPEG-1 audiostrøm. Strømmene får filnavn som den opprinnelige fila med en ekstra filendelse. Filendelsen er «.vid» for videostrømmen og «.aud» for audiostrømmen.

Programmet tar også vare på systemdekodingsinformasjonen. Dvs. når de forskjellige pakkene skal dekodes og presenteres i forhold til systemklokka.

6.1.2 Kommandolinjeopsjoner – «mpeg_demux»

```
mpeg_demux InputMPS
```

InputMPS er MPEG-1 systemstrømmen.

6.1.3 Input til «system_encode»

Programmet «system_encode» trenger en MPEG-1 audiostrøm og en MPEG-1 videostrøm for å kode en MPEG-1 systemstrøm. Programmet skriver ut en teller som teller opp rammer etter hvert som de blir puttet inn i systemstrømmen. Til slutt skrives diverse kodingsdata ut.

Programmet kan også kode flere audio- og/eller videostrømmer sammen til en MPEG-1 systemstrøm. Alle strømmene angis etter hverandre med hvert sitt starttidspunkt på kommandolinja.

6.1.4 Kommandolinjeopsjoner – «system_encode»

```
system_encode [-r <byterate>] [-k <packstørrelse] [-s <pakker per
pack>] [-p <pakkestørrelse>] [-o <utfil>] [-CDROM] fil1 start_tid1
fil2 start_tid2...
```

6.1.4.1 -r <byterate>

Spesifiserer at byteraten skal holdes konstant på dette tallet.

6.1.4.2 -k <packstørrelse>

Setter størrelsen på hver pack. Se 2.3.2 for beskrivelse av Pack-laget.

6.1.4.3 -s <pakker per pack>

Setter antall pakker per pack. Se 2.3.4 for beskrivelse av Packet-laget.

6.1.4.4 -p <pakkestørrelse>

Størrelsen på systempakkene. Se 2.3.4 for beskrivelse av Packet-laget.

6.1.4.5 -o <utfil>

Navnet på utfila. Resultatet etter multipleksingen legges i denne fila.

6.1.4.6 -CDROM

Med *CDROM*-parameteret optimaliseres kodingen for 1x CDROM. Dvs. at kodingen sørger for at dataraten holder seg under 1,44 Mbit/s.

6.1.4.7 fil1

Navnet til den første strømmen som skal inn i MPEG-1 systemstrømmen. Det er likegyldig om dette er en MPEG-1 audio- eller videostrøm.

6.1.4.8 start_tid1

Starttiden til *fil1* må angis rett etter filnavnet. Med denne opsjonen kan de forskjellige strømmene som angis på kommandolinja til «system_encode» ha forskjellig starttidspunkt.

6.2 Klippliste

Når programsystemet startes på systemnivå (i motsetning til video- eller audionivå) må inputstrømmen være en MPEG-1 systemstrøm med en audio- og en videokanal. Klipplista inneholder definisjoner av klippsekvenser for både audio og video. Audio- og videolista ligger i samme fil. Audioklipplista har overskrifta “[Audio]” og videoklipplista har overskrifta “[Video]” (store eller små bokstaver spiller ingen rolle).

Denne felles klipplista blir oppsplittet i to klipplister, en for video og en audio. Fila splittes opp etter overskriftene nevnt over. Rekkefølgen på disse overskriftene velges fritt, men alt under “[Audio]” frem til enten filslutt eller “[Video]” blir lagt i audioklipplista og alt under “[Video]” frem til enten filslutt eller “[Audio]” blir lagt i videoklipplista.

Syntaksen for klipplistene er som forklart i 7.3 for videoklipplista og i 8.2 for audioklipplista.

Eksempel på systemklippliste:

```
[Audio]
0    1.234   13.434  5.2
1    3.304   20.344  4.45
0    15.789  35.14   8.754
1    14.754  25.4    3.1
1    0       10      2
[Video]
0    0       275
3    54      338
2    954     2087
0    789     3354
4    455     2524
```

For forklaring av denne klipplista henvises det til 7.3 og 8.2.

De oppsplittede klipplistene blir kalt *cliplist.audio* og *cliplist.video*. Før programmet terminerer slettes disse to filene fra filsystemet.

6.3 Oppstart av audio- og videoeditoren

Når alle filene er splittet opp i hver sin audio- og videodel startes hhv. audio- og videoeditoren med de oppsplittede filene som argument. Editorene kjøres som selvstendige program og startes med et systemkall hvor kommandolinja er parameter.

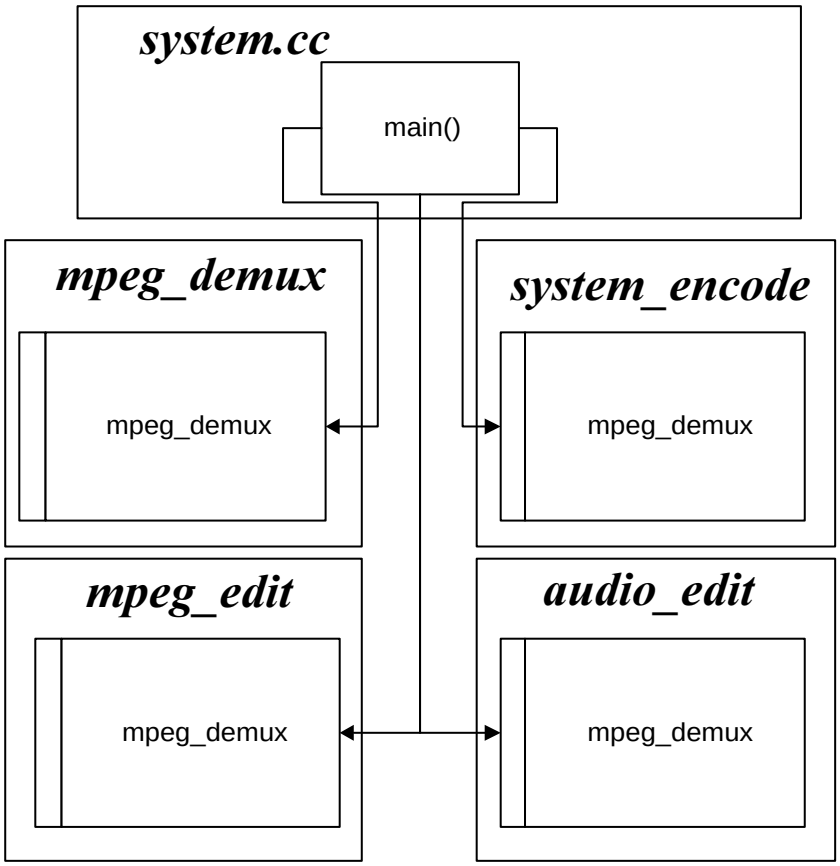
6.4 Sammensetting av MPEG-1 video og audio

Siste punkt for systemdelen er å sette sammen MPEG-1 audio- og videostrømmene til en MPEG-1 systemstrøm. Etter sammensettingen slettes audio- og videofilene.

6.5 Systemmoduler

Programmet består bare av en modul. Denne modulen heter *System* og håndterer oppsplitting av klipplista, oppstart av demultiplekseren, oppstart av audio- og videoeditoren og multipleksing av de ferdig redigerte audio- og videostrømmene.

Samspillet mellom programmene er illustrert i Figur 41.



Figur 41: Kontrollflyt for systemeditoren.

6.6 Kjøretidsinformasjon

De to programmene som benyttes til multipleksing og demultipleksing av MPEG-1 systemstrømmer skriver ut en del data om prosessen. Demultiplekseren skriver ut noe slikt:

```
Successful parse of MPEG system level
1 system headers, 9018 packs, 9018 packets
1670 audio packets, 6157 video packets, 1191 padding packets
0 reserved packets, 0/0 private type 1/2 packets
```

Multiplekseren teller opp rammer i kildestrømmene mens de legges inn i systemstrømmen. Dette gjøres for alle oppgitte strømmene. Etter denne opptellingen skrives alle strømmene til systemstrømmen, og en del data om den systemstrømmen som kodes skrives ut. Et eksempel på en utskrift kan se slik ut:

```
MPEG SYSTEM ENCODER ver. 1.0

Processing tt.mpa
13280 frames processed

Processing tt.mpg
5225 frames processed

Results of PASS 1
# Video Streams : 1
# Audio Streams : 1
Packets Per Pack: 3
Packet Size     : 2048
Output File     : tt

MPEG Video Stream
Filename        : tt.mpg
```

```
Start Time      : 0.00 sec.
Rate            : 25.00 frames/sec.
# of Frames     : 5225
Max. Frame Size : 7686 bytes

MPEG Audio Stream
Filename        : tt.mpa
Start Time      : 0.00 sec.
Rate            : 32000 samples/sec.
# of Frames     : 13280
Max. Frame Size : 2174 bytes

PASS 2: Writing tt.mps
1642 packs written
```

Multiplekseren skriver ut diverse data om de strømmene som er multiplekset til en MPEG-1 systemstrøm.

Systemnivåeditoren skriver ut en linje som forteller brukeren hvilken funksjon som blir startet rett før den startes. Denne utskriften kommer i tillegg til den ovenfor. En slik utskrift kan se slik ut:

```
Demultiplexing film2-300kbps-256-192.mps
Starting videoeditor
Starting audioeditor
Multiplexing edited audio and video into tt.mps
```

Dersom systemnivåeditoren blir kompilert med *-D_DEBUG* som en av opsjonene til *g++*, skriver programmet ut en del tilleggsinformasjon. Denne tilleggsinformasjonen er kommandolinja til alle programmene som startes med systemkall. Denne utskriften kommer i tillegg til de ovenfor. Denne kan se slik ut:

```
mpeg_demux film2-300kbps-256-192.mps
mpeg_edit film2-300kbps-256-192.mps.vid film3-300kbps-256-192.mps.vid tt.vid
cliplist.vid
audio_edit film2-300kbps-256-192.mps.aud film3-300kbps-256-192.mps.aud tt.vid
cliplist.vid
system_encode -o tt tt.vid 0 tt.aud 0
```

7. IMPLEMENTASJONSBESKRIVELSE AV VIDEODELEN

7.1 Dekoding av videorammer – Berkeleys “mpeg_play”

Berkeleys “mpeg_play” er den mest brukte MPEG-1-spilleren. Den er laget med tanke på portabilitet. I dette prosjektet kom jeg over et biblioteksgrensesnitt mot denne koden, “mpeg_lib”. Det er ikke særlig avansert, men det leverer ferdig dekodede rammer.

7.1.1 Funksjoner i “mpeg_lib”

Biblioteket består av bare noen få funksjoner for åpning og lukking av en strøm, setting av opsjoner og henting av rammer.

OpenMPEG	Denne funksjonen åpner en MPEG-1-videostrøm og leser inn parametrene som ligger i sekvensheaderen.
CloseMPEG	Denne funksjonen lukker videostrømmen og frigjør bufferne fra minnet.
GetMPEGFrame	Denne funksjonen henter en ramme fra strømmen, dekoder denne og returnerer den som en RGB-struktur. Rammene returneres i avspillingsrekkefølge.
SetMPEGOption	Denne funksjonen setter forskjellige opsjoner som har med dekodingen å gjøre, f.eks. dithering.

7.1.2 Manglende funksjonalitet i «mpeg_lib»

Dette biblioteket har ikke mulighet for posisjonering i strømmen. Det medfører at alle rammene fram til den ønskede rammen må dekodes. Koden til Berkeleys «mpeg_play» leser inn en MPEG-1-videostrøm i store porsjoner og legger de i et stort buffer. Dette vanskeliggjør posisjonering i fila på et senere tidspunkt fordi det ikke er mulig å finne ut hvor i fila en ny gruppe starter.

Det er ikke mulig å finne hvilken type ramme det er som er dekodet (I, B, eller P) ved hjelp av biblioteket. Det er heller ikke mulig å finne ut om den er hentet fra en ny gruppe.

Rammene returneres i avspillingsrekkefølge, men rammenummer eller Temporal Reference returneres ikke.

7.1.3 Henting av bestemte videorammer

På bakgrunn av manglene fra 7.1.2 er det nødvendig å parse fila på egen hånd for å generere indeksliste med aksesspunkter i fila. Funksjonen *GetMPEGFrame* måtte endres slik at den returnerer Temporal Reference.

Alle rammene etter gruppestarten som ligger rett foran klippet, må dekodes og lagres til fil etter å ha blitt konvertert til det grafiske formatet PPM.

7.2 Koding av videorammer – Berkeleys «mpeg_encode»

Berkeleys «mpeg_encode» [BERKPLA95] er den mest brukte koderen for MPEG-1 video. Denne koderen er i likhet med "mpeg_play", utviklet av Berkeley med tanke på portabilitet.

7.2.1 Input til «mpeg_encode»

Berkeleys «mpeg_encode» tar som input en bilderekkefølge i mange grafiske formater. Før kodingen må det genereres en parameterfil som forteller mpeg_encode hvordan kodingen skal utføres. Følgende parametre vil bli brukt i parameterfila til mpeg_encode:

PATTERN

Hvordan rammemønsteret i en gruppe skal være. Lovlige verdier er I, P og B. Mønster som skal benyttes: IBBBPBBBPBBBPBBBP. Dette mønsteret er avhengig av antall rammer som skal kodes. Den siste rammen er ALLTID en referanseramme (P), fordi «mpeg_encode» avslutter kodingen når siste referanseramme er møtt. I verste fall kunne dette føre til at de tre siste rammene ikke ble kodet. Dersom mønsteret avslutter med to P-rammer, blir den nest siste forandret til en B-ramme for å spare båndbredde.

OUTPUT

Navnet på fila hvor resultatet skal legges (MPEG-1-fil). Dette blir bare en midlertidig fil i programmet som slettes før programmet terminerer.

GOP_SIZE

Hvor mange rammer som skal være i en gruppe. Programmet setter GOP_SIZE til det antallet rammer som skal kodes. Dette tallet er alltid lavere enn gruppestørrelsen i kildefilene.

SLICES_PER_FRAME

Hvor mange snitt («slice») hver ramme skal inneholde. Det kan lønne seg å velge flere snitt dersom det er mye støy på linja siden det er lettere å gjenopprette etter feil, men det fører til dårligere komprimering. Ett snitt er mest brukt. Dette kan defineres i programmet.

CLOSED_GOP

Hvorvidt «closed_gop» skal brukes eller ikke. closed_gop er standard og blir derfor også benyttet i programmet.

FRAME_RATE

Rammehastigheten i antall rammer per sekund. Det antas at vanlig PAL bildehastigheten i kildefilene skal benyttes (disse blir sjekket for konsistens).

BASE_FILE_FORMAT

Hvilket bildeformat inn-filene er i. PPM-formatet er benyttet i programmet.

YUV_SIZE

Rammestørrelsen i pel x linjer. Denne verdien er kodet i de eksisterende sekvensheadere. De samme verdiene blir brukt.

INPUT_CONVERT

Hvilket konverteringsprogram som må kjøres for å få inn-filene på BASE_FILE_FORMAT. Dette feltet settes lik * når det ikke benyttes. Her skal det ikke benyttes.

INPUT_DIR

Hvilken katalog rammefilene skal hentes fra. Denne settes til stående katalog.

INPUT

Navn på rammefilene og nummerområdet som skal kodes. Disse kalles frame.*.ppm, hvor * erstattes av et tall i bildefølgen.

END_INPUT

Slutt på parametre som har med inputfiler å gjøre.

IQSCALE / BQSCALE / PQSCALE

Kvalitetsfaktor for hhv. I-, B- og P-rammer. Med denne verdien velges kompromisset mellom kvalitet og komprimering. Desto høyere verdien er desto dårligere blir kvaliteten. Men komprimeringen blir bedre med høyere verdi. Verdiene går fra 1 til 31. Tabell 11 viser noen Q-verdier oppført med tilhørende kvalitet målt med signal/støyforhold og komprimering. Disse parametrene kan defineres i programmet.

QSCALE	I-rammer	P-rammer	B-rammer
1	43,2 (2)	46,3 (2)	46,5 (2)
6	32,6 (7)	34,6 (10)	34,3 (15)
11	28,6 (11)	29,5 (18)	30,0 (43)
16	26,3 (15)	26,8 (29)	28,6 (97)
21	24,7 (19)	25,0 (41)	27,9 (173)
26	23,5 (24)	23,9 (56)	27,5 (256)
31	22,6 (28)	23,0 (73)	27,3 (330)

Tabell 11: Kvalitetsfaktorer med resulterende kvalitet (SNR). Komprimering i parentes (hentet fra[BERKPLA95]).

PIXEL

Spesifiserer om det skal foretas full- eller halvpikselvektorer ved søk etter bevegelsesvektorer. Denne kan defineres i programmet.

RANGE

Spesifiserer størrelsen på søkevinduet for bevegelsesvektorer i antall piksler. Denne kan defineres i programmet.

PSEARCH_ALG / BSEARCH_ALG

Spesifiserer søkealgoritmen som skal benyttes ved søk etter bevegelsesvektorer. Det er forskjellige algoritmer for B- og P-rammer. Disse er listet opp i Tabell 12 og Tabell 13 med tilhørende verdier for komprimering, hastighet og kvalitet. Hvilken algoritme som skal benyttes kan defineres i programmet.

Teknikk	Komprimering[4]	Hastighet[5]	Kvalitet[6]
Exhaustive	1000	1000	1000
SubSample	1008	2456	1000
TwoLevel	1009	3237	1000
Logarithmic	1085	8229	998

Tabell 12: P-ramme bevegelsesvektorsøk (hentet fra [BERKPLA95]).

Teknikk	Komprimering[4]	Hastighet[5]	Kvalitet[6]
Exhaustive	1000	?	1000
Cross2	975	1000	996
Simple	938	1765	991

Tabell 13: B-ramme bevegelsesvektorsøk (hentet fra [BERKPLA95]).

7.2.2 Parallell utførelse av «mpeg_encode»

Det er også mulig å kode rammer parallelt over et nettverk, men dette feltet er ikke behandlet eller benyttet i dette prosjektet.

7.3 Klippliste for videoredigering

For editeringen må det genereres en klippliste som inneholder opplysninger om hvilke deler av filene som skal trekkes ut, og i hvilken rekkefølge de skal settes sammen. Denne klipplista må genereres manuelt av brukeren før oppstart og gies som parameter på kommandolinja. Dersom programmet startes fra systemdelen, holder det at brukeren oppgir navnet på den felles klipplista når han skal starte systemdelen.

Formatet for klipplistefila er enkelt:

[4] Mindre tall betyr bedre komprimering.
[5] Større tall betyr hurtigere utførelse.
[6] Større tall betyr bedre kvalitet.

```
<strøm #><tab><første ramme><tab><siste ramme>
```

Eks.:

```
0     0     275
3     54    338
2     954   2087
0     789   3354
4     455   2524
```

Resultatet av denne klipplista vil bli en strøm som begynner med ramme 0 i strøm 0, og slutter med ramme 2524 i strøm 4. Strøm 0 er første inn-fil som ble angitt som startparameter.

7.4 Indeksfil

For å kunne editere på grunnlag av klipplista må det være mulig å gå inn til en bestemt ramme ved hjelp av posisjonen den har i fila («file offset»). For å løse dette problemet genereres det en indeksfil. Denne indeksfila inneholder data som trengs for å finne igjen hvor i fila denne rammen kan hentes ut.

Dersom det ikke blir funnet en indeksfil for alle MPEG-1 strømmene blir de filene som mangler generert av programmet ved oppstart.

Det har vist seg umulig eller svært vanskelig å benytte mpeg_lib for å generere en indeksfil. GetMPEGFrame returnerer ikke rammetype, og det finnes ingen mulighet for å finne ut om rammen er hentet fra starten av en ny gruppe.

«mpeg_lib» har blitt oppgradert til å returnere rammetype, og det er mulig å hente ut informasjon om ny gruppe dersom datastrukturen til Berkeley-koden leses (privat i biblioteket).

Det er dessverre ikke mulig å finne ut hvor i fila en ny gruppe har blitt lest fra fordi Berkeley-koden kjører med et buffer på ca. 80kB. Dvs. at ca. 80kB blir lest inn på en gang i minnet. Filposisjonen ligger derfor fast inntil bufferet blir fylt opp på nytt.

For å bøte på dette problemet, har jeg skrevet egne funksjoner for å parse av headerne (sekvens-, gruppe- og bildeheader). Først blir strømmen sjekket for om det er en MPEG-1-videostrøm (sekvensheader). For hver bildeheader blir rammetypen (I-, B- eller P-ramme) dekodet og tatt vare på for senere bruk.

