ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΕΦΑΡΜΟΣΜΕΝΩΝ ΜΑΘΗΜΑΤΙΚΩΝ
ΚΑΙ ΦΥΣΙΚΩΝ ΕΠΙΣΤΗΜΩΝ
ΤΟΜΕΑΣ ΦΥΣΙΚΗΣ

ΔΠΜΣ
«ΜΙΚΡΟΣΥΣΤΗΜΑΤΑ ΚΑΙ ΝΑΝΟΔΙΑΤΑΞΕΙΣ»

Αυτοματοποίηση Μετρήσεων Αισθητήρων Τύπου Αντίστασης

ΜΕΤΑΠΤΥΧΙΑΚΗ ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Στόλης Νικόλαος

Επιβλέπων: Τσουκαλάς Δημήτριος,

Καθηγητής Τμήματος Σχολής Εφαρμοσμένων Μαθηματικών και Φυσικών Επιστημών

Αθήνα, Ιούνιος 2021

Περίληψη

Η παρούσα διπλωματική εργασία έχει ως στόχο τον σχεδιασμό και κατασκευή του ηλεκτρονικού συστήματος μέτρησης στα πλαίσια πειραματικής εφαρμογής αισθητήρων με νανοσωματίδια μεταβαλλόμενης αντίστασης. Ως μελέτη περίπτωσης χρησιμοποιούνται αισθητήρες μέτρησης υγρασίας, καθώς και κοινές αντιστάσεις. Ο σχεδιασμός του συστήματος περιλαμβάνει την επίλυση προβλημάτων του αρχικού λογισμικού του μικροελεγκτή και την προσθήκη νέων δυνατοτήτων. Στη συνέχεια έλαβε χώρα ο σχεδιασμός πλακέτας βασισμένης σε προηγούμενο κύκλωμα που είχε υλοποιηθεί πρόχειρα, προκειμένου να είναι εύκολη η διεπαφή του με το μικροελεγκτή. Επιπλέον, πραγματοποιήθηκε ανάπτυξη του λογισμικού μέσω της γλώσσας προγραμματισμού python, με σκοπό τη λήψη δεδομένων από το Arduino και τον έλεγχό του και εν τέλει, την επιτυχή λήψη μετρήσεων.

Οι μετρήσεις που πραγματοποιήθηκαν με το σύστημα που κατασκευάστηκε στην παρούσα εργασία έλαβαν χώρα στο εργαστήριο Ηλεκτρονικών Νανοϋλικών και Συσκευών του Εθνικού Μετσοβείου Πολυτεχνείου. Η διάταξη των μετρήσεων περιελάμβανε ηλεκτροχημικούς αισθητήρες υποστρώματος πολυιμιδίου με επικάλυψη πολυμερούς PIBMA. Ο πειραματικός σχεδιασμός συνίσταται σε δοκιμές διαφόρων ποσοστώσεων υγρασίας και διαφορετικών τυπικών αντιστάσεων.

Τα αποτελέσματα των μετρήσεων με το υπό κατασκευή σύστημα αποτυπώνονται διαγραμματικά συναρτήσει του χρόνου μέτρησης. Παρουσιάζει ιδιαίτερο ενδιαφέρον το γεγονός ότι, εν συγκρίσει με τα αποτελέσματα αντίστοιχων μετρήσεων του εμπορικού συστήματος Keithley 2400 δεν υπάρχει αξιοσημείωτη απόκλιση στα ακρότατα των μετρήσεων. Το εύρημα αυτό είναι σημαντικό, καθώς το κόστος του εμπορικού συστήματος είναι κατά περίπου δύο τάξεις μεγέθους υψηλότερο σε σχέση με το σύστημά μας. Ωστόσο, έχουμε πιο έντονη παρουσία θορύβου, στην περίπτωση μέτρησης αντιστάσεων εκτός του εύρους μετρήσεως του συστήματος. Παρ' όλα αυτά, το σύστημα το οποίο ανατπύξαμε έχει μεγαλύτερη ευελιξία, καθώς είναι φορητό και μπορεί να τροποποιηθεί με πληθώρα επιλογών.

Τα αποτελέσματα του παρόντος έργου μπορούν να θέσουν τα θεμέλια για τη δημιουργία μίας πλατφόρμας χαμηλού κόστους και υψηλής προσαρμοστικότητας, η οποία μπορεί να υλοποιηθεί σε μικρό χρόνο, δυνητικά αυτόνομα και σε περιοχές χαμηλής προσβασιμότητας από εργαστηριακό εξοπλισμό, για όλους τους αισθητήρες μεταβαλλόμενης αντίστασης.

Λέξεις κλειδιά: αισθητήρες, νανοσωματίδια, πολυμερές PIBMA, hardware, software, data acquisition, σύστημα μέτρησης

Abstract

The present dissertation aims at the design and construction of the electronic measurement system in the context of experimental application of nanoparticle infused sensors with variable resistance. Humidity sensors as well as common resistors are used as a case study. System design includes troubleshooting the original microcontroller software and adding new features. Then it involves the design of a printed circuit board based on a previous circuit that had been implemented tentatively, in order to be easy to interface with the microcontroller. In addition, the data acquisition software was developed via the aid of the python programming language, in order to control the Arduino platform, and finally, to successfully take measurements.

The measurements carried out with the system constructed in the present work took place in the laboratory of Electronic Nanomaterials and Devices of the National Technical University of Athens. The measurement device included electrochemical sensors of polyimide substrate coated with PIBMA polymer. The experimental design consists of tests of different humidity percentages and different standard resistances.

The results of the measurements with the given system are plotted as a function of time. The fact that, compared with the results of similar measurements of the Keithley 2400 commercial system, there is no significant deviation in the extremes of the measurements is of particular interest. This finding is important, as the cost of the commercial system is about two orders of magnitude higher than our system. However, our results have more intense presence of noise, in the case of measuring resistances outside the range of our system. Nevertheless, the system we have developed is more flexible, as it is portable and can be modified with a variety of options.

The results of this project can lay the foundation for the creation of a low cost and high adaptability platform, which can be implemented in a short time, potentially autonomous and in areas with no option of using laboratory equipment, for all sensors of variable resistance.

Keywords: sensors, nanoparticles, PIBMA polymer, hardware, software, data acquisition, measurement system

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον κύριο Δημήτριο Τσουκαλά για την εξαιρετική συνεργασία και την συγκαταβατικότητά του.

Επίσης ιδιαίτερη υποστήριξη είχα από τον μεταδιδακτορικό ερευνητή Βαγγέλη Ασλανίδη, ο οποίος ήταν πάντοτε επικοινωνιακός και εξυπηρετικός.

Ιδιαίτερη μνεία στον κύριο Ιωάννη Ράπτη ως υπεύθυνο του ΔΠΜΣ, ο οποίος αφουγκραζόταν αδιαλείπτως τους προβληματισμούς και τις ανάγκες των φοιτητών.

Φυσικά δεν θα μπορούσα να παραλείψω την οικογένειά μου, τους φίλους μου, Σταμάτη και Πέτρο και την αρραβωνιαστικιά μου για την υποστήριξή τους καθ' όλο το διάστημα εκπόνησης της διπλωματικής μου εργασίας.

Στους δικούς μου ανθρώπους που με στήριξαν σε αυτό το έργο,

Και στην Ερμιόνη

Πίνακας Περιεχομένων

1. Εισαγωγή

1.1. Αισθητήρες και η εξέλιξή τους

Οι αισθητήρες αποτελούν πλέον αναπόσπαστο κομμάτι της καθημερινής μας ζωής και παίζουν σημαντικό ρόλο τόσο σε απλές καθημερινές δραστηριότητες όσο και σε κρίσιμες διαδικασίες σύγχρονων τεχνολογικών εφαρμογών. Με αυτόν τον τρόπο η ανίχνευση, η επεξεργασία και η καταγραφή διαφόρων ειδών πληροφορίας πραγματοποιείται αποτελεσματικά, καθιστώντας εφικτή την παρουσίαση και τη σύγκριση αποτελεσμάτων με τελικό σκοπό της λήψη μιας βέλτιστης απόφασης.

Η επεξεργασία ηλεκτρικών σημάτων καταλαμβάνει πλέον το μεγαλύτερο μέρος σε μια διαδικασία μέτρησης, επειδή δεν επηρεάζονται εύκολα από περιβαλλοντικούς παράγοντες σε σχέση με άλλες μεθόδους. Παράλληλα η διαρκής μελέτη υλικών με καλύτερες φυσικές και μηχανικές ιδιότητες αλλά και η ανάπτυξη μικρομηχανικών διεργασιών οδήγησαν στην κυριαρχία των ψηφιακών συστημάτων σε πολλά 10 Η/Υ (Προβολή , Επεξεργασία , Αποθήκευση Δεδομένων) τεχνολογικά πεδία, εφόσον είναι εφικτή η μαζική παραγωγή σε συνδυασμό με μειωμένο κόστος. Η ανάπτυξη μεγάλου πλήθους μικροδομών σε διάφορες διατάξεις είναι ένα σημαντικό όφελος, τόσο στα ηλεκτρονικά κυκλώματα όσο και στους αισθητήρες. Στο πεδίο των αισθητήρων η δυνατότητα δημιουργίας μικροδομών ή ακόμα και νανοδομών κατέστησε δυνατή την ανίχνευση μεταβολών ενός φυσικού μεγέθους, κάτι το οποίο θα ήταν αδύνατο διαφορετικά. Παραδείγματα τέτοιων αισθητήρων είναι οι πιεζοηλεκτρικοί, οι αισθητήρες επιτάχυνσης, αισθητήρες παραμόρφωσης και άλλοι.

Παρόλο το μεγάλο εύρος και τη διαφοροποίηση των μετρητικών οργάνων μπορούμε να διακρίνουμε ένα κοινό τρόπο λειτουργίας, ο οποίος υφίσταται σε κάθε όργανο. Συγκεκριμένα, μια μετρητική διάταξη μπορεί να θεωρηθεί ως μια αλυσίδα επιμέρους στοιχείων, που έχουν προκαθορισμένο ρόλο και αλληλοσυνδέονται κατάλληλα, ώστε να είναι εφικτή η μέτρηση ενός ή περισσότερων φυσικών μεγεθών.

Η γενικευμένη μορφή ενός συστήματος συλλογής και καταγραφής πληροφορίας που είναι γνωστό ως Data Acquisition System στη διεθνή βιβλιογραφία [1], συνίσταται στις παρακάτω λειτουργίες:

1. Είσοδος του μετρούμενου μεγέθους (p, T) στον αισθητήρα
2. Αποστολή σήματος διέγερσης στο αναλογικό σύστημα μέτρησης (ενίσχυση, μετατροπή σε τάση V ή συχνότητα f) και ανατροφοδότηση σήματος στον αισθητήρα
3. Αποστολή τροποποιημένου σήματος εξόδου στο Κεντρικό Σύστημα Ελέγχου (μικροελεγκτής, μικροεπεξεργαστής) και έλεγχος από το χρήστη
4. Ψηφιοποίηση δεδομένων και αποστολή σε Η/Υ μέσω ενσύρματης ή ασύρματης διεπαφής μικροελεγκτή – υπολογιστή
5. Χρήση Η/Υ για προβολή, επεξεργασία και αποθήκευση δεδομένων

Στην εν λόγω διπλωματική εργασία, το κύκλωμα που κατασκευάσαμε βγάζει στην έξοδο τετραγωνικό παλμό. Αυτό σημαίνει ότι δε χρειάζεται να προβούμε σε μετατροπή από αναλογικό σε ψηφιακό σήμα.

Ο σχεδιασμός ενός ολοκληρωμένου συστήματος υλοποιείται σύμφωνα με ορισμένες προδιαγραφές. Μερικές από αυτές είναι:

• Το κόστος σχεδίασης και ανάπτυξης του συστήματος

• Το κόστος κάθε μεμονωμένου εξαρτήματος

• Η συνολική κατανάλωση ισχύος

• Η συνολική απόδοση ως προς την αξιοπιστία και την ταχύτητα εκτέλεσης προκαθορισμένων ενεργειών

• Ο χρόνος προτυποποίησης, κατά τον οποίο το σύστημα λειτουργεί δοκιμαστικά, παρακολουθείται και βελτιστοποιείται προτού βγει στην αγορά

• Οι δυνατότητες του συστήματος να αλλάζει πλήρη ή μερική λειτουργικότητα

Πρωτεύουσα διαδικασία ενός ηλεκτρονικού συστήματος είναι η μέτρηση. Ως μέτρηση ορίζουμε την αντιστοίχιση της ποσότητας εξόδου του αισθητήρα, εξαιτίας μιας μεταβολής στην ποσότητα του φυσικού μεγέθους, στο εξωτερικό περιβάλλον που μελετούμε. Το αποτέλεσμα μιας μέτρησης είναι το μέτρο του φυσικού μεγέθους, συνοδευόμενο με τις αντίστοιχες μονάδες και αντιπροσωπεύει ένα υπαρκτό, φυσικό χαρακτηριστικό [10].

1.2. Χαρακτηριστικά των αισθητήρων

Η σύνδεση του μετρητικού συστήματος με οποιοδήποτε φυσικό περιβάλλον επιτυγχάνεται με τη βοήθεια αισθητήρων. Ένας αισθητήρας διαθέτει σύνολο φυσικών, χημικών και μηχανικών ιδιοτήτων που επιτρέπουν τη μετατροπή μιας μεταβολής ενός φυσικού μεγέθους (πίεση P, θερμοκρασία T, δύναμη F κ.ά.) σε μεταβολή ηλεκτρικού μεγέθους.

Σημαντική διαδικασία στη λειτουργία του οργάνου αποτελεί η αντιστοίχιση των ερεθισμάτων που δέχεται ο αισθητήρας από το μετρούμενο φυσικό μέγεθος σε ενδείξεις κατάλληλων μονάδων στην κλίμακα του οργάνου, λόγω της μεταβολής ενός άλλου γνωστού φυσικού μεγέθους με καλά καθορισμένο τρόπο. Γενικά μας ενδιαφέρει να έχουμε καλή γνώση του αισθητήρα και του φυσικού μηχανισμού ανίχνευσης. Τα χαρακτηριστικά των αισθητήρων παίζουν μεγάλο ρόλο στην συνολική απόδοση του μετρητικού μας συστήματος. Διακρίνουμε τα στατικά και τα δυναμικά χαρακτηριστικά (Καραδήμας, 2017).

Στατικά χαρακτηριστικά

- **Ακρίβεια**. Είναι η απόκλιση της μετρούμενης τιμής από την πραγματική τιμή και καθορίζεται από το μέγιστο σφάλμα που προκύπτει από πολλές μετρήσεις για το ίδιο φυσικό μέγεθος με σταθερές τις παραμέτρους του πειράματος. Το εύρος της κατανομής των μετρήσεων για το ίδιο ερέθισμα σχετίζεται με την ακρίβεια.
- **Πιστότητα**. Η ικανότητα του αισθητήρα να δίνει ένα αποτέλεσμα που να ανταποκρίνεται στην πραγματική τιμή (τιμή αναφοράς). Υπάρχει περίπτωση ένας αισθητήρας να έχει μεγάλη ακρίβεια, αλλά κακή πιστότητα το οποίο σημαίνει ότι υπάρχει συστηματικό σφάλμα. Για να έχουμε καλή πιστότητα στις μετρήσεις πρέπει το εύρος λειτουργίας του αισθητήρα να βρίσκεται όσο το δυνατόν πιο κοντά στο εύρος των μετρούμενων τιμών.
- **Στατιστικό σφάλμα**. Είναι η ποσοστιαία διαφορά της πραγματικής τιμής του φυσικού μεγέθους από την μετρούμενη, ως προς την πραγματική. $[\Delta(A_{μετρ.} - A_{αναφ.}) / A_{αναφ.}] \cdot 100$.
- **Υστέρηση.** Η διαφορά της τιμής εξόδου για το ίδιο ερέθισμα εισόδου, σε δύο αντίθετες φάσεις μέτρησης. Για παράδειγμα ένας αισθητήρας παραμόρφωσης μπορεί να δώσει μια τιμή εξόδου για μια τιμή της εξωτερικά επιβαλλόμενης πίεσης, κατά τη φάση που αυτή αυξάνεται και να δώσει διαφορετική τιμή εξόδου για την ίδια τιμή της πίεσης στη φάση που αυτή μειώνεται.
- **Θόρυβος.** Ο θόρυβος αφορά την επίδραση διάφορων ενδογενών (σχετιζόμενες με το μηχανισμό ανίχνευσης του εκάστοτε αισθητήρα) και εξωγενών (περιβαλλοντικών) παραγόντων στο τελικό αποτέλεσμα μιας μέτρησης. Ένας συνήθης τρόπος αναγνώρισής

του σε μια μέτρηση είναι οι διακυμάνσεις στην έξοδο, χωρίς ο αισθητήρας να έχει δεχθεί κάποιο ερέθισμα. Η απομόνωση του θορύβου από τη μέτρηση δεν είναι πάντοτε εφικτή, ωστόσο η καλή γνώση του φυσικού μηχανισμού ανίχνευσης του αισθητήρα και των παραμέτρων που επιδρούν στη μέτρηση μπορούν να οδηγήσουν σε μια καλή εκτίμηση για τη συνιστώσα του (μέτρο και κυματομορφή) στο μετρούμενο σήμα.

- **Σταθερότητα.** Η έννοια της σταθερότητας δηλώνει την ικανότητα ενός αισθητήρα να δίνει την ίδια τιμή εξόδου στο ίδιο περιβάλλον μέτρησης (σταθερή είσοδος), όταν πραγματοποιείται μέτρηση για μεγάλο χρονικό.
- **Γραμμικότητα**. Υπάρχει μια περιοχή στο πεδίο μετρούμενων τιμών, όπου οι τιμές εξόδου έχουν ένα σταθερό συντελεστή αναλογίας συναρτήσει των τιμών της μετρούμενης φυσικής ποσότητας, ενώ έξω από αυτά τα όρια ο αισθητήρας συμπεριφέρεται μή γραμμικά. Είναι σημαντικό να λειτουργεί ο αισθητήρας στη γραμμική περιοχή του, καθώς είναι πιο προβλέψιμη η συμπεριφορά του και η μελέτη των καταγεγραμμένων δεδομένων γίνεται απλούστερη.
- **Εύρος.** Ένας αισθητήρας μπορεί να δίνει ένα σήμα εξόδου για μια μεγάλη γκάμα τιμών εισόδου. Ωστόσο, ως εύρος ορίζουμε την απόλυτη τιμή μεταξύ της ελάχιστης και της μέγιστης τιμής ενός φυσικού μεγέθους για τις οποίες οι αντίστοιχες τιμές εξόδου είναι αξιόπιστες και μεγαλύτερες από το σφάλμα του αισθητήρα.
- **Ευαισθησία.** Η ευαισθησία εκφράζεται από το μέγεθος της μεταβολής του σήματος εισόδου ως προς το αντίστοιχο ερέθισμα του αισθητήρα. Συγκεκριμένα δίνεται από το λόγο

$$\text{ευαισθησία} = \frac{\text{Μέγιστη τιμή εξόδου } - \text{ Ελάχιστη τιμή εξόδου}}{\text{Μέγιστη τιμή εισόδου } - \text{ Ελάχιστη τιμή εισόδου}}$$

και οι μονάδες της εξαρτώνται από τις μονάδες του εξερχόμενου προς του μετρούμενου μεγέθους. Ένας ηλεκτρονικός αισθητήρας θερμοκρασίας για παράδειγμα θα έχει μονάδες ευαισθησίας mV/^{o}C. Αν ο αισθητήρας λειτουργεί στη γραμμική περιοχή, τότε η ευαισθησία θα είναι σταθερή.

- **Διακριτική ικανότητα.** Είναι η ελάχιστη ποσότητα που μπορεί να ανιχνεύσει ένας αισθητήρας. Όσο μεγαλύτερη είναι τόσο μεγαλύτερο είναι το πλήθος των μεταβολών που μπορεί να ανιχνεύσει ένας αισθητήρας σε ένα εύρος τιμών εισόδου.
- **Χρόνος απόκρισης**. Ο χρόνος απόκρισης σχετίζεται με το χρόνο που χρειάζεται ένας αισθητήρας ώστε να φτάσει στην επιθυμητή τελική τιμή εξόδου για δεδομένη είσοδο. Τυπικά οι κατασκευαστές ορίζουν μία χρονική παράμετρο, όπου εκφράζεται ο χρόνος (sec, msec, μsec ανάλογα τον αισθητήρα) για τιμή εξόδου πολύ κοντά στην ιδανική τιμή. Για

παράδειγμα, η απόκριση $t90 = 5sec$ σημαίνει ότι σε $5sec$ ο αισθητήρας θα έχει φτάσει στο 90% της τελικής του τιμής).

- **Μέσος χρόνος ζωής.** Ο κάθε κατασκευαστής δίνει μια εκτίμηση για το χρόνο ζωής του αισθητήρα, δηλαδή για το χρονικό διάστημα που θα μπορεί να λειτουργεί σωστά και σύμφωνα με τις προδιαγραφές του. Με την πάροδο του χρόνου ένας αισθητήρας ενδέχεται να χάσει την ακρίβεια ή την ευαισθησία του λόγω κάποιας σταδιακής μικρής μεταβολής κάποιων ιδιοτήτων του. Έτσι ο αισθητήρας είτε αντικαθίσταται είτε υπόκειται σε ένα πλήθος διεργασιών από τον κατασκευαστή, αναβαθμονομείται και τοποθετείται στην διάταξη.
- **Νεκρή ζώνη.** Σε αυτή την περιοχή λειτουργίας του αισθητήρα δεν υπάρχει καμία μεταβολή στην έξοδο και είναι δυνατή η ανίχνευση του φυσικού μεγέθους. Συνήθως ένας αισθητήρας βρίσκεται σε αυτήν την περιοχή κατά την αύξηση της μετρούμενη ποσότητας, μέχρι να γίνει αντιληπτή. Η τιμή της εισόδου όπου η μεταβολή εξόδου είναι μή μηδενική ονομάζεται τιμή κατωφλίου.
- **Ολίσθηση.** Ως ολίσθηση ορίζουμε την αργή μεταβολή του σήματος εξόδου, όταν δεν υπάρχει μεταβολή στην φυσική ποσότητα στην είσοδο. Οι παράγοντες που οδηγούν σε ολίσθηση είναι συνήθως η υγρασία και η θερμοκρασία, οι οποίοι μπορούν να τροποποιήσουν τις φυσικές ή ηλεκτρικές ιδιότητες ενός υλικού [9].

Δυναμικά χαρακτηριστικά

Τα δυναμικά χαρακτηριστικά σχετίζονται με την συμπεριφορά ενός αισθητήρα κατά την διαρκή μεταβολή του σήματος εισόδου και διαφέρουν από τα στατικά χαρακτηριστικά, όπου βρίσκεται σε σταθερή κατάσταση. Η δυναμική απόκριση ενός αισθητήρα δεν μπορεί να περιγραφεί με γενικευμένες παρατηρήσεις και εξαρτάται από το είδος του αισθητήρα και το πεδίο εφαρμογής του. Βασικός λόγος της αλλαγής των στατικών του στοιχείων σε δυναμικά ερεθίσματα είναι οι διάφορες μορφές αδράνειας που δημιουργούνται, όπως ενεργειακή, μάζας και άλλες.

1.3. Θόρυβος στα ηλεκτρονικά όργανα

Όπως αναφέρουμε και παραπάνω, ως θόρυβο στα ηλεκτρονικά ορίζουμε μια ανεπιθύμητη μορφή ενέργειας, η οποία προστίθεται στο τελικό επιθυμητό σήμα. Αποτελεί ένα σοβαρό ζήτημα τόσο στα αναλογικά όσο και στα ψηφιακά ηλεκτρονικά και χρήζει ιδιαίτερης σημασίας σε όλα τα στάδια υλοποίησης ενός κυκλώματος. Μπορεί να εμφανιστεί με διάφορους τρόπους ανάλογα την εφαρμογή, όπως για παράδειγμα τα 'χιόνια' στις αναλογικές τηλεοράσεις ή ένας υψίσυχνος άτονος ήχος σε ένα ραδιόφωνο. Και στις δύο αυτές περιπτώσεις ο θόρυβος προέρχεται κυρίως από διαταραχές κατά την μετάδοση πληροφορίας από τον πομπό στο δέκτη. Αντίστοιχα στην ψηφιακή μετάδοση μπορεί να οδηγήσει σε απώλεια μερικής πληροφορίας. Αν μια τυχαία κυματομορφή προστεθεί σε ένα λογικό – παλμικό – σήμα ενδέχεται να αντιστρέψει τη λογική κατάσταση στην έξοδο.

Γενικότερα περιορίζει το ελάχιστο μέγεθος του σήματος που μπορεί να επεξεργαστεί ένα κύκλωμα, το οποίο είναι πολύ σημαντικό, ειδικά όταν επιζητούμε μεγάλη ακρίβεια όπως η μέτρηση ενός αισθητήρα. Συνήθως κατηγοριοποιείται με βάση τον τύπο, την πηγή και την επίδραση με το δέκτη. Ένα άλλο κριτήριο είναι η προέλευση του θορύβου και διακρίνουμε τις εξωτερικές και τις εσωτερικές πηγές θορύβου [2].

1.4. Ηλεκτροχημικοί αισθητήρες αντίστασης με αγώγιμα νανοσωματίδια

Οι ηλεκτροχημικοί αισθητήρες αντίστασης με αγώγιμα νανοσωματίδια έχουν παρεμφερή δομή και λειτουργία με τους κλασσικούς ηλεκτροχημικούς αισθητήρες με την διαφορά ότι αντί για τη χρήση ως ευαίσθητου υμενίου κάποιου συνθετικού πολυμερούς, το πολυμερές επικάλυψης είναι μη αγώγιμο. Η αγωγιμότητα των διατάξεων αυτών επιτυγχάνεται με την εναπόθεση ενός στρώματος μεταλλικών νανοσωματιδίων ανάμεσα και πάνω από τα ηλεκτρόδια. Το στρώμα των νανοσωματιδίων μπορεί να θεωρηθεί ως ένα φύλλο δύο διαστάσεων και διέπεται από τους μηχανισμούς αγωγιμότητας βάσει της γεωμετρία που διαθέτουν, της επιφανειακής τους κατανομής, της πυκνότητάς τους, του μεγέθους τους, του βαθμού σύζευξης των νανοσωματιδίων και άλλων παραμέτρων [3], [4], [11]. Η ανίχνευση του αναλύτη επιτυγχάνεται μέσω της αλλαγής στην αντίσταση της διάταξης, η οποία οφείλεται στη μετατόπιση των αγώγιμων νανοσωματιδίων και στην αλλαγή της ενέργειας των φορέων τους, όταν το πολυμερές διογκώνεται λόγω της αλληλεπίδρασής του με τον αναλύτη. Αντί για την επικάλυψη της διάταξης με κάποιο πολυμερές, μπορεί να γίνει επικάλυψη των νανοσωματιδίων με κάποιο ηλεκτροστατικό περίβλημα ή με κατάλληλα χημικά μόρια,

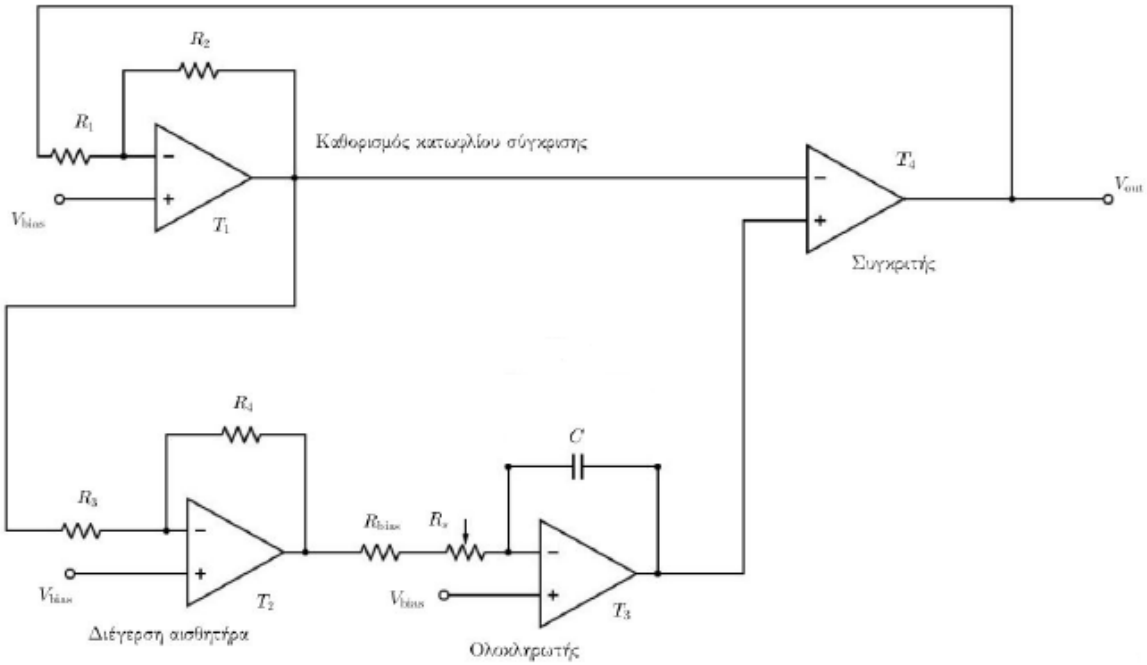
οδηγώντας στο ίδιο αποτέλεσμα, αλλά με διαφορετικές ιδιότητες του ευαίσθητου υμενίου [5]. Τα πιο διαδεδομένα νανοσωματίδια με ηλεκτροστατικό περίβλημα είναι αυτά που παράγονται με την μέθοδο του Turkevich 135 και περιβάλλονται από κιτρικό οξύ.

Οι ηλεκτροχημικοί αισθητήρες αντίστασης με αγώγιμα νανοσωματίδια εκμεταλλευόμενοι τις ιδιότητες της νανοκλίμακας, πετυχαίνουν την μείωση των ορίων ανίχνευσης, την σμίκρυνση των διατάξεων, την αύξηση της ευαισθησίας των αισθητήρων κ.α. Η απόκριση των αισθητήρων στους αναλύτες συνδέεται κατά κύριο λόγο με τα φυσικά, χημικά και γεωμετρικά χαρακτηριστικά του ευαίσθητου υμενίου που τελικά επιλέγεται να χρησιμοποιηθεί [6].

Στην παρούσα εργασία έχει γίνει η χρήση ηλεκτροχημικών αισθητήρων αντίστασης με αγώγιμα νανοσωματίδια. Ως βάση του αισθητήρα χρησιμοποιήθηκε το πυρίτιο [7]. Τα ηλεκτρόδια κατασκευάστηκαν με την τεχνική της εξάχνωσης μετάλλου μέσω θερμιονικής εκπομπής ηλεκτρονίων και ως υλικό επιλέχθηκε ο χρυσός [8]. Στη συνέχεια έγινε εναπόθεση νανοσωματιδίων πλατίνας διαμέτρου περίπου 5 νανομέτρων στην επιφάνεια των διατάξεων, με την τεχνική της μαγνητικής ιοντοβολής. Ενώ ως ευαίσθητο υμένιο χρησιμοποιήθηκε το πολυμερές PIBMA, το οποίο εναποτέθηκε κατόπιν περιστροφής των διατάξεων γύρω από τον εαυτό του [12].

2. Ανάλυση του Αναλογικού και Ψηφιακού Κυκλώματος

2.1. Συνοπτική Ανάλυση Κυκλώματος



Εικόνα 1. Αναλογικό Κύκλωμα

Στην παραπάνω σχηματική απεικόνιση βλέπουμε το αναλογικό κύκλωμα το οποίο μετράει την τιμή της αντίστασης του αισθητήρα. Πιο συγκεκριμένα, παράγει ως έξοδο μία τετραγωνική κυματομορφή, η περίοδος της οποίας είναι ανάλογη της αντίστασης του αισθητήρα μας, που απεικονίζεται στο κύκλωμα ως Rs.

Η ψηφιακή μορφή της εξόδου μας επιτρέπει να παρακάμψουμε την χρήση ενός μετατροπέα A to D και να αποφύγουμε το επιπλέον κόστος και την προσθήκη σφαλμάτων και θορύβου.

Ο *T1* δημιουργεί έναν τετραγωνικό παλμό, ο οποίος λειτουργεί σαν κατώφλι σύγκρισης, στον συγκριτή *T4* (ο οποίος θα δώσει ως έξοδο ψηφιακή κυματομορφή 0-5V). Ο *T2* παράγει επίσης μία τετραγωνική κυματομορφή. Με άλλα λόγια, τροποποιεί το πλάτος της τετραγωνικής κυματομορφής που παράγει ο *T1* και με αυτήν τροφοδοτεί τον αισθητήρα προς μέτρηση. Ο

T3 παράγει έναν τριγωνικό παλμό, ο οποίος στον συγκριτή *T4* θα συγκριθεί με την τάση κατωφλίου του *T1*, για την δημιουργία της παλμοσειράς εξόδου.

Πιο συγκεκριμένα, η έξοδος του *T1* (ενισχυτής σε αναστρέφουσα συνδεσμολογία) είναι:

$$u_{T_{1,o}} = \left(1 + \frac{R_2}{R_1}\right) V_{\text{bias}} - \frac{R_2}{R_1} u_{\text{out}} \tag{2.1}$$

Αυτή η τάση είναι μετέπειτα η είσοδος στον *T2*, ο οποίος είναι παρομοίος ενισχυτής σε αναστρέφουσα συνδεσμολογία, άρα η έξοδος του *T2*:

$$u_{T_{2,o}} = \left(1 + \frac{R_4}{R_3}\right) V_{\text{bias}} - \frac{R_4}{R_3} u_{T_{1,o}} \tag{2.2}$$

Αντικαθιστώντας στην παραπάνω σχέση την τάση V1, έχουμε:

$$u_{T_{2,o}} = \left(1 - \frac{R_2}{R_1}\frac{R_4}{R_3}\right) V_{\text{bias}} + \frac{R_2}{R_1}\frac{R_4}{R_3} u_{\text{out}} \tag{2.3}$$

Ο *T3* είναι ένας ολοκληρωτής, η έξοδος του οποίου είναι:

$$u_{T_{3,o}}(t) = u_{T_{3,o}}(0) + \frac{1}{(R_s + R_{\text{bias}})C}\frac{R_2}{R_1}\frac{R_4}{R_3}(V_{\text{bias}} - u_{\text{out}})t \tag{2.4}$$

Όσον αφορά στον συγκριτή, αλλαγή στην τάση εξόδου θα πραγματοποιηθεί όταν η τάση στον θετικό και αρνητικό ακροδέκτη εξισωθούν.

Η αντίσταση *Rbias*, είναι για να εξομαλύνει το σήμα , σε περίπτωση που η *Rs* είναι μηδέν και επέλθει ψαλιδισμός και παραμόρφωση της κυματομορφής.

