για αρχάριους

με έμφαση στη χρήση του στο εργαστήριο Φυσικών Επιστημών

Νούσης Βασίλης, Ηγουμενίτσα 2019

ARDUINO

για αρχάριους

με έμφαση στη χρήση του στο εργαστήριο Φυσικών Επιστημών

Νούσης Βασίλης

ISBN : 978-618-83502-3-6

Πίνακας περιεχομένων

Πρόλογος

Το βιβλίο που κρατάτε στα χέρια σας δε φιλοδοξεί να αποτελέσει ένα πλήρη οδηγό προγραμματισμού και χρήσης της μικροϋπολογιστικής πλατφόρμας Arduino. Άλλωστε είμαι μόνο ένας αυτοδίδακτος χρήστης της πλατφόρμας, και συνεπώς το βιβλίο, που είναι αποτέλεσμα της προσωπικής μου ενασχόλησης με τον Arduino με στόχο την αξιοποίησή του στο εργαστήριο Φυσικών Επιστημών, απευθύνεται πρωτίστως στον αρχάριο χρήστη.

Τρεις ενότητες συναποτελούν το βιβλίο:

1. Στην εισαγωγική ενότητα σύντομα παρουσιάζονται τα χαρακτηριστικά (υλικό, λογισμικό και κοινότητα χρηστών) που αποτελούν τη βάση της επιτυχίας της πλατφόρμας.
2. Στη δεύτερη ενότητα αναπτύσσονται τα βασικά στοιχεία που αφορούν στον προγραμματισμό του Arduino, και παρουσιάζεται ένας σημαντικός αριθμός αισθητήρων και άλλων διατάξεων που μπορούν να συνδεθούν στον Arduino.
3. Στην τρίτη ενότητα παρουσιάζονται μερικά πιο απαιτητικά projects που συνδυάζουν αρκετά από τα χαρακτηριστικά που παρουσιάστηκαν στην προηγούμενη ενότητα.

Συνολικά 37 sketches (προγράμματα) παρουσιάζονται στο βιβλίο, τα οποία ως ένα συμπιεσμένο αρχείο τύπου "rar" είναι διαθέσιμα προς "μεταφόρτωση" στον υπολογιστή σας μέσω του συνδέσμου: https://drive.google.com/file/d/11G-NhAVXfLMoZvyMmLW-7qSVBh4yl2QO/view ή μέσω του επόμενου QR Code:

Κάθε παράδειγμα του βιβλίου συνοδεύεται και από το σχετικό κύκλωμα με τις απαραίτητες συνδέσεις, σχεδιασμένο με τη βοήθεια του εξαιρετικού ελεύθερου λογισμικού Fritzing.

Ελπίζω το βιβλίο να φανεί χρήσιμο στον αρχάριο που ενδιαφέρεται να ανακαλύψει τις δυνατότητες της πλατφόρμας Arduino.

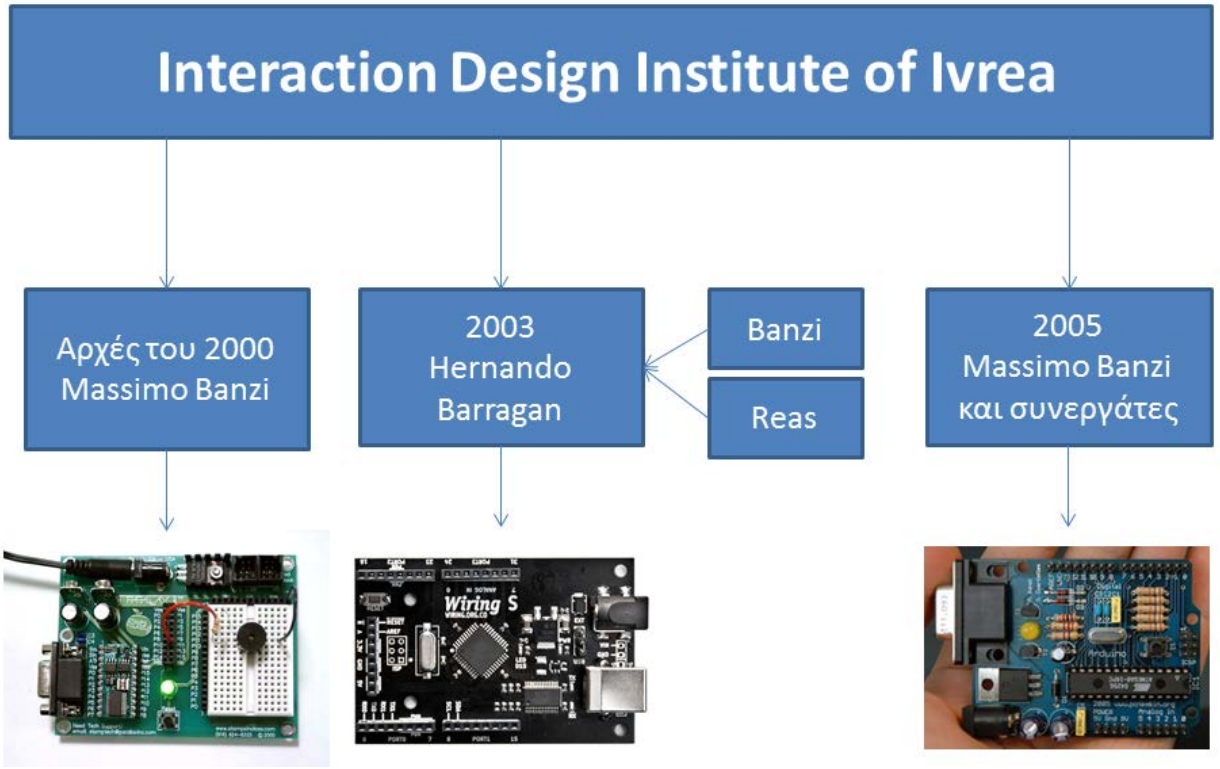
Ηγουμενίτσα 1/3/2019

Βασίλης Νούσης

Arduino

Εισαγωγή

Η πλατφόρμα Arduino είναι ένα ανοιχτού υλικού και ανοιχτού κώδικα σύστημα ανάπτυξης ηλεκτρονικών πρωτοτύπων, βασισμένο σε ευέλικτο και εύκολο στη χρήση υλικό και λογισμικό.



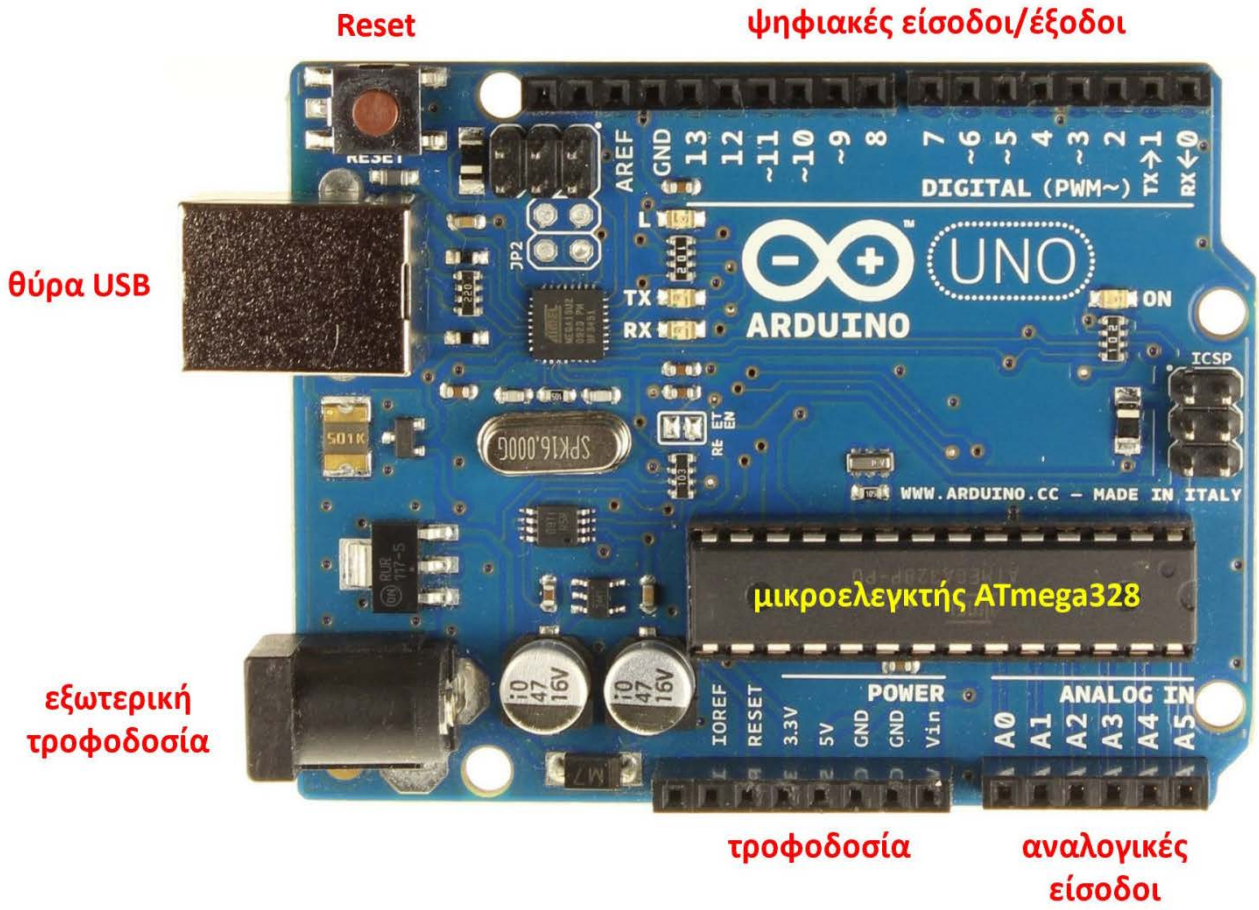
Εικόνα 1: Arduino - Ιστορική εξέλιξη

Προέκυψε ως το αποτέλεσμα της ανάγκης για ένα φθηνότερο, ισχυρότερο, περισσότερο ευέλικτο και εύκολο στη χρήση αναπτυξιακό σύστημα, σε σχέση με το Basic Stamp της Parallax που στις αρχές του 2000 χρησιμοποιούσαν οι φοιτητές του Massimo Banzi στο Interaction Design Institute (IDI) στην Ivrea της Ιταλίας. Το 2003 ο Hernando Barragan στα πλαίσια της μεταπτυχιακής διπλωματικής του εργασίας στο IDI, υπό την επίβλεψη του Massimo Banzi και του Ceasey Reas, ανέπτυξε το σύστημα Wiring, ένα συνδυασμό μικροϋπολογιστικού συστήματος και κατάλληλου προγραμματιστικού περιβάλλοντος βασισμένου στη γλώσσα προγραμματισμού Processing που λίγα χρόνια νωρίτερα είχαν παρουσιάσει ο Ben Fry και ο Ceasey Reas [1]. Το 2005 ο Massimo Banzi με μια ομάδα συνεργατών (στην οποία δε συμμετείχε ο Barragan) με βάση το λογισμικό του Wiring παρουσίασαν ένα σημαντικά φθηνότερο μικροϋπολογιστικό σύστημα με το όνομα "Arduino", δανεισμένο από το όνομα ενός μπαρ που υπήρξε το στέκι της ομάδας του Banzi εκείνη την εποχή [2].

Πολλές διαφορετικές εκδόσεις του Arduino έχουν παρουσιαστεί μέχρι σήμερα με πιο επιτυχημένη την έκδοση Arduino Uno R3.

Το υλικό του Arduino

Το υλικό του Arduino (μοντέλο Uno R3 και αντίστοιχοι κλώνοι, στο οποίο θα αναφερόμαστε εν συνεχεία) συναρμολογείται σε μια μικρή πλακέτα διαστάσεων 68mm x 53mm.



Εικόνα 2: Η πλακέτα του Arduino Uno R3

Στην καρδιά του συστήματος βρίσκεται ο οκτάμπιτος μικροελεγκτής ATmega 328 της Atmel χρονισμένος στα 16MHz, που διαθέτει ενσωματωμένη μνήμη τριών τύπων:

1. 2 kB στατικής μνήμης RAM στην οποία τα προγράμματα κατά την εκτέλεσή τους αποθηκεύουν μεταβλητές, πίνακες κ.λπ. Η μνήμη αυτή χάνει τα δεδομένα της όταν η παροχή ρεύματος σταματήσει ή αν γίνει επανεκκίνηση (reset) του συστήματος.

2. 1 kB μνήμης EEPROM στην οποία κατά την εκτέλεση των προγραμμάτων μπορούν να εγγραφούν/διαβαστούν δεδομένα byte προς byte. Η EEPROM δεν χάνει τα περιεχόμενά της με απώλεια τροφοδοσίας ή επανεκκίνηση του συστήματος.

3. 32 kB μνήμης Flash, από τα οποία τα 0,5 kB χρησιμοποιούνται από τον bootloader του Arduino, το λογισμικό δηλ. εκείνο που είναι απαραίτητο για την εγκατάσταση των δικών μας προγραμμάτων στον μικροελεγκτή μέσω της θύρας USB, χωρίς να χρειάζεται εξωτερικός προγραμματιστής. Το υπόλοιπο της μνήμης Flash χρησιμοποιείται για την αποθήκευση αυτών ακριβώς των προγραμμάτων. Και η μνήμη Flash δε χάνει τα περιεχόμενά της με απώλεια τροφοδοσίας ή επανεκκίνηση [3].

Στο υλικό του μικροελεγκτή ATmega 328 συμπεριλαμβάνονται επίσης:

- 32 καταχωρητές γενικής χρήσης.
- 23 προγραμματιζόμενες γραμμές εισόδου/εξόδου (I/O lines).

- Μια μονάδα USART (Universal Synchronous/Asynchronous Receiver Transmitter) μέσω της οποίας επιτυγχάνεται η σειριακή επικοινωνία με διάφορες συσκευές.
- Δύο μονάδες για τη σειριακή επικοινωνία μέσω των διαύλων SPI και I^2C.
- Μία εξακάναλη μονάδα μετατροπέα αναλογικού σε ψηφιακό (A/D C), διακριτικής ικανότητας 10bit.
- Τρεις χρονιστές/απαριθμητές: timer0 και timer2 διακριτικής ικανότητας 8bit και ο timer1 των 16bit. Το λογισμικό του Arduino εκμεταλλεύεται τους χρονιστές για τη χρονομέτρηση ή την εισαγωγή καθυστέρησης κατά την εκτέλεση των εντολών κάποιου προγράμματος, την εκτέλεση συγκεκριμένων εντολών σε καθορισμένες χρονικές στιγμές ή ανά τακτά χρονικά διαστήματα, κ.ά.

Στη μια πλευρά της πλακέτας του Arduino (Uno R3) και σε αντίστοιχες θηλυκές ακίδες σύνδεσης καταλήγουν 14 από τις ψηφιακές γραμμές εισόδου εξόδου (I/O lines) του μικροελεγκτή ATMega 328, οι οποίες λειτουργούν με τάση 5V και μπορούν να δώσουν ή να «αντλήσουν» μέχρι 40mA ηλεκτρικού ρεύματος. Μπορούν για παράδειγμα ως έξοδοι να χρησιμοποιηθούν για να ανάψουν ή να σβήσουν ένα LED ή ως είσοδοι να καταγράψουν την κατάσταση ενός διακόπτη. Όλες οι ψηφιακές είσοδοι/έξοδοι έχουν εσωτερικές pull-up (πρόσδεσης στην τροφοδοσία) αντιστάσεις των 20-50 kΩ οι οποίες εξ ορισμού είναι απενεργοποιημένες [3]. Μερικές από αυτές τις ψηφιακές ακίδες (που αριθμούνται από 0 έως 13), έχουν και ειδικές χρήσεις:

- Οι ακίδες 0 και 1 χρησιμοποιούνται για τη λήψη και μετάδοση σειριακών (ένα bit τη φορά) δεδομένων.
- Οι ακίδες 3, 5, 6, 9, 10, 11 λειτουργούν ως «ψευδοαναλογικές» έξοδοι (PWM) των 8-bit.
- Οι ακίδες 2 και 3 μπορούν να ρυθμιστούν ώστε να προκαλέσουν εξωτερική διακοπή (interrupt) του εκτελούμενου κώδικα στον μικροελεγκτή.
- Οι ακίδες 10, 11, 12 και 13 υποστηρίζουν τη σειριακή επικοινωνία SPI.

Στην ίδια πλευρά της πλακέτας υπάρχει μία ακίδα γείωσης (GND) και επιπλέον οι ακίδες:

- **AREF** μέσω της οποίας υπάρχει η δυνατότητας παροχής εξωτερικής τάσης αναφοράς για τη λειτουργία του αναλογικο-ψηφιακού μετατροπέα (A/D C) του μικροελεγκτή.
- **SCL** και **SDA** που με το κατάλληλο λογισμικό χρησιμοποιούνται για την υλοποίηση του δισύρματου σειριακού πρωτοκόλλου επικοινωνίας I^2C.

Στην άλλη πλευρά της πλακέτας, στις ακίδες που σημαίνονται από A0 - A5, καταλήγουν οι 6 αναλογικές είσοδοι του Atmega328 οι οποίες μέσω πολυπλέκτη συνδέονται στον εντός του μικροελεγκτή ευρισκόμενο μετατροπέα αναλογικού σε ψηφιακό, που έχει διακριτική ικανότητα 10bit δηλ. ψηφιοποιεί μια αναλογική τάση 0-5V σε 1024 διαφορετικά βήματα. Μεγάλος αριθμός διαφορετικών αισθητήρων (π.χ. θερμοκρασίας, πίεσης, δύναμης, μαγνητικού πεδίου, φωτοπύλες κ.ά.) μπορεί να συνδεθεί στις εισόδους αυτές δίνοντάς μας τη δυνατότητα να πραγματοποιήσουμε μετρήσεις των αντίστοιχων φυσικών μεγεθών. Και οι έξι αναλογικές είσοδοι μπορούν να χρησιμοποιηθούν ως ψηφιακές είσοδοι/έξοδοι (ψηφιακές ακίδες 14 μέχρι 19), ενώ οι αναλογικές είσοδοι A4 και A5 είναι εσωτερικά συνδεδεμένες με τις ακίδες SCL και SDA του δισύρματου σειριακού πρωτοκόλλου επικοινωνίας I^2C.

Στην ίδια πλευρά της πλακέτας υπάρχει μια σειρά ακίδων σχετικών με την τροφοδοσία του συστήματος, ως εξής:

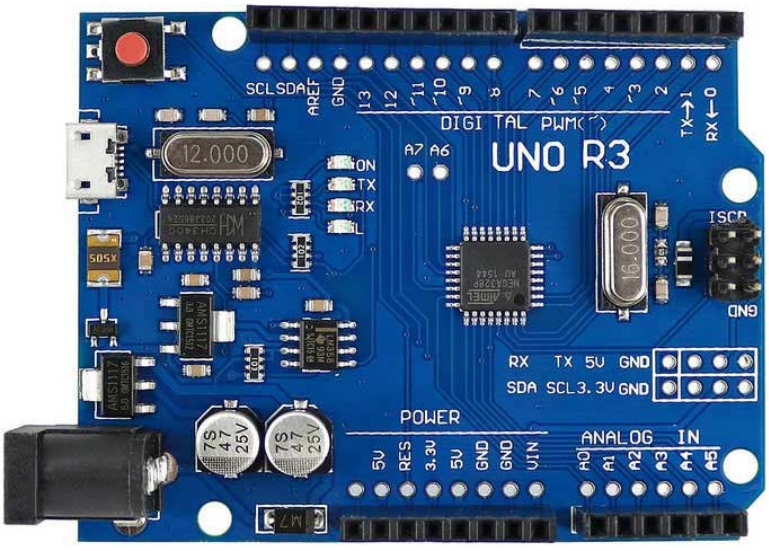
- **Vin**: Η τάση τροφοδοσίας του συστήματος όταν χρησιμοποιείται εξωτερική πηγή. Μπορεί να χρησιμοποιηθεί και ως είσοδος τροφοδοσίας του συστήματος.
- **5V**: Έξοδος σταθεροποιημένης τάσης 5V. Χρησιμεύει για την τροφοδοσία συσκευών ή αισθητήρων που πρόκειται να συνδεθούν στον Arduino.

- **3.3V**: Έξοδος σταθεροποιημένης τάσης 3,3V.
- **GND**: Γείωση.
- **Reset**: Ακίδα επανεκκίνησης. Το σύστημα επανεκκινεί όταν η ακίδα γειωθεί.
- **IOREF**: Παρέχει ένδειξη για την τάση αναφοράς με την οποία λειτουργεί ο μικροελεγκτής.

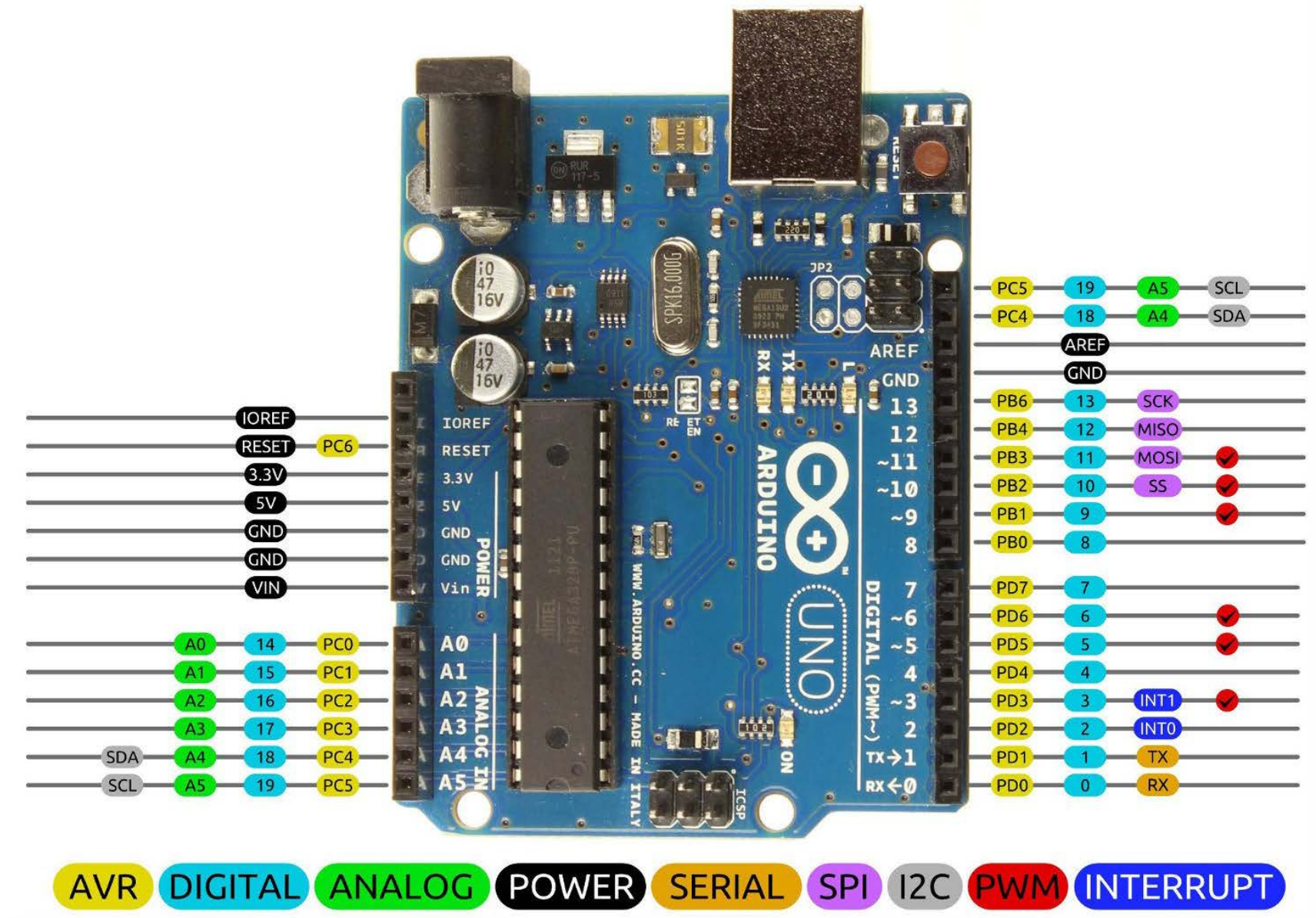
Η τροφοδοσία του Arduino μπορεί να γίνει είτε μέσω της θύρας USB είτε από εξωτερική τάση (7-12V). Η πηγή τροφοδοσίας επιλέγεται αυτόματα.

