ΤΟΜΕΑΣ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ – ΤΜΗΜΑ ΗΛΕΚΤΡΟΝΙΚΗΣ
ΕΡΓΑΣΤΗΡΙΟ ΜΙΚΡΟΚΥΜΑΤΙΚΩΝ ΕΠΙΚΟΙΝΩΝΙΩΝ ΚΑΙ
ΗΛΕΚΤΡΟΜΑΓΝΗΤΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

Τ.Ε.Ι ΚΡΗΤΗΣ – ΠΑΡΑΡΤΗΜΑ ΧΑΝΙΩΝ

ΧΑΝΙΑ
2010

FREQUENCY METER

Ψηφιακό Συχνόμετρο 0-1.5GHz, υψηλής ακρίβειας (0.1Hz)

υπό

Νικολάου Ν. Πλάτων &

Τόλια Θ. Ηλία

Επιβλέπων: Γεώργιος Α.Αδαμίδης, Εργαστηριακός Συνεργάτης

Πτυχιακή Εργασία

για το τμήμα Ηλεκτρονικής Παραρτήματος Χανίων

του Τ.Ε.Ι Κρήτης

Τομέας Τηλεπικοινωνιών

Χανιά, Ιούνιος 2010

FREQUENCY METER
Frequency measurement device



Students:

Platon Nikolaou
Ilias Tolias

Chania 2010

PROLOGUE

The Frequency Meter is an electrical device, which is able to perform frequency measurements with high resolution. The device's theory of operation is based on counting the instances of any periodic waveform during a precise time interval.

The present Frequency Meter performs automated measurements in a wide frequency range by using a PIC18F2620-I/SP microcontroller.

The usage of the Frequency Meter is not only to perform frequency measurements, but it can be also used, for example, in displaying frequency values from a radio receiver and other synchronous electronic measurement devices.

ΠΕΡΙΕΧΟΜΕΝΑ

Σελίδα

ΕΥΧΑΡΙΣΤΙΕΣ.....8
ΠΕΡΙΛΗΨΗ 9
ΕΙΣΑΓΩΓΗ 10

ΜΕΡΟΣ ΠΡΩΤΟ: Αρχές λειτουργίας

1.1 Αρχές μέτρησης συχνότητας.....12
1.2 Μέτρηση συχνότητας με χρήση του PIC18F2X2.....13
1.2.1 Η εσωτερική δομή του timer0.....13
1.2.1.1 Λειτουργία του timer0 14
1.2.2 Τρόπος μέτρησης συχνότητας με χρήση counters.....15

ΜΕΡΟΣ ΔΕΥΤΕΡΟ: Περιγραφή του κυκλώματος

2.1 Διάγραμμα βαθμίδων του Frequency counter.....17
2.2 Περιγραφή διαγράμματος βαθμίδων.....17
2.3 Ανάλυση βαθμίδων κυκλώματος 19
2.3.1 Βαθμίδα 1 19
2.3.1.1 Ανάλυση πρώτης βαθμίδας 19
2.3.2 Βαθμίδα 2.....20
2.3.2.1 Ανάλυση δεύτερης βαθμίδας.....21
2.3.3 Βαθμίδα 3.....22
2.3.3.1 Ανάλυση τρίτης βαθμίδας.....22
2.3.2.2 Ανάλυση της απεικόνισης μέσω λογικής σχεδίασης.....23
2.3.4 Βαθμίδα 4.....25
2.3.4.1 Ανάλυση τέταρτης βαθμίδας.....25

ΜΕΡΟΣ ΤΡΙΤΟ: Περιγραφή του λογισμικού & του αλγόριθμου λειτουργίας

3.1 Αλγόριθμος λειτουργίας 28
3.2 Software του συχνομέτρου.....29-66
3.3 Αρχεία του software.....67
3.4 Επεξήγηση βασικών ρουτινών.....67
3.4.1 Υπορουτίνα «display freq(0)» 67
3.4.2 Υπορουτίνα «channel».....68
3.4.3 Υπορουτίνα «function».....68
3.4.4 Υπορουτίνα «units».....69
3.4.5 Υπορουτίνα «disst».....69
3.4.6 Υπορουτίνα «led».....69
3.4.7 Υπορουτίνα «freq (void)».....69
3.4.8 Υπορουτίνα «action».....70
3.4.9 Υπορουτίνα «counterup».....70
3.5 Υπορουτίνα «prescaler».....70

3.6 Ρουτίνα Main.....71-74
3.6.1 Διάγραμμα ροής της ρουτίνας Main.....75
3.6.2 Ανάλυση της ρουτίνας Main…….....76

ΜΕΡΟΣ ΤΕΤΑΡΤΟ: Χαρακτηριστικά της συσκευής

4.1 Ακρίβεια και σφάλματα στη μέτρηση συχνότητας …….....80
4.2 Ευαισθησία.....80
4.3 Πρόσθετες δυνατότητες της συσκευής …….....81

ΜΕΡΟΣ ΠΕΜΠΤΟ: Τυπωμένο κύκλωμα και λεπτομέρειες κατασκευής

5.1 Κατασκευή πλακέτας…….....83
5.2 Βοηθητικές κατασκευές…….....83
5.3 Πρόσοψη Frequency counter…….....83
5.4 Τυπωμένο κύκλωμα.....84
5.4.1 Τυπωμένο κύκλωμα πλακέτας.....84
5.4.2 Τυπωμένο κύκλωμα πληκτρολογίου.....85

ΜΕΡΟΣ ΕΚΤΟ: Βαθμονόμηση της συσκευής

6.1 Τρόπος βαθμονόμησης της συσκευής…….....88

ΜΕΡΟΣ ΕΒΔΟΜΟ: Εγχειρίδιο χρήσης

7.1 Εγχειρίδιο χρήσης της συσκευής…….....90-92
7.2 Τεχνικά χαρακτηριστικά.....93
7.3 ΒΙΒΛΙΟΓΡΑΦΙΑ…….....94

ΠΕΡΙΕΧΟΜΕΝΑ ΨΗΦΙΑΚΟΥ ΔΙΣΚΟΥ …….....95

ΕΥΧΑΡΙΣΤΙΕΣ

Θα θέλαμε να εκφράσουμε τις ευχαριστίες και την εκτίμησή μας στον κ. Αδαμίδη Γεώργιο, εργαστηριακό καθηγητή και επιβλέπων της πτυχιακής μας εργασίας, του τμήματος Ηλεκτρονικής του ΤΕΙ Κρήτης (παράρτημα Χανίων) για την καθοδήγηση και πολύτιμη βοήθεια που μας προσέφερε. Επίσης τους φίλους μας Κώστα, Μιχάλη, Γιάννη και Φανή για την σημαντική στήριξή τους.

ΠΕΡΙΛΗΨΗ

Στην παρούσα εργασία παρουσιάζεται ο σχεδιασμός, η κατασκευή και η βαθμονόμηση μιας ηλεκτρονικής συσκευής συχνομέτρου (Frequency counter) στο Εργαστήριο Μικροκυματικών Επικοινωνιών και Ηλεκτρομαγνητικών Εφαρμογών του Τ.Ε.Ι Κρήτης (παράρτημα Χανίων). Η προσπάθεια μας επικεντρώνεται στους παρακάτω στόχους:

- Υλοποίηση συσκευής συχνομέτρου για την ακριβή μέτρηση συχνότητας με ακρίβεια έως και 0.1Hz.
- Αυτοματοποιημένη μέτρηση στην περιοχή 0.1Hz έως 1.5GHz.
- Μικρό μέγεθος και βάρος συσκευής.
- Μικρό κόστος
- Απουσία μηχανικών μερών
- Απεικόνιση σε οθόνη τύπου υγρού κρυστάλλου (LCD) 2x16 χαρακτήρων.
- Χρήση μικροεπεξεργαστή για ακριβείς αριθμητικούς υπολογισμούς με χρήση αριθμών κινητής υποδιαστολής 32-bit (32 bit floating point number, representation according to IEEE 754 standard for single precision floats)

ΕΙΣΑΓΩΓΗ

Το συχνόμετρο είναι μία συσκευή μέτρησης, απαραίτητη σε κάθε ηλεκτρονικό εργαστήριο. Υπάρχουν διάφοροι τύποι συχνομέτρων με ποικίλα χαρακτηριστικά και με ευρύ φάσμα δυνατοτήτων. Το κόστος των επαγγελματικών συσκευών συχνομέτρων που κυκλοφορούν στο εμπόριο είναι αρκετά υψηλό. Επιπλέον οι περισσότερες συσκευές που κυκλοφορούν στο εμπόριο καταλαμβάνουν αρκετό χώρο στον πάγκο του ηλεκτρονικού εργαστηρίου.

Λαμβάνοντας υπόψη τις απαιτήσεις του μέσου τεχνικού για ευκολία και ακρίβεια στη μέτρηση, αναρωτηθήκαμε αν θα ήταν δυνατό να κατασκευάσουμε ένα συχνόμετρο που να ικανοποιεί αυτές τις απαιτήσεις και συγχρόνως να είναι αρκετά μικρότερο και ασύγκριτα φθηνότερο από τις αντίστοιχες συσκευές του εμπορίου. Στην πορεία της διερεύνησής μας, διαπιστώσαμε ότι ένα τέτοιο εγχείρημα είναι απολύτως εφικτό με βάση την τεχνογνωσία και τις δυνατότητες μας, στο Τ.Ε.Ι Κρήτης.

Η συσκευή που τελικώς κατασκευάσαμε είναι εφάμιλλα ακριβής και ασύγκριτα φθηνότερη από παρόμοιες συσκευές της αγοράς. Για να κατασκευαστεί ένα συχνόμετρο με καλή ευαισθησία εισόδου, το οποίο θα μπορεί να μετρήσει σήματα με πλάτος μικρότερο από μερικά mV σε συχνότητες από 0.1Hz έως 1.5GHz, είναι απαραίτητη η χρησιμοποίηση δύο διαφορετικών βαθμίδων εισόδου. Κάθε βαθμίδα εισόδου διαθέτει τη δική της είσοδο, CH1 και CH2, αντίστοιχα. Η πρώτη είσοδος καλύπτει την περιοχή συχνοτήτων από 0.1Hz-100MHz, προσφέροντας ακρίβεια μέχρι 0.1Hz ενώ η δεύτερη, CH2, καλύπτει τη μπάντα συχνοτήτων από 70MHz-1.5GHz προσφέρωντας ακρίβεια μέχρι 4 Hz. Ο χρήστης είναι ικανός με το πάτημα ενός κουμπιού να επιλέξει το κανάλι που επιθυμεί.

Η συσκευή που αναπτύξαμε μπορεί να χρησιμοποιηθεί είτε αυτόνομα ως συσκευή μετρήσεων ακριβείας, είτε να αποτελέσει την βασική βαθμίδα σε κυκλώματα πλέον σύνθετων συχνομέτρων ή σε κυκλώματα απεικόνισης συχνότητας ραδιοφωνικών δεκτών, πομπών και άλλων συσκευών.

ΜΕΡΟΣ ΠΡΩΤΟ

ΑΡΧΕΣ ΛΕΙΤΟΥΡΓΙΑΣ

1.1) Αρχές μέτρησης συχνότητας

Η τιμή της βασικής συχνότητας μίας περιοδικής κυματομορφής αποτελεί φυσική ποσότητα που μπορεί εύκολα να προσδιοριστεί, μετρώντας τους κύκλους επανάληψης της κυματομορφής σε συγκεκριμένο χρονικό διάστημα. Καθώς γνωρίζουμε από την φυσική, η συχνότητα F είναι ένα φυσικό μέγεθος που ορίζεται από τη σχέση:

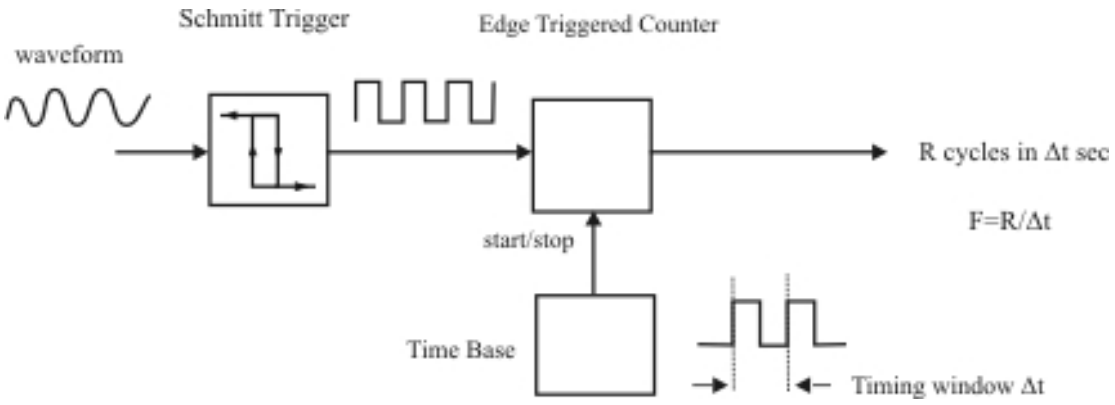
$$F = \frac{N}{t} \qquad \textbf{(1)}$$

Όπου N, ο αριθμός των κύκλων επανάληψης στο χρονικό διάστημα t.
Η μονάδα μέτρησης της συχνότητας είναι το 1Hz (Hertz) και ορίζεται ως ένας κύκλος επανάληψης ανά δευτερόλεπτο. Επομένως η μέτρηση συχνότητας μπορεί να αναχθεί σε απλή μέτρηση κύκλων επανάληψης R σε προκαθορισμένο χρονικό διάστημα ΔT, βάσει της φόρμουλας:

$$F = \frac{R}{\Delta \mathrm{T}} \qquad \textbf{(2)}$$

Όπου ΔT ένα προκαθορισμένο ακριβές χρονικό διάστημα το οποίο εύκολα μπορεί να καθοριστεί με ακρίβεια κάνοντας χρήση ενός ταλαντωτή αναφοράς-βάσης χρόνου (reference oscillator). Το πλήθος των κύκλων επανάληψης R, μιας περιοδικής κυματομορφής μπορεί εύκολα να μετρηθεί από ένα ψηφιακό κύκλωμα που ονομάζεται μετρητής (counter) αφού πρώτα η κυματομορφή μορφοποιηθεί σε δυαδική παλμοσειρά διατηρώντας όλα τα χαρακτηριστικά της, διερχόμενη μέσα από ένα κύκλωμα συγκριτή τύπου Schmitt Trigger.

Επομένως η μέτρηση συχνότητας μιας τυχούσας κυματομορφής μπορεί να αναχθεί στην μέτρηση των παλμών μιας δυαδικής παλμοσειράς, σύγχρονης (coherent) με την αρχική κυματομορφή, από έναν μετρητή (counter), σε ένα συγκεκριμένο και σαφώς ορισμένο χρονικό διάστημα που ορίζεται από μια σταθερή βάση χρόνου ενός ταλαντωτή αναφοράς.



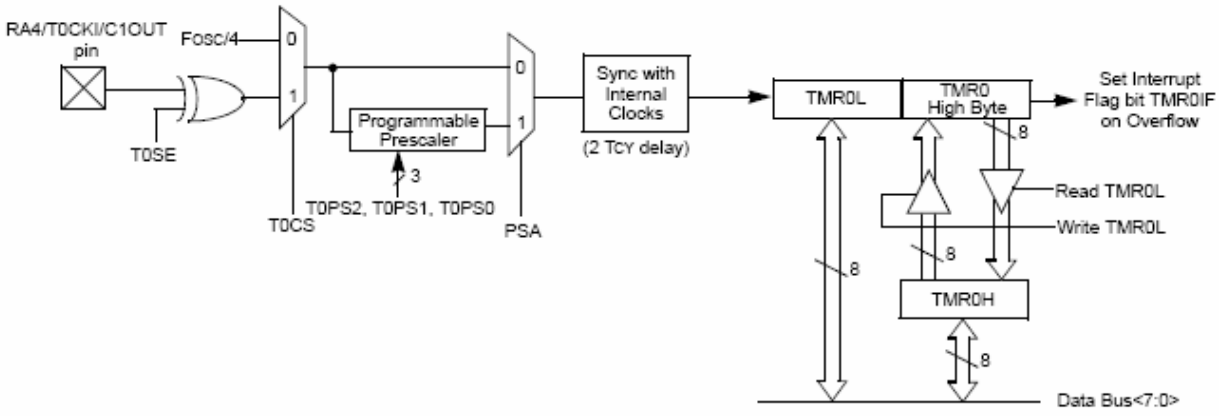
Σχήμα 1. Αρχή μέτρησης συχνότητας

1.2) Μέτρηση συχνότητας με χρήση του PIC18F2X2

Η διαδικασία μέτρησης συχνότητας, που περιγράφηκε στη παράγραφο 1.1, μπορεί να υλοποιηθεί κάνοντας χρήση ενός μικροελεγκτή (microcontroller). Ένας μικροελεγκτής είναι μία μονάδα βασικού επεξεργαστή (microprocessor) που περιέχει επιπρόσθετα, κάποια περιφερειακά στο ίδιο ολοκληρωμένο κύκλωμα. Όλοι οι μικροελεγκτές που διαθέτουν κάποιον εσωτερικό counter, πλήρως ελεγχόμενο από software, μπορούν να χρησιμοποιηθούν για την κατασκευή ενός συχνομέτρου. Ανάμεσα σε πλήθος λύσεων, για την δική μας κατασκευή διαλέξαμε τον PIC18F242 της Microchip.

Ο PIC18F242 είναι ένας 8bit μικροελεγκτής τύπου Flash που διαθέτει μνήμη 4Kbytes, 10 κανάλια των 10bit(A/D channels), 3 θύρες διπλής κατεύθυνσης και 1 θύρα εισόδου, 4 timers/counters , 2 σταθερούς CCP καταχωρητές (capture/compare/PWM(master/slave duty ducle register) modules), ενώ η συχνότητα χρονισμού του (clock), μπορεί να είναι έως και 20MHz. Για την υλοποίηση του αλγορίθμου μέτρησης συχνότητας με τον PIC κάνουμε χρήση του ενός από τους τέσσερις timers που αυτός διαθέτει και συγκεκριμένα τον timer0. Διαβάζοντας το φυλλάδιο πληροφοριών του PIC παρατηρούμε ότι οι ελάχιστοι χρόνοι ανόδου και καθόδου της κυματομορφής εισόδου στο pin TOCKI είναι 10nsec, οπότε η μεγαλύτερη συχνότητα ρολογιού που μπορεί να δεχτεί το pin TOCKI είναι 50MHz. Γι' αυτό αναμένεται ότι ένας PIC μπορεί να χρησιμοποιηθεί απευθείας για μέτρηση συχνοτήτων πάνω από 50MHz. Για να το ξεπεράσουμε αυτό χρησιμοποιούμε έναν εξωτερικό μετρητή 4-bit. Έτσι το ανώτατο όριο συχνότητας καθορίζεται μόνο από την μέγιστη ταχύτητα του εξωτερικού μετρητή και όχι από τον PIC.

