Portierung eines Video-Codecs auf DROPS

Carsten Rietzschel
cr7@os.inf.tu-dresden.de
Technische Universität Dresden
Fakultät Informatik

17.12.2002

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Aufgabe

Im Rahmen einer Projektarbeit soll ein echtzeitfähiger Videorekorder für DROPS (Dresden Real-Time OPerating System) implementiert werden. Eine Integration des zu erstellenden Systems in andere Anwendungen - vorzugsweise über das DROPS Streaming Interface (DSI) - sollte möglich sein und somit die Wiederverwendbarkeit der Software steigern.

Dieser Große Beleg widmet sich den Teilaufgaben, einen modernen Video-Codec auf DROPS zu portieren, sowie die nötigen Programme zum Testen des Enkodierens und Dekodierens von Videodaten zu implementieren. Dabei sollen bereits die oben genannten Projekt-Ziele Berücksichtigung finden.

1.2 Aufbau der Arbeit

Innerhalb der insgesamt sechs Kapitel wird der Entwicklungsprozess von einer Bestandsaufnahme über den Entwurf und die Implementierung hin zu einer Leistungsbewertung und dem daraus folgenden Fazit mit Hinblick auf zukünftige Entwicklungen dargestellt.

In Kapitel 2 werden die Grundlagen für das Verständnis der weiteren Arbeit vermittelt. Auch wird in diesem Kapitel auf aktuelle verwandte Arbeiten eingegangen und diese kurz vorgestellt. Insbesondere umfaßt dies Informationen zum *Dresden Realtime OPerating System* sowie Informationen zum Thema Video und Videoverarbeitung.

Das darauf folgende Kapitel ist dem Entwurf gewidmet. Dabei werden verschiedene Möglichkeiten zur Realisierung des Projektzieles "Videorekorder" vorgestellt und diskutiert. Besonderes Augenmerk liegt auf der Durchführbarkeit dieser Ideen sowie deren Eigenschaften im Hinblick auf die geforderte Wiederverwendbarkeit sowie Echtzeitfähigkeit.

Interessante Aspekte der Implementierung der Testumgebung und der Portierung eines Codecs auf DROPS werden in Kapitel 4 behandelt. Dieses Kapitel kann auch als Anhaltspunkt für die Übertragung anderer Codecs auf DROPS Verwendung finden, da aufgetretene Probleme erläutert werden. Weiterhin wird die Testumgebung detailliert vorgestellt und die wichtigsten Aspekte im Hinblick auf die Aufgabenstellung betrachtet.

Das vorletzte Kapitel befaßt sich mit einer Leistungsbewertung der implementierten Testumgebung inklusive des portierten Codecs. Dabei finden sich Details über die einzelnen Komponenten der Testumgebung. Ebenfalls wird ein Vergleich mit der Leistungsfähigkeit der unter Linux verfügbaren Programme gezogen.

Kapitel 6 gibt eine kurze Zusammenfassung dieses Belegs und Einblicke in die weitere Entwicklung hin zum Projektziel.

2 Grundlagen und verwandte Arbeiten

2.1 Echtzeitfähigkeit

Echtzeitfähigkeit ist gleichbedeutend mit der Fähigkeit eines Systems, alle seine *Deadlines* zu erreichen, d.h. das System muß alle seine Berechnungen innerhalb einer durch die Umgebung festgelegten Zeitspanne, deren Ende als Deadline bezeichnet wird, abgeschlossen haben. Ein wichtiger Punkt der Echtzeitfähigkeit eines Systems (*Echtzeitsystem*, *Real-Time System*) ist also die Kenntnis der Dauer der durchzuführenden Berechnungen. Die maximale Dauer unter den schlecht möglichsten Bedingungen (*Worst-Case*) für eine solche Berechnung wird als *Worst-Case-Time* bezeichnet.

Ein mögliche Gruppierung von Echtzeitsystemen kann dahingehend vorgenommen werden, daß

1. die Deadlines unbedingt eingehalten werden müssen oder
2. die Deadlines eingehalten werden sollten.

Der erstgenannte Fall wird als *Hartes Echtzeitsystem* bezeichnet, letzterer als *weiche*s.

Relevant für die Wiedergabe von Videos sind u.a. die Eigenschaften der menschlichen Sinnesorgane. So nimmt der Mensch Filme, die mit 25 Bildern pro Sekunde und mehr dargestellt werden, flüssig wahr. Auch sollte das Bild und der zugehörige Ton nicht weiter als ± 80 ms versetzt abgespielt werden, da dies sonst als nicht lippensynchron empfunden wird [2]. Diese Bedingungen bestimmen die Parameter für das Scheduling eines Echtzeitsystems zum Aufzeichnen und Wiedergeben von Videos. So ergibt sich ein sich wiederholender Vorgang mit einer *Periode* von höchstens 40 ms. In dieser Periode muß ein komplettes Bild verarbeitet werden.

2.2 DROPS

Das Dresden Real-Time OPerating System (kurz DROPS) ist ein an der TU-Dresden entwickeltes Forschungsprojekt. Ziel dieses Projektes ist es, Quality-of-Service-Forderungen von Anwendungen zu unterstützen. DROPS ermöglicht das Betreiben von Echtzeitanwendungen zusammen mit Time-sharing-Anwendungen. Es basiert auf einem Microkernel der 2. Generation - dem DROPS-Microkernel. Ein wichtiger Bestandteil ist L4Linux. Dies ist ein Linux-Server, der auf dem Microkernel aufsetzt und die Möglichkeit bietet, "Time-sharing"-Komponenten betreiben zu können. So können weiterhin normale Linux-Anwendungen gleichzeitig mit Echtzeit-Anwendungen ausgeführt werden. Abbildung 1 zeigt schematisch die Struktur von DROPS.

2.2.1 DROPS Streaming Interface

Das DROPS Streaming Interface (kurz *DSI*) bietet eine schnelle und komfortable Möglichkeit, größere Datenmengen sowohl zwischen verschiedenen Adreßräumen als auch innerhalb eines Adreßraumes auszutauschen. Realisiert ist dies durch zwei shared-memory-Bereiche und durch Kontrollfunktionen, die in Form einer Bibliothek verfügbar sind. Durch ein vom Microkernel unterstütztes Einblenden von Seiten, dem sogenannten *Mapping*, in die beteiligten Adreßräume wird der Datenaustausch ohne zusätzliche und

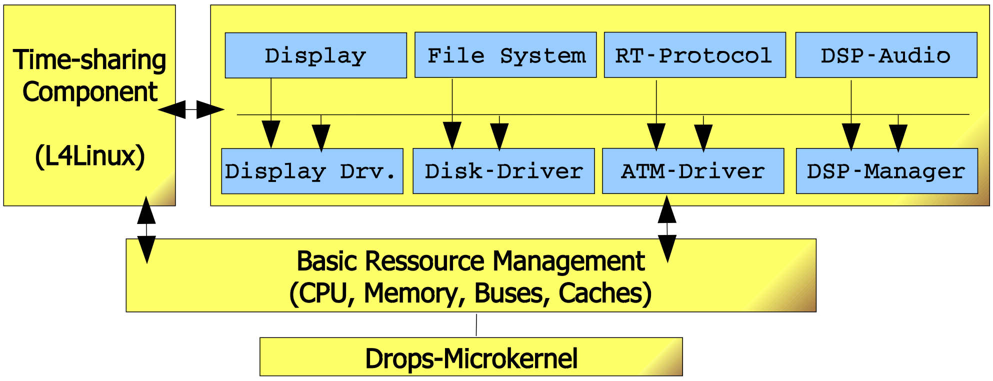

Abbildung 1: DROPS aus [3]

damit langsame Kopieroperationen realisiert. Eine typische Beispielapplikation wäre das in Abbildung 2 gezeigte Erzeuger-Verbraucher-Problem.

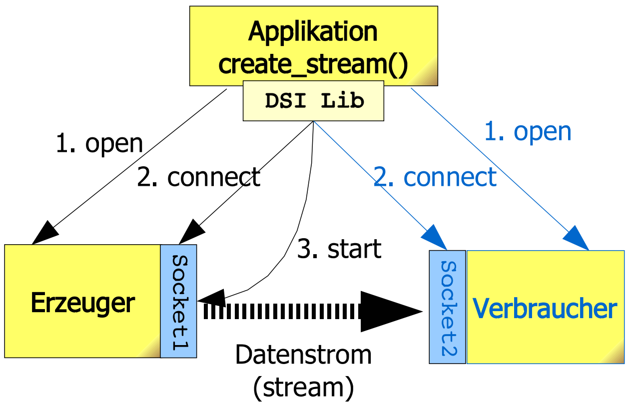

Abbildung 2: DSI - Struktur des DROPS Streaming Interface aus [4]

Nach dem Initialisieren des DROPS Streaming Interfaces mit *dsi_init* können die zur Datenübertragung nötigen Funktionen aufgerufen werden. Der Datenaustausch erfolgt über einen Datenstrom, der von der Anwendung mit *create_stream* erstellt wird. Er verbindet die *sockets* des Erzeugers und des Verbrauchers. Dieser *stream* sorgt danach selbständig für den Austausch der Daten. Als Transporter dienen dabei eine endliche Anzahl von Paketen, welche in einem Ringpuffer organisiert sind. Die Verwaltung der Pakete erfolgt in deren Kontrollbereich. Die für die zu transportierenden Daten zuständige Region des Hauptspeichers wird Datenbereich genannt. Er unterliegt keiner Verwaltung oder Kontrolle durch DSI. Für das DROPS Streaming Interface ist der Datenbereich ein unstrukturierter Teil im Adreßraum. Sowohl auf Erzeuger- als auch auf Verbraucherseite werden die gleichen Bibliotheksfunktionen genutzt. Zum Beispiel liefert die Funktion *get_packet* einen Paketdeskriptor, während *commit_packet* einen Deskriptor freigibt. Daten werden mit *add_data* vom Erzeuger bereitgestellt, während *get_data* verbraucherseitig die Daten liefert. Abbildung 3 demonstriert diese Vorgänge.

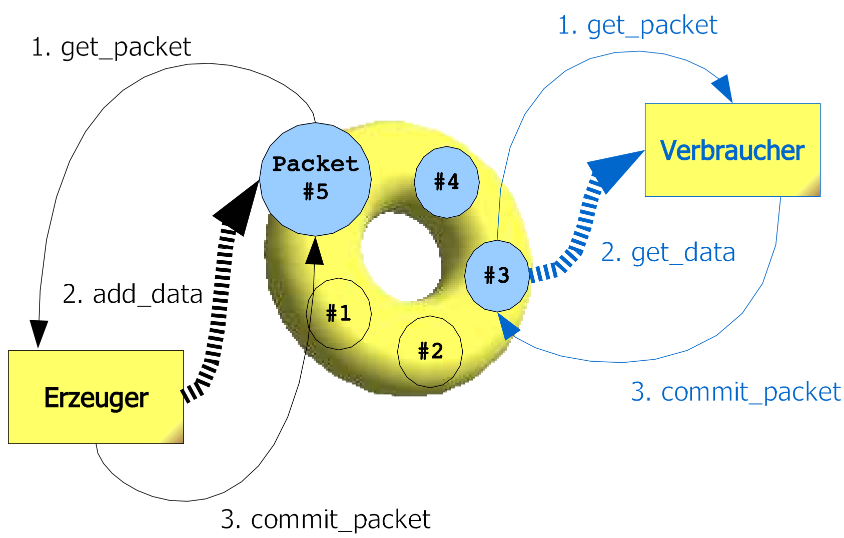

Abbildung 3: DSI - Ringpuffer mit Paketfunktionen

Zusammenfassend lassen sich folgenden Vorteile von DSI nennen:

- einfache Schnittstelle (gekapselt in einer Client-Bibliothek)
- Reduzierung der notwendigen Kopieroperationen
- die Struktur der zu übertragenden Daten ist irrelevant für das Funktionieren von DSI

Für den Fall, daß Erzeuger und Verbraucher denselben Adreßraum benutzen, bringt die Verwendung von DSI keine Vorteile gegenüber einer Sicherung des shared-memory-Bereiches durch Semaphoren. DSI verursacht in diesem speziellen Fall einen höheren Rechenaufwand aufgrund seiner komplexeren Synchronisationsmechanismen.

