University of Salford, Department of Acoustics, Electronic and Electrical Engineering

Investigation into Digital Video Streams

Abstract

The world of broadcasting has recently undergone a revolution, the digital transmission of television and radio services has been pioneered in the UK. This has brought about many advantages for the viewer, including many more channels, high quality widescreen pictures, high fidelity stereo sound, audio description, digital text services, and electronic programme guides.

This project has researched the methods used by broadcasters to encode and transmit digital television services. In the UK all digital television services on any medium are encoded to the MPEG-2 international standard. From discussions with broadcast engineers in the industry it was apparent that there is a lack of software tools capable of analysing these MPEG-2 transport streams. In particular, no software is currently available to decode the widescreen signalling codes used by the BBC.

To solve these problems a Windows application was written using Visual C++. The application is capable of finding and decoding transport stream headers, extracting and displaying video picture properties, de-multiplexing a transport stream into elementary streams and analysing widescreen format descriptors.

Acknowledgements

The project was suggested and supported by BBC Technology, Consulting and Projects, Technology Group. In particular I would like to acknowledge to help of:

Andrew Smith, BBC Technology, for providing MPEG-2 specifications, transport streams, computer hardware, advice and testing the software as it has been developed.

Andy Woodhouse, BBC Training and Development, for making his MPEG-2 lecture notes available to me.

Ben Shirley, University of Salford, for supervising the project and providing programming assistance.

Contents

Chapter 1 **Introduction**

This chapter will present an introduction to digital broadcasting and explain the need for a MPEG-2 transport stream analyser. A project time plan is also included.

1.1 Introduction

The use of digital video is becoming widespread, especially in the field of broadcasting. In the last few years digital broadcasting has become a reality and it is now possible to receive digital television in every house in the UK by either satellite, cable or terrestrial transmission.

This has brought many advantages to the viewer, including; more choice, high quality widescreen pictures, high quality stereo sound, audio description, digital text services, and electronic programme guides.

These improvements are only made possible because of the nature of digital video systems. The Moving Pictures Expert Group (MPEG) was setup in 1988 to establish digital video standards. There are many possible applications for digital video and audio streams, including:

- Transmission by satellite, terrestrial, and cable broadcasting
- Video conferencing
- Interactive storage media
- DVD and Video CD
- Networked video servers
- Digital tape recorders
- Electronic news gathering
- Remote video surveillance
- Internet, intranet and mobile communications

Due to the very wide range of applications for digital video, four MPEG standards were devised.

MPEG-1 : Designed for playback of feature films from a standard speed CD-ROM drive. The data rate cannot exceed 1.5 Mbits per second and consequently, the picture quality is poor.

MPEG-2 : A superset of the MPEG-1 standard, this specification can handle broadcast quality picture resolutions. In addition high definition television is possible, although this has not been implemented in the UK at present. Provision is also made for multi channel sound.

MPEG-3 : Original intended to specify high definition television techniques, the work was absorbed into MPEG-2.

MPEG-4 : Intended for very low bit rate applications, including web streaming, mobile and video phones, and multimedia. Despite the low bit rate, high quality pictures are possible.

MPEG-7 : Work is underway to add content management tools to enable tracking and copy protection for media throughout the complete programme path.

The MPEG standard specifies the exact format of a particular data stream, the method of producing this stream from an analogue source is entirely at the discretion the encoder designer. By adopting this approach it is hoped that improvements in encoder technology will be made whilst still producing streams which can be decoded by older decoders.

In order for the broadcaster to be able to fault find and maintain these digital video systems it is essential to have a tool for analysing the contents of a stream. Several software tools are commercially available which perform analysis. However there is no software currently available which is able to find and decode the active format descriptor (AFD) control codes used by the BBC in their digital television broadcasts. One of the aims of this project is to address this problem, and provide a software tool which can be used to analyse a MPEG-2 transport stream.

1.2 Conventions used in this report

All source code is written in the C++ programming language.
Hexadecimal (base 16) numbers are preceded by 0x, all other numbers are decimal (base 10).

1.3 Project objectives

Several objectives were set at the start of the project, with the eventual aim of displaying a still video frame and its associated properties. The original aims were defined as:

- Research into digital video streams
- Develop a DOS command line utility to decode and display transport stream headers
- De-multiplex a transport stream into packetised streams and elementary streams
- Develop the software to decode and display video picture and AFD properties
- Develop a Windows interface
- Extract and display video frames

The transport streams to be analysed are supplied as computer files on a CD-ROM, capturing a live stream is not within the scope of this project.

1.4 Project time plan

A time plan covering the project duration was drawn up at the start of the project. As the project progressed it was clear that the time plan would need to be modified. Some areas of development caused problems whilst others were simpler than expected. The final plan is shown in table 1.1.

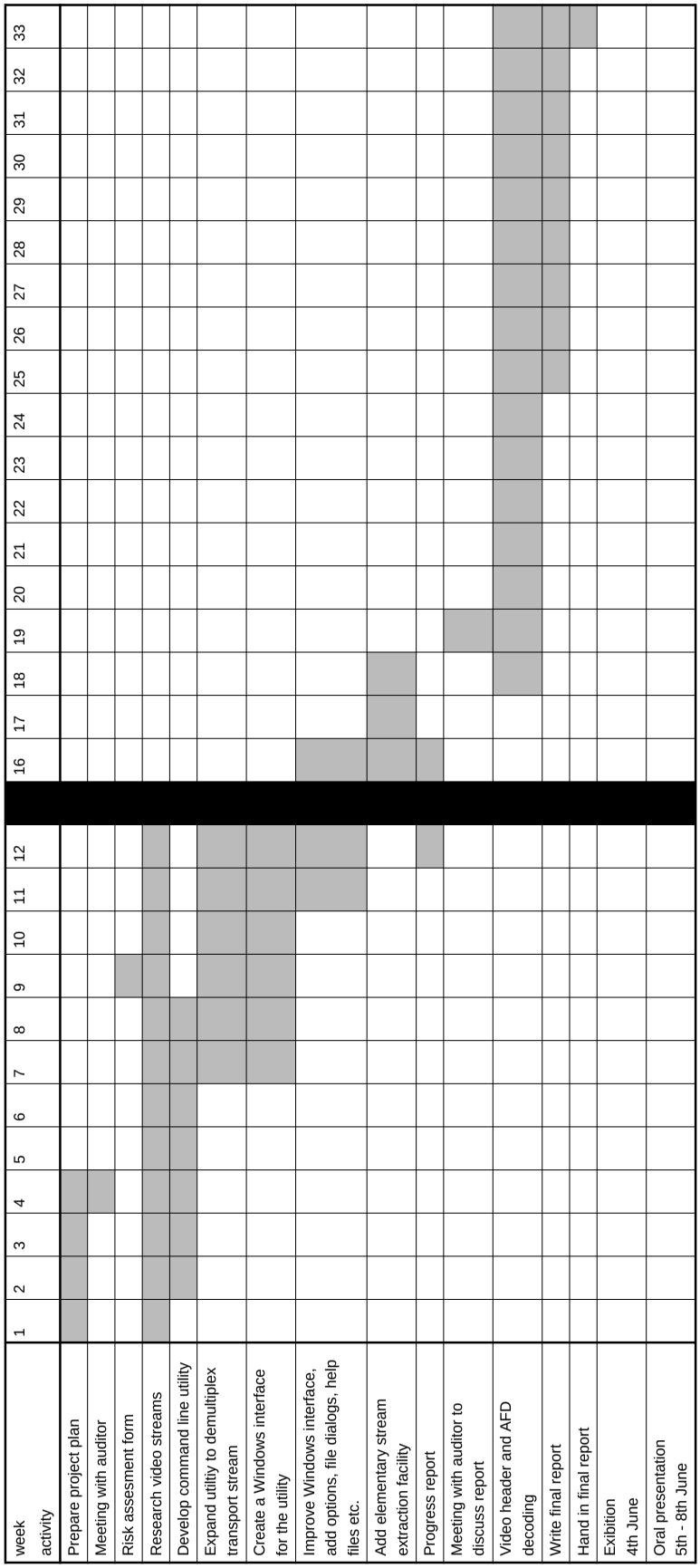

Table 1.1, Project time plan

Chapter 2 **MPEG Theory**

The MPEG data structures and compression methods are very complex and powerful. In this chapter the structure of a transport stream and video sequence will be examined. The theory of MPEG video compression will also be discussed.

2.1 Construction of a transport stream

The MPEG-2 transport stream system is designed to encode all of the elementary streams that make up a television service into a single transport stream (TS) suitable for transmission. The TS is a continuous string of transport packets, and can be decoded from any point. This feature is vital for broadcasting and other random access applications. A TS is created by time division multiplexing packets of elementary streams. An elementary stream is continuous digital audio, video or data.

Diagram 2.1 shows a simplified block diagram of how a TS is constructed. Only one service is shown, although normally many services are included in a single TS.

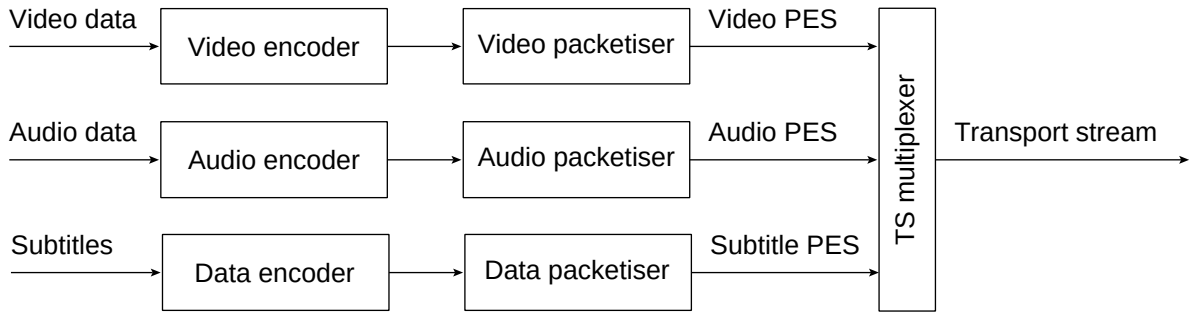

Diagram 2.1, Construction of a transport stream

2.1.1 Elementary streams

The first stage in creating a TS is to compress a digital data source to the MPEG standard, producing an elementary stream. The data source may be uncompressed digital video, audio, subtitles, teletext or other data. The compression methods used will depend on the type of source data. This project is only concerned with video streams, and these are described in the section 2.2.

2.1.2 Packetised elementary streams

Before elementary streams can be combined together to form a TS they must first be split into chunks and a data envelope added, this process is know as packetisation. The size of the data chunks will depend on the data, a single video frame for example. The process is shown in diagram 2.2.

A packetised elementary stream (PES) contains all the data of an elementary stream but is packetised to give error protection and enable random access. The PES packet begins with a header, which starts with a prefix code of 0x000001 followed by the stream ID, which identifies the elementary stream. Many other data fields may follow, including; scrambling control flags, PES priority flags, copyright flags, time stamp flags. A complete PES header structure can be seen in table A.2.

