Σχεδιασμός και υλοποίηση ψηφιακού ισοσταθμιστή ήχου.

Ευάγγελος Λάμπρου

Περίληψη

Στα πλαίσια αυτής της εργασίας γίνεται ο σχεδιασμός και υλοποίηση ενός ισοσταθμιστή ήχου. Η ανάπτυξή του συστήματος ψηφιακής επεξεργασίας σήματος, όπως και η γραφική διεπαφή έγιναν μέσα στο περιβάλλον του JUCE Framework χρησιμοποιώντας τη γλώσσα C++. Στην εργασία αυτή γίνεται η ανάλυση της θεωρίας πίσω από την ανάπτυξη των φίλτρων, την υλοποίηση του συστήματος από το χαμηλό έως το υψηλό επίπεδο αφαιρετικότητας που προσφέρει το πακέτο JUCE. Τέλος, υπάρχει ανάλυση των επιδόσεων του φίλτρου όπως και μία σύντομη αναφορά πάνω στην ευκολία χρήσης του.

1 Εισαγωγή

1.1 Στόχοι

Τελικός στόχος της εργασίας είναι η δημιουργία ενός ψηφιακού ισοσταθμιστή. Μία τέτοια εφαρμογή μπορεί να απλοποιηθεί ως ένα φίλτρο. Προφανώς, το φίλτρο αυτό θα είναι ψηφιακό και θα πρέπει να δέχεται ως είσοδο διακριτές τιμές οι οποίες θα είναι δείγματα ήχου. Η επεξεργασία των σημάτων αυτών πρέπει να γίνεται σε πραγματικό χρόνο. Αυτή η ιδιότητα ίσως να μην είναι υποχρεωτική, εφόσων ο χρήστης θα μπορούσε απλά να εισάγει ένα αρχείο ήχου στο φίλτρο μας και να παίρνει ένα άλλο αρχείο πίσω. Τότε, ο χρόνος επεξεργασίας θα μπορούσε να είναι αρκετά μεγάλος (εντός λογικών ορίων). Ωστόσο, στόχος μας είναι η ανάπτυξη μιας εφαρμογής που θα μπορεί να χρησιμοποιηθεί εντός ενός περιβάλλοντος δημιουργίας μουσικής. Εκεί, η επεξεργασία του σήματος σε πραγματικό χρόνο είναι απαραίτητη μιας και ο χρήστης θα χρειάζεται να ακούει το αποτέλεσμα της επεξεργασίας ενώ δουλεύει. Ακόμα, θα πρέπει να μπορεί να αλλάζει τα χαρακτηριστικά του φίλτρου ενώ αυτό δουλεύει. Ο τρόπος με τον οποίο θα αλλάζουν οι ιδιότητες του φίλτρου είναι μέσω μίας γραφικής διεπαφής η οποία επίσης πρέπει να υλοποιηθεί.

1.2 Αντίστοιχες Εφαρμογές

Η ιδέα για την υλοποίηση ενός φίλτρου δεν είναι καινούρια. Υπάρχουν ήδη πολλές υλοποιήσεις. Στο σημείο αυτό θα κάνουμε μία σύντομη παρουσίαση μερικών άλλων filter plugins [1], επισημαίνοντας ορισμένα τους αξιοσημείωτα χαρακτηριστικά.

1.2.1 LSP Parametric Equalizer

Η οικογένεια plugins των LSP (Linux Studio PLugins) είναι πολύ δημοφιλή στους κύκλους των μουσικών σε πλατφόρμες GNU/Linux. Στην περίπτωση αυτού του φίλτρου [1], ο χρήστης έχει μπορστά του μία πολύ σύνθετη διεπαφή, μέσω της οποίας μπορεί να κόψει ή να αυξήσει συχνότητες σε 16 διαφορετικά σημεία του φάσματος. Ακόμα, υπάρχει οπτικοποίηση του σήματος πριν και μετά το φίλτρο με FFT. Ακόμα, δίνεται η δυνατότητα επιλογής μεταξύ IIR, FIR φίλτρου.
Στόχος του plugin δεν είναι ο χρωματιστός του ήχου άλλα η διόρθωση του σήματος στο πεδίο της συχνότητας. Κάποιος μπορεί να επιλέξει το ακριβές είδος φιλτραρίσματος μεταξύ επιλογών όπως Bell Filter, Resonance Filter, Billinear Transform Filter (BT), Match Z Transform (MZ) Filter κ.α. Στις οδηγίες χρήσης της εφαρμογής δίνονται οδηγίες σχετικά με σενάρια στα οποία υπερτερεί το κάθε είδος φίλτρου όταν εφαρμόζεται κατά την ισοστάθμιση.

1.2.2 Calf Equalizer

Η οικογένεια των Calf [2] plugins αποτελεί επίσης μία πολύ δημολιφής επιλογή στον κόσμο των plugins ανοιχτού κώδικα. Η γραφική διεπαφή κρατάει τα απαραίτητα ώστε να γίνει λεπτομερής επεξεργασία του φάσματος, χωρίς όμως να προσφέρει εξαντλητικές επιλογές. Ωστόσο, η πιο απλοϊκή εμφάνιση ίσως να το κάνει καταλληλότερο για καθημερινή χρήση στα πλαίσια παραγωγής μουσικής.

2 Σχεδιασμός

2.1 Θεωρία IIR φίλτρων

Στην εργασία αυτή θα περιοριστούμε στην υλοποίηση φίλτρων 2ης τάξης, με τους αλγόριθμους όμως να είναι ικανούς να παράγουν φίλτρα οσοδήποτε μεγάλης τάξης (χωρίς βέβαια να εγγυάται η ορθότητα του τελικού αποτελέσματος λόγω πεπαρασμένης ακρίβειας στις τιμές των συντελεστών).

[1] Όσα λογισμικά αναφερθούν είναι ανοιχτού κώδικα και είναι διαθέσιμα σε κάθε κύρια πλατφόρμα.

