ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Μελέτη σχεδίαση και κατασκευή ψηφιακού συνθετητή – synthesizer μέσω μικροελεγκτή-microcontroller

Του φοιτητή
Γεωργίου Δημητρίου
Αρ. Μητρώου: 518025

Επιβλέπων
Χατζόπουλος Αργύριος
Αναπ. Καθηγητής

Ημερομηνία 08-09-2026

Τίτλος Δ.Ε. Μελέτη σχεδίαση και κατασκευή ψηφιακού συνθετητή–synthesizer μέσω μικροελεγκτή-microcontroller

Κωδικός Π.Ε. 22195

Ονοματεπώνυμο φοιτητή/των: Γεωργίου Δημήτριος

Ονοματεπώνυμο εισηγητή: Χατζόπουλος Αργύριος

Ημερομηνία ανάληψης Π.Ε. 12-11-2023

Ημερομηνία περάτωσης Π.Ε. 08-09-2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

«Στους ανθρώπους που με στήριξαν σε όλη αυτή τη διαδρομή»

Πρόλογος

Η επιλογή της Πτυχιακής Εργασίας προέκυψε κυρίως από την επιθυμία μου να σχεδιάσω και να κατασκευάσω ένα ολοκληρωμένο σύστημα. Η παραγωγή και επεξεργασία του ήχου αποτέλεσε ενδιαφέρον πεδίο εξερεύνησης, καθώς πάντοτε έβρισκα ενδιαφέρον την ηλεκτρονική μουσική, χωρίς ωστόσο να διαθέτω ιδιαίτερες γνώσεις σχετικά με τη μουσική σύνθεση και τη δομή ενός ηλεκτρονικού μουσικού οργάνου.

Παράλληλα, η ενασχόληση μου με τον προγραμματισμό αποτέλεσε κίνητρο για τη σύνδεση του με τον σχεδιασμό ηλεκτρονικών κυκλωμάτων και την εξοικείωση μου με μικροελεγκτές. Η διαδικασία ανάπτυξης του ψηφιακού συνθετητή από το στάδιο του σχεδιασμού έως την τελική κατασκευή αποτέλεσε σημαντική εμπειρία, καθώς συνέβαλε στην αντιμετώπιση και επίλυση πραγματικών προβλημάτων που προκύπτουν κατά τη μετατροπή μιας ιδέας σε λειτουργικό πρωτότυπο.

Περίληψη

Η παρούσα Πτυχιακή εργασία αφορά τον σχεδιασμό και την κατασκευή ενός ψηφιακού συνθετητή με την χρήση μικροελεγκτή. Το ολοκληρωμένο σύστημα συνδυάζει την αναπαραγωγή μουσικών νοτών, την επεξεργασία του ήχου σε πραγματικό χρόνο και τον έλεγχο των ηχητικών παραμέτρων από τον χρήστη. Για την υλοποίηση του συστήματος κατασκευάστηκαν ηλεκτρονικά υποσυστήματα, τα οποία περιλαμβάνουν το κύκλωμα τροφοδοσίας, τα πλήκτρα επιλογής νοτών, τον μικροελεγκτή ESP32, εξωτερικό μετατροπέα ψηφιακού σήματος σε αναλογικό, στάδιο ενίσχυσης ηχείο και διεπαφή χρήστη. Τα ηχητικά δεδομένα προετοιμάστηκαν αρχικά στο MATLAB και αποθηκευτήκαν σε κατάλληλη μορφή για την αναπαραγωγή τους. Η παραγωγή του ηχητικού σήματος πραγματοποιείται με δειγματοληπτική σύνθεση και συχνότητα δειγματοληψίας 32kHz. Επιπλέον εφαρμόζονται τεχνικές επεξεργασίας δεδομένων σε πραγματικό χρόνο για τη διαμόρφωση του τελικού ήχου. Στο τελικό σύστημα υλοποιήθηκαν επτά διαφορετικοί ηχητικοί χαρακτήρες καθώς και δύο επιπλέον εφέ, με τις αντίστοιχες παραμέτρους να ελέγχονται από τον χρήστη.

Συνολικά, το σύστημα επιβεβαίωσε τη δυνατότητα υλοποίησης ενός λειτουργικού ψηφιακού μουσικού οργάνου που να συνδυάζει ηλεκτρονικό σχεδιασμό, ενσωματωμένο προγραμματισμό και ψηφιακή επεξεργασία ήχου.

«Desing, Development and Implementation of a digital Synthesizer based on a microcontroller»

«Georgiou Dimitrios»

Abstract

This thesis focuses on the design and development of a digital synthesizer using a microcontroller. The complete system combines the reproduction of musical notes, real-time audio processing, and user control of sound parameters. The implementation consists of several electronic subsystems, including the power supply circuit, the note selection keys, the ESP32 microcontroller, an external digital to analog converter, a speaker amplifier stage and a user interface. The audio data was initially processed in MATLAB and saved in a format suitable for playback. The audio signal is generated using sample-based synthesis at a sampling rate of 32kHz. In addition, real time data processing techniques are applied to shape the final sound. The final system implemented seven different sound characters as well as two additional effects, with the parameters controlled by the user.

Overall, the system confirmed the feasibility of creating a functional digital musical instrument that combines electronic design, embedded programming, and digital audio processing.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω θερμά τον επιβλέποντα καθηγητή μου κ. Χατζόπουλο Αργύριο για την καθοδήγηση και τις συμβουλές του καθ' όλη τη διάρκεια της εργασίας. Επίσης οφείλω να ευχαριστήσω την οικογένεια μου και την σύντροφο μου για την υποστήριξη που μου πρόσφεραν.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

Δ.Ε.	Διπλωματική Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Π.Ε.	Πτυχιακή Εργασία
VCO	Voltage Controlled Oscillator
VCF	Voltage Controlled Filter
FM	Frequency Modulation
DFT	Discrete Fourier Transform
FFT	Fast Fourier Transform
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
SoC	System on Chip
ROM	Read Only Memory
RAM	Random Access Memory
WAV	Waveform Audio File Format
DC	Direct Current
LFO	Low Frequency Oscillator

Κεφάλαιο 1ο: Εισαγωγή και εξέλιξη των συνθετητών ήχου

1.1 Εισαγωγή

Η εξέλιξη της ηλεκτρονικής και ψηφιακής τεχνολογίας επηρέασε σημαντικά τον τρόπο παραγωγής και επεξεργασίας του μουσικού ήχου. Χαρακτηριστικό παράδειγμα αποτελεί ο συνθετητής ήχου (synthesizer), ένα ηλεκτρονικό μουσικό όργανο που επιτρέπει τη δημιουργία και τη διαμόρφωση διαφορετικών ηχοχρωμάτων μέσω ποικίλων τεχνικών σύνθεσης[1]. Στο κεφάλαιο αυτό παρουσιάζεται συνοπτικά η εξέλιξη των synthesizer από τις πρώτες αναλογικές κατασκευές έως τα ψηφιακά συστήματα, καθώς και οι βασικές τεχνικές σύνθεσης ήχου που αναπτύχθηκαν κατά την εξέλιξη τους.

1.2 Ιστορική εξέλιξη των synthesizers

Η ανάπτυξη των synthesizers συνδέεται άμεσα με την εξέλιξη της ηλεκτρονικής τεχνολογίας κατά τη διάρκεια του 20ου αιώνα. Οι πρώτες προσπάθειες ηλεκτρονικής παραγωγής μουσικών ήχων βασίζονταν σε αναλογικά ηλεκτρονικά κυκλώματα και χρησιμοποιούσαν στοιχεία όπως λυχνίες κενού και ηλεκτρονικούς ταλαντωτές[1]. Οι κατασκευές αυτές αποτελούσαν τη βάση για την ανάπτυξη των μεταγενέστερων αναλογικών synthesizers.

Σημαντικό στάδιο στην εξέλιξη των ηλεκτρονικών μουσικών οργάνων αποτέλεσαν οι συνθετητές που αναπτύχθηκαν από τον Robert Moog κατά τη δεκαετία του 1960. Η αρχιτεκτονική τους βασιζόταν στη συνεργασία διαφορετικών αναλογικών υποσυστημάτων, όπως ταλαντωτών ελεγχόμενων από τάση (VCO), φίλτρων ελεγχόμενων από τάση (VCF) και γεννητριών φακέλων (Envelope Generators)[2]. Η δυνατότητα μεταβολής των παραμέτρων αυτών επέτρεπε τη δημιουργία και τη δυναμική διαμόρφωση διαφορετικών ηχοχρωμάτων.

Η εμφάνιση του Minimoog στις αρχές της δεκαετίας του 1970 συνέβαλε σημαντικά στην ευρύτερη διάδοση των synthesizers. Σε αντίθεση με τα μεγάλα αρθρωτά συστήματα της προηγούμενης περιόδου, συνδύαζε τα βασικά τμήματα σύνθεσης και το μουσικό πληκτρολόγιο σε μία περισσότερο συμπαγή και πρακτική κατασκευή[3]. Η εξέλιξη αυτή διευκόλυνε την ενσωμάτωση του synthesizer τόσο στις ζωντανές εμφανίσεις όσο και στις μουσικές παραγωγές.

Εικόνα 1.1: Το Minimoog[1]

Παράλληλα με την εξέλιξη των αναλογικών συστημάτων, η ανάπτυξη των ψηφιακών ηλεκτρονικών και των μικροεπεξεργαστών δημιούργησε νέες δυνατότητες για την παραγωγή μουσικού ήχου. Κατά τις επόμενες δεκαετίες άρχισαν να εμφανίζονται συστήματα στα οποία μέρος ή το σύνολο της διαδικασίας σύνθεσης πραγματοποιούνταν μέσω αριθμητικής επεξεργασίας ψηφιακών δεδομένων.

Χαρακτηριστικό σταθμό στην εμπορική διάδοση της ψηφιακής σύνθεσης αποτέλεσε το Yamaha DX7, το οποίο παρουσιάστηκε το 1983 και αξιοποίησε τη σύνθεση μέσω διαμόρφωσης συχνότητας (FM synthesis)[4][5]. Η δυνατότητα δημιουργίας σύνθετων φασματικά ήχων με ψηφιακή επεξεργασία διαφοροποίησε σημαντικά το ηχητικό αποτέλεσμα από τις κλασικές αναλογικές αφαιρετικές αρχιτεκτονικές και συνέβαλε στην ευρεία χρήση των ψηφιακών synthesizers στη μουσική παραγωγή.

Εικόνα 1.2: Yamaha DX7[5]

Η εξέλιξη της υπολογιστικής ισχύος και των ψηφιακών συστημάτων επέτρεψε στη συνέχεια την ανάπτυξη διαφορετικών τεχνικών σύνθεσης, καθώς και τη χρήση αποθηκευμένων κυματομορφών και πραγματικών ηχητικών δειγμάτων. Σήμερα, η παραγωγή ψηφιακού ήχου δεν περιορίζεται σε εξειδικευμένα εμπορικά μουσικά όργανα, αλλά μπορεί να υλοποιηθεί και μέσω προγραμματιζόμενων ενσωματωμένων συστημάτων και μικροελεγκτών, γεγονός που επιτρέπει την ανάπτυξη μικρότερων και οικονομικότερων αυτόνομων κατασκευών.

1.3 Αναλογικοί και ψηφιακοί synthesizers

Τα synthesizer μπορούν να διακριθούν, ως προς τον τρόπο παραγωγής και επεξεργασίας του σήματος, σε αναλογικούς και ψηφιακούς. Παρόλο που ο τελικός στόχος είναι κοινός, δηλαδή η δημιουργία και διαμόρφωση ενός ηχητικού σήματος, η διαδικασία με την οποία επιτυγχάνεται διαφέρει σημαντικά.

Στους αναλογικούς synthesizer, η δημιουργία και η επεξεργασία των κυματομορφών πραγματοποιείται μέσω ηλεκτρονικών κυκλωμάτων. Οι ταλαντωτές παράγουν βασικές περιοδικές κυματομορφές, ενώ ενισχυτές και γεννήτριες φίλτρων χρησιμοποιούνται για τη μεταβολή του φασματικού και χρονικού περιεχομένου του παραγόμενου σήματος. Το ηχητικό σήμα παραμένει αναλογικό κατά τη διαδρομή του μέσα στα βασικά στάδια επεξεργασίας[2].

Στους ψηφιακούς synthesizer, αντίθετα, το σήμα αναπαρίσταται μέσω αριθμητικών τιμών. Η κυματομορφή μπορεί να υπολογίζεται αλγοριθμικά ή να προέρχεται από αποθηκευμένα ψηφιακά δεδομένα, ενώ οι διάφορες επεξεργασίες πραγματοποιούνται μέσω μαθηματικών πράξεων. Για την τελική αναπαραγωγή του ήχου απαιτείται η μετατροπή των ψηφιακών τιμών σε αναλογικό ηλεκτρικό σήμα, το οποίο στη συνέχεια μπορεί να ενισχυθεί και να οδηγηθεί σε κατάλληλο σύστημα αναπαραγωγής[6].

Η ψηφιακή προσέγγιση παρέχει τη δυνατότητα υλοποίησης διαφορετικών μεθόδων σύνθεσης και επεξεργασίας μέσα στην ίδια υπολογιστική πλατφόρμα. Η συμπεριφορά του συστήματος μπορεί να τροποποιηθεί κυρίως μέσω του λογισμικού, χωρίς να απαιτείται αντίστοιχη μεταβολή της φυσικής δομής των ηλεκτρονικών κυκλωμάτων[6]. Το χαρακτηριστικό αυτό είναι ιδιαίτερα σημαντικό στην παρούσα εργασία, καθώς διαφορετικές λειτουργίες παραγωγής και διαμόρφωσης του ήχου πραγματοποιούνται από τον ίδιο μικροελεγκτή.

1.4 Βασικές τεχνικές σύνθεσης ήχου

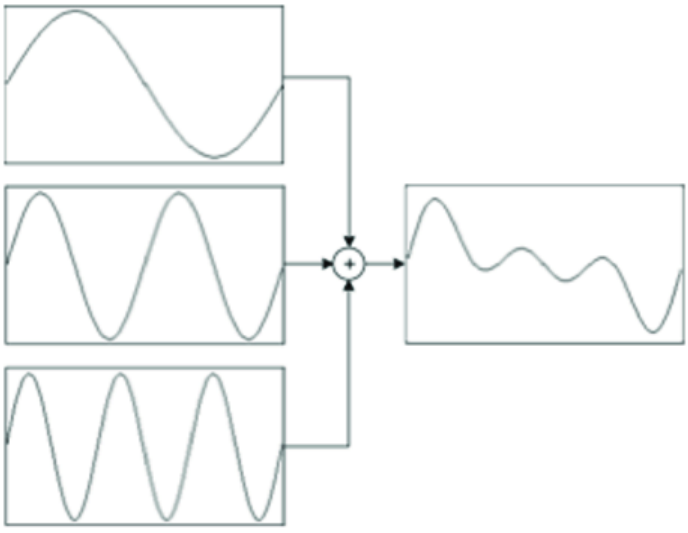
Η σύνθεση ήχου περιλαμβάνει ένα σύνολο τεχνικών μέσω των οποίων δημιουργούνται ή διαμορφώνονται ηχητικές κυματομορφές. Η κάθε τεχνική ακολουθεί διαφορετική προσέγγιση ως προς τη δημιουργία του φασματικού και χρονικού περιεχομένου του ήχου. Στις σημαντικότερες μεθόδους περιλαμβάνονται η προσθετική σύνθεση, η αφαιρετική σύνθεση, η σύνθεση μέσω διαμόρφωσης συχνότητας και η σύνθεση φυσικού μοντέλου. Παράλληλα, στα ψηφιακά συστήματα χρησιμοποιούνται τεχνικές που βασίζονται στην αποθήκευση και αναπαραγωγή ψηφιακών κυματομορφών ή ηχητικών δειγμάτων.

1.4.1 Προσθετική σύνθεση

Η προσθετική σύνθεση (additive synthesis) βασίζεται στη δημιουργία ενός σύνθετου ήχου μέσω του αθροίσματος απλούστερων ημιτονοειδών συνιστωσών διαφορετικής συχνότητας, πλάτους και φάσης, ανάλογα με την εφαρμογή του συστήματος[6]. Η αρχή της συνδέεται με την ανάλυση Fourier, σύμφωνα με την οποία, κάθε περιοδικό σήμα μπορεί να εκφραστεί ως ένα άθροισα απείρου αριθμού ημιτονοειδών συνιστωσών.

Μία από τις συνιστώσες αντιστοιχεί συνήθως στη θεμελιώδη συχνότητα του ήχου, η οποία έχει και την μεγαλύτερη ένταση του σήματος, ενώ οι υπόλοιπες μπορούν να αποτελούν αρμονικές της. Μεταβάλλοντας το πλάτος και τη συχνότητα των επιμέρους συνιστωσών είναι δυνατό να διαμορφωθεί το συνολικό φάσμα και να επηρεαστεί η χροιά του παραγόμενου ήχου.

Η μέθοδος παρέχει σημαντικό έλεγχο στο φασματικό περιεχόμενο, αλλά η παραγωγή σύνθετων ηχοχρωμάτων μπορεί να απαιτεί μεγάλο αριθμό επιμέρους ταλαντωτών ή αντίστοιχων ψηφιακών υπολογισμών.



Εικόνα 1.3: Απεικόνιση προσθετικής σύνθεσης[7]

1.4.2 Αφαιρετική σύνθεση

Η αφαιρετική σύνθεση (subtractive synthesis) ακολουθεί την αντίστροφη λογική. Η διαδικασία ξεκινά από μία κυματομορφή με σχετικά πλούσιο φασματικό περιεχόμενο και στη συνέχεια χρησιμοποιούνται φίλτρα για την εξασθένηση και την απομάκρυνση συγκεκριμένων περιοχών συχνοτήτων.

Στην κλασσική αναλογική υλοποίηση, ένας ταλαντωτής μπορεί να παράγει κυματομορφές όπως πριονωτή, τετραγωνική ή παλμική, οι οποίες περιλαμβάνουν πολλαπλές αρμονικές συνιστώσες. Το παραγόμενο σήμα οδηγείται στη συνέχεια σε φίλτρο, του οποίου παράμετροι όπως η συχνότητα αποκοπής και ο συντονισμός μεταβάλλουν το φάσμα του ήχου[2].

Η αφαιρετική σύνθεση αποτελεί βασική τεχνική των αναλογικών synthesizers και παραμένει ιδιαίτερα διαδεδομένη και σε ψηφιακές υλοποιήσεις, όπου οι αντίστοιχες λειτουργείες των ταλαντωτών και των φίλτρων προσομοιώνονται μέσω λογισμικού.

1.4.3 Σύνθεση διαμόρφωσης συχνότητας - FM

Η σύνθεση μέσω διαμόρφωσης συχνότητας (FM), βασίζεται στην αλληλεπίδραση δύο ή περισσότερων ταλαντωτών. Στην απλή μορφή της, ένας ταλαντωτής λειτουργεί ως φορέας (carrier), ενώ ο δεύτερος λειτουργεί ως διαμορφωτής (modulator). Το σήμα του διαμορφωτή μεταβάλει στιγμιαία τη συχνότητα του φέροντος, με αποτέλεσμα τη δημιουργία πρόσθετων φασματικών συνιστωσών[8].

Το τελικό φάσμα επηρεάζεται σημαντικά από τη σχέση μεταξύ των συχνοτήτων των δύο ταλαντωτών και από το βάθος της διαμόρφωσης. Με σχετικά μικρό αριθμό ταλαντωτών μπορούν επομένως να παραχθούν σύνθετα και δυναμικά μεταβαλλόμενα ηχοχρώματα.

Η εφαρμογή της FM σύνθεσης στην ψηφιακή παραγωγή μουσικού ήχου συνδέθηκε ιδιαίτερα με το έργο του John Chowning και αργότερα με την εμπορική αξιοποίηση της υλοποίησης του από την Yamaha. Χαρακτηριστικό εγχείρημα αποτελεί η δημιουργία του DX7 που αναλύθηκε στην ενότητα 1.2[4].

1.4.4 Σύνθεση φυσικού μοντέλου

Η σύνθεση φυσικού μοντέλου (physical modelling synthesis) επιδιώκει την παραγωγή ήχου μέσω μαθηματικής περιγραφής των φυσικών μηχανισμών που συμμετέχουν στη λειτουργία ενός μουσικού οργάνου. Αντί να χρησιμοποιείται αποκλειστικά μία προκαθορισμένη κυματομορφή, μοντελοποιούνται στοιχεία όπως η πηγή διέγερσης, η ταλάντωση και συντονισμός του φυσικού συστήματος.

Χαρακτηριστικό παράδειγμα αποτελεί ο αλγόριθμος Karplus-Strong, ο οποίος μπορεί να χρησιμοποιηθεί για τη σύνθεση ήχων με χαρακτηριστικά χορδών μέσω μιας σχετικά απλής δομής καθυστέρησης και ανάδρασης[8].

Το βασικό πλεονέκτημα της φυσικής μοντελοποίησης είναι η δυνατότητα δυναμικής μεταβολής των παραμέτρων του μοντέλου και συνεπώς της συμπεριφοράς του παραγόμενου ήχου. Ωστόσο, η πολυπλοκότητα και οι υπολογιστικές απαιτήσεις εξαρτώνται σημαντικά από την λεπτομέρεια του φυσικού μοντέλου.

1.4.5 Σύνθεση με κυματομορφές και ηχητικά δείγματα

Στα ψηφιακά συστήματα παραγωγής ήχου είναι δυνατό η κυμματομορφή να μην παράγεται εξ ολοκλήρου σε πραγματικό χρόνο από έναν μαθηματικό ταλαντωτή, αλλά να αποθηκεύεται προηγουμένως στη μνήμη ως μία ακολουθία αριθμητικών τιμών. Κατά την αναπαραγωγή, το σύστημα διαβάζει διαδοχικά τις τιμές αυτές και δημιουργεί τ χρονική εξέλιξη του ψηφιακού ηχητικού σήματος.

Στην wavetable synthesis, μια ή περισσότερες ψηφιακές κυματομορφές αποθηκεύονται σε πίνακες μνήμης. Η παραγωγή του ήχου πραγματοποιείται μέσω ανάγνωσης των κατάλληλων θέσεων του πίνακα, ενώ η ταχύτητα και τρόπος με τον οποίο μεταβάλλεται η θέση ανάγνωσης καθορίζουν τη συχνότητα του παραγόμενου σήματος. Σε περισσότερο σύνθετες υλοποιήσεις μπορούν να χρησιμοποιούνται πολλαπλοί πίνακες και να πραγματοποιείται μετάβαση ή παρεμβολή μεταξύ διαφορετικών κυματομορφών[6].

Μία συγγενική προσέγγιση είναι η δειγματοληπτική σύνθεση (sample based synthesis), στην οποία τα αποθηκευμένα δεδομένα προέρχονται από ψηφιακές ηχογραφήσεις πραγματικών μουσικών οργάνων. Αντί να δημιουργείται εξαρχής το φασματικό περιεχόμενο ενός οργάνου, χρησιμοποιείται ένα καταγεγραμμένο ηχητικό δείγμα ως βάση για την παραγωγή του ήχου[9]. Το δείγμα μπορεί στη συνέχεια να υποστεί επεξεργασία ως προς το πλάτος, τη διάρκεια, το τονικό ύψος και τη χροιά.

Οι δυο τεχνικές παρουσιάζουν κοινά χαρακτηριστικά ως προς την αποθήκευση και την ανάγνωση ψηφιακών δεδομένων, ωστόσο, διαφέρουν ως προς τη φύση και τη χρήση τους. Στην πρώτη περίπτωση χρησιμοποιούνται αποθηκευμένες κυματομορφές, οι οποίες λειτουργούν ως βάση για την παραγωγή του ηχητικού σήματος και μπορούν να αναπαράγονται με διαφορετικό ρυθμό ή να συνδυάζονται μεταξύ τους. Στη δεύτερη περίπτωση χρησιμοποιούνται ψηφιακές ηχογραφήσεις πραγματικών ή συνθετικών ήχων, διατηρώντας μεγαλύτερο μέρος των χρονικών και φασματικών χαρακτηριστικών του αρχικού ηχητικού σήματος.

Κατά την αναπαραγωγή αποθηκευμένων κυματομορφών ή δειγμάτων, είναι επίσης δυνατό να εφαρμοστούν τεχνικές όπως η επανάληψη συγκεκριμένων τμημάτων (looping), η μεταβολή του ρυθμού ανάγνωσης και η παρεμβολή μεταξύ διαδοχικών τιμών. Οι τεχνικές αυτές επιτρέπουν την προσαρμογή της διάρκειας και του τονικού ύψους του παραγόμενου ήχου, καθώς και την ομαλότερη ανακατασκευή του ψηφιακού σήματος.

1.5 Επίλογος

Στο κεφάλαιο αυτό παρουσιάζεται συνοπτικά η εξέλιξη των synthesizers από τις πρώτες αναλογικές κατασκευές έως τα σύγχρονα ψηφιακά συστήματα, καθώς και οι βασικές τεχνικές που χρησιμοποιούνται για τη σύνθεση του ήχου. Η μετάβαση στην ψηφιακή τεχνολογία επέτρεψε την ανάπτυξη διαφορετικών μεθόδων παραγωγής και επεξεργασίας ηχητικών σημάτων, μεταξύ των οποίων περιλαμβάνονται και τεχνικές που βασίζονται στην αποθήκευση κυματομορφών και ηχητικών δειγμάτων. Για την κατανόηση της λειτουργίας των ψηφιακών συστημάτων σύνθεσης είναι απαραίτητη η εξέταση των βασικών αρχών του ψηφιακού ήχου, οι οποίες παρουσιάζονται στο επόμενο κεφάλαιο.

Κεφάλαιο 2ο: Βασικές αρχές ψηφιακού ήχου

2.1 Εισαγωγή

Η δημιουργία ψηφιακού ήχου βασίζεται στην αριθμητική αναπαράσταση ενός ηχητικού σήματος, ώστε να είναι δυνατή η αποθήκευση, η επεξεργασία και η αναπαραγωγή του από ψηφιακά συστήματα. Για την κατανόηση της διαδικασίας αυτής είναι απαραίτητη η εξέταση τόσο των βασικών χαρακτηριστικών του ήχου όσο και εννοιών όπως η δειγματοληψία, η κβάντιση και η μετατροπή μεταξύ αναλογικών και ψηφιακών σημάτων. Στο κεφάλαιο αυτό παρουσιάζονται οι βασικές αρχές του ψηφιακού ήχου που αποτελούν το θεωρητικό υπόβαθρό για την ανάπτυξη ενός ψηφιακού συστήματος σύνθεσης.

2.2 Βασικά χαρακτηριστικά του ήχου

Ήχος ορίζεται η ενέργεια που παράγεται από την κίνηση ενός σώματος που βρίσκεται σε αδράνεια, μέσα σε ένα ελαστικό μέσο, όπως ο αέρας[6]. Οι μεταβολές της πίεσης του ατμοσφαιρικού αέρα, μπορούν να περιγράφουν ως συνάρτηση του χρόνου και να αναπαρασταθούν με τη μορφή κυματομορφής. Στην περίπτωση ενός ηλεκτρονικού συστήματος ήχου, η κυματομορφή αντιστοιχεί σε ηλεκτρική τάση η οποία μεταβάλλεται χρονικά ανάλογα με το ηχητικό περιεχόμενο.

Για την ανάλυση ενός ηχητικού σήματος χρησιμοποιούνται χαρακτηριστικά όπως η συχνότητα, το πλάτος, το φασματικό περιεχόμενο και η χροιά. Ο συνδυασμός των χαρακτηριστικών αυτών καθορίζει τον τρόπο με τον οποίο γίνεται αντιληπτός ο ήχος, επιτρέποντας την μελέτη και την επεξεργασία του από μαθηματικά μοντέλα[9].

2.2.1 Συχνότητα και περίοδος

Η συχνότητα εκφράζεται ως το πλήθος των επαναλήψεων μιας περιόδου συναρτήσει του χρόνου, η οποία μετριέται σε Hertz(Hz). Για ένα περιοδικό ηχητικό σήμα, η συχνότητα συνδέεται με την περίοδο T, δηλαδή τον χρόνο που απαιτείται για την ολοκλήρωση ενός πλήρους κύκλου:

$$f = \frac{1}{T} \tag{2.1}$$

όπου f είναι η συχνότητα σε Hz και T η περίοδος σε δευτερόλεπτα[6][9].

Ένας ήχος μπορεί να αποτελείται από κυματισμούς διαφόρων συχνοτήτων γι' αυτό, η θεμελιώδης συχνότητα, σχετίζεται άμεσα με το αντιλαμβανόμενο τονικό ύψος. Όσο αυξάνεται η θεμελιώδης συχνότητα, ο ήχος γίνεται αντιληπτός ως υψηλότερος, ενώ η μείωση της αντιστοιχεί σε χαμηλότερο τονικό ύψος. Για παράδειγμα η νότα Λα4 (A4) ενός πιάνου, ενώ αποτελείται από πολλές παράγωγες συχνότητες, η θεμελιώδης είναι τα 440Hz. Ωστόσο ο διπλασιασμός της θεμελιώδους συχνότητας προσφέρει τονικό ύψος μια οκτάβα ψιλότερα, την νότα Λα5(A5).

2.2.2 Πλάτος και ένταση

Το πλάτος ενός ηχητικού σήματος περιγράφει το μέγεθος της μεταβολής του σε σχέση με μία θέση αναφοράς. Στην ηλεκτρική αναπαράσταση του ήχου, μεγαλύτερο πλάτος της κυματομορφής αντιστοιχεί γενικά σε μεγαλύτερο επίπεδο του παραγόμενου ηχητικού σήματος[6].

2.2.3 Φάσμα και αρμονικές

Ένα ηχητικό σήμα μπορεί να εξεταστεί τόσο στο πεδίο του χρόνου όσο και στο πεδίο της συχνότητας. Η χρονική αναπαράσταση περιγράφει τη μεταβολή του πλάτους ως προς τον χρόνο, ενώ η φασματική αναπαράσταση παρουσιάζει τις επιμέρους συχνότητες που περιέχονται στο σήμα[6].

Οι περιοδικοί μουσικοί ήχοι, αποτελούνται από την θεμελιώδη συχνότητα και τις παράγωγες. Παράλληλα μπορούν να υπάρχουν συχνότητες οι οποίες είναι ακέραια πολλαπλάσια της θεμελιώδους και ονομάζονται, αρμονικές. Η σχετική στάθμη των αρμονικών, καθώς και η μεταβολή τους στον χρόνο επηρεάζουν σημαντικά τα χαρακτηριστικά του παραγόμενου ήχου.

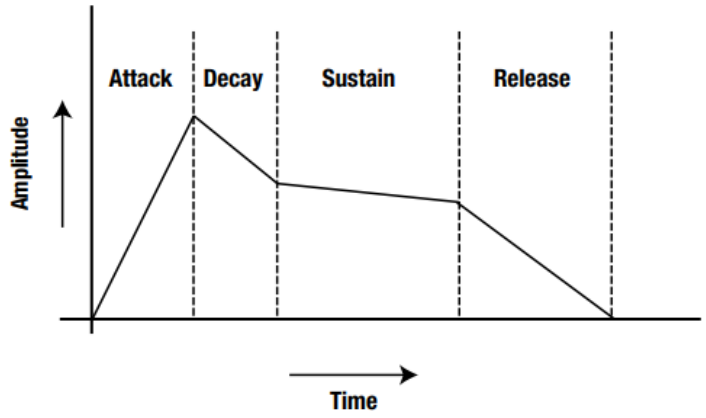
Η μετάβαση από τη χρονική στη συχνοτική περιγραφή ενός σήματος μπορεί να πραγματοποιηθεί μαθηματικά μέσω της ανάλυσης Fourier. Στα ψηφιακά συστήματα χρησιμοποιείται ευρέως ο Διακριτός Μετασχηματισμός Fourier (DFT) και η αποδοτικότερη υλοποίηση του, ο Γρήγορος Μετασχηματισμός Fourier (FFT), για την ανάλυση του φασματικού περιεχομένου ψηφιακών σημάτων[8].

2.2.4 Χροιά και τονικό ύψος

Το τονικό ύψος αποτελεί αντιληπτικό χαρακτηριστικό που συνδέεται κυρίως με τη θεμελιώδη συχνότητα ενός μουσικού ήχου. Δύο νότες διαφορετικών μουσικών οργάνων μπορούν να έχουν το ίδιο τονικό ύψος, χωρίς όμως να ακούγονται ίδιες[10].

Η διαφοροποίηση αυτή σχετίζεται με τη χροιά του ήχου. Η χροιά επηρεάζεται από το φασματικό περιεχόμενο, τις σχετικές στάθμες των αρμονικών συνιστωσών και τη χρονική εξέλιξη του σήματος. Για τον λόγο αυτό, διαφορετικά μουσικά όργανα που έχουν την ίδια θεμελιώδη συχνότητα μπορούν να παρουσιάζουν διαφορετικό ηχόχρωμα.

Η χρονική εξέλιξη του πλάτους αποτελεί επίσης σημαντικό χαρακτηριστικό της χροιάς. Ο χρόνος έναρξης ενός ήχου, η μεταβολή του κατά τη διάρκεια του, η συγκράτηση και ο τρόπος με τον οποίο εξασθενεί συμβάλουν στην αναγνώριση του αντίστοιχου μουσικού οργάνου[11] [6].



Εικόνα 2.1: Χρονική εξέλιξη του πλάτους- ADSR[11]

2.3 Ψηφιοποίηση του ηχητικού σήματος

Η μετατροπή ενός αναλογικού ηχητικού σήματος σε ψηφιακή μορφή απαιτεί τη μετατροπή τόσο του συνεχούς χρόνου όσο και του συνεχούς πλάτους του σε διακριτές αριθμητικές τιμές. Η διαδικασία αυτή πραγματοποιείται σε δύο βασικά στάδια: τη δειγματοληψία και την κβάντιση[2]. Οι δύο διαδικασίες καθορίζουν σε σημαντικό βαθμό την ανάλυση του αρχικού σήματος σε ψηφιακή μορφή.

2.3.1 Δειγματοληψία και συχνότητα Nyquist

Η δειγματοληψία είναι η διαδικασία όπου, τιμές ενός συνεχούς σήματος, λαμβάνονται, σε διαδοχικές χρονικές στιγμές. Όταν οι χρονικές στιγμές έχουν την ίδια απόσταση μεταξύ τους, ο αριθμός των δειγμάτων που λαμβάνονται ανά δευτερόλεπτο ονομάζεται συχνότητα δειγματοληψίας f_s και εκφράζεται σε Hz[6]. Το χρονικό διάστημα μεταξύ δυο διαδοχικών δειγμάτων αποτελεί την περίοδο δειγματοληψίας T_s και ορίζεται από την σχέση:

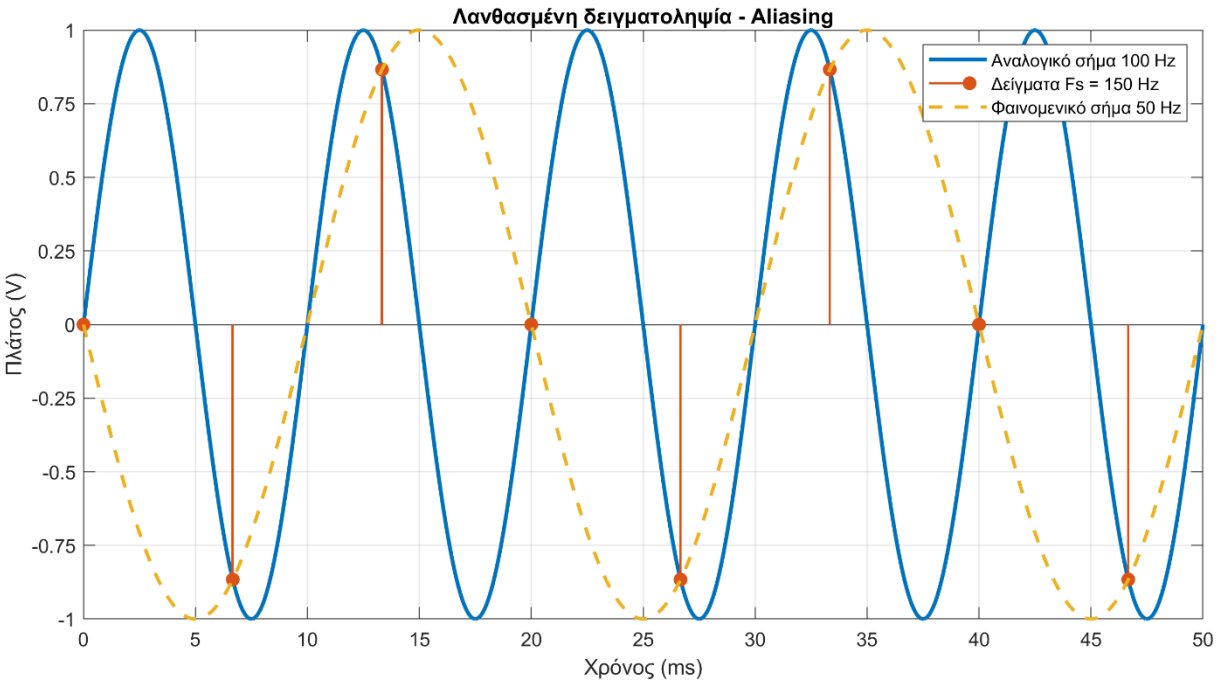
$$T_s = \frac{1}{f_s} \tag{2.2}$$

Η επιλογή της συχνότητας δειγματοληψίας συνδέεται άμεσα με το συχνοτικό περιεχόμενο που μπορεί να αναπαρασταθεί από το ψηφιακό σήμα. Σύμφωνα με το θεώρημα δειγματοληψίας Nyquist-Shannon, για την πλήρως ανακατασκευή ενός συνεχούς σήματος από τα δείγματα του, η συχνότητα δειγματοληψίας πρέπει να είναι τουλάχιστον διπλάσια της μέγιστης συχνότητας που περιέχεται στο σήμα B.

$$f_s > 2B \tag{2.3}$$

Συχνότητα Nyquist ονομάζεται η ποσότητα $f_s/2$ και καθορίζει το ανώτερο θεωρητικό όριο συχνοτήτων που μπορούν να αναπαρασταθούν για τη δεδομένη συχνότητα δειγματοληψίας[8].

Όταν ένα σήμα περιέχει συχνοτικές συνιστώσες υψηλότερες από τη συχνότητα Nyquist, εμφανίζεται το φαινόμενο της αναδίπλωσης (aliasing) με αποτέλεσμα οι συχνότητες αυτές να εμφανιστούν μετά τη δειγματοληψία ως διαφορετικές, χαμηλότερες συχνότητες. Το φαινόμενο αυτό προκαλεί ανεπιθύμητη αλλοίωση στο ψηφιακό σήμα, καθιστώντας δύσκολη την ανακατασκευή του αρχικού. Για τον λόγο αυτό, πριν από τη δειγματοληψία χρησιμοποιείται συνήθως ένα χαμηλοπερατό φίλτρο πριν από τη διαδικασία της δειγματοληψίας, ώστε να περιορίζει κατάλληλα το φάσμα του σήματος κάτω από τη συχνότητα Nyquist[8].



Εικόνα 2.2: Παράδειγμα λανθασμένης δειγματοληψίας για αναλογικό σήμα 100Hz

2.3.2 Κβάντιση και ανάλυση bit

Μετά τη δειγματοληψία, οι τιμές των δειγμάτων που αναλογούν στο πλάτος του σήματος ήχου, μπορούν θεωρητικά να λάβουν οποιαδήποτε τιμή μέσα σε ένα συνεχές εύρος. Η διαδικασία κατά την οποία κάθε δείγμα του σήματος αντιστοιχίζεται στην πλησιέστερη τιμή από ένα πεπερασμένο σύνολο προκαθορισμένων στάθμεων ονομάζεται Κβάντιση του ήχου[6].

Ο αριθμός των διαθέσιμων επιπέδων κβάντισης εξαρτάται από την ανάλυση σε bit. Για ανάλυση N bit, ο συνολικός αριθμός των δυνατών στάθμεων είναι:

$$L = 2^N \tag{2.4}$$

Επομένως, μια αύξηση στον αριθμό των bit, αυξάνει τον αριθμό των διαθέσιμων επιπέδων με τα οποία μπορεί να αναπαρασταθεί το πλάτος του σήματος. Για παράδειγμα, μία αναπαράσταση 8 bit διαθέτει 256 διαφορετικές στάθμες, ενώ μια αναπαράσταση 16bit διαθέτει 65.536 στάθμες.

Επειδή η πραγματική τιμή ενός δείγματος στρογγυλοποιείται σε ένα από τα διαθέσιμα επίπεδα, κατά την κβάντιση προκύπτει μία διαφορά μεταξύ της αρχικής και της κβαντισμένης τιμής. Η διαφορά αυτή ονομάζεται σφάλμα κβάντισης. Η αύξηση της ανάλυσης μειώνει το διάστημα μεταξύ διαδοχικών επιπέδων και επιτρέπει ακριβέστερη αναπαράσταση του αρχικού σήματος, αυξάνοντας παράλληλα τον αριθμό των bit που απαιτούνται για κάθε δείγμα.

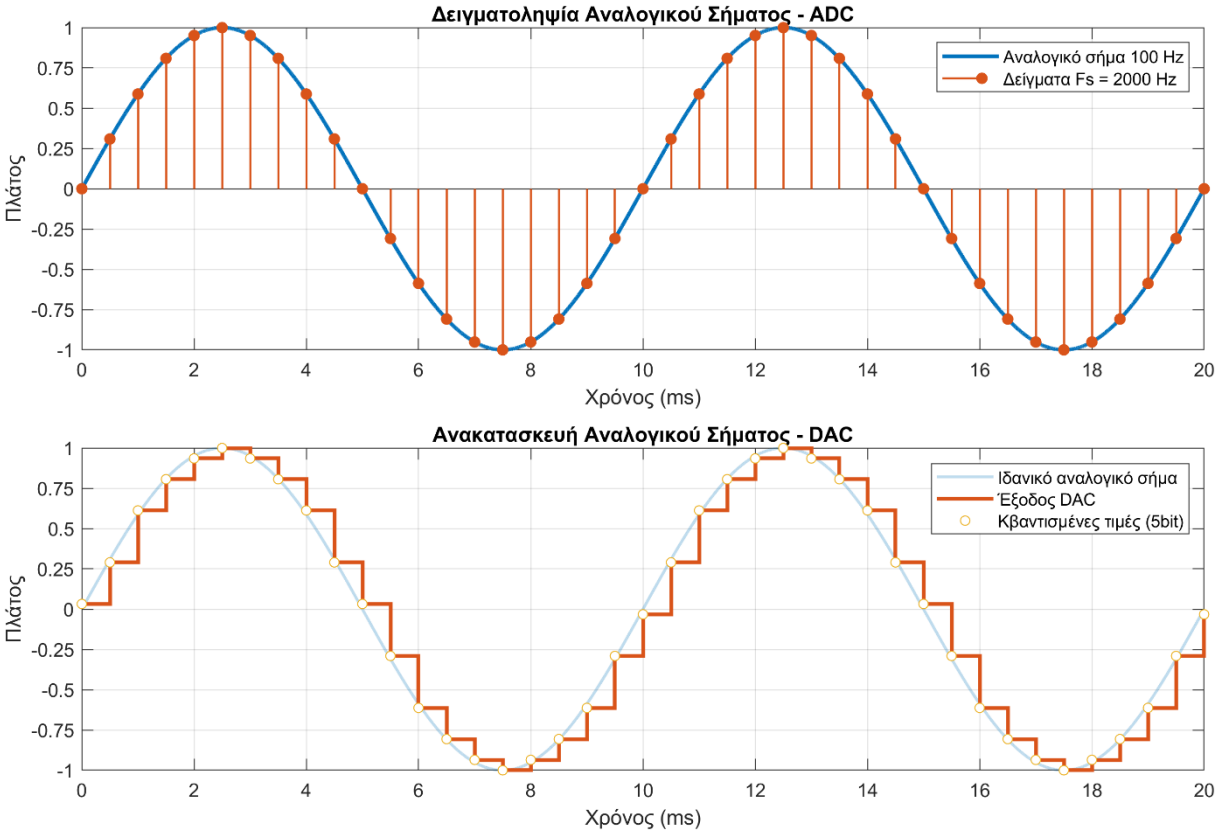
2.4 Μετατροπή αναλογικού και ψηφιακού σήματος

Η επεξεργασία ενός ηχητικού σήματος από ένα ψηφιακό σύστημα απαιτεί τη δυνατότητα μετατροπής μεταξύ αναλογικών και ψηφιακών μεγεθών. Η μετατροπή από αναλογική σε ψηφιακή μορφή πραγματοποιείται μέσω ενός μετατροπέα ADC, ενώ για την αντίστροφη διαδικασία χρησιμοποιείται ένας μετατροπέας DAC. Οι δύο βαθμίδες αποτελούν τη σύνδεση μεταξύ των συνεχών ηλεκτρικών σημάτων και της αριθμητικής αναπαράστασης.

Η λειτουργία ενός μετατροπέα αναλογικού σε ψηφιακό σήμα (ADC) ξεκινά από την εφαρμογή μιας τάσης στην είσοδο του. Η τάση αυτή συγκρίνεται με το διαθέσιμο εύρος δείγματος του μετατροπέα και αντιστοιχίζεται σε μία από τις στάθμες κβάντισης που επιτρέπει η ανάλυση του. Για παράδειγμα ένας ιδανικός ADC ανάλυσης 12 bit διαθέτει 4096 διαφορετικές στάθμες. Εάν το εύρος εισόδου αντιστοιχεί από 0 έως V_{REF} , μία τάση κοντά στο μηδέν αντιστοιχίζεται στις χαμηλότερες στάθμες κβάντισης, ενώ μια τάση που πλησιάζει το άνω όριο στις υψηλότερες. Η ακρίβεια της μετατροπής επηρεάζεται από την ανάλυση του μετατροπέα, γι’ αυτό είναι σημαντικό να επιλέγεται όσο μεγαλύτερη δυνατή τιμή σε bits, με αποτέλεσμα την πιστή αναπαράσταση του αρχικού σήματος[11][12].

Αντίθετα, ένας μετατροπέας ψηφιακού σε αναλογικό σήμα (DAC), λαμβάνει την ψηφιακή πληροφορία από τον επεξεργαστή και την μετατρέπει ξανά σε συνεχή τάση. Κατά την ανακατασκευή του σήματος, ο μετατροπέας διατηρεί την τάση εξόδου του, μέχρι την εφαρμογή του επόμενου διαδοχικού δείγματος. Η διαδικασία επαναλαμβάνεται με σταθερό ρυθμό, σχηματίζοντας ένα χαρακτηριστικό ‘σκαλοπάτι’ στην κυματομορφή, δημιουργώντας ένα μεταβαλλόμενο αναλογικό σήμα. Η ποιότητα ενός DAC επηρεάζεται από την ανάλυση του, την γραμμικότητα του και το φιλτράρισμα που ενδέχεται να ακολουθεί για την ομαλότερη αναπαράσταση του επιθυμητού αναλογικού σήματος[8][13].

Συμπερασματικά, η διαδικασίες ADC και DAC αποτελούν καθοριστικό στάδιο σε κάθε ψηφιακό σύστημα ήχου. Η ανάλυση των μετατροπέων, η συχνότητα δειγματοληψίας και η ομαλή ανακατασκευή του σήματος, επηρεάζουν άμεσα την ποιότητα της αναπαραγωγής. Επομένως, η τελική ποιότητα του ήχου βασίζεται στην σωστή επιλογή της κατάλληλης ανάλυσης και συχνότητας δειγματοληψίας.



Εικόνα 2.3: Παράδειγμα προσομοίωσης μετατροπής ADC και DAC

2.5 Αποθήκευση και αναπαραγωγή ψηφιακών δειγμάτων

Μετά την ψηφιοποίηση, ένα ηχητικό σήμα μπορεί να αναπαρασταθεί ως μία διαδοχική ακολουθία αριθμητικών τιμών, όπου κάθε τιμή αντιστοιχεί στο πλάτος του σήματος σε μία συγκεκριμένη χρονική στιγμή. Η μορφή αυτή επιτρέπει την αποθήκευση των ηχητικών δεδομένων σε ψηφιακή μνήμη και την επεξεργασία τους μέσω μαθηματικών πράξεων. Για την ορθή αναπαραγωγή του σήματος, τα αποθηκευμένα δείγματα πρέπει να ανακτώνται με καθορισμένο χρονικό ρυθμό, ο οποίος σχετίζεται με τη συχνότητα δειγματοληψίας.

Η ποσότητα μνήμης που είναι αναγκαία για την αποθήκευση ενός ηχητικού σήματος χωρίς συμπίεση, εξαρτάται από τη συχνότητα δειγματοληψίας, την ανάλυση σε bit, τον αριθμό των καναλιών και τη διάρκεια του. Ο συνολικός αριθμός των απαιτούμενων θέσεων μνήμης μπορεί να εκφραστεί από τη σχέση:

$$D = f_s \cdot N \cdot C \cdot t \tag{2.5}$$

Όπου f_s είναι η συχνότητα δειγματοληψίας, N η ανάλυση σε bit ανά δείγμα, C ο αριθμός των καναλιών και t η διάρκεια του ηχητικού σήματος[14]. Επομένως, η αύξηση οποιασδήποτε παραμέτρου αυξάνει τις απαιτήσεις αποθήκευσης, γεγονός που αποκτά ιδιαίτερη σημασία σε ψηφιακά συστήματα με περιορισμένη διαθέσιμη μνήμη.

Μια βοηθητική διαδικασία στο στάδιο της αποθήκευσης είναι αυτή της παρεμβολής (interpolation). Κατά την δειγματοληψία του σήματος, επιλέγονται διαδοχικά δείγματα με σταθερό ρυθμό. Έπειτα κατά την ανακατασκευή του σήματος χρησιμοποιείται η μέθοδος της παρεμβολής η οποία, έχοντας ως σταθερά, τον διαδοχικό ρυθμό των δειγμάτων, δημιουργεί ένα νέο δείγμα με βάση τα δύο γειτονικά του. Αυτή η διαδικασία ονομάζεται γραμμική παρεμβολή, ωστόσο υπάρχουν και μέθοδοι υψηλότερης τάξης,

όπως η κυβική παρεμβολή, που χρησιμοποιούν περισσότερα γειτονικά δείγματα, προσφέροντας ομαλότερη μετάβαση μεταξύ των διαθέσιμων τιμών. Η μέθοδος της παρεμβολής χρησιμοποιείται κυρίως, σε εφαρμογές που διαθέτουν μειωμένη αποθηκευτική μνήμη και για την τροποποίηση της συχνότητας δειγματοληψίας χωρίς την αποθήκευση νέων δειγμάτων[15].

2.6 Ψηφιακή επεξεργασία ήχου σε πραγματικό χρόνο

Η ψηφιακή επεξεργασία σήματος DSP ορίζεται από την μορφοποίηση των ψηφιακών σημάτων μέσω μαθηματικών και αλγοριθμικών διαδικασιών. Στις εφαρμογές ήχου χρησιμοποιείται για τη μεταβολή ή την ανάλυση των χαρακτηριστικών ενός ηχητικού σήματος και περιλαμβάνει λειτουργίες όπως το φιλτράρισμα, η μεταβολή του πλάτους, η παρεμβολή και η εφαρμογή ηχητικών εφέ[6].

Όταν η επεξεργασία πραγματοποιείται σε πραγματικό χρόνο, το σύστημα πρέπει να μπορεί να επεξεργάζεται τα δεδομένα με τον ρυθμό τον οποίο απαιτείται η αναπαραγωγή τους, χωρίς να επηρεάζεται από τον όγκο των αλγοριθμικών διαδικασιών. Για συχνότητα δειγματοληψίας f_s , το χρονικό διάστημα μεταξύ διαδοχικών δειγμάτων δίνεται από τη σχέση (2.2)[16].

Επομένως, οι απαραίτητο οι υπολογισμοί πρέπει να ολοκληρώνονται μέσα στους χρονικούς περιορισμούς που επιβάλλει η συχνότητα δειγματοληψίας. Με αποτέλεσμα, η επεξεργασία σε πραγματικό χρόνο να απαιτεί κατάλληλη υπολογιστική ισχύ και σταθερό χρονισμό, ώστε να διατηρείται συνεχής η ροή των ηχητικών δεδομένων.

2.7 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν οι βασικές αρχές που διέπουν την αναπαράσταση και την επεξεργασία του ήχου από ψηφιακά συστήματα. Αναλύθηκαν τα βασικά χαρακτηριστικά του ηχητικού σήματος, η διαδικασία μετατροπής του από αναλογική σε ψηφιακή μορφή, καθώς και οι αρχές αποθήκευσης, αναπαραγωγής και ψηφιακής επεξεργασίας ηχητικών δεδομένων. Οι έννοιες αυτές αποτελούν το απαραίτητο θεωρητικό υπόβαθρο για την κατανόηση των απαιτήσεων ενός ψηφιακού συστήματος παραγωγής ήχου και των λειτουργιών που καλείται να εκτελέσει η υπολογιστική μονάδα που το ελέγχει. Στο επόμενο κεφάλαιο παρουσιάζεται ο μικροελεγκτής που θα χρησιμοποιηθεί και οι επιμέρους δυνατότητες του που αξιοποιούνται για την υλοποίηση του συστήματος.

Κεφάλαιο 3ο: Μικροελεγκτής ESP32 και αρχιτεκτονική του συστήματος

3.1 Εισαγωγή

Η λειτουργία ενός ψηφιακού synthesizer απαιτεί μία υπολογιστική μονάδα ικανή να διαχειρίζεται τις εισόδους του συστήματος, τα αποθηκευμένα δεδομένα και την επεξεργασία του ήχου σε πραγματικό χρόνο. Στην παρούσα κατασκευή τον κεντρικό αυτό ρόλο αναλαμβάνει ο μικροελεγκτής ESP32. Στο κεφάλαιο αυτό παρουσιάζονται τα βασικά χαρακτηριστικά και οι περιφερειακές μονάδες που χρησιμοποιούνται στην εφαρμογή, ενώ στη συνέχεια καθορίζονται οι απαιτήσεις και η συνολική αρχιτεκτονική του ψηφιακού συνθετητή.

3.2 Ο μικροελεγκτής ESP32

Ο ESP32 αποτελεί οικογένεια ολοκληρωμένων συστημάτων τύπου SoC, συστήματα μέσα σε ένα τσιπ, της εταιρίας Espressif Systems, σχεδιασμένων για εφαρμογές ενσωματωμένων συστημάτων. Στο ίδιο ολοκληρωμένο κύκλωμα συνδυάζονται η κεντρική μονάδα επεξεργασίας, η μνήμη και ένα σύνολο από περιφερειακές μονάδες για κάθε βασική λειτουργία. Η χρήση ενός SoC επιτρέπει τη δημιουργία συστημάτων που προσφέρουν βελτιωμένη απόδοση, είναι ταχύτερα και παράλληλα έχουν μικρότερο φυσικό μέγεθος σε σύγκριση με τα παραδοσιακά συστήματα πολλαπλών τσιπ[17].

Η αρχιτεκτονική του ESP32 παρέχει τόσο ψηφιακές όσο και αναλογικές εισόδους και εξόδους, καθώς και περιφερειακά για χρονισμό και σειριακή επικοινωνία. Μεταξύ αυτών περιλαμβάνονται:

- **Μετατροπείς ADC και DAC**
- **Χρονοδιακόπτες (hardware timers)**
- **Πρωτόκολλα επικοινωνίας SPI και I^2C**
- **GPIO, ακροδέκτες γενικού σκοπού**

Η λειτουργία των περιφερειακών συστημάτων μπορεί να ελεγχθεί μέσω λογισμικού, επιτρέποντας στον μικροελεγκτή να αλληλοεπιδρά με διαφορετικά εξωτερικά ηλεκτρονικά υποσυστήματα.

