TECHNISCHE UNIVERSITEIT DELFT
Faculteit der Elektrotechniek
Vakgroep Telecommunicatie- en
Verkeersbegeleidingssystemen

Titel: Opslag en Transmissie van geluids-signalen in één videoframe ten behoeve van een beeld databank

Auteur: R.F. Mous

Codenummer: 1-6824-277

Datering: 14 juni 1988

Korte inhoud: In het kader van een door Philips Telecommunicatie & Informatie-Systemen (PTIS) te ontwikkelen databank voor televisie beelden is er onderzoek verricht naar de mogelijkheden om een geluidsfragment met een duur van ongeveer 10 seconden, in de tijd te comprimeren, te coderen en binnen één NTSC video frame te versturen. De genoemde compressie zal digitaal plaatsvinden. Hieraan ontleent het systeem zijn naam, te weten Digital Speed-Up (DSU) systeem.

De uiteindelijk gekozen codering bestaat uit digitale meer-niveau transmissie om de digitale waarden behorende bij het gecomprimeerd geluidsfragment te versturen.

Dit verslag behandelt zowel de overwegingen die hebben geleid tot de keuze van meer-niveau transmissie alsmede een aantal onderdelen van de implementatie van het DSU systeem, waaraan tijdens het afstudeeronderzoek is meegewerkt.

VOORWOORD

Dit verslag is geschreven in het kader van het afstuderen bij de vakgroep Telecommunicatie- en Verkeersbegeleidingssystemen aan Faculteit der Electrotechniek van de Technische Universiteit Delft. Het onderzoek waarover verslag wordt gedaan is uitgevoerd bij de afdeling Business Development van Philips Telecommunicatie- en Informatiesystemen te Den Haag.

Bij mijn afstudeerwerkzaamheden ben ik begeleid door Dhr. T. Coenen (TU), Dhr. L. Kool (PTIS) en Dhr. C. Ruisch (PTIS), waarvoor ik hen zeer erkentelijk ben.

Robert Mous
Juni 1988

INHOUD pag. nr.

SAMENVATTING

In het kader van een door Philips Telecommunicatie & Informatie-Systemen (PTIS) te ontwikkelen databank voor televisie beelden is er onderzoek verricht naar de mogelijkheden om een geluidsfragment met een duur van ongeveer 10 seconden, in de tijd te comprimeren, te coderen en binnen één NTSC video frame te versturen.

Een analyse van dit probleem leidt al snel tot de conclusie dat de genoemde compressie digitaal zal moeten plaatsvinden. Hieraan ontleent het systeem zijn naam, te weten **Digital Speed-Up** systeem (DSU).

Om een geschikte kanaalcodering te kiezen voor transmissie van de digitale bemonsteringen is een afweging gemaakt van een aantal mogelijkheden, te weten: meer niveau modulatie, analoge modulatie en meer-niveau modulatie in kwadratuur. De uiteindelijk gekozen codering bestaat uit digitale meer-niveau transmissie.

Dit verslag behandelt zowel de overwegingen die hebben geleid tot de keuze van meer-niveau transmissie alsmede een aantal onderdelen van de implementatie van het DSU systeem, waaraan tijdens het afstudeer-onderzoek is meegewerkt. Deze onderwerpen bestaan uit het maken van analoge schakelingen, digitale schakelingen en het schrijven van computer programma's.

LIJST VAN AFKORTINGEN EN SYMBOLEN

ACK	ACKnowledge
ADC	Analoog Digitaal Convertor
AM	Amplitude Modulatie
ASCII	American Standard Code for Information Interchange
BDB	Beeld DataBank
BER	Bit-Error-Rate
bit	Binary unIT of Binary digIT
byte	Acht bits
CCITT	Comité Consultatif International Télégraphique et Téléphonique
CD	Compact Disc
CPU	Central Processing Unit
DAC	Digitaal Analoog Convertor
dB	deci Bel
DSU	Digital Speed-Up
FM	Frequentie Modulatie
h	hexadecimaal
IC	Integrated Circuit
ISDN	Integrated Services Digital Network
I/O	Input Output
kb	Kilo byte
kbps	Kilo bit per seconde
kHz	Kilo Hertz
OOK	On Off Keying
Mb	Mega byte
MHz	Mega Hertz
Modem	Modulator-demodulator
NACK	Negative ACKnowledge
NTSC	National Television Systems Committee
PAL	Phase Alternation by Line
PDOS	Paul's Disk Operating System
PM	Fase Modulatie
P&S Base	Picture and Sound Base
QAM	Kwadratuur Amplitude Modulatie
RAM	Random Access Memory
VAX	Computer van Digital Equipment Corporation
VLP	Video Long Play
VSB+C	Vestigial Side Band + Carrier
WORM	Write Once Read Many
Z-80	Zilog 80 microprocessor
680X0	Serie microprocessoren van Motorola

1. INLEIDING

De telecommunicatie-infrastructuur heeft in de afgelopen jaren een enorme ontwikkeling meegemaakt. De huidige satellietverbindingen en de meer uitgebreide toepassingen van telefooncentrales vormen hiervan een voorbeeld. Ook de ontwikkeling van informatie-voorzieningen in de huiskamer staat niet stil. Het Franse Minitel-systeem is hier een voorbeeld van. Ook binnen Nederland is een aantal projecten op telecommunicatie gebied gaande. Een daarvan is het kabelexperiment Zuid-Limburg. Binnen dit project wordt gebruik gemaakt van de door Philips ontwikkelde beeld databank (BDB). Wat dit precies inhoudt zal later in deze inleiding worden uitgelegd. Bij het ontwikkelen van een nieuwe beeld databank is het de bedoeling om geluid toe te voegen aan stilstaande beelden. Het doel van deze afstudeeropdracht is het onderzoeken en eventueel implementeren van de mogelijkheden om een geluidsfragment te comprimeren, op te slaan, te versturen en bij een gebruiker van de beeld databank weer om te zetten in het oorspronkelijke geluidsfragment. Dit alles binnen het kader van het te ontwikkelen nieuwe Philips beeld databank systeem. Alvorens de opdracht wat duidelijker te omschrijven is het verstandig om eerst in te gaan op het Zuid-Limburg project en de daar gehanteerde beeld databank. Dit vormt een achtergrond voor het ontstaan van deze opdracht.

Het Zuid-Limburg project heeft als doel het ontwikkelen en testen van nieuwe kabeltelevisiediensten voor een grote groep gebruikers, alsmede het bepalen van het effect van die nieuwe diensten op die gebruikers. Een belangrijk kenmerk van het Zuid-Limburg project is dat het daarbij toegepaste systeem interactief is. Dat wil zeggen dat de abonnee van het kabeltelevisienet de mogelijkheid wordt geboden om specifieke diensten aan te vragen. Het aanvragen van diensten gaat via het telefoonnet. De door de abonnee te ontvangen diensten geschieden in de vorm van tekst pagina's en/of videobeelden. Het ontvangen van stilstaande videobeelden op aanvraag is hierbij een wezenlijk nieuwe dienst van het Zuid-limburg project. Deze videobeelden zijn opgeslagen in de reeds eerder genoemde Philips beeld databank. Om van deze diensten gebruik te kunnen maken, beschikt de gebruiker over een huiskamermodule. De huiskamermodule is een losstaand kastje welke door

de gebruiker via een toetsenbord met afstandsbediening wordt aangestuurd. Deze huiskamermodule vormt de schakel tussen de televisie, het kabeltelevisienet en het telefoonnet.

Uit de beeld databank kan men beelden van een laservision disk ophalen en die versturen via het kabelnet. In dit verslag zal het begrip Video Long Play (VLP) worden gebruikt als verzamelnaam van laservision disks. Tenzij anders vermeld wordt hiermee ook WORM (Write Once Read Many) disks, een andere type laservision disk, bedoeld. Zo'n VLP is een optische schijf waarop analoge beeldinformatie is opgeslagen. Een nadeel van het ontvangen van stilstaande beelden is dat er slechts 30 msec. geluid met zo'n videobeeld ontvangen kan worden. De beeld databank in Zuid-Limburg heeft dan ook geen geluid behorende bij een videobeeld. Wel is het mogelijk om bijvoorbeeld achtergrond muziek bij het gebruikte kanaal uit te zenden. Dit kan dan door alle abonnees worden ontvangen. Zoals in het begin van de inleiding is aangegeven is het de bedoeling om bij een nieuw te ontwikkelen beeld databank geluid toe te voegen aan stilstaande beelden. Dit zal gebeuren door het geluidsfragment apart van het videobeeld naar de gebruiker te versturen. Om het breedbandige kabelnet zo effectief mogelijk te gebruiken zal eerst onderzocht moeten worden in hoeverre het mogelijk is smalbandig geluid in de tijd te comprimeren. Dit comprimeren in de tijd wordt gerealiseerd door het geluid ten koste van bandbreedte te versnellen; dit noemt men bij Philips "Digital Speed-Up" (DSU).

De nieuwe beeld databank waarin het DSU systeem geïmplementeerd zal worden wordt op dit moment door Philips voor GTE Service Corporation ontworpen. GTE Service Corporation is een in Needham (Massachusets USA) gevestigd bedrijf, dat zich onder andere bezighoudt met de exploitatie van kabeltelevisienetten. De beeld databank zal deel uit gaan maken van een op te zetten Teleservices Network van GTE.

Deze afstudeeropdracht zal zich richten op het onderzoeken van de mogelijkheden van bovengenoemde DSU systeem en en zal de volgende zes onderwerpen bevatten.

Ten eerste een beschrijving van de uitgangssituatie van het onderzoek. Binnen het GTE netwerk zijn de beeld databank en de huiskamermodule van belang voor die uitgangssituatie. Deze worden in hoofdstuk 2 beschreven.

Ten tweede zullen er aan de hand van de uitgangssituatie in hoofdstuk 3 criteria worden opgesteld waaraan het DSU systeem zal moeten voldoen. Hier zal het eigenlijke probleem van de digitalisering (broncodering) van het geluid en modulatie (kanaalcodering) van het versneld geluid worden gedefinieerd.

Ten derde zal er een inventarisatie worden gemaakt van bron- en kanaalcoderingen voor het DSU systeem. Deze inventarisatie wordt voor de broncodering in hoofdstuk 4 en voor de kanaalcodering in hoofdstuk 5 gedaan.

Ten vierde zullen in hoofdstuk 6 de geïnventariseerde systemen aan de hand van de opgestelde criteria worden getoetst op bruikbaarheid voor het DSU systeem. Deze toetsing zal uitmonden in een conclusie.

Ten vijfde zal het verslag in hoofdstuk 7 een aantal onderdelen van de implementatie van het DSU systeem belichten.

Als laatste zal in hoofdstuk 8 een algehele conclusie van het afstudeeronderzoek met een aanbeveling worden gegeven.

2. GTE TELESERVICES NETWORK

Het GTE Teleservices Network zal diensten verlenen aan gebruikers van een vijftal verschillende kabeltelevisienetten. Dit hoofdstuk zal de structuur van het GTE Teleservices Network beschrijven en daarna ingaan op de belangrijkste onderdelen van het netwerk voor wat betreft de ontwikkeling van het DSU systeem. Die onderdelen zijn de beeld databank en de huiskamermodule.

2.1. Opbouw van het GTE teleservices network

Figuur 2.1 geeft schematische opbouw van het netwerk weer. Bij de figuren in dit verslag zullen dikke lijnen gebruikt worden om breedbandige verbindingen aan te geven. Andere lijnen zullen worden gebruikt om smalbandige verbindingen aan te geven. In figuur 2.1 is te zien dat de beeld databank, in dit nog op te zetten netwerk, vijf breedbandige uitgangen en drie smalbandige ingangen heeft. De ingangen van de beeld databank zullen met een aantal VAX computers communiceren. Deze VAX computers ontvangen via de huiskamermodule en het telefoonnet aanvragen voor diensten van de gebruiker (abonnee van het kabeltelevisienet). De abonnee heeft deze aanvraag ingetoetst via het bedieningspaneel van de huiskamermodule. De stippellijn in figuur 2.1 stelt de infrarode verbinding tussen de huiskamermodule en het bedieningspaneel voor. De aldus door de abonnee aangevraagde dienst wordt door één van de VAX computers verwerkt. Indien een beeld van de beeld databank naar de gebruiker moet worden verstuurd zal deze VAX computer een aanvraag hiervoor naar de beeld databank versturen. Dit videobeeld wordt aan één van de vijf uitgangen van de beeld databank aangeboden (afhankelijk van de bestemming van het beeld) en wordt via het kabeltelevisienet door de huiskamer module ontvangen. De huiskamermodule zal het beeld hetzij als stilstaand videobeeld, hetzij als gecomprimeerd geluidsfragment herkennen. De aldus ontvangen beeld zal in de juiste vorm (video c.q. audio) aan de gebruiker worden aangeboden.

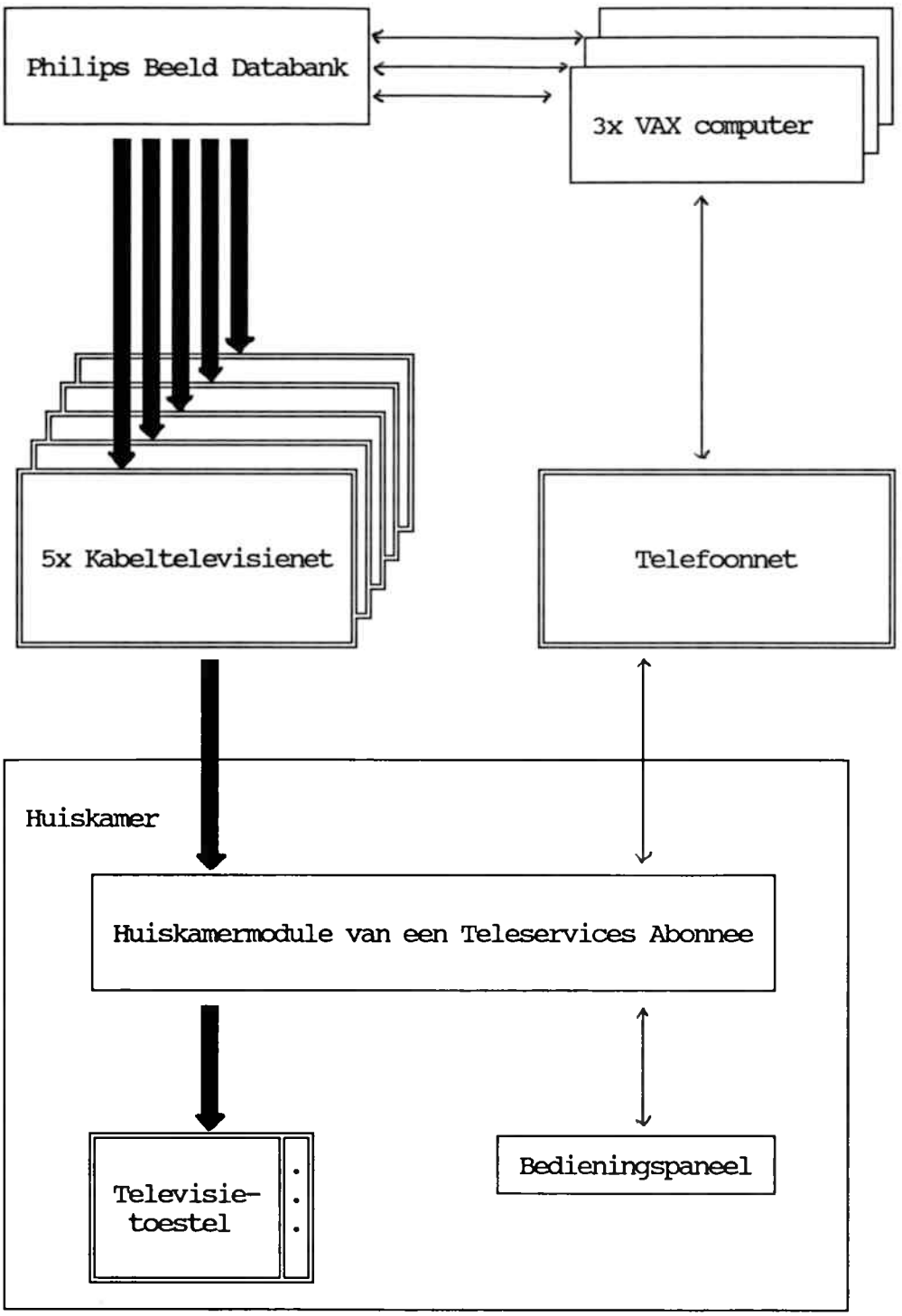


Fig. 2.1 De opbouw van de Teleservices Network van GTE, met:

| Smalbandig signaal verbinding.

▮ Breedbandig signaal verbinding.

| Infrarode afstandsbediening.

2.2. De Philips beeld databank

Over het algemeen wordt het begrip databank gebruikt voor de opslag en zoek systemen met betrekking tot gegevensbestanden van moderne computers. Deze databanken kunnen voor allerlei doeleinden worden gebruikt, met gegevens meestal alfanumeriek van aard. De Philips beeld databank is echter een databank waarbij de opgeslagen gegevens uit videobeelden bestaan. Bij deze databank is het mogelijk om een of meerdere videobeelden op te vragen.

De te ontwerpen beeld databank voor GTE Service Corporation in Amerika wordt P&S BASE (Picture & Sound Base) genoemd. In de rest van dit rapport zal het begrip beeld databank worden gehanteerd, ook al wordt er over opslag van geluidssignalen in de databank gesproken. In dit verslag wordt tevens onderscheid gemaakt tussen de volgende begrippen:

- Beeldsignaal: informatie (hetzij video, hetzij gecomprimeerd audio) in de vorm van een NTSC videoframe.
- Videobeeld: een beeldsignaal dat video-informatie bevat.
- Audiobeeld: een beeldsignaal dat gecomprimeerd audio-informatie bevat.

De volgende subparagrafen zullen een beschrijving van de beeld databank te zien geven. Voor een nog uitgebreider technische beschrijving wordt verwezen naar [Philips PTIS]. [Philips PTIS] is een systeembeschrijving van de Philips beeld databank geschreven door Dhr. C. Ruisch.

2.2.1. Structuur van de beeld databank

De Philips beeld databank is een modulair opgebouwd systeem dat bestaat uit drie delen, te weten:

- Een management systeem.
- Een opslag systeem.
- Een verzend systeem.

Deze drie gedeelten van de beeld databank zijn in figuur 2.2 weergegeven. De beeld databank werkt als volgt: Het management systeem ontvangt van een abonnee via een VAX een aanvraag voor een bepaald beeldsignaal. Deze aanvraag wordt vervolgens doorgestuurd naar een van de VLP controlers van het opslag systeem. Het beeldsignaal wordt van VLP opgehaald en het management systeem ontvangt hiervan bericht. Als laatste krijgt het verzend systeem bericht dat er een beeldsignaal klaar is om verzonden te worden. Het verzend systeem biedt het beeldsignaal, voorafgegaan door abonnee-adres, aan de uitgang van de beeld databank aan. Het abonnee-adres zorgt ervoor dat de betreffende huiskamermodule van de abonnee het beeldsignaal herkent. In de volgende subparagrafen wordt nader ingegaan op de werking van deze drie delen van de beeld databank.

2.2.1.1. Het management systeem

Het Management systeem bestaat uit een 68000 microprocessorsysteem. Dit systeem is opgebouwd uit de volgende onderdelen.

- CPU kaart met 512Kb RAM en 68020 processor (25 MHz).
- 2 seriële I/O kaarten met elk 8 RS-232 kanalen.
- 52Mb Winchester disk met controller.
- 640Kb diskette drive met controller.
- Voeding en behuizing.

Het management systeem wordt bestuurd door het PDOS besturings systeem. Het systeem ontvangt aanvraag gegevens via een drietal RS-232 poorten. In totaal heeft het management systeem 16 RS-232 poorten. Figuur 2.2 laat zien dat naast deze drie poorten er nog 8 poorten zijn gereserveerd voor communicatie met de opslag- en verzend-systemen en twee voor de terminal en printer van de beheerder. Er blijven dus drie poorten ongebruikt. De eigenlijke functie van het management systeem bestaat uit vijf delen.

1. Het ontvangen van aanvragen.
2. Het controleren van die aanvragen op geldigheid.

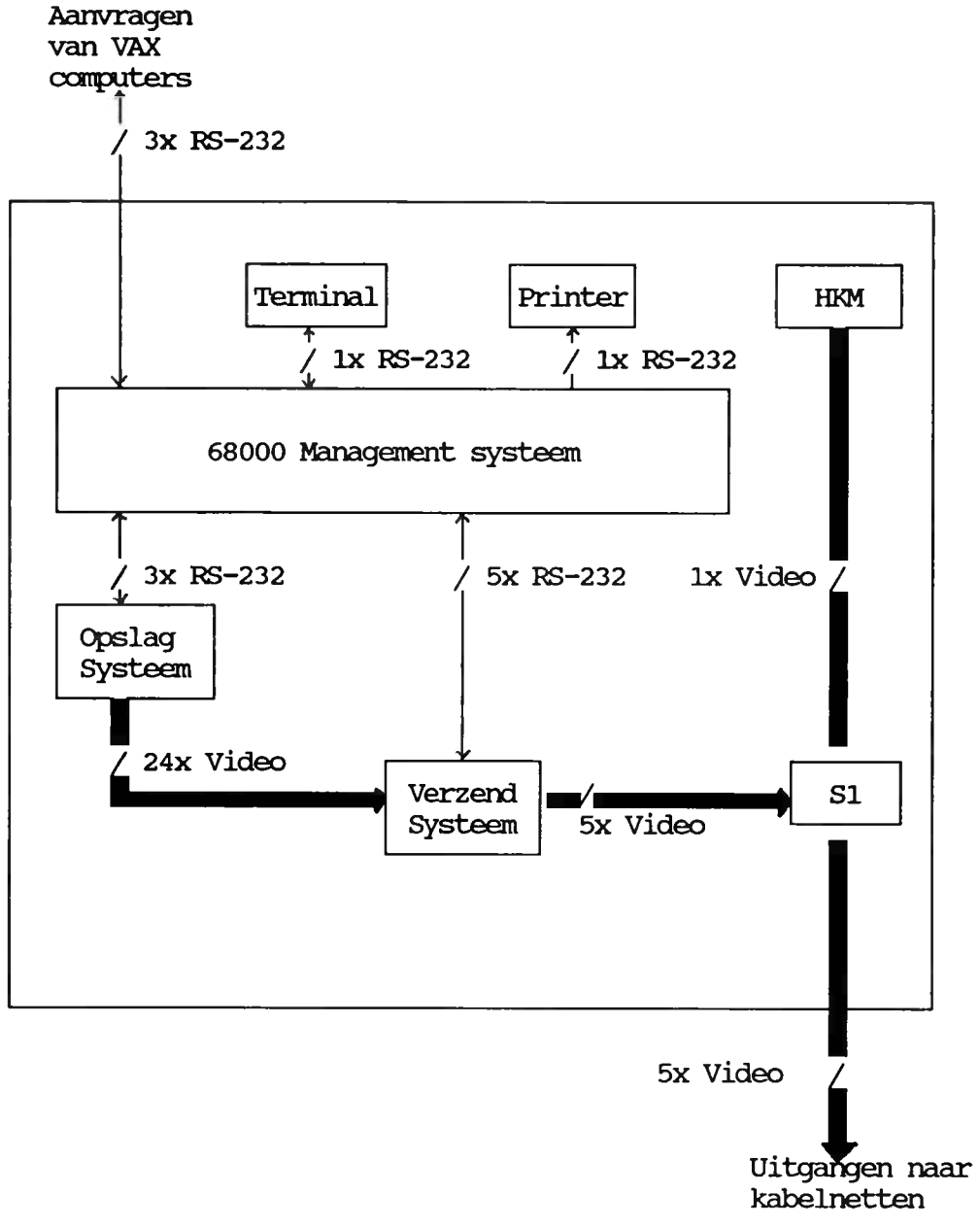


Fig. 2.2 Schematische opbouw Philips beeld databank, met:

| Smalbandig signaal verbinding.

▮ Breedbandig signaal verbinding.

HKM Aangepaste huiskamermodule.

S1 Schakelaar voor signaal keuze HKM.

3. Het optimaliseren en doorsturen van die aanvraag naar de controller van de VLP disk die het snelst het gewenste beeld kan leveren.
4. Het aansturen van het verzend systeem.
5. Het onderhouden, testen en loggen van beeld databank activiteiten.

Het ontvangen van aanvragen gebeurt, via een RS-232 poort, met een snelheid van 19.2 Kbps. Deze snelheid is nodig om de gewenste 150 aanvragen per seconde te kunnen ontvangen. Hierop wordt nader ingegaan in paragraaf 2.2.4.1. Er bestaat een protocol voor communicatie tussen het 68000 systeem en de VAX computers. Dit protocol wordt beschreven in paragraaf 2.2.2., en is op dit ogenblik niet van belang voor de verwerking van aanvragen. De ontvangen gegevens zijn na het verwijderen van het omhullende protocol te beschrijven als in- en uitvoergegevens van een beeldsignaal. Zo'n aanvraag bevat de volgende vier gegevens.

1. Frame header ten behoeve van het beeldsignaal. Dit bestaat uit het abonnee-adres, sequence nummer van het bericht, mode bits, background music bits en checksum voor de frame header.
2. Disk groep letter. Dit is een letter dat aangeeft op welke VLP het beeldsignaal staat.
3. Track nummer. Dit is het adres van het beeldsignaal op de betreffende VLP.
4. Channel nummer. Dit nummer geeft het uitgangskanaal behorende bij deze aanvraag aan.

Deze ontvangen gegevens zullen worden gecontroleerd op geldigheid. Dit houdt twee dingen in.

1. Het ontvangen adres van het videobeeld moet binnen een bepaald adresbereik liggen.
2. De gespecificeerde VLP moet aanwezig zijn op een van de VLP spelers.

Indien hierop gecontroleerd is, zal het management systeem bovengenoemde disk groep letter en track nummer doorsturen naar het opslag systeem. Dit doorsturen houdt in het verzenden van letter en track gegevens naar een van de drie Z-80 microcomputer systemen die zich bezighouden met de besturing van de VLP spelers. Omdat van iedere plaat bekend is wat het laatst verzonden beeld is, weet het management systeem precies waar de leeskop van iedere VLP speler staat. Hierdoor is het management systeem in staat om de meest gunstige VLP speler te kiezen en het beeld ervan op te halen. Uiteraard kan dit alleen indien dezelfde plaat op meerdere VLP spelers draait. Het is gebleken dat deze optimalisatie een verbetering van "performance" geeft doordat de VLP spelers geen onnodige bewegingen met hun leeskop hoeven te maken.

Het aansturen van het verzend systeem gebeurt zodra het management systeem een positief bericht heeft ontvangen van het opslag systeem. Dit betekent dat het opslag systeem een beeld van VLP heeft opgehaald en dit aan de uitgang van de betreffende VLP speler voor verdere verwerking aanbiedt.

Voor het onderhouden, testen en loggen van beeld databank activiteiten is er een beheerders terminal met printer aanwezig. Zoals figuur 2.2 laat zien is er ook een lokale "huiskamermodule" (HKM) met monitor voor testdoeleinden gereserveerd. Het woord huiskamermodule is in dit verband niet helemaal juist, het gaat hier over een module dat beelden onafhankelijk van het meegestuurd adres kan decoderen. Deze huiskamermodule kan middels schakelaar S1 in figuur 2.2 gekoppeld worden aan een willekeurig uitgangs kanaal.

2.2.1.2. Het opslag systeem

Het opslag systeem van de beeld databank bestaat uit 16 VLP spelers en 8 WORM spelers. Op iedere VLP kunnen ongeveer 50000 beeldsignalen worden opgeslagen. De WORM schijven kunnen slechts de helft (25000) van het aantal beeldsignalen opslaan. Elk beeld heeft een uniek adres op een plaat. Er zijn drie Z-80 systemen die elk acht spelers aansturen.

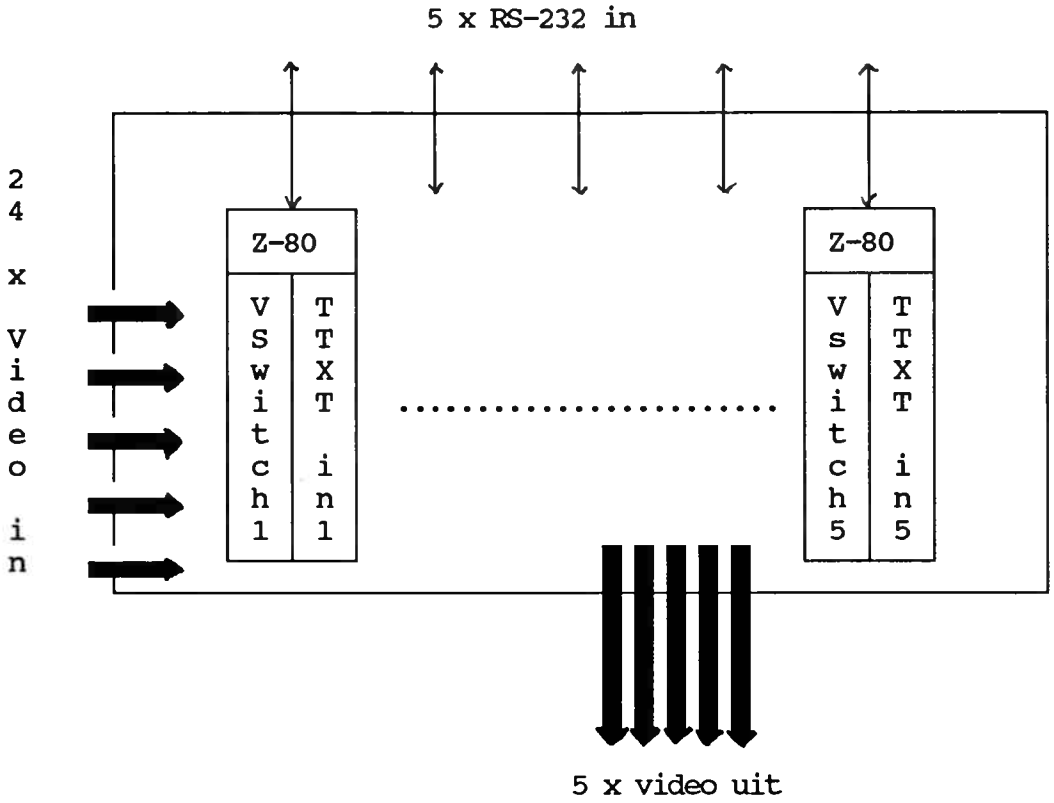


Fig. 2.3 Opbouw van het verzend systeem van de Philips beeld data-bank. (Vswitch1 = Video switch nummer 1, TTXT in1 = Teletext insertor nummer 1 enz.)

De opslag van beelden op de VLP's is eenmalig. Bij de WORM schijven is het ook mogelijk de VLP's te beschrijven. Dit gedeelte van de opslag zal echter los van de beeld databank gebeuren. In principe wordt er dus uitgegaan van VLP's die slechts gelezen kunnen worden. Het door de Z-80 ontvangen adres wordt door gegeven aan de VLP speler die na ongeveer 1/2 seconde het beeldsignaal aan de uitgang aanbiedt. Daarna zal het Z-80 systeem een acknowledge aan het management systeem sturen. De VLP speler blijft het beeldsignaal aan zijn video-uitgang aanbieden tot een volgende aanvraag is verwerkt.

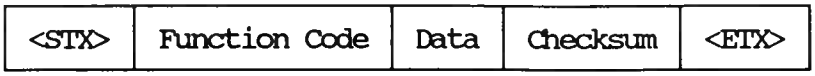
2.2.1.3. Het verzend systeem

Figuur 2.2 laat zien dat het verzend systeem 24 beeldsignaal ingangen, 5 RS-232 ingangen en 5 beeldsignaal uitgangen heeft. Figuur 2.3 geeft een meer uitgewerkte weergave van het ontvangst systeem weer. Zoals in deze figuur is te zien bestaat het verzend systeem uit een vijftal videoswitches gecombineerd met teletekst insertors. De vijf schakel-eenheden worden elk door één Z-80 systeem aangestuurd. Deze systemen ontvangen elk via een eigen RS-232 verbinding met het management systeem een aantal gegevens, samengevat in de in paragraaf 2.2.1.1. genoemde frame header. Dit wordt volgens het NTSC teletekstnorm in het beeldsignaal geplaatst dat door een van de VLP spelers wordt aan-geboden. Het verzend systeem zal een acknowledge naar het management systeem sturen indien het gelukt is een beeld te versturen.

2.2.2. De aansturing van de beeld databank

De communicatie tussen de beeld databank en de VAX computers verloopt via een software protocol op basis van informatie pakketten. Het formaat van zo'n informatie pakket is in figuur 2.4 afgebeeld. Het protocol ondersteunt de volgend vijf functies.

1. Frame Request (waarde = 30 hex (h), van VAX naar BDB).
2. Keep Alive (waarde = 31h, van VAX naar BDB).
3. System Shutdown (waarde = 32h, van VAX naar BDB).
4. Error Message (waarde = 33h, van BDB naar VAX).
5. Keep Alive Ack (waarde = 34h, van BDB naar VAX).



<STX>	:	start text (ASCII waarde 2). lengte= 8 bits.
Function code	:	waarde tussen 30h en 34h. lengte= 8 bits.
Data	:	waarde afhankelijk van functie. lengte afhankelijk van functie.
Checksum	:	waarde is 8 bits som van data. lengte = 8 bits.
<ETX>	:	einde text (ASCII waarde 3). lengte = 8 bits

Fig. 2.4 Formaat van berichten tussen VAX en 68000 systeem.

Een frame request is een bericht met gegevens voor de beeld databank omtrent het op te halen beeld en de abonnee-adres. De overige vier berichten zijn om de communicatie tussen VAX en 68000 te controleren en onderhouden. Deze functies zijn tot een minimum aantal beperkt.

2.2.3. De uitvoer van de beeld databank

De beeld databank kan vijf beeldsignalen af geven welke voldoen aan de NTSC standaard. In appendix A is deze standaard uitgebreid beschreven. Omdat er vijf uitgangen zijn, kan de beeld databank op vijf verschillende kabeltelevisienetten worden aangesloten. Zoals in paragraaf 2.2.1.3. is vermeld bevat het beeldsignaal informatie uit de frame header voor de abonnee. Deze (binaire) informatie is in lijn 12 van het videobeeld gecodeerd volgens NTSC teletekst norm.

2.2.4. Performance en testmethoden van de beeld databank

Met de "performance" van de beeld databank worden drie dingen bedoeld, te weten:

- Het maximaal aantal te produceren beeldsignalen per seconde.
- De kwaliteit van het video uitgangssignaal.
- De robuustheid tegen fouten en storingen.

Over de kwaliteit van het signaal, de robuustheid en de testmethoden kan nog weinig worden gezegd. Dit komt omdat er nog weinig over bekend is. Het maximaal aantal te produceren beeldsignalen wordt in de volgende paragraaf besproken.

2.2.4.1. Maximaal aantal te produceren beeldsignalen

Het maximaal aantal te produceren beeldsignalen per seconde is vastgesteld op 150. Omdat de beeldbank vijf uitgangen heeft is dit getal gelijk aan 30 verschillende beelden per video-uitgang per seconde. Dit laatste is precies gelijk aan de NTSC beeldfrequentie. De beeld databank moet dus in staat zijn om bij elk van de vijf kabeltelevisienetten één kanaal maximaal te belasten. Het produceren van

150 beeldsignalen per seconde is overigens nog niet haalbaar binnen de huidige opzet van de beeld databank, er wordt vooralsnog uitgegaan van een performance van ongeveer 50 beeldsignalen per seconde.

Om echter wel in staat te zijn om 150 beeldsignalen per seconde te produceren, zal de beeld databank ook 150 aanvragen per seconde moeten verwerken. Hierom is het nodig dat de beeld databank, zoals vermeld in paragraaf 2.2.1.1., wordt aangestuurd met 3 verbindingen van elk 19.2 Kbps. Deze snelheid is bepaald door het aantal bits per aanvraag te vermenigvuldigen met het aantal aanvragen per seconde en daarna voor alle zekerheid een overcapaciteit van 50% in te calculeren (zie [Philips PTIS] paragraaf 1.2).

2.3. De huiskamermodule

Om videobeelden te ontvangen is een gewoon televisietoestel voldoende. Indien één videobeeld continu moet worden afgebeeld zal er aan de kant van de gebruiker, in een zogenaamde huiskamermodule, extra hardware moeten zijn om dit effect te realiseren. Daarnaast zal de huiskamermodule de communicatie van de gebruiker naar de VAX moeten verzorgen. Deze paragraaf behandelt de huiskamermodule zoals die zal worden ontworpen voor het GTE project. In fig. 2.1 is te zien dat de huiskamermodule, binnen het GTE netwerk, de schakel vormt tussen de gebruiker, zijn televisietoestel, het kabelnet en het telefoonnet. De volgende twee subparagrafen zullen respectievelijk de functionaliteit en de opbouw van de huiskamermodule bespreken.

2.3.1. De functionaliteit van de huiskamermodule

De huiskamermodule moet een viertal functies vervullen:

- Het doorsluisen van het gewone programma aanbod naar het televisietoestel.
- Het detecteren en opslaan van beeldsignalen.
- Het continu aanbieden van stilstaande videobeelden aan de uitgang van de huiskamermodule, eventueel aangevuld met geluid.

- Het onderhouden van besturingscommunicatie met één van de centrale VAX computers.

Het doorsluisen van gewone programma's naar het toestel van de gebruiker is natuurlijk een primair vereiste. Indien hieraan wordt voldaan kan de gebruiker nog dezelfde functionaliteit van zijn toestel verwachten als die vóór de installatie van de huiskamermodule.

Het detecteren van stilstaande beelden is belangrijk. Indien dit gebeurt kan zo'n beeld opgeslagen worden in een lokaal geheugen om daarna aan het televisietoestel te worden aangeboden. Dat het mogelijk moet zijn een los beeld te herkennen betekend dat zo'n beeld zich moet onderscheiden van standaard televisiebeelden. Hoe dit wordt gedaan wordt uitgelegd in paragraaf 2.3.2.

Het continu aanbieden van stilstaande beelden aan de uitgang van de huiskamermodule houdt in dat de module het opgeslagen beeld gesynchroniseerd aan de uitgang moet blijven aanbieden. Dit moet dan wel een gemoduleerd signaal zijn. Een standaard televisietoestel is namelijk in staat slechts signalen in de UHF en VHF band te ontvangen. Dit houdt in dat de huiskamermodule het gedemoduleerd en opgeslagen signaal moet remoduleren naar een gewenste band. Daarnaast zal bij dit videosignaal geluid worden gemengd. Dit mengen is afhankelijk van de toestand van de mode bits, genoemd in paragraaf 2.2.1.1. Deze mode bits geven bijvoorbeeld aan dat er een gedecodeerd audiobeeld bij het videosignaal gemengd moet worden.

