MULTIPLEXING OF AVS CHINA VIDEO WITH AAC AUDIO BIT STREAMS & DE-MULTIPLEXING WITH LIP SYNCHRONIZATION DURING PLAYBACK

by

SWAMINATHAN SRIDHAR

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ELECTRICAL ENGINEERING

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2010

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my professor Dr. K. R. Rao for his constant support and encouragement throughout the course of my research work. I have always admired his teaching methodologies and he has been a great source of inspiration in completing my thesis. I thank him for introducing me to the world of multimedia processing. It is a great honor for me to have worked under him.

I would like to thank Dr. A. Davis and Dr. J. Bredow for taking time of their busy schedule in being a part of my thesis committee.

I would like to thank my friends Dhivya, Anirudh, Laxmi Narasimhan and Rajesh for their comments and suggestions at various stages of my research work. I would also like to thank Rajesh for instilling in me a strong interest towards programming.

Finally, I am grateful to my dad K. Sridhar, my mother Geeta Sridhar and my sister Archana Sridhar for their continuous support and motivations in all my endeavors.

November 19, 2010

ABSTRACT

MULTIPLEXING OF AVS CHINA VIDEO WITH AAC AUDIO BIT STREAMS & DE-MULTIPLEXING WITH LIP SYNCHRONIZATION DURING PLAYBACK

Swaminathan Sridhar, MS

The University of Texas at Arlington, 2010

Supervising Professor: K. R. Rao

AVS China is the latest digital video coding standard released by the AVS working group of China which has been proven superior to the other video coding standards such as H.261, MPEG-1 and MPEG-2 in terms of coding efficiency and reduced complexity **[1]**. AVS standard employs the latest video coding tools which mainly target standard definition (SD) and high definition (HD) video compression and aim to achieve a similar coding efficiency of H.264/AVC but with reduced complexity. AVS video standard was developed in order to target broadcast and storage media applications such as digital video television, digital video disk (DVD and high definition disk) and broadband network multimedia applications such as video conferencing, video on demand, IPTV etc. In order to have a meaningful delivery of multimedia content to the end user, it is necessary to associate an audio stream to the video stream. Among the various audio compression schemes, MPEG-2/4 advanced audio coding (AAC) is the state-of-the art audio coding algorithm standardized by the ISO/IEC MPEG (moving pictures expert group) committee **[16]**. The audio quality of the AAC bit stream is observed to be superior than the earlier audio coding standards such as MPEG -1/2 layer 3 and AC3 which were widely used for delivering the audio content at very low bit rates.

Considering AVS as video and AAC as audio for transmission of digital multimedia content over a broadcast network provides the end users to take advantages of such leading technologies. In order for a proper transmission of multimedia content, the video and audio streams cannot be transmitted separately. These streams need to be multiplexed before transmission. The objective of the thesis is to propose an effective method for multiplexing the AVS video and AAC audio elementary streams and is followed by de-multiplexing the stream at the receiver end while achieving lip synchronization between the audio and video streams during playback. Since the video and audio streams have a frame wise arrangement, frame numbers are used as synchronization information during multiplexing which helps in achieving lip synchronization between the video and audio streams during playback. There are totally two layers of packetization adopted for multiplexing the video and audio streams before their transmission. The synchronization information is embedded in the headers of the first layer of packetization. The packetization layers adopted in the thesis conform to the MPEG-2 systems standard **[25]**, which meets the various requirements of the transmission channels. In order to prevent the buffer under or over flow at the receiver end, play back time has been chosen as reference criteria by which the audio and video data packets are allocated according to their play back time in the multiplexed stream. The advantages and limitations of the proposed method are discussed in detail.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

Figure Page

LIST OF TABLES

LIST OF ACRONYMS AND ABBREVIATIONS

AAC: Advanced audio coding

ADIF: Audio data inter change format

ADTS: Audio data transport stream

AFC: Adaptation field control

ATSC: Advanced television systems committee

AVC: Advanced video coding

AVS: Audio video standard

C2DVLC: Context based 2D variable length coding

CAVLC: Context based variable length coding

DVB: Digital video broadcasting

ES: Elementary stream

FAAC: Free advanced audio coder

GOP: Group of pictures

IDR: Instantaneous decoder refresh

IEC: International electrotechnical commission

ISO: International standards organization

ITU: International telecommunication union

MDCT: Modified discrete cosine transform

MPEG: Moving pictures expert group

NAL: Network abstraction layer

PES: Packetized elementary stream

PCM: Pulse code modulation

PID: Packet identifier

PSI: Program specific information

TNS: Temporal noise shaping

TS: Transport stream

VCL: Video coding layer

YUV: luminance and chrominance color component

CHAPTER 1

INTRODUCTION

1.1 Introduction

With the advent of HDTV and it substantially replacing the traditional television systems, there is an immense need of transmitting high quality multimedia content at significantly lower bit rates. But due to the tradeoff between the quality of the multimedia content and bit rates, selecting the right video and audio codec for compression plays an important role in the transmission of the multimedia program. In the recent decades, the development of several video and audio coding standards such as MPEG-2 **[17]**, MPEG-4 **[18]**, H.264 **[13]**, AC3 **[27]** and MP3 **[27]** have revolutionized the way of digital television transmission. AVS China is the latest video coding standard developed by the AVS work group of China employing the latest video coding tools for achieving high compression while preserving the quality of the video **[2]**. AVS China video standard has been proven superior to the existing video standards such as MPEG-2 and MPEG-4 in terms of reduced complexity and coding efficiency. In another words, this codec can provide high quality video at the same bandwidth or the same quality video at a reduced bandwidth. Due to these robust features, AVS China standard has been adopted for video compression and transmission in this thesis. Advanced audio coding (AAC) is a standardized lossy compression scheme for encoding the digital audio **[19]**. The AAC compression scheme has been standardized by ISO/IEC as part 7 of the MPEG-2 standard **[17]** and part 3 of the MPEG-4 standard **[18]**. This codec showed greater sound quality and transparency than MP3 for files coded at the same bit rates. The video and audio streams obtained from these compression schemes need to be multiplexed before they are transmitted over a broadcast channel. The multiplexing process aims at providing efficient transmission schemes such as time stamps for lip synchronization between the video and audio streams during playback, robust error codes to detect packet losses etc. In this thesis an effective algorithm is proposed for multiplexing the AVS China part 2 video with AAC audio elementary streams and de-multiplexing with lip synchronization during playback. Figure 1.1 shows the multiplexing method adopted in this thesis.

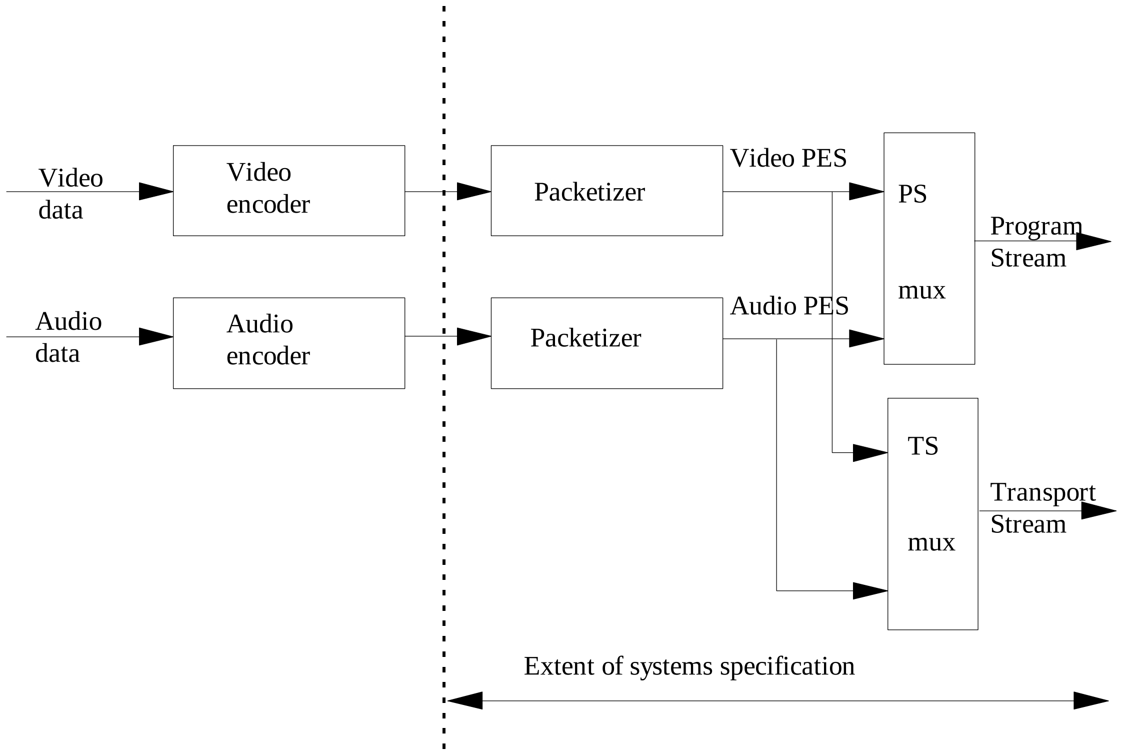


Figure 1.1 Audio-video multiplexing process **[25]**

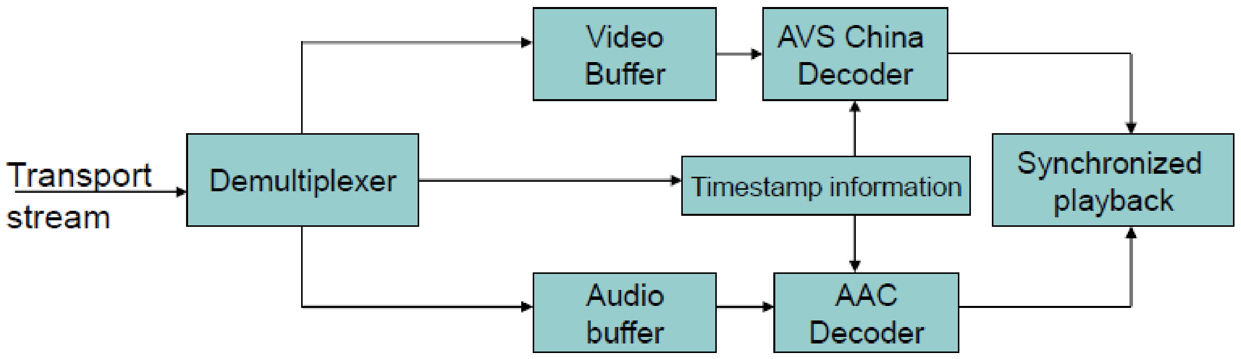


Figure 1.2 Audio-video de-multiplexing process **[15]**

1.2 Thesis outline

Chapters 2 and 3 give an overview of the AVS China video codec and AAC audio codec. The network abstraction layer (NAL) and audio data transport stream (ADTS) format used for transmission of video and audio data packets are discussed in detail.

Chapter 4 explains the process of generating the transport stream packets from the elementary video and audio streams and multiplexing them for effective transmission. The additional information embedded in the packet headers to assist the de-multiplexing process is also discussed in detail in this chapter.

Chapter 5 discusses about the de-multiplexing of transport stream packets to generate the elementary streams and it also discusses about the method adopted for achieving synchronization between video and audio frames.

Chapter 6 deals with the test conditions and results. It also includes conclusions and other suitable methods for improving this thesis.

CHAPTER 2

OVERVIEW OF AVS CHINA VIDEO CODEC

2.1 AVS China

AVS (Audio Video coding Standard) China is the latest digital video coding standard developed by the AVS Working Group of China on a need to reduce the royalty fees paid by the Chinese people for using other international video coding standards such as MPEG-2, MPEG-4 and MPEG-4 part 10 (H.264) **[1]**. The main characteristics of AVS China are that it is technically an advanced second generation source coding standard and it is totally formulated and controlled by China **[5]**. The AVS standard is a highly efficient video coder employing the latest video coding tools which primarily target standard definition (SD) and high definition (HD) video compression and aim to achieve similar coding efficiency as H.264/AVC but with lower computational complexity **[3]**. Since the video coding syntax structure of AVS China is very similar to that of the MPEG-2 video standard, it can be easily used in the present widely used MPEG-2 systems with a significant improvement in the coding efficiency **[2]**. AVS video coding standards are important parts of standardization productions of AVS working group. AVS-video is a conglomeration of all parts related to coding of video and its auxiliary information in the AVS **[4]**. The different AVS parts are listed in Table 2.1.

Table 2.1 Different parts of AVS standard **[5]**

Part	**Name**
1	System
2	Video
3	Audio
4	Conformance test
5	Reference Software
6	Digital media rights management
7	Mobile video

Table 2.1 – *Continued*

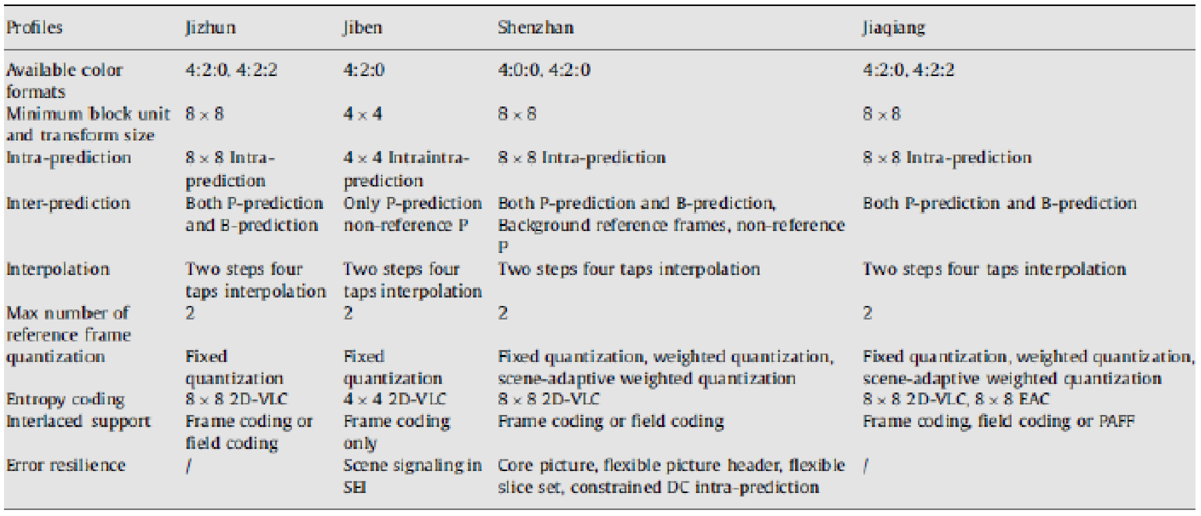
8	Transmit AVS via IP network
9	AVS file format
10	Mobile speech and audio coding

The second part of AVS China i.e. part2 (video) mainly targets high definition and high quality digital broadcast applications, digital storage media and other related applications while the AVS part 7 mainly targets the mobile multimedia applications. The AVS part2 video coder architecture is very similar to the H.264 standard but its complexity is reduced by choosing only 8x8 blocks, five intra modes and other features **[6]**.

2.2 AVS China profiles and levels

Considering the various requirements of video applications, AVS-video defines different profiles which combine advanced video coding tools with trade-off between coding efficiency and computational complexity and target to various applications. The basic profiles defined in the standard are shown in Table 2.2.

Table 2.2 AVS China profile features **[4]**



Profiles	Jizhun	Jiben	Shenzhan	Jiaqiang
Available color formats	4:2:0, 4:2:2	4:2:0	4:0:0, 4:2:0	4:2:0, 4:2:2
Minimum block unit and transform size	8 × 8	4 × 4	8 × 8	8 × 8
Intra-prediction	8 × 8 Intra-prediction	4 × 4 Intraintra-prediction	8 × 8 Intra-prediction	8 × 8 Intra-prediction
Inter-prediction	Both P-prediction and B-prediction	Only P-prediction non-reference P	Both P-prediction and B-prediction, Background reference frames, non-reference P	Both P-prediction and B-prediction
Interpolation	Two steps four taps interpolation	Two steps four taps interpolation	Two steps four taps interpolation	Two steps four taps interpolation
Max number of reference frame	2	2	2	2
quantization	Fixed quantization	Fixed quantization	Fixed quantization, weighted quantization, scene-adaptive weighted quantization	Fixed quantization, weighted quantization, scene-adaptive weighted quantization
Entropy coding	8 × 8 2D-VLC	4 × 4 2D-VLC	8 × 8 2D-VLC	8 × 8 2D-VLC, 8 × 8 EAC
Interlaced support	Frame coding or field coding	Frame coding only	Frame coding or field coding	Frame coding, field coding or PAFF
Error resilience	/	Scene signaling in SEI	Core picture, flexible picture header, flexible slice set, constrained DC intra-prediction	/

The four different profiles defined in AVS-video are

- Jizhun (base) profile
- Jiben (basic) profile

- Shenzhan (extended) profile
- Jiaqiang (enhanced) profile

2.2.1 AVS-video Jizhun (base) profile

Jizhun profile is considered to be the baseline profile with moderate computational complexity which is defined in AVS-part2 video and is targeted mainly at digital video applications like storage media and commercial broadcasting.

2.2.2 AVS-video Jiben (basic) profile

Jiben profile is a basic profile which is defined in AVS-part7 and is targeted mainly at mobile video applications.

2.2.3 AVS-video Shenzhan (extended) profile

Shenzhan profile is an extended profile defined in AVS-part2 and it focuses exclusively on solutions of standardizing the video surveillance applications.

2.2.4 AVS-video Jiaqiang (enhanced) profile

Jiaqiang profile is defined in AVS-part2 and is mainly targeted at movie compression for high-density storage.

A brief overview of the various profiles defined in AVS-video and their applications is shown in Table 2.3.

Table 2.3 AVS-video profile applications **[4]**

Profiles	**Key applications**
Jizhun profile	Television broadcasting, HDTV
Jiben profile	Mobile applications
Shenzhan profile	Video surveillance
Jiaqiang profile	Multimedia entertainment

A profile is a subset of syntax, semantics and algorithms defined by AVS video standard whereas a level puts constraints on the parameters of the stream **[7]**. There are four levels defined in AVS video standard.

- Levels 4.0 and 4.2 for standard definition (SD) video with 4:2:0 and 4:2:2 formats **[2]**
- Levels 6.0 and 6.2 for high definition (HD) video with 4:2:0 and 4:2:2 formats **[2]**

The maximum picture size defined for AVS video varies from 720x576 to 1920x1080 pixels and the maximum bit rate varies from 10 Megabits/s to 30 Megabits/s.

2.3 Data formats used in AVS

AVS codes video data in progressive scan format where in all lines of each frame are scanned in sequence while interlaced scanning involves alternate scanning of odd and even fields in a frame. A significant advantage of using the progressive scan format is the efficiency with which motion estimation operates. Progressively scanned frames can be encoded at significantly lower bitrates than the interlaced coded frames with the same perceptual quality and also motion compensated coding of progressive format data is significantly less complex than coding of interlaced data which accounts to a significant component of the reduced complexity in AVS coding **[2]**. Nevertheless AVS also provides coding tools for interlaced scan format.