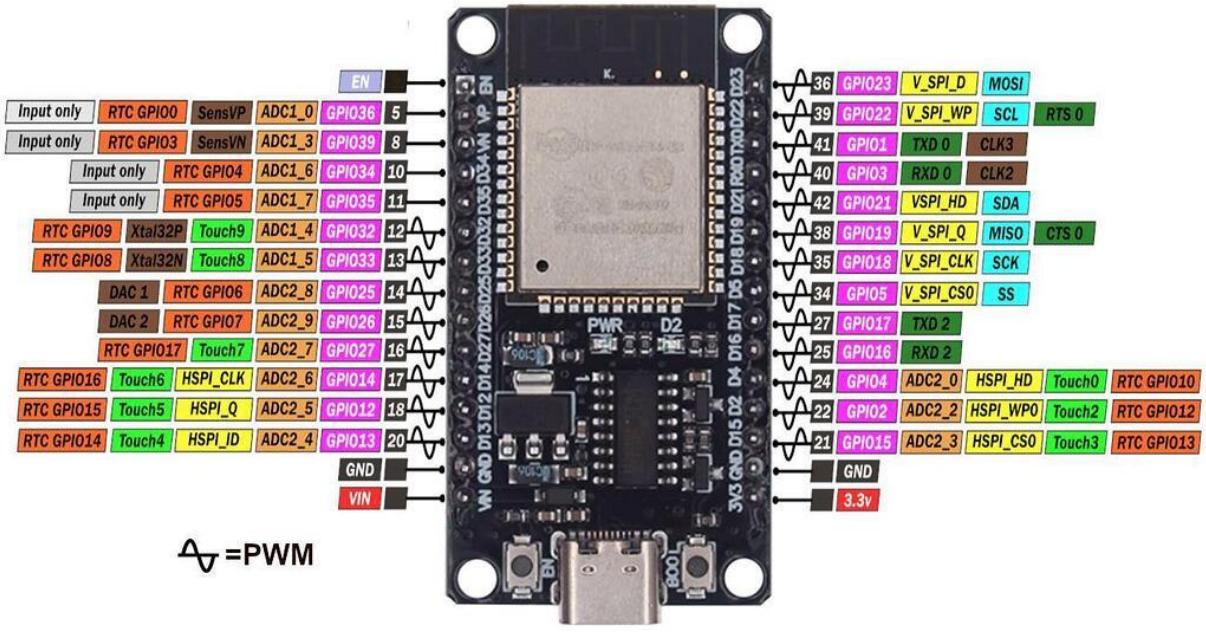
Ένα ακόμη χαρακτηριστικό του ESP32 είναι η ενσωμάτωση ασύρματων δυνατοτήτων Wi-Fi και Bluetooth. Οι δυνατότητες αυτές αποτελούν βασικά χαρακτηριστικά της συγκεκριμένης πλατφόρμας, χωρίς ωστόσο να είναι απαραίτητες για τη λειτουργία του συστήματος που εξετάζεται στην παρούσα Π.Ε. Για τη συγκεκριμένη εφαρμογή μεγαλύτερη σημασία παρουσιάζουν η υπολογιστική δυνατότητα, η διαχείριση μνήμης και τα ενσωματωμένα περιφερειακά που απαιτούνται για την ανάγνωση εισόδων, τον χρονισμό και την επικοινωνία με εξωτερικές συσκευές.

3.3 Αναπτυξιακή πλακέτα ESP32 DEVKIT V4

Η Espressif Systems κατασκευάζει αναπτυξιακές πλακέτες, παρέχοντας μια ολοκληρωμένη πλατφόρμα για την εκμάθηση και την ανάπτυξη εφαρμογών. Αυτές οι πλακέτες περιλαμβάνουν όλα τα απαραίτητα εξαρτήματα για την λειτουργία του ESP32, όπως θύρα micro USB για την άμεση τροφοδοσία και προγραμματισμό του επεξεργαστή, ενσωματωμένη κεραία για τις ασύρματες λειτουργίες, επιπλέον εξωτερική μνήμη αποθήκευσης, καθώς και εύκολη πρόσβαση με ακίδες στις GPIO του συστήματος.

Για την υλοποίηση του συγκεκριμένου συστήματος, θα χρησιμοποιηθεί η αναπτυξιακή πλακέτα ESP32 DEVKIT V4 της Espressif, η οποία ενσωματώνει το ESP-WROOM-32D module. Είναι η πιο βασική έκδοση της σειράς , η οποία υπάρχει σε αφθονία στην αγορά, πράγμα που την καθιστά πολύ οικονομική, ενώ παράλληλα , προσφέρει όλα τα απαραίτητα χαρακτηριστικά για την ανάπτυξη του synthesizer[12].

- **Επεξεργαστής:** Ο ESP-WROOM-32D μικροελεγκτής χρησιμοποιεί έναν διπύρηνο επεξεργαστή της Xtensa, LX6 των 32bit που μπορεί να φτάσει ταχύτητες έως 240Mhz και έναν συνεπεξεργαστή χαμηλότερης ισχύος. Ο πρώτος χρησιμοποιείται για τις κύριες λειτουργίες του συστήματος, ενώ ο δεύτερος χαμηλότερης ισχύος για την παρακολούθηση των GPIOs και για την λειτουργία βαθιάς αδράνειας, προσφέροντας εξοικονόμηση ενέργειας. Επιπλέον, υποστηρίζονται λειτουργίες όπως, Floating Point Unit (FPU), για γρήγορες αριθμητικές πράξεις, Digital Signal Processing (DSP), για ανάλυση ψηφιακών σημάτων και άμεση πρόσβαση στους καταχωρητές των περιφερειακών με την χρήση Xtensa RAM/ROM και Xtensa Local Memory.
- **Εσωτερική Μνήμη:** Το συγκεκριμένο μοντέλο διαθέτει 448KB ROM μνήμη, για τις για τις βασικές λειτουργίες και την εκκίνηση του πηγαίου κώδικα και 520KB SRAM όπου χρησιμοποιείται για την προσωρινή αποθήκευση δεδομένων και εντολών κατά την εκτέλεση του προγράμματος.
- **Εξωτερική Μνήμη:** Αν και ονομάζεται εξωτερική, στην πραγματικότητα ο επεξεργαστής έχει πρόσβαση στην QSPI Flash μέσω cache υψηλής ταχύτητας. Η μνήμη Flash είναι ενσωματωμένη στο σύστημα, λειτουργεί ως ROM, διαθέτει συνολικό χώρο αποθήκευσης 4MB και χρησιμοποιείται για την αποθήκευση του κυρίου προγράμματος. Το σύστημα διαθέτει επίσης επιπλέον SRAM που σχετίζεται με το RTC, για να εξυπηρετεί ειδικές λειτουργίες χαμηλής κατανάλωσης.
- **Τροφοδοσία:** Η αναπτυξιακή πλακέτα μπορεί να τροφοδοτηθεί από την θύρα micro USB ή από εξωτερική σταθερή τροφοδοσία 5V. Και στις δυο περιπτώσεις η τάση υποβιβάζεται μέσω ενός ενσωματωμένου γραμμικού ρυθμιστή πτώσης τάσης στα απαραίτητα 3.3V.



Εικόνα 3.1: Απεικόνιση της ESP32-WROOM-32D πλακέτας [18]

3.4 Χρήσιμες περιφερειακές μονάδες του ESP32

Εκτός από την κεντρική μονάδα επεξεργασίας και τη μνήμη, ο ESP32 ενσωματώνει περιφερειακές μονάδες που επιτρέπουν την επικοινωνία με εξωτερικά κυκλώματα, την ανάγνωση αναλογικών και ψηφιακών σημάτων και την εκτέλεση λειτουργιών με ακριβή χρονισμό. Παρακάτω αναλύονται οι περιφερειακές μονάδες που απαιτούνται για τη λειτουργία του συστήματος:

- **General Purpose Input / Output – GPIO:** Οι ακροδέκτες γενικής χρήσης μπορούν να διαμορφωθούν προγραμματιστικά ως ψηφιακές είσοδοι ή έξοδοι και χρησιμοποιούνται για την ανάγνωση ή τον έλεγχο εξωτερικών κυκλωμάτων. Ο μικροελεγκτής διαθέτει συνολικά 34 GPIO ακροδέκτες, οι οποίοι είναι αμφίδρομοι, επιτρέποντας τόσο την είσοδο όσο και την έξοδο σημάτων. Οι ακροδέκτες αυτοί, μπορούν να χρησιμοποιηθούν για την επικοινωνία με διάφορες περιφερειακές συσκευές, όπως οθόνες, πληκτρολόγια, μοτέρ.
- **Analog to Digital Converter – ADC:** Ο ESP32 διαθέτει δύο μετατροπείς αναλογικού σήματος σε ψηφιακό, ADC1 και ADC2, με έως και 18 διαφορετικά κανάλια εισόδου και υποστήριξη ανάλυσης 12bit. Μέσω αυτών είναι δυνατή η μετατροπή ενός αναλογικού σήματος που εισέρχεται στο module, σε ψηφιακή μορφή, καθιστώντας το σήμα κατανοητό από τον μικροελεγκτή [12].
- **Hardware Timers και Interrupts:** Ο ESP32 διαθέτει χρονοδιακόπτες, οι οποίοι μπορούν να χρησιμοποιηθούν για τη δημιουργία γεγονότων, τα οποία θα λειτουργούν ανεξάρτητα από τη διαδοχική εκτέλεση του κύριου προγράμματος. Όταν ένας χρονοδιακόπτης φτάσει στην προκαθορισμένη του κατάσταση, μπορεί να προκαλέσει διακοπή (interrupt), ώστε να εκτελεστεί μία συγκεκριμένη λειτουργία. Η δυνατότητα αυτή είναι χρήσιμη για εφαρμογές πραγματικού χρόνου, στις οποίες ορισμένες διαδικασίες, επαναλαμβάνονται σε σταθερά χρονικά διαστήματα.
- **Serial Peripheral Interface – SPI:** Το SPI είναι ένα ευρέως διαδεδομένο πρωτόκολλο σειριακής επικοινωνίας, που χρησιμοποιεί την αρχιτεκτονική Master - Slave. Η κύρια συσκευή (master) συντονίζει την επικοινωνία με μία ή περισσότερες περιφερειακές συσκευές (slaves). Για την ενεργοποίηση της επιθυμητής περιφερειακής συσκευής επιβάλλεται, η χρήση της γραμμής Chip Select. Η αρχιτεκτονική αυτή, προσφέρει αμφίδρομη μεταφορά δεδομένων, αλλά μόνο προς τη μία κατεύθυνση κάθε φορά, εξασφαλίζοντας έτσι την ακεραιότητα των δεδομένων. Ο ESP32 ενσωματώνει ελεγκτές SPI, επιτρέποντας την επικοινωνία με εξωτερικά ψηφιακά ολοκληρωμένα κυκλώματα, προσφέροντας υψηλές ταχύτητες μεταφοράς, έως και τα 80MHz[12][13].
- **Inter-Integrated Circuit Interface I2C:** Το I2C είναι σύγχρονο πρωτόκολλο σειριακής επικοινωνίας που χρησιμοποιεί μόνο δύο καλώδια για τη μεταφορά δεδομένων, την γραμμή δεδομένων (SDA) και τη γραμμή χρονισμού (SCL). Η μεταφορά των δεδομένων δεν είναι αμφίδρομη και μόνο η master συσκευή μπορεί να ξεκινήσει τη μεταφορά δεδομένων, ενώ οι slaves ανταποκρίνονται στα αιτήματα του master. Οι συσκευές που συνδέονται στον ίδιο δίαυλο διακρίνονται μέσω διευεθύνσεων, επιτρέποντας την επικοινωνία με περισσότερες από μια περιφερειακές συσκευές χωρίς να απαιτείται ξεχωριστό σύνολο γραμμών[19].

3.5 Απαιτήσεις και βασικές επιλογές σχεδίασης του συστήματος

Με βάση τις αρχές παραγωγής και επεξεργασίας ενός ψηφιακού ήχου που παρουσιάστηκαν στα προηγούμενα κεφάλαια, καθορίστηκαν οι βασικές απαιτήσεις που πρέπει να ικανοποιεί το υπό ανάπτυξη synthesizer. Οι απαιτήσεις αυτές αποτελούν την βάση για την επιλογή των επιμέρους ηλεκτρονικών υποσυστημάτων και την ανάπτυξη του λογισμικού. Συνοπτικά, το σύστημα πρέπει να παρέχει:

- **Ανίχνευση μουσικών νοτών:** δυνατότητα αναγνώρισης των διαθέσιμων πλήκτρων και αντιστοίχισης τους στις διαθέσιμες μουσικές νότες, κατά την ενεργοποίηση τους.
- **Αποθήκευση ηχητικών δειγμάτων:** δυνατότητα αποθήκευσης και ανάκτησης των ψηφιακών δεδομένων που απαιτούνται για παραγωγή των μουσικών νοτών.
- **Παραγωγή και επεξεργασία ήχου σε πραγματικό χρόνο:** τα ηχητικά δείγματα πρέπει να παράγονται με σταθερό χρονισμό και να γίνεται η ΄ψηφιακή επεξεργασία, σε πραγματικό χρόνο πριν από την έξοδο.
- **Μετατροπή του ψηφιακού σήματος σε αναλογικό:** το παραγόμενο ψηφιακό ηχητικό σήμα πρέπει να μετατρέπεται σε κατάλληλη αναλογική μορφή ώστε να μπορεί να οδηγηθεί στα επόμενα στάδια της ηχητικής αλυσίδας.

- **Ενίσχυση και αναπαραγωγή του ήχου:** το αναλογικό σήμα πρέπει να ενισχύεται σε κατάλληλο επίπεδο ισχύος για την οδήγηση του ηχείου.
- **Διεπαφή χρήστη:** πρέπει να παρέχεται η δυνατότητα επιλογής λειτουργιών και μεταβολής παραμέτρων που επηρεάζουν τον παραγόμενο ήχο.
- **Οπτική απεικόνιση:** το σύστημα πρέπει να παρέχει στον χρήστη πληροφορίες σχετικά με την επιλεγμένη λειτουργία και τις παραμέτρους που μεταβάλλονται.
- **Εξωτερική τροφοδοσία:** η επιλογή των περιφερειακών συστημάτων πρέπει να γίνει με βάση την κοινή τροφοδοσία των 5V.

Για την παραγωγή των μουσικών νοτών επιλέχθηκε ως βασική μέθοδος μία sample-based προσέγγιση, βασισμένη στην αποθήκευση ηχητικών δειγμάτων σε πίνακες δεδομένων. Η χρήση δειγμάτων πραγματικών μουσικών νοτών επιτρέπει τη διατήρηση αρκετών χαρακτηριστικών της αρχικής κυματομορφής και αποτελεί τη βάση για την αναπαραγωγή κάθε νότας. Παράλληλα, το σύστημα συνδυάζει τα αποθηκευμένα δείγματα με πρόσθετες τεχνικές ψηφιακής σύνθεσης και επεξεργασίας, όπως η χρήση ταλαντωτών, για τη διαμόρφωση διαφορετικών χροιών και ηχητικών εφέ. Η αναλυτική λειτουργία των επιμέρους τεχνικών παρουσιάζεται στο κεφάλαιο της λογισμικής υλοποίησης.

Η επιλογή της αποθήκευσης πραγματικών ηχητικών δειγμάτων δημιουργεί αυξημένες απαιτήσεις ως προς τη διαθέσιμη μνήμη, καθώς η ποσότητα των δεδομένων εξαρτάται από τη διάρκεια, την ανάλυση και τη συχνότητα δειγματοληψίας, όπως αναλύθηκε στην ενότητα 2.6. Για τον λόγο αυτό, η διαχείριση των ηχητικών δεδομένων και η χρήση τεχνικών παρεμβολής αποτελούν σημαντικά στοιχεία της λογισμικής υλοποίησης.

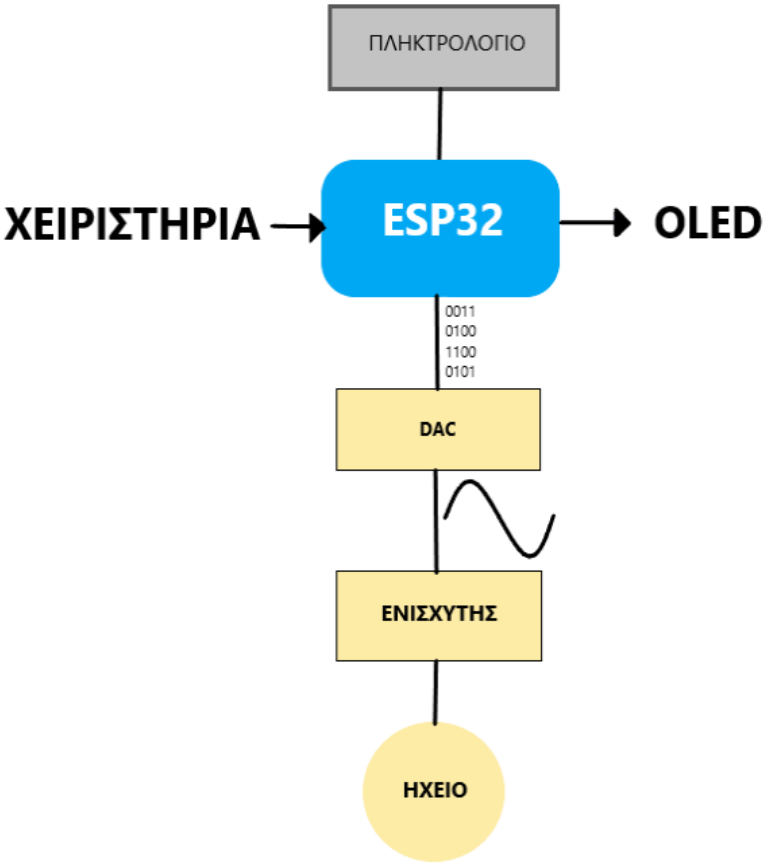
3.6 Συνολική αρχιτεκτονική του συστήματος

Με βάση τις παραπάνω απαιτήσεις, η κατασκευή οργανώθηκε σε επιμέρους υποσυστήματα, τα οποία αναλαμβάνουν συγκεκριμένο τμήμα της συνολικής λειτουργίας. Κεντρικό ρόλο στην κατασκευή αποτελεί η αναπτυξιακή πλακέτα ESP32, η οποία διαχειρίζεται τις εισόδους του χρήστη, την επεξεργασία και αναπαραγωγή των ηχητικών δεδομένων, καθώς και την επικοινωνία με τα υπόλοιπα περιφερειακά υποσυστήματα.

Η είσοδος των μουσικών νοτών πραγματοποιείται μέσω του πληκτρολογίου, ενώ πρόσθετα ποτενσιόμετρα και πλήκτρα, επιτρέπουν τη μεταβολή των διαθέσιμων παραμέτρων. Τα σήματα αυτά διαβάζονται από τον ESP32 και χρησιμοποιούνται για τον καθορισμό της λειτουργίας του συστήματος. Παράλληλα, η οθόνη παρέχει οπτική πληροφορία σχετικά με τις επιλεγμένες λειτουργίες και τις τιμές των παραμέτρων.

Για την παραγωγή του ήχου, τα κατάλληλα αποθηκευμένα δεδομένα ανακτώνται από τον ESP32 και τροποποιούνται με την απαιτούμενη ψηφιακή επεξεργασία. Τα τελικά ψηφιακά δείγματα στέλνονται σε έναν DAC, από την έξοδο του οποίου προκύπτει το αναλογικό ηχητικό σήμα. Στη συνέχεια, το σήμα οδηγείται στο στάδιο ενίσχυσης και τελικά στο ηχείο για την ακουστική αναπαραγωγή του.

Η λειτουργία των παραπάνω υποσυστημάτων ολοκληρώνεται από το κύκλωμα τροφοδοσίας, το οποίο παρέχει τις απαιτούμενες τάσεις στα επιμέρους ηλεκτρονικά μέρη της κατασκευής. ΜΕ τον τρόπο αυτό διαμορφώνεται μία ενιαία αλυσίδα από την ενέργεια του χρήστη στο ανάλογο πλήκτρο έως την τελική παραγωγή του ακουστικού σήματος.



Σχήμα 3.1: Διάγραμμα Ροής του συστήματος

3.7 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκαν τα βασικά χαρακτηριστικά της αναπτυξιακής πλακέτας ESP32-WROOM-32D και οι περιφερειακές μονάδες που αξιοποιούνται για την ανάπτυξη του ψηφιακού synthesizer. Η συνολική αρχιτεκτονική του συστήματος διαμορφώθηκε, με βάση τις απαιτήσεις και τις δυνατότητες του μικροελεγκτή για την παραγωγή και επεξεργασία ψηφιακού ήχου. Η αρχιτεκτονική αυτή διαχωρίζει την κατασκευή σε επιμέρους ηλεκτρονικά υποσυστήματα, τα οποία συνεργάζονται μεταξύ τους για την περάτωση της συνολικής ροής λειτουργίας του synthesizer.

Κεφάλαιο 4ο: Σχεδιασμός ηλεκτρονικών υποσυστημάτων

4.1 Εισαγωγή

Μετά τον καθορισμό της συνολικής αρχιτεκτονικής του συστήματος, το επόμενο στάδιο αφορά τον σχεδιασμό των επιμέρους ηλεκτρονικών υποσυστημάτων που απαιτούνται για την υλοποίηση του. Στο κεφάλαιο αυτό παρουσιάζονται τα κυκλώματα τροφοδοσίας, εισόδου και αναπαραγωγής του ήχου, καθώς και τα συστήματα διεπαφής με τον χρήστη. Για κάθε τμήμα αναλύονται τα χαρακτηριστικά των εξαρτημάτων που επιλέχθηκαν, οι απαιτήσεις που οδήγησαν στην επιλογή τους και ο αντίστοιχος ηλεκτρονικός σχεδιασμός. Η πρακτική υλοποίηση των επιμέρους κυκλωμάτων παρουσιάζονται στο επόμενο κεφάλαιο.

4.2 Κύκλωμα τροφοδοσίας

Η τροφοδοσία αποτελεί βασικό τμήμα για την σωστή λειτουργία του ψηφιακού synthesaizer, καθώς παρέχει αξιόπιστα, τις απαραίτητες τάσεις λειτουργίας στα επιμέρους κυκλώματα. Ως κυριά πηγή τροφοδοσίας χρησιμοποιείται συνεχής τάση 12V, από την οποία πρέπει να προκύψουν δύο επιπλέον επίπεδα σταθεροποιημένης τάσης, 5V και 3.3V, για την τροφοδοσία του μικροελεγκτή και των περιφερειακών συστημάτων αντίστοιχα.

Η αναπτυξιακή πλακέτα ESP32 υποστηρίζει τρείς διαφορετικούς τρόπους τροφοδοσίας. Απευθείας μέσω της θύρας micro USB, μέσω των ακροδεκτών 5V, καθώς και μέσω απευθείας τροφοδοσίας 3.3V μεγάλης ακριβείας, διότι παρακάμπτεται ο εσωτερικός ρυθμιστής τάσης[12]. Για την τροφοδοσία του ESP32 επιλέχθηκε η δεύτερη μέθοδος τροφοδοσίας, με την χρήση τροφοδοσίας 5V. Επιπλέον, η γραμμή των 5V, προορίζεται για την τροφοδοσία του μετατροπέα DAC και του ενισχυτή ήχου. Ενώ η γραμμή των 3.3V, χρησιμοποιείται για τα κυκλώματα εισόδου και διεπαφής, όπως το πληκτρολόγιο, τα ποτενσιόμετρα και η οθόνη. Η χρήση εξωτερικού κυκλώματος τροφοδοσίας, βοηθάει στην εξασφάλιση μεγαλύτερης σταθερότητας, επαρκής παροχής ρεύματος και καλύτερης απομόνωσης θορύβου μεταξύ των επιμέρους κυκλωμάτων.

Για την παραγωγή των δύο τάσεων επιλέχθηκε η χρήση δύο γραμμικών ρυθμιστών LM317 σε διαδοχική σύνδεση. Το πρώτο στάδιο υποβιβάζει την τάση από 12V σε 5V, ενώ η έξοδος του χρησιμοποιείται παράλληλα ως είσοδος του δεύτερου σταδίου, το οποίο παράγει την τάση των 3.3V:

Σχήμα 4.1: Διάγραμμα διαδοχικής σύνδεσης δύο ρυθμιστών τάσης

4.2.1 Ρυθμιστής τάσης LM317

Ο LM317 είναι ρυθμιζόμενος γραμμικός ρυθμιστής θετικής τάσης και χρησιμοποιείται ευρέως σε εφαρμογές όπου απαιτείται σταθερή τάση εξόδου. Διαθέτει εσωτερική τάση αναφοράς 1.25V μεταξύ των ακροδεκτών εξόδου (OUT) και ρύθμισης (ADJ), επιτρέποντας την διέλευση ενός σταθερού ρεύματος, για τον καθορισμό της εξόδου μέσω εξωτερικού διαιρέτη τάσης[20].

Η τάση εξόδου του LM317 προσδιορίζεται από τη σχέση:

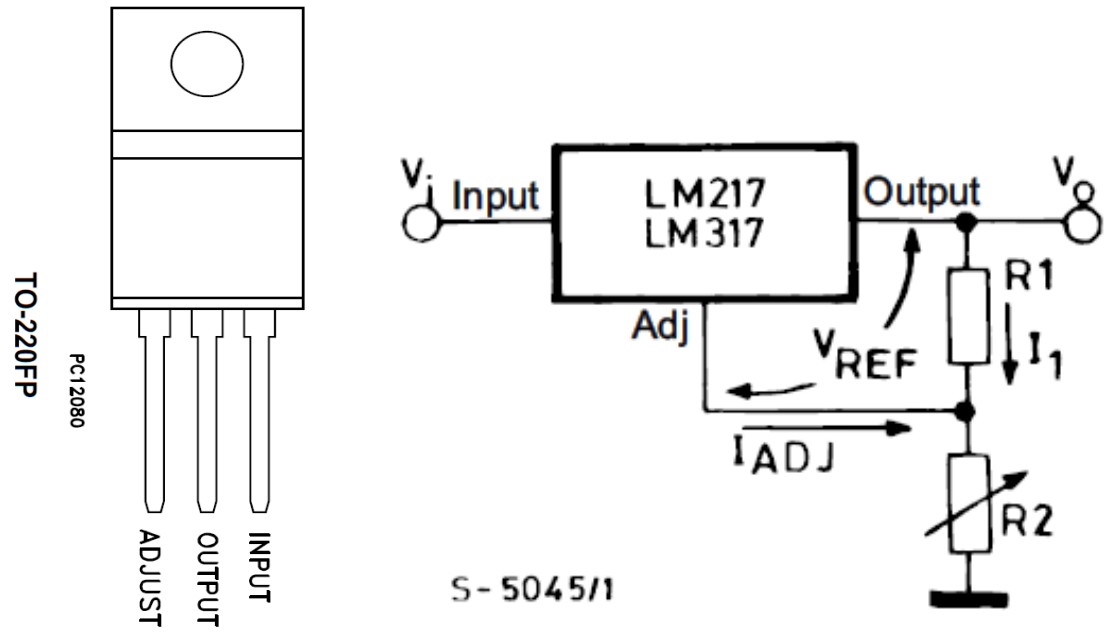
$$V_{OUT} = V_{REF}\left(1 + \frac{R_2}{R_1}\right) + I_{ADJ} \cdot R_2 \quad (4.1)$$

όπου V_{REF} είναι η εσωτερική τάση αναφοράς του ρυθμιστή και I_{ADJ} το ρεύμα του ακροδέκτη Adjust. Επειδή το I_{ADJ} είναι αρκετά μικρό και μένει σταθερό σε όλες τις μεταβολές του φορτίου, η επίδραση του μπορεί να θεωρηθεί περιορισμένη, με αποτέλεσμα η σχέση (4.1) να απλοποιείται ως:

$$V_{OUT} \approx 1{,}25V \left(1 + \frac{R_2}{R_1}\right) \quad (4.2)$$

Η λειτουργία του LM317 προϋποθέτει επίσης μια ελάχιστη διαφορά μεταξύ της τάσης εισόδου και της επιθυμητής τάσης εισόδου (dropout Voltage). Το χαρακτηριστικό αυτό είναι σημαντικό ιδιαίτερα στο δεύτερο στάδιο, όπου από την τάση των 5V παράγονται τα 3.3V.

Η χρήση πυκνωτών στην είσοδο και την έξοδο του ρυθμιστή συμβάλει στη βελτίωση της μεταβατικής απόκρισης και στη μείωση του θορύβου, ενώ η χρήση διόδων προστασίας αποτρέπει πιθανά προβλήματα που προκαλούνται από αντίστροφα ρεύματα κατά την αποφόρτιση των πυκνωτών. Τέλος, το ποτενσιόμετρο παρέχει την δυνατότητα ακριβής ρύθμισης της εξόδου.



Εικόνα 4.1: Στα δεξιά φαίνεται η διάταξη του LM317 και στα αριστερά το προτεινόμενο κύκλωμα λειτουργίας[20]

4.2.2 Σχεδιασμός του κυκλώματος τροφοδοσίας

Για την υλοποίηση του κυκλώματος τροφοδοσίας χρησιμοποιήθηκαν δύο διαδοχικά στάδια ρύθμισης τάσης με χρήση ρυθμιστών LM317, με στόχο την δημιουργία κυκλώματος υποβιβασμού ρυθμιζόμενης τάσης (adjustable step-down converter) στο οποίο παράγει τάσεις 5V και 3.3V αντίστοιχα από την αρχική τροφοδοσία των 12V[21]. Το κύκλωμα των 5V για να τροφοδοτήσει την αναπτυξιακή πλακέτα του ESP32, ρυθμίζεται αρχικά εξωτερικά και στην συνέχεια οδηγείται στον ενσωματωμένο ρυθμιστή της πλακέτας, ο οποίος την υποβιβάζει περαιτέρω στην ονομαστική τάση λειτουργίας του μικροελεγκτή, τα 3.3V.

Στην συνέχεια, η τάση των 5V τροφοδοτεί το δεύτερο στάδιο ρύθμισης, το οποίο παράγει 3.3V ταση, για την τροφοδοσία των περιφερειακών κυκλωμάτων. Κάθε στάδιο ρύθμισης χρησιμοποιεί θεωρητικά δύο αντιστάσεις, πυκνωτές εξομάλυνσης, διόδους και τον ρυθμιστής LM317. Στην περίπτωση του

πρώτου κυκλώματος, η επιθυμητή τάση εξόδου είναι 5V. Ορίζοντας την αντίσταση R_1 ίση με 470Ω και χρησιμοποιώντας την εξίσωση (3.2) της προηγούμενης ενότητας, υπολογίζεται η τιμή τη της R_2:

$$V_{OUT} = 1{,}25\left(1 + \frac{R_2}{R_1}\right) \Leftrightarrow \tag{4.3}$$

$$5V = 1{,}25V\left(1 + \frac{R_2}{470\Omega}\right) \Leftrightarrow$$

$$\frac{5V}{1{,}25V} - 1 = \frac{R_2}{470} \Leftrightarrow 3 = \frac{R_2}{470\Omega} \Leftrightarrow$$

$$3 \cdot 470\Omega = R_2 \Leftrightarrow$$

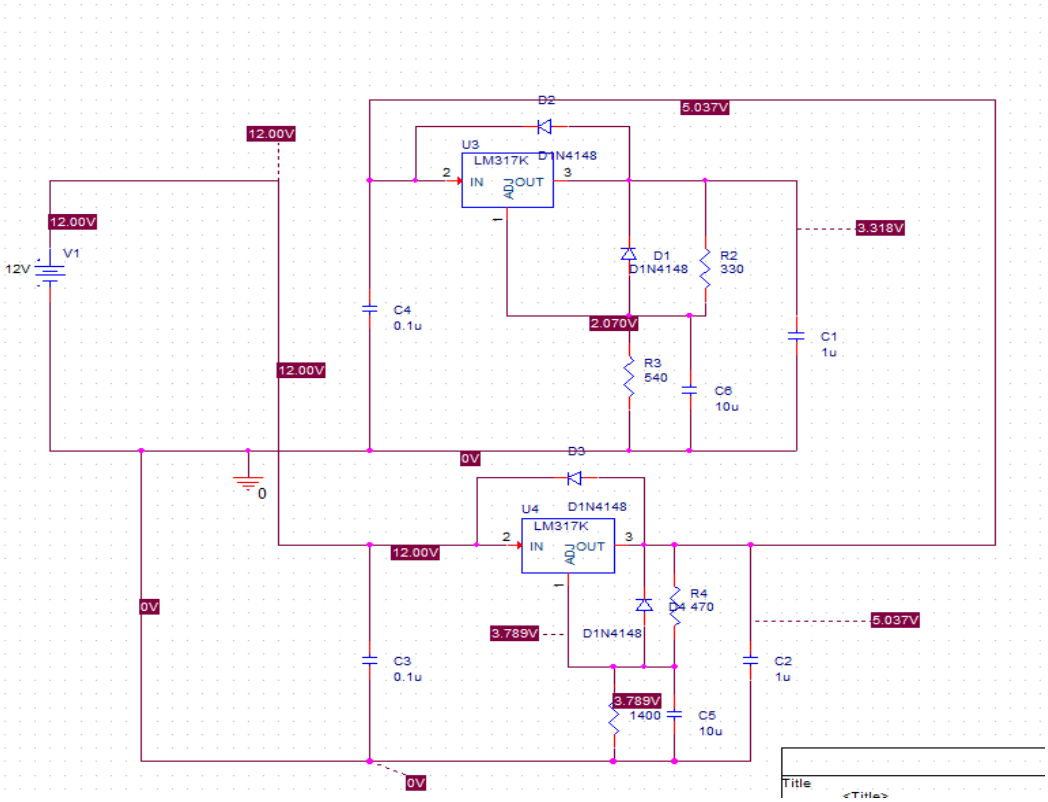
Άρα:

$$R_2 = 1410\Omega$$

Με τον ίδιο τρόπο υπολογίζεται και το δεύτερο στάδιο, όπου για επιθυμητή τάση εξόδου 3.3V, τάση εισόδου 5V και με δεδομένη την τιμή της αντίστασης R_3 ίση με 330Ω, προκύπτει η τιμή των 540Ω[21]. Άρα η τιμή της αντίστασης R_4 ισούται περίπου με 540Ω. Στα κυκλώματα τοποθετούνται επίσης πυκνωτές παράκαμψης (decoupling capacitors) στην είσοδο και την έξοδο των ρυθμιστών, 100μF και 100nF αντίστοιχα, οι οποίοι συμβάλουν στη μείωση του θορύβου και στην σταθεροποίηση της τάσης[21].

Η επιλογή της διαδοχικής (cascade) σύνδεσης των ρυθμιστών, έγινε με σκοπό τη μείωση της θερμικής καταπόνησης των εξαρτημάτων, καθώς η απευθείας πτώση τάσης από 12V σε 3.3V θα οδηγούσε σε σημαντικές απώλειες ισχύος. Για τον ίδιο λόγο επιβάλλεται η χρήση ψήκτρας στους ρυθμιστές, με στόχο τη βελτίωση της απαγωγής θερμότητας κατά τον υποβιβασμό της τάσης για τη σταθερή λειτουργία του κυκλώματος.

Πριν από την πρακτική κατασκευή, το κύκλωμα τροφοδοσίας προσομοιώθηκε στο περιβάλλον PSpice, με σκοπό να επαληθευτούν οι αναμενόμενες τάσεις εξόδου των δύο σταδίων. Σύμφωνα με την εικόνα 4.2 τα αποτελέσματα της προσομοίωσης βρίσκονται στο στις επιθυμητές τάσεις των 5.037V και 3.318V αντίστοιχα.



Εικόνα 4.2: Προσομοίωση του κυκλώματος τροφοδοσίας στο Pspice

4.2.3 Διαχείριση γειώσεων

Κατά τον σχεδιασμό της τροφοδοσίας και των διασυνδέσεων δόθηκε ιδιαίτερη προσοχή στη διαχείριση των γειώσεων, ώστε να περιοριστεί η αλληλεπίδραση μεταξύ των διαφορετικών υποσυστημάτων. Τα ψηφιακά κυκλώματα, όπως ο ESP32 , χαρακτηρίζονται από ταχείες μεταβολές στάθμης και παλμικά ρεύματα, ενώ τα αναλογικά κυκλώματα, όπως ο ενισχυτής, είναι περισσότερο ευαίσθητα σε τέτοιου είδους διαταραχές, καθώς μικρές μεταβολές της τάσης αναφοράς ενδέχεται να επηρεάσουν την ποιότητα του παραγόμενου σήματος[23].

Για τον λόγο αυτό, επιλέχθηκε ο διαχωρισμός της δρομολόγησης των αντίστοιχων αγωγών γείωσης, με διατήρηση κοινής ηλεκτρικής αναφοράς για το σύνολο του κυκλώματος. Με την πρακτική αυτή, οι διαδρομές επιστροφής του ρεύματος οργανώνονται με τέτοιο τρόπο που να αποφεύγεται η διέλευση ισχυρών ή παλμικών ρευμάτων προς τα ευαίσθητα αναλογικά μονοπάτια[24].

4.3 Πληκτρολόγιο επιλογής νοτών

Το πληκτρολόγιο αποτελεί το κύριο μέσο αλληλεπίδρασης του χρήστη με το ψηφιακό synthesizer, καθώς μέσω αυτού πραγματοποιείται η επιλογή της επιθυμητής νότας. Στην κατασκευή απαιτούνται 16 πλήκτρα, καθένα από τα οποία πρέπει να αναγνωρίζεται ανεξάρτητα από τον ESP32. Η σύνδεση κάθε πλήκτρου σε ανεξάρτητες ψηφιακές εισόδους θα περιόριζε σημαντικά τις δυνατότητες σύνδεσης των υπόλοιπων περιφερειακών του συστήματος. Επιπλέον, μια τέτοια προσέγγιση θα οδηγούσε σε περισσότερο εκτεταμένη καλωδίωση και σε πιο σύνθετη διαχείριση εισόδων σε επίπεδο λογισμικού.

Για τον λόγο αυτό επιλέχθηκε η σχεδίαση ενός αναλογικού πληκτρολογίου 16 πλήκτρων σε διάταξη 16X1, στο οποίο όλα τα πλήκτρα αναγνωρίζονται μέσο μίας μόνο αναλογικής εισόδου (ADC) του μικροελεγκτή. Η αρχή λειτουργίας βασίζεται στην παραγωγή διαφορετικής τάσης εξόδου ανάλογα με

το πλήκτρο που πιέζεται, μέσω κατάλληλης διάταξης αντιστάσεων. Με αυτόν τον τρόπο, κάθε πλήκτρο αντιστοιχεί σε διαφορετική τιμή τάσης και, κατ' επέκταση σε, σε διαφορετική τιμή ADC.

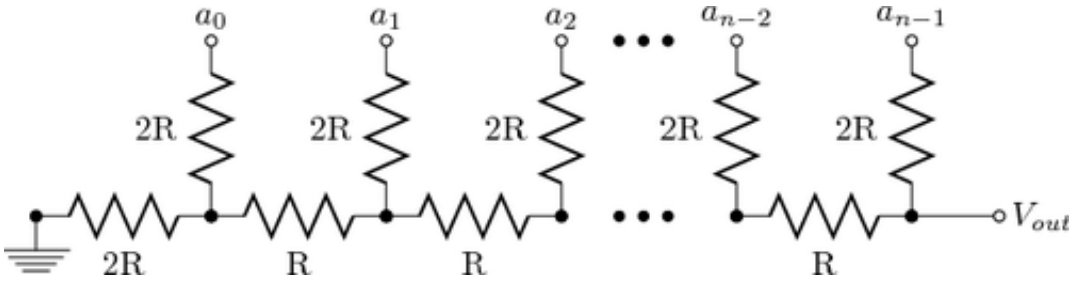
Η συγκεκριμένη σχεδίαση χρησιμοποιεί τρείς βασικούς κλάδους:

- **VCC:** κλάδος τροφοδοσίας 3.3V
- **GND:** κλάδος γείωσης
- **Vo:** κλάδος εξόδου προς την αναλογική είσοδο

Η επιλογή αυτής της μεθόδου επιτρέπει την εξοικονόμηση GPIOs, τη διατήρηση απλού κυκλώματος διεπαφής και την ικανοποιητική αναγνώριση των πλήκτρων, υπό την προϋπόθεση ότι οι τιμές των αντιστάσεων επιλέγονται κατάλληλα ώστε να προκύπτουν διακριτικές στάθμες τάσης.

4.3.1 Σχεδιασμός του κυκλώματος

Κατά την υλοποίηση του πληκτρολογίου, κάθε πλήκτρο συνδέθηκε με αντιστάσεις R ίσης τιμής, 470Ω, σχηματίζοντας μια διαδοχική αλυσίδα μεταξύ των πλήκτρων. Ενώ το κύκλωμα ολοκληρώνει η αντίσταση Rz, η οποία λειτουργεί ως αναλογική αντίσταση pull-down. Με τη διάταξη αυτή, κάθε πάτημα οδηγεί στη δημιουργία διαφορετικού διαιρέτη τάσης και επομένως σε διαφορετική τάση εξόδου προς την αναλογική είσοδο του ESP32.



Εικόνα 4.3: Πρότυπο κύκλωμα πλήκτρων σε διάταξη resistor ladder[25]

Όταν ενεργοποιείται ένα πλήκτρο, το σημείο στο οποίο πραγματοποιείται η σύνδεση, χωρίζει την αλυσίδα σε δύο τμήματα. Το τμήμα προς την τροφοδοσία σχηματίζει την αντίσταση R_{1P} , ενώ το τμήμα προς τη γείωση συνδυάζεται παράλληλα με την R_z, σχηματίζοντας την ισοδύναμη αντίσταση R_{2P} [25].

Η αντίσταση προς την πλευρά της τροφοδοσίας υπολογίζεται από τη σχέση:

$$R_{1P} = N_1 \cdot R \tag{4.5}$$

Όπου R = 470Ω είναι η τιμή κάθε αντίστασης της αλυσίδας και N_1 ο αριθμός των αντιστάσεων μεταξύ της τροφοδοσίας και του ενεργοποιημένου πλήκτρου. Αν υποθέσουμε πως το πλήκτρο A4 βρίσκεται στην θέση δεκατέσσερα του πληκτρολογίου έχει 13 αντιστάσεις προς την τροφοδοσία και 3 προς τον κλάδο της γείωσης.

Για την πλευρά προς τη γείωση, οι υπόλοιπες αντιστάσεις της αλυσίδας έχουν συνολική τιμή $N_2 \cdot R$. Η αντίσταση αυτή είναι παράλληλη με την Rz = 47KΩ, επομένως:

$$R_{2P} = \frac{(N_2 \cdot R) R_Z}{N_2 \cdot R + R_Z} \tag{4.6}$$

Σύμφωνα με τις σχέσεις (4.5) και (4.6) η τάση εξόδου του πληκτρολογίου προκύπτει στη συνέχεια από τον διαιρέτη τάσης:

$$V_o = V_{CC} \frac{R_{2P}}{R_{1P} + R_{2P}} \tag{4.7}$$

Έτσι, για κάθε πλήκτρο υπολογίζονται οι αντίστοιχες τιμές, για τη θεωρητική τάση εξόδου προς τον μετατροπέα ADC και την αναμενόμενη, 12bit ανάλυσης, ψηφιακή τιμή που προκύπτει.

4.3.2 Υπολογισμός τάσεων εξόδου και τιμών ADC

Με βάση τις παραπάνω σχέσεις μπορεί να υπολογιστεί η αναμενόμενη τάση εξόδου για κάθε ένα από τα 16 πλήκτρα. Στη συνέχεια, η τάση αυτή αντιστοιχίζεται στην αντίστοιχη θεωρητική ψηφιακή τιμή του ADC[25].

Για έναν μετατροπέα ADC 12bit ανάλυσης υπάρχουν $2^{12} = 4096$ διακριτές στάθμες. Για έυρος τάσης από 0 έως 3.3V, θεωρητική ψηφιακή τιμή ορίζεται από τη σχέση:

$$NADC \approx \frac{V_o}{3.3V}(2^{12} - 1) \tag{4.8}$$

δηλαδή:

$$NADC \approx \frac{V_o}{3.3V} \cdot 4095$$

Παράδειγμα αποτελεί η ενεργοποίηση του πλήκτρου A4, το οποίο βρίσκεται στην θέση 14 του πληκτρολογίου. Με βάση τον υπάρχοντα σχεδιασμό και τις παραπάνω σχέσεις προκύπτει ότι:

$$R_{1P} = 6110\Omega$$

Και:

$$R_{2P} = 1084.62\Omega$$

Ειπωμένος με τη σχέση (4.7) έχουμε:

$$V_o = 3.3V \frac{1084.62\Omega}{6110\Omega + 1084.62\Omega} \Leftrightarrow$$

$$V_o = 3.3V \cdot 0.150 \Leftrightarrow$$

$$V_o = 0.497V$$

Άρα η αντίστοιχη θεωρητική τιμή του ADC, βάση της εξίσωσης (4.) είναι περίπου:

$$NADC \approx \frac{0.497V}{3.3V} \cdot 4095 \Leftrightarrow NADC \approx 614$$

Ακολουθώντας την ίδια διαδικασία για όλα τα πλήκτρα του πληκτρολογίου προκύπτει ο πίνακας 4.1.

Τέλος, για τη βελτίωση της σταθερότητας της αναλογικής εξόδου χρησιμοποιήθηκαν αντιστάσεις μικρής ανοχής, καθώς και πυκνωτές παράλληλα με την αντίσταση Rz, με στόχο τη μείωση του ανεπιθύμητου θορύβου και την εξομάλυνση των μεταβολών της τάσης εξόδου.

Πίνακας 4.1: Θεωρητικές αναλογικές τιμές πλήκτρων

Πλήκτρο	R_{1P}	R_{2P}	V_o	$NADC$
Ab3	0	6519.46	3.30	4095
A3	470	6163.25	3.07	3805
Bb3	940	5801.01	2.84	3524
B3	1410	5432.59	2.62	3251
C4	1880	5057.83	2.41	2985
Db4	2350	4676.57	2.20	2725
D4	2820	4288.64	1.99	2471
Eb4	3290	3893.86	1.79	2220
E4	3760	3492.04	1.59	1972
F4	4230	3083.00	1.39	1726
Gb4	4700	2666.54	1.19	1482
G4	5170	2242.45	1.00	1239
Ab4	5640	1810.53	0.80	995
A4	6110	1370.56	0.60	750
Bb4	6580	922.31	0.41	503
B4	7050	465.53	0.20	254

4.4 Μετατροπέας Ψηφιακού Σήματος σε Αναλογικό

Μετά την παραγωγή και την επεξεργασία των ηχητικών δειγμάτων από τον ESP32, απαιτείται η μετατροπή τους σε αναλογικό ηλεκτρικό σήμα, ώστε να είναι δυνατή η μεταφορά τους στο επόμενο στάδιο της ηχητικής αλυσίδας. Παρόλο που ο μικροελεγκτής διαθέτει ενσωματωμένες δυνατότητες μετατροπής ψηφιακού σήματος σε αναλογικό, για την κατασκευή επιλέχθηκε η χρήση εξωτερικού μετατροπέα MCP4911 της microchip[13]

Ο μετατροπέας MCP4911, διαθέτει ανάλυση 10bit, παρέχοντας 1024 διακριτές ψηφιακές στάθμες, καθορίζοντας τα επίπεδα με τα οποία μπορεί να αναπαρασταθεί το πλάτος του παραγόμενου ηχητικού σήματος. Διαθέτει μονοκάναλη έξοδο και η επικοινωνία του με τον μικροελεγκτή επιτυγχάνεται μέσω του πρωτοκόλλου σειριακής επικοινωνίας SPI.

Το ολοκληρωμένο αυτό κύκλωμα υποστηρίζει εξωτερική τάση αναφοράς (Vref) με απομόνωση, διαθέτει ενσωματωμένο rail-to-rail ενισχυτή εξόδου, και παρέχει δυνατότητα επιλογής του κέρδος (Gain) στο σήμα εξόδου. Η προτεινόμενη τάση λειτουργίας του, κυμαίνεται από 2,7V έως 5,5V, γεγονός που το καθιστά κατάλληλο για διασύνδεση με μικροελεγκτές χαμηλής τάσης.

Εικόνα 4.3: Διάταξη ακροδεκτών του MCP4911[13]

Η αναλογική τάση εξόδου του DAC εξαρτάται από την εξωτερική τάση αναφοράς, τα παραγόμενα ψηφιακά δεδομένα και από την επιλογή της ενίσχυσης. Η σχέση αυτή δίνεται από:

$$V_{out} = \frac{V_{REF} \times D_n}{2^n} \times G \quad (4.8)$$

Όπου V_{REF} η εξωτερική τάση αναφοράς, D_n η είσοδος προς τον DAC σε δυαδική μορφή, G η επιλογή της ενίσχυσης και n η ανάλυση του DAC, που στην προκειμένη περίπτωση n = 10.

Στη συγκεκριμένη σχεδίαση ο MCP4911τροφοδοτείται με 5V, ενώ η τάση αναφοράς που ισούται με την τροφοδοσία καθορίζει το διαθέσιμο εύρος της αναλογικής εξόδου. Με επιλογή κέρδος G = 1, κάθε αύξηση της αποστολής κατά μία μονάδα προκαλεί ιδανικά μεταβολή:

$$\Delta V = \frac{V_{REF}}{2^n} \quad (4.9)$$

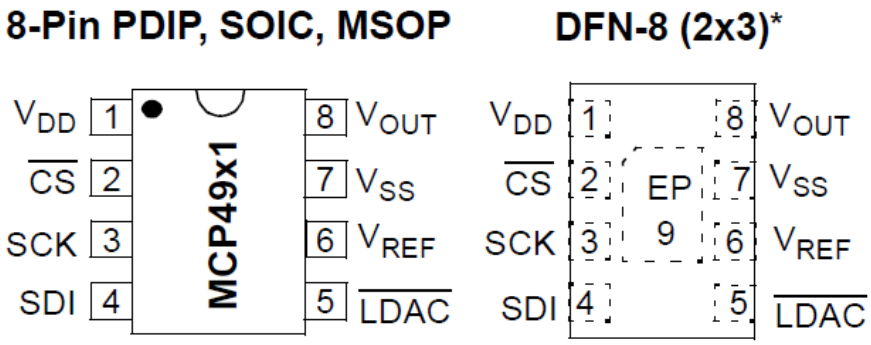
Εφόσον V_{REF} = 5V, προκύπτει ότι:

$$\Delta V = \frac{5V}{1024} \approx 4.88mV$$

Ανά ψηφιακό επίπεδο.

4.4.1 Λειτουργία του MCP4911

Η έξοδος του DAC οδηγείται από έναν ενσωματωμένο CMOS ενισχυτή ακριβείας, ο οποίος συμβάλει στη χαμηλή κατανάλωση ισχύος και στην ικανότητα οδήγησης του αναλογικού σήματος προς τα επόμενα στάδια. Ο ενισχυτής αυτός προσφέρει στην έξοδο μειωμένο θόρυβο, ενώ ταυτόχρονα, η τάση εξόδου παρουσιάζει μικρές απώλειες, με αποτέλεσμα, η τάση εξόδου, να πλησιάζει τα όρια της τάσης τροφοδοσίας.



Εικόνα 4.4: Διάταξη ακροδεκτών του Dac

Το ολοκληρωμένο κύκλωμα έχει σχεδιαστεί για να διασυνδέεται απευθείας με τη Σειριακή Περιφερειακή Διασύνδεση (SPI), η οποία υποστηρίζεται από τον μικροελεγκτή του συστήματος. Η αρχικοποίηση και τα δεδομένα τροφοδοτούνται στο ολοκληρωμένο, μέσω του ακροδέκτη SDI, ενώ ο χρονισμός των δεδομένων καθορίζεται από την ανερχόμενη άκρη του ακροδέκτη SCK. Η μεταφορά δεδομένων είναι μονής κατεύθυνσης, επομένως η συσκευή λειτουργεί μόνο ως δέκτης και δεν επιστρέφει δεδομένα προς τον μικροελεγκτή[26].

Κατά την διάρκεια της αποστολής δεδομένων, ο ακροδέκτης επιλογής CS, πρέπει να διατηρείται γειωμένος, ώστε ο DAC να δεχθεί τα δεδομένα. Η πληροφορία αποστέλλεται με τη μορφή πακέτου 16bit, στο οποίο περιλαμβάνονται 4bit ελέγχου, 10bit της ψηφιακής λέξης δεδομένων και 2bit τα οποία είναι κενά[13].

Πίνακας 4.2: Δομή πακέτου SPI προς αποστολή

0	BUF	GA	SHDN	D9	D8	D7	D6	D5	D4	D3	D2	D1	D0	X	X
bit15															bit0

Στον πίνακα 4.2, εμφανίζεται η διάταξη του πακέτου αποστολής. Η συγκεκριμένη δομή είναι απαραίτητη ώστε να καθορίζονται σωστά η λειτουργία της συσκευής, η ενίσχυση, η κατάσταση λειτουργίας και η τιμή εξόδου.

4.4.2 Υλοποίηση

Για την σωστή λειτουργία του DAC, ακολουθήθηκε μία προσεκτικά σχεδιασμένη τοπολογία τροφοδοσίας και φιλτραρίσματος. Συγκεκριμένα, στην είσοδο τροφοδοσίας (VDD) τοποθετήθηκε παράλληλα προς την γείωση, ένας πυκνωτής τανταλίου 15uF, για την απόσβεση μεταβολών χαμηλότερης συχνότητας και ένας κεραμικός πυκνωτής 100nF για την καταστολή θορύβου υψηλών συχνοτήτων[13]. Ο συνδυασμός των δύο αυτών πυκνωτών, τοποθετημένων κοντά στον ακροδέκτη τροφοδοσίας (VDD)του ολοκληρωμένου, βοηθά στην σταθεροποίηση της τάσης και στην μείωση των ανεπιθύμητων διαταραχών.

Στην έξοδο του DAC τοποθετήθηκε αντίσταση pull-down, ώστε να αποφεύγεται η αιώρηση της εξόδου σε περιπτώσεις κατά τις οποίες δεν υπάρχει ενεργό σήμα ή όταν ο μετατροπέας βρίσκεται σε κατάσταση απενεργοποίησης. Με τον τρόπο αυτό διασφαλίζεται πιο σταθερή συμπεριφορά του κυκλώματος και αποφεύγονται μεταβολές στην τάση εξόδου.

Λόγω του ότι η έξοδος ενός DAC αποτελείται από διαδοχικές στάθμες τάσης, χρειάζεται η δημιουργία ενός χαμηλοπερατού φίλτρου πρώτης τάξης RC, για την αποφυγή υψηλών συνιστωσών, πριν το σήμα οδηγηθεί στο στάδιο ενίσχυσης[6].

4.5 Ενισχυτής κυκλώματος

Στην προηγούμενη ενότητα, αναλύθηκε ο μετατροπέας ψηφιακού σε αναλογικό σήμα MCP4911. Στην παρούσα ενότητα εξετάζεται το στάδιο ενίσχυσης που ακολουθεί, δηλαδή ο ενισχυτής τάξης D που συνδέεται στην αναλογική έξοδο του DAC. Ειδικότερα, παρουσιάζεται η βασική αρχή λειτουργίας των ενισχυτών κλάσης D, τα χαρακτηριστικά του ενισχυτή PAM8302A και ο λόγος για τον οποίο επιλέχθηκε για την ενίσχυση του σήματος πριν από την οδήγηση του τελικού ηχείου.