Στην πλακέτα του Arduino, υπάρχουν επίσης:

- Το κύκλωμα χρονισμού του μικροελεγκτή, αποτελούμενο είτε από ένα κρύσταλλο 16MHz, είτε από αντίστοιχης συχνότητας κεραμικό ταλαντωτή, και τα κατάλληλα παθητικά εξαρτήματα.
- Μια διπλή σειρά ακίδων ICSP (In Circuit Serial Programming) για τον προγραμματισμό του μικροελεγκτή μέσω εξωτερικού προγραμματιστή.
- Τα κατάλληλα ηλεκτρονικά εξαρτήματα για την παροχή σταθερής τάσης τροφοδοσίας στο κύκλωμα.
- Βύσμα τύπου jack για την τροφοδοσία του Arduino από εξωτερική πηγή.
- Βύσμα USB για τη σειριακή σύνδεση του Arduino με υπολογιστή.
- Ένας δεύτερος μικροελεγκτής (ATmega 16U2) με τον κρύσταλλο χρονισμού του, που κατάλληλα προγραμματισμένος λειτουργεί ως μετατροπέας σειριακού σε USB για την επικοινωνία του Arduino με ηλεκτρονικό υπολογιστή. Ουσιαστικά με τον τρόπο αυτό ο Arduino αναγνωρίζεται από τον ηλεκτρονικό υπολογιστή ως μια σειριακή θύρα (π.χ. COM5, κλπ). Για τον προγραμματισμό του Atmega 16U2 στην πλακέτα του αυθεντικού Arduino Uno R3 συμπεριλαμβάνεται και μία ακόμη διπλή ακιδοσειρά ICSP, που βρίσκεται δίπλα από τον ATmega 16U2. Σε νεότερους Arduino-κλώνους συναντάμε ένα αρκετά φθηνότερο και εξειδικευμένο ολοκληρωμένο κύκλωμα (CH 340), που έχει αναλάβει τη σειριακή επικοινωνία Arduino-υπολογιστή, μέσω της θύρας USB.
- Διακόπτης επανεκκίνησης (reset).
- 4 ενδεικτικά LED (Power, Tx, Rx και L συνδεδεμένο στην ψηφιακή έξοδο 13) καθώς και μικρός ακόμη αριθμός συνοδευτικών ηλεκτρονικών εξαρτημάτων.

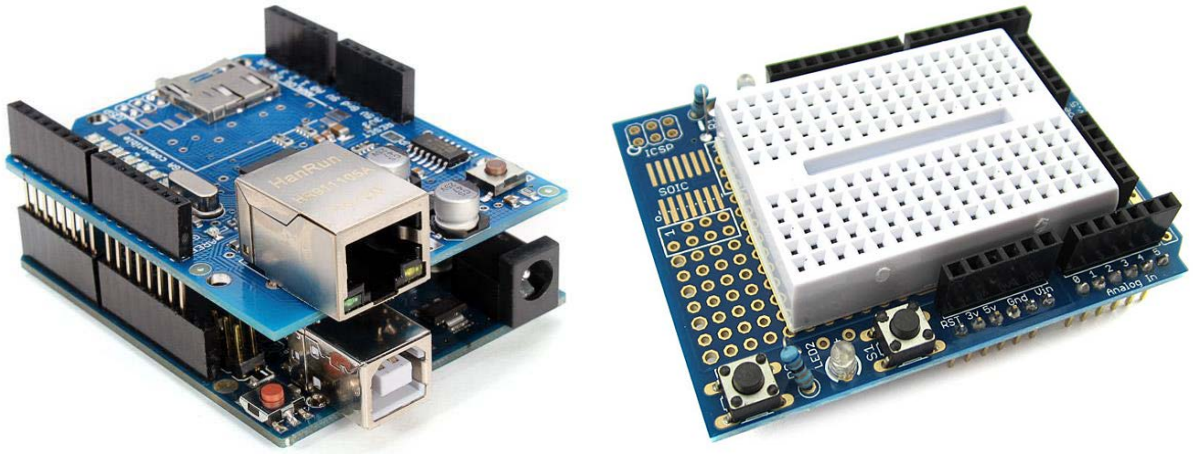


Εικόνα 3: Arduino Uno R3 κλώνος



Εικόνα 4 : Οι ακίδες του Arduino Uno R3

Σε επίπεδο υλικού πρέπει να αναφέρουμε πως έχει αναπτυχθεί μεγάλος αριθμός πλακετών (shields) οι οποίες «κουμπώνουν» κατευθείαν πάνω στην πλακέτα του Arduino αυξάνοντας τις δυνατότητες και τη λειτουργικότητά του συστήματος.



Εικόνα 5: Arduino Shields

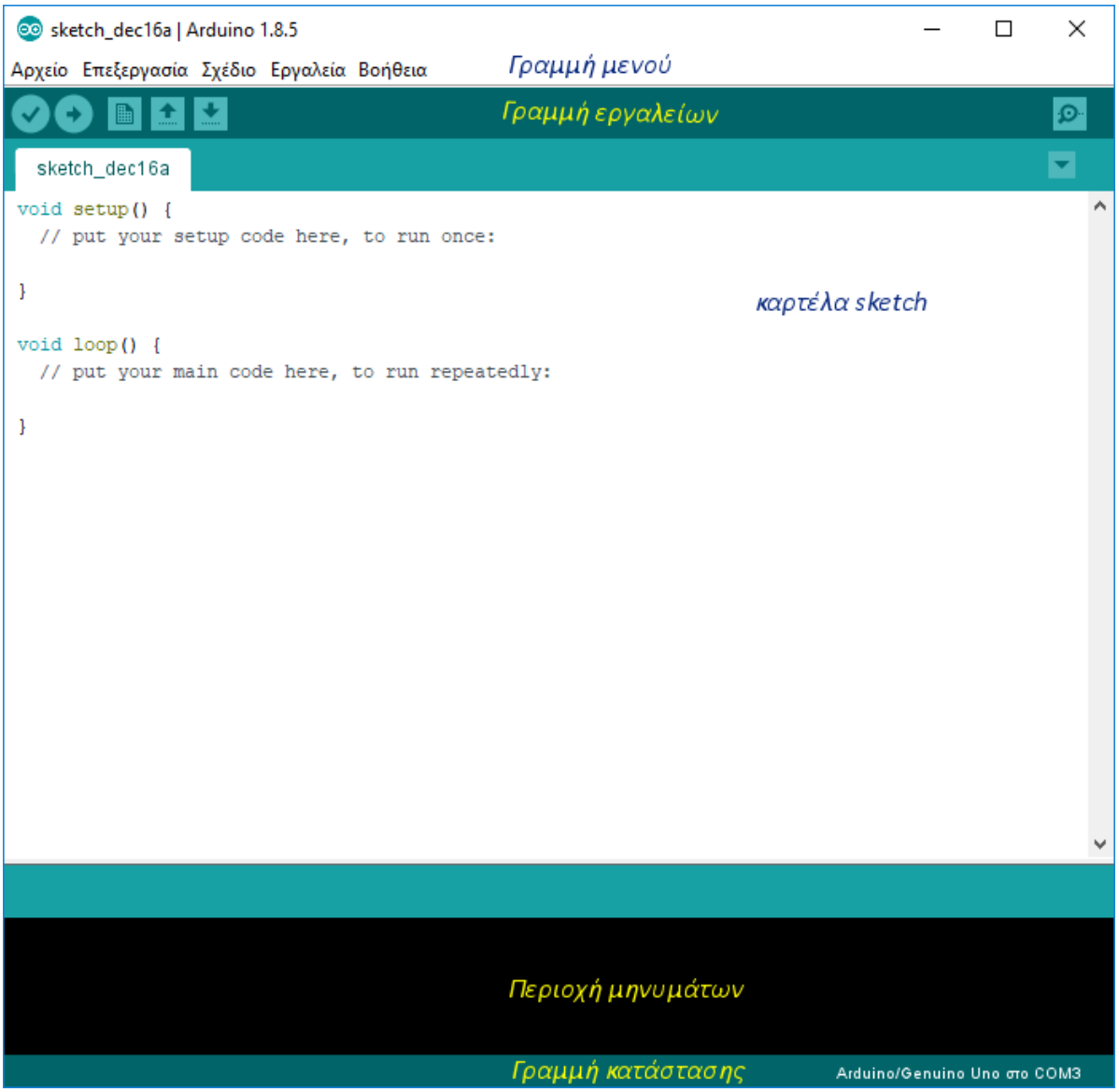
Το λογισμικό του Arduino

1. Bootloader

Ο bootloader είναι ένα μικρού μεγέθους πρόγραμμα (0,5 kB στην περίπτωση του Arduino Uno) προεγκατεστημένο από τον κατασκευαστή στη μνήμη flash του μικροελεγκτή, που εκτελείται κατά την τροφοδοσία ή την επανεκκίνηση του Arduino. Ο ρόλος του είναι να επιτρέπει τη μεταφόρτωση των προγραμμάτων μας στη μνήμη του Arduino από τον υπολογιστή μέσω του καλωδίου USB, χωρίς να είναι απαραίτητη για το σκοπό αυτό η χρήση ειδικών συσκευών προγραμματισμού.

2. Ολοκληρωμένο περιβάλλον ανάπτυξης (Arduino IDE)

Το ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) του Arduino είναι μια εφαρμογή που εκτελείται στον υπολογιστή, στον οποίο μέσω της θύρας USB συνδέεται ο Arduino. Είναι γραμμένη σε Java, και προέρχεται από το IDE της γλώσσας προγραμματισμού Processing. Η σχεδίαση του IDE είναι μινιμαλιστική. Περιλαμβάνει ένα επεξεργαστή κειμένου για τη συγγραφή των προγραμμάτων (που στη γλώσσα του Arduino αποκαλούνται sketches), ενώ με τη χρήση εξωτερικών προγραμμάτων εκτελεί τη μεταγλώττιση του sketch σε κώδικα κατανοητό από τον μικροελεγκτή ("γλώσσα μηχανής"), και μεταφορτώνει με ένα «κλικ» τον κώδικα στον Arduino.



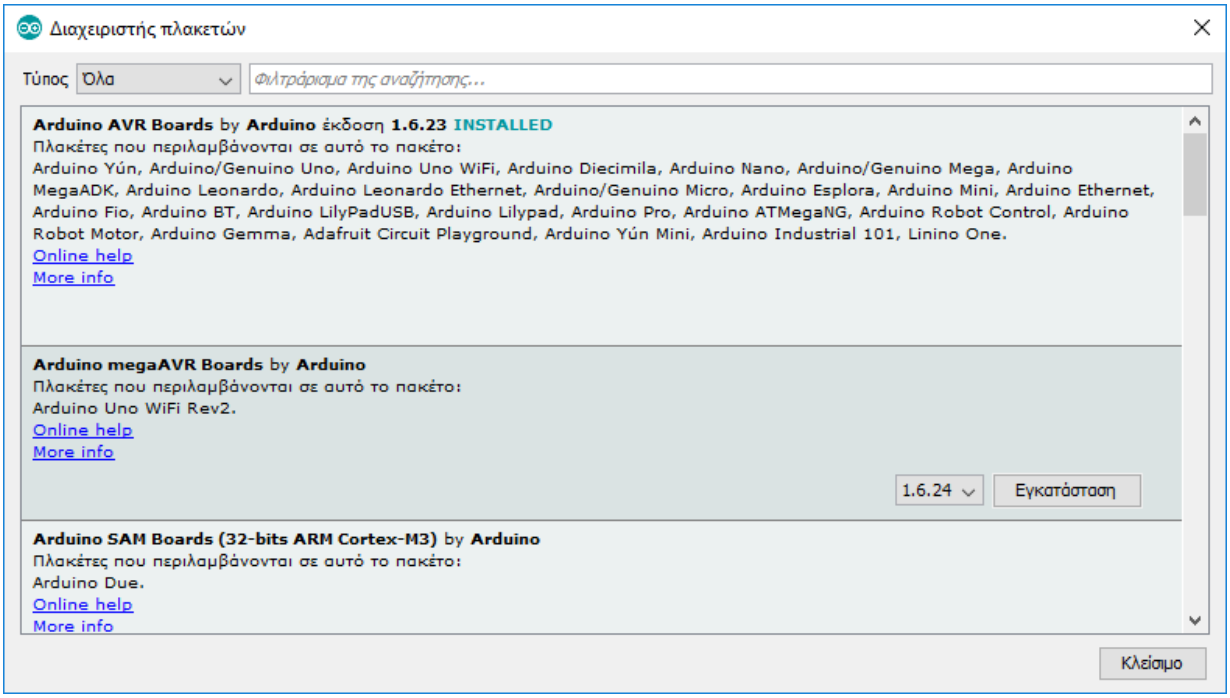
Εικόνα 6: Το παράθυρο του ολοκληρωμένου περιβάλλοντος εργασίας του Arduino

Στο κύριο παράθυρο του Arduino IDE περιλαμβάνονται :

- Η γραμμή μενού.
- Η γραμμή εργαλείων (Επικύρωση, Ανέβασμα, Δημιουργία, Άνοιγμα, Αποθήκευση και Παρακολούθηση σειριακής).
- Η περιοχή με τις καρτέλες του επεξεργαστή κειμένου (για τη συγγραφή των sketches).
- Η περιοχή εμφάνισης διαφόρων μηνυμάτων (π.χ. μηνύματα λάθους του μεταγλωττιστή, μηνύματα κατά το ανέβασμα του κώδικα στον Arduino, κ.λ.π.).
- Η γραμμή κατάστασης με τις πληροφορίες για το είδος της χρησιμοποιούμενης πλακέτας, και τη σειριακή θύρα μέσω της οποίας γίνεται η επικοινωνία υπολογιστή - Arduino.

Το ολοκληρωμένο περιβάλλον ανάπτυξης περιλαμβάνει και μια ***σειριακή κονσόλα*** *(τερματικό)*, μέσω της οποίας ο χρήστης μπορεί να αλληλεπιδρά με τον Arduino. Ο Arduino μπορεί προγραμματιστικά να αποστέλλει πληροφορίες, οι οποίες θα εμφανίζονται στο τερματικό, ή να λαμβάνει πληροφορίες που του αποστέλλονται μέσω του τερματικού.

Τέλος με τον ***διαχειριστή πλακετών*** και το ***διαχειριστή βιβλιοθηκών*** το λογισμικό του Arduino μπορεί να αναβαθμίζεται, είτε για τη βελτίωση της λειτουργίας των υποστηριζόμενων πλακετών και δυνατοτήτων, είτε για την υποστήριξη και άλλων νεότερων πλακετών μικροελεγκτών ή την εγκατάσταση επιπλέον δυνατοτήτων.



Εικόνα 7: Το παράθυρο του διαχειριστή πλακετών

Παρότι ο έμπειρος προγραμματιστής θα το βρει αρκετά περιοριστικό, το ολοκληρωμένο περιβάλλον ανάπτυξης του Arduino παρέχει στον αρχάριο όλα όσα χρειάζεται για τη διαχείριση του Arduino από τον υπολογιστή του.

3. Βιβλιοθήκες

Οι τυπικές βιβλιοθήκες που συνοδεύουν τον Arduino είναι αρχεία κώδικα γραμμένα σε C ή C++, που παρέχουν επιπλέον λειτουργικότητα στη διαχείριση του υλικού και των δεδομένων και βελτιώνουν την αναγνωσιμότητα του κώδικα. Η βιβλιοθήκη πυρήνα (core library) συμπυκνώνει χαμηλού επιπέδου πτυχές του προγραμματισμού του μικροελεγκτή (π.χ. διαχείριση καταχωρητών),

επιτρέποντας στους χρήστες να επικεντρωθούν στο κάθε φορά ιδιαίτερο έργο τους, και όχι στην χρονοβόρα και επίπονη διαδικασία εκμάθησης του προγραμματισμού σε χαμηλό επίπεδο (γλώσσα μηχανής). Η βιβλιοθήκη πυρήνα εξ ορισμού συμπεριλαμβάνεται σε οποιοδήποτε πρόγραμμά μας (χωρίς αυτό να απαιτεί κάποια ιδιαίτερη ενέργεια από εμάς). Λόγω της περιορισμένης μνήμης του μικροελεγκτή, μέρος του κώδικα έχει διαχωριστεί σε επιμέρους βιβλιοθήκες, οι οποίες μπορεί κατά περίπτωση να συμπεριληφθούν όταν απαιτείται σε κάποιο πρόγραμμα. Μεταξύ αυτών περιλαμβάνονται:

- **EEPROM** - διαχείριση της μνήμης EEPROM του μικροελεγκτή
- **Ethernet** - Για σύνδεση στο διαδίκτυο μέσω του Ethernet Shield
- **Firmata** - Για την επικοινωνία με εφαρμογές στον υπολογιστή χρησιμοποιώντας το τυπικό σειριακό πρωτόκολλο.
- **GSM** - για τη σύνδεση σε ένα δίκτυο GSM/GRPS με το GSM shield.
- **LiquidCrystal** - για τον έλεγχο οθόνης υγρών κρυστάλλων (LCD).
- **SD** - διαχείριση καρτών μνήμης SD.
- **Servo** - για τον έλεγχο σερβοκινητήρων.
- **SPI** - για την επικοινωνία με συσκευές μέσω του Serial Peripheral Interface (SPI) Bus
- **SoftwareSerial** - για τη σειριακή επικοινωνία με χρήση οποιωνδήποτε ψηφιακών ακίδων.
- **Stepper** - για τον έλεγχο βηματικών κινητήρων.
- **TFT** - για τον έλεγχο οθόνης TFT.
- **WiFi** - Για ασύρματη σύνδεση στο διαδίκτυο μέσω του Arduino WiFi shield.
- **Wire** - για την επικοινωνία με συσκευές μέσω του Two Wire Interface (TWI/I^2C).

Ιδιαίτερη προσπάθεια έχει γίνει, ώστε οι βιβλιοθήκες πυρήνα να απλοποιούν τη διαδικασία συγγραφής του κώδικα χωρίς να περιορίζουν υπερβολικά την ευελιξία του χρήστη. Παρ' όλα αυτά διαδικασίες όπως η ανάγνωση μιας ψηφιακής εισόδου ή εγγραφή μιας τιμής σε κάποια ψηφιακή έξοδο (digitalRead ή digitalWrite) ή η ψηφιακή μετατροπή και ανάγνωση της επιστρεφόμενης τιμής σε μια αναλογική είσοδο (analogRead) -διαδικασίες βασικές για ένα σύστημα μετρήσεων- μπορούν να υλοποιηθούν ώστε να τρέχουν πολλές φορές γρηγορότερα.

Η κοινότητα

Αποτελεί την τρίτη συνιστώσα της επιτυχίας του Arduino. Το ανοιχτό υλικό, ο ανοιχτός κώδικας και η ευκολία διαχείρισης του συστήματος συνέβαλλαν καθοριστικά στη δημιουργία μιας κοινότητας περί το project. Ίσως η σημαντικότερη συνεισφορά της είναι το πλήθος των επιπλέον βιβλιοθηκών που έχει αναπτυχθεί από τα μέλη της κοινότητας των χρηστών του Arduino αυξάνοντας τόσο τη λειτουργικότητα όσο και την απόδοση της πλατφόρμας.

Στοιχεία προγραμματισμού του Arduino

Τα βασικά

Η δύναμη της γλώσσας προγραμματισμού C/C++ κρύβεται πίσω από ένα πρόγραμμα (sketch) του Arduino, παρότι η δομή του sketch δεν είναι η τυπική ενός προγράμματος C/C++. Μόνο δύο συναρτήσεις είναι απαραίτητες για τη δημιουργία ενός απλού προγράμματος κυκλικής εκτέλεσης στον Arduino:

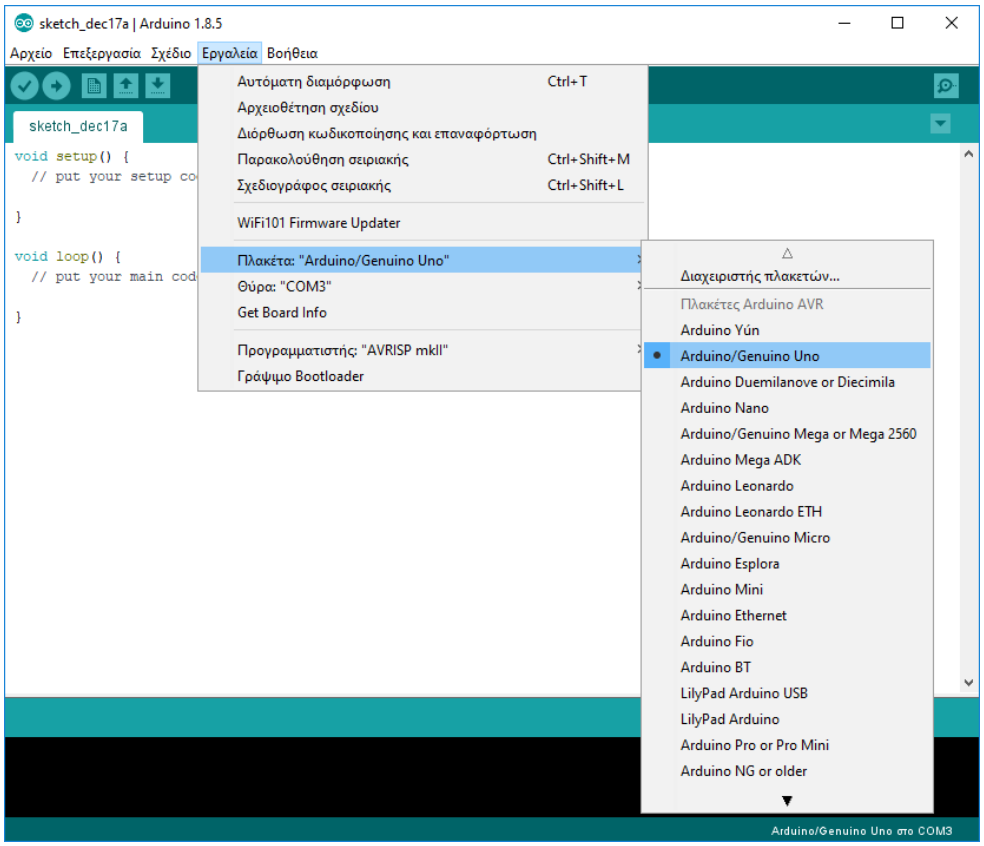
- Η συνάρτηση **setup()**: Εκτελείται μόνο μια φορά στην αρχή του προγράμματος, και καθορίζει της αρχικές ρυθμίσεις του συστήματος, π.χ. ορισμός εισόδων/εξόδων, ενεργοποίηση σειριακής επικοινωνίας, κλπ.
- Η συνάρτηση **loop()**: Περιέχει τον κυρίως κώδικα του προγράμματός μας, και εκτελείται κυκλικά μέχρι την απενεργοποίηση ή την επανεκκίνηση του Arduino.

Και οι δύο συναρτήσεις είναι τύπου **void**, που σημαίνει πως κατά την εκτέλεσή τους δεν επιστρέφουν κάποια τιμή (αποτέλεσμα) στον Arduino.