2.2.2 DSI_staticfs

DSI_staticfs ist eine RAM-Disk, die für andere Systemkomponenten über das DROPS Streaming Interface Zugriffsmöglichkeiten auf Dateien und deren Inhalt bereitstellt. Dazu stehen neben den DSI-üblichen Funktionen u.a. folgende bereit:

- *dsi_staticfs_get_noof_files* - gibt die Anzahl der auf der RAM-Disk gespeicherten Dateien an
- *dsi_staticfs_get_file_name* - gibt den Namen einer bestimmten Datei zurück, dies erfordert die Übergabe einer Dateinummer
- *dsi_staticfs_open* - initialisiert auf der Seite des staticfs alle nötigen Datenstrukturen zur folgenden Datenübertragung

Das DSI_staticfs ist momentan die geeignetste Komponente, mit der sich die Funktionalität von DSI in andere Anwendungen einbauen und testen läßt.

2.2.3 DROPSCON

Die DROPS Konsolenanwendung (kurz *DROPSCON* oder *CON*) ist eine Konsolenkomponente für die DROPS Umgebung.

CON bietet eine definierte Schnittstelle zur Ausgabe von Grafiken und Texten (*pSlim*) sowie die Möglichkeit, Werte von Eingabegeräten auszulesen. Diese Funktionalität wird anderen Systemkomponenten durch Bibliotheken zur Verfügung gestellt. Desweiteren ermöglicht CON das parallele Nutzen mehrerer virtueller Konsolen gleichzeitig, die vom Nutzer umgeschalten werden können.

Eine genauere Betrachtung der Konsolenkomponente kann in [4] gefunden werden. Für zukünftige Entwicklungen ist auch das seit kurzem verfügbare *DOpE (Desktop Operation Environment)* [5] zu berücksichtigen, welches in naher Zukunft die CON ersetzen soll.

2.3 Container

Beim Darstellen eines Videos ist die Interpretation zeitrelevanter Informationen für die Wiedergabe unverzichtbar. Stellt man sich im einfachen Fall ein Video nur als eine Folge von Bildern vor, wird ein Problem deutlich: Die Zusammenstellung der Bilder sagt nichts darüber aus, mit welcher Geschwindigkeit diese dargestellt werden sollen. Schnell wird klar, daß dies durch die Angabe einer Framerate gelöst wird. So bedeutet beispielsweise eine Framerate von 25 Bildern pro Sekunde (kurz *FPS* für *frames per second*), daß alle 40 ms ein Bild dargestellt werden muß.

Wird das Modell der Bildfolgen um Audio erweitert, wird sichtbar, daß die Angabe der Framerate nicht ausreichend ist. Die Abhängigkeit des Tons vom jeweiligen Bild läßt sich daraus aber nicht zum Abspielen rekonstruieren.

Um alle nötigen Informationen zur korrekten Darstellung von Audio- und Video zu kapseln, wird ein *Container* verwendet. In diesem ist unter anderem die Framerate und die Beziehung zwischen Bild und Ton festgelegt. Ein Container ist *unabhängig vom verwendeten Codec* (siehe Abschnitt 2.4 Codecs).

Als bekanntester und am häufigsten anzutreffender Vertreter gilt das *Audio Video Interleave* (kurz *AVI*), welches von Microsoft entwickelt wurde [6]. Bild 4 zeigt eine mögliche Anordnung der Daten in einem Container. Tabelle 1 informiert über weitere Container-Formate, die im Rahmen dieser Arbeit jedoch nicht detaillierter betrachtet werden.

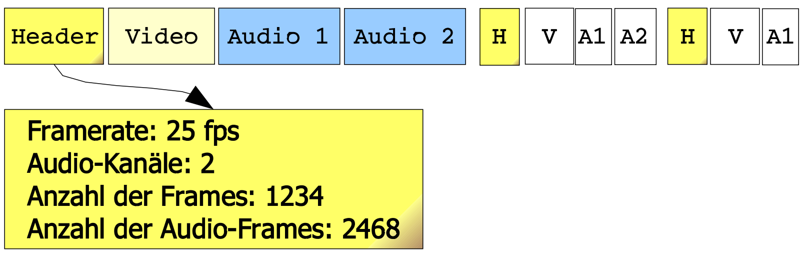

Abbildung 4: Container - Schema

Container	kurze Beschreibung
AVI	*Audio Video Interleave*; Microsoft; große Verbreitung; nicht Streaming-fähig
ASF	*Advanced Streaming Format*; Microsoft; verbesserte Streaming-Eigenschaften [7]
MOV	Quicktime-Movie-Format; Apple
MP4	MPEG-4; standardisiert; an MOV angelehnt; für MPEG-4 Video und Audio konzeptioniert

Tabelle 1: Container - Formate

2.4 Codecs

Als Codec wird ein Algorithmus bezeichnet, der Daten kodiert und dekodiert (*enCOde* und *DECode*, engl.). Ein typisches Merkmal moderner Codecs ist, daß der Enkodiervorgang im allgemeinen sehr viel länger dauert als das Dekodieren. Meist ist das Ziel des Enkodierens die Kompression des zu bearbeitenden Materials. Unterschieden wird nach verlustbehafteter und verlustfreier Kompression. Die im folgenden vorgestellten Video-Codecs sind der Kategorie verlustbehaftet zuzuordnen. Um die Notwendigkeit dieser Kompression zu verdeutlichen, zeigt Tabelle 2 ein typisches Beispiel der Datenmengen für Videos ohne Sound. Die angegebenen Werte können nur Richtwerte sein, exakte Werte sind vom Bildmaterial und der gewünschten visuellen Qualität[1] abhängig.

Format	Bemerkung	etwaige Größe	Qualität
RAW	Unkodiert, CIF, YUV, 25FPS,ca. 60s Länge, ohne Ton	200 MB (100%)	$\oplus\oplus$
MPEG-1	MPEG-1 kodiert	20 MB (10 %)	$\ominus$
MPEG-2	MPEG-2 kodiert , DVD	30 MB (15 %)	$\oplus$
MPEG-4	XviD kodiert	3 MB (1,5%)	$\oplus$

Tabelle 2: Kompression - Videogrößen

2.4.1 Moving Pictures Experts Group (MPEG)

Die Moving Pictures Experts Group (kurz MPEG) ist eine Arbeitsgruppe der ISO/IEC, die Standards für kodiertes digitales Audio und Video erstellt. Die 1988 gegründete Gruppe erstellte u.a. die MPEG-1,2,4 und 7 Video-Standards (vgl. Tabelle 3). Weitere Details zur Thematik MPEG finden Sie in [10].

Im allgemeinen wird MPEG auch als Name der erstellten Standards bzw. Codecs angesehen. Über die Standards MPEG-7 und 21 sind zur Zeit sehr wenige Informationen öffentlich bekannt. Ein Problem, welches die Verbreitung der Informationen über aktuelle MPEG Standards einschränkt, ist die Tatsache, daß die standardbeschreibenden Dokumente nur gegen eine Gebühr bei den nationalen Vertretungen der ISO erworben werden können. Daher wird sehr oft auf eine Referenz-Implementierung der ISO zurückgegriffen, diese verbessert und für die verschiedenen Architekturen optimiert.

Als Gemeinsamkeit von MPEG-1, MPEG-2 und MPEG-4 gilt die Aufteilung der Videodaten in Einzelbilder, den Frames. Dabei werden verschiedene Arten von Frames unterschieden, die teilweise in Abhängigkeit zueinander stehen, was Tabelle 4 zeigt.

[1] subjektive visuelle Qualitätsbeurteilung: sehr gut $\oplus\oplus$, gut $\oplus$, schlecht $\ominus$

Standard	Verwendung	Eigenschaften
MPEG-1	Video-CD, MP3 (MPEG-1 Layer III Audio)	Video: Frame basierend; Kompression von Bildgruppen nach JPEG-Verfahren Audio: Stereo/2-Kanal mit 3 Qualitätsstufen
MPEG-2	DVD, Digitales Fernsehen	Video: Frame basierend; ähnlich MPEG-1, jedoch gesteigerte Skalierbarkeit und Qualität Audio: 5-Kanal + 1-Bass und/oder 7-Kanal für Sprachen oder Kommentare
MPEG-4	Internet Übertragung, "Mobil Multimedia"	flexible,objektbasierende Kodierung u.a. für Audio und Video Video: Frame basierend; Audio: AAC
MPEG-7	Metadaten	Standard für die Beschreibung und das Suchen von Audio- und Video-Material
MPEG-21	"Multimedia Framework"	in der Entwicklung

Tabelle 3: MPEG - Formate

I-Frame	intra coded frame: vollständiges Einzelbild als Referenz für folgende Frames
P-Frame	predictive coded picture: Bildunterschiede zum Vorausgehenden I- oder P-Frame
B-Frame	bidirectionally predictive coded picture: Daten zur Interpolation von benachbarten I- und P-Frame
D-Frame	DC coded picture: niedrige Frequenzanteile der I-Frames für schnellen Vor- oder Rücklauf

Tabelle 4: MPEG - Frametypen aus [11]

Zwei Stichworte bzw. Techniken des modernsten Vertreters der MPEG-Video-Codecs, dem MPEG-4, werden hier im Auszug vorgestellt. Im hier teilweise übersetzten und wiedergegebenen Dokument [12] werden diese ausführlicher erläutert und weitere Details zu MPEG-4 dargestellt.

- *Media Objects:* Der MPEG-4 Standard ist komplett objektbasierend. Audiovisuelle Szenen bestehen ebenfalls aus einer Reihe von Objekten, die in einer Hierachie geordnet sind.
  Mögliche Objekte einer solchen Szene sind beispielsweise:
  - unbewegte Bilder (z.B. der Hintergrund einer Szene)
  - Video-Objekte (z.B. eine sprechende Person im Vordergrund ohne Hintergrund)
  - Audio-Objekte (z.B. die Stimme dieser Person)
- *Sprite Coding:* Angenommen das Vordergrund-Objekt läßt sich vom Hintergrund vor der Enkodierung trennen. Das große Hintergrundpanorama (Sprite) wird dann als erstes Frame zum Client übertragen und dort in einem Puffer gespeichert. Dort verbleibt es. In den folgenden Frames werden nur Informationen zur Kamerasteuerung übertragen. Dies ermöglicht das Rekonstruieren des Hintergrundbildes für jedes einzelne Frame. Der bewegte Vordergrund (im Beispiel der Tennisspieler) wird seperat frameweise als Video-Objekt übertragen. Der Empfänger kombiniert den Hinter- und Vordergrund zu einem darstellbaren Frame. Abbildung 5 illustriert die Vorgänge.

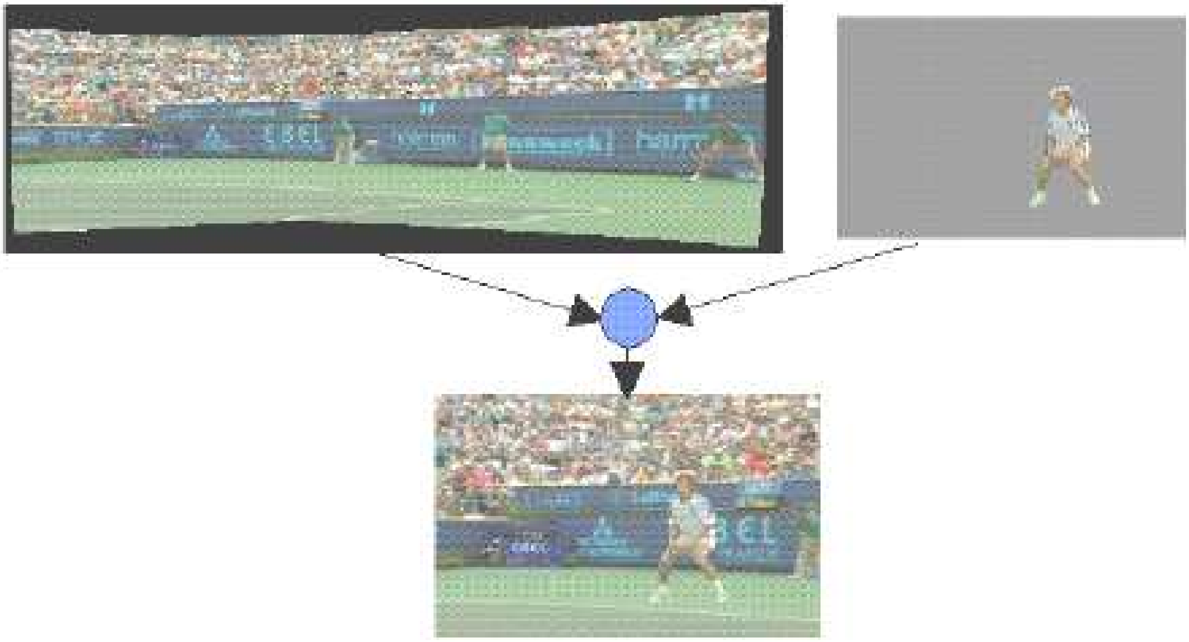

Abbildung 5: MPEG-4 - Sprite Coding aus [12]

2.4.2 DivX

DivX ist wohl der bekannteste Vertreter einer neuen Generation von Video-Codecs [13]. DivX Version 4 und 5 sind nicht zu verwechseln mit Divx3 oder DivX:-). Letzterer ist ein illegal gehackter und verbreiteter Microsoft Codec. Die Versionen 4 und 5 basieren nicht auf diesem, sondern auf dem MPEG-4 Standard und wurden von der Firma DivX Networks, Inc. entwickelt. Wie Tabelle 2 zeigt, ermöglicht DivX eine sehr viel bessere Kompression, als MPEG-2 Codecs bei vergleichbarer Qualität. Der Einsatz von MPEG-4-basierenden Codecs ermöglicht erstmals Übertragung von Video-Daten in anschaulicher Qualität über das Internet. Die Erwähnung des Namen DivX bezieht sich in der gesamten Arbeit auf die Version 4 und 5.