1.2.1) Η εσωτερική δομή του timer0



Σχήμα 1. Η εσωτερική δομή του timer0

Η εσωτερική δομή του timer0 παρουσιάζεται στο σχήμα 1. Αυτός αποτελείται από :

- Έναν προγραμματιζόμενο προδιαιρέτη
- Έναν 16bit counter, πλήρως ελεγχόμενο από software

- Ένα σύστημα συγχρονισμού της πηγής με εσωτερικό ρολόι
- Τον καταχωρητή εισόδου/ εξόδου

1.2.1.1) Λειτουργία του timer0.

Ο timer0 έχει τα παρακάτω χαρακτηριστικά:

- Δυνατότητα λειτουργίας ως 8 bit ή ως 16 bit asynchronous ή synchronous timer/counter – σημειωτέου ότι στην τρέχουσα εφαρμογή επιλέγουμε την 16bit λειτουργία
- Ελεγχόμενος μέσω software
- Περιέχει προδιαιρέτη 8bit (prescaler), προγραμματιζόμενου λόγου διαίρεσης αλλά όχι άμεσα αναγνώσιμου μέσω software - σημειωτέου ότι στην τρέχουσα εφαρμογή επιλέγουμε λόγο διαίρεσης 256 για τον προδιαιρέτη.
- Υποστηρίζει επιλογή του τύπου πυροδότησης (trigger) του εξωτερικού ρολογιού (high to low ή low to high)
- Μέγιστη συχνότητα ασύγχρονης λειτουργίας, 50MHz.

Ο 8bit καταχωρητής T0CON είναι αυτός στον οποίο μέσω software δίνουμε μια τιμή και ελέγχει την κατάσταση λειτουργίας (8bit ή 16bit mode) του Timer0, καθώς επίσης και τον λόγο διαίρεσης του προδιαιρέτη (prescaler). Ο timer0 μπορεί να λειτουργήσει είτε σαν χρονιστής(timer), είτε σαν μετρητής (counter). Αυτό μπορούμε να το επιλέξουμε με την τιμή που θα δώσουμε στο bit T0CS του T0CON. Αν αυτό έχει την τιμή “0”, ο timer θα αυξάνει το περιεχόμενό του σε κάθε κύκλο μηχανής(χωρίς τον prescaler). Αν το T0CS έχει τιμή “1” έχουμε λειτουργία μετρητή(counter), δηλαδή ο timer0 θα αυξάνει σε κάθε θετικό ή αρνητικό μέτωπο παλμού το οποίο εφαρμόζεται στο pin RA4/TOCKI. Αν το μέτωπο του παλμού που αυξάνει τον timer είναι αρνητικό ή θετικό, καθορίζεται από το bit T0SE. Αν το Bit T0SE είναι “0”, επιλέγουμε αύξηση με θετικό μέτωπο παλμού, αν είναι “1”, με αρνητικό μέτωπο παλμού.

PRESCALER

Ένας 8bit counter, που αποτελείται από 8 JK flip-flops, έχει διατεθεί αποκλειστικά για τον timer0, σαν προδιαιρέτης (prescaler). Αυτός δεν είναι άμεσα αναγνώσιμος, παρά μόνο μέσω software. Το bit PSA, του T0CON, καθορίζει αν θα χρησιμοποιηθεί ο prescaler ή όχι, ενώ τα bits T0PS0, T0PS1, TOPS2 αποφασίζουν τον λόγο διαίρεσης του prescaler. Θέτοντας το PSA στην τιμή 0 παρεμβάλλουμε τον prescaler στην υπομονάδα του timer0 και μπορούμε να επιλέξουμε λόγο διαίρεσης 1:2, 1:4, …., 1:256.

1.2.2) Τρόπος μέτρησης συχνότητας με χρήση counters

Η άγνωστη συχνότητα f, τυχούσας δυαδικής παλμοσειράς που εφαρμόζεται στην είσοδο του 4-bit counter, υπολογίζεται από τη σχέση (1) όπως αναφέρθηκε στην παράγραφο (1.1). Για να επιτευχθούν μετρήσεις συχνότητας μέχρι τα 1.5GHz, με ακρίβεια 0.1Hz, απαιτείται μετρητής των 28-bit. Ο μετρητής των 28-bit που χρησιμοποιούμε αποτελείται από την εσωτερική μονάδα 16-bit του Timer0 (διαμορφωμένη σαν ένας μετρητής 16-bit), έναν επιπρόσθετο εξωτερικό μετρητή 4-bit (74F161) και τον εσωτερικό προδιαιρέτη (prescaler) του PIC (με λόγο διαίρεσης 1:256) σε σειρά. Έτσι έχουμε στο σύνολο 28-bit. Το πλήθος N των παλμών που μετρώνται στο χρονικό διάστημα t, προσδιορίζεται από τις τιμές A, B & C, σύμφωνα με τη σχέση:

$$N = A + B * 16 + C * 4096 \qquad \textbf{(3)}$$

Όπου A= η τιμή του 4 bit counter
B= η τιμή του 8 bit Prescaler
C= η τιμή του 16 bit counter

Η τιμή A, του εξωτερικού counter “διαβάζεται” από τον επεξεργαστή μέσω των I/O pins που διαθέτει. Η τιμή C, του εσωτερικού 16-bit counter, είναι πλήρως ελεγχόμενη και προσπελάσιμη μέσω software. Δεν ισχύει όμως το ίδιο και για την τιμή B, του prescaler. Ο εσωτερικός προδιαιρέτης (prescaler) του PIC, με λόγο διαίρεσης 1:256, είναι στη πραγματικότητα ένας 8-bit μετρητής αλλά δεν είναι άμεσα αναγνώσιμος μέσω software και για το λόγο αυτό, εφαρμόζουμε την παρακάτω τεχνική:

Αρχικά, βρίσκουμε την τιμή των μετρητών 16-bit και 4-bit και στη συνέχεια εφαρμόζουμε παλμούς στην είσοδο του prescaler, έως ότου αυτός υπερχειλίσει. Μετά από κάθε εφαμογή παλμού, ελέγχουμε την τιμή του Timer0 για δούμε εάν ο Timer0 έχει αυξηθεί. Εάν ο αριθμός των παλμών που απαιτήθηκαν για να προκαλέσουν τον Timer0 να αυξηθεί κατά 1 είναι R, τότε η τιμή του prescaler μπορεί να υπολογισθεί από τη σχέση B=(256-R).

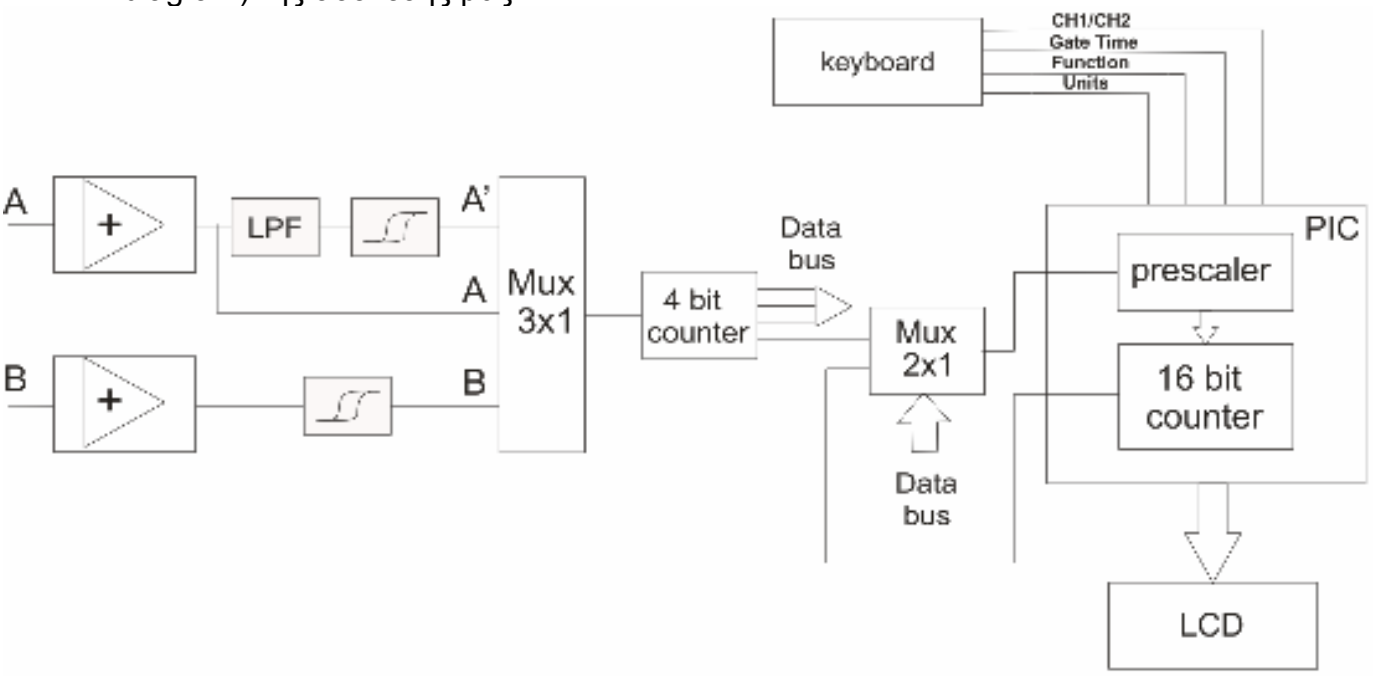
Στη συνέχεια ο υπολογισμός των μετρητών σταματάει και ξεκινάει η ανάγνωσή τους. Μόλις ο επεξεργαστής διαβάσει την υπολογισμένη τιμή του prescaler και τις τιμές του Timer0 και του εξωτερικού μετρητή (74F161), χρησιμοποιεί τη σχέση **(3)** και αφού υπολογίσει τον αριθμό των παλμών κάνει την απεικόνιση στο LCD.

ΜΕΡΟΣ ΔΕΥΤΕΡΟ

ΠΕΡΙΓΡΑΦΗ ΚΥΚΛΩΜΑΤΟΣ

2.1) Διάγραμμα βαθμίδων του frequency counter

Στο σχήμα εικονίζεται το απλοποιημένο διάγραμμα βαθμίδων (block diagram) της συσκευής μας.



Σχ.1 Απλοποιημένο διάγραμμα βαθμίδων συσκευής

2.2) Περιγραφή διαγράμματος βαθμίδων

Η βαθμίδα Α αποτελεί την πρώτη βαθμίδα εισόδου και αφορά σήματα συχνότητας από 0.1MHz-99MHz. Εδώ επιτυγχάνουμε την ρύθμιση της στάθμης σύγκρισης καθώς και την ρύθμιση της ευαισθησίας μέσω των επιμέρους κυκλωμάτων. Λόγω της ύπαρξης του συγκριτή MAX 9203 επιτυγχάνουμε την παραγωγή ψηφιακών σταθμών TTL από 0.2V-4.5V. Η βαθμίδα Β αποτελεί την δεύτερη βαθμίδα εισόδου και αφορά σήματα εισόδου από 70MHz-1.5GHz. Αποτελείται από έναν προδιαιρέτη (MC12080 prescaler) με λόγο διαίρεσης 1:40 καθώς και από έναν συγκριτή που μετατρέπει την ημιτονοειδής έξοδο του prescaler σε ψηφιακό σήμα πλάτους 1.2Vp-p.

Κύριος σκοπός των βαθμίδων εισόδου είναι να μετατρέπουν τα σήματα εισόδου σε αντίστοιχα δυαδικά ισοδύναμα μέσω του συγκριτή υψηλής ταχύτητας MAX 9203 που διαθέτει η κάθε βαθμίδα εισόδου. Ο MAX είναι ένας συγκριτής υψηλής ταχύτητας, ο οποίος έχει μια καθυστέρηση διάδοσης των 7msec περίπου και μια ευαισθησία εισόδου 5mV περίπου. Αυτά τα χαρακτηριστικά κάνουν τον MAX 9203 ιδανικό για συχνότητες μέχρι τα 100MHz. Για υψηλότερες συχνότητες, πρέπει να χρησιμοποιηθεί ένας επιπρόσθετος προδιαιρέτης (prescaler). Γι'αυτό χρησιμοποιούμε τον prescaler MC12080, στην δεύτερη βαθμίδα εισόδου. Χρησιμοποιώντας τον MC12080, είμαστε ικανοί να επεκτείνουμε το φάσμα συχνοτήτων μέχρι τα 1.5GHz.

Η επόμενη βαθμίδα αποτελείται από 2 πολυπλέκτες και έναν εξωτερικό μετρητή 4-bit. Ο μετρητής αυτός είναι απαραίτητος διότι μαζί με τον 8-bit prescaler και τον 16-bit counter του PIC, εξασφαλίζει τα 28-bit που είναι απαραίτητα για τη μέτρηση συχνοτήτων μέχρι 1.5 GHz. Στο πρώτο πολυπλέκτη (MUX 3x1) οδηγούνται οι έξοδοι των βαθμίδων εισόδου A & B. Επιπρόσθετα, οδηγείται στον πολυπλέκτη και μια τρίτη είσοδος A' για συχνότητες κάτω των 100MHz που φιλτράρεται μέσω ενός LPF (Low Pass Filter). Ο πολυπλέκτης (MUX 2x1), στην έξοδο του εξωτερικού μετρητή 4-bit, υπάρχει είτε για να δουλεύει κανονικά σαν μετρητής, είτε για να απομονώνει τον εξωτερικό μετρητή 4-bit, προκειμένου ο επεξεργαστής να εφαρμόσει παλμούς για να μπορέσει να εξάγει την τιμή του εσωτερικού prescaler.

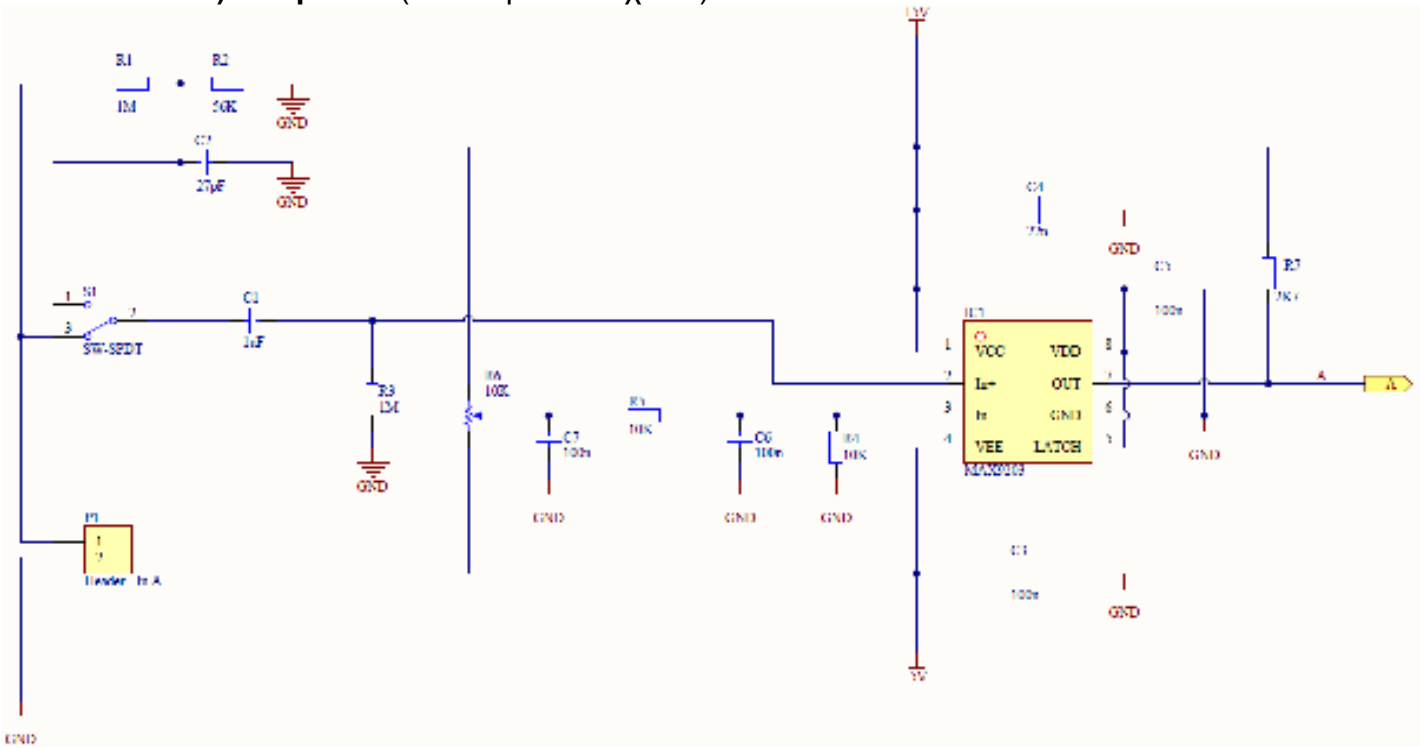
Η τελευταία βαθμίδα αποτελείται από έναν PIC18F242-I/SP ο οποίος εσωτερικά περιέχει έναν 16-bit μετρητή και έναν 8-bit προδιαιρέτη (prescaler). Μαζί με τον εξωτερικό 4-bit μετρητή εξασφαλίζονται τα 28-bit που απαιτούνται προκειμένου ο επεξεργαστής να μπορέσει να μετρήσει συχνότητες μέχρι 1.5GHz. Εδώ περιλαμβάνεται και ο κρύσταλλος του συχνομέτρου συχνότητας F=4MHz. Οι εντολές ελέγχου στον μικροεπεξεργαστή δίνονται μέσω του πληκτρολογίου που αποτελείται από 4 buttons.

Τέλος, η απεικόνιση της συχνότητας και κάποιων βοηθητικών μυνημάτων γίνεται με τη χρήση μιας οθόνης υγρών κρυστάλλων HD44780 της WINSTAR, η οποία τροφοδοτείται με τάση τροφοδοσίας ±5Volt και είναι τύπου 2X16. Δηλαδή, με δυνατότητα απεικόνισης 2 γραμμών και 16 χαρακτήρων ανά γραμμή.

2.3) Ανάλυση βαθμίδων κυκλώματος

Η εργασία μας αποτελείται από τις παρακάτω βαθμίδες τις οποίες και θα αναλύσουμε παρακάτω.