Het versturen van abonnee-aanvragen naar de centrale computer gebeurt via het telefoonnet. In principe is de keuze van transportsysteem willekeurig. De communicatie kan op vele manieren geschieden. Het telefoonnet is hiervoor wel het meest praktische. Ten eerste is bijna iedereen in het bezit van een telefoon. Ten tweede kan een computer op eenvoudige wijze aan het telefoonnet worden gekoppeld voor datacommunicatie. Ten slotte zijn er in de V.S. geen lokale telefoon tarieven, de kosten hiervan zijn bij het abonnement inbegrepen. Dit betekent dat het gebruiken van de telefoon voor Teleservices (in

eerste instantie) geen extra kosten voor de abonnee met zich meebrengt.

2.3.2. De opbouw van de huiskamermodule

Om de bovenstaande functies te vervullen zal de huiskamermodule de volgende zeven onderdelen moeten bevatten.

- Besturings mechanisme voor de afzonderlijke componenten.
- Modem voor het telefoonnet.
- Demodulator van ontvangen beeldsignalen.
- Detectiemechanisme voor beeldsignalen (hetzij video, hetzij audio).
- Videobeeld decoder met geheugen.
- Audiobeeld decoder met geheugen.
- Remodulator voor videobeelden.

De besturing van de huiskamermodule bestaat uit een microprocessor welke de communicatie met de centrale computer en de gebruiker verzorgt. Daarnaast bestuurt deze microprocessor de verschillende onderdelen van de huiskamermodule.

De modem verzorgt de interface tussen het telefoonnet en de microprocessor (besturing).

De demodulator van de ontvangen beeldsignalen wordt hier volledigheidhalve genoemd en zal slechts dienen om één kanaal van het binnenkomende signaal te demoduleren ten behoeve van verwerking door de huiskamermodule. Alle andere kanalen worden gemoduleerd doorgegeven aan de televisietoestel van de gebruiker.

De detectie mechanisme voor beeldsignalen zal als eerste het beeldsignaal verwerken. Het detectie mechanisme moet de volgende drie functies vervullen.

- Het herkennen van het abonnee adres welke zich in lijn 12 van het beeldsignaal bevindt.
- Het herkennen van een beeldsignaal als videobeeld.

- Het herkennen van een beeldsignaal als audiobeeld.

Het detectie mechanisme zal alleen die beeldsignalen doorlaten die bestemd zijn voor het ontvangend huiskamermodule.

De videobeeld en audiobeeld decoder bemonsteren beide het beeldsignaal om deze informatie zodoende in een digitaal geheugenbank te plaatsen. Een enkele geheugenbank bestaat uit een groot aantal seriële geheugen IC's welke speciaal ontworpen zijn voor video opslag. De organisaties van beide geheugens zullen echter verschillen omdat de beeldsignalen op verschillende manieren worden bemonsterd.

De remodulator van videobeelden maakt het mogelijk de eenmalig in het videogeheugen opgeslagen beeld, continu aan de uitgang van de huiskamermodule aan te bieden. De besturing zorgt ervoor dat het videobeeld steeds herhaald uit het videogeheugen wordt gelezen en in analoge vorm aan de remodulator wordt aangeboden.

3. HET DIGITAL SPEED-UP SYSTEEM

In het voorgaande hoofdstuk is beschreven op welke wijze de beeld databank en de huiskamermodule functioneren binnen het teleservices network. Hierbij zijn twee belangrijke verschillen op te merken tussen de beeld databank en de huiskamermodule.

Ten eerste is de beeld databank een opslagpunt van beeldsignalen, die onafhankelijk is van de informatie welke zich in zo'n beeldsignaal bevindt. Zo vormt de beeld databank een onderdeel van het communicatie kanaal van het DSU systeem.

Ten tweede zal de huiskamermodule de decoder van het DSU systeem bevatten. In dit opzicht is de huiskamermodule in tegenstelling tot de beeld databank een onderdeel van het DSU systeem zelf.

Dit hoofdstuk zal de doelstelling van het DSU systeem binnen de beeld databank verklaren. Daarna wordt het DSU systeem als systeem beschreven en zal ook de encoder aan de orde komen en dit zal tenslotte uitmonden in een aantal criteria die voor het ontwerp van het systeem van belang zullen zijn.

3.1. De doelstelling van het Digital Speed-Up systeem

De wens tot uitbreiding van de Philips beeld databank heeft geleid tot het opzetten van het systeem DSU met als doel het uiteindelijk realiseren van deze uitbreiding. Dit doel luidt als volgt:

> Het doel van het Philips Digital Speed-Up systeem is het aanbieden van een geluidsfragment van 10 seconden geluid van "medium quality", behorende bij een stilstaand videobeeld afkomstig van de Philips beeld databank, aan gebruikers van bovengenoemde beeld databank.

Met "medium quality" wordt een audio signaal bedoeld met een bandbreedte van 30 Hz tot 7 KHz, een S/N van minimaal 45 dB (psofometrisch gemeten) en een dynamisch bereik van 60 dB. Dat het nastreven van deze

doelstelling gepaard gaat met het comprimeren van geluid in de tijd is al in de inleiding van dit verslag ter sprake gekomen. In de volgende paragraaf zal het principe van deze compressie binnen het DSU systeem verduidelijkt worden.

3.2. Het principe van Digital Speed-Up

Het Digital Speed-Up systeem heeft de in de inleiding vermelde functie van het comprimeren van een geluidsfragment, om op deze wijze zo efficiënt mogelijk gebruik te maken van een video communicatiekanaal ten behoeve van geluids transport. Met betrekking tot deze compressie zal er een opslag methode voor het geluid moeten zijn. In dit verband is het belangrijk een functionele definitie van het DSU systeem te geven.

> Het Digital Speed-Up systeem zal een audio fragment van ongeveer 10 seconden transformeren naar een beeldsignaal van 1/30 seconde en het (na transport) terug transformeren van dat beeldsignaal naar het oorspronkelijke audio fragment.

Fig. 3.1 toont schematisch het DSU principe. Zoals in dit figuur is te zien, wordt het geluid gecomprimeerd door het in te lezen in een geheugen gedurende een tijd t, en het uit lezen van het geheugen gedurende een tijd t/G. De factor G is de compressiefactor van het systeem.

De benaming Digital Speed-Up is ontstaan omdat de opslag van audio-informatie digitaal zal geschieden en de compressie ontstaat uit versneld uitlezen van het geheugen. De omgekeerde bewerking van vertragen cq. expansie in de tijd (digital slow down) zal ook gebeuren via een geheugen maar heeft geen aparte naam. Het gehele systeem staat bekend als het Digital Speed-Up systeem.

3.3. De structuur van het Digital Speed-Up systeem

Uit de functionele definitie van het DSU systeem in de vorige paragraaf is af te leiden dat het DSU systeem een onderdeel van het audio

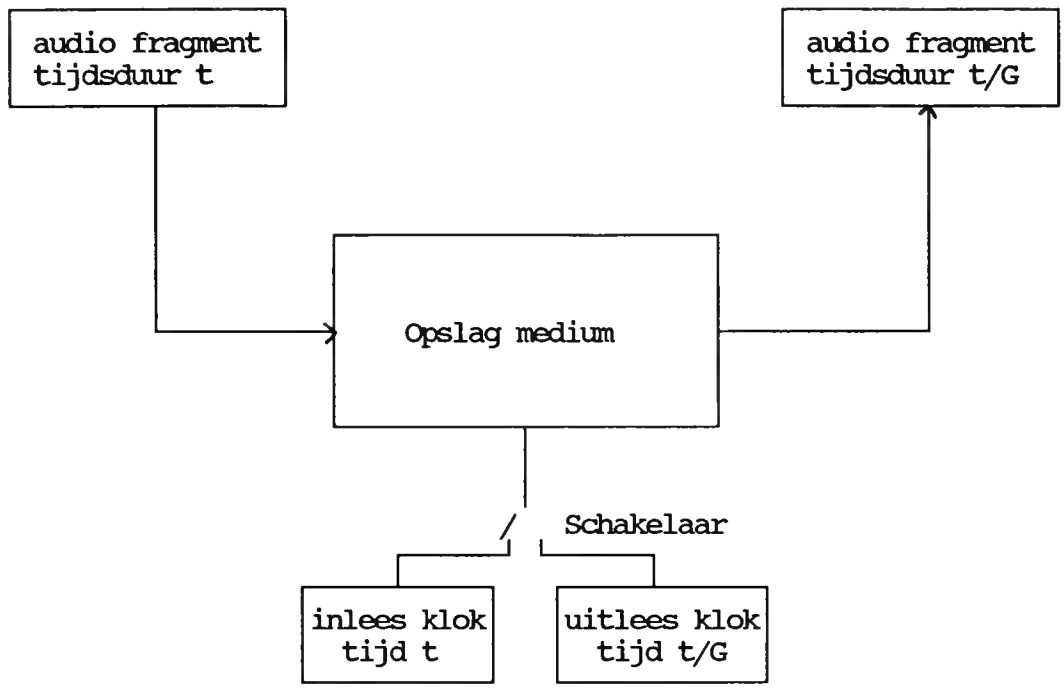


Fig. 3.1 Het principe van Digital Speed-Up

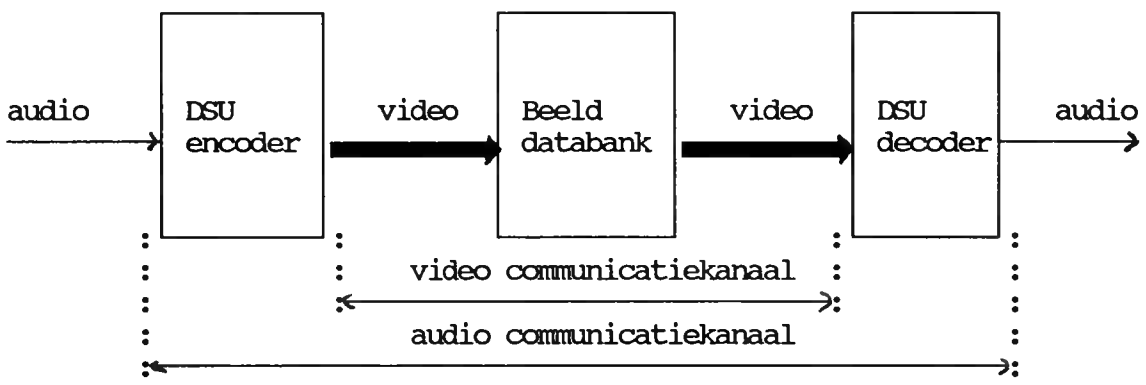


Fig. 3.2 Het DSU systeem als onderdeel van een audio communicatie kanaal

communicatiekanaal is. Deze voorstelling van het DSU systeem is in figuur 3.2 weergegeven. Zoals uit deze figuur is af te leiden is het DSU systeem opgebouwd uit een encoder en een decoder. Het DSU systeem maakt op zijn beurt gebruik van een video communicatiekanaal om audio over te dragen. De beeld databank is, zoals reeds vermeld, een onderdeel van dat video communicatiekanaal.

De DSU encoder kan worden gedefinieerd als een omzetter van audio-informatie naar beeldinformatie. De decoder van het DSU systeem kan dan worden gedefinieerd als een omzetter van beeldinformatie naar audio informatie. De interne structuur van zowel de encoder als de decoder zullen in het onderstaande nader worden besproken.

In figuur 3.3 is de DSU-encoder nader uitgewerkt. Men ziet dat de encoder zal moeten bestaan uit een viertal onderdelen, te weten:

- Omzetting van analoge audio fragmenten naar digitale bemonsteringen. Dit is de eigenlijke broncodering en komt in hoofdstuk 5 ter sprake.
- Tijdelijke data opslag om genoeg gegevens te verzamelen voor het versneld uitlezen.
- Omzetting van digitaal naar beeldsignalen. Dit is de kanaalcodering van het DSU systeem en komt in hoofdstuk 4 ter sprake.
- De besturings eenheid zal zorgdragen voor de interactie tussen de verschillende modules, met besturing van buitenaf.

In figuur 3.4 is de DSU-decoder nader uitgewerkt. Men ziet dat de decoder zal moeten bestaan uit een viertal onderdelen, te weten:

- Omzetting van beeldsignalen naar digitale bemonsteringen. In dit gedeelte van de decoder zullen de aan de kant van de encoder geproduceerde bemonsteringen moeten worden gereconstrueerd vanuit het analoge beeldsignaal.
- Tijdelijke data opslag, gelijk aan de encoder, maar in omgekeerde volgorde gebruikt (eerst snel inlezen, dan langzaam uitlezen).
- Omzetting van digitaal naar een analoog geluidsfragment. Dit is de kanaalcodering van het DSU systeem.

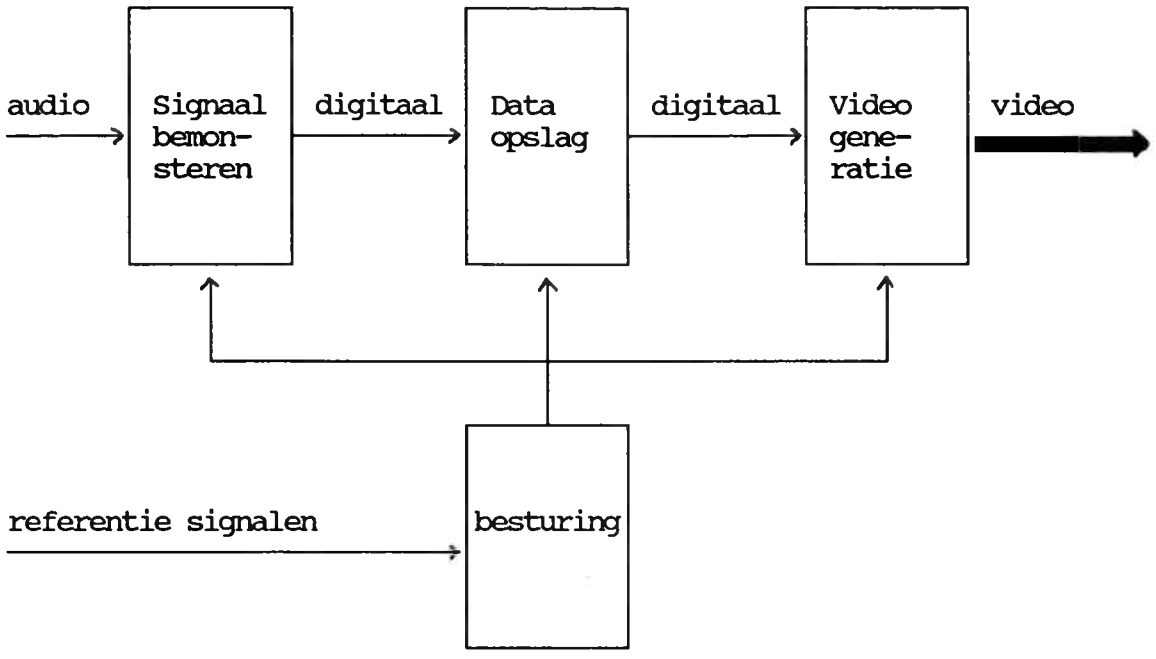


Fig. 3.3 Schematische opbouw van de DSU encoder

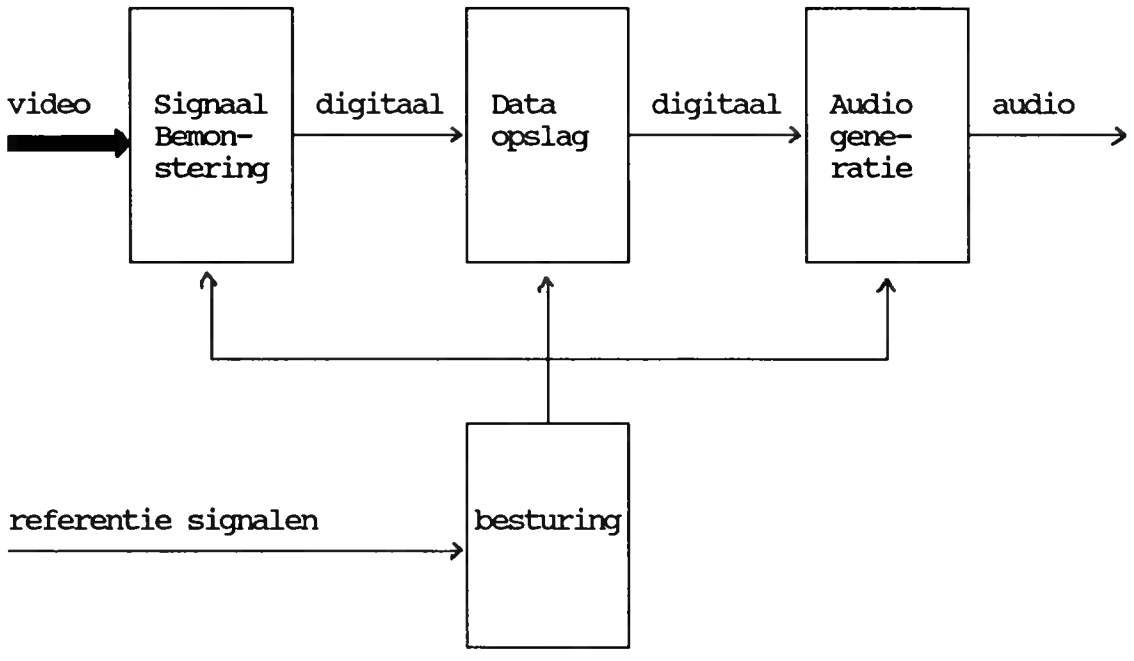


Fig. 3.4 Schematische opbouw van de DSU decoder

- De besturings eenheid zal, net als bij de encoder, zorgdragen voor de interactie tussen de verschillende modules, met besturing van buitenaf.

Nu het principe en de structuur van het DSU systeem zijn besproken, volgen de criteria voor het ontwerp van zo'n systeem.

3.4. Criteria voor het ontwerp van het Digital Speed-Up systeem

Om de bovengenoemde onderdelen van het DSU systeem te kunnen realiseren, zullen er allereerst criteria nodig zijn om een raamwerk vast te stellen voor zo'n ontwerp. De eerste afbakening is reeds in het vorige hoofdstuk genoemd, namelijk de beeld databank. Voor het DSU systeem gelden twee soorten criteria:

- criteria voor de broncodering.
- criteria voor de kanaalcodering.

In eerste instantie moeten de criteria voor de kanaalcodering worden vastgesteld. Dit is nodig omdat de signalen welke via het communicatie kanaal zullen worden verzonden reeds zijn bepaald, te weten signalen die voldoen aan de NTSC videostandaard. Vanuit de hierdoor ontstane criteria kan worden teruggewerkt naar een geschikte broncodering.

3.4.1. Criteria voor de kanaalcodering van het DSU systeem

Zoals hierboven is beschreven moet het uitgangs-signaal van het DSU systeem voldoen aan de NTSC videostandaard. Dit gegeven ligt vast en kan worden samengevat als criterium nummer 1.

Criterium 1:

Het uitgangssignaal van het Digital Speed-Up systeem zal voldoen aan de NTSC videostandaard.

Deze videostandaard wordt in appendix A beschreven.

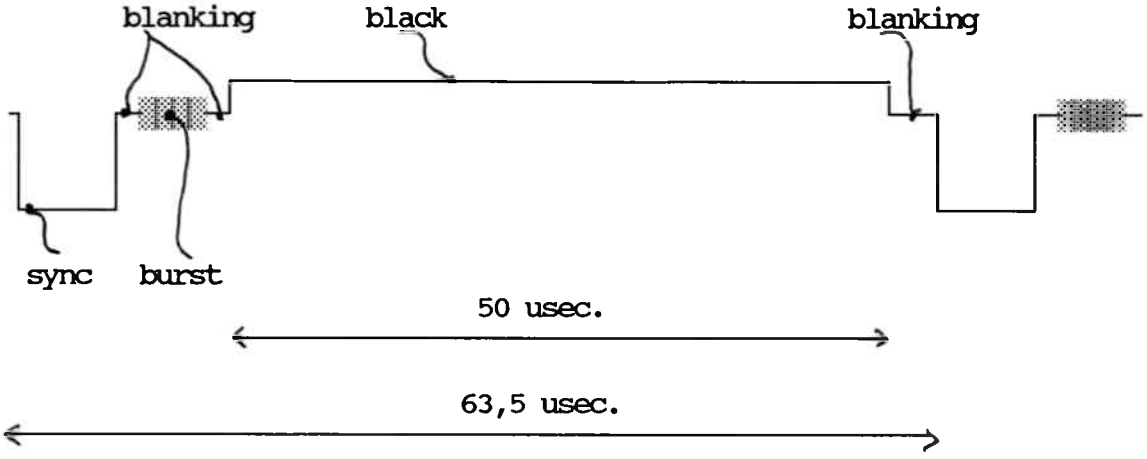


Fig 3.5 De vorm van een video lijn bij een zwart beeld.

In fig 3.5 is te zien hoe een horizontale lijn van zo'n NTSC frame er uit kan zien. De daar afgebeelde beeldlijn is zichtbaar als een zwarte lijn (minimum helderheid) op een beeldbuis, en behoort dus niet tot de lijnen van de vertikale blanking.

Van een lijn, zoals in de figuur afgebeeld, wordt slechts het gedeelte aangegeven met "black" gebruikt voor gecomprimeerd audioinformatie transport. De zojuist genoemde vertikale blanking bevat geen black level en is derhalve niet bruikbaar voor informatie transport. Er is sprake van twee verschillende signaal gebieden binnen een video-signaal.

- Variabel signaalgebied: Het gedeelte van het signaal waar zich beeldinformatie kan bevinden, welke door het DSU systeem gemodificeerd mag worden. Dit gedeelte bestaat uit ongeveer 50 usec. per horizontale lijn van 63,5 usec. en 480 lijnen per video frame van 525 lijnen.
- Constant signaalgebied: Het gedeelte van het signaal welke blanking en synchronisatie informatie bevat, en niet door het DSU systeem gemodificeerd mag worden.

Slechts binnen het variabel signaalgebied zal de DSU encoder gecomprimeerd audio-informatie mogen toevoegen aan het beeldsignaal. Deze beschouwing leidt tot het volgende criterium.

Criterium 2:

Het DSU systeem zal slechts 480 * 50 usec. per beeldsignaal van 525 * 63,5 usec. mogen modificeren ten behoeve van bron informatie transport. De verdeling van deze tijdsduur gedurende het beeldsignaal is tevens vastgesteld.

De technische uitwerking van dit criterium tezamen met het in de volgende paragraaf beschreven criterium nummer 3 is een onderdeel van

het afstudeerwerk en bevindt zich in appendix B. In deze uitwerking is voor een NTSC videoframe, aan de hand van bovenstaande criterium, bepaald of er wel of geen informatie mag worden toegevoegd aan het beeldsignaal.

Binnen het DSU systeem is er sprake van digitale compressie (in de tijd) van opgeslagen audio bemonsteringen. Het is nu mogelijk om uit te gaan van twee transmissie technieken, te weten:

1. Het versturen van de opgeslagen digitale bemonsteringen naar de decoder van het DSU systeem.
2. Het versturen van een, G maal versneld, analoge representatie van het oorspronkelijke audio fragment naar de decoder van het DSU systeem.

In het eerste geval kan worden gesproken van digitaal informatie transport waarbij het variabel signaal gebied van een beeldsignaal voor een digitaal signaal wordt gebruikt, binnen het video communicatiekanaal. Dit proces kan worden vergeleken met de digitale communicatie tussen computers die gebruikmaken van een digitaal signaal via het, voor analoge signalen geschikte, telefoonnet.

In het tweede geval wordt het communicatiekanaal gebruikt voor analoog informatie transport. Hierbij is, ten opzichte van het oorspronkelijke audio fragment, de benodigde transmissietijd verkort ten koste van de benodigde transmissiebandbreedte.

Deze twee technieken zullen voortaan met digitale, respectievelijk analoge transmissie worden aangeduid. In het geval van digitale transmissie gelden geheel andere eisen voor wat betreft reconstructie van de verzonden bemonsteringen dan bij signaal reconstructie in het geval van analoge transmissie. De volgende paragraaf zal ingaan op de eisen ten opzichte van de broncodering in het geval van digitale transmissie.

3.4.2. Criteria voor de broncodering van het DSU systeem

Uit het bovenstaande is af te leiden dat, in het geval van digitale transmissie:

$$r <= B * 2 \qquad (3.1)$$

Met r de "symbol rate" en B de beschikbare bandbreedte. In het geval van NTSC video is het mogelijk om B = 4.5 MHz te kiezen, dus:

$r <= 9 * 10^6$ symbolen per seconde.

Dit geldt voor onafhankelijke symbolen. Bij afhankelijkheid van opeenvolgende symbolen kan deze snelheid omhoog. Als we uitgaan van onafhankelijke symbolen dan is de maximale aantal symbolen dat per videoframe verstuurd mag worden gelijk aan:

$$R_{tot} = t_{eff} * r \qquad (3.2)$$

met t_{eff} als de effectief te gebruiken tijd van een NTSC videoframe. Deze effectieve tijd is gelijk aan de tijd van het variabel signaalgebied. In de vorige paragraaf is deze gegeven als 480 lijnen van elk 50 usec, dus:

t_{eff} = 50 * 480 uSec = 24 msec.

en

R_{tot} <= 216000 samples per NTSC videoframe.

Dit kan worden samengevat in een criterium.

Criterium 3:

Het totaal aantal onafhankelijke signaal bemonsteringen welke per frame kunnen worden verstuurd is begrensd met een theoretisch maximum van 216000.

Dit criterium geeft een indicatie voor de symboolsnelheid. Waar nog niet op in is gegaan is het symbool alfabet. Hiermee wordt niet bedoeld welke symbolen worden gebruikt, maar juist hoeveel verschil-

lende symbolen er kunnen zijn. Dit is afhankelijk van de dynamiek van het communicatiekanaal.

Bij een vaststaand dynamiek zullen transmissie fouten veroorzaakt door storingen op symbolen afhankelijk zijn van het aantal verschillende symbolen (kwantisatie niveaus) in het alfabet. Bij een groter alfabet zullen er meer niveaus zijn met kleiner wordende onderlinge afstand. Hierdoor ontstaat een grotere kans op fouten. De afhankelijkheid tussen het aantal symbolen en de signaal ruis verhouding (S/N) is in het kader van het afstuderen onderzocht en wordt in appendix C beschreven. In deze appendix worden ook 2 verschillende symbool alfabetten vergeleken. Voor wat betreft de keuze van broncodering zal men rekening moeten met het communicatiekanaal en de kanaalcodering. Dit leidt tot het volgende criterium.

Criterium 4:

> Bij het kiezen van een kanaalcodering voor het DSU-systeem zal het symbolen alfabet zo gekozen moeten worden dat een S/N verhouding aan de ingang van de DSU decoder (SNR) van 35 dB, een maximale Bit-Error-Rate (BER) van 10^{-5} veroorzaakt als gevolg van ruis.

De verhouding van 35 dB is als uitgangspunt gekozen omdat de te gebruiken communicatiekanaal (coaxiale kabel) in een "worst case" situatie aan deze voorwaarde behoort te voldoen.

4. INVENTARISATIE VAN MOGELIJK TOEPASBARE BRONCODERINGEN

In dit hoofdstuk wordt nagegaan op welke manier het mogelijk is om een geluidsfragment digitaal te coderen om zodoende een gunstige aanpassing aan het communicatie kanaal te verkrijgen. Dit zal worden gedaan aan de hand van een vergelijking van, tijdens het afstudeeronderzoek, beschikbare systemen. Omdat het ontwikkelen van het DSU systeem een praktisch project betreft is er voornamelijk gekeken naar de mogelijkheden om met bestaande bouwstenen tot een oplossing te komen. Dit is een projectmatige aangelegenheid. De volgende vier systemen zijn onderzocht:

1. CD-Audio
2. Video 8
3. ADPCM volgens CCITT norm G-722
4. ADPCM volgens OKI Semiconductors

De keuze van de oplossing zal uit deze verzameling van mogelijkheden moeten worden gemaakt. De volgende subparagrafen zullen deze vier systemen bespreken.

4.1. CD-Audio digitalisering

Bij de ontwikkeling van CD-audio [Goedhart, Heemskerk, Hoeve] is uitgegaan van geheel andere eisen te aanzien van geluidskwaliteit dan bij de ontwikkeling van het DSU systeem. Bij CD-audio gaat het voornamelijk om opslag en reproductie van "high-quality" audio. Hiermee wordt bedoeld audio informatie met een bandbreedte van 20 KHz en een dynamisch bereik van meer dan 96 dB. Dit is van een geheel andere kwaliteit dan de voor het DSU systeem in paragraaf 3.1 omschreven medium quality. Toch is het zinvol om de bij CD-audio gebruikte techniek te bestuderen. Misschien bestaat er de mogelijkheid om met bestaande (en goedkope) CD componenten een goede oplossing te vinden voor het DSU systeem.

In figuur 4.1 zijn de stappen behorende bij digitalisering voor CD audio weergegeven. CD-audio maakt gebruik van een 16 bits lineaire

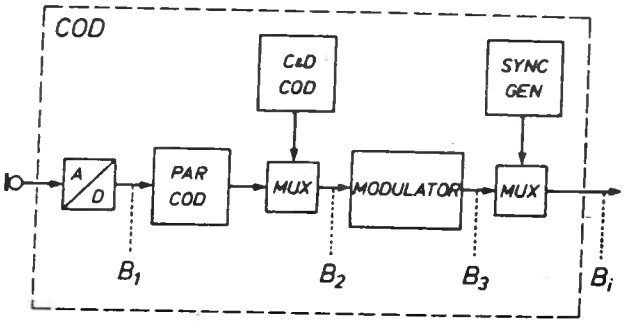


Fig. 2. Het codeersysteem (*COD* in fig. 1). Het systeem is zeer vereenvoudigd weergegeven; in werkelijkheid zijn er b.v. ten behoeve van stereoweergave twee audiokanalen aan de ingang die te zamen via PCM de bitreeks B_1 leveren, en worden de verschillende digitale bewerkingen gestuurd door een 'klok', die niet is aangegeven. De bitstroom B_1 wordt aangevuld met pariteits- en C&D-bits (B_2), gemoduleerd (B_3), en voorzien van synchronisatiesignalen (B_i). *MUX*: multiplexers. Fig. 9 geeft de diverse bitstromen in meer detail.

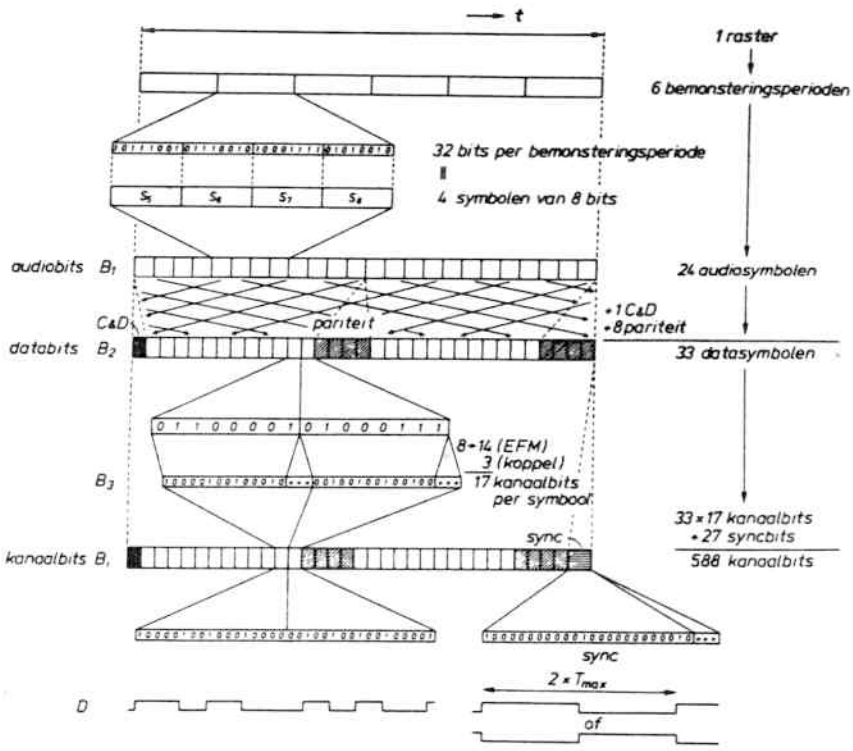


Fig. 9. Bitstromen in het codeersysteem (fig. 2). De informatie is verdeeld in rasters; de figuur geeft een raster van de successieve bitstromen. Er zijn per raster zes bemonsteringsperioden, die elk 32 bits leveren (16 voor elk der beide audiokanalen). De 32 bits zijn in de 'audiobitreeks' B_1 verdeeld in vier symbolen. In de 'databitreeks' B_2 zijn aan de 24 audiosymbolen acht pariteitssymbolen en een C&D-symbool toegevoegd. Ter verspreiding van eventuele fouten zijn de symbolen van verscheidene rasters in B_1 met elkaar vervlochten, zodat de audiosymbolen in één raster van B_2 afkomstig zijn van verschillende rasters in B_1. Door de modulatie worden de acht databits van een symbool van B_2 vertaald in veertien kanaalbits, waaraan nog drie 'koppelbits' worden toegevoegd (B_3). De rasters worden gemarkeerd met een synchronisatiesignaal van de getekende vorm (rechts onder); daarmee is de 'kanaalbitreeks' (B_i) verkregen die gebruikt wordt om de 'master'-plaat te beschrijven, zodanig dat elke '1' een overgang put/dam of dam/put levert (D).

Fig 4.1 Het digitaliserings-proces bij CD-audio.

kwantisatie van het audio signaal met een frequentie van 44,1 KHz. Voor het DSU systeem is slechts dit gedeelte van de codering interessant. Eventueel kan er nog gebruik worden gemaakt van de error coding. In ieder geval bestaat het principe uit lineaire PCM.

Een eerste onderzoek van de gebruikte broncodering bij het CD-audio systeem geeft aan dat het 16 bits lineair PCM ruimschoots zal voldoen aan de eisen gesteld voor het DSU systeem. Er is zelfs sprake van een behoorlijke over-dimensionering. Bij een 10 seconden durende audio fragment van 7 KHz bemonsterd met de horizontale lijnfrequentie van een TV signaal, zullen er in totaal 16 * 15750 * 10 = 2,52 Mbits verzonden moeten worden. Het is de vraag of dit wel mogelijk is binnen één enkele beeldsignaal.

In de vorige hoofdstuk is er bij criterium 3 afgeleid dat er maximaal 216000 onafhankelijke samples kunnen worden verstuurd. Het aantal samples (15750 * 10 = 157500) behorende bij lineaire PCM past dus in principe in een beeldsignaal. In het volgende hoofdstuk zal worden aangetoond dat het niet mogelijk is om, uitgaande van een beeldsignaal, 16 bits per sample zonder fouten te transporteren en reconstrueren uitgaande van de in hoofdstuk 3 genoemde digitale transmissie.

Vooruitlopend op het bovenstaande kan nu al gezegd worden dat 16 bits lineaire PCM niet geschikt is voor het DSU systeem, indien het de bedoeling is alle 16 bits foutloos over te brengen. Wel is het mogelijk om de in de vorige hoofdstuk besproken analoge transmissie te realiseren. In verband met de storingen veroorzaakt tijdens transmissie zal een S/N bij de ontvanger te verwachten zijn van 35 dB. Hierdoor gaat dus een groot gedeelte van het geluidskwaliteit verloren. Een klasieke oplossing om een beter kwaliteit te behalen dan het communicatie kanaal toelaat, is het toepassen van analoge of digitale companders (compressor/expander). De bij Video 8 toegepaste coderings techniek maakt gebruik van beide technieken.

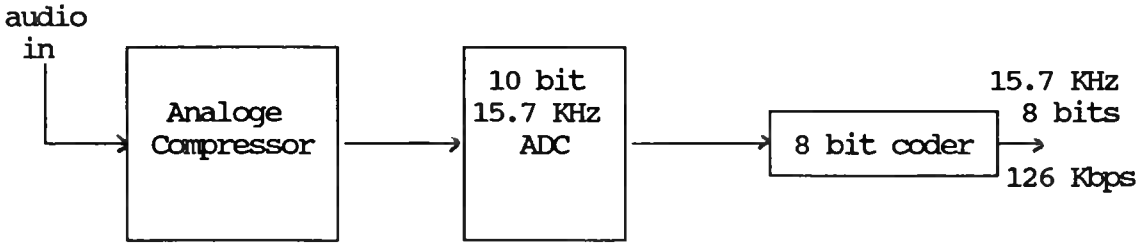


Fig. 4.2 Het Digitaliserings-proces bij Video 8.

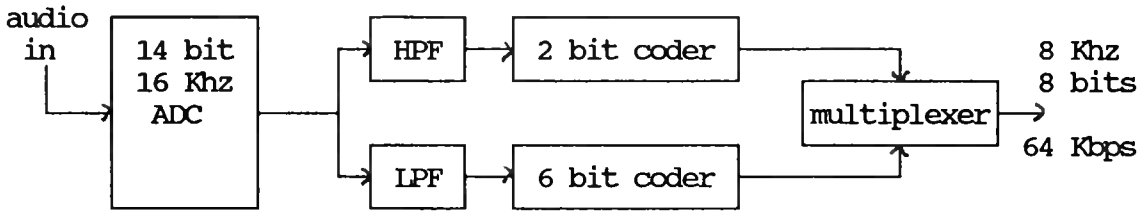


Fig 4.3 Het digitaliserings proces bij CCITT norm G-722.

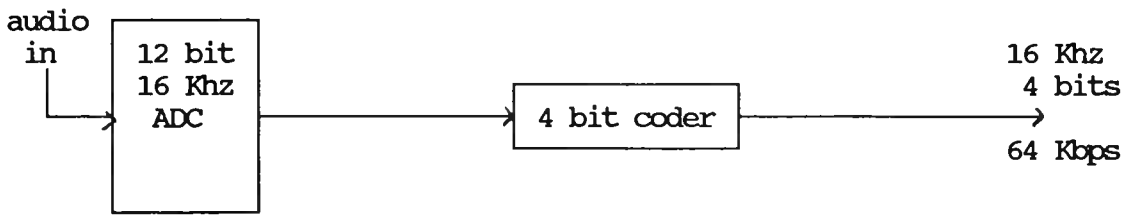


Fig 4.4 Het digitaliserings proces bij OKI Semiconductors.

