Study of audio AAC encoding and developing an optimized Encoder

Auteur : Heddari, Wafaa
Promoteur(s) : Redouté, Jean-Michel
Faculté : Faculté des Sciences appliquées
Diplôme : Master : ingénieur civil électricien, à finalité spécialisée en "signal processing and intelligent robotics"
Année académique : 2020-2021
URI/URL : http://hdl.handle.net/2268.2/11545

University of Liege - Faculty of Applied Sciences in collaboration with
EVS Broadcasting equipment

Master Thesis

Study of audio AAC encoding and developing an optimized Encoder

Author :
HEDDARI Wafaa

Supervisors:
Pr. REDOUTE Jean-Michel
Mr MANNESBERG Justin
Mr. LEFRANC Frederic

Master in Civil Electrical Engineering: Signal Processing and Intelligent robotic
Academic Year 2020-2021

Abstract

For high-quality, multichannel, and digital audio compression, MPEG advanced audio coding (AAC) and its successors become the most important pillars of the Moving Picture Experts Group (MPEG) family. MPEG AAC offers a wide range of applications, from low bitrate Internet audio to multichannel broadcasting services, thanks to its numerous working modes and configurations. In this thesis, we are going to study the different blocks of an AAC encoder based on the standard ISO/IEC 13818-7 such as the psychoacoustic model, Gain control, Transform block, Spectral processing, Quantization and Entropy coding. Then From the FFMPEG Codec which is based on the standard we will propose some Optimizations regarding the Encoder, like Removal of block switching, Fast MDCT, and Optimized TNS. And then test their implementation on C Code to compare the compilation speed between the original Codec and the optimized one.

Acknowledgement

2020/2021 were the most difficult period for each human being in the world. Students cannot be exception. Covid-19 pandemic had a big impact not only on the world economy but also on mental health of many people, especially young people.

Despite the hard situation that we are living, The University of Liège made a great and a noticeable effort to help students to accomplish their goals. Thus I want to thank all administration members of Uliege, Professors and assistants who allowed me to live a wonderfull experience in Liège in Belgium. It was not just about improving my technical skills, But i had also the opportunity to discover new domain, new culture, and new personalities.

I would like to send my deepest gratitude to my supervisors, Pr. REDOUTE Jean-Michel, Mr MANNESBERG Justin, and Mr. LEFRANC Frederic. Mr. Jean-Michel, for sharing extraordinary knowledge on electronics (especially on bioelectronics) in class, and for always being supportive and in response to all of my questions. And Mr Justin for his welcoming and accepting me as intern at EVS Broadcasting in Liège. The meeting we had each two weeks pushed me always to continue and focus on my goal despite the crisis we are living, i will never forget your last question in every meeting "What can i do for you?", Thank your support and encouragement. I would like to thank also Mr. Frederic for his welcome, his availability, his sense of listening and sharing, and his support to add this new experience and knowledge to my career. Thanks to his guide i could go on on this AAC subject which was my first time to handle this kind of audio compression. I will not forget to thank Mr. THILMANT Bernard who offered me a big help on software matters.

I would also like to thank all the staff I contacted during my internship at EVS, for their warm welcome, and for the time they devoted to me throughout this period of intership to respond to all my questions.

My sincere thanks also go to the members of the jury for their interest in my research by agreeing to examine my work and enrich it with their proposals.

I would like also to thank my room-mates Majda and Siham for their support in every step i had during this journey in Belgium I have learned more than I

could have ever imagined during the Master. I left my family in Morocco, but i found two other sisters here in Liège. Thank you so much.

And Finally, I thank all my family for their love, their support and their patience which has always encouraged me during the preparation of this report. the Separation was hard but thank to their online call, i felt always their presence in every step i took.

Contents

List of Tables

List of Figures

1 Introduction

Digital images, audio, videos or even articles become a necessity all over our daily life, and the rate at which they are posted to social media or loaded to your phone is only getting faster. Improving devices and network infrastructure play a part in this speed, but software techniques like data compression techniques, which reduce the amount of data files, are also significant. In general data compression is a technique for minimizing the size of data. as an example of this data, let us focus on audios. The MPEG-1 project was the first that introduce the compressed digital audio in the late 1980s. MPEG is an acronym for Moving Picture Exerts Group, formed by ISO and IEC. Then, this project was developed to create other layers of MPEG-1, the most famous one which gained more popularity MPEG-1 Layer III, which known by MP3.

1.1 Motivation:

Advanced Audio Coding (AAC) is a lossy digital audio compression standard. Which was created to succeed the MP3 format, and at the same bit rate, it generally produces better sound quality than MP3. As part of the MPEG 2 and MPEG-4 specifications, ISO and IEC have standardized AAC under the standard ISO/IEC 13818-7.
The Goal of this Thesis is to optimize this codec in an optimum way for EVS's product XT-VIA. The XT-VIA is a server built to handle the most stringent demanding live broadcast production needs, combining all new formats and protocols from HD to 8K, SDR to HDR, and SDI to IP.

FIGURE 1: Live production server XT-VIA [1]

FIGURE 2: Backpanel view of XT-VIA [1]

We can list some advantages of this product [1]:

- **Fast server:** Thans to its speed, reliability and loop recording technology, XT-VIA can give an output wih the best quality.
- **Flexibility for UHD or HD operation:** it supports a variety of video formats like UHD-4K, 1080p. It also comes with over 370 tested I/O configurations, allowing to deploy workflows adapted to the exact requirements from the same server.
- **Live HDR production:** using its built-in Multiviewer,XT-VIA supports HDR at all resolutions and can convert HDR to SDR, avoiding the need for further monitoring.
- **Advanced connectivity:** Thanks to certified SMPTE 2110, PTP, and NMOS standards, the server supports hybrid SDI/IP communication and ensures full IP interoperability with other systems.

The characteristic of XT-VIA are given as the following [1]:

- **Channels:**
  - 16+ch FHD and HD (720p, 1080i, 1080p)
  - 6ch UHD-4K (2160p) including integrated upscale from 1080p
  - 2ch UHD-8K Supported on special edition (4320p)
- **Codecs:**

  - UHD-4K: XAVC-Intra, DNxHR
  - HD/FHD: XAVC-I, AVC-I, DNxHD or ProRes
  - PROXY: MJPEG or H.264
- **IP I/O:**
  - 10G Ethernet SFP+
  - 100G as an option
  - ST2022-6, ST2022-7, ST2022-8, ST2110 (-10, -20, -21, -30, -40)
  - NMOS IS-04, IS-05, EMBER+
  - PTP
- **SD I/O** :
  - 1.5G-SDI, 3G-SDI and 12G-SDI selectable in software settings
- **Audio:**
  - 192 uncompressed audio tracks
  - Embedded, AES and MADI support
- **Storage:**
  - Internal storage of 18TB
  - Expandable to 54TB
  - Recording capacity of up to 130 hours of UHD-4K
  - Hot swap storage
- **Embedded Multiviewer:**
  - 2 external inputs
  - 4 individual outputs
  - Tally support
  - 16 different layouts
  - Conversion from HDR to SDR

1.2 Organization

The EVS RD team were trying to optimize the AAC codec in order to improve the XT-VIA product regarding its audio compression, and that was my task in this internship. I had to study first the standard ISO/IEC 13818-7 to present a simplified description of this standard. Then, after studying the FFMPEG Library (which was also based on ISO/IEC 13818-7), i had to propose optimization we can implement to have a faster codec with the best quality.

Thus in this thesis's report, we will start by defining what do we mean by digital audio as a first thing, Then explaining the theoretical aspects of the encoder of an AAC compression (based on ISO/IEC 13818-7). After that we will describe the most important blocks described in FFMPEG : Psychoacoustic Model, MDCT and TNS, defining the different functions used in the code of these blocks. Then it comes the Optimization part, where we will show and explain the proposed optimizations, in particular Removing switching block, Fast MDCT and optimized TNS. And the last chapter of this report will be dedicated for the results, where we will present the results of the test done on different audio samples to compare the computational speed of the original code and the optimized one.

1.3 Used platform

As it will be explained in the Result's chapter, we used VISUAL CODE STUDIO as a software to test and modify the C codes and we compiled under Linux terminal. VS code allows you, from its debug system, to follow every step in the compilation of the code, in this way we were able to understand the flow of the encoder of FFMPEG, which makes our task more easier despite the complexity of the code.

2 Presentation of the host Company : EVS

EVS Broadcast equipment is a Belgian company specified in live video technology for broadcast and media production. After its foundation in 1994 by Pierre Lhoest and Laurent Minguet [1], EVS introduce to the world Live Slow-Motion system which becomes after a standard for all broadcast sporting events across the world. EVS started by developing disc recorders (for digital recording of pictures) to offer this technologies to clients in the television industry and broadcasters.
In 2000, EVS invested in developing video broadcasting systems and digital cinema by introducing a high recorder prototype for NHK (Japanese television station) in the context of the Olympic Games in Sydney. Other products and solution were designed and developed by EVS such as [1] :

- XT-VIA : A video sever that allows to record, control and play clips.
- **XS:** A four channel video production server, designed in 2009 to increase the speed of production of pre-recorded studio and near live shows
- **VIA Xsquare :** Production teams can use VIA Xsquare as a single point of entry to modify, orchestrate, and monitor media files as they move through the live workflow from ingest to shared storage and delivery.
- **DYVI:** a next-generation video switcher designed in 2016
- **PMZ:** PMZ is a cutting-edge virtualization platform built exclusively for OB trucks and short-term events and designed for high-demand situations,

FIGURE 3: XT VIA Live production server [1]

FIGURE 4: Via Xsquare Centralized media workflow management [1]

FIGURE 5: Virtualization Processing Module for Mobile Setups PMZ

Besides its growing in the side of developing new technologies, EVS tried to globalize its brand all over the world, by opening two offices in 1997 in the United State and Hong Kong in order to markets its products in the American and Asian continents. After that, in late of 1998 EVS opened an other subsidiary in France to handle the development in the World cup markets.Then it comes to Italy and United Kingdom, when EVS set up other offices there in 1999

The history of this company over 25 years, was one of the reason why i choose to analyse it, Morover it will help me to understand more the methodology of this company as being an intern student there. Thus, this assignment will help me to accomplish two goals at the same time, first, practicing the SWOT Analysis on a company that is maintaining its growing even the threats on the media field, second, to understand more this company where i am doing my internship.

3 Theoretical Background

Before starting developing the project and explaining its different aspects, in this section we are going to explain first all the theoretical concepts we will be using at each step in this project.

3.1 Introduction to Digital Audio

the sound is analog by its nature, which has some mechanical, electrical and magnetic characteristics that can be used to record it as an analog audio using microphones by converting continuous variations in sound pressure into continuous variations in electrical voltage. This method had proven some weakness, with the fact that with analog recorders we cannot differentiate between unwanted and wanted signals, which can increase noises and distortions in the output signal, without being able to detect the errors.
Here where it comes the use of digital recorders. In order to have a digital representation of the sound, the continuous analog sound must be converted to a non-continuous stream of numbers, that's mean the values between samples will be thrown away.
Mathematically, the continuous function of the analog sound will be converted to a discrete function as shown in the Figure bellow

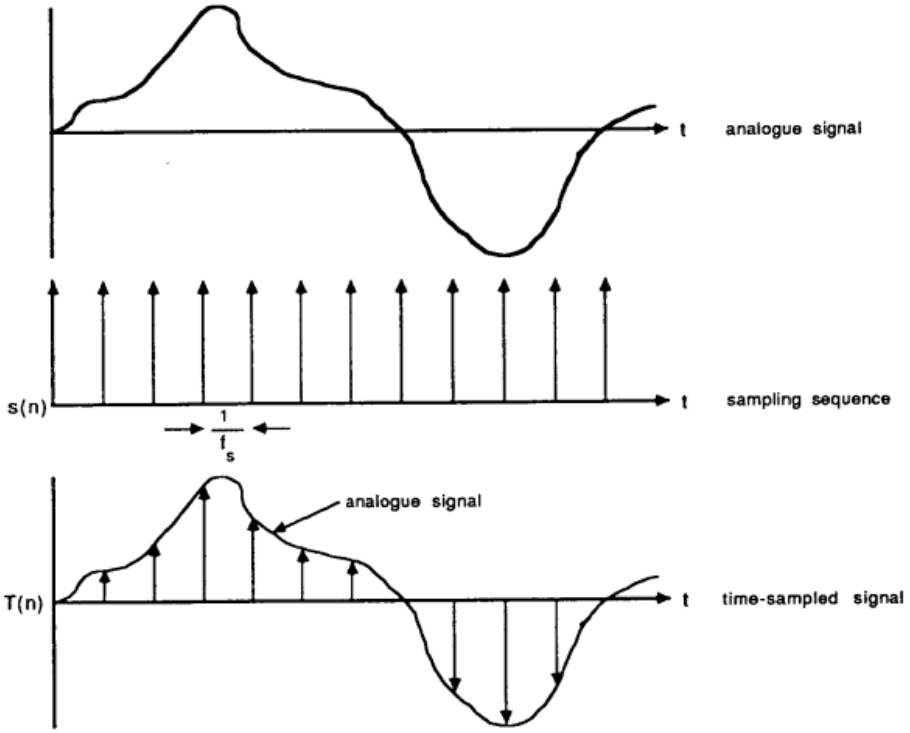

FIGURE 6: Process of sampling in time domain [5]

The analog signal ϕ is sampled by a uniform sequences of impulses s(n) which has the mathematical formula as :

$$s(n) = 1 + 2\sum_{k=1}^{\infty} cos(2\pi k f_s t) \tag{1}$$

And the sampling is done by the multiplication of the signal ϕ and the sampling function s(n) :

$$T(n) = \phi.s(n) \tag{2}$$

Thus we got the expression of the time sampled sequence :

$$T(n) = \phi + 2\sum_{k=1}^{\infty} \phi cos(2\pi k f_s t) \tag{3}$$

Figure 7 shows the sampling in frequency domain, with the term $\phi cos(2\pi k f_s)$ represent amplitude modulated carriers, and by assuming that the input signal was bandlimited to f_c Hz where $f_c < f_s/2$, to prevent the aliasing distortion.

