2007

Video Quality Measurement for 3G Handset

Zeeshan

Video Quality Measurement for 3G Handset

by

Zeeshan

Dissertation submitted in partial fulfilment of the requirements for the award of

Master of Research in Communications Engineering and Signal Processing

in

School of Computing, Communication and Electronics

University of Plymouth

January 2007

Supervisors

Professor Emmanuel C. Ifeachor

Dr. Lingfen Sun

Mr. Zhuoqun Li

Declaration

This is to certify that the candidate, Mr. Zeeshan, carried out the work submitted herewith

Candidate's Signature:

Mr. Zeeshan _____ Date: 25/01/2007

Supervisor's Signature:

Dr. Lingfen Sun _____ Date: 25/01/2007

Second Supervisor's Signature:

Mr. Zhuoqun Li _____ Date: 25/01/2007

Copyright & Legal Notice

Your attention is drawn to the University of Plymouth's Examination and Assessment Offences listed below. Read these carefully then sign below to indicate that you understand them, and understand that any breach of the regulations can result in a penalty such as a mark of zero being awarded for the work, a mark of zero being awarded for the module, or a mark of zero being awarded and the student being barred from taking the module again.

If this is an individual assignment (ie not a group project) then sign 1 below.

If this is a group project sign 2 below.

Examples of Examination and Assessment Offences
Note: this is an extract not the complete list

2.8 Copying or attempting to copy the work of another student, whether by overlooking his/her work, asking him/her for information, or by any other means.

2.9 The submission for assessment of material (written, computer-generated, visual or oral) or ideas originally produced by another person or persons, without clearly indicating both in the text and by use of bibliographic referencing, that the material is not original, such that the work could be assumed to be the student's own. This includes inter alia: the use of quotations or close paraphrasing without the use of quotation marks and referencing (plagiarism); the use of intellectual data or ideas without acknowledgement; copying, summarising or paraphrasing the work of another student or graduate; commissioning another person to complete the work which is then submitted as the students own work; the use of professional essay writing services or work drawn from the internet.

2.10 The unauthorised use of the work of another student (whether by taking a hard copy without permission or through access to a hard or floppy disk).

2.11 The representation of work produced in collaboration with another person or persons as the work of a single candidate.

Your signature(s)

1. I understand the above regulations and affirm that this is my own independent work.

Signed .....K. A. Ze. Shan..... Date 25-01-2007

2. We understand the above regulations and affirm that this is the independent work of the group and that we have not submitted un-referenced or un-acknowledged work which is wholly or partially the work of others outside our assignment group.

Signed ..... Date .....

.....

.....

ABSTRACT

The quality of video has become a decisive factor for the consumer of 3G video services to choose his mobile operator. It is, therefore, critical for 3G network operator, equipment provider and service provider to measure and hence maintain the video quality of video services they offer. A project has been proposed in University of Plymouth to develop a test platform to evaluate video quality for 3G handset using Asterisk PBX server. For this purpose, support for 3G-324M protocol and all the audio and video codecs (i.e. H.263 baseline level 10 and MPEG-4 simple profile @ level 0) mandated and recommended by 3G-324M standard should be added in to Asterisk®.

The purpose of this thesis is to identify the correct software implementation of H.263 baseline level 10 and MEPG-4 simple profile @ level 0 video codecs so that they can then be incorporated in to Asterisk®. This is the part of the above mentioned project. Open source FFmpeg-libavcodec is believed to support both MPEG-4 and H.263 codecs. Similarly Telenor H.263 codec is also free to use. This project tests both the capabilities and suitability of the above mentioned software packages/codecs for adding in to Asterisk to perform the required encoding and decoding. Experiments showed that FFmpeg-libavcodec can neither decode nor encode to MPEG-4 simple profile @ level 0. It seems that FFmpeg requires some major modifications in its source code to support MPEG-4 simple profile @ level 0 codec. Although FFmepg can decode and encode to H.263 baseline level 10, but it does not offer a fine control over bitrate while encoding, and reports very high muxing overhead while decoding, H.263 baseline level 10. Telenor H.263 codec can decode and encode to H.263 baseline level 10.without any problem. Telenor H.263 codec is, therefore, more suitable for incorporating in to Asterisk® than FFmpeg for decoding and encoding to H.263 baseline level 10 bitstreams.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank Professor Emmenuel C. Ifeachor who gave me the opportunity to work on this project despite my ignorance of Linux, and who has been a source of continuous encouragement and inspiration for me throughout my MRes.

Special thanks to my supervisor Dr. Lingfen Sun for her guidance and unending support throughout this project. She has always been very kind, cooperative and helpful to me.

I would also like to thank Mr. Zhuoqun Li for his supervision and constructive feedback. I learnt a lot from him. It was he who taught me how to handle large open source softwares without any documentation.

I am really grateful to my mother for her tireless and lifelong support and for always supporting my decisions (even when they lead to detours!).

This work is supported in part by Motorola.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

This chapter provides an introduction to the project "Video Quality Measurement for 3G Handset", jointly proposed by Motorola Basingstoke and University of Plymouth. It discusses the motivation behind this project, examines various video quality measurement techniques, and explains various components of the video quality test platform for 3G handset proposed by University of Plymouth. The chapter then throws light on aims and objectives of this project. It discusses the breakdown of the project into various tasks, distribution of these tasks among members of the team at University of Plymouth, and the task assigned to author (i.e. integrating H.263 and MPEG-4[1] video codecs in Asterisk® PBX for the purpose of developing video quality test platform for 3G handset). The chapter concludes with providing an outline of the rest of this thesis.

1.1 Motivation

With the introduction of third generation (3G) mobile telecommunication systems, mobile network operators are able to offer a wider range of video services to subscribers. Increased bandwidth and handset functionality in 3G mobile networks have made it possible to offer video services like video telephony, video streaming, and video clip download and mobile web surfing. Transmission of video over wireless has a specific set of requirements including small frame sizes, low frame rates, and low bit-rates. Furthermore, radio links are error-prone. Therefore, the videos over wireless are subject to errors resulting not only from compression but also from transmission errors (S. Winkler and F. Dufaux, 2003). It is, therefore, critical for 3G mobile network operator, equipment provider and service provider to measure the video quality of the services they offer to ensure customer satisfaction which is of vital importance for generating revenues. According to a recent market research although customers of 3G video services want the new video services at a cheaper price, they are not ready to accept a reduction in perceived picture quality (Psytechnics, 2005). That is why a project was set up in University of Plymouth in 2005 to develop a video quality test platform for 3G handset based on Asterisk PBX server.

1.2 How to measure video quality

The techniques to measure video quality can be divided in to two main categories: subjective video quality measurement, and objective video quality measurement.

The basic idea behind subjective video quality assessment methods is the use of human observers to evaluate the quality of video clips. These are obviously the most authenticated methods used for evaluating video quality because videos are watched by human beings and they are the best to assess their quality. However subjective video quality can be challenging because it may require an expert to assess it. Subjective methods have been standardized in

[1] H.263 and MPEG-4 here and in the rest of this thesis refer to H.263 baseline level 10 and MPEG-4 simple profile @ level 0, respectively.

ITU-R Recommendation BT.500 to evaluate video quality for television services (Wikipedia, 2006). Methods for multimedia quality assessment have been standardized in ITU-T recommendation P.910 (ITU, 2003). In essence, video sequences are shown to a number of viewers and then their opinion is averaged. This is called mean opinion score (MOS) and is a measure of subjective video quality. Since subjective assessments always require a special room and a certain number of people to evaluate video quality they are not suitable for continuous in-service evaluation of video quality (Watson and Winkler, 2000). Therefore, objective methods are used to measure the video quality algorithmically. These methods emulate the results of subjective assessment and measure video quality based on certain quality metrics that can be measured objectively. What follows is a brief introduction of various subjective and objective video quality evaluation methods.

1.2.1 Subjective video quality measurement

There are many factors that affect the results of subjective evaluation experiments like ambient light, observer vision ability, translation of quality perception into ranking score and display devices, levels etc. The following are two methods for subjective evaluation of video quality which are least suffered from these factors and have been standardized by the international telecommunications union (ITU) (Furht, F. and Marqure, O., 2003):

- Single stimulus continuous quality evaluation (SSCQE)
- Double stimulus continuous quality scale (DSCQS)

a) Single Stimulus Continuous Quality Evaluation

In this method observers dynamically intimate their assessment of the video quality of an arbitrarily long video sequence on a linear scale that is divided into five levels, as shown in Figure 1.1.

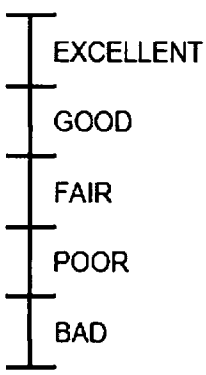

Figure 1.1 SSCQS sample quality scale (Furht, F. and Marqure, O., 2003)

The observers move a slider to any point on the scale that best reflects their impression of quality at that instant of time (Furht, F. and Marqure, O., 2003). This method is useful for identifying the rapid changes in video quality and is therefore suitable for real-time quality monitoring systems. However, in SSQCE viewers only view a single video without any reference video.

b) Double Stimulus Continuous Quality Scale

In the DSCQS method both the reference and the degraded video clips are shown one after other in the same session. The observers assess both sequences using sliders similar to those for SSCQE. The difference between the assessed scores of the reference and the distorted sequences gives the subjective impairment judgment (S. Winkler and F. Dufaux, 2003).

Figure 1.2 shows the evaluation procedure of DSCQS recommended by VQEG FR-TV phase-2 test.

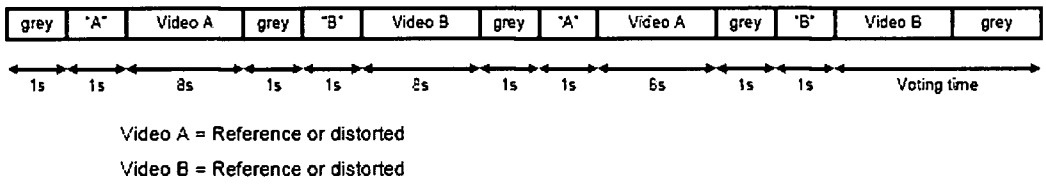

Figure 1.2 DSCQS testing procedure recommended by VQEG FR-TV Phase-II test (Furht, F. and Marqure, O., 2003)

The advantage of DSCQS over SSCQS is that assessment scores in the former least suffer from adaptation and background effects.

1.2.2 Objective video quality measurement

For digital videos, the traditional QoS indicators for measuring video quality such as mean squared error (MSE), peak signal to noise ratio (PSNR) or network statistics like packet loss rate (PLR) and block error rate (BLER) do not give results that conform to quality evaluation perceived by the user (S. Winkler and F. Dufaux, 2003). There are two basic reasons for it. First, different bits in an encoded video bit stream do not have the same importance. Secondly, the human visual system (HVS) is very complex and processes information in a very adaptive and non-uniform way (S. Winkler and F. Dufaux, 2003). These two issues highlighted the need for perceptual approach for quality measurement. Based on the availability of the original or reference video, objective video quality algorithms can be classified into full-reference (FR), reduced-reference (RR) and no-reference (NR) (LIVE, 2003).

a) Full-reference

In objective full reference (FR) video quality measurement method the algorithm has access to both the perfect and degraded version of the image or video to be evaluated. FR algorithms attempts to measure the error between perfect and distorted videos as would be measured by human visual error sensitivity features (LIVE, 2003). An example of such algorithm is PEVQ or perceptual evaluation of video quality by OPTICOM (OPTICOM, 2006).

b) No-reference

In many practical video service applications, the reference video is not available (LIVE, 2003). In such applications blind or no reference (NR) video quality assessment is used. Although human observers can successfully assess the quality of distorted video without using any reference, objective NR quality assessment is not easy (LIVE, 2003). NR methods can be used in any application because they do not require reference video. Hence they are more flexible but less accurate.

c) Reduced-reference

In reduced reference (RR) video quality assessment method the perfect video signal is not fully available. However, certain features are extracted from the reference video and transmitted to the quality assessment system on a side channel (called an RR channel) to help evaluate the quality of the degraded video (LIVE, 2003).

1.3 Proposed solution

The SPMC research group at the University of Plymouth and the team at Motorola Basingstoke jointly proposed a test platform for measuring video quality for a 3G handset during a video call. The proposed test plat form, based on open source Asterisk® PBX server is shown in figure 1.3. It will measure video quality using full-reference objective video quality measurement method. The two most important components of this test platform are:

a) 3G handset and local PC

Motorola is responsible for this part of the project which consists of an interface between 3G handset and a local PC. The purpose of this part of the test platform is to record a video clip from 3G handset during a video call (through an adapter between PC and 3G handset), and/or send a particular video clip from the PC to the 3G handset through the same adapter.

b) Asterisk® PBX server

Asterisk® is an open source software platform for converged telecommunications. It provides PBX functions and applications, as well as connectivity via TDM and packet voice (Millengence, 2006). For measuring video quality during a video between two 3G handsets, Asterisk® server should be able to:

1. Record video and audio streams during a video call sent by 3G handset via ISDN link
2. Loopback audio and video streams to 3G handset during a video call
3. Inserting video and/or audio streams into H.223 channel and send to 3G handset for testing
4. Act as a 3G-324M-to-SIP gateway and thus allow/bridge a video call from 3G handset to a SIP phone

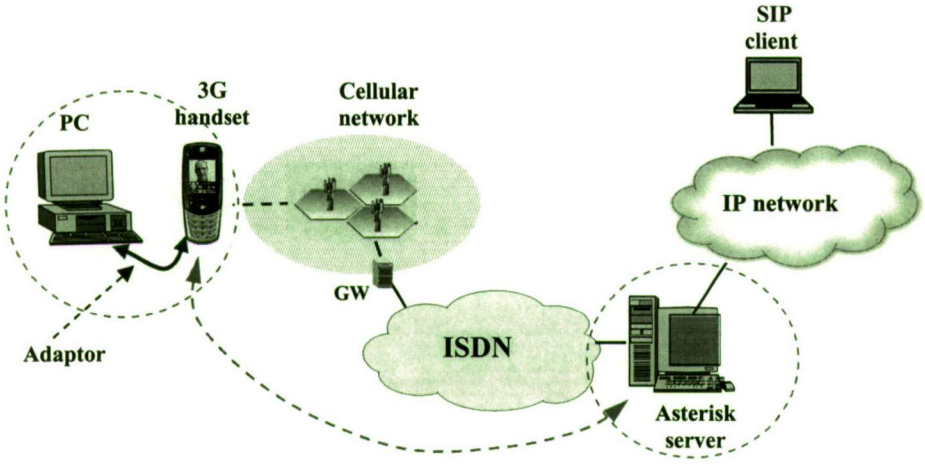

Figure 1.3 Video quality test platform for 3G handset (Sun, 2005)

A more detailed description of the role of Asterisk in the video quality test platform is given in chapter 2.

The video quality can be measured as follows. Consider Fig. 1. Suppose that the local PC contains a reference video clip recorded during a video call. It may send copy of a selected clip to 3G handset through an adapter. The handset may call Asterisk server and send this clip to Asterisk which may loop back it to the handset. After passing through the network the video clip would be degraded. The handset may send/record this degraded clip back to PC through the adapter. In PC, an objective FR algorithm (e.g. PEVQ by OPTICOM or PVA-Communications by Psytechnics) would measure the video quality of the degraded clip against the already saved reference clip.

1.4 Aim of the project

The aim of this project is to develop a video quality test platform based on Asterisk® server for measuring video quality for 3G handset during a video call using a full reference objective video quality measurement model. The role of University of Plymouth in this project is to extend Asterisk® to enable it to act as a 3G terminal and also serve as a gateway between ISDN/3G-324M and IP network in the video quality test platform and perform all the intended functions. For this, Asterisk® should support 3G-324M protocol stack (for video telephony over low bit-rate wireless circuit-switched networks) and all the audio and video codecs specified by it. At the time of writing this thesis, Asterisk® does not support this protocol or any of the audio or video codecs specified by it. Therefore, the following capabilities are required to be incorporated in Asterisk® before it can be used as a part of video quality test platform for measuring video quality during a video call between two 3G handsets:

- Support for 3G-324M protocol stack for real-time multimedia transmission over circuit-switched 3G networks
- Audio codec specified by 3G-324M standard
- Video codecs specified by 3G-324M standard and used by SIP phones

For implementing a video call between a 3G handset and a SIP phone and measuring video quality during this video call Asterisk should be able to translate between 3G-324M audio and video stream to RTP audio and video stream (respectively), and between 3G-324M signalling to SIP signalling.

The task of identifying the correct open source software implementations of video codecs mandated and recommended by 3G-324M standard, and then testing the capabilities and suitability of these software implementations to be incorporated in to Asterisk® was given to the author. This work is part of the project 'Video Quality measurement for 3G Handset'.

1.5 Outline of this thesis

This dissertation is structured into 6 chapters. Chapter 2 provides an introduction of Asterisk® PBX server, and discusses its history, architecture and features. It also explains the role of Asterisk® in this project and then provides information for the people who want to extend Asterisk®. Finally it discusses the future of Asterisk®. Chapter 3 an introduction of 3G-324M protocol stack for conversational video telephony over low bit-rate circuit-switched channels. Chapter 4 gives a brief introduction of MPEG-4 and H.263 video codecs and discusses specific profile and level of each codec used by 3G handset for conversational video telephony. Chapter 5 is the core chapter of this thesis. It discusses the proposed action plan to incorporate MPEG-4 simple profile @ level 0 and H.263 baseline level 10 video codecs in to Asterisk®. It gives an introduction of the two software packages (i.e. FFmpeg and Telenor H.263 Codec) used in this project. It then evaluates the feasibility of open source FFmpeg *libavcodec* and Telenor H.263 codec to be added to Asterisk®.

Finally, chapter 6 discusses the outcomes of research and results of experiments carried out in this project. It compares the original objectives with the work that have been done so far. It also gives some suggestions for future work and then concludes the thesis.

2 Asterisk® PBX Server

Most of the current Asterisk® literature available on the internet and in the books is intended for the user's of Asterisk®. Information about its installation, configuration, operation and applications is readily available. However, there is not much information available for the people who want to extend its features or modify it to use it for their own custom applications. This chapter is intended to elaborate the role of Asterisk® in this project as well as to provide useful information for people who want to extend Asterisk® or incorporate new functionalities in it.

2.1 Introduction to Asterisk®

Asterisk® is an open source, software based hybrid TDM and packet voice PBX and IVR (interactive voice response) platform with ACD (automatic call distribution) functionality (Millengence, 2006). It runs on Linux, OpenBSD, FreeBSD, Mac OS X and Sun Solaris (Digium, 2006). It is perhaps the most comprehensive, extensible and flexible PBX available in the world. Yet it is far less expensive than any hardware based proprietary PBX. It can perfectly integrate traditional standards based telephony systems and VoIP systems. It can interoperate with Voice over IP (VoIP) without any additional hardware. Asterisk® also supports a variety of TDM protocols for handling and transmission of voice over traditional telephony interfaces (Digium, 2007). It can interface with all standard digital and analog telephony equipment using inexpensive hardware. The features offered by Asterisk® are found only in state-of-the-art and expensive proprietary PBXs from companies such as Lucent and Nortel. It offers features like voicemail services with directory, interactive voice response, call queuing, call data record (CDR) generation and integration of open source text-to-speech and voice recognition systems (voip-info.org, 2006). Since Asterisk® is an extensible piece of software, the list of features it offers is ever increasing. According to Mark Spencer, "the beauty of Asterisk® is its ability to be extended using its APIs, dynamic module loader, and AGI scripting interface. End users can even write their own applications that run on the system in C or any scripting language of their choice".

2.2 History of Asterisk®

The saying "necessity is the mother of invention" particularly holds for all open source projects. Mark Spencer, the creator of Asterisk®, could not afford to buy a telephony system for his new company 'Linux Support Services' which was supposed to provide telephone support round the clock (Allison *et al.* 2003). He thought it feasible to connect the telephone line to a computer with some expansion card and process the phone calls in software (Mandriva, 2006). He wrote software to control such expansion cards which resulted in the first version of Asterisk® (Mandriva, 2006). After that he teamed up with Jim Dixon of the Zapata Telephony Project to create first PCI telephony interface card to connect Asterisk® to the PSTN (Allison et al. 2003) and changed the name of his company to Digium™.

Traditional TDM (time division multiplexing) hardware available at that time was very expensive. Mark and Jim therefore decided to move the TDM and DSP (digital signal processing) processing in software inside Asterisk® and let the interface card contain only the basic electronic components required to interface with a telephone line (Allison et al. 2003). Zapata Telephony developed cards for pseudo TDM (i.e. software TDM) interfaces which they called Zaptel (Mandriva, 2006).

It was still required to enable Asterisk® to interoperate with VoIP systems. The proprietary H.323 VoIP protocol was bulky. Mark, therefore, developed his efficient VoIP protocol called IAX (Inter Asterisk Exchange) (Mandriva, 2006). He did not like the proprietary H.323 VoIP protocol being bulky (Mandriva, 2006). Later on, support for other packet voice protocols like SIP, H.323, MGCP and VoFR (voice over frame relay) was also added in Asterisk® (Mandriva, 2006). Mark Spencer and the Asterisk® community from around the world are continuously making improvements to the source code of Asterisk®.

2.3 Asterisk® download and installation

Asterisk® users can easily download the tarball releases of various Asterisk® projects (i.e. Asterisk, Asterisk-Addons, Asterisk-Sounds, Libpri, Libiax, and Zaptel) from the website *http://www.asterisk.org/downloads*.

For successful installation of Asterisk® on the Linux SuSE system, there are certain packages that must also be installed from SuSE distribution disks. These packages include cvs, bison, readline-devel, termcap, newt, ncurses-devel, zlib-devel, openssl-devel, libidn-devel, e2fsprogs-devel, krb5-devel and glibc-devel.

To install Asterisk on the Linux system, login as root and extract the tarball source in the *usr/src/* directory. In the console, change directory to *usr/src/asterisk* and then issue the following commands in the given order under root:

```
#make clean

#make

#make install
```

If Asterisk is to be used only for pure VoIP network then only Asterisk package is required. However, if you want to connect to PSTN or ISDN PRI line using Digium™ interface cards then Zaptel drivers are also required. Libpri library is required only if you are connecting to an ISDN PRI line otherwise it is optional. Please note that exactly the same procedure is used for installing Zaptel and Libpri. However, if Zaptel and Libpri are also required, they should be installed before installing Asterisk® (Madsen *et al.* 2005).

After successful installation of Asterisk, run the command `make samples`. This will copy the standard sample configuration from the directory *configs/* to the directory */etc/asterisk*. To start Asterisk®, simply issue the command `asterisk -vvvgc` in the directory *usr/src/asterisk*. The number of 'v's in the above command sets the level of verbosity of the output.

Asterisk® developers need the most up-to-date source code which can be obtained from Digium™ SVN repository using the following command:

```
#svn co http://svn.digium.com/svn/asterisk/trunk asterisk
```

Similary for Zaptel and Libpri use the following commands:

```
#svn co http://svn.digium.com/svn/zaptel/trunk zaptel
#svn co http://svn.digium.com/svn/libpri/trunk libpri
```

2.4 Architecture of Asterisk®

Asterisk® is essentially a middleware that connects telephony technologies with telephony applications (Digium, 2006). Telephony technologies include both VoIP technologies (e.g. SIP, H.323 etc.) and traditional TDM technologies (e.g. ISDN PRI, analog POTS etc.). A few examples of telephony applications are call bridging, call parking, call queuing, voicemail and IVR (Digium, 2006). As shown in figure 2.1, Asterisk® has a central switching core comprising various engines that are imperative for its operation. There are four APIs around this central core for modular loading of telephony applications, hardware interfaces, file format handling, and codecs (OSTG, 2006). Asterisk® provides transparent switching between all supported interfaces and can, therefore, join different telephony systems into a single switching network (Mandriva, 2006). Probably, the most prominent features of the architecture of Asterisk® are its flexibility and modularity. The modular design of Asterisk® allows the central core to deal with the internal interconnections of the PBX independent of the specific channel interfaces, protocols or codecs being used (OSTG, 2006).