Σχήμα 1.1: Στιγμιότυπο του LSP Parametric Equalizer

Σχήμα 1.2: Στιγμιότυπο του Calf Equalizer

Έχοντας ως βάση ένα χαμηλοπερατό φίλτρο 2ης τάξης [3]

Σχήμα 2.1: Βασική δομή του plugin.

$$H(s) = \frac{1}{s^2 + s/Q + 1} \tag{2.1}$$

Μέσω του διπολικού Z μετασχηματισμού, θέτωντας $s = \frac{1}{K}\frac{z-1}{z+1}$ $K = \frac{1}{2F_s}$ έχουμε [4]

$$H(z) = \frac{K^2 + 2K^2z^{-1} + K^2z^{-2}}{(K^2 + K/Q + 1) + 2(K^2 - 1)z^{-1} + (K^2 - K/Q + 1)z^{-2}} \tag{2.2}$$

Φίλτρα Butteworth Επιλέξαμε το φίλτρο Butterworth σχεδόν αυθαίρετα. Ωστόσο, το φίλτρο έχει ορσιμένες ιδιότητες που το κάνουν θεμιτό στα πλαίσια επεξεργασίας ήχου. Αρχικά, τα φίλτρα αυτά δεν έχουν ripple κοντά στη συχνότητα αποκοπής και έτσι δεν υπάρχει ενίσχυση καμίας συχνότητας. Ακόμα, έχουν πιο αργή αποκοπή ($-6ndb/οκτβα$, όπου n η τάξη του φίλτρου). Αυτό σε εφαρμογές ήχου είναι πολλές φορές θεμιτό μιας και δίνει σαν αποτέλεσμα ένα πιο μαλακό κόψιμο των συχνοτήτων. Συνύθως δεν θέλουμε να διώξουμε ολοκληρωτικά τις αρμονικές από κάποιο όργανο.
Για τους συντελεστές του φίλτρου Butterworth μπορούμε απλά να υπολογίσουμε την τιμή της σταθεράς Q και να πάρουμε τους συντελεστές από την εξ. (2.1) [3, 5]

$$1/Q = 2cos(\frac{2i+1}{n}\pi),\ i = 0..\frac{n}{2} \qquad n = \text{άρτιο} \tag{2.3}$$

$$1/Q = 2cos(\frac{i+1}{2n}\pi),\ i = 0..\frac{n}{2} \qquad n = \text{περιττό} \tag{2.4}$$

Αλγόριθμος Υπολογισμού Συντελεστών Από την προηγούμενη μαθηματική ανάλυση, ο υπολογισμός των συντελεστών για το εκάστοτε φίλτρο είναι πλέον τετριμένος.
Αντικαθιστώντας τις τιμές για το Q για ένα φίλτρο Butteworth στη σχέση 2.2 έχουμε πλέον τους συντελεστές του φίλτρου.

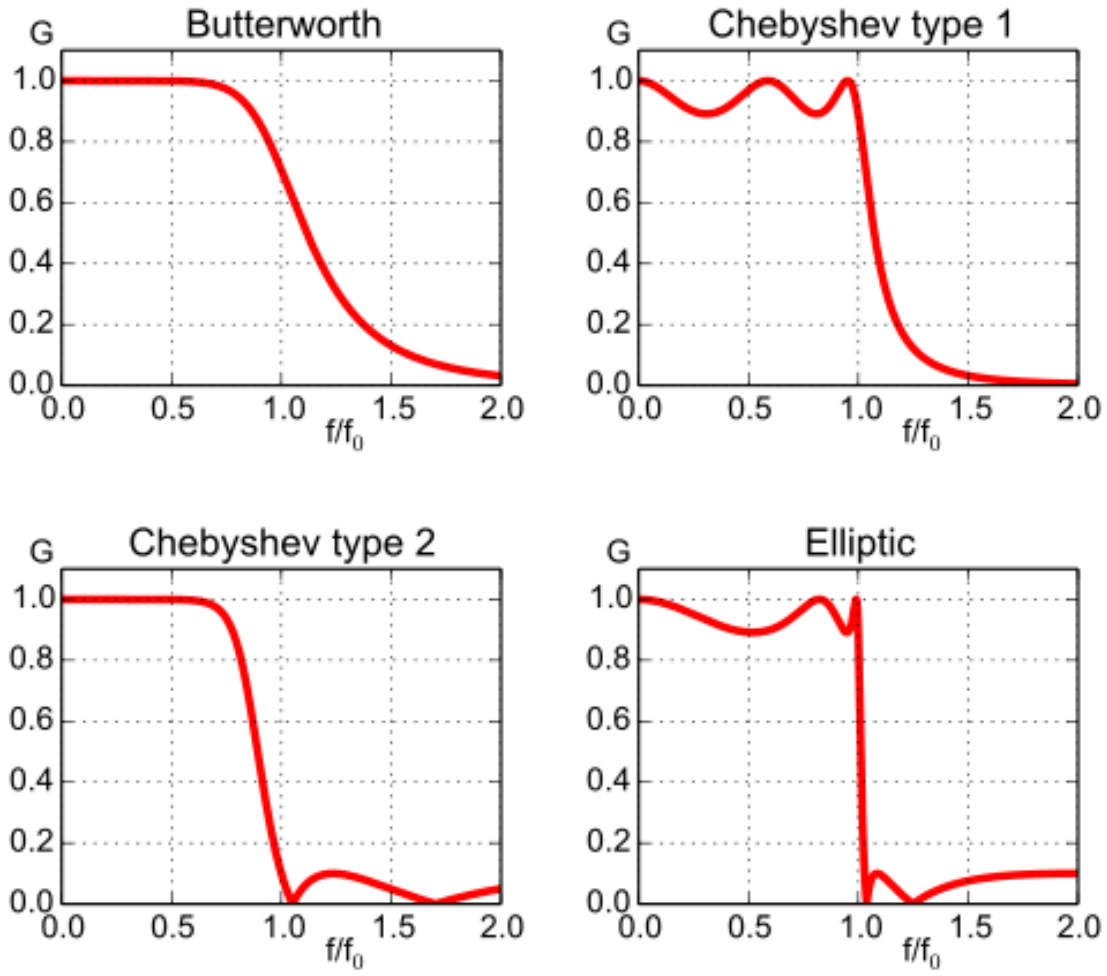
$$a_0 = \frac{K^2}{K^2 + K/Q + 1} \qquad a_1 = 2a_0 \qquad a_2 = a_0 \tag{2.5}$$

$$b_0 = 1 \qquad b_1 = \frac{2(K^2 - 1)}{K^2 + K/Q + 1} \qquad b_2 = \frac{K^2 - K/Q + 1}{K^2 + K/Q + 1} \tag{2.6}$$

Ένα φίλτρο Butteworth 1ης τάξης έχει αποκοπή $-6dB/octave$, 2ης τάξης έχει αποκοπή $-12dB/octave$, ένα 3ης τάξης $-18dB/octave$ κοκ.