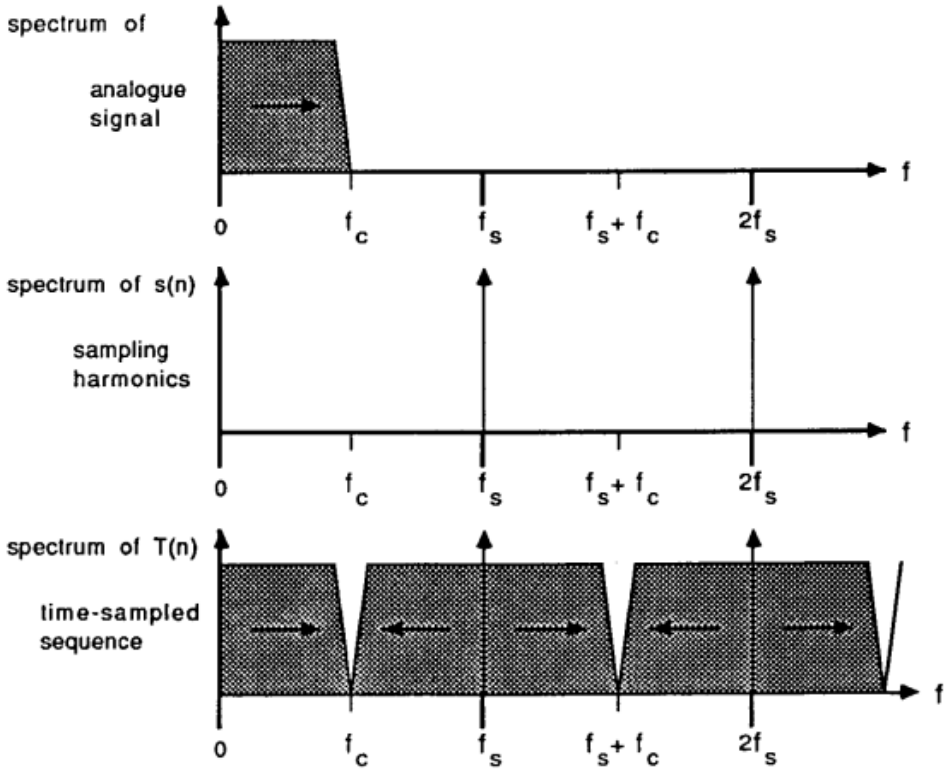

FIGURE 7: Process of sampling in time domain [5]

Then it comes the quantization step. As in the sampling step we convert the time varying voltage into a sequence of real numbers, the quantization replaces each real number with an approximation from a finite set of discrete values which represent a fixed-point word.

The process of this quantization can produce some noise-like residue or even a non-linear distortion, to reduce the impact of there effects we should concentrate on a uniform amplitude quantization as well as defining what we call Signal-To-Noise ration (SNR) to know the quantization distortion spectrum's effect on sampling.
The figure 8 shows the transforming process which is based on stair-case transfer characteristic.

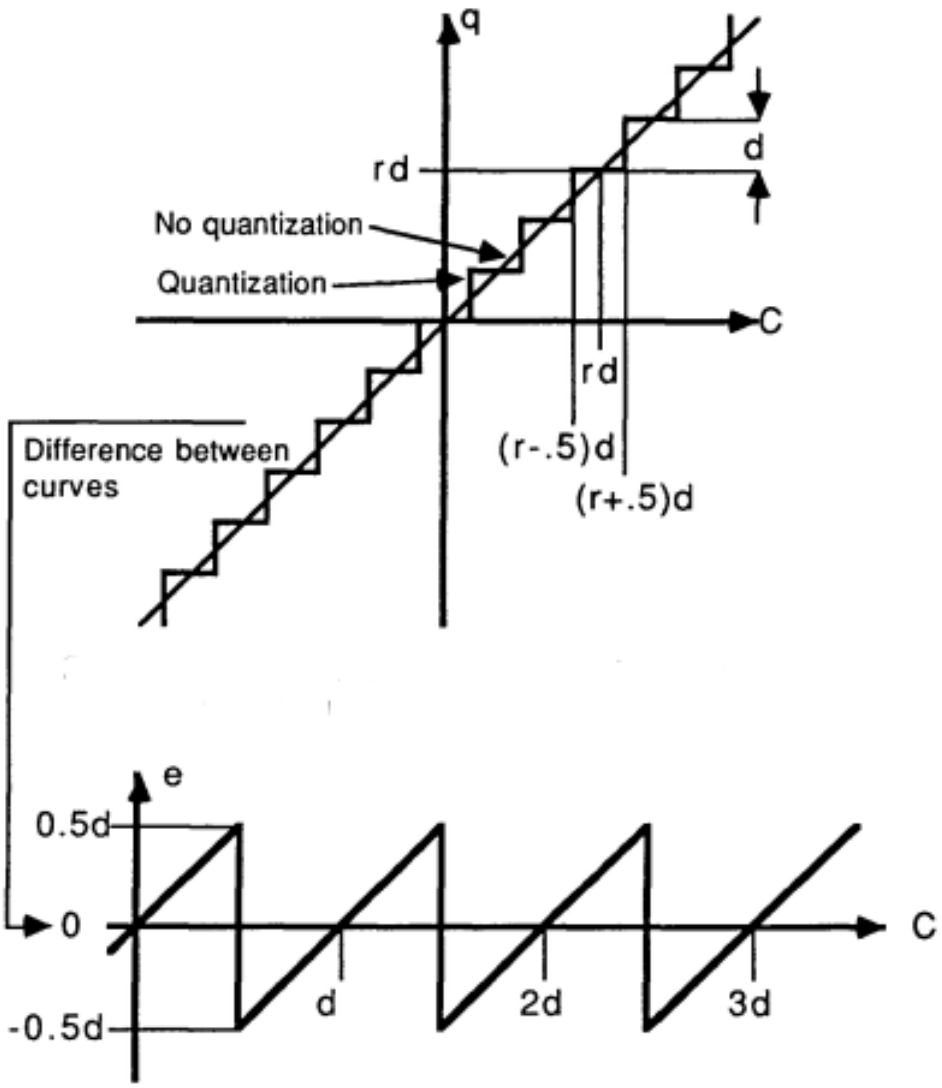

FIGURE 8: Uniform quantization transfer characteristic (top) and the quantization distorsion as a function of level signal(down) [5]

As explained in [5] ans as shown in the top curve of figure 8 the characteristic here works as a look-up table, allowing a signal in the range (r - 0.5)d to (r + 0.5)d to be approximated to a new level rd where r is an integer and d is a constant quantization interval for uniform quantization. As a result, the quantization distortion e ranges from -d/2 to d/2.

the down curve of figure 8 illustrates the quantization distortion's periodic behavior presented as a function of signal level where we can deduce the SNR.

$$SNR = 10log_{10}[\frac{meansquaresignal}{meansquarenoise}] = (6.02N + 1.76)dB \tag{4}$$

Where N represent the number of bit of the quantizer.

3.2 Audio Compression Techniques

There is two categories of compressions, Lossless and lossy one. in the first one no data is lost, what means that when the file is uncompressed we retrieve the exact original file. However in the case of lossy type, we encode an approximation of the original signal, by keeping just the needed information which makes its compression ratio more higher than lossless compression one. Moreover, because the difference in quality between the two categories is not really noticeable, thus almost all compression techniques are lossy.

3.2.1 Wave form coding

One from the most popular techniques of waveform coding is Pulse Code Modulation (PCM), to code the raw digital audio data. It is a method used most frequently in digital audio systems. However it becomes a compression technique when it used with μ-Law or A-Law.
We start by filtering out first the higher frequency component of the analog signal to facilitate the downstream to convert the signal. we defined a bandwidth (of 4000 Hz in general) to prevent aliasing.
The next step is sampling the filtered signal at a constant sampling frequency by using the original signal to modulate the amplitude of a pulse train that has a constant amplitude and frequency which will be called later on the sampling frequency. To determine this sampling frequency, the Nyquist criterion should be verified which states that the sampling frequency should be at least twice the highest frequency of the original input analog voice signal.
Then we digitize the signal by the Quantization as described in the previous section, it's the process of converting each analog sample value into a discrete value that will be assigned a unique digital code word.Each input sample is assigned to a quantization interval that is the closet to its amplitude height, and each quantization interval is assigned to to a discrete value in the form of binary word. This generation can create an error called the Quantization Noise which is equivalent to the noise that impact the signal-to-noise ratio :

$$SNR = 20log_{10}\frac{V_s}{V_n} \tag{5}$$

Where V_s is the input signal, and V_n is the noise level. The higher the SNR the better quality sound, however low signals have small SNR, So to improve the

sound quality at low frequencies, we proceed Companding by compressing the analog signal at the source, and expanding it to its original size where it comes the name "Compading" that refers to combining the two steps. the compression process here is logarithmic, the input signal samples are compressed into a logarithmic segments when each segment is quantized and coded with a uniform quantization. We can use two possible standards for companding : the A-Law and the μ-Law, that was designed for use in telephony at first, they can compress 16-bit linear PCM data down to 8-bits of logarithmic data.

- A-Law Compander

This compander is defined by the following equation :

$$F(x) = \begin{cases} \frac{A.|x|}{1+ln(A)}, & 0 \leq |x| < \frac{1}{A} \\ \frac{sgn(x).(1+ln(A|x|)}{1+ln(A)}, & \frac{1}{A} \leq |x| \leq 1. \end{cases} \tag{6}$$

Where the A is the compression parameter (87.7 in Europe), and x is the normalized integer to be compressed.

- μ-Law Compander

This one is defined by :

$$F(x) = \frac{sgn(x).(1 + ln(1 + \mu|x|)}{ln(1 + \mu)} \tag{7}$$

With $0 \leq |x| \leq 1$. m is the compression parameter and x is the normalized integer to be compressed.

There exist more complex techniques of wave form coding such as the Differential Pulse Code Modulation (DPCM) which is designed to compute the differences between input sample signals are minimal, and then transmit this difference instead of the entire signal, thus the number of bit required for transmission is reduced.

3.2.2 Introduction to MPEG

MPEG, or Moving Picture Exerts Group, was created by ISO and IEC in 1988, it describes the standard of an audio and video compression and its transmission. The MPEG standard is also a lossy compression technique which compress audio signal and remove the unwanted signals produced in the communication path. The following figures show the MPEG Encoder/ Decoder Diagrams:

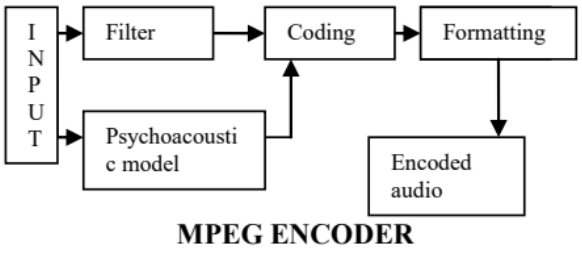

FIGURE 9: MPEG Encoder Diagram [6]

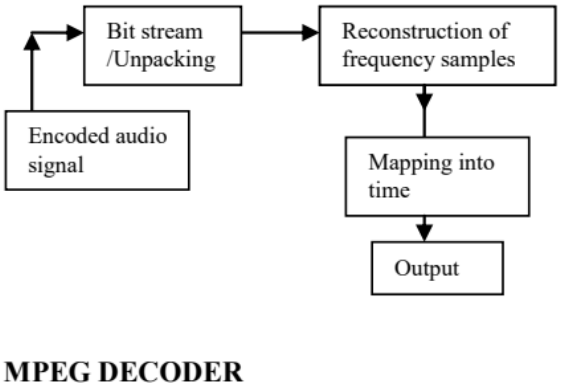

FIGURE 10: MPEG Decoder Diagram [6]

This is the basic audio compression method of an MPEG encoder/ decoder, first the input audio signal passes through a filter bank, to divide the input signal into multiple sub-bands, and then the outputs of this filter bank are passed simultaneously through a psychoacoustic model to apply the masking and and define this signal-to-mask ratio, then the output of this model will be used by the noise allocation block to decide the number of code bits will be used for the quantization process, finally after the quantization step, the sample will be transformed into decodable bit stream.
The MPEG decompression process is done by reversing the formatting, and then reconstructing the quantized sub-band values to be transformed later into a time-domain audio signal.

the most famous technique of this MPEG technology was MP3 which was made in 1999, then a numerous of layers and formats were developed.