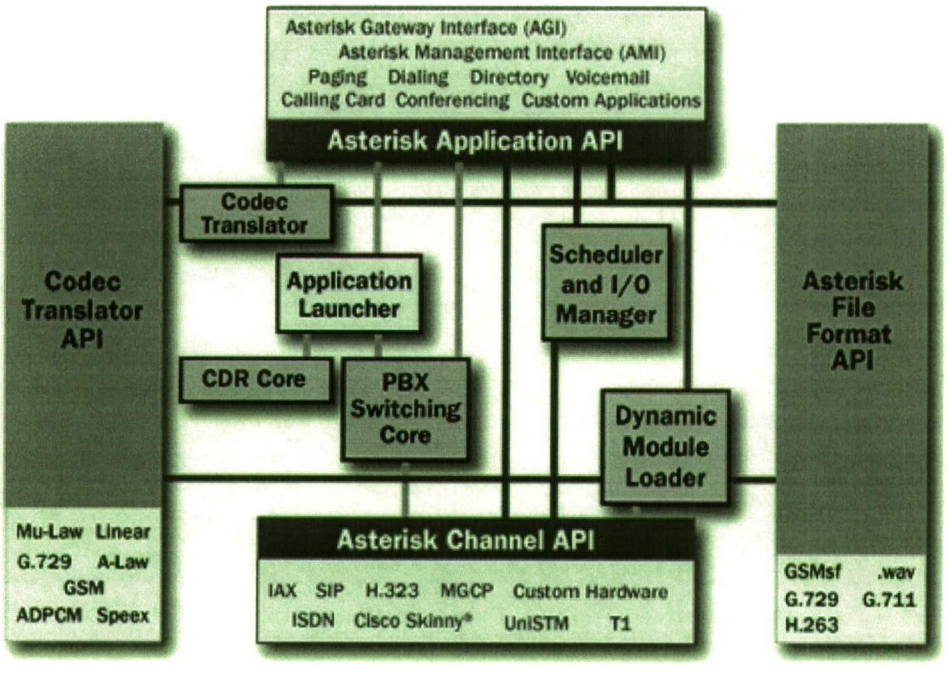

Figure 2.1 Architecture of Asterisk® (Digium, 2006)

2.4.1 Asterisk® Core

The Asterisk® core comprises the following:

a) Dynamic Module Loader

When Asterisk® gets started, the dynamic module loader, being a run-time loader, loads all the modules at run-time for channel drivers, applications, codecs, file formats, functions and call data record back-ends (Digium, 2006). The source code for dynamic module loader is contained in the file `loader.c` in the directory */usr/src/asterisk*[1].

b) PBX switching core

The PBX switching core transparently handles calls from various software and hardware interfaces according to the dial plan. It connects call between two or more users or between a user and an automated task. It uses application launcher for ringing phones, connecting to voicemail and dialling out, etc. (OSTG, 2006). The dial plan is contained in the file `extensions.conf` in the directory */etc/asterisk*. It contains the sequence of instructions to be followed by Asterisk PBX to handle inbound and outbound calls.

c) Application launcher

Application Launcher launches applications to serve the users of Asterisk PBX (Digium, 2006). Examples of such applications are voicemails, file playback, directory listing and dialling out etc. The source code of Application Launcher is integrated in the file `pbx.c` in */usr/src/asterisk*. In file `pbx.c`, applications are launched through a function `st_pbx_run_app` which calls another function `pbx_exec`. The function `pbx_exec` takes address of the required application structure and address of the channel structure (on which this application is to be run) as arguments and executes the application on the specified channel.

d) Codec translator

Codec translator allows users using different audio codecs to transparently talk with each other (Allison et al. 2003). It makes use of codec modules (in the folder *codec* in the directory */usr/src/asterisk*) to encode outgoing audio stream from linear to a specific audio compression format and to decode the incoming audio stream from a specific audio compression format to linear. Codec translator is contained in the file *translate.c* in *usr/src/asterisk*.

[1] It is assumed that source code of Asterisk® has been extracted in */usr/src/asterisk*.

e) Scheduler and I/O manager

The scheduler and I/O manager ensures best efficiency under different load conditions by scheduling low-level tasks and performing system management (Digium, 2006). It allows many users to talk at the same time. The source code for scheduler and I/O manager is contained in the files `sched.c` and `manager.c` in *usr/src/asterisk* respectively.

2.4.2 APIs for loadable modules

Modularity is another salient feature of Asterisk®. Different functions of the server are implemented as modules that are loaded and initialized by the dynamic module loader (Millengence, 2006). Asterisk® has four basic sets of APIs for interfacing with different components.

a) Channel API

The channel API allows the Asterisk® switching core to handle the type of connection a caller is arriving on (OSTG, 2006). The modules are loaded dynamically to handle lower layer (technology specific) details of these connections (Digium, 2006). Connections based on both VoIP and circuit switched technologies are supported. Implementation of channels as modules allows Asterisk® to handle all kinds of conversations in the same way (OSTG, 2006).

b) Codec API

Asterisk® can transcode between various audio encoding and decoding formats allowing users using different audio codecs to communicate with each other. The codec translator API enables Asterisk® core to deal with encoded voice flexibly (OSTG, 2006).

c) File format API

The File Format API enables Asterisk® to read and play sound files in different file formats. It also allows writing of various file formats for storage of data in the file system (Digium, 2006). This makes Asterisk® based applications more flexible in dealing with ring tones, DTMF etc. (Mandriva, 2006).

d) Application API

The application API permits Asterisk® to run any task (or application) to perform various functions which a PBX system performs now, or might perform in future (Digium, 2006). Each task has been incorporated in Asterisk® as a separate module. Applications are used in the dial plan.

The application API enables developers to develop new telephony applications that can interact with Asterisk® core directly (OSTG, 2006). Third party applications like calling

cards, conferencing and voicemail can also make use of the application API to take advantage of a variety of Asterisk® features (Mandriva, 2006).

Note:

There is another interface provided by Asterisk® called Asterisk Gateway Interface or AGI, which allows external programs to be launched from within Asterisk® in accordance with the dial plan rules (Mandriva, 2006).

2.5 Asterisk® start-up

The following is the sequence of events that take place on Asterisk® start-up:

1. Asterisk® event logger is initialised. The `event_log` file can be found in the directory *var/log/asterisk*. Review of the source code of the file `logger.c` in the directory *usr/src/asterisk* indicates that this file keeps a log of the activities on each channel.

2. DNS manager (i.e. file `dnsmgr.c` in *usr/src/asterisk*) is initialised which may permit the creation of managed DNS lookups. However, by default the creation of managed DNS lookups is disabled.

3. The dynamic module loader loads any modules defined as preload modules (using "preload") in file `modules.conf` located in directory *etc/asterisk*. Preload modules are modules that need to be loaded before the Asterisk core has been initialised.

4. Asterisk® manager interface, abbreviated as AMI, is initialised and the default actions that can be performed on AMI are registered. The Asterisk CLI command `show manager commands` displays the list of actions allowed on AMI.

5. The Asterisk PBX core (the file `pbx.c` in *usr/src/asterisk*) is initialised and built-in applications in PBX core are registered.

6. The Asterisk dynamic module loader is started. It first parses the file `modules.conf` and loads any modules found in it that needs to be loaded right away. Such modules are indicated after `'load =>'` in the aforementioned file. Then it loads all the modules listed in */usr/lib/asterisk/modules*. First the modules related to resources are loaded then the PBX modules, then the channel modules and then all other modules related to applications, codecs and formats are loaded.

Figure 3.2 shows the flow chart for sequence of events at Asterisk's start-up.

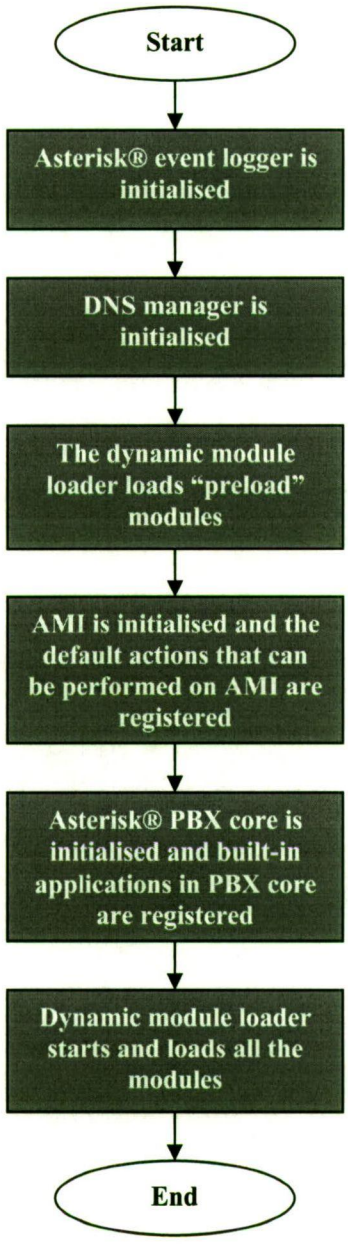

Figure 2.2 Flow chart for Asterisk's start-up

2.6 Features of Asterisk®

The list of features supported by Asterisk® includes the features supported by a classical PBX system plus the features offered by any state-of-the-art proprietary PBX in the world. Moreover the list of features supported by Asterisk® can never be exhaustive because Asterisk® developers, both from the community and from Digium™ Inc., are continuously working to extend it. Some of the Asterisk's features that are found only in most modern proprietary PBXs are listed below. The complete list of Asterisk's features can be found in Appendix A.

2.6.1 Call Features

The following are some of the call features of Asterisk®:

a) Voice mail

Asterisk® PBX provides the facility for the caller to record a message when the callee is away. The voice mail can also be delivered to the user via email or a simple email notification without the attached voice mail can also be sent (Asterisk IT, 2006). Voice mails can be retrieved by dialling the voice mail number (Asterisk IT, 2006). Voice mails are stored as audio files in Asterisk®.

b) Call queuing

Asterisk® allows the telephony system of an organisation (e.g. a call centre) to receive more calls then the number of attendants or extensions available. Even if all attendants are busy, the calling customers will still be automatically answered and held in a queue (Voipfone, 2005). This feature is mostly used by call centres.

c) Call parking

Calls can be parked by transferring to a parking lot system (or virtual extension). The number of parking lot is announced while parking the call. Anyone can retrieve the call by dialling the number of parking lot (Asterisk IT, 2006). The parked call is sent back to the extension that parked it, if not retrieved within a specified time (Asterisk IT, 2006).

d) Call routing

Asterisk® allows each user to have a separate phone number for each extension without the need of a separate phone line for each user. This facilitates external callers (e.g. customers of a company) to call each user directly (Wikipedia, 2006).

e) CDR (call details records)

Asterisk® can store log of calls including phone numbers of source and destination, duration of call, date and time in a text file. A third party database can also be integrated with Asterisk® and call details recorded in the database instead of a text file (Asterisk IT, 2006).

f) IVR (interactive voice response)

Asterisk® is not just a PBX; it is also an IVR platform. IVR technology allows organisations to deal with customers or clients without human interaction. It allows callers to navigate through a voice menu system to interact with a database. Asterisk® may play a voice prompt (saved as audio file) asking user to press a certain key for the desired option or task. This feature is also intended to be used by Motorola as a component of video quality measurement test platform.

g) VoIP gateway

Asterisk® can perfectly bridge IP network and the PSTN and thus serve as a VoIP gateway (voip-info.org, 2006).

h) Conference bridging

Asterisk® allows more than two callers at different locations using different connection types (e.g. a conference bridging between remote landline, mobile phone and VoIP connection like SIP) to have a conference call (Asterisk IT, 2006).

2.6.2 Supported codecs

Asterisk® can perform transcoding between different audio compression formats. This transcoding is completely transparent to users using different audio compression formats. As a part of this project AMR (Adaptive multi-rate) codec has been integrated in Asterisk®. However, testing of AMR is still to be performed. Current audio codecs supported by Asterisk® are (Digium, 2006):

- ADPCM (adaptive differential pulse code modulation)
- G.711 (A-Law & μ-Law pulse code modulation)
- G.723.1 (supported only in pass through mode)
- G.726
- G.729 (supported through purchase of commercial license through Digium)
- GSM
- iLBC (internet low bit-rate codec)
- Linear
- LPC-10 (linear prediction coefficient 10)
- Speex (used for packet voice)

2.6.3 Support for packet voice protocol

Asterisk® supports following protocols for voice over packet networks without any hardware (Digium, 2006).

- SIP (Session Initiation Protocol)
- H.323
- IAX™ (Inter-Asterisk® Exchange)
- MGCP (Media Gateway Control Protocol)
- Cisco SCCP (Skinny Client Control Protocol)

2.6.4 Interoperability with traditional telephony

The flexibility of Asterisk® makes it easy for the developers to add new interfaces and technologies in it. Interoperability with traditional telephony is provided through both Zaptel Pseudo-TDM interfaces and Non-Zaptel interfaces (Digium, 2006).

a) Zaptel hardware interfaces

The Zaptel Pseudo-TDM interfaces are as fast as hardware TDM. They offer the same real time performance as offered by hardware TDM interfaces but are less expensive and more flexible (Digium, 2006). Zaptel interfaces support traditional telephony protocols including ISDN PRI (i.e. both E1 and T1), RBS (Robbed-bit signalling), FXS (Foreign Exchange Subscriber), FXO (Foreign Exchange Office), E&M, Wink, Feature Group D, Groundstart, and Loopstart (Digium, 2007). Digium™ sells the hardware for all compatible Zaptel interfaces at a relatively low price.

b) Non-Zaptel hardware interfaces

Non-Zaptel hardware interfaces provide interoperability with traditional telephony but do not offer real time performance like Zaptel interfaces. A few examples are (Digium, 2006):

- OpenPCI - a 4 or 8 port FXS or FXO interface by Voicetronix™
- Internet PhoneJack single FXS interface (supports Linux telephony interface) by Quicknet
- Internet LineJack single FXS or FXO interface (supports Linux telephony interface) by Quicknet
- ISDNLinux – Basic Rate ISDN interface for Linux
- OSS/Alsa – Full duplex sound card interfaces

2.7 Role of Asterisk® in this project

To act as a component of video quality test platform for 3G handset (shown in figure 1.3), Asterisk should be able to perform the following functions:

- Record video and audio bitstreams from H.223[2] channel during a video call to files.
- Loop back audio and video streams (sent by 3G handset) to 3G handset in order to get a degraded version of the original video clip
- Inserting video and/or audio streams into H.223 channel and send to 3G handset for testing
- To act as a terminal for a 3G video call so that Motorola can analyse the success rate of 3G video calls by making video calls from their 3G handsets to Asterisk®.
- To act as a 3G-324M-SIP gateway as shown in Figures 1.3 and 2.3 so that a 3G handset can call a SIP client on the Internet and video quality can be measured during this video call.

  For this Asterisk® should be able to translate between two protocols — 3G-324M on the wireless UMTS side and SIP on the fixed IP network side. This implies that it should be capable of translating 3G-324M audio and video stream to RTP audio and video stream; and 3G-324M signalling to SIP signalling, and vice versa

  However, to translate between 3G-324M audio/video stream and RTP audio/video stream, the audio and video codecs specified by 3G-324M and used by SIP clients should also be incorporated in Asterisk®. Most SIP phones support G.711 audio codec. This audio codec is already supported by Asterisk. However 3G-324M terminals are required to support AMR codec. Hence this audio codec should be added in to Asterisk. Likewise most SIP phones support H.263 video codec with a CIF (352 x 288) resolution but 3G-324M terminals are required to support H.263 baseline level 10 codec with QCIF (176 x 144) resolution so Asterisk might also require a transcoder function (in addition to the support for H.263 codec) to dynamically convert between the two different video screen sizes (NMS Communications, 2006). MPEG-4 simple profile @ level 0 codec should also be added in to Asterisk as it is recommended for 3G-324M terminal by 3GPP (3GPP, 2004) and it might be possible in future that some SIP phones also support it.

[2] H.223 is a multiplexing protocol and is a component of 3G-324M.

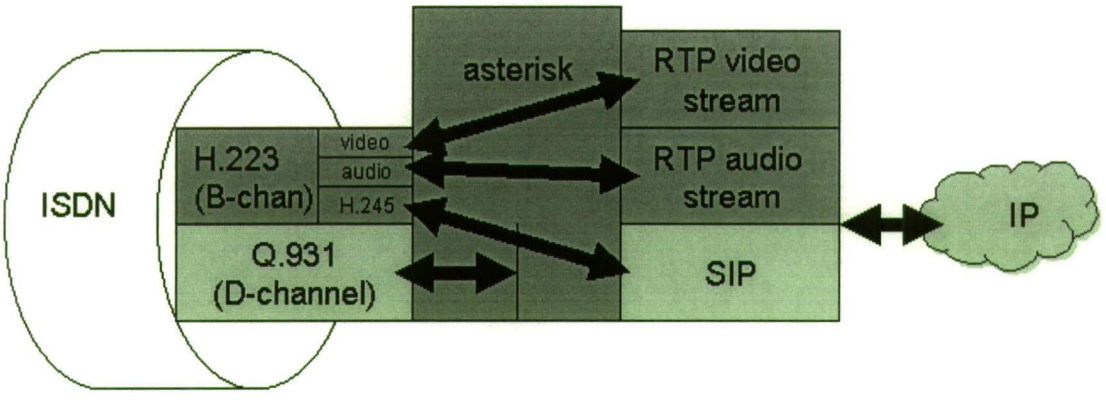

Figure 2.3 Asterisk® acting as a 3G-324M-SIP gateway (voip-info.org, 2006)

2.8 Writing extensions to Asterisk® source code

As mentioned earlier, all of the applications, channel drivers, codecs, file formats and even some functions have been implemented in Asterisk® as modules. Therefore to add a new application, channel, codec or format it is required to learn how to add a new module in to Asterisk®. Writing a new module and integrating it with Asterisk® involves the following steps:

2.8.1 Write the module

All modules have the same format. They are compiled into shared object (`.so`) files and are installed to */usr/lib/asterisk/modules* (Lobster Technologies, 2006). The loading of a module can be switched on and off by modifying the file `modules.conf` in the directory */etc/asterisk/modules*. A module must export the following functions (Lobster Technologies, 2006):

a) int load module (void)

When a module is loaded by Asterisk® core, this function is called. What is in the body of this function depends on the type of module (i.e. application, channel, codec etc.) being loaded. Usually there is a function inside this function which registers the module with Asterisk. The best approach towards writing this function and the following function (i.e. `int unload module (void)`) is to look at the same function in already implemented modules in the same folder. For example if a new application is being implemented as a module; look at other modules in the */usr/src/asterisk/applications* folder.

b) int unload module (void)

When Asterisk unloads a module, this function is called. This function also contains another function which unregisters the services offered by the module.

c) char *description (void)

This function gives a brief description of the module which is displayed when user types the command `show modules` at Asterisk CLI (Lobster Technologies, 2006).

d) int reload module (void)

This function is called when a user types the command `reload` at Asterisk CLI to reload a module (Lobster Technologies, 2006).

e) char *key (void)

This function returns the value of ASTERISK_GPL_KEY which makes the writer of the module state that his module is licensed under GNU General Public License. This ensures that non-free modules for business edition of Asterisk® are not linked against the free edition of Asterisk® (Lobster Technologies, 2006).

2.8.2 Naming the module

The module name should also follow the naming convention of Asterisk®. For example applications are named as 'app_*app-name*.c', codes are named as 'codec_*codec-name*.c' and so on.

2.8.3 Copy the module in the relevant folder

Once a module has been written completely including above mentioned functions, it should be copied into the relevant folder. For example, if a module `app_3g324m.c` is being added, it should be copied in *usr/src/asterisk/apps* folder.

2.8.4 Modify the make file in the relevant folder

The make file in the relevant folder contains a list of already implemented modules of the same type. Suppose a new module named `app_3g324m.c` is being added then we need to add `app_3g324m.so` to the APPS list in *usr/src/Asterisk/apps/Makefile*.

2.8.5 Clear /usr/lib/asterisk/modules

Stop Asterisk if it is running and clear the directory */usr/lib/asterisk/modules* by running the command `rm *.*` (Lobster Technologies, 2006)

2.8.6 Recompile and run Asterisk®

Recompile by running the command `make upgrade` in *usr/src/asterisk* folder (Lobster Technologies, 2006).

Run Asterisk (if there are no errors in the last step) and that's it.

2.9 Future of Asterisk®

The features and capabilities of Asterisk are growing rapidly. VoIP guru Jeff Pulver states: "*They are developing a sophisticated PBX on a PC with the (capability) of a $100,000 PBX...It will be a world class PBX that runs on Linux. You can have a PBX for the cost of a PC*". Jon 'Maddog' Hall, president of Linux International, states: "*I predict that over next three years, VoIP using an open-source solution, such as Asterisk; will generate more business than the entire Linux marketplace today*" (Xorcom, 2006).

Mobile to mobile video calls are becoming very popular in both Asia and Europe. This enables mobile users separated by geographical distances to share visual information with each other as if they were standing side by side. Network operators who realise the potential revenues generated by video calls are now offering mobile to IP terminal video terminal communication. For that purpose, a device acting as a 3G-324M and SIP gateway is required (NMS Communications, 2006). Companies like NMS Communications are selling such 3G-324M-SIP video gateways at considerably high price. If the complete functionality of 3G-324M and SIP gateway is successfully added in to Asterisk, it would become a much cheaper alternative to expensive commercial 3G-324M-SIP gateway.

The architecture of Asterisk is such that it should easily adapt to new protocols and the technologies which are still to be dreamed. In the future, Asterisk should support any method of passing voice calls. In addition, Asterisk will be having new functionalities offered by VoIP as this technology fully grows (Linuxdevices.com, 2006).

3 3G-324M protocol

Although 3GPP and 3GPP2 envisage 3G as an all IP wireless network that would seamlessly stream audio and video over IP links, the all IP network is still years away due to a number of reasons. First, the current IPv4 networks are not capable of handling delay sensitive applications. Second, they do not have sufficient address to handle so many 3G mobiles. Third, it is very difficult (if not impossible to) to deliver IP packets over wireless networks with high BER (bit-error rates) (CommsDesign, 2003). However, the inability of today's IP networks to handle the delay sensitive applications like video conferencing and video telephony could not stop 3G to fulfil its promise of delivering these differentiating multimedia services. The newly emerged 3G-324M standard can support the real-time streaming multimedia services over existing wireless circuit switched networks (Telephony online, 2003). Due to its circuit switched nature it possess all the qualities required for streaming real-time media, including fixed delay, low overhead of codecs, and no header overheads like IP/UDP/RTP (CommsDesign, 2003).

The 3G-324M standard has been derived from the ITU H.324 protocol standard developed for video telephony over public switched telephone network. H.324 is an umbrella standard. Other main standards and components included in H.324 are H.223 multiplexer, V.34 modem and H.245 call control, audio (G.723.1) and video (H.263 and H.261) codecs. Through a series of Annexes to H.324 and its component protocol H.223, it has been made suitable for use on mobile devices and networks. Annex C of H.324 and Annexes A, B, C, and D of H.223 are referred to as H.324M. Annexes A, B, C and D of H.223 provide increasing levels of error resilience to deal with error-prone mobile networks (Smith and Jabri, 2004).

The 3GPP is responsible for creating technical specifications for 3G systems based on UMTS/WCDMA. It has modified H.324M to create 3G-324M standard. The following are the basic differences between 3G-324M and H.324M:

- H.324M mandates the use of G.723.1 as audio, and both H.263 and H.261 as video codecs as shown in figure 3.1 whereas 3G-324M mandates the use of AMR as audio and only H.263 as video codecs as shown in figure 3.2.
- H.324M mandates the use of Annex A, B, C, and D of H.223 whereas 3G-32M mandates the use of only Annex A and B of H.223 as can be seen from figures 3.1 and 3.2.