4.5.1 Ενισχυτής κλάσης D

Οι ενισχυτές κλάσης D, γνωστοί και ως ενισχυτές μεταγωγής, αποτελούν μια κατηγορία ενισχυτών ισχύος στην οποία τα στοιχεία εξόδου λειτουργούν ως διακόπτες και όχι ως γραμμικά ενεργά στοιχεία, όπως στους ενισχυτές κλάσης A ή AB. Στην πράξη χρησιμοποιούνται συνήθως mosfets, τα οποία εναλλάσσονται γρήγορα μεταξύ κατάστασης πλήρους αγωγής και πλήρους αποκοπής. Με τον τρόπο αυτό μειώνεται σημαντικά η ισχύς που χάνεται με τη μορφή θερμότητας και επιτυγχάνεται πολύ υψηλή απόδοση σε σχέση με τους γραμμικούς ενισχυτές, η οποία υπερβαίνει το 90%.

Για να επιτευχθεί η ενίσχυση του σήματος, χρησιμοποιούνται τεχνικές διαμόρφωσης, όπως η διαμόρφωση πλάτους παλμών (PWM), ή παρόμοιες τεχνικές διαμόρφωσης. Οι τεχνικές αυτές, μπορούν να κωδικοποιούν το αναλογικό σήμα εισόδου σε μια σειρά από παλμούς υψηλής συχνότητας, των οποίων η χρονική συμπεριφορά σχετίζεται με το πλάτος του αρχικού σήματος. Το αποτέλεσμα στην έξοδο του ενισχυτή δεν είναι άμεσα ένα συνεχές αναλογικό σήμα, αλλά μία παλμική μορφή ισχύος

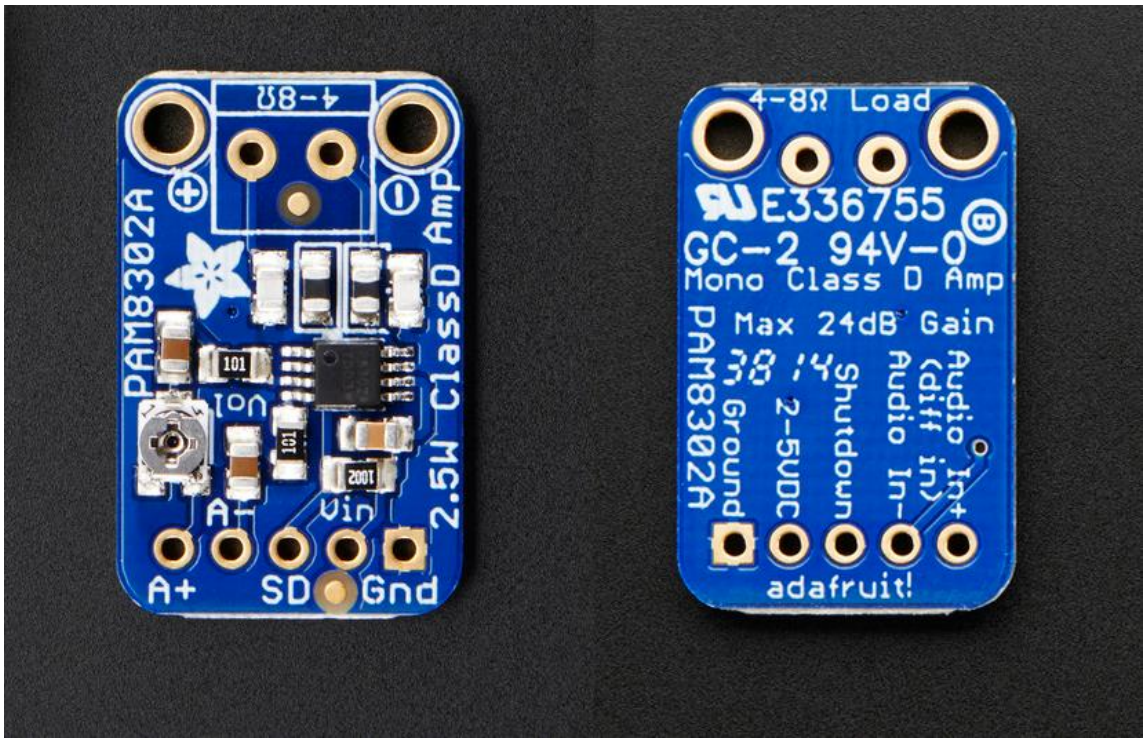
υψηλής συχνότητας, από την οποία πρέπει να ανακτηθεί η χρήσιμη συνιστώσα του ηχητικού σήματος[27].

Για τον λόγο αυτό, στο τελικό στάδιο εφαρμόζεται ένα φίλτρο χαμηλής διέλευσης, το οποίο αφαιρεί τα ανεπιθύμητα στοιχεία υψηλής συχνότητας, με σκοπό να ανακτήσει την μορφή του αρχικού αναλογικού σήματος. Παράλληλα, λόγω της λειτουργίας μεταγωγής, η απώλεια ισχύος είναι ελάχιστη, μειώνοντας την ανάγκη για εκτεταμένη ψύξη, γεγονός ιδιαίτερα χρήσιμο σε συμπαγή ηλεκτρονικά συστήματα. Τέλος, η γρήγορη μεταγωγή μπορεί να προκαλέσει ηλεκτρομαγνητικές παρεμβολές (EMI), γι΄ αυτό απαιτείται προσεκτική σχεδίαση της τροφοδοσίας, των γειώσεων και των διαδρομών του σήματος.

4.5.2 Adafruit PAM8302A

Για την υλοποίηση του σταδίου ενίσχυσης χρησιμοποιήθηκε μια έτοιμη πλακέτα ενισχυτή της εταιρίας Adafruit, βασισμένη στο ολοκληρωμένο κύκλωμα PAM8302A. Πρόκειται για μονοφωνικό ενισχυτή ήχου κλάσης D, κατάλληλο για οδήγηση ηχείων χαμηλής αντίστασης, ο οποίος μπορεί να αποδώσει ισχύ έως περίπου 2.5w, ανάλογα με την τάση τροφοδοσίας και το φορτίο εξόδου. Η πλακέτα λειτουργεί σε εύρος τάσης 2.0V έως 5.5V, γεγονός που επιτρέπει την εύκολη ενσωμάτωσή της σε συστήματα χαμηλής τάσης και φορητές εφαρμογές[28].

Ένα από τα βασικά πλεονεκτήματα του PAM8302A είναι η υψηλή ενεργειακή απόδοση, η οποία τον καθιστά κατάλληλο για μικρά ενσωματωμένα συστήματα , όπου η κατανάλωση ισχύος και η θερμική συμπεριφορά είναι κρίσιμοι παράγοντες. Επιπλέον, η πλακέτα ενσωματώνει προστατευτικά χαρακτηριστικά, όπως θερμική προστασία, ενώ διαθέτει και δυνατότητα ρύθμισης της στάθμης εξόδου μέσω ενός ενσωματωμένου ποτενσιόμετρού.



Εικόνα 4.5: Πλακέτα Adafruit PAM8302A[29]

Οι ακροδέκτες εισόδου A+ και A- του ενισχυτή, αντιστοιχούν σε διαφορική είσοδο. Στην περίπτωση της παρούσας εργασίας, όπου το σήμα που εξέρχεται από τον DAC είναι μονοτελικό ως προς την

γείωση, το σήμα οδηγείται στην είσοδο A+, ενώ η είσοδος A- συνδέεται στη γείωση του κυκλώματος, σύμφωνα με την προτεινόμενη χρήση της πλακέτας για μη διαφορικό σήμα εισόδου. Η έξοδος του ενισχυτή, είναι σχεδιασμένη ως "Bridge Tied", δηλαδή οι ακροδέκτες εξόδου συνδέονται απευθείας με το ηχείο, δίχως να χρειάζεται περεταίρω σύνδεση κάποιου άκρου με την γείωση[29]. Η διάταξη αυτή επιτρέπει καλύτερη αξιοποίηση της τάσης εξόδου και συνεπώς μεγαλύτερη ισχύ στο φορτίο.

Πίνακας 4.3 Λειτουργίες ακροδεκτών του ενισχυτή

Όνομα	Λειτουργία
VIN	Τροφοδοσία πλακέτας
GND	Κοινή γείωση τροφοδοσίας
A+ και A-	Διαφορική είσοδος του ενισχυτή
SD	Ακροδέκτης τερματισμού λειτουργίας
Output + και Output -	Έξοδος του ενισχυτή

4.5.3 Σύνδεση ενισχυτή με DAC και Ηχείο

Η χρήση μιας έτοιμης πλακέτας ενισχυτή, όπως η ADAFRUIT PAM8302A, προσφέρει σημαντικά πλεονεκτήματα σε σχέση με την κατασκευή ενός αντίστοιχου σταδίου από το μηδέν. Η επιλογή αυτή μειώνει την πολυπλοκότητα της κατασκευής, περιορίζει τις πιθανότητες σχεδιαστικού σφάλματος και επιτρέπει την ταχύτερη ενσωμάτωση ενός αξιόπιστου σταδίου ισχύος στο συνολικό σύστημα.

Οι ακροδέκτες του ενισχυτή συνδέθηκαν στο κύκλωμα του synthesizer, σύμφωνα με τα χαρακτηριστικά της Εικόνας 4.5 και του Πίνακα 4.3. Ο ακροδέκτης VIN συνδέθηκε στην τροφοδοσία των 5V, ενώ ο ακροδέκτης GND στην γείωση τροφοδοσίας του σταδίου ενίσχυσης. Η αναλογική έξοδος του DAC οδηγήθηκε στον ακροδέκτη A+ του ενισχυτή, ενώ ο ακροδέκτης A- συνδέθηκε στην γείωση, διότι το σήμα εισόδου είναι μονοτελικό και όχι διαφορικό. Ο ακροδέκτης SD χρησιμοποιήθηκε για τον έλεγχο ενεργοποίησης και απενεργοποίησης του ενισχυτή. Συγκεκριμένα, όταν δεν υπάρχει αναπαραγωγή ήχου, ο ενισχυτής τίθεται σε κατάσταση τερματισμού λειτουργίας, ώστε να μειώνεται ο ανεπιθύμητος θόρυβος στην έξοδο. Όταν ξεκινά η αναπαραγωγή, ο ενισχυτής ενεργοποιείται, επιτρέποντας την κανονική λειτουργία του σταδίου εξόδου[29].

Τέλος, οι ακροδέκτες Output+ και Output- συνδέθηκαν απευθείας στα αντίστοιχα άκρα του τελικού ηχείου, καθώς η έξοδος του ενισχυτή είναι εσωτερικά γεφυρωμένη στους ακροδέκτες της εξόδου.

4.6 Ηχείο

Για την τελική αναπαραγωγή του ηχητικού σήματος χρησιμοποιήθηκε ηλεκτροδυναμικό ηχείο ονομαστικής αντίστασης 4Ω. Η επιλογή του φορτίου είναι συμβατή με τον ενισχυτή PAM8302A που χρησιμοποιείται στο σύστημα, επιτρέποντας την απευθείας οδήγηση του ηχείου από την έξοδο του ενισχυτή. Η χρήση ενός μόνο ηχείου επαρκεί για την αναπαραγωγή του τελικού ηχητικού σήματος, καθώς η αρχιτεκτονική του synthesizer είναι μονοφωνική. Το ηχείο συνδέεται απευθείας στους ακροδέκτες εξόδου του ενισχυτή και όχι μεταξύ εξόδου και γείωσης, λόγο της λειτουργίας Bridge Tied" που προσφέρει ο ενισχυτής[26].

4.7 Υποσυστήματα διεπαφής χρήστη

Στο παρόν σύστημα, πέρα από τα βασικά υποσυστήματα παραγωγής ήχου, ενσωματώθηκαν και επιμέρους συστήματα διεπαφής χρήστη, με σκοπό την βελτίωση της λειτουργικότητας του ψηφιακού synthesizer. Τα υποσυστήματα αυτά επιτρέπουν στον χρήστη να τροποποιεί τους παραμέτρους του τελικού ηχητικού αποτελέσματος, σε πραγματικό χρόνο. Ενώ με την βοήθεια της οπτικής απεικόνισης, γνωρίζει την πραγματική τιμή των χαρακτηριστικών που επιλέγει.

4.7.1 Οθόνη ενδείξεων

Η χρήση οθόνης ενδείξεων σε ένα ενσωματωμένο μουσικό σύστημα προσφέρει άμεση πληροφόρηση σχετικά με την τρέχουσα κατάσταση λειτουργίας της συσκευής. Στην παρούσα εργασία χρησιμοποιήθηκε OLED ανάλυσης 128 x 64 pixels, βασίζεται στον ελεγκτή SSD1306, είναι δίχρωμη και επικοινωνεί με τον μικροελεγκτή μέσω του διαύλου I2C[30]. Η επιλογή της συγκεκριμένης οθόνης έγινε λόγω του μικρού μεγέθους, της χαμηλής κατανάλωσης ισχύος και της απλής σύνδεσης με τον μικροελεγτή. Η χρήση του πρωτοκόλλου I2C επιτρέπει τη σύνδεση της οθόνης με μόνον δύο γραμμές επικοινωνίας, την γραμμή δεδομένων SDA και την γραμμή χρονισμού SCL, πέραν των γραμμών τροφοδοσίας και γείωσης[19]. Η συγκεκριμένη οθόνη λειτουργεί με τάση τροφοδοσίας από 3.3V εώς 5V, διότι διαθέτει εσωτερικό ρυθμιστή τάσης.

4.7.2 Ποτενσιόμετρα ελέγχου

Στο σύστημα ενσωματώθηκαν δύο ποτενσιόμετρα, με στόχο την παροχή άμεσου ελέγχου στον χρήστη πάνω σε παραμέτρους που σχετίζονται με τη χροιά του παραγόμενου ήχου. Τα ποτενσιόμετρα αποτελούν αναλογικά στοιχεία εισόδου, μέσω των οποίων είναι δυνατή η συνεχής μεταβολή μίας παραμέτρου, ανάλογα με τη γωνιακή θέση του άξονα τους. Στην παρούσα εργασία χρησιμοποιήθηκαν ποτενσιόμετρα ίσης ονομαστικής τιμής 10kΩ, τα οποία επιτρέπουν την εισαγωγή αναλογικής πληροφορίας προς τον μικροελεγτή μέσω των εισόδων ADC.

Τα δύο ποτενσιόμετρα χρησιμοποιούνται για τον έλεγχο των παραμέτρων του χαρακτήρα ή την κατεύθυνση μεταβολής, character και της έντασης επίδρασης, amount, οι οποίες επηρεάζουν το τελικό αποτέλεσμα του παραγόμενου ήχου. Το πρώτο ποτενσιόμετρο λειτουργεί μονοκατευθυντικά, με περιοχή ελέγχου από 0% έως 100%, ακολουθώντας δεξιόστροφη φορά περιστροφής. Το δεύτερο ποτενσιόμετρο λειτουργεί συμμετρικά γύρω από μία ουδέτερη θέση, στην οποία η τιμή θεωρείται μηδενική. Συγκεκριμένα, η μεσαία θέση του αντιστοιχεί σε θεωρητική τιμή μηδέν, ενώ η περιστροφή προς τη μία κατεύθυνση αντιστοιχεί σε θετική μεταβολή, ενώ η αντίθετη περιστροφή οδηγεί σε αρνητική μεταβολή της αντίστοιχης παραμέτρου.

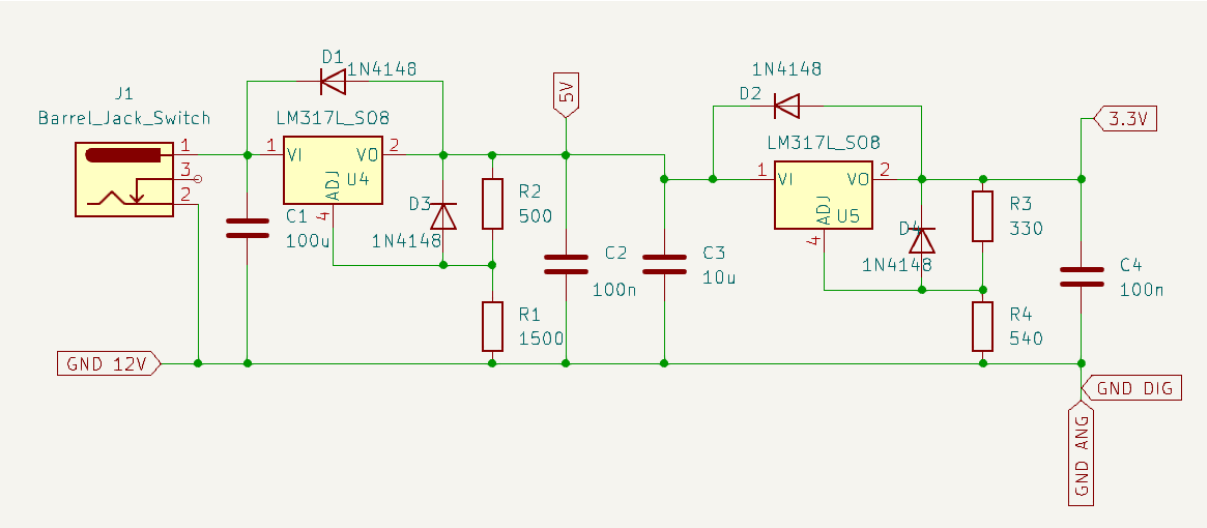
4.7.3 Διακόπτης εναλλαγής λειτουργίας

Στο σύστημα χρησιμοποιήθηκε ένας διακόπτης τύπου tactile switch, ο οποίος λειτουργεί ως μέσο ελέγχου της εναλλαγής λειτουργιών του synthesizer. Σε αντίθεση με το πληκτρολόγιο, το οποίο χρησιμοποιείται για την επιλογή της νότας, το συγκεκριμένο πλήκτρο αξιοποιείται για την πλοήγηση του χρήστη μεταξύ διαφορετικών καταστάσεων λειτουργίας του συστήματος. Με τον τρόπο αυτό παρέχεται μια απλή μέθοδος χειρισμού, χωρίς να απαιτείται μεγάλος αριθμός πρόσθετων πλήκτρων.

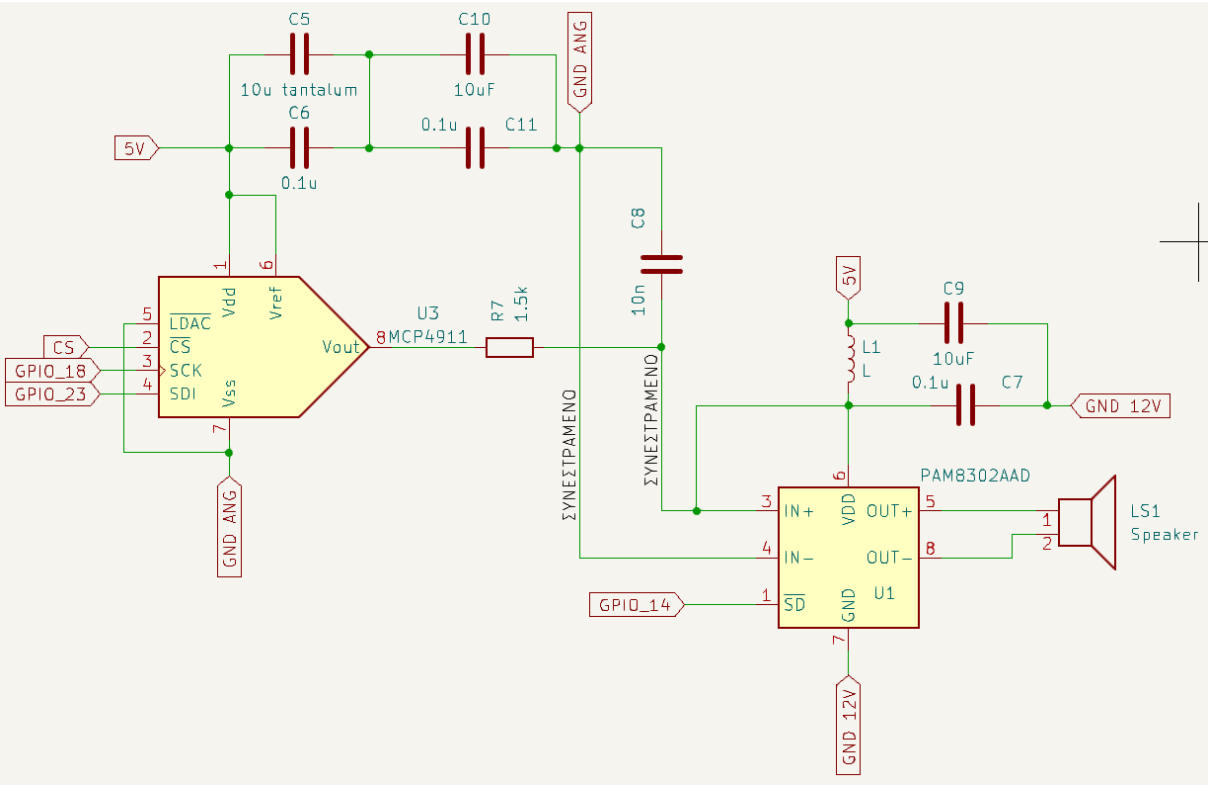
Η λειτουργία του διακόπτη είναι διπλή. Με κάθε απλό πάτημα πραγματοποιείται κυκλική εναλλαγή μεταξύ των διαθέσιμων σελίδων παραμετροποίησης και προσθήκης εφέ του ήχου. Επιπλέον, μέσω της λογικής press-and-release, όταν το πλήκτρο κρατηθεί πατημένο και έπειτα απελευθερωθεί,

πραγματοποιείται επαναφορά των ρυθμίσεων, μηδενίζοντας όλες τις επιλογές του χρήστη. Το πρόγραμμα ξεκινάει από την καθαρή μορφή αναπαραγωγής νοτών πιάνου.

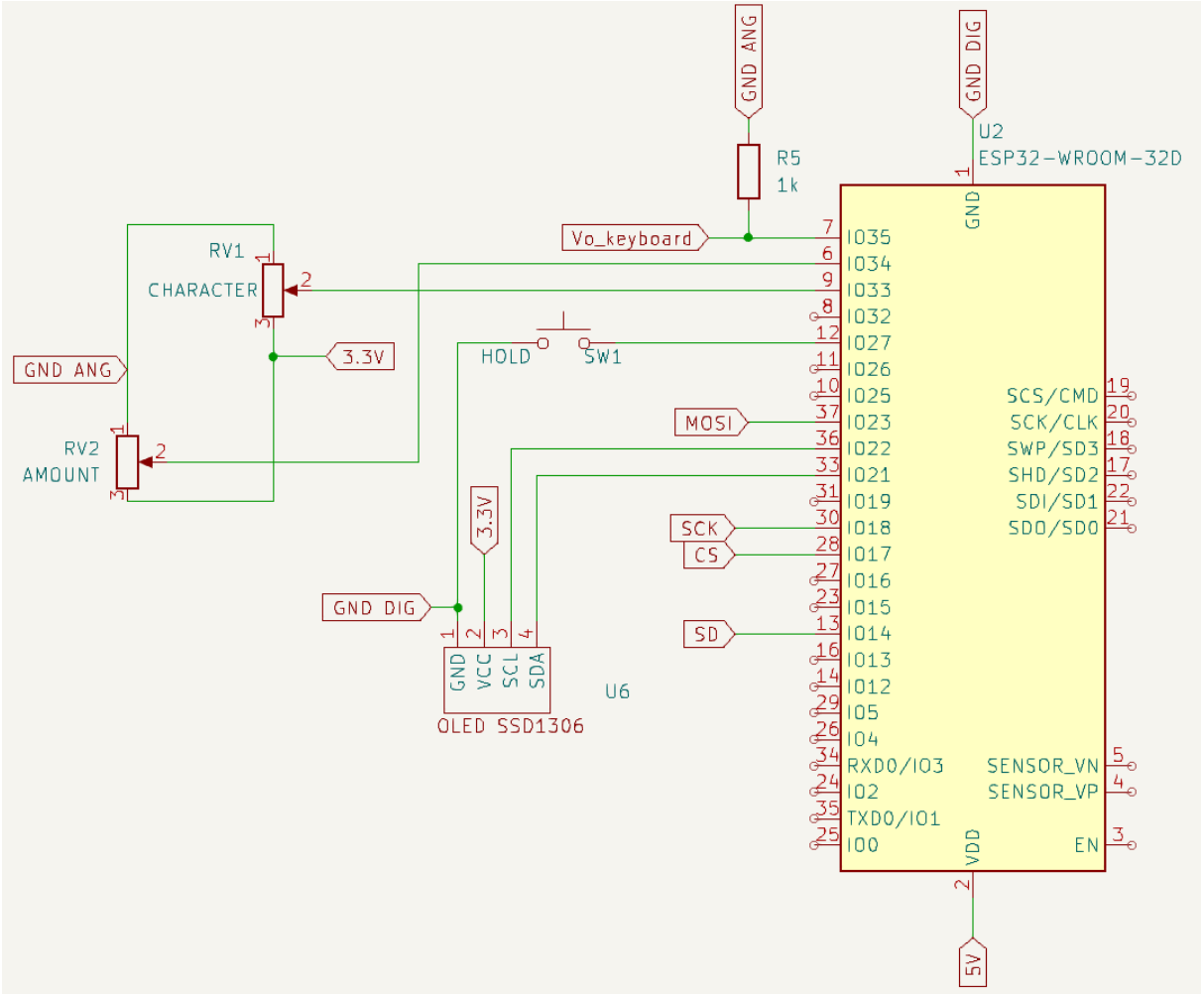
4.8 Πλήρες ηλεκτρονικό σχηματικό του ψηφιακού synthesizer



Εικόνα 4.6: Πλήρες ηλεκτρονικό σχηματικό κυκλώματος τροφοδοσίας μέσω του KiCAD



Εικόνα 4.7: Πλήρες ηλεκτρονικό σχηματικό κυκλώματος DAC και ενισχυτή μέσω του KiCAD



Εικόνα 4.8: Συνδεσμολογία περιφερειακών συστημάτων με τον ESP32 μέσω KiCAD

4.9 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε ο σχεδιασμός των επιμέρους ηλεκτρονικών υποσυστημάτων του ψηφιακού synthesizer, από το κύκλωμα τροφοδοσίας και το πληκτρολόγιο επιλογής νοτών έως τα στάδια παραγωγής και αναπαραγωγής του ηχητικού σήματος, ενώ παρουσιάστηκαν και τα στοιχεία της διεπαφής χρήστη. Για κάθε υποσύστημα εξετάστηκαν τα βασικά χαρακτηριστικά των επιλεγμένων εξαρτημάτων και πραγματοποιήθηκαν οι απαραίτητοι υπολογισμοί για την τεκμηρίωση των σχεδιαστικών επιλογών.

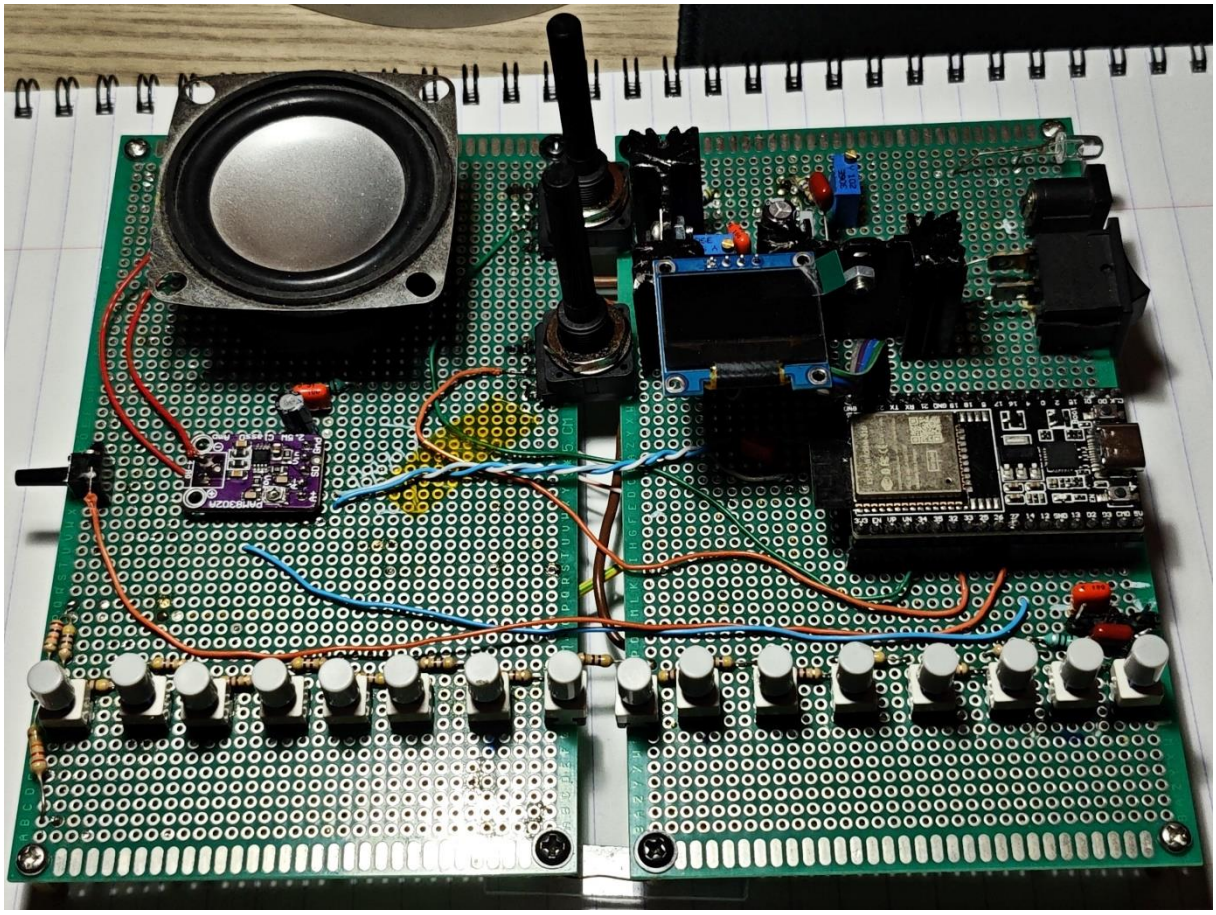
Κεφάλαιο 5ο: Κατασκευή και διασύνδεση του συστήματος

5.1 Εισαγωγή

Μετά την ολοκλήρωση του ηλεκτρονικού σχεδιασμού των επιμέρους υποσυστημάτων, ακολουθεί η πρακτική κατασκευή σε μια ενιαία λειτουργική διάταξη. Στο κεφάλαιο αυτό παρουσιάζεται η γενική αρχιτεκτονική της κατασκευής, ο τρόπος τοποθέτησης και σύνδεσης των εξαρτημάτων πάνω στις διάτρητες πλακέτες, καθώς και οι σχεδιαστικές επιλογές που πραγματοποιήθηκαν με στόχο τη λειτουργικότητα, την σταθερότητα και την εργονομία του συστήματος.

5.2 Γενική αρχιτεκτονική της κατασκευής

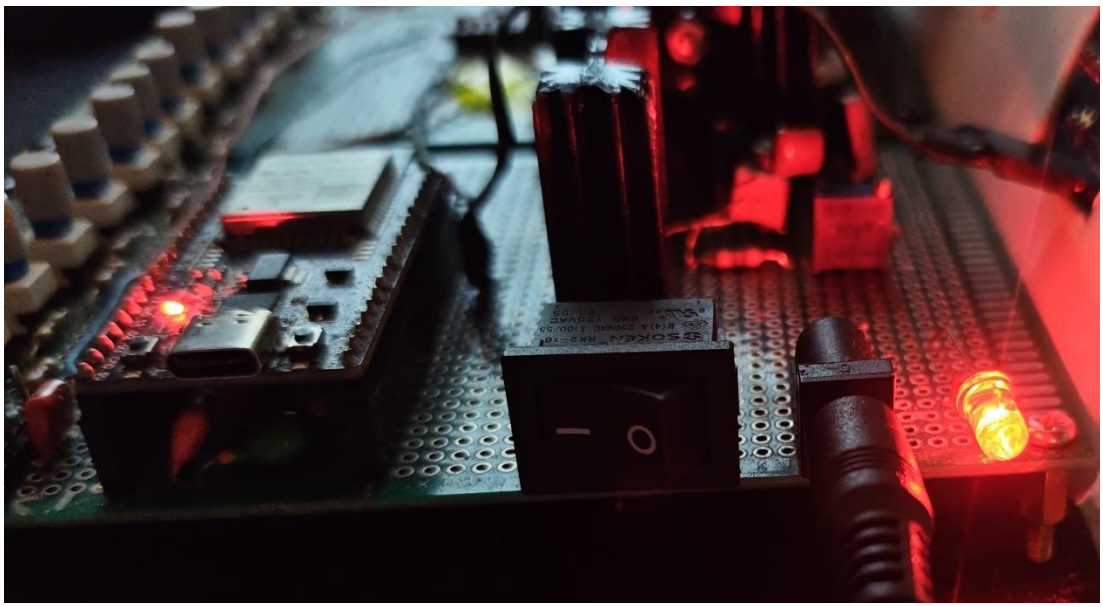
Η κατασκευή του συστήματος βασίστηκε στη χρήση δύο διάτρητων πλακετών, διαστάσεων 9cm x 15cm, πάνω στις οποίες τοποθετήθηκαν τα επιμέρους ηλεκτρονικά κυκλώματα. Οι πλακέτες τοποθετήθηκαν παράλληλα με ενδιάμεσο κενό 1cm, δημιουργώντας συνολικό πλάτος 19cm, το οποίο εξυπηρετεί την σωστή χωρομέτρηση των 16 πλήκτρων του πληκτρολογίου. Η συγκεκριμένη προσέγγιση επέτρεψε την καλύτερη μηχανική οργάνωση του συστήματος και την δημιουργία μιας κατασκευής κοντά στην μορφή ενός πραγματικού synthesizer.



Εικόνα 5.1: Γενική κάτοψη του synthesizer

Η τοποθέτηση των ηλεκτρονικών υποσυστημάτων οργανώθηκε με βάση τη μεταξύ τους λειτουργική σύνδεση και τοποθετήθηκαν σε διακριτές λειτουργικές περιοχές επάνω στην κατασκευή. Το τμήμα της τροφοδοσίας, τοποθετήθηκε στην επάνω δεξιά πλευρά της κατασκευής, διευκολύνοντας την διανομή των απαιτούμενων τάσεων προς τα υπόλοιπα κυκλώματα. Η αναπτυξιακή πλακέτα ESP32, βρίσκεται

κάτω από το υποσύστημα τροφοδοσίας, έτσι ώστε η θύρα microUSB να βρίσκεται από την δεξιά πλευρά, προσφέροντας κοινή πλευρά εισόδων εξωτερικών βυσμάτων. Ο μετατροπέας ψηφιακού σε αναλογικό σήματος τοποθετήθηκε ακριβώς κάτω από την αναπτυξιακή πλακέτα, ώστε να μειωθεί το μήκος των γραμμών επικοινωνίας.



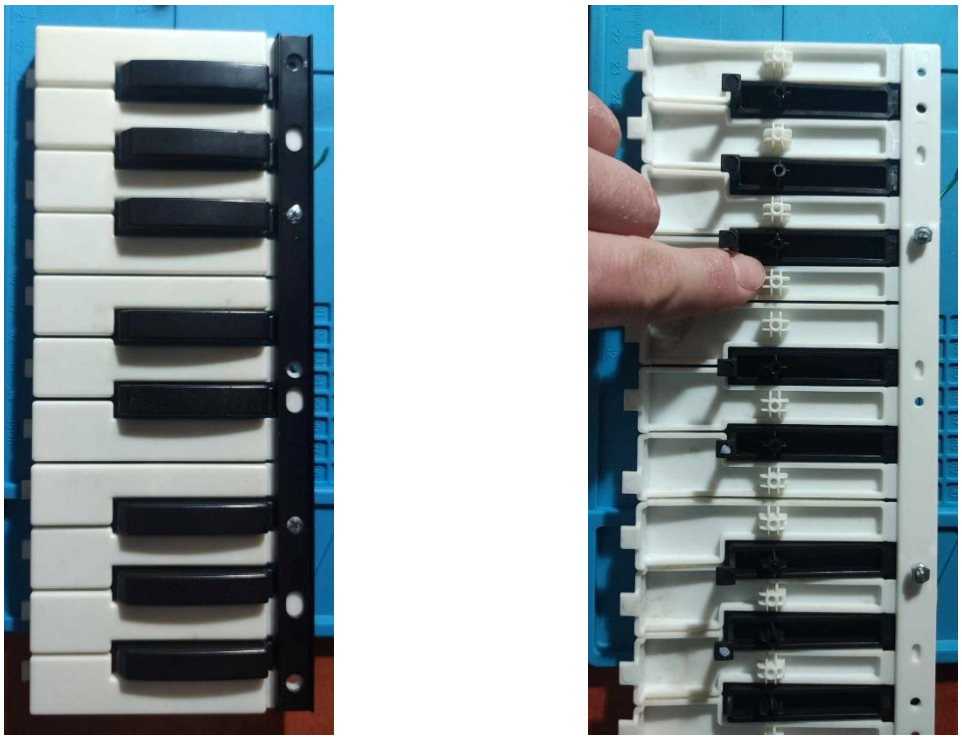
Εικόνα 5.2: δεξιά όψη του συστήματος ενώ είναι σε λειτουργία

Το στάδιο εξόδου, οργανώθηκε γύρω από τον ενισχυτή και το ηχείο, το οποίο βρίσκεται στην αριστερή πλευρά του συστήματος. Παράλληλα, τα υποσυστήματα διεπαφής χρήστη, όπως η οθόνη, τα ποτενσιόμετρα και ο διακόπτης εναλλαγής λειτουργίας, τοποθετήθηκαν σε προσβάσιμες θέσεις, ώστε να διευκολύνεται ο χειρισμός κατά τη χρήση.

Με τον τόπο αυτό, η συνολική κατασκευή δεν περιορίστηκε μόνο στη λειτουργική διασύνδεση των επιμέρους κυκλωμάτων, αλλά οργανώθηκε και με γνώμονα την εργονομία και την πρακτική αξιοποίηση της συσκευής ως ένα ολοκληρωμένο synthesaizer.

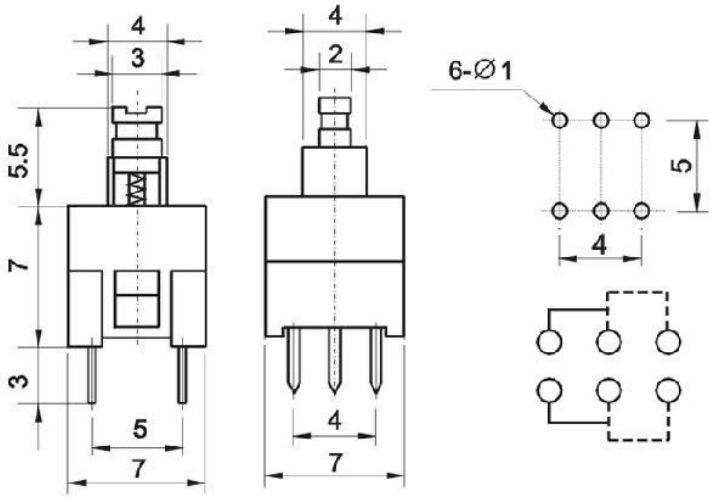
5.3 Μηχανική διάταξη της πλακέτας και ενσωμάτωση πληκτρολογίου

Η μηχανική διαμόρφωση της κατασκευής βασίστηκε στη γεωμετρία του ίδιου του πληκτρολογίου. Η επιλογή της διάταξης, 19cm x 15cm, έγινε ώστε η συσκευή να αποκτήσει πιο τετραγωνισμένο και λειτουργικό σχήμα, σε αντίθεση με μία εναλλακτική διάταξη 30cm x 9cm, η οποία θα οδηγούσε σε περισσότερο επιμήκη και λιγότερο πρακτική κατασκευή.



Εικόνα 5.3: Μπροστινή και πίσω όψη των πλήκτρων

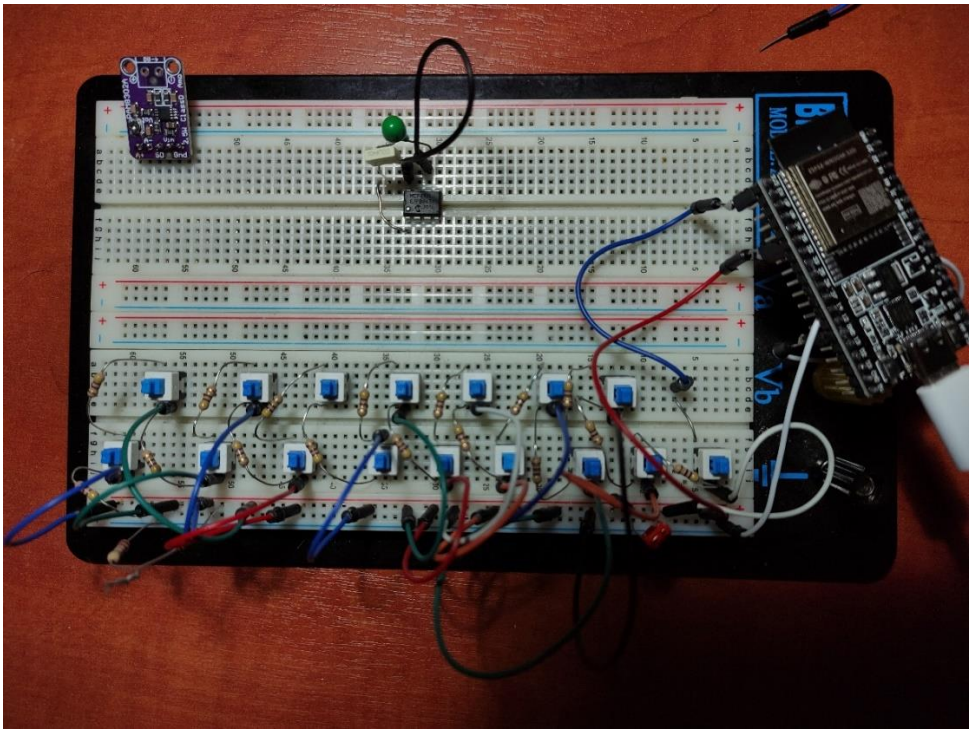
Η απόσταση μεταξύ των δύο πλακετών προέκυψε από τη δομή των πλήκτρων της εικόνας 5.2. Τα πλήκτρα που χρησιμοποιούνται στο κύκλωμα, διαθέτουν συγκεκριμένες θέσεις στήριξης και ευθυγράμμισης στην πίσω όψη τους, οι οποίες αξιοποιήθηκαν άμεσα κατά την τοποθέτηση του συστήματος. Πιο συγκεκριμένα, τα πλήκτρα κουμπώνουν στις αντίστοιχες οπές της έτοιμης κατασκευής, γεγονός που επέτρεψε την εύκολη τοποθέτηση τους. Με τον τρόπο αυτό, η σχεδίαση της βάσης δεν ήταν ανεξάρτητη από το πληκτρολόγιο, αλλά διαμορφώθηκε με βάση τις πραγματικές διαστάσεις και τα σημεία στήριξης του.



Εικόνα 5.4: Διακόπτης έξι επαφών[31]

Οι επιμέρους διακόπτες του πληκτρολογίου διαθέτουν έξι ακροδέκτες, οι οποίοι οργανώνονται σε συμμετρική διάταξη επαφών. Στην παρούσα εφαρμογή, όπου χρησιμοποιήθηκαν διακόπτες στιγμιαίας επαφής με επαναφορά, αξιοποιήθηκαν μόνο οι ακροδέκτες που ήταν απαραίτητοι για τη βασική

λειτουργία μεταγωγής, ενώ οι υπόλοιποι δεν χρησιμοποιήθηκαν. Οι ακροδέκτες 1 των πλήκτρων, συνδέονται μεταξύ τους με ίσες αντιστάσεις, δημιουργώντας μια αλυσίδα αντιστάσεων. Κάθε θέση περιλαμβάνει την αντίστοιχη αντίσταση των 470Ω, ενώ στην πίσω πλευρά των πλήκτρων, ο ακροδέκτης 2 του κάθε πλήκτρου συνδέεται στην κοινή γείωση του συστήματος.

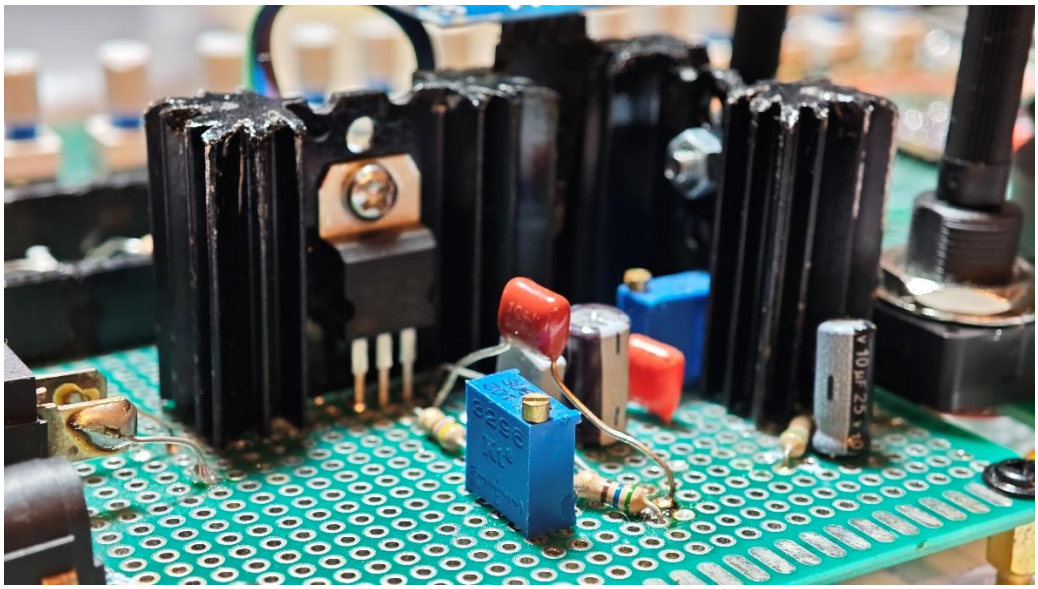


Εικόνα 5.5: Δοκιμαστική λειτουργία του πληκτρολογίου

Η έξοδος του πληκτρολογίου Vo συνδέθηκε στην αναλογική είσοδο GPIO 35 του ESP32, σύμφωνα με τη λογική του αναλογικού πληκτρολογίου που αναλύθηκε στο προηγούμενο κεφάλαιο. Επιπλέον, στην κατασκευή τοποθετήθηκαν ακροδέκτες δοκιμής, ώστε να είναι δυνατός ο έλεγχος και ο επαναπροσδιορισμός των τιμών λειτουργίας του πληκτρολογίου με τη χρήση εξωτερικών οργάνων μέτρησης, εφόσον αυτό κριθεί απαραίτητο κατά τη διαδικασία δοκιμών. Η τροφοδοσία του πληκτρολογίου πραγματοποιείται απευθείας με 3.3V, ενώ η επιστροφή της γείωσης οδηγείται στον αναλογικό κόμβο γείωσης, ώστε να περιορίζεται η επίδραση του ψηφιακού θορύβου.

5.4 Κατασκευή του κυκλώματος τροφοδοσίας

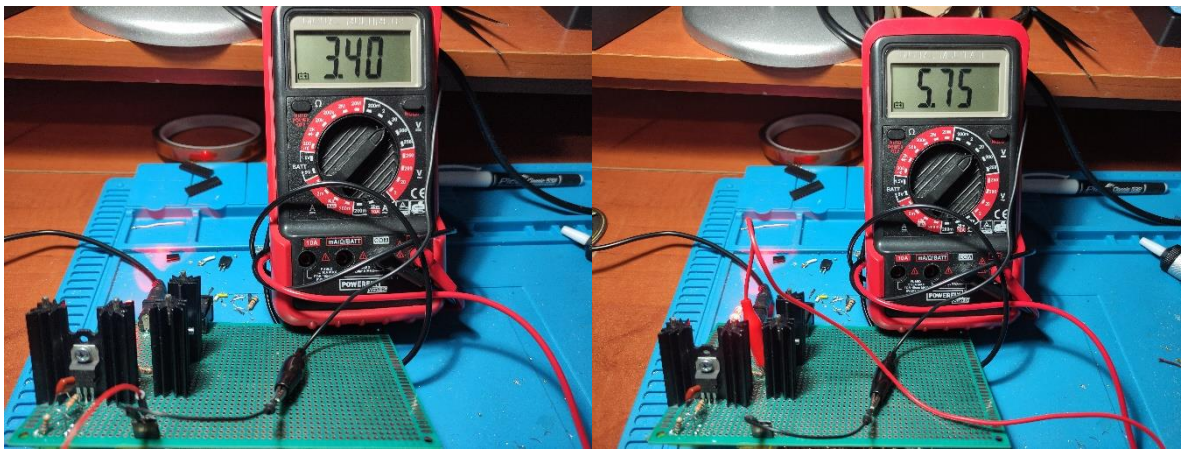
Η υλοποίηση του κυκλώματος τροφοδοσίας βασίστηκε στη διάταξη δύο διαδοχικών σταδίων LM317 που αναλύθηκε στην ενότητα 4.2. Η πρώτη βαθμίδα LM317 τροφοδοτείται από εξωτερικό τροφοδοτικό τύπου AC/DC, ονομαστικής τάσης 12V και μέγιστού ρεύματος 5A, το οποίο οδηγεί την τάση στο σύστημα μέσω του βύσματος τροφοδοσίας, συνεχούς τάσης. Η κύρια γραμμή τροφοδοσίας των 12V, διέρχεται αρχικά από τον κεντρικό διακόπτη on/off, ο οποίος επιτρέπει την πλήρη ενεργοποίηση ή απενεργοποίηση του συστήματος. Ενώ όπως διακρίνεται και στην εικόνα 5.2, τοποθετήθηκε και ενδεικτική λυχνία LED, ώστε να παρέχεται οπτική ένδειξη παρουσίας των 12V στο κύκλωμα.



Εικόνα 5.6: Κοντινή άποψη των ρυθμιστών τάσης με ψήκτρες και τα στοιχεία ρύθμισης

Το θεωρητικό υπόβαθρο της ενότητας 4.2, σε συνδυασμό με την επίλυση της εξίσωσης (4.2) για τις απαιτούμενες τάσεις τροφοδοσίας, μπορούν να τεκμηριώσουν τις τιμές των αντιστάσεων R_2 και R_4 για το κύκλωμα που εμφανίζεται στην εικόνα 4.2. Ωστόσο για την πρακτική υλοποίηση του κυκλώματος, επιλέχθηκε η χρήση ποτενσιόμετρων ακριβείας, trimmer, σε συνδυασμό με σταθερές αντιστάσεις[21]. Έτσι, είναι δυνατή η ακριβής ρύθμιση των τελικών τάσεων εξόδου. Συγκεκριμένα, για την δημιουργία των 5V χρησιμοποιήθηκε αντίσταση 470Ω στο άνω άκρο του διαιρέτη τάσης και trimmer 1kΩ σε σειρά με αντίσταση 560Ω στο κάτω άκρο. Αντίστοιχα, για την δημιουργία της τάσης των 3.3V χρησιμοποιήθηκε αντίσταση 560Ω στο κάτω άκρο και trimmer 500Ω στο άνω άκρο του αντίστοιχου κυκλώματος. Η επιλογή των trimmer έγινε ώστε να αντισταθμιστούν αποκλίσεις που οφείλονται στις ανοχές των εξαρτημάτων και στις πραγματικές συνθήκες λειτουργίας του κυκλώματος.

Η τελική ρύθμιση των τάσεων πραγματοποιήθηκε πρακτικά με χρήση πολυμέτρου, το οποίο συνδέθηκε μεταξύ της τάσης εξόδου κάθε ρυθμιστή και της γείωσης. Με τη σταδιακή περιστροφή των trimmer επιτεύχθηκε ακριβής ρύθμιση των γραμμών τροφοδοσίας στα 5V και 3.3V αντίστοιχα, εξασφαλίζοντας σταθερή και αξιόπιστη λειτουργία των επιμέρους υποσυστημάτων.

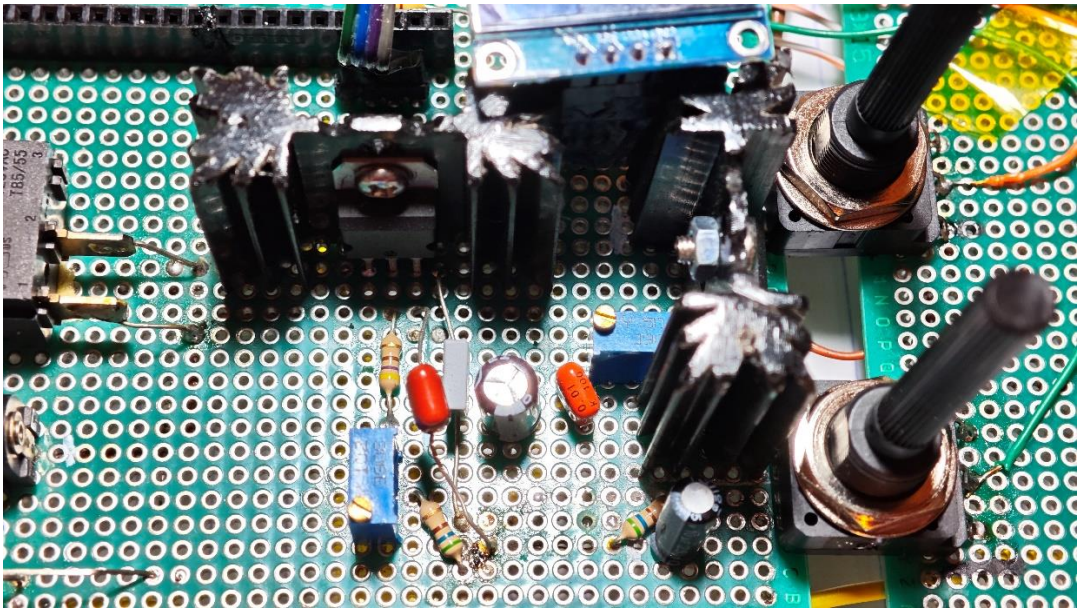


Εικόνα 5.7: Έλεγχος των τάσεων εξόδου του κυκλώματος τροφοδοσίας, 3.3V στα δεξιά και 5V στα αριστερά

Στην τελική κατασκευή ενσωματώθηκαν οι πυκνωτές 0.1μF, 1μF και 10 μF, σύμφωνα με το κύκλωμα που έχει προκύψει από τη θεωρητική σχεδίαση της ενότητας 4.2 και την προσομοίωση του. Η παρουσία

τους συμβάλει στην σταθερότητα των γραμμών τροφοδοσίας και στην ομαλή λειτουργία των ρυθμιστών τάσης. Επιπλέον, στο πληκτρολόγιο και στον ενισχυτή τοποθετήθηκαν πηνία 10μH σε σειρά με την τροφοδοσία, ώστε να περιορίζονται διαταραχές που θα μπορούσαν να επηρεάσουν την λειτουργία των αντίστοιχων υποσυστημάτων.

