ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Υλοποίηση Ολοκληρωμένου Συστήματος Παθητικής Καταγραφής Ήχων στο Υδάτινο Περιβάλλον με χρήση Field Programmable Gate Array (FPGA)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θεόδωρος Μ. Δακλαδάς

Επιβλέπων : Ελευθέριος Καγιάφας

Καθηγητής Ε.Μ.Π.

Αθήνα, Απρίλιος 2013

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδίαση και Υλοποίηση Ολοκληρωμένου Συστήματος Παθητικής Καταγραφής Ήχων στο Υδάτινο Περιβάλλον με χρήση Field Programmable Gate Array (FPGA)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Θεόδωρος Μ. Δακλαδάς

Επιβλέπων : Ελευθέριος Καγιάφας

Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 25 Απριλίου 2013.

.....	.....	.....
Ελευθέριος Καγιάφας	Γιώργος Οικονομάκος	Βασίλειος Λούμος

Αθήνα, Απρίλιος 2013

.....

Θεόδωρος Μ. Δακλαδάς

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Copyright © Θεόδωρος Μ. Δακλαδάς

Με επιφύλαξη παντός δικαιώματος. All rights reserved.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα.

Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και δεν πρέπει να ερμηνευθεί ότι αντιπροσωπεύουν τις επίσημες θέσεις του Εθνικού Μετσόβιου Πολυτεχνείου.

ΠΕΡΙΛΗΨΗ

Σκοπός της εργασίας αυτής είναι η σχεδίαση και υλοποίηση ενός ολοκληρωμένου Συστήματος Παθητικής Καταγραφής και Ανάλυσης Ήχων (Passive Aquatic Listener - PAL) για υδάτινο περιβάλλον. Η γενική λειτουργία του συστήματος αυτού, παρέχει την δυνατότητα καταγραφής και ανάλυσης κάθε ηχητική συχνότητα στο εύρος των 100Hz – 50kHz, ενώ ανάλογα με την μηχανολογική κατασκευή στην οποία θα τοποθετηθεί, επιτρέπεται η χρήση του στο υδάτινο περιβάλλον και σε βάθος από μερικά μέχρι αρκετές εκατοντάδες μέτρα. Τα συστήματα αυτά χρησιμοποιούνται για την αναγνώριση και μέτρηση κάθε ήχου που προέρχεται από δραστηριότητες φυσικές (άνεμος, βροχή), βιολογικές (φάλαινες, θαλάσσια ζώα) ή ανθρώπινες (μηχανές σκαφών).

Στο Κεφάλαιο 1 Περιγράφεται η γενική δομή και τα χαρακτηριστικά του συστήματος. Δίνονται οι γενικές προδιαγραφές της αναλογικής και της ψηφιακής βαθμίδας.

Στο Κεφάλαιο2 αναλύεται η θεωρία των βασικών δομικών μονάδων και οι παράμετροι λειτουργίας. Σύντομη θεωρητική περιγραφή της μετατροπής Analog-to-Digital και αντίστροφα, του θεωρήματος δειγματοληψίας, η μετατροπή Discrete Fourier Transform και ψηφιακή ανάλυση με χρήση αλγορίθμου Fast Fourier Transform. Περιγραφή Αναλογικών Φίλτρων και φίλτρων για μετατροπή δεδομένων. Σχεδίαση Ενεργών Αναλογικών Φίλτρων με χρήση Τελεστικών (Operational Amplifiers).

Στο Κεφάλαιο 3 περιγράφεται η σχεδίαση της αναλογικής βαθμίδας που περιλαμβάνει τον ενισχυτή, τα ενεργά φίλτρα, τον Analog to Digital μετατροπέα (ADC) και της μονάδας τροφοδοσίας. Υλοποίηση της σχεδίασης με επιλογή των κατάλληλων εξαρτημάτων του εμπορίου. Υπολογισμός Θορύβου τελεστικών ενισχυτών και ADC.

Στο Κεφάλαιο 4 παρουσιάζεται το Xilinx EDK Suite και περιγράφεται η σχεδίαση της Ψηφιακής Βαθμίδας που περιλαμβάνει τον μικροϋπολογιστή PowerPC 440, τον πυρήνα FFT, τους ελεγκτές μνήμης και εξωτερικής αποθήκευσης καθώς και το λογισμικό διαχείρισης του συστήματος.

Λέξεις Κλειδιά

Παθητική καταγραφή ήχων, PAL, αναλογικά φίλτρα, ενεργά φίλτρα, Sallen key, ψηφιακή επεξεργασία σήματος, τελεστικοί ενισχυτές, διακριτός μετασχηματισμός Fourier, DFT, FFT, ADC, Xilinx EDK, FPGA.

ABSTRACT

The purpose of this thesis is the design and implementation of a complete passive sound recorder and analyzer (Passive Aquatic Listener – PAL) for the aquatic environment. The main operation of this system is to record and analyze any frequencies in the range of 100Hz to 50kHz while, according to the mechanical construction that the system is incorporated, allows the deployment in the aquatic environment of several hundred meters depth. These systems are used for recognition and measurement of any kind of sound that comes from physical activities (wind, rain), biological (whales, sea animals) or human (marine engines).

In chapter 1 the general structure and the main features are described. The general specifications of the analog and digital stages are also given.

In chapter 2 the theory of the structural modules and operation parameters of the system are analyzed. There are short theoretical descriptions of the Analog – to - Digital and Digital – to – Analog conversions, the Discrete Fourier Transform and the digital signal processing using Fast Fourier Transform algorithm. Analog Filters and Filters used for data conversions are described. Designing procedures of Active Filters using Operational Amplifiers are shown.

In chapter 3 is described the design of the analog stage which includes the amplifier, the active filter, the Analog to Digital converter and the power supply unit. The implementation of the design with proper selection of commercial components is analyzed. The calculation procedure of noise in Operational Amplifiers and ADC is also shown.

In chapter 4 is presented the Xilinx EDK Suite that used for the design of the digital stage which involves the PowerPC processor, FFT ip core, memory and external storage device controllers and the system management software.

Keywords

Passive Aquatic Listener, PAL, analog filters, active filters, Sallen key, digital signal processing, Operational Amplifiers, Discrete Fourier Transform, Fast Fourier Transform , DFT, FFT, ADC, Xilinx EDK, FPGA.

ΕΥΧΑΡΙΣΤΙΕΣ

Ευχαριστώ τους καθηγητές μου κ. Ελευθέριο Καγιάφα και κ. Γιώργο Οικονομάκο για την πολύτιμη βοήθειά τους κατά την εκπόνηση της εργασίας αυτής. Επίσης ευχαριστώ τους φίλους της εταιρίας INTRACOM Χατζηαγάπη Στέφανο, Αθανασόπουλο Χρήστο και Οικονόμου Θανάση, για τη αμέριστη βοήθειά τους στην εξαιρετικά δύσκολη δουλειά της κατασκευής της πλακέτας του FPGA.

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

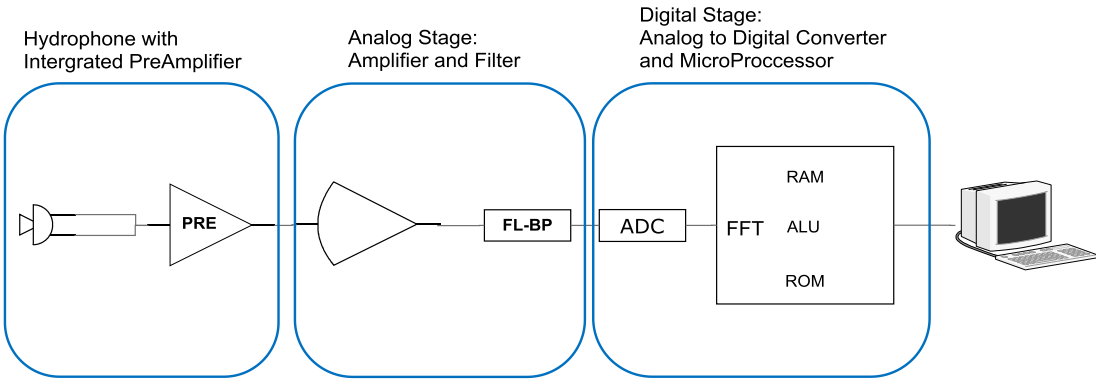
ΠΙΝΑΚΑΣ ΣΧΗΜΑΤΩΝ

ΠΙΝΑΚΑΣ ΠΙΝΑΚΩΝ

1. Γενική Περιγραφή Ολοκληρωμένου Υδάτινου Συστήματος Παθητικής Καταγραφής Ήχων (Passive Aquatic Listener - PAL)

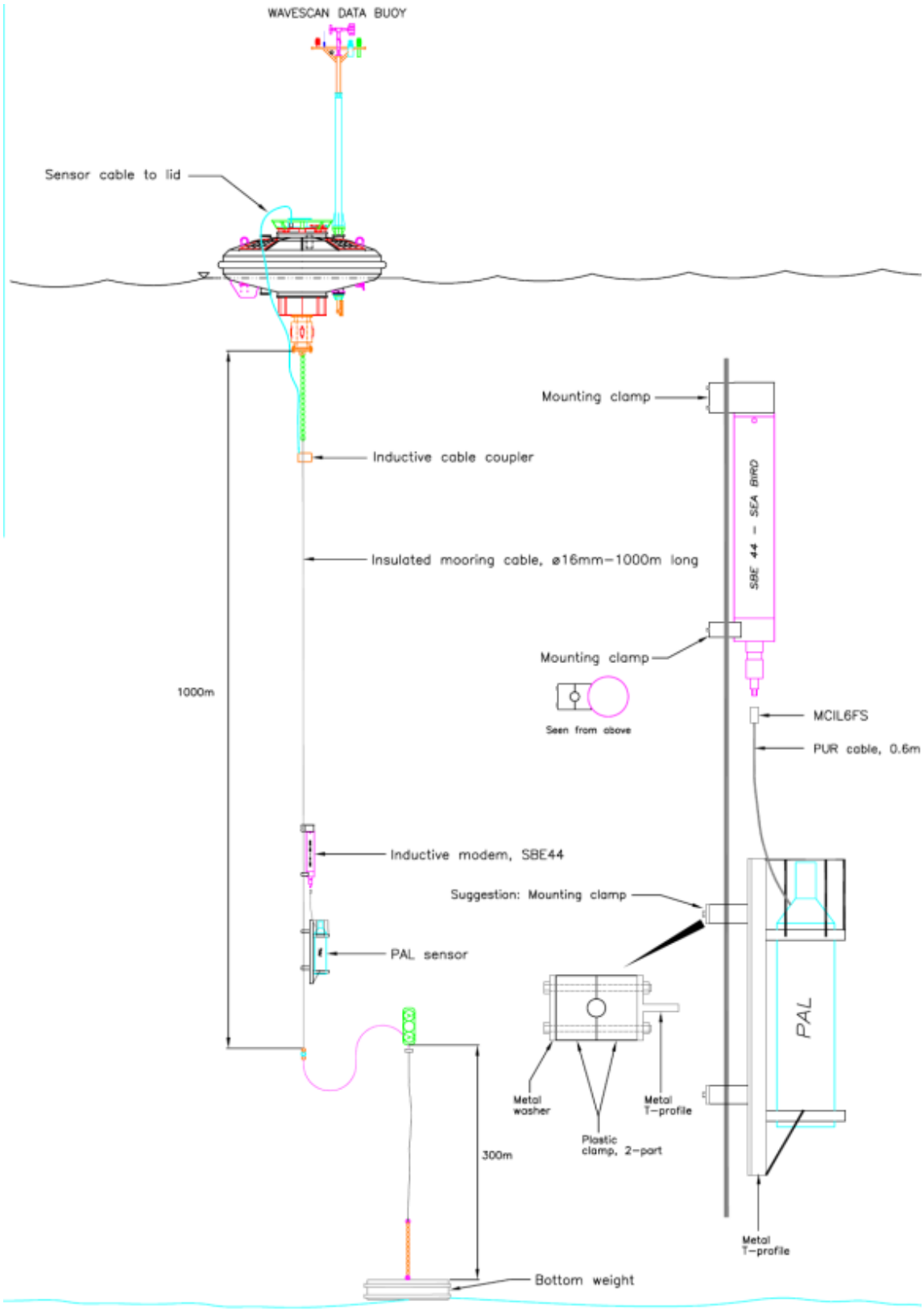
1.1 Εισαγωγή

Οι ήχοι στο θαλάσσιο περιβάλλον περιέχουν πληροφορίες για φυσικές, βιολογικές και ανθρώπινες δραστηριότητες και επομένως μπορούν να χρησιμοποιηθούν για να αξιολογηθεί το περιβάλλον αυτό.



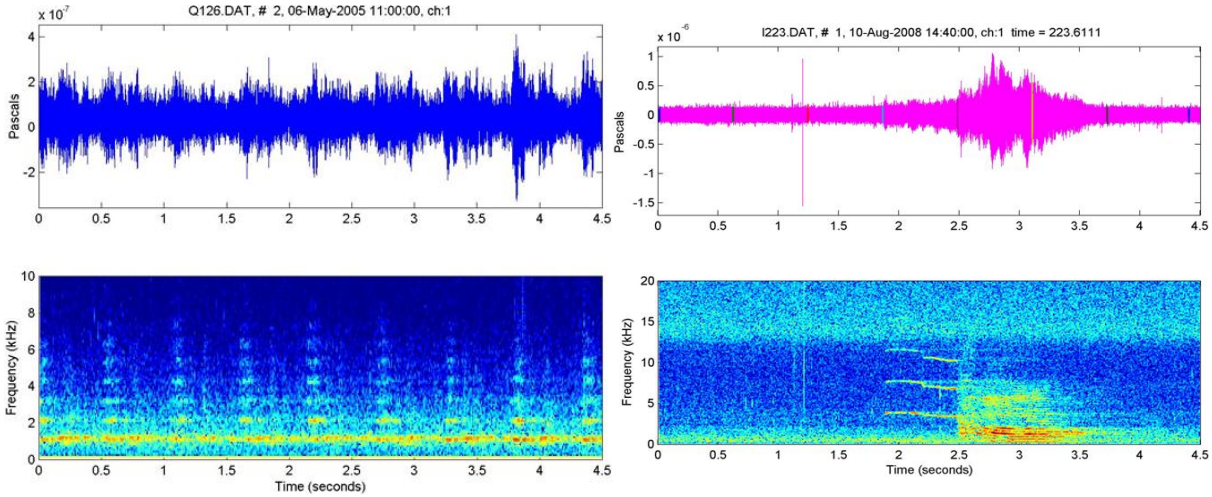
Σχήμα 1.1 Block Διάγραμμα Συστήματος

Το σύστημα PAL (Σχ.1.1) αποτελεί στην ουσία έναν αυτόνομο ηχητικό καταγραφέα, που είναι σχεδιασμένο να προσαρτάται σε αγκυροβόλια (Σχ.1.2) και να καταγράφει τους ήχους από διάφορες πηγές, οι οποίες βρίσκονται είτε στην επιφάνεια του νερού είτε εντός αυτού. Τα συνήθως ιδιαίτερα χαρακτηριστικά του ηχητικού φάσματος μιας πηγής μπορούν να οδηγήσουν σε ταυτοποίηση αυτής, επιτρέποντας την παρακολούθηση και μέτρηση των φασματικών επιπέδων διαφόρων συχνοτήτων, την μεταβολή των φασμάτων καθώς και την χρονική διάρκεια που η πηγή είναι ενεργή. Οι εφαρμογές του PAL περιλαμβάνουν: την ανίχνευση και μέτρηση του ηχητικού ίχνους ψαριών και θαλάσσιων θηλαστικών, την μέτρηση της έντασης της βροχής και ταχύτητας του ανέμου, την ανίχνευση σεισμικής δραστηριότητας, την ανίχνευση σκαφών, κ.λ.π. Ο αλγόριθμος λειτουργίας και ανάλυσης του συστήματος μπορεί να μεταβληθεί και να προσαρμοσθεί στις απαιτήσεις της εκάστοτε εφαρμογής.



Σχήμα 1.2 Ανάπτυξη Συστήματος PAL

Η γενική λειτουργία του PAL έχει ως εξής: το ηχητικό σήμα λαμβάνεται από το υδρόφωνο σε καθορισμένα χρονικά διαστήματα και διάρκεια (χρονοσειρά) και ενισχύεται σε πρώτο βαθμό από τον ενσωματωμένο προενισχυτή. Στη συνέχεια οδηγείται στην επόμενη αναλογική βαθμίδα όπου περνά από κατάλληλο antialiasing φίλτρο και γίνεται η τελική ενίσχυση. Ακολουθεί η δειγματοληψία και η ψηφιοποίηση του σήματος από Analog to Digital Converter. Στην ψηφιακή βαθμίδα υπάρχει ο μικρουπολογιστής με τη χρήση του οποίου, το ψηφιακό πλέον σήμα υπόκειται σε μετατροπή Fast Fourier και αφού αποθηκευθεί, εφαρμόζεται σε αυτό κατάλληλος αλγόριθμος που ανάλογα με την εφαρμογή, θα επιτρέψει την επιθυμητή παραπέρα ανάλυση των δεδομένων που έχουν καταγραφεί. Τα αποθηκευμένα δεδομένα, είτε σε μορφή raw data είτε σε φασματική, μπορούν να αναλυθούν με τη βοήθεια ενός PC.



Σχήμα 1.3 Παραδείγματα Φασμάτων PAL

1.2 Γενικές Απαιτήσεις Σχεδιασμού

Ένα ολοκληρωμένο σύστημα PAL πρέπει να είναι μικρό σε όγκο και να έχει τη δυνατότητα να λειτουργεί αυτόνομα για όσο το δυνατό μεγαλύτερο χρόνο, καθώς η διαδικασία ανάπτυξής του στο θαλάσσιο περιβάλλον είναι δαπανηρή και δύσκολη. Επίσης, τα αποτελέσματα της μέτρησής του θα πρέπει να είναι ακριβή και λεπτομερή, τόσο για ασθενείς όσο και για ισχυρούς ήχους. Σύμφωνα με τα παραπάνω, το σύστημα απαιτείται να παρουσιάζει:

- Χαμηλή κατανάλωση
- Αρκετά μεγάλη δυναμική περιοχή λειτουργίας
- Χαμηλό θόρυβο
- «Έξυπνο» αλγόριθμο επεξεργασίας δεδομένων
- Ακρίβεια μέτρησης
- Μεγάλη χωρητικότητα μνήμης
- Μικρό σχετικά μέγεθος

Σχήμα 1.4
Σύστημα PAL

Για το σχεδιασμό και την κατασκευή του υπόψη συστήματος, τα παραπάνω χαρακτηριστικά λαμβάνονται υπόψη, κάποια σε μεγαλύτερο και κάποια σε μικρότερο βαθμό, ανάλογα με τη σημασία τους.

Στη συνέχεια θα αναλυθεί η δομή του υπό κατασκευή συστήματος, θα περιγραφεί η λειτουργία όλων των μονάδων αυτού και θα προδιαγραφούν οι απαιτήσεις που πρέπει να πληρούνται από κάθε βαθμίδα.

1.3 Βιβλιογραφία

[1] Prof. E. N. Anagnostou, Dr. M. N. Anagnostou & Prof. Jeff Nystuen

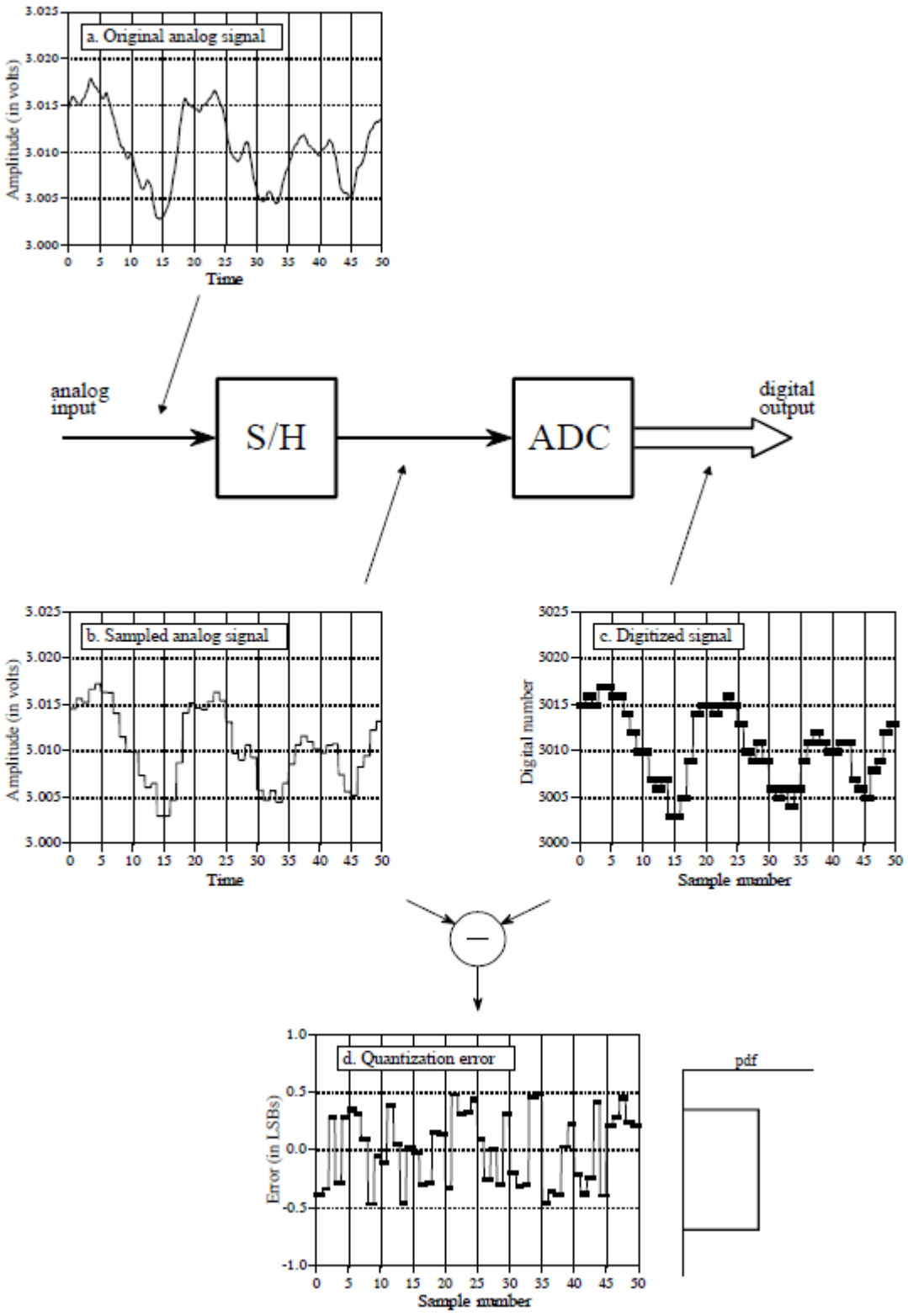
2. Θεωρητική Ανάλυση των Στοιχείων και Παραμέτρων Λειτουργίας

2.1 Μετατροπή Analog to Digital και αντίστροφα

Τα περισσότερα σήματα που συναντάμε στις επιστήμες και τη φύση είναι συνεχή: ένταση φωτός που αλλάζει με την απόσταση, τάση που μεταβάλλεται με το χρόνο κ.λ.π. Αναλογική σε Ψηφιακή μετατροπή (ADC) και Ψηφιακή σε Αναλογική μετατροπή (DAC) είναι οι διαδικασίες που επιτρέπουν στους ηλεκτρονικούς υπολογιστές να διαχειριστούν τέτοια σήματα. Η Ψηφιακή πληροφορία διαφέρει από την αντίστοιχη συνεχή που συναντάμε στη φύση σε δύο σημαντικές απόψεις: είναι δειγματική (sampled) και είναι κβαντισμένη (quantized). Οι δύο αυτοί παράγοντες περιορίζουν την ποσότητα της πληροφορίας που περιέχεται στο ψηφιακό σήμα. Παρακάτω θα εξετάσουμε συνοπτικά τις αρχές διαχείρισης της πληροφορίας έτσι ώστε να γίνει κατανοητό τι μέρος της πληροφορίας χρειάζεται να φυλάξουμε και τι επιτρέπεται να αγνοήσουμε. Κατά συνέπεια, αυτό θα μας υπαγορεύσει την επιλογή για τη συχνότητα δειγματοληψίας, τον αριθμό των bits καθώς και τον τύπο των αναλογικών φίλτρων που απαιτούνται για την μετατροπή μεταξύ αναλογικής και ψηφιακής φύσης της πληροφορίας.

Στο Σχήμα 2.1 φαίνονται οι κυματομορφές της τυπικής διαδικασίας ADC. Η εικόνα (a) δείχνει το αναλογικό σήμα που πρόκειται να ψηφιοποιηθεί και το οποίο είναι μια χρονικά μεταβαλλόμενη τάση. Υποθέτουμε για λόγους απλότητας ότι το πλάτος της τάσης μπορεί να μεταβληθεί από 0 έως 4,095 volts που αντιστοιχεί στους ψηφιακούς αριθμούς από 0 έως 4095 οι οποίοι θα παραχθούν από ένα ψηφιακό μετατροπέα των 12 bit. Το block διάγραμμα έχει χωρισθεί σε δύο τμήματα, το sample-and-hold και το ADC. Το πρώτο τμήμα απαιτείται για να διατηρηθεί σταθερή η τάση που εισέρχεται στον ADC για όσο χρόνο διαρκεί η διαδικασία της μετατροπής.

Όπως φαίνεται από τις διαφορές μεταξύ των διαγραμμάτων (a) και (b), η έξοδος του sample-and-hold μπορεί να αλλάξει μόνο σε περιοδικά χρονικά διαστήματα, κατά τη διάρκεια των οποίων η έξοδος αυτού παραμένει σταθερή και είναι ίδια με τη στιγμιαία τιμή της τάσης που εμφανίσθηκε στην είσοδο στην αρχή του παραπάνω χρονικού διαστήματος. Αλλαγές στο σήμα εισόδου που συμβαίνουν μεταξύ των χρονικών στιγμών δειγματοληψίας, αγνοούνται πλήρως. Έτσι, η δειγματοληψία μετατρέπει την ανεξάρτητη μεταβλητή (το χρόνο στην παρούσα περίπτωση) από συνεχή σε διακριτή.



Σχήμα 2.1 Διαδικασία Analog-To-Digital

Από τις διαφορές μεταξύ των διαγραμμάτων (b) και (c) φαίνεται ότι ο ADC παράγει μία ακέραια τιμή μεταξυ 0 και 4095 για κάθε επίπεδη περιοχή της κυματομορφής της τάσης στο διάγραμμα (b). Αυτό εισάγει ένα σφάλμα, καθώς κάθε επίπεδο μπορεί να είναι οποιαδήποτε τάση μεταξύ των τιμών 0 και 4,095V. Για παράδειγμα, τόσο η τάση των

2,56000 V όσο και αυτή των 2,56001 V αντιστοιχίζονται από τον ADC στον ακέραιο αριθμό 2560. Με άλλα λόγια, η κβάντιση μετατρέπει την εξαρτημένη μεταβλητή (την τάση εδώ) από συνεχή σε διακριτή. Επισημαίνουμε εδώ ότι στο διάγραμμα (c) που απεικονίζει το ψηφιοποιημένο σήμα, οι ψηφιακές στάθμες φαίνονται να συνδέονται μεταξύ τους (παχιές γραμμές). Όπως θα πούμε και παρακάτω, αυτή η αναπαράσταση καλό είναι να αποφεύγεται καθώς δίνει την ψευδή εντύπωση ότι έχουμε να κάνουμε με μια συνεχή κυματομορφή ως προς τον χρόνο, κάτι που δεν ισχύει για το ψηφιακό σήμα. Εδώ το κάνουμε για να φανεί καλύτερα το σφάλμα κβάντισης.

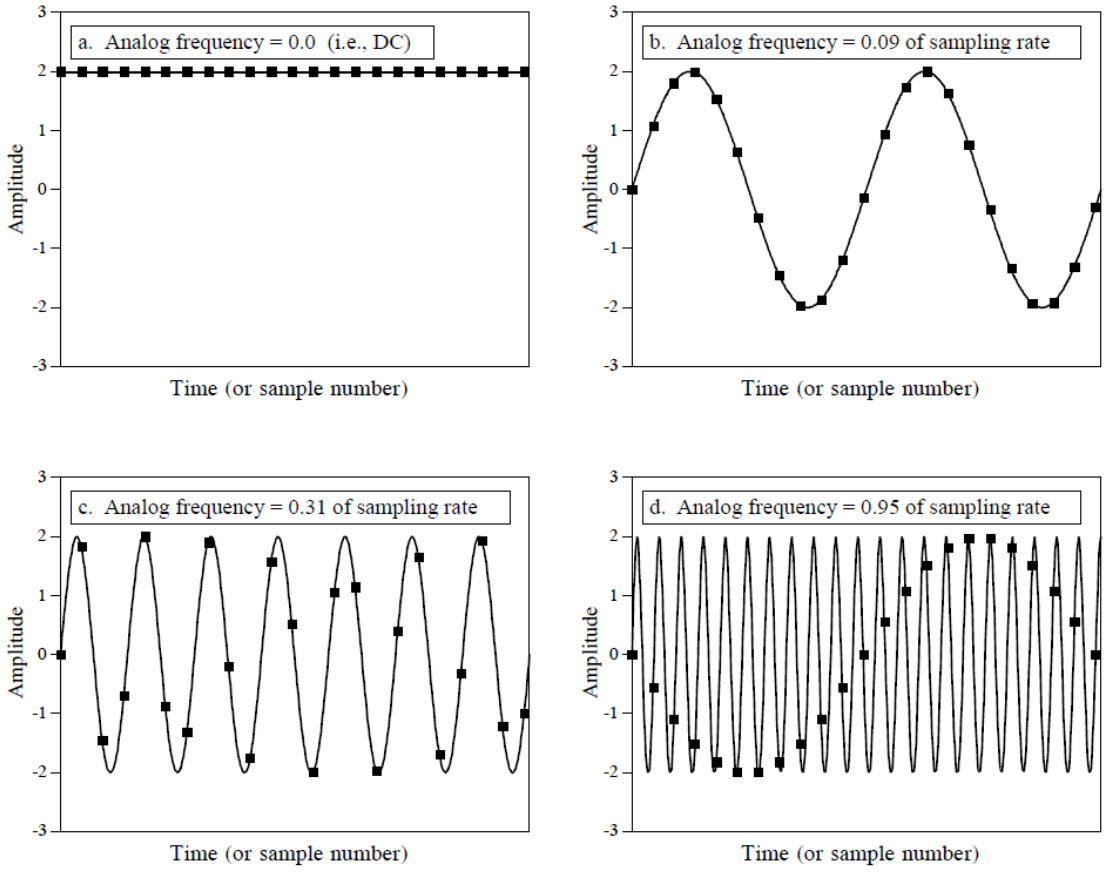
Εξετάζοντας πρώτα την επίδραση της κβάντισης, βλέπουμε ότι κάθε δείγμα του ψηφιοποιημένου σήματος μπορεί να παρουσιάζει μέγιστο σφάλμα ±1/2 LSB (Least Significant Bit). Το διάγραμμα (d) δείχνει το σφάλμα κβάντισης του συγκεκριμένου παραδείγματος, το οποίο προκύπτει με αφαίρεση της κυματομορφής (b) από την (c) μετά από κατάλληλη μετατροπή. Ένα σημαντικό στοιχείο της παραπάνω ανάλυσης είναι ότι το σφάλμα κβάντισης μοιάζει πολύ με τυχαίο θόρυβο.

Η τελευταία διαπίστωση θέτει τη βάση ενός σημαντικού μοντέλου του σφάλματος κβάντισης. Στις περισσότερες περιπτώσεις η κβάντιση επιφέρει την πρόσθεση συγκεκριμένης ποσότητας λευκού θορύβου στο σήμα. Ο πρόσθετος αυτός θόρυβος κατανέμεται μεταξύ ±1/2 LSB, έχει μέση τιμή 0 και τυπική απόκλιση $1/\sqrt{12}$ LSB (~0,29 LSB). Για παράδειγμα, περνώντας ένα αναλογικό σήμα από έναν 8bit ADC, προσθέτει θόρυβο με τιμή rms 0,29/256 ή περίπου 1/900 της full scale τιμής ενώ ένας 12 bit ADC προσθέτει αντίστοιχα θόρυβο 0,29/4096 που είναι περίπου 1/14000 της full scale τιμής. Καθώς το σφάλμα κβάντισης είναι τυχαίος θόρυβος, ο αριθμός των bits καθορίζει την ακρίβεια των δεδομένων.

Το παραπάνω μοντέλο είναι πολύ σημαντικό καθώς ο τυχαίος θόρυβος που παράγεται από τη κβάντιση, προστίθεται στον οποιοδήποτε θόρυβο που υπάρχει στο αναλογικό σήμα. Για παράδειγμα, φανταστείτε ένα αναλογικό σήμα με μέγιστο πλάτος 1 Volt και τυχαίο θόρυβο του 1 milivolt rms. Ψηφιοποιώντας αυτό το σήμα σε 8 bits έχουμε ως αποτέλεσμα 1 Volt να μετατρέπεται στον ψηφιακό αριθμό 255, και 1 milivolt να μετατρέπεται σε 0,255 LSB. Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, τυχαία επίπεδα θορύβου σήματος συνδυάζονται προσθέτοντας τα τετράγωνα τυπικής απόκλισής τους. Δηλαδή, τα σήματα προσθέτονται ως εξής : $\sqrt{\mathrm{A}^2 + \mathrm{B}^2} = C$. Ο συνολικός θόρυβος στο ψηφιοποιημένο σήμα είναι επομένως : $\sqrt{0{,}255^2 + 0{,}29^2} = 0{,}386$ LSB. Αυτό είναι μια αύξηση περίπου 50% πέρα από το θόρυβο που είχε ήδη το αναλογικό σήμα. Ψηφιοποώντας το ίδιο σήμα στα 12 bits δεν θα παρήγαγε θεωρητικά καμία αύξηση στο θόρυβο, και τίποτα δεν θα χανόταν κατά την κβαντοποίηση. Όταν έρχεσαι αντιμέτωπος με την απόφαση του πόσα bits χρειάζονται για ένα σύστημα πρέπει να έχεις στο μυαλό σου δύο ερωτήσεις: (1) Πόσος θόρυβος υπάρχει ήδη στο αναλογικό σήμα; (2) Πόσος θόρυβος μπορεί να γίνει ανεκτός στο ψηφιοποιημένο σήμα;

2.1.1 Το θεώρημα δειγματοληψίας

Ο ορισμός της σωστής δειγματοληψίας είναι αρκετά απλός. Υποθέστε ότι δειγματοληπτείται ένα σήμα με κάποιο τρόπο. Εάν μπορείτε να αναδομήσετε ακριβώς το αναλογικό σήμα από τα δείγματα, θα έχετε πραγματοποιήσει τη δειγματοληψία σωστά. Ακόμη και αν τα δεδομένα δειγματοληψίας φαίνονται να είναι μπερδεμένα ή ημιτελή, οι σημαντικές πληροφορίες έχουν συλληφθεί, εάν μπορείτε να αντιστρέψετε τη διαδικασία.



Σχήμα 2.2 Δειγματοληψία Ημιτονικών Σημάτων

Το Σχήμα 2.2 δείχνει διάφορες ημιτονικές καμπύλες πριν και μετά την ψηφιοποίηση. Οι συνεχείς γραμμές αναπαριστούν το αναλογικό σήμα το οποίο εισέρχεται στο ADC ενώ τα σημεία με τα τετράγωνα είναι το ψηφιακό σήμα που εξέρχεται από τον ADC. Στο (a), το αναλογικό σήμα είναι μία σταθερή DC τιμή, δηλαδή ένα κύμα συνημιτόνου μηδενικής συχνότητας. Δεδομένου ότι το αναλογικό σήμα είναι μια σειρά ευθειών γραμμών μεταξύ κάθε ενός από τα δείγματα, όλες οι πληροφορίες που απαιτούνται για να αναδημιουργήσουν το αναλογικό σήμα περιλαμβάνονται στην ψηφιακά δεδομένα. Σύμφωνα με τον ορισμό μας, αυτό είναι η σωστή δειγματοληψία.

Το κύμα ημιτόνου που παρουσιάζεται στο (b) έχει μια συχνότητα 0.09 του ρυθμού δειγματοληψίας. Αυτό θα μπορούσε να αντιπροσωπεύσει παραδείγματος χάριν, ένα κύμα ημιτόνου συχνότητας 90 cycle/sec που δειγματοληπτείται στα 1000 samples/sec. Αν το θέσουμε διαφορετικά, υπάρχουν 11.1 δείγματα που λαμβάνονται κατά τη διάρκεια κάθε πλήρη κύκλου του ημιτονοειδούς. Αυτή η κατάσταση είναι πιο περίπλοκη από την

προηγούμενη περίπτωση, επειδή το αναλογικό σήμα δεν μπορεί να αναδημιουργηθεί απλά σύροντας τις ευθείες γραμμές μεταξύ των σημείων δειγματοληψίας. Αυτά τα δείγματα αντιπροσωπεύουν κατάλληλα το αναλογικό σήμα; Η απάντηση είναι ναι, επειδή κανένα άλλο ημιτονικό ή συνδυασμός ημιτονικών σημάτων, δεν θα μπορούσε να παραγάγει αυτό το μοτίβο των δειγμάτων (μέσα στους λογικούς περιορισμούς που απαριθμούνται παρακάτω). Αυτά τα δείγματα αντιστοιχούν μόνο σε ένα συγκεκριμένο αναλογικό σήμα, και επομένως το αναλογικό σήμα μπορεί να είναι αναδημιουργηθεί ακριβώς. Πάλι έχουμε επομένως μια περίπτωση ορθής δειγματοληψίας.

Στο (c), η κατάσταση γίνεται ακόμη δυσκολότερη με την αύξηση της συχνότητας του ημιτονικού σήματος σε 0.31 του ρυθμού δειγματοληψίας. Αυτό οδηγεί σε μόνο 3.2 δείγματα ανά πλήρη κύκλο του ημιτόνου. Εδώ τα δείγματα είναι τόσο αραιά που ούτε καν εμφανίζονται για να ακολουθήσει τη γενική μορφή του αναλογικού σήματος. Αντιπροσωπεύουν αυτά τα δείγματα κατάλληλα το αναλογικό κυματοειδές; Πάλι, η απάντηση είναι ναι, και για ακριβώς τον ίδιο λόγο. Τα δείγματα είναι μια μοναδική αναπαράσταση του αναλογικού σήματος. Όλες οι πληροφορίες που απαιτούνται για να αναδημιουργήσουν το συνεχή κυματομορφή, περιλαμβάνονται στα ψηφιακά δεδομένα. Ο τρόπος που γίνεται αυτό θα συζητηθεί αργότερα. Προφανώς, πρέπει να είναι πιο περίπλοκο από έναν απλό σχεδιασμό των ευθειών γραμμών μεταξύ των σημείων δειγματοληψίας. Όσο περίεργο κι εάν φαίνεται, αυτό αποτελεί τη σωστή δειγματοληψία σύμφωνα με τον ορισμό που δόθηκε στην αρχή.

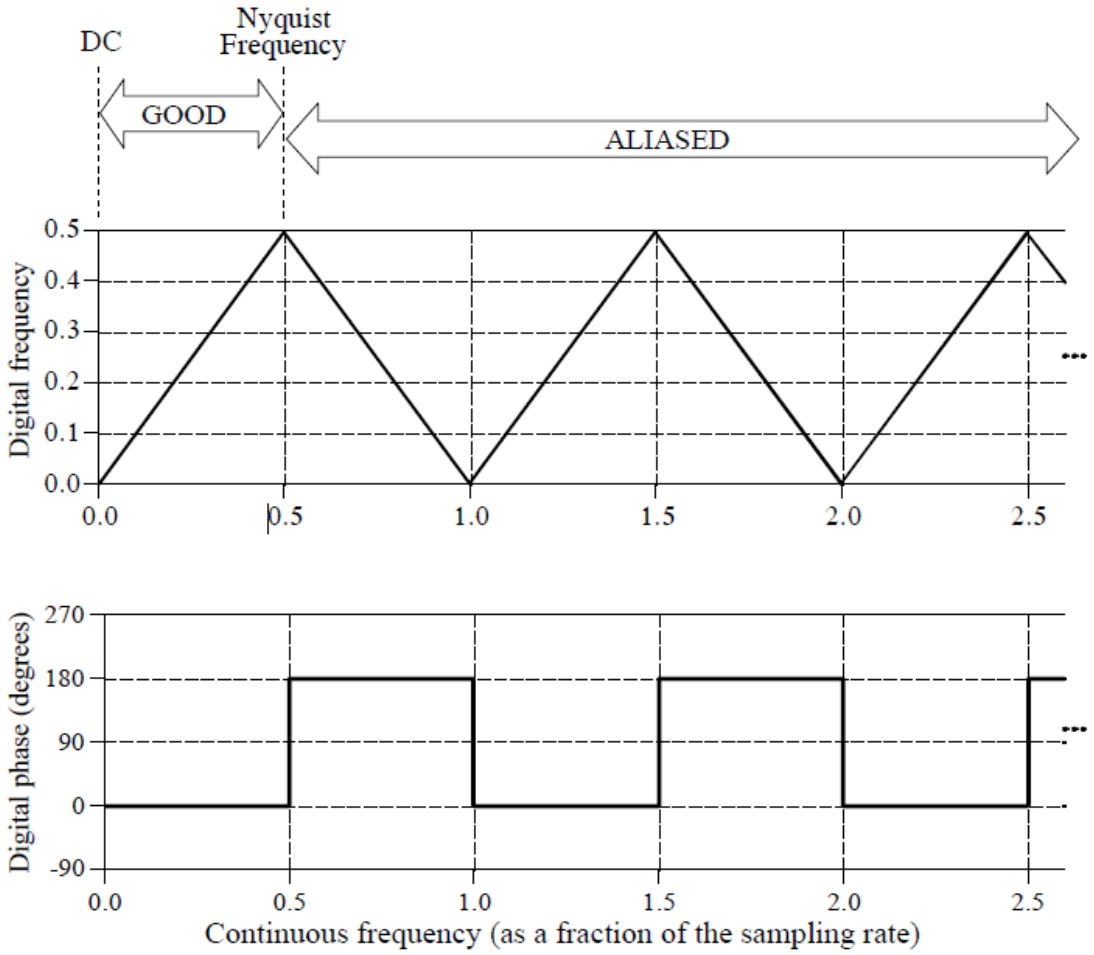
Στο (d), η αναλογική συχνότητα ωθείται ακόμα υψηλότερα, σε 0.95 του ρυθμού δειγματοληψίας, με 1.05 δείγματα ανά πλήρη κύκλο του ημιτόνου. Αντιπροσωπεύουν αυτά τα δείγματα κατάλληλα τα στοιχεία δεδομένων; Όχι..! Τα δείγματα αντιπροσωπεύουν μία διαφορετική κυματομορφή από αυτή που περιλαμβάνεται στο αναλογικό σήμα. Συγκεκριμένα, το αρχικό ημίτονο συχνότητας 0.95 του ρυθμού δειγματοληψίας παρουσιάζεται εσφαλμένα ως κυματομορφή συχνότητας 0.05 του ρυθμού δειγματοληψίας στο ψηφιακό σήμα. Αυτό το φαινόμενο των ημιτονικών σημάτων να αλλάζουν τη συχνότητά τους κατά τη διάρκεια της δειγματοληψίας καλείται **aliasing** (απόκτηση ψευδωνύμου). Όπως ακριβώς ένας εγκληματίας θα λάμβανε ένα ψευδές όνομα ή ταυτότητα, η κυματομορφή αποτιμάται σε μία άλλη συχνότητα που δεν είναι η δική της. Δεδομένου ότι τα ψηφιακά δεδομένα δεν συσχετίζονται πλέον με κάποιο συγκεκριμένο αναλογικό σήμα, μια σαφής αναδημιουργία είναι αδύνατη. Δεν υπάρχει τίποτα στα δείγματα δεδομένων που να δηλώνει ότι το αρχικό αναλογικό σήμα είχε συχνότητα 0.95 και όχι 0.05 του ρυθμού δειγματοληψίας. Η κυματομορφή έχει κρύψει την αληθινή ταυτότητά της εντελώς. Σύμφωνα με τον ορισμό μας, αυτό είναι παράδειγμα της μη ορθής δειγματοληψίας.

Αυτή η γραμμή συλλογισμού οδηγεί σε ένα κύριο σημείο της Ψηφιακής επεξεργασίας (DSP), στο **θεώρημα δειγματοληψίας**. Συχνά αυτό καλείται **θεώρημα Shannon**, ή **θεώρημα Nyquist**, μετά από τις επιστημονικές ανακοινώσεις των συντακτών για το θέμα στη δεκαετία του 1940. Το θεώρημα δειγματοληψίας δείχνει ότι ένα συνεχές σήμα μπορεί να υποστεί δειγματοληψία σωστά, μόνο εάν δεν περιέχει συχνότητες μεγαλύτερες από το μισό του ρυθμού δειγματοληψίας. Για παράδειγμα, ένας ρυθμός δειγματοληψίας 2.000

samples/sec απαιτεί το αναλογικό σήμα να αποτελείται από συχνότητες κάτω από 1000 cycles/sec. Εάν σε ένα σήμα υπάρχουν συχνότητες πάνω από αυτό το όριο, αυτές θα αντιστοιχηθούν εσφαλμένα σε συχνότητες μεταξύ 0 και 1000 cycles/sec, παραποιώντας οποιεσδήποτε πληροφορίες που σωστά προϋπήρχαν στις αντίστοιχες θέσεις.

Δύο όροι χρησιμοποιούνται ευρέως στο θεώρημα δειγματοληψίας: **Συχνότητα Nyquist** και **ρυθμός Nyquist**. Δυστυχώς, η σημασία τους δεν είναι προτυποποιημένη. Για να το καταλάβουμε αυτό, θεωρούμε ένα αναλογικό σήμα που αποτελείται από συχνότητες μεταξύ DC και 3 kHz. Για να ψηφιοποιηθεί κατάλληλα αυτό το σήμα πρέπει να γίνει η δειγματοληψία του με 6.000 samples/sec (6 kHz) ή περισσότερο. Υποθέστε ότι επιλέγουμε δείγματα με ρυθμό 8.000 samples/sec (8 kHz), επιτρέποντας να απεικονισθούν κατάλληλα όλες οι συχνότητες μεταξύ DC και 4 kHz. Σε αυτήν την κατάσταση, υπάρχουν τέσσερις σημαντικές συχνότητες: (1) η υψηλότερη συχνότητα στο σήμα, 3 kHz (2) δύο φορές αυτή η συχνότητα, 6 kHz (3) ο ρυθμός δειγματοληψίας, 8 kHz και (4) το μισό του ρυθμού δειγματοληψίας, 4 kHz. Ποιο από αυτά τα τέσσερα είναι η συχνότητα Nyquist και ποιο από αυτά είναι ο ρυθμός Nyquist; Εξαρτάται ποιον ρωτάτε! Όλοι οι πιθανοί συνδυασμοί χρησιμοποιούνται. Ευτυχώς, οι περισσότεροι συγγραφείς είναι προσεκτικοί στο να καθορίσουν πώς χρησιμοποιούν οι ίδιοι αυτούς τους όρους. Στο κείμενο αυτό, και οι δύο χρησιμοποιούνται για να σημαίνουν το μισό του ρυθμού δειγματοληψίας.

Στο Σχήμα 2.3 φαίνεται πώς οι συχνότητες αλλάζουν κατά τη διάρκεια του aliasing. Το σημείο κλειδί που πρέπει να θυμόμαστε είναι ότι ένα ψηφιακό σήμα δεν μπορεί να περιέχει συχνότητες μεγαλύτερες από το μισό του ρυθμού δειγματοληψίας (δηλαδή τη συχνότητα/ρυθμό Nyquist). Όταν η συχνότητα της συνεχούς κυματομορφής είναι κάτω από το ρυθμό Nyquist, η συχνότητα που προκύπτει από τα δεδομένα δειγματοληψίας συμπίπτει με αυτή. Εντούτοις, όταν η συχνότητα του συνεχούς σήματος είναι μεγαλύτερη από το ρυθμό Nyquist, το aliasing αλλάζει τη συχνότητα αυτή σε κάποια μικρότερη, η οποία μπορεί να αναπαρασταθεί από τα δεδομένα δειγματοληψίας. Όπως φαίνεται από τη ζιγκ-ζαγκ γραμμή στο γράφημα 2.3, κάθε συχνότητα του συνεχούς σήματος επάνω από το ρυθμό Nyquist έχει μια αντίστοιχη ψηφιακή συχνότητα μεταξύ του μηδενός και του μισού του ρυθμού δειγματοληψίας. Εάν συμβεί να υπάρχει ήδη ένα ημιτονοειδές σήμα σε αυτή τη χαμηλότερη συχνότητα, το αλλαγμένο σήμα θα προστεθεί σε αυτό, έχοντας ως αποτέλεσμα την απώλεια πληροφοριών. Το aliasing αποτελεί μια διπλή πληγή: οι πληροφορίες μπορεί να χαθούν για τις υψηλότερες αλλά και τις χαμηλότερες συχνότητες. Υποθέστε ότι δίνεται ένα ψηφιακό σήμα το οποίο περιέχει μια συχνότητα 0.2 του ρυθμού δειγματοληψίας. Εάν αυτό το σήμα προέκυψε από σωστή δειγματοληψία, το αρχικό αναλογικό σήμα θα πρέπει να περιέχει μια συχνότητα 0.2. Εάν υπάρχει aliasing κατά τη διάρκεια της δειγματοληψίας, η ψηφιακή συχνότητα των 0.2 μπορεί να προήλθε από οποιοδήποτε άπειρο αριθμό συχνοτήτων στο αναλογικό σήμα: 0.2, 0.8, 1.2, 1.8, 2.2, ...



Σχήμα 2.3 Μετατροπή αναλογικών Συχνοτήτων σε ψηφιακές κατά τη δειγματοληψία

Ακριβώς όπως το aliasing μπορεί να αλλάξει τη συχνότητα κατά τη διάρκεια της δειγματοληψίας, μπορεί επίσης να αλλάξει και τη φάση. Για παράδειγμα, κοιτάξτε το παραποιημένο σήμα στο γράφημα d του Σχήματος 2.2. Το σήμα που προέκυψε από τη δειγματοληψία είναι το αντεστραμμένο του αυθεντικού αναλογικού σήματος. Με άλλα λόγια, το aliasing έχει αλλάξει τη συχνότητα και επιπρόσθετα εισήγαγε μια μετατόπιση φάσης κατά 180^o. Μόνο δύο μετατοπίσεις είναι δυνατές: 0ο (καμία μετατόπιση-διαφορά φάσης) και 180^o (αντιστροφή). Η μηδενική μετατόπιση φάσης εμφανίζεται για αναλογικές συχνότητες από 0 έως 0.5, 1.0 έως 1.5, 2.0 έως 2.5, κ.λπ. η αντεστραμμένη φάση εμφανίζεται για τις αναλογικές συχνότητες από 0.5 έως 1.0, 1.5 έως 2.0, 2.5, 3.0 και ούτω καθεξής.

Σε μια πιό λεπτομερή ανάλυση της δειγματοληψίας και πως πραγματοποιείται το aliasing, ο γενικός στόχος μας είναι να καταλάβουμε τι συμβαίνει στις πληροφορίες όταν ένα σήμα μετατρέπεται από μια συνεχή σε μια διακριτή μορφή. Το πρόβλημα είναι ότι αυτά είναι πολύ διαφορετικά πράγματα για να συγκριθούν: το ένα είναι μία συνεχής κυματομορφή ενώ το άλλο είναι ένας πίνακας από αριθμούς. Αυτή η «μήλα σε πορτοκάλια» σύγκριση

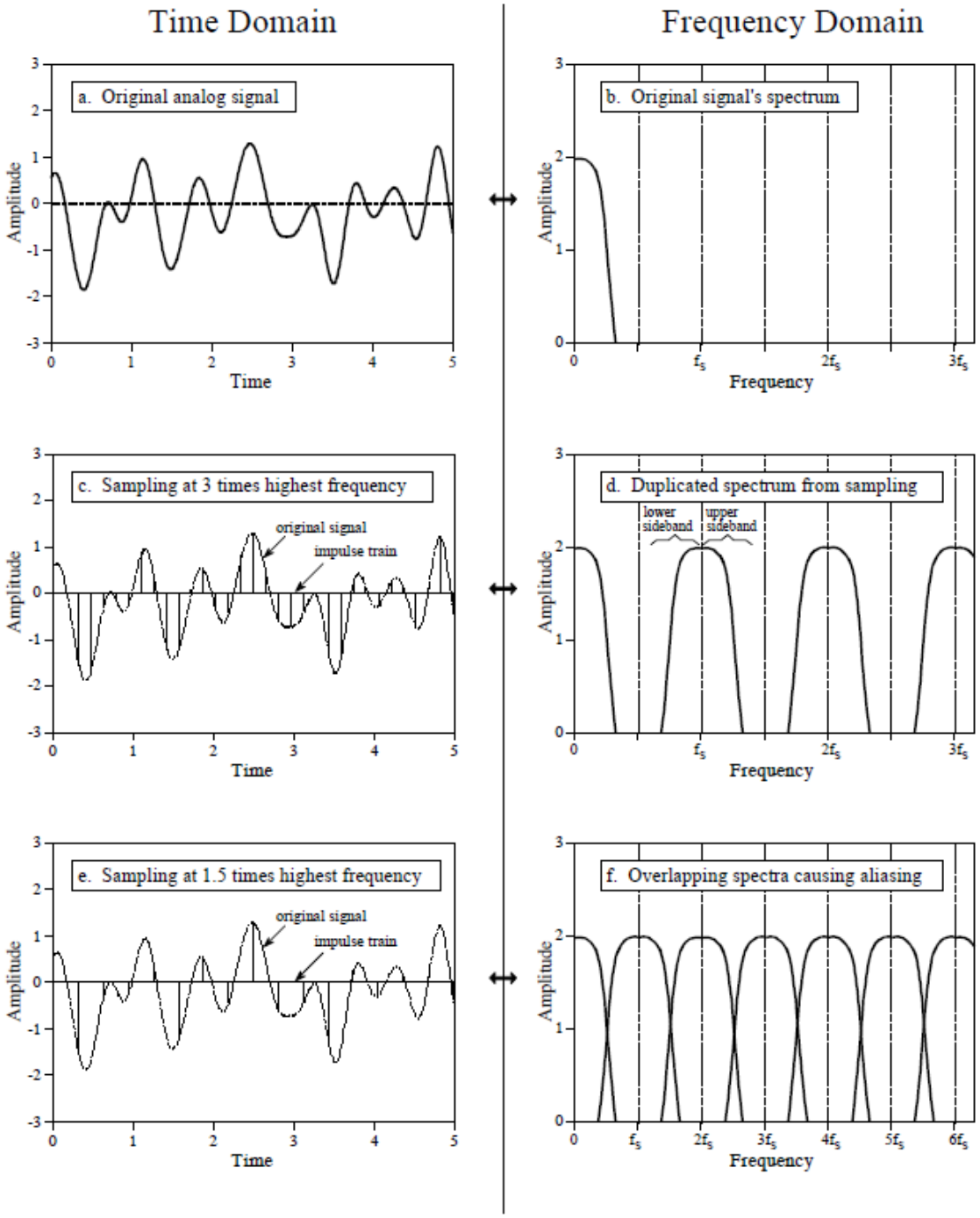
κάνει την ανάλυση πιο δύσκολη. Η λύση έρχεται με την εισαγωγή μιας θεωρητικής έννοιας που αποκαλείται ακολουθία κρουστικών παλμών.

Το γράφημα (a) του Σχήματος 2.4 παρουσιάζει ένα παράδειγμα αναλογικού σήματος. Το γράφημα (c) παρουσιάζει το σήμα που δειγματοληπτείται με τη χρησιμοποίηση μιας ακολουθίας κρουστικών παλμών. Η ακολουθία κρουστικών παλμών είναι ένα συνεχές σήμα αποτελούμενο από τετραγωνικούς παλμούς πολύ μικρής διάρκειας (κρουστικοί παλμοί - impulses) και οι οποίοι συμπίπτουν με τις στιγμές δειγματοληψίας του αρχικού σήματος. Μεταξύ αυτών των χρόνων δειγματοληψίας η τιμή της κυματομορφής είναι μηδέν. Λάβετε υπόψη ότι η ακολουθία κρουστικών παλμών είναι μια θεωρητική έννοια, όχι μία κυματομορφή που μπορεί να υπάρχει σε ένα ηλεκτρικό κύκλωμα. Αφού και το αρχικό αναλογικό σήμα και η ακολουθία κρουστικών παλμών είναι συνεχείς κυματομορφές, μπορούμε πλέον να κάνουμε σύγκριση μεταξύ αυτών των δύο.

Τώρα πρέπει να εξετάσουμε τη σχέση μεταξύ της ακολουθίας κρουστικών παλμών και του διακριτού σήματος (ένας πίνακας αριθμών). Αυτό είναι εύκολο: από την άποψη των περιεχόμενων πληροφοριών, είναι πανομοιότυπα. Εάν το ένα είναι γνωστό, είναι εύκολο να υπολογίσουμε και το άλλο. Τα σκεφτόμαστε ως τις άκρες μιας γέφυρας που διασχίζει και ενώνει τον αναλογικό με τον ψηφιακό κόσμο. Αυτό σημαίνει ότι επιτύχαμε τον απώτερο σκοπό μας να καταλάβουμε τις συνέπειες μετατροπής της κυματομορφής του γραφήματος (a) στη κυματομορφή (c).

Στην αριστερή στήλη του Σχήματος 2.4 φαίνονται τρεις συνεχείς κυματομορφές. Τα αντίστοιχα **φάσματα συχνοτήτων** φαίνονται στη δεξιά στήλη. Αυτή η έννοια είναι γνωστή από τις γνώσεις μας στα ηλεκτρονικά: κάθε κυματομορφή μπορεί να θεωρηθεί ότι απαρτίζεται από ημιτονικά σήματα με διάφορα πλάτη και συχνότητες.

Το γράφημα (a) παρουσιάζει το αναλογικό σήμα στο οποίο θέλουμε να πραγματοποιήσουμε δειγματοληψία. Όπως φαίνεται από το διάγραμμα συχνοτήτων του στο (b), αποτελείται μόνο από τις συνιστώσες συχνοτήτων μεταξύ 0 και περίπου 0.33 fs, όπου fs είναι η συχνότητα δειγματοληψίας που σκοπεύουμε να χρησιμοποιήσουμε. Για παράδειγμα, αυτό μπορεί να είναι ένα σήμα ομιλίας που έχει φιλτραριστεί για να αφαιρεθούν όλες οι συχνότητες πάνω από τα 3,3 kHz. Αντίστοιχα, το fs θα είναι 10kHz (10.000 samples/sec), ο επιδιωκόμενος ρυθμός δειγματοληψίας.



Σχήμα 2.4 Το θεώρημα δειγματοληψίας στο πεδίο χρόνου και συχνότητας

Πραγματοποιώντας δειγματοληψία στο σήμα (a) χρησιμοποιώντας μια ακολουθία κρουστικών παλμών, προκύπτει το σήμα που φαίνεται στο (c), καθώς και το διάγραμμα φάσματος συχνοτήτων του στο (d). Αυτό το φάσμα είναι αντίγραφο του φάσματος του αρχικού σήματος. Κάθε πολλαπλάσιο της συχνότητας δειγματοληψίας, fs, 2fs, 3fs, 4fs, κ.λπ., έχει λάβει ένα αντίγραφο φάσματος συχνοτήτων του αρχικού σήματος. Το αντίγραφο ονομάζεται upper sideband (USB, άνω πλευρική ζώνη συχνοτήτων), ενώ το αντεστραμμένο αντίγραφο καλείται lower sideband (LSB, κάτω πλευρική ζώνη συχνοτήτων). Η δειγματοληψία έχει δημιουργήσει νέες συχνότητες. Είναι όμως αυτό ορθή

δειγματοληψία; Η απάντηση είναι ναι, επειδή από το σήμα στο (c) μπορεί να προκύψει το αρχικό σήμα στο (a) με την εξάλειψη όλων των συχνοτήτων επάνω από ½fs. Δηλαδή, ένα αναλογικό χαμηλής διέλευσης φίλτρο θα μετατρέψει την ακολουθία κρουστικών παλμών που φαίνεται στο (b) στο αρχικό αναλογικό σήμα, (a).

Στο πεδίο του χρόνου, η δειγματοληψία επιτυγχάνεται με πολλαπλασιασμό του αρχικού σήματος με μια ακολουθία κρουστικών παλμών μοναδιαίου πλάτους. Το φάσμα συχνοτήτων αυτής της ακολουθίας κρουστικών παλμών είναι επίσης μια ακολουθία κρουστικών παλμών μοναδιαίου πλάτους, με τα μέγιστα κυματομορφής να εμφανίζονται στα πολλαπλάσια της συχνότητα δειγματοληψίας fs, 2fs, 3fs, 4fs, κ.λπ. Όταν δύο σήματα πολλαπλασιάζονται στο πεδίο του χρόνου, τα αντίστοιχα φάσματα συχνότητάς τους συνελλίσονται (convolved). Αυτό οδηγεί στο διπλασιασμό του αρχικού φάσματος σε κάθε σημείο όπου υπάρχει μέγιστο πλάτος στο φάσμα της ακολουθίας κρουστικών παλμών. Το αρχικό σήμα εμφανίζεται να αποτελείται από τις θετικές και τις αρνητικές συχνότητες, που είναι η άνω και κάτω πλευρικές ζώνες, αντίστοιχα. Αυτό είναι το ίδιο φαινόμενο με τη διαμόρφωση κατά πλάτος.

Το γράφημα (e) Σχήματος 2.4 παρουσιάζει ένα παράδειγμα μη ορθής δειγματοληψίας, ως αποτέλεσμα του πάρα πολύ χαμηλού ρυθμού δειγματοληψίας. Το αναλογικό σήμα περιέχει όπως πριν τις συχνότητες μέχρι και 3.3 kHz, αλλά ο ρυθμός δειγματοληψίας έχει μειωθεί τώρα στα 5 kHz. Σημειώστε ακόμα ότι τα fs, 2fs, 3fs,…, κατά μήκος του οριζόντιου άξονα είναι τοποθετημένα σε πιό κοντινές αποστάσεις στο (f) από ότι στο (d). Το φάσμα συχνοτήτων, (f), δείχνει το πρόβλημα: τμήματα από τα αντίγραφα του φάσματος συχνοτήτων επικαλύπτουν τη ζώνη συχνοτήτων μεταξύ 0 και του μισού της συχνότητας δειγματοληψίας. Αν και το (f) παρουσιάζει αυτές τις επικαλυπτόμενες συχνότητες να διατηρούν την ξεχωριστή τους ταυτότητα, πρακτικά προστίθενται, δημιουργώντας έτσι ένα μεγάλο μπέρδεμα. Με δεδομένο ότι δεν υπάρχει κανένας τρόπος να διαχωριστούν οι επικαλυπτόμενες συχνότητες, οι πληροφορίες χάνονται, και το αρχικό σήμα δεν μπορεί να αναδημιουργηθεί. Αυτή η επικάλυψη εμφανίζεται όταν *το αναλογικό σήμα περιέχει συχνότητες μεγαλύτερες από το μισό του ρυθμού δειγματοληψίας*, δηλαδή, έχουμε αποδείξει το θεώρημα δειγματοληψίας.

2.1.2 Digital-to-Analog Conversion (Μετατροπή ψηφιακού σε αναλογικό)

Θεωρητικά, η απλούστερη μέθοδος για τη digital-to-analog μετατροπή είναι να ανακληθούν τα δείγματα από τη μνήμη και να μετατραπούν σε μία ακολουθία κρουστικών παλμών. Αυτό φαίνεται στο γράφημα 2.5a, με το αντίστοιχο φάσμα συχνοτήτων στο (b). Όπως μόλις περιγράφηκε, το αρχικό αναλογικό σήμα μπορεί να αναδημιουργηθεί τέλεια με την διάβαση αυτής της σειράς κρουστικών παλμών μέσα από ένα χαμηλής διέλευσης φίλτρο, με συχνότητα αποκοπής ίση με το μισό του ρυθμού δειγματοληψίας. Με άλλα λόγια, το αρχικό σήμα και η ακολουθία κρουστικών παλμών έχουν όμοια φάσματα συχνοτήτων για την περιοχή κάτω από τη συχνότητα Nyquist. Στις υψηλότερες συχνότητες, η σειρά κρουστικών παλμών παρουσιάζει αντίγραφα της πληροφορίας αυτής, ενώ το αρχικό αναλογικό σήμα δεν περιέχει τίποτα (αν υποθέσουμε ότι δεν υπάρχει aliasing).

Ενώ αυτή η μέθοδος δουλεύει από μαθηματική άποψη, είναι δύσκολο να παραχθούν οι απαραίτητοι κρουστικοί παλμοί στην ηλεκτρονική. Για να μπορέσουμε να το προσπεράσουμε αυτό, σχεδόν όλα τα DACs λειτουργούν με τη διατήρηση της τελευταίας τιμής έως ότου παραληφθεί το επόμενο δείγμα. Αυτό καλείται **zeroth-order hold**, το ισοδύναμο του sample-and-hold που χρησιμοποιείται κατά τη διάρκεια της ADC διαδικασίας. (Η first-order hold είναι ευθείες γραμμές μεταξύ των σημείων, η second-order-hold κάνει χρήση παραβολών, κ.λπ.). Η zeroth-order-hold παράγει την εμφάνιση σκαλοπατιών όπως φαίνονται στο 2.5c.

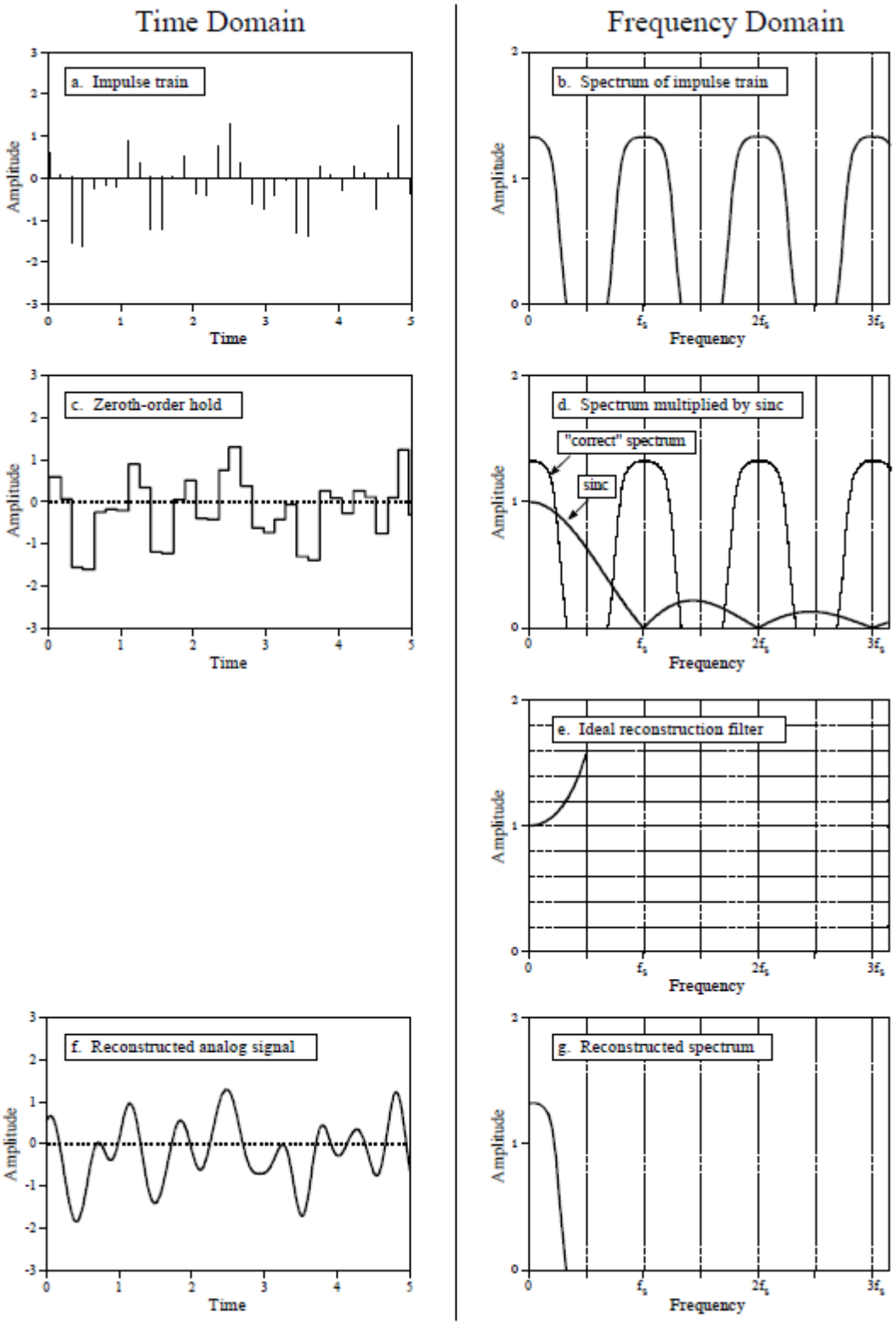
Στο πεδίο συχνότητας, η zeroth-order-hold οδηγεί στον πολλαπλασιασμό του φάσματος της ακολουθίας κρουστικών με τη σκούρόχρωμη καμπύλη που φαίνεται στο 2.5d και δίνεται από την εξίσωση:

$$H(f) = \left| \frac{\sin(\pi f / f_s)}{\pi f / f_s} \right|$$

Αυτή είναι της γενικής μορφής: sin(πx)/(πx), και καλείται **συνάρτηση sinc** ή **sinc(x)**. Η συνάρτηση sinc είναι πολύ γνωστή στη DSP. Έχοντας ήδη ένα γνωστικό υπόβαθρο, βλέπουμε ότι η zeroth-order-hold μπορεί να γίνει κατανοητή ως η συνέλιξη της ακολουθίας κρουστικών παλμών με έναν τετραγωνικό παλμό, που έχει πλάτος ίσο με την περίοδο δειγματοληψίας. Αυτό οδηγεί, στο πεδίο συχνότητας, στον πολλαπλασιασμό του φάσματος αυτής με το μετασχηματισμό Fourier του ορθογώνιου παλμού, δηλαδή τη συνάρτηση sinc. Στο γράφημα 2.5d, η λεπτή γραμμή παρουσιάζει το φάσμα συχνοτήτων της ακολουθίας κρουστικών παλμών (το «σωστό» φάσμα), ενώ η πιο έντονη γραμμή παρουσιάζει τη συνάρτηση sinc. Το φάσμα συχνοτήτων του σήματος zeroth-order-hold είναι ίσο με το γινόμενο αυτών των δύο καμπυλών.

Το αναλογικό φίλτρο που χρησιμοποιείται για να μετατρέψει το σήμα zeroth-order-hold, 2.5c, στο αναδομημένο σήμα 2.5f, πρέπει να κάνει δύο πράγματα: (1) να αφαιρεί όλες τις συχνότητες που είναι μεγαλύτερες από το μισό του ρυθμού δειγματοληψίας, και (2) να ενισχύει τις συχνότητες κατά 1/sinc(x), δηλαδή κατά το αντίστροφο της zeroth-order-hold. Αυτό επιφέρει ενίσχυση περίπου 36% του μισού του ρυθμού δειγματοληψίας. Το γράφημα 2.5e παρουσιάζει την ιδανική απόκριση συχνότητας αυτού του αναλογικού φίλτρου.

Αυτή η αύξηση συχνότητας κατά 1/sinc(x) μπορεί να αντιμετωπιστεί με τέσσερις τρόπους: (1) Να αγνοηθεί και να δεχτούμε τις συνέπειες, (2) Να σχεδιάσουμε ένα αναλογικό φίλτρο όπου θα συμπεριληφθεί η απόκριση 1/sinc(x), (3) Να χρησιμοποιείστε μια multirate τεχνική (που δεν περιγράφεται εδώ) ή (4) Να κάνουμε τη διόρθωση στο λογισμικό που προηγείται της DAC.



