ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ
ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ:

ΣΧΕΔΙΑΣΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΗΧΗΤΙΚΩΝ ΕΦΕ ΣΕ FPGA

Παναγιώτης Πετρόπουλος

Εξεταστική επιτροπή:
Πνευματικάτος Διονύσιος (επιβλέπων, καθηγητής),
Παπαευσταθίου Ιωάννης (αναπληρωτής καθηγητής),
Ποταμιάνος Αλέξανδρος (αναπληρωτής καθηγητής)

Χανιά, Οκτώβριος 2012

Στον πατέρα μου, τη μητέρα μου, το Γιώργο
...και τη γιαγιά.

Στον Αυγουστίνο, το Γιάννη, το Μιχάλη, το Νίκο
...και το Μήτσο από την επαρχία

Στην Τίνα

Σε όλους μου τους συντρόφους

Περίληψη

Η παρούσα διπλωματική εργασία είχε ως στόχο να δημιουργήσει ένα standalone audio effect unit, δηλαδή μια μονάδα ψηφιακής επεξεργασίας σήματος ήχου που μετασχηματίζει την είσοδό της με βάση τους αλγορίθμους διαφόρων ηχητικών εφέ.

Η επεξεργασία,η μοντελοποίηση και η μελέτη των αλγορίθμων έγινε σε περιβάλλον MATLAB. Αργότερα, και με χρήση του εργαλείου της Xilinx “System Generator for DSP” καταφέραμε και υλοποιήσαμε τους αλγορίθμους σε block diagrams και τους προσωμειώσαμε στο περιβάλλον του Simulink, ακούγοντας τα αποτελέσματα. Στη συνέχεια και με τη βοήθεια του ίδιου εργαλείου παράχθηκε ο κώδικας VHDL για τα συγκεκριμένα εφέ. Με χρήση αυτού του κώδικα εν τέλει προγραμματίσαμε επιτυχώς μία XUP Virtex-II Pro FPGA, έτσι ώστε να τρέχουν αυτά τα εφέ σε real-time, κάνοντας έτσι αυτή την FPGA να λειτουργεί σαν standalone audio effect unit. Θέτοντας σε λειτουργία το AC97 codec που έχει στη διάθεσή της η Virtex-II Pro, δέχεται αναλογική είσοδο σήματος ήχου και στέλνει την ίδια στιγμή στην έξοδο το σήμα μετασχηματισμένο με βάση το εφέ που έχουμε επιλέξει.

Τα ηχητικά εφέ που υλοποιήσαμε είναι τα εξής: Distortion – fuzz, echo, reverb, vibrato, flanger, equalizer, wah, και pitch shifting.

Abstract

The present final undergraduate thesis aims to create a standalone audio effect unit, thus a digital signal processing unit which transforms its audio input based in some audio effects algorithms.

The processing, modeling, and design of these algorithms was done in MATLAB environment. Later, using the powerful tool “Xilinx System Generator for DSP”, we implemented those algorithms in block diagrams, simulating in the environment of Simulink, listening to the results. Then, using the same tool we generated the VHDL code of those effects. The code eventually helped us in programming a XUP Virtex-II Pro FPGA, in order to run these affect in real-time, making this FPGA to run as a standalone audio effect unit. Putting into operation the embedded AC97 codec available in Virtex-II Pro, it accepts analog input audio signal and at the same time it sends the output signal transformed by the effect we have chosen.

The implemented audio effects are: Distortion – fuzz, echo, reverb, vibrato, flanger, equalizer, wah, and pitch shifting.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κ. Πνευματικάτο για τη συνεργασία και τη συνεννόηση που είχαμε όλο το διάστημα της εκπόνησης αυτής της διπλωματικής εργασίας. Επίσης ευχαριστώ θερμά το Γρηγόρη Χρυσό που ήταν εκεί όσες φορές χρειάστηκα τη βοήθειά του. Χρωστάω επίσης ένα μεγάλο ευχαριστώ στους φίλους μου Αλέξανδρο Καρατζαφέρη και Θάνο Σαρίγγελο για την έμπνευση που μου έδωσαν και σκέφτηκα το παρόν θέμα διπλωματικής εργασίας. Επίσης ένα μεγάλο "gracias" στον Jaime Laino από την Κολομβία, που με βοήθησε πολύ ακόμα και από τόσο μεγάλη απόσταση, πιστός στη γνώμη του "after all we are all here to share knowledge".

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1
Εισαγωγή στην παρούσα διπλωματική εργασία

Ο ήχος και η αναπαραγωγή του είναι κομμάτι της καθημερινής μας ζωής. Σημαντικότατες ανθρώπινες δραστηριότητες όπως η επικοινωνία και η ψυχαγωγία είναι συνδεδεμένες με τη διάδοση του ήχου και με την επεξεργασία του.

Για πρώτη φορά η αναπαράσταση και η καταγραφή σημάτων ήχου έγινε το 1876 από τον Alexander Graham Bell, για την εφεύρεση του ακουστικού τηλεφώνου, μια εφεύρεση τεράστιας σημασίας για την ανθρωπότητα. Έκτοτε, και κατά τη διάρκεια του εικοστού αιώνα υπήρξε σημαντική πρόοδος σχετικά με την επεξεργασία του ήχου και της μετάδοσής του κυρίως στον τομέα των τηλεπικοινωνιών και της μουσικής βιομηχανίας.

Ενώ στα μέσα του εικοστού αιώνα τα πάντα σχετικά με τον ήχο γίνονταν με αναλογικά μέσα, προς το τέλος του ίδιου αιώνα, η μετάβαση από το αναλογικό στο ψηφιακό σήμα οδήγησε αυτόν τον κλάδο σε ραγδαία ανάπτυξη, με τη χρήση ψηφιακών πλέον συστημάτων. Ο κλάδος αυτός ονομάστηκε ψηφιακή επεξεργασία σήματος.

Στον τομέα της παραγωγής μουσικής διάφοροι καλλιτέχνες, παραγωγοί και μουσικοί έχουν σκεφτεί καινοτόμες ιδέες για το πώς θα επεξεργαστούν τον ήχο που παράγεται από μία φυσική πηγή, όπως ένα μουσικό όργανο ή μια ανθρώπινη φωνή, έτσι ώστε να του δώσουν διαφορετική μορφή και χροιά ανάλογα με το τι επιθυμούν.

Ένα τέτοιο ψηφιακό σύστημα επεξεργασίας ήχου είναι και ο στόχος της παρούσας διπλωματικής εργασίας. Θέλαμε να δεχόμαστε σαν είσοδο ένα οποιοδήποτε σήμα ήχου, και να το μετασχηματίζουμε στην έξοδο με βάση κάποια γνωστά ηχητικά εφέ που είναι γνωστά στη μουσική -και όχι μόνο- βιομηχανία. Αυτό γίνεται εφικτό με την κατασκευή και τον προγραμματισμό ενός standalone audio effect unit. Το ρόλο αυτής της μονάδας αναλαμβάνει μία FPGA, την οποία και προγραμματίσαμε κατάλληλα.

1.1. Τα ηχητικά εφέ

Τα ηχητικά εφέ προκύπτουν από συστήματα τα οποία αλλάζουν τον τρόπο με τον οποίο μπορεί και ακούγεται ένα μουσικό όργανο ή κάποια άλλη πηγή ήχου. Μερικά εφέ αλλάζουν διακριτικά και "χρωματίζουν" το σήμα ήχου, κάποια άλλα το αλλάζουν δραματικά. Τα ηχητικά εφέ χρησιμοποιούνται κατά κόρον στην μουσική, από μουσικούς αλλά και από παραγωγούς μουσικής σε στούντιο ηχογραφήσεων.

Τα πρώτα ηχητικά εφέ παράγονταν αποκλειστικά σε στούντιο ηχογραφήσεων. Γύρω στα μέσα με τέλη της δεκαετίας του 40, πολλοί μουσικοί όπως ο Les Paul, πειραματίζονταν με την τεχνική ηχογράφησης με δύο μπομπίνες (reel to reel recording tape) για να δημιουργήσουν διάφορους "φουτουριστικούς ήχους". Η τεχνική τοποθέτησης μικροφώνων σε έναν ακουστικό χώρο με διάφορους τρόπους για να προσωμοιώσει την "ηχώ", χρησιμοποιούνταν επίσης συχνά. Στις αρχές της δεκαετίας του 50' σιγά σιγά άρχισαν να κατασκευάζονται μεμονωμένες μονάδες ηχητικών εφέ (standalone effects units), ή πολλές φορές τα εφέ αυτά ενσωματώνονταν μέσα σε ενισχυτές μουσικών οργάνων. Οι εταιρίες που καθιέρωσαν τα ηχητικά εφέ στην παραγωγή της μουσικής ήταν κυρίως η Fender και η Gibson. Από τις αρχές της δεκαετίας του 50' και μετά, καθιερώθηκε η μαζική χρήση πολλών και διαφόρων ηχητικών εφέ.

Κάποια από τα πιο γνωστά ηχητικά εφέ, υλοποιήσαμε στην παρούσα εργασία:

Εφέ απλών μαθηματικών πράξεων:

- **Distortion ή fuzz**. Κάνει τον ήχο να ακούγεται πιο "βρώμικος".

Εφέ βασισμένα στην καθυστέρηση:

- **Το εφέ της ηχούς (echo).** Ο ήχος ακολουθείται από επαναλήψεις του εαυτού του, όπως ακριβώς συμβαίνει με την "ηχώ" του βουνού.
- **Το εφέ της αντήχησης (reverb).** Οι επαναλήψεις του ήχου είναι σύντομες και παραπάνω της μίας, όπως ακριβώς σε ένα μεγάλο δωμάτιο ή ένα χωλ.
- **Το εφέ vibrato.** Κάνει τον ήχο να ακούγεται "τρεμάμενος", με συνεχείς διακυμάνσεις του τόνου (pitch) του ήχου, βασισμένες στο doppler effect.
- **Το εφέ flanger.** Κάνει τον ήχο να ακούγεται σαν από αεροπλάνο που απογειώνεται (!).

Εφέ βασισμένα στα FIR φίλτρα:

- **Ένα απλό equalizer.** Μπορεί με την απλή ρύθμιση κάποιων παραμέτρων να αυξομειώσει τα μπάσα, τα μεσαία, και τα πρίμα του ήχου, κάνοντάς τον να ακουστεί ανάλογα.
- **Το wah effect.** Κάνει τον ήχο να ακουστεί όπως μία ανθρώπινη φωνή που προφέρει τη λέξη "wah".

Υπόλοιπα εφέ:

- **Pitch shifting.** Αλλάζει τον τόνο του ήχου (pitch) κάνοντας τον πιο "λεπτό" ή πιο "χοντρό", χωρίς όμως να αλλάζει τη διάρκειά του.

Εικόνα 1.1: Μονάδες ηχητικών εφέ για ηλεκτρική κιθάρα, συνδεδεμένες σε σειρά.

1.2. Η μεθοδολογία της υλοποίησης

Κατά την εκπόνηση αυτής της διπλωματικής εργασίας, ακολουθήσαμε συγκεκριμένα βήματα.

Στην αρχή έγινε μια γενική μελέτη για το ποια είναι τα ηχητικά εφέ που χρησιμοποιούνται στην παραγωγή μουσικής και αλλού, και επιλέξαμε τα πιο σημαντικά. Τα χωρίσαμε σε κατηγορίες ανάλογα με το αλγοριθμικό τους περιεχόμενο (απλές μαθηματικές πράξεις, εφέ βασισμένα στην καθυστέρηση, εφέ με χρήση φίλτρων και άλλα), και μελετήσαμε σε βάθος τους αλγορίθμους αυτών των εφέ στο πρόγραμμα MATLAB, ταυτόχρονα με την ανάγνωση συγκεκριμένης βιβλιογραφίας για την εκμάθηση τους και τη σε βάθος κατανόηση.

Έπειτα και σε επόμενο στάδιο, κάναμε μία μελέτη σχετικά με το πώς όλα αυτά μπορούν να υλοποιηθούν σε υλικό. Η ειδοποιός διαφορά σε σχέση με το περιβάλλον της MATLAB είναι ότι πλέον η ροή των δεδομένων είναι συνεχής, ότι δηλαδή δεν είναι γνωστό το μήκος του σήματος εισόδου. Αυτό στα περισσότερα από τα εφέ μας οδήγησε σε υλοποιήσεις εντελώς διαφορετικής “φιλοσοφίας”.

Για να γίνει λοιπόν περισσότερο αντιληπτή αυτή η φιλοσοφία, έπρεπε να είναι κάπως πιο “σχηματική” η αναπαράσταση της σχεδίασης σε υλικό. Σε συνδιασμό λοιπόν με το πρόβλημα της δοκιμής που αντιμετωπίσαμε, χρησιμοποιήσαμε το εργαλείο της Xilinx “System Generator for DSP” για να αναπαραστήσουμε και να προσωμοιώσουμε τη σχεδίασή μας. Το εργαλείο αυτό μας έδωσε επίσης τη δυνατότητα με βάση την αναπαράσταση μιας σχεδίασης σε block diagram να δημιουργηθεί και ο αντίστοιχος κώδικας σε γλώσσα περιγραφής υλικού, VHDL.

Τέλος, χρειάστηκε να χρησιμοποιήσουμε μία FPGA XUP Virtex-II Pro της εταιρίας Xilinx για να εφαρμόσουμε ότι έχουμε κάνει. Με την ανάγνωση χρήσιμης βιβλιογραφίας σε σχέση με τη φύση των FPGA και της λειτουργίας, του προγραμματισμού και της χρήσης της Virtex II Pro για αναπαραγωγή και επεξεργασία σημάτων ήχου, μπορέσαμε και υλοποιήσαμε τη σχεδίασή μας για όλα τα εφέ έτσι ώστε να “τρέχουν” σε πραγματικό χρόνο (real-time) πάνω στην FPGA.

1.3. Περιληπτική περιγραφή του κάθε κεφαλαίου

- **Κεφάλαιο 2:** Περιγράφει τις βασικές γνώσεις που χρειάζεται κανείς να έχει για το θέμα της ψηφιακής επεξεργασίας σήματος. Περιγράφει το πώς γίνεται η μετάβαση από ένα αναλογικό σε ένα ψηφιακό σήμα, καθώς και τα πλεονεκτήματα της ψηφιακής από την αναλογική επεξεργασία.

- **Κεφάλαιο 3:** Αποτελεί μία εισαγωγή στις FPGA ως τύπο ολοκληρωμένου κυκλώματος, την ιστορία τους και την αρχιτεκτονική τους. Κάνει μία σύντομη περιγραφή στην XUP Virtex-II Pro που χρησιμοποιήσαμε, καθώς και στον τρόπο προγραμματισμού της.

- **Κεφάλαιο 4:** Περιγράφει το εργαλείο "Xilinx System Generator for DSP" που χρησιμοποιήσαμε, καθώς και τον τρόπο λειτουργίας, τις δυνατότητες, τα πλεονεκτήματα και τους λόγους που το προτιμήσαμε.

- **Κεφάλαιο 5:** Η κύρια δουλειά της διπλωματικής εργασίας. Περιέχει περιγραφή όλων των εφέ που υλοποιήθηκαν καθώς και το υπόβαθρο γνώσεων που χρειάζεται για την κατανόησή τους, αναλυτική εξήγηση του αλγορίθμου τους σε MATLAB και αναλυτική περιγραφή της υλοποίησης και σχεδίασής τους σε υλικό, και τους πόρους που αυτά καταλαμβάνουν.

- **Κεφάλαιο 6:** Περιγράφεται αναλυτικά ο τρόπος με τον οποίο αυτά εφαρμόστηκαν στην FPGA. Αναλύει το πώς εφαρμόσαμε το interface ανάμεσα στον AC97' codec που διαθέτει η πλακέτα για ανάγνωση και αναπαραγωγή ηχητικών σημάτων. Αναλύεται βήμα προς βήμα πώς πήγαμε από τη σχεδίαση στη netlist και αργότερα στο .bit αρχείο με το οποίο προγραμματίσαμε την FPGA.

ΚΕΦΑΛΑΙΟ 2

Ψηφιακή Επεξεργασία Σήματος

2.1. Εισαγωγή

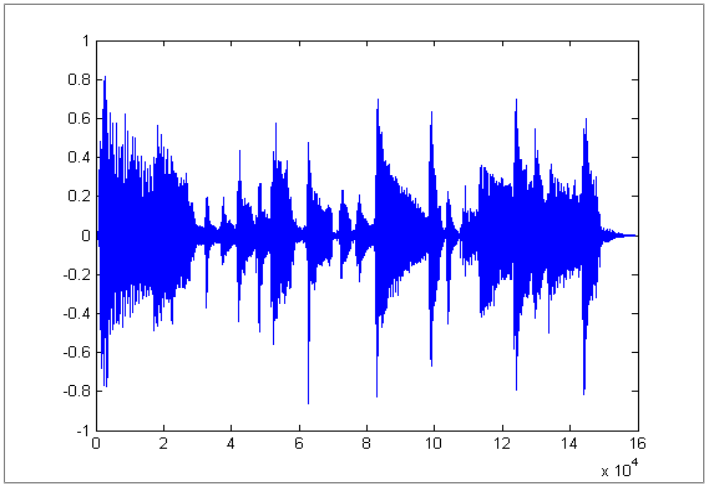
Η ψηφιακή επεξεργασία σήματος (Digital Signal Processing, DSP) είναι το αντικείμενο εκείνο που ασχολείται με την αναπαράσταση σημάτων διακριτού χρόνου, καθώς και με την επεξεργασία τους.

Ως σήμα μπορούμε να θεωρήσουμε το σύνολο τιμών που μπορεί να λάβει μια φυσική ποσότητα, όταν αυτό εκφράζεται ως συνάρτηση ή ως ακολουθία μιας ή περισσοτέρων ανεξάρτητων μεταβλητών. Τα σήματα αυτά περιέχουν κάποιο είδος πληροφορίας σχετικά με τη συμπεριφορά, την έκταση, ή τη φύση ενός μετρήσιμου φαινομένου.

Όταν λέμε επεξεργασία, εννοούμε την κατά οποιονδήποτε τρόπο επίδραση πάνω στο σήμα, και πιθανά την εξαγωγή χρήσιμης πληροφορίας. Η ψηφιακή επεξεργασία προϋποθέτει ότι το αρχικό μας σήμα, ανεξάρτητα από την αρχική του υπόσταση, πρέπει να έχει μετατραπεί σε μία ακολουθία αριθμών. Η ψηφιακή επεξεργασία σήματος χρησιμοποιεί αριθμούς συγκεκριμένης βάσης και μήκους κάθε φορά για την αναπαράσταση των προς επεξεργασία αριθμών. Η βάση είναι σχεδόν πάντα το 2, που σημαίνει ότι τα σύμβολα που είναι δυνατόν να χρησιμοποιηθούν είναι το 0 και το 1.

Η εντυπωσιακή ανάπτυξη των υπολογιστών και της μικροηλεκτρονικής τα τελευταία χρόνια, είχε καθοριστική επίδραση στην ψηφιακή επεξεργασία σημάτων, με συνεχή τάση μείωσης του κόστους και καθώς και τη συνεχή αύξηση της αξιοπιστίας όλων των ψηφιακών συστημάτων. Οι εφαρμογές της ψηφιακής επεξεργασίας σήματος καλύπτουν πολλούς τομείς της σύγχρονης δραστηριότητας.

Κάποια πεδία της ψηφιακής επεξεργασίας σήματος είναι η επεξεργασία σημάτων ήχου, η επεξεργασία σημάτων φωνής, η ψηφιακή επεξεργασία εικόνας, η επεξεργασία σήματος στις τηλεπικοινωνίες, ο έλεγχος συστημάτων και ο αυτόματος έλεγχος, η επεξεργασία βιοϊατρικών σημάτων, η επεξεργασία σεισμικών δεδομένων και η γεωφυσική.



Εικόνα 2.1. (αριστερά) Ένα τσιπ ψηφιακής επεξεργασίας σήματος ήχου ενσωματωμένο σε μία μονάδα ηχητικών εφέ.

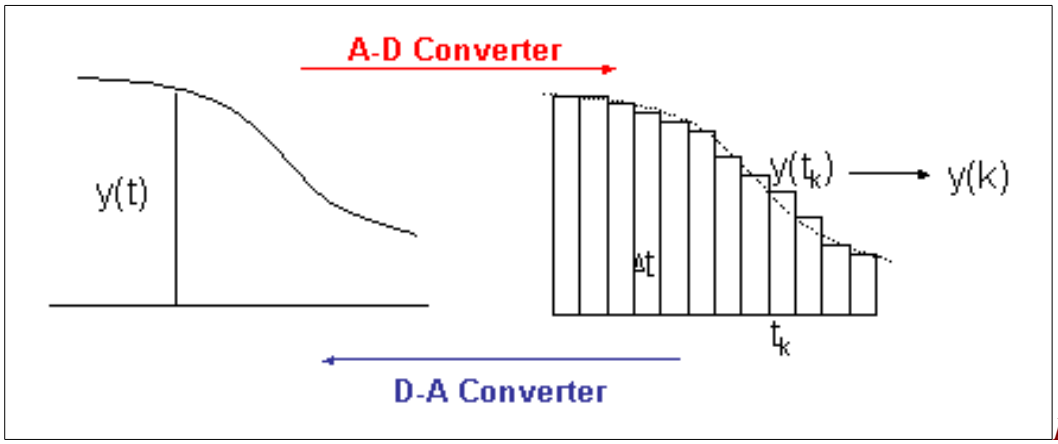
Εικόνα 2.2. (δεξιά) Η γραφική αναπαράσταση ενός ψηφιακού σήματος ήχου στο πρόγραμμα MATLAB.

2.2. Η μετατροπή αναλογικού σε ψηφιακό

Εικόνα 2.3 : Η διαδικασία μετατροπής ενός αναλογικού σήματος σε ψηφιακό.

Τα περισσότερα σήματα στον φυσικό κόσμο υπάρχουν σε αναλογική μορφή. Ένα αναλογικό σήμα είναι συνεχές, δηλαδή λαμβάνει συνεχείς τιμές. Όταν συμβαίνουν διάφορα φυσικά φαινόμενα, παράγεται πληροφορία η οποία μπορεί να καταγραφεί σε ένα αναλογικό σήμα. Τέτοια πληροφορία μπορεί να είναι η ένταση του ήχου, η ένταση του φωτός, η διαφορά δυναμικού, η ατμοσφαιρική πίεση, η ταχύτητα και άλλα.

Για να μπορέσει να πραγματοποιηθεί η επεξεργασία όλων αυτών των σημάτων ψηφιακά, υπάρχει η ανάγκη ενός interface (διεπαφής) μεταξύ του αναλογικού σήματος και του ψηφιακού επεξεργαστή. Αυτό το interface ονομάζεται μετατροπέας αναλογικού σήματος σε ψηφιακό (A/D converter). Ο ψηφιακός επεξεργαστής σήματος μπορεί να είναι ένας προγραμματιζόμενος υπολογιστής ή ένας μικροεπεξεργαστής ο οποίος είναι προγραμματισμένος να εκτελέσει κάποιες λειτουργίες πάνω στο σήμα εισόδου. Αν στην έξοδο του ψηφιακού επεξεργαστή επιθυμώ πάλι ένα αναλογικό σήμα, χρησιμοποιώ ένα αντίστοιχο interface που είναι ο μετατροπέας ψηφιακού σήματος σε αναλογικό (D/A converter). Ωστόσο, υπάρχουν και διάφορες εφαρμογές που δεν απαιτούν μετατροπέα D/A στην έξοδο (όπως πχ. η ένδειξη της θέσης ενός αεροπλάνου σε ραντάρ).

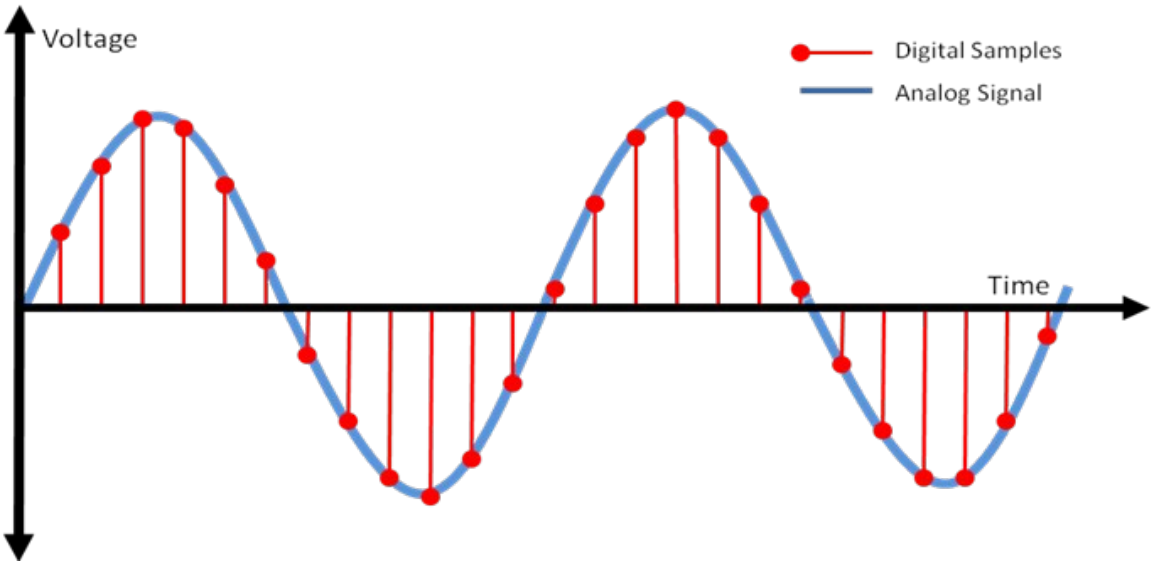


Εικόνα 2.4: Γραφική αναπαράσταση μετατροπής αναλογικού σε ψηφιακό.

Δειγματοληψία

Αξίζει σε αυτό το σημείο να γίνει μια σύντομη αναφορά στον τρόπο με τον οποίο μπορούμε να μεταφέρουμε ένα σήμα από αναλογική σε ψηφιακή μορφή. Η μετατροπή ενός αναλογικού σήματος σε ψηφιακό περνά από τρία βασικά στάδια, αυτό της δειγματοληψίας (sampling), του κβαντισμού (quantization), και της κωδικοποίησης (coding).

Δειγματοληψία είναι η τεχνική της λήψης τιμών του σήματος (δείγματα - samples) σε επιλεγμένες χρονικές στιγμές. Στην περίπτωση της ομοιόμορφης δειγματοληψίας (uniform sampling) οι χρονικές αυτές στιγμές ισαπέχουν μεταξύ τους. Η χρονική αυτή απόσταση που έχουν δύο διαδοχικά δείγματα μεταξύ τους ονομάζεται περίοδος δειγματοληψίας Ts. Από τη διαδικασία της δειγματοληψίας απαιτούμε να χάνεται η λιγότερη δυνατή πληροφορία από το σήμα συνεχούς χρόνου. Αυτό επιτυγχάνεται με όσο το δυνατόν μικρότερη τιμή του Ts. Με τον όρο συχνότητα δειγματοληψίας (Fs = 1/Ts) εννοούμε το πόσα δείγματα θα έχω εκλάβει σε ένα sec, κατά τη διάρκεια της δειγματοληψίας.

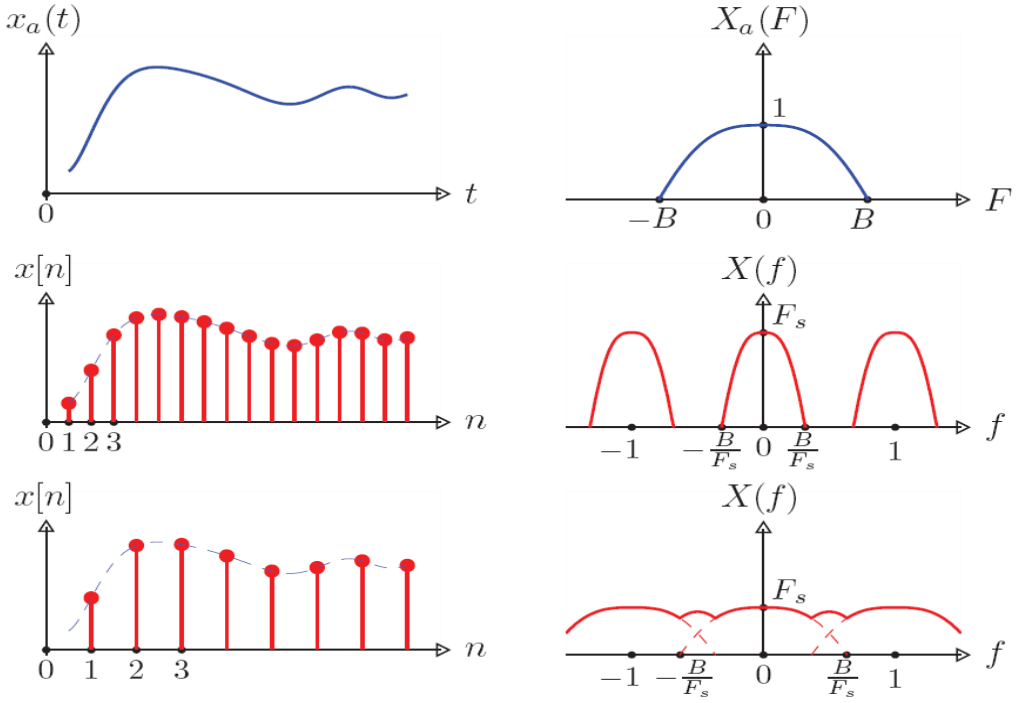


Εικόνα 2.5: Παράδειγμα δειγματοληψίας ενός ημιτόνου.

Το θεώρημα Nyquist – Shannon

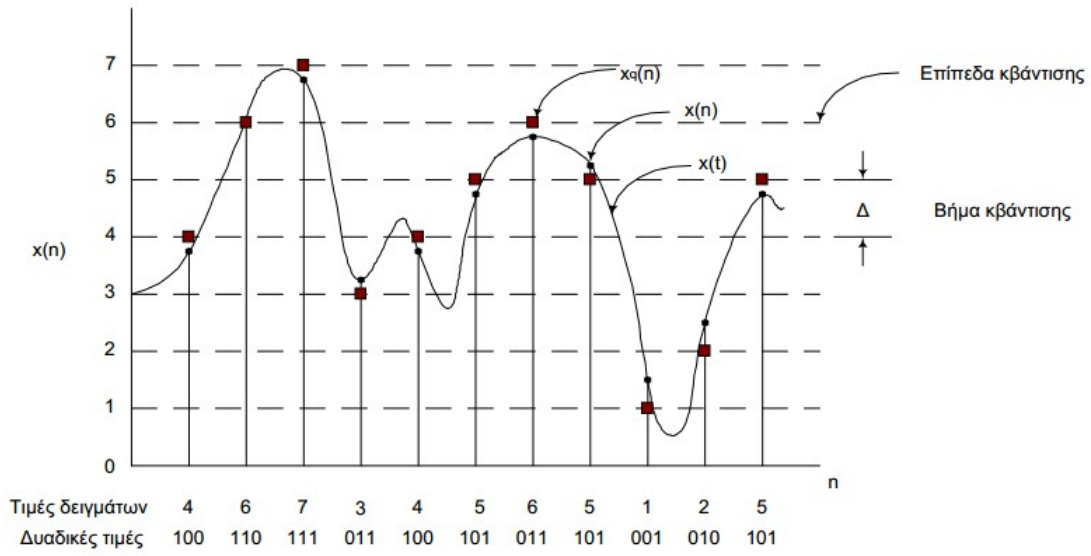
Το θεώρημα δειγματοληψίας Nyquist – Shannon δηλώνει ότι για να μην έχω απώλεια πληροφορίας κατά τη δειγματοληψία ενός αναλογικού σήματος, το σήμα πρέπει να δειγματοληπτηθεί με συχνότητα δειγματοληψίας Fs τουλάχιστον διπλάσια από τη μεγαλύτερη φυσική συχνότητα του σήματος. Έτσι γίνεται δυνατή και η διαδικασία ανάκτησής του.

Αυτό το θεώρημα είναι εξαιρετικά σημαντικό σε όλους τους τομείς της ψηφιακής επεξεργασίας σήματος και κυρίως ήχου, όπου όλα τα σήματα είναι περιοδικά, ή άθροισμα περιοδικών σημάτων. Δεν είναι τυχαίο που παγκοσμίως η δειγματοληψία σημάτων ήχου γίνεται σχεδόν όλες τις φορές στα 44100 δείγματα ανά δευτερόλεπτο (Hz). Αυτό συμβαίνει διότι το ανθρώπινο αυτί είναι δυνατόν να ακούσει συχνότητες έως και 20000Hz περίπου. Άρα, ο αριθμός 44100 είναι αρκετά πάνω από το διπλάσιο της συχνότητας που φτάνει να ακούσει το ανθρώπινο αυτί. Έτσι, τα σήματα ήχου μπορούν να ανακτηθούν από έναν ψηφιακό δίσκο ή ένα αρχείο mp3 επιτυχώς από τη διαδικασία D/A και να ακουστούν από τον άνθρωπο χωρίς απώλειες.



Εικόνα 2.6: Το θεώρημα Nyquist κατά τη δειγματοληψία.

Κβάντιση και κωδικοποίηση

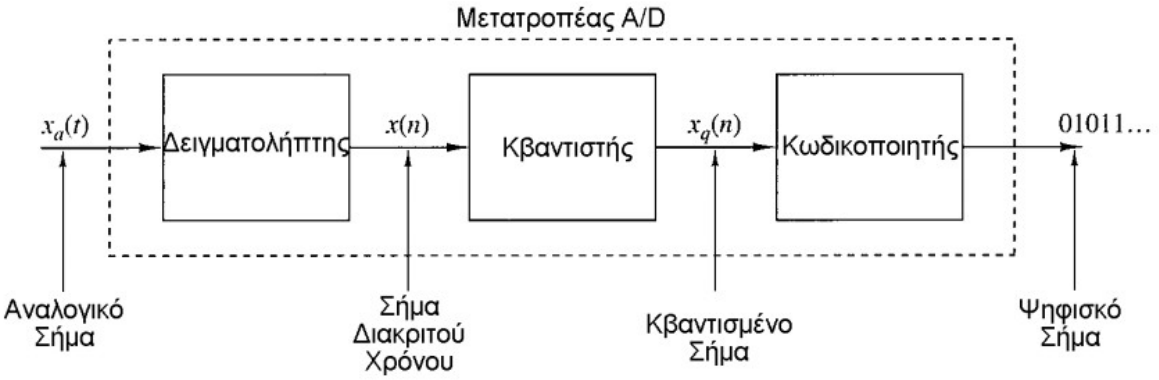


Εικόνα 2.7: Αναπαράσταση της κβάντισης ενός σήματος.

Με τον όρο κβάντιση εννοούμε την αντιστοίχηση κάθε δείγματος που έχει προκύψει μετά τη δειγματοληψία στην κοντινότερη τιμή από ένα πεπερασμένο πλήθος από προκαθορισμένες στάθμες. Με τον τρόπο αυτό, επιβάλλουμε διακριτές τιμές πλάτους για κάθε δείγμα, και του δίνουμε έτσι πεπερασμένη ακρίβεια.