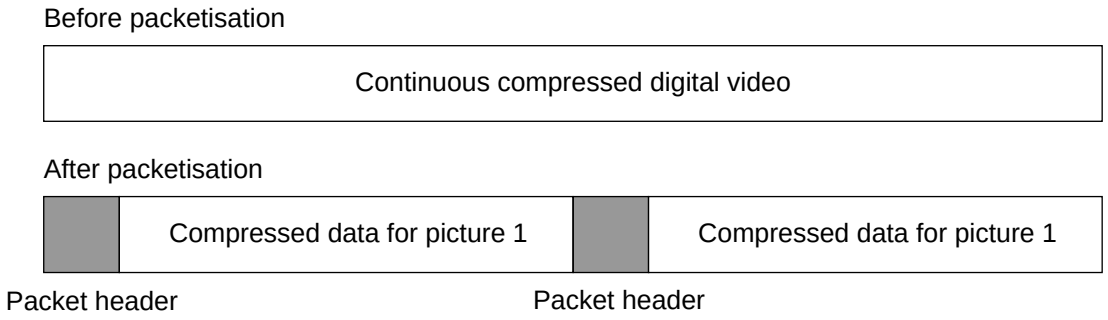

Diagram 2.2, Constructing a PES

2.1.3 Transport packets

A transport stream (TS) is a continuous string of TS packets, each exactly 188 bytes long, which includes a 4 byte long header. Every TS packet will contain part or all of a PES packet. No more than one PES packet maybe contained within the TS packet.

The first byte of the TS packet header is a synchronisation byte of value 0x47. The remaining 3 bytes of the header are used to identify the payload of the packet and provide additional decoding information. The payload is identified by means of a packet identifier (PID). Every PES is assigned a unique 13 bit long PID code, this enables the decoder to recreate all of the PES streams from the TS. A complete transport stream header structure can be seen in table A.1. A typical transport stream is shown in diagram 2.3.

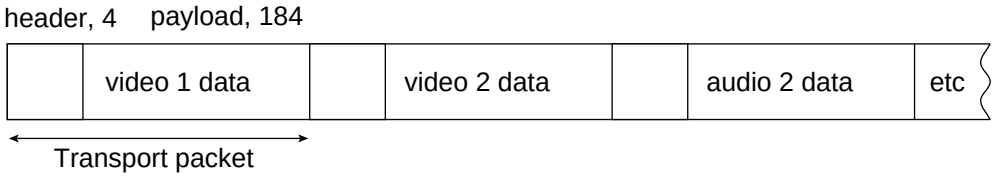

Diagram 2.3, Transport packets

The relatively short length of a transport packet is designed to give the system a high immunity from errors. If a packet is lost or corrupted during transmission then the decoder should be able to lock onto the next packet without losing a large chunk of data. Additional decoding headers are present after the TS header if the adaption field flag is set to 2 or 3.

2.2 MPEG-2 video sequence

Broadcast video is made up of 25 still pictures or frames per second. When pictures are presented to the viewer at this rate the illusion of motion is achieved, however the data rate required is 270Mbits/second. Broadcasting at this data rate is not feasible, therefore some compromises must be made. The MPEG system uses several methods to reduce the data rate considerably, which makes digital broadcasting possible.

The video compression system used is very complex and powerful, a detailed explanation is beyond the scope of this project. Within the video PES many data chunks and headers exist. Diagram 2.4 shows the sections of a complete video sequence and how they repeat.

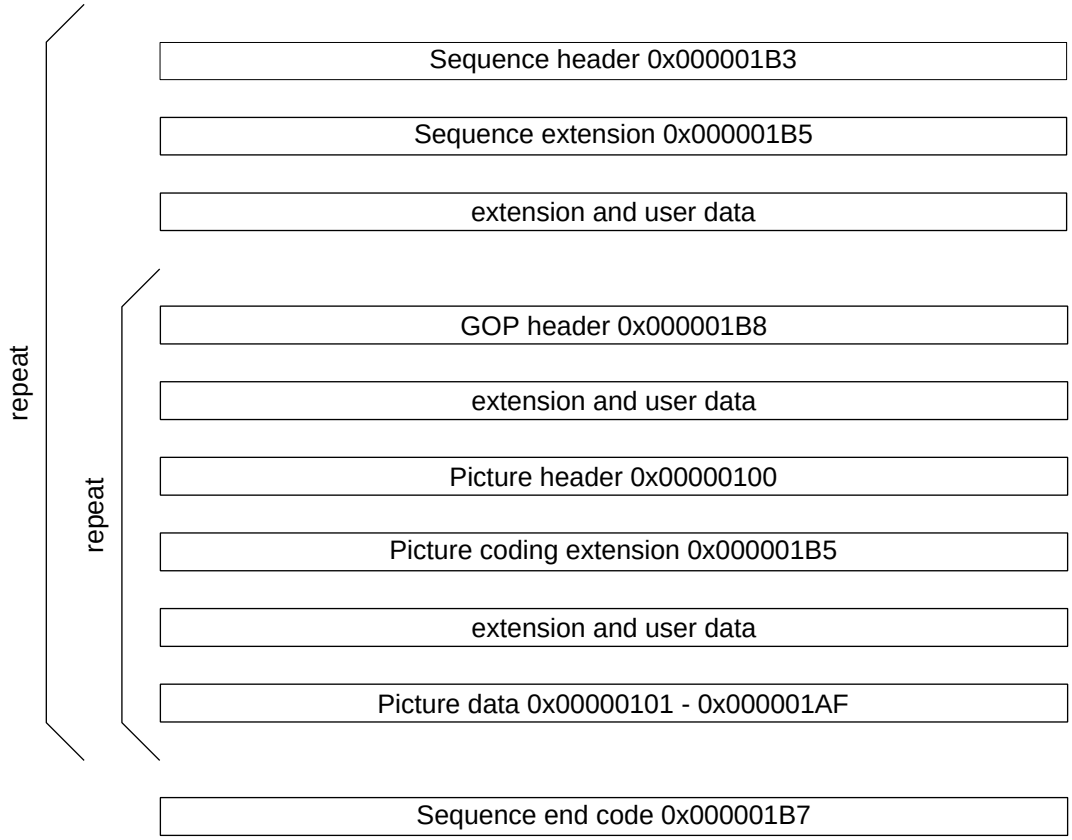

Diagram 2.4, Video sequence

2.2.1 Sequence header (0x000001B3)

The sequence header starts the video sequence, within this header picture properties such as horizontal and vertical size are specified. A complete sequence header is shown in table A.3. The extension header and extension data which follow contain additional picture properties.

2.2.2 Group of pictures header (0x000001B8)

The group of pictures (GOP) header is at the start of a 12 frame loop, the header contains the GOP type and timecode data.

2.2.3 Picture header (0x00000100)

The picture header identifies the coding method used for the current picture. Three types of picture coding are used. The sequence of picture coding methods is arranged to satisfy the following criteria:

- decoding can start without excessive delay
- the system can recover from errors quickly
- the overall data rate is suitable for the transmission medium

Depending on the application the coding sequence may vary, a typical GOP as used for transmission is shown in diagram 2.5.

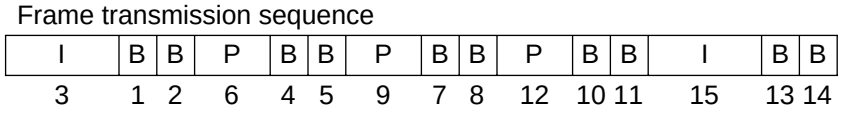

Diagram 2.5, A typical GOP sequence

2.2.4 Picture data

The compressed data for a single frame is contained within the picture data section, this will be coded in one of the three ways described in sections 2.2.5 - 2.2.7.

2.2.5 Intra coded (I) frames

An I coded frame contains all the data required to recreate a complete video frame. This provides a random access point into the data stream and flushes out any errors which may otherwise be propagated through the stream. A JPEG style compression which removes spatial redundancy is used, this exploits the fact that considerable amounts of picture data can be discarded without the human eye noticing. There are six main stages, which are described in sections 2.2.5.1 - 2.2.5.6.

2.2.5.1 Splitting into picture blocks

The frame, in component YUV format is divided into elementary blocks of 8 x 8 pixels. For a standard video frame there are 6480 luminance (Y) blocks, and 3240 for each of the two colour difference (U, V) components.

2.2.5.2 Discrete cosine transfer (DCT)

The DCT is applied to each of the elementary blocks which generates an 8 x 8 matrix of coefficients. The top left value is the DC coefficient which represents the average luminance or chrominance for the block. The value of the other coefficients usually decreases quickly, so values near the bottom right are very small or zero. If the block has uniform luminance or chrominance then all of the values will be zero except for the DC coefficient.

2.2.5.3 Thresholding and quantisation

The human eye cannot distinguish fine detail below a certain level. The coefficients generated by the DCT which are below a certain threshold are made equal to zero. The remaining coefficients are quantised with decreasing accuracy as frequency increases.

2.2.5.4 Zigzag scan

All of the matrix coefficients apart from the DC coefficient are read using a zigzag scan into a number sequence. This will maximise the effect of the next two stages.

2.2.5.5 Run length coding

Long strings of consecutive values are replaced by a single code indicating the number of times the value occurs.

2.2.5.6 Huffman coding

A conversion table is used to encode the most frequently occurring values with short codes and the less frequently occurring values with longer codes.

Together these techniques can achieve a compression factor of about 8 times, resulting in a typical I coded frame size of 100Kbytes.

2.2.6 Predictive coded (P) frames

It is common for two consecutive video frames to be very similar, hence it is only necessary to transmit the differences between the previous frame and the current one. A P frame uses this technique and is typically 33Kbytes in size. P frames require the data of a previous I or P frame and cannot be decoded independently.

2.2.7 Bidirectionally predictive (B) coded frames

The B coded frame uses both previous and/or future I and P frames as references, and can achieve a very high level of compression, a typical size being only 12Kbytes. B frames cannot be decoded independently, and are not used as references.

2.2.8 Extension and user data 0x000001B2

The specification allows for user data to be added to the video sequence which can be used for any purpose. User data follows the picture header. This project is concerned with active format descriptors (AFD) which are included as user data by the BBC to convey aspect format information to the decoder.

2.3 Active format descriptors

AFD's are used by all of the UK broadcasters to inform digital receiving equipment, ie set top box (STB) or integrated television, of the desired display format for the current picture. From the start of digital television in the UK it has been the aim of the broadcasters to present an increasing amount of material in the new widescreen (16:9) format. There is a great deal of debate over whether this is necessary, but this is not relevant to this project.

All programme material, widescreen in origin or not, is broadcast as a full height anamorphic 4:3 frame of video. In the case of 4:3 programme material, vertical black bars will be added to the left and right edges of the active frame prior to transmission, so when it is un-squeezed to fill a 16:9 screen the correct aspect will be maintained. As a consequence vertical black bars will be present. On a 4:3 screen the vertical black bars are removed by the STB and the complete picture will be seen. This is shown in diagram 2.6.

If the programme material is in 16:9 format then the frame will also be squashed in the horizontal direction for transmission and un-squeezed by the STB to fill the screen of a widescreen television. In the case of a 4:3 television the STB must either add horizontal black bars or crop the left and right edges of the frame. The viewer can make this choice. This process is shown in diagram 2.7.