Σχήμα 2.5 Ανάλυση της μετατροπής Digital-to-Analog

Πρίν αφήσουμε αυτό το κεφάλαιο, πρέπει να διαλύσουμε έναν κοινό μύθο για τα αναλογικά έναντι των ψηφιακών σημάτων. Όπως περιγράφηκε, η ποσότητα πληροφοριών που υπάρχει σε ένα ψηφιακό σήμα περιορίζεται με δύο τρόπους: Κατ' αρχήν, ο αριθμός των bits ανά δείγμα περιορίζει την ανάλυση της εξαρτώμενης μεταβλητής. Δηλαδή οι

μικρές αλλαγές στο πλάτος του σήματος μπορούν να χαθούν στο θόρυβο κβαντοποίησης. Δεύτερον, ο ρυθμός δειγματοληψίας περιορίζει την ανάλυση της ανεξάρτητης μεταβλητής, δηλαδή, πολύ κοντινά γεγονότα στο αναλογικό σήμα μπορεί να χαθούν μεταξύ των δειγμάτων. Αυτό είναι ένας άλλος τρόπος για να πούμε ότι οι συχνότητες επάνω από το μισό του ρυθμού δειγματοληψίας χάνονται.

Ο μύθος που προαναφέρθηκε είναι ο εξής: « Καθώς τα αναλογικά σήματα χρησιμοποιούν συνεχείς παραμέτρους, έχουν απείρως καλή ανάλυση τόσο στην ανεξάρτητη όσο και στην εξαρτημένη μεταβλητή τους». Αυτό δεν είναι αλήθεια! Τα αναλογικά σήματα υποφέρουν από τα δύο ίδια προβλήματα που αναφέρθηκαν για τα ψηφιακά: θόρυβος και εύρος ζώνης συχνοτήτων. Ο θόρυβος στα αναλογικά σήματα περιορίζει τη μέτρηση του πλάτους της κυματομορφής όπως ακριβώς ο θόρυβος κβάντισης σε ένα ψηφιακό σήμα. Ομοίως, η ικανότητα να διαχωρισθούν πολύ κοντινά χρονικά γεγονότα σε ένα αναλογικό σήμα εξαρτάται από τη μέγιστη συχνότητα που επιτρέπεται στην κυματομορφή. Για να το καταλάβουμε αυτό, θεωρούμε ένα αναλογικό σήμα που περιέχει δύο πολύ κοντινούς χρονικά παλμούς. Αν οδηγήσουμε το σήμα αυτό μέσα από ένα βαθυπερατό φίλτρο (αφαιρώντας τις υψηλές συχνότητες), οι δύο παλμοί θα ενσωματωθούν σε μία ενιαία και παραποιημένη μορφή παλμού.

2.2 Διακριτές ακολουθίες και ο συμβολισμός τους

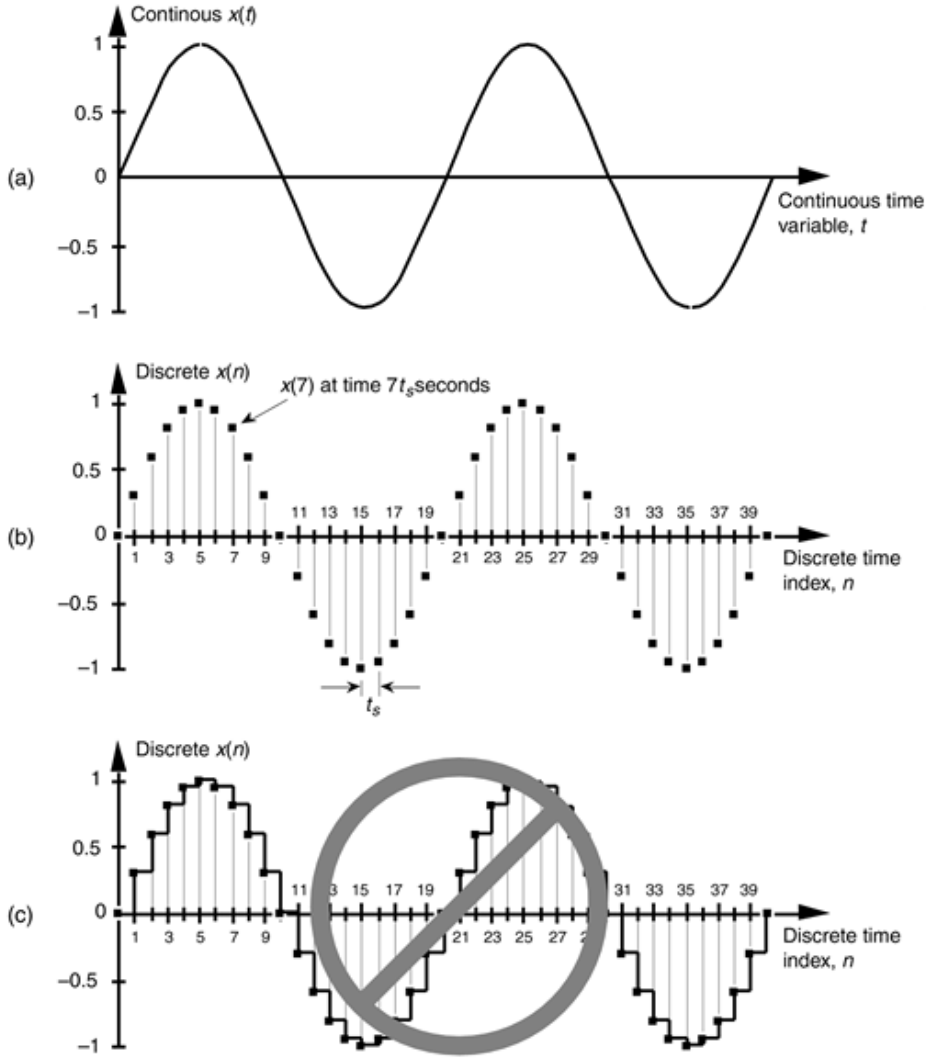
Γενικά, ο όρος επεξεργασία σήματος αναφέρεται στην επιστήμη ανάλυσης χρονικά μεταβαλλόμενων φυσικών διεργασιών. Η επεξεργασία σήματος χωρίζεται σε δύο κατηγορίες, στην αναλογική επεξεργασία και στην ψηφιακή. Ο όρος αναλογικός χρησιμοποιείται για να περιγράψει μια κυματομορφή συνεχή ως προς το χρόνο ενώ ο όρος ψηφιακό χρησιμοποιείται για την περιγραφή σημάτων που η ανεξάρτητη μεταβλητή του χρόνου είναι κβαντισμένη με αποτέλεσμα να γνωρίζουμε τις τιμές του σήματος μόνο σε διακριτές χρονικές τιμές. Επομένως, ένα διακριτό σήμα δεν απεικονίζεται με μια συνεχή κυματομορφή αλλά με μία ακολουθία τιμών. Εκτός από την κβάντιση του χρόνου, το διακριτό σήμα έχει κβαντισμένες και τις τιμές του πλάτους του. Αυτή η αρχή φαίνεται με το εξής παράδειγμα: Ας θεωρήσουμε ένα ημιτονικό σήμα με πλάτος 1 και συχνότητα f_0 που περιγράφεται από την εξίσωση

$$\boldsymbol{x(t) = \sin(2\pi f_0 t)} \qquad \textbf{Εξ. 2.1}$$

Η συχνότητα f_0 μετράται σε hertz (Hz) και με το χρόνο t να εκφράζεται σε seconds, ο όρος $2\pi f_0 t$ μετρά γωνία σε rad.

Στο Σχήμα 2.6 φαίνεται το αναλογικό σήμα καθώς και η κυματομορφή του μετά από δειγματοληψία ανά t_s seconds από ένα ADC, που μας δίνει μία ακολουθία τιμών. Λέμε ότι η κυματομορφή 2.6b είναι η αναπαράσταση διακριτού χρόνου του αναλογικού (συνεχούς στο χρόνο) σήματος 2.6a. Η ανεξάρτητη μεταβλητή t παίρνει συνεχείς τιμές ενώ η μεταβλητή δείκτη n είναι διακριτή και παίρνει μόνο ακέραιες τιμές. Για το λόγο αυτό κατά την απεικόνιση σε διακριτό χρόνο, δεν πρέπει να υπάρχει συνέχεια μεταξύ των κουκίδων,

καθώς οι τιμές του σήματος ορίζονται μόνο στα σημεία των κουκίδων και όχι μεταξύ αυτών.



Σχήμα 2.6 Δειγματοληψία ημιτονικού σήματος

Ενισχύοντας την αρχή που διέπει το διακριτό σήμα, γράφουμε ορισμένες τιμές δειγματοληψίας:

x(0) = 0 ($1^{η}$ τιμή της ακολουθίας. Δείκτης n = 0)

x(1) = 0,31 ($2^{η}$ τιμή της ακολουθίας. Δείκτης n = 1)

x(2) = 0,59 ($3^{η}$ τιμή της ακολουθίας. Δείκτης n = 2)

x(3) = 0.81 ($4^{η}$ τιμή της ακολουθίας. Δείκτης n = 3)

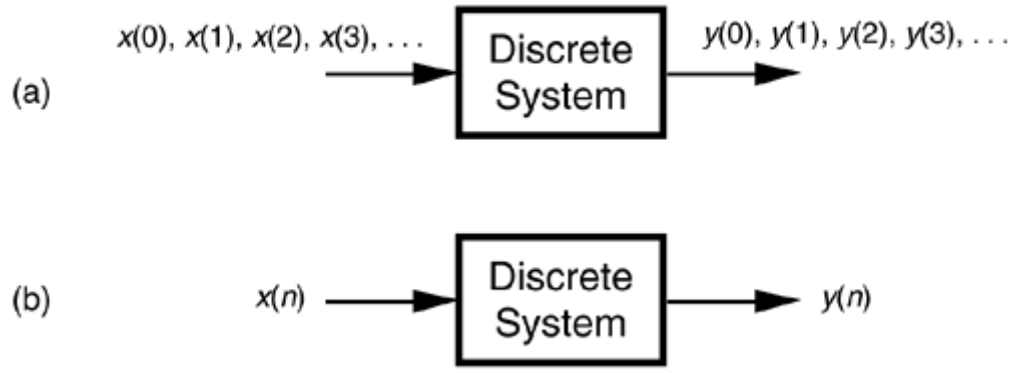
…… ……..

κ.λ.π. **Εξ. 2.2**

όπου το n εκφράζει το δείκτη της ακέραιας ακολουθίας του χρόνου 0,1,2,3,… και t_s είναι μια σταθερή χρονική περίοδος. Οι παραπάνω τιμές μπορούν να περιγραφούν με τη γενική μορφή της εξίσωσης

$$x(n) = \sin(2\pi f_0 n t_s) \quad \text{Εξ. 2.3}$$

Και οι δύο περιγραφές των εξισώσεων 2.1 και 2.3 εκφράζουν σήματα στο πεδίο του χρόνου (time domain signals), καθώς και στις δύο περιπτώσεις η ανεξάρτητη μεταβλητή (t και nt_s) παίρνει τιμές χρόνου.



Σχήμα 2.7 Είσοδος και έξοδος σε διακριτό σύστημα

Έχοντας υπ' όψη τα παραπάνω, υποθέτουμε ότι ένα διακριτό σύστημα είναι ένα σύνολο από υλικά στοιχεία ή ρουτίνες λογισμικού που εφαρμόζονται σε μία ακολουθία διακριτού χρόνου. Για παράδειγμα, ένα διακριτό σύστημα μπορεί να είναι μία διεργασία που μας δίνει στην έξοδο μία διακριτή ακολουθία y(0), y(1), y(2), κ.λ.π, όταν στην είσοδό της εφαρμοσθεί μία επίσης διακριτή ακολουθία x(0), x(1), x(2), κ.λ.π όπως φαίνεται στο Σχήμα 2.7a Οι συμβολισμοί x(n) και y(n) είναι γενικές μεταβλητές που εκφράζουν δύο ξεχωριστές ακολουθίες αριθμών. Το Σχήμα 2.7b μας επιτρέπει να περιγράψουμε την έξοδο ενός συστήματος με μία απλή έκφραση, όπως

$$\mathbf{y(n) = 2x(n) - 1} \quad \text{Εξ. 2.4}$$

Σύμφωνα με την εξίσωση 2.4, αν x(n) είναι η ακολουθία των 5 όρων: x(0) = 1, x(1) = 3, x(2) = 5, x(3) = 7, και x(4) = 9, τότε η y(n) είναι η ακολουθία επίσης 5 όρων y(0) = 1, y(1) = 5, y(2) = 9, y(3) = 13, and y(4) = 17.

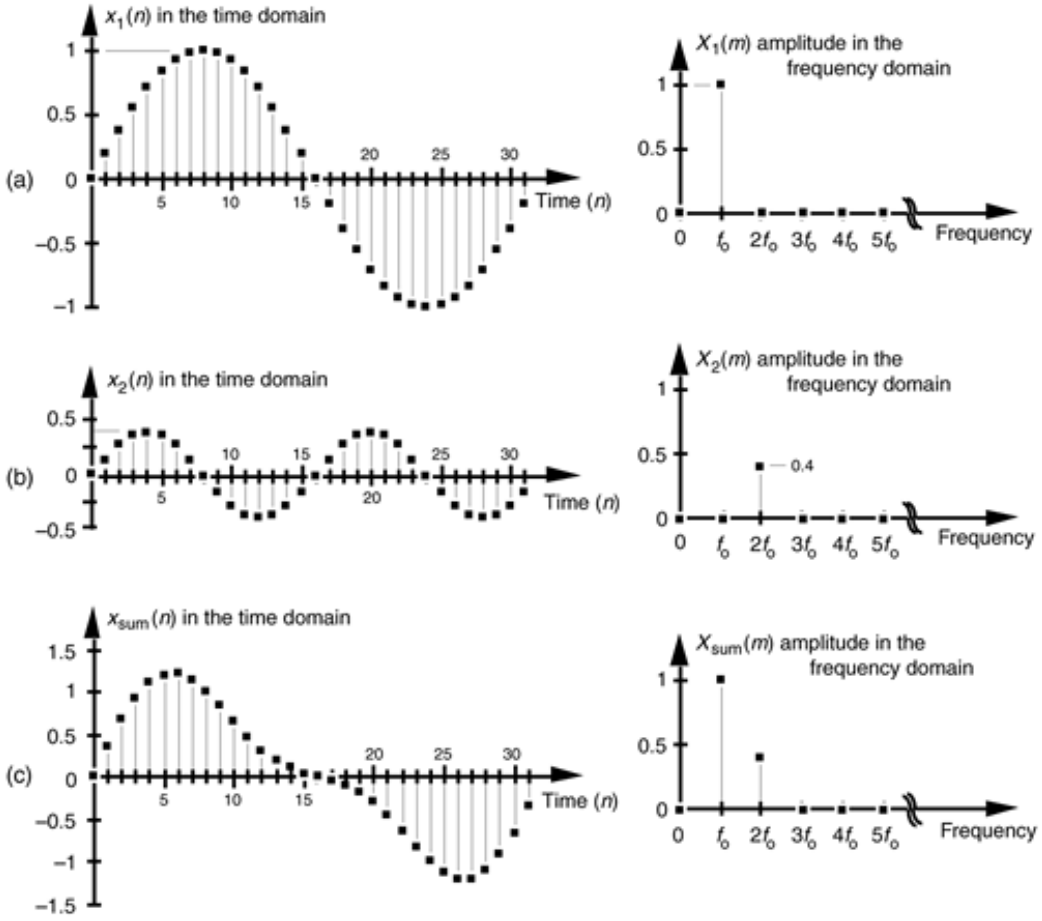
Η βασική διαφορά του τρόπου με τον οποίο απεικονίζεται ο χρόνος στα συνεχή και τα διακριτά συστήματα, οδηγεί σε μία σημαντική διαφορά για το πώς χαρακτηρίζεται η συχνότητα σε κάθε ένα από αυτά τα συστήματα. Για να φανεί αυτό, θεωρούμε τη συνεχή κυματομορφή του Σχήματος 2.6a. Αν αυτή δείχνει τάση, μπορούμε να μετρήσουμε τη συχνότητα με κάποιο όργανο (παλμογράφο, συχνόμετρο). Θα έχουμε όμως πρόβλημα εάν να μας έχουν δοθεί μόνο οι τιμές δειγματοληψίας της εξίσωσης 2.2 και μας ζητηθεί να προσδιορίσουμε τη συχνότητα της κυματομορφής που αυτές απεικονίζουν. Μπορούμε να πούμε ότι στο Σχήμα 2.6b φαίνεται ένα ημίτονο που επαναλαμβάνεται κάθε 20 δείγματα αλλά δεν μπορούμε να προσδιορίσουμε την ακριβή συχνότητά του μόνο από τις διακριτές

τιμές που μας έχουν δοθεί. Αν όμως γνωρίζαμε το χρόνο t_s μεταξύ των δειγμάτων θα μπορούσαμε να υπολογίσουμε την απόλυτη συχνότητα της διακριτής κυματομορφής. Για παράδειγμα, για t_s = 0,05ms, η περίοδος του ημιτόνου είναι:

$$\text{sinewave period} = \frac{20 \text{ samples}}{\text{period}} \cdot \frac{0.05 \text{ milliseconds}}{\text{sample}} = 1 \text{ millisecond.}$$

και επομένως η συχνότητα βρίσκεται να είναι 1Khz . Ομοίως, αν η περίοδος των δειγμάτων ήταν t_s = 2ms, η συχνότητα θα προέκυπτε 40Hz. Επομένως βλέπουμε ότι *ο προσδιορισμός της απόλυτης συχνότητας εξαρτάται από τη συχνότητα δειγματοληψίας* $f_s = 1/t_s$.

Στη ψηφιακή επεξεργασία σήματος, συχνά χρειάζεται να χαρακτηρίσουμε το φασματικό περιεχόμενο διακριτών σημάτων. Στην περίπτωση αυτή η απεικόνιση των συχνοτήτων υλοποιείται στο λεγόμενο *πεδίο συχνοτήτων*. Για παράδειγμα υποθέτουμε ότι έχουμε μια διακριτή ημιτονική ακολουθία $x_1(n)$ όπως φαίνεται στο Σχήμα 2.8a και που επισημαίνεται ότι έχει μια συχνότητα 1 που μετράται σε μονάδες της f_0 , και καμία άλλη συχνότητα. Παρατηρούμε επίσης ότι η απεικόνιση στο πεδίο συχνοτήτων είναι και αυτή σε διακριτή μορφή.



Σχήμα 2.8 Γραφική αναπαράσταση στα πεδία χρόνου και συχνότητας

Στο διάγραμμα 2.8b απεικονίζεται ένα διαφορετικό διακριτό σήμα $x_2(n)$, που έχει πλάτος 0,4 και περιέχει τη συχνότητα της $2f_0$. Οι διακριτές τιμές των δειγμάτων του $x_2(n)$ εκφράζονται από την εξίσωση:

$$\boldsymbol{x_2(n) = 0,4 * \sin(2\pi 2 f_0 n t_s)} \quad \textbf{Εξ. 2.5}$$

Όταν τα δύο ημίτονα $x_1(n)$ και $x_2(n)$ προστεθούν για να προκύψει μία νέα κυματομορφή $x_{sum}(n)$, η εξίσωση αυτής στο πεδίο του χρόνου είναι:

$$\boldsymbol{x_{sum}(n) = x_1(n) + x_2(n) = \sin(2\pi f_0 n t_s) + 0,4 * \sin(2\pi 2 f_0 n t_s)} \quad \textbf{Εξ. 2.6}$$

και οι αναπαραστάσεις αυτής στο πεδίο του χρόνου και της συχνότητας φαίνονται στο διάγραμμα 2.8c. Από το *φάσμα* της $x_{sum}(n)$ φαίνεται ότι αυτή περιέχει πλέον την f_0 καθώς και τη συνιστώσα $2f_0$ με μειωμένο πλάτος.

Σημειώνεται ότι για τον συμβολισμό των διακριτών ακολουθιών χρησιμοποιούμε πεζά γράμματα στο πεδίο του χρόνου και κεφαλαία στο πεδίο συχνότητας.

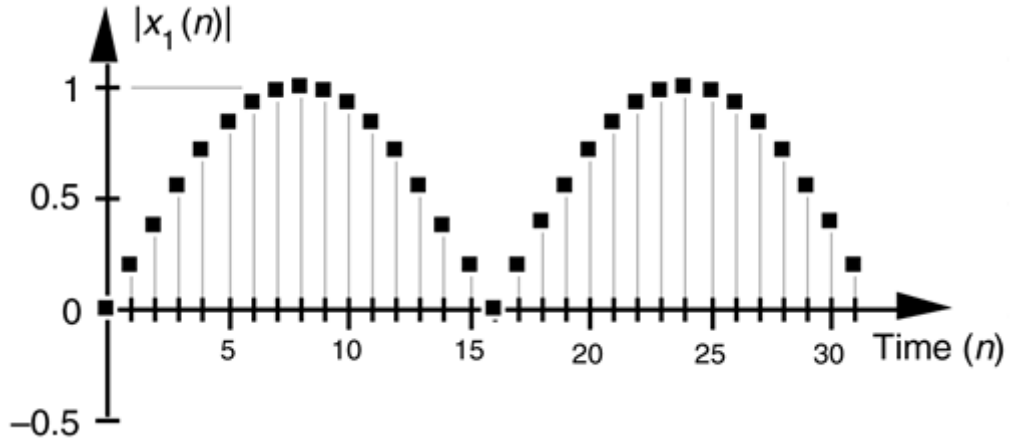
Όπως προκύπτει από τα παραπάνω, τα σήματα διακριτού χρόνου που εξετάζουμε δεν έχουν διακριτές τιμές μόνο ως προς το χρόνο αλλά και ως προς το πλάτος. Καθώς

απεικονίζουμε όλες τις ψηφιακές ποσότητες με δυαδικούς αριθμούς, υπάρχει ένα όριο στο resolution που διαθέτουμε κάθε φορά για την απεικόνιση αυτών των αριθμών.

2.3 Πλάτος (amplitude) Σήματος, Μέγεθος (magnitude), Ισχύς (power)

Δύο σημαντικοί όροι της ψηφιακής ανάλυσης σημάτων, που πολλές φορές μάλιστα χρησιμοποιούνται χωρίς σαφή διάκριση μεταξύ τους, είναι το πλάτος (amplitude) και το μέτρο (magnitude).

Το πλάτος μιας μεταβλητής είναι το μέτρο του πόσο μακριά και σε ποια κατεύθυνση η μεταβλητή αυτή απέχει από το μηδέν, δηλαδή το πλάτος του σήματος μπορεί να είναι θετικό ή αρνητικό. Η διακριτή χρονική ακολουθία του Σχήματος 2.8 παρουσίασε τις τιμές του πλάτους των δειγμάτων τριών διαφορετικών κυματομορφών, όπου μερικές τιμές είναι θετικές και άλλες αρνητικές.



Σχήμα 2.9 Τα μέτρα των δειγμάτων της κυματομορφής 2.8a

Το μέτρο (magnitude) μιας μεταβλητής είναι το μέτρο του πόσο μακριά, ανεξάρτητα από την κατεύθυνση, η τιμή αυτής απέχει από το μηδέν, δηλαδή το μέτρο είναι πάντοτε θετικό. Το Σχήμα 2.9 δείχνει ότι το magnitude της χρονικής ακολουθίας $x_1(n)$ του Σχήματος 2.8 είναι ίσο με το πλάτος αυτής, αλλά με πρόσημο πάντα θετικό.

Χρησιμοποιούμε το σύμβολο της απόλυτης τιμής για να συμβολίσουμε το magnitude της $x_1(n)$, γι' αυτό πολλές φορές στη βιβλιογραφία ο όρος magnitude αναφέρεται και ως απόλυτη τιμή (absolute value).

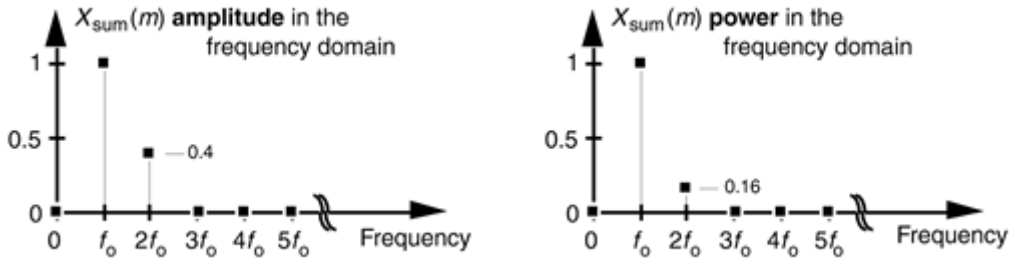
Πολλές φορές όταν εξετάζουμε σήματα στο πεδίο της συχνότητας μας ενδιαφέρει το επίπεδο της ισχύος αυτών. Η ισχύς ενός σήματος είναι ανάλογη του τετραγώνου του πλάτους (ή του magnitude). Αν υποθέσουμε ότι η σταθερά αναλογίας είναι 1, μπορούμε να εκφράσουμε την ισχύ μιας ακολουθίας στο πεδίο του χρόνου ή της συχνότητας ως

$$\boldsymbol{x_{pwr}(n) = x(n)^2 = |x(n)|^2} \qquad \textbf{Εξ. 2.7}$$

ή

$$\boldsymbol{X_{pwr}(m) = X(m)^2 = |X(m)|^2}$$ **Εξ. 2.8**

Συχνά χρειάζεται να γνωρίζουμε την διαφορά των επιπέδων ισχύος μεταξύ δύο σημάτων στο πεδίο της συχνότητας. Λόγω του τετραγώνου των παραπάνω σχέσεων, δύο σήματα με μικρή διαφορά στα πλάτη τους θα παρουσιάζουν πολύ μεγαλύτερη διαφορά στις αντίστοιχες τιμές των ισχύων τους (Σχήμα 2.10).

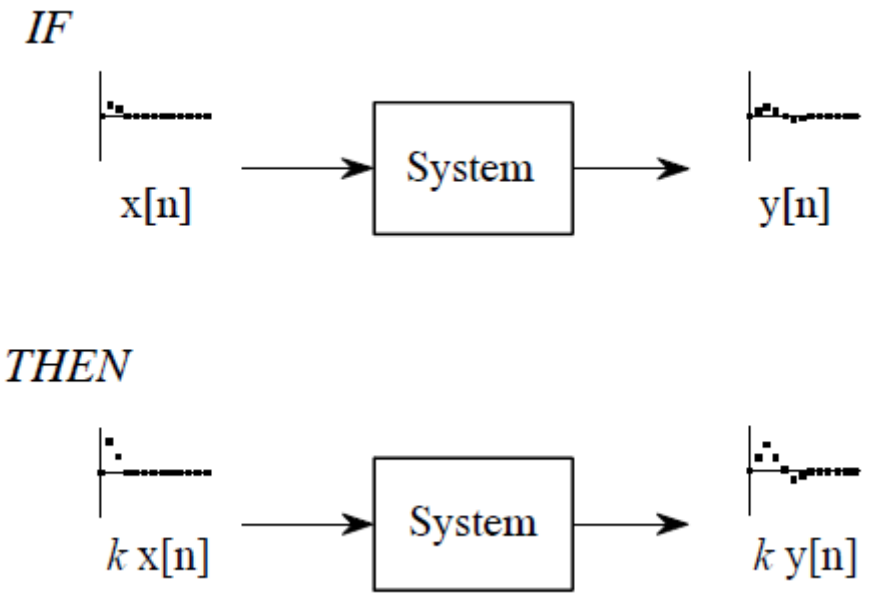


Σχήμα 2.10 Πλάτος και ισχύς της κυματομορφής 2.8c στο πεδίο της συχνότητας

Επειδή τα διαγράμματα ισχύος, όπως είναι αντιληπτό, μπορεί να χρειασθεί να παρουσιάζουν πολύ μεγάλες και πολύ μικρές τιμές ταυτόχρονα, προτιμάται η χρήση λογαριθμικής κλίμακας σε dB.

2.4 Διακριτά Γραμμικά Συστήματα

Ένα σύστημα καλείται *γραμμικό* εάν χαρακτηρίζεται από δύο μαθηματικές ιδιότητες: ομοιογένεια (**homogeneity**) και προσθετικότητα (**additivity**). Αν αποδειχθεί ότι το σήμα διαθέτει και τις δύο παραπάνω ιδιότητες, τότε αποδεικνύεται ότι είναι γραμμικό. Μία τρίτη ιδιότητα, η αμεταβλητότητα ως προς τη μετατόπιση (shift invariance) δεν αποτελεί αυστηρό περιορισμό για την ύπαρξη γραμμικότητας αλλά είναι απαιτούμενη για τις περισσότερες τεχνικές της ψηφιακής επεξεργασίας σημάτων.

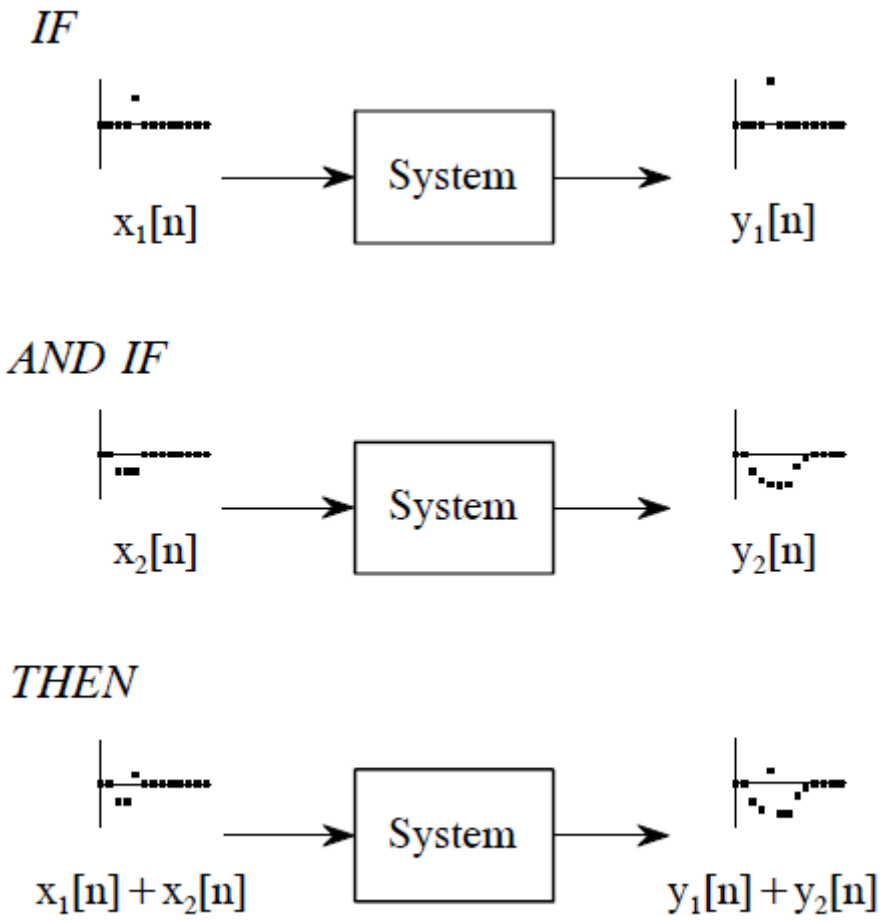


Σχήμα 2.11 Ορισμός της Homogeneity (Ομογένειας)

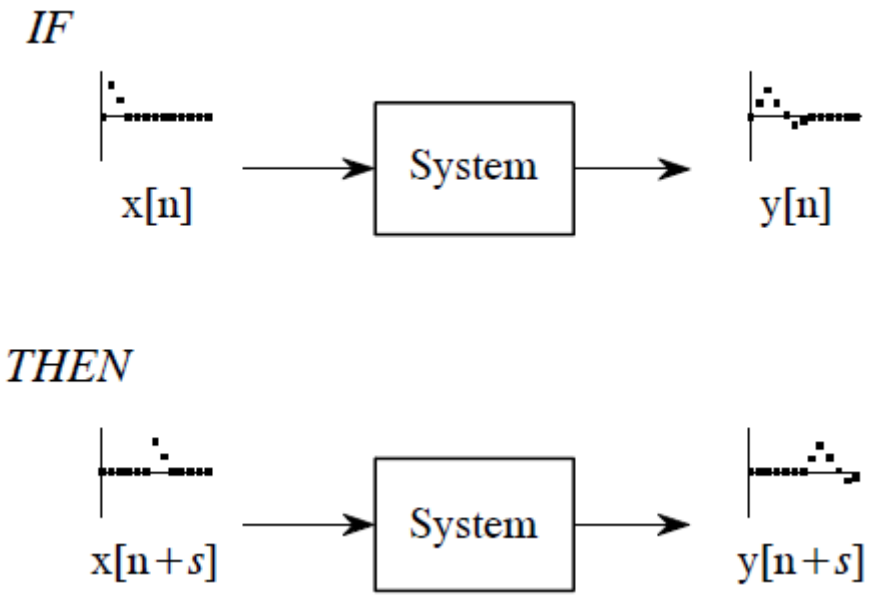
Η ομογένεια (Σχήμα 2.11) σημαίνει ότι οποιαδήποτε αλλαγή του πλάτους του σήματος εισόδου έχει ως αποτέλεσμα μια αντίστοιχη αλλαγή στο πλάτος του σήματος εξόδου. Με μαθηματική ορολογία, εάν ένα σήμα εισόδου x[n] δίνει στην έξοδο σήμα y[n], τότε αντίστοιχα ένα σήμα k x[n] θα δίνει έξοδο k y[n], για οποιοδήποτε σήμα εισόδου και σταθερά k.

Η ιδιότητα additivity φαίνεται στο Σχήμα 2.12. Θεωρούμε ένα σύστημα όπου μία είσοδος $x_1[n]$ παράγει έξοδο $y_1[n]$. Επιπρόσθετα θεωρούμε ότι μια άλλη είσοδος $x_2[n]$ παράγει έξοδο $y_2[n]$. Το σύστημα θα λέγεται additive εάν μία είσοδος $x_1[n] + x_2[n]$ δίνει έξοδο $y_1[n] + y_2[n]$, για κάθε δυνατό συνδυασμό εισόδων. Με άλλα λόγια, σήματα που προστίθενται στην είσοδο δίνουν στην έξοδο την πρόσθεση των αντίστοιχων σημάτων εξόδου.

Το σημαντικό σημείο της παραπάνω ανάλυσης είναι ότι τα σήματα που προστίθενται περνούν από το σύστημα χωρίς να αλληλεπιδρούν μεταξύ τους.



Σχήμα 2.12 Ιδιότητα Additivity (Προσθετικότητα)



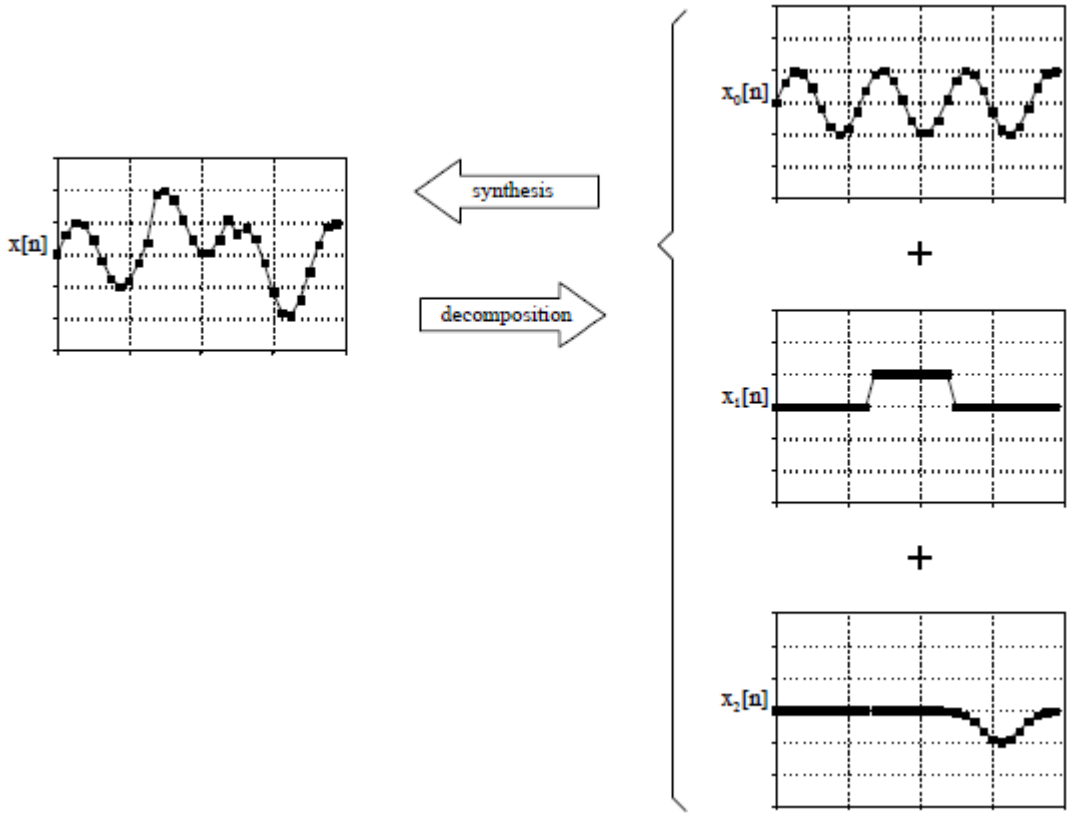
Σχήμα 2.13 Ιδιότητα Shift Invariance

Η ιδιότητα shift invariance σημαίνει ότι οποιαδήποτε ολίσθηση του σήματος εισόδου επιφέρει όμοια ολίσθηση στο σήμα εξόδου, ή διαφορετικά, αν ένα σήμα x[n] δίνει έξοδο y

[n], τότε αντίστοιχα το σήμα x[n + s] θα δώσει έξοδο y[n + s], για κάθε σήμα εισόδου και κάθε σταθερά s. Πρέπει να δώσουμε ιδιαίτερη προσοχή στη μαθηματική έννοια της ολίσθησης: προσθέτοντας μία σταθερά s στην ανεξάρτητη μεταβλητή n, η κυματομορφή μετατοπίζεται μπροστά ή πίσω κατά την οριζόντια κατεύθυνση. Για παράδειγμα, αν s = 2, το σήμα ολισθαίνει προς τα αριστερά κατά δύο δείγματα, ενώ αν s = -2, ολισθαίνει προς τα δεξιά κατά δύο δείγματα. Η ιδιότητα shift invariance είναι σημαντική διότι σημαίνει ότι τα χαρακτηριστικά του συστήματος δεν μεταβάλλονται με το χρόνο (ή οποιαδήποτε είναι η ανεξάρτητη μεταβλητή) – χρονικά αμετάβλητα συστήματα .

2.5 Superposition (Υπέρθεση ή Επαλληλία): το θεμέλιο του DSP

Όταν διαχειριζόμαστε γραμμικά συστήματα, ο μόνος τρόπος που μπορούν διάφορα σήματα να συνδυαστούν μεταξύ τους είναι με κλιμάκωση - **scaling** (πολλαπλασιασμός των σημάτων με κάποια σταθερά) και ακολούθως πρόσθεση (**addition**) αυτών. Το Σχήμα 2.14 δείχνει τον τρόπο που τα τρία σήματα $x_0[n], x_1[n] και\, x_2[n]$ προστίθενται για να προκύψει το x[n]. Η διαδικασία του συνδυασμού σημάτων μέσω scaling και addition ονομάζεται **σύνθεση** (**synthesis**).

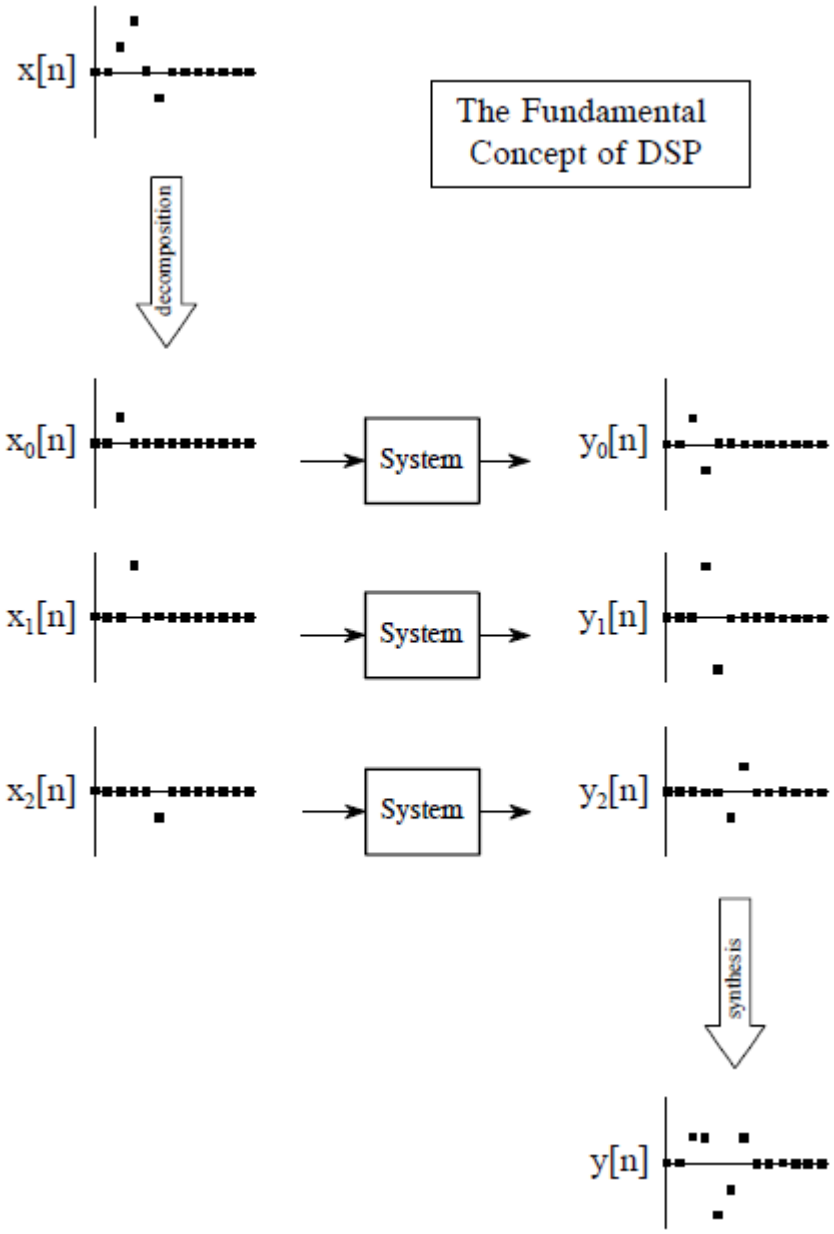


Σχήμα 2.14 Σύνθεση και ανάλυση σήματος

Ανάλυση (**decomposition**) είναι η αντίστροφη διαδικασία της σύνθεσης, όπου ένα σήμα αναλύεται σε δύο ή περισσότερες συνιστώσες. Αυτή η διαδικασία είναι πιο περίπλοκη από τη σύνθεση καθώς υπάρχουν άπειροι δυνατοί συνδυασμοί ανάλυσης ενός σήματος. Για

παράδειγμα, οι αριθμοί 15 και 25 μπορούν να συνθέσουν (προστιθέμενοι) μόνο τον αριθμό 40, ενώ αντίθετα, ο αριθμός 40 μπορεί να αναλυθεί σε 1+39 ή 2+38 ή -30.5+60+10.5 κ.λ.π

Ερχόμαστε τώρα στη θεμελιώδη αρχή του DSP: την υπέρθεση ή επαλληλία (**superposition**). Θεωρούμε ένα σήμα εισόδου x[n], το οποίο περνά μέσα από ένα γραμμικό σύστημα και δίνει στην έξοδο ένα σήμα y[n].

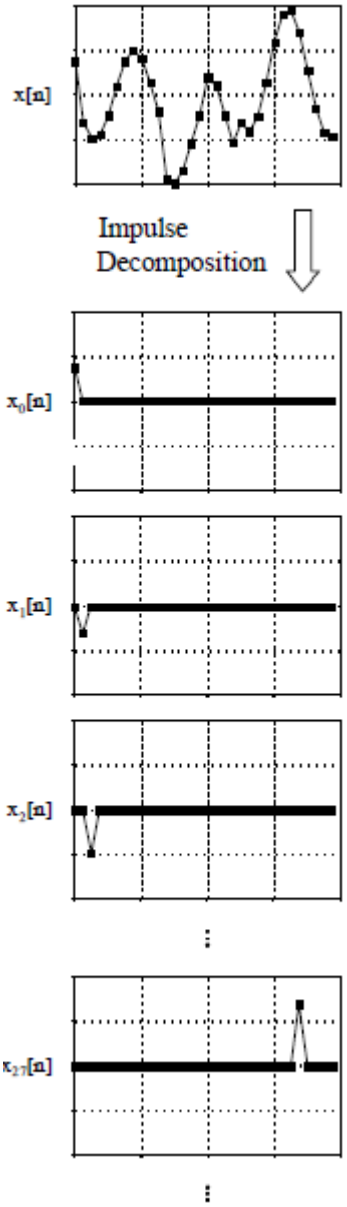


Σχήμα 2.15 Θεμελιώδης Αρχή του DSP

Όπως φαίνεται στο Σχήμα 2.15, το σήμα εισόδου μπορεί να αναλυθεί σε μία ομάδα πιο απλών σημάτων: $x_0[n], x_1[n], x_2[n]$, κ.λ.π. που ονομάζονται συνιστώσες του σήματος εισόδου. Στη συνέχεια, κάθε μια από τις συνιστώσες αυτές περνά από το ίδιο γραμμικό σύστημα και δίνει στην έξοδο τις συνιστώσες του σήματος εξόδου $y_0[n]$, $y_1[n]$, $y_2[n]$ κ.λ.π. που αν συντεθούν μεταξύ τους, προκύπτει το σήμα εξόδου y[n]. Εδώ βρίσκεται το σημαντικό σημείο: το σήμα εξόδου που προκύπτει με τον τρόπο αυτό, είναι *όμοιο* με αυτό

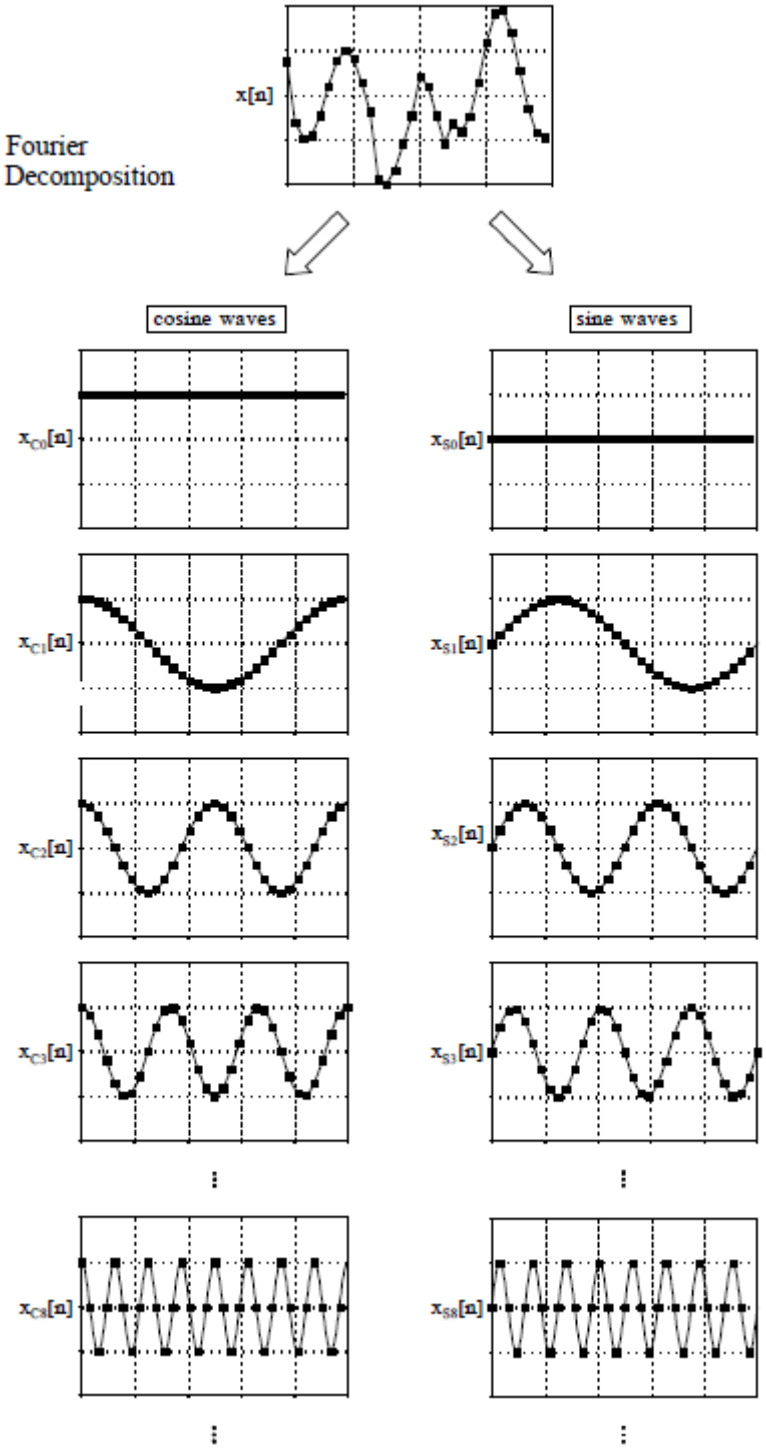
που προέκυψε όταν περάσαμε από το σύστημα απ' ευθείας το αρχικό σήμα εισόδου. Αντί λοιπόν να προσπαθούμε να καταλάβουμε πως ένα σύστημα μεταβάλλει *περίπλοκα* σήματα, το μόνο που χρειάζεται είναι να γνωρίζουμε πως μεταβάλλονται από αυτό πολύ πιο *απλά* σήματα. Στη γλώσσα του DSP, τα σήματα εισόδου και εξόδου θεωρούνται ως η υπέρθεση ή επαλληλία (άθροισμα) απλούστερων κυματομορφών. Αυτή είναι η βασική αρχή σχεδόν όλων των τεχνικών της Ψηφιακής Επεξεργασίας Σήματος.

Υπάρχουν δύο βασικοί τρόποι ανάλυσης (**decomposition**) σημάτων στην ψηφιακή επεξεργασία: ανάλυση σε κρουστικούς παλμούς (**impulse decomposition**) και ανάλυση κατά Fourier (**Fourier decomposition**). Ο πρώτος τρόπος αναλύει τα N ληφθέντα δείγματα ενός σήματος σε N συνιστώσες, που η κάθε μία περιέχει επίσης N δείγματα. Επίσης, κάθε συνιστώσα περιέχει ένα μη μηδενικό δείγμα του αρχικού σήματος έχοντας όλα τα υπόλοιπα δείγματα μηδενικά. Αυτό φαίνεται στο Σχήμα 2.16. Η τεχνική του **convolution** είναι ένα παράδειγμα ανάλυσης σήματος με τη βοήθεια κρουστικών παλμών.



Σχήμα 2.16 Impulse Decomposition

Ο δεύτερος τρόπος της ανάλυσης κατά Fourier είναι μια κατά βάση μαθηματική (και καθόλου προφανής) ανάλυση. Η ανάλυση αυτή επιτρέπει το διαχωρισμό ενός σήματος N δειγμάτων σε N+2 συνιστώσες (επίσης N δειγμάτων η κάθε μία), εκ των οποίων οι μισές είναι κυματομορφές ημιτόνου και οι άλλες μισές συνημιτόνου, όπως φαίνεται στο Σχήμα 2.17. Το συνημίτονο της μικρότερης συχνότητας που αναφέρεται ως $x_{C0}[n]$ διαγράφει μηδέν κύκλους κατά μήκος των N δειγμάτων, δηλαδή είναι ένα DC σήμα. Οι επόμενες συνιστώσες συνημιτόνου $x_{C1}[n], x_{C2}[n], και\ x_{C3}[n]$ διαγράφουν αντίστοιχα 1, 2 και 3 πλήρεις κύκλους κατά τη διάρκεια των N δειγμάτων και αυτό ισχύει για όλες τις συνιστώσες συνιμητόνου και ημιτόνου. Καθώς λοιπόν η συχνότητα κάθε συνιστώσας είναι σταθερή το μόνο που αλλάζει σε διάφορα σήματα που αναλύονται είναι το πλάτος κάθε συνιστώσας. Ο Διακριτός Μετασχηματισμός Fourier (**Discrete Fourier Transform**) που θα εξετασθεί στο παρόν κείμενο ανήκει σε αυτό το είδος της τεχνικής.



Σχήμα 2.17 Fourier Decomposition

2.6 Discrete Fourier Transform – DFT (Διακριτός Μετασχηματισμός Fourier)

Ο Διακριτός Μετασχηματισμός Fourier (**DFT**) είναι μία από τις δύο πιο γνωστές και ισχυρές διαδικασίες που συναντάμε στο πεδίο της Ψηφιακής Επεξεργασίας Σήματος (η άλλη είναι τα Ψηφιακά Φίλτρα). Ο DFT μας επιτρέπει να αναλύουμε, να διαχειριζόμαστε

και να συνθέτουμε σήματα με τέτοιο τρόπο, που δεν είναι εφικτό να γίνει με την συνεχή (αναλογική) επεξεργασία σήματος. Είναι μια μαθηματική διαδικασία που χρησιμοποιείται για να προσδιορισθεί το φασματικό περιεχόμενο μιας διακριτής ακολουθίας η οποία αναπαριστά ένα σήμα. Γενικά είναι χρήσιμη για την ανάλυση οποιασδήποτε διακριτής ακολουθίας, ανεξάρτητα από το τι ακριβώς αναπαριστά αυτή.

Η προέλευση του DFT είναι φυσικά ο **συνεχής μετασχηματισμός Fourier**:

$$\boldsymbol{X(f) = \int_{-\infty}^{+\infty} x(t) e^{-j2\pi f t} dt} \qquad \textbf{Εξ. 2.9}$$

όπου x(t) είναι κάποιο σήμα συνεχές στο πεδίο του χρόνου. Για την επεξεργασία αυτών των σημάτων, η παραπάνω εξίσωση χρησιμοποιείται για να μετασχηματίσει μια συνεχή συνάρτηση του χρόνου x(t) σε μία άλλη συνεχή συνάρτηση της συχνότητας X(f). Παραπέρα μελέτη της X(f) μας επιτρέπει να προσδιορίσουμε το φασματικό περιεχόμενο πρακτικά οποιουδήποτε σήματος μας ενδιαφέρει.

Με την έλευση των ψηφιακών υπολογιστών αναπτύχθηκε ο διακριτός Μετασχηματισμός Fourier που ορίζεται ως η διακριτή ακολουθία συχνοτήτων X(m), όπου:

$$\boldsymbol{X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}} \qquad \textbf{Εξ. 2.10}$$

Η συνάρτηση x(n) είναι μια διακριτή ακολουθία τιμών δειγματοληψίας μιας συνεχούς ως προς το χρόνο συνάρτησης x(t).

2.6.1 Κατανοώντας την εξίσωση DFT

Η εξίσωση 2.10 μπορεί να γραφεί σε μια πιο φιλική μορφή εφαρμόζοντας σε αυτή τη γνωστή ιδιότητα του Euler $e^{-j\theta} = \cos(\theta) - jsin(\theta)$:

$$\boldsymbol{X(m) = \sum_{n=0}^{N-1} x(n)[\cos\left(\frac{2\pi nm}{N}\right) - jsin\left(\frac{2\pi nm}{N}\right)]} \quad \textbf{Εξ. 2.11}$$

διαχωρίζοντας έτσι την εκθετική μορφή του μιγαδικού της εξ.2.10 στο πραγματικό και φανταστικό μέρος του, όπου:

X(m) = η m-ιοστή συνιστώσα εξόδου του DFT, για παράδειγμα X(0), X(1), X(2) κ.α.

m = ο δείκτης της ακολουθίας εξόδου του DFT στο πεδίο του χρόνου, όπου παίρνει τις τιμές 0,1,2,3,….N-1

x(n) = η ακολουθία των δειγμάτων εισόδου, x(0), x(1), x(2),….κ.α

n = ο δείκτης της ακολουθίας των δειγμάτων στο πεδίο του χρόνου (είσοδος), που παίρνει τιμές 0,1,2,3,…N-1

j = $\sqrt{-1}$

N = ο αριθμός των δειγμάτων της ακολουθίας εισόδου καθώς και ο αριθμός των σημείων της συχνότητας στην έξοδο του DFT .

Οι δείκτες των δειγμάτων εισόδου n και εξόδου m παίρνουν τιμές από 0 έως N-1 που σημαίνει ότι με N δείγματα εισόδου στο πεδίο του χρόνου, ο DFT εξάγει το φασματικό περιεχόμενο της εισόδου σε N ισαπέχοντα διαστήματα στο πεδίο της συχνότητας. Ο αριθμός N είναι μια σημαντική παράμετρος καθώς δείχνει πόσα δείγματα εισόδου απαιτούνται, τη διακριτικότητα (resolution) του αποτελέσματος στο πεδίο της συχνότητας καθώς και το χρόνο που απαιτείται για να γίνει η επεξεργασία των N δειγμάτων.

Ως παράδειγμα, γράφουμε την εξ.2.11 για N=4:

$$\boldsymbol{X(m) = \sum_{n=0}^{3} x(n)[\cos\left(\frac{2\pi nm}{4}\right) - jsin\left(\frac{2\pi nm}{4}\right)]} \quad \textbf{Εξ. 2.12}$$

και την αναλύουμε σε όλους τους όρους της. Για m=0 έχουμε:

$$\begin{aligned} X(0) = {} & x(0)\cos(2\pi\cdot 0\cdot 0/4) - jx(0)\sin(2\pi\cdot 0\cdot 0/4) \\ & + x(1)\cos(2\pi\cdot 1\cdot 0/4) - jx(1)\sin(2\pi\cdot 1\cdot 0/4) \\ & + x(2)\cos(2\pi\cdot 2\cdot 0/4) - jx(2)\sin(2\pi\cdot 2\cdot 0/4) \\ & + x(3)\cos(2\pi\cdot 3\cdot 0/4) - jx(3)\sin(2\pi\cdot 3\cdot 0/4). \end{aligned}$$

Για τον δεύτερο όρο της που αντιστοιχεί σε m=1, η εξ.2.12 γίνεται:

$$\begin{aligned} X(1) = {} & x(0)\cos(2\pi\cdot 0\cdot 1/4) - jx(0)\sin(2\pi\cdot 0\cdot 1/4) \\ & + x(1)\cos(2\pi\cdot 1\cdot 1/4) - jx(1)\sin(2\pi\cdot 1\cdot 1/4) \\ & + x(2)\cos(2\pi\cdot 2\cdot 1/4) - jx(2)\sin(2\pi\cdot 2\cdot 1/4) \\ & + x(3)\cos(2\pi\cdot 3\cdot 1/4) - jx(3)\sin(2\pi\cdot 3\cdot 1/4). \end{aligned}$$

Για m=2 δίνει:

$$\begin{aligned} X(2) = {} & x(0)\cos(2\pi\cdot 0\cdot 2/4) - jx(0)\sin(2\pi\cdot 0\cdot 2/4) \\ & + x(1)\cos(2\pi\cdot 1\cdot 2/4) - jx(1)\sin(2\pi\cdot 1\cdot 2/4) \\ & + x(2)\cos(2\pi\cdot 2\cdot 2/4) - jx(2)\sin(2\pi\cdot 2\cdot 2/4) \\ & + x(3)\cos(2\pi\cdot 3\cdot 2/4) - jx(3)\sin(2\pi\cdot 3\cdot 2/4). \end{aligned}$$

Τέλος για m= 3 δίνει:

$$\begin{aligned} X(3) = {} & x(0)\cos(2\pi\cdot 0\cdot 3/4) - jx(0)\sin(2\pi\cdot 0\cdot 3/4) \\ & + x(1)\cos(2\pi\cdot 1\cdot 3/4) - jx(1)\sin(2\pi\cdot 1\cdot 3/4) \\ & + x(2)\cos(2\pi\cdot 2\cdot 3/4) - jx(2)\sin(2\pi\cdot 2\cdot 3/4) \\ & + x(3)\cos(2\pi\cdot 3\cdot 3/4) - jx(3)\sin(2\pi\cdot 3\cdot 3/4). \end{aligned}$$

Όπως βλέπουμε, κάθε όρος X(m) της εξόδου του DFT προκύπτει από το συνολικό άθροισμα των γινομένων σημείο προς σημείο μεταξύ μιάς ακολουθίας τιμών ενός σήματος (δείγματα εισόδου) και ενός μιγαδικού της μορφής $\cos(\theta) - jsin(\theta)$. Οι ακριβείς συχνότητες των διαφόρων ημιτονικών όρων εξαρτώνται τόσο από τη συχνότητα δειγματοληψίας f_s, με την οποία δειγματοληπτείται το αρχικό σήμα, όσο και και από τον αριθμό N των δειγμάτων. Για παράδειγμα, αν κάνουμε δειγματοληψία ενός σήματος

(συνεχούς ως προς το χρόνο) με ρυθμό 500 δείγματα/δευτερόλεπτο και στη συνέχεια εφαρμόσουμε στα ληφθέντα δεδομένα DFT 16 σημείων, η βασική συχνότητα των ημιτονικών όρων είναι $^{f_s}/_{N} = {}^{500}/_{16} = 31.25Hz$. Οι υπόλοιποι όροι συχνοτήτων X(m) της ανάλυσης είναι ακέραια πολλαπλάσια της βασικής συχνότητας, π.χ:

X(0) = 1ος όρος συχνότητας, με συχνότητα $0 \cdot 31.25 = 0\ Hz$

X(1) = 2ος όρος συχνότητας, με συχνότητα $1 \cdot 31.25 = 31.25\ Hz$

X(2) = 3ος όρος συχνότητας, με συχνότητα $2 \cdot 31.25 = 62.5\ Hz$

X(3) = 4ος όρος συχνότητας, με συχνότητα $3 \cdot 31.25 = 93.75\ Hz$

……

……

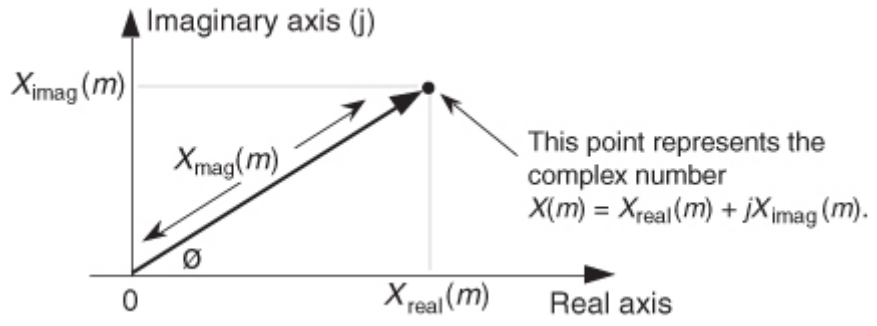
X(15) = 16ος όρος συχνότητας, με συχνότητα $15 \cdot 31.25 = 468.75\ Hz$

Οι N διαφορετικοί όροι συχνοτήτων προκύπτουν από την εξίσωση:

$$\boldsymbol{f(m) = \frac{m f_s}{N}} \quad \textbf{Εξ. 2.13}$$

Στο παραπάνω παράδειγμα, ο όρος X(0) μας δείχνει το μέτρο (πλάτος) της συνιστώσας συχνότητας 0 Hz (DC), ο όρος X(1) το μέτρο της συνιστώσας των 31.25Hz κ.λ.π. Επίσης, όπως θα δούμε, αυτοί οι όροι συχνοτήτων δείχνουν και τη σχετική φάση μεταξύ των συνιστωσών των συχνοτήτων που περιέχονται στο σήμα εισόδου.

Συχνά μας ενδιαφέρει να γνωρίζουμε τόσο το μέτρο όσο και την ισχύ (το μέτρο στο τετράγωνο) των όρων συχνοτήτων, οπότε εφαρμόζεται ο ορισμός του μέτρου μιγαδικού αριθμού, όπως φαίνεται στο Σχήμα 2.18.



Σχήμα 2.18 Τριγωνομετρικές σχέσεις ενός μιγαδικού όρου X(m)

Αν εκφράσουμε κάθε όρο X(m) με το πραγματικό και φανταστικό μέρος του,

$$\boldsymbol{X(m) = X_{real}(m) + jX_{imag}(m) = X_{mag}(m)\ με\ ορισμα\ X_{\theta}(m)} \quad \textbf{Εξ. 2.14}$$

το μέτρο του X(m) θα είναι:

$$\boldsymbol{X_{mag}(m) = |X_m| = \sqrt{X_{real}(m)^2 + X_{imag}(m)^2}} \quad \textbf{Εξ. 2.15}$$

Το όρισμα είναι:

$$\boldsymbol{X_{\theta}(m) = tan^{-1}\left(\frac{X_{imag}(m)}{X_{real}(m)}\right)} \qquad \textbf{Εξ. 2.16}$$

Η ισχύς κάθε όρου X(m), που θα μας δώσει το φάσμα ισχύος, είναι:

$$\boldsymbol{X_{PS}(m) = X_{mag}(m)^2 = X_{real}(m)^2 + X_{imag}(m)^2} \textbf{ Εξ. 2.17}$$

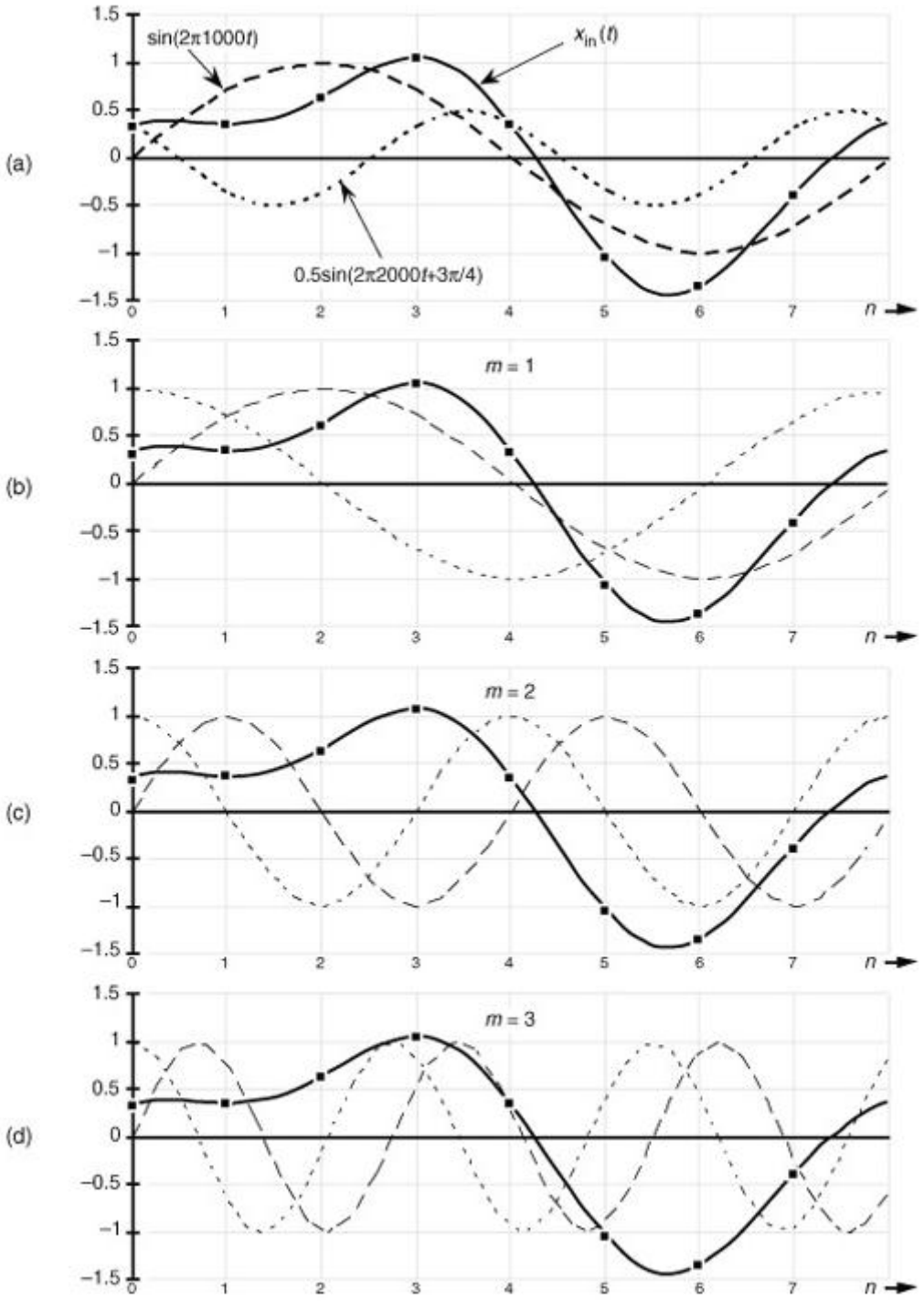
2.6.2 DFT Παράδειγμα

Υποθέτουμε ότι θέλουμε να πάρουμε δείγματα και να εφαρμόσουμε DFT 8 σημείων σε ένα συνεχές σήμα που περιέχει συνιστώσες συχνοτήτων του 1kHz και 2 kHz, το οποίο εκφράζεται ως:

$$\boldsymbol{x_{in}(t) = \sin(2\pi \cdot 1000 \cdot t) + 0.5\sin(2\pi \cdot 2000 \cdot t + \frac{3\pi}{4})} \quad \textbf{Εξ. 2.18}$$

όπου φαίνεται ότι η συνιστώσα των 2kHz έχει διαφορά φάσης 135^o σε σχέση με αυτή του 1kHz. Με ρυθμό δειγματοληψίας f_s , παίρνουμε δείγματα της εισόδου κάθε $1/f_s$ seconds. Αφού N=8, χρειαζόμαστε 8 δείγματα εισόδου στα οποία θα εφαρμόσουμε DFT. Έτσι, η ακολουθία x(n) των 8 στοιχείων ισούται με $x_{in}(t)$ που δειγματοληπτείται στις χρονικές στιγμές nt_s, έτσι ώστε

$$\boldsymbol{x(n) = x_{in}(nt_s) = \sin(2\pi \cdot 1000 \cdot nt_s) + 0.5\sin(2\pi \cdot 2000 \cdot nt_s + \frac{3\pi}{4})} \quad \textbf{Εξ. 2.19}$$



Σχήμα 2.19 Παράδειγμα του DFT

Αν επιλέξουμε να λάβουμε δείγματα με ρυθμό 8000samples/sec, τα αποτελέσματα της ανάλυσης DFT δείχνουν ότι θα έχουμε μη μηδενικά πλάτη της x(n) στις συχνότητες που καθορίζονται από τη σχέση f(m) = mf_s/N, δηλαδή στις 0kHz, 1kHz, 2kHz,,…,7kHz. Με f_s = 8000 samples/sec, τα 8 δείγματα της x(n) είναι

$$x(0) = 0.3535, \quad x(1) = 0.3535,$$
$$x(2) = 0.6464, \quad x(3) = 1.0607,$$
$$x(4) = 0.3535, \quad x(5) = -1.0607,$$
$$x(6) = -1.3535, \quad x(7) = -0.3535.$$

Εξ. 2.20

Τα δείγματα αυτά σημειώνονται με τις τετράγωνες κουκίδες στη συμπαγή συνεχή κυματομορφή του Σχήματος 2.19, ενώ οι διακεκομμένες γραμμές δείχνουν τις δύο συνιστώσες του σήματος.