3 Υλοποίηση

Για την υλοποίηση του φίλτρου χρησιμοποιήθηκαν οι βιβλιοθήκες του πακέτου Juce. Το πακέτο αυτό χωρίζεται σε ενότητες (modules), από τις οποίες χρησιμοποιούμε κατά βάση την ενότητα `juce::dsp`. Εδώ, περιλαμβάνονται κλάσεις μέσω την οποίων μπορεί να γίνει η δημιουργία ενός φίλτρου.
Ο τρόπος με τον οποίο η εφαρμογή μας επεξεργάζεται τα εισερχόμενα δείγματα (samples) είναι στη ρουτίνα `processBlock` όπου δεχόμαστε σαν είσοδο έναν buffer από δείγματα στα οποία μπορούσαμε να εφαρφμόσουμε οποιονδήποτε αλγόριθμο.



Σχήμα 2.2: Σύγκριση φίλτρου Butterworth με άλλα φίλτρα.

Το plugin στην πραγματικότητα αποτελείται από τρία (3) φίλτρα:

1. Ένα χαμηλοπερατό φίλτρο (low pass filter)
2. Ένα φίλτρο κορυφής
3. Ένα υψηπερατό φίλτρο

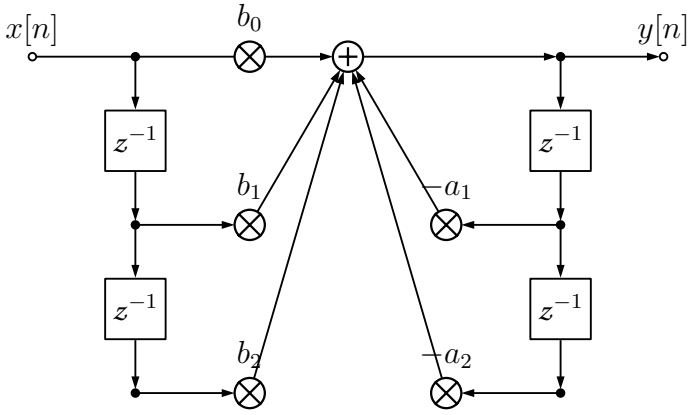
Σχήμα 3.1: Βασική δομή του plugin.

Ο τρόπος με τον οποίο αυτό περιγράφεται στον κώδικα είναι μέσω της δομής `ProcessorChain` η οποία αναπαριστά μία αλυσιδά επεξεργαστών. Κατά την εκτέλεση της εφαρμογής, η ακολουθία από δείγματα περνάει από την ρουτίνα επεξεργασίας του κάθε φίλτρου διαδοχικά. [5]

Στην περίπτωση που θέλαμε να έχουμε φίλτρο με πιο απότομη καμπύλη αποκοπής συχνοτήτων (slope), θα μπορούσαμε να βάλουμε σε σειρά στο `ProcessorChain` περισσότερα φίλτρα σε σειρά. Έτσι, ανάλογα με την καμπύλη που θέλαμε θα μπορούσαμε να θέτουμε σε πέρασμα (passthrough) όλα τα επίπεδα της αλυσίδας επεξεργασίας και να ενεργοποιούσαμε τον κατάλληλο αριθμό από φίλτρα σε σειρά ανάλογα με το πόσο 'απότομη' θα έπρεπε να είναι η καμπύλη.

Ο υπολογισμός της νέας αριθμητικής τιμής του κάθε δείγματος γίνεται με τη σειρά που φαίνεται στη φιγ. 3.2. Η καθυστέρηση του σήματος εισόδου γίνεται εφόσων το plugin αποθηκεύει την κατάστασή του, η οποία είναι ένας πίνακας με τις τιμές της εξόδου του στα προηγούμενα δείγματα. Στην περίπτωση ενός φίλτρου 2ης τάξης αρκεί να αποθηκεύουμε τις δύο προηγούμενες τιμές εξόδου του.

Η υλόποιση σε κώδικα όπως φαίνεται στην καταχώριση 1 ταυτίζεται με τη σχηματική αναπαράσταση. Η βιβλιοθήκη `JUCE` έχει προνοήσει για τη χρήση τον ρουτίνων της με αριθμητικούς τύπους διαφορετικής ακρίβειας, εξού και η χρήση της λέξης-κλειδί `auto` [6], η οποία αναθέτει στον compiler το συμπέρασμα του τύπου της εκάστοτε μεταβλητής.



Σχήμα 3.2: Υλοποίηση του IIR φίλτρου

Listing 1: Ο υπολογισμός της νέας τιμής για το κάθε δείγμα σε C++

```
auto* coeffs = coefficients->getRawCoefficients();

auto b0 = coeffs[0];
auto b1 = coeffs[1];
auto b2 = coeffs[2];
auto a1 = coeffs[3];
auto a2 = coeffs[4];

auto lv1 = state[0];
auto lv2 = state[1];

for (size_t i = 0; i < numSamples; ++i)
{
    auto input = src[i];
    auto output = (input * b0) + lv1;
    dst[i] = bypassed ? input : output;

    lv1 = (input * b1) - (output* a1) + lv2;
    lv2 = (input * b2) - (output* a2);
}

util::snapToZero (lv1); state[0] = lv1;
util::snapToZero (lv2); state[1] = lv2;
```