7.4.1 Data som ligger i indeksfila

Verdiene som ligger i lagret for hver ramme i indeksfila er absolutt rammenummer (fra starten av fila), relativt rammenummer (internt i denne gruppa) og hvilken type ramme det er. Det blir også skrevet innslag i indeksfila hver gang en GOPheader blir møtt. For hver ny GOP blir filposisjonen skrevet.

Format for indeksfila:

Første linje:

```
<rammehøyde> <rammebredde> <rammehastighet> <aspektrate>
```

Øvrige linjer:

```
<absolutt rammenr.> <relativt rammenr.> <rammetype>
```

Dersom innslaget i indeksfila er en gruppe, er rammetype ‘G’. Da blir absolutt rammenr. satt til filposisjonen til gruppeheaderen og relativt rammenr. settes til 1024 fordi den bare er en «plassholder» slik at formatet er konsistent. Grunnen til at 1024 er valgt er at dette tallet aldri opptrer som relativt rammenummer [ISO 11172].

Alle tall er heksadesimale.

Eks.:

```
0070 00a0 0002 ffffffff
0000000c 400 G
0000 00 I
0003 03 P
0001 01 B
0002 02 B
0006 06 P
0004 04 B
0005 05 B
0009 09 P
0007 07 B
0008 08 B
000c 0c P
000a 0a B
000b 0b B
0000138a 400 G
000c 00 I
000f 03 P
```

7.4.2 Konsistens mellom videostrømmene som skal editeres

For at det skal være noen vits i å editere strømmer bør de samme parametrene for rammehøyde, -bredde, -hastighet og aspektrate gjelde for alle strømmene. Dette er ikke strengt nødvendig, men det er lite poeng i å endre rammestørrelsen midt inne i fila.

For å forsikre seg om at slikt ikke skjer, blir sekvensheaderen dekodet av OpenMPEG i mpeg_lib og lagt i toppen av indeksfila. Etter at alle indeksfilene er åpnet blir parametrene sammenlignet for konsistens. Programmet terminerer dersom parametrene ikke er de samme for alle filene.

7.5 Sammensetting av to videostrømmer

Rutinene for sammensetting av et vilkårlig antall strømmer, brytes ned til sammensetting av to strømmer. Dette gjøres for enkelthetens skyld. Det fører til dårligere kvalitet dersom to klipp ligger i den samme gruppa, men siden en gruppe sjelden er større enn ett sekund kommer ikke dette til å bli noe problem.

7.5.1 Ny sekvensheader

Ved hvert klipp legges det inn en ny sekvensheader i utfila. Dette er fullt lovlig ifølge standarden [ISO 11172], og benyttes her fordi det er en viss mulighet for at enkelte parametre er forskjellige, f.eks. hvorvidt nye kvantifiseringsmatriser skal benyttes. En teststrøm med flere sekvensheadere ble testet på «mpeg_play» før det ble benyttet i prosjektet.

7.5.2 Klipping av fil

Når fila skal klippes søkes det etter første ramme i klippet. Filposisjonen til gruppestarten som kommer etter, noteres. Deretter søkes det frem til siste bilde i klippet, og filposisjonen til siste gruppestarten før noteres.

Filposisjonen til den første gruppa søkes opp. Fra dette punktet kopieres fila byte for byte frem til filposisjonen til den siste gruppa over i en midlertidig fil. Denne prosessen gjentas for alle klippene.

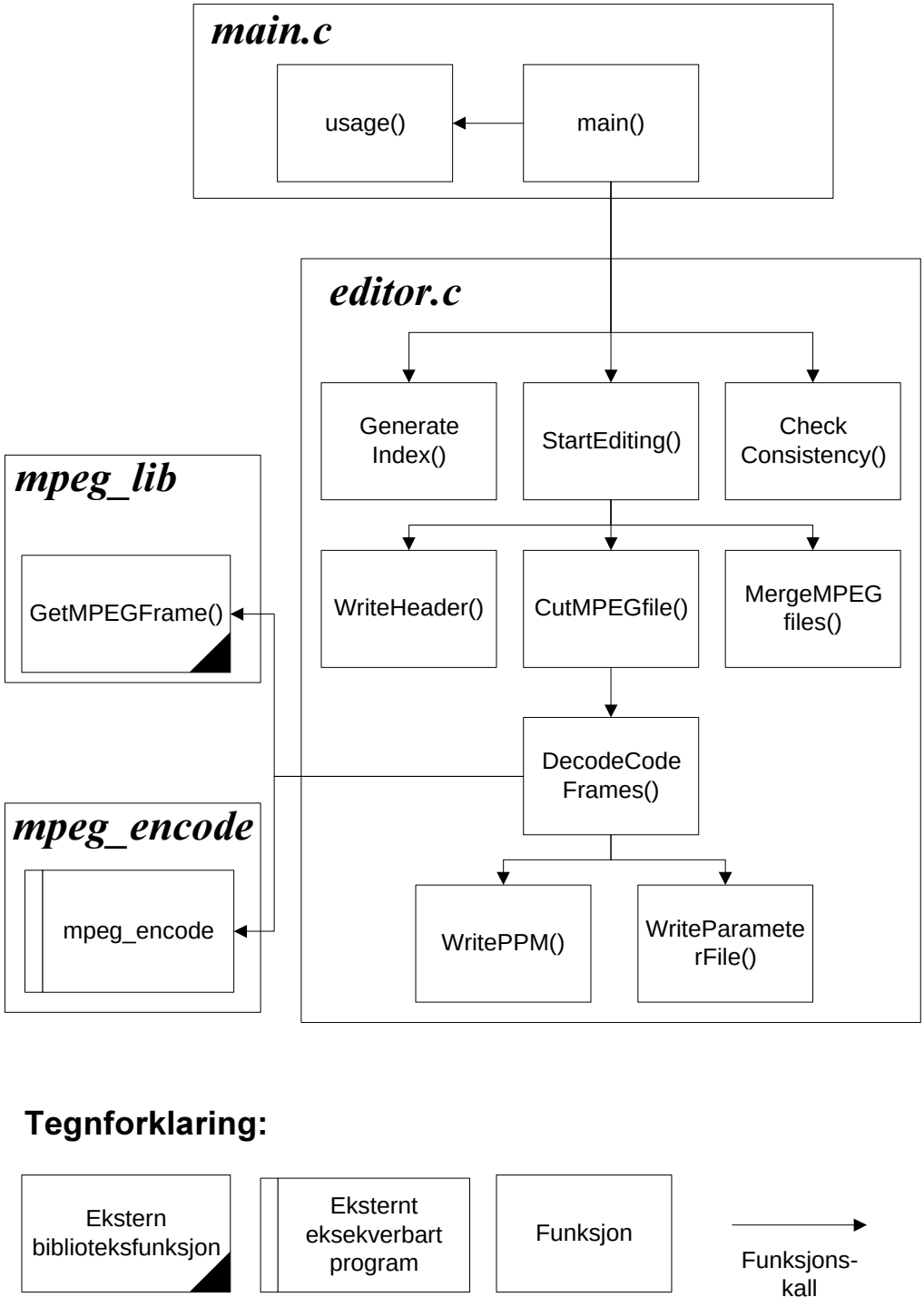
Deretter gjenstår den vanskeligste delen: dekoding og koding av rammene som ligger i de gruppene kuttene er foretatt. For å utføre denne oppgaven, må «mpeg_lib» benyttes.

GETMPEGFrame() kalles gjentatte ganger inntil den rammen det skal kuttes ved er nådd (gjelder både starten og slutten av klippet). Alle rammene i denne gruppen leses inn i et rammebuffer og ordnes i visningsrekkefølge. Deretter skrives rammene som ligger innenfor klippet til disk. Til slutt skrives parameterfila og gies som argument til «mpeg_encode».

7.6 Systemmodulene

Systemet er delt inn i to moduler – en for oppstart og initialisering, og en for å utføre selve editeringen som vist i Figur 42. Et resultat av dette er at editormodulen (*editor.c*) er blitt ganske stor (ca. 800 linjer, 22k) i forhold til mainmodulen (*main.c*) (ca. 100 linjer, 2k).

Wrappermodulen (*wrapper.c*) fra «mpeg_lib» er tatt med i beskrivelsen fordi den er noe endret fra sin originale versjon.



Figur 42: Kontrollflyt for videoeditoren.

7.6.1 Mainmodulen

Mainmodulen starter programmet, starter rutinene for generering av indeksfiler, konsistenssjekk og selve editeringen.

Modulen inneholder disse funksjonene:

*int main(int argc, char **argv)*

Denne funksjonen utfører alle de punktene som står nevnt ovenfor. I tillegg starter den funksjonen *usage(argv[0])* dersom det ikke er tilstrekkelig antall parametre på kommandolinjen.

*void usage(char *pname)*

Denne funksjonen skriver ut bruksmåte.. Bruksmåte:

```
mpeg_edit <inputfile#1> <inputfile#2> <destination> <cliplist>
```

Kalles av *main().*

7.6.2 Editormodulen

Editormodulen inneholder alle funksjonene som har med editeringen å gjøre. Den styrer bruken av «mpeg_lib» og «mpeg_encode». I toppen av modulen ligger definisjonene som forteller «mpeg_encode» hvordan rammene skal kodes, hvor stort buffer som skal brukes, navnet på parameterfila og navnet (med sti) til «mpeg_encode».

Modulen inneholder disse funksjonene:

*unsigned long int readByte(FILE *stream)*

Denne funksjonen leser inn en byte (8 bit) fra *stream* inn i en unsigned long int. Fordi bitene er venstrejustert, må de skiftes 24 plasser mot høyre slik at de blir høyrejustert.

Kalles av *picture(), generateIndex(), WriteHeader()* og *CutMPEGfile().*

*void picture(FILE *video, pictureinfo *info)*

Denne funksjonen parser av bildeheaderen i fila *video* og leser informasjonen som ligger der. Denne returneres i datastrukturen *info* som er av typen *pictureinfo.* Denne datastrukturen inneholder informasjon om Temporal Reference og rammetype (I, P eller B). Fordi MPEG-1-standarden [ISO 11172] bruker enkeltbiter for å sette variabler, må det en del bitskifting til.

Kalles av *generateIndex().*

*int generateIndex(FILE *video, FILE *index)*

Denne funksjonen genererer indeksfila som er beskrevet i 7.4.

Den starter med å sjekke om det er en gyldig MPEG-1-videostrøm i filen *video* ved å sjekke om de første 32 bitene er en sekvensstartkode. Dersom dette er tilfelle, parses fila helt til filslutt. Indeksen legges i fila *index.*

Åtte bit leses inn om gangen og skiftes inn fra høyre i et 32 biters buffer, for å sjekke om det er en startkode. Startkodene starter alltid med 23 biter som er null. Ved hjelp av Temporal Reference fra *picture()*, telles rammenummeret opp fra starten av fila. Hver gang en gruppeheader blir møtt, legges filposisjonen inn i indeksfila.

Kalles av *main().*

*int WritePPM(char *rawpic, unsigned int size, unsigned int height, unsigned int width, FILE *PPMfile)*

WritePPM konverterer BGR-data til RGB og skriver til fil. I toppen av fila legges en *PPMheader* med informasjon om filformatet (P6), bildets bredde og høyde, og maksimumsverdien av RGB-verdiene som er 255 (24 bits grafikk).

GetMPEGFrame returnerer rammen på et BGR-format (blå, grønn, rød) i bytekvartetter. Første byte i kvartetten er *alltid* null, andre byte står for blåkomponenten, tredje for grønnkomponenten og fjerde for rødkomponenten for et piksel.

Før rammen kan skrives til fil, må første byte i denne kvartetten kastes, og de tre gjenværende bytte rekkefølge slik at resultatet blir gyldig RGB-data som kan skrives til fil. PPM-formatet er valgt fordi det innebærer minst konvertering. Det fører også til raskere koding fordi det er ukomprimert.

Kalles av *DecodeCodeFrames().*

int WriteParameterFile(int first_frame, int last_frame, unsigned long int clipno)

Denne funksjonen skriver parameterfila som brukes av «mpeg_encode» til kodingen av rammene. De fleste av parametrene kan skaleres i toppen av modulen.

Rammene som ligger rett etter kuttet frem til første gruppestart kodes til en fil som heter *pre$i.mpg*, og rammen fra siste gruppeheader frem til klippet kodes til en fil som heter *post$i.mpg*. *$i* er indeksnummeret til klippet (linjenummeret i klipplista).

Ved å beregne antall rammer som skal kodes, beregnes rammemønsteret. Mønster som benyttes: *IBBBPBBBPBBBP.* Mønsteret starter *alltid* med en I-ramme, og slutter *alltid* med en P-ramme. Årsaken til at klippet alltid må slutte med en referanseramme (her en P-ramme) er at ellers avslutter «mpeg_encode» kodingen ved siste referanseramme. Dette kan i verste fall føre til at de to siste rammene ikke blir kodet.

For å bøte på dette, settes siste ramme til *P.* Dersom den nest siste rammen også er en *P*, settes denne til *B* for å spare båndbredde.

Alle andre parametrene skrives til parameterfila med de verdiene som er definert i toppen av modulen.

Kalles av *DecodeCodeFrames().*

int DecodeCodeFrames(unsigned long int start_frameno, unsigned long int end_frameno, unsigned long int clipno)

DecodeCodeFrames er funksjonen som styrer dekoding og koding av de rammene som ligger utenfor de hele gruppene.

Denne funksjonen leser seg fram til første ramme som skal dekodes ved hjelp av *GetMPEGFrame.* Når denne rammen er nådd, legges alle rammene i denne gruppen i et rammebuffer. De rammene som ligger innenfor klippet skrives så til fil ved hjelp av *WritePPM().* Deretter startes *WriteParameterFile()* for å skrive parameterfila til «mpeg_encode». Til slutt startes «mpeg_encode» med filnavnet til parameterfila som argument. Både navnet på programmet «mpeg_encode» og navnet på parameterfila kan defineres i toppen av modulen.

Samme prosessen gjentas for rammene fra siste gruppeheader før slutten av klippet fram til kuttet. Dersom «mpeg_encode» feiler av en aller annen grunn, medfører det bare at programmet blir redusert til et gruppe-editeringsprogram.

Kalles fra *CutMPEGfile().*

*int CheckConsistency(FILE **index, unsigned char fcount)*

Denne funksjonen sjekker om filene har samme kritiske parametre utfra første linje i indeksfila (beskrevet i 7.4.1). Disse parametrene er bildehøyde, -bredde og -hastighet.

Kalles av *main().*

*int WriteHeader(FILE *index, FILE *input, FILE *tempfile)*

WriteHeader skriver hele sekvensheaderen til en midlertidig fil (*tempfile)*. *CutMPEGfile()* skriver senere de hele gruppene til samme fila.

Funksjonen starter med å finne filposisjonen til første gruppestart fra indeksfila. Deretter kopieres byte for byte over fra *input* til *tempfile* inntil den første gruppestarten er nådd.

Kalles av *StartEditing().*

*int CutMPEGfile(FILE *index, FILE *input, cliplistentry cliplist, FILE *tempfile, unsigned long int clipno)*

Denne funksjonen kutter av de hele gruppene som ligger innenfor klippet. Gruppene skrives til en midlertidig fil (*tempfile*). Dette er den samme midlertidige fila som *WriteHeader()* skrev sekvensheaderen til.

Funksjonen starter med å lese seg fram til rammenummeret til starten av klippet. Filposisjonen til første gruppestart etter denne rammen taes vare på (fra indeksfila). Deretter søkes slutten av klippet opp, og filposisjonen til siste gruppestart før denne rammen taes vare på. Til slutt posisjoneres filpekeren til den første gruppestarten, og byte for byte kopieres over fra *input* til *tempfile* inntil den siste gruppestarten er nådd.

Deretter kalles *DecodeCodeFrames()* for å dekode og kode de løse rammene.

Kalles av *StartEditing().*

*int MergeMPEGfiles(FILE **tempfile, FILE *output, int entries)*

Denne funksjonene kalles helt til slutt i programmet for å sette sammen alle klippene til ett.

Den starter med å sjekke om det finnes en *pre$i.mpg*-fil (forklart i *DecodeCodeFrames()*) hvor *$i* er et tall fra 0 til *entries*. Denne skrives så til *output* byte for byte med unntak av sekvens-stoppkoden (forklart i 2.4.3). Deretter starter kopieringen av de midlertidige filene som hører til dette klippet til *output*. Til slutt sjekkes det om det om det finnes en *post$i.mpg*-fil fil (forklart i *DecodeCodeFrames()*). Denne skrives, i likhet med *pre$i.mpg*, til output byte for byte med unntak av sekvens-stopkoden.

For å unngå å skrive ut sekvens-stoppkoden, må det litt buffring til. Dersom to byte har verdien ‘0’ og den neste ‘1’, har vi en kandidat til sekvens-stoppkode. Da må det

sjekkes om neste byte stemmer overens med de to siste bytene i sekvens-stoppkoden. I såfall må ingen av de siste 32 bitene (4 byte) skrives til *output*. Dette innebærer en del bitskifting fordi kodene ikke er justert på 32-bits plasser, men 8-bits plasser (byte) i strømmen, selv om kodene har en 32-bits verdi.

Kalles av *StartEditing().*

*int StartEditing(FILE **input, FILE *output, FILE **index, FILE *clipfile, int fcount)*

StartEditing() startes av *main()* etter at alt er klart for start av editeringen. Denne funksjonen styrer rekkefølgen på de forskjellige underprosessene.

Den starter med å lese klipplisten inn i en datastruktur som består av klippnummer, filnummer, første ramme og siste ramme i klippet. For hvert klippnummer (egentlig linjenummer i klippfila) opprettes en midlertidig fil og *WriteHeader()* og *CutMPEGfile()* kalles.

Til slutt kalles *MergeMPEGfiles()* for å sette sammen klippene i riktig rekkefølge.

Kalles av *main().*

7.6.3 Wrappermodulen

Wrappermodulen (*wrapper.c*) inneholder implementasjonen til grensesnittsfunksjonene til «mpeg_lib». Denne modulen tar seg av all kontroll med koden til «mpeg_play».

Beskrivelse av funksjonene:

*Boolean OpenMPEG(FILE *MPEGfile, ImageDesc *ImgInfo)*

Denne funksjonen tar seg opprettelsen av et videostrøm-objekt. Den kaller initialiseringsfunksjonen til «mpeg_play» (*NewVidStream()*), og kaller deretter *mpegVidRscr()* for å lese inn alle parametre fra sekvensheaderen til videostrømmen.

void CloseMPEG()

Denne funksjonen stenger videostrømmen. Den deallokerer alle objekter fra minnet.

*Boolean RewindMPEG(FILE *MPEGfile, ImageDesc *Image)*

Denne funksjonen spoler tilbake strømmen til begynnelsen. Den lukker først strømmen med *CloseMPEG()*, spoler tilbake fila og åpner strømmen igjen med *OpenMPEG()*.

Denne funksjonen blir ikke benyttet.

*void GetMPEGInfo(VidStream *vid_stream, ImageDesc *Info)*

Denne funksjonen returnerer informasjonen fra sekvensheaderen. Den blir benyttet av *OpenMPEG()*.

Forøvrig blir denne funksjonen ikke benyttet.

void SetMPEGOption(MPEGOptionEnum Option, int Value)

Denne funksjonen setter en del dekodingsparametre som har med fremvisning av farger å gjøre (*MPEG_DITHER, MPEG_LUM_RANGE, MPEG_CR_RANGE* og *MPEG_CB_RANGE*). Alle fargemetodene som er tilgjengelig under «mpeg_play» kan benyttes. Denne funksjonen må kalles før *OpenMPEG()*.

Denne funksjonen benyttes til å sette dithering til full-farge (24 bit).

*Boolean GetMPEGFrame(char *Frame, unsigned int *temp_ref)*

Denne funksjonen henter ned én ramme fra strømmen, ferdig dekodet i henhold til metoden(e) som er valgt i *SetMPEGOption()* og returnerer denne i *Frame*-variabelen. Denne funksjonen har fått et tilleggsparameter i forhold til originalen; *temp_ref*. Denne variabelen sender med temporær referanse til den rammen som nettopp er dekodet.

Variabelen blir sendt videre til funksjonen *VidStream *mpegVidRsrc(time_stamp, vid_stream, temp_ref)* i modulen *video.c* som hører til koden til «mpeg_play». Funksjonen *mpegVidRsrc()* er også endret tilsvarende fra originalen slik at den setter *temp_ref* til den rammen som nettopp er blitt dekodet.