Τέλος, αναφέρουμε ότι η τάση *Vbias* προκύπτει από έναν διαιρέτη τάσης, δύο αντιστάσεων *Rdiv1,2*, τιμής 1ΜΩ, η οποία μειώνει στο ήμισυ την τάση *Vdd*, τιμής 5V. Άρα η *Vbias* είναι της τάξεως των 2.5V [13].

2.2. Περίοδος Κυματομορφής Εξόδου

Η περίοδος της κυματομορφής εξόδου είναι:

$$T = \frac{V_{DD}^2}{V_{\text{bias}}(V_{DD} - V_{\text{bias}})} \frac{(R_s + R_{\text{bias}})C}{|G_{T_2}|} \tag{2.5}$$

Για *Vbias* = *Vdd*/2, η σχέση γίνεται:

$$T = \frac{4(R_s + R_{\text{bias}})C}{|G_2|} \tag{2.6}$$

2.3. Προδιαγραφές Αισθητήρων Προς Μέτρηση

Στην προηγούμενη εξίσωση αναφέραμε την τελική μορφή της περιόδου της κυματομορφής εξόδου. Εάν υποθέσουμε την μικρότερη αντίσταση που μπορούμε να μετρήσουμε με ακρίβεια *Rs,min* και την μέγιστη αντίσταση που μπορεί να μετρηθεί, ως *Rs,max*, τότε η προηγούμενη σχέση γίνεται:

$$\frac{4R_{s,\min}}{|G_2|} \leq \frac{T}{C} \leq \frac{4R_{s,\max}}{|G_2|} \tag{2.7}$$

Το κέρδος G2 επιλέγεται ώστε η μέγιστη δυνατή τάση εισόδου σε κάθε τελεστικό ενισχυτή να μην τον οδηγεί σε περιοχή κορεσμού. Για να διασφαλιστεί αυτό, φροντίζουμε να ικανοποιούνται οι κάτωθι εξισώσεις.

Για τους ενισχυτές *T1* και *T3*, καθώς και για τον *T2*, πρέπει να ικανοποιούνται οι αντίστοιχες σχέσεις:

$$\frac{R_2}{R_1} \leq 1 \quad , \quad \frac{R_4}{R_3}\frac{R_2}{R_1} \leq 1 \tag{2.8}$$

Καθώς οι εξισώσεις που περιγράφουν το κέρδος έχουν επιλεγεί, η μόνη παράμετρος που καθορίζει το εύρος των αντιστάσεων προς μέτρηση είναι η τιμή του πυκνωτή *C*. Από την προηγούμενη ανισότητα του κλάσματος *T/C*, προκύπτουν δύο συνθήκες που πρέπει να ικανοποιούνται ταυτόχρονα:

$$C \geq \frac{|G_2|T_{\min}}{4R_{s,\min}} \quad \text{και} \quad C \leq \frac{|G_2|T_{\max}}{4R_{s,\max}} \tag{2.9}$$

Στην πρώτη εξίσωση επιλέγεται η περίοδος που θα αντιστοιχεί στην ελάχιστη αντίσταση που θα μπορεί να μετρηθεί από το σύστημα. Στη δεύτερη εξίσωση καθορίζεται ο απαιτούμενος χρόνος μιας περιόδου στο ανώτατο όριο του εύρους μετρήσεων, το οποίο συνίσταται στη μέγιστη συχνότητα που μπορεί να αποδώσει το αναλογικό κύλωμα.

Από τις δύο εξισώσεις παρατηρούμε ότι, όσο αυξάνεται η *Rs*, αυξάνεται και η περίοδος της κυματομορφής. Προφανώς αυτό συνεπάγεται πως υπάρχει και ένα άνω όριο στην αντίσταση που μπορεί να μετρηθεί, λόγω των περιορισμών του επεξεργαστή.

Επίσης, όσο μεγαλύτερη είναι η τιμή του πυκνωτή *C*, τόσο μικρότερη αβεβαιότητα θα έχει το σύστημα (η οποία μπορεί να είναι ως και μερικές δεκάδες Ω). Αναλόγως, η επιλογή ενός μικρού πυκνωτή μπορεί να οδηγήσει σε μέτρηση αντιστάσεων της τάξεως ΜΩ, σε πολύ μικρό χρόνο (της τάξεως μsec), αλλά με μεγάλο σφάλμα.

Στην περίπτωσή μας, με τις τιμές των στοιχείων που έχουμε επιλέξει, μπορούμε να ‘διαβάσουμε’ αντιστάσεις που ανήκουν στο εύρος 300kΩ < *Rs* < 1ΜΩ.

Παρακάτω παρατίθενται οι τιμές των παθητικών στοιχείων του αναλογικού κυκλώματος.

Στοιχεία	Τιμή	
R_1	9.81	$k\Omega$
R_2	1.98	$k\Omega$
R_3	9.88	$k\Omega$
R_4	34.9	$k\Omega$
R_{bias}	98.7	$k\Omega$
C	400	pF
$R_{\text{div},1}$	1.00	$M\Omega$
$R_{\text{div},2}$	1.00	$M\Omega$

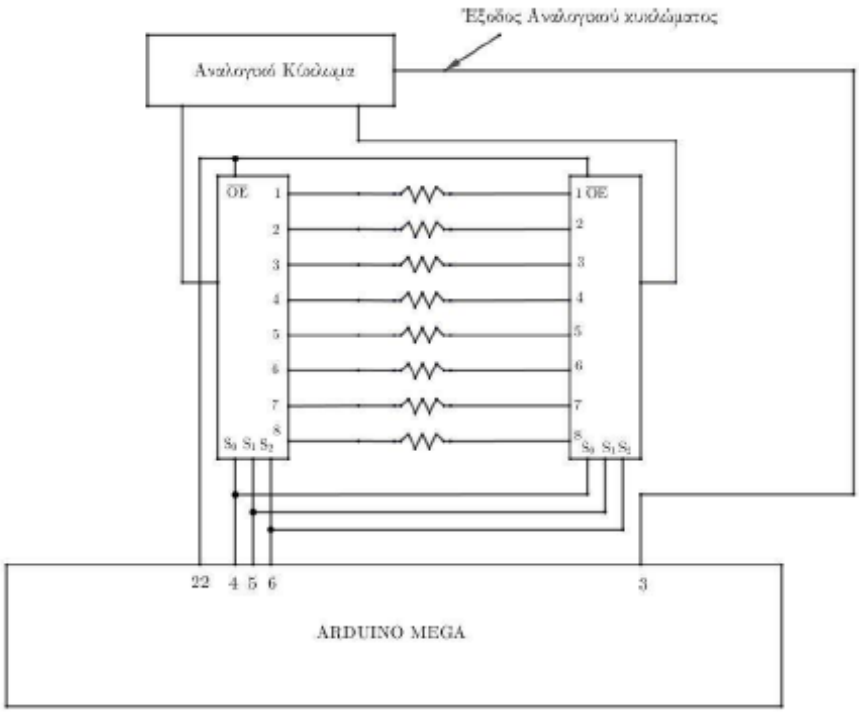
Πίνακας 1. Τιμές παθητικών στοιχείων

2.4. Μήτρα Αισθητήρων

Με το συγκεκριμένο κύκλωμα που περιγράφηκε, μπορούμε να μετρήσουμε την τιμή μίας αντίστασης *Rs*. Θα ήταν καλό να μπορούσαμε να μετρήσουμε πολλαπλούς αισθητήρες. Αντί λοιπόν να πολλαπλασιάσουμε τον αριθμό των κυκλωμάτων, για να ισούται με τον αριθμό των αισθητήρων προς μέτρηση, μπορούμε να κρατήσουμε το αναλογικό κύκλωμα ως έχει και να προσθέσουμε έναν συνδυασμό πολυπλεκτών, οι οποίοι με τον κατάλληλο έλεγχο από τον

μικροελεγκτή, θα οδηγούν το αναλογικό κύκλωμα στον συγκεκριμένο αισθητήρα που επιθυμούμε να μετρήσουμε.

Παρακάτω παρατίθεται η μορφή της μήτρας αισθητήρων:



Εικόνα 2. Σχηματικό διάγραμμα πολυπλεξίας

Η μορφή του αναλογικού κυκλώματος παραμένει η ίδια και απλά παρεμβαίνει η συστοιχία πολυπλεκτών, ανάμεσα στο αναλογικό κύκλωμα και τον μικροελεγκτή.

Κάθε αισθητήρας της μήτρας συνδέεται με τις αντίστοιχες εξόδους των δύο πολυπλεκτών. Οι εν λόγω αισθητήρες αντιστοιχούν στην αντίσταση *Rs* του αναλογικού κυκλώματος. Η μία έξοδος τοποθετείται στο άκρο της αντίστασης *Rbias* και η δεύτερη έξοδος στον αρνητικό ακροδέκτη εισόδου του ολοκληρωτή.

Από τον μικροελεγκτή μπορούμε μέσω κώδικα να επιλέξουμε εάν οι πολυπλέκτες θα δίνουν έξοδο, μέσω του σήματος Output Enable (OE στο σχηματικό διάγραμμα). Επίσης ελέγχουμε τα σήματα *S0, S1, S2*, με τα οποία επιλέγουμε την επιθυμητή αντίσταση προς μέτρηση, η οποία συνδέεται με το αναλογικό κύκλωμα, ενώ οι υπόλοιπες αντιστάσεις απομονώνονται και δεν επηρεάζουν την μέτρηση.

Στον παρακάτω πίνακα παρατίθενται τα σήματα ελέγχου (τα οποία είναι κοινά και για τους δύο πολυπλέκτες), με τα οποία καθορίζουμε την αντίσταση προς μέτρηση.

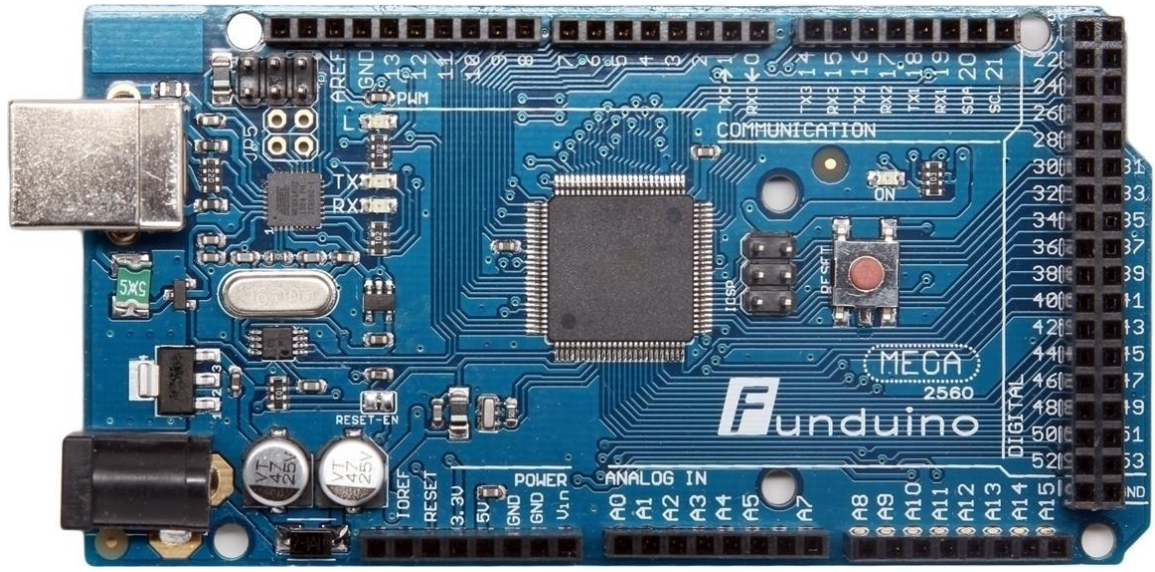
Αντίσταση	Σήμα επιλογής
R_0	000
R_1	001
R_2	010
R_3	011
R_4	100
R_5	101
R_6	110
R_7	111

Πίνακας 2. Σήματα ελέγχου πολυπλεκτών

3. Ανάλυση του Υπάρχοντος Κώδικα του Μικροελεγκτή

3.1. Πλατφόρμα Εκτέλεσης Κώδικα

Ο συγκεκριμένος κώδικας ενσωματωμένου συστήματος είναι σχεδιασμένος για να τρέχει σε πλατφόρμα Funduino Mega, το οποίο είναι μία παραλλαγή του Arduino Mega.



Εικόνα 3. Η αναπτυξιακή πλατφόρμα Funduino Mega

Το Arduino Mega 2560 είναι μία αναπτυξιακή πλακέτα που βασίζεται στον μικροελεγκτή ATmega2560. Διαθέτει 54 ψηφιακούς ακροδέκτες εισόδου / εξόδου (εκ των οποίων 15 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 16 αναλογικές εισόδους, 4 UART (hardware σειριακές θύρες), ταλαντωτή κρυστάλλου 16 MHz, διεπαφή USB, υποδοχή τροφοδοσίας, ICSP header, και ένα κουμπί reset. Για την τροφοδοσία του, μπορούμε να το συνδέσουμε σε έναν υπολογιστή με καλώδιο USB ή να το τροφοδοτήσουμε με έναν προσαρμογέα AC-DC ή να χρησιμοποιήσουμε μπαταρία. Μπορούμε επίσης να επεκτείνουμε τις δυνατότητές του, χρησιμοποιώντας μία προσαρμοζόμενη πλακέτα, η οποία συνήθως αποκαλείται shield.

3.2. Φιλοσοφία του κώδικα

Όπως προαναφέρθηκε, η έξοδος του αναλογικού συστήματος είναι ένας τετραγωνικός παλμός, η περίοδος του οποίου μας δίνει την τιμή της αντιστάσεως του ασιθητήρα.

Ο τρόπος με τον οποίο ο μικροελεγκτής μπορεί να βρεί την περίοδο μιας τετραγωνικής κυματομορφής είναι μέσω της μέτρησης του χρονικού διαστήματος ανάμεσα στις δύο θετικές ακμές του τετραγωνικού παλμού. Η ακριβής ανίχνευση των θετικών ακμών πραγματοποιείται με ενεργοποίηση διακοπών (interrupts) στον μικροελεγκτή.

Συγκεκριμένα, όταν καταφτάσει ο πρώτος θετικός παλμός, ενεργοποιείται η ρουτίνα της διακοπής, η οποία με τη σειρά της ενεργοποιεί ένα μετρητή. Ο μετρητής σταματά όταν καταφτάσει ο δεύτερος θετικός παλμός και αφού ενεργοποιηθεί η επόμενη διακοπή.

Ο μετρητής όμως είναι μεγέθους 16 bits, πράγμα το οποίο σημείνει πως μπορεί να πραγματοποιήσει μετρήσεις απο 0 εώς (2^16-1). Σε περίπτωση που υπάρχει υπερχείλιση του μετρητή, ενεργοποιείται μία ακόμα ρουτίνα διακοπής, η οποία οδηγεί στην αύξηση ενός δεύτερου μετρητή που διατηρεί το πλήθος των υπερχειλίσεων.

Είναι προφανές ότι, εάν γνωρίζουμε την τιμή του μετρητή και άρα, τον αριθμό των υπερχειλίσεων, μπορούμε να βρούμε το χρονικό διάστημα που μεσολαβεί ανάμεσα στους δύο παλμούς και συνεπώς, την περίοδο του παλμού.

Η τιμή του μετρητή δίνεται απο την εξίσωση:

$$N_{\text{total}} = N_{\text{overflows}} \cdot 2^{16} + N_{\text{timer}} \tag{3.1}$$

και η περίοδος του τετραγωνικού παλμού δίνεται από τον τύπο:

$$T = N_{\text{total}} \cdot T_{\text{timer}} \tag{3.2}$$

Για να αυξήσουμε όμως την ακρίβεια της μέτρησης, αντί για δύο θετικές ακμές, μετράμε k δείγματα και στο τέλος υπολογίζουμε το μέσο όρο. Οπότε, η προηγούμενη σχέση γίνεται:

$$T = \frac{N_{\text{total}} \cdot T_{\text{timer}}}{k-1} \tag{3.3}$$

3.3. Λειτουργία της setup()

```
void setup() {
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), count_time_of_k_pulses, RISING);

  pinMode(OE_BAR_PIN, OUTPUT);
  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);

  TIMER1_init();

  Serial.begin(BAUD_RATE);
  while (! Serial);
}
```

Στη συνάρτηση σεταρίσματος του μικροελεγκτή πραγματοποιούμε όλες τις απαραίτητες αρχικοποιήσεις προκειμένου να λειτουργήσει σωστά το σύστημα. Αυτές περιλαμβάνουν την ανάθεση της συνάρτησης count_time_of_k_pulses() ως interrupt, όταν ανιχνευθεί θετική ακμή στο pin INTERRUPT_PIN, τον ορισμό των pins που ελέγχουν τους πολυπλέκτες ως pins εξόδου και τέλος, την αρχικοποίηση του timer και της σειριακής επικοινωνίας, με την συχνότητα που έχουμε ορίσει στη global μεταβλητή BAUD_RATE.

3.4. Λειτουργία της loop()

```
void loop() {
  if (Serial.available()) {
    /***** DEFINE MEASUREMENT MODE *****/
    String input = Serial.readString();
    if (input[0] == 'r' && interrupts_enabled == false) {
      if (input[1] == 'f') {
        full_mode = true;
        sensor_coord = 0;
      } else
        sensor_coord = input[2] - '0';

      select_sensor();
```

Εάν υπάρχουν διαθέσιμα σειριακά δεδομένα και εάν η εντολή που δίνεται στο arduino είναι προς εκκίνηση ανάγνωσης, ο μικροελεγκτής εκτελεί τον παρακάτω κώδικα:

```
      /***** SET VARIABLES FOR DUMMY MEASUREMENT *****/
      number_of_samples = 10;

      pulse_counter = 0;
      measurement_counter = 0;

      TIMER1_clear();

      interrupts_enabled = true;
    }
  }
```

Στην περίπτωση αυτή μηδενίζονται όλες οι μεταβλητές των μετρητών της κάθε επιμέρους μέτρησης και ενεργοποιείται το interrupts_enabled flag έτσι ώστε το interrupt να είναι σε θέση να ξεκινήσει, σε περίπτωση που οι συνθήκες το ωθήσουν. Θέτουμε number_of_samples = 10, καθώς η συγκεκριμένη μέτρηση η οποία αποτελεί dummy measurement, είναι η πρώτη και παρουσιάστηκε προβληματική όπως αναφέρθηκε παραπάνω. Με τον τρόπο αυτό, έχοντας λίγα δείγματα ολοκληρώνεται γρήγορα η εν λόγω μέτρηση.

Στο σημείο αυτό τερματίζεται το πρωτο block of code, το οποίο εκτελείται την πρώτη φορά που θα σταλεί σειριακά η εντολή στο arduino για την έναρξη της ανάγνωσης.

```
  if (measurement_finished) {
    measurement_counter++;
```

Μόλις εκτελεστεί και ολοκληρωθεί η λειτουργία του interrupt, το measurement counter θα αυξηθεί.

```
  ticks = (overflows * 65536) + TCNT1;
  period = (ticks * 0.0625) / (number_of_samples - 1);

  if (measurement_counter == 1) {
    /***** SET VARIABLES FOR MEASUREMENT *****/
    number_of_samples = 500;

    pulse_counter = 0;

    TIMER1_clear();

    interrupts_enabled = true;
  }
```

Εάν το measurement counter είναι 1, αυτό συνεπάγεται πως το dummy measurement έχει εκτελεστεί με επιτυχία. Συνεπώς, στη συνέχεια πραγματοποιούμε μία επιπλέον μέτρηση με 500 samples για την ορθή εκτέλεση.

```
else {
  TIMER1_print_results();

  if (full_mode) {
    sensor_coord++;

    select_sensor();
```

Το παραπάνω συμβαίνει σε όλες τις περιπτώσεις, δηλαδή σε όλες τις καταστάσεις λειτουργίας (modes). Εάν όμως έχουμε την περίπτωση full mode, τότε θα πρέπει να προχωρήσουμε στον επόμενο αισθητήρα, και να ξεκινήσουμε από την αρχή τη διαδικασία dummy measurement, regular measurement για κάθε έναν από τους υπόλοιπους 7 αισθητήρες.

```
/***** SET VARIABLES FOR DUMMY MEASUREMENT *****/
number_of_samples = 10;

pulse_counter = 0;
measurement_counter = 0;

TIMER1_clear();

interrupts_enabled = true;
```

Σε αυτό το σημείο ξεκινάει η dummy μέτρηση.

```
        if (sensor_coord == NUM_OF_SENSORS) {
          full_mode = false;
          interrupts_enabled = false;
        }
      }
    }
    measurement_finished = false;
  }
}
```

Παραπάνω παρουσιάζεται η συνθήκη που έχουν μετρηθεί όλοι οι αισθητήρες (και οι 8) και το τέλος του measurement mode. Παρατηρούμε πως πραγματοποιείται μία μόνο μέτρηση για

κάθε αισθητήρα και στη συνέχεια τερματίζει. Ο κώδικας στην τωρινή του μορφή δεν εχει τη δυνατότητα για πολλαπλές μετρήσεις – κάτι το οποίο είναι απαραίτητο.

Σε περίπτωση που δεν έχουν ολοκληρωθεί οι μετρήσεις για όλους τους αισθητήρες, τοτε εκτελείται ξανά η loop() και επειδή δεν παρέχουμε σειριακά δεδομένα στο σύστημα, θα παρακαμφθεί το τμήμα του κώδικα που εκτελεί τη dummy μέτρηση (με τα 10 samples), γιατί αυτό το κομμάτι εκτελείται μόνο όταν λάβουμε σειριακά δεδομένα. Αυτή η συμπεριφορά είναι επιθυμητή, καθώς μόλις πραγματοποιήσαμε μέτρηση με 10 samples στο νέο αισθητήρα. Στη συνέχεια το σύστημα θα προχωρήσει σε κανονική μέτρηση με τα 500 samples.

3.5. Ρουτίνες Διακοπής

```
void count_time_of_k_pulses() {
  if (interrupts_enabled) {
    pulse_counter++;

    if (pulse_counter == 1)
      TIMER1_start();

    if (pulse_counter == number_of_samples) {
      TIMER1_stop();

      measurement_finished = true;
      interrupts_enabled = false;
    }
  }
}
```

Παραπάνω βλέπουμε τη ρουτίνα διακοπής (interrupt routine).
Ουσιαστικά, όταν ενεργοποιηθεί το συγκεκριμένο interrupt routine εκκινεί το μετρητή (timer). Όταν ολοκληρωθεί η μέτρηση (συμπληρώνοντας τον αριθμό των δειγμάτων που θέλουμε), σταματάει τον timer, και απενεργοποιεί τα interrupts.

```
ISR (TIMER1_OVF_vect) {
  overflows++;
}
```

Όταν ο timer υπερχειλίσει, τρέχει αυτό το δευτερεύον interrupt το οποίο αυξάνει την τιμή ενός μετρητή, ο οποίος μας υποδεικνύει πόσες φορές είχαμε υπερχείλιση.

3.6. Συναρτήσεις Μετρητή

```
void TIMER1_init() {
  TCCR1A  = 0;
  TCCR1B  = 0;
  TIMSK1  = 0;
  TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);
}

void TIMER1_start() {
  TIMSK1 |= (1 << TOIE1);
}

void TIMER1_stop() {
  TIMSK1 = 0;
}

void TIMER1_clear() {
  TCNT1     = 0;
  overflows = 0;
}
```

Οι παραπάνω τέσσερεις συναρτήσεις αφορούν στην αρχικοποίηση, εκκίνηση, τερματισμό και καθαρισμό των timers.

Εν τέλει, απεδείχθη πως οι εν λόγω συναρτήσεις αποτελούν την πηγή ενός εκ των προβλημάτων. Το συγκεκριμένο πρόβλημα αφορά στην αδυναμία λήψης σωστών αποτελεσμάτων στην πρώτη μέτρηση. Υπενθυμίζουμε ότι για το λόγο αυτό, πραγματοποιούμε μία μικρή πρώτη μέτρηση με 10 samples και μόλις ολοκληρωθεί, πραγματοποιούμε την κανονική μέτρηση με 500 samples.

3.7. Λοιπές Συναρτήσεις

```
void select_sensor() {
  digitalWrite(OE_BAR_PIN, LOW);
  digitalWrite(S2_PIN, ((sensor_coord & 4) == 4) ? HIGH : LOW );
  digitalWrite(S1_PIN, ((sensor_coord & 2) == 2) ? HIGH : LOW );
  digitalWrite(S0_PIN, ((sensor_coord & 1) == 1) ? HIGH : LOW );
}
```

Με αυτή την συνάρτηση ορίζουμε τους κατάλληλους παλμούς που θα παραμετροποιήσουν τα τσιπάκια των πολυπλεκτών, για την επιλογή του κατάλληλου αισθητήρα, ετσι ώστε το αναλογικό κύκλωμα να έρχεται σε επαφή μόνο με τον επιθυμητό αισθητήρα.

```
void TIMER1_print_results() {
  Serial.println("---------------------------------------");
  Serial.print("SENSOR COORDINATE         = ");
  Serial.println(sensor_coord);

  Serial.print("NUMBER OF PULSES MEASURED = ");
  Serial.println(pulse_counter);

  Serial.print("TIMER1 VALUE              = ");
  Serial.println(TCNT1);

  Serial.print("TIMER1 OVERFLOWS          = ");
  Serial.println(overflows);

  Serial.print("TICKS                     = ");
  Serial.println(ticks);

  Serial.print("MEASURED PERIOD           = ");
  Serial.print(period);
  Serial.println(" US");

  Serial.print("MEASURED RESISTANCE       = ");

  double resistanse = ((period * GAIN_VALUE * 1000) / (4 * CAPACITOR_VALUE)) - R_BIAS_VALUE; //kΩ
  Serial.print(resistanse);

  Serial.println(" kΩ");

  Serial.println("---------------------------------------");
}
```

Αυτή η συνάρτηση μας εκτυπώνει τα δεδομένα που χρειαζόμαστε – και παραπάνω.

3.8. Προβλήματα – Παραλείψεις Κώδικα

1. Το πρόβλημα που απαιτεί δύο μετρήσεις. Η πρώτη μέτρηση – για κάθε αισθητήρα είναι πάντα λάθος. Για αυτό τον λόγο, πραγματοποιούμε δύο μετρήσεις, τη dummy με λίγα δείγματα για γρήγορο (λανθασμένο) αποτέλεσμα και την κανονική μέτρηση με 500 δείγματα.

2. Δεν υπάρχει η δυνατότητα για συνεχόμενες μετρήσεις. Φυσικό επακόλουθο, λείπει η λειτουργία τερματισμού της διαρκούς μέτρησης, ενός ή όλων των αισθητήρων.

3. Σε περίπτωση ανοικτοκυκλώματος, παραδείγματος χάρην, όταν ένας αισθητήρας δεν είναι συνδεδεμένος και έχουμε δρομολογήσει τη μέτρησή του, τότε το πρόγραμμα είναι στην κατάσταση οπου είναι ενεργοποιημένα τα timer interrupts, και αναμένει διαρκώς παλμούς – για να αυξηθεί το pulse counter (μέχρι να φτάσει τον αριθμό των samples). Επειδή το pulse counter δεν αυξάνεται και προφανώς δεν θα φτάσει ποτέ τον αριθμό των samples, το πρόγραμμα δεν μπορεί να εξέλθει της διαδικασίας μέτρησης του ανοιχτοκυκλωμένου αισθητήρα, άρα έχει κολλήσει σε εκείνο το σημείο ατέρμονα.

4. Ανάπτυξη Κώδικα Μικροελεγκτή

4.1. Διόρθωση του προβλήματος των πολλαπλών μετρήσεων

Όπως έχει προαναφερθεί, ο αρχικός κώδικας ήταν προβληματικός και το αποτέλεσμα της πρώτης μέτρησης για κάθε μετρητή ήταν λανθασμένο. Για το λόγο αυτό, οι συγγραφείς του κώδικα είχαν προβεί σε πρόχειρες λύσεις, όπως να πραγματοποιούν την πρώτη μέτρηση με λίγα δείγματα και να αγνοούν το αποτέλεσμά της και την δεύτερη, κύρια μέτρηση με ικανοποιητικό αριθμό δειγμάτων.

Ένα ακόμα επακόλουθο ήταν πως, η παράκαμψη του προβλήματος καθιστούσε τον κώδικα ιδιαίτερα περίπλοκο και περιπεπλεγμένο. Αυτό δυσχέραινε την εργασία μας με τον κώδικα και τη δυνατότητα να εξελιχθεί. Πιο συγκεκριμένα, λόγω του ότι στον αρχικό κώδικα υπήρχε αρκετό state μέσω πολλών flag, ήταν δύσκολο να κατανοήσουμε την κατάσταση του συστήματος ανα πάσα στιγμή. Η επέκταση και περαιτέρω ανάπτυξη του κώδικα αύξανε την πολυπλοκότητα σε μεγάλο βαθμό.

```
void loop() {
  if (Serial.available()) {
    /***** DEFINE MEASUREMENT MODE *****/
    String input = Serial.readString();
    if (input[0] == 'r' && interrupts_enabled == false) {
      if (input[1] == 'f') {
        full_mode = true;
        sensor_coord = 0;
      } else
        sensor_coord = input[2] - '0';

      select_sensor();

      /***** SET VARIABLES FOR DUMMY MEASUREMENT *****/
      number_of_samples = 10;

      pulse_counter = 0;
      measurement_counter = 0;

      TIMER1_clear();

      interrupts_enabled = true;
    }
  }
```

Όπως έχει αναλυθεί, παρατηρούμε πως οι dummy μετρήσεις γίνονταν μέσα στο κομμάτι που ρυθμίζει την σειριακή επικοινωνία. Μπορεί αυτό να ήταν αποδεκτό για την απλή περίπτωση που εφαρμοζόταν ως τώρα, αλλα εάν θέλαμε να επεκτείνουμε τον κώδικα, αυτό το κομμάτι έπρεπε να καθαριστεί και να διορθωθεί.

Οι συγγραφείς της αρχικής μελέτης αδυνατούσαν να κατανοήσουν γιατί συμβαίνει αυτό το φαινόμενο και το απέδιδαν σε πρόβλημα του hardware.

Για την αντιμετώπιση του συγκεκριμένου προβλήματος, αρχικά δήλωσα όλες τις μεταβλητές των οποίων η τιμή αλλάζει εντός της ρουτίνας διακοπής ως volatile. Στην αντίθετη περίπτωση, ο compiler ενδέχεται να επιχειρήσει βελτιστοποίηση στη μεταβλητή και να χρησιμοποιήσει μια cached τιμή αντί της πραγματικής.

```
//variables that get changed inside the ISR -
volatile int pulse_counter = 0;
volatile boolean interrupts_enabled = false;
volatile boolean measurement_finished = false;
```

Παρ'όλα αυτά, αυτό δεν ήταν αρκετό για να ξεπεραστεί το πρόβλημα. Για το λόγο αύτο, στη συνέχεια προχωρήσαμε στη μελέτη των συναρτήσεων των interrupt timers. Υπενθυμίζουμε τις συναρτήσεις αυτές:

```
/***** TIMER1 FUNCTIONS *****/

void TIMER1_init() {
  TCCR1A  = 0;
  TCCR1B  = 0;
  TIMSK1  = 0;
  TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);
}

void TIMER1_start() {
  TIMSK1 |= (1 << TOIE1);
}

void TIMER1_stop() {
  TIMSK1 = 0;
}

void TIMER1_clear() {
  TCNT1     = 0;
  overflows = 0;
}
```

Έπειτα απο ανάγνωση του datasheet του επεξεργαστή και έρευνα σε σχετικά forum, τροποποιήσαμε τις συναρτήσεις αυτές, εκτός απο τη συνάρτηση που καθαρίζει το μετρητή. Παρακάτω παρατίθεται ο νέος κώδικας.

```
void TIMER1_init()
{
  TCCR1A  = 0;
  TCCR1B  = 0;
  TIMSK1 |= (1 << TOIE1);
}

void TIMER1_start()
{
   TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);
}

void TIMER1_stop()
{
  TCCR1B  = 0;
}

void TIMER1_clear()
{
  TCNT1     = 0;
  overflows = 0;
}
```

Μετά απο αυτές τις αλλαγές, το πρόβλημα λύθηκε. Πλέον κάθε μεμονωμένη μέτρηση είναι ακριβής. Άρα δε χρειάζεται να προβαίνουμε σε dummy μετρήσεις. Συνέπεια αυτού, ο κώδικας γίνεται πολύ πιο απλός και ευανάγνωστος.

Ας δούμε λοιπόν πως απλοποιήθηκε το full mode στο πρόγραμμα:

```
    if (full_mode) {
      sensor_coord++;

      select_sensor();

      /***** SET VARIABLES FOR DUMMY MEASUREMENT
      number_of_samples = 10;

      pulse_counter = 0;
      measurement_counter = 0;

      TIMER1_clear();

      interrupts_enabled = true;

      if (sensor_coord == NUM_OF_SENSORS) {
        full_mode = false;
        interrupts_enabled = false;
      }
    }
  }
  measurement_finished = false;
}
```

Παλαιότερα σε κάθε full mode έπρεπε πρώτα να λάβουν χώρα οι dummy μετρήσεις, στη συνέχεια η κανονική με 500 samples, και τέλος το loop στον επόμενο αισθητήρα και να επαναληφθούν τα δύο βήματα. Πλέον η μέτρηση γίνεται απ' ευθείας.

```
if (full_mode)
{
  sensor_coord++;
  select_sensor();

  if (sensor_coord == NUM_OF_SENSORS)
  {

    sensor_coord = 0;
  }
}
```

4.2. Περίπτωση Ανοιχτοκυκλωμένου Αισθητήρα

Όταν μία αντίσταση είναι ανοιχτοκυκλωμένη, δηλαδή ένα pin είναι στον αέρα, ο κώδικας σταματάει.

Αυτό συμβαίνει διότι, το πρόγραμμα αναμένει το σήμα του παλμού. Το σήμα ωστόσο δε θα καταφτάσει ποτέ, καθώς το κύκλωμα δεν είναι σε σύνδεση, άρα δεν παράγεται κάποιος παλμός απο το αναλογικό κύκλωμα. Κατά συνέπεια, η ρουτίνα διακοπής δεν ενεργοποιείται, άρα το πρόγραμμα παραμένει ατέρμονα στην κατάσταση αναμονής μίας διακοπής.

