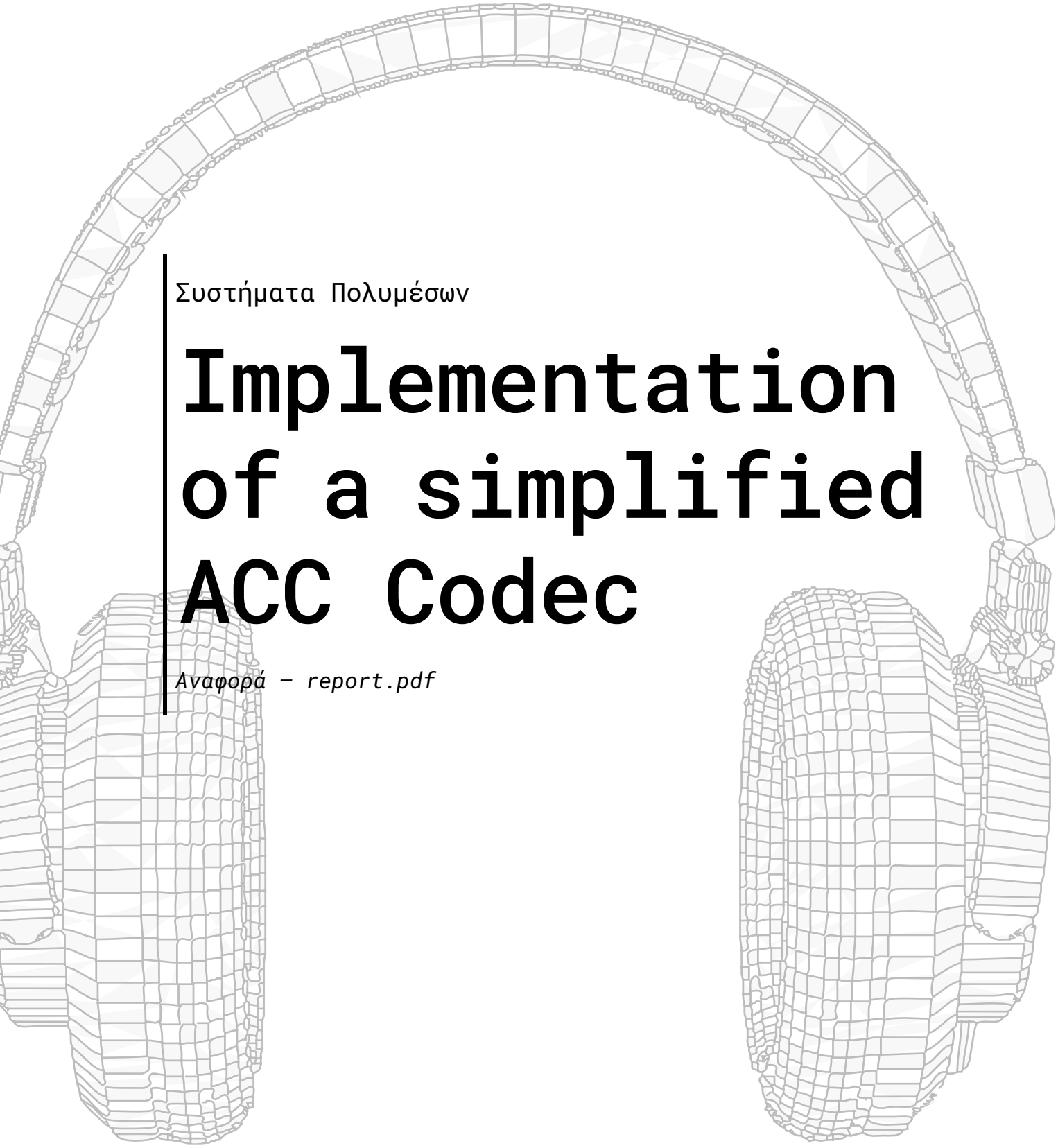
**Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης**
Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών



Συστήματα Πολυμέσων

# Implementation of a simplified ACC Codec

*Αναφορά – report.pdf*

Μάθημα: **Συστήματα Πολυμέσων και Εικονική Πραγματικότητα**
Εκπονητές: 1) **Χαρισούδης Θανάσης** ( ΑΕΜ: **9026** )
2) **Κακάλης Λουκάς** ( ΑΕΜ: **8450** )
Επιβλέπων: κ. **Α. Ντελόπουλος**

# Πίνακας περιεχομένων

# Πίνακας εικόνων



# Πίνακας scripts

# Σημαντικές Επισημάνσεις

## Διάρθρωση Παραδοτέου

Το παρών αρχείο βρίσκεται στο αρχικό κατάλογο του παραδοτέου αρχείου ( *9026_8450.zip* ). Ο κώδικας της εργασίας βρίσκεται στον κατάλογο *matlab*, και συγκεκριμένα στους παρακάτω υπο-καταλόγους:

| Όνομα Καταλόγου | Περιγραφή |
|---|---|
| **dev** | Περιέχεται ο κώδικας όπως υπάρχει στο περιβάλλον ανάπτυξης. Περιέχει τους υπο-καταλόγους *L{$l_i$}*, με $l_i$ = {1, 2, 3}, καθένας από τους οποίους έχει μόνο τις συναρτήσεις που είναι απαραίτητες στον level-$l_i$ codec.<br>Για εκτέλεση πρέπει να μπει όλος ο *dev* φάκελος στο path και κατόπιν να εκτελεστούν τα demos. |
| **prod_di**<br>*dependencies injected* | Περιέχεται ο κώδικας σε τέτοια διάρθρωση ώστε σε κάθε υπό-κατάλογο, *Level-{$l_i$}*, με $l_i$ = {1, 2, 3}, να υπάρχουν όλες οι συναρτήσεις που απαιτούνται στον level-$l_i$ codec. Αυτό απαιτεί την ύπαρξη όλων των συναρτήσεων των υπο-καταλόγων L{$l_j$} του *dev*, με $l_j$ < $l_i$, στον κατάλογο *Level-{$l_i$}*.<br><br>**Εδώ βρίσκεται ο κώδικας όπως ζητείται στην εκφώνηση της εργασίας.** |

Προσοχή: Δεν συμπεριλαμβάνεται το δοσμένο αρχείο ήχου ( *LicorDeCalandraca.wav* ) σε κανέναν από τους παραπάνω καταλόγους.

## Η εργασία στο GitLab®

Όλος ο κώδικας της εργασίας βρίσκεται σε ένα public repository στο GitLab. Το repository μπορεί να βρεθεί στο ακόλουθο link:
https://gitlab.com/auth_ece/courses/9th_semester/multimedia-systems

## Βιβλιογραφικές Αναφορές

Για την υλοποίηση της αναφοράς της εργασίας χρησιμοποιήθηκαν πηγές από:


- Wikipedia / PQMF, MDCT, TNS, Psychoacoustic Model
- MathWorks / Simulink Toolbox, DSP Toolbox
- Introduction to Digital Audio Coding and Standards/by Marina Bosi, Richard E. Goldberg, SPRINGER 2003

# Εισαγωγή

Η παρούσα εργασία υλοποιήθηκε κατά τη διάρκεια του χειμερινού εξαμήνου 2019 κατά την ακαδημαϊκή χρονιά 2018 – 2019 και το μάθημα «Συστήματα Πολυμέσων και Εικονική Πραγματικότητα» του 9ου εξαμήνου του τμήματος Η.Μ.Μ.Υ. του Α.Π.Θ.. Στόχος της εργασίας είναι η υλοποίηση μιας απλοποιημένης έκδοσης του προτύπου ψηφιακής κωδικοποίησης ήχου Advance Audio Coding ( AAC ) όπως υλοποιήθηκε στο MPEG-2.

Συγκεκριμένα, ο codec ( encoder – decoder: κωδικοποιητής – αποκωδικοποιητής ) υλοποιήθηκε σε τρία επίπεδα ( levels ) με κάθε επόμενο level να χρησιμοποιεί το αποτέλεσμα του προηγούμενου και να προσθέτει επιπλέον συμπίεση της πληροφορίας, ως εξής:

1. *Level-1*: Σπάει το δειγματοληπτημένο σήμα σε παράθυρα ( frames ) και αναθέτει σε κάθε ένα από αυτά ένα τύπο ( Long ή Eight Short ) ανάλογα με το στασιμότητα του σήματος στο παράθυρο αυτό ( αναλυτική εξήγηση παρακάτω ). Επίσης, επιτελεί το time-to-frequency mapping για κάθε frame εφαρμόζοντας έναν DFT-like μετασχηματισμό, τον MDCT με την σημαντική ιδιότητα ότι διατηρεί τον αριθμό των δειγμάτων στο νέο space ( critically sampled ).
2. *Level-2*: Εφαρμόζει την τεχνική Temporal Noise Shaping ( TNS ) η οποία εκμεταλλεύεται την δυαδικότητα χρόνου-συχνότητας στην προσπάθειά της να αφαιρέσει περιοδικότητες από το μετασχηματισμένο κατά MDCT σήμα στο πεδίο της συχνότητας.
3. *Level-3*: Εφαρμόζει μια απλοποιημένη εκδοχή του ψυχοακουστικού μοντέλου στα παράθυρα ώστε το σφάλμα του κβαντιστή να είναι σε συχνοτικές περιοχές που δεν επιδρούν αισθητά στην ακουστικότητα του σήματος. Αυτό είναι εφικτό λόγω παρατηρήσεων κατά τη δημιουργία του ψυχοακουστικού μοντέλου σχετικά τόσο με το κατώφλι ακουστικότητας όσο και με το φαινόμενο επικάλυψης ( masking ) στο πεδίο της συχνότητας. Επίσης, στο επίπεδο αυτό υλοποιείται και η βαθμίδα κωδικοποίησης εντροπίας ( entropy coding ) μέσω της κωδικοποίησης Huffman στα κβαντισμένα δείγματα της συχνότητας.

