

Implementation of a Personal Digital Radio
Recorder for Digital Multimedia Broadcasting by
Adapting the Open-Source Personal Digital
Video Recorder Software MythTV

Communications Research Center Canada
Written by Jean-François Roy, Université Laval

May 2006

Copyright © 2006 Jean-François Roy, All Rights Reserved.

This work is licensed under the Creative Commons Attribution-NoDerivs 2.5 Canada License.

To view a copy of this license, visit <http://creativecommons.org/licenses/by-nd/2.5/ca/>
or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California,
94105, USA.

Abstract

This report presents the technical and implementation details of the adaptation of the open-source personal video recorder software MythTV to Digital Multimedia Broadcasting. Such an integration aims at creating a “personal digital radio recorder” capable of recording audiovisual programs broadcasted using DMB technology. The primary objective was to add minimal bitstream reception and playback support to MythTV. Consequently, further research will have to address a number of additional issues, such as Electronic Program Guide information transport over DMB and its reception, as well as MPEG-4 SL support in ffmpeg and tuning capabilities, in order to arrive at a complete and viable system that complies with applicable international standards.

Topics that will be covered herein include Digital Multimedia Broadcasting, the PDRR project, MythTV’s architecture, the H.264 video codec, implementation details for the modifications made to MythTV and research recommendations for the continuation of the PDRR project.

Contents

Foreword	vi
Acknowledgements	vii
Abbreviations	viii
1 Introduction	1
2 Digital Multimedia Broadcasting Overview	2
2.1 DMB Architecture	2
2.1.1 DMB Transport Layer	5
2.1.2 DMB Application Layer	5
2.2 DMB Codecs	5
2.2.1 MPEG-4 Part 3: Audio	6
2.2.2 H.264 / MPEG-4 Part 10: AVC	7
3 The PDRR Project	8
3.1 Conceptual Architecture	8
3.1.1 CRC Dataport	8
3.1.2 PDRR Host	10
3.2 Research Objectives	10
3.3 Lab Environment	10
3.3.1 Hardware	10
3.3.2 Software	10
3.3.3 Methodology	11
3.4 Analysis of Previous Efforts	11
3.4.1 Summary of Gallant's Research Line	11
3.4.2 Rejecting the Virtual Driver Approach	12
4 The MythTV Architecture	14
4.1 Back-end and Front-end	14
4.2 The <i>ProgramInfo</i> Class	14
4.2.1 Recording Markup	15
4.3 Recording Subsystem	15
4.3.1 <i>TVRec</i>	15

4.3.2	Recorders	16
4.3.3	Channels	17
4.4	Playback Subsystem	17
4.4.1	<i>TV</i>	17
4.4.2	<i>NuppelVideoPlayer</i>	17
4.4.3	Audio and Video Outputs	18
4.4.4	Decoders	19
4.4.5	<i>AvFormatDecoder</i>	19
4.4.6	Typical Playback Sequence	20
4.5	The MythTV Database	20
4.5.1	<i>capturecard</i> Table	21
4.5.2	<i>videosource</i> Table	21
4.5.3	<i>cardinput</i> Table	22
4.5.4	<i>channel</i> Table	22
4.5.5	Miscellaneous Tables	23
5	ITU-T Rec. H.264 ISO/IEC 144496-10 AVC	24
5.1	Basics of Video Compression	25
5.1.1	Chroma Subsampling	25
5.1.2	Macroblocks	25
5.1.3	Exploiting Spatial Statistical Dependencies	25
5.1.4	Exploiting Temporal Statistical Dependencies	26
5.1.5	Further Entropy Reduction	26
5.2	Improvements Over Previous Video Codecs	26
5.2.1	Prediction Improvements	27
5.2.2	Coding Improvements	29
5.2.3	Error Resilience and Flexibility Improvements	29
5.3	Structure of a Bitstream	30
5.3.1	Basic Definitions	30
5.3.2	Video Coding Layer	31
5.3.3	Network Abstraction Layer	32
5.4	Profiles and Levels	34
6	Implementation Details of MythTV Modifications	35
6.1	The IP Network Recorder	35
6.1.1	Architecture	36
6.1.2	Limitations	36
6.2	H.264 Support	37
6.2.1	H.264 Utilities	37
6.2.2	<i>AvFormatDecoder</i> Modifications	38
6.2.3	Miscellaneous Modifications	40
6.2.4	Limitations	40
7	Conclusions	41

8 Recommendations	42
8.1 Address Relevant ffmpeg Limitations	42
8.1.1 Implement MPEG-4 SL Demultiplexing	42
8.1.2 Improve AAC Decoding	42
8.2 Implement Dataport Control In MythTV	43
8.3 Implement Correct H.264 AU Delimitation	43
8.4 Implement DMB EPG Data Loading	43
A Amendment 1	44

List of Figures

2.1	Digital Multimedia Broadcasting's layered architecture	3
2.2	OSI reference model for networking protocols [22]	4
2.3	Simplified representation of SBR [25]	7
3.1	Conceptual architecture of a complete PDRR DMB chain	9
3.2	MPEG-2 TS encapsulation in UDP/IP	9
4.1	Class relationships within the recording subsystem	16
4.2	Doxygen class hierarchy for <i>RecorderBase</i>	17
4.3	Doxygen class hierarchy for <i>ChannelBase</i>	17
4.4	Class relationships within the playback subsystem	18
4.5	Doxygen class hierarchy for <i>AudioOutput</i>	18
4.6	Doxygen class hierarchy for <i>VideoOutput</i>	19
4.7	Doxygen class hierarchy for <i>DecoderBase</i>	19
4.8	Sequence diagram of playback subsystem	20
5.1	Evolution of ITU-T and MPEG video coding standards [12]	24
5.2	Simplified block diagram of intra-picture encoding in H.264 [12]	26
5.3	Simplified block diagram of inter-picture coding in H.264 [12]	27
5.4	Frame dependency comparison [10]	28
5.5	Reliable parameter set exchange [16]	30
5.6	Subdivision of a picture into slices when not using FMO [16]	32
5.7	Structure of an access unit [4]	33
5.8	H.264 profiles [10]	34
6.1	Updated Doxygen class hierarchy for <i>RecorderBase</i>	36
6.2	Simplified state diagram of <i>KeyframeSequencer</i>	38

List of Tables

2.1	DMB profiles	6
-----	------------------------	---

Foreword

Communications Research Center Canada has had and continues to have an important role in the development of new communication technologies. From advancements in radio telecommunication, psycho-acoustic coding (such as MPEG-1/2 Audio Layer II and III) and HDTV to more recent efforts in rich media radio broadcasting, CRC's contributions to science and technology are indeed numerous. It is therefore with great pride that I bring you this report detailing the research I conducted during my work term as a junior research software engineer in the Radio Broadcast Systems and Transmission group.

There should be little doubt left in our minds that the broadcasting landscape is presently undergoing a dramatic transformation. Ever since broadband and highly efficient coding algorithms became widely available and usable during the second half of the last decade, new ways of distributing content have appeared that have revolutionized how people can access culture and information, much to the horror of the existing infrastructure who is seeing its control over content distribution slip away into the hands of others.

But beyond these social issues, we have also seen rich media appear in unexpected places, beginning with the Internet and now on mobile devices. Indeed, there already exists a number of services to receive news, songs or even television on mobile phones, and that is only the beginning. Thus, it comes as little surprise that broadband radio is also expanding its offerings by using new digital radio technologies such as DMB and DVB-H to deliver better audio and even video.

Yet that new content is of little value without applications to take advantage of it. This is where the idea of a personal digital radio recorder comes in. Imagine being able to record your favorite shows on radio just as easily as you can subscribe to podcasts or program your DVR to record television without having to worry about channels and program grids. Such a system would tremendously increase the value of digital radio, which is more than ever hard-pressed to deliver a better experience to its customers.

Acknowledgments

I would like to acknowledge the following people for their help and support during my work term and the writing of this report.

- **Adam Bertrand**, for your overall support and friendship.
- **François Lefèbvre**, for giving me the extraordinary chance of working at CRC and bearing with my somewhat optimistic time predictions. I truly appreciate your trust in me and value the experience and knowledge you have given me.
- **Jean-Michel Bouffard**, for all the help, experience and knowledge you gave me during the 4 months we worked together. You were a source of inspiration as a young engineer making significant contributions to technology.
- **Josh Hattersley**, for your expert counsel on all manners of English grammar and style issues.
- **Justin Olbrantz**, for proof-reading for me on several occasions and for your relentless attempts to distract me.
- **Pascal Charest**, for the numerous discussions we had about technology, science and software engineering. You continually challenged my views and assumptions, and drove me to improve myself and my work. Of course, I am also extremely grateful for your expertise.

Abbreviations

AAC Advanced Audio Coding

ATSC Advanced Television Systems Committee

AVC Advanced Video Coding

B-frame Bi-predictive Frame

BIFS Binary Format for Scenes

BSAC Bit-Sliced Arithmetic Coding

CABAC Context-based Adaptive Binary Arithmetic Coding

CAVLC Context-based Adaptive Variable Length Coding

CRC Communications Research Center Canada or Cyclic Redundant Check

DAB Digital Audio Broadcasting

DMB Digital Multimedia Broadcasting

DTS Decoding Time Stamp

DVB Digital Video Broadcasting

DVR Digital Video Recorder

ES Elementary Stream

ETSI European Telecommunications Standards Institute

FMO Flexible Macroblock Ordering

GOP Group Of Pictures

IDR Instantaneous Decoding Refresh

IEC International Electrochemical Commission

I-frame Intra Frame

IP Internet Protocol

ISO International Organization for Standardization

ITU International Telecommunication Union

JVT Joint Video Team

MPEG Moving Picture Experts Group

NAL Network Abstraction Layer

NTSC National Television Systems Committee

P-frame Predictive Frame

PAL Phase-Alternating Line

PAT Program Association Table

PDRR Personal Digital Radio Recorder

PES Packetized Elementary Stream

PMT Program Map Table

PPS Picture Parameter Set

PTS Presentation Time Stamp

PVR Personal Video Recorder

RTP Real-time Transport Protocol

RTSP Real Time Transport Protocol

SEI Supplemental Enhancement Information

SL Synchronization Layer

SPS Sequence Parameter Set

TCP Transmission Control Protocol

TS Transport Stream

UDP User Datagram Protocol

V4L2 Video For Linux 2

VCEG Video Coding Experts Group

VCL Video Coding Layer

VLC Video LAN Client

1

Introduction

Digital radio is not a new technology in the sense that the methods and systems it uses were defined years ago. However, for various reasons, it is only beginning to appear in mainstream markets, at least in North America. Nevertheless, research centers like CRC have been diligently working on developing applications and services that will take advantage of the vastly superior capabilities digital radio offers over conventional analog broadcasting. Such an application is the digital radio equivalent of the massively popular PVR appliance, which allows its users to easily record their favorite television shows and offers an enhanced viewing experience with features such as time shifting (pausing and resuming a live broadcast without losing content) and automatic commercial skipping.

Presented herein are the technical and implementation details of the adaptation of the open-source PVR software MythTV to Digital Multimedia Broadcasting, the ETSI standard for carrying rich media over Digital Audio Broadcasting. Indeed, MythTV makes the assumption that it will be used with television capture hardware for current analog and digital television standards. However, it also offers an amazingly rich set of features and a solid architecture which made it an ideal candidate for the PDRR project nonetheless.

This report begins with an overview of Digital Multimedia Broadcasting, focusing on the top-most layers defining the various codecs that are allowed in DMB services. Second, it introduces the PDRR project's architecture and main components in order to set the stage for the following chapters. It then moves on to an in-depth discussion of MythTV's architecture, including a close look at the major software classes that drive MythTV's recording and playback subsystems. Next, this report brings forward important technical information about the H.264 video codec to support the ensuing chapter on implementation details of the modifications that were made to MythTV. Finally, it ends with the research's conclusions and a series of recommendations for the continuation of the PDRR project.

2

Digital Multimedia Broadcasting Overview

Digital Multimedia Broadcasting (DMB) describes a digital transmission system capable of carrying synchronized audio, video and data content for mobile devices. It is an extension of the EUREKA 147 standard, more commonly known as Digital Audio Broadcasting (DAB). In a manner similar to DVB, DMB can operate via satellite (S-DMB) or terrestrial (T-DMB) transmission. The European Telecommunications Standards Institute ratified DMB in 2005 (see [3]). Because this report's main subject matter is a software-based PDRR, there are no discussions on DAB and very little on sections of the DMB specification that deal with physical technicalities such as radio frequency transmission and error correction. Readers are encouraged to read the relevant literature (such as [13] for an introduction to DAB) if they wish to familiarize themselves with those topics.