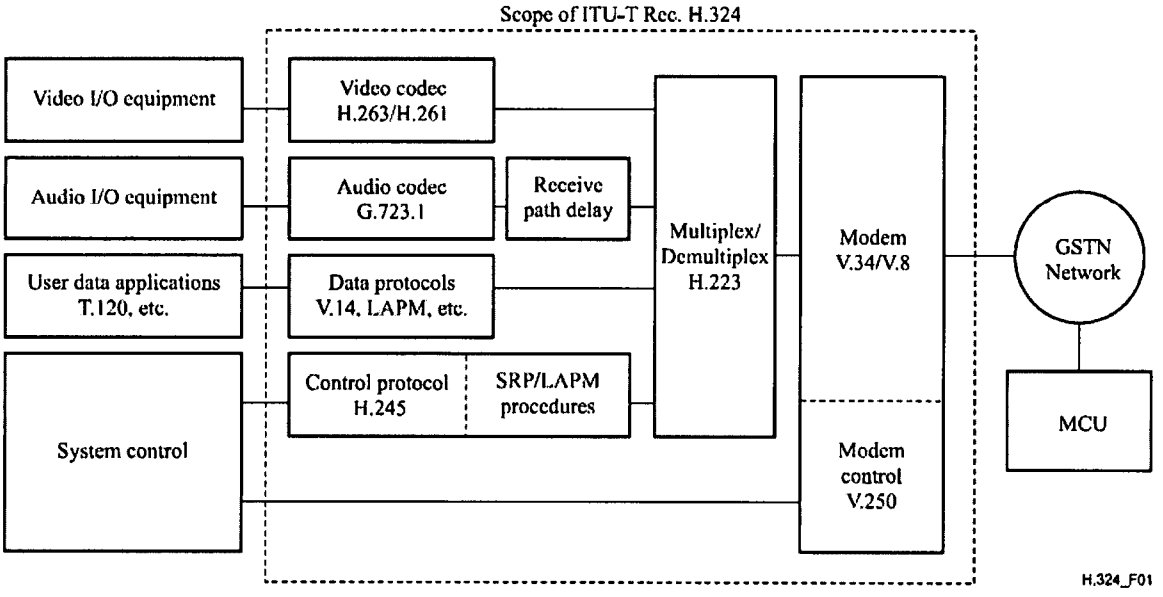

Figure 3.2 Block diagram of H.324 system (ITU, 2005)

3GPP has introduced two technical specifications for structure and implementation requirement of 3G-324M. It defines TS 26.112 for call set up procedures using the 3G air interfaces and TS 26.111 for 3G-324M initiation and operations occurring when the call is established.

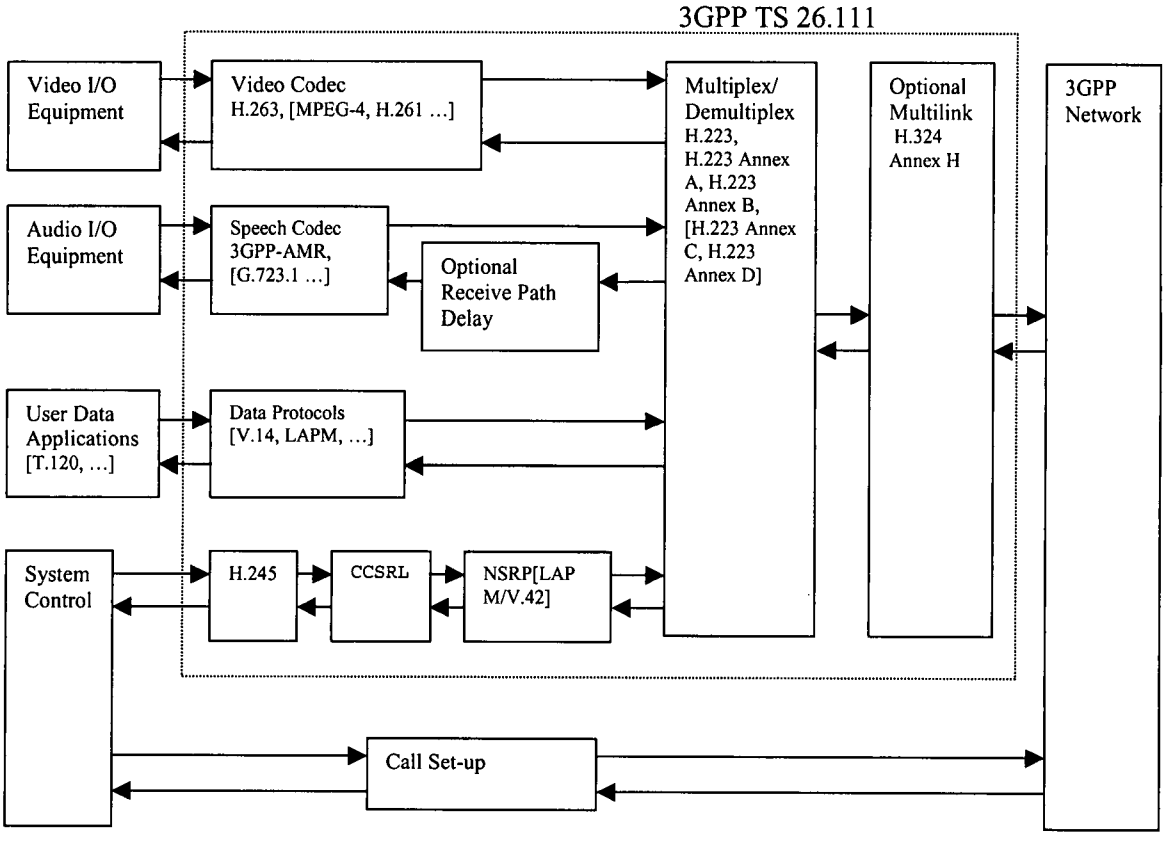

Figure 3.1 Block diagram of 3G-324M system (3GPP, 2004)

3.1 H.223 Multiplexing and Demultiplexing

3G-324M defines the following four levels of error-resilience for H.223 transport (NMS communications, 2006):

3.1.1 Level 0

Level 0 corresponds to baseline H.223. It offers little or no error resilience services. In baseline H.223 high-level data link controller (HDLC) performs the framing of multiplexed data. There are 16 multiplexing patterns to assemble media, control, and data packets in level 0. The endpoints negotiate these patterns between them. In level 0, bit errors can break the HDLC protocol and can interfere with bit stuffing. Bit errors can also corrupt framing flags (resulting in lost or concatenated multiplexed packets), and may cause flag emulations in the payload due to short length of framing flag. Flag emulation may break the multiplexed packet at the incorrect position (CommsDesign, 2003).

3.1.2 Level 1

Level 1 corresponds to annex A of H.223. It improves the MUX-PDU synchronisation over error prone channels by using a 16 bit pseudorandom noise (PN) sequence instead of 8 bit HDLC synchronisation flag. Due to its low probability the problem of this 16 bit PN flag in payload is almost negligible. But at the same time it has high probability of being corrupted due to its longer length (CommsDesign, 2003). The 3G-324M specifications define Annex A as a mandatory error resilience level for a light bit error prone channel.

3.1.3 Level 2

Level 2 corresponds to annex B of H.223. It adds to error resilience and synchronisation of Level 1 by introducing a MUX-PDU payload length information and FEC in the header. Level 2 is also mandatory error resilience level for 3G-324M and is intended for handling medium BER channels (Radvision, 2002).

3.1.4 Level 3

Level 3, corresponding to H.223 annex C and D, is optional level. It offers the best error resilient system. It offers adaptation layers and redundancy coding for highly error prone channels. It also includes forward error correction (FEC) and retransmission (ARQ) schemes (Smith and Jabri, 2004).

Within every level in H.223 there are adaptation, multiplexing, and demultiplexing layer. The following are the three adaptation layers (NMS communications, 2006):

a) Adaptation layer 1 (AL1)

Usually AL1 is used to transport user data and H.245 control messages. It is designed for data transfer. It does not handle or control errors and has to rely on upper layers for this purpose (NMS Communications, 2006).

b) Adaptation layer 2 (AL2)

AL2 is suitable adaptation layer for the transport of audio data. It offers 8-bit cyclic redundancy check (CRC) for error-detection and optional sequence numbering to allow loss detection. It can handle variable length AL service data units (SDUs) (NMS Communications, 2006).

Adaptation layer 3 (AL3)

AL3 is intended to be used for video. It offers 16-bit CRC for error detection and optional sequence numbering to detect missing and misdelivered AL-PDUs. It provides an optional retransmission procedure in addition to the capability to handle AL SDUs of variable length (CommsDesign, 2003).

3.2 H.245 Call Control

H.245 is basically a call control protocol. For reliability in error-prone environments, H.245 uses simple retransmission protocol (SRP) with an option of numbered simple retransmission protocol (NSRP) and offers a control channel segmentation and reassembly layer (CCSRL). H.245 requires mobile terminals to support SRP and NSRP. If both terminals start session in level 0, the system must operate in SRP. If terminals start the session at level 2, NSRP is used. CCSRL is used for carrying large H.245 packet for operation (Smith and Jabri, 2004). H.245 defines messages parameters using Abstract Syntax Notation 1 (ASN.1). Furthermore, the messages are binary encoded according to the packed encoding rule (PER) (NMS Communications, 2006).

During an H.245 conversation, the endpoints has to decide which one of them is master. The master terminal decides the conditions in case of conflict. Endpoints may have different capabilities regarding H.223 multiplexing/demultiplexing, video and speech codecs supported by them, data sharing, and other optional features (NMS Communications, 2006).

H.245 provides a mechanism by which endpoints can exchange capabilities and settle to a set of capabilities common to both.

The media and data flows are divided in logical channels. H.245 provides logical channel signalling to allow the open/close of logical channel and operations related to parameter exchange. This procedure also uses the exchange of parameter for the use of this logical channel (CommsDesign, 2006).

In H.245, the choice of audio and video codecs and their parameters are decided at the transmitter side on the basis of the capabilities that the receiver has sent. The receiver can

also signal a preference within its capability, to the transmitter via a mode request(NMS Communications, 2006).

Finally, H.245 has a range of call control commands and indications that it uses for flow control, video codec control, user input indications, jitter indication, and skew (NMS Communications, 2006).

H.245 uses the abstract syntax notation 1 (ASN.1) to define each message parameters that provides readability and extensibility effectively. To encode these ASN.1 messages into binary, the packed encoding rule (PER) is employed to ensure bandwidth effective message transmission (CommsDesign, 2003) .

3.3 Voice Channel

3G-324M mandates the use of adaptive multi-rate (AMR) codec as speech codec. G.723.1 is an optional legacy codec included in the 3rd Generation Partnership Project (3GPP) recommendation for compatibility with standards such as H.323 (3GPP, 2004).

3.4 Video Channel

H.263 baseline level 10 and MPEG-4 simple profile level 0 (ISO/IEC 14496-2) are mandatory and recommended video codecs for 3G-324M terminals (3GPP, 2004). H.263 is a legacy codec used by H.323 devices. However the error resilience and high efficiency make MPEG-4 more suitable for 3G-324M. It is also baseline compatible with H.263.

3G-324M is interoperable with other multimedia protocols like H.323 and SIP. A media gateway with transcoding functions is required for this purpose. Companies like NMS Communications have already developed such gateways.

4 3GPP video codecs

As mentioned in chapter 2, the 3G-324M standard specifies H.263 baseline level 10 as mandatory and MPEG-4 simple profile @ level 0 as recommended video codecs. This chapter describes some basics of video coding followed by an overview of both H.263 baseline level 10 and MPEG-4 simple profile @ level 0 video codecs.

4.1 Basics of video coding

The basic purpose of encoding video is to compress it. The compression is achieved by trying to remove the redundancies in the video. There are two types of redundancies present in video, namely, spatial and temporal. Spatial redundancy refers to the correlation present between different parts of a picture. Temporal redundancies are the redundancies present between consecutive frames. If there is not much motion present in the video, it will have high temporal redundancy as the successive frames will be very similar to each other (CiteSeer, 2006). A very high degree of compression is required for video 3GPP codecs as the maximum bandwidth allocated to video for video telephony is 42 kbps (MOTODEV, 2006). Therefore all of the 3GPP video codecs achieve the degree of compression required for video telephony by using both spatial and temporal redundancy reduction to reduce the bandwidth required by the video bitstream (3GPP, 2004).

To reduce spatial redundancy input signal is converted from the time domain to the frequency domain using Discrete Cosine Transform (DCT). The transform gives a DC value and other coefficients at various frequencies. Most of the scene information is concentrated in coefficients which correspond to the lower frequencies. These coefficients are then quantised such that most of them are encoded to zero. The non-zero coefficients are significant and occupy a very small range of values after encoding (ITU, 2005). After this, usually, all of the coefficients are re-arranged such that, the larger magnitude values will occur first followed by 0 value coefficients. Finally, the coefficients are replaced with a count of the run of coefficients with zero value followed by the value of a nonzero coefficient. This arrangement is translated into a variable length code (VLC) (3GPP, 2004). If this type of compression is applied to the entire frame, an intra coded frame is produced (CiteSeer, 2006).

Although intra coding is efficient, the degree of compression it achieves is much less than that required by 3G-324M. Therefore, to achieve a higher degree of compression, all the 3GPP video codecs also exploit temporal redundancy in the video. Motion estimation is used to reduce temporal redundancy in the video. The current frame is compared with the previous frame. A best match of a particular region in the current frame is found in the previous frame. A set of vectors (called motion vectors) are determined such that when they are applied to corresponding regions in the previous frame would create the current frame (3GPP, 2004). Mostly this match is not perfect; hence the difference in the pixel values of the two regions is calculated as error (called prediction errors). The DCT is then applied only to this error and the resulting coefficients are quantized. The quantized DCT coefficients, motion vectors and side information are then coded using VLCs (3GPP, 2004). Before quantised DCT coefficients are coded suing VLCs, reconstruction of current frame is required to provide reference frame for motion estimation. The values of pixels of an intra macroblock or the prediction errors of the inter macroblock are obtained by dequantising the

quantized DCT coefficients then applying the inverse DCT to the dequantized DCT coefficients. If the macroblock is an intra macroblock, the reconstructed macroblock will contain the pixel values. If the macroblock is an inter-block, the outputs of IDCT are prediction errors which are added to the pixel values of the macroblock in the previous frame used as reference frame to obtain the reconstructed-macroblock (ITU, 2005).

This method results in much higher compression efficiency than that obtained by coding an intra frame. A frame coded in this way is called inter coded frame. If the error for a particular region or block is too large then it can be encoded as an intra block (ITU, 2005).

4.2 H.263 baseline codec

H.263 is the ITU-T standard for video coding for low bitrate communication. H.263 baseline (i.e. without any annexes) codec is based on this standard. H.263 is a frame based codec Fitzek and Reisslein, 2001). Figure 4.1 shows the block diagram of H.263 baseline encoder. The description of this encoder has already been given in the previous section.

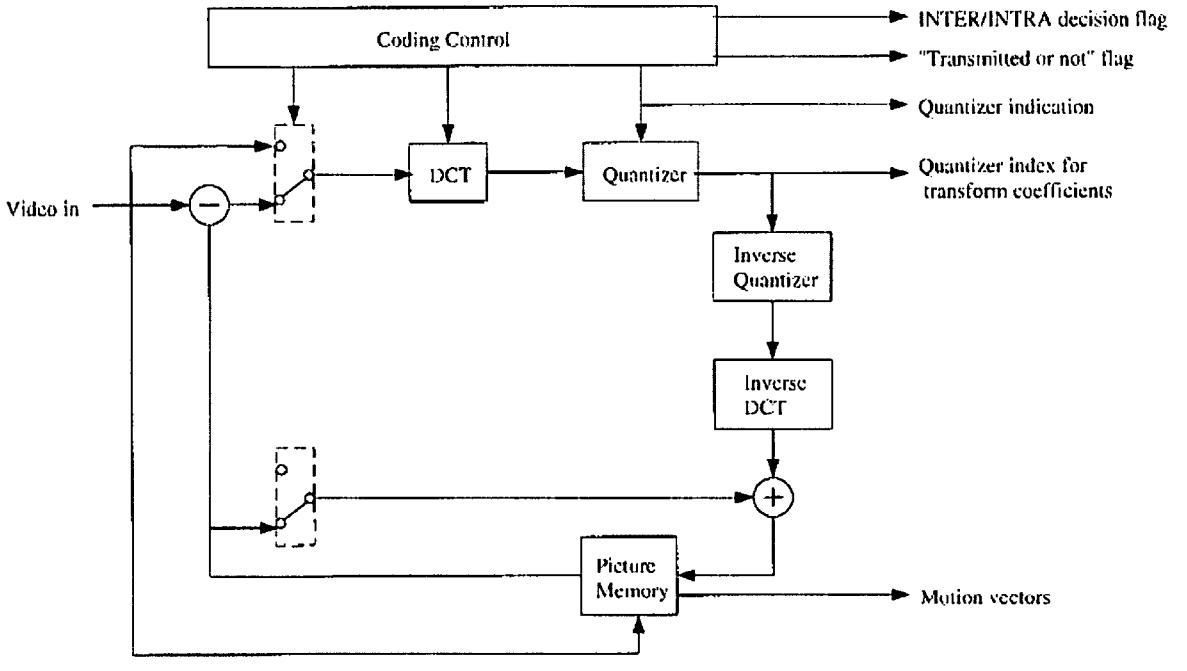

Figure 4.1 Block diagram of H.263 baseline encoder (Cote *et al.* 1998)

4.2.1 Frame structure of H.263 codec

According to ITU-T recommendation H.263, five picture sizes will be supported by H.263 codec. They are sub-QCIF, QCIF, CIF, 4CIF, and 16CIF. At these resolutions the luminance component of the picture is sampled. The chrominance components Cb and Cr are sampled one-half of these resolutions in both horizontal and vertical directions. Frame structure of H.263 codec for the QCIF resolution is shown in Figure 4.2. Each picture/frame is divided into macroblocks. A macroblock consists of 4 luminance blocks of 8 x 8 and one 8x8 block each of Cb and Cr. At QCIF resolution a GOB consists of one row of macroblocks. A frame consists of 9 GOBs at QCIF resolution.

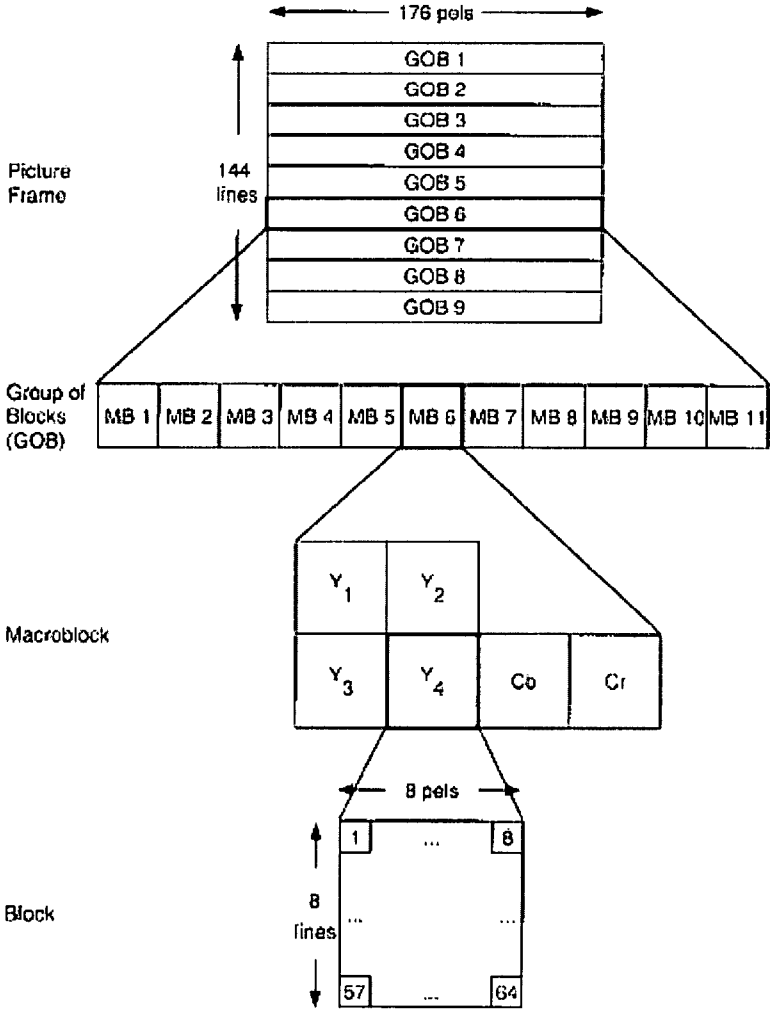

Figure 4.2 Frame structure of H.263 codec at QCIF resolution (Cote *et al.* 1998)

4.2.2 Coding Tools supported by H.263

H.263 uses motion estimation and compensation to remove temporal redundancy and DCT to remove spatial redundancy. It means that it supports both intra coded and inter coded frames/picture described in section 4.1. In addition to this H.263 supports the following optional coding modes: unrestricted motion vectors (Annex D), advanced prediction (Annex E), PB frames (Annex F), and syntax-based arithmetic coding (Annex G) (ITU, 2005).

The unrestricted motion vectors mode and advanced prediction mode improve inter picture prediction. With PB-frames mode picture rate can be increased without significantly increasing the bitrate. In syntax-based arithmetic coding mode, arithmetic coding is used instead of the default VLC coding. This results in slightly less bitrate. These optional modes improve the compression at the cost of complexity (cote *et al.* 1998).

4.2.3 H.263 baseline level 10

The 3G-324M standard specifies H.263 baseline level 10 as the mandatory video codec for 3G handset (3GPP, 2004). H.263 baseline codec does not include any annexes. Level 10 of H.263 supports QCIF and sub-QCIF resolutions, a maximum bitrate of 64 kbps and a picture decoding rate up to (15 000)/1001 (ITU, 2005).

4.3 MPEG-4 Visual

ISO 14496 MPEG-4 is a standard for storing and delivering multimedia content (TellaSonera, 2004). The three main parts to the MPEG-4 standard are (IndigoVision, 2004):

- ISO 14496-1: Systems
- ISO 14496-2: Visual
- ISO 14496-3: Audio

The MPEG-4 Visual standard was published in 1999 by International standard Organisation (ISO). It was targeted for very low bitrate coding. But in a few years, the range of its applications expanded from mobile phones to broadcasting (IndigoVision, 2004).

Unlike H.263, MPEG-4 video is an object based codec. In MPEG-4 terminology, a scene is composed of Video Objects (VOs). Each VO is coded individually. For very simple scenes like in mobile video applications, the whole scene is defined as one VO. Each VO may consist of various scalability layers called Video Object Layers (VOLs). Each VOL consists of a series of snapshots in time, called Video Object Planes (VOPs) which are equivalent of frames (Fitzek and Reisslein, 2001). For each VOP the MPEG-4 encoder shown in figure 4.3 processes the shape, motion, and texture characteristics. The shape information is encoded by defining a block around a VO. The block is divided in to small equally sized squares of 8x8 or 16 x 16 pels called macro blocks (MBs) (TellaSonera, 2004). Based on its location, each MB is classified as lying (1) inside the object, (2) on the border of the object, and (3) outside the object but inside the bounding block. The MBs located at the border of the object are shape coded. Like H.263, the texture coding is done on a single block basis. For an intra

coded (I) VOP, DCT is applied on the absolute texture values in each MB (Fitzek and Reisslein, 2001). The DCT coefficients are then quantized and variable length coded. In forward predicted (P) VOPs, a MB is predicted from the best matching MB in the previous I or P VOP, as in H.263. In Bi-directionally predicted (B) VOPs a MB is predicted from the previous I (or P) and succeeding P (or I) VOP. The errors called prediction errors are DCT coded, quantized and variable length coded ((Fitzek and Reisslein, 2001).

