ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΩΡΙΚΗΣ Τ.Ε.
ΤΟΜΕΑΣ ΑΡΧΙΤΕΚΤΟΝΙΚΗΣ ΥΠΟΛΟΓΙΣΤΩΝ
ΚΑΙ ΒΙΟΜΗΧΑΝΙΚΩΝ ΕΦΑΡΜΟΓΩΝ

• Σχεδίαση και υλοποίηση απλού μετεωρολογικού σταθμού, με μικροελεγκτή.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Χασούρας Αντώνιος (2413)

Επιβλέπων : Ιωάννης Καλόμοιρος, Αναπληρωτής Καθηγητής

ΣΕΡΡΕΣ, ΜΑΡΤΙΟΣ 2016

ΠΕΡΙΛΗΨΗ

Η εργασία αυτή εκπονήθηκε στο τμήμα Μηχανικών Πληροφορικής του Τεχνολογικού Ιδρύματος Κεντρικής Μακεδονίας, στον τομέα Αρχιτεκτονικής Υπολογιστών και Βιομηχανικών Εφαρμογών.

Σκοπός είναι η δημιουργία ενός σταθμού παρακολούθησης καιρικών φαινομένων και μέτρησης βασικών μετεωρολογικών δεδομένων, αποθήκευσης αυτών και αποστολής τους μέσω δικτύου κινητής τηλεφωνίας.

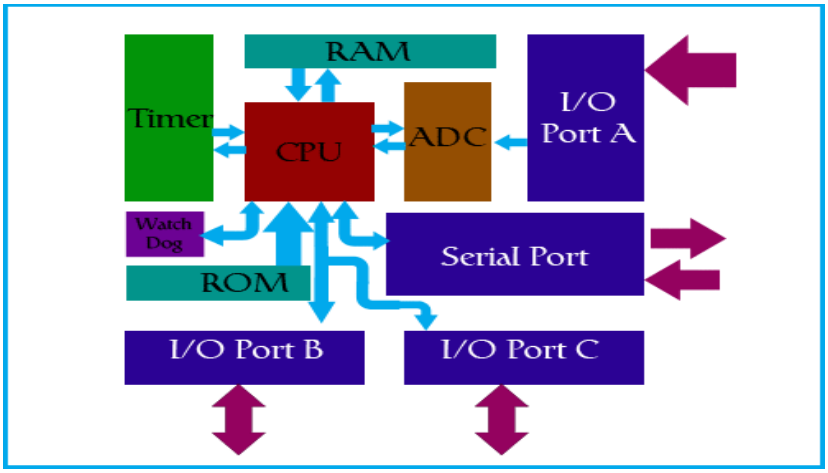
Συγκεκριμένα, κεντρικός συντονιστής του συστήματος είναι ο μικροελεγκτής PIC18F4550, ο οποίος συντονίζει και επικοινωνεί με τους αισθητήρες μέσω του πρωτοκόλλου IIC(I2C). Η οθόνη LCD ελέγχεται σειριακά χρησιμοποιώντας έξι ακροδέκτες E/E(Εισόδου/Εξόδου). Το σύστημα GSM ελέγχεται με σειριακή επικοινωνία από τον μικροελεγκτή, χρησιμοποιώντας AT εντολές. Στο σύστημα υπάρχει και δεύτερος μικροελεγκτής PIC18F4550, ο οποίος λειτουργεί σαν υπηρέτης στον PIC18F4550. Ο σκοπός του είναι ο έλεγχος της SD-CARD του συστήματος, που λειτουργεί σαν αποθηκευτικός χώρος. Η επικοινωνία υπηρέτη-κυρίου(SLAVE-MASTER) γίνεται σειριακά χρησιμοποιώντας έξι ακροδέκτες E/E(Εισόδου/Εξόδου) και ειδικά σχεδιασμένο κώδικα.

ΚΕΦΑΛΑΙΟ 1
ΕΙΣΑΓΩΓΗ ΣΤΟΥ ΜΙΚΡΟΕΛΕΓΚΤΕΣ

Εισαγωγή

Ως μικροελεγκτής ορίζεται ένα ολοκληρωμένο κύκλωμα, εφάμιλλο με μικροεπεξεργαστή, το οποίο είναι μια προγραμματιζόμενη υπολογιστική μονάδα, σχετικά αυτόνομη με ενσωματωμένα διάφορα περιφερειακά συστήματα και θύρες Ε/Ε (Εισόδου/Εξόδου) για επικοινωνία με εξωτερικά ολοκληρωμένα και μη.

Υπάρχουν μεγάλες διαφορές ανάμεσα σε ένα μικροελεγκτή και σε ένα μικροεπεξεργαστή. Αρχικά, η υπολογιστική ισχύ του μικροελεγκτή είναι πολύ μικρότερη απο αυτή ενός μικροεπεξεργαστή, όμως ο μικροελεγκτής ξεχωρίζει για την αυτονομία, την ευκολία στη χρήση, το χαμηλό κόστος, και του συνολικού μεγέθους του συνολικού υπολογιστικού συστήματος.



Εικόνα 1.1: Μια γενική αρχιτεκτονική ενός μικροελεγκτή.
(www.qrp.gr)

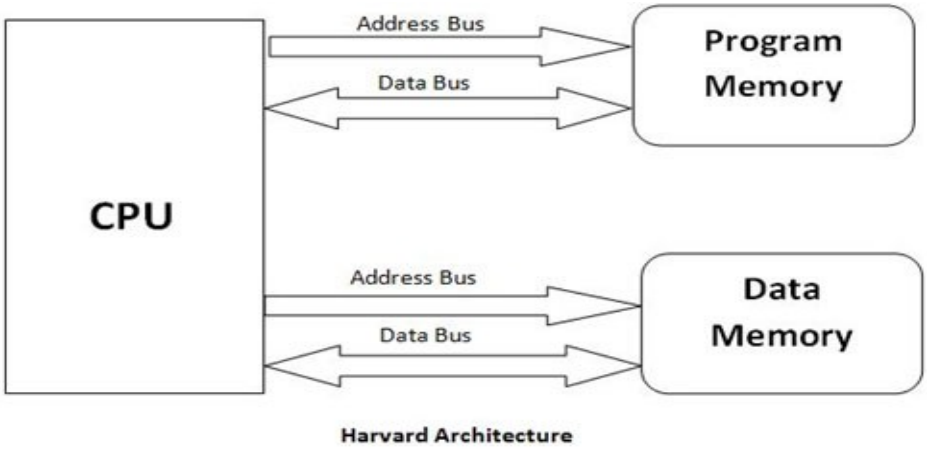
Ένας μικροελεγκτής διαθέτει επεξεργαστή, μνήμη και διάφορα περιφερειακά καθιστώντας τον αυτόνομο, έτσι η πλειοψηφία των μικροελεγκτών δεν χρειάζονται άλλο ολοκληρωμένο κύκλωμα για να λειτουργήσουν.

Για την ευκολία στη χρήση ευθύνεται και πάλι η ενσωμάτωση περιφερειακών, έτσι ο χρήστης μπορεί να προγραμματίσει την εσωτερική μνήμη και να χρησιμοποιήσει τα περιφερειακά χωρίς ιδιαίτερη δυσκολία.

Οι μικροελεγκτές έχουν ιδιαίτερα διευρυμένο πεδίο χρήσης, λόγω της ευκολίας, της απλότητας, της ακρίβειας και του χαμηλού κόστους που χαρακτηρίζουν τη χρήση τους.

Μερικά απο τα πεδία χρήσης τους είναι τα συστήματα αυτοματισμών, ηλεκτρικές συσκευές, τα κυκλώματα τηλεπικοινωνιών και πολλές άλλες κατηγορίες. Γενικά οι μικροελεγκτές χρησιμοποιούνται όταν απαιτείται έλεγχος συστημάτων.

Υπάρχουν διάφορες αρχιτεκτονικές μικροελεγκτών, αλλά ένα μεγάλο κομμάτι των κατασκευαστών, όπως Atmel και Microchip, χρησιμοποιούν την αρχιτεκτονική Harvard (εικ. 1-2).



Εικόνα 1.2: Αρχιτεκτονική Harvard.
(http://innovatelogics.com)

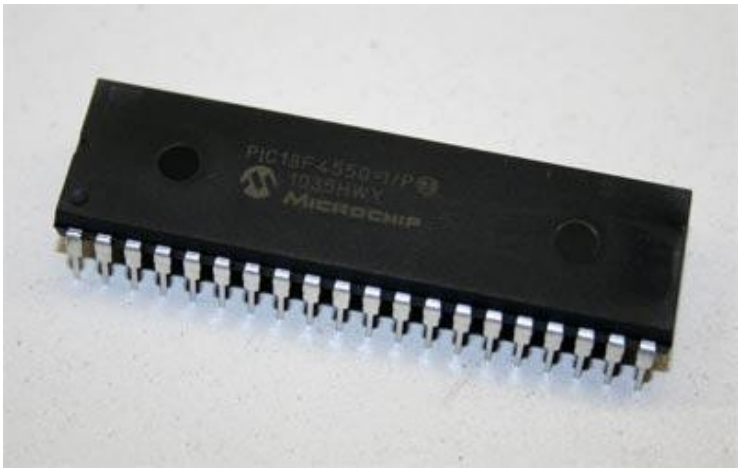
Σε αυτή την αρχιτεκτονική η μνήμη προγράμματος και δεδομένων είναι διαχωρισμένες και η κεντρική μονάδα έχει πρόσβαση σε αυτές με διαφορετικούς διαύλους. Έτσι το μέγεθος μιας εντολής μπορεί να διαφέρει από αυτά των δεδομένων, μιας και οι δίαυλοι μπορεί να έχουν διαφορετικά μεγέθη.

Ο προγραμματισμός των μικροελεγκτών μπορεί να επιτευχθεί πλέον με γλώσσες υψηλού επιπέδου (όπως C, C++) , αλλά και χαμηλού επιπέδου (όπως assembly, γλώσσα μηχανής). Στην παρούσα εργασία χρησιμοποιείται ο compiler XC8 της Microchip, ο οποίος διανέμεται δωρεάν απο την εταιρία, και μεταφράζει το πρόγραμμα σε γλώσσα μηχανής. Οι γλώσσες χαμηλού επιπέδου έχουν μερικά πλεονεκτήματα, όπως απόλυτο έλεγχο, μικρότερο χώρο προγράμματος και χαμηλό κόστος, μιας και ο assembler διατίθεται απο τις εταιρίες δωρεάν. Όμως τα τελευταία χρόνια υποχωρούν στις γλώσσες υψηλού επιπέδου, αφού οι καινούργιοι compilers βελτιστοποιούν τον κώδικα, είναι ευκολότερες στην εκμάθηση, μπορούν πολλοί προγραμματιστές να δουλέψουν συλλογικά και γενικά, είναι πιο εύκολες στη χρήση απο τις γλώσσες χαμηλού επιπέδου.

Για τον προγραμματισμό των μικροελεγκτών απαιτείται ειδικός εξοπλισμός που συνήθως κατασκευάζεται και πωλείται απο τις ίδιες τις κατασκευάστριες εταιρίες, υπάρχουν όμως και τρίτες εταιρίες που εξειδικεύονται σε αυτόν τον τομέα.

1.1 PIC18F4550

Στην εργασία χρησιμοποιείται ο μικροελεγκτής PIC18F4550 (εικ. 1-3) σε μορφή pdip της Microchip, ο οποίος χαρακτηρίζεται απο ποιότητα, αξιοπιστία, πολλούς ακροδέκτες γενικής χρήσης Ε/Ε (Εισόδου/Εξόδου) και μεγάλο εύρος περιφερειακών συσκευών.



Εικόνα 1.3: PIC18F4550 PDIP PACKAGE.

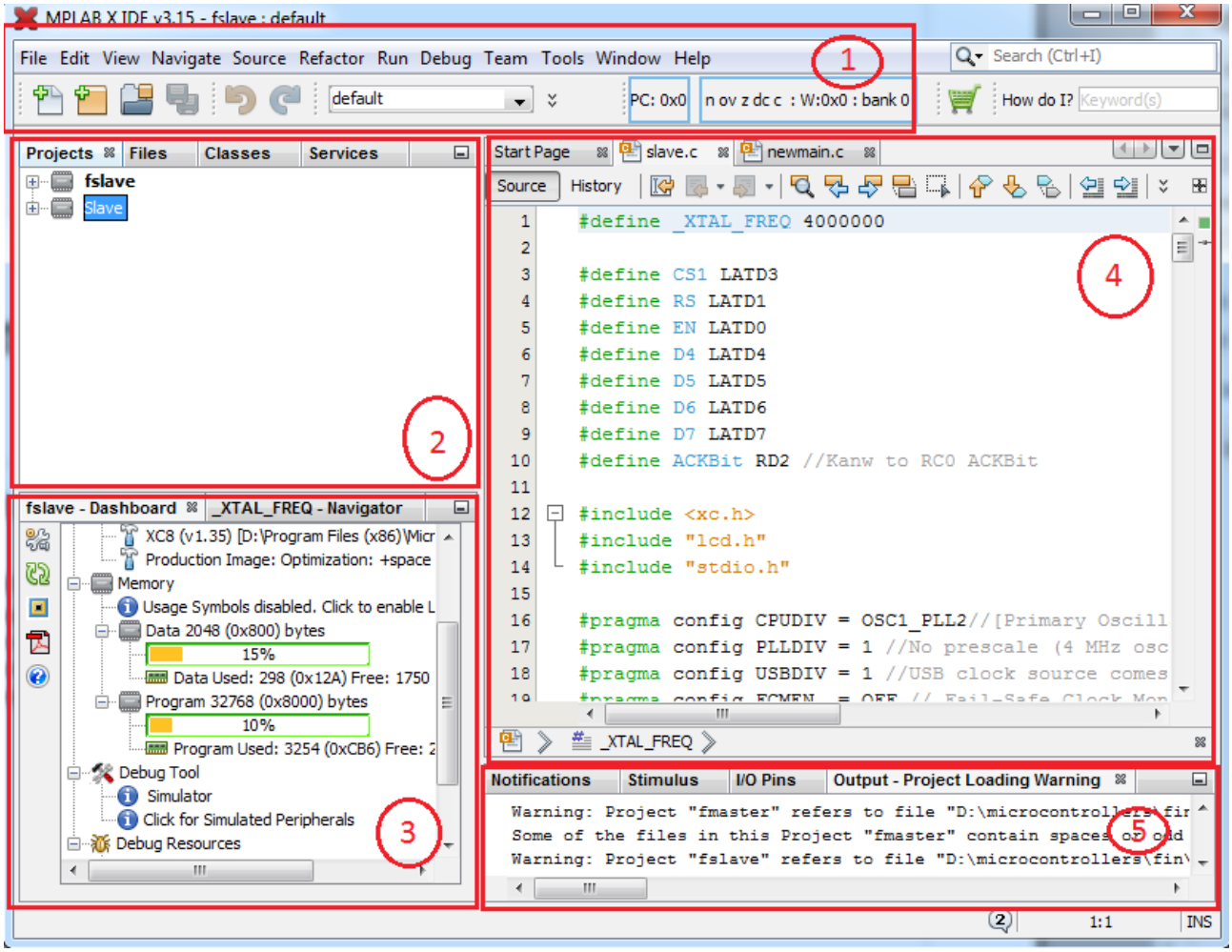
Ειδικότερα, ο PIC18F4550 διαθέτει εσωτερικό **ταλαντωτή** που μπορεί μέσω προγράμματος να ρυθμιστεί απο 31kHz (killo Herz) έως 8 Mhz (Mega Herz), 32KB (killo bytes) μνήμης **Flash** και 2KB (killo bytes) μνήμης **SRAM** (Static random-access memory). Επίσης, διαθέτει μία θύρα EUSART (Enchanted Universal Asynchronous Receiver/Transmitter) για σύγχρονη ή ασύγχρονη συνομιλία. Μία θύρα MSSP (Master Synchronous Serial Port) που υποστηρίζει τα πρωτόκολλα IIC (I^2C) και SPI. Επίσης διαθέτει τριάντα πέντε (35) ακροδέκτες γενικής χρήσης Ε/Ε (Εισόδου/Εξόδου). Τα παραπάνω χαρακτηριστικά είναι αυτά που χρησιμοποιούνται σε αυτή την εργασία.

Ο PIC18F4550 διαθέτει και άλλα χαρακτηριστικά, μερικά απο αυτά είναι 10-bit A/D (Analog/Digital) μετατροπέα, υποστήριξη USB, τέσσερις (4) εσωτερικούς μετρητές (timers).

1.2 Περιβάλλον προγραμματισμού (MPLABX -XC8-MICROCHIP)

Για την παραγωγή του κώδικα χρησιμοποιήθηκε περιβάλλον προγραμματισμού MPLABX IDE (http://www.microchip.com/mplab/mplab-x-ide) σε γλώσσα C με την χρήση του compiler XC8 (http://www.microchip.com/mplab/compilers) και τα δύο προϊόντα της MICROCHIP που διατίθενται δωρεάν. Το MPLABX IDE βασίζεται στην πολύ γνωστή και διαδεδομένη πλατφόρμα NetBeans IDE της Oracle. Επίσης είναι εξοπλισμένο με πολύ καλά εργαλεία που βοηθούν στην συγγραφή του κώδικα αλλά και στην αποσφαλμάτωση του.

Στην εικόνα 1.4 φαίνεται ένα τυπικό παράθυρο του MPLABX IDE και παρακάτω επεξηγούνται οι τομείς αυτού.



Εικόνα 1.4: Στιγμιότυπο απο MPLABX IDE.

1) Η μπάρα μενού, εδώ βρίσκονται όλες οι επιλογές που μπορεί να κάνει ο χρήστης.
2) Σε αυτό το πάνελ, ο χρήστης βλέπει τις εργασίες οι οποίες είναι ενεργές.
3) Σε αυτό το πάνελ υπάρχουν πληροφορίες για την τρέχουσα ενεργή εργασία, όπως το ποσοστό της μνήμης που χρειάζεται το πρόγραμμα, πρόσβαση στο φύλλο δεδομένων του μικροελεγκτή, κ.α.
4) Το κύριο παράθυρο συντάκτη, στο οποίο συντάσσεται ο κώδικας
5) Πάνελ στο οποίο εμφανίζονται τα αποτελέσματα του compile, αλλά και άλλα εργαλεία που βοηθούν στην αποσφαλμάτωση του κώδικα.

Να σημειωθεί οτι το MPLABX IDE είναι πολύ ευέλικτο και τα διάφορα πάνελ μπορούν να αλλάξουν θέση με απλή drag & drop μέθοδο.

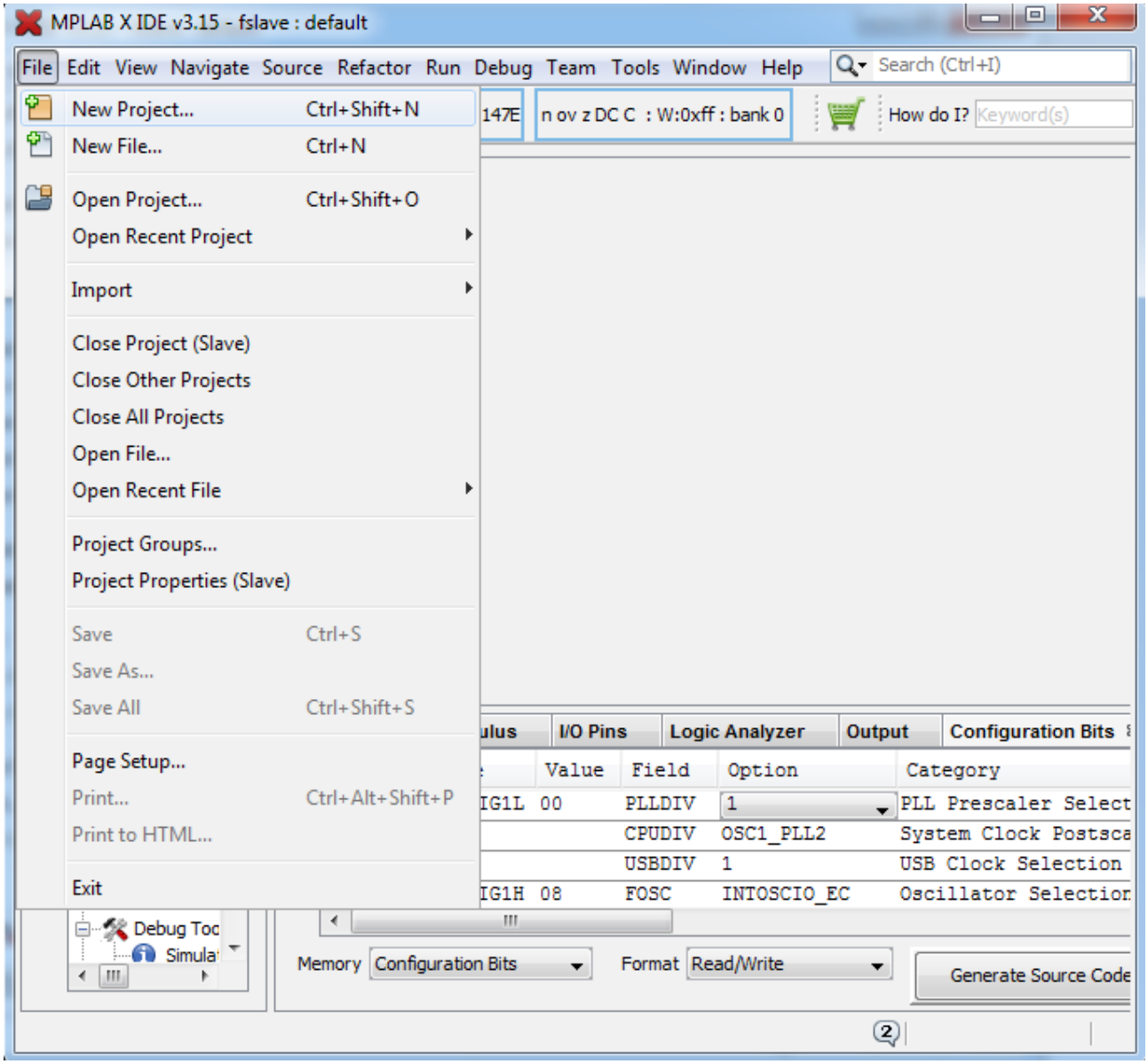
Μερικά απο τα σημαντικότερα εργαλεία που διαθέτει το MPLABX IDE είναι, το Stimulus και το IOPin που βρίσκονται στην μπάρα μενού → Windows → Simulator.

Με το Stimulus μπορούμε να εξομοιώσουμε αλλαγές λογικής στους ακροδέκτες του PIC και να παρατηρήσουμε την συμπεριφορά του.

Το IOPin είναι το εργαλείο με το οποίο παρατηρούμε την συμπεριφορά των ακροδεκτών.