2.3.1 AVS video layered structure

AVS is built on a layered structure representing the video data as shown in figure 2.1

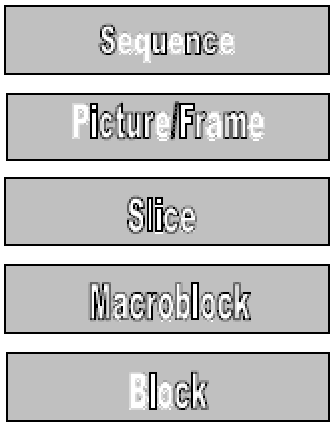


Figure 2.1 AVS layered data structure **[2]**

2.3.1.1 Sequence

The sequence layer consists of sets of frames of continuous video. It provides an entry point into the coded video. It contains a set of mandatory and optional system parameters. Mandatory system parameters are necessary to initialize the decoder system while optional system parameters are used for other system settings at the discretion of the network provider. Optional user data can also be sent in the sequence header **[2]**. An example of video sequence is shown in figure 2.2.

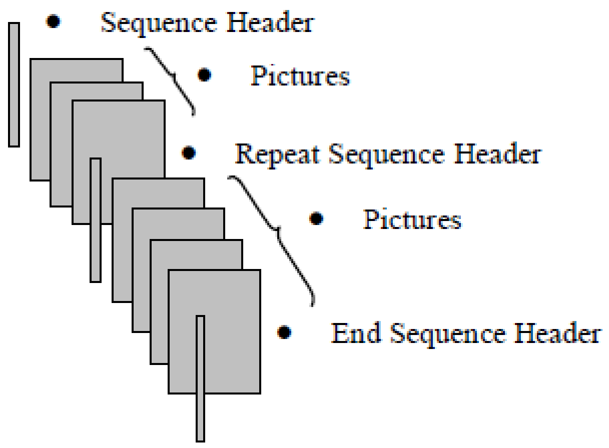


Figure 2.2 Video sequence **[2]**

2.3.1.2 Picture

The picture layer provides the coded information of video data and it also contains a header with mandatory and optional parameters and optionally with a user data. The three types of pictures that are defined in AVS are

- Intra pictures (I- pictures)
- Forward inter decoded pictures (P- pictures)
- Bidirectional inter decoded pictures (B- pictures)

There are three different orders of pictures specified in a video standard **[9]**.

- Decoding order signifies the order in which pictures are decoded from a bit stream.
- Display order signifies the order in which the pictures are displayed.
- Reference order signifies the order in which reference pictures are arranged for inter prediction of other pictures.

In the AVS-video standard, if the bit stream does not contain a coded B-picture the decoding order of the pictures is the same as the display order, but when B-pictures are present the decoding order is different from the display order and the decoded pictures should be re-ordered according to the following rules **[8]**.

- If the current decoded picture is a B-picture the current decoded picture is outputted for display directly.

- If the current decoded picture is an I or P picture, the previous decoded I or P picture is transmitted for display or if it existed, otherwise no picture is transmitted.
- When all the pictures have been decoded if there are still decoded pictures in the buffer, they are transmitted to the display.

A pictorial representation of the picture handling process in AVS video standard is shown in figure 2.3.

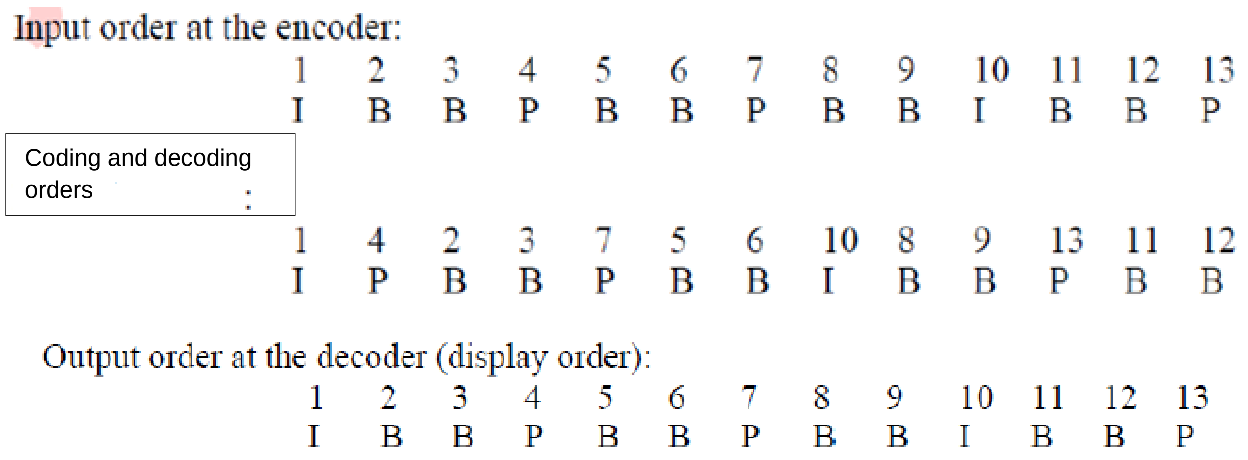

Input order at the encoder:
1 2 3 4 5 6 7 8 9 10 11 12 13
I B B P B B P B B I B B P
Coding and decoding orders :
1 4 2 3 7 5 6 10 8 9 13 11 12
I P B B P B B I B B P B B
Output order at the decoder (display order):
1 2 3 4 5 6 7 8 9 10 11 12 13
I B B P B B P B B I B B P

Figure 2.3 Picture handling in AVS video standard **[8]**.

A group of pictures (GOP) specifies the order in which the intra and inter-frames are arranged. It can contain the following picture types.

- frame (intra coded frame): It is a reference picture which is independent of other picture types. Each GOP begins with an I frame.
- P- frame (predictive coded frame): It contains the motion compensated difference information from the preceding either I or P frame.
- B- frame (bi-directionally predictive coded picture): It contains the motion compensated difference information from the preceding and following I or P frame.

The frame re-ordering in figure 2.3 can be explained as follows.

For example, there are two B- pictures between successive I and P pictures and also two B- pictures between successive P- pictures. Picture '1I' is used by picture '4P' for prediction. Pictures '4P' and ‘1I’ are both used by pictures ‘2B’ and ‘3B’ for predictions. Therefore the decoding order of pictures is ‘1I’, ‘4P’, ‘2B’, ‘3B’ and the display order is ‘1I’, ‘2B’, ‘3B’, ‘4P’.

2.3.1.3 Slice

The slice structure provides the lowest layer mechanism for resynchronization in case of transmission bit errors. It comprises a variable number of macro blocks.

2.3.1.4 Macro block

A macro block represents a 16x16 region of a picture which includes the luminance (Y) and chrominance component (Cb & Cr) pixels. There are a total of three sampling patterns for Y, Cb and Cr of which two are supported in the AVS video standard. These are the 4:2:0 and the 4:2:2 format **[7]**. The number indicates the relative sampling rate of each component in horizontal direction. The 4:2:2 format indicates that for every 4 luminance samples in the horizontal direction there are 2 Cb and 2 Cr samples where as in the 4:2:0 sampling format Cb and Cr each have half the horizontal and vertical resolution of Y. A pictorial representation of the macro block formats are shown in figures 2.4 and 2.5.

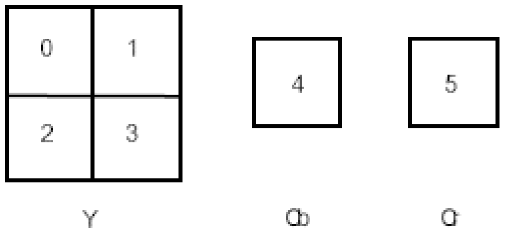


Figure 2.4 4:2:0 macro block format **[2]**.

In the 4:2:0 sampling format shown in figure 2.4, each chrominance component i.e. the Cr (8x8) and Cb (8x8) component have half the horizontal and vertical resolution of Y (16x16).

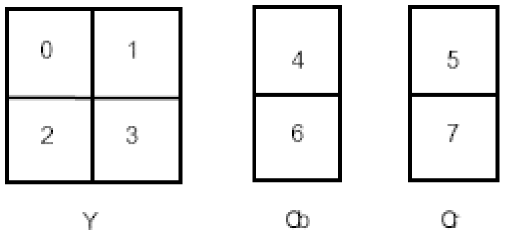


Figure 2.5 4:2:2 macro block format **[2]**.

In the 4:2:2 sampling format shown in figure 2.5, it indicates that for every 4 luminance samples in the horizontal direction there are 2 Cr and 2 Cb samples.

2.3.1.5 Block

A block is the smallest coded unit and it contains the transform coefficient data of the prediction errors.

2.4 AVS video encoder

The basic encoding procedure of AVS video is shown in figure 2.6

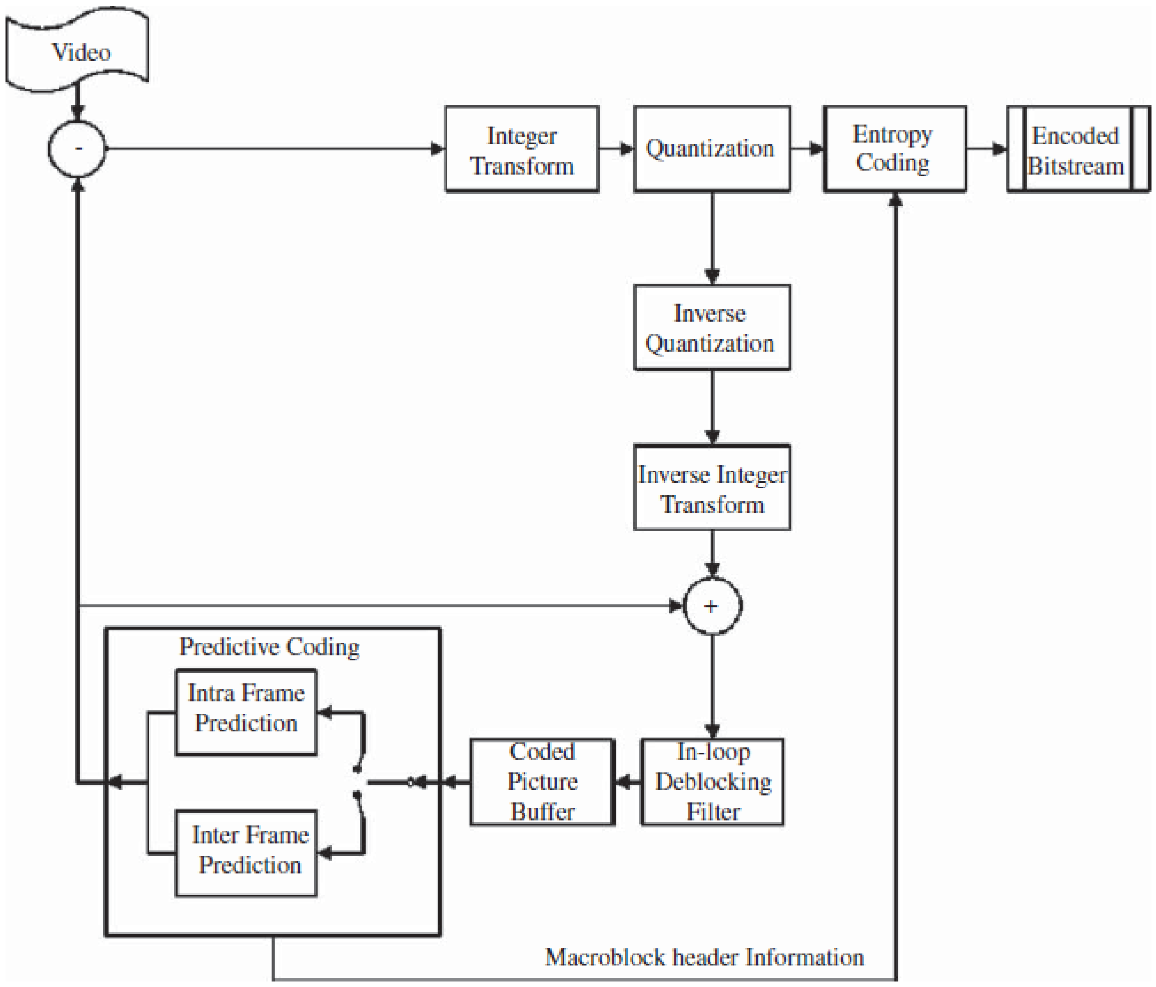


Figure 2.6 Block diagram of AVS video encoder **[4]**.

Similar to the previous video coding standards such as MPEG-1, MPEG-2 and H.264 **[9]**, AVS part 2 also employs hybrid block based video compression techniques such as spatial and temporal predictions, transform, quantization, entropy coding and de-blocking filter to acquire better trade-off between coding efficiency and complexity **[3]** . Temporal redundancy is removed by motion-compensated

DPCM coding, residual spatial redundancy is first removed by spatial prediction and finally by transform coding and statistical redundancy is removed by entropy coding.

2.4.1 Encoder process outline

A video consists of a sequence of frames (YUV) and each frame is split into several rectangular blocks known as macro blocks which contain a fixed size of 16x16 luminance components and their corresponding chrominance components. Predictive type coding is performed on each of the macro blocks that can be classified into either as intra-frame coding or inter-frame coding. Then transform is performed on the macro blocks corresponding to the prediction residuals, which are the differences between original pixel values of the current image and the predicted pixel values. The transform coefficients are further quantized and scanned (zigzag) (Fig. 2.9) before entropy coding and finally the entropy coded information is converted into a bit stream **[4]**.

2.4.2 Coding tools used in AVS video coder

The functions of various coding tools used in an AVS video coder are described below:

2.4.2.1 Transform

Unlike H.264 and MPEG-2, AVS uses a separable, integer precision 8x8 discrete cosine transform. The transform is designed in conjunction with the quantization to minimize the decoder implementation complexity which is known as the pre-scaled integer transform (PIT) **[10]**. Figures 2.7 and 2.8 show the block diagram of conventional ICT and PIT scheme used in the H.264 and AVS respectively.

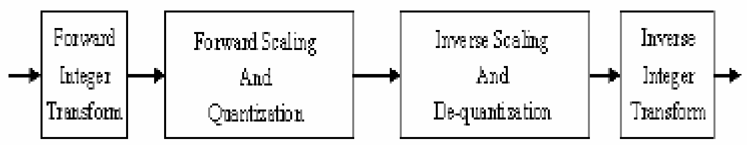

Forward Integer Transform
Forward Scaling And Quantization
Inverse Scaling And De-quantization
Inverse Integer Transform

Figure 2.7 Block diagram of conventional ICT scheme used in H.264 **[10]**

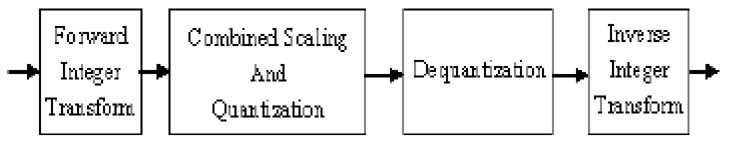
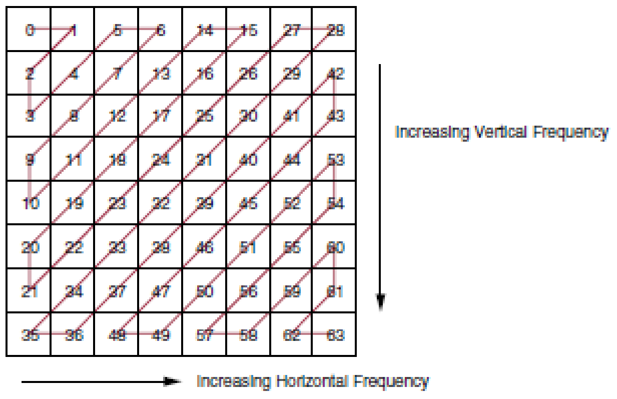

Forward Integer Transform
Combined Scaling And Quantization
Dequantization
Inverse Integer Transform

Figure 2.8 Block diagram of a PIT scheme used in the AVS-video **[10]**

2.4.2.2 Quantization and scan

The 2D coefficients generated by the transform coding are converted into a 1D sequence for quantization and coding using a zigzag scan (Fig. 2.9) for progressive data and an alternate scan for interlaced data. Figure 2.9 shows the zigzag scan used for progressive data.



Increasing Vertical Frequency
Increasing Horizontal Frequency

Figure 2.9 Zigzag scan used in AVS for scanning progressive data **[2]**

Quantization of the transform coefficients in AVS is adopted with an adaptive uniform quantizer. AVS-video does not provide a direct option to vary the bit rates for which instead the step size of the quantizer can be varied to provide rate control which is useful in constant bit rate operations to prevent buffer overflow. The quantization parameter can either be fixed for an entire picture or slice or it can be updated differentially at every macro block.

2.4.2.3 Entropy coding

Entropy coding plays an important role in video compression in reducing the statistical correlation among the DCT coefficients. The AVS standard employs a context based 2D-VLC (C2DVLC) entropy coder. C2DVLC uses multiple 2D-VLC tables with simple exponential Golomb codes employing context based adaptive multiple table coding to exploit the statistical correlation among the DCT coefficients of each block which arises because as DCT sub-band frequency increases the magnitude of non-zero coefficient gets smaller and the run-length of successive zero coefficients becomes longer. Exp-golomb codes are used to code the run lengths of zero and non-zero coefficients for higher coding efficiency. In terms of coding efficiency, when compared with context based adaptive variable length coding (CAVLC) used in H.264/ AVC, C2DVLC has a similar coding efficiency with less computational complexity **[11]**.

2.4.2.4 De-blocking filter

A de-blocking filter is applied to macro blocks after motion compensation and residual coding to improve the visual quality and prediction performance by smoothing the edges around the macro blocks. An In-loop de-blocking filter is applied to reduce blocking artifacts and thereby to improve the visual quality. Several de-blocking techniques are defined in AVS-video that vary according to the filter strength which is determined by the coding modes of adjacent blocks, quantization step size and the steepness of the luminance gradient between blocks **[4]**.

- Default de-blocking filter is performed on the 8x8 boundaries.
- Simplified de-blocking filter is performed on 4x4 boundaries.
- Pixel level in-loop de-blocking filter decides the boundary strength at the pixel level instead of the boundary level.

2.4.2.5 Mode decision

A mode decision unit selects the best motion compensation mode for picture and macro blocks. Rate distortion optimization is used to increase the efficiency of a mode decision.

2.4.2.6 Intra prediction

Intra prediction is a prediction technique to exploit the spatial dependencies in the current frame. It uses the decoded information in the current frame as the reference for prediction to eliminate spatial correlation among pixels of the same frame. The technique of 8x8 intra prediction is used in AVS part2 which allows five prediction modes namely the DC (mode 2), horizontal (mode 1), vertical (mode 0), down left (mode 3) and down right (mode 4) for the luminance component and four prediction modes namely the DC, horizontal, vertical and plane for the chrominance component. Each of the four 8x8 luminance blocks can be predicted by choosing the best of the five different prediction modes. Before using the DC, diagonal down left and diagonal down right mode a three-tap low pass filter (1/4, 2/4, 1/4) is applied on the samples that will be used as a reference for prediction **[4, 3]**.