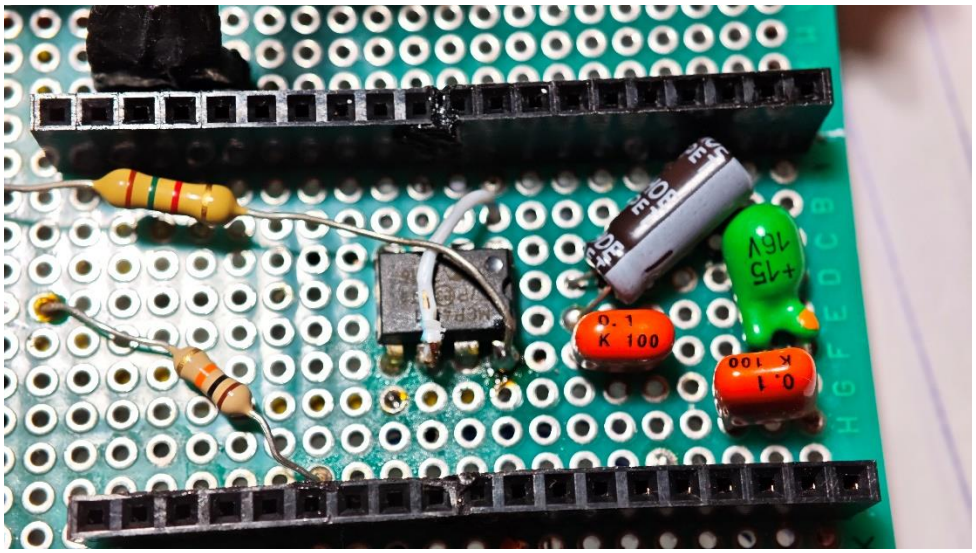
Τέλος, στους ρυθμιστές τοποθετήθηκαν ψήκτρες, ώστε να διευκολύνεται η απαγωγή θερμότητας κατά τη λειτουργία του συστήματος, ενώ οι δίοδοι προστατεύουν το κύκλωμα από επιστρεφόμενες τάσεις προς τους ρυθμιστές. Με την συγκεκριμένη αρχιτεκτονική επιτυγχάνεται σαφής διαβάθμιση της τροφοδοσίας από τα 12V προς τα 5V και στη συνέχεια προς τα 3.3V, επιτρέποντας την ασφαλή και σταθερή λειτουργία του synthesizer.



Εικόνα 5.8: Κάτοψη του κυκλώματος τροφοδοσίας 5V στα αριστερά, 3.3V στα δεξιά.

5.5 Κατασκευή του σταδίου παραγωγής ESP32 και MCP4911

Η αναπτυξιακή πλακέτα ESP32, δεν συγκολλήθηκε απευθείας πάνω στην κατασκευή, αλλά τοποθετήθηκε σε θηλυκές υποδοχές. Η επιλογή αυτή πραγματοποιήθηκε για δύο λόγους. Πρώτον, επιτρέπει την εύκολη αφαίρεση και αντικατάσταση του μικροελεγκτή σε περίπτωση βλάβης και δεύτερον, δημιουργεί ελεύθερο χώρο στο κάτω μέρος του. Ο χώρος αυτός αξιοποιήθηκε για την τοποθέτηση του μετατροπέα MCP4911 και των συνοδευτικών παθητικών στοιχείων του. Με αυτόν τον τρόπο οι γραμμές της σειριακής επικοινωνίας έχουν το δυνατόν μικρότερο μήκος[26].



Εικόνα 5.9: Φυσική θέση του μετατροπέα MCP4911

Η συνδεσμολογία της εικόνας 5.8, επιλέχθηκε συνειδητά, καθώς συμβάλει στον περιορισμό πιθανών παρεμβολών και στην βελτίωση της αξιόπιστης ψηφιακής επικοινωνίας μεταξύ μικροελεγκτή και DAC. Η σύνδεση πραγματοποιείται μέσω των γραμμών MOSI, SCK και CS, οι οποίες αντιστοιχούν στα GPIO 23 , 18 και 17 αντίστοιχα.

Ο MCP4911 τροφοδοτείται με τάση λειτουργίας 5V, ενώ και η τάση αναφοράς Vref συνδέεται απευθείας στην γραμμή τροφοδοσίας των 5V, μέσω του λευκού καλωδίου που είναι εμφανή στην εικόνα 5.8. Η επιλογή αυτή επιτρέπει στον DAC να αξιοποιεί σταθερή τάση αναφοράς και να παράγει αναλογική έξοδο στο αντίστοιχο επιτρεπτό εύρος λειτουργίας του. Η γείωση του μετατροπέα δεν συνδέθηκε στον ψηφιακό κόμβο του ESP32, αλλά στον αναλογικό κόμβο γείωσης, ώστε να περιορίζεται η επίδραση του ψηφιακού θορύβου στο ευαίσθητο αναλογικό σήμα εξόδου.

Περιμετρικά του MCP4911 τοποθετήθηκαν τα απαραίτητα στοιχεία αποσύζευξης και σταθεροποίησης της τροφοδοσίας. Συγκεκριμένα, χρησιμοποιήθηκε πυκνωτής τανταλίου 10μF παράλληλα με κεραμικό πυκνωτή 0.1μF, σύμφωνα με τις οδηγίες του κατασκευαστή[13], ενώ στην ίδια περιοχή της πλακέτας για την προστασία του μικροελεγκτή, τοποθετήθηκαν επιπλέον πυκνωτές 0.1μF και 10μF μεταξύ τροφοδοσίας και γείωσης.

5.5.1 Φίλτρο εξόδου

Η αναλογική έξοδος του μετατροπέα οδηγείται σε ένα παθητικό φίλτρο χαμηλής διέλευσης πρώτης τάξης, αποτελούμενο από αντίσταση 1.5kΩ σε σειρά και πυκνωτή 0.01μF προς την αναλογική γείωση. Το φίλτρο αυτό τοποθετήθηκε αμέσως μετά τον ακροδέκτη Vout του DAC, με σκοπό την εξομάλυνση του αναλογικού σήματος και την καταστολή ανεπιθύμητων υψηλών συχνοτήτων πριν από την είσοδό του στο στάδιο ενίσχυσης[32]. Η θεωρητική συχνότητα αποκοπής του φίλτρου δίνεται από τη σχέση:

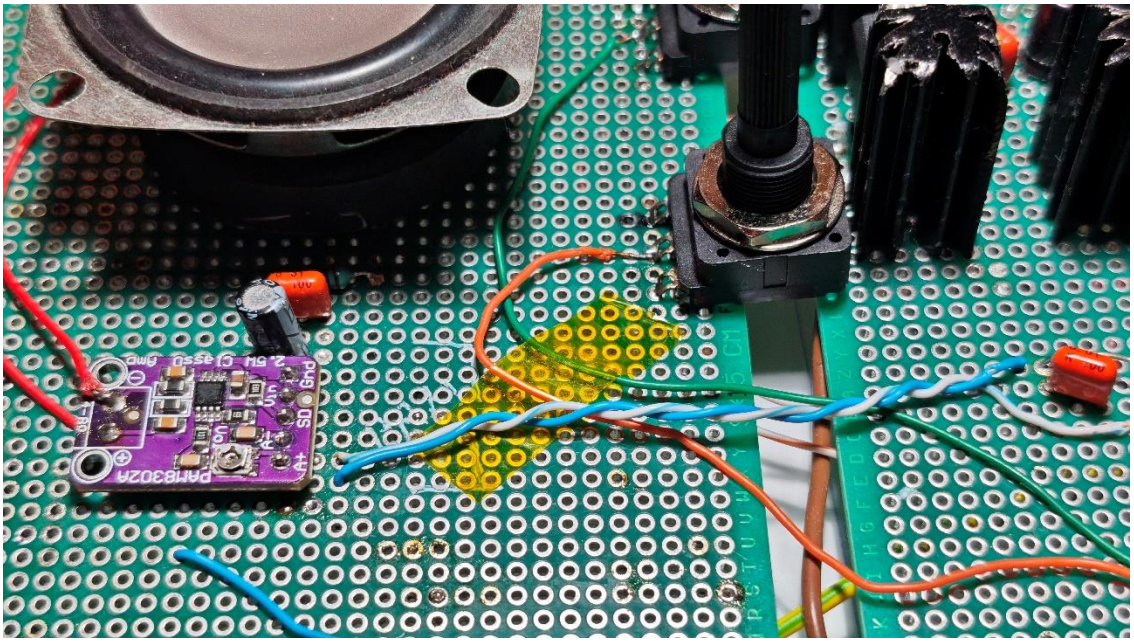
$$f_c = \frac{1}{2\pi RC} \tag{5.1}$$

και υπολογίζεται περίπου ίση με 10.6kHz. Κατά συνέπεια, το σήμα εξόδου του DAC οδηγείται προς τον ενισχυτή σε πιο ομαλή μορφή. Παράλληλα, η αιώρηση της εξόδου όταν δεν υπάρχει ενεργό σήμα, αποφεύγεται με την τοποθέτηση μιας pull-down αντίστασης.

5.6 Στάδιο ενίσχυσης και έξοδος ήχου

Μετά το στάδιο μετατροπής και το παθητικό φίλτρο εξόδου, το αναλογικό σήμα οδηγείται προς τον ενισχυτή PAM8302A, ο οποίος αποτελεί το τελικό στάδιο ενίσχυσης του συστήματος πριν από την οδήγηση του ηχείου. Η διασύνδεση του DAC με τον ενισχυτή πραγματοποιείται μέσω δύο συνεστραμμένων καλωδίων, από τα οποία το ένα μεταφέρει το αναλογικό σήμα και το δεύτερο τη γραμμή της αναλογικής γείωσης. Τα καλώδια αυτά καταλήγουν στους ακροδέκτες A+ και A- του ενισχυτή αντίστοιχα.

Η χρήση συνεστραμμένου ζεύγους καλωδίων επιλέχθηκε με στόχο την μείωση της επιφάνειας βρόχου μεταξύ της γραμμής σήματος και της γραμμής γείωσης. Με τον τρόπο αυτό περιορίζεται η μαγνητική σύζευξη με γειτονικές αγώγιμες διαδρομές και μειώνεται η πιθανότητα επαγωγής ανεπιθύμητου θορύβου στο αναλογικό σήμα. Η επιλογή αυτή κρίθηκε ιδιαίτερα σημαντική στο τμήμα μεταξύ μετατροπέα και ενισχυτή καθώς το σήμα σε αυτό το σημείο είναι χαμηλού επιπέδου και ευαίσθητο σε ηλεκτρομαγνητικές παρεμβολές[33].



Εικόνα 5.10: Τοποθέτηση του ενισχυτή PAM8302A, του ηχείου και της καλωδίωσης εισόδου του αναλογικού σήματος

Η τοποθέτηση του ενισχυτή στην αριστερή πλευρά της κατασκευής, επιλέχθηκε τόσο για ηλεκτρικούς όσο και για πρακτικούς λόγους. Από ηλεκτρικής πλευράς, ο διαχωρισμός του σταδίου ισχύος από το ψηφιακό τμήμα συμβάλλει στον περιορισμό των παρεμβολών. Από μηχανικής πλευράς, η συγκεκριμένη θέση επιτρέπει την τοποθέτηση του ηχείου ακριβώς επάνω από τον ενισχυτή, μειώνοντας το μήκος των γραμμών εξόδου και διευκολύνοντας την οργάνωση της συνολικής κατασκευής.

Η τροφοδοσία του ενισχυτή πραγματοποιείται από την γραμμή των 5V, στην οποία έχουν τοποθετηθεί επιπλέον πυκνωτές αποσύζευξης 0.1μF, 10μF καθώς και πηνίο 10μH σε σειρά, έτσι ώστε να υπάρχει σταθερότητα στην τροφοδοσία και να περιορίζονται ανεπιθύμητες διαταραχές στο στάδιο ενίσχυσης. Επίσης ο ακροδέκτης SD, ο οποίος ελέγχεται από τον ESP32, μέσω του GPIO 14, χρησιμοποιείται για την ενεργοποίηση ή απενεργοποίηση του ενισχυτή.

Το ηχείο 4Ω συνδέθηκε απευθείας στους ακροδέκτες Output+ και Output- του ενισχυτή, χωρίς την παρεμβολή πρόσθετου σταδίου μεταξύ ενισχυτή και φορτίου.

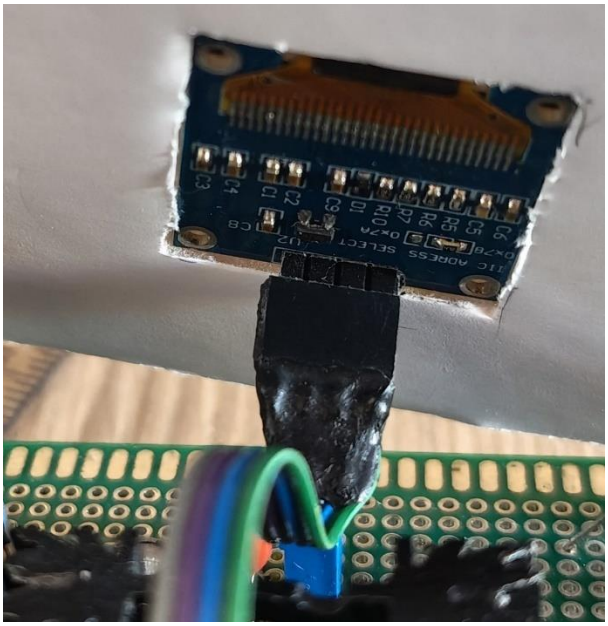
Τέλος, ιδιαίτερη σημασία δόθηκε στην γείωση του ενισχυτή. Η γείωση του ενισχυτή δεν οδηγήθηκε στον αναλογικό ή στον ψηφιακό κόμβο γείωσης, της κατασκευής, αλλά συνδέθηκε απευθείας στην γείωση της τροφοδοσίας των 12V. Η επιλογή αυτή πραγματοποιήθηκε έπειτα από πρακτική παρατήρηση παρεμβολών και θορύβου στο αναπαραγόμενο σήμα, όταν το στάδιο ισχύος μοιραζόταν κοινές διαδρομές επιστροφής με τα υπόλοιπα κυκλώματα. Με την απομόνωση αυτή μειώθηκε σημαντικά η επίδραση του ενισχυτή στα πιο ευαίσθητα τμήματα του συστήματος.

5.7 Κατασκευή και διασύνδεση διεπαφής χρήστη

Η διεπαφή χρήστη αποτελείται από την οθόνη OLED, δύο ποτενσιόμετρα και έναν διακόπτη στιγμιαίας επαφής. Τα στοιχεία τοποθετήθηκαν σε θέσεις εύκολα προσβάσιμες από τον χρήστη, συμβάλλοντας στην εργονομία του synthesizer.

5.7.1 Τοποθέτηση και σύνδεση οθόνης

Η οθόνη OLED, τοποθετήθηκε στο επάνω δεξί μέρος της κατασκευής, στο νοητό ύψος των ποτενσιόμετρων. Η θέση αυτή επιλέχθηκε ώστε ο χρήστης να μπορεί να παρακολουθεί εύκολα τις ενδείξεις της οθόνης κατά την ρύθμιση των παραμέτρων του ήχου, χωρίς να εμποδίζεται η πρόσβαση στο πληκτρολόγιο ή στα υπόλοιπα στοιχεία χειρισμού.



Εικόνα 5.11: Σύνδεση οθόνης OLED

Η σύνδεση της οθόνης με την αναπτυξιακή πλακέτα ESP32 πραγματοποιήθηκε μέσω καλωδιοταινίας τεσσάρων επαφών. Σύμφωνα με την εικόνα 5.11, η πρώτη επαφή συνδέεται στην γείωση του κυκλώματος, η δεύτερη στην τάση τροφοδοσίας των 3.3V, ενώ οι δύο επιπλέον γραμμές τροφοδοσίας συνδέθηκαν ως εξής:

$$SDA \rightarrow GPIO21$$

$$SCL \rightarrow GPIO\ 22$$

Η σχεδιαστική αυτή επιλογή έγινε για πρακτικούς λόγους, καθώς επιτρέπει την ενσωμάτωση της οθόνης στο επάνω καπάκι του τελικού κελύφους. Με αυτόν τον τρόπο, η οθόνη μπορεί να στερεωθεί στο

εξωτερικό μέρος της κατασκευής, ενώ παράλληλα παραμένει δυνατή η αφαίρεση του καπακιού χωρίς να απαιτείται αποκόλληση των καλωδίων από την πλακέτα.

5.7.2 Τοποθέτηση ποτενσιόμετρων ελέγχου

Στο κεντρικό και επάνω τμήμα της πλακέτας τοποθετήθηκαν δύο ποτενσιόμετρα των 10kΩ, τα οποία χρησιμοποιούνται για τον χειρισμό παραμέτρων που επηρεάζουν το τελικό αποτέλεσμα του παραγόμενου ήχου. Η θέση τους επιλέχθηκε ώστε να είναι άμεσα προσβάσιμα από τον χρήστη κατά την αναπαραγωγή, χωρίς να εμποδίζουν τη χρήση του πληκτρολογίου.

Τα ποτενσιόμετρα τροφοδοτούνται με 3.3V, ενώ οι γειώσεις τους επιστρέφουν στον αναλογικό κόμβο γείωσης, καθώς τα σήματα που παράγονται είναι αναλογικές τάσεις και επηρεάζουν άμεσα τις παραμέτρους επεξεργασίας του ήχου. Οι μεσαίοι ακροδέκτες τους οδηγήθηκαν στις αντίστοιχες αναλογικές εισόδους του ESP32. Συγκεκριμένα, χρησιμοποιούνται οι είσοδοι GPIO 33 και GPIO 35, μέσω των οποίων ο μικροελεγκτής διαβάζει τη θέση κάθε ποτενσιόμετρου.

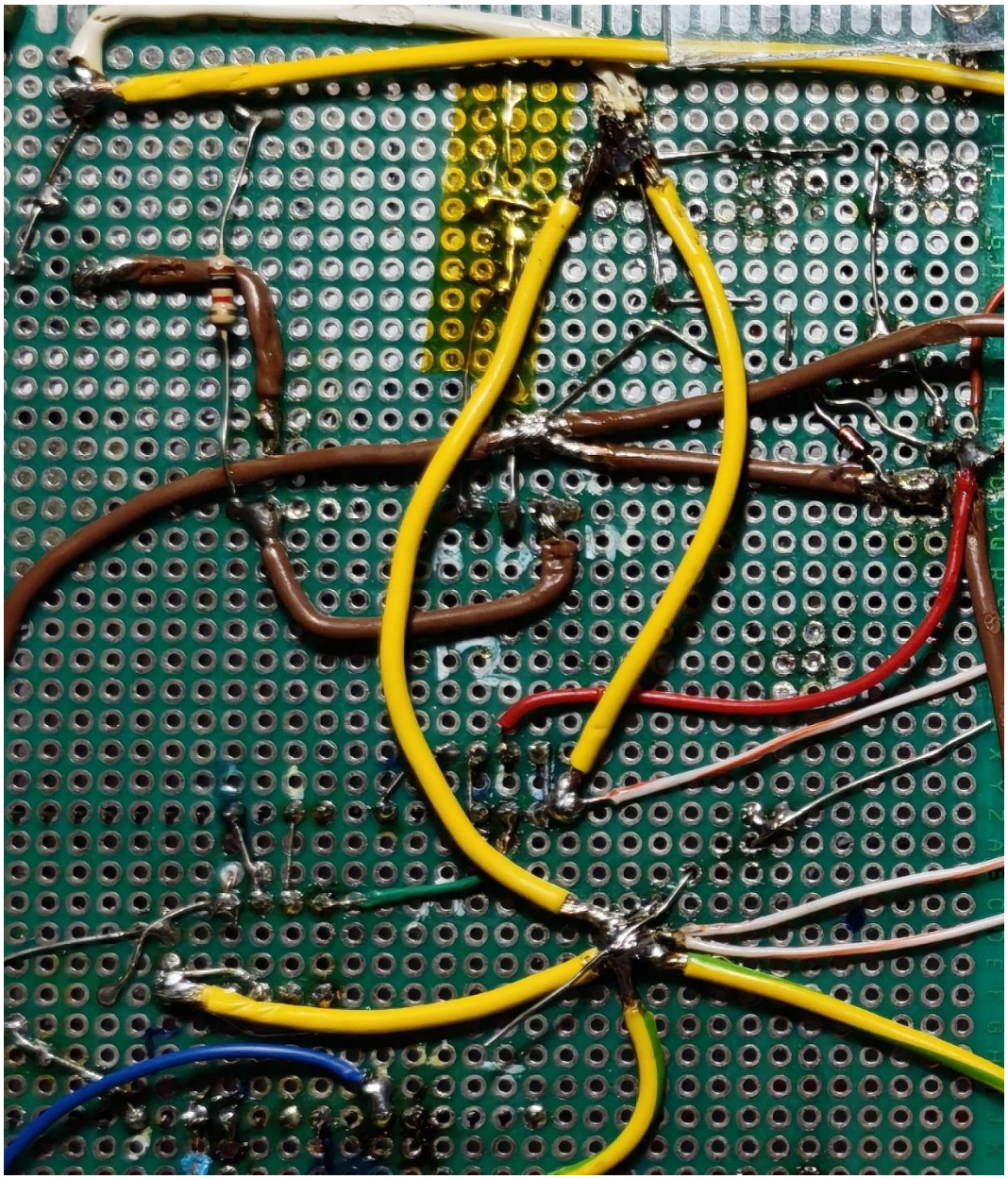
5.7.3 Διακόπτης εναλλαγής λειτουργιών

Για την εναλλαγή των βασικών λειτουργιών του synthesaizer χρησιμοποιήθηκε ένας απλός διακόπτης ο οποίος συνδέεται στον ακροδέκτη GPIO 27 του ESP32. Ο διακόπτης τοποθετήθηκε στο αριστερό άκρο της πλακέτας σε πλάγια θέση. Η επιλογή αυτή έγινε για εργονομικούς και κατασκευαστικούς λόγους. Αν ο διακόπτης τοποθετούνταν κατακόρυφα, θα απαιτούνταν μεγαλύτερο ύψος ώστε να φτάνει στο επίπεδο της τελικής επιφάνειας της κατασκευής. Με την πλάγια τοποθέτηση, ο χρήστης μπορεί να τον χειρίζεται ευκολότερα, ενώ παράλληλα πατά κάποιο πλήκτρο του πληκτρολογίου.

Για την καλύτερη μηχανική σταθερότητα του διακόπτη χρησιμοποιήθηκαν επιπλέον θηλυκά header pins, τα οποία ενισχύουν τη στήριξη του και τον καθιστούν πιο συμπαγή πάνω στην πλακέτα. Η σύνδεση του με την ψηφιακή πλακέτα ESP32, πραγματοποιείται μέσω απλής ψηφιακής εισόδου, ενώ η απαραίτητη λογική pull-up υλοποιείται εσωτερικά από το σύστημα του μικροελεγκτή. Με την σύνδεση αυτή, η είσοδος διατηρείται σε υψηλή λογική στάθμη όταν το πλήκτρο δεν είναι πατημένο. Ενώ κατά την ενεργοποίηση του οδηγείται σε χαμηλή λογική στάθμη μέσω σύνδεσης με την γείωση. Η επιλογή αυτή απλοποιεί την καλωδίωση του διακόπτη, καθώς απαιτούνται μόνο δύο αγωγοί, χωρίς τη χρήση πρόσθετων εξωτερικών στοιχείων

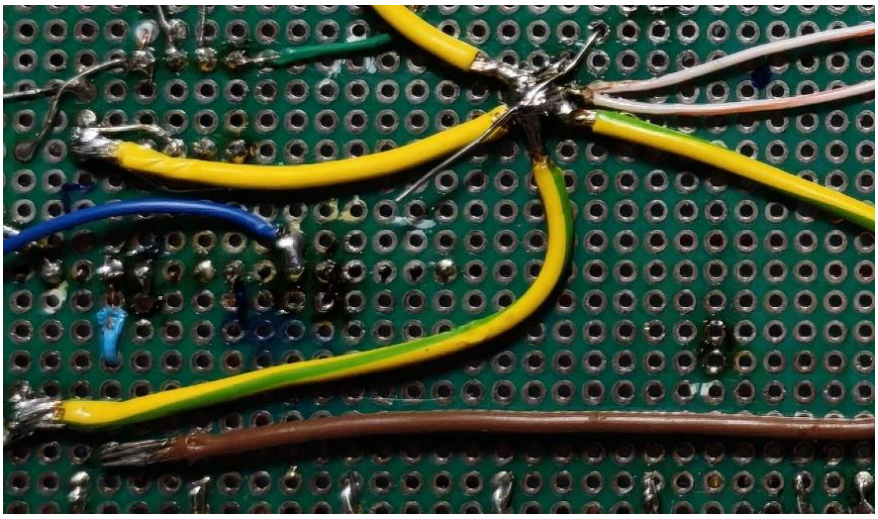
5.8 Διαχείριση γειώσεων και δρομολόγηση καλωδίων

Κατά την κατασκευή της πλακέτας δόθηκε ιδιαίτερη προσοχή στην διαχείριση των γειώσεων και στην δρομολόγηση των αγωγών τροφοδοσίας. Επειδή το σύστημα αποτελείται από μίξη ψηφιακών κυκλωμάτων, αναλογικών σημάτων χαμηλής στάθμης και κύκλωμα ενίσχυσης ήχου, η κοινή επιστροφή όλων των γειώσεων στο το ίδιο σημείο, θα μπορούσε να προκαλέσει ανεπιθύμητο θόρυβο στο τελικό ακουστικό αποτέλεσμα. Για τον λόγο αυτό, η γείωση του κυκλώματος οργανώθηκε σε τρείς βασικούς κόμβους: έναν αναλογικό, έναν ψηφιακό και έναν ξεχωριστό κόμβο επιστροφής για τον ενισχυτή.



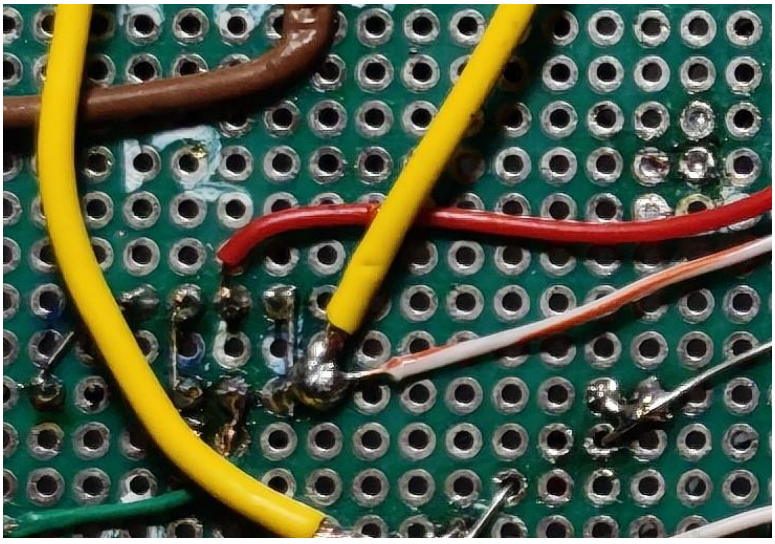
Εικόνα 5.12: Οργάνωση των γειώσεων σε τρεις βασικούς κόμβους

Ο αναλογικός κόμβος γείωσης, εικόνα 5.13, χρησιμοποιήθηκε για τα υποσυστήματα που σχετίζονται με αναλογικά σήματα, όπως ο DAC, το πληκτρολόγιο, τα ποτενσιόμετρα και η αναφορά του σήματος που οδηγείται στην είσοδο του ενισχυτή. Με αυτόν τον τρόπο, τα σήματα χαμηλής στάθμης διατηρούν κοινό σημείο αναφοράς, χωρίς να επηρεάζεται άμεσα από τις επιστροφές ρεύματος των υπόλοιπων κυκλωμάτων.



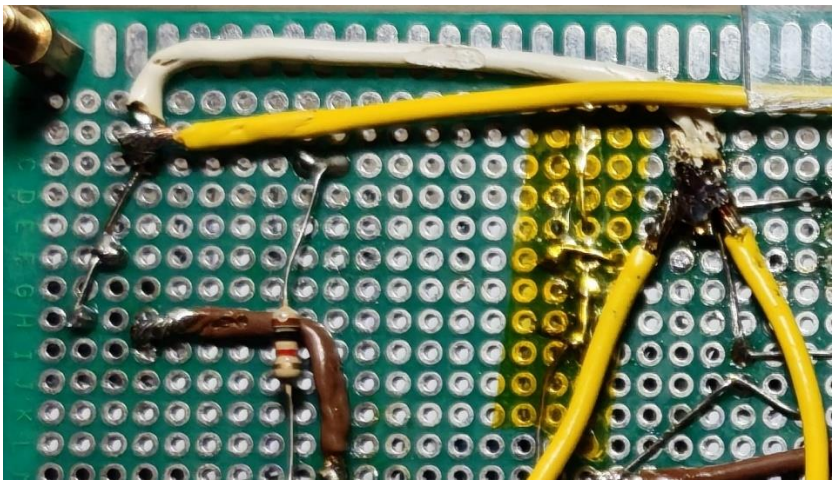
Εικόνα 5.13: Αναλογικός κόμβος γείωσης

Ο ψηφιακός κόμβος γείωσης, εικόνα 5.14, χρησιμοποιήθηκε για την αναπτυξιακή πλακέτα ESP32, την οθόνη και το πλήκτρο εναλλαγής λειτουργιών. Τα στοιχεία αυτά παράγουν ή διαχειρίζονται ψηφιακά σήματα, όπως γραμμές επικοινωνίας, παλμούς ελέγχου και μεταβολές λογικών καταστάσεων. Ο διαχωρισμός τους από τον αναλογικό κόμβο έγινε ώστε οι απότομες μεταβολές των ψηφιακών σημάτων να μην μεταφέρονται άμεσα στα σημεία αναφοράς του αναλογικού ήχου.



Εικόνα 5.14: Ψηφιακός κόμβος γείωσης

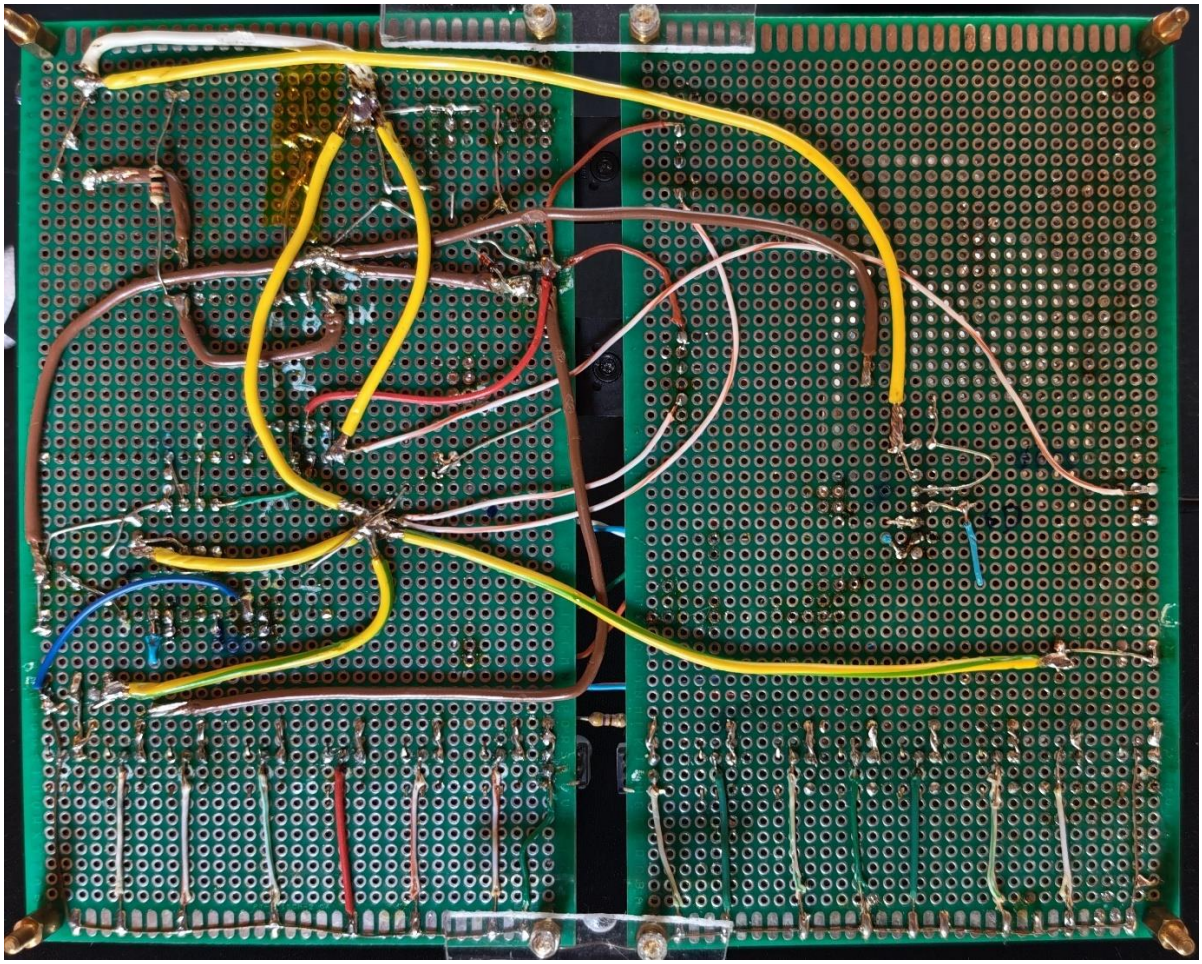
Η γείωση του ενισχυτή οδηγήθηκε με ξεχωριστό αγωγό απευθείας προς την γείωση της κύριας τροφοδοσίας των 12V, κοντά στο σημείο εισόδου του συστήματος. Η επιλογή αυτή έγινε ώστε τα ρεύματα επιστροφής του ενισχυτή να μην διέρχονται από τον ίδιο κόμβο με τον DAC και τον ESP32. Στην κάτω όψη της πλακέτας, εικόνα 5.15, φαίνεται η ξεχωριστή επιστροφή της γείωσης του ενισχυτή, η οποία έχει κίτρινο αγωγό, ενώ οι επιστροφές των αναλογικών και ψηφιακών κόμβων οδηγούνται προς την πρώτη βαθμίδα τροφοδοσίας, στην γείωση των 5V.



Εικόνα 5.15: Γείωση ενισχυτή στα αριστερά, κόμβος επιστροφών αναλογικού και ψηφιακού κόμβου στα δεξιά

Εκτός από την διαχείριση των γειώσεων, ιδιαίτερη σημασία δόθηκε στην δρομολόγηση των γραμμών τροφοδοσίας. Κάθε βασικό υποσύστημα τροφοδοτήθηκε με ξεχωριστούς αγωγούς, αντί να χρησιμοποιείται ένας κοινός αγωγός που διακλαδώνεται σε πολλά σημεία της πλακέτας. Η επιλογή αυτή μειώνει την εμφάνιση πτώσεων τάσης πάνω σε κοινές διαδρομές.

Παράλληλα, στα σημεία όπου κρίθηκε απαραίτητο προστέθηκαν τοπικά στοιχεία σταθεροποίησης, όπως πυκνωτές αποσύζευξης και πηνία σε σειρά με την τροφοδοσία των επιμέρους υποσυστημάτων.



Εικόνα 5.16: Συνολική κάτω όψη της κατασκευής και δρομολόγηση των καλωδιώσεων

Η συνολική δρομολόγηση της καλωδίωσης έγινε χειροκίνητα στην κάτω όψη των διάτρητων πλακετών. Παρόλο που η κατασκευή δεν βασίζεται σε τυπωμένο κύκλωμα, οι αγωγοί οργανώθηκαν με βάση την λειτουργία κάθε υποσυστήματος, ώστε να γίνεται η χρήση της συντομότερης διαδρομής των αγωγών, να διαχωρίζονται οι γραμμές τροφοδοσίας με καφέ αγωγούς, οι ψηφιακές γραμμές και τα αναλογικά σήματα. Η προσέγγιση αυτή συνέβαλε στην σταθερή λειτουργία του συστήματος και στην μείωση των ανεπιθύμητων παρεμβολών κατά την αναπαραγωγή ήχου.

5.9 Εξωτερικό κέλυφος και τελική μηχανική συναρμολόγηση

Στην τελική φάση της κατασκευής δημιουργήθηκε ένα εξωτερικό κέλυφος, για να τοποθετηθεί στο εσωτερικό του, το συνολικό ηλεκτρονικό σύστημα. Έτσι η κατασκευή του synthesizer θα αποκτήσει συμπαγή μορφή και θα μπορεί να χρησιμοποιηθεί ως ολοκληρωμένο πρωτότυπο. Τα πλήκτρα του πιάνου, ενσωματώθηκαν στο κάτω μέρος της κατασκευής, ενώ ακριβώς από πάνω διακρίνονται τα ποτενσιόμετρα ελέγχου και η οθόνη, έτσι ώστε ο χρήστης να έχει άμεση πρόσβαση και οπτική επαφή με την οθόνη.

Εικόνα 5.17: Επάνω όψη της τελικής κατασκευής

Παράλληλα, στις πλευρικές επιφάνειες διαμορφώθηκαν τα απαραίτητα ανοίγματα για τον διακόπτη τροφοδοσίας, το βύσμα τροφοδοσίας, την ενδεικτική λυχνία, στα δεξιά και το κουμπί ελέγχου των παραμέτρων στα αριστερά.

Εικόνα 5.18: Αριστερή όψη της τελικής κατασκευής

Η τελική τοποθέτηση του επάνω καλύμματος έγινε με βίδες, έτι ώστε να είναι δυνατή η αφαίρεση του, καθώς και το ηχείο στερεώθηκε στο κάλυμμα προσφέροντας κενό διάστημα μεταξύ της πλακέτας και του ηχείου. Το τελικό αποτέλεσμα αποτελεί ένα πλήρως συναρμολογημένο λειτουργικό πρωτότυπο, ενσωματώνοντας τα επιμέρους ηλεκτρονικά τμήματα για την δημιουργία μιας ενιαίας κατασκευής.

5.10 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε η πρακτική κατασκευή του ψηφιακού synthesizer. Η μηχανική διάταξη, η κατασκευή του πληκτρολογίου και τις τροφοδοσίας, η εγκατάσταση της ηχητικής ακολουθίας και της διεπαφής χρήστη, καθώς και η οργάνωση των καλωδιώσεων ολοκληρώθηκαν σε μια ενιαία λειτουργική κατασκευή. Με την ολοκλήρωση του υλικού μέρους του συστήματος, στο επόμενο κεφάλαιο παρουσιάζεται η ανάπτυξη του λογισμικού, από την προετοιμασία των ηχητικών δεδομένων έως την παραγωγή και επεξεργασία του ήχου σε πραγματικό χρόνο.

Κεφάλαιο 6ο: Λογισμικό και αλγοριθμική υλοποίηση

6.1 Εισαγωγή

Η λειτουργία του ψηφιακού synthesizer βασίζεται στη συνεργασία δύο βασικών σταδίων λογισμικής επεξεργασίας. Αρχικά, το MATLAB χρησιμοποιείται για την προετοιμασία των ηχητικών δειγμάτων και τη μετατροπή τους σε μορφή κατάλληλη για αποθήκευση και αναπαραγωγή από τον ESP32. Στη συνέχεια, το κύριο πρόγραμμα που αναπτύχθηκε στο Arduino IDE αναλαμβάνει την ανάγνωση των εισόδων του χρήστη, την επιλογή και επεξεργασία των ηχητικών δεδομένων, καθώς και την εφαρμογή τεχνικών μορφοποίησης του ήχου πριν από την αποστολή των τελικών τιμών στον DAC.. Στο παρόν κεφάλαιο παρουσιάζεται η συνολική λογισμική υλοποίηση, από την προετοιμασία των αρχικών ηχογραφήσεων μέχρι την παραγωγή του τελικού ψηφιακού δείγματος σε πραγματικό χρόνο.

6.2 Προετοιμασία ηχητικών δεδομένων στο MATLAB

Η βασική παραγωγή των μουσικών νοτών στηρίζεται σε προ επεξεργασμένα ηχητικά δείγματα πραγματικού πιάνου. Για να μπορούν τα δεδομένα αυτά να χρησιμοποιηθούν από τον ESP32, απαιτείται προηγουμένως η κατάλληλη επεξεργασία και μετατροπή τους. Η διαδικασία πραγματοποιείται στο MATLAB, όπου κάθε αρχικό ηχητικό αρχείο προετοιμάζεται ώστε να αποκτήσει κοινή μορφή, κατάλληλο αριθμητικό εύρος και μειωμένες απαιτήσεις αποθήκευσης.

6.2.1 Εισαγωγή και επεξεργασία ηχητικών δεδομένων

Τα αρχικά αρχεία των μουσικών νοτών εισάγονται στο MATLAB σε μορφή WAV. Η επεξεργασία πραγματοποιείται ξεχωριστά για κάθε νότα, ώστε το τελικό σύνολο δεδομένων να παρουσιάζει κοινά χαρακτηριστικά ως προς τη μορφή και τον ρυθμό δειγματοληψίας.

Αρχικά το αρχείο φορτώνεται στο MATLAB με τη συνάρτηση audioread(), η οποία επιστρέφει το ακουστικό σήμα και τη συχνότητα δειγματοληψίας του. Στην περίπτωση που το αρχείο είναι στερεοφωνικό, τα δύο κανάλια μετατρέπονται σε ένα μονοφωνικό σήμα μέσω του υπολογισμού της μέσης τιμής τους. Η διαδικασία αυτή είναι συμβατή με την τελική μορφή του συστήματος, καθώς το synthesizer διαθέτει μονοφωνική έξοδο.

```
%% 1. Φόρτωση αρχείου
filename = 'Ab3_16_32000.wav';
[y, fs] = audioread(filename);

if size(y,2) > 1
    y = mean(y,2); % stereo -> mono
end
```

Στη συνέχεια, τα δείγματα προσαρμόζονται στη συχνότητα δειγματοληψίας που χρησιμοποιεί το τελικό σύστημα. Ο στόχος είναι η συχνότητα:

$$f_s = 32kHz \tag{6.1}$$

ώστε, ο ρυθμός με τον οποίο αποθηκεύονται τα δεδομένα να συμφωνεί με τον ρυθμό αναπαραγωγής του ESP32. Αν το αρχείο έχει διαφορετική συχνότητα, γίνεται μετατροπή μέσω της συνάρτησης resample(), ώστε όλα τα δείγματα να αποκτήσουν κοινή συχνότητα[34].

```
%% 2. Έλεγχος sample rate
targetFs = 32000;

if fs ~= targetFs
```

```
    fprintf('Το αρχείο έχει fs = %d Hz. Γίνεται resample σε %d Hz.\n', fs,
targetFs);
    y = resample(y, targetFs, fs);
              fs = targetFs;
    end
```

Η επιλογή κοινού ρυθμού δειγματοληψίας είναι σημαντική, διότι επιτρέπει στο λογισμικό του ESP32 να χειρίζεται όλους τους πίνακες δεδομένων με ενιαίο τρόπο.

Μετά την φόρτωση και την προσαρμογή της συχνότητας δειγματοληψίας, αφαιρείται η μέση τιμή του σήματος. Το βήμα αυτό περιορίζει πιθανό DC offset, ώστε το ακουστικό σήμα να είναι κεντραρισμένο γύρω από το μηδέν πριν μετατραπεί σε αριθμητικές τιμές.

```
    %% 3. Αφαίρεση DC offset
    y = y - mean(y);

    %% 4. Αποθήκευση 1/4 του σήματος
    compressionFactor = 4;
    fsStored = fs / compressionFactor;

    % anti-alias φίλτρο
    y_stored = resample(y, 1, compressionFactor);
```

Για τη μείωση του απαιτούμενου αποθηκευτικού χώρου, χρησιμοποιείται η διαδικασία της υπο δειγματοληψίας, αποθηκεύοντας το ¼ των συνολικών δειγμάτων. Η μείωση αυτή πραγματοποιείται με τη συνάρτηση resample(), ώστε να εφαρμοστεί κατάλληλο anti-aliasasing φίλτρο πριν από την επεξεργασία[34]. Έτσι, το αποθηκευμένο σήμα αντιστοιχεί σε συχνότητα:

$$f_s\text{Stored} = \frac{f_s}{4} = 8000\text{Hz} \tag{6.2}$$

Η επιλογή αυτή έγινε με σκοπό την μείωση του απαιτούμενου αποθηκευτικού χώρου. Η αποθήκευση λιγότερων σημείων μειώνει σημαντικά τις απαιτήσεις μνήμης, ενώ η διαθέσιμη ισχύς του ESP32 επιτρέπει την ανακατασκευή των ενδιάμεσων δειγμάτων κατά την αναπαραγωγή.

6.2.2 Κανονικοποίηση και μετατροπή σε 10-bit δείγματα

Στην συνέχεια , το σήμα πρέπει να μετατραπεί από μορφή ακουστικού δείγματος σε αριθμητικές τιμές συμβατές με τον DAC MCP4911. Το αρχικό ηχητικό αρχείο περιλαμβάνει θετικές και αρνητικές τιμές γύρο από το μηδέν, ενώ ο DAC έχει ανάλυση 10bit[13]. Επομένως, οι τελικές τιμές που αποθηκεύονται στον πίνακα πρέπει να βρίσκονται στο διάστημα από 0 έως 1023.

Για τον λόγο αυτό, απαιτείται κατάλληλη μετατροπή του εύρους των δεδομένων πριν από την αποθήκευση τους.

```
    %% 5. Κανονικοποίηση γύρω από το 512
    amplitude = max(abs(y_stored));

    if amplitude == 0
        error('Το σήμα είναι μηδενικό ή δεν φορτώθηκε σωστά.');
    end
    normalized = y_stored / amplitude;   % περιοχή περίπου [-1, 1]
```

Αρχικά το σήμα κανονικοποιείται ως προς τη μέγιστη απόλυτη τιμή του. Με αυτόν τον τρόπο, το πλάτος του μετατρέπεται στο διάστημα περίπου:

$$-1 \leq x_N[n] \leq 1 \tag{6.3}$$

ανεξάρτητα από την αρχική στάθμη του ηχητικού αρχείου. Η διαδικασία αυτή επιτρέπει σε όλες τις νότες να αποκτήσουν κοινή κλίμακα πριν μετατραπούν σε αριθμητικές τιμές.

Στην συνέχεια, το κανονικοποιημένο σήμα μετατοπίζεται γύρο από την μέση τιμή της 10 bit κλίμακας, η οποία είναι η τιμή 512. Για την κλιμάκωση χρησιμοποιείται η μεταβλητή dacHalfRange με τιμή 460, ώστε το σήμα να μεταβάλλεται περίπου στο εύρος 512 ± 460.

```
% 10-bit DAC με κέντρο 512.
dacCenter = 512;
dacHalfRange = 460;

scaled = round(dacCenter + normalized * dacHalfRange);
```

Το νέο θεωρητικό εύρος του σήματος γίνεται περίπου 52 έως 972. Αυτό το εύρος δεν αξιοποιεί πλήρως τα όρια της κλίμακας, αφήνοντας κατάλληλο περιθώριο ασφαλείας για την αποφυγή φαινομένου clipping κατά την αναπαραγωγή.

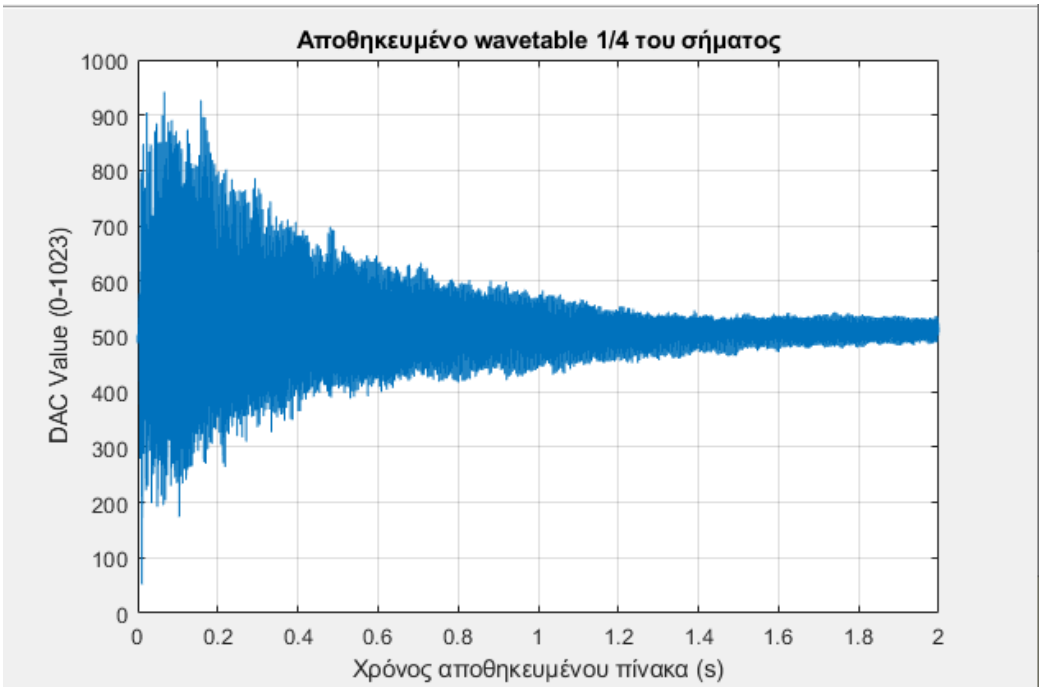
Τέλος, εφαρμόζεται περιορισμός στα επιτρεπτά όρια της ανάλυσης 10 bit. Το βήμα αυτό λειτουργεί ως δικλείδα ασφαλείας, ώστε καμία αποθηκευμένη τιμή να μην ξεπερνά το επιτρεπόμενο διάστημα 0 έως 1023.

6.2.3 Αποθήκευση δεδομένων

Μετά την κανονικοποίηση και τον περιορισμό, οι τιμές αποθηκεύονται με μορφή πίνακα γλώσσας C σε ένα αρχείο txt. Η επιλογή της μορφής αυτής επιτρέπει την άμεση μεταφορά του πίνακα στο αρχείο κεφαλίδας , το οποίο χρησιμοποιείται από το κύριο πρόγραμμα του ESP32. Κάθε δείγμα αποθηκεύεται σε δεκαεξαδική τιμή, ώστε ο πίνακας να είναι πιο συμπαγής και ευανάγνωστος κατά την εισαγωγή του στον κώδικα.

```
%% 7. Αποθήκευση σε C πίνακα
arrayName = 'TSignalAb3';
fileID = fopen('TSignal_32k_x4.txt', 'w');

fprintf(fileID, 'const uint16_t %s[%d] = {\n', arrayName, length(scaled));

for i = 1:length(scaled)
    fprintf(fileID, '0x%03X', scaled(i));

    if i < length(scaled)
        fprintf(fileID, ', ');
        if mod(i, 8) == 0
            fprintf(fileID, '\n');
        end
    end
end

fprintf(fileID, '\n};\n');
fclose(fileID);
```

Τέλος, για τον έλεγχο του αποτελέσματος, το MATLAB δημιουργεί επίσης γραφική αναπαράσταση των αποθηκευμένων δεδομένων. Με αυτόν τον τρόπο μπορεί να ελεγχθεί οπτικά αν το σήμα έχει μετατοπιστεί σωστά γύρω από την τιμή 512 και αν οι τιμές του παραμένουν εντός της επιτρεπτής περιοχής λειτουργίας του DAC.



Εικόνα 6.1: Γραφική αναπαράσταση του συμπιεσμένου σήματος

```
Command Window
  Ο πίνακας δημιουργήθηκε με κέντρο 512.
  Αρχικό fs: 32000 Hz
  Stored fs: 8000 Hz
  Compression factor: x4
  Stored samples: 15996
  Min DAC: 52
  Max DAC: 942
fx >>
```

Εικόνα 6.2: Βοηθητικός έλεγχος του αποθηκευμένου σήματος

6.3 Οργάνωση του αρχείου κεφαλίδας

Το αρχείο κεφαλίδας, λειτουργεί ως αποθηκευτικός χώρος των ηχητικών δεδομένων και των βασικών παραμέτρων που απαιτούνται από την επιλογή και αναπαραγωγή κάθε νότας. Με αυτόν τον τρόπο, το κύριο πρόγραμμα παραμένει πιο οργανωμένο, καθώς οι μεγάλοι πίνακες δειγμάτων δεν βρίσκονται απευθείας στο βασικό κορμό του κώδικα, αλλά στο ξεχωριστό αρχείο piano32.h.

Το κύριο μέρος του αρχείου αποτελείται από τους πίνακες δεδομένων των νοτών. Κάθε νότα αποθηκεύεται σε ξεχωριστό πίνακα τύπου const uint16_t, ο οποίο περιέχει τις τιμές που έχουν εξαχθεί από το MATLAB. Παρόλο που ο μετατροπέας ψηφιακού σε αναλογικό σήμα έχει ανάλυση 10-bit, χρησιμοποιείται τύπος uint16_t, επειδή η αποστολή των δεδομένων προς αυτόν, γίνεται μέσω 16-bit λέξης. Ενώ ο χαρακτηρισμός const δηλώνει ότι τα δεδομένα του πίνακα δεν μπορούν να τροποποιηθούν κατά την εκτέλεση του προγράμματος. Η δομή του πίνακα δεδομένων μιας νότας εμφανίζεται στο επόμενο απόσπασμα:

```
const uint16_t TSignalB4[15996] = {
0x200, 0x201, 0x200, 0x200, 0x201, 0x200, 0x200, 0x201,
0x200, 0x200, 0x202, 0x203, 0x202, 0x203, 0x1FB, 0x208,
0x202, 0x200, 0x203, 0x20F, 0x1F7, 0x206, 0x216, 0x213,
0x20C, 0x210, 0x1E9, 0x1EB, 0x207, 0x1FB, 0x1D5, 0x204,
0x1F6, 0x1D3, 0x1F9, 0x266, 0x20D, 0x1D9, 0x1FE, 0x224,
0x1C3, 0x1FA, 0x21A, 0x1AA, 0x198, 0x1EB, 0x1FB, 0x1EB,
0x21E, 0x1D3, 0x197, 0x24B, 0x257, 0x20D, 0x1ED, 0x24B,
0x1EE, 0x230, 0x294, 0x223, 0x1AC, 0x1DB, 0x258, 0x1E6,
0x1FA, 0x1F2, 0x15F, 0x1F3, 0x286, 0x245, 0x207, 0x234,
0x23B, 0x1B9, 0x24A, 0x1B4, 0x12E, 0x157, 0x205, 0x19E,
0x1E6, 0x1A0, 0x13E, 0x20C, 0x281, 0x264, 0x1E4, 0x1DA,
0x221, 0x1D2, 0x2C2, 0x1FC, 0x111, 0x169, 0x206, 0x1B4....}
```

Επιπλέον, στο αρχείο υπάρχουν πρόσθετοι πίνακες οι οποίοι επιτρέπουν στο κύριο πρόγραμμα να διαχειρίζεται τις νότες με κοινό τρόπο.

- **noteSignals [ ]:** περιέχει δείκτες προς τους πίνακες των 16 μουσικών νοτών.
- **noteNames [ ]:** αποθηκεύει τα ονόματα των αντίστοιχων νοτών.
- **Frequencies [ ]:** περιλαμβάνει την θεμελιώδη συχνότητα που αντιστοιχεί στις διαθέσιμες νότες. Χρησιμοποιείται από το κύριο πρόγραμμα και από τις συναρτήσεις μορφοποίησης χροιάς
- **NADCp [ ] και NADCn [ ]:** ορίζουν τα άνω και κάτω όρια των τιμών ADC για κάθε πλήκτρο. Χρησιμοποιούνται για την αναγνώριση των πλήκτρων του αναλογικού πληκτρολογίου.
- **wavetableSizes:** δηλώνεται το μέγεθος των πινάκων.