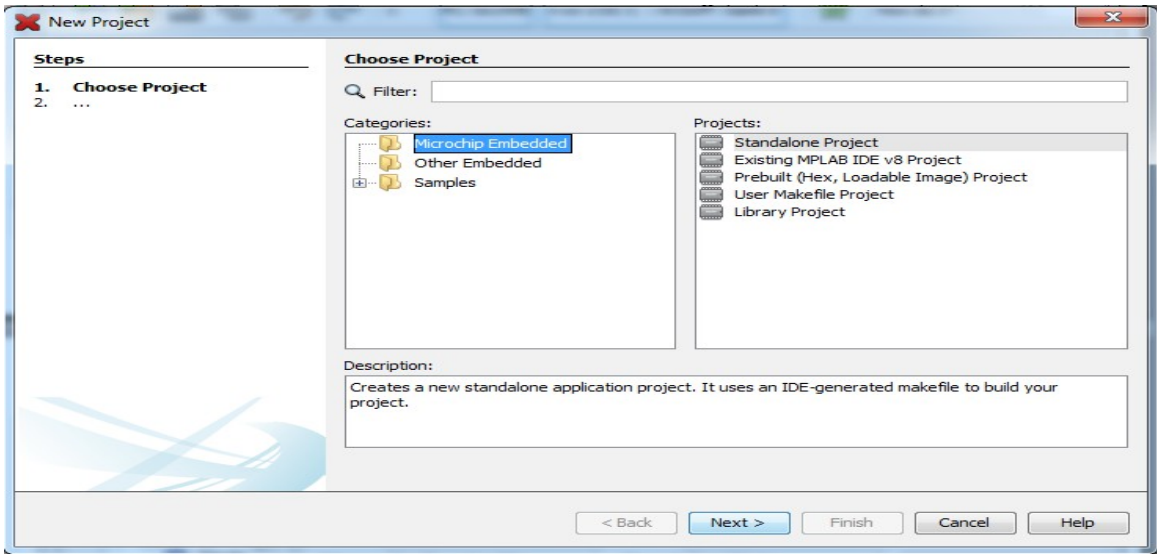
Δημιουργία εργασίας (New project)

Για τη δημιουργία ενός νέου project, πατάμε απο το μενού File→New project.(εικ 1-5)



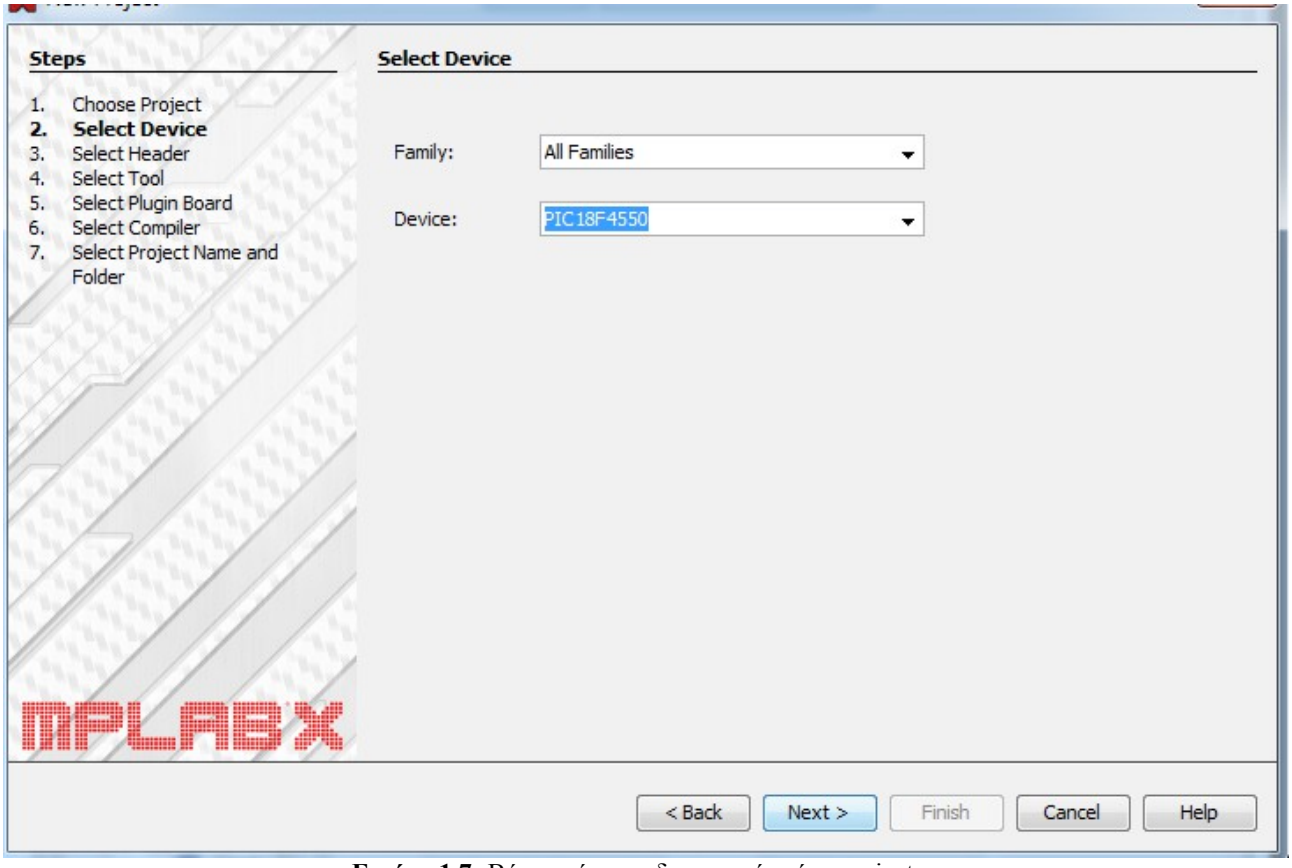
Εικόνα 1.5: Βήμα πρώτο για δημιουργία νέου project.

Στο παράθυρο που εμφανίζεται επιλέγουμε το Standalone project και πατάμε Next.(εικ 1-6)



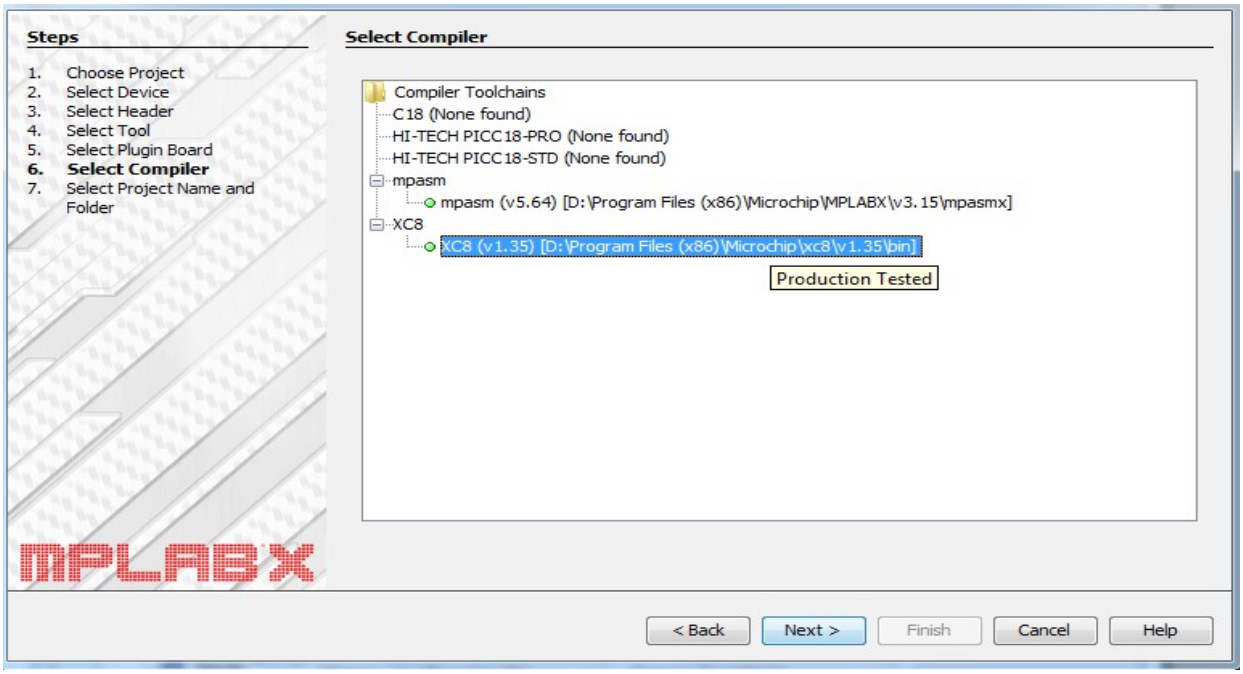
Εικόνα 1.6: Βήμα δεύτερο για δημιουργία νέου project.

Έπειτα επιλέγουμε τον μικροελεγκτή που προορίζεται το πρόγραμμα και πατάμε Next.(εικ 1-7)



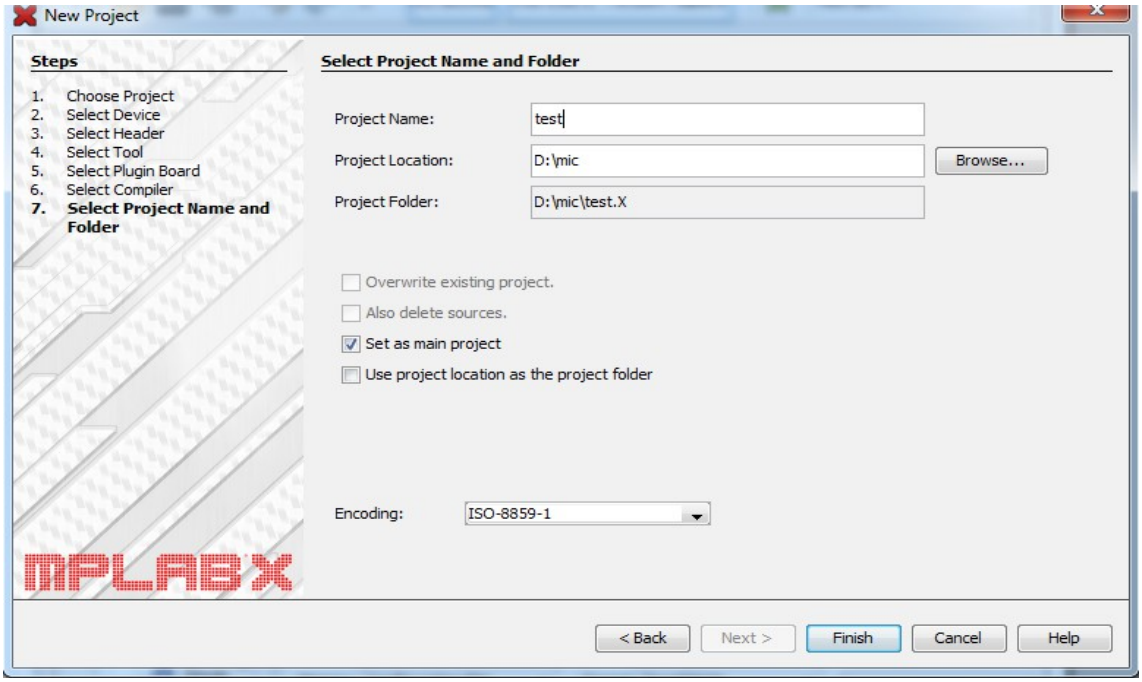
Εικόνα 1.7: Βήμα τρίτο για δημιουργία νέου project.

Στη συνέχεια επιλέγουμε τον compiler που θέλουμε και μετά πατάμε Next. (εικ. 1.8)



Εικόνα 1.8: Βήμα τέταρτο για δημιουργία νέου project.

Τέλος επιλέγουμε τον φάκελο που θα αποθηκευτεί το project και πατάμε Finish.



Εικόνα 1.9: Βήμα πέμπτο για δημιουργία νέου project.

ΚΕΦΑΛΑΙΟ 2
ΠΕΡΙΓΡΑΦΗ ΕΡΓΑΣΙΑΣ

Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφεται η λειτουργία του μετεωρολογικού σταθμού καθώς και το κύκλωμα αυτού. Επίσης υπάρχει και το διάγραμμα ροής του προγράμματος.

2.1 Περιγραφή λειτουργίας μετεωρολογικού σταθμού.

Στο μετεωρολογικό σταθμό δεν έχει εφαρμοστεί κάποιος μηχανισμός υπολογισμού χρόνου, έτσι η όλη διαδικασία δεν επαναλαμβάνεται. Απο τη στιγμή που τεθεί σε λειτουργία το κύκλωμα, στο τέλος της διαδικασίας οι μικροελεγκτές μπαίνουν σε ένα ατέρμονα βρόχο. Για επανάληψη των μετρήσεων θα πρέπει να γίνει επανεκκίνηση του συστήματος.

Στην εργασία τέσσερα (4) υποσυστήματα που πρέπει να διαχειριστεί ο μικροελεγκτής, την οθόνη LCD, την κάρτα μνήμης, τους αισθητήρες και το GSM module. Κάθε υποσύστημα απο τα παραπάνω επικοινωνεί και με διαφορετικό πρωτόκολλο επικοινωνίας. Η οθόνη LCD καθοδηγείτε απο ακροδέκτες γενικής χρήσης, η κάρτα μνήμης χρησιμοποιεί το πρωτόκολλο SPI, οι αισθητήρες το πρωτόκολλο IIC (I^2C). Τέλος το GSM module χρησιμοποιεί τη θύρα EUSART (Enhanced Universal Synchronous/Asynchronous Receiver/Transmitter) για σειριακή ασύγχρονη επικοινωνία.

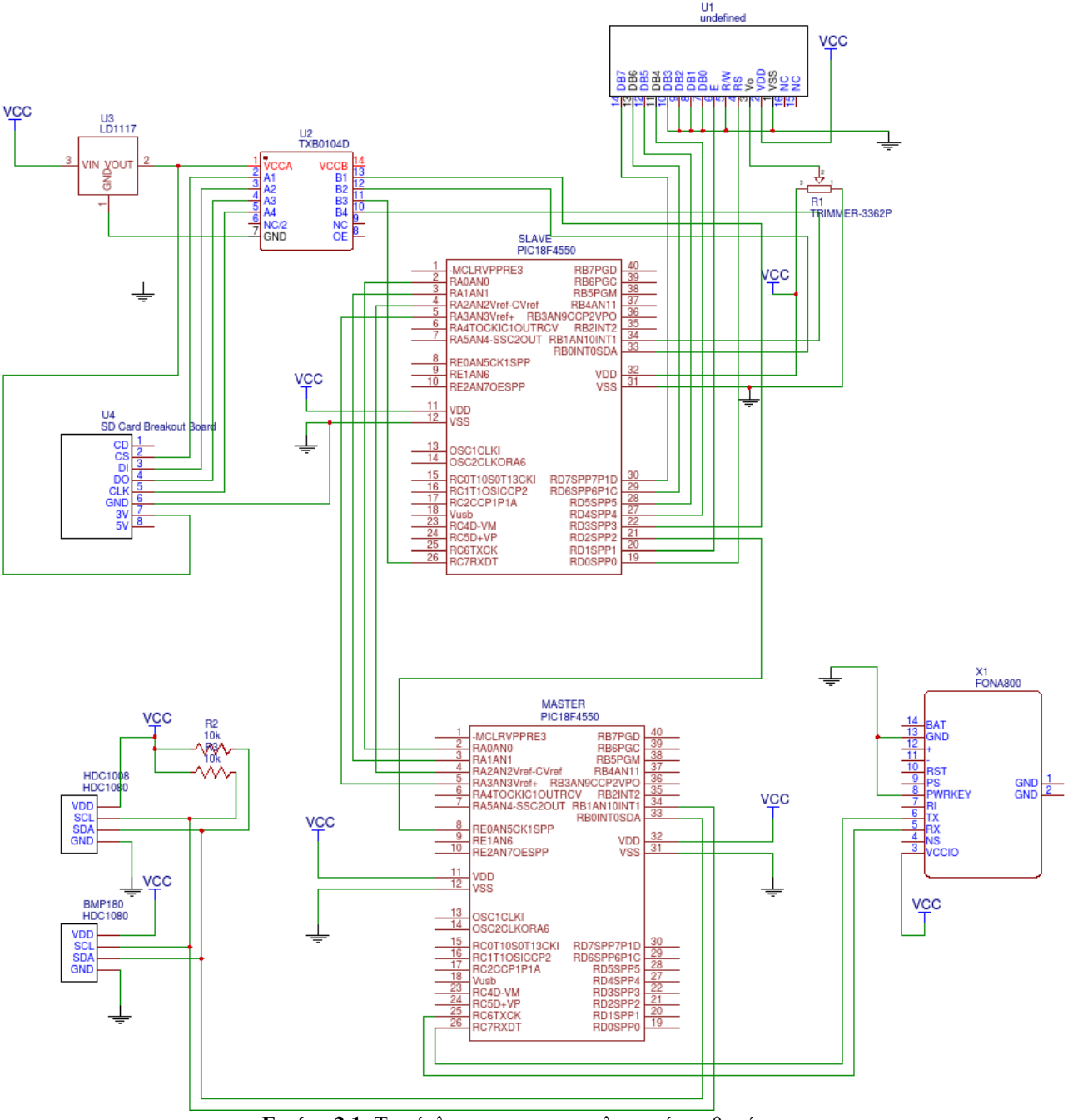
Τα πρωτόκολλα SPI και IIC (I^2C) χρησιμοποιούν τη θύρα MSSP (MASTER SYNCHRONOUS SERIAL PORT), επίσης το SPI πέρα απο τους ακροδέκτες της θύρας MSSP χρησιμοποιεί και έναν ακροδέκτη της θύρα EUSART τον RX.

Το PIC18F4550 όμως διαθέτει μία θύρα MSSP και EUSART έτσι στην εργασία προστέθηκε και ένας PIC18F4550 ακόμα, όπου λόγω των παραπάνω περιορισμών, ο ένας μικροελεγκτής επικοινωνεί με την κάρτα μνήμης σε SPI, και ο άλλος επικοινωνεί με τους αισθητήρες και το GSM module, με IIC (I^2C) και ασύγχρονη σειριακή επικοινωνία αντίστοιχα.

Ο μικροελεγκτής που επικοινωνεί με τους αισθητήρες και το GSM module, ονομάζεται master και ο άλλος slave. Η διαδικασία ξεκινά όταν τεθεί σε λειτουργία το κύκλωμα. Αφού ο master αρχικοποιήσει μεταβλητές και υποσυστήματα μπαίνει σε μια κατάσταση αναμονής είκοσι (20) δευτερολέπτων ώσπου να τεθεί σε λειτουργία το GSM module. Στη συνέχεια ο master επικοινωνεί με τους αισθητήρες, παίρνει τις μετρήσεις και τις μετατρέπει απο τύπο float σε String. Μετά ενεργοποιείται η σειριακή θύρα EUSART και δίνονται οι κατάλληλες εντολές στο GSM module έτσι ώστε να σταλεί σε μορφή SMS το String με τις μετρήσεις. Έπειτα οι μετρήσεις στέλνονται ψηφίο προς ψηφίο στο slave. Σε όλη αυτή τη διάρκεια ο slave βρίσκεται σε κατάσταση αναμονής, περιμένοντας τις μετρήσεις απο το master. Αφού λάβει τις μετρήσεις, τις γράφει στη κάρτα μνήμης μέσω πρωτοκόλλου SPI.

2.2 Κύκλωμα.

Το κύκλωμα (εικόνα 2.1) αποτελείται απο οκτώ (8) υποσυστήματα, την οθόνη LCD, τον μετατροπέα τάσης, το GSM module, την κάρτα SD, τον αισθητήρα HDC1008, τον αισθητήρα BMP180, και τους δύο μικροελεγκτές. Η πηγή τάσης είναι κοινή σε όλα τα υποσυστήματα εκτός απο την κάρτα SD που προέρχεται απο ρυθμιστή τάσης. Ο ακροδέκτης V_0 της οθόνης LCD συνδέεται σε ένα ποτενσιόμετρο, απο το οποίο ρυθμίζει το ποσοστό μαύρου της οθόνης. Ο δίαυλος IIC (I^2C) συνδέει με τα δύο του καλώδια, τους ακροδέκτες SCL και SDA των αισθητήρων με τους ακροδέκτες RB0 και RB1. Τα δύο αυτά καλώδια συνδέονται μέσω **αντίσταση τερματισμού** (pull-up resistor) στη γραμμή τροφοδοσίας. Η αντίσταση αυτή έχει τιμή 10kΩ (killo Ohm). Αντίσταση τερματισμού χρησιμοποιείται και στους ακροδέκτες του πρωτοκόλλου SPI, οι γραμμές SDO και SDI, συνδέονται μέσω αντιστάσεων με τιμή 50kΩ (killo Ohm) στη γραμμή τροφοδοσίας. Οι αντιστάσεις τερματισμού του SPI έχουν υψηλή τιμή γιατί συνδέονται σε μετατροπέα τάσης, ο οποίος ορίζει πως οι αντιστάσεις τερματισμού που συνδέονται σε αυτόν πρέπει να έχουν αυτή τη

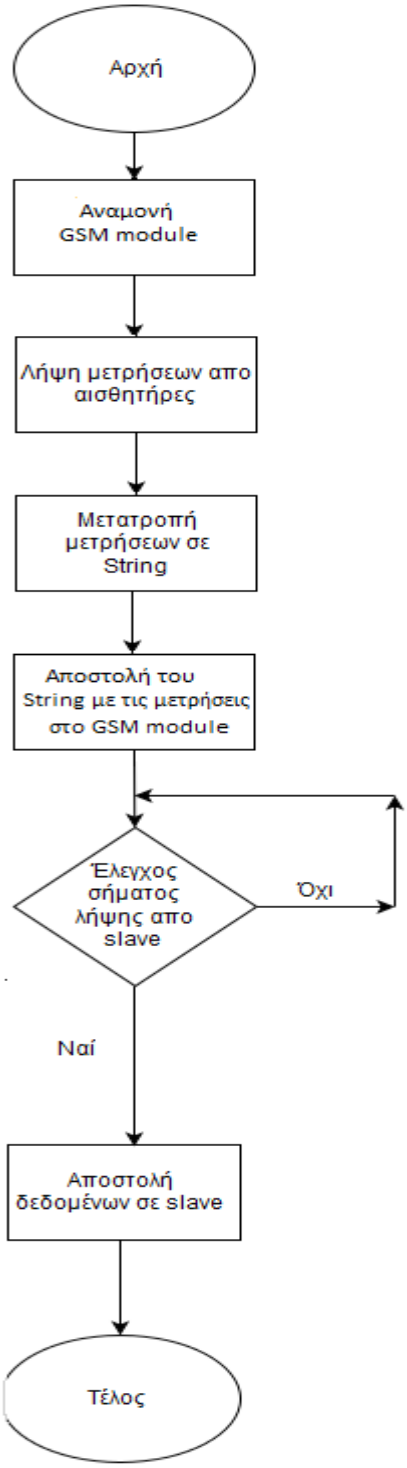


Εικόνα 2.1: Το κύκλωμα του μετεωρολογικού σταθμού.