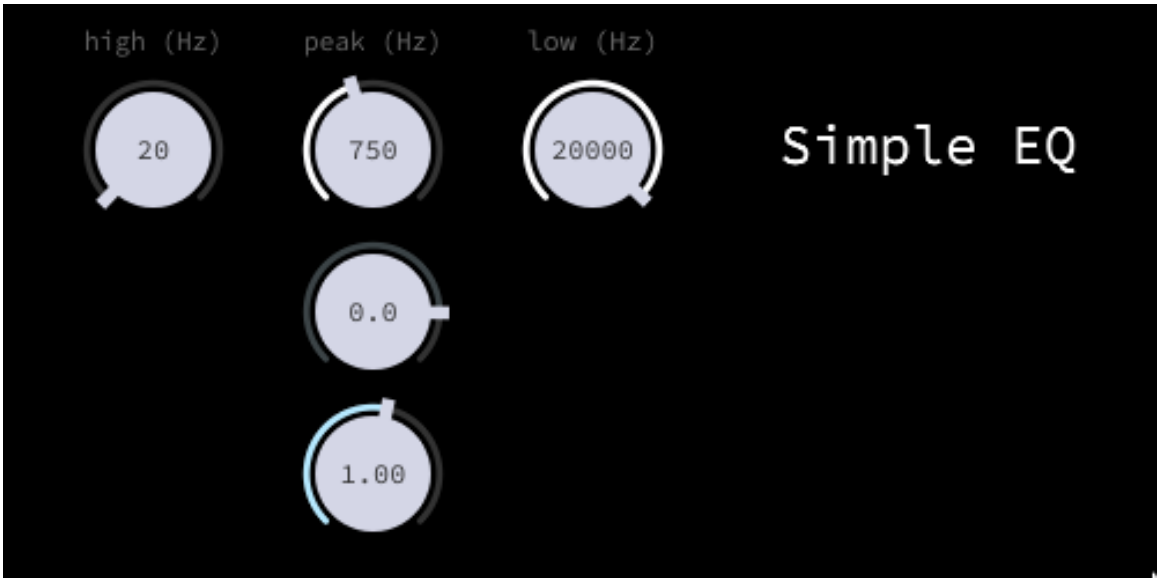
Αξίζει να σημειωθεί πως στον πυρήνα του ένα οποιοδήποτε plugin επεξεργασίας ήχου είναι είναι ένα σύνολο από προσθέσεις, πολλαπλασιασμούς και καθυστερήσεις. Σε γενικές γραμμές, η υλοποίηση των περισσότερων αλγορίθμων δεν είναι αξιοσημείωτη, περά από πιθανές βελτιστοποιήσεις με τη χρήση παραλληλισμού.
Σαφώς, βιβλιοθήκες όπως η JUCE φροντίζουν στην καλύτερη δυνατή υλοποίηση, αφού δεν υπάρχουν κατά το δυνατόν περιτές πράξεις και γίνεται προσπάθεια στη βελτιστοποίηση του κώδικα από τη φάση της μετάφρασης (compilation).

Γραφική Διεπαφή

> The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time
>
> *Tom Cargill, Bell Labs*

Η βιβλιοθήκη JUCE παρέχει τη δυνανότητα να αναπτύξει ο χρήστης σύνθετες γραφικές διεπαφές μέσα από τις οποίες μπορεί κάποιος να ελέγχει το DSP μέρος της εφαρμογής (συχνότητες αποκοπής κτλ).
Έχουμε στη διάθεση μας πολλές επιλογές όσων αφορά την εμφάνιση της εφαρμογής. Η παρουσίαση έχει πολλά να κάνει και με την ευχρηστεία της εφαρμογής. Ελπίζουμε η απλή διεπαφή να συμβάλει σε αυτό το σκοπό.
Είναι άξιο αναφοράς πως σε γραμμές κώδικα, η γραφική διεπαφή χρειάστηκε περισσότερη συγγραφή σε σχέση

με τον επεξεργαστή συμάτων. Αυτό δεν είναι βέβαια σίγουρο αν είναι πρωτέρημα ή όχι της βιβλιοθήκης.



Σχήμα 3.3: Η τελική γραφική διεπαφή του ισοσταθμιστή.

Στην τελική εφαρμογή υπάρχουν πέντε knobs:

- Δύο για τον έλεγχο του υψιπερατού και του χαμηλοπερατού φίλτρου.
- Τρία για την επιλογή της peak συχνότητας, του gain το οποίο θα λάβει και του πάχους της καμπύλης, αντίστοιχα.

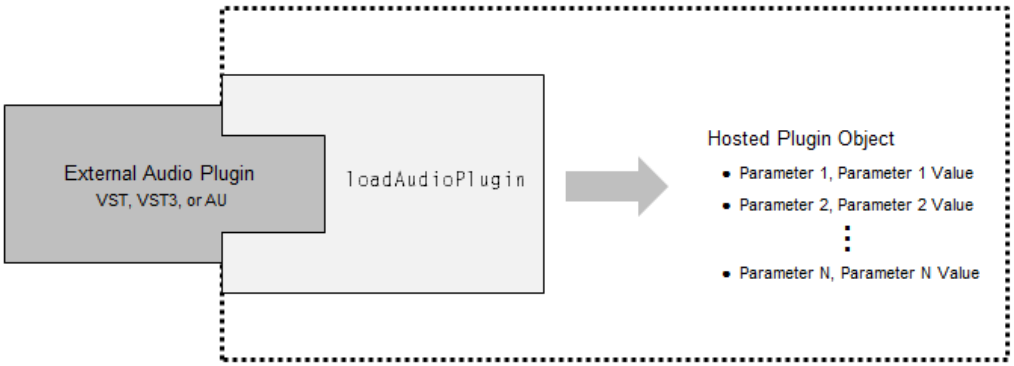
Ο έλεγχος του DSP μέσω της διεπαφής γίνεται μέσα από την ανταλαγή μυνημάτων που προκύπτουν από συμβάντα events [7]. Αυτού του είδους η αρχιτεκτονική είναι συνήθης σε πολλές βιβλιοθήκες ανάπτυξης διεπαφών.