Με την κωδικοποίηση του σήματος οι διακριτές τιμές πλάτους που έχουν προκύψει από την κβάντιση, κωδικοποιούνται σε ξεχωριστές ψηφιακές “λέξεις”, που έχουν μήκος b bits. Σε κάθε επίπεδο κβάντισης ανατίθεται ένας μοναδικός ψηφιακός αριθμός.

Έπειτα από αυτά τα τρία στάδια, μπορεί πλέον και προκύπτει το ψηφιακό σήμα, το οποίο αναπαρίσταται φυσικά μόνο από bits, 0 και 1. Όσο μεγαλύτερη είναι η συχνότητα δειγματοληψίας, ο αριθμός σταθμών στη κβάντιση, και τα bits αναπαράστασης των σταθμών κβάντισης, τόσο "ακριβότερο" γίνεται και το σύστημά μας.



Εικόνα 2.8: Ένας μετατροπέας αναλογικού σε ψηφιακό σήμα.

2.3. Πλεονεκτήματα της ψηφιακής από την αναλογική επεξεργασία

Υπάρχουν διάφοροι λόγοι για τους οποίους η ψηφιακή επεξεργασία είναι πιο χρήσιμη από την αναλογική. Ας δούμε μερικούς από αυτούς.

- Τα ψηφιακά συστήματα είναι πάντα προγραμματιζόμενα. Ένα προγραμματιζόμενο ψηφιακό σύστημα επιτρέπει την ευκολία αλλαγής των λειτουργιών της ψηφιακής επεξεργασίας σήματος τροποποιώντας απλά το πρόγραμμα. Αλλαγή των λειτουργιών σε ένα αναλογικό σύστημα συνήθως απαιτεί επανασχεδιασμό του υλικού που απαιτείται, που συνήθως είναι μεγάλα κυκλώματα που περιέχουν πυκνωτές, αντιστάσεις, τρανζίστορ, και άλλα ηλεκτρονικά στοιχεία, και επίσης επαλήθευση της σωστής λειτουργίας του.

- Η ακρίβεια παίζει σημαντικό ρόλο στον καθορισμό της μορφής του επεξεργαστή σήματος. Η ψηφιακή επεξεργασία σήματος παρέχει καλύτερο έλεγχο της ακρίβειας που απαιτείται. Οι ανοχές ενός αναλογικού συστήματος κάνουν πολύ δύσκολο για τον σχεδιαστή του συστήματος τον έλεγχο της ακρίβειας του συστήματος αναλογικής επεξεργασίας σήματος.

- Τα ψηφιακά συστήματα είναι εύκολο να αποθηκευτούν σε μαγνητικά ή άλλα μέσα αποθήκευσης χωρίς φθορά ή απώλεια της πιστότητας του σήματος, έτσι το σήμα μπορεί να μεταφερθεί και να επεξεργαστεί σε άλλο εργαστηριακό χώρο ή υπολογιστή.

- Η μέθοδος ψηφιακής επεξεργασίας σήματος επιτρέπει την εφαρμογή περισσότερο εξεζητημένων αλγορίθμων επεξεργασίας σήματος. Είναι συνήθως πολύ δύσκολο να γίνουν ακριβείς μαθηματικές πράξεις σε ένα σήμα αναλογικής μορφής. Παρ' όλα αυτά όλες αυτές οι πράξεις μπορούν να γίνουν εύκολα με τη βοήθεια ενός υπολογιστή.
- Σε μερικές περιπτώσεις εφαρμογών η ψηφιακή επεξεργασία σήματος έχει μικρότερο κόστος από την αναλογική. Το μικρότερο κόστος μπορεί να οφείλεται στο γεγονός ότι ο ψηφιακός εξοπλισμός είναι φθηνότερος, ή είναι αποτέλεσμα της ευκολίας αλλαγών της ψηφιακής εφαρμογής.

Είναι φανερό ότι όλο και περισσότερα προβλήματα θα βρίσκουν λύσεις που θα απολαμβάνουν τα πλεονεκτήματα της ψηφιακής επεξεργασίας σήματος, όπως για παράδειγμα σταθερότητα, δυνατότητα για επαναπρογραμματισμό, βελτιωμένη απόδοση σε σχέση με τα αναλογικά εισοδύναμά τους. Ένα τρανταχτό παράδειγμα, στο οποίο βρίσκουν ανταπόκριση και διευρυμένη χρήση πολλές τεχνικές ψηφιακής επεξεργασίας σήματος, είναι η παραγωγή ήχου και μουσικής.

Η επεξεργασία των ακουστικών σημάτων που πραγματοποιείται σε αίθουσες ηχογραφήσεων, είναι μία από τις πιο ενδιαφέρουσες εφαρμογές της ψηφιακής επεξεργασίας σήματος. Οι εργασίες ηχογράφησης και επεξεργασίας ήταν από πάντα πολύπλοκες και ποικίλες. Πριν από την επέκταση διαφόρων ψηφιακών τεχνικών, που έκαναν τη ζωή των παραγωγών μουσικής πολύ πιο εύκολη, διαδικασίες όπως το κόψιμο, το ράψιμο, η επανεγγραφή, η ενίσχυση, το φιλτράρισμα, τα ηχητικά εφέ και άλλες πολλές είχαν τεράστια δυσκολία στο να υλοποιηθούν, και πολλές φορές οδηγούσαν και σε απώλεια πληροφορίας.

ΚΕΦΑΛΑΙΟ 3
FPGA

3.1. Μία εισαγωγή στις FPGA

Η FPGA (Field Programmable Gate Array) ή συστοιχία επιτόπια προγραμματιζόμενων πυλών, είναι ένας τύπος προγραμματιζόμενου ολοκληρωμένου κυκλώματος γενικής χρήσης. Είναι κατασκευασμένο έτσι ώστε να δίνει τη δυνατότητα στο χρήστη να το προγραμματίζει ο ίδιος κατά το δοκούν, έτσι ώστε να εκτελεί τις επιθυμητές λειτουργίες. Οι FPGAs προγραμματίζονται με κώδικα γλώσσας περιγραφής υλικού, VHDL ή Verilog.

Εικόνα 3.1: Μια FPGA. Μοντέλο "Spartan" της Xilinx.

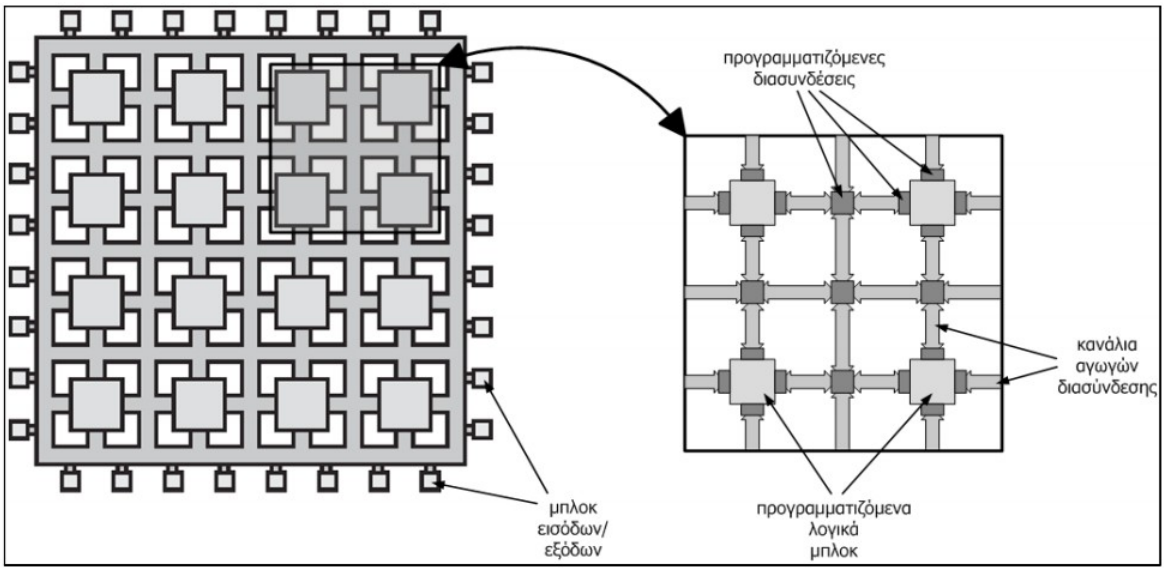
Οι FPGAs μπορούν να χρησιμοποιηθούν και να υλοποιήσουν οποιοδήποτε λογικό κύκλωμα όπως καταχωρητές, μετρητές, μπορούν ακόμα να υλοποιήσουν και μικρούς επεξεργαστές. Τα τελευταία χρόνια έχουν αντικαταστήσει πολλά από τα ήδη υπάρχοντα ολοκληρωμένα, λόγω του ότι έχουν αυξήσει σημαντικά τις επιδόσεις και έχουν μειώσει το κόστος. Το μεγαλύτερό τους πλεονέκτημα είναι η δυνατότητά τους να επαναπρογραμματίζονται, πράγμα που μέχρι την εμφάνισή τους φάνταζε αδύνατο. Κυρίως αυτή τους η δυνατότητα τις έκανε να επικρατήσουν έναντι των ASIC (Application-Specific integrated circuits, ολοκληρωμένα κυκλώματα για συγκεκριμένη εφαρμογή) σε πολλές εφαρμογές.

3.2. Η αρχιτεκτονική των FPGAs

Οι FPGAs αποτελούνται από ομάδες από λογικές πύλες οι οποίες είναι διασυνδεδεμένες μεταξύ τους με τέτοιο τρόπο, ώστε να μπορούν να εκτελέσουν συγκεκριμένες λειτουργίες. Αυτές οι ομάδες λογικών πυλών ονομάζονται “λογικά μπλοκ”. Στην αρχιτεκτονική αυτών των λογικών μπλοκ εμείς δεν έχουμε τη δυνατότητα να επέμβουμε, μπορούμε όμως να τα διαχειριστούμε, να τα προγραμματίσουμε να εκτελούν μία συγκεκριμένη δουλειά, να τα συνδέσουμε μεταξύ τους και να τους δώσουμε μια ιεραρχία από κανόνες έτσι ώστε να μπορούν να φέρουν εις πέρας πολλές και πολύπλοκες λειτουργίες, από μαθηματικές πράξεις μέχρι εξεζητημένες συναρτήσεις.

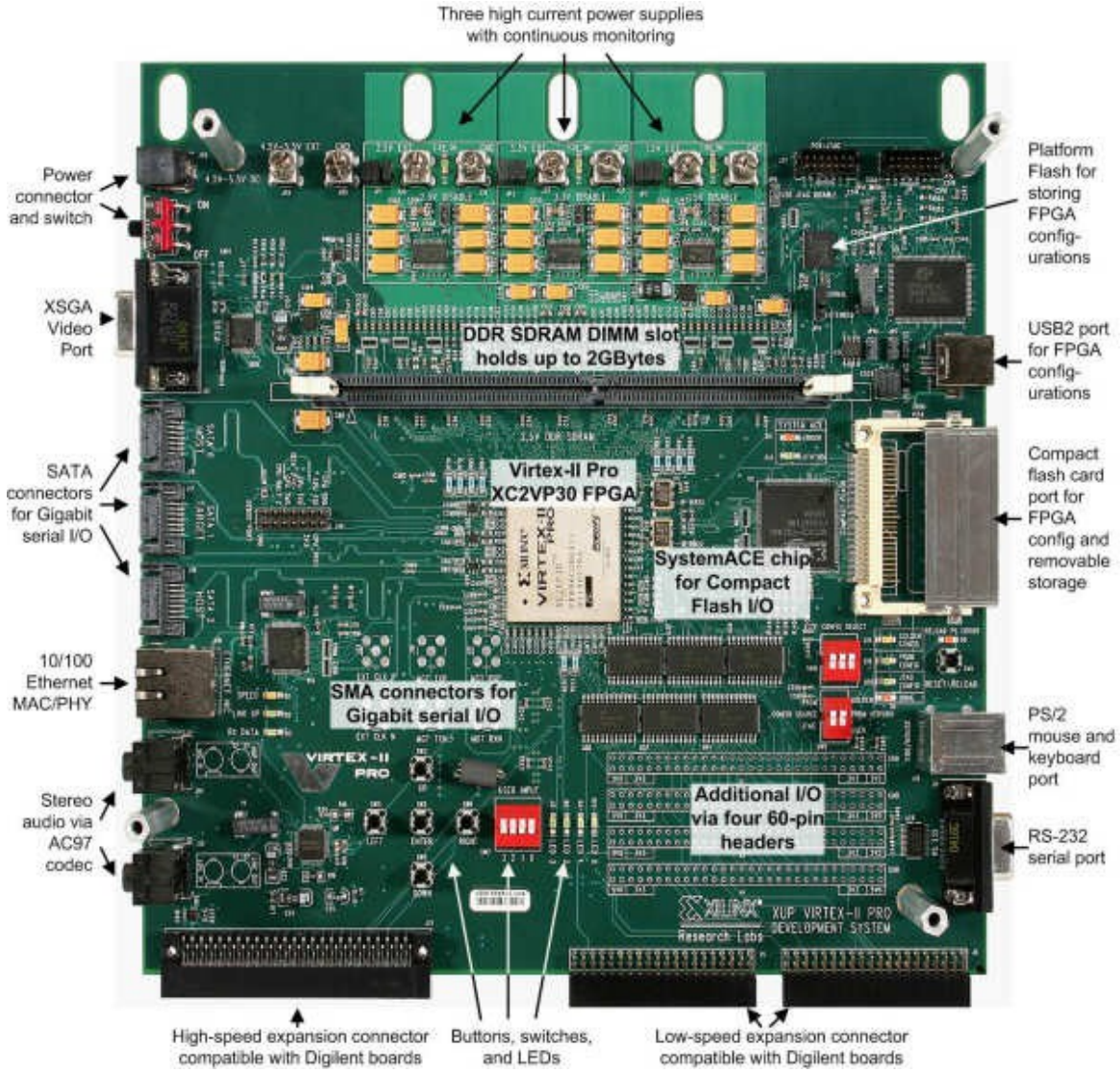
Τα λογικά αυτά μπλοκ (configurable logic blocks, CLBs) μπορούν να εκτελέσουν λειτουργίες βασικών λογικών πυλών (AND, OR, XOR) και ορισμένες μαθηματικές πράξεις. Στην ουσία, όταν προγραμματίζουμε μια FPGA, προγραμματίζουμε κυρίως τον τρόπο με τον οποίο θα γίνουν οι διασυνδέσεις μεταξύ των λογικών αυτών μπλοκ.

Η διάταξη των προγραμματιζόμενων λογικών μπλοκ έχει κανονική μορφή, όμοια με αυτήν ενός δισδιάστατου πίνακα, όπως φαίνεται στην εικόνα. Σε κάθε κανάλι διασύνδεσης υπάρχει ένας αριθμός αγωγών διασύνδεσης, οι οποίοι διατρέχοντας κάθετα και οριζόντια όλο το κύκλωμα μας δίνουν τη δυνατότητα πολυεπίπεδων προγραμματιζόμενων διασυνδέσεων, άρα και πολυεπίπεδων υλοποιημένων συναρτήσεων μέσω των διασυνδέσεων αυτών μεταξύ των λογικών μπλοκ. Φυσικά υπάρχουν συνδεδεμένα με τους ακροδέκτες του ολοκληρωμένου και μπλοκ εισόδου/εξόδου, που συνδέονται με τα προγραμματιζόμενα λογικά μπλοκ και αυτά με αγωγούς διασύνδεσης.



Εικόνα 3.2: Κάτοψη της γενικής αρχιτεκτονικής μιας FPGA.

Οι FPGAs σήμερα σπανίως συναντώνται μόνες τους, αλλά τις πιο πολλές φορές είναι μέρος ενός μεγάλου τυπωμένου κυκλώματος με διάφορα περιφερειακά, που μπορούν να τους δώσουν τη δυνατότητα εισόδων / εξόδων για να εκτελέσουν πάμπολλες λειτουργίες. Παραδείγματα τέτοιων περιφερειακών μονάδων είναι ο μετατροπέας αναλογικού σε ψηφιακό σήμα και το αντίστροφο, είσοδοι line-in για μικρόφωνο, έξοδοι για ηχεία, κάρτες ήχου, υποδοχή για μνήμη RAM ή σκληρό δίσκο, υποδοχή USB και άλλα.



Εικόνα 3.3: Η Virtex-II Pro που χρησιμοποιήσαμε. Στο κέντρο είναι η FPGA, και γύρω της όλες οι διαθέσιμες υποδοχές και τα περιφερειακά: υποδοχή USB, μνήμης RAM, ποντικιού και πληκτρολογίου, σκληρού δίσκου, δυνατότητα για audio input/output, video, και άλλα πολλά. Η εικόνα αυτή βρίσκεται και στην ιστοσελίδα της κατασκευάστριας εταιρίας Digilent, www.digilentic.com

3.3. Λίγα λόγια για την ιστορία των FPGA

Η βιομηχανία των FPGAs δημιουργήθηκε ύστερα από την ένωση δύο προϊόντων, της PROM (προγραμματιζόμενη μνήμη μόνο για ανάγνωση – programmable read-only memory) και των PLDs (προγραμματιζόμενες λογικές διατάξεις – programmable logic devices). Και τα δύο αυτά προϊόντα είχαν την ιδιότητα του προγραμματιζόμενου ολοκληρωμένου κυκλώματος.

Το 1992, ο Steve Casselman ολοκλήρωσε ένα μακρόχρονο project χρηματοδοτούμενο από το υπουργείο άμυνας των ΗΠΑ το οποίο ήταν η ανάπτυξη ενός υπολογιστή ο οποίος θα περιείχε 600.000 επαναπρογραμματιζόμενες πύλες.

Το 1985, οι ιδρυτές της Xilinx, Ross Freeman και Bernard Vonderschmitt, εφηύραν την πρώτη εμπορικά βιώσιμη FPGA (XC2064). Αυτή η FPGA είχε προγραμματιζόμενες πύλες και διαμορφώσιμες διασυνδέσεις μεταξύ τους, και ήταν η απαρχή της σύγχρονης τεχνολογίας των FPGA στην αγορά. Το XC2064 καυχιόταν για 64 διαμορφώσιμα μπλοκ

λογικής, με δύο look-up tables. Περισσότερα από 20 χρόνια αργότερα, ο ιδρυτής αυτής της FPGA, Ross Freeman, εντάχθηκε στο inventor hall of fame για την εφεύρεσή του.

Η Xilinx συνέχισε να είναι η μόνη “εκπρόσωπος” της τεχνολογίας αυτής, μέχρι τα μέσα της δεκαετίας του 90', το οποίο υπήρξε ένα εκρηκτικό διάστημα στην τεχνολογία των FPGAs όπου οι εταιρίες παραγωγής τους πολλαπλασιάστηκαν. Οι FPGAs εκείνη την εποχή ανέβηκαν πολλά επίπεδα τόσο στην πολυπλοκότητα του σχεδιασμού τους, όσο και στον όγκο της παραγωγής. Τότε χρησιμοποιούνταν κυρίως στις τηλεπικοινωνίες και στη δικτύωση. Μέχρι το τέλος της δεκαετίας αυτής βρήκαν εφαρμογή στις αυτοκινητιστικές και άλλες βιομηχανίες.

Αν το 1992 ο Casselman καινοτομούσε φτιάχνοντας 600.000 επαναπρογραμματιζόμενες πύλες, τη σήμερον ημέρα οι FPGAs απαριθμούν πολλά εκατομμύρια από λογικά μπλοκ. Επίσης οι εφαρμογές στη χρήση των FPGAs απογειώνονται σε πλήθος από τη στιγμή που αυτές συναντώνται με ενσωματωμένους μικροεπεξεργαστές και συναφή περιφερειακά ώστε να δημιουργείται ένα πλήρες σύστημα ενσωματωμένο σε ένα και μοναδικό ολοκληρωμένο κύκλωμα. Παραδείγματα τέτοιας τεχνολογίας μπορούμε να τα δούμε σήμερα σε FPGAs της γνωστής εταιρίας Xilinx, όπως η Virtex-II Pro και η Virtex-4, οι οποίες περιέχουν έναν ή περισσότερους μκροεπεξεργαστές και διάφορα περιφερειακά, όπως D/A-A/D converter 2 καναλιών.

Χρήση επεξεργαστών μπορεί επίσης να γίνει με υλοποίηση πυρήνα επεξεργαστή μέσα στην FPGA, αποτελώντας μια σημαντικότατη δυνατότητα των FPGAs.

3.4. Εφαρμογές των FPGAs

Τα FPGAs συναντώνται σε ποικίλλες εφαρμογές. Τέτοιες είναι η ψηφιακή επεξεργασία σήματος, το ψηφιακό ραδιόφωνο, ιατρική απεικόνιση, υπολογιστική όραση, αναγνώριση ομιλίας, εξομείωση πειραματικών μονάδων υπολογιστών, αστρονομία, ανίχνευση μετάλλων και πολλές άλλες αναπτυσσόμενες εφαρμογές.

Όταν υλοποιήθηκαν και παρουσιάστηκαν στην αγορά, είχαν δυνατότητες πολύ μεγαλύτερες των προκατόχων τους τόσο σε ταχύτητα όσο και σε μέγεθος. Πιο συγκεκριμένα, μετά την εισαγωγή κυκλωμάτων πολλαπλασιαστών στην αρχιτεκτονική τους, γύρω στα τέλη του 90', κυριάρχησαν στις εφαρμογές DSP. Ο εγγενής παραλληλισμός των λογικών πόρων σε μία FPGA επιτρέπει σημαντική υπολογιστική απόδοση ακόμα και σε χαμηλές τιμές χρονισμού μερικών Mhz. Η ευελιξία του FPGA επιτρέπει την επίτευξη ακόμα υψηλότερων επιδόσεων. Αυτό το γεγονός δικαιολογεί σε μεγάλο βαθμό τη χρήση τους παρά το υψηλό κόστος κατασκευής τους.

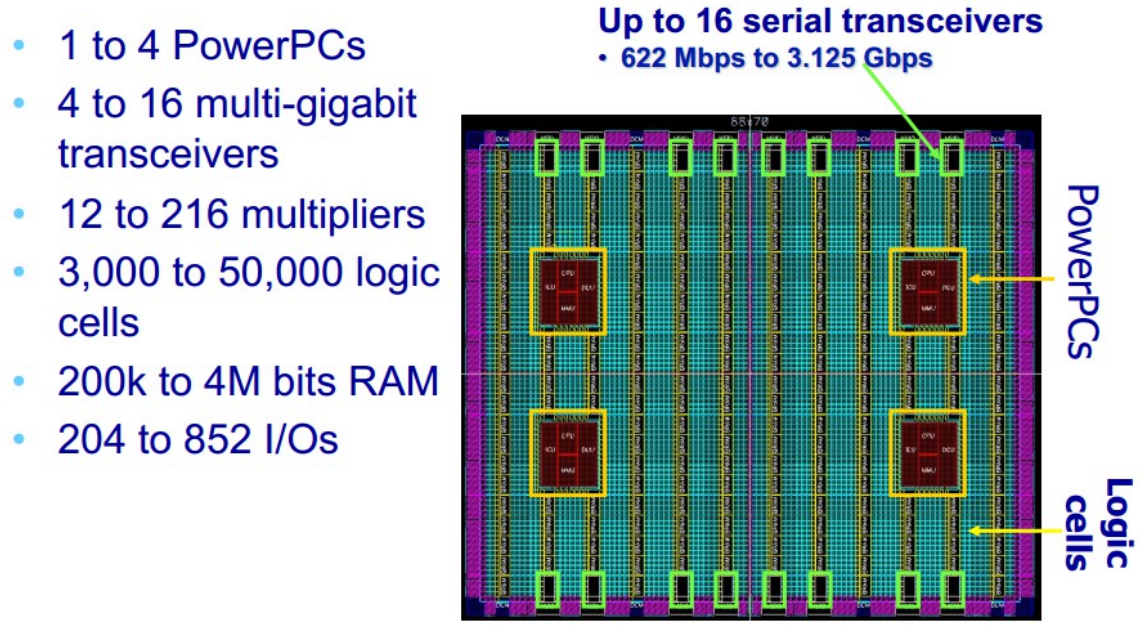
3.5. Η XUP Virtex-II Pro που χρησιμοποιήσαμε

Η οικογένεια Virtex της Xilinx, περιλαμβάνει μια σειρά από FPGAs σχεδιασμένες να προσφέρουν υψηλή απόδοση και χωρητικότητα. Αποτελούν μία από τις πλέον επιτυχημένες και διαδεδομένες οικογένειες FPGAs. Η κατασκευή τους είναι βασισμένη στην τεχνολογία SRAM, που τους προσδίδει την ιδιότητα των άπειρων φορών επαναπρογραμματισμού. Αυτά τα SRAM προγραμματιζόμενα στοιχεία, όπως αποκαλούνται, έχουν τη δυνατότητα αποθήκευσης πληροφορίας ενός bit στο εσωτερικό τους. Αυτά είναι που προσδίδουν στην Virtex FPGA την ιδιότητα του προγραμματιζόμενου ολοκληρωμένου.

Η XUP (Xilinx University Program) Virtex II pro είναι ένα πολύ αντιπροσωπευτικό παράδειγμα ολοκληρωμένου κυκλώματος, που συνδυάζει την FPGA με διάφορα περιφερειακά. Τα περιφερειακά που χρησιμοποιήσαμε εμείς ήταν φυσικά για επεξεργασία ήχου η είσοδος/έξοδος για ηχητικό σήμα line-in και amp out, καθώς και τον ενσωματωμένο AC97' codec με χρήση ενός interface, στο οποίο αναφερόμαστε παρακάτω.

Η Virtex II pro χρησιμοποιείται κατά κόρον για πανεπιστημιακές εργασίες, καθώς είναι το κατάλληλο μοντέλο FPGA για να μπορέσει να ξεκινήσει και να εξοικειωθεί κανείς,

λόγω της μεγάλης ευελιξίας της σε πολλές εφαρμογές, όπως DSP και άλλα. Η υποστήριξη που δίδεται από τη Xilinx όσον αφορά τη χρήση της και την ανάπτυξή της, με διάφορα tutorial και sheets, την καθιστά ένα πολύ ισχυρό εργαλείο. Μειονέκτημά της όμως, είναι η απουσία συμβατότητας με πιο σύγχρονα εργαλεία προγραμματισμού και διαχείρησης, δυσκολία που αντιμετωπίσαμε και στην παρούσα διπλωματική εργασία. Η μη-συμβατότητά της το εργαλείο της Xilinx ISE από την έκδοση 10.1 και μετά, είναι κάτι το οποίο πρέπει να ληφθεί υπ' όψιν πριν την προτίμησή της για χρήση.



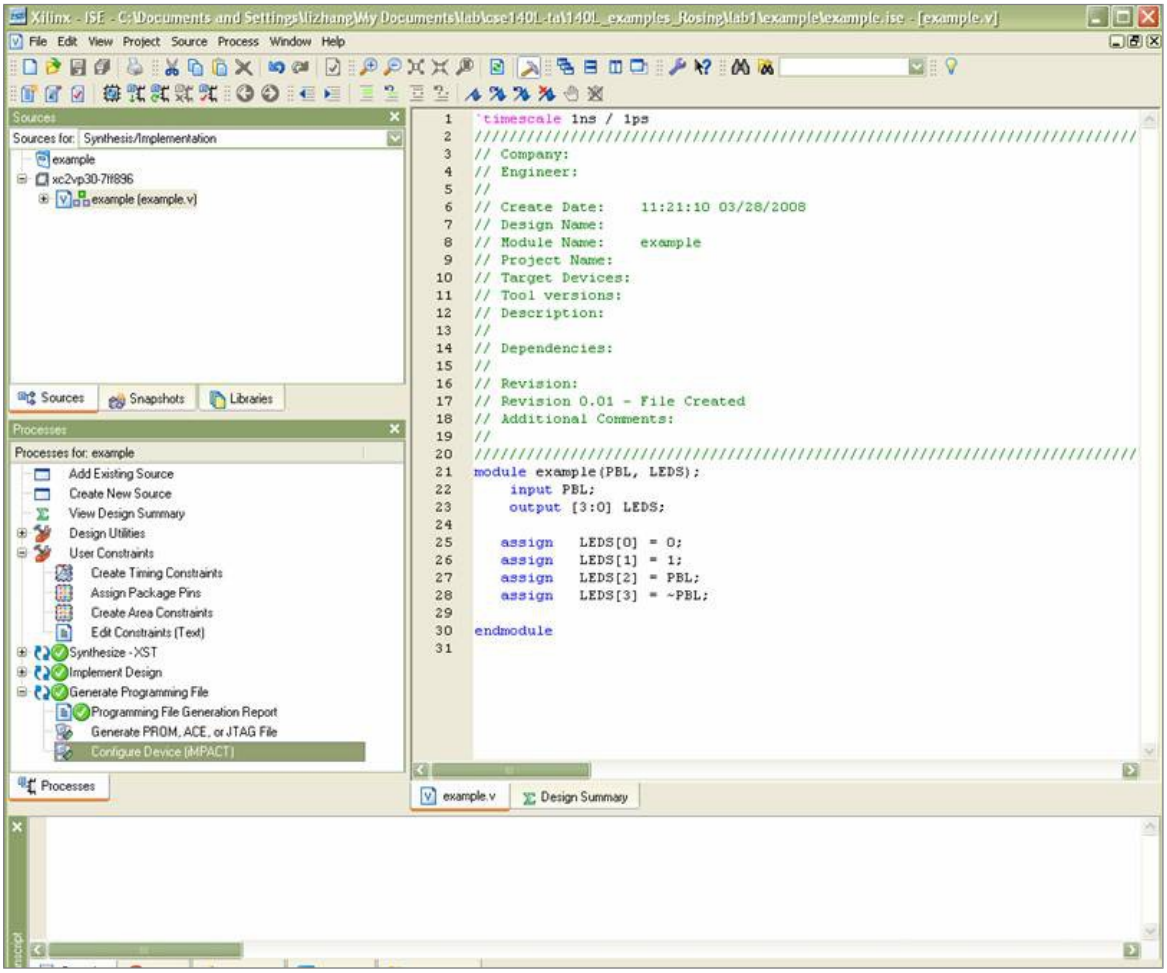
Εικόνα 3.4: Γενική κάτοψη και δυνατότητες της Virtex-II Pro που χρησιμοποιήσαμε.

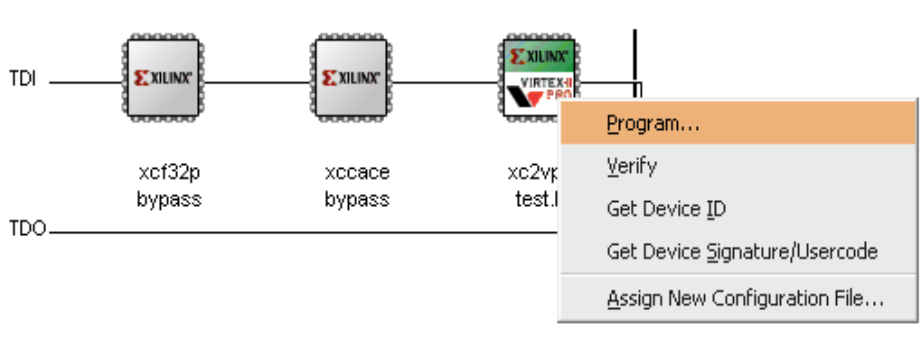
Φυσικά υπάρχουν και πιο σύγχονα μοντέλα FPGAs από την Xilinx όπως είναι οι Virtex-4 και 5, που παρουσιάζουν ακόμα μεγαλύτερη χωρητικότητα, ταχύτητες επεξεργασίας με ρολόγια έως και 100MHz, και προφανώς συμβατότητα με τα πιο σύγχρονα εργαλεία της Xilinx.

3.6. Ο προγραμματισμός μιας FPGA

Για να προγραμματίσουμε μια FPGA, κάνουμε χρήση μια γλώσσα περιγραφής υλικού, VHDL ή Verilog. Πολλές φορές μπορεί να γίνει και με χρήση ενός σχηματικού (block diagram), πράγμα που μας δίνει τη δυνατότητα ευκολότερης απεικόνισης, αλλά όχι λεπτομερή εικόνα πολύπλοκων δομών εντός της υλοποίησης.

Στη συνέχεια χρησιμοποιούμε κάποιο εργαλείο που θα μπορέσει να πραγματοποιήσει τη διασύνδεση ανάμεσα στον κώδικά μας και την FPGA, τη λεγόμενη netlist. Συγκεκριμένα για την υλοποίηση του δικού μας project χρησιμοποιήσαμε το εργαλείο Xilinx ISE 10.1. Μέσω λοιπόν της εντολής place-and-route, η netlist προσαρμόζεται στην αρχιτεκτονική της FPGA που θέλουμε να προγραμματίσουμε. Από τη στιγμή που ολοκληρωθεί η διαδικασία σχεδιασμού του προγράμματος, δημιουργείται ένα αρχείο bitstream (δυαδικό αρχείο) μέσω της εντολής Generate Programming file, το οποίο πρόκειται να προγραμματίσει την FPGA. Για την ακρίβεια αυτά τα bits του αρχείου αυτού περιέχουν πληροφορία για το πώς θα γίνουν οι διασυνδέσεις και οι λειτουργίες των (εκατοντάδων χιλιάδων) λογικών μπλοκ, που περιγράψαμε παραπάνω. Τέλος, με τη βοήθεια του εργαλείου Impact, το .bit αρχείο κατεβαίνει στην FPGA, προσδίδοντάς της πλέον συγκεκριμένες λειτουργίες.





Εικόνες 3.5 και 3.6: Το περιβάλλον του προγράμματος ISE της Xilinx και ο προγραμματισμός μιας FPGA Virtex-II Pro μέσω του προγράμματος iMPACT.