4.2. Video 8 digitalisering

Bij het Video 8 systeem is gebruik gemaakt van de broncoderings techniek afgebeeld in figuur 4.2. Dit bestaat uit een analoge compressor gevolgd door een IC met daarin onder andere een 10 bits lineaire PCM ADC gevolgd door een 10 naar acht bits digitale compressor. Het resultaat zijn 8 bits niet lineaire PCM bemonsteringen. Bij Video 8 wordt met dit systeem een audio signaal met een bandbreedte van 7 KHz bemonsterd met de horizontale lijnfrequentie van een video signaal. Door gebruikmaking van zowel analoge als digitale compressie krijgt het systeem een dynamisch bereik van 85 dB.

Voor wat betreft geluidskwaliteit voldoet het Video 8 systeem, evenals CD-audio, ruimschoots aan de doelstelling van het DSU systeem. Het versturen van 8 bit bemonsteringen zal echter niet mogelijk blijken indien digitale transmissie wordt toegepast. Althans niet als alle 8 bits gereconstrueerd moeten worden.

In het geval van analoge transmissie lijkt het Video 8 systeem goede mogelijkheden te bieden. Door het gebruik van de bovengenoemde compressie technieken zal de S/N van het audio fragment naar verwachting hoger zijn dan de S/N van 35 dB van het ontvangen signaal. Een uitspraak over de exacte grootte van de audio S/N bij de ontvanger kan echter niet worden gegeven.

4.3. ADPCM digitalisering volgens CCITT aanbeveling G-722

In het kader van de ISDN (Integrated Services Digital Network) norm van 64 Kbps digitale data transport, bestaat er tegenwoordig een CCITT aanbeveling voor het coderen en versturen van 7 KHz audio over een 64 Kbps verbinding. Fig 4.3 geeft aan hoe deze codering tot stand komt.

De ADC bemonsterd het audio met een snelheid van 16 KHz. Dit resulteert in een 14 bits datastroom dat in een hoog en een laag gedeelte wordt gesplitst. Het hoogfrequent deel wordt met een frequentie van 8 KHz door 2 bits gerepresenteerd. Het laagfrequent deel wordt met dezelfde frequentie door 6 bits gerepresenteerd. Dit is een voor

spraak gunstige verdeling omdat zich het meeste informatie in het laagfrequent deel bevindt [Sluyter 1983]. Figuur 4.3 laat zien dat deze twee coderingen worden samengevoegd en zodoende een 64 Kbps datastroom produceren.

Belangrijk bij de G-722 norm is dat er een enorme data-reductie wordt toegepast in de vorm van een ADPCM algoritme. Nu blijven er van een 14 bits bemonstering per saldo 4 bits over. In wezen worden er 8 bits samples met een gehalveerde kloksnelheid geproduceerd. Bij een bemonsterings frequentie van 15750 Hz (horizontale lijnfrequentie) betekent dit bij elkaar 630 Kbits (= 10 sec * 15750 Hz * 4 bits = 10 * 15750 / 2 * 8) data om over te dragen. Aanzienlijk minder dan bij lineaire PCM. Een bijkomend voordeel is dat er, bij een eventuele implementatie, aan de kant van de DSU decoder kan worden volstaan met minder groot geheugen, wat de decoder tevens goedkoper maakt.

4.4. ADPCM digitalisering volgens OKI Semiconductors

Naast de aanbevolen ADPCM algoritme van de CCITT bestaat er tevens een (reeds verkrijgbare) methode ontwikkeld door OKI Semiconductors. Deze methode is ook ontworpen voor gebruik bij 64 Kbps toepassingen. Figuur 4.4 geeft de door OKI gehanteerde techniek weer. In de figuur is te zien dat er van een 12 bits bemonstering die met een frequentie van 16 KHz aan de ADPCM coder wordt aangeboden 4 bits per bemonstering over-blijven. Het resultaat is wat betreft bitrate gelijk aan dat van de ISDN norm. De door OKI gehanteerde methode resulteert echter in een minder goede geluids kwaliteit voor spraak dan de ADPCM algoritme aanbevolen door de CCITT.

5. INVENTARISATIE VAN MODULATIE SYSTEMEN

In het voorgaande hoofdstuk is er aandacht besteed aan de criteria voor de kanaalcodering van het DSU systeem. Het zal nodig zijn om binnen de grenzen van de in hoofdstuk 3 genoemde criteria, een modulatie techniek te vinden die geschikt is voor het transporteren van gecomprimeerd audio informatie. De beperking van criterium 1 houdt in dat er slechts een aantal modulatie technieken behoeven te worden onderzocht. Kern van het verhaal is dat er een kanaal met een bandbreedte van 6 MHz ter beschikking staat binnen een systeem waarbij VSB+C (Vestigial Side Band + Carrier) gemoduleerd wordt.

In dit hoofdstuk zullen een drietal verschillende modulatie technieken worden vergeleken.

Ten eerste het versturen van een signaal waarbij de over te dragen informatie bestaat uit een M_1-tal verschillende kwantisatie niveaus. Hierbij wordt onder andere ingegaan op de bitfoutenkans gedurende transmissie.

Ten tweede het versturen van een in kwadratuur gemoduleerd signaal op de frequentie van de kleuren draaggolf (3.6 MHz). In dit geval gaat het om het versturen van een M_2-tal verschillende kwantisatie niveaus. Het onderzoek richt zich in dit geval ook op de bit foutenkans gedurende transmissie.

Beide bovengenoemde technieken zijn in wezen vormen van digitale transmissie. Hoewel de signaal transmissie zelf altijd een analoog verschijnsel blijft, kan er van digitale transmissie gesproken worden indien men voornamelijk geïnteresseerd is in bijvoorbeeld de bit-error-rate (BER), in plaats van vervorming.

De derde modulatie techniek gaat uit van analoge transmissie. Hiermee wordt bedoeld transmissie van een signaal waarbij alleen analoge signaal reconstructie van het getransporteerde signaal van belang is.

Voordat kan worden ingegaan op de verschillende modulatie technieken is het noodzakelijk om eerst een beschrijving te geven van de gehanteerde definitie van BER.

5.1. Bit-Error-Rate bij digitale transmissie

Bij de in dit verslag gehanteerde begrip BER behorende bij meer-niveau transmissie wordt uitgegaan van de in [Carlson blz. 400 ev.] vermelde functie:

$$P_e = {}^{2(M-1)}/_{M} * Q({}^{A}/_{2\sigma}) \qquad (5.1)$$

met:

- M = Het aantal signaal niveau's
- P_e = Transmissie foutkans
- ${}^{A}/_{2}$ = Halve amplitude
- σ = Standaard deviatie van gaussisch ruis.

Voor de formule behorende bij Q(k) wordt verwezen naar [Carlson blz. 665]. Uitgaande van gaussisch ruis en fouten tussen aangrenzende niveaus (1-niveau fouten, Gray code) geldt tevens:

$$({}^{A}/_{2\sigma})^2 = {}^{3}/_{(M^2-1)} * SNR \qquad (5.2)$$

Deze voorstelling maakt het eenvoudig om van een gegeven SN_R en een gekozen aantal niveau's M, een berekening te maken voor wat betreft de te verwachten transmissie fouten veroorzaakt door gaussisch ruis.

5.2. Modulatie van een meer-niveau signaal

Bij het gebruik van meer-niveau transmissie zullen, binnen het videosignaal, op de plaats van de luminantie, discrete amplitude waarden de verschillende niveau's voorstellen. Deze waarden zullen equidistant worden verdeeld tussen het meest heldere en het minst heldere deel van het luminantie signaal. Dit luminantie deel bestrijkt 70% van de totale amplitude van het videosignaal. Voor het luminantie deel geldt een SN_R van 35 dB als een realistische waarde. Dit getal zal daarom worden gebruikt voor berekening van de BER.

Tabel 5.1 geeft een overzicht van een aantal verschillende combinaties van broncoderingen en kanaalcoderingen uitgezet tegen het aantal bits die kunnen worden getransporteerd bij een kanaal bandbreedte van 4.2 MHz en de theoretische BER. De in de tabel afgedrukte waarden van getransporteerde bits zijn (met uitzondering van tele-text) theoretische maxima bij de opgegeven bandbreedte.

De in tabel 5.1 afgedrukte waarden van getransporteerde Kbits/frame kunnen worden berekend door toepassing van de formules 3.1 en 3.2 van paragraaf 3.4.2. Samengevoegd vormen zij de onderstaande relatie:

$$R_{tot} = t_{eff} * B * 2 \quad \text{symbolen/frame.}$$

Om hieruit het aantal Kbits/frame te berekenen moet het aantal bits per symbool bekend zijn. Dit is gelijk aan de $^2\log$ van het aantal transmissie niveau's M. 16-niveau transmissie draagt per getransporteerde symbool 4 bits over. De formule om het aantal Kbits/frame te berekenen ziet er dan als volgt uit:

$$R_{frame} = (t_{eff} * B * 2 * {}^2\log M) / 1000 \quad \text{kbits/frame.} \qquad (5.4)$$

Door het invullen van de constanten:

$t_{eff} = 24$ msec.
$B = 4.2$ MHz.

volgt hieruit:

$$R_{frame} = 201{,}6 * {}^2\log M \quad \text{kbits/frame.} \qquad (5.5)$$

De in tabel 5.1 afgedrukte waarden van BER's zijn berekend door toepassing van formules 5.1 en 5.2. Tezamen vormen zijn de onderstaande relatie voor de BER uitgedrukt in de Q functie, de SN_R en het aantal data niveau's M:

$$P_e = {}^{2(M-1)}/_{M} * Q(\mathrm{sqrt}({}^{3}/_{(M^2-1)} * SNR)) \qquad (5.6)$$

	getransporteerde bits bij 4.2 MHz transmissie bandbreedte (Kbits/Frame)	Bit Error Rate (Theoretisch)	gemiddelde fouten per frame (Bits/Frame)
Full channel Tele-Text	140	$2,00 * 10^{-5}$	2,8
4 niveau modulatie	403	$2,25 * 10^{-3}$	906
8 niveau modulatie	604	$5,83 * 10^{-2}$	35000
16 niveau modulatie	806	$7,03 * 10^{-2}$	56000
2*2 niveau kwadratuur modulatie	403	$2,00 * 10^{-5}$	8
2*4 niveau kwadratuur modulatie	806	$2,25 * 10^{-3}$	1812

Tabel 5.1 Vergelijking van enige digitale modulatie technieken.
Bij de berekening van het aantal getransporteerde Kbits/Frame is uitgegaan van formule 5.5:

$$R_{frame} = 201.6 * {}^{2}\log M \quad \text{Kbits/frame}$$

Alleen bij Full Channel Tele-Text is er op een andere manier R_{frame} berekend, te weten:

500 lijnen van 35 bytes/lijn van 8 bits = 140 Kbits.

Bij de berekening van de Bit Error Rate (BER) is gebruik gemaakt van SNR = 35 dB, de Q functie van formule 5.2 en formule 5.6:

$$Pe = 2(M-1)/M * Q(\mathrm{sqrt}(3/(M^2-1) * SNR))$$

Tele-text, dat gebruik maakt van binaire (2-niveau) transmissie, is als voorbeeld toegevoegd aan het tabel. Te zien is dat het invoeren van meerdere niveau's de foutkans danig verhoogd. Zo zal 4 niveau transmissie een (ongeveer) 100 maal grotere foutkans opleveren dan binaire transmissie.

5.3. Modulatie van een meer-niveau signaal in kwadratuur

Ten opzichte van de in de vorige paragraaf genoemde modulatie techniek is door middel van kwadratuur modulatie (QAM Quadrature Amplitude Modulation) 2 keer zo veel binaire informatie te transporteren. Bij video worden de kleuren kwadratuur gemoduleerd. Dit geeft de mogelijkheid om, gebruikmakend van de kleuren draaggolf, kwadratuur gemoduleerde gegevens te versturen.

Bij NTSC video is echter de bandbreedte van de beide kleursignalen U en V beperkt tot respectievelijk 1,5 MHz en 0,5 MHz. Door deze bandbreedte beperking kan er slechts weinig data worden overgedragen. Onderzocht is of het mogelijk zou zijn om de gehele luminantie bandbreedte van 4.2 MHz te gebruiken voor kwadratuur modulatie.

Tabel 5.1 geeft voor twee soorten QAM de bijbehorende waarden van het maximum aantal te transporteren bits en de BER indien 4.2 MHz bandbreedte wordt toegepast. Uit dit tabel is te zien dat QAM een kleinere BER heeft dan een vergelijkbare meer-niveau transmissie besproken in paragraaf 5.2. Hieruit is af te leiden dat QAM een betere modulatie techniek is voor wat betreft de foutbestendigheid bij gelijke data overdracht, of bij gelijke foutkans een grotere hoeveelheid data kan overdragen.

Tijdens het onderzoek is gebleken dat het QAM systeem met vergrootte bandbreedte niet goed bruikbaar is. Dit is een praktisch probleem en vindt zijn oorzaak in het feit dat gebruikelijke NTSC decoders het breedbandige QAM signaal niet juist kunnen decoderen. Indien er zich ook maar één video demodulator in het communicatie kanaal bevindt, is het niet meer mogelijk om het breedbandige QAM van de DSU encoder naar decoder over te dragen. De toepassing van één zo'n demodulator is

reeds bekend, de beeld databank maakt gebruik van demodulatoren waarbij het composiet video-signaal bestaande uit Y,U en V wordt omgezet naar R,G en B signalen. Omdat het breedbandige QAM signaal geen Y informatie bevat zullen de R,G en B signalen niet meer de juiste digitale waarden voorstellen als het oorspronkelijke breedbandig QAM.

5.4. Analoge modulatie van het tijd-gecomprimeerd audio signaal

In de vorige 2 paragrafen is behandeld hoe het mogelijk is om een bepaald aantal bits via een TV kanaal te versturen. Een fundamenteel andere oplossing bestaat uit het rechtstreeks versturen van het in de tijd gecomprimeerd oorspronkelijke analoge audiofragment. Hierbij wordt het digitaliseringsproces alleen gebruikt voor versnelling en vertraging van het signaal.

Het grote voordeel van analoge kanaalcodering schuilt in de eenvoud van de hardware aan de kant van de decoder. Het binnenkomend signaal behoeft slechts gereconstrueerd te worden door middel van bemonsteringen (A/D omzetting) om daarna deze bemonstering met een vertraagde snelheid via een D/A omzetter gevolgd door een basisband filter tot het originele geluidsfragment terug te brengen. De exacte aanvang van de bemonsteringen aan de kant van de decoder behoeft niet synchroon te lopen met die van de encoder. Ook de snelheid van de bemonsteringen behoeft bij de decoder niet gelijk te zijn aan die van de encoder. De compressiefactor G (zie paragraaf 3.1) van het DSU systeem moet echter bij de encoder en de decoder gelijk zijn, dit is voldoende voorwaarde voor wat betreft de bemonsteringsfrequenties van het DSU systeem om het analoog gecomprimeerd audiofragment te kunnen expanderen tot het oorspronkelijke audiofragment.

In het geval van analoge transmissie is het van belang om de consequenties van storingen op het gecomprimeerd audiofragment te extrapoleren naar een 400 maal geëxpandeerde tijdsschaal. De effecten van bijvoorbeeld gaussisch ruis, zijn onafhankelijk van de compressie. Na expansie in de tijd van de bemonsteringen van het gecomprimeerd audio zal er nog altijd een S/N van 35 dB resteren. Deze eigenschap

zal van belang zijn bij het combineren van bron- en kanaalcoderingen welke besproken worden in het volgende hoofdstuk.

5.5. Conclusie

In dit hoofdstuk zijn methoden voor zowel digitale als analoge transmissie ter sprake gekomen.

Bij een algemeen geaccepteerde BER van 10^{-5} zal bij de besproken digitale transmissie methoden alleen full channel tele-text (2-niveau transmissie) voldoen. Daarna volgen in oplopende volgorde van BER:

- 4-niveau modulatie (10^{-3})
- 8-niveau modulatie (10^{-2})
- 16-niveau modulatie (10^{-1})

Bij analoge transmissie van een signaal moet rekening worden gehouden met een SN_R van 35 dB voor bemonsteringen te behoeve van signaal reconstructie.

6. DE VOOR HET DSU SYSTEEM GEKOZEN CODERING EN TRANSMISSIE TECHNIEK

In de hoofdstukken 4 en 5 zijn zowel broncoderings systemen als kanaalcoderings systemen aan de orde gekomen. Om nu een systeem oplossing te bieden is het noodzakelijk om de verschillende mogelijke combinaties onderling te vergelijken en zo mogelijk in een volgorde van optimale oplossingen te zetten. Aan de hand van deze vergelijking zal de uiteindelijk gekozen oplossing worden gemotiveerd.

Tabel 6.1 geeft een overzicht van een aantal modulatie technieken gecombineerd met broncoderingen en de daarbij behorende tijdsduur en kwaliteit van het ongecomprimeerd audio-fragment bij de DSU decoder.

De kwaliteit is uitgedrukt in dB voor het geval van foutloos digitale transmissie, er is hierbij geen rekening gehouden met kwaliteits-verlies veroorzaakt door transmissie fouten. Alleen in het geval van analoge transmissie is wel rekening gehouden met de bij de ontvanger te verwachten SN_R.

De tijdsduur van het ongecomprimeerde audiofragment is uitgedrukt in seconden per frame en wordt berekend door de waarde van Kbits/frame, behorende bij een kanaalcodering, (tabel 5.1) te delen door het aantal Kbits/seconde behorende bij een broncodering. Bijvoorbeeld:

> In het geval van lin PCM met 16 bits bemonsteringen en een bemonsterings frequentie van 15750 Hz worden er 252 Kbits per seconde bemonsterd. Bij de in tabel 5.1 beschreven 4-niveau modulatie betekend dat (403 / 252) = 1,6 seconden/frame.

Dit voorbeeld maakt het aannemelijk dat algemeen geldt:

$$T_{frame} = R_{frame} / (F_S * B_S / 1000) \quad \text{seconden/frame} \qquad (6.1)$$

met:

F_S = bemonsteringsfrequentie

B_S = aantal bits per bemonstering

Waarbij de waarde van R_{frame} kan worden afgelezen van tabel 5.1.

	Over te dragen 3.5 Khz audio (seconden/frame)	Over te dragen 7 KHz audio (seconden/frame)	S/N van audio aan de uitgang van de decoder (dB)
Lin PCM + Tele-Text *	1,1	0,6	
Lin PCM + 4 niveau	3,2	1,6	96
Lin PCM + 16 niveau	6,4	3,2	
Log PCM + Tele-Text *	2,2	1,1	
Log PCM + 4 niveau	6,4	3,2	85
log PCM + 16 niveau	12,8	6,4	
ADPCM + Tele-Text *	4,4	2,2	
ADPCM + 4 niveau	12,8	6,4	
ADPCM + 16 niveau	25,6	12,8	
Log PCM + analoog	28	14	70
Lin PCM + analoog	28	14	35

Tabel 6.1 Vergelijking van enige combinaties van bron- en kanaalcoderingen. De waarden van de SNR van hoorbaar audio aan de kant van de decoder is voor alle gevallen geschat. De berekening van seconde/frame is gedaan volgens:

$$T_{frame} = R_{frame} / (F_s * B_s / 1000)$$

Waarbij de waarde van R_{frame} kan worden afgelezen van tabel 5.1.

* Bij het gebruik van Tele-Text als kanaalcodering wordt bedoeld Full Channel Tele-Text zoals beschreven bij figuur 5.1.

In de volgende paragrafen komen de 3 onderstaande broncoderingen ter sprake:

- Lineaire puls code modulatie (Lin PCM)
- Logaritmische puls code modulatie (Log PCM)
- Adaptieve differentieel puls code modulatie (ADPCM)

6.1. Lin PCM gecombineerd met twee kanaalcoderingen

Bij het gebruik van Lin PCM is het volgens tabel 6.1 mogelijk om het gecodeerde audio zowel **analoog** als **digitaal** te versturen.

In het geval van **digitale** modulatie is het mogelijk om een audio kwaliteit van 96 dB [Goedhart, Heemskerk, Hoeve] over te zenden van de encoder naar de decoder. Tabel 6.1 laat zien dat de daarbij behorende ongecomprimeerde audio tijdsduur welke per frame kan worden verstuurd zeer klein is. Om deze reden valt digitale modulatie gecombineerd met Lin PCM af.

In het geval van **analoge** modulatie zal het signaal aan de uitgang van de DSU encoder een SN_T (signaal ruis verhouding bij transmissie) van 96 dB hebben. Bij de ontvanger zal dat zijn gereduceerd tot een SN_R van 35 dB voor het gecomprimeerde geluidsfragment. Indien wordt verondersteld dat de ruis gaussisch verdeeld is, zullen de bemonsteringen van het analoog signaal een gemiddelde SNR van 35 dB bezitten. Bij het ongecomprimeerd weergeven van het geluidsfragment zal het fragment en SNR van 35 dB hebben. Het is dus wel mogelijk om met behulp van analoge transmissie, Lin PCM te versturen, mits genoegen wordt genomen met een audio kwaliteit van 35 dB.

6.2. Log PCM gecombineerd met twee kanaalcoderingen

Bij het gebruik van Log PCM is het net als bij lineaire PCM mogelijk om zowel **analoge** als **digitale** transmissie toe te passen.

In geval van **digitale** transmissie geeft tabel 6.1 aan dat het mogelijk is om een twee keer zo lange geluidsfragment te versturen als in het

geval van lin PCM. Toch is dit geluidsfragment nog maar half zo lang als ADPCM gecombineerd met digitale transmissie. Dit komt omdat het ADPCM algoritme slechts 4 bits per bemonstering gebruikt om de bemonsterde audio informatie te transporteren. Voor wat betreft digitale transmissie verdient ADPCM in ieder geval de voorkeur.

In geval van **analoge** transmissie blijkt dat Log PCM beter geschikt is dan lin PCM voor wat betreft de kwaliteit van het overgedragen geluidsfragment. Dit komt door de amplitude dichtheid van spraaksignalen [Sluyter 83] gecombineerd met de logaritmische compressie karakteristiek van de bemonsterde waarden van het geluidsfragment. Hierdoor is een SNR van het ongecomprimeerde audio bij de DSU decoder haalbaar die 2 keer groter (in dB) is dan de SN_R, Dit betekent een theoretische kwaliteit van 70 in plaats van 35 dB voor de lagere amplituden van het oorspronkelijk signaal en 35 dB voor de hogere amplituden. De verloop van de compressie-karakteristiek is verder beschreven in [Philips Elcoma].

Gegeven de voordelen van analoge transmissie boven die van digitale transmissie lijkt de Log PCM met analoge transmissie de beste oplossing te zijn voor het DSU systeem. Toch zal deze oplossing niet kunnen worden gebruikt. Tijdens het onderzoek naar de mogelijkheden van codering en transmissie is gebleken dat er reeds een patent [Pargee 31-1-84] bestaat op het gecomprimeerd versturen van 8 bits audio bemonsteringen binnen een NTSC videoframe, gebaseerd op analoge signaal reconstructie bij de ontvanger.

6.3. ADPCM gecombineerd met twee kanaalcoderingen

Bij het gebruik van ADPCM als broncodering is het noodzakelijk de digitale waarden, welke het audio fragment representeren, naar de DSU encoder te versturen. Het zal dus noodzakelijk zijn om digitale transmissie te gebruiken.

Zoals tabel 6.1 laat zien is het in principe mogelijk om met ADPCM de in de doelstellingen (zie hoofdstuk 3) vereiste tijdsduur en bandbreedte van het ongecomprimeerd geluidsfragment te behalen. Hiervoor

is 16-niveau transmissie noodzakelijk. De in de vorige hoofdstuk te vinden tabel 5.1 geeft echter aan dat de foutkans wel erg groot is bij 16-niveau transmissie. Het ADPCM algoritme moet dan wel zeer foutbestendig zijn. Bij de ontwikkeling van het DSU systeem is deze methode in eerste instantie toegepast. De gebruikte ADPCM IC's van de firma OKI Semiconductors bleken echter bij een audio bandbreedte van 7 KHz niet goed bestand tegen zoveel transmissiefouten. De handleiding [OKI] behorende bij deze IC's geeft echter géén beschrijving van de foutbestendigheid van de door OKI gebruikte algoritme. In appendix F is een artikel opgenomen over de foutbestendigheid van het OKI ADPCM algoritme bij 3.5 KHz audio. Dit artikel geeft aan dat bij een 4 bits block [zie OKI] een block-error-rate (BLER) van 10^{-2} als "performance threshold" voor de geluidskwaliteit geldt.

De CCITT aanbeveling G-722 maakt eveneens gebruik van een ADPCM algoritme. De ontwikkelafdeling van Philips, welke op dit ogenblik het algoritme implementeert voor een DSP (Digital Signal Processor) van Philips stelt dat het CCITT algoritme eveneens bestendig is tegen fouten in de orde van 10^{-2}.

Omdat in eerste instantie gebruik wordt gemaakt van het OKI algoritme zal er met deze combinatie van modulatie techniek en broncodering niet aan de doelstellingen kunnen worden voldaan. Er bestaan twee alternatieven, te weten:

- De bandbreedte van de over te dragen geluidsfragment verkleinen.
- De tijdsduur van de over te brengen geluidsfragment verkleinen.

Bij de ontwikkeling van het DSU systeem is gekozen voor het verkleinen van de bandbreedte naar 3.5 KHz bij gebruikmaking van de 4-niveau transmissie techniek. Hierdoor kan ongeveer 12 seconden geluid met een bandbreedte van 3.5 KHz worden overgedragen.

6.4. Systeemkeuze momenten en Conclusies

Tijdens de ontwikkeling van het DSU systeem zijn achtereenvolgens de volgende combinaties van bron- en kanaalcoderingen overwogen en eventueel gedeeltelijk ontwikkeld.

1. In augustus 1987 is als eerste besloten log PCM met analoge modulatie te gebruiken voor het DSU systeem. Dit is niet doorgegaan wegens het bestaan van een patent op deze implementatie.
2. In october 1987 is besloten het ADPCM algoritme van de CCITT tezamen met breedbandige 2*4-niveau QAM te implementeren.
3. In september 1987 is de keuze gevallen op het ADPCM algoritme van de firma OKI Semiconductors, omdat was gebleken dat het CCITT algoritme niet verkrijgbaar was. Tevens is besloten om 16-niveau modulatie parallel aan 2*4-niveau QAM te ontwikkelen.
4. In november 1987 is de mogelijkheid van breedbandige QAM wegens praktische problemen komen te vervallen.
5. Begin januari 1988 is na de eerste ontwikkelingen besloten om af te stappen van 16-niveau modulatie en door te gaan met de ontwikkeling van 4-niveau modulatie. 16-niveau modulatie bleek al in een testomgeving onder laboratorium omstandigheden zoveel transmissie fouten op te leveren dat het gereproduceerde audio voor het gehoor niet gelijk klonk aan het oorspronkelijke audiofragment.

Geconcludeerd kan worden dat de oplossing van log-PCM met analoge modulatie het meest eenvoudig te implementeren zal zijn, en waarschijnlijk het meest foutvrij. Het is ook het enig systeem waarvan (op dit moment) kan worden gezegd dat het nog voldoet aan de doelstelling om 7 KHz audio over te dragen. Jammer genoeg bestaat er reeds een patent op deze implementatie.

Mede wegens de ongeschiktheid van (relatief) breedbandig QAM binnen de NTSC video norm is er uiteindelijk gekozen voor het 4 niveau modulatie systeem gecombineerd met ADPCM volgens de CCITT aanbeveling G-722.
Wegens gebrek aan een DSP (Digital Signal Processor) met de G-722 aanbeveling is er voor het eerste ontwerp van het DSU systeem gekozen

voor de algoritme van OKI semiconductors. De hiervoor benodigde IC's zijn eenvoudig verkrijgbaar. Er wordt in dit geval gekozen voor een geluidsbandbreedte van 3.5 KHz met een duur van 10 seconden.

7. TECHNISCHE REALISATIE VAN HET DSU-SYSTEEM

Inmiddels is het DSU systeem, gebruikmakend van ADPCM broncodering gecombineerd met 16-niveau kanaalcodering in ontwikkeling. Gedeeltelijk door ervaringen opgedaan gedurende tests is door Philips besloten om af te zien van 16-niveau kanaalcodering. In plaats daarvan is gekozen voor de in hoofdstuk 5 genoemde, 4-niveau kanaalcodering. Dit wordt dan nog wel gecombineerd met ADPCM broncodering.

Een gehele beschrijving van de ontwikkeling van het DSU systeem is te uitgebreid om in het kader van dit verslag te behandelen. Wat in dit hoofdstuk wel aan de orde komt, zijn die gedeelten van de ontwikkeling waaraan ik heb (mee)gewerkt zoals:

- De omzetting van in een geheugen beschikbare **digitale** representatie van één NTSC audiobeeld (beeldsignaal welke gecomprimeerd audio bevat) naar een **analoog** audiobeeld. Deze omzetting vindt plaats bij de encoder van de DSU systeem.
- De omzetting van een **analoog** audiobeeld naar **digitale** waarden van 4 bits (16-niveau) per bemonstering. Deze omzetting vindt plaats bij de decoder van het DSU systeem.
- Het ontwerpen van een schakeling ten behoeve van de aansturing van een kwadratuur modulator.
- Het ontwerpen en schrijven van het "DSU-Utility" communicatieprogramma.

7.1. Omzetting van digitaal NTSC frame naar een analoog audiobeeld

Bij het omzetten van een digitaal NTSC frame naar een analoog audiobeeld (figuur 7.1) wordt van de volgende gegevens uitgegaan:

- De digitaal representatie van het NTSC frame bestaat uit een voorgedefinieerd NTSC frame (zie appendix B) met daarin, op de juiste plaatsen, ADPCM bemonsteringswaarden toegevoegd. Deze digitale representatie bestaat uit 210000 bemonsteringen. Dit aantal ontstaat door het aantal horizontale lijnen per NTSC frame

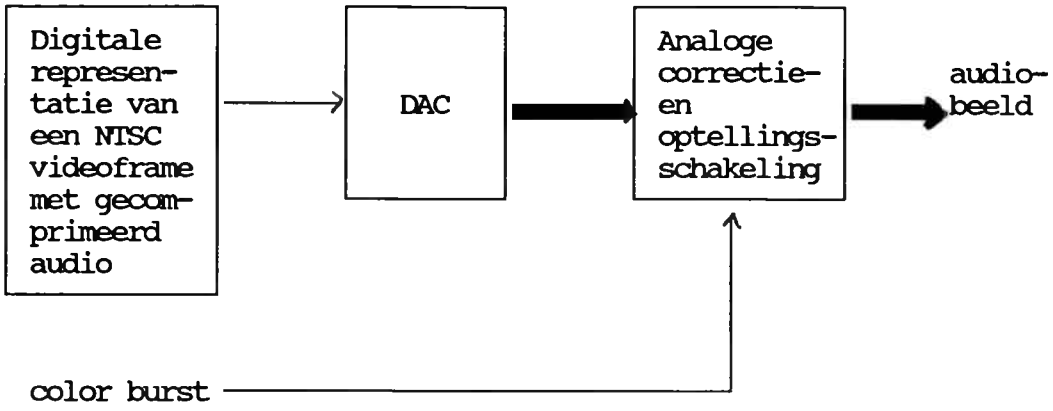


Fig. 7.1 De omzetting van digitaal gegenereerd NTSC frame gevuld met gecomprimeerde audiobemonsteringen tot een analoog audiobeeld.

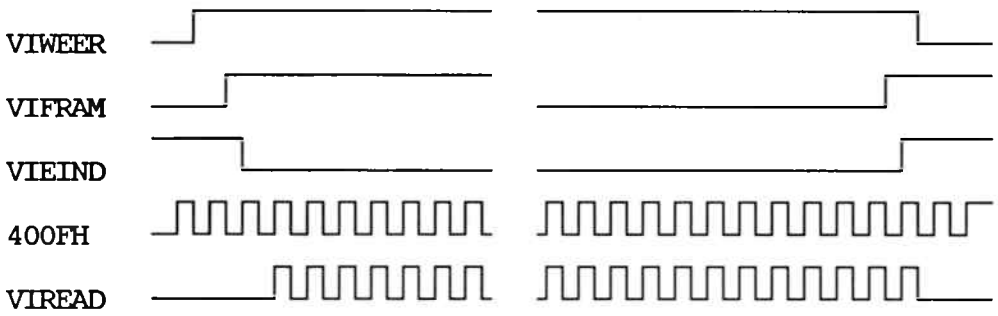


Fig. 7.2 Het verloop van een aantal besturingssignalen van de DSU encoder ten behoeve van de generatie van een analoog audiobeeld.

(525) te vermenigvuldigen met het aantal bemonsteringen per lijn (400).

- De digitale waarden worden in een onafgebroken stroom, op de juiste snelheid, aan de DAC aangeboden.
- Er wordt bij deze implementatie gebruik gemaakt van 16-niveau kanaalmodulatie.

In deze paragraaf is zowel de correctie- en optellings-schakeling van figuur 7.1 aan de orde alsmede de (niet getekende) besturingssignalen van de DAC.

Ten eerste de analoge correctie- en optellings-schakeling. Deze schakeling heeft drie functies, te weten:

- Het toevoegen van de color burst aan het uitgangssignaal van de DAC.
- Het verlagen van de offset van het signaal afkomstig van de DAC van 0,5 Volt naar 0 Volt.
- Het verkleinen van de seinspan van het signaal afkomstig van de DAC van 1,43 Volt naar 1.0 Volt.

Het signaal aan de uitgang van de correctie- en optellings-schakeling zal na deze drie bewerkingen voldoen aan de in appendix A genoemde video-norm, waarbij het audiobeeld voorzien van color burst, een offset van 0 volt heeft een seinspan van 1 Volt heeft.

In appendix D bevindt zich een schema van de analoge schakeling die deze functie vervult. De schakeling heeft twee invoer signalen: de color burst en het uitgangs-signaal van de DAC.

Ten tweede de besturingssignalen van de DAC. Deze besturingssignalen zijn logische TTL (Transistor Transistor Logic) signalen en worden gerepresenteerd door de waarden '0' en '1'. De genoemde signalen zijn in figuur 7.2 getekend en worden als volgt gedefinieerd:

- VIWEER Video Weergave. Dit signaal geeft aan dat de encoder van het DSU systeem een audiobeeld moet afgeven aan de

uitgang. Dit signaal blijft tijdens de weergave van een audiobeeld altijd '1'.

- VIFRAM Video Frame. De overgang van '0' naar '1' geeft het begintijdstip van een audiobeeld aan. Dit signaal gaat van '1' naar '0' ten tijde van het einde van het eerste video field (halve frame). Deze '1'-'0' overgang is niet in figuur 7.2 weergegeven.
- 400FH Dit signaal is gelijk aan 400 maal de horizontale lijnfrequentie.
- VIEIND Video Eind. Dit signaal is '0' tijdens het genereren van een analoog audiobeeld.
- VIREAD Video Read. Indien VIWEER hoog is en VIEIND laag, heeft het VIREAD signaal een frequentie gelijk aan 400FH. Dit signaal stuurt de DAC aan, en bepaalt tevens te snelheid waarmee het digitale audiobeeld uit het geheugen wordt gelezen.

De signalen VIWEER, VIFRAM en 400FH zijn afkomstig van de besturing van het DSU systeem. De signalen VIEIND en VIREAD zijn van de eerstgenoemde drie signalen afgeleid. De (logische) schakeling die VIEIND en VIREAD creëert is in appendix D weergegeven.

7.2. Omzetting van een audiobeeld naar digitaal ADPCM waarden

Bij de decoder van het DSU systeem zal het binnenkomende audiobeeld moeten worden bemonsterd om de oorspronkelijke digitale ADPCM waarden te kunnen reconstrueren. De aldus bemonsterde waarden worden vervolgens in een geheugen geplaatst voor verdere bewerking. Figuur 7.3 laat zien dat het audiobeeld voor bemonstering door een analoge correctie-schakeling moet worden verwerkt.

In deze paragraaf wordt zowel deze analoge correctie-schakeling als de (niet getekende) besturingssignalen van de ADC van de DSU decoder beschreven.

Ten eerste de analoge correctie-schakeling. Deze vervult de volgende twee functies:

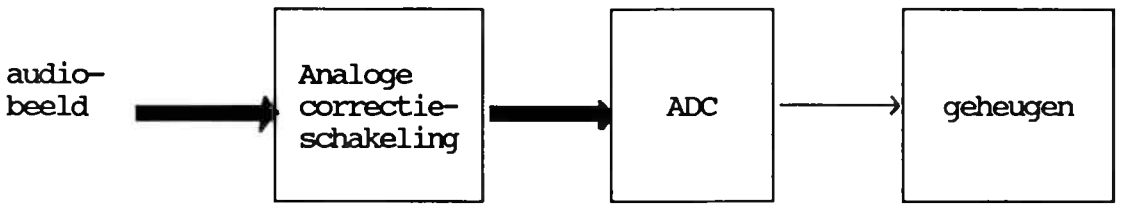


Fig. 7.3 De omzetting van een analoog audiobeeld tot de oorspronkelijk verstuurde digitale waarden. De niet getekende besturingssignalen van de ADC voor de bemonsterings tijdstippen worden afgeleid van de color carrier frequentie.

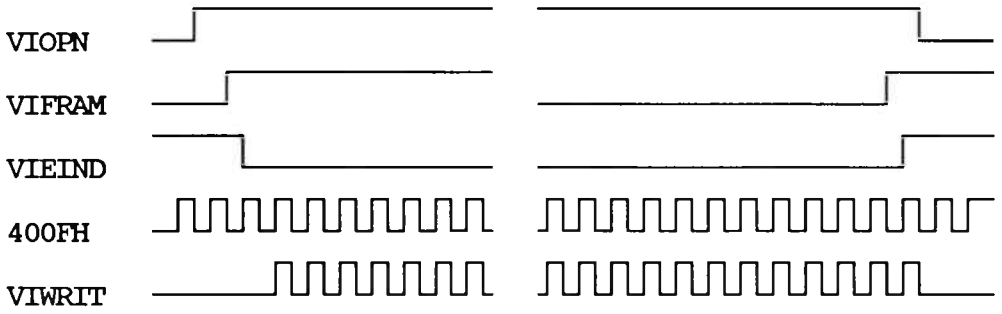


Fig. 7.4 Het verloop van een aantal besturingssignalen van de DSU encoder ten behoeve van de generatie van een analoog audiobeeld.