Η λύση του προβλήματος αυτού έγκειται στον προσδιορισμό της χρονικής διάρκειας κατά την οποία δεν σημειώθηκαν διακοπές και να "μαρκάρουμε" αυτήν την αντίσταση ως ανοιχτοκυκλωμένη. Στην περίπτωση που μία αντίσταση είναι ανοιχτοκυκλωμένη, θα προχωράμε σε αλλαγή της αντίστοιχης σειριακήςς εκτύπωση για τη συγκεκριμένη αντίσταση.

```
unsigned long timestamp = 0;
unsigned long measurement_timeout = 5000;
boolean sensor_corruption[8]; //each elem
boolean ISR_unavailable = false; //this f
```

Το timestamp είναι ο μετρητής της χρονικής διάρκειας από την έναρξη και καθ'όλη τη διάρκεια αναμονής του παλμού για έναν συγκεκριμένο αισθητήρα. Το timeout είναι το τελικό όριο που θα αναμένει το πρόγραμμά μας για παλμό. Στην εν λόγω περίπτωση το έχουμε ορίσει στα πέντε δευτερόλεπτα.

Το sensor_corruption[8] είναι ένας πίνακας απο boolean τιμές που αντιπροσωπεύει την κατάσταση κάθε αισθητήρα. Εάν μία καταχώρηση είναι false, τότε ο αντίστοιχος αισθητήρας δουλεύει (με την έννοια ότι είναι ορθά συνδεδεμένος, το αναλογικό κύκλωμα λειτουργεί και στέλνει παλμό στον μικροελεγκτή). Αντίστοιχα, εάν η καταχώρηση είναι true, τότε ο αντίστοιχος αισθητήρας είναι ανοιχτοκυκλωμένος.

Το ISR_unavailable είναι ένα flag με αρχική τιμή false, το οποίο αλλάζει όταν το interrupt routine δεν εκτελεστεί, επειδή δεν δεν καταφτάνει σήμα απο το αναλογικό κύκλωμα, καθώς ο αισθητήρας είναι ανοιχτοκυκλωμένος. Όταν βρισκόμαστε σε μία τέτοια κατάσταση πρέπει να το γνωρίζουμε, προκειμένου να μπορούμε να τη διαχειριστούμε. Η μεταβλητή ISR_unavailable φροντίζει γι' αυτό.

Όπως αρχικοποιήσαμε το ISR_unavailable flag σε false, ετσι πρέπει να αρχικοποιηθούν και όλες οι τιμές του boolean πίνακα sensor_corruption σε false. Αυτό γίνεται μέσα στην setup() συνάρτηση, η οποία εκτελείται όταν εκκινηθεί ο μικροελεγκτής. Αυτή η αρχικοποίηση παρουσιάζεται στην ακόλουθη εικόνα.

```
void setup()
{
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), count_time_of_k_pulses, RISING);

  digitalWrite(OE_BAR_PIN, HIGH);
  pinMode(OE_BAR_PIN, OUTPUT);
  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);

  TIMER1_init();

  Serial.begin(BAUD_RATE);
  while (! Serial);

  //Initializing all array elements to false!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
  for (int i=0; i<=7; i++)
  {
    sensor_corruption[i] = false; // false means it's working ok. true means it's open-cir
  }
}
```

Έπειτα μένει το κομμάτι οπου ανιχνεύουμε την κατάσταση ανοιχτοκυκλώματος. Ο κώδικας που ακολουθεί βρίσκεται στην κύρια loop() συνάρτηση, μετά την αρχικοποίηση της σειριακής επικοινωνίας και πρίν το κομμάτι που γίνεται η επεξεργασία και ο υπολογισμός των δεδομένων (όταν πλέον η μέτρηση έχει ολοκληρωθεί).

```
// TO DETERMINE IF A RESISTANCE IS OPEN CIRCUITED
if(interrupts_enabled && millis() - timestamp > measurement_timeout)
{
  sensor_corruption[sensor_coord] = true;
  measurement_finished = true;
  ISR_unavailable = true;
}
```

Όταν είμαστε σε mode που αναμένουμε interrupts και η χρονική διάρκεια από τη στιγμή της έναρξης της μέτρησης γίνει μεγαλύτερος απο το timeout, το οποίο είναι πέντε δευτερόλεπτα, τότε εκτελείται το ακόλουθο κομμάτι κώδικα.

Αρχικά δηλώνουμε το index του συγκεκριμένου αισθητήρα στον πίνακα boolean τιμών ως true, που υποδηλώνει πως ο συγκεκριμένος αισθητήρας είναι ανοιχτοκυκλωμένος.

Στη συνέχεια, με την γραμμή measurement_finished = true; δηλώνουμε πως η μέτρηση ολοκληρώθηκε, ετσί ώστε ο κώδικας να μπορεί να προχωρήσει στο επόμενο στάδιο.

Τέλος, η γραμμή ISR_unavailable = true είναι ένα flag, το οποίο χρησιμοποιείται ετσι ώστε να μην προβούμε σε υπολογισμούς δεδομένων (λίγες γραμμές παρακάτω), αφού δεν υπάρχουν δεδομένα – εξαιτίας ανοιχτοκυκλώματος.

4.3. Επεξεργασία Δεδομένων

Στην περίπτωση επιτυχημένης μέτρησης, στη συνέχεια εισερχόμαστε στο κομμάτι που επεξεργαζόμαστε τα δεδομένα. Η επεξεργασία δεδομένων όπως προαναφέραμε, αφορά στην περίπτωση που πραγματοποιήθηκε μία μέτρηση και έχουμε δεδομένα.

Όμως για απλούστευση, ο κώδικάς μας θα εισέρχεται πάντα στο κομμάτι επεξεργασίας δεδομένων μετα απο την ολοκλήρωση μίας μέτρησης, είτε ο αισθητήρας λειτουργεί κανονικά, είτε είναι ανοιχτοκυκλωμένος.

Παρ' όλα αυτά, εάν έχουμε περίπτωση ανοιχτοκυκλώματος, η κατάσταση θα γίνει αντιληπτή μέσω του flag ISR_unavailable, και δεν θα γίνει επεξεργασία δεδομένων, αφού έτσι κι αλλιώς δεν υπάρχουν. Δηλαδή, επεξεργασία δεδομένων θα πραγματοποιηθεί μόνο όταν ο αισθητήρας είναι κανονικά συνδεδεμένος στο αναλογικό κύκλωμα, άρα η μεταβλητή ISR_unavailable είναι false.

```
if (measurement_finished)
{

  if (!ISR_unavailable) //only make these calcs, if the ISR_unavailable is false
  {
      ticks = (overflows * 65536) + TCNT1;
      period = (ticks * 0.0625) / (number_of_samples - 1);
      sensor_corruption[sensor_coord] = false;
  }
```

Μετά το πέρας των υπολογισμών, χρησιμοποίουμε τη γραμμή sensor_corruption[sensor_coord] = false. Αυτή η γραμμή επανατοποθετεί τον πίνακα σε κατάσταση false, το οποίο υποδηλώνει ότι λειτουργεί κανονικά.

Γιατί προβαίνουμε σε μία τέτοια ενέργεια όμως, εφόσον το κομμάτι κώδικα είναι για την περίπτωση που ο αισθητήρας είναι συνδεδεμένος κανονικά; Εφ'όσον είναι κανονικά συνδεδεμένος, τότε δεν υπήρχε περίπτωση το sensor_corruption αυτού του αισθητήρα να ήταν true.

Αυτή η γραμμή εν τέλει χρησιμοποιείται εδώ για την περίπτωση που ο συγκεκριμένος αισθητήρας ήταν ανοιχτοκυκλωμένος πριν αλλά πλέον είναι κανονικά συνδεδεμένος. Στην

περίπτωση που ένας αισθητήρας ήταν ανοιχτοκυκλωμένος, η καταχώρηση γι' αυτό τον αισθητήρα στον sensor_corruption θα ήταν true. Εάν όμως μετά ο αισθητήρας συνδεόταν κανονικά στο κύκλωμα, η συγκεκριμένη καταχώρηση θα παρέμενε true. Ο λόγος ύπαρξης αυτής της γραμμής είναι για να αποκαταστήσει τη δήλωση αυτού του αισθητήρα στον πίνακα καταστάσεων, σε μορφή που αντιστοιχεί σε αισθητήρα που είναι κανονικά συνδεδεμένος.

```
TIMER1_print_results();
ISR_unavailable = false; //reset ISR_unavailable
if (full_mode)
{
  sensor_coord++;
  select_sensor();

  if (sensor_coord == NUM_OF_SENSORS)
  {

    sensor_coord = 0; //back from the beginning

  }
```

Ανεξάρτητα με το εάν ο αισθητήρας μας ήταν ανοιχτοκυκλωμένος η μη, το επόμενο βήμα είναι να καλέσουμε την συνάρτηση η οποία πραγματοποιεί εκτύπωση των δεδομένων. Έπειτα επαναφέρουμε την τιμή της μεταβλητής ISR_unavailable σε false, έτσι ώστε να προετοιμαστεί για την επόμενη μέτρηση. Σε διαφορετική περίπτωση, στις επόμενες μετρήσεις δεν θα λάμβαναν χώρα υπολογισμοί ακόμα και αν οι αισθητήρες δεν ήταν ανοιχτοκυκλωμένοι.

Το επόμενο βήμα είναι κοινό με την προηγούμενη έκδοση του κώδικα, και εφαρμόζεται σε περίπτωση που έχει επιλεγεί full mode λειτουργίας, δηλαδή μέτρηση όλων των αισθητήρων. Σε αυτή την περίπτωση, το πρόγραμμα θα αυξήσει την τιμή (index) του αισθητήρα προς μέτρηση κατά μία μονάδα και θα καλέσει τη συνάρτηση που ρυθμίζει τους πολλυπλέκτες, για να επιλεγεί ο κατάλληλος αισθητήρας απο το αναλογικό κύκλωμα. Τέλος, σε περίπτωση που ξεπεράσουμε το διαθέσιμο αριθμό αισθητήρων, μηδενίζουμε την τιμή του index του αισθητήρα και ξεκινάμε από την αρχή.

Πρέπει να αναφέρουμε ότι, επειδή πλέον υπάρχει η δυνατότητα αναγνώρισης ανοιχτοκυκλωμένου (μη συνδεδεμένου) αισθητήρα, κάτι που η πρώτη έκδοση του κώδικα δεν είχε την ικανότητα να το αντιληφθεί, πρέπει αντιστοίχως να τροποποιήσουμε και τη συνάρτηση που φροντίζει για τις εκτυπώσεις των δεδομένων.

Σε περίπτωση λοιπόν που η συνάρτηση εντοπίσει πως ο πίνακας sensor_corruption, για τον συγκεκριμένο αισθητήρα είναι true, και άρα έχουμε ανοιχτοκυκλωμένο αισθητήρα, τυπώνει το αντίστοιχο μήνυμα στην σειριακή έξοδο.

```
void TIMER1_print_results()
{
  Serial.println("----------------------------------------");
  Serial.print("SENSOR COORDINATE         = ");
  Serial.println(sensor_coord);

  //Added the if-else statement, to cover the possibility of open circuited resistance !!!!!!!!!!!!!
  if (sensor_corruption[sensor_coord])
  {
    Serial.println("This resistance is open circuited!!!");
  }
  else
  {
    Serial.print("NUMBER OF PULSES MEASURED = ");
    Serial.println(pulse_counter);

    Serial.print("TIMER1 VALUE              = ");
    Serial.println(TCNT1);

    Serial.print("TIMER1 OVERFLOWS          = ");
    Serial.println(overflows);

    Serial.print("TICKS                     = ");
    Serial.println(ticks);

    Serial.print("MEASURED PERIOD           = ");
    Serial.print(period);
    Serial.println(" US");

    Serial.print("MEASURED RESISTANCE       = ");
    double resistanse = ((period * GAIN_VALUE * 1000) / (4 * CAPACITOR_VALUE)) - R_BIAS_VALUE; //kΩ
    Serial.print(resistanse);
    Serial.println(" kΩ");

    Serial.println("----------------------------------------");
  }
}
```

4.4. Υλοποίηση συνεχών μετρήσεων

Το επόμενο βήμα είναι να δώσουμε στο πρόγραμμα τη δυνατότητα να πραγματοποιεί επαναλαμβανόμενες μετρήσεις, είτε ο χρήστης δηλώσει ένα συγκεκριμένα αισθητήρα, είτε επιλέξει να μετράει όλους τους αισθητήρες επαναλαμβανόμενα.

Μια ιδιαιτερότητα αυτής της λειτουργίας είναι πως πρέπει να είμαστε σε συνεχή επικοινωνία με το χρήστη. Για παράδειγμα, ο χρήστης πρέπει να έχει τη δυνατότητα να σταματάει τη μέτρηση με την αποστολή της αντίστοιχης εντολής.

Επίσης, όταν μετράμε έναν αισθητήρα, θέλουμε ο χρήστης να μπορεί να αλλάξει τον αισθητήρα προς μέτρηση, ή να αλλάξει mode λειτουργίας σε full mode.

Όταν είναι σε mode συνεχόμενης μέτρηση ενός αισθητήρα, είναι εύκολο ο χρήστης να αλλάξει τον αισθητήρα ή να αλλάξει το mode λειτουργίας σε full. Αυτό γίνεται γιατί στην αρχή κάθε loop πραγματοποιείται έλεγχος για σειριακή είσοδο δεδομένων (δηλαδή εντολών του χρήστη).

Όμως, στην περίπτωση που ο χρήστης είχε προεπιλέξει full mode, τότε δεν θα είναι δυνατή η μετάβαση σε μέτρηση μεμονωμένου αισθητήρα, γιατί θα πρέπει να αλλάξει το flag του full mode.

Άρα χρειάζεται να κάνουμε μια ακόμη αλλαγή, στο κομμάτι του κώδικα που αφορά τη σειριακή επικοινωνία. Σε περίπτωση που ο χρήστης επιλέξει να μετρήσει έναν αισθητήρα, αλλάζουμε το full_mode flag σε false.

```
//CODE FOR SERIAL INTERFACE (RESISTANCE SELECTION)
if (Serial.available())
{
  /***** DEFINE MEASUREMENT MODE *****/
  String input = Serial.readString();
  if (input[0] == 'r' && interrupts_enabled == false)
  {
    if (input[1] == 'f')
    {
      full_mode = true;
      sensor_coord = 0;
    }
    else
    {
      full_mode = false; //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      sensor_coord = input[2] - '0';  // r *SPACE* [0-7]
    }

    select_sensor();

    pulse_counter = 0;
    TIMER1_clear();

    interrupts_enabled = true;
```

Το επόμενο βήμα είναι η συγγραφή κώδικα για να έχουμε τη δυνατότητα να σταματάμε μία συνεχόμενη μέτρηση. Επειδή μία μέτρηση πλέον θα τρέχει συνεχόμενα, το ζητούμενο είναι να τερματιστεί με εντολή του χρήστη όποτε αυτός το επιθυμεί. Άρα ας ανατρέξουμε ξανά στο κομμάτι κώδικα που ρυθμίζει την σειριακή επικοινωνία:

```
//CODE FOR SERIAL INTERFACE (RESISTANCE SELECTION)
if (Serial.available())
{
  /***** DEFINE MEASUREMENT MODE *****/
  String input = Serial.readString();
  if (input[0] == 'r' && interrupts_enabled == false)
  {
    if (input[1] == 'f')
    {
      full_mode = true;
      sensor_coord = 0;
    }
    else
    {
      full_mode = false; //!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
      sensor_coord = input[2] - '0';  // r *SPACE* [0-7]
    }

    select_sensor();

    pulse_counter = 0;
    TIMER1_clear();

    interrupts_enabled = true;
  } //first if ends here
  else if (input[0] == 'C' || input[0] == 'c') //not sur
  {
    full_mode = false;
    measurement_finished = false;
    interrupts_enabled = false;  //If the user types 'C'
  }
} //Serial.available portion ends here
```

Όταν ο χρήστης στείλει σειριακά τον χαρακτήρα ‘C’ ή ‘c’, τότε σταματάμε ένα πιθανό full mode, με το να ρυθμίσουμε καταλλήλως το flag. Επίσης αλλάζουμε το measurement_finished σε false, ετσι ώστε να μην εισέρχεται στο κομμάτι του loop που πραγματοποιείται η επεξεργασία των μετρήσεων. Τέλος, το interrupts_enabled γίνεται false, έτσι ώστε να μην εκτελεστεί η ρουτίνα διακοπής σε περίπτωση παλμού.

Τελειώνοντας, κατά τη λήξη του measurement_finished block (δηλαδή όταν τελειώνει η επεξεργασία των δεδομένων του αισθητήρα) και πρίν το τέλος της loop(), εκτελείται ο κάτωθι κώδικας:

```
pulse_counter = 0;
TIMER1_clear();
interrupts_enabled = true;
measurement_finished = false;
timestamp = millis();  // ref
```

Αρχικά, μηδενίζουμε τον pulse_counter, δηλαδή το μετρητή παλμών για την επόμενη μέτρηση (μετράει το πλήθος των παλμών που κατέφθασαν). Στην συνέχεια, η TIMER1_clear() καθαρίζει τον μετρητή που μετράει το χρονικό διάστημα μεταξύ δύο θετικών παλμών.

Την interrupts_enabled = true τη χρησιμοποιούμε γιατί πλέον είμαστε έτοιμοι να δεχτούμε νέα μέτρηση, άρα ενεργοποιούμε τα interrupts. Η measurement_finished γίνεται false, διότι μόλις επεξεργαστήκαμε δεδομένα. Άρα είμαστε σε θεση να λάβουμε νέα δεδομένα (και όχι να προβούμε σε επεξεργασία δεδομένων).

Συνεπώς, απενεργοποιούμε το flag ετσι ώστε το loop να μην εισέρχεται στο κομμάτι που επεξεργάζεται δεδομένα και να παραμένει στο κομμάτι ελέγχου σειριακής επικοινωνίας ή σε νέο Interrupt Service Routine.

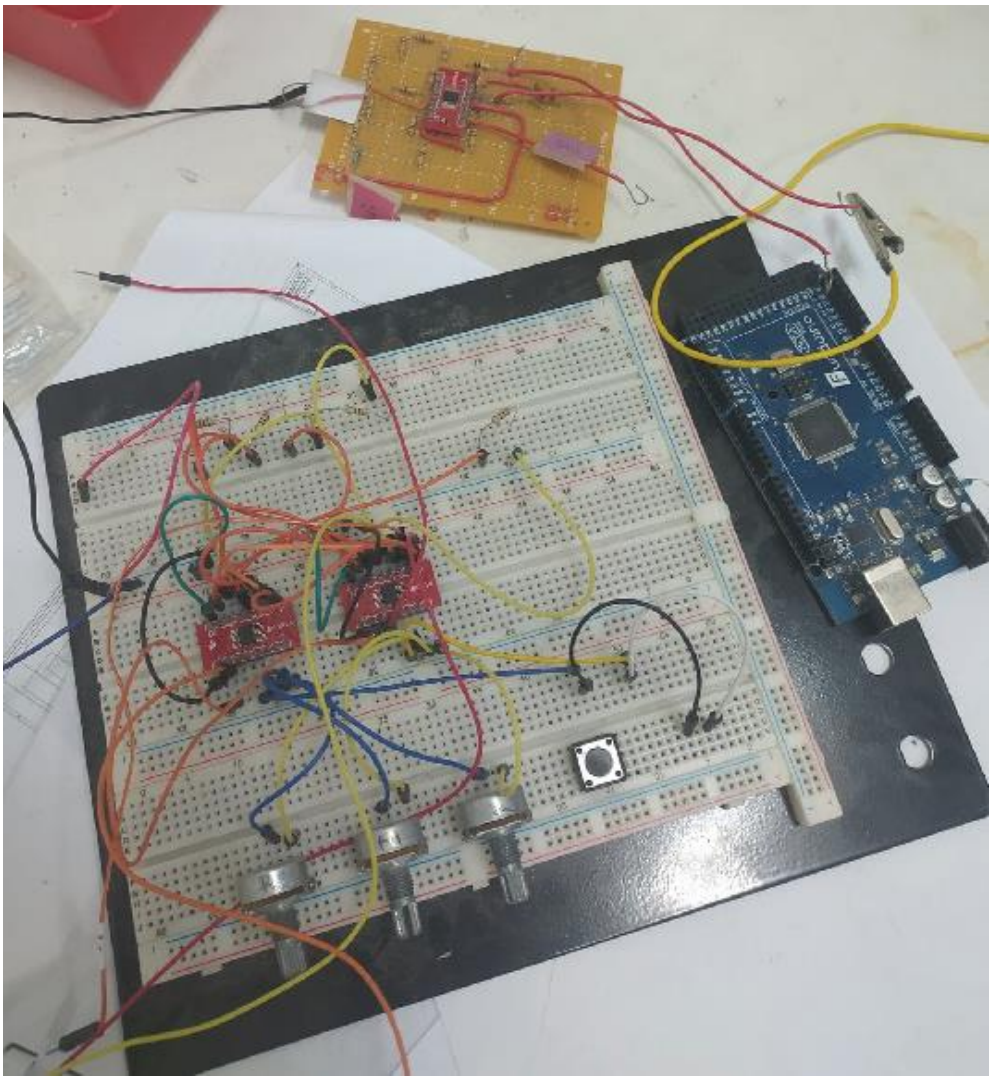
Κλείνοντας, αναλύουμε την τελευταία γραμμή της αναφοράς. Η μεταβλητή millis() μετρά τη χρονική διάρκεια απο τη στιγμή της εκκίνησης του μικροελεγκτή. Άρα, με τη γραμμή timestamp = millis(), εμείς ανανεώνουμε τη μεταβλητή timestamp με την τωρινή τιμή του χρόνου, ετσι ώστε να είμαστε σε θέση να ξανακρατάμε το χρόνο απο την αρχή της μέτρησης κάποιου αισθητήρα και στην περίπτωση που έχει περάσει κάποιο χρονικό διάστημα χωρίς να έχει εκτελεστεί η ρουτίνα διακοπής. Συνεπώς, συμπεραίνουμε οτι ο συγκεκριμένος αισθητήρας είναι ανοιχτοκυκλωμένος

5. Καταγραφή της Διαδικασίας Κατασκευής Πλακέτας

5.1. Αντίστροφη Μηχανική και Εύρεση Συνδεσμολογίας

Λόγω έλλειψης των απαραίτητων πληροφοριών για την κατασκευή της πλακέτας, έπρεπε να προβούμε στη μέθοδο της αντίστροφης μηχανικής, με σκοπό την αναζήτηση των αναγκαίων δεδομένων.

Το κύκλωμα που παρουσιάστηκε παραπάνω δεν ήταν 100% ακριβές στην υλοποίηση που είχε δοθεί αρχικά στο breadboard. Παραθέτουμε μία φωτογραφία του κυκλώματος στην αρχική του κατάσταση, υλοποιημένο με διάτρητη πλακέτα και breadboard.

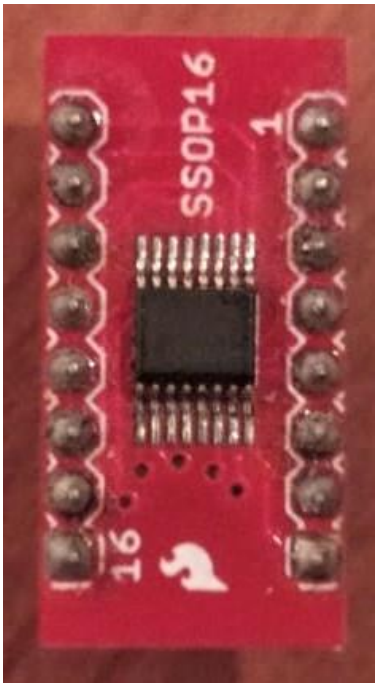


Εικόνα 4. Το κύκλωμα στην αρχική του μορφή

Ο λόγος της ύπαρξης διαφορών μεταξύ του σχηματικού διαγράματος και του υλοποιημένου κυκλώματος, είναι γιατί οι τέσσερις τελεστικοί ενισχυτές βρίσκονται μέσα σε ένα τσιπάκι, και άρα πρέπει να βρεθεί η συνδεσμολογία του ολοκληρωμένου κυκλώματος. Επιπλέον, το σχεδιάγραμμα δεν περιελάμβανε τη συνδεσμολογία των πολυπλεκτών και τον διαιρέτη τάσης που θα παρέχει τροφοδοσία στους τελεστικούς ενισχυτές.

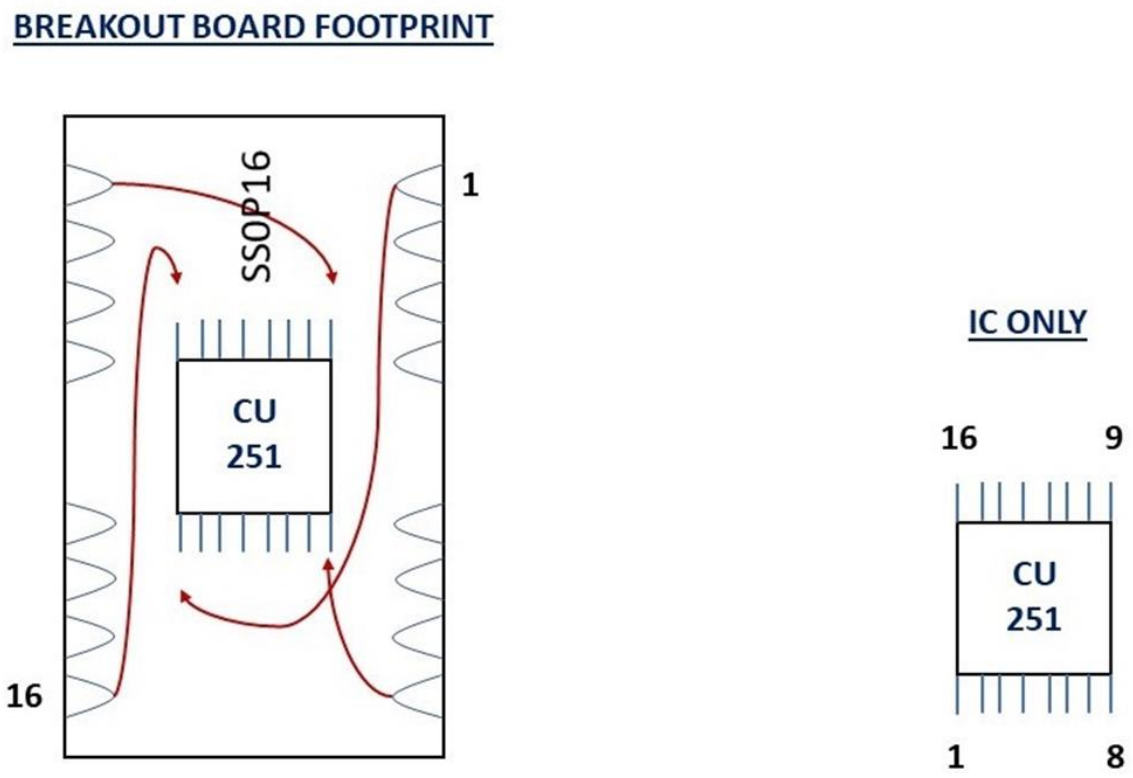
Οι πολυπλέκτες είναι και αυτοί υλοποιημένοι μέσω ολοκληρωμένων κυκλωμάτων, τα οποία είναι προσαρτημένα πάνω σε breakout boards. Έτσι, επειδή το board δεν είχε σήμανση, αρχικά επιχειρήσαμε να βρούμε τον τρόπο με τον οποίο το breakout board συνδέεται με το τσιπάκι.

Παρατίθεται η μορφή του ολοκληρωμένου του πολυπλέκτη, τοποθετημένο πάνω στο breakout board.



Εικόνα 5. Breakout board

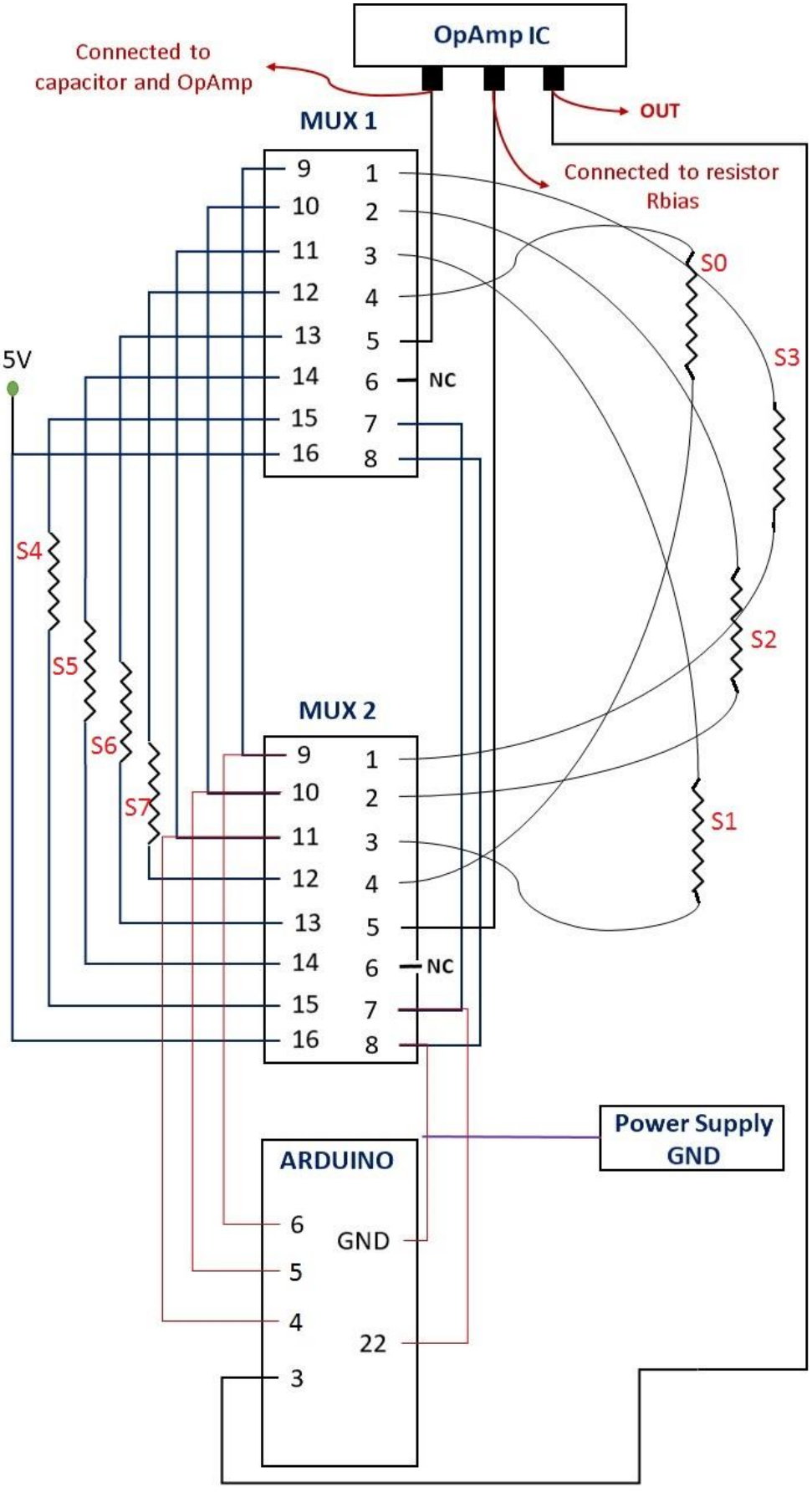
Γι’ αυτό το λόγο, χρησιμοποιήσαμε έναν ελεγκτή συνέχειας, και ένα προς ένα, ελέγξαμε τα ποδαράκια του board με το τσιπάκι. Ανάλογα με το εαν υπήρχε σύνδεση, βρήκαμε το πώς συνδέονται τα pins του ολοκληρωμένου κυκλώματος με το board. Αυτή η σχέση αποτυπώνεται στο ακόλουθο σχεδιάγραμμα.



Εικόνα 6. Αντιστοιχία footprint pins με IC pins

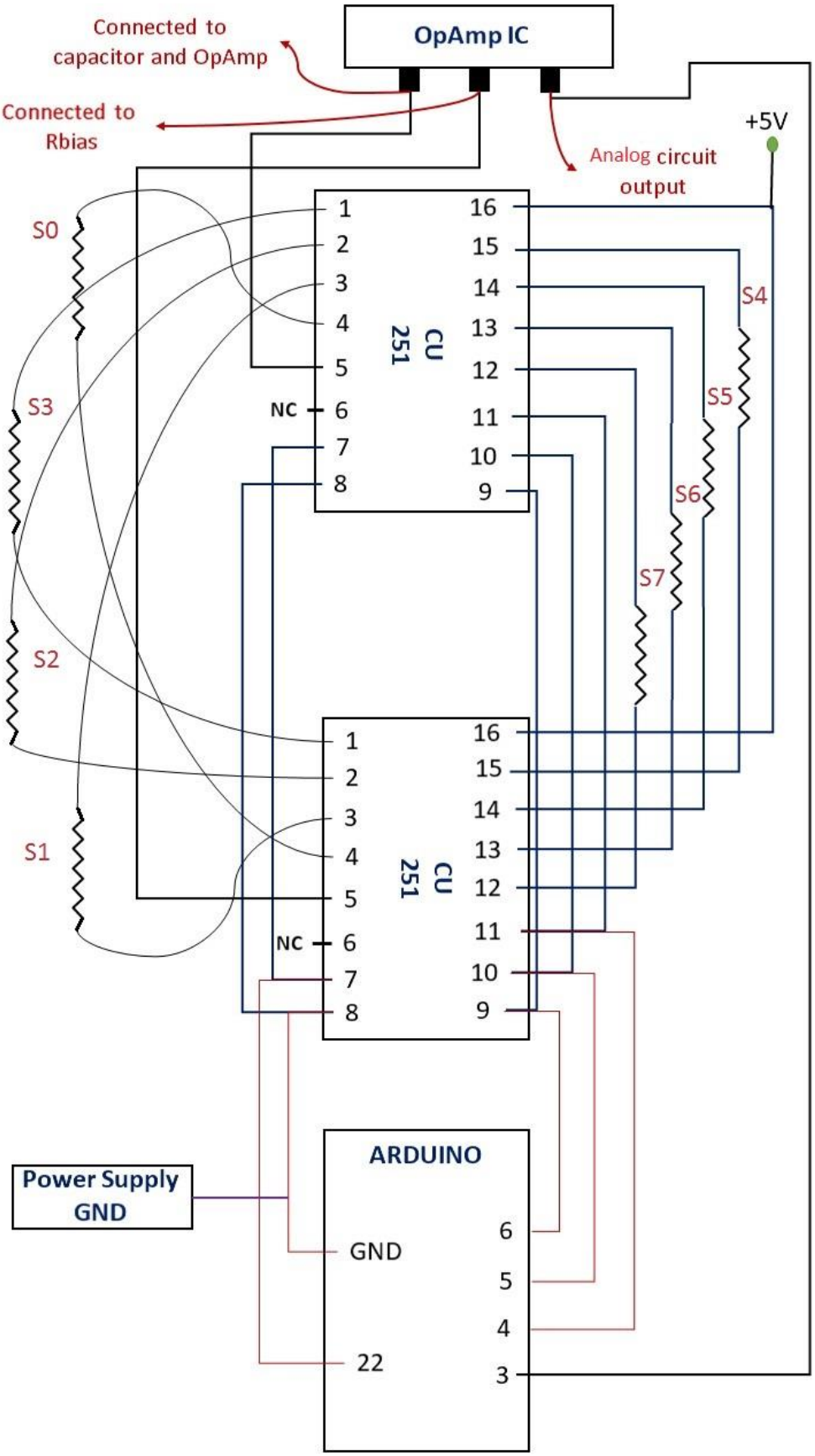
Μπορούμε να δούμε πως το pin 1 του board συνδέεται με το κάτω αριστερά pin του τσιπ και επίσης βλέπουμε τη σχέση των υπόλοιπων pins.