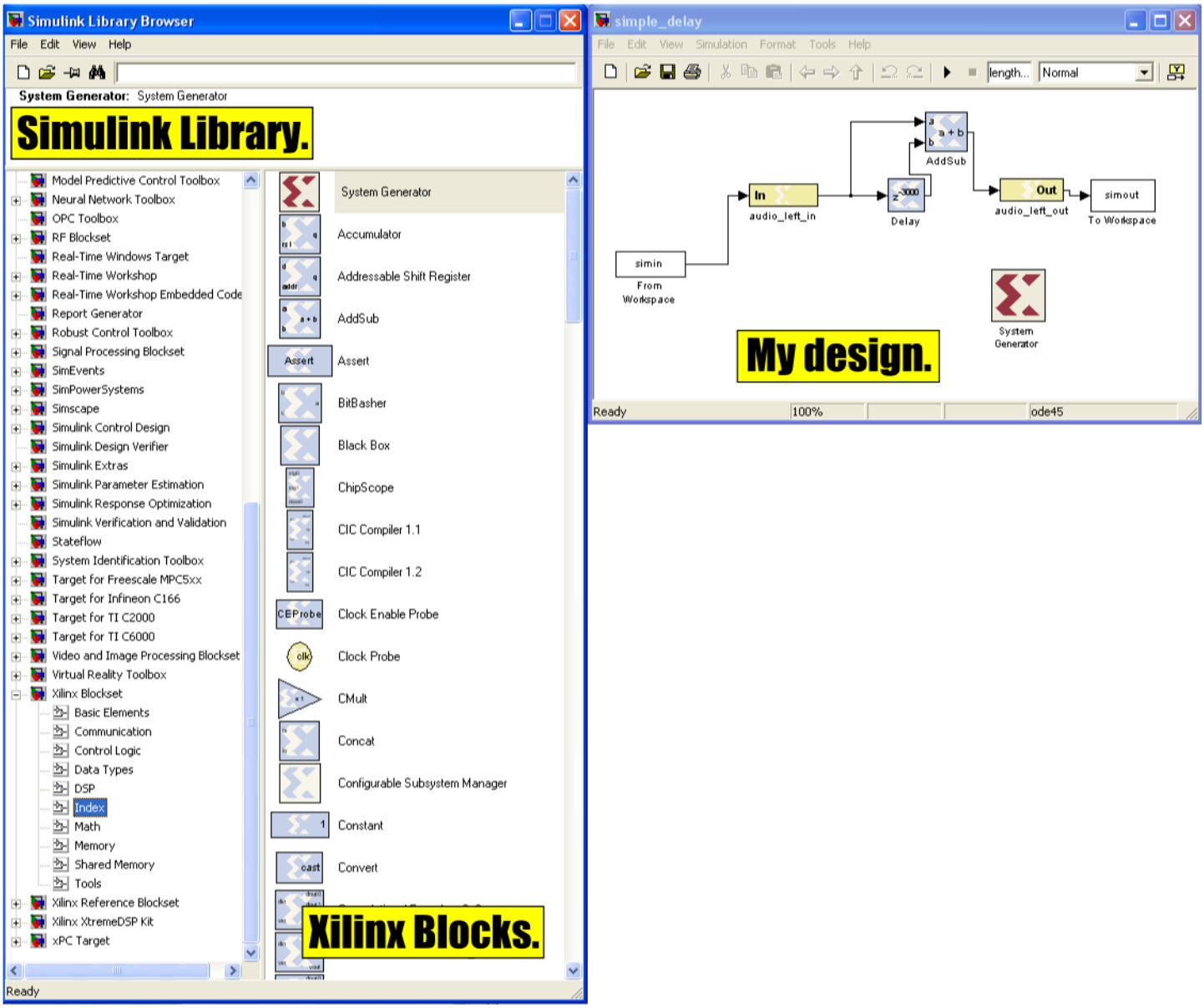
ΚΕΦΑΛΑΙΟ 4

Το εργαλείο Xilinx System Generator for DSP

4.1. Εισαγωγή

Το Xilinx System Generator for DSP είναι ένα εργαλείο σχεδίασης συστημάτων ψηφιακής επεξεργασίας σήματος (DSP) μέσω του περιβάλλοντος Simulink της MATLAB με στόχο την υλοποίηση της σχεδίασης σε FPGA. Η σχεδίαση ενός τέτοιου συστήματος έχει τη λογική του Block Diagram (όπως ακριβώς το simulink), δηλαδή την κατάλληλη διασύνδεση μονάδων-components που εκτελούν κάποια συγκεκριμένη λειτουργία. Κάθε μπλοκ φυσικά έχει τις δικές του εισόδους και εξόδους. Αυτά τα σχέδια (designs) υλοποιούνται με μπλοκ που η ίδια η Xilinx έχει δημιουργήσει.

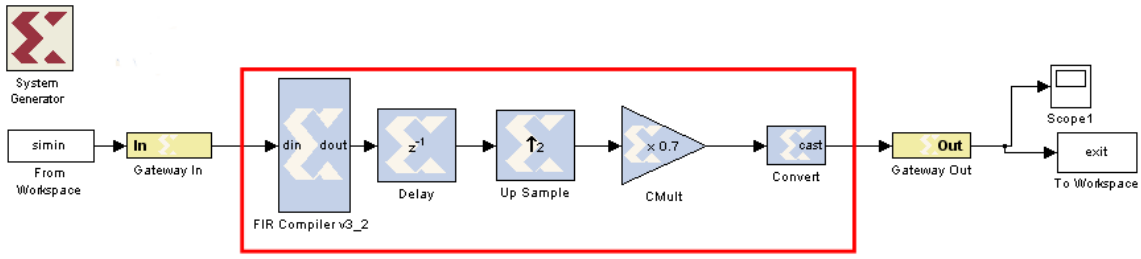
Το εργαλείο παρέχει τη δυνατότητα για πλήρη εξομοίωση του συστήματος που έχουμε σχεδιάσει, έλεγχο, αποτύπωση των αποτελεσμάτων είτε στο workspace της MATLAB είτε σε παλμογράφο (scope block) για αποτύπωση στην οθόνη. Φυσικά, για κάθε σχεδίαση μπορεί να παράγει και τον αντίστοιχο κώδικα περιγραφής υλικού VHDL που την περιγράφει πλήρως.



Εικόνα 4.1: Το περιβάλλον του System Generator for DSP.

4.2. Περιγραφή και τρόπος χρήσης

Τα DSP blocks που μπορούν να χρησιμοποιηθούν παρέχονται από τη Xilinx στις βιβλιοθήκες του Simulink. Αυτή τη στιγμή διατίθενται περίπου 100 μπλοκ προς χρήση. Σε αυτά περιλαμβάνονται τα βασικότερα στοιχεία που μπορεί να αποτελούν μια εφαρμογή επεξεργασίας σήματος, αθροιστές, πολλαπλασιαστές και καταχωρητές. Επίσης περιλαμβάνονται και ένα σύνολο από πιο πολύπλοκα συστήματα όπως FFTs, φίλτρα και μνήμες. Πέρα όμως από τα ήδη υπάρχοντα μπλοκ στις βιβλιοθήκες του System Generator υπάρχει η δυνατότητα ο σχεδιαστής να δημιουργήσει το δικό του System Generator μπλοκ που μπορεί να εκτελεί ακόμα και ιδιαίτερα περίπλοκους αλγορίθμους. Ένα τέτοιο μπλοκ μπορεί να αποτελεί ένα υποσύστημα της σχεδίασης και μπορεί να συνεργάζεται με τα ήδη διαθέσιμα μπλοκ.



Εικόνα 4.2: Τα Gateway In και Gateway Out blocks. Μόνο ό,τι είναι ανάμεσά τους βρίσκεται στο εσωτερικό της FPGA. Πάνω αριστερά φαίνεται και το System Generator Token.

Μία σχεδίαση ενός συστήματος στο System Generator πρέπει πάντα να ξεκινά με τον προσδιορισμό των "ορίων" της FPGA. Τα δύο μπλοκ "Gateway in" και "Gateway out" μας δίνουν αυτή τη δυνατότητα. Στην ουσία, ό,τι υπάρχει ανάμεσα σε αυτά τα δύο μπλοκ αποτελεί το "περιεχόμενο" της FPGA, και ότι είναι έξω από αυτά χρησιμοποιείται είτε για να δώσει μία είσοδο στο σύστημά μας "από τον έξω κόσμο", όπως πχ. μπορεί να είναι μία μεταβλητή από το workspace της MATLAB, ή να δώσει την έξοδο του συστήματος στον "έξω κόσμο". Όλα τα υπόλοιπα μπλοκ της Xilinx, πρέπει οπωσδήποτε να παίρνουν δεδομένα εισόδου/εξόδου από και προς άλλα μπλοκ της Xilinx. Τα μπλοκ αυτά ξεχωρίζουν εύκολα από τα αντίστοιχα του Simulink, έχοντας το logo της Xilinx να φαίνεται επάνω στο κουτί.

Μία σχεδίαση ενός τέτοιου συστήματος επίσης δεν μπορεί να δουλέψει αν δε θέσουμε κάποιες παραμέτρους. Αυτές λοιπόν τις παραμέτρους της σχεδίασης τις ρυθμίζει το System Generator Token. Το συγκεκριμένο μπλοκ δε συνδέεται με το υπόλοιπο σύστημα, αλλά περιέχει πληροφορίες για το πώς αυτό θα τρέξει. Μέσω αυτού του μπλοκ ρυθμίζεται μεταξύ άλλων το clocking της εξομοίωσης, ή, σε περίπτωση που θελήσουμε να κατεβάσουμε αυτή τη σχεδίαση σε κάποια FPGA, ρυθμίζει τα Mhz στα οποία θα τρέχει, κάνει generate στον VHDL κώδικα σε συγκεκριμένο directory και για συγκεκριμένο μοντέλο FPGA.

Όταν ολοκληρώνουμε τη σχεδίασή μας μέσω της επιλογής Generate του System Generator Token δημιουργούνται τα απαραίτητα αρχεία για την υλοποίηση στην FPGA μέσω του προγράμματος ISE. Φυσικά μπορούν να υποστούν επεξεργασία περαιτέρω χειροκίνητα με το συγκεκριμένο εργαλείο, καθώς είναι στη διάθεσή μας ο κώδικας VHDL. Κάθε μπλοκ που έχουμε χρησιμοποιήσει, αποτελεί ένα ξεχωριστό component στον παραγόμενο κώδικα VHDL.

4.3. Οι λόγοι για τους οποίους χρησιμοποιήσαμε το Xilinx System Generator for DSP

Τρεις είναι οι βασικότεροι λόγοι για τους οποίους χρησιμοποιήσαμε το Xilinx System Generator for DSP.

- Πριν από όλα, και στο ξεκίνημα της παρούσας διπλωματικής εργασίας, υλοποιήσαμε, μελετήσαμε και εξοικιωθήκαμε με τους αλγορίθμους DSP των ηχητικών εφέ στο περιβάλλον της MATLAB. Θα ήταν εξαιρετικά δύσκολο να μεταβούμε κατευθείαν στην συγγραφή και υλοποίηση αυτών των αλγορίθμων σε VHDL, χωρίς να έχουμε πρώτα μία "οπτική", "γραφική" εικόνα στο πώς μπορούν αυτοί οι αλγόριθμοι να λειτουργήσουν σε hardware. Ακόμα και να γράφαμε τελικά τους αλγορίθμους σε VHDL από την αρχή ως το τέλος, θα μας βοηθούσε σαν ενδιάμεσο στάδιο η "σχηματική" απεικόνισή τους.

- Το πρόβλημα της δοκιμής. Σίγουρα ποτέ κανένας μηχανικός υπολογιστών ανά τον κόσμο, όσο διάνοια και να ήταν, δεν έγραψε ένα πολύπλοκο πρόγραμμα σε κάποια γλώσσα προγραμματισμού, και του δούλεψε από την πρώτη κιόλας φορά. Για τη σωστή συγγραφή και λειτουργία ενός προγράμματος η διαδικασία των δοκιμών είναι αναπόσπαστο στοιχείο. Πόσο μάλλον στη δικιά μας περίπτωση, που είχαμε να κάνουμε με αλγορίθμους επεξεργασίας ήχου, θα μας ήταν εξαιρετικά δύσκολο και χρονοβόρο να κατεβάζουμε στην FPGA μία υλοποίηση, να δούμε αν δουλεύει, αν όχι επιδιόρθωση και πάλι κατέβασμα, κοκ. Η δυνατότητα που μας έδωσε ο System Generator για εξομείωση του συστήματος, μας έλυσε τα χέρια. Και επίσης, η δυνατότητα να κάνουμε import σαν είσοδο κάποιο σήμα (διάνυσμα) ήχου από το workspace της MATLAB, μας απάλλαξε από το γεγονός να αναμειχθούμε με πολύπλοκα και τεράστια testbench για σήματα ήχου.

- Για έναν μηχανικό ο οποίος είναι εξοικειωμένος με το εργαλείο αυτό, ο χρόνος που μεσολαβεί από την κατανόηση και τη λειτουργία του αλγορίθμου μέχρι την υλοποίησή του σε hardware και FPGA μειώνεται πάρα πολύ. Ο χρήστης γλιτώνει χρόνο και δοκιμής, και υλοποίησης, και δύσκολων DSP πράξεων και συναρτήσεων οι οποίες προϋποθέτουν πολλές γραμμές κώδικα σε VHDL αλλά στο System Generator αποτελούν μπλοκ, ή συνδυασμό τους. Η γνώση της χρήσης του εργαλείου αυτού από τους μηχανικούς κάνει την παραγωγικότητα ολόκληρων project σε τομείς ηλεκτρονικής και ενσωματωμένων συστημάτων να αυξάνεται, και τον απαιτούμενο χρόνο να μειώνεται.

Δυστυχώς υπάρχουν και μειονεκτήματα στη χρήση του εργαλείου αυτού, που όμως σε καμία περίπτωση δεν υπονομεύουν τη χρησιμότητά του και την αξία του. Βασικότερο μειονέκτημα είναι η αυξημένη δυσκολία του κώδικα VHDL που παράγεται και σε όγκο, αλλά και σε πολυπλοκότητα. Ενώ απαιτούνται βασικές γνώσεις VHDL για να φέρει κάποιος εις πέρας ένα project χρησιμοποιώντας αυτό το εργαλείο, χρειάζεται τεράστια εμπειρία και εξοικείωση με τη γλώσσα αυτή για να μπορέσει κάποιος να “βάλει χέρι” αργότερα στον κώδικα που θα παραχθεί. Αξίζει να αναφέρουμε, ότι σε κανένα από όλα τα ηχητικά εφέ που υλοποιήσαμε, ο παραγόμενος κώδικας δεν πήγε κάτω από τις 1400 γραμμές (!).

Ένα ακόμα μειονέκτημα που συνήθως λύνεται, είναι η συμβατότητα που απαιτείται μεταξύ των εργαλείων που “συνεργάζονται” μεταξύ τους. Η έκδοση του System Generator πρέπει πάντα φυσικά να συνοδεύεται και από την αντίστοιχη ακριβώς έκδοση του ISE, αλλά και από μία αντίστοιχη έκδοση του λογισμικού της MATLAB, να τρέξει σε συγκεκριμένο λειτουργικό, για ένα συγκεκριμένο μοντέλο FPGA της Xilinx. Για να μπορέσουμε να βρούμε την “χρυσή τομή” μεταξύ όλων αυτών των απαιτήσεων συμβατότητας ταλαιπωρηθήκαμε αρκετά. Παρ' όλα αυτά, δεν είναι κάτι που δε λύνεται.

Για την εκπόνηση της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκαν τα εργαλεία της Xilinx ISE 10.1, System Generator for DSP 10.1, Impact 10.1, και η MATLAB R2006b, σε περιβάλλον Windows XP (λόγω μη συμβατότητας με Windows 7 ή Linux όπου δοκιμάστηκαν στην αρχή).

ΚΕΦΑΛΑΙΟ 5

Τα ηχητικά εφέ που υλοποιήθηκαν

Σ' αυτό το κεφάλαιο γίνεται η πλήρης περιγραφή όλων των ηχητικών εφέ που καταφέραμε να προσομοιώσουμε και να υλοποιήσουμε. Είναι η βασική δουλειά της παρούσας διπλωματικής εργασίας. Όλα τα εφέ μελετήθηκαν πρώτα σε περιβάλλον MATLAB έτσι ώστε να γίνει πλήρως κατανοητός ο αλγόριθμος που περιγράφει τη λειτουργία τους. Στη συνέχεια σχεδιάστηκαν στο εργαλείο System Generator for DSP και προσομοιώθηκαν στο περιβάλλον του Simulink δεχόμενα ως είσοδο αρχεία ήχου από τη MATLAB και εξάγοντας αποτελέσματα ξανά σε αυτήν. Αναφέρεται επίσης και ο υπολογισμός του κόστους όπως μας το εξήγαγε το εργαλείο ISE της Xilinx κατά τη διάρκεια της σύνθεσης.

5.1. Το εφέ distortion ή fuzz

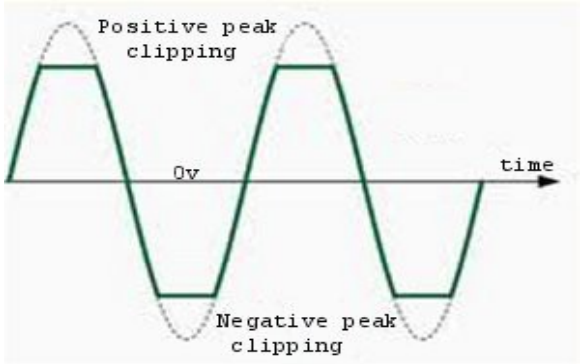
Εικόνα 5.1: Ένα από τα πιο διάσημα distortion effect units (pedals) της εταιρίας Boss.

Πρόκειται για ένα πάρα πολύ απλό εφέ, το οποίο υλοποιήθηκε στο ξεκίνημα της παρούσας διπλωματικής εργασίας πιο πολύ για εξοικείωση με τα εργαλεία και με την επεξεργασία του ήχου. Το εφέ αυτό βρωμίζει τον ήχο παραμορφώνοντάς τον, εξ' ου και το όνομα fuzz.

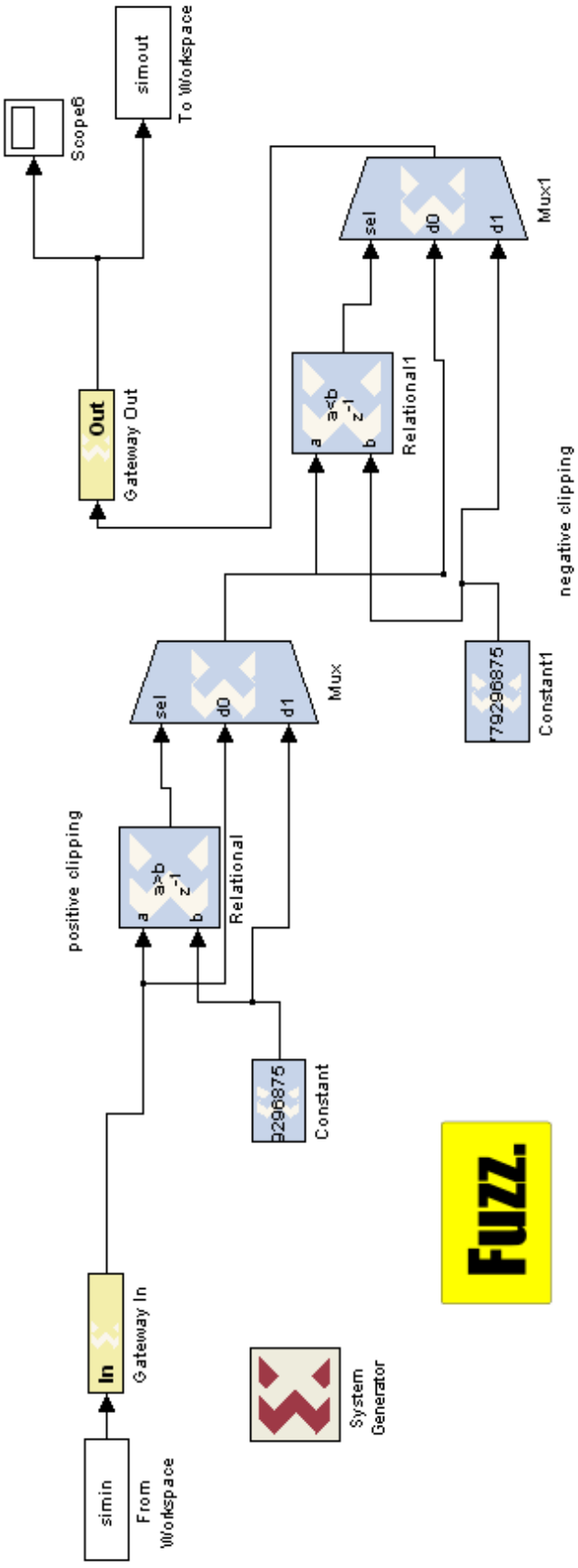
Μπορεί να δημιουργείται η απορία του γιατί να είναι χρήσιμο ένα εφέ το οποίο "χειροτερεύει" τον ήχο: Το εφέ αυτό χρησιμοποιείται σχεδόν αποκλειστικά από κιθαρίστες (ίσως και γενικότερα παίκτες εγχόρδων), καθώς προσδίδει τον γνωστό "ηλεκτρικό" ήχο σε μία κιθάρα. Τέτοιοι παραμορφωμένοι ήχοι χρησιμοποιήθηκαν πρώτη φορά όταν οι μουσικοί έβαζαν στο "φουλ" να παίζουν οι ενισχυτές της κιθάρας τους.

Ο αλγόριθμος είναι πάρα πολύ απλός, και λειτουργεί ως εξής. Επιλέγουμε μία συγκεκριμένη τιμή και περιορίζουμε το σήμα ήχου να μην πάρει τιμές μεγαλύτερες από αυτή. Καθώς λοιπόν εισέρχεται το σήμα ήχου στην είσοδο, πραγματοποιούμε για κάθε δείγμα μία σύγκριση με τη συγκεκριμένη τιμή. Σε περίπτωση που η τιμή είναι μικρότερη από αυτήν, τότε παραμένει αμετάβλητο στην έξοδο. Σε αντίθετη περίπτωση, εξισώνεται με τη συγκεκριμένη τιμή. Έτσι δημιουργείται το φαινόμενο clipping (ψαλίδισμα), που δίνει και αυτή την παραμορφωμένη αίσθηση στον ήχο.

Προφανώς το σημείο που μπορεί να γίνει το clipping κάθε φορά εξαρτάται και από το μέγιστο πλάτος (amplitude) που έχει το σήμα, κοινώς η έντασή του. Αν το σημείο του clipping προσπεράσει το μέγιστο πλάτος, προφανώς και δεν υπάρχουν αλλαγές στο σήμα. Αν, αντίστοιχα το σημείο αυτό βρίσκεται πολύ χαμηλά, υπάρχει κίνδυνος να αποκοπεί πολλή σημαντική πληροφορία από το σήμα μας, κι έτσι να καταλήξει να μην έχει πλέον φυσική σημασία.



Εικόνα 5.2: Το φαινόμενο clipping σε ένα ημιτονοειδές σήμα.


simin
From Workspace
In
Gateway In
System Generator
9296875
Constant
positive clipping
a a>b b
z-1
Relational
sel d0 d1
Mux
79296875
Constant1
negative clipping
a a<b b
z-1
Relational1
sel d0 d1
Mux1
Out
Gateway Out
Scope6
simout
To Workspace
Fuzz.

Ο χώρος που καταλαμβάνει το εφέ fuzz.

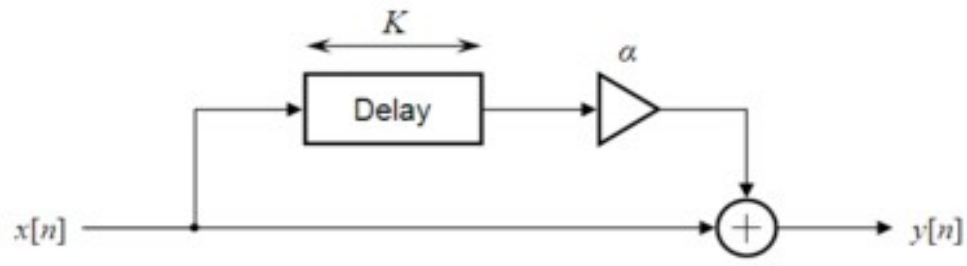
```
Logic Utilization:
  Number of Slice Flip Flops:           711 out of  27,392    2%
  Number of 4 input LUTs:               606 out of  27,392    2%
Logic Distribution:
  Number of occupied Slices:            657 out of  13,696    4%
     Number of Slices containing only related logic:       657 out of
657 100%
     Number of Slices containing unrelated logic:            0 out of
657   0%

  Total Number of 4 input LUTs:         707 out of  27,392    2%
    Number used as logic:               399
    Number used as a route-thru:        101
    Number used as Shift registers:     207
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      5 out of     136    3%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           32
Peak Memory Usage:  210 MB
```

5.2. Delay

Η γενική του χρήση

Σε πάρα πολλά είδη μουσικής, το εφέ της καθυστέρησης (delay), χρησιμοποιείται ποικιλοτρόπως, σε πολλά μουσικά όργανα όπως κιθάρες, πιάνο, αρμόνιο, πνευστά και άλλα, όπως επίσης και σε φωνητικά. Οι μουσικοί το χρησιμοποιούν κατά κόρον για να δώσουν στη μουσική τους μεγαλύτερο όγκο και έμφαση, για να γίνει ο ήχος που παράγεται πιο “ζεστός” και φαντασμαγορικός. Στην ανθρώπινη φωνή επίσης δίνει όγκο, και “πυκνότητα”. Κατα γενική ομολογία, η σωστή χρήση του delay (ή άλλων ηχητικών εφέ βασισμένων σε αυτό), μπορεί να ανεβάσει ποιοτικά ένα σήμα ήχου για χρήση στον τομέα της μουσικής ή και αλλού.

Η λογική του delay είναι αρκετά απλή. Το αρχικό ηχητικό σήμα ακολουθείται από μία επανάληψη του εαυτού του, όπως ακριβώς η ηχώ (echo), το γνωστό σε όλους φυσικό φαινόμενο. Ο χρόνος του delay, δηλαδή σε πόσο ακριβώς χρόνο ξεκινά η επανάληψη του σήματος, μπορεί να διαφέρει. Χρησιμοποιούνται ευρέως αλγόριθμοι με χρόνους καθυστέρησης από μερικά milliseconds μέχρι και μερικά δευτερόλεπτα. Το εφέ μπορεί να περιέχει μία μόνο καθυστερημένη επανάληψη, ή και πολλές, οι οποίες συνήθως “χαμηλώνουν” (μικραίνει το πλάτος τους) η μία μετά την άλλη.

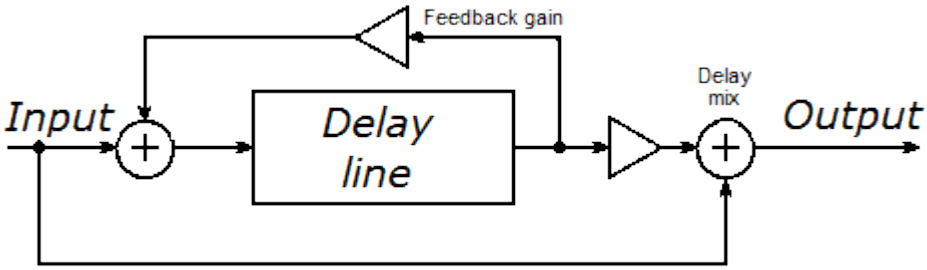


Εικόνα 5.3: Το σχήμα ενός απλού delay, γνωστό και ως comb filter.

Το delay επίσης αποτελεί τη βάση για μία σειρά από άλλα πιο πολύπλοκα εφέ, όπως τα **reverb, vibrato, και flanger.**

5.2.1. Το απλό echo effect

Αυτό το εφέ μπορεί να γίνει εύκολα αντιληπτό σε διάφορους ακουστικούς χώρους που συναντάμε στην καθημερινότητα. Προκαλείται όταν το ηχητικό μας σήμα "προσκρούσει" σε μία επιφάνεια, και η ανάκλασή του επιστρέψει πάλι πίσω. Αν αυτή η "επιφάνεια" είναι μακριά (όπως πχ ο απέναντι γκρεμός όταν είμαστε στο βουνό), τότε ο χρόνος που θα επιστρέψει η επανάληψη πάλι στα αφτιά μας θα είναι μεγάλος. Αντίστοιχα, σε ένα μεγάλο δωμάτιο χωρίς έπιπλα, ο χρόνος επιστροφής θα είναι αρκετά μικρότερος. Αν ο τοίχος από την πηγή καταλήξει να είναι πολύ κοντά, τότε ο χρόνος αυτός θα είναι τόσο ανεπαίσθητος, που μόνο αν συγκεντρωθούμε θα μπορέσουμε να ακούσουμε μία μικρή αλλαγή στο "χρώμα" του ήχου.



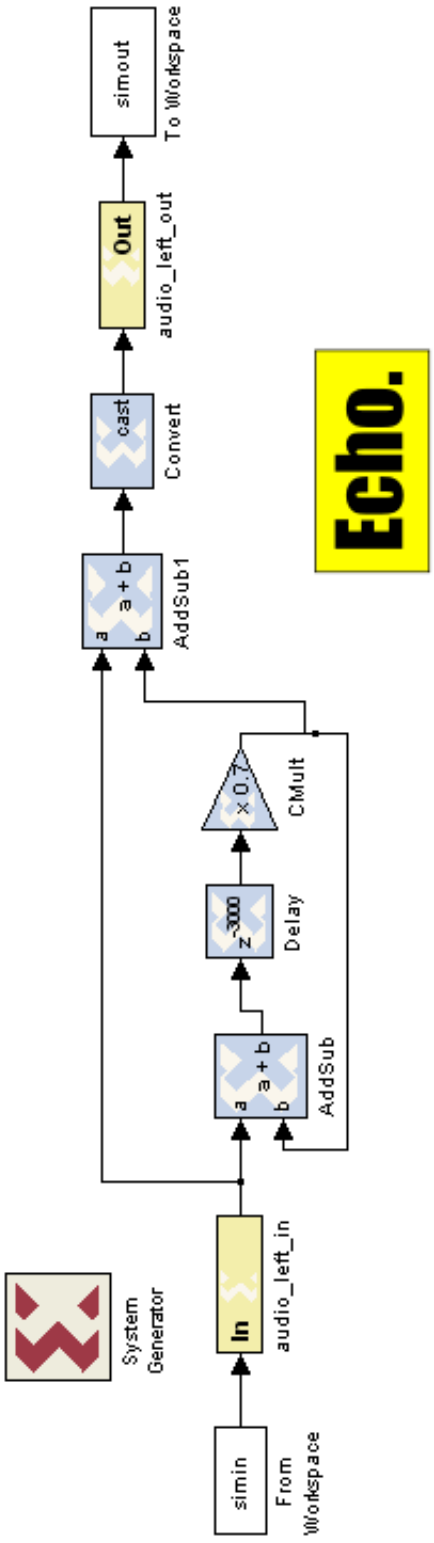
Εικόνα 5.4: Σχήμα ενός απλού echo effect.

Όλα λοιπόν αυτά τα ακουστικά φαινόμενα έχουν υλοποιηθεί και με αλγορίθμους ψηφιακής επεξεργασίας ηχητικού σήματος.

Όπως φαίνεται και στην εικόνα, κάθε επανάληψη πολλαπλασιάζεται με έναν αριθμό (feedback gain) ο οποίος είναι μικρότερος του 1. Αυτός δηλώνει και την "εξασθένιση" των επαναλήψεων σε σχέση με το αρχικό σήμα.

Ο χώρος που καταλαμβάνει το εφέ echo.

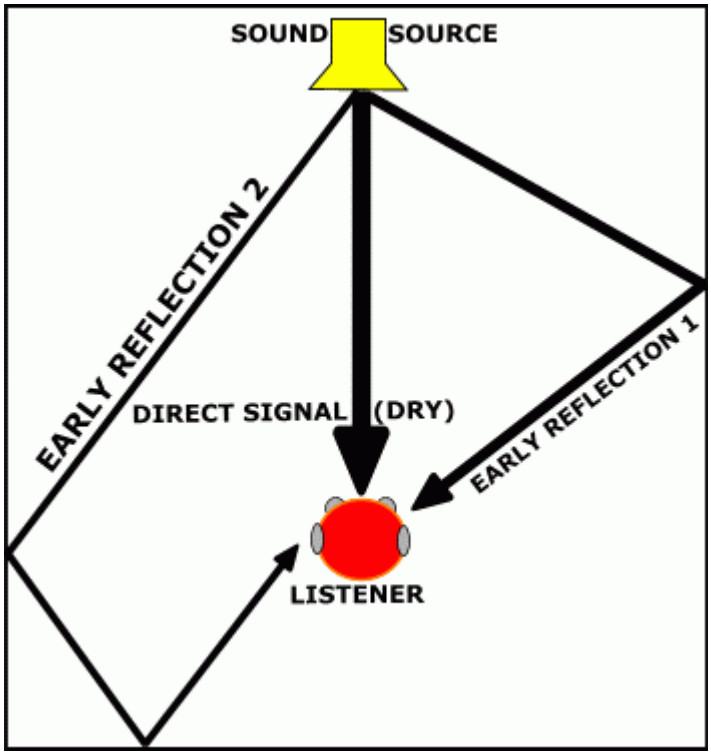
```
Logic Utilization:
  Number of Slice Flip Flops:         3,733 out of  27,392   13%
  Number of 4 input LUTs:             3,729 out of  27,392   13%
Logic Distribution:
  Number of occupied Slices:          2,322 out of  13,696   16%
      Number of Slices containing only related logic:     2,322 out of
2,322 100%
      Number of Slices containing unrelated logic:            0 out of
2,322   0%

  Total Number of 4 input LUTs:       3,854 out of  27,392   14%
    Number used as logic:               513
    Number used as a route-thru:        125
    Number used as Shift registers:   3,216
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      5 out of     136    3%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           34
Peak Memory Usage:  254 MB
```


simin
From Workspace
In
audio_left_in
System Generator
a
a + b
b
AddSub
z^-3000
Delay
x 0.7
CMult
a
a + b
b
AddSub1
cast
Convert
Out
audio_left_out
simout
To Workspace
Echo.

5.2.2. Reverb (αντήχηση)

Το εφέ αυτό είναι από αυτά που συναντάμε πιο συχνά στην παραγωγή μουσικής και ήχου. Μπορεί να κάνει ένα “στεγνό” ηχητικό σήμα να ακουστεί σαν να προέρχεται από κάποιον ακουστικό χώρο – hall με καλή ακουστική, και άλλα πολλά.

Με τον όρο αντήχηση εννοούμε την παραμονή του ήχου σε έναν συγκεκριμένο χώρο, στιγμές αφ' ότου παράχθηκε το αρχικό μας σήμα ήχου. Αυτό προκαλείται όταν ένας αριθμός από επαναλήψεις του ήχου παραχθούν όταν αυτός προσπίπτει σε επιφάνειες οι οποίες έχουν διαφορετικές αποστάσεις από την πηγή κάθε φορά. Οι επαναλήψεις αυτές έχουν διαφορά χρόνου η μία με την άλλη μερικά ms, και η έντασή τους μικραίνει όσο συνεχίζονται λόγω της απορρόφησής τους από τον αέρα και τις επιφάνειες. Ένα φυσικό ανάλογο από την καθημερινότητά μας είναι αυτό του δωματίου.



Εικόνα 5.5: Μία απλή οπτική απεικόνιση της αντήχησης 2 μικρών επαναλήψεων.

5.2.2.A. Ο κώδικας MATLAB

Ο κώδικας MATLAB που έχουμε υλοποιήσει, είναι ένα υπερσύνολο των τριών αυτών εφέ βασισμένων στο delay. Ο λόγος για το απλό delay/slapback, το echo, και το reverb. Η αλλαγή από εφέ σε εφέ γίνεται με βάση κάποιες παραμέτρους, οι οποίες αλλάζουν.

Η γενική λογική του αλγορίθμου είναι ότι αποφασίζει ο χρήστης πόσα θα είναι τα samples της καθυστέρησης, και έπειτα πόσες επαναλήψεις θα γίνουν, καθώς και το “σβήσιμό” τους. Ανάλογα με αυτά συμπεριφέρεται και ο αλγόριθμος.