## Ανάλυση Καταλόγων Κώδικα ( Directories )

Στον αρχικό κατάλογο διακρίνονται τρεις υπό-κατάλογοι:

1. **L1**: περιέχει τις συναρτήσεις του πρώτου επιπέδου του codec καθώς και το demo file για το πρώτο επίπεδο ( *demoAAC1.m* ). Στον κατάλογο αυτό περιέχονται μόνο οι συναρτήσεις ( με το signature ) που ζητούνται ενώ

για οποιεσδήποτε βοηθητικές συναρτήσεις των συναρτήσεων αυτών έχουν δημιουργηθεί υπό-κατάλογοι με τα ονόματα των συναρτήσεων σε uppercase και αυτές έχουν τοποθετηθεί εκεί μέσα, αντίστοιχα για κάθε βασική συνάρτηση ( με τον όρο βασική νοείται μια συνάρτηση που μας ζητήθηκε στην εκφώνηση )

2. **L2**: περιέχει τις συναρτήσεις του δεύτερου επιπέδου του codec καθώς και το demo file για το δεύτερο επίπεδο ( *demoAAC2.m* ). Αντίστοιχα με το πρώτο level στο root του καταλόγου αυτού περιέχονται μόνο οι βασικές συναρτήσεις του level-2 codec ενώ οι βοηθητικές έχουν τοποθετηθεί σε υπό-φακέλους με την ίδια λογική.
3. **L3**: περιέχει τις συναρτήσεις του τρίτου επιπέδου του codec καθώς και το demo file για το τρίτο επίπεδο ( *demoAAC3.m* ). Αντίστοιχα με το προηγούμενα levels στο root του καταλόγου αυτού περιέχονται μόνο οι βασικές συναρτήσεις του level-3 codec ενώ οι βοηθητικές έχουν τοποθετηθεί σε υπό-φακέλους με την ίδια λογική.

## Περιβάλλον Ανάπτυξης Κώδικα

Ο κώδικας αναπτύχθηκε στο λογισμικό MATLAB έκδοση R2018b ενώ έχει δοκιμαστεί για την σωστή λειτουργία του και στην έκδοση R2015a του ίδιου λογισμικού.

Τα χαρακτηριστικά του υπολογιστή που χρησιμοποιήθηκε τόσο για την ανάπτυξη όσο και για τις δοκιμές του κώδικα είναι:

| Χαρακτηριστικό | Τιμή |
|---|---|
| Επεξεργαστής | Intel Core i7 6700HQ 2.6GHz × 8 |
| Κύρια Μνήμη | DDR3 8GB |
| Δευτερεύουσα Μνήμη | SSD 128GB |
| Κάρτα Γραφικών | Intel HD Graphics 530 (Skylake GT2) |
| Λειτουργικό Σύστημα | Ubuntu 18.04.2 LTS |
| Έκδοση MATLAB | R2018a |

# 1. Level-1 AAC Codec

## 1.1 Υλοποίηση των παραθύρων σε MATLAB

Βασική απαίτηση για τον υπολογισμό του DFT ενός σήματος είναι το σήμα να είναι time limited ( θεώρ. Δειγματοληψίας ). Για να περιορίσουμε τα frames στον χρόνο πρέπει να εφαρμόσουμε ένα παράθυρο σε αυτά. Το απλούστερο παράθυρο θα ήταν το rectangular window όμως αυτό επηρεάζει σημαντικά το συχνοτικό περιεχόμενο του κάθε frame καθώς συνελίσσεται με αυτό. Έτσι, χρησιμοποιήθηκε το παραμετρικό παράθυρο Kaiser-Bessel όπου «παίζοντας» με την παράμετρο “alpha” έχουμε από rectangular μέχρι hanning windows.

Για την αντίστροφη διαδικασία ( «ξε-παραθύρωση» ) η απλούστερη προσέγγιση θα ήταν απλώς να διαιρέσουμε το παραθυρωμένο σήμα με το παράθυρο ( για κάθε n ). Ωστόσο, επειδή το σήμα μέχρι να φτάσει στο σημείο του unwindowing στον decoder έχει υποστεί μετασχηματισμό στην συχνότητα και κυρίως κβαντισμό, κάτι τέτοιο θα αύξανε τον θόρυβο κβαντισμού στα άκρα του κάθε παραθύρου. Επιλέγεται συνεπώς η εφαρμογή της τεχνικής **overlap-and-add** όπου το κάθε frame έχει κάποιο overlap ( για τον codec που υλοποιήθηκε αυτό είναι 50% ) με τα γειτονικά του. Το κάθε frame ξανά περνάει από παραθύρωση στον decoder μειώνοντας έτσι το quantization noise στα άκρα του παραθύρου ( δεδομ. ότι χρησιμοποιείται Kaiser-Bessel window ). Αυτό απαιτεί μια τροποποίηση του χρησιμοποιούμενου παραθύρου ώστε να πληρεί κάποια συνθήκη στις overlapped regions, και συγκεκριμένα:

$$w[n] = \begin{cases} \sqrt{\dfrac{\sum_{p=0}^{n} w'[p]}{\sum_{p=0}^{N-M} w'[p]}} & \text{for } n = 0,\ldots,N-M-1 \\ 1 & \text{for } n = N-M,\ldots,M-1 \\ \sqrt{\dfrac{\sum_{p=n-M+1}^{N-M} w'[p]}{\sum_{p=0}^{N-M} w'[p]}} & \text{for } n = M,\ldots,N-1 \end{cases}$$

*Εικόνα 1: Τροποποίηση παραθύρου w'[n] ώστε να πληρεί τις συνθήκες Overlap-and-Add ( πηγή: Introduction to Digital Audio Coding and Standards/by Marina Bosi, Richard E. Goldberg, SPRINGER 2003 )*