Έπειτα, εφαρμόζοντας την ίδια λογική της αντίστροφης μηχανικής μέσω του ελεγκτή συνέχειας, βρήκαμε τη συνδεσμολογία όλου του κυκλώματος, η οποία είναι σχεδιασμένη στο παρακάτω schematic.



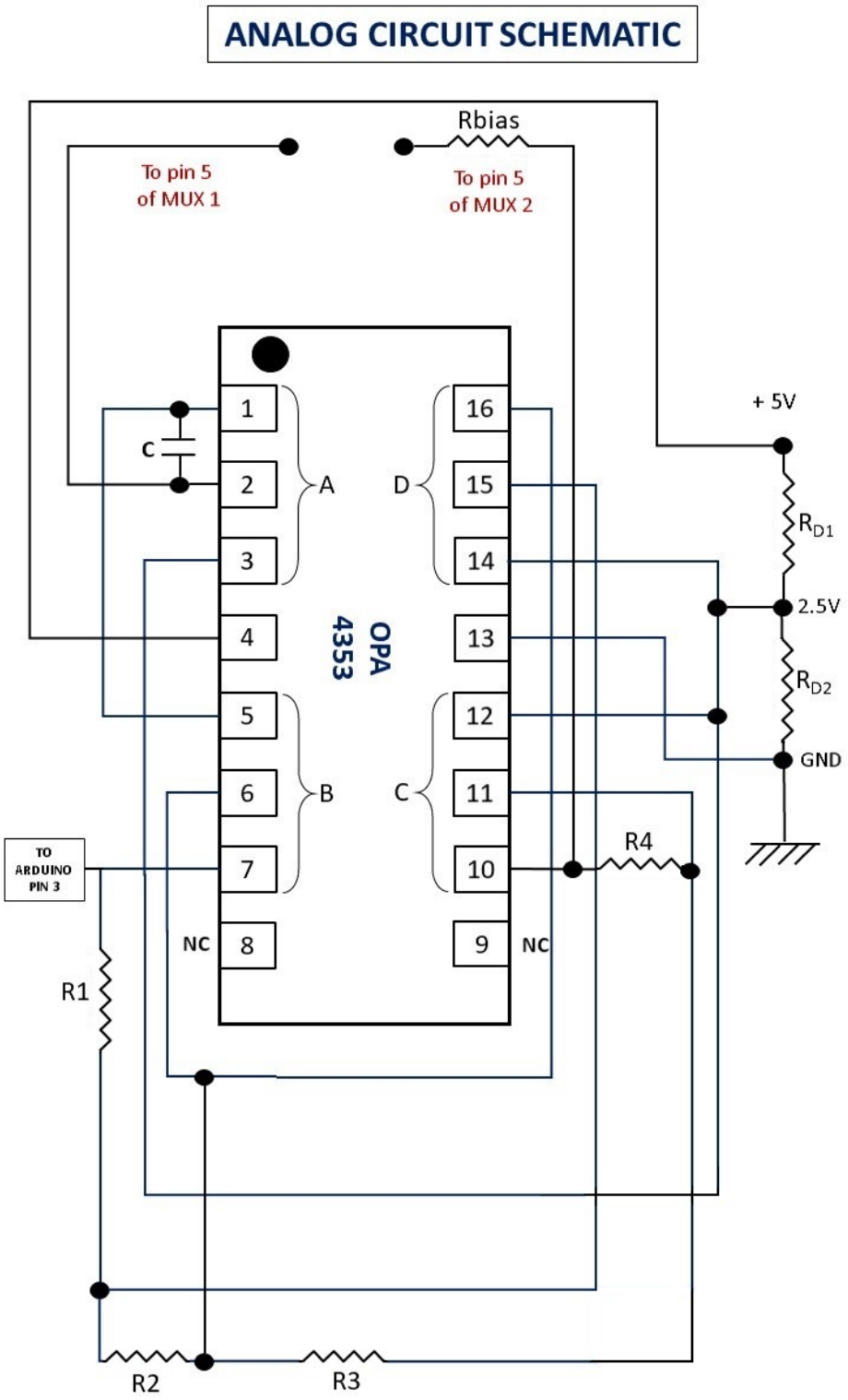
Εικόνα 7. Σχηματικό διάγραμμα πολυπλεκτών στο breakout board

Συνδιάζοντας το παραπάνω σχεδιάγραμμα του κυκλώματος με το footprint του breakout board, είμαστε σε θέση να αντιληφθούμε την συνδεσμολογία του κυκλώματος πολυπλεκτών με το μικροελεγκτή και το αναλογικό κύκλωμα, σε επίπεδο ολοκληρωμένων κυκλωμάτων. Αυτό παρουσιάζεται στο παρακάτω σχεδιάγραμμα.



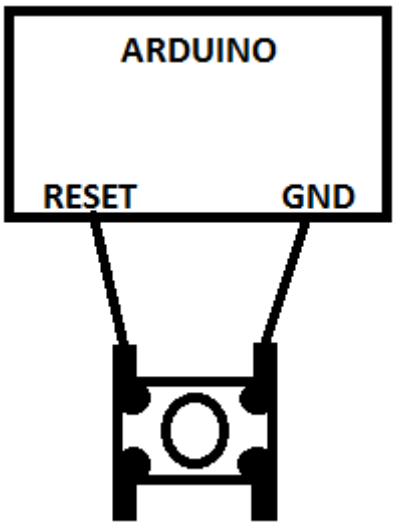
Εικόνα 8. Σχηματικό διάγραμμα πολυπλεκτών σε επίπεδο ολοκληρωμένου κυκλώματος

Η συνδεσμολογία του κυκλώματος με τους τελεστικούς ενισχυτές είναι η ακόλουθη.



Εικόνα 9. Συνδεσμολογία ολοκληρωμένου κυκλώματος τελεστικών ενισχυτών

Το κύκλωμα για το κουμπί reset που θα δημιουργηθεί είναι το εξής.



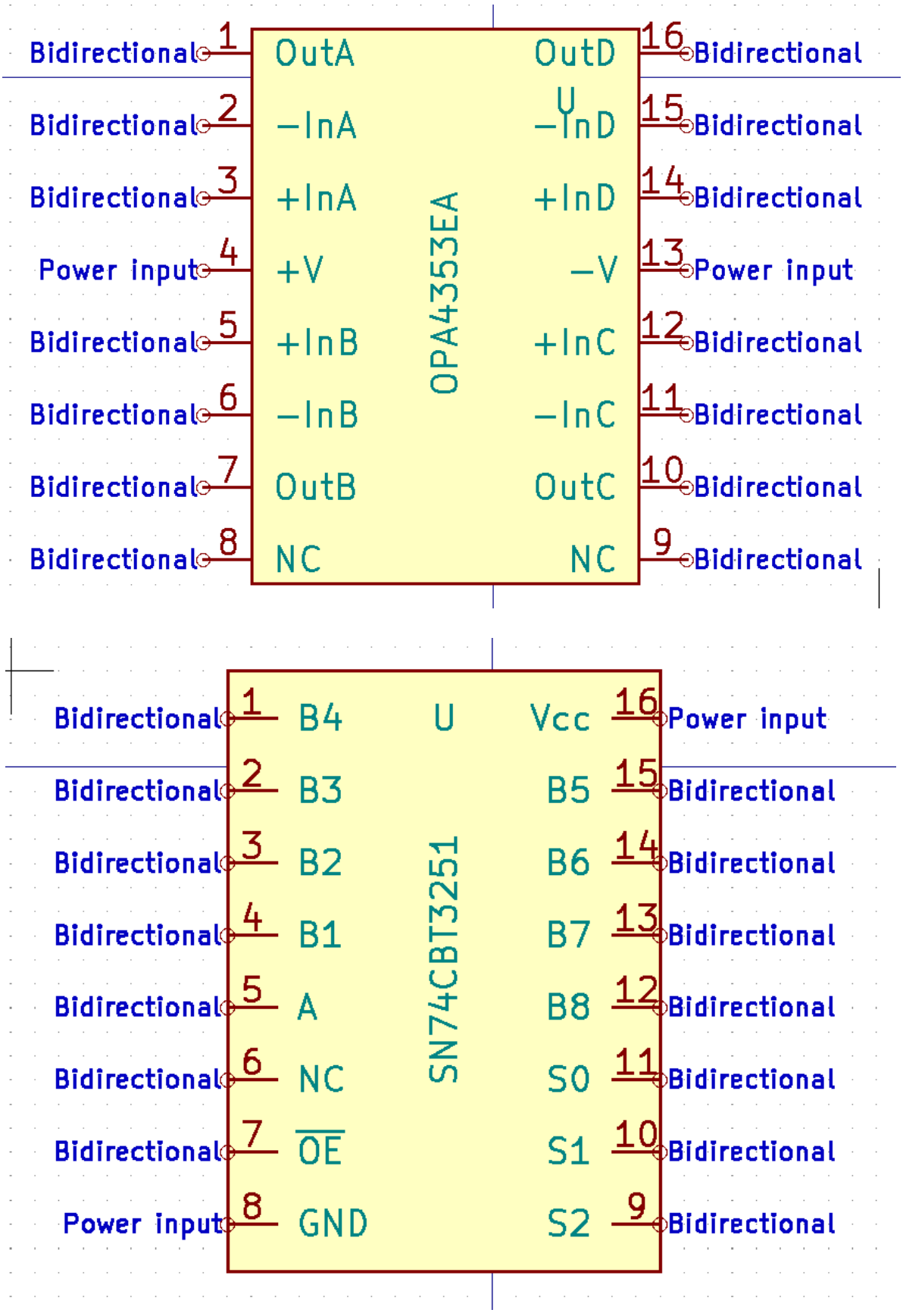
Εικόνα 10. Συνδεσμολογία κουμπιού reset

5.2. Κατασκευή Συμβόλων των Ολοκληρωμένων Κυκλωμάτων

Η πρώτη ενέργεια έπειτα από δημιουργία μίας σχηματικής απεικόνισης είναι η δημιουργία συμβόλων για τα ολοκληρωμένα κυκλώματα. Τα σύμβολα αναπαριστούν την σχέση ανάμεσα στην γεωμετρία των pins και την αρίθμησή τους και μπορούν να σχεδιαστούν με οποιονδήποτε τρόπο, προκειμένου να συνάδει με τη γεωμετρική σχεδίαση της σχηματικής απεικόνισης.

Στην παρούσα εργασία, η γεωμετρία σχεδιάστηκε ίδια με το κανονικό πακέτο του ολοκληρωμένου κυκλώματος. Αυτό σημαίνει ότι, όπως είναι η καταμέτρηση των pins ενός ολοκληρωμένου κυκλώματος, κατά τον ίδιο τρόπο είναι και στο σύμβολο.

Έτσι, ανοίγουμε τον symbol editor από το αρχικό μενού του KiCAD. Έπειτα σχεδιάζουμε δύο σύμβολα, ένα για το ολοκληρωμένο κύκλωμα των τελεστικών ενισχυτών και ένα για το ολοκληρωμένο κύκλωμα με τον πολυπλέκτη.



Εικόνα 11. Σύμβολα ολοκληρωμένων κυκλωμάτων

Όπως είναι εμφανές, χρησιμοποιήσαμε παντού bidirectional επιλογή για να περιγράψουμε την κίνηση του σήματος. Η μόνη εξαίρεση ήταν στα pins της τροφοδοσίας (είσοδος τροφοδοσίας και γείωσης) οπου, αυτή η επιλογή έγινε καθώς, όταν πραγματοποιούσαμε Electrical Rules Check λαμβάναμε error.

5.3. Σχεδιασμός του σχηματικού διαγράμματος

Στο σημείο αυτό ξεκινάμε τη δημιουργία του schematic. Επειδή η πλακέτα μας θα τοποθετείται επάνω στο arduino (arduino shield), το σχεδιαστικό πρόγραμμα KiCAD παρέχει έτοιμο template για αυτό το μικροελεγκτή, το οποιο και επιλέγουμε. Έτσι, το πρόγραμμα θα μας δημιουργήσει όλα τα pins που έχει ο συγκεκριμένος μικροελεγκτής και στη συνέχεια εμείς θα πρέπει να συνδέσουμε το υπόλοιπο κύκλωμα επάνω στα pins.

Συνεπώς, στο στάδιο αυτό ξανασχεδιάζουμε το κύκλωμα που παρουσιάσαμε νωρίτερα, τοποθετώντας όλα τα εξαρτήματα στο KiCAD. Τα εξαρτήματα αποτελούνται απο αντιστάσεις, πυκνωτές, header pins (για τη σύνδεση των καλωδίων που θα φθάνουν στον αισθητήρα, διακόπτης - για reset και barrel jack connector - για την εισοδο τροφοδοτικού για την τροφοδοσία 5V). Η μόνη διαφορά με το αρχικό κύκλωμα είναι ότι, επειδή δεν υπήρχε η δυνατότητα να βρεθεί εύκολα στο εμπόριο πυκνωτής 400pf, τοποθετήσαμε παράλληλα 4 πυκνωτές των 100pf.

Πέραν αυτών, υλοποιήσαμε και μία προσθήκη στο υπάρχον κύκλωμα, η οποια συνίσταται σε έναν decoupling πυκνωτή, για κάθε ένα ολοκληρωμένο κύκλωμα. Πλέον είναι στανταρ διαδικασία, κάθε ολοκληρωμένο κύκλωμα να έχει έναν decoupling πυκνωτή ανάμεσα στην τροφοδοσία και την γείωση, για να σταθεροποιεί την τροφοδοσία του ολοκληρωμένου κυκλώματος. Το κομμάτι αυτό έλειπε απο τον αρχικό σχεδιασμό του κυκλώματος.

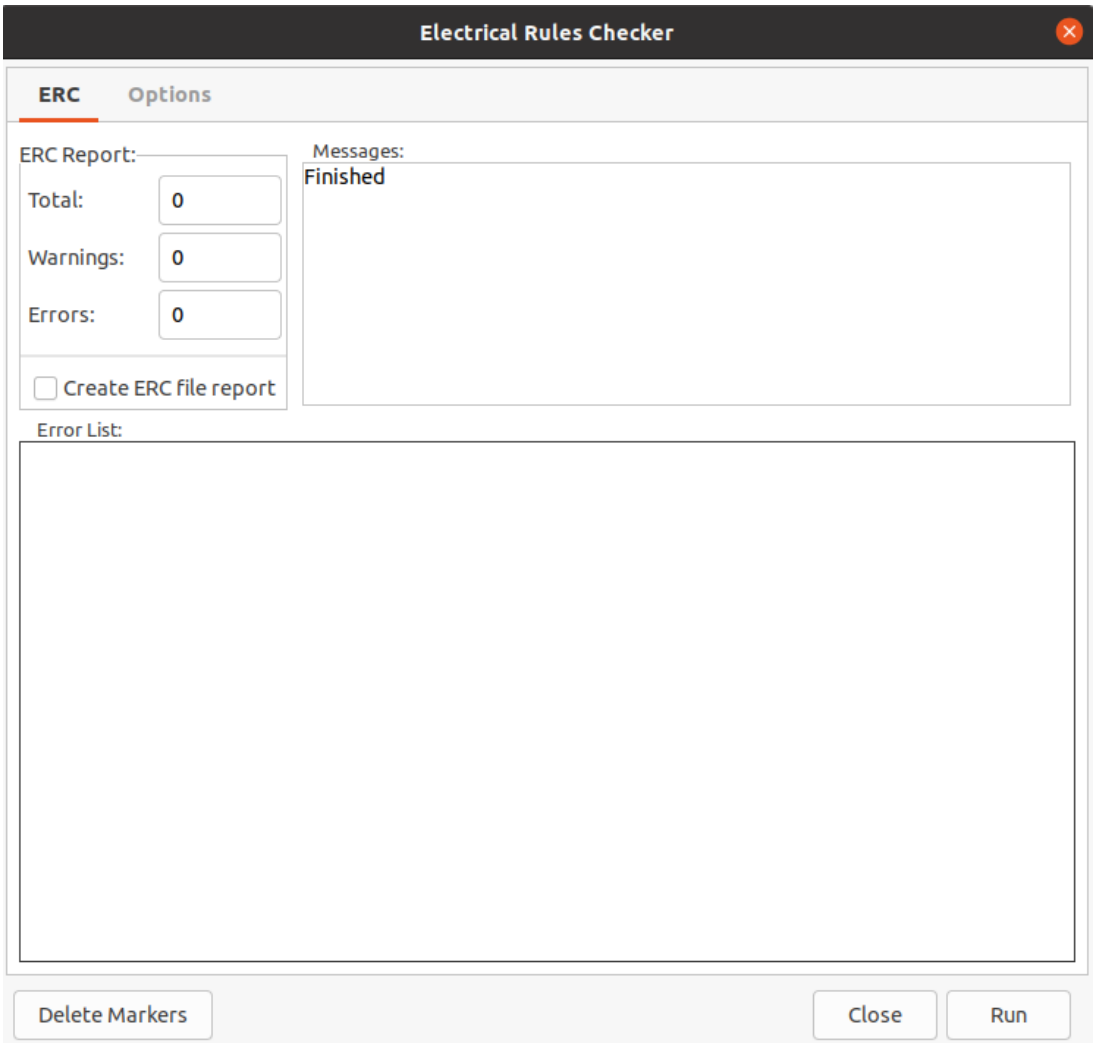
Το επόμενο στάδιο ήταν να κάνουμε “Annotate” όλα τα εξαρτήματα, μέσω του KiCAD. Αυτή η διαδικασία πραγματοποιεί εσωτερική ονοματοδοσία σε όλα τα εξαρτήματα και είναι απαραίτητη για τη σχεδίαση της πλακέτας σε επόμενο βήμα.

Μόλις πραγματοποιήθηκε και αυτή η διαδικασία, τρέξαμε Electrical Rules Check, για να βεβαιωθούμε πως όλα τα εξαρτήματα συνδέονται σωστά. Παρ’ όλα αυτά, αυτή η διαδικασία μας έβγαζε σφάλματα. Τα σφάλματα αυτά λύθηκαν με τις εξής ενέργειες:

- Προσθήκη της ταμπέλας POWER_FLAG στα δύο καλώδια τροφοδοσίας του barrel jack

- Προσθήκη της ταμπέλας GND στο καλώδιο γείωσης του barrel jack
- Προσθήκη της ταμπέλας NC (Not Connected) σε όλα τα pin headers του arduino (που μας δημιούργησε αυτόματα το template) τα οποία δεν χρησιμοποιούνται
- Διαγραφή όλων των ταμπελών και καλωδίων στα pin headers του arduino που είχε δημιουργήσει αυτόματα το template

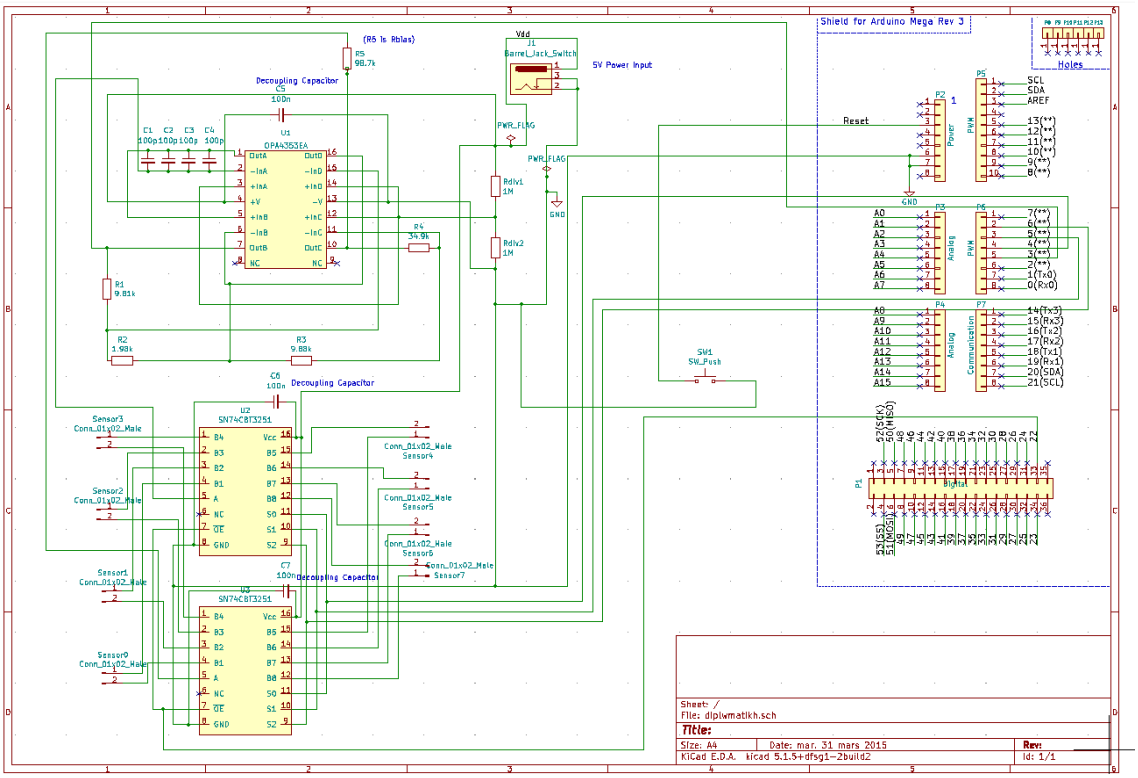
Έπειτα απο τις παραπάνω ενέργειες, ο έλεγχος Electrical Rules είναι επιτυχής, χωρίς κανένα error, όπως φαίνεται και την παρακάτω εικόνα.



Εικόνα 12. Electric rules check

Ως μια τελευταία προσθήκη, τοποθετήσαμε την ταμπέλα Vdd στο καλώδιο τροφοδοσίας του barrel jack, το οποίο θα είναι χρήσιμο αργότερα στο pcbNew υποπρόγραμμα, για να δηλώσουμε την πάνω στρώση χαλκού ως Vdd (δηλαδή το άνω στρώμα χαλκού θα έχει όλο τάση 5V).

Η μορφή του τελικού σχηματικού διαγράμματος είναι η εξής:



Εικόνα 13. Σχηματικό διάγραμμα κυκλώματος

5.4. Ανάθεση footprints

Η ανάθεση footprints αφορά στην εύρεση και δήλωση της γεωμετρίας και μορφολογίας της βάσης των εξαρτημάτων. Μπορεί να έχουμε επιλέξει τα εξαρτήματά μας στο σχηματικό διάγραμμα, αλλά αυτό δεν παρέχει πληροφόρηση σχετικά με τη γεωμετρία και μορφολογία των άκρων τους που θα άπτονται της πλακέτας.

Για παράδειγμα, ένας πυκνωτής έχει ένα συγκεκριμένο σύμβολο για να σχεδιαστεί στο σχηματικό διάγραμμα, παρόλα αυτά στο εμπόριο υπάρχουν δεκάδες πυκνωτές, άλλοι μεγαλύτεροι, άλλοι μικρότεροι, με μεγαλύτερη ή μικρότερη διάμετρο κτλπ.

Φυσικά, πρέπει να γνωρίζουμε την μορφή του footprint των εξαρτημάτων μας, γιατί αλλιώς θα βρεθούμε στη δυσάρεστη θέση να μην έχουμε τη δυνατότητα να τοποθετήσουμε τα εξαρτήματά μας στην πλακέτα.

Από το υποπρόγραμμα του schematic editor, πατάμε το κουμπί Assign PCB Footprints to Schematic Symbols. Η διαδικασία που ακολουθήσαμε για να βρούμε τα footprints είναι η εξής:

- Οι αντιστάσεις μας είναι Through Hole, αυτό σημαίνει πως η μορφή τους είναι Axial. Το DIN0207 είναι ο πιο συχνός τύπος των αντιστάσεων. Μετρήσαμε και τις διαστάσεις των αντιστάσεων για να σιγουρευτούμε και στη συνέχεια επιλέξαμε να είναι horizontal, επειδή είναι τοποθετημένες σε οριζόντια θέση επάνω στην πλακέτα
- Οι πυκνωτές που προμηθευτήκαμε είναι κεραμικοί, άρα επιλέγουμε C_Disk. Η παράμετρος P είναι η μόνη σημαντική. Σημαίνει pitch και είναι η απόσταση μεταξύ των δύο pads που θα τοποθετηθεί ο πυκνωτής
- Για το barrel jack, μετρήσαμε τις διαστάσεις για να βρούμε το συγκεκριμένο footprint και επιλέξαμε ένα που να είναι κατάλληλο. Στη βιβλιοθήκη των footprints του KiCAD, μπορούμε να επιλέξουμε ένα footprint και με δεξί κλίκ να επιλέξουμε View Footprint για να δούμε τη μορφή του και να μετρήσουμε τις διαστάσεις του, ώστε να είμαστε σε θέση να ελέγξουμε εάν συνάδει με το εξάρτημα που έχουμε
- Η ίδια διαδικασία με παραπάνω εφαρμόστηκε για να βρούμε και το κατάλληλο footprint για το διακόπτη
- Τα τρία ολοκληρωμένα κυκλώματα έχουν ίδιο package, άρα θα έχουν και το ίδιο footprint. Στο διαδίκτυο βρήκαμε footprint για το συγκεκριμένο package εδώ: https://www.snapeda.com/parts/OPA4353EA/250G4/Texas%20Instruments/view-part/

Στη συνέχεια, κάναμε import απο το κύριο μενού, ανοίγοντας το υποπρόγραμμα Footprint Editor. Επιλέγουμε Preferences → Manage Footprint Libraries. Κλικάρουμε στο Browse Libraries και μεταβαίνουμε στο κατεβασμένο .kicad_mod αρχείο. Το επιλέγουμε και πατάμε OK. Σιγουρευόμαστε οτι το plugin type είναι σε Legacy Mode και πατάμε OK.

Για αναφορά παρατίθεται ο πίνακας με όλα τα footprints.

```
 1      C1 -             100p : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 2      C2 -             100p : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 3      C3 -             100p : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 4      C4 -             100p : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 5      C5 -             100n : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 6      C6 -             100n : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 7      C7 -             100n : Capacitor_THT:C_Disc_D5.0mm_W2.5mm_P5.00mm
 8      J1 - Barrel_Jack_Switch : Connector_BarrelJack:BarrelJack_Horizontal
 9      P1 -          Digital : Socket_Arduino_Mega:Socket_Strip_Arduino_2x18
10      P2 -            Power : Socket_Arduino_Mega:Socket_Strip_Arduino_1x08
11      P3 -           Analog : Socket_Arduino_Mega:Socket_Strip_Arduino_1x08
12      P4 -           Analog : Socket_Arduino_Mega:Socket_Strip_Arduino_1x08
13      P5 -              PWM : Socket_Arduino_Mega:Socket_Strip_Arduino_1x10
14      P6 -              PWM : Socket_Arduino_Mega:Socket_Strip_Arduino_1x08
15      P7 -    Communication : Socket_Arduino_Mega:Socket_Strip_Arduino_1x08
16      P8 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin
17      P9 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin
18     P10 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin
19     P11 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin
20     P12 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin

21     P13 -       CONN_01X01 : Socket_Arduino_Mega:Arduino_1pin
22      R1 -            9.81k : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
23      R2 -            1.98k : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
24      R3 -            9.88k : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
25      R4 -            34.9k : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
26      R5 -            98.7k : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P7.62mm_Horizontal
27   Rdiv1 -               1M : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
28   Rdiv2 -               1M : Resistor_THT:R_Axial_DIN0207_L6.3mm_D2.5mm_P10.16mm_Horizontal
29 Sensor0 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
30 Sensor1 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
31 Sensor2 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
32 Sensor3 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
33 Sensor4 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
34 Sensor5 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
35 Sensor6 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
36 Sensor7 - Conn_01x02_Male : Connector_PinHeader_2.54mm:PinHeader_1x02_P2.54mm_Vertical
37     SW1 -          SW_Push : Button_Switch_SMD:SW_SPST_PTS645
38      U1 -        OPA4353EA : SOP63P599X175-16N:SOP63P599X175-16N
39      U2 -      SN74CBT3251 : SOP63P599X175-16N:SOP63P599X175-16N
40      U3 -      SN74CBT3251 : SOP63P599X175-16N:SOP63P599X175-16N
```

Εικόνα 14. Footprints των εξαρτημάτων

Ενδεικτικά, οι τιμές και οι διαστάσεις των αντιστάσεών μας, που είναι το πιο σύνηθες εξάρτημα στο κύκλωμά μας είναι:

r1	9.81k	1.9 x 3.7mm
r2	1.98k	2.5 x 6.8mm
r3	9.88k	1.9 x 3.7mm
r4	34.9k	2.5 x 6.8mm
r5	98.7k	1.9 x 3.7mm
rdiv1,2	1M	2.5 x 6.8mm

Πίνακας 3. Τιμές αντιστάσεων

Το επόμενο βήμα είναι να επιστρέψουμε ξανά στο schematic editor και να πατήσουμε το κουμπί Generate Netlist. Αυτό θα δημιουργήσει ένα αρχείο που θα περιγράφει τις συνδέσεις ανάμεσα στα εξαρτήματά μας.

Τέλος, πατάμε το κουμπί Run pcbNew to layout printed circuit board. Αυτό θα μεταφέρει όλα τα εξαρτήματα και τις συνδέσεις τους στο πρόγραμμα PcbNew, όπου θα σχεδιάζουμε όλα τα traces της πλακέτας.

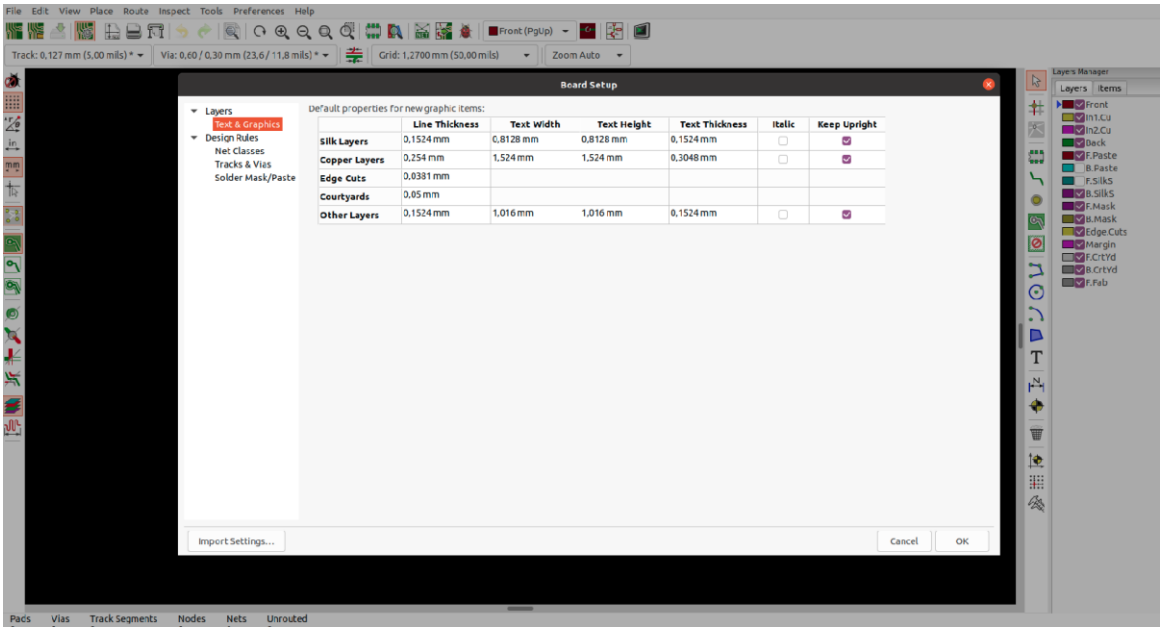
5.5. Σχεδιασμός του layout της πλακέτας (PCBNew)

Αρχικά, πρέπει να παραμετροποιήσουμε τις ρυθμίσεις των διαστάσεων των μονοπατιών των σημάτων ώστε να συμμορφώνονται με τις προδιαγραφές που θέτει η εταιρεία που θα μας κατασκευάσει την πλακέτα.
Στο internet βρήκαμε ένα αρχείο με όλες τις ελάχιστες ρυθμίσεις απο όλες τις κύριες κατασκευαστικές εταιρείες (https://github.com/sethhillbrand/kicad_templates).
Χρησιμοποιήσαμε την jlcpcb για την κατασκευή της πλακέτας, άρα κατεβάσαμε τις ρυθμίσεις για αυτή την εταιρεία.