In order for the STB to make the correct aspect ratio conversion it must know how the programme has been made and is intended to be viewed. AFD codes are used for this reason.

A list of seven AFD values has been developed, these each specify how the programme material was shot and edited, and define how it should be presented on normal and widescreen receivers. A complete list of AFD values and their meanings is included in table A.6.

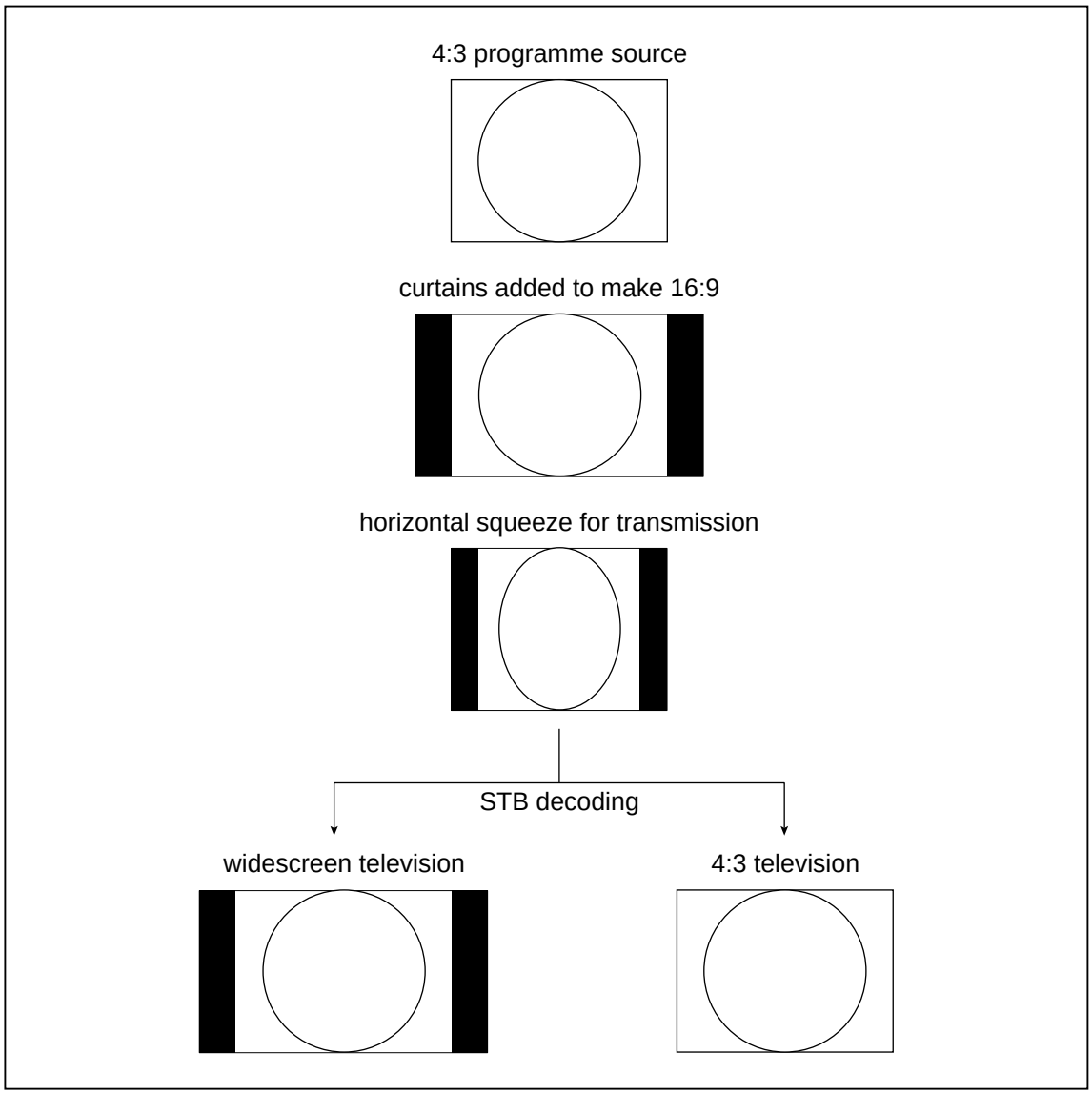

Diagram 2.6, Transmission and decoding of 4:3 material

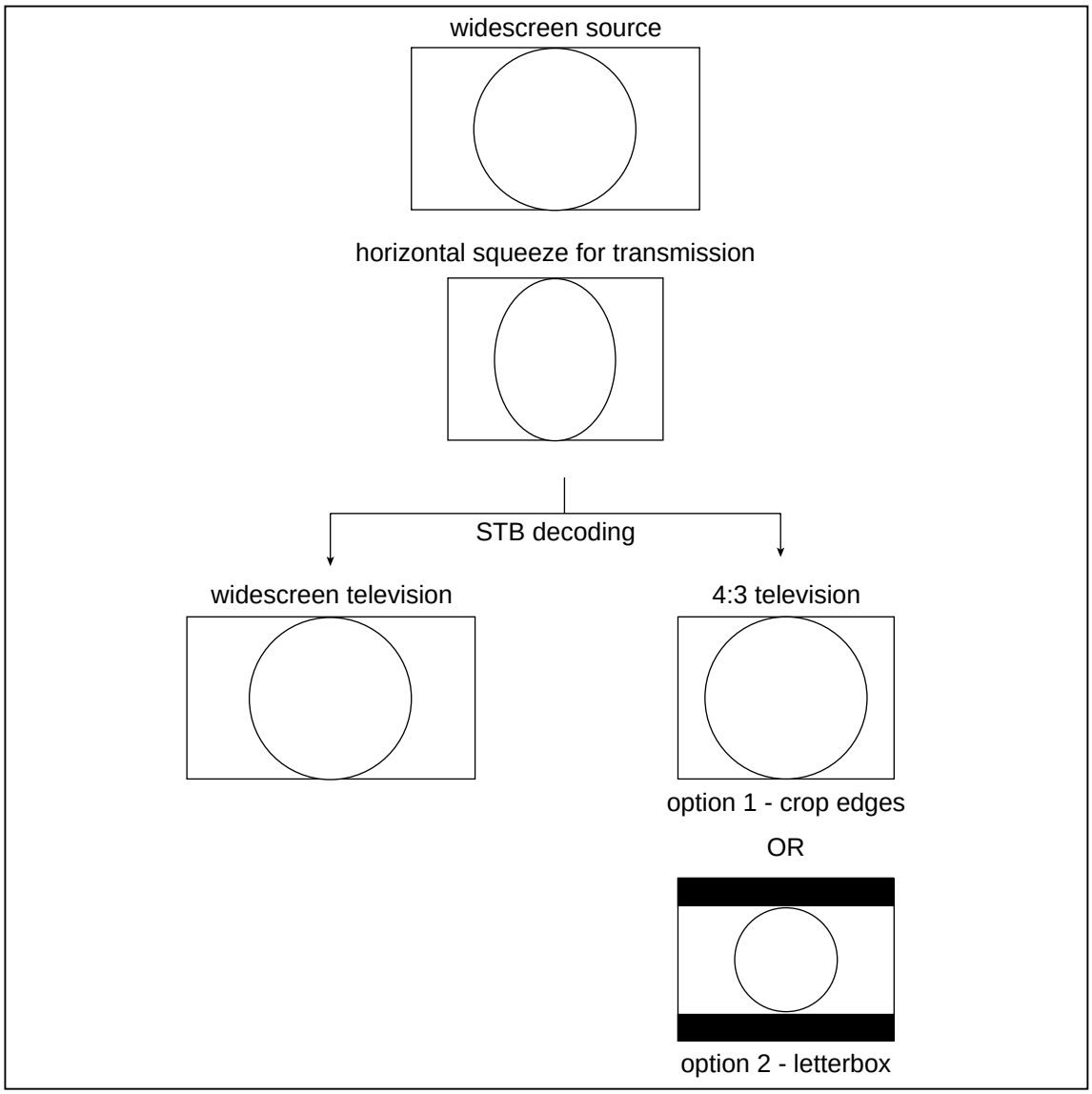

Diagram 2.7, Transmission and decoding of 16:9 material

Chapter 3 **Design**

The software development element of the project required the vast majority of the time available. Difficult targets were set, most of which have been achieved. The software went through many stages of development and was constantly improved. A useful and functional tool was produced which is included on the accompanying CD-ROM.

3.1 Overall design

After researching the structure of MPEG transport streams work began on a software tool to decode and display important pieces of information. It was clear from the start that with such a huge amount of data there must be provision for stepping through the headers of a TS, so the user can quickly and easily find the information they are interested in. A simple and easy to use interface would also be important. Facilities to de-multiplex a TS into a PES or elementary video stream be useful.

3.2 Development environment

The MPEG specifications are written in a pseudo C code, so this was the obvious choice as a programming language. To make the application easy to use a graphical user interface would be required. Microsoft Visual C++ 6 was chosen as it fulfilled these requirements.

3.3 Overview of the stages of development

Very large (500Mb) transport stream files were supplied on CD-ROM. To enable testing of the software it was necessary to open these files in a hex editor. This was not possible with such large files, so a small utility, extract 1Mb.exe, was written to copy the first 1Mb of data into a new file.

Work started by developing a DOS command line utility, tsdecode.exe, which would find TS synchronisation bytes and lock onto the stream so that TS headers could be viewed in sequence. This was expanded to add a PID filter facility, so only TS packets which matched a specified PID value were displayed. Further improvements to this utility were made. A table listing the addresses and values of all the PIDs in the stream can be displayed and saved as a text file. A PES can be extracted from the TS and saved as a new file.

A second DOS utility, pesdecode.exe, was developed to analyse the PES streams extracted by tsdecode. This utility will open a PES and decode the header, which includes the stream ID.

The two DOS utilities were very important to the development of the final application. It was very useful to be able to concentrate on decoding the stream headers without the complications of Windows programming. However, for the application to be more functional and user friendly a Windows version was required. A simple dialog type application was designed, with a toolbar and menu system for controlling stream position. The main dialog area is used to display the current transport stream header and file information. This basic idea has been expanded and improved over several months into the final version which is found on the CD-ROM.

All of the techniques and methods used in the first two utilities have been incorporated into the Windows application, they will be described fully in the following sections.

3.4 Transport stream synchronisation int find_TS_sync_byte()

The first step in developing the software was to open a TS and find the TS synchronisation bytes. As described in section 2.1.3 every transport packet is 188 bytes long and begins with 0x47. It is possible for the value 0x47 to appear elsewhere in the stream, other than as a synchronisation code. Therefore to be certain of locking onto the stream five synchronisation bytes must be found at 188 byte intervals.

This was achieved by using a while loop and switch case statement. Bytes are read from the file in succession until the value 0x47 is found. The file pointer then moves forward 187 bytes to where the next synchronisation byte should be found. This is repeated until 5 synchronisation bytes have been found. If a synchronisation byte is not found where it is expected then the occurrence counter is reset and a byte by byte search resumes for the next synchronisation byte. If the end of file is reached before 5 synchronisation bytes are found an error is generated. A flow chart is shown in diagram 3.1 and the source code in listing B.1.