- **MPEG-1**

MPEG-1 is the first format of MPEG standard which was created in 1988 and adopted by ISO/IEC IS 11172 in 1992. It was designed to be used in multiple application. This MPEG-1 consistes of three operating layers, with Layer-3 is the most complex mode which has the highest quality and low bit-rates (around 128 kbit/s for a stereo signal). The following figure describe the MPEG-1 Layer-3 Encoder:

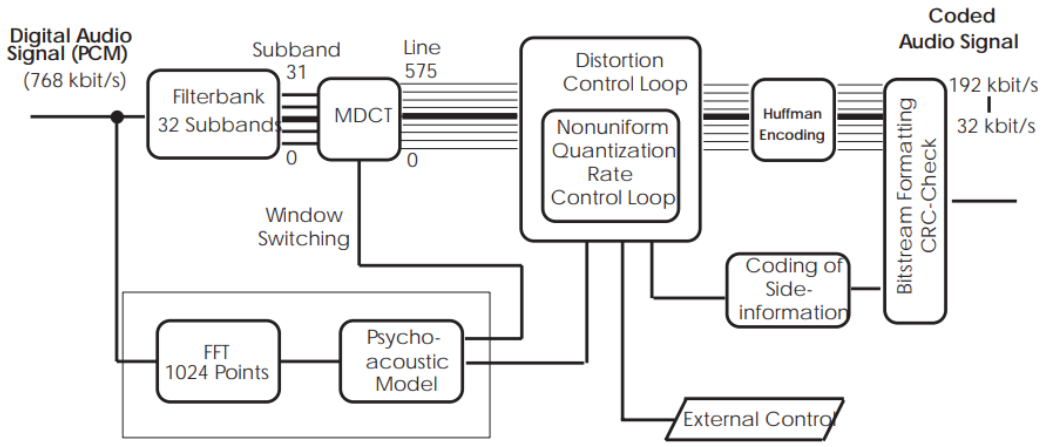

FIGURE 11: Block Diagram of an MPEG-1 Layer-3 Encoder [4]

The input signal passes through a filter bank block to decompose it into sub-sampled spectral components. This filter is hybrid since it's built by cascading 2 filter banks, a polyphase filter bank and a Modified Discrete Cosine Transform (MCDT), this architecture provide the same frame sizes as in Layer-1 and Layer-2.
As for the perceptual model, it is used to compute values for the masking threshold for each coder partition, and that by using either a separate filterbank or combining the the calculation energy values for the masking calculation, and the main filter bank.
The quatization process of MPEG-1 Layer-3 is done by the way that larger values are coded with less accuracy, and there is also a noise shaping process inside this process to keep the quatization noise bellow the masking threshold. the output values of the quantizer are coded by Huffman coding which works on a pairs quadruples.

- **MPEG-2**

MPEG-2 Advanced Audio Coding (AAC) (1997) is the second generation audio coding scheme for generic coding of stereo and multichannel signals, that has a capability of up to 48 main audio channels, 16 low frequency effects channels and 16 data stream. AAC follows the same concept of MPEG-1 Layer 3, however in addition there is new coding tools and details, such as a filter bank with a higher frequency resolution, and a better stereo coding. AAC is about 30% more efficient than MPEG-1 concerning the bit-rate.

3.3 AAC Encoder

The MPEG-2 AAC is a part of the MPEG-2 standard (ISO/IEC 13818-7) that was finalized in 1997. This version provides three different profiles that can be chosen according to the complexity and quality requirements.

- Main profile :
  This profile provide the best audio quality, however both memory requirement and computational complexity are much higher than the other profiles, the reason is that we enable all coding tools except the gain control (these tools will be explained later in this section).
- Low complexity profile :
  As Figure 12 shows, this profile is less complex than the main one, it does not use the gain control and prediction tools, and concerning the TNS filter, it has a lower order than the first one. Thus, the requirement memory and computational complexity are less comparing to the main profile, however, the reconstructed sound quality is not obvious.
- Scalable sampling rate profile :
  As its name refers to, an AAC with this profile can provide a frequency scalable signal. The gain control block here is activated, and concerning the other tools, they are functioning as the previous profile (low complexity profile) that means the requirement and the computational complexity is lower than the main profile.

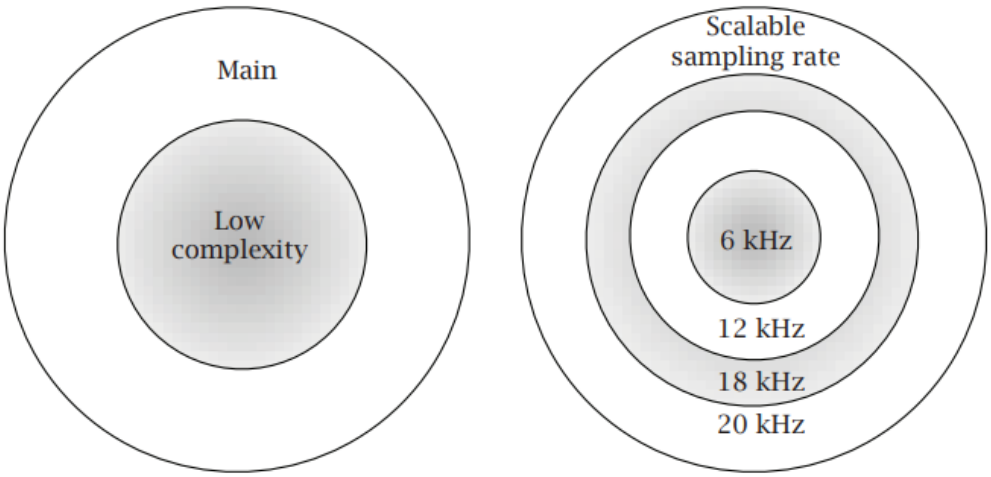

FIGURE 12: The three profiles of MPEG-2 AAC [9]

As Figure 13 shows, First, the input signals can be pre-processed if it's necessary, by being down-sampled to have a better quality regarding the limited bit

budget. Then depending on the profile we chose, the gain block can be activated or not. After that, we transform the signal data from the time domain to the frequency domain in order to obtain the spectral data. Then we remove all the irrelevant information from this data in the spectral processing block. Finally we apply the quantization and a noiseless code to have a compressed spectral data that will be multiplexed in in the bitstream along with all the information generated at each block of the encoder. Moreover, during all this encoding process, the output of the psychoacoustic model is used to control each block.

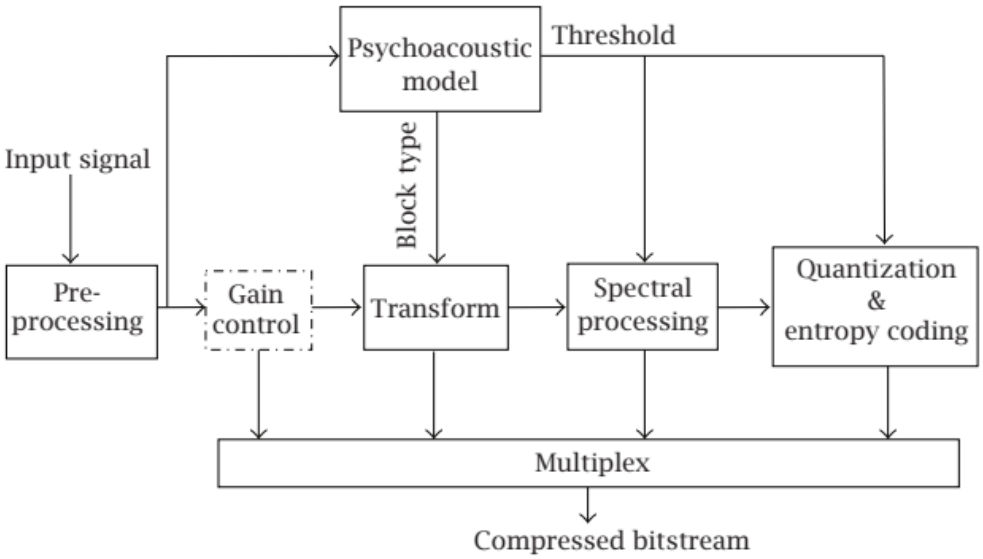

FIGURE 13: The block diagram of the AAC Encoder [9]

Now let us describe each block of this encoder diagram.

3.3.1 Psychoacoustic model

In order to achieve a significant reduction in bitrate while keeping a high audio quality, the psychoacoustic model is used by exploiting the limitations of perceptions and identifying which parts of digital audio signal can be removed. It is known that the human ear can hear sounds in the range of 20Hz and 20KHz, but this range can decrease with age. Moreover the intensity range of audible sounds can change from person to person, or due to age, gender and also the frequency. The following graph shows how the hearing threshold can be dependent to the frequency:

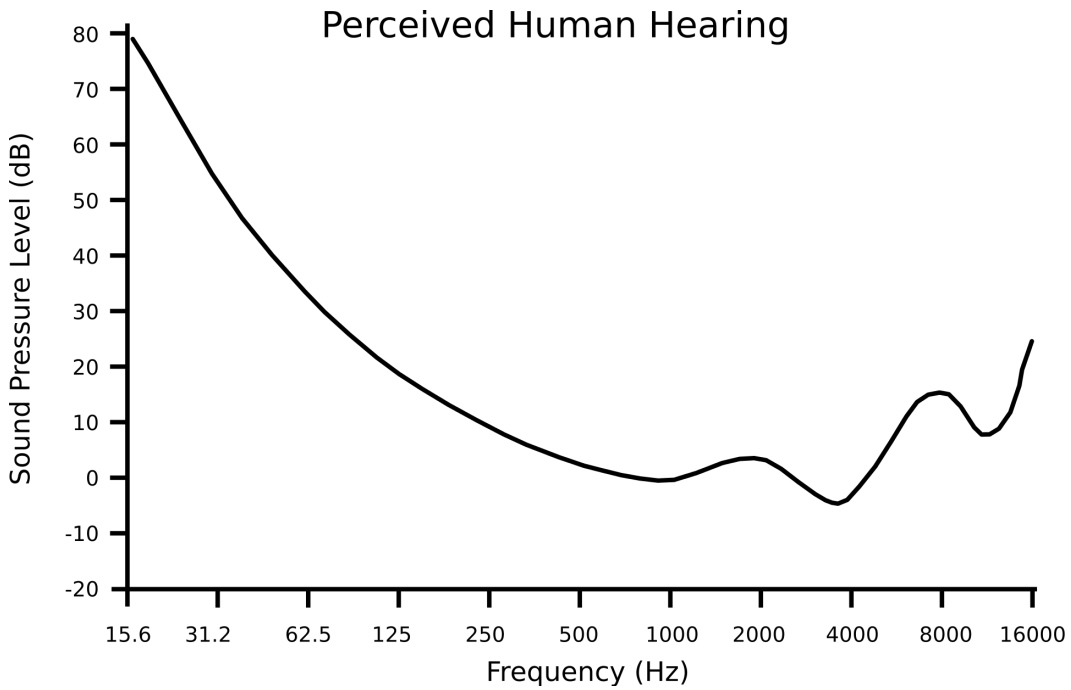

FIGURE 14: Perceived Human Hearing

The main idea of the psychoacoustic model is to use psychoacoustic effects such as masking, which means that louder sounds (Masker) tend to suppress the weaker sounds(Prob), one example to illustrate this phenomena is how a bird sound can be drown by the sound of a car passing by. There exists two categories of such a masking effects :

- **Simultaneous masking**
  The simultaneous masking or spectral masking is illustrated in which an audible sound in a specific frequency becomes inaudible due to the presence of masker at this frequency (that can be a tone or narrow banded noise). As figure 15 shows, the dashed line represent the hearing threshold in quite while the solid line show the masking threshold with the presence of a masker signal what makes the weaker signals at neighboring frequencies become inaudible.