Πριν το πρώτο sketch - προαπαιτούμενες ρυθμίσεις

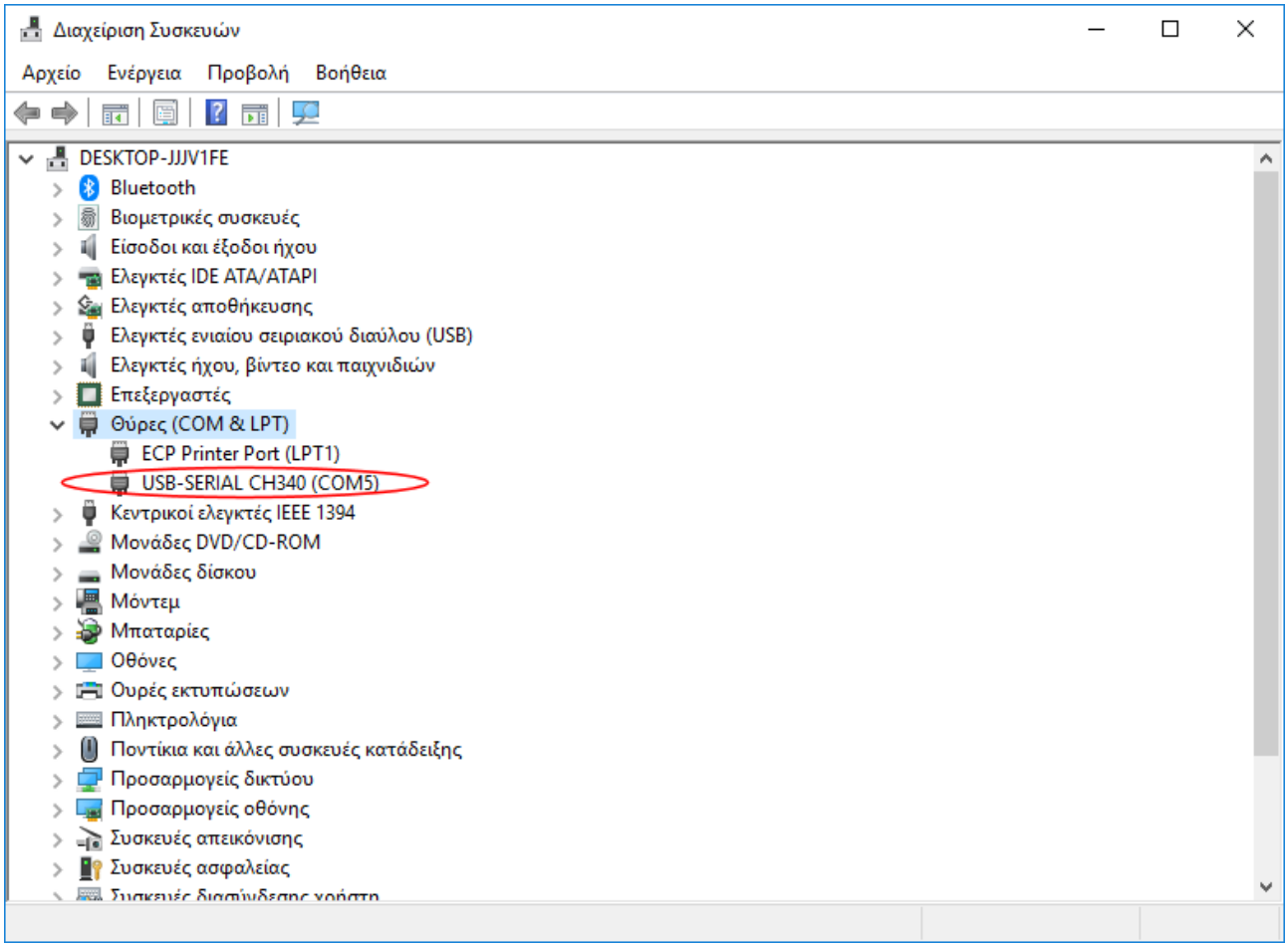
Πριν γράψουμε οποιοδήποτε πρόγραμμα (sketch), και για τη σωστή ολοκλήρωση της διαδικασίας του "ανεβάσματος" του κώδικα στον Arduino, απαιτούνται οι εξής ρυθμίσεις:

1. Να επιλέξουμε, μέσω του μενού *"Εργαλεία/Πλακέτα"*, το μοντέλο του Arduino (ή άλλης συμβατής πλακέτας) που χρησιμοποιούμε, π.χ. Arduino/Genuino Uno, Arduino Leonardo, Arduino Mini, κ.λ.π.
2. Να επιλέξουμε, μέσω του μενού *"Εργαλεία/Θύρα"*, τη σειριακή θύρα (π.χ. COM5) μέσω της οποίας η πλακέτα του Arduino επικοινωνεί με τον υπολογιστή στον οποίο συνδέεται.



Εικόνα 8: Προαπαιτούμενες ρυθμίσεις

Στην περίπτωση που η σειριακή θύρα, στην οποία ο υπολογιστής "συνομιλεί" με τον Arduino, δεν είναι γνωστή, μπορούμε να την εντοπίσουμε ως εξής: Αφού αποσυνδέσουμε το USB καλώδιο από τον Arduino, ανοίγουμε τον *"Πίνακα Ελέγχου"* των Windows, επιλέγουμε την εμφάνιση της καρτέλας *"Υλικό και Ήχος"*, και στην περιοχή εφαρμογών *"Συσκευές και εκτυπωτές"* επιλέγουμε την εφαρμογή *"Διαχείριση συσκευών"*. Στον πίνακα με τις κατηγορίες των συνδεδεμένων συσκευών επιλέγουμε την εμφάνιση ("ξεδίπλωμα" της λίστας) των συσκευών τύπου *"Θύρες (COM & LPT)"* και μετά συνδέουμε με το USB καλώδιο τον Arduino με τον υπολογιστή. Μια νέα σειριακή θύρα εμφανίζεται στον πίνακα των συνδεδεμένων συσκευών. Αυτή είναι η σειριακή θύρα μέσω της οποίας επιτυγχάνεται η επικοινωνία Arduino - υπολογιστή.



Εικόνα 9: Διαχείριση συσκευών - στη θύρα COM5 ο υπολογιστής "συνομιλεί" με τον Arduino (CH-340 κλώνο)

Τόσο η επιλεγμένη πλακέτα, όσο και η επιλεγμένη σειριακή θύρα αναγράφονται στη γραμμή κατάστασης του Arduino IDE (Εικόνα 6).

Το πρώτο sketch - χρήση ψηφιακής ακίδας ως εξόδου

Στην πλακέτα του Arduino ένα LED, μέσω κατάλληλης αντίστασης, συνδέεται στην ψηφιακή ακίδα 13. Το πρώτο απλό πρόγραμμα κάνει χρήση αυτού του χαρακτηριστικού:

```
/*
        Ex.1 : Αναβοσβήνοντας το ενσωματωμένο στον Arduino LED.
*/
void setup ()
{
        pinMode (13, OUTPUT);           // Η ακίδα 13 καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
        digitalWrite (13, HIGH);        // Άναψε το LED
        delay (1000);                   // Αναμονή 1sec (1000 milliseconds)
        digitalWrite (13, LOW);         // Σβήσε το LED
        delay (1000);                   // Αναμονή 1sec
}
```

Σημείωση: *Σε μια γραμμή του sketch ότι υπάρχει μετά τις δύο γραμμές "//" θεωρείται επεξηγηματικό σχόλιο. Σχόλια που εκτείνονται σε περισσότερες γραμμές περικλείονται μεταξύ των "/*" και "*/" (χωρίς τα εισαγωγικά εννοείται). Το σώμα των εντολών μιας οποιασδήποτε συνάρτησης πρέπει να περικλείεται μεταξύ δύο αγκίστρων {...}. Γενικά χρησιμοποιούμε τα δύο άγκιστρα για να δημιουργήσουμε ένα μπλοκ εντολών, οι οποίες εκτελούνται υπό συγκεκριμένες συνθήκες στο πρόγραμμά μας.*

Παρότι το όνομα των χρησιμοποιούμενων εντολών χαρακτηρίζει με ικανοποιητική σαφήνεια και τη λειτουργία τους, ας σχολιάσουμε κάποια σημεία [4]:

- Η εντολή *pinMode(pin, mode)* καθορίζει αν μια ψηφιακή ακίδα (pin) θα χρησιμοποιείται από το πρόγραμμά μας ως έξοδος (OUTPUT) ή ως είσοδος (INPUT). Μια επιπλέον ειδική επιλογή μιας ακίδας ως εισόδου είναι η INPUT_PULLUP με την οποία ενεργοποιείται μια εσωτερική αντίσταση και η αντίστοιχη είσοδος προσδένεται στην τάση τροφοδοσίας.
- Με την εντολή *digitalWrite(pin, value)* "γράφουμε" την τιμή HIGH (1) ή LOW (0) σε μια ψηφιακή έξοδο (παράμετρος *pin*). Στην περίπτωση του Arduino Uno λογική τιμή HIGH σε μια ψηφιακή ακίδα σημαίνει ότι θέτουμε την τάση στην ακίδα αυτή στην τιμή 5 V. Αντίστοιχα λογική τιμή LOW σημαίνει πρόσδεση της ακίδας αυτής στη γείωση (τάση 0 V).
- Η εντολή *delay(value)* διακόπτει την εκτέλεση του κώδικα για χρόνο (σε ms) ίσο με την τιμή της παραμέτρου *value*. Γενικά η χρήση της πρέπει να αποφεύγεται για μεγάλες χρονικές διάρκειες παύσης (εκτός βέβαια από την περίπτωση πολύ απλών προγραμμάτων), αφού κατά τη διάρκεια της λειτουργίας της, και πέρα από κάποιες εξειδικευμένες λειτουργίες (π.χ. διακοπές υλικού ή χρονιστή), παύει κάθε δραστηριότητα του μικροελεγκτή.

Αφού αποθηκεύσουμε το sketch μέσω του μενού *"Αρχείο/Αποθήκευση ως..."* με κάποιο όνομα της αρεσκείας μας, κάνουμε στη συνέχεια κλικ στο κουμπί *"Ανέβασμα"* (ή *"Upload to I/O Board"*) στη γραμμή εργαλείων του IDE του Arduino, και το λογισμικό αναλαμβάνει:

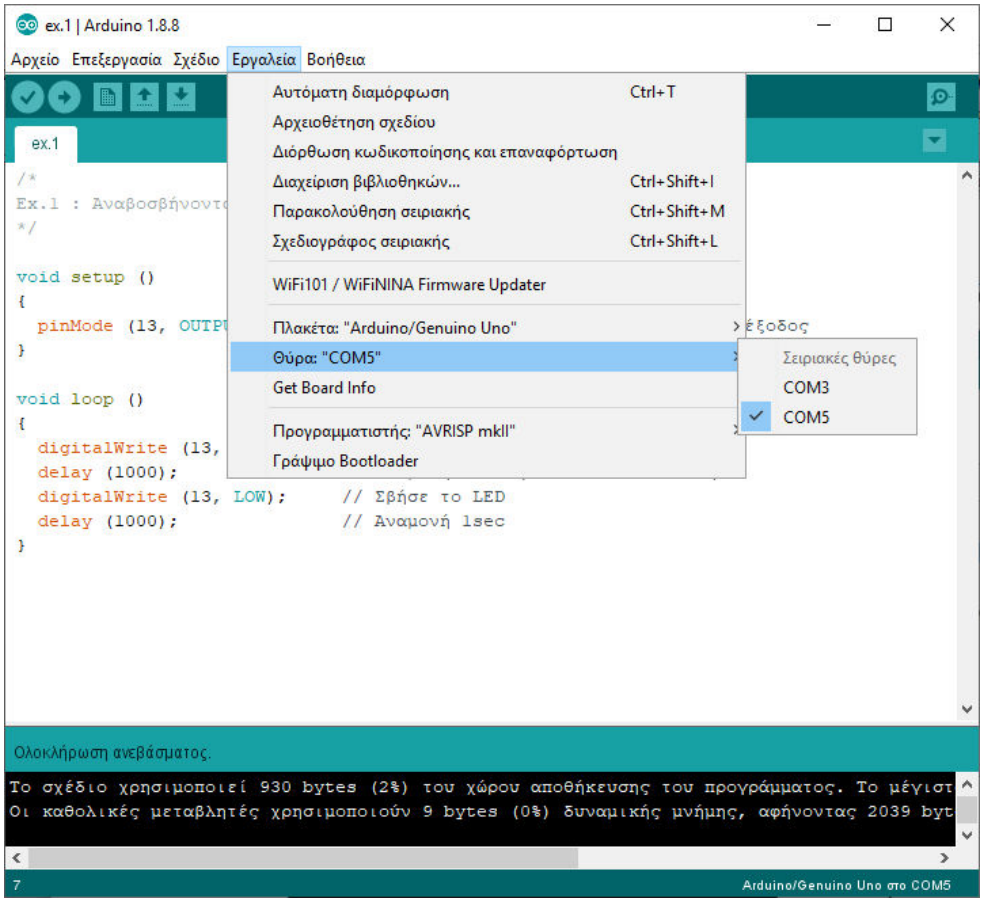
- Να πραγματοποιήσει ένα συντακτικό προέλεγχο του προγράμματος.
- Να τροποποιήσει το πρόγραμμα ώστε να το μετατρέψει σε έγκυρο C++ πρόγραμμα.

- Να μεταγλωττίσει το πρόγραμμα με χρήση του εξωτερικού προγράμματος ***avr-gcc***, δημιουργώντας ένα αρχείο που περιέχει τον κώδικα σε μορφή κατανοητή από τον μικροελεγκτή (γλώσσα μηχανής σε δεκαεξαδικό κώδικα).
- Να ανεβάσει το αρχείο με τον κώδικα στη μνήμη του Arduino, με χρήση του εξωτερικού προγράμματος ***avrdude***, και αφού προηγουμένως έχει εξαναγκάσει τον Arduino σε επανεκκίνηση (reset).

Μετά το ανέβασμα του κώδικα στον Arduino αρχίζει αυτόματα να εκτελείται, μέχρι να αποσυνδέσουμε τον Arduino από την τροφοδοσία ή να εξαναγκάσουμε το σύστημα σε επανεκκίνηση.

Ας τονίσουμε για μια ακόμη φορά τις προαπαιτούμενες για κάθε νέο sketch ρυθμίσεις:

1. Μέσω του μενού *"Εργαλεία/Πλακέτα"*, επιλέγουμε το μοντέλο του Arduino (ή άλλης συμβατής πλακέτας) που χρησιμοποιούμε, π.χ. Arduino/Genuino Uno, Arduino Leonardo, Arduino Mini, κ.λ.π..
2. Μέσω του μενού *"Εργαλεία/Θύρα"*, επιλέγουμε τη σειριακή θύρα (π.χ. COM5) μέσω της οποίας η πλακέτα του Arduino επικοινωνεί με τον υπολογιστή στον οποίο συνδέεται.

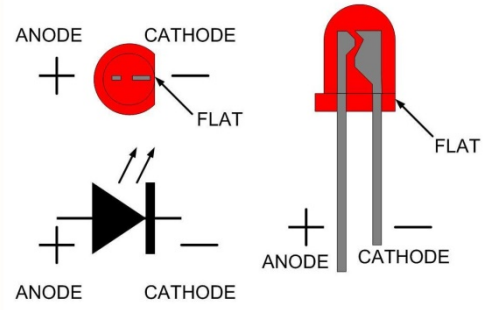


Εικόνα 10 : Επιλογή σειριακής θύρας

Σύνδεση LED σε άλλη ψηφιακή έξοδο - τροποποίηση του σχετικού sketch

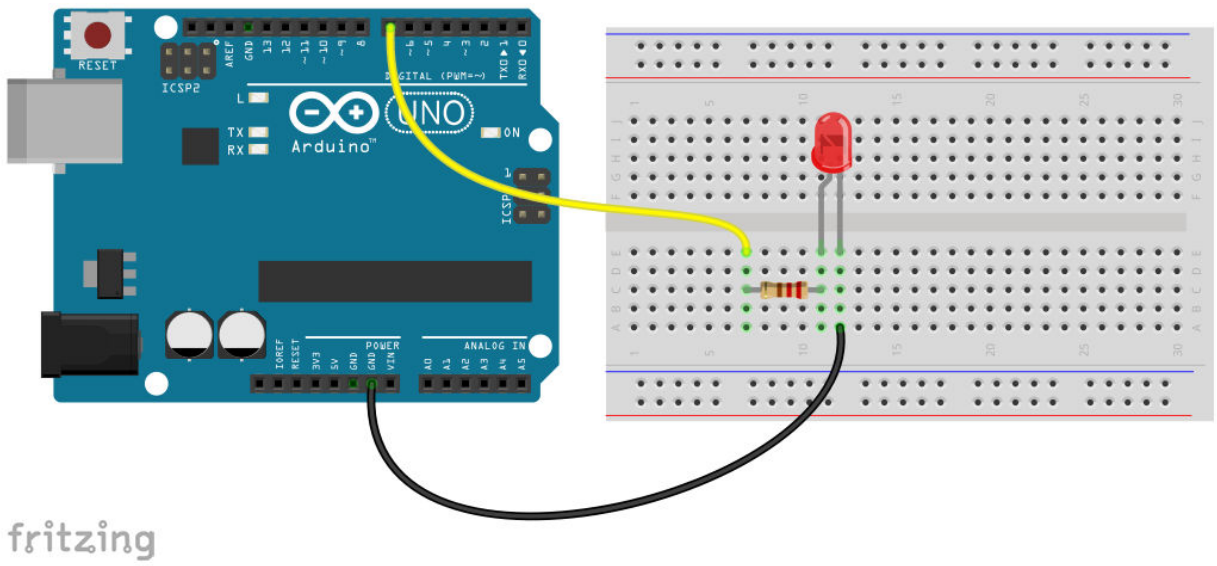
Η δίοδος εκπομπής φωτός (Light Emitting Diode ή απλά LED) είναι ένα σύστημα αποτελούμενο από ένα ημιαγωγό τύπου ***p*** και ένα ημιαγωγό τύπου ***n*** σε άμεση επαφή που, όταν συνδεθεί σε ηλεκτρική τάση κατά την ορθή φορά πόλωσης (δηλ. ο p-ημιαγωγός σε υψηλότερο δυναμικό σε σχέση με τον n-ημιαγωγό), εκπέμπει φως στενού φασματικού εύρους. Η διάταξη περικλείεται μέσα σε πλαστικό έγχρωμο ή διαφανές περίβλημα. Το χρώμα του φωτός που εκπέμπεται μπορεί να είναι υπέρυθρο, ορατό, ή υπεριώδες, και εξαρτάται από τη χημική σύσταση του ημιαγώγιμου υλικού που χρησιμοποιείται.

Τα απλά LED διαθέτουν δυο ακροδέκτες για τη σύνδεσή τους σε οποιοδήποτε κύκλωμα: Το θετικό άκρο, που είναι ο μεγαλύτερου μήκους ακροδέκτης και ονομάζεται *"άνοδος"*, και το αρνητικό άκρο που ονομάζεται *"κάθοδος"*. Σε περίπτωση που οι δύο ακίδες έχουν το ίδιο μήκος, η κάθοδος μπορεί να αναγνωριστεί, καθώς είναι η ακίδα που βρίσκεται πλησιέστερα στην επίπεδη πλευρά του πλαστικού περιβλήματος του LED. Το LED συνδέεται στην ηλεκτρική τάση σε σειρά με μια αντίσταση για τον περιορισμό του ηλεκτρικού ρεύματος, και κατά τέτοιο τρόπο ώστε η άνοδος να βρίσκεται σε υψηλότερο δυναμικό σε σχέση με την κάθοδο (ορθή φορά πόλωσης).



Εικόνα 11: Η δίοδος εκπομπής φωτός (LED)

Θα χρησιμοποιήσουμε ένα LED, το οποίο -μέσω κατάλληλης αντίστασης (μας κάνουν αντιστάσεις από 220Ω μέχρι 1kΩ) για τον περιορισμό του ρεύματος- θα συνδέσουμε σε κάποια ψηφιακή ακίδα του Arduino (π.χ. στην ακίδα 7). Οι σχετικές συνδέσεις, που γίνονται με τη βοήθεια κατάλληλων καλωδίων και breadboard, φαίνονται στο ακόλουθο σχήμα:



Εικόνα 12: Σύνδεση LED στον Arduino

Το πρόγραμμα (sketch) του Arduino για το διαδοχικό άναμμα και σβήσιμο του LED διαμορφώνεται ως εξής:

```
/*
        Ex.2 : Αναβοσβήνουμε ένα LED που συνδέεται στην ψηφιακή ακίδα 7 του Arduino
*/

void setup ()
{
        pinMode (7, OUTPUT);            // Η ακίδα 7 καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
        digitalWrite (7, HIGH);         // Άναψε το LED
        delay (1000);                   // Αναμονή 1sec (1000 milliseconds)
        digitalWrite (7, LOW);          // Σβήσε το LED
        delay (1000);                   // Αναμονή 1sec
}
```

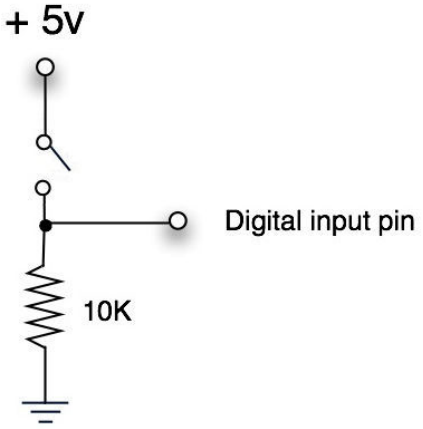
Αν θέλουμε να αλλάξουμε την ψηφιακή ακίδα σύνδεσης του LED στον Arduino, οι αλλαγές στον κώδικα διευκολύνονται με χρήση του ορισμού ***#define***. Για παράδειγμα:

```
/*
        Ex.3 : Αναβοσβήνουμε ένα LED που συνδέεται σε κάποια ψηφιακή ακίδα του Arduino
*/

#define LEDPIN      8

void setup ()
{
        pinMode (LEDPIN, OUTPUT);       // Η ακίδα LEDPIN καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
        digitalWrite (LEDPIN, HIGH);    // Άναψε το LED
        delay (1000);                   // Αναμονή 1sec (1000 milliseconds)
        digitalWrite (LEDPIN, LOW);     // Σβήσε το LED
        delay (1000);                   // Αναμονή 1sec
}
```

Ας παρατηρήσουμε ότι στο sketch όλες οι αριθμητικές αναφορές στην ακίδα που συνδέεται το LED έχουν αντικατασταθεί με το λεκτικό LEDPIN. Στην περίπτωση αυτή, πριν το τελικό στάδιο της μεταγλώττισης, κάθε εμφάνιση του ονόματος LEDPIN στο sketch αυτόματα αντικαθίσταται από το λογισμικό με τη σταθερή τιμή 8. Έτσι αν θέλουμε να αλλάξουμε την ακίδα σύνδεσης του LED στον Arduino (και εκτός από τις όποιες αλλαγές στο κύκλωμα), στο sketch χρειάζεται μόνο να αλλάξουμε την αριθμητική τιμή που εμφανίζεται στον ορισμό LEDPIN.