2.4.3 XviD

XviD ist ein unter der *GNU General Public License* (*GPL*) lizenzierter ISO MPEG-4-konformer Video-Codec. Entwickler aus verschiedenen Ländern arbeiten an der Verbesserung dieses Codecs. Mittlerweile erreicht XviD eine mit DivX vergleichbare Qualität, Geschwindigkeit und Kompression. Die aktive Entwicklung läßt auf weitere Verbesserungen hoffen.

2.4.4 RT-XviD

Real-Time-XviD ist ein Projekt der Universität Friedrich-Alexander in Erlangen-Nürnberg mit dem Ziel, den XviD-Codec echtzeitfähig zu machen [15]. Zur Zeit ist lediglich die Aufgabenstellung ausgeschrieben, jedoch noch nichts Praxisrelevantes umgesetzt worden.

2.5 Vorhandene Software

2.5.1 Transcode

Transcode ist ein konsolenbasiertes Werkzeug zur Videoverarbeitung unter Linux bzw. Unix. Es wurde von Dr. Thomas Östreich entwickelt und wird von einer recht großen Gemeinschaft gepflegt und weiterentwickelt [16].

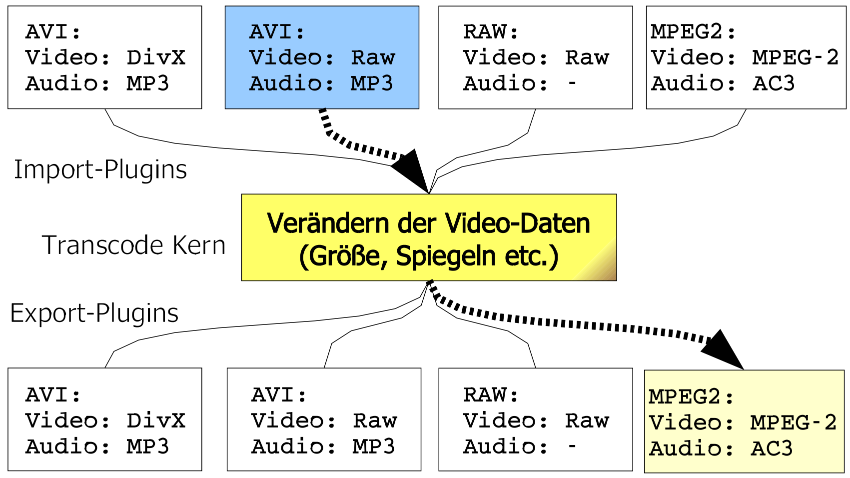

Abbildung 6: Transcode - Struktur

Hauptsächlich dient Transcode der Konvertierung von Videos in andere Ausgangsformate. Es ist unter der GPL freigegeben. Das Vorliegen des Quelltextes ermöglicht genauere Einblicke in Transcode. So arbeitet Transcode mit Plugins, welche zum Import und Export der Videos dienen. Auf Filter-Plugins wird nicht näher eingegangen, da diese für die Funktionalität nicht wesentlich sind. Plugins sind als *shared libraries* realisiert. Der Nutzer bzw. die automatische Erkennung des Videos (tcprobe) ist für die Wahl des Plugins verantwortlich. Die Import-Plugins geben Bild- und Audiodaten im RAW-Format an den Transcode-Kern. Die Export-Plugins erwarten ihrerseits ebenfalls RAW-Daten, um diese ins gewählte Ausgangsformat zu enkodieren und in einen Container gepackt auszugeben. Der eigentliche Kern von Transcode dient dem Verändern der Video- bzw. Audiodaten. So sind Größenänderungen, verschiedene Filter, Spiegelungen und vieles mehr möglich. Die Abbildung 6 zeigt die Struktur von Transcode.

Diese Struktur ermöglicht eine recht einfache Anbindung verschiedener Plugins und somit auch von verschiedenen Codecs und deren Video- und Audio-Formate. Dies erklärt die hohe Anzahl der vorhandenen Plugins. Einen Auszug der zur Zeit verfügbaren Plugins zeigt Tabelle 5.

2.5.2 retavic und memo.REAL

Ziel des an der Universität Erlangen-Nürnberg entwickelten Projektes ist die Schaffung eines "*REal-Time-Audio-Video-Conversion*"-Tools (kurz *retavic*) für Multimedia Datenbankserver, die in einer Echtzeit-Umgebung arbeiten, um Quality-of-Service (QoS) Anwendungen gerecht zu werden. In [17] sind folgende Ziel beschrieben:

Plugin	kurze Beschreibung
avi	RGB/PCM aus RIFF AVI-Dateien
raw	RGB/PCM aus RIFF AVI/WAVE files oder RAW-Dateien
net	RAW-Audio und -Video von Netzwerk-Sockets lesen.
mov	Quicktime-Dateien (quicktime4linux)
avifile	vom Avifile-Bibliothek unterstützte Codecs und Dateiformate
ffmpeg	vom ffmpeg-Bibliothek unterstützte Codecs
divx	AVI-Dateien mit DivX, OpendivX, DivX 4.xx/5.xx, XviD (nutzt DivX Codec)
xvid	AVI-Dateien mit DivX, OpendivX, DivX 4.xx/5.xx, XviD (nutzt XviD Codec)
mpeg2	MPEG extrahiert und dekodiert von mpeg2dec
dv	Digital Video und PCM Audio aus RIFF AVI-Dateien oder RAW-Dateien
v4l	Grabben vom Video for Linux kompatiblen Geräten
ac3	AC3 Audio
mp3	MPEG Audio
ogg	Ogg Vorbis Audio

Tabelle 5: Transcode - Plugins

- Charakterisierung bereits existierender Multimedia-Datenbank-Server
- Klassifizierung und Vereinheitlichung der Beschreibung von Audio- und Videoformaten
- Charakterisierung vorhandener Konverter
- Entwicklung und Implementierung von Konvertern, welche QoS-Unterstützung bieten und in einer Echtzeit-Umgebung funktionieren
- Performance-Messungen der kompletten Echtzeit-Umgebung
- Interaktion mit clientseitigen Medien-Spielern

Ein ähnliches Projekt führt ebenfalls die Technische Universität Dresden durch. Es trägt den Namen *memo.REAL* [18], was die Kurzform für "Media object Encoding by Multiple Operations in REALtime" ist. Das Projektziel ist, einen Medien-Server zu konstruieren, der QoS- und Echtzeitbedingungen genügt.

Die beiden genannten Projekte arbeiten kooperativ zusammen. In Form von Beleg- und Diplomarbeiten werden derzeit verschiedene Aufgaben gleichzeitig bearbeitet; zum Beispiel wird demnächst im Rahmen von *memo.REAL* am Konvertermodell gearbeitet. Das Team von *retavic* kümmert sich dabei eher um die Erstellung von RealTime-Audio- und -Video-Konvertern. Diese Arbeiten befinden sich nach Kenntnissen des Autors jedoch noch im Planungsstadium.

2.5.3 Smart-MPEG

Das Smart-MPEG-Projekt ist ein sowohl unter Linux als auch unter DROPS laufender MPEG-1- und MPEG-2-Dekoder [19]. Smart-MPEG wurde am Lehrstuhl Betriebssysteme der TU-Dresden entwickelt. Ziel des Projektes ist die Wiedergabe von Audio und Video unter Echtzeitbedingungen und Unterstützung von Quality-of-Service-Anforderungen. Realisiert wurde dies dadurch, daß bei steigender Systemlast einzelne Frames ausgelassen werden. Die Qualität der Audio-Wiedergabe bleibt unverändert. Es wird dafür gesorgt, daß der Ton kontinuierlich ausgegeben werden kann, indem keine Audio-Daten verworfen werden [20]. Die Video-Frames werden in 3 Ströme zerlegt: je ein Strom für I-, P- und B-Frames. Bei zu hoher Last werden zuerst B-Frames ausgelassen, danach P-Frames und I-Frames. Die Regeln zum Verwerfen von Frames sind flexibel austauschbar, was es ermöglicht, eigene Strategien zur QoS-Unterstützung einzusetzen. Der komponentenweise Aufbau von Smart-MPEG, das Kapseln der Funktionalität in einer C++-Bibliothek und die Abstraktion *Mpeg_buffer* tragen zur Flexibilität bei.

3 Entwurf

In diesem Kapitel werden verschiedene Möglichkeiten einer Realisierung eines echtzeitfähigen Videorekorders dargestellt. Es ist bereits an dieser Stelle nötig, die Projektziele zu analysieren und zu berücksichtigen, um die bei diesem Beleg geforderte Implementierung des Testprogramms bereits in Richtung Videorekorder, Echtzeitfähigkeit und Wiederverwendbarkeit voranzutreiben. Daher wird in diesem Kapitel das gesamte Projektziel behandelt und nicht nur die Portierung eines Codecs sowie dessen Testprogramm.

3.1 Portierung von Transcode

Eine naheliegende Alternative wäre die Portierung von Transcode auf DROPS. Die Vorteile der bereits zahlreich vorhanden Plugins wären immens. Es wäre so möglich, eine große Anzahl von Codecs unter DROPS nutzbar zu machen, sowie ein bereits durch sehr viele Benutzer getestetes und ausgereiftes Programm zur Verfügung zu stellen.

Leider ist dieser Ansatz unpraktikabel. Um dies zu erläutern bedarf es einer genaueren Untersuchung der Transcode-Funktionsweise. Die Vorgänge beim Bearbeiten eines Videos werden hier kurz und unvollständig in einem Pseudo-Code wiedergegeben. Die vorhandenen "||" zeigen eine Nebenläufigkeit dieser Programmzweige in Form von Threads an.

```
   // Transcode analysiert import-Dateien um zu wissen, welche Plugins nötig sind
   // Ausnahme: Parameter dafür wurden übergeben
   probe_Files();
   load_plugIns();
   import_Start(); // starte Import-Plugins mit Kommando zum Datei öffnen (bzw. einer
Pipe zu externen Programmen)
   export_Init();  // Export Plugins initialisieren
   export_Open();  // export Plugin öffnet Ausgabe Datei(en)
   //" Hauptschleife"
   while (nicht terminieren)
   {
       // Video Verarbeitung ist parallel (||) zur Audio-Verarbeitung. Import- und Ex-
port sind ebenfalls parallel
       video_work()
       {
           video_import()
           {
               if (dateiende) terminieren = true;
               transcode_video-processing();
           }
           ||
           video_export()  // gebe nötige Parameter an Export-Plugin mit
           {
              export_Frame_Init();
               export_Encode();
               if(Fehler beim Kodieren) terminieren = true;
```

```
            }
        }
        ||
        audio_work()
        {
            audio_import()
            {
                if (datei ende) terminieren = true;
                transcode_Audio-processing();
            }
            ||
            audio_export()  // gebe nötige Parameter an Export-Plugin mit
            {
                export_Frame_Init();
                export_Encode();
                if(Fehler beim Kodieren) terminieren = true;
            }
        }
    }
    // fertig mit Kodieren; alles schliessen und Ressourcen freigeben
    export_Close();
    import_Close();
```

Die Funktionsweise entspricht der üblichen Form: Daten holen → Dekodieren → Bearbeiten → Kodieren → Daten ausgeben. Jedoch laufen diese Prozesse parallel für Audio- und Video ab, sowie parallel in dem Sinne, daß Import, Bearbeitung und Export ebenfalls nebenläufig sind. Dies ist natürlich der Performance zuträglich, behindert jedoch nach der Portierung auf DROPS die Impementierung eines eigenen Scheduling-Schemas.