- Het verhogen van de offset van het binnenkomend signaal van 0 Volt naar 0,5 Volt.
- Het vergroten van de seinspanning van het binnenkomend signaal van 1,0 Volt naar 1,43 Volt.

Het uitgangssignaal van deze schakeling wordt rechtstreeks aan de ADC, afgebeeld in fig 7.3, aangeboden. De schema van de correctie schakeling is in appendix D weergegeven.

Ten tweede de besturingssignalen van de ADC. Deze besturingssignalen zijn in figuur 7.4 getekend en worden als volgt gedefinieerd:

- VIOPN Video Opname. Dit signaal geeft aan wanneer een binnenkomend audiobeeld moet worden ingelezen in het geheugen. Dit signaal is analoog aan het signaal VIWEER bij de encoder.
- VIFRAM Dit signaal is gelijk aan VIFRAM bij de encoder.
- 400FH Dit signaal is gelijk aan 400FH bij de encoder.
- VIEIND Dit signaal is gelijk aan VIEIND bij de encoder.
- VIWRIT Video Write, Indien VIOPN hoog is en VIEIND laag, heeft het VIWRIT signaal een frequentie gelijk aan 400FH. Dit signaal bepaalt de bemonsteringstijdstippen van de ADC.

Evenals bij de encoder zijn de signalen VIEIND en VIWRIT afgeleid van VIOPN, VIFRAM en 400FH. De schakeling in de decoder die de signalen VIEIND en VIWRIT creëert is identiek aan de schakeling bij de encoder.

7.3. Implementatie-ontwerp uitgaande van kwadratuur modulatie

Voordat er wegens praktische redenen (zie paragraaf 5.3) afgezien werd van het gebruik van QAM voor het verzenden van binaire gegevens is er eerst nog enig onderzoek naar en ontwikkeling van deze mogelijkheid geweest. Het was de bedoeling een prototype DSU systeem te ontwikkelen welke, volgens keuze, hetzij met meer-niveau, hetzij met 16-QAM kon werken. Fig 7.5 geeft een schematische voorstelling van het gecombineerde meer-niveau en 16-QAM systeem.

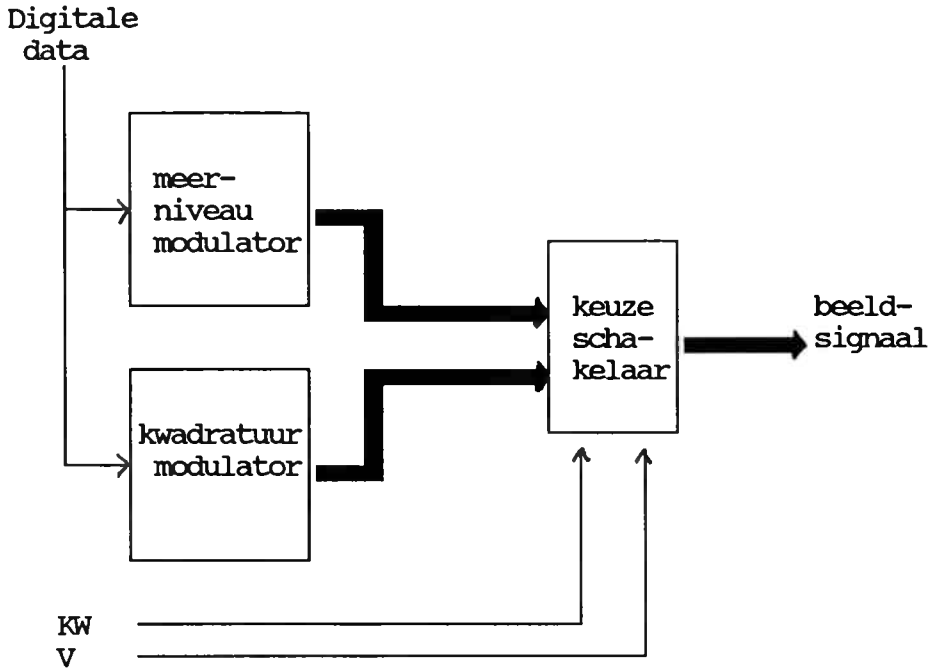


Fig. 7.5 Schematische voorstelling van een DSU systeem met zowel een meer-niveau modulator als een kwadratuur modulator. De besturings-signalen van de keuze schakelaar zijn als volgt gedefinieerd:

KW Geeft aan of het kwadratuur danwel meer-niveau signaal moet worden doorgelaten.

V Geeft aan of het signaal gebied constant danwel variabel is (zie hst 3), in het geval van het constant signaalgebied moet er te allen tijden het meer-niveau signaal worden doorgelaten. Het V signaal wordt ook in appendix B beschreven.

Samen bepalen deze twee signalen volgens onderstaand waarheidstabel het uitgangs-signaal

KW	V	Uitgang
0	0	meer-niveau
0	1	meer-niveau
1	0	meer-niveau
1	1	kwadratuur

Figuur 7.5 laat zien dat besturingssignalen bepalen of er 16-QAM danwel meer-niveau kanaalcodering wordt gebruikt. De uitgangssignalen van zowel de meer-niveau schakeling als de 16-QAM schakeling worden continu gegenereerd, onafhankelijk van de vraag. Er wordt onderscheid gemaakt tussen twee situaties:

1. Er is gekozen voor een meer-niveau uitgangssignaal. In dit geval wordt het 16-QAM voor niets gegenereerd.
2. Er is gekozen voor een 16-QAM uitgangssignaal. In dit geval is het uitgangssignaal tijdens het variabel signaalgebied (zie hoofdstuk 3) afkomstig van de 16-QAM modulator, en tijdens het constant signaalgebied afkomstig van de meer-niveau modulator. Dit is noodzakelijk omdat de meer-niveau modulator tijdens het constant signaalgebied de voor het NTSC signaal correcte signaalvorm afgeeft. De 16-QAM modulator doet dat niet. Dit constant signaalgebied bestaat onder andere uit horizontale sync, blanking en color burst.

De kwadratuur modulator afgebeeld als één blok in figuur 7.5 is nader uitgewerkt in figuur 7.6. Hierin is te zien dat voor de eigenlijke modulatie gebruik is gemaakt van de Philips TDA 2501 PAL-NTSC encoder. Deze TDA 2501 heeft als invoer voor kwadratuur modulatie de U en V signalen afgebeeld in figuur 7.6. Deze U en V signaal worden gegenereerd door de vier data signalen VID0-VID3 (Video data 0 ... Video Data 3). Vanwege specifieke eisen verbonden aan de U en V signalen moest er voor de 2 bits DAC's afgebeeld in figuur 7.6 een analoge schakeling worden ontworpen. Deze schakeling is in appendix D weergegeven.

7.4. DSU-Utility communicatie programma

Het DSU-Utility communicatie programma is in het kader van het afstudeeronderzoek ontworpen ten behoeve van onderhoud en debugging van het DSU systeem. Het programma wordt in appendix G beschreven en wordt in dit hoofdstuk volledigheidshalve genoemd.

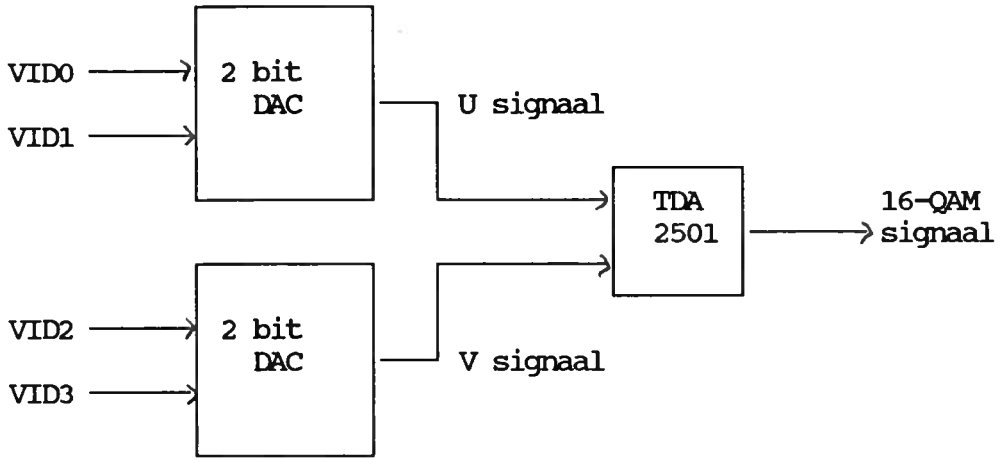


Fig. 7.6 Schematische voorstelling van de kwadratuur modulator afgebeeld in figuur 7.4. Deze bestaat uit 2 DAC's voor aanpassing van de data signalen voor de eigenlijke modulator TDA 2501.
De signalen VID0 t/m VID3 zijn de te versturen databits. De signalen U en V zijn de aansturings signalen van de Philips TDA 2501 PAL-NTSC encoder. De signalen U en V hebben beide vier niveau's.

8. CONCLUSIE EN AANBEVELINGEN

Om tot een eindconclusie te komen zal eerst een kort overzicht worden gegeven van de voor deze conclusie relevant gegevens. Tijdens het afstudeeronderzoek zijn een aantal aspecten van het te bouwen DSU systeem belicht. Het belangrijkste aspect betreft de keuze van bron- en kanaal codering om een zo goed mogelijke geluids reconstructie aan de kant van de DSU decoder te garanderen.

Daarbij is ingegaan op drie kanaalcoderings technieken, te weten:

- Digitale meer-niveau transmissie
- Digitale kwadratuur transmissie
- Analoge transmissie

Daarnaast zijn ook 3 verschillende broncoderingen belicht, te weten:

- Lineaire PCM
- Logaritmische PCM
- Adaptief differentieel PCM

Hoewel de combinatie van Log PCM met analoge transmissie het meest eenvoudig uitvoerbaar lijkt te zijn, is het niet mogelijk van deze methode gebruik te maken omdat er al een patent op rust. Uiteindelijk is dan ook gekozen is voor de combinatie van het versturen van ADPCM bemonsteringen volgens een digitale 4-niveau codering op de plaats van het luminantie gedeelte van één videoframe. Wegens de te verwachten bit-foutenkans is het rendement uigedrukt in tijsduur of bandbreedte van het audiofragment bij 4-niveau transmissie van ADPCM waarden 2 keer zo slecht als bij analoge transmissie van Log PCM waarden.

Bij het gebruik van 4-niveau transmissie ligt naar mijn mening de bit-error-rate te hoog (10^{-3}) voor het gebruikte ADPCM algoritme van OKI Semiconductors. Dit komt omdat in de praktijk is gebleken da het ADPCM algoritme van OKI zeer gevoelig is voor storingen. Er valt nog te bezien hoe het ADPCM algoritme na implementatie volgens de CCITT aanbeveling G-722 tegen fouten zal zijn bestand.

Mijns inziens is het om deze reden van belang om door middel van een praktijk test, op korte termijn de bit-error-rate bij 4-niveau transmissie vast te stellen. De aldus empirisch bepaalde BER kan dan worden gebruikt om na te gaan welke geluidskwaliteit een aantal bekende ADPCM algoritmen kunnen overdragen. Deze gemeten BER kan ook worden gebruikt om na te gaan hoe nog in ontwikkeling zijnde algoritmen zullen presteren tezamen met het 4-niveau systeem. Zo'n test is echter tot op heden nog niet uitgevoerd.

Daarnaast is het waarschijnlijk alsnog interressant om te proberen het bestaande patent [Pargee 31-1-84] te gebruiken, bijvoorbeeld in licentie vorm. Alvorens dit te doen zal er echter eerst moeten worden getest of het systeem goed functioneert.

LITERATUUR

[1] A.J. van den Berg,
"Dialoog TV; kabelexperiment Zuid-Limburg",
I²-Electrotechniek/Electronica, pp. 35-39, nr 4 1986.

[2] A.J. van den Berg, ea.,
"De kabelexperimenten in Zuid-Limburg, Amsterdam en Zaltbommel",
Informatie en Informatiebeleid, pp. 7-10, nr 12 1985.

[3] D. Boekee en J. Biemond,
Theorie van de stochastische signalen,
collegediktaat Technische Universiteit Delft, 1983-84

[4] LJ. Boxma en D. Boekee,
Informatie theorie I,
collegediktaat Technische Universiteit Delft, 1984-85

[5] M.G. Carasso e.a.,
"Het systeem Compact Disc digital audio",
Philips technisch tijdschrift, pp. 267-272, nr 9 1982.

[6] A.B. Carlson,
Communication systems, third edition,
Signapore: McGraw Hill, 1986.

[7] D. Goedhart ea.,
"Compact Disc - de omzetting van digitaal naar analoog bij het afspelen",
Philips technisch tijdschrift, pp. 290-295, nr 9 1982.

[8] R.V.L. Hartley,
"Transmission of information",
Bell Systems Technical Journal pp. 535-563, juli 1928.

[9] J.P. Heemskerk,
"Compact Disc - systeem aspecten en modulatie",

Philips technisch tijdschrift, pp. 274-281, nr 9 1982.

[10] K. Helwig, Selbach, Vary,
Hohe sprachqualitat mit 64 Kbits/s experimental codec fur ISDN und konferenzeinrichtungen mit 7 Khz bandbreite, Philips Kommunikations Industrie A.G., Fachbericht 12, maart 1985.

[11] H. Hoeve,
"Fouten correctie en maskering in het Compact Disc systeem",
Philips technisch tijdschrift, pp. 282-288, nr 9 1982.

[12] J. Lynch en Thomas,
Data compression techniques and applications,
Lifetime Learning publications,1985.

[13] M. Magel,
"Still frame audio: the voice in the video",
Danbury Ct USA: E-ITV magazine, vol 17 nr 8 aug 1985.

[14] NEC Electronics
Ram and Eprom product description,
NEC Electronics (BNL), Boschdijk 187a, Eindhoven.

[15] H. Nyquist,
"Certain factors affecting telegraph speed",
Bell Systems Technical Journal pp. 324-346, april 1924.

[16] OKI Electric Industry Company, Ltd.,
OKI Voice Synthesis LSI Data Book,
eerste editie, USA, augustus 1987.

[17] R.W. Pargee,
"Television compressed audio",
U.S. Patent nr 4429332, 31-1-1984, 1981.

[18] R.W. Pargee,
"Television burst service",

U.S. Patent nr 4422093, 20-12-1983, 1983.

[19] R.W. Pargee,
"Television digital data frame with error detection",
U.S. Patent nr 4426698, 17-1-1984, 1981.

[20] Philips Elcoma,
A four-IC set for 8 mm PCm audio recorders,
Ongedateerd.

[21] Philips PTIS,
Philips/GTE Picture & Sound Base Head-End system,
Philips PTIS, Maanweg 156 Den Haag, 1988.

[22] R.J. Sluyter,
"Digitalisering van spraak",
Philips technisch tijdschrift, pp. 209-232, nr 7/8 1983.

[23] R.J. Sluyter,
Scouting the possibilities for CD-ROM sound coding,
Nat Lab Philips, februari 1985.

[24] Teleservices Report,
"Teleaction set for june debut in Chicago",
Washington D.C.: Arlen communications, pp. 1-5, nr 57, maart 1987.

[25] Video Print,
"Teleaction: an exciting leap into the home",
International rescource development inc., pp. 1-5, vol 8 nr 5, 9 maart 1987.

[26] R.E. Yablonski en J.E. Roe,
"Adaptive automatic gain control",
U.S. patent nr 4625240, 25-11-1986, 1984.

Appendix A NTSC VIDEO STANDAARD

PHILIPS

Basic characteristics of video and synchronizing signals

Item	Characteristics	
1	Number of lines per picture (frame)	525
2	Field frequency (fields/second) (nominal value)	59.94 see also note (1)
3	Line frequency f_H and tolerance when operated non synchronously (Hz)	15734.264 ± 0.0003% see also note (2)
4	Nominal video bandwidth (MHz)	4.2
5	Line synchronization	see pages.4 and 5
6	Field synchronization	see pages 6 and 7
7	Assumed gamma of display device for which pre-correction of monochrome signal is made (see note 3)	2.2

Notes : (1) For monochrome transmission : 60.

(2) For monochrome transmission : 15750.
The maximum variation rate of line frequency valid for monochrome transmission : 0.15%/s. (These values are not valid when the reference of synchronism is being changed). The values used in Japan are ± 0.1
For colour transmission, further study is required to define maximum variation rate of line frequency.

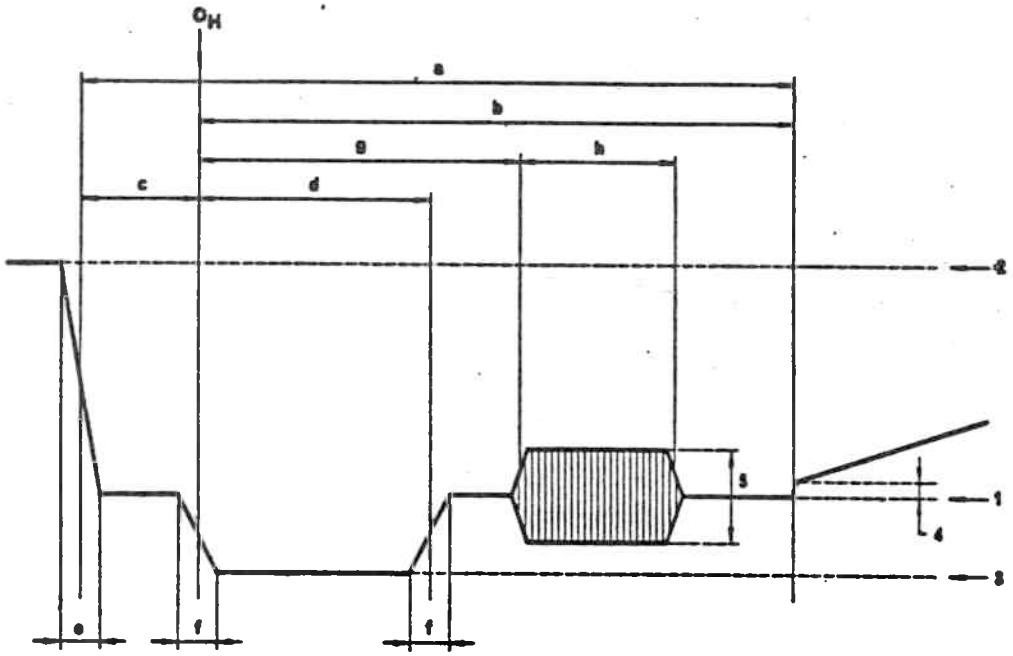
(3) The gamma of the picture tube is defined as the slope of the curve giving the logarithm of the luminance produced as a function of the logarithm of the video signal voltage when the brightness control of the receiver is set so as to make this curve as straight as possible in a luminance range corresponding to a contrast of at least 1/40.

Pre-correction is intended to compensate for the non-linearities of the transfer characteristics of picture tubes in a luminance range corresponding to a contrast of at least 1/40.
It is assumed that the transfer characteristic of the picture tube follows a power law, the exact exponent of which is still under study.

PHILIPS

Details of line synchronizing signals

Durations measured between half-amplitude points on the appropriate edges.



(a) NTSC and PAL systems Fig. 1

Item	Characteristics	
1	Blanking level (reference level) (%)	0
2	Peak-white level (%)	100
3	Synchronizing level (%)	- 40
4	Difference between black and blanking level (%)	7.5 ± 2.5
5	Peak to peak value of burst (%)	4/10 (100) ± 10%
H	Nominal line period (µs)	63.5555 (63.492)
a	Line blanking interval (µs)	10.5 - 11.4 (10.2-11.4)
b	Interval between time datum O_H and back edge of line-blanking signal (µs)	9.2 - 10.3 (8.9 - 10.3)
c	Front porch (µs)	1.27 - 2.22 (1.27-2.54)

nominal

PHILIPS

Details of line synchronizing signals (continued)

Item	Characteristics	
d	Synchronizing pulse (µs)	4.13 - 5.08 (4.19-5.71)
e	Build-up time (10-90%) of the edges of the line blanking signal (µs)	≤ 0.48 (≤ 0.64)
f	Build-up time (10-90%) of the line synchronizing pulses (µs)	≤ 0.25
g	Start of sub-carrier burst (µs)	4.71 - 5.71 at least 0.38 µs after the trailing edge of line synchronization signal
h	Duration of sub-carrier burst (µs)	2.23 - 3.11 minimum 8 cycles

The values in brackets apply to monochrome transmission

Details of field synchronizing signals (see also fig. 2 on page 7)

Item	Characteristics	
v	Field period (ms)	16.6833 (16.667)
j	Field-blanking period (for H and a, see page 4)	(19 - 21) H + a The following values are used in Japan $0.07V^{+0.01V}_{-0}$ for colour transmission $0.05V^{+0.03V}_{-0}$ for monochrome transmission
j'	Build-up time (10-90%) of the edges of field-blanking pulses (µs) (j' is not indicated in the diagram)	≤ 6.35
k	Interval between front edge of field-blanking interval and front edge of first equalizing pulse (µs) (k is not indicated in the diagram)	—
l	Duration of first equalizing pulse sequence	3H
m	Duration of synchronizing pulse sequence	3H
n	Duration of second equalizing pulse sequence	3H
p	Duration of equalizing pulse (µs)	2.29 - 2.54 The following specification is also applied in Japan. An equalizing pulse has 0.45 ~ 0.5 times the area of a line-synchronizing pulse
q	Duration of field-synchronizing pulse (µs)	26.4 - 28.0
r	Interval between field-synchronizing pulses (µs)	3.81 - 5.34
s	Build-up time (10-90%) of synchronizing and equalizing pulses (µs)	≤ 0.25

The value in brackets applies to monochrome transmission.

PHILIPS

FIGURE 2

Details of field-synchronizing waveforms

FIGURES 2-3

Diagrams applicable to system M

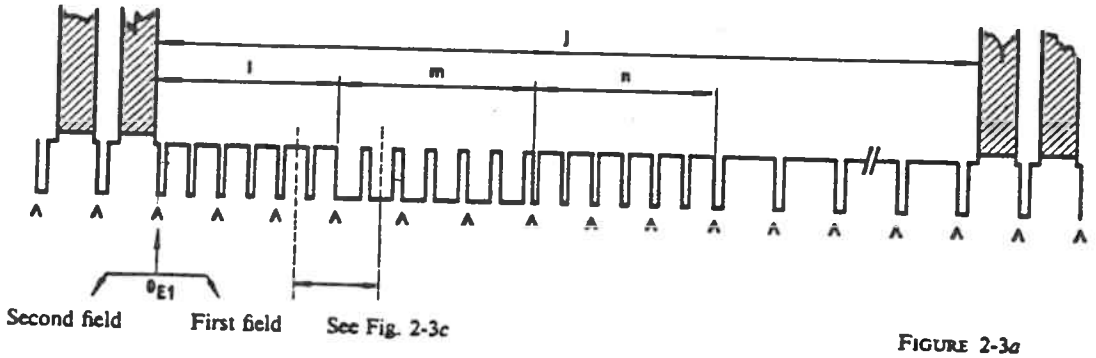


FIGURE 2-3*a*

Signal at beginning of each first field

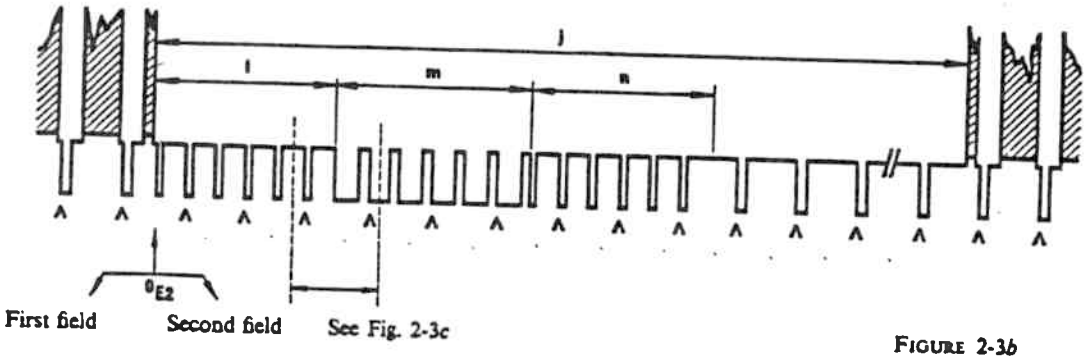


FIGURE 2-3*b*

Signal at beginning of each second field

Note 1. — ∧ indicates an unbroken sequence of edges of line-synchronizing pulses throughout the field-blanking period.

Note 2. — Field-one line numbers start with the first equalizing pulse in Field 1, designated O_{E1} in Fig. 2-3*a*.

Note 3. — Field-two line numbers start with the second equalizing pulse in Field 2, one-half-line period after O_{E2} in Fig. 2-3*b*.

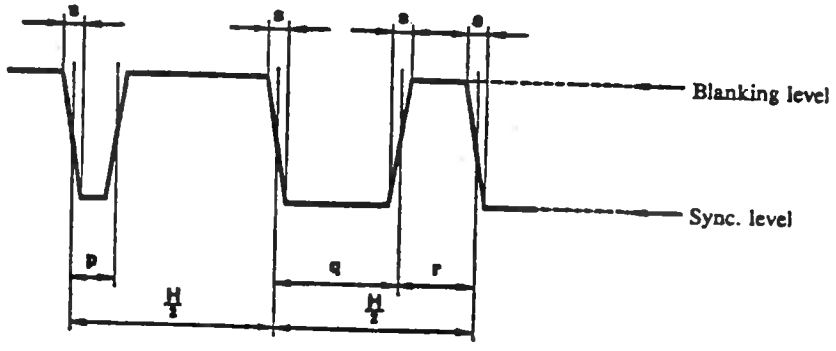


FIGURE 2-3*c*

Details of equalizing and synchronizing pulses

PHILIPS

Characteristics of the video signal for colour television

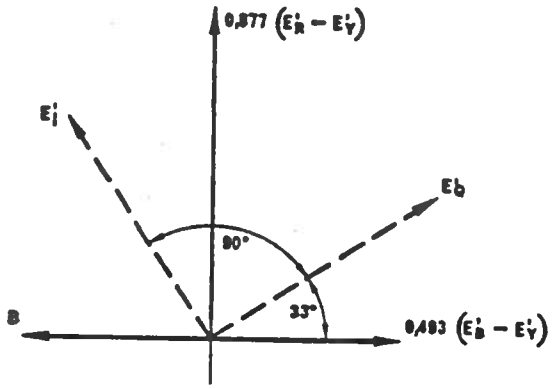
Item	Characteristics	
3.1	Assumed chromaticity coordinates (CIE 1931) for primary colours of receiver	x / y Red 0.67 / 0.33 Green 0.21 / 0.71 Blue 0.14 / 0.08
3.2	Chromaticity coordinates for equal primary signals $E'_R = E'_B = E'_G$	Illuminant C x=0.310 y=0.316 In Japan the chromaticity of studiomonitors is adjusted to a D-white at 9300 K
3.3	Assumed gamma value of the receiver for which the primary signals are pre-corrected The primary signals are pre-corrected so that the optimum quality is obtained with a display having the indicated value of gamma	2.2
3.4	Luminance signal E'_R, E'_G and E'_B are gamma-pre-corrected primary signals	$E'_Y = 0.299\ E'_R + 0.587\ E'_G + 0.114\ E'_B$ see note 1 on page 10
3.5	Chrominance signals (colour difference)	$E'_I = -0.27\ (E'_B - E'_Y) + 0.74\ (E'_R - E'_Y)$ $E'_Q = 0.41\ (E'_B - E'_Y) + 0.48\ (E'_R - E'_Y)$
3.6	Attenuation of colour difference signals (dB at MHz)	E'_I = < 2 at 1.3, ⩾20 at 3.6 E'_Q = < 2 at 0.4, <6 at 0.5, ⩾6 at 0.6
3.7	LF pre-correction of colour difference signal	–
3.8	Time-coincidence error between luminance and chrominance signals (µs)	< 0.05 Excluding pre-correction for receiver response

Characteristics of the video signal for colour television

Item	Characteristics	
3.9	Equation of composite colour signal	$E_M = E'_Y + E'_Q \sin(2\pi f_{sc}t + 33°) + E'_I \cos(2\pi f_{sc}t + 33°)$ where : E'_Y, see item 3.4 E'_Q and E'_I, see item 3.5 f_{sc}, see item 3.11 see also figure 4a on page 10
3.10	Type of chrominance sub-carrier modulation	Quadrature amplitude modulation with suppressed carrier
3.11	Chrominance sub-carrier frequency a) nominal value and tolerance (Hz)	3579545 ± 10
	b) relationship between chrominance sub-carrier frequency f_{sc} and line-frequency f_H	$f_{sc} = \frac{455}{2} f_H$
3.12	Bandwidth of chrominance sidebands (kHz)	f_{sc} + 620 − 1300
3.13	Amplitude of chrominance sub-carrier	$G = \sqrt{E'^2_I + E'^2_Q}$
3.14	Synchronization of chrominance sub-carrier g) start of sub-carrier burst (μs)	Sub-carrier burst on blanking back porch 4.71 - 5.71 at least 0.38 μs after the trailing edge of line synchronization signal see fig. 1a on page 4
	h) duration of sub-carrier burst (μs)	2.23 - 3.11 minimum 8 cycles see fig. 1a on page 4
3.15	Peak-to-peak value of chrominance sub-carrier burst For the use of automatic gain control circuits it is important that the burst amplitude should maintain the correct ratio with the chrominance signal amplitude	4/10 of difference between blanking level and peak-white level ± 10% see fig. 1a on page 4

Characteristics of the video signal for colour television (continued)

Chrominance axes and phase of the burst



B: phase of the burst

(a) NTSC system

Fig. 4

Item	Characteristics	
3.16	Phase of chrominance sub-carrier burst see fig. 1a on page 4	180° relative to $(E'_B - E'_Y)$ axis see fig. 4a on page 10
3.17	Blanking of chrominance sub-carrier	Following each equalizing pulse and also during the broad synchronizing pulses in the field blanking interval
3.18	Synchronization of chrominance sub-carrier switching during line blanking	Does not apply to NTSC systems

Note : 1) In certain countries using the SECAM systems and in Japan it is also permitted to obtain the luminance signal as a direct output from an independent photo-electric analyser instead of from the primary signals.

PHILIPS

Characteristics of the radiated signals

Item	Characteristics		
4.1	Nominal radio frequency channel bandwidth (MHz)	6	see fig. 10 on page 13
4.2	Sound carrier relative to vision carrier (MHz)	+ 4.5 see note 1 on page 12	
4.3	Nearest edge of channel relative to vision carrier (MHz)	- 1.25	
4.4	Nominal width of main sideband (MHz)	4.2	
4.5	Nominal width of vestigial sideband (MHz)	0.75	
4.6	Minimum attenuation of vestigial sideband (dB at MHz) See note 2 on page 12	20 (- 1.25) 42 (- 3.58)	
4.7	Type and polarity of vision modulations	C3F negative	
4.8	Levels in the radiated signal (% of the peak carrier)		
	a) synchronizing level	100	
	b) blanking level	72.5 to 77.5	
	c) difference between black level and blanking level	2.88 to 6.75	
	d) peak-white level	10 to 15	
4.9	Type of sound modulation	F3E	
4.10	Frequency deviation (kHz)	± 25	
4.11	Pre-emphasis for modulation (µs)	75	
4.12	Ratio of effective radiated powers of vision and sound (see note 3 on page 12)	10/1 to 5/1 see note 4 on page 12	
4.13	Pre-correction for receiver group-delay characteristics at :		
	a) medium video frequencies (ns)	0	
	b) colour sub-carrier frequencies (ns)	- 170 (nominal) see fig. 3 on page 13	

Characteristics of the radiated signals (continued)

Notes : 1) In Japan, the values + 4.5 ± 0.001 are used.

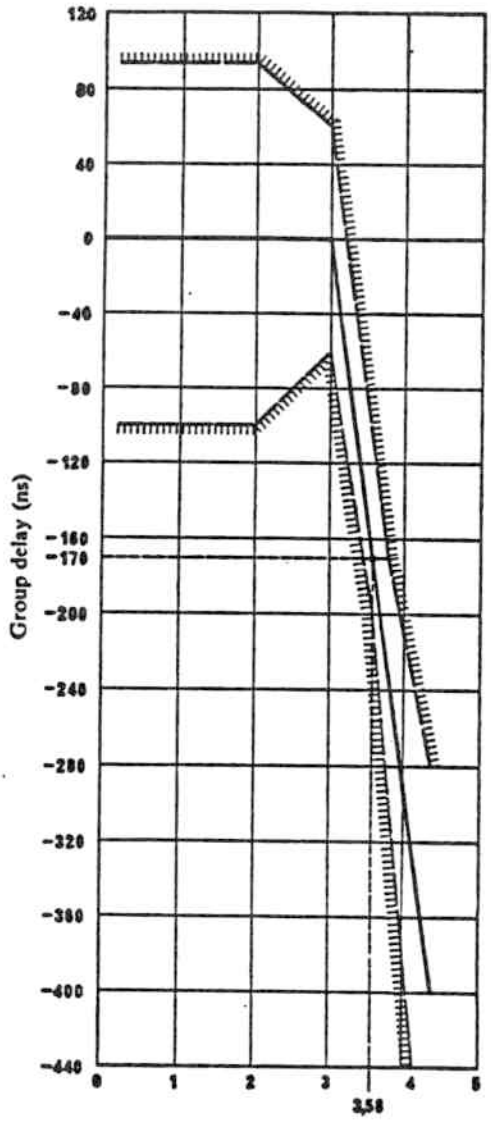
2) In some cases, low-power transmitters are operated without vestigial-sideband filter.

3) The values to be considered are :

- the r.m.s. value of the carrier at the peak of the modulation envelope for the vision signal.
- the r.m.s. value of the unmodulated carrier for amplitude-modulated and frequency-modulated sound transmissions.

4) In Japan a ratio of 1/0.15 to 1/0.35 is used.

Characteristics of the radiated signals (continued)



(b) M/PAL and M/NTSC systems

FIGURE 3

Curve of pre-correction for receiver group-delay characteristics

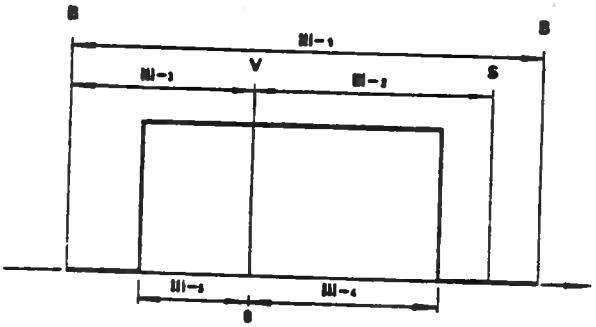


FIGURE 10

Significance of items 1 to 5 on page 11

B: Channel limit
V: Vision carrier
S: Sound carrier

Appendix B EEN DIGITALE NTSC VIDEO FRAME GENERATOR

Om het DSU systeem correct te laten functioneren is het nodig dat er, aan de kant van de encoder, een digitale representatie van een NTSC video frame aanwezig is. Om deze representatie te genereren is een PASCAL programma ontwikkeld. Deze appendix beschrijft de achtergronden van de bovengenoemde representatie.

Bij het ontwerp van de encoder van het DSU systeem is gekozen voor het gebruik van een UVC-3100 [appendix E] AD/DA converter voor video. Daarnaast wordt gebruik gemaakt van een serieel videogeheugen bestaande uit 12 NEC uPD41221C [NEC] IC's. Om een analoog beeldsignaal aan de uitgang van de DSU encoder te verkrijgen wordt de inhoud van het serieel geheugen aan de digitaal naar analoog converter (DAC) van de UVC-3100 aangeboden.

Het serieel geheugen is 6 bits breed. De organisatie van die 6 bits is in figuur B.1 weergegeven. Uit deze figuur is te zien dat er naast de 4 data bits VD0-VD3, een NOT Burst (/B) bit en een Variabel (V) bit is. De betekenis van de databits spreekt voor zich, zij stellen de bemonsterde audio waarden voor. Om de betekenis van de /B en V bits uit te leggen is het eerst zinvol om te bezien hoe de 6 bits aan de UVC-3100 worden aangeboden. Figuur B.2 laat dit zien.

Figuur B.2 geeft aan hoe de UVC-3100 zal worden aangesloten voor wat betreft de data ingangen. Naast de aangesloten databits is er het V bit. Dit bit geeft aan of de aangeboden bemonstering een constant of variabel signaalbemonstering is. Het V bit wordt als most significant bit (MSB) aan de DAC aangeboden. Dit heeft als consequentie dat alle uitgangs signalen onder de 1 Volt tot het constant signaalgebied zullen gaan behoren en alle signalen boven de 1 Volt tot het variabel signaalgebied. Het in figuur B.2 afgebeelde /B bit bepaalt of er wel of niet burst aan het analoog signaal moet worden toegevoegd.

Om digitale gegevens afkomstig van het audiofragment toe te mogen voegen aan de digitale afbeelding van het voorgedefinieerde beeldsignaal hoeft er alleen op het V bit getest te worden. Voor waarden

NOT BURST /B	VAR V	DATA VD3	DATA VD2	DATA VD1	DATA VD0
bit 5	bit 4	bit 3	bit 2	bit 1	bit 0

Fig B.1 Organisatie van 6 bits van het serieel video geheugen.