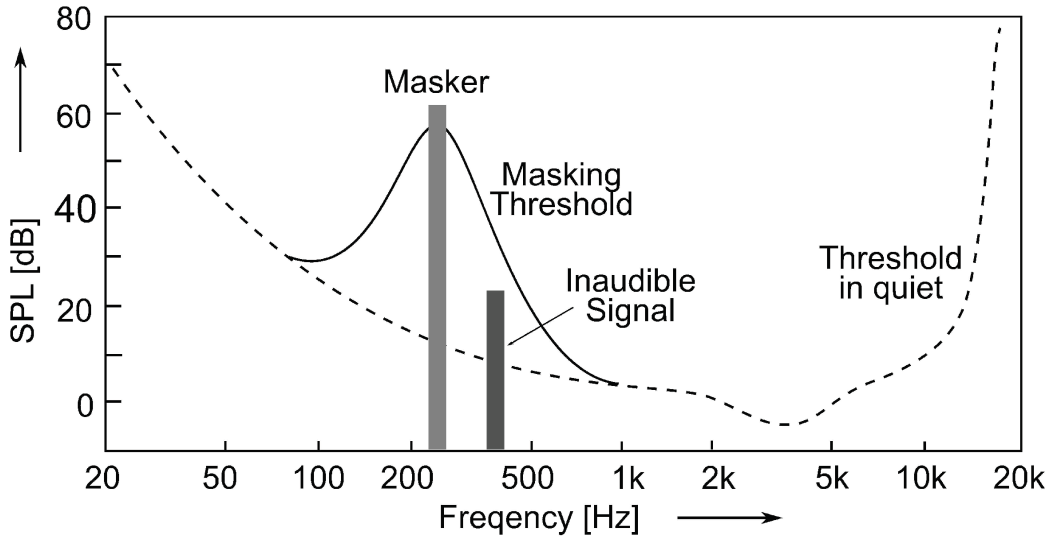

FIGURE 15: Spectral Masking [6]

- **Temporal masking**
  As its name refers to, it is masking in time domain while the masker and the prob are not present in the same point in time. There is three ways of masking with this category:

  1. Pre-Masking : the perception of soft sounds is masked by subsequent louder signal before it turned on. Pre-masking can only occur about 20 ms before the masker sound starts.
  2. Simultaneous Masking : the sound here is masked while the masker sound is playing.
  3. Post-Masking : Here the sound is masked after the masker is turned off, it occurs within 200 ms, after these 20 ms we return the the typical hearing threshold in quiet behaviour.

  Figure 16 illustrate the phenomena of Temporal masking category:

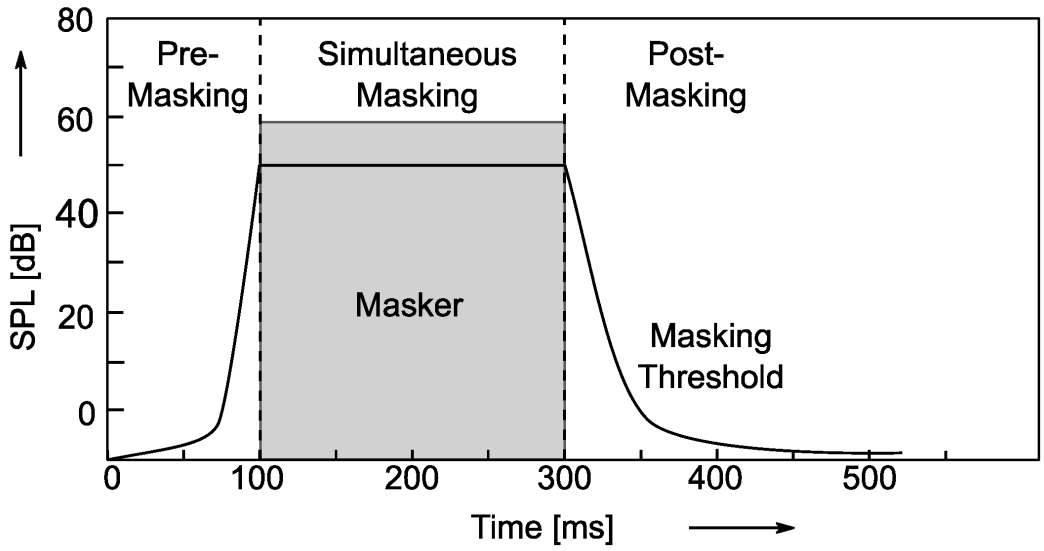

FIGURE 16: Temporal Masking [6]

In general, in a psychoacoustic model we try to compute the maximum distortion energy which is masked, what we call it a Threshold. As the standard ISO/IEC 13818-7 (related to Advanced Audio Coding) describes, we need three main input to generate this threshold [3]:

1. the shift length (called also iblen)
2. the newest iblen sample of the signal with the samples delayed for each FFT type.
3. the sampling rate

To output then from this psychoacoustic model :

1. a set of Signal-to-Mask ratios and thresholds
2. delayed time domain data to be used by the MDCT
3. Block type for the MDCT
4. Number of bits should be used for encoding in addition to the average available bits

3.3.2 Gain control

This tool, which is activated only in the SSR profile, outputs a gain-control data and a gain controlled signal while receiving the input time domain signals ans window sequence. it includes three different blocks, a PQF, a gain detectors,

and a gain modifiers.

- **PQF (Polyphase Quadrature Filter:**

This is the first block in the gain filter tool, which divide the input signal into four equal width frequency bands, with the coefficients of each band are given by :

$$h_i(n) = \frac{1}{4} cos(\frac{(2i+1)(2\pi+5)\pi}{16} Q(n), \qquad 0 \leq n \leq 95, \quad 0 \leq i \leq 3 \qquad (8)$$

with Q(n) = Q(95-n), and $48 \leq n \leq 95$

- **Gain Detector**

This block produces the gain control data which consist on the number of gain changes, the index of gain change position, and finally index of gain change level. There is in total eleven regions where the detection is done, one region for ONLY-LONG-SEQUENCE, two regions for LONG-START-SEQUENCE and LONG-STOP-SEQUENCE and finally eight for EIGHT-SHORT-SEQUENCE. Once we divide the samples in each region into subregions, we select the peak value in these subregions. then we calculate the ratio between this selected value and the value of the last subregion. after that, we compare this ratio with 2^n (n is an integer between -4 et 11), so that we can detect thesesubregions as the gain change points of the signals, and the position data will be the number of this detected subregion, and the exponent of the ratio is set to be the gain data.

- **Gain Modifier**

The gain modifier should know which band of the four sub-band needs amplitude adjustment by controllig the gain of each signal band and applying the GMF (Gain Modificattion Function) to the corresponding band signals.

3.3.3 Transform

Before we move to the spectral processing, The input signals must be transformed from time domain to their time-frequency representations. This transformation is done with what we call it MDCT (Modified Discrete Cosine Transform) after modulating the signals with the appropriate window function, each block of time samples can have either a length of 2048 samples or 256 samples.

- **Window function and Block switching**

Basically a window function is a function that has 0 value at the begin and the end of the length of the input array samples, and 1 at the half of length. Figure 17 illustrate an example of a window function

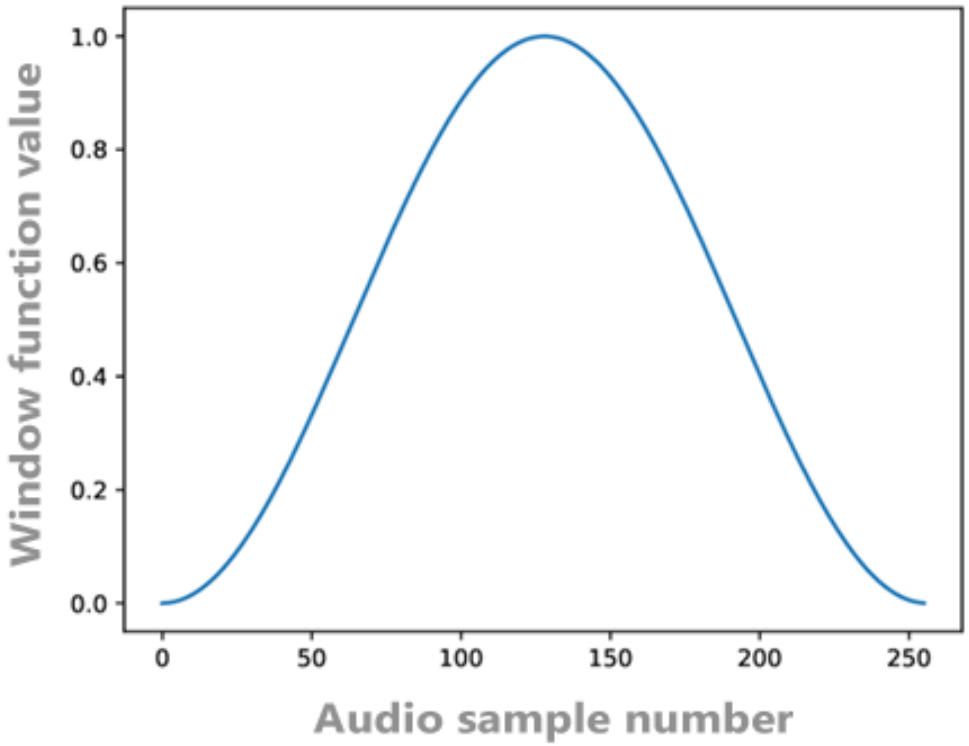

FIGURE 17: Window function

The reason why we apply this function is that while making a lossy audio data compression we remove some information by masking, so we produce some audible noise, and sometimes during this process we got a clearly audible noise at the border of the block of samples. Thus, to eliminate these noises we eliminate block borders by multiplying the block samples with an appropriate window function.
We proceed with two length blocks 2048 (Long blocks) and 256 (Short blocks) to avoid the preecho effect that is caused by transient signals, that's why we encoded these type of signals with shorter transform blocks. However we got inefficient coding for stationary signals with this size of blocks, the reason why we adopt two blocks length and allow block switching when different signal type is detected. According to the block type, we select an appropriate window

shape and we apply it to the second half of the window function, knowing that the first half use the window shape from the preceding frame.

- **MDCT**

After performing the windowing,we proceed the conversion to the frequency representation by MDCT (Modified Discrete Cosine Transform), each block of time samples (either with length 2048, or 256 samples) gathers 50% of old signals from the previous frame, and 50% of new one. The analytical expression of MDCT is given by :

$$X_{i,k} = 2.\sum_{n=0}^{N-1} z_{i,n} cos(\frac{2\pi}{N}(n+n_0)(k+0.5)), \qquad for\ 0 \leq k < N/2 \qquad (9)$$

Where :
$z_{i,n}$: windowed input sequence
n : sample index
k : spectral coefficient index
i : block index
N : window length of the transform window based on the window sequence value
n_0 : (N/2 +1)/2

3.3.4 Spectral processing

In order to prepare our data for the quantization block, we proceed first what we call a Spectral processing that consists in four differents coding tools : Temporal Noise shaping (TNS), Intensity stereo (IS), Prediction, and Mid/Side (M/S) tool.

- **Temporal Noise Shaping**

TNS is one of the newest tool that was introduced in AAC technique in order to improve the coding performance and the speech quality at low bit-rates. This tool helps to avoid the effect of the temporal mismatches between masking thresholds and the quantization noise that can happen especially with transient signal, what will create a preecho effect. So basically we want to keep the quantization noise under the time-dependent masking threshold, we achieve that with this TNS tool by doing a frequency domain prediction while adopting

LPC filter (Linear Predictive Coding)
The following steps describe how we can apply TNS on one window of spectral data :

1. Choosing the target frequency :
   in AAC only necessay frequency regions have active TNS module, this decision depends essentially on the chosen profile and the sampling rate.

2. LPC calculation :
   We can use the standard LPC procedures that uses the Levinsion Durbin algorithm. Basically, the LPC determines the coefficients of a Finite Impulse Response Filter by predicting the next value in a sequence from the current and the previous value inputs. This Linear Predective analysis is based on the transfer function :

$$H(z) = \frac{1}{A(z)} = \frac{1}{1 - \sum^{p} a_k.z^{-k}} \tag{10}$$

   Where $a_k|(1 \leq k \leq p)$ are the predictor coefficients and p is the order of the filter, this order will be the maximum permitted order of the noise shaping filter (TNS-MAX-ORDER) and it depends the profile we have chosen for our encoder. If we transform the equation 10 to the time-domain we got :

$$s'(n) = \sum_{k=1}^{p} a_k.s(n-k) \tag{11}$$

   Where s'(n) is the predicted value of the speech signal based on the previous value s(n). Thus the idea of LPC is finding the coefficient a_k to get the closet approximation to the speech samples. this prediction is based on the minimizing the error between the predicted value s'(n) and the real value s(n):

$$e(n) = s(n) - s'(n) \tag{12}$$

$$E = \sum_{n} e^2(n) \tag{13}$$

   the equation 13 shows the summed error E over a finite window of length N. The minimum value of E is when the derivative is zero, thus we got a set of p equations:

$$\begin{bmatrix} r(0) & r(1) & r(p-1) \\ r(1) & r(0) & r(p-2) \\ \dots & \dots & \dots \\ r(p-1) & r(p-2) & r(0) \end{bmatrix} \times \begin{bmatrix} a_1 \\ a_2 \\ \dots \\ a_p \end{bmatrix} = \begin{bmatrix} r(1) \\ r(2) \\ \dots \\ r(p) \end{bmatrix} \tag{14}$$