7.6.4 Videomodulen

Denne er en del av Berkeleys «mpeg_play» kode.

Funksjonen *mpegVidRscr()* i fila *video.c* måtte endres noe i tillegg til det som er nevnt under *GetMPEGFrame().*

Dersom slutten ble nådd på MPEG-1-strømmen, sørget funksjonen for å deallokere videostrømmen uten å informere om det. Denne funksjonaliteten ble fjernet fordi *CloseMPEG()* fører til kjøretidsfeil dersom den prøver å deallokere en allerede deallokert videostrøm. Det ble heller ikke ansett som nødvendig å beholde denne funksjonaliteten siden programmet alltid deallokerer videostrømmen.

7.7 Kjøretidsinformasjon

Programmet skriver kontinuerlig ut meldinger til brukeren om hva som blir gjort.

Dersom det må genereres indeksfiler, skrives meldingen *«Generating index files...»* ut. Mens rammen i strømmen telles opp, skrives rammenummrene opp for første ramme i hver gruppe.

Når indeksfilene er generert/åpnet for alle filene, starter sammenligningen av de kritiske parametrene som ligger lagret i toppen av indeksfila. Mens sammenligningen utføres skrives parametrene ut. Dersom det er inkonsistens mellom parametrene, skrives det ut en feilmelding og programmet avslutter.

Deretter leses klipplisten inn og skrives ut til skjermen. Dette gir brukeren en mulighet til å sjekke at korrekt klippliste blir brukt.

Mens programmet søker seg fram til kuttet, skrives rammenumrene ut til skjermen med teksten *«Reading to start of clip $i: $j»* (*$i* er klippnummeret og *$j* er rammenummeret som leses akkurat nå). Når rammen kuttet ligger ved er funnet og rammene skal skrives til disk, skrives meldingen *«Writing frame $j»* ut til skjermen.

Etter at rammen er skrevet til disk, startes «mpeg_encode» med meldingen *«Starting mpeg_encode»*. «mpeg_encode» skriver ut sine egne meldinger til skjermen. Blant

disse er hvilke metoder som skal benyttes ved kodingen, gruppemønsteret, antall rammer som skal kodes og hvor lang tid som er igjen før kodingen er fullført.

Det samme gjentar seg for slutten av klippet og for alle klippene.

8. IMPLEMENTASJONSBESKRIVELSE AV AUDIODELEN

8.1 Dekoding og koding av audiorammer

For dekoding av audiorammer benyttes “musicout”, og for koding av PCM benyttes “musicin”. Disse to programmene er hentet fra samme pakke og konverterer mellom MPEG-1 audio layer I eller II og PCM rådata. PCM rådata er forklart under 8.1.1.

Dekoderen “musicin”, håndterer både psykoakustisk modell 1 og 2. Begge programmene håndterer alle lydmodi (stereo, mono og to kanaler; se også 2.5.6), faste bitrater (fri bitrate er ikke implementert) og samplingsfrekvenser (32 kHz; 44,1 kHz; 48 kHz). Programmene startes med kommandolinjeparametre som er forklart i 8.1.3 for “musicin” og i 8.1.4 for “musicout”.

Ytterligere dokumentasjon på disse to programmene har dessverre ikke latt seg oppdrive. Det lille som foreligger, forstås kun ut fra informasjon programmet skriver ut til skjermen. Disse programmene er først og fremst brukt til koding og dekoding av musikk, noe som skulle gi en forklaring på navnene.

8.1.1 Input til “musicin”

Programmet “musicin” koder MPEG-1 audiostrømmer utfra PCM rådata. PCM rådata vil si at samplingene ligger sekvensielt etter hverandre i fila. Fila inneholder ikke noen header, det vil si at opplysninger om samplingshastighet, stereo eller mono, bitoppløsning på samplingene (8 eller 16 bit) og om det er signed eller unsigned data. Dersom det er stereo ligger kanalene multiplekset med høyre, venstre, høyre, venstre, høyre osv.

Dersom det blir oppgitt feil parametre til “musicin” for samplingshastighet eller lydmodus, kommer lyden til å bli kodet utfra de oppgitte parametrene. Med andre ord kan det bli nærmest umulig å rekonstruere lyden uten rare effekter.

8.1.2 Input til “musicout”

Programmet “musicout” dekoder en MPEG-1 audiostrøm og genererer PCM rådata. Siden PCM rådata er uten headerinformasjon, må samplingshastighet og lydmodus tas vare på dersom rådatafila skal være til nytte på et senere tidspunkt.

PCM rådata er forklart i 8.1.1.

8.1.3 Kommandolinjeopsjoner – “musicin”

Alle parametrene fra de innkommende audiostrømmene tas vare på og brukes for å starte koderen med de rette argumentene. På denne måten får hele den ferdig redigerte strømmen samme karakteristikk. Blant disse er konstant bitrate, dvs. bitraten er konstant gjennom hele strømmen, også over klippunktene.

```
musicin -l <layer> -m <sndmode> -p <psymod> -s <smplfrq> -b
<bitrate> -d <emphasis> InputPCM OutputMPA
```

8.1.3.1 -l <layer>

Dette parameteret forteller koderen hvilket lag MPEG-1 audiostrømmen skal være kodet i. Koderen håndterer bare lag 1 og lag 2. Disse lagene er forklart i 2.5.10.

‘1’ brukes for lag 1 og ‘2’ brukes for lag 2.

8.1.3.2 -m <sndmode>

Hvilken lydmodus som skal benyttes for kodingen. Det er mulig å kode i mono, stereo, joint stereo eller to kanaler. De forskjellige lydmodi er forklart i 2.5.6. ‘m’ benyttes for mono, ‘s’ for stereo, ‘d’ for to kanaler og ‘j’ for joint stereo koding.

8.1.3.3 -p <psymod>

Psykoakustisk modell. Koderen støtter både psykoakustisk modell 1 og psykoakustisk modell 2. ‘1’ brukes for psykoakustisk modell 1 og ‘2’ brukes for psykoakustisk modell 2. De psykoakustiske modellene er forklart i 2.5.12.

8.1.3.4 -s <smplfrq>

Samplingsfrekvens på PCM-strømmen. Denne må oppgis fordi det ikke er mulig å finne ut noe som helst om PCM rådata fordi denne ikke inneholder noen startheader. Samplingsfrekvenser er forklart i 2.5.5. Samplingsfrekvensen skrives rett ut i antall Hz (32000, 44100 eller 48000).

8.1.3.5 -b <bitrate>

Hvilken bitrate som skal benyttes for kodingen. Denne varierer fra 32 kbit/s til 448 kbit/s for lag 1 og fra 32 kbit/s til 384 kbit/s for lag 2. Bitraten er forklart i 2.5.8.

8.1.3.6 -d <emphasis>

Betoning på lyden. Denne er stort sett ikke brukt, men er med fordi standarden krever det [ISO 11172]. ‘n’ brukes for ingen (none), ‘c’ for CCITT J.17 og ‘5’ for 50/15 μs.

8.1.3.7 InputPCM

Navnet på fila som inneholder PCM rådata. Denne må være som forklart i 8.1.1.

8.1.3.8 OutputMPA

Navnet på MPEG-1 audiofila som skal kodes utfra opsjonene over og PCM-fila *InputPCM*.

8.1.4 Kommandolinjeopsjoner – “musicout”

For dekode en MPEG-1 audiostrøm trenger ikke brukeren å vite noe som helst om kodingsparamtrene. Dette finner «musicout» ut selv fordi denne informasjonen ligger i MPEG-1 headeren.

```
musicout InputMPA OutputPCM
```

8.1.4.1 InputMPA

Navnet på MPEG-1 audiofila som skal dekodes.

8.1.4.2 OutputPCM

Navnet på PCM rådatafila den dekodede audiostrømmen skal legges i.

8.2 Klippliste for audioredigering

For editeringen må det, i likhet med videodelen, genereres en klippliste som forteller programmet hvilke deler av filene som skal trekkes ut, og i hvilken rekkefølge de skal settes sammen. Denne klipplisten må genereres manuelt av brukeren før oppstart og gies som parameter på kommandolinja. Dersom programmet startes fra systemdelen, holder det at brukeren oppgir navnet på den felles klipplista når han skal starte systemdelen.

Formatet for klipplistefila er enkelt:

```
<strøm #><tab><starttid><tab><sluttid><tab><"fade"-tid>
```

Alle tidsangivelser skjer i sekunder med *inntil* tre desimalers nøyaktighet (millisekunds nøyaktighet).

Eks.:

```
0    0.540   275.458 1.45
3    54.852  338.782 1.12
2    954.458 2087.452        0.75
0    789.159 3354.126        2
4    455.654 2524.782        1.75
```

Resultatet av denne klipplista vil bli en strøm som begynner 540 ms ut i strøm 0, og slutter 2524,782 sekunder ut i strøm 4. Strøm 0 er første inn-fil som ble angitt som startparameter.

8.3 Konsistens mellom audiostrømmene som skal redigeres

For at det skal være mulig å sette sammen to eller flere audiostrømmer, må enkelte parametre i alle strømmen stemme overens. Noen dekodere kan takle endringer av disse parametrene midt i en audiostrøm, men standarden krever det ikke [ISO 11172]. Det er derfor mest naturlig å legge seg på det som de aller fleste dekodere klarer å håndtere.

Headeren på alle strømmene blir lest inn i begynnelsen av programmet og sammenlignet. Dersom det blir funnet inkonsistens, avbrytes programmet med en feilmelding som sier hvilket parameter som ikke hadde samme verdi i de to strømmene som ble sammenlignet.

8.4 Sammensetting av to audiostrømmer

Sammensetting av to audiostrømmer er ganske likt sammensetting av videostrømmer.

8.4.1 Klipping av fil

Ved klipping i fila regnes det ut hvilken rammestart som ligger nærmest det angitte klipptidspunktet. Dersom det allerede er foretatt ett eller flere klipp, trekkes den synkroniseringsfeilen som allerede eksisterer med i vurderingen av hvor klippet skal ligge. F.eks. dersom vi har valget mellom en rammestart som ligger +10 ms og en

som ligger -5 ms, kommer -5 ms til å bli valgt. Men dersom vi allerede har en synkroniseringsfeil på -7 ms, kommer den akkumulerte synkroniseringsfeilen til å bli -12 ms
(-5 ms + (-7ms) = -12 ms) med denne rammestarten, men ikke mer enn +3 ms
(+10 ms + (-7 ms) = +3 ms) med den andre rammestarten.

8.4.2 Nivåjustering – "fading"

Nivåjustering som internasjonalt kalles "fading", utføres ved en monotont stigende eller avtagende "gain-control"-funksjon. Denne operasjonen må utføres på PCM for at den skal fungere.

Dersom det er valgt en "fade"-tid som er større enn null, regner programmet ut hvilket rammeantall som må dekodes for å få til dette. F.eks. dersom det er valgt en "fade"-tid på 1,750 sekunder vil vi velge å dekode 68 rammer (44,1 kHz, lag 2: 1,750 ms/26 ms/ramme = 67,31).

Resultatet av dekodingen av disse rammene er PCM rådata som kan manipuleres direkte. Det velges en funksjon som vil resultere i et nullsignal i klippunktet. For beregning av fade-formel benyttes denne formelen for fading inn:

$$g_{inn} = \frac{i}{f_{sampl} \cdot t_{fade}} \ , \quad 0 \le i \le f_{sampl} \cdot t_{fade}$$

(2)

Og denne for fading ut:

$$g_{ut} = 1 - \frac{i}{f_{sampl} \cdot t_{fade}} \ , \quad 0 \le i \le f_{sampl} \cdot t_{fade}$$

(3)

Bokstaven *i* representerer en tellevariabel som økes med en for hver sampling, t_{fade} er fadetiden og f_{sampl} er samplingsfrekvensen. Telleren *i* må ikke overstige $f_{sampl} \cdot t_{fade}$ fordi *g* da vil overstige 1. Resultatet blir da et signal som blir høyere i amplitude enn det originale.

Med eksempelet ovenfor vil vi velge funksjonen $g_{inn}=i$/44100Hz*1,750s = i/77175 for "fading" inn og $g_{ut}=1-i$/77175 for "fading" ut. Men vi må holde oversikt over hvor mange samplinger inne i PCM-strømmen vi skal begynne å fade. Dette beregnes med følgende formel:

$$t_{dekodet} - t_{fade} = t_{fadestart}$$
$$s_{fadestart} = t_{fadestart} \cdot f_{sampl}$$

(4)

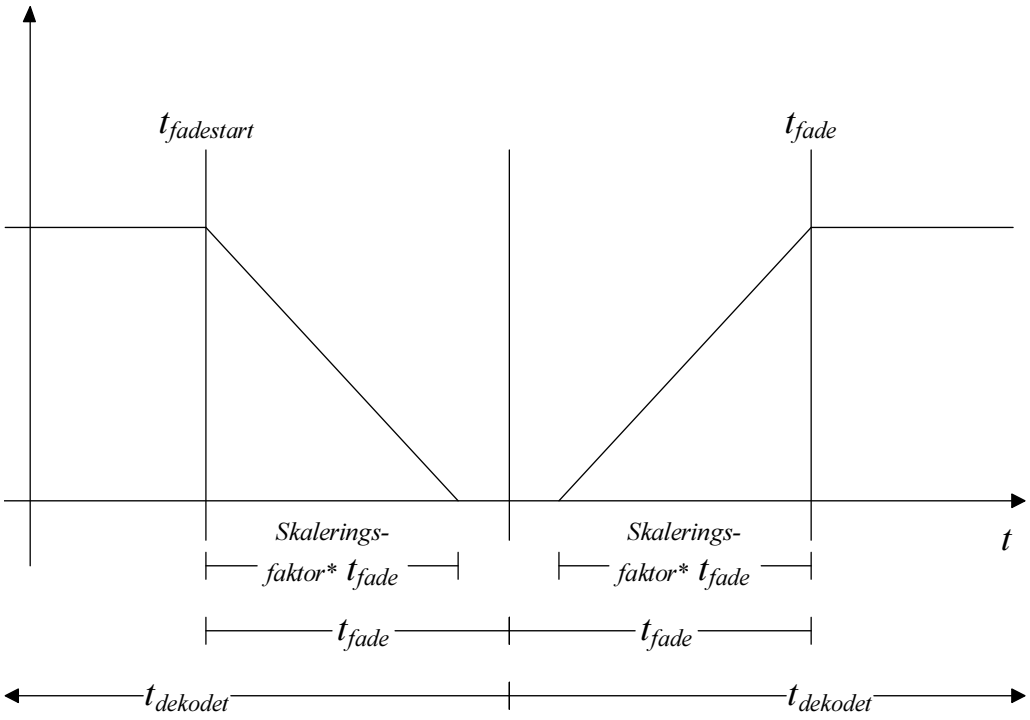
Hvor $t_{dekodet}$ er den tiden som er dekodet (ant. rammer multiplisert med rammelengde), $t_{fadestart}$ er på hvilket tidspunkt i de dekodede rammene fadingen skal starte og $s_{fadestart}$ er antall samplinger ute i de dekodede rammene fadingen skal starte. Øvrige parametre som før. Fading er illustrert i Figur 43.

Det er mulig å definere en skaleringsfaktor for fadingen. Denne skal være '1' dersom innfadingen skal starte akkurat i overgangen mellom klippene og utfadingen skal

ende akkurat i overgangen. Dersom denne settes til en verdi mindre enn null kommer et område rundt overgangen til å være helt stille (amplitude 0).

Skaleringsfaktoren må være større enn '0'. Dersom skaleringsfaktoren er '0' blir ikke resultatet en fading, men et stille område i hele det området som skulle vært brukt til fading. Lyden blir med andre ord kuttet tvert av ved utfading og kommer momentant på topp amplitude ved innfading. Dersom større verdi enn '1' velges kommer ikke lyden til å være fadet ordentlig ut til null amplitude før den nye kommer inn, og den nye kommer til å starte tilsvarende over null amplitude.

Med en verdi på '0,9' som er valgt som standard, kommer 10 % av total fadetid rundt klippet til å være stille. Det har ved tester vist seg å være svært fornuftig å legge inn et lite stille område rundt klippet dersom hvert klipp skal fades ordentlig ut før det nye skal fades inn. Dersom det ikke er ønskelig med en ordentlig utfading rundt klippet velges det en verdi som ligger over '1'. Da vil nullpunktet i fadingen ligge i et område som ikke blir med i resultatstrømmen.



Figur 43: Fading av audio i overgangen mellom to klipp.

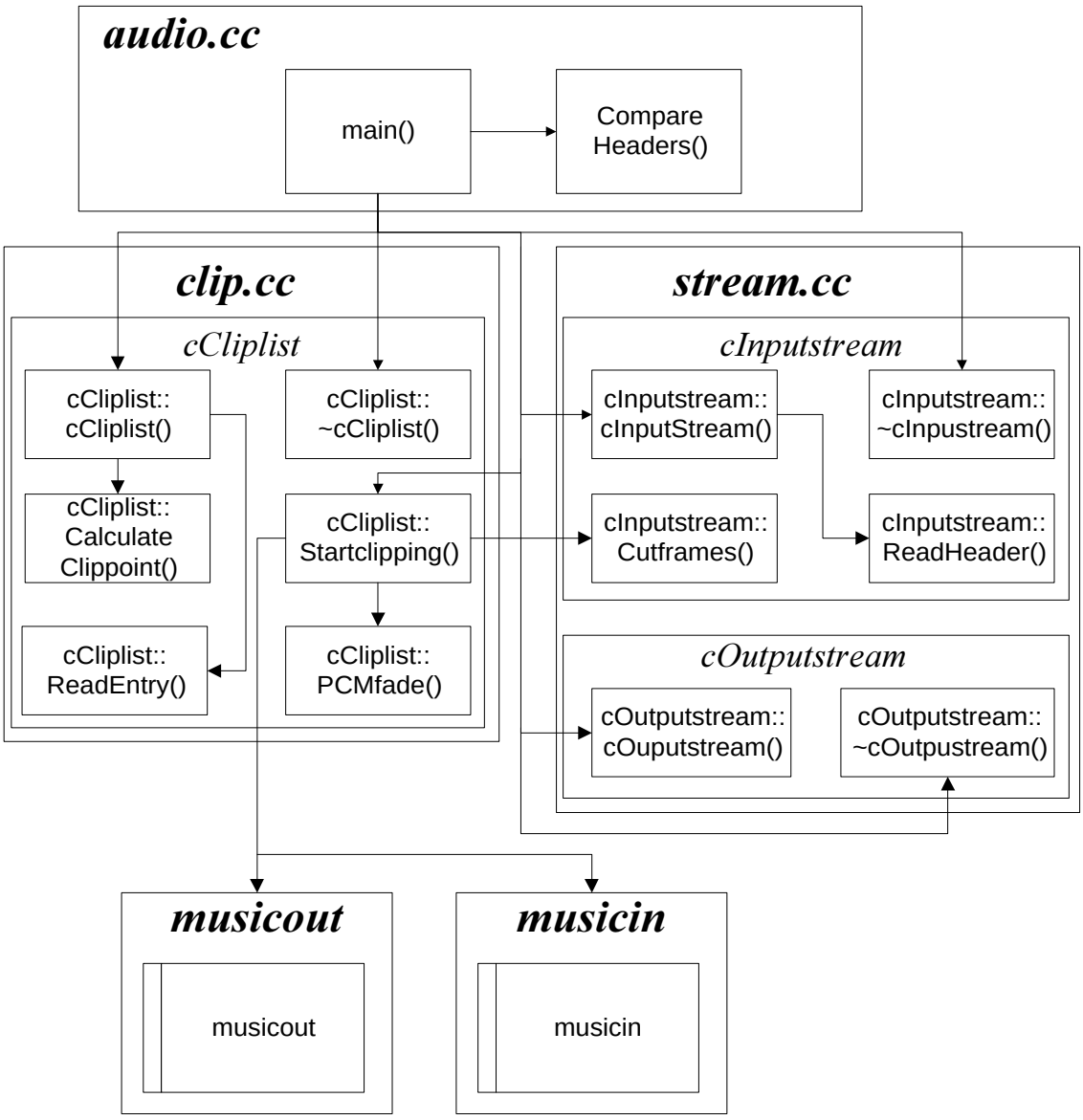
8.5 Systemmodulene

Audiosystemet er delt inn i tre moduler. *Audio* håndterer oppstart og initialisering, *Clip* håndterer klipplisten og *Stream* håndterer input- og outputstrømmene. Modulene og samspillet mellom disse er illustrert i Figur 44.