2.1 DMB Architecture

Digital Multimedia Broadcasting uses a layered architecture which specifies how source content should be encoded (compressed), multiplexed, packetized, error-protected and transmitted using DAB technology. Figure 2.1¹ presents a simplified version of this architecture. One can immediately draw a parallel between it and the Open System Interconnection reference model for networking protocols, shown in Figure 2.2. Indeed, both share the fundamental concept and requirement that each layer must be oblivious to the previous one. This allows the construction of vertical protocol stacks from common foundations adapted to the needs of different applications.

¹My thanks to Pascal Charest for the figure.

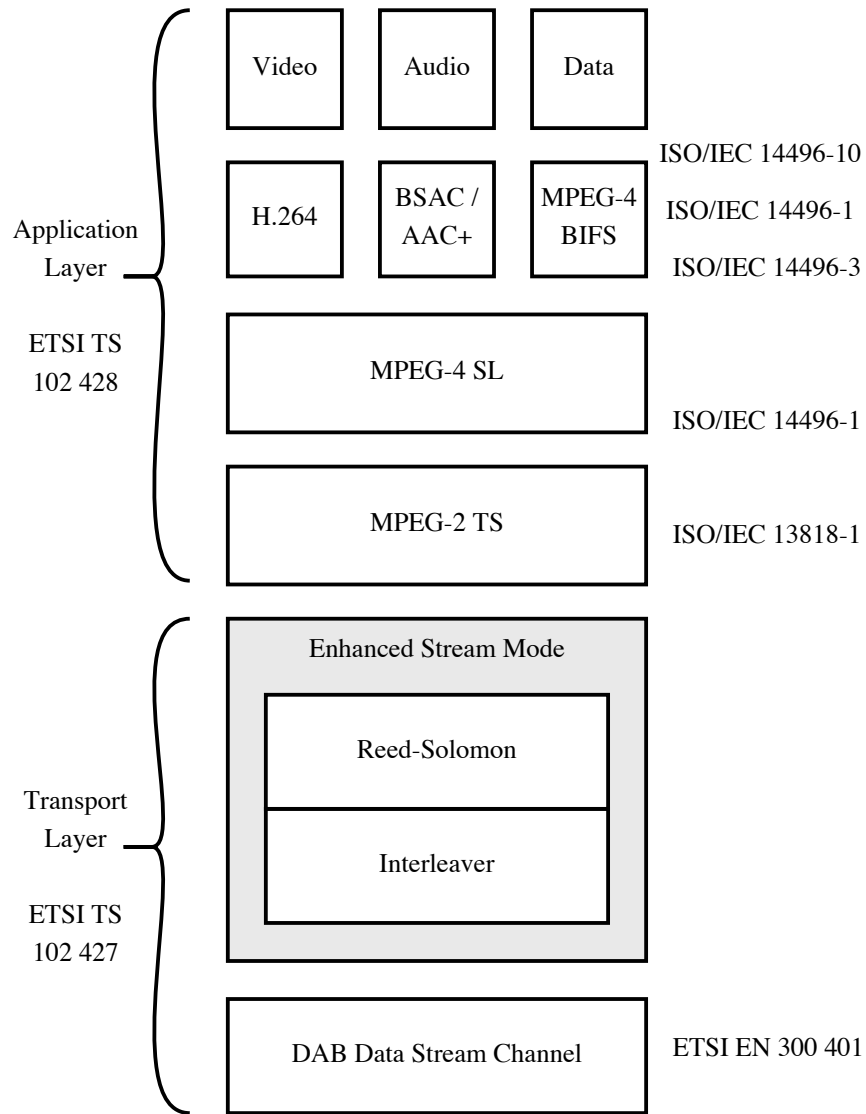


Figure 2.1: Digital Multimedia Broadcasting's layered architecture

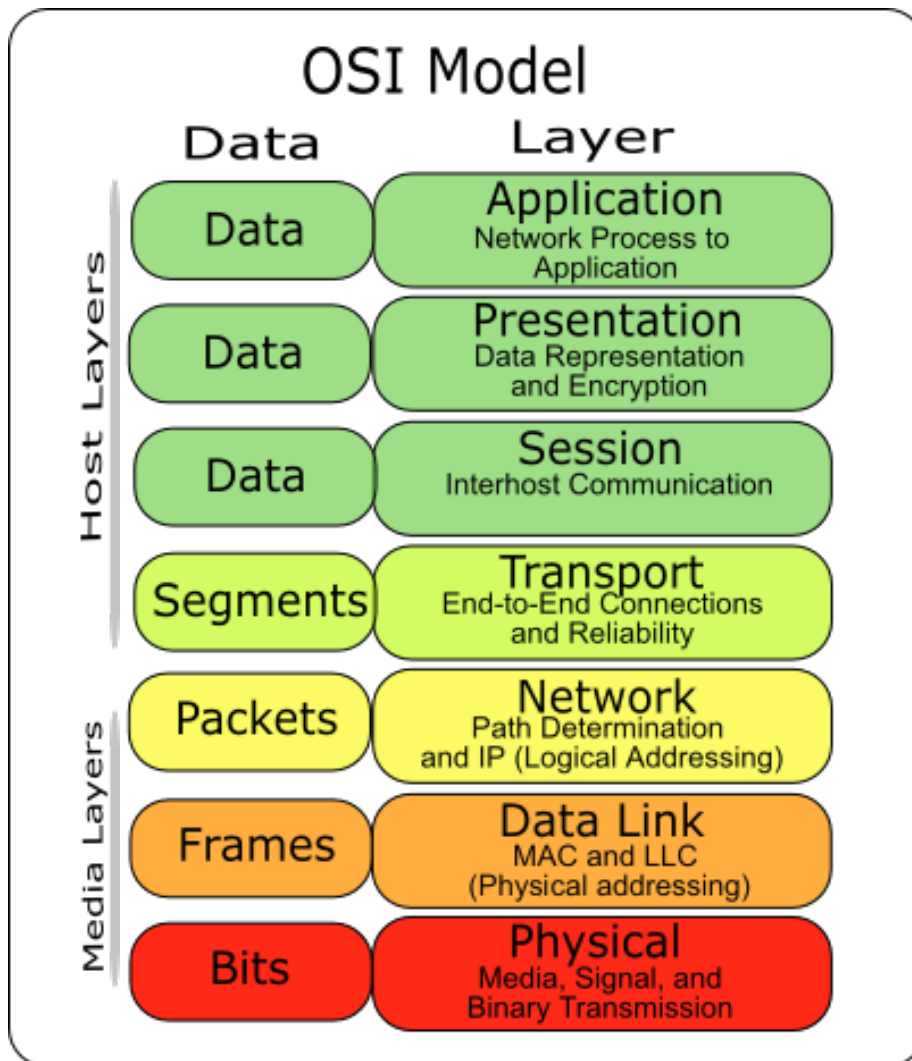


Figure 2.2: OSI reference model for networking protocols [22]

2.1.1 DMB Transport Layer

The Transport Layer represents the methods and systems used to transmit the Application Layer data. At the bottom of the TL stack, DMB uses an MSC stream mode sub-channel which allows data to be transmitted transparently from source to destination (see [1, 13] for details). As for the enhanced stream mode block, it represents the methods and systems ratified by the DAB standard (see [2]) to encapsulate MPEG-2 Transport Stream within a DAB MSC stream data sub-channel. It essentially adds two error protection strategies on top of regular stream mode: Reed-Solomon error correction coding (see [24] for details) and byte interleaving (distributing errors to improve Reed-Solomon efficiency).

2.1.2 DMB Application Layer

The Application Layer represents the methods and systems used to packetize, composite and encode content that is to be distributed with DMB. Because they are of particular importance, the top-layer codecs will be discussed further in Section 2.2.

As can be seen in Figure 2.1, MPEG-2 Transport Stream (as defined in ISO/IEC 13818-1 [5]) is used to packetize and multiplex the multimedia content. Although MPEG-2 TS can technically carry video and audio elementary streams and keep them synchronized within the context of a program (a set of elementary streams sharing a common time base), it lacks the flexibility and power necessary to describe complex scene compositions. For instance, MPEG-2 TS is incapable of describing a program where two distinct video streams play at the same time but at different spacial coordinates. Thus, DMB relies on the far more capable MPEG-4 Synchronization Layer (as defined in ISO/IEC 14496-1 [8]) to composite elementary streams and keep them synchronized. As such, the MPEG-2 TS layer is used to transport MPEG-4 SL packets, which themselves transport the ES and MPEG-4 BIFS data. It should be noted that strict restrictions are enforced on the MPEG-2 TS and MPEG-4 SL layers by the DMB specification, so not all features of those two extensive standards can be used in valid DMB bitstreams. Because MPEG-4 scenes are very complex, readers are encouraged to read supplemental material on the topic, such as Chapter 4 in [15].

2.2 DMB Codecs

DMB has chosen MPEG-4 Part 3 and H.264 / Advanced Video Coding as the audio and video codecs that are to be used by DMB services, respectively. Since these are international standards, it ensures that any device manufacturer or equipment vendor has equal and fair access to their specifications, and that DMB products from different origins easily inter-operate. They are also extremely efficient and deliver impressive results at bit rates suitable for digital radio broadcasts (see Sections 2.2.1 and 2.2.2). Finally, in a manner similar to that of the MPEG group, the DMB specification defines 2 DMB profiles that limit

the set of features from MPEG-4 Part 3 and H.264 / AVC that are allowed in DMB broadcasts. Table 2.1 lists the profiles along with associated video and audio codecs. For full details on the profiles, refer to Chapter 8 in [3].

Table 2.1: DMB profiles

Profile Number	Video Codec	Audio Codec
1	H.264 / MPEG-4 AVC	MPEG-4 ER-BSAC
2	H.264 / MPEG-4 AVC	MPEG-4 HE-AAC v2

2.2.1 MPEG-4 Part 3: Audio

ISO/IEC 13818-7 (MPEG-2 Part 7) first introduced Advanced Audio Coding, however the methods and systems it describes have since been enhanced and expanded in ISO/IEC 14496-3 (MPEG-4 Part 3 [6], see [17] for an overview and Chapter 11 in [15] for a more complete discussion). Indeed, as perceptual audio coding technology developed, new algorithms and techniques were added to the basic AAC codec (now known as Low Complexity AAC), resulting in a number of different but related codecs. Initially, DMB selected only one such derivative called BSAC. However, a second profile was added to the DMB specification in which HE-AAC must be used instead. Both of these variants offer far better performances at low bit rates than LC-AAC or MPEG-1/2 Audio Layer III and were chosen for that reason.

BSAC

LC-AAC uses normal Huffman coding in its lossless entropy reduction stage. BSAC instead uses Bit-Sliced Arithmetic Coding (thus the codec's common name) to reduce bit rates significantly. BSAC can be used from 16 kbps up to 64 kbps, although 40 kbps is a more practical lower bound. Clause 8.1.1 and Table 8 in [3] detail restrictions on BSAC coding for DMB Profile 1.

HE-AAC

High-Efficiency AAC is a relatively new addition to MPEG-4 Part 3 and was known as aacPlus before it was standardized by the MPEG group (see [6, 7]). HE-AAC uses spectral band replication (see Figure 2.3 and [25]) and parametric stereo (see [23]) to dramatically improve its bit rate efficiency. SBR is a technique that takes advantage of the fact that high frequencies contribute less than low frequencies to the acoustic envelope of a signal. Therefore, an encoder using SBR will encode low frequencies and SBR information and cut the high frequencies. An SBR-aware decoder can then use this information to re-create the high frequencies. As for parametric stereo, it's a technique that exploits the typically high redundancy between the left channel and the right channel where the encoder will encode a difference or sum of the two channels plus additional

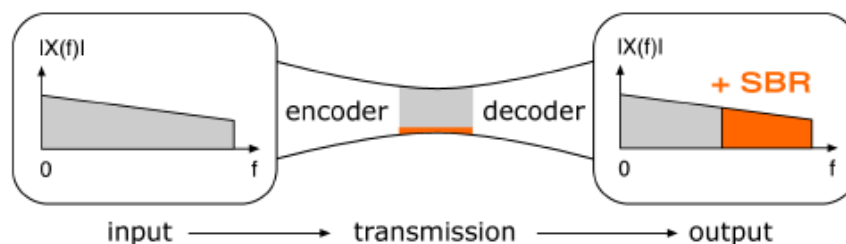


Figure 2.3: Simplified representation of SBR [25]

PS information that a PS-aware decoder can then use to re-create the original channels. HE-AAC bit rates are typically between 32 kbps and 64 kbps. Clause 8.2.1 and Tables 10 and 11 in [3] detail restrictions on HE-AAC coding for DMB Profile 2.