Ο αλγόριθμος αρχικά κάνει import ως διάνυσμα το σήμα ήχου που του έχουμε δώσει μέσω της εντολής wavread. Στη συνέχεια αρχικοποιεί το διάνυσμα εξόδου, το οποίο φυσικά θα ξεκινάει με το σήμα εισόδου. Η έξοδος θα έχει μήκος όσο το αρχικό σήμα, συν τον αριθμό των επαναλήψεων πολλαπλασιασμένο επί τα δείγματα επανάληψης. Άρα αν έχουμε δώσει

εντολή για 2 επαναλήψεις των 10000 δειγμάτων, το μήκος εξόδου θα είναι κατά 20000 μεγαλύτερο από το αρχικό. Στη συνέχεια, για κάθε αριθμό επαναλήψεων που έχουμε δώσει εντολή, υπολογίζει αντίστοιχους μονοδιάστατους πίνακες ίσους με το μήκος εξόδου, με το σήμα καθυστερημένο στο "σημείο" που θέλουμε, αποδυναμωμένο τόσο όσο του έχουμε δώσει στην αρχή (μεταβλητή c). Όλος ο υπόλοιπος πίνακας είναι γεμισμένος με μηδενικά. Το άθροισμα λοιπόν όλων αυτών των πινάκων σε έναν μονοδιάστατο αποτελεί και το σήμα εξόδου.

Για παράδειγμα: Αν έχουμε δώσει εντολή για 2 επαναλήψεις από 10000 samples η κάθε μία, το σήμα εξόδου θα είναι το άθροισμα τριών πινάκων μήκους του αρχικού συν 20000 που θα έχουν: Ο πρώτος το αρχικό μας σήμα στην αρχή και στη συνέχεια 20000 μηδενικά, ο δεύτερος 10000 μηδενικά στην αρχή και μετά το αρχικό μας σήμα (ενδεχομένως αποδυναμωμένο) και ξανά στο τέλος 10000 μηδενικά, και ο τρίτος 20000 μηδενικά στην αρχή και στο τέλος το αρχικό μας σήμα (ενδεχομένως αποδυναμωμένο).

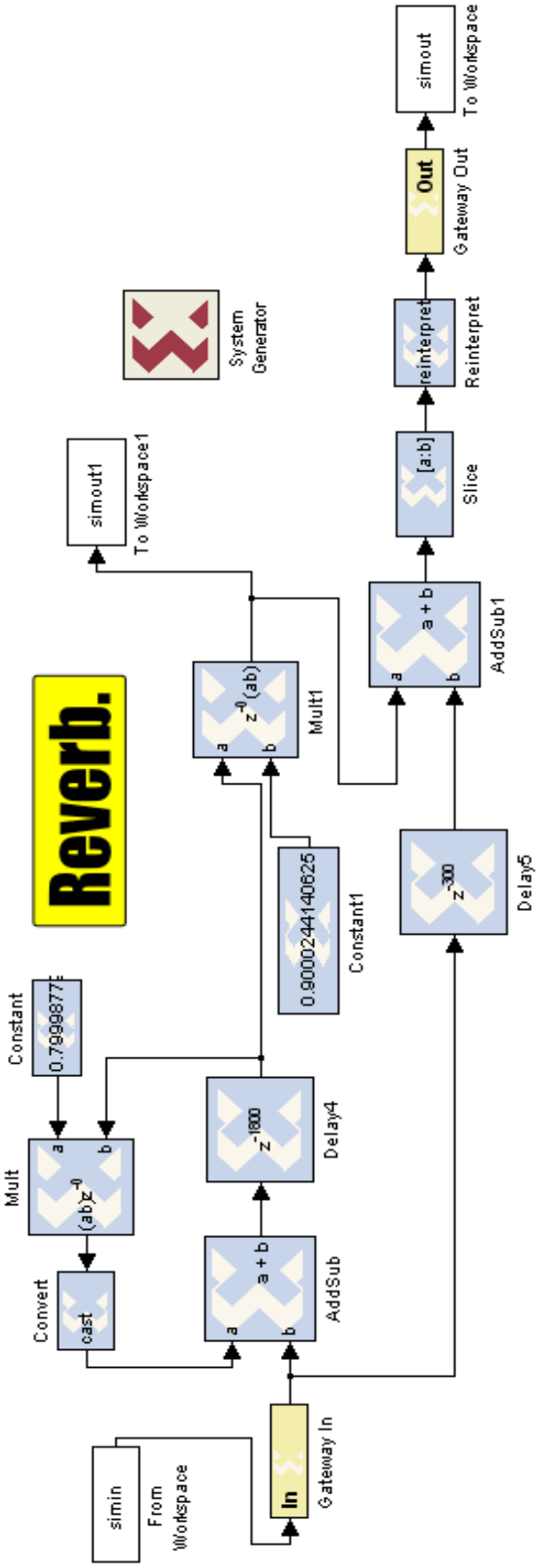
Έτσι λοιπόν στην έξοδο έχουμε σήμα με όσες και όποιες καθυστερήσεις του έχουμε υποδείξει στις αρχικές παραμέτρους. Εάν φυσικά οι καθυστερήσεις είναι για μεγάλο αριθμό δειγμάτων (πχ. γύρω στις 10-15 χιλιάδες) και ο αριθμός των αποδυναμωμένων επαναλήψεων είναι μεγάλος, έχουμε ένα echo effect. Αν πάλι έχουμε 2-3 διαφορετικές καθυστερήσεις διαφορετικού αριθμού δειγμάτων η κάθε μία (από 300 έως 1500) με σύντομη αποδυνάμωση, τότε λαμβάνει χώρα το εφέ reverb.

5.2.2.Β Η υλοποίηση σε hardware

Η δική μας υλοποίηση έγινε με δύο καθυστερήσεις. Η πρώτη είναι μία καθυστέρηση των 1800 δειγμάτων, (αυτό ισοδυναμεί στο χρόνο με 40 milliseconds) η οποία προστίθεται στην είσοδο και "σβήνει" κατά 0.8 σε κάθε επανάληψη. Αμέσως μετά έρχεται να προστεθεί πριν την έξοδο και μία δεύτερη καθυστέρηση των 300 δειγμάτων (6,8 milliseconds) της οποίας το πλάτος μειώνεται κατά 0.9 μετά από κάθε επανάληψη. Το αποτέλεσμά μας ακούγεται απολύτως ικανοποιητικό, με τον ήχο μας να "μένει" για ακόμα λίγα κλάσματα του δευτερολέπτου μέχρι να χαθεί.

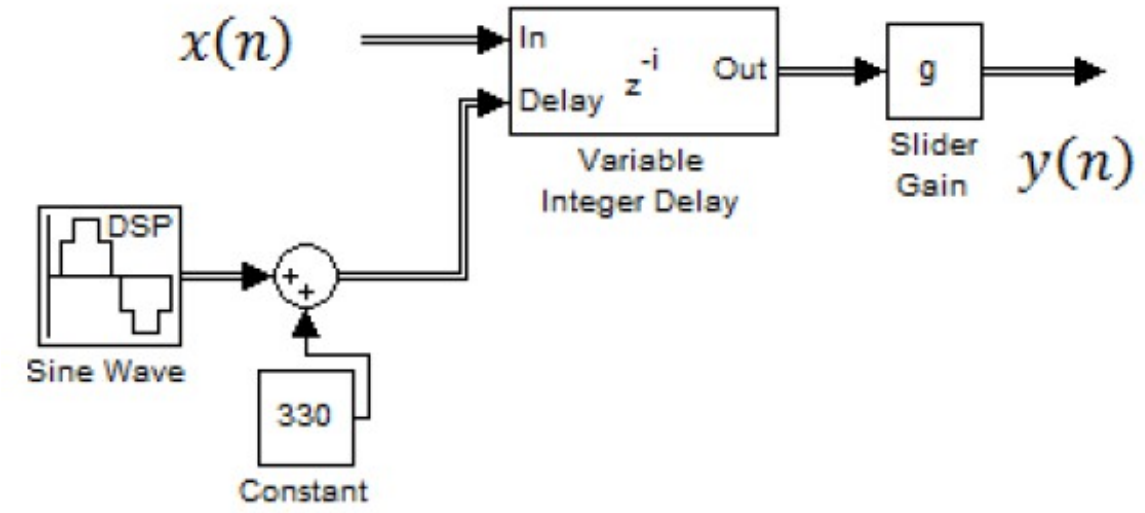
Ο χώρος που καταλαμβάνει το εφέ Reverb.

```
Logic Utilization:
  Number of Slice Flip Flops:         2,906 out of  27,392   10%
  Number of 4 input LUTs:             2,715 out of  27,392    9%
Logic Distribution:
  Number of occupied Slices:          1,816 out of  13,696   13%
      Number of Slices containing only related logic:     1,816 out of
1,816 100%
     Number of Slices containing unrelated logic:             0 out of
1,816   0%

  Total Number of 4 input LUTs:       2,826 out of  27,392   10%
    Number used as logic:               418
    Number used as a route-thru:        111
    Number used as Shift registers:   2,297
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      5 out of     136    3%
  Number of MULT18X18s:                   2 out of     136    1%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           34
Peak Memory Usage:  241 MB
```


simin
From Workspace
In
Gateway In
Mult
Convert
cast
AddSub
Delay4
Constant
0.7999877
0.9000244140625
Constant1
Reverb.
Mult1
simout1
To Workspace1
System Generator
Delay5
AddSub1
Slice
[a:b]
reinterpret
Reinterpret
Out
Gateway Out
simout
To Workspace

5.2.3. Vibrato

Όταν ένα αυτοκίνητο περνά από δίπλα μας με ταχύτητα, ακούμε μία αλλαγή τον τόνο (pitch) του ήχου που βγάζει, η οποία οφείλεται στο φαινόμενο Doppler. Όπως γνωρίζουμε η αλλαγή αυτή στον τόνο του ήχου οφείλεται στο ότι η απόσταση μεταξύ του αυτοκινήτου και του ακροατή συνεχώς αλλάζει. Αλλαγή στην απόσταση, ως εκ τούτου, είναι για εμάς και αλλαγή στο χρόνο που χρειάζεται ο ήχος να φτάσει από την πηγή στα αυτιά μας. Άρα αν αλλάζουμε συνεχώς τιμή στο χρόνο αυτό, επιτυγχάνουμε και συνεχή αλλαγή του pitch του ηχητικού μας σήματος. Σε αυτή τη λογική είναι βασισμένο και το ηχητικό εφέ vibrato, στο οποίο η καθυστέρηση του ήχητικού μας σήματος αυξομειώνεται περιοδικά (ημιτονοειδώς).



Εικόνα 5.6: Το σχηματικό διάγραμμα του εφέ vibrato.

5.2.3.Α Ο κώδικας MATLAB

Ο κώδικας MATLAB του εφέ αυτού λειτουργεί ως εξής: Αφού έχει γίνει import το σήμα ήχου σαν διάνυσμα μέσω της εντολής wavread, δίνονται κάποιες παράμετροι για το πώς θέλουμε να λειτουργήσει το εφέ. Αυτές είναι, ο μέγιστος χρόνος μεταβλητής καθυστέρησης σε milliseconds, καθώς και η συχνότητα του ημιτόνου που προσδίδει την περιοδικότητα σε Hz. Αμέσως μετά υπολογίζεται το ημίτονο με βάση τα δεδομένα που έχουμε εισάγει, καθώς και η μέγιστη καθυστέρηση σε δείγματα. Αφού αρχικοποιήσουμε με 0 το διάνυσμα στο οποίο θα μπει το σήμα εξόδου, ξεκινάει μια δομή επανάληψης. Γίνονται τόσες επαναλήψεις όσα είναι τα δείγματα του σήματός μας (ξεκινώντας φυσικά από όχι από την αρχή του, αλλά από τη μέγιστη τιμή καθυστέρησης δειγμάτων), στις οποίες υπολογίζεται η καθυστέρηση σε δείγματα όπως προκύπτει από το ημίτονο που έχουμε θέσει, και έπειτα αυτή αφαιρείται από το “σημείο” που βρίσκεται εκείνη την ώρα το σήμα μας.

Για να γίνει πιο κατανοητό: έστω ότι το ημίτονο σε μία συγκεκριμένη επανάληψη έχει την τιμή 0.4119. Αυτή, αν πολλαπλασιαστεί με την μέγιστη τιμή καθυστέρησης δειγμάτων (max_samp_delay), θα δώσει μία ακέραια τιμή ίση με 55. Τόση θα είναι λοιπόν η καθυστέρηση που θα δοθεί στο σήμα σε εκείνη την επανάληψη [vibrato(i) = in_wave(i-cur_delay)]. Φυσικά το ημίτονο παίρνει διαφορετική τιμή σε κάθε επανάληψη, άρα και διαφορετική καθυστέρηση για το σήμα μας. Στο τελικό μας σήμα φαίνεται ξεκάθαρα πλέον το vibrato effect.

5.2.3.Β. Η υλοποίηση σε hardware

Για να γίνει περισσότερο κατανοητή η υλοποίηση αυτή, θα χρειαστεί πρώτα να κατασκευάσουμε ένα απλό delay και στη συνέχεια να “εξελίξουμε” την ίδια σχεδίαση

φτιάχνοντας το vibrato. Το βασικό component που χρησιμοποιήσαμε για την υλοποίηση σε hardware είναι μία Dual Port RAM, η οποία μας δίνει την δυνατότητα για ανάγνωση και εγγραφή σε δύο διαφορετικές διευθύνσεις τον ίδιο χρόνο. Η RAM είναι αρχικοποιημένη με 0 σε όλες τις διευθύνσεις.

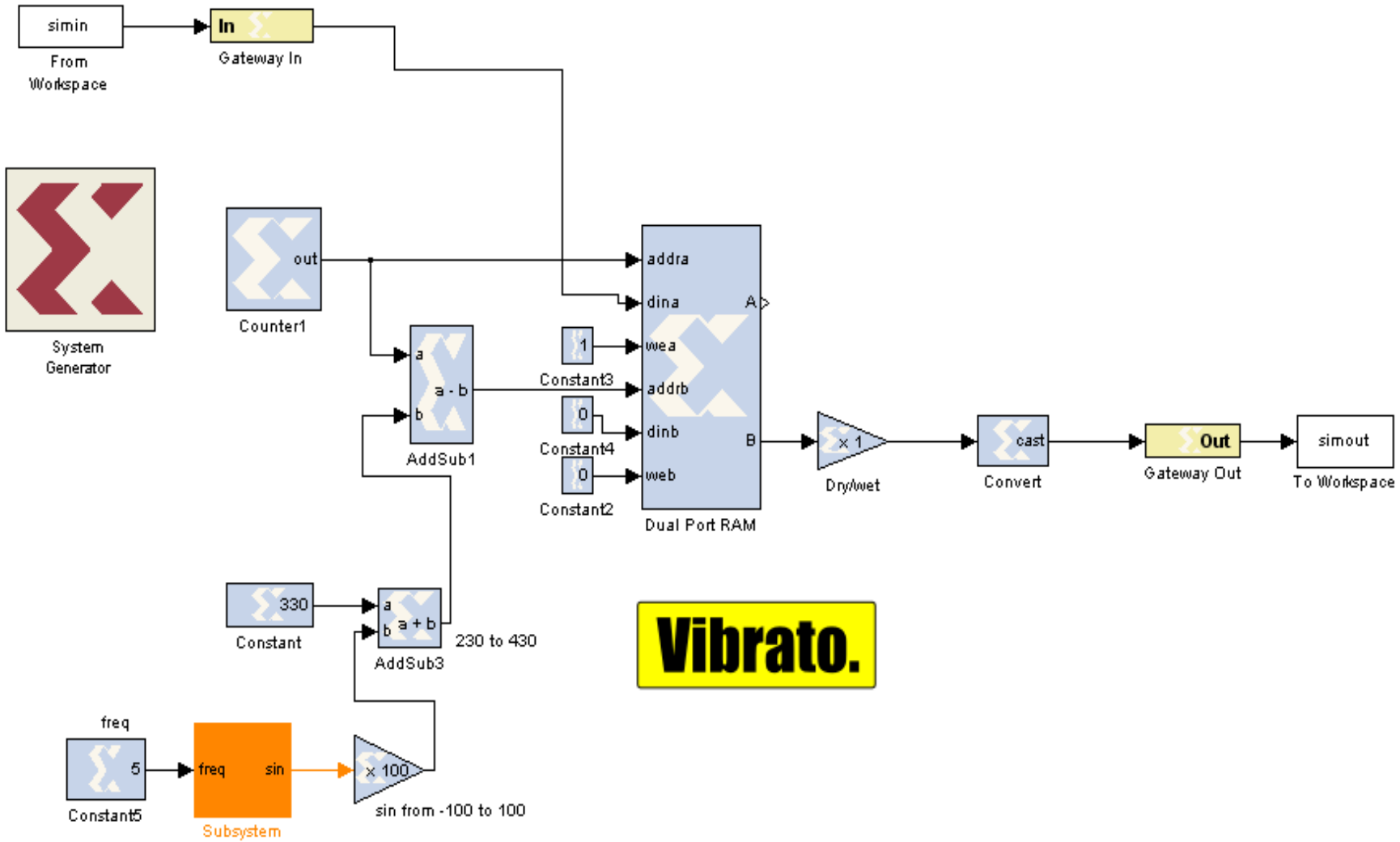
Έτσι στη διεύθυνση Α γράφεται το σήμα ήχου μας δείγμα προς δείγμα. Για δείκτη στη διεύθυνση αυτή χρησιμοποιούμε έναν μετρητή ο οποίος δείχνει στην επόμενη θέση μνήμης σε κάθε χτύπο του ρολογιού. Με αυτόν τον τρόπο η μνήμη γεμίζει με τα δείγματα από το σήμα εισόδου με τη σειρά σε κάθε θέση μνήμης.

Σαν δείκτη στη διεύθυνση Β χρησιμοποιούμε την τιμή του ίδιου μετρητή, που όμως θα είναι μειωμένη κατά έναν αριθμό. Έτσι στην ανάγνωση της διεύθυνσης Β της μνήμης θα έχουμε το σήμα μας καθυστερημένο για έναν αριθμό δειγμάτων. Αν ο αριθμός αυτός που θα αφαιρούμε από τον μετρητή είναι το 4410, τότε από την διεύθυνση μνήμης Β θα εξέρχεται το σήμα μας "καθυστερημένο" κατά 0.1 sec. Αν λοιπόν αυτό προστεθεί με το αρχικό μας σήμα τότε στην έξοδο θα έχουμε το απλό echo effect που περιγράψαμε παραπάνω.

Το vibrato effect στην ουσία επιτυγχάνεται αν ο αριθμός αυτός που αφαιρούμε από τον μετρητή (που μας δείχνει στη διεύθυνση μνήμης Α, όπου γράφεται το σήμα) δεν έχει σταθερή τιμή, αλλά συνεχώς διακυμένεται, ανάμεσα στις τιμές "καθυστέρησης" των δειγμάτων. Αυτή την εναλλασσόμενη τιμή που θέλουμε να αφαιρείται από τον μετρητή την επιτυγχάνουμε φυσικά χρησιμοποιώντας ένα ημίτονο, του οποίου το πλάτος να διακυμαίνεται πχ. από 230 έως 430 δείγματα που σημαίνει από 5 έως 9 ms καθυστέρησης. Έτσι λοιπόν κατά την ανάγνωση της διεύθυνσης μνήμης Β, το σήμα εξέρχεται άλλοτε καθυστερημένο κατά 230, και άλλοτε κατά 430, όσο του υποδεικνύει το ημίτονο αυτό. Έτσι έχουμε επιτύχει το λεγόμενο variable delay που απαιτείται για το vibrato effect. Το αποτέλεσμα είναι αρκετά ικανοποιητικό έως τέλειο, και το σήμα μας στην έξοδο ακούγεται "τρεμάμενο". Φυσικά αλλαγές στο πλάτος και στη συχνότητα του ημιτόνου που μας δίνει τα δείγματα καθυστέρησης, σημαίνουν και αλλαγές στον τρόπο που ακούγεται το εφέ. Έχουμε διαμορφώσει δύο διαφορετικές υλοποιήσεις, μία για καθυστέρηση από 230 έως 430 δείγματα και με συχνότητα ημιτόνου στα 5Hz, και η δεύτερη από 0 έως 50 δείγματα με συχνότητα ημιτόνου 10Hz.

Ο χώρος που καταλαμβάνει το εφέ vibrato (για την πρώτη υλοποίηση).

```
Logic Utilization:
  Number of Slice Flip Flops:           756 out of  27,392    2%
  Number of 4 input LUTs:               664 out of  27,392    2%
Logic Distribution:
  Number of occupied Slices:            711 out of  13,696    5%
     Number of Slices containing only related logic:       711 out of
711 100%
     Number of Slices containing unrelated logic:            0 out of
711   0%

  Total Number of 4 input LUTs:         812 out of  27,392    2%
    Number used as logic:               457
    Number used as a route-thru:        148
    Number used as Shift registers:     207
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                     10 out of     136    7%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           35
Peak Memory Usage:  211 MB
```

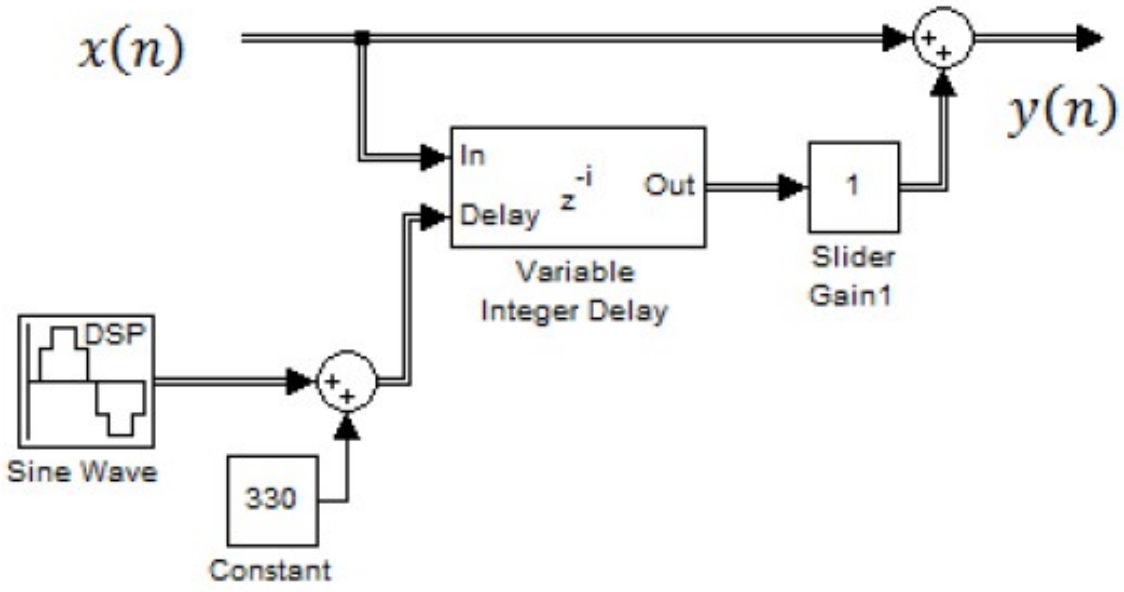

simin
From Workspace
In
Gateway In
System Generator
out
Counter1
a
a - b
b
AddSub1
1
Constant3
0
Constant4
0
Constant2
addra
dina
wea
addrb
dinb
web
A
B
Dual Port RAM
x 1
Dry/wet
cast
Convert
Out
Gateway Out
simout
To Workspace
330
Constant
a
b
a + b
AddSub3
230 to 430
Vibrato.
freq
5
Constant5
freq
sin
Subsystem
x 100
sin from -100 to 100

5.2.4. Flanger

Εφευρέτης της ηχητικής τεχνικής “flanging”, θεωρείται ο Ken Townsend, ο βασικός ηχολήπτης των “EMI Abbey Road Studios”, ο οποίος την επινόησε το 1966. Κατά τη διάρκεια των ηχογραφήσεων του δίσκου “Revolver”, ο John Lennon των Beatles ενθουσιάστηκε τόσο με την νέα αυτή τεχνική του ηχολήπτη του, που της έδωσε το όνομα flanger. Αυτό λοιπόν το όνομα, χωρίς να σημαίνει κάτι, προσδίδεται ακόμα σε αυτό το εφέ, 46 χρόνια μετά. Το πρώτο τραγούδι στο οποίο χρησιμοποιήθηκε ποτέ η τεχνική αυτή, είναι το “Tomorrow never knows”. Όταν ο δίσκος “Revolver” των Beatles ηχογραφήθηκε και εκδόθηκε το 1966, έφτασε να έχει σε όλα του τα τραγούδια το εφέ flanger. Είναι μία από τις πολλές παρακαταθήκες που άφησαν οι Beatles.

Το flanger είναι ένα ηχητικό εφέ το οποίο αποτελεί μία μίξη δύο ίδιων ηχητικών σημάτων, με τη διαφορά ότι το δεύτερο είναι συνεχώς καθυστερημένο με μία περιοδικά μεταβαλλόμενη καθυστέρηση, όπως ακριβώς στο ηχητικό εφέ vibrato που είδαμε παραπάνω.

Η ειδοποιός διαφορά με το vibrato effect είναι πως τα δύο σήματα, το αρχικό και το τελικό, συνδυάζονται (προστίθενται) πριν την έξοδο. Αυτό δημιουργεί εντελώς άλλη αίσθηση στο άκουσμα του τελικού σήματος ήχου. Επίσης, για την πιο “επιτυχή” εμφάνιση του εφέ, πρέπει να αλλάζουν και οι παράμετροι της υλοποίησης, όπως τα Hz της συχνότητας του ημιτόνου που θα αλλάζει περιοδικά την καθυστέρηση του σήματός μας, όπως επίσης και ο μέγιστος αριθμός δειγμάτων καθυστέρησης.

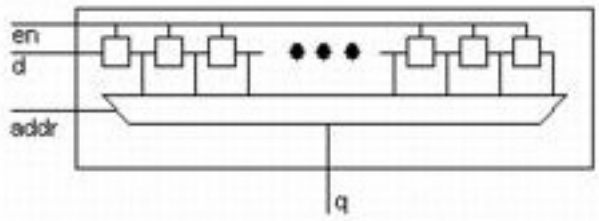


Εικόνα 5.7: Το σχηματικό διάγραμμα του εφέ flanger.

5.2.4.A. Η υλοποίηση σε hardware

Θα μπορούσαμε να υλοποιήσουμε αυτό το εφέ ακριβώς όπως ακριβώς και το vibrato, χρησιμοποιώντας τη dual port ram, αλλά τότε δε θα είχε αξία η αναφορά του, παρά μόνο εγκυκλοπαιδικά. Όμως υλοποιήσαμε το εφέ flanger χρησιμοποιώντας ένα άλλο component, το οποίο μπορεί να κάνει ακριβώς την ίδια δουλειά, πολλές φορές και με λιγότερο κόστος σε χώρο. Ο λόγος για τον addressable shift register.

Ο addressable shift register (προσπελάσιμος καταχωρητής ολίσθησης) είναι ένας καταχωρητής ολίσθησης μεταβλητού μεγέθους, εντός του οποίου οποιοσδήποτε καταχωρητής μπορεί να προσπελαστεί και να δοθεί στην έξοδο. Μπορούμε να το σκεφτούμε σαν μία αλυσίδα από καταχωρητές, οι οποίοι είναι συνδεδεμένοι με έναν πολυπλέκτη ισάριθμων εισόδων, όπως φαίνεται στο σχήμα. Για το ποιος καταχωρητής της σειράς θα επιλεχθεί αποφασίζει η πύλη εισόδου addr. Η έξοδος φαίνεται ως q.



Εικόνα 5.8: Ο addressable shift register.

Άρα μπορούμε να εκμεταλλευτούμε τη λειτουργία του addressable shift register για να υλοποιήσουμε την μεταβαλλόμενη καθυστέρηση που χρειαζόμαστε στο σήμα ήχου μας. Το δείγμα του ήχου μας που θα εισέλθει μια χρονική στιγμή σε αυτό το component, θα περάσει από τόσους καταχωρητές όσους είναι το μέγεθος του addressable shift register. Σε περίπτωση που η address επικαλεστεί ένα δείγμα το οποίο βρίσκεται σε μία θέση, στην ουσία αυτό θα δοθεί στην έξοδο καθυστερημένο για τόσους χτύπους ρολογιού όσους του υποδείξει η τιμή του addr.

Έαν λοιπόν στην είσοδο addr αυτού του component δώσουμε μία σταθερή τιμή, τότε θα πάρουμε το απλό echo effect που έχουμε περιγράψει σε προηγούμενο κεφάλαιο. Σε περίπτωση που δώσουμε ακέραιες τιμές οι οποίες αυξομειώνονται περιοδικά σε ένα συγκεκριμένο εύρος, έχουμε το ίδιο ακριβώς αποτέλεσμα με αυτό στο vibrato effect.

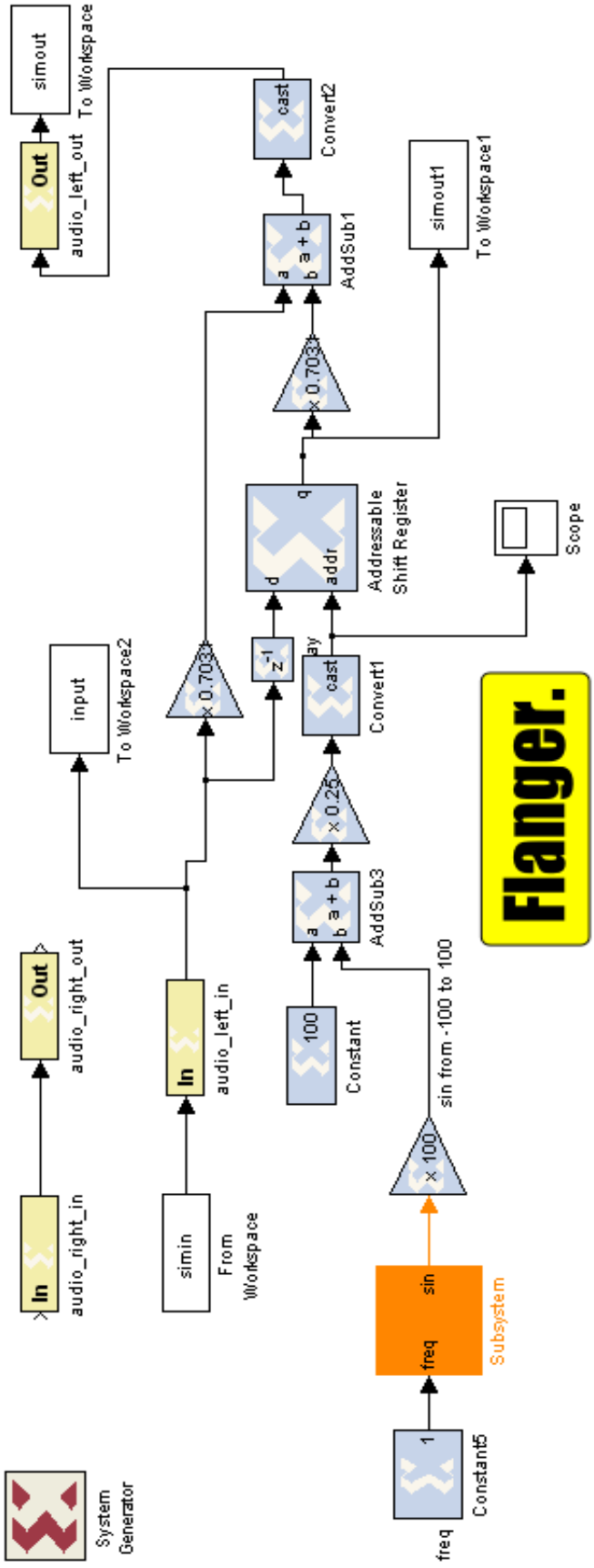
Όπως είπαμε και προηγουμένως, η ειδοποιός διαφορά των εφέ vibrato και flanger είναι πως στο δεύτερο υπάρχει και μία μίξη αρχικού και τελικού σήματος. Αυτή λοιπόν η μίξη, είναι μία απλή πρόσθεση, της οποίας έχει προηγηθεί μία “ενίσχυση” (στην ουσία είναι αποδυνάμωση) των δύο σημάτων, για να μην έχει το τελικό μας σήμα πολύ μεγάλο πλάτος σε σχέση με το αρχικό, άρα και πολύ μεγαλύτερο σε ένταση. Στη σχεδίασή μας έχουμε επιλέξει μια αποδυνάμωση και των δύο σημάτων κατά 0.7.

Αξίζει εδώ να σημειωθεί, πως οι παράμετροι που χρησιμοποιήσαμε για την υλοποίηση αυτού του εφέ φυσικά και μπορούν να αλλάζουν, ανάλογα με την θέληση του χρήστη κάθε φορά. Τέτοιες είναι το εύρος των καθυστερημένων δειγμάτων, η συχνότητα της περιοδικής μεταβολής τους (κοινώς τα Hz του ημιτόνου), και η ενίσχυση αρχικού/επεξεργασμένου σήματος που συμβαίνει στο τέλος. Επίσης, το αποτέλεσμα του εφέ μπορεί να ακούγεται διαφορετικά (στο αφτί), ανάλογα με την πηγή του ήχου που χρησιμοποιήσαμε. Ενδεικτικά, αλλιώς ακούγεται το flanger effect σε ένα σήμα ήχου από φωνητική πηγή, αλλιώς από πνευστό όργανο, και αλλιώς από έγχορδο όργανο. Η τελική αίσθηση βεβαίως παραμένει η ίδια. Έχουμε πραγματοποιήσει δύο διαφορετικές υλοποιήσεις στην FPGA, η μία από 0 μέχρι 50 δείγματα και συχνότητα ημιτόνου στο 1Hz, και η δεύτερη από 0 έως 200 δείγματα με συχνότητα στα 0.5Hz.

Ο κώδικας MATLAB για αυτό το εφέ φυσικά δε διαφέρει από αυτόν του vibrato, με τη διαφορά ότι γίνεται μία απλή πρόσθεση αρχικού και επεξεργασμένου σήματος στο τέλος.

Ο χώρος που καταλαμβάνει το εφέ flanger.

```
Logic Utilization:
  Number of Slice Flip Flops:           762 out of  27,392    2%
  Number of 4 input LUTs:             2,380 out of  27,392    8%
Logic Distribution:
  Number of occupied Slices:          1,585 out of  13,696   11%
     Number of Slices containing only related logic:    1,585 out of
1,585 100%
     Number of Slices containing unrelated logic:           0 out of
1,585   0%

  Total Number of 4 input LUTs:       2,535 out of  27,392    9%
    Number used as logic:             1,149
    Number used as a route-thru:        155
    Number used as Shift registers:   1,231
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      9 out of     136    6%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           35
Peak Memory Usage:  229 MB
```


Flanger.
System Generator
In
audio_right_in
Out
audio_right_out
simin
From Workspace
In
audio_left_in
100
Constant
a
b
a + b
AddSub3
× 0.25
cast
Convert1
× 0.703
z -1
input
To Workspace2
× 0.703
d
addr
q
Addressable Shift Register
a
b
a + b
AddSub1
cast
Convert2
simout
To Workspace
Out
audio_left_out
simout1
To Workspace1
Scope
× 100
sin from -100 to 100
freq
sin
Subsystem
1
Constant5
freq

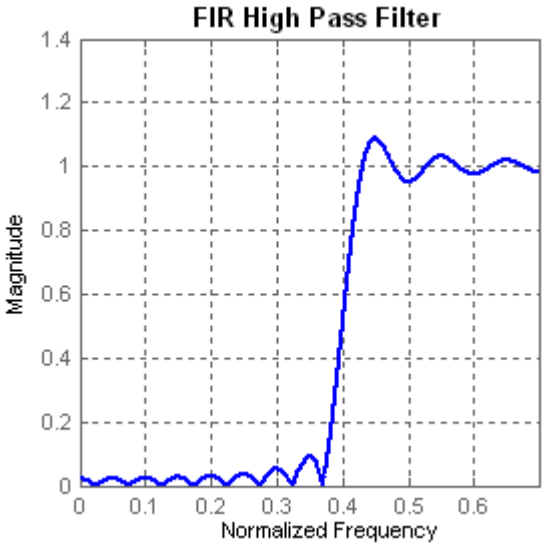
5.3. Το ψηφιακό φίλτρο

5.3.1. Εισαγωγή

Στην ηλεκτρονική και την επιστήμη των υπολογιστών, το ψηφιακό φίλτρο είναι ένα σύστημα, το οποίο εκτελεί μαθηματικές πράξεις και διαδικασίες σε ένα ψηφιακό διακριτό σήμα, με σκοπό να μειώσει ή να ενισχύσει συγκεκριμένες "πτυχές" και χαρακτηριστικά του σήματος. Στην ουσία μετασχηματίζει μία ακολουθία δειγμάτων σε μία άλλη, η οποία αποτελεί και το σήμα εξόδου.