Χρήση ψηφιακής ακίδας ως εισόδου

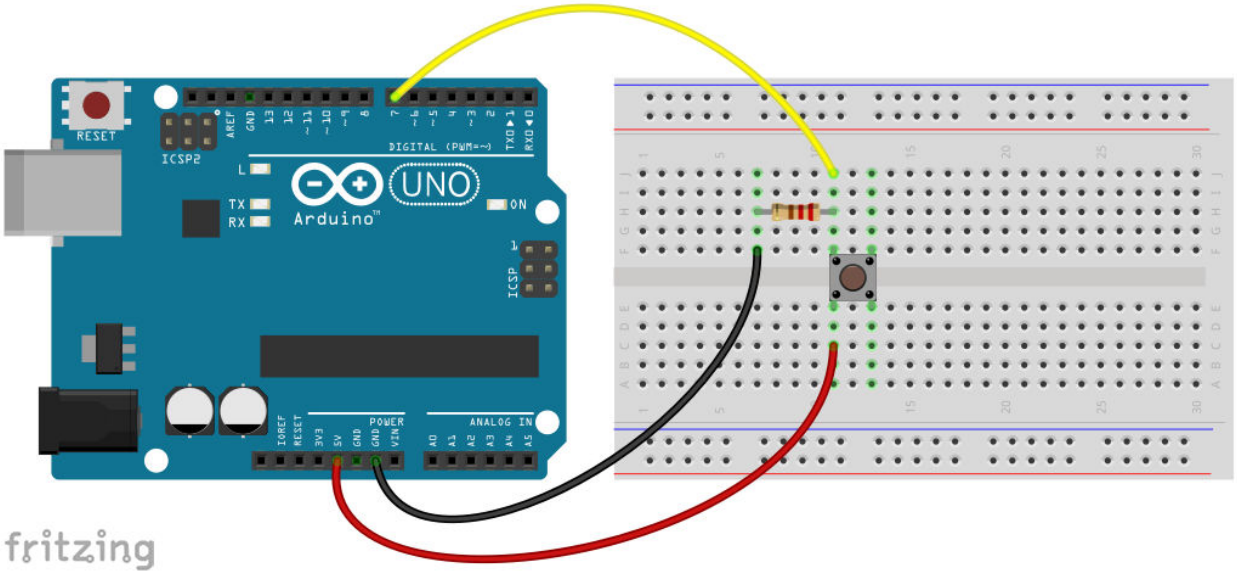
Στα προηγούμενα sketch είδαμε πώς μπορούμε να χρησιμοποιήσουμε μια ψηφιακή ακίδα του Arduino ως έξοδο, και συνεπώς πώς γράφουμε -με την εντολή *digitalWrite()*- στην ακίδα αυτή λογικό 1 ή 0 (HIGH ή LOW). Αυτό έχει σαν αποτέλεσμα την εμφάνιση τάσης +5V ή 0V αντίστοιχα στην ακίδα, και άρα μπορεί να χρησιμεύσει για να ανάψει ή να σβήσει ένα LED, που συνδέεται στην ίδια ακίδα. Θα εξετάσουμε τώρα τη χρήση μιας ψηφιακής ακίδας ως εισόδου, την τιμή της οποίας μπορούμε να διαβάσουμε με τη σχετική εντολή *digitalRead()*. Εννοείται πως την τιμή αυτή "εγγράφει" στην ακίδα κάποιο κατάλληλο εξωτερικό ηλεκτρικό κύκλωμα που συνδέεται σ' αυτή. Το απλούστερο τέτοιο κύκλωμα είναι ένα που αποτελείται από ένα στιγμιαίο διακόπτη (momentary push button) και μία αντίσταση σε σειρά. Το σύστημα τροφοδοτείται από τάση 5V, και το κοινό σημείο της αντίστασης και του διακόπτη οδηγείται σε κάποια ψηφιακή ακίδα (π.χ. την ακίδα 7) του Arduino.



Εικόνα 13: Σχηματικό διάγραμμα σύνδεσης διακόπτη στον Arduino

Όταν ο διακόπτης είναι ανοικτός η ψηφιακή ακίδα μέσω της (pull down) αντίστασης των 10 kΩ οδηγείται στη γη (γειώνεται), άρα λογικό 0 εγγράφεται στην ακίδα. Όταν ο διακόπτης κλείσει η ακίδα οδηγείται σε υψηλό δυναμικό +5V (λογικό 1).

Οι αντίστοιχες συνδέσεις στον Arduino φαίνονται στο επόμενο σχήμα:



Εικόνα 14: Σύνδεση διακόπτη push button στον Arduino

Ο (ημιτελής προς το παρόν) κώδικας του Arduino, μπορεί να έχει τη μορφή:

```
/*
        Ανάγνωση ψηφιακής ακίδας του Arduino
*/

#define BUTTONPIN           7

void setup ()
{
        pinMode (BUTTONPIN, INPUT);             // Η ακίδα 7 καθορίζεται ως ψηφιακή είσοδος
}

void loop ()
{
        int value = digitalRead (BUTTONPIN);    // Διάβασε την κατάσταση του διακόπτη

        // Εδώ πρέπει να κάνουμε κάτι, ανάλογα με την τιμή της ακίδας 7

}
```

Το αποτέλεσμα της ανάγνωσης της ψηφιακής ακίδας BUTTONPIN (7), το αποθηκεύουμε σε μια μεταβλητή με το περιγραφικό όνομα *value*. Μπορούμε να σκεφτόμαστε μια μεταβλητή ως μια περιοχή της μνήμης RAM του Arduino, στην οποία μπορούμε να γράφουμε κάποια τιμή, να την τροποποιούμε, και να τη διαβάζουμε κατά την εκτέλεση ενός προγράμματος. Επιπλέον ενημερώνουμε τον μεταγλωττιστή πως η μεταβλητή value είναι ακέραιου (*int*) τύπου, ώστε να δεσμευτεί κατάλληλου μεγέθους περιοχή της μνήμης (περισσότερα για τους τύπους δεδομένων που υποστηρίζει ο Arduino στο Παράρτημα Α). Στο sketch η τιμή της μεταβλητής *value* γίνεται 1 (HIGH) όταν ο διακόπτης είναι πατημένος, και 0 (LOW) όταν ο διακόπτης απελευθερωθεί.

Παρατήρηση : *Ο ορισμός μιας μεταβλητής (variable) ακολουθεί το γενικό σχήμα:*

Επιστρεφόμενος_τύπος Όνομα_μεταβλητής = αρχική τιμή;

Η απόδοση κάποιας αρχικής τιμής στη μεταβλητή είναι προαιρετικός. Μεταβλητές που ορίζονται μέσα στο σώμα μιας συνάρτησης έχουν ισχύ (δηλ. είναι προσβάσιμες για εγγραφή ή ανάγνωση) μόνο εντός της συνάρτησης και χαρακτηρίζονται ως ***τοπικές*** *(local), ενώ μεταβλητές που ορίζονται έξω από οποιαδήποτε συνάρτηση έχουν καθολική ισχύ και χαρακτηρίζονται ως* ***σφαιρικές*** *(global). Εκτός από μεταβλητές μπορούμε στα προγράμματά μας να ορίσουμε και σταθερές (constants), δηλ. τιμές που δε μπορούν να μεταβληθούν κατά την εκτέλεση του προγράμματος, σύμφωνα με το γενικό σχήμα:*

const Επιστρεφόμενος_τύπος Όνομα_μεταβλητής = αρχική τιμή;

Για παράδειγμα με τη δήλωση:

const float pi = 3.14159;

ορίζουμε μια σταθερά τύπου κινητής υποδιαστολής, με το όνομα pi και τιμή 3,14159.

Βέβαια δεν είναι αρκετό να διαβάσουμε την τιμή μιας ψηφιακής ακίδας, αλλά χρειάζεται το πρόγραμμά μας να παίρνει αποφάσεις, και να εκτελεί τις αντίστοιχες εντολές ανάλογα με την κατάσταση της ακίδας. Για το λόγο αυτό στο παραπάνω ημιτελές sketch υπάρχει το σχόλιο:

// Εδώ πρέπει να κάνουμε κάτι, ανάλογα με την τιμή της ακίδας 7

Ας δεχτούμε στο σημείο αυτό πως θέλουμε, όταν ο διακόπτης είναι κλειστός, να ανάβει το LED που συνδέεται στην ακίδα 13 του Arduino (και είναι, όπως έχουμε ήδη πει, ενσωματωμένο στην πλακέτα του Arduino Uno), και όταν ο διακόπτης είναι ανοιχτός το LED να είναι σβηστό. Μια πρώτη προσέγγιση στο πρόβλημα είναι η εξής: Η ψηφιακή ακίδα 13, που πρέπει να έχει χαρακτηριστεί ως έξοδος (OUTPUT), θα πρέπει να βρίσκεται πάντα στην ίδια κατάσταση (HIGH ή LOW)

με την ακίδα 7, στην οποία είναι συνδεδεμένος ο διακόπτης.

Το αντίστοιχο sketch παίρνει τη μορφή:

```
/*
        Ex.4 : Ανάγνωση ψηφιακής ακίδας του Arduino
*/

#define BUTTONPIN       7
#define LEDPIN          13

void setup ()
{
        pinMode(BUTTONPIN, INPUT);          // Η ακίδα 7 καθορίζεται ως ψηφιακή είσοδος
        pinMode(LEDPIN, OUTPUT);            // Η ακίδα 13 καθορίζεται ως ψηφιακή έξοδος
}
void loop ()
{
        int value = digitalRead(BUTTONPIN);     // Διάβασε την κατάσταση του διακόπτη

        digitalWrite(LEDPIN, value);            // Στην ακίδα 13 γράφουμε την τιμή της ακίδας 7
}
```

Οι δύο εντολές στη συνάρτηση *loop()*, θα μπορούσαν να αντικατασταθούν με μία, ως εξής:

```
digitalWrite(LEDPIN, digitalRead(BUTTONPIN));
```

Ο κατ' αυτό τον τρόπο συνδυασμός των δύο εντολών έχει το πλεονέκτημα πως δεν απαιτείται δέσμευση της περιορισμένης μνήμης RAM για την αποθήκευση της κατάστασης της ακίδας 7 (BUTTONPIN). Φυσικά, στο συγκεκριμένο απλό sketch η περιορισμένη μνήμη RAM του Arduino δεν αποτελεί πρόβλημα.

Ας σημειώσουμε πως με το τελευταίο sketch ο Arduino είναι σε θέση αυτόματα να λαμβάνει μια απλή απόφαση: Πότε να ανάβει και πότε να σβήνει ένα LED. Θα συζητήσουμε σε επόμενη παράγραφο πώς μπορούμε να υλοποιούμε πολυπλοκότερες διαδικασίες λήψης αποφάσεων στον Arduino. Νωρίτερα όμως θα συζητήσουμε το θέμα της σειριακής επικοινωνίας μεταξύ Arduino και άλλων συσκευών, π.χ. του υπολογιστή με τον οποίο ο Arduino συνδέεται μέσω καλωδίου USB.

Παρατήρηση : *Ο Arduino αναγνωρίζει ως λογικό 1 (HIGH) σε μια είσοδό του οποιαδήποτε τάση μεγαλύτερη από 3V, και ως λογικό 0 (LOW) οποιαδήποτε τάση μικρότερη από 1,5V.*

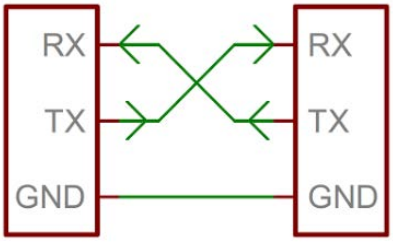
Σειριακή επικοινωνία μεταξύ Arduino και υπολογιστή ή άλλων συσκευών

Η σειριακή επικοινωνία μεταξύ δύο συσκευών είναι μια διαδικασία αποστολής δεδομένων διαδοχικά -ένα bit τη φορά- μέσω ενός διαύλου, ο οποίος αποτελείται από δύο αγωγούς: ένα για την αποστολή και ένα για τη λήψη δεδομένων. Αντίστοιχα οι συνδεδεμένες συσκευές πρέπει να διαθέτουν δύο σειριακές ακίδες:

- RX για τη λήψη δεδομένων
- TX για την αποστολή δεδομένων

Η επιτυχής σειριακή επικοινωνία δύο συσκευών προϋποθέτει:

1. Σταυρωτή σύνδεση των ακίδων RX και TX των δύο συσκευών, δηλ. η ακίδα RX της μιας συσκευής στην ακίδα TX της άλλης και αντίστροφα (Εικόνα 15).
2. Ίδιο ρυθμό επικοινωνίας (baud rate σε bits per second ή bps) στις δύο συσκευές.
3. Ίδια επίπεδα τάσης στις σειριακές ακίδες των δύο συσκευών, δηλ. δε μπορούμε να συνδέσουμε απευθείας τις σειριακές ακίδες του Arduino (επίπεδα τάσης 5V TTL) στις αντίστοιχες ακίδες μιας σειριακής θύρας υπολογιστή RS232 (που έχει επίπεδα τάσης 12V TTL).



Εικόνα 15: Σειριακή σύνδεση δύο συσκευών

Έχουμε ήδη χρησιμοποιήσει -χωρίς να το αναφέρουμε- τις δυνατότητες σειριακής επικοινωνίας του Arduino με τον υπολογιστή: Με τον τρόπο αυτό το Arduino IDE "ανεβάζει" τα μεταγλωττισμένα sketch στον Arduino. Μάλιστα κάποιος προσεκτικός πιθανόν να έχει ήδη παρατηρήσει πως, κατά τη διάρκεια του ανεβάσματος του sketch, τα LED Rx και Tx της πλακέτας του Arduino αναβοσβήνουν.

Στο υλικό του μικροελεγκτή του Arduino Uno συμπεριλαμβάνεται μια μονάδα (USART) που εγγενώς υποστηρίζει τη σειριακή επικοινωνία (Harware Serial), και υλοποιείται μέσω των ψηφιακών ακίδων 0 και 1: Τα δεδομένα λαμβάνονται από τον Arduino μέσω της ακίδας 0 (RX) και στέλνονται απ' αυτόν μέσω της ακίδας 1 (TX), και για το λόγο αυτό, όταν το sketch μας χρησιμοποιεί τη δυνατότητα σειριακής επικοινωνίας, οι ακίδες 0 και 1 δεν είναι διαθέσιμες (δηλ. δε μπορούν να χρησιμοποιηθούν από το πρόγραμμά μας) ως ψηφιακές είσοδοι ή έξοδοι.

Αν θέλουμε να χρησιμοποιήσουμε στο sketch μας τη δυνατότητα σειριακής επικοινωνίας του Arduino με τον υπολογιστή (και ουσιαστικά να χρησιμοποιήσουμε τη σειριακή κονσόλα του Arduino IDE ως τη "διαδραστική" οθόνη που δε διαθέτει ο Arduino), θα πρέπει πρώτα να εκκινήσουμε την επικοινωνία και να καθορίσουμε το ρυθμό επικοινωνίας.

Για παράδειγμα η εντολή:

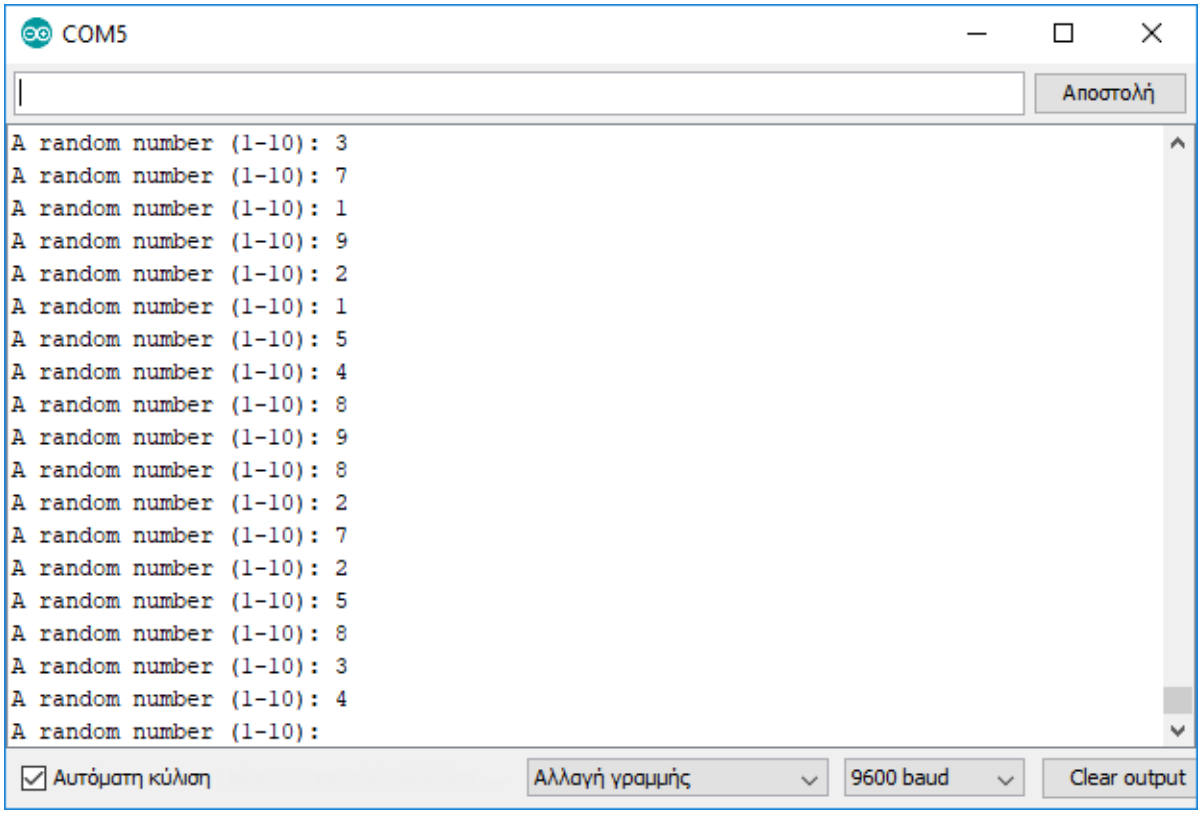
Serial.begin(9600);

ξεκινάει τη σειριακή επικοινωνία με ρυθμό 9600 bps, και πρέπει να συμπεριλαμβάνεται στη συνάρτηση *setup()* του sketch μας. Για να επιτευχθεί σωστή επικοινωνία πρέπει απαραιτήτως και οι δύο πλευρές (ο Arduino από τη μια, και ο υπολογιστής από την άλλη) να χρησιμοποιούν τον ίδιο ρυθμό επικοινωνίας (baud rate) π.χ. 9600 bps (bits per second).

Η αποστολή δεδομένων από τον Arduino στον υπολογιστή γίνεται συνήθως (αλλά όχι μόνο) μέσω των εντολών (συναρτήσεων):

- **Serial.print()**
- **Serial.println()**

Πρόκειται για δύο αρκετά ευέλικτες εντολές, καθώς μπορούμε με αυτές να τυπώσουμε δεδομένα στη σειριακή κονσόλα ως συνδυασμό αλφαριθμητικών χαρακτήρων. Εξ ορισμού οι δεκαδικοί αριθμοί τυπώνονται με δύο δεκαδικά ψηφία. Η διαφορά μεταξύ των δύο εντολών είναι πως με τη δεύτερη, αφού τυπωθούν τα δεδομένα στη σειριακή κονσόλα, ο δρομέας μετακινείται στην επόμενη γραμμή της σειριακής κονσόλας.



Εικόνα 16: Το παράθυρο της σειριακής κονσόλας του Arduino IDE

Παραδείγματα:

- Serial.print("Hello!!!"); *// Τυπώνει Hello!!! στη σειριακή κονσόλα*
- Serial.print(value); *// Τυπώνει στη σειρ. κονσόλα την τιμή της μεταβλητής value*
- Serial.print(value,1); *// Τυπώνει στη σειρ. κονσόλα την τιμή της μεταβλητής value*
  // με ακρίβεια ενός δεκαδικού ψηφίου
- Serial.print(78, HEX) *// Τυπώνει τον αριθμό 78 σε δεκαεξαδική μορφή ("4E")*
- Serial.print("\t") *// Στέλνει στη σειρ. κονσόλα το χαρακτήρα ελέγχου tab*

Μπορούμε να διαβάσουμε ένα χαρακτήρα από τη σειριακή θύρα χρησιμοποιώντας την εντολή **Serial.read()**, ενώ με τη **Serial.available()** ελέγχουμε αν και πόσα σειριακά δεδομένα υπάρχουν έτοιμα προς λήψη.

Παραδείγματα:

- Serial.print(Serial.available()); *// Τυπώνει στη σειρ. κονσόλα τον αριθμό των bytes*
  // που είναι διαθέσιμα για λήψη
- int c = Serial.read(); *// Διαβάζει ένα byte που έχει αποσταλεί στη σειριακή*
  // θύρα και το αποθηκεύει σε μια ακέραιου τύπου
  // (int) μεταβλητή

Για τη σύνταξη των σχετικών συναρτήσεων μπορούμε να πάρουμε περισσότερες πληροφορίες στην ιστοσελίδα Arduino Language Reference (https://www.arduino.cc/reference/en/).

Σημείωση: *Οι συναρτήσεις που σχετίζονται με τη σειριακή επικοινωνία Arduino-υπολογιστή περιέχονται σε μια βιβλιοθήκη κώδικα (library) του Arduino με το όνομα Serial. Η συμπερίληψη αυτής της βιβλιοθήκης στα προγράμματά μας γίνεται αυτόματα, χωρίς δηλαδή να απαιτείται οποιαδήποτε πρόσθετη ενέργεια από εμάς. Οι κλήσεις συναρτήσεων που περιέχονται σε βιβλιοθήκες γίνεται με τη μορφή:* ***(όνομα βιβλιοθήκης).(όνομα συνάρτησης)*** *και ακολουθούν οι όποιες παράμετροι δέχεται η συνάρτηση.*

Στο παράδειγμα που ακολουθεί, τυπώνεται στη σειριακή κονσόλα (οθόνη) ένας τυχαίος αριθμός που υπολογίζεται με βάση την ενσωματωμένη στο λογισμικό του Arduino γεννήτρια ψευδοτυχαίων αριθμών:

```
/*
        Ex.5 : Παράδειγμα χρήσης σειριακής κονσόλας
*/


void setup()
{
        Serial.begin(9600);                     // Ενεργοποίηση σειριακής επικοινωνίας στα 9600 bps
}

void loop()
{
        int value = random(1,10);               // Παραγωγή ψευδοτυχαίου αριθμού μεταξύ 1 και 10
        Serial.print("A random number (1-10): "); // Εκτύπωση μηνύματος στη σειριακή κονσόλα
        Serial.println(value);                  // Εκτύπωση του ψευδοτυχαίου αριθμού
        delay (1000);                           // Αναμονή 1 sec
}
```

Σημείωση: *Οι αριθμοί που παράγει η συνάρτηση random() χαρακτηρίζονται ως ψευδοτυχαίοι, αφού ο χρησιμοποιούμενος αλγόριθμος παράγει μια σειρά φαινομενικά τυχαίων αλλά στην πραγματικότητα πλήρως καθορισμένων αριθμών. Δηλαδή κάθε φορά που "τρέχει" το πρόγραμμα, παράγει την ίδια ακριβώς σειρά "τυχαίων" αριθμών. Χρησιμοποιώντας στη συνάρτηση setup() του sketch μας τη συνάρτηση randomSeed(seed) μπορούμε να παράγουμε μια διαφορετική σειρά ψευδοτυχαίων αριθμών. Αν όμως η τιμή της μεταβλητής seed είναι σταθερή, τότε και αυτή η σειρά αριθμών θα είναι πλήρως καθορισμένη. Η λύση για να παίρνουμε πραγματικά τυχαίους αριθμούς κάθε φορά που εκτελείται το λογισμικό είναι να χρησιμοποιούμε κάθε φορά τυχαία τιμή για τη μεταβλητή seed.*

Όπως έχουμε ήδη αναφέρει ο μικροελεγκτής ATMega 328 διαθέτει στο υλικό του μία μόνο μονάδα USART και συνεπώς μπορεί να επικοινωνεί με μία μόνο σειριακή συσκευή. Για την περίπτωση που θέλουμε ο Arduino Uno να επικοινωνεί σειριακά με περισσότερες από μία συσκευές, μπορούμε να χρησιμοποιούμε την εξωτερική βιβλιοθήκη *SoftwareSerial*, η οποία αναπαράγει μέσω λογισμικού τη λειτουργία μιας σειριακής θύρας σε οποιεσδήποτε άλλες ψηφιακές ακίδες του Arduino.

Λήψη απόφασης

Όταν θέλουμε στο sketch μας να ληφθεί μια απόφαση ανάλογα με κάποια συνθήκη (και συνεπώς να εκτελεστούν οι αντίστοιχες εντολές) χρησιμοποιούμε την εντολή ***if***, η οποία στην ποιο απλή της μορφή συντάσσεται ως εξής:

```
if (συνθήκη)
{
        // εντολές που εκτελούνται αν η συνθήκη είναι αληθής
}
```

Τα δύο άγκιστρα *{…}* καθορίζουν το μπλοκ των εντολών που θα εκτελεστούν αν η συνθήκη που ακολουθεί την εντολή *if* είναι αληθής. Στο επόμενο παράδειγμα:

```
if (value == 5)
{
        Serial.println("value is equal to 5");
}
```

το σύμβολο "==" (διπλό =) που χρησιμοποιήσαμε είναι ένας *τελεστής σύγκρισης* (βλέπε και Πίνακα 1) και χρησιμοποιείται εδώ για τη σύγκριση της μεταβλητής value με την τιμή 5: Δηλαδή η συνθήκη (value == 5) είναι αληθής αν η μεταβλητή *value* έχει τιμή ίση με 5, και τότε εκτελείται η εντολή Serial.println("value is equal to 5"), ενώ είναι ψευδής σε οποιαδήποτε άλλη περίπτωση, και το πρόγραμμα συνεχίζει με την πρώτη εντολή αμέσως μετά το μπλοκ εντολών του *if*.