Εφαρμόζοντας την παραπάνω διαδικασία στο Kaiser-Bessel για $M = \frac{N}{2}$ προκύπτει το επίσης παραμετρικό παράθυρο Kaiser-Bessel-Derived ( KBD ), του οποίου η υλοποίηση σε MATLAB φαίνεται ακολούθως:

```
% Kaiser-Bessel-derived (KBD) window function ( alpha = {window_param} )
wn = kaiser( window_size / 2 + 1, window_param * pi );
wn_cs = cumsum(  wn(  1 : window_size / 2 )  );
wn = sqrt( [ wn_cs; wn_cs( window_size / 2 : -1 : 1 ) ] ./ sum( wn ) );
```

*Script 1: Παραγωγή της συνάρτησης KBD για δοσμένο αριθμό δειγμάτων και παράμετρο alpha*

## 1.2 Υλοποίηση MDCT σε MATLAB

### 1.2.1 Υλοποίηση του Marios Athineos

Χρησιμοποιήθηκαν δύο υλοποιήσεις MDCT σε MATLAB. Η πρώτη από αυτές είναι από τον Mario Athineo και βρίσκεται εδώ: *<link>*.

Πρέπει να τονιστεί ότι αυτή η υλοποίηση παράγει καλύτερο SNR στα Levels 1 και 2 του AAC Codec. Στο Level 3 ωστόσο δεν υπάρχει καμία αλλαγή τόσο στο παραγόμενο SNR όσο και στη συμπίεση των δεδομένων.

### 1.2.2 Υλοποίηση από το βιβλίο της M. Bosi

Για να παραμείνει το σήμα critically sampled μετά το overlap των παραθύρων αντί για τον DFT εφαρμόζεται ο MDCT ο οποίος έχει την πολύ σημαντική ιδιότητα ότι για N input time samples παράγει N/2 output frequency samples.

Το βασικότερο μέρος του χρόνου στον Level-1 encoder είναι ο MDCT σε κάθε frame των 2048 δειγμάτων ( αντίστοιχα για τα short frames ). Υλοποιήθηκε μία «γρήγορη» μορφή MDCT με την καρδιά της υλοποίησης είναι ο **O(N*lgN) FFT** ως εξής:

$$
\begin{aligned}
X[k] &= \sum_{n=0}^{N-1} x[n]w[n]\cos\left(\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})\right) \\
&= \mathrm{Re}\left\{\sum_{n=0}^{N-1} x[n]w[n]e^{-j\frac{2\pi}{N}(n+n_0)(k+\frac{1}{2})}\right\} \\
&= \mathrm{Re}\left\{e^{-j\frac{2\pi}{N}n_0(k+\frac{1}{2})}\sum_{n=0}^{N-1}[x[n]w[n]e^{-j\frac{2\pi n}{2N}}]e^{-j\frac{2\pi kn}{N}}\right\}
\end{aligned}
$$

*Εικόνα 2: MDCT Implementation using DFT ( πηγή: Introduction to Digital Audio Coding and Standards/by Marina Bosi, Richard E. Goldberg, SPRINGER 2003 )*

όπου φαίνεται ότι ο DFT ( το άθροισμα ) στο παραθυρωμένο ( windowed ) σήμα $x_w[n] = x[n] \times w[n]$.

Ακολούθως, φαίνεται η υλοποίηση στο MATLAB:

```
% Complex Exponentials
c1 = exp( ( -1i * pi * n ) / N );
c2 = exp( ( ( -1i * 2*pi * n0 ) / N ) * (k + 0.5 ) );

%% Pre-twiddle ( multiplication with complex term )
xn_pre_twjddled = c1 .* xn;

%% N-point FFT
Xk_untwiddled = fft( xn_pre_twjddled );

%% Post-twiddle
Xk_post_twiddled = c2 .* Xk_untwiddled( 1:K );

%% Real
Xk = real( Xk_post_twiddled ) / sqrt( N );
```

*Script 2: Υλοποίηση MDCT κάνοντας χρήση του FFT για επιτάχυνση*

Η παραπάνω υλοποίηση δίνει πολύ σημαντική επιτάχυνση ( 100x ) στην υπολογισμό του MDCT σε σύγκριση με την υλοποίηση του τύπου που μας δόθηκε, κάτι που βοήθησε στην επίτευξη πολύ καλών χρόνων στο Level-1 AAC Codec.

## 1.3 Εκτέλεση Level-1 Codec – Αποτελέσματα

Το πρώτο επίπεδο του codec που υλοποιήθηκε αναλαμβάνει δύο απλές αλλά αρκετά σημαντικές λειτουργίες: 1) το σπάσιμο του αρχικού σήματος σε frames και 2) τον μετασχηματισμό του κάθε frame ( ή subframe ) στην συχνότητα μέσω MDCT αφού πρώτα το κάθε frame ( ή subframe ) παραθυρωθεί στον χρόνο με ένα από τα διαθέσιμα overlap-and-add compliant windows ( KBD ή sine ).