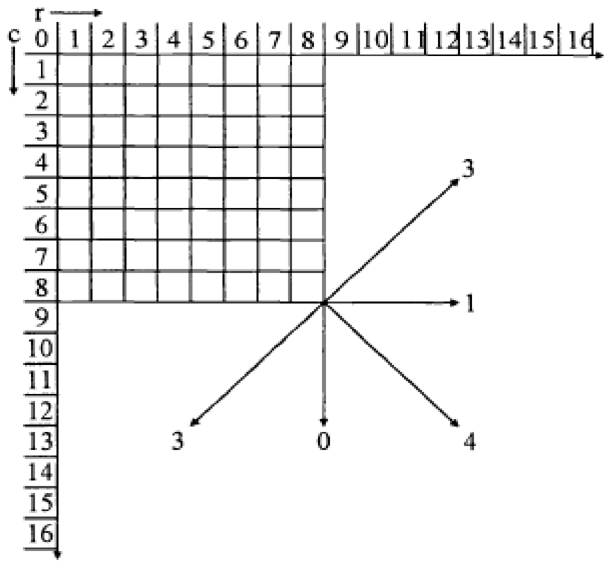


Figure 2.10 Directions and reference pixels used for 8x8 intra prediction of luminance component **[1]**

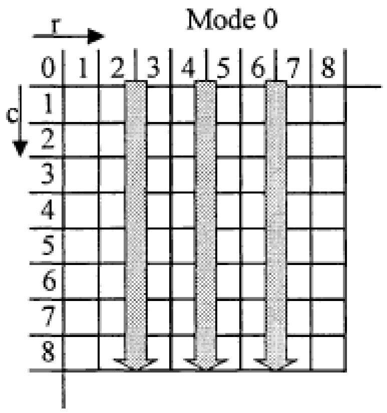


Figure 2.11 vertical mode (mode 0) **[1]**

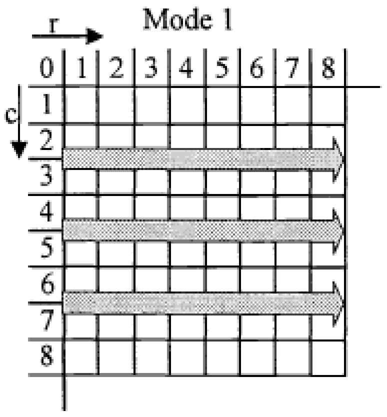


Figure 2.12 horizontal mode (mode 1) **[1]**

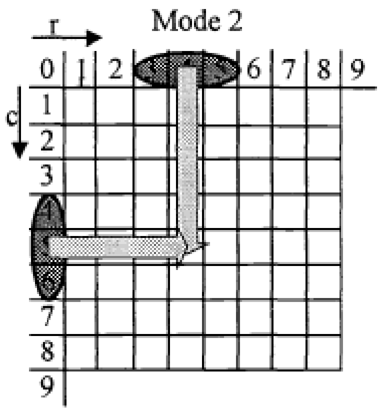


Figure 2.13 DC mode (mode 2) **[1]**

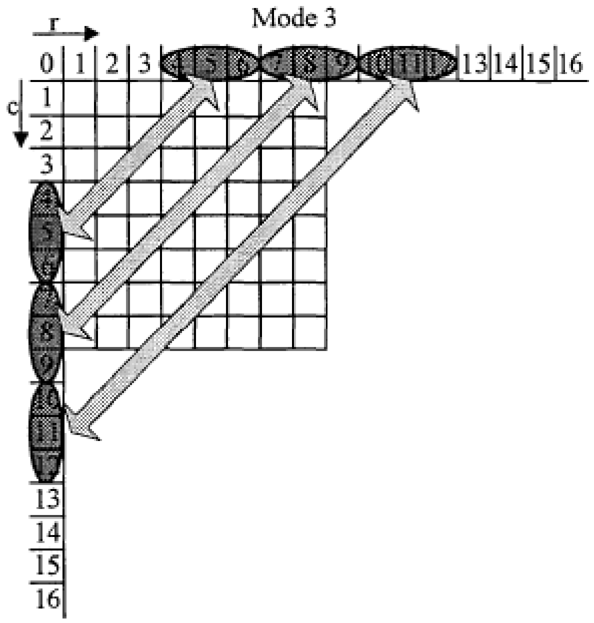


Figure 2.14 Diagonal down left mode (mode 3) **[1]**

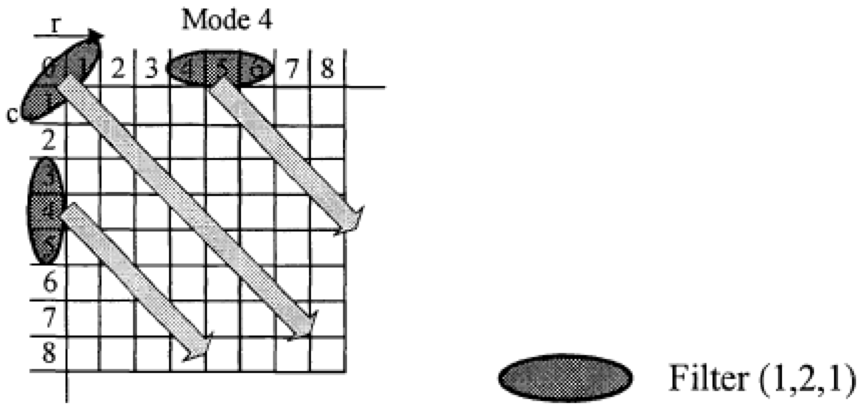


Figure 2.15 Diagonal down right mode (mode 4) **[1]**

- In the vertical mode (mode 0) the upper samples are extrapolated vertically.
- In the horizontal mode (mode 1) left samples are extrapolated horizontally.
- In the DC mode (mode 2) all the samples are predicted by the means of their corresponding left and top samples.
- In the diagonal down left mode (mode 3), the samples are interpolated at an 45° angle between lower-left and upper-right.
- In the diagonal down right mode (mode 4), the samples are extrapolated at an 45° angle down to the right.

2.4.2.7 Inter-prediction

Inter-prediction is another type of prediction technique which exploits temporal correlation among various frames. Inter-prediction involves two important concepts known as motion estimation (ME) and motion compensation (MC). An inter-coded frame is first divided into macro blocks. Instead of directly encoding the raw pixel values of each the encoder will search for a similar block in the previously encoded frame known as the reference frame. If the encoder successfully finds the matching block the current macro block is encoded using a motion vector which points to the location of the matched macro block in the reference frame. This process of determining the motion vectors is known as motion estimation. The chosen macro block region becomes the prediction for the current macro block and is subtracted from the current macro block to form a residual block. This process of forming a residual block is known as motion compensation. The residual block is encoded and transmitted along with the differential motion vectors. The decoder receives the motion vectors and uses them to generate the predictor region which is further added to the residual block to reconstruct the original macro block.

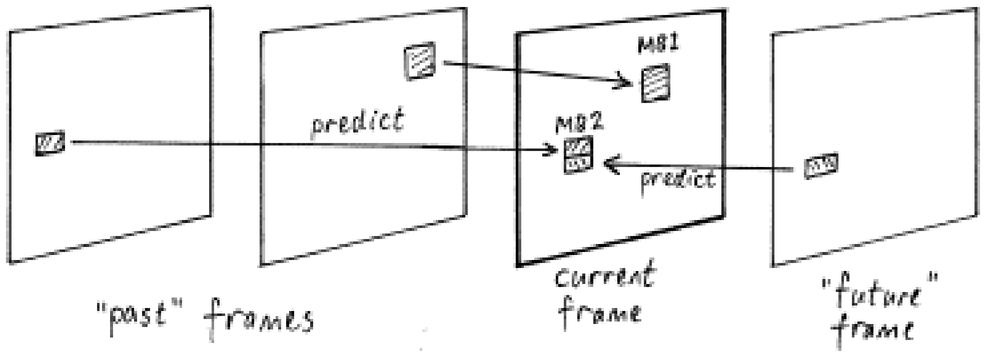


Figure 2.16 Inter prediction **[38]**

AVS part2 supports variable block sizes for motion compensation in inter frame prediction with block sizes from 16x16 to 8x8 to better exploit the temporal dependencies. The various block sizes used in AVS part2 for motion compensation are shown in figure 2.17.

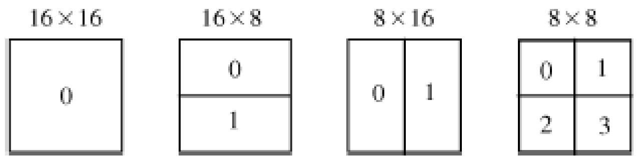


Figure 2.17 Block sizes supported for inter frame prediction in AVS part2 **[3]**

AVS part2 supports quarter (¼) sample accurate motion compensation. AVS part2 uses a ¼ pixel accuracy 2-D separable interpolation method named two steps four taps interpolation (TSFT) **[3]**. The fractional samples used for interpolation in the case where an object has moved by a non-integer number of pixels between frames are shown in figure 2.18.

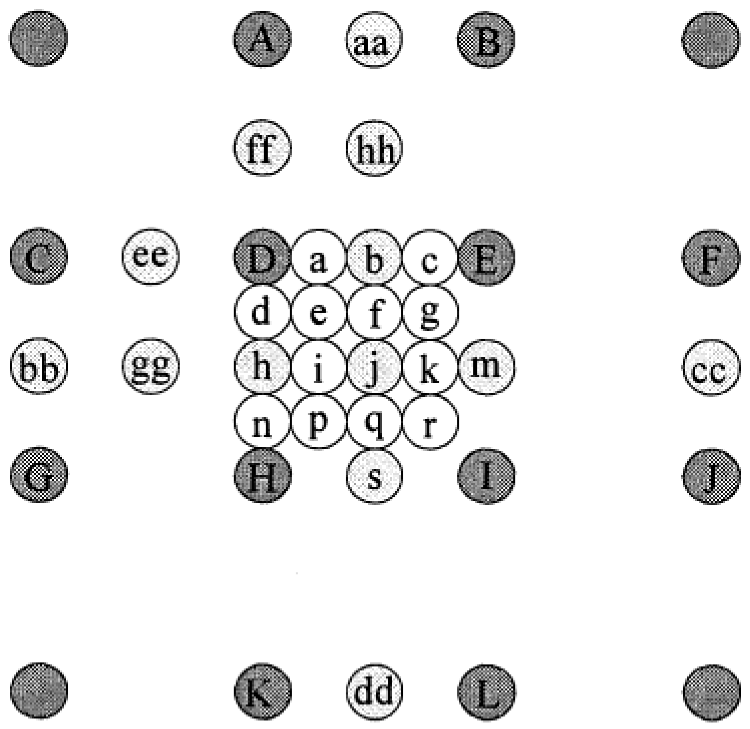


Figure 2.18 Fractional samples with ¼ pixel accuracy used in AVS part2 **[1]**

The sample values at half pixel locations *b, h* are interpolated by applying a 4-tap cubic convolution interpolation filter with tap values ($\frac{-1}{8}, \frac{5}{8}, \frac{5}{8}, \frac{-1}{8}$) on the values at integer pixel positions and *j* is obtained by interpolating the filter on the half-pixel locations. The sample values at quarter pixel locations *a, c, d, f, I, k, n,* and *q* are interpolated by a 4-tap cubic spine filter with tap values ($\frac{1}{16}, \frac{7}{16}, \frac{7}{16}, \frac{1}{16}$) on the samples at integer and half-pixel locations. The sample values at quarter pixel locations *e, g, p* and *r* interpolated by applying a bi-linear filter on the values at half-pixel location *j* and at integer pixel locations namely *D, E, H* and *I* **[3]**.

AVS part2 also supports multiple reference picture motion compensation for P and B frames but the maximum number of reference frames that can be used by either P or B frame is restricted to two to reduce the storage requirement and computational complexity. P frames can use at most of two previous pictures for inter prediction while for B frames no more than one forward picture and one backward picture is allowed for inter prediction **[3]**.

2.5 AVS video decoder

The block diagram of the AVS video decoder is shown in the figure 2.19.

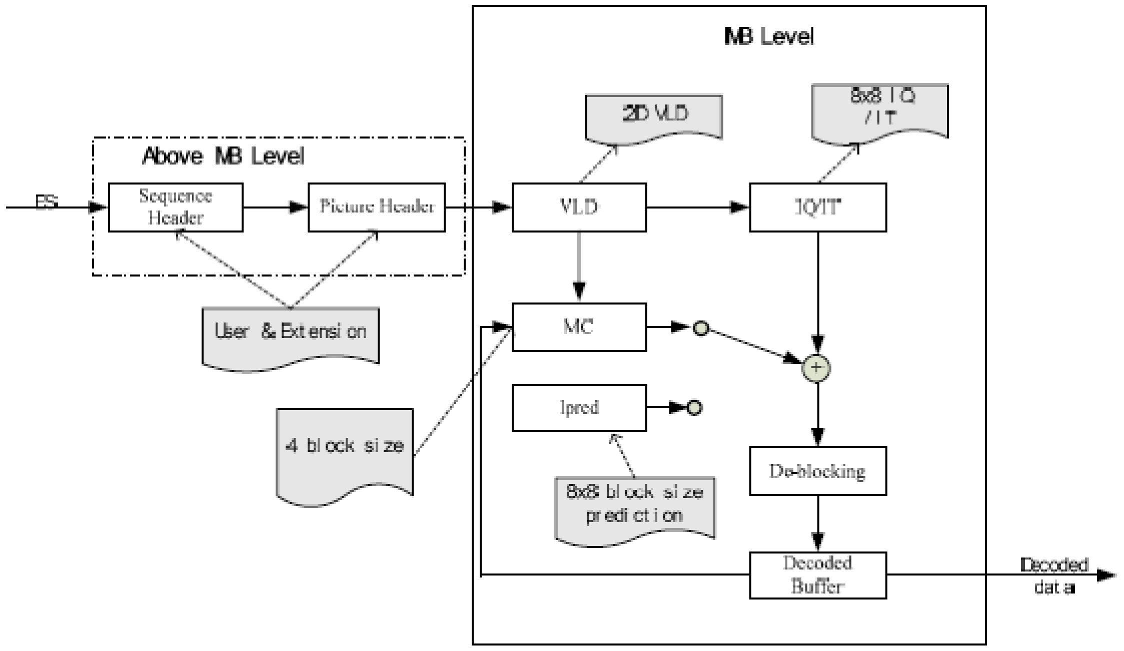


Figure 2.19 AVS video decoder **[12]**

The AVS decoder takes in the compressed video elementary stream from the storage or transmission media as its input and stores it in a rate buffer from which the data is read out at a rate demanded by the decoding of each macro block and picture. This is followed by a bit stream parser which separates the quantization parameter, motion vectors and other side information from the coded data. The data is then passed through the VLD entropy decoder which extracts the header information and the slice data along with the motion vectors. The signal is then decoded by the inverse quantizer and inverse DCT to reconstruct the prediction error or the coded data. The motion vectors are decoded by the motion compensation unit to generate the prediction of the current picture which is further added to the prediction error to generate the output signal.

2.6 AVS video bit stream

Some of the syntax and semantics used in the AVS video bit stream are explained as follows.

2.6.1 Start code

Start code is a special bit pattern that is used in the AVS video stream. Each start code consists of a start code prefix and a start code value. The start code prefix is a string of 23 zero bits followed by a single bit with a value of ‘1’ i.e. the bit string ‘0000 0000 0000 0000 0000 0001’ or ‘0x000001’ and all the start codes are byte aligned.

2.6.2 Start code value

Each start code prefix is followed by a start code value which is an 8-bit integer that identifies the start code type. Table 2.4 describes the various start code types and their values used in an AVS video bit stream.

Table 2.4 Start code types and start code values used in AVS **[8]**

Start code type	**Start code value (Hexadecimal)**
slice_start_code (I, P or B)	00 ~ AF
video_sequence_start_code	B0
video_sequence_end_code	B1
user_data_start_code	B2
i_picture_start_code	B3
Reserved	B4
extension_start_code	B5
pb_picture_start_code	B6
video_edit_code	B7
Reserved	B8
system start code	B9 ~ FF

2.6.2.1 video_edit_code

The bit string used is ‘0x000001B7’. This syntax means that there may be missing reference pictures for the successive P or B picture that immediately follows an I-picture and this P or B picture cannot be coded directly.

2.6.2.2 video_sequence_end_code

The bit string is ‘0x000001B1’. This syntax indicates the end of video sequence.

2.6.2.3 video_sequence_start_code

The bit string is ‘0x000001B0’. This syntax identifies the start of a video sequence.

2.6.2.4 extension_start_code

The bit string is ‘0x000001B5’. This syntax identifies the beginning of extensions.

2.6.2.5 user_data_start_code

The bit string is ‘0x000001B2’. This syntax identifies the beginning of user data. The user data is continuously stored until next start code.

2.6.2.6 i_picture_start_code

The bit string is ‘0x000001B3’. This syntax indicates the beginning of I frame.

2.6.2.7 pb_picture_start_code

The bit string is ‘0x000001B6’. This syntax is the start code of P or B pictures.

2.6.2.8 slice_start_code

The bit string contains a total of 32 bits just like the other start codes. The first 24 bits have the value of ‘0x000001’ and the last 8 bits are the slice_vertical_position ranging from 0x00 to 0xAF. The slice_vertical_position gives the vertical position of the first macro block in the slice in macro block units.

2.6.3 Picture_coding_type

Another important syntax used in AVS video bit stream is picture_coding_type. This syntax is a 2-bit unsigned which specifies the coding type of a picture. The syntax structure is shown in table 2.5

Table 2.5 Coding type of P or B picture **[8]**

picture_coding_type	Coding type
00	Forbidden
01	Forward inter prediction (P)
10	Bidirectional inter prediction (B)
11	Reserved

2.7 NAL unit for AVS video stream

NAL unit stands for network abstraction layer unit. It is a type of packetization layer that prefixes certain headers to encoded video bit stream. NAL unit was primarily designed to provide a network friendly environment for the transmission of video data. It mainly addresses the video related applications such as video telephony, video storage, broadcast and streaming applications, IPTV etc. The syntax for the NAL unit is defined in the H.264/AVC standard **[13]** but AVS part2 standard does not define any syntax format for the NAL unit. The encoded bit stream from the AVS encoder is a raw format bit stream and hence it needs to be converted to NAL unit before sending it over a network.

2.7.1 NAL unit mapping with AVS video stream

The basic syntax defined for a NAL unit is shown in figure 2.20

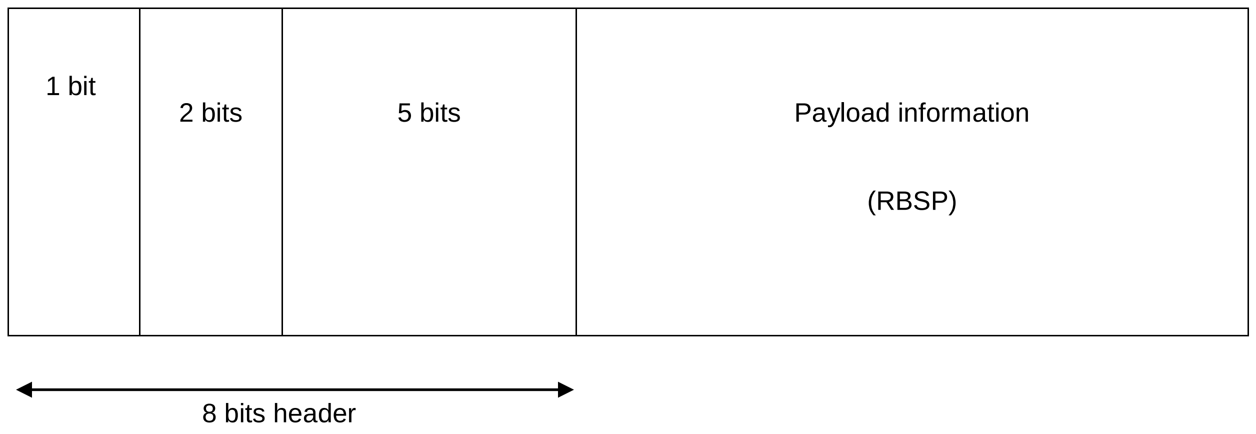


Figure 2.20 NAL unit syntax **[13]**

The procedure for mapping an AVS video stream with a NAL unit is to map the data between every start code prefix ‘0x000001’ in the AVS part2 video bit stream into a NAL unit (which includes the start code value but not code prefixes)and then add a one-byte NAL unit header before the start code value.