Εφαρμόζουμε τώρα την εξ.2.11 για να προσδιορίσουμε το DFT της εισόδου x(n), αρχίζοντας με m=1, καθώς για m=0 έχουμε μια ειδική περίπτωση που θα δούμε ξεχωριστά. Έτσι, για m=1 ή για τον όρο DFT που αντιστοιχεί σε 1kHz (mf_s/N = 1*8000/8), η εξ.2.11 γίνεται:

$$\boldsymbol{X(1) = \sum_{n=0}^{7} x(n)[\cos\left(\frac{2\pi n}{8}\right) - j\sin\left(\frac{2\pi n}{8}\right)]} \qquad \textbf{Εξ. 2.21}$$

Αντικαθιστώντας στην παραπάνω τους όρους της x(n) από την εξ.2.20, προκύπτει:

$$\begin{aligned}
X(1) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) && \leftarrow \text{this is the } n = 0 \text{ term} \\
& + 0.3535 \cdot 0.707 && - j(0.3535 \cdot 0.707) && \leftarrow \text{this is the } n = 1 \text{ term} \\
& + 0.6464 \cdot 0.0 && - j(0.6464 \cdot 1.0) && \leftarrow \text{this is the } n = 2 \text{ term} \\
& + 1.0607 \cdot -0.707 && - j(1.0607 \cdot 0.707) && \quad \ldots \\
& + 0.3535 \cdot -1.0 && - j(0.3535 \cdot 0.0) && \quad \ldots \\
& - 1.0607 \cdot -0.707 && - j(-1.0607 \cdot -0.707) && \quad \ldots \\
& - 1.3535 \cdot 0.0 && - j(-1.3535 \cdot -1.0) && \quad \ldots \\
& - 0.3535 \cdot 0.707 && - j(-0.3535 \cdot -0.707) && \leftarrow \text{this is the } n = 7 \text{ term}
\end{aligned}$$

$$\begin{aligned}
= {} & 0.3535 && + j0.0 \\
& + 0.250 && - j0.250 \\
& + 0.0 && - j0.6464 \\
& - 0.750 && - j0.750 \\
& - 0.3535 && - j0.0 \\
& + 0.750 && - j0.750 \\
& + 0.0 && - j1.3535 \\
& - 0.250 && - j0.250
\end{aligned}$$

$$= 0.0 - j4.0 = 4 \angle -90°.$$

Έτσι βλέπουμε ότι η είσοδος x(n) περιέχει μη μηδενική συνιστώσα συχνότητας 1kHz. Επίσης βλέπουμε ότι η X(1) έχει μέτρο $X_{mag}(1) = 4$, $X_{PS}(1) = 16$ και διαφορά φάσης $X_\theta(1) = -90^o$ σε σχέση με το συνημίτονο του 1 kHz (Σχήμα 2.19(b)).

Ομοίως για m=2 (που αντιστοιχεί στη συνιστώσα των 2 kHz, Σχήμα 2.19(c)), έχουμε:

$$
\begin{aligned}
X(2) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot 0.0 && - j(0.3535 \cdot 1.0) \\
& + 0.6464 \cdot -1.0 && - j(0.6464 \cdot 0.0) \\
& + 1.0607 \cdot 0.0 && - j(1.0607 \cdot -1.0) \\
& + 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& -1.0607 \cdot 0.0 && - j(-1.0607 \cdot 1.0) \\
& -1.3535 \cdot -1.0 && - j(-1.3535 \cdot 0.0) \\
& -0.3535 \cdot 0.0 && - j(-0.3535 \cdot -1.0)
\end{aligned}
$$

$$
\begin{aligned}
= {} & 0.3535 && + j0.0 \\
& + 0.0 && - j0.3535 \\
& - 0.6464 && - j0.0 \\
& - 0.0 && + j1.0607 \\
& + 0.3535 && - j0.0 \\
& + 0.0 && + j1.0607 \\
& + 1.3535 && - j0.0 \\
& - 0.0 && - j0.3535
\end{aligned}
$$

$$= 1.414 + j1.414 = 2 \angle 45°.$$

όπου φαίνεται ότι επίσης έχουμε μη μηδενική συνιστώσα, με σχετικό πλάτος 2 και διαφορά φάσης 45^o σε σχέση με το συνημίτονο των 2 kHz.

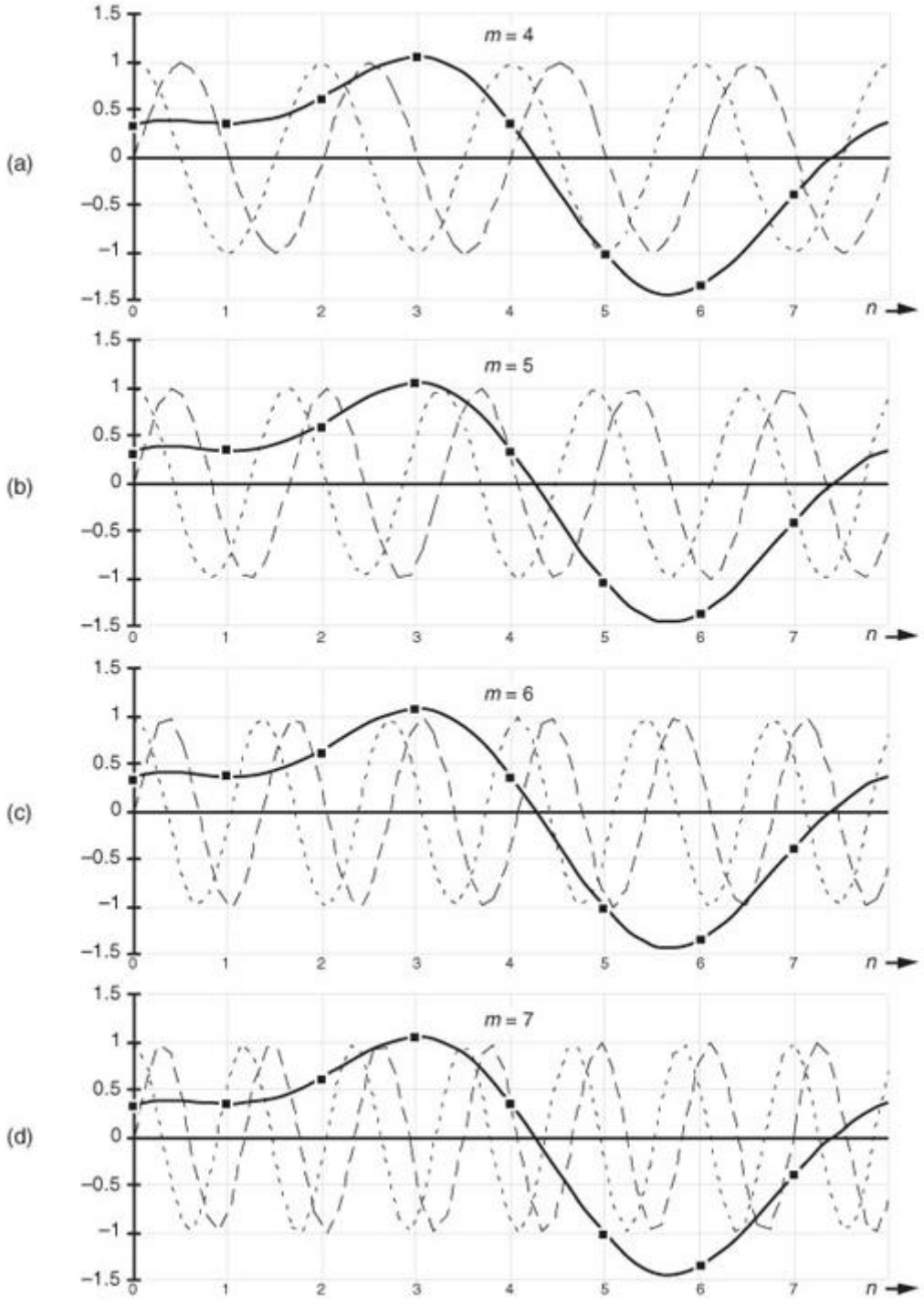
Για τη συνιστώσα των 3kHz όπου m=3 έχουμε:

$$
\begin{aligned}
X(3) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot -0.707 && - j(0.3535 \cdot 0.707) \\
& + 0.6464 \cdot 0.0 && - j(0.6464 \cdot -1.0) \\
& + 1.0607 \cdot 0.707 && - j(1.0607 \cdot 0.707) \\
& + 0.3535 \cdot -1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot 0.707 && - j(-1.0607 \cdot -0.707) \\
& - 1.3535 \cdot 0.0 && - j(-1.3535 \cdot 1.0) \\
& - 0.3535 \cdot -0.707 && - j(-0.3535 \cdot -0.707)
\end{aligned}
$$

$$
\begin{aligned}
= {} & 0.3535 && + j0.0 \\
& - 0.250 && - j0.250 \\
& + 0.0 && + j0.6464 \\
& + 0.750 && - j0.750 \\
& - 0.3535 && - j0.0 \\
& - 0.750 && - j0.750 \\
& + 0.0 && + j1.3535 \\
& + 0.250 && - j0.250
\end{aligned}
$$

$$= 0.0 - j0.0 = 0 \angle 0°.$$

Η DFT ανάλυση παραπάνω δείχνει ότι η ακολουθία x(n) δεν περιέχει συνιστώσα συχνότητας 3kHz.



Σχήμα 2.20 Παράδειγμα DFT (συνέχεια)

Για m=4 και με χρήση των ημιτόνων του Σχήματος 2.20(α), η εξίσωση 2.11 δίνει:

$$
\begin{aligned}
X(4) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot -1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.6464 \cdot 1.0 && - j(0.6464 \cdot 0.0) \\
& + 1.0607 \cdot -1.0 && - j(1.0607 \cdot 0.0) \\
& + 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot -1.0 && - j(-1.0607 \cdot 0.0) \\
& - 1.3535 \cdot 1.0 && - j(-1.3535 \cdot 0.0) \\
& - 0.3535 \cdot -1.0 && - j(-0.3535 \cdot 0.0) \\
= {} & 0.3535 && - j0.0 \\
& - 0.3535 && - j0.0 \\
& + 0.6464 && - j0.0 \\
& - 1.0607 && - j0.0 \\
& + 0.3535 && - j0.0 \\
& + 1.0607 && - j0.0 \\
& - 1.3535 && - j0.0 \\
& + 0.3535 && - j0.0 \\
= {} & 0.0 - j0.0 = 0 \angle 0^\circ.
\end{aligned}
$$

Ομοίως για m=5 έχουμε:

$$
\begin{aligned}
X(5) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot -0.707 && - j(0.3535 \cdot -0.707) \\
& + 0.6464 \cdot 0.0 && - j(0.6464 \cdot 1.0) \\
& + 1.0607 \cdot 0.707 && - j(1.0607 \cdot -0.707) \\
& + 0.3535 \cdot -1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot 0.707 && - j(-1.0607 \cdot 0.707) \\
& - 1.3535 \cdot 0.0 && - j(-1.3535 \cdot -1.0) \\
& - 0.3535 \cdot -0.707 && - j(-0.3535 \cdot 0.707) \\
= {} & 0.3535 && - j0.0 \\
& - 0.250 && + j0.250 \\
& + 0.0 && - j0.6464 \\
& + 0.750 && + j0.750 \\
& - 0.3535 && - j0.0 \\
& - 0.750 && + j0.750 \\
& + 0.0 && - j1.3535 \\
& + 0.250 && + j0.250 \\
= {} & 0.0 - j.0 = 0 \angle 0^\circ.
\end{aligned}
$$

Για m=6:

$$
\begin{aligned}
X(6) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot 0.0 && - j(0.3535 \cdot -1.0) \\
& + 0.6464 \cdot -1.0 && - j(0.6464 \cdot 0.0) \\
& + 1.0607 \cdot 0.0 && - j(1.0607 \cdot 1.0) \\
& + 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot 0.0 && - j(-1.0607 \cdot -1.0) \\
& - 1.3535 \cdot -1.0 && - j(-1.3535 \cdot 0.0) \\
& - 0.3535 \cdot 0.0 && - j(-0.3535 \cdot 1.0)
\end{aligned}
$$

$$
\begin{aligned}
= {} & 0.3535 && - j0.0 \\
& + 0.0 && + j0.3535 \\
& - 0.6464 && - j0.0 \\
& + 0.0 && - j1.0607 \\
& + 0.3535 && - j0.0 \\
& + 0.0 && - j1.0607 \\
& + 1.3535 && - j0.0 \\
& + 0.0 && + j0.3535
\end{aligned}
$$

$$= 1.414 - j1.414 = 2 \angle -45°.$$

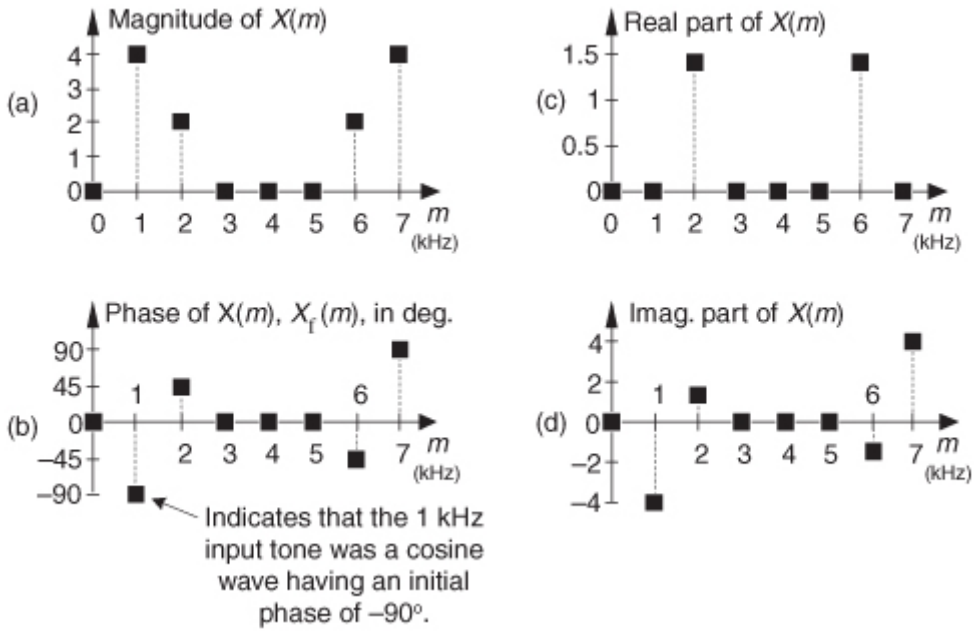
Για m=7:

$$
\begin{aligned}
X(7) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot 0.707 && - j(0.3535 \cdot -0.707) \\
& + 0.6464 \cdot 0.0 && - j(0.6464 \cdot -1.0) \\
& + 1.0607 \cdot -0.707 && - j(1.0607 \cdot -0.707) \\
& + 0.3535 \cdot -1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot -0.707 && - j(-1.0607 \cdot 0.707) \\
& - 1.3535 \cdot 0.0 && - j(-1.3535 \cdot 1.0) \\
& - 0.3535 \cdot 0.707 && - j(-0.3535 \cdot 0.707)
\end{aligned}
$$

$$
\begin{aligned}
= {} & 0.3535 && + j0.0 \\
& + 0.250 && + j0.250 \\
& + 0.0 && + j0.6464 \\
& - 0.750 && + j0.750 \\
& - 0.3535 && - j0.0 \\
& + 0.750 && + j0.750 \\
& + 0.0 && + j1.3535 \\
& - 0.250 && + j0.250
\end{aligned}
$$

$$= 0.0 + j4.0 = 4 \angle 90°.$$

Αναπαριστώντας γραφικά τα πλάτη των εξόδων X(m) ως συνάρτηση της συχνότητας, λαμβάνουμε το φάσμα των πλατών της ακολουθίας εισόδου x(n), όπως το Σχήμα 2.21(a). Στο Σχήμα 2.21(b) φαίνονται οι φάσεις των όρων X(m).



Σχήμα 2.21 Αποτελέσματα μετασχηματισμού DFT για το Παράδειγμα

Τέλος εξετάζουμε τον όρο που αντιστοιχεί στο m=0, καθώς αυτός έχει κάποια ιδιαίτερη σημασία. Η εξίσωση 3-3, για cos(0)=1 και sin(0)=0, δίνει:

$$\boldsymbol{X(0) = \sum_{n=0}^{N-1} x(n)} \qquad \textbf{Εξ. 2.22}$$

Όπως φαίνεται, η παραπάνω εξίσωση δίνει το άθροισμα των δειγμάτων x(n), που φυσικά είναι ανάλογο της μέσης τιμής των όρων x(n). Ειδικότερα, ο όρος X(0) προκύπτει ίσος με N φορές την μέση τιμή της x(n), κάτι που έχει νόημα καθώς ο όρος συχνότητας X(0) αποτελεί την DC συνιστώσα του x(n). Υπολογίζοντας αναλυτικά το X(0) από την εξίσωση 2.11, έχουμε:

$$
\begin{aligned}
X(0) = {} & 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& + 0.6464 \cdot 1.0 && - j(0.6464 \cdot 0.0) \\
& + 1.0607 \cdot 1.0 && - j(1.0607 \cdot 0.0) \\
& + 0.3535 \cdot 1.0 && - j(0.3535 \cdot 0.0) \\
& - 1.0607 \cdot 1.0 && - j(-1.0607 \cdot 0.0) \\
& - 1.3535 \cdot 1.0 && - j(-1.3535 \cdot 0.0) \\
& - 0.3535 \cdot 1.0 && - j(-0.3535 \cdot 0.0)
\end{aligned}
$$

$$
\begin{aligned}
X(0) = {} & 0.3535 && - j0.0 \\
& + 0.3535 && - j0.0 \\
& + 0.6464 && - j0.0 \\
& + 1.0607 && - j0.0 \\
& + 0.3535 && - j0.0 \\
& - 1.0607 && - j0.0 \\
& - 1.3535 && - j0.0 \\
& - 0.3535 && - j0.0
\end{aligned}
$$

$$= 0.0 - j0.0 = 0 \angle 0°.$$

Έτσι προκύπτει ότι το σήμα x(n) δεν περιέχει DC συνιστώσα και επομένως η μέση τιμή του είναι 0. Επίσης παρατηρούμε από το Σχήμα 2.21 ότι το σήμα εισόδου $x_{in}(t)$ της εξίσωσης 2.18, περιέχει όρους συχνότητας 1kHz (m=1) και 2kHz (m=2) και μάλιστα ο όρος του 1kHz έχει διπλάσιο πλάτος του όρου των 2 kHz. Το διάγραμμα του σχ.2.21 που κατασκευάσαμε με βάση την DFT ανάλυση δείχνει ακριβώς το περιεχόμενο του σήματος που ορίζεται από τις εξισώσεις 2.18 και 2.19.

Παρατηρώντας το Σχήμα 2.21(b), βλέπουμε για παράδειγμα τη φάση του όρου X(1) ότι είναι -90^o και θα αναρωτηθούμε εύλογα: «σε σχέση με τι;». Η απάντηση, λαμβάνοντας υπόψη το Σχήμα 2.18 της αναπαράστασης του όρου X(m) σε πολική μορφή, είναι: η φάση ενός όρου DFT συχνότητας mf_s/N είναι η φάση ενός συνημιτόνου ίδιας συχνότητας mf_s/N Hz, όπου m=0, 1, 2, …N-1. Για παράδειγμα, η φάση του X(1) είναι -90^o, που σημαίνει ότι η ημιτονοειδής συνιστώσα του σήματος εισόδου που έχει συχνότητα $1 \cdot fs/N = 1000$ Hz ήταν ένα συνημίτονο με αρχική ολίσθηση φάσης -90^o. Από την τριγωνομετρική ταυτότητα $\cos(\alpha - 90°) = \sin(\alpha)$, βλέπουμε ότι ο τόνος εισόδου των 1kHz ήταν ένα ημίτονο αρχικής φάσης 0^o. Αυτό συμφωνεί με την εξίσωση 2.19 του σήματος της εισόδου. Η φάση του όρου X(2) είναι 45^o, που σημαίνει ότι ο τόνος εισόδου των 2 kHz αντιστοιχεί σε ένα συνημίτονο αρχικής φάσης 45^o, που είναι ισοδύναμο με ένα ημίτονο αρχικής φάσης 135^o ($3\pi/4$ στην εξίσωση 2.19).

Στο Σχήμα 2.21 παρατηρούμε όμως και δύο λίγο περίεργα φαινόμενα: το πρώτο, εμφανίζονται δύο μη μηδενικοί όροι που αντιστοιχούν στις τιμές m=6 και m=7 και το δεύτερο, τα πλάτη των συνιστωσών φαίνονται να είναι τετραπλάσια από τα αναμενόμενα. Αυτά τα δύο φαινόμενα θα εξηγηθούν παρακάτω. Πέρα από αυτά, το παράδειγμα της DFT ανάλυσης 8 σημείων που εξετάσαμε, αν και απλό, μας δείχνει δύο πολύ βασικά χαρακτηριστικά του DFT που δεν πρέπει να ξεχνάμε ποτέ. Το πρώτο: κάθε συνιστώσα X(m) προκύπτει ως άθροισμα των γινομένων όρο με όρο, μιά συσχέτιση (*correlation*) της

ακολουθίας των δειγμάτων της εισόδου με ένα συνημίτονο και ένα ημίτονο των οποίων οι συχνότητες είναι m πλήρεις κύκλοι στο συνολικό διάστημα των N δειγμάτων. Αυτό ισχύει ανεξάρτητα από το ποιος είναι ο ρυθμός δειγματοληψίας f_s ή πόσο μεγάλος είναι ο αριθμός N των σημείων του DFT. Το δεύτερο σημαντικό χαρακτηριστικό της ανάλυσης DFT πραγματικών (όχι μιγαδικών) δειγμάτων εισόδου είναι η συμμετρία των όρων εξόδου.

2.6.3 Συμμετρία του DFT

Παρατηρώντας το διάγραμμα του Σχήμα 2.21(a), φαίνεται ξεκάθαρα η συμμετρία που υπάρχει στο φάσμα εξόδου. Αν και η διαδικασία DFT κανονικά δέχεται μιγαδικές ακολουθίες δειγμάτων, για τα περισσότερα σήματα που συναντάμε στη φύση θεωρούμε ότι δίνουν τιμές δειγματοληψίας που έχουν μη μηδενικό πραγματικό μέρος ενώ το φανταστικό μέρος τους είναι μηδενικό. Για τέτοιες πραγματικές ακολουθίες εισόδου οι μιγαδικοί όροι DFT της εξόδου, εκτός των όρων για m=1 έως m=(N/2)-1, περιλαμβάνουν επιπλέον τιμές συχνοτήτων εξόδου για m>N/2. Ο όρος m έχει το ίδιο πλάτος και αρνητική τιμή φάσης σε σχέση με το πλάτος και τη φάση του όρου N-m. Επομένως, οι όροι m και N-m του DFT μιας πραγματικής ακολουθίας εισόδου, συνδέονται με τη σχέση

$$\boldsymbol{X(m) = X^*(N - m)} \qquad \textbf{Εξ. 2.23}$$

όπου το σύμβολο * δηλώνει το συζυγή μιγαδικό.

Με βάση τα παραπάνω παρατηρούμε ότι στο σχήμα Σχήμα 2.21 οι όροι X(5), X(6) και X(7) είναι οι *συζυγείς μιγαδικοί* των όρων X(3), X(2) και X(1) αντίστοιχα. Η συζυγής συμμετρία έχει ως αποτέλεσμα, όταν εφαρμόσουμε DFT N δειγμάτων σε μία ακολουθία εισόδου πραγματικών τιμών, να προκύπτουν N μιγαδικοί όροι εξόδου που όμως μόνον οι πρώτοι N/2 +1 (από X(0) έως X(N/2)) είναι ανεξάρτητοι. Οι υπόλοιποι (από X(N/2 +1) έως X(N-1)) είναι περιττοί, καθώς δεν παρέχουν κάποια επιπλέον πληροφορία για το φάσμα της εισόδου.

2.6.4 Μέγεθος όρων DFT (DFT Magnitudes)

Όπως παρατηρήσαμε στο φάσμα εξόδου του προηγούμενου παραδείγματος, προκύπτει ότι |X(1)| = 4 και |X(2)| = 2, αν και από την εξίσωση 2.19 στο σήμα εισόδου x(n) τα μέγιστα πλάτη είναι 1.0 και 0.5 αντίστοιχα. Υπάρχει ένα σημαντικό σημείο που πρέπει να έχουμε υπόψη μας, όσον αφορά το DFT όπως αυτό εκφράζεται με την εξίσωση 2.10. Όταν ένα **πραγματικό** σήμα εισόδου περιέχει μια ημιτονοειδή συνιστώσα πλάτους A_0 και ακέραιο αριθμό κύκλων εντός του διαστήματος των N δειγμάτων, το μέγεθος του όρου εξόδου DFT για τη συγκεκριμένη συνιστώσα θα είναι M_r, όπου:

$$\mathbf{M_r = A_0 N/2.}$$

Αν το σήμα εισόδου είναι **μιγαδικό** ημιτονοειδές πλάτους A_0, π.χ. $A_0 e^{j2\pi ft}$, με ακέραιο πλήθος κύκλων κατά μήκος των N δειγμάτων, τότε το πλάτος εξόδου DFT θα είναι M_c, όπου

$$\mathbf{M_c = A_0 N.}$$

Αν το σήμα εισόδου περιέχει DC συνιστώσα πλάτους D_0, τότε ο αντίστοιχος όρος X(0) στην έξοδο του DFT θα πλάτος **D_0N.**

Έτσι για το παράδειγμα που είδαμε παραπάνω, η συνιστώσα του 1kHz έχει $A_0 = 1$, οπότε με N=8 προκύπτει $M_r = 1*8/2 = 4$.

Οι παραπάνω εξισώσεις πρέπει να λαμβάνονται υπόψη κατά την υλοποίηση ενός μηχανισμού DFT, σε software ή hardware. Οταν για παράδειγμα υλοποιήσουμε σε hardware έναν αλγόριθμο DSP με αριθμούς σταθερής υποδιαστολής (fixed point), πρέπει να υπολογίσουμε ότι τα πλάτη των δειγμάτων της εισόδου μπορεί να αυξηθούν κατά N/2 και επομένως πρέπει να προβλεφθεί κατάλληλα το μέγεθος των registers της μνήμης ώστε να μπορούν να αποθηκεύσουν τις αυξημένες τιμές.

2.6.5 Άξονας συχνοτήτων DFT

Ο οριζόντιος άξονας συχνοτήτων του αποτελέσματος που προκύπτει από την ανάλυση DFT (Σχήμα 2.21), μπορεί να εκφραστεί με 4 διαφορετικούς τρόπους. Με τον $1^ο$ τρόπο, ο οριζόντιος άξονας αναφέρει τους ακέραιους αριθμούς m από 0 έως N/2 και κάθε ένας αντιστοιχεί στον δείκτη m κάθε όρου της ακολουθίας εξόδου X(m), δηλαδή έχει βήμα 1. Ο τρόπος αυτός ευνοεί τον προγραμματισμό καθώς συμπίπτει με τον τρόπο γραφής κώδικα για την πρόσβαση των πινάκων μέσω δεικτών.

Με τον δεύτερο τρόπο ο οριζόντιος άξονας αναφέρεται ως κλάσμα του ρυθμού δειγματοληψίας. Αυτό σημαίνει ότι οι τιμές που παίρνει κυμαίνονται από 0 έως 0,5, καθώς τα διακριτά δεδομένα μπορούν να περιέχουν συχνότητες από DC έως το μισό του ρυθμού δειγματοληψίας. Ο δείκτης που χρησιμοποιείται σε αυτό το συμβολισμό είναι f, ως συχνότητα και παίρνει N/2 + 1 ισαπέχουσες τιμές μεταξύ 0 και 0,5. Έτσι, για να πάμε από τον πρώτο συμβολισμό m στον δεύτερο f, διαιρούμε τον οριζόντιο άξονα με το N και παίρνουμε τα σημεία f = m/N, όπου m από 0 έως N/2 με βήμα 1/N και άρα το f από 0/N = 0 έως N/(N/2) = ½. Ο τρόπος αυτός υποδεικνύει συνεχώς ότι τα διακριτά σήματα περιέχουν συχνότητες από 0 έως 0,5 του ρυθμού δειγματοληψίας.

Ο τρίτος τρόπος είναι παρόμοιος με τον δεύτερο, εκτός του ότι οι τιμές του οριζόντιου άξονα πολλαπλασιάζονται με 2π. Ο δείκτης που χρησιμοποιείται για το συμβολισμό αυτό είναι το ω, που καλείται *φυσική συχνότητα* και έχει μονάδες *ακτίνια* (*radians*). Αυτό βασίζεται στην ιδέα ότι ο κύκλος έχει 2π ακτίνια και απλοποιεί τη γραφή των εξισώσεων DFT.

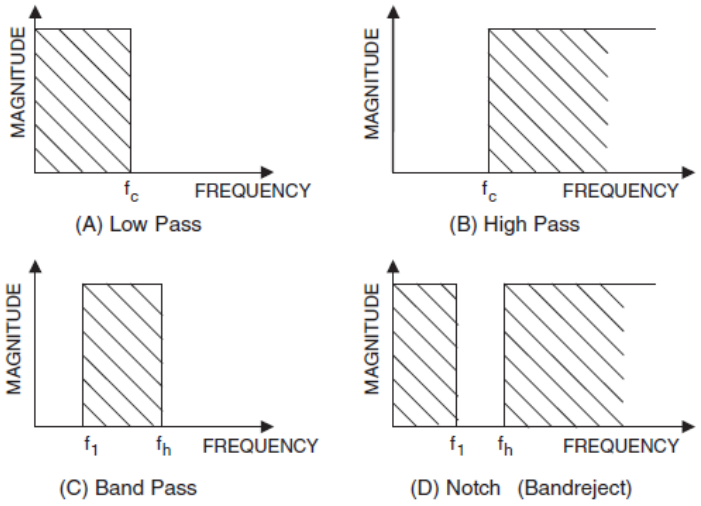
Ο τέταρτος τρόπος εμφανίζει στον οριζόντιο άξονα τιμές των αναλογικών συχνοτήτων που αντιστοιχούν σε συγκεκριμένη εφαρμογή. Αν θεωρήσουμε ότι μια εφαρμογή έχει ρυθμό δειγματοληψίας 10.000 samples/sec, τότε ο άξονας συχνοτήτων θα παίρνει τιμές από 0 έως 5.000 Hz με βήμα f_s/N, όπου f_s ο ρυθμός δειγματοληψίας. Το θετικό αυτού του συμβολισμού είναι ότι παρουσιάζει τα δεδομένα της συχνότητας με τον γνωστό τρόπο που γνωρίζουμε, αλλά αφορά μόνο κάποιο συγκεκριμένο ρυθμό δειγματοληψίας και επομένως δεν μπορεί να χρησιμοποιηθεί με γενικό τρόπο.

2.7 Αναλογικά Φίλτρα

2.7.1 Γενικές παράμετροι Φίλτρων

Τα φίλτρα είναι δίκτυα που επεξεργάζονται το σήμα στο πεδίο της συχνότητας και έχουν πολλές πρακτικές εφαρμογές. Ένα απλό βαθυπερατό φίλτρο ενός πόλου (ολοκληρωτής) χρησιμοποιείται συχνά για να σταθεροποιήσει τη λειτουργία ενός ενισχυτή, μειώνοντας το κέρδος στις υψηλότερες συχνότητες για την αποφυγή αντιστροφής φάσης. Ομοίως, ένα υψυπερατό φίλτρο μπορεί να χρησιμοποιηθεί για να απαγορεύσει τη διέλευση DC συνιστωσών που περιέχονται σε ένα σήμα. Επίσης τα φίλτρα χρησιμοποιούνται για το διαχωρισμό σημάτων έτσι ώστε να επιτρέπεται η διέλευση των επιθυμητών και να απορρίπτονται τα υπόλοιπα.

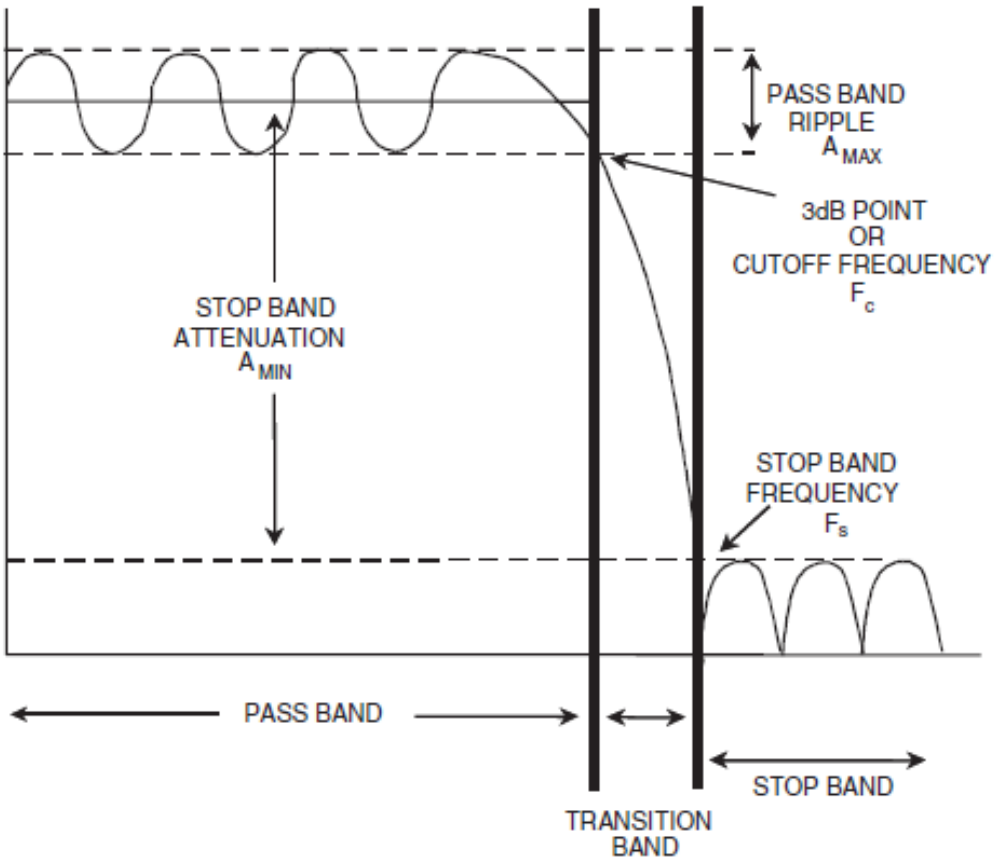
Ένα ιδανικό φίλτρο θα πρέπει να παρουσιάζει απόκριση με σταθερό κέρδος για τις επιθυμητές συχνότητες (ζώνη διέλευσης - pass band) και μηδενικό σε όλες τις υπόλοιπες (ζώνη αποκοπής - stop band). Η συχνότητα στην οποία παρουσιάζεται μετάβαση από τη μια ζώνη στην άλλη ονομάζεται συχνότητα αποκοπής (cutoff frequency).



Σχήμα 2.22 Απόκριση ιδανικών Φίλτρων

Τα ιδανικά φίλτρα είναι σχεδόν αδύνατο να υλοποιηθούν, καθώς η μετάβαση από τη ζώνη διέλευσης στην αποκοπής δεν πραγματοποιείται απότομα αλλά υπάρχει μια περιοχή μετάβασης. Επίσης η εξασθένιση του σήματος στη ζώνη αποκοπής δεν μπορεί να είναι άπειρη.

Ένα πραγματικό φίλτρο που υλοποιείται στην πράξη, χαρακτηρίζεται από 5 παραμέτρους, όπως φαίνονται στο Σχήμα 2.23.



Σχήμα 2.23 Χαρακτηριστικές περιοχές και παράμετροι πραγματικών Φίλτρων

Η συχνότητα αποκοπής (Fc) είναι η συχνότητα στην οποία η απόκριση του φίλτρου αφήνει τη ζώνη σφάλματος (ή το σημείο −3 dB για ένα φίλτρο τύπου Butterworth). Η συχνότητα ζώνης αποκοπής (Fs) είναι η συχνότητα στην οποία επιτυγχάνεται η ελάχιστη εξασθένιση στη ζώνη αποκοπής. Η κυμάτωση ζώνης διέλευσης (Amax) είναι η διακύμανση (ζώνη σφάλματος) της απόκρισης της ζώνης διέλευσης. Η ελάχιστη εξασθένηση ζώνης διέλευσης (Amin) ορίζει την ελάχιστη εξασθένιση του σήματος εντός της ζώνης διέλευσης. Το μέγεθος που δείχνει πόσο απότομα πραγματοποιείται η μετάβαση στη ζώνη αποκοπής του φίλτρου ορίζεται ως η τάξη (M) του φίλτρου, που δηλώνει επίσης τον αριθμό των πόλων της συνάρτησης μεταφοράς του φίλτρου. Κάθε πόλος έχει απόκριση –6 dB/οκτάβα ή –20 dB/δεκάδα ενώ κάθε μηδενικό +6 dB/οκτάβα ή +20 dB/δεκάδα.

Σημειώνεται ότι ορισμένα είδη φίλτρων δεν διαθέτουν όλα τα παραπάνω χαρακτηριστικά. Για παράδειγμα η σχεδίαση μόνο με πόλους – all pole (δηλαδή χωρίς μηδενικά στη συνάρτηση μεταφοράς) δεν παρουσιάζει κυμάτωση στη ζώνη αποκοπής. Τα φίλτρα Butterworth και Bessel είναι παραδείγματα all pole σχεδιασμού που δεν παρουσιάζουν κυμάτωση ούτε στη ζώνη διέλευσης.

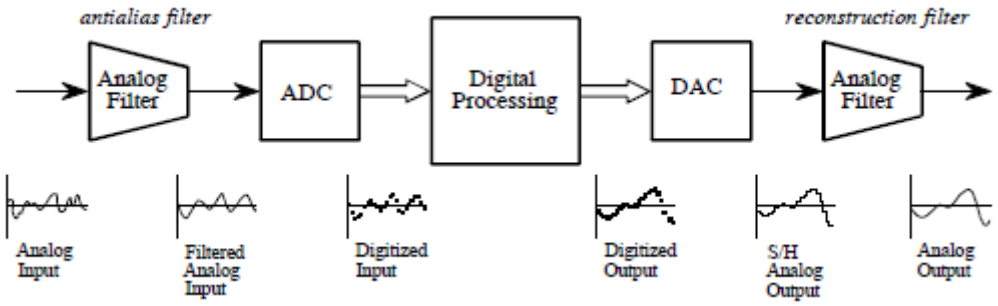
Γενικά, μία ή περισσότερες από τις παραπάνω παραμέτρους είναι μεταβαλλόμενες. Για παράδειγμα, κατά το σχεδιασμό ενός φίλτρου antialiasing για μετατροπέα ADC, είναι γνωστή η συχνότητα αποκοπής (που γενικά ταυτίζεται με τη συχνότητα Nyquist) καθώς

και η ελάχιστη εξασθένηση που απαιτείται με βάση το resolution του ADC ή τη δυναμική περιοχή του συστήματος. Οι υπόλοιπες παράμετροι υπολογίζονται.

Τέλος επισημαίνεται ότι το φίλτρο επιδρά στο πλάτος αλλά και στη φάση του σήματος. Για παράδειγμα, ένας απλός πόλος μεταβάλει τη φάση κατά 90^o στη μεταβατική συχνότητα ενώ ένα ζεύγος πόλων κατά 180^o . Ο συντελεστής ποιότητας Q του φίλτρου είναι αυτός που καθορίζει το ρυθμό μεταβολής της φάσης.

2.7.2 Αναλογικά φίλτρα για μετατροπή δεδομένων

Το Σχήμα 2.24 παρουσιάζει ένα μπλοκ διάγραμμα ενός DSP συστήματος, όπως το θεώρημα δειγματοληψίας επιβάλλει να είναι.

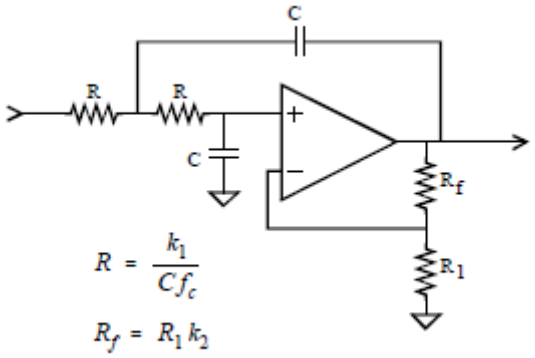


Σχήμα 2.24 Αναλογικά φίλτρα σε ένα σύστημα DSP

Πριν συναντήσει τον analog-to-digital μετατροπέα, το εισερχόμενο σήμα υποβάλλεται σε επεξεργασία με ένα ηλεκτρονικό βαθυπερατό φίλτρο για να αφαιρέσει όλες τις συχνότητες επάνω από τη συχνότητα Nyquist (μισό του ρυθμού δειγματοληψίας). Αυτό γίνεται για να αποτρέψει το aliasing κατά τη διάρκεια της δειγματοληψίας, και καλείται αντίστοιχα φίλτρο antialiasing. Από την άλλη πλευρά, το τροποποιημένο ψηφιακό σήμα περνάει μέσα από έναν digital-to-analog μετατροπέα και ένα άλλο χαμηλής διέλευσης φίλτρο ρυθμισμένο στη συχνότητα Nyquist. Αυτό το φίλτρο εξόδου καλείται φίλτρο αναδημιουργίας (**reconstruction filter**), και μπορεί να συμπεριλάβει την ενίσχυση των zeroth-order-hold συχνοτήτων. Δυστυχώς, υπάρχει ένα σοβαρό πρόβλημα με αυτό το απλό μοντέλο: οι περιορισμοί των ηλεκτρονικών φίλτρων μπορούν να είναι τόσο κακοί όσο τα προβλήματα τα οποία προσπαθούν να αντιμετωπίσουν.

Συνήθως χρησιμοποιούνται τρεις τύποι αναλογικών φίλτρων: **Chebyshev**, **Butterworth**, και **Bessel** (επίσης αποκαλούμενο φίλτρο Thompson). Κάθε ένας από αυτά έχει ως σκοπό να βελτιστοποιήσει μια διαφορετική παράμετρο απόδοσης. Η πολυπλοκότητα κάθε φίλτρου μπορεί να ρυθμιστεί με την επιλογή του αριθμού πόλων και μηδενικών της συνάρτησης μεταφοράς του, μαθηματικοί όροι που θα συζητηθούν αργότερα. Όσο περισσότεροι οι πόλοι σε ένα φίλτρο, τόσο περισσότερα ηλεκτρονικά απαιτούνται, αλλά και τόσο καλύτερα αυτό αποδίδει. Κάθε ένα από τα παραπάνω ονόματα περιγράφει το τι κάνει το φίλτρο και όχι μια ιδιαίτερη ρύθμιση των αντιστάσεων και των πυκνωτών. Για παράδειγμα, ένα φίλτρο Bessel έξι πόλων μπορεί να υλοποιηθεί από πολλούς

διαφορετικούς τύπους ηλεκτρονικών κυκλωμάτων, οι οποίοι έχουν τα ίδια γενικά χαρακτηριστικά. Για τη DSP, τα χαρακτηριστικά αυτών των φίλτρων είναι σημαντικότερα από το πώς κατασκευάζονται. Εντούτοις, θα αρχίσουμε με μια σύντομη αναφορά στην ηλεκτρονική σχεδίαση αυτών των φίλτρων, έτσι ώστε να έχουμε ένα γενικό πλαίσιο.



Σχήμα 2.25 Τροποποιημένο κύκλωμα Sallen-Key

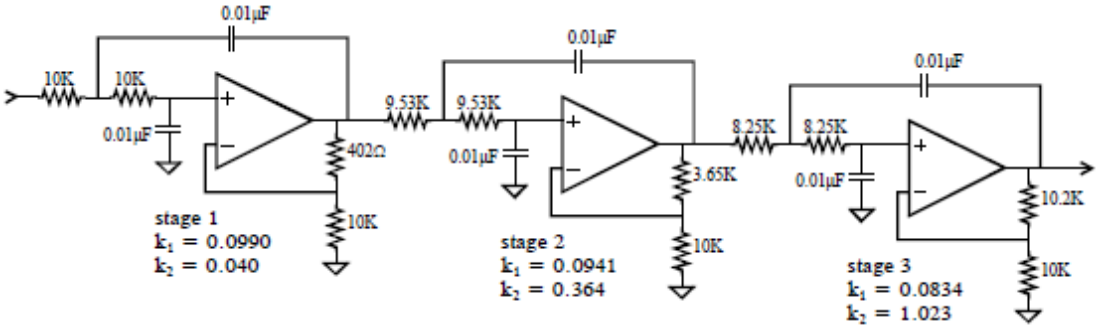
Το Σχήμα 2.25 παρουσιάζει μία συνηθισμένη δομική μονάδα για τη σχεδίαση των αναλογικών φίλτρων, το τροποποιημένο κύκλωμα **Sallen-Key**. Αυτό πήρε το όνομα των συγγραφέων που περιγράφουν την αντίστοιχη τεχνική σε μελέτης της δεκαετίας του '50. Το κύκλωμα που παρουσιάζεται είναι ένα βαθυπερατό φίλτρο 2 πόλων που μπορεί να διαμορφωθεί σε οποιονδήποτε από τους τρεις βασικούς τύπους. Ο πίνακας 2.1 παρέχει τις απαραίτητες πληροφορίες για την επιλογή των κατάλληλων αντιστάσεων και πυκνωτών. Για παράδειγμα, για να σχεδιάσουμε ένα φίλτρο Butterworth 2 πόλων 1 kHz, ο πίνακας 3-1 παρέχει τις παραμέτρους: k1 = 0.1592 και k2 = 0.586. Αυθαίρετα επιλέγοντας R1 = 10K και το C = 0.01uF (κοινές τιμές για κυκλώματα τελεστικών ενισχυτών), οι R και Rf μπορούν να υπολογιστούν ως 15.95K και 5.86K, αντίστοιχα. Στρογγυλεύοντας αυτές τις τελευταίες δύο τιμές στις κοντινότερες 1% τυποποιημένες αντιστάσεις, δίνουν R = 15.8K και Rf = 5.90K ενώ όλα τα στοιχεία πρέπει να έχουν ακρίβεια 1% ή καλύτερη.

# poles		Bessel		Butterworth		Chebyshev	
		k_1	k_2	k_1	k_2	k_1	k_2
2	stage 1	0.1251	0.268	0.1592	0.586	0.1293	0.842
4	stage 1	0.1111	0.084	0.1592	0.152	0.2666	0.582
	stage 2	0.0991	0.759	0.1592	1.235	0.1544	1.660
6	stage 1	0.0990	0.040	0.1592	0.068	0.4019	0.537
	stage 2	0.0941	0.364	0.1592	0.586	0.2072	1.448
	stage 3	0.0834	1.023	0.1592	1.483	0.1574	1.846
8	stage 1	0.0894	0.024	0.1592	0.038	0.5359	0.522
	stage 2	0.0867	0.213	0.1592	0.337	0.2657	1.379
	stage 3	0.0814	0.593	0.1592	0.889	0.1848	1.711
	stage 4	0.0726	1.184	0.1592	1.610	0.1582	1.913

Πίνακας 2.1 Συντελεστές υλοποίησης φίλτρων Bessel, Butterworth, Chebyshev

Οι συγκεκριμένοι τελεστικοί που χρησιμοποιούνται δεν είναι κρίσιμοι, εφ' όσον η συχνότητα μοναδιαίου κέρδους είναι μεγαλύτερη από 30 έως 100 φορές υψηλότερη από τη συχνότητα αποκοπής του φίλτρου. Αυτό είναι μια εύκολη απαίτηση στην περίπτωση που η συχνότητα αποκοπής του φίλτρου είναι χαμηλότερη από περίπου 100 kHz. Τεσσάρων, έξι,

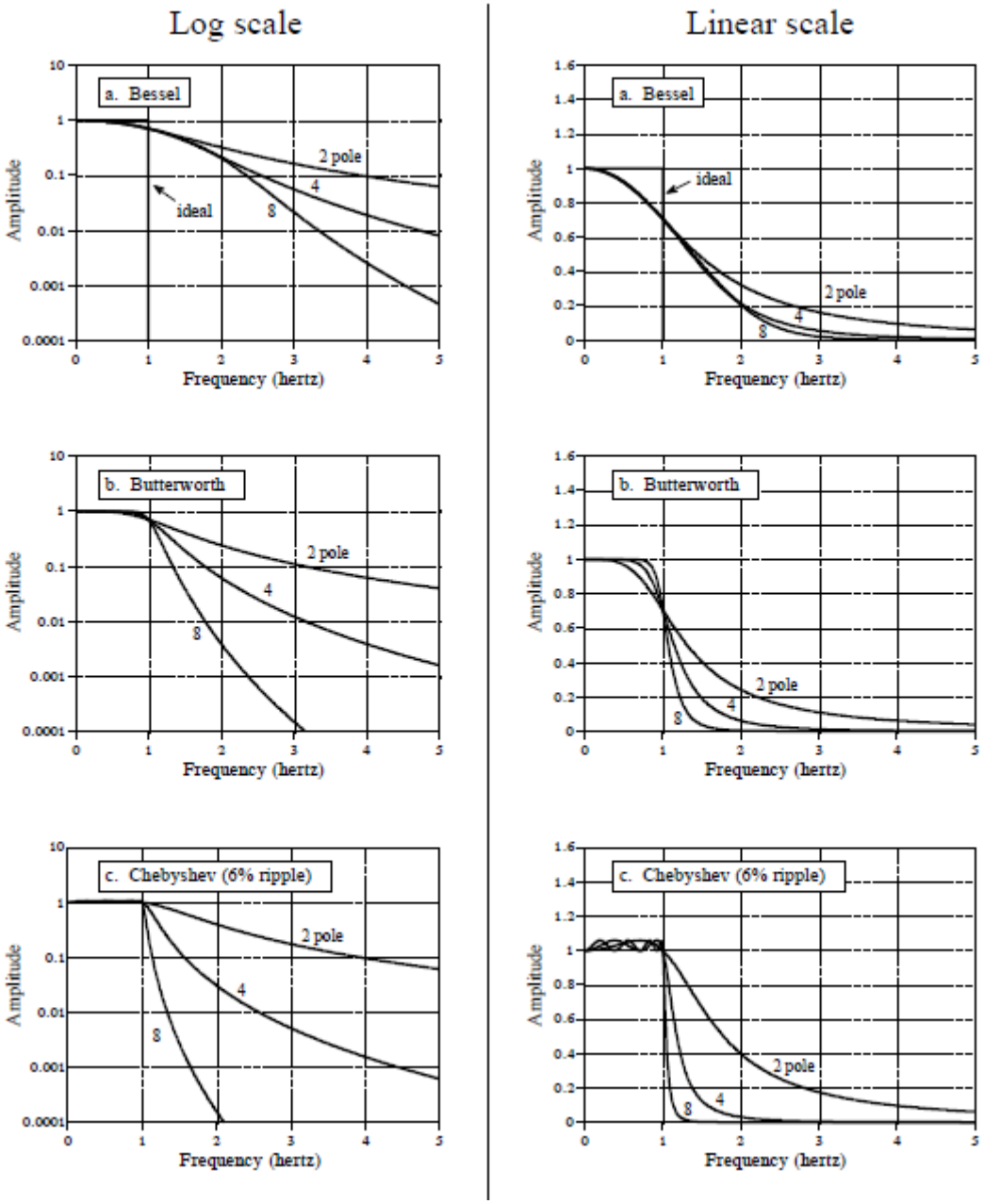
και οκτώ πόλων φίλτρα υλοποιούνται εύκολα συνδέοντας σε σειρά 2,3, και 4 από αυτά τα κυκλώματα, αντίστοιχα. Παραδείγματος χάριν, το Σχήμα 2.26 παρουσιάζει τη σχηματική αναπαράσταση ενός φίλτρου Bessel 6 πόλων που δημιουργείται συνδέοντας σε σειρά τρείς τέτοιες βαθμίδες. Κάθε βαθμίδα έχει διαφορετικές τιμές για τα k1 και k2, όπως προβλέπεται από τον πίνακα 2.1, με συνέπεια να χρησιμοποιούνται διαφορετικές αντιστάσεις και πυκνωτές. Εάν χρειάζεται να υλοποιήσουμε ένα υψυπερατό φίλτρο, απλά αντιμεταθέτουμε τα στοιχεία R και C στα κυκλώματα (αφήνοντας στη θέση τους μόνο το Rf και R1).



Σχήμα 2.26 Φίλτρο Bessel 6 πόλων με 3 κυκλώματα Sallen-Key στη σειρά

Αυτός ο τύπος κυκλώματος είναι πολύ συνηθισμένος για κατασκευές μικρής ποσότητας και R&D εφαρμογές, εντούτοις, μία σοβαρή παραγωγή απαιτεί το φίλτρο να υλοποιηθεί ως *ολοκληρωμένο κύκλωμα*. Το πρόβλημα είναι, ότι είναι δύσκολο να υλοποιηθούν αντιστάσεις άμεσα στο πυρίτιο. Στην περίπτωση αυτή, η λύση έρχεται με το φίλτρο διακοπτόμενου πυκνωτή (**switched capacitor filter**).

Στη συνέχεια θα εξετάσουμε τα χαρακτηριστικά των τριών κλασικών τύπων φίλτρων. Η πρώτη παράμετρος απόδοσης που θέλουμε να εξερευνήσουμε είναι η οξύτητα της συχνότητας αποκοπής (**cutoff frequency sharpness**). Ένα βαθυπερατό φίλτρο σχεδιάζεται για να αποκόψει όλες τις συχνότητες που είναι μεγαλύτερες από τη συχνότητα αποκοπής (περιοχή αποκοπής - **stopband**), επιτρέποντας όλες τις χαμηλότερες συχνότητες από αυτή (περιοχή διέλευσης - **passband**). Το σχήμα 2.27 παρουσιάζει την απόκριση συχνότητας αυτών των τριών φίλτρων σε μια λογαριθμική κλίμακα (dB). Αυτές οι γραφικές παραστάσεις παρουσιάζονται για φίλτρα με συχνότητα αποκοπής 1 Hz, αλλά μπορούν να μετατραπούν άμεσα σε οποιαδήποτε συχνότητα αποκοπής επιλέξουμε να χρησιμοποιήσουμε. Πώς αυτά τα φίλτρα αξιολογούνται; Το Chebyshev είναι σαφώς το καλύτερο, το Butterworth είναι χειρότερο, και το Bessel είναι κάκιστο! Αυτό που το Chebyshev έχει σχεδιασθεί να κάνει είναι η μείωση (πτώση) στο πλάτος του σήματος, όσο το δυνατόν πιο γρήγορα.



Σχήμα 2.27 Απόκριση συχνότητας των τριών φίλτρων σε λογαριθμική και γραμμική κλίμακα

Δυστυχώς, ούτε ένα Chebyshev 8 πόλων δεν είναι τόσο καλό όσο θα θέλαμε για ένα φίλτρο antialias. Για παράδειγμα, θεωρούμε ένα σύστημα 12 bit με δειγματοληψία 10.000 δείγματα ανά δευτερόλεπτο. Το θεώρημα δειγματοληψίας υπαγορεύει ότι οποιαδήποτε συχνότητα μεγαλύτερη από τα 5 kHz θα παρουσιάσει aliasing, κάτι το οποίο θέλουμε να αποφύγουμε. Με λίγη εργασία βάση της εμπειρίας μας, αποφασίζουμε ότι όλες οι συχνότητες επάνω από το 5 kHz πρέπει να μειωθούν στο πλάτος κατά έναν παράγοντα 100, εξασφαλίζοντας ότι οποιεσδήποτε συχνότητες προκαλούν aliasing θα έχουν πλάτος λιγότερο από 1%. Εξετάζοντας το γράφημα 2.27c, διαπιστώνουμε ότι ένα φίλτρο Chebyshev 8 πόλων, με μια συχνότητα αποκοπής 1 Hz, δεν αγγίζει τη θεωρούμενη μείωση κατά τον παράγοντα 100 παρά μόνο μετά περίπου τη συχνότητα 1.35 Hz. Εφαρμόζοντας αυτό στο παράδειγμα, η συχνότητα αποκοπής του φίλτρου πρέπει να τεθεί στα 3.7 kHz

έτσι ώστε όλες οι συχνότητες πάνω από τα 5 kHz να έχουν την απαιτούμενη μείωση. Αυτό οδηγεί στο να χάνεται η ζώνη συχνοτήτων μεταξύ του 3.7 kHz και 5 kHz καθώς σε αυτή έχουμε ανεπαρκή μείωση του πλάτους του σήματος.

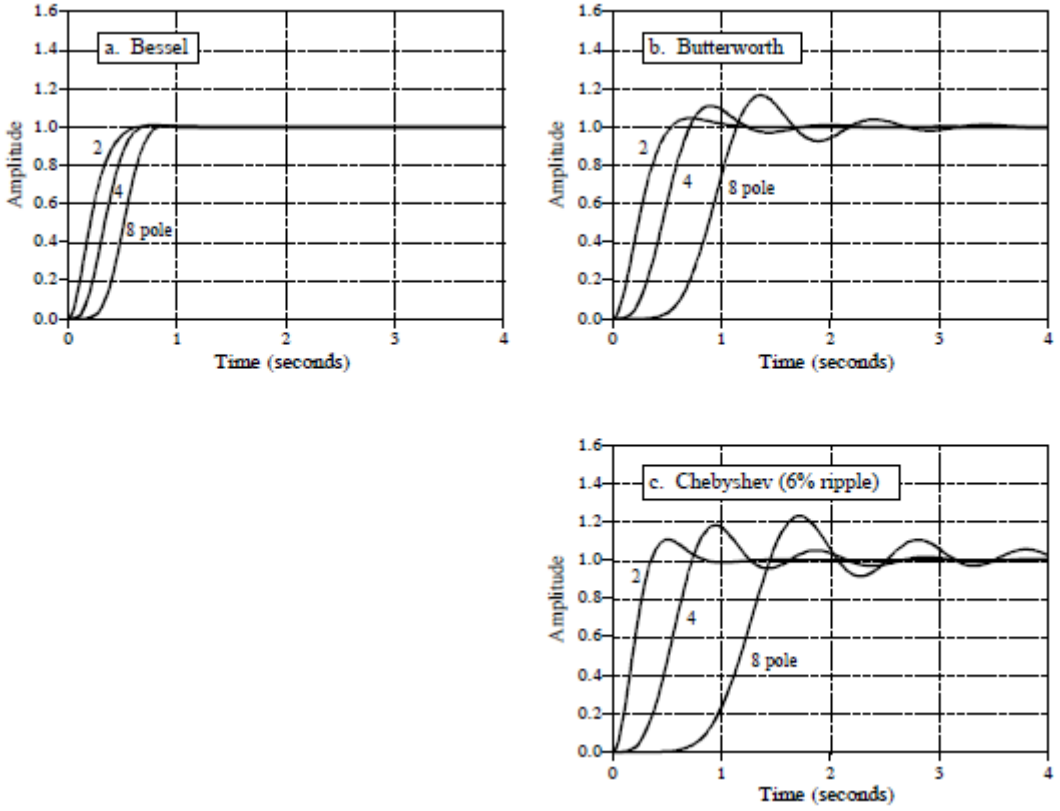
Ένα λεπτό σημείο: ο παράγοντας μείωσης 100 σε αυτό το παράδειγμα είναι πιθανώς ικανοποιητικός ακόμα κι αν υπάρχουν 4096 βήματα στα 12 bits. Από το γράφημα 3-4 βλρέπουμε ότι τα 5100 Hz προκαλούν alias στα 4900 Hz, τα 6000 Hz στα 4000 Hz, κ.λπ. Δεν ενδιαφερόμαστε για το πλάτος των σημάτων μεταξύ 5000 και 6300 Hz, επειδή αυτά προκαλούν alias στην ανεπιθύμητη προς χρήση περιοχή μεταξύ 3700 Hz και 5000 Hz. Για να προκαλέσει μια συχνότητα alias στην passband του φίλτρου (0 kHz έως 3.7), πρέπει αυτή να είναι μεγαλύτερη από 6300 Hz, ή 1.7 φορές η συχνότητα αποκοπής του φίλτρου 3700 Hz. Όπως φαίνεται στο γράφημα 2.27c, η μείωση που παρέχεται από ένα Chebyshev 8 πόλων φίλτρο σε 1.7 φορές της συχνότητας αποκοπής, είναι περίπου 1300, αρκετά επαρκέστερη από τα 100 που είχαμε στην προηγούμενη ανάλυση. Επομένως το συμπέρασμα είναι ότι *στα περισσότερα συστήματα, η ζώνη συχνοτήτων μεταξύ περίπου 0.4 και 0.5 της συχνότητας δειγματοληψίας είναι μια ακατάλληλη προς χρήση περιοχή που χάνεται λόγω της ζώνης μετάβασης των φίλτρων (από την περιοχή διέλευσης στην αποκοπής) καθώς και λόγω alias συχνοτήτων*. Αυτό είναι ένα άμεσο αποτέλεσμα των περιορισμών των αναλογικών φίλτρων.

Η απόκριση συχνότητας του ιδανικού χαμηλής διέλευσης φίλτρου είναι επίπεδη σε ολόκληρη την passband. Όλα τα φίλτρα φαίνονται άψογα από αυτή την άποψη στο Σχήμα 2.27, αλλά μόνο επειδή ο κάθετος άξονας παρουσιάζεται σε λογαριθμική κλίμακα. Το ζήτημα αλλάζει όταν οι γραφικές παραστάσεις μετατρέπονται σε μια γραμμική κάθετη κλίμακα. Ο κυματισμός της passband μπορεί τώρα να φανεί στο Chebyshev φίλτρο (κυματιστές μεταβολές στο πλάτος των συχνοτήτων διέλευσης). Στην πραγματικότητα, το Chebyshev παρέχει άριστη αποκοπή σε αντιστάθμισμα αυτού τού κυματισμού στην passband. Όταν σε ένα φίλτρο επιτρέπεται μεγαλύτερος κυματισμός στη ζώνη διέλευσης, μπορεί να επιτευχθεί μια ταχύτερη αποκοπή. Όλα τα Chebyshev φίλτρα που σχεδιάζονται με τη χρησιμοποίηση του πίνακα 2.1 έχουν έναν κυματισμό στην passband περίπου 6% (0.5 dB), που αποτελεί έναν καλό συμβιβασμό και μια συνήθη επιλογή. Ένας παρόμοιος σχεδιασμός, το ελλειπτικό φίλτρο, παρουσιάζει κυματισμό και στην passband και στην stopband. Αν και δυσκολότερα να σχεδιαστούν, τα ελλειπτικά φίλτρα μπορούν να επιτύχουν μια ακόμα καλύτερη ανταλλαγή μεταξύ της ταχύτητας αποκοπής και του κυματισμού της passband.

Συγκριτικά, το φίλτρο Butterworth παρέχει την βέλτιστη ταχύτητα αποκοπής χωρίς να επιτρέπει τη δημιουργία του κυματισμού στην passband. Καλείται συνήθως ως maximally flat filter, και είναι ίδιο με το Chebyshev που σχεδιάζεται για μηδενικό κυματισμό στην passband. Το φίλτρο Bessel δεν παρουσιάζει κανέναν κυματισμό, αλλά έχει ταχύτητα αποκοπής πολύ χειρότερη από το Butterworth.

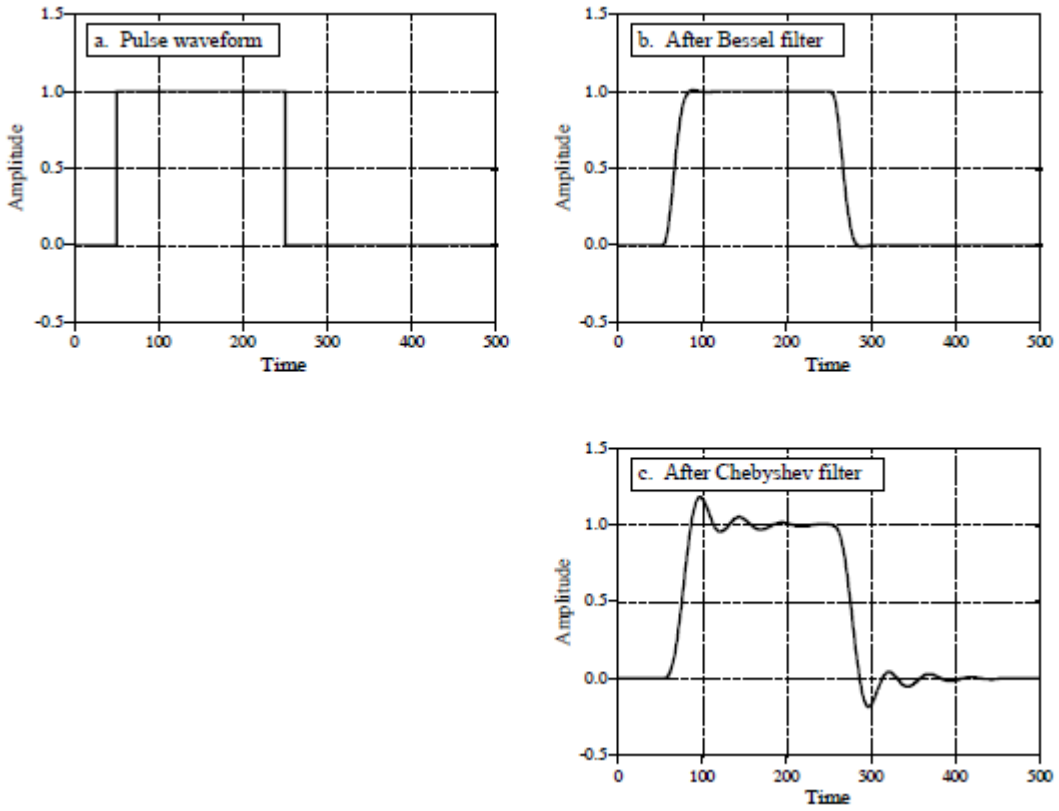
Η τελευταία παράμετρος που αξιολογούμε είναι η απόκριση σε βηματική είσοδο, δηλαδή πώς αποκρίνεται το φίλτρο όταν αλλάζει απότομα η είσοδος από μια τιμή σε μίαν άλλη. Το σχήμα 2.28 παρουσιάζει τη βηματική απόκριση για κάθε ένα από τα τρία φίλτρα. Ο

οριζόντιος άξονας παρουσιάζεται για τα φίλτρα με συχνότητα αποκοπής 1 Hz, αλλά μπορεί να μετατραπεί ώστε να αντιστοιχεί σε υψηλότερες συχνότητες αποκοπής. Παραδείγματος χάριν, μια συχνότητα αποκοπής 1000 Hz θα παρουσίαζε βηματική απόκριση σε χιλιοστά του δευτερολέπτου, όχι δευτερόλεπτα. Τα φίλτρα Butterworth και Chebyshev παρουσιάζουν overshoot (υπερανύψωση) και ringing (ταλαντώσεις που μειώνονται αργά κατά πλάτος) ενώ το φίλτρο Bessel δεν παρουσιάζει κανένα από αυτά τα δυσάρεστα φαινόμενα.



Σχήμα 2.28 Βηματική απόκριση των τριών Φίλτρων

Το Σχήμα 2.29 επεξηγεί περαιτέρω αυτό το πολύ ευνοϊκό χαρακτηριστικό του φίλτρου Bessel. Το γράφημα (a) παρουσιάζει την κυματομορφή ενός τετραγωνικού παλμού. Τα γραφήματα (b) και (c) δείχνουν πώς θα εμφανιζόταν αυτή η κυματομορφή μετά τη διέλευσή της από φίλτρα Bessel και Chebyshev, αντίστοιχα. Εάν αυτό ήταν για παράδειγμα ένα σήμα video, η παραμόρφωση που εισάγει το Chebyshev φίλτρο θα ήταν καταστρεπτική! Η υπερανύψωση θα άλλαζε τη φωτεινότητα των άκρων των αντικειμένων σε σχέση με τα κέντρα τους. Ακόμη χειρότερα, η αριστερή πλευρά των αντικειμένων θα φαινόταν φωτεινή, ενώ η δεξιά σκοτεινή. Πολλές εφαρμογές δεν μπορούν να ανεχτούν την κακή απόδοση στην βηματική απόκριση. Εκεί είναι όπου το φίλτρο Bessel υπερτερεί (καμία υπερύψωση και συμμετρικές άκρες).



Σχήμα 2.29 Απόκριση Bessel και Chebyshev σε τετραγωνικό παλμό

2.7.3 Επιλέγοντας το κατάλληλο φίλτρο antialias

Ο πίνακας 2.2 συνοψίζει τα χαρακτηριστικά των τριών φίλτρων, δείχνοντας πώς το κάθε ένα βελτιστοποιεί μια ιδιαίτερη παράμετρο εις βάρος όλων των άλλων. Το Chebyshev βελτιστοποιεί την ταχύτητα αποκοπής, το Butterworth την επίπεδη μορφή της passband και το Bessel την βηματική απόκριση.

Η επιλογή του φίλτρου antialias εξαρτάται σχεδόν εξ ολοκλήρου από την εφαρμογή, δηλαδή τον τρόπο που μεταφέρονται οι πληροφορίες στα σήματα που σκοπεύουμε να επεξεργαστούμε. Ενώ υπάρχουν πολλοί τρόποι κωδικοποίησης για τις πληροφορίες με ένα αναλογικό σήμα, μόνο δύο μέθοδοι χρησιμοποιούνται ευρέως: η κωδικοποίηση στο πεδίο του χρόνου (**time domain encoding**), και κωδικοποίηση στο πεδίο συχνότητας (**frequency domain encoding**). Η διαφορά μεταξύ αυτών των δύο είναι κρίσιμη στην DSP.