Bei der Untersuchung der Plugins war festzustellen, daß diese für Ein- und Ausgabeoperationen zuständig sind. In manchen Plugins war auffällig, daß diese externe Programme starten und die Daten via Pipe an Transcode übergeben werden. Dies ist eine elegante und schnelle Lösung zum Erstellen der Plugins, jedoch gegenwärtig unter DROPS nicht praktikabel, da I/O-Funktionen mangels verfügbarem Dateisystem nicht nutzbar sind. In Form einer Pipe ist das Nutzen externer Programme ebenfalls nicht möglich.

Erwähnenswert ist die Tatsache, daß das Export-Plugin dafür sorgt, daß Audio und Video wieder synchronisiert werden, falls dies erforderlich ist. Transcode arbeitet sowohl bei Video- als auch bei Audio-Daten mit Frames. Diese Frames werden mit einer eindeutigen ID versehen, um zu gewährleisten, daß das Export-Plugin die richtige Reihenfolge beim Synchronisieren und nachfolgendem Speichern bestimmen kann.

Zusammenfassend läßt sich feststellen, daß folgende Eigenschaften eine Umsetzung auf DROPS unter Berücksichtigung der Echtzeitfähigkeit ver- und behindern:

- Es gibt bisher keine Arbeiten an Transcode in Richtung Echtzeitfähigkeit.
- Transcode besteht aus mehreren Threads, die es schwierig machen, ein eigenes Scheduling-Schema zu implementieren.

- Die Plugins nutzen Datei-I/O-Funktionen.
- Die Plugins nutzen teilweise externe Programme.

Dagegen erscheint es möglich, Teile von Transcode zu verwenden, beispielsweise Funktionen zur Größenänderung und Farbraumkonvertierung. Diese sind von den genannten Problemen nicht betroffen und somit einfach portierbar.

3.2 Erweiterung vorhandener Codecs um Echtzeitfähigkeit

Am Beispiel des XviD-Codecs wird gezeigt, daß es im Rahmen dieser Projektarbeit für eine Person zu komplex ist, in einen vorhanden Codec nachträglich Echtzeitfähigkeit einzubauen.

Momentan ist der XviD-Codec in keinster Weise auf den Echtzeiteinsatz ausgerichtet [21]. Die einfachste Möglichkeit voraussagbare Zeiten für das Enkodieren- bzw. Dekodieren zu bekommen, wäre die Worst-Case-Zeiten in die Planung des Systems einzubeziehen. Es wäre aber nicht sinnvoll, da die Entwickler den Codec auf häufig vorkommende Bilder optimiert haben. Im Regelfall ist also die Kodierzeit wesentlich kürzer als im Worst-Case - laut Entwicklern ist der Faktor 100 durchaus realistisch. Daher würde diese Lösung zu hohe Leistungsansprüche an die Hardware stellen und diese zum Großteil der Zeit nicht ausnutzen. Eine Messung der Worst-Case-Zeiten würde sich ebenfalls sehr schwierig gestalten, da unterschiedliche Teile des Codecs bei unterschiedlichen Bildern bzw. Bildfolgen ihre Worst-Case-Zeit haben. So zum Beispiel wäre ein Rauschen als Bild nahe dem Worst-Case der DCT, aber für die Motion Estimation wäre es eine ungerichtete Bewegung der Kamera mit gleichzeitigen Zoomen sowie mehrere Bewegungen in andere Richtungen. Ein Beispiel für letzteres, wäre ein Zoom weg von einem applaudierenden Publikum.

Zusammenfassend sei bemerkt, daß es sehr schwierig ist, den XviD-Codec echtzeitfähig zu machen. Folgende Punkte sprechen dagegen:

- Es gibt bisher keine Arbeiten innerhalb des Codecs in Richtung Echtzeit.
- Laut XviD-Entwicklern ist es kompliziert, Echtzeitfähigkeit nachträglich zu implementieren.
- Es scheint unsinnig, Worst-Case-Zeiten zu nutzen. Davon abgesehen wird es schwierig, diese zu bestimmen.
- Der Detailgrad des MPEG-4 Standards sowie die internen Details von XviD sind zu komplex, um im Rahmen dieser Projektarbeit betrachtet zu werden.

Eine Portierung auf L4 ohne Berücksichtigung der Echtzeitfähigkeit scheint dagegen - dank ANSI C und dem Verzicht auf Datei-I/O - trivial.

3.3 Existierende, nicht echtzeitfähige Codecs als Basis des Videorekorders

Es wurde dargestellt, daß es nicht ohne weiteres möglich ist, einen vorhandenen Codec um Echtzeitfähigkeit zu erweitern. Was bleibt, ist die Möglichkeit, eine Architektur um einen bestehenden Codec herum zu entwickeln, die so gut wie möglich dem Projektziel entgegenkommt. Im vorherigen Abschnitt wurde geschildert, daß die Planung des Systems auf Basis von Worst-Case-Zeiten nicht sinnvoll ist. Daher muß ein anderer Ansatz gewählt werden, um ein System zu erstellen, daß hohe Leistung mit voraussagbarem Verhalten kombiniert.

Beim einem Videorekorder sollen Bilder beispielsweise von TV-Karten, Webcams o.ä. importiert, dann enkodiert und in einer Datei gespeichert werden. Bei der Wiedergabe von Videos wird dann aus einer aufgezeichneten Datei gelesen, Bild und Ton dekodiert und anschließend auf dem Monitor dargestellt bzw. von der Soundkarte ausgegeben. Die Verarbeitungsschritte sind dabei annähernd gleich: zuerst der Import von der Quelle, danach das Enkodieren bzw. Dekodieren in der Verarbeitungskomponente (*Kern*) und als letztes der Export der Ton- bzw. Bild-Daten zur Senke. Dieses vereinfachte Modell läßt sich sowohl als Basis für das Enkodieren und Dekodieren als auch für Audio- und Video-Kodierung nutzen. Deshalb steht das Wort *Kodieren* im folgenden ersatzweise für das Dekodieren und Enkodieren. Ebenfalls wird allgemein von *Frames* gesprochen, die sowohl Einzelbilder als auch Audiostücke sein könnten.

Als Schnittstellen zwischen den einzelnen Teilbereichen soll DSI dienen, da es die Möglichkeit bietet, Daten mit anderen Komponenten (z.B. Hardware-Treibern) schnell und komfortabel auszutauschen. Es können jedoch auch andere Mechanismen, wie z.B. shared memory oder IPCs benutzt werden. Um auftretende Schwankungen der Verarbeitungszeit zu kompensieren, wird ein Puffer verwendet. Eine wichtige Charakterisierungsgröße des Puffers ist sein Füllstand. Diese Information ist nötig, um einen Überlauf und somit dem Verlust von Bild- bzw. Ton-Informationen nach Möglichkeit zu vermeiden. Im Falle von DSI ist dies beispielsweise das Verhältnis von den benutzten Paketen und der Gesamtzahl verfügbarer Pakete. Durch eine kleine Erweiterung an DSI wäre diese Größe schnell bestimmbar. Die Verarbeitungskomponente nimmt einzelne Frames aus dem Puffer und bearbeitet diese. Die kodierten Daten werden dann via DSI o.ä. an die Export-Komponente, beispielsweise ein Dateisystem, einen Displaytreiber (CON, DOpE) oder sonstige, weitergegeben.

Bei aktuellen Codecs ist das Einstellen der gewünschten visuellen bzw. akustischen Qualtität beim Enkodieren und Dekodieren möglich. Für den XviD-Codec wurden dazu Messungen unter Linux durchgeführt. Dabei wurden die in den Tabellen 6 und 7 dargestellten Parametern zur Steuerung von XviD verwendet. Eine Kombination dieser Parameter wurde genutzt, um sieben Qualitätsstufen zu definieren. Tabelle 8 zeigt diese. Die genaue Bedeutung der einzelnen Parameter konnte aufgrund fehlender Dokumentation nicht eindeutig geklärt werden. Es ist jedoch gesichert, daß eine höhere Qualtitätsstufe zur Verbesserung der visuellen Qualtiät führt.

General-Flag	Erklärung aus XviD-Dokumentation
XVID_H263QUANT	informs xvid to use H263 quantization algorithm
XVID_MPEGQUANT	informs xvid to use MPEG quantization algorithm.
XVID_HALFPEL	informs xvid to perform a half pixel motion estimation
XVID_INTER4V	forces XviD to search a vector for each 8x8 block within the 16x16 Macro Block

Tabelle 6: XviD-Codec - Erklärung der General-Flags aus [22]

Die Ergebnisse der Messung der Enkodierzeit in Abhängigkeit zu den Qualitätsstufen zeigt Abbil-

Motion-Flag	Erklärung aus XviD-Documentation
PMV_EARLYSTOP16	PMVfast and EPZS stop search if current best is below some dynamic threshhold. No diamond search is done, only halfpel refinement (if active). Without EARLYSTOP diamond search is always done. That would be much slower, but not really lead to better quality.
PMV_HALFPELREFINE16	After normal diamond search, an extra halfpel refinement step is performed. Should always be used if XVID_HALFPEL is on, because it gives a rather big increase inquality.
PMV_EXTSEARCH16	Normal PMVfast predicts one start vector and does diamond search around this position.
	EXTSEARCH means that 2 more start vectors are used: (0,0) and
	median predictor and diamond search is done for those, too. Makes search slightly slower, but quality sometimes gets better.
PMV_USESQUARES16	Replace the diamond search with a square search.
PMV_EARLYSTOP8 PMV_HALFPELREFINE8	They have the same meaning as their 16x16 counter part

Tabelle 7: XviD-Codec - Erklärung der Motion-Flags aus [22]

Qualitätsstufe	Motion-Flags	General-Flag
0	-	XVID_H263QUANT
1	PMV_EARLYSTOP16	XVID_MPEGQUANT
2	PMV_EARLYSTOP16	XVID_H263QUANT |
3	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16,	XVID_H263QUANT | XVID_HALFPEL
4	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16,	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V
5	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16 | PMV_EARLYSTOP8 | PMV_HALFPELREFINE8	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V
6	PMV_EARLYSTOP16 | PMV_HALFPELREFINE16 | PMV_EXTSEARCH16 | PMV_USESQUARES16 | PMV_EARLYSTOP8 | PMV_HALFPELREFINE8	XVID_H263QUANT | XVID_HALFPEL | XVID_INTER4V

Tabelle 8: XviD-Codec - Definition der Qualitätsstufen aus [22]

dung 7, während Bild 8 die Abhängigkeit zur *Bitrate* darstellt. Weitere Details zu den durchgeführten Messungen sind in [22] zu finden. Wie diese Messungen bei Betrachtung der Qualitätsstufen 2-6 zeigen, sind Einsparungen der Rechenzeit von bis zu ca. 60% möglich. Das Reduzieren der Bitrate spart etwa 15 Prozent der Verarbeitungszeit. Natürlich sind die genauen Werte abhängig vom Bildmaterial, insbesondere der Auflösung. Aber generell läßt sich feststellen, daß eine höhere Bitrate und höhere Qualitätsstufe zu längerer Rechenzeit führen. Im folgenden wird die Abhängigkeit von Verarbeitungszeit und visueller oder akustischer Qualität allgemein als *Rechenzeit-Qualitäts-Funktion* bezeichnet.