2.7.2 NAL unit header description

The NAL unit header description is shown in figure 2.21

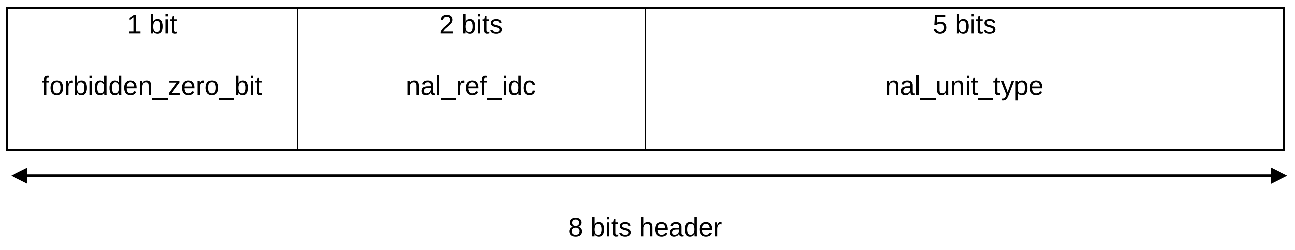


Figure 2.21 NAL unit header description **[14]**

According to the procedure defined in section 2.7.1 the video stream is mapped into NAL unit stream. The data between every two consecutive NAL headers is considered as RBSP byte. The syntax and semantics used in NAL unit header are described as follows

2.7.2.1 forbidden_zero_bit or Forbidden bit

Its value should always be ‘0’.

2.7.2.2 nal_ref_idc or NAL reference ID

It is a 2-bit unsigned integer. A non-zero value indicates data contained in this NAL unit is sequence header or reference frame data where as a zero indicates data contained in this NAL unit is not a reference frame data. Nal_ref_idc for I frames should not be zero. Nal_ref_idc indicates the priority of the type of data carried in the NAL unit based upon the start code type with a maximum 2-bit priority value being ‘11’ and the least being ‘00’. Table 2.7 shows the nal_ref_idc values assigned in the NAL unit header based on the start code type.

Table 2.6 Nal_ref_idc (priority) values based upon the start code values **[14]**

Start code type	**Nal_ref_idc value (2-bits)**
Sequence header	11
Picture header of I frame	11
Slice data of I frame	11
Picture header of P frame	10
Slice data of P frame	10
Picture header of B frame	01
Slice data of B frame	01
Video extension start code	00
User data start code	00
Video edit start code	00

2.7.2.3 nal_unit_type or NAL unit type

It is a 5-bit unsigned integer and therefore 32 types of NAL units are allowed. Nal_unit_type defines the type of RBSP (payload) data structure in a NAL unit according to the start code value followed and (or) information contained in the header. Table 2.7 shows the various NAL unit types according to their start code values.

Table 2.7 NAL unit type according to their start code values and (or) picture coding type **[14]**

nal_unit_type	**NAL Type**	**Stuffing reason**
0	Reserved	
1	sequence header	Start code value is B0
2	video extension	Start code value is B5
3	user data	Start code value is B2
4	video edit	Start code value is B7
5	I frame picture header	Start code value is B3
6	P frame picture header	Start code value is B6，and the encoding mode in the picture header is 01
7	B frame pricture header	Start code value is B6，and the encoding mode in the picture header is 10
8	I frame slice	Start code value is 00~AF，and start code value of the belonged picture’s picture header is B3
9	P frame slice	Start code value is 00~AF，and start code value of the belonged picture’s picture header is B6, and the encoding mode in the picture header is 01
10	B frame slice	Start code value is 00~AF，and start code value of the belonged picture’s picture header is B6, and the encoding mode in the picture header is 10
11-23	Reserved	
24-31	Undefined	

2.7.2.4 RBSP byte

This is used for byte aligning the payload data which includes the start code value but excludes the start code prefix.

2.7.2.5 NAL unit de-limiters

A 4 byte sequence of NAL unit de-limiter is added after every NAL unit. Its value is ‘0000 0000 0000 0000 0000 0000 0000 0000 0000 0001’ or ‘0x000001’. NAL unit de-limiters are added just to aid in locating the start of NAL unit. The de-limiters are discarded during the decoding process.

Only when AVS part2 video bit stream has been mapped into NAL unit stream in this format, can the video bit stream be used to transmit over a network. After receiving the NAL unit stream and in order to decode, NAL unit headers along with the NAL unit de-limiters are discarded and replaced with a start code prefix value of ‘0x000001’ to convert the NAL unit stream back to AVS part2 video raw bit stream.

2.8 Summary

In this chapter AVS video standard and its encoding and decoding procedure are discussed in detail. AVS NAL format as discussed in section 2.7 is adopted in this thesis.

CHAPTER 3

OVERVIEW OF AAC STANDARD

3.1 Advanced audio coding standard

Perceptual encoding of high quality audio has found its way into many applications including digital radio, electronic music distribution systems and portable audio devices **[21]**. Over the recent years several audio coding technologies have been developed to cater to the needs of low bit rate audio applications such as digital audio broadcasting (DAB), accompanying audio for digital TV (DVB, ATSC) **[27]**, internet streaming etc. Advanced audio coding (AAC) is such a state-of-the-art technology employing a standardized lossy compression and encoding scheme for digital audio which was developed in a joint collaboration between the companies including AT&T Bell Laboratories, Farunhofer IIS, Dolby, Sony Corporation and Nokia. It has been standardized under the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) as part 7 of the MPEG-2 standard and part 3 of the MPEG-4 standard **[17], [18]**. AAC supports up to 48 audio channels with a wide range of sampling rates from 8 kHz to 96 kHz **[19]**. AAC also supports a variety of data rates which depend on the sampling rates. The maximum bit rate per channel for some most commonly used sampling rates are shown in table 3.1.

Table 3.1 Maximum bit rates/channel vs. sampling rate used in AAC **[19]**

Sampling rate (kHz)	**Maximum bit rate/channel (Kb/s)**
96000	576
48000	288
44100	264.6
32000	192

AAC can encode a CD quality audio signal of 48-64 Kbits/channel and the encoding efficiency is nearly 30 % more than MP3 (MPEG -1/2 audio layer 3) **[20]**.

3.2 Perceptual audio coding system

The basic task of a perceptual audio coding system is to encode the digital audio data in such a way that:

- The compression is as efficient as possible.
- The decoded audio data sounds exactly or as close as possible to the original audio data **[21]**.

The basic block diagram of the functional blocks involved in a perceptual audio coding system is shown in figure 3.1

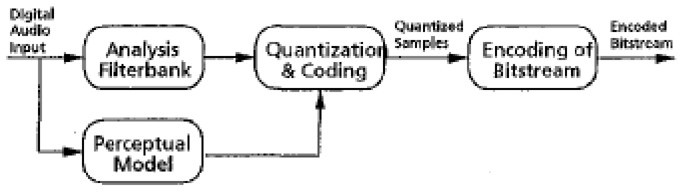


Figure 3.1 Block diagram of perceptual audio coding system **[21]**

The functional blocks used in a perceptual audio coding system are explained as follows.

3.2.1 Analysis filter bank

A filter bank consists of a series of band-pass filters that separates the input signal into sub-sampled spectral components. The process of decomposing the input signal at the encoder is called analysis and the process of reconstructing the original signal at the decoder is called as synthesis which together forms the analysis/synthesis section.

3.2.2 Perceptual model

The perceptual model is based on the principles of psychoacoustics which provides high quality lossy compression by describing a threshold on the input audio signal so that some parts of the input audio signal can be removed without significantly affecting the perceived quality of sound.

3.2.3 Quantization

It is another lossy compression scheme that is used to approximate a continuous range of values by a relative small set of discrete values.

3.2.4 Encoding of bit stream

The quantized data and the coded spectral coefficients along with other side information are converted into a bit stream using a bit stream formatter.

3.3 AAC profiles

AAC uses a modular approach to encoding. Depending upon the type of application, complexity and the desired performance an implementer can choose among the different profiles provided in AAC standard **[19]**. The three default profiles that have been defined in AAC standard are briefly explained as follows.

- Main profile: The main profile provides the highest audio quality and it is the most complex of the three profiles as it uses all the encoding and decoding tools except the gain control module.
- Low-complexity (LC) profile: The LC profile achieves nearly the same audio quality as the main profile but with significant savings on the memory and processing requirements.
- Scalable sampling rate (SSR) profile: The SSR profile provides flexibility for scalable and low-complexity applications. It adds the gain control module to the low complexity profile. It is more appropriate in applications where bandwidth is a constraint.

3.4 AAC encoder

The block diagram of the AAC encoder is shown in figure 3.2.

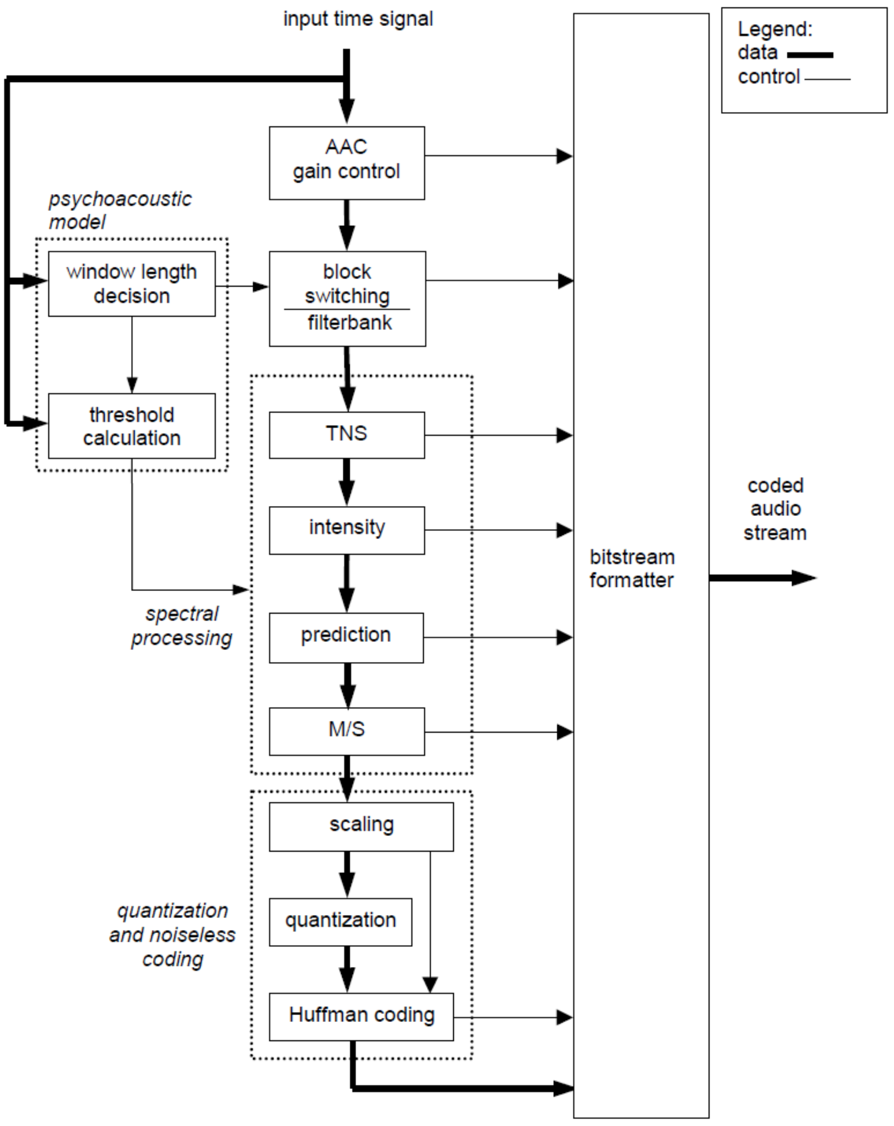


Figure 3.2 AAC encoder block diagram **[17]**

The description of the each functional block is as follows:

- Filter Bank: A filter bank is used to decompose the input audio signal into sub-sampled audio components called as blocks. The filter bank for the main and low complexity profiles uses modified discrete cosine transform (MDCT) **[19]** which is a Fourier-related transform based on the type–IV discrete cosine transform (DCT –IV). The MDCT is used with a transform length block dynamically switching between 1024 and 128 frequency points. In the case of SSR profile, a hybrid filter bank consisting of a poly-phase quadrature filter (PQF) is used in cascade with the MDCT block dynamically varying between 256 and 32 frequency points.

The MDCT transforms *2N* real numbers $x_0 \dots\dots\dots, x_{2N-1}$ into the *N* real numbers $X_0 \dots\dots\dots, X_{N-1}$ according to (3-1).

$$X_k = \sum_{n=0}^{2N-1} x_n \cos[\frac{\pi}{N}\left(n + \frac{1}{2} + \frac{N}{2}\right)\left(k + \frac{1}{2}\right)] \qquad (3\text{-}1)$$

The inverse MDCT transforms *N* real numbers $X_0 \dots\dots\dots, X_{N-1}$ into *2N* real numbers $y_0 \dots\dots\dots, y_{2N-1}$ according to (3-2).

$$y_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k \cos[\frac{\pi}{N}\left(n + \frac{1}{2} + \frac{N}{2}\right)\left(k + \frac{1}{2}\right)] \qquad (3\text{-}2)$$

- Temporal Noise Shaping (TNS): It employs frequency domain prediction technique to improve the temporal resolution and to reduce the quantization noise over time in the audio signal. It uses prediction in the frequency domain to control or shape the time-domain quantization noise in a frequency-dependant way.
- Joint Stereo Tool: Joint stereo is an encoding technique used in audio data compression to reduce the audio bit rate by exploiting redundancies across the channels. It consists of three different type of tools namely the mid/side stereo (M/S), intensity stereo and coupling. M/S stereo transforms the left/right channels into a mid and a side channel before quantization where mid channel is the sum of left and right channels and side channel is the difference of the left and right channels. In the case of intensity stereo and coupling, only the transform coefficients for one composite channel instead of all other input channels are transmitted along with the individual spectral envelopes of all the input channels so that the signal is scaled appropriately to reconstruct the original channels at the decoder. This technique reduces the audio bit rate with little or no change in sound quality **[19]**.
- Prediction Tool: The prediction tool is based on backward adaptive model which is used to eliminate the redundant information in the audio blocks.
- Quantization and Coding: The spectral data is quantized using a psychoacoustic model with a controlled allocation of bits. Huffman coding is used for the noiseless coding of these quantized coefficients which uses multiple switched codebooks with multiple dimensions.

- Bit Stream Formatter: The bit stream formatter places the quantized and the coded data along with other side information into a stream for transmission. The AAC bit stream is arranged in a frame wise manner with varying frame sizes.

3.5 AAC bit stream formats

The length of the AAC frames varies from frame to frame and each frame represents 1024 PCM samples per channel **[19]**. AAC supports two basic formats namely the audio data interchange format (ADIF) and audio data transport stream (ADTS) format. A brief description of the AAC bit stream formats is as follows.

- Audio Data Interchange Format (ADIF): This format allows only one header at the beginning of the file. The rest of the data are continuous raw audio data blocks. The ADIF header consists of information such as ADIF ID, copyright information, bit rate information and number of program configuration elements. This format is used for simple file storage purposes.
- Audio Data Transport Stream (ADTS): This header is suited more for transport related applications. This has one header for each AAC frame followed by the raw audio data blocks. ADTS header can also be present before 2 or 4 raw data blocks in a frame. Because of its error resilient features, ADTS format has been chosen for packetizing the audio streams in this thesis. ADTS header descriptions are given in Tables 3.2 and 3.3.

Table 3.2 ADTS header format **[18]**

Field name	Field size in bits	Comment
		ADTS Fixed header: This do not change from frame to frame
Syncword	12	always: '111111111111'
ID	1	0: MPEG-4, 1: MPEG-2
Layer	2	always: '00'
protection_absent	1	
Profile	2	
Sampling_frequency_index	4	
private_bit	1	
channel_configuration	3	
original/copy	1	
Home	1	
		ADTS Variable header: This can change from frame to frame
Copyright_identification_bit	1	
Copyright_identification_start	1	

Table 3.2 – *Continued*

aac_frame_length	13	length of the frame including header (in bytes)
ADTS_buffer_fullness	11	0x7FF indicates VBR
No_raw_data_blocks_in_frame	2	
		ADTS Error check
crc_check	16	Only if protection_absent == 0
Raw block of data	Variable	

Table 3.3 ADTS profile bits **[18]**

Profile bits	*ID 1 (MPEG-2 profile)*
00 (0)	Main profile
01 (1)	Low complexity profile (LC)
10 (2)	Scalable sample rate profile (SSR)
11 (3)	(reserved)

3.6 AAC summary

In this chapter the AAC audio coding standard and its encoding scheme are described in detail. The advantages of AAC over the other audio coding standards as detailed in this chapter justify the reason for choosing AAC standard in this thesis. Low complexity AAC profile along with ADTS bit stream formatting has been adopted in this thesis. The next chapter deals with the packetization techniques used in the multiplexing of AVS part 2 video and AAC audio bit streams.

CHAPTER 4

MULTIPLEXING

4.1 Multiplexing fundamentals

Multiplexing is the process of converting multiple elementary streams into a single transport stream for data transmission. In the case of digital television transmission standards (DTV) such as ATSC **[22, 27]**, DVB-T and DVB-H **[24]** or in the case of streaming technologies such as IPTV, video telephony services **[24]**, it is necessary to encode both sound and picture signals and multiplex them along with other supplementary information into a format suitable for the transmission link **[23]**. In the case of high definition television (HDTV) services high quality video and audio data are transmitted which occupy a lot of bandwidth over a broadcast channel. To address this issue the video and audio data are compressed using efficient compression schemes such as AVS China **[4]** for video and AAC **[17]** for audio which preserve the data quality but at the same time also reduces the bandwidth for transmission. A standard has been defined in MPEG-2 **[25]** by the (ISO)/ (IEC) on multiplexing and synchronizing the coded video, audio and other supplementary data. The same frame work with slight modifications has been adopted in this thesis for multiplexing the AVS China part 2 video and AAC audio elementary streams. The digital transmission and reception process used in the ATSC standard is shown in figure 4.1.