	Voltage gain at DC	Step Response			Frequency Response		
		Overshoot	Time to settle to 1%	Time to settle to 0.1%	Ripple in passband	Frequency for x100 attenuation	Frequency for x1000 attenuation
Bessel							
2 pole	1.27	0.4%	0.60	1.12	0%	12.74	40.4
4 pole	1.91	0.9%	0.66	1.20	0%	4.74	8.45
6 pole	2.87	0.7%	0.74	1.18	0%	3.65	5.43
8 pole	4.32	0.4%	0.80	1.16	0%	3.35	4.53
Butterworth							
2 pole	1.59	4.3%	1.06	1.66	0%	10.0	31.6
4 pole	2.58	10.9%	1.68	2.74	0%	3.17	5.62
6 pole	4.21	14.3%	2.74	3.92	0%	2.16	3.17
8 pole	6.84	16.4%	3.50	5.12	0%	1.78	2.38
Chebyshev							
2 pole	1.84	10.8%	1.10	1.62	6%	12.33	38.9
4 pole	4.21	18.2%	3.04	5.42	6%	2.59	4.47
6 pole	10.71	21.3%	5.86	10.4	6%	1.63	2.26
8 pole	28.58	23.0%	8.34	16.4	6%	1.34	1.66

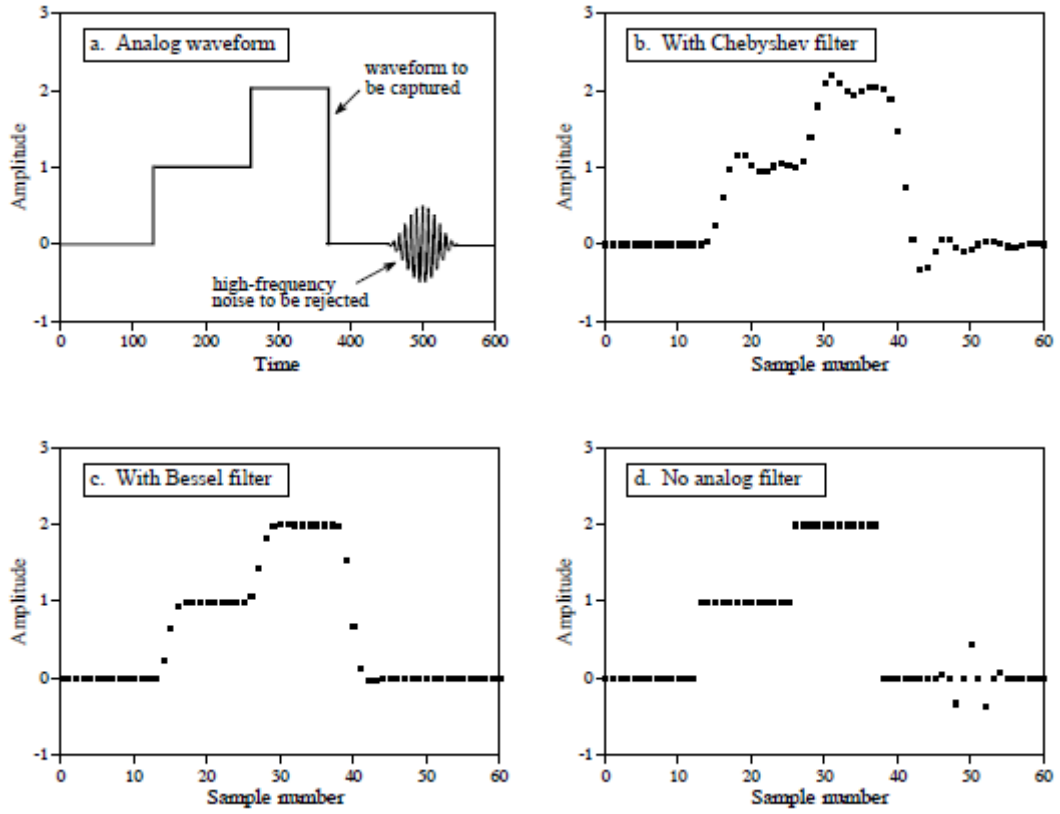
Πίνακας 2.2 Χαρακτηριστικά των τριών βασικών φίλτρων

Στην κωδικοποίηση στο πεδίο συχνότητας, οι πληροφορίες περιλαμβάνονται στα ημιτονοειδή σήματα που συνδυάζονται για να μορφοποιήσουν το τελικό σήμα. Τα ακουστικά σήματα είναι ένα άριστο παράδειγμα. Όταν ένα πρόσωπο ακούει την ομιλία ή τη μουσική, ο αντιληπτός ήχος εξαρτάται από τις συχνότητες που περιλαμβάνει, και όχι από την ιδιαίτερη μορφή της κυματομορφής του. Αυτό μπορεί να φανεί με τη διαβίβαση ενός ακουστικού σήματος μέσα από ένα κύκλωμα το οποίο αλλάζει τη φάση των διάφορων κυματομορφών, αλλά διατηρεί τη συχνότητα και το πλάτος τους. Το σήμα εξόδου φαίνεται τελείως διαφορετικό σε έναν παλμογράφο, αλλά ηχεί το ίδιο. Οι απαιτούμενες πληροφορίες έχουν μείνει άθικτες, ακόμα κι αν η κυματομορφή έχει αλλάξει σημαντικά ως προς το σχήμα της. Δεδομένου ότι το aliasing, εφ' όσον δεν περιορισθεί, επιφέρει αλλοίωση της ταυτότητας και του πλάτους συχνοτήτων, καταστρέφει άμεσα τις πληροφορίες που κωδικοποιούνται στο πεδίο συχνότητας. Συνεπώς, η ψηφιοποίηση αυτών των σημάτων περιλαμβάνει συνήθως ένα φίλτρο antialias με μια απότομη αποκοπή, όπως το Chebyshev, το ελλειπτικό, ή το Butterworth. Αντίθετα, η φτωχή βηματική απόκριση αυτών των φίλτρων δεν επηρεάζει τις πληροφορίες.

Αντίθετα, η κωδικοποίηση στο πεδίο του χρόνου χρησιμοποιεί τη μορφή της κυματομορφής για να αποθηκεύσει τις πληροφορίες. Για παράδειγμα, οι γιατροί μπορούν να ελέγξουν την ηλεκτρική δραστηριότητα της καρδιάς ενός ατόμου ενώνοντας σε αυτόν ηλεκτρόδια (ηλεκτροκαρδιογράφημα ή EKG). Η μορφή του καρδιογραφήματος παρέχει τις απαιτούμενες πληροφορίες για την λειτουργία της καρδιάς. Οι εικόνες είναι ένα άλλο παράδειγμα αυτού του τύπου σήματος. Αντί για μία κυματομορφή που μεταβάλλεται με την πάροδο του χρόνου, οι εικόνες κωδικοποιούν τις πληροφορίες στο σχήμα κυματομορφής που μεταβάλλεται με την απόσταση. Οι εικόνες υλοποιούνται από περιοχές φωτεινότητας και χρώματος, καθώς και τον τρόπο που συνδέονται αυτές μεταξύ τους.

Το θεώρημα δειγματοληψίας είναι μια ανάλυση για το τι συμβαίνει στο πεδίο συχνότητας κατά τη διάρκεια της ψηφιοποίησης. Αυτό το καθιστά ιδανικό να καταλάβουμε την μετατροπή από αναλογικό σε ψηφιακό των σημάτων που έχουν τις πληροφορίες τους κωδικοποιημένες στο πεδίο συχνότητας. Εντούτοις, το θεώρημα δειγματοληψίας προσφέρει μικρή βοήθεια στην κατανόηση του πώς τα κωδικοποιημένα στο πεδίο του χρόνου σήματα πρέπει να ψηφιοποιηθούν.

Το Σχήμα 2.30 παρουσιάζει τις επιλογές για την ψηφιοποίηση ενός σήματος κωδικοποιημένο στο πεδίο του χρόνου. Το γράφημα (a) είναι ένα παράδειγμα αναλογικού σήματος που ψηφιοποιείται. Σε αυτήν την περίπτωση, οι πληροφορίες που θέλουμε να συλλάβουμε είναι η μορφή των τετραγωνικών παλμών. επίσης σε αυτό το σήμα του παραδείγματος συμπεριλαμβάνεται και μια σύντομη ριπή μιας κυματομορφής υψηλής συχνότητας. Αυτό αντιπροσωπεύει έναν θόρυβο ευρείας ζώνης συχνοτήτων, παρεμβολές καθώς και άλλα ανεπιθύμητα φαινόμενα που εμφανίζονται πάντοτε στα αναλογικά σήματα. Τα άλλα γραφήματα επιδεικνύουν πως θα εμφανιζόταν το ψηφιοποιημένο σήμα με την εφαρμογή διαφόρων επιλογών antialias φίλτρων: ένα Chebyshev, ένα φίλτρο Bessel, και χωρίς κανένα φίλτρο.



Σχήμα 2.30 Τρείς επιλογές φίλτρου antialias για σήματα κωδικοποιημένα στο πεδίο του χρόνου

Είναι σημαντικό να γίνει κατανοητό ότι καμία από αυτές τις επιλογές δεν θα επιτρέψει την αναδημιουργία του αρχικού σήματος από τα δεδομένα δειγματοληψίας. Αυτό συμβαίνει επειδή το αρχικό σήμα περιέχει εγγενώς συνιστώσες συχνοτήτων που είναι μεγαλύτερες από το μισό του ρυθμού δειγματοληψίας. Δεδομένου ότι αυτές οι συχνότητες δεν μπορούν να υπάρξουν στο ψηφιοποιημένο σήμα, το αναδημιουργημένο σήμα επίσης δεν μπορεί να

τις περιέχει. Αυτές οι υψηλές συχνότητες προκύπτουν από δύο πηγές: (1) από το θόρυβο και τις παρεμβολές, που θα επιθυμούσαμε να αποβάλουμε, και (2) από τις απότομες μεταβολές στην κυματομορφή, οι οποίες πιθανώς περιέχουν πληροφορίες που θέλουμε να διατηρήσουμε.

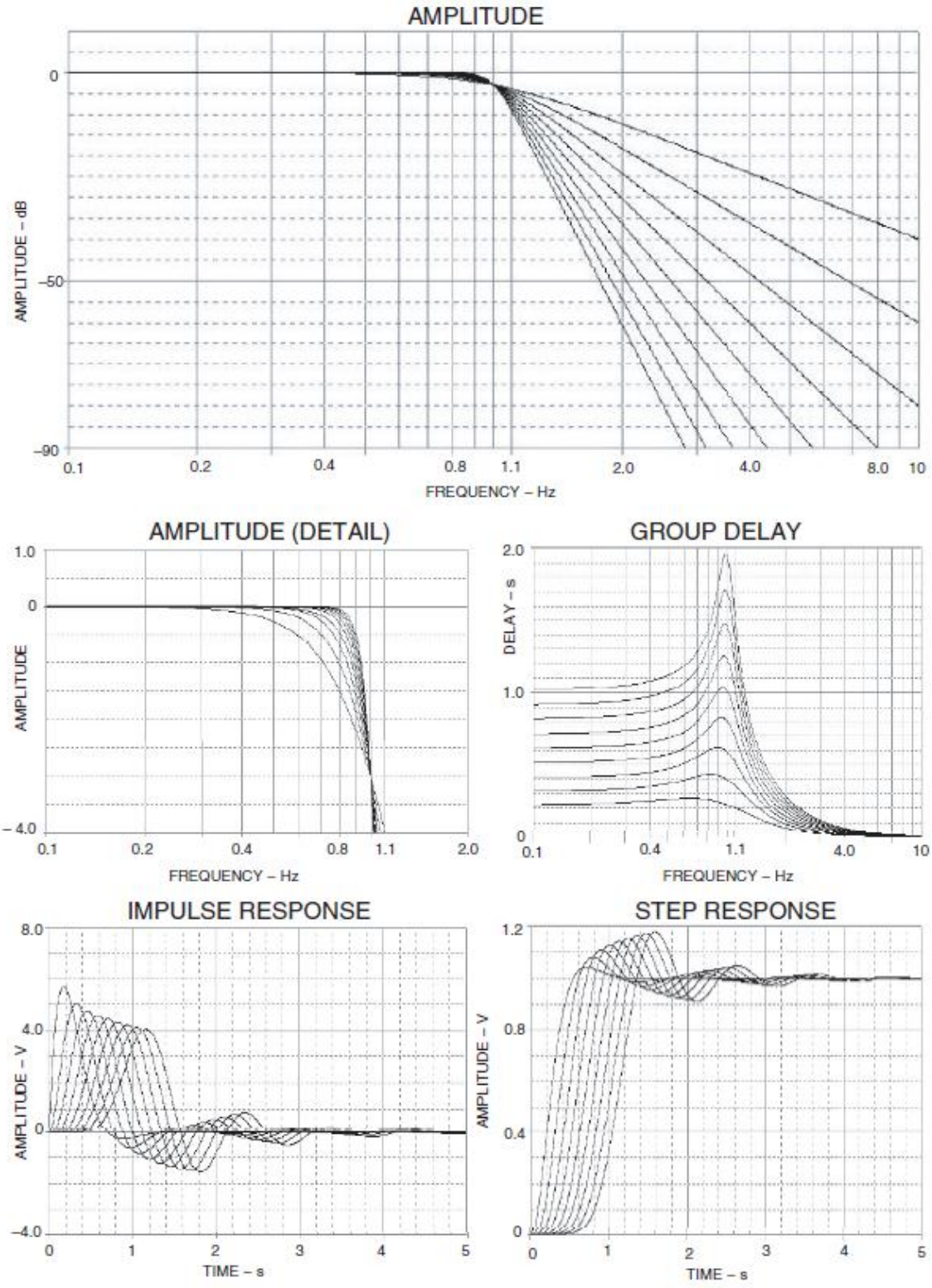
Το φίλτρο Chebyshev που παρουσιάζεται στο (b), αντιμετωπίζει το πρόβλημα επιθετικά, αφαιρώντας όλες τις συνιστώσες υψηλών συχνοτήτων. Αυτό έχει ως αποτέλεσμα να προκύπτει ένα φιλτραρισμένο αναλογικό σήμα που μπορεί να δειγματοληφθεί και στη συνέχεια να αναδημιουργηθεί ιδανικά. Εντούτοις, το αναδημιουργημένο αναλογικό σήμα είναι ίδιο με το φιλτραρισμένο και όχι με το αρχικό σήμα. Αν και τίποτα δεν χάνεται στη δειγματοληψία, η κυματομορφή έχει παραμορφωθεί σημαντικά από το φίλτρο antialias. Όπως φαίνεται στο (b), η θεραπεία είναι χειρότερη από την ασθένεια!

Το φίλτρο Bessel, (c), έχει σχεδιαστεί για αυτό ακριβώς το πρόβλημα. Το αποτέλεσμα που δίνει είναι μια κυματομορφή που μοιάζει πολύ με την αρχική, με μόνο μια μικρή στρογγυλοποίηση των ακρών. Με κατάλληλη ρύθμιση της συχνότητας αποκοπής του φίλτρου μπορούμε να ανταλλάξουμε το βαθμό της ομαλότητας των ακρών με τον αντίστοιχο βαθμό αποκοπής των συνιστωσών υψηλών συχνοτήτων από το σήμα. Η χρησιμοποίηση περισσότερων πόλων στο φίλτρο επιτρέπει μια καλύτερη ανταλλαγή μεταξύ αυτών των δύο παραμέτρων. Μια συνήθης οδηγία είναι να τεθεί η συχνότητα αποκοπής στο ένα τέταρτο περίπου της συχνότητας δειγματοληψίας. Αυτό έχει σαν αποτέλεσμα να έχουμε περίπου δύο δείγματα κατά μήκος κάθε πλευράς ανόδου της κυματομορφής. Παρατηρούμε ότι και το φίλτρο Bessel αλλά και το Chebyshev, έχουν αφαιρέσει την ανεπιθύμητη ριπή του θορύβου υψηλής συχνότητας που υπήρχε στο αρχικό σήμα.

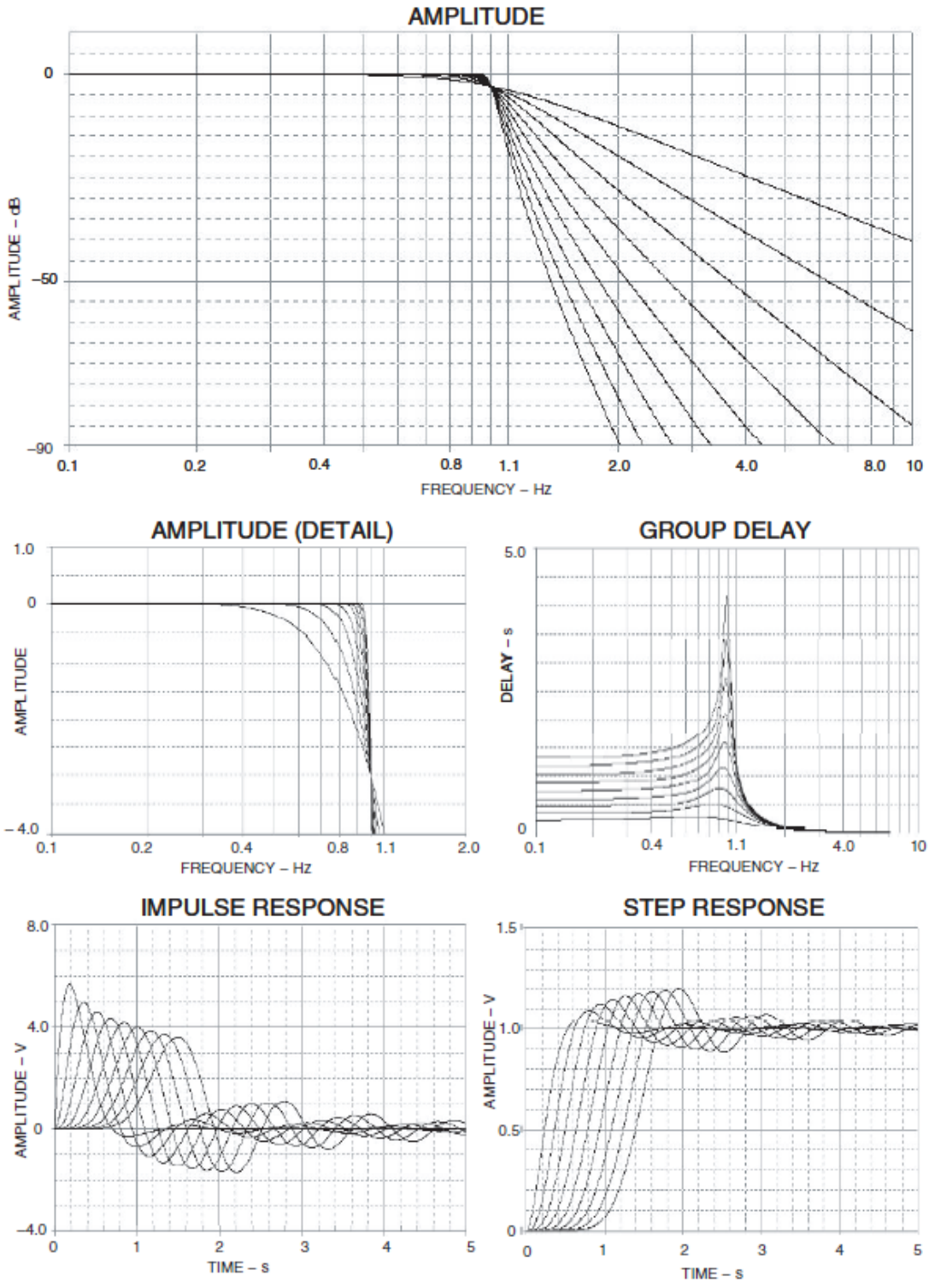
Η τελευταία επιλογή είναι να μην χρησιμοποιήσουμε κανένα antialias φίλτρο, όπως φαίνεται στο (d). Αυτό έχει το ισχυρό πλεονέκτημα ότι η τιμή του κάθε δείγματος είναι ακριβώς ίδια με την τιμή του αρχικού αναλογικού σήματος. Με άλλα λόγια, παρουσιάζει ιδανική οξύτητα των πλευρών, που σημαίνει ότι οποιαδήποτε αλλαγή στο αρχικό σήμα αντικατοπτρίζεται αμέσως στα ψηφιακά δεδομένα. Το μειονέκτημα είναι ότι το aliasing μπορεί να παραμορφώσει το σήμα. Αυτό παρουσιάζεται με δύο διαφορετικές μορφές. Κατ' αρχάς, οι περεμβολές υψηλών συχνοτήτων και ο θόρυβος, όπως η ημιτονοειδής ριπή του παραδείγματος, θα εμφανισθούν ως δείγματα που δεν σημαίνουν τίποτα, όπως φαίνεται στο (d). Δηλαδή οποιοσδήποτε θόρυβος υψηλών συχνοτήτων υπάρχει στο αναλογικό σήμα θα εμφανισθεί σαν παρασιτικός θόρυβος στο ψηφιακό σήμα. Υπό μια γενικότερη έννοια, αυτό δεν είναι ένα πρόβλημα της δειγματοληψίας, αλλά ένα πρόβλημα των αναλογικών ηλεκτρονικών. Δεν είναι ο σκοπός του ADC να μειώσει το θόρυβο και τις παρεμβολές, αυτό είναι ευθύνη των αναλογικών ηλεκτρονικών που προηγούνται της ψηφιοποίησης. Μπορεί να φαίνεται ότι ένα φίλτρο Bessel πρέπει να τοποθετηθεί πριν από τον ψηφιοποιητή για να ελέγξει αυτό το πρόβλημα. Εντούτοις, αυτό σημαίνει ότι το φίλτρο πρέπει να αντιμετωπισθεί ως τμήμα της αναλογικής επεξεργασίας, όχι κάτι που γίνεται προς όφελος του ψηφιοποιητή.

Η δεύτερη εκδήλωση aliasing είναι πιο περίπλοκη. Όταν ένα γεγονός εμφανίζεται στο αναλογικό σήμα (όπως μια απότομη μεταβολή), το ψηφιακό σήμα στο (d) ανιχνεύει την αλλαγή στο επόμενο δείγμα. Δεν υπάρχει καμία πληροφορία στα ψηφιακά δεδομένα για να προσδιορισθεί τι ακριβώς συμβαίνει μεταξύ των δειγμάτων. Τώρα, συγκρίνουμε το αποτέλεσμα μη χρήσης φίλτρου με αυτό της χρήσης ενός φίλτρου Bessel για αυτό το πρόβλημα. Για παράδειγμα, θεωρούμε ότι γράφουμε ευθείες γραμμές μεταξύ των δειγμάτων στο (c). Ο χρόνος κατά τον οποίο αυτή η γραμμή διασχίζει το μισό του πλάτους του βήματος, μας παρέχει μια εκτίμηση υποδειγμάτων του πότε εμφανίστηκε η απότομη μεταβολή στο αναλογικό σήμα. Όταν δεν χρησιμοποιείται κανένα φίλτρο, αυτές οι πληροφορίες των υποδειγμάτων χάνονται εντελώς. Είναι επομένως σημαντικό κάθε φορά να κατανοούμε καλά πώς θα χρησιμοποιήσουμε τα δεδομένα που έχουμε συλλέξει με τη δειγματοληψία.

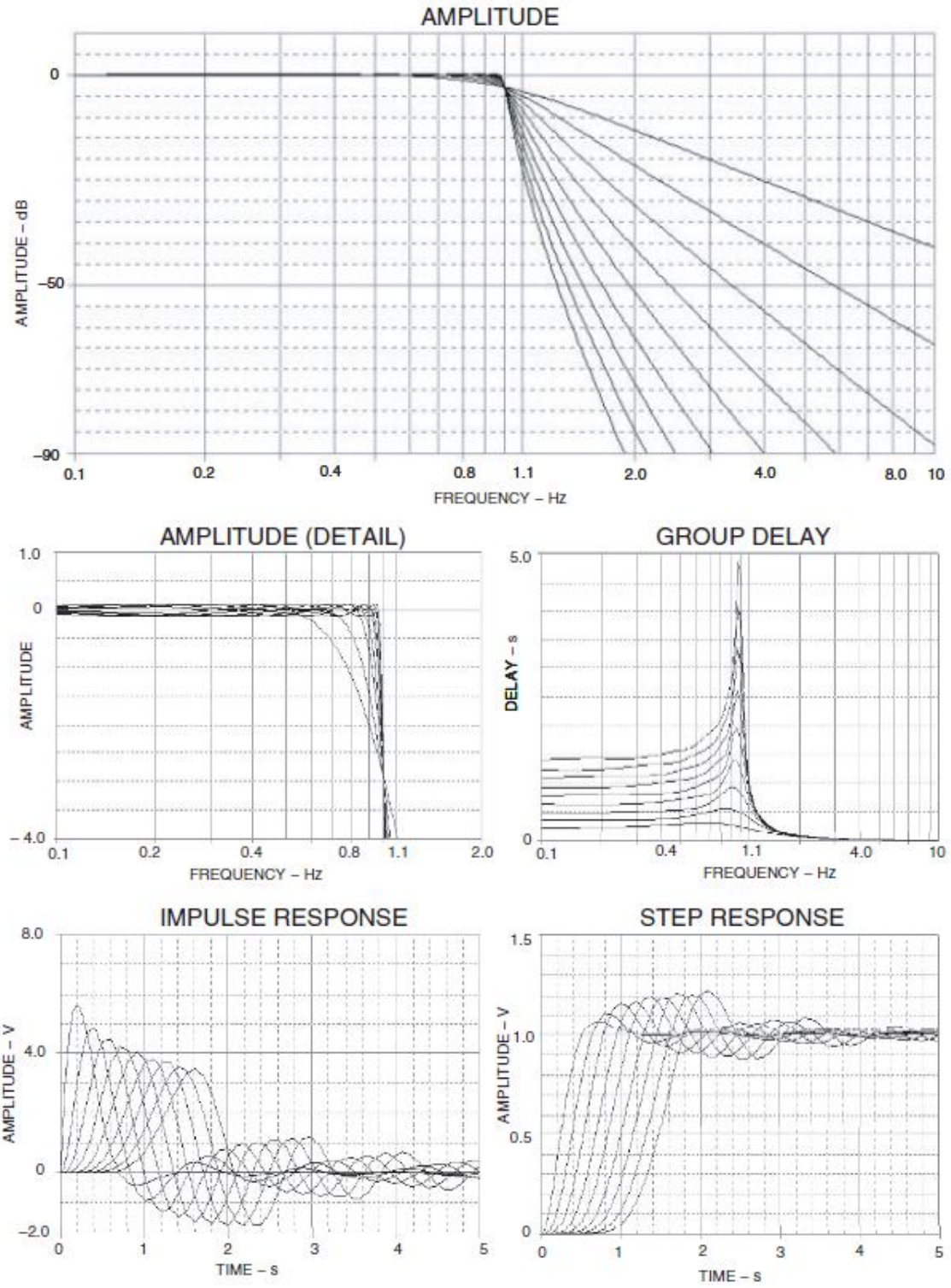
Παρακάτω παραθέτουμε τα λεπτομερή διαγράμματα αποκρίσεων των φίλτρων Butterworth, Chebyshev με κυμάτωση 0.01dB, 0.1dB, 1dB και Bessel. Η συχνότητα αποκοπής είναι κανονικοποιημένη στο 1Hz και οι καμπύλες που παρουσιάζονται σε κάθε διάγραμμα αφορούν τις αποκρίσεις φίλτρων από 2 έως 10 πόλων. Ειδικότερα τα διαγράμματα του φίλτρου Butterworth, θα μας βοηθήσουν στη συνέχεια να προσδιορίσουμε τη συχνότητα δειγματοληψίας του ADC που θα χρειασθούμε σε συνδυασμό με τον κατάλληλο αριθμό πόλων του φίλτρου που θα χρησιμοποιηθεί.



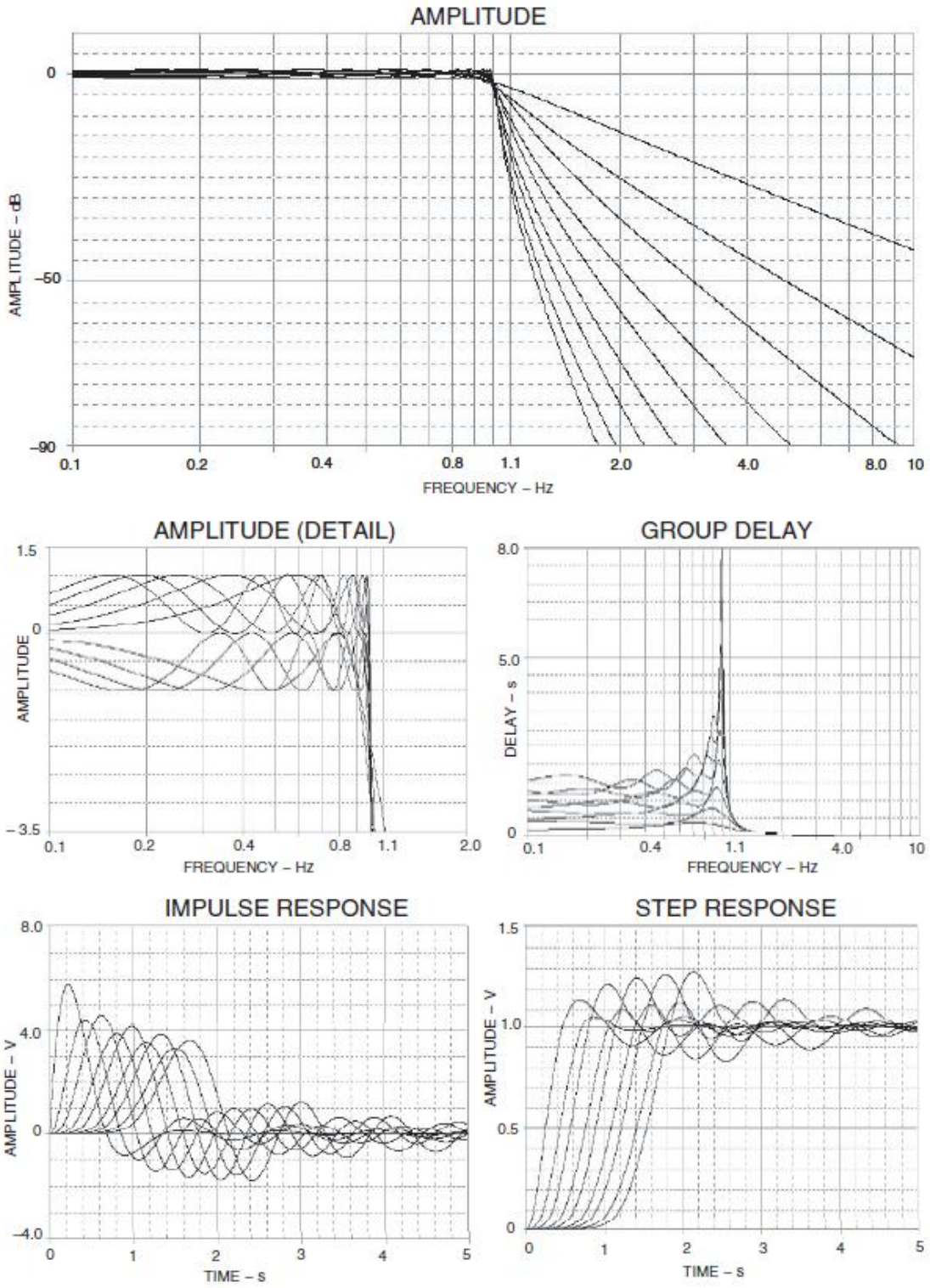
Σχήμα 2.31 Butterworth response



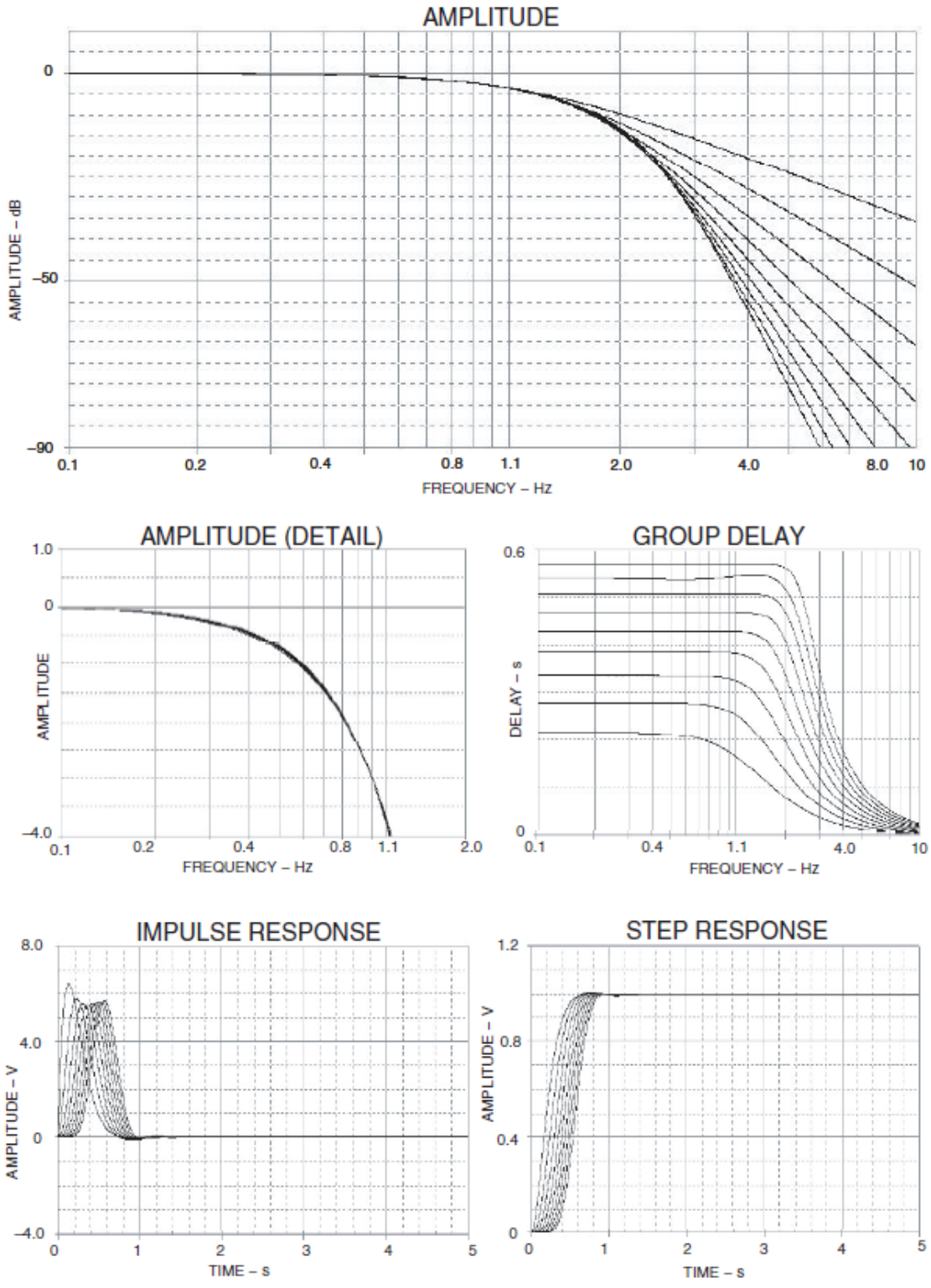
Σχήμα 2.32 Chebyshev 0.01dB response



Σχήμα 2.33 Chebyshev 0.1dB response



Σχήμα 2.34 Chebyshev 1dB response



Σχήμα 2.35 Bessel response

(διαγράμματα σελ.332 από Operational Amplifier Applications Handbook(Newnes)(2004) - Jung)

2.8 Βιβλιογραφία

[1] Steven W. Smith, 1999, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing

[2] Richard G. Lyons, 2011, Understanding Digital Signal Processing, Prentice Hall

[3] Walt Jung, 2005, Op Amp Applications Handbook, Newnes

3. Σχεδίαση Αναλογικής Βαθμίδας και ADC

3.1 Εισαγωγή

Οι εφαρμογές υψηλής ακρίβειας απαιτούν ένα καλά σχεδιασμένο αναλογικό σύστημα που να δίνει το βέλτιστο δυνατό SNR και σε συνδυασμό με τον κατάλληλο ADC θα έχει ως αποτέλεσμα την σωστή και ακριβή σύλληψη των σημάτων διαφόρων αισθητήρων. Η καλή σχεδίαση προϋποθέτει την σωστή επιλογή των επί μέρους εξαρτημάτων που θα συνθέσουν το συνολικό ηλεκτρονικό κύκλωμα. Γενικά, τα βήματα που θα ακολουθήσουμε για το σχεδιασμό και την επιλογή των εξαρτημάτων που θα δώσουν ένα σύστημα χαμηλού θορύβου και κατανάλωσης, είναι τα εξής:

1 Περιγραφή και προσδιορισμός του σήματος εξόδου του αισθητήρα. Στην προκειμένη περίπτωση, ο αισθητήρας μας είναι ένα συγκεκριμένο υδρόφωνο με ενσωματωμένο προενισχυτή.
2 Υπολογισμός των απαιτήσεων του μετατροπέα ADC και επιλογή του κατάλληλου από το εμπόριο.
3 Υπολογισμός των απαιτήσεων και επιλογή των κατάλληλων τελεστικών ενισχυτών για τον σχεδιασμό της ενίσχυσης και του ενεργού φίλτρου.
4 Έλεγχος της λύσης που επιλέχθηκε, ως προς τις προδιαγραφές που θέσαμε.

3.2 Υδρόφωνο και προενισχυτής

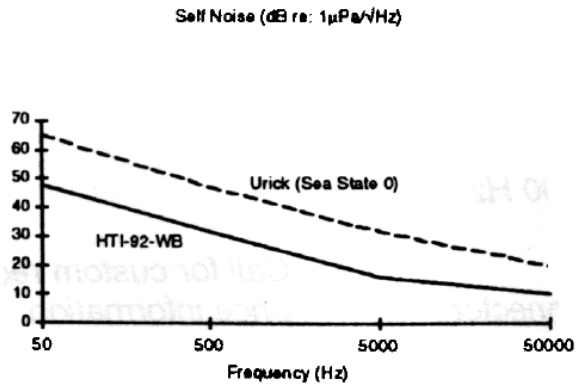
Το υδρόφωνο που θα χρησιμοποιηθεί στην κατασκευή είναι η μονάδα HTI-90-WB της εταιρίας HIGH TECH Inc., που χαρακτηρίζεται ως wideband, ακριβείας, χαμηλού θορύβου και πανκατευθυντικό. Η μονάδα αυτή έχει επίπεδη ευαισθησία λήψης από 100Hz έως 50kHz και είναι πανκατευθυντική μέχρι τα 30kHz. Περιέχει έναν πιεζοηλεκτρικό κεραμικό κύλινδρο που αποτελεί την ενεργή ακουστική μονάδα του μικροφώνου καθώς και έναν προενισχυτή. Τα παραπάνω στοιχεία εμπεριέχονται μέσα σε ένα προστατευτικό περίβλημα πολυουρεθάνης με ενσωματωμένη εσωτερική ηλεκτροστατική προστασία. Το σύστημα μικροφώνου – προενισχυτή μπορεί να οδηγήσει φορτίο 50Ω και λειτουργεί με τάση 12VDC, καταναλώνοντας λιγότερο από 300mW.

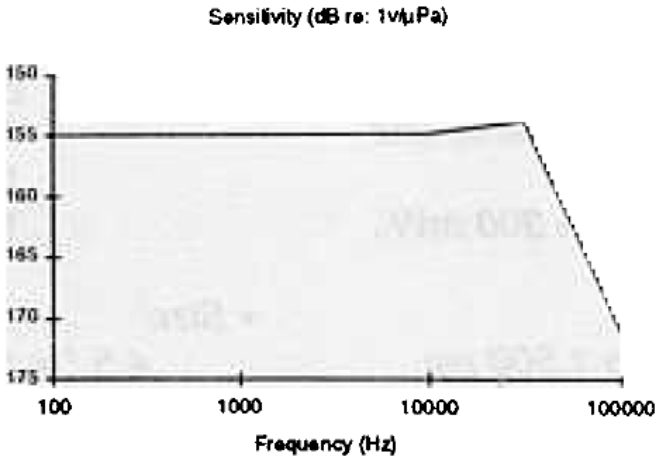
Τα τεχνικά χαρακτηριστικά της παραπάνω μονάδας είναι:

- Ευαισθησια (nominal): -155dB re: 1 volt/μPa typical
- Απόκριση συχνότητας: 100Hz to 50kHz
- Τάση λειτουργίας: 12 ή 24 VDC
- Πίεση λειτουργίας: δοκιμασμένο σε 1500psi
- Βάθος λειτουργίας: 1000μ

• *Self Noise:*
Greater than 10 dB below Sea State "0" (dB re: 1µPa/√Hz)

43 dB @ 100 Hz
27 dB @ 1,000 Hz
15 dB @ 10,000 Hz
12 dB @ 20,000 Hz
10 dB @ 50,000 Hz





Σχήμα 3.1 Διαγράμματα Self Noise και Ευαισθησίας Υδροφώνου

Αρχίζουμε τη σχεδίαση της αναλογικής βαθμίδας λαμβάνοντας υπόψη τα δοθέντα χαρακτηριστικά του υδροφώνου. που στην περίπτωσή μας αποτελεί τον αισθητήρα που προσδιορίζει το σήμα εισόδου της αναλογικής βαθμίδας. Με τη βοήθεια των παραπάνω στοιχείων θα προσδιορίσουμε την ισοδύναμη τάση θορύβου του αισθητήρα καθώς και τον αντίστοιχο σηματοθορυβικό λόγο (SNR), που στη συνέχεια θα χρησιμοποιήσουμε για το σχεδιασμό.

Εξετάζοντας το διάγραμμα θορύβου, βλέπουμε ότι η καμπύλη της φασματικής πυκνότητας του θορύβου παρουσιάζει δύο διαφορετικές περιοχές: την περιοχή των χαμηλότερων συχνοτήτων όπου έχουμε αύξηση του θορύβου αντιστρόφως ανάλογα της συχνότητας (flicker noise ή 1/f noise) και την περιοχή λευκού θορύβου (white noise), όπου η φασματική πυκνότητα θορύβου έχει σχεδόν σταθερή τιμή.

Όπως επίσης βλέπουμε, οι τιμές της φασματικής πυκνότητας θορύβου μας δίνονται σε μονάδες $^{uPa}/_{\sqrt{Hz}}$. Για να συνεχίσουμε, θα πρέπει να μετατρέψουμε τις τιμές αυτές σε μονάδες $^{V}/_{\sqrt{Hz}}$, που είναι η συνήθης μονάδα μέτρησης της φασματικής πυκνότητας θορύβου για τα ηλεκτρονικά στοιχεία. Αυτό γίνεται με χρήση του τύπου[5,p.3]:

$e_{n,dB} = G + P_{sn,dB}$, όπου:

$e_{n,dB}$: η φασματική πυκνότητα της τάσης θορύβου ($dB\ relative\ to\ {}^{V}/_{\sqrt{Hz}}$)

G: η ευαισθησία του υδροφώνου (dB relative to V/uPa).

P_{sn} : η φασματική πυκνότητα της ηχητικής πίεσης θορύβου ($dB\ relative\ to\ {}^{uPa}/_{\sqrt{Hz}}$)

Θεωρώντας ότι η πυκνότητα της ηχητικής πίεσης θορύβου έχει σταθερή τιμή 10dB (διάγραμμα θορύβου), θα προσδιορίσουμε τη συχνότητα f_{nc} , που είναι η συχνότητα στην οποία ο θόρυβος 1/f και ο λευκός θόρυβος, έχουν την ίδια τιμή. Οι τιμές της φασματικής πυκνότητας τάσης θορύβου στα 100Hz και 50kHz, είναι:

$$e_{n,100Hz} = G + P_{sn,100Hz} = -155 + 43 = -112\ dB\ \left(\text{rel to } {}^{V}/_{\sqrt{Hz}}\right) = 10^{-112/20}\ {}^{V}/_{\sqrt{Hz}} = 2{,}5\ {}^{uV}/_{\sqrt{Hz}}$$

$$e_{n,50kHz} = G + P_{sn,50kHz} = -155 + 10 = -145\ dB\ \left(\text{rel to } {}^{V}/_{\sqrt{Hz}}\right) = 10^{-145/20}\ {}^{V}/_{\sqrt{Hz}} = 56\ {}^{nV}/_{\sqrt{Hz}}$$

Θεωρήσαμε ότι $e_{nw} = e_{n,50kHz} = 56\ {}^{nV}/_{\sqrt{Hz}}$

Η ισχύς θορύβου 1/f στα 100Hz δίνεται από τη σχέση[6,p.177]:

$${}^{1}/_{f}\ noise^2\ @\ 100Hz = \left[\left(e_{n,100Hz}\right)^2 - \left(e_{wn}\right)^2\right] * 100Hz =$$

$$\left[\left(2500\ {}^{nV}/_{\sqrt{Hz}}\right)^2 - \left(56\ {}^{nV}/_{\sqrt{Hz}}\right)^2\right] * 100Hz = 6{,}246864 * 10^6\ nV^2$$

Η συχνότητα f_{nc} δίνεται από τη σχέση:

$$f_{nc} = \frac{{}^{1}/_{f}\ noise^2\ @\ 100Hz}{{e_{wn}}^2} = \frac{6{,}246864 * 10^6\ nV^2}{(56\ {}^{nV}/_{\sqrt{Hz}})^2} = 2kHz$$

Θεωρώντας ότι το σύστημα του υδροφώνου έχει απόκριση φίλτρου ενός πόλου, το ισοδύναμο εύρος ζώνης θορύβου είναι[7,p16]:

NPB = 50kHz * 1,57 = 78500Hz

Για το εύρος ζώνης που μας ενδιαφέρει, 100Hz – 50kHz, η συνολική φασματική πυκνότητα τάσης θορύβου, που περιλαμβάνει τον θόρυβο 1/f καθώς και το λευκό θόρυβο, δίνεται από την εξίσωση:

$e_n = e_{wn}\sqrt{1 + \ln(\frac{f_2}{f_1}) * \frac{f_{nc}}{f_2 - f_1}}$, όπου:

$e_{wn} = 56\ {}^{nV}/_{\sqrt{Hz}}$,

f_{nc} = 2kHz,

f_1 = 100Hz,

$f_2 = 78{,}5kHz$.

Η παραπάνω εξίσωση ισχύει με το δεδομένο ότι η f_{nc} που προσδιορίσθηκε, βρίσκεται εντός του φάσματος που μας ενδιαφέρει.

Είναι:

$$e_n = e_{wn}\sqrt{1 + \ln(\frac{f_2}{f_1}) * \frac{f_{nc}}{f_2 - f_1}} = 56\sqrt{1 + \ln(\frac{78500}{100}) * \frac{2000}{78400}} = 60\ {}^{nV}/_{\sqrt{Hz}}$$

Ολοκληρώνοντας στο φάσμα $f_2 - f_1 = 78400$, έχουμε:

$$E_n = e_n\sqrt{f_2 - f_1} = 60\ {}^{nV}/_{\sqrt{Hz}} * \sqrt{78400} = 16{,}8uV_{rms}$$

που είναι η ισοδύναμη τάση του θορύβου στην έξοδο του αισθητήρα μας (υδροφώνου). Το επίπεδο αυτό του θορύβου θα το χρησιμοποιήσουμε ως το μέγιστο επιτρεπτό όριο που θα πρέπει να παρουσιάζει ο θόρυβος των επόμενων βαθμίδων (ενισχυτής, φίλτρο και ADC), έτσι ώστε να μην έχουμε επιπλέον υποβάθμιση του σήματος.

Στη συνέχεια θα προσδιορίσουμε το λόγο SNR του υδροφώνου.

Για να αποκτήσουμε μια καλύτερη εικόνα για την ηχητική πίεση, παραθέτουμε τον παρακάτω πίνακα στον οποίο φαίνονται ενδεικτικές τιμές πίεσης διαφόρων ήχων:

Sound Sources (Noise) Examples with distance	Sound Pressure Level L_p dB SPL	Sound Pressure p N/m^2 = Pa sound field quantity	Sound Intensity I W/m^2 sound energy quantity
Jet aircraft, 50 m away	140	200	100
Threshold of pain	130	63.2	10
Threshold of discomfort	120	20	1
Chainsaw, 1 m distance	110	6.3	0.1
Disco, 1 m from speaker	100	2	0.01
Diesel truck, 10 m away	90	0.63	0.001
Kerbside of busy road, 5 m	80	0.2	0.0001
Vacuum cleaner, distance 1 m	70	0.063	0.00001
Conversational speech, 1 m	60	0.02	0.000001
Average home	50	0.0063	0.0000001
Quiet library	40	0.002	0.00000001
Quiet bedroom at night	30	0.00063	0.000000001
Background in TV studio	20	0.0002	0.0000000001
Rustling leaves in the distance	10	0.000063	0.00000000001
Threshold of hearing	0	0.00002	0.000000000001

Πίνακας 3.1 Πίνακας ενδεικτικών ήχων

Λαμβάνοντας υπόψη τον παραπάνω πίνακα, αποφασίζουμε ότι το μέγιστο πλάτος τάσης στην είσοδο της αναλογικής βαθμίδας (δηλαδή στην έξοδο του μικροφώνου) μας ικανοποιεί εάν είναι V_p = 500mV ή $V_{p\text{-}p}$ = 1V ή V_{rms} = 500/ $\sqrt{2}$ = 353mV_{rms}. Η τάση αυτή, χρησιμοποιώντας την τιμή της ευαισθησίας του υδροφώνου που μας δίνεται, βρίσκουμε ότι αντιστοιχεί σε ηχητική πίεση:

$$\frac{V_{rms}}{G} = \frac{0{,}353\ V_{rms}}{17{,}78 * 10^{-3}\ {V_{rms}}/{Pa}} = 19{,}8\ Pa$$

όπου θέσαμε:

$$G = (10)^{{G_{dB}}/{20}}\ {V}/{uPa} = 10^{-7{,}75} = 17{,}78 * 10^{-9}\ {V}/{uPa} = 17{,}78 * 10^{-3}\ {V}/{Pa}$$

με $G_{dB} = -155\ dB\ rel\ to\ {V}/{uPa}$

Με τον ίδιο τρόπο και λαμβάνοντας υπόψη το κατώφλι θορύβου του συστήματος του υδροφώνου, υπολογίζουμε τον ελάχιστο ήχο που δύναται να καταγραφεί:

$$\frac{E_n}{G} = \frac{16{,}8\ uV_{rms}}{17{,}78 * 10^{-3}\ {V_{rms}}/{Pa}} = 0{,}000945\ Pa$$

με την προυπόθεση ότι ο συνολικός θόρυβος του συστήματος θα παραμείνει σε αυτό το επίπεδο.

Με βάση τα παραπάνω, έχουμε το λόγο σήματος προς θόρυβο του υδροφώνου:

$$SNR_H = 20 * \log\left(\frac{V_{signal,rms}}{E_n}\right) = 20 * \log\left(\frac{0{,}5/\sqrt{2}}{16{,}8 * 10^{-6}}\right) = 86{,}4dB$$

3.3 Επιλογή ADC

Ο αριθμός των bits που θα πρέπει να έχει ο ADC καθορίζεται από το επιθυμητό resolution που θέλουμε να έχουμε σε συνδυασμό με το θόρυβο που υπάρχει στο σήμα. Το μέγιστο πλάτος της τάσης του θορύβου δίνεται από τη σχέση:

$V_{p,noise} = E_n * 3{,}3 = 16{,}8 * 3{,}3 = 55{,}44uV_p$,

όπου 3,3 είναι κατάλληλος συντελεστής (crest factor) για τη μετατροπή της τάσης rms του θορύβου στο μέγιστο πλάτος αυτής[17].

Οποιοδήποτε πλάτος σήματος μεγαλύτερο από το παραπάνω πλάτος της τάσης θορύβου, μπορεί να καταγραφεί από το σύστημα. Για να έχουμε επομένως για τη μέγιστη τάση $V_{p\text{-}p}$ = 1V resolution με $LSB = 55{,}44uV_p$ χρειαζόμαστε:

$$2^n - 1 = \frac{V_{p-p}}{LSB_n} = \frac{1V_{p-p}}{55{,}44uV_p} = 18037\ στάθμες => n = 14{,}1\ bits$$

Επιβεβαιώνουμε το αποτέλεσμα αυτό χρησιμοποιώντας ένα διαφορετικό αλλά πιο ορθολογικό τρόπο: υπολογίζουμε τον αριθμό ENOB (Effective Number of Bits) από το SNR_H που υπολογίσαμε στην προηγούμενη παράγραφο. Ισχύει[2,eq12-15]:

$$SNR_H = 6{,}02 * b_{eff} + 1{,}76 => b_{eff} = \frac{86{,}4 - 1{,}76}{6{,}02} = 14{,}1\ bits$$

που σημαίνει ότι χρειαζόμαστε τουλάχιστον 14,1 bits. Τα παραπάνω μας οδηγούν να επιλέξουμε έναν ADC των 16 bits που δίνει resolution συστήματος:

$$LSB_{16} = \frac{V_{p-p}}{2^{16} - 1} = \frac{1V_{p-p}}{65535} = 15{,}2\ uV_p$$

Το SNR_{ADC} του ADC που θα επιλέξουμε θα πρέπει να είναι καλύτερο από αυτό του σήματος του υδροφώνου, έτσι ώστε να μην υπάρχει παραιτέρω υποβάθμιση της ποιότητας του σήματος λόγω του ADC. Από την άλλη, δεν έχει νόημα να επιλέξουμε έναν ADC με SNR κατά πολύ μεγαλύτερο από αυτό του σήματος. Ενδεικτικά[2,p988] θα θέλαμε έναν ADC με SNR μεγαλύτερο κατά περίπου 6dB από αυτό του σήματος.

Τέλος θα εξετάσουμε τον τρόπο που επιλέγουμε την απαιτούμενη συχνότητα δειγματοληψίας του ADC, σε συνδυασμό με το αριθμό πόλων του antialias φίλτρου που θα προηγηθεί αυτού. Σύμφωνα με τη θεωρία και όπως είδαμε στο προηγούμενο κεφάλαιο, θα πρέπει η ταχύτητα δειγματοληψίας να είναι τουλάχιστον διπλάσια της μέγιστης

συχνότητας που μας ενδιαφέρει να καταγράψουμε (συχνότητα Niquist), που στη δική μας εφαρμογή η συχνότητα αποκοπής που θέλουμε να έχουμε είναι F_c = 50kHz, δηλαδή:

$F_{Niquist}$ = 50kHz = F_c

Αυτό σημαίνει ότι χρειαζόμαστε έναν ADC με συχνότητα δειγματοληψίας τουλάχιστον 100kHz. Το σκεπτικό αυτό όμως αποτελεί μια ιδανική προσέγγιση, καθώς προυποθέτει ότι όλες οι συχνότητες που είναι μεγαλύτερες από 50kHz αποκόπτονται ιδανικά δηλαδή παρουσιάζουν άπειρη εξασθένιση, κάτι που σημαίνει ότι το antialias φίλτρο έχει ιδανική συνάρτηση μεταφοράς. Στην πράξη όμως τα πράγματα δεν είναι τόσο ιδανικά καθώς δεν υπάρχει αναλογικό φίλτρο με τέτοια συμπεριφορά. Έτσι, αν για παράδειγμα εμφανισθεί στην είσοδο του συστήματος μια συχνότητα f_0 = 60kHz που δεν μας ενδιαφέρει και η οποία δεν έχει υποστεί πλήρη εξασθένιση από το φίλτρο, τότε θα προκαλέσει την εμφάνιση μιας ψευδοσυχνότητας (alias)

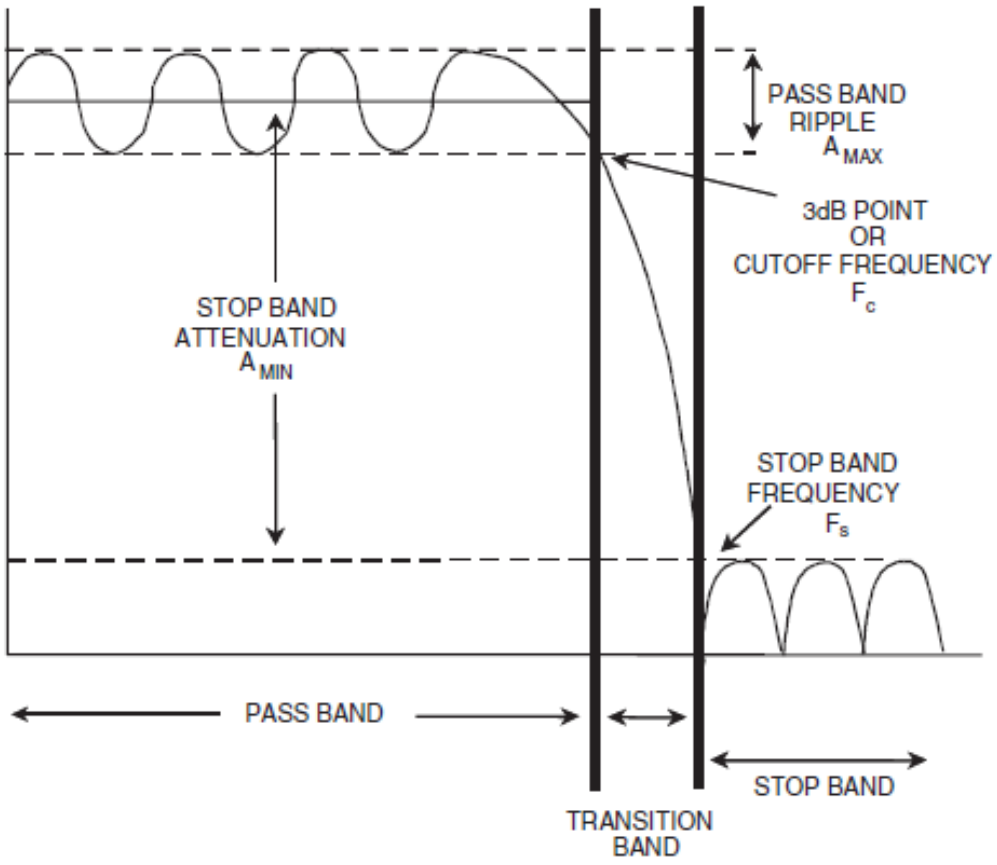
$$f_{alias} = f_0 - f_{sample} = 60\text{kHz} - 100\text{kHz} = -40\text{kHz}$$

η οποία όπως βλέπουμε εμφανίζεται μέσα στο φάσμα που μας ενδιαφέρει (με αντίστροφη φάση). Το ίδιο θα συμβεί για οποιαδήποτε συχνότητα που είναι μεγαλύτερη από την $F_{Niquist}$. Αυτό σημαίνει ότι κάθε συχνότητα μεγαλύτερη από τη συχνότητα Niquist, θα πρέπει να εξασθενίσει επαρκώς πριν εισέλθει στον ADC έτσι ώστε να περιορισθεί η εμφάνιση ψευδοσυχνοτήτων στο φάσμα ενδιαφέροντος. Πόση όμως πρέπει να είναι αυτή η εξασθένιση;

Ιδανικά, η εξασθένιση των συχνοτήτων μεγαλύτερων από τη συχνότητα Niquist θα πρέπει να φέρνει το πλάτος αυτών κάτω από το κατώφλι θορύβου του ADC. Για παράδειγμα, με χρήση της σχέσης που δίνει τον ιδανικό λόγο SNR ενός ADC των 16 bits, έχουμε:

$$SNR_{16bits} = 6.02 * 16 + 1{,}76 = 98.08dB$$

που αντιστοιχεί σε ένα συντελεστή περίπου 80.000 και μας δίνει το μέτρο εξασθένισης (stop band attenuation) που θα πρέπει να εφαρμοσθεί ιδανικά στις συχνότητες μεγαλύτερες της Niquist (stop band frequency F_s) , έτσι ώστε να μην εμφανισθεί καθόλου aliasing. Λαμβάνοντας υπόψη τη συμπεριφορά ενός πραγματικού βαθυπερατού φίλτρου και με δεδομένη τη συχνότητα αποκοπής F_c (cutoff frequency) που χρειαζόμαστε, για να πετύχουμε την επιθυμητή εξασθένιση επιλέγουμε τον κατάλληλο συνδυασμό συχνότητας Niquist (και άρα ρυθμού δειγματοληψίας) και antialias φίλτρου απαιτούμενης τάξης.



Σχήμα 3.2 Χαρακτηριστικές περιοχές και παράμετροι πραγματικών Φίλτρων

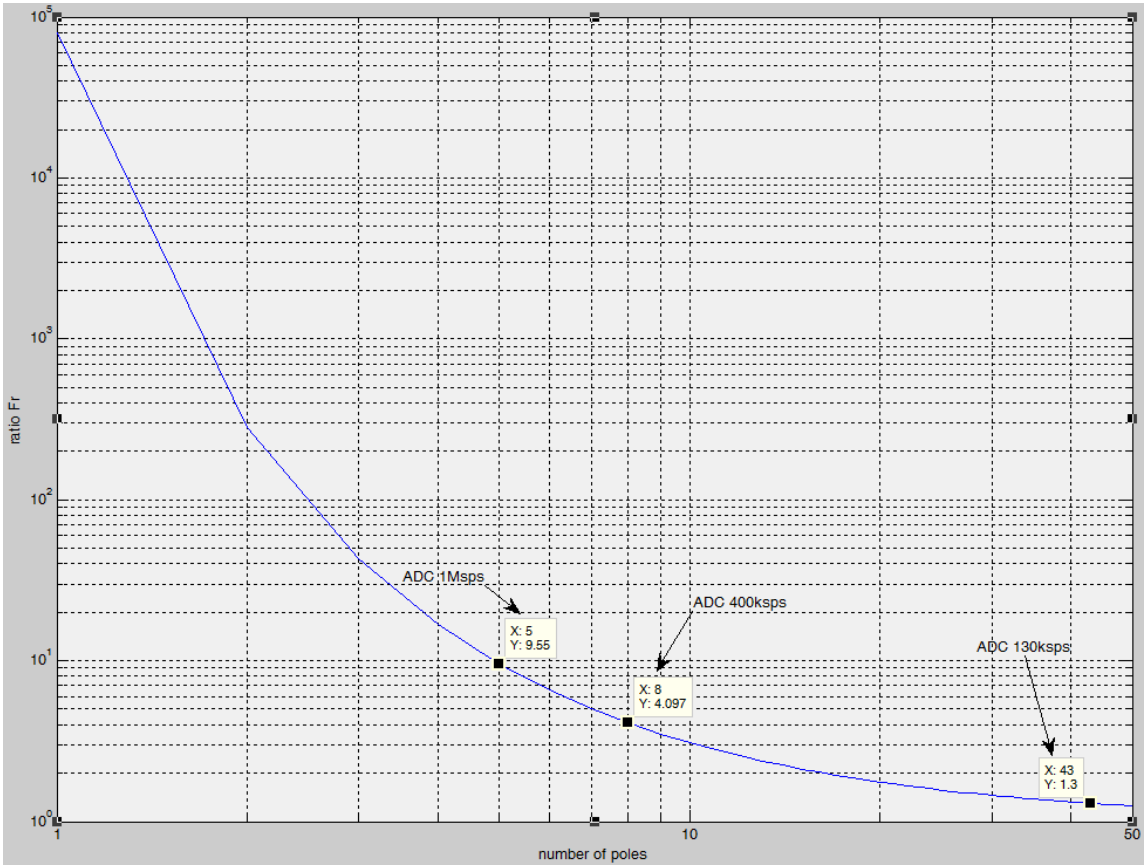
Όπως φαίνεται στο παραπάνω διάγραμμα, το εύρος της ζώνης μετάβασης (transition band) δηλαδή της ζώνης μεταξύ συχνότητας αποκοπής F_c και συχνότητας εξασθένησης F_s, έχει άμεση συνάρτηση με τον αριθμό των πόλων του βαθυπερατού φίλτρου καθώς δηλώνει το πόσο απότομα φθάνουμε στην επιθυμητή εξασθένηση. Οι συχνότητες που περιλαμβάνονται στη ζώνη αυτή αποτελούν ένα φάσμα που δεν είναι χρήσιμο, αφού βρίσκεται πάνω από τη συχνότητα αποκοπής και επίσης, αν δεν επιλέξουμε κατάλληλη συχνότητα F_s (που σημαίνει κατάλληλο ρυθμό δειγματοληψίας) θα συμβάλλει στη δημιουργία ψευδοσυχνοτήτων που θα εμφανισθούν στο φάσμα ενδιαφέροντος.

Ταυτίζοντας τη συχνότητα εξασθένησης F_s με τη συχνότητα Niquist, εκφράζουμε το εύρος της ζώνης μετάβασης με το λόγο[3,p403]:

$$F_r = \frac{F_s}{F_c}$$

Κανονικοποιούμε τον παραπάνω λόγο ως προς συχνότητα F_c = 1Hz όπου η απόκριση είναι 0dB και χρησιμοποιούμε τη θεωρία για τα διαγράμματα Bode. Κάθε πόλος στη συνάρτηση μεταφοράς του φίλτρου προσθέτει κλίση -20dB/decade στην απόκρισή του δηλαδή ένα n-pole φίλτρο (n ο αριθμός των πόλων) προσθέτει -20*n dB/decade στην εξασθένιση. Άρα η συνολική εξασθένιση θα είναι:

$$A = -20 * n * \log(F_r)$$



Σχήμα 3.3 Εύρος ζώνης μετάβασης σε συνάρτηση του αριθμού πόλων του φίλτρου

Αν θεωρήσουμε ότι επιθυμούμε να έχουμε δεδομένη εξασθένιση αυτή που αντιστοιχεί στον ADC των 16bits δηλαδή A = 98.08 dB, χρησιμοποιούμε την προηγούμενη σχέση για να εντοπίσουμε τους συνδυασμούς του n και F_r που μας δίνουν τη δεδομένη εξασθένιση. Για να έχουμε μια πιο συγκεκριμένη εικόνα, θα κατασκευάσουμε το αντίστοιχο διάγραμμα όπου θα απεικονίζεται η μεταβολή του n σε σχέση με το F_r. Από αυτό το διάγραμμα μπορούμε εύκολα να προσδιορίσουμε τις απαιτήσεις μας στον αριθμό των πόλων του antialias φίλτρου που θα χρειασθούμε καθώς και στο ρυθμό δειγματοληψίας του ADC, για τη δεδομένη εξασθένιση και συχνότητα αποκοπής.

Συμπερασματικά, εξετάζοντας ορισμένα ενδεικτικά σημεία του διαγράμματος, βλέπουμε ότι για να πετύχουμε την ιδανική εξασθένιση ενός ADC των 16 bits πρέπει να χρησιμοποιήσουμε:

- συχνότητα δειγματοληψίας 1Msps με φίλτρο 5 πόλων ή
- συχνότητα δειγματοληψίας 400Ksps με φίλτρο 8 πόλων ή
- συχνότητα δειγματοληψίας 130Ksps με φίλτρο 43 πόλων

Στη δική μας εφαρμογή οι απαιτήσεις για τον ADC και το φίλτρο δεν τηρήθηκαν τόσο αυστηρά, καθώς υπήρχαν πολλοί επιπρόσθετοι περιορισμοί (επιλογές εμπορίου, απλότητα συνολικού σχεδιασμού, χαμηλή κατανάλωση, πρωτόκολλο επικοινωνίας με ψηφιακή βαθμίδα κ.ά.)

Με τα δεδομένα που τέθηκαν και κατόπιν έρευνας στο εμπόριο, καταλήξαμε στην επιλογή του MAX1179 της MAXIM. Τα βασικά χαρακτηριστικά αυτού του ADC είναι:

Μέγιστος ρυθμός δειγματοληψίας: 135ksps

Resolution : 16 bits

Κατανάλωση: 23 – 29 mW

SNR: 91dB

SINAD: 90dB

Η εξασθένιση που επιτυγχάνουμε στη συχνότητα Niquist F_s = 67.5 kHz (άρα F_r = 67.5/50 =1.35) με τον συγκεκριμένο ADC και χρήση φίλτρου 8 πόλων, είναι:

$$A = -20 * n * \log(F_r) = -20 * 8 * \log(1.35) = -20.85dB$$

που αντιστοιχεί σε συντελεστή 10.

Όπως βλέπουμε το SNR_{ADC} είναι κατά 5 dB περίπου μεγαλύτερο από αυτό του σήματος του αισθητήρα και επομένως καλύπτεται η απαίτηση για τον ADC να έχει καλύτερο SNR από το σήμα του αισθητήρα.

Το εύρος της τάσης εισόδου του ADC μας επιτρέπει να έχουμε ενίσχυση του σήματος στην αναλογική βαθμίδα κατά 10V/V. Η ενίσχυση του σήματος πρίν από τον ADC, έχει ως πλεονέκτημα να αυξάνεται η διακριτικότητα (resolution) του συστήματος ενώ ταυτόχρονα έχουμε το μειονέκτημα να μειώνεται η δυναμική περιοχή του σήματος εισόδου. Στην δική μας εφαρμογή το εύρος εισόδου του $1V_{pp}$ θεωρείται, όπως είδαμε, ικανοποιητικό, οπότε η ενίσχυση των 10V/V θα μας προσφέρει resolution 15uV.

Όπως προαναφέρθηκε, επιδίωξή μας είναι ο θόρυβος της αναλογικής βαθμίδας και του ADC να διατηρηθεί τουλάχιστον στο επίπεδο του θορύβου που υπολογίσαμε για το σήμα του υδροφώνου. Ως γνωστό, όταν έχουμε πολλαπλές πηγές θορύβου που θεωρούνται ασυσχέτιστες μεταξύ τους (uncorrelated), τότε ο συνολικός θόρυβος προκύπτει ως η τετραγωνική ρίζα του αθροίσματος των τετραγώνων (RSS – Root Sum of Squares) των επι μέρους τάσεων των θορύβων. Δηλαδή, αν E_{n1}, E_{n2} οι τάσεις θορύβου δύο ασυσχέτιστων πηγών, η συνολική τάση θορύβου είναι:

$$E_{n,total} = \sqrt{{E_{n1}}^2 + {E_{n2}}^2}$$

Αυτό έχει ως αποτέλεσμα ο μεγαλύτερος θόρυβος να συνεισφέρει σε μεγαλύτερο βαθμό στο συνολικό θόρυβο, όπως φαίνεται και στον παρακάτω πίνακα:

E_{nout2}/E_{nout1}	E_{nout}/E_{nout1}
1/1	1.414
1/2	1.118
1/3	1.054
1/5	1.020
1/7	1.010
1/10	1.005

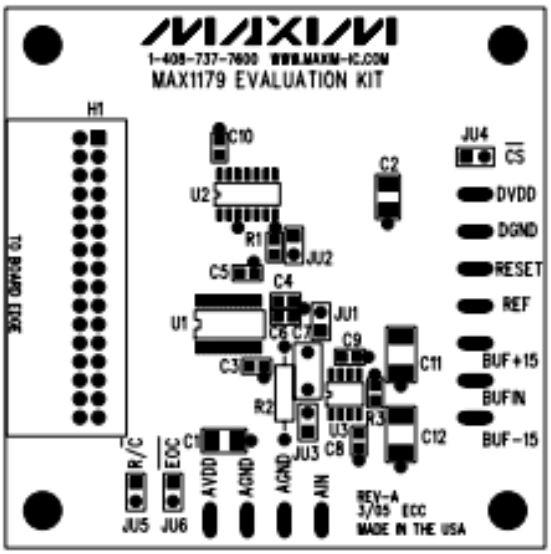
Πίνακας 3.2 Λόγοι επι μέρους τάσεων θορύβου

Από τον πίνακα φαίνεται ότι όσο μικρότερος είναι ο λόγος των δύο επι μέρους θορύβων, τόσο ο συνολικός θόρυβος πλησιάζει την τιμή του μεγαλύτερου.

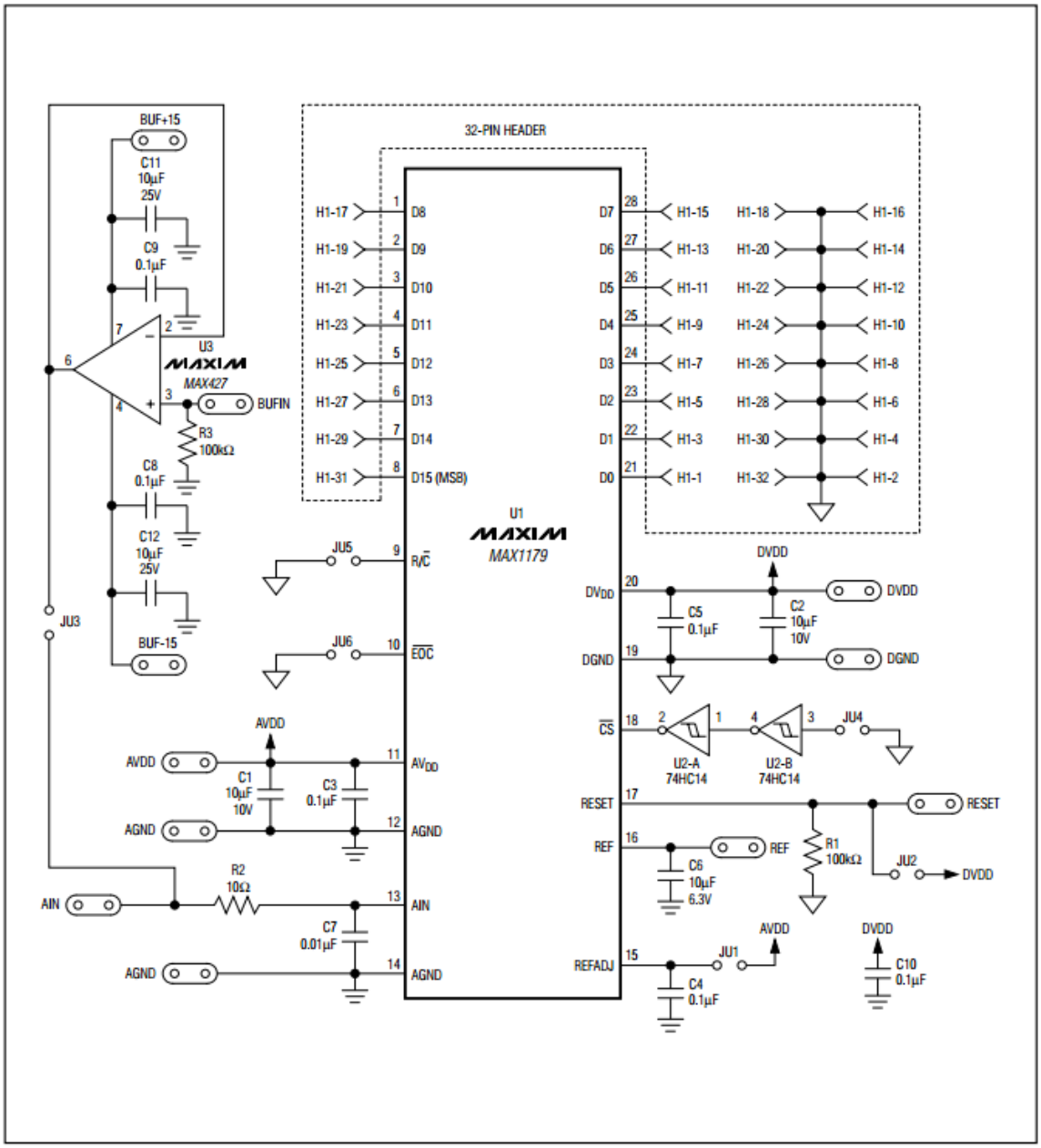
Τα παραπάνω θα μας βοηθήσουν να προσδιορίσουμε το επιτρεπόμενο μέγεθος της τάσης θορύβου της αναλογικής βαθμίδας, έτσι ώστε να μην αυξηθεί σημαντικά ο συνολικός θόρυβος του συστήματος.