Έπειτα εισαγάγαμε τις ρυθμίσεις πηγαίνοντας στο

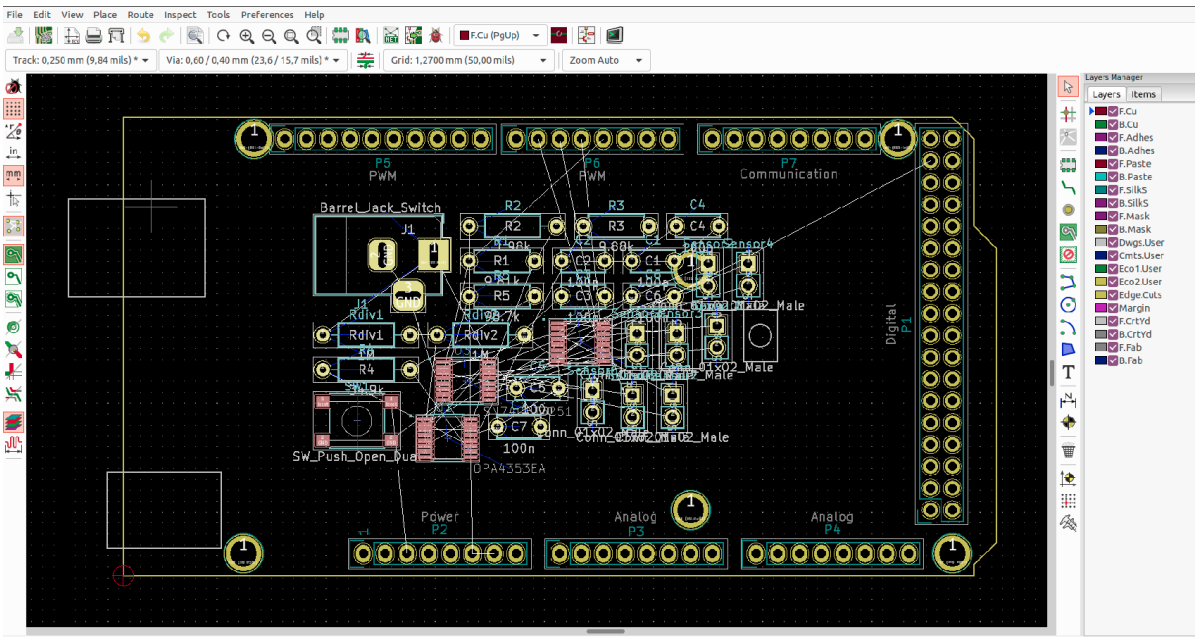
PCBnew -> Board setup -> Import Settings -> select all -> select pro file

Στην παρακάτω εικόνα βλέπουμε τις αρχικές τιμές, οι οποίες άλλαξαν αυτόματα με την εισαγωγή του αρχείου:



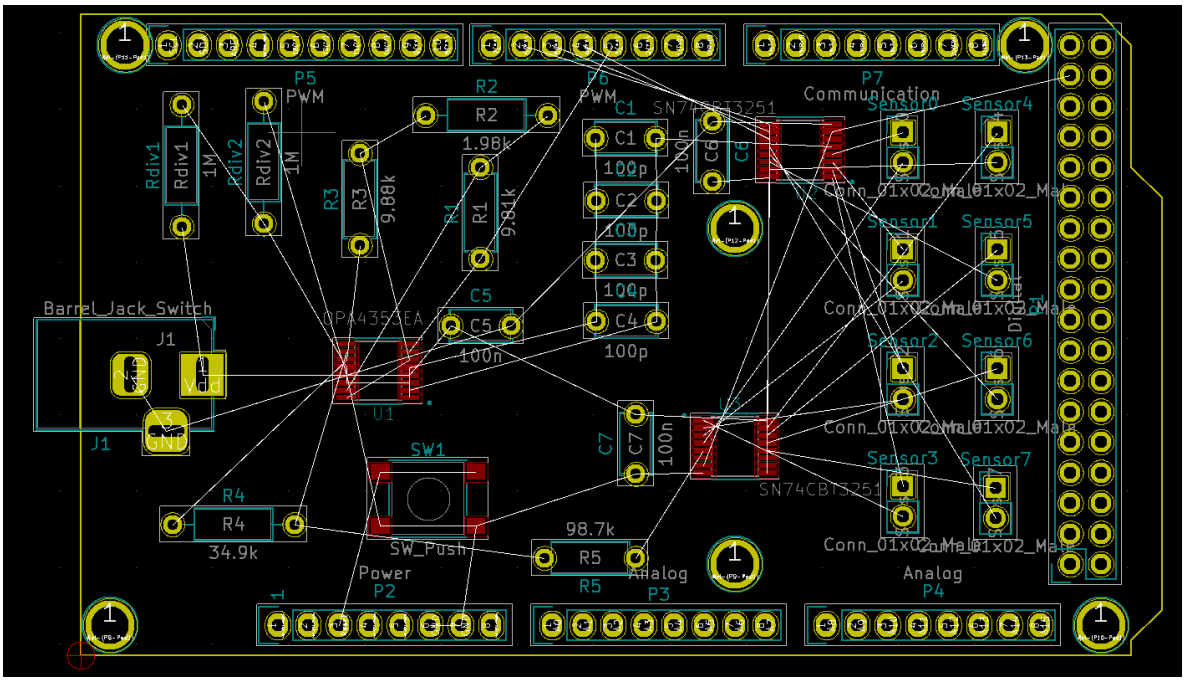
Εικόνα 15. Τιμές παραμέτρων layout

Μετά τη μεταφορά των εξαρτημάτων απο το schematic editor και την εισαγωγή των συνδέσεων απο το netlist αρχείο, τα εξαρτήματα εμφανίζονται συγκεχυμένα στο κέντρο και οι λευκές γραμμές που τα ενώνουν (οι οποίες ονομάζονται ratsnest) μας δείχνουν την συνδεσμολογία που πρέπει να υλοποιήσουμε. Το πώς θα την υλοποιήσουμε, επαφίεται στις δυνατότητές μας:



Εικόνα 16. Αρχική κατάσταση πλακέτας

Αρχικά αναπτύσσουμε όλα τα εξαρτήματα στο χώρο ετσι ώστε να έχουμε την πιο εύκολη υλοποίηση των συνδέσεων.



Εικόνα 17. Ανάπτυγμα εξαρτημάτων της πλακέτας στο χώρο

Στο επόμενο βήμα θα δημιουργήσουμε fill zones στην πάνω και στην κάτω μεριά της πλακέτας.

Αυτό σημαίνει ότι, το κύριο κομμάτι του χαλκού στις δύο μεριές θα έχει την ίδια τάση.

Αυτό συνήθως γίνεται για την τάση τροφοδοσίας και τη γείωση, που είναι τάσεις στις οποίες πολλά εξαρτήματα απαιτούν πρόσβαση. Συνεπώς, με αυτό τον τρόπο μειώνεται η πολυπλοκότητα των συνδέσεων, αφού αντί να σχεδιάζουμε πολλά και μακριά traces, έχουμε πρόσβαση στην τάση που επιθυμούμε, η οποία πλέον βρίσκεται σε πολύ κοντινή απόσταση από την πλειοψηφία των εξαρτημάτων.

Ο τρόπος για να το πετύχουμε αυτό είναι με το να επιλέξουμε Add Fill Zone, στη συνέχεια την όπισθεν μεριά της πλακέτας και τέλος, GND. Συνεπώς, μετατρέψαμε το χαλκό στην πίσω μεριά στο να έχει τάση γείωσης.

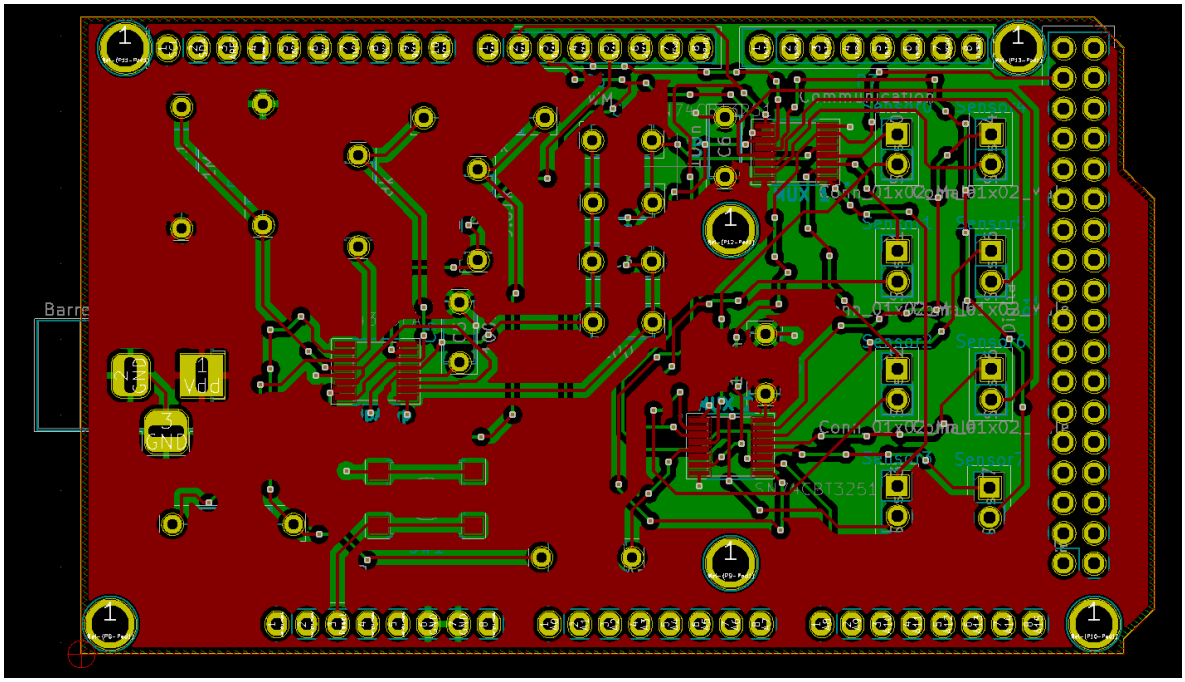
Για το έμπροσθεν μέρος, αφού έχουμε ήδη το όπισθεν μέρος, επιλέγουμε duplicate layer as VDD (5v). Για το λόγο αυτό, τοποθετησαμε την ταμπέλα Vdd στο σχηματικό διάγραμμα σε κάποιο από τα προηγούμενα βήματα. Αυτό σημαίνει ότι, πλέον ο χαλκός του μπροστινού μέρους έχει τάση 5V.

Πλέον φτάσαμε στο στάδιο σχεδιασμού των traces, δηλαδή των μονοπατιών χαλκού που ενώνουν τα εξαρτήματα και μεταφέρουν το σήμα. Σκοπός μας είναι να μη διχοτομούνται δύο traces μεταξύ τους. Στην περίπτωση όπου δεν υπάρχει διαθέσιμη διαδρομή, τότε μπορούμε να σχεδιάσουμε μία Via, που είναι ουσιαστικά μία τρύπα που θα συνεχίσει το trace στην άλλη πλευρά της πλακέτας.

Στο σημείο αυτό θα θέλαμε να κάνουμε μία ακόμα σημείωση όσον αφορά στους decoupling πυκνωτές. Αναφέραμε πριν οτι αποτελούν μία επιπλέον προσθήκη η οποία έλειπε απο το αρχικό κύκλωμα. Για να υλοποιηθούν σωστά οι decoupling πυκνωτές, πρέπει να είναι όσο το δυνατόν πλησιέστερα στο ολοκληρωμένο κύκλωμα. Οπότε, τους τοποθετήσαμε στην πλησιέστερη απόσταση από το ολοκληρωμένο κύκλωμα και παράλληλα συνδέσαμε το GND pin του πυνκωτή απευθείας με το GND plane της όπισθεν μεριάς, μέσω μιας via (και όχι με το GND του ολοκληρωμένου κυκλώματος).

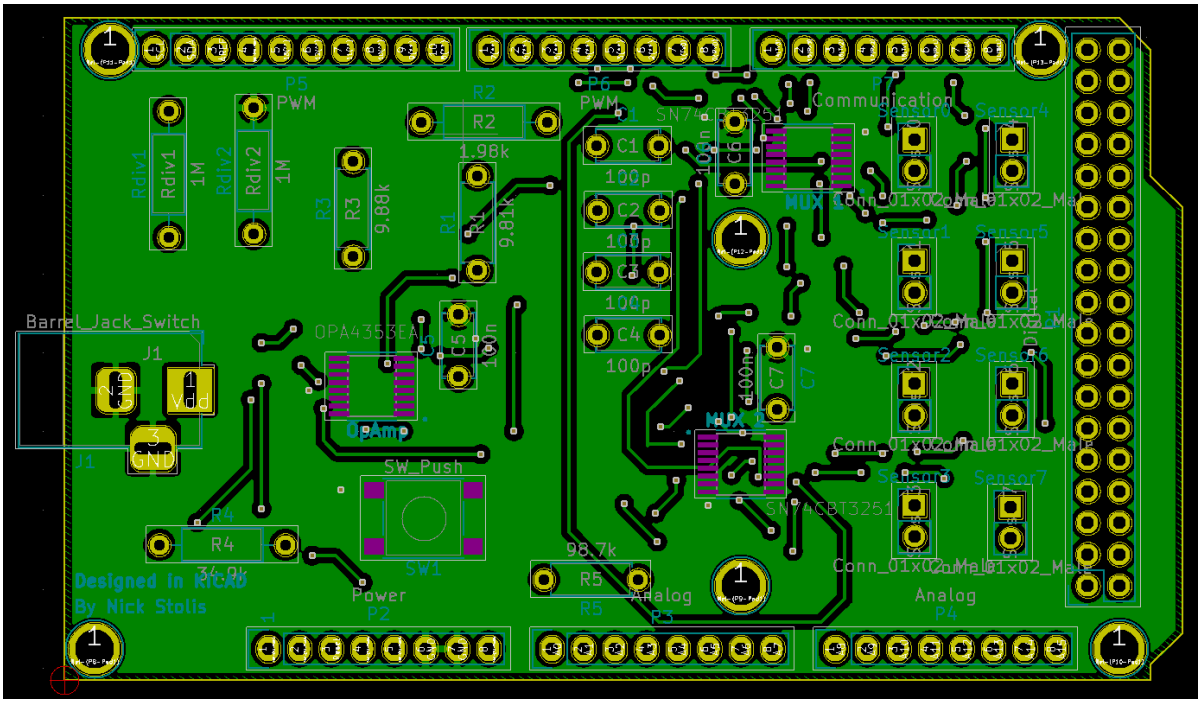
Το pin του πυκνωτή με την τάση τροφοδοσίας συνδέεται με το Vcc του ολοκληρωμένου κυκλώματος και με μία άλλη σύνδεση στο power plane (5V) που βρίσκεται στην επάνω μεριά της πλακέτας.

Η τελική μορφή του layout είναι η εξής:



Εικόνα 18. Layout πλακέτας

Με κόκκινο χρώμα απεικονίζονται τα traces στην επάνω μεριά της πλακέτας και με πράσινο τα traces στην όπισθεν μεριά. Λόγω του ότι η όπισθεν μεριά επικαλύπτεται απο μέρος της έμπροσθεν, παρατίθουμε και την πίσω πλευρά.



Εικόνα 19. Layout πλακέτας – πίσω όψη

Τέλος, παράγουμε τα αρχεία gerber τα οποία θα στείλουμε στην εταιρεία για την κατασκευή της πλακέτας. Απο το pcbNew, πατάμε το Plot κουμπί και επιλέγουμε το τι θα τυπωθεί. Αυτά που κρατήσαμε είναι:

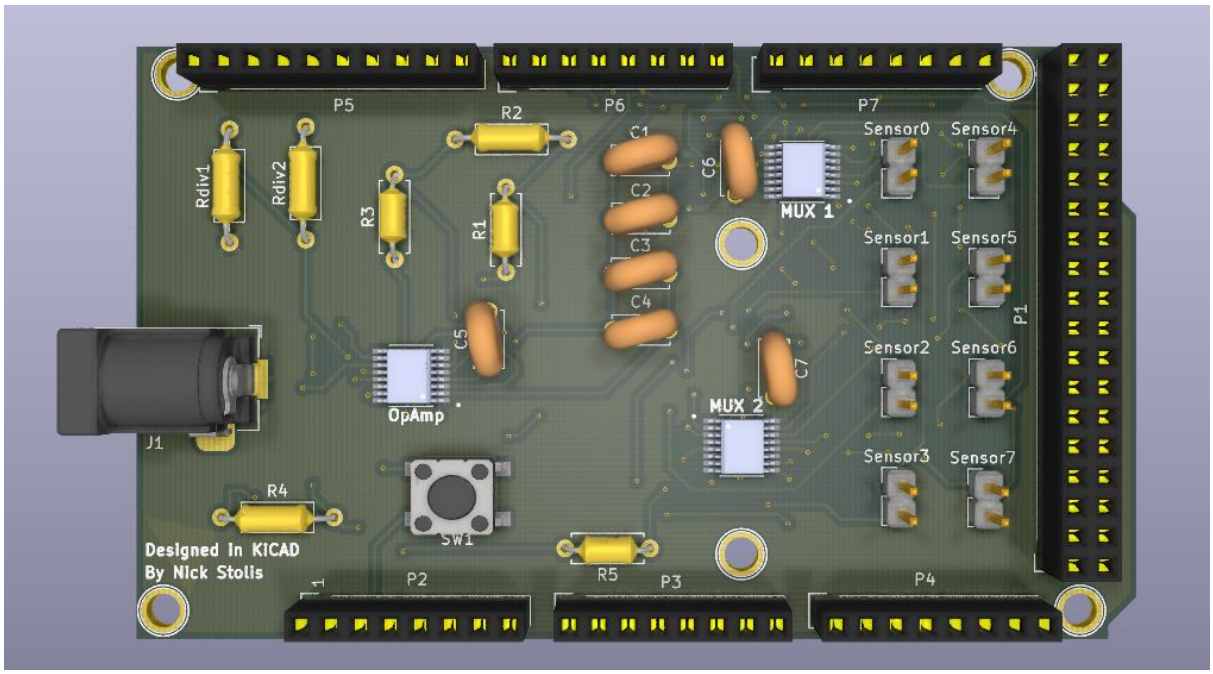
- .drl file (περιέχει πληροφορίες για τις τρύπες της πλακέτας)
- back και front .gbr files (τα gerbers με την έμπροσθεν και όπισθεν μεριά της πλακέτας)
- back και front mask .gbr files (τα gerbers με πληροφορίες για τη μάσκα – τη ζωγραφισμένη περιοχή που προστατεύει το χαλκό στην όπισθεν και έμπροσθεν μεριά της πλακέτας)
- back και front silkscreen .gbr files (τα gerbers με πληροφορίες για τα γραφικά όπως κείμενο, στην όπισθεν και έμπροσθεν μεριά της πλακέτας)
- edge cuts .gbr file (το gerber με τις πληροφορίες των ορίων της πλακέτας μας)

5.6. Κατασκευή 3d μοντέλου

Η κατασκευή του τρισδιάστατου μοντέλου της πλακέτας μας είναι ένα προαιρετικό βήμα για την κατασκευή της φυσικής πλακέτας. Δεν είναι απαραίτητη ενέργεια και συμπεριλαμβάνεται στην εργασία για λόγους πληρότητας.

Αρχικά, κατεβάσαμε απο το διαδίκτυο 3d μοντέλα για κοινά εξαρτήματα σαν και αυτά που χρησιμοποιούμε εμείς. Έπειτα, ανοίγουμε το pcbNew, επιλέγουμε ένα εξάρτημα και πατάμε το 'E'. Επιλέγουμε 3d settings, διαλέγουμε το μοντέλο που επιθυμούμε και το μετακινούμε στον τρισδιάστατο χώρο με τις μεταβλητές rotate, scale και offset.

Παραθέτουμε το 3d μοντέλο της πλακέτας μας στην παρακάτω εικόνα.



Εικόνα 20. Τρισδιάστατο μοντέλο πλακέτας

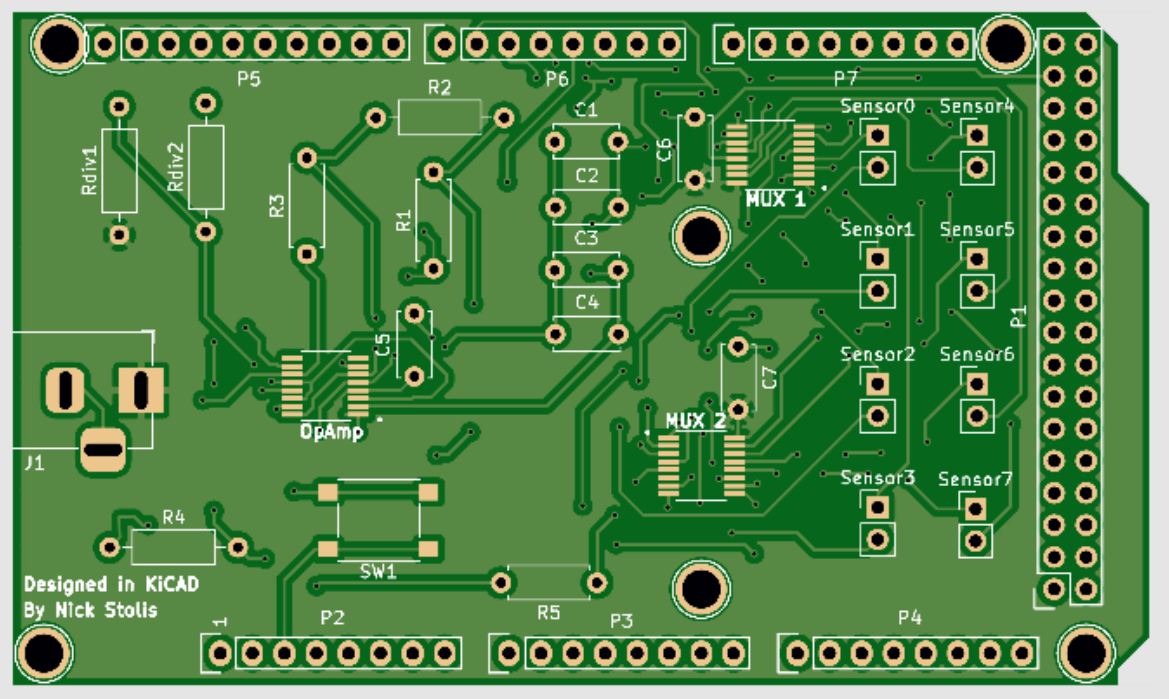
5.7. Δεύτερη Έκδοση Πλακέτας

Λόγω κάποιων λανθασμένων επιλογών, προβήκαμε στην κατασκευή μιας δεύτερης πλακέτας. Οι αλλαγές της νέας πλακέτας σε σχέση με την πρώτη ήταν οι εξής:

- Μεγαλύτερα pin headers. Η προηγούμενη πλακέτα είχε μικρά pin headers και ως αποτέλεσμα, κάποια εξαρτήματα έρχονταν σε επαφή με τον μικροελεγκτή απο την κάτω μεριά.
- Μικρότερο μέγεθος πλακέτας

- Σωστή τοποθέτηση των decoupling πυκνωτών ακριβώς δίπλα απο τα ολοκληρωμένα κυκλώματα. Στην προηγούμενη πλακέτα είχαν μεγάλη απόσταση και δεν είχε νόημα η υπαρξή τους
- Αύξηση του μήκους των traces. Πιο συγκεκριμένα, ο διπλασιασμός τους. Στην προηγούμενη πλακέτα είχαμε επιλέξει το μικρότερο δυνατό μήκος που παρείχε το board house. Αλλα επειδή η πλακέτα ήταν προβληματική, διπλασιάσαμε το trace στα πλαίσια της αποσφαλμάτωσης. Δεν υπάρχει λόγος να έχουμε το μικρότερο δυνατό μήκος, αλλά την ακραία τιμή απο αυτές που μας παρέχει η εταιρεία παραγωγής της πλακέτας.

Η πλακέτα μετά την εκτύπωσή της:



Εικόνα 21. Τυπωμένη πλακέτα

5.8. Συγκόληση Πλακέτας

Τα περισσότερα εξαρτήματα της πλακέτας ήταν τύπου through-hole, οπότε η συγκόλλησή τους πραγματοποιήθηκε με τον κλασσικό τρόπο, δηλαδή κολλητήρι-καλάι. Το πιο δύσκολο μέρος των κολλήσεων των through-hole εξαρτημάτων ήταν οι κολλήσεις των pin headers, οι οποίες εγκυμονούν τους κινδύνους να καεί το πλαστικό ή να μετακινηθεί το pin κατα την διάρκεια της κόλλησης και έτσι να κολληθεί υπο γωνία.

Για το λόγο αυτό, τα pin headers ισορροπήθηκαν πρίν πραγματοποιηθεί η κόλληση. Παραδείγματος χάριν, τοποθετήσαμε την πλακέτα στο arduino για να εξασφαλιστεί ότι τα pin headers θα είναι ίσια. Έπειτα πραγματοποιήθηκε κόλληση στο ένα άκρο της συστοιχίας και μετά στο άλλο άκρο. Μόλις είχε εξασφαλιστεί πως είχαν κολληθεί ίσια, πραγματοποιήθηκε κόλληση και στο υπόλοιπα pins.

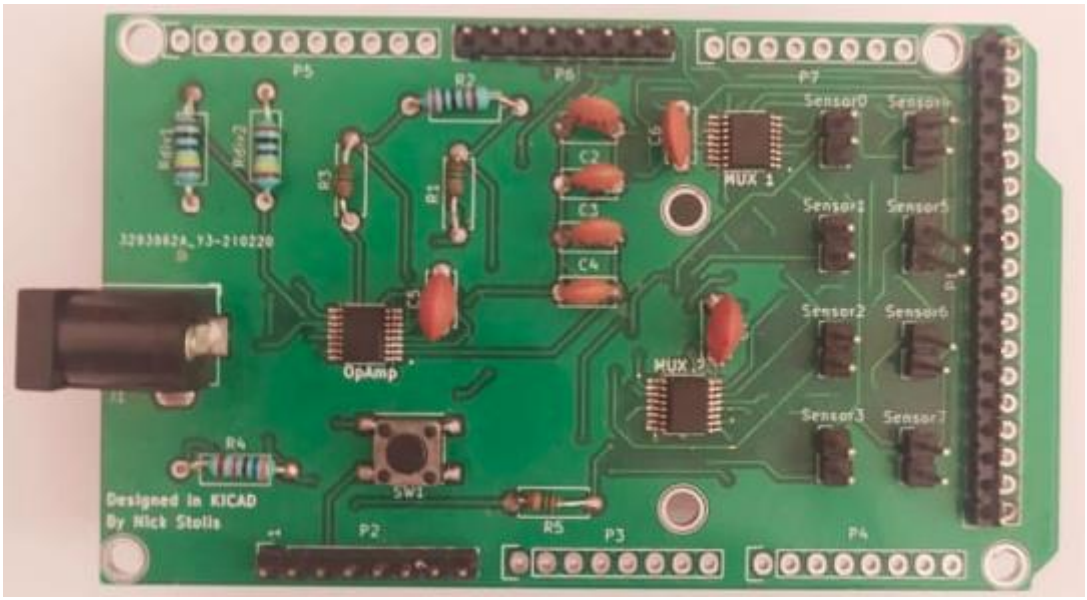
Το πιο δύσκολο ήταν να κολληθούν τα τρία ολοκληρωμένα κυκλώματα που ήταν σε μορφή smd. Για να πραγματοποιηθεί αυτό χρησιμοποιήσαμε solder paste. Τοποθετήσαμε μικρή ποσότητα solder paste η οποία αλείφθηκε κατα μήκος της περιοχής συγκόλησης του ολοκληρωμένου κυκλώματος με ένα μικρό αντικείμενο. Φροντίσαμε στις άκρες να έχει λίγη ποσότητα, ετσι ώστε να μπορούμε να ευθυγραμμίσουμε το ολοκληρωμένο κύκλωμα με την πλακέτα.

Έπειτα τοποθετήσαμε προσκετικά το ολοκληρωμένο κύκλωμα, βάλαμε το heat gun σε χαμηλές στροφές και ξεκινήσαμε παράλληλα με την πλακέτα, αλλά απο ψηλό ύψος προκειμένου να εξασφαλιστεί πως η ροή του αέρα είναι κάθετη στην πλακέτα και δεν έρχεται υπο γωνία, ώστε να μην μετακινηθεί το ολοκληρωμένο κύκλωμα. Στην πορεία κατεβάσαμε το heat gun προοδευτικά σε όλο και χαμηλότερες στροφές ώστε να είναι σε κατάλληλο ύψος και περιμέναμε μέχρι να είναι οπτικά εμφανές πως πραγματοποιήθηκε η κόλληση.

Η σειρά των κολλήσεων ήταν smd εξαρτήματα, pin headers και τέλος τα υπόλοιπα through-hole εξαρτήματα. Ο λόγος ήταν ότι, ο θερμός αέρας του heat gun θα μπορούσε να προξενήσει βλάβες στα άλλα εξαρτήματα, ειδικά σε αυτά που αποτελούνται απο πλαστικό.

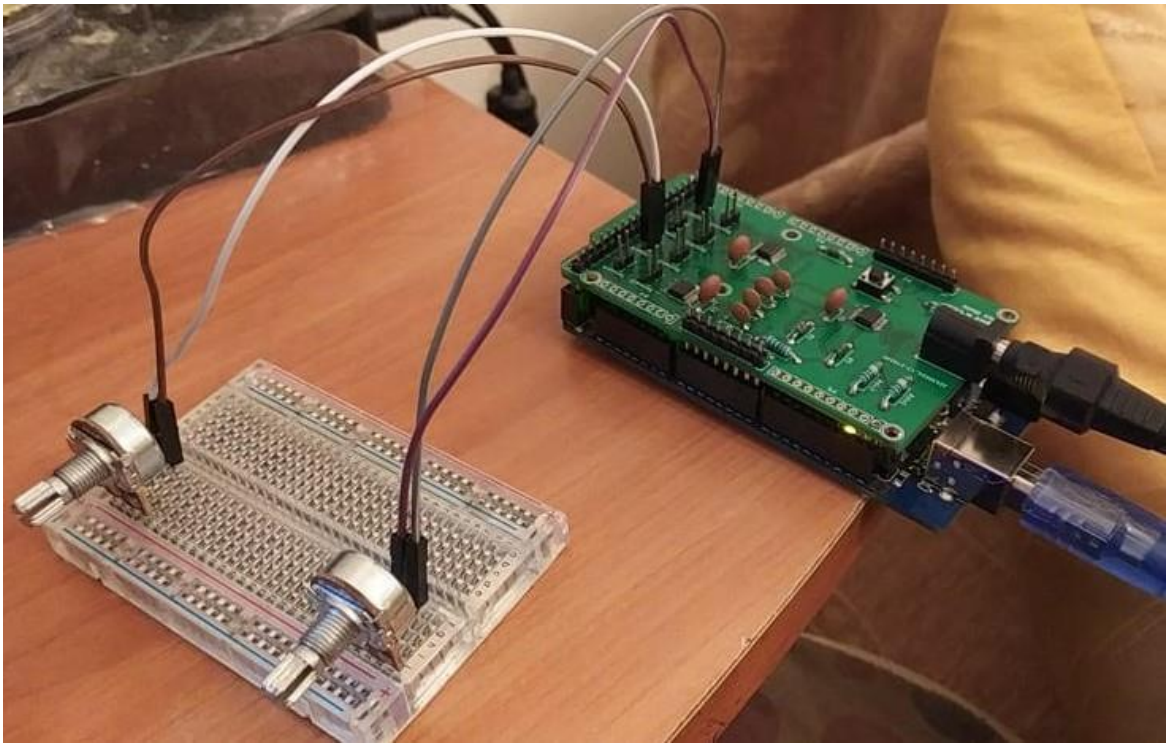
Τέλος, με την ολοκλήρωση των κολλήσεων, πραγματοποιήθηκε καθαρισμός με αιθανόλη, προκειμένου να εξαλειφθούν τα υπολοίματα τα οποία ενδέχεται να προκαλέσουν βραχυκυκλώματα και βλάβες.

Στην παρακάτω φωτογραφία βλέπουμε την πλακέτα συναρμολογημένη.



Εικόνα 22. Συναρμολογημένη πλακέτα

Παρακάτω παραθέτουμε την ολοκληρωμένη πλακέτα συνδεδεμένη με ποτενσιόμετρα που προσομοιάζουν αισθητήρες μεταβαλλόμενης αντίστασης.



Εικόνα 23. Η πλακέτα σε λειτουργία

6. Πρόγραμμα Συλλογής Δεδομένων και Ελέγχου του Μικροελεγκτή

Για να τρέξει ο κώδικας που παραθέτουμε, θα πρέπει να έχουμε την python εγκατεστημένη και επίσης το πακέτο pyserial, το οποίο μπορεί να εγκατασταθεί με την εντολή pip install pyserial.

6.1. Imports και global μεταβλητές

Η πρώτη ενέργεια στην οποία πρέπει να προβούμε είναι να δηλώσουμε τις βιβλιοθήκες που θα χρησιμοποιηθούν στο πρόγραμμά μας.

```
import tkinter as tk
import tkinter.ttk as ttk
from tkinter import scrolledtext
import serial.tools.list_ports
from time import sleep
from datetime import datetime


HEIGHT = 800
WIDTH = 800

baudRate = 9600

ser = None
after_id = None
found_coordinate = None
coordinate = None
dropdown_value = None
modeOfOperation = None

serial_ready = False

sensor = [[] for i in range(8)]
```

Οι τρείς πρώτες γραμμές αφορούν στην εισαγωγή του γραφικού περιβάλλοντος της εφαρμογής μας. Συγκεκριμένα χρησιμοποιούμε το tkinter το οποίο είναι μία βιβλιοθήκη της python που μας επιτρέπει να δημιουργούμε γραφικό περιβάλλον διεπαφής με το χρήστη (gui – Graphical User Interface).

Η γραμμή που εισάγουμε το serial.tools.list_ports καλεί μία βιβλιοθήκη η οποία απαριθμεί όλες τις σειριακές θύρες του συστήματός μας, έτσι ώστε ο χρήστης να έχει τη δυνατότητα να επιλέξει σε ποιά είναι συνδεδεμένος ο μικροελεγκτής και να αρχικοποιήσει μία σύνδεση με αυτόν.

Η γραμμή που καλεί το sleep μας δίνει τη δυνατότητα να «παγώνουμε» το πρόγραμμά μας, πράγμα απολύτως απαραίτητο για την εφαρμογή αυτή, όπως θα διαπιστώσουμε αργότερα.

Τέλος, η γραμμή που καλεί το datetime, θα μας δώσει την δυνατότητα να λαμβάνουμε την τρέχουσα ώρα του συστήματος, πράγμα το οποίο θα το χρησιμοποιούμε στην ονοματοδοσία αρχείων όταν κάνουμε export τα δεδομένα σε αρχείο.
Σε επόμενη φάση θα προχωρήσουμε σε μία σύνοψη των global μεταβλητών. Ο ορισμός των HEIGHT και WIDTH πραγματοποιείται για να ορίσουμε το μέγεθος του tkinter παραθύρου. Με το baudRate θέτουμε την ταχύτητα με την οποία θα γίνεται η μεταφορά δεδομένων μεταξύ του μικροελεγκτή και του υπολογιστή (το οποίο θα περάσουμε αργότερα στο serial object). Οι υπόλοιπες μεταβλητές που έχουν ως αρχική τιμή None θα χρειαστούν αργότερα, πάντα σαν global μεταβλητές. Η αρχική τιμή None εξασφαλίζει πως δεν θα γίνει κάποια λάθος αρχικοποίηση από το πρόγραμμα.
Η serial_ready είναι άλλη μία global μεταβλητή η οποία είναι boolean, δηλαδή παίρνει τιμή true η false. Τέλος, η μεταβλητή sensor συνίσταται σε μία λίστα με τους επιμέρους αισθητήρες, όπου θα καταγράφονται οι τιμές που λαμβάνουμε απο το αναλογικό κύκλωμα.