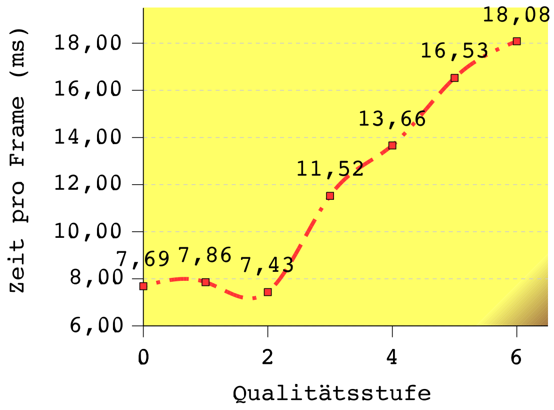

Abbildung 7: Qualitätsstufen - Zeit pro Frame

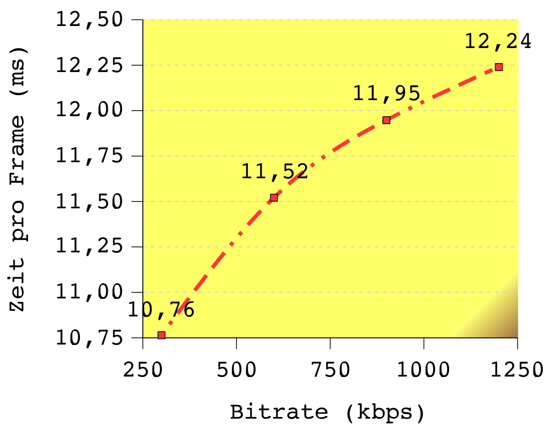

Abbildung 8: Bitrate - Zeit pro Frame

Die durch die Messungen gewonnenen Erkenntnisse sowie die Verwendung von DSI in Kombination mit einem Puffer lassen sich zu der in Abbildung 9 dargestellten Infrastruktur kombinieren.

Die Besonderheit dieser Infrastruktur liegt in der dynamischen Anpassung der Kodierzeiten abhängig vom Füllstand des Puffers. Ist der Puffer nahezu voll, wird die Kodierzeit auf ein Minimum reduziert; ist der Puffer jedoch fast leer, könnte eine längere Kodierzeit ermöglicht werden, ohne daß Frames verloren gehen würden, also *gedropt* werden müßten. Die Zeitvariation zwischen einem vollen und einem leeren

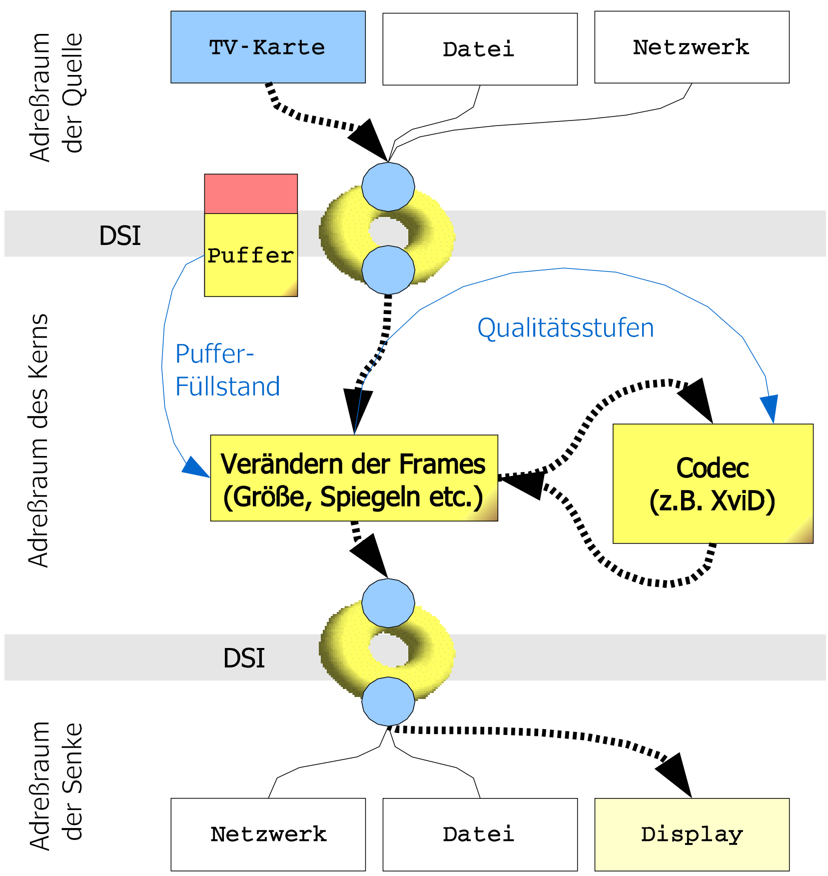

Abbildung 9: Mögliche Infrastruktur auf Basis nicht echtzeitfähiger Codecs

Puffer schlägt sich in der visuellen oder akustischen Qualität des kodierten Materials nieder. So ist es im Falle der Nutzung des XviD-Codecs möglich, die Qualtitätsstufen 2-6 zur Anpassung der Rechenzeit zu nutzen; dies wäre dann die spezifische Rechenzeit-Qualitäts-Funktion des XviD-Codecs. Die Änderung der Bitrate dagegen läßt sich nicht nutzen, da eine Re-Initialisierung des Codecs nötig ist, um die Bitrate zu verändern. Dadurch würde sich die Enkodierzeit für das Frame, bei dem die Bitraten-Änderung stattfindet, um das zwanzigfache der üblichen Zeit verlängern.

Eine zusätzliche Optimierung bezüglich der Arbeitsgeschwindigkeit ist, daß die Entscheidung über notwendige Framedrops (bei vollem Puffer) getroffen wird, bevor die Frames überhaupt in den Puffer gelangen. Daher ist es für die Verarbeitungskomponente nicht nötig, bereits teilweise oder vollständig kodierte Frames zu verwerfen und so Kodierzeit zu verschwenden. Ein weiterer Vorteil ist, daß es im Falle des XviD-Codecs eine Re-Initialisierung benötigen würde, wenn bereits teilweise kodierte Frames gedroppt werden sollen. Die chronologische Ordnung der Frames wird durch eine Frame-ID gewährleistet, die bereits von der Import-Komponente generiert wird. Bei fehlenden ID-Nummern kann so der Rückschluß gezogen werden, wann und wieviele Frames verworfen worden sind. So ist es beispielsweise möglich, nachträglich leere Frames einzufügen, um die Synchronisation zwischen Bild und Ton gewährleisten zu können.

Da die konkrete Verarbeitungszeit abhängig vom Video- bzw. Audio-Material ist und von Frame zu Frame unterschiedlich sein kann, sollte eine durchschnittliche Bearbeitungszeit für das Scheduling eingeplant werden. Diese kann z.B. während der Laufzeit durch Heuristiken bestimmt und verändert oder auf Grund von Erfahrungen bzw. Messungen konstant gewählt werden. Idealerweise würde die Zeit pro Frame so gewählt werden, daß der Puffer weder voll noch leer würde. Problematisch ist der Fall, daß die Verarbeitungszeit länger oder kürzer dauert, als eingeplant. Dies hat bei der gewählten Infrastruktur zur Folge, daß folgende Frames später bzw. früher bearbeitet werden. Das wiederum führt zu Schwankungen der Verweildauer eines Frames im Kern, die u.a. zu Synchronisationsproblemen zwischen Ton und Bild führen können. Diese Eigenschaft ist für das Aufzeichnen des Videos unproblematisch, da es nicht darauf ankommt, wann das Frames gespeichert wird, sondern daß es überhaupt gespeichert wird. Beim Dekodieren hingegen können die unterschiedlichen Verweildauern jedoch zu einer Art "Gummiband-Transformation" bei der Wiedergabe führen. Dieser Effekt kann aber mit einem Puffer auf der Export-Seite weitgehend vermieden werden. Ein weiteres Problem beim Dekodieren ist jedoch, daß teilweise Abhängigkeiten unter den Frames existieren können und zu berücksichtigen sind. Dies ist unter anderem dann der Fall, wenn sich ein P-Frame auf das vorhergehende I-Frame bezieht. Die angesprochenen Probleme zum Scheduling gilt es im Rahmen des Projektzieles in den folgenden Arbeiten zu diskutieren und zu lösen.

Zusammengefaßt hat die oben vorgeschlagene Infrastruktur neben den genannten Vorteilen auch folgende Nachteile:

- Es gibt möglicherweise Schwierigkeiten bei der Bestimmung der einzuplanenden Zeit.
- Mögliche Schwankungen der Verweildauer pro Frame im Kern müssen berücksichtigt werden.
- Es gibt keine Garantie, daß Framedrops nicht auftreten. Es sei denn, die eingeplante Zeit wird größer oder gleich der Worst-Cast-Zeit gewählt.

- Beim Dekodieren sind auch Abhängigkeiten der Frames untereinander zu berücksichtigen.

Demgegenüber stehen folgende Vorteile:

- Teilweise kodierte Frames müssen nicht abgebrochen werden und somit wird keine Rechenzeit vergeudet.
- Frames, die in den Puffer gelangen, werden zu Ende kodiert. So ist sichergestellt, daß irgendwann mindestens ein Frame fertig kodiert wird.
- Es gibt eine große Anzahl möglicher nutzbarer Codecs, die nur folgenden Anforderungen genügen müssen:
  - Die für das Scheduling einzuplanende Zeit muß bestimmbar sein, sei es durch Messungen, Schätzungen oder sonstige theoretische Überlegungen.
  - Die Bearbeitungszeit der Rechenzeit-Qualitäts-Funktion muß für die gewählten Qualitätsstufen monoton wachsend sein.
  - Der Codec muß frameweise arbeiten können.
- Das Funktionsprinzip ist auf Audio- und Video-Kodierung anwendbar.
- Durch die Wahl der einzuplanenden Zeit im Bereich von größer null bis einschließlich Worst-Case-Zeit, läßt sich die gewünschte Anzahl der kodierten Frames festlegen, die wahrscheinlich (abhängig von der realen Verarbeitungszeit) bei der Export-Komponente ankommen. Dies ließe sich als Quality-of-Service-Eigenschaft nutzen.

3.4 Schlussfolgerung

Der Vorschlag, einen bestehenden Codec mit einer im Abschnitt 3.3 beschriebenen Infrastruktur zu kombinieren und als Basis für einen Videorekorder zu nutzen, scheint der vielversprechenste Weg. Die gewählte Infrastruktur ermöglicht eine hohe Ausnutzung der Rechenleistung unter Berücksichtigung der geforderten Wiederverwendbarkeit durch Nutzung des DROPS Streaming Interfaces. Weiterführend ermöglicht dieser Ansatz, weiche Echtzeitfähigkeit in Verbindung mit Quality-of-Service zu realisieren.

Um harte Echtzeit zu erreichen, bedarf es jedoch der Bestimmung der Worst-Case-Zeiten und der Nutzung dieser als für das Scheduling einzuplanenden Zeit. Die Entwicklung des RT-XviD an der Universität Erlangen-Nürnberg oder anderer RT-Codecs werden nach jetzigem Kenntnisstand einfach in die gewählte Infrastruktur einzubauen sein, was es ermöglicht, ein hartes Echtzeitsystem zu implementieren.

Im folgenden Kapitel wird die Implementation des Testprogramms in Anlehnung an diesen Vorschlag vorgestellt und um ein Plugin-Konzept ähnlich dem von Transcode erweitert.

4 Implementierung

Die im folgenden gegebenen Informationen beziehen sich, im Gegensatz zu Kapitel 3 "Entwurf", auf die für den Großen Beleg relevanten Aufgaben: die Portierung eines Codecs, sowie die Erstellung des Testprogramms. Auf das Projektziel Videorekorder wird in diesem Kapitel nicht näher eingegangen.

4.1 Portierung des XviD-Codecs

Die Wahl des im Rahmen dieses Belegs zu portierenden Codecs fiel auf den XviD-Codec, da er folgende Vorzüge mitbringt:

- moderner MPEG-4-basierter Codec
- gute Bildqualität bei niedriger Bitrate
- ANSI C
- kein Datei-I/O

Durch Umstellung der Makefiles konnte der Codec zügig für DROPS portiert werden. Durch das Einbinden des übertragenen Codecs in das Testprogramm konnte anschließend ein einfaches Standbild bzw. eine kurze Video-Sequenz enkodiert und dekodiert werden.