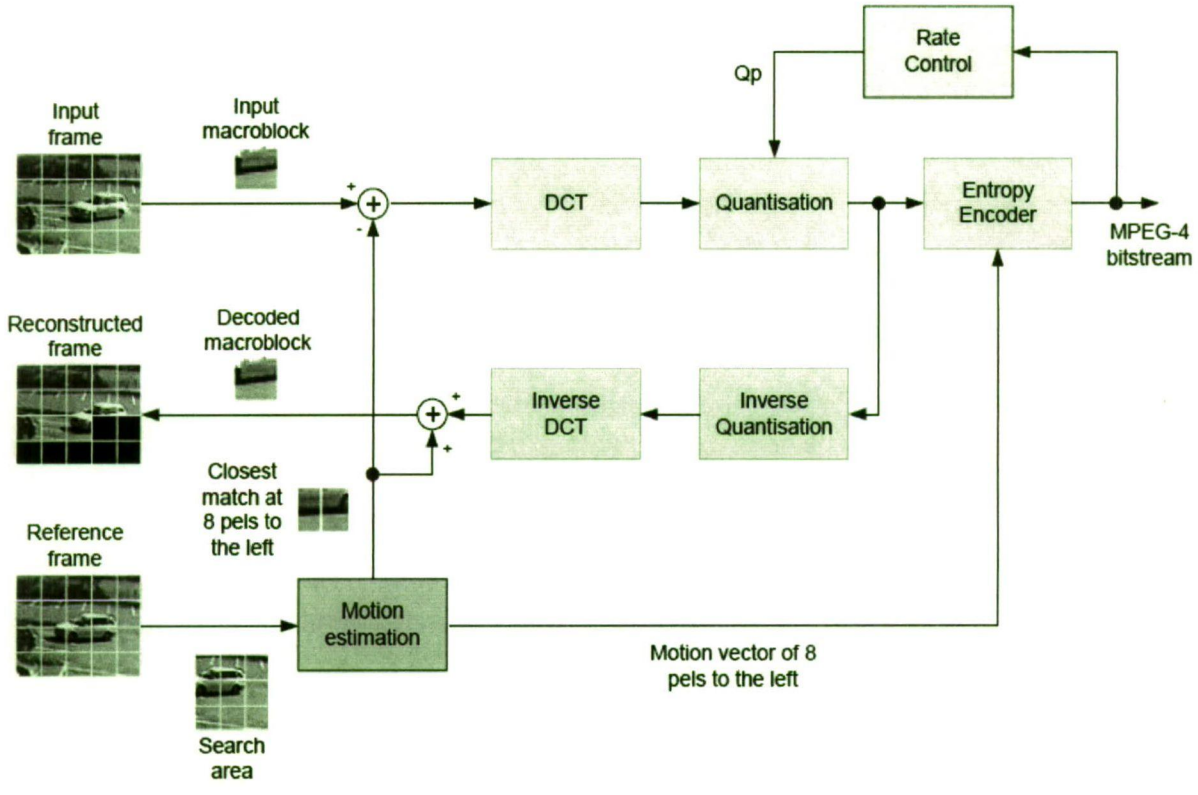

Figure 4.3 Block diagram of MPEG-4 encoder (IndigoVision, 2004)

The MPEG-4 decoder is identical to the reverse path shown in green in figure 4.3. With the exception that the bitstream is first passed through an entropy decoder before passing the data to inverse quantisation unit (IndigoVision, 2004).

4.3.1 MPEG-4 profiles and levels

The profiles in MPEG-4 define a set of tools that are used with content which conforms to that profile (TellaSonera, 2004). In other word, the division to profiles permits the implementation of only those tools that are necessary for that task, as it would be very expensive to design a decoder that implements every tool. Another purpose for defining profiles to make sure that implementations from various parties can interoperate. The most basic profile is "Simple" profile and is suitable for very low bitrate applications like mobile communication. The tools contained in the simple profile are basic, Error resilience and short header (equivalent to H.263 baseline) (IndigoVision, 2004). Each profile is further sub-divided in to levels. A level determines the level of complexity of tools used on that level (TellaSonera, 2004). Table 4.1 shows the MPEG-4 Visual Profiles and Levels (MPEG-4 Industry Forum, 2006).

Table 4.1 / MPEG-4 Visual Profiles and Levels

Visual profile	Level	Typical visual session size	Max. number of objects	Maximum number objects per type	Max. unique quant. tables	Max. VCV buffer size (MB)	VCV decoder rate (MB/s)	Max. total VBV buffer size (units of 16384 bits)	Max. VOL VBV buffer size (units of 16384 bits)	Max. video packet length (bits)	Max. sprite size (MB units)	Max. bitrate (kbit/s)
Simple	L0	QCIF	1	1 x Simple	1	99	1485	10	10	2048	N. A.	64
Simple	L1	QCIF	4	4 x Simple	1	99	1485	10	10	2048	N. A.	64
Simple	L2	CIF	4	4 x Simple	1	396	5940	40	40	4096	N. A.	128
Simple	L3	CIF	4	4 x Simple	1	396	11880	40	40	8192	N. A.	384
Advanced Real Time Simple	L1	QCIF	4	4 x Simple or Adv. Real Time Simple	1	99	1485	10	10	8192	N. A.	64
Advanced Real Time Simple	L2	CIF	4	4 x Simple or Adv. Real Time Simple	1	396	5940	40	40	16384	N. A.	128
Advanced Real Time Simple	L3	CIF	4	4 x Simple or Adv. Real Time Simple	1	396	11880	40	40	16384	N. A.	384
Advanced Real Time Simple	L4	CIF	16	16 x Simple or Adv. Real Time Simple	1	396	11880	80	80	16384	N. A.	2000
Simple Scalable	L1	CIF	4	4 x Simple or Simple Scalable	1	495	7425	40	40	2048	N. A.	128
Simple Scalable[3]	L2	CIF	4	4 x Simple or Simple Scalable	1	792	23760	40	40	4096	N. A.	256
Core	L1	QCIF	4	4 x Core or Simple	4	198	5940	16	16	4096	N. A.	384
Core	L2	CIF	16	16 x Core or Simple	4	792	23760	80	80	8192	N. A.	2000
Advanced Core	L1	QCIF	4	4 x Core or Simple or Adv. Scalable Texture	4	198	5940	16	8	4096	N. A.	384
Advanced Core	L2	CIF	16	16 x Core or Simple or Adv. scalable Texture	4	792	23760	80	40	8192	N. A.	2000
Core Scalable	L1	CIF	4	4 x Core or Simple or Core scalable or Simple	4	792	14850	64	64	4096	N.A.	768

Visual profile	Level	Typical visual session size	Max. number of objects	Maximum number objects per type	Max. unique quant. tables	Max. VCV buffer size (MB)	VCV decoder rate (MB/s)	Max. total VBV buffer size (units of 16384 bits)	Max. VOL VBV buffer size (units of 16384 bits)	Max. video packet length (bits)	Max. sprite size (MB units)	Max. bitrate (kbit/s)
				Scalable								
Core Scalable	L2	CIF	8	8 x Core or Simple or Core scalable or Simple	4	990	29700	80	80	4096	N.A.	1500
Core Scalable	L3	CCIR601	16	16 x Core or Simple or Core scalable or Simple Scalable	4	4032	120960	80	80	16384	N. A.	4000
Main	L2	CIF	16	16 x Main or Core or Simple	4	1188	23760	80	80	8192	1584	2000
Main	L3	CCIR 601	32	32 x Main or Core or Simple	4	3240	97200	320	320	16384	6480	15000
Main	L4	1920 x 1088	32	32 x Main or Core or Simple	4	16320	489600	760	760	16384	65280	38400
Advanced Coding Efficiency	L1	CIF	4	4 x Adv. Coding Efficiency or Core or Simple	4	792	11880	40	40	8192	N. A.	384
Advanced Coding Efficiency	L2	CIF	16	16 x Adv. Coding Efficiency or Core or Simple	4	1188	23760	80	80	8192	N. A.	2000
Advanced Coding Efficiency	L3	CCIR 601	32	32 x Adv. Coding Efficiency or Core or Simple	4	3240	97200	320	320	16384	N. A.	15000
Advanced Coding Efficiency	L4	1920 x 1088	32	32 x Adv. Coding Efficiency or Core or Simple	4	16320	489600	760	760	16384	N. A.	38400
N-Bit	L2	CIF	16	16 x Core or Simple or N-Bit	4	792	23760	80	80	8192	N. A.	2000

4.3.2 Simple profile

The profile and level of MPEG-4 codec recommended by 3G-324M is simple profile @ level 0. Although having low complexity, Simple profile supports only I and P VOPs and provides error resilience through data partitioning, Reversible Variable Length Coding (RVLC), resynchronisation marker and header extension code (3GPP, 2004).

a) Data partitioning

It separates DCT coefficients and motion vector information by markers so that errors in one do not affect the other. For instance, if errors are found only in the DCT coefficients of a macro-block, the macro-block can be reconstructed using the correct motion information while concealing the errors detected in the DCT coefficients. This results in a higher video quality of the decoded frame than replacing that macroblock with the corresponding macroblock in the previous frame (NMS Communications, 2006).

b) Reversible variable length coding

Reversible variable length coding allow for the decoding of block of data starting from its beginning (forward) or from its end (backward). This increases the probability of correct decoding of a part of corrupted data (NMS communications, 2006).

c) Resynchronization markers

Resynchronization markers are codes that are inserted in the bit steam so that the decoder can resynchronize the decoding process.

d) Header extension codes

Header extension codes increase the effectiveness of resynchronization of the decoding process by including the timing information in resynchronization marker.

4.3.3 Level 0

Level 0 was not included in the original MPEG-4 visual standard. It was included in MPEG-4 on the recommendation of 3GPP 2^{nd} extension to the 2^{nd} edition of MPEG-4 visual standard (MPEG-4 Industry Forum). Level 0 limits the maximum bitrate to 64kbps and allows only one object in the MPEG-4 bitstream. The following are the additional restrictions applied for simple profile @ level 0 to limit the complexity of MPEG-4 codec for 3G-324M terminals ((MPEG-4 Industry Forum):

1. The maximum frame rate shall be 15 frames/second
2. The maximum f_code shall be 1

3. The intra_dc_vlc_threshold shall be 0
4. The maximum horizontal luminance pixel resolution shall be 176 pels/line
5. The maximum vertical luminance pixel resolution shall be 144 pels/VOP
6. If AC prediction is used, the following restriction applies : QP (quantisation parameter) value shall not be changed within a VOP (or within a video packet if video packets are used in a VOP). If AC prediction is not used, there are no restrictions to changing QP value.

5 FFmpeg and Telenor H.263 codec

To be used as a part of video quality measurement test platform Asterisk should be able to act as 3G-324M terminal which implies that it must support 3G-324M protocol and audio and video codecs mandated and recommended by 3G-324M standard. Given that the support for 3G-324M protocol and AMR codec has been added in to Asterisk; by incorporating video codecs mandated and recommended by 3G-324M (i.e. H.263 baseline level 10 and MPEG-4 simple profile @ level 0 codecs into Asterisk), Asterisk can be used as a component of video quality test platform for 3G handset. Open source FFmpeg libavcodec is believed to support both MPEG-4 and H.263 codecs. Moreover, Telenor H.263 codec is also freely available. This chapter evaluates capabilities of MPEG-4 and H.263 codecs in FFmpeg libavcodec and Telenor H.263 codec, and their suitability for adding in to Asterisk®. The chapter is organised into 4 sections. Section 5.1 gives a description of the action plan to incorporate the above mentioned video codecs in Asterisk. Section 5.2 introduces FFmpeg libavcodec and evaluates its capability to decode and encode to MPEG-4 and H.263 bitstreams traced from a video call. Section 5.3 gives an introduction of H.263 codec and examines its capability to decode and encode to H.263. Finally the suitability of these two software packages for incorporating in to Asterisk is discussed in section 5.4.

5.1 Action plan

The proposed step-by step approach to incorporate video codecs, specified by 3GPP for circuit-switched video telephony, in Asterisk is given below:

Step-1: Get the H.263 and MPEG-4 video trace files (or bitstream) during a 3G video call.

Note-1: The pre-requisite for step-1 is that Asterisk must support 3G-324M protocol so that video bitstreams can be traced out from H.223 channel during a video call.

Note-2: Unless otherwise specified, H.263 and MPEG-4 here and in the rest of this chapter refer to H.263 baseline (profile 0) level 10 and MPEG-4 simple profile @ level 0 respectively.

Step-2: Select, download and install the available free software package and open source codec for decoding and encoding of MPEG-4 and H.263 video trace files.

Step-3: Perform offline decoding of the traced H.263 and MPEG-4 bitstream to YUV format with the selected H.263/MPEG-4 decoder. Verify that the decoded file can be played in a YUV player[1]. If the decoded YUV file can be played in a YUV player, go to next step. Otherwise go back to step-2 (i.e. select another decoder and encoder). A video sequence or file which has been decoded from an encoded file is called reconstructed sequence.

Step-4: Encode, offline, the reconstructed YUV video file back to MPEG-4/H.263 bitstream with the corresponding MPEG-4/H.263 encoder. Check if the encoded bitstream can be played in a Motorola's 3G handset[2]. If yes, go to next step. Otherwise go back to step-2.

[1] The YUV player can be freely downloaded from http://www.yuvplayer.com.

[2] This test is performed as follows. A video call is made from a 3G handset to Asterisk and then the recorded video clip is sent to3G handset. If the handset can play the video, the video clip is playable and vice versa. However, this test assumes that there is nothing wrong with the wireless channel and the 3G handset.

Step-5: Make modifications (if necessary) to the free software package/open source codec to make it compatible with the format of codecs already in Asterisk PBX and incorporate it in Asterisk PBX.

What follows is a description of the FFmpeg-libavcodec and Telenor's TMN (H.263) encoder and decoder and the reason why these were selected to be used in this project.

5.2 FFmpeg-libavcodec

FFmpeg is an open source project licensed under the GNU Library General Public License (FFMPEG, 2006). This implies that its source code can be incorporated in commercial products/programs. It was developed in Linux but can also be compiled under Windows and Mac OS X. FFmpeg is a trademark of Fabrice Bellard who is the originator of the FFmpeg project. The most prominent features of FFmpeg include the following (FFMPEG, 2006):

- It has a command line utility called *ffmpeg* which can convert between various audio and video file formats. It can also convert audio and video files from one sample rate to another and encode to a target bit-rate specified by the user.
- FFmpeg contains a library *libavcodec* which is the leading open source audio and video codec library. It deals with decoding raw audio and video streams and can decode most multimedia formats faster than the available alternatives (The MPlayer Project, 2006).
- FFmpeg's library *libavformat* handles parsing and generation of almost all commonly used audio and video file formats (FFmpeg, 2006). In other words *libavformat* deals with parsing video files and separating the streams contained in them.

5.2.1 Why FFmpeg/libavcodec?

At the start of the project an MPEG-4 simple profile @ level 0 video clip recorded by a 3G handset was provided by Motorola. It was found that the above video clip can be played in MPlayer which is an open source movie player for Linux and uses FFmpeg's *libavcodec* as its default codec library (The MPlayer Project, 2006). This seemed to be suggesting that FFmpeg's *libavcodec* supports MPEG-4 simple profile @ level 0 codec. This was the reason for the selection of FFmpeg/libavcodec to be used for decoding and encoding of MPEG-4 simple profile @ level 0 video clip.

5.2.2 Downloading FFmpeg

There are both formal releases and SVN repository for FFmpeg. SVN is an abbreviation of Subversion which is an open source application for version/revision control (Wikipedia, 2006). The latest released version of FFmpeg at the time of writing is 0.4.9-pre1. Following are some links from where the latest released version of FFmpeg along with some required patches can be downloaded:

http://www.at.linuxfromscratch.org/patches/downloads/ffmpeg/

http://www.ca.linuxfromscratch.org/patches/downloads/ffmpeg/

http://www.nl.linuxfromscratch.org/patches/downloads/ffmpeg/

There are also many other links for downloading FFmpeg but it has been found that they generally do not contain the patches required for successful installation of FFmpeg (e.g. patches for compiling ffmpeg-0.4.9 under gcc-3.4 and gcc-4). However the team maintaining the FFmpeg project discourages the use of released versions.

Since FFmpeg is an open source project and developers from all over the world are contributing in it and adding new features in it constantly, it is recommended to download the FFmpeg from ffmpeg SVN repository hosted by the Mpalyer project to exploit all the latest features (FFMPEG, 2006). It has been found that SVN snapshots of FFmpeg can be compiled and installed without any problem.

To get FFmpeg via Subversion, login to your Linux machine as *root* user and simply issue the following command at the command line in console (FFMPEG, 2006):

```
svn checkout svn://svn.mplayerhq.hu/ffmpeg/trunk ffmpeg
```

A directory named *ffmpeg* containing the source code will be created in the current directory. The source code can be later updated by simply issuing the command *svn update* from within the *ffmpeg* directory. The configuration of FFmpeg and versions of *libavutil, libavcodec* and *libavformat* used in this project are given below:

Configuration: --enable-shared

Version of libavutil: 49.0.0

Version of libavcodec: 51.11.0

Version of libavformat: 50.5.0

The version of GNU C compiler used is gcc-4.0.2.

5.2.3 Installing FFmpeg

Installation of FFmpeg is straight forward procedure given below:

Issue the command `./configure` in the directory *ffmpeg* (containing the source code of FFmpeg). This will configure the FFmpeg with the default settings. Use the command `'./configure --help'` to see the configuration options to configure FFmpeg with customised settings.

After successful execution of the command `./configure` type `make` in the directory *ffmpeg* to build FFmpeg. Type `gmake` instead of `make` on BSD systems but make sure that `GNU make` is installed before running the command `gmake`.

Finally, type `make install` to install ffmpeg in the directory */usr/local/bin*.

5.2.4 Generic syntax of using FFmpeg

The generic syntax used to convert a file from one video format to another is (FFmpeg, 2006):

```
ffmpeg [[input-file options] ['-i' input-file]] [[output-file options]
[output-file]
```

If no parameters are given for output file, FFmpeg will use the same parameters for output file as given for input file.

5.2.5 Decoding MPEG-4 simple profile level 0 video with FFmpeg

The command `ffmpeg -formats` displays all (i.e. audio, image and video) file formats and codecs supported by FFmpeg. A detailed list of all audio, image and video codecs and file formats supported by FFmpeg, at the time of writing, is given in Appendix B2. According to the output of the command `ffmpeg -formats`, FFmpeg can not decode the MPEG-4 simple profile @ level 0 bitstream having .bits extension. Experiments also show that the CLU (command line utility) of FFmpeg (i.e. source file *ffmpeg.c*) does not recognise video bitstreams (i.e. video files with .bits extension). For instance, when an attempt was made to decode MPEG-4 simple profile @ level 0 video bitstream (traced from a 3G video call) to YUV format using the following command line parameters:

```
ffmpeg -s qcif -vcodec mpeg4 -i videotrace.bits -s qcif -vcodec rawvideo
decoded_videotrace.yuv
```

where `videotrace.bits` is the input MPEG-4 bitstream and `decoded_videotrace.yuv` is the decoded output video file in YUV format, FFmpeg reported the following output:

```
FFmpeg version SVN-rUNKNOWN, Copyright (c) 2000-2004 Fabrice Bellard

  configuration:  --enable-shared

  libavutil version: 49.0.0

  libavcodec version: 51.11.0

  libavformat version: 50.5.0

  built on Sep 18 2006 20:41:51, gcc: 4.0.2 20050901 (prerelease) (SUSE
Linux)

videotrace.bits: Unknown format
```

The above output seems to be suggesting that FFmpeg is not able to recognise (and hence decode) the video bitstreams with .bits extension. But when an identical command was given to decode the traced H.263 baseline level 10 bitstream (discussed later in this chapter), FFmpeg recognised it and generated a decoded file. This means that FFmpeg does recognise the bitstreams with .bits extension but it can not understand the format of MPEG-4 simple profile @ level 0 bitstream.

According to Bohme (2004), "*libavformats* deals with parsing video files and separating the streams contained in them and *libavcodec* deals with decoding raw audio and video bitstreams". This seemed to be suggesting that raw MPEG-4 video bitstreams might be decoded if we use *libavcodec* directly without using *libavformat*. Fortunately, there is an example program file, named `apiexample.c`, available in the folder *libavcodec* (within the directory *ffmpeg*) which illustrates how to use *libavcodec* in your program. It was found that with certain modifications, the file `apiexample.c` can decode the video bitstream to raw pgm (portable gray map) images (rather than a video file). The C code for the modified file `apiexample1.c` that can decode MPEG-4 video bitstreams with .bits extension to pgm format is given in Appendix C1 and also in the folder *Analysis-SW* on the disc provided with this thesis. The decoded pgm images are also provided on the disc in the folders *dec-vdo-mpeg4*. It might also be possible to decode video bitstreams to standard YUV format as well with some more modifications in `apiexample.c`. But no work was done on decoding MPEG-4 video bitstream to YUV format, as it was also found that FFmpeg can not encode YUV files to MPEG-4 simple profile @ level 0. Therefore even if decoding to YUV format was accomplished there was no use of it because FFmpeg can not encode the decoded (YUV) file back to MPEG-4 simple profile @ level 0 as discussed in the next section.

5.2.6 Encoding to MPEG-4 simple profile level 0 video with FFmpeg

To test if FFmpeg can encode to MPEG-4 simple profile @ level 0, three standard QCIF video sequences[3] (i.e. akiyo, carphone and foreman) in YUV formats were encoded to MPEG-4 simple profile @ level 0. The following command was used to encode the 'akiyo' video test sequence to MPEG-4. The same command was used for carphone and foreman test sequences.

```
ffmpeg -s qcif -vcodec rawvideo -i akiyo_qcif_ori90.yuv -s qcif -vcodec
mpeg4 -b 40 -bt 2 -r 10 akiyo.m4v
```

The table 5.1 gives general description of various encoding parameters used in the above command. A list of complete FFmpeg options and their usage is given in Appendix B1.

To encode the video file to MPEG-4 simple profile @ level 0, the frame size for output file has been set to QCIF using the `-s` option. The frame size for input sequence is also set to QCIF, as input YUV sequence is also QCIF.

Video codec is set to `rawvideo` for input file and to `mpeg4` for output file, using option `-vcodec`.

The format for output file is '.m4v'. According to the output of the command `ffmpeg -formats`, .m4v format is raw MPEG-4 video only format.

The maximum possible data rate used on the circuit switched radio access bearer is 64 kbps (MOTODEV, 2006). This includes bit-rates for audio plus video plus 3G-324M protocol overhead. Hence a total bit rate of 64 kbps allocates 42 kbps to video, 12 kbps to audio, and 10 kbps to 3G-324M protocol overhead. Therefore, the target bit-rate for output file is set to 40 kbps (using option `-b`) with a tolerance of 2 kbps (set using option `-bt`) allowing a maximum target bit-rate of 42 kbps.

Since the maximum frame rate allowed for MPEG-4 simple profile @ level 0 is 15 (MPEG-4 Industry Forum, 2006), the frame rate for output video is set to 10 frames per second using option `-r`.

[3] The standard QCIF video sequences saved in YUV format were downloaded from http://eeweb.poly.edu/~yao/VideobookSampleData/doc/sample/H263.htm. Each test sequence consists of 90 frames with a frame rate of 30 frames per second.

Table 5.1 / Description of parameters used to encode a YUV file to MPEG-4 using FFmpeg

Encoding parameter	Description
`-s`	Used to set frame size for: Input file, if precedes the input file name, and output file, if precedes the output file name. The format is 'width*height'. However the following abbreviations are also recognised by FFmpeg: 'sqcif' for '128*96', 'qcif' for '176*144', 'cif' for '352*288' and '4cif' for '704*576'. The default frame size is '160*128'.
`-i`	Used to specify the name and location of input file
`-vcodec`	Used to specify the codec for: input file, if precedes the input file name; output file, if precedes the output file name.
`-b`	Used to set the bit-rate for output video file in Kilo bits/second[4]. The default is 200 kilobits/second.
`-bt`	Used to set the tolerance in bit-rate for output video file in kbps.
`-r`	Sets frame rate in frames per second. The default is 25 frames per second.