2.2.2 H.264 / MPEG-4 Part 10: AVC

H.264 / AVC² is a recent video codec that represents an important evolution in macroblock-based video coding systems. At the cost of significantly more complex encoding and decoding processes, H.264 offers substantial quality and efficiency improvements over ISO/IEC 14496-2 (MPEG-4 Video or Natural Video Coding) and ISO/IEC 13818-2 (MPEG-2 Video) in addition to much better scalability, from 500 kbps Internet streaming up to 10 Mbps High Definition video (see [12] and [14]). Clause 8.1.2 and Table 9 in [3] detail restrictions on H.264 coding for both DMB profiles (see Section 5.4 for details on H.264 profiles and levels).

²H.264 is the ITU-T recommendation series and number while AVC is the ISO/IEC name. Both are used interchangeably in this document.

3

The PDRR Project

The PDRR project was started at CRC in the summer of 2005. As was previously stated in the introduction, the ability to automatically record programs based on an electronic guide and to consume those programs in a flexible manner made the PVR an extremely popular device in television, and as rich audiovisual media makes its way to digital radio broadcasting, one can expect a similar type of device or application to be just as popular. Not only that, but one of RBSC's principal goal is to demonstrate and promote convergence for the way people receive and consume media and information distributed over broadcast networks, particularly in the context of mobile devices. Integrating digital radio and television in a single central application is an excellent way of achieving that objective.

3.1 Conceptual Architecture

In order to have a complete solution where one can broadcast a DMB multiplex at one end and record or watch programs in the PDRR at the other end, one needs to be able to broadcast a DMB bitstream, receive it, decode it (apply byte deinterleaving and inverse Reed-Solomon coding, see Section 2.1.1) and send the MPEG-2 TS stream to the PDRR (using UDP/IP over Ethernet for example). Figure 3.1 offers a graphical representation of this basic conceptual architecture.

Because this report is mainly concerned with the reception side of things, there will be no discussion on how one may broadcast a DMB multiplex. Therefore, if we take the modulator as granted, there are 2 unidentified components: the receiver / decoder and the PDRR host.

3.1.1 CRC Dataport

CRC has developed a device which is informally known as the dataport. It is an embedded PC running the Linux operating system coupled with a DAB

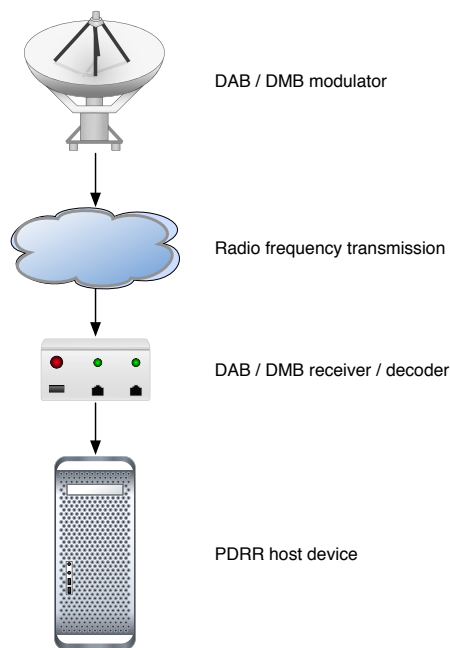


Figure 3.1: Conceptual architecture of a complete PDRR DMB chain

receiver card that allows the PC to search for DAB broadcasts, tune on an ensemble and receive data. Communication between the PC and the DAB card is managed by an application that was also developed by CRC. It is capable of querying the DAB card for its status and instruct it to perform actions such as a frequency scan. It also exposes these capabilities through a network-enabled telnet interface which is primarily designed to be used by a human being. Finally, it is capable of outputting selected DAB services, with full support for DMB decoding, to various destinations, such as a PC by using unicast UDP/IP. That particular method consists in encapsulating an integer number of MPEG-2 TS packets in UDP packets (see Figure 3.2).

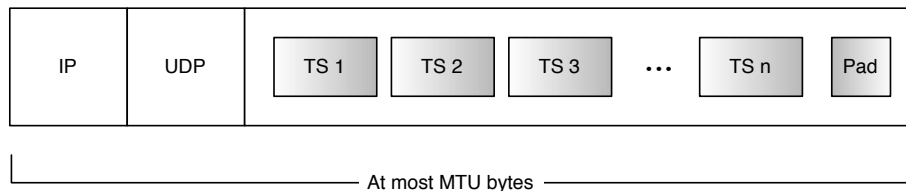


Figure 3.2: MPEG-2 TS encapsulation in UDP/IP

3.1.2 PDRR Host

Since MythTV was elected as the basis for building the PDRR, the PDRR host is any computer and operating system combination capable of running MythTV. The best choice at the time of writing was an x86-based PC running the Linux operating system. However, MythTV also runs on Mac OS X and Windows to varying degrees.

3.2 Research Objectives

The main objective of the research was to enable MythTV to record and playback an audiovisual MPEG-2 TS multiplex, streamed across a standard 100 Mbps LAN network using UDP/IP encapsulation (see Section 3.1.1), and conforming as much as possible to the DMB specification.

3.3 Lab Environment

This section describes the lab conditions in which the research was conducted.

3.3.1 Hardware

The following hardware configuration was used for the PDRR host machine:

- AMD Athlon™ 64 Processor 3200+
- 1024980 KB of RAM

3.3.2 Software

The following software was used:

- MythTV¹ compiled from Subversion² trunk.
- ffmpeg³ compiled from CVS trunk.
- x264⁴ compiled from Subversion trunk.
- VLC⁵ compiled from Subversion trunk.

¹<http://www.mythtv.org>

²<http://subversion.tigris.org/>

³<http://www.ffmpeg.org>

⁴<http://developers.videolan.org/x264.html>

⁵<http://www.videolan.org/>

3.3.3 Methodology

Test MPEG-2 TS bitstreams were sent to the PDRR host over unicast UDP/IP as described in Section 3.1.1, using VLC's streaming server capabilities. x264 was used for video encoding while VLC was used for audio encoding (MPEG-1 Audio Layer III and Dolby Digital were used instead of AAC, see Section 8.1.2) and MPEG-2 TS multiplexing. VLC uses ffmpeg's LAME⁶-based MP3 encoder behind the scenes. It should be noted that VLC's MPEG-2 TS multiplexer as well as ffmpeg's MPEG-2 TS demultiplexer (which is used by MythTV as described in Section 4.4.5) do not support MPEG-4 SL (see Section 8.1.1). All MythTV channels were marked as commercial free (see Section 4.5.4) to avoid commercial removal processing.

Not only was VLC more flexible than setting up a complete broadcast chain, it was also the sole option until the very end of the research, at which point Jean-Michel Bouffard finished implementing DMB support in the dataport's control software.

3.4 Analysis of Previous Efforts

Before the research presented in this report was conducted, another student named Luc Gallant worked on the PDRR project for a period of 4 months. His objectives were fairly similar to those stated in Section 3.2. This section analyzes his main work term report[11] recommendation pertaining to further development of the PDRR and why it was eventually dismissed in favor of the implementation detailed in Chapter 6.

3.4.1 Summary of Gallant's Research Line

Gallant developed a virtual V4L2 loop-back device driver based on the vloop-back⁷ project in order to transport DMB MPEG-2 TS bitstreams from a network source to MythTV. From his introduction:

The way PVR application grabs its video is through a video capture device, in most cases an actual hardware capture card. The driver type that is usually used for capture cards is of type V4L2, which provides certain functions and an API. Thus, it was necessary to create a virtual device driver that could be used to take the DMB video stream and send it to mythtv, the PVR software. [11]

This architecture requires that an application be written to feed the virtual device (see Figure 2 in [11]) and to possibly modify client programs to support capturing compressed frames through V4L2. In fact, because this feature of the V4L2 API is so new, not a single publicly available application supports it

⁶<http://lame.sourceforge.net/>

⁷<http://www.lavrsen.dk/twiki/bin/view/Motion/VideoFourLinuxLoopbackDevice>

yet. In addition, Gallant indicates in Section 4.1 of his report that the feeding application and the virtual device driver would also need to be updated:

Since the driver already supports the V4L2 specification, there are some straightforward changes that need to be done for the driver to support compressed video/audio. [...] For the fillerApp however, a few lines would need to be changed to ensure that the driver receives the correct video format when the fillerApp sets the format using the S_FMT ioctl. [11]

Consequently, Gallant recommends in his report to finish implementing the virtual device driver and to update both the feeding application and MythTV so that all 3 support the full V4L2 API, including compressed formats.

3.4.2 Rejecting the Virtual Driver Approach

It quickly became clear that this architecture was far more problematic than it first seemed. Below are the principal reasons why a different design was considered.

Driver development is harder

It is inherently harder to work on kernel-level software than on user space applications because mistakes can cause complete system crashes and because it is impossible to do conventional debugging. Indeed, one must go to great lengths to install and use virtualization software in order to have a stable development platform. In addition, debugging usually ends up being done with a ridiculous amount of console text output, as can be seen by the numerous debugging messages in vloopback. Also, kernel modules are far more likely to require maintenance as the kernel evolves than user space applications might, since they are able to rely on more abstract and stable APIs.

vloopback was not designed to support compressed formats

The loop-back driver used by Gallant as a starting point for his V4L2 virtual device driver was not designed at all for compressed formats, in particular MPEG-2 TS. Indeed, vloopback has a small number of frame buffers which are allocated when the filling application registers the image format it will use. Normally with uncompressed formats, such a design will perform acceptably. However, MPEG-2 TS packets are only 188 bytes long and so it becomes impossible to write only one MPEG-2 TS packet per buffer, since the required buffer cycling rate becomes far too large. Consequently, substantial changes to vloopback would have been necessary to keep the device driver responsive, such as storing an arbitrary number of bytes per buffer when MPEG-2 TS is used.

Significant changes to the filling application was required

The V4L2 structures used to describe compressed formats are tailored at MPEG codecs and contain a large number of fields, including video and audio codecs, bit rates and sampling rates. Normally, a capture application would fill these structures from user selected or default values and transmit them to some capture device. However, in the case of the virtual device driver, the filling application would have to read this information from the input bitstream, or feed a virtual device with dummy values and modify the virtual device driver to ignore invalid compression formats, none of which being clean or simple solutions.

KISS principle

Whenever an equivalent but simpler solution exists, it should take precedence over more complicated ones. Because compressed formats through V4L2 is not widely supported (if at all), even if a functional virtual device driver and filling application were developed, one would still need to update every single V4L2 capture client before getting functional playback. As such, focusing efforts exclusively on MythTV, which is arguably the true goal of the PDRR project, is a better engineering decision.

V4L2 is not cross-platform

V4L2 is a Linux-only API. Is it better software design to choose cross-platform solutions, especially when the advantages of a less portable one are not significant.

4

The MythTV Architecture

MythTV is a large and complex body of code organized in several libraries, sub-programs and plug-ins. This chapter does not aim at presenting the entire MythTV architecture, but rather to detail the components that are relevant to the objectives stated in Chapter 3's introduction and in Section 3.2. Particular attention is given to the recording and playback subsystems. In addition, MythTV's source code is heavily documented using Doxygen¹, and as such readers are encouraged to use this powerful tool for more detailed information on all the software components of MythTV.

4.1 Back-end and Front-end

Perhaps the most important separation of functionality in MythTV is that of *mythfrontend* and *mythbackend*. All recording and post-recording processing (such as commercials removal and transcoding) is done by a back-end, while a front-end manages the human graphical interface through which users will utilize a back-end's services. In addition, there can be multiple back-ends and front-ends in a given MythTV "site". For example, a particular "site" could have several back-ends in order to record a large number of channels simultaneously. When multiple back-ends are networked, one of them is promoted to master back-end, which manages and orchestrates the back-end network in addition to performing normal back-end operations. Of course, there is also a large amount of shared code across the various MythTV sub-programs, thus increasing maintainability, reliability and efficiency.

4.2 The *ProgramInfo* Class

The *ProgramInfo* class is the embodiment of a program throughout MythTV. Everything you may want to record or track about a particular program is done

¹<http://www.stack.nl/~dimitri/doxygen/>

through an instance of *ProgramInfo*. Through it, one can schedule a recording or query about past recordings, one can begin playing back a particular episode of that program, and so forth. Of particular interest to the research, *ProgramInfo* also gives access to each recording's markup.