Παρακάτω παρατίθενται μέρη του level-1 encoder & decoder καθώς επίσης και τα αποτελέσματα από την εκτέλεση της *demoAAC1()*:

```
%% SSC: Find frame Type
% first frame ( only right half has non-zero values ) is set arbitarily
% to OLS
AACSeq1( 1 ).frameType = AACONFIG.L1.SSC_FIRST_FRAME_TYPE;
for frame_i = 2: NFRAMES - 1

    AACSeq1( frame_i ).frameType = SSC(...
        channel_frames_by_channel( :, frame_i, : ), ...
        channel_frames_by_channel( :, frame_i + 1, : ), ...
        AACSeq1( frame_i - 1 ).frameType ...
    );

end

%% FilterBank: Time-to-Frequency Mapping for each frame using MDCT
for channel = 'lr'

    for frame_i = 1 : NFRAMES

        % Per Channel
        AACSeq1( frame_i ).(['ch' channel]).frameF = permute( filterbank( ...
            channel_frames_by_channel( :, frame_i, strfind( 'lr', channel ) ), ...
            AACSeq1( frame_i ).frameType, ...
            AACONFIG.L1.WINDOW_SHAPE ...
        ), [ 1, 3, 2 ] );

    end
```

*Script 3: Level-1 Encoder ( φαίνονται οι κλήσεις των συναρτήσεων SSC() & filterbank() )*

Τα αποτελέσματα ( χρόνος εκτέλεσης & SNR ) της εκτέλεσης τους στο δοσμένο αρχείου ήχου φαίνονται ακολούθως:

i. Με βάση την δική μας υλοποίηση του MDCT:

```
>> demoAAC1( 'sample.wav', 'sample_out.wav', ConfSets.Default );

Level 1
=======
Coding: time elapsed is 0.25749 seconds
Decoding: time elapsed is 0.22649 seconds
      -> total time: 0.48401 seconds
Channel 1 SNR: 257.5172 dB
Channel 2 SNR: 257.4526 dB
      -> mean SNR: 257.4849 seconds
```

*Script 4: Αποτελέσματα εκτέλεσης του Level-1 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την δική μας υλοποίηση του MDCT*

ii. Με βάση την υλοποίηση του Marios Athineos:

```
>> demoAAC1( 'sample.wav', 'sample_out.wav', ConfSets.Marios );

Level 1
=======
Coding: time elapsed is 0.38086 seconds
Decoding: time elapsed is 0.31432 seconds
      -> total time: 0.69522 seconds
Channel 1 SNR: 301.5860 dB
Channel 2 SNR: 301.7060 dB
      -> mean SNR: 301.6460 seconds
```

*Script 5: Αποτελέσματα εκτέλεσης του Level-1 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την υλοποίηση MDCT του Marios Athineos*

# 2. Level-2 AAC Codec

## 2.1 Υπολογισμός των συντελεστών γραμμικής πρόβλεψης

Για τον υπολογισμό των συντελεστών του φίλτρου γραμμικής πρόβλεψης 4ης τάξης χρησιμοποιήθηκε η builtin συνάρτηση της MATLAB, *lpc()*. Κατόπιν οι κανονικοποιημένοι MDCT coefficients φιλτραρίστηκαν με το φίλτρο αυτό.

Σημειώνεται ότι έγινε και αναλυτικός υπολογισμός των παραπάνω συντελεστών με βάση την δοθείσα σχέση, κάτι που αλλάζει ελάχιστα τα τελικά αποτελέσματα ( προς το χειρότερο ) και αυξάνει και ελάχιστα τον χρόνο εκτέλεσης.

Στα frames που το αντίστροφο φίλτρο δεν ήταν ευσταθές ( μηδενικά έξω από τον μοναδιαίο κύκλο στο z-pane ) έγινε σταθεροποίηση του φίλτρου με την *polystab()*, η οποία δεν αλλάζει την απόκριση πλάτους της συνάρτησης μεταφοράς του φίλτρου παρά μόνο την απόκριση φάσης.

## 2.2 Κβαντισμός Συντελεστών

Για τον κβαντισμό των συντελεστών χρησιμοποιήθηκαν οι κλάσεις *ScalarQuantizerEncoder* και *ScalarQuantizerDecoder* του DSP Toolbox του MATLAB. Για αύξηση της ταχύτητας εκτέλεσης, οι παραπάνω κλάσεις αρχικοποιούνται την πρώτη φορά που καλείται η συνάρτηση που υπολογίζει τους κβαντισμένους συντελεστές, αφού γίνεται η παραδοχή ότι κατά τη διάρκεια εκτέλεσης του codec δεν αλλάζουν τα στοιχεία του κβανιστή / αποκβαντιστή.

## 2.3 Εκτέλεση Level-2 Codec – Αποτελέσματα

Το δεύτερο επίπεδο του codec που υλοποιήθηκε αναλαμβάνει σημαντική λειτουργία: την εφαρμογή temporal noise shaping ( TNS ) στους MDCT συντελεστές του κάθε frame με στόχο την μείωση της reductant πληροφορίας σε περιοχές μεγάλων μεταβολών του σήματος στο πεδίο του χρόνου.

Παρακάτω παρατίθενται μέρη του level-2 encoder & decoder καθώς επίσης και τα αποτελέσματα από την εκτέλεση της *demoAAC2()*:

```
% Initialize output struct
AACSeq2 = AACSeq1;

% Apply TNS
for channel = 'lr'

    for frame_i = 1 : NFRAMES

        % Per Channel
        [AACSeq2( frame_i ).(['ch' channel]).frameF, AACSeq2( frame_i ).(['ch'
channel]).TNScoeffs] = TNS( ...
            AACSeq2( frame_i ).(['ch' channel]).frameF, ...
            AACSeq2( frame_i ).frameType ...
        );

    end

end
```