On issuing the above command

```
ffmpeg -s qcif -vcodec rawvideo -i akiyo_qcif_ori90.yuv -s qcif -vcodec
mpeg4 -b 40 -bt 2 -r 10 akiyo.m4v
```

FFmpeg generates the following output:

```
FFmpeg version SVN-rUNKNOWN, Copyright (c) 2000-2004 Fabrice Bellard
  configuration:  --enable-shared
  libavutil version: 49.0.0
  libavcodec version: 51.11.0
  libavformat version: 50.5.0
```

[4] The FFmpeg documentation says that '–b' and '-bt' set the bit-rate and bit-rate tolerance respectively in bits per second whereas the help command in FFmpeg (i.e. 'ffmpeg -h') shows that '–b' and '–bt' set the bit-rates in kilo bits/second. From the experiments, it was found that the latter is true. This can also be seen in the output of the command used to convert a YUV file to raw MPEG-4 format (i.e. '.m4v').

```
  built on Sep 18 2006 20:41:51, gcc: 4.0.2 20050901 (prerelease) (SUSE
Linux)

Input #0, rawvideo, from 'akiyo_qcif_ori90.yuv':

  Duration: N/A, bitrate: N/A

  Stream #0.0: Video: rawvideo, yuv420p, 176x144, 25.00 fps(r)

Output #0, m4v, to 'ffmpeg_akiyo.m4v':

  Stream #0.0: Video: mpeg4, yuv420p, 176x144, q=2-31, 60 kb/s, 10.00
fps(c)

Stream mapping:

  Stream #0.0 -> #0.0

Press [q] to stop encoding

frame=   37 q=24.8 Lsize=      18kB time=3.7 bitrate=  40.6kbits/s

video:18kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

The output shows that the number of frames in encoded video is 37. The frame-rate for input sequence was 30 fps (although FFmpeg perceived it as 25 fps) and we decreased it to one-third (i.e. 10 fps) for the output video. Out of 90 frames in the original sequence only 37 has been encoded and the remaining frames have been skipped by FFmpeg.

The parameter `q` is the quantization step size. Its value ranges from 1 to 31. The lower the value of `q`, the better is the quality of encoded video (FFmpeg Documnetation, 2006). A value of 24.8 for `q` indicates that the quality of encoded MPEG-4 video is not very good. For a given bit-rate, the quality of encoded video can be increased by decreasing the frame rate because this results in the increased picture (or frame) quality. But in this particular case, the frame rate can not be decreased further as it is already low enough and if it is decreased further, the output video will have jerkiness (i.e. perception of still images in a video).

The time of encoded video is 3.7 seconds whereas that of original sequence was 3 seconds.

The achieved bit-rate for output encoded video is 40.6 kbps.

Table 5.2 shows the number of frames in the output sequence, quantization step size, play time and achieved bit-rate for the three encoded videos obtained by encoding standard QCIF YUV test sequence namely akiyo, carphone and foreman. The quality of video (indicated by 'q') and achieved bit-rate depends on the motion and spatial details in original test sequences. The reason why the quantisation step size and achieved bit-rate are different for the three MPEG-4 videos can be discussed in detail but it would be beyond the scope of this thesis.

Table 5.2 / Achieved quality and bit-rate for FFmpeg encoded MPEG-4 video sequences

MPEG-4 encoded sequence	Number of encoded frames	Quantization step size (q)	Play time of encoded video in seconds	Achieved bit-rate in kbps
Akiyo.m4v	37	24.8	3.7	40.6
Carphone.m4v	37	11.2	3.7	31.7
Foreman.m4v	37	24.8	3.7	35.5

It was, however, found that none of these files can be played in the 3G handset which seems to be suggesting that MPEG-4 encoder in FFmpeg is different from the one in 3G handset.

What follows is a detailed description of the differences found between the two encoders after analysis of the MPEG-4 files encoded by FFmpeg; and the modifications made in FFmpeg source code to bring the MPEG-4 encoder in FFmpeg equivalent to the one in 3G handset.

a) Profile and Level

Both the MPEG-4 traced file from a video call and the MPEG-4 file encoded by FFmpeg were viewed in 'XV Text Viewer' in Linux SUSE 10. As a reference, the mpeg4 files encoded by FFmpeg (i.e. akiyo.m4v, carphone.m4n and foreman.m4v) are given in the folder *enc-vdo-mpeg4* on the disc provided. It was found that unlike traced[5] MPEG-4 file (which was simple profile @ level 0), the MPEG-4 file encoded by FFmpeg was simple profile @ level 1 as indicated by the Profile and Level Indication byte/field in the MPEG-4 bit-stream. This is the byte immediately following the 32-bit Visual Object Sequence Start Code (i.e. the fifth byte in the MPEG-4 bit-stream).

After more experimentation with FFmpeg command line utility `ffmpeg.c`, looking closer at the output of the help command `ffmpeg -h` and exploring the source code of FFmpeg, it appeared that FFmpeg can encode to a certain profile and level of MPEG-4. The 'help' of FFmpeg shows that by including the parameters `-profile` and `-level` in the command line we can set the profile and level. But no information is available on what values should be entered in the command line for simple profile and level 0. To get this information source code had to be explored. Below is the section of source code from the file 'h263.c' in the folder *libavcodec* of FFmpeg that writes the Profile and Level Indication byte in the MPEG-4 bit-stream:

[5] The traced MPEG-4 file here refers to the file traced during a video call from one 3G handset to another 3g handset where as the encoded MPEG-4 file refers to the MPEG-4 file encoded by FFmpeg.

Folder: `libavcodec`

File: `h263.c`

Starting line: `2298`

```
static void mpeg4_encode_visual_object_header(MpegEncContext * s){
    int profile_and_level_indication;
    int vo_ver_id;
    if(s->avctx->profile != FF_PROFILE_UNKNOWN){
        profile_and_level_indication = s->avctx->profile << 4;
    }else if(s->max_b_frames || s->quarter_sample){
        profile_and_level_indication= 0xF0; // adv simple
    }else{
        profile_and_level_indication= 0x00; // simple
    }
    if(s->avctx->level != FF_LEVEL_UNKNOWN){
        profile_and_level_indication |= s->avctx->level;
    }else{
       profile_and_level_indication |= 1; //level 1
    }
```

It is obvious from the source code that if the user does not include the encoding parameters of profile and level in the command line of FFmpeg, by default it sets the profile either to Advanced Simple Profile or Simple Profile. If the user has set the encoder to use the B-frames or quarter-sample motion compensation the profile is set to Advanced Simple Profile otherwise it is set to Simple Profile. So to set the profile to Simple, a value of 0 should be used in the command line for `-profile`. It is also clear from the source code that if the user does not enter a value for the parameter 'level', it is set to level 1 by default. In other words if the level is unknown, it is set to 1. So to set the 'level' to 0, a value of 8 should be used for `-level` in the command line because a value of 8 for `-level` maps to level 0 of MPEG-4 Simple Profile (3GPP, 2004).

After above findings, the following command which includes settings for profile and level was used to encode akiyo test sequence to MPEG-4.

```
ffmpeg -s qcif -vcodec rawvideo -i akiyo_qcif_ori90.yuv -s qcif -vcodec
mpeg4 -profile 0 -level 8 -b 40 -bt 2 -r 10 akiyo.m4v
```

The encoded MPEG-4 file `akiyo.m4v` was also not playable in the 3G handset. On viewing this file in XV viewer, it was found that this file is similar to the file encoded without using 'profile' and 'level' options in the command line except that the profile and level byte/field was written according to the values set in the command line by the user.

It seems that although a value of 8 for the parameter '-level' (corresponding to level 0 of MPEG-4 visual simple profile) is accepted as an input at the command line, FFmpeg currently does not have the capability to respond to this input (i.e., it seems unable to encode to level 0 of simple profile).

b) Start codes and headers in bit-stream

To dig further in to it, a C program named 'mp4_sc.c' (given in Appendix C2 and in the folder *Analysis-SW* on the disc provided with this thesis) was written to analyse encoded as well as traced MPEG-4 files. This program reads the whole MPEG-4 file, displays it in hexadecimal format and also gives information about the various types of start codes present in MPEG-4 bit-stream. Table 5.3 compares the results of analysis of both the traced and encoded MPEG-4 files. The program `mp4_sc.c` gives information about all types of start codes that can be present in MPEG-4 visual bit-stream. However, for the ease of comparison, table 5.3 shows only those start codes that are present in either of the traced or encoded or both of the MPEG-4 files. It is quite obvious from Table 5.3 that user_data_start_code and group_of_VOP_start_code are present only in MPEG-4 files encoded by FFmpeg. The presence of these two start codes seems to imply that the corresponding headers are also present in the encoded MPEG-4 bit-streams. The traced MPEG-4 file did not contain any of the above two headers. The 3GPP recommendation TS 26.111 version 6.1.0 section 6.6 states that the Visual Object Sequence Header, Visual Object Header, and Video Object Layer Header shall be present in MPEG-4 simple profile @ level 0 bit-stream.

Table 5.3 / MPEG-4 start codes in either of the traced or encoded or both of the MPEG-4 files

Start codes in MPEG-4 bitstream	MPEG-4 File			
	Traced MPEG-4 file	Encoded MPEG-4 files		
	motorola_video_trace.bits	akiyo.m4v	carphone.m4v	foreman.m4v
video_object_start_code	1	4	4	4
video_object_layer_start_code	1	4	4	4
visual_object_sequence_start_code	1	4	4	4
user_data_start_code	0	4	4	4
group_of_VOP_start_code	0	4	4	4
visual_object_start_code	1	4	4	4
VOP_start_code	31	37	37	37

As an attempt to solve this problem the piece of code that writes the User Data and Group of VOP headers in the MPEG-4 bit-stream was searched and ignored. The source code along with its location in FFmpeg is given below. As a reference, both the traced and encoded MPEG-4 files have been given on the disc in the folders *trcd-vdo-mpeg4* and *enc-vdo-mpeg4* (respectively) provided with this thesis.

Folder: `libavcodec`

File: `h263.c`

Starting line: `2434`

```
/* user data */
    //if(!(s->flags & CODEC_FLAG_BITEXACT)){
      //  put_bits(&s->pb, 16, 0);
       // put_bits(&s->pb, 16, 0x1B2);    /* user_data */
       // ff_put_string(&s->pb, LIBAVCODEC_IDENT, 0);
    //}
```

Folder: `libavcodec`

File: `h263.c`

Starting line: `2455`

```
if(!(s->workaround_bugs & FF_BUG_MS))
            mpeg4_encode_gop_header(s);
```

After ignoring the piece of code that writes the user data and group of VOP headers in the MPEG-4 bit-stream, the akiyo test sequence was encoded to MPEG-4 using the command:

```
ffmpeg -s qcif -vcodec rawvideo -i akiyo_qcif_ori90.yuv -s qcif -vcodec
mpeg4 -profile -level 8 -b 40 -bt 2 -r 10 ffmpeg_ak_spl0.m4v
```

When encoded file `ffmpeg_ak_spl0.m4v` was viewed in XV viewer, it was found that encoded MPEG-4 bitstream now does not contain the unnecessary headers (i.e. the user data and group of VOP headers). But still the encoded file could not be played in 3G handset.

c) Restrictions of MPEG-4 simple profile @ level 0

As discussed in chapter 4, MPEG-4 visual simple profile @ level 0 has the following restrictions (3GPP, 2006):

1. The maximum frame rate shall be 15 frames per second.
2. The maximum horizontal luminance pixel resolution shall be 176 pels/line.
3. The maximum vertical luminance pixel resolution shall be 144 pels/VOP.
4. The maximum f-code value shall be 1.
5. The intra_dc_vlc_threshold shall be 0.

The FFmpeg command used to encode 'akiyo' test sequence to MPEG-4 already meets the first three requirements of MPEG-4 simple profile @ level 0 because we set the frame rate to 10 and set the horizontal resolution to 176 and vertical resolution to 144 (i.e. QCIF). For the last two requirements of simple profile @ level 0, the source code of FFmpeg was further explored and the following modifications were made.

The fourth requirement states that maximum f-code value should be 1. The f-code is an encoding parameter that specifies the motion vector search range and the number of bits that can be used to encode the motion vector. An f-code=1implies that the maximum search range is +/- 16 pixels, with a half-pixel resolution (Patent Storm, 2004). The intra_dc_vlc_ threshold is a 3 bit code that allows a mechanism to switch from DC coefficient VLC to AC coefficient VLC for coding of DC coefficients of intra macro block (ISO/IEC, 1999). On exploring the source code it was found that the maximum f-code has been set to 7. So, the maximum f-code was simply changed from 7 to 1 to fulfil the requirement of the level 0. The line of code that sets the f-code to 7 is given below.

Folder: `libavcodec`

File: `mpegvideo.h`

Line: `54`

```
#define MAX_FCODE 7
```

According to the fifth requirement the intra_dc_vlc_threshold must be 0. The line of code in FFmpeg that sets the value of intra_dc_threshold was found and ignored and a new line setting intra_dc_threshold forcefully to 0 was added.

Folder: `libavcodec`

File: `h263.c`

Starting Line: `5962`

```
//s->intra_dc_threshold= mpeg4_dc_threshold[ get_bits(gb, 3) ];
s->intra_dc_threshold= 0;
```

Now having done that apparently all the requirements of MPEG-4 simple profile level 0 were fulfilled but still the MPEG-4 file encoded in FFmpeg was not playable in 3G handset. The C program `mp4_sc.c` was again used to analyse the encoded files (i.e. ffmpeg_ak_spl0.m4v, ffmpeg_cp_spl0.m4v and ffmpeg_cp_spl0.m4v). Table 5.4 shows the abridged output of the program `mp4_sc.c`. Please note that the files ffmpeg_ak_spl0.m4v, ffmpeg_cp_spl0.m4v and ffmpeg_cp_spl0.m4v are the MPEG-4 files encoded in FFmpeg after making modifications in FFmpeg source code for the requirements of simple profile @ level 0. These files have been provided in the folder *enc-vdo-mpeg4* on the disc accompanying this thesis.

Table 5.4 / MPEG-4 start codes in the MPEG-4 file encoded after making modifications in FFmpeg for simple profile @ level 0

Start codes in MPEG-4 bit-stream	MPEG-4 File			
	Traced MPEG-4 file	MPEG-4 files encoded after making modifications in FFmpeg		
	normal_call._TX_muxvideo_1.bits	ffmpeg_ak_spl0.m4v	ffmpeg_cp_spl0.m4v	ffmpeg_cp_spl0.m4v
video_object_start_code	1	4	4	4
video_object_layer_start_code	1	4	4	4
visual_object_sequence_start_code	1	4	4	4
User_data_start_code	0	0	0	0
group_of_VOP_start_code	0	0	0	0
visual_object_start_code	1	4	4	4
VOP_start_code	31	37	37	37

It is worth noting that the number of video_object_start_code, video_object_layer_start_code, visual_object_sequence_start_code and visual_object_start_code in encoded files is 4 each whereas in traced file it is only 1 each. This suggested that there might be more objects in the encoded bit-stream than in traced bit-stream. This led to further investigation and it was found from the web site of MPEG-4 industry forum that there is one more requirement of simple profile level 0 that is not much talked about. This requirement states that the maximum number of objects in MPEG-4 simple profile @ level 0 bit-stream shall be 1 (M4IF, 2006).

Since the traced file is certainly MPEG-4 simple profile @ level 0 and must be fulfilling the above requirement (i.e. maximum number of objects in the traced bitstream is 1) and number of video_object_start_code, video_object_layer_start_code, visual_object_sequence_start_code and visual_object_start_code in it is 1 each, table 5.4 seems to be indicating that number of objects in the encoded bit-stream is 4. For level 1 of simple profile the maximum number of objects is 4 (M4IF, 2006). So despite all the modifications made in FFmpeg

source code for simple profile @ level 0[6], it seemed that FFmpeg was still encoding to simple profile @ level 1.

The above findings seemed to be suggesting that at the time of writing even with moderate modifications in its source code, FFmpeg can not encode to MPEG-4 simple profile @ level 0 video bitstream. To enable it to encode to simple profile @ level 0, either major modifications in the source code of MPEG-4 codec in FFmpeg are required and/or certain parts of the source code which are hard coded to encode to simple profile @ level 1, need to be rewritten. Due to the complexity of MPEG4 codec, this could be very time consuming and might change the scope of the project. FFmpeg is, therefore, not recommended to be used for encoding videos to MPEG-4 simple profile @ level 0.

5.2.7 Encoding H.263 profile 0 (baseline) level 10 with FFmpeg

To test if FFmpeg can encode to H.263 baseline level 10, three standard QCIF video sequences (i.e. akiyo, carphone and foreman) in YUV formats were encoded. What follows are the commands used to encode the akiyo, carphone and foreman video test sequences to H.263 baseline level 10 respectively; and the corresponding outputs generated by FFmpeg:

The following command was used to encode 'akiyo' to H.263 baseline level 10:

```
ffmpeg -s qcif -vcodec rawvideo -i akiyo_qcif_ori90.yuv -s qcif -vcodec
h263 -r 10 -b 40 -bt 2 ffmpg_ak_h263.h263
```

Output for 'akiyo':

```
frame=   37 q=6.3 Lsize=      21kB time=3.7 bitrate=  46.3kbits/s

video:21kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

Command used to encode the sequence 'carphone' to H.263 baseline level 10

```
ffmpeg -s qcif -vcodec rawvideo -i carphone_qcif_ori90.yuv   -s qcif -
vcodec h263 -r 10 -b 40 -bt 2 ffmpg_cp_h263.h263
```

Output for 'carphone':

```
frame=   37 q=24.8 Lsize=      15kB time=3.7 bitrate=  33.5kbits/s

video:15kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

Command used to encode the sequence 'foreman' to H.263 baseline level 10

```
ffmpeg -s qcif -vcodec rawvideo -i foreman_qcif_ori90.yuv  -s qcif -vcodec
h263 -r 10 -b 40 -bt 2 ffmpg_fm_h263.h263
```

[6] It was found that after making modifications in FFmpeg source code for simple profile level 0, FFmpeg may not function properly.

Output for 'foreman':

```
frame=   37 q=24.8 Lsize=      19kB time=3.7 bitrate=  41.9kbits/s
video:19kB audio:0kB global headers:0kB muxing overhead 0.000000%
```

The description of encoding options/switches used in the above commands is given in table 5.1. All of the above three H.263 files encoded by FFmpeg were playable in 3G handset and are given on the disc provided with this thesis in the folder *enc-vdo-h263/ffmpeg-enc-h263*. As we can see in the output of command used to encode 'akiyo' test sequence, the achieved bitrate is 46.3 kbps which is much more than the target bitrate of 40 ± 2 kbps. Using a fixed quantizer scale 'q' for encoding akiyo sequence seems to solve this problem. But a fixed 'q' results in an excessive bitrate of 52 kbps for the sequence 'carphone' which again exceeds the target bitrate. Hence, these experiments seem to be suggesting that although FFmpeg can encode to H.263 baseline level 10, it does not offer a very fine control over the bitrate.

5.2.8 Decoding H.263 profile 0 (baseline) level 10 with FFmpeg

To test if FFmpeg can correctly decode/reconstruct the H.263 encoded test sequences to YUV format, the commands given below were used.

```
ffmpeg -s qcif -vcodec h263 -r 10 -i ffmpg_ak.h263   -s qcif -vcodec
rawvideo -r 30 ffmpg_rc_ak.yuv


ffmpeg -s qcif -vcodec h263 -r 10 -i ffmpg_cp.h263   -s qcif -vcodec
rawvideo -r 30 ffmpg_rc_cp.yuv


ffmpeg -s qcif -vcodec h263 -r 10 -i ffmpg_fm.h263   -s qcif -vcodec
rawvideo -r 30 ffmpg_rc_fm.yuv
```

The YUV player can successfully playback all of the three reconstructed sequences. The above three decoded files have been provided on the disc (in the folder `dec-vdo-h263`) accompanying this thesis.

The traced H.263 video was decoded to YUV format using the following command:

```
ffmpeg -s qcif -vcodec h263 -r 10 -i videotrace-h263-0.bits -s qcif -
vcodec rawvideo -r 30 ffmpeg_trcdh263.yuv
```

The decoded file `ffmpeg_trcdh263.yuv` was also playable in 3G handset. However, FFmpeg reports a muxing overhead of 197.92 % in the output of this command. Since CLU of FFmpeg (i.e. *ffmpeg*) converts between file formats. One possible reason for this muxing overhead might be that FFmpeg perceives the input bitstream as a file format with multiplexed audio, video and control data and not as the raw video only bitstream. As a reference the decoded file `ffmpeg_trcdh263.yuv` has been provided on the disc (in the folder `dec-vdo-h263`) accompanying this thesis.

5.3 Telenor TMN-1.7 encoder and decoder

Test Model Near-term (abbreviated as TMN) was the name of the codec simulation model used during the development of a video compression standard for video telephony on normal analog telephone lines (lillevold, 2000). This standard is now called H.263. TMN decoder (called tmndecode) version 1.7 is a decoder and player for H.263 bitstreams. Likewise, TMN encoder (called tmn) version 1.7 is a very low bitrate video encoder that produces H.263 bitstreams. Both TMN-1.7 decoder and encoder are believed to be compatible with the ITU-T standard H.263 version 1, and were basically developed by Karl Olav Lillevod of Telenor R&D. Both TMN-1.7 encoder and decoder have been tested on Linux, SunOS-4.1.3 and Windows 95/NT. Since the source code and documentation of tmn and tmndecode is no longer available on Telenor's web site, all of the above information has been obtained from the 'readme' files in the folders of both tmndecode and tmn, version 1.7.

5.3.1 Why TMN encoder and decoder?

It was clear from the start of the project that every 3G-324M terminal must support H.263 baseline level 10 (3GPP, 2006). Now a tool in the MPEG-4 simple profile known as MPEG-4 short header is equivalent to baseline H.263 (IndigoVision, 2006). So it was not known until the very last stage of the project that whether Motorola's 3G handset supports H.263 baseline level 10 as a separate codec or as a tool/part of MPEG-4 visual (i.e. MPEG-4 short header). It became clear that 3G handset supports H.263 baseline level 10 as a separate codec when an H.263 baseline level 10 bitstream was traced (by Zhuoqun Li) during a video call. It was also discovered that source code of Telenor's H.263 baseline codec, called TMN, is available on the Internet. However it was not known at this point in time whether FFmpeg's *libavcodec* supports H.263 baseline level 10 or not. But it was preferred to use Telenor's H.263 codec due to the following reasons:

- Telenor's H.263 baseline codec is a stand alone codec whereas FFmpeg's *libavcodec* is a library and even if it supports H.263 baseline level 10, the source code of H.263 baseline needs to be extracted form this library in order to be incorporated to Asterisk leading to longer development time. Telenor's codec does not require this additional step.
- The Telenor's H.263 codec can be used more confidently as its command line utility is much shorter and simpler to use than that of FFmpeg. Moreover, the default values/positions of the options/switches of Telenor's codec are exactly known but in case of FFMPEG the default values/positions of every option/switch are not readily known.

5.3.2 Downloading TMN-1.7 encoder and decoder?

At the time of writing, TMN-1.7 encoder and decoder can be downloaded from the following link:

http://www.xs4all.nl/~roalt/h263.html

TMN-1.7 encoder and decoder have also been provided on the disc accompanying this thesis.

Version 2.0 of TMN encoder and decoder are available at the following website:

http://www.h263l.com/

The basic difference between the two versions is type of license under which they have been released. Version 1.7 was released under GNU General Public License whereas Version 2.0 was distributed under "Disclaimer of Warranty".

5.3.3 Installing TMN-1.7 encoder and decoder?

Installation of both TMN-1.7 encoder and decoder in Linux is very simple. The procedure of installation is same for both TMN-1.7 encoder and decoder. Just extract the compressed file in to the desired directory and issue the `make` command in that directory.

5.3.4 Using TMN-1.7 decoder and encoder

The generic syntax of usage of TMN-1.7 decoder (tmndecode) and encoder (tmn) is given below:

a) Generic syntax of usage of tmndecode

```
./tmndec [options] bitstream [outputfilename%]
```

where the options include setting the level of verbosity of the output and selecting the format for output decoded file etc. The output file name is followed by a '%' (percentage sign) if the output is in the form of decoded images each saved as a single file. If the output is a single file (as in case of YUV concatenated format) then output filename is not followed by a '%'. A detailed description of the use of various options used as decoding parameters is given in section 5.3.5.

b) Generic syntax of usage of tmn