Nachdem das Testprogramm erweitert wurde, war es möglich, komplette XviD-Videos, welche mit Transcode unter Linux erstellt wurden, abzuspielen. Nach etwas genauerer Recherche fiel auf, daß es jetzt eigentlich möglich sein sollte, auch mit DivX(4/5) kodierte Videos wiedergeben zu können. Dies funktionierte jedoch nicht.

Durch Zufall wurde die Ursache dafür gefunden: auf einem anderen System kodierte XviD-Videos konnte das Testprogramm nicht dekodieren. Eine genauere Analyse der aktualisierten und neu übersetzten XviD-Bibliothek ergab einen sehr auffälligen Unterschied in der Dateigröße zur portierten Bibliothek. Der entscheidende Fehler war, daß beim Kompilieren des XviD-Codecs für DROPS in sämtlichen Verzeichnissen object-Dateien erstellt wurden. Ein anschließender Make-Durchlauf mit Linux-x86 als Zielplattform hat nur diese Dateien zusammengefaßt und in ein Archiv gepackt bzw. zu einer Bibliothek gelinkt. Diese wurde dann von Transcode für das Erstellen der Testvideos genutzt. Da sowohl im Testprogramm unter DROPS als auch unter Linux identische Kompilate genutzt wurden, war der Grund für die Probleme bei der Wiedergabe von auf anderen Systemen erstellter Videos gefunden.

Der für DROPS kompilierte Codec funktioniert auch unter Linux. Ein einfacher Test zeigte, daß auch der für Linux-x86 kompilierte XviD-Codec unter DROPS funktioniert. Ein Linken des durch den Linux-Make-Lauf erstellten Codecs in das Testprogramm führte dazu, daß XivD- und DivX-Videos ohne Probleme wiedergegeben werden konnten. Bei weiteren Versuchen gelang es, einen mit dem Intel C-Compiler[2] kompilierten XviD-Codec unter DROPS testweise zu nutzen.

Die einfache Lösung der Nutzung des Linux-Binaries bringt zusätzliche Vorteile mit sich:

- alle unter Linux verfügbaren Compiler-Optimierungen sind nutzbar

[2] Version 6.0, Copyright (C) 1985-2002, Intel Corporation.

- Intel C-Compiler Kompilate sind nutzbar
- Zeiteinsparung für die Portierung durch Nutzung vorhandener Makefiles und Verzicht auf Erstellung eines DROPS-spezifischen Makefiles

Mögliche Nachteile sind eventuelle Probleme bei der Übertragung auf andere Architekturen wie PPC oder IA64. Auch darf an dieser Stelle nicht auf Allgemeingültigkeit dieser einfachen Möglichkeit zur Übertragung von Codecs auf DROPS geschlossen werden. Es ist auf jeden Fall sicherzustellen, daß keinerlei Datei-I/O vom Codec genutzt wird, sowie daß möglichst nur Funktionen der für DROPS vorhandenen C-Bibliothek genutzt werden.

Um zu zeigen, daß diese Methode bei der Portierung anderer Codecs auch funktioniert, wurde testweise der Vorbis-Audio-Codec des als möglichen Nachfolger für MP3 gehandelten OGG/Vorbis ebenfalls auf die DROPS-Plattform übertragen.

4.2 Entwicklung der Testumgebung

Um eine möglichst vielseitig nutzbare Architektur des Programmes zu erstellen, wurde eine Plugin-Architektur ähnlich der von Transcode realisiert. Wie bereits in Kapitel 3.1 erwähnt, nutzen die Transcode-Plugins teilweise externe Programme sowie Datei-I/O-Operationen. Oft wird auch direkt durch die Bibliothek des Codecs bereits aus Dateien gelesen bzw. in Dateien geschrieben. Eine Abbildung dieses Konzeptes ist gegenwärtig unter DROPS sehr schwierig, da momentan kein Dateisystem zur Verfügung steht. Als Lösung dieses Problems wurde ein etwas anderes Konzept umgesetzt. Im Gegensatz zu Transcode, wo nur Import- und Export-Plugins genutzt werden, wurde eine Realisierung durch vier Plugin-Schnittstellen gewählt. Neben den *Import-* und *Export-Plugins*, die nur reine Eingabe- und Ausgabefunktionen kapseln und keine Video- oder Audio-Verarbeitung durchführen, existieren noch *Codec-Plugins* und *Controller-Plugins*. Der Sinn dieser Aufteilung liegt einerseits darin, die gegenwärtig recht komplizierte Beschaffung der Daten ohne verfügbares Dateisystem zu bewältigen, als auch in der Fähigkeit, durch andere verwendete Plugins eine komplett andere Funktionalität, z.B. Audio-Kompression, nutzen zu können. Das Auslagern der Interaktion mit dem Benutzer in Controller-Plugins ermöglicht einen einfachen Austausch des Benutzerinterfaces. So sind sowohl DROPSCON-, DOpE- oder textbasierte Interfaces denkbar. Die Abbildung 10 zeigt die Struktur des implementierten Testprogramms.

Import- und Export-Plugins: Momentan ist es nötig, die Video- oder Audio-Daten durch *Grub* in den Hauptspeicher laden zu lassen und entweder direkt auf die Daten oder über das durch DSI angebundene *dsi_staticfs* zuzugreifen. Im übrigen wurden zu Testzwecken auch Video- bzw. Bilddaten direkt in den Quellcode einkompiliert. Die Daten werden dem Kern im kodierten oder unkodierten Format frameweise übergeben. Dies geschieht unabhängig von Grub bzw. dsi_staticfs und ebenfalls unabhängig vom verwendeten Container (z.B. AVI). Die einzelnen Frames werden dann vom Kern und/oder Codec verarbeitet.

Anschließend werden die Frames an das Export-Modul übergeben. Momentan steht als Export-Plugin für alle Formate nur ein Dummy-Plugin zur Verfügung, welches die Daten entgegennimmt und

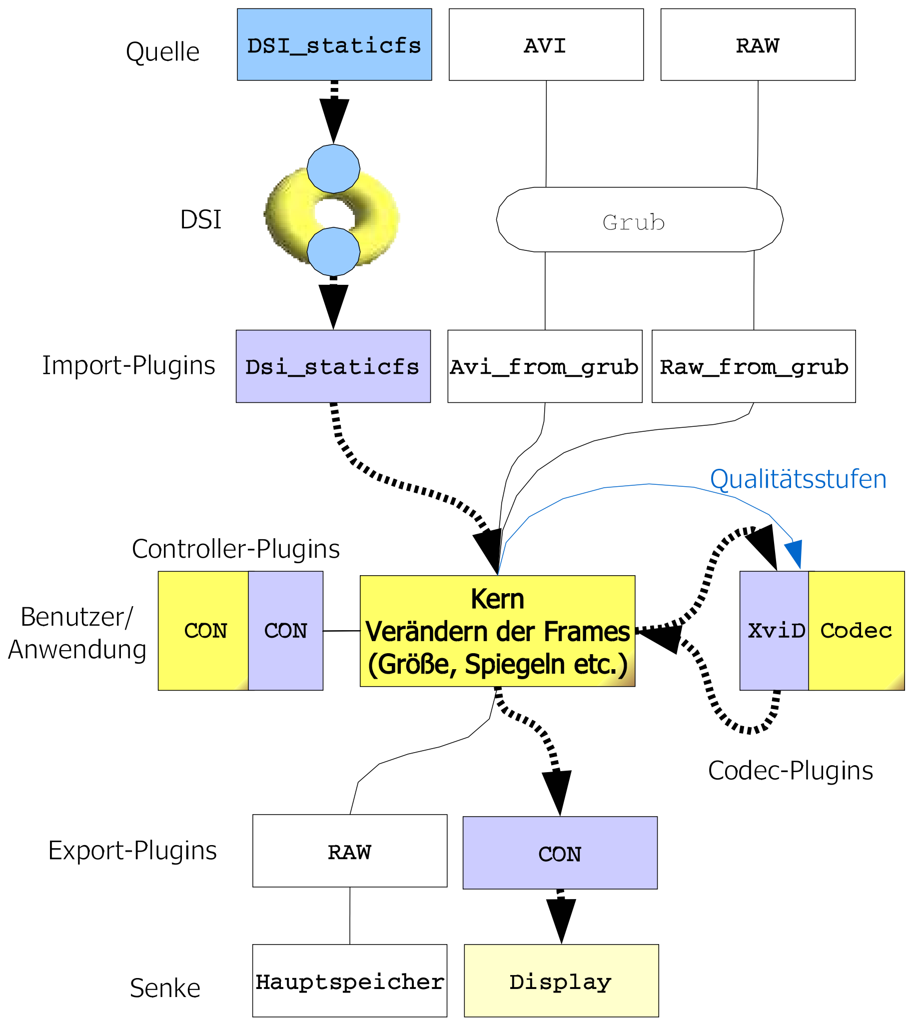

Abbildung 10: Testprogramm - Struktur

verwirft. Für das RAW-Datenformat steht zusätzlich ein Modul bereit, welches die Video-Daten auf die DROPSCON exportiert und darstellt.

Die Schnittstellen der Import- und Export-Module sind identisch. Es stehen Funktionen für das Initialisieren (*init*) und das Schließen (*close*) bereit. Für den eigentlichen Austausch der Daten stehen die Funktionen *step* und *commit* bereit, welche sehr stark an DSI angelehnt sind. So gibt die Funktion step eine Adresse zurück, die bei der Verarbeitung genutzt wird. Ist die Verarbeitung des Frames abgeschlossen, wird dem Import- bzw. Export-Plugin durch den Aufruf des entsprechenden commits mitgeteilt, daß die Daten nicht mehr benötigt werden und verworfen bzw. exportiert werden können. Die Anlehnung an DSI ermöglicht eine einfache Anbindung an DSI-Komponenten, z.B. an *dsi_staticfs*.

Controller-Plugins: Das Controller-Plugin stellt neben den obligatorischen Funktionen *init* und *close* auch Funktionen zur Interaktion mit dem Benutzer oder einer Anwendung bereit. Die Funktion *ctrl_show_text* dient beispielsweise dazu, Textnachrichten auszugeben, während *ctrl_show_original* und *ctrl_show_preview* dazu dienen, RAW-Bilder bzw. erst enkodierte und dann wieder dekodierte Frames als ”Preview” darzustellen. Noch nicht realisiert sind Funktionen, die die Funktionsweise des Kerns während der Verarbeitung beeinflussen. Angedacht und vorbereitet sind Funktionen wie *stop* und *start*, die später für die ”Videorekorder-Tasten” genutzt werden können. Für weitere Arbeiten wären einige Ergänzungen, wie z.B. für das visuelle Darstellen von Audio-Signalen, denkbar.

Codec-Plugins und Kern: Als Codec-Plugin existiert beispielsweise das XviD-Plugin (*codec_xvid*). Die Funktionen dieses Plugins werden durch die typischen Schnittstellenfunktionen vom Kern genutzt: Während *codec_init* den Codec initialisiert, also z.B. Video-Größe und Farbformat festlegt, dient *codec_close* dem Freigeben der vom Codec benötigten Ressourcen. Die Funktion *codec_step* übergibt den XviD-Codec einen Speicherbereich, der abhängig vom Import-Plugin und den Videodaten entweder kodierte oder unkodierte Frames enthält. Anschließend werden diese dann dekodiert bzw. enkodiert. Diese verarbeiteten Daten werden in die an *codec_step* übergebene Speicher-Region geschrieben und vom Kern weiterverarbeitet bzw. vom Kern dem Export-Modul übergeben. Erwähnenswert sind Funktionen zur Veränderung des Video-Materials, hier *Processing* genannt. Namentlich handelet es sich dabei um *resize*, *flip*, *mirror* und *swap* (Tabelle 9). Diese sind als Teil des Kerns verfügbar.

Funktion	kurze Beschreibung
resize	ein einfaches Ändern der Bildgröße (halbieren, dritteln, vierteln , ...)
flip	Bild auf den Kopf stellen
mirror	Bild spiegeln
swap	Farbwerte tauschen RGB $\Longleftrightarrow$ BGR

Tabelle 9: Testprogramm - Processing

Eine Übersicht der zur Zeit vorhanden Plugins zeigt Tabelle 11, während die in der gegenwärtigen Implementierung vorhandenen Funktionen der Plugin-Schnittstellen in Tabelle 10 zu finden sind.