Where $r(i)$ is the autocorrelation coefficients computed as :

$$r(i) = \sum_{m=0}^{N-1-i} s(m).s(m+i) \tag{15}$$

The Levinson-Durbin algorithm solves the p^{th} order system of the liner equation $R.a = b$ by using the autocorrelation coefficients r(i) to compute the LP filter coefficients a_i, and that by solving the the following set of equations :

$$\sum_{i=1}^{p} a_i.r(|i-k|) = r(k) \tag{16}$$

The Algorithm 2 shows the pseudo-code of the Levinson-Durbin Principle

[h] [1] /* *input data* */ $R[i] \leftarrow$ *autocorrelation coefficients* $A[i] \leftarrow$ *filter coefficients* $k \leftarrow$ *reflection coefficients* $Alpha \leftarrow$ *prediction gain* /* *initialization* */ $A[0] \leftarrow 0$ $K \leftarrow -R[1]/R[0]$ $A[1] \leftarrow K$ $Alpha \leftarrow R[0] * (1-k^2)$ $i \leftarrow 2ToM$ $S \leftarrow SUM(R[j] * A[i-j]; j = 1, i-1) + R[i]$ $K \leftarrow -S/Alpha$ $An[j] \leftarrow A[j] + K * A[i-j]$ /**for j=1 to i-1 where An[i] = new A[i]*/* $An[i] \leftarrow K$ $Alpha \leftarrow Alpha * (1-K^2)$

3. Quantization of reflection coefficients:
   Once we got the expected prediction gain and the reflection coefficients, we compare the prediction gain g_p to a certain threshold (usually fixed at t=1.4). If g_p does not exceed the threshold t, we do not use TNS, then the TNS-Data-Present bit is set to zero. In the other case we continue TNS processing and we choose 4 bits ($coef\_res$) to quantize the reflection coefficients by the following pseudo-code: [1] $iqfac \leftarrow ((1 << (coef\_res-1))-0.5)/(\pi/2)$ $iqfac\_m \leftarrow ((1 << (coef\_res-1))+0.5)/(\pi/2)$ /* *Reflection Coefficient quantization* */$(i = 0; i < TNS\_MAX\_ORDER; i++)$ $index[i] \leftarrow NINT(arcsin(r[i]) * ((r[i] >= 0)\ ?\ iqfac\ :\ iqfac\_m))$; /* *Inverse Quantization* */$(i = 0; i < TNS\_MAX\_ORDER; i++)$ $rq[i] \leftarrow sin(index[i]/((index[i] >= 0)\ ?\ iqfac : iqfac\_m))$;

   The order is fixed by removing all reflections coefficients that are smaller than a threshold p (named threshold for truncation and it is set to 0.1). Then the remaining reflection coefficients are converted into order+1 linear prediction coefficients.

- **Joint Stereo Coding**

This block is used for stereo coding signals to allow transferring signals at low bit rate with high quality for multichannel audio signals. It includes two techniques : M/S stereo coding and intensity stereo coding.

M/S (Mid/Side) stereo coding is used for signals at low frequency region within each channel pair of the multichannel signal (Channels are arranged in this case symmetrically on the left and the right of the listener). So this technique helps to avoid imaging problems caused by the spatial unmasking. The idea of M/S stereo coding technique is to transform the left and right channels into a mid and side channels, by calculating $M = \frac{L+R}{2}$ and $S = \frac{L-R}{2}$ with L and R are left and right raw threshold of each noiseless coding band. Instead of using the tonality for the M and S threshold, we use the most tonal value from the L and R threshold in each band, in order to proceed the imaging control process by using the following values that are already extracted from the psychoacoustic model block :

- Mthr, Sthr, Rthr, Lthr : raw thresholds
- Mengy, Sengy, Rengy, Lengy : spread energy
- Mfthr, Sfthr, Rfthr, Lfthr: final threshold
- bmax(b) : BMLD protection ratio.

$$bmax(b) = 10^{-3[0.5+0.5cos(\pi \frac{min(bval(b).15.5}{15.5})]} \tag{17}$$

This imaging control process can be done for each band by the following pseudo-code:
[1] $t = Mthr/Sthr\ t > 1 = 1/t\ Rfthr \leftarrow max(Rthr * t, min(Rthr, bmax * Rengy))\ Lfthr \leftarrow max(Lthr*t, min(Lthr, bmax*Lengy))\ t \leftarrow min(Lthr, Rthr)$
$Mfthr \leftarrow min(t, max(Mthr, min(Sengy*bmax, Sthr))\ Sfthr \leftarrow min(t, max(Sthr, min$
$bmax, Mthr))$
As for the Intensity stereo coding, it functions with the principle of sound localization

- **Prediction:**

The Prediction tool is adopted in MPEG-2 AAC main profile, which uses previous sample to predict the current samples. It is used to improve redundancy reduction and which is especially useful when there are more or less stationary segments of a signal that are among the most demanding in terms of bitrate

requirements.

The difference between the predictor of TNS and this tool, is that in the case of the first one, within a frame, prediction is conducted on adjacent frequency samples, unlike the prediction tool which can be performed within different frames. One another difference between these two tools is that te TNS tool is useful for any block type and especially for transient signals, while the prediction tool prediction is only used if window-sequence is of type ONLY-LONG-SEQUENCE, LONG-START-SEQUENCE or LONG-STOP-SEQUENCE [3], it cannot be used for non-stationary signals.
The prediction error is calculated by :

$$e(n) = x(n) - x_{est}(n) \tag{18}$$

Where x(n) denotes the current spectral component and $x_{est}(n)$ the estimated one.

The predictor has two fundamental elements that are cascaded as a result of their realization in a lattice structure. They are calculated according to :

$$x_{est,m}(n) = b.k_m(n).r_{q,m-1}(n-1), \tag{19}$$

Where $m = 1, 2$ and :

$$r_{q,0}(n) = ax_{rec}(n), \tag{20}$$

$$r_{q,1}(n) = a(r_{q,0}(n-1) - b.k_1(n).e_{q,0}(n)), \tag{21}$$

$$e_{q,m}(n) = e_{q,m-1}(n) - x_{est,m} \tag{22}$$

To deduce the estimate as :

$$x_{est}(n) = x_{est,1}(n) + x_{est,2}(n) \tag{23}$$

with the constants $0 < a, b < 1$ are the attenuation factors, which are chosen (based on the standard [3]) as a=b=0.953125. This value is chosen as a compromise between high prediction gain and small fade out time [3].

To compute the predictor coefficients we use the Least Mean Square (LMS) approach, which can be calculated as the following:

$$k_m(n+1) = \frac{COR_m(n)}{VAR_m(n)} \tag{24}$$

With :

$$COR_m(n) = \alpha.COR_m(n-1) + r_{q,m-1}(n-1).e_{q,m-1}(n) \tag{25}$$

$$VAR_m(n) = \alpha.VAR_m(n-1) + 0.5.(r^2_{q,m-1}(n-1) + e^2_{q,m-1}(n)) \tag{26}$$

Where α is the adaptation time constant that influences the impact of the current sample on the predicted value estimate. It is chosen to be equal to 0.90625.

3.3.5 Quantization

All the steps that we have seen so far was to prepare the audio signal for this quantization step, Basically the reason for this step is to reduce the amount of storage space required for each value in a digital system. This process involves converting the sampled value of the audio signal into binary values. In AAC, a nonuniform quantizer is used according to :

$$s_q(i) = sign(s(i)) \times round[(\frac{|s(i)|}{\sqrt[4]{2}^{sf}} - \alpha] \tag{27}$$

Where $s_q(i)$ and $s(i)$ are the quantized spectral and the original samples, with the *round* function we get the nearest integer value. α is a constant defined to be 0.4054, and the sf denotes the quantization parameter for the scale factor band. The key advantage of employing a nonuniform quantizer is that it has built-in noise shaping based on the amplitude of coefficients.

There are three stages to the AAC quantization module. The top one is *Loops Frame Program*, the second is *Outer Iteration Loop*, and the last one is *Inner Iteration Loop*.

- Loops Frame Program:

This iteration process gather two nested loops, it calls as first an outer loop (used to shape the quantization noise), while this calls an inner loop to control the required coding bits. The figure 18 shows a diagram of this process.

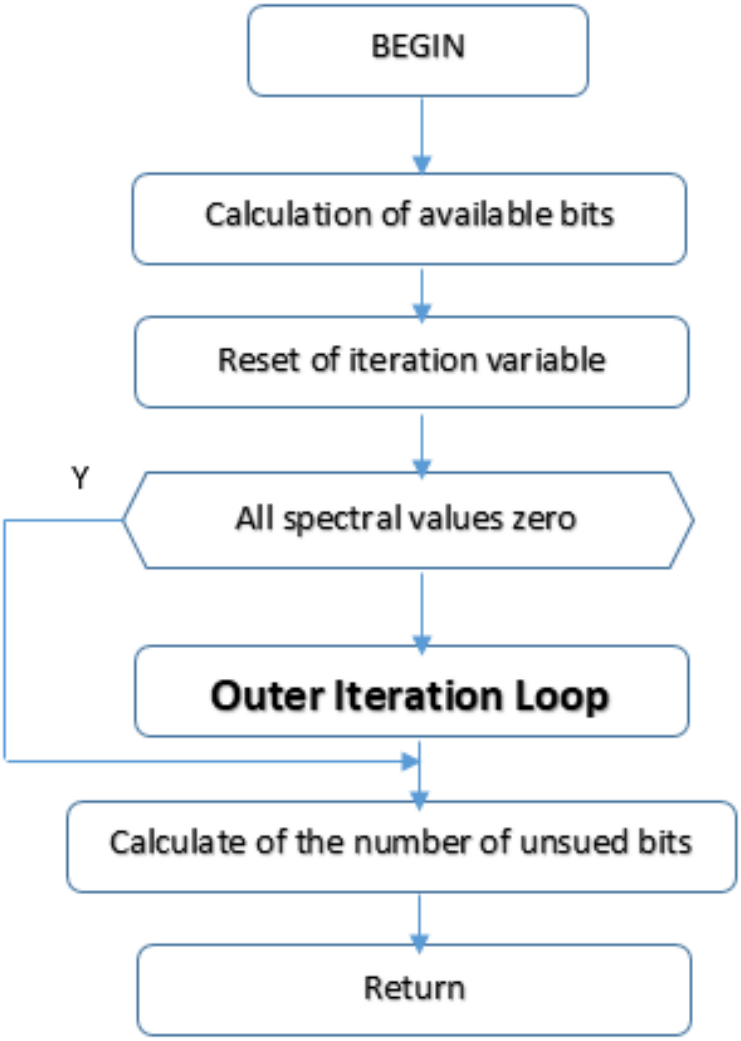

FIGURE 18: AAC Quantization iteration loop [3]

- Outer Iteration Loop

The outer iteration loop (or as it can be called also Distortion control loop) is used to control the quantization noised generated within the inner iteration loop, the different steps of this loop is explained by the diagram shown in figure 19:

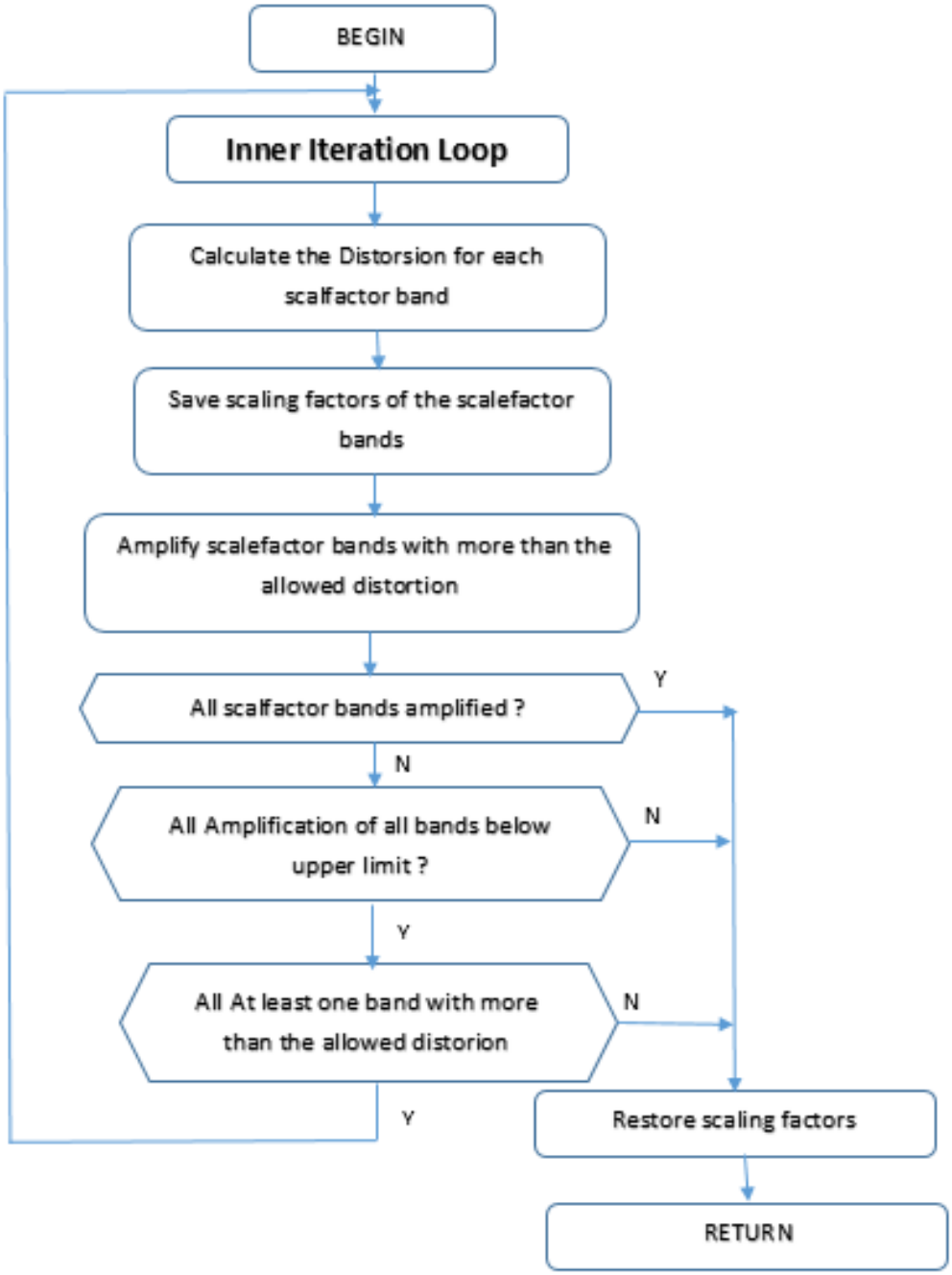

FIGURE 19: AAC Outer iteration loop [3]