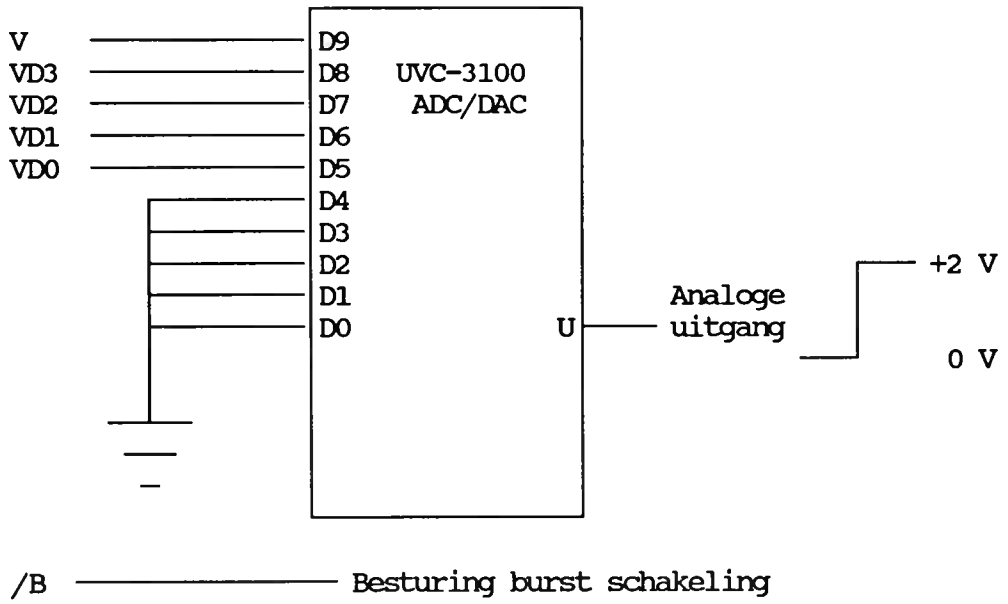


Fig B.2 Data aansluiting van de UVC-3100 ten behoeve van video generatie. De laagste 4 bits zijn logisch 0. De uitgang varieert tussen de 0 en +2 Volt.

van het voorgedefinieerde beeldsignaal waar het V bit niet geldig is (het constante gebied) moet het niveau echter ook vaststaan. Het serieel geheugen zal gevuld moeten worden met of variabele of constante bemonsterings waarden. Voor het constante gebied zijn drie verschillende waarden gedefinieerd. Tezamen met het variabel niveau vormen de vier verschillende signaalniveau typen, te weten:

- Sync level. Dit is het niveau van de horizontale sync zoals gedefinieerd in appendix A.
- Blanking level. De blanking level is eveneens gedefinieerd in appendix A.
- Burst level. De burst level vormt een uitzondering. Hiermee wordt bedoeld de blanking level met daarop gesuperponeerd de color burst frequentie van 3.4 MHz. Omdat deze burst frequentie te hoog ligt voor de gebruikte bemonsterings frequentie wordt dit signaal later analoog toegevoegd.
- Variabel level. Dit level kan worden gebruikt voor het toevoegen van informatie. Als default wordt hiervoor zwart (Black level) gekozen.

Tabel B.1 geeft een overzicht van deze vier niveaus en de daarbij behorende spanning aan de uitgang van de DAC. Deze tabel geeft aan dat het laagste signaal niveau (Sync) een spanning van 0,5 volt heeft. Dit is het laagste spanning dat de DAC door de aangeboden data bits genereert. Het hoogste niveau is dat als VD0-VD3 = 1111 en V = 1 (niet in de tabel aangegeven), wat resulteert in een maximale spanning van 1,93 Volt. Het aldus aan de uitgang van de DAC te meten signaal heeft een 1,43 V_{pp} (Voltage peak to peak) heeft en een 0,5 V_o (Voltage offset) heeft. Vanwege de eenvoud van programmering en besturing voor de encoder en decoder is voor deze opzet gekozen. Die eenvoud komt door het bestaan van de V bit voor constant en variabel signaalgebied en de B bit voor de besturing van het toevoegen van de colorburst aan het uitgangssignaal. Als gevolg van deze keuze zal het signaal omlaag worden verschoven en verkleind moet worden. Dit is noodzakelijk om te voldoen aan de video eisen dat V_o = 0 Volt en V_{pp} = 1 Volt. De

Level	Bit combinatie /B V DATA	Spanning uitgang DAC	Toevoegen Burst
SYNC	1 0 1000	0,5 Volt	nee
BLANKING	1 0 1111	0,973 Volt	nee
BURST	0 0 1111	0,973 Volt	ja
BLACK	1 1 0000	1,0 Volt	nee

Tabel B.1. Verschillende voorgedefinieerde signaal niveaus.

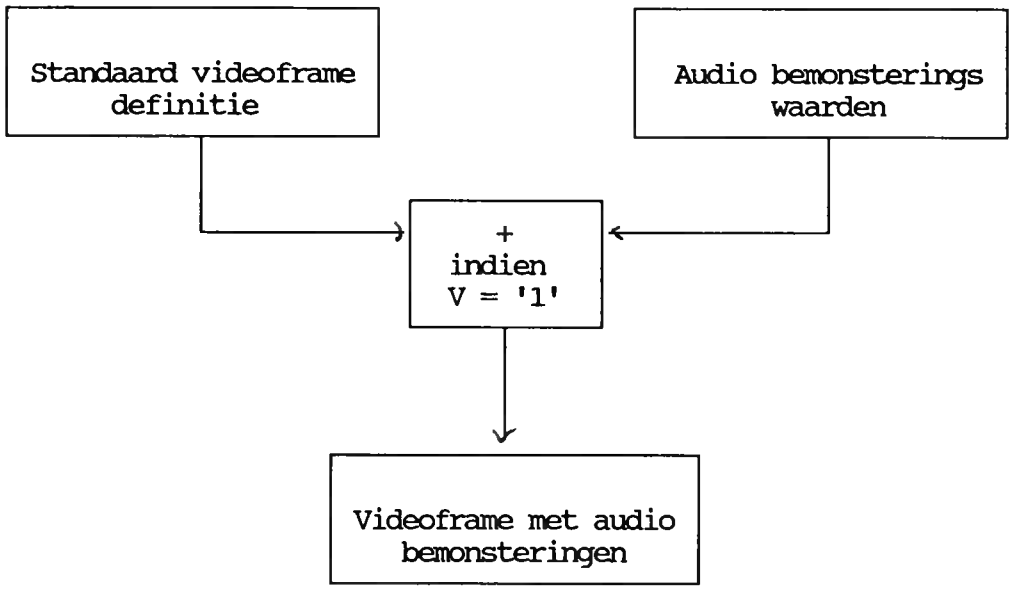


Fig B.3 Opbouw van een digitaal videoframe gemengd met audio bemonsteringen.

schakelingen die voor verschuiving en verkleining zorgen zijn eveneens binnen het kader van het afstudeeronderzoek ontworpen en zijn in hoofdstuk 7 beschreven.

Om een digitale representatie van een videoframe op te bouwen, is gekozen voor een opzet zoals getekend in figuur B.3. De figuur laat zien dat er data aan een voorgedefinieerde videoframe definitie wordt toegevoegd. Dit gebeurt afhankelijk van het V bit binnen de videoframe definitie. De aangeboden datawaarden zijn afkomstig van het ADPCM bemonsterings circuit. Deze videoframe definitie bestaat uit 400 (samples per lijn) * 525 (aantal lijnen)= 210000 acht bits bytes, en representeert zowel het variabel als het constant signaal gebied.

Deze definitie zal binnen het systeem aanwezig moeten zijn. Toch is het niet noodzakelijk om alle 210 Kb aan samples te bewaren. Er is gekozen voor een PASCAL record structuur om die 210 Kb te definiëren. Hierbij zijn per lijn 7 "tuples" (getallenparen) gedefinieerd ten behoeve van een variabele lengte code. Het blijkt dat 7 tuples per horizontale videolijn voldoende is om ieder willekeurige lijn vast te leggen. Het PASCAL record ziet er als volgt uit:

```
ARRAY [1..525] OF
    ARRAY [0..6] OF
       RECORD
          Length : INTEGER;
          Level  : BYTE;
       end;
```

Iedere tuple geeft een level en een aantal samples (length) aan. De 2 MSB's van de level bytes zijn niet significant, er zijn slechts zes bits nodig, uit praktische overweging is toch gekozen voor een acht bits representatie van het voorgedefinieerde videoframe. Per lijn moet het totaal aantal samples 400 zijn. Het in de PASCAL datastructuur aangegeven Level byte is een van de vier voorgedefinieerde Levels vermeld in tabel B.1. De in beslag genomen geheugen ruimte van dit PASCAL datastructuur is:

$$525 * (7 * (1 + 2)) = 11025 \text{ Bytes.}$$

met

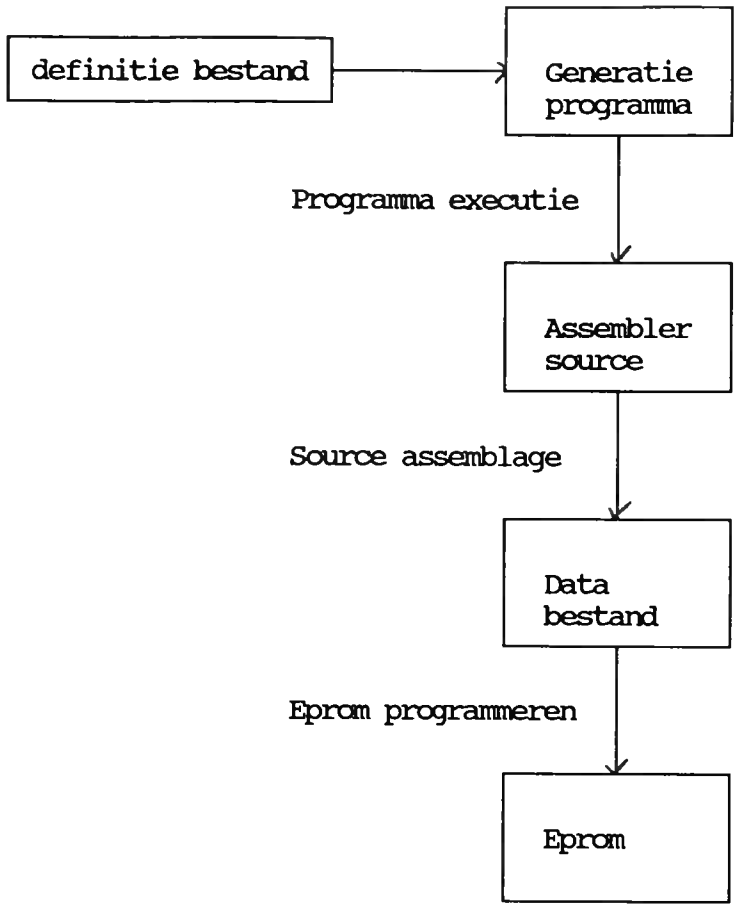


Fig B.4 Te doorlopen stappen bij het genereren van een NTSC frame-definitie.

525 lijnen van
  7 tuples met
  1 byte voor de level
  2 bytes voor de length (2 byte integer)

De meeste lijnen kunnen met 6 of minder tuples worden vastgelegd, de array is dus niet optimaal gevuld. Een verbetering voor wat betreft geheugen gebruik zou zijn een dynamische array of een in PASCAL bruikbaar "linked list". In dit geval is gekozen van eenvoud van programmering, arrays zijn eenvoudig adresseerbaar.

De voorgedefinieerde videoframe staat in een acht bits breed EPROM geheugen. Om deze 11 Kb te produceren is een pascal programma ontwikkeld. Dit programma genereert een assembler source programma welke geassembleerd wordt op een HP 64000 computer. Deze computer kan de geassembleerde code in een EPROM schrijven. Figuur B.4 geeft aan welke stappen doorlopen worden om zo'n NTSC frame definitie te genereren.

De volgende bladzijden geven het definitie bestand NTSC.DEF, een listing van het generatie programma NTSC.PAS en de tekst uitvoer NTSC.S van NTSC.PAS welke door de HP64000 als assembler source code wordt herkend.

Bestand NTSC.DEF:

```
const
   line_per              = 0.0000635555;
   sample_time           = 0.0000001588889;

   error                 = 0.00000001;
   blanking_interval     = 0.00001095;  (*(0.0000105 + 0.0000114)/2;*)
   equ_pulse_lines_1     = 3;
   sync_pulse_lines      = 6;
   equ_pulse_lines_2     = 9;
   equ_pulse_time        = 0.00000239;
   blanking_pulse_time   = 0.00002938775;(*line_per/2-equ_pulse_time*)
   field_sync_pulse_time = 0.0000272;    (*(0.0000264 + 0.000028)/2;*)
   sync_pulse_time       = 0.00000461;
   data_time             = 0.0000526055;(*line_per-blanking_interval*)
   half_data             = 0.000022244446;(* 140 * sample_time; *)
   from_sync_to_burst    = 0.000000954;
   from_burst_to_audio   = 0.00000143;
   front_porch           = 0.00000143;
   duration_of_burst     = 0.00000261;

   first_frame_data_start = 22;
   first_frame_data_end   = 262;
   secon_frame_data_start = 285;
   secon_frame_data_end   = 525;

   totfields = 2;
   data_lines= 241;
     (* (secon_frame_data_end - 1) / 2 - (first_frame_data_start-1);*)
   first_frame = 263;
```

Bestand NTSC.PAS

```
Program NTSC(input,output);
(*
 * Author: R.F. Mous
 * Date  : 24 NOV 1987
 *
 * Input : File NTSC.DEF
 *         Ntsc.def is an include file for the source program. It
 *         contains video timing information.
 *
 * Output: File NTSC.S
 *         Ntsc.s is the source code for generation of an NTSC
 *         video frame in bytes. Per byte 6 significant bits:
 *
 *            Bit 5   NOT Burst mode bit.
 *                     1 = Do not add burst to signal.
 *                     0 = Add burst to signal.
 *            Bit 4   Data mode bit.
 *                     0 = Do not add data samples.
 *                     1 = Add data samples.
 *            Bit 3-0 If data mode not set then these bits represent
 *                    the level of sync or blanking.
 *                    If data mode set then these bits represent data
 *                    samples. Default value is black = 0000.
 *
 * Description:
 *         This program creates a source file for the HP 80188
 *         cross-assembler in order to make a default record value
 *         accesable to a pascal program on an 80188 processor.
 *         The record values create a video-like signal after
 *         expansion by the 80188 and processing by an UVC 3100
 *         digital to analog converter. The video-like signal will
 *         have a 0.5 V offset and 1.43 Vpp value.
 *         The video-like signal will have the same timing and
 *         shape as a NTSC videosignal.
 * Important:
 *         The timing is completely defined by the values in the
 *         program static data area (constants). Subsequent changes
 *         in timing should be made only in this section. See file
 *         NTSC.DEF
 *)

(*$i ntsc.def*)

type
   s5        = string[5];

var
   fieldno,
   lineno    : integer;
   outf      : text;
   empty,
   black_level,
```

```
   blanking_level,
   sync_level,
   burst_level : s5;


procedure write_header;
   (*
    *  Make a header for the HP cross-assembler
    *)
begin
   writeln(outf,'''80188''');
   writeln(outf);
   writeln(outf,'         GLB    TVFRAME');
   writeln(outf,'         ASSUME CS:PROG');
   writeln(outf);
   writeln(outf,'Blank    EQU    101111B');
   writeln(outf,'Black    EQU    110000B');
   writeln(outf,'Sync_    EQU    101000B');
   writeln(outf,'Burst    EQU    001111B');
   writeln(outf);
   writeln(outf,'         PROG');
   writeln(outf);
   writeln(outf,';Equivalent pascal data structure:');
   writeln(outf,';ARRAY[1..525] OF');
   writeln(outf,';              ARRAY[0..6] OF');
   writeln(outf,';                          RECORD');
   writeln(outf,';                             COUNT : INTEGER;');
   writeln(outf,';                             VALUE : BYTE; ');
   writeln(outf,';                          END;');
   writeln(outf);
   writeln(outf,'TVFRAME');
   writeln(outf);
   writeln(outf,'TVLINE   MACRO   &T1,&L1,&T2,&L2,&T3,&L3,&T4,&L4,',
                                 '&T5,&L5,&T6,&L6,&T7,&L7');
   writeln(outf,'         DW      &T1');
   writeln(outf,'         DB      &L1');
   writeln(outf,'         DW      &T2');
   writeln(outf,'         DB      &L2');
   writeln(outf,'         DW      &T3');
   writeln(outf,'         DB      &L3');
   writeln(outf,'         DW      &T4');
   writeln(outf,'         DB      &L4');
   writeln(outf,'         DW      &T5');
   writeln(outf,'         DB      &L5');
   writeln(outf,'         DW      &T6');
   writeln(outf,'         DB      &L6');
   writeln(outf,'         DW      &T7');
   writeln(outf,'         DB      &L7');
   writeln(outf,'         MEND ');
   writeln(outf);
end;


procedure init;
  (*
```

```
   * Initialisation of output file, variables and assembler header
   *)
begin
   fieldno := 1;
   lineno  := 1;
   empty          := '00000';
   black_level    := 'Black';
   blanking_level := 'Blank';
   burst_level    := 'Burst';
   Sync_level     := 'Sync_';
   assign(outf,'ntsc.s');
   rewrite(outf);
   write_header;
end;

procedure closedown;
   (*
    * Last statement for cross-assembler and close of output
    * Error mesage if wrong number of lines are printed.
    *)
begin
   writeln(outf,'END');
   close(outf);
   if (lineno - 1) <> secon_frame_data_end
   then
      writeln('WARNING-- Incorrect number of lines: ',lineno:3);
end;


procedure write_tv_line(l1:s5;t1:real;
                        l2:s5;t2:real;
                        l3:s5;t3:real;
                        l4:s5;t4:real;
                        l5:s5;t5:real;
                        l6:s5;t6:real;
                        l7:s5;t7:real);

   (*
    * Write a tvline macro call for the cross-assembler. layout:
    *
    *    TVLINE + 7 times number_of_samples,Level_of_sample + lineno
    *
    * Write a warning to the default output device (terminal) if
    * the number of samples doesn't add up line_per/sample_time
    *)


var
   t_s,
   tot_samples : integer;
type
   str3 = string[3];

   function add_zero(int:integer):str3;
   var
```

```
      s3 : str3;
      i  : integer;
   begin
      str(int:3,s3);
      for i := 1 to 3 do
         if s3[i] = ' '
         then
            s3[i] := '0';
      add_zero := s3;
   end;

begin
   t_s := round(t1/sample_time);
   tot_samples := t_s;
   write(outf,' TVLINE ',add_zero(t_s),',',l1,',');

   t_s := round(t2/sample_time);
   tot_samples := tot_samples + t_s;
   write(outf,add_zero(t_s),',',l2,',');

   t_s := round(t3/sample_time);
   tot_samples := tot_samples + t_s;
   write(outf,add_zero(t_s),',',l3,',');

   t_s := round(t4/sample_time);
   tot_samples := tot_samples + t_s;
   write(outf,add_zero(t_s),',',l4,',');

   t_s := round(t5/sample_time);
   tot_samples := tot_samples + t_s;
   write(outf,add_zero(t_s),',',l5,',');

   t_s := round(t6/sample_time);
   tot_samples := tot_samples + t_s;
   write(outf,add_zero(t_s),',',l6,',');

   t_s := round (t7/sample_time);
   tot_samples := tot_samples + t_s;
   writeln(outf,add_zero(t_s),',',l7,' ;',lineno:3);
   if tot_samples <> round(line_per/sample_time)
   then
      writeln('WARNING-- line ',lineno:3,' has ',tot_samples:3,'
         samples.');
end;


procedure startup(field:integer);
   (*
    * Check which line type of the field blanking period we're in:
    *    1. first equalizing pulse sequence.
    *    2. syncronizing pulse sequence.
    *    3. second equalizing pulse sequence.
    *    4. 'normal' field blanking lines.
    *
    * Create the coresponding line times and levels.
```

```
    *)
begin
   if (field > 0) and (field<=totfields)
   then
   begin
      while (lineno mod first_frame) <= equ_pulse_lines_1 do
      begin
         write_tv_line(sync_level,equ_pulse_time,
                       blanking_level,blanking_pulse_time,
                       sync_level,equ_pulse_time,
                       blanking_level,blanking_pulse_time,
                       empty,0,
                       empty,0,
                       empty,0);
         lineno := lineno + 1;
      end;
      while (lineno mod first_frame) <= sync_pulse_lines do
      begin
         write_tv_line(sync_level,field_sync_pulse_time,
                       blanking_level,sync_pulse_time,
                       sync_level,field_sync_pulse_time,
                       blanking_level,sync_pulse_time,
                       empty,0,
                       empty,0,
                       empty,0);
         lineno := lineno + 1;
      end;
      while (lineno mod first_frame) <= equ_pulse_lines_2 do
      begin
         write_tv_line(sync_level,equ_pulse_time,
                       blanking_level,blanking_pulse_time,
                       sync_level,equ_pulse_time,
                       blanking_level,blanking_pulse_time,
                       empty,0,
                       empty,0,
                       empty,0);
         lineno := lineno + 1;
      end;
      while (lineno mod first_frame) <= (first_frame_data_start -1) do
      begin
         if (field=2) and
            ((lineno mod first_frame) = (first_frame_data_start-1))
         then
         begin
            write_tv_line(sync_level,sync_pulse_time,
                          blanking_level,line_per/2-sync_pulse_time,
                          black_level,line_per/2 - front_porch,
                          blanking_level,front_porch,
                          empty,0,
                          empty,0,
                          empty,0);
         end
         else
         begin
            write_tv_line(sync_level,sync_pulse_time,
```

```
                        blanking_level,line_per - sync_pulse_time,
                        empty,0,
                        empty,0,
                        empty,0,
                        empty,0,
                        empty,0);
         end;
         lineno := lineno + 1;
      end;
   end;
end;

procedure breakdown(field:integer);
   (*
    * Start of second blanking beriod is halfway the last line of
    * first field. This requires special attention.
    * layout:
    *    sync, blanking, burst, blanking, black, sync, and backporch.
    *)
begin
   if (field = 1)
   then
   begin
      writeln(outf);
      write_tv_line(sync_level,sync_pulse_time,
                    blanking_level,from_sync_to_burst,
                    burst_level,duration_of_burst,
                    blanking_level,from_burst_to_audio,
                    black_level,half_data,
                    sync_level,equ_pulse_time,
                    blanking_level,blanking_pulse_time);
      writeln(outf);
      lineno := lineno + 1;
   end;
end;


procedure standard;
   (*
    * write a standard black data line consisting of:
    *
    *    sync, blanking, burst, blanking, black and backporch.
    *)
begin
   writeln(outf);
   while (((lineno >= first_frame_data_start) and
           (lineno <= first_frame_data_end)) or
          ((lineno >= secon_frame_data_start) and
           (lineno <= secon_frame_data_end))) do
   begin
      write_tv_line(sync_level,sync_pulse_time,
                    blanking_level,from_sync_to_burst,
                    burst_level,duration_of_burst,
                    blanking_level,from_burst_to_audio,
                    black_level,data_time,
```

```
                    blanking_level,front_porch,
                    empty,0);
   lineno := lineno + 1;
   end;
   writeln(outf);
end;



begin
   init;
   while fieldno <= totfields do
   begin
      startup(fieldno);
      standard;
      breakdown(fieldno);
      fieldno := fieldno + 1;
   end;
   closedown;
end.
```

Bestand NISC.S

```
'80188'

        GLB    TVFRAME
        ASSUME CS:PROG

Blank   EQU    101111B
Black   EQU    110000B
Sync_   EQU    101000B
Burst   EQU    001111B

        PROG

;Equivalent pascal data structure:
;ARRAY[1..525] OF
;             ARRAY[0..6] OF
;                         RECORD
;                            COUNT : INTEGER;
;                            VALUE : BYTE;
;                         END;

TVFRAME

TVLINE  MACRO   &T1,&L1,&T2,&L2,&T3,&L3,&T4,&L4,&T5,&L5,&T6,&L6,&T7,&L7
        DW      &T1
        DB      &L1
        DW      &T2
        DB      &L2
        DW      &T3
        DB      &L3
        DW      &T4
        DB      &L4
        DW      &T5
        DB      &L5
        DW      &T6
        DB      &L6
        DW      &T7
        DB      &L7
        MEND

 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  1
 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  2
 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  3
 TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;  4
 TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;  5
 TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;  6
 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  7
 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  8
 TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;  9
 TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 10
 TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 11
 TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 12
 TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 13
 TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 14
```

```
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 15
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 16
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 17
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 18
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 19
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 20
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ; 21

TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 22
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 23
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 24
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 25
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 26
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 27
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 28
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 29
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 30
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 31
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 32
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 33
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 34
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 35
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 36
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 37
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 38
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 39
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 40
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 41
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 42
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 43
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 44
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 45
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 46
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 47
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 48
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 49
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 50
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 51
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 52
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 53
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 54
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 55
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 56
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 57
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 58
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 59
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 60
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 61
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 62
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 63
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 64
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 65
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 66
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 67
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 68
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 69
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 70
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 71
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 72
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 73
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 74
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 75
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 76
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 77
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 78
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 79
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 80
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 81
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 82
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 83
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 84
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 85
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 86
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 87
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 88
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 89
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 90
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 91
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 92
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 93
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 94
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 95
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 96
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 97
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 98
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ; 99
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;100
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;101
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;102
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;103
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;104
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;105
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;106
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;107
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;108
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;109
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;110
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;111
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;112
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;113
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;114
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;115
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;116
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;117
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;118
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;119
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;120
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;121
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;122
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;123
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;124
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;125
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;126
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;127
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;128
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;129
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;130
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;131
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;132
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;133
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;134
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;135
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;136
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;137
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;138
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;139
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;140
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;141
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;142
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;143
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;144
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;145
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;146
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;147
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;148
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;149
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;150
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;151
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;152
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;153
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;154
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;155
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;156
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;157
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;158
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;159
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;160
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;161
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;162
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;163
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;164
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;165
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;166
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;167
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;168
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;169
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;170
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;171
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;172
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;173
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;174
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;175
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;176
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;177
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;178
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;179
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;180
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;181
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;182
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;183
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;184
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;185
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;186
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;187
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;188
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;189
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;190
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;191
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;192
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;193
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;194
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;195
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;196
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;197
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;198
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;199
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;200
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;201
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;202
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;203
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;204
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;205
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;206
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;207
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;208
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;209
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;210
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;211
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;212
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;213
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;214
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;215
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;216
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;217
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;218
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;219
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;220
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;221
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;222
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;223
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;224
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;225
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;226
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;227
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;228
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;229
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;230
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;231
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;232
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;233
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;234
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;235
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;236
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;237
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;238
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;239
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;240
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;241
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;242
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;243
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;244
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;245
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;246
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;247
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;248
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;249
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;250
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;251
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;252
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;253
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;254
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;255
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;256
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;257
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;258
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;259
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;260
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;261
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;262


TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,140,Black,015,Sync_,185,Blank ;263

TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;264
TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;265
TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;266
TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;267
TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;268
TVLINE 171,Sync_,029,Blank,171,Sync_,029,Blank,000,00000,000,00000,000,00000 ;269
TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;270
TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;271
TVLINE 015,Sync_,185,Blank,015,Sync_,185,Blank,000,00000,000,00000,000,00000 ;272
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;273
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;274
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;275
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;276
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;277
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;278
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;279
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;280
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;281
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;282
TVLINE 029,Sync_,371,Blank,000,00000,000,00000,000,00000,000,00000,000,00000 ;283
TVLINE 029,Sync_,171,Blank,191,Black,009,Blank,000,00000,000,00000,000,00000 ;284
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;285
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;286
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;287
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;288
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;289
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;290
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;291
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;292
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;293
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;294
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;295
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;296
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;297
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;298
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;299
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;300
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;301
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;302
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;303
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;304
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;305
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;306
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;307
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;308
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;309
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;310
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;311
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;312
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;313
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;314
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;315
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;316
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;317
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;318
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;319
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;320
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;321
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;322
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;323
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;324
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;325
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;326
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;327
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;328
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;329
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;330
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;331
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;332
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;333
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;334
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;335
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;336
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;337
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;338
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;339
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;340
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;341
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;342
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;343
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;344
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;345
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;346
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;347
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;348
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;349
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;350
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;351
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;352
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;353
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;354
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;355
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;356
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;357
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;358
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;359
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;360
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;361
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;362
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;363
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;364
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;365
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;366
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;367
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;368
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;369
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;370
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;371
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;372
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;373
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;374
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;375
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;376
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;377
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;378
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;379
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;380
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;381
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;382
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;383
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;384
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;385
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;386
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;387
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;388
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;389
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;390
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;391
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;392
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;393
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;394
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;395
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;396
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;397
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;398
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;399
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;400
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;401
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;402
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;403
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;404
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;405
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;406
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;407
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;408
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;409
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;410
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;411
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;412
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;413
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;414
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;415
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;416
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;417
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;418
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;419
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;420
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;421
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;422
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;423
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;424
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;425
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;426
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;427
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;428
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;429
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;430
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;431
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;432
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;433
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;434
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;435
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;436
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;437
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;438
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;439
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;440
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;441
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;442
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;443
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;444
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;445
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;446
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;447
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;448
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;449
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;450
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;451
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;452
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;453
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;454
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;455
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;456
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;457
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;458
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;459
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;460
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;461
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;462
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;463
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;464
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;465
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;466
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;467
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;468
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;469
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;470
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;471
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;472
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;473
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;474
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;475
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;476
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;477
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;478
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;479
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;480
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;481
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;482
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;483
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;484
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;485
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;486
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;487
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;488
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;489
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;490
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;491
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;492
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;493
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;494
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;495
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;496
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;497
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;498
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;499
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;500
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;501
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;502
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;503
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;504
```

```
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;505
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;506
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;507
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;508
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;509
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;510
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;511
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;512
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;513
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;514
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;515
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;516
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;517
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;518
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;519
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;520
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;521
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;522
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;523
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;524
TVLINE 029,Sync_,006,Blank,016,Burst,009,Blank,331,Black,009,Blank,000,00000 ;525

END
```

Appendix C INFORMATIE CAPACITEIT VAN EEN CONTINU COMMUNICATIE KANAAL

De maximaal mogelijke capaciteit van een continu communicatie kanaal per seconde kan worden gegeven als:

$$C = B * \log(1 + S/N) \qquad (1)$$

Deze vergelijking staat bekend als de Hartley-Shannon wet en wordt afgeleid in [Carlson blz 585 ev.]. Om een minimum waarde voor de signaal ruis verhouding te bepalen welke nodig zal zijn voor het foutloos versturen van een bepaalde hoeveelheid informatie moet deze hoeveelheid eerst worden berekend. We gaan er van uit dat de ontvanger onderscheid moet kunnen maken tussen M verschillende signaal niveau's. De totale hoeveelheid informatie door de bron per seconde gegenereerd is dan:

$$C_b = \frac{\log M}{T} = F_s * \log M \qquad (2)$$

Waarbij T de tijdsduur van een bemonstering is, en F_s de bemonsterings frequentie.

Indien de capaciteit van de bron als minimum van de kanaal capaciteit wordt gekozen, is het mogelijk om een minimum waarde voor S/N te bepalen.

$$F_s * \log M = B * \log(1+S/N) \qquad (3a)$$

dus:

$$S/N = 10^{(F_s/B * \log M)} - 1 \qquad (3b)$$

Nu er een minimum waarde voor de signaal ruis verhouding is bepaald kunnen de volgende 2 modulatie systemen worden vergeleken. Beide systemen gaan uit van 4 bits Adaptive Differential Pulse Code Modulation (ADPCM) codering. Het eerste systeem gebruikt bij modulatie 16 ($=2^4$) verschillende symbolen. Het tweede systeem maakt gebruik van

kwadratuur modulatie en gebruikt dus per draaggolf slecht 4 ($=2^2$) symbolen.

Vergelijking directe kwantisering met kwadratuur

Voor deze vergelijking wordt uitgegaan van de volgend systeem constanten:

$F_s = 6.8\ 10^6$ }
} Algemeen: $F_s / B = 2$
$B = 3.4\ 10^6$ }

In het geval van directe codering met 4 bits (M=16 niveau's) geldt, met gebruik van (3b):

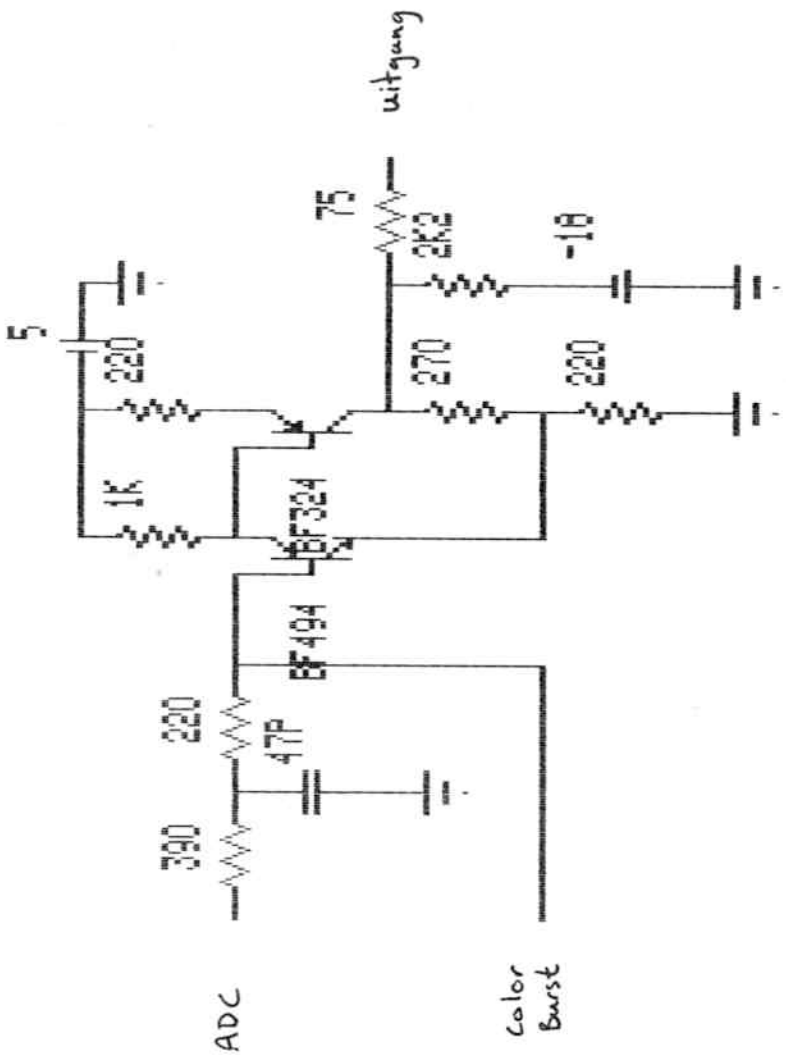
$S/N = 255$, in dB uitgedrukt wordt dat ongeveer 24 dB.

In het geval van codering bij kwadratuur modulatie zijn er slecht 2 bits per signaal, dus M=4, nu geldt:

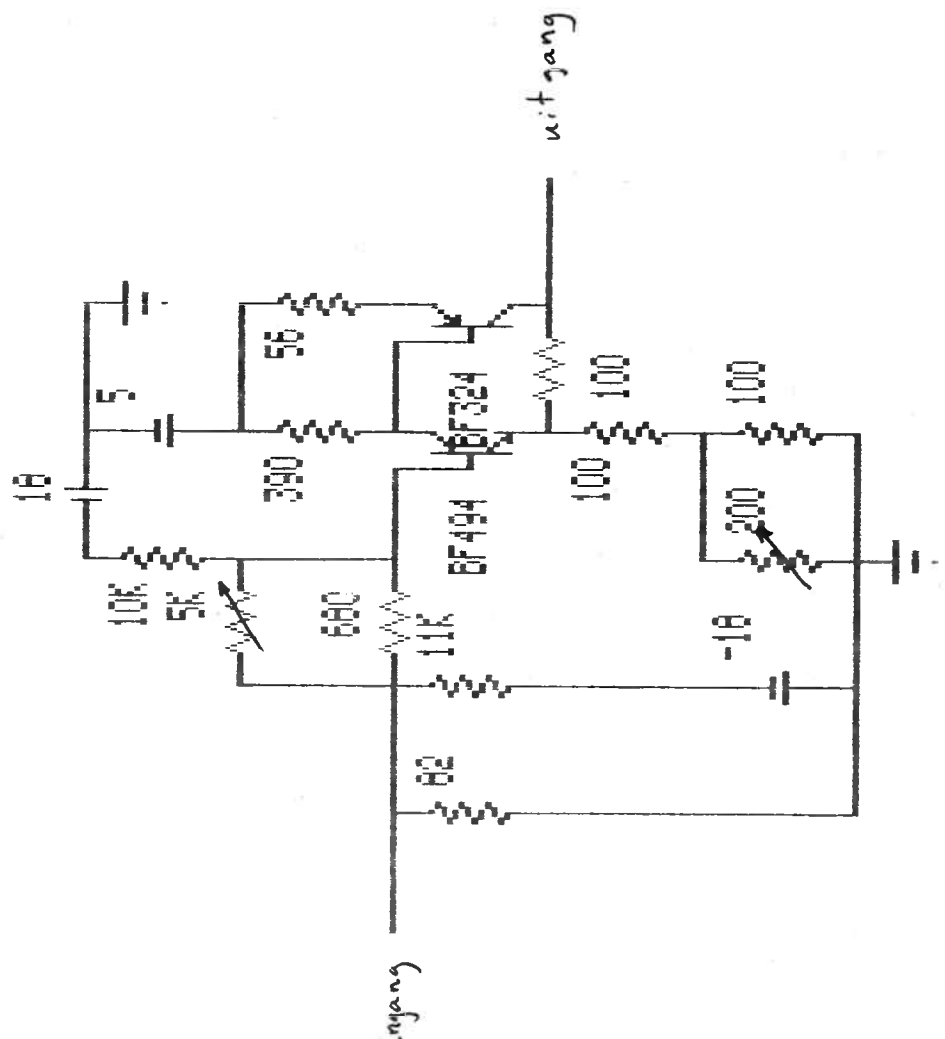
$S/N = 15$, in dB uitgedrukt wordt dat ongeveer 11 dB

Hiermee is aannemelijk gemaakt dat kwadratuur modulatie een betere performance zal geven voor wat betreft het aantal fouten veroorzaakt door ruis storingen tijdens transmissie van het analoog signaal. De berekende uitkomsten zijn slechts geldig in het ideale geval van een omgeving met slechts witte ruis en verder geen storingen of niet-lineairiteiten van apparatuur.