Σημείωση: *Παρότι στην περίπτωση του παραδείγματός μας (όπου μόνο μία εντολή εκτελείται όταν η συνθήκη της εντολής if είναι αληθής) τα δύο άγκιστρα θα μπορούσαν να παραληφθούν, είναι καλή τακτική (τουλάχιστον στη φάση αυτή) να μην το κάνουμε.*

Πίνακας 1: Τελεστές σύγκρισης

!=	Δεν είναι ίσο με
<	Μικρότερο από
<=	Μικρότερο από ή ίσο με
>	Μεγαλύτερο από
>=	Μεγαλύτερο από ή ίσο με
==	Ίσο με

Ένας ελαφρά πολυπλοκότερος τρόπος σύνταξης της εντολής ***if*** προκύπτει με το συνδυασμό της με την εντολή ***else***, ως εξής:

```
if (συνθήκη)
{
        // εντολές που εκτελούνται αν η συνθήκη είναι αληθής
}
else
{
        // εντολές που εκτελούνται αν η συνθήκη είναι ψευδής
}
```

Ένα απλό παράδειγμα αποτελεί ο συνδυασμός εντολών:

```
if (value == HIGH)
{
```

```
        Serial.println(" value is HIGH");       // Εκτελείται όταν η μεταβλητή value είναι HIGH
}
else
{
        Serial.println(" value is LOW");        // Εκτελείται όταν η μεταβλητή value ΔΕΝ είναι HIGH
}
```

Πολυπλοκότερες συνθήκες ελέγχου μπορούν να υλοποιηθούν με χρήση των λογικών τελεστών (βλέπε και Πίνακα 2). Για παράδειγμα με την εντολή:

```
if ((val1 == HIGH) && (val2 == LOW))
```

γίνεται έλεγχος αν η μεταβλητή *val1* έχει τιμή 1 (HIGH) και ταυτόχρονα η μεταβλητή *val2* έχει τιμή 0 (LOW).

Πίνακας 2: Λογικοί τελεστές

!	Λογική άρνηση (NOT)
&&	Λογική σύζευξη (AND)
||	Λογική διάζευξη (OR)

Επιπλέον ***if*** μπορούν να συνδυαστούν με την ***else***, δίνοντας τη δυνατότητα λήψης πολυπλοκότερων αποφάσεων, όπως στο ακόλουθο παράδειγμα:

```
if (value < 5)                  // Αν η μεταβλητή value  έχει τιμή μικρότερη από 5
{
        Serial.println("value is less than 5");
}
else if (value > 5)             // Αλλιώς, αν η μεταβλητή value  έχει τιμή μεγαλύτερη από 5
{
        Serial.println("value is greater than 5");
}
else                            // Σε άλλη περίπτωση, δηλ. αν value = 5
{
        Serial.println("value is equal to 5");
}
```

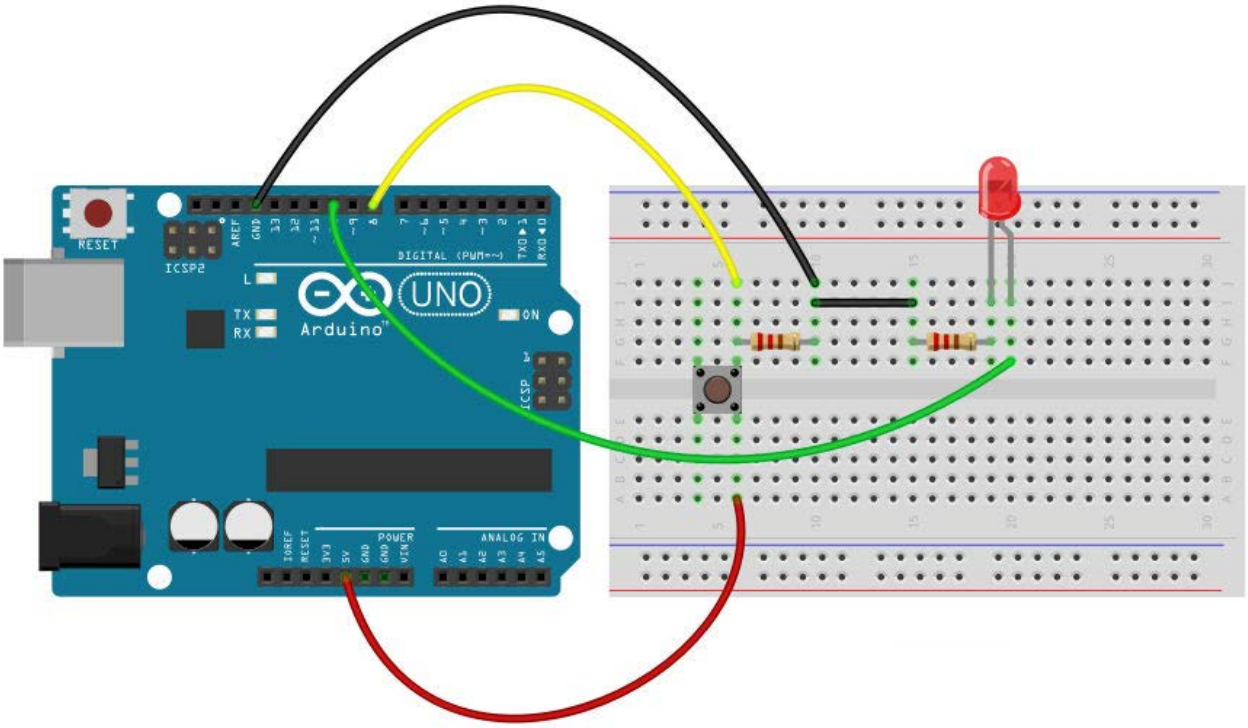
Επανερχόμενοι στο παράδειγμα όπου ένας διακόπτης συνδέεται σε μια αναλογική είσοδο του Arduino (Εικόνες 13 &14), ας δεχτούμε πως θέλουμε, όταν πατάμε το διακόπτη, να τυπώνει στη σειριακή κονσόλα το μήνυμα "HIGH", ενώ όταν ο διακόπτης έχει απελευθερωθεί, να τυπώνει το μήνυμα "LOW". Το σχετικό sketch μπορεί να πάρει τη μορφή:

```
/*
        Ex.6 : Ανάγνωση ψηφιακής ακίδας του Arduino και λήψη απόφασης
*/

#define BUTTONPIN 7

void setup ()
{
        Serial.begin(9600);                     // Εκκίνηση σειρ. επικοινωνίας στα 9600 bps
```

```
        pinMode(BUTTONPIN, INPUT);              // Η ακίδα 7 καθορίζεται ως ψηφιακή είσοδος
}

void loop ()
{
        int value = digitalRead(BUTTONPIN);     // Διάβασε την κατάσταση του διακόπτη

        if (value == HIGH)                      // Ελέγχει αν ο διακόπτης είναι κλειστός
        {
                Serial.println("Switch is closed");
        }
        else                                    // Αν ο διακόπτης δεν είναι κλειστός
        {
                Serial.println("Switch is opened");
        }

        delay(1000);                            // Αναμονή 1sec
}
```

Βρόχοι επανάληψης

Ας υποθέσουμε πως έχοντας συνδέσει ένα διακόπτη σε μια ψηφιακή είσοδο του Arduino θέλουμε κάθε φορά που κλείνουμε το διακόπτη να αναβοσβήνει πέντε φορές, και με συγκεκριμένο ρυθμό, ένα LED που είναι συνδεδεμένο σε μια άλλη ψηφιακή ακίδα (έξοδο) του Arduino. Οι συνδέσεις του Arduino με το διακόπτη και το LED φαίνονται στο επόμενο σχήμα:



Εικόνα 17: LED και διακόπτης push button στον Arduino

Έχουμε ήδη δει σε προηγούμενα παραδείγματα:

- Πώς να ορίσουμε μια ακίδα του Arduino ως ψηφιακή είσοδο ή έξοδο:

```
pinMode (7, INPUT);
pinMode (8, OUTPUT);
```

- Πώς να "διαβάζουμε" την κατάσταση ενός διακόπτη:

```
int value = digitalRead (7);
```

- Πώς να παίρνουμε αποφάσεις με βάση μια συνθήκη:

```
if (value == HIGH)
{
        // Εντολές που εκτελούνται αν η συνθήκη είναι αληθής
}
```

- Πώς να αναβοσβήσουμε μια φορά το LED:

```
digitalWrite (8, HIGH);      // Άναψε το LED
delay (200);                 // Αναμονή 200 milliseconds
digitalWrite (8, LOW);       // Σβήσε το LED
delay (200);                 // Αναμονή 200 milliseconds
```

Είναι φανερό πως, αφού θέλουμε το LED να αναβοσβήσει πέντε φορές, οι τέσσερεις γραμμές κώδικα με τις οποίες το λαμπάκι αναβοσβήνει πρέπει να εκτελεστούν συνολικά πέντε φορές. Μια απλοϊκή προσέγγιση είναι να επαναλάβουμε αυτές τις τέσσερεις γραμμές συνολικά πέντε φορές στο

σώμα της συνάρτησης *loop()* του sketch μας. Όμως η γλώσσα προγραμματισμού του Arduino διαθέτει εξειδικευμένες εντολές για τη δημιουργία βρόχων επανάληψης, δηλ. όταν θέλουμε ένα μέρος του κώδικα να επαναληφθεί κάποιες φορές.

α. Ο βρόχος for (for loop)

Όταν είναι γνωστός εκ των προτέρων ο αριθμός των επαναλήψεων του κώδικα, μπορούμε να χρησιμοποιήσουμε την εντολή ***for***, η σύνταξη της οποίας έχει την ακόλουθη μορφή:

```
for  (εντολή αρχικοποίησης; συνθήκη; εντολή επανάληψης)
{
        // Εντολές που πρέπει να εκτελεστούν
}
```

Ένα απλό παράδειγμα θα διευκρινίσει τη χρήση της εντολής:

```
for  (int i = 0; i < 5; i++)
{
        Serial.println(i);
}
```

Κατά την εκτέλεση του βρόχου:

1. Με την εντολή αρχικοποίησης (*int i = 0*) ορίζεται μια τοπική ακέραια μεταβλητή απαρίθμησης με το όνομα *"i"* και της δίνεται η αρχική τιμή 0.
2. Μετά συγκρίνεται η τιμή της μεταβλητής αυτής με την τιμή 5, και αν ικανοποιείται η συνθήκη "*i < 5*" τυπώνεται σε μια γραμμή της σειριακής κονσόλας η τιμή της μεταβλητής "*i*" (εντολή *Serial.println…*).
3. Εν συνεχεία η τιμή της μεταβλητής "*i*" αυξάνεται κατά ένα (αυτό είναι το νόημα της εντολής επανάληψης *i++*).
4. Ο βρόχος επαναλαμβάνεται από το βήμα (2) και κάτω. Όταν όμως η μεταβλητή "*i*" γίνει ίση με 5, η συνθήκη "*i < 5*" γίνεται ψευδής, οπότε το sketch εξέρχεται από το βρόχο επανάληψης και συνεχίζει την εκτέλεση με την πρώτη εντολή αμέσως μετά το μπλοκ εντολών του βρόχου.

Το αποτέλεσμα είναι η επανάληψη όλων των εντολών του βρόχου για συνολικά πέντε φορές (δηλ. από *i = 0* μέχρι και *i = 4*). Αν τώρα στο προηγούμενο παράδειγμα την εντολή *Serial.println(i)* την αντικαταστήσουμε με τις τέσσερεις γραμμές κώδικα που κάνουν το λαμπάκι να αναβοσβήνει, έχουμε πετύχει το στόχο μας.

Το σχετικό sketch διαμορφώνεται ως εξής:

```
/*
        Ex.7 : Παράδειγμα βρόχου επανάληψης με την εντολή for
*/

#define BUTTONPIN       8
#define LEDPIN          10

void setup ()
{
        pinMode(LEDPIN,    OUTPUT);          // Η ακίδα 10 καθορίζεται ως ψηφιακή έξοδος
        pinMode(BUTTONPIN, INPUT);           // Η ακίδα 8 καθορίζεται ως ψηφιακή είσοδος
}
```

```
void loop ()
{
        int value = digitalRead(BUTTONPIN);             // Διάβασε την κατάσταση του διακόπτη

        if (value == HIGH)                              // Ελέγχει αν ο διακόπτης είναι κλειστός
        {
                for (int i = 0; i < 5; i++)             // Βρόχος επανάληψης
                {
                        digitalWrite(LEDPIN, HIGH);     // Άναψε το LED
                        delay(200);                     // Αναμονή 200 milliseconds
                        digitalWrite(LEDPIN, LOW);      // Σβήσε το LED
                        delay(200);                     // Αναμονή 200 milliseconds
                }
        }
}
```

β. Ο βρόχος while (while loop)

Με χρήση του βρόχου *while* επιτυγχάνουμε την εκτέλεση ενός συγκεκριμένου μπλοκ εντολών ενόσω μια λογική συνθήκη είναι αληθής. Για παράδειγμα το προηγούμενο sketch μπορεί να διαμορφωθεί ως εξής:

```
/*
        Ex.8 : Παράδειγμα βρόχου επανάληψης με την εντολή while
*/

#define BUTTONPIN 8
#define LEDPIN    10

void setup ()
{
        pinMode(LEDPIN,OUTPUT);         // Η ακίδα 10 καθορίζεται ως ψηφιακή έξοδος
        pinMode(BUTTONPIN, INPUT);      // Η ακίδα 8 καθορίζεται ως ψηφιακή είσοδος
}

void loop ()
{
        int cnt = 0;                                    // Ορισμός μεταβλητής απαριθμητή
        int value = digitalRead(BUTTONPIN);             // Διάβασε την κατάσταση του διακόπτη

        if (value == HIGH)                              // Ελέγχει αν ο διακόπτης είναι κλειστός
        {
                while (cnt < 5)                         // Βρόχος επανάληψης while
                {
                        // Το μπλοκ των εντολών εκτελείται ενόσω
                        // ο απαριθμητής έχει τιμή μικρότερη από 5

                        digitalWrite(LEDPIN, HIGH);     // Άναψε το LED
                        delay(200);                     // Αναμονή 200 milliseconds
                        digitalWrite(LEDPIN, LOW);      // Σβήσε το LED
                        delay(200);                     // Αναμονή 200 milliseconds
                        cnt++;                          // Αύξηση κατά 1 του απαριθμητή
                }
```

```
        }
}
```

Και σ' αυτή την περίπτωση ορίσαμε μια ακέραια μεταβλητή απαρίθμησης με το όνομα *cnt*, της δώσαμε την αρχική τιμή μηδέν, ενώ μέσα στο σώμα εντολών του βρόχου *while*, η τιμή της αυξάνεται κατά 1 (εντολή *cnt++*). Η μεταβλητή αυτή είναι τοπική στη συνάρτηση *loop()*, δηλ. έχει ισχύ μόνο εντός της συνάρτησης, τόσο εντός όσο και εκτός του βρόχου *while {…}*.

Η χρήση του βρόχου *while* ενδείκνυται όταν δεν είναι εκ των προτέρων γνωστός ο αριθμός των επαναλήψεων. Για παράδειγμα με την εντολή:

```
while (! Serial.available());
```

που έχει το ίδιο αποτέλεσμα με την:

```
while (Serial.available() == 0) ;
```

δημιουργείται ένας βρόχος αναμονής, που συνεχίζει δηλ. να εκτελείται ενόσω δεν υπάρχουν δεδομένα προς λήψη μέσω της σειριακής θύρας.

Ιδιαίτερου ενδιαφέροντος είναι και η δημιουργία ατέρμονα βρόχου (infinite or endless loop) με χρήση των εντολών [5]:

```
while (true) ;   ή   while (1) ;
```

Μια παραλλαγή του βρόχου *while* έχει τη μορφή:

```
do
{
        // Εντολές του βρόχου
}
while (συνθήκη)
```

Η διαφορά μεταξύ των δύο μορφών του βρόχου *while* είναι πως με την πρώτη μορφή υπάρχει η περίπτωση (όταν εξ' αρχής η συνθήκη είναι αληθής) οι εντολές του βρόχου να μην εκτελεστούν ούτε μία φορά, ενώ με τη δεύτερη μορφή θα εκτελεστούν οπωσδήποτε μία φορά.

Στοιχεία δομημένου προγραμματισμού

Η λειτουργικότητα και αναγνωσιμότητα του κώδικα μπορεί να βελτιωθεί με χρήση τεχνικών δομημένου προγραμματισμού, όπως για παράδειγμα ο χωρισμός του προβλήματος σε ανεξάρτητα υποπροβλήματα, τα οποία προγραμματιστικά μπορούν να επιλύονται με ανεξάρτητα τμήματα κώδικα, υλοποιούμενα από συναρτήσεις οριζόμενες από το χρήστη. Το πρόγραμμα μέσω κατάλληλων μεταβλητών (ορίσματα) μπορεί να μεταφέρει μέσα στη συνάρτηση κάποιες τιμές απαραίτητες για τη λειτουργία της. Επιπλέον οι συναρτήσεις μπορεί να επιστρέφουν κάποια τιμή στο πρόγραμμα από το οποίο καλούνται. Στα προγράμματά μας ο ορισμός μιας συνάρτησης έχει τη μορφή:

```
Επιστρεφόμενος_τύπος  Όνομα_συνάρτησης ( δηλώσεις ορισμάτων )
{
   // Σώμα της συνάρτησης
   .................
   return τιμή;     // Μη απαραίτητη
}
```

Επανερχόμενοι στο προηγούμενο πρόβλημα (δηλ. το αναβόσβημα 5 φορές ενός LED με το πάτημα ενός διακόπτη, βλέπε και Εικόνα 17), μια τέτοια νέα συνάρτηση θα μπορούσε να αναλάβει την υλοποίηση του αναβοσβησίματος του LED, ως εξής:

```
void blink()    // Ορισμός συνάρτησης blink()
{
      digitalWrite (LEDPIN, HIGH);      // Άναψε το LED
      delay (200);                      // Αναμονή 200 milliseconds
      digitalWrite (LEDPIN, LOW);       // Σβήσε το LED
      delay (200);                      // Αναμονή 200 milliseconds
}
```

Στη νέα συνάρτηση δόθηκε το περιγραφικό όνομα *blink*, και εφόσον δεν επιστρέφει κάποια τιμή στον Arduino ο τύπος ορίστηκε της ως *void (= χωρίς τύπο)*.

Το αντίστοιχο sketch διαμορφώνεται ως:

```
/*
      Ex.9 : Παράδειγμα δομημένου προγραμματισμού
*/

#define BUTTONPIN         8
#define LEDPIN            10

void setup ()
{
      pinMode(LEDPIN, OUTPUT);              // Η ακίδα 10 καθορίζεται ως ψηφιακή έξοδος
      pinMode(BUTTONPIN, INPUT);            // Η ακίδα 8 καθορίζεται ως ψηφιακή είσοδος
}

void loop ()
{
      int value = digitalRead(BUTTONPIN);   // Διάβασε την κατάσταση του διακόπτη
      if (value == HIGH)                    // Ελέγχει αν ο διακόπτης είναι κλειστός
      {
```

```
        for (int i = 0; i <5; i++)              // Επανάληψη 5 φορές
        {
            blink();                            // Αναβοσβήνει το Led
        }
    }
}

void blink()                                    // Συνάρτηση για αναβόσβημα του led
{
    digitalWrite(LEDPIN, HIGH);                 // Άναψε το LED
    delay(200);                                 // Αναμονή 200 milliseconds
    digitalWrite(LEDPIN, LOW);                  // Σβήσε το LED
    delay(200);                                 // Αναμονή 200 milliseconds
}
```

Μια ακόμη πιο ευέλικτη εκδοχή της συνάρτησης *blink* και του αντίστοιχου sketch, δίνεται αμέσως μετά:

```
/*
    Ex.10 : Δεύτερο παράδειγμα δομημένου προγραμματισμού
*/

#define BUTTONPIN 8
#define LEDPIN    10

void setup ()
{
    pinMode(LEDPIN, OUTPUT);        // Η ακίδα 10 καθορίζεται ως ψηφιακή έξοδος
    pinMode(BUTTONPIN, INPUT);      // Η ακίδα 8 καθορίζεται ως ψηφιακή είσοδος
}

void loop ()
{
    int value = digitalRead(BUTTONPIN);     // Διάβασε την κατάσταση του διακόπτη
    if (value == HIGH)                      // Ελέγχει αν ο διακόπτης είναι κλειστός
    {
        blink(LEDPIN, 200, 5);              // Αναβοσβήνει το Led
    }
}

void blink(int ledpin, int dtime, int count)    // Συνάρτηση για αναβόσβημα του led
{
    for (int i = 0; i < count; i++)
    {
        digitalWrite(ledpin, HIGH);         // Άναψε το LED
        delay(dtime);                       // Αναμονή
        digitalWrite(ledpin, LOW);          // Σβήσε το LED
        delay(dtime);                       // Αναμονή
    }
}
```

Στη νέα εκδοχή της συνάρτησης *blink* ο βρόχος για συγκεκριμένο αριθμό επαναλήψεων έχει μεταφερθεί εντός της συνάρτησης, η οποία δέχεται τρεις ακέραιου (*int*) τύπου παραμέτρους, ως εξής:

- *ledpin* : Είναι η ακίδα του Arduino στην ποία έχει συνδεθεί το LED
- *dtime* : Το χρονικό διάστημα σε msec που το LED παραμένει αναμμένο ή σβηστό
- *count* : Ο αριθμός επαναλήψεων του αναβοσβησίματος

Μια συνάρτηση οριζόμενη από το χρήστη μπορεί επιπλέον να επιστρέφει κάποια τιμή στο πρόγραμμα από το οποίο καλείται. Ο επιστρεφόμενος τύπος μιας τέτοιας συνάρτησης κατά τον ορισμό της πρέπει να είναι του ίδιου τύπου με τη μεταβλητή που επιστρέφει. Για παράδειγμα η επόμενη συνάρτηση δέχεται ως παράμετρο μια ακέραια μεταβλητή και επιστρέφει το (ακέραιου τύπου) τετράγωνό της:

```
int square(int x)
{
      return x*x;
}
```

Παρατήρηση : *Σε ένα απλό πρόγραμμα C/C++ οι οριζόμενες από το χρήστη συναρτήσεις πρέπει να δηλώνονται στην αρχή του προγράμματος, ώστε στη συνέχεια να μπορούν να καλούνται από το κύριο σώμα του προγράμματος. Κάτι τέτοιο δεν είναι απαραίτητο στη γλώσσα προγραμματισμού του Arduino.*

Ανάγνωση αναλογικής εισόδου

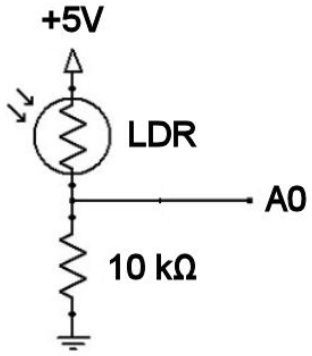
Στον μικροελεγκτή ATmega 328 του Arduino Uno έξι διαφορετικές αναλογικές είσοδοι, μέσω της μεθόδου της πολυπλεξίας, οδηγούνται στην είσοδο ενός μετατροπέα αναλογικού σε ψηφιακό (A/D C) διακριτικής ικανότητας 10bit [3]. Αυτό σημαίνει ότι με τις εξ ορισμού ρυθμίσεις ο μετατροπέας αντιστοιχίζει αναλογικές τάσεις μεταξύ 0 και 5V σε ακέραιες τιμές μεταξύ 0 και 1023, το οποίο οδηγεί σε διακριτική ικανότητα: 5V/1024 βήματα, ή περίπου 4.9 mV/βήμα.