τιμή (αναφέρεται στα φύλλα δεδομένων του). Επίσης το κανάλι της γείωσης και το κανάλι της τάσης συνδέονται με ένα κεραμικό πυκνωτή απόζευξης με τιμή 100nF (nano Farad). Ο ακροδέκτης PWRKEY του GSM module είναι συνδεδεμένος με τη γείωση έτσι ώστε το σύστημα να βρίσκεται ενεργό πάντα. Οι ακροδέκτες D_0, D_1, D_2 και D_3 της LCD, είναι συνδεδεμένοι με τη γείωση, μιας και η οθόνη LCD βρίσκεται σε λειτουργία τεσσάρων bit (4-bit). Η σύνδεση μεταξύ των δύο μικροελεγκτών δεν απαιτεί καμία ιδιαίτερη μεταχείριση παρά μόνο κατάλληλο προγραμματισμό των ακροδεκτών μέσω λογισμικού.

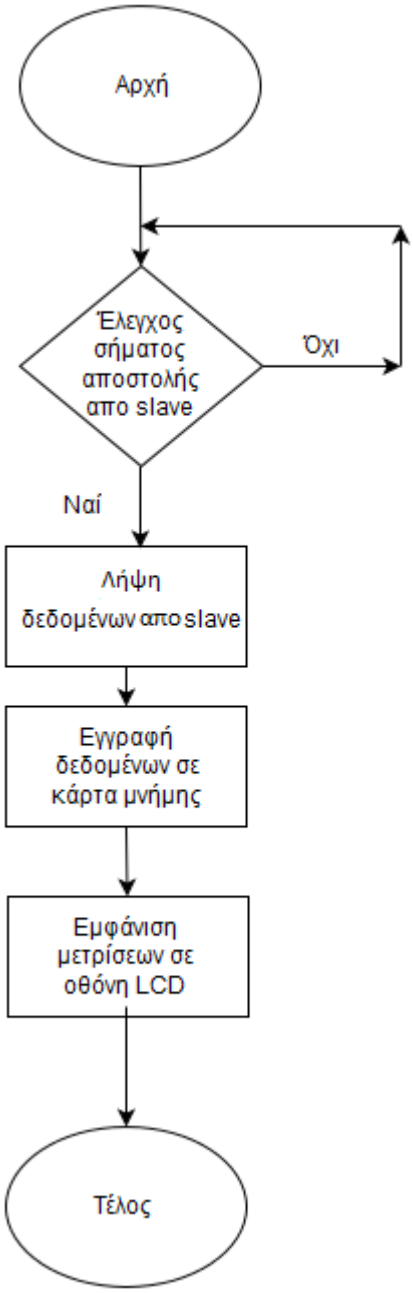
2.3 Διαγράμματα ροής

Το παρακάτω διάγραμμα ροής είναι του master.



Εικόνα 2.2: Διάγραμμα ροής master.

Τα διαγράμματα ροής είναι πολύ απλοποιημένα, μιας και οι διεργασίες που εκτελούν οι μικροελεγκτές είναι πολλές, όμως παρουσιάζονται οι βασικές λειτουργίες τους με σαφήνεια. Το διάγραμμα ροής του Master (εικόνα 2.2) παρουσιάζει τη λειτουργία του η οποία είναι η εξής. Αφού κλείσει ο διακόπτης λειτουργίας του κυκλώματος, ο master αφού αρχικοποιήσει τα διάφορα περιφερειακά που διαθέτει, όπως ορίζει το λογισμικό, μπαίνει σε κατάσταση αναμονής για είκοσι (20) δευτερόλεπτα, για να γίνει η αυτόματη έναρξη του GSM συστήματος. Στη συνέχεια μέσω πρωτοκόλλου IIC (I2C) επικοινωνεί με τους αισθητήρες, τους οποίους αρχικοποιεί και στη συνέχεια λαμβάνει τις μετρήσεις απο αυτούς. Στη συνέχεια οι μετρήσεις μετατρέπονται απο τύπο float σε string, γιατί μόνο έτσι αναγνωρίζονται απο GSM module. Αφού αποσταλούν στο σύστημα GSM οι μετρήσεις μέσω κατάλληλων εντολών αποστέλλονται σε κινητό τηλέφωνο. Έπειτα ενεργοποιεί τους ακροδέκτες που είναι συνδεδεμένοι με τον slave, σε αλληλουχία εκκίνησης, έως ότου λάβει σήμα απο αυτόν, πως είναι έτοιμος να παραλάβει τα ψηφία των μετρήσεων. Στο πέρας και αυτής της διεργασίας, τελειώνει η λειτουργία του και μπαίνει σε ατέρμονα βρόχο, έτσι ώστε να μην παρουσιάσει περίεργη συμπεριφορά.



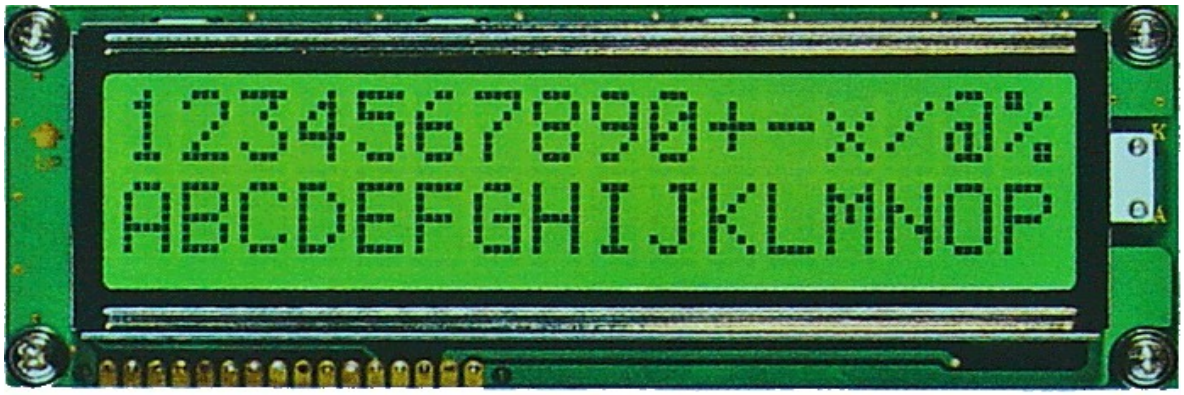
Εικόνα 2.3: Διάγραμμα ροής slave.

Στο διάγραμμα ροής του slave (εικόνα 2.3) φαίνεται η λειτουργία του, η οποία είναι η εξής. Αφού ενεργοποιηθεί το σύστημα, ο slave αρχικοποιεί τα περιφερειακά του υποσυστήματα. Έπειτα μπαίνει σε μία δομή επανάληψης έως ότου ανιχνεύσει, μία αλληλουχία εκκίνησης απο τον master. Αυτό επαναλαμβάνεται τέσσερις (4) φορές, για τα τέσσερα ψηφία των μετρήσεων. Στη συνέχεια ενεργοποιεί τη θύρα MSSP σε λειτουργία SPI και επικοινωνεί μέσω αυτού του πρωτοκόλλου με την κάρτα μνήμης. Αφού την ενεργοποιήσει, κάνει εγγραφή των μετρήσεων σε αυτή. Μετά εμφανίζει της μετρήσεις στην οθόνη της LCD. Με το πέρας και της τελευταίας διεργασίας, μπαίνει και αυτός όπως ο master σε ατέρμονα λούπα, για την προστασία του συστήματος.

ΚΕΦΑΛΑΙΟ 3
ΟΘΟΝΗ LCD, ΑΙΣΘΗΤΉΡΕΣ

Εισαγωγή

Στην εργασία χρησιμοποιείται μια οθόνη υγρών κρυστάλλων LCD (Liquid Crystal Display) (εικ. 3-1), για την εμφάνιση των μετρήσεων και για επιβεβαίωση. Μια οθόνη υγρών κρυστάλλων χρησιμοποιεί τις ιδιότητες των υγρών κρυστάλλων, δηλαδή τη δυνατότητα να αλλάζει την πολικότητα του φωτός που τα διαπερνάει αναλόγως της τάσης ρεύματος που τους διατρέχει. Εκμεταλλευόμενες αυτή την ιδιότητα χρησιμοποιώντας δύο (2) φίλτρα οριζόντιας πόλωσης του φωτός, οι οθόνες LCD είναι ικανές να εμφανίσουν γράμματα, νούμερα και διάφορα άλλα σύμβολα.



Εικόνα 3.1: Μια κλασική οθόνη LCD Display.

Οι ιδιότητες των υγρών κρυστάλλων είναι γνωστές απο καιρό, μιας και ο Friedrich Reinitzer (1858–1927) Αυστριακός βοτανολόγος και χημικός, τις ανακάλυψε το 1888. Οι υγροί κρύσταλλοι και οι ιδιότητες του εκμεταλλεύθηκαν πολύ αργότερα, το 1964, και κατ'επέκταση η πρώτη πειραματική οθόνη υγρών κρυστάλλων (LCD) εφευρέθηκε τότε στα εργαστήρια της RCA.

Απο τότε εφευρέθηκαν νέες τεχνολογίες πάνω στις οθόνες LCD, χρησιμοποιώντας τες είτε για οθόνες Lap-top, είτε σαν τηλεοράσεις, στην εργασία αυτή χρησιμοποιήσαμε την LCM-1602A, μια LCD δύο (2) σειρών και δεκαέξι (16) χαρακτήρων ανά σειρά.

Οι οθόνες LCD πωλούνται συνήθως μαζί με ένα ολοκληρωμένο, τον ελεγκτή, που διαχειρίζεται την διεπαφή της οθόνης, και μέσω αυτού διαχειριζόμαστε την οθόνη. Ο ελεγκτής της LCM-1602A είναι ο SPLC780D.

3.1 Τρόπος λειτουργίας LCM-1602A.

Ο έλεγχος της οθόνης γίνεται μέσω δεκατεσσάρων (14) ακροδεκτών που απεικονίζονται στην εικόνα 3.2.

PIN ASSIGNMENT

No.	Symbol	Level	Function	
1	Vss	--	0V	Power Supply
2	Vdd	--	+5V	
3	V0	--	for LCD	
4	RS	H/L	Register Select: H:Data Input L:Instruction Input	
5	R/W	H/L	H--Read L--Write	
6	E	H,H-L	Enable Signal	
7	DB0	H/L	Data bus used in 8 bit transfer	
8	DB1	H/L		
9	DB2	H/L		
10	DB3	H/L		
11	DB4	H/L	Data bus for both 4 and 8 bit transfer	
12	DB5	H/L		
13	DB6	H/L		
14	DB7	H/L		
15	BLA	--	BLACKLIGHT +5V	
16	BLK	--	BLACKLIGHT 0V-	

Εικόνα 3.2: Πίνακας ακροδεκτών της LCM-1602A.

Οι ακροδέκτες ελέγχου είναι οι RS, R/W και E (4,5,6). Οι ακροδέκτες 7 έως 14 είναι οι ακροδέκτες με τους οποίου μεταφέρονται τα δεδομένα. Στην παρούσα εργασία εφαρμόζεται η 4-bit μετάδοση δεδομένων, οπότε οι ακροδέκτες 7 έως 10 είναι συνδεδεμένοι με την γείωση. Οι υπόλοιποι τέσσερις (4) ακροδέκτες δεδομένων συνδέονται με τους ακροδέκτες του PIC που διαχειρίζεται την οθόνη LCD. Επίσης υπάρχει η δυνατότητα μεταφοράς δεδομένων απο την οθόνη προς τον μικροελεγκτή, όμως αυτό δεν εφαρμόζεται στην εργασία, έτσι ο ακροδέκτης R/W που είναι υπεύθυνος για τη φορά των δεδομένων είναι συνδεδεμένος στην γείωση, κάτι που κλειδώνει τη φορά των δεδομένων απο τον μικροελεγκτή προς την οθόνη.

Μία οθόνη εκλαμβάνει τα δεδομένα που βρίσκονται στους ακροδέκτες 7 έως 14 εάν το RS βρίσκεται στο λογικό ένα (1) σαν δεδομένα που προορίζονται να εμφανιστούν στην οθόνη, ενώ εάν το RS είναι στο λογικό μηδέν (0) τα δεδομένα θεωρούνται σαν εντολές.

Οποιαδήποτε δεδομένα επιθυμούμε να επεξεργαστούν απο την LCD είτε είναι χαρακτήρες είτε είναι εντολές, πρέπει ο μικροελεγκτής να ανεβάσει το E στο λογικό ένα (1) για να ενεργοποίηση την οθόνη. Στη συνέχεια θα πρέπει να το ξαναρίξει στο λογικό μηδέν (0) αλλιώς οποιεσδήποτε τυχαίες μεταβολές στους ακροδέκτες δεδομένων, θα επεξεργαστούν και ίσως προκαλέσουν απρόβλεπτη συμπεριφορά.

3.2 Βασικές εντολές.

Παρακάτω υπάρχει ένας πίνακας με βασικές εντολές.

Περιγραφή	RS	R/W	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	Χρόνος ολοκλήρωσης
Καθαρισμός οθόνης	0	0	0	0	0	0	0	0	0	1	1.53ms
Επιστροφή κέρσορα	0	0	0	0	0	0	0	0	1	-	1.53ms
Ρύθμιση οθόνης	0	0	0	0	1	DN	L	F	-	-	39us
Ολίσθηση κέρσορα ή οθόνης	0	0	0	0	0	1	S/C	R/L	-	-	39us

Εικόνα 3.3: Πίνακας βασικών εντολών της LCM-1602A.

Στην εντολή "Ρύθμιση οθόνης" υπάρχουν τρεις (3) μεταβλητές, οι DN, L και F.

- DN = 1 → 8-bit mode : Οκτώ (8) ακροδέκτες δεδομένων
- DN = 0 → 4-bit mode : Τέσσερις (4) ακροδέκτες δεδομένων

- L = 0 → Εμφάνισης μίας (1) γραμμής στην οθόνη
- L = 1 → Εμφάνισης δυο (2) γραμμών στην οθόνη

- F = 0 → Εμφάνιση κάθε χαρακτήρα με 5x7 βούλες.
- F = 1 → Εμφάνιση κάθε χαρακτήρα με 5x10 βούλες.

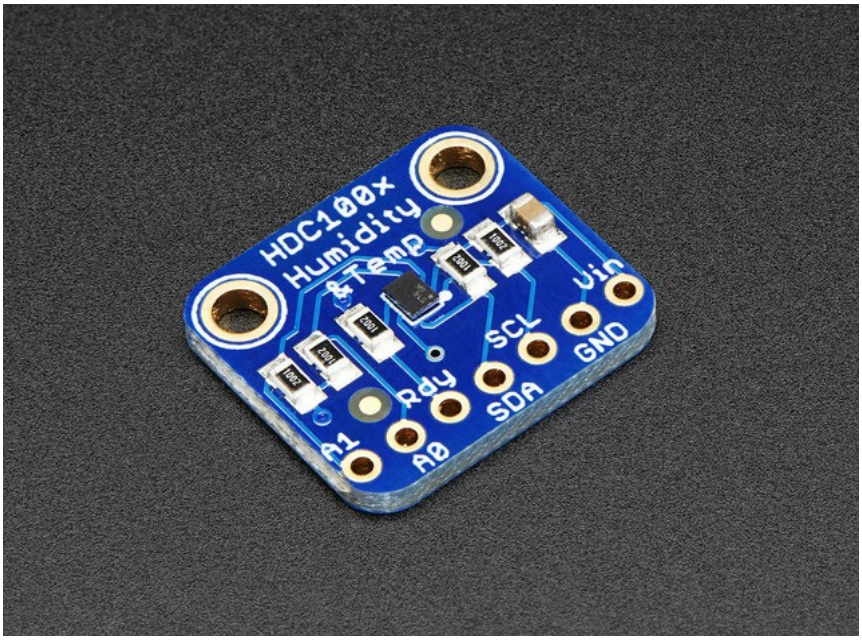
Στην εντολή "Ολίσθηση κέρσορα ή οθόνης" υπάρχουν δύο (2) μεταβλητές S/C και R/L.

- S/D = 0 → Ολίσθηση κέρσορα
- S/D = 1 → Ολίσθηση οθόνης

- R/L = 0 → Ολίσθηση αριστερά
- R/L = 1 → Ολίσθηση δεξιά.

3.3 Αισθητήρας HDC1008

Ο αισθητήρας HDC1008 (εικόνα 3.4) είναι ταυτόχρονα αισθητήρας υγρασίας και θερμοκρασίας. Έχει μεγάλη εμβέλεια λειτουργίας, για θερμοκρασία απο -40°C έως +125°C και υγρασία απο 0% έως 100% με απόκλιση ±0.2°C και ±4% αντίστοιχα. Επίσης έχει μεγάλο εύρος τάσης λειτουργίας απο 3V (Volt) έως 5V.



Εικόνα 3.4: Αισθητήρας HDC1008.

Διαθέτει ενσωματωμένο πρωτόκολλο επικοινωνίας IIC με μεταβαλλόμενη διεύθυνση απο 0b1000 000 έως 0b100 0011 αναλόγως που θα συνδέσουμε τους ακροδέκτες A1 και A0. Στην παρούσα εργασία είναι συνδεδεμένοι στη γείωση άρα ο αισθητήρας ακούει στη διεύθυνση 0b1000 000.

Ο HDC1008 έχει έξι (6) καταχωρητές, οι τρεις (3) τελευταίοι αφορούν τον σειριακό αριθμό της συσκευής. Στο πρώτο αποθηκεύεται η τελευταία μέτρηση της θερμοκρασίας, στο δεύτερο αποθηκεύεται η τελευταία μέτρηση της υγρασίας και ο τρίτος είναι υπεύθυνος για το τρόπο λειτουργίας του αισθητήρα.

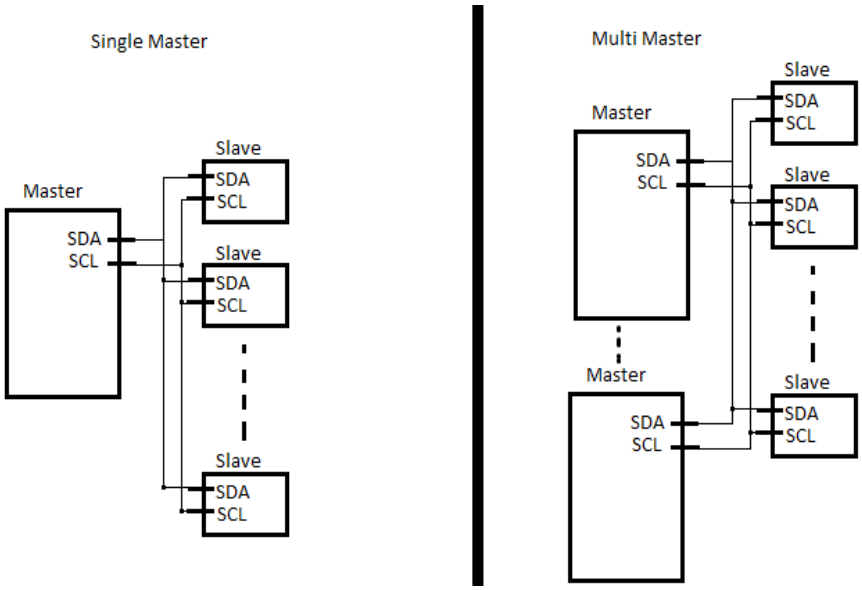
ΚΕΦΑΛΑΙΟ 4
ΠΡΩΤΟΚΟΛΛΑ ΕΠΙΚΟΙΝΩΝΙΑΣ

Εισαγωγή

Σαν πρωτόκολλο επικοινωνίας ορίζεται ένα σύνολο απο κανόνες και όρους που ορίζουν τη μεταφορά δεδομένων σε ένα δίκτυο. Δηλαδή ορίζει όλα τα πιθανά σενάρια που προκύπτουν σε μία μεταφορά δεδομένων, όπως για παράδειγμα σύμβολο για έναρξη μετάδοσης ή ρυθμό μετάδοσης.

Το πρωτόκολλο **I2C** δημιουργήθηκε απο τη εταιρία Philips το 1983 με αρχική ταχύτητα διαύλου 100kbps (kilo bit per second- χίλια bit ανά δευτερόλεπτο). Στη συνέχει αναβαθμίστηκε σε 400kbps το 1995 και σε 3,4Mbps(mega bit per second) το 1998. Χρησιμοποιείται κυρίως για την επικοινωνία περιφερειακών συσκευών με κάποιο κεντρικό επεξεργαστή.

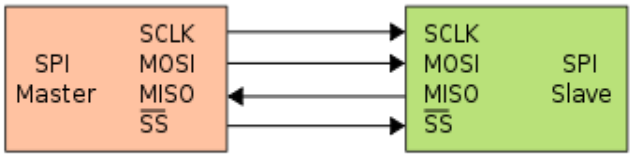
Στο **IIC** εισήχθησαν οι ρόλοι του **Master** και του **Slave**. Σε μία τυπική **IIC** επικοινωνία υπάρχει ένας Master ο οποίος διευθύνει την επικοινωνία και πια περιφερειακή συσκευή πρόκειται να επικοινωνήσει, και ένας ή παραπάνω Slaves οι οποίοι αναμένουν τον **Master** να τους καλέσει ή να γράψει στην μνήμη τους. Επίσης υπάρχει και η δυνατότητα **MultiMaster**, δηλαδή πληθυντικός αριθμός Masters, που χρησιμοποιείται σε ειδικές περιπτώσεις. Στο σχήμα 4.1 φαίνεται ένα παράδειγμα Single Master και Multimaster.



Εικόνα 4.1 : Παράδειγμα IIC, Single Master και Multimaster

Το πρωτόκολλο SPI (**Serial Peripheral Interface**) είναι μια σύγχρονη σειριακή διεπαφή επικοινωνίας για μικρές αποστάσεις, που χρησιμοποιείτε κυρίως σε ενσωματωμένα συστήματα. Κατασκευάστηκε απο την Motorola. Μοιάζει με το IIC με διαφορές στην διευθυνσιοδότηση, τον δίαυλο και την ταχύτητα μετάδοσης δεδομένων, μίας και στο SPI έχει δοθεί έμφαση στην ταχύτητα.

Το SPI εφαρμόζει Full-Dublex επικοινωνία, δηλαδή ο Slave μπορεί να στείλει δεδομένα στο Master και ο Master στο Slave απο διαφορετικού διαύλους. Επίσης λέγεται και four-wire σειριακός δίαυλος, και ο λόγος είναι, οτι ο μικρότερος αριθμός καλωδίων που απαιτείται για την εφαρμογή του πρωτοκόλλου είναι τέσσερα καλώδια. Παρακάτω υπάρχει ένα υποτυπώδες κύκλωμα SPI (εικόνα 4-2).



Εικόνα 4.2: Απλό κύκλωμα SPI.