```
./tmn [options] -i <filename> [more options]
```

where the options include selecting images at which to start and stop encoding, selecting the size of picture for encoded video, frame rate and target bitrate etc. The options used as encoding parameters have been discussed in detail in section 5.3.6.

5.3.5 Decoding H.263 baseline level 10 with TMN-1.7 decoder

As mentioned in the action plan in the beginning of this chapter, after obtaining the traced H.263 bitstream and installing the TMN-1.7 decoder and encoder, the traced h.263 video file was decoded to YUV format using tmndecode. Since this YUV file was obtained after decoding an encoded video file, it is in fact a reconstructed video sequence. Figure 4.1 shows what exactly a reconstructed video sequence is. While encoding a video sequence, an encoder can be set to encode only a fraction of total number of frames in the original sequence and skip the rest in order to meet a low target bitrate for the encoded video.

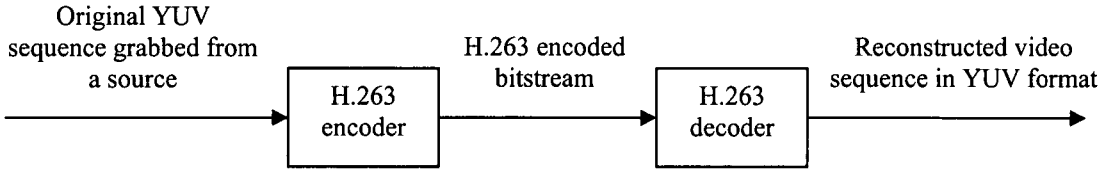

Figure 5.1 A reconstructed video sequence.

Quite obviously, in a reconstructed video sequence only the encoded frames (which might be only a fraction of the total number of frames present in the original sequence grabbed from source) are decoded/reconstructed. Therefore, the information contained by the skipped frames can not be completely recovered. In this case, the video quality of a reconstructed video sequence can never match that of the original video sequence.

The following command was used to decode the traced H.263 bitstream to concatenated YUV file format:

```
./tmndec -v1 -o5 videotrace-h263-0.bits tmnrvt.yuv
```

Table 5.5 gives the description of decoding options used to decode the traced h.263 file to YUV format.

Table 5.5 / Description of options used to decode traced h.263 bitstream to YUV format using TMN H.263 decoder

Decoding parameter	Description
`-v <n>`	Sets the level of verbosity of the output of TMN decoder (tmndecode). Increasing the value of n (where n is an integer) results in more detailed output. For instance, using –v2 will result in more detailed output than using –v1.
`-o <n>`	The value of n (where n = 0,1,2,...,6) specifies the output format. Use • `-o0` for obtaining decoded raw YUV images (where each image is stored as a separate file) • `-o1` for SIF (standard interchange format) • `-o2` for Truevision's TGA file format • `-o3` for PPM (portable pixel map file format where each image is stored as a separate file) • `-o4` for X11display • `-o5` for obtaining a single YUV concatenated file • `-o6` for Windows 95/NT display

Options other than those listed in table 5.5 were not used in the decoding command because their default values were suitable for our application.

The file 'tmnrvt.yuv' was playable in a YUV player without any problem. This verified that it has been correctly decoded by TMN deocder. It also means that unlike H.263 codec in FFmpeg, TMN-1.7 decoder can decode an H.263 bitstream encoded by the H.263 coder in 3G handset. This moved the project to next stage (i.e. encode the decoded/reconstructed YUV file back to h.263 bitstream using TMN-1.7 encoder and test if the encoded file is playable in a 3G handset).

5.3.6 Encoding to H.263 baseline level 10 with TMN-1.7 encoder

After decoding/reconstructing the h.263 video clip (traced during a video call) to YUV format using TMN decoder, the TMN encoder was used to encode it back to H.263. The following command was used in the directory '*tmn-1.7*' (i.e. TMN encoder's home directory) to test if Telenor's H.263 encoder can encode reconstructed YUV clip (named tmnrvt.yuv) back to h.263 profile 0 level 10:

```
./tmn -x 2 -a 0 -b 143 -s 15 -r 42000 -Z 30.0 -O 3 -i tmnrvt.yuv -B
enc_tmn_h263.bits
```

The encoded file `enc_tmn_h263.bits` (which is obtained after encoding back the reconstructed file `tmnrvt.yuv` to H.263 baseline level 10) can be played in 3G handset. This means that Telenor H.263 coder is equivalent to H.263 coder in 3G handset.

As mentioned earlier a reconstructed file is the file obtained after decoding back an encoded video file.

Likewise the following commands were used to encode standard YUV video clips grabbed at 30Hz to H.263 baseline level 10:

```
./tmn -x 2 -a 0 -b 89 -s 15 -r 40000 -S 2 -Z 30.0 -O 0 -i
akiyo_qcif_ori90.yuv -B tmn_akiyo_h263.bits

./tmn -x 2 -a 0 -b 89 -s 15 -r 40000 -S 2 -Z 30.0 -O 0 -i
carphone_qcif_ori90.yuv -B enc-vdo/tmn_cp_h263.bits

./tmn -x 2 -a 0 -b 89 -s 15 -r 40000 -S 2 -Z 30.0 -O 0 -i
foreman_qcif_ori90.yuv -B enc-vdo/tmn_fm_h263.bits
```

All of these encoded H.263 files were also playable in 3G handset. Table 5.6 describes the options/switches used to encode both the decoded/reconstructed traced H.263 video clip (reconstructed/decoded using TMN decoder) and standard YUV video clips to H.263 baseline level 10.

Table 5.6 / Description of options/switches used to encode reconstructed YUV clip back to H.263 using TMN h.263 encoder

Encoding par ameter	Description
`-x <n>`	The value of 'x' specifies the resolution of output encoded file as follows: x=1 for SQCIF with a resolution of 128 x 96 x=2 for QCIF with a resolution of 176 x 144 x=3 for CIF with a resolution of 352 x 288 x=4 for 4CIF with a resolution of 704x576 x=5 for 16CIF with a resolution of 1408x1152 The default encoding format is CIF (i.e. x = 3). Here <n> means that x can have only integer value.
`-a <n>`	The value of 'a' determines the image or frame number from which the encoding starts
`-b <n>`	The value of 'b' determines the image or frame number at which the encoding ends

-s <n>	The value of 's' specifies the integer pel search seek distance for motion estimation. 's' ranges from 0 to 15.
-q <n>	The value of q, if given as an integer, determines the value of inter quantization parameter. Default is 10.
-r <n>	Used to specify the target bit-rate in bits/second. As mentioned earlier, for video telephony on circuit switched radio access bearer, the maximum bandwidth allocated to video is 42 kbps (MOTODEV, 2006). For standard video clips the target bitrate is set to 40 kbps because the bitrate achieved using a target bit-rate of 42 kbps slightly exceeds 42 kbps.
-Z <n>	Frame rate at which the original video sequence was grabbed. It can be 25 or 30. Default is 30.
-S <n>	Tells the encoder the number of frames to skip between each encoded frame with respect to the original sequence reference frame rate. This option is used to control the frame rate (and hence the bitrate) of the output encoded video sequence when encoding an original YUV sequence grabbed from a source to H.263 as shown in figure 4.2. The default value of 'S' is 2. From the source code of Telenor's H.263 encoder (i.e. file config.h), the relation between the encoded sequence frame rate and the number of frames skipped between each encoded frame is given as: **Encoded sequence frame rate = Reference frame rate/(frame skip(S)+1) .....(1)** To encode standard YUV clips to H.263 a frame skip (i.e. 'S') of 2 seems to be sufficient to achieve a target bitrate less than 42 kbps. For reconstructed video clips this option is invalid because no frames are skipped while encoding a reconstructed video clip. To achieve a lower bitrate a) some frames are always skipped in order to encode an original YUV sequence to H.263 baseline level 10; and b) a reconstructed video sequence is obtained by decoding back that H.263 encoded video sequence to YUV format. This implies that a reconstructed video sequence already contains only a fraction of total number of frames in the original sequence. Therefore, no further frames are skipped during it's encoding for second time. However we need to tell the H.263 encoder what fraction of the total number of frames this reconstructed sequence contains. This is achieved using option 'O' discussed below which replaces 'S' when we are encoding a reconstructed video clip to H.263. Figure 4.3 illustrates the use of option 'O' and the difference between the uses of options 'O' and 'S'.
-O <n>	As discussed above, this option is used to tell H.263 encoder the number of frames that were skipped while encoding the original YUV sequence to H.263 (i.e. when the H.263 was being encoded for the first time). This option is used while encoding both the standard and the reconstructed

	video clips. From the source code of Telenor H.263 encoder (i.e. file config.h), the relation between the encoded sequence frame rate and the number of frames skipped between each encoded frame is given as: **Original sequence frame rate=Reference frame rate/(original frame skip(O)+1)...(2)** For instance, O=3 implies that the original sequence was grabbed at 7.5Hz provided that reference frame rate is 30. Conversely, if the original sequence was grabbed at a frame rate of 7.5Hz we should set 'O' to 3. Since the reconstructed sequence (named `tmnrvt.yuv`) was obtained by decoding the traced H.263 video clip, it is not known what fraction of the total number of frames (in the original YUV sequence), it contains. The reason is that we do not know the number of frames skipped during the encoding of traced H.263 video clip. So the exact value of 'O' while encoding a reconstructed sequence is not known. However, experiments showed that setting 'O' to 3 (corresponding to a frame rate of 7.5Hz according to equation (2)) while encoding reconstructed video clip `tmnrvt.yuv` gives the best video quality for the encoded file `enc_tmn_h263.bits`. For original standard YUV sequences (i.e. akiyo, carphone and foreman) grabbed from a video source the value of 'O' is zero as they have been grabbed at a frame rate of 30 frames per second (POLY, 2006) and the reference picture rate is also 30. Hence no frame is skipped with respect to reference frame rate.
`-i`	What follows after 'i', is the name of input file to be encoded.
`-B`	The filename for output H.263 bitstream is written after 'B'.

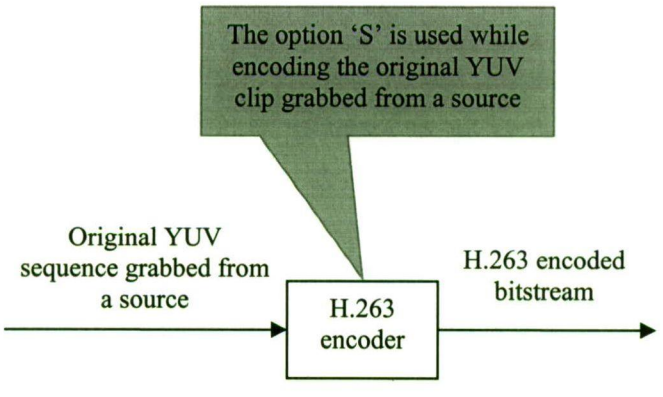

Figure 5.2 Illustrating the use of option 'S'

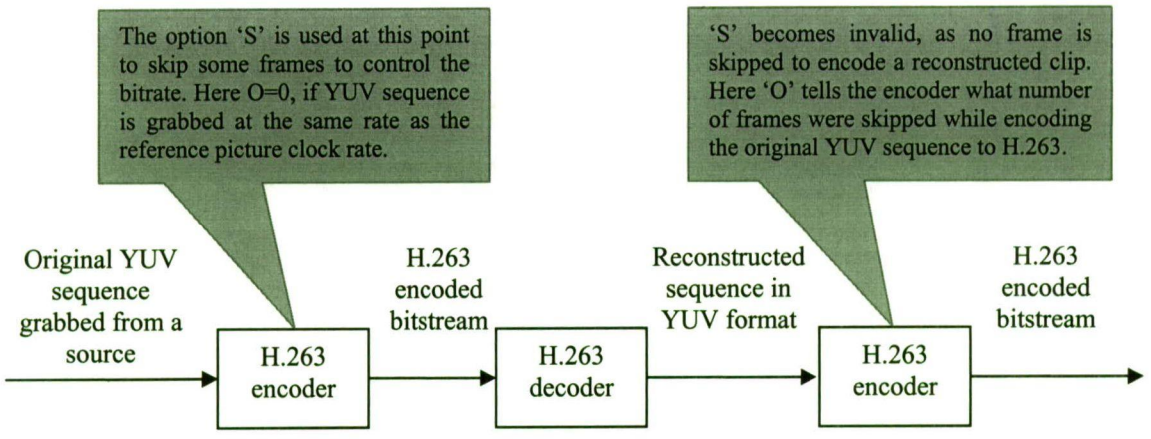

Figure 5.3 Illustrating the relation between the usages of options 'S' and 'O'.

5.3.7 Reconstructing H.263 baseline level 10 with TMN-1.7 decoder

To reconstruct the three test sequences, namely akiyo, carphone and foreman, the following commands were used:

```
./tmndec -o5 tmn_akiyo_h263.bits tmn_rc_ak.yuv
./tmndec -o5 tmn_cp_h263.bits tmn_rc_cp.yuv
./tmndec -o5 tmn_fm_h263.bits tmn_rc_fm.yuv
```

The resulting reconstructed files were playable in a YUV player. Hence, it was verified that the TMN decoder can successfully reconstruct the encoded H.263 baseline level 10 sequences to YUV format.

5.4 Findings

After verifying that Telenor's H.263 codec can decode, as well as encode to, H.263 baseline level 10 bitstream, it seems reasonable to conclude that it should be preferred over FFmpeg to be incorporated into Asterisk due to the following reasons:

- It can correctly decode the traced H.263 baseline level 10 bitstream encoded by H.263 coder in 3G handset without producing any muxing overhead.
- Offers a finer control over various encoding parameters like target bitrate, rate control mode, frame skip and integer pel search window etc. As mentioned above FFmpeg does not offer a very fine control over the bitrates of H.263 encoded video.
- Can be used more confidently because the default values/positions of every options/switches are known. But in case of FFMPEG the default values/positions of every option/switch are not readily known.

- Has a much simpler and shorter command line utility than that of FFmpeg and therefore easier to use.
- Is a stand alone codec and therefore relatively easier to incorporate in to Asterisk PBX.

6 Discussion, suggestions for future work and conclusion

This chapter summarises the work that has been done so far on the project 'Video Quality Measurement for 3G Handset' by the team at University of Plymouth. It then discusses the work reported by this thesis in more detail and gives suggestions for future work. Finally a conclusion is drawn based on the results of research and experiments carried out.

6.1 Discussion and suggestions for future work

According to the project proposal, University of Plymouth was supposed to focus on extending the capabilities of Asterisk server to enable it to act as a part of the test platform for measuring video quality for 3G handset during a video call. The team at University of Plymouth has added the support for 3G-324M protocol and AMR codec in to Asterisk. Due to these added functionalities, Asterisk can now act as a 3G-324M terminal and we can record video and audio clips from H.223 channel during a video call.

These clips were used by the author to test if FFmpeg-libavcodec and Telenor H.263 codec can decode these clips. Similarly, due to these new capabilities of Asterisk, we were also able to insert video clips encoded by FFmpeg-libavcodec and Telenor H.263 codec in to H.223 channel during a video call to test if they are playable in 3G handset. Due to the limitation of time the complete functionality of 3G-324M-SIP gateway could not be incorporated in to Asterisk. However there is likelihood that it might be the future work.

Since incorporating H.263 baseline level 10 and MPEG-4 simple profile @ level 0 first involves identification of the correct H.263 and MPEG-4 codecs used by 3G handset and then adding these codecs into Asterisk PBX. At this stage the correct H.263 codec, mandated by 3GPP for 3G-324M terminals, has been identified. The successful decoding of traced H.263 video clips has been accomplished using Telenor H.263 codec. Similarly standard YUV clips encoded by Telenor codec to H.263 baseline level 10 are also playable by 3G handset. Moreover, Telenor codec seems to be suitable for adding in to Asterisk because it is a stand alone codec. It has a small command line utility but that can be easily removed, and the encoding and decoding parameters can be set in the header files. However, due to the shortage of time coupled with the problem that there is no example of the implementation of video codec in Asterisk; Telenor H.263 codec could not be added in to Asterisk. Again this might be accomplished in the next phase of the project in future.

Our research and experiments showed that FFmpeg, through its command line utility *ffmpeg*, can neither decode nor encode to MPEG-4 simple profile @ level 0 bitstream. However FFmpeg-libavcodec when used directly with the program `apiexample.c` given on the disk accompanying this thesis can decode MPEG-4 simple profile @ level 0 bitstream to pgm images. This has also invoked discussion on FFmpeg-developer's forum. One guy claimed that he can encode to simple profile @ level 0 by first encoding the file in FFmppeg to 3gp format and then hinting it with MP4Box, as given below:

```
ffmpeg -i file.avi -f 3gp -s 176x144 -b 64k -vcodec mpeg4 -profile 0 -
level 8 -r 10 -acodec amr_nb -ar 8000 -ab 8 -ac 1 file.3gp