*Script 6: Level-2 Encoder ( φαίνονται η κλήση της συνάρτησης TNS() )*

Τα αποτελέσματα ( χρόνος εκτέλεσης & SNR ) της εκτέλεσης τους στο δοσμένο αρχείου ήχου φαίνονται ακολούθως:

i. Με βάση την δική μας υλοποίηση του MDCT:

```
>> demoAAC2( 'sample.wav', 'sample_out.wav', ConfSets.Default );

Level 2
=======
Coding: time elapsed is 0.68325 seconds
Decoding: time elapsed is 0.32809 seconds
      -> total time: 1.01135 seconds
Channel 1 SNR: 257.5174 dB
Channel 2 SNR: 257.4526 dB
      -> mean SNR: 257.4850 seconds
```

*Script 7: Αποτελέσματα εκτέλεσης του Level-2 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την δική μας υλοποίηση του MDCT*

ii. Με βάση την υλοποίηση του Marios Athineos:

```
>> demoAAC2( 'sample.wav', 'sample_out.wav', ConfSets.Marios );

Level 2
=======
Coding: time elapsed is 0.75668 seconds
Decoding: time elapsed is 0.35719 seconds
      -> total time: 1.11390 seconds
Channel 1 SNR: 301.5826 dB
Channel 2 SNR: 301.6911 dB
      -> mean SNR: 301.6369 seconds
```

*Script 8: Αποτελέσματα εκτέλεσης του Level-2 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την υλοποίηση MDCT του Marios Athineos*

# 3. Level-3 AAC Codec

## 3.1 Υλοποίηση του Ψυχοακουστικού Μοντέλου – *psycho()*

Η υλοποίηση του ψυχοακουστικού μοντέλου στην αντίστοιχη συνάρτηση έγινε ακολουθώντας την εκφώνηση. Έγιναν, ωστόσο, οι ακόλουθες μεταβολές με βάση την περιγραφή του AAC προτύπου που μας δόθηκε:

- στον υπολογισμό του μέτρου προβλεψιμότητας, *c(w)*, όταν έχουμε long frames θέτουμε σταθερή τιμή 0.4 σε όλους του συντελεστές του μηχαμηλού φάσματος. Συγκεκριμένα:

$$c(w) = \begin{cases} \sqrt{\left(r(w) * \cos\left(f(w)\right) - rpred(w) * \cos(fpred(w))\right)^2 + (r(w) * \sin(f) - rpred(w) * \sin(fpred(w)))^2}, & 1 \le v \le 6 \\ \mathbf{0.4}, & 7 \le w \le 1024 \end{cases}$$

- μετά τον υπολογισμό του tonality index για κάθε μπάντα, αυτά clipάρονται ώστε να ανήκουν στο [0, 1]

## 3.2 Υλοποίηση Κβαντιστή – *AACquantizer()*

### 3.2.1 Κβαντισμός ανά Μπάντα Συχνοτήτων

Στην αρχική μας υλοποίηση του κβαντιστή, σε κάθε βήμα υπολογισμού ο κβαντιστής έπαιρνε μία μπάντα συχνοτήτων, έστω την b-οστή, και αύξανε το *a(b)* έως ότου το σφάλμα κβαντισμού στην συγκεκριμένη μπάντα ξεπερνούσε το κατώφλι ακουστικότητας της μπάντας, *T(b)*. Η διαδικασία επαναλαμβάνονταν σε κάθε frame για όλες της μπάντες του frameType του frame.

Τα αποτελέσματα από την πρώτη υλοποίηση του κβαντιστή ήταν αρκετά ικανοποιητικά, ωστόσο η συμπίεση ήταν χαμηλότερη από την αναμενόμενη.

### 3.2.2 Πέρασμα όλων των Μπάντων σε βήμα υπολογισμού

Στη δεύτερη υλοποίηση εξ' αρχής του κβαντιστή, σε κάθε βήμα υπολογισμού υπολογίζει για κάθε μπάντα το σφάλμα κβαντισμού και ακολούθως αυξάνει κατά ένα ή μειώνει κατά ένα τον αντίστοιχο scale factor, ανάλογα με το εάν το σφάλμα αυτό είναι κάτω ή πάνω από το κατώφλι ακουστικότητας αντίστοιχα. Η διαδικασία συνεχίζεται για όσο δεν έχει φτάσει το σφάλμα για όλες τις μπάντες στο κατώφλι ακουστικότητας. Η διαδικασία σταματάει εάν στο τέλος κάποιου βήματος βρεθεί ότι το μέγιστο της διαφοράς μεταξύ διαδοχικών scale factors ξεπεράσει το 60;

Η συγκεκριμένη υλοποίηση, αν και πιο αργή και με χειρότερο SNR του ανακατασκευασμένου αρχείου, παρουσιάζει καλύτερη συμπίεση, λίγο πάνω από το 16% του αρχικού, ασυμπίεστου, αρχείου ήχου.

## 3.3 Υλοποίηση βαθμίδας Κωδικοποίησης Huffman

Για την κωδικοποίηση ( εντροπίας ) τόσο των κβαντισμένων συντελεστών MDCT όσο και των DPCM'ed scale factors για κάθε μπάντα χρησιμοποιήθηκαν οι δοσμένες συναρτήσεις που αναλαμβάνουν την κωδικοποίηση & αποκωδικοποίηση χρησιμοποιώντας κώδικες Huffman.