Χρήση

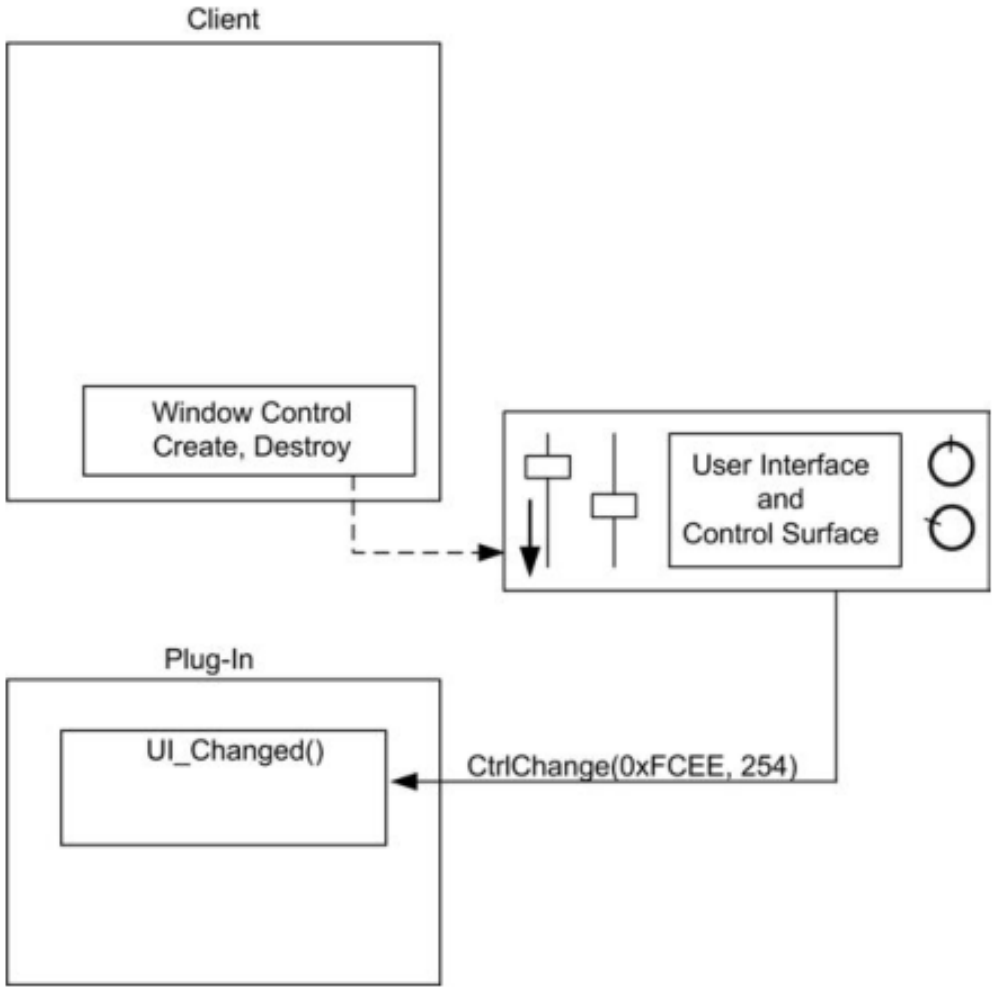
3.1 Audio Plugin Hosts

Πλέον, η χρήση των περισσότερων εφαρμογών DSP γίνεται μέσα από άλλα προγράμματα τα οποία παίζουν το ρόλο του οικοδεσπότη και φιλοξενούν το δικό μας λογισμικό. Η επικοινωνία μεταξύ του plugin μας και του host γίνεται μέσα από ένα API, μέσα από το οποίο έχουν οριστεί οι δομές δεδομένων και οι ρουτίνες με βάση τις οποίες θα γίνει η μεταφορά πληροφορίας μεταξύ των δύο.
Αυτή η επικοινωνία μπορεί να γίνει μέσα από το δυναμικό φόρτωμα κώδικα [8], όπου ο κώδικας του plugin μας φορτώνεται και εκτελείται όσο τρέχει το host πρόγραμμα. [9]
Οι hosts συνοδεύονται συχνά από γραφικές διεπαφές και επιτρέπουν ακόμα το χειρισμό των client προγραμμάτων μέσα από τις δικές του γραφικές διεπαφές. Αυτή η επικοινωνία καθοδηγείται επίσης από ένα API.

Αρχιτεκτονική ήχου σε Linux σύστημα Η ιστορία για οποιδήποτε σύνολο από πρότυπα είναι πάντα περίπλοκη και πολύ συχνά δυσάρεστη. Εδώ θα παραθέσουμε τα κύρια σημεία και θα δείξουμε τον τρόπο με τον οποίο έχουμε στήσει τώρα ένα σύστημα ικανό για περίπλοκη διαχείριση των πηγών ήχων του.
Στο διάγραμμα, αυτό με το οποίο ένας χρήστης θα έρχεται περισσότερο σε επαφή είναι ο διακοσμιτής ήχου (audio server). Αυτός βρίσκεται σε άμεση επικοινωνία με τους drivers οι οποίοι τελικά θα πουν στο υλικό πώς να συμπεριφερθεί ώστε να ακουστεί ήχος από μία συσκευή. Ο διακοσμιτής δέχεται δεδομένα ήχου από πολλές διαφορετικές εφαρμογές και βγάζει μία ενιαία σειρά από δεδομένα. Αυτό μας επιτρέπει να ακούμε πολλές

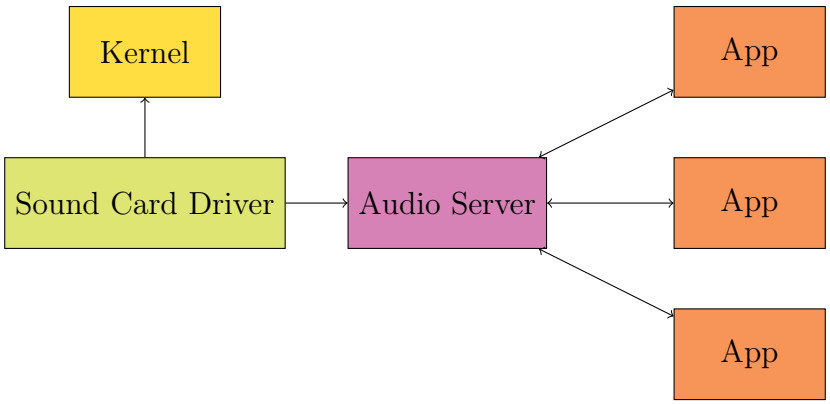


Σχήμα 3.4: Διάγραμμα ενός audio plugin host.