Με τα ψηφιακά φίλτρα μπορούμε να απομακρύνουμε ανεπιθύμητα κομμάτια του σήματός μας (όπως πχ. ο θόρυβος) και να κρατήσουμε ή να ενισχύσουμε την επιθυμητή και χρήσιμη πληροφορία του.

Παρ' όλο που στον αναλογικό κόσμο κανένα ψηφιακό φίλτρο δεν έχει το ακριβώς ανάλογό του, μπορεί και πραγματοποιεί (τις περισσότερες φορές και καλύτερα) όλες τις λειτουργίες, γι' αυτό και τα ψηφιακά φίλτρα υπερτερούν. Προτιμώνται επίσης για ένα σωρό από λόγους. Πραγματοποιούνται με ολοκληρωμένα κυκλώματα, ο επαναπρογραμματισμός τους είναι εύκολος, το χαμηλό τους κόστος (δεν απαιτούν ολόκληρο κύκλωμα από αντιστάσεις, πυκνωτές, και τελεστικούς ενισχυτές), η ευκολία και η ευελιξία τους στο σχεδιασμό και στη δοκιμή, η μεταβλητότητα και η ανθεκτικότητα είναι οι βασικότεροι λόγοι προτίμησής τους.



Εικόνα 5.9: Η απόκριση συχνότητας ενός υψηπερατού φίλτρου.

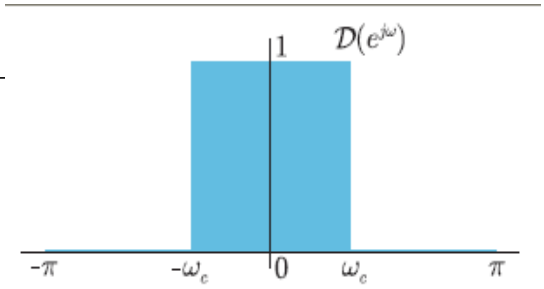
Ένα ψηφιακό φίλτρο προγραμματίζεται, δηλαδή η λειτουργία του καθορίζεται από ένα πρόγραμμα στη μνήμη ενός επεξεργαστή. Για αυτό το λόγο με τη χρήση των ψηφιακών φίλτρων, μπορούμε να αντιμετωπίσουμε σημαντική καθυστέρηση, η οποία είναι συνάρτηση της πολυπλοκότητας της επεξεργασίας μας.

Στην ψηφιακή επεξεργασία ηχητικών σημάτων, τα ψηφιακά φίλτρα χρησιμοποιούνται κατά κόρον. Τέτοιες είναι η ενίσχυση, η απομόνωση συγκεκριμένου εύρους συχνοτήτων από ένα ηχητικό σήμα και η απόρριψη άλλων, η καθυστέρηση του σήματος, και άλλες.

5.3.2. Είδη κλασικών ψηφιακών φίλτρων

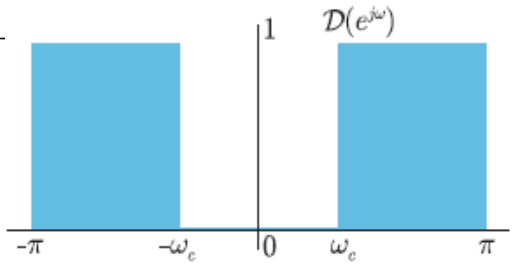
Τα πλέον συνηθισμένα φίλτρα που χρησιμοποιούνται στην επεξεργασία ήχου και μουσικής στην πράξη, είναι φίλτρα, τα οποία, είτε αποκόπτουν εντελώς, είτε αφήνουν αναλλοίωτες κάποιες συχνότητες. Διακρίνουμε τα ακόλουθα είδη ψηφιακών φίλτρων, τα οποία, αν και παρουσιάζονται σαν ψηφιακά, έχουν τα ακριβή αντίστοιχά τους και στον αναλογικό κόσμο.

Το κατωπερατό ή κατωδιαβατό φίλτρο (low pass filter).
Είναι τα φίλτρα τα οποία αποκόβουν όλες τις συχνότητες πέρα από μία τιμή. Η τιμή αυτή ονομάζεται συχνότητα αποκοπής Fc.



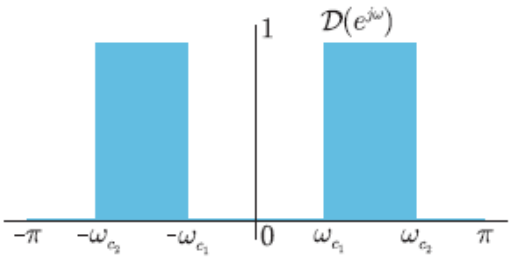
Εικόνα 5.10: Low-pass filter.

Το ανωπερατό ή ανωδιαβατό φίλτρο (high pass filter).
Κάνει στην ουσία την αντίθετη δουλειά από το κατωπερατό. Αφήνει να περάσουν όλες οι συχνότητες που είναι μεγαλύτερες από μία συγκεκριμένη συχνότητα αποκοπής Fc, και μηδενίζει τις υπόλοιπες.



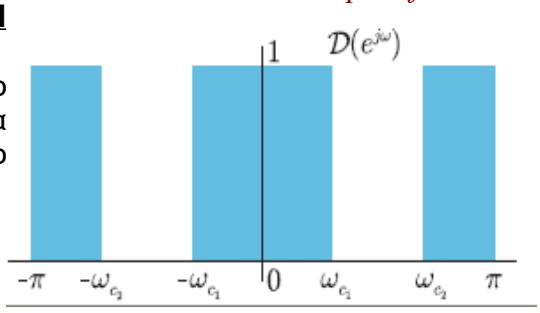
Εικόνα 5.11: High-pass filter.

Το ζωνοπερατό ή ζωνοδιαβατό φίλτρο (band pass filter).
Αφήνει να περάσει μία συγκεκριμένη ζώνη συχνοτήτων μεταξύ δύο συγκεκριμένων συχνοτήτων αποκοπής, Fc_1 και Fc_2.



Εικόνα 5.12: Band-pass filter.

Φίλτρο αποκοπής ζώνης (band stop filter).
Κάνει την αντίθετη δουλειά από το ζωνοπερατό φίλτρο. Αποκόβει μία συγκεκριμένη ζώνη μεταξύ δύο συχνοτήτων αποκοπής, Fc_1 και Fc_2.



Εικόνα 5.13: Band-stop filter.

Αυτά είναι και τα βασικότερα είδη ψηφιακών φίλτρων που χρησιμοποιούνται στην επεξεργασία ήχου και μουσικής. Η χρήση τους ποικίλλει ανάλογα πάντα με το τι έχει να κάνει ο χρήστης. Τα παραπάνω φίλτρα ανήκουν στην κατηγορία “Πεπερασμένης Κρουστικής Απόκρισης” (Finite Impulse Response).

Εδώ αξίζει να αναφερθούμε στη φυσική σημασία της συχνότητας, και στο τι εννοούμε όταν λέμε ότι ένα φίλτρο “αποκόβει” ή “κρατάει” συχνότητες.

Σίγουρα ο όρος συχνότητα μαρτυρά κάτι περιοδικό. Όλοι οι ήχοι που ακούμε γύρω μας είναι περιοδικά σήματα. Ο,τιδήποτε δεν έχει περιοδικότητα τότε είναι απλά ένας θόρυβος. Ο ήχος που κινείται σε μεγάλες/υψηλές συχνότητες είναι πιο “λεπτός” στο αφτί, πιο πρίμος. Αντίστοιχα, στις μικρές/χαμηλές συχνότητες είναι οι πιο μπάσοι, πιο “χοντροί” ήχοι για το αφτί μας. Το ανθρώπινο αφτί μπορεί να αντιληφθεί ήχους από 20Hz έως 20KHz περίπου (γι'αυτό και η δειγματοληψία από αναλογικό σε ψηφιακό γίνεται σχεδόν πάντα στα 44100Hz, για να πληροί το κριτήριο Nyquist).

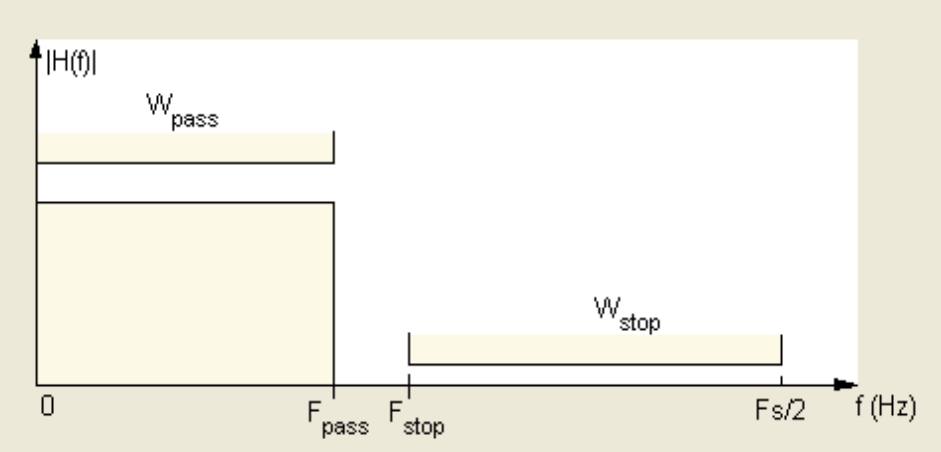
Για παράδειγμα, το εύρος συχνοτήτων της ανθρώπινης φωνής κυμαίνεται περίπου από 85 έως 300 Hz, όταν ο άνθρωπος είναι ενήλικας και ομιλεί σε ήρεμο τόνο. Σε άλλες συνθήκες (όπως πχ. τραγούδι) η ανθρώπινη φωνή μπορεί να φτάσει και τα 3000Hz. Αντίστοιχο εύρος συχνοτήτων έχουν και όλα τα μουσικά όργανα, όπως πχ μια κιθάρα από 80 έως 1000Hz περίπου, και ένα πεντάχορδο μπάσο από τα 25Hz έως τα 280.

Άρα λοιπόν, με τη χρήση των ψηφιακών φίλτρων που περιγράψαμε παραπάνω, μπορούμε να αποκόψουμε μη επιθυμητές ζώνες συχνοτήτων, να ενισχύσουμε άλλες που μας ενδιαφέρουν, κοκ. Το σύστημα το οποίο αποτελείται από διάφορα κατωπερατά, υψιπερατά, και ζωνοπερατά φίλτρα και έχει τη δυνατότητα να αυξομειώνει εντάσεις συχνοτήτων σε σήματα ήχου είναι το λεγόμενο equalizer (ισοσταθμιστής). Με αυτό το εργαλείο μπορεί ο χρήστης και “χρωματίζει” τον ήχο όπως εκείνος θέλει.

5.3.3. Το δικό μας απλό equalizer, και η σχεδίασή του

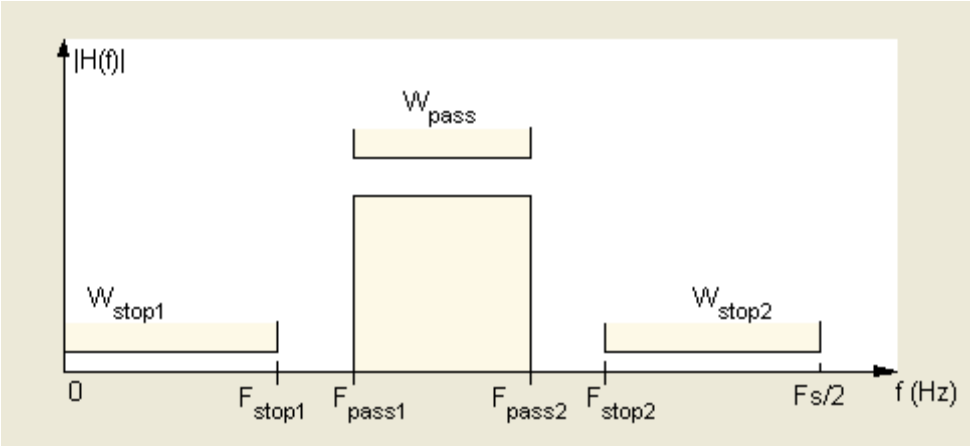
Για μία πολύ απλή σχεδίαση ενός equalizer χρησιμοποιήσαμε τρία φίλτρα, ένα κατωπερατό, ένα ζωνωπερατό και ένα ανωπερατό.

Το κατωπερατό φίλτρο κρατάει τις συχνότητες από 0 έως 500 Hz (μπάσα).



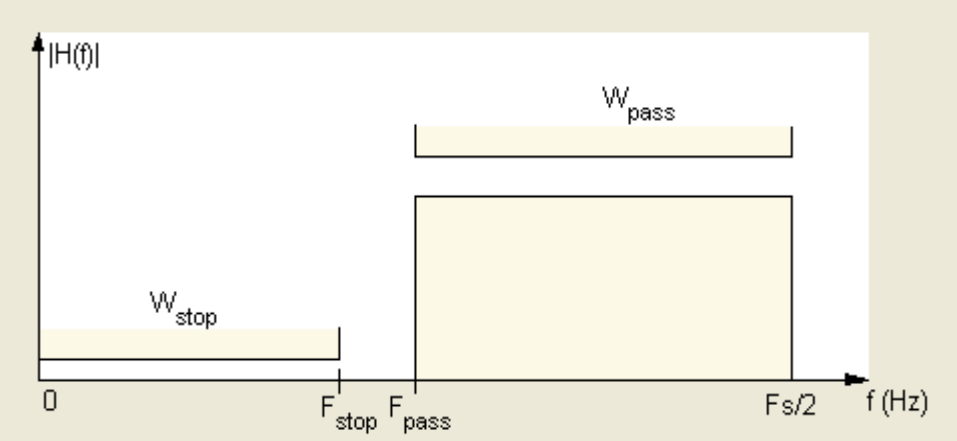
Εικόνα 5.14: Low-pass filter για Fpass = 500Hz.

Το ζωνοπερατό φίλτρο κρατάει τις συχνότητες από 500 έως 1500Hz (μεσαία).



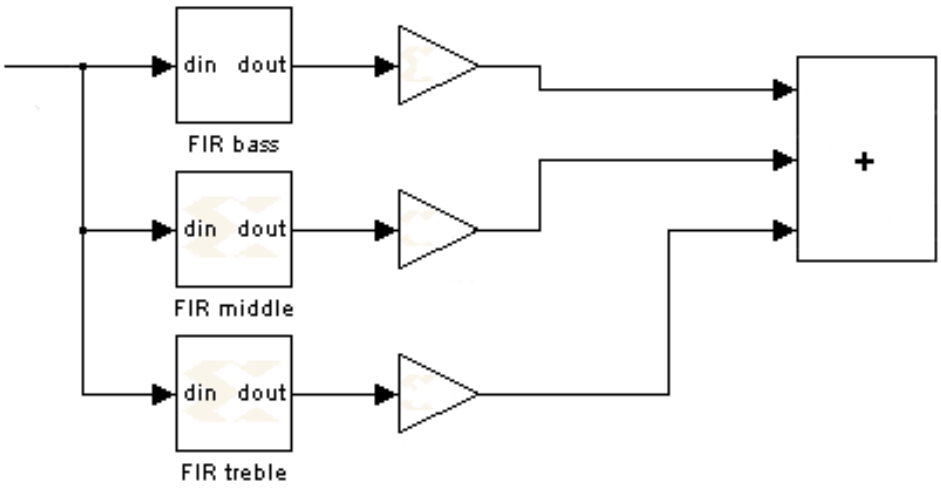
Εικόνα 5.15: Band-pass filter για Fpass1 = 500Hz και Fpass2 = 1500Hz.

Το υψιπερατό φίλτρο κρατάει τις συχνότητες από 1500Hz και άνω (πρίμα).



Εικόνα 5.16: High-pass filter για Fpass = 1500Hz.

Η έξοδος καθενός από αυτά τα τρία φίλτρα περνά από έναν ενισχυτή. Στη συνέχεια αυτά τα τρία σήματα προστίθενται σε ένα, το οποίο βγαίνει στην έξοδο του συστήματος. Η τιμή της ενίσχυσης που θα δοθεί στην έξοδο του κάθε φίλτρου είναι που θα "χρωματίσει" και τον ήχο ανάλογα. Αν, για παράδειγμα, η τιμή της ενίσχυσης του πρώτου φίλτρου είναι μεγάλη (>1) και τα υπόλοιπα δύο φίλτρα αποδυναμωθούν στο μισό, ο ήχος θα βγει αισθητά πιο μπάσος, καθώς θα έχουν ενδυναμωθεί οι χαμηλές συχνότητες, από 0 έως 500 Hz.

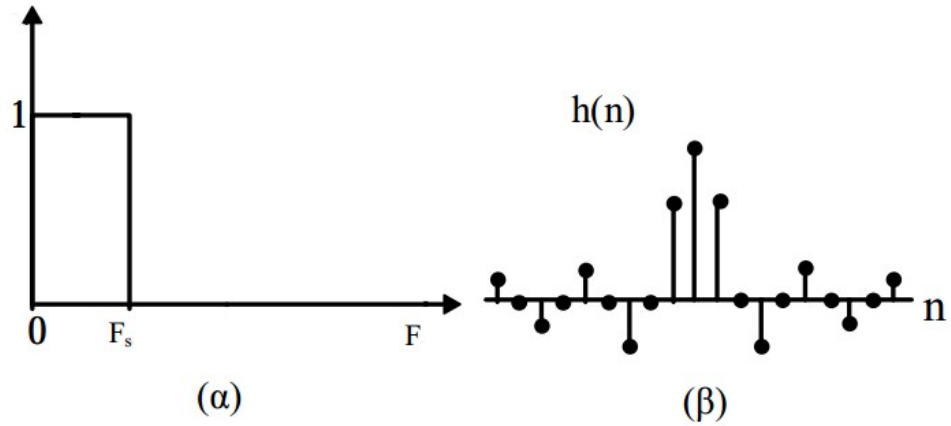


Εικόνα 5.17: Ένα απλό equalizer.

5.3.4. Οι συντελεστές του φίλτρου (filter coefficients)

Το βασικότερο πλεονέκτημα των ψηφιακών φίλτρων, αυτό που τους προσδίδει την μοναδικότητά τους αλλά και την τεράστια ευελιξία τους στην υλοποίηση, είναι οι συντελεστές των φίλτρων. Κάθε ψηφιακό φίλτρο έχει μία και μοναδική αναπαράσταση από ένα διάνυσμα συντελεστών.

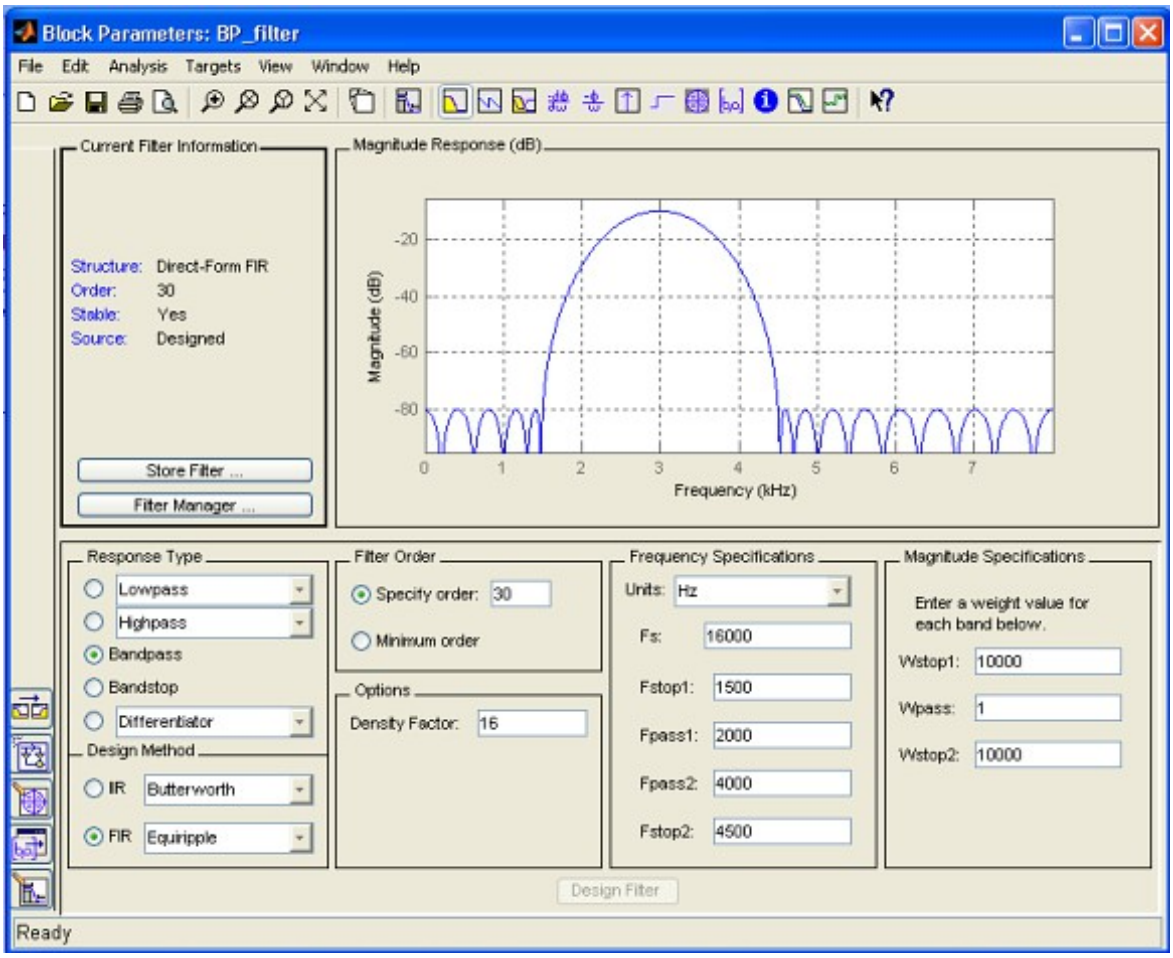
Στα σχήματα δίπλα από τα φίλτρα που περιγράψαμε παραπάνω, φαίνεται η απόκριση συχνότητάς τους. Σε κάθε μία αντιστοιχεί και η κρουστική απόκριση του φίλτρου, η οποία αποτελείται από διάφορες διακριτές τιμές. Αυτές οι τιμές είναι λοιπόν οι συντελεστές του φίλτρου, που αποδίδουν και στο φίλτρο μοναδικά χαρακτηριστικά.



Εικόνα 5.17: α) Η απόκριση συχνότητας ενός φίλτρου και β) η αντίστοιχη κρουστική απόκριση με τους συντελεστές του φίλτρου.

5.3.5. Η χρήση του FDA Tool της MATLAB

Το FDA Tool της MATLAB είναι ένα πολύ χρήσιμο εργαλείο που μας δίνει τη δυνατότητα να σχεδιάσουμε με γρήγορο τρόπο ένα ψηφιακό FIR ή IIR φίλτρο μόνο με την εισαγωγή διαφόρων χαρακτηριστικών που επιθυμούμε για το φίλτρο. Μας δίνει τη δυνατότητα να εξάγουμε κατευθείαν στο workspace της MATLAB ένα διάνυσμα που περιέχει τους συντελεστές του φίλτρου που έχουμε σχεδιάσει.

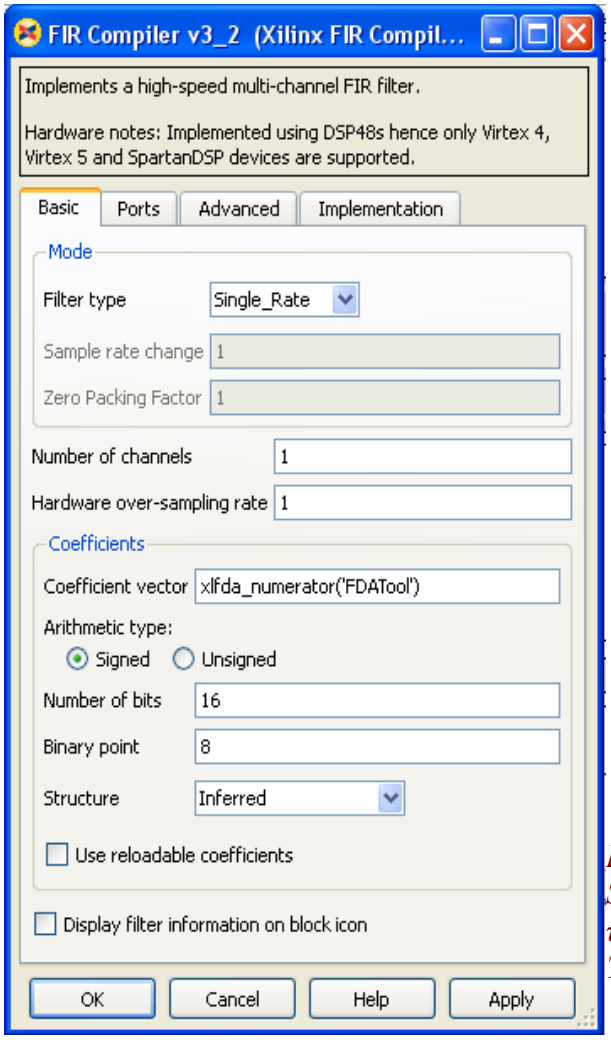


Εικόνα 5.19: Το FDA tool της MATLAB.

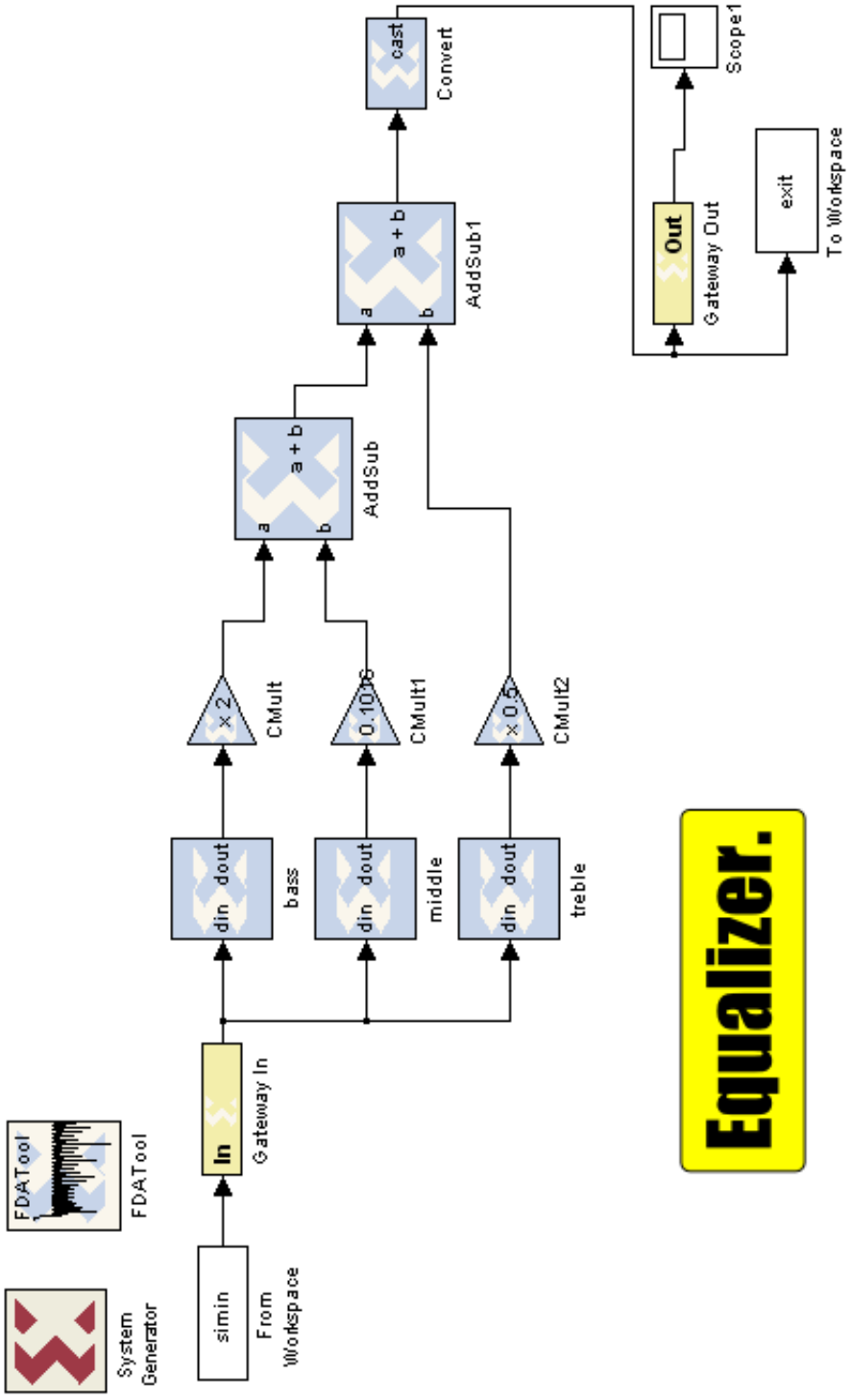
Έτσι λοιπόν, για τη σχεδίαση του equalizer προηγουμένως, μπορέσαμε και εξαγάγαμε με τη χρήση του FDA Tool τους επιθυμητούς συντελεστές για τα φίλτρα που χρησιμοποιήσαμε στη σχεδίασή μας. Για λόγους σύντομης επεξεργασίας για κάθε φίλτρο επιλέξαμε ένα διάνυσμα αποτελούμενο από 96 συντελεστές. Το block FIR compiler μας δίνει τη δυνατότητα να εισάγουμε κατευθείαν τους συντελεστές, και αυτό απλώς να αναλάβει τη συνέλιξή τους με το σήμα ήχου που μπαίνει σαν είσοδος.

Ο χώρος που καταλαμβάνει η σχεδίαση του equalizer.

```
Logic Utilization:
  Number of Slice Flip Flops:        15,612 out of  27,392   56%
  Number of 4 input LUTs:            11,160 out of  27,392   40%
Logic Distribution:
  Number of occupied Slices:          8,621 out of  13,696   62%
     Number of Slices containing only related logic:     8,621 out of
8,621 100%
     Number of Slices containing unrelated logic:            0 out of
8,621   0%

  Total Number of 4 input LUTs:      11,755 out of  27,392   42%
    Number used as logic:            10,781
    Number used as a route-thru:        595
    Number used as Shift registers:     379
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      5 out of     136    3%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           34
Peak Memory Usage:  386 MB
```



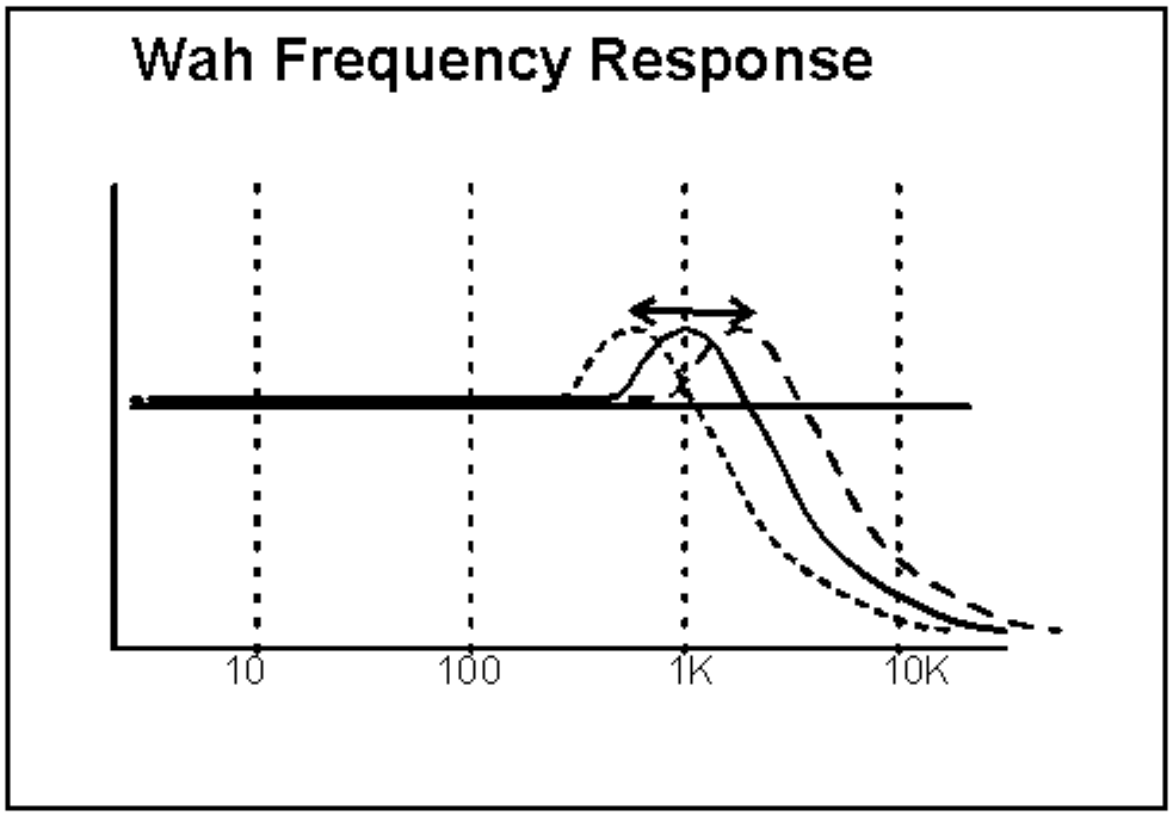
Εικόνα 5.20: Το block FIR Compiler του System Generator και η επιλογή εισαγωγής των εξαγώμενων coefficients από το FDA Tool.


Equalizer.
System Generator
FDATool
FDATool
simin
From Workspace
In
Gateway In
din dout
bass
din dout
middle
din dout
treble
x 2
CMult
0.1016
CMult1
x 0.5
CMult2
a + b
AddSub
a + b
AddSub1
cast
Convert
Out
Gateway Out
Scope1
exit
To Workspace

5.4. Το wah effect

Το εφέ wah ονομάστηκε έτσι γιατί ακριβώς μετατρέπει τον ήχο ώστε να ακούγεται όπως μία ανθρώπινη φωνή η οποία προφέρει τη λέξη “wah”. Πρόκειται για μια ολίσθηση ενός ζωνωπερατού φίλτρου σε ένα εύρος συχνοτήτων από τις χαμηλές στις υψηλές, κοινώς από τις μπάσες στις πρίμες, και πίσω.