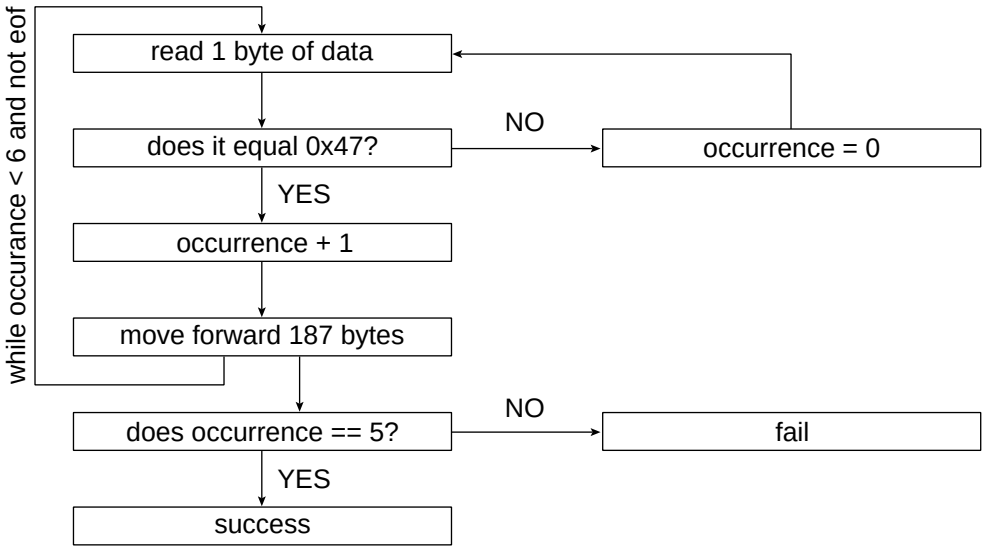

Diagram 3.1, Method for finding 5 synchronisation bytes

3.5 TS header decoding int decodeTSheader()

After the synchronisation code, 0x47, the remaining 3 bytes of the TS header are used for 7 data fields. A complete TS header data structure can be seen in table A.1. The payload start and PID fields must be decoded to enable stream filtering and de-multiplexing. Unfortunately the data fields are not byte or word aligned, which makes extracting the information much more difficult. The problem was solved by using bit masking and shifting techniques. This involves using the binary AND, OR and SHIFT operators to extract the required length of binary string from the 8 bit string (1 byte) which is read from the file. Two examples from the source code are shown below.

```
// the PID is the last 5 bits of byte 1 and all of byte 2
TS_header.PID = ((TS_raw_header[1] & 31) << 8) | TS_raw_header[2];
// the continuity counter is the last 4 bits of byte 3
TS_header.continuity_counter = (TS_raw_header[3] & 0xF);
```

The data decoded from the header is displayed in the main dialog and used by the searching and exporting functions. The PID value identifies which packetised stream the payload of the TS packet belongs to. If the payload start field of a TS header equals 1 then a PES packet begins in the current TS packet payload. This means a PES header will directly follow the TS header. The TS continuity counter increments with each TS packet of the same PID, upon reaching 15 it will loop back to zero. The complete function can be seen in listing B.2.

3.6 PES header decoding void decodePESheader()

A PES header will follow a TS header when the payload start indicator is set to 1. The PES header will identify the current PES type; video, audio or data by means of the stream ID field. The PES header structure can be seen in table A.2.

The stream ID, presentation time stamp (PTS) and decoding time stamp (DTS) are decoded using the method described in section 3.5. The information is presented to the user in the lower section of the main window.

3.7 Video and AFD header decoding void decodeVideoheader()

If the stream ID of the PES header indicates a video stream then the video header will be decoded. The video header starts immediately after the PES header, unfortunately the PES header length is not fixed but specified in the header. This length must be decoded before the start of the video header can be found.

In the case of a video sequence header, video picture properties are decoded. A search is then made for the next picture header which will contain the frame type. AFD data, when present, will be found preceding the picture header.

A picture header contains only the picture coding type. In all cases bit masking and shifting techniques are used to extract the data. A detailed description of the data fields displayed to the user is in section 3.12.8 and 3.12.9. Sequence, picture and AFD header structures can be seen in tables A.3 - A.5. The source code for this function is shown in listing B.3.

3.8 Stream navigation void OnNext(), void OnBack()

The navigation controls allow the user to move forwards and backwards through the stream examining all of the TS headers. It is possible to filter the stream so that the next header to match the users criteria is found and displayed. Four independent filters may be applied in any combination.

Payload start	Find the next TS header where payload start = 1
PID <value>	Find the next TS header with the specified PID value
Sequence start	Find the next video sequence start code (0x000001B3)
AFD <value>	Find the next AFD with the specified value

When the user clicks the ‘next’ button TS, PES, video and AFD headers are decoded in sequence until all of the search criteria are met. If the end of the file is reached before a match is found then an error message will be displayed. It is also possible to search backwards. The source code for the OnNext() function is shown in listing B.4.

3.9 TS header export void OnFileExportTextfile()

The TS header export feature allows the user to create a text file of TS header data. Start and end packets may be specified, the user can also choose which of the TS header fields are included in the file. After the user has entered an output file path in a save dialog the export will begin. The process is shown in diagram 3.2. A header is added to the text file which contains the source file path, date and time. The TS header data is presented in columns, each with a heading.

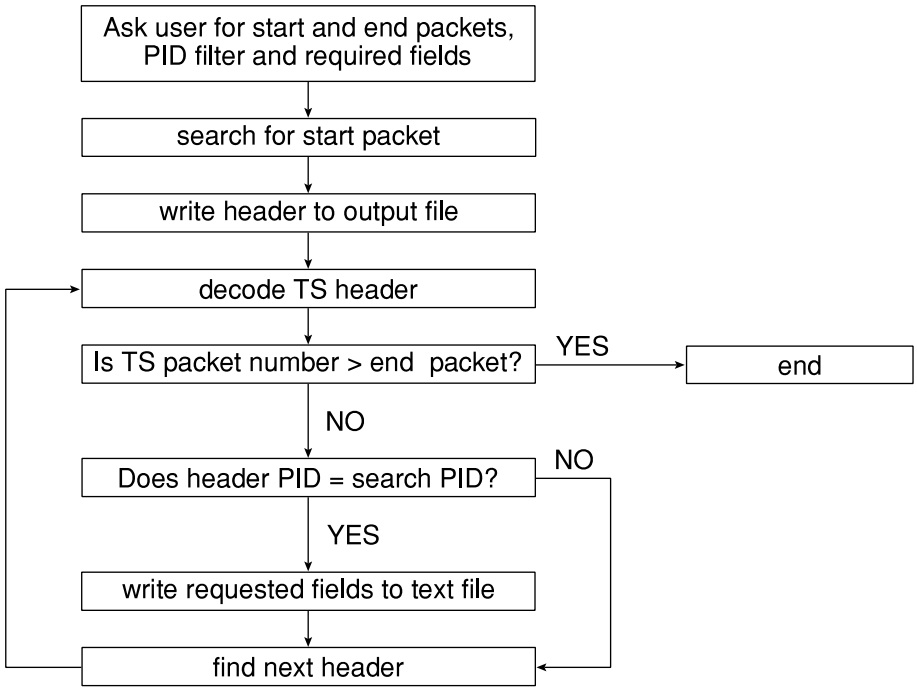

Diagram 3.2, TS header export

3.10 PES export void OnExtractPES()

A PES stream is created by combining all the payload data from TS packets with the same PID into a new file. The PES stream must start with a PES header, so extraction cannot start until the TS header payload start indicator is set to 1. The process is shown in diagram 3.3. During extraction the TS header continuity field is checked to ensure no data is missing. PES streams cannot be played using a MPEG-2 video player because the data is still packetised and contains PES headers.

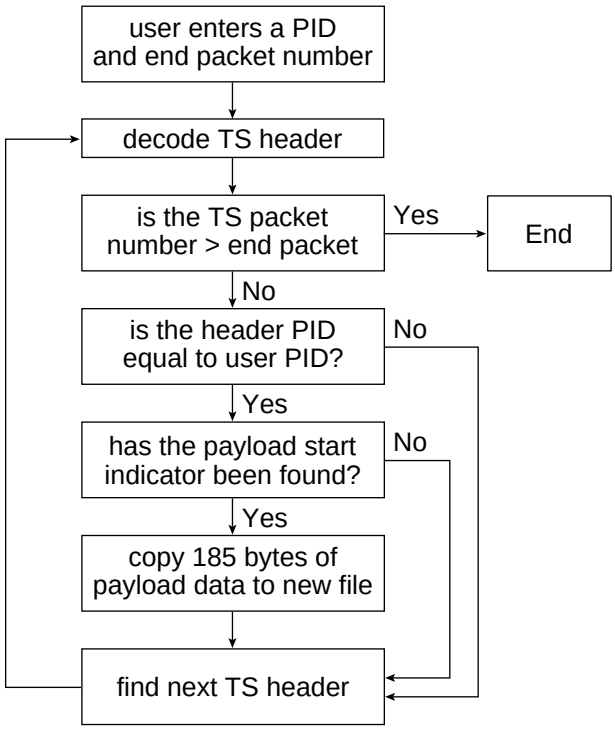

Diagram 3.3, PES export

3.11 MPEG-2 video export void OnFileExportElementarystream()

This is the most complicated of the export functions. The payload data of TS packets with the chosen PID must be combined in sequence and TS headers, adaption fields and packet headers removed. The process can be summarised as follows:

1 User chooses a video PID, end packet number and export filename
2 A search is made for the video sequence start code of the chosen PID
3 Data is copied from the start of the video sequence code to the end of the TS packet
4 The next TS packet with the correct PID is found
5 Check for correct continuity and alert user of any problems
6 Data is copied from the end of the headers to the end of the packet
7 Steps 4 to 6 are repeated until the end packet number is reached

The search process must find the sequence start header for the chosen video PID before extraction can begin. The sequence header must be at the start of the MPEG-2 video file. Data following the sequence header is copied until the end of the current TS packet payload. From this point forward the payload of every matching TS packet is required. However the raw data length will not always be 184 bytes due to adaption headers and PES headers. These must be detected and removed. Any subsequent video sequence start headers are required. Diagram 3.4 shows the procedure in more detail. The source code can be seen in listing B.5.