## 3.4 Αποθήκευση Συμπιεσμένης Δομής - Μέτρηση μεγέθους

Πριν την αποθήκευση της struct που έχει το αποτέλεσμα του *AACquantizer* αφαιρούνται από αυτή τα πεδία *frameF* και *frameT* για κάθε frame, παράγοντας έτσι μια νέα δομή με τα απαραίτητα δεδομένα που πρέπει να αποθηκευτούν στο τελικό binary αρχείο ( π.χ. το .aac αρχείο ).

Ακολούθως, η συνάρτηση *L3_AACODER_fwrite_bin()* αναλαμβάνει να γράψει τα περιεχόμενα της παραπάνω compressed struct σε ένα αρχείο πεδίο ανά πεδίο.

## 3.5 Εκτέλεση Level-3 Codec – Αποτελέσματα

Παρακάτω παρατίθενται μέρη του level-3 encoder & decoder καθώς επίσης και τα αποτελέσματα από την εκτέλεση της *demoAAC3()*:

```
% Initialize output struct
AACSeq3 = AACSeq2;

% Psychoaccoustic Model
for channel = 'lr'

    %  - initialize
    if ( AACONFIG.L3.ON_PREV_MISSING_POLICY == L3_PSYCHO_MissingPolicies.Defer )

        frameTprev1_C = AACSeq3( 2 ).(['ch' channel]).frameT;
        frameTprev2_C = AACSeq3( 1 ).(['ch' channel]).frameT;

        frame_i_start = 3;
        deferred_frame_i = frame_i_start;
        frame_indices_sequence = [ ...
            frame_i_start ...
            1 : frame_i_start - 1 ...
            frame_i_start + 1 : NFRAMES ...
        ];

    else

        frameTprev1_C = zeros( FRAME_LENGTH, 1 );
        frameTprev2_C = zeros( FRAME_LENGTH, 1 );

        frame_indices_sequence = 1 : NFRAMES;

    end
```

```
    % Reset ESH previous time frames ( used in psycho_mono() )
    frameTprev1S_C = zeros( 256, 1);
    frameTprev2S_C = zeros( 256, 1);

    %  - run
    deferred_execution = false;
    for frame_i = frame_indices_sequence

        if ( AACONFIG.DEBUG )

            sprintf( ...
                '\t- frame: #%03d ( %s )', ...
                frame_i, ...
                L1_SSC_Frametypes.getShortCode( AACSeq3( frame_i ).frameType ) ...
            )

        end

        %% Per channel staff
        %   - save frame in time
        frameT_C = AACSeq3( frame_i ).(['ch' channel]).frameT;

        %   - check if frame is EightShott
        is_esh = AACSeq3( frame_i ).frameType == L1_SSC_Frametypes.EightShort;

        %   - compute SMR
        if ( ~deferred_execution )

            % NOTICE: on ESH frames, previous frames are the last two
            % sub-frames from the last ESH frame of the encoder
            if ( is_esh )

                SMR_C = psycho( ...
                    frameT_C, ...
                    AACSeq3( frame_i ).frameType, ...
                    frameTprev1S_C, frameTprev2S_C ...
                );

            else

                SMR_C = psycho( ...
                    frameT_C, ...
                    AACSeq3( frame_i ).frameType, ...
                    frameTprev1_C, frameTprev2_C ...
                );

            end

            %   - slide Frames
            if ( is_esh )

                FRAME_LENGTH_S = FRAME_LENGTH / 8;

                fp1s_start = FRAME_LENGTH - FRAME_LENGTH_S - 447;
                fp1s_end = FRAME_LENGTH - 448;
```

```
                fp2s_start = FRAME_LENGTH - 448 - FRAME_LENGTH_S - FRAME_LENGTH_S / 2 + 1;
                fp2s_end = FRAME_LENGTH - 448 - FRAME_LENGTH_S / 2;

                frameTprev1S_C( :, 1 ) = frameT_C( fp1s_start : fp1s_end );
                frameTprev2S_C( :, 1 ) = frameT_C( fp2s_start : fp2s_end );

            else

                frameTprev2_C = frameTprev1_C;
                frameTprev1_C = frameT_C;

            end

        else

            % SMR_C retains its last value, thus 3rd frame's SMR ( for
            % left & right channel )

        end

        %   - quantize
        [ S, sfc, AACSeq3( frame_i ).(['ch' channel]).G ] = AACquantizer( ...
            AACSeq3( frame_i ).(['ch' channel]).frameF, ...
            AACSeq3( frame_i ).frameType, ...
            SMR_C ...
        );

        %   - Huffman encode
        if ( AACONFIG.L3.HUFFMAN_ENCODE )

            % encode mdcts
            [ AACSeq3( frame_i ).(['ch' channel]).stream, AACSeq3( frame_i ).(['ch'
channel]).codebook ] = ...
                encodeHuff( S, HUFFMAN_LUT );

            % encode sfcs
            if ( is_esh )

                AACSeq3( frame_i ).(['ch' channel]).sfc = strings( 8, 1 );
                for subframe_i = 1 : 8

                    % Get huffman bit-sequence
                    AACSeq3( frame_i ).(['ch' channel]).sfc( subframe_i ) = ...
                        encodeHuff( sfc( :, subframe_i ), HUFFMAN_LUT, 12 );
                end

            else

                    % Get huffman bit-sequence
                AACSeq3( frame_i ).(['ch' channel]).sfc = ...
                        encodeHuff( sfc, HUFFMAN_LUT, 12 );

            end
```

```
        else

            AACSeq3( frame_i ).(['ch' channel]).stream = S;
            AACSeq3( frame_i ).(['ch' channel]).sfc = sfc;

        end


        %% Deferred Execution ( if chosen )
        if ( AACONFIG.L3.ON_PREV_MISSING_POLICY == L3_PSYCHO_MissingPolicies.Defer )
            % On first loop's end, the 3rd frame has been processed and
            % thus, its SMR can be used for the deferred execution of the 2
            % preceding frames.
            % So, at this point, we activate the
            % deferred execution flag and set the frame index variable to
            % the last deferred frame, gradually decrementing its value
            % until 1st deferred frame has been processed.
            % After that, we deactivate this flag and let the coder proceed
            % with all unprocessed frames

            deferred_frame_i = deferred_frame_i - 1;
            deferred_execution = deferred_frame_i > 0;

        end

    end

end
```