Η χρήση του πίνακα δεικτών noteSignals[ ] επιτρέπει στο κύριο πρόγραμμα να επιλέγει τη μουσική νότα μέσω ενός κοινού δείκτη θέσης αντί να απαιτείται διαφορετικός κώδικας για κάθε πίνακα. Η ίδια θέση μπορεί να χρησιμοποιηθεί για την ανάκτηση του ονόματος, της συχνότητας και των υπόλοιπων παραμέτρων της αντίστοιχης νότας.

```
const uint16_t* noteSignals[16] = {
TSignalAb3,
TSignalA3,
TSignalBb3,
TSignalB3,
TSignalC4,
TSignalDb4,
TSignalD4,
TSignalEb4,
TSignalE4,
TSignalF4,
TSignalGb4,
TSignalG4,
TSignalAb4,
TSignalA4,
TSignalBb4,
TSignalB4};
```

```
const int NADCcenter[16] = {
  4014, 3313, 2810, 2603,
  2304, 2081, 1808, 1637,
  1200, 944, 1045, 853,
   663, 150, 250, 81
};
```

Παρόλο που κατά τον σχεδιασμό του συστήματος υπολογίστηκαν οι θεωρητικές τιμές του ADC για κάθε πλήκτρο, στην τελική υλοποίηση χρησιμοποιήθηκαν πρακτικά βαθμονομημένες τιμές. Για τον Έτσι έγινε καταγραφή των πραγματικών τιμών που εμφανίζονταν στην είσοδο ADC του μικροελεγκτή κατά την ενεργοποίηση κάθε πλήκτρου. Αυτές οι τιμές χρησιμοποιήθηκαν για την δημιουργία ενός πίνακα NADCcenter[16],

όπου κάθε στοιχείο αντιστοιχεί στην κεντρική τιμή ADC ενός συγκεκριμένου πλήκτρου και χρησιμοποιείται ως σημείο αναφοράς. Η πρακτική βαθμονόμηση κρίθηκε απαραίτητη, καθώς οι πραγματικές μετρήσεις επηρεάζονται από τις ανοχές των αντιστάσεων, τις αποκλίσεις της αναλογικής εισόδου και τον θόρυβο που προκαλούν στοιχεία του κυκλώματος.
Συνοψίζοντας, η οργάνωση αυτή κάνει το αρχείο piano32.h κεντρικό σημείο αποθήκευσης των δεδομένων αναπαραγωγής. Το κύριο πρόγραμμα του ESP32 μπορεί να επιλέγει μία νότα με βάση τη θέση της στον πίνακα, να εντοπίζει το όνομά της, τη συχνότητα της και τον αντίστοιχο πίνακα δεδομένων.

6.4 Δομή και αρχικοποίηση του κύριου προγράμματος

Το κύριο πρόγραμμα του συστήματος αναπτύχθηκε στο περιβάλλον Arduino IDE και εκτελείται από την αναπτυξιακή πλακέτα ESP32. Το αρχείο του προγράμματος περιλαμβάνει τη βασική ροή λειτουργίας του synthesizer, όπως την αρχικοποίηση των περιφερειακών, την ανάγνωση των διαθέσιμων πλήκτρων, τη διαχείριση και επεξεργασία των ηχητικών δειγμάτων, καθώς και την αποστολή τους στον MCP4911.

Στην αρχή του κύριου προγράμματος εισάγονται οι απαραίτητες βιβλιοθήκες, σταθερές, μεταβλητές κατάστασης και βοηθητικές συναρτήσεις, συμβάλλοντας στην καλύτερη οργάνωση του προγράμματος.

Βασικές βιβλιοθήκες:

- **#include "SPI.h":** χρησιμοποιείται για την επικοινωνία του ESP32 με τον μετατροπέα MCP4911.
- **#include "Wire.h":** χρησιμοποιείται για τη λειτουργία του διαύλου I2C.
- **#include "Adafruit_SSD1306.h" και "Adafruit_GFX.h":** για τον έλεγχο και την δημιουργία γραφικών στην OLED.
- **#include "math.h":** παρέχει μαθηματικές συναρτήσεις όπως sinf(), fabs() και άλλες που χρησιμοποιούνται κατά την ψηφιακή επεξεργασία του ήχου.
- **#include "piano32.h":** το αρχείο κεφαλίδας που περιλαμβάνει πίνακες ηχητικών δεδομένων και παραμέτρους.

Επιπλέον υπάρχουν βιβλιοθήκες και αρχεία για τη διαχείριση των hardware timers και των GPIOs.

Βασικές σταθερές και παράμετροι:

- **#define SAMPLE_RATE:** καθορίζει τον ρυθμό παραγωγής των ηχητικών δειγμάτων στα 32kHz.
- **#define AUDIO_BUFFER_SIZE:** χρησιμοποιείται ως ενδιάμεση μνήμη δειγμάτων ανάμεσα στη διαδικασία παραγωγής ήχου και στη διαδικασία αποστολής των τιμών στον DAC και έχει μέγεθος 256 δείγματα.
- **#define INVERT_POT_CHARACTER και INVERT_POT_AMOUNT:** χρησιμοποιούνται για την αντιστροφή της φοράς λειτουργίας των δύο ποτενσιόμετρων.

Σαν σταθερές ορίζονται οι βασικοί ακροδέκτες που χρησιμοποιούνται από το σύστημα, ώστε να μπορούν να χρησιμοποιηθούν σε όλο το πρόγραμμα χωρίς επανάληψη των αριθμών των ακροδεκτών. Με αυτόν τον τρόπο, ο κώδικας γίνεται πιο ευανάγνωστός και μπορεί να τροποποιηθεί με ευκολία σε περίπτωση αλλαγής της συνδεσμολογίας.

Πίνακας 6.1: Αντιστοίχιση βασικών ακροδεκτών στο κύριο πρόγραμμα

Υποσύστημα	Σταθερά	Ακροδέκτης GPIO
Αναλογικό πληκτρολόγιο	const int keybPin	35
DAC	#define CS_PIN	17
Ποτενσιόμετρο Character	const int POT_CHARACTER	34
Ποτενσιόμετρο Amount	const int POT_AMOUNT	33
Ενισχυτής	#define SD_PIN	14
Πλήκτρο mode	#define HOLD_SWITCH_PIN	27

Βασικές μεταβλητές και αντικείμενα:

- **Voice currentVoice:** περιέχει την κατάσταση της ενεργής νότας και τις παραμέτρους που απαιτούνται για την αναπαραγωγή της.
- **volatile uint16_t audioBuffer[AUDIO_BUFFER_SIZE]:** αποθηκεύει προσωρινά δείγματα πριν από την αποστολή τους προς τον DAC.
- **currentInstrument, selectedInstrument και currentPage:** καθορίζουν το επιλεγμένο ηχητικό μοντέλο και την τρέχουσα σελίδα της διεπαφής.
- **hw_timer_t *timer και hw_timer_t *keyTimer:** χρησιμοποιούνται από τους χρονοδιακόπτες για τις συναρτήσεις διακοπής.

6.4.1 Αρχικοποίηση μέσω της Setup()

Η συνάρτηση setup() εκτελείται μία φορά κατά την εκκίνηση του προγράμματος και είναι υπεύθυνη για την αρχικοποίηση των βασικών περιφερειακών συστημάτων. Κατά την εκκίνηση του προγράμματος αρχικοποιείται η οθόνη. Έπειτα, ορίζονται οι ακροδέκτες για τον DAC και τον ενισχυτή, καθώς και ο ακροδέκτης εισόδου του πλήκτρου με εσωτερική αντίσταση pull-up.

```
// Αρχικοποίηση ψηφιακών ακροδεκτών
  pinMode(CS_PIN, OUTPUT);
  CS_HIGH();

  pinMode(SD_PIN, OUTPUT);
  digitalWrite(SD_PIN, LOW);
pinMode(HOLD_SWITCH_PIN, INPUT_PULLUP);
```

Στη συνέχεια αρχικοποιείται η επικοινωνία SPI και ορίζεται η ανάλυση των αναλογικών εισόδων στα 12bit. Παράλληλα εφαρμόζεται κατάλληλη ρύθμιση εξασθένησης στους ADC ακροδέκτες, ώστε να αναγνωρίζονται με ακρίβεια οι τάσεις από το πληκτρολόγιο και τα ποτενσιόμετρα[35].

```
SPI.begin(SCK_PIN, 19,  MOSI_PIN);

// Ρύθμιση του ADC για το πληκτρολόγιο και τα δύο ποτενσιόμετρα
  analogReadResolution(12);
  analogSetPinAttenuation(keybPin, ADC_11db);
  analogSetPinAttenuation(POT_CHARACTER, ADC_11db);
  analogSetPinAttenuation(POT_AMOUNT, ADC_11db);
```

Πριν ξεκινήσει η κανονική λειτουργία του συστήματος, αποστέλλεται στον DAC η τιμή 512. Η τιμή αυτή αντιστοιχεί στο κέντρο της 10bit κλίμακας και χρησιμοποιείται ως στάθμη ηρεμίας του αναλογικού σήματος. Αυτό έχει ως αποτέλεσμα η έξοδος του DAC να βρίσκεται σε ουδέτερη κατάσταση πριν από την αναπαραγωγή οποιαδήποτε νότας.

```
SPI.beginTransaction(SPISettings(20000000, MSBFIRST, SPI_MODE0));
CS_LOW();
SPI.transfer16(0x3000 | (512 << 2));
CS_HIGH();
delay(50);
```

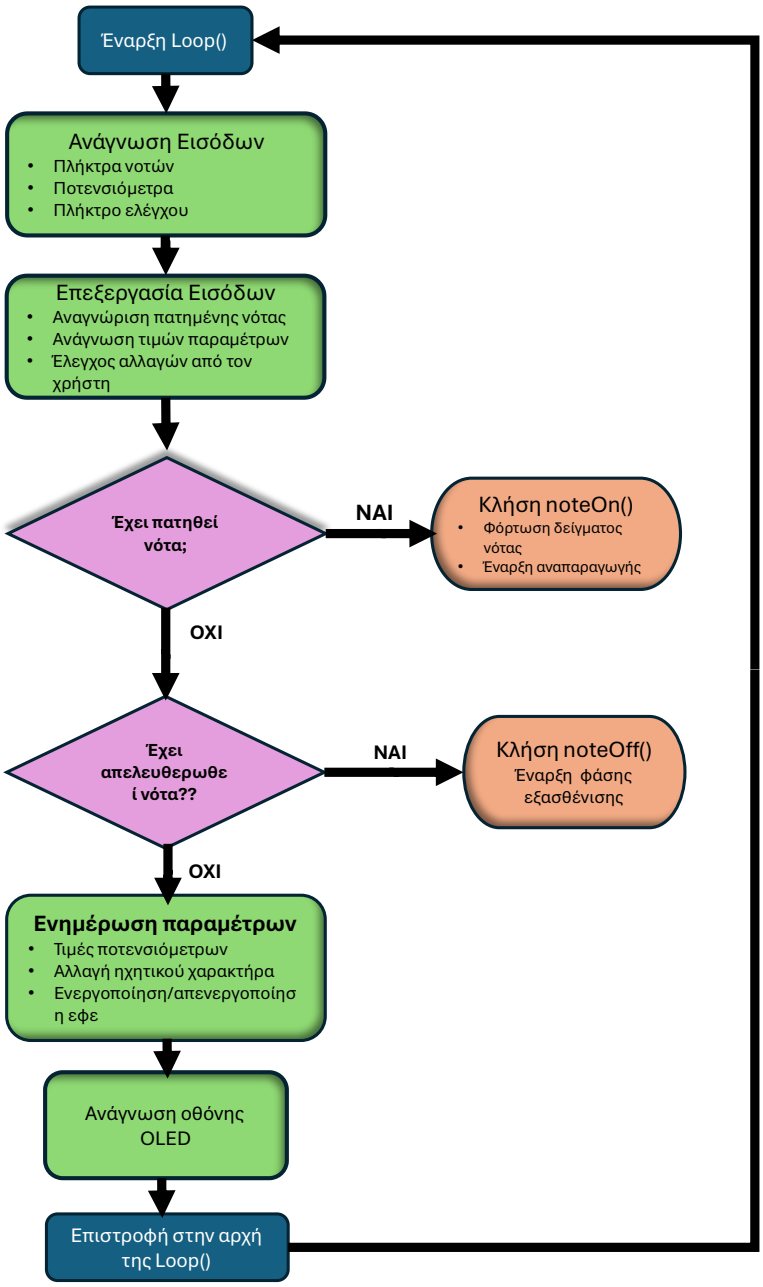
Τέλος, καλείται η συνάρτηση SetupTimers(), μέσω της οποίας ενεργοποιούνται οι hardware timers του ESP32. Στο πρόγραμμα χρησιμοποιούνται δύο timers. Ο πρώτος είναι υπεύθυνος για τη σταθερή αποστολή δειγμάτων στον DAC, ενώ ο δεύτερος χρησιμοποιείται για την περιοδική ανάγνωση του πληκτρολογίου και των εισόδων ελέγχου.

6.4.2 Κύρια λειτουργία της loop()

Μετά την ολοκλήρωση της αρχικοποίησης, το πρόγραμμα εισέρχεται στην συνάρτηση loop(), η οποία εκτελείται συνεχώς. Η συνάρτηση δεν χρησιμοποιείται για την άμεση αποστολή των δειγμάτων, καθώς η διαδικασία αυτή πρέπει να γίνεται με σταθερό χρονισμό μέσω timer interupt. Αντίθετα, η loop() αναλαμβάνει εργασίες ελέγχου και ενημέρωσης του συστήματος όπως:

- **Ανάγνωση και αναγνώριση του ενεργού πλήκτρου,**
- **Ενεργοποίηση ή απελευθέρωση της αντίστοιχης νότας,**
- **Ανάγνωση των δύο ποτενσιόμετρων ελέγχου,**
- **Ελέγχει το διακόπτη εναλλαγής σελίδων,**
- **Ενημέρωση των παραμέτρων DSP,**
- **Ενημέρωση της OLED.**

Η επιλογή αυτή διαχωρίζει τις χρονικά κρίσιμες λειτουργίες από τίς λιγότερο κρίσιμες. Η αποστολή των δειγμάτων στον DAC εκτελείται με σταθερό ρυθμό, ενώ οι υπόλοιπες λειτουργίες εκτελούνται στο κύριο πρόγραμμα. Με αυτόν τον τρόπο μειώνεται η πιθανότητα καθυστερήσεων στην αναπαραγωγή του ήχου και βελτιώνεται η σταθερότητα του συστήματος.



Σχήμα 6.1: Ροή λειτουργίας της loop()

6.5 Χρονισμός και αποστολή δειγμάτων στον DAC

Στην παρούσα ενότητα παρουσιάζεται ο τρόπος με τον οποίο υλοποιείται ο χρονισμός της της αποστολής δεδομένων, καθώς και η χρήση ενδιάμεσου audio buffer για την ομαλή αποστολή των δειγμάτων.

Η αρχικοποίηση των χρονοδιακόπτης πραγματοποιείται στη συνάρτηση SetupTimers(). Ο πρώτος timer συνδέεται με τη συνάρτηση διακοπής sendToDAC(), η οποία είναι υπεύθυνη για την αποστολή των δειγμάτων στον μετατροπέα MCP4911. Ο χρονοδιακόπτης αρχικοποιείται με βάση χρόνου 1MHz, δηλαδή κάθε μονάδα του αντιστοιχεί σε 1 μs[36].

```
void SetupTimers(){
    timer = timerBegin(1000000);
    timerAttachInterrupt(timer , &sendToDAC);
```

```
    timerAlarm(timer, 31, true, 0); // Διακοπή κάθε 31 μs

    keyTimer = timerBegin(1000000);
    timerAttachInterrupt(keyTimer , &checkKeyPress);
    timerAlarm(keyTimer, 2500, true, 0); // Διακοπή κάθε 2.5 ms
}
```

Η επιθυμητή συχνότητα αναπαραγωγής είναι τα 32kHz και συνεπώς η ιδανική περίοδος μεταξύ των δύο διαδοχικών δειγμάτων ορίζεται από τη σχέση (2.2):

$$T_s = \frac{1}{32000} \tag{6.4}$$

Οπότε:

$$T_s = 31.25\mu s$$

Η τιμή αυτή καθορίζει τον διαθέσιμο χρόνο μεταξύ δύο διαδοχικών ενημερώσεων του DAC. Η τιμή αυτή επιλέχθηκε ώστε η τελική αναπαραγωγή να είναι συμβατή με την συχνότητα δειγματοληψίας των αρχικών ηχητικών δειγμάτων.

Ο δεύτερος timer συνδέεται με τη συνάρτηση checkKeyPress() και ενεργοποιείται κάθε 2500μs, δηλαδή περίπου 400 φορές το δευτερόλεπτο. Η διαδικασία αυτή δεν εκτελεί όλη τη διαδικασία αναγνώρισης πλήκτρων μέσα στην ISR, αλλά χρησιμοποιείται κυρίως ως μηχανισμός χρονισμού, ενεργοποιώντας σημαία ελέγχου που επεξεργάζεται στη βασική ροή του προγράμματος.

6.5.1 Ενδιάμεσο audio buffer

Μεταξύ της παραγωγής και της τελικής αποστολής των δειγμάτων χρησιμοποιείται ενδιάμεσος audio buffer. Ο buffer αποτελείται από 256 θέσεις και συνοδεύεται από δείκτες ανάγνωσης και εγγραφής. Ο δείκτης ανάγνωσης χρησιμοποιείται από τη sendToDAC(), ενώ ο δείκτης εγγραφής χρησιμοποιείται από τη συνάρτηση που τον γεμίζει με νέα δείγματα.

Η χρήση του buffer επιτρέπει την προσωρινή αποθήκευση διαδοχικών τιμών και τον διαχωρισμό της παραγωγής των δειγμάτων από την αποστολή τους. Η παραγωγή του δείγματος είναι πιο σύνθετη διαδικασία, καθώς περιλαμβάνει ανάγνωση πίνακα δεδομένων, παρεμβολή, μορφοποίηση χροιάς, εφαρμογή εφέ και τελική διαμόρφωση πλάτους. Για τον λόγο αυτό η παραγωγή γίνεται εκτός της ISR, μέσα στη συνάρτηση fillAudioBuffer().

```
void fillAudioBuffer() {
  while (true) {

    portENTER_CRITICAL(&timerMux);
    bool bufferFull = (audioCount >= AUDIO_BUFFER_SIZE);
    portEXIT_CRITICAL(&timerMux);
    // Διακοπή πλήρωσης όταν ο buffer γεμίσει ή η φωνή έχει ολοκληρωθεί
    if (bufferFull) break;
    if (!currentVoice.active) break;

    float sample = currentVoice.update(maxInterp);// Παραγωγή επόμενου δείγματος

    if (sample < 0.0f) sample = 0.0f;//Περιορισμός σε 10-bit τιμή
    if (sample > 1023.0f) sample = 1023.0f;

    uint16_t dacSample = (uint16_t)sample;
    // Προστατευμένη εγγραφή δείγματος και ενημέρωση της κατάστασης του buffer
    portENTER_CRITICAL(&timerMux);
```

```
    if (audioCount < AUDIO_BUFFER_SIZE) {
      audioBuffer[audioWriteIndex] = dacSample;
      audioWriteIndex++; // Μετακίνηση της θέσης εγγραφής με κυκλική επαναφορά
      if (audioWriteIndex >= AUDIO_BUFFER_SIZE) audioWriteIndex = 0;
      audioCount++;
    }
    portEXIT_CRITICAL(&timerMux);
  }
}
```

Η fillAudioBuffer() αρχικά ελέγχει αν υπάρχει διαθέσιμος χώρος μέσα στον πίνακα. Αν ο buffer δεν είναι γεμάτος και υπάρχει ενεργή νότα, καλείται η currentVoice.update(), η οποία παράγει το επόμενο ηχητικό δείγμα και στην συνέχεια αποθηκεύεται στον buffer. Ο buffer λειτουργεί κυκλικά με την βοήθεια των audioReadIndex και audioWriteIndex, ώστε όταν ο δείκτης φτάσει στο τέλος του πίνακα να επιστρέφει στην αρχή. Έτσι αποφεύγεται η ανάγκη μετακίνησης δεδομένων στη μνήμη και η διαδικασία παραμένει αποδοτική.

Στα σημεία όπου η ISR και η loop() χρησιμοποιούν κοινές μεταβλητές, χρησιμοποιούνται κρίσιμες περιοχές με portENTER_CRITICAL και portEXIT_CRITICAL, portENTER_CRITICAL_ISR() και portEXIT_CRITICAL_ISR(). Με αυτόν τον τρόπο αποφεύγονται λανθασμένες τιμές στους δείκτες και στο πλήθος των διαθέσιμων δειγμάτων, διότι οι μεταβλητές του Buffer μπορούν να τροποποιηθούν τόσο από τη διακοπή αποστολής όσο και από τη συνάρτηση που τον γεμίζει.

6.5.2 Συνάρτηση αποστολής δειγμάτων προς τον MCP4911

Η αποστολή των ηχητικών δειγμάτων γίνεται μέσω της συνάρτησης sendToDAC(). Η συνάρτηση αυτή εκτελείται περιοδικά από τον πρώτο timer και διαβάζει κάθε φορά μια έτοιμη τιμή από τον κυκλικό audio buffer. Αν ο buffer δεν περιέχει διαθέσιμο δείγμα τότε αποστέλλεται η τιμή 512 ως στάθμη ηρεμίας.

```
void IRAM_ATTR sendToDAC() {
    uint16_t dacSample = 512;
    portENTER_CRITICAL_ISR(&timerMux);
    if (audioCount > 0) {// Μετακίνηση της θέσης ανάγνωσης με κυκλική επαναφορά
        dacSample = audioBuffer[audioReadIndex];
        audioReadIndex++;
        if (audioReadIndex >= AUDIO_BUFFER_SIZE) audioReadIndex = 0;
        audioCount--;
    }
    portEXIT_CRITICAL_ISR(&timerMux);
    uint16_t outVal = 0x3000 | (dacSample << 2);
    // Επιλογή του DAC, μετάδοση της λέξης και ολοκλήρωση της αποστολής
    CS_LOW();
    SPI.transfer16(outVal);
    CS_HIGH();
}
```

Η τιμή του δείγματος μετατρέπεται σε κατάλληλη μορφή που αναγνωρίζεται από τον MCP4911 μέσω της εντολής[13] :

```
uint16_t outVal = 0x3000 | (dacSample << 2);
```

Η μάσκα 0x3000 καθορίζει τα bit ελέγχου που χρησιμοποιούνται για να ρυθμίσουν τον DAC, ενώ η dacSample << 2 μετακινεί τα 10 bit του δείγματος στις θέσεις $D_9 - D_0$ του Πίνακα 4.2. Η τελική λέξη μεταφέρεται στον DAC μέσω SPI, με τον ακροδέκτη CS να οδηγείται σε χαμηλή στάθμη πριν από την αποστολή και να επανέρχεται σε υψηλή στάθμη μετά την ολοκλήρωση της.

Τέλος, για τον έλεγχο του ακροδέκτη CS του DAC χρησιμοποιούνται οι μακροεντολές CS_LOW() και CS_HIGH(). Οι μακροεντολές αυτές γράφουν απευθείας στους καταχωρητές GPIO του ESP32. Με αυτόν τον τρόπο μειώνεται ο χρόνος που απαιτείται για την αλλαγή κατάστασης του ακροδέκτη CS, κάτι που είναι σημαντικό για να υπάρχει σταθερή συχνότητα αποστολής στα 32kHz.

Συνολικά, η λειτουργία της ενότητας αυτής βασίζεται στον διαχωρισμό της παραγωγής και της αποστολής δειγμάτων. Η fillAudioBuffer() δημιουργεί και αποθηκεύει τα επόμενα δείγματα στον κυκλικό buffer, ενώ η sendToDAC() αναλαμβάνει μόνο την αποστολή τους στον DAC με σταθερό χρονισμό. Με αυτόν τον τρόπο μειώνεται ο υπολογιστικός φόρτος μέσα στην ISR και η αναπαραγωγή παραμένει πιο σταθερή. Η διαδικασία παραγωγής του κάθε δείγματος, η οποία πραγματοποιείται μέσω της δομής Voice, αναλύεται στην επόμενη ενότητα.

6.6 Παραγωγή και αναπαραγωγή μουσικής νότας

Η αναπαραγωγή κάθε μουσικής νότας οργανώνεται μέσω της δομής Voice, η οποία συγκεντρώνει τις μεταβλητές κατάστασης και τις συναρτήσεις που απαιτούνται από τη στιγμή που ενεργοποιείται ένα πλήκτρο έως την ολοκλήρωση της αναπαραγωγής.

6.6.1 Δομή Voice και βασικές μεταβλητές

Η χρήση της δομής επιτρέπει την καλύτερη οργάνωση του κώδικα, καθώς όλες οι πληροφορίες που σχετίζονται με την ενεργή νότα βρίσκονται συγκεντρωμένες στο ίδιο αντικείμενο. Εκτός από τους δείκτες αναπαραγωγής, η δομή αποθηκεύει και τις παραμέτρους χροιάς και εφέ που ισχύουν τη στιγμή που ενεργοποιείται μία νότα. Με αυτόν τον τρόπο, κάθε νότα μπορεί να αναπαράγεται με συγκεκριμένη κατάσταση ήχου, ακόμη και αν ο χρήστης πραγματοποιεί κάποια αλλαγή στις ρυθμίσεις του ήχου μέσω της διεπαφής.

Βασικές μεταβλητές της Voice:

- **data:** είναι δείκτης ο οποίος συνδέει την δομή με τον πίνακα δειγμάτων της ενεργής νότας. Οι πίνακες βρίσκονται στο αρχείο κεφαλίδας και η Voice διατηρεί μόνο την αναφορά προς τον κατάλληλο πίνακα.
- **sampleIndex:** καθορίζει τη θέση του δείγματος στον πίνακα που γίνεται ανάγνωση.
- **interStep**: χρησιμοποιείται για τη θέση του ενδιάμεσου δείγματος κατά τη διαδικασία παρεμβολής.
- **note:** αποθηκεύει τον αριθμητικό δείκτη της ενεργής νότας, ώστε να εφαρμοστούν επιπλέον ρυθμίσεις που εξαρτώνται από τη συγκεκριμένη νότα, όπως συχνότητες βοηθητικών ταλαντωτών.
- **active:** δηλώνει ότι υπάρχει ενεργή νότα.
- **keyHeld:** δηλώνει αν το πλήκτρο παραμένει πατημένο από τον χρήστη.
- **isReleasing:** δηλώνει ότι το πλήκτρο έχει απελευθερωθεί και η νότα βρίσκεται στην φάση της εξασθένισης.

Στην ίδια δομή αποθηκεύονται και πρόσθετες παράμετροι που χρησιμοποιούνται στα επόμενα στάδια επεξεργασίας, όπως τιμές χροιάς, counters και καταστάσεις των επιμέρους αλγορίθμων. Στο υπάρχον πρόγραμμα, οι τιμές αυτές αντιγράφονται κατά την έναρξη της νότας ώστε η ενεργή φωνή να διαθέτει τη δική της κατάσταση κατά την αναπαραγωγή.

6.6.2 Ενεργοποίηση και απελευθέρωση νότας

Η ενεργοποίηση μίας νέας νότας πραγματοποιείται μέσω της συνάρτησης noteOn(), η οποία βρίσκεται εσωτερικά στη δομή Voice. Η συνάρτηση αυτή καλείται όταν το πρόγραμμα αναγνωρίζει ότι έχει

πατηθεί κάποιο πλήκτρο. Ως ορίσματα δέχεται τον δείκτη data της αντίστοιχης νότας και τον αριθμητικό δείκτη της.

Στην αρχή της noteOn() καλείται η συνάρτηση clearAudioBuffer(). Η κίνηση αυτή μηδενίζει τον audio buffer, ώστε η νέα νότα να μην ξεκινήσει με δείγματα που είχαν παραμείνει από προηγούμενη αναπαραγωγή. Με αυτόν τον τρόπο μειώνεται η πιθανότητα να ακουστούν απότομες μεταβάσεις κατά την έναρξη μιας νέας νότας.

```
void noteOn(const uint16_t* newData, int newNote) {
      clearAudioBuffer();// Καθαρισμός του buffer

      data = newData;
      note = newNote;
      sampleIndex = 0;
      interpStep = 0;
      active = true;
      isReleasing = false; // Reset Release State
      releaseCounter = 0;
      keyHeld = true;

      // μεταβλητές μοεφοποίησης
      pmToneState  = 0.0f;
```

Στη συνέχεια, η μεταβλητή data συνδέεται με τον πίνακα δειγμάτων της επιλεγμένης νότας, ενώ η μεταβλητή note αποθηκεύει τον αριθμητικό δείκτη της νότας. Οι μεταβλητές sampleIndex και interStep μηδενίζονται με σκοπό κάθε νέα νότα να μην ξεκινά από από κάποιο λάθος σημείο του πίνακα.

Οι μεταβλητές active , keyHeld και isReleasing ορίζονται στις αρχικές τους καταστάσεις, έτσι ώστε να καθορίζουν την κατάσταση της δομής. Παράλληλη, ο μετρητής releaseCounter μηδενίζεται, ώστε σε πιθανή απελευθέρωση της νότας η εξασθένιση να ξεκινήσει από την αρχή. Στο ίδιο σημείο μηδενίζονται και οι εσωτερικές μεταβλητές της δομής, που σχετίζονται με το αρχικό fade-in, τα εφέ LFO και Delay, καθώς και οι επιμέρους μεταβλητές για την μορφοποίηση της χροιάς. Η λογική αυτή είναι σημαντική επειδή κάθε νέα νότα πρέπει να ξεκινά με καθαρή και προβλέψιμη κατάσταση επεξεργασίας.

```
void latchUiValuesForNote() {
    // Το όργανο που θα παίξει αυτή η νότα
    voiceInstrument = selectedInstrument;
    currentInstrument = voiceInstrument;

    // ORG page
    voiceOrgAmount = clamp01(uiVals.orgAmount);

    // SND page
    voiceSndChar   = clamp01(uiVals.sndChar);
    voiceSndAmount = clamp01(uiVals.sndAmount);

    // LFO page
    voiceLfoRate  = clamp01(uiVals.lfoRate);
    voiceLfoDepth = clamp01(uiVals.lfoDepth);

    // DLY page
    voiceDlyTime = clamp01(uiVals.dlyTime);
    voiceDlyMix  = clamp01(uiVals.dlyMix);
}
```

Η συνάρτηση latchUiValuesForNote(), αντιγράφει μέσα στη δομή voice τις τιμές της διεπαφής που ισχύουν τη συγκεκριμένη στιγμή που ενεργοποιείται ένα πλήκτρο. Με αυτόν τον τρόπο, η ενεργή νότα

αποκτά ένα στιγμιότυπο των ρυθμίσεων που έχουν επιλεγεί από τον χρήστη την συγκεκριμένη στιγμή. Η μεταβλητή voiceInstrument μεταφέρει την πληροφορία της μορφοποίησης, ενώ οι υπόλοιπες μεταβλητές αποθηκεύουν τις επιλεγμένες παραμέτρους χροιάς και εφέ για την συγκεκριμένη νότα.

Μετά την αποθήκευση των τιμών της διεπαφής, καλείται η συνάρτηση configureInstrument(). Η συνάρτηση αυτή δεν παράγει ήχο, αλλά χρησιμοποιείται για την αρχικοποίησή των παραμέτρων που για τη χροιά που έχει επιλεγεί. Για παράδειγμα μπορεί να ορίσει διαφορετική διάρκεια release, διαφορετικές συχνότητες ταλαντώσεων ή διαφορετικές αρχικές τιμές για συγκεκριμένες εσωτερικές μεταβλητές. Τέλος, με την εντολή digitalWrite(SD_PIN, HIGH) ο ακροδέκτης SD_PIN οδηγείται σε υψηλή στάθμη, ώστε να ενεργοποιηθεί ο ενισχυτής κατά την αναπαραγωγή.

```
void noteOff() {
  keyHeld = false;
  // Εκκίνηση του release μόνο μία φορά, εφόσον η φωνή παραμένει ενεργή
    if (active && !isReleasing) {
        isReleasing = true;
        releaseCounter = 0;
    }
}
```

Η απενεργοποίηση της νότας πραγματοποιείται με τη συνάρτηση noteOff(). Η συνάρτηση αυτή καλείται όταν το πρόγραμμα αναγνωρίζει ότι το ενεργό πλήκτρο έχει αφεθεί από τον χρήστη. Σε αντίθεση με μια απλή διακοπή της αναπαραγωγής, ο ήχος οδηγείται σε μια κατάσταση εξασθένισης.

Αρχικά, η μεταβλητή keyHeld γίνεται false, δηλώνοντας την απελευθέρωση του πλήκτρου από τον χρήστη. Στη συνέχεια, πραγματοποιείται έλεγχος για την κατάσταση του ηχητικού δείγματος, με αποτέλεσμα η μεταβλητή isReleasing να γίνεται true και ο δείκτης releaseCounter να μηδενίζεται. Με τον τρόπο αυτό, η νότα συνεχίζει να αναπαράγεται σε κατάσταση σταδιακής εξασθένισης.

Αν η έξοδος του DAC μεταφερόταν άμεσα από μία τυχαία τιμή του ηχητικού σήματος στη στάθμη ηρεμίας, θα δημιουργούσε ασυνέχεια στο σήμα, η οποία θα μπορούσε να γίνει αντιληπτή ως ανεπιθύμητος θόρυβος click ή pop. Με τη χρήση της φάσης release, η ένταση μειώνεται σταδιακά μέχρι το σήμα να επιστρέψει ομαλά στην περιοχή ηρεμίας.

Συνολικά, οι συναρτήσεις noteOn() και noteOff() δεν παράγουν άμεσα το ηχητικό αποτέλεσμα, αλλά καθορίζουν την κατάσταση της ενεργής φωνής. Η παραγωγή του τελικού δείγματος πραγματοποιείται στη συνέχεια από τις επιπλέον εσωτερικές συναρτήσεις τη Voice, οι οποίες διαβάζουν τον πίνακα δειγμάτων, εφαρμόζουν παρεμβολή, επεξεργάζονται το σήμα και επιστρέφουν την τελική τιμή στον audio buffer.

6.6.3 Παραγωγή αρχικού δείγματος και κυβική παρεμβολή

Μετά την ενεργοποίηση της νότας η ν συνάρτηση generateRawSample(), αναλαμβάνει την παραγωγή των διαδοχικών αριχκών δειγμάτων που χρησιμοποιούνται στα επόμενα στάδια επεξεργασίας. Η διαβάζει τις αποθηκευμένες τιμές από τον πίνακα δειγμάτων που υποδεικνύεται από το data και χρησιμοποιεί τη sampleIndex για τον προσδιορισμό της τρέχουσας θέσης ανάγνωσης.

```
float y0 = data[(i > 0) ? i - 1 : 0];
float y1 = data[i];
float y2 = data[(i + 1 < size) ? i + 1 : size - 1];
float y3 = data[(i + 2 < size) ? i + 2 : size - 1];

float frac = (float)interpStep / maxInterp;
float interp = cubicInterpolate(y0, y1, y2, y3, frac);
```

Σύμφωνα με την ενότητα 6.2, στην μνήμη αποθηκεύεται μικρότερο πλήθος σημείων σε σχέση με τον τελικό αριθμό δειγμάτων που πρέπει να παραχθεί. Γι' αυτό απαιτείται ο υπολογισμός ενδιάμεσων τιμών, χρησιμοποιώντας την διαδικασία της κυβικής παρεμβολής, η οποία βασίζεται σε τέσσερα γειτονικά δείγματα:

$$y0,\ y1,\ y2,\ y3 \tag{6.5}$$

Η επιλογή περισσότερων από δύο σημεία, παράγει πιο ομαλές ενδιάμεσες τιμές σε σχέση με την απλή γραμμική παρεμβολή[15].

Η μεταβλητή interStep εκφράζει το ενδιάμεσο βήμα ανάμεσα σε δύο διαδοχικά δείγματα του πίνακα, ενώ η μεταβλητή maxInterp είναι σταθερή και ορίζει πόσα ενδιάμεσα βήματα θα δημιουργηθούν. Η κλασματική θέση της παρεμβολής υπολογίζεται από τη σχέση:

$$frac = \frac{\text{interStep}}{\text{maxInterp}} \tag{6.6}$$

Η maxInterp ισούται με 4, ενώ frac είναι η κανονικποιημένη θέση ανάμεσα σε δύο διαδοχικά δείγματα. Όταν frac = 0, η έξοδος βρίσκεται στο αρχικό δείγμα, ενώ όταν η τιμή του frac αυξάνεται, η έξοδος μετακινείται σταδιακά προς το επόμενο δείγμα.

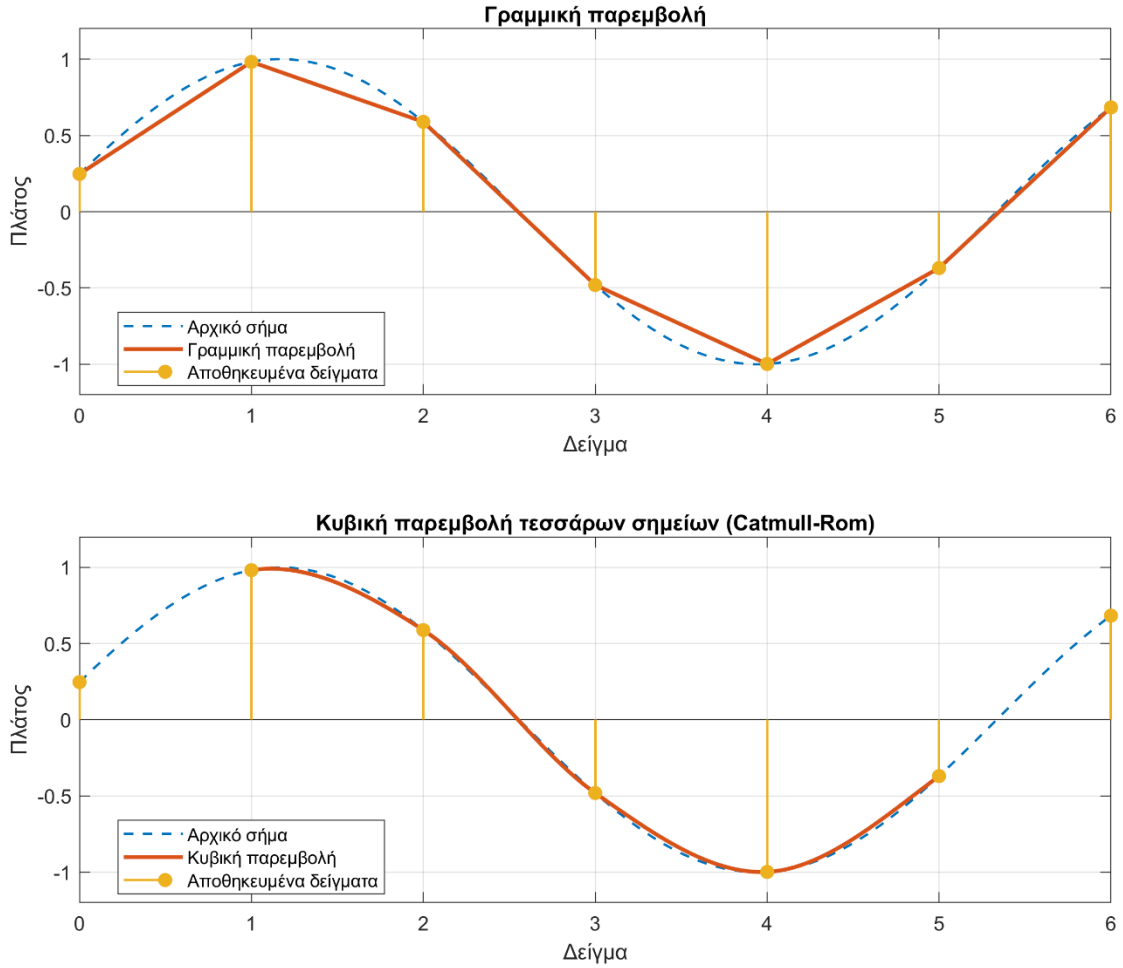
Η δημιουργία της παρεμβολής πραγματοποιείται με τη συνάρτηση cubicInterpolate(), η οποία δέχεται ως είσοδο τα τέσσερα γειτονικά δείγματα και την κλασματική θέση μεταξύ δύο διαδοχικών δειγμάτων.

```
float cubicInterpolate(float y0, float y1, float y2, float y3, float mu) {
    float a0, a1, a2, a3, mu2;
        // Συντελεστές του κυβικού πολυωνύμου
        mu2 = mu * mu;
        a0 = -0.5f * y0 + 1.5f * y1 - 1.5f * y2 + 0.5f * y3;
        a1 = y0 - 2.5f * y1 + 2.0f * y2 - 0.5f * y3;
        a2 = -0.5f * y0 + 0.5f * y2;
        a3 = y1;


    return a0 * mu * mu2 + a1 * mu2 + a2 * mu + a3;
}
```

Η έξοδος της κυβικής παρεμβολής υπολογίζεται ως πολυώνυμο τρίτου βαθμού:

$$y(u) = a_0u^3 + a_1u^2 + a_2u + a_3 \tag{6.7}$$

όπου u ισούται με την μεταβλητή frac και οι συντελεστές a_0, a_1, a_2, a_3, προκύπτουν από την ανάπτυξη της σχέσης Catmull-Rom[37]. Οι σταθεροί παράγοντες που εμφανίζονται στον κώδικα, όπως 0.5, 1.5 και 2.5, αποτελούν τους συντελεστές της αντίστοιχης κυβικής μορφής και δεν επιλέχθηκαν εμπειρικά. Με αυτόν τον τρόπο, η μετάβαση ανάμεσα στα δείγματα γίνεται πιο ομαλή, περιορίζοντας απότομες μεταβολές που θα μπορούσαν να δημιουργηθούν με την χρήση γραμμική παρεμβολής[15].



Εικόνα 6.3: Σύγκριση γραμμικής και κυβικής παρεμβολής

Στο τέλος αφαιρείται από το δείγμα που δημιουργείται η τιμή 512. Η πράξη αυτή μετατρέπει το σήμα από μια μονοπολική μορφή, σε σήμα κεντραρισμένο γύρο από το μηδέν. Η μορφή αυτή είναι κατάλληλη για την εφαρμογή επεξεργασιών, όπως η αλλαγή χροιάς, διάφορα εφέ και την διαδικασία εξασθένισης.

6.6.4 Διαμόρφωση πλάτους μέσω της applyADSR()

Μετά την παραγωγή του αρχικού δείγματος, απαιτείται η ομαλή διαμόρφωση του πλάτους του, ώστε να αποφεύγονται απότομες μεταβολές κατά την έναρξη, την απελευθέρωση και το τέλος της αναπαραγωγής. Η διαδικασία αυτή πραγματοποιείται από τη συνάρτηση applyADSR().

Παρόλο που η ονομασία της παραπέμπει σε πλήρες ADSR envelope, Attack - Decay – Sustain – Release, στην παρούσα υλοποίηση η συνάρτηση χρησιμοποιείται κυρίως για την εφαρμογή ήπιας εξομάλυνσης στην αρχή της νότας, fade-in, τελικού fade-out κοντά στο τέλος της νότας και ομαλού release.

```
float applyADSR(float sample){
    float shapedLocal = 512.0f + sample;
    // Αρχική αύξηση του πλάτους με κυβική καμπύλη εξομάλυνσης
    if (startFadeCounter < startFadeLength) {
      float x = (float)startFadeCounter / (float)startFadeLength;
      float g = x * x * (3.0f - 2.0f * x);
      shapedLocal = 512.0f + (shapedLocal - 512.0f) * g;
     }
```

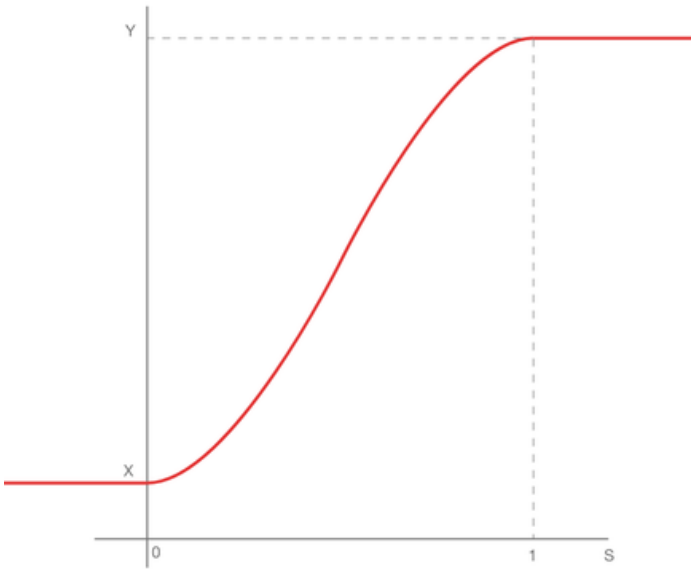
Αρχικά, το δείγμα που προκύπτει από το στάδιο επεξεργασίας μεταφέρεται ξανά γύρο από το κέντρο της 10bit κλίμακας, με την πρόσθεση της τιμής 512. Στην συνέχεια εφαρμόζεται πολύ μικρό fade-in στην αρχή της νότας. Η διαδικασία αυτή ελέγχεται από τις μεταβλητές startFadeCounter και startFadeLength. Στην παρούσα υλοποίηση ο συντελεστής ανόδου δεν αυξάνεται γραμμικά, αλλά υπολογίζεται με μία ομαλότερη καμπύλη τύπου smoothstep[38].

$$g[n] = x^2(3 - 2x) \qquad x = \frac{startFadeCounter}{startFadeLength} \tag{6.8}$$

Ο συντελεστής g[n] αυξάνεται σταδιακά από το 0 προς το 1 και εφαρμόζεται στο δείγμα με την σχέση:

$$y[n] = 512 + (s[n] - 512 \cdot g[n]) \tag{6.9}$$

όπου $s[n]$ είναι το δείγμα πριν την εφαρμογή της διαδικασίας fade-in, $y[n]$ είναι το τελικό δείγμα μετά τη διαμόρφωση και $g[n]$ είναι ο συντελεστής έντασης. Η χρήση της καμπύλης smoothstep κάνει την αρχική άνοδο πιο ομαλή σε σχέση με μία απλή γραμμική μεταβολή, μειώνοντας την πιθανότητα εμφάνισης ανεπιθύμητων click ή pop κατά την έναρξη της νότας.



Εικόνα 6.4: Αναπαράσταση καμπύλης smoothstep[39]

Επιπλέον, στα πρώτα 64 δείγματα της νότας εφαρμόζεται ένα πολύ ήπιο φίλτρο εξομάλυνσης πρώτης τάξης, ώστε να περιορίζει ακόμη περισσότερο τις απότομες μεταβολές στην αρχή της αναπαραγωγής. Το φίλτρο συνδυάζει το τρέχον δείγμα με την προηγούμενη εξομαλυμένη τιμή, χρησιμοποιώντας συντελεστή εξομάλυνσης a = 0.70. Με αυτόν τον τρόπο η εξομάλυνση επηρεάζει κυρίως την αρχή της νότας και δεν αλλοιώνει τη συνολική απόκριση του ήχου.

Παράλληλα, η συνάρτηση εφαρμόζει και ένα τελικό fade-out όταν η αναπαραγωγή πλησιάζει στο τέλος του πίνακα δειγμάτων, ενώ το πλήκτρο εξακολουθεί να είναι πατημένο από τον χρήστη. Η διαδικασία αυτή χρησιμοποιείται ώστε να αποφευχθεί απότομη διακοπή του ήχου σε περίπτωση που ο δείκτης αναπαραγωγής φτάσει στο τέλος του διαθέσιμου πίνακα δειγμάτων.

```
    if (!isReleasing && keyHeld) {
    int remainingIdx = (SampleIndexSize - 1) - sampleIndex; // Απόσταση από την
τελευταία θέση του αποθηκευμένου πίνακα
      if (remainingIdx <= endFadeLen) {
```

```
        float f = (float)remainingIdx / (float)endFadeLen;
        shapedLocal = (shapedLocal - 512.0f) * f + 512.0f;
      }
    }
```

Η μεταβλητή remainingIdx δείχνει πόσα δείγματα απομένουν μέχρι το τέλος του πίνακα. Όταν η τιμή αυτή γίνει μικρότερη ή ίση από το endFadeLen, το σήμα μειώνεται σταδιακά προς τη στάθμη ηρεμίας. Ο συντελεστής f βασίζεται στην εξίσωση (6.9). Με αυτόν τον τρόπο, ακόμη και αν η αναπαραγωγή φτάσει στο τέλος του πίνακα, το σήμα οδηγείται ομαλά προς την τιμή ηρεμίας, έτσι ώστε να μην διακόπτεται απότομα.

Η φάση release ενεργοποιείται όταν η μεταβλητή isReleasing γίνει true, κάτι που συμβαίνει όταν ο χρήστης αφήσει το πλήκτρο και κληθεί η συνάρτηση noteOff(). Χρησιμοποιείται η ίδια λογική, με τη μόνη διαφορά ότι ο συντελεστής έντασης μειώνεται σταδιακά από το 1 προς το 0.

```
if (isReleasing) {
      if (releaseCounter < releaseLength) {
        float fadeFactor = 1.0f - (float)releaseCounter / releaseLength;
        shapedLocal = (shapedLocal - 512.0f) * fadeFactor + 512.0f;
        releaseCounter++;

      } else {
          // Ολοκλήρωση της νότας και επαναφορά της κατάστασης αναπαραγωγής
          active = false;
          isReleasing = false;
          data = nullptr;
          sampleIndex = 0;
          interpStep = 0;
          digitalWrite(SD_PIN, LOW);
          return 512.0f;
       }
    }
    return shapedLocal;
  }
```

Ο συντελεστής έντασης fadeFactor υπολογίζεται από τη σχέση:

$$g[n] = 1 - \frac{\text{releaseCounter}}{\text{releaseLength}} \tag{6.10}$$

όπου στην αρχή της διαδικασίας, ο μετρητής releaseCounter είναι μηδενικός και ο συντελεστής έχει τιμή κοντά στο 1. Καθώς ο μετρητής αυξάνεται, ο συντελεστής μειώνεται σταδιακά μέχρι να φτάσει στο 0, με αποτέλεσμα το σήμα να επιστρέφει ομαλά στη στάθμη ηρεμίας.

Η χρήση αυτής της τεχνικής είναι σημαντική, επειδή η άμεση διακοπή της αναπαραγωγής θα δημιουργούσε ασυνέχεια στο σήμα. Αν η έξοδος του DAC μεταφερόταν απότομα από μια τυχαία τιμή του πίνακα δειγμάτων στη στάθμη ηρεμίας, θα δημιουργούνταν θόρυβος. Με τη φάση release, η μετάβαση γίνεται σταδιακά και το τέλος της νότας ακούγεται πιο φυσικό.

Όταν ολοκληρωθεί η διάρκεια του release, η φωνή απενεργοποιείται και οι βασικές μεταβλητές της επανέρχονται στην αρχική τους κατάσταση σαν επιπλέον ασφάλεια. Ο ακροδέκτης SD_PIN οδηγείται σε χαμηλή στάθμη, απενεργοποιώντας τον ενισχυτή. Η λειτουργία αυτή μειώνει τον ανεπιθύμητο θόρυβο κατά την διάρκεια που δεν αναπαράγεται κάποια νότα.

Συνολικά, η συνάρτηση applyADSR δεν παράγει νέα δείγματα από τον πίνακα δειγμάτων, αλλά επεξεργάζεται το ήδη παραγόμενο δείγμα ώστε η έναρξη , το τέλος και η φάση απελευθέρωσης της

νότας να πραγματοποιούνται ομαλά. Με αυτόν τον τρόπο βελτιώνεται η ακουστική συμπεριφορά του συστήματος.

6.6.5 Συνολική ροή παραγωγής δείγματος

Η βασική συνάρτηση που συντονίζει όλα τα στάδια παραγωγής είναι η update(). Η συνάρτηση αυτή καλείται από τη fillAudioBuffer κάθε φορά που χρειάζεται να δημιουργηθεί ένα νέο δείγμα για τον audio buffer.

```
//Κεντρική αλυσίδα αναπαργωγής
float update( int maxInterp){
  // Επιστροφή στάθμης ηρεμίας όταν δεν υπάρχει ενεργή πηγή δειγμάτων
   if (!active || data == nullptr) return 512.0f;

   float raw = generateRawSample(maxInterp);

   raw = applyInstrumentSound(raw);
   raw = applyLfoEffect(raw);
   raw = applyDelayEffect(raw);

   float shapedOut = applyADSR(raw);
  //Ο χρήστης απελευθέρωσε την νότα, κλείνουμε τον ενισχυτή
   if (!active) {
      digitalWrite(SD_PIN, LOW);
      return 512.0f;
  }

   advancePlayback(maxInterp);
    //Ο χρήστηςδεν απελευθέρωσε την νότα, τελείωσε ο πίνακας δείγμάτων
    if (!active) {
      digitalWrite(SD_PIN, LOW);
      return 512.0f;
  }

   return constrain(shapedOut, 32.0f, 992.0f);

}
```

Αρχικά, η update(), ελέγχει αν η φωνή είναι ενεργή και αν υπάρχει διαθέσιμος πίνακας δειγμάτων. Αν κάτι από τα δύο δεν είναι έγκυρο, η συνάρτηση επιστρέφει στον buffer την τιμή 512, δηλαδή τη στάθμη ηρεμίας.

Στη συνέχεια, καλείται η generateRawSample() η οποία παράγει το αρχικό δείγμα της νότας, ενώ αμέσως μετά εφαρμόζονται τα στάδια επεξεργασίας που θα αναλυθούν στην επόμενη ενότητα. Η applyInstrumentSound() εφαρμόζει την επιλεγμένη μορφοποίηση χροιάς στο αρχικό δείγμα, ενώ οι applyLfoEffect() και applyDelayEffect() εφαρμόζουν τα αντίστοιχα εφέ, όταν αυτά χρησιμοποιούνται από τον χρήστη.

Αφού ολοκληρωθεί η επεξεργασία του ηχητικού περιεχομένου, καλείται η applyADSR(), η οποία εφαρμόζει την τελική διαμόρφωση πλάτους. Έπειτα, η advancePlayback() προωθεί τους δείκτες αναπαραγωγής, ώστε το επόμενο κάλεσμα της update να παράγει το επόμενο χρονικό δείγμα. Η τιμή εξόδου περιορίζεται στο εύρος από 32 έως 992. Ο περιορισμός αυτός αφήνει περιθώριο ασφαλείας στα άκρα της 10bit κλίμακας και μειώνει την πιθανότητα clipping στην έξοδο του DAC. Έτσι, πρακτικά το τελικό σήμα διατηρείται σε πιο ασφαλές εύρος λειτουργίας.

Η βοηθητική συνάρτηση advancePlayback(), ενημερώνει τις μεταβλητές interpStep και sampleIndex. Η μεταβλητή interpStep αυξάνεται σε κάθε παραγόμενο δείγμα. Όταν η τιμή της φτάσει το maxInterp, μηδενίζεται ο δείκτης interpStep και αυξάνεται ο δείκτης sampleIndex, ώστε η ανάγνωση να περάσει στο επόμενο αποθηκευμένο δείγμα του πίνακα. Με αυτόν τον τρόπο, το πρόγραμμα μπορεί να παράγει περισσότερα δείγματα εξόδου από τα αποθηκευμένα που υπάρχουν στον πίνακα δειγμάτων, αξιοποιώντας την παρεμβολή που παρουσιάστηκε προηγουμένως.

```
  void advancePlayback(int maxInterp){
    interpStep++;
    // Μετά την ολοκλήρωση των βημάτων παρεμβολής προχωρά στο επόμενο αποθηκευμένο
δείγμα
    if(interpStep >= maxInterp){
      interpStep = 0;
      sampleIndex++;
    }
    if (sampleIndex >= SampleIndexSize) {
      if (keyHeld && !isReleasing) {
        active = false;
        data = nullptr;
        sampleIndex = 0;
        interpStep = 0;

      } else {
        sampleIndex = 0;
      }
    }
  }
```

Έτσι, ο hardware timer συντονίζει την αποστολή κάθε δείγματος στον dac, ενώ οι μεταβλητές interpStep, maxInterp και sampleIndex καθορίζουν ποια θέση του πίνακα δειγμάτων θα αντιστοιχεί σε κάθε παραγόμενο δείγμα.