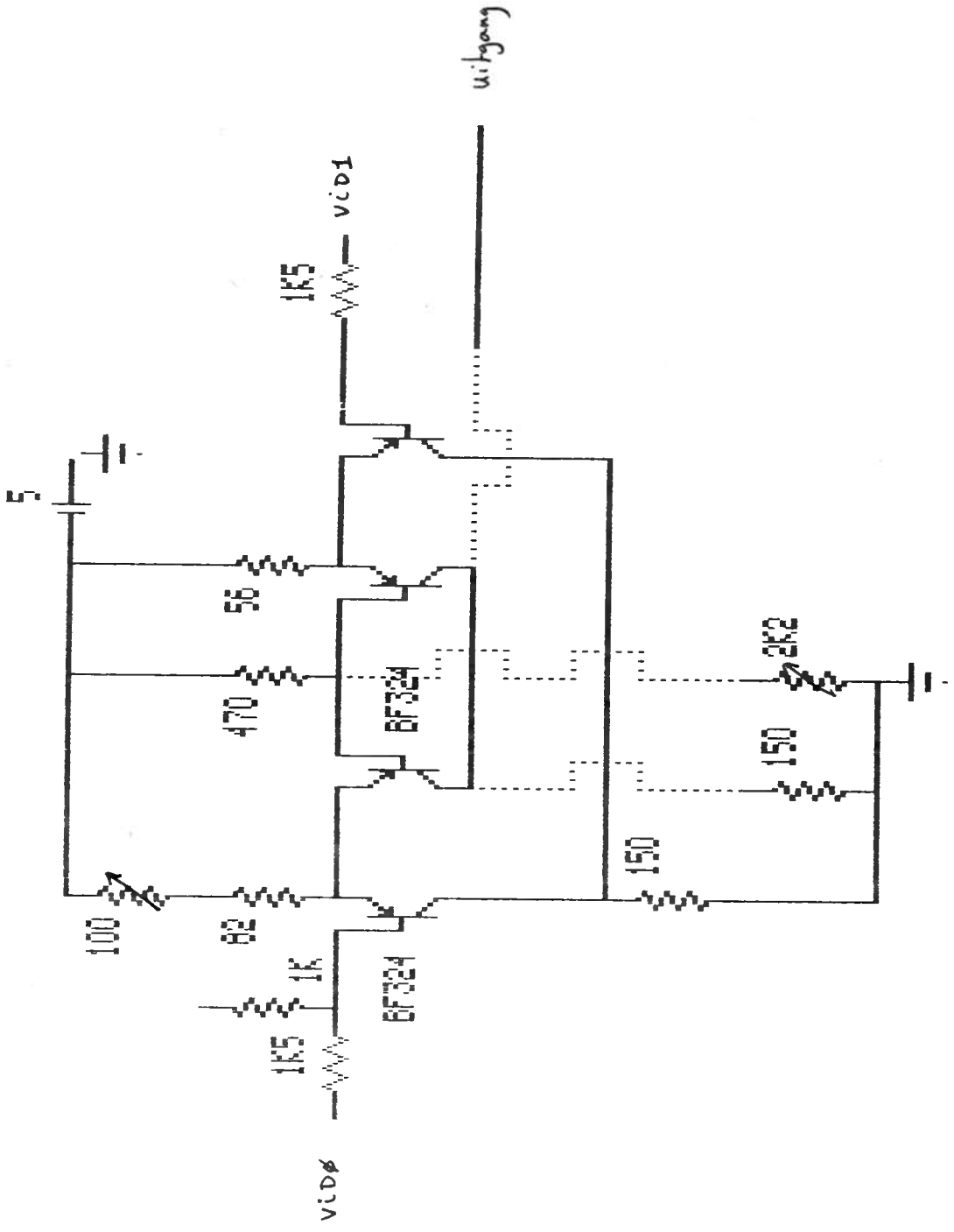
Appendix D ENIGE SCHAKELINGEN VAN HET DSU SYSTEEM



Figuur D.1 Een analoge correctie en optellingsschakeling aan de uitgang van de DAC van de encoder van het DSU systeem.



Figuur D.2 Een analoge correctieschakeling voor de ingang van de ADC van de decoder van het DSU syteem.



Figuur D.3 Een proefschakeling voor de omzetting van 2 digitale signalen (VID0 en VID1) tot een 4 niveau signaal (uitgang) ten behoeve van een kwadratuur modulator.

Appendix E DOCUMENTATIE UVC-3100

UVC 3100, UVC 3101
High-Speed
A/D-D/A Converters

6251-240-5E

semic...

ITT

.igh-Speed A/D-D/A Converters

VLSI circuits in CI technology, featuring the following circuits:

- a high-speed flash type 8-bit A/D converter
- a high-speed low-glitch 10-bit D/A converter, designed as an R-2R network with switched current sources
- various auxiliary circuits, as reference voltage sources, preamplifier, input clamping circuit, and feed-in output amplifier.

UVC 3100 and UVC 3101 are different only in the linearity of the D/A converter. In the case of UVC 3100, the differential linearity is ± 1/2 LSB (referred to 10 bit), and with the UVC 3101 it is ± 1/2 LSB (referred to 8 bit).

UVC 3100 and UVC 3101 have been developed for use in all applications which call for a high-speed A/D-D/A converter. For instance, these circuits can be used to advantage to decode television signals in Pay-TV converters or for D2-MAC converters used in direct satellite broadcast. Other promising applications can be seen in industrial electronics, e. g. in conjunction with digital signal processing. Although UVC 3100 and UVC 3101 were initially designed as high-speed codecs for the video-range, they can be used with equal benefits for lower frequencies, even down to zero.

1. General

The above auxiliary circuits contained on-chip provide versatile potential applications needing a minimum of external components. For example, an impedance converter is connected upstream of the A/D converter to provide a high-impedance signal input, in spite of the high input capacitance of the A/D converter. The reference voltage for the A/D converter is generated on-chip, but both the ground of that circuit and the reference voltage are fed to pins, so that an external filter capacitor may be connected. Further, the input is equipped with switches which optionally provide operation with keyed clamping or peak clamping or without clamping (see also section 6.).

Also the D/A converter's reference voltage is generated on-chip, and a gated amplifier is arranged at the output of the D/A converter so that an external analog signal can be fed-in instead of the signal delivered by the D/A converter.

Separate clock inputs are provided for the A/D converter and the D/A converter thus enabling the application of time compression procedures. All inputs and outputs are TTL compatible.

2. Outline Dimensions and Pin Connections

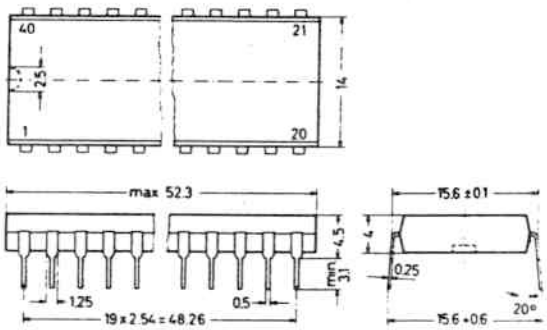


Fig 2:
UVC 3100/UVC 3101 in 40-pin Dil Plastic Package, 20 B 40 according to DIN 41866

Weight approx. 6 g
Dimensions in mm

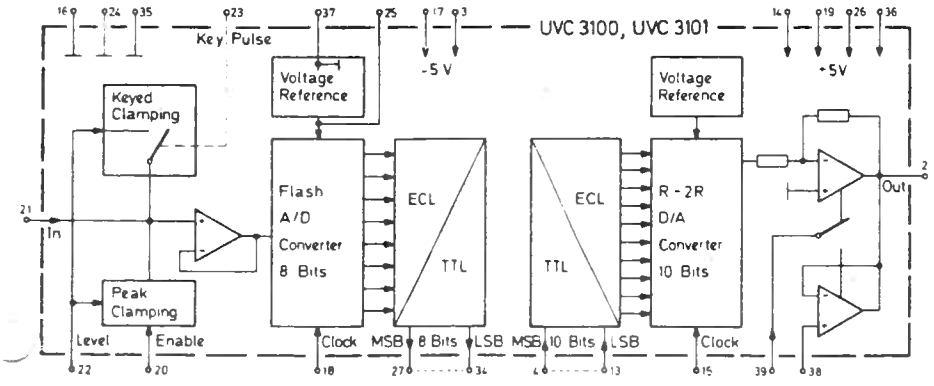


Fig. 1: UVC 3100 and UVC 3101 block diagram

Pin Connections

1 Leave Vacant
2 Analog Output D/A Converter
3 −5 V Supply D/A Converter analog
4 Digital Input Bit 9 (MSB)
5 Digital Input Bit 8
6 Digital Input Bit 7
7 Digital Input Bit 6
8 Digital Input Bit 5
9 Digital Input Bit 4
10 Digital Input Bit 3
11 Digital Input Bit 2
12 Digital Input Bit 1
13 Digital Input Bit 0 (LSB)
14 +5 V Supply D/A Converter digital
15 Clock Input D/A Converter
16 GND D/A Converter and Clock A/D Converter
17 −5 V Supply A/D Converter analog
18 Clock Input A/D Converter
19 +5 V Supply A/D Converter
20 Peak Clamping Enable Input
21 Analog Input A/D Converter
22 Clamping Level Input
23 Key Pulse Input
24 Analog Ground A/D Converter
25 Reference Voltage A/D Converter
26 +5 V Supply A/D Converter digital
27 Digital Output Bit 7 (MSB)
28 Digital Output Bit 6
29 Digital Output Bit 5
30 Digital Output Bit 4
31 Digital Output Bit 3
32 Digital Output Bit 2
33 Digital Output Bit 1
34 Digital Output Bit 0 (LSB)
35 Digital Ground A/D Converter
36 +5 V Supply A/D Converter analog
37 GND of Ref. Voltage A/D Converter
38 External Analog Input
39 Output Signal Switchover Input
40 Leave Vacant

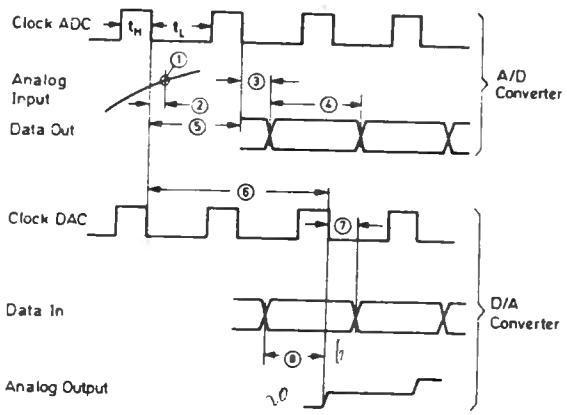


Fig. 3:
Timing diagram of the UVC 3100/UVC 3101 A/D-D/A Converters
① Sample
② Aperture delay
③ Digital output delay
④ Data valid (after sample ①)
⑤ Transfer time A/D
⑥ Total transfer time A/D-D/A with common clock
⑦ Input register hold time
⑧ Input register setup time

3. Electrical Characteristics

All voltages are referred to pins 16, 24, 35 and 37.

Absolute Maximum Ratings

	Symbol	Value	Unit
Positive Supply Voltage	$+V_B$	6	V
Negative Supply Voltage	$-V_B$	6	V
Input Voltages			
Digital Inputs	V_I	-0.5 V to $(+V_B +0.5$ V)	–
Analog Input	V_I	-0.5 V to $(+V_B +0.5$ V)	–
Output Current Pin 2	I_O	± 10	mA
Ambient Operating Temperature Range	T_A	0 to +65	°C
Storage Temperature Range	T_S	−40 to +125	°C

Recommended Operating Conditions

	Symbol	Min.	Typ.	Max.	Unit
Positive Supply Voltage	$+V_B$	4.75	5	5.25	V
Negative Supply Voltage	$-V_B$	4.75	5	5.25	V
A/D Converter					
Analog Input Voltage	V_I	0	–	2	V
Input Frequency, Analog Input	f_I	–	–	$< \frac{f_{cl}}{2}$	–
Clock Amplitude	V_{18H}	2.4 V	–	$+V_B$	–
	V_{18L}	0	–	0.8	V
Clock Frequency	f_{18}	0	–	38.5	MHz
Clock High Time (see Fig. 3)	t_H	8.0	–	–	ns
Clock Low Time (see Fig. 3)	t_L	18.0	–	–	ns
Clamping Level	V_{22}	−1	–	+2	V
Key Pulse	V_{23H}	2.4 V	–	$+V_B$	–
	V_{23L}	0	–	0.8	V
Activation of Peak Clamping	Resistor of 20 to 60 kΩ from Pin 20 to +5 V				–
D/A Converter					
Clock Amplitude	V_{15H}	2.4 V	–	$+V_B$	–
	V_{15L}	0	–	0.8	V
Clock Frequency	f_{15}	0	–	38.5	MHz
Digital Input Voltages	V_{IH}	2.4 V	–	$+V_B$	–
	V_{IL}	0	–	0.8	V
Analog Input Voltage at pin 38	V_{38}	−1	–	+3	V
Control Voltage for the Output Gate Amplifier					
Input Signal from Pin 21 at the Output Pin 2	V_{39}	0	–	0.8	V
Input Signal from Pin 38 at the Output Pin 2	V_{39}	2 V	–	$+V_B$	–

Characteristics at $+V_B = 5$ V, $-V_B = 5$ V, $f_{15} = 35$ MHz, $f_{18} = 35$ MHz, $T_A = 25$ °C

	Symbol	Min.	Typ.	Max.	Unit
Current Consumption	I_B	–	–	120	mA
	$-I_B$	–	–	110	mA
Power Dissipation	P_{tot}	–	–	1.2	W
Total Transfer Time A/D-D/A (⑥ in Fig. 3)	t_{tot}	–	see Fig. 3	–	–
A/D Converter					
Input Current Pin 21	I_I	–	1	–	µA
Input Impedance Pin 21 at f = 1 kHz	Z_I	–	20	–	MΩ
at f = 10 MHz	Z_I	–	100	–	kΩ
Input Capacitance Pin 21	C_I	–	10	–	pF
3 dB Bandwidth of the Input Amplifier	–	–	50	–	MHz
Clamping Active at	V_{23}	2.4	–	–	V
ON Resistance of the Clamping Switch Between Pins 21 and 22	R_{on}	–	300	–	Ω
Input Current of the Clamping Level Input Pin 22	I_{22}	–	200	–	µA
Aperture Delay (② in Fig. 3)	t_{sd}	–	–	10	ns
Digital Output Delay (③ in Fig. 3)	t_{dv}	–	–	14	ns
Transfer Time (⑤ in Fig. 3)	t_{tr}	–	one Clock Period	–	–
Differential Non-Linearity	–	–	±1/2 LSB	–	–
Absolute Non-Linearity	–	–	1	–	%
Number of Bits	–	–	8	–	–
Code of the Digital Output Signals	–	–	binary	–	–
Output Signal					
at $V_{21} = 0$ V	–		0 0 0 0 0 0 0 0		–
at $V_{21} = V_{ref}$	–		1 1 1 1 1 1 1 1		–
Internal Reference Voltage, accessible from outside	V_{25}	–	2.0	–	V
D/A Converter					
Input Register Hold Time (⑦ in Fig. 3)	t_{ih}	6.0	–	–	ns
Input Register Setup Time (⑧ in Fig. 3)	t_{is}	20	–	–	ns
Differential Non-Linearity					
UVC 3100 (referred to 10 bit)	–	–	±1/2 LSB	–	–
UVC 3101 (referred to 8 bit)	–	–	±1/2 LSB	–	–
Absolute Non-Linearity	–	–	1	–	%
Number of Bits	–	–	10	–	–
Code of the Digital Input Signal	–	–	binary	–	–
Output Signal					
with 0 0 0 0 0 0 0 0 0 0 0 at the Inputs	V_2	–	0	–	V
with 1 1 1 1 1 1 1 1 1 1 1 at the Inputs	V_2	–	2	–	V
Output Impedance Pin 2	Z_O	–	15	–	Ω
Input Current Pin 38	I_I	–	1	–	µA
Internal Reference Voltage	V_{ref}	–	2	–	V

.. Inner Configuration of the Connection Pins

The following figures schematically show the circuitry at the various pins.

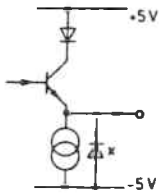


Fig 4:
Pin 2, Output

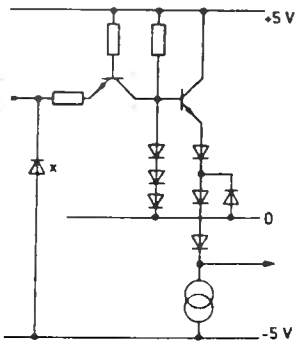


Fig. 5:
Pins 4 to 13, 15, 18, 23 and 39, Inputs

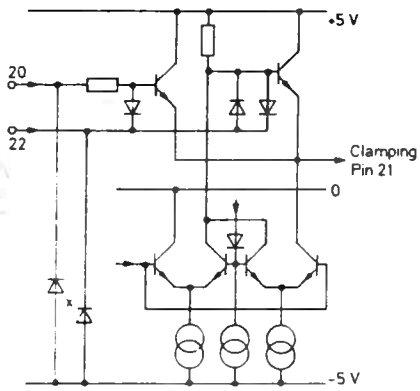


Fig. 6:
Pins 20 and 22, Inputs

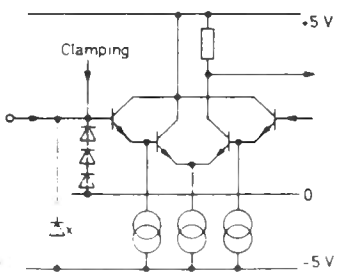


Fig. 7:
Pin 21, Input

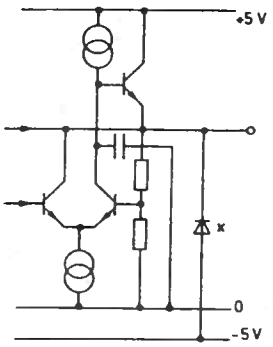


Fig. 8:
Pin 25, Reference Voltage Pin

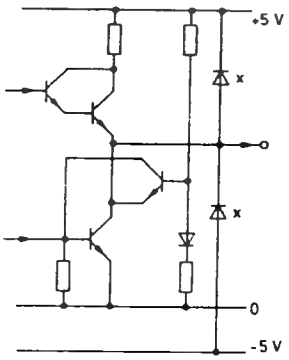


Fig. 9:
Pins 27 to 34, Outputs

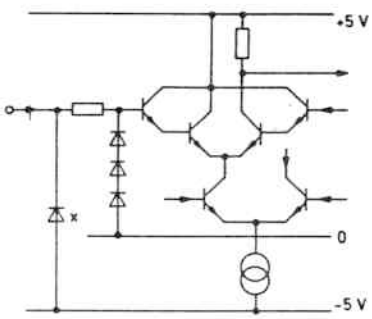


Fig. 10:
Pin 38, Input

× = protection diode

5. Description of the Connections and the Signals

Pin 2 – Analog Output D/A Converter
This pin whose diagram is shown in Fig. 4, is the output for the processed analog signal either originating from the D/A converter or from the external analog input pin 38.

Pin 3 – –5 V Supply D/A Converter Analog
This pin gets the negative supply for the analog part of the D/A converter.

Pins 4 to 13 – Digital Inputs Bit 9 to Bit 0
The diagram of these pins is shown in Fig. 5. They are the inputs of the D/A converter. Not-used inputs should be connected to ground.

Pin 14 – +5 V Supply D/A Converter Digital
This pin gets the positive supply for the digital part of the D/A converter.

Pin 15 – Clock Input D/A Converter
This pin whose diagram is shown in Fig. 5 must be supplied with the clock signal for the D/A converter.

Pin 16 – GND D/A Converter and Clock A/D Converter
This pin serves as ground pin for the D/A converter and for the clock of the A/D converter.

Pin 17 – –5 V Supply A/D Converter Analog
This pin is the negative supply pin for the analog part of the A/D converter.

Pin 18 – Clock Input A/D Converter
The diagram of this pin is shown in Fig. 5. Pin 18 is supplied with the clock of the A/D converter.

Pin 19 – +5 V Supply A/D Converter
Via this pin the A/D converter gets its positive supply.

Pin 20 – Peak Clamping Enable Input
Via pin 20 whose diagram is shown in Fig. 6, the peak clamping facility can be enabled.

Pin 21 – Analog Input A/D Converter
Fig. 7 is the diagram of this input. To pin 21 is applied the analog signal to be convered into digital.

Pin 22 – Clamping Level Input
Via this pin whose diagram is shown in Fig. 6, the input of the A/D converter is supplied with the desired clamping level.

Pin 23 – Key Pulse Input
Fig. 5 is the diagram of this input. Pin 23 must be supplied with the key pulse if keyed clamping is required.

Pin 24 – Analog Ground A/D Converter
This pin serves as ground pin for the analog part of the A/D converter.

Pin 25 – Reference Voltage A/D Converter
This pin whose diagram is shown in Fig. 8, is intended for connecting a decoupling capacitor to the A/D converter's reference voltage, the other end of this capacitor to pin 37.

Pin 26 – +5 V Supply A/D Converter Digital
This pin is the positive supply pin for the digital part of the A/D converter.

Pins 27 to 34 – Digital Outputs Bit 7 to Bit 0
Fig. 9 shows the diagram of these outputs which supply the digitalized analog signal in parallel 8-bit code.

Pin 35 – Digital Ground A/D Converter
This pin is the ground connection for the digital part of the A/D converter.

Pin 36 – + 5 V Supply A/D Converter Analog
This pin is the positive supply pin for the analog part of the A/D converter.

Pin 37 – Ground of Reference Voltage A/D Converter
To this pin must be connected the ground end of the decoupling capacitor which is at pin 25.

Pin 38 – External Analog Input
The diagram of this input is shown in Fig. 10. Pin 38 serves for feeding an external analog signal into the output amplifier of the UVC 3100/UVC 3101 instead of the D/A-converted signal originating from pins 4 to 13.

Pin 39 – Output Signal Switchover Input
This pin whose diagram is shown in Fig. 5, is intended for enabling the external analog signal fed to pin 38.

6. Appendix: Application Circuits

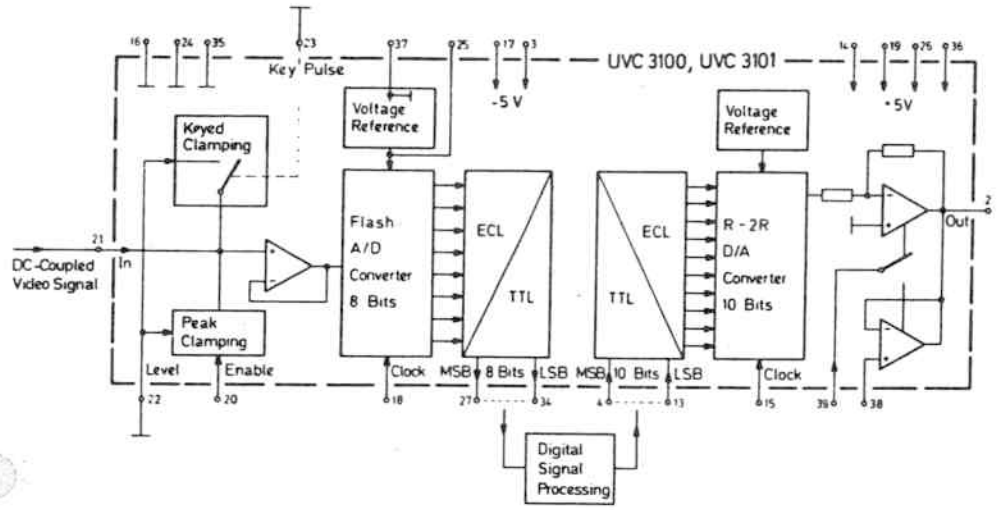


Fig. 11: Operation without clamping of the input signal
Pin 20 must not be connected.

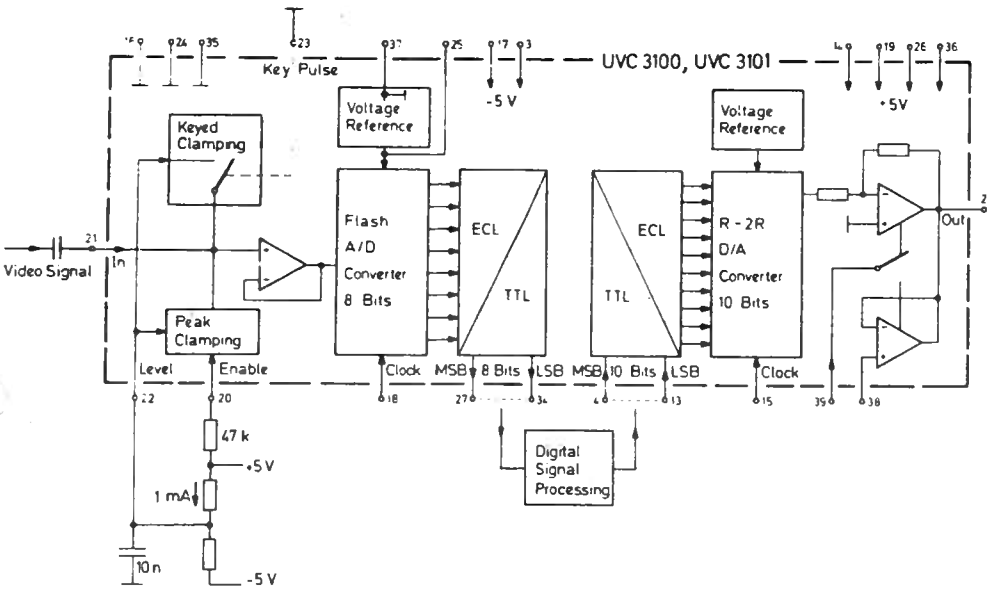


Fig. 12: Operation with peak clamping
The input signal is clamped automatically to the negative peak value. No key pulse is needed.

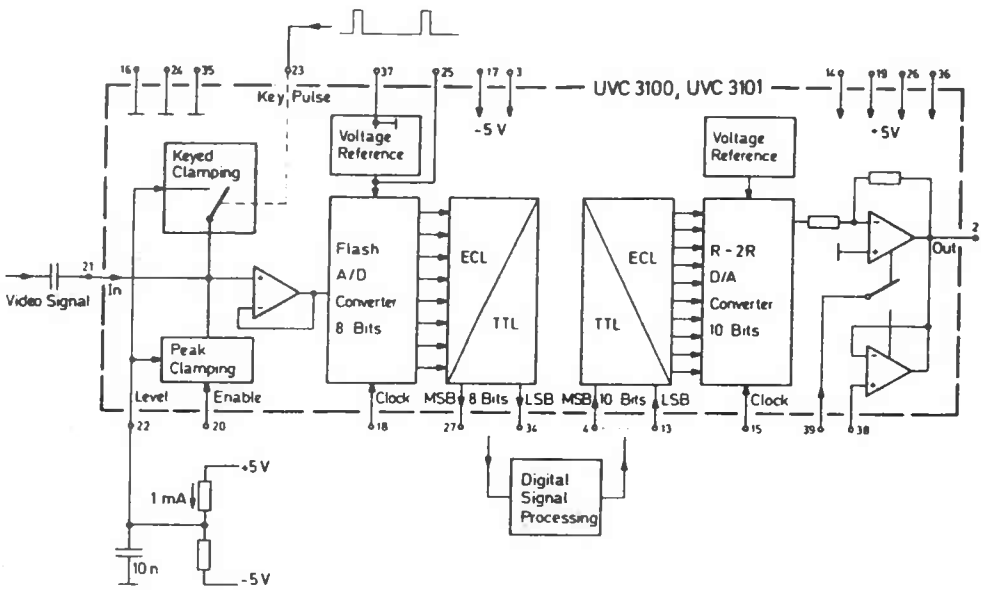


Fig. 13: Operation with keyed clamping
During the key pulse, the input signal must have that level to which it is to be clamped. Pin 20 must not be connected.

Appendix F ADPCM EQUIPMENT FOR 9.6 KBPS DATA

ADPCM Equipment for 9.6-kbps Data

The ADPCM algorithm proposed by OKI Electric of Japan seems to be a formidable alternative for the standard.

■ Yoshihiko Yokoyama

In 1982, the CCITT started work on developing a second digital encoding standard for speech, after decades of extensive use of PCM at 64 kbps in the A-law or μ-law formats. The result of that effort was the encoding standard of the the 32-kbps ADPCM algorithm, known as CCITT Recommendation G.721. It was recognized from the beginning that the algorithm must support speech applications in the network. However, it was also decided that the algorithm should maintain adequate performance for voice-band data signals, although it was acknowledged that such signals were limited to data rates of up to 4800 bps for the state-of-the-art ADPCM algorithms. This has resulted in a virtual hesitation of widespread application of the standard in the public switched telephone networks (PSTNs), for which it was intended. Network operators have concluded that a fast-growing need exists for transmitting data at 9600 bps for their customers, and using G.721 makes that impossible.

Subsequently, the CCITT has embarked on a course of defining a digital encoding standard for digital circuit multiplication equipment (DCME), which combines time assignment speech interpolation (TASI) and a low-rate encoding technique such as ADPCM to form a very efficient means of transmitting speech. How to transmit data in such a system has been the subject of considerable debate and extensive effort by many experts in the field. It should be pointed out that, similar to the transcoding standard of G.721, interfacing with the DCME must be accomplished by means of an A-law or μ-law encoded PCM signal format.

The need for transmitting data up to 9600 bps has been recognized, and three algorithms have undergone scrutiny by a group of experts in the field. Two of the algorithms have the inherent capability of transmitting 9600-bps voice-band data at the 32-kbps rate, whereas the third algorithm under consideration is G.721, which does not have that capability.

■ PRESENT STANDARDIZATION EFFORTS

DCME Aspects

A DCME system is basically an all-digital implementation of the old concept of TASI. DCME systems operate on the statistical behavior of a group of talkers in a communication system. This is characterized by the average time that a talker on a connection is actually *active*, nominally assumed to be 35-40 percent of the total time the circuit is used for a call. Thus, the remaining time is available for time-interleaving the speech of other talkers. On the average, circuit usage can be increased or multiplied by a factor called digital speech interpolation (DSI) gain. Gain factors between 2 and 2.5 are commonly used, but these gain factors are dependent on the actual speech activity exhibited by the talkers. The larger the group of talkers, the more statistical stability is attained, and individual fluctuations in speech activity can be accommodated by the system. Long talk spurts by one talker are simultaneously compensated by silence or short spurts by another.

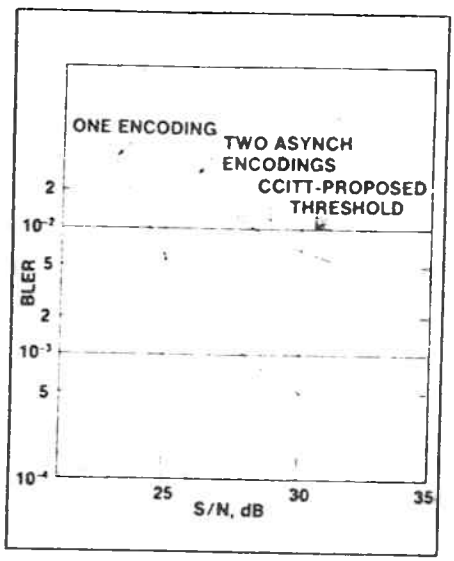


Fig. 1 ADPCM performance with a V.29 modem.

Short durations of active speech, more than can be accommodated by available transmission capacity, do occur. Without "special means," this would result in what is known as clipped speech. In DCME, this special means is provided by instantly reducing the coding rate of one or more channels (talkers). That is, when the DCME operates nominally with ADPCM at 32 kbps dur-

ing overload, this rate is reduced to 24 kbps for one or more channels. As sampling occurs at 8000 times per second, this means that the nominal channel being encoded at 4 bits/sample is reduced to encoding at 3 bits/sample during overload. This brings about a small degradation in performance by increased quantizing noise, but it occurs only sporadically due to the statistical nature of the phenomenon. Therefore, it is virtually imperceptible as long as the signal load to the DCME is strictly speech. When an appreciable part of the DCME load is data (more than 20 percent), special precaution must be taken to prevent noticeable degradation, because data signals do not exhibit the same on-off activity as speech. In fact, data are considered, generally, as being 100 percent active, thus providing no bearer circuit-sharing capability.

When the DCME load is a mix of speech and data, it is clear that overload will occur more often for the speech signals, resulting in an associated decrease in performance in the form of higher quantizing distortion. The choice of ADPCM algorithm for the DCME has an important bearing on this problem.

■ CCITT EFFORTS

The CCITT is considering using the basic G.721 algorithm for speech at 32 kbps for DCME, but due to that algorithm's inability to handle 9600-bps data at 32 kbps, encoding at 40 kbps per channel is needed for data signals at such rates. This is clearly having a more profound influence on the use of available bearer transmission capacity than if encoding of data could be limited to using the 32-kbps bearer rate per channel. For example, a 60-channel DCME system employing a proprietary ADPCM developed by OKI Electric of Japan can accommodate 10 percent data traffic up to 9.6 kbps, whereas G.721 ADPCM can only accommodate 6.7 percent data and maintain the same speech performance. Moreover, the DCME design is considerably simpler with the proprietary ADPCM, since there is no need to reconfigure the frame structure for including 5-bit/sample encoding for data.

Another aspect of ADPCM in DCME systems is the need to tandem such systems for multilink networking purposes. It can generally be argued that no more than two DCME links should be allowed to be switched in any end-to-end connection. If such switching is performed by an analog switch (asynchronous tandeming), an accumulation of distortion will be experienced in the second link. However, if a digital switch would be em-

ployed, directly operating on the PCM output of the first DCME link, passing it digitally on to the second link (synchronous tandeming), no additional distortion will be experienced. Both the OKI ADPCM and the G.721-related technique in DCME application will have the "synchronous" capability as an inherent part of the design. A third algorithm, mentioned earlier, does not possess that capability, and it will not be discussed.

Digital switching will increasingly be employed in the public networks. Therefore, the loss of performance due to asynchronous tandeming, if it occurs at all, may only be temporarily experienced and should not pose a serious concern. This aspect of tandeming is not uniquely related to DCME systems. Any application of 32 kbps could encounter the need for tandeming in a network. As digital switching will be increasingly applied, either by replacing analog switches or in new installations, the advantage of the ADPCM technique will be even more evident because of its capability of transmitting up to 9.6-kbps voice-band data signals.

The CCITT nevertheless has decided to hold on to the G.721 technique, even though a clearly superior technique is now available.

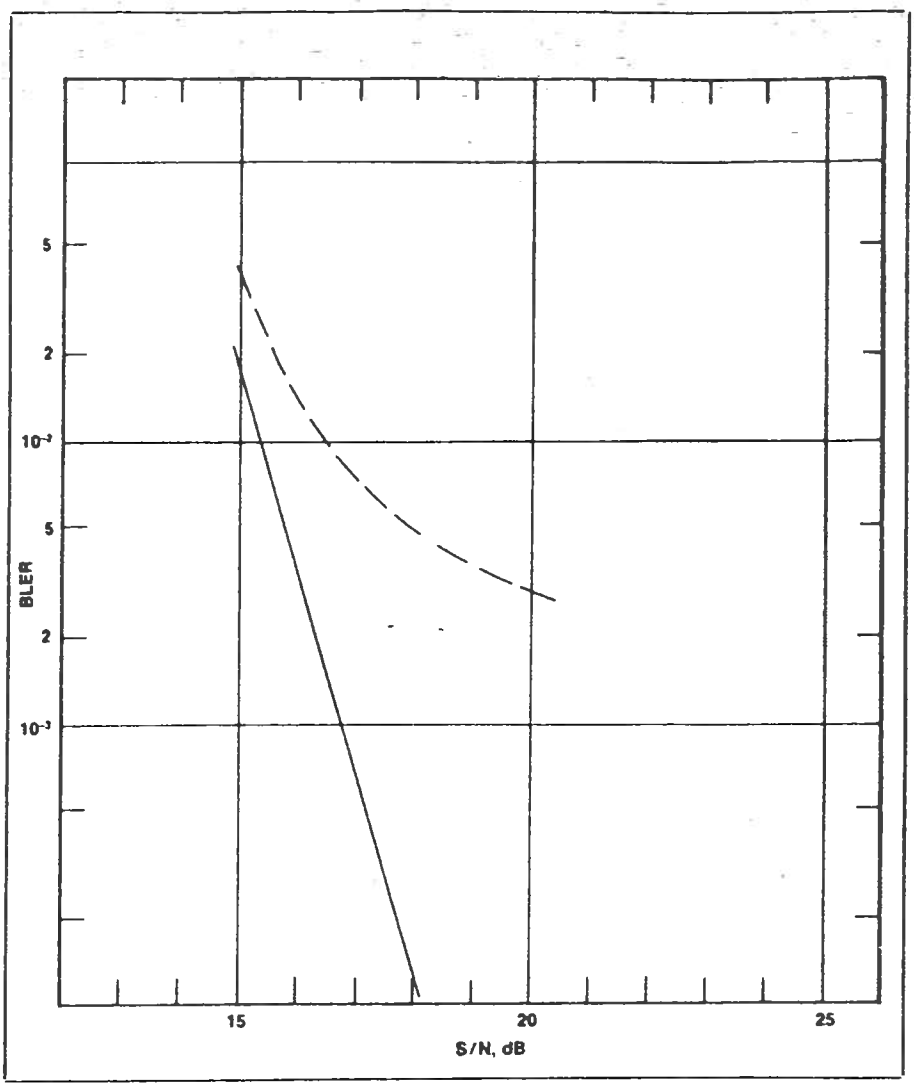


Fig. 2 The solid curve represents ADPCM performance with a V.29 modem operating at 4.8-kbps tandem through four asynchronous encodings of the OKI ADPCM equipment. The dashed curve represents the performance of the same modem when four asynchronous links of G.721 ADPCM equipment are substituted for the OKI equipment.

■ OKI ADPCM PERFORMANCE

Data

Extensive performance measurements have been made in a carefully assembled test bed at COMSAT Laboratories.* The circuit in which the ADPCM equipment was tested included a simulated analog access link which introduced typical distortion effects (analog impairments) that a voice-band data signal may experience before being encoded by the ADPCM link.

The typical performance after encoding by OKI ADPCM of a CCITT V.29 modem** in terms of block error rate (BLER), as a function of S/N ratio of the data signal in the analog impairment circuit (i.e, just before being encoded), is illustrated in **Figure 1**. The lower curve shown resulted after a single ADPCM encoding, whereas the higher curve resulted after a second ADPCM link was added to the first by means of an analog interconnection between the two links. Thus, this second curve is the result of asynchronous tandeming of two links. The curve showing single encoding performance applies also for the case of multiple encodings via digital switches, referred to as synchronous tandeming. A reference performance threshold of BLER = 10^{-2} at S/N = 30.5 dB† is well met by both curves. This indicates the excellent capability of the ADPCM equipment for transmitting 9.6-kbps V.29 signals.

*This test bed received approval by the organizations that submitted ADPCM equipment for evaluation and comparison in a CCITT context. This made comparisons between algorithms valid and accurate

**The V.32 modems will perform even better than V.29 modems because of their inherent design. Thus, V.29 performance shown here is more critical to the user.

†This reference point was selected by an SG XVIII expert group.

The performance of a V.29 modem operating at the back-off rate of 4.8-kbps tandem through four asynchronous encodings of the ADPCM equipment is shown in **Figure 2**. For comparison, the dashed curve in **Figure** 2 shows the performance of the same modem when four asynchronous links of G.721 ADPCM equipment are substituted for the OKI equipment. At S/N values to be expected in the networks, the OKI advanced ADPCM can perform two or more orders of magnitude better than G.721. This may not be required for this modem speed, but it is simply a consequence of its inherently more powerful predictor than that employed in G.721, and, as such, it provides an increased performance margin.