2.3.1) Βαθμίδα 1 (Κυκλωματικό σχέδιο)



Σχήμα 1: Είσοδος A (0.1 MHz -99 MHz)

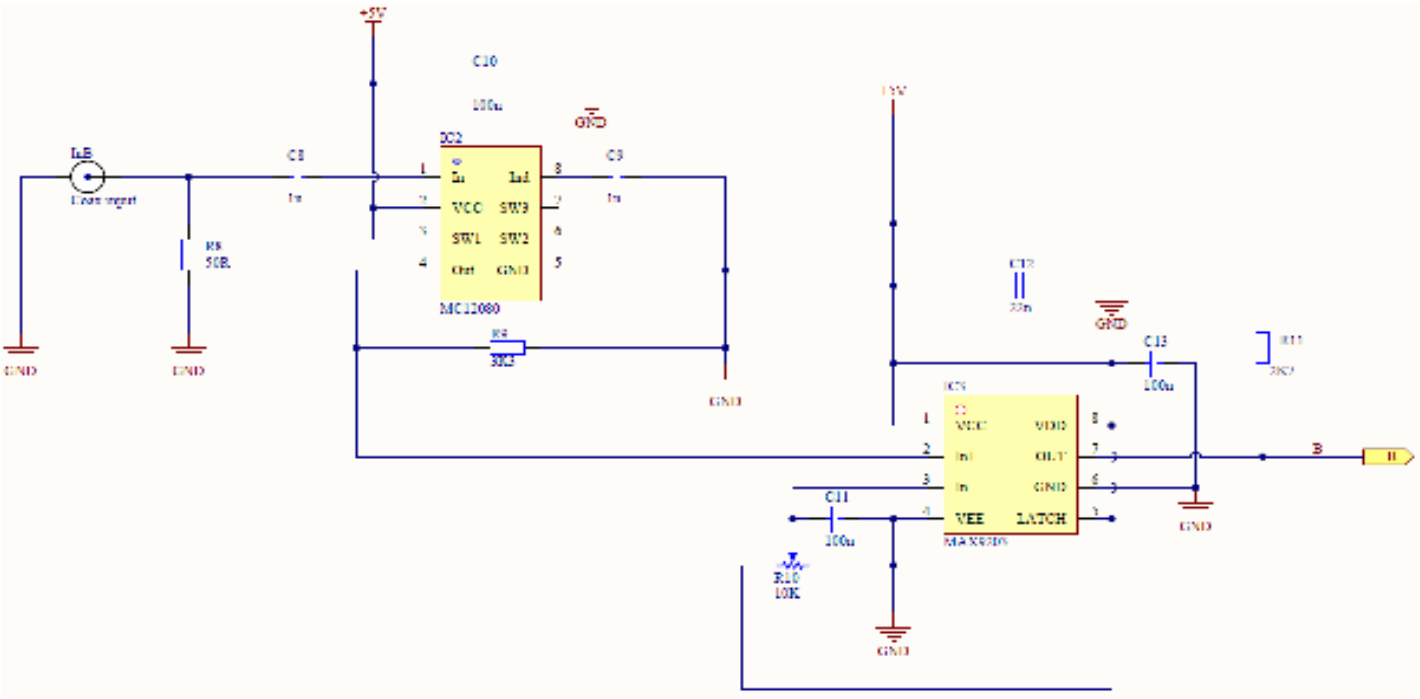
2.3.1.1) Ανάλυση βαθμίδας 1

Η πρώτη βαθμίδα, όπως βλέπουμε και από το σχήμα, αποτελείται από έναν διαιρέτη τάσης (R1,R2) ο οποίος λειτουργεί σαν εξασθενητής και είναι αναγκαίος στις περιπτώσεις όπου έφαρμόζεται πολύ ισχυρό σήμα στην είσοδο. Το μέγιστο πλάτος σήματος που μπορεί με ασφάλεια να εφαρμοστεί στην είσοδο CH1 είναι περίπου 10Vp-p χωρίς τη χρήση εξασθενητή και περίπου 200Vp-p με τη χρήση του 1:20 εξασθενητή. Το σήμα μας θα περάσει από τον εξασθενητή αν επιλεγεί από το χρήστη μέσω του διακόπτη S1 και όπως φαίνεται από τις τιμές των αντιστάσεων η τιμή του θα πέσει στο 1:20 της αρχικής.

Στη συνέχεια, ακολουθεί ένας πυκνωτής ο οποίος αποκόπτει την DC συνιστώσα και ρυθμίζει την στάθμη σύγκρισης στα 0 Volt. Όμως, επειδή ο συγκριτής έχει μια τιμή offset, η στάθμη σύγκρισης δεν είναι τα 0 Volt αλλά μεταβάλλεται λίγο πάνω από τα 0 Volt (την ακριβή τιμή του offset την βρίσκουμε πειραματικά). Με την βοήθεια ενός ποτενσιόμετρου ρυθμίζουμε την τιμή του offset σαν στάθμη σύγκρισης αλλά και την ευαισθησία. Έτσι μικραίνοντας με το ποτενσιόμετρο το σήμα, επιτυγχάνουμε μεγαλύτερη ευαισθησία. Η τιμή του offset μπορεί να μεταβληθεί λόγω τις θερμοκρασίας. Για την καλύτερη λειτουργία του κυκλώματος θα μπορούσε το ποτενσιόμετρο να είναι εξωτερικό και να ρυθμίζουμε κάθε φορά την στάθμη σύγκρισης.

Μετά το διαιρέτη τάσης για το offset και το φίλτρο του, ακολουθεί ο συγκριτής ο οποίος στην είσοδό του δέχεται $\pm 5V$ και στην έξοδό του βγάζει TTL στάθμες από 0.2V - 4.5V.

2.3.2) **Βαθμίδα 2** (Κυκλωματικό σχέδιο)



Σχήμα 2: Είσοδος Β (70 MHz-1.5 GHz)

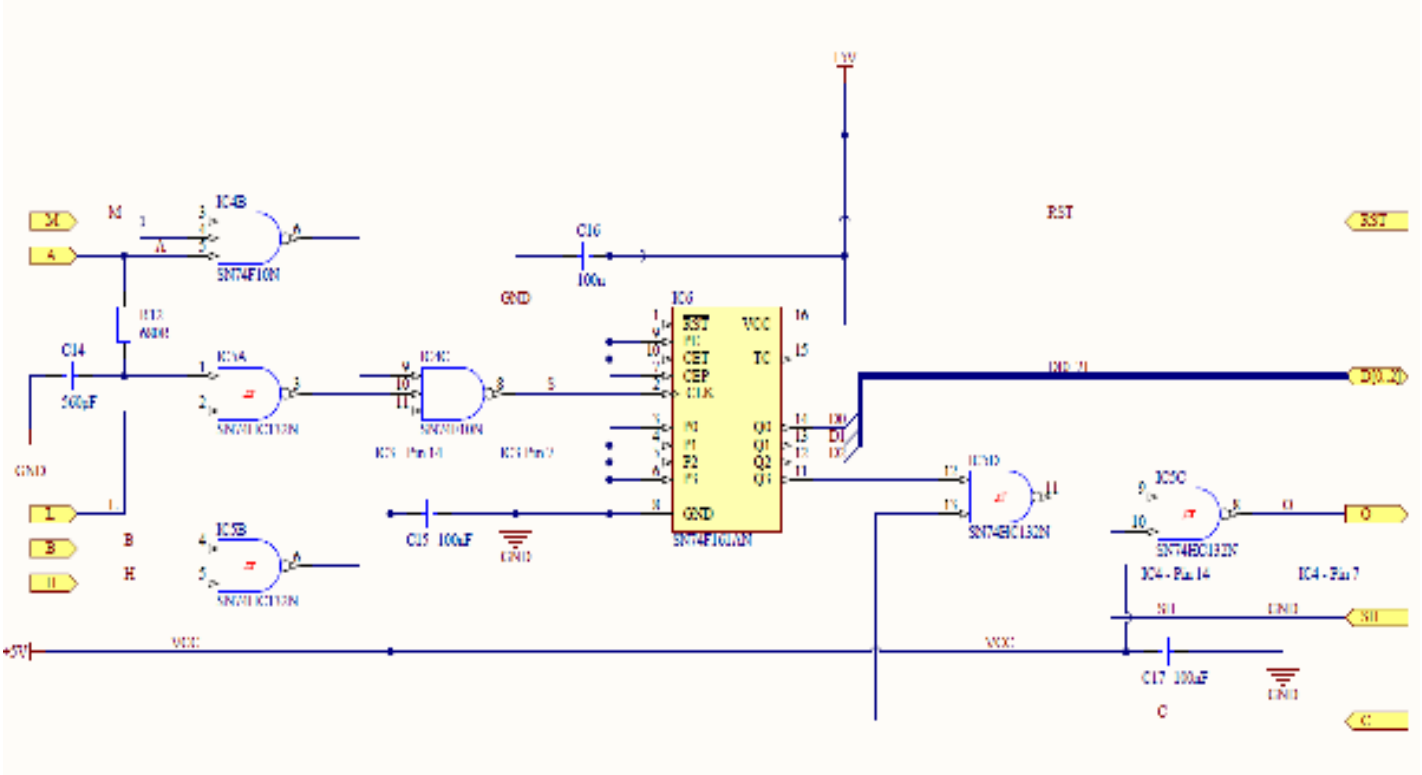
2.3.2.1) Ανάλυση Βαθμίδας 2

Αυτή η βαθμίδα αποτελεί την δεύτερη είσοδο του κυκλώματος η οποία χρησιμοποιείται για τις υψηλές συχνότητες. Στην είσοδο υπάρχει ένα φίλτρο το οποίο φιλτράρει τις χαμηλές συχνότητες και μια αντίσταση για την απόσβεση του θορύβου.

Στη συνέχεια, ακολουθεί ο prescaler στον οποίο έχει οριστεί λόγος διαίρεσης 1:40 όπως είδαμε από το φυλλάδιο πληροφοριών. Ο prescaler δεν βγάζει στάθμες TTL, αλλά ημίτονο με 1.2 Vp-p και οδηγεί ένα Schmitt trigger. Όμως για να σκανδαλίσουμε το Schmitt trigger χρειάζεται 1.5 Vp-p, οπότε τα 1.2 Vp-p δεν είναι αρκετά. Αυτό το πρόβλημα θα μπορούσαμε να το αντιμετωπίσουμε ενισχύοντας το σήμα με έναν τελεστικό. Όμως αυτό δεν είναι επιθυμητό καθώς το σήμα είναι ημιτονοειδές κι εδώ απαιτούνται στάθμες TTL. Έτσι το σήμα από την έξοδο του prescaler οδηγείται σε έναν συγκριτή. Ο συγκριτής μετατρέπει τα 1.2 Vp-p σε στάθμες TTL. Επιπλέον παράγεται ένα σήμα ON-OFF ψηφιακό, το οποίο είναι ιδανικό για τον σκανδαλισμό του Schmitt trigger. Όπως εξηγήθηκε και στην προηγούμενη παράγραφο έτσι κι εδώ υπάρχει ένα ποτενσιόμετρο για τη ρύθμιση της στάθμης σύγκρισης.

2.3.3) Βαθμίδα 3

Σχήμα 3: Λογική σχεδίαση



2.3.3.1) Ανάλυση Βαθμίδας 3

Σε αυτή τη βαθμίδα εμπεριέχονται τα εξής στοιχεία:

- **ΦΙΛΤΡΟ LPF**

Είναι ένα φίλτρο RC που αποτελείται από την αντίσταση R12=680Ω και τον πυκνωτή C14=560pF. Αυτό, το χρησιμοποιούμε διότι ο υψηλής ταχύτητας συγκριτής MAX9203 είναι προβληματικός στις χαμηλές συχνότητες και παρουσιάζει προβλήματα ταλάντωσης. Αυτά τα προβλήματα ταλάντωσης θα μπορούσαν να δημιουργήσουν εσφαλμένο σκανδαλισμό στο επόμενο στάδιο. Για καθαρή μεταγωγή της εξόδου χωρίς ταλαντώσεις, απαιτούνται 0.5V/sec στην έξοδο του MAX9203. Αλλά αυτό δεν μπορεί να διασφαλιστεί στις χαμηλές συχνότητες (0-1MHz). Γι' αυτό το λόγο χρειάζεται το LPF σε συνδυασμό με έναν Schmitt Trigger για να "καθαρίσει" οποιαδήποτε προβλήματα ταλάντωσης στις χαμηλές συχνότητες. Διαφορετικά μπορεί να συμβεί μια εσφαλμένη μέτρηση συχνότητας εξαιτίας οποιουδήποτε θορυβώδους σήματος σκανδαλισμού στον μετρητή.

- **ΠΟΛΥΠΛΕΚΤΗΣ (MUX 3-1)**

Ο πολυπλέκτης αυτός ενεργοποιεί τη μεταγωγή μεταξύ των καναλιών εισόδου Α, Β αλλά επίσης ενεργοποιεί και τη μεταγωγή του καθαρού από ταλαντώσεις σήματος Α', οποιαδήποτε χρονική στιγμή κριθεί απαραίτητο αυτό να γίνει (σε χαμηλές συχνότητες) εξασφαλίζοντας έτσι ακριβείς μετρήσεις σε χαμηλές συχνότητες. Με αυτό το τρόπο, στην έξοδο του πολυπλέκτη εμφανίζεται το επιθυμητό σήμα το οποίο και ρυθμίζεται μέσω software.

- **4-bit COUNTER**

Είναι ένας εξωτερικός μετρητής 4-bit, στον οποίο οδηγείται η έξοδος του πολυπλέκτη (MUX 3X1). Ο μετρητής αυτός χρησιμοποιείται προκειμένου να καλυφθούν τα 4 bit που υπολοίπονται στο hardware. Από την μέγιστη ταχύτητα του συγκεκριμένου μετρητή καθορίζεται και το ανώτατο όριο της συχνότητας.

- **ΠΟΛΥΠΛΕΚΤΗΣ (MUX 2x1)**

Είναι ένας πολυπλέκτης (MUX 2x1), στην είσοδο του οποίου οδηγείται η έξοδος του μετρητή 4-bit. Αποτελείται από τις πύλες IC5D & IC5C, και εξυπηρετεί στην ανάγνωση της τιμής του prescaler που υπάρχει στην επόμενη βαθμίδα. Ουσιαστικά, η λειτουργία του πολυπλέκτη αυτού, είναι είτε να δουλεύει κανονικά σαν μετρητής, είτε να εφαρμόζει ο επεξεργαστής παλμούς προκειμένου να εξάγει την τιμή του εσωτερικού prescaler 8-bit.

2.3.3.2) Ανάλυση της απεικόνισης μέσω λογικής σχεδίασης

Σε αυτή τη βαθμίδα εκτελείται η ακόλουθη διαδικασία μέτρησης παλμών προκειμένου το συχνόμετρο να απεικονίσει την τελική συχνότητα που έχει μετρηθεί.

- **Μηδενισμός των μετρητών**

Ξεκινώντας, γίνεται μηδενισμός των μετρητών που διαθέτονται για την μέτρηση της συχνότητας μέσω του επεξεργαστή. Ο εξωτερικός μετρητής 4-bit μηδενίζεται μέσω του RST pin, ενώ ο 8-bit και ο 16-bit μηδενίζονται μέσω του software.

- **Έναρξη της διαδικασίας της μέτρησης**

Ο επεξεργαστής ξεκινάει να μετράει, χρησιμοποιώντας τους counters, για όσο έχει επιλεχθεί η βάση χρόνου. Δηλαδή για 0.1sec , 1sec ή 10 sec.

- **Τερματισμός της λειτουργείας των μετρητών**

Για να μπορέσει ο επεξεργαστής να σταματήσει τους μετρητές θα πρέπει να σταματήσει τον εξωτερικό μετρητή 4-bit. Λόγω του ότι βρίσκεται σε σειρά με τους υπόλοιπους, ο τερματισμός του συγκεκριμένου μετρητή θα σημάνει τον τερματισμό και των υπόλοιπων μετρητών. Αυτό επιτυγχάνεται ως εξής:
Μετατρέποντας σε λογική κατάσταση “0” τα σήματα εισόδου L,M,H επιτυγχάνουμε λογική κατάσταση “1” στις εξόδους των πυλών IC4B,IC5A & IC5B. Με αυτό το τρόπο η έξοδος της εσωτερικής NAND (IC4C) γίνεται “0” και ο μετρητής 4-bit σταματά τη λειτουργεία του αφού δεν περνάει σήμα από τη πηγή στο μετρητή.

- **Ανάγνωση των μετρητών**

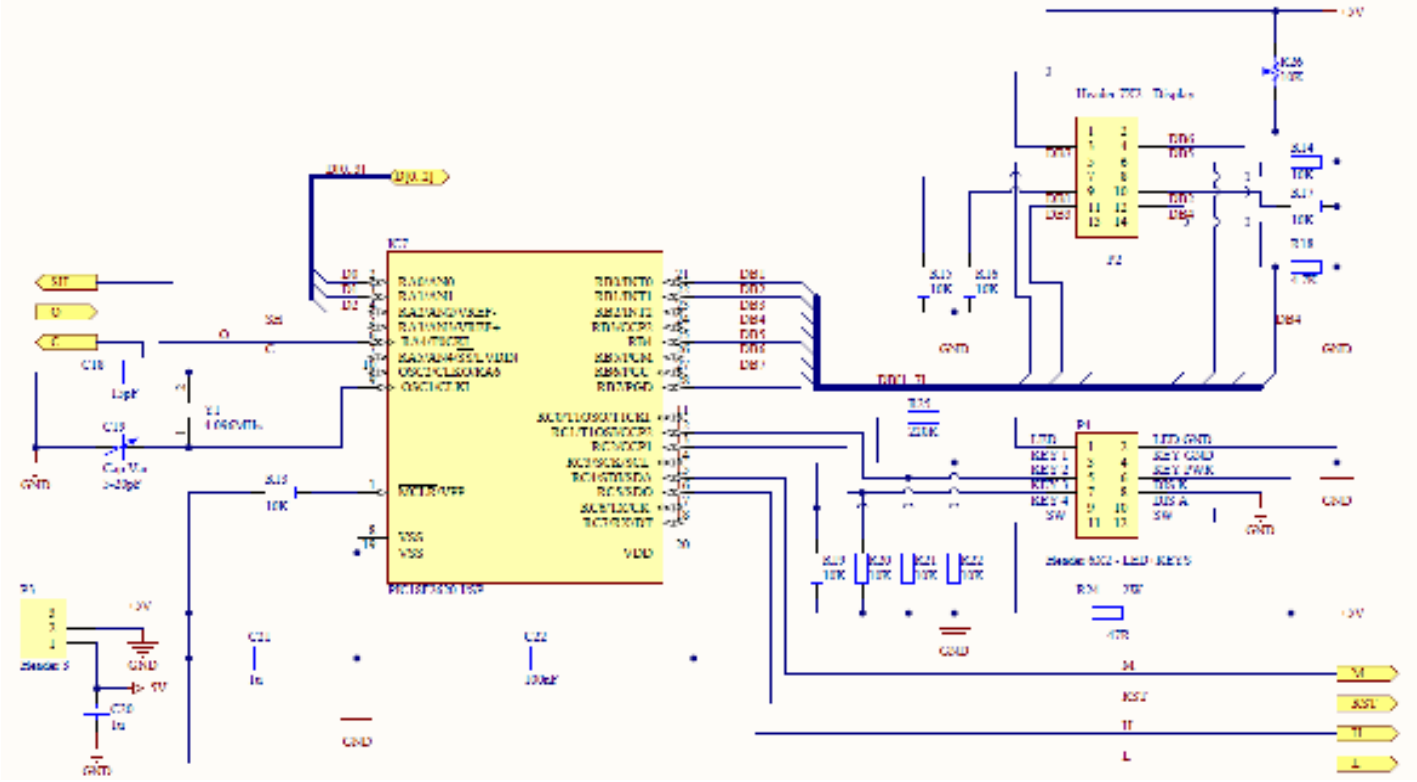
Αφού σταματήσουν να μετρούν οι μετρητές ξεκινά από τον επεξεργαστή η διαδικασία ανάγνωσής τους.
Ο εξωτερικός μετρητής 4-bit μπορεί να “διαβαστεί” κατευθείαν από τον επεξεργαστή μέσω των pins που τον συνδέουν με αυτόν. Επίσης, άμεσα γίνεται και η αναγνώριση και ο έλεγχος του εσωτερικού μετρητή 16-bit του μικροεπεξεργαστή.