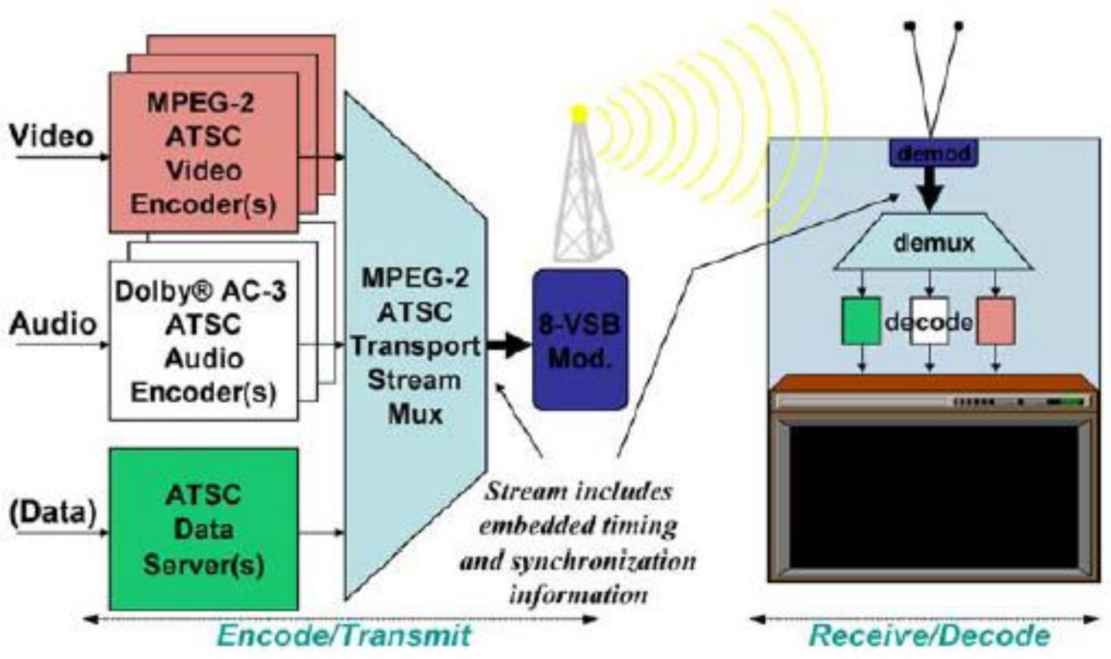

Video
MPEG-2 ATSC Video Encoder(s)
Audio
Dolby® AC-3 ATSC Audio Encoder(s)
(Data)
ATSC Data Server(s)
MPEG-2 ATSC Transport Stream Mux
8-VSB Mod.
Stream includes embedded timing and synchronization information
demod
demux
decode
Encode/Transmit
Receive/Decode

Figure 4.1 The digital transmission/reception process used in the ATSC standard **[22]**

In order for an effective transmission and reception of the elementary streams the following factors should be considered while multiplexing these streams. First, while multiplexing the elementary streams, every elementary stream should be given equal priority which is needed to prevent any buffer overflow or underflow at the receiver side which leads to loss in data packets. Secondly, apart from the coded elementary streams the multiplexed streams should also contain information to play the elementary streams with synchronization at the receiver. These types of additional information which are used for synchronization are called time stamps. Finally if the transmission takes place in an error prone network, certain additional information needs to be added in the multiplexed stream to detect these errors and rectify them.

4.2 Packetization

The first step in the process of multiplexing is known as packetization. Packetization refers to splitting the long stream of data i.e. the audio and video elementary streams into small data packets which helps in transmitting the data efficiently and reliably. A packet usually consists of two kinds of data namely the control information and the user data which is also known as payload. The time stamps and other control information are embedded in the control information header. There are two layers of packetization method adopted in the MPEG-2 systems **[25]**. The first layer of packetization is known as the packetized elementary stream (PES) format and the second layer is known as the transport stream (TS) format. The multiplexing process takes place after the second layer of packetization which is used for transmission. Figure 4.2 shows the two layer of packetization adopted in the MPEG-2 systems.

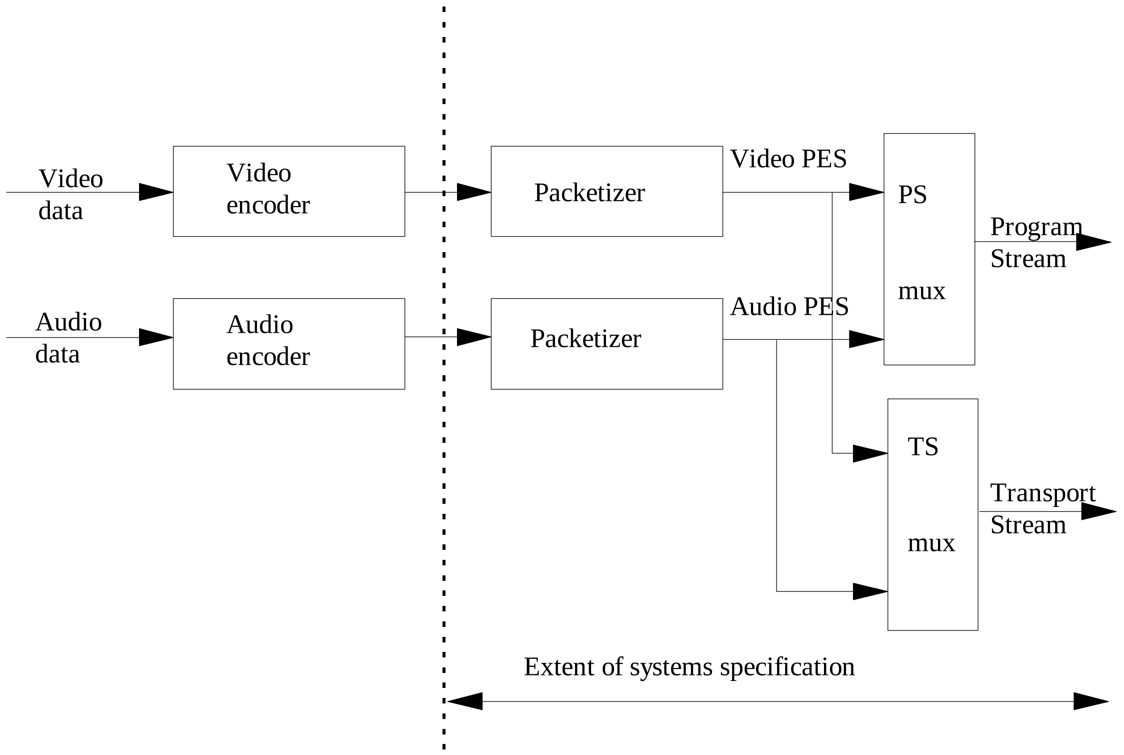


Figure 4.2 Two layers of packetization method adopted in MPEG-2 systems **[25, 26]**.

The previous two chapters described in detail the method of obtaining and encoding the elementary AVS China video and AAC audio streams. Only video and audio streams have been chosen as the elementary streams for the implementation part in this thesis. The adopted algorithm can also be extended to support data streams and other supplementary information.

4.3 Packetized elementary stream (PES)

The packetized elementary stream (PES) forms the first layer of packetization. The PES packets are obtained by encapsulating the coded video, audio and data elementary streams. The encapsulation process in the case of video and audio elementary streams is done by sequentially splitting the packets into access units which are the audio and video frames of audio and video elementary streams respectively. Each PES packet can contain data from only one elementary stream. The PES packet consists of the PES packet header followed by the PES packet payload. The PES packet payload can be of variable size since the frame size in both the audio and video elementary stream is variable. The header information differentiates the various elementary streams and it also carries the control information for synchronization in the form of time stamps and other useful information. The encapsulation process of PES packets from the elementary streams is shown in figure 4.3.

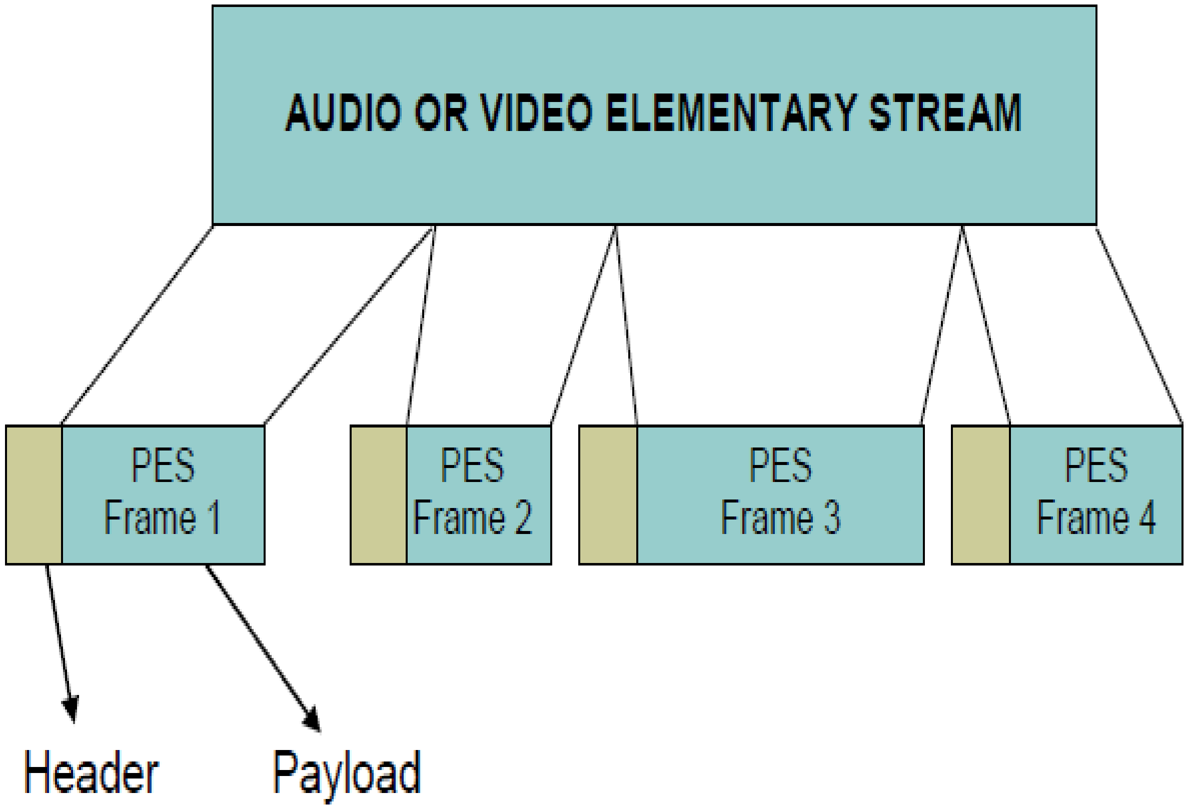
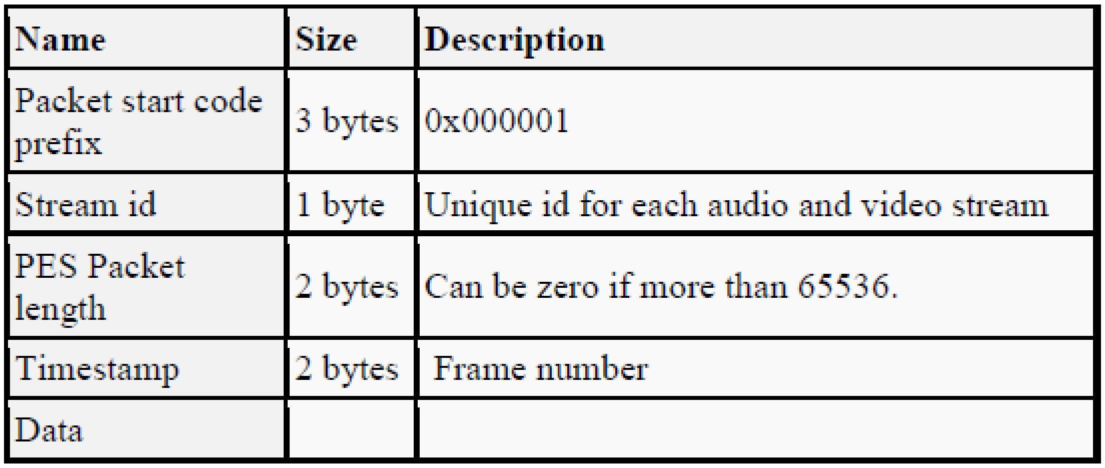


Figure 4.3 Encapsulation process of PES packets from the video or audio elementary stream **[25]**.

4.3.1 PES packet header

The PES packet header is described in Table 4.1.

Table 4.1 PES packet header **[25]**



Name	Size	Description
Packet start code prefix	3 bytes	0x000001
Stream id	1 byte	Unique id for each audio and video stream
PES Packet length	2 bytes	Can be zero if more than 65536.
Timestamp	2 bytes	Frame number
Data		

The PES packet header is an 8 byte header consisting of the following parameters.

- Packet start code prefix is a fixed 3 byte value consisting of ‘0x000001’ bit stream value.
- Stream id is a 1 byte value which is unique for each video and audio elementary stream.
- PES packet length is a 2 byte value which indicates the frame size of the video or the audio elementary stream contained in that particular PES packet in bytes. Its value can take up to a maximum size of 65536 bytes (i.e. from 0-65535 bytes). In the case of elementary streams with longer frame size the PES packet length may be indicated as unbounded by setting its value to zero.
- Timestamp is a 2 byte value which consists of the frame number of the corresponding video or elementary stream which is used for synchronization. This is discussed in detail in section 4.6.

4.3.2 PES packet payload

The PES packet payload consists of either video or audio frame. If it is an audio PES packet, the AAC packet is searched for the 12 bit sync word in the ADTS header. Then the frame length is obtained from the aac_frame_length field value in the audio data transport stream (ADTS) header which is a 13 bit value and the corresponding block of data is encapsulated with the audio stream ID to form the audio PES packet. Audio frame number is calculated from the beginning of the audio frame and it is encapsulated as the 2 byte timestamp value. If the payload consists of a video frame, the AVS encoded bit stream is searched for its NAL unit using the 4 byte NAL de-limiter sequence i.e. ‘0x00000001’ value which is the prefix of the NAL unit. The five LSB bits of the NAL unit header are analyzed to determine the frame type. If the NAL unit contains I, P or B frame the frame number is calculated from the beginning of the stream and is encapsulated as time stamp along with the video stream ID and frame length. Since the PES packet payload has a variable size it is more desirable to transmit fixed size packets in an error prone transmission channel. This leads to another layer of packetization known as the transport stream format.

4.4 Second layer of packetization methods

There are basically two types of second layer of packetization methods specified in the MPEG-2 systems **[25]**. They are the program stream format and the transport stream (TS) format. Program stream (PS) format uses the variable length transport packets approach where as the transport stream format uses the fixed length transport packets approach. In the TS approach each PES packet of video or audio stream occupies a variable number of transport packets, and data from video and audio bit streams are interleaved with each other at the final transmitted stream along with the transport headers consisting of the unique identification of each elementary bit stream and other useful information. In the PS approach PES packets of video or audio bit stream are multiplexed by transmitting the bits for the complete PES packets aligned in sequence, which results in a sequence of variable length packets **[28]**. Transport stream packetization is adopted in environments which are error prone where errors and loss of data packets are more likely to occur such as transmission on noisy channels, satellite and cable DTV systems etc. Program stream on the other hand is adopted for relatively error free media such as DVD-ROMs, Blu-ray disc etc. The transport stream format is adopted in this thesis.

4.5 Transport stream format

A transport stream packet is of fixed length of 188 bytes and it always begins with a synchronization byte of 0x47. The choice of this packet structure was motivated by few important factors. They are,

- The packets need to be large enough so that the overhead of the transport headers does not become a significant portion of the total data being carried **[28]**.
- The packet size should not be too large that the probability of packet error becomes significant under the standard operating conditions **[28]**.

- Another factor is the interoperability with the ATM packets as each MPEG-2 transport stream packet is transmitted in four ATM packets **[28, 29]**.

There are few factors which need to be considered while forming the transport stream packets. They are,

- The total packet size should be of fixed length i.e. 188 bytes.
- Each packet can have data from only one PES.
- PES header should be the first byte of the transport packet payload.
- If the above constraints are not met, the PES packet is split and additional stuffing bytes are added.

The encapsulation process of TS packets from PES packets is shown in the figure 4.4.

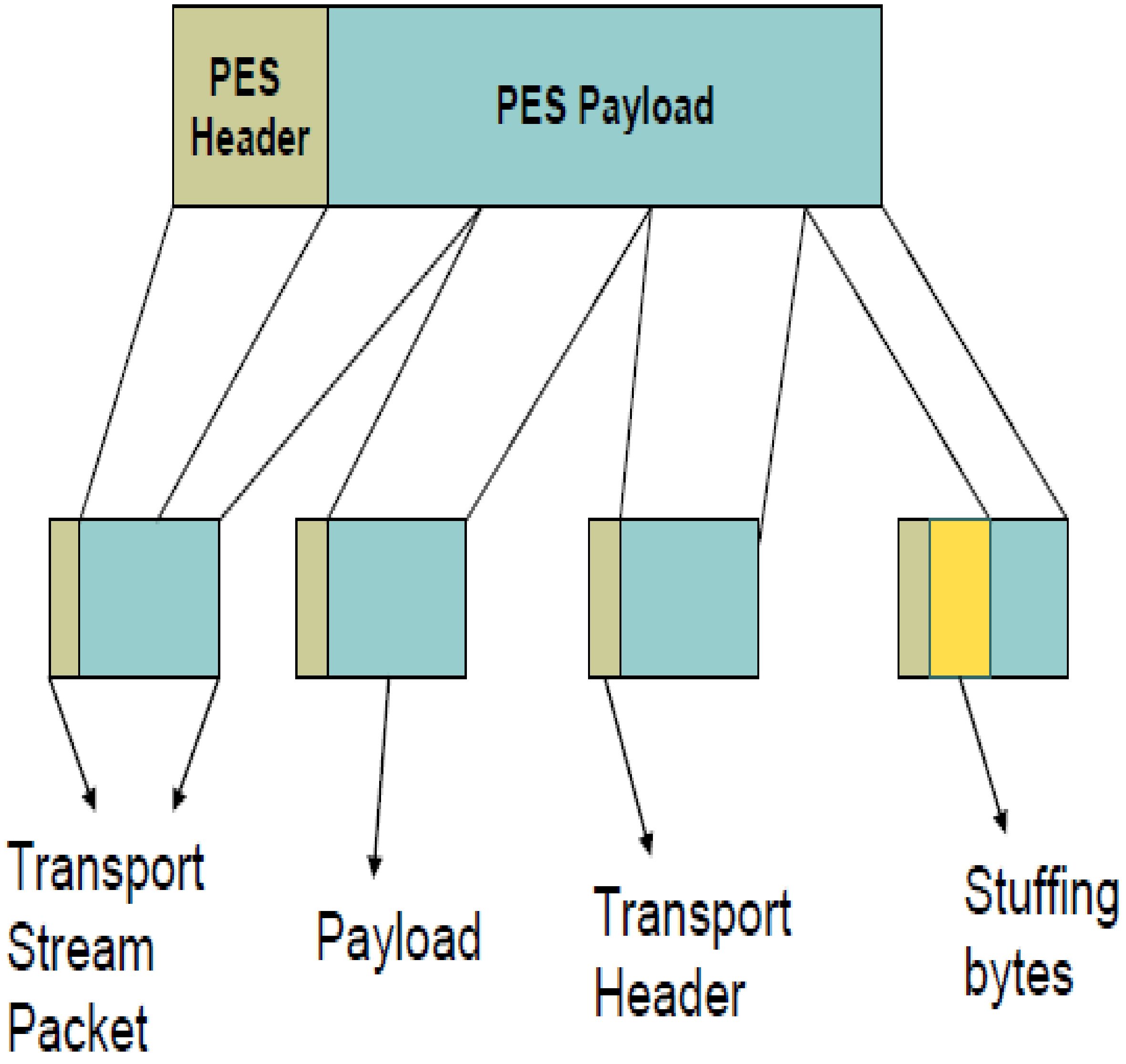
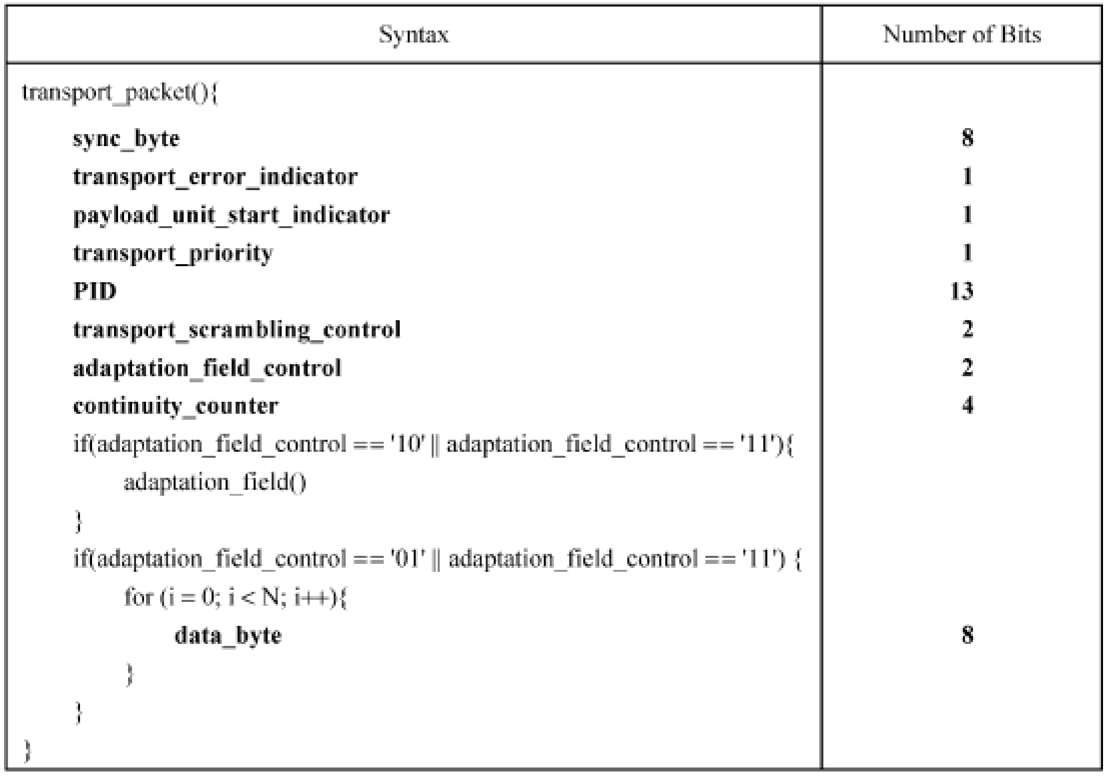


Figure 4.4 Encapsulation of TS packets from PES packets **[25]**

4.5.1 TS packet header

A TS packet normally consists of a 3 byte header with 185 bytes allocated for payload however if adaption field is present, then an additional byte is added to the header and 184 bytes are allocated for the payload. The TS packet header description as adopted in MPEG-2 systems is shown in Table 4.2.