Σχήμα 3.5: Περίπτωση όπου ο host ελέγχει το UI το οποίο με τη σειρά του ελέγχει το plugin

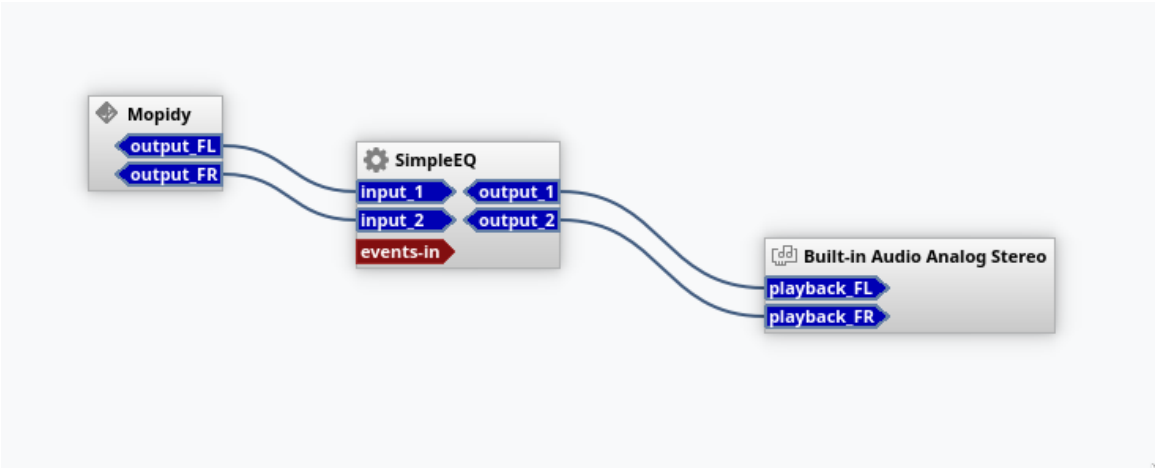
ερφαρμογές ταυτόχρονα παιρνόντας την καθεμία από το κανάλι ενός εικονικού mixer. Φυσικά, θα μπορούσαμε να μην έχουμε έναν audio server και να επικοινωνούσε η οποιαδήποτε διεργασία μας που παρήγαγε ήχο με τον driver. Αυτό θα σήμαινε όμως πως μόνο μία διεργασία θα μπορούσε να μιλήσει με την κάρτα ήχου ανά πάσα στιγμή, δεδομένου το πώς είναι δομημένο το audio backend στο σύστημά μας. Πλέον το πιο συνήθες σύστημα για audio driver είναι το `ALSA`. [10] Τώρα, για τον audio server υπάρχουν διάφορες επιλογές. Για πολύ καιρό η βασική επιλογή ήταν το PulseAudio [11], το οποίο όμως ήταν αδύναμο σε εφαρμογές χαμηλής καθυστέρησης



Σχήμα 3.6: Στοίβα τεχνολογιών ήχου σε ένα Linux σύστημα.

όπως είναι η παραγωγή μουσικής. Σε αυτές τις περιπτώσεις χρησιμοποιούταν ο διακοσμιτής JACK [12]. Έτσι, υπάρχει πλέον προσπάθεια δημιουργίας μίας νέα υλοποίησης η οποία θα συμπεριλαμβάνει τα θετικά και των δύο audio servers. Αυτό είναι το Pipewire [13] και δίνει τη δυνατότητα εύκολης επικοινωνίας μεταξύ όλων των συσκευών και διεργασιών του συστήματος, ακόμα και σε περιβάλλοντα υψηλών επιδόσεων. Η μηχανή του Pipewire γίνεται σταδιακά ευρέως αποδεκτή στην κοινότητα ακόμα και από τους προγραμματιστές των 'αντίπαλων' audio servers. [14]

Carla Για την χρήση του plugin `Simple EQ` μπορεί να γίνει σύνδεσή του με άλλες ροές ήχου στον υπολογιστή μέσω ενός audio plugin host. Ένα πολύ ικανό και δημοφιλές είναι το λογισμικό Carla [15], μέσα από το οποίο μπορούμε να κατευθύνουμε τη ροή ήχου από μία εφαρμογή στον υπολογιστή μας, μέσα από κάποιο plugin και τελικά να έχουμε την έξοδο στα ηχεία (ή οποιαδήποτε άλλη συσκευή).
Για να γίνει πιο κατανοητός ο τρόπος με τον οποίο το λογισμικό Carla παίρνει θέση ανάμεσα στις άλλες τεχνολογίες που χρησιμοποιούμε, μπορούμε να το φανταστούμε σαν ένα μεσάζοντα μεταξύ του audio server (Pipewire) και του plugin μας.
Η εφαρμογή Carla 'κουμπώνει' στον audio server ή στον εγχώριο sound driver. Έτσι, μπορεί να ελέγχει την πορεία των σημάτων ήχου. Στην γραφική της διεπαφή το μπλε χρώμα υποδηλώνει μία είσοδο/έξοδο audio, ενώ το κόκκινο χρώμα αντιστοιχεί σε μία είσοδο/έξοδο που χειρίζεται συμβάντα. Αυτά συνήθως αποτελούν MIDI events.