Με τον διαχωρισμό αυτό, η συνάρτηση update()λειτουργεί ως συντονιστής των επιμέρους σταδίων, ενώ οι εξειδικευμένες λειτουργίες παραμένουν κατανεμημένες σε ανεξάρτητες συναρτήσεις μέσα στη δομή Voice. Η αρχιτεκτονική αυτή διευκολύνει τη προσθήκη διαφορετικών μορφών επεξεργασίας χωρίς να απαιτείται αλλαγή του βασικού μηχανισμού ανάγνωσης και αναπαραγωγής των νοτών.

6.7 Επιλογή και μορφοποίηση χροιάς

Η παραγωγή των διαφορετικών ηχητικών χαρακτήρων βασίζεται στην επεξεργασία του αρχικού δείγματος μέσω διαφορετικών αλγορίθμων. Το αποθηκευμένο ηχητικό δείγμα πιάνου αποτελεί τη βασική πηγή του σήματος, ενώ ανάλογα με την επιλεγμένη χροιά εφαρμόζονται διαφορετικές μορφές ψηφιακής επεξεργασίας ή προστίθενται νέες συνιστώσες. Με αυτόν τον τρόπο, το ίδιο βασικό ηχητικό υλικό μπορεί να χρησιμοποιηθεί για περισσότερες από μία ακουστικές συμπεριφορές, μειώνοντας τις απαιτήσεις αποθήκευσης καθώς, οι διαφορετικοί χαρακτήρες παράγονται σε πραγματικό χρόνο από τον ESP32.

Η λειτουργία βασίζεται στη μεταβλητή selectedInstrument, η οποία καθορίζει τον ηχητικό χαρακτήρα που έχει επιλεγεί από τον χρήστη. Όταν ενεργοποιείται μια νέα νότα, η επιλογή αυτή μεταφέρεται στη μεταβλητή voiceInstrument, ώστε η συγκεκριμένη νότα να αναπαραχθεί με τη χροιά που ίσχυε τη στιγμή της ενεργοποίησης της. Στη συνέχεια, οι συναρτήσεις της δομής Voice εφαρμόζουν την αντίστοιχη ψηφιακή επεξεργασία στο δείγμα πριν αυτό οδηγηθεί στα εφέ και στο στάδιο διαμόρφωσης πλάτους.

6.7.1 Επιλογή ηχητικού χαρακτήρα

Για την επιλογή, των διαθέσιμων χροιών χρησιμοποιείται η απαρίθμηση Instrument, στην οποία κάθε επιλογή αντιστοιχίζεται, σε μια διακριτή αριθμητική τιμή.

```
enum Instrument {
    INST_PIANO  = 0,
    INST_EPIANO = 1,
    INST_AIR    = 2,
    INST_STRING = 3,
    INST_HIT    = 4,
    INST_METAL  = 5,
    INST_RETRO  = 6

};
```

Οι διαθέσιμες επιλογές είναι Piano, Electric Piano, Air, String, Hit, Metal και Retro. Η χρήση της απαρίθμησης επιτρέπει την αναφορά στις διαφορετικές χροιές μέσω συγκεκριμένων ονομάτων και διευκολύνει την επιλογή του αντίστοιχου αλγόριθμου επεξεργασίας.

Ο χρήστης επιλέγει τον ηχητικό χαρακτήρα από τη πρώτη σελίδα ORG, με την περιστροφή του πρώτου ποτενσιόμετρου. Η κανονικοποιημένη τιμή orgSelect, βρίσκεται στο διάστημα από 0 εώς 1 και μετατρέπεται σε μία από τις επτά διαθέσιμες επιλογές μέσω της συνάρτησης instrumentFromOrgPot().

```
static Instrument instrumentFromOrgPot(float x) {
  x = clamp01(x);

  int idx = (int)floorf(x * 7.0f);

  if (idx < 0) idx = 0;
  if (idx > 6) idx = 6;// Στο x = 1 διατηρεί την τελευταία έγκυρη επιλογή

  return (Instrument)idx;
}
```

Σύμφωνα με την παραπάνω εικόνα, ο αλγόριθμος μπορεί να περιγραφεί από τη σχέση:

$$i = \lfloor 7x \rfloor \tag{6.11}$$

όπου x είναι η κανονικοποιημένη θέση του ποτενσιόμετρου και i ο δείκτης της επιλεγμένης χροιάς. Η τελική τιμή περιορίζεται στο διάστημα:

$$0 \leq i \leq 6 \tag{6.12}$$

ώστε να αντιστοιχεί πάντοτε σε ένα έγκυρο στοιχείο του Instrument. Με αυτόν τον τρόπο, η συνεχής τιμή του ποτενσιόμετρου μετατρέπεται σε διακριτή επιλογή ηχητικού χαρακτήρα.

Τέλος για λόγους σταθερότητας, η μεταβολή της selectetInstrument εφαρμόζεται όταν δεν υπάρχει ενεργή νότα. Έτσι αποφεύγεται η απότομη αλλαγή της επεξεργασίας κατά τη διάρκεια της αναπαραγωγής.

6.7.2 Μεταφορά παραμέτρων και αρχικοποίηση της χροιάς

Κατά την ενεργοποίηση μιας νέας νότας, οι τρέχουσες ρυθμίσεις της διεπαφής μεταφέρονται στη δομή Voice μέσω της LatchUiValuesForNote(). Η διαδικασία αυτή αποθηκευει την επιλεγμένη χροιά και τις τιμές των παραμέτρων που θα χρησιμοποιηθούν κατά την επεξεργασία της συγκεκριμένη νότας. Στον κώδικα, η voiceInstrument λαμβάνει την τιμή της selectetInstrument, ενώ η currentInstrument

ενημερώνεται με την ίδια επιλογή. Παράλληλα αποθηκεύονται οι παράμετροι των σελίδων ORG, SND, LFO και DLY.

Για τη μορφοποίηση της χροιάς χρησιμοποιούνται τρείς παράμετροι:

- **voiceOrgAmount:** προερχεται από την παράμετρο Amount της σελίδας ORG και καθορίζει τη συμμετοχή του πρόσθετου μοντέλου επεξεργασίας.
- **voiceSndAmount:** προέρχεται από την παράμετρο Character της σελίδας SND και χρσιμοποιείται για τη μεταβολή του χαρακτήρα του ήχου ή συγκεκριμένων παραμέτρων του εκάστοτε μοντέλου.
- **voiceSndAmount**: προέρχεται από την παράμετροAmount της σελίδας SND και καθορίζει το βάθος ή τη λεπτομέρεια της επεξεργασίας.

Οι τιμές των ποτενσιόμετρων περιορίζονται στο διάστημα [0,1] μέσω της συνάρτησης clamp01(). Όπου απαιτείται έλεγχος με διπολική ουδέτερη θέση, η voiceSndChar μετατρέπεται στο διάστημα [-1,1] μέσω της σχέσης:

$$C_b = 2(C - 0.5) \tag{6.13}$$

η οποία υλοποιείται από τη συνάρτηση toBipolar().

Μετά την αποθήκευση των παραμέτρων εκτελείται η configureInstrument(). Η συνάρτηση αυτή δεν παράγει άμεσα κάποιο δείγμα ήχου. Ο ρόλος της είναι να προετοιμάζει τις εσωτερικές μεταβλητές της δομής, ώστε η επεξεργασία που θα ακολουθήσει να γίνει με τις κατάλληλες τιμές. Για παράδειγμα, μπορεί να ορίσει διαφορετική διάρκεια release, να υλοποιήσει τιμές για βοηθητικά ημιτονικά σήματα ή να προσαρμόσει παραμέτρους που εξαρτώνται από τη συχνότητα της νότας.

Πιο συγκεκριμένα, στην βασική λειτουργία PIANO, η συνάρτηση ορίζει την διάρκεια της εξασθένισης και μηδενίζει μεταβλητές οι οποίες δεν χρησιμοποιούνται.

Στη λειτουργία EPIANO, η συνάρτηση υπολογίζει το tinePhaseStep, το οποίο χρησιμοποιείται για τη δημιουργία ενός πρόσθετου ημιτονικού στοιχείου. Η συχνότητα αυτού του στοιχείου εξαρτάται από τη συχνότητα της νότας και από την παράμετρο tineRatio, η οποία επηρεάζεται άμεσα από την τιμή του SND Character. παρόμοια λογική χρησιμοποιείται και για το hammerPhaseStep, το οποίο παράγει ένα ο σύντομο transiet.

Στη λειτουργία String, η αρχικοποίηση αφορά κυρίως τον υπολογισμό του pmStringDelaySamples. Η τιμή αυτή προκύπτει από τη σχέση ανάμεσα στη συχνότητα δειγματοληψίας και τη συχνότητα της νότας. Με αυτόν τον τρόπο, η καθυστέρηση που χρησιμοποιείται στο εσωτερικό μοντέλο της χροιάς, προσαρμόζεται στη συχνότητα της νότας που αναπαράγεται.

Στη λειτουργία HIT, υπολογίζεται η συχνότητα ενός πρόσθετου transiet στοιχείου. Η τιμή αυτή επηρεάζεται από την παράμετρο SND Character, ώστε ο χρήστης να μπορεί να μεταβάλει τον χαρακτήρα προς χαμηλότερη ή υψηλότερη περιοχή.

Αντίστοιχα, στη λειτουργία METAL, υπολογίζεται το pmModelStepMetal1, το οποίο χρησιμοποιείται για τη δημιουργία του μεταλλικού χαρακτήρα μέσω διαμόρφωσης του σήματος. Σε αυτή την περίπτωση, η τελική συχνότητα προκύπτει από τη βασική συχνότητα της νότας και από έναν λόγο που επηρεάζεται από τη ρύθμιση του χρήστη.

Στη λειτουργία RETRO, η αρχικοποίηση είναι πιο απλή και αφορά κυρίως τη μικρότερη διάρκεια εξασθένισης. Οι βασικές παράμετροι της ψηφιακής παραμόρφωσης, υπολογίζονται αργότερα μέσα στη συνάρτηση processRetro(), όπου εφαρμόζεται η πραγματική επεξεργασία του σήματος.

6.7.3 Επιλογή του αλγορίθμου επεξεργασίας

Μετά την αρχικοποίηση των παραμέτρων της ενεργής νότας, η επιλογή της κατάλληλης επεξεργασίας χροιάς πραγματοποιείται από τη συνάρτηση applyInstrumentSound(). Η συνάρτηση αυτή βρίσκεται εσωτερικά στη δομή Voice και καλείται για κάθε νέο δείγμα από την συνάρτηση update().

Η συνάρτηση δεν εφαρμόζει η ίδια πολύπλοκη επεξεργασία. Ο ρόλος της είναι να ελέγχει τη μεταβλητή voiceInstrument και μέσω της δομής switch να οδηγεί το δείγμα στην αντίστοιχη συνάρτηση επεξεργασίας.

```
float applyInstrumentSound(float raw) {
  switch (voiceInstrument) {
      case INST_PIANO:
          return processPiano(raw);

      case INST_EPIANO:
          return processEPiano(raw);

      case INST_AIR:
          return processAir(raw);

      case INST_STRING:
          return processString(raw);

      case INST_HIT:
          return processHit(raw);

      case INST_METAL:
          return processMetal(raw);

      case INST_RETRO:
          return processRetro(raw);

      default:
          return raw;
  }
}
```

Επομένως, τα διαφορετικά μοντέλα δεν εκτελούνται παράλληλα. Για κάθε παραγόμενο δείγμα ενεργοποιείται μόνο η συνάρτηση που αντιστοιχεί στην επιλεγμένη χροιά και η έξοδος της επιστρέφει στην κύρια ροή επεξεργασίας.

6.7.4 Κοινή μορφοποίηση του βασικού ηχητικού δείγματος

Η συνάρτηση applyTimberShaping() αποτελεί το κοινό στάδιο μορφοποίησης που χρησιμοποιείται από τα περισσότερα μοντέλα επεξεργασίας. Η συνάρτηση δεν παράγει ανεξάρτητο ήχο, αλλά μεταβάλει το φασματικό και αρμονικό περιεχόμενο του αρχικού δείγματος σύμφωνα με τις μεταβλητές voiceSndAmount και voiceSndChar[40].

Αρχικά, εφαρμόζοντας ένα φίλτρο πρώτης τάξης, η μεταβλητή pmToneState λειτουργεί ως μνήμη του φίλτρου και ακολουθεί σταδιακά το αρχικό σήμα. Η λειτουργία του φίλτρου μπορεί να περιγράφει από τη σχέση[41]:

$$y[n] = y[n-1] + a(x[n] - y[n-1] \tag{6.14}$$

Όπου x[n] είναι το αρχικό δείγμα, y[n] η κατάσταση του φίλτρου και a είναι ο συντελεστής απόκρισης του φίλτρου. Η έξοδος χρησιμοποιείται ως χαμηλοπερατό τμήμα και αντιστοιχεί στη μεταβλητή lowPart:

$$x_L[n] = y[n] \tag{6.15}$$

Ενώ το υψηλοπερατό φασματικό τμήμα highPart υπολογίζεται από τη διαφορά του αρχικού δείγματος και του χαμηλοπερατού μέρους:

$$x_H[n] = x[n] - x_L[n] \tag{6.16}$$

Ο διαχωρισμός αυτός επιτρέπει τη διαφορετική στάθμιση των χαμηλοπερατών και υψηλοπερατών συχνοτικών συνιστωσών. Όσο μεγαλύτερη είναι η τιμή του amt01, τόσο αυξάνεται και η τιμή του συντελεστή απόκρισης lpAlfa, με αποτέλεσμα το φίλτρο να αποκρίνεται πιο γρήγορα.

Παράλληλα δημιουργείται ένα πρόσθετο αρμονικό περιεχόμενο μέσω μη γραμμικής μορφοποίησης. Το αρχικό δείγμα κανονικοποιείται περίπου στο διάστημα [-1,1] και εφαρμόζεται ο συντελεστής drive. Η μεταβλητή drive αυξάνεται με βάση το amt01, ενώ η σχέση:

$$shaped = \frac{x \cdot drive}{1+|x \cdot drive| \cdot 0.65} \tag{6.17}$$

δημιουργεί ήπια συμπίεση του σήματος και περιορίζει τις υπερβολικές τιμές. Η διαφορά ανάμεσα στο shaped και στο αρχικό x χρησιμοποιείται ως harmonicPart, δηλαδή ως πρόσθετο αρμονικό περιεχόμενο που μπορεί να προστεθεί στη χροιά.[42]

Με βάση τις χαμηλές, υψηλές και μη γραμμικά παραγόμενες συνιστώσες, δημιουργούνται τρεις διαφορετικές μορφές χροιάς.

- **warmTarget:** ενισχύει περισσότερο το σώμα του ήχου και περιορίζει τις υψηλές συνιστώσες.
- **richTarget:** κρατάει το αρχικό δείγμα πιο κοντά στην αρχική του μορφή, προσθέτοντας όμως ήπια αρμονική πληροφορία.
- **brightTarget:** διατηρεί το αρχικό δείγμα και προσθέτει περισσότερο υψηλό περιεχόμενο και αρμονική διέγερση.

```
// Warm: περισσότερο σώμα, λιγότερες ψηλές
    float warmTarget =
        lowPart * 1.08f +
        highPart * 0.35f +
        harmonicPart * 0.20f;

    // Rich: κρατάει το piano αλλά προσθέτει αρμονικές
    float richTarget =
        raw +
        harmonicPart * 0.45f;

    // Bright: προσθέτει ψηλές και harmonic excitation
    float brightTarget =
        raw +
        highPart * 0.65f +
        harmonicPart * 0.55f;
```

Η επιλογή ανάμεσα σε αυτές τις μορφές γίνεται με βάση το ποτενσιόμετρο Character όταν η σελίδα SND είναι ενεργή. Όταν το ποτενσιόμετρο οδηγείται προς την αριστερή πλευρά, η χροιά μετακινείται από τη richTarget προς τη warmTarget. Αντίθετα, προς τη δεξιά πλευρά, μετακινείται από τη richTarget

προς τη brightTarget. Έτσι, το ποτενσιόμετρο δεν αλλάζει απλώς μία αριθμητική παράμετρο, αλλά λειτουργεί ως έλεγχος μετάβασης ανάμεσα σε τρείς διαφορετικές χρωματικές συμπεριφορές.

Η τελική έξοδος της μορφοποίησης προκύπτει από τη μίξη του αρχικού δείγματος με την επιλεγμένη χροιά βάση της ποσότητας που καθορίζεται από την amt01. Η βασική σχέση της μίξης είναι:

$$y[n] = x[n] + A(t[n] - x[n]) \tag{6.18}$$

όπου, $x[n]$ είναι το αρχικό δείγμα, $t[n]$ η επιλεγμένη μορφή χροιάς και A η τιμή του amt01. Όταν το Amount είναι μηδενικό, η έξοδος είναι ίση με το αρχικό δείγμα. Όταν αυξάνεται, η έξοδος πλησιάζει περισσότερο προς τη μορφοποιημένη χροιά. Ο πολλαπλασιασμός που ακολουθεί λειτουργεί ως μικρή αντιστάθμιση έντασης, ώστε η προσθήκη αρμονικού περιεχομένου να μην αυξάνει υπερβολικά τη στάθμη του σήματος.

```
float mixed = raw + amt01 * (target - raw);

mixed *= (1.0f - 0.08f * amt01);
return mixed;
```

Τέλος χρησιμοποιούνται δύο ακόμη βοηθητικές συναρτήσεις από τα επιμέρους μοντέλα. Η updateTimbreEnvelope() λειτουργεί ως ανιχνευτής περιβάλλουσας, υπολογίζοντας μια ομαλοποιημένη εκτίμηση του στιγμιαίου πλάτους του σήματος, ενώ η limitCenteredAudio() περιορίζει την τελική τιμή ώστε οι πρόσθετες επεξεργασίες να μην οδηγήσουν το σήμα εκτός του προβλεπόμενου εύρους.

6.7.5 Piano

Η λειτουργία INST_PIANO αποτελεί τη βασική μορφή αναπαραγωγής του πίνακα δειγμάτων. Σε αυτή τη λειτουργία δεν προστίθεται κάποιο επιπλέον μοντέλο ή ανεξάρτητο ηχητικό στοιχείο, όπως συμβαίνει σε άλλα μοντέλα χροιάς. Αντίθετα, το αρχικό δείγμα περνά από τη συνάρτηση processPiano(), η οποία εφαρμόζει μόνο τη γενική μορφοποίηση χροιάς applyTimbreShaping().

Επομένως, η βασική τεχνική παραγωγής του ήχου παραμένει η sample-based synthesis, ενώ η μεταβολή του χαρακτήρα πραγματοποιείται μέσω της φασματικής και μη γραμμικής μορφοποίησης. Σύμφωνα με την ενότητα 6.7.4:

- **To Character:** καθορίζει τη μετάβαση μεταξύ warm, Rich και Bright.
- **To Amount:** ελέγχει το βάθος της επεξεργασίας.

```
float processPiano(float raw) {
    return applyTimbreShaping(raw);
}
```

6.7.6 Electric Piano

Η λειτουργία INST_EPIANO δημιουργήθηκε με στόχο να διαφοροποιήσει το αρχικό δείγμα πιάνου και να προσεγγίσει έναν πιο ηλεκτρικό και μεταλλικό χαρακτήρα ήχου. Η επεξεργασία πραγματοποιείται στη συνάρτηση processEPiano(), η οποία δεν αντικαθιστά το αρχικό δείγμα, αλλά προσθέτει ημιτονοειδείς συνιστώσες και ένα σύντομο transient, ώστε να αυξηθεί ο μεταλλικός και κρουστικός χαρακτήρας της έναρξης.

Η λογική του μοντέλου βασίζεται στη διάκριση μεταξύ του μεταλλικού συντονισμού και της αρχικής μηχανικής διέγερσης που χαρακτηρίζουν ένα ηλεκτρομηχανικό πιάνο. Στα ηλεκτρικά πιάνα τύπου Rhodes, το hammer διεγείρει ένα μεταλλικό tine, του οποίου η ταλάντωση μετατρέπεται σε ηλεκτρικό σήμα μέσω ηλεκτρομαγνητικού pickup (μαγνήτης)[43].

Η ένταση του πρόσθετου μοντέλου συνδέεται με τη τιμή:

$$epAmt = voiceOrgAmount$$

αν η τιμή αυτή είναι πρακτικά μηδενική, η συνάρτηση επιστρέφει το αρχικό δείγμα χωρίς επιπλέον επεξεργασία. Ενώ η μεταβλητή:

$$detail = voiceSndAmount$$

χρησιμοποιείται για τη μεταβολή της δεύτερης μεταλλικής συνιστώσας και του hammer transient.

Κατά την αρχικοποίηση της νότας, η configureInstrument() υπολογίζει τη συχνότητα της μεταλλικής συνιστώσας από τη σχέση:

$$R_t = 12 + 4C_b \tag{6.19}$$

Όπου C_b η διπολική τιμή του ποτενσιόμετρου Character. Επομένως το tineRatio μεταβάλλεται από 8 έως 16. Η αντίστοιχη σχέση της φάσης tinePhaseStep υπολογίζεται από τη σχέση:

$$\Delta\Phi_t = \frac{2\pi f_0 R_t}{f_s} \tag{6.20}$$

καθώς η φάση αυξάνεται σε κάθε νέο δείγμα ως tinePhase και χρησιμοποιείται για την παραγωγή της κύριας ημιτονοειδούς συνιστώσας:

$$S_t[n] = \sin(\Phi_t[n]) \tag{6.21}$$

Η μεταβλητή έντασης, tineEnvelope μειώνεται σταδιακά μέσω του tineDecay, ώστε το πρόσθετο μεταλλικό στοιχείο να είναι πιο έντονο στην αρχή της νότας και να εξασθενεί στη συνέχεια.

```
float detail =  voiceSndAmount;

    tinePhase += tinePhaseStep;
    if (tinePhase >= TWO_PI) tinePhase -= TWO_PI;

    float tineSine = sinf(tinePhase); // Ενημέρωση φάσης και υπολογισμός της
κύριας ημιτονοειδούς συνιστώσας tine

    tineEnvelope *= tineDecay;
    if (tineEnvelope < 0.001f) tineEnvelope = 0.0f;

    float clickGain = 0.9f + (0.025f * note); // Προσαρμογή του πλάτους ανάλογα με
τον δείκτη της νότας
    float metalSound = tineSine * tineEnvelope * clickGain * tineIntensity;
```

Επιπλέον, προστίθεται μία δεύτερη αρμονική συνιστώσα, η οποία ενισχύει περισσότερο τον μεταλλικό χαρακτήρα του ήχου βάση της σχέσης:

$$S_2[n] = \sin(2.01\Phi_t[n]) \tag{6.22}$$

Της οποίας η ένταση επηρεάζεται από την παράμετρο detail. Ο συντελεστής 2.01 επιλέχθηκε ως παράμετρος ακουστικής ρύθμισης και δημιουργεί μία συνιστώσα ελαφρώς μετατοπισμένη από την δεύτερη αρμονική. Οι δύο ημιτονοδειδείς συνιστώσες χρησιμοποιούνται ως πρόσθετο αρμονικό περιεχόμενο πάνω στο πραγματικό δείγμα και επομένως η μέθοδος παρουσιάζει στοιχεία προσθετικής σύνθεσης, χωρίς όμως να αποτελεί καθαρή εκδοχή της[8].

```
float metal2Gain = 0.16f + 0.16f * detail;
float metal2 = sinf(tinePhase * 2.01f) * tineEnvelope * metal2Gain *
tineIntensity;
raw += metalSound + metal2
```

Στην συνέχεια, η αρχή της νότας εμπλουτίζεται επιπλέον μέσω του hammer transient. Ο συντελεστής env του υπολογίζεται ως:

$$E_h(x) = (1 - x)^2 \tag{6.23}$$

και μειώνεται γρήγορα από την αρχή προς το τέλος του transient, με αποτέλεσμα το hammer στοιχείο να ακούγεται μόνο στην αρχή της νότας. Η προσθήκη μικρού τυχαίου θορύβου βοηθά ώστε η έναρξη του ήχου να μην είναι τελείως καθαρή αλλά, να αποκτά πιο φυσικό και κρουστικό χαρακτήρα.

```
if (hammerCounter < hammerLength) {
      //τετραγωνική εξασθένηση από 1 προς 0
      float x = (float)hammerCounter / (float)hammerLength;
      float env = 1.0f - x;
      env = env * env;

      hammerPhase += hammerPhaseStep;
      if (hammerPhase >= TWO_PI) hammerPhase -= TWO_PI;

      float hammerTone = sinf(hammerPhase);

      rngEp = rngEp * 1664525u + 1013904223u;
      int n = (int)((rngEp >> 24) & 0xFF) - 128;
      float hammerNoise = (float)n * 0.10f;
      float hammerGain = 45.0f + 35.0f * detail;

      raw += (hammerTone * hammerGain + hammerNoise) * env * epAmt;

      hammerCounter++;
}
```

Στο τέλος της επεξεργασίας, εφαρμόζεται ήπια μη γραμμική συμπίεση και περιορισμός του σήματος, ώστε η πρόσθετες συνιστώσες να μη προκαλούν απότομο clipping.

6.7.7 Air

Η λειτουργία INST_AIR χρησιμοποιείται για τη δημιουργία μιας πιο αέρινης και ελαφριάς χροιάς ενός πνευστού οργάνου. Η επεξεργασία πραγματοποιείται από τη συνάρτηση processAir(), η οποία αρχικά εφαρμόζει τη βασική μορφοποίηση χροιάς και το αποτέλεσμα διαχωρίζεται σε χαμηλότερο και υψηλότερο φασματικό τμήμα. Στη συνέχεια προστίθεται ένα ελεγχόμενο στοιχείο θορύβου το οποίο εξομαλύνεται από τη σχέση pmAirNoiseState[44]:

$$n_f[n] = n_f[n-1] + 0.15(n[n] - n_f[n-1] \tag{6.24}$$

Η ένταση του θορύβου ακολουθεί τη δυναμική του αρχικού σήματος μέσω της updateTimbreEnvelope(). Με αυτόν τον τρόπο, η πρόσθετη συνιστώσα αυξάνεται και μειώνεται μαζί με την ενεργειακή εξέλιξη της νότας και δεν παραμένει σταθερή.

```
pmModelRng = pmModelRng * 1664525u + 1013904223u; // Παραγωγή ψευδοτυχαίου θορύβου
    float n = ((float)((pmModelRng >> 24) & 0xFF) - 128.0f) / 128.0f;

    pmAirNoiseState = pmAirNoiseState + 0.15f * (n - pmAirNoiseState);

    pmModelPhase1 += 0.001f;
    if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

    float breathLfo = 0.6f + 0.4f * sinf(pmModelPhase1); // Συντελεστής έντασης
από 0.2 έως 1.0
```

```
    float breathGain = 24.0f + 22.0f * detail;
    float breath = pmAirNoiseState * breathGain * env * breathLfo;
```

Επίσης, εφαρμόζεται αργή ημιτονοειδής μεταβολή:

$$L[n] = 0.6 + 0.4\sin(\Phi[n]) \tag{6.25}$$

και η τελική έξοδος του σήματος προκύπτει από τη μίξη του αρχικού διαμορφωμένου δείγματος με το πρόσθετο στοιχείο φιλτραρισμένου θορύβου και την ημιτονοειδή μεταβολή.

```
float highKeep = map01(char01, 0.15f, 0.65f);
    float target = lowPart + highPart * highKeep + breath;

    float out = sample + modelAmt * (target - sample); // Ανάμειξη με το σήμα
εισόδου σύμφωνα με το ORG Amount

    out *= (1.0f - 0.05f * modelAmt);
    return limitCenteredAudio(out);
}
```

Επιπλέον, παράμετροι όπως το voiceSndChar καθορίσει πόσο από το υψηλό περιεχόμενο του σήματος θα διατηρηθεί, μεταβάλλοντας τον ήχο από ήπιο και απαλό έως πιο φωτεινό, ενώ το voiceSndAmount αυξάνει την ένταση της θορυβώδους συνιστώσας. Η voiceOrgAmount καθορίζει την συνολική ανάμιξη του μοντέλου με το αρχικό δείγμα.

Συνολικά, η λειτουργία Air δεν αλλάζει ριζικά τη βασική ταυτότητα του δείγματος, αλλά το εμπλουτίζει με έναν πιο φωτεινό και ατμοσφαιρικό τρόπο. Χρησιμοποιεί τεχνικές όπως το filtered-noise excitation και το spectral shaping, δημιουργώντας μια πιο ελαφριά χροιά, κατάλληλη για ήχους που χρειάζονται περισσότερη παρουσία στις υψηλές συχνότητες.

6.7.8 String

Η λειτουργία INST_STRING χρησιμοποιείται για τη δημιουργία μιας πιο συνεχούς και συντονισμένης χροιάς, η οποία προσεγγίζει βασικές αρχές σύνθεσης ταλαντευόμενης χορδής. Η επεξεργασία πραγματοποιείται στη συνάρτηση processString() και βασίζεται στη χρήση εσωτερικού κυκλικού buffer, καθυστέρησης και ανάδρασης[45].

Κατά την αρχικοποίηση υπολογίζεται το μήκος του πίνακα καθυστέρησης:

$$N_D = \frac{f_s}{f_0} \tag{6.26}$$

όπου f_0 είναι η συχνότητα της ενεργής νότας. Η τιμή pmStringDelaySamples περιορίζεται ώστε να βρίσκεται μέσα στα διαθέσιμα όρια του buffer.

```
float detail = voiceSndAmount;
    int readIndex = pmStringWriteIndex - pmStringDelaySamples; // Υπολογισμός
θέσης ανάγνωσης στον κυκλικό buffer
    if (readIndex < 0) readIndex += PM_STRING_BUF_LEN;

    float delayed = pmStringBuf[readIndex];

    pmStringDampState = 0.5f * delayed + 0.5f * pmStringDampState; // φιλτράρισμα
του καθυστερημένου σήματος πριν από την ανάδραση

    float feedback = 0.66f + 0.20f * detail + 0.04f * modelAmt;
if (feedback > 0.92f) feedback = 0.92f;
```

Κατά την επεξεργασία η μεταβλητή delayed περιέχει ένα προηγούμενο δείγμα από τον κυκλικό buffer. Το νέο δείγμα περνά από ήπια απόσβεση μέσω της pmStringDampState, ενώ η μεταβλητή feedback επιστρέφει στον πίνακα καθυστέρησης μέσω συντελεστή ανάδρασης:

$$g = 0.66 + 0.20D + 0.04A \tag{6.27}$$

όπου D είναι το detail και A το modelAmt. Ο συντελεστής περιορίζεται σε μέγιστη τιμή 0.92, ώστε η ανατροφοδότηση να παραμένει ελεγχόμενη. Με αυτόν τον τρόπο δημιουργείται η αίσθηση συντονισμού.

Η διέγερση του μοντέλου χορδής προέρχεται κυρίως από το υψηλότερο φασματικό τμήμα του δείγματος, δηλαδή το highPart και όχι από ανεξάρτητη τυχαία ακολουθία. Η δομή διέγερση - καθυστέρηση - φίλτρο απωλειών – ανάδραση παρουσιάζει άμεση συγγένεια με τον αλγόριθμο Karplus-Strong[45].

Η τελική έξοδος προκύπτει από τη μίξη του αρχικού διαμορφωμένου δείγματος με το καθυστερημένο σήμα του buffer. Η ένταση της μίξης ελέγχεται από το modelAmt, ενώ η παράμετρος detail επηρεάζει το πόσο έντονά συμμετέχει το στοιχείο χορδής στο τελικό αποτέλεσμα.

```
float stringMix = 0.55f + 0.35f * detail;
    float out = sample + delayed * modelAmt * stringMix;

    out *= (1.0f - 0.05f * modelAmt);
```

Συνολικά, η λειτουργία String προσθέτει στο αρχικό δείγμα έναν πιο συντονισμένο και παρατεταμένο χαρακτήρα, εμπνευσμένος από τη μέθοδο Karplus-Strong χρησιμοποιώντας καθυστέρηση και ανάδραση. Έτσι, δημιουργείται ένας χαρακτήρας που θυμίζει χορδή, χωρίς να απαιτείται διαφορετικός πίνακας δειγμάτων.

6.7.9 Hit

Η λειτουργία INST_HIT χρησιμοποιείται για να δώσει στο αρχικό δείγμα πιο άμεσο και κρουστικό χαρακτήρα. Η επεξεργασία πραγματοποιείται στη συνάρτηση processHit() και βασίζεται στην προσθήκη ενός σύντομου transient στην αρχή της νότας το οποίο παράγεται από ημιτονοειδή ταλαντωτή μεταβαλλόμενης συχνότητας.

Η διάρκεια του hitLength υπολογίζεται από:

$$N_h = 900 + 1200D \tag{6.28}$$

όπου D είναι η voiceSndAmount. Με συχνότητα δειγματοληψίας 32kHz η περιοχή αυτή αντιστοιχεί περίπου σε 28 έως 66 ms.

Η παράμετρος hitEnv μειώνεται γρήγορα με κυβική μορφή, ώστε το κρουστικό στοιχείο να είναι έντονο στην αρχή της νότας και να σβήνει σύντομα. Το dynamicStep μεταβάλλει προσωρινά την συχνότητα του ταλαντωτή, δίνοντας πιο επιθετική αρχική αίσθηση.

```
int hitLength = 900 + (int)(detail * 1200.0f);
    // Παραγωγή της κρούσης μόνο κατά το αρχικό διάστημα της νότας
    if (pmModelCounter < hitLength) {
        float x = (float)pmModelCounter / (float)hitLength;
        // Κυβική εξασθένηση από  1 προς 0
        float hitEnv = 1.0f - x;
        hitEnv = hitEnv * hitEnv * hitEnv;

        float dynamicStep = pmModelStepHit * (1.0f + 3.0f * hitEnv);
```

```
        pmModelPhase1 += dynamicStep;
        if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

        float hitGain = 70.0f + 40.0f * detail;
        // Ημιτονοειδής κρούση με έλεγχο πλάτους από την hitenv και το ORG Amount
        hit = sinf(pmModelPhase1) * hitGain * hitEnv * hitAmt;
```

Η τελική έξοδος προκύπτει από τη μίξη του βασικού σώματος του δείγματος με την πρόσθετη συνιστώσα hit. Το dryBody κρατά μέρος του αρχικού σήματος, ενώ το hit προσθέτει το σύντομο κρουστικό στοιχείο. Η voiceSndAmount μεταβάλει ταυτόχρονα τη διάρκεια και την ένταση της συνιστώσας hit, ενώ η voiceOrgAmount καθορίζει τη συνολική συμμετοχή του μοντέλου.

6.7.10 Metal

Η λειτουργία INST_METAL χρησιμοποιείται για τη δημιουργία πιο μεταλλικής χροιάς. Η επεξεργασία πραγματοποιείται στη συνάρτηση processMetal() και βασίζεται στη τεχνική διαμόρφωσης Δακτυλίου (Ring Modulation) του αρχικού σήματος, πολλαπλασιάζοντας το με έναν επιπλέον ταλαντωτή[8].

```
pmModelPhase1 += pmModelStepMetal1; // Ενημέρωση της φάσης του ημιτονοειδούς διαμορφωτή
    if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

    float modulator = sinf(pmModelPhase1);
    float rmSignal = sample * modulator; // Δακτυλιοειδής διαμόρφωση με πολλαπλασιασμό του σήματος και του διαμορφωτή
```

Αρχικά δημιουργείται το ημιτονοειδές σήμα διαμόρφωσης με τη σχέση:

$$m[n] = \sin(\Phi_m[n]) \tag{6.29}$$

όπου, $\Phi_m[n]$ ορίζεται η μεταβλητή pmModelPhase1 αυξάνοντας την τιμή της σε κάθε νέο δείγμα με βάση το pmModelStepMetal, το οποίο έχει υπολογιστεί κατά την αρχικοποίηση της νότας. Στη συνέχεια, το αρχικό δείγμα πολλαπλασιάζεται με το modulator, δημιουργώντας το σήμα rmSignal, η οποία καθορίζεται από τη σχέση:

$$r[n] = x[n]m[n] \tag{6.30}$$

Η μεταβλητή που καθορίζει πόσο έντονα θα συμμετέχει το μεταλλικό σήμα στο τελικό αποτέλεσμα είναι η metAmt. Η τιμή της εξαρτάται από το modelAmt και από την παράμετρο detail. Η τελική έξοδος προκύπτει από τι μίξη του αρχικού δείγματος με το διαμορφωμένο rmSignal. Στην πράξη, ο πολλαπλασιασμός δύο σημάτων δημιουργεί νέες συχνοτικές συνιστώσες που σχετίζονται με το άθροισμα και τη διαφορά των αρχικών συχνοτήτων[6].

Στην υλοποίηση του μοντέλου, η voiceOrgAmount καθορίζει τη συνολική συμμετοχή του επεξεργασμένου σήματος, ενώ η voiceSndAmount μεταβάλλει επιπλέον το βάθος της μεταλλικής χροιάς. Το αρχικό δείγμα παραμένει στη βάση του ήχου και αναμιγνύεται γραμμικά με το επεξεργασμένο σήμα, επιτρέποντας τη σταδιακή μετάβαση του αρχικό δείγματος προς τη μεταλλική χροιά.

6.7.11 Retro

Η λειτουργία INST_RETRO χρησιμοποιείται για τη δημιουργία πιο τραχιάς και ψηφιακής χροιάς, παρόμοιας με ήχο χαμηλότερης ανάλυσης. Η επεξεργασία πραγματοποιείται στη συνάρτηση processRetro() και βασίζεται στη σκόπιμη υποβάθμιση της ακρίβειας του πλάτους και στη συγκράτηση του δείγματος για περισσότερους κύκλους.

```
downsampleFactor = 8 + (int)(charBipolar * 6.0f);
    if (downsampleFactor < 2)  downsampleFactor = 2;
    if (downsampleFactor > 16) downsampleFactor = 16;

    bitStep = (int)map01(detail, 6.0f, 48.0f); // To SND Amount καθορίζει το βήμα
κβάντισης πλάτους από 6 έως 48
    if (bitStep < 6) bitStep = 6;

    int crushedInt = (int)sample;
    crushedInt = (crushedInt / bitStep) * bitStep;
    float crushedRaw = (float)crushedInt;
```

Η πρώτη επεξεργασία αφορά την κβάντιση του πλάτους, επηρεάζοντας πόσο "τετραγωνισμένο" γίνεται το σήμα. Το βήμα καθορίζεται από τη bitStep, η οποία μεταβάλλεται από 6 έως 48. Η τιμή του δείγματος προσεγγίζεται από[6]:

$$x_q = \left(\frac{x}{Q}\right) Q \tag{6.31}$$

Όπου Q είναι η τιμή bitStep. Έπειτα το δείγμα μετατρέπεται σε ακέραια τιμή και περιορίζεται σε διακριτά επίπεδα με βάση το bitStep. Όσο το βήμα μεγαλώνει, τόσο λιγότερες διαφορετικές στάθμες πλάτους μπορούν να εμφανιστούν, με αποτέλεσα να αυξάνεται η παραμόρφωση κβάντισης.

Παράλληλα η μεταβλητή crushCounter καθορίζει πότε θα ανανεωθεί η τιμή της heldSample. Μέχρι να ξεπεράσει την μεταβλητή downsampleFactor, η ίδια τιμή επεξεργασίας παραμένει σταθερή, δημιουργώντας την αίσθηση χαμηλότερης χρονικής ανάλυσης.

Η voiceSndChart μετατρέπεται σε διπολική τιμή και επηρεάζει τον υπολογισμό του downsampleFactor, ενώ η voiceSndAmount καθορίζει το βήμα κβάντισης. Τέλος, η voiceOrgAmount ελέγχει την ένταση R του υποβαθμισμένου σήματος, η οποία ελέγχει την τιμή της εξόδου, που προκύπτει από τη μίξη του αρχικού δείγματος με το heldSample:

$$y[n] = (1 - R)x[n] + Rx_{held}[n] \tag{6.32}$$

Συνολικά, η συγκεκριμένη τεχνική δεν αποτελεί κανονική διαδικασία επαναδειγματοληψίας, καθώς δεν εφαρμόζεται φίλτρο anti-aliasing πριν από την μείωση του sample rate. Η παραμόρφωση του ψηφιακού σήματος αποτελεί μέρος του επιθυμητού ηχητικού αποτελέσματος[46].

6.8 Εφαρμογή εφέ LFO και Delay

Μετά την παραγωγή του αρχικού δείγματος και τη διαμόρφωση της χροιάς του, το σήμα μπορεί να υποστεί πρόσθετη πεξεργασία μέσω δύο εφέ. Διαμόρφωση πλάτους με ταλαντωτή χαμηλής συχνότητας (LFO) και χρονική καθυστέρηση (Delay). Οι παράμετροι των εφέ ρυθμίζονται από τις αντίστοιχες σελίδες της διεπαφής χρήστη και μεταφέρονται στις μεταβλητές της ενεργής νότας κατά την ενεργοποιησή της. Η επεξεργασία πραγματοποιείται ξεχωριστά για κάθε παραγόμενο δείγμα σε πραγματικό χρόνο πριν από την τελική διαμόρφωση πλάτους μέσω της applyADSR().

6.8.1 Διαμόρφωση πλάτους μέσω LFO – Tremolo

Το πρώτο επιπλέον στάδιο επεξεργασίας είναι το εφέ LFO, το οποίο υλοποιείται μέσω της συνάρτησης applyLfoEffect(). Στην παρούσα υλοποίηση ο ταλαντωτής χαμηλής συχνότητας χρησιμοποιείται έτσι ώστε, να μεταβάλλει περιοδικά το πλάτος του τελικού ήχου[46]. Επομένως, το ακουστικό αποτέλεσμα που παράγεται είναι tremolo.

Η ταχύτητα του LFO καθορίζεται από τη μεταβλητή lfoRate01, η οποία προέρχεται από τη διεπαφή χρήστη. Η τιμή αυτή μέσω της συνάρτησης mapLfoRate() μετατρέπεται σε πραγματική συχνότητα. Με αυτόν τον τρόπο, η συχνότητα μεταβάλλεται περίπου από 0.3 Hz έως 8 Hz. Οι χαμηλές συχνότητες διαμόρφωσης δημιουργούν αργή μεταβολή της έντασης, ενώ οι υψηλότερες οδηγούν σε πιο γρήγορο τρέμουλο.

```
float rateHz = mapLfoRate(lfoRate01);   // 0.3 .. 8.0 Hz
    float depth  = maxDepth * clamp01(lfoDepth01); // 0 .. 0.8
      // Εφαρμογή tremolo μόνο όταν το βάθος υπερβαίνει το κατώφλι ενεργοποίησης
      if (depth > 0.001f) {
        uint32_t step = (uint32_t)((4294967296.0f * rateHz) / SAMPLE_RATE);
        tremoloPhase += step; //Επαναφέρει κυκλικά τη φάση

        float lfo = triLFO(tremoloPhase);          // -1 .. +1
        float lfo01 = 0.5f + 0.5f * lfo;       // 0 .. 1
```

Η φάση του LFO αποθηκεύεται στη μεταβλητή tremoloPhase. Σε κάθε νέο δείγμα, η φάση αυξάνεται κατά ένα βήμα step, το οποίο εξαρτάται από τη συχνότητα του LFO και τη συχνότητα δειγματοληψίας του συστήματος. Η διαδικασία αυτή βασίζεται στη λογική του συσσωρευτή φάσης. Η πλήρης περίοδος αντιστοιχεί στο εύρος τιμών ενός 32-bit μετρητή, δηλαδή το βήμα της φάσης υπολογίζεται από τη σχέση:

$$step = \frac{2^{32} \cdot f_{LFO}}{f_s} \tag{6.33}$$

όπου f_{LFO} είναι η συχνότητα του ταλαντωτή και f_s η συχνότητα δειγματοληψίας 32kHz.

Η κυματομορφή του LFO παράγεται από τη συνάρτηση triLFO(), η οποία δημιουργεί τριγωνική κυμματομορφή στο διάστημα από -1 έως 1. Η συνάρτηση χρησιμοποιεί σαν είσοδο την μεταβλητή phase, η οποία προέρχεται από την μεταβλητή της φάσης tremoloPhase.

```
inline float triLFO(uint32_t phase) {
  uint32_t x = phase >> 16;
  uint32_t t = (x & 0x8000) ? (0xFFFF - x) : x; // Αναδίπλωση

  return ((float)t / 32767.0f) * 2.0f - 1.0f;
}
```

Η συνάρτηση triLFO(), χρησιμοποιεί αρχικά τα 16 πιο σημαντικά bit της φάσης. Στη συνέχεια, η τιμή αυτή διπλώνεται ώστε να δημιουργηθεί τριγωνική μορφή. Για το πρώτο μισό της περιόδου, η τιμή αυξάνεται από το 0 έως το 32767, ενώ το δεύτερο μισό μειώνεται ξανά προς το 0. Τέλος η τιμή t , μετατρέπεται ξανά στο διάστημα -1 έως 1, ώστε η έξοδος να μπορεί να χρησιμοποιηθεί ως κανονικοποιημένο σήμα LFO.

Στην συνέχεια, η έξοδος αυτή μετατρέπεται σε συντελεστή πλάτους, καθώς κανονικοποιείται στο διάστημα 0 έως 1 και εφαρμόζεται στο ηχητικό δείγμα. Η γενική λειτουργία του tremolo μπορεί να περιγραφεί μαθηματικά ως:

$$y[n] = x[n]G[n] \tag{6.34}$$

όπου $x[n]$ είναι το αρχικό δείγμα και $G[n]$ ο περιοδικά μεταβαλλόμενος συντελεστής έντασης που προκύπτει από τον LFO.

Το βάθος του εφέ καθορίζεται από τον χρήστη με την παράμετρο depth. Όταν το depth έχει μικρή τιμή, η μεταβλητή gain παραμένει κοντά στη μονάδα και η μεταβολή της έντασης είναι σχετικά μικρή. Όταν το depth αυξάνεται, η περιοδική μεταβολή γίνεται πιο έντονη και το tremolo γίνεται περισσότερο αντιληπτό.

Συνολικά, το εφέ αποτελεί τεχνική διαμόρφωσης πλάτους χαμηλής συχνότητας, η οποία προσθέτει κίνηση στο ηχητικό σήμα, μεταβάλλοντας περιοδικά το πλάτος του. Η υλοποίηση με συμπιεστή φάσης επιτρέπει σταθερή χρονική συμπεριφορά, ενώ η χρήση τριγωνικής κυματομορφής προσφέρει ομαλή και ελεγχόμενη μεταβολή του πλάτους.

6.8.2 Εφαρμογή χρονικής καθυστέρησης – Delay

Το δεύτερο επιπλέον στάδιο επεξεργασίας delay εφέ, βασίζεται στην αποθήκευση προηγούμενων δειγμάτων του ηχητικού σήματος και στην ανατροφοδότηση τους στην έξοδο μετά από συγκεκριμένο χρονικό διάστημα[46]. Η λειτουργία αυτή υλοποιείται μέσω της συνάρτησης applyDelayEffect(), την οποία χρησιμοποιείται κυκλικός buffer, ο οποίος επιτρέπει τη συνεχή εγγραφή νέων δειγμάτων και την ανάγνωση παλαιότερων χωρίς να απαιτείται μετακίνηση των δεδομένων στη μνήμη.

Η μεταβλητή delayTime εκφράζεται σε ms και μετατρέπεται σε αριθμό δειγμάτων με βάση τη συχνότητα δειγματοληψίας:

$$D = \frac{T_d \cdot f_s}{1000} \tag{6.35}$$

όπου D είναι η καθυστέρηση σε δείγματα, T_d ο χρόνος καθυστέρησης σε ms και f_s η συχνότητα δειγματοληψίας 32kHz. Για παράδειγμα, μεγαλύτερος χρόνος καθυστέρησης οδηγεί σε μεγαλύτερη απόσταση ανάμεσα στο αρχικό σήμα και την επανάληψη του.

Ο πίνακας dlyBuffer[] λειτουργεί κυκλικά. Ο δείκτης dlyWriteIndex προσδιορίζει τη θέση στην οποία θα αποθηκευτεί το νέο δείγμα, ενώ η θέση ανάγνωσης υπολογίζεται μετατοπισμένη κατά N_D θέσεις:

$$i_{read} = (i_{write} - N_D) mod N \tag{6.36}$$

όπου N είναι το συνολικό μήκος του πίνακα καθυστέρησης. Όταν ο δείκτης ανάγνωσης, readIndex, περάσει κάτω από το μηδέν, επιστρέφει στο τέλος του πίνακα. Επειδή ο buffer είναι κυκλικός η πρόσθεση του DLY_BUF_LEN είναι απαραίτητη, έτσι ώστε ο ίδιος πίνακας να χρησιμοποιείται συνεχώς χωρίς να χρειάζεται μετακίνηση δεδομένων στη μνήμη.

Το καθυστερημένο δείγμα μπορεί επομένως να γραφεί ως:

$$y[n] = x[n - N_D] \tag{6.37}$$

Και η τελική έξοδος προκύπτει από την ανάμιξη του αρχικού και του καθυστερημένου σήματος, σύμφωνα με τον συντελεστή μίξης:

$$y[n] = x[n] + M \cdot d[n] \tag{6.38}$$

όπου, x[n] το αρχικό δείγμα, d[n] το καθυστερημένο δείγμα και M ο συντελεστής μίξης. Παράλληλα, στον buffer αποθηκεύεται το σήμα με μειωμένη ένταση:

$$b[n] = x[n] + F \cdot d[n] \tag{6.39}$$

Όπου,

$$F = 0.45 \cdot M \tag{6.40}$$

Με αυτόν τον τρόπο δημιουργείται ελεγχόμενη ανάδραση, η οποία επιτρέπει την ύπαρξη επαναλήψεων χωρίς υπερβολική αύξηση της στάθμης. Έτσι, το σύστημα αποκτά μορφή ήπιας ανάδρασης, καθώς μέρος της προηγούμενης εξόδου επανέρχεται ξανά στο σήμα. Η τιμή του delayMix περιορίζεται μέχρι 0.35, ώστε οι επαναλήψεις να είναι ακουστικά αντιληπτές, αλλά να μην αυξάνεται υπερβολικά η στάθμη του σήματος.

Μετά την αποθήκευση του νέου δείγματος, ο δείκτης εγγραφής dlyWriteIndex αυξάνεται κατά μία θέση. Ο buffer έχει μέγεθος 16384 δείγματα, 2^{14} γεγονός που διευκολύνει την υλοποίηση κυκλικής προσπέλασης στη μνήμη. Σε συνδυασμό με συχνότητα δειγματοληψίας 32kHz η διάρκεια καθυστέρησης αντιστοιχεί σε περίπου 512ms. Όταν ο δείκτης εγγραφής φτάσει στο τέλος της διαθέσιμης μνήμης, επιστρέφει στην αρχική θέση, υλοποιώντας με αυτόν τον τρόπο κυκλική λειτουργία του buffer.

Συνολικά, η χρήση κυκλικού buffer είναι ιδιαίτερα σημαντική σε εφαρμογές πραγματικού χρόνου, επειδή επιτρέπει συνεχή αποθήκευση και ανάγνωση δειγμάτων με σταθερό κόστος επεξεργασίας. Οι αριθμητικές τιμές του delay εφέ, όπως το εύρος 80-420 ms και το μέγιστο ποσοστό μίξης 0.45, επιλέχθηκαν πειραματικά. Οι τιμές αυτές επιτρέπουν την παραγωγή σύντομων έως μέτριων επαναλήψεων, χωρίς το εφέ να καλύπτει πλήρως το αρχικό σήμα, αλλά να προσφέρει στο σήμα αίσθηση χώρου και διάρκειας.

6.9 Διεπαφή χρήστη και έλεγχος παραμέτρων

Η διεπαφή χρήστη του συστήματος επιτρέπει την ρύθμιση των βασικών παραμέτρων του synthesizer μέσω δύο ποτενσιόμετρων, ενός διακόπτη επαφής και της οθόνης OLED. Επειδή ο αριθμός των διαθέσιμων παραμέτρων είναι μεγαλύτερος από τον αριθμό των διαθέσιμων χειριστηρίων, οι λειτουργίες οργανώνονται σε διαφορετικές σελίδες. Σε κάθε σελίδα τα δύο ποτενσιόμετρα αποκτούν διαφορετική λειτουργία, ενώ στην οθόνη εμφανίζεται η ενεργή σελίδα και οι αντίστοιχες τιμές της. Η λογική της διεπαφής εκτελείται ανεξάρτητα από τη διαδικασία παραγωγής των ηχητικών δειγμάτων, ώστε η ανάγνωση των χειριστηρίων και η ενημέρωση της οθόνης να μην επιβαρύνουν τον μηχανισμό αποστολής δεδομένων.

Οι λειτουργίες των σελίδων οργανώνονται αναλυτικά στον πίνακα 6.2:

Πίνακας 6.2: Ανάλυση των σελίδων ελέγχου

Σελίδα	Πρώτο Ποτενσιόμετρο	Δεύτερο Ποτενσιόμετρο
ORG	Επιλογή ηχητικού μοντέλου	Ένταση μορφοποίησης
SND	Χαρακτήρας	Ποσότητα
LFO	Ρυθμός	Βάθος/Ένταση
DLY	Χρόνος	Μίξη

Η χρήση κοινών χειριστηρίων για διαφορετικές παραμέτρους μειώνει τον απαιτούμενο αριθμό εξαρτημάτων, αλλά δημιουργεί την ανάγκη για επιπλέον λειτουργίες διαχείριση της τιμής αποθηκευμένης τιμής των ποτενσιόμετρων, όπως κβάντιση των εμφανιζόμενων τιμών και λειτουργία soft takeover.

6.9.1 Οργάνωση των σελίδων

Για τη διαχείριση των παραμέτρων της διεπαφής χρησιμοποιείται η δομή UiValues. Κάθε στοιχείο αντιστοιχεί σε ξεχωριστές τιμές για κάθε σελίδα ρυθμίσεων, ώστε τα δύο ποτενσιόμετρα να χρησιμοποιούνται για διαφορετικές λειτουργίες ανάλογα με την ενεργή σελίδα.

Πιο συγκεκριμένα , η σελίδα ORG αφορά την επιλογή του ηχητικού μοντέλου και το ποσοστό συμμετοχής της αντίστοιχης μορφοποίησης. Η σελίδα SND αφορά τις παραμέτρους χροιάς, η σελίδα LFO αφορά το tremolo εφέ και η σελίδα DLY αφορά το εφέ καθυστέρησης.

Οι τιμές των παραμέτρων αποθηκεύονται ανεξάρτητα. Επομένως, η μετάβαση σε άλλη σελίδα δεν διαγράφει τις προηγούμενες ρυθμίσεις. Για παράδειγμα, οι τιμές lfoRate και lfoDepth παραμένουν αποθηκευμένες όταν ο χρήστης αλλάξει σελίδα, ενώ οι τιμές διατηρούνται όταν ο χρήστης επιστρέψει στην ίδια σελίδα.

Η αντιστοίχιση των δύο ποτενσιόμετρων μεταβάλλεται επομένως λογικά και όχι ηλεκτρικά. Οι ίδιες δύο αναλογικοί είσοδοι διαβάζονται συνεχώς, αλλά το λογισμικό αποφασίζει ποια παράμετρο θα ενημερώσει με βάση την currentPage.