3.4 Κυκλωματικό διάγραμμα και PCB του ADC

Για τη διευκόλυνση της κατασκευής, χρησιμοποιήθηκε το evaluation kit του ADC MAX1179 που παρέχει κατασκευασμένο και δοκιμασμένο κύκλωμα του ADC σε πλακέτα:



Σχήμα 3.4 PCB evaluation kit του ADC MAX1179



Σχήμα 3.5 Σχηματικό διάγραμμα του evaluation kit

3.5 Υπολογισμός Θορύβου ADC

Για τον υπολογισμό του θορύβου του ADC χρησιμοποιούμε την τιμή του λόγου SINAD, ο οποίος περιλαμβάνει το θόρυβο και την παραμόρφωση. Είναι:

$$SINAD = \frac{V_{in,rms}}{E_{n,ADC}} => E_{n,ADC} = \frac{3{,}55\ V_{rms}}{10^{90/20}} = 112{,}3\ uV_{rms}$$

3.6 Όριο Θορύβου Αναλογικής Βαθμίδας (Ενισχυτής και Φίλτρο)

Υποθέτουμε ότι ο θόρυβος στην είσοδο της αναλογικής βαθμίδας είναι το άθροισμα των θορύβων αυτής + ο θόρυβος του ADC (χωρίς την ενίσχυση)[18,p3]. Ο συνολικός αυτός θόρυβος (αναλογικής βαθμίδας + ADC) πρέπει να είναι μικρότερος ή το πολύ ίσος με το θόρυβο του σήματος που υπολογίσαμε παραπάνω, έτσι ώστε το σήμα μας να μην υποβαθμίζεται επιπρόσθετα λόγω του self noise των βαθμίδων του ενισχυτή, του φίλτρου και του ADC. Με άλλα λόγια, θα πρέπει να ισχύει η σχέση (πάντοτε υπο μορφή ισχύος):

$$E_{n,RTI}{}^2 \geq E_{n,allowed,RTI}^2 + E_{n,ADC,RTI}{}^2$$

Το θόρυβο του σήματος (που εμφανίζεται στην είσοδο της αναλογικής βαθμίδας) τον υπολογίσαμε πριν ότι είναι:

$$E_{n,RTI} = 16{,}8\ uV_{rms}$$

Για το θόρυβο στην είσοδο του ADC, απαλείφοντας την ενίσχυση που δίνει η αναλογική βαθμίδα, τον μεταφέρουμε στην είσοδο αυτής και έχουμε:

$$E_{n,ADC,RTI} = \frac{E_{n,ADC}}{Gain} = \frac{112{,}3\ uV_{rms}}{10} = 11{,}23\ uV_{rms}$$

Επομένως θα πρέπει:

$$E_{n,allowed,RTI} \leq \sqrt{E_{n,RTI}{}^2 - E_{n,ADC,RTI}{}^2} = \sqrt{16{,}8^2 - 11{,}23^2} = 12{,}5\ uV_{rms}$$

Η παραπάνω τιμή είναι η μέγιστη επιτρεπόμενη τιμή της τάσης θορύβου που θα πρέπει να έχει το στάδιο της αναλογικής βαθμίδας, έτσι ώστε το σήμα να μην επιβαρύνεται με επιπρόσθετο θόρυβο.

3.7 Σχεδίαση Βαθμίδας Ενισχυτή και Φίλτρου

Πριν προχωρήσουμε στη σχεδίαση του σταδίου της αναλογικής βαθμίδας, υπενθυμίζουμε τα βασικά δεδομένα στα οποία έχουμε καταλήξει μέχρι τώρα:

- Το BW (εύρος ζώνης) του σήματος που μας ενδιαφέρει είναι 100Hz – 50kHz
- Το μέγιστο πλάτος σήματος του αισθητήρα (υδροφώνου) τίθεται στα 500mV_p
- ADC input range: $\pm 5V$
- Ενίσχυση 10V/V

Συνοπτικά, οι βασικές επιλογές σχεδίασης που έχουμε είναι:

α. Ενισχυτική βαθμίδα ενός instrumentation amplifier και παθητικό φίλτρο

β. Ενισχυτική βαθμίδα ενός instrumentation amplifier και ενεργό φίλτρο

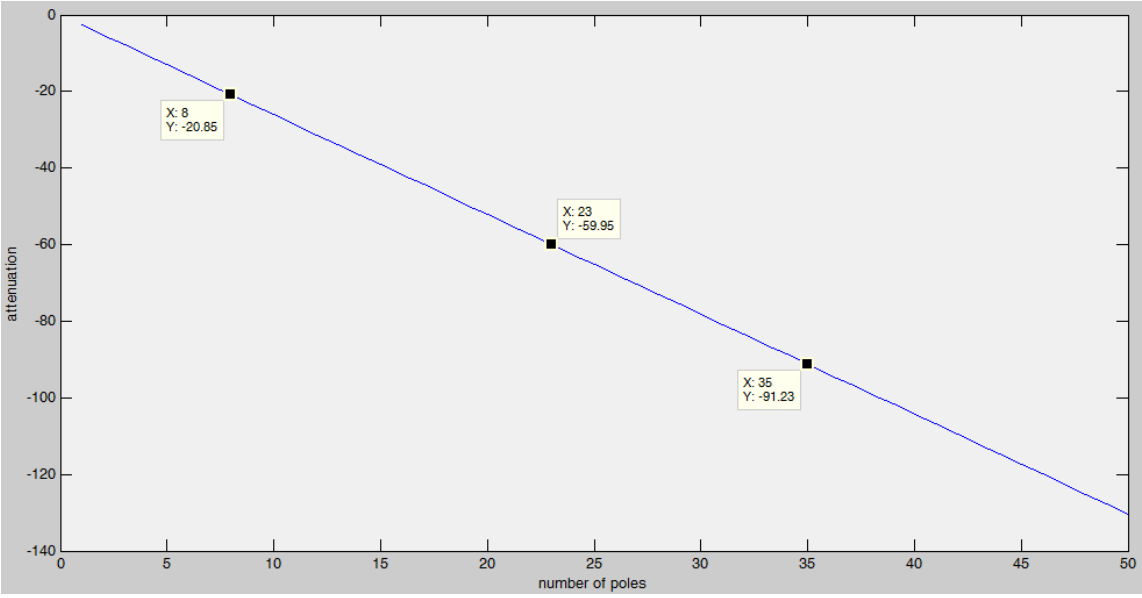
γ. Ενεργό φίλτρο με ταυτόχρονη ενίσχυση.

- Η επιλογή (α) με τη χρήση παθητικού φίλτρου έχει το πλεονέκτημα της απλούστερης σχεδίασης αλλά παρουσιάζει σημαντικά μειονεκτήματα όπως απώλειες του πλάτους του σήματος, ειδικά για φίλτρο τάξης μεγαλύτερης του 2, μεγάλη ευαισθησία στις ανοχές των παθητικών στοιχείων καθώς και πιθανώς πολύ μεγάλη αντίσταση εξόδου με αποτέλεσμα να δυσκολεύει η οδήγηση του ADC, οπότε θα απαιτηθεί και ένας buffer.
- Η επιλογή (β) θα χρειασθεί λιγότερες αντιστάσεις σε σχέση με την (γ), κάτι που σημαίνει μικρότερος θερμικός θόρυβος. Επίσης η ενίσχυση ενός INA ρυθμίζεται πολύ πιο εύκολα απ' ότι η ενίσχυση σε ένα ενεργό φίλτρο.

Σύμφωνα με την ανάλυση που προηγήθηκε στο κεφάλαιο 2 σχετικά με την επιλογή ενός antialias φίλτρου και καθώς η πληροφορία που μας ενδιαφέρει να συλλέξουμε περιέχεται στο πλάτος ηχητικών συχνοτήτων, θα χρησιμοποιήσουμε φίλτρο τύπου Butterworth. Η βασική σχεδίαση υλοποίησης αυτού θα είναι το κύκλωμα Sallen Key, το οποίο έναντι άλλων υλοποιήσεων (Multiple Feed Back, twin-T, Fliege, Akerberg-Mossberg, Biquad, State Variable) παρέχει απλότητα στη σχεδίαση με σχετικά καλή απόδοση, μειωμένο αριθμό παθητικών στοιχείων και τελεστικών ενισχυτών, ενώ όταν έχει gain=1, παρουσιάζει μικρότερη ευαισθησία σε μεταβολές των παθητικών στοιχείων.

Η τάξη του φίλτρου που θα χρησιμοποιήσουμε, όπως είδαμε στην προηγούμενη παράγραφο έχει σχέση με τον αριθμό των bits και τη συχνότητα δειγματοληψίας του ADC. Με χρήση της γνωστής σχέσης που μας δίνει την εξασθένηση που απαιτείται να έχουμε στη συχνότητα Niquist, αλλά με δεδομένη τη χρήση του ADC MAX1179 των 16bits που έχει συχνότητα δειγματοληψίας 135ksps, κατασκευάζουμε το διάγραμμα μεταβολής της εξασθένησης στη συχνότητα Niquist σε σχέση με τον αριθμό των πόλων του φίλτρου, για συχνότητα αποκοπής 50kHz:

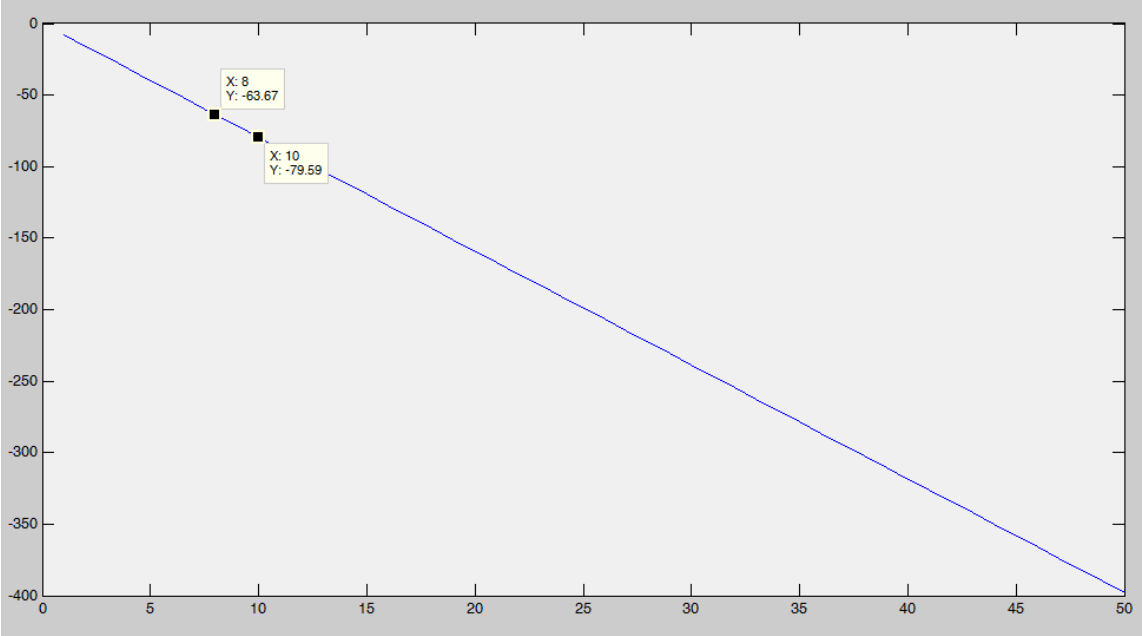
$$A = -20 * n * \log(F_r) => A = -20 * n * \log(1.35)$$



Σχήμα 3.6 Εξασθένιση σε σχέση με τον αριθμό πόλων του φίλτρου για ADC 135ksps

Όπως βλέπουμε από το διάγραμμα, για να έχουμε την ιδανική εξασθένηση των 91dB που απαιτεί ο επιλεγμένος ADC, χρειαζόμαστε φίλτρο 35 πόλων, που σημαίνει χρήση 18 τελεστικών! Μια πιο λογική επιλογή που θα δώσει εξασθένηση 60dB (1000 φορές) απαιτεί φίλτρο 23 πόλων, δηλαδή 12 τελεστικούς.

Χρησιμοποιώντας έναν ADC με ρυθμό δειγματοληψίας 250ksps (LTC1606, LTC2376, LTC2377), παίρνουμε το παρακάτω διάγραμμα:



Σχήμα 3.7 Εξασθένιση σε σχέση με τον αριθμό πόλων του φίλτρου για ADC 250ksps

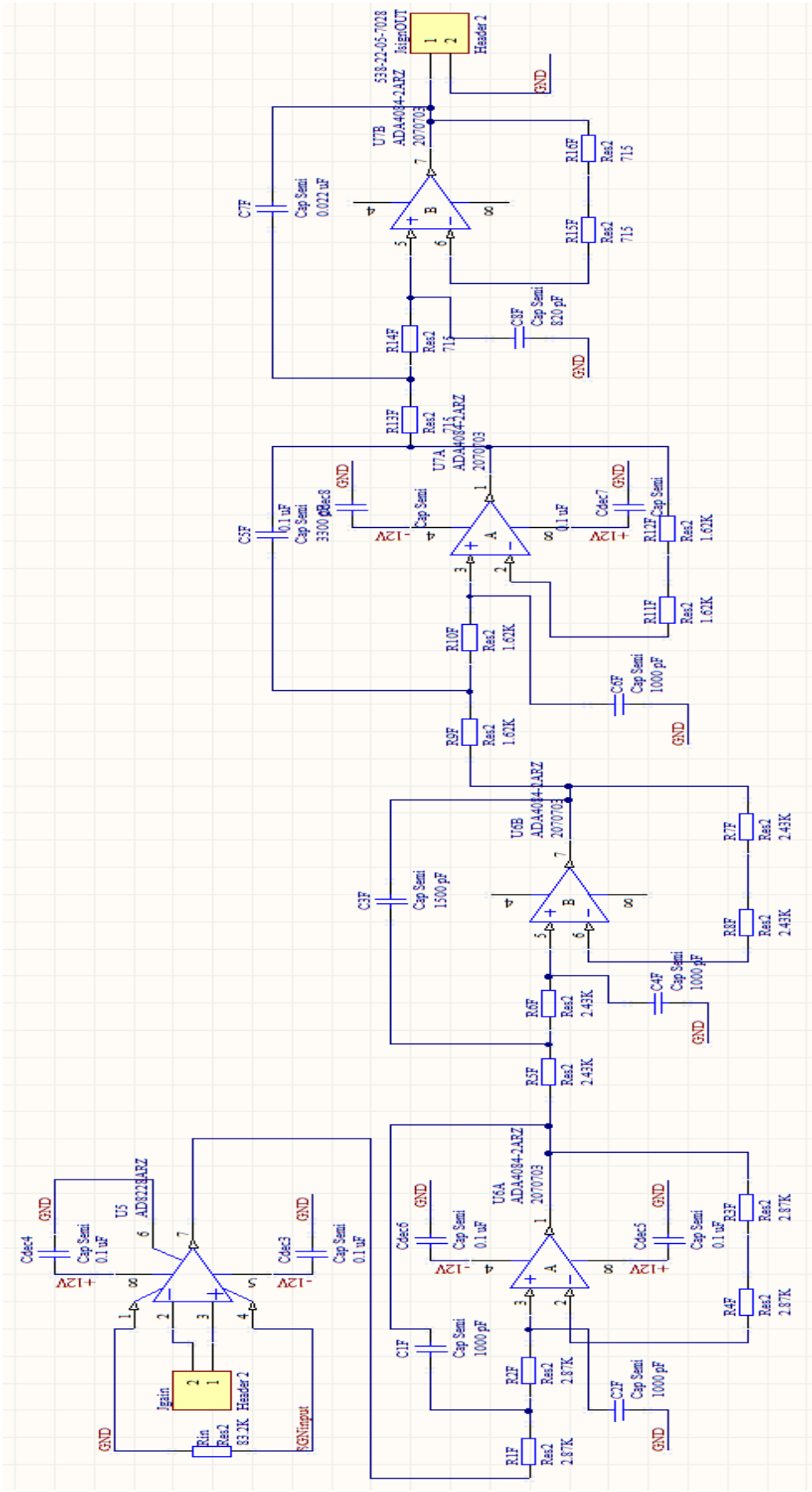
Λόγω των περιορισμών στην κατανάλωση που απαιτεί η εφαρμογή μας και για λόγους απλότητας της κατασκευής, θα χρησιμοποιήσουμε φίλτρο 8 πόλων.

Στην είσοδο του κυκλώματος χρησιμοποιούμε τον instrumentation amplifier AD8228 που δίνει τη συνολική ενίσχυση των 10V/V. Την είσοδο του INA δεν τη χρησιμοποιούμε σε συνδεσμολογία differential αλλά ως single-ended input, καθώς, ούτως ή άλλως, η έξοδος του υδροφώνου έχει ως αναφορά τη γείωση του κυκλώματος και κατά συνέπεια δεν έχει νόημα η διαφορική είσοδος. Η αντίσταση στην είσοδο τίθεται για να δημιουργήσουμε το απαραίτητο dc-bias current return path, όπως απαιτείται από το datasheet για σήματα εισόδου που είναι capacitive coupling.

Σε κάθε στάδιο του φίλτρου χρησιμοποιούμε, το ζεύγος τελεστικών ADA4084-2 που έχουν GBWP 15,9 MHz. Το φίλτρο έχει συνολικό gain = 1V/V και συχνότητα αποκοπής 50kHz. Για τη σχεδίασή του, επιλέξαμε όμοιες τιμές αντιστάσεων για κάθε βαθμίδα. Η αντίσταση στον κλάδο ανάδρασης των τελεστικών, η οποία έχει ίδια τιμή με τη συνολική τιμή της αντίστασης εισόδου, τέθηκε για να αντισταθμήσουμε το ρεύμα πόλωσης εισόδου (input bias current compensation), έτσι ώστε να μην προκαλείται αυξημένο σφάλμα λόγω του offset voltage.

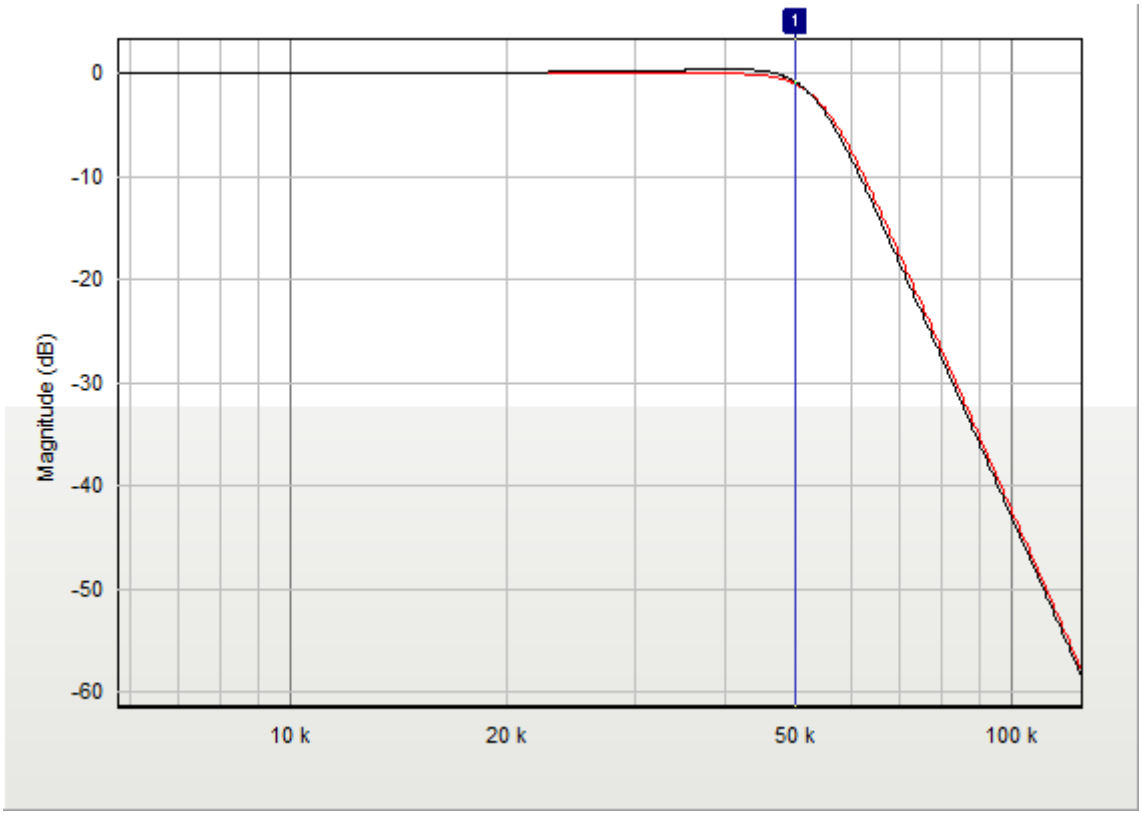
Για το φίλτρο χρησιμοποιήθηκαν αντιστάσεις SMD τύπου thin film, υψηλής ακρίβειας με ανοχή ±0,1% και συντελεστή θερμοκρασίας ±15ppm/^{o}C, καθώς και πυκνωτές multi layer ceramic (MLCC) με ανοχή ±1% και συντελεστή θερμοκρασίας COG (NPO), έτσι ώστε να έχουμε στα χαρακτηριστικά του φίλτρου όσο το δυνατό υψηλή ακρίβεια αλλά και χαμηλή επίδραση αυτών με τη θερμοκρασία.

Το σχηματικό διάγραμμα του κυκλώματος του ενισχυτή και του φίλτρου είναι:



Σχήμα 3.8 Διάγραμμα κυκλώματος Ενισχυτή και Φίλτρου

Η γραφική παράσταση της απόκρισης του φίλτρου, φαίνεται στην εικόνα παρακάτω:



Σχήμα 3.9 Απόκριση Φίλτρου

3.8 Σχεδίαση κυκλώματος τροφοδοσίας (Power Supply Unit)

Οι απαραίτητες τάσεις λειτουργίας για όλες τις βαθμίδες, υλοποιούνται στο ίδιο PCB στο οποίο υλοποιούνται ο ενισχυτής και το φίλτρο. Η μονάδα τροφοδοσίας σχεδιάσθηκε να λειτουργεί με μπαταρία που έχει τάση εισόδου από 4,5V έως 14,5V (nominal 12V). Οι τάσεις εξόδου που δίνει για τις διάφορες βαθμίδες, είναι:

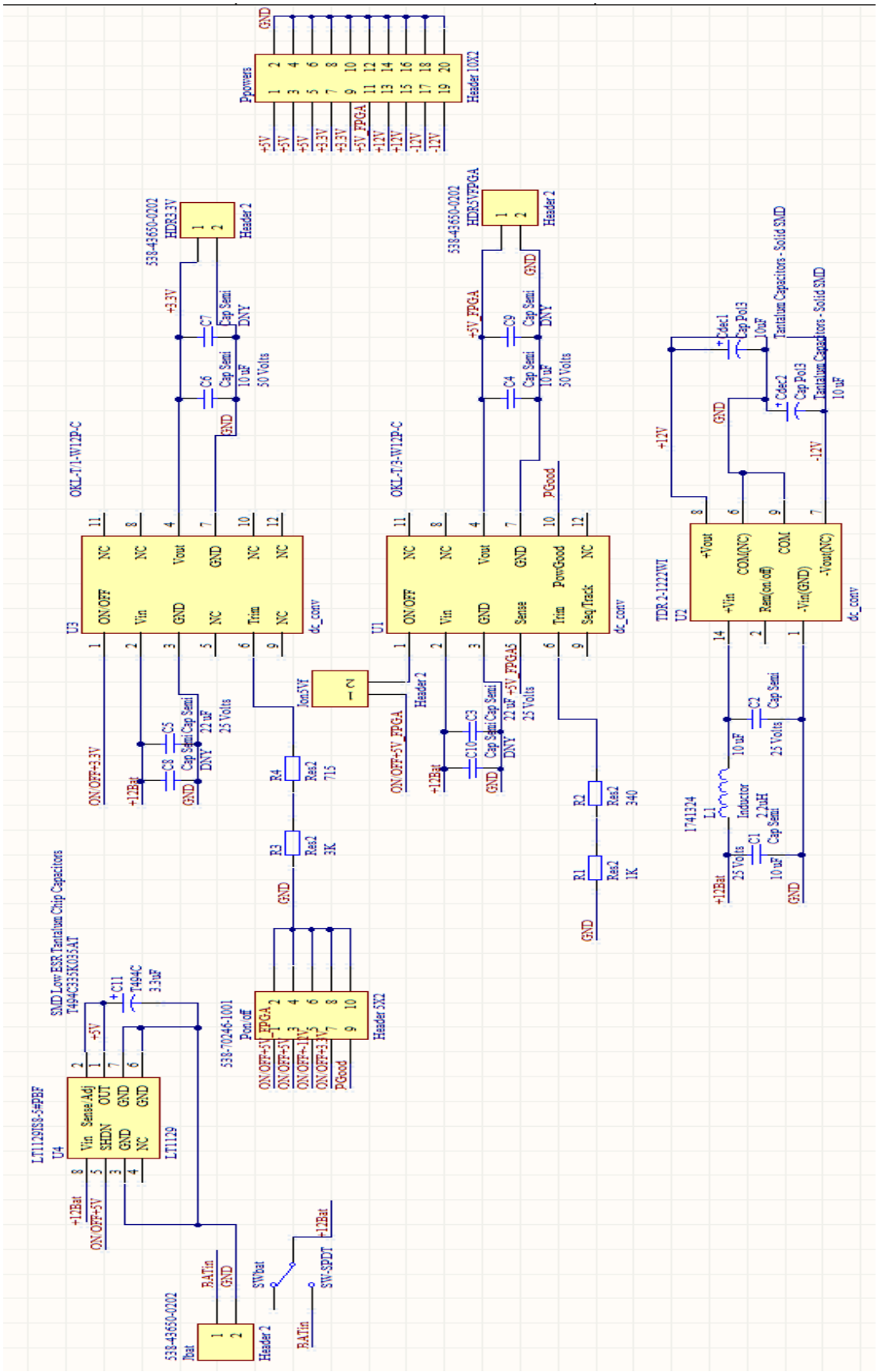
- ±12V για τον ενισχυτή και το φίλτρο
- 12V για το υδρόφωνο
- 5V για τον ADC (analog voltage) και το PCB του FPGA
- 3,3V για τον ADC (digital voltage)
- 5V εφεδρική τάση, για την λειτουργία του ADC με ξεχωριστό converter από το FPGA (στην περίπτωση που επιθυμούμε να απενεργοποιούμε το FPGA αλλά όχι τον ADC.

Για να πάρουμε τις παραπάνω τάσεις από την nominal τάση εισόδου της μπαταρίας, χρησιμοποιήθηκαν επιλεγμένοι DC-DC converters υψηλής απόδοσης, με σχεδίαση buck-

boost, έτσι ώστε να μπορούν να δώσουν τις απαιτούμενες τάσεις εξόδου από μεγάλο εύρος τάσης εισόδου (wide input). Έτσι, αφού προηγουμένως υπολογίσθηκε η απαιτούμενη ισχύς που απαιτεί κάθε βαθμίδα, σύμφωνα με το ρεύμα που καταναλώνει, επιλέχθηκαν:

- TDR 2-1222WI, με input voltage range 4,5 – 18V, output ±12V, typical efficiency 81%, maximum output current ±83mA, power 2 watt.
- OKL-T/3-W12P-C, με input voltage range 4,5 – 14V, programmable output (programmed on 5V), typical efficiency 93%, maximum output current 3A, power 15 watt
- OKL-T/1-W12P-C, με input voltage range 2,9 – 14V, programmable output (programmed on 3,3V), typical efficiency 90%, maximum output current 1A, power 5 watt.
- LT1129IS8-5, Linear, Low Dropout Output regulator, 5V fixed output, 400mV dropout voltage, 700mA maximum output current.

Το διάγραμμα του κυκλώματος τροφοδοσίας , φαίνεται παρακάτω:



Σχήμα 3.10 Διάγραμμα κυκλώματος Τροφοδοσίας

Σύμφωνα με το διάγραμμα, οι τάσεις τροφοδοσίας παρέχονται σε ένα 20-pin header (“Ppower”). Η τάση εισόδου παρέχεται μέσω του “SWbat” single pole – single throw

switch. Επίσης, όλοι οι converters μπορούν να ενεργοποιούνται/απενεργοποιούνται με κατάλληλη τάση (ή ψηφιακή λογική) που εφαρμόζεται στο header "Pon/off". Ο πίνακας λειτουργίας των εισόδων αυτών είναι:

	TDR 2-1222WI	**LT1129IS8**	**OKL-T/3-W12P**	**OKL-T/1-W12P**
ON	open or high Z	open or HIGH	open or HIGH	open or HIGH
OFF	3.3V	0V	0V	0V

Πίνακας 3.3 Τάσεις ON/OFF των DC converters

Τον converter TDR 2-1222WI, αν και είναι isolated I/O, εμείς τον χρησιμοποιούμε με κοινή γείωση εισόδου/εξόδου, για να υπάρχει κοινή αναφορά της εξόδου με τους υπόλοιπους converters που είναι non isolated.

Επίσης αναφέρουμε ότι στη σχεδίαση του κυκλώματος του FPGA, συμπεριλαμβάνονται οι DC-DC converters που απαιτούνται για να πάρουμε όλες τις τάσεις που χρειάζεται το FPGA, από τάση εισόδου 5V. Αυτές οι μονάδες τροφοδοσίας φαίνονται στο κυκλωματικό διάγραμμα του FPGA και περιλαμβάνονται στο αντίστοιχο PCB.

3.9 Υπολογισμός Θορύβου Ενισχυτή και Φίλτρου

3.9.1 Βαθμίδα ενίσχυσης AD8228

Υπολογίζουμε πρώτα τη συχνότητα f_{nc} για την τάση θορύβου. Από το αντίστοιχο διάγραμμα, , προσδιορίζουμε την πυκνότητα της τάσης θορύβου στη χαμηλότερη συχνότητα του διαγράμματος, που για gain G=10, είναι:

$$e_{nf}(1Hz) = 25\text{nV}/\sqrt{Hz}$$

Επίσης από το datasheet, η πυκνότητα τάση του λευκού θορύβου για gain G=10, είναι:

$$e_{wn} = 15\text{nV}/\sqrt{Hz}$$

Η ισχύς θορύβου 1/f στο 1Hz δίνεται από τη σχέση[6,p.177]:

$$^1/_f \; noise^2 \; @ \; 1Hz = \left[\left(e_{n,1Hz}\right)^2 - \left(e_{wn}\right)^2\right] * 1Hz =$$

$$\left[\left(25\,{}^{nV}/_{\sqrt{Hz}}\right)^2 - \left(15\,{}^{nV}/_{\sqrt{Hz}}\right)^2\right] * 1Hz = 400\,nV^2$$

Η συχνότητα f_{nc} δίνεται από τη σχέση:

$$f_{nc} = \frac{^1/_f \, noise^2 \; @ \; 1Hz}{{e_{wn}}^2} = \frac{400\,nV^2}{(15\,{}^{nV}/_{\sqrt{Hz}})^2} = 1{,}8Hz$$

Θεωρώντας για τον INA ότι έχουμε απόκριση φίλτρου ενός πόλου, το ισοδύναμο εύρος ζώνης θορύβου είναι[7,p.16]:

NPBW = 50kHz * 1,57 = 78500Hz

Για το εύρος ζώνης των 50kHz που μας ενδιαφέρει και στην οποία περιλαμβάνεται η f_{nc} που υπολογίσαμε, η συνολική rms τάση θορύβου, που περιλαμβάνει τον θόρυβο 1/f καθώς και το λευκό θόρυβο, δίνεται από την εξίσωση:

$E_n = e_{wn}\sqrt{f_{nc} * \ln(\frac{f_2}{f_1}) + (f_2 - f_1)}$, όπου:

$e_{wn} = 15\ {}^{nV}/_{\sqrt{Hz}}$,

f_{nc} = 1,8Hz,

f_1 = 1Hz,

$f_2 = 78{,}5kHz$.

Είναι:

$$E_{n\_v} = e_{wn}\sqrt{f_{nc} * \ln(\frac{f_2}{f_1}) + (f_2 - f_1)} = 15\sqrt{1{,}8 * \ln\left(\frac{78500}{1}\right) + 78499} = \mathbf{4,2uV_{rms}}$$

που είναι η ισοδύναμη τάση θορύβου του AD8228.

Ομοίως, για το ρεύμα θορύβου και με τη βοήθεια των διαγραμμάτων του datasheet, έχουμε:

$i_{nf}(1Hz) = 200\text{fA}/\sqrt{Hz}$

Επίσης από το datasheet, η πυκνότητα ρεύματος του λευκού θορύβου για gain G=10, είναι:

$i_{wn} = 40\text{f}A/\sqrt{Hz}$

Η ισχύς θορύβου 1/f στο 1Hz:

$${}^{1}/_{f}\ noise^2\ @\ 1Hz = \left[\left(i_{n,1Hz}\right)^2 - (i_{wn})^2\right] * 1Hz =$$

$$\left[\left(200\ {}^{fA}/_{\sqrt{Hz}}\right)^2 - \left(40\ {}^{fA}/_{\sqrt{Hz}}\right)^2\right] * 1Hz = 38400\ fA^2$$

Η συχνότητα f_{nc} δίνεται από τη σχέση:

$$f_{nc} = \frac{{}^{1}/_{f}\, noise^2 \text{ @ } 1Hz}{{i_{wn}}^2} = \frac{38400\, fA^2}{(40\ {}^{fA}/_{\sqrt{Hz}})^2} = 24Hz$$

Έχουμε:

$I_n = i_{wn}\sqrt{f_{nc} * \ln(\frac{f_2}{f_1}) + (f_2 - f_1)}$, όπου:

$i_{wn} = 40\ {}^{fA}/_{\sqrt{Hz}}$,

f_{nc} = 24Hz,

f_1 = 1Hz,

$f_2 = 78{,}5kHz.$

Είναι:

$$I_{n\_i} = i_{wn}\sqrt{f_{nc} * \ln(\frac{f_2}{f_1}) + (f_2 - f_1)} = 40\sqrt{24 * \ln\left(\frac{78500}{1}\right) + 78499} = 11{,}2pA_{rms}$$

που είναι το ισοδύναμο ρεύμα θορύβου του AD8228, το οποίο, πάνω στην εξωτερική αντίσταση των 83,2kOhm, δημιουργεί αντίστοιχη τάση θορύβου:

$\mathrm{E_{n\_i}} = \mathrm{I_{n\_i}} * \mathrm{R} = 11{,}2 * 10^{-12} * 83200 = \mathbf{0,93uV_{rms}}$

Τέλος, ο θερμικός θόρυβος της εξωτερικής αντίστασης R, είναι:

$$E_{n\_R} = 0{,}13 * \sqrt{R} * \sqrt{NPBW} = 0{,}13 * \sqrt{83200} * \sqrt{78500} = \mathbf{10,5uV_{rms}}$$

καθώς σε θερμοκρασία δωματίου, μια αντίσταση R, παρουσιάζει πυκνότητα θορύβου ίση με $0{,}13\mathrm{nV}/\sqrt{Hz}$. Αυτό προκύπτει με την εφαρμογή της σχέσης για το θερμικό θόρυβο αντίστασης (Johnson noise):

$$E_{n\_R} = \sqrt{4 * T_a * K_b * R * NPBW}$$

όπου $\mathrm{T_\alpha}$ = 298K και $\mathrm{K_b}$ = 13,8*10^{-24}.

Συνολικά, η ισοδύναμη τάση θορύβου στην είσοδο του INA, είναι:

$$\boldsymbol{E_{n,INA} = E_{n,RTI} = \sqrt{E_{n\_v}^2 + E_{n\_i}^2 + E_{n\_R}^2} = \sqrt{4,2^2 + 0,93^2 + 10,5^2} = 11,3\mathrm{uV_{rms}}}$$

Εδώ σημειώνουμε ότι, στην περίπτωση που δεν είχαμε την εξωτερική αντίσταση R, μπορούμε να αγνοήσουμε το ρεύμα θορύβου καθώς και το θερμικό θόρυβο της αντίστασης.

3.9.2 Βαθμίδα Φίλτρου

Για τη μελέτη του θορύβου η βαθμίδα αυτή των 4 σταδίων αποτελεί συνολικά ένα ισοδύναμο βαθυπερατό φίλτρο 8 πόλων, καθώς κάθε στάδιο είναι ένα βαθυπερατό φίλτρο 2 πόλων και όλα έχουν την ίδια συχνότητα αποκοπής 50kHz. Άρα το ισοδύναμο εύρος ζώνης θορύβου συμπίπτει με το πραγματικό εύρος ζώνης[7]:

NPB = 50kHz * 1 = 50kHz, καθώς και

f_1 = 1Hz (η μικρότερη συχνότητα που δίνει το διάγραμμα)

Από το datasheet και τα αντίστοιχα διαγράμματα του OPAMP ADA4084, παίρνουμε τα εξής στοιχεία:

f_{nc} = 15Hz η συχνότητα όπου ο λευκός θόρυβος και ο 1/f έχουν την ίδια πυκνότητα

e_{wn} = 3,9 nV/rtHz η φασματική πυκνότητα τάσης λευκού θορύβου

i_{wn} = 0,5pA/rtHz η φασματική πυκνότητα ρεύματος λευκού θορύβου, ενώ δεν υπάρχει ρεύμα θορύβου τύπου 1/f.

Σε όλα τα στάδια, οι rms τιμές της τάσης και του ρεύματος θορύβου προκύπτει το ίδιο και είναι:

$$E_{n\_v} = e_{wn}\sqrt{f_{nc} * \ln(\frac{f_2}{f_1}) + (f_2 - f_1)} = 3{,}9\sqrt{15 * \ln\left(\frac{50000}{1}\right) + 4999} = \mathbf{0,87}\boldsymbol{uV_{rms}}$$

$$I_{n\_i} = 0{,}5\frac{pA}{\sqrt{Hz}} * \sqrt{NPBW} = 0{,}5\frac{pA}{\sqrt{Hz}} * \sqrt{50000} = \mathbf{112}\boldsymbol{pA_{rms}}$$

Για το 1ο στάδιο έχουμε:

Ισοδύναμη τάση θορύβου που προκαλεί το ρεύμα θορύβου πάνω στις R_1 και R_2:

$$\mathrm{E_{n\_i}} = \mathrm{I_{n\_i}} * (R_1 + R_2) = 112 * 10^{-12} * 2 * 2870 = \mathbf{0,64uV_{rms}}$$

Ισοδύναμη τάση θερμικού θορύβου των R_1 και R_2:

$$E_{n\_R} = 0{,}13 * \sqrt{R_1 + R_2} * \sqrt{NPBW} = 0{,}13 * \sqrt{5740} * \sqrt{50000} = \mathbf{2,2uV_{rms}}$$

Ο συνολικός (RTI) θόρυβος του 1ου σταδίου είναι:

$$\boldsymbol{E_{n,1}} = \sqrt{\boldsymbol{E_{n\_v}^2} + \boldsymbol{E_{n\_i}^2} + \boldsymbol{E_{n\_R}^2}} = \sqrt{\mathbf{0,87^2 + 0,64^2 + 2,2^2}} = \mathbf{2,45uV_{rms}}$$

Για το 2ο στάδιο έχουμε:

Ισοδύναμη τάση θορύβου που προκαλεί το ρεύμα θορύβου πάνω στις R_1 και R_2:

$$\mathrm{E_{n\_i}} = \mathrm{I_{n\_i}} * (R_1 + R_2) = 112 * 10^{-12} * 2 * 2430 = \mathbf{0,54uV_{rms}}$$

Ισοδύναμη τάση θερμικού θορύβου των R_1 και R_2:

$$E_{n\_R} = 0{,}13 * \sqrt{R_1 + R_2} * \sqrt{NPBW} = 0{,}13 * \sqrt{4860} * \sqrt{50000} = \mathbf{2uV_{rms}}$$

Ο συνολικός (RTI) θόρυβος του 2ου σταδίου είναι:

$$\boldsymbol{E_{n,2} = \sqrt{E_{n\_v}^2 + E_{n\_i}^2 + E_{n\_R}^2} = \sqrt{0,87^2 + 0,54^2 + 2^2} = 2,25uV_{rms}}$$

Για το 3ο στάδιο έχουμε:

Ισοδύναμη τάση θορύβου που προκαλεί το ρεύμα θορύβου πάνω στις R_1 και R_2:

$$\mathrm{E_{n\_i}} = \mathrm{I_{n\_i}} * (R_1 + R_2) = 112 * 10^{-12} * 2 * 1620 = \mathbf{0,36uV_{rms}}$$

Ισοδύναμη τάση θερμικού θορύβου των R_1 και R_2:

$$E_{n\_R} = 0{,}13 * \sqrt{R_1 + R_2} * \sqrt{NPBW} = 0{,}13 * \sqrt{3240} * \sqrt{50000} = \mathbf{1,65uV_{rms}}$$

Ο συνολικός (RTI) θόρυβος του 3ου σταδίου είναι:

$$\boldsymbol{E_{n,3} = \sqrt{E_{n\_v}^2 + E_{n\_i}^2 + E_{n\_R}^2} = \sqrt{0,87^2 + 0,36^2 + 1,65^2} = 1,9uV_{rms}}$$

Για το 4ο στάδιο έχουμε:

Ισοδύναμη τάση θορύβου που προκαλεί το ρεύμα θορύβου πάνω στις R_1 και R_2:

$$\mathrm{E_{n\_i}} = \mathrm{I_{n\_i}} * (R_1 + R_2) = 112 * 10^{-12} * 2 * 715 = \mathbf{0,16uV_{rms}}$$

Ισοδύναμη τάση θερμικού θορύβου των R_1 και R_2:

$$E_{n\_R} = 0{,}13 * \sqrt{R_1 + R_2} * \sqrt{NPBW} = 0{,}13 * \sqrt{1430} * \sqrt{50000} = \mathbf{1,1uV_{rms}}$$

Ο συνολικός (RTI) θόρυβος του 4ου σταδίου είναι:

$$\boldsymbol{E_{n,4} = \sqrt{E_{n\_v}^2 + E_{n\_i}^2 + E_{n\_R}^2} = \sqrt{0,87^2 + 0,16^2 + 1,1^2} = 1,41uV_{rms}}$$

Για το συνολικό θόρυβο στην έξοδο (RTO) της αναλογικής βαθμίδας (ενισχυτής + φίλτρο), έχουμε:

$G_1 = 10$, η ενίσχυση του INA

$G_2 = 1$, η ενίσχυση όλων των 4 σταδίων του βαθυπερατού φίλτρου

οπότε οι συνιστώσες των επί μέρους τάσεων θορύβου στην έξοδο θα είναι :

$$\boldsymbol{E_{INA,RTO} = E_{n,INA} * G_1 * G_2 = 11{,}3\mathrm{uV_{rms}} * 10 * 1 = 113uV_{rms}}$$

$E_{1,RTO} = E_{n,1} * G_2 = 2{,}45\text{uV}_{\text{rms}} * 1 = \mathbf{2,45uV_{rms}}$

$E_{2,RTO} = E_{n,2} * G_2 = 2{,}25\text{uV}_{\text{rms}} * 1 = \mathbf{2,25uV_{rms}}$

$E_{3,RTO} = E_{n,3} * G_2 = 1{,}9\text{uV}_{\text{rms}} * 1 = \mathbf{1,9uV_{rms}}$

$E_{4,RTO} = E_{n,4} * G_2 = 1{,}41\text{uV}_{\text{rms}} * 1 = \mathbf{1,41uV_{rms}}$

Επομένως ο συνολικός RTO θόρυβος είναι:

$$\boldsymbol{E_{n,RTO} = \sqrt{E_{INA,RTO}^2 + E_{1,RTO}^2 + E_{2,RTO}^2 + E_{3,RTO}^2 + E_{4,RTO}^2}}$$
$$= \sqrt{\mathbf{113^2 + 2,45^2 + 2,25^2 + 1,9^2 + 1,41^2}} = \mathbf{113uV_{rms}}$$

Όπως παρατηρούμε, ο συνολικός θόρυβος που προκύπτει οφείλεται κυρίως στο πρώτο στάδιο του instrumentation amplifier που δίνει και την ενίσχυση της βαθμίδας. Επίσης βλέπουμε ότι ο συνολικός θόρυβος είναι μικρότερος από το όριο που τέθηκε σε προηγούμενη παράγραφο.

Ο συνολικός θόρυβος (σήματος υδροφώνου και αναλογικής βαθμίδας) που φθάνει στην είσοδο του ADC είναι:

$$E_{in,ADC} = \sqrt{(E_s * G1 * G2)^2 + E_{n,RTO}^2} = \sqrt{(16{,}8uV * 10 * 1)^2 + 113uV^2}$$
$$= 202{,}5uV_{rms}$$

και επομένως:

$$\boldsymbol{SNR = 20 * \log\left(\frac{3,55V_{rms}}{202,5 * 10^{-6}V_{rms}}\right) = 84,9dB}$$

που σημαίνει ότι η υποβάθμιση του SNR του σήματος των 86,4dB που υπολογίσθηκε σε προηγούμενο βήμα και οφείλεται στην αναλογική βαθμίδα, είναι πολύ μικρή και ίση με μόλις 2,5dB.

3.10 Βιβλιογραφία

[1] Steven W. Smith, 1999, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing
[2] Richard G. Lyons, 2011, Understanding Digital Signal Processing, Prentice Hall
[3] Walt Jung, 2005, Op Amp Applications Handbook, Newnes
[4] Don Lancaster, 1975, Active Filter Cookbook, Howard W. Sams & Co.
[5] T. B. Straw, 1993, Noise Prediction for Hydrophone/Preamplifier Systems, Naval Undersea Warfare Center Detachment
[6] Ron Mancini and Bruce Carter, 2009, Op Amps for Everyone, Newnes
[7] Art Kay, 2012, Operational Amplifier Noise, Newnes
[8] Jim Karki, 1998, White Paper SLOA011: Understanding Operational Amplifier Specifications, Texas Instruments
[9] Glen Brisebois, Design Note 355: Op Amp Selection Guide for Optimum Noise Performance, LINEAR TECHNOLOGY

[10] Paul Lee, 2011, Application Note 940:Low Noise Amplifier Selection Guide for Optimal Noise Performance, Analog Devices
[11] Texas Instruments, 2002, Application Report SLOA024B:Analysis of the Sallen-Key Architecture, Texas Instruments
[12] Jim Karki, 2002, Application Report SLOA049B: Active Low-Pass Filter Design, Texas Instruments
[13] Kenneth A. Kuhn, 2013, Choosing Resistors and Capacitors for Op-Amp Active Filters, Kenneth A. Kuhn
[14] Analog Devices, 2010, Circuit Note CN-0127: 8-Pole Active Low-Pass Filter Optimized for Precision, Low Noise, and High Gain Using the AD8622 and the ADA4062-2 Op Amps, Analog Devices
[15] Dave Van Ess, What Sallen-Key Articles Don’t tell you (parts I, II, III), analogZONE
[16] Bonnie C. Baker, 1999, Application Note 699: Anti-Aliasing, Analog Filters for Data Acquisition Systems, Microchip
[17] Bonnie C. Baker, 2006, Analog Applications Journal:Matching the noise performance of the operational amplifier to the ADC, Texas Instruments
[18] Reza Moghimi, 2011, Technical Article MS-2022: Seven Steps to Successful Analog to-Digital Signal Conversion (Noise Calculation for Proper Signal Conditioning), Analog Devices
[19] Texas Instruments, 1998, Application Report SLVA043: Noise Analysis in Operational Amplifier Circuits, Texas Instruments
[20] Kumen Blake, 2008, Application Note 1228, Op Amp Precision Design: Random Noise, Microchip
[21] Intersil, 1996, Application Note 519, Operational Amplifier Noise Prediction, Intersil
[22] Analog Devices, 2009, Tutorial MT-048: Op Amp Noise Relationships: 1/f Noise, RMS Noise and Equivalent Noise Bandwidth, Analog Devices
[23] Maxim, 2001, Application Note 748, The ABCs of ADCs: Understanding How ADC Errors Affect System Performance, Maxim
[24] Walt Kester, 2009, Tutorial MT-003: Understand SINAD, ENOB, SNR, THD, THD + N and SFDR so You Don't Get Lost in the Noise Floor, Analog Devices
[25] Walt Kester, 2009, Tutorial MT-001: Taking the Mystery out of the Infamous Formula,"SNR = 6.02N + 1.76dB," and Why You Should Care, Analog Devices

4. Σχεδίαση Ψηφιακής Βαθμίδας

4.1 Εισαγωγή

Η σχεδίαση της ψηφιακής βαθμίδας, θα βασισθεί στη θεωρία της Ψηφιακής Επεξεργασίας Σήματος (DSP) καθώς και στις ειδικές απαιτήσεις και τις προδιαγραφές της εφαρμογής μας. Η βαθμίδα αυτή επιτελεί τις εξής κρίσιμες λειτουργίες:

- Την κατάλληλη επεξεργασία των δεδομένων που έχουν συλλεχθεί
- Την εξαγωγή των επιθυμητών αποτελεσμάτων
- Την διασύνδεση με το χρήστη για τον έλεγχο και διαμόρφωση της λειτουργίας του συστήματος

Η απλούστερη επιλογή για τις παραπάνω απαιτήσεις θα μπορούσε να είναι ένας συνηθισμένος μικροελεγκτής (MicroController Unit – MCU), που σε συνδυασμό με τα απαραίτητα περιφερειακά (μνήμες, ελεγκτές κ.λ.π.), θα μας δίνει τα επιθυμητά αποτελέσματα.

Μια άλλη επιλογή θα μπορούσε να είναι ένας εξειδικευμένος DSP μικροελεγκτής – μικροεπεξεργαστής σε συνδυασμό επίσης με τα απαραίτητα περιφερειακά.

Μια τρίτη επιλογή είναι ένα FPGA, στο οποίο μπορούμε να υλοποιήσουμε έναν επεξεργαστή καθώς και άλλα περιφερειακά που χρειαζόμαστε.

Από τις παραπάνω επιλογές υιοθετήσαμε την τελευταία. Έναντι των άλλων, η υλοποίηση με το FPGA έχει βέβαια το μειονέκτημα της σχετικά αυξημένης κατανάλωσης (που μπορεί όμως να αντιμετωπισθεί), αλλά και πολλά πλεονεκτήματα, όπως:

- υλοποίηση επεξεργαστή που λειτουργεί σε μεγαλύτερη συχνότητα και επομένως, ταχύτερη επεξεργασία των δεδομένων.
- σχεδόν όλα τα περιφερειακά που χρειαζόμαστε, μπορούν να υλοποιηθούν στο ίδιο FPGA με τη τη μορφή λογικής σχεδίασης και τη βοήθεια Hardware Discription Language (VHDL), μειώνοντας έτσι κατά πολύ την πολυπλοκότητα του σχεδιασμού.
- προσφέρει μεγάλα περιθώρια ευελιξίας και μελλοντικών αναβαθμίσεων, τόσο του hardware. όσο και του software ελέγχου του συστήματος.

Ο σχεδιασμός υλοποιήθηκε με βάση ένα FPGA Virtex-5 της Xilinx και με χρήση του πακέτου λογισμικού Xilinx ISE Design Suite, version 12.4.

4.2 Γενική Περιγραφή Υλοποίησης

Η υλοποίηση του hardware και του PCB βασίσθηκε στο ML507 evaluation platform της XILINX. Από το συγκεκριμένο board, κρατήσαμε τα περιφερειακά που χρειαζόμαστε

καθώς και ορισμένα για πιθανές μελλοντικές ανάγκες (USB, ETHERNET), και αφού επανασχεδιάσθηκε το PCB σε CAD, κατασκευάσθηκε το νέο PCB της ψηφιακής βαθμίδας του συστήματος. Το βασικό hardware που περιλαμβάνει αυτή είναι:

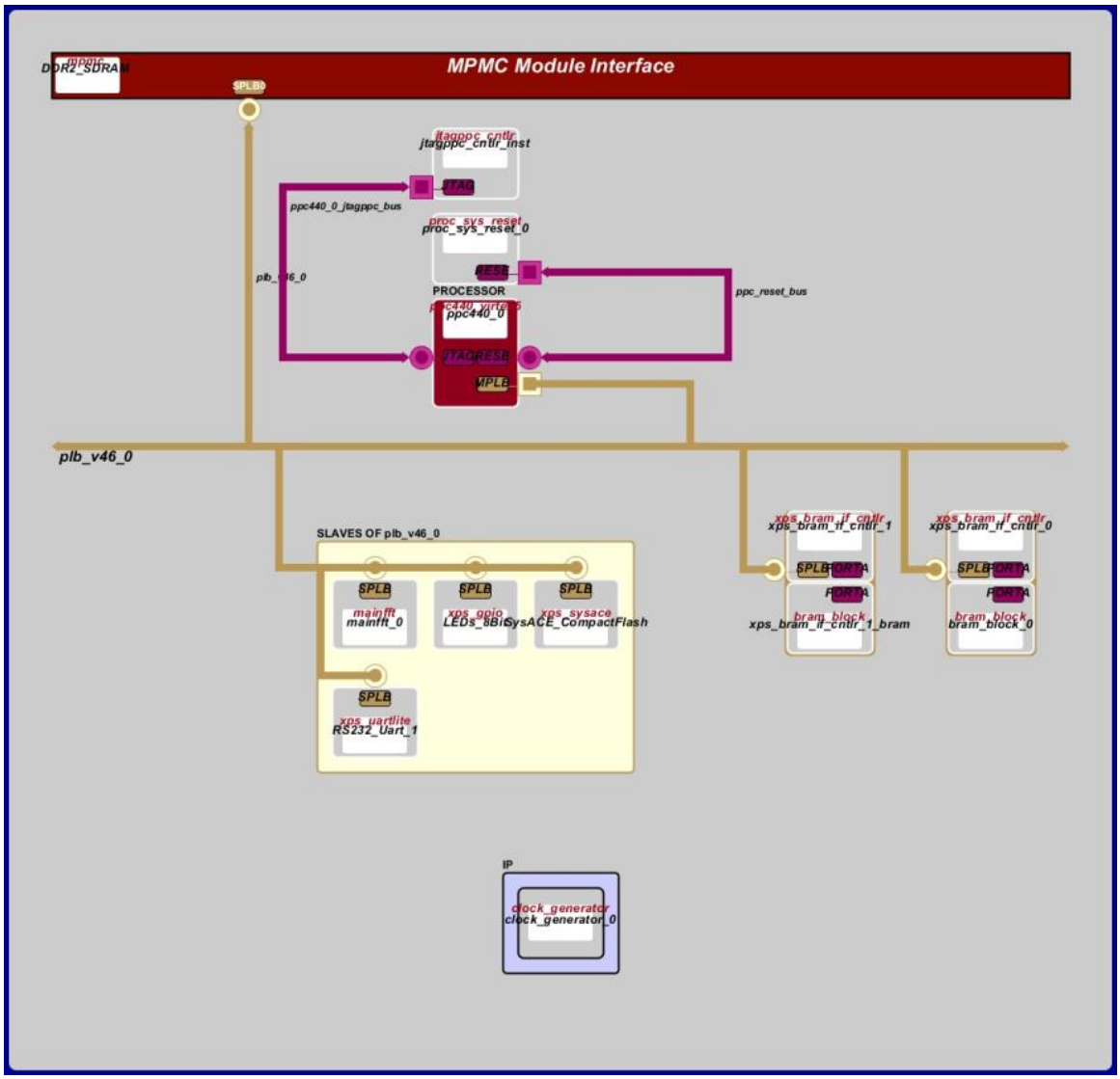
- Xilinx Virtex-5 FPGA XC5VFX70T-1FFG1136
- Xilinx System ACE™ CompactFlash configuration controller with Type I CompactFlash connector
- Xilinx XC95144XL CPLD for glue logic
- 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM)
- Clocking
    - Programmable system clock generator chip
    - One open 3.3V clock oscillator socket
- General purpose DIP switches (8), LEDs (8) and pushbuttons
- Expansion headers with 32 single-ended I/O, 16 LVDS-capable differential pairs, 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion
- RS-232 serial port, DB9 and header for second serial port
- One 8-Kb IIC EEPROM
- 10/100/1000 tri-speed Ethernet PHY transceiver and RJ-45 with support for MII, GMII, RGMII, and SGMII Ethernet PHY interfaces
- USB interface chip with host and peripheral ports
- JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform USB download cable
- Onboard power supplies for all necessary voltages
- MII, GMII, RGMII, and SGMII Ethernet PHY Interfaces
- GTP/GTX: SATA (dual host connections) with loopback cable
- GTP/GTX: Clock synthesis ICs
- BDM debug port
- Soft touch port
- System monitor

Το hardware που υλοποιήθηκε στο FPGA (IP cores) υπο τη μορφή προγραμματιζόμενης λογικής σχεδίασης, είναι:

- PPC440 processor που λειτουργεί σε συχνότητα 400Mhz
- MPMC memory controller για τη μνήμη DDR2
- SYSACE Xilinx controller για τη μνήμη Compact Flash
- UART Lite controller για τη θύρα RS232, που λειτουργεί στα 115200 baud rate
- 2 block RAMs, η μία των 128KB και η άλλη των 64KB καθώς και οι αντίστοιχοι controllers.
- Processor Local Bus (PLB) για την επικοινωνία του επεξεργαστή με τα περιφερειακά
- JTAG controller, για τον προγραμματισμό και το debugging του FPGA
- Processor System Reset module
- Clock Generator module, για την παραγωγή των αναγκαίων σημάτων clock

- mainfft core: είναι το βασικό περιφερειακό user defined, που σχεδιάζεται να περιλαμβάνει όλες τις λειτουργίες των απαιτήσεών μας.

Το block διάγραμμα της σχεδίασης του FPGA φαίνεται παρακάτω:



Σχήμα 4.1 Block διάγραμμα σχεδίασης FPGA

Επίσης, με την εφαρμογή "Software Development Platform - SDK" της πλατφόρμας "Xilinx ISE Design Suite" αναπτύξαμε το απαραίτητο software που τρέχει στον επεξεργαστή και ελέγχει το σύστημα.

4.3 Υλοποίηση Hardware

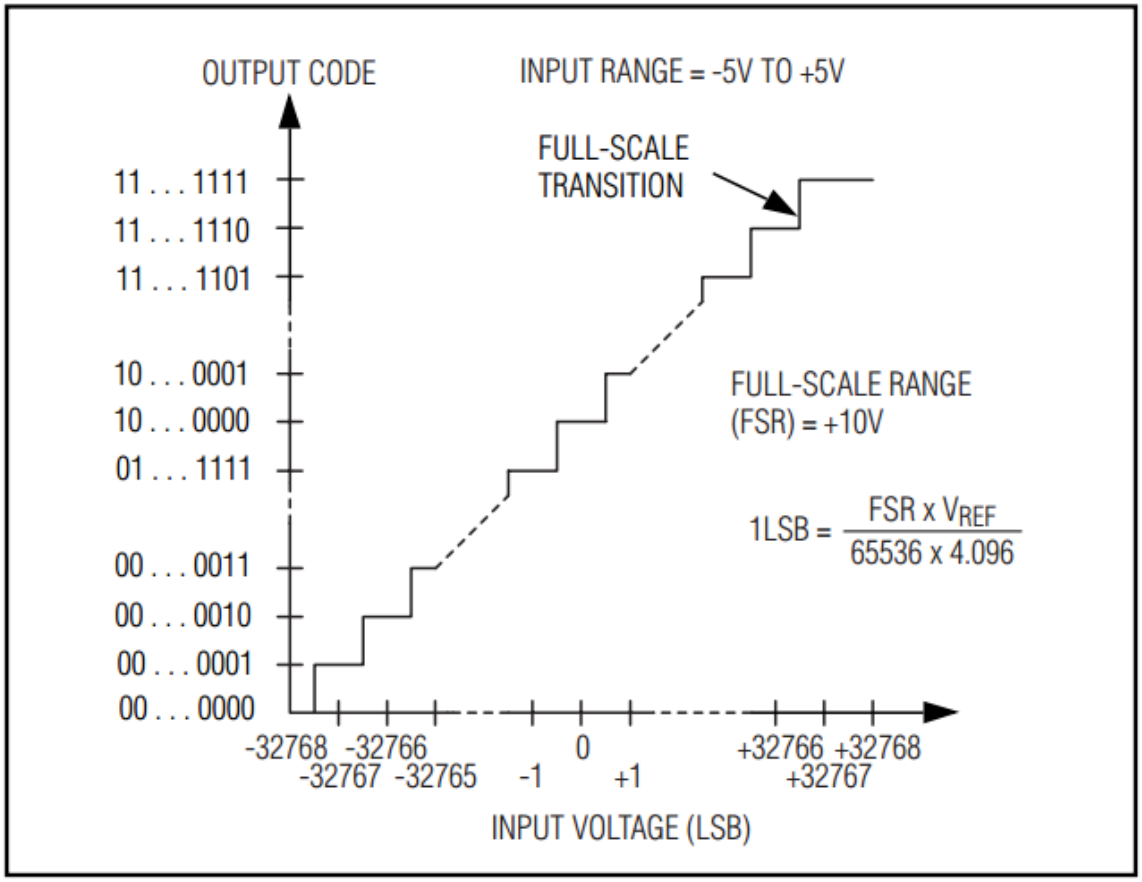
Από τις μονάδες hardware που αναφέρθηκαν παραπάνω και απαιτούνται να υλοποιηθούν στο FPGA, η ποιο σημαντική είναι η "mainfft", η οποία ουσιαστικά περιλαμβάνει όλες τις σημαντικές λειτουργίες που αφορούν τις απαιτήσεις μας. Οι υπόλοιπες μονάδες αφορούν

έτοιμα hardware cores της Xilinx που εισάγονται στη σχεδίαση και ρυθμίζονται οι τυχόν απαραίτητες παράμετροι.

4.3.1 Περιγραφή Λειτουργίας Hardware - Απαιτήσεις

Εδώ θα αναλύσουμε τα βήματα λειτουργίας του συστήματος. Από τη λειτουργία αυτή θα προκύψει και το απαραίτητο Hardware που θα υλοποιήσουμε στο FPGA.

Ανάγνωση Δεδομένων Δειγματοληψίας: Η αρχική φάση της λειτουργίας αφορά την ανάγνωση των ψηφιακών δεδομένων που παράγονται από τον ADC που χρησιμοποιούμε (MAX1179). Ο συγκεκριμένος ADC όπως είδαμε δέχεται bipolar είσοδο των $\pm 5V$ και τα δεδομένα εξόδου είναι σε μορφή offset binary (unsigned integer των 16 bits). Η συνάρτηση μεταφοράς του ADC φαίνεται στο διάγραμμα παρακάτω:



Σχήμα 4.2 Συνάρτηση Μεταφοράς ADC

Με το software ελέγχου καθορίζουμε τη διάρκεια της χρονοσειράς, δηλαδή το χρονικό διάστημα δειγματοληψίας (ή αλλιώς τον αριθμό δειγμάτων που θέλουμε να συλλάβουμε). Τα 16 bits των δεδομένων εξόδου τα λαμβάνουμε όλα μαζί παράλληλα (parallel interface) με τη βοήθεια των σημάτων ελέγχου του ADC. Αυτό σημαίνει ότι, με βάση την περιγραφή λειτουργίας του ADC που αναφέρεται στο datasheet και χρησιμοποιώντας τα σήματα ελέγχου, θα πρέπει να υλοποιήσουμε το κατάλληλο πρωτόκολλο επικοινωνίας έτσι ώστε να λαμβάνουμε τα ψηφιακά δεδομένα δειγματοληψίας. Το πρωτόκολλο αυτό το

υλοποιούμε με VHDL, υπό τη μορφή Μηχανής Πεπερασμένων Καταστάσεων - Finite State Machine (FSM).

Επιπλέον της FSM θα χρειασθούμε εξωτερικές θύρες για την επικοινωνία του FPGA με τον ADC: 16 θύρες για τα δεδομένα και 4 θύρες για τα σήματα ελέγχου.

Αποθήκευση Δεδομένων Δειγματοληψίας: Για την αποθήκευση των δεδομένων δειγματοληψίας χρησιμοποιούμε τη μνήμη DDR2. Η μνήμη αυτή χρησιμοποιείται αποκλειστικά για το σκοπό αυτό, καθώς και για την μετέπειτα προσωρινή αποθήκευση του φάσματος που θα προκύψει από την επεξεργασία FFT. Για το λόγο αυτό, η μνήμη που μας χρειάζεται για άλλες λειτουργίες όπως και η μνήμη όπου φορτώνεται το software, υλοποιείται με πόρους του FPGA (block RAM, distributed RAM).

Η διευθυνσιοδότηση της μνήμη DDR2 γίνεται με τρόπο 4Byte alignment, που σημαίνει ότι γράφουμε ή διαβάζουμε 4 byte και το address offset είναι 2 bits. Η διεύθυνση και τα δεδομένα αυτής είναι σε μορφή Big Endian (στη μικρότερη διεύθυνση το MSB), αλλά αυτό δεν μας απασχολεί όταν μεταφέρουμε δεδομένα σε μια μεταβλητή ή register. Κάθε δείγμα, το οποίο είναι 16 bits, αποθηκεύεται σε μία θέση μνήμης των 32 bit: Τα 16 bits του δείγματος απεικονίζουν το πραγματικό μέρος (σε μορφή offset binary) ενώ τα υπόλοιπα 16 bits τίθενται μηδενικά και απεικονίζουν το φανταστικό μέρος του αριθμού. Ο καθορισμός φανταστικού μέρους απαιτείται διότι το FFT IP core που χρησιμοποιούμε λειτουργεί με μιγαδικές τιμές δειγμάτων.

Σύμφωνα με τα παραπάνω και λαμβάνοντας υπ΄όψη το θεωρητικό ρυθμό δειγματοληψίας των 135ksps του ADC, υπολογίζουμε ότι η χωρητικότητα μνήμης που απαιτείται για κάθε δευτερόλεπτο δειγματοληψίας είναι

135.000 samples/sec * 4 byte/sample = 540.000 bytes/sec

Αρα, η μνήμη των 256MB (268.435.456 bytes) που χρησιμοποιούμε, μπορεί να αποθηκεύσει:

268.435.456 bytes / 540.00 bytes/sec = 497sec = 8,28 minutes

Όταν ολόκληρη η χρονοσειρά των δεδομένων (raw data) μεταφερθεί στην (προσωρινή) μνήμη DDR2, ακολουθεί η επεξεργασία αυτών ανά τμήμα (slice) των 1024 δειγμάτων. Στη φάση αυτή, τα δεδομένα μπορούν να αποθηκευθούν και στη μόνιμη μνήμη flash της CF card, ως αρχείο Fatfs file system.

Υπολογισμός και Αφαίρεση DC συνιστώσας: Αφού υπολογισθεί ο αριθμός των slices που έχει η χρονοσειρά, για κάθε slice υπολογίζεται η DC συνιστώσα και αφαιρείται από όλα τα δείγματα, τα οποία αποθηκεύονται πάλι στην ίδια θέση μνήμης των 32 bit. Η DC συνιστώσα υπολογίζεται ως η μέση τιμή των 1024 δειγμάτων που είναι αρχικά σε μορφή unsigned integer. Η αφαίρεση της DC συνιστώσας, έχει επιπλέον ως αποτέλεσμα την μετατροπή των δεδομένων από offset binary (unsigned integer) σε signed integer των 16 bits: τα 2 LSB απεικονίζουν το προσημασμένο πραγματικό μέρος και τα 2 MSB το προσημασμένο φανταστικό μέρος. Η αριθμητική αναπαράσταση που χρησιμοποιείται για

τους αριθμούς αυτούς, όπως θα δούμε στη συνέχεια, είναι σε Q0.15 format δηλαδή προσημασμένοι σταθερής υποδιαστολής (fixed point) με μηδενικό ακέραιο μέρος.

Επομένως τώρα οι θέσεις μνήμης των 32 bit της DDR2, περιέχουν τα δεδομένα δειγματοληψίας, από τα οποία έχει αφαιρεθεί η DC συνιστώσα και απεικονίζονται το καθένα με 2 προσημασμένους αριθμούς των 16 bits (πραγματικό και φανταστικό μέρος).

Εφαρμογή Παραθύρου Hanning: Μέσω του software, διαβάζουμε τους συντελεστές hanning που βρίσκονται σε αρχείο της CF card και τους εφαρμόζουμε (πολλαπλασιάζουμε) στα δεδομένα της μνήμης DDR2. Η αποθήκευση του αποτελέσματος για κάθε δείγμα γίνεται στην ίδια θέση μνήμης.

Η εφαρμογή του παραθύρου έχει σκοπό τη μείωση του φαινομένου DFT leakage που παρατηρείται όταν το σήμα περιέχει συχνότητες που δεν είναι ακέραιο πολλαπλάσιο της συχνότητας ανάλυσης του μετασχηματισμού DFT.

Μετασχηματισμός FFT : Τα δεδομένα μεταφέρονται ανά 1024 δείγματα (slice) από τη μνήμη DDR2 στη μονάδα FFT, μέσω μιας FIFO (write FIFO), και εκτελείται ο μετασχηματισμός FFT. Ο έλεγχος της λειτουργίας του FFT module επιτελείται με τη βοήθεια μίας Finite State Machine – FSM, που υλοποιήθηκε με VHDL για το σκοπό αυτό.

Τα αποτελέσματα που εξάγονται από τη μονάδα FFT, μέσω μιάς άλλης FIFO (read FIFO) αποθηκεύονται πάλι στις ίδιες θέσεις μνήμης της DDR2 και είναι σε μορφή μιγαδικών, προσημασμένων ακεραίων, με το πραγματικό και φανταστικό μέρος σε αναπαράσταση Q0.15. Από κάθε slice των 1024 δειγμάτων προκύπτουν 1024 συντελεστές (μιγαδικοί) που σύμφωνα με τη θεωρία απεικονίζουν 2 συμμετρικά φάσματα συχνοτήτων. Από αυτά κρατάμε το άνω φάσμα (το σημείο με index 0 που είναι η DC συνιστώσα και τα σημεία με index 513 έως 1023) καθώς, σύμφωνα με τη λειτουργία της μονάδας FFT, αυτό παρουσιάζει χαμηλότερο θόρυβο.

Στη φάση αυτή, μπορούμε να αποθηκεύσουμε τους συντελεστές FFT που προκύπτουν, στη μόνιμη μνήμη flash της CF card, ως αρχείο.

Υπολογισμός Power Spectrum: Για κάθε slice των 1024 δειγμάτων, υπολογίζουμε το μέτρο των 512 μιγαδικών συντελεστών του άνω φάσματος που προέκυψαν από τον μετασχηματισμό FFT και παίρνουμε 512 σημεία που απεικονίζουν το φάσμα ισχύος. Κατά τον υπολογισμό του μέτρου κάθε συντελεστή FFT, πολλαπλασιάζουμε αυτό επί 2, καθώς κρατάμε το μισό του συνολικού φάσματος που προέκυψε από το FFT. Το αποτέλεσμα τώρα του μέτρου κάθε συντελεστή, είναι ένας πραγματικός αριθμός signed integer των 32 bit, σε αναπαράσταση Q.0.30 και αποθηκεύεται στην ίδια θέσης της μνήμης DDR2.

Τα φάσματα αυτά τα αποθηκεύουμε επίσης στη μόνιμη μνήμη flash της CF card και μπορούν να χρησιμοποιηθούν από τον αλγόριθμο ανάλυσης, για να εντοπισθεί π.χ. η ύπαρξη ενός ήχου transient.