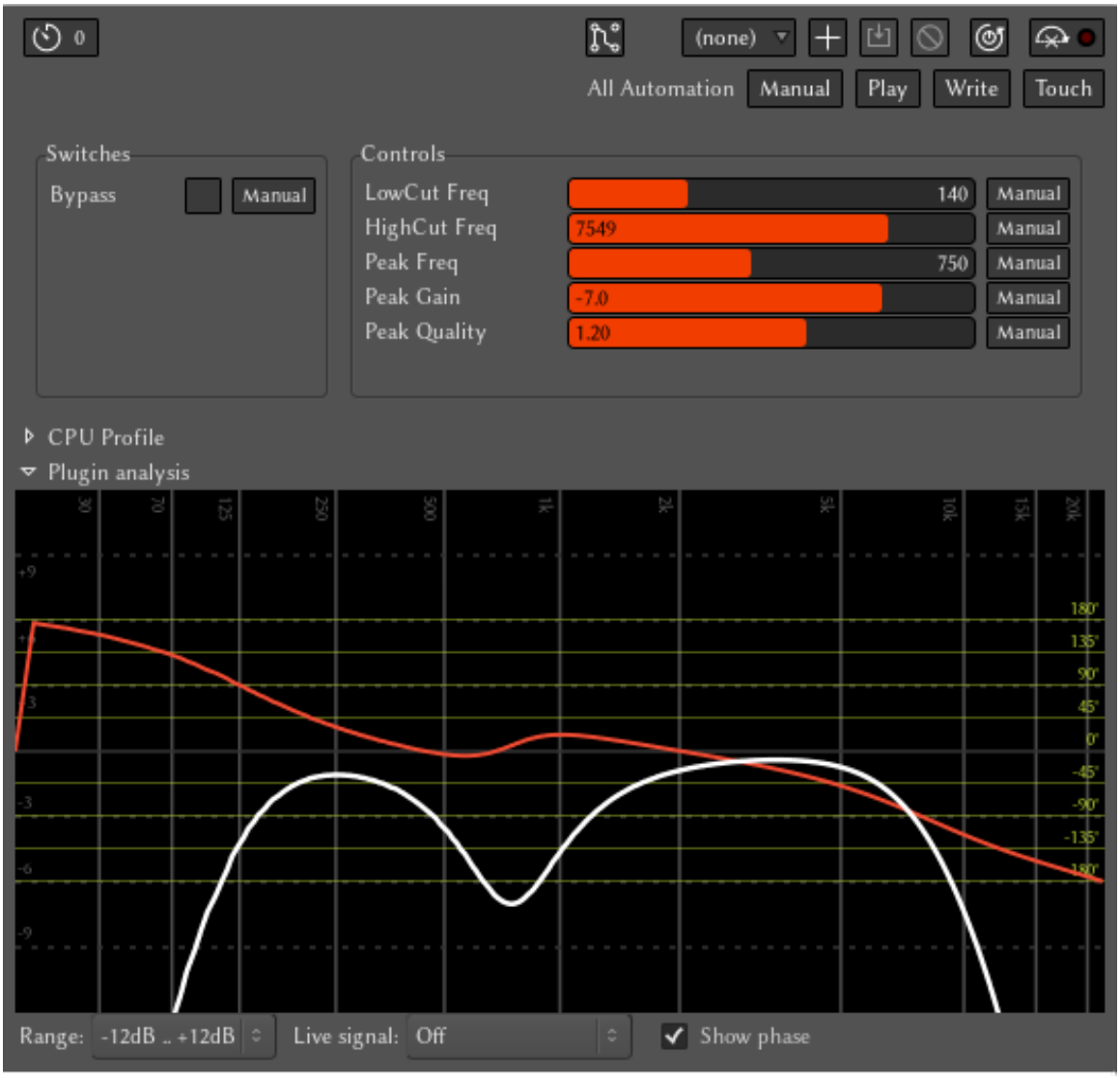
Σχήμα 3.7: Εφαρμογή του `SimpleEq plugin` στο περιβάλλον Carla.

Digital Audio Workstation Συνήθης είναι επίσης η χρήση του plugin μέσα από ένα λογισμικό DAW (Digital Audio Workstation), όπου μπορούμε να φορτώσουμε δυναμικά το `Simple EQ`, να προσθέσουμε σε ένα από τα κομμάτια του mixer και έτσι όποιος ήχος περνάει σε εκείνο το κανάλι θα επεξεργάζεται από τον plugin.

Η βιβλιοθήκη `JUCE` προσφέρει συστήματα για το χτήσιμο (build) του κώδικα ως plugin (VST3) [16], ή ως αυτόνομη εφαρμογή.

4 Μετρήσεις

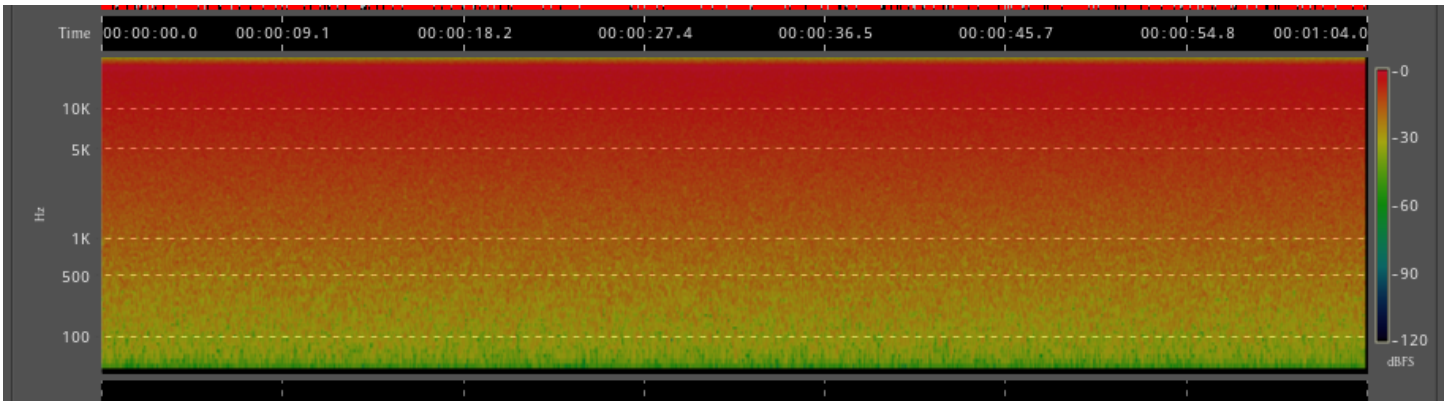
Μετά τη δημιουργία της εφαρμογής ισοστάθμισης, έγινε εισαγωγή του plugin `SimpleEQ` στο ψηφιακό περιβάλλον επεξεργασίας ήχου (DAW) Ardour, το οποίο λειτουργεί σαν υποδοχέας και παρέχει δυνατότητες μέτρησης της απόκρισης συχνότας και φάσης του φίλτρου (φιγ. 4.1).



Σχήμα 4.1: Ανάλυση του `SimpleEQ` μέσα από το Ardour. Φαίνεται η απόκριση συχνότητας (άσπρο) και η απόκριση φάσης (κόκκινο).