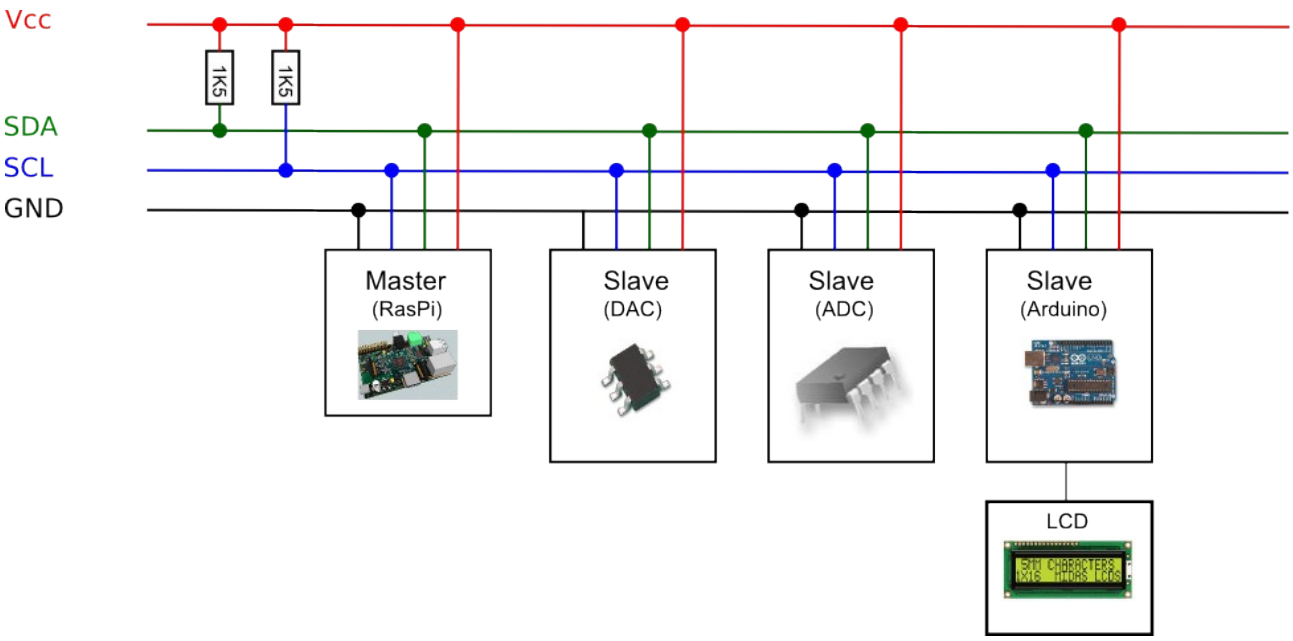
Επίσης για κάθε Slave που προστίθεται στο σύστημα, απαιτείται και ένα καλώδιο επιπλέον, για το λόγω αυτό πρέπει να γίνεται προσεκτική μελέτη πριν την εφαρμογή του πρωτοκόλλου ειδικά εάν ο Master έχει περιορισμένο αριθμό I/O (Input/Output) θυρών.

Στη παρούσα εργασία το πρωτόκολλο SPI χρησιμοποιείται για επικοινωνία του μικροελεγκτή με την κάρτα μνήμης.

4.1 Δίαυλος IIC (I^2C)

Το πρωτόκολλο αυτό χρησιμοποιεί δύο καλώδια, αμφίδρομης κατεύθυνσης: τα **SCL** (Serial Clock Line) και **SDA** (Serial Data Line).Τα οποία είναι είναι τύπου **ανοικτού συλλέκτη** (open drain) που σημαίνει ότι και οι δύο αυτές γραμμές πρέπει να συνδέονται η κάθε μία με μία αντίσταση, που ονομάζεται **αντίσταση τερματισμού** (pull-up)**,** στην γραμμή τροφοδοσίας. Το καλώδιο SCL είναι η γραμμή του ρολογιού που συγχρονίζει την επικοινωνία. Το SDA είναι η γραμμή με την οποία μεταφέρονται τα δεδομένα. Τα δύο αυτά καλώδια συνδέονται σε όλες τις συσκευές που πρόκειται να επικοινωνήσουν. Ο μέγιστος αριθμός κόμβων, που μπορούν να συνδεθούν στον δίαυλο, περιορίζεται από τον αριθμό των διαθέσιμων διευθύνσεων, αλλά και από τη χωρητική συμπεριφορά του διαύλου. Στο παρακάτω σχήμα υπάρχει ένα παράδειγμα ενός τυπικού διαύλου IIC.

Ο δίαυλος **IIC** είναι σχεδιασμένος για μικρές αποστάσεις. Ο περιορισμός βρίσκεται στη συνολική χωρητικότητα του διαύλου που δεν πρέπει να ξεπερνά τα 400pF όπως ορίζεται απο την Philllips, πέραν αυτού του ορίου δεν είναι εγγυημένη η σωστή συμπεριφορά του διαύλου. Εάν πχ. έχουμε ένα καλώδιο με χαρακτηριστικά 20pF/30cm και άλλα 50pF απο τα άλλα υποσυστήματα αυτό μας δίνει ένα περιθώριο 2,5m. μήκος διαύλου. Αυτό φυσικά είναι διαφορετικό για κάθε σύστημα. Γενικά, ένας δίαυλος κάτω του ενός μέτρου είναι μια ασφαλής επιλογή.



Εικόνα 4.3 : Παράδειγμα IIC, ένας τυπικός δίαυλος IIC.

4.2 Διευθυνσιοδότηση IIC (I^2C)

Το **IIC χρησιμοποιεί** δύο τρόπους διευθυνσιοδότησης έναν των 7-bit και έναν των 10-bit. Κάποιοι κατασκευαστές δημιούργησαν συσκευές με διεύθυνση των 8-bit, αυτού του τύπου η διευθυνσιοδότηση δεν συνιστάτε μιας και δεν αναφέρεται στην αναθεωρημένη έκδοση των προδιαγραφών του **IIC** το 2000. Η διεύθυνση του κάθε ολοκληρωμένου κυκλώματος δίνεται απο τον κατασκευαστή του, σε κάποια δίνεται η δυνατότητα να επιλεγεί απο τον χρήστη.

Στο IIC κάθε εντολή αποτελείτε απο 8 bit τα επτά πρώτα αποτελούνε μια διεύθυνση και το τελευταίο είναι το **R\W** bit**,** 0 εάν ο **master** θέλει να γράψει στον **slave**, 1 εάν θέλει να λάβει απο το τον **slave**. Κάποιες διευθύνσεις δεν μπορούν να αλλάξουν,οι 1111XXX και 0000XXX οι οποίες εξυπηρετούν ειδικούς σκοπούς. Επίσης η 0x00 είναι η καθολική διεύθυνση, ο master δηλαδή μιλάει σε όλους τους slaves, όπως φαίνεται και στο παρακάτω πίνακα (εικόνα 4.4)

Address	R/W	Purpose
0000000	0	General Call
0000000	1	Start Byte
0000001	X	CBUS
0000010	X	Reserved
0000011	X	Reserved
00001XX	X	High Speed Master
11110XX	X	10 bit addressing
11111XX	X	Reserved

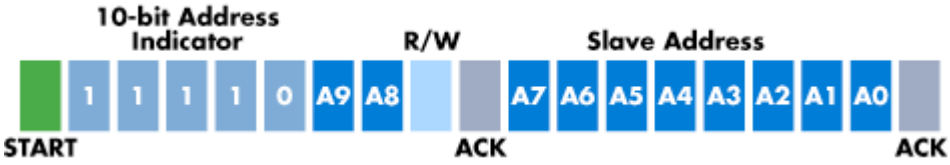
Εικόνα 4.4: Πίνακας ειδικών διευθύνσεων πρωτοκόλλου **IIC**.

7-bit διευθυνσιοδότηση - Αυτού του τύπου η διευθυνσιοδότηση είναι και η πιο κοινή. Συνολικά δίνεται μία επιλογή απο $2^7 - 16 = 112$, διαθέσιμες διευθύνσεις για κάθε **slave**.Παρακάτω παρατίθεται ένα τυπικό frame IIC διεύθυνσης.

Εικόνα 4.5:Τυπικό frame **IIC** διεύθυνσης.

10-bit διευθυνσιοδοτηση – Η διευθυνσιοδότηση αυτού του τύπου είναι ελαφρώς περιπλοκότερη απο την προηγούμενη, αλλά σχεδιάστηκε έτσι ώστε συστήματα και των δύο μεθόδων να μπορούν να συνυπάρξουν στον ίδιο δίαυλο.

Σε αυτό τον τρόπο, κάθε επιλογή του slave αποτελείται απο 2 bytes. Στο πρώτο byte, τα πέντε πρώτα bit είναι προκαθορισμένα ως εξής : 11110, τα επόμενα δύο bit είναι τα δύο πρώτα bit της διεύθυνσης, όπως φαίνεται στην εικόνα 3.5. Το τελευταίο όπως και στην 7-bit διευθυνσιοδότηση είναι το **R\W**(Read/Write). Το δεύτερο byte αποτελείται απο τα οκτώ υπόλοιπα bit της διεύθυνσης.



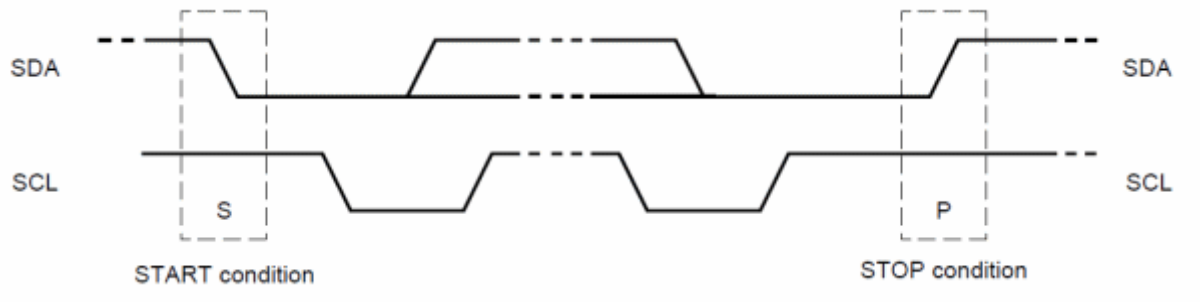
Εικόνα 4.6: Τυπικό frame 10-bit **IIC** διεύθυνσης.

4.3 Τρόπος λειτουργίας IIC(I^2C)

Η επικοινωνία ξεκινά όταν ο **master** επιθυμεί να επικοινωνήσει με κάποιον απο τους **slaves**. Η συνομιλία ξεκινά με τον master να μεταδίδει μία αλληλουχία **εκκίνησης (Start Sequence)** και τελειώνει μια μια **αλληλουχία τέλους (Stop Sequence).** Η αλληλουχία **τέλους** και η αλληλουχία **αρχής** είναι τα μόνα bit στα οποία επιτρέπεται να αλλάξουν την τιμή τους όσο ο κύκλος του ρολογιού είναι ακόμα υψηλός.

Στην αλληλουχία αρχής ο αποστολέας σηκώνει σε λογικό ένα (1) τη γραμμή **SDA** και στη συνέχεια, όσο ο κύκλος του ρολογιού **SCL** είναι ακόμα σε λογικό ένα και στη μέση περίπου, ρίχνει τη γραμμή SDA σε λογικό μηδέν.

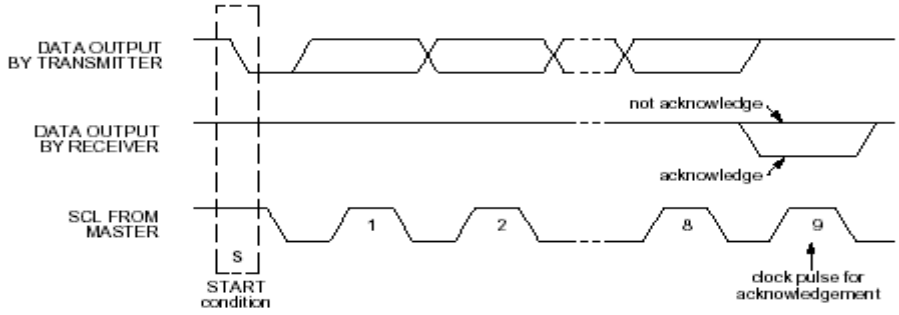
Στην αλληλουχία τέλους γίνεται ακριβώς το αντίστροφο, δηλαδή ο αποστολέας ρίχνει αρχικά τη γραμμή SDA στο λογικό μηδέν (0) και ενδιάμεσα του υψηλού κύκλου ρολογιού τη σηκώνει σε λογικό ένα (1). Στο παρακάτω σχήμα (εικ. 4.7) υπάρχει μία οπτική αναπαράσταση αλληλουχία αρχής και τέλους.



Εικόνα 4.7: Αναπαράσταση αλληλουχία αρχής και τέλους.

Όλα τα υπόλοιπα bit που μεταδίδονται πρέπει να αλλάζουν απο μηδέν σε ένα ή απο ένα σε μηδέν, μόνο όταν η γραμμή του ρολογιού βρίσκεται σε λογικό μηδέν, ειδάλλως ο παραλήπτης θα το εκλάβει εσφαλμένα σαν αλληλουχία **αρχής** ή **τέλους.**

Αφού ο **master** στείλει μια αλληλουχία **αρχής** μεταδίδει τα επτά bit της διεύθυνσης του **slave** και το **R\W** bit, και τέλος μια αλληλουχία **τέλους.** Στη συνέχεια περιμένει μια επιβεβαίωση απο τον slave, το **acknowledgement bit**, όπως φαίνεται στην εικόνα 4.7. Δηλαδή ο slave να ρίξει τη γραμμή δεδομένων **SDA** σε λογικό μηδέν. Εδώ υπενθυμίζεται ότι ο master μεταδίδοντας το stop sequence σήκωσε τη γραμμή δεδομένων σε λογικό ένα. Έπειτα η επικοινωνία θα συνεχιστεί όπως ορίζεται στο φύλλο δεδομένων του slave.



Εικόνα 4.8: Αναπαράσταση **acknowledgement bit** και **not acknowledgement bit**.

Επίσης για διευκόλυνση των συνομιλιών υπάρχει και το **Not acknowledgement bit** (εικ. 4.8) το οποίο είναι το αντίστροφο του **acknowledgement bit**.

Γενικά η συνομιλία μεταξύ master και slave ορίζεται εξολοκλήρου απο το φύλλο δεδομένων του κάθε ολοκληρωμένου, έτσι κάποιος που θέλει να χρησιμοποιήσει το πρωτόκολλο I2C, πρέπει γνωρίζει πρωτίστως τη συμπεριφορά του slave έτσι ώστε να ρυθμίσει κατάλληλα τον μικροελεγκτή που χρησιμοποιεί.

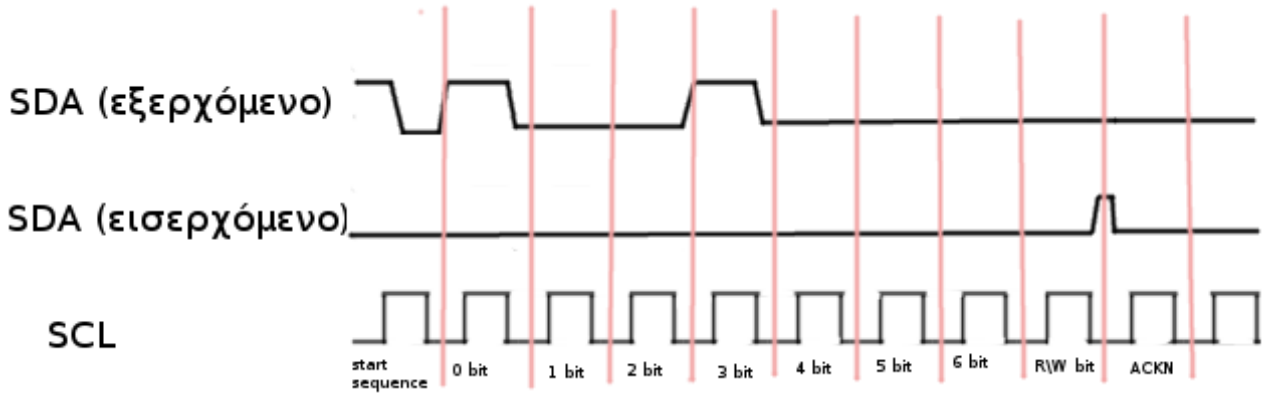
4.4 Παράδειγμα λειτουργίας πρωτοκόλλου IIC(I2C) στη παρούσα λειτουργία

Στο παράδειγμα χρησιμοποιούμε τον αισθητήρα θερμοκρασίας και υγρασίας HDC1008 ο οποίος υπάρχει και στην εργασία. Ο τρόπος που χειριζόμαστε τον αισθητήρα ορίζεται αυστηρά απο το φύλλο δεδομένων του κατασκευαστή.

Εγγραφή Δεδομένων Σε Αισθητήρα.

Η διαδικασία ξεκίνα ενημερώνοντας όλους τους αισθητήρες που είναι συνδεδεμένοι στον δίαυλο I2C, ότι ο master πρόκειτε να επικοινωνήσει με κάποιον απο αυτούς, δηλαδή ο μικροελεγκτής πραγματοποιεί μία αλληλουχία εκκίνησης.

Στη συνέχεια,όπως φαίνεται στην εικόνα 4.9, μεταδίδει την διεύθυνση του αισθητήρα προσθέτοντας το R\W bit που επιθυμεί, θέλοντας να κάνουμε εγγραφή στέλνουμε 0.

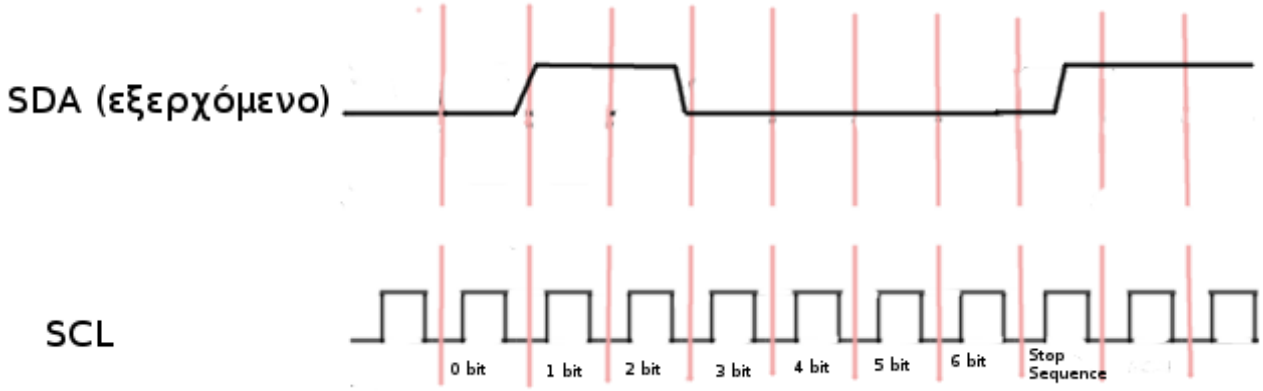


Εικόνα 4.9: Αναπαράσταση αποστολής bit απο τον master.

Στο παράδειγμα ο HDC1008 έχει διεύθυνση 0x80 (0b10000000) και εφόσον θέλουμε να κάνουμε εγγραφή το R\W bit είναι 0 άρα στέλνουμε ένα byte 0x80 και περιμένουμε την επιβεβαίωση απο τον αισθητήρα. Να σημειωθεί ότι η επιβεβαίωση και γενικά οι απαντήσεις των slaves μπορεί να αργήσουν κάποια μs ίσως και ms, στο διάστημα αυτό ο master περιμένει όσο ορίζει ο χρήστης.

Αφού λάβουμε επιβεβαίωση απο το slave του στέλνουμε την εσωτερική διεύθυνση του, που θέλουμε να κάνουμε την εγγραφή. Στέλνοντας το byte 0x02 ενημερώνουμε το slave πως επιθυμούμε να κάνουμε εγγραφή στον καταχωρητή 3 του αισθητήρα ο οποίος είναι υπεύθυνος για την λειτουργία του αισθητήρα.

Όταν ο αισθητήρας αποστείλει επιβεβαίωση είμαστε έτοιμοι να στείλουμε το byte που θα εγγραφεί στον καταχωρητή 3. Το byte που στέλνουμε στην εργασία είναι 0x60, που ρυθμίζει τον αισθητήρα. Στη συνέχεια στέλνουμε το Stop Sequence που ενημερώνει τον αισθητήρα ότι ο master έχει τελειώσει με τη συνομιλία τους, όπως φαίνεται στην εικ. 4.10

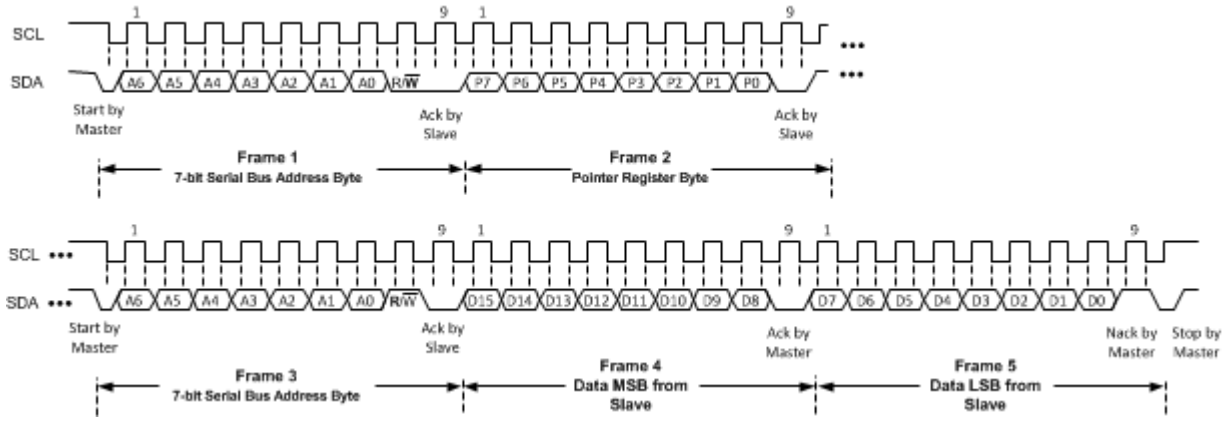


Εικόνα 4.10: Αναπαράσταση παλμού που ορίζει το Config του αισθητήρα.

Η διαδικασία επαναλαμβάνεται άλλη μία φορά, με διαφορά πως αφού στείλουμε τη διεύθυνση με W bit, στη συνέχεια στέλνουμε το byte 0x00, έτσι ώστε να ρυθμίσουμε τον δείκτη του αισθητήρα στον καταχωρητή 0 ο οποίος είναι υπεύθυνος για την αποθήκευση της θερμοκρασίας.

Αποστολή Δεδομένων Από Αισθητήρα.

Αφού έχουμε ρυθμίσει τον δείκτη του αισθητήρα στον καταχωρητή 0, στέλνουμε τη διεύθυνση του αισθητήρα μόνο που αυτή τη φορά το όγδοο bit είναι 1 δηλαδή με R bit,δηλαδή 1.
Η διαδικασία αυτή φαίνεται παρακάτω εικ.3.11. Στη συνέχεια μετά τη λήψη επιβεβαίωσης απο τον αισθητήρα, ο αισθητήρας θα στείλει δύο byte με την μέτρηση που έχει κάνει. Στο πρώτο byte ο μικροελεγκτής πλέον είναι αυτός που θα στείλει επιβεβαίωση και όταν παραλάβει και το δεύτερο byte πρέπει να στείλει μη παραλαβή (not acknowledgement).



Εικόνα 4.11: Αποστολή Δεδομένων Από Αισθητήρα(www.ti.com).