Table 4.2 TS packet header description as adopted in MPEG 2 systems **[22]**



Syntax	Number of Bits
transport_packet(){	
sync_byte	**8**
transport_error_indicator	**1**
payload_unit_start_indicator	**1**
transport_priority	**1**
PID	**13**
transport_scrambling_control	**2**
adaptation_field_control	**2**
continuity_counter	**4**
if(adaptation_field_control == '10' || adaptation_field_control == '11'){	
adaptation_field()	
}	
if(adaptation_field_control == '01' || adaptation_field_control == '11') {	
for (i = 0; i < N; i++){	
data_byte	**8**
}	
}	
}	

The TS packet header with slight modifications as shown in Table 4.3 has been adopted in this thesis.

Table 4.3 TS packet header adopted in the thesis **[22, 25]**

Syntax	**Number of bits**
sync byte	8
Payload unit start indicator	1
adaptation field control	1
PID	10
Continuity Counter	4
if adaptation field control = =1	
payload byte offset	8
stuffing bytes or additional header	
payload	

The description of TS packet header is as follows.

- Sync byte: A TS packet always starts with a sync byte of ‘0x47’.
- Payload unit start indicator: This bit is set to indicate that the first byte of the PES packet is present in the payload of the current TS packet.
- Adaptation field control: This bit is set if the data carried in the TS packet payload is other than the PES data. This can be a stretch of stuffing bytes (0xff) in case the length of PES data is less than 185 bytes or any other data. When this bit is set, an additional 1 byte header is added just after the TS header which indicates the byte offset.
- PID (Packet Identifier): This is a 10 bit packet identifier value. This is used to uniquely identify the video or audio elementary streams. Some values of the PID are pre-defined and are used to indicate some control information. A TS packet with an unknown PID is discarded at the receiver.

The particular PID value of ‘0x1024’ is used to indicate a null packet. The null TS packet is a special TS packet designed to create a specific overall constant bit stream. Null TS packets are transmitted when there are no other TS packets available for transmission **[22]**. These packets are discarded when received at the de-multiplexer.

- Continuity counter: This is a 4 bit counter which is incremented by one every time the data from the same PES is encapsulated into a TS packet i.e. for each consecutive TS packet of the same PID.
- Payload byte offset: If adaptation field control bit is set to 1, byte offset value of the start of the payload or the length of adaptation field is mentioned here.

The entire above mentioned processes are pictorially represented in figure 4.5.

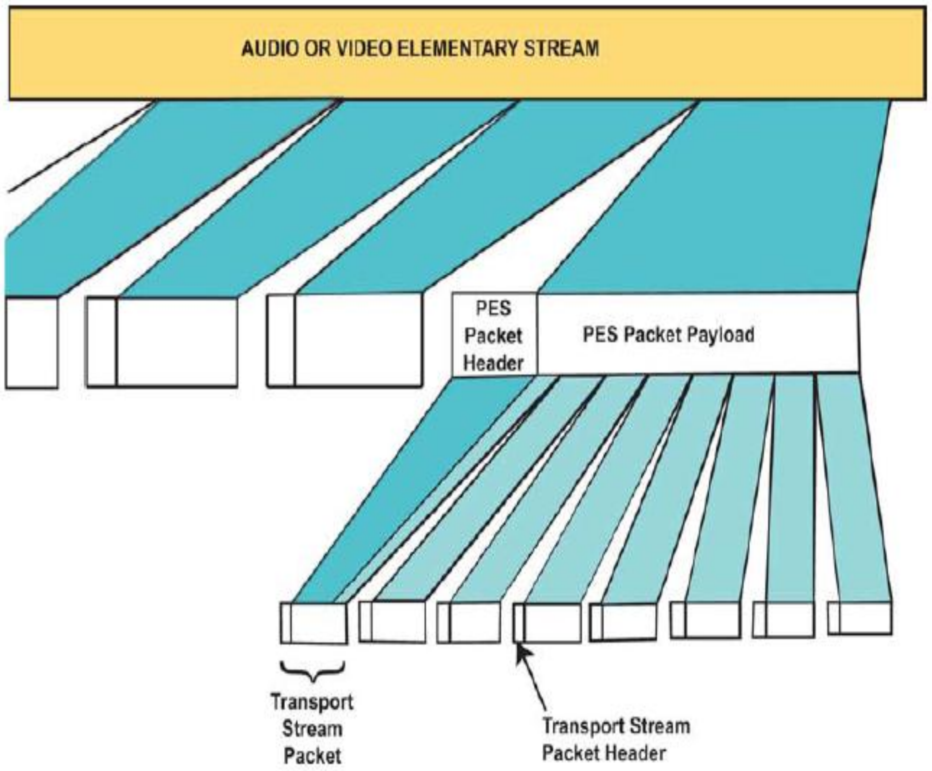

AUDIO OR VIDEO ELEMENTARY STREAM
PES Packet Header
PES Packet Payload
Transport Stream Packet
Transport Stream Packet Header

Figure 4.5 Hierarchy of MPEG-2 TS packet structure **[22]**

<u>4.6 Frame number as time stamp</u>

The method adopted in this thesis uses the frame numbers as time stamps **[30]** for the synchronization of AVS video and AAC audio streams. AVS video and AAC audio bit streams are arranged in frame wise format. This factor is used for synchronizing the audio and video elementary streams at the de-multiplexer for playback. A particular video bit stream has a constant bit rate during playback which is initially set while encoding. The frame rate for the video stream is specified by the frames per second (fps) value. Since the frame rate is already known, the time of occurrence of a particular frame in the video sequence while playback can be calculated using the formula specified in (4-1).

$$\text{Playback time} = \text{Frame number/fps} \qquad (4\text{-}1)$$

The AAC compression defines an AAC frame to contain 1024 samples. The sampling frequency specified in the AAC standard varies between 8 and 96 KHz. The frame duration increases from 96 KHz to 8 KHz. Although AAC standard supports different sampling frequencies, the sampling rate is fixed while encoding the AAC bit streams hence the frame duration also remains constant throughout a particular audio stream. Since the sampling frequency is already known, the time of occurrence of a particular audio frame can be calculated using the formula specified in (4-2).

$$\text{Playback time} = 1024 * (1/\text{sampling frequency}) * \text{frame number} \qquad (4\text{-}2)$$

Thus by using (4.1) and (4.2) the playback time of the encoded video and audio sequences can be calculated using frame numbers as time stamps which means that given the frame number of one stream, the frame number of the other streams played at the same time during playback can be calculated. This helps in synchronizing the video and audio sequences during playback. The same concept can be extended to synchronize multiple elementary streams i.e. in the case of single video stream with multiple audio channels (stereo). The time stamps are assigned in the last 2 bytes of the PES header, hence the maximum frame number that can be carried in this header is 65535. In the case of long video or audio elementary streams the frame number is rolled over to zero and this takes place simultaneously on both the video and audio frame numbers. Since the buffer size at the de-multiplexer is much smaller than the maximum allowed frame number, at no point of time there will be two frames in the buffer with the same time stamp.

4.7 Advantages of frame numbers over the existing method for time stamps

In the MPEG-2 systems standard the method used for audio-video synchronization is based on the presentation time stamps (PTS). The encoder attaches a PTS to video and audio frames which is a

33 bit value in cycles of a 90-kHz system time clock (STC) **[22]**. This STC clock is regenerated at the receiver and the clock samples are compared before presentation to achieve synchronization. For this to work effectively, the clocks at the multiplexer and de-multiplexer should be exactly synchronized. For achieving this, additional information known as program clock reference (PCR) which is the value of the STC at the encoder is periodically transmitted. The advantages of using frame numbers over the adopted method in MPEG-2 systems are as follows.

- Less complex and is suitable for software implementation.
- No synchronization problem due to clock jitters.
- No propagation of delay between audio and video elementary streams.
- Saves the extra over head in the PES header bytes used for sending the PCR bytes.

4.8 Proposed multiplexing method

The final transmission stream is formed by multiplexing the transport stream packets. The number of packets allocated for a particular elementary stream plays an important role in avoiding the buffer overflow or underflow at the de-multiplexer end. A buffer overflow or underflow situation arises at the de-multiplexer when the video buffer is full and the audio buffer does not have enough data or vice versa. This will eventually result in the loss of data while playback. To avoid such situations, timing counters are employed at the multiplexer which gets incremented every time a TS packet from the same elementary stream is transmitted. The incremented value depends on the playback time of each TS packet which can be calculated from the playback time of PES packet since the frame duration is known and it remains constant. The elementary stream with the least timing counter value is given preference. This method will make sure that at any point of time, the difference in the buffer fullness in terms of playback time is less than the playback time of one TS packet which is never more than the duration of a single frame. A brief example of calculating the playback time of a TS packet is as follows.

Suppose,

The PES packet length of a video ES = 1000 bytes

Since the frame rate is already known (say 25 fps), duration of each PES =1/ 25=40ms.

The number of TS packets generated from the video PES = 1000/185 ~= 6 packets

Therefore, duration of each TS packet can be calculated as 40/6 = 6.67ms.

4.9 Summary

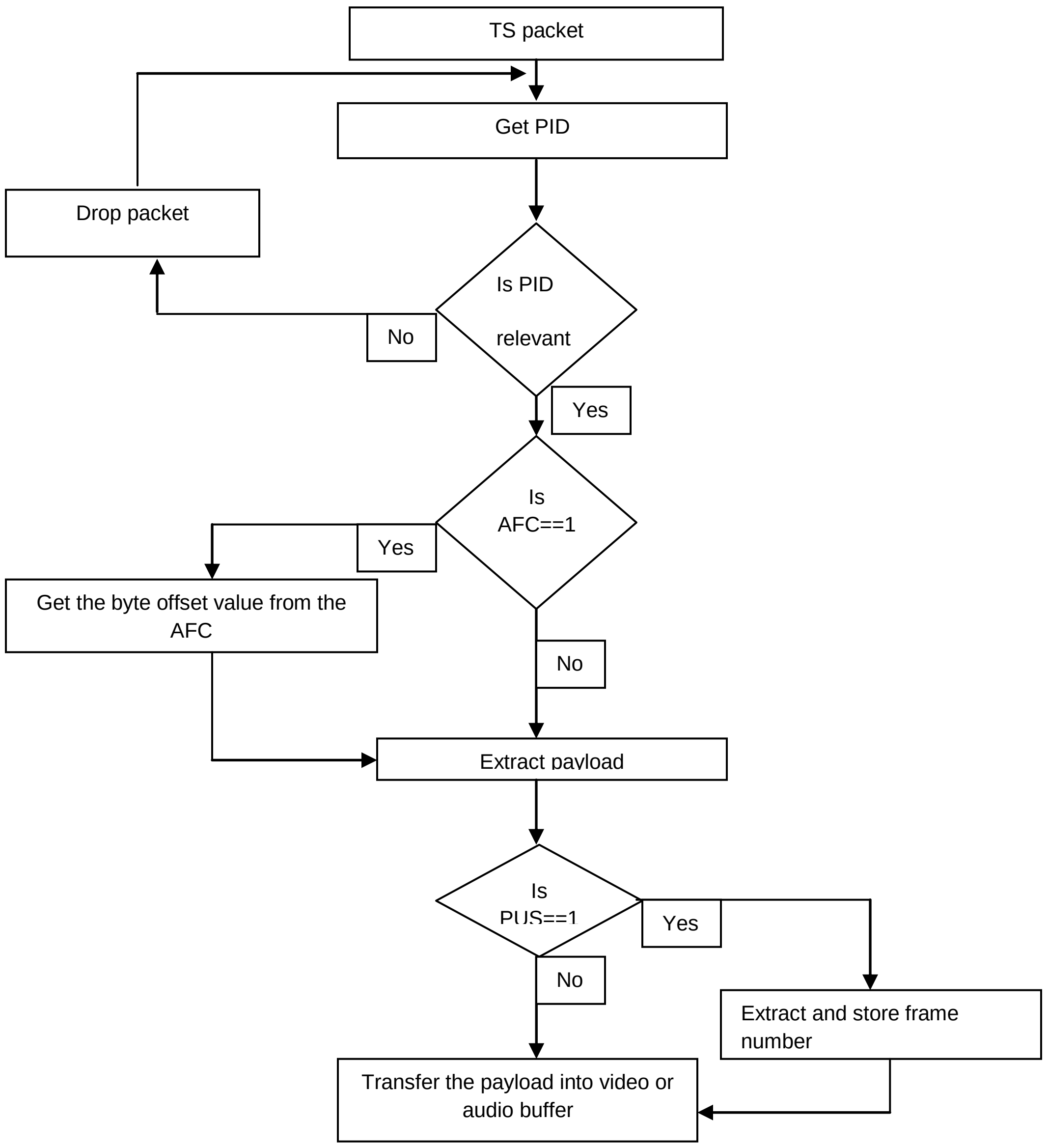
In this chapter the adopted multiplexing method is discussed in detail. The advantages of using frame numbers as time stamps over the existing method are described. Finally, an effective method of multiplexing the transport stream packets to prevent buffer overflow or underflow at the de-multiplexer is presented. The next chapter deals with de-multiplexing the transport stream packets with lip synchronization between the elementary streams during playback.

CHAPTER 5

DEMULTIPLEXING WITH LIP SYNCHRONIZATION

5.1 De-multiplexing

The flow chart of the de-multiplexing process adopted in this thesis is shown in figures 5.1 and 5.2.



TS packet
Get PID
Drop packet
Is PID relevant
No
Yes
Is AFC==1
Yes
Get the byte offset value from the AFC
No
Extract payload
Is PUS==1
Yes
No
Extract and store frame number
Transfer the payload into video or audio buffer

Figure 5.1 Flowchart of the de-multiplexer

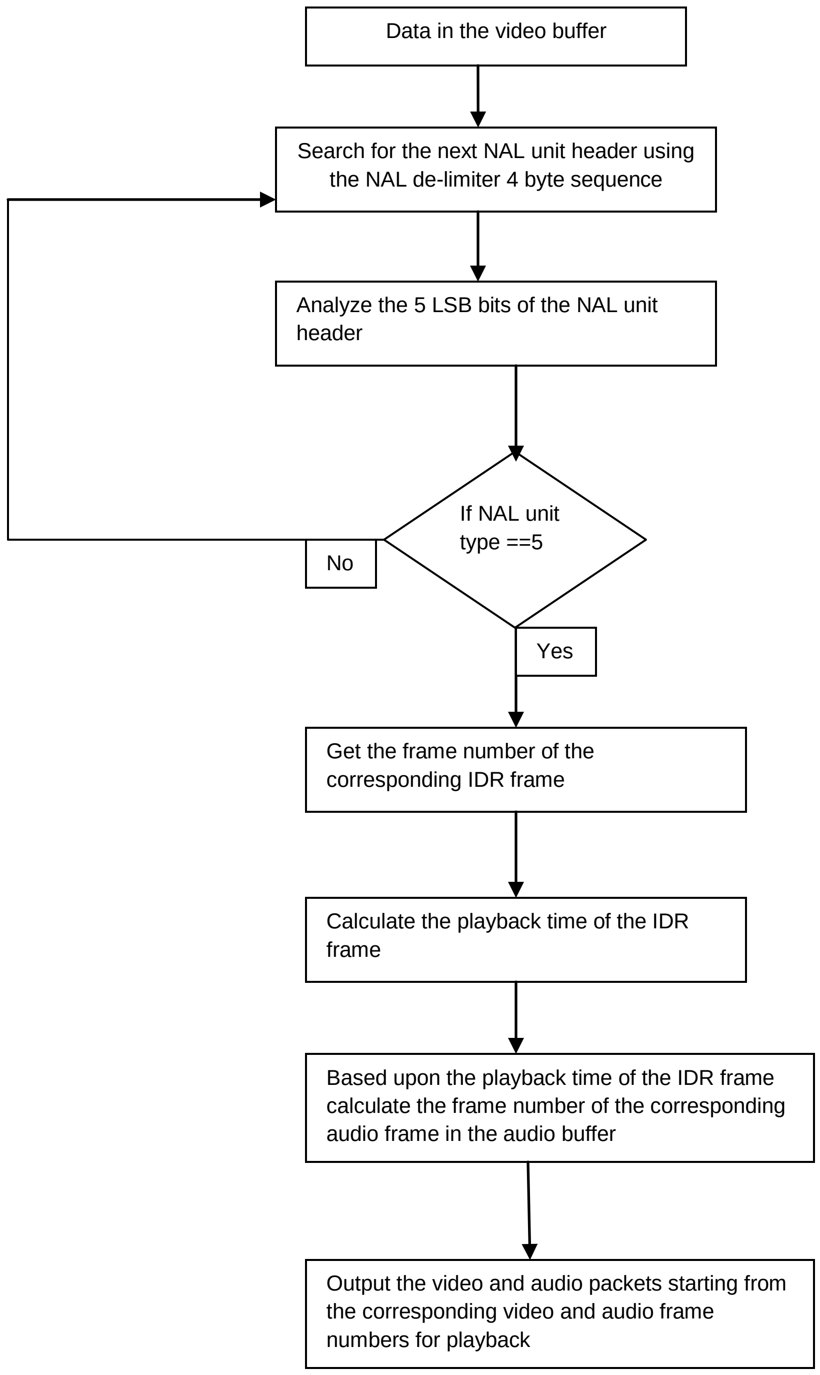

Data in the video buffer
Search for the next NAL unit header using the NAL de-limiter 4 byte sequence
Analyze the 5 LSB bits of the NAL unit header
If NAL unit type ==5
No
Yes
Get the frame number of the corresponding IDR frame
Calculate the playback time of the IDR frame
Based upon the playback time of the IDR frame calculate the frame number of the corresponding audio frame in the audio buffer
Output the video and audio packets starting from the corresponding video and audio frame numbers for playback

Figure 5.2 Flow chart for achieving lip synchronization between the video and audio streams during playback at the de-multiplexer end

The de-multiplexing process at the receiver end is explained in a step by step process as follows.