4.2.1 Recording Markup

Several key subsystems in MythTV need to know about certain important locations within the bitstream of recorded programs. For example, the playback subsystem need to know about the position of key frames and commercials in order to properly fast forward and skip commercials. MythTV stores this information as marks, which are accessible using the *ProgramInfo* class. The key types of marks are:

- *MARK_COMM_START*, which indicates the beginning of a commercial.
- *MARK_COMM_END*, which indicates the end of a commercial.
- *MARK_GOP_BYFRAME*, which indicates a key frame beginning a group of pictures (in MPEG terminology).

Recording markup is stored in a table within the MythTV database (see Section 4.5 for details).

4.3 Recording Subsystem

Recording is handled by back-ends and essentially consists in capturing, processing, transcoding and storing programs, generally from some kind of television source such as analog cable television or ATSC / DVB. This subsystem is composed of a large number of software classes that cooperate to handle recording tasks. Figure 4.1 offers a simplified view of class relationships within the recording subsystem, where the arrows indicate ownership.

Front-ends are also involved in the recording subsystem to the extent that they need to communicate with back-ends so that the latter know what the user wants to watch or record and the former receive content to playback.

4.3.1 TVRec

The *TVRec* class is the arbiter of the recording subsystem. As indicated on Figure 4.1, recorder and channel objects (see Sections 4.3.2 and 4.3.3) belong to *TVRec* objects, in addition to several others such as *RingBuffer* and *ProgramInfo* instances. The *TVRec* class also possesses the knowledge about all available *RecorderBase* and *ChannelBase* sub-classes and for which device types (see Section 4.5.1) a particular *RecorderBase-ChannelBase* pair should be used. In other words, *TVRec*'s source code defines a tuple (*RecorderBase*, *ChannelBase*).

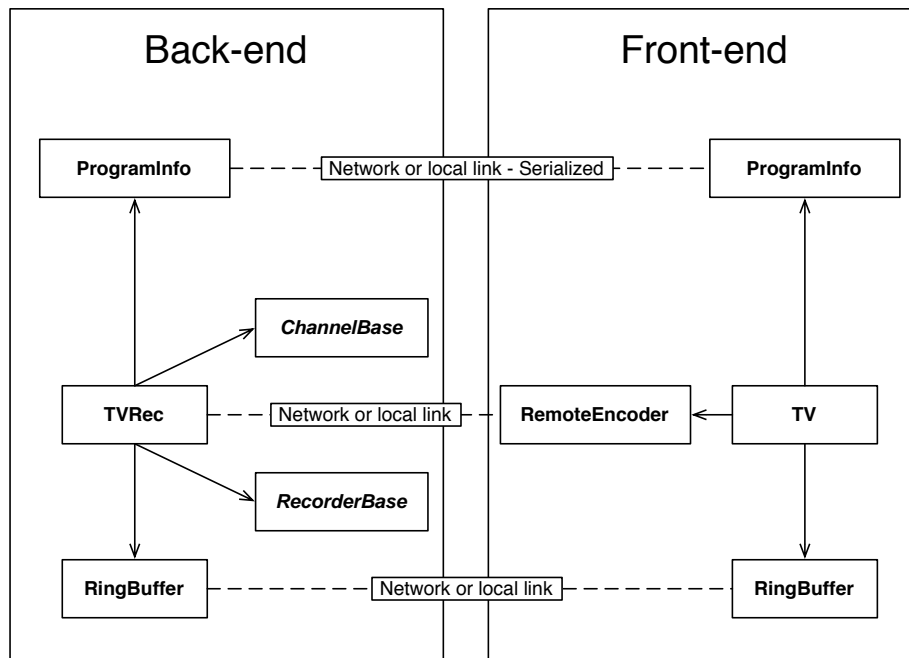
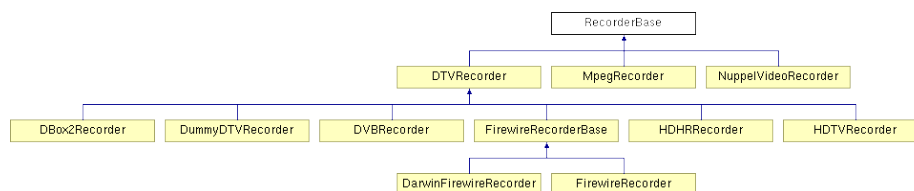


Figure 4.1: Class relationships within the recording subsystem

4.3.2 Recorders

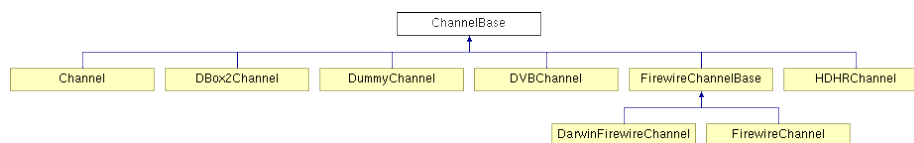
A recorder is essentially responsible for capturing a bitstream or a signal from a source and outputting it to a *RingBuffer* instance. Thereafter, that data will either be sent to a particular front-end to be played back, written on disk for future access or a combination of those and a few other actions. The *RecorderBase* abstract class formalizes the interface a concrete recorder class needs to implement. Intuitively, MythTV has a concrete sub-class of *RecorderBase* for each capture device type it supports. Figure 4.2 shows the class hierarchy of *RecorderBase* at the time of writing.

It is important to understand a recorder can get data any way it wants to, including interfacing with hardware devices, reading files or accessing network resources. This gives MythTV a fair amount of flexibility and a capacity to easily add compatibility for new hardware. Finally, recorders should attempt to mark the bitstream (see Section 4.2.1) they are recording as best they can. Indeed, the playback system will ask the recorder providing the bitstream being played (when MythTV is in LiveTV mode, e.g. watching live content) for marks, as well as accessing the markup table in the database.

Figure 4.2: Doxygen class hierarchy for *RecorderBase*

4.3.3 Channels

A channel is an abstraction designed to fully qualify a specific channel of a specific standard (NTSC, PAL, DVB, etc.). The *ChannelBase* abstract class formalizes the interface a concrete channel class needs to implement. Channel objects are used by MythTV to essentially tune capture hardware. Figure 4.3 shows the class hierarchy of *ChannelBase* at the time of writing.

Figure 4.3: Doxygen class hierarchy for *ChannelBase*

4.4 Playback Subsystem

Playback is handled by front-ends and essentially consists in demultiplexing, decoding and displaying a program or bitstream. This subsystem is composed of a large number of software classes that cooperate to handle playback tasks. Figure 4.4 offers a simplified view of class relationships within the playback subsystem, where the arrows indicate ownership. Upcoming sections offer more details on each of the represented class or group of classes.

4.4.1 TV

The *TV* class is the playback subsystem's manager. It is responsible for processing user events, such as changing the volume or the channel, and communicating with the other components of the playback subsystem as well as with the back-end providing the bitstream for the currently playing program, as indicated by Figure 4.1.

4.4.2 NuppelVideoPlayer

The *NuppelVideoPlayer* class is responsible for actually playing back a bitstream by orchestrating all the necessary components. Indeed, *NuppelVideoPlayer* cre-

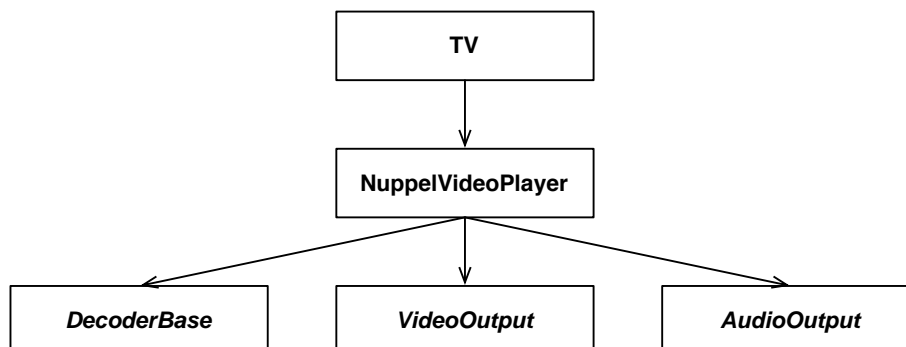
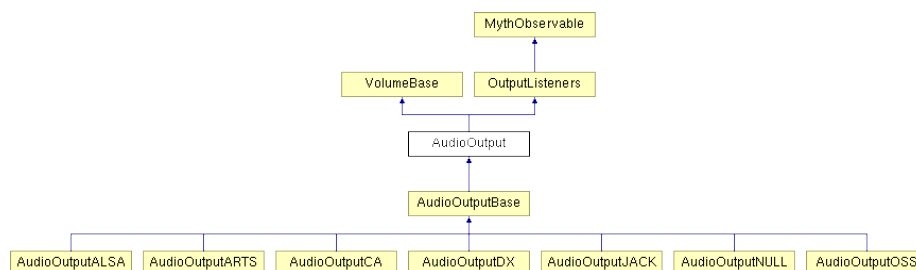


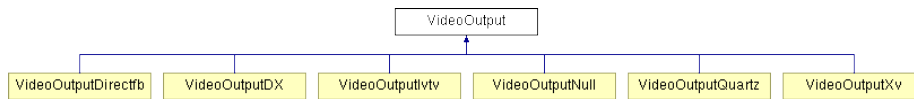
Figure 4.4: Class relationships within the playback subsystem

ates appropriate audio and video output instances (see Section 4.4.3), determines which decoder class is the most appropriate for a particular bitstream (see Section 4.4.4), and finally drives the playback by properly managing these objects.

4.4.3 Audio and Video Outputs

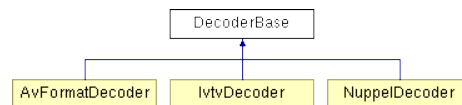
MythTV supports many different audio and video output methods and APIs through an abstraction represented in the source code by the *AudioOutput* and the *VideoOutput* abstract classes. They respectively formalizes the interface for an object capable of outputting audio and for an object capable of outputting video frames. This is a significant win in terms of hardware configuration flexibility (not all methods may be available on a certain piece of hardware) and host operating system support (Linux APIs are not available on Mac OS X, and so forth). Figure 4.5 shows the class hierarchy of *AudioOutput* while Figure 4.6 shows the class hierarchy of *VideoOutput* both at the time of writing.

Figure 4.5: Doxygen class hierarchy for *AudioOutput*

Figure 4.6: Doxygen class hierarchy for *VideoOutput*

4.4.4 Decoders

Decoders are objects capable of producing usable representations of compressed data such as audiovisual sequences. The interface that a decoder class needs to implement is formalized by the *DecoderBase* abstract class. In particular, decoders are capable of determining if they can or cannot decode a particular bitstream by looking at probing data read from that bitstream. *NuppelVideoPlayer* uses this capability to determine which decoder it will use behind the scenes when it is asked to open a bitstream for playback. This process is done by the `NuppelVideoPlayer::OpenFile()` method, which iterates over available decoder classes and calls the `DecoderBase::CanHandle()` method until it returns *true* or all decoders have been exhausted. If a decoder class is found, an instance is created and the `DecoderBase::OpenFile()` method is called. Figure 4.7 shows the class hierarchy of *DecoderBase* at the time of writing.

Figure 4.7: Doxygen class hierarchy for *DecoderBase*

4.4.5 *AvFormatDecoder*

As the name might suggest, *AvFormatDecoder* uses ffmpeg (see Section 3.3.2) to demultiplex and decode bitstreams. Therefore, it is capable of handling an incredible amount of containers and codecs. Of particular interest to the PDRR project, *AvFormatDecoder* is the only decoder that supports MPEG-2 TS demultiplexing. Therefore, if a recorder writes MPEG-2 TS packets to its *RingBuffer* instance, *AvFormatDecoder* will be used for that recording.

Opening sequence

When `AvFormatDecoder::OpenFile()` is called, it begins by executing `av_open_input_file()` which prepares the *AvFormatContext* structure necessary for demultiplexing and decoding the recording's bitstream. Then, `AvFormatDecoder::ScanStreams()` is called to iterate over all the elementary streams ffmpeg was able to identify in the recording. When a video ES is found, `AvFormatDecoder::ScanStreams()` calls `AvFormatDecoder::InitVideoCodec()`, which essentially gathers information about the ES (resolution, aspect ratio,

etc.) and then calls `NuppelVideoPlayer::SetVideoParams()`. At that point, the owning `NuppelVideoPlayer` instance is able to properly configure the video output object for the recording.