Σαν αίσθηση ήχου συναντάται από πιο παλιά, όταν τρομπετίστες χρησιμοποιούσαν την τεχνική “mute” για να δώσουν έμφαση στον ήχο τους, στα ξεκινήματα της jazz μουσικής το 1920. Αργότερα υλοποιήθηκε και ηλεκτρονικά σαν ηχητικό εφέ, με τη μορφή πεταλιού. Στο οποίο πετάλι, η τέρμα κάτω θέση του σημαίνει ότι το ζωνωπερατό φίλτρο βρίκεται στο άκρο των χαμηλών συχνοτήτων, και ή τέρμα πάνω θέση σημαίνει για το φίλτρο ότι βρίσκεται στο άκρο των υψηλών. Τα δύο άκρα είναι από τα 200 έως τα 3000 Hz περίπου. Έτσι ο χρήστης έχει τη δυνατότητα να “ολισθαίνει” το φίλτρο ανάμεσα σε αυτό το εύρος συχνοτήτων. Από το 1950 μέχρι και σήμερα, το εφέ αυτό χρησιμοποιείται ευρέως από πολλούς μουσικούς, και κυρίως από ηλεκτρικούς κιθαρίστες.



Εικόνα 5.21: Απόκριση συχνότητας του Wah effect. Το ζωνοπερατό φίλτρο πρέπει να μεταβάλλεται συνεχώς.

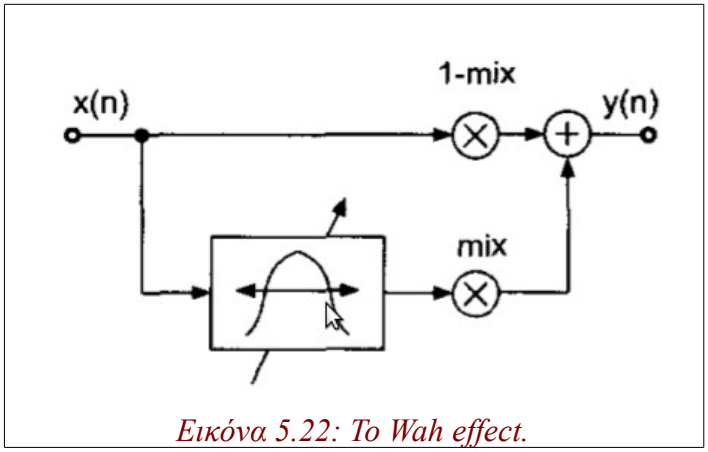
5.4.1. Ο κώδικας MATLAB

Φυσικά στον δικό μας αλγόριθμο δεν έχουμε κάτι το οποίο χειροκίνητα να μας υποδεικνύει σε ποια “θέση” θα βρίσκεται το φίλτρο κάθε φορά, οπότε το θέσαμε σε κίνηση αυτόματα.

Ο αλγόριθμός μας σε MATLAB, αφού κάνει import ένα αρχείο ήχου μέσω της εντολής wavread, δέχεται ορισμένες παραμέτρους. Τέτοιες είναι ο damping factor, που υποδηλώνει πόσο “στενό” θα είναι το ζωνοπερατό φίλτρο, η χαμηλότερη και η υψηλότερη συχνότητα ανάμεσα στις οποίες θα κινείται, και η συχνότητα κίνησής του. Στη συνέχεια ο αλγόριθμός

μας χωρίζεται σε τρεις βασικές λειτουργίες. Πρώτον, δημιουργεί ένα διάνυσμα (Fc) με τις τιμές που θα παίρνει η κεντρική συχνότητα του φίλτρου. Αξίζει να σημειώσουμε εδώ ότι αυτές τις τιμές θα μπορούσε να μας τις υποδείξει οποιοδήποτε περιοδικό σήμα. Στον συγκεκριμένο αλγόριθμο έχει επιλεχθεί ένας τριγωνικός παλμός, αλλά θα μπορούσε κάλλιστα να είναι ένα ημίτονο ή οτιδήποτε άλλο. Το περιοδικό σήμα επαναλαμβάνεται για όλο το μήκος του σήματος εισόδου. Δεύτερον, κατασκευάζει το ζωνοπερατό φίλτρο (χρησιμοποιώντας για κεντρική συχνότητα την πρώτη τιμή του διανύσματος Fc). Τρίτον σε μια δομή επανάληψης επανακατασκευάζει το ζωνοπερατό φίλτρο κατά μήκος του σήματος εισόδου, σε κάθε επανάληψη. Φυσικά κάθε επανάληψη θα έχει διαφορετική τιμή κεντρικής συχνότητας Fc για το φίλτρο, οπότε και θα απομονώνει διαφορετική ζώνη (band) συχνοτήτων.

Στο σήμα εξόδου πλέον έχει εφαρμοστεί το wah effect.



Εικόνα 5.22: Το Wah effect.

5.4.2. Η υλοποίηση σε hardware

Σε αντίθεση με τον αλγόριθμό μας σε MATLAB, θα ήταν αδύνατο να κατασκευάσουμε τόσα πολλά διαφορετικά φίλτρα σε hardware. Κάτι τέτοιο θα είχε πολύ μεγάλο κόστος και σε επεξεργασία και σε "χώρο".

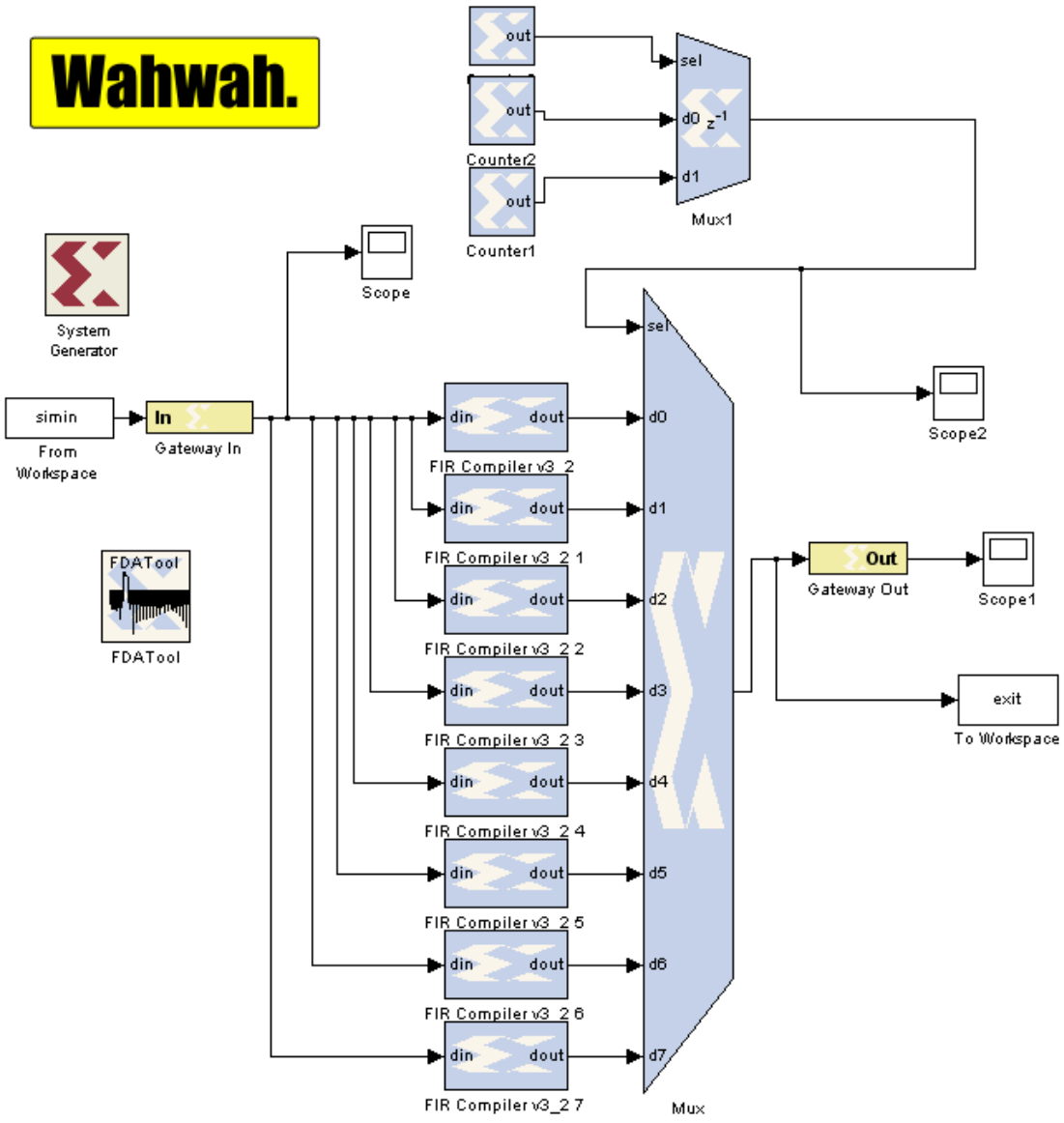
Μία πρώτη προσέγγιση που κάναμε για να υλοποιήσουμε αυτό το εφέ, είναι η εξής: Κατασκευάσαμε με χρήση του FDA Tool της MATLAB 8 διαφορετικά ζωνοπερατά φίλτρα, "απλωμένα" στις συχνότητες από 100 έως 4000Hz. Όλα αυτά παίρνουν σαν είσοδο το αρχικό μας σήμα. Η έξοδος καθενός από αυτά τα 8 φίλτρα καταλήγει σε έναν πολυπλέκτη 8 διαφορετικών εισόδων. Στην ουσία στις 8 αυτές εισόδους του πολυπλέκτη εισέρχονται 8 "φιλτραρισμένες εκδοχές" του σήματός μας. Για το ποια θα είναι αυτή που θα επιλεχθεί στην έξοδο, αποφασίζει η πύλη "select" του πολυπλέκτη. Στην πύλη select εισέρχεται ένας αριθμός μεγέθους τριών bits ο οποίος μετρά από το 0 στο 7 και τούμπαλιν. Αυτός αλλάζει τιμή ανά 4000 samples, άρα κάθε 4000 samples του σήματός μας φιλτράρονται με διαφορετικό φίλτρο.

Λόγω του ότι δεν είναι ένα μοναδικό ζωνοπερατό φίλτρο το οποίο ταλαντώνεται στη συχνότητα, αλλά είναι 8 διαδοχικά (και στην ουσία "ταλαντώνεται" ο πολυπλέκτης που αποφασίζει ποιο θα στείλει), στο τελικό σήμα που παίρνουμε στην έξοδο είναι πολύ πιθανό να ακουστούν "ασυνέχειες" στη συχνοτική μεταβολή του σήματος, αισθητές στο αφτί. Η αλλαγή στον ήχο γίνεται απότομα και "κοφτά". Ένα πιο ταιριαστό όνομα σε αυτό το εφέ όπως το υλοποιήσαμε, θα ήταν το "quantized wah".

Δυστυχώς, λόγω χωρητικότητας της FPGA που χρησιμοποιήσαμε, δεν ήταν εφικτό να κατασκευάσουμε το quantized wah με 8 διαφορετικά FIR φίλτρα. Οι υψηλές απαιτήσεις των FIR φίλτρων σε πολυπλοκότητα πράξεων, μας περιορίζουν στο να χρησιμοποιήσουμε μόνο 5, και αυτά με 49 coefficients το καθένα. Αυτά δυστυχώς είναι πολύ λίγα σε σχέση με αυτά που απαιτούνται για να έχω το τέλειο αποτέλεσμα. Όταν λοιπόν αυτή η υλοποίηση "κατέβει" στην FPGA φαίνεται χαρακτηριστικά το απότομο πέρασμα από τη μία ζώνη συχνοτήτων στην άλλη. Όπως φαίνεται και παρακάτω, η υλοποίηση με τα 5 FIR φίλτρα, καταλαμβάνει το 99% των λογικών slices που είναι διαθέσιμα στην Virtex II Pro.

Ο χώρος που καταλαμβάνει το εφέ wah.

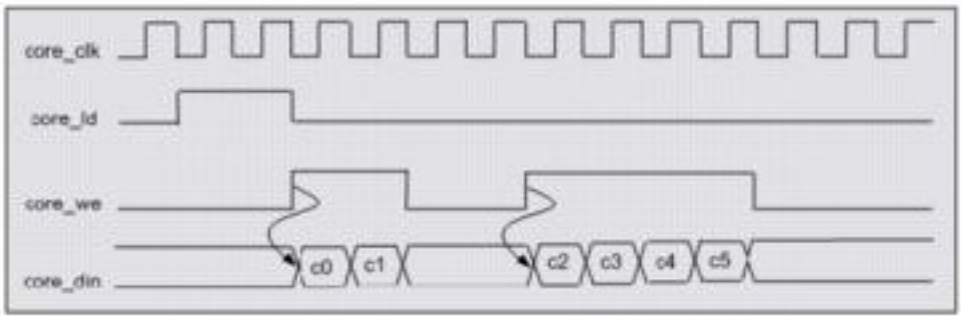
```
Logic Utilization:
  Number of Slice Flip Flops:        25,270 out of  27,392   92%
  Number of 4 input LUTs:            17,923 out of  27,392   65%
Logic Distribution:
  Number of occupied Slices:         13,626 out of  13,696   99%
     Number of Slices containing only related logic:   13,626 out of
13,626 100%
     Number of Slices containing unrelated logic:             0 out of
13,626   0%

  Total Number of 4 input LUTs:      18,808 out of  27,392   68%
    Number used as logic:            17,436
    Number used as a route-thru:        885
    Number used as Shift registers:     487
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                      6 out of     136    4%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           32
Peak Memory Usage:  480 MB
```


Wahwah.
out
sel
out
d0 z-1
Counter2
d1
out
Mux1
Counter1
Scope
System Generator
sel
simin
In
From Workspace
Gateway In
din
dout
d0
FIR Compiler v3_2
d1
FIR Compiler v3_2 1
d2
FIR Compiler v3_2 2
d3
FIR Compiler v3_2 3
d4
FIR Compiler v3_2 4
d5
FIR Compiler v3_2 5
d6
FIR Compiler v3_2 6
d7
FIR Compiler v3_2 7
Mux
Scope2
FDATool
FDATool
Out
Gateway Out
Scope1
exit
To Workspace

5.4.3. Το "quantized wah" σε μία υλοποίηση χαμηλότερου κόστους

Το block "FIR Compiler" του System Generator, μας δίνει τη δυνατότητα να χρησιμοποιήσουμε την επιλογή "reloadable coefficients". Όπως εξηγήσαμε και παραπάνω, οι συντελεστές (coefficients) ενός ψηφιακού φίλτρου είναι αυτοί που του προσδίδουν τα χαρακτηριστικά του. Για κάθε μοναδικό σύνολο συντελεστών υπάρχει και ένα μοναδικό ψηφιακό φίλτρο. Μπορούμε λοιπόν να εκμεταλλευτούμε τη δυνατότητα αυτή που μας δίνει το πρόγραμμά μας, και να προσδίδουμε ανά τακτά χρονικά διαστήματα, διαφορετικούς συντελεστές στο ίδιο FIR φίλτρο της σχεδίασής μας. Έτσι, προκύπτει σημαντική εξοικονόμιση χώρου και χρόνου επεξεργασίας στο σύστημά μας.

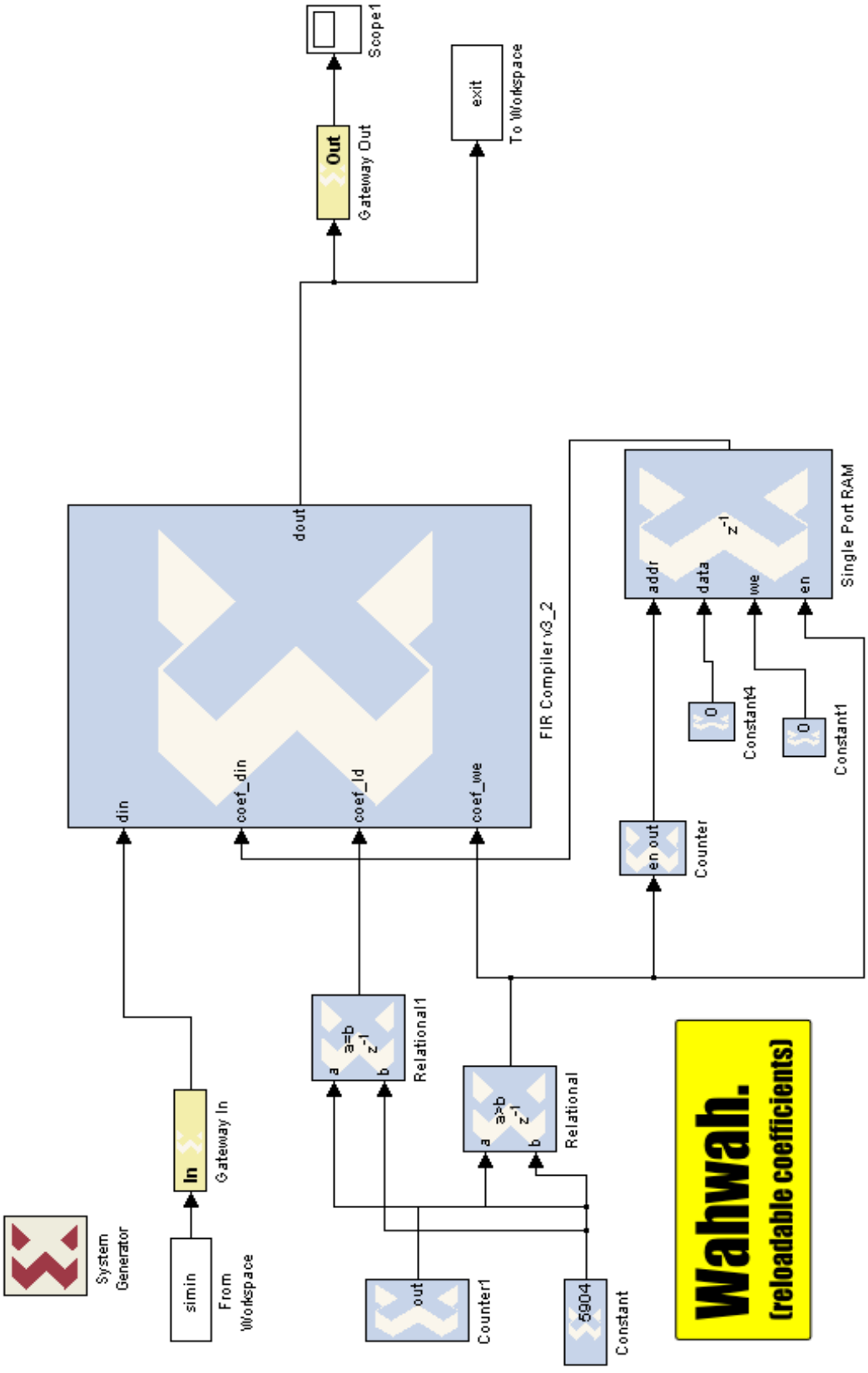
Όταν είναι ενεργοποιημένη η επιλογή των "reloadable coefficients" στο component αυτό ανοίγουν τρεις νέες πύλες στις οποίες χρειάζεται να δώσουμε παραμέτρους. Η πύλη coeff_in είναι αυτή που δέχεται τις καινούριες τιμές των συντελεστών του φίλτρου. Όσο το σήμα coeff_ld είναι 1, τότε δεν μπορεί να γίνει καμία εγγραφή καινούριων συντελεστών. Από τη στιγμή που γίνει 0, τότε το σήμα coeff_we μπορεί οποιαδήποτε στιγμή να γίνει 1, και να σημάνει την αρχή της "φόρτωσης" καινούριων συντελεστών. Επίσης υπάρχει η δυνατότητα όλοι οι συντελεστές να μην εισαχθούν συνεχόμενοι στο φίλτρο (πράγμα που δε χρειάστηκε). Με το που το σήμα coeff_ld ξαναγίνει 1, σημαίνει ότι έχει εισαχθεί το καινούριο "σετ" συντελεστών στο φίλτρο μας, και η λειτουργία του πλέον γίνεται με τους καινούριους συντελεστές.



Εικόνα 5.22: Τα σήματα και ο τρόπος λειτουργίας του block FIR Compiler με την επιλογή "reloadable coefficients".

Σε αυτή τη σχεδίαση χρησιμοποιήσαμε και μία single port RAM για να αποθηκεύσουμε τα σετ των συντελεστών που χρειαζόμαστε. Κάθε σετ αποτελείται από 96 συντελεστές. Άρα εάν θέλουμε να αλλάζει το φίλτρο μας κάθε 4000 δείγματα, θα πρέπει στο σημείο 3904 το σήμα coeff_ld να γίνει 0 και να ενεργοποιηθεί το coeff_we. Τότε είναι που θα διαβαστούν από τη μνήμη τα επόμενα 96 στοιχεία, να φορτωθούν στο φίλτρο μας έτσι ώστε στο σημείο 4000 το coeff_ld να ξαναγίνει 1 και να αρχίσει πλέον να λειτουργεί το φίλτρο μας με τους καινούριος συντελεστές.

Αυτός ο τρόπος υλοποίησης του εφέ μας δίνει ακριβώς τα ίδια αποτελέσματα. Αν λάβουμε όμως υπ' όψιν μας ότι τα FIR φίλτρα είναι από τα πιο "ακριβά" components σε πύλες, χώρο, και χρόνο επεξεργασίας, τότε το κέρδος μας σε κόστος είναι πολύ σημαντικό.


Scope1
Out
Gateway Out
exit
To Workspace
dout
FIR Compiler v3_2
din
coef_din
coef_ld
coef_we
addr
data
we
en
Single Port RAM
0
Constant4
0
Constant1
en out
Counter
a
a>b
z-1
b
Relational1
Relational
In
Gateway In
simin
From Workspace
System Generator
out
Counter1
5904
Constant
Wahwah.
(reloadable coefficients)

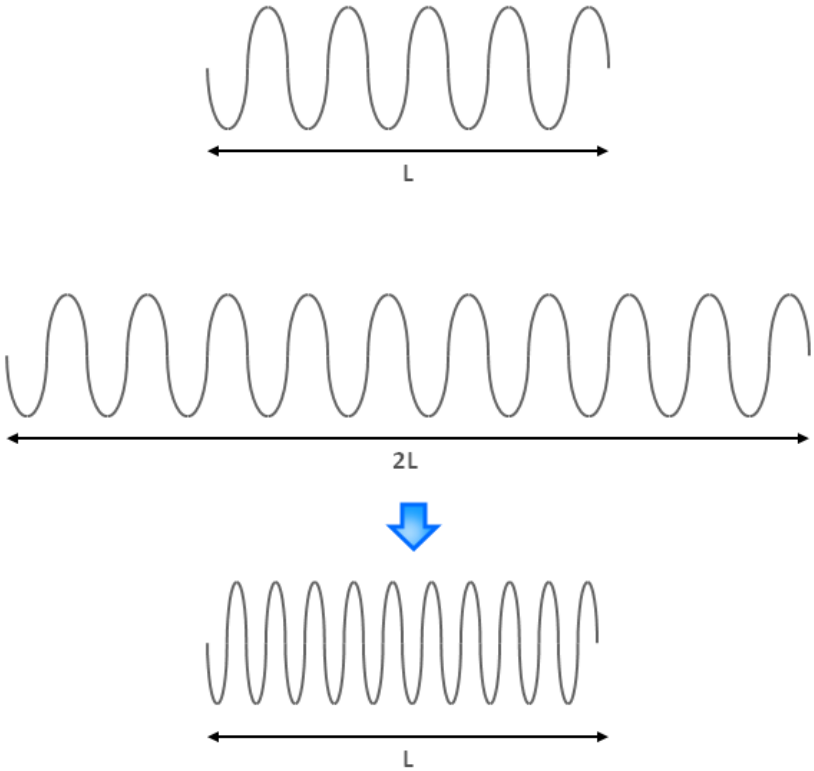
5.5. Το pitch shifting effect

Το pitch shifting είναι μία ηχητική τεχνική κατά την οποία ο τόνος (pitch) του ήχου μπορεί να αλλάζει, προς τα πάνω ή προς τα κάτω. Με άλλα λόγια, είναι η τεχνική εκείνη που κάνει τον ήχο να γίνεται “πιο λεπτός” ή “πιο χοντρός”. Αποτελεί ίσως έναν από τους δυσκολότερους αλγορίθμους στην παραγωγή ήχου και μουσικής. Το εφέ αυτό μπορεί να επιτευχθεί είτε στη διάσταση του χρόνου (time domain pitch shifting) είτε στη συχνότητα (frequency domain pitch shifting).

Υπάρχουν διάφοροι τύποι αλγορίθμων οι οποίοι το επιτυγχάνουν, καθένας από αυτούς κατάλληλος για ήχο με διαφορετικό “χαρακτήρα” (άλλου τύπου ήχος η ανθρώπινη φωνή, άλλο ένα μουσικό όργανο).

Ο πιο απλός τρόπος να αλλάξουμε τον τόνο ενός ήχου είναι με το να τον αναπαράγουμε πιο γρήγορα ή πιο αργά. Αν αυξήσουμε τη διάρκεια του ηχητικού μας σήματος τότε ο ήχος “βαραίνει” και γίνεται πιο “χοντρός”, αν τη μειώσουμε τότε συμβαίνει το αντίθετο. Αυτή όμως η μέθοδος για να αλλάξουμε το pitch του ήχου έχει δύο ελαττώματα τόσο βασικά, που ουσιαστικά την καθιστούν άχρηστη. Πρώτον, δε θα μπορούσε ποτέ να λειτουργήσει σε real-time συστήματα, διότι η είσοδος και η έξοδος έχουν διαφορετικό μέγεθος. Δεύτερον, ενώ θα μπορούσε κανείς να πει ότι η υλοποίησή της στον αναλογικό κόσμο θα μπορούσε να επιτευχθεί εύκολα (πχ. ένα πικ-απ που κινεί το βινύλιο με μεγαλύτερη-μικρότερη ταχύτητα), στον ψηφιακό κόσμο δε θα είχε καμία χρηστική αξία, καθώς οποιαδήποτε διαδικασία δειγματοληψίας από αναλογικό σε ψηφιακό και αντίστροφα (είτε ηχογράφηση είτε αναπαραγωγή), γίνεται πάντα στα 44100Hz. Από κάρτες ήχου υπολογιστών μέχρι ψηφιακούς δίσκους, όλα γίνονται με sample rate στα 44100Hz.

Έτσι λοιπόν στρεφόμαστε σε αλγόριθμους οι οποίοι επιτυγχάνουν με πιο πολύπλοκο τρόπο την αλλαγή του pitch, κρατώντας -φυσικά- και την έξοδο στο ίδιο μήκος με την είσοδο. Εμείς υλοποιήσαμε σε hardware αυτόν στο πεδίο του χρόνου.



Εικόνα 5.23: Σχηματική απεικόνιση ενός απλού Pitch Shifting αλγορίθμου.

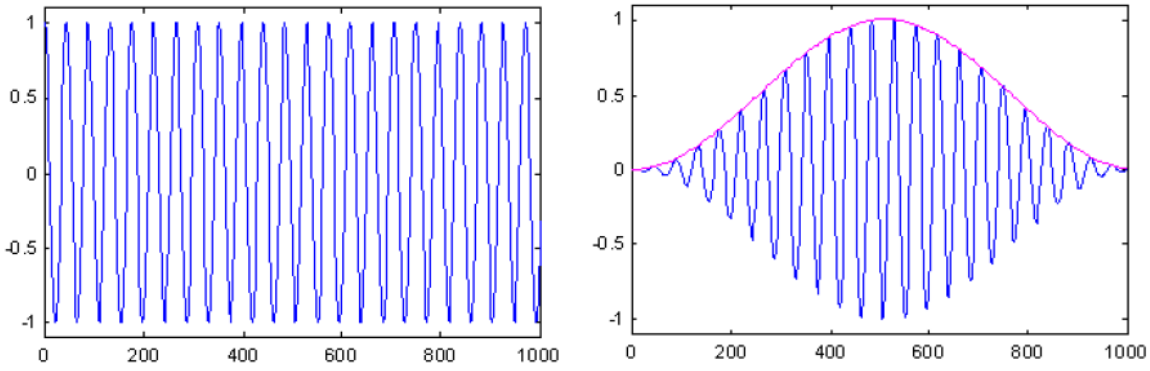
5.5.1. Time domain pitch shifting

Η βασική λογική του αλγορίθμου, είναι ότι προσπαθούμε να αυξήσουμε ή να μειώσουμε το μήκος του σήματός μας, έτσι ώστε με το κατάλληλο resample να πάρουμε μία έξοδο η οποία θα έχει ίδιο με το μήκος εισόδου αλλά διαφορετικό pitch. Στην διαδικασία της αλλαγής του μήκους του σήματός μας, είτε αυτό είναι με επιμήκυνση είτε επιβράχυνση, πρέπει να προσέξουμε αυτή να γίνει με τρόπο έτσι ώστε να έχω τη λιγότερη δυνατή απώλεια πληροφορίας - ποιότητας σήματος.

Ο αλγόριθμος είναι χωρισμένος σε δύο διαδικασίες. Κατά τη διάρκεια της πρώτης διαδικασίας γίνεται το κόψιμο του σήματος εισόδου σε μικρά πλαίσια (frames), τα οποία επικαλύπτονται μεταξύ τους κατά ένα ποσοστό. Στη συνέχεια, κατά τη διάρκεια της δεύτερης διαδικασίας τα πλαίσια επανατοποθετούνται ξανά, όμως με διαφορετική επικάλυψη μεταξύ τους. Εν τω μεταξύ, για καλύτερη και ποιοτικά πιο “λεία”, και “μαλακή” επανατοποθέτησή τους, και για αποφυγή ασυνεχειών και θορύβου, τα πλαίσια αυτά έχουν πολλαπλασιαστεί με ένα παράθυρο hanning.

Το παράθυρο hanning

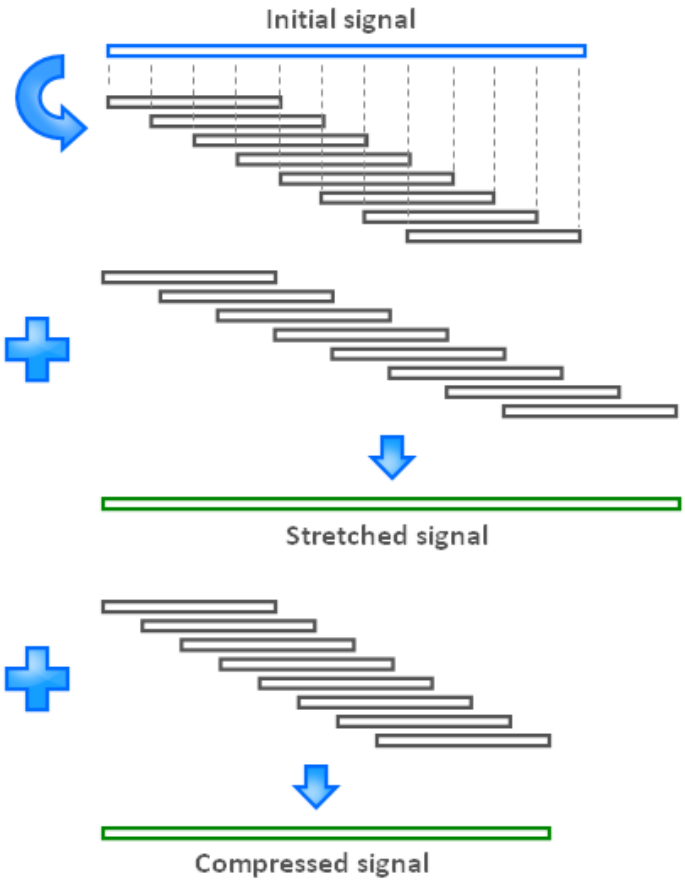
Η βασική χρήση του παραθύρου hanning γίνεται για αναλύσουμε ένα μακρύ σήμα. Η διαδικασία της ανάλυσης των σημάτων είναι ο χωρισμός τους σε διαδοχικά πλαίσια. Όταν τα πλαίσια αυτά επικαλύπτονται, ο πολλαπλασιασμός τους με το παράθυρο hanning είναι που διευκολύνει τη διαδικασία της επικάλυψης, εξομαλύνοντας έτσι τις ασυνέχειες που ενδεχομένως να προκύψουν. Αυτό συμβαίνει διότι η πληροφορία στην οποία θέλουμε να δώσουμε έμφαση, βρίσκεται πάντα προς το κέντρο του παραθύρου. Άρα λοιπόν, εν ολίγοις, το παράθυρο hanning είναι κατάλληλο για την “παραθυροποίηση” ενός μεγάλου σήματος. Ο αλγόριθμος που χρησιμοποιούμε για pitch shifting στο πεδίο του χρόνου, ίσως κάνει κάπως “ανορθόδοξη” χρήση του παραθύρου αυτού, καθώς πολλαπλασιάζει τα παράθυρα που έχουμε με το παράθυρο hanning στο πεδίο του χρόνου, χωρίς καμία μεταβολή στη συχνότητα (η ορθή χρήση του παραθύρου γίνεται στην ανάλυση σημάτων στο πεδίο της συχνότητας, με χρήση μετασχηματισμού Fourier, βλ. Frequency domain pitvh shifting). Παρ' όλα αυτά, χωρίς έστω και με αυτή του τη χρήση, ο θόρυβος στο τελικό μας σήμα λόγω ασυνεχειών είναι πολύ πιο εμφανής.



Εικόνα 5.24: Ημιτονοειδές σήμα και η "παραθυροποίησή" του με Hanning window.

Για να ορίσουμε και ποσοτικά το πόσο θέλουμε να αλλάξει το pitch του ήχου μας, έχουμε θέσει ένα κλάσμα. Στον αλγόριθμο MATLAB είναι η μεταβλητή alpha. Επί της ουσίας, αυτό το κλάσμα μας δείχνει πόσο μεγαλύτερο ή μικρότερο θα γίνει το σήμα μας μετά την δεύτερη διαδικασία επανατοποθέτησης των πλαισίων.

Ακόμα δύο παράμετροι στον αλγόριθμό μας είναι το μέγεθος των frames που θα χωρίσουμε το σήμα μας, και το πόσο αυτά θα επικαλύπτονται μεταξύ τους, το λεγόμενο overlap. Ένα ικανοποιητικό ποσοστό επικάλυψης που χρησιμοποιούμε στον αλγόριθμό μας είναι το 75%, για frames των N=512 samples. Φυσικά ο αλγόριθμος λειτουργεί και για διαφορετικές τιμές αυτών των δύο. Όμως, για πολύ μικρότερο overlapping, μικρότερου του 50%, το σήμα εξόδου αρχίζει και χαλάει αισθητά, λόγω του ότι οι ασυνέχειες -όταν ξαναγίνεται η επικόλληση των frames μεταξύ τους- γίνονται πλέον πολύ πιο εμφανείς, με αποτέλεσμα να δημιουργείται θόρυβος. Για πολύ μεγάλο overlapping (από 90% και πάνω) ο αλγόριθμος λειτουργεί κανονικά, όμως οι διαφορές μεταξύ εισόδου εξόδου δεν είναι και τόσο μεγάλες, οπότε το κλάσμα alpha πρέπει να πάρει πιο ακραίες τιμές (θα εξηγήσουμε το γιατί και στη συνέχεια).