Όμως, λόγω του ότι η ανάγνωση του 8-bit prescaler δεν μπορεί να γίνει απευθείας χρησιμοποιούμε την εξής τεχνική μέσω του software. Όταν τα σήματα εισόδου C και SH του πολυπλέκτη (MUX) βρίσκονται σε λογική κατάσταση “1” και “1” αντίστοιχα, το MSB του μετρητή 4-bit περνάει στην έξοδο του πολυπλέκτη. Προκειμένου όμως να “διαβάσουμε” την τιμή του prescaler θα πρέπει να μη περάσει το MSB στην έξοδο αλλά να εφαρμόσουμε παλμούς. Αυτό το επιτυγχάνουμε θέτοντας σε λογική κατάσταση “0” την είσοδο C του πολυπλέκτη, οπότε η έξοδος της πύλης IC5D θα βρίσκεται πάντα σε κατάσταση “1”. Με αυτό το τρόπο απομονώνεται ο μετρητής και οτιδήποτε σήμα έρθει εδώ, βγαίνει στην έξοδο αντεστραμμένο. Στη συνέχεια ο επεξεργαστής εφαρμόζει παλμούς μέσω του pin SH της πύλης IC5C και μετράει τον αριθμό των παλμών που χρειάστηκε να εφαρμόσει για να υπερχειλήσει ο 8-bit μετρητής. Η απομόνωση του μετρητή είναι απαραίτητη καθώς δε γνωρίζουμε αν το MSB του βρίσκεται σε κατάσταση “0” ή “1”, οπότε και τίθεται σε κατάσταση “1”.

- **Απεικόνιση της συχνότητας**

Όταν ολοκληρωθεί η διαδικασία ανάγνωσης όλων των τιμών των μετρητών, ο επεξεργαστής καλεί και χρησιμοποιεί τη σχέση **(3)**, μέσω της οποίας προκύπτει η τελική τιμή της μετρηθείσας συχνότητας, και στο τέλος πραγματοποιεί την απεικόνισή της στο LCD.

2.3.4) Βαθμίδα 4



Σχήμα 4: Επεξεργαστής PIC18F242-I/SP

2.3.4.1) Ανάλυση Βαθμίδας 4

Σε αυτή τη βαθμίδα εμπεριέχονται τα εξής στοιχεία:

- **ΕΠΕΞΕΡΓΑΣΤΗΣ PIC**

Είναι ένας επεξεργαστής PIC18F242-I/SP, ο οποίος περιέχει τον 16-bit μετρητή και τον προδιαιρέτη, ο οποίος χρησιμοποιείται σαν ένας 8 bit-μετρητής. Έτσι, μαζί με τον εξωτερικό 4-bit μετρητή από την προηγούμενη βαθμίδα, εξασφαλίζεται ο 28 bit counter που απαιτείται.

- **ΚΡΥΣΤΑΛΛΟΣ**

Ένας κρύσταλλος συχνότητας F=4MHz, ο οποίος αποτελεί και τη βάση χρόνου μας. Αυτός συνδέεται στους ακροδέκτες εισόδου OSC1 και OSC2 του μικροεπεξεργαστή. Οι ακροδέκτες OSC1 και OSC2 αποτελούν την είσοδο του εσωτερικού ταλαντωτή ρολογιού (clock-oscillator) που περιέχει ο μικροεπεξεργαστής. Στο κύκλωμα μας χρησιμοποιούμε έναν μεταβλητό πυκνωτή C19(5-20pF), ο οποίος είναι απαραίτητος για την βαθμονόμηση του οργάνου.

Θεωρητικά, η μικρότερη καθυστέρηση που μπορεί να γίνει είναι ένας παλμός ρολογιού. Επειδή όμως οι επεξεργαστές PIC εκτελούν μια εντολή κάθε 4 παλμούς ρολογιού η μικρότερη καθυστέρηση που μπορεί να επιτευχθεί στη προκειμένη περίπτωση είναι 4T, με $T=\frac{1}{F}$ & F=4 MHz.

Όπου T η περίοδος και F η συχνότητα του κρυστάλλου.

Το γινόμενο αυτό που ισοδυναμεί με 4T ονομάζεται TCY και είναι στην ουσία ο χρόνος για την εκτέλεση μιας εντολής. Άρα σε αυτή τη περιπτωσή το TCY θα είναι ίσο με 4*0,25μsec, οπότε TCY=1μsec. Οπότε, η μικρότερη καθυστέρηση που μπορεί να επιτευχθεί στο software είναι της τάξεως του 1μsec. Για αυτό το λόγο και οι εντολές που βρίσκονται στο software είναι όλες πολλαπλάσιες του TCY. Από τα παραπάνω συμπεραίνεται πως όσο πιο μεγάλη είναι η συχνότητα του κρυστάλλου τόσο καλύτερη ακρίβεια θα έχουμε.
Εμείς έχουμε τρεις βάσεις χρόνου: 0.1s, 1s και 10s

Το 0.1s ισούται με : 10^5 TCY

Το 1s ισούται με : 10^6 TCY

Τα 10 s ισούνται με : 10^7 TCY

- **ΠΛΗΚΤΡΟΛΟΓΙΟ ΕΛΕΓΧΟΥ**

Το πληκτρολόγιο ελέγχου της συσκευής αποτελούμενο από 4 buttons. Μέσω των πλήκτρων ο χρήστης μπορεί και δίνει τις παρακάτω εντολές στον μικροεπεξεργαστή :

- Εντολή για επιλογή καναλιού μεταξύ CH1 και CH2.
- Εντολή για μέτρηση συχνότητας με ακρίβεια (0,1Hz) –fast mode
- Εντολή για μέτρηση συχνότητας με ακρίβεια (1Hz) – normal mode
- Εντολή για μέτρηση συχνότητας με ακρίβεια (10Hz) – slow mode
- Εντολή για μέτρηση συχνότητας πραγματικού χρόνου –real time
- Εντολή για μέτρηση συχνότητας με συγκράτηση κορυφής –max hold
- Εντολή για μέτρηση συχνότητας με συγκράτηση ελαχίστου –min hold
- Εντολή για απεικόνιση της μετρούμενης συχνότητας σε Hz, KHz ή MHz
- Εντολή για απεικόνιση της μετρούμενης συχνότητας σε Hz, KHz ή MHz

Τα pins του μικροεπεξεργαστή που συνδέονται με τα buttons γειώνονται διαμέσου $4^{ων}$ αντιστάσεων (pull-up resistors). Οι αντιστάσεις είναι απαραίτητες για να κρατάνε την κατάσταση εισόδου των pins σε λογικό “0” όταν τα button βρίσκονται σε κατάσταση off. Ο μικροεπεξεργαστή, «αντιλαμβάνεται» το πάτημα ενός button σε πραγματικό χρόνο ελέγχοντας συνεχώς την λογική κατάσταση των io pins 11 (RC0), 12 (RC1), 13 (RC2), 14 (RC3). Τη στιγμή που ανιχνευτεί το πάτημα κάποιου button, ο επεξεργαστής εκτελεί την εντολή που αντιστοιχεί στο συγκεκριμένο button.

ΜΕΡΟΣ ΤΡΙΤΟ

ΠΕΡΙΓΡΑΦΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ ΚΑΙ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ ΛΕΙΤΟΥΡΓΙΑΣ

3.1) Αλγόριθμος λειτουργίας

Ο αλγόριθμος λειτουργίας που εκτελεί ο επεξεργαστής παρουσιάζεται αμέσως παρακάτω:

1. Μετάβαση στο κανάλι Α ή Β σύμφωνα με τη τρέχουσα κατάσταση.

2. Εάν είναι επιλεγμένο το κανάλι Α, τότε επιλέγει ο επεξεργαστής αν θα χρησιμοποιήσει ή όχι το LPF *.

3. Μηδενισμός μετρητών

4. Έναρξη μετρητών

5. Αναμονή για χρονικό διάστημα Δt=0.1, 1 ή 10sec (σύμφωνα με την τρέχουσα κατάσταση). Κατά τη διάρκεια της αναμονής γίνεται περιοδικός έλεγχος του πληκτρολογίου. Αν πατηθεί οποιοδήποτε κουμπί, τότε εκτελούνται οι κατάλληλες διαδικασίες, γίνεται ενημέρωση της τρέχουσας κατάστασης και επιστροφή στο βήμα 1, διαφορετικά συνεχίζει.

6. Διακοπή των μετρητών

7. Ανάγνωση της τιμής Ν των μετρητών (ακέραιος 28-bit) και υπολογισμός συχνότητας μέσω του τύπου F=N/Δt **.

8. Απεικόνιση της τρέχουσας συχνότητας (real time mode) ή της μέγιστης συχνότητας που έχει μετρηθεί (Max hold mode) ή της ελάχιστης συχνότητας που έχει μετρηθεί (Min hold mode) σε μονάδες Hz, KHz ή MHz σύμφωνα με την τρέχουσα κατάσταση.

9. Αλλαγή κατάστασης του LED.

10. Επιστροφή στο βήμα 1.

* Η απόφαση σχετικά με το αν θα χρησιμοποιηθεί το LPF γίνεται ως εξής:
Ο επεξεργαστής πραγματοποιεί μια γρήγορη μέτρηση συχνότητας χωρίς το LPF. Αν η μετρηθήσα συχνότητα βρεθεί κάτω των 980KHz τότε ο επεξεργαστής ενεργοποιεί το LPF για μετρήσεις ακριβείας.

** Ο επεξεργαστής διαβάζει την τιμή (Α) του εξωτερικού μετρητή 4-bit, την τιμή (Β) του εσωτερικού prescaler 8-bit, και την τιμή του εσωτερικού μετρητή 16-bit του Timer0 (C) και υπολογίζει την τιμή (Ν) του μετρητή 28-bit από τη σχέση: N = A + B * 16 + C * 4096. Εάν το επιλεγμένο κανάλι εισόδου είναι το CH1 τότε η συχνότητα που έχει μετρηθεί υπολογίζεται από τον τύπο F=N/Δt. Εάν όμως, το επιλεγμένο κανάλι εισόδου είναι το CH2 τότε η συχνότητα που έχει μετρηθεί υπολογίζεται από τον τύπο F=N*40/Δt (λαμβάνοντας υπόψιν τον προδιαιρέτη MC12080 ο οποίος ρυθμίζεται με λόγο διαίρεσης 1:40).