4.5 Δίαυλος SPI

Περιγραφή διεπαφής

Στο πρωτόκολλο SPI περιγράφονται τέσσερα λογικά σήματα, το SCLK (Serial Clock), MOSI (Master Out- Slave In), MISO (Master In- Slave Out), και SS (Slave Select).

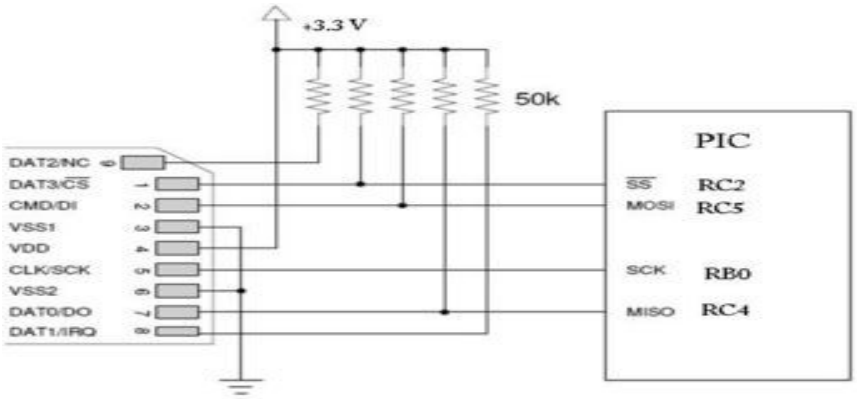
SCLK	Serial Clock
MOSI	Master Out- Slave In
MISO	Master In- Slave Out
SS	Slave Select

Εικόνα 4.12: Πίνακας ακροδεκτών **SPI**.

Το SCLK, το σήμα ρολογιού που εξέρχεται απο τον Master και καθορίζει την ταχύτητα επικοινωνίας. Το MOSI (Master Out- Slave In) είναι το σήμα που μεταφέρει δεδομένα απο τον Master στον Slave. Το MISO (Master In- Slave Out) είναι το αντίστροφο του MOSI, δηλαδή το σήμα που μεταφέρει δεδομένα απο τον Slave στον Master. Τέλος το SS (Slave Select) είναι το σήμα

που καθορίζει ποιος απο τους Slaves θα επικοινωνήσει με τον Master.

Στο πρωτόκολλο SPI οι γραμμές SCLK, MOSI, MISO είναι κοινές για όλους τους Slaves. Τα SCLK, MOSI και MISO είναι είναι τύπου **ανοικτού συλλέκτη** (open drain) που σημαίνει ότι και οι τρεις αυτές γραμμές πρέπει να συνδέονται η κάθε μία με μία αντίσταση, που ονομάζεται **αντίσταση τερματισμού** (pull-up), στην γραμμή τροφοδοσίας, όπως φαίνεται στην εικόνα 4.13.



Εικόνα 4.13: Παράδειγμα τοποθέτησης pull-up αντιστάσεων σε SPI σύνδεση.
(http://hades.mech.northwestern.edu)

Ο μέγιστος αριθμός υπηρετών (slaves) που μπορούν να συνδεθούν στον δίαυλο περιορίζεται μόνο απο τον αριθμό των διαθέσιμων θυρών επικοινωνίας απο τον Master.

4.6 Εφαρμογή πρωτοκόλλου SPI

Το SPI χρησιμοποιεί σύγχρονη σειριακή μεταφορά δεδομένων, δηλαδή ο χρονισμός μετάδοσης δεδομένων εξέρχεται αποκλειστικά απο τον master (**SCLK**) ο οποίος είναι ο μοναδικός συντονιστής μια επικοινωνίας SPI, και βάση αυτού του χρονισμού εξέρχονται ή εισέρχονται δεδομένα στα υποσυστήματα του πρωτοκόλλου . Επομένως δεν υποστηρίζεται multimaster εφαρμογή όπως στο IIC.

Η διαδικασία εφαρμογής του SPI είναι σχετικά απλή και ευκολότερη απο αυτή του IIC. Ο λόγος είναι πως αν και οι slaves μοιράζονται τους διαύλους SCLK, **MOSI** και **MISO** δεν πρόκειται να αποστέλλουν ή να λάβουν ταυτόχρονα δεδομένα με τους υπόλοιπους, μιας και στο πρωτόκολλο αυτό ο κάθε slave έχει ένα ξεχωριστό ακροδέκτη τον SS, ο οποίος ορίζει την λειτουργία τους.

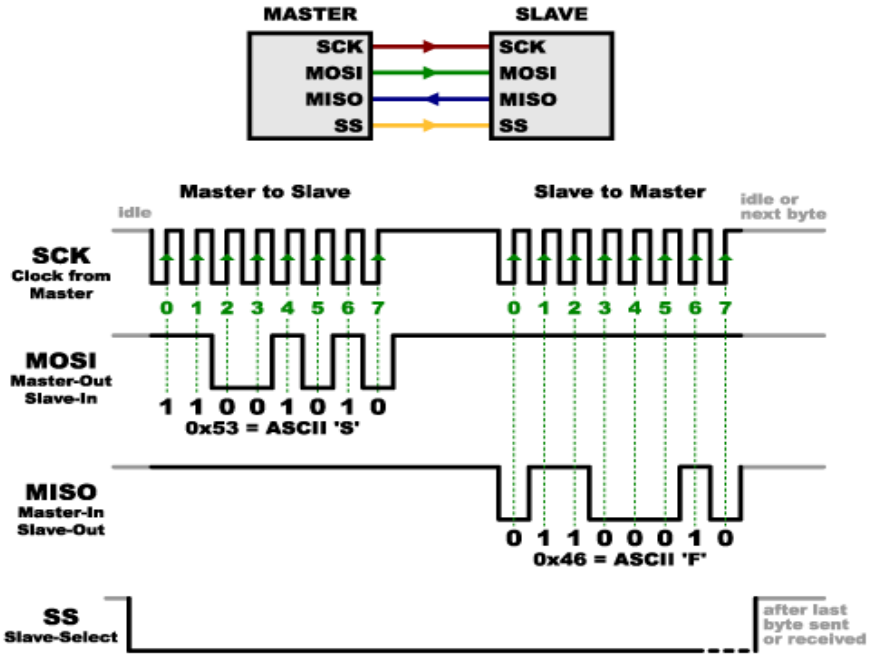
Οι απαντήσεις απο τους slaves δεν ίδιες, όπως και οι εντολές τους, και ορίζονται αυστηρά απο τα **φύλλα δεδομένων** (**datasheet**) του εκάστοτε SPI slave.

Απλουστεύοντας την εφαρμογή του πρωτοκόλλου, μπορεί περιγραφή ως εξής :

Αρχικά ρυθμίζεται κατάλληλα ο μικροελεγκτής, στη συνέχεια διαβάζοντας προσεκτικά τα φύλλα δεδομένων των slaves, αποστέλλουμε την εντολή μέσω του διαύλου MOSI. Τέλος θέτουμε τον μικροελεγκτή σε κατάσταση αναμονής έως ότου ο slave απαντήσει στον δίαυλο MISO.

Πριν την αποστολή της εντολής πρέπει πρώτα να ενεργοποιήσουμε τον slave. Ο ακροδέκτης SS (Slave-Select) του slave κατά τη διαδικασία αρχικοποίησης βρίσκεται στο λογικό ένα (1) ή όπως στην παρούσα εργασία μία τάση 5 V (volt). Για την ενεργοποίηση του slave κατεβάζουμε την τάση του ακροδέκτη SS στα 0 V ή στο λογικό μηδέν (0). Κατά τη διάρκεια που ο ακροδέκτης SS του slave βρίσκετε στο λογικό μηδέν (0), θεωρεί πως είναι ενεργός και πως οτιδήποτε αποσταλεί μέσω του διαύλου MOSI, προορίζεται για αυτόν. Έτσι ο προγραμματισμός

του master πρέπει να είναι προσεκτικός σε ότι αφορά τους ακροδέκτες SS των slaves, αλλιώς υπάρχει κίνδυνος να καταστραφεί η επικοινωνία. Στην εικόνα 4.14 φαίνεται η διαδικασία αποστολής και παραλαβής δεδομένων απο master και slave.



Εικόνα 4.14: Παράδειγμα εφαρμογής πρωτοκόλλου SPI.
(https://learn.sparkfun.com)

4.7 Μεταφορά δεδομένων χρησιμοποιώντας ακροδέκτες κοινής χρήσης.

Λόγω χρήσης όλων των θυρών επικοινωνίας EUSART, και MSSP και απο τους δύο μικροελεγκτές και αφού ο ένας χρησιμοποιεί το πρωτόκολλο IIC και ο άλλος το SPI, δεν είναι δυνατό να επικοινωνήσουν. Επίσης δεν μπορούν ούτε μέσω της σειριακής θύρας EUSART επειδή κάτι τέτοιο θα εμπόδιζες την επικοινωνία με το GSM module.

Η λύση βρέθηκε χρησιμοποιώντας ακροδέκτες γενικής χρήσης. Για τον master τους A0, A1, A2, A3 και E0 για το slave A0, A1, A2, A3 και C0. Με το τρόπο που θα περιγραφεί παρακάτω, μπορεί να σταλεί ψηφίο προς ψηφίο, ακέραιος αριθμός απο ένα (1) έως εννιά (9). Δηλαδή μπορεί να αποσταλεί απο το master οποιοσδήποτε αριθμός, αρκεί να είναι ακέραιος και αφού έχει επεξεργαστεί ώστε να παρθεί και να αποσταλεί ψηφίο προς ψηφίο.

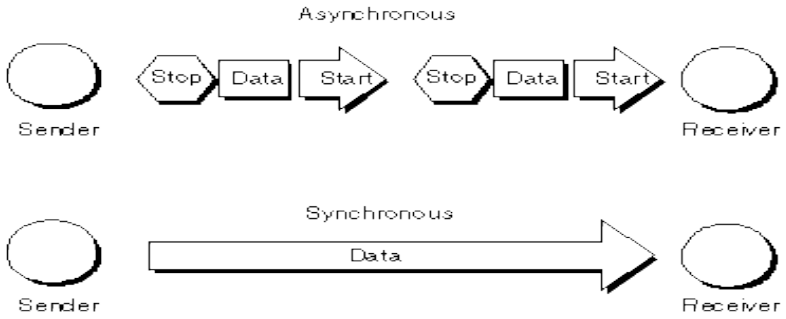
Η διαδικασία ξεκινά ως εξής, ο slave μπαίνει σε ένα διαρκή έλεγχο για αλληλουχία εκκίνησης στους ακροδέκτες A0 έως A3, η αλληλουχία εκκίνησης είναι όλα τα ψηφία να τεθούν στο λογικό ένα (1). Εάν ο slave αναγνωρίσει μια αλληλουχία εκκίνησης τότε ανεβάζει τον ακροδέκτη C0 που το ονομάζω ACKBIT σε λογικό ένα (1).Ο master αφού αναγνωρίσει το ACKBIT σαν λογικό ένα (1), τότε στέλνει το ψηφίο. Εάν για παράδειγμα είναι το 7 τότε οι ακροδέκτες A0-A3 θα είναι 0b0111 (επτά (7) σε δεκαδικό. Στη διάρκεια αυτή ο slave προσπαθεί να ανιχνεύσει αλλαγή στους ακροδέκτες A0-A3 απο την αλληλουχία εκκίνησης, η αλλαγή σημαίνει αποστολή ψηφίου και μετατρέπει τη λογική τιμή στους ακροδέκτες A0-A3 σε ακέραιο αριθμό με τον εξής αλγόριθμο : A0 + A1*2 + A2*4 + A3*8. Αφού γίνει η μετατροπή τότε ενημερώνει τον master ότι έχει λάβει το ψηφίο, ρίχνοντας τη τάση στο ACKBIT σε 0V ή ρίχνοντας το ACKBIT σε λογικό μηδέν (0). Ο master αφού αναγνωρίσει την πτώση τάσης στο ACKBIT τότε είτε επαναλαμβάνει την διαδικασία για να στείλει άλλο ψηφίο, είτε προχωρά σε επόμενη διεργασία.

Στην εργασία η διαδικασία επαναλαμβάνεται τέσσερις (4) φορές δύο ψηφία για κάθε μέτρηση, με αποτέλεσμα στην κάρτα μνήμης και στην οθόνη LCD να εμφανίζονται μόνο μετρήσεις απο 0 έως 99 για θερμοκρασία και απο 0 έως 99 για υγρασία.

4.8 Σειριακή επικοινωνία

Με τον όρο σειριακή επικοινωνία, εννοούμε την μετάδοση δεδομένων ανά ένα bit μιας σειράς απο bit, μέσω ενός διαύλου επικοινωνίας.

Υπάρχουν δύο είδη σειριακής επικοινωνίας η **σύγχρονη** και η **ασύγχρονη** μορφή σειριακής επικοινωνίας(εικόνα 3-14). Στη σύγχρονη σειριακή επικοινωνία τα bits μεταδίδονται ανάλογα με τον παλμό ενός ρολογιού. Δηλαδή η σύγχρονη σειριακή επικοινωνία απαιτεί έναν ακόμα δίαυλο για τη μετάδοση του παλμού ρολογιού. Στην ασύγχρονη σειριακή επικοινωνία ο ρυθμός μετάδοσης δεν καθορίζεται απο παλμό ρολογιού, για το λόγω αυτό καθορίζεται αλληλουχία έναρξης και λήξης. Για παράδειγμα στο Adafruit Fona, το τέλος κάθε μετάδοσης καθορίζεται απο το σύμβολο CR (“\r”, carriage return).



Εικόνα 4.15: Σύγχρονη και ασύγχρονη σειριακή επικοινωνία.

ΚΕΦΑΛΑΙΟ 5
SD-CARD

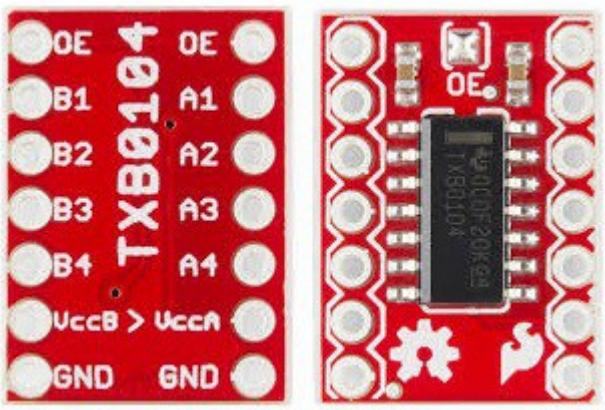
Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφεται η κάρτα μνήμης SD καθώς και τρόπος εγγραφής αλλά και εξαγωγής δεδομένων. Επίσης περιγράφονται και τα εργαλεία που είναι απαραίτητα για την επικοινωνία με τη κάρτα SD.

5.1 Μεταφραστής τάσης (Logic Level Converter)

Στην παρούσα εργασία υπάρχει η ανάγκη εισαγωγής ενός μεταφραστή τάσης, ο οποίος λύνει το πρόβλημα των διαφορετικών τάσεων λειτουργίας μεταξύ της κάρτας SD και του υπόλοιπου συστήματος. Η κάρτα SD έχει μέγιστη τάση λειτουργίας 3.6 V (Volt), δηλαδή η μέγιστη τάση που τροφοδοτείται δεν μπορεί να ξεπεράσει τα 3.6 V. Συνεπώς η μέγιστη τάση που αναπαριστά το λογικό ένα (1) στους ακροδέκτες επικοινωνίας της κάρτας πρέπει να είναι μικρότερη απο 3,6 V. Το υπόλοιπο σύστημα λειτουργεί σε τάση 5 V. Έτσι οποιαδήποτε επικοινωνία μεταξύ κάρτας SD και μικροελεγκτή μπορεί να βλάψει την κάρτα ή να έχει περίεργη συμπεριφορά. Για το λόγω αυτό, η εισαγωγή ενός μεταφραστή τάσης ήταν αναγκαία στην εργασία.

Στην εργασία χρησιμοποιείται ο TXB-0104 (εικ. 5.1) ο οποίος είναι μεταφραστής τάσης δύο κατευθύνσεων. Δηλαδή εάν υπάρξει μία τάση στην μία μεριά, είτε στην υψηλή τάση είτε στην χαμηλή, μεταφράζει αυτόματα στην άλλη μεριά σε κατάλληλη τάση.



Εικόνα 5.1: Μεταφραστής τάσης TXB-0104.

Ο τρόπος λειτουργίας του TXB-0104 είναι απλός. Εφαρμόζονται στους ακροδέκτες VccB και VccA, οι δυο τάσεις οι οποίες θέλουμε να μεταφραστούν, είτε απο την μικρότερη στην μεγαλύτερη, είτε απο τη μεγαλύτερη στη μικρότερη, με μόνο περιορισμό η τάση στο VccB να είναι μεγαλύτερη απο την τάση στο VccA. Στη συνέχεια στη μεριά B (port B), εφαρμόζονται οι ακροδέκτες επικοινωνίας του μικροελεγκτή, που είναι στην ουσία δίαυλοι του πρωτοκόλλου SPI.

Στη μεριά A (port A) εφαρμόζονται οι ακροδέκτες επικοινωνίας της κάρτας SD, που πλέον έχει μεταφραστεί σε λογική τάσης 3V6.

Να σημειωθεί επίσης ότι δεν χρησιμοποιείται διαφορετική πηγή τάσης για τα 3,6V (volt),

αλλά ένας ρυθμιστής τάσης (**voltage regulator**) ο οποίος ρίχνει την τάση στα 3,6V (Volt).

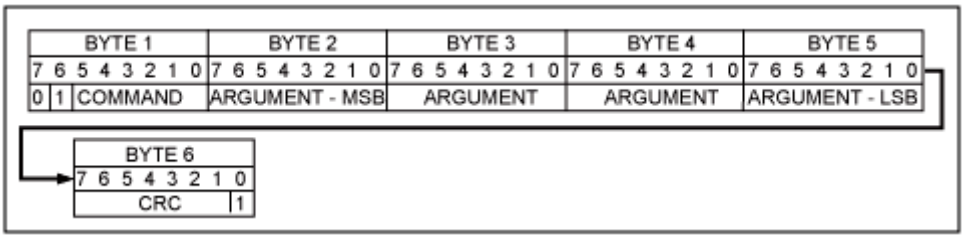
5.2 SD-CARD και τρόποι/μεθόδοι επικοινωνίας

Η SD-CARD (Secure Digital) είναι ένα μέσω αποθήκευσης δεδομένων η οποία δεν χάνει δεδομένα όταν σταματά η τροφοδοσία. Πρωτοεμφανίστηκε το 1999 απο την ενωμένη προσπάθεια των εταιριών SanDisk, Panasonic και Toshiba σε μια προσπάθεια να προσπεράσουν την τότε διαδεδομένη MultiMediaCard(MMC) . Τα στάνταρτ των SD καρτών θέτονται απο την SD Card Association (SDA). Οι κάρτες SD είναι εσωτερικά χωρισμένες σε **Sectors** οι οποίοι αποτελούνται απο 512 bytes, συνήθως, ο καθένας.

Υπάρχουν τέσσερις (4) τρόποι μεταφοράς δεδομένων από και προς μιας κάρτας SD, οι : SPI bus mode, One-bit SD bus mode, Four-bit SD bus mode και Two differential lines SD UHS-II mode. Στην παρούσα εργασία εφαρμόζεται το SPI bus mode, μιας και είναι ο ευκολότερος τρόπος χρησιμοποιώντας μικροελεγκτή.

Για τη δημιουργία αρχείων όπως txt κ.α. Πρέπει να εφαρμοστεί κάποιο σύστημα αρχείων όπως NTFS ή FAT, ούτως ώστε να καταστούν αναγνωρίσιμα τα δεδομένα απο τα διάφορα λειτουργικά συστήματα. Στη παρούσα εργασία δεν εφαρμόζεται κάποιο σύστημα αρχείων, αλλά τα δεδομένα εγγράφονται σαν **raw data** στους **sectors**, μία μέτρηση ανά sector.

Μία έγκυρη εντολή σε μια SD κάρτα, αποτελείτε απο 48 bit, όπως φαίνεται στην εικόνα 5.2. Τα δυο πρώτα bit είναι πάντα 01 και δηλώνουν στην κάρτα ότι ακολουθεί εντολή. Στα επόμενα πέντε (5) bit τοποθετείτε η εντολή. Τα επόμενα τέσσερα (4) byte περιέχουν περαιτέρω πληροφορίες που ίσως χρειάζεται η εντολή. Το επόμενο byte περιέχει bit ασφαλείας για λάθει μετάδοσης αλλά συνήθως δεν χρησιμοποιείται στο SPI.



Εικόνα 5.2: Ένα στιγμιότυπο εντολής κάρτας SD.
(www.maximintegrated.com)

Στην ουσία κάθε εντολή είναι ένα νούμερο, με κάθε μία να επιστρέφει συγκεκριμένες απαντήσεις απο την κάρτα. Δηλαδή εάν θέλουμε να στείλουμε την εντολή CMD0 η οποία κάνει reset την κάρτα. Στο πρώτο byte θα βάλουμε στα δυο πρώτα bit το μηδέν (0) και το ένα (1). τα υπόλοιπα θα είναι μηδέν (0). Εάν θέλουμε την CMD9, μετά τα δυο πρώτα bit βάζουμε το εννέα (9) σε δυαδικό, δηλαδή 0b001001.Ακολουθεί πίνακας με βασικές εντολές.

Command	Argument	Response	Data	Abbreviation	Description
CMD0	None(0)	R1	No	GO_IDLE_STATE	Software reset.
CMD1	None(0)	R1	No	SEND_OP_COND	Initiate initialization process.
ACMD41(*1)	*2	R1	No	APP_SEND_OP_COND	For only SDC. Initiate initialization process.
CMD8	*3	R7	No	SEND_IF_COND	For only SDC V2. Check voltage range.
CMD9	None(0)	R1	Yes	SEND_CSD	Read CSD register.
CMD10	None(0)	R1	Yes	SEND_CID	Read CID register.
CMD12	None(0)	R1b	No	STOP_TRANSMISSION	Stop to read data.
CMD16	Block length[31:0]	R1	No	SET_BLOCKLEN	Change R/W block size.
CMD17	Address[31:0]	R1	Yes	READ_SINGLE_BLOCK	Read a block.
CMD18	Address[31:0]	R1	Yes	READ_MULTIPLE_BLOCK	Read multiple blocks.
CMD23	Number of blocks[15:0]	R1	No	SET_BLOCK_COUNT	For only MMC. Define number of blocks to transfer with next multi-block read/write command.
ACMD23(*1)	Number of blocks[22:0]	R1	No	SET_WR_BLOCK_ERASE_COUNT	For only SDC. Define number of blocks to pre-erase with next multi-block write command.
CMD24	Address[31:0]	R1	Yes	WRITE_BLOCK	Write a block.
CMD25	Address[31:0]	R1	Yes	WRITE_MULTIPLE_BLOCK	Write multiple blocks.
CMD55(*1)	None(0)	R1	No	APP_CMD	Leading command of ACMD<n> command.
CMD58	None(0)	R3	No	READ_OCR	Read OCR.
*1:ACMD<n> means a command sequense of CMD55-CMD<n>. *2: Rsv(0)[31], HCS[30], Rsv(0)[29:0] *3: Rsv(0)[31:12], Supply Voltage(1)[11:8], Check Pattern(0xAA)[7:0]					

Εικόνα 5.3:Λίστα μερικών εντολών SD κάρτας
(http://people.ece.cornell.edu)

Φυσικά πριν τις εντολές πρέπει να γίνει πρώτα μία διαδικασία αναγνώρισης και αρχικοποίησης. Η διαδικασία διαφέρει αναλόγως του τύπου της κάρτας. Η κάρτα που χρησιμοποιείται στην παρούσα εργασία είναι τύπου SDHC (Secure Digital High-Capacity) και η διαδικασία διαφοροποιείται σε σχέση με τη κάρτα τύπου SD.