Εικόνα 5.25: Το pitch shifting effect. Επιλογή επικαλυπτώμενων frames από το σήμα, επανατοποθέτησή τους σε διαφορετική θέση, και επαναφορά στο αρχικό μήκος.

Ο αριθμός των samples per frame όπως είναι φυσικό χαλάει την ποιότητα του ήχου εάν πέσει πολύ κάτω από 512, όμως εάν αυξηθεί, ποιοτικά στον αλγόριθμο δεν αλλάζει κάτι, μερικές φορές ίσως και να τον κάνει να δουλεύει καλύτερα. Πολύ μεγάλα frames όμως, σημαίνουν και πολύ χρόνο επεξεργασίας. Όταν δουλεύαμε τον αλγόριθμο σε MATLAB αυτό δεν ήταν πρόβλημα, όταν όμως αυτό περνάει σε hardware και θέλουμε να δουλεύει σε real-time, μπορεί να σημαίνει σημαντικό latency κατά την επεξεργασία.

Με δεδομένο ότι έχω frames των 512 samples και overlap στο 75%, σημαίνει ότι επικαλύπτονται 384 samples μεταξύ δύο διαδοχικών frames. Τα samples που περισσεύουν, δηλαδή τα 128 στην προκείμενη περίπτωση, είναι το λεγόμενο hop.

Μπαίνουμε λοιπόν στη διαδικασία που “κόβουμε” το σήμα εισόδου σε frames. Το πρώτο frame όπως είναι φυσικό θα περιέχει τα δείγματα από 1 μέχρι 512, το δεύτερο από 129 μέχρι 630, κοκ. Αφού λοιπόν πολλαπλασιαστούν με ένα παράθυρο hanning που περιγράψαμε παραπάνω, έρχονται να επανατοποθετηθούν ξανά, αυτή τη φορά όμως με διαφορετικό hop. Στον αλγόριθμο MATLAB τα hop της πρώτης και της δεύτερης διαδικασίας είναι οι μεταβλητές Ra και Rs. Το Rs θα είναι, όπως είναι φυσικό, alpha φορές μεγαλύτερο ή μικρότερο από το Ra. Έτσι, για παράδειγμα, όταν έχω alpha=14/12 το Rs θα είναι ίσο με 149. Αυτή λοιπόν θα είναι η νέα απόσταση μεταξύ των frames. Άρα λοιπόν, τα frames θα επανατοποθετηθούν ως εξής: το πρώτο από 1 έως 512, το δεύτερο από 150 έως 661, κοκ. Κατά τη διάρκεια της επανατοποθέτησης, όταν πάμε να τοποθετήσουμε ένα frame σε “θέση” που έχει ήδη τιμή, τότε προσθέτουμε.

Έτσι λοιπόν έχει προκύψει στην έξοδο ένα σήμα το οποίο είναι alpha φορές το αρχικό μας. Μας μένει λοιπόν, να κάνουμε ένα resampling για να το φέρουμε ακριβώς στο μήκος της εισόδου μας. Το resampling αυτό, θα έχει παραμέτρους ακριβώς αντίστροφες με το κλάσμα alpha. Δηλαδή, εάν το κλάσμα μας είναι 14/12, τότε θα χρειαστεί να κάνουμε resampling με παρεμβολή (interpolation) 12, και με αποδεκάτιση (decimation) 14. Αυτό στη MATLAB μπορεί να γίνει με την εντολή resample πολύ έυκολα. Έτσι επιτυγχάνουμε και η έξοδος πλέον να είναι στο ίδιο μήκος με την είσοδο, αλλά πλέον να έχει διαφορετικό pitch.

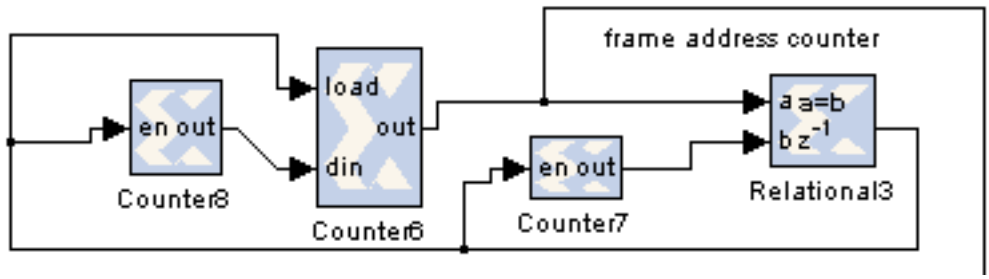
Προφανώς, όταν έχουμε να κάνουμε με πολύ μικρές ή πολύ μεγάλες τιμές του κλάσματος alpha, παρ' όλο που ενδεχομένως ο αλγόριθμος να βγάζει αποτελέσματα, το σήμα ήχου που προκύπτει καταλήγει να μην έχει φυσική σημασία. Αυτό συμβαίνει για τιμές του κλάσματος κοντά στο 0, και για τιμές άνω του 2, λόγω της σχεδόν ολικής ή καθόλου επικάλυψης των πλαισίων αντίστοιχα.

5.5.2. Η υλοποίηση σε hardware

Η λογική της υλοποίησης ενός αλγορίθμου σε hardware είναι πολύ διαφορετική από αυτή του απλού αλγορίθμου σε MATLAB. Ο βασικότερος λόγος για τον οποίο συμβαίνει αυτό είναι πως πλέον έχουμε συνεχές streaming με δεδομένα, και όχι μία “μονοκόμματη” είσοδο, που μπορούμε να την πάρουμε και να την επεξεργαστούμε αυτούσια. Έτσι λοιπόν, λόγω του ότι θέλουμε η σχεδίασή μας να τρέχει real-time, δεν μπορούμε να πάρουμε “ολόκληρη” την είσοδο, να την αποθηκεύσουμε σε μια μνήμη, να την χωρίσουμε σε frames και να κάνουμε την επεξεργασία που θέλουμε, και έπειτα να την δώσουμε στην έξοδο. Χρειάζεται ένας τρόπος να λειτουργεί η σχεδίασή μας και για “άπειρο” μέγεθος εισόδου.

Έτσι λοιπόν, για την υλοποίηση αυτού του αλγορίθμου, χρησιμοποιήσαμε δύο dual port RAM. Αυτού του είδους οι μνήμες μας είναι χρήσιμες σε αυτή την περίπτωση διότι μας επιτρέπουν να έχουμε ανάγνωση και εγγραφή σε δύο διαφορετικές θέσεις μνήμης.

Στην πρώτη μνήμη σκοπός μας είναι στην διεύθυνση A να γράφεται το σήμα ήχου με τον ρυθμό που έρχεται, και από τη διεύθυνση B να διαβάζονται τα frames. Δηλαδή, ο δείκτης της διεύθυνσης B να ξεκινάει από τη 0 έως την 511 θέση μνήμης, μετά από 128 έως 639, κοκ. Για την υλοποίησή αυτή χρησιμοποιήσαμε δύο counters, ο ένας αυξάνεται κανονικά κατά ένα οπότε να γράφει το σήμα στη μνήμη, κι ο άλλος να ακολουθεί την “πορεία” των frames, όπως περιγράψαμε πριν.

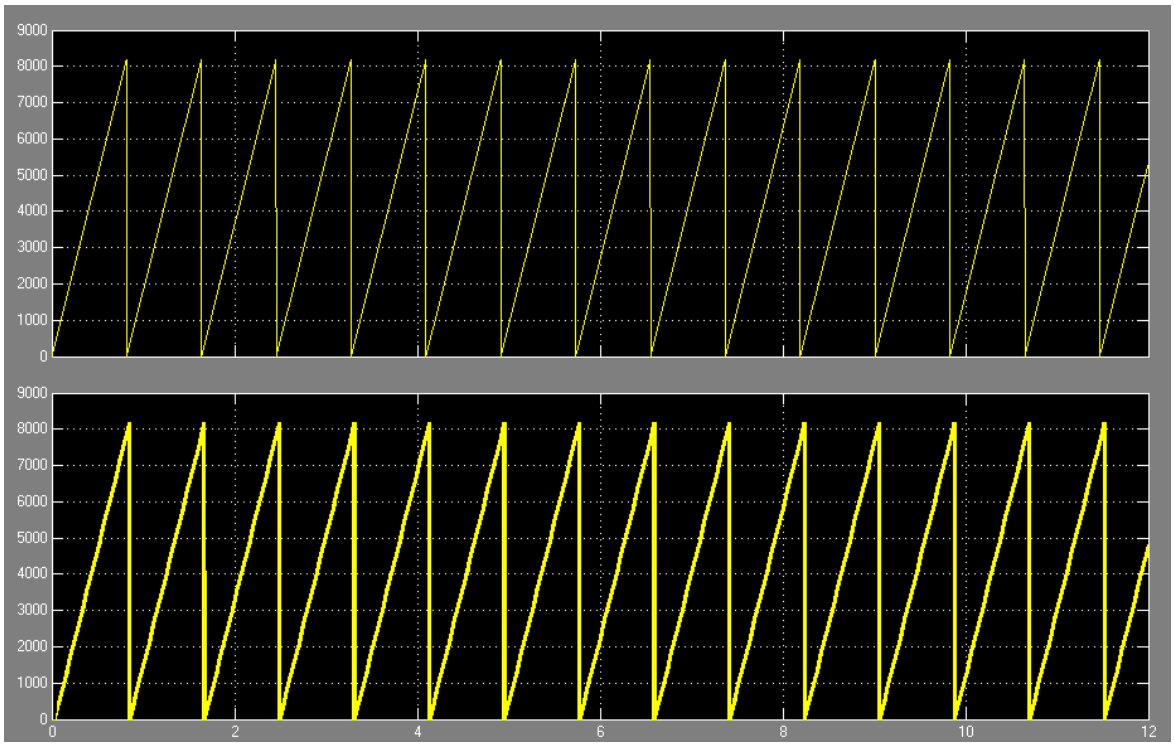


Εικόνα 5.26: Το σχήμα που μετράει τα frames στη μνήμη. Counter6: ο μετρητής που μετράει τα frames. Counter7: ξεκινά από το 512 και μετρά με βήμα Ra (ή Rs αν μιλάμε για τη δεύτερη). Counter 8: Ξεκινά από Ra και έχει βήμα Ra (ή Rs).

Εδώ λοιπόν ήρθαμε αντιμέτωποι με το εξής πρόβλημα: Για δεδομένο πεπερασμένο μέγεθος μνήμης, υπάρχει κίνδυνος να πανωγράψουμε σε θέση μνήμης της οποίας τα δεδομένα να μην έχουν ακόμα αξιοποιηθεί, άρα να έχουμε απώλεια πληροφορίας. Αυτό μπορεί να συμβεί πολύ εύκολα, καθώς ο πρώτος μετρητής που δεν ξαναγυρνάει προηγούμενες θέσεις μνήμης όπως ο δεύτερος, μπορεί εύκολα να τον προφτάσει, έχοντας πρώτα προπελάσει ολόκληρη τη μνήμη.

Το πρόβλημα λοιπόν λύνεται, εάν αυτοί οι δύο μετρητές τρέχουν σε διαφορετικές

“ταχύτητες”. Εάν λάβουμε υπόψιν μας ότι ο πρώτος μετρητής προχωράει κανονικά μία προς μία τις θέσεις μνήμης ενώ ο άλλος κάθε 512 θέσεις μνήμης πάει πίσω 384, πρέπει ο δεύτερος να είναι πιο ταχύς. Στη συγκεκριμένη περίπτωση και για τις παραμέτρους που έχουμε θέσει, εάν είναι 4 φορές πιο ταχύς από τον άλλον, τότε δεν “συναντιούνται” ποτέ. Άρα λοιπόν θέτουμε ο ταχύς μετρητής να αλλάζει τιμή σε κάθε χτύπο του ρολογιού, με τον άλλο μετρητή ανά τέσσερις χτύπους. Αυτό αλλάζει αντίστοιχα, εάν στον αλγόριθμό μας χρησιμοποιήσουμε διαφορετικό hop.



Εικόνα 5.27: Η γραφική παράσταση με τους δύο μετρητές. Ο πρώτος απλά μετράει ανά μήνα τις θέσεις μνήμης για να γραφτεί το σήμα, και ο δεύτερος όντας 4 φορές πιο γρήγορος μετράει frames από τη μνήμη.

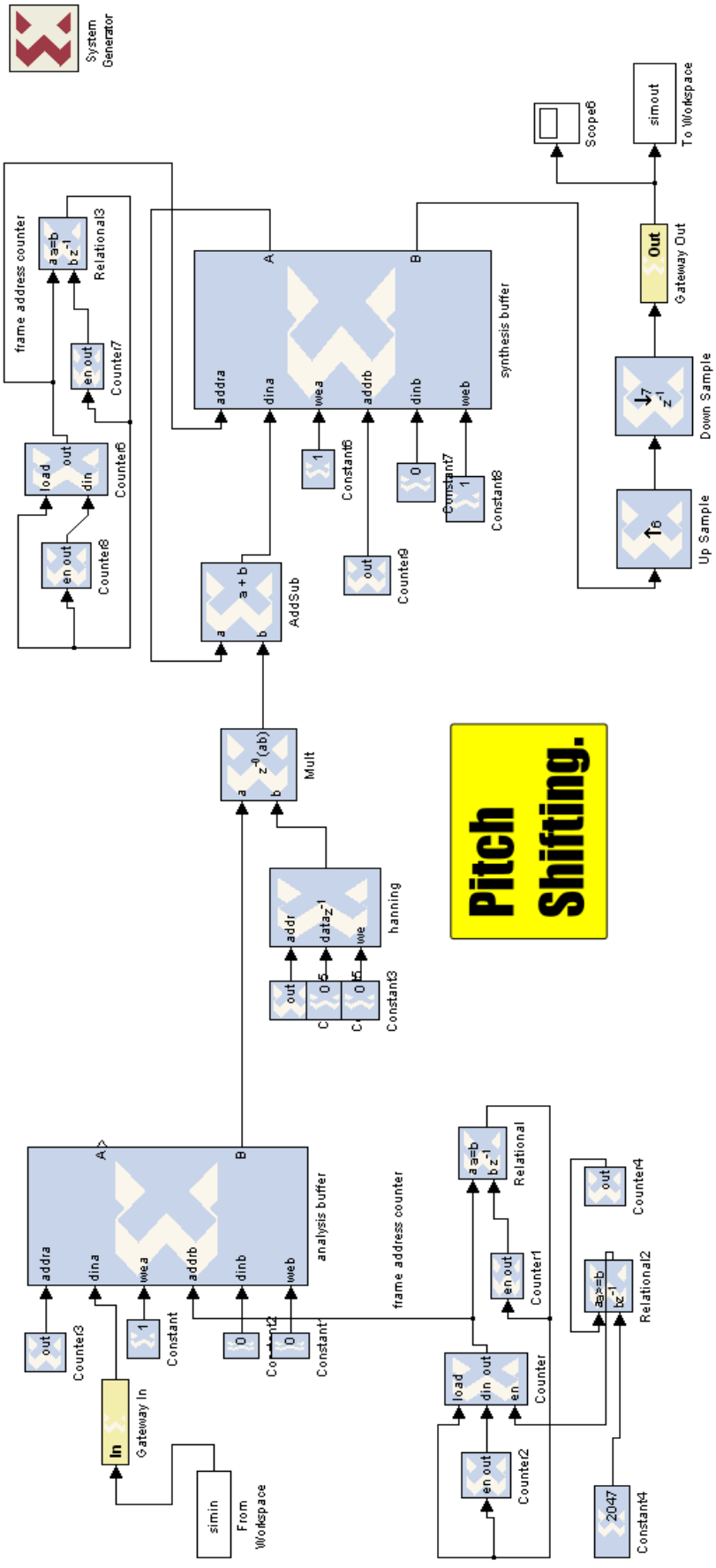
Αντίστοιχα χρησιμοποιούμε τον ίδιο μηχανισμό και όταν είναι να επανατοποθετήσουμε τα frames μεταξύ τους. Και αυτή τη φορά χρησιμοποιούμε μία dual port RAM. Αυτή τη φορά ο μετρητής της διεύθυνσης A θα είναι εκείνος που θα γράφει στη μνήμη, και άρα χρειάζεται να είναι πιο ταχύς για πηγαινοέρχεται στις διευθύνσεις μνήμης που είναι να μπουν τα frames. Ο μετρητής που θα δείχνει στη διεύθυνση B, είναι εκείνος που θα διαβάζει το τελικό μας σήμα έτσι όπως έχει προκύψει. Αυτή τη στιγμή, το σήμα αυτό είναι μεγαλύτερο σε μέγεθος απ' ότι το αρχικό μας. Χρειάζεται λοιπόν να το φέρουμε στο επιθυμητό μέγεθος. Έτσι λοιπόν χρησιμοποιούμε δύο components, το ένα κάνει upsample όσο και ο παρονομαστής του κλάσματός alpha (στην προκείμενη περίπτωση 12) και το άλλο κάνει downsample όσο ο αριθμητής του ίδιου κλάσματος (14).

Έτσι στην έξοδο έχω το τελικό σήμα, μήκους ίδιου με το αρχικό, με αλλαγμένο το pitch.

Σημείωση: *Αυτή η σχεδίαση λειτουργεί κανονικά σε simulation στο περιβάλλον του simulink, όπου και η έξοδος αποθηκεύεται σαν ένα διάνυσμα στο workspace της MATLAB. Όταν κατεβαίνει στην fpga και με δεδομένη τη λειτουργία του AC97 codec, το τελικό resample παύει να έχει νόημα, καθώς ο codec είναι από μόνος του ρυθμισμένος σε έναν συγκεκριμένο χρονισμό. Άρα λοιπόν, όσα δείγματα και να στείλω στην έξοδο, αυτός θα αναπαράγει μόνο αυτά που "έπεσαν" πάνω στο χτύπο του ρολογιού του.*

Ο χώρος που καταλαμβάνει το εφέ Pitch Shifting.

```
Logic Utilization:
  Number of Slice Flip Flops:           897 out of  27,392    3%
  Number of 4 input LUTs:             1,044 out of  27,392    3%
Logic Distribution:
  Number of occupied Slices:            974 out of  13,696    7%
     Number of Slices containing only related logic:       974 out of
974 100%
     Number of Slices containing unrelated logic:            0 out of
974   0%

  Total Number of 4 input LUTs:       1,294 out of  27,392    4%
    Number used as logic:               837
    Number used as a route-thru:        250
    Number used as Shift registers:     207
  Number of bonded IOBs:                  7 out of     556    1%
  Number of RAMB16s:                     50 out of     136   36%
  Number of MULT18X18s:                   2 out of     136    1%
  Number of BUFGMUXs:                     4 out of      16   25%
  Number of DCMs:                         1 out of       8   12%
  Number of BSCANs:                       1 out of       1  100%

  Number of RPM macros:           33
Peak Memory Usage:  221 MB
```


System Generator
Scope6
simout
To Workspace
Out
Gateway Out
Down Sample
Up Sample
frame address counter
Relational3
Counter7
Counter6
Counter8
synthesis buffer
Constant6
Constant7
Constant8
Counter9
AddSub
Mult
hanning
Constant3
Pitch Shifting.
analysis buffer
Counter3
Gateway In
simin
From Workspace
Constant
Constant1
Constant2
frame address counter
Relational
Counter1
Counter
Counter2
Counter4
Relational2
Constant4
2047

5.5.3. Frequency domain pitch shifting

Ένας ακόμα πιο περίπλοκος αλγόριθμος για να πετύχουμε το pitch shifting, είναι αυτός στο πεδίο της συχνότητας. Λόγω της μεγάλης του πολυπλοκότητας και λόγω της εμπλοκής του με τον μετασχηματισμό Fourier και τη συχνότητα, δεν καταφέραμε να τον υλοποιήσουμε σε hardware. Αξίζει όμως να γίνει μια αναφορά.

Η λογική του αλγορίθμου είναι σχεδόν ίδια με αυτή στο πεδίο του χρόνου. Χωρίζουμε πάλι το σήμα μας σε διάδοχικά επικαλυπτόμενα πλαίσια, τα οποία τα πολλαπλασιάζουμε με ένα παράθυρο hanning και υπολογίζουμε στη συνέχεια τον μετασχηματισμό Fourier τους, παίρνοντας έτσι το φάσμα κάθε παραθύρου. Η επανατοποθέτηση των παραθύρων στη διαδικασία της σύνθεσης γίνεται πάλι με την ίδια λογική, όπου και επαναφέρονται στο χρόνο με τη χρήση του αντίστροφου μετασχηματισμού Fourier.

Η μετάβαση όμως στη συχνότητα μας δίνει τη δυνατότητα να χρησιμοποιήσουμε μία πολύ σημαντική τεχνική, αυτή του Phase Propagation (διάδοση φάσης). Σύμφωνα με αυτή, μπορούμε και εξισοροποούμε τις ασυνέχειες στη φάση μεταξύ δύο διαδοχικών παραθύρων. Έτσι μπορεί και διατηρείται πολύ καλύτερα το συχνοτικό περιεχόμενο του σήματός μας. Με τον όρο φάση, εννοούμε τον όρο θ στον εκθέτη του e όταν ο μιγαδικός συμβολίζεται με εκθετική μορφή ($z=re^{j\theta}$).

Ο αλγόριθμος αυτός είναι πάρα πολύ αποτελεσματικός στην επεξεργασία φωνής, καθώς μπορεί και διατηρεί τη “χροιά” της ανθρώπινης φωνής πολύ καλύτερα. Επίσης η τεχνική διάδοσης φάσης είναι πολύ χρήσιμη και για τον κατά κάποιο τρόπο “αντίστροφο” αλγόριθμο του pitch shifting, το time stretching, κατά τον οποίο το σήμα εξόδου διατηρεί το ίδιο pitch, αλλά έχει διαφορετικό μήκος. Στον time stretching αλγόριθμο δεν υπάρχει real-time υπόσταση, καθώς η έξοδος δεν έχει το ίδιο μέγεθος με την είσοδο.

ΚΕΦΑΛΑΙΟ 6

Η υλοποίηση στη Virtex-II Pro FPGA και η αναπαραγωγή ήχου από αυτήν

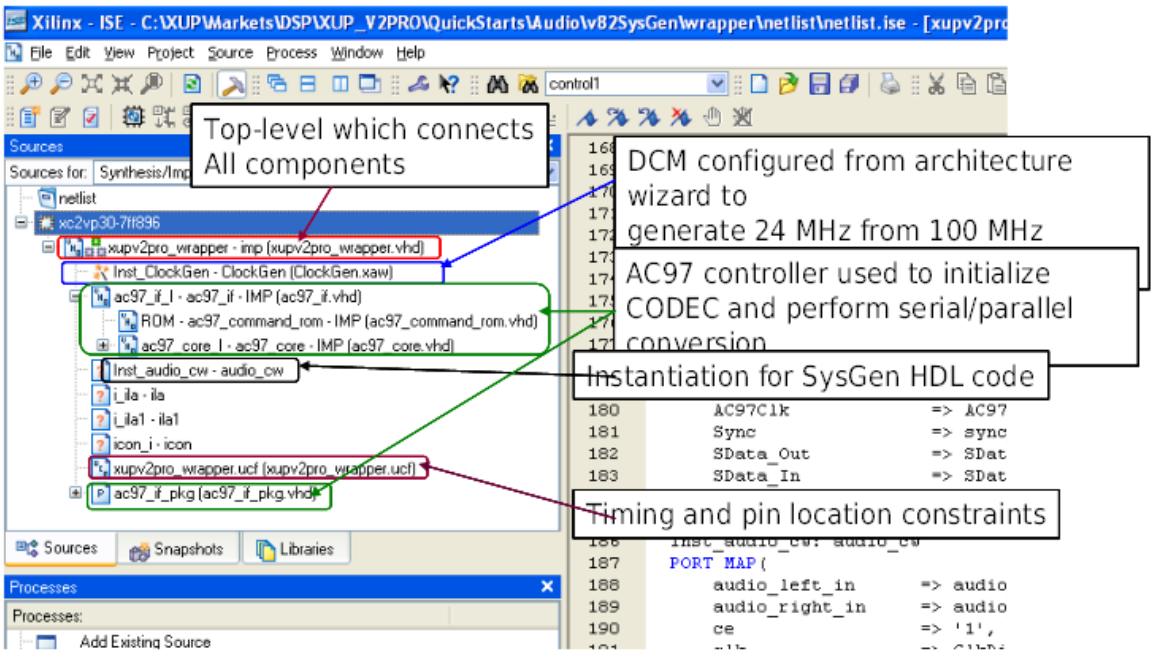
Αξίζει να γίνει μία περιγραφή στον τρόπο με τον οποίο μπόρεσαν οι παραπάνω υλοποιήσεις των ηχητικών εφέ να εφαρμοστούν στην FPGA που χρησιμοποιήσαμε. Στο σημείο που είχαμε έτοιμα τα designs στο System Generator και έτοιμο τον κώδικα VHDL για αυτά, αντιμετωπίσαμε το πρόβλημα του πώς αυτά θα “κατέβουν” στην FPGA.

Ένας από τους λόγους που χρησιμοποιήσαμε το XUP Virtex-II Pro FPGA board, είναι ότι έχει στη διάθεσή του το audio codec AC97' για τη διαχείρηση του ήχου. Αυτό είναι που μπορεί να μας δώσει είσοδο ενός σήματος ήχου δύο καναλιών στο σύστημα μέσω του A/D converter που περιέχει, να μας δώσει τη δυνατότητα να το επεξεργαστούμε, και να το εξάγουμε ξανά στην έξοδο σαν ένα σήμα ήχου μέσω του D/A converter.

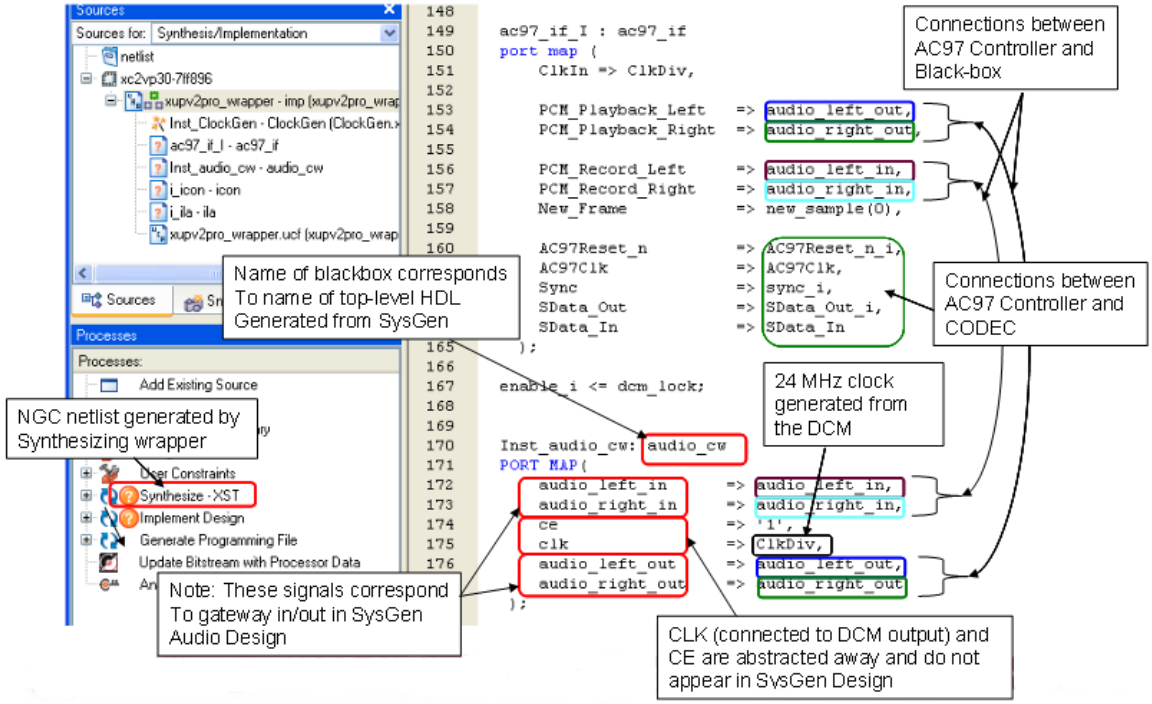
6.1. Ο AC97 codec και η χρήση του

Για να μπορέσουμε να θέσουμε σε λειτουργία τον AC97 codec χρειαστήκαμε ένα interface που να μπορεί να “συνδέει” τη σχεδίασή μας με το αναλογικό σήμα ήχου που δίνουμε στην είσοδο line-in. Αυτή τη δουλειά θα την κάνει για εμάς ένας HDL Wrapper διαθέσιμος στη σελίδα του documentation της Virtex-II pro μαζί με το λεγόμενο Board Support Package (BSP) να τον συνοδεύει.

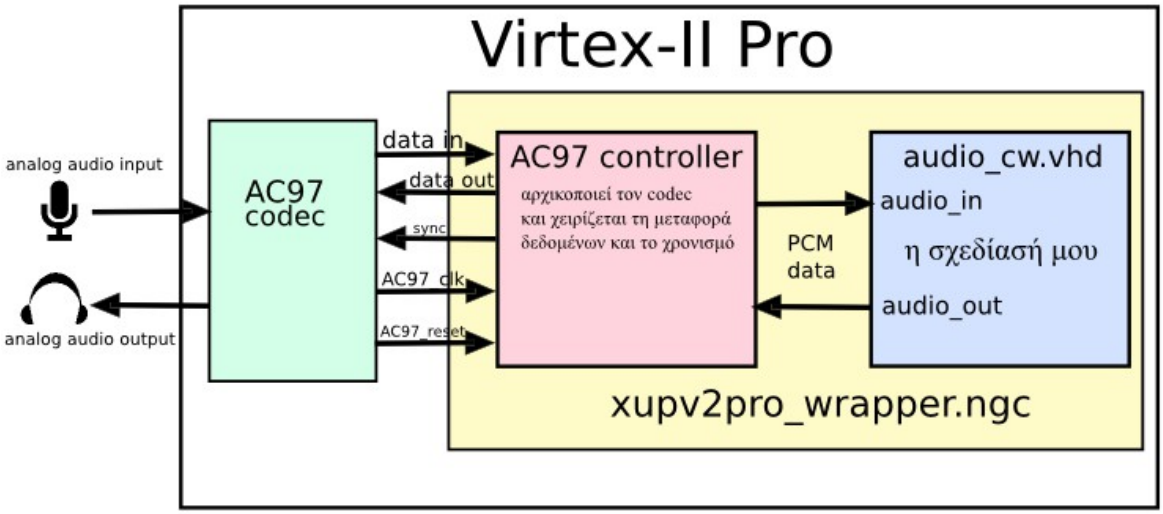
Το Board Support Package πρώτο μας βήμα ήταν να εγκατασταθεί στο directory του System Generator έτσι ώστε να είναι έτοιμο για χρήση. Περιέχει διάφορα αρχεία μεταξύ των οποίων το xupv2pro_wrapper.ngc. Το αρχείο αυτό είναι στην ουσία η σχεδίαση του HDL wrapper η οποία μπορεί και χειρίζεται τον AC97 codec, και το δικό μας design. Αυτός ο wrapper είναι στην ουσία στο Top Level της σχεδίασης, γιατί λαμβάνει/δίνει την συνολικη είσοδο/έξοδο του συστήματος. Η δική μας σχεδίαση είναι ένα component εντός του.



Εικόνα 6.1: Ο HDL Wrapper για το AC97 codec της FPGA. "Τυλίγει" τη σχεδίαση που πρόκειται να φτιάξουμε (audio_cw) για να δώσει είσοδο/έξοδο ήχου στο σύστημα..



Εικόνα 6.2: Ο Top-Level κώδικας του wrapper. Υπάρχουν οι συνδέσεις και για τη σχεδίασή μας, αλλά και για τον codec. Έτσι γίνεται η σύνδεση μεταξύ τους.

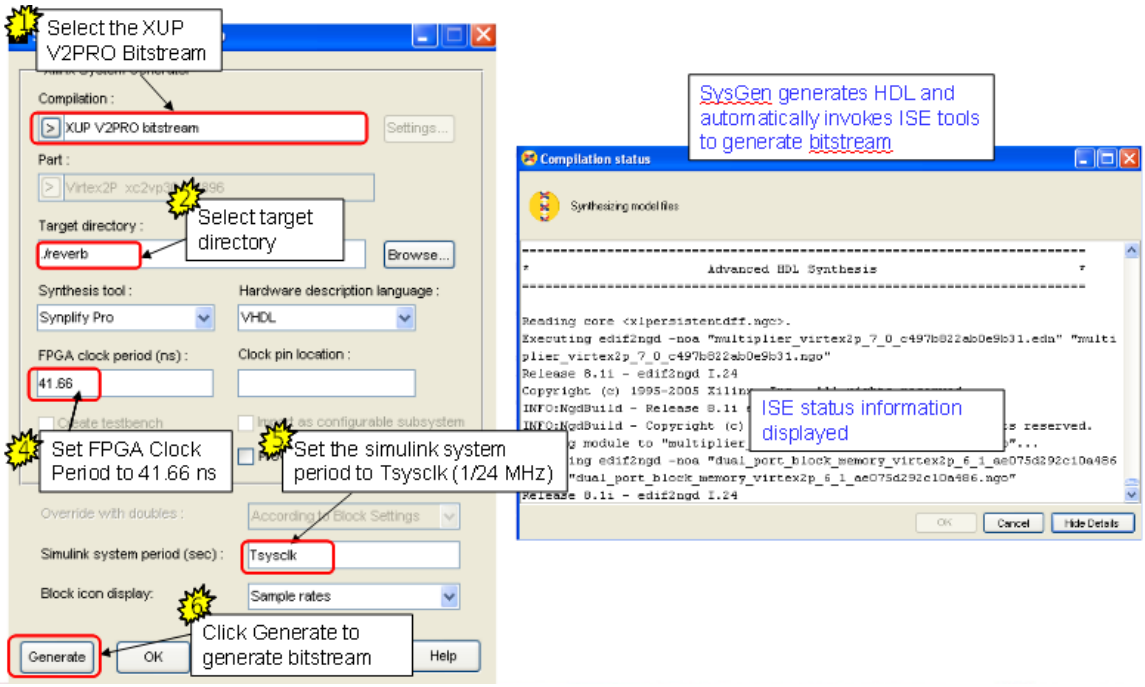


Εικόνα 6.3: Σχηματικό διάγραμμα για το πώς λειτουργεί ο AC97' controller. Παρατηρούμε ότι το αρχείο .ngc φροντίζει για το πώς θα επικοινωνήσει η σχεδίαση μας με τον AC97 controller.