3.2) Software του συχνομέτρου

```
/*****************************************************************************/
/*                          * Frecuency-Counter code *                       */
/*                         By George A. Adamidis                             */
/*                         Ver 1.0 - March 17, 2009                          */
/*****************************************************************************/
// All rights are reserved


// Compiled using MPLAB-C18 V 8.10
// Loaded on a PIC18F242 microcontroler
// All timing routines have been written for 4MHz master clock.
//---------------------------------------------------------------------------

#include "Common.h"        /* General Definitions          */
#include <delays.h>        /* Delay purpose routines       */
#include <timers.h>        /* timer purpose routines       */
#include <xlcd.h>          /* LCD purpose routines         */
#include <p18cxxx.h>       /* TRISB and PORTB declarations */
#include <p18f242.h>       /* for pic18f242                */

/* Set configuration bits */
#pragma config OSC = XT   // set XS oscillator
#pragma config WDT = OFF  // disable watchdog timer
#pragma config LVP = OFF  // disable low voltage programming


/* Device turned off, if VDD<4,5V */
```

```
#pragma config BOR = ON
#pragma config BORV = 45


/*Code protection*/
#pragma config CP0 = ON
#pragma config CP1 = ON


/*functions declaration*/
void timer_isr (void);                          //Interrupt routine, take place at TMR0's counter overflow
void freq (void);                               // Finds current frequency, Max Hold frequency, Min Hold frequency  in dHz (values are
stored in "frequency", "maxhold" & "minhold" variables.
void displayfreq (unsigned long freq);          // Displays frequency freq. - Affects digit[] array
void delay (char gate);                                         // Delay for 0.1, 1 or 10 sec for gate =0,1 or 2, respectively
extern void disst(void);                                // Displays current device state at LCD's second line,

extern char low_or_medium (void);         // Returns L or M if a low or medium frequency measurement must take place, respectively
extern void pause_counter(char mode);   // Pause the counter through L,M or H-PIN - according to mode (mode=L,M or H)
extern void start_counting (char mode); // Starts Counting through L,M or H-PIN - according to mode (mode=L,M or H)
extern void resetcounter (void);        // Resets the counter  (works only when counter is in pause mode)
extern char buttonpressed (void);
extern void action(unsigned char but);  // Perfoms the appropriate function if button "but" has been pressed (but - 1,2,3 or 4)
extern int prescaler(void);
extern int external_counter (void);

// variables - declaration
unsigned long Overflow=0;      // Overflow = how many "overflows" took place on TMR0 counter during counting
unsigned char button=0;          // button-->0,1,2 or 3 if no button, mode button, function button or units button has been pressed
during counting, respectively
```

```
char mode=1;                    // mode=1 --> Normal counting mode (counts for 1sec), mode=0--> fast counting mode (counts for
0.1sec),mode=2--> Slow counting mode (counts for 10sec)
char function=0;                // function=0 --> Realtime measuring, function=1 --> Max Hold measuring, function=2 --> Min Hold
measuring
char units=0;               // units=0,1,2 --> Dsplays frequency in Hz, KHz, MHz units respectively
char channel=1;                         // channel=1 or 2 --> for channel A and channel B respectively
unsigned long frequency=0;                      // Current frequency - in dHz
unsigned long maxhold=0;                        // Current Max Hold frequency
unsigned long minhold=19999999999;              // Current Min Hold frequency in dHz
char digit[12]="          ";                    // Frequency digits - display
char led=0;                             // Current Led state (1 =on, 0=off)

// High priority interrupt vector
#pragma code InterruptVectorHigh = 0x08
void InterruptVectorHigh (void)
{
  _asm
  goto timer_isr        //jump to interrupt routine
  _endasm
}

// High priority interrupt routine
#pragma code
#pragma interrupt timer_isr
/*Interrupt routine, take place at TMR0's counter overflow*/
void timer_isr ()
{
        INTCONbits.TMR0IF = 0;      // Interrupt done - Interrupt bits set to 0
      Overflow++;             //
}
```

```
/****************** Main body starts here *************************/
void main ()
{
        char meas;

        Delay10KTCYx(20);                // wait 0.2s for peripherals to be initialized

        TRISB=0;                        // configure portB for output
        TRISC=15;                        // configure portC for input (RC0-RC3) and output (RC4-RC7)
        TRISA=23;                        // configure portA for input (RA0-RA2), (RA4) and output (RA3,RA5)
        ADCON1=6;                          // Set ADCON1 register to 6 - ADCON1 must be 6 or 7 to enable PORTA bits as digital I/O.

        //Initializes LCD I/O pins used to control HD44780 LCD controller in 4bit interface and initializes LCD as a 5x7 multiple line.
        while( BusyXLCD() );              // waits until LCD not busy
        OpenXLCD(FOUR_BIT & LINES_5X7 );

        while( BusyXLCD() );              // waits until LCD not busy
        WriteCmdXLCD( CURSOR_OFF );         // Cursor off on LCD

        while( BusyXLCD() );              // waits until LCD not busy
        SetDDRamAddr(0x00);                 // Sets LCDs cursor on first position on first line
        putrsXLCD("Frequency  Meter");      // Displays " FREQUENCY  METER" on LCD's first line

        while( BusyXLCD() );              // waits until LCD not busy
        SetDDRamAddr(0x40);                 // Sets LCDs cursor on first position on second line
        putrsXLCD("Chania.TEI-Crete");      // Displays "Chania.TEI-Crete" on LCD's second line
        LED=off;                              // Turns off Led
        led=off;
```

```
Delay10KTCYx(200);              // wait 2s
Delay10KTCYx(100);              // wait 1s
disst();
displayfreq(0);

// infinitive loop starts here
label:SH_PIN=on;                 // Sets the counter strobe control pin at high logic level
pause_counter  ('L');
pause_counter  ('M');
pause_counter  ('H');
Delay1KTCYx(5);                                         // wait 5ms
resetcounter ();
Delay1KTCYx(5);                                         // wait 5ms
button=0;                                               // resets button variable
Overflow=0;                  // resets overflow variable

switch (channel)

        {

                case 1:
                        meas=low_or_medium();
                break;

                case 2:
                        meas='H';
                break;
        }
```

```
  switch (mode)                                 // mode=0,1,2--> fast, normal, or slow counting mode respectively
 {
  case 0:                       // Fast counting mode has been selected (counts for 0.1sec)
                        start_counting (meas);      // Starts the counter
                        delay (0);                                      // wait 0.1 sec
                        pause_counter(meas);        // Stops counting
        break;

        case 1:                                     // Normal counting mode has been selected (counts for 1sec)
                        start_counting (meas);      // Starts the counter
                        delay (1);                                      // wait 1 sec
                        pause_counter(meas);        // Stops counting
                break;

                case 2:                              // Slow counting mode has been selected (counts for 10sec)
                        start_counting (meas);      // Starts the counter
                        delay (2);                                      // wait 10 sec
                        pause_counter(meas);        // Stops counting
                break;

}

  Delay1KTCYx(1);                                       // wait 1ms
  if (button>0) goto label;

  freq();                                               // Finds current frequency

  switch (function)
          {
                case 0:                                         // Displays Frequency
```

```
                displayfreq(frequency);
            break;

            case 1:                                 // Displays Max Hold frequency
                displayfreq(maxhold);
            break;

            case 2:                                 // Displays Min Hold frequency
                displayfreq(minhold);
            break;
        }

        switch (led)
            {
            case 0:
                LED=on;             // Turns on Led
                led=on;
            break;

            case 1:
                LED=off;            // Turns off Led
                led=off;
            break;
            }

        goto label;                                 // infinitive loop ends here
}

/****************** Main body Ends here ***********************************************************************/
```

```
 // Finds current frequency, Max Hold frequency, Min Hold frequency  in dHz or Hz (values are stored in "frequency", "maxhold" &
"minhold" variables.
void freq (void)
{
unsigned long buffer=0;
unsigned long buf=0;
unsigned long counter=0;
unsigned long counter2=0;
counter=ReadTimer0();                                                    // Reads current TMR0's (counter) value
buf=prescaler();                                                         // Reads the internal prescaler
counter2=external_counter();
buffer=(counter2)+(16*buf)+(4096*counter);          // Finds Counter's value

if (channel==1)
        {

                switch (mode)                                                                    // Finds Frequency in dHz
                        {
                                case 0:
                                        buffer=buffer*100;
                                break;

                                case 1:
                                        buffer=buffer*10;
                                break;

                        }
```

```
        }


if (channel==2)
        {
                switch (mode)                                                   // Finds Frequency in Hz
                        {
                                case 0:
                                        buffer=buffer*10;
                                break;

                                case 2:
                                        buffer=buffer/10;
                                break;

                        }

                buffer=buffer*PRESC;                                    // Taking into acount the external prescaler for channel 2

        }

frequency=buffer;                                                       // Finds frequency
if (buffer<minhold) minhold=buffer;             // Finds Min Hold frequency
if (minhold==0) minhold=buffer;                                 // Finds Min Hold frequency
if (buffer>maxhold)maxhold=buffer;                      // Finds Max Hold frequency

}
```

```
// Displays frequency freq. - Affects digit[] array
void displayfreq (unsigned long freq)
{
unsigned long d[12]="           ";
char *dig;
char i;
char j;
unsigned long fr;

fr=freq;

        // Convert to freq to ASCII

                        d[11]=48+(freq%10);
                        freq=freq/10;
                        d[10]=48+(freq%10);
                        freq=freq/10;
                        d[9]=48+(freq%10);
                        freq=freq/10;
                        d[8]=48+(freq%10);
                        freq=freq/10;
                        d[7]=48+(freq%10);
                        freq=freq/10;
                        d[6]=48+(freq%10);
                        freq=freq/10;
                        d[5]=48+(freq%10);
                        freq=freq/10;
                        d[4]=48+(freq%10);
                        freq=freq/10;
```

```
                        d[3]=48+(freq%10);
                        freq=freq/10;
                        d[2]=48+(freq%10);
//                      freq=freq/10;
//                      d[1]=48+(freq%10);
//                      freq=freq/10;
//                      d[0]=48+(freq%10);


if (channel==1)
        {

        switch (units)                                  // units=1,2,3 --> frequency in Hz, KHz, MHz units respectively

                {

                        case 0:                 // frequency is going to be displayed in Hz units
                                digit[0]=d[3];
                                digit[1]=d[4];
                                digit[2]=d[5];
                                digit[3]=d[6];
                                digit[4]=d[7];
                                digit[5]=d[8];
                                digit[6]=d[9];
                                digit[7]=d[10];
                                digit[8]='.';
                                digit[9]=d[11];
                                digit[10]=' ';
                                digit[11]=0;
```

```
            break;

            case 1:                  // frequency is going to be displayed in KHz units
                    digit[0]=d[3];
                    digit[1]=d[4];
                    digit[2]=d[5];
                    digit[3]=d[6];
                    digit[4]=d[7];
                    digit[5]='.';
                    digit[6]=d[8];
                    digit[7]=d[9];
                    digit[8]=d[10];
                    digit[9]=d[11];
                    digit[10]=' ';
                    digit[11]=0;


break;

            case 2:                  // frequency is going to be displayed in MHz units
                    digit[0]=d[3];
                    digit[1]=d[4];
                    digit[2]='.';
                    digit[3]=d[5];
                    digit[4]=d[6];
                    digit[5]=d[7];
                    digit[6]=d[8];
```

```
                                digit[7]=d[9];
                                digit[8]=d[10];
                                digit[9]=d[11];
                                digit[10]=' ';
                                digit[11]=0;



                        break;
                }

                if (fr>999999999)Overflow++;       //Display overflow for frequency>99,9999999 MHz
        }

if (channel==2)
        {

        switch (units)                                          // units=1,2,3 --> frequency in Hz, KHz, MHz units respectively

                {

                case 0:                 // frequency is going to be displayed in Hz units
                                digit[0]=d[2];
                                digit[1]=d[3];
                                digit[2]=d[4];
                                digit[3]=d[5];
                                digit[4]=d[6];
                                digit[5]=d[7];
                                digit[6]=d[8];
                                digit[7]=d[9];
```

```
                digit[8]=d[10];
                digit[9]=d[11];
                digit[10]=' ';
                digit[11]=0;

        break;

    case 1:             // frequency is going to be displayed in KHz units
                digit[0]=d[2];
                digit[1]=d[3];
                digit[2]=d[4];
                digit[3]=d[5];
                digit[4]=d[6];
                digit[5]=d[7];
                digit[6]=d[8];
                digit[7]='.';
                digit[8]=d[9];
                digit[9]=d[10];
                digit[10]=d[11];
                digit[11]=0;

break;

    case 2:             // frequency is going to be displayed in MHz units
                digit[0]=d[2];
                digit[1]=d[3];
                digit[2]=d[4];
                digit[3]=d[5];
                digit[4]='.';
                digit[5]=d[6];
```

```
                        digit[6]=d[7];
                        digit[7]=d[8];
                        digit[8]=d[9];
                        digit[9]=d[10];
                        digit[10]=d[11];
                        digit[11]=0;
                break;
        }

        if (fr>1600000000)Overflow++;           //Display overflow for frequency>1.6 GHz

        if (fr<1000)                                    // Bug fix for
                {
                        digit[0]=' ';
                        digit[1]=' ';
                        digit[2]=' ';
                        digit[3]=' ';
                        digit[4]=' ';
                        digit[5]=' ';
                        digit[6]=' ';
                        digit[7]=' ';
                        digit[8]=' ';
                        digit[9]=' ';
                        digit[10]='0';
                        digit[11]=0;

                }
}
```

```
if (Overflow>0)
	{
		digit[0]=' ';
		digit[1]=' ';
		digit[2]='O';
		digit[3]='V';
		digit[4]='E';
		digit[5]='R';
		digit[6]='F';
		digit[7]='L';
		digit[8]='O';
		digit[9]='W';
		digit[10]=' ';
		digit[11]=0;
	}


// Remove zeroes before dot
i=0;
j=digit[i];

while ((j==48)&(i<12))
	{
		digit[i]=' ';
		i++;
		j=digit[i];
	}

if (j=='.') digit[i-1]='0';
```

```
dig=&digit[0];

while( BusyXLCD() );              // waits until LCD not busy
SetDDRamAddr(0x00);               // Sets LCDs cursor on first position on first line

if (channel==1)
        {
                while( BusyXLCD() );
                putrsXLCD("A ");
        }

if (channel==2)
        {
        while( BusyXLCD() );
                putrsXLCD("B ");
        }

        while( BusyXLCD() );
        putsXLCD(dig);


if (Overflow==0)
                {
                        switch (units)
                                {
                                        case 0:
                                                while( BusyXLCD() );
                                                putrsXLCD(" Hz");
                                        break;
```

```
                                        case 1:
                                                while( BusyXLCD() );
                                                putrsXLCD("KHz");
                                        break;

                                        case 2:
                                                while( BusyXLCD() );
                                                putrsXLCD("MHz");
                                        break;

                                }

                }

if (Overflow>0)
        {
        while( BusyXLCD() );
                putrsXLCD("    ");
        }

}

/*****************************************************************/
// Delay for 0.1, 1 or 10 sec for gate =0,1 or 2, respectively
void delay (char gate)
{
        char i;

        switch (gate)
```

```
{
        case 0:                                                 // 0.1s delay
                Delay10KTCYx(9);        // wait 0.09s  for the counter to count
                Delay1KTCYx(9);                 // wait 0.009s for the counter to count
                Delay100TCYx(9);            // wait 0.0009s for the counter to count
                Delay10TCYx(2);         // wait 0.00002s for the counter to count
//              Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count
                Nop();                          // wait 1µs for the counter to count

                button=buttonpressed();             // Checks if any button is being pressed
                if (button!=0) action(button);     // If any button has been pressed, performs the appropriate action

        break;

        case 1:                                                 // 1s delay

                for (i=1; i<4; i++)
                        {
                                Delay10KTCYx(25);               // wait 0.25s for the counter to count
                                button=buttonpressed();         // Checks if any button is being pressed
                                        if (button!=0)
                                                {
```

```
                                        action(button);         // If any button has been pressed,
performs the appropriate action
                                        return;
                                    }
                }

                Delay10KTCYx(24);               // wait 0.24s for the counter to count
                button=buttonpressed();         // Checks if any button is being pressed
                        if (button!=0)
                                {
                                        action(button);         // If any button has been pressed, performs the
appropriate action
                                        return;
                                }

                Delay1KTCYx(9);                                 // wait 0.009s for the counter to count
                Delay100TCYx(8);                // wait 0.0008s for the counter to count
                Delay10TCYx(8);                 // wait 0.00008s for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count
```

```
                Nop();                  // wait 1µs for the counter to count
                Nop();                  // wait 1µs for the counter to count



        break;

        case 2:                                 // 10s delay

                for (i=1; i<40; i++)
                        {
                                Delay10KTCYx(25);               // wait 0.25s for the counter to count
                                button=buttonpressed();         // Checks if any button is being pressed
                                        if (button!=0)
                                                {
                                                        action(button);     // If any button has been pressed, performs
the appropriate action
                                                        return;
                                                }

                        }

                Delay10KTCYx(24);               // wait 0.24s for the counter to count
                button=buttonpressed();         // Checks if any button is being pressed
                        if (button!=0)
                                        {
                                                action(button);     // If any button has been pressed, performs the
appropriate action
                                                return;
```

```
                                        }
                                Delay1KTCYx(8);                         // wait 0.009s for the counter to count
                                Delay100TCYx(11);                       // wait 0.0009s for the counter to count
                                Delay10TCYx(28);                        // wait 0.00009s for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
                                Nop();                  // wait 1µs for the counter to count
//                              Nop();                  // wait 1µs for the counter to count
//                              Nop();                  // wait 1µs for the counter to count
//                              Nop();                  // wait 1µs for the counter to count
//                              Nop();                  // wait 1µs for the counter to count

                        break;

                }
}
```

```
//-------------------------------------------------------------------------

#include "Common.h"        /* General Definitions          */
#include <timers.h>        /* timer purpose routines       */
#include <p18f242.h>       /* pic18f242                    */
#include <p18cxxx.h>       /* TRISB and PORTB declarations */
#include <delays.h>        /* Delay purpose routines       */

/*functions declaration*/
void pause_counter(char mode);                  // Pause the counter through L,M or H-PIN - according to mode (mode=L,M
or H)
void start_counting (char mode);                // Starts Counting through L,M or H-PIN - according to mode (mode=L,M or
H)
void resetcounter (void);                       // Resets the counter  (works only when counter is in pause mode)
void counterup (void);                          // Counter increment by 1 (works only when counter is in pause mode)
int prescaler(void);                    // Shifts out & returns, prescaler's internal counter value
char low_or_medium (void);                      // Returns L or M if a low or medium frequency measurement must
take place, respectively
int external_counter (void);            // Returns the value of the external counter (return value is from 0 to 15)
extern char buttonpressed (void);

// variables - declaration
extern volatile unsigned long Overflow;
extern char mode;
```

```
/*************************************************************************/
// Pause the counter through L,M or H-PIN - according to mode (mode=L,M or H)
void pause_counter  (char mode)
{
        switch (mode)
                        {
                                case 'L':
                                        L_PIN=off;
                                break;

                                case 'M':
                                        M_PIN=off;
                                break;

                                case 'H':
                                        H_PIN=off;
                                break;
                        }

}


/*************************************************************************/
 // Starts Counting through L,M or H-PIN - according to mode (mode=L,M or H)
 void start_counting (char mode)
{
                switch (mode)
                        {
                                case 'L':
                                        SH_PIN=on;
```

```
                C_PIN=on;
                L_PIN=on;
            break;

            case 'M':
                SH_PIN=on;
                C_PIN=on;
                M_PIN=on;
            break;

            case 'H':
                SH_PIN=on;
                C_PIN=on;
                H_PIN=on;

                // Some Calibration Delay
                Nop();                  // wait 1µs for the counter to count
                Nop();
            break;
        }

}


/*************************************************************************/
// Resets the counter  (works only when counter is in pause mode)
void resetcounter (void)
{

// Reset external counter
```

```
RST_PIN=off;

// Resets Timer0 and starts the Timer 0 - interrupt, whitch takes place at timer's 0 overflow

OpenTimer0( TIMER_INT_ON &        // Interrupt enabled
T0_16BIT &                  // 16 BIT
T0_SOURCE_EXT &                        // External clock source (I/O pin)
T0_PS_1_256 &                          // 1:256 prescale
T0_EDGE_FALL);              // External clock on falling edge
WriteTimer0(0);                        // Initializes Timer0 at 0.


INTCON2bits.TMR0IP=1;      // Sets TMR0 Overflow Interrupt Priority at High level
RCONbits.IPEN=1;           // 1 = Enable priority levels on interrupts
INTCONbits.PEIE=0;                 // Disables all low priority peripheral interrupts
INTCONbits.GIE=1;          // Enables all high priority interrupts
INTCONbits.TMR0IE=1;       // Enables the TMR0 overflow interrupt
Overflow=0;                // Resets overflow variable

RST_PIN=on;
}


/*****************************************************************************/
// Counter increment by 1 (works only when counter is in pause mode)
void counterup (void)
{
//Creates one pulse at the counter strobe control pin - transition from high to low logic level
SH_PIN=on;
Nop();
```

```
SH_PIN=off;
Nop();
}


/*************************************************************************/
// Shifts out & returns, prescaler's internal counter value
 int prescaler(void)
{
int value=0;
int countr=0;
int i=0;
countr=ReadTimer0();                    // Reads TMR0 counter value and passes it to the countr variable
SH_PIN=off;
C_PIN=off;
        while (countr==ReadTimer0())
                {
                counterup();
                i++;
                }
value=256-i;
return value;
}


/*************************************************************************/
// Returns L or M if a low or medium frequency measurement must take place, respectively
char low_or_medium (void)
{
        char i='L';
```

```
        int k;
        start_counting('M');       // Starts the counter
        Delay1KTCYx(50);                    // wait 0.05s for the counter to count
        pause_counter('M');
        k=ReadTimer0();
        if (k>12) i='M';
        resetcounter ();
        return i;
}


/*****************************************************************************/
// Returns the value of the external counter (return value is from 0 to 15)
int external_counter (void)
{
        int a;
        a=0;
        if (D0_PIN) a=a+1;
        if (D1_PIN) a=a+2;
        if (D2_PIN) a=a+4;
        SH_PIN=on;
        C_PIN=on;
        if (D3_PIN) a=a+8;
        C_PIN=off;
        return a;
}
```

```
/* $Id: busyxlcd.c,v 1.1 2003/12/09 22:52:09 GrosbaJ Exp $ */
#include <p18cxxx.h>
#include <xlcd.h>

/************************************************************************
*       Function Name:  BusyXLCD                                        *
*       Return Value:   char: busy status of LCD controller             *
*       Parameters:     void                                            *
*       Description:    This routine reads the busy status of the       *
*                       Hitachi HD44780 LCD controller.                 *
************************************************************************/
unsigned char BusyXLCD(void)
{
        RW_PIN = 1;                     // Set the control bits for read
        RS_PIN = 0;
        DelayFor18TCY();
        E_PIN = 1;                      // Clock in the command
        DelayFor18TCY();
#ifdef BIT8                             // 8-bit interface
        if(DATA_PORT&0x80)              // Read bit 7 (busy bit)
        {                               // If high
                E_PIN = 0;              // Reset clock line
                RW_PIN = 0;             // Reset control line
                return 1;               // Return TRUE
        }
        else                            // Bit 7 low
```

```
	{
		E_PIN = 0;		// Reset clock line
		RW_PIN = 0;		// Reset control line
		return 0;		// Return FALSE
	}
#else				// 4-bit interface
#ifdef UPPER			// Upper nibble interface
	if(DATA_PORT&0x80)
#else				// Lower nibble interface
	if(DATA_PORT&0x08)
#endif
	{
		E_PIN = 0;		// Reset clock line
		DelayFor18TCY();
		E_PIN = 1;		// Clock out other nibble
		DelayFor18TCY();
		E_PIN = 0;
		RW_PIN	=	0;		// Reset control line
**********************************************************************************************************
		return 1;		// Return TRUE - Κατά τη διάρκεια της προσομοίωσης βάλτε 0 αντί 1 σ' αυτή εδώ την γραμμή - διαφορετικά το πρόγραμμα κολλάει.
	}
	else			// Busy bit is low
	{
		E_PIN = 0;		// Reset clock line
		DelayFor18TCY();
		E_PIN = 1;		// Clock out other nibble
		DelayFor18TCY();
		E_PIN = 0;
		RW_PIN = 0;		// Reset control line
```

```
            return 0;               // Return FALSE
        }
#endif
}
#include "Common.h"         /* General Definitions          */
#include <p18cxxx.h>        /* TRISB and PORTB declarations */
#include <xlcd.h>           /* LCD purpose routines         */
#include <delays.h>         /* Delay purpose routines       */

/*functions declaration*/
char buttonpressed (void);     // Return 0,1,2,3 or 4 if no button, mode button, function button, units button or channel button is being
pressed.
void action(unsigned char but);  // Perfoms the appropriate function if button "but" has been pressed (but - 1,2,3 or 4)
void disst(void);                                // Displays current device state at LCD's second line,
extern void displayfreq (unsigned long freq);


/*variables declaration*/
extern volatile float Overflow;
extern char mode;
extern char function;
extern float maxhold;
extern float minhold;
extern char units;
extern char channel;
extern unsigned long frequency;

/*****************************************************************************************************************/
// Return 0,1,2,3 or 4 if no button, mode button, function button,  units button, or channel button is being pressed.
char buttonpressed (void)
```

```
{
// Button MODE           ---> KEY1
// Button FUNCTION       ---> KEY2
// Button UNITS          ---> KEY3
// Button CHANNEL        ---> KEY4

char state=0;

if (KEY1) state=3;
if (KEY2) state=1;
if (KEY3) state=2;
if (KEY4) state=4;
return state;

}


/*****************************************************************************************************************/
// Perfoms the appropriate function if button "but" has been pressed (but - 1,2,3 or 4)
void action(unsigned char but)
{
Overflow=0;               // resets overflow variable

if (but==1)                                   // Mode button has been pressed
        {
        mode++;
      if (mode>2) mode=0;
                displayfreq (frequency);   // Display frequency
                disst();
```

```
        }

if (but==2)                             // Function button has been pressed
        {
                function++;
                if (function>2) function=0;
        maxhold=0;
                minhold=19999999999;
                disst();
        }

if (but==3)                             // Units button has been pressed
        {

                units++;
                if (units>2) units=0;
        displayfreq (frequency);   // Display frequency
        }

if (but==4)                             // Channel button has been pressed
        {
                channel++;
                if (channel>2) channel=1;
                displayfreq (frequency);   // Display frequency
        }


//      // Displays current device state on the LCD
//      while (buttonpressed()>0)Delay10KTCYx(1);    //waits for any button to be de-pressed
```

```
        Delay10KTCYx(25);
}


#include "Common.h"        /* General Definitions          */
#include <p18f242.h>       /* pic18f242                    */
#include <p18cxxx.h>       /* TRISB and PORTB declarations */
#include <delays.h>        /* Delay purpose routines       */
#include <xlcd.h>          /* LCD purpose routines         */

/*functions declaration*/
void disst(void);                                   // Displays current device state at LCD's second line,
                                                                //according to current channel, mode and function
void DelayFor18TCY( void);                          /* Delay of 18TCY - needed for xlcd.h routines*/
void DelayPORXLCD (void);                       /* Delay of 18ms - needed for xlcd.h routines */
void DelayXLCD (void);                              /* Delay of 7ms - needed for xlcd.h routines */


// variables - declaration
extern char mode;
extern char units;
extern char function;

/*************************************************************************/
// Displays current device state at LCD's second line
void disst(void)
{
while( BusyXLCD() );                    // waits until LCD not busy
SetDDRamAddr(0x40);                     // Sets LCDs cursor on first position on second line
```

```
switch (mode)                           // mode=0,1,2--> normal, fast or slow counting mode respectively
       {
         case 1:                        // Normal counting mode
                                while( BusyXLCD()) ;            // waits until LCD not busy
                                putrsXLCD( "Normal-" );                 // Displays "Normal-"
              break;

         case 0:                        // Fast counting mode
                                while( BusyXLCD()) ;            // waits until LCD not busy
                                putrsXLCD( "Fast - " );                 // Displays "Fast-"
         break;

                  case 2:                       // Slow counting mode - valid only for channel A

                                while( BusyXLCD()) ;     // waits until LCD not busy
                                putrsXLCD( "Slow - " );       // Displays "Slow-"

         break;
      }

switch (function)                       // function=0 --> Realtime measuring, function=1 --> Max Hold measuring, function=2 -->
Min Hold measuring
       {
              case 0:                   // Realtime measurement
                                while( BusyXLCD() );         // waits until LCD not busy
                                putrsXLCD( "Real Time" );   // Displays "Real Time"
                        break;

          case 1:                       // Max Hold measurement
                                while( BusyXLCD() );         // waits until LCD not busy
```

```
                                        putrsXLCD( "Max Hold " );    // Displays "Max Hold"
            break;

                        case 2:                             // Min Hold measurement
                                        while( BusyXLCD() );            // waits until LCD not busy
                                        putrsXLCD( "Min Hold " );      // Displays "Min Hold"
            break;
                    }


}


/*********************************************************************/
/* Delay of 18TCY - needed for xlcd.h routines*/
void DelayFor18TCY( void )
{
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
Nop();
```

```
Nop();
Nop();
Nop();
Nop();
Nop();
return;
}

/*********************************************************************/
/* Delay of 18ms - needed for xlcd.h routines */
void DelayPORXLCD (void)
{
Delay1KTCYx(18); // Delay of 18ms
// Cycles = (TimeDelay * Fosc) / 4
// Cycles = (18ms * 4MHz) / 4
// Cycles = 18,000
return;
}
/*********************************************************************/
/* Delay of 7ms - needed for xlcd.h routines */
void DelayXLCD (void)
{
Delay1KTCYx(7); // Delay of 7ms
// Cycles = (TimeDelay * Fosc) / 4
// Cycles = (7ms * 4MHz) / 4
// Cycles = 7,000
Nop();
Nop();
return;
}
```