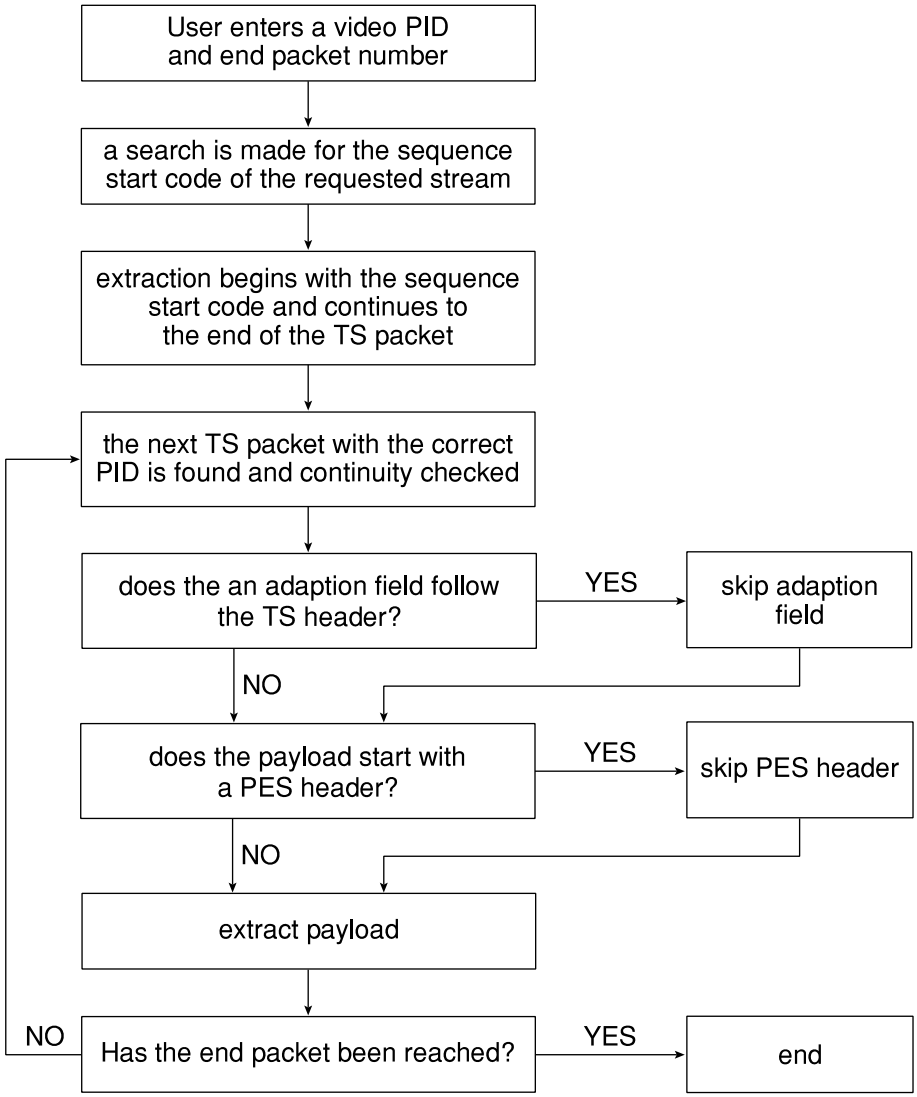

Diagram 3.4, MPEG-2 video extraction

3.12 User interface

An easy to use, consistent, well designed user interface is vital to any software application. Throughout the project considerable time and effort has been spent on creating such a user interface for the analyser. Many common Windows tools and practises have been adopted to give the application a familiar look and feel.

A detailed on line help system has been included. If a user enters incorrect or invalid information an error message is generated. To ensure consistence with other Windows applications reference was made to the Microsoft document ‘Windows interface guidelines for software design’. Feedback from users has also been used to improve the interface. The main window showing the menu, toolbar, filter options, TS header details, PES header details and status bars is shown in figure 3.5.

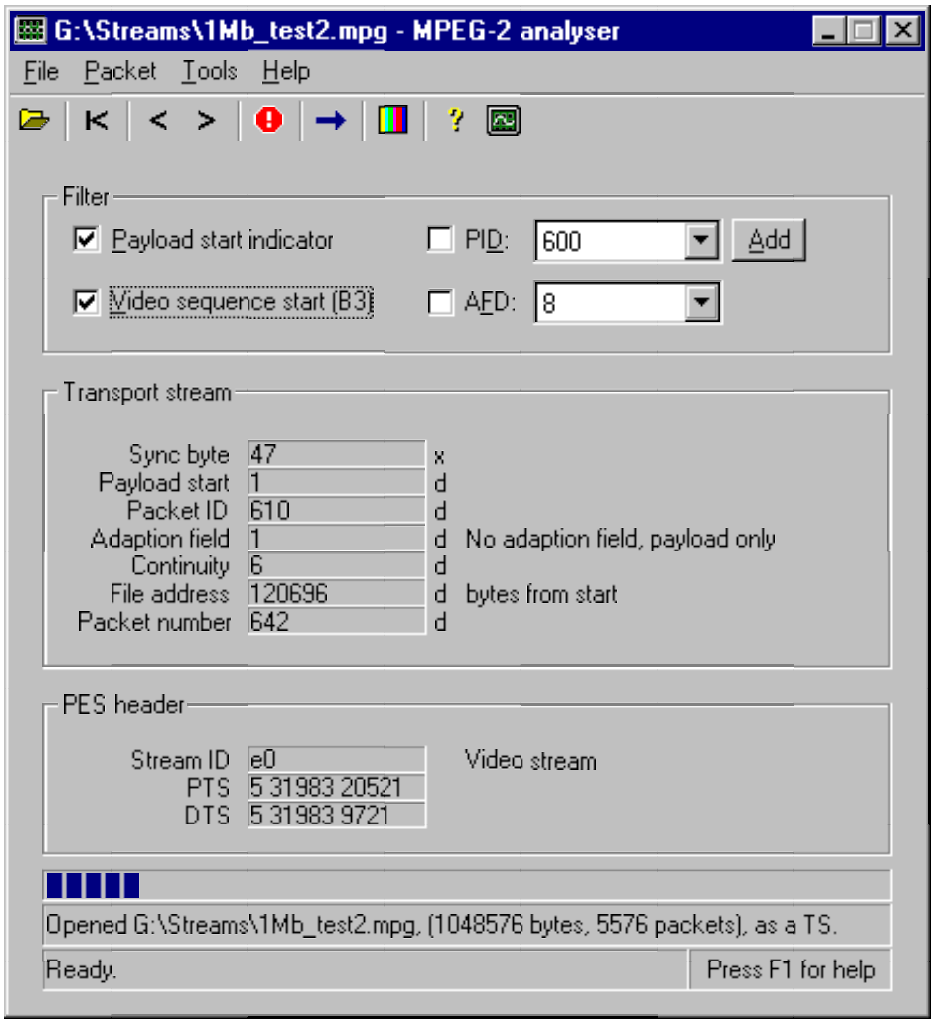

Figure 3.5, MPEG-2 analyser main window

3.12.1 Menu system IDR_MENU1

A menu bar at the top of the main dialog gives access to file and navigation commands. All menu items can be accessed by keyboard shortcuts. The naming of menus is consistent with other Windows applications.

3.12.1.1 File | Open

A standard Windows open dialog will appear from which a MPEG-2 transport stream should be chosen. The default file type filter, *.mpg, is applied. PES files and MPEG-2 video streams cannot be opened. If the input file does not contain TS synchronisation bytes an error message is generated.

3.12.1.2 File | Export

See section 3.12.11

3.12.1.3 File | Exit

Leave the program, all open files are closed.

3.12.1.4 Packet | First

Find, decode and display the first packet in the transport stream.

3.12.1.5 Packet | Back
Find, decode and display the previous TS header which matches the filter criteria.

3.12.1.6 Packet | Next
Find, decode and display the next TS header which matches the filter criteria.

3.12.1.7 Packet | Goto IDD_GOTO
Open a dialog into which a TS packet number can be entered. The specified packet will be found and displayed. If the packet does not exist a message box will alert the user.

3.12.1.8 Tools | Options IDD_OPTIONS
Open a dialog from which the number format used to display TS header data can be changed. The user may choose decimal (base 10) or hexadecimal (base 16). The same number format will be used in an exported text file.

3.12.1.9 Help | Contents
The contents of the help file is displayed.

3.12.1.10 Help | About IDD_ABOUTBOX
Open the about dialog which displays the software version number and website address.

3.12.2 Toolbar IDR_TOOLBAR
To make navigation through the stream quick and simple, a toolbar has been added to the top of the main window. Most buttons replicate functions on the menu system, the other buttons are described in sections 3.12.2.1 - 3.12.2.3.

3.12.2.1 Stop

If the user wishes to abort a search they may do so by clicking on the stop button. The first packet in the stream will then be displayed.

3.12.2.2 View video headers

Open or close the floating video header window. A full description is given in section 3.12.8.

3.12.2.3 Help

Display the help page for the main window.

3.12.3 Filter options IDD_MAIN
The top section of the main window contains the TS filter options. Four filters can be applied in any combination. In the case of PID and AFD, a search value can be specified. When a filter is checked clicking ‘next packet’ will cause the program to search through the stream until a packet which satisfies all of the filter requirements is found. The packet is then decoded and displayed, and the search stops. Similarly, the filters are also applied to the ‘previous packet’ button. The filters are each described in detail in sections 3.12.3.1 - 3.12.3.4.

3.12.3.1 Payload start
TS headers are examined in sequence until the payload start indicator equals 1. When this is true a PES header will follow the TS header. The PES header is automatically decoded and the stream ID displayed in the lower section of the main window. This is described in more detail in section 3.12.5.

3.12.3.2 PID <value>

Every TS header contains a PID value, headers are decoded until the specified PID value is found. A drop down list contains common PID values, additional values may be added by clicking on the 'Add' button. After adding a new value it is automatically selected.

3.12.3.3 Sequence start

Video headers are examined until the video sequence start code (0x000001B3) is found. The most detailed picture information is available in the video sequence start header.

3.12.3.4 AFD <value>

One of the most important parts of the application is the AFD search facility. When present in a stream, AFD values (see table A.6) are coded in the user data before a picture start header. An AFD code may be chosen from the drop down list and searched for. Only the values 8 - 15 inclusive are permissible.

3.12.4 TS header details IDD_MAIN

All of the useful fields in a TS header are displayed in the main window. The number format of many fields can be chosen to be decimal or hexadecimal by opening the options dialog on the tools menu. The number format is displayed to the right of the data box. Additional fields which are not in the specification have been added to show the file address and header number.

Additional information is shown to the right of the TS header data, this includes synchronisation errors, stream names, and adaption header information codes.

3.12.5 PES header details IDD_MAIN

The PES header will follow a TS header when the payload start indicator is set to 1. Three data fields from the PES header are displayed in the lower section of the main window. The stream ID field indicates if the PES contains video, audio or other data. The presentation time stamp (PTS) and decoding time stamp (DTS) fields can be used to determine the position of the current frame within a GOP sequence. The values are not in time format, but are integers which increment up to 2^{33} before repeating.

3.12.6 Progress and status bars IDD_MAIN

At the bottom of the main window are two status bars and a progress bar. The progress bar is used to give a visual indication of the current stream position. During long searches the bar is constantly updated to give an indication of how quickly the search is progressing. The status bars give information about the currently open TS file and indicate when a search is taking place.

3.12.7 Options IDD_OPTIONS

The options dialog box is opened from the tools menu. The display format of TS header data can be chosen to be decimal (base 10) or hexadecimal (base 16). Changes will be made instantly but are not saved.