6.9.2 Λειτουργία του πλήκτρου ελέγχου

Το πλήκτρο ελέγχου χρησιμοποιείται για την μετάβαση μεταξύ των σελίδων και για την επαναφορά του συστήματος στην αρχική του κατάσταση. Το λογισμικό παρακολουθεί την κατάσταση του διακόπτη και αναγνωρίζει τις ενέργειες του χρήστη.

Με ένα σύντομο πάτημα, η μεταβλητή currentPage αυξάνεται κυκλικά με τη χρήση της πράξης modulo. Επειδή οι διαθέσιμες σελίδες είναι τέσσερις, χρησιμοποιείται η τιμή τέσσερα, ώστε μετά τη σελίδα DLY η διεπαφή να επιστρέφει ξανά στη σελίδα ORG. Ενώ, με παρατεταμένο πάτημα καλείται η resetSoundRawPiano(), η οποία επαναφέρει το σύστημα σε καθαρή αρχική μορφή Piano, μηδενίζοντας παράλληλα και όλες τις προηγούμενες παραμέτρους επεξεργασίας.

Για την αποφυγή ψευδών ενεργοποιήσεων από τη μηχανική αναπήδηση του διακόπτη, εφαρμόζεται χρονική καθυστέρηση επιβεβαίωσης της αλλαγής κατάστασης. Με αυτόν τον τρόπο, μία φυσική πίεση του πλήκτρου αναγνωρίζεται ως μία μόνο εντολή και αποφεύγονται ανεπιθύμητες διπλές αλλαγές σελίδας ή λανθασμένη αναγνώριση παρατεταμένου πατήματος.

6.9.3 Ανάγνωση και επεξεργασία των ποτενσιόμετρων

Τα δύο ποτενσιόμετρα της διεπαφής διαβάζονται από τις εισόδους ADC του ESP32. Η ψηφιακή τιμή που προκύπτει μετατρέπεται σε κανονικοποιημένη αριθμητική τιμή. Η κανονικοποιημένη τιμή μπορεί γενικά να εκφραστεί ως:

$$\mathrm{x} = \frac{ADC}{ADC_{max}} \tag{6.41}$$

όπου $ADC_{max} = 4095$ λόγο της ανάλυσης 12-bit του ADC και περιορίζεται στο εύρος:

$$0 \leq x \leq 1 \tag{6.42}$$

Έτσι, η ελάχιστη τιμή μετατρέπεται κοντά στο 0, ενώ η μέγιστη τιμή μετατρέπεται κοντά στο 1. Στην περίπτωση παραμέτρων που απαιτούν ουδέτερη κεντρική θέση, η κανονικοποιημένη περιοχή μετατρέπεται σε διπολική μορφή σύμφωνα με την εξίσωση (σελίδα 78) και διαθέτει εύρος από -1 έως 1.

Στην υλοποίηση χρησιμοποιείται επίσης δυνατότητα αντιστροφής της φοράς των ποτενσιόμετρων. Με αυτόν τον τρόπο, η αύξηση ή μείωση της τιμής μπορεί να προσαρμοστεί στη φυσική τοποθέτηση των ποτενσιόμετρων πάνω στην κατασκευή, ώστε η φορά περιστροφής να είναι πιο φυσική για τον χρήστη.

Για την βελτίωση της σταθερότητας, οι τιμές των ποτενσιόμετρων εξομαλύνονται με απλό φίλτρο πρώτης τάξης. Η διαδικασία αυτή περιορίζει τις μικρές διακυμάνσεις των μετρήσεων του ADC και κάνει τη μεταβολή των παραμέτρων πιο ομαλή. Σύμφωνα με την ανάπτυξη της εξίσωσης (6.12) και $a = 1 - a$ έχουμε την σχέση:

$$y[n] = a \cdot y[n-1] + (1-a)x[n] \tag{6.43}$$

Στην παρούσα υλοποίηση, η προηγούμενη τιμή έχει βάρος 0.90 και η νέα μέτρηση βάρος 0.10. Με αυτόν τον τρόπο περιορίζονται οι μικρές διακυμάνσεις των ADC μετρήσεων και η μεταβολή των παραμέτρων γίνεται πιο ομαλή. Οι δύο εξομαλυμένες τιμές δεν έχουν πάντα την ίδια σημασία. Η λειτουργία τους εξαρτάται από την ενεργή σελίδα της διεπαφής.

```
potCharacterSmooth = 0.90f * potCharacterSmooth + 0.10f * characterNow;
potAmountSmooth    = 0.90f * potAmountSmooth    + 0.10f * amountNow;
```

6.9.4 Κβάντιση των εμφανιζόμενων τιμών

Για την απλούστερη και περισσότερο ευανάγνωστη παρουσίαση των παραμέτρων στην OLED, οι εμφανιζόμενες τιμές στις μπάρες, δεν ακολουθούν κάθε μικρή μεταβολή του ADC. Αντίθετα, εφαρμόζεται από τον κώδικα αλγοριθμική κβάντιση, ώστε ο χρήστης να βλέπει πιο καθαρές και κατανοητές τιμές.

Στις μονοπολικές παραμέτρους χρησιμοποιείται η συνάρτηση quantizeTo10(), η οποία μετατρέπει την τιμή 0-1 σε βαθμίδες από 0 έως 10.

```
static uint8_t quantizeTo10(float x) {
  x = clamp01(x);
  int steps = (int)lroundf(x * 10.0f);   // 0..10
  if (steps < 0) steps = 0;
  if (steps > 10) steps = 10;
  return (uint8_t)steps;
}
```

Η συνάρτηση αυτή χρησιμοποιείται για παραμέτρους που έχουν φυσική ερμηνεία από ελάχιστη έως μέγιστη τιμή, όπως Amount, Lfo Rate, Lfo Depth, Delay Time και Delay Mix. Με αυτόν τον τρόπο, η οθόνη μπορεί να εμφανίζει απλές διακριτές ενδείξεις, χωρίς να παρουσιάζει συνεχώς μικρές μεταβολές που οφείλονται στις αναλογικές μετρήσεις.

Για παραμέτρους που έχουν ουδέτερη κεντρική θέση, όπως το Character της σελίδας SND, χρησιμοποιείται η συνάρτηση quantizeBipolar(). Η συνάρτηση αυτή δέχεται επίσης κανονικοποιημένη τιμή, αλλά πρώτα τη μετατρέπει σε διπολική μορφή μέσω της toBipolar(). Έπειτα, η απεικόνιση γίνεται σε βαθμίδες από -5 έως +5, με το 0 να αντιστοιχεί στη μεσαία θέση του ποτενσιόμετρου.

```
static int8_t quantizeBipolarTo5(float x) {
  x = clamp01(x);
  float b = toBipolar(x);                // -1..+1
  int steps = (int)lroundf(b * 5.0f);    // -5..+5
  if (steps < -5) steps = -5;
  if (steps > 5)  steps = 5;
  return (int8_t)steps;
}
```

Οι συναρτήσεις παραγωγής και μορφοποίησης του σήματος χρησιμοποιούν τις εξομαλυμένες τιμές των ποτενσιόμετρων ως συνεχείς μεταβλητές float, χωρίς να επηρεάζονται από την κβάντιση των τιμών. Με αυτόν τον τρόπο, η μεταβολή του ήχου παραμένει ομαλή, ενώ η οθόνη παρουσιάζει πιο σταθερές και ευανάγνωστες ενδείξεις.

Τέλος, στη σελίδα ORG, η πρώτη παράμετρος αντιστοιχεί στην επιλογή ηχητικού χαρακτήρα και εμφανίζεται ως όνομα ή κατάσταση, ενώ η δευτερη παράμετρος εμφανίζεται ως ποσοστό έντασης της μορφοποίησης. Στις υπόλοιπες σελίδες, οι τιμές παρουσιάζονται με αντίστοιχες μπάρες ή αριθμητικές ενδείξεις, ανάλογα με τη λειτουργία κάθε παραμέτρου.

6.9.5 Soft takeover των ποτενσιόμετρων

Η χρήση των ίδιων ποτενσιόμετρων σε διαφορετικές σελίδες δημιουργεί ένα πρακτικό πρόβλημα. Όταν ο χρήστης αλλάζει σελίδα, η φυσική θέση ενός ποτενσιόμετρού μπορεί να μη συμφωνεί με την αποθηκευμένη τιμή της νέας παραμέτρου. Αν η νέα μέτρηση εφαρμόζονταν αμέσως, θα μπορούσε να προκληθεί απότομο άλμα στην τιμή της παραμέτρου και συνεπώς ξαφνική μεταβολή στον ήχο.

Για την αποφυγή του προβλήματος χρησιμοποιείται μηχανισμός soft takeover, γνωστός και ως pickup[47]. Μετά την αλλαγή σελίδας, η νέα παράμετρος του ποτενσιόμετρου δεν ενημερώνεται μέχρι η φυσική θέση του θέση να προσεγγίσει την ήδη αποθηκευμένη τιμή της παραμέτρου. Η συνθήκη μπορεί να εκφραστεί ως:

$$x_{pot} - x_{stored} < E \tag{6.44}$$

όπου x_{pot} είναι η τρέχουσα θέση του ποτενσιόμετρου, x_{stored} η αποθηκευμένη τιμή και E το επιτρεπόμενο όριο προσέγγισης που ισούται με 0.06.

Όταν αλλάζει η ενεργή σελίδα, φορτώνονται οι αποθηκευμένες τιμές της νέας σελίδας και τα ποτενσιόμετρα τίθενται προσωρινά σε μη ενεργή κατάσταση ελέγχου. Στη συνάρτηση loadPageToPots(), φαίνεται ότι κάθε σελίδα έχει δικό της ζεύγος αποθηκευμένων τιμών.

```
static void loadPageToPots(UiPage page) {
  switch (page) {
    case PAGE_ORG:
      potCharacterSmooth = uiVals.orgSelect;
      potAmountSmooth    = uiVals.orgAmount;
      break;

    case PAGE_SND:
      potCharacterSmooth = uiVals.sndChar;
      potAmountSmooth    = uiVals.sndAmount;
      break;

    case PAGE_LFO:
      potCharacterSmooth = uiVals.lfoRate;
      potAmountSmooth    = uiVals.lfoDepth;
      break;

    case PAGE_DLY:
      potCharacterSmooth = uiVals.dlyTime;
      potAmountSmooth    = uiVals.dlyMix;
      break;
  }

  potCharacterNormalized = potCharacterSmooth;
  potAmountNormalized    = potAmountSmooth;
  pot1Latched = false;
  pot2Latched = false;
}
```

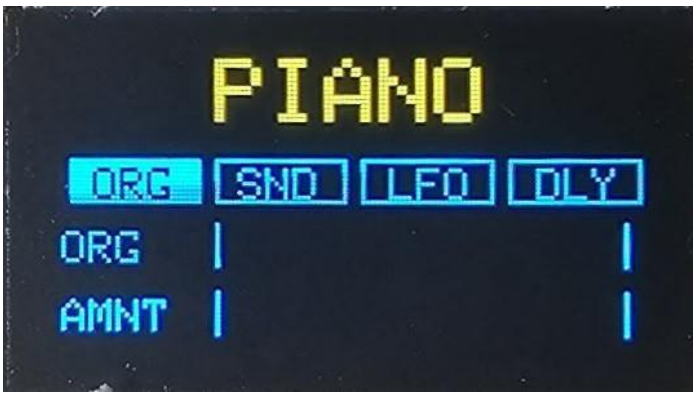
Στη συνέχεια, ο κώδικας ελέγχει αν η φυσική θέση του ποτενσιόμετρού έχει πλησιάσει αρκετά την αποθηκευμένη τιμή.

Η τεχνική αυτή βελτιώνει σημαντικά, την εργονομία του συστήματος, επειδή επιτρέπει τη χρήση δύο ποτενσιόμετρων για πολλές διαφορετικές παραμέτρους. Ο χρήστης δεν χρειάζεται να επαναφέρει χειροκίνητα τα ποτενσιόμετρα σε συγκεκριμένη θέση κάθε φορά που αλλάζει σελίδα, ενώ παράλληλα αποφεύγονται απότομες και ανεπιθύμητες μεταβολές στον ήχο.

6.9.6 Ενημέρωση της οθόνης OLED

Η οθόνη OLED χρησιμοποιείται για την εμφάνιση της τρέχουσας κατάστασης του συστήματος. Σε επίπεδο λογισμικού, η ενημέρωση της οθόνης πραγματοποιείται από τη συνάρτηση updateDisplayUI(). Η οθόνη εμφανίζει στο πάνω μέρος, τον επιλεγμένο ηχητικό χαρακτήρα ή προσωρινά την τιμή μίας παραμέτρου. Κάτω από αυτό εμφανίζονται οι τέσσερις σελίδες ρυθμίσεων ORG, SND, LFO και DLY, καθώς και δύο μπάρες ενδείξεων που αντιστοιχούν στις τιμές των ποτενσιόμετρων.

Η λογική απεικόνισης ακολουθεί την ίδια οργάνωση με τη λογική ελέγχου:



Εικόνα 6.5: Απεικόνιση σελίδας ORG

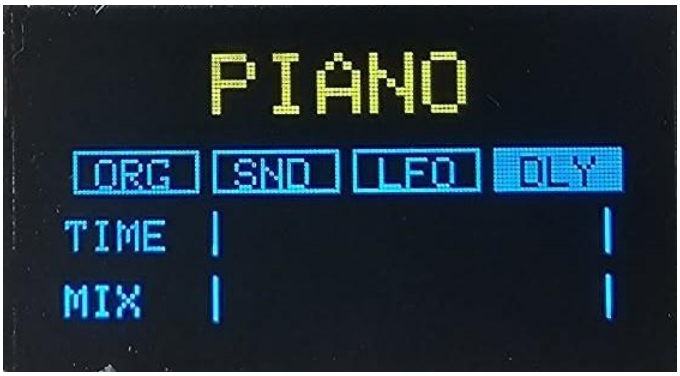
Στη σελίδα ORG εμφανίζεται το επιλεγμένο μοντέλο χροιάς και η τιμή Amount.

Εικόνα 6.6: Απεικόνιση σελίδας SND

Στη σελίδα SND εμφανίζονται Character και Amount.

Εικόνα 6.7: Απεικόνιση σελίδας LFO

Στη σελίδα LFO εμφανίζονται Rate και Depth.



Εικόνα 6.8: Απεικόνιση σελίδας DLY

Στη σελίδα DLY εμφανίζονται Time και Mix.

Για την απεικόνιση των τιμών χρησιμοποιούνται δύο διαφορετικοί τύποι μπάρας. Η μονοπολική μπάρα χρησιμοποιείται για τιμές που κυμαίνονται από 0 έως 100%, ενώ η διπολική μπάρα χρησιμοποιείται για τις μεταβλητές, όπου το κέντρο τους αντιστοιχεί στην ουδέτερη τιμή.

Όταν ο χρήστης μετακινεί κάποιο ποτενσιόμετρο, η οθόνη μπορεί να εμφανίσει προσωρινά την αριθμητική τιμή ή την περιγραφή της αντίστοιχης παραμέτρου. Μετά από μικρό χρονικό διάστημα, η οθόνη επιστρέφει στην κανονική προβολή. Επιπλέον, η οθόνη ενημερώνεται μόνο όταν έχει αλλάξει κάποια κατάσταση, για να μην γίνεται συνεχής ανανέωση των εμφανιζόμενων τιμών.

Η επιλογή αυτή μειώνει τον φόρτο του κύριου προγράμματος, καθώς η ενημέρωση της οθόνης δεν εκτελείται συνεχώς χωρίς λόγο. Αυτό είναι σημαντικό σε ένα σύστημα πραγματικού χρόνου, όπου η παραγωγή και η αποστολή των ηχητικών δειγμάτων πρέπει να παραμένει σταθερή.

6.10 Συνολική ροή λειτουργίας του κώδικα

Η συνολική λειτουργία του λογισμικού βασίζεται στη συνεργασία των επιμέρους τμημάτων που αναλύθηκαν στις προηγούμενες ενότητες. Το MATLAB χρησιμοποιείται για την προετοιμασία των ηχητικών δειγμάτων, στο αρχείο piano32.h αποθηκεύονται οι νότες σε πίνακες δεδομένων και βασικές παράμετροι, ενώ το κύριο πρόγραμμα του ESP32 αναλαμβάνει την ανάγνωση των εισόδων, την παραγωγή του ήχου και την αποστολή των δειγμάτων στον DAC.

Κατά την εκκίνηση του συστήματος εκτελείται αρχικά η συνάρτηση setup(). Σε αυτό το στάδιο αρχικοποιούνται τα βασικά περιφερειακά συστήματα, όπως η επικοινωνία SPI για τον DAC, η οθόνη OLED, οι αναλογικές είσοδοι του πληκτρολογίου και των ποτενσιόμετρων, το πλήκτρο ελέγχου και οι χρονοδιακόπτες. Παράλληλα, ο DAC οδηγείται αρχικά στη στάθμη ηρεμίας, ώστε η έξοδος να βρίσκεται στο κέντρο της 10-bit κλίμακας πριν ξεκινήσει οποιαδήποτε αναπαραγωγή.

Μετά την ολοκλήρωση της αρχικοποίησης το πρόγραμμα εισέρχεται στη συνάρτηση loop(), η οποία εκτελείται συνεχώς. Η loop() δεν είναι υπεύθυνη για την άμεση αποστολή των δειγμάτων στον DAC, καθώς η διαδικασία αυτή απαιτεί σταθερό χρονισμό και εκτελείται μέσω χρονικής διακοπής. Αντίθετα, αναλαμβάνει τον έλεγχο των περιφερειακών του συστήματος, την ανάγνωση των ενεργειών του χρήστη, την επεξεργασία των εισόδων και την ενημέρωση της OLED όταν απαιτείται.

Όταν αναγνωριστεί ότι έχει πατηθεί κάποιο πλήκτρο του πληκτρολογίου, η τιμή του ADC αντιστοιχίζεται στην κατάλληλη νότα, χρησιμοποιώντας τα όρια που έχουν αποθηκευτεί στο αρχείο piano32.h. Στη συνέχεια ενεργοποιείται η noteOn(), η οποία συνδέει τη δομή currentVoice με τον αντίστοιχο πίνακα δεδομένων, μηδενίζει τους δείκτες αναπαραγωγής και αποθηκεύει τις τρέχουσες

παραμέτρους της διεπαφής χρήστη για τη νέα νότα. Με αυτόν τον τρόπο η ενεργή φωνή είναι έτοιμη να ξεκινήσει την παραγωγή δειγμάτων.

Η επιλογή του ηχητικού χαρακτήρα και των παραμέτρων ελέγχου γίνεται μέσω της διεπαφής χρήστη. Οι αντίστοιχες παράμετροι αποθηκεύονται στη φωνή κατά την ενεργοποίηση της νότας, ώστε η αναπαραγωγή της να παραμένει σταθερή ακόμη και αν ο χρήστης αλλάξει σελίδα ή μετακινήσει κάποιο ποτενσιόμετρο στη συνέχεια.

Η παραγωγή του ηχητικού σήματος πραγματοποιείται μέσω της συνάρτησης currentVoice.update(). Για κάθε νέο δείγμα ανακτώνται τα κατάλληλα δεδομένα της ενεργής νότας και πραγματοποιείται κυβική παρεμβολή x4 για την ανακατασκευή των ενδιάμεσων δειγμάτων. Στη συνέχεια η συνάρτηση, διαμορφώνεται η χροιά ανάλογα με το επιλεγμένο ηχητικό μοντέλο, εφαρμόζονται τα διαθέσιμα ηχητικά εφέ και στο τέλος προσαρμόζεται το πλάτος του σήματος μέσω της applyADSR(). Το αποτέλεσμα είναι μία τιμή κατάλληλη για τον DAC, η οποία αποθηκεύεται στον audio buffer μέσω της συνάρτησης fillAudioBuffer().

Η χρήση του audio buffer επιτρέπει τον διαχωρισμό της παραγωγής των δειγμάτων από την διαδικασία αποστολή του. Η παραγωγή του δείγματος, η οποία είναι πιο σύνθετη υπολογιστικά, πραγματοποιείται στο κύριο πρόγραμμα, ενώ η αποστολή στον DAC γίνεται με σταθερό χρονισμό από τη συνάρτηση sendToDAC(). Η διαδικασία αυτή εκτελείται από την διακοπή του χρονοδιακόπτη και διαβάζει κάθε φορά την επόμενη διαθέσιμη τιμή από τον buffer. Στη συνέχεια σχηματίζεται τη 16-bit λέξη ελέγχου του MCP4911 και αποστέλλεται μέσω SPI.

Η αρχιτεκτονική αυτή επιτρέπει στο σύστημα να λειτουργεί με σταθερό ρυθμό αναπαραγωγής, διατηρώντας ταυτόχρονα τη δυνατότητα ελέγχου παραμέτρων από τον χρήστη. Οι χρονικά κρίσιμες λειτουργίες, όπως η αποστολή δειγμάτων στον DAC, εκτελούνται μέσω χρονισμένης διακοπής, ενώ οι υπόλοιπες λειτουργίες ελέγχου, παραγωγής δειγμάτων και ενημέρωσης της διεπαφής εκτελούνται στη βασική ροή του προγράμματος. Με αυτόν τον διαχωρισμό μειώνεται ο φόρτος μέσα στη διακοπή ISR και βελτιώνεται η σταθερότητα της αναπαραγωγής.

Συνολικά, ο κώδικας του συστήματος συνδυάζει προεπεξεργασμένους πίνακες δεδομένων, ανακατασκευή των δειγμάτων, παραγωγή δείγματος σε πραγματικό χρόνο, μορφοποίηση χροιάς, εφαρμογή ηχητικών εφέ και αλληλεπίδραση με τον χρήστη. Με αυτόν τον τρόπο ολοκληρώνεται η λειτουργική σύνδεση του λογισμικού με το πρακτικό υλικό που παρουσιάστηκε στα προηγούμενα κεφάλαια, οδηγώντας από την επιλογή μίας νότας μέχρι την τελική ακουστική αναπαραγωγή της από το ηχείο.

6.11 Επίλογος

Στο κεφάλαιο αυτό παρουσιάστηκε η λογισμική υλοποίηση του συστήματος, από την προετοιμασία και οργάνωση των ηχητικών δεδομένων έως την παραγωγή και επεξεργασία των δειγμάτων σε πραγματικό χρόνο. Ιδιαίτερη έμφαση δόθηκε στον χρονισμό της αναπαραγωγής, στη διαχείριση της ενεργής νότας, στη διαμόρφωση διαφορετικών ηχητικών χαρακτήρων, στην εφαρμογή των εφέ και στη λειτουργία της διεπαφής. Η οργάνωση των επιμέρους λειτουργιών σε διακριτά στάδια επέτρεψε τον σαφή διαχωρισμό της παραγωγής ήχου από τις λειτουργίες ελέγχου και αναπαράστασης, ολοκληρώνοντας έτσι το λογισμικό μέρος του ψηφιακού synthesizer.

Κεφάλαιο 7ο: Συμπεράσματα και προτάσεις υλοποίησης

7.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζονται τα βασικά συμπεράσματα που προέκυψαν από την ερεύνα την ανάπτυξη και την λειτουργική αξιολόγηση του ψηφιακού synthesizer. Η αξιολόγηση επικεντρώνεται στη γενική συμπεριφορά της κατασκευής, στην ποιότητα του παραγόμενου ηχητικού σήματος και την αποτελεσματικότητα των επιμέρους μοντέλων επεξεργασίας που υλοποιήθηκαν. Παράλληλα, εξετάζονται οι περιορισμοί που διαπιστώθηκαν κατά την δημιουργία του πρωτότυπου και διατυπώνονται προτάσεις για την μελλοντική εξέλιξη και βελτίωση του.

7.2 Αξιολόγηση και συμπεράσματα της υλοποίησης

Η ολοκλήρωση της κατασκευής και του λογισμικού οδήγησε σε ένα λειτουργικό πρωτότυπο ψηφιακού synthesizer, ικανό να αναπαράγει τις διαθέσιμες νότες και να διαμορφώσει τον ηχητικό τους χαρακτήρα σε πραγματικό χρόνο. Κατά τη συνολική δοκιμή του συστήματος δεν παρατηρήθηκαν δυσλειτουργίες που να εμποδίζουν την χρήση του, ενώ τα επιμέρους ηλεκτρονικά και λογισμικά τμήματα συνεργάζονται με ικανοποιητική σταθερότητα.

Η ποιότητα του παραγόμενου ήχου κρίνεται ικανοποιητική, καθώς οι νότες μεταξύ τους διατηρούν ξεκάθαρη ακουστική διαφοροποίηση. Η ανάλυση των 10-bit του DAC και ο μικρός θόρυβος περιορίζουν σε έναν βαθμό την καθαρότητα του τελικού σήματος. Ωστόσο, η αύξηση της συχνότητας δειγματοληψίας από τα 16kHz στα 32kHz επέφερε αισθητή βελτίωση στην ποιότητα αναπαραγωγής.

Θετικά αξιολογείται η λειτουργία διαμόρφωσης διαφορετικών ηχητικών χαρακτήρων από το ίδιο ηχητικό υλικό. Οι χροιές Electric Piano, Metal και Retro παρουσίασαν την εντονότερη ακουστική διαφοροποίηση, ενώ οι Air και Hit είχαν περιορισμένη επίδραση στον αρχικό χαρακτήρα των νοτών. Παράλληλα, τα εφέ Tremolo και Delay είναι ικανοποιητικά, προσθέτοντας αντίστοιχα κίνηση και αίσθηση βάθους στον παραγόμενο ήχο.

Η διεπαφή χρήστη ανταποκρίθηκε στις απαιτήσεις του σχεδιασμού, επιτρέποντας τον έλεγχο των διαθέσιμων παραμέτρων με περιορισμένο αριθμό φυσικών χειριστηρίων. Συνολικά, η υλοποίηση του συστήματος επιβεβαίωσε τη δυνατότητα ανάπτυξης ενός αυτόνομου ψηφιακού synthesizer με βάση έναν μικροελεγκτή χαμηλής ισχύος.

7.3 Περιορισμοί της υλοποίησης

Παρά την ικανοποιητική λειτουργία του τελικού συστήματος, κατά τις δοκιμές διαπιστώθηκαν ορισμένοι περιορισμοί που σχετίζονται τόσο με το υλικό όσο και με την υλοποίηση του λογισμικού. Σε περιπτώσεις γρήγορης εναλλαγής μεταξύ διαδοχικών πλήκτρων μπορεί να εμφανιστεί στιγμιαία παραμόρφωση του ηχητικού σήματος, ενώ σε ορισμένες περιπτώσεις γίνεται αντιληπτή μικρή καθυστέρηση κατά τη μετάβαση από μία νότα στην επόμενη.

Ένας επιπλέον περιορισμός αφορά την ποιότητα του παραγόμενου ήχου. Η ανάλυση του DAC περιορίζει τον αριθμό των διαθέσιμων σταθμών κβάντισης, ενώ στην τελική κατασκευή παραμένει ένας μικρός βαθμός θορύβου. Παράλληλα, η κατασκευή του πληκτρολογίου μέσω μίας κοινής αναλογικής εισόδου, δυσκολεύει την υλοποίηση πολυφωνίας.

Κατά την ανάπτυξη του λογισμικού, εξετάστηκε επίσης η δημιουργία του ήχου από μία περίοδο του αρχικού σήματος και την διαμόρφωση του τελικού ήχου μέσω πρόσθετων ταλαντωτών και τεχνικών σύνθεσης. Η συγκεκριμένη πειραματική προσέγγιση αύξησε σημαντικά τις απαιτήσεις επεξεργασίας σε

πραγματικό χρόνο και δεν κατέστη δυνατό να ενσωματωθεί με σταθερότητα στην τελική έκδοση. Για τον λόγο αυτό επιλέχθηκε η sample-based προσέγγιση, η οποία παρουσίασε σταθερή συμπεριφορά.

7.4 Προτάσεις βελτίωσης και μελλοντικής εξέλιξης

1. **Υποστήριξη πολυφωνικής αναπαραγωγής:** Μία από τις σημαντικότερες βελτιώσεις αφορά τον επανασχεδιασμό του πληκτρολογίου, ώστε να είναι δυνατή η ταυτόχρονη ανάγνωση περισσότερων πλήκτρων. Η αλλαγή αυτή θα απαιτούσε τη διαχείριση πολλαπλών ενεργών φωνών, καθώς και την ταυτόχρονη παραγωγή και αναπαραγωγή των αντίστοιχων ηχητικών δειγμάτων.
2. **Αύξηση ανάλυσης του DAC:** Η αντικατάσταση του MCP4911 από έναν DAC μεγαλύτερης ανάλυσης, θα μπορούσε να βελτιώσει την ποιότητα του αναλογικού σήματος, αυξάνοντας τον αριθμό των διαθέσιμων σταθμών εξόδου και περιορίζοντας την επίδραση της κβάντισης.
3. **Κατασκευή PCB και νέου κελύφους:** Η μεταφορά του κυκλώματος από διάτρητες πλακέτες σε μία κατάλληλα σχεδιασμένη PCB θα επέτρεπε καλύτερη δρομολόγηση των σημάτων, της τροφοδοσίας και των γειώσεων, συμβάλλοντας στον περιορισμό του ηλεκτρικού θορύβου. Παράλληλα, ένα κέλυφος κατασκευασμένο με τρισδιάστατη εκτύπωση θα μπορούσε να προσφέρει μεγαλύτερη μηχανική αντοχή, προστασία και καλύτερη συνολική εργονομία.
4. **Προσθήκη ελέγχου έντασης:** Η προσθήκη ενός επιπλέον ποτενσιόμετρου για τη ρύθμιση της συνολικής έντασης εξόδου θα παρείχε μεγαλύτερη ευελιξία στον χρήστη, καθώς στην υπάρχουσα κατασκευή η στάθμη έντασης παραμένει ουσιαστικά προκαθορισμένη.
5. **Αλλαγή μεθόδου παραγωγής ήχου:** Σε μελλοντική εξέλιξη θα μπορούσαν να επανεξεταστούν πιο σύνθετες τεχνικές ψηφιακής σύνθεσης, εφόσον χρησιμοποιηθεί αναπτυξιακή πλακέτα με μεγαλύτερη υπολογιστική ισχύ. Με αυτόν τον τρόπο το σύστημα θα μπορούσε να επεκταθεί πέρα από την υπάρχουσα sample-based προσέγγιση.

7.5 Επίλογος

Η ανάπτυξη της Πτυχιακής Εργασίας ολοκληρώθηκε με την κατασκευή ενός λειτουργικού ψηφιακού synthesizer, το οποίο συνδυάζει ηλεκτρονικό σχεδιασμό, ψηφιακή επεξεργασία και διαμόρφωση ήχου. Μέσα από τη διαδικασία ανάπτυξης αναδείχθηκαν τόσο οι δυνατότητες όσο και οι περιορισμοί που προκύπτουν κατά την υλοποίηση ενός ψηφιακού μουσικού οργάνου. Το τελικό πρωτότυπο ανταποκρίθηκε στους βασικούς στόχους που τέθηκαν, παρέχοντας αναπαραγωγή μουσικών νοτών, διαμόρφωση της χροιάς, εφαρμογή ηχητικών εφέ και άμεσο έλεγχο των παραμέτρων από τον χρήστη. Παρά τους επιμέρους περιορισμούς, η κατασκευή αποτελεί μία ολοκληρωμένη βάση πάνω στην οποία μπορούν να εφαρμοστούν βελτιώσεις τόσο στην ποιότητα του ήχου όσο και στις δυνατότητες του συστήματος.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Wikipedia, “Συνθεσάιζερ”, https://en.wikipedia.org/wiki/Synthesizer

[2]Metehan Malakçı & Berkay Erdoğan, ANALOG SYNTHESIZER, , https://www.researchgate.net/profile/Metehan-Malakci/publication/353578757_Analog_Synthesizer/links/6103f8c40c2bfa282a0e4090/Analog-Synthesizer.pdf

[3] Mastered Blog, "Αρθρωτή σύνθεση: Ο πλήρης οδηγός για αρχάριους", Tyler Connaghan , https://emastered.com/el/blog/modular-synthesis

[4] Wikipedia, " Yamaha DX7", https://en.wikipedia.org/wiki/Yamaha_DX7

[5] WOLF design review, "Yamaha DX7 Synthesizer (1983)”, https://review.wolfarchitects.design/yamaha-dx7-synthesizer-1983/

[6] Λωτης Θ. & Διαμαντόπουλος Τ , *Μουσική Πληροφορική & Μουσική με Υπολογιστές*, https://www.calameo.com/read/003094022486552c54e37

[7] ResearchGate. net, "Additive Synthesis", https://www.researchgate.net/figure/An-example-of-the-basic-principle-of-the-additive-synthesis-process-3-simple-tones_fig5_327494789

[8] Roads, Curtis, *The Computer Music Tutorial*, MIT Press, 1995

[9] Prof. Dmitri Tymoczko, "What Makes Music Sound Good?", 2010

[10] Dr. Rachel Hall, "Sounding Number", Intervals and Pitch

[11] Mike Cook, *Arduino Music and Audio Projects*, Apress, 2015

[12] Espressif, “ESP32 Series, Version 5.3 datasheet”, https://documentation.espressif.com/esp32_datasheet_en.pdf

[13] Microchip, “8/10/12-Bit Voltage Output Digital-to-Analog Converter

with SPI Interface”**,** MCP4911 datasheet

[14] Wikipedia, "Pulse-code modulation", https://en.wikipedia.org/wiki/Pulse-code_modulation

[15] Paul Bourke, "Interpolation methods", December 1999, https://www.researchgate.net/profile/Paul-Bourke-2/publication/246924712_Nearest_neighbour_weighted_interpolation/links/0c96052c36d58b3a9e000000/Nearest-neighbour-weighted-interpolation.pdf

[16] Tamara Smyth, "Music 270a: Fundamentals of Digital Audio and Discrete-Time Signals", https://musicweb.ucsd.edu/~trsmyth/digitalAudio/Sampling.html

[17] Wikipedia, "ESP32", https://en.wikipedia.org/wiki/ESP32

[18] Espressif, “ESP32-WROOM-D32 dev board image”, https://forum.arduino.cc/t/esp32-wroom-d32-dev-board-uart-flash-etc-question/1267145

[19] Joseph Wu, A Basic Guide to I2C, SBAA565 – NOVEMBER 2022, https://www.ti.com/lit/an/sbaa565/sbaa565.pdf?ts=1788590318682

[20] ST, LM217, LM317 datasheet

[21] Guo Jian&Zhu Jie&Zhou Li, "Design of a DC Linear Power Supply with Adjustable Voltage", 2012 International Conference on Mechanical and Electronics Engineering, IERI Procedia 3 73-80

[22] The Sierra Circuits Team, "What is the use of a decoupling capasitor", https://www.protoexpress.com/blog/decoupling-capacitor-use/

[23] Walt Kester,James Bryant, Mike Byrne, "Grounding Data Converters and Solving the Mystery of

AGND and DGND", https://www.analog.com/media/en/training-seminars/tutorials/MT-031.pdf

[24] Hank Zumbahlen, "Staying Well Grounded", https://www.analog.com/en/resources/analog-dialogue/articles/staying-well-grounded.html

[25] Wikipedia, "Ressistor ladder", https://en.wikipedia.org/wiki/Resistor_ladder

[26] Piyu Dhaker, "Intrduction to SPI Interface", https://www.analog.com/en/resources/analog-dialogue/articles/introduction-to-spi-interface.html

[27] Eric Gaalaas, "Class D Audio Amplifiers: What, Why, and How", https://www.analog.com/en/resources/analog-dialogue/articles/class-d-audio-amplifiers.html

[28] Diodes Incorporated, “2.5W FILTERLESS CLASS-D MONO AUDIO AMPLIFIER”, PAM8302A datasheet, https://cdn-shop.adafruit.com/datasheets/PAM8302A.pdf

[29] Adafruit PAM8302, “Mono 2.5W Class D Audio Amplifier", https://learn.adafruit.com/adafruit-pam8302-mono-2-5w-class-d-audio-amplifier/overview

[30] SOLOMON SYSTECH, “128 x 64 Dot Matrix OLED/PLED Segment/Common Driver with Controller”, SSD1306 datasheet”, https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf

[31] Makerselectronics.com, "Push Button 6 Pins Momentary 7mm x 7mm" image, https://makerselectronics.com/product/push-button-6-pins-momentary-7mm-x/

[32] Wikipedia, "Low-pass filter", https://en.wikipedia.org/wiki/Low-pass_filter

[33] Alan Rich, AN-347, Shielding and Guarding How to Exclude Interference-Type Noise What to Do and Why to Do Ilt, https://www.analog.com/media/en/technical-documentation/application-notes/41727248AN_347.pdf

[34] Mathworks, "Resample uniform or nonuniform data to new fixed rate", https://www.mathworks.com/help/signal/ref/resample.html#d127e250793

[35] Espressif, "Arduino-ESP32 ADC API", https://docs.espressif.com/projects/arduino-esp32/en/latest/api/adc.html

[36] Espressif, "Timer", https://docs.espressif.com/projects/arduino-esp32/en/latest/api/timer.html

[37] Christopher Twigg, Catmull-Rom splines, March 11, 2003

[38] Wikipedia, "SmoothStep", https://en.wikipedia.org/wiki/Smoothstep

[39] Unity, "Smoothstep", https://docs.unity3d.com/Packages/com.unity.visualeffectgraph@12.0/manual/Operator-Smoothstep.html

[40] MARC LE BRUN, Digital Waveshaping Synthesis, Center for Computer Research in Music and Acoustics,

Stanford University, Stanford, CA 94305, https://www.researchgate.net/profile/Marc-Lebrun-2/publication/405734209_Digital_WaveshapingSynthesis/links/6a1f1118cda78712cc3a3d43/Digital-WaveshapingSynthesis.pdf

[41] DSPRelated.com, "One-Pole", https://www.dsprelated.com/freebooks/filters/One_Pole.html

[42] JULIUS O. SMITH III, *PHYSICAL AUDIO SIGNAL PROCESSING*, Nonlinear Distortion, https://www.dsprelated.com/freebooks/pasp/Nonlinear_Distortion.html

[43] Rhodes, “Rhodes V8 Manual v3”, https://download.rhodesmusic.com/assets/RHODES-V8PRO/Rhodes%20V8%20Manual%20v3.pdf

[44] JULIUS O. SMITH III, Physical Modeling Synthesis, The Computer Music Journal, vol. 20, no. 2, , pp. 44–56, Summer 1996, https://scispace.com/pdf/physical-modeling-synthesis-update-1e4o3kdlt6.pdf

[45] Kevin Karplus and Alex Strong, Digital Synthesis of Plucked-String and Drum Timbres, Computer Music Journal, Vol. 7, No. 2 (Summer, 1983), pp. 43-55, https://users.soe.ucsc.edu/~karplus/papers/digitar.pdf

[46] Miller Puckette, *The Theory and Technique of Electronic Music*, World Scientific Publishing, 2006

[47] Native Instruments, "Automation and MIDI Control", https://docs.native-instruments.com/ni-tech-manuals/komplete-kontrol-manual/en/automation-and-midi-control