Η περιοχή της τάσης που μπορούν να διαχειριστούν οι αναλογικές είσοδοι, και συνεπώς και η αντίστοιχη διακριτική ικανότητα, μπορούν, εφόσον είναι επιθυμητό, να ρυθμιστούν μέσω της εντολής *analogReference()*. Απαιτείται η σύνδεση της επιθυμητής τάσης αναφοράς (V_{REF}) στην είσοδο *AREF* του Arduino, η οποία όμως σε καμία περίπτωση δε μπορεί να υπερβεί τα 5V.

Με τις εξ ορισμού ρυθμίσεις του λογισμικού του ο Arduino χρειάζεται λίγο περισσότερο από 100 μs για να ολοκληρώσει μια μετατροπή αναλογικού σε ψηφιακό, γεγονός που περιορίζει σε κάτι λιγότερο από 10.000 το (θεωρητικό) πλήθος των μετρήσεων αναλογικών μεγεθών που μπορούν να ληφθούν κάθε δευτερόλεπτο.

Το αποτέλεσμα μιας αναλογικο-ψηφιακής μετατροπής σε κάποια αναλογική είσοδο του Arduino μπορούμε να "διαβάσουμε" με την εντολή *analogRead()* [4], η οποία δέχεται ως παράμετρο ην αναλογική είσοδο (A0 μέχρι A5) στην οποία η προς μέτρηση τάση έχει συνδεθεί.

Για τις ανάγκες της επόμενης άσκησης θα χρησιμοποιήσουμε μια φωτοαντίσταση (LDR Light Dependent Resistor) σε συνδεσμολογία διαιρέτη τάσης με μια αντίσταση σταθερής τιμής 10kΩ. Το σύστημα των δύο αντιστάσεων τροφοδοτείται από σταθερή τάση 5V, ενώ η αναλογική είσοδος A0 του Arduino τροφοδοτείται με την τάση που αναπτύσσεται στα άκρα της σταθερής αντίστασης.



Εικόνα 18: Διαιρέτης τάσης με φωτοαντίσταση

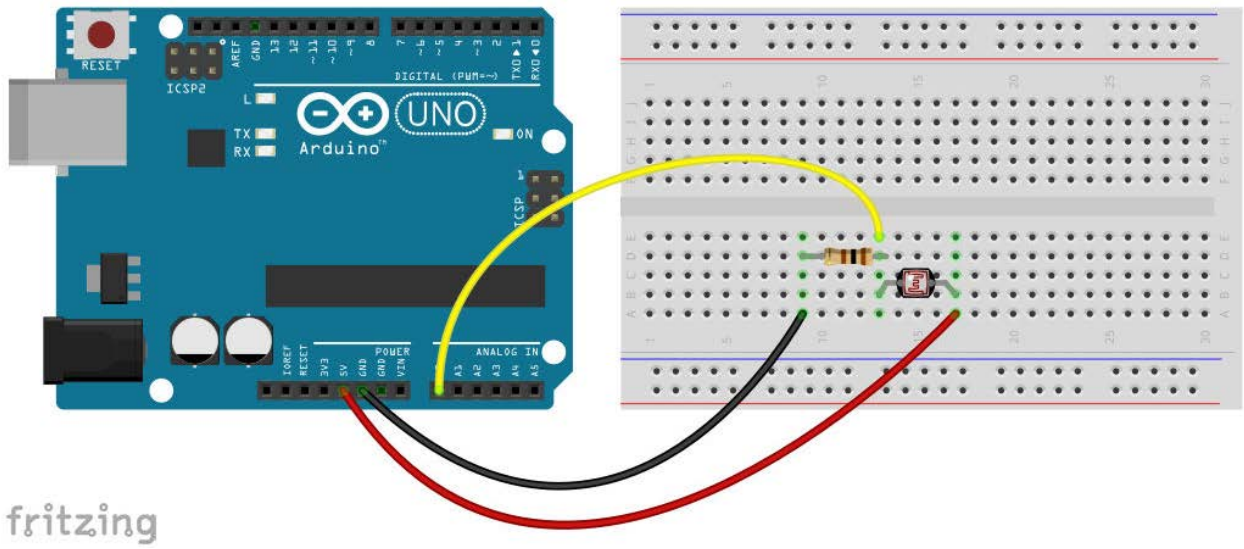
Αν συμβολίσουμε με R_x την αντίσταση της φωτοαντίστασης, εύκολα μπορούμε να αποδείξουμε πως ο λόγος της τάσης στα άκρα της φωτοαντίστασης προς την τάση στα άκρα της σταθερής αντίστασης R = 10 kΩ είναι:

$$\frac{V_x}{V_R} = \frac{R_x}{R}$$

Δηλαδή, χαρακτηριστικό του διαιρέτη τάσης είναι ότι διαιρεί την τάση τροφοδοσίας του σε επιμέρους τάσεις με τιμές ανάλογες των αντίστοιχων αντιστάσεων.

Χαρακτηριστικό της φωτοαντίστασης είναι πως η αντίστασή της μεταβάλλεται σε σχέση με την ένταση του φωτός που πέφτει πάνω της: Έχει πολύ μεγάλη αντίσταση (της τάξης των ΜΩ) στο σκοτάδι, η οποία μειώνεται σημαντικά καθώς αυξάνεται η ένταση του φωτός [6]. Συνεπώς όταν μεταβάλλεται η ένταση του φωτός που πέφτει στη φωτοαντίσταση, μεταβάλλεται αντίστοιχα και η τιμή της τάσης στα άκρα της σταθερής αντίστασης των 10 kΩ. Μετρώντας λοιπόν την τάση αυτή παίρνουμε μια ένδειξη της έντασης του φωτός που πέφτει στη φωτοαντίσταση.

Η σύνδεση του διαιρέτη τάσης στον Arduino (από τον οποίο και τροφοδοτείται) φαίνεται στην επόμενη εικόνα:



Εικόνα 19: Σύνδεση φωτοαντίστασης στον Arduino

Το σχετικό sketch διαμορφώνεται ως εξής:

```
/*
        Ex.11 : Μετατροπή αναλογικού σε ψηφιακό
*/

#define ANALOGPIN          A0

void setup ()
{
        Serial.begin(9600);                     // Εκκίνηση σειρ. επικοινωνίας στα 9600 bps
        //pinMode(ANALOGPIN, INPUT);            // Καλή τακτική, όχι όμως απαραίτητη
}

void loop ()
{
        int value = analogRead(ANALOGPIN);      // Μετατροπή αναλογικού σε ψηφιακό, και
                                                // αποθήκευση αποτελέσματος στην
                                                // ακέραια μεταβλητή value

        float voltage = getVoltage(value);      // Μετατροπή σε τάση
        Serial.print("Input Voltage is: ");     // Αποστολή του αποτελέσματος στη
        Serial.print(voltage);                  // σειριακή κονσόλα
        Serial.println(" V");

        delay (1000);                           // Αναμονή 1sec
}

float getVoltage(int value)
{
        return value*5.0 / 1024.0;
}
```

Η οριζόμενη από το χρήστη συνάρτηση *getVoltage()* δέχεται ως παράμετρο την ακέραια τιμή μιας

αναλογικοψηφιακής μετατροπής (ADC *value*) και επιστρέφει την αναλογική τάση στην οποία η τιμή αυτή αντιστοιχεί. Η μετατροπή γίνεται με βάση το γεγονός πως ο αναλογικοψηφιακός μετατροπέας του Arduino έχει διακριτική ικανότητα (5/1024) V/βήμα. Δηλαδή είναι:

$$V = (ADC\ value) \times \left(\frac{5}{1024} \right) \quad \text{ή γενικότερα} \quad V = (ADC\ value) \times \left(\frac{V_{REF}}{1024} \right)$$

όπου V_{REF} είναι η τάση αναφοράς του αναλογικοψηφιακού μετατροπέα.

Προσοχή πρέπει να δοθεί στο γεγονός ότι το πηλίκο 5/1024 στη γλώσσα προγραμματισμού του Arduino επιστρέφει ακέραια τιμή (διαίρεση ακεραίων). Για να πάρουμε αποτέλεσμα τύπου κινητής υποδιαστολής (*float*) το πηλίκο πρέπει να γραφεί ως 5.0/1024.0.

Χρήση προεγκατεστημένης βιβλιοθήκης

Όπως έχουμε ήδη αναφέρει μαζί με το ολοκληρωμένο περιβάλλον εργασίας του Arduino εγκαθίστανται και κάποιες βιβλιοθήκες, οι οποίες δε συμπεριλαμβάνονται εξ' ορισμού στα sketches που δημιουργούμε. Τέτοια είναι και η βιβλιοθήκη **Servo**, που μας επιτρέπει να εκμεταλλευτούμε τις δυνατότητες των σερβοκινητήρων.

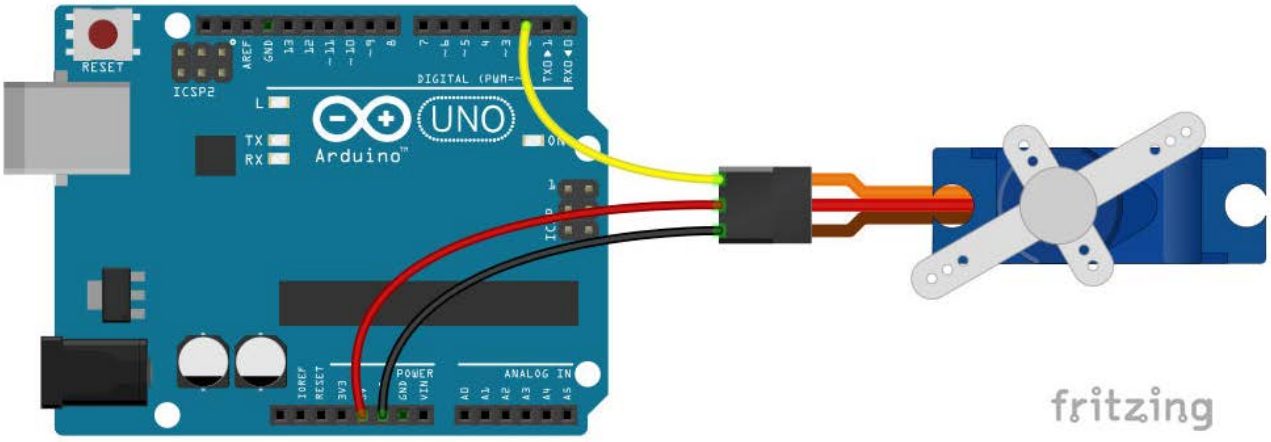
Εικόνα 20: Σερβοκινητήρας

Ένας σερβοκινητήρας αποτελείται από ένα μικρό ηλεκτρικό κινητήρα, ένα συνδυασμό γραναζιών ώστε να μπορεί να περιστρέφεται με μικρή ταχύτητα αλλά με μεγάλη ροπή, και ένα σύστημα για τον ακριβή έλεγχο της θέσης του άξονά του. Συνήθως το εύρος περιστροφής του άξονα περιορίζεται μεταξύ 0 και 180 μοιρών (μισή περιστροφή).

Η βιβλιοθήκη *Servo* χρησιμοποιώντας τεχνικές αντικειμενοστραφούς προγραμματισμού μας επιτρέπει τη δημιουργία αντικειμένων κλάσης *Servo* για το χειρισμό μέχρι και δώδεκα (12) σερβοκινητήρων συνδεδεμένων στις ακίδες του Arduino Uno. Οι βασικές συναρτήσεις της βιβλιοθήκης, είναι:

- *attach(pin)* : Δημιουργεί μια σύναψη μεταξύ ενός σερβοκινητήρα και μιας I/O ακίδας του Arduino
- *write(pos)* : Περιστρέφει τον άξονα του σερβοκινητήρα σε θέση που καθορίζεται σε μοίρες από την παράμετρο *pos*.
- *read()* : Επιστρέφει την τρέχουσα γωνία του άξονα του σερβοκινητήρα (δηλ. την τιμή της παραμέτρου της τελευταίας κλήσης της συνάρτησης *write()*).

Ένας σερβοκινητήρας συνδέεται στον Arduino, όπως φαίνεται στην επόμενη εικόνα:



Εικόνα 21 : Σύνδεση σερβοκινητήρα στον Arduino

Για να αξιοποιήσουμε τις δυνατότητές του σερβοκινητήρα πρέπει να συμπεριλάβουμε στο sketch μας τη σχετική βιβλιοθήκη. Αυτό μπορεί να γίνει μέσω του μενού "*Σχέδιο/Συμπερίληψη βι-*

βλιοθήκης", και επιλογή της βιβλιοθήκης "*Servo*". Το λογισμικό ανταποκρίνεται εισάγοντας στην αρχή του sketch τη σχετική εντολή συμπερίληψης:

```
#include <Servo.h>
```

Εννοείται πως μπορούμε να γράψουμε και χειροκίνητα τη σχετική εντολή συμπερίληψης στην αρχή του sketch. Το επόμενο βήμα είναι να δημιουργήσουμε ένα αντικείμενο κλάσης *Servo*, για να αποκτήσουμε πρόσβαση στις διαθέσιμες εντολές διαχείρισης του σερβοκινητήρα.

Ας σημειώσουμε στο σημείο αυτό πως οι βιβλιοθήκες συνήθως συνοδεύονται από έτοιμα παραδείγματα, με τα οποία διασαφηνίζεται ο τρόπος χρήσης της βιβλιοθήκης. Στο επόμενο sketch, που αποτελεί παραλλαγή ενός από τα έτοιμα παραδείγματα της βιβλιοθήκης *Servo*, κάνοντας χρήση των επιλογών και δυνατοτήτων που η βιβλιοθήκη προσφέρει, περιστρέφουμε τον άξονα του κινητήρα παλινδρομικά μεταξύ 60° και 120° με βήμα περιστροφής 1 μοίρα.

```
/*
        Ex.12 : Έλεγχος σερβοκινητήρα
*/

#include <Servo.h>

Servo myservo;              // Δημιουργία αντικειμένου κλάσης Servo
int pos = 0;                // Μεταβλητή για την αποθήκευση της θέσης του σερβοκινητήρα

void setup()
{
        myservo.attach(2);  // Ο σερβοκινητήρας συνδέεται στην ακίδα 2 του Arduino
        myservo.write(60);  // Η αρχική θέση του σερβοκινητήρα καθορίζεται στις 60°
}

void loop()
{
        // Περιστροφή από τις 60 μέχρι τις 120 μοίρες με βήμα 1 μοίρα
        for (pos = 60; pos <= 120; pos += 1)
        {
                myservo.write(pos);   // Ενημερώνει τον κινητήρα να μεταβεί σε θέση "pos"
                delay(10);            // Αναμονή για να ολοκληρωθεί η περιστροφή
        }
        // Περιστροφή από τις 120 μέχρι τις 60 μοίρες με βήμα 1 μοίρα
        for (pos = 120; pos >= 60; pos -= 1)
        {
                myservo.write(pos);   // Ενημερώνει τον κινητήρα να μεταβεί σε θέση "pos"
                delay(10);            // Αναμονή για να ολοκληρωθεί η περιστροφή
        }
}
```

Ας σημειώσουμε μια ενδιαφέρουσα παραλλαγή του βρόχου επανάληψης *for* στο συγκεκριμένο sketch:

```
        for (pos = 120; pos >= 60; pos -= 1)
        {
                // Εντολές που πρέπει να εκτελεστούν
        }
```

Ο βρόχος εκτελείται με αρχική τιμή 120° στη μεταβλητή *pos*, η οποία μετά κάθε εκτέλεση των εντολών του βρόχου μειώνεται με βήμα 1 (αυτό είναι το νόημα της εντολής pos -= 1, δες και το Παράρτημα Β), και μέχρι η τιμή της μεταβλητής *pos* να γίνει μικρότερη από 60°. Ο βρόχος :

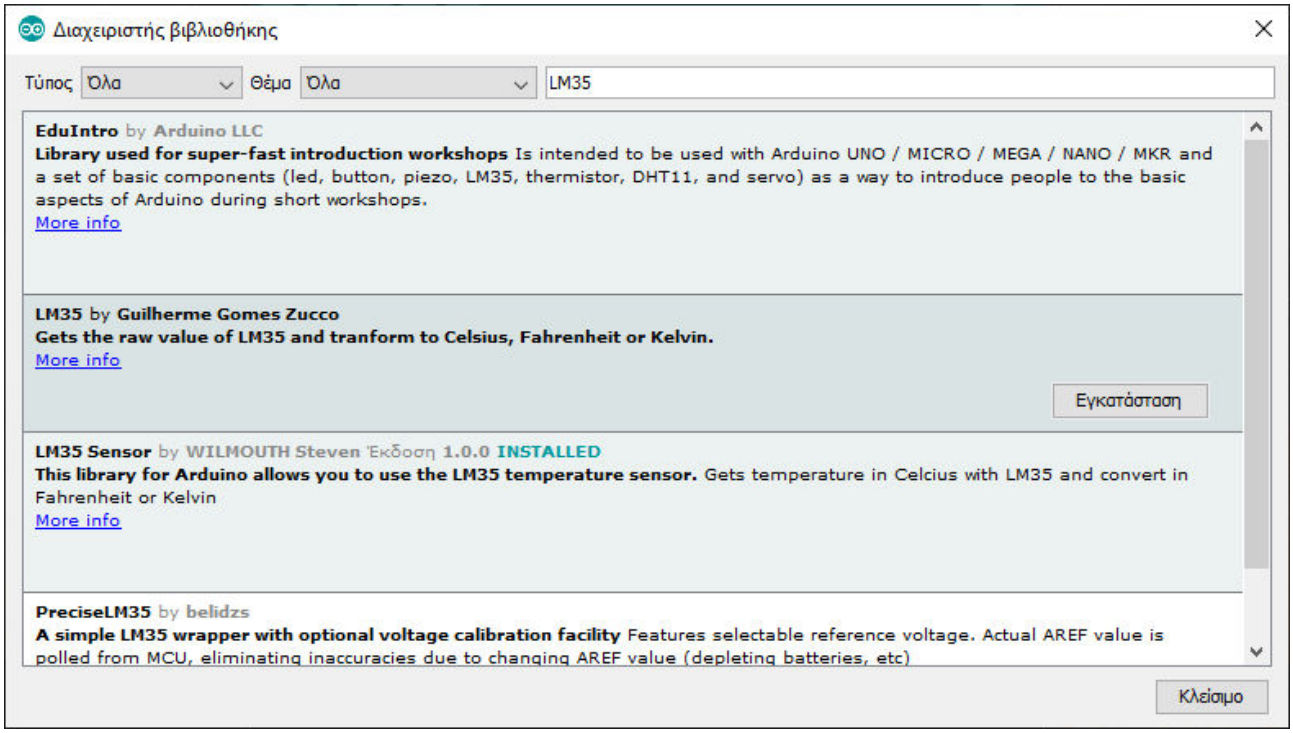
```
for (pos = 120; pos >= 60; pos -= 2)
{
        // Εντολές που πρέπει να εκτελεστούν
}
```

έχει το ίδιο αποτέλεσμα αλλά με βήμα μείωσης 2, δηλ. οδηγεί σε μικρότερο αριθμό επαναλήψεων και τελικά στην περίπτωση του σερβοκινητήρα σε γρηγορότερη περιστροφή του άξονά του.

Εγκατάσταση και χρήση εξωτερικής βιβλιοθήκης

Πολλές φορές, για παράδειγμα όταν θέλουμε να συνδέσουμε στον Arduino ένα αισθητήρα ή μια οθόνη υγρών κρυστάλλων, κ.λ.π., είναι απαραίτητη η επέκταση των δυνατοτήτων του Arduino με τη χρήση επιπλέον βιβλιοθηκών, οι οποίες δεν είναι εξ ορισμού εγκατεστημένες.

Ο απλούστερος τρόπος να εγκαταστήσουμε μια βιβλιοθήκη είναι μέσω του *"Διαχειριστή βιβλιοθήκης"* του Arduino IDE. Το παράθυρο *"Διαχειριστής βιβλιοθήκης"* ανοίγει μέσω του μενού *"Εργαλεία/Διαχείριση βιβλιοθηκών..."* ή "*Σχέδιο/Συμπερίληψη βιβλιοθήκης/Διαχείριση βιβλιοθηκών...*". Με βάση τις διάφορες επιλογές που διαθέτει ο διαχειριστής βιβλιοθήκης μπορούμε να αναζητήσουμε και να εγκαταστήσουμε όποια από τις διαθέσιμες βιβλιοθήκες θέλουμε, π.χ. τη βιβλιοθήκη του Steven Wilmouth για τον αναλογικού τύπου αισθητήρα θερμοκρασίας LM35.



Εικόνα 22 : Διαχειριστής βιβλιοθήκης

Σε έναν υπολογιστή με Windows λειτουργικό σύστημα τα αρχεία που συναποτελούν μια βιβλιοθήκη εγκαθίστανται από το διαχειριστή βιβλιοθηκών σε ιδιαίτερο φάκελο μέσα στο φάκελο *"Εγγραφα\Arduino\libraries"*.

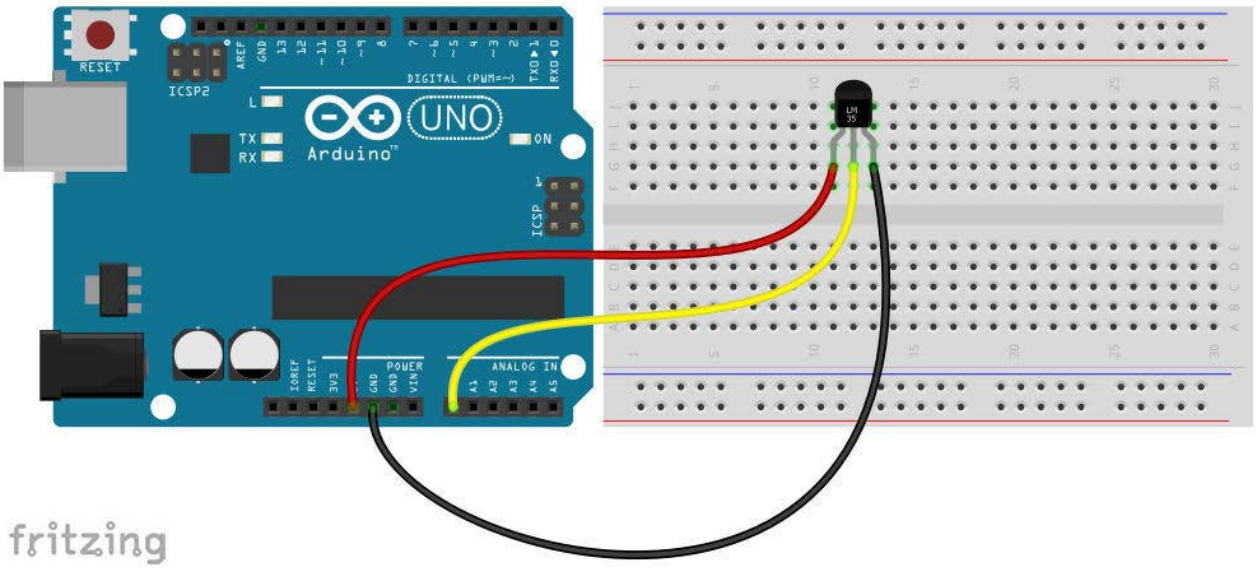
Οι βιβλιοθήκες στην απλούστερη μορφή τους αποτελούνται από ένα αρχείο επικεφαλίδων και ορισμών (με κατάληξη ".h"), και συνήθως ακόμη ένα αρχείο κώδικα γραμμένο σε C ή C++. Στην περίπτωση της βιβλιοθήκης LM35_Sensor το αρχείο επικεφαλίδων (LM35.h) έχει τη μορφή:

```
#ifndef LM35_h
#define LM35_h

#include "Arduino.h"

enum Unity
{
	CELCIUS,
	KELVIN,
	FAHRENHEIT
};
```

```
class LM35
{
	public:
		LM35(int analogPin);
		double getTemp();
		double getTemp(Unity unity);
	private:
		int _analogPin;
};

#endif
```

Η συγκεκριμένη βιβλιοθήκη ακολουθεί τεχνικές αντικειμενοστραφούς προγραμματισμού, και περιέχει όλο τον κώδικα που απαιτείται για τη δημιουργία και τη χρήση ενός αντικειμένου κλάσης LM35 (class LM35), μέσω του οποίου αποκτάμε πρόσβαση στις δυνατότητες μέτρησης της θερμοκρασίας με το συγκεκριμένο αισθητήρα. Χωρίς να μπούμε σε περισσότερες λεπτομέρειες, θα πούμε μόνο πως στο τμήμα public της κλάσης περιέχονται οι επικεφαλίδες των συναρτήσεων που είναι διαθέσιμες στο χρήστη, ενώ στο τμήμα private περιέχονται μεταβλητές και συναρτήσεις που δεν είναι διαθέσιμες στον χρήστη, αλλά χρησιμοποιούνται εσωτερικά από την κλάση.