4.4.6 Typical Playback Sequence

This section presents a highly simplified playback sequence starting at the moment `NuppelVideoPlayer` is told to start playing. It is assumed a recording has already been opened. Figure 4.8 models this sequence:

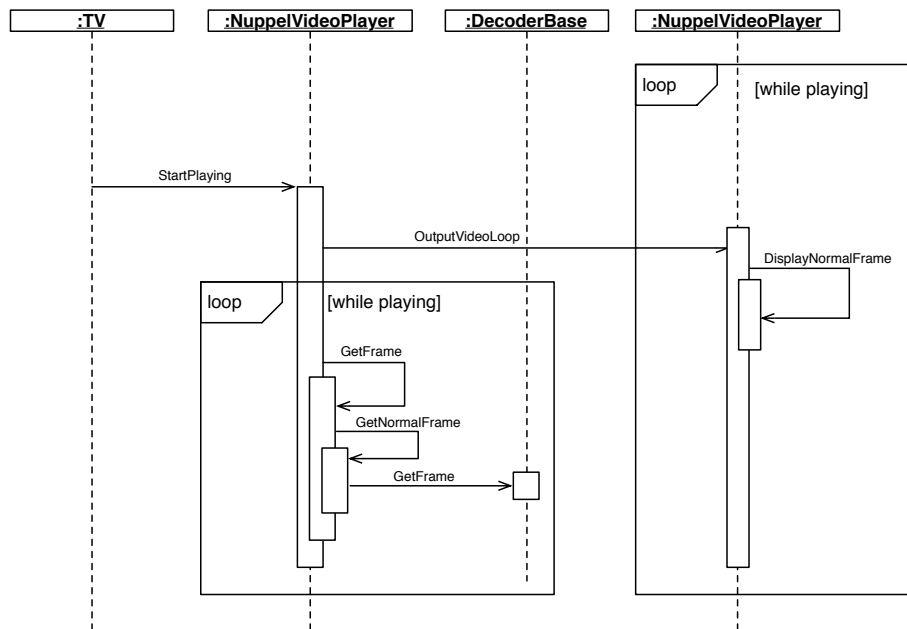


Figure 4.8: Sequence diagram of playback subsystem

As can be seen, `NuppelVideoPlayer` spawns a new thread dedicated to outputting decompressed video frames to its video output object after which it enters another loop dedicated to decompressing the recording. Decompression is handled by `NuppelVideoPlayer`'s decoder object whose `DecoderBase::GetFrame()` method gets periodically called.

4.5 The MythTV Database

MythTV relies on a MySQL² database to store most of the information it needs to operate, including nearly all its configuration parameters. In that respect,

²<http://www.mysql.com>

MythTV is somewhat different from other UNIX programs that have traditionally relied on text-based configuration files. The database schema is somewhat documented in the MythTV Doxygen documentation, however it is necessary to read the `InitializeDatabase()` function in `dbcheck.cpp` for a complete list of tables and fields.

4.5.1 *capturecard* Table

A *capturecard* record stores all the relevant information about a capture device. This table is particularly important because each record stores the type of the device, which is used later on by *TVRec* to select the appropriate recorder and channel classes. If a device has more than one input, such as a TV card with coaxial and S-Video connectors, more than one record in this table will exist for that device if the inputs can be independently tuned.

cardtype field

The *cardtype* field stores the device type as a string. Below are some valid values:

- V4L
- MJPEG
- HDTV
- DVB
- MPEG
- FIREWIRE
- GO7007
- DBOX2

videodevice field

The *videodevice* field stores the information necessary to access the device. It usually is a path inside `/dev`, the device directory of UNIX-based systems.

4.5.2 *videosource* Table

A *videosource* record stores all the relevant information about a TV signal, such as the broadcaster's name, frequencies it uses and EPG data retrieval credentials.

name field

The *name* field stores the broadcaster's name. For example, "Videotron" or "Rogers".

freqtable field

The *freqtable* field stores the name of the frequency table the represented signal uses, such as "us-bcast" or "us-cable".

4.5.3 *cardinput* Table

A *cardinput* record stores all the relevant information about a video source and capture card relationship. That is, it models the connection of a signal to a capture device input. In order for MythTV to function, it must have at least one *cardinput* record, which implies having at least one *videosource* record and one *capturecard* record.

cardid field

The *cardid* field stores the capture card ID of the relationship.

sourceid field

The *sourceid* field stores the video source ID of the relationship.

inputname field

The *inputname* field stores the name of the relationship. This is used by MythTV to present a list of available *cardinput* records to the user.

startchan field

The *startchan* field stores the default channel number. When MythTV uses the record for the first time, it will tune the associated capture card to this channel (see Section 4.5.4 for details on *channel* records).

tunechan field

The *tunechan* field stores the last channel number the associated capture card was successfully tuned to.

4.5.4 *channel* Table

A *channel* record stores all the relevant information about a channel, including a unique channel ID, its number (what a people typically use to select a channel), frequency ID, video source ID, call sign and name. It should be noted that at the time of writing, MythTV required at least 2 visible (enabled) channel records to operate without errors.

channum field

The *channum* field stores the channel's number.

sourceid field

The *sourceid* field stores the ID of the video source from which this channel is captured.

***callsign* field**

The *callsign* field stores the channel's call sign (usually a 4 capital letters code).

***name* field**

The *name* field stores the channel's full name.

***commfree* field**

The *commfree* field should store 0 if the channel has commercials or 1 if the channel does not.

***visible* field**

The *visible* field should be 0 to disable the channel or 1 to enable it.

4.5.5 Miscellaneous Tables

This section contains brief details on a number of other tables.

***program* table**

The *program* table stores all the relevant information about a TV program, such as the channel ID on which it runs, its start and end times, its title, and so on. Normally, this table gets filled with EPG data MythTV retrieves from various online services, based on information in the *videosource* table (see Section 4.5.2).

***recordedmarkup* table**

The *recordedmarkup* table stores program markup information (see Section 4.2.1) for all available recordings (both completed and in-progress ones).

5

ITU-T Rec. H.264 | ISO/IEC 144496-10 AVC

The H.264 video codec is the result of many years of work by a joint group of experts formed by ITU-T's VCEG and MPEG. VCEG started the project in 1998 as H.26L, with the long term objective of achieving double the coding efficiency of the best existing video coding standards. Previous successful collaboration between VCEG and MPEG (on the development of H.262 / MPEG-2, in particular) led the two groups in 2001 to combine their expertise. The JVT completed the final specification for H.264 / AVC in March 2003 [12, 16]. Figure 5.1 presents a timeline of the evolution of ITU-T and MPEG video coding standards.

This chapter aims at introducing the H.264 video codec and at giving sufficient technical details to support upcoming chapters. Most of the information presented in the following sections was taken directly from the H.264 recommendation [4] which is to be considered the authoritative source in all cases where discrepancies between it and this report may be found.

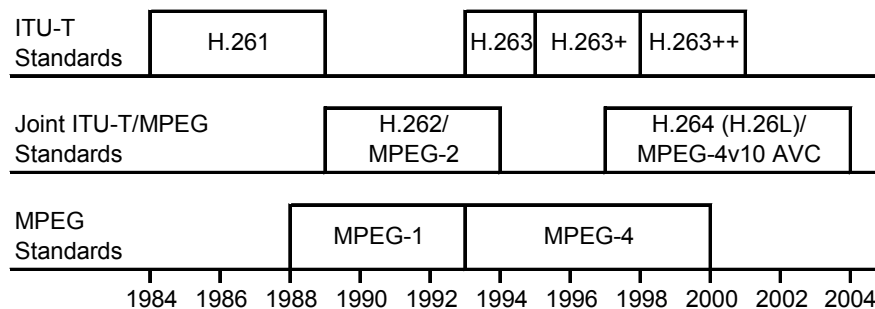


Figure 5.1: Evolution of ITU-T and MPEG video coding standards [12]

5.1 Basics of Video Compression

The ultimate goal of video compression is to reduce the size (or bit rate) of a sequence of correlated pictures (otherwise known as a video sequence) without affecting the human perception of the sequence as much as possible. To that end, a video codec will generally seek to remove information that humans cannot sense from the video sequence (the fundamental principle of perceptual coding). It should be noted that this report only considers lossy video compression (where information is lost); there exists lossless video codecs (where a perfect coded version of the signal is transmitted), but their compression ratios are far lower than those of lossy codecs.

5.1.1 Chroma Subsampling

The human eye is less sensitive to color than to intensity. Consequently, many codecs use chroma subsampling to reduce the amount of chroma information that is sent along with luma information. MPEG codecs, including H.264¹, typically use 4:2:0 chroma subsampling, which yields a quarter of the original color resolution overall (see [19] for details). This also means that picture data is converted (if necessary) from the computer-friendly RGB color space to the YCbCr color space for processing.

5.1.2 Macroblocks

All MPEG and ITU-T video codecs use the concept of macroblocks, which is essentially the division of a picture into a sequence of square pixel regions. As of MPEG-1, macroblocks have an area of 16 square pixels². Processing entire frames at once would be prohibitively costly and would not allow prediction methods such as motion compensation (see Section 5.1.4 below).

5.1.3 Exploiting Spatial Statistical Dependencies

Within any given picture, it is typical that some information is repeated in the spacial domain. For example, grass and blue skies are spatially correlated because a specific luma and chroma pattern is repeated though space. Therefore, a picture can be compressed by predicting the value of luma and chroma samples from already decoded samples elsewhere within the picture and spacial correlation data, which requires far fewer bits to encode in a bitstream than full samples. This is usually known as intra-picture prediction. Figure 5.2 shows a simplified version of H.264's intra-picture prediction process.

¹Disregarding the more recent High (Fidelity Range Extension) profiles.

²XviD has a 32x32 macroblock mode, but it is rarely used and incompatible with ISO MPEG-4 decoders.

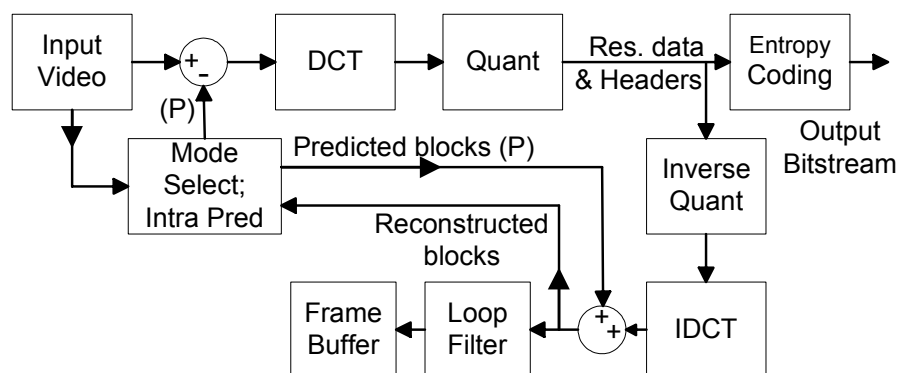


Figure 5.2: Simplified block diagram of intra-picture encoding in H.264 [12]

5.1.4 Exploiting Temporal Statistical Dependencies

Within any given sequence of pictures, it is typical that some information is repeated in the time domain. For example, the pictures forming a video sequence of a grassy plain without a lot of motion are temporally correlated because a specific luma and chroma pattern is repeated across many pictures over a period of time. Therefore, that sequence can be compressed by predicting the value of luma and chroma samples from samples already decoded elsewhere within the sequence and temporal correlation data, which requires far fewer bits to encode in a bitstream than full samples. Removing temporal redundancies statistically yields the greatest bit rate gains and as such codecs generally spend a considerable amount of time performing this type of analysis, which is usually known as inter-picture prediction and typically involves motion compensation (see [21] for details). Figure 5.3 presents a simplified version of H.264's inter-picture prediction process.

5.1.5 Further Entropy Reduction

Beyond spacial and temporal redundancy reduction, codecs will typically apply an energy compaction transform followed by quantization on the prediction residuals to further reduce entropy. Once this transform space information is stored along with structural information, codecs can further compact bitstreams by applying lossless compression methods such as run length coding (see [20]) or arithmetic coding (see [18]).