Η διαδικασία αρχικοποίησης έχει ως εξής, αρχικά πρέπει να ζεσταθεί (warm-up) η κάρτα, στέλνοντας 74 τουλάχιστον παλμούς έχοντας όμως τον ακροδέκτη SS σε λογικό ένα (5V).

Στη συνέχεια αποστέλλεται η εντολή CMD0 αφού επιστραφεί απάντηση διάφορη του R1 δηλαδή ένα (1), αποστέλλεται η εντολή CMD8 εάν απαντήσει με 0x01AA, τότε αποστέλλεται η εντολή CMD41 και την CMD55 έως ότου η CMD41 επιστρέψει 0. Τέλος αποστέλλεται η CMD58 όπου και η απάντηση καθορίζει αν είναι HC ή SD, στην προκειμένη περίπτωση HC.

Επειδή δεν χρησιμοποιείται σύστημα αρχείων, στην ουσία δεν υπάρχει αρχείο αλλά μόνο δεδομένα σε δεκαεξαδική μορφή (hex), πρέπει να υπάρχει κάποιος δείκτης που να δείχνει ποιος είναι ο επόμενος Sector για εγγραφή. Ο δείκτης αποθηκεύεται στην ίδια την κάρτα στον Sector 9 και είναι τα δύο (2) πρώτα byte τα οποία και μεταφράζονται σε δεκαδικό, φτάνοντας μέγιστο αριθμό μετρήσεων τις 65.535.

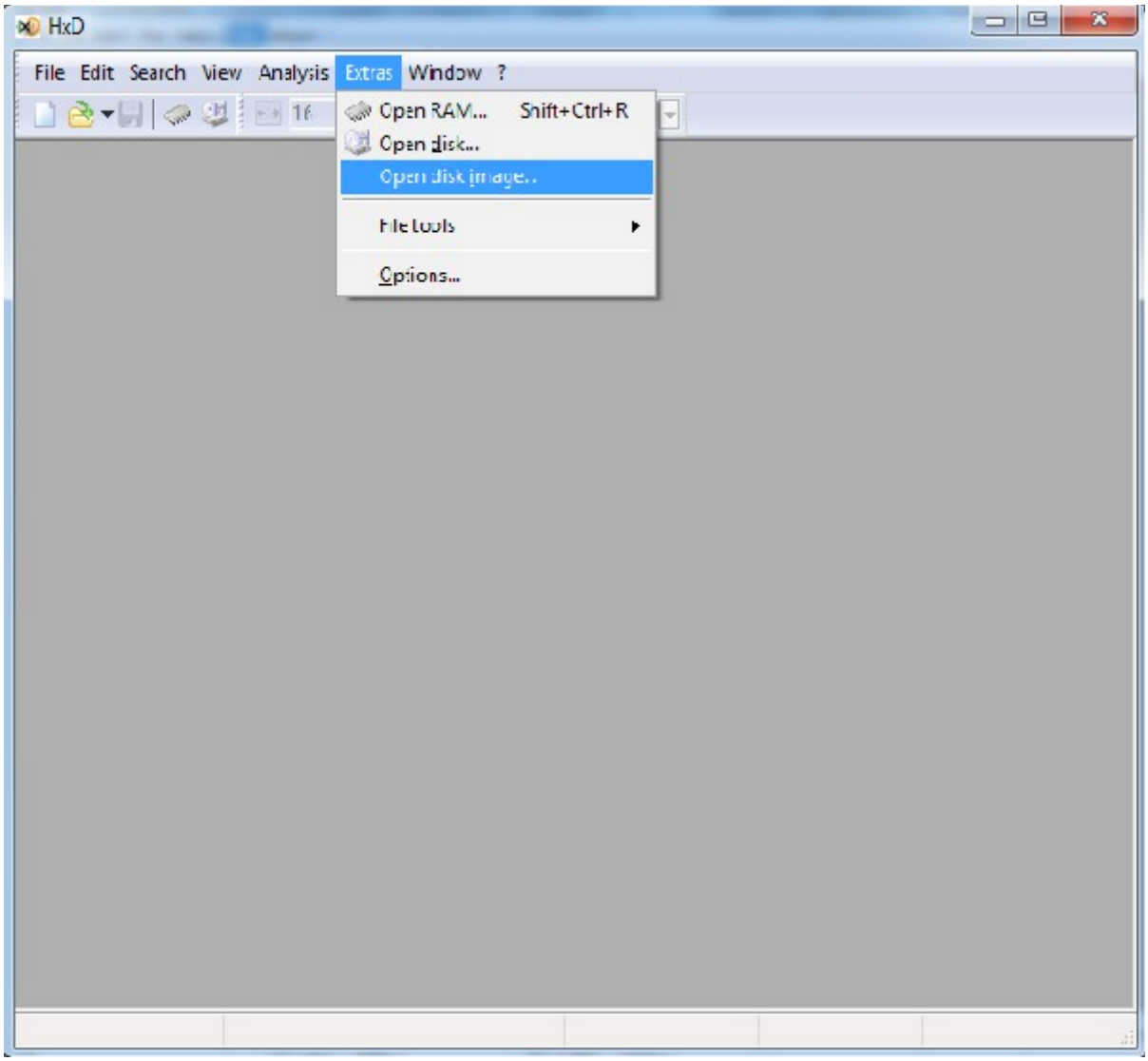
Πιο αναλυτικά αφού ληφθούν οι μετρήσεις απο τον pic που ελέγχει τους αισθητήρες τότε ο μικροελεγκτής διαβάζει τον sector 9 της κάρτας μνήμης, παίρνει τα δύο (2) πρώτα bytes και τα μετατρέπει σε δεκαδικό, κατόπιν εγγράφει στον sector που υποδεικνύει ο sector 9. Στη συνέχεια αυξάνει τον αριθμό της διεύθυνσης κατά ένα και αποθηκεύεται πάλι στον sector 9. Να σημειωθεί πως πρέπει να γεμίσει όλο το sector (512 bytes) σε κάθε εγγραφή, οπότε συμπληρώνεται τυχαία με το γράμμα p.

5.3 Εξαγωγή μετρήσεων απο την κάρτα

Για την εξαγωγή των μετρήσεων πρέπει να χρησιμοποιηθεί ειδικό λογισμικό, μιας και δεν υπάρχει σύστημα αρχείων. Μια λύση είναι το HxD (https://mh-nexus.de/en/hxd/), το οποίο η χρήση και η διανομή του, είναι δωρεάν.
(Περισσότερα για την άδεια στο https://mh-nexus.de/en/hxd/license.php,)

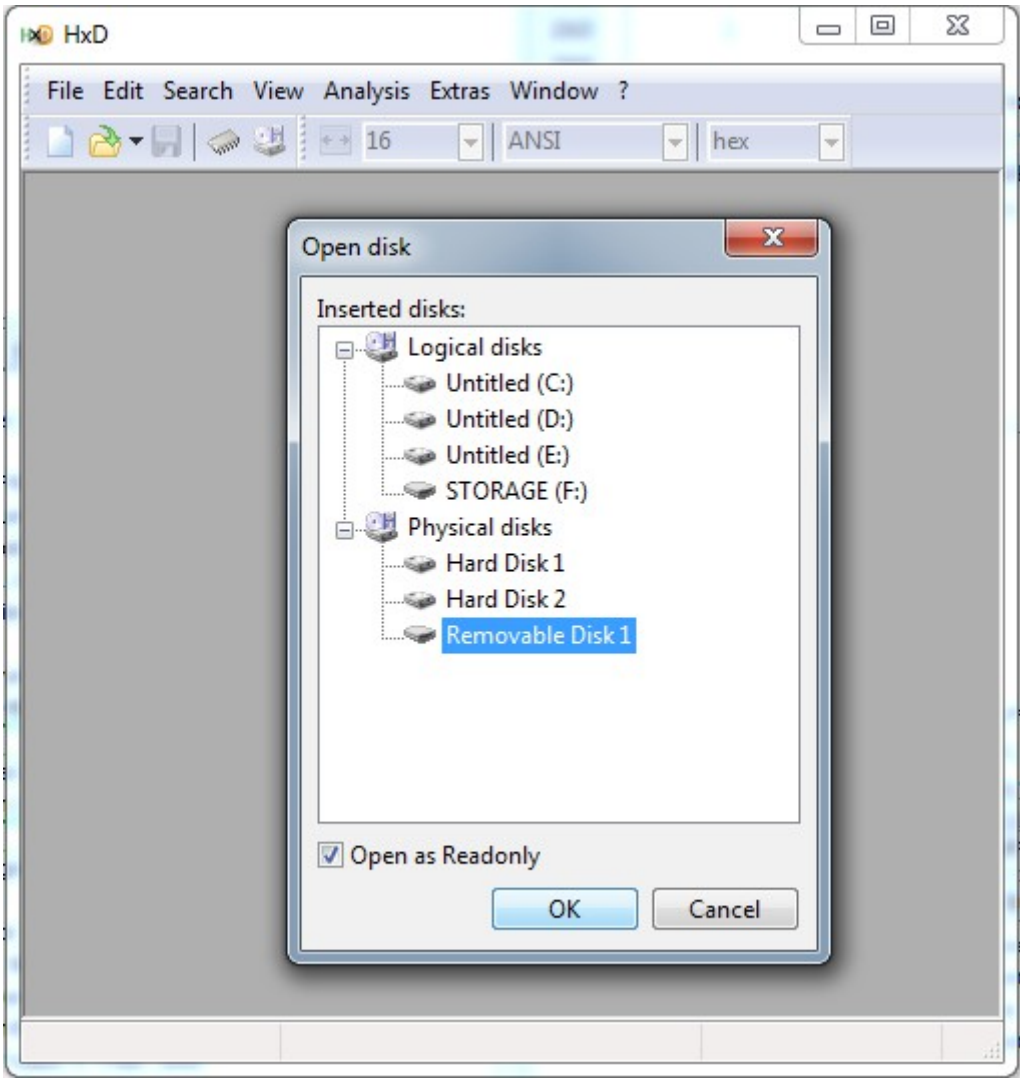
Η διαδικασία έχει ως εξής :

Αρχικά τοποθετούμε την κάρτα στον υπολογιστή, με ειδικό αντάπτορα. Αφού την αναγνωρίσει, ανοίγουμε την εφαρμογή HxD.
Στη συνέχεια MENU → EXTRAS → Open disk



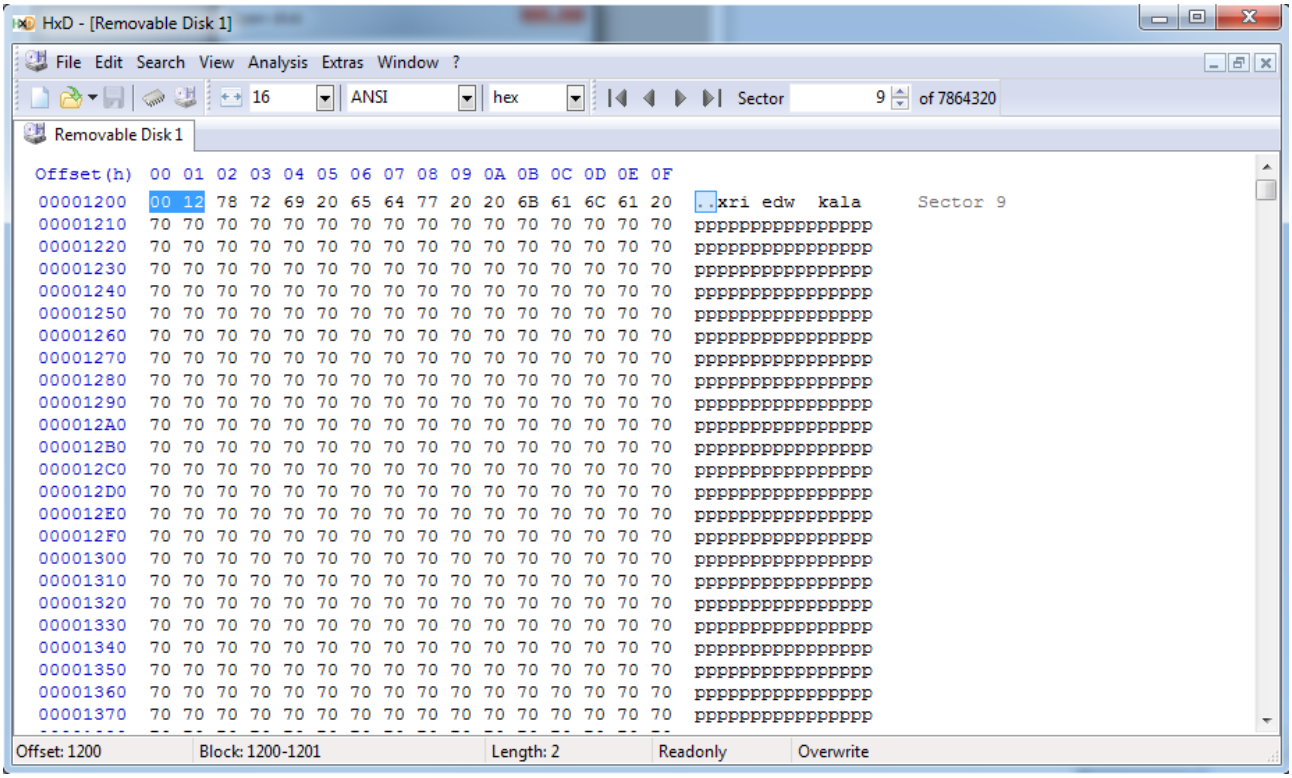
Εικόνα 5.4: Βήμα πρώτο για εξαγωγή δεδομένων της κάρτας μνήμης.

Έπειτα στο παράθυρο που ανοίγει, επιλέγουμε απο την λίστα Physical disks την κάρτα μνήμης.



Εικόνα 5.5: Βήμα δεύτερο για εξαγωγή δεδομένων της κάρτας μνήμης.

Αφού ανοίξει το παράθυρο πηγαίνουμε στον sector 9 και παρατηρούμε τα δύο πρώτα byte.



Εικόνα 5.6: Βήμα τρίτο για εξαγωγή δεδομένων της κάρτας μνήμης.

Τα δυο πρώτα byte είναι 0012 σε δυαδικό 0000 0000 0001 0010 που μεταφράζεται σε δεκαοκτώ (18) σε δεκαδικό, άρα η τελευταία μέτρηση βρίσκεται στον sector 17 και η επόμενη θα εγγραφεί στον sector 18. Άρα οι μετρήσεις βρίσκονται απο τον sector 10 έως και τον 17, μία μέτρηση ανά sector.

ΚΕΦΑΛΑΙΟ 6
GSM MODULE

Εισαγωγή

Στην παρούσα εργασία οι μετρήσεις μεταδίδονται μέσω δικτύου κινητής τηλεφωνίας. Το **GSM** (Global System for Mobile communication) **MODULE** είναι το κομμάτι που διευκολύνει αυτή τη διεργασία. Το GSM MODULE είναι το υποσύστημα το οποίο διαχειρίζεται την επικοινωνία ενός συστήματος, όπως ένας υπολογιστής ή ένα σύστημα με μικροελεγκτή, με το δίκτυο κινητής τηλεφωνίας

Συνήθως αποτελείται απο ένα GSM modem, ένα σύστημα τροφοδοσία και κάποια διεπαφή επικοινωνίας, όπως USB, IIC κ.α..

Το GSM modem είναι ένας ειδικός τύπος modem το οποίο λειτουργεί με μία κάρτα **SIM** (**subscriber identity module** ή **subscriber identification module**) και στην ουσία αποτελεί μια διεπαφή επικοινωνίας ενός συστήματος με το δίκτυο κινητής τηλεφωνίας. Το GSM modem χρησιμοποιεί σειριακή επικοινωνία και δέχεται εντολές τύπου AT.

Στην εργασία χρησιμοποιούμε το Adafruit FONA GSM module της Adafruit με GSM modem το SIM800L της SimCom.

6.1 Adafruit FONA- SIM 800L

Το Adafruit Fona (εικόνα 6.1) είναι ένα GSM module που επιτρέπει τη διεκπεραίωση κλήσεων, αποστολή/παραλαβή μηνυμάτων SMS (**Short Message Service**) και χρήση δεδομένων δικτύου. Είναι ιδανικό για έλεγχο με μικροελεγκτή, αλλά με ένα TTL (**Transistor-Transistor-Logic**)-to-USB (**Universal Serial Bus**) καλώδιο μπορεί εύκολα να συνδεθεί με έναν υπολογιστή.
Λειτουργεί με εξωτερική πηγή τάσης, μια μπαταρία Πολυμερών Λιθίου ή Ιόντων Λιθίου, με περιορισμό να είναι μεγαλύτερη ή ίση με 500mAh (milli Ambere per hour).

Επίσης είναι εξοπλισμένο με ό,τι χρειάζεται για να καταστεί εύκολη η χρήση του, όπως Led ένδηξης ισχύς σήματος, σύστημα ελέγχου τάσης μπαταρίας, micro-USB για φόρτιση της μπαταρίας, αλλά και πολλά άλλα με κύριο χαρακτηριστικό το μικρό του μέγεθος.

Ειδικότερα διαθέτει, micro-USB θύρα για φόρτιση μπαταρίας αλλά και για προγραμματισμό του modem. Βύσμα για ακουστικά (4-pole TRRS headphone jack), ακροδέκτες για χρήση εξωτερικού ηχείου τύπου 8Ω αλλά και για μικρόφωνο, μετατροπέα τάσης ούτως ώστε είναι συμβατό με μικροελεγκτές τάσης 2.8V έως 5V. Σύστημα ελέγχου ηλεκτροκινητήρα δόνησης για αθόρυβη ειδοποίηση, SMA (SubMiniature version A) βύσμα για την κεραία και μια ειδική θύρα τύπου slide για την κάρτα SIM.

Εικόνα 6.1: Adafruit Fona.

Συνολικά το Adafruit Fona έχει 12 ακροδέκτες, τους BAT, GND, SPKR +, SPKR - , Rst, ps, key, RI, TX, RX, NS και Vio. Ο ακροδέκτης BAT είναι για οδήγηση ηλεκτροκινητήρα δόνησης, ο GND για σύνδεση με τη γείωση, ο SPKR+ και SPKR – για ηχείο, ο Rst επανεκκινεί την συσκευή, ο ps δείχνει εάν η συσκευή είναι ενεργή ή ανενεργή με λογικό ένα (1) για ενεργή και λογικό μηδέν (0) για ανενεργή. Ο ακροδέκτης key ενεργοποιεί ή απενεργοποιεί τη συσκευή εάν απο λογικό ένα (1) ο μικροελεγκτής ρίξει τη τάση σε λογικό μηδέν (0) για δύο δευτερόλεπτα και λειτουργεί με μέθοδο toggle. Ο ακροδέκτης RI (Ring Indicator) εκπέμπει ένα παλμό στον ελεγκτή όταν υπάρχει εισερχόμενη κλήση ή SMS. Οι ακροδέκτες RX και TX είναι η θύρα σειριακής επικοινωνίας, ο RX είναι ο παραλήπτης και ο TX αυτός που εκπέμπει. Ο NS (Network Status) εκπέμπει σε μορφή παλμού την κατάσταση δικτύου. Τέλος ο Vio τροφοδοτεί το μετατροπέα τάσης αναγνωρίζοντας και μετατρέποντας στην κατάλληλη τάση, 5V (volt) στην παρούσα εργασία. Παρακάτω υπάρχει και πίνακας με τους ακροδέκτες.

BAT	Οδήγηση ηλεκτροκινητήρα δόνησης.
GND	Γείωση.
SPKR+	Θετικός πώλος ακουστικών.
SPKR-	Αρνητικός πώλος ακουστικών.
Rst	Επανεκκίνηση συσκευής.
Ps	Ένδειξη λειτουργίας.
Key	Ενεργοποίηση/Απενεργοποίηση συσκευής.
RI	Ένδειξη εισερχόμενης κλήσης.
TX	Θύρα σειριακής επικοινωνίας.
RX	Θύρα σειριακής επικοινωνίας.
NS	Κατάσταση δικτύου.
Vio	Τάση λειτουργίας.

Εικόνα 6.2: Πίνακας ακροδεκτών Adafruit Fona.

6.2 AT commands (εντολές AT)

Οι εντολές AT είναι το σύνολο εντολών που χρησιμοποιούνται για τον έλεγχο ενός modem. Η λέξη AT είναι συντομογραφία της αγγλικής λέξης ATtension. Μία εντολή AT πάντα ξεκινά με το πρόθεμα AT. Υπάρχουν δύο τύποι εντολών, οι βασικές και οι σύνθετες. Οι βασικές μετά το AT δεν έχουν το σύμβολο + αλλά απλά την εντολή, για παράδειγμα ATD (dial). Οι σύνθετες μετά το AT έχουν το σύμβολο + και μετά ακολουθεί η εντολή, όπως για παράδειγμα στο Sim800L η σύνθετη εντολή AT+CBC επιστρέφει την κατάσταση της μπαταρίας. Μετά απο κάθε εντολή πρέπει να ακολουθεί ο ειδικός χαρακτήρας "CR" (\r, carriage return).

Το Adafruit FONA υποστηρίζει τέσσερις (4) τύπους εντολών (εικόνα 5-3) τις Test Commands, Read commands, write commands, και execution commands. Οι Test commands επιστρέφουν μια λίστα με τα ορίσματα και την εμβέλεια αυτών, της εντολής που δόθηκε. Οι Read commands επιστρέφουν την τρέχουσα τιμή της εντολής που δόθηκε. Οι Write commands στέλνουν στο modem εντολές με συμπληρωμένα ορίσματα. Τέλος οι Execution Commands είναι οι εντολές χωρίς ορίσματα.

Test Command	AT+<x>=?	The mobile equipment returns the list of parameters and value ranges set with the corresponding Write Command or by internal processes.
Read Command	AT+<x>?	This command returns the currently set value of the parameter or parameters.
Write Command	AT+<x>=<…>	This command sets the user-definable parameter values.
Execution Command	AT+<x>	The execution command reads non-variable parameters affected by internal processes in the GSM engine.

Εικόνα 6.3: Τύποι εντολών Adafruit Fona απο φύλλο δεδομένων.