Voice

When considering ADPCM designs, the primary purpose has always been to provide high performance for voice signals. This objective has unquestionably been attained by the G.721 designers Extensive subjective tests have proven the algorithm delivers the speech performance required for the networks.

The primary purpose has been to provide high performance for voice....

Similarly, the OKI ADPCM equipment provides the required high performance when speech is transmitted through it. Tests similar to those used for evaluating the G.721 algorithm have been performed with the OKI ADPCM equipment, particularly for the English and Japanese languages.

DCME Gain

As has been pointed out earlier in the article, when applied in DCME systems, the proprietary ADPCM technique offers the advantage of encoding all voice-band data by using only 4 bits/sample. This offers a bearer-channel efficiency advantage of up to 20 percent when transmitting 60 channels with 20 percent data. This includes a bearer-capacity increase to avoid speech degradation. Such an advantage may be particularly important for countries that may want to minimize their cost of communication.

It should be emphasized, however, that without DCME, the main advantage of the proprietary ADPCM resides in its capability of transmitting up to 9.6-kbps voice-band data. This has an important bearing on networks, since meeting this requirement is or will become indispensable. □

Yoshihiko Yokoyama *is the General Representative for OKI America, Inc., New York office.*

Appendix G COMMUNICATIE PROGRAMMA "DSU UTILITY"

Het DSU Utility communicatie programma is geschreven voor gebruik op een IBM (of Compatible) Personal Computer (PC). Het maakt de communicatie mogelijk tussen zowel de encoder als de decoder van het DSU systeem. Alvorens in te gaan op het programma zelf, zal ten eerste het communicatie protocol met het DSU systeem worden beschreven.

DSU communicatie protocol

Ten behoeve van besturing en controle van buitenaf is er bij zowel de encoder als de decoder (encoder/decoder voortaan genoemd codec) van het DSU systeem één RS-232 in- en uitgangspoort aanwezig. Deze RS-232 poort wordt aan de kant van de codec, door het besturingssysteem van de codec ondersteund. Dit houdt in dat er vanaf de buitenwereld met de DSU codec via die RS-232 poort gecommuniceerd kan worden. Het RS-232 protocol houdt in: 2400 baud full duplex, even parity, 1 stopbit en 7 bits data-woordlengte. De DSU codec onderhoudt met de buitenwereld een heel eenvoudig software protocol op karakter niveau.

Dit protocol kan als volgt worden beschreven:

1. De DSU codec zal ieder te ontvangen legaal karakter na correcte verwerking terugsturen (echo). Bij ontvangst van een illegaal karakter, of het niet kunnen verwerken van een legaal karakter zal het besturingssysteem een 'X' terugsturen.
2. Legale karakters zijn:
   Voor zowel de encoder als de decoder:
   '0'..'9','A'..'F','G','H','I','J','N','S','T','V','W'
   Voor alleen de encoder:
   'K'
   Voor alleen de decoder:
   'O','P','Q'
3. Verwerking van de karakters '0'..'9','A'..'F' betekent het in een buffer plaatsen van de ontvangen karakter. De buffer moet, voor zover nodig, reeds gevuld zijn bij ontvangst van een legaal karakter ongelijk aan '0'..'9','A'..'F'.

4. Verwerking van de overige legale karakters houdt in het uitvoeren van een functie behorende bij het ontvangen karakter.
5. De DSU codec mag geen communicatie initieren.

De ontvangen karakters '0'..'9','A'..'F' worden gezien als de hexadecimale representatie van een binair getal. Bijvoorbeeld, de binaire representatie van een 8 bits integer:

0100 1110

Deze 8 bits (1 byte) kunnen hexadecimaal worden voorgesteld als:

4E

dit getal zal worden verstuurd als 2 bytes te weten:

'4' 'E' = ascii 52 en 69

Na ontvangst door de DSU-Utility of een codec zullen de 2 ontvangen bytes moeten worden getransformeerd naar één 8 bits byte.

DSU-Utility programma

Het DSU-Utility programma is geschreven in PASCAL, gebruikmakend van de Turbo pascal versie 3.0 ontwikkel omgeving van de firma Borland.
Het DSU-Utility programma kent de volgende voordelen:

- Op gebruikersniveau is het communicatieprotocol volledig afgeschermd.
- Het programma heeft de mogelijkheid om alle belangrijke gegevens van het DSU systeem te bekijken en/of te wijzigen.
- Het programma kan quasi gelijktijdig met zowel een encoder als een decoder communiceren.
- Het programma heeft de mogelijkheid om op byte niveau te communiceren ten behoeve van "debugging".
- Het programma is zeer flexibel opgezet voor wat betreft de gebruikers interface en de definitie van legale commando's welke

verstuurd mogen worden. Dit is gedaan omdat het programma wordt gebruikt in een omgeving waar de specificaties van gebruikers en de betekenis van functies van legale karakters aan verandering onderhevig zijn.

- Het programma is robuust (bestand tegen foutief invoer) en foutarm ontworpen omdat het ook zal worden gebruikt buiten de ontwikkel omgeving van het DSU systeem.

Het DSU-Utility programma kent de volgende nadelen:

- Het programma ondersteunt geen dataflow controle. Een software XON/XOFF protocol is niet ingebouwd omdat het DSU systeem deze ook niet ondersteunt.
- Het programma kan niet als achtergrond proces worden geactiveerd. Interactief gebruik is verplicht.

```
(*$i-*)
(*$v-*)
program com80188;

const
   esc_scan = 27;
type
   xchar = string[5];
   str3  = string[3];
   str6  = string[6];
   str7  = string[7];
   str12 = string[12];
   str13 = string[13];
   str40 = string[40];
   str80 = string[80];
   str255 = string[255];

   dmarec = packed array[1..128]of byte;
   dmapntr=^dmarec;

   RegPack = record case integer of
                1:( AX, BX, CX, DX, BP, SI, DI, DS, ES, Flags: Integer);
                2:( AL, AH, BL, BH, CL, CH, DL, DH            : Byte);
             end;

   scrn_type = record
                  adres:integer;
                  card:(mono,color,ega,plantronics,hercules);
               end;

   int80   = array[0..80]of integer;
   scrbuff = array[1..25] of int80;
   scrbufflistptr = ^scrbufflist;
   scrbufflist = record
                    ptr : ^scrbuff;
                    next : scrbufflistptr;
                 end;

var
   firstvar:byte;
   up_key,
   down_key,
   home_key,
   end_key,
   pgup_key,
   pgdn_key,
   mark_key,
   pick_key,
   unpick_key,
   left_key,
   right_key,
   prev_key,
   next_key,
   exit_key,
   help_key,
```

```
   56       brk_key,
   57       esc_key,
   58       term_key,
   59       spc,ret,bksp    : xchar;
   60       screentype : scrn_type;
   61       old_sp,
   62       old_sseg,
   63       bytes_moved : integer;
   64       prog_name,
   65       dosname,
   66       the_cmd     : str80;
   67       prm         : array [0..8] of integer;
   68       regs        : regpack;
   69       scr_last,
   70       scr_base : scrbufflistptr;
   71       lastvar : byte;
   72
   73    const
   74       version_no    : integer = 9;
   75       black         : byte    = 0;
   76       green         : byte    = 2;
   77       cyan          : byte    = 3;
   78       red           : byte    = 4;
   79       yellow        : byte    = 14;
   80       white         : byte    = 15;
   81
   82    const
   83       max_cmds = 30;
   84       sample_buffer_length = 1400;
   85       low_level_commands   = 'LOWLEVEL.188';
   86       high_level_commands. = 'HIGHLEVEL.188';
   87       low_level_helpfile   = 'LOWLEVEL.HLP';
   88       main_help_file       = 'COM80188.HLP';
   89       max_lines = 525;
   90       tv_sync   = 40; (*101000B*)
   91       tv_black  = 48; (*110000B*)
   92       tv_blank  = 47; (*101111B*)
   93       tv_burst  = 15; (*001111B*)
   94
   95       top = 4;
   96       botcenter = 23;
   97       left = 3;
   98
   99    type
  100       status  = (valid,error,abort,unknown,empty,readbuf,writebuf,
  101                  mixdef_show,edit_mixdef_block,put_mixdef,enter_number,
  102                  show_errorlog,change_port,show_dsu_tvline,edit_audio,
  103                  show_audio_data);
  104
  105       machine_type = (encoder,decoder);
  106
  107       command_type = record
  108                         word : string[15];
  109                         rec,
  110                         send : boolean;
```

```
111                          stat : status;
112                       end;
113
114   cmd_list_ptr_type   = ^cmd_list_type;
115
116   actual_cmd_ptr_type = ^actual_cmd_type;
117
118   actual_cmd_type     = record
119                            the_cmd : str40;
120                            next    : actual_cmd_ptr_type;
121                         end;
122
123   cmd_list_type       = record
124                            text        : str40;
125                            xpos,
126                            ypos,
127                            fore,
128                            back        : integer;
129                            redraw      : boolean;
130                            helpfile    : str40;
131                            actual_cmd  : actual_cmd_ptr_type;
132                            prev,
133                            next        : cmd_list_ptr_type;
134                         end;
135
136   (*
137    * some types used in encoder and decoder
138    *)
139
140      block_type = record
141                      start,
142                      length : integer;
143                   end;
144
145      ntsc_line_type = array [0..6] of record
146                                          level : byte;
147                                          count : integer;
148                                       end;
149
150      mixdef_type = record
151                       block      : array[0..5] of block_type;
152                       audio,
153                       marker     : block_type;
154                       marker_def: ntsc_line_type;
155                       sample_tot: array[0..1] of integer;
156                    end;
157   (*
158    *
159    *)
160
161   var
162      end_of_sesion : boolean;
163      hour,min,sec,frac,
164      day,month,year,
165      toprow,
```

```
166       botrow,
167       back,
168       boxcolor : integer;
169       blanks   : str80;
170       all_cmd  : array [1..max_cmds] of command_type;
171       sample_buffer : array[0..sample_buffer_length] of byte;
172       buff_index    : integer;
173       mixdef        : mixdef_type;
174       command_level : (low,high,exit);
175       cmd_list_ptr  : cmd_list_ptr_type;
176       expanded_buffer : boolean;
177
178       port,
179       total_rs232_ports : integer;
180
181       last_line_number : str12;
182
183       errorlog : array [1..50] of integer;
184
185       procedure savewindow(var alot:int80;index,start,length:integer);extern
186       procedure writescr(var alot:int80;index,start,length:integer);external
187
188    procedure screen(lin:str255;row,col,fore,back:integer);
189
190    procedure scrsub(var lin:str255;row,col,colr:integer);external 'scrsub.bi
191
192    begin
193    scrsub(lin,row,col,(fore and 16) shl 3+(back shl 4)+(fore and 15));
194    end;
195    procedure clrcenter(from,till:integer);
196    var index,
197        oldrow,
198        oldcol:integer;
199        s:string[80];
200    begin
201     s:='';
202     for index:=1 to 78 do s:=s+' ';
203     for index:=from to till do
204       begin
205       screen(s,index,2,boxcolor,back);
206       end;
207    end;
208    procedure box(topline,topcol,botline,botcol:integer);
209    var x:integer;
210    line,box1,box2:string[80];
211    begin
212       line:=chr(205);
213       for x:=topcol to (botcol-3) do line:=concat(line,chr(205));
214       box1:=concat(chr(213),line,chr(184));
215       box2:=concat(chr(212),line,chr(190));
216       screen(box1,topline,topcol,boxcolor,back);
217       screen(box2,botline,topcol,boxcolor,back);
218       for x:=topline+1 to (botline-1) do
```

```
219       begin
220          screen(chr(179),x,topcol,boxcolor,back);
221          screen(chr(179),x,botcol,boxcolor,back);
222       end;
223       line := chr(196);
224       for x:=topcol to botcol-3 do
225          line := line + chr(196);
226       line := concat(chr(195),line,chr(180));
227       screen(line,topline+2,topcol,boxcolor,back);
228    end;
229       procedure eattb(var s:str255);
230       begin
231          while s[length(s)] = ' ' do
232             s[0]:=chr(ord(s[0])-1);
233       end;
234       procedure eatlb(var s:str255);
235       var
236          leng:integer;
237          done:boolean;
238       begin
239          done:=false;
240          leng:=length(s);
241          while (leng > 0) and (not done) do
242          begin
243             if s[1]=' '
244             then
245             begin
246                delete(s,1,1);
247                leng:=length(s);
248             end
249             else
250                done:=true;
251          end;
252       end;
253       function strcmp(a,b:str80):boolean;
254
255       var
256          index:integer;
257          ready:boolean;
258       begin
259          index:=0;
260          ready:=false;
261          while (upcase(a[index])=upcase(b[index])) and (not ready) do
262          begin
263             ready:=index=length(a);
264             index:=index+1;
265          end;
266          strcmp:=ready;
267       end;
268      function spekeys(xch:xchar):boolean;
269      begin
270         spekeys := (xch=ret) or (xch=help_key) or (xch=exit_key) or
271                    (xch=prev_key) or (xch=next_key) or (xch=term_key);
272      end;
273       function getxchar:xchar;
```

```
274       var
275          xch:xchar;
276          ch:char;
277       begin
278          xch := '';
279          read(kbd,ch);
280          if (keypressed) and (ch=chr(esc_scan))
281          then
282          begin
283             xch := ch;
284             read(kbd,ch);
285          end;
286          getxchar:=xch+ch;
287       end;
288    procedure time(var h,m,s,f:integer);
289    var
290       regs : regpack;
291    begin
292       with regs do
293       begin
294          ax := $2c00;
295          msdos(regs);
296          h := hi(cx);
297          m := lo(cx);
298          s := hi(dx);
299          f := lo(dx);
300       end;
301    end;
302    function escapepressed:boolean;
303    var
304      xch : xchar;
305    begin
306       if keypressed
307       then
308       begin
309          xch := getxchar;
310          escapepressed := xch = esc_key;
311       end
312       else
313          escapepressed := false;
314    end;
315      Procedure hitreturn;
316      var
317         xch : xchar;
318      begin
319        gotoxy(29,botrow);
320        screen('Hit space bar to continue...'+copy(blanks,1,50),botrow,1,boxc
321        xch := getxchar;
322    {
323        if xch = esc_key
324        then
325          end_of_sesion := true;
326    }
327        screen(blanks,botrow,1,boxcolor,back);
```

```
328     end;
329   procedure beep;
330   begin
331      write(chr(7));
332   end;
333   procedure readstr(x,y:integer;var astring:str80;len:byte);
334
335     var
336        the_set : set of char;
337        xch : xchar;
338
339     begin
340        textcolor(white);
341        textbackground(black);
342        the_set := [chr(0)..chr(255)];
343        astring := '';
344        xch:='';
345        gotoxy(x,y);
346        screen(copy(blanks,1,len),wherey,wherex,white,black);
347        repeat until keypressed;
348        while not spekeys(xch)  do
349        begin
350           xch := getxchar;
351           if not spekeys(xch)
352           then
353           begin
354              if xch = bksp
355              then
356              begin
357                 if length(astring) > 0
358                 then
359                 begin
360                    astring[0] := pred(astring[0]);
361                    gotoxy(wherex-1,wherey);
362                    screen(' ',wherey,wherex,white,black);
363                 end
364                 else
365                    beep;
366              end
367              else
368              begin
369                 if (length(astring)<len) and ((xch[1] in the_set) and (leng
370                 then
371                 begin
372                    astring := astring + xch;
373                    screen(xch,wherey,wherex,white,black);
374                    gotoxy(wherex+1,wherey);
375                 end
376                 else
377                    beep;
378              end;
379           end
380           else
381           begin
```

```
382                if xch <> ret
383                then
384                   astring := xch;
385             end;
386          end;
387          screen('    ',y,x+len+1,white,black);
388       end;
389    procedure colormonitor(var st:scrn_type);
390
391    (* returns true if colormonitor attached. Otherwise false *)
392    var store:integer;
393
394    begin
395       store:=memw[$0000:1040];
396       if ((store and 48) div 16)=2
397       then
398       begin
399          st.card:=color;
400          st.adres:=$b800;
401       end
402       else
403       begin
404          st.card:=mono;
405          st.adres:=$b000;
406       end;
407    end;
408    procedure getcomspec(var result:str80);
409
410    var environment,index,index2,max_env_size,i:integer;
411        hulp:byte;
412        found:boolean;
413        tempstr:string[255];
414        lastbyte:integer;
415
416       begin
417       lastbyte:=1;
418       hulp:=1;
419       result:='\COMMAND.COM'+chr(0);  (* ASCIIZ string *)
420       environment:=memw[cseg:$2c];    { get environment segment from offset
421       found:=false;
422       index:=-1;
423       max_env_size:=0;
424       while (not found) and (max_env_size<32767) do
425       begin
426          index2:=0;
427          repeat
428             index:=index+1;
429             index2:=index2+1;
430             tempstr[0]:=chr(index2);
431             lastbyte:=hulp;
432             hulp:=mem[environment:index];
433             tempstr[index2]:=chr(hulp);
434             max_env_size:=max_env_size+1;
435          until (index2=255) or (hulp=0);     { get a string from the environm
```

```
436
437         found:=pos('COMSPEC=',tempstr)>0;
438       end; (* while *)
439
440       tempstr:=copy(tempstr,pos('=',tempstr)+1,length(tempstr)-7);  { remove C
441       if found then
442        result:=tempstr;
443
444
445        hulp:=1;
446        lastbyte:=1;
447        index:=-1;
448        while  not((lastbyte=0) and (hulp=0)) do
449        begin
450           index:=index+1;
451           lastbyte:=hulp;
452           hulp:=mem[environment:index];
453        end;
454        index:=index+2;
455        hulp:=mem[environment:index];
456        tempstr:='';
457        i:=1;
458        while (mem[environment:index+i]<>0) do
459        begin
460          tempstr:=tempstr+chr(mem[environment:index+i]);
461          i:=i+1;
462        end;
463        if pos(':',tempstr)>0
464        then
465           delete(tempstr,1,pos(':',tempstr));
466        while pos('\',tempstr)>0 do
467           delete(tempstr,1,pos('\',tempstr));
468        prog_name:=copy(tempstr,1,12);
469       end; (* getcomspec *)
470
471
472       procedure init_dataseg;
473       begin
474          fillchar(memw[dseg:ofs(firstvar)],ofs(lastvar)-ofs(firstvar)+1,0);
475          fillchar(blanks[1],80,' ');
476          blanks[0] := chr(80);
477          ret       := chr(13);
478          bksp      := char(8);
479          spc       := chr(32);
480          prev_key  := chr(27) + chr(59);  (* F1 *)
481          next_key  := chr(27) + chr(60);  (* F2 *)
482          help_key  := chr(27) + chr(61);  (* F3 *)
483          brk_key   := chr(27) + chr(62);  (* F4 *)
484          term_key  := chr(27) + chr(67);  (* F9 *)
485          exit_key  := chr(27) + chr(68);  (* F10*)
486          left_key  := chr(27) + chr(75);
487          right_key := chr(27) + chr(77);
488          pick_key  := chr(27) + chr(65);
489          unpick_key:= chr(27) + chr(66);
```

```
490         mark_key  := chr(27) + chr(67);
491         home_key  := chr(27) + chr(71);
492         end_key   := chr(27) + chr(79);
493         pgup_key  := chr(27) + chr(73);
494         pgdn_key  := chr(27) + chr(81);
495         up_key    := chr(27) + chr(72);
496         down_key  := chr(27) + chr(80);
497         esc_key   := chr(27);
498         getcomspec(dosname);
499         dosname:=dosname+chr(0);
500         colormonitor(screentype);
501         scr_base := nil;
502         scr_last := nil;
503      end;
504       procedure savescreen(save:boolean);
505       var
506          i:integer;
507          temp : scrbufflistptr;
508       begin
509          if save
510          then
511          begin
512             new(temp);
513             new(temp^.ptr);
514             temp^.next := nil;
515             if scr_base = nil
516             then
517             begin
518                 scr_base := temp;
519             end;
520             if scr_last <> nil
521             then
522             begin
523                scr_last^.next := temp;
524             end;
525             scr_last := temp;
526
527             for i:=1 to 25 do
528                savewindow(temp^.ptr^[i],i,1,80);
529
530          end
531          else
532          begin
533             temp := scr_base;
534             scr_last := scr_base;
535             while temp^.next <> nil do
536             begin
537                write('.');
538                scr_last := temp;   (* point to one before last*)
539                temp := temp^.next;
540             end;
541
542             for i:=1 to 25 do
543                writescr(temp^.ptr^[i],i,1,80);
544
```

```
545             scr_last^.next:=nil;
546             if temp=scr_base
547             then
548             begin
549                scr_base:=nil;
550                scr_last:=nil;
551             end;
552             dispose(temp^.ptr);
553             dispose(temp);
554    {
555             if (scr_last = scr_base)
556             then
557             begin
558                if scr_base^.next = nil
559                then
560                begin
561                   scr_base := nil;
562                   scr_last := nil;
563                end
564                else
565                   scr_base^.next:= nil;
566             end;
567
568             if scr_last<> nil
569             then
570                scr_last^.next := nil;
571    }
572          end;
573       end;
574
575    const
576      baud_9600          = $E0;
577      baud_2400          = $A0;
578      even_parity        = $18;
579      no_parity          = $00;
580      one_stop           = $00;
581      data_7_bit         = $02;
582      data_8_bit         = $03;
583      time_out           = $8000;
584      framing_err        = $0800;
585      parity_err         = $0400;
586      overrun_err        = $0200;
587      trans_shift_empty  = $4000;
588      trans_hold_empty   = $2000;
589      data_ready         = $0100;
590
591
592    procedure bioscom(cmd:byte;var ch :byte;port:integer;var status:integer);
593    var
594       regs :regpack;
595    begin
596       if (port in [0,1]) and (cmd in [0..3])
597       then
598       begin
599          regs.ah := cmd;
```

```
600          regs.dx := port;
601          if regs.ah in [0,1]
602          then
603             regs.al := ch;
604          intr($14,regs);
605          case cmd of
606             0,3 : status := regs.ax;
607             1,2 : begin
608                      if cmd = 2
609                      then
610                      begin
611                         ch := regs.al;
612                      end;
613                      status := regs.ah;
614                   end;
615          end; (*case*)
616       end;
617    end;
618
619    function rs232_stat(port:byte):integer;
620    var
621       stat : integer;
622       init : byte;
623    begin
624       if port in [0,1]
625       then
626       begin
627          bioscom(3,init,port,stat);
628          rs232_stat := stat;
629       end;
630    end;
631
632    function rs232_init(port:byte):boolean;
633    var
634       stat : integer;
635       init : byte;
636    begin
637       if port in [0,1]
638       then
639       begin
640          init := (baud_2400 or even_parity or one_stop or data_7_bit);
641          bioscom(0,init,port,stat);
642          rs232_init :=  (stat and (time_out or framing_err or
643                                    parity_err or overrun_err)) = 0;
644          while ((rs232_stat(port) and data_ready) <> 0 ) and (not escapepres
645          begin
646             bioscom(2,init,port,stat);
647          end;
648       end
649       else
650          rs232_init := false;
651    end;
652
653
654    procedure rs232_send(ch,port:byte);
```

```
655   var
656      i,
657      stat : integer;
658   begin
659      repeat
660         stat := rs232_stat(port);
661      until ((stat and (trans_shift_empty or trans_hold_empty)) <> 0)
662            or (escapepressed);
663      bioscom(1,ch,port,stat);
664   end;
665
666   function rs232_rec(var ch:byte;port:byte):boolean;
667   var
668      stat : integer;
669   begin
670      repeat
671         stat := rs232_stat(port);
672      until ((stat and data_ready) <> 0)
673            or (escapepressed);
674      bioscom(2,ch,port,stat);
675      rs232_rec := stat = 0;
676   end;
677
678
679   function rs232_message(end_val,port:byte):boolean;
680   const
681      F = $46;
682   var
683      ok : boolean;
684   begin
685      buff_index := -1;
686      repeat
687         buff_index := buff_index + 1;
688         ok := rs232_rec(sample_buffer[buff_index],port);
689      until (sample_buffer[buff_index] > F ) or
690            (buff_index>sample_buffer_length) or (not ok);
691      rs232_message := sample_buffer[buff_index] = end_val;
692   end;
693
694   function rs232ports:integer;
695   var
696     regs : regpack;
697   begin
698      intr($11,regs);
699      rs232ports := (regs.ax shr 9) and $0007;
700   end;
701
702      procedure do_error(i:integer);
703      var
704         txt : str80;
705      begin
706         case i of
707            1 : txt := 'Unknown low level command';
708            2 : txt := 'Encoder/Decoder status not returned';
709            3 : txt := 'Character not sent';
```

```
710             4 : txt := 'Character not echoed after buffer or buffer not sent
711             5 : txt := 'Character not echoed';
712             6 : txt := 'Invalid integer';
713             7 : txt := 'Key not implemented';
714             8 : txt := 'No RS232 ports attached';
715             9 : txt := 'Invalid RS232 port number, defaulted to com1';
716            10 : txt := 'Error initializing RS232 port, try again';
717            11 : txt := 'Too many samples in this line';
718            12 : txt := 'Error opening file '+high_level_commands;
719            13 : txt := 'Error reading high level command';
720            14 : txt := 'Encoder/Decoder not detected';
721            15 : txt := 'Help function not yet implemented';
722            16 : txt := 'No high level commands available';
723            17 : txt := 'Error opening file '+low_level_commands;
724            18 : txt := 'No low level commands available';
725             else
726             begin
727                str(i:8,txt);
728                txt := 'Undefined error'+ txt;
729
730             end;
731          end; (*case*)
732          screen('ERROR-- '+txt,botrow-1,1,black,(white mod 8));
733          hitreturn;
734          box(toprow,1,24,80);
735       end;
736
737
738       procedure show_port;
739       begin
740          screen('I/O port = COM'+chr(port+49),2,65,boxcolor,back);
741       end;
742
743       procedure setupscreen;
744       begin
745          botrow := 25;
746          toprow := 1;
747          back := 0;
748          boxcolor := 2;
749          clrscr;
750          box(toprow,1,24,80);
751          screen('Digital Speed-Up Utility',2,2,boxcolor,back);
752          show_port;
753       end;
754
755       function get_token(var f:text;var s:str80):boolean;
756       var
757          temp : boolean;
758       begin
759          repeat
760             s := '';
761             readln(f,s);
762             temp := ioresult = 0;
763             eatlb(s);
764             eattb(s);
```

```
765         until ((s<>'') and (s[1]<>'*')) or (not temp);
766         get_token:= temp;
767      end;
768
769
770      procedure set_up_cmd_list;
771      const
772         row_one = 3;
773         row_two = 43;
774      var
775         f     : text;
776         first : boolean;
777         xxx,
778         tcp: cmd_list_ptr_type;   (* temp command pointer*)
779
780
781
782         function new_field_ok(var p:cmd_list_ptr_type):boolean;
783         var
784            ok : boolean;
785
786            procedure get_text(var p:cmd_list_ptr_type;var ok:boolean);
787            var
788               s : str80;
789            begin
790               if not get_token(f,s)
791               then
792               begin
793                  ok := false;
794                  p^.text := '';
795               end
796               else
797                  p^.text := s;
798            end;
799
800            procedure get_xpos(var p:cmd_list_ptr_type;var ok:boolean);
801            const
802               xmin = 0;
803               xmax = 80;
804            var
805               s : str80;
806               i,err : integer;
807            begin
808               if not get_token(f,s)
809               then
810               begin
811                  ok := false;
812                  p^.xpos := 1;
813               end
814               else
815               begin
816                  val(s,i,err);
817                  if (err = 0) and (i>=xmin) and (i<=xmax)
818                  then
819                     p^.xpos := i
```

```
820                  else
821                  begin
822                     ok := false;
823                     p^.xpos := 1;
824                  end;
825               end;
826            end;
827
828            procedure get_ypos(var p:cmd_list_ptr_type;var ok:boolean);
829            const
830               ymin = 0;
831               ymax = 24;
832            var
833               s : str80;
834               i,err : integer;
835            begin
836               if not get_token(f,s)
837               then
838               begin
839                  ok := false;
840                  p^.ypos := 1;
841               end
842               else
843               begin
844                  val(s,i,err);
845                  if (err = 0) and (i>=ymin) and (i<=ymax)
846                  then
847                     p^.ypos := i
848                  else
849                  begin
850                     ok := false;
851                     p^.ypos := 1;
852                  end;
853               end;
854            end;
855
856            procedure get_fore(var p:cmd_list_ptr_type;var ok:boolean);
857            const
858               colmin = 0;
859               colmax = 24;
860            var
861               s : str80;
862               i,err : integer;
863            begin
864               if not get_token(f,s)
865               then
866               begin
867                  ok := false;
868                  p^.fore := white;
869               end
870               else
871               begin
872                  val(s,i,err);
873                  if (err = 0) and (i>=colmin) and (i<=colmax)
874                  then
```

```
875                     p^.fore := i
876                  else
877                  begin
878                     ok := false;
879                     p^.fore := white;
880                  end;
881               end;
882            end;
883
884            procedure get_back(var p:cmd_list_ptr_type;var ok:boolean);
885            const
886               colmin = 0;
887               colmax = 24;
888            var
889               s : str80;
890               i,err : integer;
891            begin
892               if not get_token(f,s)
893               then
894               begin
895                  ok := false;
896                  p^.back := black;
897               end
898               else
899               begin
900                  val(s,i,err);
901                  if (err = 0) and (i>=colmin) and (i<=colmax)
902                  then
903                     p^.back := i
904                  else
905                  begin
906                     ok := false;
907                     p^.back := black;
908                  end;
909               end;
910            end;
911
912            procedure get_redraw(var p:cmd_list_ptr_type;var ok:boolean);
913            var
914               s : str80;
915            begin
916               if not get_token(f,s)
917               then
918               begin
919                  ok := false;
920                  p^.redraw := true;
921               end
922               else
923               begin
924                  p^.redraw := not strcmp(s,'FALSE');
925               end;
926            end;
927
928            procedure get_helpfile(var p:cmd_list_ptr_type;var ok:boolean);
929            var
```

```
930             s : str80;
931          begin
932             if not get_token(f,s)
933             then
934             begin
935                ok := false;
936                p^.helpfile := main_help_file;
937             end
938             else
939             begin
940                p^.helpfile := s;
941             end;
942          end;
943
944          procedure get_low_commands(var p:cmd_list_ptr_type;var ok:boolean
945          var
946             s : str80;
947             i : integer;
948             first : boolean;
949             yyy,
950             tap : actual_cmd_ptr_type; (* temp actual pointer *)
951
952             function low_level_ok(var y:actual_cmd_ptr_type;s:str80):bool
953             begin
954                if strcmp(s,'LOWEND')
955                then
956                   low_level_ok := false
957                else
958                begin
959                   low_level_ok := true;
960                   y^.the_cmd := s;
961                end;
962             end;
963
964          begin
965             ok := ok and get_token(f,s);
966             if strcmp(s,'LOWSTART')
967             then
968             begin
969                first := true;
970                repeat
971                   new(yyy);
972                   ok := (ok and get_token(f,s));
973                   if low_level_ok(yyy,s)
974                   then
975                   begin
976                      if first
977                      then
978                      begin
979                         first := false;
980                         tap := yyy;
981                         tap^.next := nil;
982                         p^.actual_cmd := tap;
983                      end
984                      else
```

```
 985                         begin
 986                            tap^.next := yyy;
 987                            yyy^.next := nil;
 988                            tap := yyy;
 989                         end;
 990                      end
 991                      else
 992                      begin
 993                         dispose(yyy);
 994                      end;
 995                   until (strcmp(s,'LOWEND') or eof(f));
 996                   ok := ok and strcmp(s,'LOWEND');
 997                end
 998                else
 999                   ok := false;
1000                if not ok
1001                then
1002                   p^.actual_cmd := nil;
1003             end;
1004
1005          begin
1006             ok := true;
1007             get_text(p,ok);
1008             get_xpos(p,ok);
1009             get_ypos(p,ok);
1010             get_fore(p,ok);
1011             get_back(p,ok);
1012             get_redraw(p,ok);
1013             get_helpfile(p,ok);
1014             get_low_commands(p,ok);
1015             new_field_ok := ok;
1016          end;
1017
1018       begin
1019          cmd_list_ptr := nil;
1020          assign(f,high_level_commands);
1021          reset(f);
1022          if ioresult = 0
1023          then
1024          begin
1025             tcp := nil;
1026             first := true;
1027             while not eof(f) do
1028             begin
1029                new(xxx);
1030                if new_field_ok(xxx)
1031                then
1032                begin
1033                   if first
1034                   then
1035                   begin
1036                      first := false;
1037                      tcp := xxx;
1038                      cmd_list_ptr := tcp;
1039                      tcp^.prev := nil;
```

```
1040                    tcp^.next := nil;
1041                 end
1042                 else
1043                 begin
1044                    tcp^.next := xxx;
1045                    xxx^.next := nil;
1046                    xxx^.prev := tcp;
1047                    tcp := xxx;
1048                 end;
1049              end
1050              else
1051              begin
1052                 dispose(xxx);
1053                 do_error(13);
1054              end;
1055           end;
1056           (* now make circular*)
1057           cmd_list_ptr^.prev := tcp;
1058        end
1059        else
1060           do_error(12);
1061     end;
1062
1063     procedure init_cmds;
1064     var
1065        err,
1066        i,max : integer;
1067        f     : text;
1068        temp  : str80;
1069     begin
1070        assign(f,low_level_commands);
1071        reset(f);
1072        err := ioresult;
1073        if err <> 0
1074        then
1075           do_error(17);
1076        max := 0;
1077        fillchar(all_cmd[1],sizeof(all_cmd),0);
1078        if get_token(f,temp)
1079        then
1080           val(temp,max,err);
1081        for i := 1 to max do
1082        begin
1083           if max <= max_cmds
1084           then
1085           begin
1086              if not get_token(f,temp)
1087              then
1088                 err := err + 1;
1089              all_cmd[i].word := temp;
1090              if not get_token(f,temp)
1091              then
1092                 err := err + 1;
1093              all_cmd[i].rec  := strcmp(temp,'TRUE');
1094              if not get_token(f,temp)
```

```
1095                then
1096                   err := err + 1;
1097                all_cmd[i].send := strcmp(temp,'TRUE');
1098                if not get_token(f,temp)
1099                then
1100                   err := err + 1;
1101                if strcmp(temp,'VALID')
1102                then
1103                   all_cmd[i].stat := valid
1104                else if strcmp(temp,'READBUF')
1105                then
1106                   all_cmd[i].stat := readbuf
1107                else if strcmp(temp,'WRITEBUF')
1108                then
1109                   all_cmd[i].stat := writebuf
1110                else if strcmp(temp,'MIXDEF')
1111                then
1112                   all_cmd[i].stat := mixdef_show
1113                else if strcmp(temp,'EDITMIXDEFBLOCK')
1114                then
1115                   all_cmd[i].stat := edit_mixdef_block
1116                else if strcmp(temp,'PUTMIXDEF')
1117                then
1118                   all_cmd[i].stat := put_mixdef
1119                else if strcmp(temp,'SHOWERRORLOG')
1120                then
1121                   all_cmd[i].stat := show_errorlog
1122                else if strcmp(temp,'CHANGEPORT')
1123                then
1124                   all_cmd[i].stat := change_port
1125                else if strcmp(temp,'ENTERNUMBER')
1126                then
1127                   all_cmd[i].stat := enter_number
1128                else if strcmp(temp,'SHOWTVLINE')
1129                then
1130                   all_cmd[i].stat := show_dsu_tvline
1131                else if strcmp(temp,'EDITAUDIO&MARKER')
1132                then
1133                   all_cmd[i].stat := edit_audio
1134                else if strcmp(temp,'SHOWAUDIODATA')
1135                then
1136                   all_cmd[i].stat := show_audio_data
1137                else
1138                   all_cmd[i].stat  := abort;
1139             end;
1140          end;
1141          close(f);
1142          err := err + ioresult;
1143          if err <> 0
1144          then
1145             do_error(err);
1146       end;
1147
1148
1149       Procedure Show_logo;
```

```
1150     const
1151        msg1 = 'Digital Speed-Up Utility';
1152        msg2 = 'Version 1.2';
1153     var
1154        center : integer;
1155
1156     begin
1157        center := 37;
1158        screen(msg1,10,center-(length(msg1) div 2),white,black);
1159        screen(msg2,11,center-(length(msg2) div 2),white,black);
1160        gotoxy(center,12);
1161     end;
1162
1163     Procedure Initialize;
1164     var
1165        end_time,
1166        h,m,s,f : integer;
1167        xc      : xchar;
1168     begin
1169       init_dataseg;
1170       port := 0;
1171       total_rs232_ports := rs232ports;
1172       setupscreen;
1173       show_logo;
1174       time(h,m,s,f);
1175       end_time := m * 60 + s + 6;    (* wait for 6 seconds *)
1176       init_cmds;
1177       buff_index := 0;
1178       cmd_list_ptr := nil;
1179       end_of_sesion := false;
1180       if (paramstr(1) = '/L') or (paramstr(1) = '/l')
1181       then
1182          command_level := low
1183       else
1184       begin
1185          command_level := high;
1186          set_up_cmd_list;
1187       end;
1188       repeat
1189          time(h,m,s,f);
1190       until ((m*60 + s) > end_time) or (keypressed);
1191       if keypressed
1192       then
1193          xc := getxchar;
1194       clrcenter(top,botcenter);
1195     end {initialize};
1196
1197      procedure change_rs232port;
1198      begin
1199         case port of
1200            0 : port := 1;
1201            1 : port := 0;
1202            else
1203            begin
1204               port := 0;
```

```
1205                do_error(9);
1206             end;
1207          end; (*case*)
1208          show_port;
1209       end;
1210
1211
1212
1213       procedure do_help(filename:str80);
1214       begin
1215          savescreen(true);
1216          clrscr;
1217          do_error(15);
1218          savescreen(false);
1219       end;
1220
1221
1222       procedure dump_help(filename:str80);
1223       const
1224          top_help_line = 4;
1225          bot_help_line = 20;
1226       var
1227          helpfile : text;
1228          dummy    : str255;
1229          j,
1230          helpcolor: integer;
1231
1232       begin
1233          helpcolor := boxcolor;
1234          assign(helpfile,filename);
1235          reset(helpfile);
1236          j := top_help_line;
1237          while (j<=bot_help_line) and (not eof(helpfile)) and (ioresult=0) do
1238          begin
1239             readln(helpfile,dummy);
1240             screen(copy(dummy,1,76),j,3,helpcolor,back);
1241             j := j + 1
1242          end;
1243          close(helpfile);
1244          j := ioresult;
1245       end;
1246
1247       procedure get_command(var cmd:str80);
1248       begin
1249          repeat
1250             screen('Enter command>',23,2,boxcolor,back);
1251             readstr(17,23,cmd,15);
1252             if (cmd = help_key) or (cmd = prev_key)
1253             then
1254                do_help(main_help_file);
1255          until (cmd <> help_key) and (cmd <> prev_key); (* f1 and f3 are both
1256       end;
1257
1258       function parse_command(var ret_cmd:command_type;cmd:str80):status;
1259       var
```

```
1260        ind : integer;
1261     begin
1262        eatlb(cmd);
1263        eattb(cmd);
1264        if cmd = ''
1265        then
1266        begin
1267           parse_command := empty;
1268        end
1269        else
1270        begin
1271           ind := 1;
1272           while (ind<max_cmds) and (not strcmp(all_cmd[ind].word,cmd)) do
1273           begin
1274              ind := ind + 1;
1275           end;
1276           if strcmp(all_cmd[ind].word,cmd)
1277           then
1278           begin
1279              parse_command := all_cmd[ind].stat;
1280              ret_cmd := all_cmd[ind];
1281           end;
1282        end;
1283     end;
1284
1285      procedure compress_sample_buffer;
1286      var
1287         low,high : integer;
1288
1289         function convert(h,l:byte):integer;
1290         begin
1291            if chr(h) in ['0'..'9']
1292            then
1293               h := h - $30
1294            else
1295               h := h - $37;
1296            if chr(l) in ['0'..'9']
1297            then
1298               l := l - $30
1299            else
1300               l := l - $37;
1301            convert := (h shl 4) or l;
1302         end;
1303      begin
1304         if expanded_buffer
1305         then
1306         begin
1307            high := 0;
1308            low  := 0;
1309            while high < buff_index do
1310            begin
1311               sample_buffer[low] := convert(sample_buffer[high],sample_buff
1312               low := low + 1;
1313               high := high + 2;
```

```
1314            end;
1315            buff_index := low;
1316            expanded_buffer := false;
1317         end;
1318      end;
1319
1320      procedure expand_sample_buffer;
1321      var
1322        i : integer;
1323
1324         function low_nible(b:byte):byte;
1325         begin
1326            b := b and $0F;
1327            if b < $A
1328            then
1329               low_nible := b+$30
1330            else
1331               low_nible := b+$37
1332         end;
1333
1334         function high_nible(b:byte):byte;
1335         begin
1336            b := (b and $F0) shr 4;
1337            if b < $A
1338            then
1339               high_nible := b+$30
1340            else
1341               high_nible := b+$37
1342         end;
1343
1344      begin
1345         if not expanded_buffer
1346         then
1347         begin
1348            for i := buff_index downto 0 do
1349            begin
1350               sample_buffer[2*i+1] := low_nible(sample_buffer[i]);
1351               sample_buffer[(2*i)] := high_nible(sample_buffer[i]);
1352            end;
1353            buff_index := 2 * buff_index;
1354            expanded_buffer := true;
1355         end;
1356      end;
1357
1358     procedure send_command(cmd_el:command_type);
1359     const
1360        all_rs232_errs = $8E;
1361        esc_str = 'Press escape to interrupt communication';
1362     var
1363        temp : byte;
1364        i    : integer;
1365        ok   : boolean;
1366        oldx,
1367        oldy :byte;
1368     begin
```

```
1369          oldx := wherex;
1370          oldy := wherey;
1371          show_port;
1372          screen(esc_str,botrow,1,white,black);
1373          gotoxy(length(esc_str)+1,botrow);
1374          if rs232_init(port) then;
1375          if ((rs232_stat(port) and (all_rs232_errs)) = 0)
1376          then
1377          begin
1378             if cmd_el.send
1379             then
1380             begin
1381                expand_sample_buffer;
1382                for i := 0 to buff_index-1 do
1383                begin
1384                   rs232_send(sample_buffer[i],port);
1385                   if not rs232_rec(temp,port)
1386                   then
1387                   begin
1388                      do_error(3);
1389                      exit;
1390                   end;
1391                end;
1392             end;
1393             rs232_send(ord(cmd_el.word[1]),port);
1394             if cmd_el.rec
1395             then
1396             begin
1397                ok := rs232_message(ord(cmd_el.word[1]),port);
1398                expanded_buffer := true;
1399                if not ok
1400                then
1401                   do_error(4)
1402                else
1403                begin
1404                   compress_sample_buffer;
1405                end;
1406             end
1407             else
1408             begin
1409                if (not rs232_rec(temp,port)) or (temp<>ord(cmd_el.word[1]))
1410                then
1411                   do_error(5);
1412             end;
1413          end
1414          else
1415             do_error(rs232_stat(port));
1416          screen(blanks,botrow,1,white,black);
1417          gotoxy(oldx,oldy);
1418       end;
1419
1420
1421        procedure write_sample_buffer;
1422        const
1423           top = 4;
```

```
1424        var
1425           s : str80;
1426           i,j : integer;
1427        begin
1428           expanded_buffer := true;
1429           i := 1;
1430           buff_index := 0;
1431           clrcenter(top,botcenter);
1432           screen('Enter text lines for buffer and end with blank line',top,3,w
1433           repeat
1434               screen('Enter>',top+i,3,white,black);
1435               readstr(9,top+i,s,65);
1436               if s<>''
1437               then
1438                  for j := 1 to length(s) do
1439                  begin
1440                     sample_buffer[buff_index] := ord(s[j]);
1441                     buff_index := (buff_index + 1) mod sample_buffer_length;
1442                  end;
1443               i := i + 1;
1444               if i = 18
1445               then
1446                  i := 1;
1447           until s = '';
1448           clrcenter(top,botcenter);
1449        end;
1450
1451        procedure read_sample_buffer;
1452        const
1453           top = 4;
1454           left = 3;
1455           right = 79;
1456           width = 76;
1457        var
1458           buflenstr,
1459           s : str80;
1460           i,j : integer;
1461        begin
1462           expand_sample_buffer;
1463           i := 1;
1464           clrcenter(top,botcenter);
1465           screen('The sample buffer:',top,left,white,black);
1466           str(buff_index,buflenstr);
1467           screen('length:'+buflenstr,top,right-20,white,black);
1468           for i := 0 to buff_index-1 do
1469           begin
1470              j := i mod width;
1471              screen(chr(sample_buffer[i]),
1472                     (top+1+(i div width)),
1473                     (left+j),
1474                     white,black);
1475           end;
1476           compress_sample_buffer;
1477           hitreturn;
```

```
1478          clrcenter(top,botcenter);
1479       end;
1480
1481
1482       procedure show_mixdef;
1483       var
1484          a,b,
1485          lev,cnt      : str80;
1486          i, mix_ind   : integer;
1487          tempreal     : real;
1488       begin
1489          mix_ind := 1;
1490          move(sample_buffer[0],mixdef,sizeof(mixdef));
1491          clrcenter(top,botcenter);
1492          screen('Mixdef:',top,left,white,black);
1493          for i := 0 to 5 do
1494          begin
1495             str(mixdef.block[i].start:8,lev);
1496             str(mixdef.block[i].length:8,cnt);
1497             screen('Mixdef.block['+chr(i+48)+'].start, length    : '+lev+cnt,
1498          end;
1499          i := 6;
1500          str(mixdef.audio.start:8,lev);
1501          str(mixdef.audio.length:8,cnt);
1502          screen('Mixdef.audio.start, length       : '+lev+cnt,top + i + 1,le
1503          i := 7;
1504          str(mixdef.marker.start:8,lev);
1505          str(mixdef.marker.length:8,cnt);
1506          screen('Mixdef.marker.start, length      : '+lev+cnt,top + i + 1,le
1507          for i := 8 to 14 do
1508          begin
1509             str(mixdef.marker_def[i-8].level:8,lev);
1510             str(mixdef.marker_def[i-8].count:8,cnt);
1511             screen('Mixdef.marker_def['+chr(i+40)+'].level, count: '+lev+cnt
1512          end;
1513          str(mixdef.sample_tot[0]:8,lev);
1514          str(mixdef.sample_tot[1]:8,cnt);
1515          i := 15;
1516          screen('Mixdef.sample_tot two integers   : '+lev+cnt,top + i + 1,le
1517          hitreturn;
1518          clrcenter(top,botcenter);
1519       end;
1520
1521       function valid_int(s:str80;var int:integer):boolean;
1522       var
1523          err : integer;
1524       begin
1525          val(s,int,err);
1526          valid_int := (err=0) and (s<>'');
1527       end;
```

```
1528
1529        procedure show_tvline(machine:machine_type);
1530        const
1531           tot_samples = 399;
1532           len         = 3;
1533           local_bot   = 23;
1534        var
1535           this_x,
1536           this_y,
1537           i,
1538           count : integer;
1539        type
1540           the_sample_types = (sync_sample,blank_sample,burst_sample,data);
1541
1542           procedure ask_save;
1543           var
1544              msg,
1545              thename : str80;
1546              ans     : xchar;
1547
1548              procedure save_buff_to_file(name : str80);
1549              var
1550                 f   : text;
1551                 ind,
1552                 err : integer;
1553              begin
1554                 assign(f,name);
1555                 rewrite(f);
1556                 err := ioresult;
1557                 ind := 0;
1558                 while (err = 0) and (ind < buff_index) do
1559                 begin
1560                    write(f,sample_buffer[ind]);
1561                    err := err + ioresult;
1562                    ind := ind + 1;
1563                 end;
1564                 close(f);
1565                 err := err + ioresult;
1566                 if err<>0
1567                 then
1568                    do_error(err);
1569              end;
1570
1571           begin
1572              thename := 'LINE' + last_line_number;
1573              if machine = encoder
1574              then
1575                  thename := thename + '.ENC'
1576              else
1577                  thename := thename + '.DEC';
1578              msg := 'Save as file '+thename+' [Y/n]? ';
1579              screen(msg,botrow,1,white,black);
1580              gotoxy(length(msg)+1,botrow);
1581              ans := getxchar;
1582              if (ans = 'Y') or (ans = 'y') or (ans = ret)
```

```
1583            then
1584            begin
1585               screen('Yes',botrow,length(msg)+1,white,black);
1586               save_buff_to_file(thename);
1587            end;
1588            screen(blanks,botrow,1,white,black);
1589         end;
1590
1591         function get_sample_type(sample:byte):the_sample_types;
1592         begin
1593            case sample of
1594               tv_sync  : get_sample_type := sync_sample;
1595               tv_blank : get_sample_type := blank_sample;
1596               tv_burst : get_sample_type := burst_sample;
1597               else
1598                  get_sample_type := data;
1599            end; (*case*)
1600         end;
1601
1602         procedure update_y;
1603         begin
1604            this_y := this_y + 1;
1605            if this_y > local_bot
1606            then
1607            begin
1608               do_error(11);
1609               clrcenter(top,local_bot);
1610               this_y := top;
1611            end;
1612         end;
1613
1614         procedure update_xy;
1615         begin
1616            this_x := left;
1617            update_y;
1618         end;
1619
1620
1621         procedure show_count(s_t:the_sample_types);
1622         var
1623            temp1 : string[len];
1624            temp2 : str12;
1625         begin
1626            str(count:len,temp1);
1627            case s_t of
1628               sync_sample  : temp2 := ' SYNC ';
1629               blank_sample : temp2 := ' BLANK';
1630               burst_sample : temp2 := ' BURST';
1631               else
1632                  temp2 := ' UNKNOWN';
1633            end; (*case*)
1634            if this_x <> left
1635            then
1636               update_xy;
1637            screen(temp1+temp2+' samples',this_y,this_x,white,black);
```

```
1638            update_y;
1639         end;
1640
1641         procedure show_single_sample(ind:integer);
1642         var
1643            temp : string[len];
1644         begin
1645            str((sample_buffer[ind] and $000f):len,temp);
1646            screen(temp,this_y,this_x,white,black);
1647            this_x := this_x + len;
1648            if this_x > 75
1649            then
1650               update_xy;
1651         end;
1652
1653         procedure show_reverse_sample(ind:integer);
1654         var
1655            temp : string[len];
1656         begin
1657            str((sample_buffer[ind] and $000f):len,temp);
1658            screen(temp,this_y,this_x,black,(white mod 8));
1659            this_x := this_x + len;
1660            if this_x > 75
1661            then
1662               update_xy;
1663         end;
1664
1665
1666         procedure show_enc_sample(i:integer);
1667         begin
1668            case sample_buffer[i] of
1669               tv_sync,
1670               tv_blank,
1671               tv_burst  : begin
1672                              if count <> 0
1673                              then
1674                              begin
1675                                 if (get_sample_type(sample_buffer[i-1]) =
1676                                     get_sample_type(sample_buffer[i]))
1677                                 then
1678                                    count := count + 1
1679                                 else
1680                                 begin
1681                                    show_count(get_sample_type(sample_buffer[
1682                                    count := 1;
1683                                 end;
1684                              end
1685                              else
1686                                 count := 1;
1687                           end;
1688               else
1689               begin
1690                  if count > 0
1691                  then
1692                  begin
```

```
1693                    show_count(get_sample_type(sample_buffer[i-1]));
1694                    count := 0;
1695                 end;
1696                 show_single_sample(i);
1697              end;
1698           end; (*case*)
1699        end;
1700
1701        procedure show_dec_sample(i:integer);
1702        begin
1703           if ((sample_buffer[i] and $10) <> 0)
1704           then
1705              show_single_sample(i)
1706           else
1707              show_reverse_sample(i);
1708        end;
1709
1710     begin
1711        compress_sample_buffer;
1712        count := 0;
1713        clrcenter(top,local_bot);
1714        this_y := top;
1715        this_x := left;
1716        for i := 0 to (buff_index - 1) do
1717        begin
1718           case machine of
1719              encoder : show_enc_sample(i);
1720              decoder : show_dec_sample(i);
1721              else
1722                 do_error(14);
1723           end; (*case*)
1724        end;
1725        if count > 0
1726        then
1727        begin
1728           show_count(get_sample_type(sample_buffer[i]));
1729           count := 0;
1730        end;
1731        hitreturn;
1732        clrcenter(top,local_bot);
1733        ask_save;
1734     end;
1735
1736     procedure enter_tvline_number;
1737     const
1738        len = 8;
1739        msg = 'Enter tvline number: ';
1740     var
1741        temp   : string[len];
1742        newval : integer;
1743     begin
1744        last_line_number:= 'ERR';
1745        screen(copy(blanks,2,78),23,2,boxcolor,back);
1746        screen(msg,23,2,boxcolor,back);
1747        readstr(length(msg)+2,23,temp,len);
```

```
1748          if (valid_int(temp,newval)) and (newval<=max_lines)
1749          then
1750          begin
1751             move(newval,sample_buffer[0],sizeof(newval));
1752             buff_index := sizeof(newval);
1753             expanded_buffer := false;
1754             expand_sample_buffer;
1755             str(newval,last_line_number);
1756          end
1757          else
1758          begin
1759             do_error(6);
1760          end;
1761          screen(copy(blanks,2,78),23,2,boxcolor,back);
1762       end;
1763
1764
1765       procedure get_cursor_movement(     xc         : xchar;
1766                                     var ytemp      : integer;
1767                                         top,
1768                                         ystart     : integer;
1769                                     var local_end: boolean);
1770       begin
1771          if (xc = up_key)
1772          then
1773          begin
1774             if ystart <> ytemp
1775             then
1776                ytemp := ytemp - 1
1777             else
1778                ytemp := ystart + top;
1779          end
1780          else if xc = down_key
1781          then
1782          begin
1783              if (ystart + top) <> ytemp
1784              then
1785                 ytemp := ytemp + 1
1786              else
1787                ytemp := ystart;
1788          end
1789          else if (xc = esc_key )
1790          then
1791             local_end := true
1792          else if (xc = home_key) or (xc = pgup_key)
1793          then
1794             ytemp := ystart
1795          else if (xc = end_key) or (xc = pgdn_key)
1796          then
1797          begin
1798             ytemp := ystart + top
1799          end
1800    (* changed to supress error of unimplemented key V1.10
1801          else
1802             do_error(7);
```

```
1803    *)
1804       end;
1805
1806       function get_int(var val:integer;max:integer):boolean;
1807       const
1808          len = 3;
1809          the_str = 'Enter value between 0 and ';
1810       var
1811          temp : str80;
1812       begin
1813          str(max:len,temp);
1814          eatlb(temp);
1815          temp := the_str + temp + ':    ';
1816          screen(temp,23,2,boxcolor,back);
1817          readstr(length(temp),23,temp,len);
1818          get_int := (valid_int(temp,val)) and (val<=max) and (val>=0);
1819       end;
1820
1821
1822       procedure edit_mix_block;
1823       const
1824          len = 8;
1825       var
1826          maxfields,
1827          xstart,
1828          ystart,
1829          ytemp,
1830          newval,
1831          i          : integer;
1832          temp       : string[len];
1833          local_end  : boolean;
1834          xc         : xchar;
1835
1836
1837       begin
1838          clrcenter(top,botcenter);
1839          maxfields := 10;
1840          xstart := 15;
1841          ystart := top+2;
1842          for i := 0 to 5 do
1843          begin
1844             str(mixdef.block[i].start:len,temp);
1845             screen('Skip   line:'+temp,ystart+(2*i),left,white,black);
1846             str(mixdef.block[i].length:len,temp);
1847             screen('Length line:'+temp,ystart+(2*i)+1,left,white,black);
1848          end;
1849          screen(copy(blanks,2,78),23,2,boxcolor,back);
1850          gotoxy(xstart,ystart);
1851          ytemp := ystart;
1852          local_end := false;
1853          repeat
1854             if (ytemp mod 2) = 0
1855             then
1856                str(mixdef.block[trunc((ytemp-ystart)/2)].start:len,temp)
1857             else
```

```
1858                str(mixdef.block[trunc((ytemp-ystart)/2)].length:len,temp);
1859             screen(temp,ytemp,xstart,black,(white mod 8));
1860             screen(copy(blanks,2,78),23,2,boxcolor,back);
1861             gotoxy(xstart,ytemp);
1862             xc := getxchar;
1863             screen(temp,ytemp,xstart,white,black);
1864             if xc = ret
1865             then
1866             begin
1867                if get_int(newval,525)
1868                then
1869                begin
1870                   if (ytemp mod 2) = 0
1871                   then
1872                      mixdef.block[trunc((ytemp-ystart)/2)].start := newval
1873                   else
1874                      mixdef.block[trunc((ytemp-ystart)/2)].length := newval;
1875                end
1876                else
1877                   do_error(6);
1878             end
1879             else
1880                get_cursor_movement(xc,ytemp,11,ystart,local_end);
1881          until local_end;
1882          clrcenter(top,botcenter);
1883       end;
1884
1885
1886       procedure edit_audio_and_marker;
1887       const
1888          len = 8;
1889       type
1890          strlen = string[len];
1891       var
1892          xstart,
1893          ystart,
1894          tot,
1895          ytemp,
1896          newval,
1897          i          : integer;
1898          temp       : strlen;
1899          local_end  : boolean;
1900          xc         : xchar;
1901
1902          function get_val_mix(ypos:byte):strlen;
1903          var
1904             temps : strlen;
1905             tempi : integer;
1906          begin
1907             case (ypos-ystart) of
1908                  0 : tempi := mixdef.audio.start;
1909                  1 : tempi := mixdef.audio.length;
1910                  2 : tempi := mixdef.marker.start;
1911                  3 : tempi := mixdef.marker.length;
1912                  4,6,8,10,12,14,16 :
```

```
1913                    tempi := (mixdef.marker_def[(ypos-ystart-4) div 2].level
1914                5,7,9,11,13,15,17 :
1915                    tempi := mixdef.marker_def[(ypos-ystart-5) div 2].count;
1916            end;(*case*)
1917            str(tempi:len,temps);
1918            get_val_mix := temps;
1919         end;
1920
1921         procedure put_mix_val(ypos,val:integer);
1922         begin
1923            case (ypos-ystart) of
1924                0 : mixdef.audio.start := val;
1925                1 : mixdef.audio.length := val;
1926                2 : mixdef.marker.start := val;
1927                3 : mixdef.marker.length := val;
1928                4,6,8,10,12,14,16 :
1929                    mixdef.marker_def[(ypos-ystart-4) div 2].level  := val;
1930                5,7,9,11,13,15,17 :
1931                    mixdef.marker_def[(ypos-ystart-5) div 2].count := val;
1932            end;(*case*)
1933         end;
1934
1935
1936         procedure try_get_int(ypos:integer);
1937         var
1938            dummy : boolean;
1939            val   : integer;
1940         begin
1941            case (ypos-ystart) of
1942                4,6,8,10,12,14,16:
1943                    dummy := get_int(val,15);
1944                0,1,2,3,5,7,9,11,13,15,17 :
1945                    dummy := get_int(val,525);
1946            end;(*case*)
1947            if dummy
1948            then
1949               put_mix_val(ypos,val)
1950            else
1951               do_error(6);
1952         end;
1953
1954      begin
1955         clrcenter(top,botcenter);
1956         xstart := 17;
1957         ystart := top+1;
1958         ytemp := ystart;
1959         tot    := 17;
1960
1961         screen('Audio  skip  :',ytemp,left,white,black);
1962         ytemp := ytemp + 1;
1963         screen('Audio  Length:',ytemp,left,white,black);
1964         ytemp := ytemp + 1;
1965
1966         screen('Marker skip  :',ytemp,left,white,black);
1967         ytemp := ytemp + 1;
```

```
1968          screen('Marker Length:',ytemp,left,white,black);
1969          ytemp := ytemp + 1;
1970
1971          for i := 0 to 6 do
1972          begin
1973             screen('Marker level :',ytemp+(2*i),left,white,black);
1974             screen('Level  length:',ytemp+(2*i)+1,left,white,black);
1975          end;
1976
1977          for ytemp := ystart to (ystart + tot) do
1978          begin
1979             screen(get_val_mix(ytemp),ytemp,xstart,white,black);
1980          end;
1981          gotoxy(xstart,ytemp);
1982          ytemp := ystart;
1983          local_end := false;
1984          repeat
1985             temp := get_val_mix(ytemp);
1986             screen(temp,ytemp,xstart,black,(white mod 8));
1987             screen(copy(blanks,2,78),23,2,boxcolor,back);
1988             gotoxy(xstart,ytemp);
1989             xc := getxchar;
1990             screen(temp,ytemp,xstart,white,black);
1991             if xc = ret
1992             then
1993             begin
1994                try_get_int(ytemp);
1995             end
1996             else
1997                get_cursor_movement(xc,ytemp,tot,ystart,local_end);
1998          until local_end;
1999
2000          clrcenter(top,botcenter);
2001       end;
2002
2003       procedure put_mix;
2004       begin
2005          compress_sample_buffer;
2006          move(mixdef,sample_buffer[0],sizeof(mixdef));
2007          buff_index := sizeof(mixdef);
2008          expand_sample_buffer;
2009       end;
2010
2011       procedure show_dsu_errorlog;
2012       const
2013          len = 8;
2014          msg = 'DSU errorlog: ';
2015       var
2016          temp : string[len];
2017          i,j  : integer;
2018          l    : integer;
2019       begin
2020          l := length(msg);
2021          compress_sample_buffer;
2022          move(sample_buffer[0],errorlog,sizeof(errorlog));
```

```
2023          clrcenter(top,botcenter);
2024          screen(msg,top+1,left,white,black);
2025          for i := 1 to 10 do
2026          begin
2027             for j := 1 to 5 do
2028             begin
2029                str(errorlog[10*(j-1)+i] div 17:len,temp);
2030                screen(temp,
2031                       (top+i),
2032                       left+1+len*(j-1),
2033                       white,black);
2034             end;
2035          end;
2036          hitreturn;
2037          clrcenter(top,botcenter);
2038       end;
2039
2040       procedure execute_command(the_cmd:str80);
2041       var
2042          cmd_array_element : command_type;
2043       begin
2044          case parse_command(cmd_array_element,the_cmd) of
2045                valid : begin
2046                           send_command(cmd_array_element);
2047                        end;
2048                writebuf:
2049                        begin
2050                           write_sample_buffer;
2051                        end;
2052                readbuf:begin
2053                           read_sample_buffer;
2054                        end;
2055                mixdef_show
2056                       :begin
2057                             show_mixdef;
2058                        end;
2059                edit_mixdef_block
2060                       :begin
2061                             edit_mix_block;
2062                        end;
2063                put_mixdef
2064                       :begin
2065                             put_mix;
2066                        end;
2067                change_port
2068                       :begin
2069                             change_rs232port;
2070                        end;
2071                enter_number
2072                       :begin
2073                             enter_tvline_number;
2074                        end;
2075                edit_audio
2076                       :begin
2077                             edit_audio_and_marker;
```

```
2078                        end;
2079                  show_dsu_tvline
2080                        :begin
2081                             show_tvline(encoder);
2082                        end;
2083                  show_audio_data
2084                        :begin
2085                             show_tvline(decoder);
2086                        end;
2087                  show_errorlog
2088                        :begin
2089                             show_dsu_errorlog;
2090                        end;
2091                  abort : begin
2092                             end_of_sesion := true;
2093                        end;
2094                  empty : begin
2095                             (*nothing to do*)
2096                        end;
2097                  else
2098                  begin
2099                     do_error(1);
2100                  end;
2101            end; (*case*)
2102      end;
2103
2104
2105      procedure show_cmd_list;
2106      var
2107         tcp : cmd_list_ptr_type;
2108      begin
2109         tcp := cmd_list_ptr;
2110         while tcp <> nil do
2111         begin
2112            with tcp^ do
2113               screen(text,ypos,xpos,fore,back);
2114            tcp := tcp^.next;
2115         end;
2116      end;
2117
2118      procedure set_up_menu;
2119      begin
2120         show_cmd_list;
2121      end;
2122
2123      procedure get_choice(var c_ptr:cmd_list_ptr_type);
2124      var
2125         chosen : boolean;
2126         xc     : xchar;
2127      begin
2128         chosen := false;
2129         repeat
2130            with c_ptr^ do
2131            begin
2132               if redraw
```

```
2133                then
2134                   show_cmd_list;
2135                screen(text,ypos,xpos,back,(fore mod 8));
2136                gotoxy(xpos+length(text),ypos);
2137                xc := getxchar;
2138                screen(text,ypos,xpos,fore,back);
2139             end;
2140             if xc = ret
2141             then
2142                chosen := true
2143             else if (xc = up_key) or (xc = left_key)
2144             then
2145                c_ptr := c_ptr^.prev
2146             else if (xc = down_key) or (xc = right_key)
2147             then
2148             begin
2149                if c_ptr^.next <> nil
2150                then
2151                   c_ptr := c_ptr^.next
2152                else
2153                   c_ptr := cmd_list_ptr;
2154             end
2155             else if (xc = home_key) or (xc = pgup_key)
2156             then
2157                c_ptr := cmd_list_ptr
2158             else if (xc = end_key) or (xc = pgdn_key)
2159             then
2160             begin
2161                 while c_ptr^.next <> nil do
2162                    c_ptr := c_ptr^.next;
2163             end
2164             else if (xc = help_key) or (xc = prev_key)  (* F3 and F1 *)
2165             then
2166             begin
2167                 do_help(c_ptr^.helpfile);
2168             end
2169    (* changed to supress error of unimplemented key V1.10
2170             else
2171                do_error(7);
2172    *)
2173          until chosen;
2174       end;
2175
2176       function test_info:boolean;
2177       begin
2178          test_info := true;
2179          if all_cmd[1].word = ''
2180          then
2181          begin
2182             do_error(18);
2183             test_info := false;
2184          end;
2185          if (cmd_list_ptr = nil) and (command_level = high)
2186          then
2187          begin
```

```
2243   begin
2244      initialize;
2245      if total_rs232_ports > 0
2246      then
2247         doit
2248      else
2249         do_error(8);
2250      closedown;
2251   end.
```