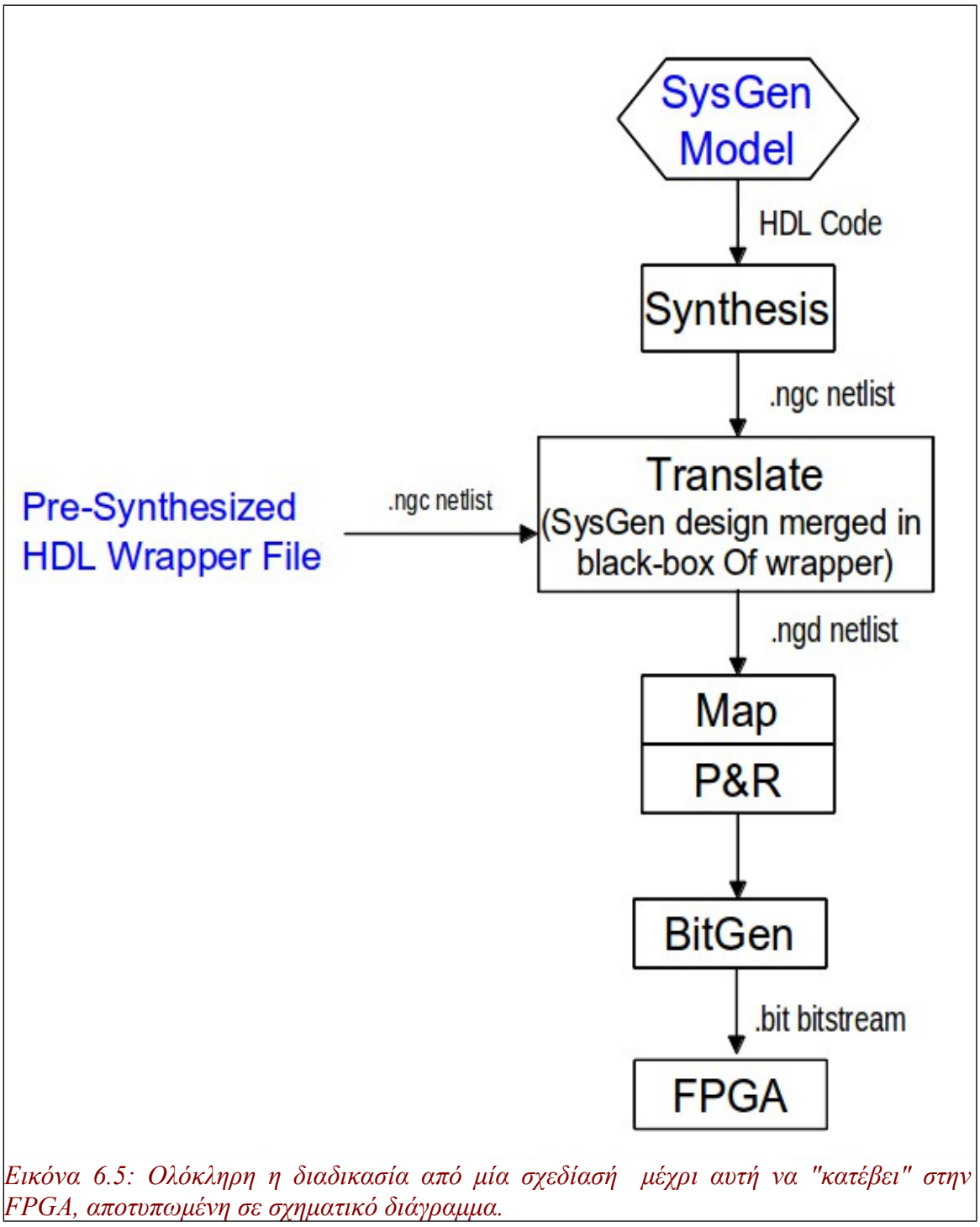
6.2. Η εφαρμογή της σχεδίασής μας στη Virtex II Pro

Το Board Support Pagkage, μας δίνει επίσης τη δυνατότητα όταν κάνουμε Generate τον κώδικα .vhd και τη netlist της σχεδίασής μας, να "στοχεύσουμε" κατευθείαν πάνω στη Virtex II pro και να μπορέσουμε να συμπεριλάβουμε τη σχεδίασή μας αυτόματα ως component της top level σχεδίασης του wrapper. Στην ουσία με το που επιλέξουμε Generate, και έχουμε σαν επιλογή το compilation να γίνει με χρήση του συγκεκριμένου BSP, το System Generator αυτομάτως ανοίγει το ISE για να παράξει στην αρχή τον κώδικα VHDL, έπειτα κάνοντας synthesize παράγει την .ngc netlist, όμως, στη φάση της υλοποίησης, τη "συγχωνεύει" με τη netlist του wrapper.

Έτσι κατά την διαδικασία της δημιουργίας του bitstream, το .bit αρχείο περιέχει τη σχεδίασή μας, μαζί με το interface που χρειάζεται για τη χρήση και τον έλεγχο του AC97 codec. Στη συνέχεια με χρήση του προγράμματος Impact μπορούμε πια να κατεβάσουμε το .bit αρχείο στην FPGA μας. Εμείς με τη βοήθεια του οδηγού αυτού κατασκευάσαμε ένα batch αρχείο το οποίο έχουμε επεξεργαστεί, και κάθε φορά που γίνεται διπλό κλικ σε αυτό το αρχείο, "κατεβάζει" αυτόματα (καλώντας φυσικά το Impact) αυτή τη σχεδίαση στην Virtex-II Pro.



Εικόνα 6.4: Πώς δημιουργείται ο κώδικας VHDL και το bitstream για μία σχεδίασή μας.



Εικόνα 6.5: Ολόκληρη η διαδικασία από μία σχεδίασή μέχρι αυτή να "κατέβει" στην FPGA, αποτυπωμένη σε σχηματικό διάγραμμα.

ΚΕΦΑΛΑΙΟ 7

Συμπεράσματα και προτάσεις για μελλοντική δουλειά

Έπειτα από το σύνολο της δουλειάς που έγινε, μπορούμε να εξάγουμε πολλά χρήσιμα συμπεράσματα.

Για κάποια εφέ, και κυρίως τα delay-based, μπορούμε να πούμε πως η υλοποίησή τους είναι καθ' όλα ικανοποιητική έως τέλεια, τόσο που θα μπορούσαν να σταθούν αξιοπρεπώς απέναντι σε άλλα κατασκευασμένα του εμπορίου.

Επίσης, μπορούμε να πούμε για κάποια εφέ ότι λειτούργησαν στο βαθμό του επιθυμητού, αλλά αποδέχονται περιθώρια βελτίωσης. Τέτοια είναι κυρίως όσα εφέ χρησιμοποίησαν FIR φίλτρα, και το pitch shifting.

Το equalizer το οποίο κατασκευάσαμε βρίσκεται προφανώς στην απλούστερή του μορφή, χρησιμοποιώντας μόνο 3 FIR φίλτρα. Ένα πολύ πιο “πετυχημένο” equalizer θα μπορούσε να αποτελείται από πολύ περισσότερα, για ακόμα περισσότερη γκάμα ζωνών συχνοτήτων και αντίστοιχη ρύθμιση της στάθμης τους.

Το εφέ wah έχει σαφώς το ελάττωμα της μη συνεχούς ολίσθησης του ζωνοπερατού φίλτρου στις συχνότητες. Σε μελλοντική δουλειά, θα μπορούσε κάποιος βασισμένος στον αλγόριθμο MATLAB να κατασκευάζει συνεχώς ένα νέο φίλτρο για κάθε δείγμα του σήματος εισόδου, ή για κάθε ομάδα δειγμάτων, έτσι ώστε να εξαλειφθούν οι ασυνέχειες και να γίνεται η ολίσθηση χωρίς απότομες μεταβάσεις από το ένα φίλτρο στο άλλο. Κάτι τέτοιο βέβαια θα είχε τεράστιες απαιτήσεις σε κόστος, οπότε σκοπός της μελλοντικής αυτής δουλειάς θα είναι και η ελαχιστοποίησή του.

Για το εφέ pitch shifting, είναι πολλά τα συμπεράσματα που μπορούμε να βγάλουμε και ακόμα περισσότερες είναι οι προτάσεις που έχουμε να κάνουμε για μελλοντική δουλειά. Πρώτα απ' όλα, πρέπει να σημειώσουμε ότι ο αλγόριθμος pitch shifting που ακολουθήσαμε βρίσκεται στην πιο απλή του μορφή, και με δουλειά αποκλειστικά στο πεδίο του χρόνου. Αυτό του δημιουργεί και διάφορα ελαττώματα, τα οποία φυσικά δεν υπάρχουν σε περίπτωση υλοποίησής του στο πεδίο της συχνότητας. Παρ' όλα αυτά, η διαδικασία του χωρίσματος του σήματος σε frames, η επανατοποθέτησή τους και το resampling, μπορούν να αποτελέσουν έναν πολύ χρήσιμο “προθάλαμο” για όποιον θα ήθελε να το συνεχίσει με υλοποίηση στη συχνότητα, καθώς η διαδικασία είναι παρόμοια.

Θα μπορούσε λοιπόν να αποτελέσει μία πολύ ενδιαφέρουσα διπλωματική εργασία η εφαρμογή του εφέ αυτού ακολουθώντας όλη τη διαδικασία της ημιτονοειδούς ανάλυσης και σύνθεσης, με την τεχνική εξισορρόπησης φάσης μεταξύ των frames με χρήση FFT, καθώς και χρήση διαφόρων άλλων τεχνικών που υπάρχουν, όπως πχ. όταν μιλάμε για φωνή, τη μοντελοποίηση του φάρυγγα ενός ανθρώπου, που μπορεί ενώ αλλάζει το pitch της φωνής να μην αλλάζει η “χροιά”.

Γενικότερα, η εφαρμογή της ημιτονοειδούς ανάλυσης και σύνθεσης ενός ηχητικού σήματος με χρήση FFT είναι πάρα πολύ χρήσιμη σε πάρα πολλές εφαρμογές ψηφιακής επεξεργασίας σήματος. Η real time υλοποίησή της σε hardware είναι ένα πολύ ενδιαφέρον πεδίο που θα μπορούσε να προσδώσει πολύ εντυπωσιακά αποτελέσματα.

Είναι επίσης γεγονός ότι πολλά εφέ μπορούν να δώσουν καλύτερα αποτελέσματα όταν η υλοποίησή τους γίνεται με χρήση FFT, στο πεδίο της συχνότητας.

Ένα ακόμα θέμα που μας απασχόλησε στην παρούσα διπλωματική εργασία είναι το θέμα της χωρητικότητας, και του κόστους των υλοποιήσεών μας. Είναι γεγονός πως όλα τα εφέ μαζί δεν μπορούν να εφαρμοστούν στην FPGA που χρησιμοποιούμε, λόγω του ότι δεν έχει τους απαιτούμενους πόρους για να τα υποστηρίξει. Παρατηρούμε όμως επίσης, πως εαν εξαιρέσουμε το εφέ Wah από αυτά, τότε οι πόροι της Virtex-II Pro αρκούν για να τα χωρέσουν.

Features	XC2VP20	XC2VP30
Slices	9280	13969
Array Size	56 x 46	80 x 46
Distributed RAM	290 Kb	428 Kb
Multiplier Blocks	88	136
Block RAMs	1584 Kb	2448 Kb
DCMs	8	8
PowerPC RISC Cores	2	2
Multi-Gigabit Transceivers	8	8

Εικόνα 7.1: Οι πόροι της Virtex II Pro που χρησιμοποιήσαμε.

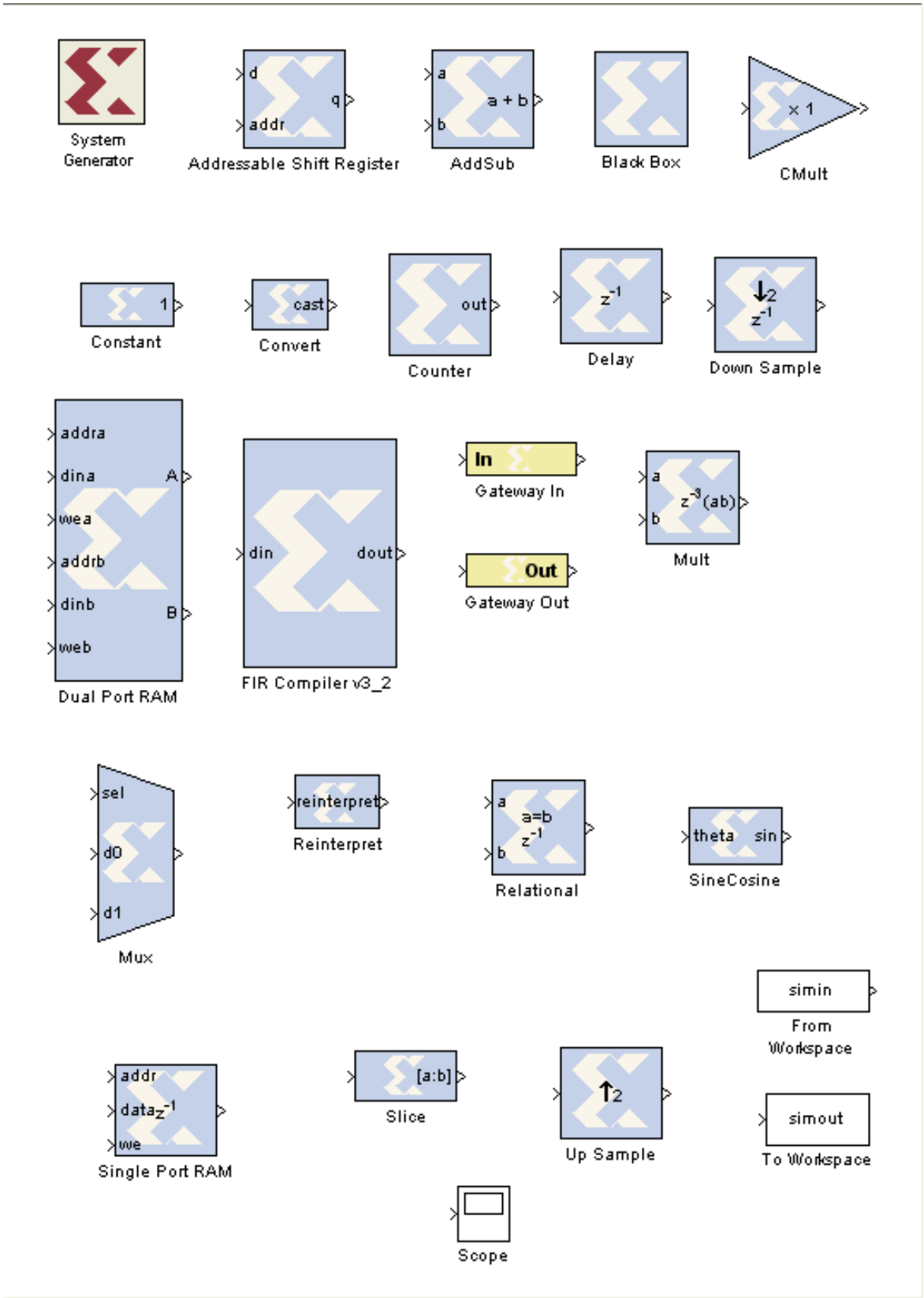
Πρόταση λοιπόν για μελλοντική δουλειά θα μπορούσε να αποτελέσει η βελτιστοποίηση των αλγορίθμων, της σχεδίασής τους σε hardware, και της υλοποίησής τους έτσι ώστε να ελαχιστοποιηθεί το κόστος τους σε πόρους. Πιθανότατα να χρειαστεί για μεγαλύτερη διευκόλυνση, συμβατότητα με πιο σύγχρονα εργαλεία της Xilinx, μεγαλύτερες ταχύτητες και περισσότερους πόρους η χρήση FPGAs όπως η Virtex-4 και η Virtex-5.

Χρήσιμο θα ήταν επίσης όσο είναι "κατεβασμένο" κάποιο εφέ στην FPGA να μπορούσε να είναι και real-time παραμετροποιήσιμο. Δηλαδή, με κάποιο τρόπο (όπως πχ από το πληκτρολόγιο, ή από ένα knob) να μπορούσα real-time να δώσω συγκεκριμένη τιμή πχ. για το σε ποιο σημείο θα γίνει το clipping του fuzz, πόσα δευτερόλεπτα καθυστέρηση θέλω στο echo, και ένα σωρό από άλλες παραμέτρους που χρησιμοποιούμε στις υλοποιήσεις όλων των εφέ. Αυτό αποτελεί επίσης και πρόταση για βελτίωση της υπάρχουσας ή μελλοντική δουλειά.

Υπάρχουν φυσικά και άλλα εφέ τα οποία δεν υλοποιήσαμε στην παρούσα διπλωματική εργασία, που όμως θα είχαν ενδιαφέρον και θα είχαν εντυπωσιακά αποτελέσματα. Μερικά από αυτά είναι το tremolo, το chorus, το time-stretching, ένας compressor, vocal remover και διάφοροι άλλοι αλγόριθμοι αφαίρεσης θορύβου.

ΠΑΡΑΡΤΗΜΑ Α

Τα μπλοκ της Xilinx που χρησιμοποίησα και η λειτουργία τους



From workspace και To workspace.
Ο τρόπος με τον οποίο παίρνω ένα διάνυσμα από τη MATLAB σαν είσοδο, και αντίστοιχα το ξαναστέλνω στο workspace της MATLAB ως έξοδο. Βασικότατα εργαλεία κατά την υλοποίηση των εφέ, καθώς σε συνδυασμό με το simulation μου επέτρεπαν την δοκιμή μιας σχεδίασης επί τόπου, χωρίς να χρειαστεί να το "κατεβάσω" στην FPGA για να δω αν δουλεύει. Βρίσκονται στη βιβλιοθήκη με τα μπλοκ του Simulink, και όχι της Xilinx.

Scope.
Λειτουργεί όπως ακριβώς ένας ψηφιακός παλμογράφος, με τη διαφορά ότι φαίνεται στην οθόνη του υπολογιστή. Σημαντικό εργαλείο, όταν θέλουμε να έχουμε οπτική επαφή με ένα σήμα ήχου (όπως αν θέλουμε να δούμε σε ποια σημεία μηδενίζεται) και όχι να προσπαθούμε να αναγνώσουμε πίνακες της MATLAB εκατοντάδων χιλιάδων στοιχείων.

Addressable Shift Register.
Η λειτουργία αυτού του μπλοκ περιγράφεται αναλυτικά στην υλοποίηση του εφέ flanger παραπάνω.

AddSub.
Επιτελεί την αριθμητική πράξη της πρόσθεσης ή της αφαίρεσης μεταξύ δύο εισόδων. Μπορεί να δώσει σαν έξοδο και τα Carry in ή Carry out της πράξης. Μπορώ επίσης να επιλέξω και τον τύπο της εξόδου (signed, unsigned, σε ποιο σημείο είναι το binary point, διαχείρηση του overflow κλπ.).

Black Box.
Το μπλοκ εκείνο που μου επιτρέπει να εισάγω δικό μου "χειροποίητο" VHDL κώδικα, και να τον συνδυάσω με άλλα μπλοκ του System Generator. Το κουτί φυσικά πρέπει να "βλέπει" ένα αρχείο .vhd, το οποίο φυσικά θα του προσδώσει αντίστοιχες εισόδους/εξόδους και τον τύπο τους. Με το που επιλέγω το αρχείο αυτό, γίνεται ένα αντίστοιχο check syntax, όμοιο με αυτό του ISE. Σε περίπτωση που υπάρχει λάθος αυτό αναφέρεται, χωρίς όμως περαιτέρω λεπτομέρειες.

Constant.
Μία σταθερά με τιμή και τύπο που θα εισάγω εγώ. Μπορώ να την επιλέξω signed ή unsigned, από πόσα bits θα αποτελείται και σε ποιο σημείο θα βρίσκεται το binary point.

Convert.
Αλλάζει στην έξοδο τον τύπο της τιμής που έχει δεχθεί σαν είσοδο. Αν πχ. έχω δώσει σαν είσοδο έναν αριθμό των 16 bits με το binary point στο 14, μπορώ να το μετατρέψω σε αριθμό 8 bits με binary point στο 7. Μπορώ επίσης να επιλέξω και τη μέθοδο "στρογγυλέματος" rounding mode.

Counter.
Είναι ένας μετρητής. Μπορώ να επιλέξω το βήμα, την αρχική του τιμή, τα bits που θα τον αποτελούν, μέχρι ποια τιμή μπορεί να φτάσει και το χρονισμό του. Μπορώ επίσης αν θέλω να έχω και άλλες εισόδους όπως reset, load, enable. Η είσοδος reset τον επαναφέρει στην αρχική του τιμή, η είσοδος load του δίνει νέα τιμή, και όταν η είσοδος enable είναι 1, τότε μετράει, αλλιώς μένει σταθερός.

Delay.
Υλοποιεί την καθυστέρηση της εισόδου για L κύκλους ρολογιού. Ό,τιδήποτε δεδομένο δοθεί στην είσοδο, θα βγει στην έξοδο μετά το πέρας L κύκλων ρολογιού.

FIR Compiler.
Κατασκευάζει ένα FIR φίλτρο με παραμέτρους που του έχει δώσει ο χρήστης. Αναλυτική περιγραφή έχουμε κάνει σε προηγούμενο κεφάλαιο. Μπορούμε να επιλέξουμε τον τύπο του φίλτρου, τους συντελεστές του και τον τύπο τους, και το αν θα είναι reloadable.

Gateway In και Gateway Out.
Συμβολίζουν τα "όρια" της FPGA μου. Ό,τι βρίσκεται ενδιάμεσα από αυτά αποτελεί κομμάτι που θα προγραμματιστεί στην FPGA, ό,τι βρίσκεται έξω από αυτά υπάρχει μόνο για να δίνει είσοδο/έξοδο από και προς τον "έξω κόσμο".

Mult και Cmult.
Το μπλοκ Mult πραγματοποιεί πολλαπλασιασμό 2 εισόδων και στέλνει το αποτέλεσμα στην έξοδο. Το μπλοκ Cmult κάνει το ίδιο, αλλά η μία από τις δύο εισόδους είναι μια σταθερά. Στην ουσία το μπλοκ Cmult πραγματοποιεί ενίσχυση, εφόσον μιλάμε για ψηφιακή επεξεργασία σήματος. Μπορώ κι εδώ να επιλέξω τύπο εισόδου.

Multiplexer.
Ο γνωστός σε όλους μας πολυπλέκτης. Δέχεται έναν αριθμό από εισόδους και στέλνει την κατάλληλη στην έξοδο. Για το ποια θα είναι η έξοδος τελικά, μας το υποδεικνύει το σήμα sel.

Reinterpret.
Αλλάζει το binary point της εισόδου, χωρίς να αλλάξει την αναπαράστασή της σε bit. Έτσι μπορώ να μετατρέψω πχ. τον αριθμό 1001010.010 σε 10.01010010 επιλέγοντας απλά το binary point να μεταφερθεί στη θέση 8.

Relational.
Πραγματοποιεί μια σύγκριση ανάμεσα σε δύο εισόδους, και ανάλογα με το αν αυτή επαληθεύεται ή όχι, στέλνει στην έξοδο μια boolean τιμή 0 ή 1. Οι συγκρίσεις μπορούν να είναι a=b, a!=b, a<b, a>b, a<=b και a>=b.

Single και Dual Port RAM.
Οι δύο πιο γνωστοί τύποι μνημών. Η single port RAM έχει τρεις εισόδους. Η μία (addr) είναι για τη διεύθυνση που θα σαρώνεται εκείνη τη στιγμή, η δεύτερη (data) για τα δεδομένα που είναι να γραφτούν και η τρίτη είναι το write enable, όπου αν είναι 1 τότε έχω εγγραφή. Η dual port RAM έχει τις ίδιες ακριβώς εισόδους αλλά για 2 διαφορετικές διευθύνσεις, άρα 6 στο σύνολο. Στην έξοδο βγάζουν αντίστοιχα το στοιχείο της addr που διαβάζεται. Και στους δύο τύπους μνημών προσδίδω μέγεθος και αρχικοποίηση. Η αρχικοποίηση μπορεί να γίνει και με ένα διάνυσμα από το workspace της MATLAB. Έχω επίσης και την επιλογή να γίνεται πρώτα η ανάγνωση και μετά η εγγραφή, ή το αντίστροφο.

Slice.
Κόβει ένα μέρος από τα bits της εισόδου. Μπορώ πχ. αν η είσοδος μου είναι η 10000111 στην έξοδο να έχω 111 ή 100, ανάλογα από πού θα του υποδείξω να ξεκινήσει το κόψιμο, από το MSB ή το LSB. Το binary point της εξόδου είναι πάντα στο σημείο 0. Η έξοδος του μπλοκ αυτού μπορεί να είναι και boolean, αν το slice γίνει μόνο για ένα bit.

Upsample και Downsample.
Το μπλοκ Upsample εισάγει παραπάνω δείγματα στο σήμα μου. Μπορώ να επιλέξω αν τα δείγματα θα είναι μηδενικά ή αντίγραφα των ήδη υπάρχοντων δειγμάτων. Πρέπει και το αντίστοιχο ρολόι της σχεδίασής μου να το υποστηρίζει αυτό, αλλιώς υπάρχει σφάλμα κατά την υλοποίηση. Αν δηλαδή του υποδείξω να κάνει upsample κατά 2, που σημαίνει ότι εισάγει ένα ακόμα sample μετά από το καθένα, πρέπει να μειώσω και το ρολόι της συνολικής μου σχεδίασης στο μισό. Αυτό θέλει λεπτή προσοχή κατά το χειρισμό του, διότι μπορεί και άλλα

μπλοκ της σχεδίασής μου να είναι χρονισμένα, και άρα να πρέπει να αλλάξω το ρολόι και σε αυτά. Το μπλοκ downsample αφαιρεί δείγματα από την είσοδο, μειώνοντας έτσι τον ρυθμό των δεδομένων που έχω. Φυσικά αυτό σημαίνει και απώλεια πληροφορίας.

ΠΑΡΑΡΤΗΜΑ Β

Οι κώδικες MATLAB που χρησιμοποιήθηκαν

Κώδικας MATLAB για delay, echo, reverb

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Echo-delay-reverb effect:
% for a small amount of samples N (ap. 2000-3000), the effect operates
% as reverb. For a big amount of samples N, but with no weakening during
% the repeat, we have a single delay effect. If we add a sound
weakening, it
% turns out to an echo effect.
%
% created by Panagiotis Petropoulos
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

[input, fs, bits] = wavread('abc.wav');
x = input(:,1);

N=15000;  % delay of N/44100 seconds
ep=4; %number of repeats

c = 4; % sound weakening after repeat

y=[x;zeros(N*ep,1)];  %set up output size

% I create a table of repeats.

A(1,:) = [x;zeros(N*ep,1)]; % initialization

if c==1
    for i=2:ep+1
    A(i,:) = [zeros((i-1)*N,1);x;zeros((ep+1-i)*N,1)];
    end

    for i=2:ep+1
        for j=1:length(y)
        y(j) = y(j) + A(i,j);
        end
    end
end

if c~=1
    for i=2:ep+1
    A(i,:) = [zeros((i-1)*N,1);x./((i-1)*c);zeros((ep+1-i)*N,1)];
    end

    % the sum of table A, will be my output signal.
    for i=2:ep+1
        for j=1:length(y)
        y(j) = y(j) + A(i,j);
        end
    end
end
```

```
%sound(x,fs) % original
sound(y,fs) % new signal
```

Ο κώδικας MATLAB για vibrato-flanger

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% M Code that creates a single delay  with the delay time ocilating from
%
% either 0-3 ms or 0-15 ms at 0.1 - 5 Hz
%                                                                       %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%

% Current sample is 11kHz so 0-3 ms is 0 - 33 samples

% Reads a WAVE file specified by the string 'name.wav', returning the
sampled data in 'in_wave'.
% Amplitude values are in the range [-1,+1].
% Fs is the sample rate in Hertz. NBITS is the number of bits per sample
used to encode the data
[in_wave,Fs,NBITS] = wavread('arm.wav');

% Initialize parameters to vary the effect
% 3ms max delayin seconds
max_time_delay=0.003;

% Rate in Hz, from 5 to 10 for vibrato, 0.5 to 5 for flanger
rate=0.4;

index=1:length(in_wave);

% Sin reference to create oscillating delay
sin_ref = (sin(2*pi*index*(rate/Fs)))';    % sin(2pi*fa/fs);

% Convert delay in ms to max delay in samples
max_samp_delay=round(max_time_delay*Fs);

% Create a vector of zeros of length 'in_wave' for the output wave
flanger = zeros(length(in_wave),1);
vibrato = zeros(length(in_wave),1);

% To avoid referencing of negative samples
flanger(1:max_samp_delay)=in_wave(1:max_samp_delay);
vibrato(1:max_samp_delay)=in_wave(1:max_samp_delay);

% Suggested coefficient from page 71 DAFX
amp=0.7;

% For each sample
for i = (max_samp_delay+1):length(in_wave),
    cur_sin=abs(sin_ref(i));    %abs of current sin
    cur_delay=ceil(cur_sin*max_samp_delay);  % generate delay from 1-
max_samp_delay and ensure whole number

    flanger(i) = (amp*in_wave(i)) + (amp*in_wave(i-cur_delay));   %
delayed sample + the original for flanging
    vibrato(i) = in_wave(i-cur_delay); % only the delayed for vibrato.
end
```

```
sound(in_wave, Fs);
sound(flanger, Fs);
sound(vibrato, Fs);
```

Κώδικας MATLAB για το εφέ Wah

```
% wah_wah.m   state variable band pass
% written by Ronan O'Malley
% October 2nd 2005
%
% BP filter with narrow pass band, Fc oscillates up and down the
spectrum
% Difference equation taken from DAFX chapter 2
%
% Changing this from a BP to a BS (notch instead of a bandpass) converts
this effect to a phaser
%
% yl(n) = F1*yb(n) + yl(n-1)
% yb(n) = F1*yh(n) + yb(n-1)
% yh(n) = x(n) - yl(n-1) - Q1*yb(n-1)
%
% vary Fc from 500 to 5000 Hz
% 44100 samples per sec

clear all;
close all;
clc;

infile = 'x.wav';

% read in wav sample
[ x, Fs, N ] = wavread(infile);


%%%%%%% EFFECT COEFFICIENTS %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% damping factor
% lower the damping factor the smaller the pass band
damp = 0.05;

% min and max centre cutoff frequency of variable bandpass filter
minf=500;
maxf=3000;

% wah frequency, how many Hz per second are cycled through
Fw = 19000;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%

% change in centre frequency per sample (Hz)
% delta=0.1;
delta = Fw/Fs;
%0.1 => at 44100 samples per second should mean  4.41kHz Fc shift per
sec
```

```
% create triangle wave of centre frequency values
Fc=minf:delta:maxf;
while(length(Fc) < length(x) )
    Fc= [ Fc (maxf:-delta:minf) ];
    Fc= [ Fc (minf:delta:maxf) ];
end
plot(Fc)
% trim tri wave to size of input
Fc = Fc(1:length(x));


% difference equation coefficients
F1 = 2*sin((pi*Fc(1))/Fs);  % must be recalculated each time Fc changes
Q1 = 2*damp;                % this dictates size of the pass bands


yh=zeros(size(x));          % create emptly out vectors
yb=zeros(size(x));
yl=zeros(size(x));

% first sample, to avoid referencing of negative signals
yh(1) = x(1);
yb(1) = F1*yh(1);
yl(1) = F1*yb(1);

% apply difference equation to the sample
for n=2:length(x),

    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);
    yb(n) = F1*yh(n) + yb(n-1);
    yl(n) = F1*yb(n) + yl(n-1);

    F1 = 2*sin((pi*Fc(n))/Fs);
end

%normalize

maxyb = max(abs(yb));
yb = yb/maxyb;

% write output wav files
% wavwrite(yb, Fs, N, 'out_wah.wav');
% sound(x, Fs);

sound(yb, Fs);

figure()
hold on
plot(x,'r');
plot(yb,'b');
title('Wah-wah and original Signal');
```

Κώδικας MATLAB για το εφέ pitch shifting

```
%% Define input signal and sampling rate

[input, fs, bits] = wavread('abc.wav'); % input WAV

if (length(input(:,1))>1)
```

```
    input = input';
end

%% Define global constants

alpha = 16/12; % pitch-shift factor
N = 512; % frame length
overlap = .75; % overlap fraction
window = hanning(N)'; % input window

%% Calculate working variables

input_length = length(input); % length of input signal
frame_count = floor((input_length-2*N)/(N*(1-overlap)));

% number of frames in input %%

Ra = floor(N*(1-overlap)); % analysis time hop
Rs = floor(alpha*Ra); % synthesis time hop
output = zeros(1, input_length*alpha); % output signal initialization

%% Process input frames

Xu_current = window.*input(1:N); % analyze initial frame

for u=1:frame_count
    Xu_current = window.*input(u*Ra:u*Ra+N-1);% analyze current frame
    output(u*Rs:u*Rs+N-1) = output(u*Rs:u*Rs+N-1) + Xu_current;%  add
current frame to output
end

[t,d]=rat(alpha); % determine integer shift ratio
shifted = resample(output,d,t); % resample for pitch shift

sound(input,fs);
sound(shifted,fs);
```

ΠΑΡΑΡΤΗΜΑ Γ
ΒΙΒΛΙΟΓΡΑΦΙΑ

Www.wikipedia.org (προφανώς)

Για ψηφιακή επεξεργασία σήματος ήχου:

Για την μελέτη και την κατανόηση των ηχητικών εφέ χρησιμοποιήθηκε κατά κόρον το σύγγραμα: DAFX, *Digital Audio Effects,* edited by Udo Zolzer.

Sound processing in MATLAB.
http://homepages.udayton.edu/~hardierc/ECE203/sound.htm

Physical audio signal processing.
https://ccrma.stanford.edu/~jos/pasp/

DSP Guru: Resampling.
http://www.dspguru.com/dsp/faqs/multirate/resampling

Για τα ηχητικά εφέ.

Modeling guitar distortion.
http://cnx.org/content/m14221/latest/

Delay-based effects.
http://www.ee.columbia.edu/~ronw/adst-spring2010/lectures/lecture3.pdf

Sound effects.
http://sound.eti.pg.gda.pl/student/eim/synteza/adamx/eindex.html#poczatek

Filtering and EQ.
http://www2.gibson.com/News-Lifestyle/Features/en-us/effects-explained-filtering-an.aspx

Technology of wah pedals.
http://www.geofex.com/article_folders/wahpedl/wahped.htm

Pitch shifting algorithm explanation.
http://www.katjaas.nl/pitchshift/pitchshift.html

Real time pitch shifting of musical signals.
http://dafx.labri.fr/main/papers/p007.pdf

DSP Dimension – Time stretching and pitch shifting of audio signals.
http://www.dspdimension.com/admin/time-pitch-overview/

Real time speech pitch shifting.
http://www56.homepage.villanova.edu/scott.sawyer/fpga/II_time_domain.htm

Guitar Pitch Shifter.
http://www.guitarpitchshifter.com/

Hardware implementation.

Xilinx forums.
http://forums.xilinx.com/

Xilinx System Generator for DSP 10.1 user guide.
http://www.xilinx.com/support/sw_manuals/sysgen_ref.pdf

XUP Virtex II Pro Documentation.
http://www.xilinx.com/univ/xupv2p.html

Implementing a digital audio equalizer (XUPV2Pro).
http://www.ent.mrt.ac.lk/rds/index.php/curriculum-support/dsd-projects/30-batch-07-dsd-projects/29-implementing-a-digital-audio-equalizer-xup-virtex-ii-pro-fpga

XUP Virtex II Pro performing handware in-the-loop.
http://www.decom.fee.unicamp.br/~cardoso/ie344b/roteiro_xupv2p_jtag_cosim.pdf

Digital audio effects and implementation on FPGAs, Jaime Andrés Laino Guerra.
http://kosmos.upb.edu.co/web/uploads/articulos/(A)_ANALISIS_DE_EFECTOS_DIGITALES_DE_AUDIO_BASADOS_EN_RETARDOS_E_IMPLEMENTACION_SOBRE_FPGA_dxPW9D.pdf