Programmet er implementert i *C++*. Programmet er nesten i sin helhet objektorientert. Løse funksjoner forekommer bare hvor det ikke falt naturlig å putte funksjonen inn i en klasse.

I alle returspesifikasjoner er suksess det samme som TRUE og feil det samme som FALSE. Disse to verdiene er definert i *audio.h.*

Alle klassenavnene har liten *c*, og strukturer har liten *s* som første bokstav etterfulgt av et navn som begynner med stor bokstav. Klassens innhold kan forstås utfra dette navnet.



Tegnforklaring:

Figur 44: Kontrollflyt for audioeditoren.

8.5.1 Audiomodulen

Denne modulen inneholder rutiner for oppstart og initialisering. Den oppretter objekter for alle inputstrømmene, outputstrømmen og for klipplista.

*int main(int argc, char **argv)*

Denne funksjonen er oppstartfunksjonen. Den starter med å sjekker at antall argumenter er korrekt. Det skal være minst tre argumenter: en eller flere inputstrømmer, en outputstrøm og en klipplistefil.

Dersom antall argumenter stemte opprettes det objekter for alle inputstrømmene og for klipplista. Klipplista leses inn ved opprettelse. MPEG-1 audioparametrene for inputstrømmen blir sjekket ved hjelp av *CompareHeaders()*. Dersom parametrene er inkonsistente, avslutter programmet.

Deretter startes klippinga med funksjonen *Startclipping()* i Clipmodulen. Alle klippene blir satt sammen til ett når outputstrømmen blir opprettet.

Før programmet terminerer slettes alle objektene.

*int CompareHeaders(cInputstream *stream1, cInputstream *stream2)*

Compareheaders sammenligner allerede innlest headerinformasjon tilhørende *stream1* og *stream2*.

Opplysningene som blir sjekket er lag, bitrate, samplingsfrekvens, lydmodus, betoning (emphasis) og måten joint stereo er kodet på. Dersom ett eller flere av parametrene ikke stemmer overens, blir navnet på alle de inkonsistente parametrene skrevet ut og funksjonen returnerer feil.

Kalles av *main()*.

8.5.2 Streammodulen

Streammodulen håndterer oppretting, sletting og operasjoner på input- og outputstrømmer.

8.5.2.1 cInputstream

Denne klassen er for alle inputstrømmene. Den inneholder alle data om én enkelt strøm og kan utføre operasjoner på denne. Filhåndtaket er privat slik at funksjoner som ikke er medlem ikke kan manipulere strømmen direkte.

Klassen inneholder variable for lag, bitrate, samplingsfrekvens, lydmodus, hvilken måte joint stereo er kodet på og betoning (emphasis).

*cInputstream::cInputstream(const char *filename)*

Denne funksjonen er *public*.

Denne funksjonen er klassens konstruktor og blir derfor automatisk kalt opp ved opprettelse av et objekt av denne typen. Filnavnet må sendes med i oppkallet som argument for at opprettelsen skal bli godtatt.

Funksjonen åpner fila med filnavnet *filename*. Dersom fila ikke finnes eller noe annet har ført til at fila ikke kunne åpnes, terminerer programmet med en feilmelding.

Til slutt kalles medlemsfunksjonen *ReadHeader()* for å lese inn strømmens header. Dersom denne funksjonen returnerer suksess, returnerer funksjonen. I motsatt fall terminerer programmet med en feilmelding.

Kalles (indirekte) av *int main()* i *audiomodulen.*

cInputstream::~cInputstream()

Denne funksjonen er *public*.

Denne funksjonen er klassens destruktor og blir derfor automatisk kalt opp når objektet slettes.

Funksjonen lukker fila som *filestream* peker på.

Kalles (indirekte) av *int main()* i *audiomodulen.*

int cInputstream::ReadHeader()

Denne funksjonen er *private*.

Denne funksjonen leser inn headeren til inpustrømmen.

Det første som blir gjort er å undersøke at det ikke er en fil uten innhold ved å lese den. Dersom lesingen gikk greit, parses headeren for parametre. Headeren blir sjekket for lag, og godtar bare lag 1 og 2. De tilhørende variable for lag blir satt.

Dersom det er en lag 3 audiostrøm returnerer funksjonen med feil, og skriver ut en feilmelding om at lag 3 ikke er støttet.

Deretter blir resten av headeren parset og bitverdiene for de forskjellige parametrene konvertert til verdier som bl.a. “musicin” kan forstå. Dette gjelder bitrate, samplingsfrekvens, lydmodus, hvilken måte joint stereo er kodet på og betoning (emphasis). Alle de konverterte verdiene blir lagt tilbake i sine respektive variable.

Ved parsingen av samplingsfrekvensen blir også rammelengden i millisekunder satt. Som forklart i 3.3 og Tabell 7 avhenger denne av lag og samplingsfrekvens. Ved parsingen av bitrate må det også tas med i betraktningen hvilket lag det er. De samme bitverdiene representerer forskjellige bitrater avhengig av hvilket lag det er.

Dersom det gikk greit å lese headeren returnerer funksjonen suksess.

Kalles av medlemsfunksjonen *cInputstream()* (konstruktoren).

*cInputstream::Cutframes(int firstframe, int lastframe, FILE *outfile)*

Denne funksjonen er *public*.

Denne funksjonen kutter audiostrømmen mellom *firstframe* og *lastframe* og legger innholdet i *outfile*. Filene må være åpnet før funksjonen kalles.

Funksjonen deklarerer statiske variable for rammenummer og buffer. Derfor trenger ikke funksjonen å starte fra begynnelsen av strømmen dersom *firstframe* ligger etter den stående rammen. Årsaken til dette er at statiske variable beholder sin verdi fra oppkall til oppkall.

Dersom *firstframe* ligger etter stående ramme beholdes bufferen og stående rammenummer, i motsatt fall slettes begge disse og filpeker settes til starten av fila.

Deretter leses en og en byte inn med kontroll av verdien av de siste to byte mellom hver innlesning. For dette benyttes bitskifting internt i bufferen. Dersom verdien er *syncword* (0xfff?) økes stående rammeteller med 1 og sjekkes mot *firstframe*. Dersom verdien er *firstframe*, skrives bufferens innhold til *outfile*.

Deretter søkes det etter *lastframe* ved bruk av samme metode som søket etter *firstframe*. Alle innleste bytes skrives til *outfile* inntil rammeteller en har nådd samme verdi som *lastframe*.

Dersom slutten av fila ble nådd før enten *firstframe* eller *lastframe*, skrives det ut en feilmelding og programmet terminerer.

Kalles av *cCliplist::Startclipping()*.

8.5.2.2 cOutputstream

Denne klassen håndterer sammensettingen av klippene til et ferdig sluttprodukt.

cOutputstream::cOutputstream(int clipno, const char filename)*

Denne funksjonen er *public*.

Denne funksjonen er klassens konstruktor og kalle opp automatisk ved opprettelse av et objekt av denne typen. Konstruktoren tar som argument antall klipp som skal settes sammen (*clipno*) og filnavnet til utfila (*filename*).

Funksjonen sjekker først om det er mulig å åpne utfila. Programmet terminerer med en feilmelding dersom dette ikke gikk.

Deretter åpnes filene for hvert klipp fra *0* til *clipno-1* og kopieres over til utfila i riktig rekkefølge. Disse file har navn etter klippnummer: *clip_*.mpa*, hvor * erstatter klippnummer.

Kalles (indirekte) av *int main()* i *audiomodulen.*

cOutputstream::~cOutputstream()

Denne funksjonen er *public*.

Denne funksjonen er klassens destruktor og kalles automatisk når et objekt av denne typen slettes.

Funksjonen lukker utfila.

Kalles (indirekte) av *int main()* i *audiomodulen.*

8.5.3 Clipmodulen

Clipmodulen håndterer alt som har med klipplista å gjøre. Den inneholder også funksjoner for oppstart av klippinga og fading av PCM rådatafila.

8.5.3.1 cCliplist

Denne klassen er klassen for klipplista. Den inneholder en peker til klipplista som er av typen *sClippoint*. Denne er en lenket liste med alle klippunkt. Klassen sørger for innlesing av klipplista, oppstart av klippinga og kalkulering av klippunkt og rammer som trengs til fading.

Klassen inneholder også variable for antall klipp (*public*), synkroniseringsfeil (*private*) og filpeker til klipplistefila (*private*).

*cCliplist::cCliplist(char *filename, int numstreams)*

Denne funksjonen er *public*.

Denne funksjonen er klassens konstruktor og kalles opp automatisk når et objekt av denne typen opprettes. Den tar som argument filnavnet til klipplista og antall strømmer som skal redigeres.

Funksjonen forsøker først å åpne klipplistefila. Programmet terminerer med en feilmelding dersom det ikke gikk å åpne fila.

I motsatt fall kalles medlemsfunksjonene *ReadEntry()* og *CalculateClippoint()* opp for hver linje inntil fila er slutt. Funksjonen foretar en sjekk på om strømnumrene som ligger i klipplistefila ikke er høyere enn antall strømmer som ble oppgitt som argumenter til programmet. I tilfelle det ligger et for høyt strømnummer i klippfila, skrives det ut en feilmelding til skjermen.

Klipplisteinnslagene puttes inn i en lenket liste av type *sClippoint*. Funksjonen *ReadEntry()* kalles opp inntil denne returnerer feil. Da settes nestepekeren til det siste objektet i den lenkede listen til NULL. Funksjonen sørger for å inkrementere telleren for antall klipp for hvert vellykket oppkall av *ReadEntry()*.

Kalles (indirekte) av *int main()* i *audiomodulen.*

cCliplist::~cCliplist()

Denne funksjonen er *public*.

Denne funksjonen er klassens destruktor og kalles opp automatisk når et objekt av denne typen slettes.

Funksjonen sørger for å slette alle midlertidige filer som ble opprettet under klippingen. Det dreier seg totalt om seks filer for hvert klipp: innfading (PCM og MPEG-1 audio), urørte rammer (MPEG-1 audio), utfading (PCM og MPEG-1 audio) og hele klippet (sammensetting av innfading, urørte rammer og utfading – MPEG-1 audio). Deretter lukkes klipplistefila.

Kalles (indirekte) av *int main()* i *audiomodulen.*

*int cCliplist::CalculateClippoint(sClippoint *clippoint)*

Denne funksjonen er *public*.

Denne funksjonen kalkulerer klippunktet som ligger nærmest det som ligger oppgitt i klipplista. Funksjonen sørger for å ta med tidligere akkumulert (oppsamlet) synkroniseringsfeil (på grunn av den faste rammelengden) i beregningen.

Alle tidsangivelser regnes om til millisekunder (multipliseres med 1000). Deretter regnes det ut hvilken ramme som skal være klippets første ramme. Tidligere akkumulert synkroniseringsfeil taes med i beregningen av denne rammen. Etter at startrammen er beregnet, regnes den nye akkumulerte synkroniseringsfeilen ut.

Synkroniseringsfeilen tilstrebes alltid å ligge innenfor kravene oppgitt i 4.3.5 (-1/3 +2/3 av rammelengden).

De samme beregningene gjentar seg for siste ramme i klippet. Både nummeret til start- og sluttrammen legges inn i datastrukturen til klippet.

Siste punktet denne funksjonen gjør, er å beregne hvor mange rammer som trengs for å fade med den oppgitte tiden. Funksjonen sørger for å avsløre fadetider som overskrider klippets totale lengde. Et klipp på 10 sekunder kan f.eks. ikke ha en fadetid på over 5 sekunder fordi klippet skal fades med 5 sekunder i starten og i 5 sekunder på slutten. En valgt fadetid på f.eks. 5,5 sekunder vil i tilfelle gi en total

fadetid (inn+ut) på 11 sekunder. I slike tilfeller skriver programmet ut en feilmelding om dette.

Dersom alle beregninger gikk bra, returnerer funksjonen suksess.

Kalles av medlemsfunksjonen *cCliplist()* (konstruktoren).

*int cCliplist::ReadEntry(sClippoint *clippoint)*

Denne funksjonen er *private*.

Denne funksjonen leser inn ett innslag fra klippfila som filpekeren *clipfile* peker på, og legger dataene i objektet som *clippoint* peker på. Dersom lesinga gikk greit returneres suksess.

Innslagene som leses inn fra klipplistefila er strømnummer (i rekkefølge som angitt i argumenter til programmet), starttidspunkt, sluttidspunkt og fadetid (alle tidspunkter i sekunder med millisekunders nøyaktighet). Funksjonen leter etter tabulator for å skille mellom verdiene i klippfila.

Kalles av medlemsfunksjonen *cCliplist()* (konstruktoren).

*cCliplist::Startclipping(cInputstream **inputstream)*

Denne funksjonen er *public*.

Denne funksjonen styrer selve klippeprosessen. Den sørger for å kalle opp andre funksjoner i rett rekkefølge og med rette parametre. Funksjonen tar peker til tabellen med inputstrømmene som argument. Dette burde ikke innebære noen fare for lesing utenfor minneområdet fordi det allerede er foretatt sjekk på at ingen av klippinnslagene refererer til strømnummer som ikke eksisterer.

Funksjonen oppretter midlertidige filer for innfadingsrammer, urørte rammer, utfadingsrammer og for hele det sammensatte klippet. Deretter regnes det ut start- og sluttrammenummer for inn- og utfading og for den delen av klippet som ikke skal dekodes. For alle disse rammeintervallene kalles *Cutframes()* for det inputstreamobjektet det jobbes på. *Cutframes()* skriver rammeintervallet til den filpekeren som ble sendt som argument.

De rammeintervallene som skal fades må dekodes. Derfor konstrueres det en kommandolinje for dekoding ved hjelp av “musicout”. Denne kommandolinja blir brukt i oppkallet til *system()*, som er en global systemfunksjon for å utføre en *shell*-kommandolinje. Den dekodede audiofila har PCM rådataformat. Denne fila blir sendt til medlemsfunksjonen *PCMfade()* sammen med en utregning av hvor mange samplinger fadingen skal gå over (fadelengde * antall samples per ramme) og om det er inn- eller utfading.

Etter at fadingen er utført på PCM-fila, må fila kodes igjen. Derfor konstrueres det en kommandolinje for koding ved hjelp av “musicin”. Denne linja blir i likhet med den for “musicout”, brukt i oppkallet til *system()*. Denne kommandolinja benytter seg av de konverterte variablene for bitrate, lag, lydmodus, samplingshastighet og betoning (emphasis) som ligger i objektet det jobbes mot. Disse parametrene ble lagt inn av *cInputstream::ReadHeader()* da objektet ble opprettet.

Begge de to siste avsnittene blir gjentatt både for rammene som inneholder innfadingssekvensen og for rammene som inneholder utfadingssekvensen.

Til slutt blir alle de midlertidige filene kopiert over til én fil for hele klippet. Denne filen starter med innfadingssekvensen, fortsetter med de urørte rammene og slutter

med utfadingssekvensen. Dette blir gjort for å lette sammensettingen av alle klippene til slutt.

Alle punktene gjentas for hvert klippunkt i klipplista.

Kalles av *int main()* i *audiomodulen.*

*int cCliplist::PCMfade(const char *filename, const char *fadetype, long fadesamples)*

Denne funksjonen er *private*.

Denne funksjonen foretar selve fadingen på PCM rådatafilene. Den kan foreta både inn- og utfading avhengig av verdien på *fadetype* (*INN* for inn- og *OUT* for utfading). Andre parametre funksjonen tar, er filnavnet på PCM rådatafila og antall samplinger fadingen skal foretaes over. Det kan også defineres en skaleringsfaktor for denne funksjonen. Denne forteller funksjonen hvor stor andel av total fadetid som skal benyttes til fading. Resten blir satt til 0 (stillhet). Hvorfor denne skaleringsfaktoren er innført og hvordan den virker er forklart i 8.4.2.

Funksjonen starter med og åpne PCM-fila. I tillegg blir det opprettet en midlertidig fil som skal brukes til å legge den ferdige PCM-strømmen i. Deretter foretas det en sjekk på om det skal utføres en inn- eller utfading. Avhengig av dette resultatet, antall samplinger fadingen skal utføres over og skaleringsfaktoren beregnes det en funksjon. Denne funksjonen er kontinuerlig, monotont stigende (innfading) eller avtagende (utfading) og skal multipliseres med samplingene for å skape en tilsvarende effekt på lydnivået.

Funksjonene som brukes er

```
multiplikator = 1-counter/(fadesamples*scalefactor)
```

for utfading og

```
multiplikator = (counter-hold)/(fadesamples*scalefactor)
```

for innfading. De samplede verdiene multipliseres med *multiplikator* og skrives til den midlertidige fila. *counter* er en tellevariabel som inkrementeres med 1 for hver sampling. *fadesamples* er antalle samplinger fadingen skal foretaes over. *hold* er en utregning av når signalet skal begynne å stige fra 0. Denne er basert på skaleringsfaktoren (*scalefactor*):

```
hold = fadesamples*(1-scalefactor)
```

Funksjonen starter innfadingen når *counter* har nådd samme verdi som *hold.*

Ved innfading blir det foretatt en kontinuerlig sjekk på at *multiplikator* ikke overstiger 1. Dette ville i såfall gi en økning i lydstyrke i forhold til det opprinnelige signalet. Ved utfading blir det foretatt en tilsvarende sjekk på at *multiplikator* ikke synker under 0. Dette ville føre til en utfading og deretter en innfading i motsatt fase fordi signalet etterhvert blir multiplisert med et negativt tall slik at samplingsverdiene bytter fortegn.

Lyd er som regel *signed*; dvs. med fortegn avhengig av om amplituden er på den ene eller andre siden av offset (nullpunktet). Ved en multiplikasjon med et negativt tall ville amplituden flyttes på motsatt side av offset, og gi en faseforskjell på 180° i forhold til det opprinnelige signalet. For øret vårt ville det ikke bety noen ting fordi vi ikke klarer å høre faseforskjeller med mindre vi hører begge signalene samtidig. Med andre ord vil det bare høres ut som en utfading raskt etterfulgt av en innfading.

Til slutt lukker funksjonen begge filene og overskriver den opprinnelige PCM-fila med den midlertidige fila, slik at denne nå blir den gjeldende PCM-fila.

Kalles av medlemsfunksjonen *Startclipping()*.

8.5.3.2 sClippoint

For å ta vare på all informasjon som vedgår *ett* bestemt klipp (*ett* innslag i klipplistefila), er det deklarert en struktur (*struct*). Denne strukturen inneholder informasjon om start- og sluttidspunktet til klippet, fadetid, strømnummer (i rekkefølge som angitt ved oppstart av programmet), start- og sluttramme (beregnet utfra start- og sluttidspunktet) og antall rammer som trengs for å fade (beregnet utfra fadetid). Strukturen inneholder også en nestepeker som peker på neste element i lista.

Benyttes av klassen *cCliplist()*.

8.6 Kjøretidsinformasjon

Dersom programmet kompileres med *-D_DEBUG* som ett av argumentene til *g++*, skrives det ut informasjon om nesten alt programmet foretar seg. Med optimalisert kompilering, kompileres programmet med utskrifter til skjerm som vil være interessante for en vanlig bruker av systemet. Informasjonen som skrives ut da er hvilket lag lyden er kodet i, rammelengden og når de forskjellige start- og sluttrammer for hvert klipp er funnet.

De to programmene som står for kodingen ("musicin") og dekodingen ("musicout"), skriver ut ekstensiv informasjon om hva de foretar seg. Ved dekoding skriver "musicout" ut noe slikt:

```
Input file = 'fadepre_000.mpa'  output file = 'fadepre_000.pcm'
the bit stream file fadepre_000.mpa is a BINARY file
using bit allocation table alloc_0
HDR:  s=FFF, id=1, l=2, ep=1, br=4, sf=0, pd=1, pr=0, m=3, js=0, c=0, o=0, e=0
layer=II, tot bitrate=64, sfrq=44.1, mode=single-ch, sblim=27, jsbd=27, ch=1
<Her telles rammene i MPEG-1 audiostrømmen opp>
Frame cannot be located
Input stream may be empty
Avg slots/frame = 208.982; b/smp = 1.45; br = 64.001 kbps
Decoding of "fadepre_000.mpa" is finished
The decoded PCM output file name is "fadepre_000.pcm"
```

Programmet skriver ut MPEG-1 bitverdiene og de konverterte verdiene av de forskjellige parametrene som ligger i audiostrømmen. Deretter telles rammene opp etterhvert som de dekodes. Feilmeldingen *"Frame cannot be located"; "Input stream may be empty"* kan betraktes som en feil i programmet. Denne meldinga kommer fordi programmet ikke finner en ny rammestart (*syncword*). Denne meldinga signaliserer altså ikke at det er noe i veien med MPEG-1 audiostrømmen.