Η σύνδεση του αισθητήρα LM35 στον Arduino, φαίνεται στο σχήμα που ακολουθεί:



Εικόνα 23: Σύνδεση αισθητήρα θερμοκρασίας LM35

Μετά την εγκατάσταση της βιβλιοθήκης, και αφού δημιουργήσουμε ένα νέο sketch (μέσω του μενού "*Αρχείο/Δημιουργία*" του Arduino IDE), μπορούμε να συμπεριλάβουμε τη βιβλιοθήκη μέσω του μενού "*Σχέδιο/Συμπερίληψη βιβλιοθήκης*", και επιλογή της βιβλιοθήκης "*LM35 Sensor*". Το λογισμικό ανταποκρίνεται εισάγοντας στην αρχή του sketch τη σχετική εντολή συμπερίληψης:

```
#include <LM35.h>
```

Για να εκμεταλλευτούμε τις δυνατότητες που προσφέρει η βιβλιοθήκη, πρέπει πρώτα να δημιουργήσουμε ένα αντικείμενο κλάσης (τύπου) LM35, με κάποιο όνομα της επιλογής μας π.χ. lm35temp, και να ορίσουμε τις αρχικές παραμέτρους που απαιτούνται για την ορθή λειτουργία του. Στην περίπτωσή μας η μόνη παράμετρος που απαιτείται είναι η αναλογική ακίδα στην οποία ο αισθητήρας συνδέεται. Ο σχετικός κώδικας θα έχει τη μορφή:

```
LM35 lm35temp(A0);
```

Με τη συνάρτηση *getTemp()* της βιβλιοθήκης γίνεται η ανάγνωση της τιμής της θερμοκρασίας που

επιστρέφει ο αισθητήρας. Το σχετικό sketch τελικά διαμορφώνεται ως εξής:

```
/*
        Ex.13 : Αισθητήρας θερμοκρασίας LM35 και χρήση της σχετικής βιβλιοθήκης
*/

#include <LM35.h>                   // Συμπερίληψη βιβλιοθήκης LM35

LM35 lm35temp(A0);                  // Ορισμός και αρχικοποίηση αντικειμένου τύπου LM35

void setup()
{
        Serial.begin(9600);                     // Εκκίνηση σειρ. επικοινωνίας στα 9600 bps
}

void loop()
{
        Serial.println(lm35temp.getTemp());     // Θερμοκρασία σε βαθμούς Κελσίου
        delay(1000);
}
```

Στο αρχείο επικεφαλίδων της κλάσης LM35 διαπιστώνουμε την ύπαρξη μιας δεύτερης μορφής της συνάρτησης *getTemp*, που στο sketch μας θα μπορούσε να κληθεί π.χ. ως εξής:

```
float tempF = lmd35temp.getTemp(FAHRENHEIT);
```

επιστρέφοντας τη θερμοκρασία σε βαθμούς Fahrenheit (ή σε Kelvin ή βαθμούς Celsius ανάλογα με την τιμή της παραμέτρου που θα χρησιμοποιήσουμε).

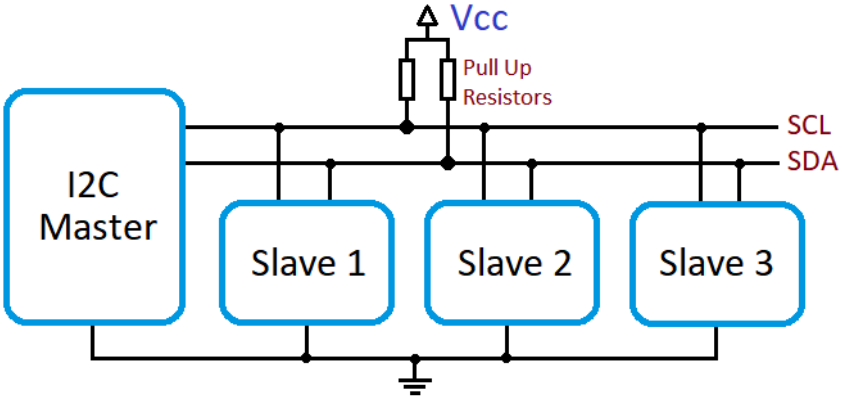
Παρατήρηση : *Υπάρχουν δύο ακόμη τρόποι εγκατάστασης μιας εξωτερικής βιβλιοθήκης στο Arduino IDE, και αφορούν την περίπτωση που από το διαδίκτυο έχουμε "κατεβάσει" τον κώδικα στον υπολογιστή μας, συνήθως με τη μορφή ενός συμπιεσμένου αρχείου τύπου "zip":*

1. *Αυτόματα μέσω του μενού "Σχέδιο/Συμπερίληψη βιβλιοθήκης/Προσθήκη βιβλιοθήκης ZIP…", και επιλογή του συμπιεσμένου αρχείου που περιέχει τα αρχεία της βιβλιοθήκης.*
2. *Χειροκίνητα αποσυμπιέζοντας στο φάκελο "Έγγραφα\Arduino\libraries\" το συμπιεσμένο αρχείο.*

Αν η βιβλιοθήκη δεν είναι διαθέσιμη για συμπερίληψη μέσω του μενού "Σχέδιο/Συμπερίληψη βιβλιοθήκης" θα χρειαστεί να κάνουμε επανεκκίνηση του Arduino IDE.

Χρήση του δισύρματου (I²C) σειριακού διαύλου επικοινωνίας

Ο δίαυλος I²C είναι ένας σειριακός δίαυλος επικοινωνίας μεταξύ διαφόρων συσκευών, ο οποίος για τη μεταφορά των δεδομένων χρησιμοποιεί μόνο δύο καλώδια (αγώγιμες γραμμές): Τη γραμμή SCL που είναι η γραμμή χρονισμού, και τη γραμμή SDA που είναι η γραμμή δεδομένων. Στις γραμμές αυτές συνδέονται όλες οι συσκευές, που υπάρχουν στο δίαυλο I²C. Προφανώς εκτός από τους παραπάνω αγωγούς που μεταφέρουν δεδομένα, απαιτούνται και οι κατάλληλο αγωγοί τροφοδοσίας των συσκευών. Για τη σωστή λειτουργία του διαύλου επιβάλλεται οι δύο γραμμές δεδομένων να προσδεθούν στη θετική τροφοδοσία μέσω κατάλληλων (pull up) αντιστάσεων [7]. Στην υλοποίηση του διαύλου στον Arduino χρησιμοποιούνται οι εσωτερικές pull up αντιστάσεις (της τάξης των 50 kΩ) που υπάρχουν στις ακίδες εισόδου/εξόδου.



Εικόνα 24: Ο δίαυλος I²C

Στο δίαυλο I²C κάθε συνδεδεμένη συσκευή αναγνωρίζεται με βάση ένα μοναδικό κωδικό (διεύθυνση) εύρους συνήθως 7 bit (ή και 10bit). Μ' αυτό τον τρόπο ο μέγιστος θεωρητικά αριθμός συσκευών που μπορούν να συνδεθούν στο δίαυλο είναι 128 (για διευθυνσιοδότηση 7bit) ή 1024 (για διευθυνσιοδότηση 10bit). Βέβαια άλλοι παράγοντες, όπως π.χ. η συνολική χωρητικότητα του διαύλου, περιορίζουν σημαντικά τον αριθμό αυτό. Οι συσκευές στον δίαυλο I²C μπορεί να είναι είτε *κύριες (masters)*, είτε *υποτελείς (slaves)*. Και οι *κύριες* και οι *υποτελείς* συσκευές μπορούν να μεταφέρουν δεδομένα στον δίαυλο, αλλά μόνο οι *κύριες* ελέγχουν την μεταφορά, δημιουργώντας τους κατάλληλους παλμούς χρονισμού. Εξ ορισμού στον Arduino ο δίαυλος I²C χρονίζεται στα 100 kHz, υπάρχει όμως η δυνατότητα χρονισμού και σε υψηλότερες συχνότητες (π.χ. 400kHz).

Δε θα επεκταθούμε σε περισσότερες λεπτομέρειες σχετικά με το δίαυλο I²C. Στον Arduino το σχετικό πρωτόκολλο επικοινωνίας υλοποιείται μέσω της βιβλιοθήκης **Wire**. Για τη χρήση της βιβλιοθήκης πρέπει:

1. Πρώτα να συμπεριληφθεί ο κώδικας που περιέχει με τη δήλωση συμπερίληψης:

   #include <Wire.h>

2. Να ενεργοποιηθεί η σύνδεση στο δίαυλο I²C με την εντολή:

   Wire.begin();

   Η συνάρτηση *Wire.begin()* δέχεται ως παράμετρο τη διεύθυνση της συσκευής που όμως δεν είναι απαραίτητη προκειμένου για master συσκευή.

Η διαχείριση της επικοινωνίας με κάποια slave συσκευή γίνεται μέσω των εντολών [4]:

1. *Wire.beginTransmission(address)*: Έναρξη επικοινωνίας με τη συσκευή στη διεύθυνση *addres* στο δίαυλο.
2. *Wire.write(data)*: Στέλνει δεδομένα (*data*) από τη master συσκευή στη slave, και επιστρέφει τον αριθμό των bytes που στάλθηκαν. Τα δεδομένα μπορεί να είναι ένα byte, μια συμβολοσειρά, ή ένας πίνακας από bytes.

3. *Wire.requestFrom(address, quantity, stop)*: Χρησιμοποιείται από τη master συσκευή για να ζητήσει ορισμένο αριθμό bytes (*quantity*) από τη slave συσκευή. Η παράμετρος *stop* καθορίζει αν μετά τη λήψη των bytes ο δίαυλος θα απελευθερωθεί ή θα συνεχίσει να παραμένει ενεργός.
4. *Wire.available()*: Επιστρέφει τον αριθμό των bytes που είναι διαθέσιμα για ανάγνωση. Καλείται από μια master συσκευή μετά από κλήση της *requestFrom()*.
5. *Wire.read()*: Διαβάζει ένα byte που έχει αποσταλεί από μια slave συσκευή στη master μετά από κλήση της *requestFrom()*.
6. *Wire.endTransmission(stop)*: Λήξη της επικοινωνίας. Με αληθή (true) τιμή της παραμέτρου *stop* ο δίαυλος απενεργοποιείται μετά τη λήξη της επικοινωνίας, ενώ με ψευδή (false) ο δίαυλος παραμένει ενεργός. Η παράμετρος μπορεί ακόμη και να παραληφθεί. Η τιμή που επιστρέφει η συνάρτηση είναι μηδέν (0) αν η επικοινωνία master - slave ήταν επιτυχής, και μεγαλύτερη του 0 στην περίπτωση που κάποιο λάθος παρουσιάστηκε κατά τη διάρκεια της επικοινωνίας.

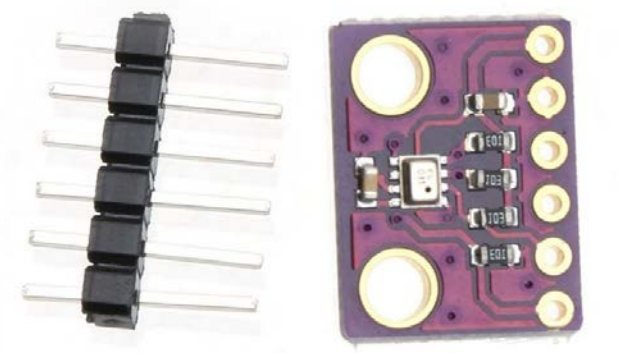
Το επόμενο sketch (που αποτελεί μια ελαφρά τροποποιημένη εκδοχή του αντίστοιχου sketch από την επίσημη ιστοσελίδα του Arduino) ελέγχει την ύπαρξη συνδεδεμένων συσκευών στο δίαυλο, και αν βρεθούν επιστρέφει τη διεύθυνσή τους.

```
/*
        Ex.14 : Ανίχνευση συνδεδεμένων συσκευών στο δίαυλο I2C
*/

#include <Wire.h>                       // Συμπερίληψη της βιβλιοθήκης Wire
int nDevices = 0;                       // Αριθμός συνδεδεμένων συσκευών στο δίαυλο

void setup()
{
        Wire.begin();                   // Ενεργοποίηση διαύλου I2C
        Serial.begin(9600);             // Ενεργοποίηση σειριακής επικοινωνίας
        Serial.println("\nI2C Scanner"); // Τυπώνει τον τίτλο του sketch
}

void loop()
{
        byte error, address;
        Serial.println("Scanning...");
        for (address = 1; address < 127; address++ )
        {
                Wire.beginTransmission(address);    // Έναρξη επικοινωνίας με τη slave συσκευή
                                                    // στη διεύθυνση address (από 1 ως 126)
                error = Wire.endTransmission();     // Λήξη της επικοινωνίας και λήψη της
                                                    // επιστρεφόμενης τιμής λάθους
                nDevices += checkErrorValue(address, error);
        }
        if (nDevices == 0)                  // Αν δε βρεθούν I2C συσκευές στο δίαυλο
```

```
            Serial.println("No I2C devices found\n");
        else                                        // Αλλιώς αν έχουν βρεθεί I2C συσκευές
            Serial.println("done\n");
        nDevices = 0;
        delay(5000);                                // Αναμονή 5 seconds
}

int checkErrorValue(byte address, int error)
{
        int ret = 0;
        // Αν η τιμή που επιστρέφει η Wire.endTransmission() είναι μηδέν
        // υπάρχει συνδεδεμένη συσκευή στη συγκεκριμένη διεύθυνση του διαύλου
        if (error == 0)
        {
            Serial.print("I2C device found at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.print(address, HEX);
            Serial.println("  !");
            ret = 1;
        }
        else if (error == 4)            // Άγνωστο λάθος στη συγκεκριμένη διεύθυνση
        {
            Serial.print("Unknown error at address 0x");
            if (address < 16)
                Serial.print("0");
            Serial.println(address, HEX);
        }
        return ret;
}
```

α. Ο αισθητήρας BMP280

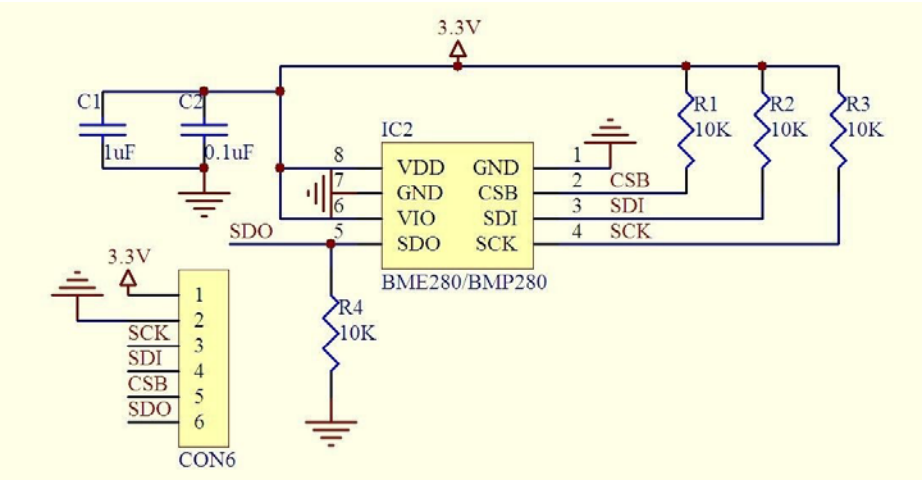
Για να ελέγξουμε το sketch θα συνδέσουμε στον Arduino μια μονάδα που ενσωματώνει τον αισθητήρα ατμοσφαιρικής πίεσης και θερμοκρασίας BMP280.



Εικόνα 25: Η μονάδα με τον αισθητήρα BMP280

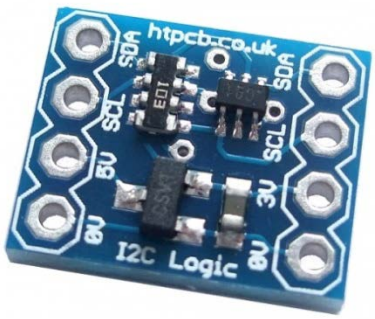
Σύμφωνα με το φυλλάδιο δεδομένων του κατασκευαστή (Bosch) ο αισθητήρας BMP280 μπορεί να λειτουργήσει με τάση τροφοδοσίας μεταξύ 1,71V και 3,6 V, και συνεπώς **ΔΕΝ είναι ασφαλές** να τον τροφοδοτήσουμε με τάση 5V. Οπότε **για την τροφοδοσία της μονάδας θα χρησιμοποιήσουμε την τάση των 3,3V από την αντίστοιχη ακίδα της πλακέτας του Arduino**. Αλλά ακόμη και έτσι υπάρχει κίνδυνος καταστροφής του αισθητήρα, αφού ο αμφίδρομος δίαυλος I^2C στον Arduino λειτουργεί στα 5V. Δηλαδή, όταν ο Arduino "διαβάζει" δεδομένα στο δίαυλο (που έχουν αποσταλεί από τον αισθητήρα), η τάση στη γραμμή δεδομένων δεν υπερβαίνει τα 3,3V (όση είναι και η τάση τροφοδοσίας του αισθητήρα), ενώ όταν ο Arduino "γράφει" δεδομένα στο δίαυλο η τάση στη γραμμή δεδομένων φτάνει μέχρι και τα 5V. Ο Arduino όμως, σύμφωνα με το φυλλάδιο δεδομένων του ATMega 328), απαιτεί τουλάχιστον 3V για ένα αξιόπιστο υψηλό (HIGH) επίπεδο στις γραμμές SDA και SCL, ενώ τάση μεγαλύτερη των 3,6V στις ίδιες γραμμές συνιστά κίνδυνο καταστροφής του αισθητήρα BMP280. Υπάρχει δηλαδή πρόβλημα προσαρμογής των τάσεων μεταξύ της master και της slave συσκευής στο δίαυλο I^2C.

Η οικονομική λύση που επέλεξε ο κατασκευαστής της μονάδας του αισθητήρα για την αντιμετώπιση του προβλήματος είναι η χρήση αντιστάσεων 10 kΩ, μέσω των οποίων οι γραμμές SCL και SDA στη μονάδα προσδένονται στην τροφοδοσία των 3,3V (αντιστάσεις pull up).



Εικόνα 26: Το σχηματικό διάγραμμα του αισθητήρα BMP280

Έτσι μέσω του συνδυασμού αυτών των αντιστάσεων με τις αντίστοιχες εσωτερικές (pull up) αντιστάσεις του Arduino, οι οποίες προσδένουν τις γραμμές δεδομένων στα 5V, δημιουργείται ουσιαστικά ένας διαιρέτης τάσης, με αποτέλεσμα η τάση στις γραμμές SDA και SCL να μην υπερβαίνει τα 3,6V. Η λύση είναι επισφαλής, αφού η σύνδεση μιας ακόμη συσκευής στο δίαυλο μπορεί να μεταβάλλει προς το χειρότερο την κατάσταση. Η απολύτως ασφαλής λύση είναι η χρήση μιας μονάδας μετατόπισης του επιπέδου της τάσης (level shifter) από τα 5V στα 3,3V, όπως αυτή που φαίνεται στην επόμενη εικόνα:



Εικόνα 27: Μονάδα αμφίδρομης μετατόπισης επιπέδου τάσης

Κατά την εκτέλεση του sketch για την ανίχνευση συνδεδεμένων συσκευών στο δίαυλο I^2C (ex.14.ino) με τη μονάδα BMP280 συνδεδεμένη στον Arduino, στη σειριακή κονσόλα παίρνουμε

την πληροφορία:

I2C device found at address 0x76 !

Θέτοντας την ακίδα SDO της μονάδας σε υψηλή λογική κατάσταση (HIGH) μπορούμε να αλλάξουμε τη διεύθυνση με την οποία αναγνωρίζεται η μονάδα στο δίαυλο I^2C σε 0x77.

Ο αισθητήρας BMP280 είναι μια πολύπλοκη ηλεκτρονική διάταξη, που μπορεί με ρυθμό μέχρι 157 Hz, να μετρήσει:

1. Πίεση μεταξύ 300 και 1100 hPa και με σχετική ακρίβεια ±0.12 hPa.
2. Θερμοκρασία μεταξύ -40 και 85 °C και με ακρίβεια ±1°C.

Μπορεί επιπλέον μέσω της μέτρησης της ατμοσφαιρικής πίεσης να χρησιμοποιηθεί και για τη μέτρηση υψομετρικών διαφορών με ακρίβεια ±1m.

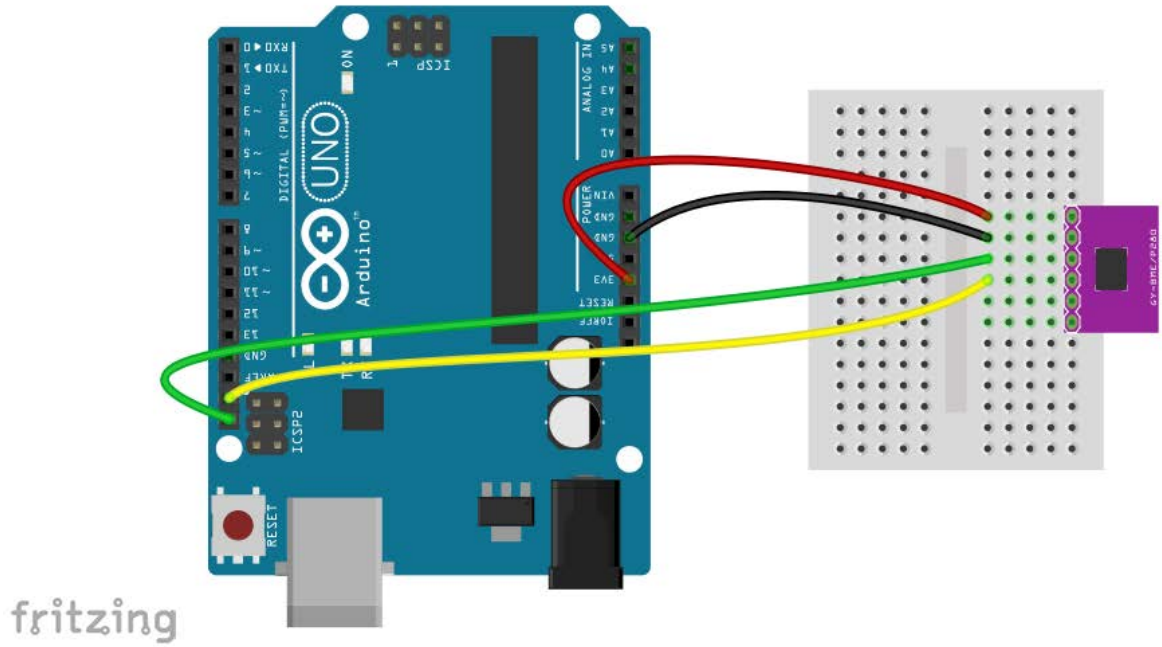
Ο αισθητήρας υποστηρίζει δύο τρόπους λειτουργίας:

α. **Εξαναγκασμένη λειτουργία:** Εκτελείται μία μέτρηση με βάση τις τρέχουσες ρυθμίσεις. Με το τέλος του κύκλου μέτρησης ο αισθητήρας μπαίνει σε κατάσταση αναμονής, και τα αποτελέσματα της μέτρησης μπορούν να διαβαστούν.

β. **Κανονική λειτουργία:** Συνεχής λειτουργία εναλλασσόμενη μεταξύ περιόδων μέτρησης και περιόδων αναμονής. Ο χρόνος αναμονής μπορεί να ρυθμιστεί μεταξύ 0,5ms και 4 s.