5.2 Improvements Over Previous Video Codecs

H.264 doesn't introduce any significantly new ideas to video coding. Instead, it builds on the techniques and methods developed over the years by MPEG and

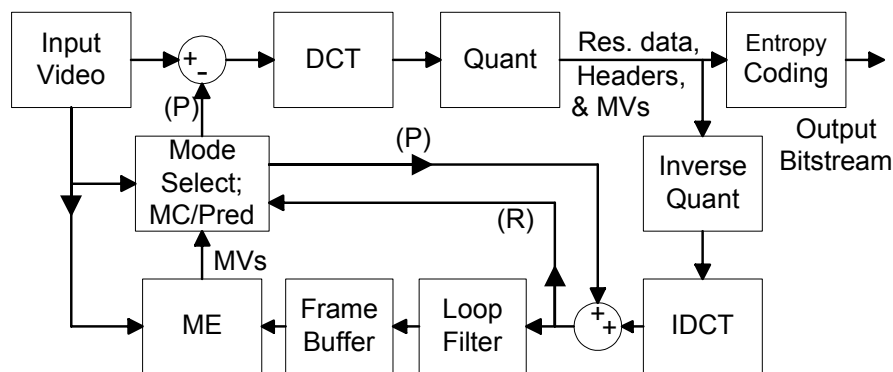


Figure 5.3: Simplified block diagram of inter-picture coding in H.264 [12]

VCEG and pushes them to the next level; it is an evolution, not a revolution. As such, H.264 is essentially a macroblock-based codec that aims at exploiting spacial and temporal redundancies in addition to entropy coding to compress video sequences. However, as stated previously, it brings several improvements, as well as a few new things, to the fundamental ideas of video compression. Some of the more important ones are briefly described below.

5.2.1 Prediction Improvements

Variable block-size motion compensation with small block sizes

Motion compensation works on sub-divisions of macroblocks called blocks. The smaller the blocks are, the more accurate motion compensation is. H.264 allows for variable (and thus adaptive) motion compensation block sizes as small as 4x4, far beyond the minimum of previous standards. This in turn allows H.264 to retain more detail in complex scenes and to be as efficient as possible.

Quarter-sample-accurate motion compensation

Quarter-sample-accurate motion compensation (commonly known as qpel) allows the motion compensation process to perform sub-pixel sampling with 4 sub-divisions accuracy. H.264 improves on this method by providing better and simpler interpolation compared to previous standards.

Motion vectors over picture boundaries

First introduced in H.263, this method allows motion vectors to reference extrapolated areas outside reference picture boundaries. This can be used by an encoder to implement better motion compensation.

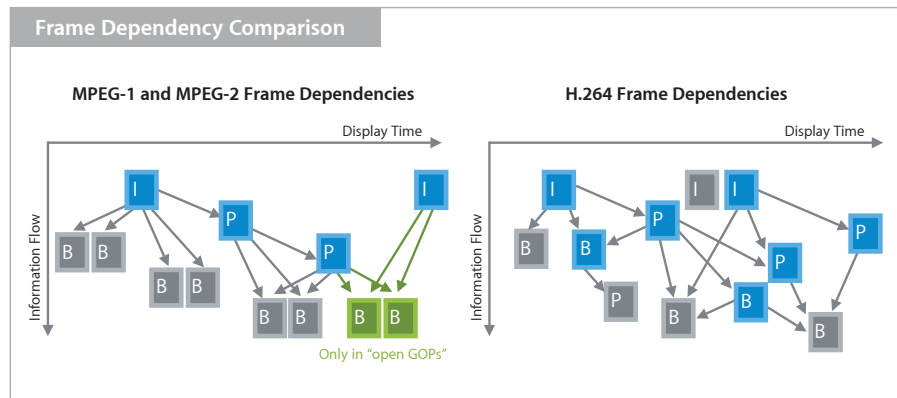


Figure 5.4: Frame dependency comparison [10]

Multiple reference picture motion compensation

H.264 dramatically increases the number of reference pictures an encoder can use for motion compensation while at the same time removing almost every restriction imposed by previous standards on the picture type and sequence position of reference pictures. H.264 allows up to 32 reference pictures per predictive or bi-predictive picture. Figure 5.4 illustrates this improvement.

Decoupling of referencing order from display order

Previous standards required strict ordering of motion compensation reference pictures with respect to display order. H.264 largely removes those restrictions, giving an encoder more flexibility to choose an optimal ordering scheme. Figure 5.4 demonstrates this new capability.

Decoupling of picture representation methods from picture referencing capability

H.264 allows any type of picture to be used as a reference picture for motion compensation, including bi-predictive pictures. Most previous standards did not allow this. Figure 5.4 represents this additional flexibility.

Weighted prediction

Weighted prediction is a new method introduced in H.264 that allows motion vectors to be offset by an encoder-specified coefficient. This is particularly advantageous when coding scenes with fades.

Directional spatial prediction for intra coding

This improvement is similar to motion vectors over picture boundaries in that it allows intra-picture prediction to reference extrapolated edge pixels from previously-decoded parts of the picture being processed, even if those regions were not coded using intra coding.

In-the-loop deblocking filtering

Block-based coding produces blocking artifacts as a result of prediction and entropy coding of residuals. H.264 applies an adaptive deblocking filter to smooth those artifacts out during the decoding process, but also integrates the filter directly inside the motion compensation loop, yielding better picture prediction. This filter loop-back during encoding can be seen on Figures 5.2 and 5.3.

5.2.2 Coding Improvements

Small block-size transform

Unlike previous standards which have typically used an 8x8 pixels entropy compaction transform, H.264 employs a 4x4 transform, mitigating blocking, smearing and ringing effects. This smaller transform size is also justified by the need to provide transform regions matching the smallest prediction regions.

Exact-match inverse transform

H.264 is the first video coding standard to have an integer inverse entropy compaction transform, resulting in bit-equal output across all compliant decoders.

Arithmetic entropy coding

Arithmetic coding (see [18]) is a more advanced and efficient entropy coding algorithm than run length coding (see [20]). H.264 offers a context-based adaptive arithmetic coder that can yield significant bit rate savings at the expense of computational complexity.

5.2.3 Error Resilience and Flexibility Improvements

Parameter set structure

In previous standards, header information (such as necessary decoding parameters) is transmitted alongside picture information. However, losing header information is far more detrimental to error covering than picture information. As such, H.264 clearly separates one from the other such that header information can be transmitted separately, perhaps on a high-reliability channel. Figure 5.5 illustrates such a transmission system.

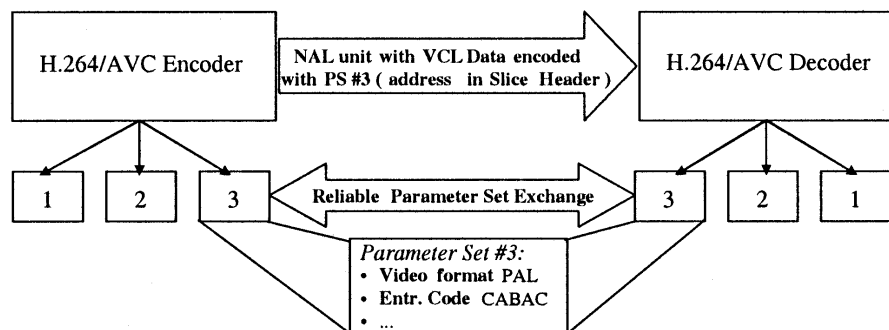


Figure 5.5: Reliable parameter set exchange [16]

NAL unit syntax structure

Each syntax structure produced by an H.264 encoder is encapsulated in a Network Abstraction Layer unit syntax structure. This allows for increased flexibility in methods used to transport H.264 video content that previous standards lacked.

5.3 Structure of a Bitstream

As stated above, an H.264 bitstream is fundamentally a sequence of NAL units encapsulating various H.264 syntax elements. The payload of each unit can either belong to H.264's video coding layer or not, and if it does, the unit is known as a VCL NAL unit. The following sub-sections offer some essential definitions as well as a more detailed description of the VCL and of the NAL, although readers are encouraged to read Clause 7.4.1.2 in [4] for more information.

5.3.1 Basic Definitions

Formally speaking, a coded video sequence is defined as a "sequence of access units that consists, in decoding order, of an IDR access unit followed by zero or more non-IDR access units including all subsequent access units up to but not including any subsequent IDR access unit" (Clause 3.30 in [4]). An access unit is defined as "a set of NAL units always containing exactly one primary coded picture. In addition to the primary coded picture, an access unit may also contain one or more redundant coded pictures or other NAL units not containing slices or slice data partitions of a coded picture" (Clause 3.1 in [4]). An IDR access unit contains an IDR primary coded picture, which is defined as "a coded picture in which all slices are I or SI slices that causes the decoding process to mark all reference pictures as "unused for reference" immediately after decoding the IDR picture. After the decoding of an IDR picture all following coded pictures in decoding order can be decoded without inter prediction from

any picture decoded prior to the IDR picture” (Clause 3.62 in [4]). Thus, an IDR access unit is the proper definition of a key frame in H.264; a point where the decoder can jump to and immediately resume decoding.

5.3.2 Video Coding Layer

The VCL contains the tools, methods and algorithms of H.264 to encode video sequences, which can be loosely equated to efficiently coding a sequence of pictures. It should be noted that the scope of the H.264 standard is limited to the the syntax and semantics of bitstreams and to the decoding process; there are otherwise no restrictions or recommendations concerning the usage of these methods, or any specific encoding process, beyond the fact that a compliant bitstream should ultimately be produced.

A coded picture can represent either an entire frame or a single field if the source material is interlaced. Each coded picture refers to one picture parameter set, which in turn refers to the sequence’s sequence parameter set. SPSs and PPSs contain necessary decoding parameters and there is only one of each active at any given time. Each coded picture is partitioned into fixed-sized macroblocks (see Section 5.1.2), which are the basic unit for which the H.264 decoding process is specified. In order words, spacial and temporal dependencies (see Sections 5.1.3 and 5.1.4) are defined at the macroblock level. To improve coding efficiency and error resilience, macroblocks are grouped into slices³ (see Figure 5.6), which are independent in the sense that a slice can be decoded without another slice’s data, assuming requirements, such as reference pictures being already decoded, are met. Consequently, slices are the basic unit for coded picture data transmission in the NAL. Finally, slices are categorized based on the prediction methods used to code their macroblocks:

- **I slice:** A slice whose macroblocks are all coded using intra-picture prediction only.
- **P slice:** A slice whose macroblocks are coded using I slice methods as well as inter-picture prediction with at most *one* motion-compensated prediction signal per prediction block.
- **B slice:** A slice whose macroblocks are coded using P slice methods as well as inter-picture prediction with at most *two* motion-compensated prediction signals per prediction block.
- **SP and SI slices:** So-called switching P and I slices, which are novel concepts introduced in H.264. See [4, 9, 14] for details.

³H.264 also includes the concept of slice groups when Flexible Macroblock Ordering is enabled.

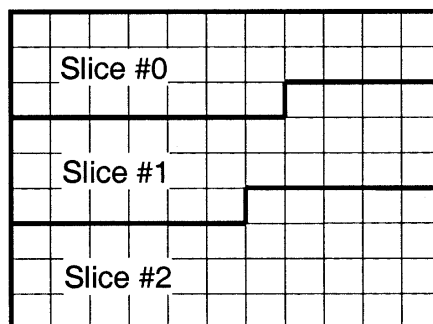


Figure 5.6: Subdivision of a picture into slices when not using FMO [16]

5.3.3 Network Abstraction Layer

The NAL contains methods and structures to encapsulate raw H.264 syntax elements for transport purposes, such as IP networks, MPEG-2 TS bitstreams or regular files. From the H.264 recommendation:

The NAL is specified to format [the VCL] data and provide header information in a manner appropriate for conveyance on a variety of communication channels or storage media. All data are contained in NAL units, each of which contains an integer number of bytes. A NAL unit specifies a generic format for use in both packet-oriented and bitstream systems. The format of NAL units for both packet-oriented transport and byte stream is identical except that each NAL unit can be preceded by a start code prefix and extra padding bytes in the byte stream format. (Clause 7.4.1 in [4])

Table 7.1 in [4] lists all valid NAL unit types and their assigned numerical IDs. Among the important ones, 1 through 4 are used to encapsulate non-IDR slice data, 5 is used to encapsulate IDR slice data, and 7 and 8 are used to encapsulate SPSs and PPSs, respectively.

MPEG-2 Transport Stream encapsulation

Changes were made to the MPEG-2 Systems standard (see [5] Amendment 3) to formally specify transport of H.264 bitstreams over MPEG-2 TS. From this amendment, it should be noted that the byte stream format (as defined by Appendix B in [4]) must be used when NAL units are encapsulated in PES packets.

Typical NAL unit sequence

Figure 5.7 illustrates a typical NAL unit sequence forming a complete access unit.