Til slutt skrives resultatet av kodinga ut, slik som bit/sample.

Ved koding skriver "musicin" ut noe slikt:

```
Encoding configuration:
Layer=II   mode=single-ch   extn=0   psy model=2
samp frq=44.1 kHz   total bitrate=64 kbps
de-emph=0   c/right=0   orig=0   errprot=0
input file: 'fadepre_000.pcm'   output file: 'fadepre_000.mpa'
using bit allocation table alloc_0
slots/frame = 208
frac SpF=0.980, tot bitrate=64 kbps, s freq=44.1 kHz
Fractional number of slots, padding required
{   0}{   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}{  29}{  30}{  31}{  32}{  33}{  34}{  35}{  36}{  37}{  38}
{  39}{  40}{  41}{  42}{  43}{  44}{  45}{  46}{  47}{  48}{  49}{  50}{  51}
{  52}{  53}{  54}{  55}{  56}{  57}{  58}{  59}{  60}{  61}{  62}{  63}{  64}
{  65}{  66}{  67}{  68}{  69}{  70}{  71}{  72}{  73}{  74}{  75}{  76}{  77}
```

```
{  78}{  79}{  80}{  81}{  82}{  83}{  84}{  85}{  86}{  87}{  88}{  89}{  90}
{  91}{  92}{  93}{  94}{  95}{  96}{  97}{  98}{  99}{ 100}{ 101}{ 102}{ 103}
{ 104}{ 105}{ 106}{ 107}{ 108}{ 109}{ 110}{ 111}{ 112}{ 113}{ 114}{ 115}{ 116}
{ 117}{ 118}{ 119}{ 120}{ 121}{ 122}{ 123}{ 124}{ 125}{ 126}{ 127}{ 128}{ 129}
{ 130}{ 131}{ 132}{ 133}{ 134}{ 135}{ 136}{ 137}{ 138}{ 139}{ 140}{ 141}{ 142}
{ 143}{ 144}{ 145}{ 146}{ 147}{ 148}{ 149}{ 150}{ 151}{ 152}{ 153}{ 154}{ 155}
{ 156}{ 157}{ 158}{ 159}{ 160}{ 161}{ 162}{ 163}{ 164}{ 165}{ 166}{ 167}{ 168}
{ 169}{ 170}{ 171}{ 172}{ 173}{ 174}{ 175}{ 176}{ 177}{ 178}{ 179}{ 180}{ 181}
{ 182}{ 183}{ 184}{ 185}{ 186}{ 187}{ 188}{ 189}{ 190}{ 191}{ 192}{ 193}{ 194}
{ 195}{ 196}{ 197}{ 198}Hit end of audio data
Avg slots/frame = 208.980; b/smp = 1.45; br = 64.000 kbps
Encoding of "fadepre_000.pcm" with psychoacoustic model 2 is finished
The MPEG encoded output file name is "fadepre_000.mpa"
```

Programmet starter med å gjenta alle argumentene. Deretter telles rammene opp etter hverandre etter hvert som de kodes. Programmet signaliserer at det har møtt slutten på PCM-fila med meldinga *"Hit end of audio data"*. Til slutt skrives resultatet av kodinga ut, slik som bit/sample.

8.6.1 Utskrifter ved debugkompilering

Dersom programmet er kompilert med debugdefinisjonen satt, starter programmet med å skrive ut headeren til alle strømmen ut. Deretter skrives den utregnede rammelengden ut.

Etter at alle inputstrømmene er åpnet, leses klipplista inn. Alle innslagene skrives ut etter hvert som de leses inn. Etter utskrift av hvert innslag i klipplista, skrives også utregningene av startramme, stopramme, faderammer (antall rammer som trengs for å fade i oppgitt tid) og akkumulert synkfeil (*out-of-sync*) etter dette klippet.

Når klipplista er ferdig innlest, starter redigeringen. Da skrives de forskjellige rammenenummrene ut etter hvert som de blir funnet i inputstrømmen som skal klippes. Etter at de forskjellige delene av et klipp er skrevet til fil, skrives den konstruerte kommandolinja for dekoding ut. Når fading er utført på PCM-fila, skrives den konstruerte kommandolinja for koding ut. Disse punktene gjentar seg for hvert klipp i klipplista.

Eksempel på utskrift etter debugkompilering:

```
buffer=fffd
Layer II, buffer=42c0
framelength: 26.1224489796 ms
Bitrate:64*Smplfrq:32000*Sndmode:m*Modeext:00*Emphasis:n
i=0;inputstream: 804f710
buffer=fffd
Layer II, buffer=42c0
framelength: 26.1224489796 ms
Bitrate:64*Smplfrq:32000*Sndmode:m*Modeext:00*Emphasis:n
i=1;inputstream: 804f728
Streamno:0*Start:1.234000*Stop:13.434000*Fadetime:5.200000
Outofsync:6.244898 ms*Startframe:47
Outofsync:-0.816327 ms*Stopframe:514*Fadeframes:200
Streamno:1*Start:3.304000*Stop:20.344000*Fadetime:4.450000
Outofsync:11.755102 ms*Startframe:126
Outofsync:17.142857 ms*Stopframe:779*Fadeframes:171
Streamno:0*Start:15.789000*Stop:35.140000*Fadetime:8.754000
Outofsync:2.061224 ms*Startframe:605
Outofsync:-3.244898 ms*Stopframe:1345*Fadeframes:336
Streamno:1*Start:14.754000*Stop:25.400000*Fadetime:3.100000
Outofsync:-8.428571 ms*Startframe:565
Outofsync:-17.408163 ms*Stopframe:972*Fadeframes:119
Streamno:1*Start:0.000000*Stop:10.000000*Fadetime:2.000000
Outofsync:-17.408163 ms*Startframe:0
Outofsync:-12.510204 ms*Stopframe:383*Fadeframes:77
Funnet:47 Funnet:247
Funnet:247 Funnet:314
Funnet:314 Funnet:514
musicout fadepre_000.mpa fadepre_000.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepre_000.pcm fadepre_000.mpa
```

```
musicout fadepost_000.mpa fadepost_000.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepost_000.pcm fadepost_000.mpa
Funnet:126 Funnet:297
Funnet:297 Funnet:608
Funnet:608 Funnet:779
musicout fadepre_001.mpa fadepre_001.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepre_001.pcm fadepre_001.mpa
musicout fadepost_001.mpa fadepost_001.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepost_001.pcm fadepost_001.mpa
Funnet:605 Funnet:941
Funnet:941 Funnet:1009
Funnet:1009 Funnet:1345
musicout fadepre_002.mpa fadepre_002.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepre_002.pcm fadepre_002.mpa
musicout fadepost_002.mpa fadepost_002.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepost_002.pcm fadepost_002.mpa
Funnet:565 Funnet:684
Funnet:684 Funnet:853
Funnet:853 Funnet:972
musicout fadepre_003.mpa fadepre_003.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepre_003.pcm fadepre_003.mpa
musicout fadepost_003.mpa fadepost_003.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepost_003.pcm fadepost_003.mpa
Funnet:0 Funnet:77
Funnet:77 Funnet:306
Funnet:306 Funnet:383
musicout fadepre_004.mpa fadepre_004.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepre_004.pcm fadepre_004.mpa
musicout fadepost_004.mpa fadepost_004.pcm
musicin -l 2 -m m -p 2 -s 44100 -b 64 -d n fadepost_004.pcm fadepost_004.mpa
```

Denne informasjonen kommer i tillegg til den som «musicin» og «musicout» skriver ut.

For hver strøm blir de to første byte, hvilket lag lyden er kodet i, de to neste byte, utregnet rammelengde, bitrate, samplingsfrekvens, lydmodus, stereokodingsmetode, emphasis, strømnummer og adressen filpekeren inneholder. Disse opplysningene blir skrevet ut i nevnte rekkefølge.

Deretter skrives hvert innslag i klipplista med utregnet start- og sluttramme, faderammer og synkroniseringsfeil ut.

Kommandolinjene til hvert oppkall av «musicin» og «musicout» blir så skrevet ut.

9. EVALUERING

Evalueringens formål er å vurdere om resultatet er tilfredsstillende. Hvert kriterium blir vurdert for seg, og til slutt foretas en samlet vurdering av alle kriterier.

Videodelen er i likhet med det øvrige stoffet om video, basert på arbeid utført våren 1996. For kompletthet er denne beskrivelsen tatt med i denne rapporten.

9.1 Kriterier for evaluering

9.1.1 Bildekvalitet

Bildekvalitet er et høyst subjektivt kvalitetsmål. Komprimeringsrate kan si noe om kvaliteten, men er vanskelig å beregne. For å måle kvaliteten på bildet må det foretaes en inspeksjon av den editerte strømmen og kildestrømmene for å foreta en sammenligning. Ved sammenligningen må det taes i betraktning at det er umulig å opprettholde kvaliteten ved koding og dekoding av en ramme. Det som må inspiseres er med andre ord om kvalitetsreduksjonen er akseptabel.

Dersom det er stor kvalitetsforringelse, vil klippsonen skille seg ut fra resten av videosekvensen. Vi risikerer dermed å miste kontinuiteten.

9.1.2 Lydkvalitet

Lydkvalitet er et minst like subjektivt mål som bildekvalitet. I likhet med video spiller bitraten en vesentlig faktor for hvor godt bevart kvaliteten til det opprinnelige signalet er. Bitraten alene sier ikke så mye, derfor må det taes med i evalueringen hvilken psykoakustisk modell som er benyttet og hvilket lag lyden er kodet i. Et lydsignal med bitrate på 32 kbit/s i lag 1 med psykoakustisk modell 1 har i utgangspunktet dårligere kvalitet enn det samme lydsignalet kodet med samme bitrate, psykoakustisk modell 2 og lag 2.

I klippsonen forventes det at lyden blir litt mer diffus og mister noen av de største variasjonene. For sikkerhets skyld blir all lyd som kodes, kodet med psykoakustisk modell 2. Dette vil innføre mindre tap ved den valgte bitraten enn psykoakustisk modell 1 ville gjort.

9.1.3 CPU-forbruk

For å evaluere CPU-forbruk kan programmet *'time'* benyttes. Dette programmet skriver ut hvor mye tid som er brukt, hvor mye effektiv *'user'* og *'system'* CPU-tid som er brukt og prosentbelastningen på CPU under programutførelsen.

Det er viktigst å holde CPU-belastningen nede på flerbrukersystem. Dersom ett program bruker for mye CPU, får ikke andre programmer kjøre.

9.1.4 Bitrate

Måling av datarate er en tredje evalueringsmulighet. For video blir denne målt for hver gruppe. Den måles ved å beregne posisjon i strømmen (filposisjon) og deler på hvor mange videorammer som er i gruppa. Vi får da et tall for bitraten per videoramme i alle gruppene. Dette tallet vil naturligvis bli større desto færre videorammer det er i en gruppe fordi alle gruppene starter med en I- og P-ramme.

For audio er ikke dette relevant fordi bitraten holdes konstant gjennom hele audiostrømmen.

Et hovedpoeng ved overføring i nettverk er at bitraten ikke skal gå (vesentlig) opp i klippunktet.

9.1.5 Synkronisering mellom audio og video

Siste evalueringsmulighet er å foreta en sjekk på hvor godt synkronisert audio og video er. Denne kan enten basere seg på den oppgitte synkroniseringsfeilen fra audioeditoren, eller ved en subjektiv vurdering. En subjektiv vurdering er svært vanskelig å gjennomføre.

Synkronisering er viktig for at ikke den som ser på plages av at lyden ikke er synkron med video. Det vil da kanskje oppfattes som om lyden ikke hører til den videostrømmen som blir avspilt. For synkroniseringsevaluering blir det brukt resultater fra [LEYTEU91].

9.2 Evaluering

Hver del (system, video og audio) blir evaluert hver for seg. Hvert punkt i kriteriene blir behandlet for alle tre delene.

9.2.1 System

Under evaluering av system blir programsystemets toppnivå evaluert. Det blir bare simulert at en redigering av video og audio har foregått. Det som testes er med andre ord bare oppsplitting av MPEG-1 system, oppsplitting av klipplista og sammensetting av MPEG-1 audio og MPEG-1 video til MPEG-1 system.

Følgende klippliste ble benyttet:

```
[video]
1    29      80
0    65      121
1    45      86
[audio]
0    1.234   13.434  5.2
1    3.304   20.344  4.45
0    15.789  35.14   8.754
1    14.754  25.4    3.1
1    0       10      2
```

Videoene som ble benyttet er hentet fra programmet «Puck» som går på NRK 2. Disse strømmene har bitrate på 300 kbit/s og er ca. 7 minutter lange med en videorammehastighet på 25 rammer/s.

9.2.1.1 Bildekvalitet

Evaluering av bildekvalitet på dette nivået er ikke relevant. Det henvises til evaluering av video.

9.2.1.2 Lydkvalitet

Evaluering av lydkvalitet på dette nivået er ikke relevant. Det henvises til evaluering av audio.

9.2.1.3 CPU-forbruk

CPU-forbruket ble målt 06/04/97 mellom klokken 14:30 og 14:40 på *staude.idt.ntnu.no.* Det var ingen andre brukere innlogget på det angjeldende tidspunkt. Målingen ble gjennomført uten oppstart av audio- og videoeditoren. Det er med andre ord ytelsen til systemnivået alene som er målt. Oppstart og gjennomføring av demultipleksing og multipleksing er tatt med i målingene. Programmet og alle filene programmet benytter var lagt på et lokalt filsystem.

Programmet ble kjørt med normal prioritet. De presenterte tallene er gjennomsnittet etter tre gjennomkjøringer. Avvik i begge retninger er presentert i parenteser.

Følgende data ble registrert:

User tid:	30,77 sekund	(+0,34/-0,37)
System tid:	16,28 sekund	(+0,05/-0,06)
Medgått tid:	252,9 sekund	(+2,1/-1,1)
CPU-belastning:	18 %	(+0/-0)

Demultipleksingen og multipleksingen av video og audio til system tar svært lang tid, men er ikke CPU-intensivt. Programmet vil med andre ord ikke legge beslag på datakraften i nevneverdig grad. Det er lite avvik i tidsmålingene og CPU-belastningen.

9.2.1.4 Bitrate

Denne ansees ikke som relevant her. Det er bare videoeditoren som eventuelt kjører opp bitraten. Det henvises derfor til evalueringen av videoeditoren.

9.2.1.5 Synkronisering mellom audio og video

Etter en gjennomkjøring av audioeditoren ble følgende utregnede synkroniseringsfeil fanget opp:

Etter 1. klipp:	-0,816 ms
Etter 2. klipp:	17,142 ms
Etter 3. klipp:	-3,245 ms
Etter 4. klipp:	-17,408 ms

Alle disse tidene ligger innenfor kravene i [LEYTEU91]**.**

9.2.1.6 Samlet vurdering

Programmet har en svært tilfredsstillene CPU-belastning, men bruker altfor lang tid på multipleksing og demultipleksing. Fordi det er brukt programmer fra to forskjellige utviklingsmiljø må det antas at det ikke er store muligheter for forbedringer på dette området.

Det ble foretatt en sjekk på hva multipleksingen og demultipleksingen brukte av tid på en gjennomkjøring. Fordelingen ble målt slik:

Demultipleksing av to MPEG-1 systemstrømmer:	ca. 122 sekunder
Multipleksing til én MPEG-1 systemstrøm:	ca. 130 sekunder

Alle strømmene var like lange. Multipleksing er derfor mye mer tidkrevende enn demultipleksing.

Konklusjonen blir at programmet blir lite brukbart til tidskrevende formål. Ellers fungerer programmet bra, og genererer en systemstrøm som har synkronisering mellom audio- og videostrømmene som tilfredsstiller kravene i [LEYTEU91].

9.2.2 Video

I denne delen følger bare et kort utdrag av evalueringen fra [EDITER96]. For ytterligere dokumentasjon henvises det til dette dokumentet.

For å foreta en evaluering ble to MPEG-1 videostrømmer generert med utgangspunkt i samlingen av JPEG-video-samlingen ved IDT. Fil 0 (gruppestørrelse på 25 rammer) er åpningen på 18:30-nyhetene på TV 2 og fil 1 (gruppestørrelse på 13 rammer) er den gamle åpningsvignetten til «Dagsrevyen».

Følgende klippliste ble benyttet:

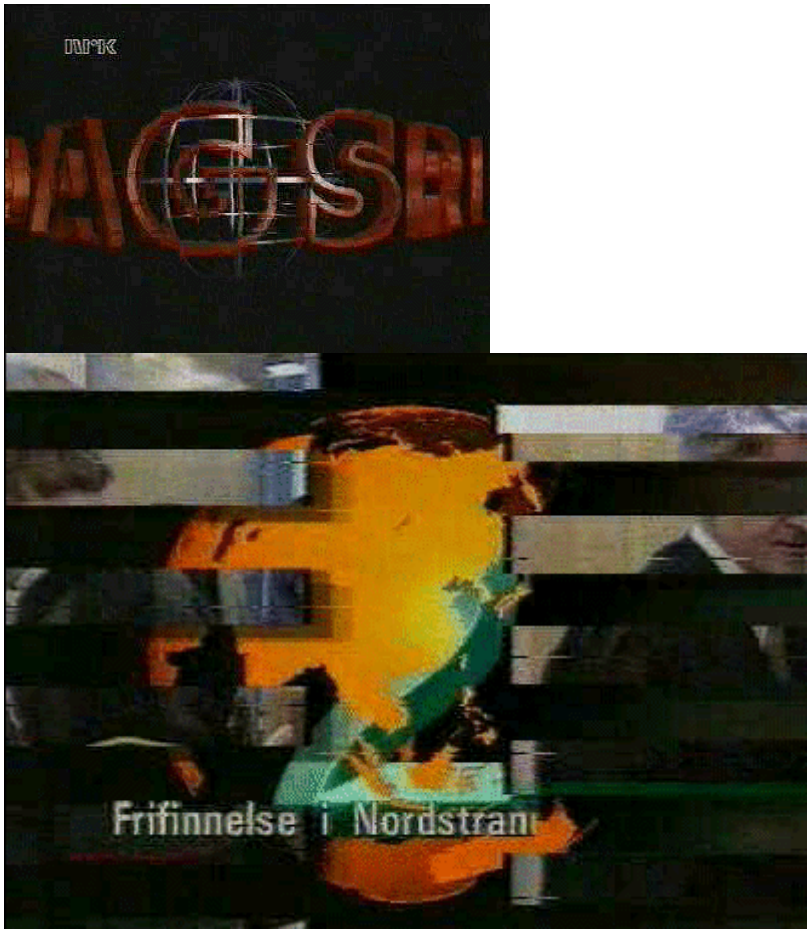
```
1    29      80
0    65      121
1    45      86
```

9.2.2.1 Bildekvalitet

Bildekvaliteten til de to klippene er ganske varierende. Fil 1 har i utgangspunktet dårligere kvalitet, sannsynligvis fordi det er eldre råmateriale.

Etter visuell vurdering, kan det fastslås at en viss degradering kan observeres. Dette gjelder akkurat som ventet, bare de rammene som er dekodet og kodet på nytt. Tidsrommet som denne degraderingen strekker seg over er i verste fall bare inntil to gruppers lengde, og oppfattes derfor ikke som noe stort problem.

Det er ikke mulig å gjengi endringen i dette dokumentet på grunn av den dårlige kvaliteten på sort/hvitt rastrert trykk.



Figur 45: Bildene på hver side av første editeringspunkt.

9.2.2.2 CPU-forbruk

Målingen av tids- og CPU-forbruk ble foretatt 26/04/96 kl. 0:34 på *staude.idi.unit.no* uten noen andre brukere innlogget på maskinen. Målingen ble foretatt med en gjennomkjøring hvor indeksfiler også måtte genereres. Programmet ble kjørt med normal prioritet. Alle data innbefatter også utføringen av «mpeg_encode».