3.3) Αρχεία του software

- **Main.c**

Το αρχείο αυτό περιέχει όλες τις κύριες ρουτίνες του προγράμματος.

- **Counter.c**

Το αρχείο αυτό περιέχει όλες τις ρουτίνες που ελέγχουν τον μετρητή 28-bit. Αυτές είναι η έναρξη, η διακοπή, ο μηδενισμός, η παύση, η αύξηση του μετρτή κατά 1, υπερχειλήσεις και επιστροφή της τιμής του εσωτερικού προδιαιρέτη, ανάγνωση και επιστροφή της τιμής του εξωτερικού μετρητή.

- **Busyxlcd.c**

Το αρχείο αυτό περιέχει τη ρούτινα BusyXLCD η οποία διαβάζει τη κατάσταση απασχόλησης του μικροελεγκτή του LCD (Hitachi HD44780 LCD compatible).

- **Buttons.c**

Το αρχείο αυτό περιέχει την ρουτίνα “buttonpressed” , η οποία ελέγχει το πληκτρολόγιο και επιστρέφει τη κατάλληλη τιμή σύμφωνα με ποιο κουμπί πατιέται και άλλη μια,την “action”, η οποία εκτελεί την κατάλληλη διαδικασία σύμφωνα με ποιο κουμπί έχει πατηθεί.

- **Display.c**

Το αρχείο αυτό περιλαμβάνει μια ρουτίνα απεικόνισης που χρησιμοποιείται για να απεικονίσει την τρέχουσα κατάσταση της συσκευής στη δεύτερη γραμμή του LCD και μερικές ρουτίνες καθυστέρησης που απαιτούνται για το LCD.

Υπάρχει επίσης ένα αρχείο στο κώδικα (Common.h), το οποίο περιέχει γενικούς ορισμούς που αφορούν κυρίως την απεικόνιση των I/O pins.

3.4) Επεξήγηση βασικών ρουτινών

3.4.1) Υπορουτίνα «Display freq(0)»

Είναι υπεύθυνη για τον υπολογισμό και την απεικόνιση της συχνότητας στην οθόνη. Επειδή την $1^{η}$ φορά που καλείται η συγκεκριμένη υπορουτίνα έχει σαν όρισμα το “0” , το αποτέλεσμα των υπολογισμών είναι επίσης “0”. Οπότε, όταν ξεκινάει η λειτουργία της συσκευής εμφανίζεται στην οθόνη “0” Hz.

- **Έλέγχει την τιμή του «Mode»:**

Αν είναι σε fast mode μετράει για 0.1sec, αν είναι σε normal mode μετράει για 1sec ενώ σε slow mode μετράει για 10sec. Κατά την διάρκεια των μετρήσεων γίνεται έλεγχος αν έχει πατηθεί κάποιο button αντίστοιχα.

- **Έπειτα ελέγχεται η τιμή του «Function»**

Εφόσον είναι “0”, ”1”, ή ”2” ανάλογα αντιστοιχεί στη Real Time Frequency, Max Hold Frequency & Min Hold Frequency. Επίσης ελέγχεται η κατάσταση του LED.

3.4.2) Υπορουτίνα «channel»

Είναι υπεύθυνη για την επιλογή του καναλιού που θα επιλεγεί ανάλογα με την τιμή της συχνότητας που έχει μετρηθεί. Μπορεί να πάρει τις τιμές “1” ή “2” ανάλογα με τη μπάντα στην οποία βρίσκεται η μετρηθήσα συχνότητα.

3.4.3) Υπορουτίνα «Function»

Αν τυχόν πατηθεί το button “Function” από τον χρήστη, ο μικροεπεξεργαστής εκτελεί ένα συγκεκριμένο block κώδικα από τα συνολικά τρία block κώδικα που περιέχονται στην εντολή «switch (function)».

- **Function= “0”**

Αν το όρισμα function πάρει την τιμή “0”, τότε εμφανίζεται στο display η τιμή της Realtime frequency. Ουσιαστικά, διακρίνεται η συχνότητα που μετριέται εκείνη την χρονική στιγμή, εμφανίζοντας το μήνυμα :

"Real Time"

- **Function= “1”**

Αν η συνάρτηση function πάρει την τιμή “1”, τότε εμφανίζεται στο display η τιμή της Maxhold frequency, δηλαδή η μέγιστη συχνότητα που έχει μετρηθεί, εμφανίζοντας το μήνυμα :

"Max Hold "

- **Function= “2”**

Αν η συνάρτηση function πάρει την τιμή “2”, τότε εμφανίζεται στο display η τιμή της Minhold frequency, δηλαδή η ελάχιστη συχνότητα που έχει μετρηθεί εμφανίζοντας στο Display το μήνυμα:

"Min Hold "

3.4.4) Υπορουτίνα «Units»

Ο διακόπτης units, ο οποίος παίρνει τιμές “0”, “1” & ”2” και ανάλογα βάζει την μονάδα Hz, KHz, MHz αντίστοιχα. Δηλαδή εμφανίζεται στο Display:

F =”___ Hz” ή F =”___ KHz” ή F =”___ MHz”

3.4.5) Υπορουτίνα «disst»

Η ρουτίνα αυτή είναι υπεύθυνη για τη κατάσταση της συσκευής εφόσον έχει πατηθεί κάποιο button, και την εμφανίζει στην $2^{η}$ γραμμή της οθόνης.

3.4.6) Υπορουτίνα «led»

Όσον αφορά το led, ο μικροεπεξεργαστής εκτελεί ένα συγκεκριμένο block κώδικα από τα συνολικά δύο block κώδικα που περιέχονται στην ρουτίνα switch (led). Μέσω αυτής γίνεται έλεγχος της κατάστασης (on ή off) του led και αναλόγως γίνεται η ενεργοποίηση ή η απενεργοποίησή του.

3.4.7) Υπορουτίνα «freq (void)»

Η ρουτίνα «freq (void)» μετράει το πλήθος παλμών που έχουν μετρηθεί και το αποθηκεύει στο buffer. Σύμφωνα με την εξίσωση αυτή υπολογίζουμε τον αριθμό των παλμών που έχουν μετρηθεί μέχρι εκείνη την στιγμή :

buffer=(counter2)+(16*buf)+(counter*4096), όπου:

counter2 = η τιμή του εξωτερικού counter 4-bit

buf = η τιμή του προδιαιρέτη (prescaler) 8-bit

counter = η τιμή του εσωτερικού counter 16-bit

3.4.8) Υπορουτίνα «action»

Εκτελεί την κατάλληλη διαδικασία, εφόσον έχει πατηθεί κάποιο κουμπί. Οι τιμές που μπορεί να πάρει είναι “1”,“2”,”3” ή ”4” όπου η καθεμία αντιστοιχεί στο channel button,mode button, function button, & units button.

3.4.9) Υπορουτίνα «counterup»

Η ρουτίνα αυτή είναι υπεύθυνη για τη δημιουργία παλμών μέσω του SH pin.

3.5) Υπορουτίνα «prescaler»

Διαβάζει την τιμή του prescaler και για όσο ταυτίζεται με την τιμή του μετρητή του TIMER0, καλεί την υπορουτίνα counterup για την δημιουργία παλμών.

3.6) POYTINA MAIN

```
void main ()
{
	char meas;

	Delay10KTCYx(20);		// wait 0.2s for peripherals to be initialized

	TRISB=0;			// configure portB for output
	TRISC=15;			// configure portC for input (RC0-RC3) and output (RC4-RC7)
	TRISA=23;			// configure portA for input (RA0-RA2), (RA4) and output (RA3,RA5)
	ADCON1=6;			// Set ADCON1 register to 6 - ADCON1 must be 6 or 7 to enable PORTA bits as digital I/O.

	//Initializes LCD I/O pins used to control HD44780 LCD controller in 4bit interface and initializes LCD as a 5x7 multiple line.
	while( BusyXLCD() );		// waits until LCD not busy
	OpenXLCD(FOUR_BIT & LINES_5X7 );

	while( BusyXLCD() );		// waits until LCD not busy
	WriteCmdXLCD( CURSOR_OFF );	// Cursor off on LCD

	while( BusyXLCD() );		// waits until LCD not busy
	SetDDRamAddr(0x00);		// Sets LCDs cursor on first position on first line
	putrsXLCD("Frequency  Meter");	// Displays " FREQUENCY  METER" on LCD's first line

	while( BusyXLCD() );		// waits until LCD not busy
	SetDDRamAddr(0x40);		// Sets LCDs cursor on first position on second line
	putrsXLCD("Chania.TEI-Crete");	// Displays "Chania.TEI-Crete" on LCD's second line
	LED=off;			// Turns off Led
	led=off;
```

```
Delay10KTCYx(200);              // wait 2s
Delay10KTCYx(100);              // wait 1s
disst();
displayfreq(0);

// infinitive loop starts here
label:SH_PIN=on;                  // Sets the counter strobe control pin at high logic level
pause_counter  ('L');
pause_counter  ('M');
pause_counter  ('H');
Delay1KTCYx(5);                                         // wait 5ms
resetcounter ();
Delay1KTCYx(5);                                         // wait 5ms
button=0;                                               // resets button variable
Overflow=0;                  // resets overflow variable

switch (channel)

        {

                case 1:
                        meas=low_or_medium();
                break;

                case 2:
                        meas='H';
                break;
        }
```

```
  switch (mode)                                 // mode=0,1,2--> fast, normal, or slow counting mode respectively
 {
  case 0:                       // Fast counting mode has been selected (counts for 0.1sec)
                        start_counting (meas);      // Starts the counter
                        delay (0);                                      // wait 0.1 sec
                        pause_counter(meas);        // Stops counting
        break;

        case 1:                                     // Normal counting mode has been selected (counts for 1sec)
                        start_counting (meas);      // Starts the counter
                        delay (1);                                      // wait 1 sec
                        pause_counter(meas);        // Stops counting
                break;

                case 2:                              // Slow counting mode has been selected (counts for 10sec)
                        start_counting (meas);      // Starts the counter
                        delay (2);                                      // wait 10 sec
                        pause_counter(meas);        // Stops counting
                break;

}

  Delay1KTCYx(1);                                       // wait 1ms
  if (button>0) goto label;

  freq();                                               // Finds current frequency

  switch (function)
          {
                case 0:                                         // Displays Frequency
```

```
                displayfreq(frequency);
            break;

            case 1:                                 // Displays Max Hold frequency
                displayfreq(maxhold);
            break;

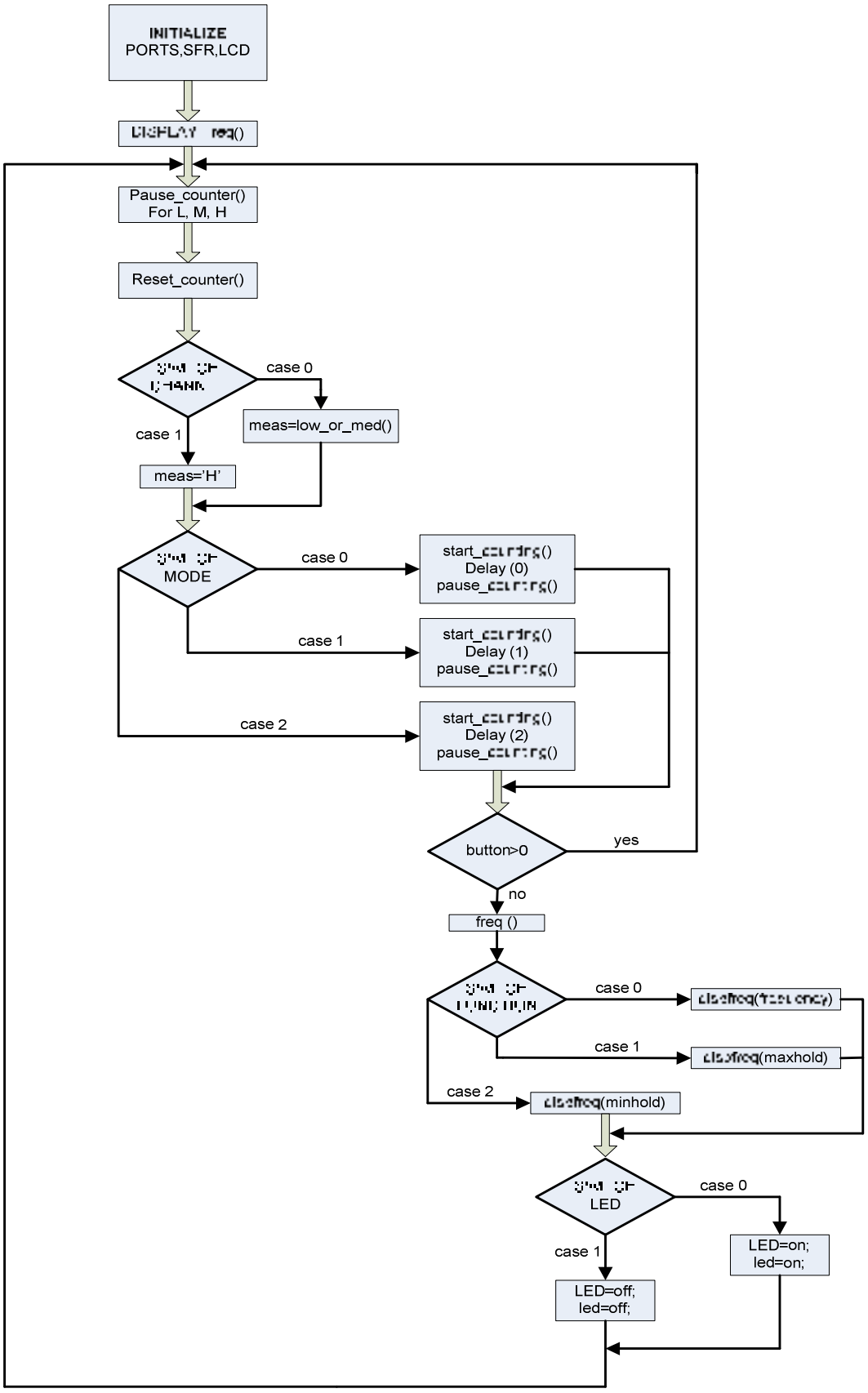
            case 2:                                 // Displays Min Hold frequency
                displayfreq(minhold);
            break;
    }

    switch (led)
        {
            case 0:
                LED=on;             // Turns on Led
                led=on;
            break;

            case 1:
                LED=off;            // Turns off Led
                led=off;
            break;
        }

    goto label;                                     // infinitive loop ends here
}
```

3.6.1) Διάγραμμα ροής της ρουτίνας Main



3.6.2) Ανάλυση της ρουτίνας Main

Μόλις ενεργοποιηθεί το κύκλωμα του f-meter, ο μικροεπεξεργαστής εκτελεί την ρουτίνα main. Στην πρώτη γραμμή της main πραγματοποιείται καθυστέρηση 200msec για να αρχικοποιηθούν τα περιφερειακά, μέσω της εντολής Delay10KTCYx(20) . Έπειτα ο καταχωρητής TRISB τίθεται σε λογικό 0 (TRISB=0) προκειμένου όλα τα pin της θύρας B του μικροεπεξεργαστή να αρχικοποιηθούν για ψηφιακή έξοδο(output). Ο καταχωρητής TRISC τίθεται σε F (TRISC=15) προκειμένου όλα τα pin της θύρας C του μικροεπεξεργαστή να αρχικοποιηθούν για ψηφιακή είσοδο(input). Μετέπειτα, ο καταχωρητής TRISA τίθεται σε λογικό 17 (TRISA=23) προκειμένου όλα τα pin της θύρας A του μικροεπεξεργαστή να αρχικοποιηθούν για ψηφιακή έξοδο(output). Εξίσου, ο καταχωρητής ADCON1 τίθεται σε λογικό 6 (ADCON1=6) προκειμένου όλα τα pin της θύρας 1 του μικροεπεξεργαστή να αρχικοποιηθούν για ψηφιακή έξοδο(output)/ είσοδο(input). Ακολουθεί μία χρονοκαθυστέρηση εφόσον απασχολημένο το LCD, μέσω της εντολής whileBusyXLCD() OpenXLCD(FOUR_BIT & LINES_5X7)) και το display παραμένει κενό με την εντολή : WriteCmdXLCD(CURSOR_OFF). Έπειτα ο αόρατος κέρσορας βρίσκεται στην $1^{η}$ θέση αριστερά της $1^{ης}$ γραμμής με την εντολή SetDDRamAddr(0x00).