REFERENCE OF PROCEDURES AND FUNCTIONS

	PAGE	LINE	NAME
2299	8	408	getcomspec(var result:str80);
2300			(10, 498) (9, 469)
2301	5	273	getxchar:xchar;
2302			(40, 2137) (37, 1989) (35, 1862) (29, 1581) (22, 11
2303	17	884	get_back(var p:cmd_list_ptr_type;var ok:boolean);
2304			(19, 1011)
2305	39	2123	get_choice(var c_ptr:cmd_list_ptr_type);
2306			(41, 2203)
2307	23	1247	get_command(var cmd:str80);
2308			(41, 2222)
2309	33	1765	get_cursor_movement(xc : xchar;
2310			(37, 1997) (35, 1880)
2311	16	856	get_fore(var p:cmd_list_ptr_type;var ok:boolean);
2312			(19, 1010)
2313	17	928	get_helpfile(var p:cmd_list_ptr_type;var ok:boolean);
2314			(19, 1013)
2315	34	1806	get_int(var val:integer;max:integer):boolean;
2316			(37, 1994) (36, 1945) (36, 1943) (35, 1867)
2317	18	944	get_low_commands(var p:cmd_list_ptr_type;var ok:boolean);
2318			(19, 1014)
2319	17	912	get_redraw(var p:cmd_list_ptr_type;var ok:boolean);
2320			(19, 1012)
2321	30	1591	get_sample_type(sample:byte):the_sample_types;
2322			(32, 1728) (32, 1693) (31, 1681) (31, 1676) (31, 1
2323	15	786	get_text(var p:cmd_list_ptr_type;var ok:boolean);
2324			(19, 1007)
2325	14	755	get_token(var f:text;var s:str80):boolean;
2326			(21, 1098) (20, 1094) (20, 1090) (20, 1086) (20, 1
2327			(17, 916) (17, 892) (16, 864) (16, 836) (15,
2328	35	1902	get_val_mix(ypos:byte):strlen;
2329			(37, 1985) (37, 1979)
2330	15	800	get_xpos(var p:cmd_list_ptr_type;var ok:boolean);
2331			(19, 1008)
2332	16	828	get_ypos(var p:cmd_list_ptr_type;var ok:boolean);
2333			(19, 1009)
2334	41	2193	high_level;
2335			(41, 2232)
2336	25	1334	high_nible(b:byte):byte;
2337			(25, 1351)
2338	6	315	hitreturn;
2339			(38, 2036) (32, 1731) (28, 1517) (27, 1477) (14,
2340	22	1163	Initialize;
2341			(42, 2244)
2342	20	1063	init_cmds;
2343			(22, 1176)
2344	9	472	init_dataseg;
2345			(22, 1169)
2346	41	2213	low_level;
2347			(41, 2231)

REFERENCE OF PROCEDURES AND FUNCTIONS

	PAGE	LINE	NAME
2252	29	1542	ask_save;
2253			(32, 1733)
2254	7	329	beep;
2255			(7, 377) (7, 365)
2256	11	592	bioscom(cmd:byte;var ch :byte;port:integer;var status:integer)
2257			(13, 674) (13, 663) (12, 646) (12, 641) (12, 6
2258	4	208	box(topline,topcol,botline,botcol:integer);
2259			(14, 750) (14, 734)
2260	22	1197	change_rs232port;
2261			(38, 2069)
2262	41	2238	closedown;
2263			(42, 2250)
2264	4	195	clrcenter(from,till:integer);
2265			(38, 2037) (38, 2023) (37, 2000) (36, 1955) (35, 18
2266			(30, 1609) (28, 1518) (28, 1491) (28, 1478) (27, 14
2267	8	389	colormonitor(var st:scrn_type);
2268			(10, 500)
2269	24	1285	compress_sample_buffer;
2270			(37, 2021) (37, 2005) (32, 1711) (27, 1476) (26, 14
2271	24	1289	convert(h,l:byte):integer;
2272			(24, 1311)
2273	41	2227	doit;
2274			(42, 2247)
2275	13	702	do_error(i:integer);
2276			(42, 2249) (41, 2188) (40, 2182) (40, 2171) (39, 2(
2277			(33, 1759) (32, 1722) (30, 1608) (29, 1568) (26, 14
2278			(23, 1217) (23, 1205) (21, 1145) (20, 1075) (20, 1(
2279	23	1213	do_help(filename:str80);
2280			(40, 2167) (23, 1254)
2281	23	1222	dump_help(filename:str80);
2282			(41, 2221)
2283	5	234	eatlb(var s:str255);
2284			(34, 1814) (24, 1262) (14, 763)
2285	5	229	eattb(var s:str255);
2286			(24, 1263) (14, 764)
2287	35	1886	edit_audio_and_marker;
2288			(38, 2077)
2289	34	1822	edit_mix_block;
2290			(38, 2061)
2291	32	1736	enter_tvline_number;
2292			(38, 2073)
2293	6	302	escapepressed:boolean;
2294			(13, 673) (13, 662) (12, 644)
2295	38	2040	execute_command(the_cmd:str80);
2296			(41, 2223) (41, 2207)
2297	25	1320	expand_sample_buffer;
2298			(37, 2008) (33, 1754) (27, 1462) (26, 1381)

REFERENCE OF PROCEDURES AND FUNCTIONS

	PAGE	LINE	NAME
2348	18	952	low_level_ok(var y:actual_cmd_ptr_type;s:str80):boolean;
2349			(18, 973)
2350	25	1324	low_nible(b:byte):byte;
2351			(25, 1350)
2352	15	782	new_field_ok(var p:cmd_list_ptr_type):boolean;
2353			(19, 1030)
2354	14	721	not yet implemented';
2355			
2356	23	1258	parse_command(var ret_cmd:command_type;cmd:str80):status;
2357			(38, 2044)
2358	37	2003	put_mix;
2359			(38, 2065)
2360	36	1921	put_mix_val(ypos,val:integer);
2361			(36, 1949)
2362	7	333	readstr(x,y:integer;var astring:str80;len:byte);
2363			(34, 1817) (32, 1747) (27, 1435) (23, 1251)
2364	27	1451	read_sample_buffer;
2365			(38, 2053)
2366	13	694	rs232ports:integer;
2367			(22, 1171)
2368	12	632	rs232_init(port:byte):boolean;
2369			(26, 1374)
2370	13	679	rs232_message(end_val,port:byte):boolean;
2371			(26, 1397)
2372	13	666	rs232_rec(var ch:byte;port:byte):boolean;
2373			(26, 1409) (26, 1385) (13, 688)
2374	12	654	rs232_send(ch,port:byte);
2375			(26, 1393) (26, 1384)
2376	12	619	rs232_stat(port:byte):integer;
2377			(26, 1415) (26, 1375) (13, 671) (13, 660) (12,
2378	10	504	savescreen(save:boolean);
2379			(23, 1218) (23, 1215)
2380	4	185	savewindow(var alot:int80;index,start,length:integer);externa
2381			(10, 528)
2382	29	1548	save_buff_to_file(name : str80);
2383			(30, 1586)
2384	4	188	screen(lin:str255;row,col,fore,back:integer);
2385			(40, 2138) (40, 2135) (39, 2113) (38, 2030) (38, 2
2386			(37, 1979) (37, 1974) (37, 1973) (37, 1968) (36, 1
2387			(35, 1860) (35, 1859) (34, 1849) (34, 1847) (34, 1
2388			(32, 1745) (31, 1658) (31, 1646) (30, 1637) (30, 1
2389			(28, 1511) (28, 1506) (28, 1502) (28, 1497) (28, 1
2390			(27, 1434) (27, 1432) (26, 1416) (26, 1372) (23, 1
2391			(22, 1172) (22, 1159) (22, 1158) (14, 751) (14,

REFERENCE OF PROCEDURES AND FUNCTIONS

```
      PAGE   LINE   NAME

2392                ( 7, 362)  ( 7, 346)  ( 6, 327)  ( 6, 320)  ( 5, 2
2393                ( 4, 216)  ( 4, 205)
2394    4    190   scrsub(var lin:str255;row,col,colr:integer);external 'scrsub.t
2395                ( 4, 193)
2396   25   1358   send_command(cmd_el:command_type);
2397                ( 38, 2046)
2398   14    743   setupscreen;
2399                ( 22, 1172)
2400   15    770   set_up_cmd_list;
2401                ( 22, 1186)
2402   39   2118   set_up_menu;
2403                ( 41, 2198)
2404   39   2105   show_cmd_list;
2405                ( 40, 2134)  ( 39, 2120)
2406   30   1621   show_count(s_t:the_sample_types);
2407                ( 32, 1728)  ( 32, 1693)  ( 31, 1681)
2408   32   1701   show_dec_sample(i:integer);
2409                ( 32, 1720)
2410   37   2011   show_dsu_errorlog;
2411                ( 39, 2089)
2412   31   1666   show_enc_sample(i:integer);
2413                ( 32, 1719)
2414   21   1149   Show_logo;
2415                ( 22, 1173)
2416   28   1482   show_mixdef;
2417                ( 38, 2057)
2418   14    738   show_port;
2419                ( 26, 1371)  ( 23, 1208)  ( 14,  752)
2420   31   1653   show_reverse_sample(ind:integer);
2421                ( 32, 1707)
2422   31   1641   show_single_sample(ind:integer);
2423                ( 32, 1705)  ( 32, 1696)
2424   29   1529   show_tvline(machine:machine_type);
2425                ( 39, 2085)  ( 39, 2081)
2426    5    268   spekeys(xch:xchar):boolean;
2427                ( 7,  351)  ( 7,  348)
2428    5    253   strcmp(a,b:str80):boolean;
2429                ( 24, 1276)  ( 24, 1272)  ( 21, 1134)  ( 21, 1131)  ( 21, 1
2430                ( 21, 1116)  ( 21, 1113)  ( 21, 1110)  ( 21, 1107)  ( 21, 1
2431                ( 19,  996)  ( 19,  995)  ( 18,  966)  ( 18,  954)  ( 17,
2432   40   2176   test_info:boolean;
2433                ( 41, 2218)  ( 41, 2200)
2434    6    288   time(var h,m,s,f:integer);
2435                ( 22, 1190)  ( 22, 1189)  ( 22, 1174)  ( 22, 1165)
2436   36   1936   try_get_int(ypos:integer);
2437                ( 37, 1994)
2438   30   1614   update_xy;
2439                ( 31, 1662)  ( 31, 1650)  ( 30, 1636)
2440   30   1602   update_y;
2441                ( 31, 1638)  ( 30, 1617)
```

EFERENCE OF PROCEDURES AND FUNCTIONS

	PAGE	LINE	NAME
2442	28	1521	valid_int(s:str80;var int:integer):boolean;
2443			(34, 1818) (33, 1748)
2444	4	186	writescr(var alot:int80;index,start,length:integer);external
2445			(10, 543)
2446	26	1421	write_sample_buffer;
2447			(38, 2050)