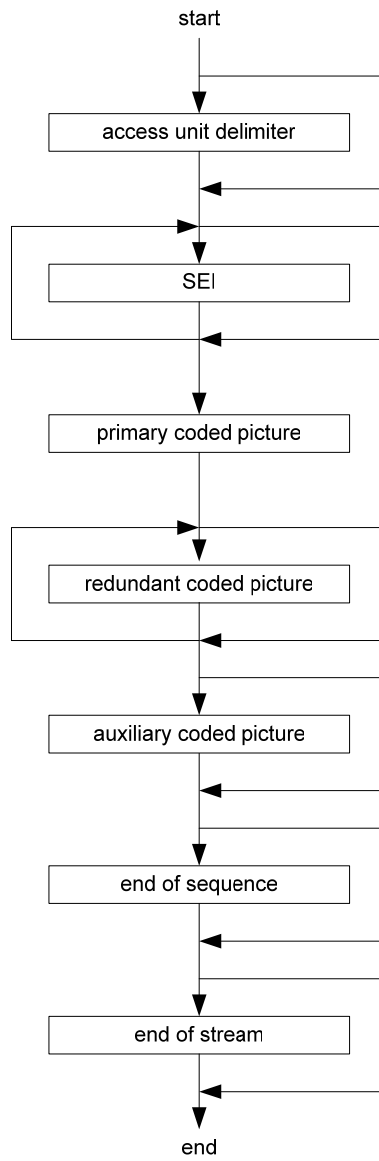


Figure 5.7: Structure of an access unit [4]

5.4 Profiles and Levels

The H.264 recommendation (see Annex A in [4] for details) specifies a number of profiles and levels. The profiles generally determine which coding tools and algorithms may be used in a bitstream, while the levels generally determine maximum resolutions and bit rates. Systems thus combine profiles and levels to flexibly impose restrictions on bitstreams they are able to process. Figure 5.8 gives an overview of the H.264 profiles specified in [4] at the time of writing. As a reminder, Clause 8.1.2 and Table 9 in the DMB specification ([3]) state the profile (baseline) and level (1,3) that must be used in DMB bitstreams.

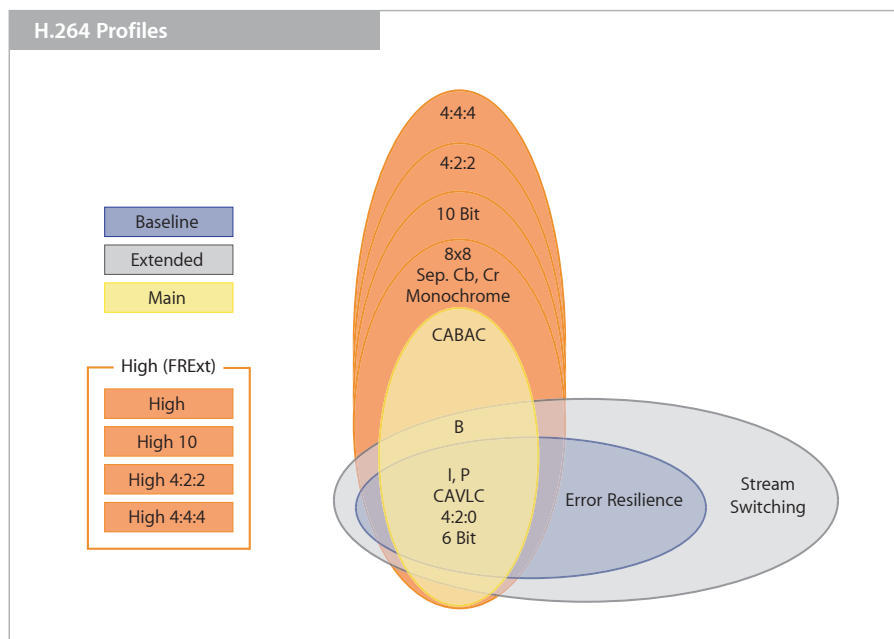


Figure 5.8: H.264 profiles [10]

6

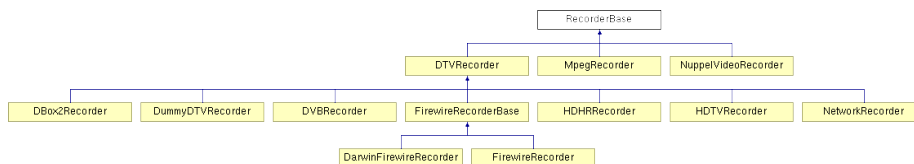
Implementation Details of MythTV Modifications

Several additions and modifications had to be made to MythTV before it was able to playback bitstreams closely matching the DMB specification (see Section 3.3.3) and originating from CRC's dataport device (see Section 3.1.1). It should be noted that known relevant limitations in `ffmpeg` (see Section 8.1) were not addressed during the research discussed in this report.

After a careful analysis of MythTV and its source code, two major problems were identified. First, there was no way to record a bitstream being multicast over an IP network using raw UDP or RTP, as is done by the dataport device (see Section 3.1.1). This was solved by writing a new MythTV recorder capable of capturing an MPEG-2 TS bitstream unicast or multicast over UDP/IP. Second, sample MPEG-2 TS bitstreams accessed directly from the MythTV host hard drive showed that MythTV was unable to properly record or playback programs using H.264 video coding. Consequently, appropriate changes were made to the aforementioned network recorder and to the MythTV playback subsystem. The following two sections examine these issues in more details.

6.1 The IP Network Recorder

A new recorder (see Section 4.3.2) was written to capture MPEG-2 TS bitstreams unicast or multicast over UDP/IP. Indeed, this delivery mode is one of the simplest available on the CRC dataport and is relatively straightforward to use compared to other modes such as RTP. The IP network recorder was sub-classed from *DTVRecorder*, which provides some useful methods for dealing with MPEG-2 TS bitstreams and video ESs using MPEG-1 or MPEG-2 video coding (the dominant codec in digital television broadcasting). Figure 6.1 shows the class hierarchy of *RecorderBase* at the time of writing with the addition of the IP network recorder. It should be noted that the class was renamed

Figure 6.1: Updated Doxygen class hierarchy for *RecorderBase*

CRCIPNetworkRecorder prior to its public release¹.

6.1.1 Architecture

The IP network recorder relies on ffmpeg’s URL access API to configure and open network sockets using the `url_open()` function, with the *videodevice* field of the associated *capturecard* record (see Section 4.5.1) as value to the *filename* argument. Therefore, *videodevice* fields should contain valid ffmpeg URLs, such as “`udp://?localport=1234`” to open a unicast UDP/IP socket on port 1234.

Once that is done, the IP network recorder queries ffmpeg for a socket’s file descriptor in order to use the `select()` function in the recording loop. Because *AvFormatDecoder* (see Section 4.4.5) has native support for MPEG-2 TS bitstreams, the IP network recorder simply writes the payload of received UDP packets to its *RingBuffer* instance.

Recording markup and codec support

Recorders are expected to provide recording marks (see Sections 4.2.1 and 4.3.2) as best they can, particularly ones that indicate the location of key frames within bitstreams they record. To do this processing, the IP network recorder relies on methods of its parent class for MPEG-1 and MPEG-2 video ESs, and on the H.264 utilities described in Section 6.2.1 for H.264 video ESs. In addition, the H.264 utilities allow the IP network recorder to wait until the beginning of an access unit (see Section 5.3.1) before writing packets to its *RingBuffer* instance, preventing incorrect decoding of H.264 ESs.

TVRec modifications

TVRec was modified to allocate *DummyChannel* instances and to recognize “`CRC_IP`” as the *cardtype* field value (see Section 4.5.1) for IP network recorders.

6.1.2 Limitations

The IP network recorder, at the time of this writing, has the following limitations:

¹See <http://svn.mythtv.org/trac/ticket/1787> for the patch.

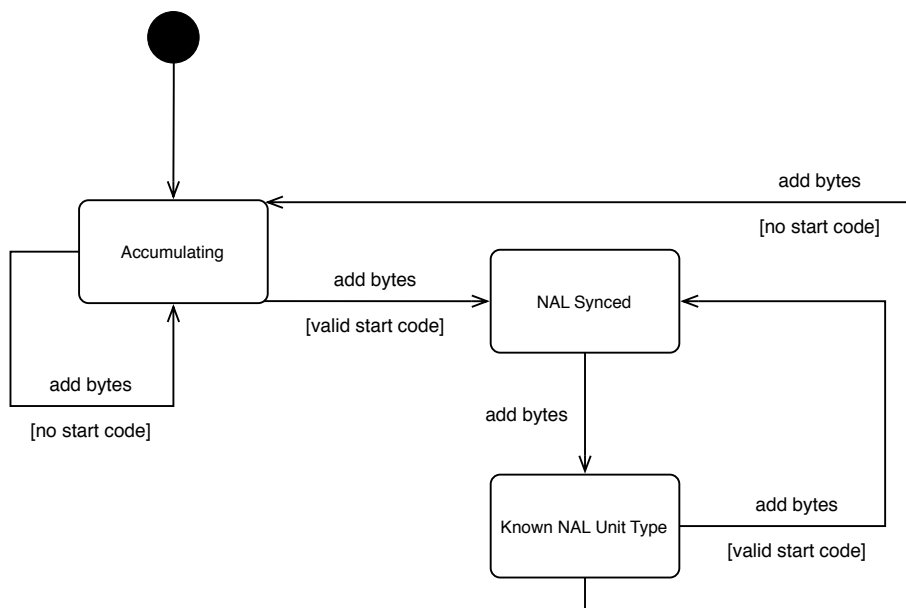
- **No RTP support:** RTP has many advantages over unicast or multicast UDP/IP, particularly for Internet streaming. Although it is possible ffmpeg would properly handle an RTP protocol URL, it is doubtful the IP network recorder would be able to process such streams without modifications.
- **Nothing but MPEG-2 TS:** Like most other recorders in MythTV, the IP network recorder assumes it will be processing MPEG-2 TS packets. Any other form of encapsulation is not supported.
- **Only one program per bitstream:** The IP network recorder analyses the MPEG-2 TS bitstream it is receiving (when recording is initiated) to determine how many programs have been multiplexed in it. If more than one is found, the bitstream is not processed and the recorder becomes errored.
- **Programs must have a video ES:** If no video ES is associated to a program, the containing bitstream is not processed and the recorder becomes errored.
- **Programs must use MPEG-1, MPEG-2 or H.264 video coding:** Video ESs must use MPEG-1, MPEG-2 or H.264 video coding to be recognized as such. It should be noted that properly supporting additional codecs would require more than merely changing this filter.
- **No MPEG-4 SL support:** As a clarification to the previous limitation, MPEG-4 SL streams are not processed at all.

6.2 H.264 Support

Despite the fact ffmpeg includes a capable H.264 decoder, MythTV was initially unable to record or playback MPEG-2 TS bitstreams with H.264 video ESs. In addition, further playback tests showed that there were no such issues when other container formats, such as QuickTime's .mov or MPEG-4's .mp4, were utilized. Eventually, it was understood that recording markup caused MythTV to abort recording and playback processes because H.264 ESs were being analysed as if they were MPEG-1 or MPEG-2 ESs. Therefore, proper H.264 support was added to MythTV by means of new H.264 utilities and modifications to the recording subsystem (specifically to the IP network recorder, see Section 6.1.1) and to the playback subsystem (specifically to *AvFormatDecoder*, see Section 6.2.2).

6.2.1 H.264 Utilities

The H.264 utilities consist of 2 classes, *NALUnitType* and *KeyframeSequencer*, which themselves are part of a new name space called *H.264*. The first class is a simple container for a C enumeration defining symbols for H.264 NAL unit

Figure 6.2: Simplified state diagram of *KeyframeSequencer*

types (see Section 5.3.3). The second is an extremely simple pseudo-state machine designed to parse raw H.264 bitstreams. In particular, it was written for recorders and *AvFormatDecoder* to enable them to detect key frames within H.264 bitstreams and track the container offset at which the last IDR access unit began (see Section 5.3.1). Figure 6.2 is a simplified state diagram of *KeyframeSequencer*. In addition to the states represented on it, *KeyframeSequencer* also maintains a handful of others such as whether or not the NAL unit is a slice, an IDR slice or not a VCL NAL unit at all while in the “Known NAL Unit Type” state.

6.2.2 *AvFormatDecoder* Modifications

Once the H.264 utilities were written, *AvFormatDecoder* was modified to take advantage of them. Indeed, although a few other minor issues were found and addressed, the biggest problem that had to be resolved in *AvFormatDecoder* lied in its `GetFrame()` method (see Section 4.4.6), which essentially extracts raw codec packets from the container of the recording being played and decompresses them into the appropriate buffers.