Følgende data ble registrert:

User tid:	161,7 sekund
System tid:	10,6 sekund
Medgått tid:	217,9 sekund
CPU-belastning:	87 %

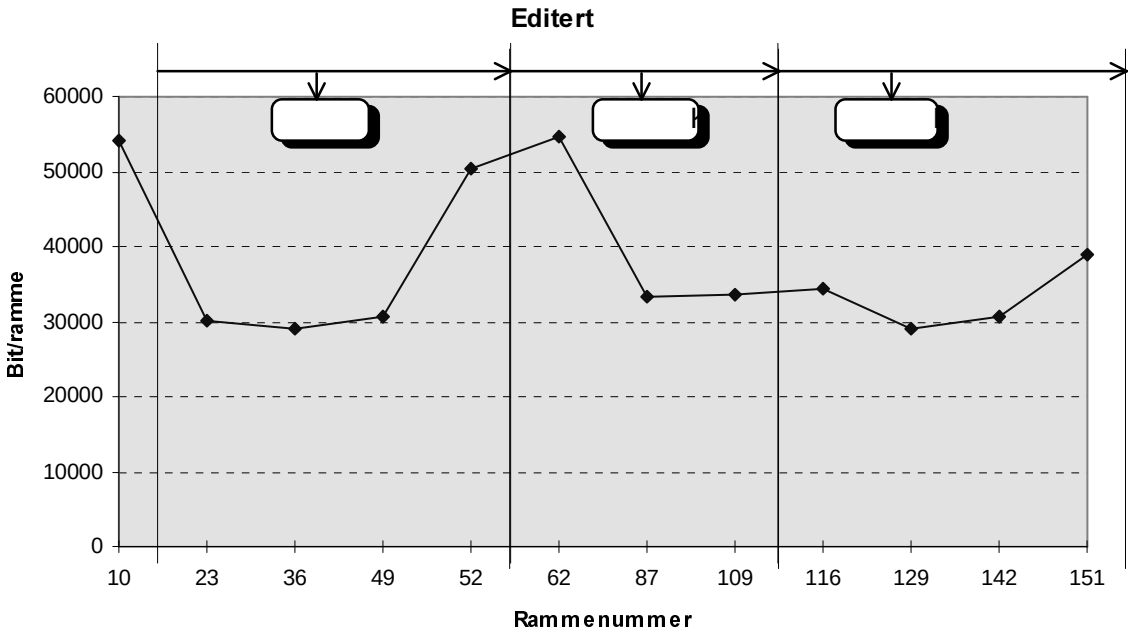
Den medgåtte tiden er slett ikke tilfredsstillende. 3,5 minutter for å editere noen korte klipp er for lang tid. I tillegg belastes CPU ganske mye. Det er definitivt mest å hente på å dekode kun de rammene som *må* dekodes. Skriving av rammer til disk er den delen av programmet som tar lengst tid, men dominerer mindre desto færre rammer som skrives.

9.2.2.3 Bitrate

Som det kan sees av resultatet av en editering i Figur 46, kan det påvises en relativt stor økning i bitrate i det første editeringspunktet (ramme 51). Årsaken er at gruppa

før editeringspunktet bare er på tre rammer, og to er referanserammer (I og P). Grunnen til at ingen økning kan registreres i det andre editeringspunktet (ramme 107) er at gruppen før er relativt stor (22 rammer) og at gruppen etter består stort sett av rammer med lite bevegelser.

Tatt i betraktning at en stor økning i bitratren var forventet, er resultatet som forventet, men ikke helt tilfredsstillende. Bitraten ligger i det første editeringspunktet på 54.659bit/ramme x 25rammer/sekund = 1.366.475bit/sekund som fremdeles ligger godt under kravene i standarden på 1,8 Mbit/sekund [ISO 11172].



Figur 46: Bitraten til den ferdig editerte filen.

9.2.2.4 Synkronisering mellom audio og video

Denne er ikke relevant her. Det henvises til evalueringen av systemeditoren.

9.2.2.5 Samlet vurdering

Programmet gir ønsket resultat: frie editeringspunkter og relativt lav bitrate med liten degradering av bildekvaliteten. Men tidsforbruket er for stort.

Ved testgjennomkjøringen fordeler tidsforbruket seg slik over fasene:

Generering av indeksfiler:	ca. 8 sekunder
Lesing frem til klipprammene:	ca. 42 sekunder
Skriving av rammefiler til disk:	ca. 72 sekunder
Koding av rammer («mpeg_encode»):	ca. 92 sekunder

Tidsforbruket kan reduseres ved å kun dekode de rammene som *må* dekodes. Dette vil spare mest tid dersom det dreier seg om lange videostrømmer.

Konklusjonen på dette blir at dersom tidsforbruket er en lite viktig faktor, er programmet bra nok.

9.2.3 Audiodelen

I audiodelen blir sammensetting av to MPEG-1 audiostrømmer med 5 klippunkt spredt rundt i strømmene evaluert. Begge strømmene kommer fra programmet «Puck» som går på NRK 2.

Disse MPEG-1 audiofilene er innspilt i lag 2 i mono med 32 kHz samplingshastighet og med 64 kbit/s datahastighet. Det er ikke mulig å si noe om hvilken psykoakustisk som er benyttet fordi denne ikke gjenspeiler seg på dekodersiden. Den er bare relevant for koderen i kodetidspunktet.

Klipplista som ble brukt ser slik ut:

```
0    1.234   13.434  5.2
1    3.304   20.344  4.45
0    15.789  35.14   8.754
1    14.754  25.4    3.1
1    0       10      2
```

9.2.3.1 Bildekvalitet

Dette punktet er ikke relevant for audiodelen.

9.2.3.2 Lydkvalitet

Endring av lydkvaliteten er bare aktuelt for den delen av audiostrømmene som må dekodes. Dette skjer bare akkurat rundt hvert klipp. Den delen som dekodes og kodes igjen er også gjenstand for enten innfading eller utfading. Det er av denne grunn ikke enkelt å høre om det skjer noen degradering av lydkvaliteten.

Jeg har ikke vært i stand til å detektere noen hørbar degradering av lyden. Dette kan skyldes akkurat det faktum at lyden blir fadet, eller at bitraten er på et akseptabelt høyt nivå. Jeg har også forsøkt å få uttalelser fra uavhengige personer, men ingen har vært i stand til å høre noen forskjell.

Til sammenligning kan det nevnes at bitraten til Dolby Digital (surround for kino) er på samme nivå (64 kbit/s), men her er lyden spilt inn med 48 kHz samplingsfrekvens og har 18 bit/sample oppløsning (MPEG-1 har 16 bit/sample).

I overgangen mellom klippene ble det oppdaget at dersom det ikke ble lagt inn en fadetid, førte det til at hørbart klikk akkurat når det nye klippet kom inn. Dette kommer av det amplitudehoppet som er nevnt tidligere. Dersom det egentlig ikke er ønskelig med en fading i overgangen mellom to klipp, bør det likevel legges inn omtrent et halvt sekund for å unngå denne effekten.

Det ble også oppdaget at lyden måtte være på 0 i amplitude litt før og litt etter overgangen. I motsatt fall rekker vi ikke å høre at lyden er fadet bort før en ny lyd plutselig kommer inn.

9.2.3.3 CPU-forbruk

Målingen av tids- og CPU-forbruk for audioeditoren ble foretatt 04/04/97 mellom klokken 17:30 og 17:40 på *baldrick.idi.ntnu.no*. Denne maskinen er av samme type som den som ble benyttet til videoevalueringen. Det var ingen andre personer som benyttet denne maskinen på det aktuelle tidspunktet. Testen ble kjørt mot lokal disk. Alle binærprogrammene som editoren benytter var også kopiert ned lokal disk.

Programmet ble kjørt med normal prioritet. Alle målinger er inludert oppstart og kjøring av «musicin» og «musicout». De presenterte tallene er gjennomsnittet etter tre gjennomkjøringer. Avvik i begge retninger er presentert i parenteser.

Følgende data ble registrert:

User tid:	75,88 sekund	(+0,26/-0,19)
System tid:	4,87 sekund	(+0,14/-0,10)
Medgått tid:	87,44 sekund	(+0,05/-0,07)
CPU-belastning:	92 %	(+0/-1)

Det er samme problemet her som i videodelen, med at kodingen til MPEG-1 tar lang tid. Siden det her er kodingen som tar lengst tid, er det ikke store muligheter for forbedringer så lenge «musicin» benyttes.

Desto mindre fadetid som velges, desto mindre rammer må dekodes, fades og kodes igjen. Mindre fadetid vil drastisk redusere redigeringstida. Av denne grunn må det velges små fadetider dersom lavt tidsforbruk er ønskelig. Dette må likevel ikke komme i konflikt med punktene nevnt i 9.2.3.2 slik at det blir et hørbart klikk i overgangen mellom klippene.

9.2.3.4 Bitrate

Evaluering av bitrate er ikke relevant for audioeditoren. På grunn av at ingen avhengigheter blir avskåret (finnes ikke i lag 1 og 2) og ingen rammer endrer lengde (forbudt i flg. standarden [ISO 11172]), er det enkelt å holde bitraten konstant gjennom hele strømmen. Bitraten er med andre ord den samme før rundt og etter overgangen mellom to klipp.

9.2.3.5 Synkronisering mellom audio og video

Denne er ikke tatt med her men i evalueringen av systemeditoren. Se avsnitt 9.2.1.5.

9.2.3.6 Samlet vurdering

Programmet gir det resultatet som var ønsket fra starten av. MPEG-1 audio kan redigeres tilnærmet uavhengig med liten eller ingen degradering av det opprinnelige signalet. Tidsforbruket er, i likhet med for videoeditoren, for stort.

Det ble foretatt en sjekk på hva de forskjellige fasene i gjennomkjøringen av audioeditoren brukte av tid. Fordelingen ble målt slik:

Dekoding av rammer:	ca. 26 sekunder
Koding av rammer:	ca. 52 sekunder
Søking/fading:	ca. 10 sekunder

Kodingen av nye rammer er med andre ord mest tidkrevende. Det er mest å hente på å effektivisere denne delen og dekodingen. En effektivisering av fading/søking ville ikke hjelpe mye på det totale tidsforbruket.

Konklusjonen blir at så lenge programmet ikke skal benyttes til tidskritiske funksjoner fungerer det godt.

10. OPPSUMMERING OG KONKLUSJON

10.1 Arbeidet som er utført

I dette prosjektet har det blitt utviklet en MPEG-1-systemeditor. I denne editoren kan det editeres på fritt valgte plasser i strømmene. Klipp kan ligge hvor som helst i video og audio, helt uavhengige av hverandre.

I utgangspunktet er ikke MPEG-1 laget med muligheter for editering. Dette gir seg utslag i at bilder i videodelen kodes etter *differansen* til forutgående og kommende bilder i samme videostrømmen. Et bilde kan med andre ord ikke bare taes ut av en strøm og settes sammen med en annen fordi differanseinformasjonen da vil bli utført på det siste bildet i den første strømmen. Dette kan gi uønskede effekter og «grums» i overgangen mellom to klipp.

For MPEG-1 audio er problemet litt mer differensiert. I lag 1 og 2 kodes et bestemt antall samplinger sammen i en ramme (temporært vindu, må ikke forveksles med videorammer som er en samling med data som skal vises samtidig). Det er ikke mulig å splitte opp denne rammen i to og bare beholde den ene delen fordi rammen må ha en forutbestemt lengde som er gitt i MPEG-1 standarden [ISO 11172]. I lag 3 er det ytterligere komplekst fordi her kodes rammene utfra innholdet i andre rammer. Det er derfor ikke mulig å plukke ut én ramme av strømmen og dekode denne.

For videodelen ble denne problematikken løst med å dekode de bildene som ligger i samme gruppe (inndeling med avhengigheter) som klippet i begge videostrømmene. Deretter kodes disse bildene med nye referansebilder slik at strømmene kan settes sammen.

I audiodelen ble det besluttet at lag 3 skulle droppes. Lag 3 er svært lite benyttet, og ville gjøre hele oppgaven svært mye vanskeligere å løse. Problematikken for lag 1 og lag 2 er svært lik. Siden det ikke var mulig å klippe midt i en ramme (untatt hvis hele strømmen dekodes), måtte alle klipp ligge ved en rammestart. Rammestarten som lå nærmest klippet måtte velges for å få minst feil. Det ble raskt funnet ut at dette alene ikke holdt. Dersom det blir klippet 18 ms feil for 100 klipp ville resultatet bli 18 ms * 100 = 1,8 sekunder. Det ble derfor bestemt at synkroniseringsfeil fra foregående klipp måtte taes vare på. Denne allerede oppsamlede synkroniseringsfeilen måtte taes med i avgjørelsen av hvilken rammestart klippet skulle ligge ved.

For å avgjøre hvilken rammestart som skulle velges, var det nødvendig å hente inn opplysninger om hvor store klippefeil som kunne tillates. I følge lest litteratur ble det kommet frem til en fordeling mellom «for tidlig lyd» og «for sen lyd» på omtrent -20 og +40 ms [LEYTEU91]. Siden dette er en fordeling på -1/3 og +2/3, ble det besluttet at den samme fordelingen skulle implementeres for å holde synkroniseringsfeilen godt innenfor kravene.

Det ble også oppdaget at dersom to audioklipp ble satt rett sammen, resulterte det i et hørbart klikk akkurat i overgangen mellom de to klippene. Siden dette ikke var tilfredsstillende, ble det besluttet at audioklippene skulle fades ned rundt

overgangen. Etter forsøk på dette, ble det også oppdaget at dersom lyden ble fadet ned slik at amplituden ble null akkurat i overgangen, vedvarte ikke nullpunktet så lenge at det ble oppfattet som en fullkommen utfading *til* null, og en fullkommen innfading *fra* null. Det ble derfor innført en skaleringsfaktor som sier hvor stor del av fadetiden som skal benyttes til faktisk fading. Denne kan varieres mellom 0 og 1, hvor null vil si ingen fading (rett ned på null) og 1 vil si at all tid skal benyttes til fading. Dersom skaleringsfaktoren settes til 1 er vi tilbake til utgangspunktet, som var før skaleringsfaktoren ble innført.

10.2 Konklusjon

Det er fullt mulig å implementere en MPEG-1 systemeditor. Vi må regne med en kvalitetsforringelse som varierer med komprimeringsrate. Det ble funnet ut at for ikke-profesjonelle formål er dette kvalitetstapet tilfredsstillende lavt, men siden det er merkbart er systemet ikke spesielt brukbart for profesjonelle formål.

På grunn av det store tidsforbruket kreves det godt forberedt redigering. I motsatt fall må redigering foretas gjenntatte ganger inntil en korrekt redigering er foretatt, og dette vil føre til en svært lang redigeringstid.

10.3 Videre arbeid

Det naturlige neste steget i utviklingen av dette systemet er et grafisk brukergrensesnitt. Et grafisk brukergrensesnitt betinger mulighet for bevegelser i begge retninger i strømmen. Når ønsket bilde er på skjermen, trykker brukeren bare på en knapp som forteller programmet at dette bildet skal skrives til klipplista (som nå blir generert av programmet selv) som start- eller sluttramme. Det grafiske grensesnittet har mulighet til å vise minst to videoer samtidig med eget kontrollpanel for hver av videoene. Dessverre blir det ikke mulig med «real time» visning av resultatet før en MPEG-1 maskinvarekoder («hardware») blir benyttet.

For audio kan samme metode benyttes som for PCM-editorer i dag. Dvs. at en tidslinje kommer opp for hver strøm. I denne tidslinja kan det velges et område. Brukeren kan deretter velge å spille av denne delen for å høre om det er denne delen han vil ha. Dersom utsnittet er som ønsket, klikker bruker på en knapp for å legge til dette området i den aktuelle audiostrømmen i klipplista. Brukeren må deretter velge hvor lang fadetid som skal benyttes, og hvilken skaleringsfaktor (denne må tillates lagt inn for hvert klipp i klipplista). Etter at fadeparametrene er valgt, skrives innslaget til klipplista.

Det vil også være naturlig å utvide systemet med flere metoder for å sette sammen strømmene som skal redigeres sammen. I denne oppgaven er det bare audiodelen som har mulighet for å spesifisere effekt for sammensetting. Denne muligheten er riktignok svært begrenset siden det kun er mulig å spesifisere lengde på fading av lyden. Det neste steget kunne være å gjøre det mulig å fade lydklippene mot hverandre, dvs. lyden før klippet fades ut samtidig som lyden etter klippet fades inn.

Det er ikke mange muligheter for mange typer avanserte klippmetoder i audio. I video derimot er det i teorien ubegrensede muligheter av fading og “wiping” (det nye klippet kommer gradvis inn i bildet uten fading, f.eks. fra midten og utover). Det kunne også la seg gjøre å få en pikselert overgang fra den ene til den andre videostrømmen.

Alle disse mulighetene kan legges inn i et menysystem som lar brukeren bestemme type effekt, varighet og diverse andre parametre som ville være effektavhengig.

10.3.1 System

Systemdelen har ikke noe særlig behov for forbedring. Det eneste kunne være en mulighet til å opprette et virtuelt filsystem for audio- og videoeditorene slik at de kunne gå rett inn i systemstrømmen og plukke rammer. Systemdelen ville da kunne fungere som en tjener som filtrerte bort all annen informasjon enn den det ble spurt etter.

10.3.2 Video

Dette programmet har helt klart potensiale til forbedring. Første punkt i så måte er å gjøre programmet i stand til å gjennomløpe en MPEG-1-videostrøm uten å måtte dekode alle rammene på veien.

For kodingen er det naturlig å benytte et eksternt program på grunn av kodingens kompleksitet. Denne delen av programmet fungerer tilfredsstillende.

En utvidelse av mulige spleisemetoder er forklart i 10.3.

10.3.3 Audio

Audioeditoren benytter eksterne program for koding og dekoding. Disse programmene fungerer tilfredsstillende. Programmet dekoder ikke hele strømmen for å finne bestemte rammer, men leser seg frem til rammen, og dekoder bare de rammene som skal dekodes.

Dersom det kommer et nytt klipp i samme audiostrøm og første ramme ligger etter siste ramme i forrige klipp, søkes det ikke fra starten, men fra den rammen filpekeren peker på. Denne metoden kunne bli ytterligere optimalisert ved å sortere klippene etter strømnummer, og deretter etter start- og sluttramme. På denne måten kunne vi unngå å starte lesing fra starten til det høyst nødvendige.

Et eksempel på en slik metode er dersom vi har tre klipp i samme audiostrømmen. Første klippet skal inneholde rammenummer 1450-2872, neste klippet skal inneholde rammenummer 8756-10582 og det siste klippet skal inneholde rammenummer 3870-7549. Slik programmet nå er søkes det fra starten til ramme nummer 1450 og kopierer frem til ramme nummer 2872. Deretter søkes det fra ramme nummer 2872 til ramme nummer 8756 og kopierer frem til ramme nummer 10582. Til slutt settes filpekeren til starten på fila, for så å søke seg frem til ramme nummer 3870 og kopiere frem til ramme nummer 7549. Med den foreslåtte forbedringen, ville det søkes etter klipp tre *før* klipp to. På denne måten hadde vi sluppet å søke på nytt fra starten etter rammene til klipp tre. Vi får da følgende søkemønster:
$0 \rightarrow 1450 \rightarrow 2872 \rightarrow 3870 \rightarrow 7549 \rightarrow 8756 \rightarrow 10582$

En utvidelse av mulige spleisemetoder er forklart i 10.3.