3.12.8 Video header details IDD_VIDEO

Video header details are displayed within a floating header viewer, shown in figure 3.6. The window can be opened or closed at any time, but will always show the current data. The video header data is decoded whenever a video PES is encountered. The level of information available depends on the type of video header. All of the fields are completed for a sequence header, but only the fields described in sections 3.12.8.1 - 3.12.8.3 are present in a picture header.

Once a field has been completed the data will remain visible until it is replaced by new data or the clear button is pressed.

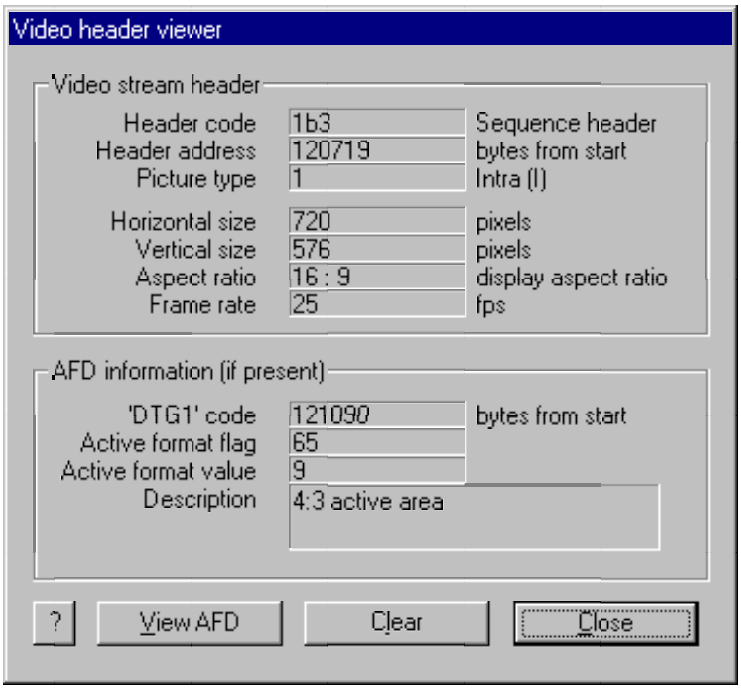

Figure 3.6, Video header viewer

3.12.8.1 Header code

For a picture header this will be 100, or 1B3 for a sequence header. The header type is described to the right of the value.

3.12.8.2 Header address

The location of the video header in the file, shown as the number of bytes from the start of the TS file. This information may be used to examine the raw header in a hex editor, by using the editors ‘goto file address’ function.

3.12.8.3 Picture type

As described in sections 2.2.5 to 2.2.7 there are three picture coding types, I, P and B. The coding type used for the current picture is displayed in this field.

3.12.8.4 Horizontal size

The number of horizontal pixels used the create the frame. Broadcast video streams use 720.

3.12.8.5 Vertical size

The number of vertical pixels used the create the frame. Broadcast video streams use 576.

3.12.8.6 Aspect ratio

Specifies the display aspect ratio for the frame, either normal (4:3) or widescreen (16:9).

3.12.8.7 Frame rate

Broadcast video in the UK is always made up of 25 frames or pictures per second, although other values are permitted within the MPEG-2 specification.

3.12.9 AFD header details IDD_VIDEO

AFD codes are contained with the user data which precedes picture data. As described in section 2.3, AFDs are used to inform receiving equipment of the intended display format for the material. When present in a stream, the AFD data is displayed in the lower half of the video header viewer.

3.12.9.1 'DTG1' code

DTG1 is used to identify AFD data, this field will show the file location of the current AFD.

3.12.9.2 Active format flag

AFD header data

3.12.9.3 Active format value

The active format type is identified by a number between 8 and 15. A table of AFD values can be seen in table A.6.

3.12.9.4 Description

A short description which explains the meaning of the AFD value.

3.12.10 AFD viewer dialog IDD_VIEW_AFD

To clarify the meaning of the AFD value a simulator is included. The AFD viewer (figure 3.7) will show the effect of the current AFD value on a normal (4:3) and widescreen (16:9) receiver. The pictures are for illustrative purposes only and have no relation to the contents of the stream.

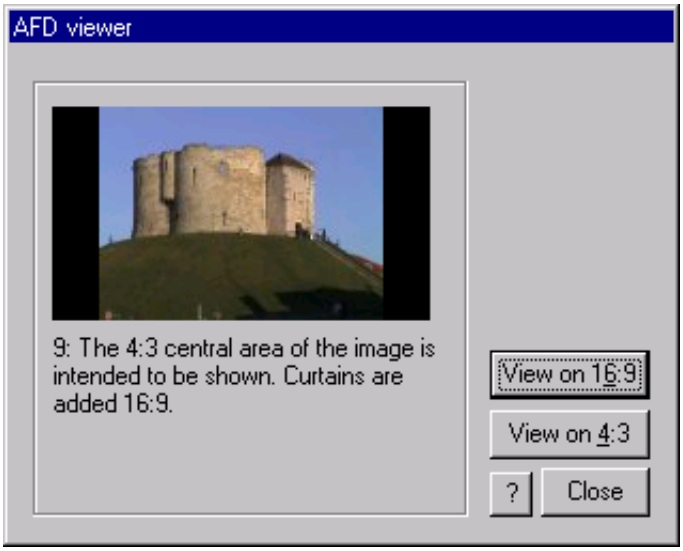

Figure 3.7, AFD viewer

3.12.11 Export dialogs EXPORT_ES, EXPORT_PES, EXPORT_TEXT

The transport stream can be exported in three different ways, an export method is chosen from the file | export sub menu. In each case a portion of the current TS can be chosen for processing by entering TS packet numbers into the export dialog box.

3.12.11.1 Export MPEG-2 video

A video PID must first be chosen using the drop down menu on the main form. The 'Save As' button will open a standard Windows save dialog. The MPEG-2 video file produced can be played on any MPEG-2 video player.

3.12.11.2 Export PES

A PID must first be chosen using the drop down menu on the main form. The ‘Save As’ button will open a standard Windows save dialog.

3.12.11.3 Headers as text file

The transport stream header data can be saved as a text file for further analysis. The export dialog allows the user to choose start and end packets, and which TS header fields are included in the file. The PID filter and display options used on the main dialog will be applied to the output file.

3.12.12 Help system

Comprehensive online help has been created using the standard Windows help system. Every dialog box has a help button which will open the relevant page. Help can also be accessed by pressing F1 at any time. The help file contains a contents and detailed index. Buttons on the toolbar will display tooltips when the mouse pointer is held over a button.

3.13 Configuration

A configuration file (mpeg2.ini) is included in the main program directory. This file may be edited in Windows notepad to set the default path used by the open dialog box.

3.14 Installation

The application may be run directly from the CD-ROM. An InstallShield wizard is supplied to install the program files and some test streams. A shortcut is added to the Windows start menu. The application may be removed using the add/remove programs utility in Windows control panel. No files are added to the Windows system directory or registry.

3.15 System requirements

The software has been fully tested with the Microsoft Windows 98 and NT operating systems. There are no special hardware requirements. Installation requires 3Mb of hard disk space. The software does not require any additional memory.

Chapter 4 **Results and analysis**

Due to the nature of the project no final results can be presented in this report, however this chapter will show how the data produced by the software has been verified to be correct.

4.1 TS synchronisation

For decoding to take place, synchronisation with TS headers must be perfect. It is for this reason that the software constantly checks to ensure the synchronisation byte value is equal to 0x47. If for any reason this is not the case a message box will alert the user and the TS file will be closed.

4.2 Header decoding

Throughout development the header information displayed by the application was checked against the raw data in the stream. A hex editor was used to view the raw data and find the start of TS packets. The hex values were converted into binary form, checked against the specification and verified.

The streams supplied on CD-ROM were created from the BBC's public service digital output. It is known which PID values are used by these streams and so the application was checked to ensure all the values were correct.

4.3 Exporting

The files created by the MPEG-2 export function have been checked and passed by an industry standard MPEG-2 video stream decoder, mpeg2dec.exe. Written by the MPEG-2 software simulation group this utility will check that a MPEG-2 video stream conforms to the specification and report any errors.

PES extraction was checked by performing the extraction by hand in a hex editor and comparing this file to that produced by the software. Generated streams (see section 4.4) were also used.

The exported text file of TS header data was checked against the TS headers displayed in the main window of the application.

4.4 Generated streams

To test the software fully under many input conditions some transport streams were generated manually using a hex editor. These streams comply with the MPEG-2 specification although they contain no actual picture data. For example, a stream is included on the accompanying CD-ROM which has all of the AFD values in order. This has been used to ensure the correct descriptions and simulation pictures are shown for all AFD values. An existing test stream was modified so the first TS header was not at the start of the file, this was used to test the software's ability to cope with such streams.

4.5 End user testing

In addition to all of the methods described the software was under constant evaluation by Andrew Smith, a BBC engineer who specialises in digital services. He was able to check the results produced were consistent and correct.

Chapter 5 **Discussion**

The software has seen many stages of development and improvement. A useful tool has been produced, although there is still much potential for further work.

5.1 Design changes

During the course of the project much has been learnt about MPEG-2. It became clear towards the end, that the design of the interface and search / filter functions was not ideal. This led to a major re-write of the code, resulting in a much improved software tool. The changes are briefly documented below.

TS header decoding and navigation was the first stage of the program to be completed. Next picture property decoding was added. This was initially achieved by performing a byte by byte search from the start of the file until the video sequence start code was found. A very similar method was used to add AFD header decoding. There are several drawbacks with this approach:

- searches take a long time
- it is impossible to know which video stream the header belongs to
- backward searches are not possible

In order to solve the above problems a new approach was adopted. All searches would start with the TS header and work through the stream in a tree like manner. This means all search results can be linked directly back to a TS packet, which in turn gives the PID value. Searches are also much quicker, only headers are examined rather than the entire file. Backward searches are possible because the program always knows the location of the previous TS header.

These internal changes also allowed a better user interface. All of the search functions could be grouped together in the main window. The same ‘next’ and ‘back’ buttons are used for every search. A floating, constantly updated, video header viewer was added to display search results.

5.2 Future improvements

Although the application is useful as it stands, there is potential for many more features and functions to be added. Andrew Smith has made several suggestions, which are documented in sections 5.2.1 - 5.2.5.

5.2.1 Still video frame display

It was the intention from the start of the project to display still video I-frames. This would require development of a JPEG style decompressor. A significant amount of time would be required to implement this.

5.2.2 Audio decoding

It advantageous to decode and display the audio streams as a visual waveform along side video frames. This would help solve lipsync problems in conjunction with a ‘hot lips’ test stream.

5.2.3 Subtitle decoding

Subtitle text decoding would be very useful for solving problems with subtitle encoders.

5.2.4 Automatic PID list generation

Currently the list of PID filter values is hard coded, additional values may be added by the user at run time. However this could be greatly improved by decoding the service information table within the stream, which lists the actual PID values present, and generating the list from this data.

5.2.5 Data viewer

A hex dump display could be built into the program which will display the raw data for a packet. This would allow more detailed analysis by hand decoding.

5.3 Other applications

As stated in the introduction, MPEG standards are used in many digital video and audio applications. Although the application and transmission or storage medium may vary greatly the data structures and compression techniques are very similar. The software produced by this project could be easily modified to analyse the MPEG data used in many other applications; digital radio, DVD, internet broadcasting etc.