Στη παρούσα εργασία χρησιμοποιούνται τέσσερις (4) εντολές οι AT, AT+CSCS="GSM", AT+ CMGF=1 και AT+CMGS="*τηλεφωνικό νούμερο*". Η εντολή AT χρησιμοποιείται για συντονισμό μεταξύ μικροελεγκτή και του GSM module. Με την εντολή AT+CSCS="GSM" επιλέγω το είδος των χαρακτήρων που θα σταλούν μέσω SMS, με το όρισμα GSM το θέτω σαν το κλασικό 7 bit ανά χαρακτήρα. Η εντολή AT+ CMGF=1 αλλάζει την μορφή των μηνυμάτων SMS σε text mode. Τέλος η εντολή AT+CMGS="*τηλεφωνικό νούμερο*" στέλνει το SMS. Η τελευταία εντολή είναι πιο σύνθετη απο τις υπόλοιπες. Μετά το τηλεφωνικό νούμερο, το modem περιμένει το μήνυμα που πρόκειται να αποσταλεί. Το σήμα ολοκλήρωσης του μηνύματος και αποστολής είναι ο ειδικός χαρακτήρας Ctrl^Z ή 0x1A σε δεκαεξαδικό.

ΣΥΜΠΕΡΑΣΜΑΤΑ - ΒΕΛΤΙΩΣΕΙΣ

Μία απο της μεγαλύτερες δυσκολίες που αντιμετώπισα, ήταν ο προγραμματισμός των μικροελεγκτών, ο οποίος έγινε με ένα απλό JDM προγραμματιστή που έφτιαξα με τη βοήθεια DIY (Do It Yourself) site (http://www.instructables.com/id/Minty-JDM-PIC-Programmer/). Ο προγραμματιστής ήταν αργός, και αποτύγχανε συχνά με αποτέλεσμα μεγάλης καθυστέρησης ακόμα και για αλλαγές της μίας εντολής στον κώδικα.

Επίσης η εξοικείωση με το περιβάλλον προγραμματισμού MPLABX και τα εργαλεία αυτού, κυρίως το STIMULUS και IOPin, με βοήθησαν σε μεγάλο βαθμό στην αποσφαλμάτωση του κώδικα.

Η εύρεση αισθητήρων και άλλων υποσυστημάτων, συμβατά με την εργασία ήταν και αυτό μια πρόκληση, καθώς η πώληση αυτών στην ελληνική αγορά, είναι ιδιαίτερα περιορισμένη, έτσι η αγορά αυτών πρέπει να γίνει νωρίς μιας και η εισαγωγή του απο το εξωτερικό μπορεί να καθυστερήσει.

Επίσης σε πολλά υποσυστήματα έπρεπε να κολλήσω τους ακροδέκτες, μια διαδικασία η οποία αν δεν γίνει σωστά μπορεί να καταστρέψει το υποσύστημα, έτσι έπρεπε να το δοκιμάσω για να διαπιστώσω ότι λειτουργεί σωστά, ώστε να μην σπαταλήσω χρόνο, πιστεύοντας οτι είναι σφάλμα λογισμικού. Η δοκιμή έγινε με το Aduino Mega, χρησιμοποιώντας βιβλιοθήκες που συνήθως δίνονται απο τον κατασκευαστή των διαφόρων συστημάτων, διασφάλιζα πως το υποσύστημα δούλευε χωρίς σφάλματα.

Μια ακόμη δυσκολία ήταν η χρήση του GSM module, το οποίο όμως συνδέεται εύκολα σε υπολογιστή με ειδικό καλώδιο και λογισμικό, έτσι μπορούσα να δοκιμάσω εντολές, χωρίς να χρησιμοποιήσω μικροελεγκτή, αλλά απο το πληκτρολόγιο.

Ένα απο τα δυσκολότερα κομμάτια της εργασίας ήταν η μεταφορά των μετρήσεων ανάμεσα στους δύο μικροελεγκτές, μιας και η σειριακή θύρα του ενός χρησιμοποιούταν για το GSM module, και η θύρα MSSP ήταν για χρήση πρωτοκόλλου SPI για τον ένα και IIC για τον άλλο. Έτσι έπρεπε να χρησιμοποιήσω ακροδέκτες γενικής χρήσης.

Τέλος, η εισαγωγή μετρήσεων στη κάρτα μνήμης χωρίς συστήματος αρχείων, ήταν περιπλοκή, καθώς κάθε μέτρηση χρησιμοποιεί διαφορετικό Sector, ο οποίος με κάποιο τρόπο έπρεπε να αποθηκευτεί, έτσι ώστε να μην γίνει η εγγραφή η μία πάνω στην άλλη. Η λύση ήταν η αποθήκευση του δείκτη εγγραφής, ο δείκτης δηλαδή που δείχνει σε πιο Sector να γίνει η επόμενη εγγραφή, σε συγκεκριμένο Sector της κάρτας μνήμης.

Η εργασία έχει αρκετούς τομείς που μπορούν να βελτιωθούν. Αρχικά η εισαγωγή ενός RTC (Real Time Clock) θα μπορούσε να μετατρέψει την εργασία απο one-shot (μίας χρήσης μετά την ενεργοποίηση) σε κανονικό μετεωρολογικό σταθμό, που θα παίρνει μετρήσεις σε τακτά χρονικά διαστήματα. Επίσης η εισαγωγή συστήματος αρχείων, θα βοηθούσε σε εξοικονόμηση χώρου της κάρτας μνήμης, αλλά και σε ευκολία εξαγωγής δεδομένων απο αυτή.

Μια ακόμη βελτίωση είναι και η αλλαγή του μικροελεγκτή με κάποιον άλλο ο οποίος θα έχει περισσότερες θύρες MSSP, έτσι ώστε να εξαλειφθεί η ανάγκη δεύτερου μικροελεγκτή που περιπλέκει το σύστημα.

Τέλος η εισαγωγή φωτοβολταϊκών πάνελ θα μεγάλωνε την αυτονομία του συστήματος έτσι ώστε να είναι διαθέσιμος και για παρακολούθηση απομακρυσμένων περιοχών.

ΠΑΡΑΡΤΗΜΑ Α
ΚΩΔΙΚΑΣ MASTER

```
#define _XTAL_FREQ 8000000 //Ενημέρωση του Compiler σχετικά με τον χρονισμό.

#define ACKBit RE0          // Θέτω τον ακροδέκτη RE0 σαν  ACKBit, χρησιμοποιείται στη
                            //μεταφορά δεδομένων απο master σε slave.

                            //Οι παρακάτω ορισμοί χρησιμοποιούνται στη βιβλιοθήκη της LCD.
#define RS LATD1            // Θέτω τον ακροδέκτη RS σαν   LATD1.
#define EN LATD0            // Θέτω τον ακροδέκτη EN σαν   LATD0.
#define D4 LATD4            // Θέτω τον ακροδέκτη D4 σαν   LATD4.
#define D5 LATD5            // Θέτω τον ακροδέκτη D5 σαν   LATD5.
#define D6 LATD6            // Θέτω τον ακροδέκτη D6 σαν   LATD6.
#define D7 LATD7            // Θέτω τον ακροδέκτη D7 σαν   LATD7.

#include <xc.h>             //Η βασική βιβλιοθήκη που περιέχει ορισμούς για το PIC.
#include "lcd.h"            //Η βιβλιοθήκη που περιέχει εντολές της LCD.
#include <plib/usart.h>     //Περιφερειακή βιβλιοθήκη για σειριακή επικοινωνία.
#include <math.h>           //Βασική βιβλιοθήκη με μαθηματικές συναρτήσεις

 //Παρακάτω ορίζεται πως θα χρησιμοποιηθούν διάφορα περιφερειακά και χρονιστές αλλά και
//ακροδέκτες του PIC
#pragma config CPUDIV = OSC1_PLL2
#pragma config PLLDIV = 1
#pragma config USBDIV = 1
#pragma config FCMEN  = OFF
#pragma config IESO  = OFF
#pragma config FOSC  = INTOSCIO_EC
#pragma config PWRT  = OFF
#pragma config VREGEN  = OFF
#pragma config BORV  = 0
#pragma config BOR = OFF
#pragma config WDTPS = 16384
#pragma config WDT = OFF
#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config MCLRE = ON
#pragma config LPT1OSC = ON
#pragma config STVREN = OFF
#pragma config DEBUG = OFF
#pragma config ICPRT = OFF
#pragma config LVP = OFF
#pragma config XINST = OFF
#pragma config CP0  = OFF
#pragma config CP1  = OFF
#pragma config CP2  = OFF
#pragma config CP3  = OFF
```

```
                        //Οι παρακάτω μεταβλητές χρησιμοποιούνται για την επικοινωνία του
                        //Pic με το GSM Module.
unsigned char ret[24];
unsigned char UART1Config = 0, baud = 0;
unsigned char MsgFromPIC[10];
unsigned char MsgFromPIC2[12];
unsigned char SmsBuffer[30];
unsigned char ATGSM[15];
void setTables(void);



                        // Οι παρακάτω μεταβλητές και συναρτήσεις χρησιμοποιούνται για
                        //την εφαρμογή του πρωτοκόλλου IIC ( I2C ).
unsigned int ACK_bit;
int byte, temp = 0;

void I2C_ACK(void);
void Send_I2C_Data(unsigned int databyte);
int RX_I2C_Data (void);
void I2C_Start_Bit(void);
void I2C_check_idle();
void I2C_restart();
void I2C_Stop_Bit(void);
void I2C_NAK(void);
void InitMCU(void);
                        //Συναρτήσεις για επικοινωνία με αισθητήρες.
int getHum(void);
int getTempHDC(void);
                        //Συναρτήσεις για επικοινωνία Pic με Pic χρησιμοποιώντας
                        //ακροδέκτες γενικού σκοπού/
void CPStart(void);
int CPACK(void);
void CPSent(int number);
int CPNACK(void);

int i=0;
int shu=0,stem=0;

void Delay1Second()             //Συνάρτηση καθυστέρησης ενός δευτερολέπτου.
{
   for(i=0;i<100;i++)
   {
       __delay_ms(10);
   }
};

void main(int argc, char** argv) {    //Εναρξη προγράμματος
```

```
InitMCU();                      // Αρχικοποίηση μικροελεγκτή
Lcd_Init();                     // Αρχικοποίηση LCD
setTables();                    //Τοποθέτηση AT εντολών σε πίνακες

__delay_ms(50);                 //Μια μικρή καθυστέρηση για τις εντολές αρχικοποίησης


Lcd_Clear();                    //Καθαρισμός LCD
Lcd_Set_Cursor(1,1);            //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD
Lcd_Write_String(MsgFromPIC2);      //Εμφάνιση AT εντολής στην LCD
Lcd_Set_Cursor(2,1);            //Τοποθέτηση κέρσορα στο σημείο 1,2 της LCD
Lcd_Write_String(MsgFromPIC);//Εμφάνιση AT εντολής στην LCD

for(int dly=0;dly<20;dly++){    //Μια καθυστέρηση 20 δευτερολέπτων για να δοθεί αρκετός
                                //χρόνος στο GSM Module να ανοίξει

   Delay1Second();

}

Lcd_Set_Cursor(1,1);            //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD
Lcd_Clear();                    //Καθαρισμός LCD
Lcd_Set_Cursor(1,1);            //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD




int temp1 = 0,temp2=0;          //Ορισμός μεταβλητών που θα γεμίσουν με τα αποτελέσματα
                                //των μετρήσεων
temp1 = getTempHDC();           //Καλώ την  getTempHDC() και εκχωρώ το αποτέλεσμα στην
                                //temp1
temp2 = getHum();               //Καλώ την  getHum()() και εκχωρώ το αποτέλεσμα στην
                                //temp2
//Διαδικασία μετατροπής μέτρησης σε ποσοστό υγρασίας
float hum = (float)temp2/65536;
hum=hum*100;
shu = hum*(-1);                 //Η μεταβλητή shu περιέχει το ποσοστό υγρασίας
//Διαδικασία μετατροπής μέτρησης σε μονάδες Celsius
float temperature = (float)temp1/65536;
temperature = temperature*165;
temperature = temperature - 45;
stem = temperature;             //Η μεταβλητή stem περιέχει τη θερμοκρασία
// Διαδικασία μετατροπής της θερμοκρασίας απο float σε String για αποστολή στο GSM Module
//και την οθόνη  LCD, με τη βοήθεια ειδικής συνάρτησης sprintf().
 sprintf(ret, "Temp=%fC, Hum=%f", temperature,hum);
 ret[strlen(ret)] = '\0';
 ret[strlen(ret)+1] = '\r';
```

```
  Lcd_Clear();                    //Καθαρισμός LCD
  Lcd_Set_Cursor(1,1);            //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD
  Lcd_Write_String(ret);          //Εγγραφή στην LCD τα αποτελέσματα των μετρήσεων

   //Προετοιμασία σειριακής θύρα
        //όχι interrupts σε λήψη και μετάδοση, ασύγχρονη επικοινωνία, 8-bit μετάδοση
   UART1Config = USART_TX_INT_OFF & USART_RX_INT_OFF &
USART_ASYNCH_MODE & USART_EIGHT_BIT & USART_BRGH_HIGH ;
   baud = 51; // ρυθμίζω το Baud Rate se 9600, Fosc / (16 * ( baud + 1))
   OpenUSART(UART1Config,baud);       //Εναρξη σειριακής θύρα


   putsUSART(MsgFromPIC);             //εντολή AT : AT
   Delay1Second();                    //καθυστέρηση για να δοθεί χρόνος επεξεργασίας

   putsUSART(ATGSM);                  // εντολή AT  : AT+CSCS = "GSM"
   Delay1Second();                    //καθυστέρηση για να δοθεί χρόνος επεξεργασίας

   putsUSART(MsgFromPIC2);            // εντολή AT : AT+CMGF = 1
   Delay1Second();                    //καθυστέρηση για να δοθεί χρόνος επεξεργασίας

   putsUSART(SmsBuffer);               // εντολή AT : AT+CMGS ="τηλεφωνικό νούμερο"
   Delay1Second();                    //καθυστέρηση για να δοθεί χρόνος επεξεργασίας

   putsUSART(ret);                    // Οι μετρήσεις
   Delay1Second();                    //καθυστέρηση για να δοθεί χρόνος επεξεργασίας

   MsgFromPIC[0] = 0x1A;              //Ctrl^Z για να βγει απο το SMS mode
   MsgFromPIC[1] = '\r';              //Λήξη αποστολής
   putsUSART(MsgFromPIC);             //Αποστολή των δύο παραπάνω χαρακτήρων

        //Αποστολή των μετρήσεων στον δεύτερο μικροελεγκτή. Για απλούστευση της διαδικασίας
        //αποστέλλεται μόνο το ακέραιο κομμάτι των μετρήσεων
   int t[4];
   int sc;
   t[0]=(stem/10)%10;                 //Παίρνω το πρώτο ψηφίο της θερμοκρασίας
   t[1]=stem%10;                      //Παίρνω το δεύτερο ψηφίο της θερμοκρασίας
   t[2]=(shu/10)%10;                  //Παίρνω το πρώτο ψηφίο της υγρασίας
   t[3]=shu%10;                       //Παίρνω το δεύτερο ψηφίο της υγρασίας
   for(sc=0;sc<4;sc++){

      CPStart();                      //Αποστολή σήματος έναρξης αποστολής

      do{
      }while(!CPACK());               //Αναμονή αποδοχή αποστολής

      CPSent(t[sc]);                  //Αποστολή ψηφίου

      do{
      }while(!CPNACK());              //Ελεγχος αποδοχής λήψης
```

```
    }
//Τέλος προγράμματος, χωρίς να τερματίζει ποτέ για να μην υπάρχουν απρόβλεπτες συμπεριφορές.
    while(1)
    {

    }
}
//Συνάρτηση αρχικοποίησης
void InitMCU()
{
                                        //Ρύθμιση χρονισμού του PIC σε 8MHz
    OSCCONbits.IRCF0 = 1;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF2 = 1;


    TRISC = 0xFF;                       //port C έξοδος για επικοινωνία I2C
    TRISD = 0x00;                       //port D έξοδος για επικοινωνία με την lcd

    SSPCON1bits.SSPM=0x08;               // I2C Master mode, clock = Fosc/(4 * (SSPADD+1))
    SSPCON1bits.SSPEN=1; // ενεργοποίηση θύρας MSSP, κάνει τα SDA κ SCK I2C = 1 kai i/o = 0
    SSPADD = 0x09;

    TRISCbits.RC6 = 0;                  //ο ακροδέκτης TX γίνετε έξοδος
    TRISCbits.RC7 = 1;                  //ο ακροδέκτης RX γίνετε είσοδος

    ADCON1 = 0x0F;                      //Απενεργοποίηση A/D λειτουργίας στη θύρα A
    TRISA = 0x00;                       //Η θύρα A γίνεται έξοδος για επικοινωνία με slave
    TRISE0 = 0b1;                       //Ο ακροδέκτης E0 γίνεται έξοδος για επικοινωνία με
                                        //slave
    PORTA = 0x00;     //Μηδενίζεται η έξοδος της θύρας A για να μην γίνει τυχαία έναρξη
                                        //αποστολής στον δεύτερο PIC
}



//Συνάρτηση αποστολής σήματος ACK στο πρωτόκολλο IIC

void I2C_ACK(void)
{
 PIR1bits.SSPIF=0;                      //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
 SSPCON2bits.ACKDT=0;                   //ο καθαρισμός αυτής της σημαίας υποδεικνύει
                                        //αποστολή ACK
 SSPCON2bits.ACKEN=1;                   // Έναρξη αποστολής
 while(!PIR1bits.SSPIF);                // Αναμονή τερματισμού μετάδοσης
}
////Συνάρτηση αποστολής δεδομένων στο πρωτόκολλο IIC
void Send_I2C_Data(unsigned int databyte)
{
```

```
 PIR1bits.SSPIF=0;                          //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
 SSPBUF = databyte;                         //Γεμίζοντας τον καταχωρητή SSPBUF με δεδομένα,
                                            //ξεκινά αυτομάτως η αποστολή τους
 while(!PIR1bits.SSPIF);                    // Αναμονή τερματισμού μετάδοσης
}
////Συνάρτηση παραλαβής δεδομένων στο πρωτόκολλο IIC
int RX_I2C_Data (void)
{
 byte = 0;
 RCEN = 1;                                  //Θέτοντας τη σημαία,σηματοδοτεί αναμονή
δεδωμένων
 while( RCEN ) continue;                    //Αναμονή προετοιμασίας του μικροελεγκτή
 while( !BF ) continue;                     //Αναμονή πτώσης BF που σημαίνει άφιξη δεδομένων
 byte = SSPBUF;                             //Παραλαβή δεδομένων απο τον καταχωρητή SSPBUF
 return byte;
}
////Συνάρτηση αποστολής αλληλουχίας έναρξης στο πρωτόκολλο IIC
void I2C_Start_Bit(void)
{
 PIR1bits.SSPIF=0;                          //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
 SSPCON2bits.SEN=1;        //Θέτοντας τη σημαία,σηματοδοτεί αποστολή αλληλουχίας έναρξης
 while(!PIR1bits.SSPIF)                     //Αναμονή αποστολής
 {
         i = 1;
 }
 PIR1bits.SSPIF=0;                          //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
}

////Συνάρτηση αποστολής αλληλουχίας τερματισμού στο πρωτόκολλο IIC
void I2C_Stop_Bit(void)
{
 PIR1bits.SSPIF=0;                          //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
 SSPCON2bits.PEN=1;                         //Θέτοντας τη σημαία,σηματοδοτεί αποστολή
                                            //αλληλουχίας τερματισμού
 while(!PIR1bits.SSPIF)                     //Αναμονή αποστολής
 {
        i = 1;
 }
}
//Συνάρτηση αποστολής σήματος  NACK στο πρωτόκολλο IIC
void I2C_NAK(void)
{
 PIR1bits.SSPIF=0;                          //καθαρισμός σημαίας υπόδειξης επιτυχούς αποστολής
 SSPCON2bits.ACKDT=1;                       //θέτοντας αυτή της σημαίας υποδεικνύει
                                            //NAK
 SSPCON2bits.ACKEN=1;                       // Έναρξη αποστολής
 while(!PIR1bits.SSPIF)                     //Αναμονή αποστολής
 {
 i = 1;
 }
}
```

```
//Συνάρτηση επικοινωνίας με αισθητήρα για μέτρηση υγρασίας με το πρωτόκολλο IIC
int getHum(){
      int dc = 0;
   I2C_Start_Bit();                    //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x80);               //Αποστολή διεύθυνσης με R/W = 0
   if (!SSPCON2bits.ACKSTAT){          //Αναμονή παραλαβή ACK απο αισθητήρα
         __delay_ms(50);
      }

    Send_I2C_Data(0x02);               //Αποστόλη διεύθυνσης δείκτη 2
      if (!SSPCON2bits.ACKSTAT){       //Αναμονή παραλαβή ACK απο αισθητήρα
         __delay_ms(50);
      }
    Send_I2C_Data(0x06);               //Ορίζω το Configuration του αισθητήρα
      if (!SSPCON2bits.ACKSTAT){       //Αναμονή παραλαβή ACK απο αισθητήρα
         __delay_ms(50);

      }

    I2C_Stop_Bit();                    //Αποστολή  αλληλουχίας τερματισμού


   for(dc=0;dc<50;dc++){               //Αναμονή ολοκλήρωσης μετρήσεων του αισθητήρα
      __delay_ms(50);
   }

    I2C_Start_Bit();                   //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x80);               //Αποστολή διεύθυνσης με R/W = 0
    if (!SSPCON2bits.ACKSTAT){         //Αναμονή παραλαβή ACK απο αισθητήρα
      for(dc=0;dc<4;dc++){
         __delay_ms(50);
      }
   }
    Send_I2C_Data(0x01);               //Αποστολή διεύθυνσης δείκτη 1 που περιέχει την
                                       //μέτρηση της υγρασίας


    I2C_Stop_Bit();                    //Αποστολή  αλληλουχίας τερματισμού

   for(dc=0;dc<50;dc++){               //Αναμονή ολοκλήρωσης μετρήσεων του αισθητήρα
    __delay_ms(50);
   }

    I2C_Start_Bit();                   //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x81);               //Αποστολή διεύθυνσης με R/W = 1
    RX_I2C_Data();                     //Παραλαβή πρώτου byte
    temp = byte;
    temp = temp << 8;                  //Ολίσθηση αριστερά κατά οκτώ bit, γιατί η μέτρηση
                                       //έχει 16 bit
    I2C_ACK();                         //Αποστολή ACK
    RX_I2C_Data();                     //Παραλαβή δεύτερου byte
```

```
    temp |= byte;                       //Πρόσθεση του δεύτερου byte στα 8 τελευταία bit

    I2C_NAK();                          //Αποστολή NAK
    I2C_Stop_Bit();                     ////Αποστολή  αλληλουχίας τερματισμού


   return temp;

}

//Συνάρτηση επικοινωνίας με αισθητήρα για μέτρηση θερμοκρασίας με το πρωτόκολλο IIC
int getTempHDC(){

   int dc = 0;
   I2C_Start_Bit();                     //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x80);                //Αποστολή διεύθυνσης με R/W = 0
      if (!SSPCON2bits.ACKSTAT){        //Αναμονή παραλαβή ACK απο αισθητήρα
         __delay_ms(50);

      }

    Send_I2C_Data(0x02);                //Αποστόλη διεύθυνσης δείκτη 2
      if (!SSPCON2bits.ACKSTAT){        //Αναμονή παραλαβή ACK απο αισθητήρα

         __delay_ms(50);

      }

    Send_I2C_Data(0x06);                //Ορίζω το Configuration του αισθητήρα
       if (!SSPCON2bits.ACKSTAT){       //Αναμονή παραλαβή ACK απο αισθητήρα
         __delay_ms(50);

      }

    I2C_Stop_Bit();                     //Αποστολή  αλληλουχίας τερματισμού


   for(dc=0;dc<50;dc++){                //Αναμονή ολοκλήρωσης μετρήσεων του αισθητήρα
    __delay_ms(50);
   }

    I2C_Start_Bit();                    //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x80);                //Αποστολή διεύθυνσης με R/W = 0
    if (!SSPCON2bits.ACKSTAT){          //Αναμονή παραλαβή ACK απο αισθητήρα
      for(dc=0;dc<4;dc++){
         __delay_ms(50);
      }
   }
    Send_I2C_Data(0x00);                //Αποστόλη διεύθυνσης δείκτη 0
```

```
    I2C_Stop_Bit();                 //Αποστολή  αλληλουχίας τερματισμού

    for(dc=0;dc<50;dc++){           //Αναμονή ολοκλήρωσης μετρήσεων του αισθητήρα
     __delay_ms(50);
    }

    I2C_Start_Bit();                //Αποστολή  αλληλουχίας έναρξης
    Send_I2C_Data(0x81);            //Αποστολή διεύθυνσης με R/W = 1
    RX_I2C_Data();                  //Παραλαβή πρώτου byte
    temp = byte;
    temp = temp << 8;               //Ολίσθηση αριστερά κατά οκτώ bit, γιατί η μέτρηση
                                    //έχει 16 bit
    I2C_ACK();                      //Αναμονή παραλαβή ACK απο αισθητήρα
    RX_I2C_Data();                  //Παραλαβή δεύτερου byte
    temp |= byte;                   //Πρόσθεση του δεύτερου byte στα 8 τελευταία bit


    I2C_NAK();                      //Αποστολή NAK
    I2C_Stop_Bit();                 //Αποστολή  αλληλουχίας τερματισμού


    return temp;

}

void setTables(){
    MsgFromPIC2[0] = 'A';
    MsgFromPIC2[1] = 'T';
    MsgFromPIC2[2] = '+';
    MsgFromPIC2[3] = 'C';
    MsgFromPIC2[4] = 'M';
    MsgFromPIC2[5] = 'G';
    MsgFromPIC2[6] = 'F';
    MsgFromPIC2[7] = '=';
    MsgFromPIC2[8] = '1';
    MsgFromPIC2[9] = '\r';

    ATGSM[0] = 'A';
    ATGSM[1] = 'T';
    ATGSM[2] = '+';
    ATGSM[3] = 'C';
    ATGSM[4] = 'S';
    ATGSM[5] = 'C';
    ATGSM[6] = 'S';
    ATGSM[7] = '=';
    ATGSM[8] = '"';
    ATGSM[9] = 'G';
    ATGSM[10] = 'S';
    ATGSM[11] = 'M';
    ATGSM[12] = '"';
    ATGSM[13] = '\r';
```

```
    SmsBuffer[0] = 'A';
    SmsBuffer[1] = 'T';
    SmsBuffer[2] = '+';
    SmsBuffer[3] = 'C';
    SmsBuffer[4] = 'M';
    SmsBuffer[5] = 'G';
    SmsBuffer[6] = 'S';
    SmsBuffer[7] = '=';
    SmsBuffer[8] = '"';
    SmsBuffer[9] = '6';
    SmsBuffer[10] = '9';
    SmsBuffer[11] = '8';
    SmsBuffer[12] = '8';
    SmsBuffer[13] = '7';
    SmsBuffer[14] = '1';
    SmsBuffer[15] = '9';
    SmsBuffer[16] = '7';
    SmsBuffer[17] = '6';
    SmsBuffer[18] = '2';
    SmsBuffer[19] = '"';
    SmsBuffer[20] = '\r';



    MsgFromPIC[0] = 'A';
    MsgFromPIC[1] = 'T';
    MsgFromPIC[2] = '\r';

    for(int retc=0;retc<24;retc++){

        ret[retc]=' ';

    }
}

void CPStart(void){

    PORTA = 0xFF;                       //Το σήμα εκκίνησης είναι όλα τα ψηφία της
                                        //θύρας Α να υψωθούν στο λογικό ένα


}

int CPACK(){

    if(ACKBit == 1){                    //Το ACKBit είναι ένας ακροδέκτης εισόδου που
                                        //μεταβάλετε απο τον άλλο PIC

        return 1;

    }
    else{
```

```
        return 0;

    }

}

void CPSent(int number){

    LATA = number;                          //Το ψηφίο περνά στη θύρα ένα, και χρειάζεται
                                            //πάντα τέσσερις ακροδέκτες μιας και είναι
                                            //μικρότερο απο το νούμερο δεκαέξι (16)

}

int CPNACK(){

    if(ACKBit == 0){

        return 1;

    }
    else{

        return 0;

    }

}
```