Summary of the `GetFrame()` problem

In `GetFrame()`, when a new packet is successfully obtained from `ffmpeg`, the following source code² is executed if the packet belongs to the video stream *AvFormatDecoder* selected for playback.

```

AVCodecContext *context = curstream->codec;
if (context->codec_id == CODEC_ID_MPEG1VIDEO ||
    context->codec_id == CODEC_ID_MPEG2VIDEO ||
    context->codec_id == CODEC_ID_MPEG2VIDEO_XVMC ||
    context->codec_id == CODEC_ID_MPEG2VIDEO_XVMC_VLD)
{
    MpegPreProcessPkt(curstream, pkt);
}
else
{
    if (pkt->flags & PKT_FLAG_KEY)
    {
        HandleGopStart(pkt);
        seen_gop = true;
    }
    else
    {
        seq_count++;
        if (!seen_gop && seq_count > 1)
        {
            HandleGopStart(pkt);
        }
    }
}
if (len == 4 && dvdvideopause)
    dvdvideopause = false;
if (framesRead == 0 && !justAfterChange &&
    !(pkt->flags & PKT_FLAG_KEY))
{
    av_free_packet(pkt);
    continue;
}

```

Essentially, this body of code calls `HandleGopStart()` if the packet contains a key frame. Because `ffmpeg` cannot always flag packets with `PKT_FLAG_KEY`, as is the case when the container is MPEG-2 TS, MythTV relies on the specialized `MpegPreProcessPkt()` method when a packet uses MPEG-1 or MPEG-2 video coding.

At the end of the segment, *AvFormatDecoder* simply drops packets until it sees a “key frame packet” when it hasn’t decoded any frames yet (which occurs

²`avformatdecoder.cpp:2655 r9919`

when playback is initiated), since it is likely that decoders would fail should they be given a packet with potential dependencies to frames that in this case will never be read. Therefore, this code drops every single H.264 packet encapsulated in MPEG-2 TS because none of them are ever going to be flagged with `PKT_FLAG_KEY`.

Solution to the `GetFrame()` problem

An instance of *KeyframeSequencer* from the H.264 utilities and a new method called `H264PreProcessPkt()`, with semantics similar to those of `MpegPreProcessPkt()`, were added to *AvFormatDecoder*. Also, an additional branch was added to the above body of code to call `H264PreProcessPkt()` when `codec_id` is equal to `CODEC_ID_H264`. `H264PreProcessPkt()` utilizes the sequencer to properly flag ffmpeg packets with `PKT_FLAG_KEY`, thus allowing *AvFormatDecoder* to smoothly decode recordings using H.264 video coding.

6.2.3 Miscellaneous Modifications

A series of new H.264 codec constants were added to MythTV, such as `kCodec_H264`, with a few appropriate modifications meant to formally recognize these new constants.

6.2.4 Limitations

At the time of this writing, the H.264 support has two major limitations:

- **Incomplete access unit delimitation algorithm:** *KeyframeSequencer* does not implement a specification-compliant algorithm for delimiting access units, which is part of the key frame detection process. More specifically, the sequencer is not able to analyse VCL NAL units according to Clause 7.4.1.2.4 in [4] such that it complies with the last condition specifying the start of a new access unit of Clause 7.4.1.2.3 in [4].
- **No hardware-assisted H.264 decoding:** MythTV supports hardware-accelerated MPEG-1 and MPEG-2 decoding on some hardware through XvMC (see [26]). However, there is no equivalent for H.264, which limits the complexity and resolution of H.264 bitstreams that can be decoded at acceptable speeds. For example, it is doubtful anything but the most powerful personal computers will be able to handle 1080@60p H.264 video.

7

Conclusions

The primary objective of the research was to bring rich media distributed with Digital Multimedia Broadcasting and PVR functionality together by adapting MythTV, an open-source PVR program. In order to do so, a number of technical challenges had to be overcome, such as bridging the gap between DMB broadcasts and MythTV, as well as adding H.264 video coding support to MythTV. As reported in Chapter 6, both of these objectives were successfully achieved, and tests in the lab confirmed the validity of the solutions that were implemented.

Previous research had concluded that a system using a V4L2 virtual loop-back driver to transfer DMB bitstreams between a feeding application and MythTV by implementing full support for V4L2 compression throughout the chain was the best course of action for CRC, principally because a V4L2 virtual loop-back driver could be used with any number of applications beyond MythTV. However, the complexity of such a design, not to mention a number of potential future issues such as DMB tuning in MythTV, was in the opinion of this research greatly underestimated. Furthermore, V4L2 compression support was practically non-existent at the time this approach was analysed, severely weakening the interoperability advantage V4L2 theoretically offered. The fact that the system presented in this report is already functional and has a clearly defined roadmap that relies on existing extensible foundations is an additional reason for continuing development of the PDRR project with the architecture established by this report.

Finally, the MythTV project has officially endorsed the H.264 utilities by committing the source code to the main MythTV subversion repository¹. This is a testament to the usefulness and re-usability of these software classes. Indeed, many broadcasters around the world are beginning to use H.264 video coding, as both ATSC and DVB standards have selected it as their next-generation video codec. Therefore, it is only natural that MythTV would want to support H.264 video coding as much as possible. Given that, CRC has truly made a lasting contribution to this high-visibility open-source project.

¹<http://svn.mythtv.org/trac/changeset/9919>

Recommendations

Although significant progress has been made by the research presented herein, there remains a number of issues that will need to be addressed before the PDRR project is completed. Some of the more pressing ones are detailed below.

8.1 Address Relevant ffmpeg Limitations

ffmpeg is at the heart of *AvFormatDecoder*. Unfortunately, at the time of this writing, it suffers from a number of limitations that prevent the proposed PDRR implementation from complying with the DMB specification.

8.1.1 Implement MPEG-4 SL Demultiplexing

At the time of this writing, ffmpeg is not able to decode MPEG-4 SL elementary streams encapsulated in MPEG-2 TS bitstreams. Consequently, this report recommends that the ffmpeg MPEG-2 TS demultiplexer be augmented with MPEG-4 SL demultiplexing. It should be noted that the DMB specification imposes very narrow restrictions on the number and types of scene objects, so a partial MPEG-4 SL implementation may be possible.

8.1.2 Improve AAC Decoding

At the time of this writing, the ffmpeg AAC decoder has a number of shortcomings which impact its usability. Although this does not prevent development of the PDRR project directly, it may ultimately limit deployment or demonstrations. Therefore, this report suggests that these issues be monitored, and that efforts be spent to address them only if they become too restrictive.

8.2 Implement Dataport Control In MythTV

The implementation detailed in this report lacks the ability to control the CRC Dataport directly from within MythTV. Ideally, it should be able to instruct the Dataport to alter the destinations of available services and to tune to other frequencies simply by changing the active channel in MythTV. To that end, this report recommends that a subclass of *ChannelBase* (see Section 4.3.3) be designed to communicate with the Dataport's telnet interface by means of a network socket. It is also likely that a new recorder, tightly coupled to this new *ChannelBase* subclass, will need to be written in order to be able to specify the Dataport address in the *videodevice* field (see Section 4.5.1).

8.3 Implement Correct H.264 AU Delimitation

As was stated in Section 6.2.4, a specification-compliant access unit delimitation algorithm was not implemented in *KeyframeSequencer*. This report recommends that the `KeyframePredicate()` method be modified to comply to the clauses identified in Section 6.2.4 should the simplified algorithm current in place cause problems. In addition, this report suggests VLC's H.264 packetizer as a starting point for the improved access unit delimitation method.

8.4 Implement DMB EPG Data Loading

MythTV stores a large amount of EPG data in its database (see Section 4.5.5). In order to keep this data up-to-date, the *mythfilldatabase* program is normally ran periodically to retrieve new EPG data from online services, based on the *videosource* records (see Section 4.5.2). However, it is entirely possible to add EPG data to the database in other ways, for example by manually adding records in the appropriate tables.

Therefore, this report recommends that a new program be written to access DMB EPG data and load it inside the MythTV database. The EPG data itself could be retrieved from the dataport device (see Section 3.1.1) or a more traditional online service.

Appendix A

Amendment 1

On May 22, the MythTV project committed *CRCIpNetworkRecorder* to their main Subversion repository¹, after some modifications were made by Jean-Michel Bouffard. Principally, the H.264 key frame code was moved to *DTVRecorder* so that other recorders based on it may produce correct marks for H.264 bit-streams.

¹<http://svn.mythtv.org/trac/changeset/9966>

Bibliography

- [1] *ETSI EN 300 401 v1.3.3: Radio Broadcasting Systems; Digital Audio Broadcasting (DAB) to mobile, portable and fixed receivers*, May 2001.
- [2] *ETSI TS 102 427 v1.1.1: Digital Audio Broadcasting (DAB); Data Broadcasting - MPEG-2 TS streaming*, Jul 2005.
- [3] *ETSI TS 102 428 v1.1.1: Digital Audio Broadcasting (DAB); DMB video service; User Application Specification*, Jun 2005.
- [4] *ITU-T Recommendation H.264: "Advanced video coding for generic audio-visual services"*, Mar 2005.
- [5] *ISO/IEC 13818-1: 2000 "Information technology - Generic coding of moving pictures and associated audio information: Systems"*, 2000.
- [6] *ISO/IEC 14496-3: 2001 "Information technology - Coding of audio-visual objects - Part 3: Audio"*, 2001.
- [7] *ISO/IEC 14496-3 (2001/AMD2:2004): "Parametric coding for high-quality audio"*, 2004.
- [8] *ISO/IEC 14496-1: 2004 "Information technology - Coding of audio-visual objects - Part 1: Systems"*, 2004.
- [9] *ISO/IEC 14496-2: 2001 "Information technology - Coding of audio-visual objects - Part 2: Visual"*, 2001.
- [10] Apple Computer, Inc. H.264 technology brief, Apr 2005. URL http://images.apple.com/quicktime/pdf/H264_Technology_Brief.pdf.
- [11] Luc Gallant. Implementation of a Personal Video Recorder using as source Digital Media Broadcasting. Technical report, University of Waterloo, 2006.
- [12] Mahdi Ghandi. The H.264/AVC Video Coding Standard for the Next Generation Multimedia Communication. *IAEEE Journal*. URL <http://privatwww.essex.ac.uk/~mahdi/papers/IAEE%20invited%20paper%20manuscript.pdf>.

-
- [13] Wolfgang Hoeg and Thomas Lauterbach. *Digital Audio Broadcasting: Principles and Applications*. John Wiley & Sons, LTD, 605 Third Avenue, New York, NY 10158-0012, USA, 2001. ISBN 0471858943.
- [14] Jörn Ostermann, Jan Bormans, Peter List, Detlev Marpe, Matthias Narroschke, Fernando Pereira, Thomas Stockhammer, and Thomas Wedi. Video coding with H.264/AVC: Tools, Performance, and Complexity. *IEEE Circuits And Systems Magazine*, 2004. URL http://www.img.lx.it.pt/~fp/artigos/H264_final.pdf.
- [15] Fernando C. Pereira and Touradj Ebrahimi. *The MPEG-4 Book*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002. ISBN 0130616214.
- [16] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra. Overview of the H.264/AVC Video Coding Standard. *IEEE Transactions On Circuits And Systems For Video Technology*, 13(7), Jul 2003. URL <http://www.cs.ubc.ca/~krasic/cpsc538a/papers/h264avc-overview.pdf>.
- [17] Wikipedia. MPEG-4 Part 3 — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=MPEG-4_Part_3&oldid=49710448. [Online; accessed 26-April-2006].
- [18] Wikipedia. Arithmetic coding — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Arithmetic_coding&oldid=48882135. [Online; accessed 4-May-2006].
- [19] Wikipedia. Chroma subsampling — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Chroma_subsampling&oldid=49917499. [Online; accessed 4-May-2006].
- [20] Wikipedia. Huffman coding — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Huffman_coding&oldid=49770754. [Online; accessed 4-May-2006].
- [21] Wikipedia. Motion compensation — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Motion_compensation&oldid=47542292. [Online; accessed 4-May-2006].
- [22] Wikipedia. OSI model — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=OSI_model&oldid=50130237. [Online; accessed 25-April-2006].
- [23] Wikipedia. Parametric Stereo — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Parametric_Stereo&oldid=47689044. [Online; accessed 26-April-2006].
- [24] Wikipedia. Reed-Solomon error correction — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Reed-Solomon_error_correction&oldid=49044586. [Online; accessed 19-April-2006].

-
- [25] Wikipedia. Spectral band replication — Wikipedia, The Free Encyclopedia, 2006. URL http://en.wikipedia.org/w/index.php?title=Spectral_band_replication&oldid=47894575. [Online; accessed 26-April-2006].
- [26] Wikipedia. XvMC — Wikipedia, The Free Encyclopedia, 2006. URL <http://en.wikipedia.org/w/index.php?title=XvMC&oldid=49467794>. [Online; accessed 12-May-2006].