LITTERATURREFERANSER

[ISO 11172] ISO/IEC 11172: **INTERNATIONAL STANDARD. INFORMATION TECHNOLOGY — CODING OF MOVING PICTURES AND ASSOCIATED AUDIO FOR DIGITAL STORAGE MEDIA AT UP TO ABOUT 1,5 MBIT/S —.** *ISO/IEC, 1993.*

[HODGE95] WINSTON WILLIAM HODGE: **INTERACTIVE TELEVISION.** *MCGRAW-HILL, 1995.*

[FLUCK95] FRAQOIS FLUCKIGER: **UNDERSTANDING NETWORKED MULTIMEDIA APPLICATIONS AND TECHNOLOGY.** *PRENTICE HALL, 1995.*

[BRASTO94] KARLHEINZ BRANDENBURG, GERHARD STOLL: **ISO-MPEG-1 AUDIO: A GENERIC STANDARD FOR CODING OF HIGH-QUALITY DIGITAL AUDIO.** *JOURNAL OF THE AUDIO ENGINEERING SOCIETY AUDIO/ACOUSTIC/APPLICATIONS. VOL 42, NUMBER 10 OCT. 94, S. 780-792.*

[EDITER96] GAUTE ANDREAS DYBVIK: **EDITERING AV MPEG VIDEO.** *NTNU 1996.*

[ISO 13818] ISO/IEC 13818: **INTERNATIONAL STANDARD. INFORMATION TECHNOLOGY — GENERIC CODING OF MOVING PICTURES AND ASSOCIATED AUDIO —.** *ISO/IEC 1995.*

[BULLIE91] D. C. A. BULTERMAN, R. V. LIERE: **MULTIMEDIA SYNCHRONIZATION AND UNIX.** *PROCEEDINGS OF SECOND INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEM SUPPORT FOR DIGITAL AUDIO AND VIDEO. HEIDELBERG, GERMANY, NOV. 18-19, 1991, S. 109-119.*

[LEYTEU91] P. LEYDEKKERS, B. TEUNISSEN: **SYNCHRONIZATION OF MULTIMEDIA DATA STREAMS IN OPEN DISTRIBUTED ENVIRONMENTS.** *PROCEEDINGS OF SECOND INTERNATIONAL WORKSHOP ON NETWORK AND OPERATING SYSTEM SUPPORT FOR DIGITAL AUDIO AND VIDEO. HEIDELBERG, GERMANY, NOV. 18-19, 1991, S. 94-104.*

[STEMEY92] R. STEINMETZ, T. MEYER: **MULTIMEDIA SYNCHRONIZATION TECHNIQUES: EXPERIENCES BASED ON DIFFERENT SYSTEM STRUCTURES.** *PROCEEDINGS OF IEEE MULTIMEDIA '92 WORKSHOP. MONTEREY, CALIFORNIA, APRIL 2-4, 1992.*

[BERKPLA95] PLATEAU RESEARCH GROUP, COMPUTER SCIENCE DIVISION, UNIVERSITY OF CALIFORNIA, BERKELEY: **BERKELEY MPEG-1 VIDEO ENCODER.** *BERKELEY 1995.*

[BROWEN95] M. A. BROADHEAD, C. B. OWEN: **DIRECT MANIPULATION OF MPEG COMPRESSED DIGITAL AUDIO.** *PROCEEDINGS OF ACM MULTIMEDIA '95, SAN FRANCISCO, CALIFORNIA, NOVEMBER 5-9, 1995, S. 499-507.*

[DAVPAN95] DAVIS PAN: **A TUTORIAL ON MPEG/AUDIO COMPRESSION.** *IEEE MULTIMEDIA, VOLUME 2, NUMBER 2, SUMMER '95, S. 60-74.*

[GUOLU96] GUOJON LU: **COMMUNACATION AND COMPUTING FOR DISTRIBUTED MULTIMEDIA STREAMS.** *ARTECH HOUSE, 1996, S. 280-305.*

[WEDUGI95] R. WEISS, A. DUDA, D. K. GIFFORD: **COMPOSITION AND SEARCH WITH A VIDEO ALGEBRA.** *IEEE SPRING '96, S. 12-25.*

[POHLMA95] K. C. POHLMANN: **PRINCIPLES OF DIGITAL AUDIO.** *MCGRAW-HILL, INC., 3RD EDITION, 1995.*

A KILDEKODE

A.1 Systemeditoren

Systemeditoren består bare av systemmodulen.

A.1.1 Systemmodulen

Modul for alle funksjonene i systemeditoren.

system.cc

```
/* Kildekode til MPEG-1 systemnivå editor */

#define NUMARGS 3
#define AUDIO   1
#define VIDEO   2

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

int
main(int argc, char **argv)
{
  char *cline = (char*)malloc(255);
  char *temp = (char*)malloc(255);
  int numstreams = argc-3;
  char *vidOutfile = (char*)malloc(255);
  char *audOutfile = (char*)malloc(255);

  char *vidClipfile = (char*)malloc(255);
  char *audClipfile = (char*)malloc(255);
  char *clipline = (char*)malloc(255);
  int whichlist = 0;
  FILE *clip, *vidclip, *audclip;

  /* Sjekker om det ble gitt nok argumenter */
  if(argc<NUMARGS)
    {
      fprintf(stderr, "Usage: %s <inputMPS #1> [<inputMPS #2> ...] <outputMPS>
<clipfile>\n\n");
      exit(0);
    }

  /* OK. Splitter opp MPEG-1 systemfilene */
  for(int i=0; i<numstreams; i++)
    {
      /* Konstruerer kommandolinja for demultipleksing */
      sprintf(cline, "mpeg_demux %s", argv[i+1]);
      printf("\nDemultiplexing %s\n", argv[i+1]);

#ifdef _DEBUG
      fprintf(stderr, "%s\n", cline);
#endif

      /* Starter demultiplekseren */
      system(cline);
```

```
  }

/* Konstruerer de midlertidige video- og audioklipplistefilnavnene */
sprintf(vidClipfile, "%s.vid", argv[argc-1]);
sprintf(audClipfile, "%s.aud", argv[argc-1]);

/* Åpner klipplista og oppretter klipplistefiler for audio og video */
if((clip = fopen(argv[argc-1], "rb")) == 0)
   {
     fprintf(stderr, "Unable to open clipfile\n");
     exit(0);
   }
if((vidclip = fopen(vidClipfile, "wb")) == 0)
   {
     fprintf(stderr, "Unable to create temporary clipfile\n");
     exit(0);
   }
if((audclip = fopen(audClipfile, "wb")) == 0)
   {
     fprintf(stderr, "Unable to create temporary clipfile\n");
     exit(0);
   }

/* Splitter opp klipplista */
for(;!feof(clip);)
  {
    /* Leser inn en linje fra klipplista */
    fgets(clipline, 255, clip);

    /* Sjekker hvilken liste det er */
    if(strncasecmp(clipline, "[Audio]", 7) == 0)
      {
        whichlist = AUDIO;
      }
    else
      {
        if(strncasecmp(clipline, "[Video]", 7) == 0)
          {
            whichlist = VIDEO;
          }
        else
          {
            /* Skriver til klipplistene */
            if(whichlist == AUDIO && strlen(clipline)>4)
              {
                fprintf(audclip, "%s", clipline);
              }
            else
              {
                if(whichlist == VIDEO && strlen(clipline)>4)
                  {
                    fprintf(vidclip, "%s", clipline);
                  }
              }
          }
      }
  }

/* Konstruerer de midlertidige video- og audiofilnavnene */
sprintf(vidOutfile, "%s.vid", argv[argc-2]);
sprintf(audOutfile, "%s.aud", argv[argc-2]);

/* Konstruerer kommandolinja for videoeditoren */
sprintf(cline, "mpeg_edit ");

/* Løper gjennom alle argumentene for å legge til innfilenes */
/* filnavn til kommandolinja til videoeditoren */
for(int i=0; i<numstreams; i++)
  {
    strcat(cline, argv[i+1]);
    /* Sørger for å få med .vid-endelsen */
    strcat(cline, ".vid ");
  }

/* Legger til utfilnavnet */
strcat(cline, vidOutfile);
```

```
  strcat(cline, " ");

  /* Legger til klipplistefilnavnet */
  strcat(cline, vidClipfile);

#ifdef _DEBUG
  fprintf(stderr, "%s\n", cline);
#endif

  /* Starter videoeditoren */
  printf("\nStarting videoeditor\n");
  system(cline);

  /* Konstruerer kommandolinja for audioeditoren */
  sprintf(cline, "audio_edit ");

  /* Løper gjennom alle argumentene for å legge til innfilenes */
  /* filnavn til kommandolinja til audioeditoren */
  for(int i=0; i<numstreams; i++)
    {
      strcat(cline, argv[i+1]);
      /* Sørger for å få med .aud-endelsen */
      strcat(cline, ".aud ");
    }

  /* Legger til utfilnavnet */
  strcat(cline, vidOutfile);
  strcat(cline, " ");

  /* Legger til klipplistefilnavnet */
  strcat(cline, vidClipfile);

#ifdef _DEBUG
  fprintf(stderr, "%s\n", cline);
#endif

  /* Starter audioeditoren */
  printf("\nStarting audioeditor\n");
  system(cline);

  /* Setter sammen audio og video til MPEG-1 systemstrøm */
  sprintf(cline, "system_encode -o %s %s 0 %s 0", argv[argc-2],
          vidOutfile, audOutfile);
  printf("\nMultiplexing edited audio and video into %s\n", argv[argc-2]);
#ifdef _DEBUG
  fprintf(stderr, "%s\n", cline);
#endif

   /* Starter systemmultiplekseren */
  system(cline);

  /* Sletter video- og audiofilene */
  unlink(vidOutfile);
  unlink(audOutfile);
}
```

A.2 Audioeditoren

Audioeditoren består av audio-, clip- og streammodulen.

A.2.1 Audiomodulen

Audiomodulen inneholder *main()* og andre klasseløse funksjoner.

audio.cc

```
/* Kildekode til audiomodulen i MPEG-1 audioeditor */

#include "audio.h"
#include "stream.h"
#include "clip.h"
```

```
/* Definerer variabel for rammelengden */
double framelength;

/* Funksjon for avrunding av double til heltall */
int
round(double d)
{
  return int( (d >= 0)? d + 0.5 : d - 0.5);
}

/* Funksjon for å sammenligne headerdata */
int
CompareHeaders(cInputstream *stream1, cInputstream *stream2)
{
  int success = TRUE;

  /* Sammenligner layer */
  if(stream1->layer != stream2->layer)
    {
      fprintf(stderr, "Audiostreams differ in layer!\n");
      success = FALSE;
    }

  /* Sammenligner bitrate */
  if(stream1->bitrate != stream2->bitrate)
    {
      fprintf(stderr, "Audiostreams differ in bitrate!\n");
      success = FALSE;
    }

  /* Sammenligner samplingsfrekvens */
  if(stream1->smplfrq != stream2->smplfrq)
    {
      fprintf(stderr, "Audiostreams differ in sampling frequency!\n");
      success = FALSE;
    }

  /* Sammenligner lymodus */
  if(stream1->sndmode != stream2->sndmode)
    {
      fprintf(stderr, "Audiostreams differ in sound mode!\n");
      success = FALSE;
    }

  /* Sammenligner sammenligner kodemetode */
  if(stream1->modeext != stream2->modeext)
    {
      fprintf(stderr, "Audiostreams differ in mode extension!\n");
      success = FALSE;
    }

  /* Sammenligner emphasis */
  if(stream1->emphasis != stream2->emphasis)
    {
      fprintf(stderr, "Audiostreams differ in emphasis!\n");
      success = FALSE;
    }

  /* Returnerer suksesstatus */
  return success;
}

/* Hovedfunksjon for initialisering og oppstart */
int
main(int argc, char **argv)
{
  int i, numstreams;
  cInputstream *inputstream[255];
  cOutputstream *outputstream;
  cCliplist *cliplist;

  /* Sjekker om det er oppgitt nok argumenter (minst NUMARGS) */
  if(argc<NUMARGS)
    {
      fprintf(stderr,
              "Usage: %s <stream 1> <stream 2> ... <output> <cliplist>\n",
```

```
              argv[0]);
        return FALSE;
      }

  /* Legger antall innstrømmer i variabelen numstreams */
  numstreams = argc-3;

  /* Oppretter strømmene for lesing */
  for(i=0; i<numstreams; i++)
    {
      inputstream[i] = new cInputstream(argv[i+1]);

#ifdef _DEBUG
      printf("i=%i;inputstream: %x;numstreams: %d", i, inputstream[i],
numstreams);
      getc(stdin);
#endif

      /* Sammenligner headerdata til inputstrømmene */
      if(i>0 && (CompareHeaders(inputstream[i], inputstream[i-1]) == 0))
        {
          return FALSE;
        }
    }

  /* Oppretter klipplista */
  cliplist = new cCliplist(argv[argc-1], numstreams);

  /* Starter klippingen */
  cliplist->Startclipping(inputstream);

  /* Oppretter utstrømmen og legger klippene inn i den */
  outputstream = new cOutputstream(cliplist->clipno, argv[argc-2]);

  /* Sletter alle objektene for inputstrømmene */
  for(i=0; i<numstreams; i++)
    {
      delete inputstream[i];
    }

   /* Sletter objektet for klipplista */
  delete cliplist;

  /* Sletter objektet for utstrømmen */
  delete outputstream;

  return TRUE;
}
```

audio.h

```
/* Headerfil til audiomodulen i MPEG-1 audioeditor */

#ifndef AUDIO_H
#define AUDIO_H
/* Felles includefiler */
#include <stdio.h>
#include <stdlib.h>
#include <iostream.h>
#include <string.h>
#include <unistd.h>

#include "clip.h"
#include "stream.h"

#define NUMARGS 4
#define TRUE 1
#define FALSE 0

/* Definerer variabel for rammelengden som extern slik at alle som
   inkluderer denne fila kan bruke den */
extern double framelength;

/* Definisjon av avrundingsfunksjonen */
extern int round(double d);
```

```
/* Definisjon av Compareheaders */
extern CompareHeaders(cInputstream *stream1, cInputstream *stream2);
#endif
```

A.2.2 Clipmodulen

Clipmodulen innholder funksjoner som har med klipplista å gjøre.