Chapter 6 **Conclusion**

Digital broadcasting in the UK is here to stay, the current government policy is to completely switch off analogue transmission within the next ten years. The MPEG system is clearly the future of broadcasting, the research undertaken by this project has given the author knowledge vital for an engineering career in broadcasting.

During the development of the software the advice and suggestions given by Andrew Smith were extremely helpful, and have certainly resulted in a more useful and user friendly tool.

Andrew was asked if the software is currently useful:
"The simple answer is yes. The ability to scan TS files for pertinent information, particularly AFDs, will help us a great deal. The development potential of having an 'in house' analyser is fantastic because it means we are not at the mercy of third party developers who have different priorities to our own."

When asked about the user interface, Andrew replied:
"User interface is pretty straightforward except for some of the icons. If the buttons displayed their application when the mouse is held over them this would help. I have not referred to the help files so it must be OK?" (Since receiving this reply tooltips have been implemented.)

This project has certainly been successful in producing a useful tool and providing a great deal of knowledge for the author. As previously stated however, there is a vast potential for additional features and functionality.

Bibliography

MPEG specifications

International standard ISO/IEC 13818-1 Information technology – Generic coding of moving pictures and associated audio information: Systems
International standard ISO/IEC 13818-2 Information technology – Generic coding of moving pictures and associated audio information: Video
PES/ES stream details for AFD, Nick Tanton, BBC Research and Development

MPEG resources

BBC Training and Development, MPEG-1 and MPEG-2, Andy Woodhouse

Programming resources

C++ from scratch, Jesse Liberty, QUE
Crash course in C, Paul J Perry, QUE
Problem solving using C, Yuksel Uckan, McGraw-Hill
Practical C programming, Steve Oualline, O'Reilly and Associates
Using visual C++, John Bates and Tim Tompkins, QUE
Data structures and algorithm analysis in C, Mark Allen Weiss, Addison-Wesley
Microsoft developers network (MSDN) CD-ROM and website
Windows interface guidelines for software design, Microsoft

Web sites

AFD information, http://afd.8bit.co.uk/
Code guru, http://www.codeguru.com/
BBMpeg software, http://members.home.net/beyeler/bbmpeg.html
Elecard MPEG-2 player, http://www.elecard.com/
LSX MPEG-2 video player, http://www.ligos.com/
MPEG background, http://bmrc.berkeley.edu/projects/mpeg/mpeg_overview.html
MPEG test streams, ftp://ftp.tek.com/tv/test/streams/Element/index.html
MPEG video resources and software, http://www.mpeg.org/MPEG/video.html
MPEG compression, http://icsl.ee.washington.edu/~woobin/papers/General/node2.html
MPEG-2 data structures, http://www.cs.ucsb.edu/~aduncan/MPEG/MPEG-2_Picts.html
MPEG-2 video test bitstreams, ftp://ftp.tek.com/tv/test/streams/Element/index.html
Introduction to MPEG compression, http://members.aol.com/symbandgrl/

Appendix A **MPEG header tables**

A.1 Transport packet header

Item name	Description	Bits
sync byte	synchronisation code of value 0x47	8
transport error indicator	when set at least one bit error is present in the packet	1
payload start indicator	when set the payload of this transport packet is the start of a packetized elementary stream (PES) packet	1
transport priority	used by the decoder	1
PID (packet identifier)	identifies which PES the data stored in this transport packet payload is from	13
scrambling control	scrambling mode	2
adaption field control	indicates if there is an adaption field (additional information) following the transport stream header	2
continuity counter	4 bit counter which increments with each transport packet containing the same PID, when it reaches 15 it loops back to zero	4

A.2 Packetised elementary stream header

Item name	Description	Bits
start code	0x000001	24
stream ID	stream identifier	8
PES packet length	length of the packet, for video it is not specified	16
sync code	'10'	2
PES scrambling control	scrambling flags	2
PES priority	used by the decoder	1
data alignment idicator	used by the decoder	1
copyright	when set PES packet payload is protected by copyright	1
original or copy	when set PES packet payload is an original	1
PTS DTS flags	10 : PTS fields are present in PES packet header 11 : PTS and DTS fields are present in PES packet header 00 : no PTS or DTS fields are present in PES packet header	2
ESCR flag	when set ESCR fields are present in PES packet header	1
ES rate flag	when set ES rate field is present in PES packet header	1
DSM trick mode flag	when set a DSM trick mode field is present	1
additional copy info flag	when set a additional copy field is present in PES packet header	1
PES CRC flag	when set a CRC field is present in the PES packet header	1
PES extension flag	when set an extension field is present in the PES packet header	1
PES header data length	total number of bytes occupied by the additional fields and any stuffing bytes in this PES packet header	8
additional header data	depending on the flags above, more headers may follow. The length of this section is set by the PES header data length field	
PES stream data	contiguous bytes of data from a stream	

A.3 Video sequence start header

Item name	Description	Bits
sequence start code	0x000001B3	32
horizontal size	width of the picture in pixels	12
vertical size	height of the picture in pixels	12
aspect ratio	0001 : 1.0 (square sample) 0010 : 3:4 0011 : 16:9 0100 : 21:9	4
frame rate	number of frames per second	4
additional data		

A.4 Picture header

Item name	Description	Bits
picture start code	0x00000100	32
temporal reference	reference associated with each picture	10
picture coding type	001 : intra-coding (I) 010 : predictive-coded (P) 011 : bidirectionally-predictive-coded (B)	3

A.5 AFD user data header

Item name	Description	Bits
user data start code	0x000001B2	32
afd identifier	"DTG1" 0x44544731	32
active format flag		8
active format value	hex value in the range 0xF8 - 0xFF. See table A.6	8

A.6 AFD values

AFD value	Description
F8	active region is the same as the coded frame
F9	4:3 active area
FA	16:9 active area
FB	14:9 active area
FC	reserved
FD	4:3 coded image with a shoot and protect 14:9 area
FE	16:9 coded image with a shoot and protect 14:9 area
FF	16:9 coded image with a shoot and protect 4:3 area

Appendix B **Source Code**

B.1 int CMPEG2analyseDlg::find_TS_sync_byte()

```
int CMPEG2analyseDlg::find_TS_sync_byte()
{    // scan file looking for sync_bytes
     BeginWaitCursor();
     m_strMainStatus.Format(”Looking for sync bytes...”);
     UpdateData(FALSE);
     int sync_data = 0; // storage for data read from file
     int occurance = 0; // keep track of how many have been found
     long int current_file_pos = 0; // where are we?

     while ((ftell(ip_file) < ip_file_length)
               && (ftell(ip_file) < 1500) && occurance < 6)
     {
          current_file_pos = ftell(ip_file);
          sync_data = 0;
          fread(&sync_data, 1, 1, ip_file);
          switch (sync_data)
          {    // if sync_byte is found then remember it in sync_bytes[]
          case 0x47 :
               sync_bytes[occurance] = current_file_pos;
               occurance++;
               fseek(ip_file, 187, SEEK_CUR); // look for next one
               break;

          default : // if byte is not equal to 0x47
               occurance = 0;
          }
     }
     if ( occurance < 6 )
     {    // report error if 5 cannot be found
          MessageBox(”Transport stream sync bytes could not be found in
     the input file.”,app_title,MB_OK | MB_ICONSTOP );
          m_strMainStatus.LoadString(MSG_READY);
          UpdateData(FALSE);
          return 0; // fail
     }
     else
          offset = sync_bytes[0];
          TRACE(”\nFirst sync byte is at %d”, offset);
          fseek(ip_file, sync_bytes[0], 0); // go back to the start
          return 1;
}
```

B.2 int CMPEG2analyseDlg::decodeTSheader()

```
int CMPEG2analyseDlg::decodeTSheader()
{
     if (end_of_file == TRUE)
     { return -1; }

     if (ip_file == NULL)
     { MessageBox(err_msg_open,app_title,MB_OK | MB_ICONSTOP );
       return -1; }

     m_cFilePosition.SetPos(ftell(ip_file)); // update progess bar
     m_strMainStatus.Format("Searching...");
     UpdateData(FALSE);

     TS_header.header_address = ftell(ip_file);

     fread(&TS_raw_header, sizeof(TS_raw_header), 1, ip_file); // read header
     TS_header.sync_byte = TS_raw_header[0];

     if(TS_header.sync_byte != 0x47)
          {    CString SyncError;
               SyncError.Format("Synchronisation with the transport stream
               has lost. Sync byte read as: x%x at d%d. The file will now be
               closed.",TS_header.sync_byte,ftell(ip_file));
               MessageBox(SyncError,app_title,MB_OK | MB_ICONSTOP );
               ip_file = NULL;
               clearall();
               return -1; }

     // payload start is bit 2 of byte 1
     TS_header.payload_start_indicator = (TS_raw_header[1] & 0x40) >> 6;
     // priority is bit 3 of byte 1
     TS_header.transport_priority = (TS_raw_header[1] & 0x20) >> 5;
     // pid is the last 5 bits of byte 1 and byte 2
     TS_header.PID = ((TS_raw_header[1] & 31) << 8) | TS_raw_header[2];
     // adaption is bits 3 and 4 of byte 3
     TS_header.adaption_field_control = (TS_raw_header[3] & 0x30) >> 4;
     // continuity is the last 4 bits of byte 3
     TS_header.continuity_counter = (TS_raw_header[3] & 0xF);
     // calculate the header number from the file position
     TS_header.header_number = (TS_header.header_address - offset) / 188;

     if (TS_header.payload_start_indicator == 1) // check for payload start
     { decodePESheader(); } // decode the PES header if found

     // move forward ready for next packet
     if (ftell(ip_file) < ip_file_length - 200) // check for end of file
     { fseek(ip_file, ((188 * (TS_header.header_number + 1)) + offset), 0); }
     else
     { MessageBox("End of input file, no match found for search pattern." ,
      app_title,MB_OK | MB_ICONINFORMATION );
      end_of_file = TRUE; }
     return 0;
}
```