- Inner Iteration Loop

This iteration loop called also Rate Control Loop which is called for each outer iteration loop. It takes as parameters, the frequency domain values, and the sacalefactors (which were applied within a scalefactor band [3]. As a result, we get the number of bits used, the quantized frequency as well as a new common-scalefactor. The Figure 20 illustrate a diagram shows the different steps of this inner iteration loop:

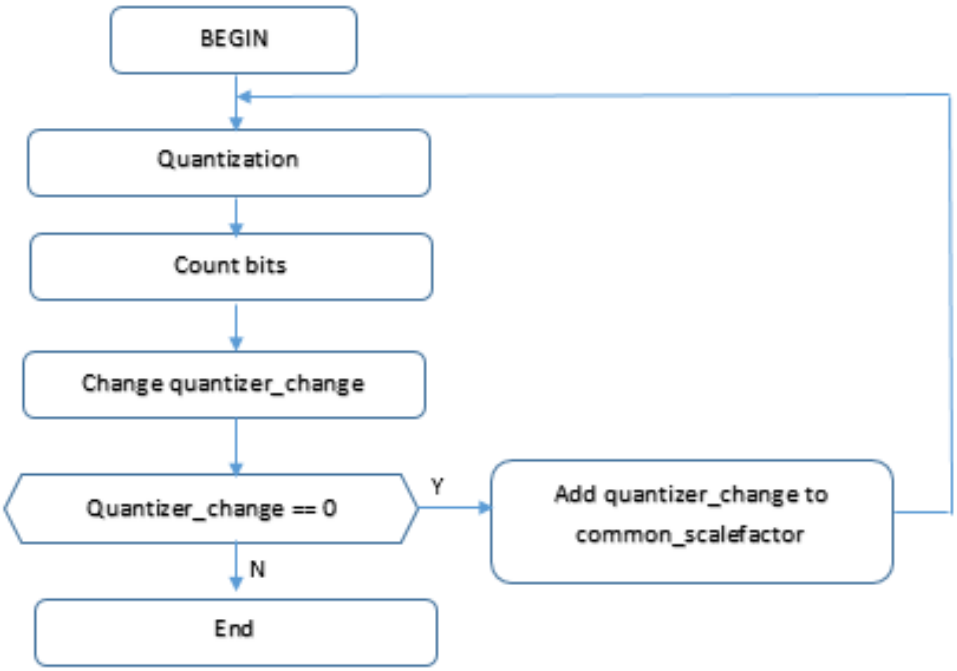

FIGURE 20: AAC Inner iteration loop [3]

3.3.6 Noiseless Coding

The Entropy coding (or Noiseless Coding) is used to reduce more the bit rate of a set of 1024 quatized spectral coefficients. The mission here is to represent the quantized samples without adding new errors if it possible. It is performed inside the inner iteration loop (explained previously). Noiseless coding is performed through three steps: Spectrum clipping, Senctioning, and Grouping and Interleaving.

- **Spectrum clipping**

This the first step of noiseless coding, where four coefficients can be coded independently as magnitudes in excess of one, with a sign carried by a value of +-1 in the quantized coefficient array.To indicate where the "clipped" coefficients are located, they are coded as integer magnitudes with an offset from the coefficient array's base. Only if this method saves bits is it used. This approach is only used if it results in a net bit savings.

- **Sectioning**

Then The set of 1024 quantized spectral coefficients is divided into sections using noiseless coding, to be able use just one single Huffman codebook to code

each section. In this way It is possible to decrease the amount of side information required to express the Huffman codebook index.
The number of bits required to represent the whole set of quantized spectral coefficients is lowered through sectioning, which is dynamic and typically changes from block to block. This is accomplished via a greedy merge process that begins with the largest number of sections feasible, each of which uses the least feasible index of the Huffman codebook.

- **Grouping and Interleaving**

More efficient coding is achieved by grouping and interleaving. In fact, the set of 1024 coefficients can be presented as a matrix of 8 by 128 frequency coefficients in the case of eight short window sequence, indicating the time-frequency evolution of the signal during this period. By rearranging the sequence of scalefactor bands and windows, the coefficients within a group are interleaved. Let us take this example:

$$c[g][w][b][k] \tag{28}$$

c denotes a set of 1024 coefficients indexed where:

- g : the index on groups
- w : the index on window in a group
- b : the index on scalefactor bands in a window
- k : the index on coefficients in a scalefadctor band

After interleaving we got the following indexed coefficients:

$$c[g][b][w][k] \tag{29}$$

Because each group is band-limited, this provides the benefit of combining all zero sections.

- **Huffman Coding**

In AAC, there are twelve predefined Huffman codebooks. Table 7.1 shows the index of each Huffman codebook, as well as the maximum absolute value of the quantized coefficients that each Huffman codebook can represent and the number of coefficients in each n-tuple for each codebook. Most codebooks contain unsigned values to save on in the codebook.The magnitude of the coefficients is Huffman coded in these codebooks, and each non-zero coefficient's sign bit is connected to the codeword.

Codebook index	n-Tulpe size	Maximum absolute Val	Signed Values
0	-	0	-
1	4	1	yes
2	4	1	yes
3	4	2	no
4	4	2	no
5	2	4	yes
6	2	4	yes
7	2	7	no
8	2	7	no
9	2	12	no
10	2	12	no
11	2	16(ESC)	no

TABLE 1: Huffman Codebooks

We note that there are two special codebooks in this table.Within a section, codebook 0 means that all coefficients are zero. And quantized coefficients with an absolute value higher than or equal to 16 can be represented by Codebook 11. An escape code is used to indicate the excess value section of the codeword for each of these coefficients. Codebook 11 can represent a maximum absolute value of 8191. As needed, the coefficient's sign bit should be attached to the codeword.

4 FFMPEG Codec

The EVS's product LSM VIA use basically the open-source FFMPEG.This source gathers wide range of libraries for interacting with video and audio files and streams.

FFMPEG includes different functionalities and libraries, such as audio/video codec, container mux and demux and also command-line program fro transcoding multimedia files. Among all these librairies, we are more interested in *libavcodec* since it contains multiples of decoder and encoders for audio and video codecs.

The codec implemented AAC encoder/decoder from the library *libavcodec*, which contains multiples classes dedicated for each bloc we have seen the theoretical background part.

In this section we will concentrate on the psycouacoustic model and the transformation block, as being the blocks we optimized in this thesis.

4.1 Psyachoustic model

The psychoacoustic model is defined by *aacpsy.c* file which contains multiples structures and functions. The following tables shows more details [2]:

Structure	**Definition**
AacPsyBand	Contains information about single band, we list band energy, energy t
AacPsyChannel	Contains information about single/pair channel for psychoacoustic m
AacPsyCoeffs	reprensent psychoacoustic model frame type-dependent coefficients i
AacPsyContext	Contains some more specific data as the number of bitrate per chann
PsyLamePreset	It's the LAME psychoacoustic model preset structure that include in

TABLE 2: Structures defined in aacpsy.c

Function	Definition
lame_calc_attack_threshold	To calculate the ABR attack threshold from the above LA
lame_window_init	LAME psy model specific initialization to initialize the a
calc_bark	to calculate the bark value
ath	to calculate the ATH value
iir_filter	infinite impulse filter used in block switching decision
psy_3gpp_window	To decide which window type to use depending on the ty
calc_bit_demand	To calculate the bit demand depending on Sequence's typ
calc_pe_3gpp	To calculate the perceptual entropy PE
calc_reduction_3gpp	To calculate the reduction that can be done based on the
calc_reduced_thr_3gpp	Recalculate the reduced threshold based the reduction ca
psy_3gpp_analyze_channel	To Calculate band thresholds
psy_3gpp_analyze	Apply the previous band threshold on each channel grou
lame_apply_block_type	To change the block type for the next window sequence c
psy_lame_window	To define the lame window for each type

TABLE 3: Functions definded in aacpsy.c

With the help of these functions, we start by calculating the attack threshold in ABR mode (average bite rate), the bark value(standard unit corresponding to one critical band width of human hearing), as well as the ATH value (Absolute threshold of hearing per band), in order to define the perceptual entropy PE and the minSNR (the minimum signal to noise ratio). Then it comes to the block switching where we decide the type of the next sequence depending to the previous one and the detection of an attack calculated as the sum over the short block size for the 8 blocks of the square of the signal after being filtered with an IIR filter, this value is compared to the sliding average of channel energy multiplied by an attack ratio (equal to 18 if channel bitrate less than 16000 and 10 if not), to decide after the block type of the previous sequence.

As outputs of this block, we define the block-type of the actual sequence (used also in the MDCT block), the perceptual entropy by one block, the SNR and the threshold ratio.

4.2 MDCT

The important step in the transform block as we saw in the theoretical background part, is the MDCT (Modified Discrete Cosinus Transform), basically

this step helps us to avoid artifacts caused by the block boundaries. FFMPEG codec implemented the MDCT in the file called *mdct.c* which includes the following functions [**?**]:

Function	**Definition**
$ff\_kbd_{w}indow\_init$	generates a Kaiser-Bessel Derived Window
$ff\_sine\_window\_init$	Generates a sine window.
ff_mdct_init	initialise variables for the MDCT/IMDCT computation
ff_mdct_calc	To compute the MDCT
ff_mdct_end	To frees the memory block which has been allocated with av_m

TABLE 4: Functions defined in mdct.c

This file gathers also some other functions for the computation of the inverse MDCT (for the decoder). The way it computed this MDCT does not follow the method of the standard ISO/IEC 13818-7 as we explained previously, but it uses a fast method for the calculation using the Forward FFT which we will explain it in the next section.

4.3 TNS Bloc

As we saw in the theoretical part of this report, Temporal Noise Shaping or TNS block was integrated in this AAC method to improve the speech quality and avoid the effect of the temporal mismatches between masking and the quantization noise that can happen with transient signal.

In the *libavcodec* library the this block is defined in the file *aacenc_tns.c* by the following functions [**?**] :

Function	**Definition**
$compress\_coeffs$	To compress the coefficients based on the order selected and
$ff\_aac\_encode\_tns\_info$	To initialise and encode all TNS data required as the compr
$ff\_aac\_apply\_tns$	to apply the TNS defining the prediction gain and the reflec
$quantize\_coefs$	To apply the quantization for the reflection coefficients
$ff\_aac\_search\_for\_tns$	This function check if should apply the TNS or not by comp

TABLE 5: Functions defined in *aacenc_tns.c*

5 Optimized Encoder

In this section we are going to explain the different optimizations implemented on the previous codec (FFMPEG Codec). Our goal was to reach an optimized encoder in term of time of compilation and reduce as well as the complexity of the code.

First, we should which block of the encoder present more computational demand and more complexity. The following table shows the demand of different blocks of as standard AAC implementation from MPEG Reference Coder:

Block	**Percentage**
Psychoacoustic Model	22%
Filter Bank	5%
Quantization	64%
Others	9%

TABLE 6: Distribution of resources in AAC-LC encoder [8]

Due to the nested loop we can see from the table 6 the quantization take the most demanding part of the encoder. The second place is for the Psychoacoustic Model and the filter bank. from these results we focused on three optimizations which will be explained in the next parts.

5.1 Removing the Block Switching

We saw previously that the psychoacoustic model took 22% of the computation effort of the encoder.

the psychoacoustic model of an AAC audio compression adopts as we saw before a dynamic block switching. Basically the main of this block is to avoid pre-echoe's phenomenon. With this block we decide whether we change the bloc type (from short to long for example) based on the perceptual entropy computed by the psychoacoustic model, and of course this calculation demands computation effort.

the role of this block switching can be replaced by TNS module which already was designed for this kind of problems. In this thesis we will test whether the absence of this block switching will cause any significant effects. If we manage to get the same results, or at least without any noticeable negative consequences,

we can reduce the computation time of the calculation of perceptual entropy related to the short blocks as well the complexity of the algorithm.