clip.cc

```
/* Clipmodulen til MPEG-1 audioeditor */

#include "audio.h"
#include "stream.h"
#include "clip.h"

/* Destruktor for klippliste */
cCliplist::~cCliplist()
{
  sClippoint *temp;
  char *ctemp = (char*)malloc(255);

  /* Sletter alle klippunktene */
  while(TheList != NULL)
    {
      temp = TheList->next;
      delete TheList;
      TheList = temp;
    }

  /* Fjerner alle midlertidige filer */
#ifndef _DEBUG
  for(int i=0; i<clipno; i++)
    {
      sprintf(ctemp, "fadepre_%03d.mpa", i);
      unlink(ctemp);
      sprintf(ctemp, "fadepre_%03d.pcm", i);
      unlink(ctemp);
      sprintf(ctemp, "fadepost_%03d.mpa", i);
      unlink(ctemp);
      sprintf(ctemp, "fadepost_%03d.pcm", i);
      unlink(ctemp);
      sprintf(ctemp, "frames_%03d.mpa", i);
      unlink(ctemp);
      sprintf(ctemp, "clip_%03d.mpa", i);
      unlink(ctemp);
    }
#endif

  /* Lukker klipplistefila */
  fclose(clipfile);

  /* Frigjør variable */
  free(ctemp);
}

/* Konstruktor for klippliste */
cCliplist::cCliplist(char *filename, int numstreams)
{
  sClippoint *temp;

  /* Initialiserer variable */
  clipno = 0;
  outofsync = 0;

  /* Åpner klipplistefila */
  if((clipfile = fopen(filename, "rb")) == 0)
    {
      fprintf(stderr, "Unable to open audio cliplist: %s\n", filename);
      exit(FALSE);
    }
  else
    {
      TheList = new sClippoint;
```

```
      TheList->clipno = clipno++;
      ReadEntry(TheList);
      temp = TheList;
      CalculateClippoint(temp);

      /* Leser inn klipplista og legger innslagene i en lenket liste */
      for(;(!feof(clipfile));)
        {
          /* Sjekker at det ikke er lagt inn for høyt strømnummer */
          if(temp->streamno<numstreams)
            {
              temp->next = new sClippoint;
            }
          else
            {
              fprintf(stderr,
                      "Too high stream number in clipfile: %d  Exiting!\n",
                      temp->streamno);
              exit(0);
            }

          /* Kaller opp funksjonen som leser inn linja fra fila */
          if((ReadEntry(temp->next)) == FALSE)
            {
              delete temp->next;
              temp->next = NULL;
            }
          else
            {
              /* Flytter oss lengre ned i lista */
              temp = temp->next;
              CalculateClippoint(temp);
              temp->clipno = clipno++;
              temp->next = NULL;
            }
        }
    }
}

/* Kalkulerer klippunktet */
int
cCliplist::CalculateClippoint(sClippoint *clippoint)
{
  /* Beregner startramme */
  clippoint->startframe =  round(clippoint->start*1000 / framelength);
  /* Sjekker om synkfeilen blir innenfor grensen -2/3 +1/3 av rammelengden */
  if((clippoint->startframe*framelength - clippoint->start*1000 - outofsync) <
     (-framelength*(double(2)/double(3))))
    {
      /* Vi ligger for langt foran. Må klippe en ramme etter */
      clippoint->startframe++;
    }
  else
    if((clippoint->startframe*framelength - clippoint->start*1000 -
outofsync)>
     (framelength*(double(1)/double(3))))
    {
      /* Vi ligger for langt etter. Må klippe en ramme før */
      clippoint->startframe--;
    }

  /* Sørger for at vi ikke får klippunkt før ramme 0 */
  if(clippoint->startframe<0)
    clippoint->startframe = 0;
  /* Beregner ny synkfeil */
  outofsync -= clippoint->startframe*framelength - clippoint->start*1000;

#ifdef _DEBUG
  printf("Outofsync:%f ms*Startframe:%d", outofsync, clippoint->startframe);
  getc(stdin);
#endif

  /* Beregner sluttramme */
  clippoint->stopframe =  round(clippoint->stop*1000 / framelength);
  /* Sjekker om synkfeilen blir innenfor grensen -1/3 +2/3 av rammelengden */
  if((clippoint->stop*1000 - clippoint->stopframe*framelength - outofsync) <
```

```
      (-framelength/(double(2)/double(3))))
    {
      /* Vi ligger for langt foran. Må klippe en ramme etter */
      clippoint->stopframe++;
    }
  else
    if((clippoint->stop*1000 - clippoint->stopframe*framelength - outofsync)>
     (framelength/(double(1)/double(3))))
    {
      /* Vi ligger for langt etter. Må klippe en ramme før */
      clippoint->stopframe--;
    }

  /* Beregner ny synkfeil */
  outofsync -= clippoint->stop*1000 - clippoint->stopframe*framelength;

  /* Beregner antall rammer som må til for å fade med den tiden som */
  /* er oppgitt i klipplisten */
  clippoint->fadeframes = round(clippoint->fadetime*1000/framelength+.5);

#ifdef _DEBUG
  printf("Outofsync:%f ms*Stopframe:%d*Fadeframes:%d",
         outofsync, clippoint->stopframe, clippoint->fadeframes);
  getc(stdin);
#endif

  /* Sjekker at fadetiden ikke overgår klipplengden
     (fading skjer i starten og slutten av et klipp) */
  if((2*clippoint->fadeframes) >= (clippoint->stopframe-clippoint-
>startframe))
    {
      fprintf(stderr, "Cannot fade more frames than clip duration!\n");
      exit(0);
    }
  return TRUE;
}

/* Leser inn en linje fra klipplistefila og putter den
   inn i datastrukturen for et klippunkt */
int
cCliplist::ReadEntry(sClippoint *clippoint)
{
  size_t lenln;
  char *dummy = (char*)malloc(256);
  char *line = (char*)malloc(256);

  /* Leser inn linja fra fila */
  if((fgets(line, 255, clipfile)) != 0)
    {
      /* Trekker ut strømnummer */
      lenln = strspn(line, "0123456789");
      strncpy(dummy, line, lenln);
      dummy[lenln] = '\0';
      clippoint->streamno = char(atoi(dummy));
      line += lenln+1;

      /* Trekker ut starttidspunkt */
      lenln = strspn(line, "0123456789.");
      strncpy(dummy, line, lenln);
      dummy[lenln] = '\0';
      clippoint->start    = atof(dummy);
      line += lenln+1;

      /* Trekker ut stoptidspunkt */
      lenln = strspn(line, "0123456789.");
      strncpy(dummy, line, lenln);
      dummy[lenln] = '\0';
      clippoint->stop     = atof(dummy);
      line += lenln+1;

      /* Trekker ut fadetid */
      lenln = strspn(line, "0123456789.");
      strncpy(dummy, line, lenln);
      dummy[lenln] = '\0';
      clippoint->fadetime = atof(dummy);
```

```
#ifdef _DEBUG
      printf("Streamno:%d*Start:%6f*Stop:%6f*Fadetime:%6f", clippoint-
>streamno,
             clippoint->start, clippoint->stop, clippoint->fadetime);
      getc(stdin);
#endif

      return TRUE;
    }
  else
    {
      return FALSE;
    }
}

/* Funksjon for oppstart av selve klippeprosessen */
cCliplist::Startclipping(cInputstream **inputstream)
{
  char filename[255];
  unsigned long buffer, fadesamples, layersamples;
  int temp1, temp2;
  float smpltemp;
  unsigned char temp;

  FILE *frames;
  FILE *fadepre;
  FILE *fadepost;
  FILE *clip;
  int i, lest;
  sClippoint *cliptemp = TheList;

  /* Finner ut hvor mange samples som ligger i en ramme */
  if(inputstream[0]->layer == 2)
    layersamples = 1152;
  else
    layersamples = 384;

  /* Åpner alle filene for rammer og fading både før og etter rammene */
  for(i=0; (i<clipno && cliptemp != NULL); i++)
    {
      /* Rammefil klipp i */
      sprintf(filename, "frames_%03d.mpa", i);
      if((frames = fopen(filename, "w+b")) == NULL)
        {
          fprintf(stderr, "Unable to create temporary file: %s\n", filename);
          exit(0);
        }

      /* Innfading klipp i */
      sprintf(filename, "fadepre_%03d.mpa", i);
      if((fadepre = fopen(filename, "wb")) == NULL)
        {
          fprintf(stderr, "Unable to create temporary file: %s\n", filename);
          exit(0);
        }

      /* Utfading klipp i */
      sprintf(filename, "fadepost_%03d.mpa", i);
      if((fadepost = fopen(filename, "wb")) == NULL)
        {
          fprintf(stderr, "Unable to create temporary file: %s\n", filename);
          exit(0);
        }

      /* Hele klippet klipp i */
      sprintf(filename, "clip_%03d.mpa", i);
      if((clip = fopen(filename, "w+b")) == NULL)
        {
          fprintf(stderr, "Unable to create temporary file: %s\n", filename);
          exit(0);
        }

      /* Skriver rammene som skal dekodes for "fading inn" til disk */
      temp1 = cliptemp->startframe + cliptemp->fadeframes;
      inputstream[cliptemp->streamno]->Cutframes(cliptemp->startframe,
                                                 temp1, fadepre);
```

```
      fclose(fadepre);

      /* Skriver rammene som bare skal kopieres over til disk */
      temp1 = cliptemp->startframe + cliptemp->fadeframes;
      temp2 = cliptemp->stopframe - cliptemp->fadeframes;
      inputstream[cliptemp->streamno]->Cutframes(temp1, temp2, frames);

      /* Skriver rammene som skal dekodes for "fading ut" til disk */
      temp1 = cliptemp->stopframe - cliptemp->fadeframes;
      inputstream[cliptemp->streamno]->Cutframes(temp1, cliptemp->stopframe,
                                          fadepost);
      fclose(fadepost);

      /* Dekoder innfadingsrammene ved hjelp av musicout */
      /* Bruker 'filnavn' for å konstruere kommandolinja */
      sprintf(filename, "musicout fadepre_%03d.mpa fadepre_%03d.pcm", i, i);

#ifdef _DEBUG
      printf("%s\n", filename);
#endif

      /* Starter musicout */
      system(filename);

      /* Foretar fading på PCM-strømmen */
      sprintf(filename, "fadepre_%03d.pcm", i);
      fadesamples = cliptemp->fadeframes*layersamples;
      PCMfade(filename, "IN", fadesamples);

      /* Koder innfadingsrammene */
      /* Bruker 'filnavn' for å konstruere kommandolinja */
      smpltemp = float(inputstream[cliptemp->streamno]->smplfrq)/float(1000);
      sprintf(filename, "musicin -l %d -m %c -p 2 -s %3.1f -b %d -d %c
fadepre_%03d.pcm fadepre_%03d.mpa",
              inputstream[cliptemp->streamno]->layer,
              inputstream[cliptemp->streamno]->sndmode,
              smpltemp,
              inputstream[cliptemp->streamno]->bitrate,
              inputstream[cliptemp->streamno]->emphasis, i, i);

#ifdef _DEBUG
      printf("%s\n", filename);
#endif

      /* Starter musicin */
      system(filename);

      /* Dekoder utfadingsrammene ved hjelp av musicout */
      /* Bruker 'filnavn' for å konstruere kommandolinja */
      sprintf(filename, "musicout fadepost_%03d.mpa fadepost_%03d.pcm", i, i);

#ifdef _DEBUG
      printf("%s\n", filename);
#endif

      /* Starter musicout */
      system(filename);

      /* Foretar fading på PCM-strømmen */
      sprintf(filename, "fadepost_%03d.pcm", i);
      PCMfade(filename, "OUT", fadesamples);

      /* Koder innfadingsrammene */
      /* Bruker 'filnavn' for å konstruere kommandolinja */
      smpltemp = float(inputstream[cliptemp->streamno]->smplfrq)/float(1000);
      sprintf(filename, "musicin -l %d -m %c -p 2 -s %3.1f -b %d -d %c
fadepost_%03d.pcm fadepost_%03d.mpa",
              inputstream[cliptemp->streamno]->layer,
              inputstream[cliptemp->streamno]->sndmode,
              smpltemp,
              inputstream[cliptemp->streamno]->bitrate,
              inputstream[cliptemp->streamno]->emphasis, i, i);

#ifdef _DEBUG
      printf("%s\n", filename);
#endif
```

```
      /* Starter musicin */
      system(filename);

      /* Setter sammen til en strøm for dette klippet */
      sprintf(filename, "fadepre_%03d.mpa", i);
      fadepre = fopen(filename, "rb");
      sprintf(filename, "fadepost_%03d.mpa", i);
      fadepost = fopen(filename, "rb");
      rewind(frames);

      /* Kopierer over data fra inn-fading til en fil for dette kuttet */
      for(;(!feof(fadepre));)
        {
          lest = fread(&temp, 1, 1, fadepre);
          fwrite(&temp, 1, 1, clip);
        }
      /* Kopierer over de hele rammene */
      for(;(!feof(frames));)
        {
          lest = fread(&temp, 1, 1, frames);
          fwrite(&temp, 1, 1, clip);
        }
      /* Kopierer over data fra ut-fadingen */
      for(;(!feof(fadepost));)
        {
          lest = fread(&temp, 1, 1, fadepost);
          fwrite(&temp, 1, 1, clip);
        }

      /* Lukker alle åpne filer */
      fclose(fadepre);
      fclose(frames);
      fclose(fadepost);
      fclose(clip);
      cliptemp = cliptemp->next;
    }
}

/* Funksjon for utføring av fading på PCM rådata */
int
cCliplist::PCMfade(const char *filename, const char *fadetype,
                   long fadesamples)
{
  FILE *fadefile, *pcmfile;
  short sampling = 0, newsampling = 0, lest;
  double counter, multiplikator = 0, scalefactor = 0.9, hold;
  char *temp = (char*)malloc(100);

  /* Åpner filene */
  fadefile = fopen("fade.pcm", "wb");
  pcmfile  = fopen(filename, "rb");

  /* Sjekker hva slags fading som skal foretaes */
  if(strncasecmp(fadetype, "OUT", 3) == 0)
    {
      /* UTFADING */
      /* Leser inn fra input */
      for(counter=0;(!feof(pcmfile)); counter++)
        {
          lest = fread(&sampling, 1, 2, pcmfile);

          /* Beregner multiplikatoren som samplingen skal multipliseres med */
          multiplikator = 1-counter/(fadesamples*scalefactor);

          /* Sjekker at multiplikator ikke er under null (->180 gr.
fasevridning) */
          if(multiplikator<0)
            multiplikator = 0;

          /* Regner ut den nye verdien av samplingen */
          newsampling = (short)(sampling * multiplikator);

          /* Skriver den nye samplingen til fil */
          fwrite(&newsampling, 1, lest, fadefile);
        }
```

```
    }
  else
    {
      if(strncasecmp(fadetype, "IN", 2) == 0)
        {
          /* INNFADING */
          /* Bestemmer hvor mange samples ut i strømmen vi skal starte */
          hold = fadesamples*(1-scalefactor);

          /* Leser inn fra input */
          for(counter=0;(!feof(pcmfile)); counter++)
            {
              lest = fread(&sampling, 1, 2, pcmfile);

              /* Sjekker om fadingen skal starte */
              if(counter>hold)
                {
                  /* Beregner multiplikatoren som samplingen skal multipliseres
med */
                  multiplikator = (counter-hold)/(fadesamples*scalefactor);
                }

              /* Sjekker at multiplikator ikke er over 1 (->over opprinnelig
verdi) */
              if(multiplikator > 1)
                {
                  multiplikator = 1;
                }

              /* Regner ut den nye verdien av samplingen */
              newsampling = (short)(sampling * multiplikator);

              /* Skriver den nye samplingen til fil */
              fwrite(&newsampling, 1, lest, fadefile);
            }
        }
    }

  /* Lukker filene */
  fclose(fadefile);
  fclose(pcmfile);

  /* legger den fadete filen over i den andre */
  sprintf(temp, "mv fade.pcm %s", filename);
  system(temp);
}
```

clip.h

```
/* Headerfil til clipmodulen i MPEG-1 audioeditor */

#ifndef CLIP_H
#define CLIP_H
#include "audio.h"
#include "stream.h"

/* Struktur for klipplistedata o.l. */
struct sClippoint
{
  int clipno;
  char streamno;
  double start;
  int startframe;
  double stop;
  int stopframe;
  double fadetime;
  int fadeframes;
  sClippoint *next;
};

/* Klassestruktur for klipplista */
class cCliplist
{
public:
  cCliplist(char *filename, int numstreams);
  ~cCliplist();
```

```
  Startclipping(cInputstream **inputstream);

  sClippoint *TheList;
  int clipno;

private:
  int ReadEntry(sClippoint *clippoint);
  int CalculateClippoint(sClippoint *clippoint);
  int PCMfade(const char *filename, const char *fadetype, long fadesamples);

  FILE *clipfile;
  double outofsync;
};
#endif
```

A.2.3 Streammodulen

Streammodulen inneholder funksjoner som har med input- og outputstrømmer å gjøre.

stream.cc

```
/* Streammodulen i MPEG-1 audioeditor */

#include "audio.h"
#include "stream.h"
#include "clip.h"

/* Destruktor for input audiostrøm */
cInputstream::~cInputstream()
{
  /* Lukker inputstrømmen */
  fclose(filestream);
}

/* Konstruktor for opprettelse av en input audiostrøm */
cInputstream::cInputstream(const char *filename)
{
  /* Åpner fila med audiostrømmen */
  if((filestream = fopen(filename, "rb")) == 0)
    {
      fprintf(stderr, "Unable to open audio stream: %s\n", filename);
      exit(FALSE);
    }
  else
    {
      /* Leser inn headeren i strømmen */
      if(ReadHeader() == FALSE)
        {
          /* Skriver ut feilmelding og terminerer
             dersom headeren ikke ble godkjent */
          fprintf(stderr,
                  "Not a supported audio stream or incorrect header: %s\n",
                  filename);
          exit(FALSE);
        }
    }
  /* Setter filpekeren til starten på inputfila */
  rewind(filestream);
}

/* Funksjon for å lese inn headeren i strømmen */
int
cInputstream::ReadHeader()
{
  int success = TRUE;
  unsigned int buffer = 0;
  unsigned char temp = 0;

  /* Sjekker at det ikke er en nullstrøm */
  if(((fread(&temp, 1, 1, filestream)) == 1))
    {
      buffer = temp;
      buffer = (buffer << 8);
```

```
      /* Leser inn en byte og skifter på plass */
      fread(&temp, 1, 1, filestream);
      buffer = ((buffer & 0xff00) | (temp & 0x00ff));

#ifdef _DEBUG
      printf("buffer=%04x", buffer);
      getc(stdin);
#endif

      /* Gjennkjenner en ISO 11172-3 layer I audiostrøm */
      if((buffer & 0xfffe) == 0xfffe)
        {
          layer = 1;

          /* Regner ut rammelengden i millisekunder */
          framelength = float(384);
          printf("Layer I, ");
        }
      else
        {
          /* Gjennkjenner en ISO 11172-3 layer II audiostrøm */
          if((buffer & 0xfffc) == 0xfffc)
            {
              layer = 2;

              /* Regner ut rammelengden i millisekunder */
              framelength = float(1152);
              printf("Layer II, ");
            }
          else
            {
              /* Gjennkjenner en ISO 11172-3 layer III audiostrøm */
              if((buffer & 0xfffa) == 0xfffa)
                {
                  fprintf(stderr, "Layer III not supported!\n");

                  /* Returnerer til kallende funksjon med feilstatus */
                  return FALSE;
                }
              else
                /* Returnerer med feilstatus fordi det ikke er en audiostrøm */
                return FALSE;
            }
        }

      /* Nullstiller buffer */
      buffer = 0;
      temp = 0;

      /* Leser inn resten av headeren */
      fread(&temp, 1, 1, filestream);
      buffer = temp;
      buffer = (buffer << 8);
      fread(&temp, 1, 1, filestream);
      buffer = ((buffer & 0xff00) | (temp & 0x00ff));

#ifdef _DEBUG
      printf("buffer=%04x", buffer);
      getc(stdin);
#endif

      /* Filtrerer ut den informasjonen vi er interessert i */
      bitrate  = (buffer & 0xf000) >> 12;
      smplfrq  = (buffer & 0x0c00) >> 10;
      sndmode  = (buffer & 0x00c0) >>  6;
      modeext  = (buffer & 0x0030) >>  4;
      emphasis = (buffer & 0x0003);

      /* Oversetter lydmodus til navn som kan brukes til musicin */
      switch(sndmode)
        {
        case 0:
          sndmode = 's';
          break;
        case 1:
```

```
    sndmode = 'j';
    break;
  case 2:
    sndmode = 'd';
    break;
  case 3:
    sndmode = 'm';
    break;
  }

/* Oversetter emphasis til navn som kan brukes til musicin */
switch(emphasis)
  {
  case 0:
    emphasis = 'n';
    break;
  case 1:
    emphasis = '5';
    break;
  case 3:
    emphasis = 'c';
    break;
  }

/* Oversetter samplingshastigheten til Hz */
switch(smplfrq)
  {
  case 0:
    smplfrq = 44100;
    framelength = framelength/44.1;
    break;
  case 1:
    smplfrq = 48000;
    framelength = framelength/48;
    break;
  case 2:
    smplfrq = 32000;
    framelength = framelength/32;
    break;
  }

/* Oversetter bitraten til kbit/s */
switch(bitrate)
  {
  case 0x1:
    bitrate = 32;
    break;
  case 0x2:
    if(layer == 1)
      bitrate = 64;
    else
      bitrate = 48;
    break;
  case 0x3:
    if(layer == 1)
      bitrate = 96;
    else
      bitrate = 56;
    break;
  case 0x4:
    if(layer == 1)
      bitrate = 128;
    else
      bitrate = 64;
    break;
  case 0x5:
    if(layer == 1)
      bitrate = 160;
    else
      bitrate = 80;
    break;
  case 0x6:
    if(layer == 1)
      bitrate = 192;
    else
      bitrate = 96;
```

```
          break;
        case 0x7:
          if(layer == 1)
            bitrate = 224;
          else
            bitrate = 112;
          break;
        case 0x8:
          if(layer == 1)
            bitrate = 256;
          else
            bitrate = 128;
          break;
        case 0x9:
          if(layer == 1)
            bitrate = 288;
          else
            bitrate = 160;
          break;
        case 0xa:
          if(layer == 1)
            bitrate = 320;
          else
            bitrate = 192;
          break;
        case 0xb:
          if(layer == 1)
            bitrate = 352;
          else
            bitrate = 224;
          break;
        case 0xc:
          if(layer == 1)
            bitrate = 384;
          else
            bitrate = 256;
          break;
        case 0xd:
          if(layer == 1)
            bitrate = 416;
          else
            bitrate = 320;
          break;
        case 0xe:
          if(layer == 1)
            bitrate = 448;
          else
            bitrate = 384;
          break;
        }

      /* Skriver ut rammelengden til skjerm */
      printf("framelength: %10.10f ms\n", framelength);

#ifdef _DEBUG
      printf("Bitrate:%d*Smplfrq:%d*Sndmode:%c*Modeext:%d*Emphasis:%c\n",
            bitrate, smplfrq, sndmode, modeext, emphasis);
#endif

    }
  else
    success = FALSE;

  return success;
}

/* Funksjon for å kutte rammer fra strømmen og skrive disse til disk */
cInputstream::Cutframes(int firstframe, int lastframe, FILE *outfile)
{
  int funnet = FALSE;
  int i, lest = 0;
  static int frameno = -1;
  static unsigned short buffer = 0;
  unsigned char temp = 0;

  /* Sjekker om vi må starte søk fra begynnelsen eller stående ramme */
```

```
  if(frameno > firstframe || frameno == -1)
    {
      /* NEI. Rammenummer, filpeker og buffer må tilbakestilles */
      frameno = -1;
      buffer = 0;
      rewind(filestream);
    }
  else
    /* Sjekker om vi står på den første ramma i klippet */
    if(frameno == firstframe)
      {
       funnet = TRUE;

#ifdef _DEBUG
       printf("Funnet:%d ", frameno);
#endif
      }

  /* Søker etter syncword */
  for(;(!feof(filestream) && !funnet);)
    {
      /* Leser inn en byte fra fila */
      fread(&temp, 1, 1, filestream);
      buffer = ((buffer << 8) & 0xff00) | (temp & 0x00ff);

      /*#ifdef _DEBUG
      printf("Cutframebuffer:%04x:%02x", buffer, temp);
      getc(stdin);
      #endif*/

      /* Sjekker om det er syncword */
      if((buffer & 0xfff0) == 0xfff0)
        {
          frameno++;

          /*#ifdef _DEBUG
          printf("Syncword:%d", frameno);
          getc(stdin);
          #endif*/

          /* Sjekker om det er korrekt ramme */
          if(frameno == firstframe)
            {
              funnet = TRUE;
#ifdef _DEBUG
              printf("Funnet:%d\n", frameno);
#endif
            }
        }
    }

  /* Sjekker at det ikke ble stopp pga filslutt */
  if(funnet)
    {
      temp = (buffer >>  8) & 0x00ff;
      lest = fwrite(&temp, 1, 1, outfile);
      funnet = FALSE;
      for(i=0;(!feof(filestream) && !funnet);i++)
        {

          /*#ifdef _DEBUG
          printf("Cutframebuffer:%04x:%02x", buffer, temp);
          getc(stdin);
          #endif*/

          /* Skriver bufferet ut til fil */
          fread(&temp, 1, 1, filestream);
          buffer = ((buffer << 8) & 0xff00) | (temp & 0x00ff);

          /* Sjekker om det er syncword */
          if((buffer & 0xfff0) == 0xfff0)
            {
              frameno++;

              /* Sjekker om det er korrekt ramme */
              if(frameno == lastframe)
```

```
                {
                  funnet = TRUE;
#ifdef _DEBUG
                  printf("Funnet:%d\n", frameno);
#endif
                }
            }

          if(!funnet)
            {
              /* Skriver resten av bufferet til utfila */
              temp = (buffer >> 8) & 0x00ff;
              lest += fwrite(&temp, 1, 1, outfile);
            }
        }
    }
  else
    {
      /* Slutten på fila ble nådd før ramma ble funnet */
      fprintf(stderr, "End of input file reached %d!\n", frameno);
      exit(0);
    }
}

/* Konstruktor for utstrøm */
cOutputstream::cOutputstream(int clipno, const char* filename)
{
  long buffer = 0;
  int lest = 0, i;
  FILE *clipfile;
  char *clipfilename = (char*)malloc(255);

  /* Åpner utstrømmen */
  if((filestream = fopen(filename, "wb")) == NULL)
    {
      fprintf(stderr, "Unable to create output stream: %s\n", filename);
      exit(FALSE);
    }

  /* Kopierer over alle klippene til utstrømmen */
  for(i=0; i<clipno; i++)
    {
      sprintf(clipfilename, "clip_%03d.mpa", i);
      clipfile = fopen(clipfilename, "rb");

      /* Leser klippet ogskriver det over til utfila */
      for(;(!feof(clipfile));)
        {
          lest = fread(&buffer, 1, 4, clipfile);
          fwrite(&buffer, 1, lest, filestream);
        }
      fclose(clipfile);
    }
}

/* Destruktor for utstrøm */
cOutputstream::~cOutputstream()
{
  /* Lukker utfila */
  fclose(filestream);
}
```

stream.h

```
/* Headerfil til streammodulen i MPEG-1 audioeditor */

#ifndef STREAM_H
#define STREAM_H
#include "audio.h"
#include "clip.h"

/* Klassestruktur for input audiostrømmer */
class cInputstream
{
public:
  char layer;
```

```
  unsigned bitrate;
  unsigned smplfrq;
  char sndmode;
  char modeext;
  char emphasis;

  cInputstream(const char *filename);
  ~cInputstream();

  Cutframes(int firstframe, int lastframe, FILE *outfile);

private:
  int ReadHeader();

  FILE *filestream;
};

/* Klassestruktur for output audiostrøm */
class cOutputstream
{
public:
  cOutputstream(int clipno, const char* filename);
  ~cOutputstream();

private:
  FILE *filestream;
};
#endif
```