MP4Box -hint -3gp file.3gp
```

It is clear from the above commands that both the input (`file.avi`) and output (`file.3gp`) files are movie files and not video only sequences. The 3gp format is a movie format with H.263 or H.264 or MPEG-4 video plus AMR-NB or AAC-LC as audio (Wikipedia, 2007). This implies that, if successful, this method would give a movie file (with multiplexed audio and video) with MPEG-4 simple profile @ level 0 video and AMR-NB audio. This is basically not our requirement. Because we are trying to find a video codec that can encode YUV video only clips to MPEG-4 simple profile @ level 0 video only bitstreams whereas this method produces a movie whose video is MPEG-4 simple profile @ level 0. Nevertheless, we used the same commands by disabling audio with (`-an` option) as we needed video only stream but still the encoded file was not playable in 3G handset. We also encoded the standard YUV clip to raw MPEG-4 video only format (with .m4v extension) and then hinted the resulting file with MP4Box but the resulting file was almost empty. This indicates that the 3gp hinting works only with files with .3gp extension.

As discussed earlier, FFmpeg does accept a value of 8 (corresponding to level 0) for the parameter 'level' and writes that value in the profile and level indication filed. This gives the impression that the file is simple profile @ level 0 video. But in fact when various start codes present in this bitstream are examined it becomes clear that the file is simple profile @ level 1 and not @ level 0. It does not mean that FFmpeg has bugs or has little usefulness. The fact is that there is a basic difference between H.263 baseline level 10 and MPEG-4 simple profile @ level 0. H.263 baseline level 10 is the simplest subset of tools of H.263 codec. This is not the case with MPEG-4 simple profile @ level 0. In case of MPEG-4 the simplest subset of tools is simple profile @ level 1 and not level 0. To make this point clear, it might be worth here to look at the history of level 0 of simple profile. As discussed earlier MPEG-4 simple profile @ level 0 was not included in the original MPEG-4 visual standard. It was included in the 2nd extension to the 2nd edition of MPEG-4 visual standard. In December 1999, due to very limited resources of mobile handsets, 3GPP recommended some restrictions to limit the working range of parameters of level 1 of simple profile to achieve low complexity. Later on, in January 2001, level 1 plus these restrictions were included in the MPEG-4 visual standard as level 0 of simple profile on the recommendation of 3GPP (M4IF, 2007).

However, considering the fact that FFmpeg has all the basic functionalities required for MEPG-4 encoding and decoding, it is suggested that instead of writing new software for MPEG-4 simple profile @ level 0 codec, source code of FFmpeg should be modified to add support for simple profile @ level 0 codec. This task can be accomplished a lot quicker if people from FFmpeg developers community work on it rather than a newbie student who may take a lot of time to develop familiarity with FFmpeg.

6.2 Conclusion

The team at University of Plymouth has successfully added the support for 3G-324M protocol and AMR audio codec in Asterisk. Likewise the correct H.263 codec, mandated by 3GPP for 3G-324M terminals, has been identified which can now be added in to Asterisk in the next phase of the project. Since Asterisk can now act as a 3G-324M terminal, this feature can be used for testing purposes to incorporate further capabilities in to Asterisk in future. It was due to this added feature that we were able to get the traced AMR, H.263 baseline level 10 and MPEG-4 simple profile @ level 0 bitstreams and use these clips for testing purposes. Also using this feature we could test if a video clip encoded by a particular encoder is playable in 3G handset; or in other words, is that particular encoder is equivalent to the one used by 3G handset.

Finally, it seems reasonable to conclude that at the time of writing, even with moderate modifications in its source code, FFmpeg-libavcodec can not encode to MPEG-4 simple profile @ level 0. To enable it to encode to simple profile @ level 0, either major modifications in the source code of MPEG-4 codec in libavcodec are required, and/or certain parts of the source code which are hard coded to encode to simple profile @ level 1, need to be rewritten. Although the H.263 files encoded by FFmpeg-libavcodec are playable by the video player in 3G handset, FFmpeg does not offer a very fine control over bitrate of encoded bitstreams. Similarly, it can decode/reconstruct the H.263 files encoded by 3G handset but reports a very high muxing overhead.

Telenor H.263 codec can decode, as well as encode to, H.263 baseline level 10 bitstream. Unlike libavcodec, it can decode/reconstruct the H.263 baseline level 10 bitstream encoded by H.263 encoder in 3G handset without producing any muxing overhead. It can also encode standard YUV clips to H.263 baseline level 10 and offers a finer control over various encoding parameters (like target bitrate, rate control mode, frame skip and integer pel search window etc.) than FFmpeg. Telenor H.263 codec can therefore be incorporated in to Asterisk for encoding and decoding of H.263 baseline level 10 bitstreams. However, due to the fact that there is no example of a video codec already implemented in Asterisk, this task can be quite challenging and time consuming.

7 References

• Books:

Madsen, L., Meggelen, V. and Smith J. (2005), "Asterisk™: The Future of Telephony", O'Reilly Media, Inc., California

Allison, M., Rhodes, C., Spencer, M. and the Asterisk® documentation team (2003), "The Asterisk® Handbook Draft Version 2", Digium, Inc.

• Journal papers:

Cote, G., Erol, B., Gallant, M. and Kossentini, F. (1998), "H.263+: Video Coding at Low Bit Rates", IEEE Transactions on Circuits and Systems for Video Technology, Vol. 8, No. 7

S. Winkler and F. Dufaux, P. (2003), "Video quality Evaluation for Mobile Applications", Proc. SPIE/IS&T Visual Communications and Image Processing Conference, Vol. 5150, pp. 593-603

Fitzek H.P.and Reisslein, M. (2001), "MPEG-4 and H.263 Video Traces for Network Performance Evaluation", IEEE Network, Vol. 15, No. 6, pp. 40-54

Smith, J.R. and Jabri, M. A. (2004), "The 3G-324M Protocol for Conversational Video Telephony", IEEE MultiMedia, Vol. 11, No. 3, pp. 102-105

• Websites:

3GPP Web Site (2000), "3GPP TSG SA WG4: Proposed LS to MPEG-4 regarding 3G-324M", http://www.3gpp.org/ftp/tsg_sa/TSG_SA/TSGS_09/Docs/PDF/SP-000401.pdf, (Accessed 23 August 2006)

3GPP Web Site (2004), "3GPP TS-26.110 v6.0.0 (2004-12), Codecs for Circuit Switched Multimedia Telephony Service, General Descripton, (Release 6)", http://www.3gpp.org/ftp/Specs/html-info/26110.htm, (Accessed 28 August 2006)

3GPP Web Site (2004), "3GPP TS-26.111 v6.1.0 (2004-12), Codecs for Circuit Switched Multimedia Telephony Service, Modifications to H.324, (Release 6)", http://www.3gpp.org/ftp/Specs/html-info/26111.htm, (Accessed 23 August 2006)

Asterisk® Integrated Telephony Web Site (2006), "Asterisk PABX Features", http://www.asteriskit.com.au/page/asterisk-PABX-Features, (Accessed 19 August 2006)

Asterisk® developer's documentation Web Site (2006), "Asterisk Developer's Documentation", http://www.asterisk.org/doxygen/trunk, (Accessed 22 August 2006)

Bohme, M. (2004), Institut für Neuro- und Bioinformatik, "Using libavformat and libavcodec", http://www.inb.uni-luebeck.de/~boehme/using_libavcodec.html (Accessed 10 July 2006)

Citeseer Web Site (2006), "ITU-T Video Coding Standards", http://citeseer.ist.psu.edu/401882.html, (Accessed 19 August 2006)

CommsDesign Web Site (2003), "Understanding the 3G-324M spec", By Eli Orr, http://www.commsdesign.com/design_corner/OEG20030121S0009, (Accessed 21 April 2006)

Digium™ Asterisk® Web Site (2006), "What is Asterisk", http://www.Asterisk®.org/about, (Accessed 11 August 2006)

Digium™ Asterisk® Web Site (2006), "Architecture", http://www.asterisk.org/architecture, (Accessed 18 August 2006)

Digium™ Asterisk® Web Site (2007), "Features", http://www.asterisk.org/features, (Accessed 11 August 2006)

Digium™ Asterisk® Web Site (2007), "Hardware", http://www.asterisk.org/hardware, (Accessed 12 August 2006)

Digium™ Web Site (2006), "Applications", http://www.digium.com/en/asteriskbusinesses/applications, (Accessed 13 March 2006)

Digium™ Web Site (2007), "Press Release", http://www.digium.com/en/mediacenter/news/viewpress.php?id=DigiumSuccessesVON, (Accessed 1 January 2007)

FFmpeg Web Site (2006), "FFmpeg Documentation", http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC6, (Accessed 21 April 2006)

Furht, F. and Marqure, O. (2003), "The Handbook of Video Databases: Design and Applications", (Ed.), CRC Press, pp. 1041-1078

Indigo Web Site (2006), "Understanding MPEG-4 Video", http://www.indigovision.com/site/modules/White_Papers/IC-COD REP012%20Understanding%20MPEG-4%20Video.pdf, (Accessed 3 June 2006)

ITU Web Site (1999), "Subjective video quality assessment methods for multimedia applications – Recommendation P.910", http://www.itu.int/rec/T-REC-P.910-199909-I/en, (Accessed 1 January 2007)

ITU Web Site (2005), "Video coding for low bitrate communication", http://www.itu.int/rec/dologin_pub.asp?lang=e&id=T-REC-H.263-200501-I!!PDF-E&type=items, (Accessed 21 November 2006)

Linuxdevices.com (2006), "Punctuating the Path to Open source Packet Voice", http://linuxdevices.com/articles/AT8678310302.html, (Accessed 27 September 2006)

Lillevold Web Site (2000), "H.263 TMN Source Code", http://www.lillevold.com/tmn.htm, (Accessed 3 October 2006)

Lobster technologies Web Site (2006), "Writing Native Extensions for Asterisk 1.2", http://www.lobstertech.com/doc/ast-12-func/, (Accessed 24 September 2006)

Laboratory for image and video engineering (2003), "Image and Video Quality Assessment at LIVE", http://live.ece.utexas.edu/research/quality/intro.htm, (Accessed 28 June 2006)

MPEG-4 Industry Forum Web Site (2006), "Levels for MPEG-4 Visual Profiles", http://www.m4if.org/resources/profiles/index.html#MPEG01B, (Accessed 23 August 2006)

Millengence Inc. Web Site (2006), "Astreisk: A Non-Technical Overview", http://www.millenigence.com/articles/asterisk-non-technical-review.pdf, (Accessed 18 August 2006)

Mandriva Ireland Web Site (2006), "Asterisk Features", http://www.mandriva.ie/asterisk/features.html#transfer, (Accessed 8 August 2006)

NMS Communications Web Site (2007), "Application Brief", http://www.nmscommunications.com/NR/rdonlyres/96EF2885-9E1E-4F53-8513-213A528414C8/0/ConversationalVideoAppBrief.pdf, (Accessed 2 January 2007)

NMS Communications Web Site (2007), "3G-324M Video Technology Overview", http://www.nmscommunications.com/DevPlatforms/OpenAccess/Technologies/3G324MandIPVideo/TechnologyOverview.htm, (Accessed 8 August 2006)

Opticom Web Site (2006), "Percetual Evaluation of Video Quality", http://www.opticom.de/technology/pevq_video-quality-testing.html, (Accessed 23 June 2006)

Open Source Telephony Group Web Site (2006), "Asterisk-Default branch", http://freshmeat.net/projects/asterisk, (Accessed 18 August 2006)

Patent Storm Web Site (2004), "Error recovery of corrupted MPEG-4 bitstreams using fuzzy decoding of start codes and resync markers", http://www.patentstorm.us/patents/6728318-description.html, (Accessed 15 January 2007)

Polytechnic University, Electrical and Computer Engineering Department Web Site (2002), http://eeweb.poly.edu/~yao/VideobookSampleData/doc/sample/H263.htm, (Accessed 3 October 2006)

Psytechnics Web Site (2006), "PVA-Communications", http://www.psytechnics.com/site/sections/products/pva.php, (Accessed 1 June 2006)

Radvision Web Site (2002), "3G Powered 3G-H.324M Protocol", http://www.radvision.com/NR/rdonlyres/6066D65E-9E2E-4786-822A-B1FDBF59A86A/0/3GPowered3GH324MProtocol.pdf, (Accessed 20 July 2006)

Sun, l. (2006), 'Research Proposal for Video Quality Measurement for 3G Handset'.

TellaSonera Web Site (2004), "MPEG-4 White Paper", http://www.medialab.sonera.fi/workspace/MPEG4WhitePaper.pdf, (Accessed 24 July 2006)

The Motorola Developer Network Web Site (2006), "Media Guides - Motorola E1000 Media Guide", http://developer.motorola.com/?path=1.2.7.39, (Accessed 15 August 2006)

Voip-info.org Web Site (2006), http://www.voip-info.org/wiki/view/asterisk+introduction, (Accessed 4 August 2006)

Voip-info.org Web Site (2006), "Asterisk as H.324M-SIP gateway", http://www.voip-info.org/wiki/view/Asterisk+H324M, (Accessed 13 July 2006)

Voipfone Web Site (2006), "Voipfone Call Queuing", http://www.voipfone.co.uk/PB_Call_Queuing.php, (Accessed 17 August 2006)

Wikipedia Web Site (2006), "Automatic Number Identification", http://en.wikipedia.org/wiki/Automatic_number_identification, (Accessed 19 August 2006)

Watson, A. and Winkler, S. (2000), "Video Quality Experts Group: Current Results and Future Directions" in Proc. SPIE Visual Communications and Image Processing, vol. 4067, pp. 742-753, Perth, Australia, June 21-23, 2000

Wireless System Design Magazine Web Site (2004), "3G-324M Helps 3G Live Up To Its Potential", http://www.wsdmag.com/Articles/ArticleID/7742/7742.html, (Accessed 25 July 2006)

Wikipedia Web Site (2006), "Video Quality", http://en.wikipedia.org/wiki/video_quality, (Accessed 09 July 2006)

Wikipedia Web Site (2007), "3GP", http://en.wikipedia.org/wiki/3GP, (Accessed 19 January 2007)

Xorcom Web Site (2006), "Risk Free Asterisk", http://www.xorcom.com/asterisk.html, (Accessed 22 October 2006)

Appendixes

Appendix A: Features of Asterisk

Copied from www.asterisk.org/features (Accessed 1 January 2007)

Features

The following id the list of features that Asterisk supports currently.

Call features

ADSI On-Screen Menu System
Alarm Receiver
Append Message
Authentication
Automated Attendant
Blacklists
Blind Transfer
Call Detail Records
Call Forward on Busy
Call Forward on No Answer
Call Forward Variable
Call Monitoring
Call Parking
Call Queuing
Call Recording
Call Retrieval
Call Routing (DID & ANI)
Call Snooping
Call Transfer
Call Waiting
Caller ID
Caller ID Blocking
Caller ID on Call Waiting
Calling Cards
Conference Bridging
Database Store / Retrieve
Database Integration
Dial by Name
Direct Inward System Access
Distinctive Ring
Distributed Universal Number Discovery (DUNDi™)
Do Not Disturb
E911
ENUM
Fax Transmit and Receive (3rd Party OSS Package)
Flexible Extension Logic
Interactive Directory Listing
Interactive Voice Response (IVR)
Local and Remote Call Agents
Macros
Music On Hold
Music On Transfer:
- Flexible Mp3-based System
- Random or Linear Play
- Volume Control

Predictive Dialer
Privacy
Open Settlement Protocol (OSP)
Overhead Paging
Protocol Conversion
Remote Call Pickup
Remote Office Support
Roaming Extensions
Route by Caller ID
SMS Messaging
Spell / Say
Streaming Media Access
Supervised Transfer
Talk Detection
Text-to-Speech (via Festival)
Three-way Calling
Time and Date
Transcoding
Trunking
VoIP Gateways
Voicemail:
- Visual Indicator for Message Waiting
- Stutter Dialtone for Message Waiting
- Voicemail to email
- Voicemail Groups
- Web Voicemail Interface
Zapateller

Computer-Telephony Integration

AGI (Asterisk Gateway Interface)
Graphical Call Manager
Outbound Call Spooling
Predictive Dialer
TCP/IP Management Interface

Scalability

TDMoE (Time Division Multiplex over Ethernet)
Allows direct connection of Asterisk PBX
Zero latency
Uses commodity Ethernet hardware
Voice-over IP
Allows for integration of physically separate installations
Uses commonly deployed data connections
Allows a unified dialplan across multiple offices

Codecs

ADPCM
G.711 (A-Law & μ-Law)
G.723.1 (pass through)
G.726
G.729 (through purchase of a commercial license)
GSM
iLBC
Linear

LPC-10
Speex

Protocols

IAX™ (Inter-Asterisk Exchange)
H.323
SIP (Session Initiation Protocol)
MGCP (Media Gateway Control Protocol
SCCP (Cisco® Skinny®)

Traditional Telephony Interoperability

E&M
E&M Wink
Feature Group D
FXS
FXO
GR-303
Loopstart
Groundstart
Kewlstart
MF and DTMF support
Robbed-bit Signaling (RBS) Types
MFC-R2 (Not supported. However, a patch is available)

PRI Protocols

4ESS
BRI (ISDN4Linux)
DMS100
EuroISDN
Lucent 5E
National ISDN2
NFAS

Appendix B1: FFmpeg Options

(Copied from http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC7 (Accessed 1 January 2007)

The following is the list of all the basic and advanced main, audio and video options supported by FFmpeg:

Main options

`-L'

show license.

`-h'

Show help.

`-version'

Show version.

`-formats'

Show available formats, codecs, protocols, ...

`-f fmt'

Force format.

`-i filename'

input filename

`-y'

Overwrite output files.

`-t duration'

Set the recording time in seconds. hh:mm:ss[.xxx] syntax is also supported.

`-fs limit_size'

Set the file size limit.

`-ss position'

Seek to given time position in seconds. hh:mm:ss[.xxx] syntax is also supported.

`-itsoffset offset'

Set the input time offset in seconds. [-]hh:mm:ss[.xxx] syntax is also supported. This option affects all the input files that follow it. The offset is added to the timestamps of the input files. Specifying a positive offset means that the corresponding streams are delayed by 'offset' seconds.

`-title string'

Set the title.

`-timestamp time'

Set the timestamp.

`-author string'

Set the author.

`-copyright string'

Set the copyright.

`-comment string'

Set the comment.

`-album string'

Set the album.

`-track number'

Set the track.

`-year number'

Set the year.

`-v verbose'

Control amount of logging.

`-target type'

Specify target file type ("vcd", "svcd", "dvd", "dv", "dv50", "pal-vcd", "ntsc-svcd", ...). All the format options (bitrate, codecs, buffer sizes) are then set automatically. You can just type:

ffmpeg -i myfile.avi -target vcd /tmp/vcd.mpg

Nevertheless you can specify additional options as long as you know they do not conflict with the standard, as in:

ffmpeg -i myfile.avi -target vcd -bf 2 /tmp/vcd.mpg

`-dframes number'

Set the number of data frames to record.

`-scodec codec'

Force subtitle codec ('copy' to copy stream).

`-newsubtitle'

Add a new subtitle stream to the current output stream.

`-slang code'

Set the ISO 639 language code (3 letters) of the current subtitle stream.

Video Options

`-b bitrate'

Set the video bitrate in bit/s (default = 200 kb/s).

`-vframes number'

Set the number of video frames to record.

`-r fps'

Set frame rate (Hz value, fraction or abbreviation), (default = 25).

`-s size'

Set frame size. The format is `wxh' (default = 160x128). The following abbreviations are recognized:

`sqcif'

128x96

`qcif'

176x144

`cif'

352x288

`4cif'

704x576

`-aspect aspect'

Set aspect ratio (4:3, 16:9 or 1.3333, 1.7777).

`-croptop size'

Set top crop band size (in pixels).

`-cropbottom size'

Set bottom crop band size (in pixels).

`-cropleft size'

Set left crop band size (in pixels).

`-cropright size'

Set right crop band size (in pixels).

`-padtop size'

Set top pad band size (in pixels).

`-padbottom size'

Set bottom pad band size (in pixels).

`-padleft size'

Set left pad band size (in pixels).

`-padright size'

Set right pad band size (in pixels).

`-padcolor (hex color)'

Set color of padded bands. The value for padcolor is expressed as a six digit hexadecimal number where the first two digits represent red, the middle two digits green and last two digits blue (default = 000000 (black)).

`-vn'

isable video recording.

`-bt tolerance'

Set video bitrate tolerance (in bit/s).

`-maxrate bitrate'

Set max video bitrate tolerance (in bit/s).

`-minrate bitrate'

Set min video bitrate tolerance (in bit/s).

`-bufsize size'

Set rate control buffer size (in bits).

`-vcodec codec'

Force video codec to codec. Use the copy special value to tell that the raw codec data must be copied as is.

`-sameq'

Use same video quality as source (implies VBR).

`-pass n'

Select the pass number (1 or 2). It is useful to do two pass encoding. The statistics of the video are recorded in the first pass and the video is generated at the exact requested bitrate in the second pass.

`-passlogfile file'

Set two pass logfile name to file.

`-newvideo'

Add a new video stream to the current output stream.

Advanced Video Options

`-pix_fmt format'

Set pixel format.

`-g gop_size'

Set the group of pictures size.

`-intra'

Use only intra frames.

`-vdt n'

Discard threshold.

`-qscale q'

Use fixed video quantizer scale (VBR).

`-qmin q'

minimum video quantizer scale (VBR)

`-qmax q'

maximum video quantizer scale (VBR)

`-qdiff q'

maximum difference between the quantizer scales (VBR)

`-qblur blur'

video quantizer scale blur (VBR)

`-qcomp compression'

video quantizer scale compression (VBR)

`-lmin lambda'

minimum video lagrange factor (VBR)

`-lmax lambda'

max video lagrange factor (VBR)

`-mblmin lambda'

minimum macroblock quantizer scale (VBR)

`-mblmax lambda'

maximum macroblock quantizer scale (VBR) These four options (lmin, lmax, mblmin, mblmax) use 'lambda' units, but you may use the QP2LAMBDA constant to easily convert from 'q' units:

ffmpeg -i src.ext -lmax 21*QP2LAMBDA dst.ext

`-rc_init_cplx complexity'

initial complexity for single pass encoding

`-b_qfactor factor'

qp factor between P- and B-frames

`-i_qfactor factor'

qp factor between P- and I-frames

`-b_qoffset offset'

qp offset between P- and B-frames

`-i_qoffset offset'

qp offset between P- and I-frames

`-rc_eq equation'

Set rate control equation (see section 3.10 FFmpeg formula evaluator) (default = tex^qComp).

`-rc_override override'

rate control override for specific intervals

`-me method'

Set motion estimation method to method. Available methods are (from lowest to best quality):

`zero'

Try just the (0, 0) vector.

`phods'

`log'

`x1'

`epzs'

(default method)

`full'

exhaustive search (slow and marginally better than epzs)

`-dct_algo algo'

Set DCT algorithm to algo. Available values are:

`0'

FF_DCT_AUTO (default)

`1'

FF_DCT_FASTINT

`2'

FF_DCT_INT

`3'

FF_DCT_MMX

`4'

FF_DCT_MLIB

`5'

FF_DCT_ALTIVEC

`-idct_algo algo'

Set IDCT algorithm to algo. Available values are:

`0'

FF_IDCT_AUTO (default)

`1'

FF_IDCT_INT

`2'

FF_IDCT_SIMPLE

`3'

FF_IDCT_SIMPLEMMX

`4'

FF_IDCT_LIBMPEG2MMX

`5'

FF_IDCT_PS2

`6'

FF_IDCT_MLIB

`7'

FF_IDCT_ARM

`8'

FF_IDCT_ALTIVEC

`9'

FF_IDCT_SH4

`10'

FF_IDCT_SIMPLEARM

`-er n'

Set error resilience to n.

`1'

FF_ER_CAREFUL (default)

`2'

FF_ER_COMPLIANT

`3'

FF_ER_AGGRESSIVE

`4'

FF_ER_VERY_AGGRESSIVE

`-ec bit_mask'

Set error concealment to bit_mask. bit_mask is a bit mask of the following values:

`1'

FF_EC_GUESS_MVS (default = enabled)

`2'

FF_EC_DEBLOCK (default = enabled)

`-bf frames'

Use 'frames' B-frames (supported for MPEG-1, MPEG-2 and MPEG-4).

`-mbd mode'

macroblock decision

`0'

FF_MB_DECISION_SIMPLE: Use mb_cmp (cannot change it yet in FFmpeg).

`1'

FF_MB_DECISION_BITS: Choose the one which needs the fewest bits.

`2'

F_MB_DECISION_RD: rate distortion

`-4mv'

se four motion vector by macroblock (MPEG-4 only).

`-part'

se data partitioning (MPEG-4 only).

`-bug param'

ork around encoder bugs that are not auto-detected.

`-strict strictness'

ow strictly to follow the standards.

`-aic'

nable Advanced intra coding (h263+).

`-umv'

nable Unlimited Motion Vector (h263+)

`-deinterlace'

einterlace pictures.

`-ilme'

orce interlacing support in encoder (MPEG-2 and MPEG-4 only). Use this option if your input file is nterlaced and you want to keep the interlaced format for minimum losses. The alternative is to einterlace the input stream with `-deinterlace', but deinterlacing introduces losses.

`-psnr'

alculate PSNR of compressed frames.

`-vstats'

ump video coding statistics to `vstats_HHMMSS.log'.

`-vhook module'

nsert video processing module. module contains the module name and its parameters separated by paces.

`-top n'

op=1/bottom=0/auto=-1 field first

`-dc precision'

ntra_dc_precision.

-vtag fourcc/tag'

orce video tag/fourcc.

`-qphist'

how QP histogram.

`-vbsf bitstream filter'

Bitstream filters available are "dump_extra", "remove_extra", "noise".

Audio Options

`-aframes number'

Set the number of audio frames to record.

`-ar freq'

Set the audio sampling frequency (default = 44100 Hz).

`-ab bitrate'

Set the audio bitrate in kbit/s (default = 64).

`-ac channels'

Set the number of audio channels (default = 1).

`-an'

Disable audio recording.

`-acodec codec'

Force audio codec to codec. Use the copy special value to specify that the raw codec data must be copied as is.

`-newaudio'

Add a new audio track to the output file. If you want to specify parameters, do so before -newaudio (-acodec, -ab, etc..). Mapping will be done automatically, if the number of output streams is equal to the number of input streams, else it will pick the first one that matches. You can override the mapping using -map as usual. Example:

ffmpeg -i file.mpg -vcodec copy -acodec ac3 -ab 384 test.mpg -acodec mp2 -ab 192 -newaudio

`-alang code'

Set the ISO 639 language code (3 letters) of the current audio stream.

Advanced Audio options:

`-atag fourcc/tag'

Force audio tag/fourcc.

`-absf bitstream filter'

Bitstream filters available are "dump_extra", "remove_extra", "noise", "mp3comp", "mp3decomp".

Subtitle options:

`-scodec codec'

Force subtitle codec ('copy' to copy stream).

`-newsubtitle'

Add a new subtitle stream to the current output stream.

`-slang code'

Set the ISO 639 language code (3 letters) of the current subtitle stream.

Audio/Video grab options

`-vd device'

sEt video grab device (e.g. `/dev/video0').

`-vc channel'

Set video grab channel (DV1394 only).

`-tvstd standard'

Set television standard (NTSC, PAL (SECAM)).

`-dv1394'

Set DV1394 grab.

`-ad device'

Set audio device (e.g. `/dev/dsp').

`-grab format'

Request grabbing using.

`-gd device'

Set grab device.

Advanced options

`-map input stream id[:input stream id]'

Set stream mapping from input streams to output streams. Just enumerate the input streams in the order you want them in the output. [input stream id] sets the (input) stream to sync against.

`-map_meta_data outfile:infile'

Set meta data information of outfile from infile.

`-debug'

Print specific debug info.

`-benchmark'

Add timings for benchmarking.

`-dump'

Dump each input packet.

`-hex'

When dumping packets, also dump the payload.

`-bitexact'

Only use bit exact algorithms (for codec testing).

`-ps size'

Set packet size in bits.

`-re'

Read input at native frame rate. Mainly used to simulate a grab device.

`-loop_input'

Loop over the input stream. Currently it works only for image streams. This option is used for automatic FFserver testing.

`-loop_output number_of_times'

Repeatedly loop output for formats that support looping such as animated GIF (0 will loop the output infinitely).

`-threads count'

Thread count.

`-vsync parameter'

Video sync method. Video will be stretched/squeezed to match the timestamps, it is done by duplicating and dropping frames. With -map you can select from which stream the timestamps should be taken. You can leave either video or audio unchanged and sync the remaining stream(s) to the unchanged one.

`-async samples_per_second'

Audio sync method. "Stretches/squeezes" the audio stream to match the timestamps, the parameter is the maximum samples per second by which the audio is changed. -async 1 is a special case where only the start of the audio stream is corrected without any later correction.

Appendix B2: Codecs and Formats supported by FFmpeg

(Copied from http://ffmpeg.mplayerhq.hu/ffmpeg-doc.html#SEC7 (Accessed 1 January 2007)

The following is the codecs and file formats supported by FFmpeg. This list can also be obtained by issuing the command `'ffmpeg -formats'` in FFmpeg command line utility interface.

File Formats

FFmpeg supports the following file formats through the `libavformat` library:

Supported File Format	Encoding	Decoding	Comments
MPEG audio	X	X	
MPEG-1 systems	X	X	muxed audio and video
MPEG-2 PS	X	X	also known as VOB file
MPEG-2 TS		X	also known as DVB Transport Stream
ASF	X	X	
AVI	X	X	
WAV	X	X	
Macromedia Flash	X	X	Only embedded audio is decoded.
FLV	X	X	Macromedia Flash video files
Real Audio and Video	X	X	
Raw AC3	X	X	
Raw MJPEG	X	X	
Raw MPEG video	X	X	
Raw PCM8/16 bits, mulaw/Alaw	X	X	
Raw CRI ADX audio	X	X	
Raw Shorten audio		X	
SUN AU format	X	X	

NUT	X	X	NUT Open Container Format
QuickTime	X	X	
MPEG-4	X	X	MPEG-4 is a variant of QuickTime.
Raw MPEG4 video	X	X	
DV	X	X	
4xm		X	4X Technologies format, used in some games.
Playstation STR		X	
Id RoQ		X	Used in Quake III, Jedi Knight 2, other computer games.
Interplay MVE		X	Format used in various Interplay computer games.
WC3 Movie		X	Multimedia format used in Origin's Wing Commander III computer game.
Sega FILM/CPK		X	Used in many Sega Saturn console games.
Westwood Studios VQA/AUD		X	Multimedia formats used in Westwood Studios games.
Id Cinematic (.cin)		X	Used in Quake II.
FLIC format		X	.fli/.flc files
Sierra VMD		X	Used in Sierra CD-ROM games.
Sierra Online		X	.sol files used in Sierra Online games.
Matroska		X	
Electronic Arts Multimedia		X	Used in various EA games; files have extensions like WVE and UV2.
Nullsoft Video (NSV) format		X	
ADTS AAC audio	X	X	
Creative VOC	X	X	Created for the Sound Blaster Pro.
American Laser Games MM		X	Multimedia format used in games like Mad Dog McCree
AVS		X	Multimedia format used by the Creature Shock game.
Smacker		X	Multimedia format used by many games.

GXF	X	X	General eXchange Format SMPTE 360M, used by Thomson Grass Valley playout servers.
CIN		X	Multimedia format used by Delphine Software games.
MXF		X	Material eXchange Format SMPTE 377M, used by D-Cinema, broadcast industry.
SEQ		X	Tiertex .seq files used in the DOS CDROM version of the game Flashback.

X means that encoding (resp. decoding) is supported.

Image Formats

FFmpeg can read and write images for each frame of a video sequence. The following image formats are supported:

Supported Image Format	Encoding	Decoding	Comments
PGM, PPM	X	X	
PAM	X	X	PAM is a PNM extension with alpha support.
PGMYUV	X	X	PGM with U and V components in YUV 4:2:0
JPEG	X	X	Progressive JPEG is not supported.
.Y.U.V	X	X	one raw file per component
animated GIF	X	X	Only uncompressed GIFs are generated.
PNG	X	X	2 bit and 4 bit/pixel not supported yet.
Targa		X	Targa (.TGA) image format.
TIFF		X	Only 24 bit/pixel images are supported.
SGI	X	X	SGI RGB image format

X means that encoding (resp. decoding) is supported.

Video Codecs

Supported Codec	Encoding	Decoding	Comments
MPEG-1 video	X	X	
MPEG-2 video	X	X	
MPEG-4	X	X	
MSMPEG4 V1	X	X	
MSMPEG4 V2	X	X	
MSMPEG4 V3	X	X	
WMV7	X	X	
WMV8	X	X	not completely working
WMV9		X	not completely working
VC1		X	
H.261	X	X	
H.263(+)	X	X	also known as RealVideo 1.0
H.264		X	
RealVideo 1.0	X	X	
RealVideo 2.0	X	X	
MJPEG	X	X	
lossless MJPEG	X	X	
JPEG-LS	X	X	fourcc: MJLS, lossless and near-lossless is supported
Apple MJPEG-B		X	
Sunplus MJPEG		X	fourcc: SP5X
DV	X	X	
HuffYUV	X	X	
FFmpeg Video 1	X	X	experimental lossless codec (fourcc: FFV1)
FFmpeg Snow	X	X	experimental wavelet codec (fourcc: SNOW)

Asus v1	X	X	fourcc: ASV1
Asus v2	X	X	fourcc: ASV2
Creative YUV		X	fourcc: CYUV
Sorenson Video 1	X	X	fourcc: SVQ1
Sorenson Video 3		X	fourcc: SVQ3
On2 VP3		X	still experimental
On2 VP5		X	fourcc: VP50
On2 VP6		X	fourcc: VP60,VP61,VP62
Theora		X	still experimental
Intel Indeo 3		X	
FLV	X	X	Sorenson H.263 used in Flash
Flash Screen Video		X	fourcc: FSV1
ATI VCR1		X	fourcc: VCR1
ATI VCR2		X	fourcc: VCR2
Cirrus Logic AccuPak		X	fourcc: CLJR
4X Video		X	Used in certain computer games.
Sony Playstation MDEC		X	
Id RoQ		X	Used in Quake III, Jedi Knight 2, other computer games.
Xan/WC3		X	Used in Wing Commander III .MVE files.
Interplay Video		X	Used in Interplay .MVE files.
Apple Animation		X	fourcc: 'rle '
Apple Graphics		X	fourcc: 'smc '
Apple Video		X	fourcc: rpza
Apple QuickDraw		X	fourcc: qdrw
Cinepak		X	

Microsoft RLE		X	
Microsoft Video-1		X	
Westwood VQA		X	
Id Cinematic Video		X	Used in Quake II.
Planar RGB		X	fourcc: 8BPS
FLIC video		X	
Duck TrueMotion v1		X	fourcc: DUCK
Duck TrueMotion v2		X	fourcc: TM20
VMD Video		X	Used in Sierra VMD files.
MSZH		X	Part of LCL
ZLIB	X	X	Part of LCL, encoder experimental
TechSmith Camtasia		X	fourcc: TSCC
IBM Ultimotion		X	fourcc: ULTI
Miro VideoXL		X	fourcc: VIXL
QPEG		X	fourccs: QPEG, Q1.0, Q1.1
LOCO		X	
Winnov WNV1		X	
Autodesk Animator Studio Codec		X	fourcc: AASC
Fraps FPS1		X	
CamStudio		X	fourcc: CSCD
American Laser Games Video		X	Used in games like Mad Dog McCree
ZMBV	X	X	Encoder works only on PAL8
AVS Video		X	Video encoding used by the Creature Shock game.
Smacker Video		X	Video encoding used in Smacker.
RTjpeg		X	Video encoding used in NuppelVideo files.

KMVC		X	Codec used in Worms games.
VMware Video		X	Codec used in videos captured by VMware.
Cin Video		X	Codec used in Delphine Software games.
Tiertex Seq Video		X	Codec used in DOS CDROM FlashBack game.

X means that encoding (resp. decoding) is supported.

Audio Codecs

Supported Codec	Encoding	Decoding	Comments
MPEG audio layer 2	IX	IX	
MPEG audio layer 1/3	IX	IX	MP3 encoding is supported through the external library LAME.
AC3	IX	IX	liba52 is used internally for decoding.
Vorbis	X	X	
WMA V1/V2		X	
AAC	X	X	Supported through the external library libfaac/libfaad.
Microsoft ADPCM	X	X	
MS IMA ADPCM	X	X	
QT IMA ADPCM		X	
4X IMA ADPCM		X	
G.726 ADPCM	X	X	
Duck DK3 IMA ADPCM		X	Used in some Sega Saturn console games.
Duck DK4 IMA ADPCM		X	Used in some Sega Saturn console games.
Westwood Studios IMA ADPCM		X	Used in Westwood Studios games like Command and Conquer.
SMJPEG IMA ADPCM		X	Used in certain Loki game ports.
CD-ROM XA ADPCM		X	

CRI ADX ADPCM	X	X	Used in Sega Dreamcast games.
Electronic Arts ADPCM		X	Used in various EA titles.
Creative ADPCM		X	16 -> 4, 8 -> 4, 8 -> 3, 8 -> 2
RA144		X	Real 14400 bit/s codec
RA288		X	Real 28800 bit/s codec
RADnet	X	IX	Real low bitrate AC3 codec, liba52 is used for decoding.
AMR-NB	X	X	Supported through an external library.
AMR-WB	X	X	Supported through an external library.
DV audio		X	
Id RoQ DPCM		X	Used in Quake III, Jedi Knight 2, other computer games.
Interplay MVE DPCM		X	Used in various Interplay computer games.
Xan DPCM		X	Used in Origin's Wing Commander IV AVI files.
Sierra Online DPCM		X	Used in Sierra Online game audio files.
Apple MACE 3		X	
Apple MACE 6		X	
FLAC lossless audio		X	
Shorten lossless audio		X	
Apple lossless audio		X	QuickTime fourcc 'alac'
FFmpeg Sonic	X	X	experimental lossy/lossless codec
Qdesign QDM2		X	there are still some distortions
Real COOK		X	All versions except 5.1 are supported
DSP Group TrueSpeech		X	
True Audio (TTA)		X	
Smacker Audio		X	
WavPack Audio		X	
Cin Audio		X	Codec used in Delphine Software games.

Intel Music Coder		X	
Musepack		X	Only SV7 is supported

X means that encoding (resp. decoding) is supported.

I means that an integer-only version is available, too (ensures high performance on systems without hardware floating point support).

Appendix C1: C source code for apiexample.c

This program decodes traced MPEG-4 simple Profile @ Level 0 bitstream to pgm (potable gray map) images using FFmpeg-libavcodec. Please note that path for the traced file might need to be modified before the file can be compiled.