Στην οθόνη της συσκευής απεικονίζεται το μήνυμα: "Frequency Meter" μέσω της εντολής putrsXLCD("Frequency Meter"). Ακολουθεί η εντολή SetDDRamAddr(0x40) και ο κέρσορας που δεν είναι ορατός, τώρα βρίσκεται στην $1^{η}$ θέση αριστερά της $2^{ης}$ γραμμής του display. Μέσω της εντολής putrsXLCD("Chania.TEI-Crete") στην οθόνη της συσκευής εμφανίζεται το δεύτερο μήνυμα: "Chania.TEI-Crete" . Στη συνέχεια σβήνει το LED (LED=off;). Ακολουθεί πάλι καθυστέρηση 3sec με τις εντολές Delay10KTCYx(200) Delay10KTCYx(100). Καλούνται οι υπορουτίνες disst( ) & displayfreq(0) που είναι υπεύθυνες για την εμφάνιση της κατάστασης της συσκευής στο LCD και την δήλωση των μεταβλητών αντίστοιχα. Μετά ο μικροεπεξεργαστής μεταπίπτει σε έναν ατέρμονο βρόχο θέτοντας το pin ελέγχου του μετρητή, που αποτελεί είσοδο/έξοδο, σε λογικό επίπεδο ‘HIGH’ με την εντολή label:SH_PIN=on;. Βλέπουμε 3 counters (L, M, H) οι οποίοι θέτονται σε κατάσταση pause. Ακολουθεί χρονοκαθυστέρηση και ξεκίνημα του Counter μέσω των εντολών Delay1KTCYx(5) & resetcounter (). Έπειτα ξανακάνει αρχικοποίηση με την βοήθεια των εντολών: button=0 Overflow=0.

- **ΕΠΙΛΟΓΗ ΚΑΝΑΛΙΟΥ**

Στη συνέχεια ακολουθεί η διαλογή μεταξύ των τριών καναλιών (L, M, H). Το κανάλι High (70MHz-1.5GHz) επιλέγεται από το χρήστη. Η διαλογή στη μπάντα συχνοτήτων (0.1Hz-99MHz) μεταξύ Low ή Medium επιτυγχάνεται κάνοντας αρχικά μια μέτρηση, ώστε να δει ο επεξεργαστής που κυμαίνεται περίπου η συχνότητα που θα διαλέξει. Σε αυτή τη περίπτωση το μηχάνημα επιλέγει αν θα χρησιμοποιηθεί Low ή Medium μπάντα, δηλαδή αν θα εφαρμοστεί το φίλτρο στη περίπτωση συχνότητας κάτω των 100MHz.

Το πρόβλημα που έχουμε να αντιμετωπίσουμε είναι ότι ο συγκριτής δεν μπορεί να διακρίνει τις χαμηλές συχνότητες. Δηλαδή, μπορεί να έχουμε μια χαμηλή συχνότητα και μετρώντας την χώρις φίλτρο, να την αντιλαμβάνεται ως υψηλή. Το πρώτο πράγμα που κάνουμε σε αυτή τη περίπτωση είναι να μετρήσουμε στη χαμηλή μπάντα. Η μέτρηση γίνεται μέσω της ρουτίνας low_or_medium. Η συγκεκριμένη ρουτίνα κάνει μια μικρή μέτρηση στη Low μπάντα. Αν μετρήσουμε και βρούμε μια συχνότητα που είναι λογική στα όρια της Low μπάντας, τότε επιλέγουμε αυτή τη μπάντα. Εάν μετρήσουμε μηδέν, τότε κάνουμε μια μέτρηση στη High μπάντα. Εφόσον μετρήσουμε μια συχνότητα που είναι λογική στα όρια της High μπάντας τότε επιλέγουμε αυτή τη μπάντα. Αν τώρα και στην χαμηλή και στην υψηλή μπάντα η μέτρηση είναι μηδέν, τότε και η συχνότητα είναι μηδέν.

- **ΕΠΙΛΟΓΗ MODE**

Ακολούθως επιλέγεται η ακρίβεια μέτρησης μέσω του mode, δηλαδή με τί σταθερά χρόνου θα μετρήσουμε. Το πλήκτρο “Gate Time” “παίρνει” λογικές τιμές “0” , “1” & “2” και αναλόγως επιλέγει fast, normal ή slow mode. Η επιλογή γίνεται από το πλήκτρο. Η μεταβλητή mode αλλάζει κατάσταση καθε φορά που πατάμε το πλήκτρο. Δηλαδή η mode αναγνωρίζει πόσες φορές έχει πατηθεί το πλήκτρο. Ανάλογα με το πάτημα του πλήκτρου, επιλέγεται το χρονικό διάστημα που θα μετρήσει. Μετά αρχίζει η διαδικασία της μέτρησης και αφού ξεκινήσει ο counter αναμένει είτε για 0.1sec είτε για 1sec είτε για 10sec, ανάλογα την επιλογή του mode.

- **ΕΠΙΛΟΓΗ FUNCTION**

Μετά το τέλος του ελέγχου του switch (mode), ακολουθεί ο έλεγχος του switch (function). Η μεταβλητή function παίρνει τιμές “0”, ”1” & ”2” και ανάλογα μετράει την Realtime freq, Maxhold freq & Minhold freq αντίστοιχα,κάθε φορά που πατάμε το κουμπί function.

- Στην περίπτωση που η function πάρει την τιμή “0” (case 0), εμφανίζει στο Display την τρέχουσα μετρούμενη συχνότητα, με την εντολή displayfreq(frequency),
- Στην περίπτωση που η function πάρει την τιμή “1” (case 1) εμφανίζει στο Display την μέγιστη μετρηθήσα τιμή της συχνότητας, με την εντολή displayfreq(maxhold).
- Στην περίπτωση που η function πάρει την τιμή “2” (case 2) εμφανίζει στο Display την ελάχιστη μετρηθήσα τιμή της συχνότητας, με την εντολή displayfreq(minhold).

- **ΕΠΙΛΟΓΗ ΜΟΝΑΔΩΝ**

Επίσης, ανάλογα με τί μόναδες (Hz,KHz,MHz) έχουν επιλεχθεί, γίνεται και με διαφορετικό τρόπο η απεικόνιση.

- **ΕΛΕΓΧΟΣ LED**

Τέλος, μέσω μιας ρουτίνας, αναβοσβήνει το LED κάθε φορά που γίνεται μια απεικόνιση. Εξετάζει αν είναι σβηστό, οπότε και το ανάβει. Εάν είναι αναμμένο το σβήνει.

ΜΕΡΟΣ ΤΕΤΑΡΤΟ

ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΗΣ ΣΥΣΚΕΥΗΣ

4.1) Ακρίβεια και σφάλματα στην μέτρηση συχνότητας

Η μέτρηση της συχνότητας γίνεται σε πραγματικό χρόνο, με αναμενόμενη ακρίβεια 10Hz, 1HZ και 0.1Hz για την είσοδο A, και 400Hz, 40Hz και 4Hz για την είσοδο B (εξαιτίας της χρήσης του προδιαιρέτη MC12080 με λόγο διαίρεσης 1:40). Η μέγιστη θεωρητική ακρίβεια για την περίπτωση μέτρησης της συχνότητας μπορεί να προσδιοριστεί από μια απλή αριθμητική εφαρμογή της σχέσης (1) του πρώτου μέρους.
Η ακρίβεια της συχνότητας επιτυγχάνεται με την χρήση του κρυστάλλου των 4MHz, ο οποίος είναι απαραίτητος γιατί αποτελεί την βάση χρόνου του συχνομέτρου. Χρησιμοποιούμε απαραίτητα κρύσταλλο, διότι η ακριβής μέτρηση της συχνότητας προϋποθέτει την ύπαρξη ακριβούς χρονισμού. Απαραίτητη είναι και η χρήση ενός μεταβλητού πυκνωτή, για τη λειτουργία του ρολογιού, ο οποίος χρησιμεύει στη βαθμονόμηση του οργάνου. Αυτό συμβαίνει, διότι ο κρύσταλλος είναι ακριβής μονάχα σε μια σταθέρη θερμοκρασία, δηλαδή όταν η θερμοκρασία αλλάξει, η τιμή του κρυστάλλου, για την ύπαρξη ακριβούς χρονισμού, επίσης θα αλλάξει.

4.2) Ευαισθησία

Ευαισθησία είναι το ελάχιστο πλάτος που έχει η κυματομορφή εισόδου για να μπορεί το συχνόμετρο να μετρήσει.

Με την βοήθεια ενός παλμογράφου και μιας γεννήτριας παλμών πήραμε τις παρακάτω τιμές:

Frequency	CH1 sensitivity (mVp-p)	CH2 sensitivity (dBm)
10Hz	120	-
1KHz	60	-
100KHz	20	-
1MHz	10	-
70MHz	10	-14
200MHz	-	-19
500MHz	-	-20
1GHz	-	-18
1.5GHz	-	-7

Παρατηρούμε ότι υπάρχει μια σημαντική πτώση της ευαισθησίας εισόδου για τις χαμηλές συχνότητες, εξαιτίας του LPF και της κακής απόδοσης του MAX9203 στις χαμηλές συχνότητες. Το μέγιστο πλάτος σήματος που μπορεί με ασφάλεια να εφαρμοστεί στην είσοδο CH1 είναι περίπου 10Vp-p, χωρίς τη χρήση εξασθενητή, και περίπου 200Vp-p με τη χρήση του 1:20 εξασθενητή.

Όσον αφορά την είσοδο CH2, η μέγιστη ισχύς σήματος δεν πρέπει να ξεπερνά τα +13dBm. Τα τρίμμερ R6 & R10 ρυθμίστηκαν για μέγιστη ευαισθησία εισόδου στις εισόδους CH1 και CH2 αντίστοιχα. Η μέγιστη ευαισθησία εισόδου επιτυγχάνεται όταν τα R10 και R6 σταθεροποιούνται για περίπου 3.5V στο 3^{o} pin του IC3 και για μερικά mV πάνω από το μηδέν στο 3^{o} pin του IC1, αντίστοιχα.

4.3) Πρόσθετες δυνατότητες της συσκευής

Η συγκεκριμένη συσκευή έχει περισσότερες δυνατότητες σε σχέση με τα απλά όργανα μέτρησης συχνότητας που υπάρχουν στην αγορά. Και αυτό διότι έχει την δυνατότητα να μετράει για Δt = 0.1sec, 1sec και 10sec, αλλάζοντας απλά την κατάσταση του button “Gate time”. Ο χρήστης έχει την δυνατότητα να μετράει συχνότητες σε τρεις διαφορετικές περιόδους, αλλά και να επιλέγει ποια συχνότητα θέλει να μετρήσει. Έτσι ο χρήστης έχει την ευχέρεια να επιλέξει την μέτρηση οποιασδήποτε συχνότητας επιθυμεί, δηλαδή την Maxhold Frequency (μέγιστη τιμή της συχνότητας), την Minhold Frequency (ελάχιστη τιμή της συχνότητας) αλλά και την Realtime Frequency (στιγμιαία τιμή της συχνότητας), αλλάζοντας απλά την κατάσταση του button “Function”. Με την ίδια λογική μπορεί ο χρήστης να αλλάξει και την μονάδα μέτρησης τους σε Hz, KHz ή MHz αλλάζοντας την κατάσταση του button “Units”.

Αυτή η συσκευή έχει την δυνατότητα να μετράει συχνότητες από 0.1Hz έως 1.5GHz με μεγάλη ακρίβεια και χρησιμοποιείται όχι μόνο για την μέτρηση συχνότητας, αλλά και ως κύρια βαθμίδα ακόμα πιο σύνθετων συχνομέτρων, μεγαλύτερης ακρίβειας και μεγαλύτερου εύρους μέτρησης συχνοτήτων. Επίσης, μπορεί να χρησιμοποιηθεί σε κυκλώματα για την απεικόνιση συχνότητας ραδιοφωνικών δεκτών και πομπών αλλά και άλλων σύγχρονων ηλεκτρονικών συσκευών μετρήσεων. Όπως για παράδειγμα σε μέτρηση μεταβλητής συχνότητας για αστάθεια συσκευής, όπου δίδεται η δυνατότητα να εντοπίζεται το εύρος τιμών που δουλεύει μια γεννήτρια (π.χ. ταλαντωτής αστάθειας). Ακόμη, μπορεί να χρησιμοποιηθεί για την μέτρηση ολίσθησης συχνότητας, όπου σημασία έχει το όριο ταλάντωσης της συχνότητας (MinHold & MaxHold) και όχι η στιγμιαία μετρούμενη τιμή (RealTime Freq).

ΜΕΡΟΣ ΠΕΜΠΤΟ

ΤΥΠΩΜΕΝΟ ΚΥΚΛΩΜΑ ΚΑΙ ΛΕΠΤΟΜΕΡΕΙΕΣ ΚΑΤΑΣΚΕΥΗΣ

5.1) Κατασκευή Πλακέτας

Για την κατασκευή της πλακέτας κάναμε χρήση του προγράμματος Altium Designer. Το πρόγραμμα αυτό μας βοήθησε στην ακριβή τοποθέτηση των εξαρτημάτων πάνω στην πλακέτα, καθώς επίσης και στον σχεδιασμό του τυπωμένου κυκλώματος.

Η πλακέτα που χρησιμοποιήσαμε είναι διπλής όψεως, που σημαίνει ότι σχεδιάσαμε δυο τυπωμένα κυκλώματα, ένα για κάθε όψη. Τα τυπωμένα κυκλώματα, αφού πρώτα εκτυπώθηκαν πάνω σε διαφάνειες, στη συνέχεια τυπώθηκαν πάνω στην πλακέτα με την βοήθεια του μηχανήματος φωτομεταφοράς. Για να φέρουμε την πλακέτα στην τελική της μορφή την περάσαμε από αραιό διάλυμα καυστικής σόδας και τέλος από διάλυμα τριχλωριούχου σιδήρου για να αποχαλκωθούν πλήρως τα σημεία που δεν χρειαζόμαστε.

Οι οπές ανοίχτηκαν με την βοήθεια ηλεκτρονικού δράπανου. Στη συνέχεια επικαλύψαμε την πλακέτα με ειδικό βερνίκι για την αποφυγή τυχών οξειδώσεων και τέλος τοποθετήσαμε και συγκολλήσαμε τα εξαρτήματα.

5.2) Βοηθητικές κατασκευές

Για τον έλεγχο του κυκλώματος του συχνομέτρου απαραίτητη ήταν η κατασκευή και μιας δεύτερης πλακέτας, η οποία ονομάστηκε keyboard, όπου περιέχονται τέσσερα buttons και ένα led.

Επίσης απαραίτητη είναι η χρήση τροφοδοτικού. Χρησιμοποιήθηκε παλμοτροφοδοτικό το οποίο τροφοδοτεί το κύκλωμά μας με $\pm 5V$.



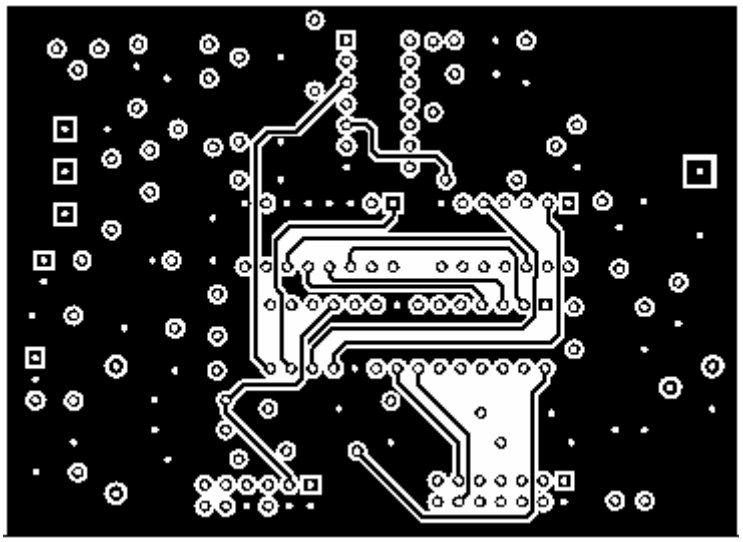
5.3) Πρόσοψη Frequency counter

Ο σχεδιασμός της πρόσοψης της συσκευής έγινε με την βοήθεια του προγράμματος Corel DRAW.