ΚΩΔΙΚΑΣ ESP32

```
// Βιβλιοθήκες και δεδομένα ηχητικών δειγμάτων
#include "SPI.h"
#include "piano32.h" //Πίνακες ηχητικών δειγμάτων και βοηθητικοί πίνακες νοτών
#include "Arduino.h"
#include "stdint.h"
#include "driver/gpio.h"
#include "math.h"
#include "esp32-hal-timer.h"
#include "Wire.h"
#include "Adafruit_GFX.h"
#include "Adafruit_SSD1306.h"
#include "string.h"
#include "soc/gpio_struct.h"
#include "soc/gpio_reg.h"

// Ακροδέκτες επικοινωνίας SPI με τον DAC
#define CS_PIN 17
#define MOSI_PIN 23
#define SCK_PIN 18

// Έλεγχος CS με απευθείας χρήση καταχωρητών GPIO
#define CS_LOW() do {REG_WRITE(GPIO_OUT_W1TC_REG, (1UL << CS_PIN)); }  while(0)
#define CS_HIGH() do {REG_WRITE(GPIO_OUT_W1TS_REG, (1UL << CS_PIN)); }  while(0)

#define INVERT_POT_CHARACTER true
#define INVERT_POT_AMOUNT    true

//Διαστάσεις και αρχικοποίηση της οθόνης
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET    -1
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);


#define SAMPLE_RATE 32000.0f  //Ρυθμός δειγματοληψίας fs
#define TWO_PI 6.283185307f
#define PI 3.14159265f

#define AUDIO_BUFFER_SIZE 256 //  Χωρητικότητα του buffer
#define PM_STRING_BUF_LEN 256 //  Χωρητικότητα του buffer καθυστέρησης για το
μοντέλο string

//Βοηθητικές μεταβλητές για την διαδικασία fillAudioBuffer()
volatile uint16_t audioBuffer[AUDIO_BUFFER_SIZE];
volatile uint16_t audioReadIndex = 0;
volatile uint16_t audioWriteIndex = 0;
volatile uint16_t audioCount = 0;

volatile int maxInterp = 4; //Αριθμός βημάτων παρεμβολής

//Ορισμός των GPIO
#define SD_PIN 14 //Ενισχυτής
#define HOLD_SWITCH_PIN 27 // tactile switch
const int keybPin = 35;  // πληκτρολόγιο
const int POT_CHARACTER = 34;  // Ποτενσιόμετρο
const int POT_AMOUNT = 33;  // Ποτενσιόμετρο
```

```
volatile bool keyCheckFlag = false;// Σημαία ελέγχου ADC πληκτρολογίου

//Μεταβλητές για την χρήση των ποτενσιομέτρων
volatile float potCharacterNormalized = 0.5f; // 0..1, αρχικό στάδιο ΚΕΝΤΡΟ
volatile float potAmountNormalized    = 0.0f; // 0..1, αρχικό στάδιο ΜΗΔΕΝ

volatile float potCharacterSmooth = 0.5f;
volatile float potAmountSmooth    = 0.0f;
// οι παρακάτω μεταβλητές χρησιμοποιούνται για να κλειδώνει το pot στην
αποθηκευμένη τιμή και να μην κάνει jump
volatile bool pot1Latched = true;
volatile bool pot2Latched = true;
const float POT_PICKUP_THRESHOLD = 0.06f;


portMUX_TYPE timerMux = portMUX_INITIALIZER_UNLOCKED; // mutex, προστασία της
κοινής πρόσβασης στον buffer και στους δείκτες του
portMUX_TYPE timerMux2 = portMUX_INITIALIZER_UNLOCKED;// προστασία κατα την
ενημέρωση των ποτενσιομέτρων

hw_timer_t *timer = NULL; //Timer αποστολής δειγμάτων στον DAC
hw_timer_t *keyTimer = NULL; //Timer  ελέγχου πληκτρολογίου

// Σελίδες ρυθμίσεων της διεπαφής χρήστη
enum UiPage : uint8_t {
  PAGE_ORG   = 0,
  PAGE_SND   = 1,
  PAGE_LFO   = 2,
  PAGE_DLY   = 3
};

// Διαθέσιμα μοντέλα χροιάς
enum Instrument {

    INST_PIANO  = 0,
    INST_EPIANO = 1,
    INST_AIR    = 2,
    INST_STRING = 3,
    INST_HIT    = 4,
    INST_METAL  = 5,
    INST_RETRO  = 6

};

//επιλογή μοντέλου επεξεργασίας
volatile Instrument currentInstrument = INST_PIANO;   // αυτό που παίζει τώρα
volatile Instrument selectedInstrument = INST_PIANO;  // αυτό που έχει επιλεγεί
από το ORG ποτεσνιόμετρο σε real-time
volatile UiPage currentPage = PAGE_ORG;

void updateDisplayUI(Instrument inst, UiPage page, uint8_t charPct, uint8_t
amtPct, bool showValue, const char* valueText);
static const char* instrumentName(Instrument inst);

struct UiValues {
  float orgSelect;
  float orgAmount;
```

```
  float sndChar;
  float sndAmount;

  float lfoRate;
  float lfoDepth;

  float dlyTime;
  float dlyMix;
};

volatile UiValues uiVals = {
  0.0f, 0.0f,   // ORG: instrument select, amount

  0.5f, 0.0f,   // SND: character, amount

  0.0f, 0.0f,   // LFO: rate, depth

  0.0f, 0.0f    // DLY: time, mix
};


// Περιορίζει την τιμή στο διάστημα 0..1
static inline float clamp01(float x) {
  if (x < 0.0f) return 0.0f;
  if (x > 1.0f) return 1.0f;
  return x;
}

// Αντιστοιχίζει γραμμικά μια τιμή από το διάστημα 0..1 στο διάστημα a..b
static inline float map01(float x, float a, float b) {
  x = clamp01(x);
  return a + x * (b - a);
}

// Μετατρέπει την τιμή του ποτενσιόμετρου ORG σε μία από τις επτά επιλογές
ηχοχρώματος
static Instrument instrumentFromOrgPot(float x) {
  x = clamp01(x);

  int idx = (int)floorf(x * 7.0f);

  if (idx < 0) idx = 0;
  if (idx > 6) idx = 6;// Στο x = 1 διατηρεί την τελευταία έγκυρη επιλογή

  return (Instrument)idx;
}
// Εμφάνιση των επιλεγμένων μοντέλων στην οθόνη
static const char* instrumentName(Instrument inst) {
  switch (inst) {
    case INST_PIANO:  return "PIANO";
    case INST_EPIANO: return "EPIANO";
    case INST_AIR:    return "AIR";
    case INST_STRING: return "STRING";
    case INST_HIT:    return "HIT";
    case INST_METAL:  return "METAL";
    case INST_RETRO:  return "RETRO";
    default:          return "----";
  }
}
```

```
// Μετατρέπει το διάστημα 0..1 σε -1..+1, με μηδενική τιμή στο κέντρο
static inline float toBipolar(float x) {
  x = clamp01(x);
  return (x - 0.5f) * 2.0f;
}

// Αντιστοιχίζει την τιμή του χειριστηρίου σε συχνότητα LFO από 0.3 έως 8 Hz
static inline float mapLfoRate(float x) {
  x = clamp01(x);
  return 0.3f + x * 7.7f;
}

// Εφαρμόζει ενίσχυση και ομαλό κορεσμό
static inline float softSaturate(float x, float amount) {
  amount = clamp01(amount);   // 0..1
  if (amount < 0.001f) return x;

  // όσο μεγαλώνει το amount, τόσο πιο έντονο drive
  float drive = 1.0f + 1.4f * amount;

  float y = x * drive;

// Ομαλή συμπίεση του πλάτους
  y = y / (1.0f + fabsf(y) / 750.0f);
  return y;
}

// Φορτώνει τις αποθηκευμένες τιμές της σελίδας στις μεταβλητές των χειριστηρίων
static void loadPageToPots(UiPage page) {
  switch (page) {
    case PAGE_ORG:
      potCharacterSmooth = uiVals.orgSelect;
      potAmountSmooth    = uiVals.orgAmount;
      break;

    case PAGE_SND:
      potCharacterSmooth = uiVals.sndChar;
      potAmountSmooth    = uiVals.sndAmount;
      break;

    case PAGE_LFO:
      potCharacterSmooth = uiVals.lfoRate;
      potAmountSmooth    = uiVals.lfoDepth;
      break;

    case PAGE_DLY:
      potCharacterSmooth = uiVals.dlyTime;
      potAmountSmooth    = uiVals.dlyMix;
      break;
  }

  potCharacterNormalized = potCharacterSmooth;
  potAmountNormalized    = potAmountSmooth;
  pot1Latched = false;
  pot2Latched = false;
}

// Ενημερώνει τις δύο αποθηκευμένες παραμέτρους της συγκεκριμένης σελίδας
```

```
static void storePotsToPage(UiPage page, float p1, float p2) {
  switch (page) {
     case PAGE_ORG:
      uiVals.orgSelect = p1;
      uiVals.orgAmount = p2;
      break;

    case PAGE_SND:
      uiVals.sndChar   = p1;
      uiVals.sndAmount = p2;
      break;

    case PAGE_LFO:
      uiVals.lfoRate   = p1;
      uiVals.lfoDepth  = p2;
      break;

    case PAGE_DLY:
      uiVals.dlyTime   = p1;
      uiVals.dlyMix    = p2;
      break;
  }
}

// Επιστρέφει τις αποθηκευμένες παραμέτρους της σελίδας μέσω των p1 και p2
static void getStoredPageValues(UiPage page, float &p1, float &p2) {
  switch (page) {
     case PAGE_ORG:
      p1 = uiVals.orgSelect;
      p2 = uiVals.orgAmount;
      break;

    case PAGE_SND:
      p1 = uiVals.sndChar;
      p2 = uiVals.sndAmount;
      break;

    case PAGE_LFO:
      p1 = uiVals.lfoRate;
      p2 = uiVals.lfoDepth;
      break;

    case PAGE_DLY:
      p1 = uiVals.dlyTime;
      p2 = uiVals.dlyMix;
      break;
  }
}
// Επαναφέρει τις ρυθμίσεις στο αρχικό PIANO και επιλέγεται η σελίδα ORG
static void resetSoundToRawPiano() {
  currentInstrument = INST_PIANO;

  // ORG
  uiVals.orgSelect = 0.0f;  // Piano
  uiVals.orgAmount = 0.0f;

  // SND
  uiVals.sndChar   = 0.5f;
  uiVals.sndAmount = 0.0f;
```

```
  // LFO
  uiVals.lfoRate  = 0.0f;
  uiVals.lfoDepth = 0.0f;

  // DLY
  uiVals.dlyTime = 0.0f;
  uiVals.dlyMix  = 0.0f;

  currentPage = PAGE_ORG;
  selectedInstrument = INST_PIANO;
  loadPageToPots(currentPage);
}

//Ελέγχει αν η θέση του ποτεσνιομέτρου βρίσκεται εντός του ορίου με βάση την
αποθηκευμένη τιμή
static inline bool isPotNearStored(float potNow, float storedValue) {
  return fabsf(potNow - storedValue) <= POT_PICKUP_THRESHOLD;
}
// Μετατρέπει την τιμή 0..1 σε ακέραιη ένδειξη 0..10
static uint8_t quantizeTo10(float x) {
  x = clamp01(x);
  int steps = (int)lroundf(x * 10.0f);   // 0..10
  if (steps < 0) steps = 0;
  if (steps > 10) steps = 10;
  return (uint8_t)steps;
}
// Μετατρέπει την τιμή 0..1 σε ακέραιη ένδειξη -5..+5
static int8_t quantizeBipolarTo5(float x) {
  x = clamp01(x);
  float b = toBipolar(x);                // -1..+1
  int steps = (int)lroundf(b * 5.0f);    // -5..+5
  if (steps < -5) steps = -5;
  if (steps > 5)  steps = 5;
  return (int8_t)steps;
}
//περιορίζει τις τιμές στα άκρα των ποτενσιομέτρων
static inline float calibratePot(float x, float inMin, float inMax) {
  if (x <= inMin) return 0.0f;
  if (x >= inMax) return 1.0f;
  return (x - inMin) / (inMax - inMin);
}
//Αντιστρέφει τη φορά μεταβολής της τιμής του ποτενσιόμετρου
static inline float invertPot(float x, bool invert) {
  x = clamp01(x);
  if (invert) {
    return 1.0f - x;
  }
  return x;
}

static inline bool instrumentUsesUiControls(Instrument inst) {
  return true;
}

// Παράγει τριγωνική κυματομορφή στο διάστημα -1..+1 από τη φάση του LFO
inline float triLFO(uint32_t phase) {

  uint32_t x = phase >> 16;
```

```
  uint32_t t = (x & 0x8000) ? (0xFFFF - x) : x; // Αναδιπλώνει το δεύτερο μισό του
κύκλου για να σχηματιστεί το τρίγωνο

  return ((float)t / 32767.0f) * 2.0f - 1.0f;
}

//κυβική interpolation
float cubicInterpolate(float y0, float y1, float y2, float y3, float mu) {
    float a0, a1, a2, a3, mu2;
        // Συντελεστές του κυβικού πολυωνύμου
        mu2 = mu * mu;
        a0 = -0.5f * y0 + 1.5f * y1 - 1.5f * y2 + 0.5f * y3;
        a1 = y0 - 2.5f * y1 + 2.0f * y2 - 0.5f * y3;
        a2 = -0.5f * y0 + 0.5f * y2;
        a3 = y1;


    return a0 * mu * mu2 + a1 * mu2 + a2 * mu + a3;
}


struct Voice {

  const uint16_t* data = nullptr; // Δείκτης στον πίνακα δειγμάτων της νότας
  int sampleIndex = 0;  // Τρέχουσα θέση στον πίνακα
  int interpStep = 0; //Τρέχον βήμα παρεμβολής
  //float shaped = 512.0f;            // Τιμή δείγματος προς DAC
  int note = -1; // Δείκτης νότας στο πληκτρολόγιο

  Instrument voiceInstrument = INST_PIANO;

  float voiceOrgAmount = 0.0f; // ORG: Ποσότητα επεξεργασίας του επιλεγμένου
ηχοχρώματος
  // SND: χαρακτήρας και ποσότητα διαμόρφωσης
  float voiceSndChar   = 0.5f;
  float voiceSndAmount = 0.0f;
  // LFO: ρυθμός και βάθος tremolo
  float voiceLfoRate  = 0.0f;
  float voiceLfoDepth = 0.0f;
  // DLY: χρόνος και ποσότητα καθυστερημένου σήματος
  float voiceDlyTime = 0.0f;
  float voiceDlyMix  = 0.0f;

  bool active = false; // ενεργή φωνή
  bool isReleasing = false; // ενεργοποίησης εξασθένισης

  int SampleIndexSize = wavetableSizes;

  int releaseLength = 10000;  //Διάρκεια release σε παραγόμενα δείγματα εξόδου
  int releaseCounter = 0;    // Πλήθος δειγμάτων που έχουν παραχθεί στη φάση
release

  bool keyHeld = false;
  int  endFadeLen = 2048;   //  256 ms εξασθένιση

  // Κοινή διαμόρφωση ηχοχρώματος
  float pmToneState = 0.0f;
```

```
  // -------EPIANO------
  float tinePhase = 0.0f;       // Η τρέχουσα γωνία του ημιτόνου (0 - 2*PI)
  float tinePhaseStep = 0.0f;   // Πόσο αυξάνεται η γωνία σε κάθε δείγμα
  float tineEnvelope = 0.0f;    // Η ένταση του tine (ξεκινάει δυνατά, σβήνει)
  const float tineDecay = 0.99910f; // Πολλαπλασιαστής εξασθένισης (0.9900  32kHz
~= 300ms)
  float tineIntensity = 240.0f; //  πόσο δυνατός είναι ο "metal/tine" ήχος

    int hammerCounter = 0;
    int hammerLength = 704;       // // Διάρκεια κρούσης σε δείγματα, 22 ms στα 32
kHz: 0.022 * 32000 = 704 samples
    float hammerPhase = 0.0f;
    float hammerPhaseStep = 0.0f;
    uint32_t rngEp = 1u;          // πηγή θορύβου

  // Κοινές καταστάσεις επεξεργασίας επιμέρους ηχοχρωμάτων
  float pmModelPhase1     = 0.0f;
  float pmStringResState = 0.0f;

  //---------Air------
  float pmModelEnvState = 0.0f;
  uint32_t pmModelRng = 1u;
  float pmAirNoiseState = 0.0f;

  //------String--------
  float pmStringBuf[PM_STRING_BUF_LEN];
  int pmStringWriteIndex = 0;
  int pmStringDelaySamples = 80;
  float pmStringDampState = 0.0f;

  //------Hit-------
  int pmModelCounter = 0;
  float pmModelStepHit    = 0.0f;

  //-----Metal-------
  float pmModelStepMetal1 = 0.0f;

  // -----Retro state -----
  int crushCounter = 0;           // Μετρητής για το downsampling
  float heldSample = 0.0f;        // Η τιμή που "κρατάμε" στη μνήμη

    // Ρυθμίσεις έντασης εφέ
     int downsampleFactor = 4; // Πόσο "αργό" είναι το sample rate (π.χ. 4)
     int bitStep = 32;         // Πόσο "τετραγωνισμένο" είναι το κύμα (π.χ. 32)

  // ADSR μεταβλητές
  int startFadeCounter = 0;
  int startFadeLength  = 16;
  float attackSmoothState = 512.0f;

  uint32_t tremoloPhase = 0; // συσσωρευτής φάσης του LFO

  static const int DLY_BUF_LEN = 16384;   // ~512ms στα 32kHz

  float dlyBuffer[DLY_BUF_LEN] = {0.0f};
  int   dlyWriteIndex = 0;
```

```
  float generateRawSample(int maxInterp){ //Η συνάρτηση δημιουργεί το realtime
δείγμα με cubic Interpolation
    int i = sampleIndex;
    int size = SampleIndexSize;

    float y0 = data[(i > 0) ? i - 1 : 0];
    float y1 = data[i];
    float y2 = data[(i + 1 < size) ? i + 1 : size - 1];
    float y3 = data[(i + 2 < size) ? i + 2 : size - 1];

    float frac = (float)interpStep / maxInterp;
    float interp = cubicInterpolate(y0, y1, y2, y3, frac);

    return interp - 512.0f; // Κεντράρει το σήμα στο 0
  }
  // Διαμορφώνει το ηχόχρωμα μέσω των παραμέτρων Character και Amount της σελίδας
SND
  float applyTimbreShaping(float raw) {

    float char01 = voiceSndChar;
    float amt01  = voiceSndAmount;

    if (amt01 < 0.001f) {
        return raw;
    }
    float charBipolar = toBipolar(char01);
    float lpAlpha = 0.025f + 0.075f * amt01;

    pmToneState = pmToneState + lpAlpha * (raw - pmToneState);

    float lowPart  = pmToneState;
    float highPart = raw - lowPart;

    float x = raw / 480.0f;

    if (x > 1.0f)  x = 1.0f;
    if (x < -1.0f) x = -1.0f;

    float drive = 1.0f + 2.2f * amt01;

    float shaped = x * drive;
    shaped = shaped / (1.0f + fabsf(shaped) * 0.65f);

    float harmonicPart = (shaped - x) * 480.0f;
    // Warm: περισσότερο σώμα, λιγότερες ψηλές
    float warmTarget =
        lowPart * 1.08f +
        highPart * 0.35f +
        harmonicPart * 0.20f;

    // Rich: κρατάει το piano αλλά προσθέτει αρμονικές
    float richTarget =
        raw +
        harmonicPart * 0.45f;

    // Bright: προσθέτει ψηλές και harmonic excitation
    float brightTarget =
        raw +
        highPart * 0.65f +
```

```
        harmonicPart * 0.55f;

    float target;

    if (charBipolar < 0.0f) {
        float w = -charBipolar;
        target = richTarget + w * (warmTarget - richTarget);
    } else {
        float w = charBipolar;
        target = richTarget + w * (brightTarget - richTarget);
    }

    float mixed = raw + amt01 * (target - raw); // Ανάμειξη αρχικού και
διαμορφωμένου σήματος με βάση το Amount
    mixed *= (1.0f - 0.08f * amt01);
    return mixed;

  }
  //Παρακολουθεί το πλάτος του σήματος για το μοντέλο Air
  float updateTimbreEnvelope(float x) {
    float target = fabsf(x) / 420.0f;

    if (target > 1.0f) target = 1.0f;

    if (target > pmModelEnvState) {
        pmModelEnvState = pmModelEnvState + 0.18f * (target - pmModelEnvState);
    } else {
        pmModelEnvState = pmModelEnvState + 0.0030f * (target - pmModelEnvState);
    }

    return pmModelEnvState;
  }
  //// Περιορίζει το σήμα γύρω από το μηδέν στο διάστημα -480..+480
  float limitCenteredAudio(float x) {
    if (x > 480.0f)  x = 480.0f;
    if (x < -480.0f) x = -480.0f;
    return x;
  }

  float processPiano(float raw) {
    return applyTimbreShaping(raw);
  }


  float processEPiano(float raw) {
    float epAmt = voiceOrgAmount;


    if (epAmt < 0.001f) {
        return raw;
    }

    float detail =  voiceSndAmount;

    tinePhase += tinePhaseStep;
    if (tinePhase >= TWO_PI) tinePhase -= TWO_PI;

    float tineSine = sinf(tinePhase); // Ενημέρωση φάσης και υπολογισμός της
κύριας ημιτονοειδούς συνιστώσας tine
```

```
    tineEnvelope *= tineDecay;
    if (tineEnvelope < 0.001f) tineEnvelope = 0.0f;

    float clickGain = 0.9f + (0.025f * note); // Προσαρμογή του πλάτους ανάλογα με
τον δείκτη της νότας
    float metalSound = tineSine * tineEnvelope * clickGain * tineIntensity;

    // Πρόσθετη μεταλλική συνιστώσα με πλάτος που εξαρτάται από το SND Amount
    float metal2Gain = 0.16f + 0.16f * detail;
    float metal2 = sinf(tinePhase * 2.01f) * tineEnvelope * metal2Gain *
tineIntensity;

    raw += metalSound + metal2;

    if (hammerCounter < hammerLength) {
      //τετραγωνική εξασθένηση από 1 προς 0
      float x = (float)hammerCounter / (float)hammerLength;
      float env = 1.0f - x;
      env = env * env;

      hammerPhase += hammerPhaseStep;
      if (hammerPhase >= TWO_PI) hammerPhase -= TWO_PI;

      float hammerTone = sinf(hammerPhase);

      rngEp = rngEp * 1664525u + 1013904223u;
      int n = (int)((rngEp >> 24) & 0xFF) - 128;
      float hammerNoise = (float)n * 0.10f;
      float hammerGain = 45.0f + 35.0f * detail;

      raw += (hammerTone * hammerGain + hammerNoise) * env * epAmt;

      hammerCounter++;
    }

    raw *= (1.0f + 0.22f * epAmt); // Ενίσχυση του συνολικού σήματος ανάλογα με το
ORG Amount
    if (raw > 0.0f)
        raw = raw / (1.0f + raw / 760.0f);
    else
        raw = raw / (1.0f + fabsf(raw) / 760.0f);

    return limitCenteredAudio(raw);
  }

  float processAir(float raw) {
    float modelAmt = voiceOrgAmount;
    float sample = applyTimbreShaping(raw);

    if (modelAmt < 0.001f) {
        return sample;
    }
    // έλεγχοω για τη ένταση του θορύβου
    float env = updateTimbreEnvelope(sample);

    float char01 = voiceSndChar;
    float detail = voiceSndAmount;
```

```
    float lpAlpha = 0.05f + 0.05f * modelAmt;
    pmStringResState = pmStringResState + lpAlpha * (sample - pmStringResState);

    float lowPart  = pmStringResState;
    float highPart = sample - lowPart;

    pmModelRng = pmModelRng * 1664525u + 1013904223u; // Παραγωγή ψευδοτυχαίου
θορύβου
    float n = ((float)((pmModelRng >> 24) & 0xFF) - 128.0f) / 128.0f;

    pmAirNoiseState = pmAirNoiseState + 0.15f * (n - pmAirNoiseState); //
Χαμηλοπερατό φιλτράρισμα του θορύβου

    pmModelPhase1 += 0.001f;
    if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

    float breathLfo = 0.6f + 0.4f * sinf(pmModelPhase1); // Συντελεστής έντασης
από 0.2 έως 1.0

    float breathGain = 24.0f + 22.0f * detail;
    float breath = pmAirNoiseState * breathGain * env * breathLfo;

    float highKeep = map01(char01, 0.15f, 0.65f);
    float target = lowPart + highPart * highKeep + breath; // Συνδυασμός
χαμηλοπερατής συνιστώσας, υψίσυχνου μέρους και θορύβου

    float out = sample + modelAmt * (target - sample); // Ανάμειξη με το σήμα
εισόδου σύμφωνα με το ORG Amount

    out *= (1.0f - 0.05f * modelAmt);
    return limitCenteredAudio(out);
  }

  float processString(float raw) {
    float modelAmt = voiceOrgAmount;
    float sample = applyTimbreShaping(raw);

    if (modelAmt < 0.001f) {
        return sample; // Με μηδενικό ORG Amount παραμένει μόνο η διαμόρφωση SND
    }

    float detail = voiceSndAmount;
    int readIndex = pmStringWriteIndex - pmStringDelaySamples; // Υπολογισμός
θέσης ανάγνωσης στον κυκλικό buffer
    if (readIndex < 0) readIndex += PM_STRING_BUF_LEN;

    float delayed = pmStringBuf[readIndex];

    pmStringDampState = 0.5f * delayed + 0.5f * pmStringDampState; // φιλτράρισμα
του καθυστερημένου σήματος πριν από την ανάδραση

    float feedback = 0.66f + 0.20f * detail + 0.04f * modelAmt;  // Συντελεστής
ανάδρασης με βάση το SND Amount και το ORG Amount
    if (feedback > 0.92f) feedback = 0.92f;

    // Εξαγωγή του υψίσυχνου μέρους
    float lpAlpha = 0.05f + 0.05f * modelAmt;
    pmStringResState = pmStringResState + lpAlpha * (sample - pmStringResState);
```

```
    float highPart = sample - pmStringResState;
    float excitation = highPart * (0.25f + 0.35f * detail); // Το SND Amount
ελέγχει τη στάθμη της διέγερσης
    float writeVal = excitation + pmStringDampState * feedback; // Συνδυασμός νέας
διέγερσης και φιλτραρισμένης ανάδρασης

    if (writeVal > 480.0f)  writeVal = 480.0f;
    if (writeVal < -480.0f) writeVal = -480.0f;

    pmStringBuf[pmStringWriteIndex] = writeVal;

    pmStringWriteIndex++;
    if (pmStringWriteIndex >= PM_STRING_BUF_LEN) {
        pmStringWriteIndex = 0;
    }
    // Το SND Amount ρυθμίζει τη στάθμη της καθυστερημένης συνιστώσας στην έξοδο
    float stringMix = 0.55f + 0.35f * detail;
    float out = sample + delayed * modelAmt * stringMix;

    out *= (1.0f - 0.05f * modelAmt);

    return limitCenteredAudio(out);
  }

  float processHit(float raw) {
    float modelAmt = voiceOrgAmount;
    float sample = applyTimbreShaping(raw);

    if (modelAmt < 0.001f) {
        return sample;
    }

    float detail = voiceSndAmount;
    // Διαχωρισμός σε χαμηλοπερατή συνιστώσα και υψίσυχνο υπόλοιπο
    float lpAlpha = 0.05f + 0.05f * modelAmt;
    pmStringResState = pmStringResState + lpAlpha * (sample - pmStringResState);

    float lowPart  = pmStringResState;
    float highPart = sample - lowPart;

    float hit = 0.0f;
    float hitAmt = modelAmt * 0.65f;


    int hitLength = 900 + (int)(detail * 1200.0f); // Διάρκεια κρούσης από 900 έως
2100 δείγματα
    // Παραγωγή της κρούσης μόνο κατά το αρχικό διάστημα της νότας
    if (pmModelCounter < hitLength) {
        float x = (float)pmModelCounter / (float)hitLength;
        // Κυβική εξασθένηση από  1 προς 0
        float hitEnv = 1.0f - x;
        hitEnv = hitEnv * hitEnv * hitEnv;

        float dynamicStep = pmModelStepHit * (1.0f + 3.0f * hitEnv);

        pmModelPhase1 += dynamicStep;
        if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

        float hitGain = 70.0f + 40.0f * detail;
```

```
        // Ημιτονοειδής κρούση με έλεγχο πλάτους από την hitenv και το ORG Amount
        hit = sinf(pmModelPhase1) * hitGain * hitEnv * hitAmt;

        pmModelCounter++;
    }

    float dryBody = lowPart * (1.0f - 0.25f * hitAmt) + highPart;
    float out = dryBody + hit; // Προσθήκη της κρούσης στο διαμορφωμένο σήμα

    out *= (1.0f - 0.05f * modelAmt); // Μείωση της συνολικής στάθμης έως 5%
    return limitCenteredAudio(out);
  }

  float processMetal(float raw) {
    float modelAmt = voiceOrgAmount;
    float sample = applyTimbreShaping(raw);

    if (modelAmt < 0.001f) {
        return sample;
    }

    float detail = voiceSndAmount;

    pmModelPhase1 += pmModelStepMetal1; // Ενημέρωση της φάσης του ημιτονοειδούς
διαμορφωτή
    if (pmModelPhase1 >= TWO_PI) pmModelPhase1 -= TWO_PI;

    float modulator = sinf(pmModelPhase1);
    float rmSignal = sample * modulator; // Δακτυλιοειδής διαμόρφωση με
πολλαπλασιασμό του σήματος και του διαμορφωτή
    // Ποσότητα διαμορφωμένου σήματος με βάση τα ORG Amount και SND Amount
    float metAmt = modelAmt * map01(detail, 0.35f, 0.62f);

    rmSignal *= 0.75f;
    float out = sample * (1.0f - metAmt) + rmSignal * metAmt; // Ανάμειξη του
σήματος εισόδου με τη διαμορφωμένη συνιστώσα
    out *= (1.0f - 0.05f * modelAmt);

    return limitCenteredAudio(out);
}

  float processRetro(float raw) {
    float modelAmt = voiceOrgAmount;
    float sample = applyTimbreShaping(raw);

    if (modelAmt < 0.001f) {
        return sample;
    }

    float charBipolar = toBipolar(voiceSndChar);
    float detail = voiceSndAmount;

    downsampleFactor = 8 + (int)(charBipolar * 6.0f);
    if (downsampleFactor < 2)  downsampleFactor = 2;
    if (downsampleFactor > 16) downsampleFactor = 16;

    bitStep = (int)map01(detail, 6.0f, 48.0f); // Το SND Amount καθορίζει το βήμα
κβάντισης πλάτους από 6 έως 48
    if (bitStep < 6) bitStep = 6;
```

```
    int crushedInt = (int)sample;
    crushedInt = (crushedInt / bitStep) * bitStep;
    float crushedRaw = (float)crushedInt;

    if (abs(crushedInt) <= bitStep) { // Μηδενισμός των κβαντισμένων τιμών μικρού
πλάτους
        crushedRaw = 0.0f;
    }

    crushCounter++; // Ανανέωση της κρατούμενης τιμής ανά downsampleFactor
δείγματα εξόδου

    if (crushCounter >= downsampleFactor) {
        crushCounter = 0;
        heldSample = crushedRaw;
    }

    // Το ORG Amount καθορίζει την ποσότητα RETRO, με μέγιστη συμμετοχή 90%
    float retroAmt = modelAmt * 0.90f;
    // Ανάμειξη του σήματος εισόδου με την κβαντισμένη και χρονικά διατηρούμενη
συνιστώσα
    float out = sample * (1.0f - retroAmt) + heldSample * retroAmt;
    out *= (1.0f - 0.05f * modelAmt);

    return limitCenteredAudio(out);
}
  // Επιλέγει τη συνάρτηση επεξεργασίας
  float applyInstrumentSound(float raw) {
    switch (voiceInstrument) {
        case INST_PIANO:
            return processPiano(raw);

        case INST_EPIANO:
            return processEPiano(raw);

        case INST_AIR:
            return processAir(raw);

        case INST_STRING:
            return processString(raw);

        case INST_HIT:
            return processHit(raw);

        case INST_METAL:
            return processMetal(raw);

        case INST_RETRO:
            return processRetro(raw);

        default:
            return raw;
    }
  }


  float applyLfoEffect(float sample){
```

```
    float lfoRate01  = voiceLfoRate;
    float lfoDepth01 = voiceLfoDepth;
    // Μέγιστο βάθος tremolo: 65%, ή 45% για το EPIANO
    float maxDepth = 0.65f;
      if (voiceInstrument  == INST_EPIANO) {
          maxDepth = 0.45f;
      }


    float rateHz = mapLfoRate(lfoRate01);   // 0.3 .. 8.0 Hz
    float depth  = maxDepth * clamp01(lfoDepth01); // 0 .. 0.8
      // Εφαρμογή tremolo μόνο όταν το βάθος υπερβαίνει το κατώφλι ενεργοποίησης
      if (depth > 0.001f) {
        // Μετατροπή της συχνότητας σε βήμα φάσης, με έναν κύκλο να αντιστοιχεί σε
2^32 μονάδες
        uint32_t step = (uint32_t)((4294967296.0f * rateHz) / SAMPLE_RATE);
        tremoloPhase += step; //Επαναφέρει κυκλικά τη φάση

        float lfo = triLFO(tremoloPhase);          // -1 .. +1
        float lfo01 = 0.5f + 0.5f * lfo;       // 0 .. 1
        // Περιοδικός συντελεστής πλάτους από 1-depth έως 1
        float gain = (1.0f - depth) + depth * lfo01;
        sample *= gain;
      }
      return sample;
  }


  float applyDelayEffect(float sample){

    float delayTime = map01(voiceDlyTime, 80.0f, 420.0f); // Χρόνος καθυστέρησης
από 80 έως 420 ms
    float delayMix  = map01(voiceDlyMix, 0.0f, 0.35f);   // Συντελεστής προσθήκης
του καθυστερημένου σήματος από 0 έως 0.35
    if (delayMix > 0.001f) {
      int delaySamples = (int)((delayTime * SAMPLE_RATE) / 1000.0f);// Μετατροπή
του χρόνου καθυστέρησης από ms σε δείγματα
      // Περιορισμός της καθυστέρησης
      if (delaySamples >= DLY_BUF_LEN)
        delaySamples = DLY_BUF_LEN - 1;

      // Θέση ανάγνωσης του καθυστερημένου δείγματος με κυκλική επαναφορά
      int readIndex = dlyWriteIndex - delaySamples;
      if (readIndex < 0)
        readIndex += DLY_BUF_LEN;

      float delayed = dlyBuffer[readIndex];
      float feedback = delayMix * 0.45f; // Η ανάδραση αυξάνεται μαζί με την
ποσότητα του delay
      float mixed = sample + delayed * delayMix; // Προσθήκη του καθυστερημένου
σήματος στην έξοδο

      //Αποθηκευση του αρχικού δείγματος μαζί με την ανάδραση
      dlyBuffer[dlyWriteIndex] = sample + delayed * feedback;
      dlyWriteIndex++;
     if (dlyWriteIndex >= DLY_BUF_LEN)
        dlyWriteIndex = 0;

    sample = mixed;
```

```
    }

    return sample;
  }

  float applyADSR(float sample){
    float shapedLocal = 512.0f + sample; // Επαναφορά της μετατόπισης στο μέσο της κλίμακας του DAC
    // Αρχική αύξηση του πλάτους με κυβική καμπύλη εξομάλυνσης
    if (startFadeCounter < startFadeLength) {
      float x = (float)startFadeCounter / (float)startFadeLength;
      float g = x * x * (3.0f - 2.0f * x);
      shapedLocal = 512.0f + (shapedLocal - 512.0f) * g;
    }
     // Πολύ ήπια εξομάλυνση μόνο στο attack
    if (startFadeCounter < 64) {
        float alpha = 0.70f;
        shapedLocal = alpha * shapedLocal + (1.0f - alpha) * attackSmoothState;
        attackSmoothState = shapedLocal;
        startFadeCounter++;
    }
    // Εξασθένηση πριν από το τέλος του πίνακα όταν το πλήκτρο παραμένει πατημένο
    if (!isReleasing && keyHeld) {
    int remainingIdx = (SampleIndexSize - 1) - sampleIndex; // Απόσταση από την τελευταία θέση του αποθηκευμένου πίνακα
      if (remainingIdx <= endFadeLen) {
        float f = (float)remainingIdx / (float)endFadeLen;
        shapedLocal = (shapedLocal - 512.0f) * f + 512.0f;
      }
    }
    // Γραμμική εξασθένηση μετά την επιβεβαιωμένη απελευθέρωση του πλήκτρου
    if (isReleasing) {
      if (releaseCounter < releaseLength) {
        float fadeFactor = 1.0f - (float)releaseCounter / releaseLength;
        shapedLocal = (shapedLocal - 512.0f) * fadeFactor + 512.0f;
        releaseCounter++;

      } else {
          // Ολοκλήρωση της νότας και επαναφορά της κατάστασης αναπαραγωγής
          active = false;
          isReleasing = false;
          data = nullptr;
          sampleIndex = 0;
          interpStep = 0;
          digitalWrite(SD_PIN, LOW);
          return 512.0f;
       }
    }
    return shapedLocal;
  }

  void advancePlayback(int maxInterp){
    interpStep++;
    // Μετά την ολοκλήρωση των βημάτων παρεμβολής προχωρά στο επόμενο αποθηκευμένο δείγμα
    if(interpStep >= maxInterp){
      interpStep = 0;
      sampleIndex++;
    }
```

```
  if (sampleIndex >= SampleIndexSize) {
    if (keyHeld && !isReleasing) {
      active = false;
      data = nullptr;
      sampleIndex = 0;
      interpStep = 0;

    } else {
      sampleIndex = 0;
    }
  }
}

void latchUiValuesForNote() {
  // Το όργανο που θα παίξει αυτή η νότα
  voiceInstrument = selectedInstrument;
  currentInstrument = voiceInstrument;

  // ORG page
  voiceOrgAmount = clamp01(uiVals.orgAmount);

  // SND page
  voiceSndChar   = clamp01(uiVals.sndChar);
  voiceSndAmount = clamp01(uiVals.sndAmount);

  // LFO page
  voiceLfoRate  = clamp01(uiVals.lfoRate);
  voiceLfoDepth = clamp01(uiVals.lfoDepth);

  // DLY page
  voiceDlyTime = clamp01(uiVals.dlyTime);
  voiceDlyMix  = clamp01(uiVals.dlyMix);
}

void configureInstrument(int newNote) {

  float sndChar01 = voiceSndChar;
  float sndAmount01 = voiceSndAmount;
  float orgAmount01 =  voiceOrgAmount;

  float sndBipolar = toBipolar(sndChar01);
  float baseFreq = Frequencies[newNote];
  switch (voiceInstrument) {
    case INST_PIANO: {
      releaseLength = 10000;
      tinePhaseStep = 0.0f;
      hammerPhaseStep = 0.0f;
      break;
    }
    case INST_EPIANO: {
      releaseLength = 12000;
      // SND Pot 1: mellow ↔ bright tine ratio
      float tineRatio = 12.0f + sndBipolar * 4.0f;
      // ORG Pot 2: πόσο έντονο είναι το EP layer
      tineIntensity = map01(orgAmount01, 0.0f, 460.0f);
      tinePhaseStep = (baseFreq * tineRatio * TWO_PI) / SAMPLE_RATE;
      float hammerFreq = baseFreq * 18.0f;

      if (hammerFreq > 9000.0f) {
```

```
    hammerFreq = 9000.0f;
  }
  hammerPhaseStep = (hammerFreq * TWO_PI) / SAMPLE_RATE;
  // SND Pot 2: hammer/detail
  hammerLength = 900 - (int)(sndAmount01 * 350.0f);
  if (hammerLength < 420) hammerLength = 420;

  break;
}
case INST_AIR: {
  releaseLength = 10000;
  break;
}
case INST_STRING: {
  releaseLength = 10000;

  pmStringDelaySamples = (int)(SAMPLE_RATE / baseFreq);
  if (pmStringDelaySamples < 20) {
    pmStringDelaySamples = 20;
  }

  if (pmStringDelaySamples > PM_STRING_BUF_LEN - 2) {
    pmStringDelaySamples = PM_STRING_BUF_LEN - 2;
  }
  break;
  }
  case INST_HIT: {
    releaseLength = 6000;

    // SND Pot 1 επηρεάζει πόσο χαμηλό/ψηλό είναι το hit
    float hitRatio = map01(sndChar01, 4.0f, 9.0f);

    float hitFreq = baseFreq * hitRatio;

    if (hitFreq > 3800.0f) {
      hitFreq = 3800.0f;
    }

    pmModelStepHit = (hitFreq * TWO_PI) / SAMPLE_RATE;

    break;
  }
  case INST_METAL: {
    releaseLength = 10000;

    // SND Pot 1 επηρεάζει τη συχνότητα του ring modulator
    float metalRatio = map01(sndChar01, 2.8f, 6.2f);

    float metalFreq1 = baseFreq * metalRatio;

    if (metalFreq1 > 8500.0f) {
      metalFreq1 = 8500.0f;
    }

    pmModelStepMetal1 = (metalFreq1 * TWO_PI) / SAMPLE_RATE;

    break;
  }
  case INST_RETRO: {
```

```
          releaseLength = 1600;

          break;
        }
        default: {
          releaseLength = 10000;
          break;
        }
    }
  }

  //Κεντρική αλυσίδα αναπαραγωγής
  float update( int maxInterp){
    // Επιστροφή στάθμης ηρεμίας όταν δεν υπάρχει ενεργή πηγή δειγμάτων
     if (!active || data == nullptr) return 512.0f;

     float raw = generateRawSample(maxInterp);

     raw = applyInstrumentSound(raw);
     raw = applyLfoEffect(raw);
     raw = applyDelayEffect(raw);

     float shapedOut = applyADSR(raw);

     if(!active) return 512.0f;

     advancePlayback(maxInterp);

     if(!active) return 512.0f;

     return constrain(shapedOut, 32.0f, 992.0f);

  }

void noteOn(const uint16_t* newData, int newNote) {
     clearAudioBuffer();// Καθαρισμός του buffer

     data = newData;
     note = newNote;
     sampleIndex = 0;
     interpStep = 0;
     active = true;
     isReleasing = false; // Reset Release State
     releaseCounter = 0;
     keyHeld = true;

     // μεταβλητές μοεφοποίησης
       pmToneState  = 0.0f;

       tinePhase = 0.0f;
       tinePhaseStep = 0.0f;
       tineEnvelope = 1.0f;
       tineIntensity = 240.0f;
       rngEp = 1u;

       hammerCounter = 0;
       hammerPhase = 0.0f;
       hammerPhaseStep = 0.0f;
```

```
        pmModelPhase1 = 0.0f;
        pmStringResState = 0.0f;

        pmModelEnvState = 0.0f;
        pmModelRng = 1u;
        pmAirNoiseState = 0.0f;

        pmStringWriteIndex = 0;
        pmStringDelaySamples = 80;
        pmStringDampState = 0.0f;
        for (int i = 0; i < PM_STRING_BUF_LEN; i++) {
            pmStringBuf[i] = 0.0f;
        }

        pmModelCounter = 0;

        crushCounter = 0;
        heldSample = 0.0f;
        downsampleFactor = 4;
        bitStep = 32;

        tremoloPhase = 0;
        dlyWriteIndex = 0;
        memset(dlyBuffer, 0, sizeof(dlyBuffer));

        startFadeCounter = 0;
        attackSmoothState = 512.0f;
      // Αντιγραφή των επιλογών της διεπαφής και υπολογισμός των παραμέτρων της
νότας
      latchUiValuesForNote();
      configureInstrument(newNote);
      // Ενεργοποίηση του ενισχυτή για την αναπαραγωγή
      digitalWrite(SD_PIN, HIGH);
  }

  void noteOff() {
    keyHeld = false;
    // Εκκίνηση του release μόνο μία φορά, εφόσον η φωνή παραμένει ενεργή
      if (active && !isReleasing) {
          isReleasing = true;
          releaseCounter = 0;
      }
  }
};

Voice currentVoice;// Μοναδική φωνή αναπαραγωγής του μονοφωνικού συνθετητή

// Αποστέλλει ένα δείγμα στον DAC σε κάθε διακοπή του timer ήχου
void IRAM_ATTR sendToDAC() {
    uint16_t dacSample = 512;
    portENTER_CRITICAL_ISR(&timerMux);
    if (audioCount > 0) {// Μετακίνηση της θέσης ανάγνωσης με κυκλική επαναφορά
        dacSample = audioBuffer[audioReadIndex];
        audioReadIndex++;
        if (audioReadIndex >= AUDIO_BUFFER_SIZE) audioReadIndex = 0;
        audioCount--;
    }
    portEXIT_CRITICAL_ISR(&timerMux);
    uint16_t outVal = 0x3000 | (dacSample << 2);
```

```
    // Επιλογή του DAC, μετάδοση της λέξης και ολοκλήρωση της αποστολής
    CS_LOW();
    SPI.transfer16(outVal);
    CS_HIGH();
}

// Θέτει αίτημα ελέγχου του πληκτρολογίου για τη loop()
void IRAM_ATTR checkKeyPress() {
    keyCheckFlag = true;
}

// Αδειάζει λογικά τον buffer μηδενίζοντας τους δείκτες και το πλήθος διαθέσιμων
δειγμάτων
void clearAudioBuffer() {

  portENTER_CRITICAL(&timerMux);
  audioReadIndex = 0;
  audioWriteIndex = 0;
  audioCount = 0;
  portEXIT_CRITICAL(&timerMux);

}

// Παράγει και αποθηκεύει δείγματα όσο υπάρχει διαθέσιμος χώρος και ενεργή φωνή
void fillAudioBuffer() {
  while (true) {

    portENTER_CRITICAL(&timerMux);
    bool bufferFull = (audioCount >= AUDIO_BUFFER_SIZE);
    portEXIT_CRITICAL(&timerMux);
    // Διακοπή πλήρωσης όταν ο buffer γεμίσει ή η φωνή έχει ολοκληρωθεί
    if (bufferFull) break;
    if (!currentVoice.active) break;

    float sample = currentVoice.update(maxInterp);// Παραγωγή επόμενου δείγματος

    if (sample < 0.0f) sample = 0.0f;//Περιορισμός σε 10-bit τιμή
    if (sample > 1023.0f) sample = 1023.0f;

    uint16_t dacSample = (uint16_t)sample;
    // Προστατευμένη εγγραφή δείγματος και ενημέρωση της κατάστασης του buffer
    portENTER_CRITICAL(&timerMux);
    if (audioCount < AUDIO_BUFFER_SIZE) {
      audioBuffer[audioWriteIndex] = dacSample;
      audioWriteIndex++; // Μετακίνηση της θέσης εγγραφής με κυκλική επαναφορά
      if (audioWriteIndex >= AUDIO_BUFFER_SIZE) audioWriteIndex = 0;
      audioCount++;
    }
    portEXIT_CRITICAL(&timerMux);
  }
}

// Αρχικοποιεί τους timers παραγωγής ήχου και ελέγχου πληκτρολογίου
void SetupTimers(){
    timer = timerBegin(1000000);
    timerAttachInterrupt(timer , &sendToDAC);
    timerAlarm(timer, 31, true, 0); // Διακοπή κάθε 31 μs

    keyTimer = timerBegin(1000000);
```

```
    timerAttachInterrupt(keyTimer , &checkKeyPress);
    timerAlarm(keyTimer, 2500, true, 0); // Διακοπή κάθε 2.5 ms
}

void setup() {
  // Σειριακή επικοινωνία και οθόνη OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC, 0x3C)) { //
    Serial.println(F("SSD1306 allocation failed"));
  }
  display.clearDisplay();

  // Αρχικοποίηση ψηφιακών ακροδεκτών
  pinMode(CS_PIN, OUTPUT);
  CS_HIGH();

  pinMode(SD_PIN, OUTPUT);
  digitalWrite(SD_PIN, LOW); // Ο ενισχυτής παραμένει απενεργοποιημένος μέχρι την
πρώτη νότα

  pinMode(HOLD_SWITCH_PIN, INPUT_PULLUP);

  // Αρχικοποίηση του SPI για την επικοινωνία με τον DAC
  SPI.begin(SCK_PIN, 19,  MOSI_PIN);
    SPI.beginTransaction(SPISettings(20000000, MSBFIRST, SPI_MODE0));
    CS_LOW();
    SPI.transfer16(0x3000 | (512 << 2));
    CS_HIGH();
    delay(50); // Σύντομη αναμονή πριν από την ενεργοποίηση του συστήματος

  // Ρύθμιση του ADC για το πληκτρολόγιο και τα δύο ποτενσιόμετρα
  analogReadResolution(12);
  analogSetPinAttenuation(keybPin, ADC_11db);
  analogSetPinAttenuation(POT_CHARACTER, ADC_11db);
  analogSetPinAttenuation(POT_AMOUNT, ADC_11db);

  // Έναρξη του συστήματος απο την αρχική επιλογή PIANO και τη σελίδα ORG
  resetSoundToRawPiano();
  // Έναρξη των χρονοδιακοπτών μετά την ολοκλήρωση της αρχικοποίησης του
συστήματος
  SetupTimers();

}

// ------Κουμπί διεπαφής----
bool lastButtonState = HIGH; // Προηγούμενη ακατέργαστη ανάγνωση του κουμπιού
unsigned long lastDebounceTime = 0; // Χρόνος τελευταίας αλλαγής
unsigned long debounceDelay = 50;  // Χρόνος σταθεροποίησης της ένδειξης στα 50ms
static bool btnWasDown = false;   // Έχει αναγνωριστεί πάτημα και αναμένεται
απελευθέρωση
static uint32_t btnDownMs = 0;   // Χρόνος αναγνώρισης του πατήματος
static const uint32_t LONG_PRESS_MS = 450; // Όριο διάκρισης παρατεταμένου
πατήματος

// --------Πληκτρολόγιο-----------
const int PRESS_STABLE = 2;    // Ίδιες διαδοχικές αναγνώσεις για επιβεβαίωση
νότας
const int RELEASE_STABLE = 8;  // Διαδοχικές αναγνώσεις χωρίς νότα για επιβεβαίωση
απελευθέρωσης
```

```
const int KEY_NONE     = -1;
const int KEY_UNSTABLE = -2;
const int RELEASE_ADC_MAX = 40;   // όριο απελευθέρωσης
const int PRESS_ADC_MIN   = 55;


static uint8_t potDiv = 0; // Μετρητής κύκλων ελέγχου πληκτρολογίου μεταξύ
αναγνώσεων ποτενσιόμετρων


// ---------- Οθόνη ----------
static Instrument lastInstrument = currentInstrument; // Ηχόχρωμα που
χρησιμοποιείται για τον τίτλο της οθόνης
static uint8_t lastCharPct = 255; // Αποθηκευμένο ποσοστό για την πρώτη μπάρα
static uint8_t lastAmtPct  = 255; // Αποθηκευμένο ποσοστό για τη δεύτερη μπάρα
static bool uiDirty = true; // Αίτημα επανασχεδίασης της οθόνης
// Αρχικά true ώστε να σχεδιαστεί η πρώτη εικόνα
// Τιμές αναφοράς για την ανίχνευση μεταβολής των χειριστηρίων
static float prevDisplayedP1 = 0.5f;
static float prevDisplayedP2 = 0.0f;
static bool showLiveValue = false; // Εμφάνιση τιμής παραμέτρου αντί του ονόματος
ηχοχρώματος
static uint32_t lastPotMoveMs = 0;
static const uint32_t LIVE_VALUE_HOLD_MS = 2500;   // Διάρκεια διατήρησης της
προσωρινής ένδειξης, 2.5ms

static uint8_t activePotNumber = 0;   // Χειριστήριο της ένδειξης: 1 ή 2
static char liveValueText[24] = "";  // Κείμενο της προσωρινής ένδειξης


int detectKeyboardNote(int val) {

  if (val <= RELEASE_ADC_MAX) {
    return KEY_NONE;
  }

  if (val < PRESS_ADC_MIN) {
    return KEY_UNSTABLE;
  }

  int bestNote = 0;
  int bestDiff = abs(val - NADCcenter[0]);

  for (int i = 1; i < 16; i++) {
    int diff = abs(val - NADCcenter[i]);

    if (diff < bestDiff) {
      bestDiff = diff;
      bestNote = i;
    }
  }

  return bestNote;
}


// Διαχειρίζεται τις εισόδους, ενημερώνει τη διεπαφή και τροφοδοτεί τον buffer
ήχου
void loop() {
```

```
  /// Κουμπί διεπαφής με έλεγχο σταθερής ένδειξης
  int reading = digitalRead(HOLD_SWITCH_PIN);
  if (reading != lastButtonState) { // Κάθε αλλαγή της ακατέργαστης ένδειξης
επανεκκινεί τον χρόνο σταθεροποίησης
    lastDebounceTime = millis();
  }
  // Επεξεργασία του κουμπιού αφού η ένδειξη παραμείνει σταθερή
  if ((millis() - lastDebounceTime) > debounceDelay) {
    // Καταγραφή της έναρξης του επιβεβαιωμένου πατήματος
    if (reading == LOW && !btnWasDown) {
      btnWasDown = true;
      btnDownMs = millis();
    }
    // Υπολογισμός διάρκειας και επιλογή ενέργειας κατά την απελευθέρωση
    else if (reading == HIGH && btnWasDown) {
      btnWasDown = false;
      uint32_t heldMs = millis() - btnDownMs;

      if (heldMs < LONG_PRESS_MS) {
        // Σύντομο πάτημα: μετάβαση στην επόμενη σελίδα
        currentPage = (UiPage)(((uint8_t)currentPage + 1) % 4);
        loadPageToPots(currentPage);

        lastCharPct = (uint8_t)lroundf(potCharacterSmooth * 100.0f);
        lastAmtPct  = (uint8_t)lroundf(potAmountSmooth    * 100.0f);

        prevDisplayedP1 = potCharacterSmooth;
        prevDisplayedP2 = potAmountSmooth;

        showLiveValue = false;
        lastPotMoveMs = 0;

        uiDirty = true;
      }
      else {
        // Παρατεταμένο πάτημα: επαναφορά των ρυθμίσεων στο αρχικό PIANO
        resetSoundToRawPiano();

        lastInstrument = currentInstrument;

        lastCharPct = (uint8_t)lroundf(potCharacterSmooth * 100.0f);
        lastAmtPct  = (uint8_t)lroundf(potAmountSmooth    * 100.0f);

        prevDisplayedP1 = potCharacterSmooth;
        prevDisplayedP2 = potAmountSmooth;

        showLiveValue = false;
        lastPotMoveMs = 0;

        uiDirty = true;
      }
    }
  }

lastButtonState = reading;

  //Ακόμα και όταν ο δείκτης δεν ελέγχει το πληκτρολόγιο, εξυπηρετεί την οθόνη και
την συνάρτηση fillAudioBuffer
```

```
  if (!keyCheckFlag) {
    if (uiDirty) {
      updateDisplayUI(lastInstrument, currentPage, lastCharPct, lastAmtPct,
showLiveValue, liveValueText);
      uiDirty = false;
    }
    fillAudioBuffer();
    return;
  }//Ενεργοποίηση της σημαία πληκτρολογίου

  keyCheckFlag = false;

  analogRead(keybPin);
  int val = analogRead(keybPin);
  // Επιλογή της πρώτης νότας η οποία βρίσκεται πιο κοντά στον πίνακα μέσης τιμής
  int detectedNote = detectKeyboardNote(val);;

    static int lockedNote = -1;
    static bool keyIsDown = false;

    static int  pressCount = 0;         // σταθερότητα πατήματος
    static int  noneCount  = 0;         // σταθερότητα απελευθέρωσης (-1)
    static int candidate = -1;        // Υποψήφια νότα
   if (!currentVoice.active && !keyIsDown) {

  if (detectedNote >= 0) {

    noneCount = 0;

    if (detectedNote == candidate) {
      pressCount++;
    } else {
      candidate = detectedNote;
      pressCount = 1;
    }

    if (pressCount >= PRESS_STABLE) {

      lockedNote = candidate;
      keyIsDown  = true;

      currentVoice.noteOn(noteSignals[lockedNote], lockedNote);

      lastInstrument = currentInstrument;
      uiDirty = true;

      pressCount = PRESS_STABLE;
    }
  }

  else {

    pressCount = 0;
    candidate = -1;
  }
}
else { //Όσο υπάρχει ενεργή νότα ή κλειδωμένο πλήκτρο,
       //ελέγχουμε μόνο αν ο χρήστης άφησε το πλήκτρο.
```

```
  pressCount = 0;
  candidate = -1;

  // Μόνο καθαρή πτώση κάτω από RELEASE_ADC_MAX
  if (detectedNote == KEY_NONE) {

    noneCount++;

    if (noneCount >= RELEASE_STABLE) {

      if (keyIsDown) {
        currentVoice.noteOff();
      }

      keyIsDown  = false;
      lockedNote = -1;
      noneCount  = 0;
    }
  }

  else {
    // Αν η τιμή είναι έγκυρη νότα ή ασταθής περιοχή,
    // ο χρήστης δεν άφησε το πλήκτρο.
    noneCount = 0;
  }
}

// Ανάγνωση ποτενσιόμετρων ανά δέκα  ελέγχους πληκτρολογίου
  if (++potDiv >= 10) {
  potDiv = 0;

  // Ανάγνωση των δύο ποτενσιόμετρων με απόρριψη της πρώτης μέτρησης κάθε εισόδου
  analogRead(POT_CHARACTER);
  int rawCharacter = analogRead(POT_CHARACTER);

  analogRead(POT_AMOUNT);
  int rawAmount = analogRead(POT_AMOUNT);
  // Βαθμονόμηση του αξιοποιήσιμου εύρους των ποτενσιόμετρων
  float characterNow = rawCharacter * (1.0f / 4095.0f);
  float amountNow    = rawAmount    * (1.0f / 4095.0f);
  characterNow = calibratePot(characterNow, 0.04f, 0.95f);
  amountNow    = calibratePot(amountNow,    0.04f, 0.95f);
  // Προσαρμογή της φοράς μεταβολής των χειριστηρίων
  characterNow = invertPot(characterNow, INVERT_POT_CHARACTER);
  amountNow    = invertPot(amountNow,    INVERT_POT_AMOUNT);


  if (currentPage == PAGE_SND && fabsf(characterNow - 0.5f) < 0.035f) characterNow
= 0.5f;

  // Αντιστοίχιση των τιμών κοντά στα άκρα σε ακριβώς 0 ή 1
  if (characterNow < 0.05f) characterNow = 0.0f;
  if (characterNow > 0.95f) characterNow = 1.0f;
  if (amountNow < 0.03f) amountNow = 0.0f;
  if (amountNow > 0.97f) amountNow = 1.0f;

  // Ανάκτηση των αποθηκευμένων τιμών της ενεργής σελίδας
  float storedP1, storedP2;
```

```
  getStoredPageValues(currentPage, storedP1, storedP2);

  // ---------------- Πρώτο χειριστήριο - soft takeover ----------------
  if (!pot1Latched) {
    if (isPotNearStored(characterNow, storedP1)) {
      pot1Latched = true;
    } else {
      characterNow = storedP1;
    }
  }

  // ---------------- Δεύτερο χειριστήριο - soft takeover ----------------
  if (!pot2Latched) {
    if (isPotNearStored(amountNow, storedP2)) {
      pot2Latched = true;
    } else {
      amountNow = storedP2;
    }
  }

  // Εξομάλυνση των τιμών πριν από την ενημέρωση των παραμέτρων
  potCharacterSmooth = 0.90f * potCharacterSmooth + 0.10f * characterNow;
  potAmountSmooth    = 0.90f * potAmountSmooth    + 0.10f * amountNow;

  //Κατά την ενεργή νότα διατηρείται η επιλογή ηχοχρώματος και αποθηκεύεται μόνο
το ORG Amount
  if (currentPage == PAGE_ORG && currentVoice.active) {
    uiVals.orgAmount = potAmountSmooth;
  }
  else {
    storePotsToPage(currentPage, potCharacterSmooth, potAmountSmooth);
  }


  if (currentPage == PAGE_ORG && !currentVoice.active) {
    Instrument newSelected = instrumentFromOrgPot(uiVals.orgSelect);
    // Ενημέρωση ηχοχρώματος μόνο όταν δεν υπάρχει ενεργή φωνή
    if (newSelected != selectedInstrument) {
      selectedInstrument = newSelected;
      lastInstrument = selectedInstrument;
      uiDirty = true;
    }
  }

  // Κατώφλι μεταβολής για ενεργοποίηση της προσωρινής ένδειξης παραμέτρου
  const float NOISE_TOLERANCE = 0.095f;

  bool pot1Moved = fabsf(potCharacterSmooth - prevDisplayedP1) >= NOISE_TOLERANCE;
  bool pot2Moved = fabsf(potAmountSmooth    - prevDisplayedP2) >= NOISE_TOLERANCE;

  if (instrumentUsesUiControls(currentInstrument) && (pot1Moved || pot2Moved)) {
    showLiveValue = true;
    lastPotMoveMs = millis();

    if (pot1Moved && !pot2Moved) {
      activePotNumber = 1;
    }
    else if (!pot1Moved && pot2Moved) {
      activePotNumber = 2;
```

```
    }
    else {
      // Το δεύτερο χειριστήριο έχει προτεραιότητα όταν μεταβληθούν και τα δύοο
      activePotNumber = 2;
    }
    // Δημιουργία του κειμένου της προσωρινής ένδειξης
    formatLiveValueText(currentInstrument, currentPage, activePotNumber,
liveValueText, sizeof(liveValueText));

    prevDisplayedP1 = potCharacterSmooth;
    prevDisplayedP2 = potAmountSmooth;
    uiDirty = true;
  }

  // Ενημέρωση των μπαρών όταν το ποσοστό διαφέρει τουλάχιστον κατά 10 μονάδες
  uint8_t charPct = (uint8_t)lroundf(potCharacterSmooth * 100.0f);
  uint8_t amtPct  = (uint8_t)lroundf(potAmountSmooth    * 100.0f);

  if (abs((int)charPct - (int)lastCharPct) >= 10) {
    lastCharPct = charPct;
    uiDirty = true;
  }

  if (abs((int)amtPct - (int)lastAmtPct) >= 10) {
    lastAmtPct = amtPct;
    uiDirty = true;
  }
}
  // Επιστροφή στο όνομα ηχοχρώματος μετά τη λήξη του χρόνου προσωρινής εμφάνισης
  if (showLiveValue && (millis() - lastPotMoveMs > LIVE_VALUE_HOLD_MS)) {
  showLiveValue = false;
  uiDirty = true;
  }

  if (uiDirty) {
    updateDisplayUI(lastInstrument, currentPage, lastCharPct, lastAmtPct,
showLiveValue, liveValueText);
    uiDirty = false;
  }
  // Παραγωγή νέων δειγμάτων για την αναπλήρωση του buffer ήχου
  fillAudioBuffer();

}//loop End



static void formatLiveValueText(
  Instrument inst,
  UiPage page,
  uint8_t potNumber,
  char* out,
  size_t outSize
) {

  float p1, p2;
  // Ανάκτηση των παραμέτρων της σελίδας
  getStoredPageValues(page, p1, p2);
  float value = (potNumber == 1) ? p1 : p2;// Επιλογή τιμής ανάλογα με το
χειριστήριο
```

```
if (page == PAGE_ORG) {
  if (potNumber == 1) {
    Instrument selected = instrumentFromOrgPot(value);
    snprintf(out, outSize, "%s", instrumentName(selected));
  } else {
    uint8_t amt = quantizeTo10(value) * 10;
    Instrument shownInst = selectedInstrument;

    if (shownInst == INST_PIANO) {
      snprintf(out, outSize, "AMNT OFF");
    } else {
      snprintf(out, outSize, "AMNT %u%%", amt);
    }
  }
}
else if (page == PAGE_SND) {

  if (inst == INST_PIANO) {
    if (potNumber == 1) {
      int8_t step = quantizeBipolarTo5(value);   // -5 .. +5
      snprintf(out, outSize, "CHAR %d", step);
    } else {
      uint8_t amt = quantizeTo10(value) * 10;
      snprintf(out, outSize, "AMNT %u%%", amt);
    }
  }
  else if (inst == INST_EPIANO) {
    if (potNumber == 1) {
      float ratio = 12.0f + toBipolar(value) * 3.0f;   // 9..15
      snprintf(out, outSize, "RATIO %.1f", ratio);
    } else {
      float tine = map01(value, 180.0f, 320.0f);
      snprintf(out, outSize, "HAMR %.0f", tine);
    }
  }
  else if (inst == INST_AIR) {
    if (potNumber == 1) {
      int8_t step = quantizeBipolarTo5(value);
      snprintf(out, outSize, "AIR %d", step);
    } else {
      uint8_t amt = quantizeTo10(value) * 10;
      snprintf(out, outSize, "MOVE %u%%", amt);
    }
  }
  else if (inst == INST_STRING) {
    if (potNumber == 1) {
      int8_t step = quantizeBipolarTo5(value);
      snprintf(out, outSize, "BODY %d", step);
    } else {
      uint8_t amt = quantizeTo10(value) * 10;
      snprintf(out, outSize, "RES %u%%", amt);
    }
  }
  else if (inst == INST_HIT) {
    if (potNumber == 1) {
      int8_t step = quantizeBipolarTo5(value);
      snprintf(out, outSize, "PITCH %d", step);
    } else {
```

```
        uint8_t amt = quantizeTo10(value) * 10;
        snprintf(out, outSize, "HIT %u%%", amt);
      }
    }
    else if (inst == INST_METAL) {
      if (potNumber == 1) {
        int8_t step = quantizeBipolarTo5(value);
        snprintf(out, outSize, "RING %d", step);
      } else {
        uint8_t amt = quantizeTo10(value) * 10;
        snprintf(out, outSize, "DEPTH %u%%", amt);
      }
    }
    else if (inst == INST_RETRO) {
      if (potNumber == 1) {
        int ds = 4 + (int)(toBipolar(value) * 3.0f);
        if (ds < 1) ds = 1;
        if (ds > 8) ds = 8;
          snprintf(out, outSize, "DS x%d", ds);
      } else {
        int bits = (int)map01(value, 8.0f, 64.0f);
        if (bits < 8) bits = 8;
        snprintf(out, outSize, "BITS %d", bits);
      }
    }
    else {
      snprintf(out, outSize, "P%d %.2f", potNumber, value);
    }
  }
  else if (page == PAGE_LFO) {
    if (potNumber == 1) {
      float hz = mapLfoRate(value);
      snprintf(out, outSize, "RATE %.1fHz", hz);
    } else {
      uint8_t depth = quantizeTo10(value) * 10;
      snprintf(out, outSize, "DEPTH %u%%", depth);
    }
  }

  else if (page == PAGE_DLY) {
    if (potNumber == 1) {
      float ms = map01(value, 80.0f, 420.0f);
      snprintf(out, outSize, "TIME %.0fms", ms);
    } else {
      float mix = map01(value, 0.0f, 45.0f);
      snprintf(out, outSize, "MIX %.0f%%", mix);
    }
  }

  else {
    snprintf(out, outSize, "P%d %.2f", potNumber, value);
  }
}
static void drawUnipolarBar(
  int x, int y, int w, int h,
  uint8_t percent
) {

  int fillW = (w * percent) / 100;
```

```
  if (fillW < 0) fillW = 0;
  if (fillW > w) fillW = w;

  // γεμάτο μέρος
  if (fillW > 0)
    display.fillRect(x, y, fillW, h, SSD1306_WHITE);

  // κενό μέρος
  if (fillW < w)
    display.fillRect(x + fillW, y, w - fillW, h, SSD1306_BLACK);

  // εφέ χάρακα
  for (int i = 0; i <= 10; i++) {

    int tx = x + (i * w) / 10;

    // Αποφυγή τροποποίησης των άκρων της μπάρας
    if (tx == x || tx == x + w - 1) continue;

    int tickH = (i == 0 || i == 5 || i == 10) ? h : h / 2;
    int tickY = y + h - tickH;

    display.drawFastVLine(tx, tickY, tickH, SSD1306_BLACK);
  }


  display.drawFastVLine(x,         y, h, SSD1306_WHITE);
  display.drawFastVLine(x + w - 1, y, h, SSD1306_WHITE);
}

static void drawBipolarBar(
  int x, int y, int w, int h,
  uint8_t percent
) {

  int center = x + w / 2;


  display.fillRect(x, y, w, h, SSD1306_BLACK);

  int bipolar = (int)percent - 50;

  if (bipolar > 0) {

    int fill = (bipolar * (w/2)) / 50;

    display.fillRect(center, y, fill, h, SSD1306_WHITE);

  }
  else if (bipolar < 0) {

    int fill = ((-bipolar) * (w/2)) / 50;

    display.fillRect(center - fill, y, fill, h, SSD1306_WHITE);
  }

  // εφέ χάρακα
  for (int i = 0; i <= 10; i++) {
```

```
    int tx = x + (i * w) / 10;

    int tickH = (i == 0 || i == 5 || i == 10) ? h : h / 2;
    int tickY = y + h - tickH;

    display.drawFastVLine(tx, tickY, tickH, SSD1306_WHITE);
  }


  display.drawFastVLine(center, y, h, SSD1306_WHITE);
}



void updateDisplayUI(Instrument inst, UiPage page, uint8_t charPct, uint8_t
amtPct, bool showValue, const char* valueText) {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);

  uint8_t drawCharPct = charPct;
  uint8_t drawAmtPct  = amtPct;



  display.setTextSize(2);

  const char* title = showValue ? valueText : instrumentName(inst);

  int textW = (int)strlen(title) * 12;
  int x = (128 - textW) / 2;
  if (x < 0) x = 0;

  display.setCursor(x, 0);
  display.print(title);

  // ---------------- Σελίδες ----------------
  display.setTextSize(1);

  int tabY = 20;
  int tabX = 2;
  int tabW = 29;
  int tabH = 10;

  auto drawTab = [&](int x0, const char* label, bool active) {
    if (active) {
      display.fillRect(x0, tabY, tabW, tabH, SSD1306_WHITE);
      display.setTextColor(SSD1306_BLACK);
    } else {
      display.drawRect(x0, tabY, tabW, tabH, SSD1306_WHITE);
      display.setTextColor(SSD1306_WHITE);
    }

    int labelX = x0 + 5;
    if (strcmp(label, "SND") == 0) labelX = x0 + 5;
    if (strcmp(label, "LFO") == 0) labelX = x0 + 6;
    if (strcmp(label, "DLY") == 0) labelX = x0 + 6;

    display.setCursor(labelX, tabY + 2);
```

```
    display.print(label);
    display.setTextColor(SSD1306_WHITE);
  };

  drawTab(tabX + 0*(tabW+3), "ORG", page == PAGE_ORG);
  drawTab(tabX + 1*(tabW+3), "SND", page == PAGE_SND);
  drawTab(tabX + 2*(tabW+3), "LFO", page == PAGE_LFO);
  drawTab(tabX + 3*(tabW+3), "DLY", page == PAGE_DLY);
  // ----------------  Χαρακτήρες εμφάνισης ----------------
  const char* p1Label = "P1";
  const char* p2Label = "P2";

  if (page == PAGE_ORG)   { p1Label = "ORG"; p2Label = "AMNT"; }
  if (page == PAGE_SND) { p1Label = "CHAR"; p2Label = "AMNT"; }
  if (page == PAGE_LFO)   { p1Label = "RATE"; p2Label = "DEPTH"; }
  if (page == PAGE_DLY)   { p1Label = "TIME"; p2Label = "MIX"; }

  // ---------------- Γραμμή 1 ----------------
  display.setTextSize(1);
  display.setCursor(0, 36);
  display.print(p1Label);

  // CHAR = bipolar, όλα τα άλλα unipolar
  if (page == PAGE_SND) {
    drawBipolarBar(34, 34, 90, 10, drawCharPct);
  } else {
    drawUnipolarBar(34, 34, 90, 10, drawCharPct);
  }

  // ---------------- Γραμμή 2 ----------------
  display.setCursor(0, 51);
  display.print(p2Label);
  drawUnipolarBar(34, 49, 90, 10, drawAmtPct);

  display.display();
}
```