ΠΑΡΑΡΤΗΜΑ Β
ΚΩΔΙΚΑΣ SLAVE

```
#define _XTAL_FREQ 4000000 //Ενημέρωση του Compiler σχετικά με τον χρονισμό.

#define CS1 LATD3           //Θέτω τον ακροδέκτη D3 σαν CS1 για έλεγχο του slave.

        //Οι παρακάτω ορισμοί χρησιμοποιούνται στη βιβλιοθήκη της LCD.
#define RS LATD1            // Θέτω τον ακροδέκτη RS σαν   LATD1.
#define EN LATD0            // Θέτω τον ακροδέκτη EN σαν   LATD0.
#define D4 LATD4            // Θέτω τον ακροδέκτη D4 σαν   LATD4.
#define D5 LATD5            // Θέτω τον ακροδέκτη D5 σαν   LATD5.
#define D6 LATD6            // Θέτω τον ακροδέκτη D6 σαν   LATD6.
#define D7 LATD7            // Θέτω τον ακροδέκτη D7 σαν   LATD7.
#define ACKBit RD2 // Θέτω τον ακροδέκτη RD2 σαν  ACKBit, χρησιμοποιείται στη
                        //μεταφορά δεδομένων απο master σε slave.

#include <xc.h>             //Η βασική βιβλιοθήκη που περιέχει ορισμούς για το PIC.
#include "lcd.h"            //Η βιβλιοθήκη που περιέχει εντολές της LCD.
#include "stdio.h"          //Βασική βιβλιοθήκη με συναρτήσεις εισόδου εξόδου
 //Παρακάτω ορίζεται πως θα χρησιμοποιηθούν διάφορα περιφερειακά και χρονιστές αλλά και
//ακροδέκτες του PIC
#pragma config CPUDIV = OSC1_PLL2
#pragma config PLLDIV = 1
#pragma config USBDIV = 1
#pragma config FCMEN  = OFF
#pragma config IESO  = OFF
#pragma config FOSC  = INTOSCIO_EC
#pragma config PWRT  = OFF
#pragma config VREGEN  = OFF
#pragma config BORV  = 0
#pragma config BOR = OFF
#pragma config WDTPS = 16384
#pragma config WDT = OFF
#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config MCLRE = ON
#pragma config LPT1OSC = ON
#pragma config STVREN = OFF
#pragma config DEBUG = OFF
#pragma config ICPRT = OFF
#pragma config LVP = OFF
#pragma config XINST = OFF
#pragma config CP0  = OFF
#pragma config CP1  = OFF
#pragma config CP2  = OFF
#pragma config CP3  = OFF
```

```
void initMCU(void);             //Συνάρτηση αρχικοποίησης του μικροελεγκτή



                                // Οι παρακάτω μεταβλητές και συναρτήσεις χρησιμοποιούνται για
                                //την εφαρμογή του πρωτοκόλλου SPI.
unsigned char SPI(unsigned char data);
char Command(unsigned char frame1, unsigned long adrs, unsigned char frame2 );
void InitSD(void);
void WriteSD(void);
long ReadSD(void);
void initMCU(void);
void WriteSectors(long sector, char *buff, int sec_count);
char in ;
char q[15];
char low[3];

                                //Συναρτήσεις για επικοινωνία Pic με Pic χρησιμοποιώντας
                                //ακροδέκτες γενικού σκοπού
int CPCStartSeq(void);
void CPSACK(void);
int CPCDigit(void);
int CPGetDigit(void);
void CPSNACK(void);

void main(void){

   initMCU();                   // Αρχικοποίηση μικροελεγκτή
   __delay_ms(50);
   __delay_ms(50);
   __delay_ms(50);
        InitSD();               // Αρχικοποίηση κάρτας SD
   __delay_ms(50);
   __delay_ms(50);
   __delay_ms(50);
   Lcd_Init();                  // Αρχικοποίηση οθόνης LCD
   __delay_ms(50);
   __delay_ms(50);
   __delay_ms(50);

   Lcd_Clear();                 //Καθαρισμός LCD
   Lcd_Set_Cursor(1,1);         //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD
   Lcd_Write_String("anamoni");
   Lcd_Set_Cursor(2,1);         //Τοποθέτηση κέρσορα στο σημείο 2,1 της LCD
   Lcd_Write_String("metrisewn");


   int dcounter;
   int nums[4];
```

```
                                    //Επανάληψη για παραλαβή των τεσσάρων ψηφίων απο το Master.
for(dcounter=0;dcounter<4;dcounter++){

   do{
   }while(!CPCStartSeq()); //Αναμονή αλληλουχίας εκκίνησης απο Master.

   CPSACK(); //Αποστολή ACK απο slave.

   do{
   }while(!CPCDigit()); //Ελεγχος αλλαγής των εισόδων της θύρας Α.


   nums[dcounter] = CPGetDigit(); //Αποδοχή του ψηφίου και εισαγωγή στο πίνακα με τα
                                  //ψηφία της μέτρησης.

   CPSNACK();  //Αποστολή NACK απο slave για τερματισμό λήψης.

}

              //Μετατροπή των δύο ψηφίων ανα μέτρηση σε ένα νούμερο.
int temp = (nums[0]*10)+nums[1];
int humid = (nums[2]*10)+nums[3];
char tempstr[8];
char humstr[8];
sprintf(tempstr, "T=%dC", temp); //Μετατροπή απο String σε int.
sprintf(humstr, "H=%d", humid);  //Μετατροπή απο String σε int.

Lcd_Clear();                     //Καθαρισμός LCD
Lcd_Set_Cursor(1,1);             //Τοποθέτηση κέρσορα στο σημείο 1,1 της LCD.
Lcd_Write_String(tempstr);       //Εμφάνιση της θερμοκρασίας στην LCD.
Lcd_Set_Cursor(1,8);             //Τοποθέτηση κέρσορα στο σημείο 1,8 της LCD.
Lcd_Write_String(humstr);        //Εμφάνιση της υγρασίας στην LCD.

                    //Διαδικασία μεταφοράς μετρήσεων στη κάρτα SD.
__delay_ms(50);
__delay_ms(50);
__delay_ms(50);
long sector;
sector = ReadSD();               //Εύρεση του επόμενου Sector για εγγραφή.

int y;
char buffer[200];                //Εγγραφή του String με τις μετρήσεις σε πίνακα String
buffer[0] = 'T';
buffer[1] = 'e';
buffer[2] = 'm';
buffer[3] = 'p';
buffer[4] = '=';
buffer[5] = tempstr[0];
```

```
    buffer[6] = tempstr[1];
    buffer[7] = 'C';
    buffer[8] = ' ';
    buffer[9] = 'H';
    buffer[10] = 'u';
    buffer[11] = 'm';
    buffer[12] = '=';
    buffer[13] = humstr[0];
    buffer[14] = humstr[1];
    buffer[15] = ' ';
                                    //Συμπλήρωση των κενών θέσεων του πίνακα με το γράμμα 'p'.
    for(y=16;y<200;y++){

       buffer[y] = 'p';

    }
                            //Εγγραφή των μετρήσεων στη κάρτα, στον Sector sector.
    WriteSectors(sector,buffer);
    sector++;               //Αύξηση του δείκτη sector κατά ένα
    char a,b;
    a= sector >> 8;         //Ολίσθηση του sector κατά οκτώ bit
    b = sector & 0x000000FF; //Παίρνω τα τέσσερα οκτώ τελευταία bit μηδενίζοντας τα υπόλοιπα.

    buffer[0] = a;
    buffer[1] = b;

    WriteSectors(9,buffer);     //Εγγραφή στον Sector εννέα τον pointer που είναι τα δύο πρώτα
                                //bytes.

    Lcd_Set_Cursor(2,1);        //Τοποθέτηση κέρσορα στο σημείο 2,1 της LCD.
    Lcd_Write_String("Egrafi telos");

          while(1) {

          }
}
          //Συνάρτηση αρχικοποίησης
void initMCU(void){
                                //Εγγραφή στον καταχωρητή IRCF τα bit 110,για χρονισμό 4mhz
    OSCCONbits.IRCF2 = 1;
    OSCCONbits.IRCF1 = 1;
    OSCCONbits.IRCF0 = 0;

    SSPCON1bits.CKP = 0; //Θέτω σαν ανενεργή τάση παλμού το χαμηλό παλμό.
    SSPSTATbits.CKE = 1; //Ορίζω τη μετάδοση να γίνεται στην αλλαγή του παλμού απο ενεργή σε
                         //ανενεργή.
    SSPSTATbits.SMP = 1;  //Θέτω να διαβάζει τα δεδομένα στο τέλος του παλμού.
    SSPCON1bits.SSPM = 0b0010; //Θέτω τη θύρα MSSP σε SPI mode με clock = FOSC/64.

    TRISD = 0x00;       //Θύρα D έξοδος για επικοινωνία με την lcd
    TRISBbits.TRISB1 = 0; //Ακροδέκτης B1 έξοδος, για SCKL του SPI.
```

```
    TRISBbits.TRISB0 = 1; //Ακροδέκτης B0 είσοδος, για SDI του SPI.
    TRISCbits.TRISC2 = 0; //Ακροδέκτης C2 έξοδος, για CS του SPI.
    TRISCbits.TRISC7 = 0; //Ακροδέκτης C7 έξοδος, για SDO του SPI
    SSPCON1bits.SSPEN = 1;          //Ενεργοποίηση του SPI.

    ADCON1 = 0x0F;          //Μετατροπή των ακροδεκτών της θύρας A απο αναλογικά σε ψηφιακά

    TRISA = 0xFF; //Θύρα A έξοδος για επικοινωνία με master
    ACKBit = 0;
    CS1 = 1;

}
                //Συνάρτηση αποστολής ενός χαρακτήρα μέσω SPI.
unsigned char SPI(unsigned char data)
{
        SSPBUF = data;              //Η αποστολή γίνεται αυτόματα εάν γεμίσει ο buffer SSPBUF.
        while (!BF);            //Αναμονή ολοκλήρωση αποστολής.
        return SSPBUF;              //Επιστροφή χαρακτήρα παραλαβής.
}
                //Συνάρτηση αποστολής εντολής μέσω SPI
char Command(unsigned char frame1, unsigned long adrs, unsigned char frame2 )
{
        unsigned char i, res;
        SPI(0xFF);
        SPI((frame1 | 0x40) & 0x7F);        //Μία έγκυρη εντολή πρέπει τα δύο πρώτα της bits να
                                            //είναι 01, εδώ εξασφαλίζεται αυτό.
        SPI((adrs & 0xFF000000) >> 24);     //Αποστέλλεται το πρώτο byte.
        SPI((adrs & 0x00FF0000) >> 16);     //Αποστέλλεται το δεύτερο byte.
        SPI((adrs & 0x0000FF00) >> 8);      //Αποστέλλεται το τρίτο byte.
        SPI(adrs & 0x000000FF);             //Αποστέλλεται το τελευταίο byte.
        SPI(frame2 | 1);                    //Αποστέλλεται το CRC και εξασφαλίζεται πως το
                                            //τελευταιό  bit είναι 1.

        for(i=0;i<10;i++)       //Αναμονή απάντησης, αποστέλλεται dummy byte γιατί οι παλμοί του
                                //ρολογιού μεταδίδονται μόνο όταν ο master στέλνει δεδομένα.
        {
                res = SPI(0xFF);
                if(res != 0xFF)break;
        }
        return res;             //Επιστροφή απάντησης.
}

//Συνάρτηση αρχικοποίησης κάρτας SD.
void InitSD(void)
{
        unsigned char i,r[4];

        CS1=1;
        for(i=0; i < 10; i++)SPI(0xFF);             //Η κάρτα απαιτεί ζέσταμα με τουλάχιστον 100
                                                    //παλμούς ρολογιού, στέλνω 80.
        CS1=0;
```

```
        i=100; //Θέτω σαν μέγιστο αριθμό προσπαθιών το 100
        while(Command(0x00,0,0x95) !=1 && i!=0)
        {
                CS1=1;
                SPI(0xFF);
                CS1=0;
                i--;
        }

        if (Command(8,0x01AA,0x87)==1){ //Αποστολή CMD8 με όρισμα 0x01AA και CRC 0x87
                                        //για έλεγχο της τάσης λειτουργίας της κάρτας


                r[0]=SPI(0xFF); r[1]=SPI(0xFF); r[2]=SPI(0xFF); r[3]=SPI(0xFF);//Dummy bytes
                                        //για να πάρω την απάντηση.
                if ( r[2] == 0x01 && r[3] == 0xAA ){  //Αποδοχή τάσης.


                        Command(55,0,0xFF); //Η CMD55 δηλώνει οτι θα σταλεί ACMD.
                        while(Command(41,0x40000000,0xFF)){//ACMD41

                                        Command(55,0,0xFF);
                                }
                }
        }

        if (Command(58,0,0xFF)==0){//Η CMD58 επιστρέφει τον CCS του OCR
                r[0]=SPI(0xFF); r[1]=SPI(0xFF); r[2]=SPI(0xFF); r[3]=SPI(0xFF);
                //Η τιμή βρίσκεται στο r[0].
        }

        CS1 = 1;
}


//Συνάρτηση η οποία διαβάζει τον Pointer sector απο τον Sector 9 της κάρτας.
long ReadSD(void)
{
        unsigned int i,r;
        long sum =0,sum1=0,sum2=0;
        unsigned char data;
        CS1 = 0;
        r = Command(18,9,0xFF);    //Η CMD18 με όρισμα 9, σηματοδοτεί διάβασμα του Sector 9.


                while(SPI(0xFF) != 0xFE);   //Αποστολή dummy bytes έως ότου η κάρτα στείλει
                                            //δεδομένα

      sum1 = SPI(0xFF);       //Το πρώτο byte του δείκτη sector
      sum2 = SPI(0xFF);       //Το δεύτερο byte του δείκτη sector
```

```
    for(i=2;i<512;i++){       //Διάβασμα του υπόλοιπου Sector

       SPI(0xFF);

                }
                SPI(0xFF);
                SPI(0xFF);



  sum = sum1 << 8; //Ολίσθηση του πρώτου byte κατά ένα byte σημαντικότερο.
  sum+=sum2;          //Πρόσθεση του λιγότερου σημαντικού byte.
       Command(12,0x00,0xFF);//Η CMD12 σηματοδοτεί το τέλος διαβάσματος απο την κάρτα.
       SPI(0xFF);
       SPI(0xFF);
       CS1 = 1;
  return sum;          //Επιστροφή του δείκτη sector.
}
//Συνάρτηση η οποία γράφει ένα Sector της κάρτας μνήμης.
void WriteSectors(long sector, char *buff)
{
       unsigned int r,i;
       CS1 = 0;

       r = Command(25,sector,0xFF);//Η CMD25 με όρισμα τον αριθμό του Sector, σηματοδοτεί
                                   //εγγραφή σε αυτόν.
       SPI(0xFF);          //Μερικά dummy bytes.
       SPI(0xFF);
       SPI(0xFF);
            SPI(0xFC);     //Σύμβολο έναρξης εγγραφής.
            for(i=0;i<200;i++)     //Εγγραφή του buffer.
            {

       SPI(buff[i]);

            }
    for(i=0;i<312;i++)        //Συμπλήρωση του Sector με το γράμμα “p”
            {

       SPI('p');

            }
            SPI(0xFF);     //Αποστολή CRC.
            SPI(0xFF);

    if((r=SPI(0xFF) & 0x0F) == 0x05){    //Εάν τα τέσσερα λιγότερο σημαντικά bits είναι 0101,
                                         //σημαίνει οτι τα δεδομένα γίνανε δεκτά απο την κάρτα.
       for(i=10000;i>0;i--){             //Αναμονή εγγραφής
          if(r=SPI(0xFF))     break;
       }
    }
    else{
```

```
        }
        while(SPI(0xFF) != 0xFF);                   //Αναμονή ετοιμασίας κάρτας.



          SPI(0xFD);     //Σύμβολο λήξης εγγραφής.

          SPI(0xFF);
          SPI(0xFF);
          while(SPI(0xFF) != 0xFF);    //Αναμονή ετοιμασίας κάρτας.

          CS1 = 1;
}

//Συνάρτηση ελέγχου για αλληλουχία έναρξης απο Master, που είναι 1111 στη θύρα Α.
int CPCStartSeq(){

   if(RA0 == 1){
      if(RA1 == 1){
         if(RA2 == 1){
            if(RA3 == 1){
               return 1;
            }
         }
      }
   }

   return 0;

}

void CPSACK(){

   ACKBit = 1;

}
//Συνάρτηση ελέγχου για αλλαγή απο της αλληλουχίας έναρξης, που σημαίνει οτι η θύρα Α έχει ένα
//ψηφίο για παραλαβή
int CPCDigit(){

   if(RA0 == 0){
      return 1;
   }

   if(RA1 == 0){
      return 1;
   }

   if(RA2 == 0){
      return 1;
   }
```

```
    if(RA3 == 0){
        return 1;
    }

    return 0;

}
//Συνάρτηση παραλαβής ψηφίου.
int CPGetDigit(){
    int sum = 0;

    if(RA0 == 1){
        sum =sum+1;
    }

    if(RA1 == 1){
        sum =sum+2;
    }

    if(RA2 == 1){
        sum =sum+4;
    }

    if(RA3 == 1){
        sum =sum+8;
    }

    return sum;

}

void CPSNACK(){

    ACKBit = 0;

}
```

ΒΙΒΛΙΟΓΡΑΦΙΑ

Πτυχιακή εργασία με θέμα : Μελέτη των περιφερειακών κυκλωμάτων του αναπτυξιακού συστήματος EASYPIC 5 και προγραμματισμός με χρήση του μεταγλωττιστή C. Ευάγγελος Σπαθάρας. Επιβλέπων : Ιωάννη Καλόμοιρος, Αναπληρωτής Καθηγητής.

Σύγχρονα Λειτουργικά Συστήματα, Τρίτη αμερικάνική έκδοση. Andrew S. Tanenbaum.

http://www.engineersgarage.com/

http://www.microchip.com/forums/

http://www.i2c-bus.org/

http://elm-chan.org

https://www.sdcard.org/

http://www.dejazzer.com/

https://en.wikipedia.org/

http://web.alfredstate.edu/weimandn/lcd/lcd_initialization/lcd_initialization_index.html

https://electrosome.com