To modify the code on order to eliminate the block switching computation, we tried to modify the first condition of the switching an a way we got always block type as LONG. This decision is made from the function *psy_3gpp_window()*, its algorithm is given as the following:

[h] [1] $next\_type = next\_window\_seq\ la\ switch\_to\_eight = 0\ attack\_n = 0\ stay\_short = 0\ (i = 0; i < 8; i++)\ (j = 0; i < 128; j++)$ v = $iir\_filter(la[i * 128 + j], iir\_state$ sum += v*v s[i] = sum sum2 += sum $(i = 0; i < 8; i++)$ s[i]¿$win\_energy * attack\_ratio\ attack\_n = i + 1;\ switch\_to\_eight = 1;$ break; . . ...
This pseudo code shows the lines responsible of the switching mode (the whole code can be found in Annex). From the value of *switch_to_eight* and the the previous type we decide the next type which could be ONLY_LONG_SEQUENCE, LONG_STAR_SEQUENCE, EIGHT_SHORT_SEQUENCE, or LONG_STOP_SEQUENC
Thus to remove this switch mode and carry on just with LONG Sequence type, we remove the For loop where we switch the value of *switch_to_eight* to 1. In this case we will not need to use iir_filter and recompute the energies for the short blocks, or even check the previous or the next block type. With all these elimination we will reduce the computation time and the complexity of the psychoacoustic model.

5.2 Fast MDCT

This optimization consist on computing the Modified Discrete Cosinus Transformation based on Forward FFT. This is feasible because the MDCT/IMDCT are actually the shifted DCT-IV/IDCT-IV, with symmetric and orthogonal transformation matrices.

Another advantage of this FAST MDCT is that the power of time samples is the same as of the spectral coefficients, assuming a window that meets the Princen-Bradley requirement for complete reconstruction is applied.

The Idea if this Fast MDCT is based on three basic steps:

- Pre-twiddling : The 2M-point temporal input is folded and organized into M/2-point complex input.
- Forward FFT : computing the M/2-point complex
- Post-twiddling : The M/2-point complex FFT spectrum is then unfolded into the M-point MDCT spectrum.

After verifying the existing code (which was based on FFMPEG library) we noticed that this optimization was already applied, thus we did not need to make modification on the MDCT with this approach.

5.3 Optimized TNS

TNS method is designed to avoid the mismatches between masking thresholds and the quantization noise due to the presence of some transient signal which will create a pre-echo effect.In TNS block we perform a linear prediction in frequency domain by using Levinson-Durbin recursive algorithm to compute linear predictive coding, or what we call LPC. the figure 21 shows the basic idea of TNS block:

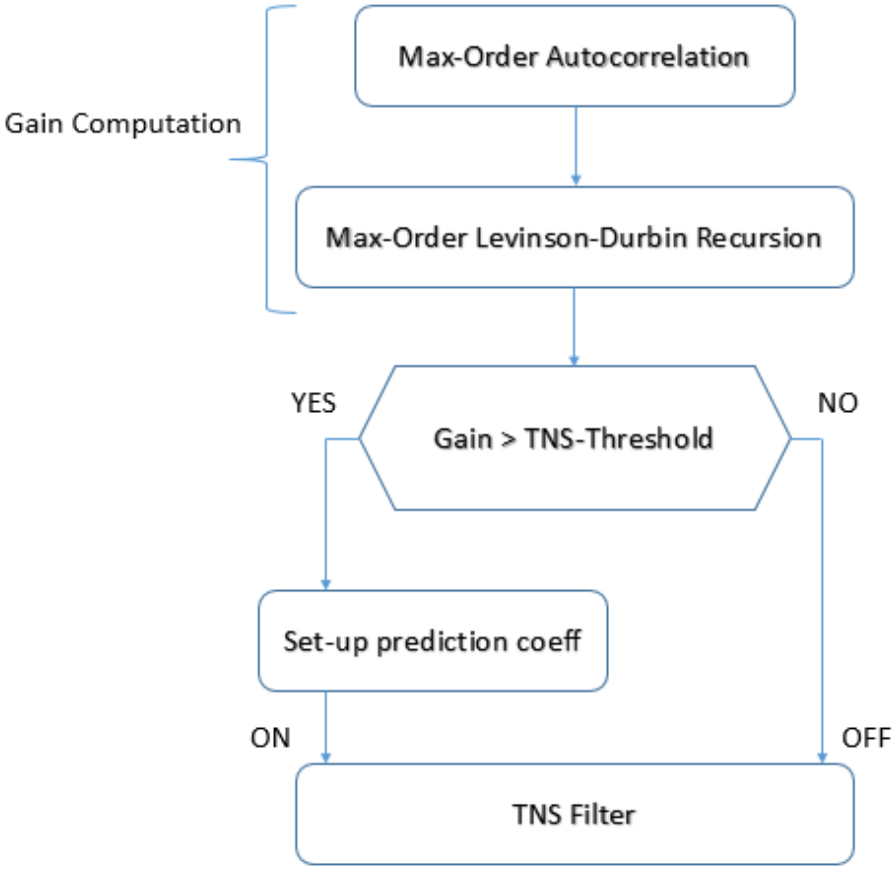

FIGURE 21: Original TNS Diagram

The Max order of predicting in our case is 20, and as we can see in the figure 21 the TNS is not always applied, we should compare first if the gain (computed from the LPC coefficients) is bigger enough to apply the TNS. That means there is some time we compute the gain until the order 20 and we do not use it (if the gain was not bigger enough).

The computation of the gain prediction can be more complex and take more effort than applying the TNS itself. The optimized TNS can be a solution to reduce this computation effort. The idea is instead of computing LPC until

order 20 and then we check, we will start by computing the LPC until the order 6 (for example) and we check if the obtained gain is bigger than the required threshold, if not the TNS will be OFF, if the gain is bigger we will continue computing the LPC coefficients until the max order and we continue as the original TNS. The figure **??** illustrate the idea of this optimization

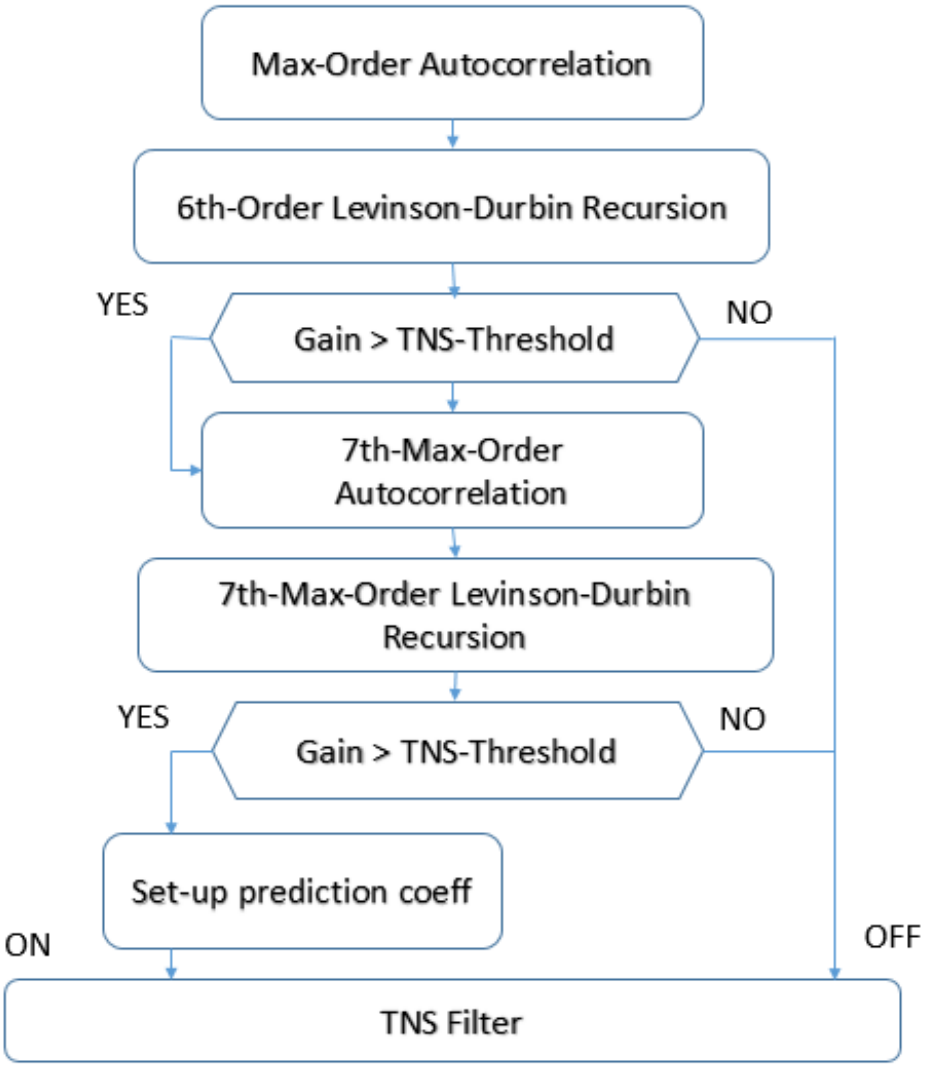

FIGURE 22: Optimized TNS Diagram

6 Results

In order to test the optimisation explained previously, we tried to encode and decode a wave file using the following steps:

- Defining the frame's vector
- Initialize and register all the muxer, demuxer and protocols we are going to using during encoding and decoding.
- Register all the codecs, parsers and bitsream filters
- Select the wanted encoder with the function $avcodec\_find\_encoder$(in our case AAC encoder)
- define the encoder paramater settings :
  - bit rate = 64000
  - sample rate = 44100
  - Number of channels = 2
  - Channel Layout = Stereo
- Open the encoder and prepare the context of the output
- Load the input wav file to start and resample it
- start a timer to be able to compute the time taken for encoding and decoding the input file
- start the conversion by sending frames to be encoded using the function $avcodec\_send\_frame(c, frame)$ which takes as arguments AVcodec context and the the AV frames, then we read the encoded data.
- return the values adequate for hardware decoding using the function $avcodec\_receive$
- write the decoded frames to an output AAC file
- Stop the timer and return the time for this compression process.

The amount of time gotten with this code, can be influenced by several factors while compiling the code with Visual Studio Code. Thus, we tried to assure the same environment for all the testing we do to be able to get significant values. Indeed we closed all the windows that can affect the computer processor and compiled the code directly via Linux terminal. We also test the code 5 times for

each audio samples without applying any modification, with the first optimization (removing block Switching), with the second optimization (the Optimized TNS), and finally withe the optimization together. The table bellow shows the average values of the Time processing for each case:

	Without Opt	**Removing Block Switching**	**Optimized TNS**	**with 2 Opt**
Mean Time(ns)	1.057e+09	8,6e+08	8,9e+08	8,5e+08
Reduced time		17%	15%	19%

Table 7: Results of testing on QRT wav file

	Without Opt	**Removing Block Switching**	**Optimized TNS**	**with 2 Opt**
Mean Time(ns)	5,93e+08	5,51e+08	5,59e+08	5,46e+08
Reduced time		7%	5%	8%

Table 8: Results of testing on Guitar wav file

	Without Opt	**Removing Block Switching**	**Optimized TNS**	**with 2 Opt**
Mean Time(ns)	5,3e+08	4,95e+08	4,94e+08	4,88e+08
Reduced time		6%	6%	8%

Table 9: Results of testing on Violoncello wav file

	Without Opt	**Removing Block Switching**	**Optimized TNS**	**with 2 Opt**
Mean Time(ns)	2,29e+09	2,11e+09	2,06e+08	2,03e+08
Reduced time		8%	10%	12%

Table 10: Results of testing on Drum wav file

The characteristic of the different audio samples used in the test are given as follow:

File Name	**Sample rate**	**Bitdepth**	**channel**	**Duration**
QRT	44100	16	Stereo	3min03s
Guitar	44100	16	Stereo	20s
Violoncello	44100	16	Stereo	19s
Drum	44100	16	Stereo	1min20s

Table 11: Characteristics of the input audio samples used in the test

From these results we observe that the optimizations implemented reduced the time of compilation with an average of 11%, this time gained can vary and depend on the audio sample. for example the first audio sample was the longest one in term of duration, and it was an acapella singing (we can say does not have a lot of sudden change in its rhythm), that's why in this case the reduced time was bigger, the same as for Drum audio sample. For other sample the duration was small which explain the difference of the reduced time.