B.3 void CMPEG2analyseDlg::decodeVideoheader()

```
void CMPEG2analyseDlg::decodeVideoheader()
{    // move forward to the start of the video header and decode
     // first move back to the start of the packet we are interested in
     fseek(ip_file, ((188 * TS_header.header_number) + offset), 0);
     // then move to where the video should be...
     file_seek(4 + 9 + PES_header.header_data_length);
     unsigned char raw_video_header[8]; // set up space for data
     video.address = ftell(ip_file);
     // read the first 24 bytes
     fread(&raw_video_header, sizeof(raw_video_header), 1, ip_file);
     fseek(ip_file, -8, SEEK_CUR);

     video.start_code = ((raw_video_header[0]<<24) |
                         (raw_video_header[1]<<16) |
                         (raw_video_header[2]<<8)  |
                         (raw_video_header[3]<<0));

     switch (video.start_code) // decode header type
     {
     case 0x100 :
          video.description = “Picture header”;
          video.coding_type = ((raw_video_header[5] & 0x38) >> 3);
          AFD.address = findHeader(0x44,0x54,0x47,0x31,30); // DTG1 code
          if (AFD.address != 0)
          { decodeAFDheader(); }
          break;
     case 0x1B3 :
          video.description = “Sequence header”;
          video.horz_size = ((raw_video_header[4]) << 4 |
                             (raw_video_header[5] & 0xF0));
          video.vert_size = ((raw_video_header[5] & 0xF) << 8
                             | raw_video_header[6]);
          video.aspect_code = (raw_video_header[7] & 0xF0) >> 4;
          video.frame_rate_code = (raw_video_header[7] & 0xF);
          if (findHeader(0x00,0x00,0x01,0x00,300) != 0) // picture header
          { fseek(ip_file, 1, SEEK_CUR);
            video.coding_type = ((fgetc(ip_file) & 0x38) >> 3); }
          else
          { video.coding_type = 0; }
          AFD.address = findHeader(0x44,0x54,0x47,0x31,300); // DTG1 code
          if (AFD.address != 0)
          { decodeAFDheader(); }
          break;
     case 0x1B5 :
          video.description = “Sequence extension”;
          break;
     case 0x1B2 :
          video.description = “User data”;
          break;
     default :
          video.description = “unknown”;
          break;
     }
}
```

B.4 void CMPEG2analyseDlg::OnNext()

```
void CMPEG2analyseDlg::OnNext()
{
     if (ip_file == NULL || end_of_file == TRUE)
     { return; }

     TS_header.payload_start_indicator = 0;
     TS_header.PID = 0;
     AFD.address = 0;
     AFD.value = 0;
     PES_header.stream_id = 0;
     video.start_code = 0; // reset

     MSG msg;
     UpdateData(); // retrieve data from radio buttons

     if (m_bSequenceFilter == TRUE)
     { m_bPayloadFilter == TRUE; }

     while ( ftell(ip_file) < ip_file_length - 200 )
     {    PeekMessage(&msg, m_wndToolBar, 0, 0, PM_REMOVE);
          if(msg.message == WM_LBUTTONDOWN)
          {
               MessageBox(”Search aborted by user. Stream position will
               be reset to packet 1.”,app_title,MB_OK | MB_ICONSTOP );
               OnPacketFirst();
               return;
          }

          decodeTSheader();
          if(( m_bPayloadFilter == FALSE
               || TS_header.payload_start_indicator == 1)
               && (m_bPIDfilter == FALSE
               || TS_header.PID == search_PID)
               && (m_bSequenceFilter == FALSE
               || video.start_code == 0x1B3)
               && (m_bAFDfilter == FALSE
               || AFD.value == search_AFD))
               {
                    UpdateDetails();
                    return;
               }
     }

     MessageBox(”End of input file, no match found for search pattern.”
     ,app_title,MB_OK | MB_ICONINFORMATION );
     OnPacketFirst();
     return;
}
```

B.5 void CMPEG2analyseDlg::OnFileExportElementarystream()

```
void CMPEG2analyseDlg::OnFileExportElementarystream()
{
     if (ip_file == NULL)
     {    MessageBox(err_msg_open,app_title, MB_OK | MB_ICONINFORMATION );
          return; }

     dlgExportES ExportES(this);
     ExportES.m_maxpackets_es = max_packets;
     if (ExportES.DoModal() == IDOK)
     {
          if (ExportES.m_maxpackets_es > max_packets)
          { MessageBox(”There are not that many packets in the input
          stream.”,app_title, MB_OK | MB_ICONINFORMATION );
          return;  }
     const char* psz = “MPEG-2 video (*.m2v)|*.m2v|MPEG-2 stream
     (*.mpg)|*.mpg|PES stream (*.pes)|*.pes|All files (*.*)|*.*||”;
     CString fileExt(psz);

     CFileDialog fileDlg(FALSE, NULL, NULL, OFN_HIDEREADONLY |
     OFN_OVERWRITEPROMPT, (CString)fileExt, NULL);

     if (fileDlg.DoModal() == IDOK)
     {
          CString op_es_filepath;
          char* pszFileName;
          op_es_filepath = fileDlg.GetPathName();

          if (op_es_filepath.Find(’.’) == -1)
          { op_es_filepath += “.m2v”; }

          pszFileName = (char*)op_es_filepath.operator const char*();

          CFile op_es_file;

          if ( !op_es_file.Open( pszFileName, CFile::modeCreate +
          CFile::modeWrite + CFile::typeBinary ) )
          {
          MessageBox(”File could not be created, check write protection and
          access permissions.”,app_title, MB_OK | MB_ICONERROR);
          m_strMainStatus.Format ( “Can’t open file %s for writing.”,
          pszFileName );
          UpdateData(FALSE);
          return ;
          }

     char data[184];
     char data1[1];
     int loop = 0;
     int next_packet = 0;
     int msgresponse = 0;
     int payload_start_found = 0;
     int seq_start_found = 0;
     int seq_start_add = 0;
     int seqloop = 0;
```

```
int adaption_length = 0;
int PES_header_length = 0;
int skip_length = 0;
int start_position = 0;
int data_length = 0;
long seq_start_code = 0;
unsigned int seq_start_code0;
unsigned int seq_start_code1;
unsigned int seq_start_code2;
unsigned int seq_start_code3;
unsigned int bytes_written = 0;

BeginWaitCursor(); // show hourgalss
OnPacketFirst();   // go back to start
UpdateWindow();    // redraw main window

while (decodeTSheader() != -1
&& TS_header.header_number < ExportES.m_maxpackets_es)
{
     if (TS_header.PID == search_PID)
     {
          if (seq_start_found == 0 && payload_start_found == 0
          && TS_header.payload_start_indicator == 1)
          {
               start_position = ftell(ip_file);
               fseek(ip_file, -176, 1);
               if (TS_header.adaption_field_control == 3)
          {
               fseek(ip_file, -8, 1); // go back to end of TS header
               adaption_length = fgetc(ip_file);
               //   skip_length = adaption_length + 1;
               fseek(ip_file, adaption_length, 1);
               fseek(ip_file, 8, 1);
               }

               PES_header_length = fgetc(ip_file);
               fseek(ip_file, PES_header_length, 1);
     seq_start_code0 = fgetc(ip_file);
     if (seq_start_code0 == 0x00)
     { seq_start_code1 = fgetc(ip_file);
       if (seq_start_code1 == 0x00)
       { seq_start_code2 = fgetc(ip_file);
           if (seq_start_code2 == 0x01)
           { seq_start_code3 = fgetc(ip_file);
               if (seq_start_code3 == 0xB3)
               { seq_start_add = ftell(ip_file) - 4;
                 seq_start_found = 1;

               fseek(ip_file, seq_start_add, 0);
               data_length = (188-13-PES_header_length.adaption_length);
               for (loop = 0; loop < data_length; loop ++)
               {
               fread(&data1, 1, 1, ip_file);
               op_es_file.Write(data1, sizeof(data1));
               bytes_written ++;
```

```
        }}}}}

        fseek(ip_file, start_position, 0);
        next_packet = TS_header.continuity_counter;
    }

    if (seq_start_found == 1)
    {   if (TS_header.continuity_counter != next_packet)
        { msgresponse = MessageBox("Continuity error detected
        whilst extracting. Press OK to continue or Cancel to abort
        the extraction.",app_title, MB_OKCANCEL |
        MB_ICONEXCLAMATION );
          if (msgresponse == 2)
          { m_strMainStatus.Format("ES extraction aborted.");
            UpdateData(FALSE);
            OnPacketFirst();
            return;
          }
          else
          { next_packet = TS_header.continuity_counter; }
        }
        fseek(ip_file, -184, 1);
        TRACE("File position before data extraction: d%d",
        ftell(ip_file));
        adaption_length = 0;
        PES_header_length = 0;
        skip_length = 0;

        if (TS_header.adaption_field_control == 3)
        {
        adaption_length = fgetc(ip_file);
        skip_length = adaption_length + 1;
        fseek(ip_file, adaption_length, 1);
        }

        if (TS_header.payload_start_indicator == 1)
        {
        fseek(ip_file, 8, 1);
        PES_header_length = fgetc(ip_file);
        skip_length = skip_length + PES_header_length + 9;
        fseek(ip_file, PES_header_length, 1);
        }

        if (skip_length != 0)
        {
        data_length = (184 - skip_length);
        for (loop = 0; loop < data_length; loop ++)
        {
        fread(&data1, 1, 1, ip_file);
        op_es_file.Write(data1, sizeof(data1));
        bytes_written ++;
        }}
        else
        {
        fread(&data, sizeof(data), 1, ip_file);
```

```
                    op_es_file.Write(data, sizeof(data));
                    bytes_written = bytes_written + 184;
                    }
               }
               next_packet = ++ next_packet%16;
          }
     }
     if (bytes_written != 0)
     {
     MessageBox(”MPEG-2 video stream extraction complete.”,app_title,MB_OK |
     MB_ICONINFORMATION );
     }
     else
     {
     MessageBox(”MPEG-2 video stream extraction failed.”,app_title,MB_OK |
     MB_ICONSTOP );
     }
     void CMPEG2analyseDlg::OnFileExportElementarystream()
     op_es_file.Close();
     OnPacketFirst();
}}}
```

Appendix C **Structure of the CD-ROM**

Documents

A PDF version of the project proposal, progress report and final report.

Acrobat Reader 4.05

Required to view the PDF files.

Extractions

MPEG-2 video

MPEG-2 extractions created using the MPEG2analyse software. These files can be played using Mpeg2ply.exe in the MPEG2player directory.

PES

PES extractions created using the MPEG2analyse software.

Text

Text files of TS header data created using the MPEG2analyse software.

MPEG2player/MPEG2ply.exe

A freeware MPEG-2 video player, not written by me.

Software

Extract1Mb/Extract 1Mb.exe

a DOS executable, input and output filepaths will be asked for.

MPEG2analyse/MPEG2analyse.exe

the main executable which may be run from Windows explorer and associated help files.

Install/Setup.exe

an InstallShield wizard for automatic installation the Windows software.

PESdecode/PESdecode.exe

a DOS executable which should be run in a DOS box, command line switches are used to specify the input file, eg:

```
PESDECODE -i<input filepath>
```

TSdecode/TSdecode.exe

a DOS executable which should be run in a DOS box, command line switches are used to specify the input file and search PID, eg:

```
TSDECODE -i<input filepath> -p<search PID>
```

SourceCode

Extract1Mb

all source code and Microsoft Visual C++ project files.

MPEG2analyse

all source code and Microsoft Visual C++ project files.

MPEG2analyse/hlp

all source files and project files for the online help system.

PESdecode

all source code and Microsoft Visual C++ project files.

TSdecode

all source code and Microsoft Visual C++ project files.

TestStreams

The files in this directory are for BBC test purposes only. Reproduction, Broadcast or non-BBC related use is strictly prohibited. The copyright of the contents of these files remains with the BBC.

A selection of transport stream files of varying length, and an AFD test sequence which includes all of the AFD values in order (8 through 15).

Website

A copy of the website as it was on 11th May 2001. The index file lists all the application development versions with dates and changes made.

MPEG-2 analyser by Peter Daniel.lnk

A link to the website.