Υπολογισμός Mean Power Spectrum: Αφού υπολογίσθηκαν τα φάσματα ισχύος όλων των slices των 1024 σημείων, υπολογίζουμε τη μέση τιμή των φασμάτων αυτών,

παίρνοντας τελικά ένα πίνακα 512 πραγματικών θετικών τιμών των 32bit, σε μορφή Q0.30, που μας δίνει το φασματικό περιεχόμενο ισχύος ολόκληρης της χρονοσειράς. Ο πίνακας αυτός τελικά αποθηκεύεται στη μνήμη CF card.

Όλα τα αποτελέσματα που προκύπτουν στις παραπάνω φάσεις, μπορούν να διαβασθούν και να απεικονισθούν με το Matlab.

Με βάση την παραπάνω περιγραφή των απαιτήσεων λειτουργίας, το "mainfft" περιλαμβάνει:

- ενσωματωμένο το FFT core της XILINX
- Finite state machine για τον έλεγχο του FFT core
- FIFOs απαραίτητες για τη λειτουργία του FFT
- Finite state machine για τον έλεγχο του ADC
- 5 Slave Registers για τον έλεγχο του συστήματος μέσω του software
- Εξωτερικές θύρες για τη διασύνδεση με τον ADC και με τυχόν άλλα εξωτερικά περιφερειακά

4.3.2 Αριθμητική Αναπαράσταση

Σημαντική επιλογή κατά την υλοποίηση του hardware αποτελεί ο τρόπος της απεικόνισης των δεδομένων, καθώς αυτός καθορίζει πολλές παραμέτρους του συστήματος: την πολυπλοκότητα των υπολογιστικών διαδικασιών, την ταχύτητα επεξεργασίας, την ακρίβεια των αποτελεσμάτων κ.α.

Για το εν λόγω σύστημα, επιλέξαμε να χρησιμοποιήσουμε αριθμούς σταθερής υποδιαστολής (fixed point numbers), κυρίως για τους εξής λόγους:

- παρέχουν σταθερό resolution, σε αντίθεση με τους αριθμούς κινητής υποδιαστολής,
- η διαχείρισή τους απαιτεί απλούστερο hardware, μικρότερη υπολογιστική ισχύ και άρα μικρότερη κατανάλωση
- ο σχεδιασμός μπορεί να υλοποιηθεί σε ολοκληρωμένα που δεν διαθέτουν δυνατότητες αριθμητικής κινητής υποδιαστολής

Το κόστος για τη χρήση αριθμών fixed point είναι ο περιορισμός στο dynamic range και το resolution.

Αρχίζοντας από τον ADC, τα δεδομένα που παράγει βρίσκονται σε offset binary format των 16 bits. Οι αριθμοί αυτοί αντιμετωπίζονται ως αριθμοί unsigned binary, μέχρι τη στιγμή που θα αφαιρέσουμε το λεγόμενο DC bias, δηλαδή το offset. Μετά την αφαίρεση του offset, έχουμε τους αριθμούς στη γνωστή μορφή 2's complement, δηλαδή προσημασμένοι δυαδικοί, και έτσι χρησιμοποιούνται στη συνέχεια.

Τους παραπάνω αριθμούς που είναι σε συμπλήρωμα ως προς 2 (2's complement), θεωρούμε ότι είναι προσημασμένοι δεκαδικοί με μηδενικό ακέραιο μέρος δηλαδή

χρησιμοποιούμε την αναπαράσταση Q0.15 (για τους 16bit) και Q0.30 (για τους 32 bit). Η αναπαράσταση αυτή γενικά μας βολεύει, καθώς οι αριθμοί αντιπροσωπεύουν τιμές στο διάστημα [-1,1] και το γινόμενό τους χωρά πάντοτε σε ένα 32bit register, καθώς για τον πολλαπλασιασμό δύο αριθμών A=0.15 και B=0.15 που βρίσκονται σε Q-format, έχουμε:

A*B = 0.15+15 = 0.30

Την ίδια αναπαράσταση Q0.15 χρησιμοποιούμε και για τους συντελεστές του Hanning window ή οποιοδήποτε άλλο window εφαρμόσουμε στα δεδομένα δειγματοληψίας..

4.3.3 Υλοποίηση του FFT IP core

Η πιο σημαντική ίσως μονάδα της ψηφιακής βαθμίδας του συστήματος η οποία και απαιτεί την μεγαλύτερη υπολογιστική ισχύ, είναι η μονάδα που εκτελεί τον αλγόριθμο Fast Fourier Transform – FFT. Για την υλοποίηση αυτής χρησιμοποιήθηκε το εργαλείο "core generator" που περιέχεται στην πλατφόρμα "Xilinx ISE Design Suite",

Το FFT core της Xilinx, υλοποιεί τον αλγόριθμο Cooley-Tukey για τον αποδοτικό υπολογισμό του μετασχηματισμού DFT που είδαμε σε προηγούμενο κεφάλαιο. Εκτελεί ορθό (forward) ή και αντίστροφο (inverse) μετασχηματισμό DFT σε N δείγματα του σήματος εισόδου, όπου N μπορεί να είναι 2^m, όπου m=3-16 δηλαδή σε 8 έως 65536 δείγματα.

Όταν η είσοδος είναι σε αριθμητική μορφή σταθερής υποδιαστολής (fixed-point), τα N δεδομένα εισόδου είναι μιγαδικές τιμές που απεικονίζονται με 2 αριθμούς των b_x bits που είναι το πραγματικό και φανταστικό μέρος κάθε δείγματος. Το b_x παίρνει τιμές από 8 έως 32.

Όταν η είσοδος είναι σε μορφή κινητής υποδιαστολής απλής ακρίβειας (single-precision floating point), τα N δεδομένα εισόδου είναι μιγαδικές τιμές που απεικονίζονται με 2 αριθμούς floating-point των 32 bits ενώ οι παράγοντες φάσης (phase factors) με έναν αριθμό fixed-point των 24 ή 25 bits.

Ολόκληρη η απαιτούμενη για τους υπολογισμούς μνήμη, υλοποιείται στο chip του FPGA, ως block RAM ή distributed RAM. Τα N στοιχεία εξόδου απεικονίζονται με b_y bits για κάθε πραγματικό και φανταστικό μέρος των δεδομένων εξόδου. Τα δεδομένα εισόδου παρουσιάζονται με φυσική σειρά (natural order) ενώ τα δεδομένα εξόδου μπορούν να εμφανισθούν κατ' επιλογή είτε με φυσική είτε με bit/digit αντίστροφη σειρά (reverse order).

Για τον υπολογισμό FFT, είναι διαθέσιμες τρείς αριθμητικές επιλογές:

- Full-precision unscaled arithmetic
- Scaled fixed-point, όπου ο χρήστης επιλέγει τον προγραμματισμό (schedule) του scaling
- Block floating-point (run-time adjusted scaling)

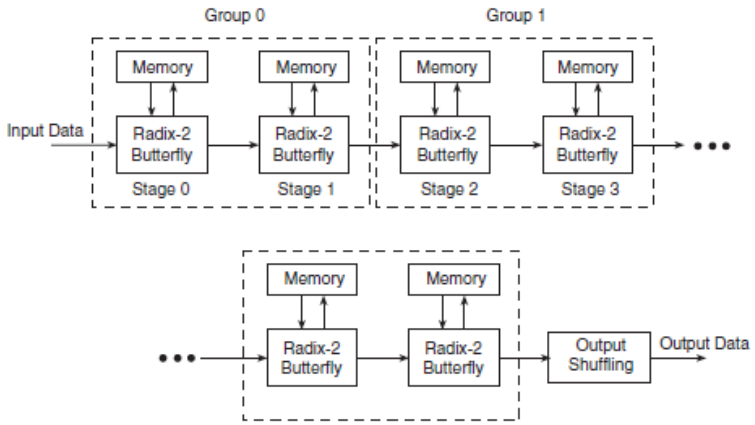
Ακόμη και κατά τη διάρκεια της λειτουργίας, ο χρήστης μπορεί να ρυθμίσει:

- τον αριθμό δειγμάτων N
- την επιλογή για forward/inverse DFT
- το scaling schedule
- το cyclic prefix length

Υποστηριζόμενες Αρχιτεκτονικές

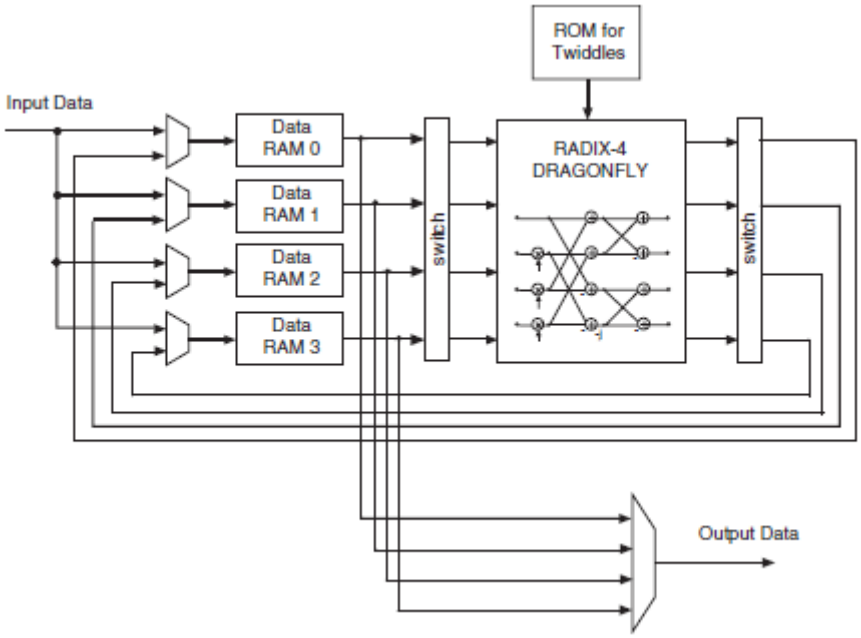
Το FFT core υποστηρίζει 4 αρχιτεκτονικές που μπορούν να επιλεχθούν, όπου κάθε μία αποτελεί ένα trade-off μεταξύ του μεγέθους του core και της ταχύτητάς του:

- "Pipelined, Streaming I/O" – Επιτρέπει συνεχή επεξεργασία των δεδομένων.



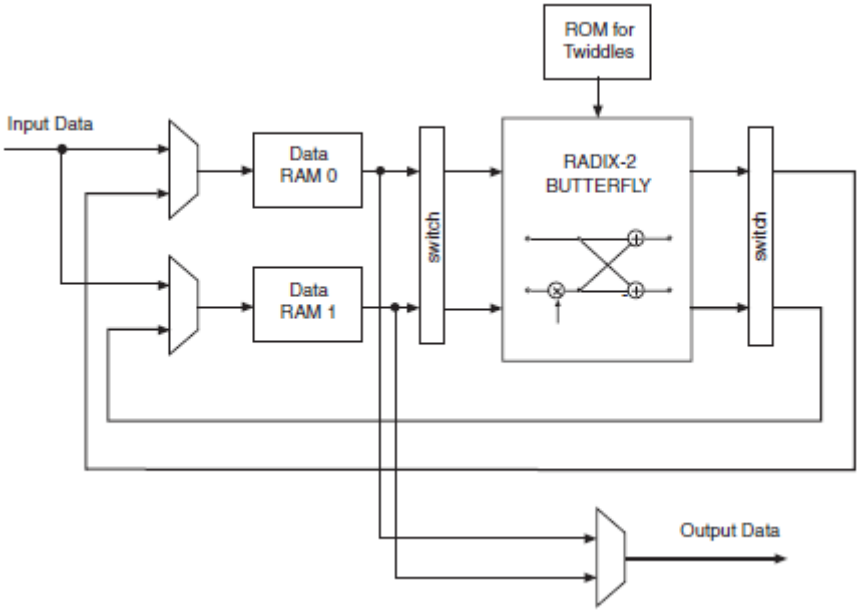
Σχήμα 4.3 Αρχιτεκτονική Pipelined Streaming I/O

- "Radix-4, Burst I/O" – Φορτώνει και επεξεργάζεται τα δεδομένα ξεχωριστά χρησιμοποιώντας επαναληπτική προσέγγιση. Είναι μικρότερη σε μέγεθος αλλά παρουσιάζει μεγαλύτερο χρόνο μετασχηματισμού.



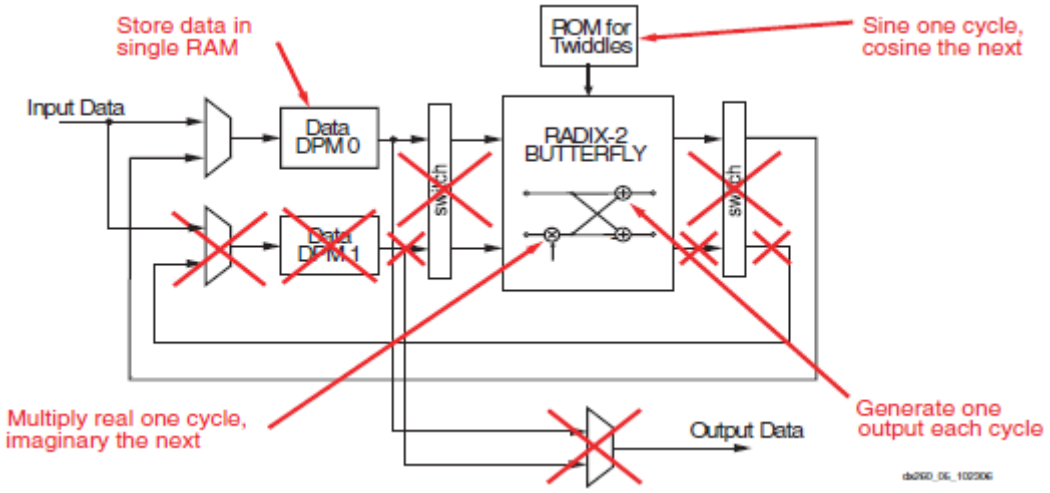
Σχήμα 4.4 Αρχιτεκτονική Radix-4, Burst I/O

- "Radix-2, Burst I/O" – Χρησιμοποιεί την ίδια επαναληπτική προσέγγιση όπως η προηγούμενη αλλά η μονάδα butterfly είναι μικρότερη. Αυτό σημαίνει ότι είναι μικρότερη σε μέγεθος αλλά παρουσιάζει μεγαλύτερο χρόνο μετασχηματισμού.



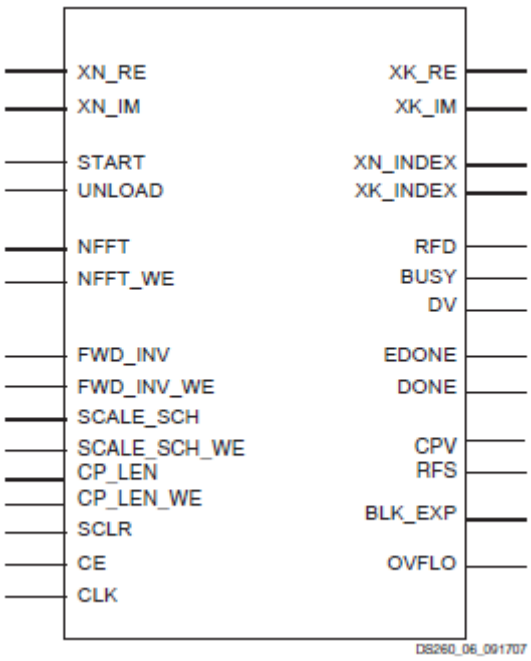
Σχήμα 4.5 Αρχιτεκτονική Radix-2, Burst I/O

- "Radix-2 Lite, Burst I/O" – Η παραλλαγή αυτή βασίζεται στην Radix-2 αρχιτεκτονική αλλά χρησιμοποιεί για τη μονάδα butterfly προσέγγιση πολυπλεξίας του χρόνου, μειώνοντας έτσι ακόμη περισσότερο το μέγεθος, με κόστος όμως τον ακόμη πιο αυξημένο χρόνο επεξεργασίας



Σχήμα 4.6 Αρχιτεκτονική Radix-2 Lite, Burst I/O

Το σχηματικό διάγραμμα του core, όπου φαίνονται οι είσοδοι/έξοδοι των σημάτων, φαίνεται στο σχήμα:



Σχήμα 4.7 Σχηματικό Διάγραμμα FFT core

Η επεξήγηση των εισόδων και εξόδων του core, υπάρχουν στο αντίστοιχο datasheet[3].

Αλγόριθμος

Χρησιμοποιούνται οι Radix-4 και Radix-2 αναλύσεις (decomposition) για τον υπολογισμό DFT. Στις αρχιτεκτονικές Burst I/O χρησιμοποιείται η μέθοδος decimation-in-time (DIT), ενώ στις Pipelined, Streaming I/O αρχιτεκτονικές η μέθοδος decimation-in-frequency (DIF). Όταν χρησιμοποιείται η ανάλυση Radix-4 ο N-point μετασχηματισμός FFT αποτελείται από $\log_4$ (N) στάδια, με κάθε ένα να περιέχει N/4 Radix-4 butterflies. Ο N-

point μετασχηματισμός FFT που χρησιμοποιεί την ανάλυση Radix-2 αποτελείται από $\log_2$ (N) στάδια, με κάθε ένα να περιέχει N/2 Radix-2 butterflies.

Εξέταση του πεπερασμένου μήκος των αριθμών

Οι αρχιτεκτονικές Burst I/O επεξεργάζονται έναν πίνακα δεδομένων εισόδου και ο αλγόριθμος, για κάθε στοιχείο του πίνακα, εκτελεί Radix-4 or Radix-2 butterflies, όπου κάθε butterfly παίρνει 4 ή 2 μιγαδικούς αριθμούς αντίστοιχα και επιστρέφει 4 ή 2 μιγαδικούς, στην ίδια μνήμη. Πιθανόν οι επιστρεφόμενοι αριθμοί να είναι μεγαλύτεροι από τους αρχικούς αριθμούς και επομένως, πρέπει να υπάρχει κάποιος τρόπος για τη διαχείριση αυτής της δυναμικής αύξησης των αριθμών.

Για τη μέθοδο Radix-4 DIT, οι τιμές στην έξοδο κάθε σταδίου butterfly μπορεί να παρουσιάσουν αύξηση κατά έναν παράγοντα $1 + 3\sqrt{2} \approx 5{,}242$, που σημαίνει αύξηση του αριθμού κατά 3 bits. Για τη μέθοδο Radix-2, η αύξηση είναι κατά ένα παράγοντα $1 + \sqrt{2} \approx 2{,}414$, δηλαδή αύξηση των bits κατά 2. Η αύξηση αυτή των bits μπορεί να αντιμετωπισθεί με 3 τρόπους:

- ✓ Εκτελώντας τους υπολογισμούς χωρίς κλιμάκωση (scaling) και μεταφέροντας όλα τα σημαντικά bits του ακεραίου στο τέλος του υπολογισμού
- ✓ Εφαρμόζοντας κλιμάκωση σε κάθε στάδιο χρησιμοποιώντας ένα σταθερό αριθμό κλιμάκωσης
- ✓ Αυτόματη κλιμάκωση με χρήση block floating-point

Όταν δεν υπάρχει κλιμάκωση, όλα τα σημαντικά ψηφία του ακεραίου διατηρούνται. Το πλάτος του data path εντός του σταδίου butterfly αυξάνεται για να ανταπεξέλθει στην αύξηση των bits, ενώ στην έξοδο του σταδίου, αποκόβονται (ή στρογγυλοποιούνται) τα bits του δεκαδικού μέρους που προήλθαν από την αύξηση μετά τον πολλαπλασιασμό. Το πλάτος της εξόδου είναι (πλάτος εισόδου + $\log_2$(transform length) + 1). Αυτό αφορά τη χειρότερη περίπτωση.

Όταν χρησιμοποιείται κλιμάκωση, χρησιμοποιείται ένα μοτίβο κλιμάκωσης (scaling schedule) για τη διαίρεση των αριθμών σε κάθε στάδιο κατά ένα παράγοντα 1,2,4, ή 8. Αν η κλιμάκωση είναι ανεπαρκής, η έξοδος του σταδίου butterfly μπορεί να υπερβεί τη δυναμική περιοχή και να προκαλέσει υπερχείλιση. Η εφαρμογή της κλιμάκωσης θα έχει ως αποτέλεσμα να εμφανισθεί με κλιμάκωση ο τελικός μετασχηματισμός. Ο παράγοντας κλιμάκωσης s ορίζεται ως:

$$s = 2^{\sum_{i=0}^{\log N-1} b_i}$$

όπου το b_i είναι η κλιμάκωση (σε bits) που εφαρμόζεται σε κάθε στάδιο i.

Η κλιμάκωση έχει ως αποτέλεσμα να μεταβληθούν οι όροι της τελικής ακολουθίας εξόδου κατά ένα παράγοντα 1/s. Για τον ορθό (forward) μετασχηματισμό FFT, η ακολουθία εξόδου X’ (k), k = 0,...,N – 1 που υπολογίζεται από το core, ορίζεται ως

$$X'(k) = \frac{1}{s}X(k) = \frac{1}{s}\sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} \ , k = 0,..N-1$$

Για τις υλοποιήσεις Burst I/O, το scaling schedule καθορίζεται με 2 bits για κάθε στάδιο, με την κλιμάκωση στο $1^{ο}$ στάδιο να δίνεται με τα δύο LSBs. Η κλιμάκωση καθορίζεται ως 3, 2, 1 ή 0 που αντιπροσωπεύει των αριθμό των bits που θα γίνει ολίσθηση μετά από κάθε στάδιο.

Στην δική μας υλοποίηση, που είναι Radix-2 Lite Burst I/O, με N=1024, θα έχουμε $\log_2$ (1024) = 10 στάδια και επομένως εφαρμόζοντας scaled fixed-point, το scaling schedule θα αποτελείται από 2*10 = 20 bits (2 bits για κάθε στάδιο). Με βάση τη θεωρία, ξέρουμε ότι το πλάτος των συνιστωσών εξόδου M_c του μετασχηματισμού DFT μιγαδικών δειγμάτων εισόδου πλάτους A_0 όπως προκύπτουν από την εξίσωση DFT

$$X'(k) = \sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N} \ , k = 0,..N-1$$

είναι:

$M_c = N* A_0$

δηλαδή τα πλάτη των συνιστωσών εξόδου είναι αυξημένα κατά παράγοντα N[2,p.151].

Επομένως, βλέπουμε ότι θέτοντας τον παράγοντα κλιμάκωσης s = 2^{10} = 1024 = N, θα έχουμε:

$$X'(k) = \frac{1}{s}X(k) = \frac{1}{N}X(k) = \frac{1}{N}\sum_{n=0}^{N-1} x(n)e^{-jnk2\pi/N}$$

δηλαδή αναιρείται ακριβώς η αύξηση του πλάτους που εισάγει η εξίσωση DFT. Έτσι, επιλέγουμε για την υλοποίησή μας να εφαρμόσουμε ως scaling schedule το μοτιβο:

[01 01 01 01 01 01 01 01 01 01]

που μας δίνει $s = 2^{10} = 1024 = N$, με το οποίο επιπρόσθετα αποφεύγεται και πιθανή υπερχείλιση[3,p.26].

Στην περίπτωση που θέλω να χρησιμοποιήσω διαφορετικό scaling schedule και ταυτόχρονα να έχω διορθωμένα πλάτη στην έξοδο, θα πρέπει να διαιρέσω τις εξόδους με κατάλληλο παράγοντα, δύναμη του 2, έτσι ώστε το συνολικό scaling να είναι ίσο με 1/N: αν για παράδειγμα το scaling schedule που θέλω να χρησιμοποιήσω έχει συνολικά 5 bits scaling, τότε θα πρέπει:

$$N = 2^k * s = 2^\kappa * 2^5 = 1024 => 2^{k+5} = 2^{10} => k = 5$$

Για τις υλοποιήσεις Pipelined, Streaming I/O, το scaling schedule καθορίζεται με 2 bits για κάθε στάδιο Radix-2, αρχίζοντας από τα δύο LSBs.

Με τη χρήση της αυτόματης κλιμάκωσης block floating-point, σε κάθε στάδιο εφαρμόζεται η αναγκαία κλιμάκωση για να διατηρούνται η αριθμοί εντός ορίων και η κλιμάκωση παρακολουθείται από ένα block exponent.

Όπως και στην περίπτωση που δεν υπάρχει κλιμάκωση, έτσι και στη κλιμάκωση με scaling schedule και block floating-point, το core δεν έχει κάποια καθορισμένη θέση για την υποδιαστολή. Η θέση της υποδιαστολής στα δεδομένα εξόδου κληρονομείται από τα δεδομένα εισόδου και μετατοπίζεται από την εφαρμογή της κλιμάκωσης.

Είσοδος Πραγματικών Δεδομένων

Η μονάδα FFT δέχεται δείγματα με τη μορφή μιγαδικών αλλά μπορεί να δεχτεί και πραγματικά δεδομένα, όπως στη δική μας περίπτωση. Στην επεξεργασία που αφορά πραγματικά δεδομένα, το φανταστικό μέρος των δεδομένων εισόδου των δειγμάτων τίθεται μηδενικό. Στην έξοδο βέβαια προκύπτει μη μηδενικό φανταστικό μέρος.

Λόγω του πεπερασμένου μήκους των αριθμών που είδαμε παραπάνω, κατά το μετασχηματισμό εισάγεται θόρυβος που έχει ως αποτέλεσμα τα δεδομένα εξόδου να μην παρουσιάζουν απόλυτη συμμετρία, όπως θα έπρεπε. Οι αλγόριθμοι DIT και DIF παρουσιάζουν διαφορετική επίδραση θορύβου λόγω της διαφορετικής σειράς υπολογισμών.

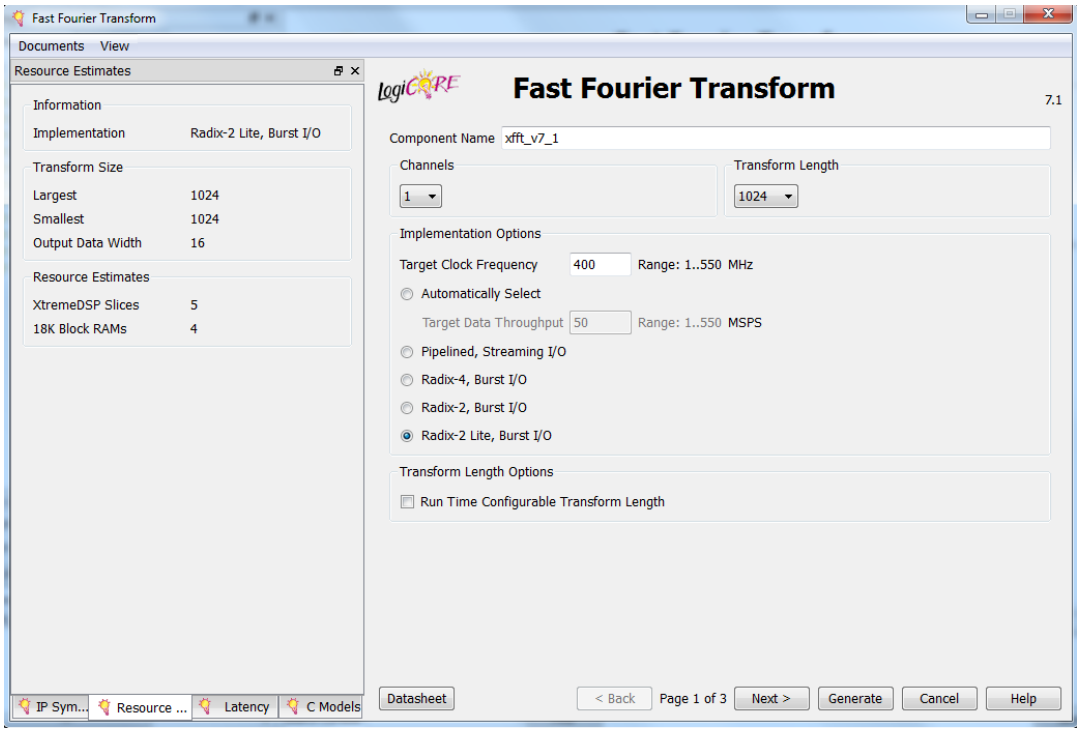
Η ασυμμετρία μεταξύ των δύο μισών του φάσματος που έχουμε στην έξοδο, εμφανίζεται εντονότερη όταν έχουμε μεγαλύτερο αριθμό δειγμάτων (N-point). Επιπρόσθετα, ο θόρυβος είναι πιο χαρακτηριστικός στις συχνότητες που βρίσκονται στις χαμηλότερες «θυρίδες» (bins) και για το λόγο αυτό είναι καλύτερο να χρησιμοποιούμε το άνω μισό του φάσματος που προκύπτει από το μετασχηματισμό, δηλαδή τα σημεία από N/2 +1 έως N.

Επιλογές Παραμέτρων Υλοποίησης

Υλοποιούμε το FFT core με το “core generator”, θέτοντας τις απαιτούμενες παραμέτρους υλοποίησης, όπως περιγράφεται παρακάτω, για να προκύψει FFT core:

- Radix-2 Lite, Burst I/O
- Transform length N = 1024
- Data width = 16 bits

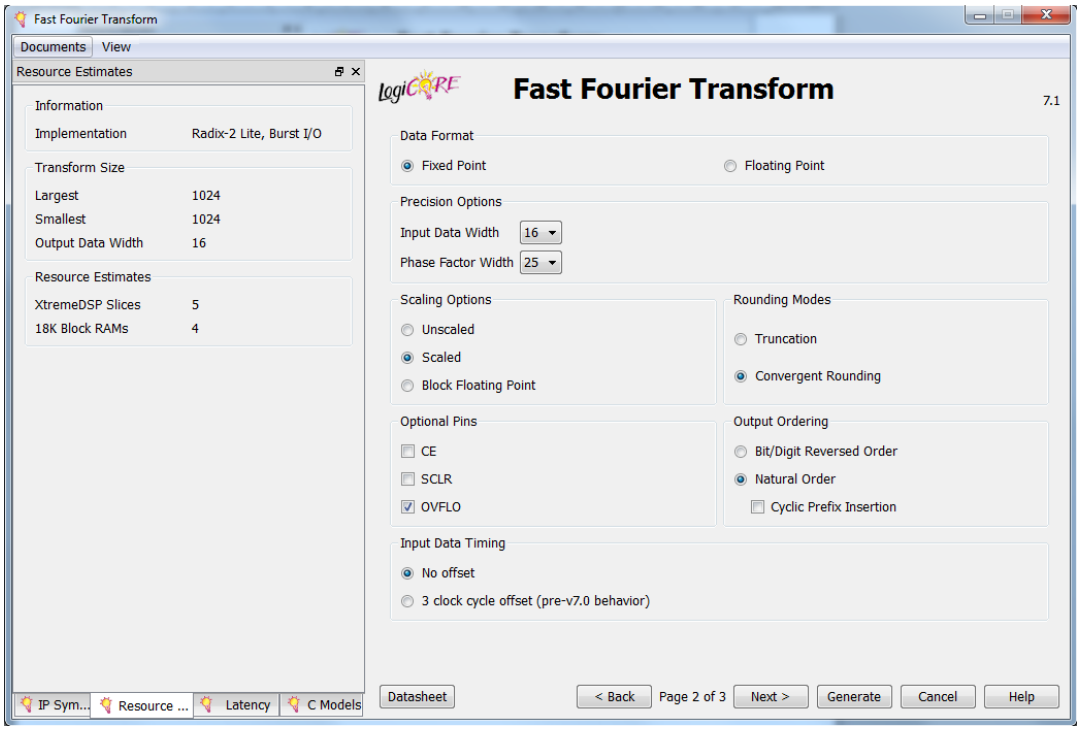
Οθόνη 1:



Σχήμα 4.8 Οθόνη 1 παραμετροποίησης FFT core

- **Channels:** Αριθμός Καναλιών λειτουργίας. Επιλέγουμε 1 κανάλι
- **Transform Length:** Επιλέγουμε των αριθμό των δειγμάτων της ακολουθίας εισόδου. Επιλέγουμε 1024, που θα διατηρούμε σταθερό συνεχώς.
- **Implementation Options:** Εδώ μπορούμε να επιλέξουμε τη συχνότητα λειτουργίας της υλοποίησης του FPGA. Αυτή η συχνότητα χρησιμοποιείται από το πρόγραμμα για να επιλέξει αυτόματα μια αρχιτεκτονική, έτσι ώστε να πετύχουμε συγκεκριμένο ρυθμό εξόδου δεδομένων που επιθυμούμε. Εμείς επιλέγουμε την "Radix-2 Lite, Burst I/O" αρχιτεκτονική η οποία καταναλώνει τους λιγότερους πόρους του chip, καθώς δεν μας απασχολεί ιδιαίτερα η καθυστέρηση μετασχηματισμού.
- **Transform Length Options:** Επιλογή για να έχουμε τη δυνατότητα να μεταβάλουμε το Transform Length κατά τη διάρκεια λειτουργίας. Δεν το επιθυμούμε.

Οθόνη 2:

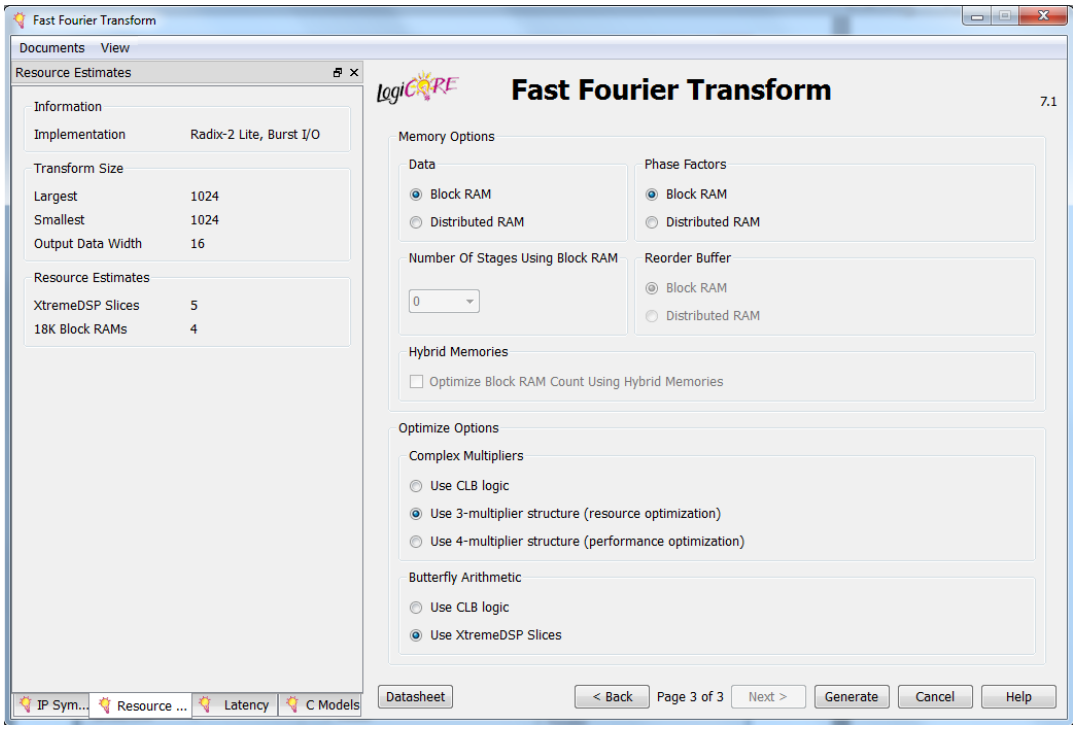


Σχήμα 4.9 Οθόνη 2 παραμετροποίησης FFT core

- **Data Format:** Επιλέγουμε το είδος της αριθμητικής αναπαράστασης που θα έχουν τα δεδομένα εισόδου και εξόδου. Επιλέγουμε "Fixed Point – σταθερής υποδιαστολής".
- **Precision Options:** Εδώ καθορίζουμε τον αριθμό των bits που θα έχουν τα πραγματικά και φανταστικά μέρη των μιγαδικών αριθμών εισόδου, καθώς και οι παράγοντες φάσης. Επιλέγουμε 16 bits για κάθε μέρος των μιγαδικών και 25 για τους phase factors (που δεν θα τους χρησιμοποιήσουμε)
- **Scaling Options:** Επιλέγουμε τον τρόπο με τον οποίο θα γίνεται και αν θα γίνεται η αυξομείωση των ενδιάμεσων τιμών μεταξύ των επιπέδων του μετασχηματισμού FFT. Επιλέγουμε "Scaled", έτσι ώστε το scaling να γίνεται με βάση το schedule που θα καθορίσουμε και που θα μπορούμε να μεταβάλουμε.
- **Optional Pins:** Χρησιμοποιούμε μόνο το Overflow (OVFLO), που μας δείχνει εάν έχει προκύψει υπερχείλιση σε κάποιο στάδιο επεξεργασίας FFT.
- **Rounding Modes:** Καθορίζεται ο τρόπος που θα γίνεται η τυχόν αναγκαία στρογγυλοποίηση των LSBs των τιμών που προκύπτουν μετά από κάθε στάδιο butterfly. Επιλέγουμε "convergent", που σημαίνει ότι όταν το δεκαδικό μέρος ενός αριθμού είναι ίσο με το μισό ακριβώς, τότε εάν ο αριθμός είναι περιττός στρογγυλοποιείται προς τα πάνω ενώ εάν είναι άρτιος προς τα κάτω. Με τον τρόπο αυτό αποφεύγεται η εισαγωγή DC απόκλισης που παρουσιάζεται στην περίπτωση της περικοπής (truncation).
- **Output Ordering:** Οι επιλογές για τα δεδομένα εξόδου είναι αυτά να εμφανίζονται είτε σε μορφή Natural Order είτε σε Bit/Digit Reversed Order. Επιλέγουμε Natural Order με κόστος μια επιπλέον καθυστέρηση στην επεξεργασία. Δεν επιλέγουμε Cyclic Prefix Insertion.

- **Input Data Timing:** Αφορά την καθυστέρηση μεταξύ κάθε δείγματος εισόδου(data) και του αντίστοιχου δείκτη (index). Δεν το επιλέγουμε.

Οθόνη 3:



Σχήμα 4.10 Οθόνη 3 παραμετροποίησης FFT core

- **Memory Options:** Καθορίζουμε το είδος της μνήμης του FPGA που θα χρησιμοποιηθεί για την υλοποίηση. Επιλέγουμε block RAM, αλλά στην αρχιτεκτονική Burst I/O δεν έχει μεγάλη σημασία η επιλογή.
- **Optimize Options:** Για οικονομία στο hardware (resource optimization), επιλέγουμε 3-multiplier structure καθώς και τα διαθέσιμα Xtreme DSP Slices.

4.4 Software Λειτουργίας και Ελέγχου

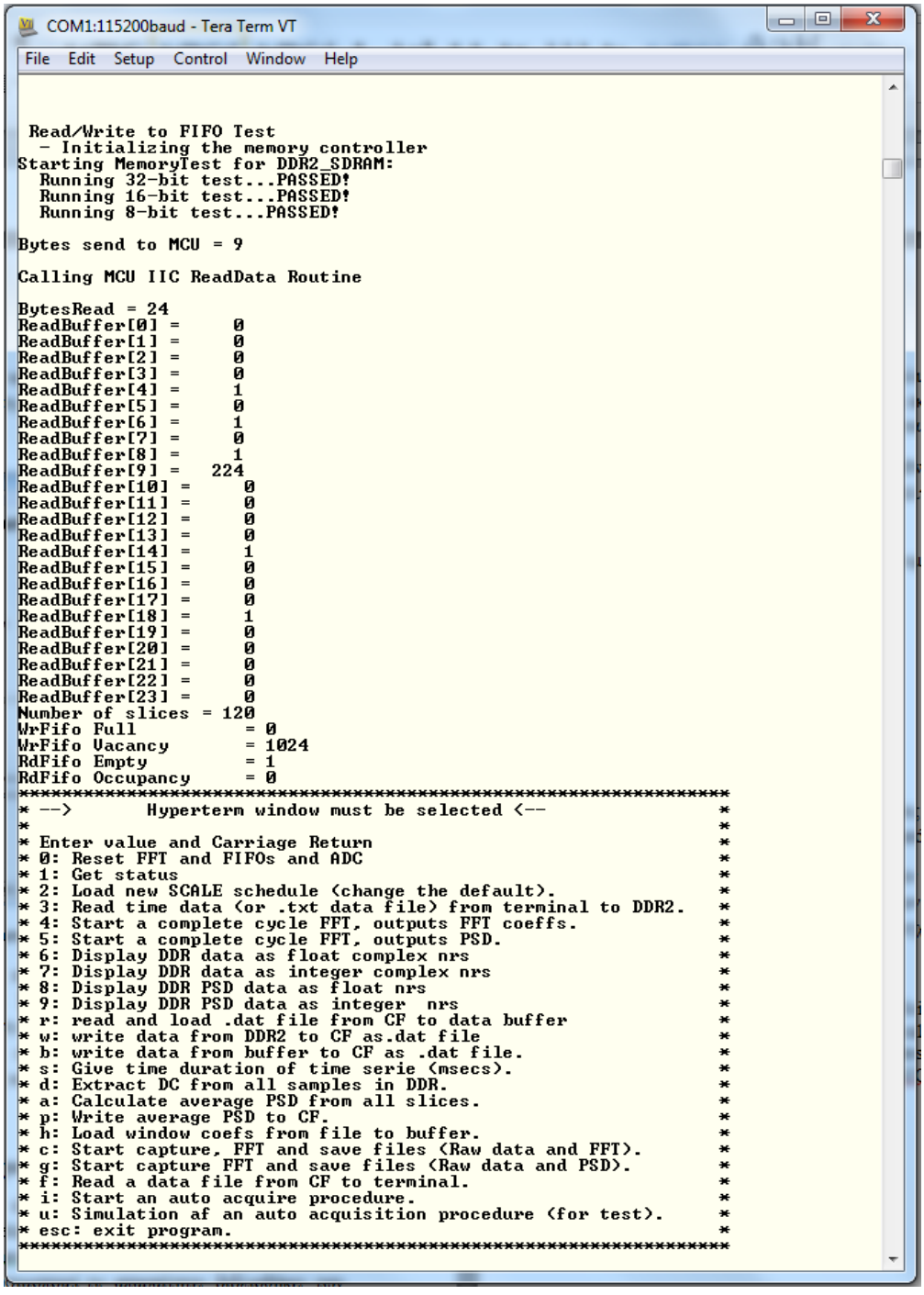
Το απαραίτητο λογισμικό λειτουργίας που ελέγχει το hardware που υλοποιήσαμε, αναπτύχθηκε με την εφαρμογή “Software Development Platform - SDK” της πλατφόρμας “Xilinx ISE Design Suite v12.4. Το software αυτό αποτελεί μία custom standalone εφαρμογή που γράφθηκε σε γλώσσα C και χρησιμοποιεί τις απαραίτητες βιβλιοθήκες των αντίστοιχων μονάδων hardware. Παρέχει τη δυνατότητα του πλήρη ελέγχου του συστήματος ενώ παράλληλα επιτρέπει στο χρήστη την επικοινωνία του με αυτό, μέσω της σειριακής θύρας RS232 και ενός υπολογιστή που τρέχει οποιαδήποτε εφαρμογή terminal emulator.

Κατά την εφαρμογή τάσης στο σύστημα, διαβάζεται συγκεκριμένο αρχείο που βρίσκεται στην κάρτα CF και περιέχει το hardware configuration του FPGA καθώς και το software που φορτώνεται και τρέχει στον επεξεργαστή που υλοποιήσαμε.

Η επικοινωνία του software με το υλικό υλοποιείται με τη χρήση registers: η τιμή ενός register διαβάζεται από το υλικό και εκτελεί την καθορισμένη λειτουργία ενώ αντίστροφα, το υλικό γράφει σε κάποιον register για να περάσει μια τιμή στο software.

Η επικοινωνία του χρήστη με το software (και επομένως με το υλικό), επιτυγχάνεται με εντολές ASCII χαρακτήρων που στέλνει ο υπολογιστής στο σύστημα μέσω της θύρας RS232

Η αρχική οθόνη του τερματικού που εμφανίζεται, φαίνεται παρακάτω:

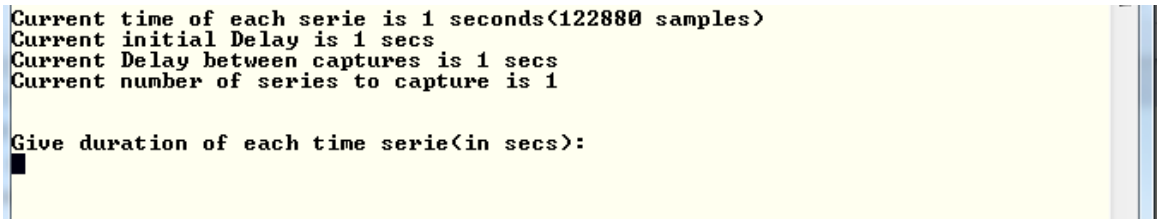


Σχήμα 4.11 Αρχική οθόνη του software ελέγχου

Αρχικά και κάθε φορά που ενεργοποιείται η ψηφιακή μονάδα του FPGA, γίνεται αρχικοποίηση του ελεγκτή και έλεγχος της μνήμης DDR2 και των FIFO, όπως φαίνεται και στις πρώτες γραμμές της οθόνης του terminal.

Στη συνέχεια, αναμένεται η είσοδος από το χρήστη ενός ASCII χαρακτήρα για να εκτελεσθεί η αντίστοιχη λειτουργία από το σύστημα. Εκτός από τις εντολές που εκτελούν ένα πλήρη κύκλο λειτουργιών (δειγματοληψία -> FFT -> αποθήκευση αποτελεσμάτων στη CF), υπάρχουν και εντολές που εκτελούν μεμονωμένες λειτουργίες, για τον έλεγχο του συστήματος.

Αρχικά, πληκτρολογώντας το χαρακτήρα ‘s’, καθορίζουμε τους χρόνους και τα delays ενός πλήρη κύκλου λειτουργίας: αρχικό delay, διάρκεια κάθε δειγματοληψίας (time series), αριθμός των χρονοσειρών και delay μεταξύ αυτών.



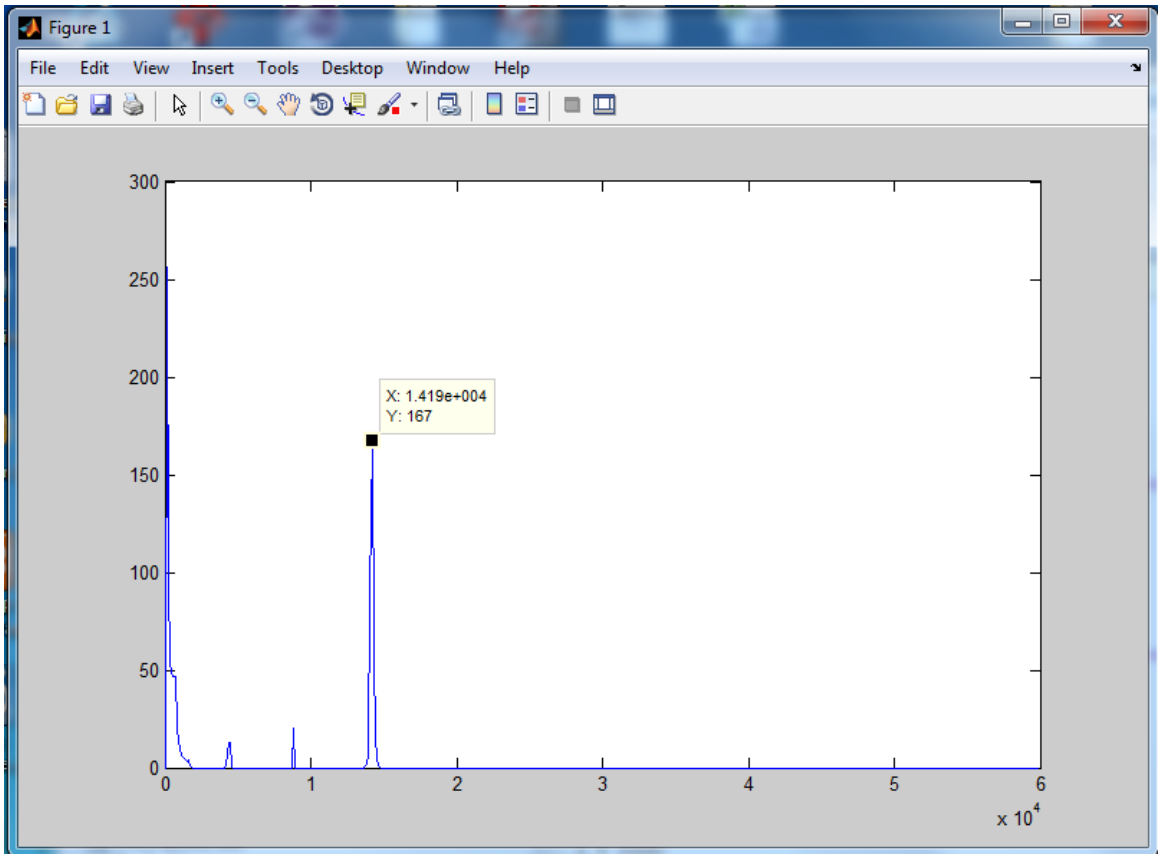
Σχήμα 4.12 Καθορισμός Παραμέτρων λειτουργίας

Αφού προσδιορισθούν αυτοί οι παράμετροι, ξεκινάμε τη λειτουργία πληκτρολογώντας το χαρακτήρα ‘i’, για να ξεκινήσει η αυτόματη διαδικασία συλλογής δεδομένων. Μετά το πέρας κάθε χρονοσειράς, τα δεδομένα αυτής, που δείχνουν την φασματική πυκνότητα ισχύος, αποθηκεύονται στη κάρτα μνήμης CF ως αρχείο με όνομα ‘Ax’, όπου x ο αύξων αριθμός της χρονοσειράς. Το αρχείο αυτό περιέχει 2052 bytes δεδομένων εκ των οποίων τα 4 πρώτα είναι η επικεφαλίδα και τα υπόλοιπα 2048 περιέχουν τους 512 αριθμούς των 32bit (512x4 bytes = 2048) που είναι οι φασματικές συνιστώσες του άνω φάσματος υπο μορφή ισχύος. Οι αριθμοί, όπως είδαμε σε προηγούμενο κεφάλαιο είναι θετικοί (αφού δείχνουν ισχύ) και είναι σε αναπαράσταση Q0.30.

Με το πρόγραμμα matlab μπορούμε τώρα να διαβάσουμε το αρχείο (αφού το μεταφέρουμε από την CF στον υπολογιστή) και να δουμε τη γραφική παράσταση του φάσματος. Οι εντολές που δίνουμε στο matlab για το σκοπό αυτό είναι:

```
 [filename, pathname] = uigetfile('A\*', 'Pick a data file');
fid=fopen([pathname filename],'r','ieee-be');
[a,count]=fread(fid,1,'uint32');
nsamps= 512;
[y]=(fread(fid,nsamps,'uint32'));
fclose(fid);
Fs = 120000;
T = 1/Fs;
L = 1024;
t = (0:L-1)*T;
f = Fs*linspace(0,1,L);
plot(f(1:L/2),y(1:L/2))
```

Το αποτέλεσμα φαίνεται στην εικόνα παρακάτω:



Σχήμα 4.13 Φάσμα που προκύπτει

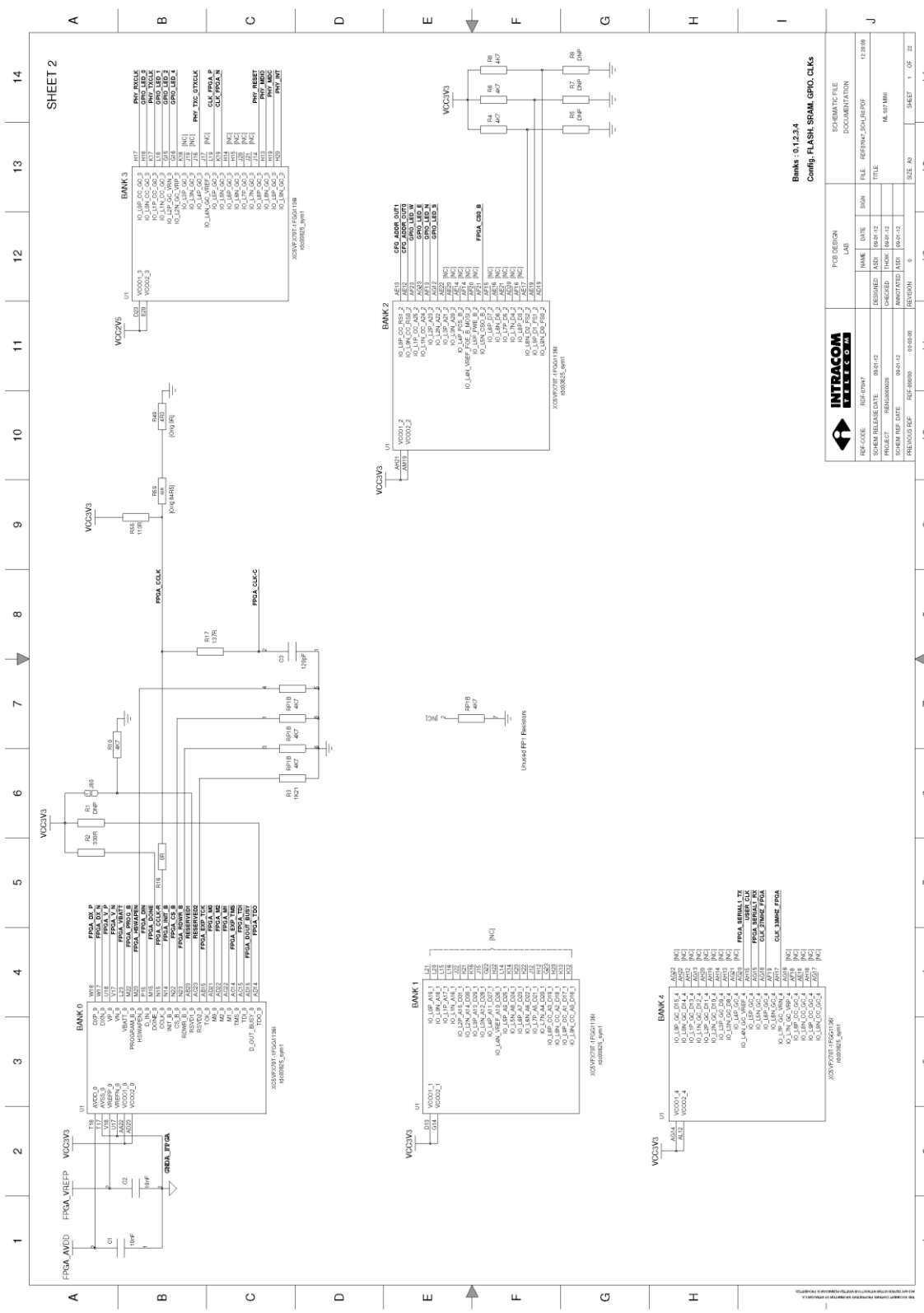
4.5 Βιβλιογραφία

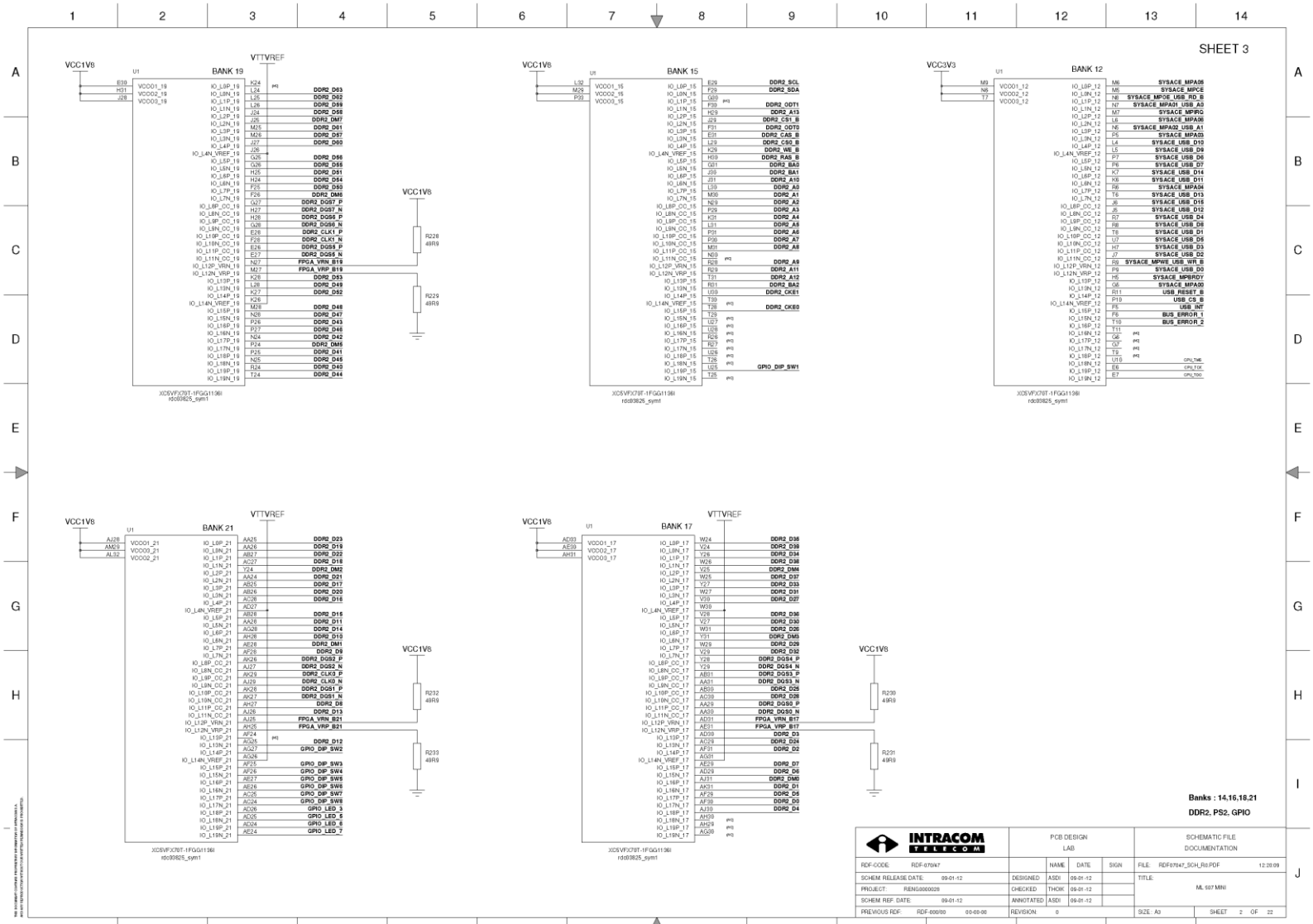
[1] Steven W. Smith, 1999, The Scientist and Engineer's Guide to Digital Signal Processing, California Technical Publishing

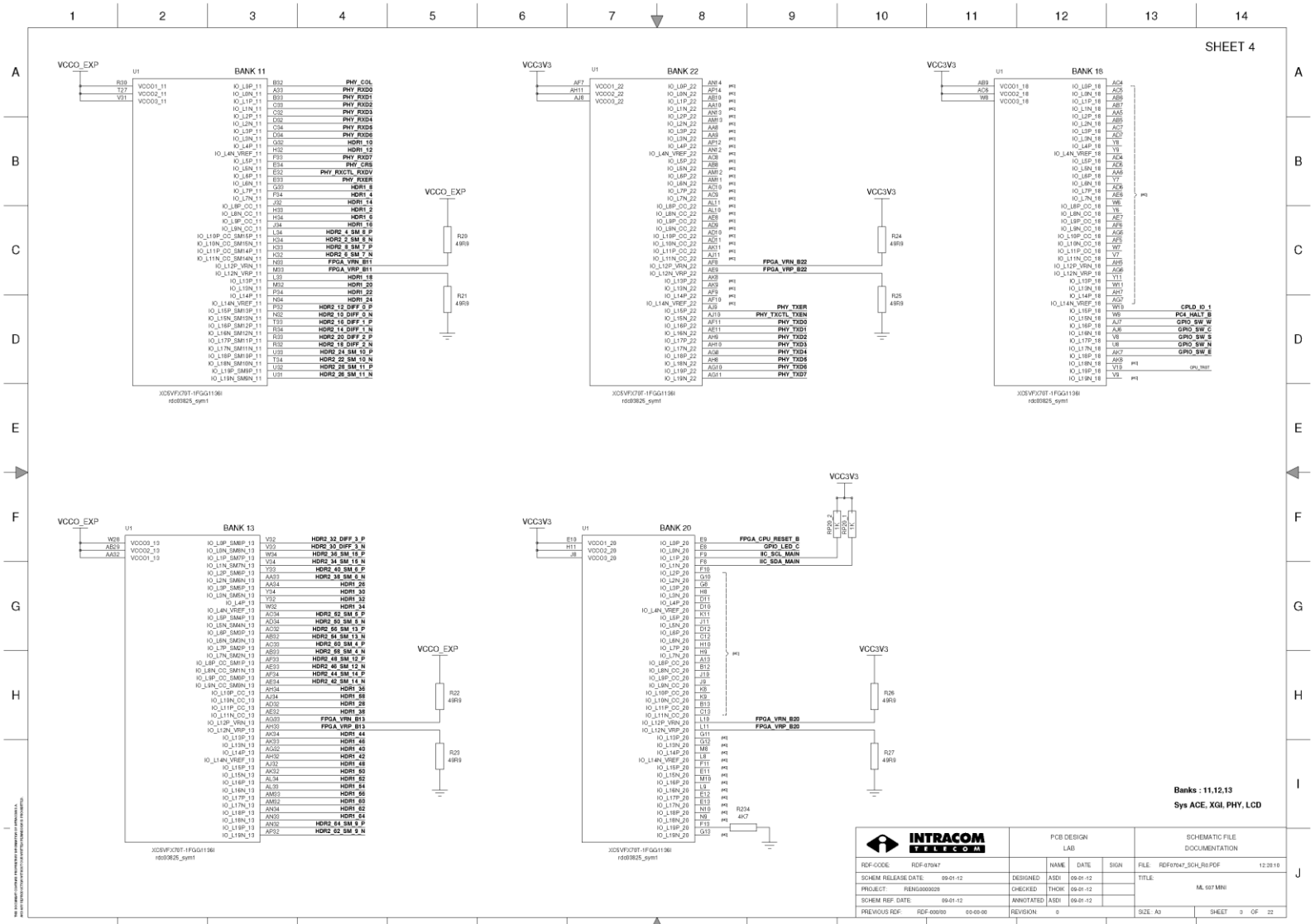
[2] Richard G. Lyons, 2011, Understanding Digital Signal Processing, Prentice Hall

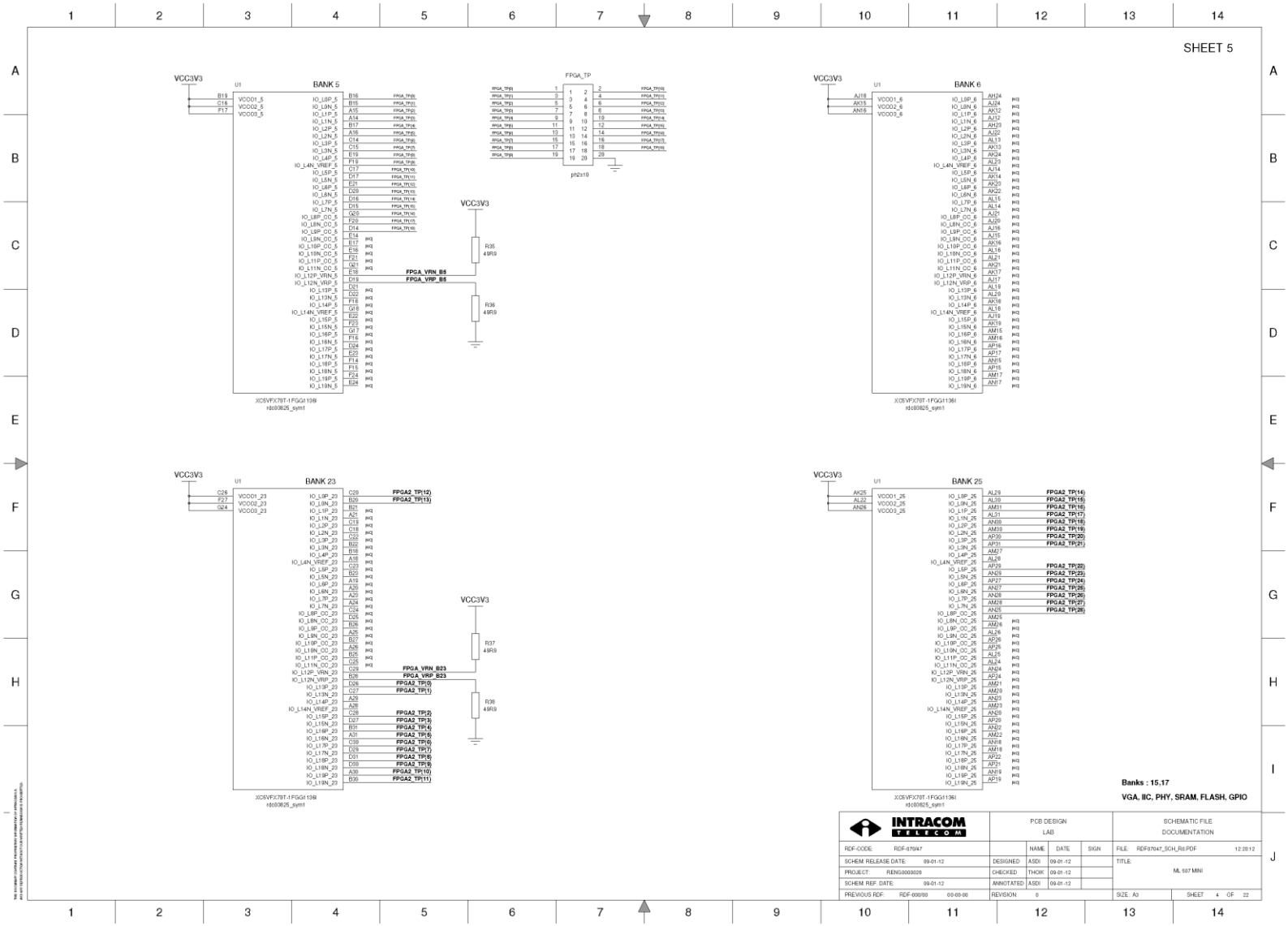
[3] Xilinx, 2011, Datasheet: LogiCORE IP Fast Fourier Transform v7.1, Xilinx

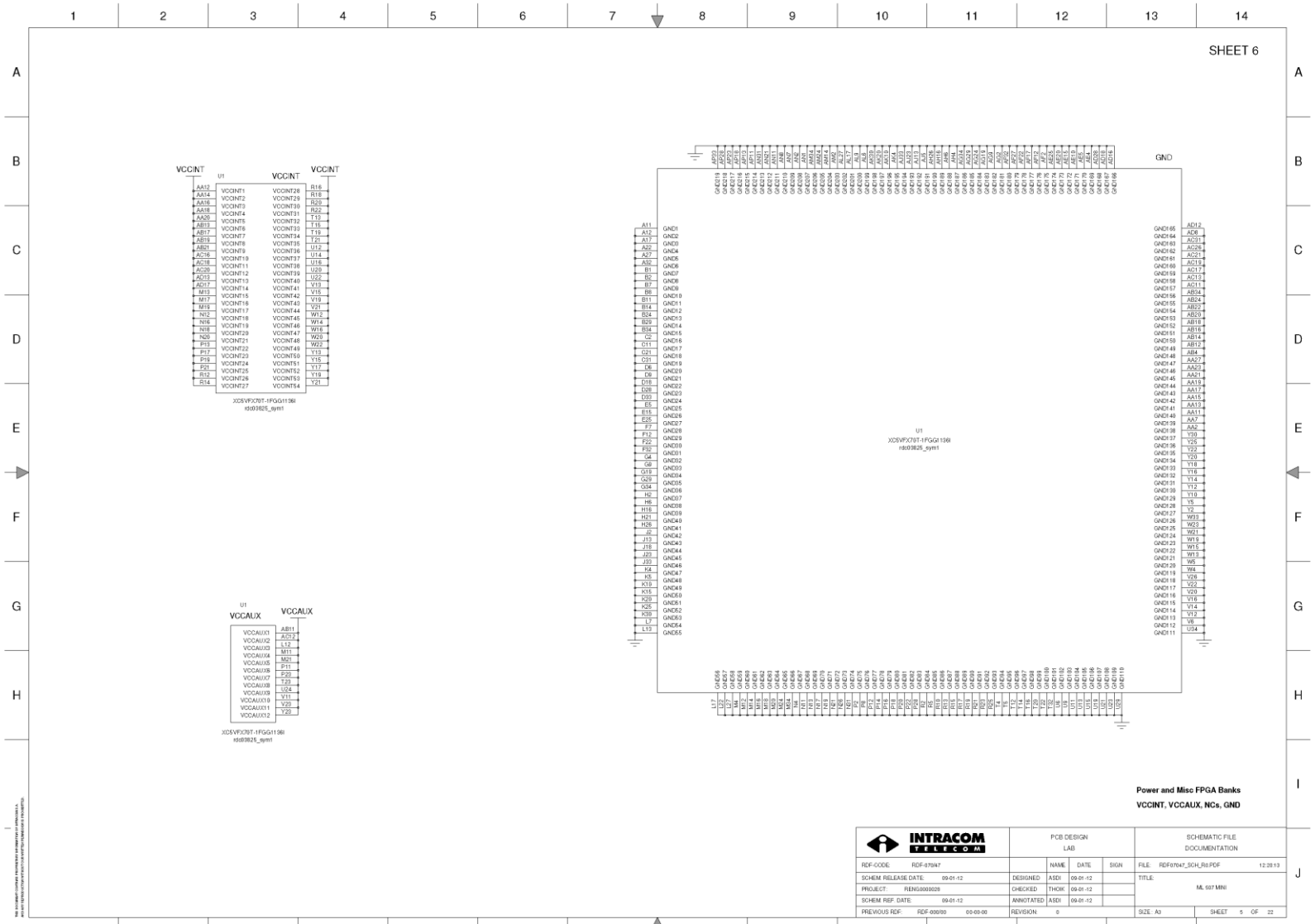
ΠΑΡΑΡΤΗΜΑ Α: Διαγράμματα Κυκλώματος Ψηφιακής βαθμίδας FPGA