```
/**
 * @file apiexample.c
 * avcodec API use example.
 *
 * Note that this library only handles codecs (mpeg, mpeg4, etc...),
 * not file formats (avi, vob, etc...). See library 'libavformat' for the
 * format handling
 */

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

#ifdef HAVE_AV_CONFIG_H
#undef HAVE_AV_CONFIG_H
#endif

#include "avcodec.h"

#define INBUF_SIZE 4096

/*
 * Audio encoding example
 */
void audio_encode_example(const char *filename)
{
    AVCodec *codec;
    AVCodecContext *c= NULL;
    int frame_size, i, j, out_size, outbuf_size;
    FILE *f;
    short *samples;
    float t, tincr;
    uint8_t *outbuf;

    printf("Audio encoding\n");

    /* find the MP2 encoder */
    codec = avcodec_find_encoder(CODEC_ID_MP2);
    if (!codec) {
        fprintf(stderr, "codec not found\n");
        exit(1);
    }

    c= avcodec_alloc_context();

    /* put sample parameters */
    c->bit_rate = 64000;
    c->sample_rate = 44100;
    c->channels = 2;
```

```
    /* open it */
    if (avcodec_open(c, codec) < 0) {
        fprintf(stderr, "could not open codec\n");
        exit(1);
    }

    /* the codec gives us the frame size, in samples */
    frame_size = c->frame_size;
    samples = malloc(frame_size * 2 * c->channels);
    outbuf_size = 10000;
    outbuf = malloc(outbuf_size);

    f = fopen(filename, "wb");
    if (!f) {
        fprintf(stderr, "could not open %s\n", filename);
        exit(1);
    }

    /* encode a single tone sound */
    t = 0;
    tincr = 2 * M_PI * 440.0 / c->sample_rate;
    for(i=0;i<200;i++) {
        for(j=0;j<frame_size;j++) {
            samples[2*j] = (int)(sin(t) * 10000);
            samples[2*j+1] = samples[2*j];
            t += tincr;
        }
        /* encode the samples */
        out_size = avcodec_encode_audio(c, outbuf, outbuf_size, samples);
        fwrite(outbuf, 1, out_size, f);
    }
    fclose(f);
    free(outbuf);
    free(samples);

    avcodec_close(c);
    av_free(c);
}

/*
 * Audio decoding.
 */
void audio_decode_example(const char *outfilename, const char *filename)
{
    AVCodec *codec;
    AVCodecContext *c= NULL;
    int out_size, size, len;
    FILE *f, *outfile;
    uint8_t *outbuf;
    uint8_t inbuf[INBUF_SIZE + FF_INPUT_BUFFER_PADDING_SIZE], *inbuf_ptr;

    printf("Audio decoding\n");

    /* set end of buffer to 0 (this ensures that no overreading happens
for damaged mpeg streams) */
    memset(inbuf + INBUF_SIZE, 0, FF_INPUT_BUFFER_PADDING_SIZE);

    /* find the mpeg audio decoder */
    codec = avcodec_find_decoder(CODEC_ID_MP2);
    if (!codec) {
        fprintf(stderr, "codec not found\n");
```

```
        exit(1);
    }

    c= avcodec_alloc_context();

    /* open it */
    if (avcodec_open(c, codec) < 0) {
        fprintf(stderr, "could not open codec\n");
        exit(1);
    }

    outbuf = malloc(AVCODEC_MAX_AUDIO_FRAME_SIZE);

    f = fopen(filename, "rb");
    if (!f) {
        fprintf(stderr, "could not open %s\n", filename);
        exit(1);
    }
    outfile = fopen(outfilename, "wb");
    if (!outfile) {
        av_free(c);
        exit(1);
    }

    /* decode until eof */
    inbuf_ptr = inbuf;
    for(;;) {
        size = fread(inbuf, 1, INBUF_SIZE, f);
        if (size == 0)
            break;

        inbuf_ptr = inbuf;
        while (size > 0) {
            len = avcodec_decode_audio(c, (short *)outbuf, &out_size,
                                       inbuf_ptr, size);
            if (len < 0) {
                fprintf(stderr, "Error while decoding\n");
                exit(1);
            }
            if (out_size > 0) {
                /* if a frame has been decoded, output it */
                fwrite(outbuf, 1, out_size, outfile);
            }
            size -= len;
            inbuf_ptr += len;
        }
    }

    fclose(outfile);
    fclose(f);
    free(outbuf);

    avcodec_close(c);
    av_free(c);
}

/*
 * Video encoding example
 */
void video_encode_example(const char *filename)
{
```

```
AVCodec *codec;
AVCodecContext *c= NULL;
int i, out_size, size, x, y, outbuf_size;
FILE *f;
AVFrame *picture;
uint8_t *outbuf, *picture_buf;

printf("Video encoding\n");

/* find the mpeg1 video encoder */
codec = avcodec_find_encoder(CODEC_ID_MPEG4);
if (!codec) {
    fprintf(stderr, "codec not found\n");
    exit(1);
}

c= avcodec_alloc_context();
picture= avcodec_alloc_frame();

/* put sample parameters */
c->bit_rate = 400000;
/* resolution must be a multiple of two */
c->width = 352;
c->height = 288;
/* frames per second */
c->time_base= (AVRational){1,25};
c->gop_size = 10; /* emit one intra frame every ten frames */
c->max_b_frames=1;
c->pix_fmt = PIX_FMT_YUV420P;

/* open it */
if (avcodec_open(c, codec) < 0) {
    fprintf(stderr, "could not open codec\n");
    exit(1);
}

/* the codec gives us the frame size, in samples */

f = fopen(filename, "wb");
if (!f) {
    fprintf(stderr, "could not open %s\n", filename);
    exit(1);
}

/* alloc image and output buffer */
outbuf_size = 100000;
outbuf = malloc(outbuf_size);
size = c->width * c->height;
picture_buf = malloc((size * 3) / 2); /* size for YUV 420 */

picture->data[0] = picture_buf;
picture->data[1] = picture->data[0] + size;
picture->data[2] = picture->data[1] + size / 4;
picture->linesize[0] = c->width;
picture->linesize[1] = c->width / 2;
picture->linesize[2] = c->width / 2;

/* encode 1 second of video */
for(i=0;i<25;i++) {
    fflush(stdout);
    /* prepare a dummy image */
```

```
        /* Y */
        for(y=0;y<c->height;y++) {
            for(x=0;x<c->width;x++) {
                picture->data[0][y * picture->linesize[0] + x] = x + y + i
* 3;
            }
        }

        /* Cb and Cr */
        for(y=0;y<c->height/2;y++) {
            for(x=0;x<c->width/2;x++) {
                picture->data[1][y * picture->linesize[1] + x] = 128 + y +
i * 2;
                picture->data[2][y * picture->linesize[2] + x] = 64 + x +
i * 5;
            }
        }

        /* encode the image */
        out_size = avcodec_encode_video(c, outbuf, outbuf_size, picture);
        printf("encoding frame %3d (size=%5d)\n", i, out_size);
        fwrite(outbuf, 1, out_size, f);
    }

    /* get the delayed frames */
    for(; out_size; i++) {
        fflush(stdout);

        out_size = avcodec_encode_video(c, outbuf, outbuf_size, NULL);
        printf("write frame %3d (size=%5d)\n", i, out_size);
        fwrite(outbuf, 1, out_size, f);
    }

    /* add sequence end code to have a real mpeg file */
    outbuf[0] = 0x00;
    outbuf[1] = 0x00;
    outbuf[2] = 0x01;
    outbuf[3] = 0xb7;
    fwrite(outbuf, 1, 4, f);
    fclose(f);
    free(picture_buf);
    free(outbuf);

    avcodec_close(c);
    av_free(c);
    av_free(picture);
    printf("\n");
}

/*
 * Video decoding example
 */

void pgm_save(unsigned char *buf,int wrap, int xsize,int ysize,char
*filename)
{
    FILE *f;
    int i;

    f=fopen(filename,"w");
    fprintf(f,"P5\n%d %d\n%d\n",xsize,ysize,255);
```

```
    for(i=0;i<ysize;i++)
        fwrite(buf + i * wrap,1,xsize,f);
    fclose(f);
}

void video_decode_example(const char *outfilename, const char *filename)
{
    AVCodec *codec;
    AVCodecContext *c= NULL;
    int frame, size, got_picture, len;
    FILE *f;
    AVFrame *picture;
    uint8_t inbuf[INBUF_SIZE + FF_INPUT_BUFFER_PADDING_SIZE], /*(4096+8
bytes)*/
 *inbuf_ptr; //FF_INPUT_BUFFER_PADDING_SIZE = 8 bytes
    char buf[1024];

    /* set end of buffer to 0 (this ensures that no overreading happens
for damaged mpeg streams) */
    memset(inbuf + INBUF_SIZE, 0, FF_INPUT_BUFFER_PADDING_SIZE);

    printf("Video decoding\n");

    /* find the mpeg1 video decoder */
    codec = avcodec_find_decoder(CODEC_ID_MPEG4);
    if (!codec) {
        fprintf(stderr, "codec not found\n");
        exit(1);
    }

    c= avcodec_alloc_context();
    picture= avcodec_alloc_frame();

    if(codec->capabilities&CODEC_CAP_TRUNCATED)
        c->flags|= CODEC_FLAG_TRUNCATED; /* we dont send complete frames
*/

    /* for some codecs, such as msmpeg4 and mpeg4, width and height
       MUST be initialized there because these info are not available
       in the bitstream */

    /* open it */
    if (avcodec_open(c, codec) < 0) {
        fprintf(stderr, "could not open codec\n");
        exit(1); //means abnormal termination
    }

    /* the codec gives us the frame size, in samples */

    f = fopen(filename, "rb");
    if (!f) {
        fprintf(stderr, "could not open %s\n", filename);
        exit(1);
    }

    frame = 0;
    for(;;) {
        size = fread(inbuf, 1, INBUF_SIZE, f);
        if (size == 0)
            break;
```

```
    /* NOTE1: some codecs are stream based (mpegvideo, mpegaudio)
       and this is the only method to use them because you cannot
       know the compressed data size before analysing it.

       BUT some other codecs (msmpeg4, mpeg4) are inherently frame
       based, so you must call them with all the data for one
       frame exactly. You must also initialize 'width' and
       'height' before initializing them. */

    /* NOTE2: some codecs allow the raw parameters (frame size,
       sample rate) to be changed at any frame. We handle this, so
       you should also take care of it */

    /* here, we use a stream based decoder (mpeg1video), so we
       feed decoder and see if it could decode a frame */
    inbuf_ptr = inbuf;
    while (size > 0) {
        len = avcodec_decode_video(c, picture, &got_picture,
                                   inbuf_ptr, size);
        if (len < 0) {
            fprintf(stderr, "Error while decoding frame %d\n", frame);
            exit(1);
        }
        if (got_picture) { /*if a complete frame has been read*/
            printf("saving frame %3d\n", frame);
            fflush(stdout);

            /* the picture is allocated by the decoder. no need to
               free it */
            snprintf(buf, sizeof(buf), outfilename, frame);
            pgm_save(picture->data[0], picture->linesize[0],
                     c->width, c->height, buf);
            frame++;
        }
        size -= len;
        inbuf_ptr += len;
    }
}

/* some codecs, such as MPEG, transmit the I and P frame with a
   latency of one frame. You must do the following to have a
   chance to get the last frame of the video */
len = avcodec_decode_video(c, picture, &got_picture,
                           NULL, 0);
if (got_picture) {
    printf("saving last frame %3d\n", frame);
    fflush(stdout);

    /* the picture is allocated by the decoder. no need to
       free it */
    snprintf(buf, sizeof(buf), outfilename, frame);
    pgm_save(picture->data[0], picture->linesize[0],
             c->width, c->height, buf);
    frame++;
}

fclose(f);

avcodec_close(c);
av_free(c);
av_free(picture);
```

```
    printf("\n");
}

int main(int argc, char **argv)
{
    const char *filename;

    /* must be called before using avcodec lib */
    avcodec_init();

    /* register all the codecs (you can also register only the codec
       you wish to have smaller code */
    avcodec_register_all();

    if (argc <= 1) {
        //audio_encode_example("/tmp/test.mp2");
        //audio_decode_example("/tmp/test.sw", "/tmp/test.mp2");

        //video_encode_example("/tmp/test.mpg");
        filename = "/root/trcd-vdo/normal_call._TX_muxvideo_1.bits";
  //filename = "/root/trcd-vdo/videotrace.bits";
    } else {
        filename = argv[1];
    }

    //    audio_decode_example("/tmp/test.sw", filename);
    video_decode_example("/root/dec/keyboard/kb%d.pgm", filename);
//video_decode_example("/root/dec/wood/wood%d.pgm", filename);

    return 0;
}
```

Appendix C2: C source code for mp4_sc.c

```
/*This program parses the MPEG-4 traced/encoded file and displays the
number of various start codes present in MPEG-4 visual bit-stream*/
/*********************************************************************/
#include<stdio.h>

main()
{
FILE *ptrvar;
/*Start codes*/
int prefix=0, vdo_sc=0, vdol_sc=0, vos_sc=0, vos_ec=0, ud_sc=0, govop_sc=0,
vs_erc=0, vo_sc=0, vop_sc=0, fo_sc=0, fop_sc=0, mo_sc=0, mop_sc=0,
sto_sc=0, tsl_sc=0, tsnl_sc=0, ssc=0;
long rcount=0;
unsigned char hx;
unsigned char c[4]={0};

/*This file is MPEG-4 simple profile @ level 0 and has been traced during
a 3G video call*/
//ptrvar=fopen("/home/engrzshan/Documents/trcd-
vdo/normal_call._TX_muxvideo_1.bits", "r");

/*The following files have been encoded by FFmpeg without making any
modifications. These files seem to be MPEG-4 simple profile @ level 1*/

//ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_fm.m4v", "r");
//ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_cp.m4v", "r");
//ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_akiyo.m4v", "r");


/*The following files have been encoded by FFmpeg after making
modifications for simple profile @ level 0 as suggested by 3gpp*/

//ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_fm1.m4v", "r");
//ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_cp1.m4v", "r");
ptrvar=fopen("/home/engrzshan/Documents/enc-vdo/ffmpeg_akiyo1.m4v", "r");



  if(ptrvar==NULL)
    printf("\n ERROR - Cannot open the designated file \n");
  else
    do
    {
    hx=fgetc(ptrvar);
    rcount++;
    //fread(c,1,4,ptrvar);
    //i[0]=c[0];
    //i[1]=c[1];
    //i[2]=c[2];
    //i[3]=c[3];

    printf("%x", hx);
    //printf("%x","%x","%x","%x",i[0],i[1],i[2],i[3]);
    if(hx==0x00)
    {hx=fgetc(ptrvar);
    printf("%x", hx);
    rcount++;
```

```
    if(hx==0x00)
    {hx=fgetc(ptrvar);
  printf("%x", hx);
    rcount++;
      if(hx==0x01)
      {
      prefix++;
      hx=fgetc(ptrvar);
      printf("%x", hx);
      rcount++;
        if(hx<=0x1F)
        {printf("%x", hx);
        vdo_sc++;}
        if((hx>=0x20)&& (hx<=0x2F))
        {printf("%x", hx);
        vdol_sc++;}
        if(hx==0xB0)
        {printf("%x", hx);
        vos_sc++;}
        else if(hx==0xB1)
        {printf("%x", hx);
        vos_ec++;}
        else if(hx==0xB2)
        {printf("%x", hx);
        ud_sc++;}
        else if(hx==0xB3)
        {printf("%x", hx);
        govop_sc++;}
        else if(hx==0xB4)
        {printf("%x", hx);
        vs_erc++;}
        else if(hx==0xB5)
        {printf("%x", hx);
        vo_sc++;}
        else if(hx==0xB6)
        {printf("%x", hx);
        vop_sc++;}
        else if(hx==0xBA)
        {printf("%x", hx);
        fo_sc++;}
        else if(hx==0xBB)
        {printf("%x", hx);
        fop_sc++; }
        else if(hx==0xBC)
        {printf("%x", hx);
        mo_sc++;}
        else if(hx==0xBD)
        {printf("%x", hx);
        mop_sc++;}
        else if(hx==0xBE)
        {printf("%x", hx);
        sto_sc++;}
        else if(hx==0xBF)
        {printf("%x", hx);
        tsl_sc++;}
        else if(hx==0xC0)
        {printf("%x", hx);
        tsnl_sc++;}
        else if((hx >= 0xC1)&&(hx <= 0xC5))
        {printf("%x", hx);
        ssc++;}
```

```
            }
        }
    }

    }while(rcount<26764);//File size: mot=24852,114255,182716, ak=26807,
cp=28828, fm=29738,ak1=26764, cp1=28740, fm1=29673
printf("\n rcount = %4d \n prefix = %4d \n vdo_sc = %4d  \n vdol_sc = %4d
\n vos_sc = %4d  \n vos_ec = %4d  \n ud_sc = %4d  \n govop_sc = %4d  \n
vs_erc = %4d  \n vo_sc = %4d  \n vop_sc = %4d  \n fo_sc = %4d  \n fop_sc =
%4d  \n mo_sc = %4d  \n mop_sc = %4d \n sto_sc = %4d \n tsl_sc = %4d \n
tsnl_sc = %4d \n ssc = %4d \n" , rcount, prefix, vdo_sc, vdol_sc, vos_sc,
vos_ec, ud_sc, govop_sc, vs_erc, vo_sc, vop_sc, fo_sc, fop_sc, mo_sc,
mop_sc, sto_sc, tsl_sc, tsnl_sc, ssc);
fclose(ptrvar);
}
```