6.2. Κύριο Πρόγραμμα

Παρακάτω παρατίθεται η μορφή του κυρίως προγράμματος

```
root = tk.Tk()
root.title("Sensor Interface")

canvas = tk.Canvas(root, height=HEIGHT, width=WIDTH)
canvas.pack()

#FRAMES GO HERE

stop =True
root.mainloop()
ser.close()
```

Αρχικά, δηλώνουμε το tkinter παράθυρο και έναν τίτλο γι' αυτό. Το canvas είναι κυριολεκτικά ο καμβάς πάνω στον οποίο θα μπορούμε να τοποθετήσουμε ξεχωριστά frames. Με τον όρο frames εννοούμε διαφορετικές διακριτές περιοχές που συνήθως εκπληρώνουν έναν συγκεκριμένο σκοπό. Αυτό θα γίνει περισσότερο κατανοητό στην πορεία της εργασίας. Στη δήλωση του canvas περνάμε σαν παραμέτρους το μήκος και πλάτος του canvas και δηλώνουμε πως το parent element θα είναι το root, δηλαδή το αρχικό παράθυρο tkinter. Κατά συνέπεια,

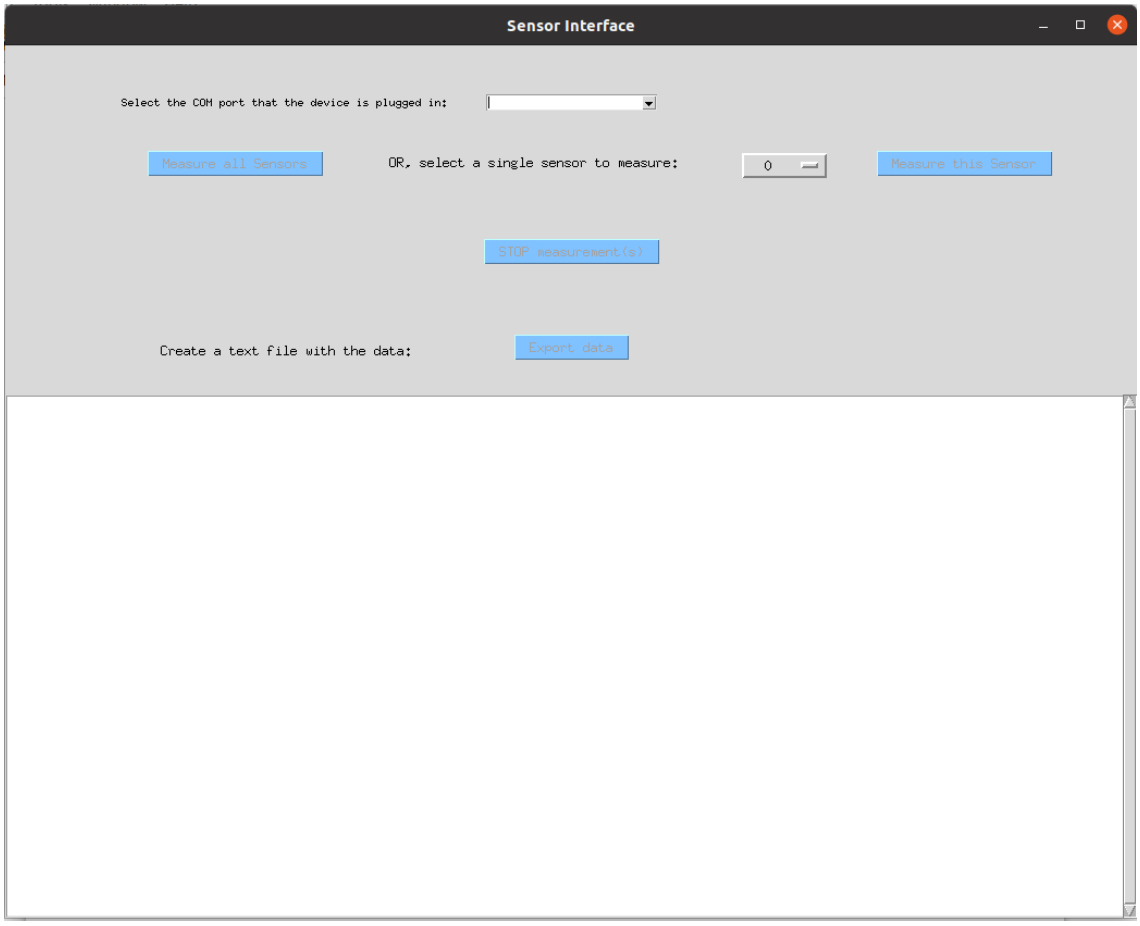
θα μπορούμε να τοποθετήσουμε frames σε όλο το παράθυρο. Με την εντολή canvas.pack() τοποθετούμε το στοιχείο στο παράθυρο.

Έπειτα τοποθετούμε όλα τα frames, τα οποία θα παρουσιαστούν στην επόμενη παράγραφο. Το stop_ είναι ένα flag, το οποίο είναι αληθές όταν δεν γίνεται μέτρηση. Ένα βήμα πριν την εκκίνηση του προγράμματος το δηλώνουμε για να γνωρίζουμε την κατάσταση του συστήματος.

Η εντολή root.mainloop() εκκινεί τον ατέρμονο βρόγχο του προγράμματος. Με άλλα λόγια, αυτή είναι η εντολή που ξεκινάει το πρόγραμμά μας. Κάθε εντολή μετά την mainloop() δεν θα εκτελεστεί όσο το πρόγραμμα και άρα το γραφικό παράθυρο παραμένει ανοιχτό. Συνεπώς, ο κώδικας υπό αυτήν την εντολή θα εκτελεστεί όταν το πρόγραμμα κλείσει. Η εντολή ser.close() θα εκτελεστεί όταν τερματίσουμε το πρόγραμμα και θα φροντίσει να κλείσει την σειριακή σύνδεση μεταξύ του κώδικά μας και του μικροελεγκτή.

6.3. Δήλωση των επιμέρους frames

Στην παρακάτω εικόνα μπορούμε να δούμε τη μορφή του τελικού προγράμματος.



Εικόνα 24. Το πρόγραμμα συλλογής δεδομένων

Όπως φαίνεται, το παράθυρο αποτελείται απο πέντα διακριτές περιοχές. Αυτές θα δηλωθούν σαν πέντε ξεχωριστά frames.

Σε αυτό το σημείο, για την κατανόηση του επόμενου κώδικα είναι χρήσιμο να εξηγήσουμε κάποια στοιχεία για το tkinter. Όλα τα widgets που θα κατασκευάσουμε θα πρέπει να τοποθετηθούν σε κάποιο parent frame. Επειδή το παράθυρό μας χωρίζεται σε πέντε περιοχές, αποφασίστηκε να έχουμε πέντε frames.

Επιπλέον, όταν δηλώνουμε είτε frames είτε widgets, πρέπει να δηλωθεί και σε ποιο σημείο θα βρίσκονται. Για το λόγο αυτό, χρησιμοποιούμε τα relheight, relwidth, relx, rely.

- Τα relheight, relwidth είναι το μήκος και το πλάτος, ορισμένα ως ένας δεκαδικός αριθμός μεταξύ των 0.0 και 1.0, ως ένα κλάσμα του μήκους και του πλάτους του parent widget
- Τα relx, rely αποτελούν το οριζόντιο και κάθετο offset, ορισμένα ως ένας δεκαδικός αριθμός μεταξύ των 0.0 και 1.0, ως ένα κλάσμα του μήκους και του πλάτους του parent widget

6.3.1. Πρώτο frame

Όπως φαίνεται απο την φωτογραφία, η πρώτη περιοχή αφορά στην επιλογή της σειριακής θύρας στην οποία είναι συνδεδεμένος ο μικροελεγκτής μας. Παρακάτω παρατίθεται το κομμάτι του κώδικα που αφορά αυτήν την περιοχή.

```
# --- frame 1 ---
frame1 = tk.Frame(root)
frame1.place(relx=0, rely=0.05, relheight=0.03, relwidth=1, anchor='nw')
label0 = tk.Label(frame1, text="Select the COM port that the device is plugged in: ")
label0.config(font=("TkDefaultFont", 8))
label0.place(relx = 0.1, rely=0.3, relwidth=0.3, relheight=0.5)

cb = ttk.Combobox(frame1, values=serial_ports())
cb.place(relx=0.5, rely=0.5, anchor='center')
cb.bind('<<ComboboxSelected>>', on_select)
# --- frame 1 ---
```

Δηλώνουμε το frame, το οποίο θα έχει ως parent το root (όπως και όλα τα frames) δηλαδή το παράθυρο tkinter. Έπειτα τοποθετούμε το frame. Στην συνέχεια δηλώνουμε ένα label και τι font θα έχει. Έπειτα δηλωνουμε πως θα είναι στο frame1 και το τοποθετούμε στο κατάλληλο σημείο.

Το Combobox είναι το dropdown που θα περιέχει όλες τις σειριακές θύρες του συστήματος. Βλέπουμε οτι η τιμή του είναι τα δεδομένα που θα επιστρέψει η συνάρτηση serial_ports().

Η συνάρτηση αυτή είναι η εξής:

```
def serial_ports():
    return serial.tools.list_ports.comports()
```

Σε περίπτωση που επιλέξουμε ένα στοιχείο απο το dropdown, πραγματοποιείται το event <<ComboboxSelected>>, το οποίο καλεί την συνάρτηση on_select(), η οποία απεικονίζεται παρακάτω:

```
def on_select(event=None):
    global ser
    global serial_ready
    COMPort = cb.get()
    string_separator = "-"
    COMPort = COMPort.split(string_separator, 1)[0]
    COMPort = COMPort[:-1]
    ser = serial.Serial(port = COMPort, baudrate=9600, timeout=0.1)
    sleep(2)
    button_all['state'] = 'normal'
    serial_ready  = True
```

Το ser είναι η μεταβλητή που θα κρατάει το σειριακό object και το serial_ready μας ενημερώνει πως έχει καθιερωθεί μία σειριακή σύνδεση. Στην γλώσσα python η μεταβλητή που θα είναι global πρέπει να δηλωθεί εντός της συνάρτησης.

Η COMPort θα λάβει το αποτέλεσμα της επιλογής του dropdown. Επειδή αυτό πέρα απο την σειριακή θύρα μας παρέχει και άλλες πληροφορίες οι οποίες είναι άχρηστες, σπάμε το αλφαριθμητικό στην πρώτη παύλα που θα συναντήσουμε και κρατάμε μόνο το πρώτο μέρος, το οποίο είναι η σειριακή θύρα στην αγνή της μορφή.

Έπειτα θέτουμε το serial object, περνώντας σαν παραμέτρους την θύρα, το baudrate και το timeout, δηλαδή τον μέγιστο χρόνο που θα αναμένουμε για σειριακα δεδομένα. Στην εν λόγω περίπτωση το timeout είναι 0.1 δευτερόλεπτα, άρα θα επιστρέψει αμέσως μόλις έχουμε διαθέσιμα δεδομένα. Διαφορετικά, θα αναμείνει μέχρι το timeout και θα επιστρέψει όλα τα bytes που ήταν στο buffer.

Στην συνέχεια, παγώνουμε το πρόγραμμα για δύο δευτερόλεπτα. Για μεγάλο χρονικό διάστημα, το πρόγραμμα ήταν ασταθές. Είχαν καταβληθεί μεγάλες προσπάθειες αποσφαλμάτωσης και δεν βρισκόταν κάποιο σοβαρό λάθος. Τελικά, μετά την αρχικοποίηση του serial object, πρέπει να περιμένουμε δύο δευτερόλεπτα πριν προβούμε σε κάποια σειριακή επικοινωνία. Το πρόγραμμα μετά την αναμονή θα ενεργοποιήσει το κουμπί που πραγματοποιεί τις μετρήσεις σε όλους τους αισθητήρες. Εφ'όσον το κουμί ήταν απενεργοποιημένο μέχρι αυτή τη στιγμή, ήταν αδύνατη η οποιαδήποτε ενέργεια και συνεπώς ήμασταν προστατευμένοι από την ενεργοποίηση κάποιας μέτρησης πριν το χρονικό διάστημα των δύο δευτερολέπτων.

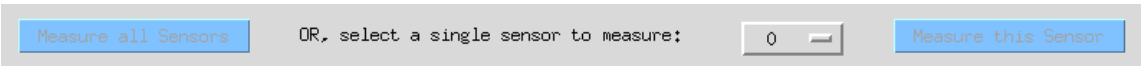
Τέλος, η global μεταβλητή serial_ready που υποδηλώνει πότε ενεργοποιείται μία σειριακή σύνδεση, μετατρέπεται σε αληθής.

6.3.2. Δεύτερο frame

Παραθέτουμε τον κώδικα για το δεύτερο frame.

```
# --- frame 2 ---
frame2 = tk.Frame(root, bd=5)
frame2.place(relx=0, rely=0.1, relheight=0.07, relwidth=1, anchor='nw')

button_all = tk.Button(frame2, text="Measure all Sensors", bg='#80c1ff', fg='red', state='disabled', command=measure_all)
button_all.place(relx=0.2, rely=0.5, anchor='center')

label1 = tk.Label(frame2, text="OR, select a single sensor to measure: ")
label1.config(font=("TkDefaultFont", 9))
label1.place(relx = 0.32, rely=0.3, relwidth=0.3, relheight=0.4)

#dropdown
OPTIONS = list(range(8))
clicked = tk.StringVar(master=frame2)
clicked.set(OPTIONS[0])
clicked.trace("w", dropdown_selection)
drop = tk.OptionMenu(frame2, clicked, *OPTIONS)
drop.place(relx = 0.65, rely=0.25, relwidth=0.08, relheight=0.6)

button_single = tk.Button(frame2, text="Measure this Sensor", bg='#80c1ff', fg='red', state='disabled', command=measure_single)
button_single.place(relx = 0.85, rely=0.5, anchor='center')
# --- frame 2 ---
```

Σαν πρώτο βήμα, ορίζουμε τη θέση του frame στο παράθυρο. Το κουμπί μέτρησης όλων των αισθητήρων είναι απενεργοποιημένο αρχικά, για τους λόγους που εξηγήσαμε προηγουμένως. Μετά δημιουργούμε μία λίστα 8 στοιχείων για το dropdown που θα αναπαριστούν καθέναν απο τους οκτώ αισθητήρες.



Εικόνα 25. Το δεύτερο frame

Η τιμή που θα επιλεγεί απο το dropdown γράφεται στην μεταβλητή clicked. Όταν η μεταβλητή αυτή αλλάξει, εκτελείται η συνάρτηση dropdown_selection(). Τοποθετούμε το dropdown, που στην γλώσσα του tkinter ονομάζεται OptionMenu, και του περνάμε ως παραμέτρους το parent frame, την μεταβλητή που θα επιστραφεί η τιμή του και τις διαφορετικές επιλογές του. Τέλος το τοποθετούμε στο παράθυρο. Επιπροσθέτως τοποθετούμε το κουμπί μέτρησης ενός μεμονωμένου αισθητήρα.

Ας δούμε την συνάρτηση dropdown_selection() που εκτελείται όταν επιλεχθεί ένα στοιχείο του dropdown widget.

```
def dropdown_selection(*args):
    global dropdown_value
    global serial_ready
    dropdown_value = clicked.get()
    if serial_ready == True:
        button_single['state'] = 'normal'
```

Η ουσία αυτής της συνάρτησης είναι ότι, λαμβάνει την τιμή της επιλογής του dropdown (ονομάζεται dropdown_value) και την επιστρέφει σε μία global μεταβλητή για να έχουν πρόσβαση σε αυτήν και άλλα κομμάτια του κώδικα. Επίσης, σε περίπτωση που έχει αρχικοποιηθεί η σειριακή επικοινωνία, μεταβάλλει την κατάσταση του κουμπιού μέτρησης ενός αισθητήρα σε ενεργό.

Με άλλα λόγια, για να μπορεί το κουμπί μέτρησης ενός αισθητήρα να είναι ενεργό, θα πρέπει να ικανοποιούνται δύο παράμετροι:

- Η σειριακή επικοινωνία να έχει οριστεί και αρχικοποιηθεί (συμπεριλαμβανομένου και τον χρόνο αναμονής δύο δευτερολέπτων)
- Υπάρχει η επιλογή μίας τιμής απο το dropdown, που δηλώνει ποιον αισθητήρα θέλουμε να μετρήσουμε

Ας δούμε τώρα τις λειτουργίες και των δύο κουμπιών που πραγματοποιούν τις μετρήσεις.

Το κουμπί του οποίου ο σκοπός είναι να μετράει όλες τις αντιστάσεις, μόλις ενεργοποιηθεί εκτελεί την συνάρτηση measure_all().

```
def measure_all():
    global modeOfOperation
    modeOfOperation = 'A'
    button_export['state']='disabled'
    button_stop['state']='normal'
    ser.write("rf".encode())
    sleep(0.05)
    readSerial()
```

Η συνάρτηση αυτή θέτει την global μεταβλητή modeOfOperation στο να έχει τιμή ‘A’. Η μεταβλητή αυτή μπορεί να λάβει τις τιμές {A,0,1,2,3,4,5,6,7} και ορίζει την κατάσταση μέτρησης στην οποία βρισκόμαστε, με A (All) να σημαίνει όλους τους αισθητήρες, ενώ με 0-7 να δηλώνει πως μετράμε κάποιον μεμονωμένο αισθητήρα.

Στη συνέχεια η συγκεκριμένη μεταβλητή απενεργοποιεί το κουμπί button_export, το οποίο όπως θα δούμε στην συνέχεια είναι το κουμπί που εξάγουμε τα δεδομένα σε αρχείο .txt, όταν έχει ολοκληρωθεί μία μέτρηση. Φυσικά, ενεργοποιούμε το κουμπί button_stop καθώς, όταν μία μέτρηση εκτελείται πρέπει να μπορούμε να την σταματήσουμε.

Στέλνουμε στον μικροελεγκτή τους χαρακτήρες ‘rf’ μέσω της σειριακής επικοινωνίας, διότι αυτή είναι η εντολή για την έναρξη μέτρησης όλων των αισθητήρων. Αναμένουμε 50 milliseconds για να δώσουμε λίγο χρόνο στον μικροελεγκτή και ξεκινάμε τη μέτρηση καλώντας τη συνάρτηση readSerial().

Παρακάτω παρουσιάζεται η συνάρτηση που καλεί το κουμπί, σκοπός του οποίου είναι να μετράει έναν μεμονωμένο αισθητήρα.

```
def measure_single():
    global modeOfOperation
    modeOfOperation = dropdown_value
    print(dropdown_value)
    button_export['state']='disabled'
    button_stop['state']='normal'
    string_to_send = 'r' + ' ' + str(dropdown_value)
    ser.write(string_to_send.encode())
    readSerial()
```

Η συνάρτηση αυτή έχει παρόμοια λειτουργία με τη συνάρτηση που μετράει πολλαπλούς αισθητήρες. Το modeOfOperation είναι η μεταβλητή που αποκτήθηκε απο την επιλογή του dropdown widget. Όπως και στην προηγούμενη συνάρτηση, έτσι και σε αυτή την περίπτωση απενεργοποιούμε το κουμπί που εξαγάγει τα δεδομένα και ενεργοποιούμε το κουμπί που σταματάει τη μέτρηση.

Η εντολή που στέλνουμε στο μικροελεγκτή προφανώς και πρέπει να αλλάξει. Για το λόγο αυτό, ετοιμάζουμε το αλφαριθμητικό σε μία ξεχωριστή μεταβλητή. Το αλφαριθμητικό αυτό αποτελείται απο τον χαρακτήρα ‘r’, ακολουθούμενο απο κενό και την τιμή απο το dropdown (που υποδηλώνει ποιον αισθητήρα θέλουμε να μετρήσουμε). Η συγκεκριμένη τιμή έχει πρώτα μετατραπεί σε αλφαριθμητικό (string), καθώς πρίν ήταν ακέραιος αριθμός.

Έπειτα στέλνουμε το αλφαριθμητικό στο μικροελεγκτή δια μέσου της σειριακής επικοινωνίας και εκκινούμε την συλλογή δεδομένων με την κλήση της συνάρτησης readSerial().

Η συνάρτηση readSerial() καλείται από τα δύο κουμπιά που πραγματοποιούν τις μετρήσεις και είναι η κύρια συνάρτηση του προγράμματός μας. Στην συνέχεια μελετάμε την συνάρτηση αυτή.

```
def readSerial():
    global after_id
    global found_coordinate
    global coordinate
    global sensor
    global stop_
    if stop_ == True:
      stop_=False
      for myList in sensor:
        myList.clear()
    while ser.in_waiting:
        try:
            ser_bytes = ser.readline()
            ser_bytes = ser_bytes.decode("utf-8")
            text.insert("end", ser_bytes)
            if vsb.get()[1]==1.0:
               text.see("end")
            if "SENSOR COORDINATE" in ser_bytes:
               found_coordinate = True
               coordinate = int(ser_bytes.split("=")[1].strip())
               print("Coordinate: ",coordinate)
            if "MEASURED RESISTANCE" in ser_bytes and found_coordinate:
               found_coordinate = False
               resistance = float(ser_bytes.split("=")[1].split("kOhm")[0].strip())
               print("Resistance: ",resistance)
               sensor[coordinate].append(resistance)
            if "This resistance" in ser_bytes and found_coordinate:
               found_coordinate = False
               resistance = 0
               print("Resistance: ",resistance, "  -  Open Circuit")
               sensor[coordinate].append(resistance)
        except UnicodeDecodeError:
            print("UnicodeDecodeError")
    after_id=root.after(50,readSerial)
```

Η συνάρτηση αυτή είναι μία περιοδική συνάρτηση και εκτελείται συνεχόμενα όταν είμαστε σε κατάσταση μετρήσεων.

Η εκκίνησή της readSerial() θα προκληθεί από το πάτημα ενός κουμπιού. Όταν συμβεί αυτό, μέχρι εκείνη τη στιγμή η μεταβλητή stop_, θα ήταν true.

Η αλλαγή της κατάστασης από true σε false είναι που μας επιτρέπει να αντιληφθούμε ότι μία νέα μέτρηση μόλις ξεκίνησε, καθώς μετά, κατα τη διάρκεια της συνεχόμενης εκτέλεσής της, η μεταβλητή αυτή θα είναι πάντα false.

Όταν πραγματοποιηθεί μία νέα μέτρηση, θέλουμε οι λίστες με τις τιμές των προηγούμενων μετρήσεων να διαγραφούν. Για το λόγο αυτό, ελέγχουμε πότε το stop_ flag είναι για πρώτη φορά αληθές και όταν ανιχνευθεί αληθές (πράγμα που σημαίνει πως μία νέα μέτρηση μόλις ξεκίνησε), τότε αλλάζουμε το stop_ σε false και διαγράφουμε όλες τις λίστες αισθητήρων με τις τιμές της προηγούμενης μέτρησης. Στην τρέχουσα μέτρηση, επειδή το stop_ είναι ήδη false, οι τιμές των λιστών θα παραμείνουν ανέπαφες.

```
if stop_ == True:
  stop_=False
  for myList in sensor:
    myList.clear()
```

Έπειτα έχουμε μία while συνθήκη, η οποία θα εκτελείται συνεχώς όσο η συνθήκη είναι αληθής, δηλαδή όσο υπάρχουν δεδομένα στην σειριακή είσοδο.

Μέσα στην while, ο κώδικάς μας θα προστατευτεί με μία συνθήκη try. Με άλλα λόγια, όσο λαμβάνουμε δεδομένα στη σειριακή είσοδο και όσο αυτά τα δεδομένα δεν παρουσιάσουν κάποιο πρόβλημα όσον αφορά στο format τους (το οποίο είναι το Unicode), τότε θα εκτελεστεί το κύριο κομμάτι κώδικα. Σε περίπτωση όμως που υπάρξει κάποιο πρόβλημα, όπως παραδείγματος χάριν να έχουμε κατεστραμμένα δεδομένα για κάποιο λόγο, τότε το πρόγραμμα δεν θα καταρρεύσει, αλλά θα τυπώσει στην κονσόλα ένα μήνυμα “UnicodeError” και θα συνεχίσει κανονικά τη λειτουργία του.

Προχωρώντας στο κυρίως κομμάτι κώδικα της συνάρτησης, αρχικά θα λάβουμε τα δεδομένα απο την σειριακή γραμμή και θα τα αποθηκεύσουμε σε μία μεταβλητή που ονομάζεται ser_bytes. Μετά θα αποκωδικοποιήσουμε τα περιεχόμενα της μεταβλητής με το utf-8 format, το οποίο είναι ένα μη-απωλεστικό σχήμα κωδικοποίησης χαρακτήρων μεταβλητού μήκους για το πρότυπο Unicode.

Ας αναλύσουμε τώρα αυτό το κομμάτι του κώδικα:

```
text.insert("end", ser_bytes)
if vsb.get()[1]==1.0:
    text.see("end")
```

Τα περιεχόμενα της μεταβλητής ser_bytes, δηλαδή τα σειριακά δεδομένα θα εισαχθούν στο πεδίο κειμένου της tkinter εφαρμογής μας. Αυτό το πεδίο βρίσκεται στο κάτω μέρος του παραθύρου, στο frame 5 και θα παρουσιαστεί σε επόμενη παράγραφο. Επιπλέον, έχουμε προσθέσει άλλο ένα χαρακτηριστικό.

Γενικά επιζητούμε η ροή των δεδομένων στο παράθυρο κειμένου να γίνεται autoscroll. Ωστόσο, αν θέλουμε να μελετήσουμε μία συγκεκριμένη μέτρηση, η λειτουργία autoscroll το καθιστά αδύνατον. Για αυτό το λόγο, το κομμάτι κώδικα του if φροντίζει πως, εάν το scrollbar βρίσκεται στο κατώτερο σημείο, τότε θα έχουμε λειτουργία autoscroll. Αντίθετα, όταν δεν βρίσκεται σε αυτή τη θέση, όπως στην περίπτωση που ο χρήστης μελετά μία συγκεκριμένη μέτρηση ενώ ταυτόχρονα νέα δεδομένα εισέρχονται, η λειτουργία autoscroll σταματάει, έτσι ώστε να είναι δυνατή η διερεύνηση των επιθυμητών δεδομένων.

Σε αυτό το στάδιο θα εξετάσουμε την επεξεργασία των δεδομένων που καταφθάνουν. Εάν θυμηθούμε την ανάλυση του κώδικα του μικροελεγκτή, οι αποκρίσεις εμπίπτουν σε τρείς κατηγορίες. Είτε θα πληροφορούμαστε τον χαρακτηριστικό αριθμό του αισθητήρα, είτε την

τιμή του ασιθητήρα, ή θα πληροφορούμαστε πως ο αισθητήρας είναι ανοιχτοκυκλωμένος. Αυτή την συμπεριφορά στα εισερχόμενα δεδομένα χειριζόμαστε στο επόμενο κομμάτι κώδικα.

```
if "SENSOR_COORDINATE" in ser_bytes:
    found_coordinate = True
    coordinate = int(ser_bytes.split("=")[1].strip())
    print("Coordinate: ",coordinate)
if "MEASURED RESISTANCE" in ser_bytes and found_coordinate:
    found_coordinate = False
    resistance = float(ser_bytes.split("=")[1].split("kOhm")[0].strip())
    print("Resistance: ",resistance)
    sensor[coordinate].append(resistance)
if "This resistance" in ser_bytes and found_coordinate:
    found_coordinate = False
    resistance = 0
    print("Resistance: ",resistance, "  -  Open Circuit")
    sensor[coordinate].append(resistance)
```

Εάν στα σειριακά δεδομένα ανιχνεύσουμε το αλφαριθμητικό “SENSOR_COORDINATE”, τότε ο μικροελεγκτής μας στέλνει τον χαρακτηριστικό αριθμό του αισθητήρα στον οποίον πραγματοποιείται η μέτρηση. Τότε ένα flag με τον ονομασία found_coordinate γίνεται αληθές, και αποθηκεύουμε τον αριθμό του αισθητήρα σε μία μεταβλητή. Ο τρόπος που επιτυγχάνεται αυτό, είναι με το να πάρουμε τα εισερχόμενα δεδομένα μετά το ίσον, να τα μετατρέψουμε σε ακέραιο (γιατί είναι αλφαριθμητικό) και να τα αποθηκεύσουμε. Οι επόμενες δύο περιπτώσεις θα εκτελεστούν μόνο όταν έχει ανιχνευθεί ο χαρακτηριστικός αριθμός ενός αισθητήρα.

Εάν ανιχνευθούν οι χαρακτήρες “MEASURED RESISTANCE”, θα αλλάξει το flag που αφορά την εύρεση του χαρακτηριστικού αριθμού των αισθητήρων και θα εξαγάγει τον αριθμό της αντίστασης του αισθητήρα, ο οποίος θα βρίσκεται ανάμεσα στον χαρακτήρα ‘=’ και στον αλφαριθμητικό “kOhm”. Εν συνεχεία, θα τον μετατρέψει σε δεκαδικό απο αλφαριθμητικό και θα τον αποθηκεύσει στην κατάλληλη λίστα για αυτό τον αισθητήρα, η οποία είναι η λίστα sensor[], με index τον αριθμό του ασιθητήρα.

Τέλος, σε περίπτωση που ανιχνευθούν οι χαρακτήρες ανοιχτοκυκλώματος, θα αποθηκεύσει στην κατάλληλη λίστα τον αριθμό μηδέν. Οι ανοιχτοκυκλωμένες αντιστάσεις στην φύση έχουν τιμή άπειρη. Παρ’όλα αυτά, επειδή δεν μπορούμε να αποτυπώσουμε το άπειρο σε ένα διάγραμμα, αποτελεί καλή ένδειξη οτι έχουμε ανοιχτοκύκλωμα διαγραμματικά, όταν μία αντίσταση πέσει στο μηδέν.

Στο τέλος της συνάρτησης υπάρχει η γραμμή `after_id=root.after(50,readSerial)`

,η οποία εξασφαλίζει πως η συνάρτηση readSerial() θα τρέχει περιοδικά κάθε 50ms.

6.3.3. Τρίτο Frame

Το τρίτο frame, όπως είναι εμφανές απο την εικόνα, είναι το κουμπί που σταματάει μία μέτρηση. Αφού βελτιώσαμε τον κώδικα του μικροελεγκτή, πλέον είναι σε θέση να πραγματοποιεί μετρήσεις ατέρμονα. Για το λόγο αυτό, θα πρέπει να έχουμε τη δυνατότητα να σταματάμε μία μέτρηση.

```
# --- frame 3 ---
frame3 = tk.Frame(root, bd=5)
frame3.place(relx=0, rely=0.2, relheight=0.07, relwidth=1, anchor='nw')

button_stop = tk.Button(frame3, text="STOP measurement(s)", bg='#80c1ff', fg='red', state='disabled', command=stop_measurement)
button_stop.place(relx=0.5, rely=0.5, anchor='center')
# --- frame 3 ---
```

Όπως φαίνεται παραπάνω, η κύρια ενέργεια που λαμβάνει χώρα στο frame είναι η τοποθέτηση ενός κουμπιού, σκοπός του οποίου είναι να σταματάει μία τρεχούμενη μέτρηση. Πιο συγκεκριμένα, με το πάτημα του κουμπιού καλείται η συνάρτηση stop_measurement(). Αξίζει να αναφέρουμε πως το κουμπί αρχικά είναι απενεργοποιημένο και ενεργοποιείται μόνο σε περίπτωση που εκκινούμε μία μέτρηση.

Η stop_measurement() είναι η εξής:

```
def stop_measurement():
    global stop_
    stop_=True
    button_stop['state']='disabled'
    ser.write("c".encode())
    root.after_cancel(after_id)
    button_export['state']='normal'
    button_all['state']='disabled'
    state_of_single_button=button_single['state']
    button_single['state']='disabled'
    print("disabled")
    sleep(1)
    button_all['state']='normal'
    if state_of_single_button == 'normal':
        button_single['state']='normal'
```

Η συνάρτηση αυτή τροποποιεί το stop_ flag σε αληθές, ούτως ώστε να γνωρίζουμε ότι σταμάτησε η μέτρηση. Στη συνέχεια, απενεργοποιεί το κουμπί “Stop Measurement” αφού δεν υπάρχει μέτρηση και στέλνουμε τους κατάλληλους χαρακτήρες στον μικροελεγκτή για να του δώσουμε την εντολή να σταματήσει τη διαδικασία μέτρησης.

Επιπλέον, η συγκεκριμένη συνάρτηση ακυρώνει την εκτέλεση της περιοδικής συνάρτησης readSerial(), μέσω του χαρακτηριστικού id της. Στην συνέχεια, ενεργοποιούμε το κουμπί εξαγωγής δεδομένων, γιατί προφανώς τα δεδομένα θα πρέπει να εξαχθούν όταν έχει ολοκληρωθεί μία μέτρηση.

Μετά απενεργοποιούμε τα κουμπιά μετρήσεων και αναμένουμε ένα δευτερόλεπτο πριν τα επανενεργοποιήσουμε. Ο λόγος είναι πως, όταν δεν υπήρχε αυτή η λειτουργία και ενεργοποιούσαμε απ' ευθείας διαφορετικά mode λειτουργίας του Arduino, συναντούσαμε προβλήματα και το πρόγραμμα κατέρεε. Συνεπώς, η αναμονή ενός δευτερόλεπτου μας εξασφαλίζει πως δεν πρόκειται να συμβεί κάτι τέτοιο.

6.3.4. Τέταρτο Frame

Ο κώδικας του τέταρτου frame είναι ο εξής.

```
# --- frame 4 ---
frame4 = tk.Frame(root, bd=5)
frame4.place(relx=0, rely=0.3, relheight=0.09, relwidth=1, anchor='nw')

label2 = tk.Label(frame4, text="Create a text file with the data: ")
label2.place(relx = 0.1, rely=0.3, relwidth=0.3, relheight=0.5)

button_export = tk.Button(frame4, text="Export data", bg='#80c1ff', fg='red', state='disabled', command=export)
button_export.place(relx = 0.5, rely=0.5, anchor='center')
# --- frame 4 ---
```

Το σημαντικό είναι ότι, δημιουργεί ένα κουμπί για την εξαγωγή δεδομένων, το οποίο τρέχει την συνάρτηση export() μόλις πατηθεί. Το κουμπί αυτό είναι απενεργοποιημένο αρχικά. Ας δούμε και την συνάρτηση export().

```
def export():
  global modeOfOperation
  now = str(datetime.now())
  now = now.split('.', 1)[0]
  now = now.replace(':', '.')
  if modeOfOperation != 'A':
      castedMode = int(modeOfOperation)
      fileName = 'S' + modeOfOperation + '-' + now + '.txt'
      with open(fileName, 'w') as f:
        for item in sensor[castedMode]:
            f.write("%s\n" % item)
      f.close()
      print("File exported")
  elif modeOfOperation == 'A':
      fileName = 'S' + modeOfOperation + '-' + now + '.txt'
      with open(fileName, 'w') as f:
          for myList in range(len(sensor)):
              f.write("\n")
              f.write("\n")
              f.write("\n")
              f.write("%s" % namestr(sensor).strip() + '[' + str(myList) + ']\n')
              f.write("--------------------------------\n")
              for item in sensor[myList]:
                  f.write("%s\n" % item)
```

Ο σκοπός της συνάρτησης αυτής είναι να αποθηκεύσει τα δεδομένα της μέτρησης σε ένα αρχείο κειμένου. Σε πρώτη φάση, πρέπει να βρούμε το όνομα του αρχείου. Η πρώτη γραμμή now λαμβάνει την ώρα του συστήματος, η οποία βρίσκεται στη μορφή '2021-05-20 02:06:46.611457'. Με τη γραμμή που καλεί την split() αφαιρούμε τους χαρακτήρες μετά την

τελεία και η γραμμή που καλεί την replace() αντικαθιστά τους χαρακτήρες ‘:’ με ‘.’, καθώς στο λειτουργικό σύστημα windows είναι απαγορευμένοι χαρακτήρες για όνομα αρχείου. Το γεγονός ότι αποθηκεύουμε ημερομηνία και ώρα στο αρχείο με τα δεδομένα μας προστατεύει και απο το ενδεχόμενο δύο αρχεία να έχουν ίδιο όνομα.