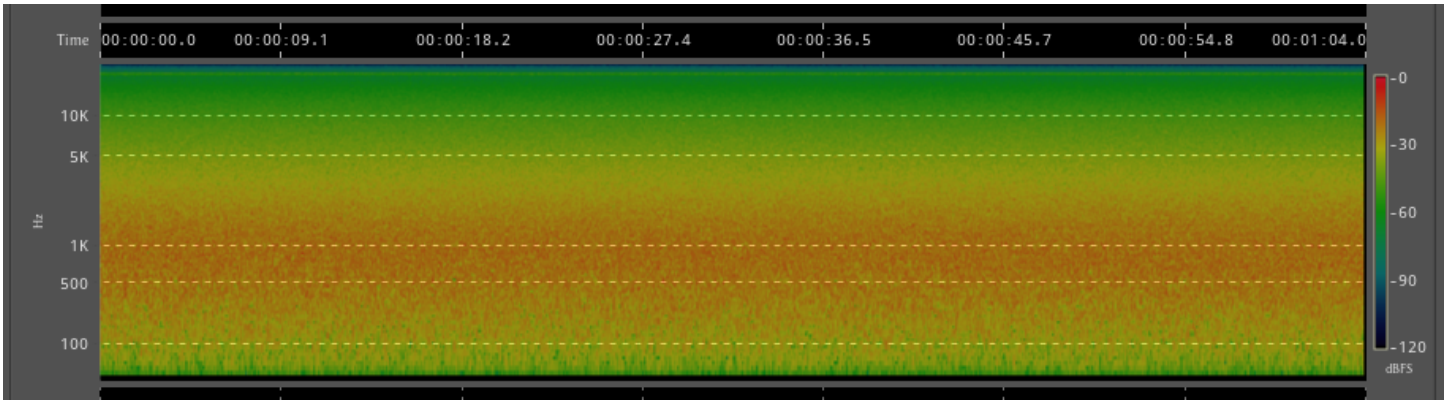
Γίνεται φανερό από την απόκριση του φίλτρου πως η αποκοπή είναι 'μαλακή' (soft-cut).

Μετρήσεις με White Noise Έπειτα, εφαρμόσαμε λευκό θόρυβο και αναλύσαμε το τελικό αρχείο ήχου.



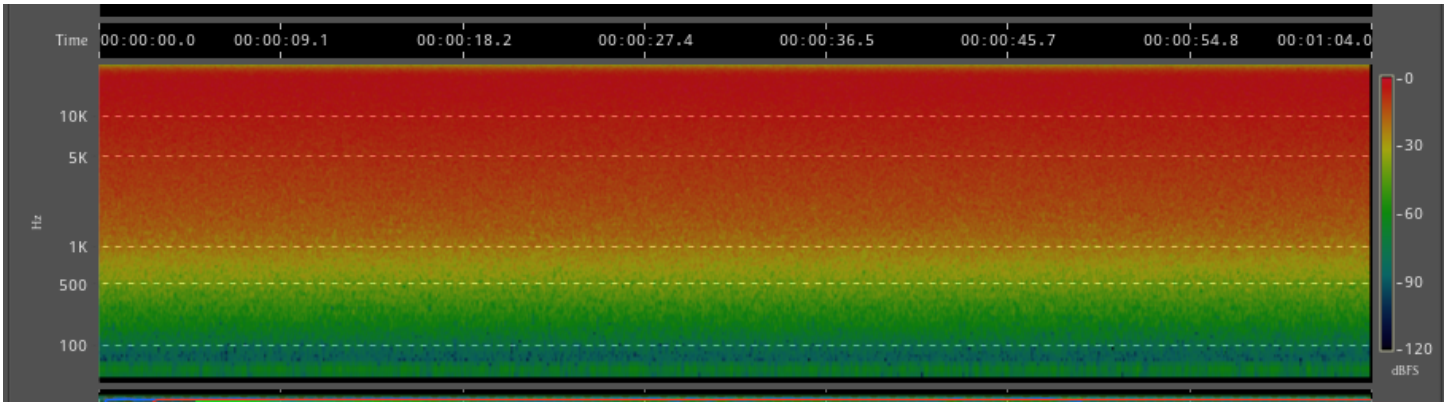
(α΄) Φασματογράφημα του τελικού αρχείου.

Σχήμα 4.2: Το σήμα λευκού θορύβου χωρίς φίλτρο.



(α΄) Φασματογράφημα του τελικού αρχείου.

Σχήμα 4.3: Το σήμα λευκού θορύβου με εφαρμογή χαμηλοπερατού φίλτρου στα $1000Hz$.



(α΄) Φασματογράφημα του τελικού αρχείου.

Σχήμα 4.4: Το σήμα λευκού θορύβου με εφαρμογή υψιπερατού φίλτρου στα $1000Hz$.

Μία παρατήρηση είναι πως στην περίπτωση του υψιπερατού φίλτρου, οι υψηλές συχνότητες φαίνεται να ενισχύονται παραπάνω από τα αρχικά τους επίπεδα. Αυτό πιθανότατα να οφείλεται στο αναπόφευκτο phase shift που προκαλεί το plugin το οποίο προκαλεί συχνότητες να συγχρονιστούν σε φάση οδηγώντας σε διακρότημα και έτσι ενίσχυσή τους.

Αναφορές

[1] LSP Plugin, 'LSP Parametric Equalizer.'

[2] Calf Studio, 'Calf Studio Gear.'

[3] Openheim, Alan, *Discrete Time Signal Processing Third Edition.* Upper Saddle River, NJ: Pearson Higher Education, Inc., 2010.

[4] Nigel Redmon, 'The bilinear z transform,' 2003.

[5] JUCE Team, *JUCE Framework Documentation.* JUCE Foundation.

[6] cppreference.com, 'Placeholder type specifiers.'

[7] Wikipedia, 'Event-driven architecture,'

[8] Wikipedia, 'Dynamic loading,'

[9] Will C. Pirkle, *Designing Audio Effect PLuings in C++.* Focal Press, 2019.

[10] ALSA Team, 'Advanced Linux Sound Architecture (ALSA).'

[11] PulseAudio Team, 'PulseAudio.'

[12] JACK Team, 'JACK Audio Connection Kit.'

[13] Pipewire Team, 'Pipewire.'

[14] Darwish, Ahmed S., 'PipeWire: The Linux audio/video bus,'

[15] KXStudio, 'Applications - Carla.'

[16] Wikipedia, 'Virtual Studio Technology,'