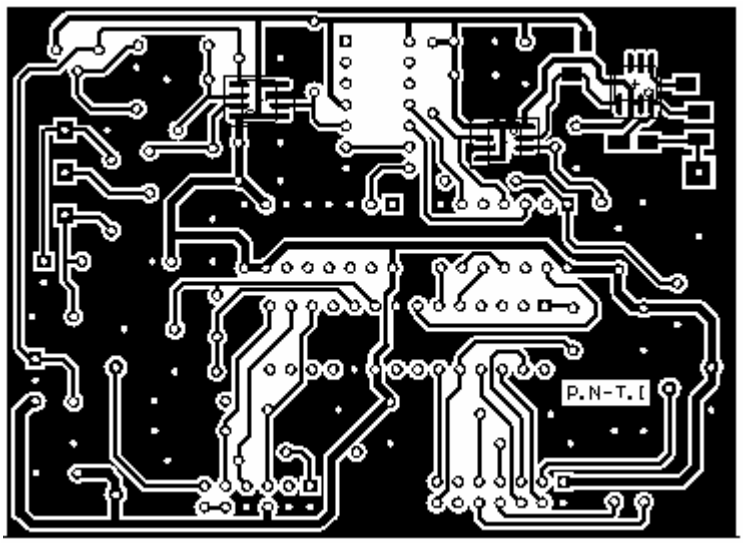
5.4) Τυπωμένο Κύκλωμα

5.4.1) Τυπωμένο Κύκλωμα Πλακέτας

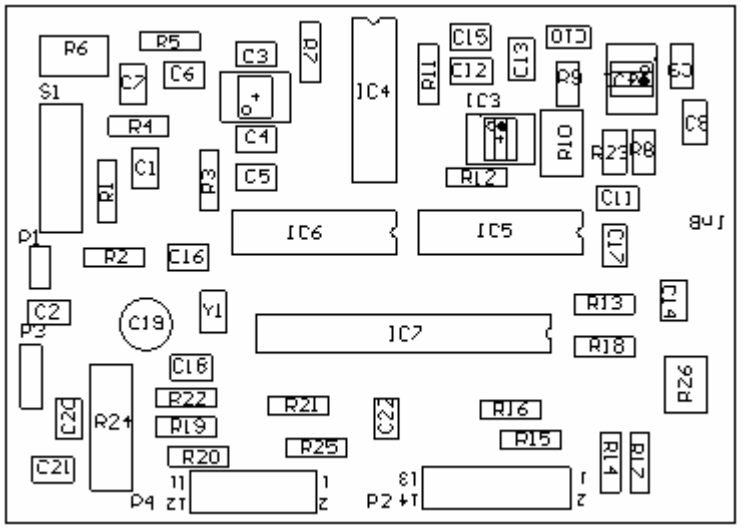
Παρακάτω φαίνεται το τυπωμένο κύκλωμα της κάθε όψης της πλακέτας, καθώς και τα εξαρτήματα της κατασκευής.



Σχήμα 1: Top Layer



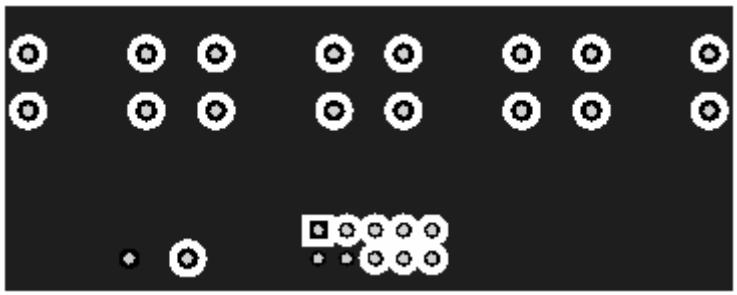
Σχήμα 2: Bottom Layer



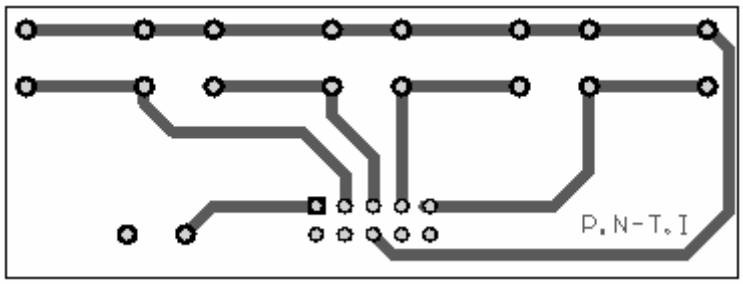
Σχήμα 3: Τοποθέτηση εξαρτημάτων

5.4.2) Τυπωμένο Κύκλωμα πληκτρολογίου

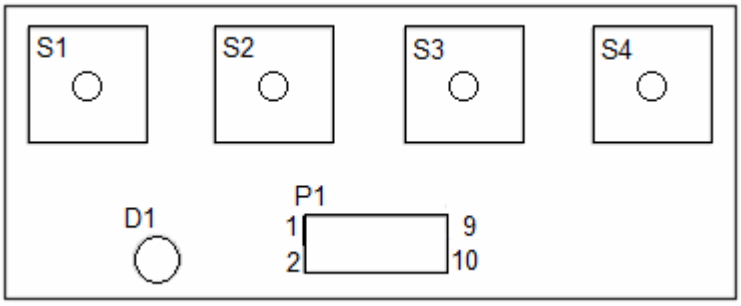
Όπως και στην προηγούμενη παράγραφο έτσι κι εδώ βλέπουμε το τυπωμένο κύκλωμα της κάθε όψης για το keyboard και την τοποθέτηση των εξαρτημάτων.



Σχήμα 1: Top Layer



Σχήμα 2: Bottom Layer



Σχήμα 3: Τοποθέτηση εξαρτημάτων

ΜΕΡΟΣ ΕΚΤΟ

ΒΑΘΜΟΝΟΜΗΣΗ ΤΗΣ ΣΥΣΚΕΥΗΣ

6.1) Τρόπος βαθμονόμησης της συσκευής

Προκειμένου η συσκευή να μετράει σωστά, απαιτείται μια αρχική βαθμονόμηση κατά την φάση της κατασκευής. Από την στιγμή που θα γίνει η βαθμονόμηση της συσκευής και μετά, η συσκευή είναι σε θέση να μετράει ανά πάσα στιγμή με το απλό πάτημα του πλήκτρου, χωρίς να χρειάζεται άλλη βαθμονόμηση. Σε όλες τις συσκευές μετρήσεων υπάρχει το ενδεχόμενο σφαλμάτων και η βαθμονόμηση είναι ένας τρόπος αντιμετώπισης τους.
Η διαδικασία της βαθμονόμησης γίνεται ως εξής:

1. Αρχικά επιλέξαμε μέσω software τις κατάλληλες ρουτίνες καθυστέρησης οι οποίες μας εξασφάλισαν τις ακριβείς βάσεις χρόνου.

2. Στη συνέχεια κάναμε μια τελική ρύθμιση στο ρολόι χρονισμού μέσω του τρίμμερ C19.

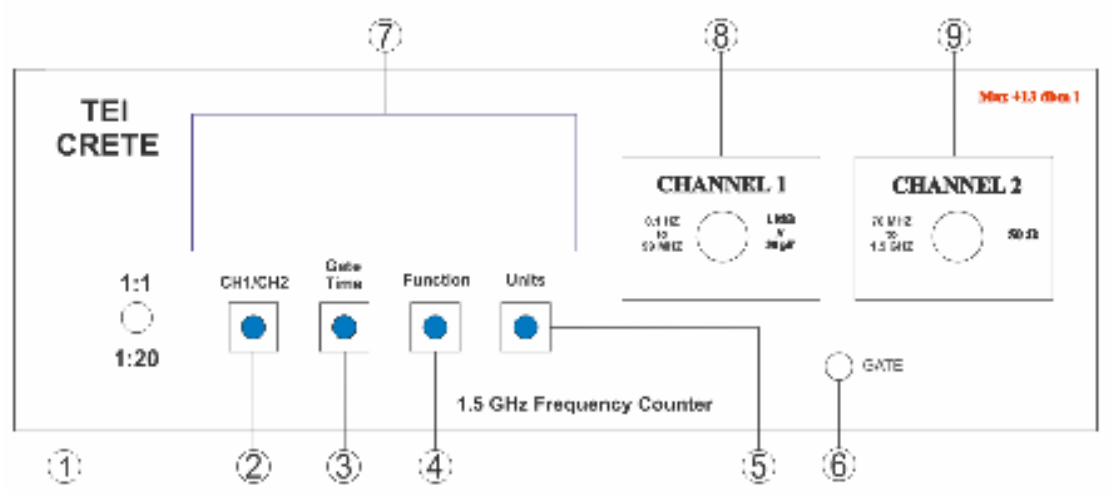
Αξίζει να σημειωθεί ότι παρά τη βαθμονόμηση της συσκευής, η τιμή του κρυστάλλου ενδέχεται να αποκλίνει από την κανονική του τιμή λόγω της θερμοκρασίας. Το φαινόμενο αυτό παρατηρείται πιο πολύ στις υψηλές συχνότητες και έχει σαν αποτέλεσμα να επηρεάζεται η ακρίβεια του συχνομέτρου. Αυτό το πρόβλημα μπορεί να αντιμετωπιστεί με την τοποθέτηση του κρύσταλλου σε “φούρνο” (περιβάλλον σταθερής θερμοκρασίας), οπότε και το συχνόμετρο γίνεται καθαρά επαγγελματικό.

ΜΕΡΟΣ ΕΒΔΟΜΟ

ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΗΣ ΤΟΥ FREQUENCY COUNTER

7.1) ΕΓΧΕΙΡΙΔΙΟ ΧΡΗΣΤΗ





1. Διακόπτης-Εξασθενητής. Ρύθμιση της τάσης του σήματος εισόδου για το κανάλι εισόδου 1.

2. Επιλογή καναλιού μεταξύ CH1 (για συχνότητες από 0.1Hz-99MHz) & CH2 (για συχνότητες από 70MHz-1,5GHz).

3. Με το πάτημα του “Gate Time” button γίνεται αυτοματοποιημένη μέτρηση, είτε σε fast mode κάθε 0.1 δευτερόλεπτο πραγματικού χρόνου, είτε σε normal mode κάθε 1 δευτερόλεπτο πραγματικού χρόνου, είτε σε slow mode κάθε 10 δευτερόλεπτα πραγματικού χρόνου. Η επιλογή αυτή γίνεται από τον χρήστη και η τιμή της αντίστοιχης τιμής της συχνότητας εμφανίζεται στην οθόνη του υγρού κρυστάλλου.

4. Με το πάτημα του “Function” button γίνεται επιλογή της συχνότητας που θέλουμε να μετρήσουμε. Οι επιλογές που διαθέτει η συσκευή είναι, η μέγιστη τιμή συχνότητας (Maxhold frequency), η ελάχιστη τιμή συχνότητας (Minhold frequency) & η μετρούμενη στιγμιαία τιμή της συχνότητας (Realtime frequency).

5. Με το “Units” button επιλέγουμε την μονάδα μέτρησης που θέλουμε να μετρήσουμε την συχνότητα. Οι επιλογές που διαθέτει η συσκευή και σε αυτήν την περίπτωση είναι Hz, KHz, MHz.

6. Το LED αναβοσβήνει όταν γίνεται απεικόνιση της μετρούμενης συχνότητας, με τον ρυθμό που έχει επιλεχθεί από το Gate Time.

7. LCD οθόνη (HD 44780 based character-LCD, V=5V, 2X16).

8. Κανάλι εισόδου συχνότητας CH1.

9. Κανάλι εισόδου συχνότητας CH2.

Τρόπος λειτουργίας

1. Με το άνοιγμα της συσκευής εμφανίζεται το μήνυμα:

“Frequency Meter
Chania.TEI-Crete”



2. Την επόμενη στιγμή μας δείχνει το μήνυμα:
“A 0.0 Hz
Fast - Real Time”

Ουσιαστικά μας δείχνει την μηδενική αρχική τιμή της συχνότητας και την αρχική κατάσταση των Mode, Function & Units buttons.

3. Πατώντας το "**CH1/CH2"** button εμφανίζεται το μύνημα:
"B 0.0 Hz
Fast - Real Time"
Ουσιαστικά ότι έχει επιλεγεί το κανάλι B σαν είσοδος του συχνομέτρου.

4. Πατώντας το "**Gate Time"** button, καθώς η συσκευή αρχίζει να μετράει μια τυχαία τιμή της συχνότητας, παρατηρούμε το led να αναβοσβήνει ανά 1sec και να εμφανίζεται το μήνυμα:
"B.....Hz
Normal - Real Time "

5. Την επόμενη στιγμή, εάν πατηθεί το "**Function"** button μας εμφανίζει το μήνυμα:
"B .....Hz
Normal - Max Hold"

Αυτή η μέτρηση είναι η μέγιστη μετρούμενη τιμή της συχνότητας σε normal mode (1sec).

6. Την επόμενη στιγμή, εάν ξανά- πατηθεί το "**Function"** button μας εμφανίζει το μήνυμα:
"B .....Hz
Normal - Min Hold"
Αυτή η μέτρηση είναι η ελάχιστη μετρούμενη τιμή της συχνότητας σε normal mode (1sec).

7. Πατώντας το "**Units"** button εμφανίζεται στην οθόνη το μήνυμα:
"B…….. KHz
Normal - Min Hold "

Αυτή είναι η στιγμιαία μετρούμενη τιμή της συχνότητας σε normal mode (1sec) σε μονάδα μέτρησης KHz.

8. Πατώντας το ξανά το "**Units"** button εμφανίζεται στην οθόνη το μήνυμα:
"B…….. MHz
Normal - Min Hold "

Αυτή είναι η στιγμιαία μετρούμενη τιμή της συχνότητας σε normal mode (1sec) σε μονάδα μέτρησης MHz.

Η ίδια διαδικασία ισχύει είτε επιλεχθεί fast, είτε normal είτε slow mode. Η μόνη διαφορά είναι ότι η μετρήσεις θα γίνονται ανά 0.1sec, 1sec και 10 sec αντίστοιχα, σε οποιοδήποτε κανάλι εισόδου (CH1/CH2).

7.2) ΤΕΧΝΙΚΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ

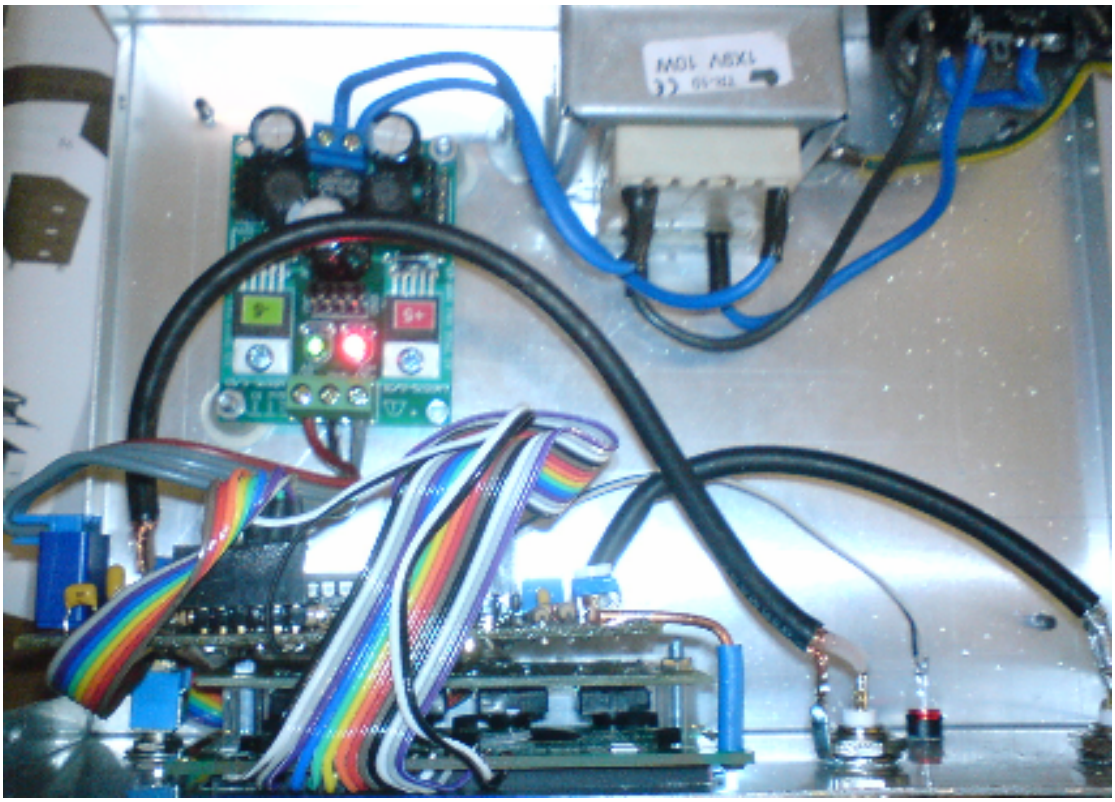
Τάση τροφοδοσίας : 220V / 50Hz
Εύρος μετρούμενης συχνότητας : 0.1Hz ~ 1.5GHz
Οθόνη τύπου υγρού κρυστάλλου (LCD) 2x16 χαρακτήρων
Πλάτος εισόδου:

- CH1

Χωρίς χρήση εξασθενητή : Από 0 έως 10Vp-p
Με χρήση εξασθενητή : Από 0 έως 200Vp-p

- CH2

Από 0 έως +13 dBm



7.3) ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Winstar LCD WP1602D

2. Microchip, PIC18F2525/2620/4525/4620 Data Sheet, DS39626B, Microchip Technology Inc. 2004.

3. HD 44780 based LCD Tutorial

4. Altium Designer 6

5. Χρήση γλώσσας προγραμματισμού Turbo C

6. MPLAB®ICD 2 In-Circuit Debugger/Programmer

7. Philips Semiconductors, 74HC/HCT132 Quad 2-input NAND Schmitt trigger Data Sheet

8. Philips Semiconductors, 74F161A / 74F163A 4-bit binary counter Data Sheet

9. Freescale Semiconductor, Inc, MC12080 Data Sheet, Motorola

10. Microsoft Visio

11. Corel Draw

ΠΕΡΙΕΧΟΜΕΝΑ ΨΗΦΙΑΚΟΥ ΔΙΣΚΟΥ

1. Φυλλάδια Πληροφοριών (Datasheets)
2. Λογισμικό (code)
3. Φωτογραφικό υλικό (photos)
4. Θεωρητικό μέρος (Text Frequency Counter)
5. Διάγραμμα Βαθμίδων
6. Σχηματικά Διαγράμματα (Schematics)
7. PCB projects (Frequency counter & keyboard)
8. Πρόσοψη Συχνόμετρου (Box Mask)