Έπειτα. στην περίπτωση που το mode λειτουργίας δεν αποτελεί τη μέτρηση όλων των αισθητήρων (υπενθυμίζουμε οτι η global μεταβητή modeOfOperation μας παρέχει αυτή την πληροφορία), τότε το αποθηκευμένο αρχείο θα έχει την μορφή ‘S’ ακολουθούμενο απο τον αισθητήρα προς μέτρηση, ακολουθούμενο απο την ώρα του συστήματος.

Στη συνέχεια, ανοίγουμε το stream για το αρχείο που θα έχει το όνομα που δημιουργήσαμε, με δικαιώματα εγγραφής. Έπειτα σκανάρουμε τη λίστα με τα δεδομένα που μας αφορούν και κάθε στοιχείο της λίστας τυπώνεται στο αρχείο. Μετά απο κάθε εκτύπωση, τυπώνεται και ο χαρακτήρας αλλαγής γραμμής (carriage return: ‘\n’), ετσι ώστε το επόμενο στοιχείο να τυπωθεί στη δικιά του γραμμή.

Σε περίπτωση που το mode ανάγνωσης αφορά σε όλους τους αισθητήρες, το όνομα του αρχείου μεταβάλλεται για να αντικατοπτρίζει την κατάσταση στην οποία βρισκόμαστε. Αντίστοιχα, στο αρχείο τυπώνεται πρώτα ο χαρακτηριστικός αριθμός του αισθητήρα, ακολουθούμενος απο τα στοιχεία της λίστας του αισθητήρα. Στη συνέχεια, η διαδικασία επαναλαμβάνεται για όλους τους αισθητήρες.

Για να τυπωθεί το όνομα της λίστας του αισθητήρα, χρησιμοποιούμε την συνάρτηση namesrt(), η οποία είναι η κάτωθι:

```
def namestr(obj):
    return [name for name in globals() if globals()[name] is obj][0]
```

6.3.5. Πέμπτο Frame

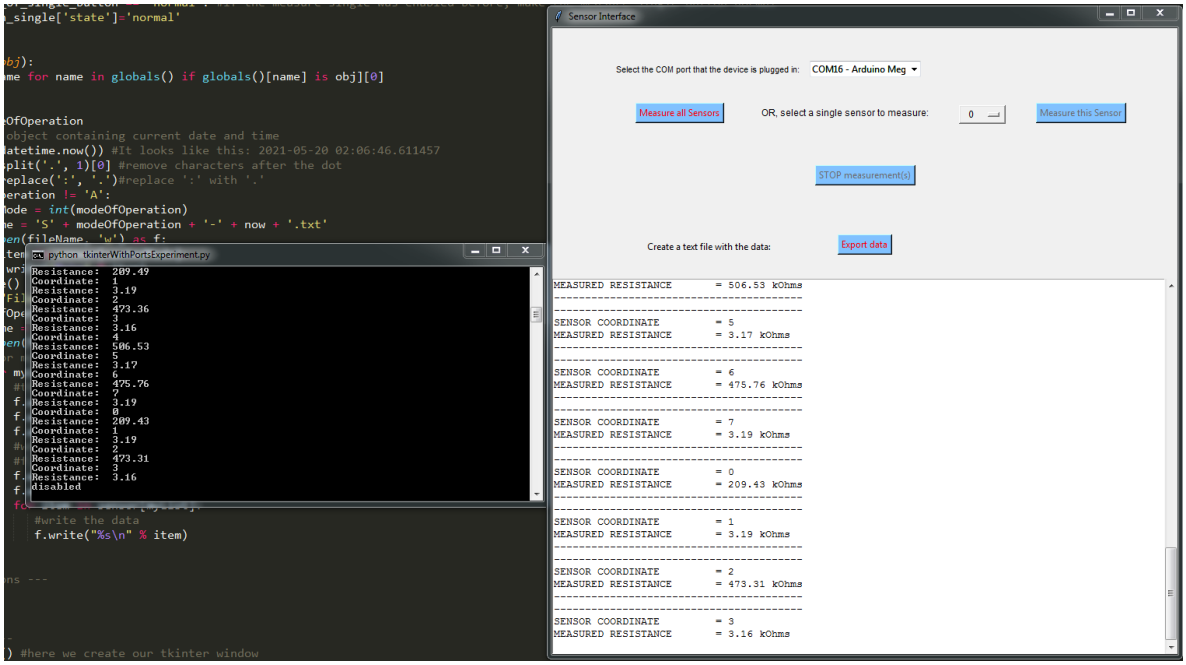
Το πέμπτο και τελευταίο frame περιλαμβάνει το πεδίο όπου το εισερχόμενο κείμενο απο την σειριακή θύρα θα αποθηκεύεται.

```
# --- frame 5 ---
frame5 = tk.Frame(root, bg='#80c1ff')
frame5.place(relx=0.0, rely=0.4, relheight=1, relwidth=1, anchor='nw')

text_frame=tk.Frame(frame5)
text_frame.place(relx=0, rely=0, relheight=0.6, relwidth=1, anchor='nw')
text=tk.Text(text_frame)
text.place(relx=0, rely=0, relheight=1, relwidth=1, anchor='nw')
vsb=tk.Scrollbar(text_frame)
vsb.pack(side='right',fill='y')
text.config(yscrollcommand=vsb.set)
vsb.config(command=text.yview)
# --- frame 5 ---
```

Περιλαμβάνει την τοποθέτηση ενός text frame, καθώς και την τοποθέτηση μιας μπάρας ανακύλησης (scrollbar) στα δεξιά του text frame.

6.3.6. Τελικό Αποτέλεσμα

Στο σημείο αυτό, ας δούμε το τελικό αποτέλεσμα του προγράμματος, μέσω φωτογραφιών.

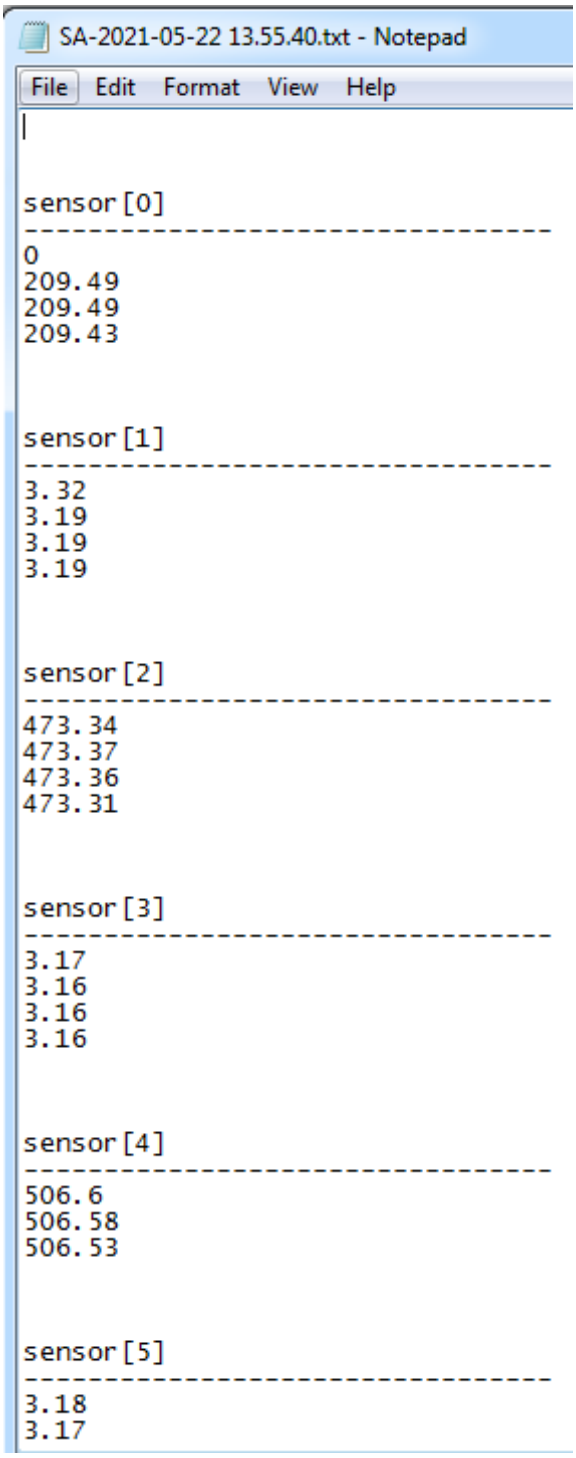


Εικόνα 26. Το πρόγραμμα συλλογής δεδομένων σε λειτουργία

Στην παραπάνω εικόνα βλέπουμε το πρόγραμμα (στα δεξιά) να τρέχει και να συλλέγει δεδομένα. Παρατηρούμε πως, προφανώς έχει επιλεγεί μία σειριακή θύρα και από τα δεδομένα που τυπώνονται, μπορούμε να δούμε ότι μετρώνται όλοι οι αισθητήρες.

Ακριβώς στα αριστερά από το πρόγραμμα, παρατηρούμε το τερματικό στο οποιο επίσης τυπώνουμε δεδομένα.

Έπειτα από τη μέτρηση, πραγματοποιήσαμε εξαγωγή των δεδομένων σε αρχείο. Παρακάτω παρουσιάζεται η μορφή του αρχείου:



Εικόνα 27. Το αρχείο εξαγωγής δεδομένων

Παρατηρούμε στο άνω μέρος τον τίτλο του αρχείου, όπως επίσης και τη μορφή του. Είναι εμφανές ότι εξαγάγονται αποτελέσματα για όλους τους αισθητήρες.

7. Κατασκευή και Χαρακτηρισμός Αισθητήρων

7.1. Παραγωγή Αισθητήρων

Έπειτα από την παραγωγή των νανοσωματιδίων με τη μέθοδο της ιοντοβολής (sputtering), έλαβε χώρα η επεξεργασία τους με υπερήχους για 24 ώρες. Στη συνέχεια, πάνω από τα λειτουργικά υποστρώματα νανοσωματιδίων υπήρξε επικάλυψη των διαλυμάτων πολυμερούς που προέκυψαν, χρησιμοποιώντας την μέθοδο της περιστροφής (spin-coating). Για τους αισθητήρες PIBMA η διαδικασία της επικάλυψης με περιστροφή διήρκησε 1 λεπτο, στις 3000 στροφές (rpm). Παράλληλα, οι παράμετροι της διαδικασίας αυτής τέθηκαν με τέτοιο τρόπο ώστε το τελικό πολυμερές να έχει πάχοςα 500 nm.

Ιοντοβολή ονομάζεται το φαινόμενο κατά το οποίο επιταχυνόμενα ιόντα προσκορούουν σε ένα στερεό σώμα, προκαλώντας οπισθοσκέδαση στα άτομα της επιφάνειας του στερεού. Εμφύτευση (implantation) επιτυγχάνεται στην περίπτωση που το ενεργητικό ιόν εισχωρήσει εντός του στερεού. Αφού λάβει χώρα αυτό το γεγονός, δημιουργείται πλάσμα αδρανούς υλικού (συνήθως αργού) στον ενδιάμεσο χώρο μεταξύ στόχου-υποστρώματος. Το πλάσμα αυτό αποκολλά μόρια του στόχου διαμέσου των κρούσεων που προκαλεί με αυτό. Τα μόρια της πρόσκρουσης στη συνέχεια κατασταλλάζουν επάνω στο υπόστρωμα.

Ο στόχος του υλικού προς εναπόθεση τοποθετείται στο ηλεκτρόδιο της καθόδου
ενώ το υπόστρωμα στο οποίο εναποτίθεται το υλικό του στόχου τοποθετείται στην άνοδο. Τα θετικά ιόντα του πλάσματος επιταχύνονται προς το αρνητικά
πολωμένο ηλεκτρόδιο. Η τάση η οποία εφαρμόζεται στην άνοδο μπορεί να φέρει τα
ιόντα να έχουν ενέργειες ακόμα και αρκετές χιλιάδες eV καθώς προσπίπτουν στον
στόχο. Καθώς τα ιόντα προσπίπτουν στο στόχο, εξάγουν άτομα του στόχου τα οποία
με τη σειρά τους μπορούν να κινηθούν μέσα στο πλάσμα και να συμπυκνωθούν
στην επιφάνεια του υποστρώματος. Εξαιτίας της φύσης της διεργασίας, είναι επιτακτική η αηωημότητα του υλικού στόχου. Σε περίπτωση που το υλικό είναι μη αγώγιμο, δύναται να χρησιμοποιηθεί το RF sputtering.

Τα νανοσωματίδια παράγονται με τη μέθοδο DC magnetron sputtering, ακολουθούμενη από συμπύκνωση αέριας φάσης. Εν τέλει τα νανοσωματίδια που παράγονται τείνουν να κατέχουν ένα επιπλέον ηλεκτρόνιο. Το πλεονάζον ηλεκτρόνιο επιτρέπει τον ηλεκτροστατικό χειρισμό

τους. Τα νανοσωματίδια μπορούν να επιταχυνθούν προς το υπόστρωμα παράγοντας μια μεγάλη ποικιλία μορφολογιών, όπως μεμονωμένες νησίδες του υλικού εναπόθεσης μέχρι πολύ λεπτά στρώματα επικάλυψης (υμένια).

Το σχήμα των νανοσωματιδίων επηρεάζεται από αρκετές διαφορετικές παραμέτρους. Η κεφαλή του magnetron μπορεί να μετακινηθεί. Μειώνοντας την απόσταση από την κεφαλή μέχρι το πρώτο άνοιγμα εκτόνωσης, μειώνεται η απόσταση και ο χρόνος όπου συμβαίνει η συμπύκνωση. Επομένως, ελαττώνεται το μέσο μέγεθος των νανοσωματιδίων.

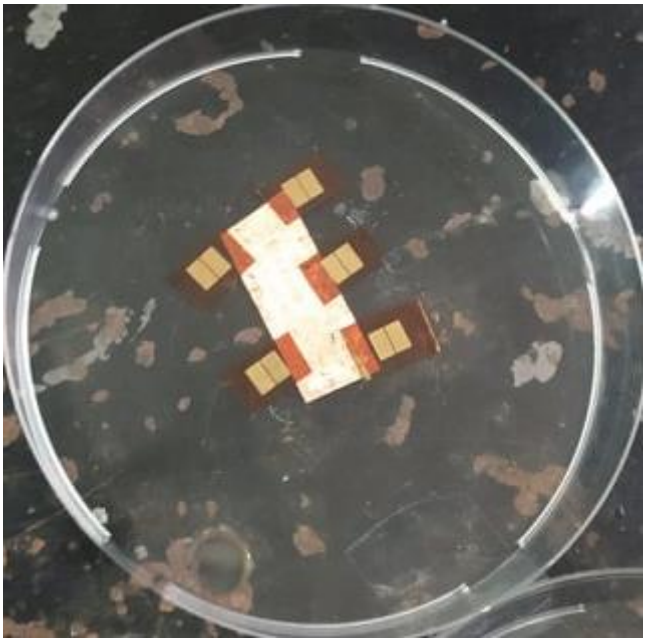
Η πηγή επιτρέπει την εισαγωγή αερίου (αργό) που θα λειτουργήσει ως φορέας των παραγόμενων νανοσωματιδίων. Αυξάνοντας το ρυθμό ροής του αερίου παρατηρείται μείωση του μέσου μεγέθους των παραγόμενων νανοσωματιδίων, εξαιτίας της μείωσης του χρόνου παραμονής τους στη ζώνη συσσωμάτωσης. Λόγω της πίεσης, τα νανοσωματίδια παρασύρονται στον θάλαμο εναπόθεσης και επικάθονται στο δείγμα.

Η ανάπτυξη στερεών από ατμούς (συμπύκνωση) στηρίζεται στο μετασχηματισμό φάσης από μια μη – κρυσταλλική σε μια κρυσταλλική κατάσταση. Η συμπύκνωση αρχίζει με την δημιουργία πυρήνων. Αφού παραχθούν οι ατμοί, κατά την διάρκεια της διαστολής ψύχονται, ξεπερνούν το όριο συνύπαρξης υγρού / στερεού και γίνονται υπέρκοροι.

Τα σωματίδια έπειτα πυρηνοποιούνται μέσω της συσσωμάτωσης και τελικά συσσωματώνονται και τα μικρά σωματίδια σε κόκκους και με την σειρά τους οι κόκκοι σε σμήνη.

Η ανάπτυξη και η πυρηνοποίηση των σμηνών διακόπτεται καθώς τα σμήνη εισέρχονται μέσω μιας μικρής οπής στη ζώνη φιλτραρίσματος, όπου επικρατεί σημαντικά χαμηλότερη πίεση. Σε αυτή τη φάση προκύπτουν τα νανοσωματίδια που θα εναποτεθούν στο υπόστρωμα. Οι συνθήκες εναπόθεσης που επηρεάζουν την επιφανειακή πυκνότητα των νανοσωματιδίων και την κατανομή του μεγέθους τους, είναι η θερμοκρασία του υποστρώματος, ο χρόνος εναπόθεσης, η ισχύς της εναπόθεσης και ο ρυθμός ροής του αδρανούς αερίου.

Τα νανοσωματίδιά μας είναι κατασκευασμένα σε εύκαμπτο υπόστρωμα πολυιμιδίου με ένα στρώμα εναποθεμένου πολυμερούς PIBMA. Τα διαλύματα των πολυμερών παρασκευάστηκαν με διάλυση 10 mg πολυμερούς σε 90 mg διαλύτη (PGMEA) για το PIBMA.
Παρακάτω παρατίθεται μια φωτογραφία από τους αισθητήρες:



Εικόνα 28. Οι αισθητήρες με τα νανοσωματίδια

7.2. Μεθοδολογία Εκτέλεσης Πειραμάτων Χαρακτηρισμού

Τα πειράματα χαρακτηρισμού ηλεκτροχημικών αισθητήρων εκτελέστηκαν στην διάταξη χαρακτηρισμού που έχει κατασκευαστεί στον Τομέα Φυσικής του Ε.Μ.Π. Η διάταξη περιλαλβάνει ελεγκτές ροής αερίου της εταιρείας Brooks για τον καθορισμό της συγκέντρωσης υγρασίας όπως περιγράφεται αναλυτικά παρακάτω, καθώς και συσκευή μέτρησης της αντίστασης των αισθητήρων.

Ο αισθητήρας νανοσωματιδίων (ΝΣ) με την επικάλυψη πολυμερικού υμενίου PIBMA, πάχους ~500 nm, μετρήθηκε σε διαφορετικές συγκεντρώσεις υγρασίας στους 25 ˚C.

Οι ατμοί της υγρασίας παράγονται από νερό (dH_2O), το οποίο εισάγεται σε γυάλινους Bubbler εντός των οποίων επιβάλλεται ροή αδρανούς αερίου N2 , μετρούμενο σε sccm (flow rate: cubic centimeters per minute), και με δυνατότητα μέγιστης ροής N2 της τάξης των 1300 sccm. Πριν την εισαγωγή των ατμών της υγρασίας στον θάλαμο μέτρησης αισθητήρων, οι ατμοί ενώνονται με την κεντρική, αδιάλειπτη και σταθερή παροχή N2 1000 sccm (diluting gas flow), η οποία γίνεται μέσω του Ελεγκτή ροής Μάζας MFC 1. Η ακριβής ρύθμιση της ροής του αερίου (μέσω

MFCs) αλλά και της θερμοκρασίας του θαλάμου (± 0.5 ˚C) είναι ικανή να προσδιορίσει την συγκέντρωση των ατμών που οδηγούνται (μέσω της ροής N2) στον θάλαμο μέτρησης των αισθητήρων.

Πιο αναλυτικά η ακολουθία των βημάτων πειραματικού χαρακτηρισμού των χημικών αισθητήρων έχει ως εξής:

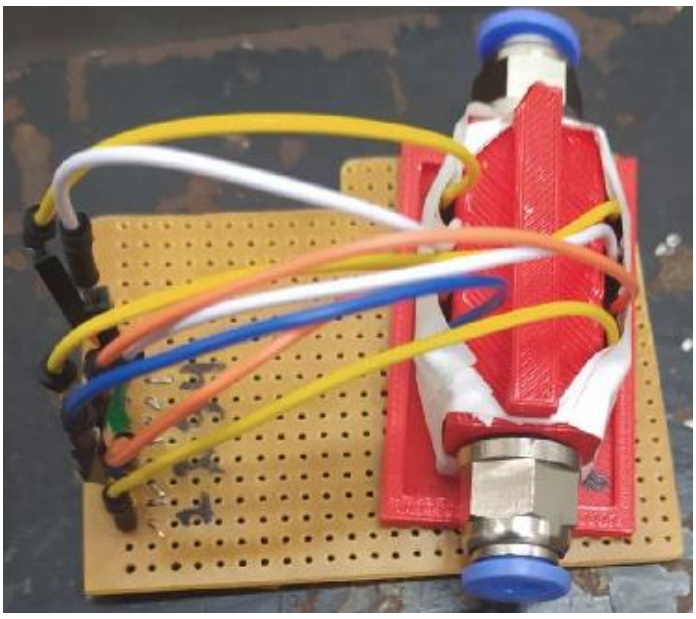
1. Ξήρανση των αισθητήρων με σταθερή ροή αζώτου 1000 sccm μέσω του MFC 1 για χρονικό διάστημα 20 min

2. Παροχή επιλεγμένης ροής αζώτου από τους MFC 2 & MFC 3 διαμέσου του bubbler για χρονικό διάστημα 10 min

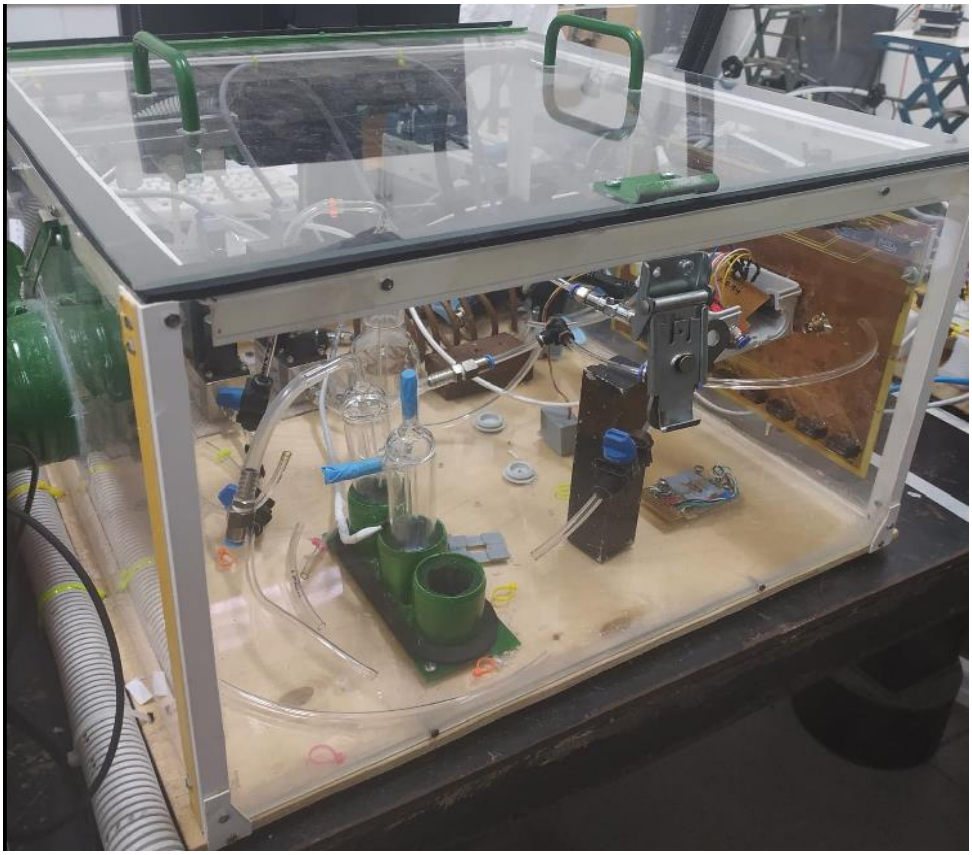
3. Ξήρανση των αισθητήρων με σταθερή ροή αζώτου 1000 sccm μέσω του MFC 1 για χρονικό διάστημα 10 min.

Στην συνέχεια πραγματοποιήσαμε μετρήσεις με 8 αντιστάσεις, εκ των οποίων η μία ήταν σταθερή και οι υπόλοιπες μεταβαλλόμενες (ποτενσιόμετρα).

Παραθέτουμε κάποιες φωτογραφίες από το εργαστήριο:



Εικόνα 29. Η 3d printed πλατφόρμα τοποθέτησης εργαστηρίων



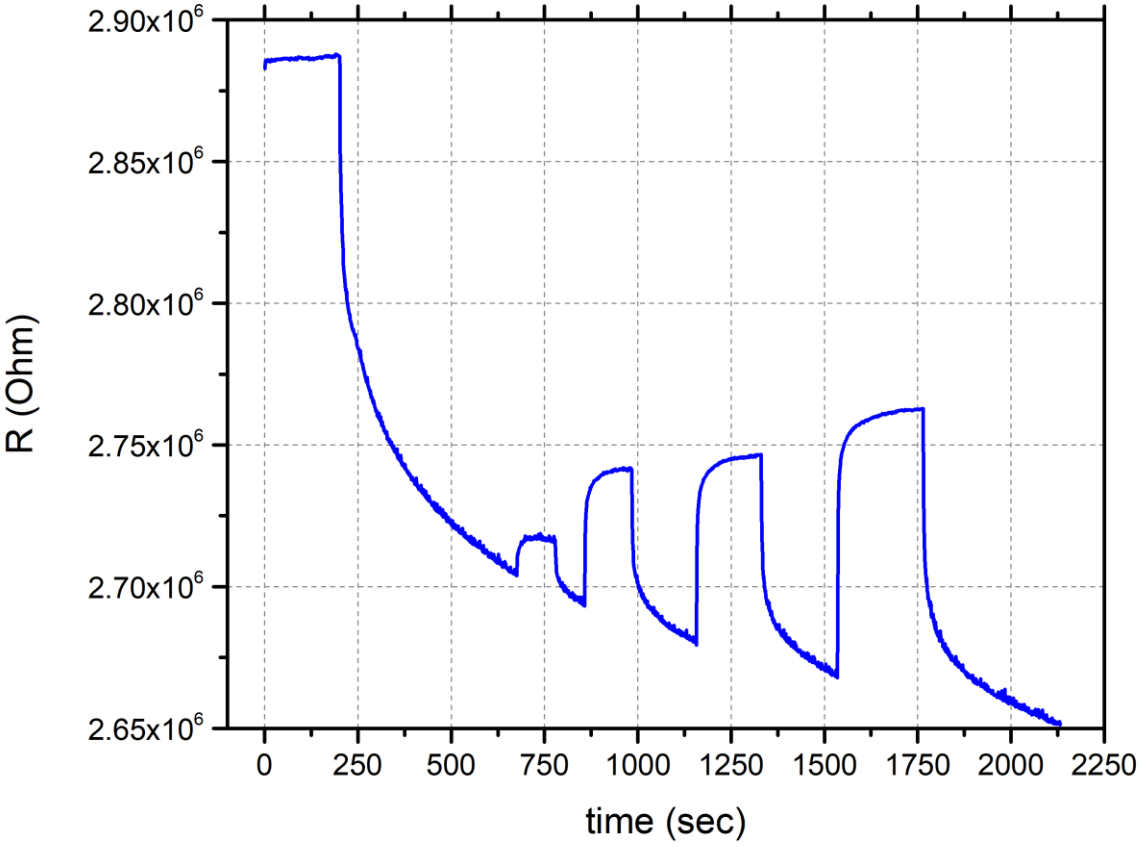
Εικόνα 30. Το δοχείο που τοποθετούνται οι αισθητήρες προς μέτρηση, το οποίο εμπεριέχει και τους bubblers

Εικόνα 31.Το σύστημα μέτρησης αντιστάσεων Keithley 2400

7.3. Αποτελέσματα Μετρήσεων

7.3.1. Μετρήσεις Αισθητήρων Υγρασίας

Στο παρακάτω διάγραμμα παραθέτουμε τα δεδομένα απο την πρώτη μέτρηση του αισθητήρα υγρασίας με το εμπορικό σύστημα Keithley 2400.



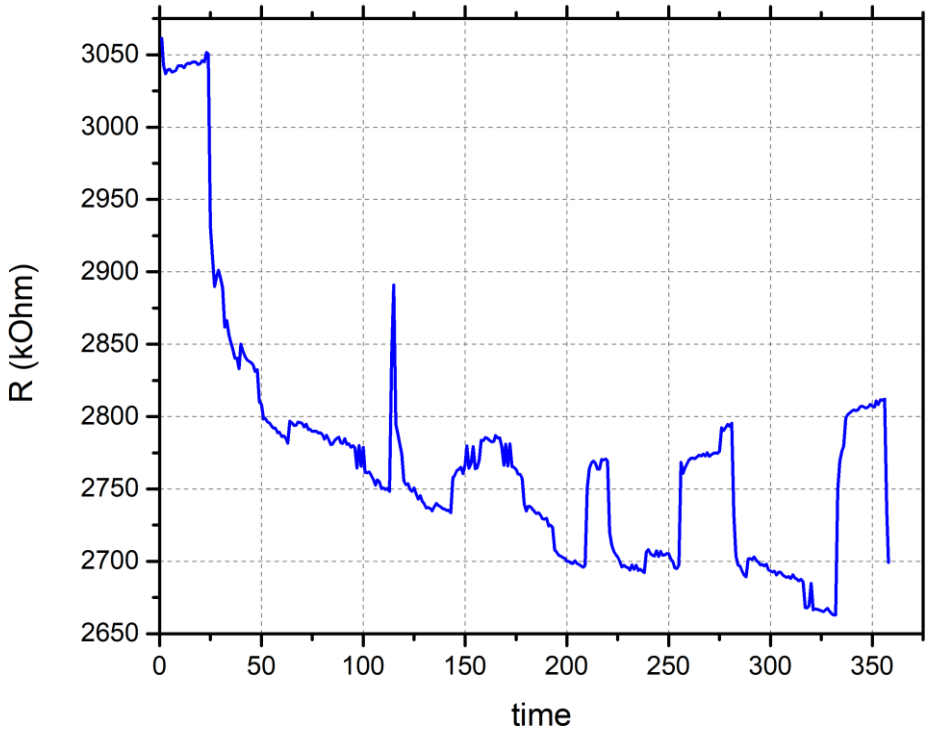
Διάγραμμα 1. Αποτελέσματα μέτρησης πρώτου αισθητήρα με το σύστημα Keithley

Όσο το σύστημα τροφοδοτείται με άζωτο, η αντίσταση πέφτει. Ο λόγος γι' αυτό το φαινόμενο έγκειται στο γεγονός ότι το άζωτο αφυδατώνει την ατμόσφαιρα, δηλαδή προκαλεί πτώση στην υγρασία του περιβάλλοντος. Ως αποτέλεσμα, οι αισθητήρες συρρικνώνονται και τα νανοσωματίδια έρχονται πιο κοντά. Αυτό οδηγεί σε πτώση της αντίστασης.

Παρατηρούμε στην φωτογραφία την πτώση της αντίστασης, αλλά παράλληλα παρατηρούμε τέσσερις κορυφές. Αυτές οι κορυφές προκύπτουν απο την εισαγωγή της υγρασίας προοδευτικά δια μέσου των bubblers. Η αύξηση της υγρασίας προκαλεί και αύξηση της αντίστασης.

Οι υγρασίες που εισαγάγαμε στο σύστημα ήταν της τάξης των 3,5% , 10,5% , 14% και 21% αντιστοίχως.

Στο κάτωθι διάγραμμα παρατηρούμε την αντίστοιχη μέτρηση απο το σύστημα που κατασκευάσαμε.



Διάγραμμα 2. Αποτελέσματα μέτρησης πρώτου αισθητήρα με το σύστημα μας

Αρχικά, παρατηρούμε μία σχετικά υψηλή κορυφή, λίγο μετά την τιμή 100 στον άξονα του χρόνου. Αυτό προκύπτει απο λανθασμένη εισαγωγή υγρασίας και την αγνούμε.

Επιπλέον, παρατηρούμε μία διαφοροποίηση στο πεδίο του χρόνου σε σχέση με το προηγούμενο διάγραμμα. Ο λόγος είναι πως, στο εμπορικό σύστημα υπάρχει σαφής επιλογή της δειγματοληψίας άρα μπορούμε να κατασκευάσουμε ακριβές διάγραμμα. Στο δικό μας σύστημα, δεν μπορούμε να παραμετροποιήσουμε τον χρόνο της κάθε μέτρησης. Για την ακρίβεια, μπορούμε να παραμετροποιήσουμε την συχνότητα δειγματολήψίας, την οποία είχαμε ορίσει στο μέγιστο και μπορούμε να αλλάξουμε τον χρόνο αναμονής για εύρεση ανοιχτοκυκλώματος.