SHEET 3
BANK 19
VTTVREF
VCC1V8
DDR2_D63
DDR2_D62
DDR2_D59
DDR2_D58
DDR2_DM7
DDR2_D61
DDR2_D57
DDR2_D60
DDR2_D56
DDR2_D55
DDR2_D51
DDR2_D54
DDR2_D50
DDR2_DM6
DDR2_DQS7_P
DDR2_DQS7_N
DDR2_DQS6_P
DDR2_DQS6_N
DDR2_CLK1_P
DDR2_CLK1_N
DDR2_DQS5_P
DDR2_DQS5_N
FPGA_VRN_B19
FPGA_VRP_B19
DDR2_D53
DDR2_D49
DDR2_D52
DDR2_D48
DDR2_D47
DDR2_D43
DDR2_D46
DDR2_D42
DDR2_DM5
DDR2_D41
DDR2_D45
DDR2_D40
DDR2_D44
R228 49R9
R229 49R9
XC5VFX70T-1FGG1136I
rdc00825_sym1
BANK 15
DDR2_SCL
DDR2_SDA
DDR2_ODT1
DDR2_A13
DDR2_CS1_B
DDR2_ODT0
DDR2_CAS_B
DDR2_CS0_B
DDR2_WE_B
DDR2_RAS_B
DDR2_BA0
DDR2_BA1
DDR2_A10
DDR2_A0
DDR2_A1
DDR2_A2
DDR2_A3
DDR2_A4
DDR2_A5
DDR2_A6
DDR2_A7
DDR2_A8
DDR2_A9
DDR2_A11
DDR2_A12
DDR2_BA2
DDR2_CKE1
DDR2_CKE0
GPIO_DIP_SW1
XC5VFX70T-1FGG1136I
rdc00825_sym1
BANK 12
VCC3V3
SYSACE_MPA06
SYSACE_MPCE
SYSACE_MPOE_USB_RD_B
SYSACE_MPA01_USB_A0
SYSACE_MPRQ
SYSACE_MPA08
SYSACE_MPA02_USB_A1
SYSACE_MPA03
SYSACE_USB_D10
SYSACE_USB_D9
SYSACE_USB_D6
SYSACE_USB_D7
SYSACE_USB_D14
SYSACE_USB_D11
SYSACE_MPA04
SYSACE_USB_D13
SYSACE_USB_D15
SYSACE_USB_D12
SYSACE_MPWE_USB_WR_B
SYSACE_USB_D0
SYSACE_MPBRDY
SYSACE_MPA00
USB_RESET_B
USB_CS_B
USB_INT
BUS_ERROR_1
BUS_ERROR_2
CPU_TMS
CPU_TDI
CPU_TDO
XC5VFX70T-1FGG1136I
rdc00825_sym1
BANK 21
VTTVREF
VCC1V8
DDR2_D23
DDR2_D19
DDR2_D22
DDR2_D18
DDR2_DM2
DDR2_D21
DDR2_D17
DDR2_D20
DDR2_D16
DDR2_D15
DDR2_D11
DDR2_D14
DDR2_D10
DDR2_DM1
DDR2_D9
DDR2_DQS2_P
DDR2_DQS2_N
DDR2_CLK0_P
DDR2_CLK0_N
DDR2_DQS1_P
DDR2_DQS1_N
DDR2_D8
DDR2_D13
FPGA_VRN_B21
FPGA_VRP_B21
DDR2_D12
GPIO_DIP_SW2
GPIO_DIP_SW3
GPIO_DIP_SW4
GPIO_DIP_SW5
GPIO_DIP_SW6
GPIO_DIP_SW7
GPIO_DIP_SW8
GPIO_LED_3
GPIO_LED_5
GPIO_LED_6
GPIO_LED_7
R232 49R9
R233 49R9
XC5VFX70T-1FGG1136I
rdc00825_sym1
BANK 17
VTTVREF
VCC1V8
DDR2_D38
DDR2_D39
DDR2_D34
DDR2_D36
DDR2_DM4
DDR2_D37
DDR2_D33
DDR2_D31
DDR2_D27
DDR2_D30
DDR2_D35
DDR3_DM3
DDR2_D29
DDR2_D32
DDR2_DQS4_P
DDR2_DQS4_N
DDR2_DQS3_P
DDR2_DQS3_N
DDR2_D25
DDR2_D28
DDR2_DQS0_P
DDR2_DQS0_N
FPGA_VRN_B17
FPGA_VRP_B17
DDR2_D3
DDR2_D24
DDR2_D2
DDR2_D7
DDR2_D6
DDR2_DM0
DDR2_D1
DDR2_D5
DDR2_D0
DDR2_D4
R230 49R9
R231 49R9
XC5VFX70T-1FGG1136I
rdc00825_sym1
Banks : 14,16,18,21
DDR2, PS2, GPIO
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:09
TITLE: ML 507 MINI
SIZE: A3
SHEET 2 OF 22

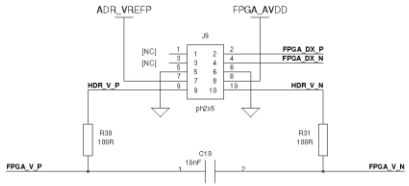

SHEET 4
BANK 11
BANK 22
BANK 16
BANK 13
BANK 20
VCCO_EXP
VCC3V3
XC5VFX70T-1FGG1136I
rdc03825_sym1
Banks : 11,12,13
Sys ACE, XGI, PHY, LCD
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
FILE: RDF07047_SCH_R0.PDF 12:28:10
TITLE: ML 507 MINI
SIZE: A3
SHEET 3 OF 22


SHEET 5
BANK 5
BANK 6
BANK 23
BANK 25
FPGA_TP
VCC3V3
R35 49R9
R36 49R9
R37 49R9
R38 49R9
FPGA_VRN_B5
FPGA_VRP_B5
FPGA_VRN_B23
FPGA_VRP_B23
XC5VFX70T-1FFG1136I
rdc03625_sym1
Banks : 15,17
VGA, IIC, PHY, SRAM, FLASH, GPIO
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0800028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:12
TITLE: ML 507 MINI
SIZE : A3
SHEET 4 OF 22

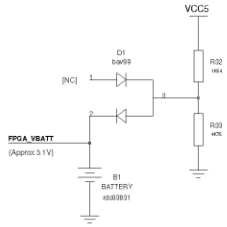

SHEET 6
VCCINT
VCCINT
VCCINT
XC5VFX70T-1FGG1136I
rdc03825_sym1
VCCAUX
VCCAUX
XC5VFX70T-1FGG1136I
rdc03825_sym1
GND
XC5VFX70T-1FGG1136I
rdc03825_sym1
Power and Misc FPGA Banks
VCCINT, VCCAUX, NCs, GND
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:13
TITLE: ML 507 MINI
SIZE: A3
SHEET 5 OF 22

SHEET 6

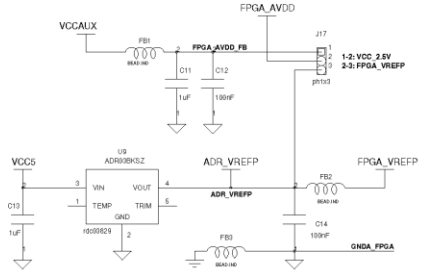
System Monitor Header



Rechargeable Battery



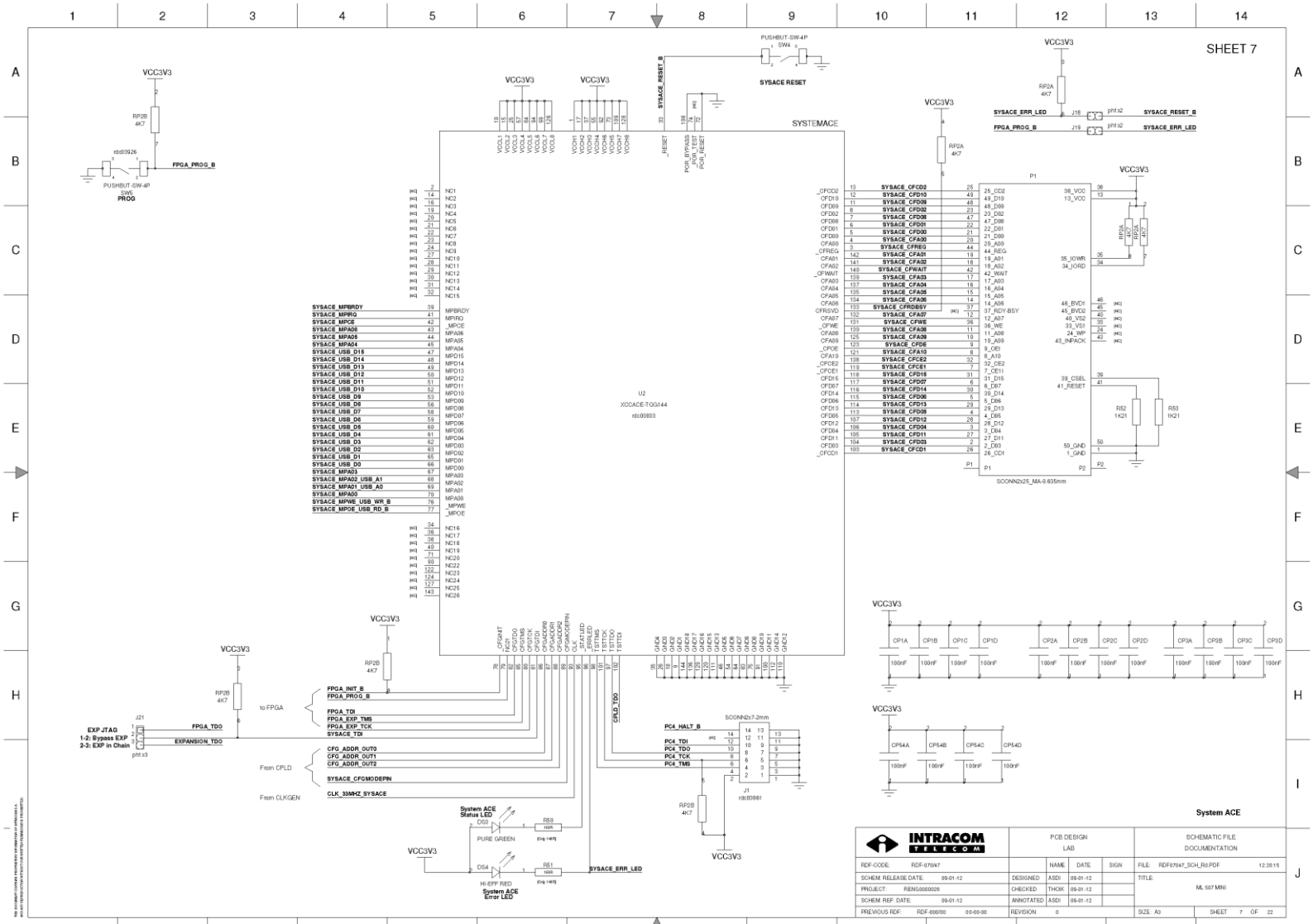
FPGA AVDD Select



Battery and System Moniter

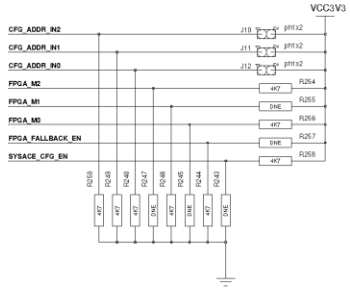
INTRACOM TELECOM	PCB DESIGN LAB				SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047		NAME	DATE	SIGN	FILE: RDF07047_SCH_R0.PDF 12:28:14
SCHEM. RELEASE DATE: 09-01-12	DESIGNED	ASDI	09-01-12		TITLE: ML 507 MINI
PROJECT: RENG0000028	CHECKED	THOM	09-01-12		
SCHEM. REF. DATE: 09-01-12	ANNOTATED	ASDI	09-01-12		
PREVIOUS RDF: RDF-000/00 00-00-00	REVISION: 0				SIZE: A3 SHEET 6 OF 22

THIS DOCUMENT CONTAINS INFORMATION PROPRIETARY OF INTRACOM S.A. AND NOT TO BE DISCLOSED OR USED EXCEPT IN ACCORDANCE WITH APPROPRIATE AGREEMENT

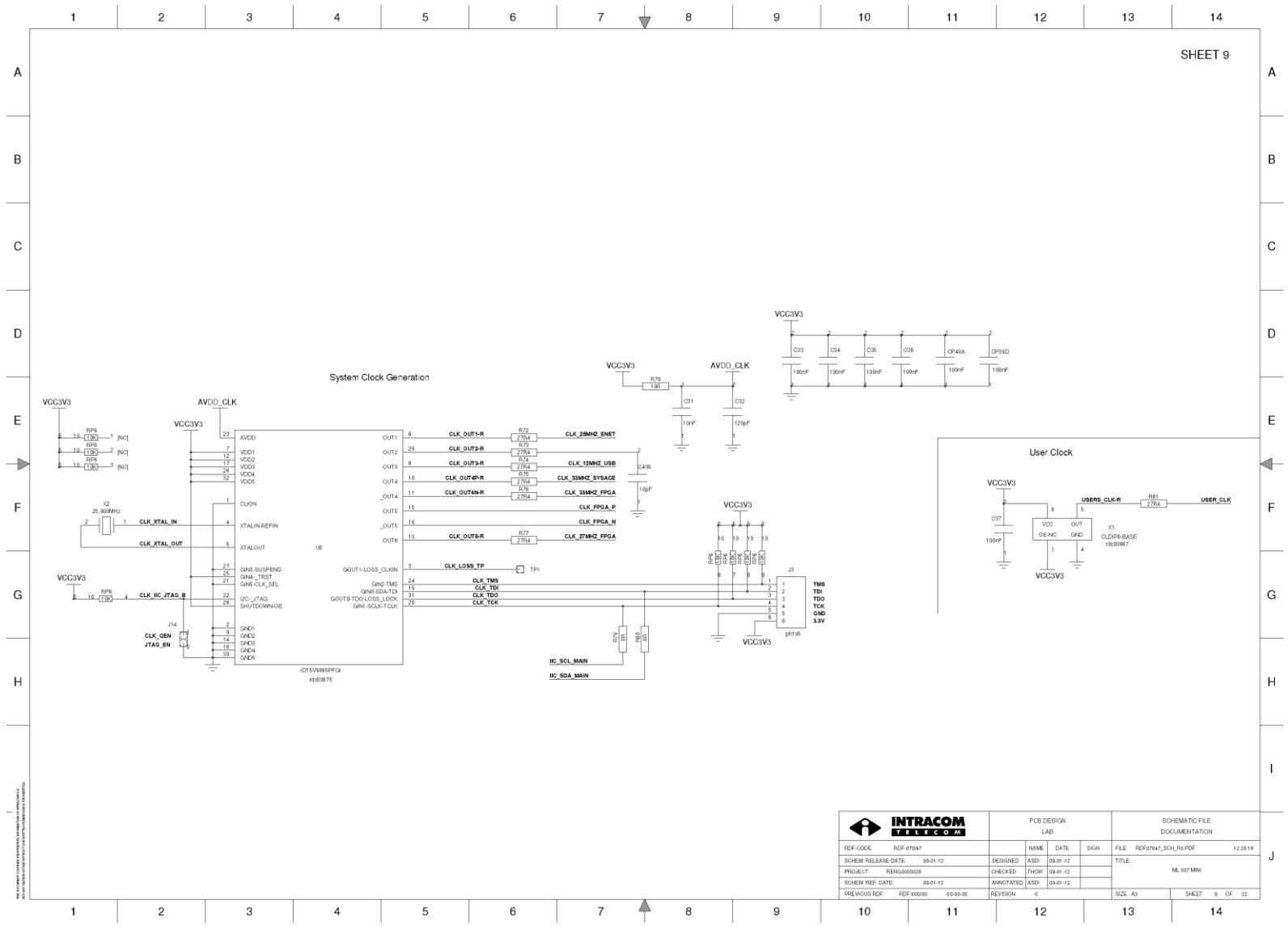

SHEET 7
SYSTEMACE
U2
XCCACE-TQG144
rdc03893
SYSACE RESET
PUSHBUT-SW-4P
SW4
PUSHBUT-SW-4P
SW5
PROG
FPGA_PROG_B
SYSACE_RESET_B
SYSACE_ERR_LED
SYSACE_MPBRDY
SYSACE_MPIRQ
SYSACE_MPCE
SYSACE_MPA06
SYSACE_MPA05
SYSACE_MPA04
SYSACE_USB_D15
SYSACE_USB_D14
SYSACE_USB_D13
SYSACE_USB_D12
SYSACE_USB_D11
SYSACE_USB_D10
SYSACE_USB_D9
SYSACE_USB_D8
SYSACE_USB_D7
SYSACE_USB_D6
SYSACE_USB_D5
SYSACE_USB_D4
SYSACE_USB_D3
SYSACE_USB_D2
SYSACE_USB_D1
SYSACE_USB_D0
SYSACE_MPA03
SYSACE_MPA02_USB_A1
SYSACE_MPA01_USB_A0
SYSACE_MPA00
SYSACE_MPWE_USB_WR_B
SYSACE_MPOE_USB_RD_B
SYSACE_CFCD2
SYSACE_CFCD10
SYSACE_CFCD09
SYSACE_CFCD02
SYSACE_CFCD08
SYSACE_CFCD01
SYSACE_CFCD00
SYSACE_CFA00
SYSACE_CFREG
SYSACE_CFA01
SYSACE_CFA02
SYSACE_CFWAIT
SYSACE_CFA03
SYSACE_CFA04
SYSACE_CFA05
SYSACE_CFA06
SYSACE_CFRDBSY
SYSACE_CFA07
SYSACE_CFWE
SYSACE_CFA08
SYSACE_CFA09
SYSACE_CFOE
SYSACE_CFA10
SYSACE_CFCE2
SYSACE_CFCE1
SYSACE_CFD15
SYSACE_CFD07
SYSACE_CFD14
SYSACE_CFD06
SYSACE_CFD13
SYSACE_CFD05
SYSACE_CFD12
SYSACE_CFD04
SYSACE_CFD11
SYSACE_CFD03
SYSACE_CFCD1
SCONN2x25_MA-0.635mm
FPGA_INIT_B
FPGA_PROG_B
FPGA_TDI
FPGA_EXP_TMS
FPGA_EXP_TCK
SYSACE_TDI
to FPGA
FPGA_TDO
EXPANSION_TDO
EXP JTAG
1-2: Bypass EXP
2-3: EXP in Chain
From CPLD
CFG_ADDR_OUT0
CFG_ADDR_OUT1
CFG_ADDR_OUT2
SYSACE_CFGMODEPIN
From CLKGEN
CLK_33MHZ_SYSACE
CPLD_TDO
PC4_HALT_B
PC4_TDI
PC4_TDO
PC4_TCK
PC4_TMS
SCONN2x7-2mm
J1
rdc03861
System ACE
Status LED
PURE GREEN
System ACE
Error LED
HI-EFF RED
SYSACE_ERR_LED
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-070/47
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000026
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:15
TITLE: ML 507 MINI
SIZE: A3
SHEET 7 OF 22
System ACE

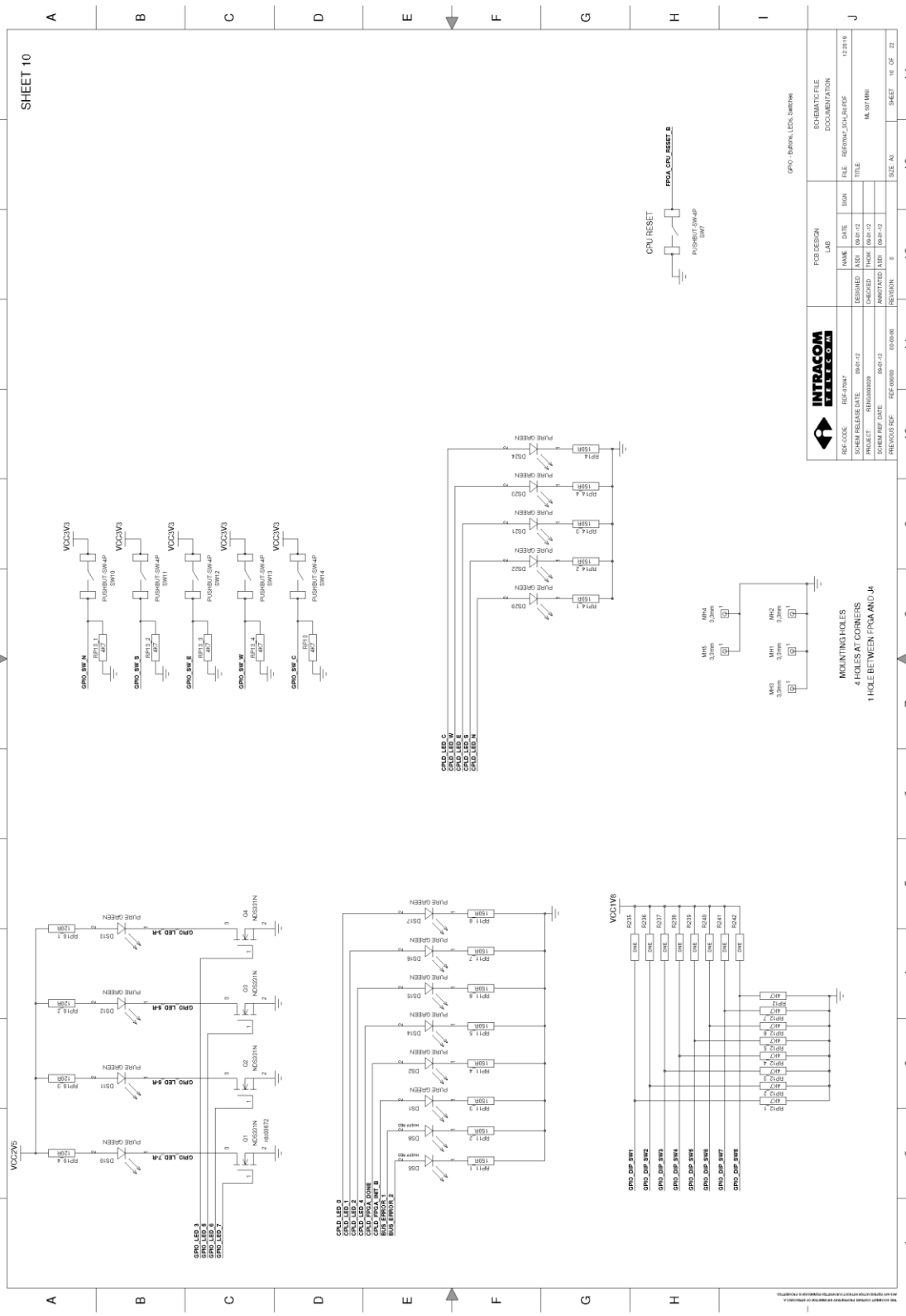
Confing DIP switches

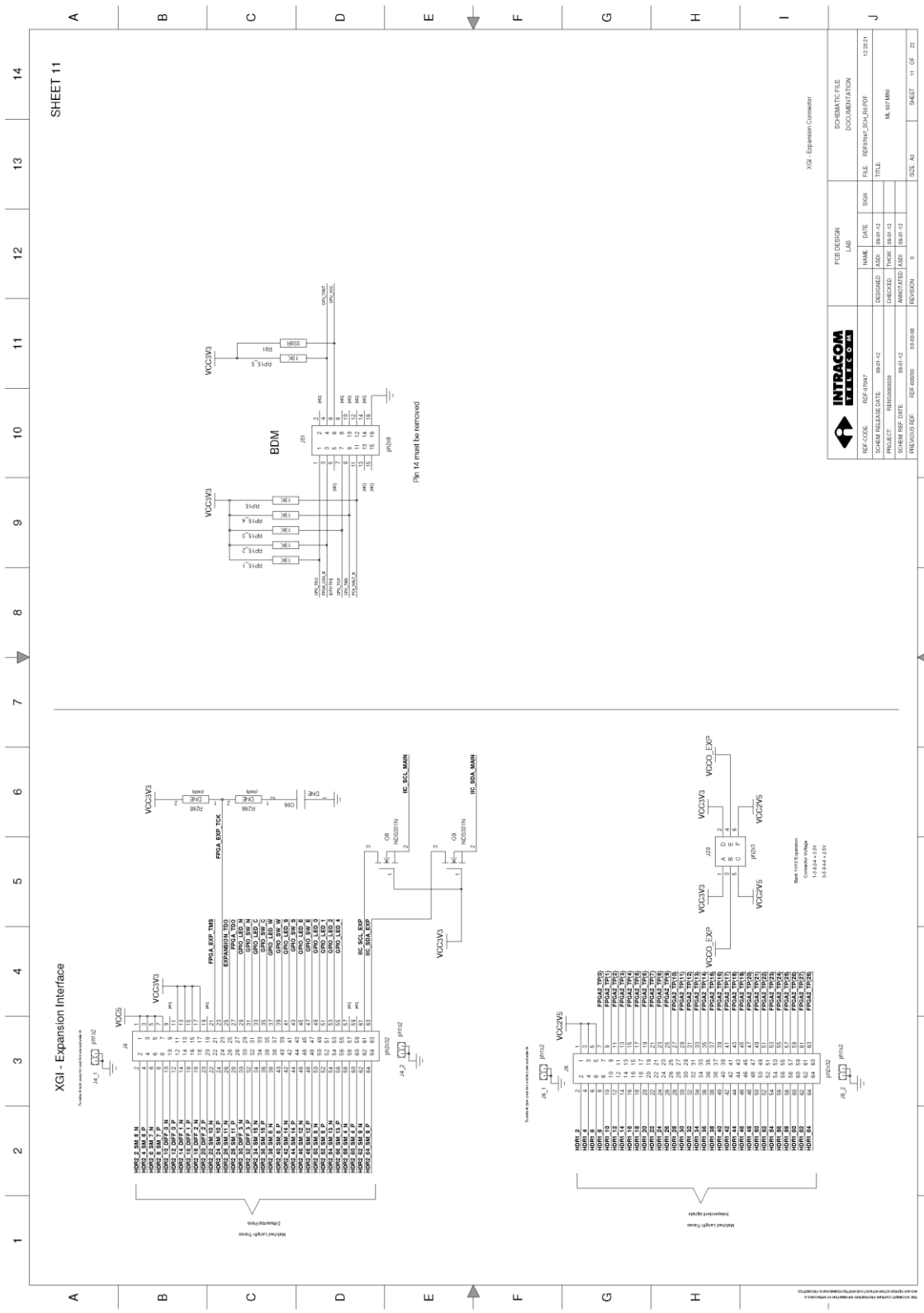
Sysace Confing, Mode pins



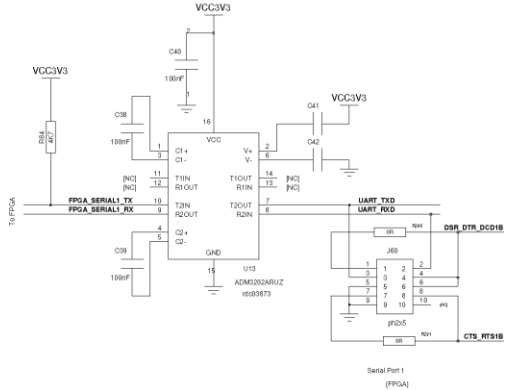
INTRACOM TELECOM		PCB DESIGN LAB			SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047		NAME	DATE	SIGN	FILE: RDF07047_SCH_R8.PDF 12:28:17
SCHEM. RELEASE DATE: 09-01-12	DESIGNED	ASDI	09-01-12		TITLE: ML 507 MINI
PROJECT: RENG0000028	CHECKED	THOM	09-01-12		
SCHEM. REF. DATE: 09-01-12	ANNOTATED	ASDI	09-01-12		
PREVIOUS RDF: RDF-000/00 00-00-00	REVISION: 0				SIZE: A3 SHEET 8 OF 22


SHEET 9
System Clock Generation
User Clock
VCC3V3
AVDD_CLK
CLK_XTAL_IN
CLK_XTAL_OUT
CLK_IIC_JTAG_B
CLK_GEN
JTAG_EN
CLK_OUT1-R
CLK_OUT2-R
CLK_OUT3-R
CLK_OUT4P-R
CLK_OUT4N-R
CLK_OUT5-R
CLK_LOSS_TP
CLK_25MHZ_ENET
CLK_12MHZ_USB
CLK_33MHZ_SYSACE
CLK_33MHZ_FPGA
CLK_FPGA_P
CLK_FPGA_N
CLK_27MHZ_FPGA
CLK_TMS
CLK_TDI
CLK_TDO
CLK_TCK
IIC_SCL_MAIN
IIC_SDA_MAIN
USERS_CLK-R
USER_CLK
TMS
TDI
TDO
TCK
GND
3.3V
IDT 6V9885PFGI
rdc09875
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION: 0
FILE: RDF07047_SCH_R0.PDF 12:28:16
TITLE: ML 507 MINI
SIZE: A3
SHEET 9 OF 22


SHEET 10
GPIO - Buttons, LEDs, Switches
CPU RESET
FPGA_CPU_RESET_B
PUSHBUT-SW-4P
SW7
MOUNTING HOLES
4 HOLES AT CORNERS
1 HOLE BETWEEN FPGA AND J4
MH1
MH2
MH3
MH4
MH5
3,3mm
PULSE GREEN
VCC3V3
VCC2V5
VCC1V8
GPIO_SW_N
GPIO_SW_S
GPIO_SW_E
GPIO_SW_W
GPIO_SW_C
GPIO_LED_0
GPIO_LED_1
GPIO_LED_2
GPIO_LED_3
GPIO_LED_4
GPIO_LED_5
GPIO_LED_6
GPIO_LED_7
GPIO_DIP_SW1
GPIO_DIP_SW2
GPIO_DIP_SW3
GPIO_DIP_SW4
GPIO_DIP_SW5
GPIO_DIP_SW6
GPIO_DIP_SW7
GPIO_DIP_SW8
CPLD_LED_0
CPLD_LED_1
CPLD_LED_2
CPLD_LED_3
CPLD_FPGA_DONE
CPLD_FPGA_INIT_B
BUS_ERROR_1
BUS_ERROR_2
SCHEMATIC FILE
DOCUMENTATION
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC_SCH_BUPDF
ML 507 MMI
SHEET 16 OF 33

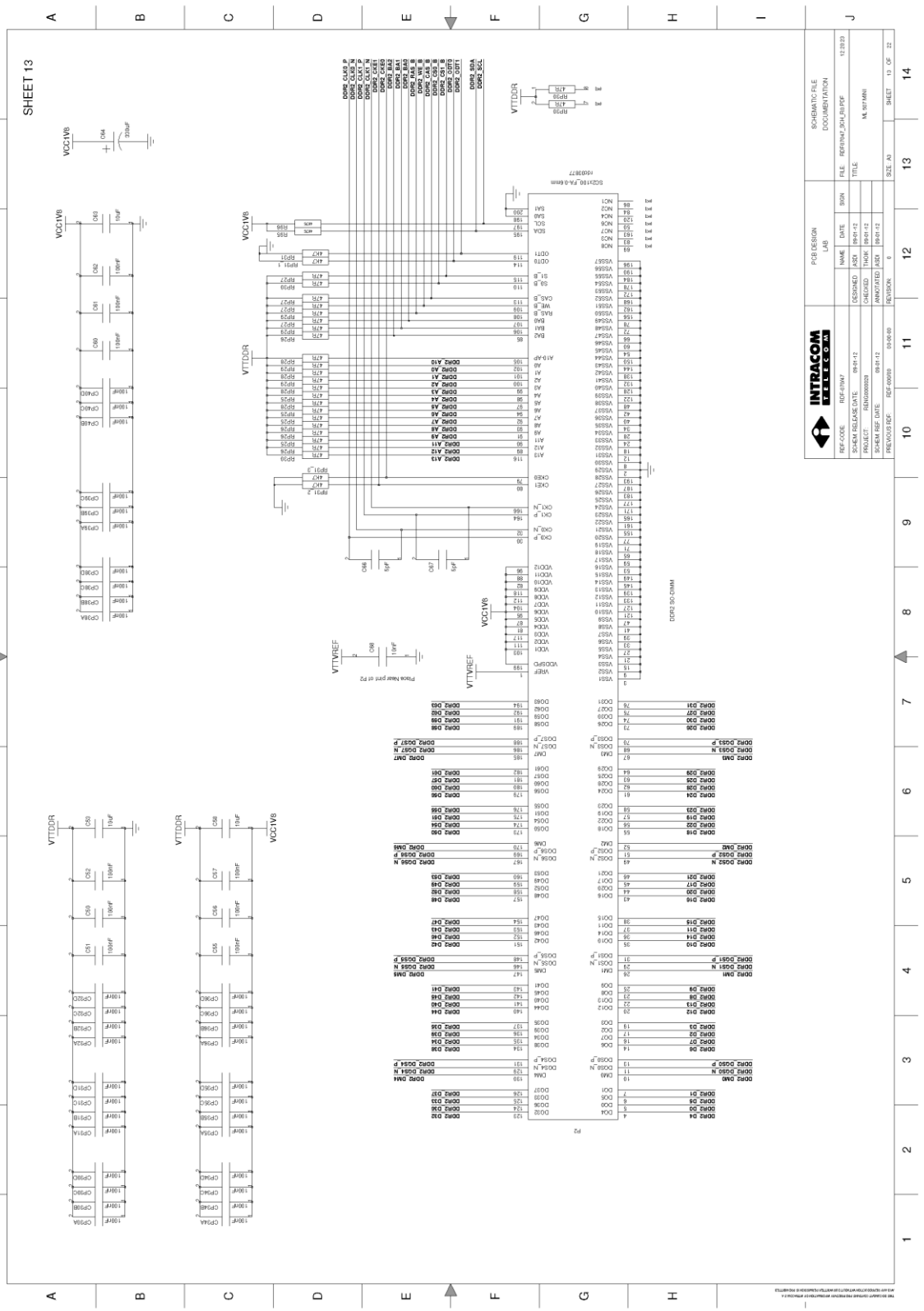

SHEET 11
XGI - Expansion Interface
BDM
Pin 14 must be removed
XGI - Expansion Connector
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
ML 507 MINI

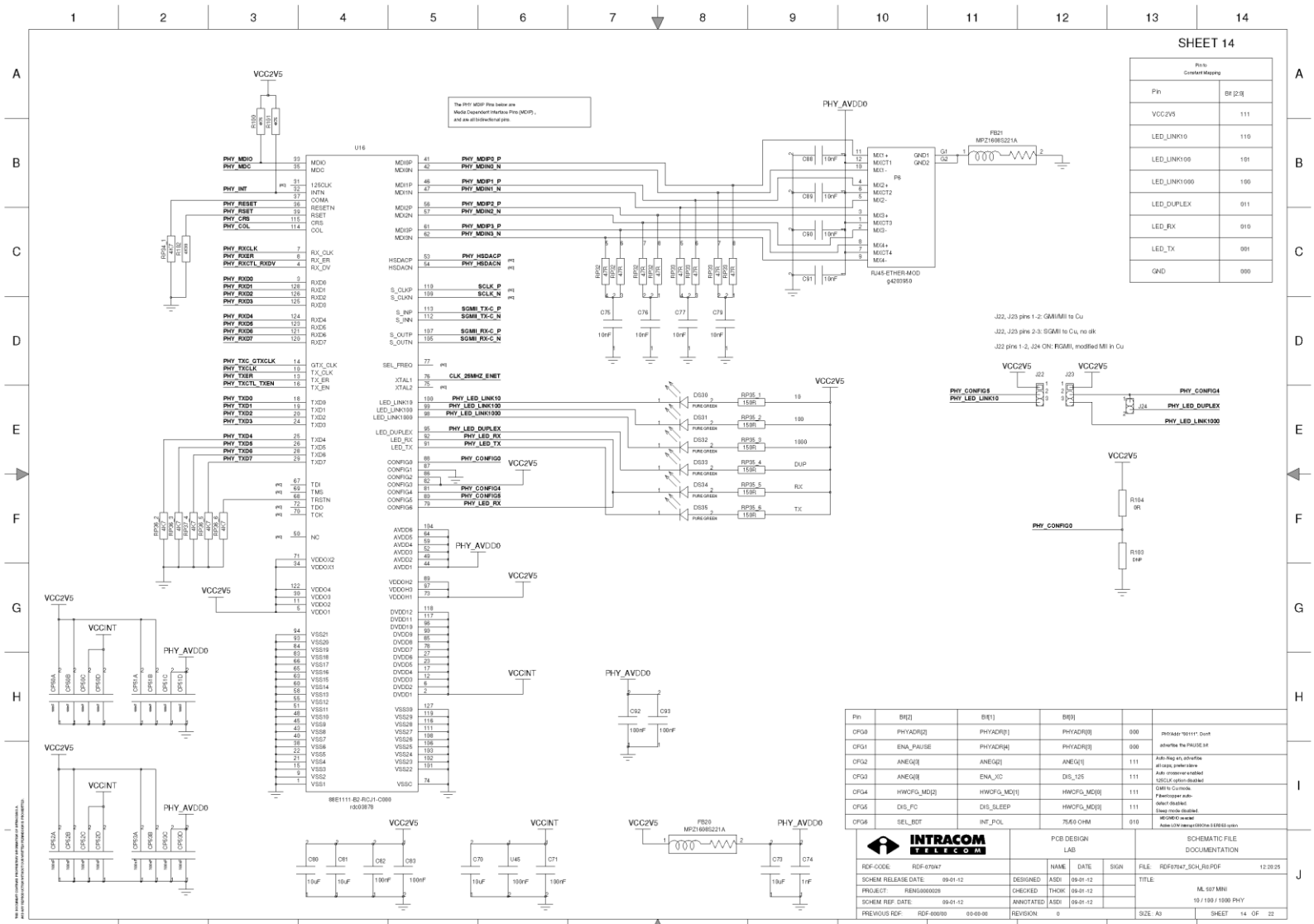
SHEET 12

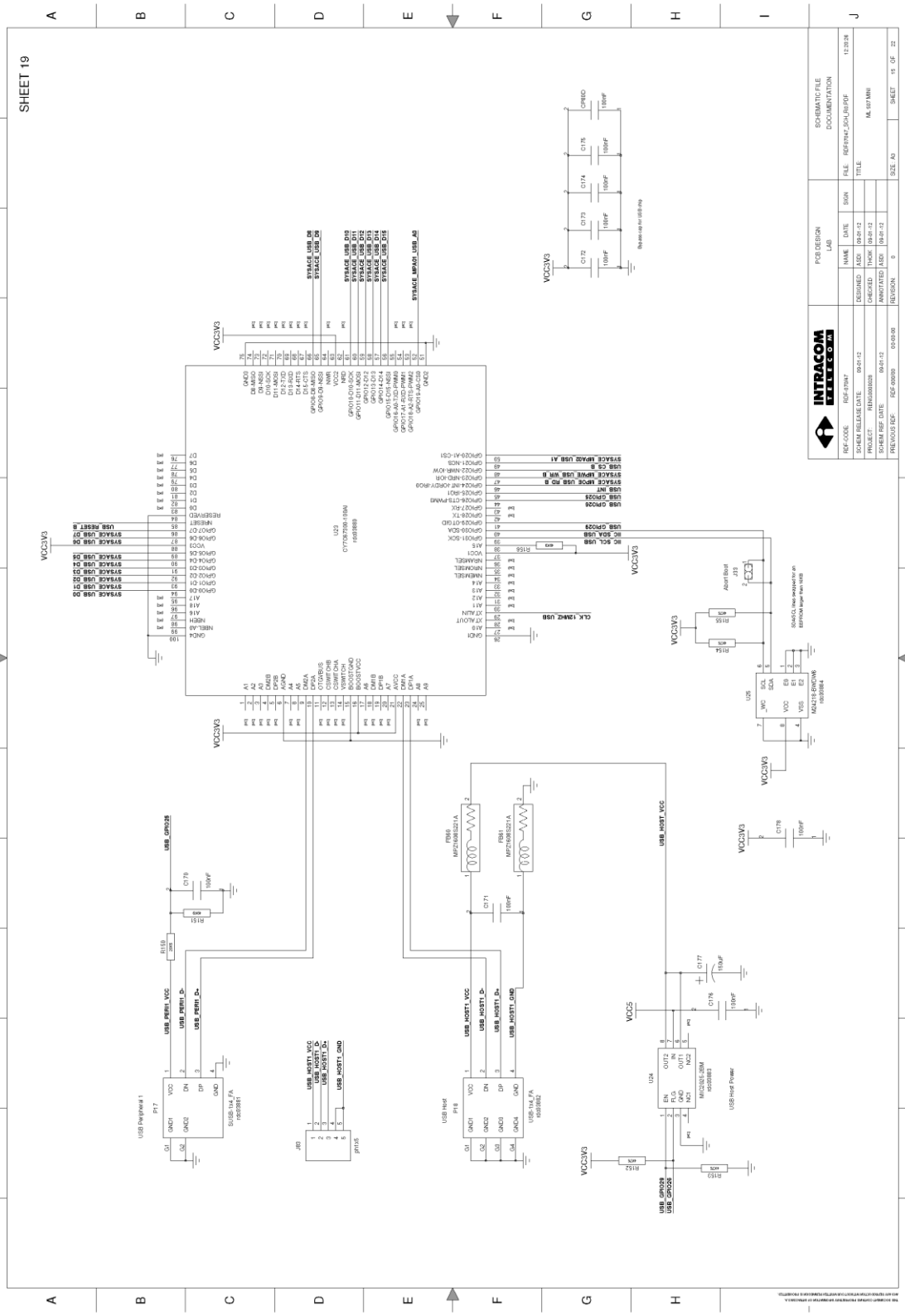


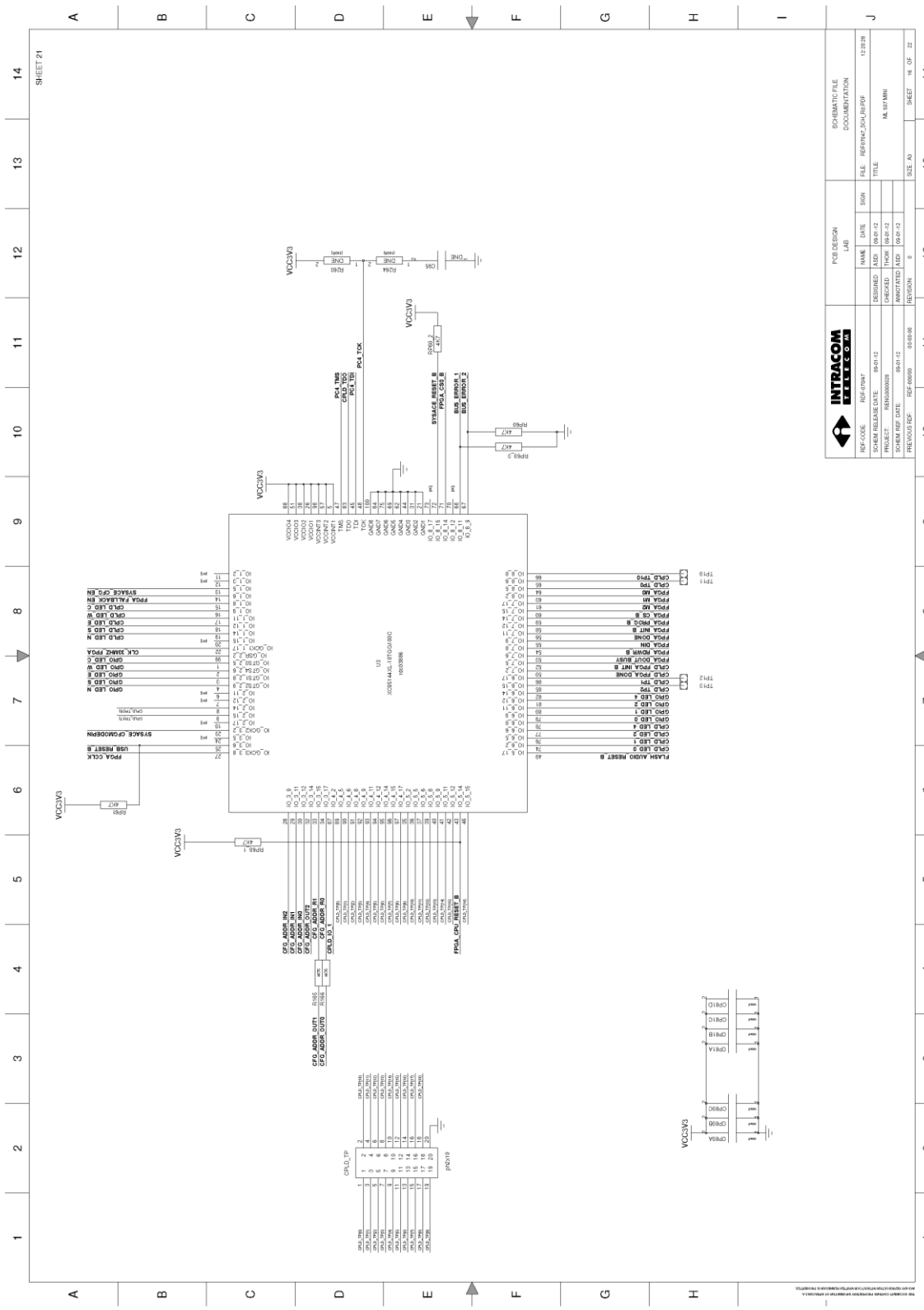
DTE (Serial Host)

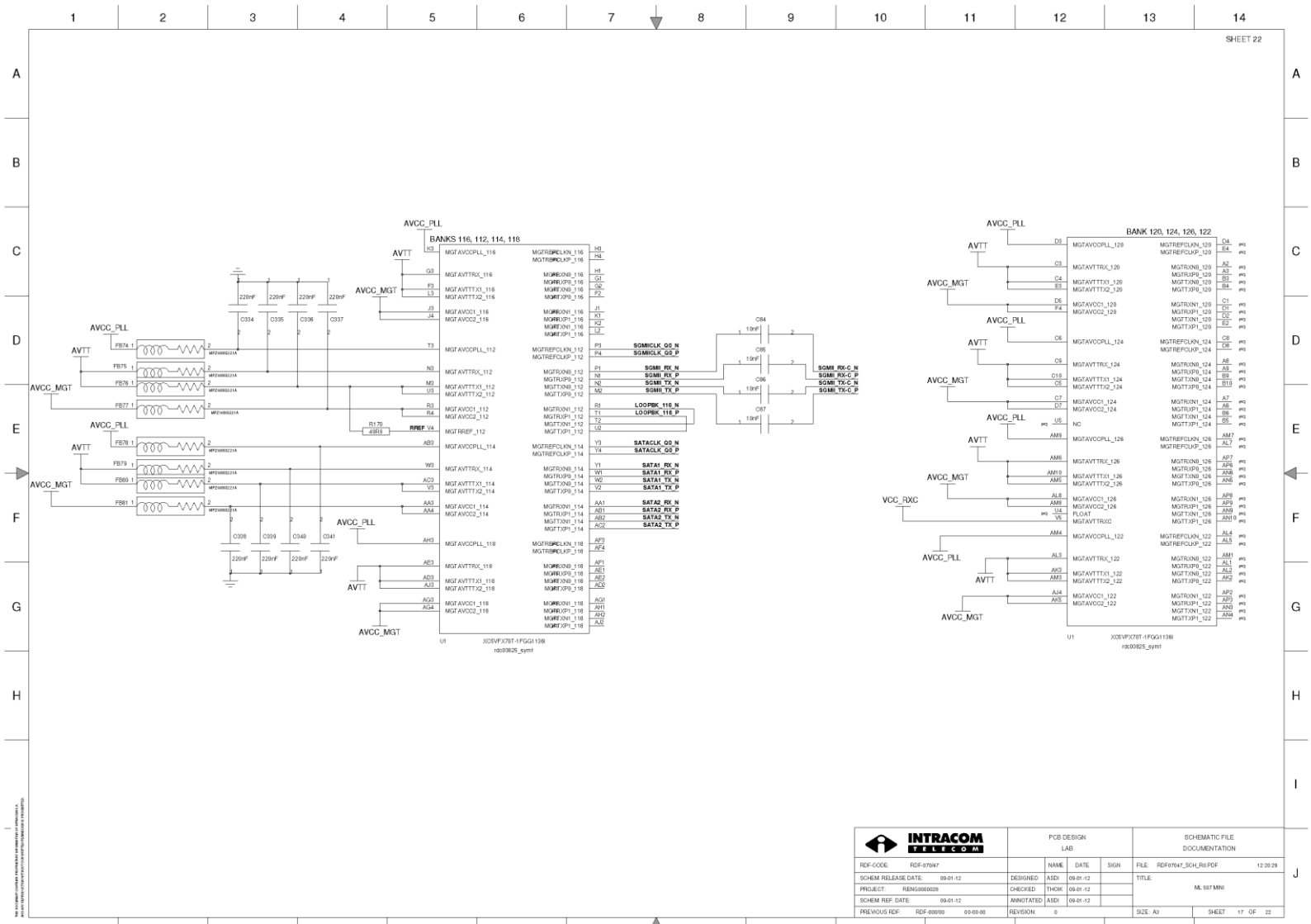
INTRACOM TELECOM	PCB DESIGN LAB				SCHEMATIC FILE DOCUMENTATION	
RDF-CODE: RDF-07047		NAME	DATE	SIGN	FILE: RDF07047_SCH_R0.PDF	12:28:22
SCHEM. RELEASE DATE: 09-01-12	DESIGNED	ASDI	09-01-12		TITLE: ML 507 MINI	
PROJECT: RENG0000026	CHECKED	THOM	09-01-12			
SCHEM. REF. DATE: 09-01-12	ANNOTATED	ASDI	09-01-12			
PREVIOUS RDF: RDF-000/00 00-00-00	REVISION: 0				SIZE: A3	SHEET 12 OF 22

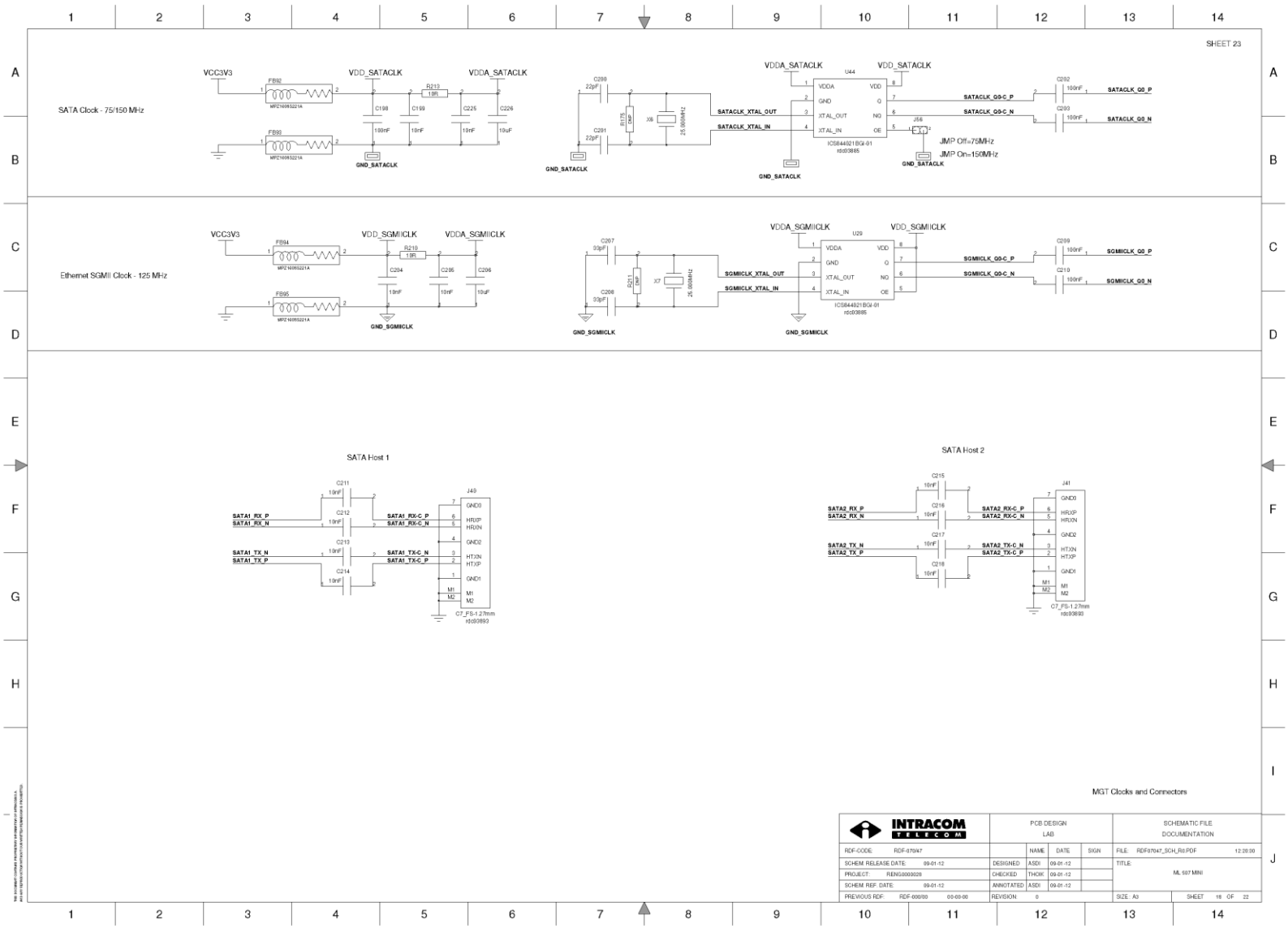

SHEET 13
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
ML 907 MINI
DDR2 SO-DIMM
VCC1V8
VTTDDR
VTTVREF


SHEET 14
U16
88E1111-B2-RCJ1-C000
rd:009870
The PHY' MDIP Pins below are
Media Dependent Interface Pins (MDIP),
and are all bidirectional pins.
PHY_AVDD0
VCC2V5
VCCINT
RJ45-ETHER-MOD
g4280950
MPZ1608S221A
J22, J23 pins 1-2: GMII/MII to Cu
J22, J23 pins 2-3: SGMII to Cu, no clk
J22 pins 1-2, J24 ON: RGMII, modified MII in Cu
Pin to
Constant Mapping
Pin
Bit [2:0]
VCC2V5
111
LED_LINK10
110
LED_LINK100
101
LED_LINK1000
100
LED_DUPLEX
011
LED_RX
010
LED_TX
001
GND
000
Pin
Bit[2]
Bit[1]
Bit[0]
CFG0
PHYADR[2]
PHYADR[1]
PHYADR[0]
000
PHYAddr "00111". Don't
advertise the PAUSE bit
CFG1
ENA_PAUSE
PHYADR[4]
PHYADR[3]
000
CFG2
ANEG[0]
ANEG[2]
ANEG[1]
111
Auto-Neg en, advertise
all caps, prefer slave
Auto crossover enabled
125CLK option disabled
CFG3
ANEG[3]
ENA_XC
DIS_125
111
CFG4
HWCFG_MD[2]
HWCFG_MD[1]
HWCFG_MD[0]
111
GMII to Cu mode.
Fiber/copper auto-
detect disabled.
CFG5
DIS_FC
DIS_SLEEP
HWCFG_MD[3]
111
Sleep mode disabled
CFG6
SEL_BDT
INT_POL
75/5.0 OHM
010
MDI/MDIX mode0
Active LOW interrupt/1000bit 5 EDGE option
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000026
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
NAME DATE SIGN
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:25
TITLE:
ML 507 MINI
10 / 100 / 1000 PHY
SIZE: A3
SHEET 14 OF 22


SHEET 19
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
ML 687 MINI
SHEET 15 OF 33

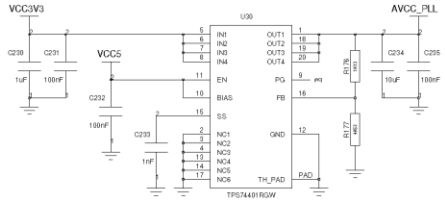



SHEET 22
BANKS 116, 112, 114, 118
BANK 120, 124, 126, 122
AVCC_PLL
AVTT
AVCC_MGT
VCC_RXC
SGMIICLK_G0_N
SGMIICLK_G0_P
SGMII_RX_N
SGMII_RX_P
SGMII_TX_N
SGMII_TX_P
LOOPBK_116_N
LOOPBK_116_P
SATACLK_G0_N
SATACLK_G0_P
SATA1_RX_N
SATA1_RX_P
SATA1_TX_N
SATA1_TX_P
SATA2_RX_N
SATA2_RX_P
SATA2_TX_N
SATA2_TX_P
SGMII_RX-C_N
SGMII_RX-C_P
SGMII_TX-C_N
SGMII_TX-C_P
RREF
R170
49R9
U1
XC5VFX70T-1FGG1136I
rdc03625_symt
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: REING0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:29
TITLE: ML 507 MINI
SIZE: A3
SHEET 17 OF 22

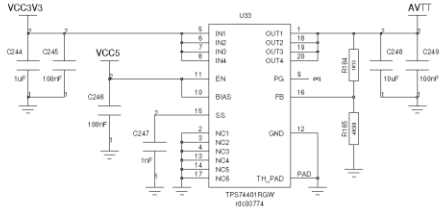

SHEET 23
SATA Clock - 75/150 MHz
VCC3V3
VDD_SATACLK
VDDA_SATACLK
FB92
FB93
MPZ1608S221A
R210
10R
C198
C199
C225
C226
100nF
10nF
10uF
GND_SATACLK
C200
22pF
C201
22pF
R175
0R0
X6
25.000MHz
SATACLK_XTAL_OUT
SATACLK_XTAL_IN
U44
VDDA
GND
XTAL_OUT
XTAL_IN
VDD
Q
NQ
OE
IC5844021BGI-01
rdc03885
J56
JMP Off=75MHz
JMP On=150MHz
SATACLK_G0-C_P
SATACLK_G0-C_N
C202
C203
100nF
SATACLK_G0_P
SATACLK_G0_N
Ethernet SGMII Clock - 125 MHz
VCC3V3
VDD_SGMIICLK
VDDA_SGMIICLK
FB94
FB95
MPZ1608S221A
R210
10R
C204
C205
C206
10nF
10uF
GND_SGMIICLK
C207
33pF
C208
33pF
R221
0R0
X7
25.000MHz
SGMIICLK_XTAL_OUT
SGMIICLK_XTAL_IN
U29
IC5844021BGI-01
rdc03885
SGMIICLK_G0-C_P
SGMIICLK_G0-C_N
C209
C210
100nF
SGMIICLK_G0_P
SGMIICLK_G0_N
SATA Host 1
C211
C212
C213
C214
10nF
J40
GND0
HRXP
HRXN
GND2
HTXN
HTXP
GND1
M1
M2
C7_FS-1.27mm
rdc03893
SATA1_RX_P
SATA1_RX_N
SATA1_TX_N
SATA1_TX_P
SATA1_RX-C_P
SATA1_RX-C_N
SATA1_TX-C_N
SATA1_TX-C_P
SATA Host 2
C215
C216
C217
C218
10nF
J41
SATA2_RX_P
SATA2_RX_N
SATA2_TX_N
SATA2_TX_P
SATA2_RX-C_P
SATA2_RX-C_N
SATA2_TX-C_N
SATA2_TX-C_P
MGT Clocks and Connectors
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000026
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
NAME DATE SIGN
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:20:00
TITLE: ML 507 MINI
SIZE: A3
SHEET 16 OF 22

SHEET 24

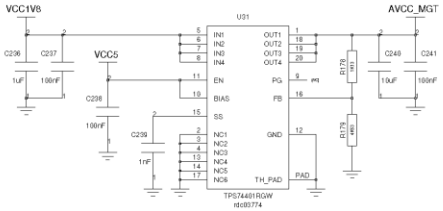
MGT PLL Regulator



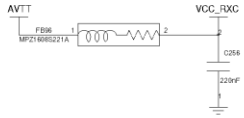
MGT VTT Regulator



MGT AVCC Regulator



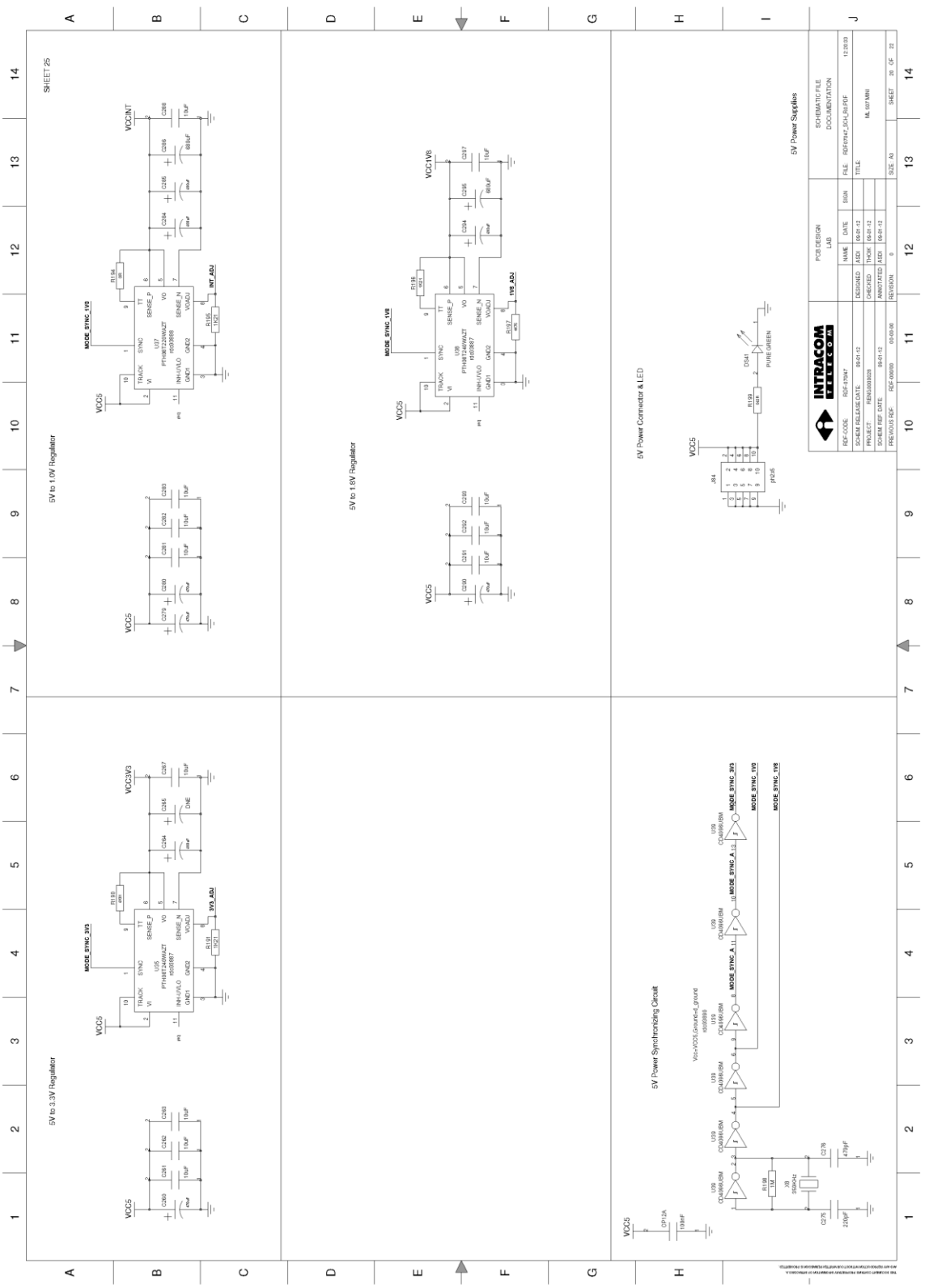
MGT RXC Regulator

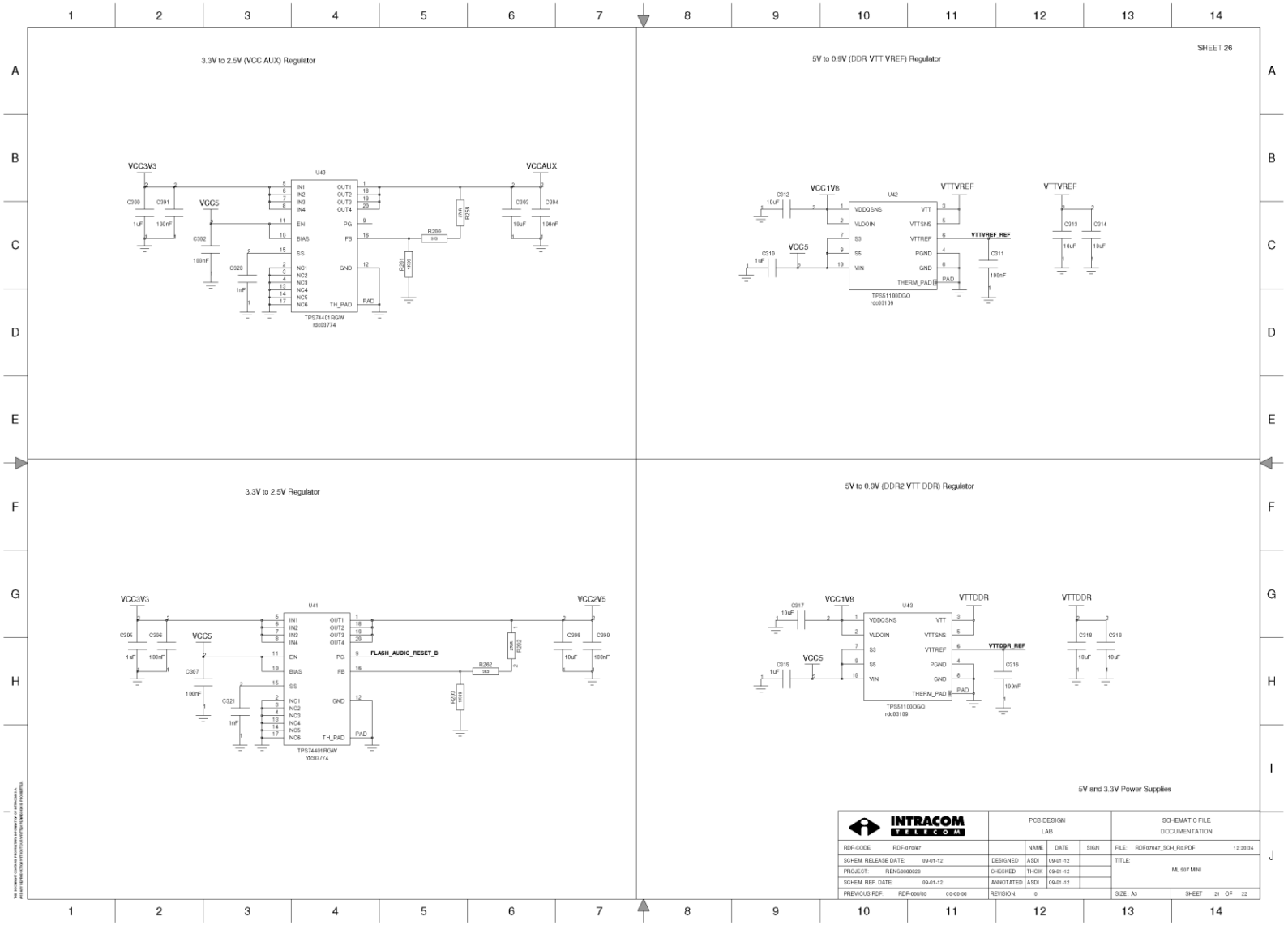


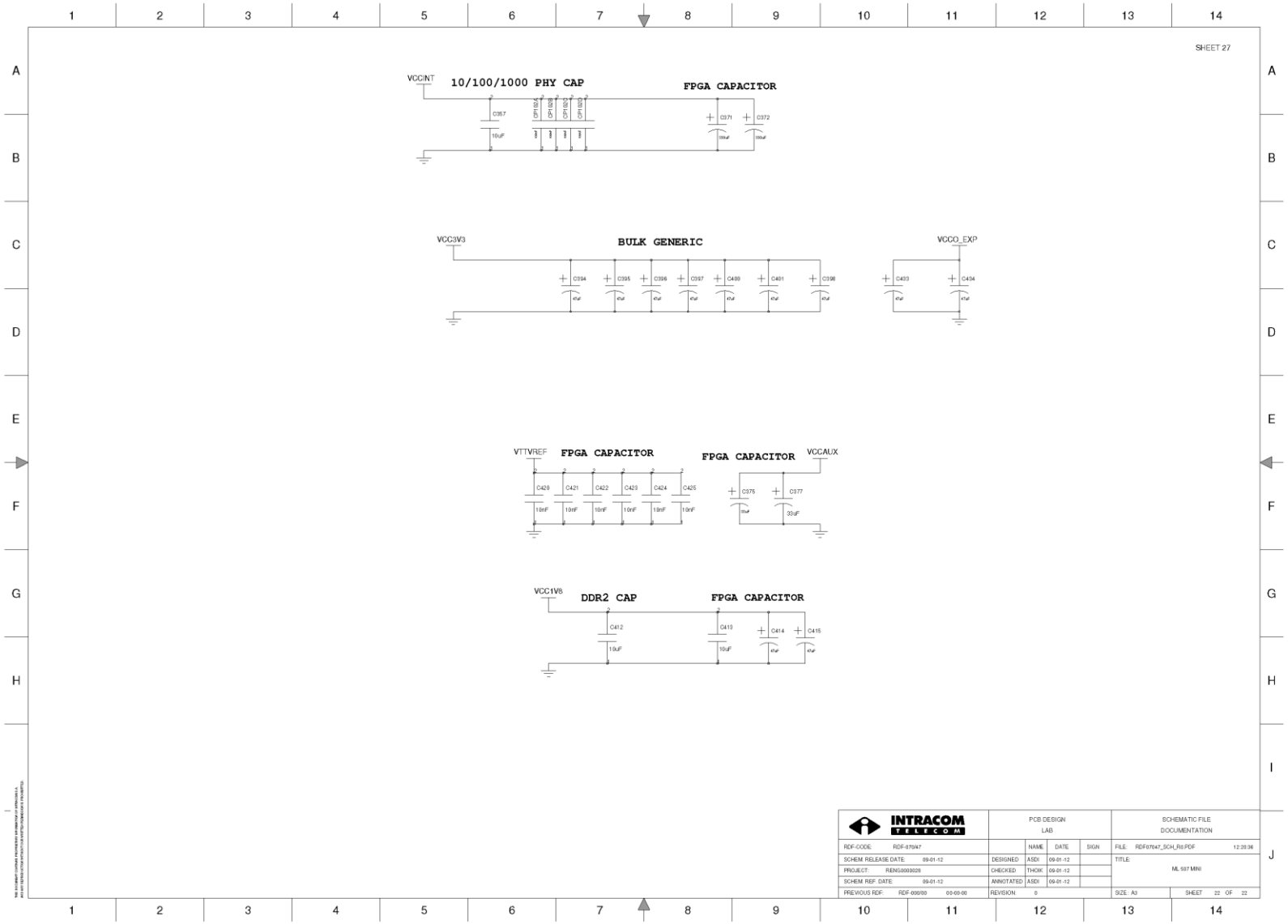
MGT Power Supplies

INTRACOM TELECOM	PCB DESIGN LAB				SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047		NAME	DATE	SIGN	FILE: RDF07047_SCH_R0.PDF 12:28:32
SCHEM. RELEASE DATE: 09-01-12	DESIGNED	ASDI	09-01-12		TITLE: ML 507 MINI
PROJECT: RENG0000026	CHECKED	THOM	09-01-12		
SCHEM. REF. DATE: 09-01-12	ANNOTATED	ASDI	09-01-12		
PREVIOUS RDF: RDF-000/00 00-00-00	REVISION: 0				SIZE: A3 SHEET 19 OF 22

THE INFORMATION CONTAINED IN THIS DRAWING IS THE PROPERTY OF INTRACOM S.A. ANY UNAUTHORIZED USE, DISCLOSURE OR REPRODUCTION IS PROHIBITED.