7 Conclusion

In this Thesis Report, we presented a simplified explanation of the Encoding process of AAC Algorithm (based on the standard ISO/IEC 13818-7) including its different blocks :

- **Psychoacoustic Model:** which is used by exploiting the limitations of perceptions and identifying which parts of digital audio signal can be removed.
- **Gain Control block:** outputs a gain-control data and a gain controlled signal while receiving the input time domain signals and window sequence.
- **Transform block:** where the input signals must be transformed from time domain to their time-frequency representations.
- **Spectral processing block:** which based on four distinct blocks: temporal noise shaping (TNS), intensity stereo (IS), prediction, and mid/side (M/S) coding.
- **Quantization and Entropy Conding block:** the final step of encoding which reduce the amount of storage space required for each value in a digital system, and then educe more the bit rate of a set of 1024 quantized spectral coefficients using the noiseless coding.

The next step was studying the FFMPEG Codec in order to optimize its encoder algorithm to improve its compilation speed without any negative effects on output quality, the optimizations were focused on three blocks: Psychoacoustic model, MDCT, and TNS:

- **Removing Block Switching:** where we reduced the computation time of the calculation of perceptual entropy related to the short blocks as well the complexity of the algorithm. From the test result, we had an average optimization of 10% (depending the audio sample)
- **Fast MDCT:** Modified Discrete Cosinus Transformation based on Forward FFT. This optimization was already implemented in the codec.
- **Optimized TNS:** the idea is instead of computing LPC until order 20 and then we check, we started by computing the LPC until the order 6 (for example) and we checked if the obtained gain is bigger than the required threshold. if yes, we can stop TNS before computing all LPC coefficient for further order. With this optimization we had an average of 9% of reduced time compilation

8 Further Works

The proposed optimization in this Thesis were concentrated on C coding, and it did not carry some further encoder blocks as Quantization and the Entropy Coding. We present here some further optimisation that could also be implemented to improve the AAC encoder:

- **Quantization:** This block demand the most computation power, due to its two nested loop. Based on the research [10], the quantization algorithm can take a one-loop recursive structure by relating the distortion with the quantization error level across frequency subbands with an auditory model
- **IntMDCT:** the Modified Discrete Cosinus Transform was perfomed in this codec. The paper [7] shows that the integer modified discrete cosine transform (IntMDCT) can be used to make lossless reconstruction more efficient, and it "does not affect the perceptual quality of the coded audio under standard playback circumstances." [7]
- **Memory usage optimization :** we did not pay more attention on this aspect but this is crucial point for resource-constrained systems.

References

[1] Evs website. "https://evs.com/".

[2] Ffmpeg library. "http://ffmpeg.org/doxygen/3.1/".

[3] Information technology — generic coding of moving pictures and associated audio information, part7 advanced audio coding (aac) iso/iec 13818-7, 2006.

[4] KARLHEINZ BRANDENBURG. Mp3 and aac explained. "https://www.aes.org/e-lib/browse.cfm?elib=8079", 1999.

[5] Malcolm John Hawksford. Introduction to digital audio. "https://www.researchgate.net/publication/3584571_An_introduction_to_digital_audio", 1991.

[6] Sascha Dick Jürgen Herre. Psychoacoustic models for perceptual audio coding—a tutorial review, 2019.

[7] Te Li; S. Rahardja; R. Yu; S.N. Koh. On integer mdct for perceptual audio coding. "https://ieeexplore.ieee.org/document/4317569", 2007.

[8] Yin-Ling Lin. Mpeg-2/4 low complexity aac encoder optimization and implementation on a strongarm platform. "https://ir.nctu.edu.tw/handle/11536/69179", 2005.

[9] E. Goldberg Marina BosiRichard. Mpeg-2 aac. "http://downloads.hindawi.com/books/9789775945242/art07.pdf", 2003.

[10] M. Charbit G. Richard O. Derrien, P. Duhamel. A new quantization optimization algorithm for the mpeg advanced audio coder using a statistical subband model of the quantization noise.

Annex 1 : TEST CODE

```
#include <iostream>
#include <vector>
#include "Timers.h"
#include "Stat.h"

using namespace std;
using namespace EvsHwLGPL;

extern "C"
{
#include <libavformat/avformat.h>
// #include <libswscale/swscale.h>
#include <libswresample/swresample.h>
}

#pragma comment(lib, "avformat.lib")
#pragma comment(lib, "avcodec.lib")
#pragma comment(lib, "avutil.lib")
// #pragma comment(lib,"swscale.lib")
#pragma comment(lib, "swresample.lib")

#define NOCCURENCE 10


void load_wav_file(const char* infile, vector<AVFrame*>& frames, SwrContext *a
ctx)
{
   int nbsamples = 1024;
   int readSize = nbsamples * 2 * 2;
   FILE *fp = fopen(infile, "rb");

   while(true)
   {
      AVFrame *frame = av_frame_alloc();
      frame->format = AV_SAMPLE_FMT_FLTP;
      frame->channels = 2;
      frame->channel_layout = AV_CH_LAYOUT_STEREO;
      frame->nb_samples = nbsamples;
      int ret = av_frame_get_buffer(frame, 0);
      if (ret < 0)
      {
         cout << "av_frame_get_buffer error" << endl;
         exit(1);
      }

      char *pcm = new char[readSize];
```

```
        int len = fread(pcm, 1, readSize, fp);
        if (len <= 0)
        {
            break;
        }
        const uint8_t *data[1];
        data[0] = (uint8_t *)pcm;

        len = swr_convert(actx, frame->data, frame->nb_samples,
                          data, frame->nb_samples);

        if (len <= 0)
        {
            break;
        }

        frames.push_back(frame);
    }
}

void do_convertion(AVCodecContext *c, AVFormatContext *oc, vector<AVFrame*>& f
rames, bool dumpToFile)
{
    for (auto* frame : frames)
    {
        AVPacket pkt;
        av_init_packet(&pkt);

        //6 audio coding
        int ret = avcodec_send_frame(c, frame);
        if (ret != 0)
            continue;
        ret = avcodec_receive_packet(c, &pkt);
        if (ret != 0)
            continue;

        //7 audio packaged into aac file
        if (dumpToFile)
        {
            pkt.stream_index = 0;
            pkt.pts = 0;
            pkt.dts = 0;
            ret = av_interleaved_write_frame(oc, &pkt);
        }
    }
}

int main()
```

```
{
    char infile[] = "Drum.wav";
    char outfile[] = "out.aac";

    vector<AVFrame*> frames;

    av_register_all();
    avcodec_register_all();

    //1 open the audio encoder
    AVCodec *codec = avcodec_find_encoder(AV_CODEC_ID_AAC);
    if (!codec)
    {
        cout << "avcodec_find_encoder error" << endl;

        return -1;
    }
    //Encoder context
    AVCodecContext *c = avcodec_alloc_context3(codec);
    if (!c)
    {
        cout << "avcodec_alloc_context3 error" << endl;

        return -1;
    }
    //Encoder parameter settings
    c->bit_rate = 64000;
    c->sample_rate = 44100;
    c->sample_fmt = AV_SAMPLE_FMT_FLTP;
    c->channel_layout = AV_CH_LAYOUT_STEREO;
    c->channels = 2;
    c->flags |= AV_CODEC_FLAG_GLOBAL_HEADER;

    //Open the encoder
    int ret = avcodec_open2(c, codec, NULL);
    if (ret < 0)
    {
        cout << "avcodec_open2 error" << endl;

        return -1;
    }
    cout << "avcodec_open2 success!" << endl;

    //2 Open the context of the output package
    AVFormatContext *oc = NULL;
    avformat_alloc_output_context2(&oc, NULL, "adts", outfile);
    if (!oc)
    {
        cout << "avformat_alloc_output_context2 error" << endl;
```

```
        return -1;
    }
    AVStream *st = avformat_new_stream(oc, NULL);
    st->codecpar->codec_tag = 0;
    avcodec_parameters_from_context(st->codecpar, c);

    //3,open io,write head
    ret = avio_open(&oc->pb, outfile, AVIO_FLAG_WRITE);
    if (ret < 0)
    {
        cout << "avio_open error" << endl;

        return -1;
    }
    //Write file header
    ret = avformat_write_header(oc, NULL);

    //4 Create audio resampling context
    SwrContext *actx = NULL;
    actx = swr_alloc_set_opts(actx,
                              c->channel_layout, c->sample_fmt, c-
>sample_rate, //output format
                              AV_CH_LAYOUT_STEREO, AV_SAMPLE_FMT_S16, 44100,
 //input format
                              0, 0);
    if (!actx)
    {
        cout << "swr_alloc_set_opts error" << endl;

        return -1;
    }
    ret = swr_init(actx);
    if (ret < 0)
    {
        cout << "swr_init error" << endl;

        return -1;
    }

    //5 Open the input audio file and resample
    load_wav_file(infile, frames, actx);


    //6 Start coding
    Stat<NOCCURENCE> stat;
    CTimers timer;
    for(int i=0; i<NOCCURENCE; ++i)
    {
```

```
      timer.Start();
      do_convertion(c, oc, frames, false);
      timer.Stop();
      stat.Add(timer.GetTimeElapsed());
   }

   stat.Compute();

   cout << endl << "Encoding done in: " << stat.Mean() << " ns. "<< stat.StdDe
v() << endl;


   //7 Do a last time the convertion to dump to file
   do_convertion(c, oc, frames, true);

   //8 cleanup
   av_write_trailer(oc);

   for(auto* frame : frames)
      av_frame_free(&frame);
   avio_close(oc->pb);
   avformat_free_context(oc);
   avcodec_close(c);
   avcodec_free_context(&c);

   return 0;
}
```

Annex 2 : Removing block Switching code :

```
* Tell encoder which window types to use.
 * @see 3GPP TS26.403 5.4.1 "Blockswitching"
 */
static av_unused FFPsyWindowInfo psy_3gpp_window(FFPsyContext *ctx,
                                                 const int16_t *audio,
                                                 const int16_t *la,
                                                 int channel, int prev_type)
{
    int i, j;
    int br               = ((AacPsyContext*)ctx->model_priv_data)-
>chan_bitrate;
    int attack_ratio     = br <= 16000 ? 18 : 10;
    AacPsyContext *pctx = (AacPsyContext*) ctx->model_priv_data;
    AacPsyChannel *pch  = &pctx->ch[channel];
    uint8_t grouping     = 0;
    int next_type        = pch->next_window_seq;
    FFPsyWindowInfo wi   = { { 0 } };

    if (la) {
        float s[8], v;
        int switch_to_eight = 0;
        float sum = 0.0, sum2 = 0.0;
        int attack_n = 0;
        int stay_short = 0;
        /*for (i = 0; i < 8; i++) {
            for (j = 0; j < 128; j++) {
                v = iir_filter(la[i*128+j], pch->iir_state);
                sum += v*v;
            }
            s[i]  = sum;
            sum2 += sum; */
        }

        for (i = 0; i < 8; i++) {
            if (s[i] > pch->win_energy * attack_ratio) {
                attack_n        = i + 1;
                /*switch_to_eight = 1;*/
                break;
            }
        }
        pch->win_energy = pch->win_energy*7/8 + sum2/64;

        wi.window_type[1] = ONLY_LONG_SEQUENCE;
```

```
        wi.window_type[0] = ONLY_LONG_SEQUENCE;
        wi.window_type[3] = ONLY_LONG_SEQUENCE;


        pch->next_grouping = window_grouping[attack_n];
                pch->next_window_seq = ONLY_LONG_SEQUENCE;
    } else {
        for (i = 0; i < 3; i++)
            wi.window_type[i] = ONLY_LONG_SEQUENCE;
        grouping = (prev_type == EIGHT_SHORT_SEQUENCE) ? window_grouping[0] :
0;
    }

    wi.window_shape   = 1;
    wi.num_windows = 1;
    wi.grouping[0] = 1;


    return wi;
}
```

Annex 2 : Optimized TNS code (calculation of LPC coefficients) :

```
double ff_lpc_calc_ref_coefs_f(LPCContext *s, const float *samples, int len,
                               int order, double *ref)
{
    int i;
    double TNS_GAIN_THRESHOLD_LOW = 1.4f;
    double TNS_GAIN_THRESHOLD_HIGH = 1.16f*TNS_GAIN_THRESHOLD_LOW;
    double signal = 0.0f, avg_err = 0.0f;
    double autoc[MAX_LPC_ORDER+1] = {0}, error[MAX_LPC_ORDER+1] = {0};
    const double a = 0.5f, b = 1.0f - a;


    /* Apply windowing */
    for (i = 0; i <= len / 2; i++) {
        double weight = a - b*cos((2*M_PI*i)/(len - 1));
        s->windowed_samples[i] = weight*samples[i];
        s->windowed_samples[len-1-i] = weight*samples[len-1-i];
    }

    s->lpc_compute_autocorr(s->windowed_samples, len, order, autoc);
    signal = autoc[0];
    s->flag =0;
    compute_ref_coefs(autoc, order, ref, error);

    for (i = 0; i < order; i++)
        avg_err = (avg_err + error[i])/2.0f;
        if ((i==6) && ((signal/avg_err)< TNS_GAIN_THRESHOLD_LOW || (signal/avg
_err) > TNS_GAIN_THRESHOLD_HIGH))
            s->flag = 1;
            return signal/avg_err;
    return signal/avg_err;
}
```