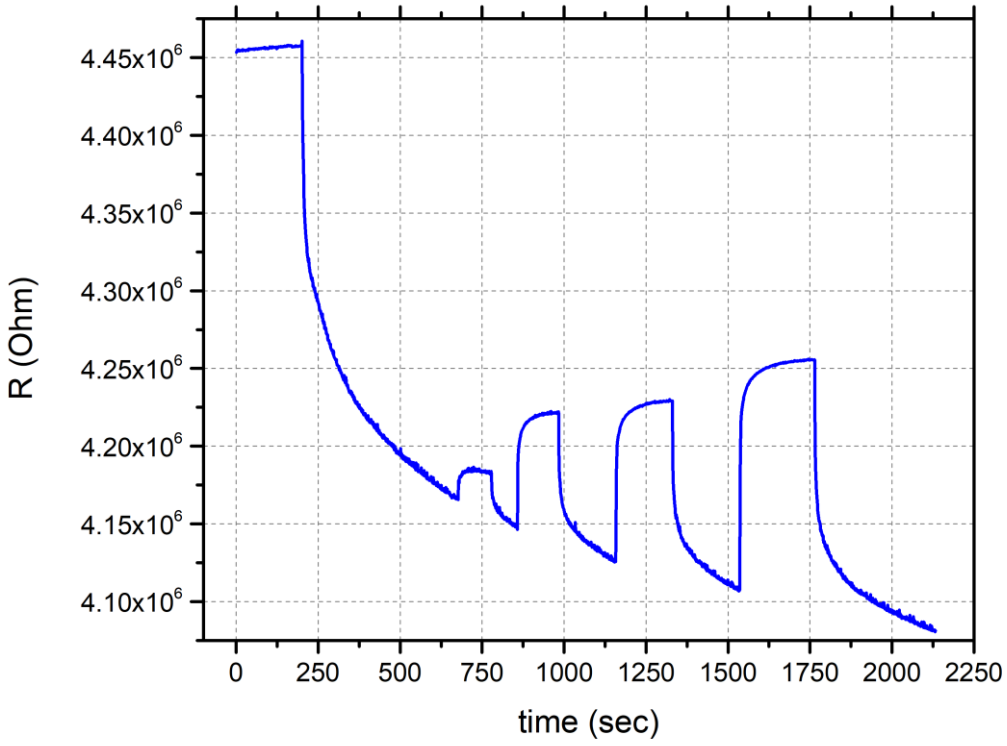
Συνεπώς, ο άξονας του χρόνου δεν είναι σε συμμετρία με τον άξονα του διαγράμματος του εμπορικού συστήματος. Για την ερμηνεία της ακρίβειας των αποτελεσμάτων μπορούμε όμως να φανταστούμε πως στα σημεία που έχουμε κορυφές, είναι ταυτόσημες οι χρονικές περίοδοι. Με άλλα λόγια, οι κορυφές του πάνω σχεδιαγράμματος με αυτές του κάτω παρουσιάζονται την ίδια στιγμή, ασχέτως εαν αυτό δεν αντικατοπτρίζεται στον άξονα του χρόνου σε απόλυτο μέγεθος.

Απο την σύγκριση των διαγραμμάτων μπορούμε να εξάγουμε τα εξής συμπεράσματα:

- Υπάρχουν μικρές αποκλίσεις στην αντίσταση στο σύστημά μας, σε σχέση με το εμπορικό σύστημα. Επιπλέον πρέπει να συνυπολιγιστεί πως οι αντιστάσεις των αισθητήρων που μετρήθηκαν είναι πέρα απο τις προδιαγραφές του αναλογικού κυκλώματος. Υπενθυμίζουμε πως λόγω των τιμών των παθητικών εξαρτημάτων που έχει το αναλογικό κύκλωμα, είναι κατασκευασμένο να μετράει αντιστάσεις στο εύρος 300kΩ εώς 1MΩ.
- Παρότι οι κορυφές των τοπικών μέγιστων δεν διαφέρουν πολύ, βλέπουμε πως στο σύστημά μας υπάρχει πιο έντονος θόρυβος, πιο τυχαία κατανεμημένος σε σχέση με το εμπορικό σύστημα. Πρέπει να συνυπολογίσουμε πως στην μέτρηση με το εμπορικό σύστημα χρησιμοποιήθηκαν ακριβά bnc connectors που παρουσιάζουν ισχυρή μόνωση απο εξωτερικούς παράγοντες που ενδέχεται να επιφέρουν θόρυβο, ενώ στο δικό μας σύστημα χρησιμοποιήθηκαν απλά jumper wires.

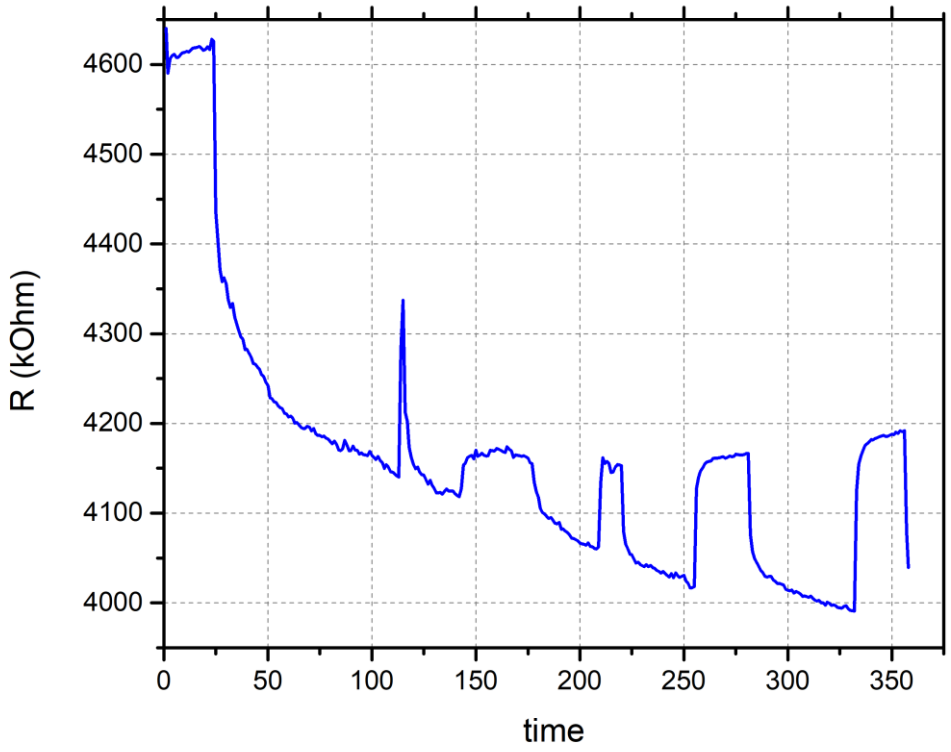
Ταυτόχρονα με την μέτρηση του ενός αισθητήρα, πραγματοποιήθηκε και μέτρηση σε δεύτερο αισθητήρα, αξιοποιώντας την δυνατότητα του συστήματος να διαβάζει μετρήσεις σε πολλαπλούς αισθητήρες συγχρόνως.

Τα αποτελέσματα απο το εμπορικό μετρητικό όργανο παρατίθενται παρακάτω.



Διάγραμμα 3. Αποτελέσματα μέτρησης δεύτερου αισθητήρα με το σύστημα Keithley

Ακολουθούν τα αποτελέσματα απο το δικό μας σύστημα.



Διάγραμμα 4. Αποτελέσματα μέτρησης δεύτερου αισθητήρα με το σύστημα μας

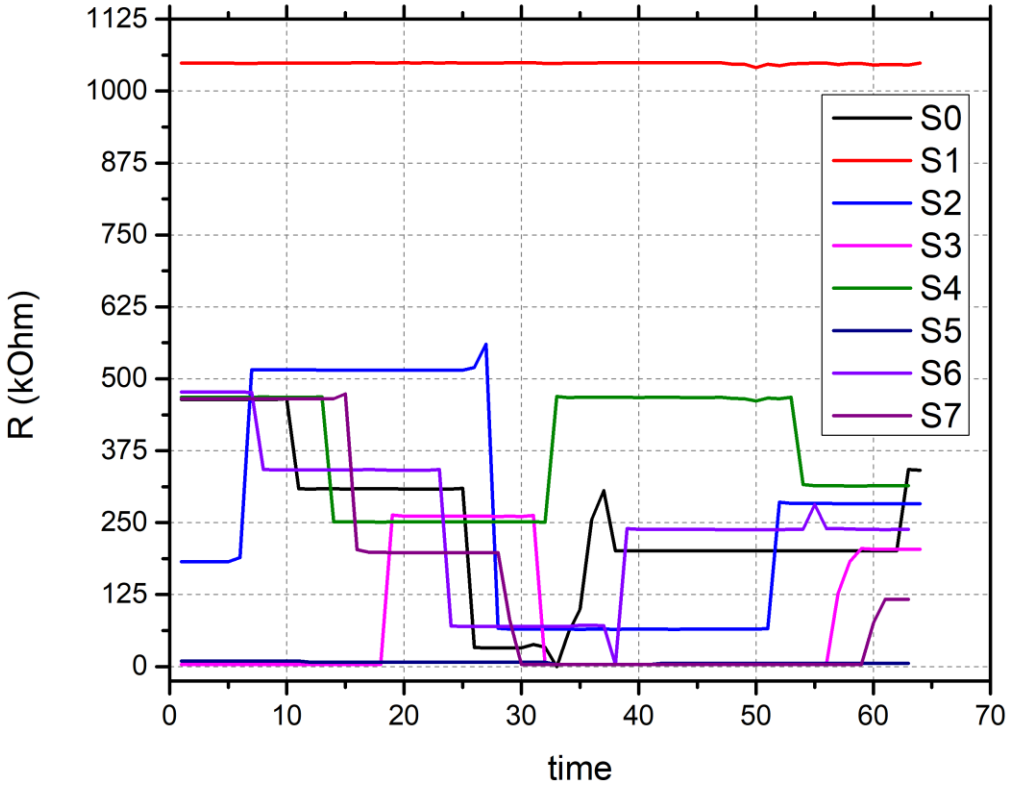
Ακόμα και σε αυτή την περίπτωση, όπου οι αντιστάσεις των αισθητήρων είναι μεγαλύτερες και υπερβαίνουν σε ακόμα μεγαλύτερο βαθμό το όριο ανάγνωσης του αναλογικού κυκλώματος, παρατηρούμε τα ίδια αποτελέσματα με την πρώτη μέτρηση.

7.3.2. Μετρήσεις Κοινών Αντιστάσεων

Στο επόμενο εδάφιο θα προβούμε σε μετρήσεις κοινών αντιστάσεων. Οι επτά απο αυτές είναι μεταβαλλόμενες, δηλαδή ποτενσιόμετρα, ενώ η μία είναι σταθερή. Όλες οι αντιστάσεις ήταν συνδεδεμένες ταυτόχρονα, αξιοποιώντας την δυνατότητα του συστήματος για πολλαπλές μετρήσεις.

Αντίθετα με τους αισθητήρες υγρασίας, το εύρος των τιμών των αντιστάσεων ήταν εντός ή πολύ κοντά στο όριο μετρήσεως του συστήματός μας.

Ας δούμε το γράφημα με τα αποτελέσματα των μετρήσεων.



Διάγραμμα 5. Αποτελέσματα μέτρησης συστοιχίας κοινών αισθητήρων

Παρατηρούμε πως η αντίσταση S1 παρέμεινε σταθερή σε όλη την διάρκεια της μετρητικής διαδικασίας με μικρό ποσοστό θορύβου προς το τέλος.

Στις υπόλοιπες μεταβαλλόμενες αντιστάσεις παρατηρούμε την μεταβολή στην αντίσταση καθώς περιστρέφουμε τα ποτενσιόμετρα. Επίσης είναι σημαντικό να παρατηρήσουμε πως οι μεταβάσεις είναι ομαλές και δεν συναντούμε θόρυβο.

8. Συμπεράσματα και Μελλοντικές Προοπτικές

Από τα δεδομένα των μετρήσεων που έλαβαν χώρα, καταλήγουμε στο ότι το σύστημά μας είναι μία πολύ πιο οικονομική εναλλακτική έναντι του εμπορικού, εάν αναλογιστούμε ότι το κόστος του εμπορικού κυμαίνεται από 5.000 έως 10.000 ευρώ, έναντι συστήματος που σχεδιάσαμε, του οποίου το κόστος είναι κάτω των 100 ευρώ για να παραχθεί. Η απόδοση είναι αρκετά ικανοποιητική όσον αφορά τις μετρήσεις ακόμα και σε αντιστάσεις που είναι εκτός των προδιαγραφών του αναλογικού κυκλώματος. Παρόλα αυτά, η διακριτική ικανότητα του αναλογικού κυκλώματος μπορεί να τροποποιηθεί με την αλλαγή της τιμής του κύριου πυκνωτή του συστήματος.

Στην περίπτωση μετρήσεων αντιστάσεων εκτός του μετρητικού ορίου του συστήματος, οι τιμές των ακρότατων του διαγράμματος αντιστάσεων απέχουν λίγο απο τις τιμές του εμπορικού συστήματος. Φυσικά, έχουμε και αισθητά περισσότερο θόρυβο, όμως αυτό δεν ήταν εμπόδιο στις ακρότατες τιμές.

Στην περίπτωση μετρήσεων αντιστάσεων εντός του μετρητικού ορίου του συστήματος, οι τιμές των αντιστάσεων είναι πάρα πού ακριβείς. Επίσης ο θόρυβος κυμαίνεται απο καθόλου εώς ελάχιστος και σε κάθε περίπτωση είναι ανεπαίσθητος.

Άλλο ένα θετικό σημείο της υλοποίησής μας είναι η φορητότητα του συστήματος. Το εμπορικό σύστημα είναι μεγάλο, βαρύ, απαιτεί τροφοδοσία και σύνδεση με υπολογιστή. Το δικό μας συστημα είναι μικρό, φορητό και ελαφρύ. Η τροφοδοσία είναι χαμηλής τάσης και θα μπορούσε να παραχθεί και απο εναλλακτικές πηγές ενέργειας (πχ φωτοβολταϊκό σύστημα) χωρίς να απαιτεί υποδομές δικτύου παροχής ηλεκτρικής ενέργειας.

Τέλος, το δικό μας σύστημα ενώ στηρίζεται στην ταυτόχρονη χρήση ηλεκτρονικού υπολογιστή, θα μπορούσε παράλληλα να τροποποιηθεί και να λειτουργεί αυτόνομα, αποθηκεύοντας τα δεδομένα σε φορητό αποθηκευτικό μέσο (πχ κάρτα sd). Περισσότερες λεπτομέρειες παρατίθενται στο εδάφιο για τις βελτιώσεις του συστήματος.

Το σύστημά μας είναι τελείως ανοιχτό και παραμετροποιήσιμο. Ως αποτέλεσμα, μπορεί να τροποποιηθεί με πληθώρα τρόπων, αναλόγως την συμπεριφορά που θέλουμε να επιτύχουμε την δεδομένη χρονική στιγμή.

Παραθέτουμε μία λίστα με πιθανές βελτιώσεις και ιδέες

Βασισμένες στο Υλικό (hardware)

- Τροποποίηση στην τροφοδοσία με χρήση φωτοβολταϊκών κυττάρων ή/και μπαταρίας.
- Υλοποίηση rf λύσεων για αποστολή δεδομένων. Αυτή η τροποποίηση θα μπορεί να επιτρέπει στο κύκλωμα να είναι πλήρως αυτόνομο, και να επικοινωνεί με υπολογιστή ο οποίος θα είναι απομακρυσμένος. Σε περίπτωση που το σύστημά μας είναι στην ύπαιθρο, δεν είναι δυνατή η χρήση υπολογιστή. Χρησιμοποιώντας ασύρματη σύνδεση όμως, η επικοινωνία με υπολογιστή θα είναι εφικτή. Τα κόστη στην αγορά ασύρματων συστημάτων έχει μειωθεί κατά πολύ και η διαθεσιμότητα έχει αυξηθεί τα τελευταία χρόνια. Ενδεικτικά, μπορούμε να αναφέρουμε bluetooth λύσεις (χρησιμοποιώντας hc-06 ή esp8266) και wifi λύσεις (esp32). Οι παραπάνω λύσεις αφορούν σε περιπτώσεις όπου ο ηλεκτρονικός υπολογιστής που καταγράφει βρίσκεται σε κοντινή απόσταση απο το κύκλωμα. Παρ'όλα αυτά, υπάρχουν και rf λύσεις που μπορούν να επικοινωνήσουν σε απόσταση χιλιομέτρων – όπως για παράδειγμα το lora mesh network. Οι εντολές σε αυτή την περίπτωση θα μπορούν να στέλνονται στον μικροελεγκτή ασύρματα.
- Υλοποίηση χωρίς υπολογιστή. Στην δική μας περίπτωση, στέλναμε τις εντολές στον μικροελεγκτή μέσω υπολογιστή και σειριακής θύρας. Ωστόσο, αυτό δεν μας εμποδίζει καθόλου στο να παρακάμψουμε τελείως τον υπολογιστή και να πετυχαίνουμε καταγραφή των αποτελεσμάτων σε μία κάρτα sd. Πλέον αποδεσμευόμαστε απο την ανάγκη διασύνδεσης με υπολογιστή (ενσύρματα ή ασύρματα) και το κύκλωμα λειτουργεί αυτόνομα. Οι εντολές θα μπορούν να στέλνονται στον μικροελεγκτή ενσύρματα ή ασύρματα, ακόμα και με την χρήση δικτύου κινητής τηλεφωνίας με χρήση gsm shield, που υλοποιεί gprs/gsm2.0 stack.

Βασισμένες στο Λογισμικό (software)

- Δημιουργία λειτουργίας real time plotting στο software συλλογής δεδομένων. Κάθε τιμή που καταφθάνει στον υπολογιστή θα μπορεί να σχεδιάζεται σε πραγματικό χρόνο, χρησιμοποιώντας μία κατάλληλη βιβλιοθήκη. Μιας και το πρόγραμμα έχει γραφεί σε python, μία πολύ γνωστή βιβλιοθήκη για αυτό τον σκοπό σε αυτήν την γλώσσα είναι η matplotlib.
- Χρήση λειτουργίας μέτρησης σε συγκεκριμένο χρόνο. Ο χρήστης θα μπορεί να θέτει ένα χρονικό παράθυρο αναφορικά με τη συχνότητα των μετρήσεων. Αυτό μπορεί να υλοποιηθεί είτε στο software του μικροελεγκτή (χρησιμοποιώντας, για παράδειγμα, κάποιον άλλο timer), είτε στο software ελέγχου και συλλογής δεδομένων – στέλνοντας εντολή για μονή μέτρηση περιοδικά και ανα συγκεκριμένο χρονικό διάστημα. Η δυσκολία της δεύτερης υλοποίησης έγκειται στο ότι θα πρέπει να συνυπολογιστούν και οι χρόνοι απόκρισης του μικροελεγκτή.

Βιβλιογραφία

[1] An, J., Song, K., Yang, J., & Cao, P. (2012). *The research and design of the data acquisition system and the control system of KTX. 2012 18th IEEE-NPSS Real Time Conference.* doi:10.1109/rtc.2012.6418100

[2] Wang, X.-L., Zhang, J.-Y., Wang, W.-L., & Zhang, X.-D. (2010). *Noise analysis for electronic circuit using Multisim. 2010 2nd IEEE International Conference on Information Management and Engineering.* doi:10.1109/icime.2010.5478022

[3] Yang, S. H.; Drabold, D. A.; Adams, J. B.; Sachdev, A. *First-Principles Local-Orbital Density-Functional Study of Al Clusters.* Phys. Rev. B 1993, 47 (3), 1567–1576

[4] Rao, C. N. R.; Kulkarni, G. U.; Thomas, P. J.; Edwards, P. P. *Metal Nanoparticles and Their Assemblies.* Chem. Soc. Rev. 2000, 29 (1), 27–35

[5] Sberveglieri, G.; Faglia, G.; Perego, C.; Nelli, P.; Marks, R. N.; Virgili, T.; Taliani, C.; Zamboni, R. *Hydrogen and Humidity Sensing Properties of C60 Thin Films.* Synth. Met. 1996, 77 (1–3), 273–275

[6] Franke, M. E.; Koplin, T. J.; Simon, U. *Metal and Metal Oxide Nanoparticles in Chemiresistors: Does the Nanoscale Matter?* Small. 2006, pp 36–50

[7] Myers, M.; Cooper, J.; Pejcic, B.; Baker, M.; Raguse, B.; Wieczorek, L. *Functionalized Graphene as an Aqueous Phase Chemiresistor Sensing Material.* Sensors Actuators, B Chem. 2011, 155 (1), 154–158

[8] Zabet-Khosousi, A.; Trudeau, P. E.; Suganuma, Y.; Dhirani, A. A.; Statt, B. Metal to Insulator Transition in Films of Molecularly Linked Gold Nanoparticles. Phys. Rev. Lett. 2006, 96 (15)

[9] Abdu I.O., Taleb M.M., *Measurement systems: Characteristics and Models*, European Scientific Journal March 2014 edition vol.10, No.9 ISSN: 1857 – 7881 (Print) e - ISSN 1857- 7431

[10] Καραδήμας, Π. *Ανάπτυξη Μετρητικού Συστήματος Ανάγνωσης Συστοιχίας Αισθητήρων Παραμόρφωσης,* 2017

[11] Σκοτάδης, Ε. *Αυτο-Οργάνωση Νανοσωματιδίων Με Εφαρμογές Σε Χημικούς Αισθητήρες,* 2013

[12] Κανάρης, Α.Ι., *Αναγνώριση φυτοφαρμάκων μέσω ανάπτυξης χημικών αισθητήρων βασισμένων σε νανοσωματίδια πλατίνας,* 2020

[13] Βασιλοπούλου, Φ., Μάρκου, Θ., Μυστιλίδης, Χ., Φερίκογλου, Α., *Σχεδιασμός Συστήματος Μέτρησης Μήτρας Αισθητήρων Πίεσης,* 2019

Παράρτημα - Κώδικας

Κώδικας Ενσωματωμένου Συστήματος (Arduino embedded software)

```
/***** CONSTANTS *****/
const int BAUD_RATE = 9600;
const int INTERRUPT_PIN = 3;
const int S0_PIN = 4;
const int S1_PIN = 5;
const int S2_PIN = 6;
const int OE_BAR_PIN = 22;
const int NUM_OF_SENSORS = 8;

/***** ANALOG CIRCUIT CONSTANTS *****/
const double CAPACITOR_VALUE = 400; //pF
const double GAIN_VALUE = 3.5222672065;
const double R_BIAS_VALUE = 98.7; //kΩ

/***** GLOBAL VARIABLES *****/
int number_of_samples = 500;
int sensor_coord;
boolean full_mode = false;

unsigned long overflows = 0;
unsigned long ticks = 0;
double period = 0;

//variables for determining if a resistance cannot be measured
unsigned long timestamp = 0;
unsigned long measurement_timeout = 5000;
boolean sensor_corruption[8];
boolean ISR_unavailable = false;


volatile int pulse_counter = 0;
volatile boolean interrupts_enabled = false;
volatile boolean measurement_finished = false;



void setup()
{
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN),
count_time_of_k_pulses, RISING);

  digitalWrite(OE_BAR_PIN, HIGH);
  pinMode(OE_BAR_PIN, OUTPUT);
```

```
  pinMode(S0_PIN, OUTPUT);
  pinMode(S1_PIN, OUTPUT);
  pinMode(S2_PIN, OUTPUT);

  TIMER1_init();

  Serial.begin(BAUD_RATE);
  while (! Serial);

  for (int i=0; i<=7; i++)
  {
    sensor_corruption[i] = false;
  }
}

void loop()
{

  if (Serial.available())
  {
    /***** DEFINE MEASUREMENT MODE *****/
    String input = Serial.readString();
    if (input[0] == 'r' && interrupts_enabled == false)
    {
      if (input[1] == 'f')
      {
        full_mode = true;
        sensor_coord = 0;
      }
      else
      {
        full_mode = false;
        sensor_coord = input[2] - '0';
      }

      select_sensor();

      pulse_counter = 0;
      TIMER1_clear();

      interrupts_enabled = true;
    }

    else if (input[0] == 'C' || input[0] == 'c')
    {
      full_mode = false;
      measurement_finished = false;
      interrupts_enabled = false;
    }
```

```
  } //Serial.available portion ends here

  if(interrupts_enabled && millis() - timestamp > measurement_timeout)
  {
    sensor_corruption[sensor_coord] = true;
    measurement_finished = true;
    ISR_unavailable = true;
  }
  //-------------------------

  if (measurement_finished)
  {

    if (!ISR_unavailable)
    {
        ticks = (overflows * 65536) + TCNT1;
        period = (ticks * 0.0625) / (number_of_samples - 1);

        sensor_corruption[sensor_coord] = false;
    }

      TIMER1_print_results();
      ISR_unavailable = false;
      if (full_mode)
      {
        sensor_coord++;
        select_sensor();

        if (sensor_coord == NUM_OF_SENSORS)
        {

          sensor_coord = 0;
        }
      }

       pulse_counter = 0;
       TIMER1_clear();
       interrupts_enabled = true;
       measurement_finished = false;
       timestamp = millis();

  }
 } //loop ends

/***** INTERRUPT ROUTINES *****/

void count_time_of_k_pulses()
{
  if (interrupts_enabled)
```

```
  {
    pulse_counter++;

    if (pulse_counter == 1)
      TIMER1_start();

    if (pulse_counter == number_of_samples)
    {
      TIMER1_stop();

      measurement_finished = true;
      interrupts_enabled = false;
    }
  }
}

ISR (TIMER1_OVF_vect)
{
  overflows++;
}

/***** FUNCTION FOR SENSOR SELECTION *****/

void select_sensor()
{
  digitalWrite(OE_BAR_PIN, HIGH);
  digitalWrite(S2_PIN, ((sensor_coord & 4) == 4) ? HIGH : LOW );
  digitalWrite(S1_PIN, ((sensor_coord & 2) == 2) ? HIGH : LOW );
  digitalWrite(S0_PIN, ((sensor_coord & 1) == 1) ? HIGH : LOW );
  digitalWrite(OE_BAR_PIN, LOW);
}

/***** TIMER1 FUNCTIONS (MODIFIED) *****/

void TIMER1_init()
{
  TCCR1A  = 0;
  TCCR1B  = 0;
  TIMSK1 |= (1 << TOIE1);
}

void TIMER1_start()
{
   TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);
}

void TIMER1_stop()
{
  TCCR1B  = 0;
```

```
}

void TIMER1_clear()
{
  TCNT1     = 0;
  overflows = 0;
}

void TIMER1_print_results()
{
  Serial.println("----------------------------------------");
  Serial.print("SENSOR COORDINATE         = ");
  Serial.println(sensor_coord);

  if (sensor_corruption[sensor_coord])
  {
    Serial.println("This resistance is open circuited!!!");
  }
  else
  {
    /*
    Serial.print("NUMBER OF PULSES MEASURED = ");
    Serial.println(pulse_counter);

    Serial.print("TIMER1 VALUE              = ");
    Serial.println(TCNT1);

    Serial.print("TIMER1 OVERFLOWS          = ");
    Serial.println(overflows);

    Serial.print("TICKS                     = ");
    Serial.println(ticks);

    Serial.print("MEASURED PERIOD           = ");
    Serial.print(period);
    Serial.println(" US");
    */
    Serial.print("MEASURED RESISTANCE       = ");
    double resistanse = ((period * GAIN_VALUE * 1000) / (4 *
CAPACITOR_VALUE)) - R_BIAS_VALUE; //kΩ
    Serial.print(resistanse);
    Serial.println(" kOhms");

    Serial.println("----------------------------------------");
  }
}
```

Κώδικας Python (data acquisition software)

```
import tkinter as tk
import tkinter.ttk as ttk
import serial.tools.list_ports
from tkinter import scrolledtext
from time import sleep
from datetime import datetime


#global variables
HEIGHT = 800
WIDTH = 800

baudRate = 9600

ser = None
after_id = None
found_coordinate = None
coordinate = None
dropdown_value = None
modeOfOperation = None


serial_ready = False
sensor = [[] for i in range(8)]


# --- functions ---

def serial_ports():
    return serial.tools.list_ports.comports()


#when the user selects one serial port from the combobox, this function will
execute
def on_select(event=None):
    global ser
    global serial_ready
    COMPort = cb.get()
    string_separator = "-"
    COMPort = COMPort.split(string_separator, 1)[0]
    COMPort = COMPort[:-1]
    ser = serial.Serial(port = COMPort, baudrate=9600, timeout=0.1)
    sleep(2)
    button_all['state'] = 'normal'
    serial_ready  = True
```

```
def readSerial():
    global after_id
    global found_coordinate
    global coordinate
    global sensor
    global stop_
    if stop_ == True:
      stop_=False
      for myList in sensor:
        myList.clear()
    while ser.in_waiting:
        try:
            ser_bytes = ser.readline()
            ser_bytes = ser_bytes.decode("utf-8")
            text.insert("end", ser_bytes)
            if vsb.get()[1]==1.0:
               text.see("end")
            if "SENSOR COORDINATE" in ser_bytes:
               found_coordinate = True
               coordinate = int(ser_bytes.split("=")[1].strip())
               print("Coordinate: ",coordinate)
            if "MEASURED RESISTANCE" in ser_bytes and found_coordinate:
               found_coordinate = False
               resistance                                                =
float(ser_bytes.split("=")[1].split("kOhm")[0].strip())
               print("Resistance: ",resistance)
               sensor[coordinate].append(resistance)
            if "This resistance" in ser_bytes and found_coordinate:
               found_coordinate = False
               resistance = 0
               print("Resistance: ",resistance, "  -  Open Circuit")
               sensor[coordinate].append(resistance)
        except UnicodeDecodeError:
            print("UnicodeDecodeError")
    after_id=root.after(50,readSerial)



# this function is triggered, when a value is selected from the dropdown
def dropdown_selection(*args):
   global dropdown_value
   global serial_ready
   dropdown_value = clicked.get()
   if serial_ready == True:
      button_single['state'] = 'normal'
```

```
# this function is triggered, when button 'Measure all Sensors' is pressed,
on frame 2
def measure_all():
    global modeOfOperation
    modeOfOperation = 'A'
    button_export['state']='disabled'
    button_stop['state']='normal'
    ser.write("rf".encode())
    sleep(0.05)
    readSerial()



# this function is triggered, when button 'Measure this Sensor' is pressed,
on frame 2
def measure_single():
    global modeOfOperation
    modeOfOperation = dropdown_value
    print(dropdown_value)
    button_export['state']='disabled'
    button_stop['state']='normal'
    string_to_send = 'r' + ' ' + str(dropdown_value)
    ser.write(string_to_send.encode())
    readSerial()



# this function is triggered, when button 'STOP measurement(s)' is pressed,
on frame 2
def stop_measurement():
    global stop_
    stop_=True
    button_stop['state']='disabled'
    ser.write("c".encode())
    root.after_cancel(after_id)
    button_export['state']='normal'
    button_all['state']='disabled'
    state_of_single_button=button_single['state']
    button_single['state']='disabled'
    print("disabled")
    sleep(1)
    button_all['state']='normal'
    if state_of_single_button == 'normal':
        button_single['state']='normal'
```

```
def namestr(obj):
   return [name for name in globals() if globals()[name] is obj][0]



def export():
  global modeOfOperation
  now = str(datetime.now())
  now = now.split('.', 1)[0]
  now = now.replace(':', '.')
  if modeOfOperation != 'A':
      castedMode = int(modeOfOperation)
      fileName = 'S' + modeOfOperation + '-' + now + '.txt'
      with open(fileName, 'w') as f:
        for item in sensor[castedMode]:
           f.write("%s\n" % item)
      f.close()
      print("File exported")
  elif modeOfOperation == 'A':
      fileName = 'S' + modeOfOperation + '-' + now + '.txt'
      with open(fileName, 'w') as f:
          for myList in range(len(sensor)):
              f.write("\n")
              f.write("\n")
              f.write("\n")
              f.write("%s" % namestr(sensor).strip() + '[' + str(myList) +
']\n')
              f.write("---------------------------------\n")
              for item in sensor[myList]:
                 f.write("%s\n" % item)

# --- functions ---



# --- main ---
root = tk.Tk()
root.title("Sensor Interface")
canvas = tk.Canvas(root, height=HEIGHT, width=WIDTH)
canvas.pack()
```

```
# --- frame 1 ---
frame1 = tk.Frame(root)
frame1.place(relx=0, rely=0.05, relheight=0.03, relwidth=1, anchor='nw')

label0 = tk.Label(frame1, text="Select the COM port that the device is
plugged in: ")
label0.config(font=("TkDefaultFont", 8))
label0.place(relx = 0.1, rely=0.3, relwidth=0.3, relheight=0.5)


cb = ttk.Combobox(frame1, values=serial_ports())
cb.place(relx=0.5, rely=0.5, anchor='center')
cb.bind('<<ComboboxSelected>>', on_select)
# --- frame 1 ---



# --- frame 2 ---
frame2 = tk.Frame(root, bd=5)
frame2.place(relx=0, rely=0.1, relheight=0.07, relwidth=1, anchor='nw')

button_all = tk.Button(frame2, text="Measure all Sensors", bg='#80c1ff',
fg='red', state='disabled', command=measure_all)
button_all.place(relx=0.2, rely=0.5, anchor='center')

label1 = tk.Label(frame2, text="OR, select a single sensor to measure: ")
label1.config(font=("TkDefaultFont", 9))
label1.place(relx = 0.32, rely=0.3, relwidth=0.3, relheight=0.4)

#dropdown
OPTIONS = list(range(8))
clicked = tk.StringVar(master=frame2)
clicked.set(OPTIONS[0])
clicked.trace("w", dropdown_selection)
drop = tk.OptionMenu(frame2, clicked, *OPTIONS)
drop.place(relx = 0.65, rely=0.25, relwidth=0.08, relheight=0.6)


button_single = tk.Button(frame2, text="Measure this Sensor", bg='#80c1ff',
fg='red', state='disabled', command=measure_single) #bg='gray'
button_single.place(relx = 0.85, rely=0.5, anchor='center')
# --- frame 2 ---
```

```
# --- frame 3 ---
frame3 = tk.Frame(root, bd=5)
frame3.place(relx=0, rely=0.2, relheight=0.07, relwidth=1, anchor='nw')

button_stop = tk.Button(frame3, text="STOP measurement(s)", bg='#80c1ff',
fg='red', state='disabled', command=stop_measurement)
button_stop.place(relx=0.5, rely=0.5, anchor='center')
# --- frame 3 ---



# --- frame 4 ---
frame4 = tk.Frame(root, bd=5)
frame4.place(relx=0, rely=0.3, relheight=0.09, relwidth=1, anchor='nw')

label2 = tk.Label(frame4, text="Create a text file with the data: ")
label2.place(relx = 0.1, rely=0.3, relwidth=0.3, relheight=0.5)

button_export = tk.Button(frame4, text="Export data", bg='#80c1ff',
fg='red', state='disabled', command=export) #bg='gray'
button_export.place(relx = 0.5, rely=0.5, anchor='center')
# --- frame 4 ---



#frame 5 will be the area with the text field
# --- frame 5 ---
frame5 = tk.Frame(root, bg='#80c1ff')
frame5.place(relx=0.0, rely=0.4, relheight=1, relwidth=1, anchor='nw')

text_frame=tk.Frame(frame5)
text_frame.place(relx=0, rely=0, relheight=0.6, relwidth=1, anchor='nw')
text=tk.Text(text_frame)
text.place(relx=0, rely=0, relheight=1, relwidth=1, anchor='nw')
vsb=tk.Scrollbar(text_frame)
vsb.pack(side='right',fill='y')
text.config(yscrollcommand=vsb.set)
vsb.config(command=text.yview)
# --- frame 5 ---
```

```
stop_=True
root.mainloop()


#Any code after mainloop() will be halted as long as the window stays alive,
so placing the code after mainloop() would execute it after the window is
closed.
ser.close() # if the application is closed, we close the serial connection
as well
# --- main ---
```