Implementationsprobleme: Ein grundlegendes Problem ist der Mangel an Mathematik-Funktionen in der DROPS-Umgebung. Dies betrifft beispielsweise die Funktionen *log, ceil, cos, tan, exp* uvm. Zum

Funktion	kurze Beschreibung
import_init	Import-Plugin initialisieren
import_step	Adresse des Frames vom Import-Plugin holen
import_commit	Frames des Import-Plugin werden nicht mehr benötigt
import_close	Import-Plugin schließen
export_init	Export-Plugin initialisieren
export_step	Zieladresse für nächstes Frame holen
export_commit	Frame ist fertig und an Export übergeben
export_close	Export-Plugin schließen
codec_init	Codec-Plugin initialisieren
codec_step	Enkodieren bzw. Dekodieren
codec_close	Codec-Plugin schließen
ctrl_init	Controller-Plugin initialisieren
ctrl_show_preview	Preview-Bild anzeigen (erste kodiertes und dann dekodiertes Bild anzeigen, ermöglicht visuelle Qualität abzuschätzen)
ctrl_show_original	Original anzeigen (zeigt RAW-Bild, wie es direkt vom Import-Plugin kommt)
ctrl_show_text	Text ausgeben
ctrl_close	Controller-Plugin schließen

Tabelle 10: Testprogramm - momentan verfügbare Plugin-Schnittstellen

Plugin-Name	kurze Beschreibung
import_dsi_staticfs	holt RAW-Daten frameweise vom DSI_staticfs
import_avi_from_grub	holt Frames aus AVI-Container, der vom Grub übergeben wurde
import_avi_from_ram	holt Frames aus AVI-Container, der einkompiliert wurde
import_raw_from_grub	holt RAW-Daten frameweise aus dem vom Grub übergebenen Speicher
import_dummy_raw	holt RAW-Daten frameweise (einkompiliert)
import_dummy	gibt ein Bild immer wieder aus (einkompiliert)
export_con	exportiert RAW-Frames an DROPSCON
export_ram	verwirft Daten
codec_xvid	Enkodiert bzw. Dekodiert mit Hilfe des XviD-Codecs
codec_dummy	keine Veränderung der Frames
ctrl_con	DROPSCON als User-Interface
ctrl_text	Normales User-Interface für Textausgaben

Tabelle 11: Testprogramm - vorhandene Plugins

größten Teil werden diese Funktionen von Codecs benötigt. Bei der Darstellung von Messergebnissen (z.B. bei Zeitmessung der Kodierzeiten) machte sich das Fehlen von *printf* für Floating-Point-Ausgaben bemerkbar. Beide Aufgaben ließen sich durch eine Portierung von Teilen der unter GPL lizenzierten *dietlibc* bewältigen.

Die Notwendigkeit einer komfortablen Wiedergabe der Testvideos wurde durch eine Übertragung und Anpassung der ebenfalls unter GPL stehenden *avilib* gelöst. Diese Bibliothek stellt Funktionalität zur Manipulation und zum Auslesen von Video- und Audio-Daten aus dem AVI-Container zur Verfügung. Die gesamten Fähigkeiten dieser Bibliothek hängen von der Verwendung von Dateioperationen zum Zugriff auf AVI-Dateien ab. Der Mangel eines verfügbaren Dateisystems machte eine Implementation von Dummy-Funktionen zum Zugriff auf eine ”Virtuelle Datei” notwendig. Diese Dummy-Funktionen bilden Dateizugriffe wie *read*, *seek*, *open* und *close* auf eine im Speicher befindliche, komplette AVI-Datei ab. In der gegenwärtigen Portierung der avilib ist nur Lese-Funktionalität verfügbar.

5 Leistungsbewertung

Relevant für die Leistungsbewertung der Implementierung sind Meßwerte mit verschiedenen Plugins sowohl beim De- als auch beim Enkodieren. Besonderes Augenmerk gilt dabei dem Vergleich mit Transcode unter Linux. Zur genaueren Analyse wurden Messungen über die Zeitverteilung in einzelnen Plugins vorgenommen. Die beim Entwurf im Kapitel 3.3 erörterten Erwartungen wurden anhand einer Messung der dort vorgestellten Rechenzeit-Qualitäts-Funktion überprüft.

Die angegebenen Werte sind als Richtlinie zu verstehen und sind von weiteren Faktoren wie Auflösung, Bildformat sowie den eigentlichen Bildinformationen der Testvideos abhängig. Alle Messungen wurden an einem mit AMD Duron 700 Mhz-Prozessor und 256MB RAM bestückten PC vorgenommen und mehrmals durchgeführt. Dabei ist zu beachten, daß unter Linux das Video von der Festplatte gelesen wird, während es unter DROPS bereits komplett im RAM verfügbar ist. Um den Einfluß der Festplattengeschwindigkeit (Stichwort: Cache) so gering wie möglich zu halten, wurden die ersten drei Meßergebnisse unter Linux nicht mit in die Auswertungen einbezogen. Unter beiden Betriebssystemen kam der gleiche binäre Code von XviD zum Einsatz (vgl. dazu Abschnitt 4.1).

Dieses Kapitel dient außerdem zu abschließenden Bemerkungen zu der im Abschnitt 4.2 beschriebenen und implementierten Infrastrukur.

5.1 Leistung beim De- und Enkodieren

Für die erste Messung wurde ein XviD-kodiertes Video in PAL-CCIR-601-Auflösung in einen AVI-Container verpackt und sowohl unter DROPS mit dem Testprogramm als auch unter Linux mit Transcode importiert, dekodiert und exportiert. Die Dauer dieses Vorgangs wurde gemessen und durch die Anzahl der zu dekodierenden Frames geteilt. Das Ergebnis liegt somit in Frames pro Sekunde (FPS) vor. Dabei wurde ein Video desselben (visuellen) Inhalts einmal im YUV- und einmal im RGB-Farbformat verwendet. Alle Messungen wurden mit der Qualitätsstufe 5 und der Bitrate 900 durchgeführt. Unter Linux wurden die gleichen Einstellungen für den Codec wie unter DROPS verwendet. Dies ermöglicht den Vergleich der beiden Systeme. Desweiteren wurden für die Messungen mit dem Testprogramm verschiedene Export-Plugins berücksichtigt (vgl. Tabelle 11). Abbildung 11 zeigt diese Ergebnisse.

Die Ergebnisse des Dekodierens lassen drei Schlüsse zu:

1. Die Leistung von Transcode unter Linux ist ca. 20-30 % höher,
2. YUV-kodiertes Material wird ca. 50 % schneller verarbeitet als RGB-kodiertes Video und
3. der Export auf die DROPSCON hat einen Leistungsnachlaß von etwa 40-50%.

Für die zweite Messung wurde die Dauer des Enkodierprozesses (Import, Enkodieren, Export) eines YUV-RAW-Videos in QCIF-Auflösung bestimmt. Unangetastet gegenüber dem Dekodieren blieben die Einstellung der Qualitätsstufen und der Bitrate. Wiederrum wurden die Messungen sowohl unter Linux mit Transcode als auch mit verschiedenen Import-Plugins unter DROPS durchgeführt. Bei dem Import von ”dsi_staticfs” und ”raw_from_grub” sowie bei ”Transcode (RAW)” wurde ein Video ohne Container verwendet. Bei den zwei anderen Messungen (”import_avi_from_grub” und ”Transcode (AVI)”) wurden

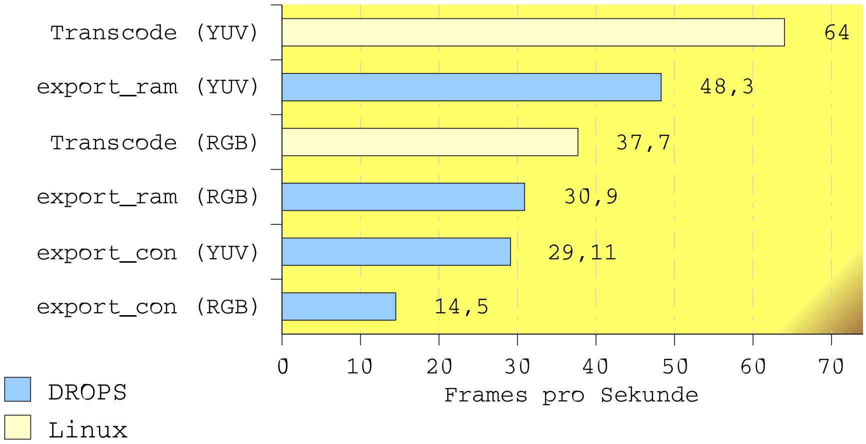

Abbildung 11: Messung 1: Leistungswerte Dekodieren

die unkodierten Frames in einen AVI-Container verpackt. An dieser Stelle sei daran erinnnert, daß bei den Messungen unter DROPS die Videos bereits komplett im RAM liegen, während diese bei Transcode auf der Festplatte gespeichert waren. Die Abbildung 12 zeigt die Ergebnisse.

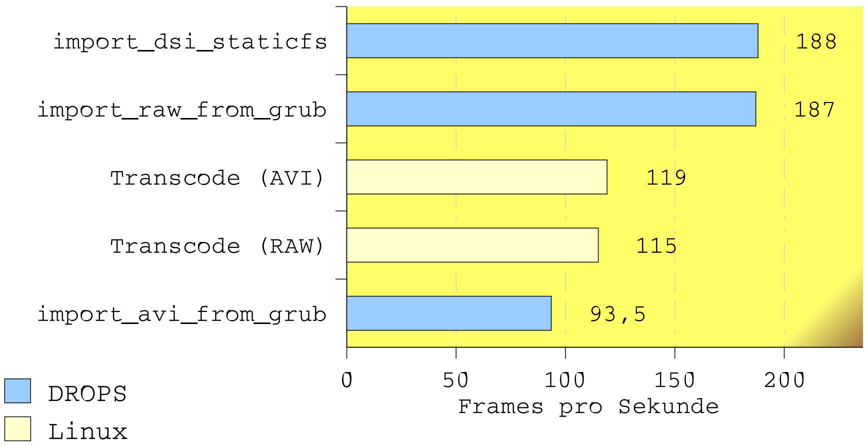

Abbildung 12: Messung 2: Leistungswerte Enkodieren

Beim Enkodieren zeigt sich, daß das Testprogramm nicht zwangsläufig langsamer als Transcode arbeitet. So sind beim Verarbeiten des Testvideos ohne Container die erreichten Frameraten unter DROPS höher als unter Linux. Die Begründung könnte dahingehend lauten, daß unter Linux das Video erst von der Festplatte gelesen werden muß.

Weiterhin gilt die Feststellung, daß bei Verwendung der avilib im Testprogramm unter DROPS die Geschwindigkeit des gesamten Vorganges nahezu halbiert wird. Daß dies kein allgemeines Problem des AVI-Containers ist, zeigen die Werte für Transcode unter Linux. Das Problem scheint in der Portierung der avilib-Bibliothek zu liegen. Auch könnte dies die schlechtere Leistung beim Dekodieren (Abbildung

11) erklären. So gilt es, bei weiteren Arbeiten hin zum Projektziel ”Videorekorder” die Portierung dieser Bibliothek zu überprüfen und gegebenenfalls deren Arbeitsgeschwindigkeit zu erhöhen. Genauere Messungen zum Import mit Hilfe der avilib sind im nächsten Abschnitt zu finden.

5.2 Zeitverteilung

Ziel dieser Meßreihe war es, die Geschwindigkeit beim Import zu analysieren. Die Import-Plugins ”import_dsi_staticfs”, ”import_raw_from_grub” und ”import_avi_from_grub” wurden dazu bei den Messungen berücksichtigt. Als Codec kam der portierte XviD-Codec als Enkodierer zum Einsatz. Beim Export war dies das ”export_ram”-Plugin. Abbildung 13 zeigt die minimale und maximale sowie die Durchschnittszeit für die Bearbeitung eines Frames in Microsekunden (μs).

Aus den Ergebnissen der Meßreihe läßt sich schlußfolgern, daß ...

- der Import vom DSI_staticfs bzw. vom ”raw_from_grub”-Plugin nahezu gleich schnell ist. Dies spricht für die in Kapitel 2.2.1 angegebenen Vorteile des DROPS Streaming Interfaces.
- das Auslesen der Daten aus dem AVI-Container extrem langsam im Vergleich zu dem containerlosen RAW-Video ist. Damit ist es möglich, die im Abschnitt 5.1 vorgestellten Ergebnisse zu erklären.