- Upon receiving the multiplexed TS packets, the PID values from the transport stream header are extracted. If the extracted PID value is unknown or if it is not relevant then the corresponding packet is dropped and the next packet is analyzed.

- Once the obtained PID value is extracted, the adaptation field control (AFC) bit is checked to verify any data other than the elementary streams is present in the TS packet. If the AFC bit is ‘1’ then the byte offset value is obtained from the header and the data is read from the byte offset value till the end and the remaining data is discarded if they are filled with stuffing bytes.. If AFC bit is not set then the data is directly read after the header.
- The packet is also analyzed to check if payload unit start indicator bit is set to ‘1’. If it is set, then it means that PES header is present in the corresponding TS packet. The PES header of the packet is analyzed to obtain the frame length and frame number of the corresponding video or audio elementary stream. The frame length and the frame number (timestamp) are stored in a separate buffer. This process is repeated until one of the elementary stream buffers is full.
- The 4 bit continuity counter is also monitored for each PID separately to check for any loss in data packets. If the counter value does not increment in sequence then a packet loss is detected. In some cases re-transmission of the packet is requested to correct the error or the corresponding frame is skipped during playback to prevent any halt in the decoding process.

The buffer fullness is also continuously monitored at the de-multiplexer end to prevent any buffer overflow or underflow as it would result in the loss of data packets. The buffer fullness at the de-multiplexer end using the method adopted is shown in Table 5.1.

Table 5.1 Buffer fullness at the de-multiplexer using the adopted method in this thesis

Test criteria / Buffer details	Video buffer 300 frames	Video buffer 300 frames	Video buffer 600 frames
Start TS packet number	1	5000	1
End TS packet number	3069	8540	6773
Video buffer fullness (KB)	323 KB	407 KB	763 KB
Audio buffer fullness (KB)	132 KB	132 KB	278 KB
Number of video frames	300	300	600
Number of audio frames	507	519	1024
Video frame rate (fps)	25	25	25
Audio sampling rate (kHz)	44100 kHz	44100 kHz	44100 kHz
Video buffer content playback time	12 sec	12 sec	24 sec
Audio buffer content playback time	11.77 sec	12.05 sec	23.77 sec

It can be clearly interpreted from Table 5.1 that the audio and video buffer fullness, using the adopted method has nearly the same amount of playback time, irrespective of the buffer size or the TS start packet.

5.2 Synchronization and playback

Once either one of the elementary stream buffer is full, the content is ready to be played back. The ADTS format in the AAC frame helps the decoder to start decoding from any AAC. For the AVS video frame, however, the decoder starts decoding only after it receives an I frame. Hence I or IDR frames are forced in the encoder at regular intervals to assist the decoding process.

First the video buffer is searched from the beginning to get the first IDR frame in that buffer. Once this is found the timestamp or frame number of that particular frame is obtained. Then the video playback time is calculated using (5-1).

$$\text{Video playback time} = \frac{Video\ frame\ number}{Video\ frame\ rate} \tag{5-1}$$

Once the video playback time is known the corresponding AAC frame number is calculated according to (5-2).

$$\text{Audio frame number} = \frac{Video\ playback\ time * sampling\ frequency}{1024} \tag{5-2}$$

If the calculated frame number is not found in the buffer then next IDR frame is obtained from the video buffer and the corresponding AAC frame for synchronization is calculated. When the calculated AAC frame is not an integer value then it is rounded off to the nearest integer value which restricts the maximum time difference between the video and audio streams to less than 40 ms, which is the maximum allowed delay as specified in the MPEG-2 systems **[25]**. Once the frame numbers are obtained, the block of data from that particular frame number till the end of the buffer is sent to the decoder for playback. Then the buffer is emptied and new set of incoming data is filled in the buffer. Then the data from the buffer is played back using MPlayer **[34]**.This completes the process of multiplexing and de-multiplexing the elementary video and audio streams with lip synchronization during playback. Since the current standard for multiplexing and de-multiplexing the elementary streams is not freely available, it makes it difficult in comparing the proposed method with the existing standard.

5.3 Summary

In this chapter the process of extracting the elementary video and audio streams from the transport stream packets has been described in detail and also an analysis of the buffer fullness at the de-multiplexer end has been shown. Finally the method for achieving synchronization between the elementary video and audio streams using frame numbers as timestamps has been proposed. The next chapter deals with the simulations and results.

CHAPTER 6

RESULTS AND CONCLUSIONS

6.1 Implementation and results

The proposed multiplexing method was implemented on a single multimedia program consisting of video and audio elementary streams. A small clip with duration of 54 sec was downloaded from YouTube and converted to an AVI format sequence. The raw YUV format video elementary stream and raw WAVE format audio elementary stream are extracted from this AVI sequence using the ffmpeg software **[33]**. The extracted video elementary stream in YUV format is encoded to AVS format sequence using the AVS China part2 software **[32]**. The extracted WAVE format elementary stream is encoded to AAC format using the FAAC audio encoder **[31]**. The audio stream is encoded in low complexity profile with ADTS as its transport header and the sampling rate chosen for the sequence is 44.1 KHz. The video elementary stream is encoded at 25 frames per second. Then the encoded video and audio streams are multiplexed to form the TS packets. The details of both the clips and multiplexed stream are given in Table 6.1.

Table 6.1 Results of multiplexed stream on test clip

Test clip details	Clip 1
Duration of clip (sec)	54
Audio frequency (Hz)	44,100
Video frame rate (fps)	25
YUV file size (MB)	196
WAVE file size (MB)	9.12
AVS China file size (MB)	2.07
AAC file size (kB)	636
Video coder compression ratio	94.685
Audio coder compression ratio	14.33
Number of TS packets	17338
Total transport stream size (kB)	3259
AVI (*.avi) file size (MB)	5.69 MB
AVS media file (*.avs) file size (MB)	2.07 MB

The results in the Table 6.1 show that the compression achieved using the AVS China part 2 video codec and AAC audio codec is much more than that of the AVI files. Synchronization between video and audio is achieved regardless of the TS packet start number as specified by the user to start the de-multiplexing process. Hence some random TS packets are chosen to begin the de-multiplexing process and the synchronization delay between the video and audio start frames is observed in Table 6.2.

Table 6.2 Observed synchronization delay on test clip

Start TS packet number	Synchronized frame numbers chosen		Video frame playback time (sec)	Audio frame playback time (sec)	Delay (msec)	Visual delay
	Video	Audio				
100	35	52	1.2	1.207	7	Not perceptible
500	65	103	2.4	2.391	9	Not perceptible
1500	185	310	7.2	7.198	2	Not perceptible
3000	305	517	12	12.004	4	Not perceptible
6000	545	930	21.6	21.594	6	Not perceptible
10000	905	1550	36	35.990	10	Not perceptible
11500	965	1654	38.4	38.405	5	Not perceptible
13000	1055	1809	42	42.004	4	Not perceptible
14500	1205	2067	48	47.995	5	Not perceptible

Table 6.2 shows that the delay between the video and audio elementary streams does not exceed 10 milliseconds. The delay almost remains constant throughout the playback once the audio and video frames are synchronized.

6.2 Conclusions

Thus an effective method for multiplexing and de-multiplexing the elementary streams with lip synchronization is proposed in this thesis. The thesis conforms to the MPEG -2 systems **[25]** by adopting two layers of packetization namely the packetized elementary stream layer and transport stream layer for multiplexing the video and audio elementary streams. An effective method for achieving lip synchronization between the video and audio elementary streams during playback which uses frame numbers as time stamps is proposed and is justified in this thesis. AVS China part 2 video **[8]** elementary stream and AAC audio elementary stream **[17, 18]** are used in this thesis to transmit high quality multimedia program with reduced bit rates. Thus, this thesis meets all the requirements to transmit a high quality multimedia program over a broadcast network.

6.3 Future research

The proposed algorithm aims at multiplexing single video and audio elementary streams. This algorithm can be extended to support multiple elementary streams such as to include subtitles while playback and other information. The proposed algorithm can also be modified to support elementary streams from different video and audio codecs depending on their NAL and ADTS formats respectively. The adopted method can also be extended to support some error resilient codes in the case of transmission of the multimedia program over error prone networks.

APPENDIX A

MATLAB SOURCE CODE FOR MULTIPLEXING

Multiplexing – Matlab source code

```
function NAL_AVS()

fp=fopen('C:\Users\sanju\Documents\Mux-demux\thesis\other softwares\fluffy_AVS.avs');

Video_raw=fread(fp);

M=length(Video_raw);

forbidden_bit='0';

m=0;

for i=1:M-3

    if (Video_raw(i)==0 && Video_raw(i+1)==0 && Video_raw(i+2)==1)

        m=m+1;

        temp2(m)=i;

        if m>=2

        A=temp2(m-1);

        end

        Start_code_type=dec2hex(Video_raw(i+3));

         if Start_code_type=='B0' %#ok<STCMP>

            NAL_ref_idc='11';

            NAL_unit_type=dec2bin(1,5);

        elseif Start_code_type=='B5'

            NAL_ref_idc='00';

            NAL_unit_type=dec2bin(2,5);

        elseif Start_code_type=='B2'

            NAL_ref_idc='00';

            NAL_unit_type=dec2bin(3,5);

        elseif Start_code_type=='B7'

            NAL_ref_idc='00';

            NAL_unit_type=dec2bin(4,5);

        elseif Start_code_type=='B3'
```

```
            NAL_ref_idc='11';

            NAL_unit_type=dec2bin(5,5);

        elseif Start_code_type=='B1'

            NAL_ref_idc='11';

            NAL_unit_type=dec2bin(11,5);

        elseif Start_code_type=='B6'

            temp1=dec2bin(Video_raw(i+6),8);

            picture_coding_type=temp1(1:2);

            if picture_coding_type=='01'

            NAL_ref_idc='10';

            NAL_unit_type=dec2bin(6,5);

            elseif picture_coding_type=='10'

            NAL_ref_idc='01';

            NAL_unit_type=dec2bin(7,5);

            end

         elseif (hex2dec(Start_code_type)>=0 & hex2dec(Start_code_type)<=175) &
dec2hex(Video_raw(A+1))=='B3'

            NAL_ref_idc='11';

            NAL_unit_type=dec2bin(8,5);

          elseif (hex2dec(Start_code_type)>=0 & hex2dec(Start_code_type)<=175) &
dec2hex(Video_raw(A+1))=='B6' & picture_coding_type=='01'

            NAL_ref_idc='10';

            NAL_unit_type=dec2bin(9,5);

           elseif (hex2dec(Start_code_type)>=0 & hex2dec(Start_code_type)<=175) &
dec2hex(Video_raw(A+1))=='B6' & picture_coding_type=='10'

            NAL_ref_idc='01';

            NAL_unit_type=dec2bin(10,5);

         end
```

```
          NAL_header=strcat(forbidden_bit,NAL_ref_idc,NAL_unit_type);
          NAL_delimiter='00000000000000000000000000000001';
          Video_NAL_header(m,1:40)=[NAL_delimiter,NAL_header];
    end
end
n=0;
    for i=1:M-3
    if (Video_raw(i)==0 && Video_raw(i+1)==0 && Video_raw(i+2)==1)
        n=n+1;
        B(n)=i+3;
    end
    end
    n=1;
    i=1;
    for j=1:m
        if j==m
            A=1;
            x=B(n);
            x1=x+4;
        else
        A=B(n+1)-B(n)-3;
        x1=B(n+1);
        x=B(n);
        end
        Video_output(i)=bin2dec(Video_NAL_header(j,1:8));
        Video_output(i+1)=bin2dec(Video_NAL_header(j,9:16));
        Video_output(i+2)=bin2dec(Video_NAL_header(j,17:24));
        Video_output(i+3)=bin2dec(Video_NAL_header(j,25:32));
        Video_output(i+4)=bin2dec(Video_NAL_header(j,33:40));

        Video_output(i+5:i+A+4)=Video_raw(x:x1-4);

        i=i+A+5;

        n=n+1;

    end

    fp=fopen('C:\Users\sanju\Documents\Mux-demux\thesis\other softwares\fluffy_NAL_AVS.avs','w');

    fwrite(fp,Video_output);

    fclose(fp);
```

```
global V;

global A;

global Aframelength;

global APES;

global VPES;

global k;

global n;

global q;

global p;

global Vmux_time;

global Amux_time;

global Vcounter;

global Acounter;

global Vnext_begin;

global Anext_begin;

global A_done;

global V_done;

global name;

global TS_number;

global Vframe_number;

global Aframe_number;

video=input('enter the input video sequence'); %The NAL formatted AVS video is taken as the input

fps=input('enter frames per second of the video sequence'); %Frame rate here is 25

fp=fopen(video);

V=fread(fp);

m=0;%m increments by a step of 1 between every successive frames

global Vlocation;

global Alocation;
```

```
for i=1:length(V)-5
   if (V(i)==0 && V(i+1)==0 && V(i+2)==0 && V(i+3)==1) %The condition is used here since NAL delimiter
is 0x00000001 (total of 32 bits)
      m=m+1;
      Vlocation(m)=i;%Vlocation is used to store the start position of each frame
%         Vframetype(m,1)=dec2hex(V(i+5,1));
   end
end
Vlocation(m+1)=length(V)+1;
Vlocation_length=length(Vlocation);
Vframe_duration=1/fps;
audio=input('enter the input audio sequence');
fp1=fopen(audio);
A=fread(fp1);
m=0;
i=1;
% ADTS header------> 72 bits
% First 12 bits syncword -----> "111111111111"---->4095 in dec
% 14th and 15th bit shoould always be zero
% 19th to 22nd bit gives the sampling_frequency_index (4 bits-----> valid
% dec values from 0 to 12)http://wiki.multimedia.cx/index.php?title=MPEG-
4_Audio#Sampling_Frequencies
% 31st to 43rd bit give the aac_frame_length including the header bytes
while(i<length(A))
   temp1=dec2bin(A(i),8);
   temp2=dec2bin(A(i+1),8);
   Asyncword=bin2dec(strcat(temp1,temp2(1:4)));
   Zero_bits=bin2dec(temp2(6:7));
```

```
    if (Asyncword==4095 && Zero_bits==0)
        m=m+1;
        temp=dec2bin(A(i+3:i+5),8);
        Aframelength(m)=bin2dec(strcat(temp(1,7:8),temp(2,:),temp(3,1:3))); % 13 bit value is stored here
        Alocation(m)=i; %Alocation is used to store the start position of each frame
        i=i+Aframelength(m);
    else
        i=i+1;
    end
end
Alocation(m+1)=length(A)+1;
temp3=dec2bin(A(Alocation(1)+2),8);
freq=bin2dec(temp3(3:6));
freq_table=[96000;88200;64000;48000;44100;32000;24000;22050;16000;12000;11025;8000;7350];
% Sampling Frequencies
%
% There are 13 supported frequencies:
% 0: 96000 Hz
% 1: 88200 Hz
% 2: 64000 Hz
% 3: 48000 Hz
% 4: 44100 Hz
% 5: 32000 Hz
% 6: 24000 Hz
% 7: 22050 Hz
% 8: 16000 Hz
% 9: 12000 Hz
% 10: 11025 Hz
% 11: 8000 Hz

% 12: 7350 Hz

% 13: Reserved

% 14: Reserved

% 15: frequency is written explictly


if freq>(length(freq_table)-1)

    disp('Unknown audio sampling frquency');

else

    frequency=freq_table(freq+1);

    Aframe_duration=1024/frequency; %1024 samples per frame for audio

end

global Vframe_number_reset;

global Aframe_number_reset;

if Vframe_duration>Aframe_duration

    Vframe_number_reset=65535;

    Aframe_number_reset=round(65535*(Vframe_duration/Aframe_duration));

else

    Aframe_number_reset=65535;

    Vframe_number_reset=round(65535*(Aframe_duration/Vframe_duration));

end

%Aframe_number_reset and Vframe_number_reset remains fixed as the Video fps

%and audio frequency is fixed.

k=1;

n=1;

q=1;

p=1;

Vmux_time=0;
```

```
Amux_time=0;
Vcounter=0;%4 bit continuity counter
Acounter=0;%4 bit continuity counter max value is 15
Vnext_begin=1;% Used as payload unit start indicator i.e if PES header byte is present in the current TS
packet
A_done=0;%flag to indicate EOS
V_done=0;%flag to indicate EOS
name=[1:54000]; % An array used for file operations
name=dec2bin(name);
TS_number=1; % It calculates the total no. packets
Vid_PES();
while(Vframe_number<=6)
     PID=bin2dec('1111'); % 10 bit value to uniquely indentify Video PES
begin=Vnext_begin;
% if begin==1
%     VTS_num=ceil(length(VPES)/185);%No. of TS packets per single frame
%     VTS_size=1000*(Vframe_duration/VTS_num);% duration of each TS packet in ms
% end
if length(VPES)>185
    data=VPES(1:185);
    Vnext_begin=0; %flag is set to zero indicating that the next TS packet doesnot contain the PES header
    VPES=VPES(186:length(VPES));
    test_frame_1='Vframe';
else
    data=VPES;
    Vnext_begin=1;
    k=k+1;
    test_frame_1='Vframe';
    if k<=Vlocation_length-1
        Vid_PES();
    end
end
test_frame1=Vframe_number;
Transport_Stream(PID,data,begin,Vcounter,test_frame_1);
if Vcounter>=15
    Vcounter=0;
else
    Vcounter=Vcounter+1;
end
end
Vcounter=0;%4 bit continuity counter
Vnext_begin=1;% Used as payload unit start indicator i.e if PES header byte is present in the current TS
packet
Anext_begin=1;% Same as above
Vid_PES();
Aud_PES();
while (A_done==0 || V_done==0) % if both are equal to zero indicates the end of packets
    vmux_chk(TS_number)=Vmux_time;
    amux_chk(TS_number)=Amux_time;
    if((Vmux_time<Amux_time || A_done==1))
        PID=bin2dec('1111'); % 10 bit value to uniquely indentify Video PES
begin=Vnext_begin;
if begin==1
    VTS_num=ceil(length(VPES)/185);%No. of TS packets per single frame
    VTS_size=1000*(Vframe_duration/VTS_num);% duration of each TS packet in ms
end
```

```
if length(VPES)>185
    data=VPES(1:185);
    Vnext_begin=0; %flag is set to zero indicating that the next TS packet doesnot contain the PES header
    VPES=VPES(186:length(VPES));
    test_frame_1='Vframe';
else
    data=VPES;
    Vnext_begin=1;
    k=k+1;
    test_frame_1='Vframe';
    if k<=Vlocation_length-1
        Vid_PES();
    end
end
test_frame1=Vframe_number;
Transport_Stream(PID,data,begin,Vcounter,test_frame_1);
if Vcounter>=15
    Vcounter=0;
else
    Vcounter=Vcounter+1;
end
Vmux_time=Vmux_time+VTS_size; % At the end gives the total duration of Video TS packets
    else
        PID=bin2dec('1110');
        begin=Anext_begin;
        if begin==1
            ATS_num=ceil(length(APES)/185);
            ATS_size=1000*(Aframe_duration/ATS_num);
        end
        if length(APES)>185
            data=APES(1:185);
            Anext_begin=0;
            APES=APES(186:length(APES));
        test_frame='Aframe';
        else
            data=APES;
            Anext_begin=1;
            n=n+1;
            test_frame='Aframe';
            if n<=length(Alocation)-1
                Aud_PES();
            end
        end
        Transport_Stream(PID,data,begin,Acounter,test_frame);
        if Acounter>=15
            Acounter=0;
        else
            Acounter=Acounter+1;
        end
        Amux_time=Amux_time+ATS_size;
    end
if (k>(length(Vlocation)-1) && Vnext_begin ==1)
    V_done =1;
end

if (n>(length(Alocation)-1) && Anext_begin ==1)
```

```
   A_done =1;
end
end
```

```
% PES packet header

% Packet code start prefix   3 bytes  0X00000001

% Stream ID                  1 byte  Unique ID for each audio and video ES

% PES packet length          2 bytes =0 if >65536

% Time stamp                 2 bytes Can be calclulated from

% Vframelength=Vlocation(k+1)-Vlocation(k)

% Data                       Variable size

function Vid_PES()

global Aframe_number_reset;

global Vframe_number_reset;

global Alocation;

global VPES;

global Vlocation;

global APES;

global n;

global q;

global p;

global V;

global A;

global k;

global vid;

global Vframe_number;

stream_id=bin2dec('11000000');

Vframe_length=Vlocation(k+1)-Vlocation(k);

if q<=Vframe_number_reset

   Vframe_number=q;

   q=q+1;

else
```

```
    Vframe_number=1;

    q=2;

end

bin_frame=dec2bin(Vframe_number,16);

Vframe_number_byte1=bin2dec(bin_frame(1:8));

Vframe_number_byte2=bin2dec(bin_frame(9:16));

Vframe_length=dec2bin((Vframe_length+8),16);

% This is including the PES header of 8 bytes

Vframe_length1=bin2dec(Vframe_length(1:8));

Vframe_length2=bin2dec(Vframe_length(9:16));

VPES=[0;0;1;stream_id;Vframe_length1;Vframe_length2;Vframe_number_byte1;Vframe_number_byte2;

V(Vlocation(k):Vlocation(k+1)-1)];

vid(k)=Vframe_number;
```

```
% PES packet header
% Packet code start prefix   3 bytes  0X00000001
% Stream ID                  1 byte  Unique ID for each audio and video ES
% PES packet length          2 bytes =0 if >65536
% Time stamp                 2 bytes Can be calclulated from aac_frame_length i.e Aframelength(m)
% Data                       Variable size
function Aud_PES()
global Aframe_number_reset;
global Vframe_number_reset;
global Aframelength;
global Alocation;
global Vlocation;
global VPES;
global APES;
global n;
global q;
global p;
global V;
global A;
global k;
global aud;
global Aframe_number;
stream_id=bin2dec('11100000');
Aframe_length=Aframelength(n);
if p<=Aframe_number_reset
   Aframe_number=p;
   p=p+1;
else
```

```
    Aframe_number=1;

    p=2;

end

bin_frame=dec2bin(Aframe_number,16);

Aframe_number_byte1=bin2dec(bin_frame(1:8));

Aframe_number_byte2=bin2dec(bin_frame(9:16));

Aframe_length=dec2bin((Aframe_length+8),16);

% This is including the PES header of 8 bytes

Aframe_length1=bin2dec(Aframe_length(1:8));

Aframe_length2=bin2dec(Aframe_length(9:16));

APES=[0;0;1;stream_id;Aframe_length1;Aframe_length2;Aframe_number_byte1;Aframe_number_byte2;

A(Alocation(n):Alocation(n+1)-1)];

aud(n)=Aframe_number;
```

```
function Transport_Stream(PID,data,begin,counter,test)

global TS_number;

global name;

TS_packet=zeros(1,188);

begin_frame=dec2bin(begin);


if length(data)<185

    offset=188-length(data);

    adaptation_field='1';

    header=strcat(begin_frame,adaptation_field,dec2bin(counter,4),dec2bin(PID,10));

    header1=bin2dec(header(1:8));

    header2=bin2dec(header(9:16));

    TS_packet(1:4)=[71;header1;header2;offset];

    TS_packet(offset+1:188)=data;

else

    adaptation_field='0';

    header=strcat(begin_frame,adaptation_field,dec2bin(counter,4),dec2bin(PID,10));

    header1=bin2dec(header(1:8));

    header2=bin2dec(header(9:16));

    TS_packet=[71;header1;header2;data];

end

filename = strcat(name(TS_number,:),'.bin');

    fid = fopen(filename,'w');

    fwrite(fid,TS_packet);

    fclose(fid);

    TS_number = TS_number+1;
```