*Script 9: Level-3 Encoder ( φαίνονται η κλήση των συναρτήσεων phycho(), AACquantizer() και encodeHuff() )*

Τα αποτελέσματα ( χρόνος εκτέλεσης, SNR, datarate και συμπίεση του παραγόμενου binary αρχείου ) της εκτέλεσης τους στο δοσμένο αρχείου ήχου φαίνονται ακολούθως:

i. Με βάση την δική μας υλοποίηση του MDCT – χωρίς Huffman:

```
>> demoAAC3( 'sample.wav', 'sample_out.wav', 'sample.myaac', ConfSets.Default );

Level 3 (no Huffman)
====================
Coding: time elapsed is 26.04149 seconds
Decoding: time elapsed is 0.50398 seconds
      -> total time: 26.54710 seconds
Channel 1 SNR: 7.7575 dB
Channel 2 SNR: 7.5119 dB
      -> mean SNR: 7.6357 seconds
```

*Script 10: Αποτελέσματα εκτέλεσης του Level-3 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την δική μας υλοποίηση του MDCT - χωρίς Huffman*

### ii. Με βάση την δική μας υλοποίηση του MDCT – με Huffman:

```
>> demoAAC3( 'sample.wav', 'sample_out.wav', 'sample.myaac', ConfSets.Default_Huffman );

Level 3
=======
Coding: time elapsed is 73.62558 seconds
Decoding: time elapsed is 1.69966 seconds
      -> total time: 75.32708 seconds
Uncompressed audio: 1.0795 MB (9,055,648 bits)
Compressed struct : 180.0645 KB (1,475,088 bits)
      -> Compression ratio : 16.2891 % (x 6.1391)
Channel 1 SNR: 7.7575 dB
Channel 2 SNR: 7.5119 dB
      -> mean SNR: 7.6357 dB
```

*Script 11: Αποτελέσματα εκτέλεσης του Level-3 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την δική μας υλοποίηση του MDCT - με Huffman*

### iii. Με βάση την υλοποίηση του Marios Athineos – χωρίς Huffman:

```
>> demoAAC2( 'sample.wav', 'sample_out.wav', 'sample.myaac', ConfSets.Marios );

Level 3 (no Huffman)
====================
Coding: time elapsed is 26.10422 seconds
Decoding: time elapsed is 0.52560 seconds
      -> total time: 26.63158 seconds
Channel 1 SNR: 7.7575 dB
Channel 2 SNR: 7.5119 dB
      -> mean SNR: 7.6357 dB
```

*Script 12: Αποτελέσματα εκτέλεσης του Level-3 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την υλοποίηση MDCT του Marios Athineos – χωρίς Huffman*

### iv. Με βάση την υλοποίηση του Marios Athineos – με Huffman:

```
>> demoAAC3( 'sample.wav', 'sample_out.wav', 'sample.myaac', ConfSets.Marios_Huffman );

Level 3
=======
Coding: time elapsed is 65.47157 seconds
Decoding: time elapsed is 1.67995 seconds
      -> total time: 67.15306 seconds
Uncompressed audio: 1.0795 MB (9,055,648 bits)
Compressed struct : 180.0645 KB (1,475,088 bits)
      -> Compression ratio : 16.2891 % (x 6.1391)
Channel 1 SNR: 7.7575 dB
Channel 2 SNR: 7.5119 dB
      -> mean SNR: 7.6357 dB
```

*Script 13: Αποτελέσματα εκτέλεσης του Level-3 AAC Codec που υλοποιήθηκε ( το SNR σε dB ) με βάση την υλοποίηση MDCT του Marios Athineos - με Huffman*

Το ηχητικό αποτέλεσμα από τον Level-3 codec είναι δύσκολο να διακριθεί από το αρχικό αρχείο ήχου, τουλάχιστο στα 2.1 stereo ηχεία του laptop όπου έγιναν οι δοκιμές.

Σημείωση:

Στον αρχικό ( root ) κατάλογο του κώδικα της εργασίας βρίσκεται η συνάρτηση *register_config()*. Η συνάρτηση αυτή αναλαμβάνει να θέσει αρκετές παραμέτρους που διαφοροποιούν την εκτέλεση των διαφόρων levels του AAC Codec που υλοποιήθηκε. Τα configuration sets ( *ConfSets* ) που φαίνονται παραπάνω είναι στην ουσία σύνολα διαφορετικών επιλογών στην συνάρτηση αυτή.



*Θεσσαλονίκη, 17/02/2019*