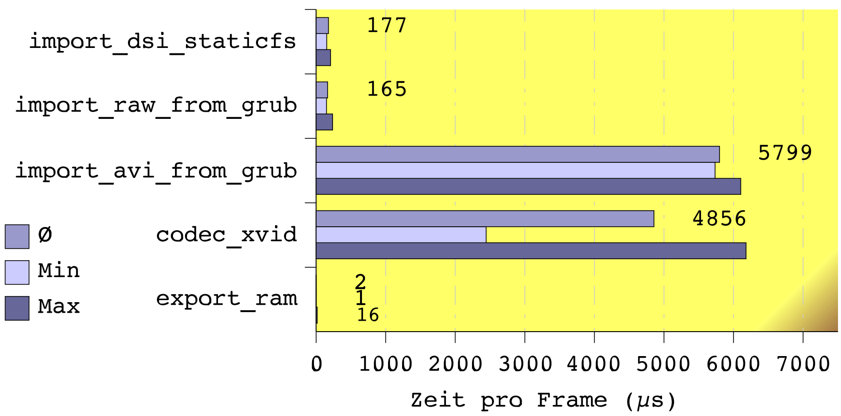

Abbildung 13: Messung 3: Zeitverteilung

5.3 Rechenzeit-Qualitäts-Funktion

Als Voraussetzung für das Funktionieren der im Abschnitt 3.3 entworfenen Infrastrukur gilt das Finden einer dort erläuterten Rechenzeit-Qualitäts-Funktion. Diese kann für den portierten Codec unter DROPS bestätigt werden. Dies verdeutlicht Abbildung 14.

5.4 Bemerkungen zur gewählten Infrastruktur

Im folgenden werden Bemerkungen zu der im Abschnitt 4.2 beschriebenen Infrastruktur gemacht.

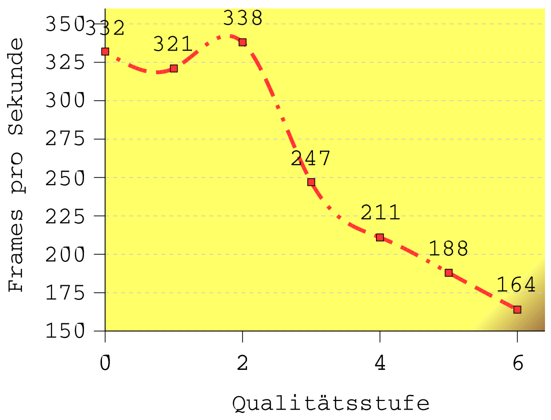

Abbildung 14: Messung 4: Qualitätsstufen - Frames pro Sekunde

Bei theoretischen Überlegungen wurde klar, daß die gewählte Infrastruktur einen erhöhten Aufwand für den "Kern" mit sich bringt, wenn bedacht wird, daß in Containern Daten mit verschiedenen Codecs enthalten sein können. So ist es möglich, daß ein Container gleichen Formates zum Beispiel ein XviD-kodiertes Video enthält, während ein anderer beispielsweise MPEG-2-Daten oder ein RAW-Video umgibt. Das Import-Plugin holt nun einzelne Frames aus dem Container. Dabei müßte das Plugin dem Kern mitteilen, welches Format im Container enthalten ist, um den passenden Codec auszuwählen und entscheiden zu können, ob zu De- oder Enkodieren ist. Das gleiche Problem tritt dabei auf der Seite des Exporteurs auf: Wurde vorher enkodiert, kann z.B. "export_con" die Daten nicht ausgeben, da es nur das RAW-Format unterstützt. Beim Dekodieren hingegen liegen die Frames RAW vor, so daß diese auf der DROPSCON darstellbar sind. Es muß vom Kern (oder dem Benutzer) sichergestellt werden, daß die gewählten Plugins und der Codec zusammenarbeiten können. Diese Logik ist aufwendig.

Ebenfalls gibt es in der momentanen Implementierung ein Problem mit dem Processing: wird vom Import-Plugin RAW-Material bereitgestellt, dann muß das Processing vor dem Enkodieren geschehen, hingegen beim Dekodieren erst nachher.

Eine Lösung beider Probleme findet sich in der Transcode-Architektur: alle Plugins haben zum Kern hin nur die Möglichkeit, RAW-Material bereitzustellen bzw. zu empfangen. Ein evtl. benötigter Codec wird innerhalb des Plugins bereits dekodieren oder enkodieren. Somit ist es möglich, unabhängig vom Video-Codec und Container-Format Frames zu bearbeiten. Die für die Rechenzeit-Qualitäts-Funktion nötigen Qualitätsstufen können dem Import- bzw. Export-Plugin übergeben werden. Diese nutzen dann die vorhandene Codec-Schnittstelle und übergeben dem gewählten Codec die Informationen zur Qualitätsstufe.

6 Fazit und Ausblicke

Der erste Schritt hin zu einem echtzeitfähigen Videorekorder ist mit der Portierung des MPEG-4-basierten XviD-Codecs bereits getan. Auch konnten durch das Entwickeln des Testprogramms Erfahrungen gesammelt werden, wie dieses im Hinblick auf die restlichen Projektziele, nämlich den Schwerpunkten Echtzeitfähigkeit und Wiederverwendbarkeit, anzupassen ist. Die Leistung des erstellten Testprogramms inklusive des portierten Codecs kann sich mit der Leistungsfähigkeit des Universal-Werkzeugs Transcode in bezug auf die Arbeitsgeschwindigkeit, jedoch nicht mit dessen Funktionsvielfalt, vergleichen lassen.

Weitere Schritte sind nötig, um das Projektziel vollends zu erreichen. Um nur einige davon zu erwähnen:

- Überdenken und Diskutieren der im Abschnitt 3.3 vorgestellten Lösung, besonders im Hinblick auf Scheduling
- Erhöhung der Benutzerfreundlichkeit durch Anpassungen auf DOpE und dynamisches Auswählen der Plugins (Autoprobing).
- Nutzung eines vorhandenen Dateisystems (wenn verfügbar)
- Entwicklung eines Video-Players (mit Synchronisation von Bild und Ton)
- Portierung und Analyse weiterer Codecs

A Glossar

AAC	Advanced Audio Coding
Autoprobing	probe, engl. untersuchen. Automatische Untersuchung von Daten
Bitrate	Anzahl der Bits pro Zeit
Cache	schneller Datenzwischenspeicher. z.B. angewendet bei Festplatten- oder Hauptspeicherzugriffen
CIF	Common Intermediate Format. Auflösung: 352x288
DCT	Diskrete Cosinus Transformation
FPS	Frames pro Sekunde
GCC	GNU project C und C++ Compiler
GNU	GNU is Not Unix
GPL	GNU General Public License
Grub	GRand Unified Bootloader
Heuristiken	schätzungsweises Bestimmen von Werten
I/O	Input/Output, engl. Eingabe/Ausgabe
Interpolation	näherungsweise Berechnung von Daten aus benachbarten Daten
IPC	InterProcess Communication. Möglichkeit zum Datenaustausch zwischen Prozessen
ISO/IEC	International Standardization Organization / International Electrotechnical Commission
JPEG	Joint Picture Experts Group. Verlustbehaftetes Kompressionsverfahren für Rasterbilder
Motion Estimation	engl. Bewegungsschätzung. Abschätzung der Bewegungen innerhalb von Bildfolgen
Mpeg_buffer	einheitliche Schnittstelle zum Zugriff auf einzelne Komponenten von Smart-MPEG
OSKit	Framework, daß es ermöglicht Betriebssysteme zu portieren
PAL	Phase Alternation Line. zwei verschiedene Normen: *Square Pixel*: sichtbare Auslösung 768x576 (Seitenverhältnis 4:3) *CCIR-601*: sichtbare Auslösung 720x576 (Seitenverhältnis 5:4)
Plugin	Softwarekomponente zur Ergänzung der Funktionalität. Wird dem Programm hinzugefügt
Processing	verarbeiten von Daten. z.B. bei Bildern: Spieglung, Drehen etc.
pSlim	PseudoSLIM. Protokoll zur Kommunikation mit DROPSCON
QCIF	Quarter Common Intermediate Format. Auflösung: 176x144
QoS	Quality of Service, engl. Güte eines Dienstes
RAW	unkodierte unkomprimierte Daten
RGB	Red Green Blue, engl. Rot Grün Blau. Farbmodell, bei dem Farben aus der Rot-, Grün- und Blauintensität zusammengesetzt sind

Scheduling	Ablaufplanung. Vorgehensweise des Betriebssytems zur Zuteilung von Betriebsmitteln
SLIM	Stateless Low-level Interface Machine. Geräteschnittstelle von SUN Microsystems entwickelt
Sprite	unbewegtes Bild. z.B. Panorama-Hintergrund
YUV	Farbmodell, bei dem die Farben aus Luminanz(Y) und Chrominanz(U,V) zusammengesetzt sind

B Literaturverzeichnis

[1] Folien zur Vorlesung ”Echtzeitsysteme”
2001, Hermann Härtig
http://os.inf.tu-dresden.de/Studium/Echtzeitsysteme/folien

[2] ”Multimedia-Systeme” - Kapitel 5
2002, Konrad Froitzheim
http://www-vs.informatik.uni-ulm.de/Lehre/MM-HTML/Kapitel5.doc

[3] ”DROPS - Overview”
2001, Michael Hohmuth
http://os.inf.tu-dresden.de/drops/overview.html

[4] ”Ein Konsolensystem für DROPS”
2001, Christian Helmuth
http://os.inf.tu-dresden.de/~ch12/sub/beleg.ps

[5] ”DOpE - a graphical user interface for DROPS”
2002, Norman Fenske
http://os.inf.tu-dresden.de/~nf2/files/DOpE/documents/DOpE-diploma.pdf

[6] ”Audio/Video Interleave Files AVI - MovieCodec.com”
1998-2002, Bjarne Lundgren
http://www.moviecodec.com/filetypes/avi.shtml

[7] ”ASF Specification - Windows Media Technologies”
http://www.microsoft.com/windows/windowsmedia/WM7/format/asfspec11300e.asp

[8] http://www.everist.org/texts/guyd/tech/10_apx_d.txt

[9] ”MPEGX.com "Daily News of VCD, DVD, DivX, MP3 and CD-R/RW Software"”
http://www.mpegx.com/videoguide.asp

[10] ”MPEG Home Page”
http://mpeg.telecomitalialab.com

[11] ”Taschenbuch der Informatik”, 4.Auflage
2001, Uwe Schneider, Dieter Werner
Fachbuchverlag Leipzig, ISBN 3-446-21753-3

[12] ”Overview of the MPEG-4 Standard”
2001, Rob Koenen

http://www.m4if.org/resources/Overview.pdf

[13] ”What is DivX?”
http://www.divx.com/support/what.php

[14] XviD.org :: Home of the XviD codec
http://www.xvid.org

[15] ”A real-time implementation of the XVID MPEG-4 codec supporting QoS for streaming videos”
http://www6.informatik.uni-erlangen.de/~ms/projects/XviD%20RealTime.pdf

[16] ”Linux Video Stream Processing Tool”
2002, Thomas Östreich
http://www.theorie.physik.uni-goettingen.de/~ostreich/transcode/

[17] ”Project RETAVIC”
2002, Maciej Suchomski
http://www6.informatik.uni-erlangen.de/retavic/

[18] ”Homepage memo.REAL”
2002, Andreas Märcz
http://wwwdb.inf.tu-dresden.de/research/memo.REAL/

[19] ”Smart_MPEG: MPEG decoder library specification and manual”
2000, Michael Hohmuth

[20] ”Entwurf und Implementierung von Sound- und Synchronisationskomponenten für das Smart-MPEG-Projekt”
2001, Jörg Nothnagel

[21] ”Forum” zu XviD
http://www.xvid.de/modules.php?op=modload&name=phpBB2&file=index

[22] ”Einfluß von Parametern des XviD-Codecs auf Enkodier-Zeiten und Qualität”
2002, Carsten Rietzschel

[23] ”avilib.h File Reference”
http://graphics.cs.uni-sb.de/NMM/Docs/nmm-0.0.3/html/avilib_8h.html

[24] ”diet libc - a libc optimized for small size”
http://www.fefe.de/dietlibc/