APPENDIX B

DE-MULTIPLEXING C++ SOURCE CODE

C++ source code for de-multiplexing

```
#include <iostream>
#include <fstream>
#include<string.h>
#include<math.h>
using namespace std;
ifstream::pos_type size;
char *memblock;
char *zbuff;
char name[20];
int flag=0;
int main()
{
  int framenumber;
  int PES_length;
  int streamid;
   int count=0;
   int payload_unit_start;
  int adaptation_field;
  int cont_counter;
  int PID;
  int offset;
  int invalid_packet;
  int vf = 0;
  int af = 0;
  int V[5000][3];
  int A[5000][3];
  int *Video;
int *Audio;
 Video = new int [10000000];
 Audio = new int [10000000];
 int *Vend;
 Vend =Video;
 int *Aend;
 Aend=Audio;
 int remainder;
 char a[16],b[16];
 int ts;
 cout<<"Enter the packet number you want to start with";
 cin>>ts;
 for(int num = ts;num<17339;num++)
  {
   strcpy(a,"");
   int number = num;
   int i = 0;
   while(number > 1)
    {
     remainder = number%2;
     if (remainder==0)
       strcat(a,"0");
     else
       strcat(a,"1");
     number = number>>1;
     i++;
    }
    strcat(a,"1");
 strrev(a);
  strcpy(name,"0000000000000000");
              for (int i=0;i<strlen(a);i++)
              {
                           name[15+i-strlen(a)+1]=a[i];
              }
              strcat(name,".bin");
  //cout<<'\n'<<name;
              //if (num==18619)
              //{
                          // cout<<'\n'<<"************"<<name<<"***********";
  //}
     ifstream file(name,ios::in|ios::binary|ios::ate);
     if (file.is_open())
      {
       size = file.tellg();
//     cout<< size;
       memblock = new char [size];
       file.seekg(0,ios::beg);
       file.read(memblock,size);
       file.close();
 //    cout<<"complete file content in memory";
      }
     else cout << "unable to open"<<"\t"<<name;
    unsigned char k = *memblock;
     if (int(k) == 71)
     {
       invalid_packet =0;
```

```
        k =*(memblock+1);
        payload_unit_start = ((k>>7)&0x1);
        adaptation_field =  ((k>>6)&0x1) ;
        cont_counter = (k>>2)&0xf;
        unsigned char l = *(memblock+2);
        PID = ((k&0x3)<<8)|l;
        if (adaptation_field ==1)
        {
            k =*(memblock+3);
            offset = int(k);
//           cout<<"\n"<<offset;
            memblock =memblock+offset;
        }
        else
        {
            offset = 3;
            memblock =memblock+3;
        }
      }
      else
      {
         invalid_packet =1;
         cout<<'\n'<< " Invalid paket Received";
      }
     if (payload_unit_start ==1)
      {
         if ( (*(memblock)==0) & (*(memblock+1)==0) & (*(memblock+2)==1))
         {
            k = *(memblock+3);
            streamid = int(k);
            k = *(memblock+4);
            unsigned char l = *(memblock+5);
            PES_length = (k<<8|l);
//            cout<<"\n"<<PES_length;
            k = *(memblock+6);
            l = *(memblock+7);
            framenumber = (k<<8|l);
//            cout<<'\n'<<framenumber;
            memblock =memblock+8;
            offset = offset+8;
          }
          else
          {
            invalid_packet =1;
            cout<<'\n'<<"Packet error : no PES header";
          }
        }
       if (PID == 15)
        {
         if (payload_unit_start ==1)
         {
           //cout<<'\n'<<"*******************"<<vf<<"**************"<<name<<"*********"<<num;
            if (vf==1362)
            {
              float temp,temp1;
              int Iframe =0;
        int aframe =0;

        int n = 0;

        int m = 0;

        int p =0;

        int vfnum;

        int afnum;

         float v_playback;

         float a_playback;

         float sync_delay;

         do

        {

            int y = *(Video+V[n][2]+4);

            p = y&0x7;

//              cout<<'\n'<<p;


            if(p == 0x5)

            {

                vfnum = V[n][1];

               // afnum = 441000*(vfnum-1)/(1024*25);


                temp1 = vfnum;

                temp = (44100*(temp1-5)/(1024*25));    //round it

                afnum = temp;

               // cout <<'\n'<<temp;

                if( (temp-afnum)>0.5)

                    afnum =ceil(temp);

                else

                    afnum=floor(temp);
```

```
       // cout <<'\n'<<afnum;
        v_playback=((vfnum-5)*1000)/25;
                                               a_playback=(afnum*(1000)*(1024))/44100;
                                               sync_delay=v_playback-a_playback;
                                               cout<<"Video playback:"<<v_playback<<"\n";
        cout<<"Audio playback:"<<a_playback<<"\n";
        cout<<"sync delay:"<<sync_delay<<"\n";
                                               cout<<"Total audio frames:"<<af<<"\n";
                                               cout<<"end TS packet:"<<num<<"\n";
                                               cout<<"sync. video frame:"<<vfnum<<"\n";
                                               cout<<"sync. audio frame:"<<afnum<<"\n";
                                               cout<<"\n";
 for(int y=0;y<af;y++)                        // add condition if afnum is not found
        {
            if(A[y][1] == afnum)
            {
                Iframe =1;
                m = y;
                y = af;
            }
        }
    }
    else cout<<"";
    n++;
    //    cout <<'\n'<<n<<" "<<m;
}while(Iframe ==0);
 n--;
```

```
            int vdata;
            int adata = A[af-1][2]-A[m][2];              // astart = A[a][2];    aend   = A[af][2]-1;
            ofstream vid;
            ofstream aud;
            count++;
            ifstream z;
            zbuff = new char [117];
            z.open("z.bin");
            z.seekg(0,ios::beg);
            z.read(zbuff,117);
            z.close();
remove("C:\\Program Files (x86)\\Aura4You\\Aura Video Converter\\new_video.avs");
remove("C:\\Program Files (x86)\\Aura4You\\Aura Video Converter\\new_audio.aac");
vid.open("C:\\Program Files (x86)\\Aura4You\\Aura Video Converter\\new_video.avs",ios::out |
ios::binary);
 aud.open("C:\\Program Files (x86)\\Aura4You\\Aura Video Converter\\new_audio.aac",ios::out |
ios::binary);
                              if (Iframe ==1)
            {
                                    for (int i =0; i<117;i++)
               {
                   unsigned char h = *zbuff;
                   vid<<h;
                   zbuff++;
               }
while (n<vf-1)
                              {
                                      vdata=V[n+1][2]-V[n][2];
                                    for ( int i =0; i<vdata-2; i++)
                  {
                                    if (i<=1) { *(Video+i+V[n][2]+2)=0; vid<< char(*( Video +i+V[n][2]+2));}
                              else if (i==2) { *(Video+i+V[n][2]+2)=1; vid<< char(*( Video +i+V[n][2]+2));}
                                    else {vid<< char(*( Video +i+V[n][2]+2));}
                                    }
                                    n++;
                               }
                  vid.close();
                 for (int i =0; i<adata; i++)
                   {
                     aud<< char (*(Audio +i+A[m][2]));
                  }
                  aud.close();
                }

                cout<< "\n one block";
                cout<<"*******************"<<vf<<"***************"<<name;
                              system( "test.bat ");
                vf =0;
                af =0;
                Vend =Video;
                Aend =Audio;

               }
//           else cout<<"\nBuffering .....";
```

```
            V[vf][0]=PES_length-8;
          //  cout<<"\nvideo\n"<<V[vf][0];
            V[vf][1]=framenumber;
          //  cout<<"\n"<<V[vf][1];
            V[vf][2]=(Vend-Video);
          //  cout<<"\n"<<V[vf][2];
            vf++;


        }
//      cout<<'\n';
        for (int i = 1;i<=(188-offset);i++)
         {
             k = *memblock;
            // Video[Vend]=int(k);
             *Vend = int(k);
                              memblock++;
             Vend++;
         }


     // cout<<'\n'<<(*(Video+V[0][2]+4)&0x7);
         }
       else if (PID == 14)
       {
        if (payload_unit_start ==1)
        {
            A[af][0]=PES_length-8;
       //     cout<<"\n audio\n"<<A[af][0];
            A[af][1]=framenumber;
```

```
    //     cout<<"\n"<<A[af][1];
           A[af][2]=(Aend-Audio);
    //     cout<<"\n"<<A[af][2];
           af++;
     }

     for (int i = 1;i<=(188-offset);i++)
      {
            k = *memblock;
            *Aend = int(k);
            memblock++;
            Aend++;
      }
    }
  }
 cout<<'\n'<<"Done";
  return 0;
}
```

Test.bat file

```
cd C:\Program Files (x86)\Aura4You\Aura Video Converter

mencoder new_video.avs -ovc copy -o video_temp.mkv

cd C:\Program Files (x86)\MKVtoolnix

mkvmerge -o nvideo.mkv --default-duration 0:25fps -d 0 -A -S "C:\Program Files (x86)\Aura4You\Aura
Video Converter\video_temp.mkv" -a 0 -D -S "C:\Program Files (x86)\Aura4You\Aura Video
Converter\new_audio.aac"

cd C:\Program Files (x86)\Aura4You\Aura Video Converter

del video_temp.mkv
```

REFERENCES

[1] L. Yu et al, "Overview of AVS-Video: tools, performance and complexity", SPIE VCIP, vol. 5960, pp. 596021-1~596021-12, Beijing, China, July 2005.

[2] W. Gao et al, "AVS- The Chinese Next-Generation Video Coding Standard", National Association of Broadcasters, Las Vegas, 2004.

[3] X. Wang et al, "Performance comparison of AVS and H.264/AVC video coding standards", J. Computer Science & Technology, vol. 21, No. 3, pp. 310-314, May 2006.

[4] L. Yu et al, "Overview of AVS-video coding standards", Special issues on AVS standards, Signal Processing: Image Communication, vol. 24, pp. 247-262, April 2009.

[5] AVS Work Group website

http://www.avs.org.cn/en/

[6] R. A. Burger et al, "A survey of digital TV standards China", IEEE Second International Conference on Communications and Networking in China, pp. 687-696, Aug. 2007.

[7] L. Fan et al, "Overview of AVS Video Standard", in the proceedings of IEEE Int'l Conf. on Multimedia and Expo, ICME '04, vol. 1, pp. 423-426, Taipei, Taiwan, June 2004.

[8] Information Technology – Advanced coding of audio and video – Part 2: Video, The standards of People's Republic of China, GB/T 20090.2 – 2006.

[9] I. E. G. Richardson, "The H.264 advanced video compression standard", II Edition, Wiley, May 2010.

[10] C. X. Zhang et al, "The technique of pre-scaled transform", IEEE Int'l Symposium on Circuits and Systems, vol.1, pp. 316-319, May 2005.

[11] Q. Wang et al, "Context-based 2D-VLC entropy coder in AVS video coding standard", J. Computer Science & Technology, vol. 21, No.3, pp. 315-322, May 2006

[12] H. Jia et al, "An AVS HDTV video decoder architecture employing efficient HW/SW partitioning", IEEE Trans. on Consumer Electronics, vol. 52, pp. 1447- 1453, Nov. 2006.

[13] T. Wiegand et al, “Overview of the H.264/AVC Video Coding Standard”, IEEE Trans. on CSVT, vol. 13, pp. 560-576, July 2003.

[14] GB 20090.2 RTP Payload Format, FG IPTV- 0512, International Telecommunications Union, May 2007.

[15] Information Technology – Generic coding of moving pictures and associated audio: Systems, International Standard 13818-1, ISO/IEC JTC1/SC29/WG11 N0801, 1994.

[16] X. Hu et al, “An efficient low complexity encoder for MPEG advanced audio coding”, in the proceedings of IEEE ICACT conference, pp. 1501-1505, Feb. 2006.

[17] Information Technology – Generic coding of moving pictures and associated audio information- Part 7: Advanced Audio Coding (AAC), ISO/IEC 13818-7, 2006.

[18] Information Technology – Coding of audio-visual objects- Part 3: Audio, ISO/IEC 14496-3, 2009.

[19] M. A Watson et al, “Design and implementation of AAC decoders”, IEEE Trans. on consumer electronics, vol. 46, pp. 819-824, Aug. 2000.

[20] K. Takagi et al, “Conversion of MP3 to AAC in the compressed domain”, IEEE 8th Workshop on Multimedia Signal Processing, pp. 132-135, Oct. 2006.

[21] K. Brandenburg, “Low bit rate audio coding-state-of-the-art, challenges and future directions”, in the proceedings of IEEE ICSP, vol. 1, pp. 1-4, Aug. 2000.

[22] B. J. Lechner et al, “The ATSC transport layer, including program and system information protocol (PSIP)”, Proceedings of the IEEE, vol. 94, no. 1, pp. 77-101, Jan. 2006.

[23] J. Stott, “Design technique for multiplexing asynchronous digital video and audio signals”, IEEE Trans. on communications, vol. 26, no. 5, pp. 601-610, May 1978.

[24] Special issue on Global Digital Television: Technology and Emerging Services, Proceedings of the IEEE, vol. 94, pp. 5-7, Jan. 2006.

[25] Information Technology – Generic coding of moving pictures and associated audio- Part 1: Systems, ISO/IEC 13818-1:2005, International Telecommunications Union.

[26] Z. Cai et al, “A RISC implementation of MPEG-2 TS packetization”, in the proceedings of IEEE HPC conference, pp. 688-691, May 2000.

[27] G. A. Davidson et al, “ATSC video and audio coding”, Proceedings of the IEEE, vol. 94, no. 1, pp. 60-76, Jan. 2006.

[28] X. Chen, “Transporting compressed digital video”, Kluwer, 2002.

[29] W. Zhu, “End-to-End modeling and simulation of MPEG-2 transport streams over ATM networks with jitter”, IEEE Trans. on CSVT, vol. 8, no. 1, pp. 9-12, Feb. 1998.

[30] H. Murugan, “Multiplexing H.264 video with AAC audio bit stream, de-multiplexing and achieving lip synchronization during playback”, M.S.E.E thesis, University of Texas at Arlington, May 2007.

[31] Reference software for AAC encoder - FAAC and AAC decoder- FAAD

www.audiocoding.com

[32]AVS China part 2 video software

ftp://incoming/dropbox/video_software/P2_software/JiZhun Profile/rm52k_r2.zip

[33] FFmpeg software and official web site.

www.ffmpeg.org

[34] Mplayer software and reference web site

http://www.mplayerhq.hu

[35] Reference software for aura video converter

www.aura4you.com

[36] Reference software for Matroska Media Container

www.matroska.org

[37] VLC software and source code website

www.videolan.org

[38] Overview of H.264/ AVC

http://www.vcodex.com/h264overview.html

BIOGRAPHICAL INFORMATION

Swaminathan Sridhar received his Bachelors of Technology degree in Electronics and Communications Engineering from SASTRA University, Tamil Nadu, India in May 2008. He started pursuing his Masters study at the University of Texas at Arlington from August 2008 and received his degree in Electrical Engineering in December 2010 with the specialization being image and video processing. He was a member of the multimedia processing research group headed by Dr. K. R. Rao from August 2008. His interests include research and development in image and video processing, mathematics, digital signal processing and computer networks.