SHEET 25
5V to 3.3V Regulator
5V to 1.0V Regulator
5V to 1.6V Regulator
5V Power Connector & LED
5V Power Synchronizing Circuit
5V Power Supplies
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
TITLE
ML 607 MINI
SHEET 25 OF 33


SHEET 26
3.3V to 2.5V (VCC AUX) Regulator
5V to 0.9V (DDR VTT VREF) Regulator
3.3V to 2.5V Regulator
5V to 0.9V (DDR2 VTT DDR) Regulator
VCC3V3
VCCAUX
VCC5
VCC1V8
VTTVREF
VTTVREF_REF
VCC2V5
VTTDDR
VTTDDR_REF
FLASH_AUDIO_RESET_B
U40
U41
U42
U43
TPS74401RGW
rdc03774
TPS51100DGQ
rdc03109
5V and 3.3V Power Supplies
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-07047
SCHEM. RELEASE DATE: 09-01-12
PROJECT: RENG0000028
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:04
TITLE: ML 507 MINI
SIZE: A3
SHEET 21 OF 22


SHEET 27
VCCINT
10/100/1000 PHY CAP
FPGA CAPACITOR
C357
10uF
CPI 0204
CPI 0204
CPI 0204
CPI 0204
C371
C372
VCC3V3
BULK GENERIC
C394
C395
C396
C397
C400
C401
C398
VCCO_EXP
C433
C434
VTTVREF
FPGA CAPACITOR
C420
C421
C422
C423
C424
C425
10nF
FPGA CAPACITOR
VCCAUX
C375
C377
33uF
VCC1V8
DDR2 CAP
FPGA CAPACITOR
C412
10uF
C410
C414
C415
INTRACOM TELECOM
PCB DESIGN LAB
SCHEMATIC FILE DOCUMENTATION
RDF-CODE: RDF-070/47
SCHEM. RELEASE DATE: 09-01-12
PROJECT: REING0000026
SCHEM. REF. DATE: 09-01-12
PREVIOUS RDF: RDF-000/00 00-00-00
NAME DATE SIGN
DESIGNED ASDI 09-01-12
CHECKED THOM 09-01-12
ANNOTATED ASDI 09-01-12
REVISION 0
FILE: RDF07047_SCH_R0.PDF 12:28:06
TITLE: ML 507 MINI
SIZE: A3
SHEET 22 OF 22

ΠΑΡΑΡΤΗΜΑ Β: Κώδικας VHDL της μονάδας mainfft core (user_logic.vhd)

```
------------------------------------------------------------------------------
-- user_logic.vhd - entity/architecture pair
------------------------------------------------------------------------------
-- Filename:          user_logic.vhd
-- Version:           1.00.a
-- Description:       User logic.
-- Date:              Tue Dec 04 14:26:38 2012 (by Create and Import Peripheral Wizard)
-- VHDL Standard:     VHDL'93
------------------------------------------------------------------------------
-- Naming Conventions:
--   active low signals:                    "*_n"
--   clock signals:                         "clk", "clk_div#", "clk_#x"
--   reset signals:                         "rst", "rst_n"
--   generics:                              "C_*"
--   user defined types:                    "*_TYPE"
--   state machine next state:              "*_ns"
--   state machine current state:           "*_cs"
--   combinatorial signals:                 "*_com"
--   pipelined or register delay signals:   "*_d#"
--   counter signals:                       "*cnt*"
--   clock enable signals:                  "*_ce"
--   internal version of output port:       "*_i"
--   device pins:                           "*_pin"
--   ports:                                 "- Names begin with Uppercase"
--   processes:                             "*_PROCESS"
--   component instantiations:              "<ENTITY_>I_<#|FUNC>"
------------------------------------------------------------------------------

-- DO NOT EDIT BELOW THIS LINE --------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

library proc_common_v3_00_a;
use proc_common_v3_00_a.proc_common_pkg.all;

-- DO NOT EDIT ABOVE THIS LINE --------------------

--USER libraries added here

------------------------------------------------------------------------------
-- Entity section
------------------------------------------------------------------------------
-- Definition of Generics:
--   C_SLV_AWIDTH                -- Slave interface address bus width
--   C_SLV_DWIDTH                -- Slave interface data bus width
--   C_NUM_REG                   -- Number of software accessible registers
--   C_NUM_MEM                   -- Number of memory spaces
--   C_RDFIFO_DEPTH              -- Read FIFO depth
--   C_WRFIFO_DEPTH              -- Write FIFO depth
--
-- Definition of Ports:
--   Bus2IP_Clk                  -- Bus to IP clock
--   Bus2IP_Reset                -- Bus to IP reset
--   Bus2IP_Addr                 -- Bus to IP address bus
--   Bus2IP_CS                   -- Bus to IP chip select for user logic memory selection
--   Bus2IP_RNW                  -- Bus to IP read/not write
--   Bus2IP_Data                 -- Bus to IP data bus
--   Bus2IP_BE                   -- Bus to IP byte enables
--   Bus2IP_RdCE                 -- Bus to IP read chip enable
--   Bus2IP_WrCE                 -- Bus to IP write chip enable
--   IP2Bus_Data                 -- IP to Bus data bus
--   IP2Bus_RdAck                -- IP to Bus read transfer acknowledgement
--   IP2Bus_WrAck                -- IP to Bus write transfer acknowledgement
--   IP2Bus_Error                -- IP to Bus error response
--   IP2RFIFO_WrReq              -- IP to RFIFO : IP write request
--   IP2RFIFO_Data               -- IP to RFIFO : IP write data bus
--   IP2RFIFO_WrMark             -- IP to RFIFO : mark beginning of packet being written
--   IP2RFIFO_WrRelease          -- IP to RFIFO : return RFIFO to normal FIFO operation
--   IP2RFIFO_WrRestore          -- IP to RFIFO : restore the RFIFO to the last packet
mark
--   RFIFO2IP_WrAck              -- RFIFO to IP : RFIFO write acknowledge
--   RFIFO2IP_AlmostFull         -- RFIFO to IP : RFIFO almost full
```

```
--   RFIFO2IP_Full               -- RFIFO to IP : RFIFO full
--   RFIFO2IP_Vacancy            -- RFIFO to IP : RFIFO vacancy
--   IP2WFIFO_RdReq              -- IP to WFIFO : IP read request
--   IP2WFIFO_RdMark             -- IP to WFIFO : mark beginning of packet being read
--   IP2WFIFO_RdRelease          -- IP to WFIFO : return WFIFO to normal FIFO operation
--   IP2WFIFO_RdRestore          -- IP to WFIFO : restore the WFIFO to the last packet
mark
--   WFIFO2IP_Data               -- WFIFO to IP : WFIFO read data
--   WFIFO2IP_RdAck              -- WFIFO to IP : WFIFO read acknowledge
--   WFIFO2IP_AlmostEmpty        -- WFIFO to IP : WFIFO almost empty
--   WFIFO2IP_Empty              -- WFIFO to IP : WFIFO empty
--   WFIFO2IP_Occupancy          -- WFIFO to IP : WFIFO occupancy
------------------------------------------------------------------------------

entity user_logic is
  generic
  (
    -- ADD USER GENERICS BELOW THIS LINE ---------------
    --USER generics added here
    -- ADD USER GENERICS ABOVE THIS LINE ---------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol parameters, do not add to or delete
    C_SLV_AWIDTH                   : integer              := 32;
    C_SLV_DWIDTH                   : integer              := 32;
    C_NUM_REG                      : integer              := 5;
    C_NUM_MEM                      : integer              := 3;
    C_RDFIFO_DEPTH                 : integer              := 1024;
    C_WRFIFO_DEPTH                 : integer              := 1024
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
  );
  port
  (
    -- ADD USER PORTS BELOW THIS LINE ------------------
    --USER ports added here
       ADC_Data : in std_logic_vector(15 downto 0); --adc data bus is permanetly connected
to slv_reg4
       ADC_EOC : in std_logic;
       ADC_CS  : out std_logic;
       ADC_RC  : out std_logic;
       ADC_Reset      : out std_logic;
    -- ADD USER PORTS ABOVE THIS LINE ------------------

    -- DO NOT EDIT BELOW THIS LINE ---------------------
    -- Bus protocol ports, do not add to or delete
    Bus2IP_Clk                     : in  std_logic;
    Bus2IP_Reset                   : in  std_logic;
    Bus2IP_Addr                    : in  std_logic_vector(0 to C_SLV_AWIDTH-1);
    Bus2IP_CS                      : in  std_logic_vector(0 to C_NUM_MEM-1);
    Bus2IP_RNW                     : in  std_logic;
    Bus2IP_Data                    : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    Bus2IP_BE                      : in  std_logic_vector(0 to C_SLV_DWIDTH/8-1);
    Bus2IP_RdCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    Bus2IP_WrCE                    : in  std_logic_vector(0 to C_NUM_REG-1);
    IP2Bus_Data                    : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2Bus_RdAck                   : out std_logic;
    IP2Bus_WrAck                   : out std_logic;
    IP2Bus_Error                   : out std_logic;
    IP2RFIFO_WrReq                 : out std_logic;
    IP2RFIFO_Data                  : out std_logic_vector(0 to C_SLV_DWIDTH-1);
    IP2RFIFO_WrMark                : out std_logic;
    IP2RFIFO_WrRelease             : out std_logic;
    IP2RFIFO_WrRestore             : out std_logic;
    RFIFO2IP_WrAck                 : in  std_logic;
    RFIFO2IP_AlmostFull            : in  std_logic;
    RFIFO2IP_Full                  : in  std_logic;
    RFIFO2IP_Vacancy               : in  std_logic_vector(0 to log2(C_RDFIFO_DEPTH));
    IP2WFIFO_RdReq                 : out std_logic;
    IP2WFIFO_RdMark                : out std_logic;
    IP2WFIFO_RdRelease             : out std_logic;
    IP2WFIFO_RdRestore             : out std_logic;
    WFIFO2IP_Data                  : in  std_logic_vector(0 to C_SLV_DWIDTH-1);
    WFIFO2IP_RdAck                 : in  std_logic;
    WFIFO2IP_AlmostEmpty           : in  std_logic;
    WFIFO2IP_Empty                 : in  std_logic;
    WFIFO2IP_Occupancy             : in  std_logic_vector(0 to log2(C_WRFIFO_DEPTH))
    -- DO NOT EDIT ABOVE THIS LINE ---------------------
```

```
  );

  attribute SIGIS : string;
  attribute SIGIS of Bus2IP_Clk    : signal is "CLK";
  attribute SIGIS of Bus2IP_Reset  : signal is "RST";

end entity user_logic;

------------------------------------------------------------------------------
-- Architecture section
------------------------------------------------------------------------------

architecture IMP of user_logic is

  --USER signal declarations added here, as needed for user logic
  -- Signals for read/write fifo loopback example
  ------------------------------------------
signal fifo_rdreq_cmb                : std_logic;

--Some helping signals for counting nr of clocks of assertion of various signals.
--Only for help, we dont need them
signal countDv                                                      : std_logic_VECTOR(15
downto 0);
signal countRfdRq                                                       :
std_logic_VECTOR(15 downto 0);
signal countReq                                                         :
std_logic_VECTOR(15 downto 0);
------------------------------------------------------------
--Signals for FFT state machine
type states is (init, starting, dataload,compute,rdy2unload,wait2unload,unloading);
signal pr_state, nx_state : states;
------------------------------------------------------------
--Signals for rdFIFO state machine
type rdFstates is (initFrd,reading);
signal pr_Fstate, nx_Fstate : rdFstates;
------------------------------------------------------------
--Signals for ADC state machine
       type adc_states is (init_adc, ready_adc, acqstart_adc, acquire_adc, convstart_adc,
convert_adc, dataout_adc, datavalid_adc);
       signal pr_adc_state, nx_adc_state : adc_states;
------------------------------------------------------------
  --signals for adc_fsm
       signal sADC_RESET: std_logic;
       --signal sADC_CS: std_logic;
       --signal sADC_RC: std_logic;
       signal adc_start: std_logic;
       signal adc_datataken: std_logic;     -- is set by the sowtware when data is read
and stored in ddr
       signal sEOC: std_logic;              -- is set by adc_fsm when EOC is set from ADC
       --signal standby_flag: std_logic; -- flags that after last stage of current
conversion, adc must be set in standby mode:next stage will be init_adc
------------------------------------------------------------
  --signals used from timer which introduce delays between stages in adc_fsm
       signal timer_count : std_logic_vector(0 to 31);
       signal adc_delay : std_logic_vector(0 to 31);
------------------------------------------------------------
       signal countCycles                                                          :
std_logic_VECTOR(0 to 31);
       signal nsamples                                                               :
std_logic_VECTOR(0 to 31);
       --signal countOn : std_logic;

signal fsm_reset: std_logic;
signal fsm_start: std_logic;
signal fsm_unload: std_logic;
signal start_fft: std_logic;
signal unload_fft: std_logic;
--signal xn_re: std_logic_VECTOR(15 downto 0);
--signal xn_im: std_logic_VECTOR(15 downto 0);
signal fwd_inv_fft: std_logic;
signal fwd_inv_we_fft: std_logic;
signal scale_sch_fft: std_logic_VECTOR(19 downto 0);
signal scale_sch_we_fft: std_logic;
signal rfd_fft: std_logic;
signal xn_index_fft: std_logic_VECTOR(9 downto 0);
signal busy_fft: std_logic;
signal edone_fft: std_logic;
```

```
signal done_fft: std_logic;
signal dv_fft: std_logic;
signal xk_index_fft: std_logic_VECTOR(9 downto 0);
signal ovflo_fft: std_logic;                                        --we can use this signal
if it needs to check for overflow
--signal xk_re: std_logic_VECTOR(15 downto 0);
--signal xk_im: std_logic_VECTOR(15 downto 0);

component xfft_v7_1
        port (
        clk: IN std_logic;
        start: IN std_logic;
        unload: IN std_logic;
        xn_re: IN std_logic_VECTOR(15 downto 0);
        xn_im: IN std_logic_VECTOR(15 downto 0);
        fwd_inv: IN std_logic;
        fwd_inv_we: IN std_logic;
        scale_sch: IN std_logic_VECTOR(19 downto 0);
        scale_sch_we: IN std_logic;
        rfd: OUT std_logic;
        xn_index: OUT std_logic_VECTOR(9 downto 0);
        busy: OUT std_logic;
        edone: OUT std_logic;
        done: OUT std_logic;
        dv: OUT std_logic;
        xk_index: OUT std_logic_VECTOR(9 downto 0);
        xk_re: OUT std_logic_VECTOR(15 downto 0);
        xk_im: OUT std_logic_VECTOR(15 downto 0);
        ovflo: OUT std_logic);
end component;
  ---------------------------------------
  -- Signals for user logic slave model s/w accessible register example
  ---------------------------------------
  signal slv_reg0                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg1                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg2                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg3                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg4                       : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_reg_write_sel              : std_logic_vector(0 to 4);
  signal slv_reg_read_sel               : std_logic_vector(0 to 4);
  signal slv_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal slv_read_ack                   : std_logic;
  signal slv_write_ack                  : std_logic;

  ---------------------------------------
  -- Signals for user logic memory space example
  ---------------------------------------
  type BYTE_RAM_TYPE is array (0 to 255) of std_logic_vector(0 to 7);
  type DO_TYPE is array (0 to C_NUM_MEM-1) of std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal mem_data_out                   : DO_TYPE;
  signal mem_address                    : std_logic_vector(0 to 7);
  signal mem_select                     : std_logic_vector(0 to 2);
  signal mem_read_enable                : std_logic;
  signal mem_read_enable_dly1           : std_logic;
  signal mem_read_req                   : std_logic;
  signal mem_ip2bus_data                : std_logic_vector(0 to C_SLV_DWIDTH-1);
  signal mem_read_ack_dly1              : std_logic;
  signal mem_read_ack                   : std_logic;
  signal mem_write_ack                  : std_logic;

begin
  --USER logic implementation added here
  fftcore : xfft_v7_1
              port map (
                      clk => Bus2IP_Clk,
                      start => start_fft,
                      unload => unload_fft,
                      xn_re => WFIFO2IP_Data(16 to 31), --2 LSBytes
                      xn_im => WFIFO2IP_Data(0 to 15),      --2 MSBytes
                      fwd_inv => fwd_inv_fft,
                      fwd_inv_we => fwd_inv_we_fft,
                      scale_sch => scale_sch_fft,
                      scale_sch_we => scale_sch_we_fft,
                      rfd => rfd_fft,
                      xn_index => xn_index_fft,
                      busy => busy_fft,
                      edone => edone_fft,
```

```
                        done => done_fft,
                        dv => dv_fft,
                        xk_index => xk_index_fft,
                        xk_re => IP2RFIFO_Data(16 to 31),
                        xk_im => IP2RFIFO_Data(0 to 15),             -- we dont need any
state machine for writing to RFIFO
                        ovflo => ovflo_fft);

       slv_reg4(16 to 31) <= ADC_Data(15 downto 0); --slv_reg4 must have only this driver,
so I disable writing to

               -- it from SLAVE_REG_WRITE_PROC below

       slv_reg4(0 to 14) <= (others => '0');
  -------------------------------------------------------
  --Combinational logic
  -------------------------------------------------------
  --Control signals for ADC use slv_reg2, bit 28 is used for sEOC as read only
  sADC_RESET <= slv_reg2(31);
  adc_start <= slv_reg2(30);
  adc_datataken <= slv_reg2(29);
  --ADC_CS <= sADC_CS;
  --ADC_RC <= sADC_RC;

  --Contrtol signals for FFT use slv_reg1, bits 27, 28 are used for busy_fft, rfd_fft as
read only
  fwd_inv_fft<='1';
 --scale_sch_fft<="01010101010101010110";--according to fft core manual pg23

  fsm_start <= slv_reg1(31);
  fsm_unload <= slv_reg1(30);
  fsm_reset <= slv_reg1(29);
  scale_sch_fft <= slv_reg1(0 to 19);
 -------------------------------------------------------
 --State machine for controlling ADC
 -------------------------------------------------------
adc_fsm_seq:process(Bus2IP_Clk, Bus2IP_Reset, sADC_RESET, adc_start)
variable counter      : std_logic_vector(31 downto 0);
begin
if ( Bus2IP_Reset = '1' or sADC_RESET='1' ) then
       ADC_Reset <= '1';
       pr_adc_state <= init_adc;
       timer_count <= (others => '0');
       countCycles <= (others => '0');
elsif ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
       ADC_Reset <= '0';
       timer_count <= timer_count+1;
       if (timer_count = adc_delay) then
              pr_adc_state<=nx_adc_state;
              timer_count <= (others => '0');
       end if;
       if(adc_start='1') then
                        countCycles <= countCycles+1;
       end if;
end if;
end process adc_fsm_seq;

adc_fsm:process(adc_start, adc_datataken, ADC_EOC, pr_adc_state)
begin
   nx_adc_state<=pr_adc_state;
       case pr_adc_state is
              when init_adc=>
                        ADC_CS <= '1';
                        ADC_RC <= '1';
                        -- standby_flag <= '0';
                        sEOC <= '0';
                        if(adc_start='1') then
                               nx_adc_state<=ready_adc;
                               --nsamples <= (others => '0');
                        else
                               nx_adc_state<=init_adc;
                        end if;
              when ready_adc=>
                        ADC_CS <= '1';
                        ADC_RC <= '0';
                        sEOC <= '0';
                        if(adc_start='1') then
```

```
                        nx_adc_state<=acqstart_adc;
                else
                        nx_adc_state<=init_adc;
                end if;
                --nx_adc_state<=acqstart_adc;
        when acqstart_adc=>
                ADC_CS <= '0';
                ADC_RC <= '0';
                sEOC <= '0';
                nx_adc_state<=acquire_adc;
        when acquire_adc=>
                ADC_CS <= '1';
                sEOC <= '0';
                if (adc_start = '1') then
                        ADC_RC <= '0';
                        -- standby_flag <= '0';
                else
                        ADC_RC <= '1';
                        -- standby_flag <= '1';
                end if;
                nx_adc_state<=convstart_adc;
        when convstart_adc=>
                ADC_CS <= '0';
                sEOC <= '0';
                if (adc_start = '0') then ADC_RC <= '1';
                else ADC_RC <= '0';
                end if;
                -- if (standby_flag = '1') then ADC_RC <= '1';
                -- else ADC_RC <= '0';
                -- end if;
                nx_adc_state<=convert_adc;
        when convert_adc=>
                ADC_CS <= '1';
                ADC_RC <= '1';
                if(ADC_EOC='0') then
                        nx_adc_state<=dataout_adc;
                        sEOC <= '1';
                else
                        nx_adc_state<=convert_adc;
                        sEOC <= '0';
                end if;
        when dataout_adc=>
                ADC_CS <= '0';
                ADC_RC <= '1';
                sEOC <= '1';
                nx_adc_state<=datavalid_adc;
        when datavalid_adc=>
                ADC_CS <= '0';
                ADC_RC <= '1';
                sEOC <= '1';
                if(adc_datataken='1') then
                        nx_adc_state<=ready_adc;
                        sEOC <= '0';
                else
                        nx_adc_state<=datavalid_adc;
                        sEOC <= '1';
                end if;
                --countOn <= '1';
                --if(standby_flag='1') then nx_adc_state<=init_adc;
                --nx_adc_state<=ready_adc;
                -- end if;
        end case;
end process adc_fsm;
with pr_adc_state select
        adc_delay <= "00000000000000000000000000001100" when ready_adc,    --tbr and teoc:
96ns @ 125Mhz
                     "00000000000000000000000000001001" when acqstart_adc, --
tcsl:72ns @ 125Mhz
                     "00000000000000000000000000001001" when convstart_adc,--
tcsl:72ns @ 125Mhz
                     "00000000000000000000000000000110" when dataout_adc, --tdo:
48ns @ 125Mhz
                     "00000000000000000000000100011000" when acquire_adc, --tacq:
2240ns @ 125Mhz
                     "00000000000000000000000000000001" when others;

  sampCounter: process( Bus2IP_Reset, sADC_RESET, ADC_EOC ) is
```

```
  begin
    if ( Bus2IP_Reset = '1' or sADC_RESET='1' ) then
                      nsamples <= (others => '0');
        elsif(ADC_EOC'event and ADC_EOC = '0') then
                              nsamples <= nsamples +1;
        end if;
  end process sampCounter;
  ------------------------------------------------
--State machine for controlling FFT
------------------------------------------------
fft_fsm_seq:process(Bus2IP_Clk)
begin
if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
      if ( Bus2IP_Reset = '1' or fsm_reset='1' ) then
        pr_state<=init;
      else
        pr_state<=nx_state;
      end if;
end if;
end process fft_fsm_seq;

fft_fsm:process(fsm_start,fsm_unload,rfd_fft,busy_fft,dv_fft,edone_fft,done_fft,pr_state)
begin
   nx_state<=pr_state;
        case pr_state is
                when init=>
                        if(fsm_start='1') then nx_state<=starting;
                        else nx_state<=init;
                        end if;
                when starting=>
                        if(rfd_fft='1') then nx_state<=dataload;
                        else nx_state<=starting;
                        end if;
                when dataload=>
                        if(rfd_fft='0') then nx_state<=compute;
                        else nx_state<=dataload;
                        end if;
                when compute=>
                        if(edone_fft='1') then nx_state<=rdy2unload;
                        else nx_state<=compute;
                        end if;
                when rdy2unload=>
                        if(fsm_unload='1') then nx_state<=wait2unload;
                        else nx_state<=rdy2unload;
                        end if;
                when wait2unload=>                                                      ---
this state in the time diagramm is the time from the assertion of 'unload' signal
                        if(dv_fft='1') then nx_state<=unloading;  --- until 'dv' signal is
asserted. It's about 8 clock cycles
                        else nx_state<=wait2unload;
                        end if;
                when unloading=>
                        if(dv_fft='0') then nx_state<=init;
                        else nx_state<=unloading;
                        end if;
        end case;
end process fft_fsm;
start_fft<='1' when pr_state=starting else '0';
unload_fft<='1' when pr_state=wait2unload else '0';
fwd_inv_we_fft<='1' when pr_state=starting else '0';
scale_sch_we_fft<='1' when pr_state=starting else '0';
 ----------------------------------------
  -- Code to transfer data between Writefifo and FFT core
  ----------------------------------------

  IP2RFIFO_WrMark    <= '0';
  IP2RFIFO_WrRelease <= '0';
  IP2RFIFO_WrRestore <= '0';

  IP2WFIFO_RdMark    <= '0';
  IP2WFIFO_RdRelease <= '0';
  IP2WFIFO_RdRestore <= '0';

IP2WFIFO_RdReq <= fifo_rdreq_cmb;
IP2RFIFO_WrReq <= dv_fft;

-------FIFO state machine for (reading) transfer data from WrFifo to FFT.
```

```
--It uses as inputs signals fsm_start, fsm_reset and rfd_fft which also used to FFT state
machine
fifo_fsm_seq:process(Bus2IP_Clk)
begin
if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
      if ( Bus2IP_Reset = '1' or fsm_reset='1' ) then
        pr_Fstate<=initFrd;
      else
        pr_Fstate<=nx_Fstate;
      end if;
end if;
end process fifo_fsm_seq;

fifo_fsm:process(fsm_start,rfd_fft,WFIFO2IP_Empty,pr_Fstate)
begin
   nx_Fstate<=pr_Fstate;
        case pr_Fstate is
                when initFrd=>
                        if(fsm_start='1') then nx_Fstate<=reading;
                        else nx_Fstate<=initFrd;
                        end if;
                when reading=>
                        if(WFIFO2IP_Empty = '1' and rfd_fft='0') then nx_Fstate<=initFrd;
                        else nx_Fstate<=reading;
                        end if;
       end case;
end process fifo_fsm;

fifo_rdreq_cmb <= '1' when pr_Fstate = reading else '0';

--Processes for the counters. We need them only for helping purposes
 rfdrdreqCounter: process( Bus2IP_Clk ) is
  begin
    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
                if ( Bus2IP_Reset = '1' or pr_state = unloading) then
                        countRfdRq <= "0000000000000000";
                else
                        if ((fifo_rdreq_cmb and rfd_fft) = '1') then
                                countRfdRq <= countRfdRq +1;
                        end if;
                end if;
    end if;
  end process rfdrdreqCounter;

reqCounter: process( Bus2IP_Clk ) is
  begin
    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
                if ( Bus2IP_Reset = '1' or pr_state = unloading) then
                        countReq <= "0000000000000000";
                else
                        if ((fifo_rdreq_cmb or rfd_fft) = '1') then
                                countReq <= countReq +1;
                        end if;
                end if;
    end if;
  end process reqCounter;

dvCounter: process( Bus2IP_Clk ) is
  begin
    if ( Bus2IP_Clk'event and Bus2IP_Clk = '1' ) then
                if ( Bus2IP_Reset = '1' or pr_state = starting) then
                        countDv <= "0000000000000000";
                else
                        if (dv_fft = '1') then
                                countDv <= countDv +1;
                        end if;
                end if;
    end if;
  end process dvCounter;
  ------------------------------------------
  -- Code to read/write user logic slave model s/w accessible registers
  --
  -- Note:
  -- Each bit of the Bus2IP_WrCE/Bus2IP_RdCE signals is configured to correspond
  -- to one software accessible register by the top level template. For example,
  -- if you have four 32 bit software accessible registers in the user logic,
  -- you are basically operating on the following memory mapped registers:
```

```
  --
  --    Bus2IP_WrCE/Bus2IP_RdCE   Memory Mapped Register
  --                        "1000"   C_BASEADDR + 0x0
  --                        "0100"   C_BASEADDR + 0x4
  --                        "0010"   C_BASEADDR + 0x8
  --                        "0001"   C_BASEADDR + 0xC
  --
  ---------------------------------------
  slv_reg_write_sel <= Bus2IP_WrCE(0 to 4);
  slv_reg_read_sel  <= Bus2IP_RdCE(0 to 4);
  slv_write_ack     <= Bus2IP_WrCE(0) or Bus2IP_WrCE(1) or Bus2IP_WrCE(2) or Bus2IP_WrCE(3)
or Bus2IP_WrCE(4);
  slv_read_ack      <= Bus2IP_RdCE(0) or Bus2IP_RdCE(1) or Bus2IP_RdCE(2) or Bus2IP_RdCE(3)
or Bus2IP_RdCE(4);

  -- implement slave model software accessible register(s)
  SLAVE_REG_WRITE_PROC : process( Bus2IP_Clk ) is
  begin

    if Bus2IP_Clk'event and Bus2IP_Clk = '1' then
      if Bus2IP_Reset = '1' then
        slv_reg0 <= (others => '0');
        slv_reg1(0 to 19) <= "01010101010101010101"; --take initial value  for fft schedule
               slv_reg1(20 to 31) <= (others => '0');
        slv_reg2 <= (others => '0');
        slv_reg3 <= (others => '0');
        --slv_reg4 <= (others => '0');
      else
        case slv_reg_write_sel is
          when "10000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg0(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          when "01000" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg1(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          when "00100" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg2(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          when "00010" =>
            for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              if ( Bus2IP_BE(byte_index) = '1' ) then
                slv_reg3(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              end if;
            end loop;
          -- when "00001" =>
            -- for byte_index in 0 to (C_SLV_DWIDTH/8)-1 loop
              -- if ( Bus2IP_BE(byte_index) = '1' ) then
                -- slv_reg4(byte_index*8 to byte_index*8+7) <= Bus2IP_Data(byte_index*8 to
byte_index*8+7);
              -- end if;
            -- end loop;
          when others => null;
        end case;
      end if;
    end if;

  end process SLAVE_REG_WRITE_PROC;

  -- implement slave model software accessible register(s) read mux
  SLAVE_REG_READ_PROC : process( slv_reg_read_sel, nsamples, slv_reg1, slv_reg2,
countCycles, slv_reg4 ) is
  begin

    case slv_reg_read_sel is
```

```
      when "10000" => slv_ip2bus_data <= nsamples;
      when "01000" =>
                        slv_ip2bus_data(0 to 26) <= slv_reg1(0 to 26);
                        slv_ip2bus_data(27) <= rfd_fft;
                        slv_ip2bus_data(28) <= busy_fft;      -- signals busy_fft, rfd_fft
are read only
                        slv_ip2bus_data(29 to 31) <= slv_reg1(29 to 31);
      when "00100" =>
                        slv_ip2bus_data(0 to 27) <= slv_reg2(0 to 27);
                        slv_ip2bus_data(28) <= sEOC;
                        slv_ip2bus_data(29 to 31) <= slv_reg2(29 to 31);
      when "00010" => slv_ip2bus_data <= countCycles;
      when "00001" => slv_ip2bus_data <= slv_reg4;
      when others => slv_ip2bus_data <= (others => '0');
    end case;

  end process SLAVE_REG_READ_PROC;

  ----------------------------------------
  -- Code to drive IP to Bus signals
  ----------------------------------------
  IP2Bus_Data  <= slv_ip2bus_data when slv_read_ack = '1' else
                  mem_ip2bus_data when mem_read_ack = '1' else
                  (others => '0');

  IP2Bus_WrAck <= slv_write_ack or mem_write_ack;
  IP2Bus_RdAck <= slv_read_ack or mem_read_ack;
  IP2Bus_Error <= '0';

end IMP;
```

ΠΑΡΑΡΤΗΜΑ Γ: Κώδικας C του software ελέγχου (pal.c)

```
/*
 * pal.c
 *
 *  Created on: 09 Ξ™ΞΩΟ…Ξ» 2010
 *      Author: tdak
 */

#include "xparameters.h"
#include "xil_types.h"               //for s32, s16 etc. types
#include "xil_exception.h"
#include "xbasic_types.h"            //for Xuint32 etc. types
#include "xstatus.h"
#include "mainfft.h"
#include "xgpio.h"
#include "xio.h"
#include "stdio.h"
#include "math.h"                            //for pow(), sqrt() functions
#include "xutil.h"
#include "stdlib.h"
#include "xuartlite_l.h"
#include "xuartlite.h"
#include "string.h" //for memset function
#include "xmpmc.h"  // for mpmc memory controller
#include "xil_testmem.h" //for memory tests
#include "sleep.h"  //for sleep() function
//for FATfs
#include <sysace_stdio.h>
#include "xsysace_l.h"
/************************** Constant Definitions *****************************/
#define ADC_RESET   0x00000001               //slvreg2(31) = 1
#define ADC_START   0x00000002          //slvreg2(30) = 1, also sets DATA_TAKE_READY
#define ADC_STOP 0x00000000                  //slvreg2(30) = 0
#define DATA_TAKEN 0x00000006        //slvreg2(29) = 1, ADC_START bit = 1

#define NFFT 1024                                    //number of FFT length
#define resetVal 0x00000004                  //3rd bit is 1
#define unloadVal 0x00000002         //2nd bit is 1
#define startVal 0x00000001                  //1st bit is 1
#define scheduleVal 0xfffff000               //20 last bit are 1
#define zeroControl 0xfffffff8               //set 1st,2nd and 3rd bits to zero
#define zeroSchedule 0x00000fff              //set schedule bits to zero
//for FATfs
#define CONFIG_WRITE
#define CONFIG_DIR_SUPPORT
/*
 * Memory controller's mpmc
*/
#define MPMC_DEVICE_ID                 XPAR_MPMC_0_DEVICE_ID

/************************** Function Prototypes ******************************/
XStatus memContrInit();
XStatus memTest();
void read_file(char FileName[], int verbose); //reads a file from CF to data buffer
void write_file(char FileName[], int verbose); //writes data from DDR2 to CF as .dat file
through data buffer
void writeDDR2CF(char FileName[], Xuint32 slices);//writes ALL data from DDR2 to CF as .dat
file
void writePSD2CF(char FileName[]);                           //writes averaged PSD data
from buffer to CF as .dat file
void sysace_reset();
void buf2fifo(Xuint32 *buffer);                                      //moves data from data
buffer to fft's FIFO
void fifo2buf(Xuint32 *buffer);                                      //moves data from fft's
FIFO to data buffer
void ddr2fifo(Xuint32 offset);                      //moves data from ddr to fft's FIFO
void fifo2ddrpsd(Xuint32 offset);                           //moves data from fft's FIFO to
ddr as PSD
void fifo2ddr(Xuint32 offset);                                       //moves data from fft's
FIFO to ddr as FFT components
void floatshow(Xuint32 offset);                                      //Display contents of
DDR as complex floating point
void intshow(Xuint32 offset);                                //Display contents of DDR as
integer complex
```

```
void floatpsdshow(Xuint32 offset);                          //Display contents of DDR (PSD)
as floating point
void intpsdshow(Xuint32 offset);                            //Display contents of DDR (PSD)
as integer
void dcRemove(Xuint32 offset);                                      //calculates and removes
DC for all DDR samples, slice by slice, starting from offset
//void mem2init(Xuint32 offset);                                    //Initialize to zero the
first 1024 32bit words of user memory2
void avePSD();                                                              //calculate
average PSD
void ADC_ReadData(Xuint32 sampnum);                         //
void filesPrep();
void read_file2term(char FileName[]); //reads a file from CF and show to terminal, fmode=1
for reading psd data files, fmode=0 for others
/************************** Variable Definitions *****************************/

static XMpmc MemController;  // Memory controller instance
XMpmc_Config *CfgPtr;
Xuint32 *mainfft_p =  (Xuint32 *)XPAR_MAINFFT_0_BASEADDR;          // mainfft core base
address pointer
Xuint32 *mpmc_p = (Xuint32 *) XPAR_MPMC_0_MPMC_BASEADDR;    //memory controller base
address pointer
//Xuint32 *usermem0_p = (Xuint32 *) XPAR_MAINFFT_0_MEM0_BASEADDR;
//Xuint32 *usermem2_p = (Xuint32 *) XPAR_MAINFFT_0_MEM2_BASEADDR;   //user memory 2 base
address pointer
Xuint32 *sysace_p = (Xuint32 *) XPAR_SYSACE_0_BASEADDR;     //sysace controller base
address pointer
Xuint32 sysace;
Xuint32 mainfft;
volatile Xuint32 ddr_offset = 0x0;
XStatus status;
Xuint32 dataBuf[1024]; //for intermediate storage
Xint16 windowCoef[1024]; //for windows coefs, which are in Q0.15 format
u64 aveBuf[512];
Xuint32 i=0,k=0;
Xint16 tempReal, tempImag;
Xint32 tempPsd;
Xint32 dcVal;
Xuint32 nsamples = 13312;            //total number of samples of the time serie. Default
13312 (about 1msec
//float captime = 10.24;             //Multiples of 10.24 msec or 0.01024 sec, which is the
time for 1024 samples
Xuint32 nslices;                            // Integer for the number of 1024 slices
(=nsamples/1024).
Xuint32 sampOnMem;                          //keeps track of nr of samples that written in
ddr
//Files prefix
char rawFile[20];
char fftFile[20];
char psdFile[20];
char psdaveFile[20];
char fxstr[4];
Xuint8 fx = 0;

union mys16orFloat {
  s32 s;
  float f;
  } realFFT, imagFFT;

int main() {
  char stringbuf[20]={0};
  char * pEnd;
  Xuint32 ddrnum;
  Xuint32 wrFifoFull, rdFifoEmpty;
  Xuint32 wrFifoVac, rdFifoOccup;
  Xuint32 temp;
  char inputch, keyboard;
  Xuint32 statusReg;
  //Xuint32 rfdRqCnt, reqCnt, dvCnt;
  Xuint8 rfd, busy, dv, unload, start ;
  Xuint16 sampTime=1;
  Xuint32 scsch;              //var for scaling schedule of FFT
  mainfft = (Xuint32) mainfft_p;

  // Clear the screen
    xil_printf("%c[2J",27);
```

```
  // Check that the peripheral exists
    XASSERT_NONVOID(mainfft_p != XNULL);
    xil_printf("Read/Write to FIFO Test\n\r");
    xil_printf("  - Initializing the memory controller\r\n");

    memContrInit();
    memTest();

  while(1) {
         nslices = nsamples/1024UL;
         xil_printf ("Number of slices = %d \r\n",nslices);
         wrFifoFull = MAINFFT_mWriteFIFOFull(mainfft);
         wrFifoVac = MAINFFT_mWriteFIFOVacancy(mainfft);
         rdFifoEmpty = MAINFFT_mReadFIFOEmpty(mainfft);
         rdFifoOccup = MAINFFT_mReadFIFOOccupancy(mainfft);

         xil_printf ("WrFifo Full          = %d \r\n",wrFifoFull );
         xil_printf ("WrFifo Vacancy       = %d \r\n", wrFifoVac);
         xil_printf ("RdFifo Empty         = %d \r\n", rdFifoEmpty);
         xil_printf ("RdFifo Occupancy     = %d \r\n", rdFifoOccup);

/*  Print initialization message, and prompt for user input. */
  xil_printf ("*****************************************************************\r\n");
  xil_printf ("* -->       Hyperterm window must be selected <--                *\r\n");
  xil_printf ("*                                                               *\r\n");
  xil_printf ("* Enter value and Carriage Return                               *\r\n");
  xil_printf ("* 0: Reset FFT and FIFOs and ADC                                *\r\n");
  xil_printf ("* 1: Get status                                                 *\r\n");
  xil_printf ("* 2: Load new SCALE schedule (change the default).              *\r\n");
  xil_printf ("* 3: Read time data (or .txt data file) from terminal to DDR2.  *\r\n");
  xil_printf ("* 4: Start a complete cycle FFT, outputs FFT coeffs.            *\r\n");
  xil_printf ("* 5: Start a complete cycle FFT, outputs PSD.                   *\r\n");
  xil_printf ("* 6: Display DDR data as float complex nrs                      *\r\n");
  xil_printf ("* 7: Display DDR data as integer complex nrs                    *\r\n");
  xil_printf ("* 8: Display DDR PSD data as float nrs                          *\r\n");
  xil_printf ("* 9: Display DDR PSD data as integer  nrs                       *\r\n");
  xil_printf ("* r: read and load .dat file from CF to data buffer             *\r\n");
  xil_printf ("* w: write data from DDR2 to CF as.dat file                     *\r\n");
  xil_printf ("* b: write data from buffer to CF as .dat file.                 *\r\n");
  xil_printf ("* s: Give time duration of time serie (msecs).                  *\r\n");
  xil_printf ("* d: Extract DC from all samples in DDR.                        *\r\n");
  xil_printf ("* a: Calculate average PSD from all slices.                     *\r\n");
  xil_printf ("* p: Write average PSD to CF.                                   *\r\n");
  xil_printf ("* h: Load window coefs from file to buffer.                     *\r\n");
  xil_printf ("* c: Start capture, FFT and save files (Raw data and FFT).      *\r\n");
  xil_printf ("* g: Start capture FFT and save files (Raw data and PSD).       *\r\n");
  xil_printf ("* f: Read a data file from CF to terminal.                      *\r\n");
  xil_printf ("* esc: exit program.                                            *\r\n");
  xil_printf ("*****************************************************************\r\r\n");
/* Read the keyboard, print the entry.  */
  keyboard = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
  xil_printf ("%c \r\r\n", keyboard);


 /* Perform requested action. */
  switch ( keyboard ) {

       case '0': {    /* FFT operation*/
              temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
              temp = temp & zeroControl;                                  //set to 0 first
3 bits
              temp = temp ^ resetVal;
              MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
              xil_printf ("Resetting FFT.....DONE\r\n");
              // Reset read and write packet FIFOs to initial state
              MAINFFT_mResetWriteFIFO(mainfft);
              MAINFFT_mResetReadFIFO(mainfft);
              xil_printf ("Resetting FIFOs.....DONE\r\n");
              MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_RESET);
              xil_printf ("Resetting ADC.....DONE\r\n");
       }
    break;
       case '1': {    /* Print the status as read from the circuit */
                        statusReg = MAINFFT_mReadSlaveReg1(mainfft, 0);
                        start = (statusReg & 0x00000001);
                        unload = (statusReg & 0x00000002) >> 1;
                        dv = (statusReg & 0x00000004) >> 2;
```

```
                        busy = (statusReg & 0x00000008) >> 3;
                        rfd = (statusReg & 0x00000010) >> 4;
                        scsch = (statusReg & 0xfffff000) >> 12;

                        xil_printf ("Ready for data  = %d \r\n", rfd);
                        xil_printf ("Busy Flag       = %d \r\n", busy);
                        xil_printf ("Data Valid Flag = %d \r\n", dv);
                        xil_printf ("Unload Flag     = %d \r\n", unload);
                        xil_printf ("Start Flag      = %d \r\r\n", start);
                        xil_printf ("Scale Schedule  = 0x%5x \r\n", scsch);

                        xil_printf ("Status  = 0x%8x \r\n", statusReg);
                        xil_printf ("address  = 0x%8x \r\n", mainfft_p);

                        }
                break;

   case '2': {    /* Change Schedule */
                xil_printf ("Give a new Scale Schedule\r\r\n");

                for(i=0;;i++){
                                inputch = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                                xil_printf ("%c", inputch);
                                        if ((inputch == '\n') || (inputch=='\r')) break;
                                        else{
                                        stringbuf[i]=inputch;
                                        }
                }
                temp = atol(stringbuf);
                xil_printf("You gave: 0x%5x\r\n",temp);
                memset(stringbuf, '\0', 20);
                temp = temp << 12;

                temp = (MAINFFT_mReadSlaveReg1(mainfft, 0) & zeroSchedule) ^  temp ;
                MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);

                 xil_printf ("Scaling changed succesfully!\r\r\n");
                 }
   break;

   case '3': {
                        /* Load sampled data from terminal to DDR2. Data file is previously
converted in matlab
                         * as 16bit integer. Each row of the file has 1 16bit number
                         * (file A013data.txt in PALJef directory has full converted data)
                         * Other files data is in Q0.15 format
                         */
                          ddr_offset = 0;
                          xil_printf ("Specify File to read data...Waiting\r\r\n");
                          for(k=0;k < nsamples;k++){
                                for(i=0;;i++){
                                        inputch =
XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                                        if ((inputch == '\n') || (inputch=='\r')) break;
                                        else stringbuf[i]=inputch;
                                }
                                ddrnum = strtol(stringbuf,&pEnd,10);
                                //ddrnum = (ddrnum << 8) ^ (strtol(pEnd,&pEnd,10));
                                //xil_printf ("Number:%d\r\n",ddrnum);
                                mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+ddr_offset);
                                *mpmc_p = ddrnum;
                                ddrnum = 0;
       //clear temporary number var
                                memset(stringbuf, '\0', 20);                            //clear
string buffer
                                ddr_offset = ddr_offset+4;
                          }
                          xil_printf ("Data Loading DONE!!!\r\r\n");
                  }
    break;
       case '4': {    /* Start FFT after loading DATA from ddr to writeFIFO, output FFT
coefficients*/
                                // MAINFFT_mResetWriteFIFO(mainfft);
                                // MAINFFT_mResetReadFIFO(mainfft);
                                // ddr_offset = 0;
                                //mem2init(0);                                  //init to zero
mem2
```

```
                                // for(k=0;k<nslices;k++){
                                        // ddr2fifo(ddr_offset);                        //loads
data to fifo without dc
                                        // temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                                        // temp = temp & zeroControl;
        //set to 0 first 3 bits
                                        // temp = temp ^ startVal;
                                        // MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                                        // while((MAINFFT_mWriteFIFOVacancy(mainfft)) !=
NFFT){}

                                        // temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                                        // temp = temp & zeroControl;
        //set to 0 first 3 bits
                                        // temp = temp ^ unloadVal;
                                        // MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                                        // while((MAINFFT_mReadFIFOOccupancy(mainfft)) !=
NFFT){}
                                        // fifo2ddr(ddr_offset);

                                        // ddr_offset = ddr_offset+4096;
                                        //xil_printf ("FFT of slice %d.....DONE\r\n",k);
                                // //}
                        }
        break;

        case '5': {    /* Start FFT after loading DATA from ddr to writeFIFO, outputs PSD
spectrum*/
                                        // MAINFFT_mResetWriteFIFO(mainfft);
                                        // MAINFFT_mResetReadFIFO(mainfft);
                                        // ddr_offset = 0;
                                        // for(k=0;k<nslices;k++){
                                                // ddr2fifo(ddr_offset);
        //loads data to fifo without dc
                                                // temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                                                // temp = temp & zeroControl;
                //set to 0 first 3 bits
                                                // temp = temp ^ startVal;
                                                // MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                                                // while((MAINFFT_mWriteFIFOVacancy(mainfft))
!= NFFT){}

                                                // temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                                                // temp = temp & zeroControl;
                //set to 0 first 3 bits
                                                // temp = temp ^ unloadVal;
                                                // MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                                                // while((MAINFFT_mReadFIFOOccupancy(mainfft))
!= NFFT){}
                                                // fifo2ddrpsd(ddr_offset);

                                                // ddr_offset = ddr_offset+4096;
                                                //xil_printf ("FFT of slice
%d.....DONE\r\n",k);
                                        }
   break;

   case '6': {
                ddr_offset = 0;
                floatshow(ddr_offset);
                }
    break;
   case '7': {   /* Dump DDR2 memory to terminal */
                ddr_offset = 0;
                intshow(ddr_offset);
        }
   break;
   case '8': {
                ddr_offset = 0;
                floatpsdshow(ddr_offset);
        }
       break;
   case '9': {   /* Dump DDR2 memory to terminal */
                ddr_offset = 0;
                intpsdshow(ddr_offset);
        }
    break;
```

```
        case '\e':   /* Exit program */
          xil_printf ("That was exciting, wasn't it? \r\r\n");
          exit(0);
          break;

        case 'r': {    /* Read data file */
                xil_printf("\nReading Data file.\n\r");

                // Reset the sysace controller to clean any bad state, leave it in MPU mode
                sysace_reset();
                read_file("a:\\test\\data0505.dat", 1);
                xil_printf("\n\rread done\n\r");

                                }
        break;
        case 'w': {    /* Write data file */
                #ifdef CONFIG_WRITE
                        writeDDR2CF("a:\\test\\ddr2all.dat", nslices);
                        xil_printf("write done\n\r");
                #endif
        }
        break;
        case 'b': {    /* Write data file */
                        xil_printf("\nWriting data from buffer to file.\n\r");
                        // Reset the sysace controller to clean any bad state, leave it in
MPU mode
                        sysace_reset();
                        #ifdef CONFIG_WRITE
                                write_file("a:\\test\\FFT0505.dat", 1);
                                xil_printf("write done\n\r");
                        #endif
                }
        break;
    case 's': {    /* Give number of total samples */
                xil_printf ("Give duration of each time serie(in msecs):\r\r\n");
                for(i=0;;i++){
                                        inputch =
XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                                        xil_printf ("%c", inputch);
                        if ((inputch == '\n') || (inputch=='\r')) break;
                        else{
                                stringbuf[i]=inputch;
                                }
                }
                                sampTime = atol(stringbuf);
                                if (sampTime > 5000 ){
                                        xil_printf("Please give a valid value, max 5000
mseconds \r\n" );
                                break;
                                }
                                nsamples = sampTime * 13312;
                                xil_printf("You set d% mseconds@d% samples \r\n" ,sampTime,
nsamples);
                memset(stringbuf, '\0', 20);
        }
    break;
    case 'd': {    /* Extract dc component from all samples, slice by slice */
        dcRemove(0);
        }
    break;
    case 'a': {    /* Calculate average PSD from all slices */
                avePSD();
                }
         break;
    case 'p': {    /* Write data file */
                /* #ifdef CONFIG_WRITE
                        writePSD2CF("a:\\test\\psdave.dat");
                        xil_printf("write done\n\r");
                #endif */
        }
    break;
    case 'h': {    /* Read winow coefs file */
        xil_printf ("Specify File to read data...Waiting\r\r\n");
                for(k=0;k < 1024;k++){
                        for(i=0;;i++){
                                inputch = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                                if ((inputch == '\n') || (inputch=='\r')) break;
```

```
                        else stringbuf[i]=inputch;
                    }
                    windowCoef[k] = (Xint16)(strtol(stringbuf,&pEnd,10));
                    memset(stringbuf, '\0', 20);                        //clear string
buffer
                }
                for(i=0;i<1024;i++){
                    xil_printf("window buffer %d:%d\n\r",i,windowCoef[i]);
                }
    }
    break;
        case 'c': {    /*Capture data and save to file (raw data and fft components)*/
        xil_printf ("Conversion will start for: %d samples\r\n",nsamples);
            //sleep(5);
                xil_printf ("Now is starting \r\n");
                MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_START);
                ADC_ReadData(nsamples);
                MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_STOP);
                xil_printf ("Conversion is done and data are stored in ddr2 memory.\n\r");
                xil_printf("Nr of cycles of total conversions (@125Mhz): %d\n\r",
MAINFFT_mReadSlaveReg3(mainfft, 0));
                xil_printf("Last memory offset 0x%08x\n\r", ddr_offset);
                xil_printf("nr of samples in mem : %d\n\r", sampOnMem);
                xil_printf("nr of samples that converted in ADC: %d\n\r",
MAINFFT_mReadSlaveReg0(mainfft, 0));

                /* FFT procedure:Now DDR has all samples, we can proceed to FFT. Start FFT
after loading DATA from ddr to writeFIFO,
                output FFT coefficients. For each slice of 1024 points, sample data will be
loaded in FIFO, do FFT,
                next back to DDR as FFT components*/
                /*First we save raw data to file. Each sample of 16bit data width is stored
in a 32bit space in DDR, as
                later will need the upper:*/
                #ifdef CONFIG_WRITE
                    writeDDR2CF("a:\\test\\ddrRaw.dat", nslices);
                    xil_printf("Raw data writen to file\n\r");
                #endif
                //Remove dc component drom all
                dcRemove(ddr_offset);
                //Proceed to FFT
                MAINFFT_mResetWriteFIFO(mainfft);
                MAINFFT_mResetReadFIFO(mainfft);
                ddr_offset = 0;
                //mem2init(0);                              //init to zero mem2
                for(k=0;k<nslices;k++){
                    ddr2fifo(ddr_offset);                 //loads data to fifo without dc
                    temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                    temp = temp & zeroControl;                                  //set to
0 first 3 bits
                    temp = temp ^ startVal;
                    MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                    while((MAINFFT_mWriteFIFOVacancy(mainfft)) != NFFT){}

                    temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                    temp = temp & zeroControl;                                  //set to
0 first 3 bits
                    temp = temp ^ unloadVal;
                    MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                    while((MAINFFT_mReadFIFOOccupancy(mainfft)) != NFFT){}
                    fifo2ddr(ddr_offset);
                    ddr_offset = ddr_offset+4096; // increments offset to FFT next slice
                    //xil_printf ("FFT of slice %d.....DONE\r\n",k);
                }
                /* Write FFT components to file: each 32bit word contains signed complex FFT
component,
                real and imaginary parts in Q0.15 format*/
                #ifdef CONFIG_WRITE
                    writeDDR2CF("a:\\test\\ddrFFT.dat", nslices);
                    xil_printf("FFT data writen to file\n\r");
                #endif
        }
        case 'g': {    /* Capture data and save to file (raw data and psd) */
        xil_printf ("Conversion will start for: %d samples\r\n",nsamples);
            //sleep(5);
                xil_printf ("Now is starting \r\n");
                MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_START);
```

```
                ADC_ReadData(nsamples);
                MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_STOP);
                xil_printf ("Conversion is done and data are stored in ddr2 memory.\n\r");
                xil_printf("Nr of cycles of total conversions (@125Mhz): %d\n\r",
MAINFFT_mReadSlaveReg3(mainfft, 0));
                xil_printf("Last memory offset 0x%08x\n\r", ddr_offset);
                xil_printf("nr of samples in mem : %d\n\r", sampOnMem);
                xil_printf("nr of samples that converted in ADC: %d\n\r",
MAINFFT_mReadSlaveReg0(mainfft, 0));
                /*Prepare files for the data*/
                filesPrep();
                /* FFT procedure:Now DDR has all samples, we can proceed to FFT. Start FFT
after loading DATA from ddr to writeFIFO,
                output FFT coefficients. For each slice of 1024 points, sample data will be
loaded in FIFO, do FFT,
                next back to DDR as FFT components*/
                /*First we save raw data to file. Each sample of 16bit data width is stored
in a 32bit space in DDR, as
                later will need the upper:*/
                #ifdef CONFIG_WRITE
                        writeDDR2CF(rawFile, nslices);
                        xil_printf("Raw data writen to file\n\r");
                #endif
                //Remove dc component drom all, starting from offset 0
                dcRemove(0);
                //Proceed to FFT
                MAINFFT_mResetWriteFIFO(mainfft);
                MAINFFT_mResetReadFIFO(mainfft);
                ddr_offset = 0;
                //mem2init(0);                                  //init to zero mem2
                for(k=0;k<nslices;k++){
                        ddr2fifo(ddr_offset);                   //loads data to fifo without dc
                        temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                        temp = temp & zeroControl;                                      //set to
0 first 3 bits
                        temp = temp ^ startVal;
                        MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                        while((MAINFFT_mWriteFIFOVacancy(mainfft)) != NFFT){}

                        temp = MAINFFT_mReadSlaveReg1(mainfft, 0);
                        temp = temp & zeroControl;                                      //set to
0 first 3 bits
                        temp = temp ^ unloadVal;
                        MAINFFT_mWriteSlaveReg1(mainfft, 0, temp);
                        while((MAINFFT_mReadFIFOOccupancy(mainfft)) != NFFT){}
                        fifo2ddrpsd(ddr_offset);
                        ddr_offset = ddr_offset+4096; // increments offset to FFT next slice
                        //xil_printf ("FFT of slice %d.....DONE\r\n",k);
                }
                /* Write PSD specrum components to file: for each slice of 1024 fft
components, we get 512 psd
                components in real Q0.30 format. PSD data are written in DDR continuously,
so now each slice has
                512 words of 32bit psd data*/
                //To write PSD data, as we have now psd spectrum, we need to access half of
the ddr memory space, means nslices/2 times
                #ifdef CONFIG_WRITE
                        writeDDR2CF(psdFile, (nslices/2UL));
                        xil_printf("PSD data writen to file\n\r");
                #endif
                //Calculate average psd from all psd slices and store it to file
                avePSD();
                //Now intermidiate buffer has the average psd spectrum, write buffer
contents to file
                #ifdef CONFIG_WRITE
                        writePSD2CF(psdaveFile);
                        xil_printf("write done\n\r");
        #endif
        fx = fx+1;  //increment file prefix for next files
        }
    break;
        case 'f': {    /* Read a file from CF to terminal */
                memset(stringbuf, '\0', 20);
         xil_printf ("Last files saved in CF:\r\n");
                xil_printf ("file1:%s\r\n", rawFile);
                xil_printf ("file2:%s\r\n", fftFile);
                xil_printf ("file3:%s\r\n", psdFile);
```

```
                xil_printf ("file4:%s\r\n", psdaveFile);
                xil_printf ("ext:%s\r\n", fxstr);

                xil_printf ("Give a new File name (full path as above) to read\r\r\n");
            for(i=0;;i++){
                inputch = XUartLite_RecvByte(XPAR_RS232_UART_1_BASEADDR);
                xil_printf ("%c", inputch);
                if ((inputch == '\n') || (inputch=='\r')) break;
                else{
                        stringbuf[i]=inputch;
                }
            }
            xil_printf ("ext:%s\r\n", stringbuf);
            read_file2term(stringbuf);
            memset(stringbuf, '\0', 20);
    }
        break;
        default: {
          mainfft_p = (Xuint32 *)XPAR_MAINFFT_0_BASEADDR;
          *mainfft_p = 0x00000000;
        }
          break;
    }
  }
}
XStatus memContrInit(){
/*
 * Initialize the MemController device.
*/
        CfgPtr = XMpmc_LookupConfig(MPMC_DEVICE_ID);
    if (CfgPtr == NULL) {
        return XST_FAILURE;
    }

    status = XMpmc_CfgInitialize(&MemController, CfgPtr, CfgPtr->BaseAddress);
    if (status != XST_SUCCESS) {
        return XST_FAILURE;
    }
    return XST_SUCCESS;
}

XStatus memTest(){
        /* Testing MPMC Memory (DDR2_SDRAM)*/
    print("Starting MemoryTest for DDR2_SDRAM:\r\n");
    print("  Running 32-bit test...");
    status = Xil_TestMem32((Xuint32*)XPAR_DDR2_SDRAM_MPMC_BASEADDR, 1024, 0xAAAA5555,
XIL_TESTMEM_ALLMEMTESTS);
    if (status == XST_SUCCESS) {
        print("PASSED!\r\n");
    }
    else {
        print("FAILED!\r\n");
    }
    print("  Running 16-bit test...");
    status = Xil_TestMem16((u16*)XPAR_DDR2_SDRAM_MPMC_BASEADDR, 2048, 0xAA55,
XIL_TESTMEM_ALLMEMTESTS);
    if (status == XST_SUCCESS) {
        print("PASSED!\r\n");
    }
    else {
        print("FAILED!\r\n");
    }
    print("  Running 8-bit test...");
    status = Xil_TestMem8((u8*)XPAR_DDR2_SDRAM_MPMC_BASEADDR, 4096, 0xA5,
XIL_TESTMEM_ALLMEMTESTS);
    if (status == XST_SUCCESS) {
        print("PASSED!\r\n");
    }
    else {
        print("FAILED!\r\n");
    }
    return status;
}
void read_file(char FileName[], int verbose)
//Reads a file from CF card byte by byte and put data to an intermediate buffer (dataBuf)
{
        SYSACE_FILE *ptest;
```

```
        int i= 0;
        int numread = 0;
        int total_bytes_read = 0;
        ptest = sysace_fopen(FileName , "r" );

        if(ptest) {
                xil_printf("Reading file : %s\n\r", FileName);
                do {
                        numread = sysace_fread(dataBuf, 1, sizeof(dataBuf), ptest); //read
data from file and store them to readbuffer
                     total_bytes_read = total_bytes_read + numread;
        //bytes are read byte by byte
                } while(numread);
                if (verbose) {
                   i = 0;
                   while( i < (numread/4)) {
                   xil_printf("%d\n\r", dataBuf[i]);
                   i++;
                   }
                }
                sysace_fclose (ptest);
        } else {
                xil_printf("Failed to open %s: check if file present\n\r", FileName);
        }
        xil_printf("Total bytes read = %d\n\r", total_bytes_read);
}
void read_file2term(char FileName[])
//Reads a file from CF card byte by byte, put data to an intermediate buffer (dataBuf) and
show to screen
//The first 4 bytes (32bits) of the file, must show the nr of bytes the file contains
{
        SYSACE_FILE *ptest;
        int k,i= 0;
        int numread = 0;
        int total_bytes_read = 0;
               Xuint32 page = 2048, slc, tempbuf[1];        //psdave files have data
multiples of 512*32bits word = 2048 bytes, so we set page = 2048
        ptest = sysace_fopen(FileName , "r" );

        if(ptest) {
                xil_printf("Reading file : %s\n\r", FileName);
                               numread = sysace_fread(tempbuf, 1, 4, ptest); //read the
first 32bit nr of the data file
                               total_bytes_read = total_bytes_read + numread;
                               slc = (tempbuf[0]/page);
              //calculate the nr of slices (multiples of page) the file contains
                               xil_printf("File slices%d\n\rtempbuf:%d\n\r", slc,
tempbuf[0]);
                                       for(k=0;k<slc;k++){
        // do loop reading
                                               do {
                                                       numread = sysace_fread(dataBuf, 1,
page, ptest); //read data from file and store them to buffer
                                                       total_bytes_read = total_bytes_read +
numread;
                                               }while(numread); //while data are read from
file
                                               for(i=0;i < 512; i++) {
        //read 512 words (of 32bit)
                                                       xil_printf("%d\n\r", dataBuf[i]);
                                               }
                                       }
                sysace_fclose (ptest);
        } else {
                xil_printf("Failed to open %s: check if file present\n\r", FileName);
        }
        xil_printf("Total bytes read = %d\n\r", total_bytes_read);
}
#ifdef CONFIG_WRITE
void write_file(char FileName[], int verbose)
//Writes the contents of intermediate buffer (dataBuf) to .dat file in CF card
//Data is written to file byte by byte in BIG ENDIAN format (MSByte in lower position)
{
        SYSACE_FILE *ptest;
        int n = 0;
        ptest = sysace_fopen(FileName ,"w" );
```

```
        if(ptest) {
                xil_printf("Writing file contents.\n\r\n\r");
                n = sysace_fwrite(dataBuf, 1, sizeof(dataBuf), ptest);
                xil_printf("# of bytes written: %d\n\r", n);
                sysace_fclose(ptest);
        } else {
               xil_printf("Failed to open: check if file exists\n\r");
        }
        if (verbose) {
           i = 0;
           while( i < (1024)) {
           xil_printf("%d\n\r", dataBuf[i]);
           i++;
           }
        }
}
void writeDDR2CF(char FileName[], Xuint32 slices)
//Writes the contents of DDR to .dat file in CF card
//Data is written to file byte by byte in BIG ENDIAN format (MSByte in lower position)
{
       // Reset the sysace controller to clean any bad state, leave it in MPU mode
               sysace_reset();
               SYSACE_FILE *ptest;
               Xuint32 n = 0;
               Xuint32 tempbuf[1];
               ptest = sysace_fopen(FileName ,"w" );

        if(ptest) {
               ddr_offset = 0;
               k=0;
               xil_printf("\nWriting data from DDR to file.\n\r");
                       //the first 4 bytes in the file will have this number, which is
                       tempbuf[0] = (slices * 1024UL * 4UL);        //the number of bytes
that the file will contain, as a 32bit integer
                       n = n + sysace_fwrite(tempbuf, 1, 4, ptest);
               for(k=0;k < slices;k++){
                       for(i=0;i<1024;i++){
                               mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+ddr_offset);
                               dataBuf[i] = *mpmc_p;
                               ddr_offset = ddr_offset+4;
                       }
                       n = n + sysace_fwrite(dataBuf, 1, sizeof(dataBuf), ptest);
                       xil_printf("# of bytes written: %d\n\r", n);
               }
               sysace_fclose(ptest);
        }
        else {
               xil_printf("Failed to open: check if file exists\n\r");
        }
}
void writePSD2CF(char FileName[])
//Writes the contents of dataBuf to .dat file in CF card
//Data is written to file byte by byte in BIG ENDIAN format (MSByte in lower position)
{
       // Reset the sysace controller to clean any bad state, leave it in MPU mode
               sysace_reset();
               SYSACE_FILE *ptest;
               Xuint32 n = 0;
               Xuint32 tempbuf[1];
               ptest = sysace_fopen(FileName ,"w" );

        if(ptest) {
               xil_printf("\nWriting PSD data from data buffer to file.\n\r");
                       tempbuf[0] = 2048;                                      //the first 4
bytes in the file will have this number which is
                       n = n + sysace_fwrite(tempbuf, 1, 4, ptest); //the number of bytes to
be written in file
               n = n + sysace_fwrite(dataBuf, 1, 2048, ptest); //512 components, 2048 bytes
               xil_printf("# of bytes written: %d\n\r", n);
               sysace_fclose(ptest);
        }
        else {
               xil_printf("Failed to open: check if file exists\n\r");
        }
}
#endif
void sysace_reset(){
```

```
	// Reset the sysace controller to clean any bad state, leave it in MPU mode
	sysace = (Xuint32) sysace_p;

	XSysAce_RegWrite16(sysace + XSA_BMR_OFFSET, XSA_BMR_16BIT_MASK);
	XSysAce_SetControlReg(sysace, XSA_CR_CFGSEL_MASK | XSA_CR_FORCECFGMODE_MASK);
	XSysAce_SetControlReg(sysace, XSA_CR_CFGSEL_MASK | XSA_CR_FORCECFGMODE_MASK |
XSA_CR_CFGRESET_MASK);
	XSysAce_SetControlReg(sysace, XSA_CR_CFGSEL_MASK | XSA_CR_FORCECFGMODE_MASK);
}
void buf2fifo(Xuint32 *buffer){
//moves data from 32bit buffer to writeFIFO
// Reset read and write packet FIFOs to initial state
	MAINFFT_mResetWriteFIFO(mainfft);
	MAINFFT_mResetReadFIFO(mainfft);
	for(i=0;i<1024;i++){
		MAINFFT_mWriteToFIFO(mainfft,0, buffer[i]);
	}
}
void fifo2buf(Xuint32 *buffer){
//moves data from readFIFO32bit to buffer
	for(i=0;i<1024;i++){
		buffer[i] = MAINFFT_mReadFromFIFO(mainfft,0);
	}
}
void ddr2fifo(Xuint32 offset){
	//moves data from ddr2 to fifo, starting from offset.
	for(i=0;i<1024;i++){
		mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
		MAINFFT_mWriteToFIFO(mainfft,0, *mpmc_p);
		offset = offset+4;
	}

}
void fifo2ddrpsd(Xuint32 offset){
/*moves data from fifo to ddr, starting from offset
The raw data contents of DDR will be replaced by the power spectrum components
For each slice of 1024 FFT points, we will keep 512 points of power spectrum,
due to symmetry of the spectrum, only the upper halve (according to FFT core manual)
PSD data will be in Q0.30 format in DDR2 memory
*/

	offset = offset/2;
	for(i=0;i<1024;i++){			//first move data to buffer
		dataBuf[i] = MAINFFT_mReadFromFIFO(mainfft,0);
	}
	//move first the zero (DC) component to buffer
	mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
	*mpmc_p = (dataBuf[0] & 0x0000ffff );		//keep first the zero component
	offset = offset+4;
	for(i=1023;i>511;i--){
		tempImag = (Xint16)((dataBuf[i] & 0xffff0000 )>>16) ;
		tempReal = (Xint16)(dataBuf[i] & 0x0000ffff );
		mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
		tempPsd = (((Xint32)tempReal * (Xint32)tempReal)+((Xint32)tempImag *
(Xint32)tempImag)) << 1;
		//calculate the power spectrum. Multiply *2 as we take half the points of
spectrum
		//now DDR has power spectrum components in signed Q0.30 format
		*mpmc_p = tempPsd;
		offset = offset+4;
		}

}
void fifo2ddr(Xuint32 offset){
/*moves back data from fifo to ddr, starting from offset, for 1 slice of 1024 points
The contents of DDR will be replaced by the FFT components which are as signed integer
complex numbers
2 MSBytes representing real part and 2 LSBytes the imaginary part, both in signed Q0.15
format
*/
	for(i=0;i<1024;i++){
		mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
		*mpmc_p = MAINFFT_mReadFromFIFO(mainfft,0);
		offset = offset+4;
	}
}
void floatshow(Xuint32 offset){
```

```
	/* Display 1024 DDR2 data to terminal as floating complex numbers
	Data are read as 4 bytes numbers, 2 MSBytes representing real part and 2 LSBytes the
imaginary part.
	Numbers are in Q0.15 format, so they are converted to float
	*/
			for(i=0;i<1024;i++){
				mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
				dataBuf[i] = *mpmc_p;
				tempImag = (Xint16)((dataBuf[i] & 0xffff0000 )>>16) ;
				tempReal = (Xint16)(dataBuf[i] & 0x0000ffff );
				realFFT.f = tempReal/32768.0f; //convert to float
				imagFFT.f = tempImag/32768.0f;
				printf("%0.15f + %0.15fj\n\r",realFFT.f,imagFFT.f); //we use
printf to print floating points
				offset = offset+4;
			}
}
void intshow(Xuint32 offset){
	/* Display 1024 DDR2 data to terminal as signed integer complex numbers
	Data are read as 4 bytes numbers, 2 MSBytes representing real part and 2 LSBytes the
imaginary part.
	Numbers are in Q0.15 format.
	*/
			for(k=0;k < 1024 ;k++){
				mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
				dataBuf[i] = *mpmc_p;
				tempImag = (Xint16)((dataBuf[i] & 0xffff0000 )>>16) ;
				tempReal = (Xint16)(dataBuf[i] & 0x0000ffff );
				xil_printf("%d + %dj\n\r",tempReal,tempImag);
				offset = offset+4;
			}
}
void floatpsdshow(Xuint32 offset){
	/* Display 1024 DDR2 PSD data to terminal as floating numbers
	Data are read as 4 bytes numbers, which are in Q0.30 format, so they are converted
to float accordingly
	*/
			for(i=0;i<512;i++){
				mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
				dataBuf[i] = *mpmc_p;
				realFFT.f = (dataBuf[i])/1073741824.0f; //convert to float
				printf("%0.15f\n\r",realFFT.f); //we use printf to print
floating points
				offset = offset+4;
			}
}
void intpsdshow(Xuint32 offset){
	/* Display 1024 DDR2 PSD data to terminal as floating numbers
		Data are read as 4 bytes numbers, which are in Q0.30 format
		*/
			for(k=0;k < 512 ;k++){
			mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
			dataBuf[i] = *mpmc_p;
			xil_printf("%d\n\r",dataBuf[i]);
			offset = offset+4;
			}
}
void dcRemove(Xuint32 offset){
	Xuint32 offset2 = offset;
	Xuint32 tempdc;
	for(k=0;k<nslices;k++){
		tempdc = 0;
		for(i=0;i<1024;i++){
			mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset2);
			tempdc = *mpmc_p + tempdc;
			offset2 = offset2+4;
		}
		tempdc = tempdc/1024UL;
		//xil_printf ("DC comp from %d slice:%d\r\r\n",k,tempdc);
		for(i=0;i<1024;i++){
			mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
			tempReal = (Xint16)(*mpmc_p - tempdc);
	//save as signed 16 bit integer
			*mpmc_p = ((Xuint32)tempReal & 0x0000ffff);		//we
prevent as zero the image part of DDR
			offset = offset+4;
		}
```

```
	}
}
//void mem2init(Xuint32 offset){
//	for(k=0;k < 1024 ;k++){
//		usermem2_p = (Xuint32 *) (XPAR_MAINFFT_0_MEM2_BASEADDR+offset);
//		*usermem2_p = 0;
//		offset = offset+4;
//	}
//}
void avePSD(){
	/* Calculates average psd from all slices and saves results to dataBuf, for storing
to CF*/
	Xuint32 offset=0;
	Xuint32 tmp=0;
	for(i=0;i<512;i++){
		aveBuf[i] = 0;			//zeroing buffer
	}
	for(k=0;k<nslices;k++){
		for(i=0;i<512;i++){
			mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+offset);
			tmp = *mpmc_p;
			aveBuf[i] = (u64)tmp + aveBuf[i];		//add all components of
same frequency
			offset = offset+4;
		}
	}
	for(i=0;i<512;i++){
		aveBuf[i] = aveBuf[i]/(u64)nslices;
	//calc the average
		dataBuf[i] = (Xuint32)aveBuf[i];
		//xil_printf ("%d\r\n",dataBuf[i]);
	}
}
void ADC_ReadData(Xuint32 sampnum)
{
  static Xuint32 adc_data;
  Xuint32 sl,tmpreg;
  ddr_offset = 0;

  for(sl=0;sl<sampnum;sl++){
	while((MAINFFT_mReadSlaveReg2(mainfft, 0) & 0x00000008) == 0x00000000)
	{}//wait until EOC signal is received
  // Read Data from ADC output
	adc_data = MAINFFT_mReadSlaveReg4(mainfft, 0);
	//send handshake signal to hardware that data has been read
	MAINFFT_mWriteSlaveReg2(mainfft, 0, DATA_TAKEN);
	MAINFFT_mWriteSlaveReg2(mainfft, 0, ADC_START);
	//Every time we have to specify the address of the memory
	mpmc_p = (Xuint32 *) (XPAR_MPMC_0_MPMC_BASEADDR+ddr_offset);
	*mpmc_p = adc_data;
	//xil_printf("Address and data written 0x%08x = 0x%x\n\r", mpmc_p,*mpmc_p);
	ddr_offset = ddr_offset+4 ;  //4 byte alignment
	sampOnMem=sampOnMem+1;
  }
}
void filesPrep()
{
//clear string buffers and initialize them
memset(rawFile, '\0', 20);
strcpy(rawFile,"a:\\rawD");
memset(fftFile, '\0', 20);
strcpy(fftFile,"a:\\fftD");
memset(psdFile, '\0', 20);
strcpy(psdFile,"a:\\psdD");
memset(psdaveFile, '\0', 20);
strcpy(psdaveFile,"a:\\psaD");
memset(fxstr, '\0', 4);
sprintf(fxstr, "%d", fx);

strcat(rawFile, fxstr);
strcat(fftFile, fxstr);
strcat(psdFile, fxstr);
strcat(psdaveFile, fxstr);
fx=fx+1;  //increment prefix for next files
}
```