Στο υλικό του αισθητήρα περιλαμβάνεται και ένας μετατροπέας αναλογικού σε ψηφιακό διακριτικής ικανότητας 16-bit. Τεχνικές βελτίωσης της διακριτικής ικανότητας και μείωσης του ψηφιακού θορύβου (oversampling) του αισθητήρα μπορούν να χρησιμοποιηθούν τόσο για την πίεση, όσο και για τη θερμοκρασία, ενώ υπάρχει και η δυνατότητα χρήσης εσωτερικού (υλοποιημένου στο υλικό του αισθητήρα) φίλτρου για την εξομάλυνση των απότομων κορυφών, που προκύπτουν λόγω ανεπιθύμητων διαταραχών κατά τη διάρκεια των μετρήσεων. Επιπλέον τόσο η μέτρηση της πίεσης όσο και η μέτρηση της θερμοκρασίας μπορούν να παρακαμφθούν, με αποτέλεσμα την αύξηση του ρυθμού μέτρησης του μεγέθους που παραμένει προς μέτρηση.

Τις δυνατότητες του αισθητήρα μπορούμε να εκμεταλλευτούμε με τη χρήση της κατάλληλης βιβλιοθήκης. Θα χρησιμοποιήσουμε στη συνέχεια τη βιβλιοθήκη BME280 του Tyler Glenn, η οποία μπορεί να εγκατασταθεί μέσω του *"Διαχειριστή βιβλιοθήκης"* του Arduino IDE. Η ίδια βιβλιοθήκη υποστηρίζει και τον αισθητήρα BME280 της ίδιας εταιρείας ο οποίος διαθέτει επιπλέον δυνατότητες μέτρησης της σχετικής υγρασίας με χρόνο απόκρισης 1s και ακρίβεια περί το 3%.



Εικόνα 28: Σύνδεση της μονάδας BMP280 στον Arduino

Για τις ανάγκες των δοκιμών μας θα συνδέσουμε τη μονάδα του αισθητήρα BMP280 απευθείας (χωρίς χρήση μονάδας μετατόπισης του επιπέδου τάσης) στον Arduino, όπως φαίνεται και στην Εικόνα 28. Αυτή η επιλογή δε δημιουργεί κίνδυνο καταστροφής του αισθητήρα, αφού -προς το παρόν- δε πρόκειται να συνδέσουμε άλλον αισθητήρα στο δίαυλο I^2C.

Όπως έχουμε ήδη αναφέρει πρέπει ιδιαίτερα να προσέξουμε ότι **ο αισθητήρας τροφοδοτείται από την ακίδα των 3,3V της πλακέτας του Arduino**.

Με το επόμενο sketch, αφού γίνει πρώτα μια μέτρηση με τον αισθητήρα BMP280 και με τις εξ ορισμού ρυθμίσεις, τυπώνονται στη σειριακή κονσόλα οι τιμές της ατμοσφαιρικής πίεσης και της θερμοκρασίας περιβάλλοντος:

```
/*
        Ex.15 : BMP280 I2C Test
*/

#include <Wire.h>              // Συμπερίληψη της βιβλιοθήκης Wire
#include <BME280I2C.h>         // Συμπερίληψη της βιβλιοθήκης BME280I2C

BME280I2C bme;      // Εξ ορισμού ρυθμίσεις : forced mode, standby time = 1000 ms
                    // Oversampling = pressure ×1, temperature ×1, humidity ×1, filter off,
void setup()
{
        Serial.begin(9600);    // Ενεργοποίηση σειριακής επικοινωνίας
        Wire.begin();          // Ενεργοποίηση διαύλου I2C

        while(!bme.begin())    // Αναμονή για ανίχνευση συμβατής συσκευής
        {
                Serial.println("Could not find BMP280 sensor!");
                delay(1000);
        }
}

void loop()
{
        printBME280Data();  // Λήψη και τύπωση των αποτελεσμάτων της μέτρησης
        delay(1000);
}

void printBME280Data()
{
        float temp(NAN), pres(NAN), hum(NAN);

        bme.read(pres, temp, hum);
        Serial.print("Temperature: ");
        Serial.print(temp);
        Serial.print("°C");
        Serial.print("\t\tPressure: ");
        Serial.print(pres);
        Serial.println(" Pa");
}
```

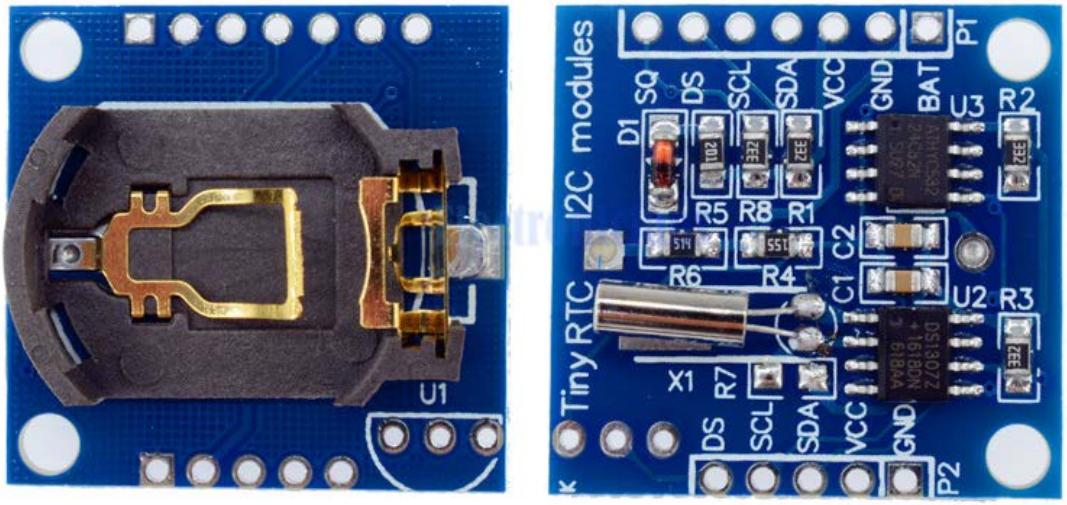
β. Ρολόι πραγματικού χρόνου (RTC) DS1307

Ένα ρολόι πραγματικού χρόνου παρέχει στον Arduino πληροφορίες για την τρέχουσα ημερομηνία και ώρα. Η πιο απλή και οικονομική λύση για χρήση ενός ρολογιού πραγματικού χρόνου στον Arduino είναι η μονάδα RTC DS1307 (Tiny RTC), που χρησιμοποιεί το σχετικό ολοκληρωμένο κύκλωμα DS1307 της Dallas Semiconducrors, στα χαρακτηριστικά του οποίου περιλαμβάνονται:

- Ρολόι πραγματικού χρόνου ακριβές μέχρι το έτος 2100.
- 56 bytes μη πτητικής μνήμης RAM διαθέσιμης στο χρήστη.
- Προγραμματιζόμενη έξοδο τετραγωνικού σήματος.
- Κύκλωμα ανίχνευσης διακοπής τροφοδοσίας, κ.ά..

Μια λίγο πιο ακριβή, αλλά και ακριβέστερη λύση, είναι η μονάδα με το ολοκληρωμένο κύκλωμα DS3231, και πάλι από τη Dallas Semiconductors.

Η μονάδα Tiny RTC επικοινωνεί με τον Arduino μέσω του σειριακού διαύλου I^2C -κάτι που απλοποιεί κατά πολύ τη σύνδεση- και αναγνωρίζεται στο δίαυλο στη δεξαεξαδική διεύθυνση 0x68.



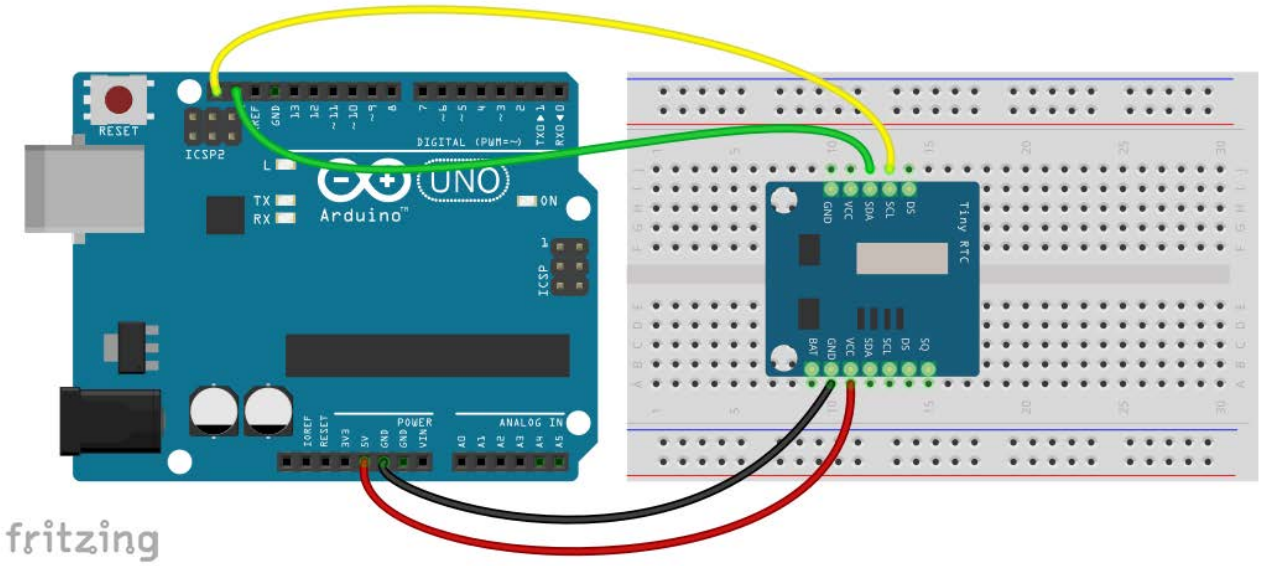
Εικόνα 29: Η μονάδα RTC DS1307 (Tiny RTC)

Εκτός από το ρολόι πραγματικού χρόνου και τα απαραίτητα για τη λειτουργία του παθητικά και ενεργά εξαρτήματα, η μονάδα Tiny RTC (Εικόνα 29) διαθέτει επιπλέον μια I^2C μνήμη EEPROM 32 kbits ή 4 kBytes (ATMEL 24C32), η οποία αναγνωρίζεται στη διεύθυνση 0x50 στο δίαυλο I^2C, ενώ έχει προβλεφθεί και η επέκταση της μονάδας με το ψηφιακό θερμόμετρο DS18B20 (περισσότερα για αυτό σε επόμενη παράγραφο). Επιπλέον διαθέτει μια μπαταριοθήκη για μπαταρίες λιθίου (τύπου CR1225 ή CR2032), για τη λειτουργία και διατήρηση των δεδομένων όταν η μονάδα δεν τροφοδοτείται από εξωτερική τάση. Η ακιδοσειρά P1 παρέχει όλες τις απαραίτητες ακίδες για την πλήρη εκμετάλλευση των δυνατοτήτων της μονάδας [8]:

- VCC : Σύνδεση της θετικής τροφοδοσίας (3,0 – 5,5V)
- GND : Γείωση
- SCL : I^2C Clock
- SDA : I^2C Data
- BAT : Για μέτρηση της τάσης της μπαταρίας
- DS : Για σύνδεση του ψηφιακού θερμομέτρου στο μονοσύρματο δίαυλο 1-Wire
- SQ : Έξοδος τετραγωνικών παλμών ρυθμιζόμενης συχνότητας

Οι ακίδες P2 της μονάδας Tiny RTC αποτελούν αντίγραφα των αντίστοιχων ακίδων της ακιδοσειράς P1. Ένα από τα μειονεκτήματα της διάταξης είναι πως η συχνότητα του ταλαντωτή που διαθέτει επηρεάζεται από εξωτερικούς παράγοντες, όπως π.χ. η θερμοκρασία με αποτέλεσμα να "χάνει" περί τα πέντε δευτερόλεπτα ανά μήνα.

Η σύνδεση της μονάδας στον Arduino γίνεται όπως φαίνεται στην Εικόνα 30 (μπορούν βέβαια να χρησιμοποιηθούν όλες οι ακίδες από μια ακιδοσειρά είτε την P1 είτε την P2):



Εικόνα 30: Σύνδεση μονάδας Tiny RTC στον Arduino

Για να εκμεταλλευτούμε τις δυνατότητες της μονάδας Tiny RTC θα χρειαστεί να εγκαταστήσουμε την κατάλληλη βιβλιοθήκη. Πρόκειται για τη βιβλιοθήκη RTCLib της Adafruit, η οποία κατά τα γνωστά μπορεί να εγκατασταθεί μέσω του *"Διαχειριστή βιβλιοθήκης"* του Arduino IDE. Το επόμενο sketch δείχνει πως μπορούμε να διαβάσουμε την τρέχουσα ημερομηνία και ώρα από τη μονάδα Tiny RTC:

```
/*
        Ex.16 : Λήψη ημερομηνίας και ώρας από τη μονάδα Tiny RTC
*/

#include <Wire.h>                       // Συμπερίληψη της βιβλιοθήκης Wire
#include <RTClib.h>                     // Συμπερίληψη της βιβλιοθήκης RTCLib

RTC_DS1307 rtc;                         // Δημιουργία αντικειμένου τύπου RTC_DS1307

void setup ()
{
        Serial.begin(9600);             // Ενεργοποίηση σειριακής επικοινωνίας
        rtc.begin();                    // Εκκίνηση επικοινωνίας με RTC μέσω I2C
}

void loop()                             // Τυπώνει ημερομηνία και ώρα στη σειριακή κονσόλα
{
        DateTime now = rtc.now();       // Λήψη τρέχουσας ημερομηνίας και ώρας

        Serial.print(now.year(), DEC);
        Serial.print('/');
        Serial.print(now.month(), DEC);
        Serial.print('/');
        Serial.print(now.day(), DEC);
```

```
        Serial.print(" ");

        Serial.print(now.hour(), DEC);
        Serial.print(':');
        Serial.print(now.minute(), DEC);
        Serial.print(':');
        Serial.println(now.second(), DEC);

        delay(1000);                                    // Αναμονή 1 sec
}
```

Πριν την πρώτη χρήση της μονάδας Tiny RTC, και αφού έχουμε τοποθετήσει την κατάλληλη μπαταρία στη μπαταριοθήκη, πρέπει να ορίσουμε τη σωστή ημερομηνία και το σωστό χρόνο για την έναρξη λειτουργίας της μονάδας. Τον τρόπο που μπορούμε να το κάνουμε, δείχνουμε στο επόμενο sketch:

```
/*
        Ex.17 : Ρύθμιση ημερομηνίας και ώρας στη μονάδα Tiny RTC
*/

#include <Wire.h>                       // Συμπερίληψη της βιβλιοθήκης Wire
#include <RTClib.h>                     // Συμπερίληψη της βιβλιοθήκης RTCLib

RTC_DS1307 rtc;                         // Δημιουργία αντικειμένου τύπου RTC_DS1307

void setup ()

{
        Serial.begin(9600);             // Ενεργοποίηση σειριακής επικοινωνίας
        rtc.begin();                    // Εκκίνηση επικοινωνίας με RTC μέσω I2C

        // Η επόμενη γραμμή θέτει το ρολόι στην ημερομηνία και ώρα μεταγλώττισης του sketch
        rtc.adjust(DateTime(__DATE__, __TIME__));
}
void loop()

{
        // Τίποτα δε χρειάζεται να γίνει εδώ
}
```

Οι μεταβλητές περιβάλλοντος του Arduino IDE __DATE__ και __TIME__ κρατούν την ημερομηνία και την ώρα που έγινε η μεταγλώττιση του sketch. Μεταξύ μεταγλώττισης και εκτέλεσης του κώδικα μεσολαβεί το ανέβασμα του κώδικα στον Arduino, και συνεπώς με αυτό το sketch το ρολόι πραγματικού χρόνου βρίσκεται εκτός συγχρονισμού κατά μισό τουλάχιστον λεπτό. Η συνάρτηση *rtc.adjust()* δέχεται μια παράμετρο τύπου DateTime (που επίσης ορίζεται στη βιβλιοθήκη RTCLib) και ρυθμίζει την ημερομηνία και την ώρα στο ρολόι πραγματικού χρόνου. Το αντικείμενο τύπου DateTime δέχεται μια ημερομηνία και ώρα με πολλούς διαφορετικούς τρόπους και επιστρέφει μέσω κατάλληλων συναρτήσεων επτά επιμέρους στοιχεία: έτος, μήνας, ημέρα του μήνα, ημέρα της εβδομάδας, ώρα, λεπτά και δευτερόλεπτα. Έτσι αν θέλουμε να ορίσουμε στο RTC την ημερομηνία ως 1/1/2019 και την ώρα ως 14:00:00, μπορούμε να χρησιμοποιήσουμε την εντολή:

```
rtc.adjust(DateTime(2019,1,1,14,0,0));        // Format: YYYY,MM,DD,HH,MM,SS
```

Το επόμενο sketch αποτελεί έναν τρόπο για ακριβέστερο ορισμό της ημερομηνίας και της ώρας στο

ρολόι πραγματικού χρόνου. Τα επιμέρους στοιχεία (έτος, μήνας, ημέρα, κ.λ.π.) εισάγονται κατά την εκτέλεση του κώδικα μέσω της σειριακής κονσόλας.

```
/*
        Ex.18 : Ακριβής ρύθμιση ημερομηνίας και ώρας στο DS1307
*/

#include <Wire.h>                       // Συμπερίληψη της βιβλιοθήκης Wire
#include <RTClib.h>                     // Συμπερίληψη της βιβλιοθήκης RTCLib

RTC_DS1307 rtc;                         // Δημιουργία αντικειμένου τύπου RTC_DS1307

void setup ()

{
        Serial.begin(115200);           // Ενεργοποίηση σειριακής επικοινωνίας
        Serial.setTimeout(30000);       // Ορισμός χρόνου αναμονής για εισαγωγή
                                        // χαρακτήρων από τη σειριακή κονσόλα
        rtc.begin();                    // Εκκίνηση επικοινωνίας με RTC μέσω I2C

        // Με το επόμενο μπλοκ εντολών γίνεται λήψη δεδομένων ημερομηνίας και ώρας
        // που έχουν αποσταλεί μέσω της σειριακής κονσόλας
        int yy = input("Enter the year.... ", 2000, 2050, 2000);
        int mm = input("Enter the month... ",1,12,1);
        int dd = input("Enter the day..... ",1,31,1);
        int hh = input("Enter hour..... ",0,23,0);
        int mn = input("Enter mins..... ",0,59,0);
        int ss = input("Enter secs..... ",0,59,0);

        // Με το επόμενο μπλοκ εντολών γίνεται εκκαθάριση της σειριακής buffer
        while (Serial.available() > 0)  // Αν υπάρχουν διαθέσιμα σειριακά δεδομένα
        {
                char t = Serial.read();  // διάβασέ τα, ώστε η buffer να αδειάσει
        }

        Serial.println();               // Τυπώνει μια κενή γραμμή

        /*
        Με το επόμενο μπλοκ εντολών τυπώνεται ένα μήνυμα οδηγιών στη σειριακή κονσόλα
        και δημιουργείται ένας βρόχος αναμονής μέχρι να αποσταλεί σειριακά ένας χαρακτήρας.
        Στη συνέχεια ο χαρακτήρας διαβάζεται και αν είναι ο 'T' ρυθμίζεται η ώρα
        και η ημερομηνία στο RTC με βάση τα δεδομένα που έχουν ήδη εισαχθεί.
        */
        Serial.println("Send T to set Datetime, or any other key to Cancel...");

        while (Serial.available() == 0);  // Αναμονή μέχρι να αποσταλεί ένας χαρακτήρας
        char inChar = Serial.read();      // Διάβασμα του χαρακτήρα από τη buffer
        if (inChar == 'T')                // Αν ο χαρακτήρας είναι ο 'T'
        {
                rtc.adjust(DateTime(yy,mm,dd,hh,mn,ss));        // Ρύθμιση του RTC
        }
        Serial.println();                 // Τυπώνει μια κενή γραμμή
}
```

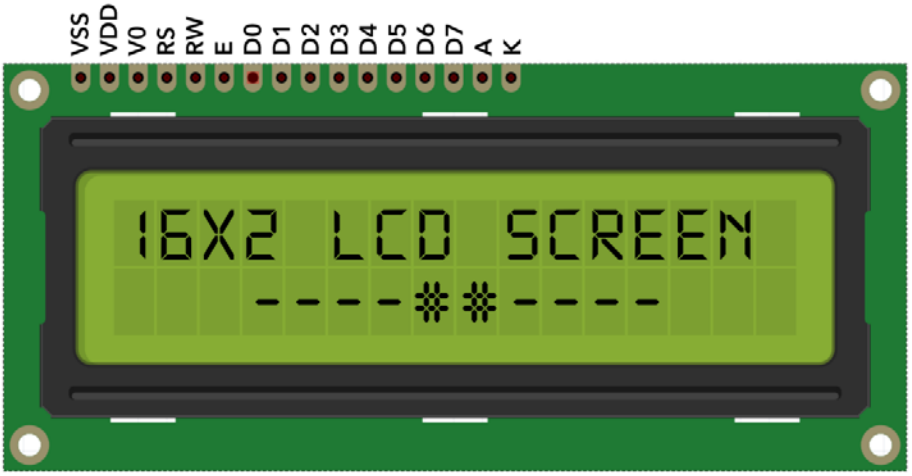
```
void loop()                    // Τυπώνει την τρέχουσα ημερομηνία και ώρα στη σειριακή κονσόλα
{
        DateTime now = rtc.now();          // Διάβασμα ημερομηνίας - ώρας από το RTC

        // Με το επόμενο μπλοκ εντολών τυπώνεται η ημερομηνία και ώρα στη σειριακή κονσόλα
        Serial.print(now.year(), DEC);
        Serial.print('/');
        Serial.print(now.month(), DEC);
        Serial.print('/');
        Serial.print(now.day(), DEC);
        Serial.print(" ");
        Serial.print(now.hour(), DEC);
        Serial.print(':');
        Serial.print(now.minute(), DEC);
        Serial.print(':');
        Serial.println(now.second(), DEC);

        delay(1000);                       // Αναμονή 1 sec
}

// Η συνάρτηση input() επιστρέφει έναν ακέραιο αριθμό μεταξύ των τιμών vmin και vmax,
// με βάση τους χαρακτήρες που αποστέλλονται από τη σειριακή κονσόλα
int input(char * msg, int vmin, int vmax, int vdefault)
{
        Serial.print(msg);                 // Τυπώνει το μήνυμα της παραμέτρου msg
        int val = Serial.parseInt();       // Διαβάζει τους χαρακτήρες που αποστέλλονται
                                           // μέσω σειριακής κονσόλας, και κωδικοποιεί
                                           // το αποτέλεσμα ως ακέραιο αριθμό
        if ((val < vmin) || (val > vmax))  // Αν το αποτέλεσμα είναι εκτός των ορίων
        {                                  // που ορίζουν οι παράμετροι vmin και vmax
                val = vdefault;            // Το αποτέλεσμα παίρνει την εξ ορισμού τιμή
        }
        Serial.println(val);               // Τυπώνει το αποτέλεσμα
        return val;                        // Επιστρέφει το αποτέλεσμα
}
```

Για τη σωστή λειτουργία του κώδικα πρέπει στη γραμμή κατάστασης της σειριακής κονσόλας να ενεργοποιηθεί η επιλογή *"Αλλαγή γραμμής"*. Τα δεδομένα που γράφουμε στη γραμμή κειμένου της σειριακής κονσόλας αποστέλλονται στον Arduino είτε με κλικ στο κουμπί *"Αποστολή"*, είτε πατώντας το πλήκτρο Enter.

Σημείωση : *Η βιβλιοθήκη RTCLib συνοδεύεται από αρκετά έτοιμα παραδείγματα κώδικα, μεταξύ των οποίων επισημαίνουμε την ύπαρξη ενός που δείχνει πώς γράφουμε ή διαβάζουμε μερικά bytes στην ή από την εσωτερική μνήμη RAM του ολοκληρωμένου DS1307, καθώς και ένα που δείχνει πως μπορούμε να ορίσουμε τη συχνότητα του τετραγωνικού παλμού που το ολοκληρωμένο παράγει.*

γ. Χρήση οθόνης υγρών κρυστάλλων (LCD) μέσω διαύλου I²C

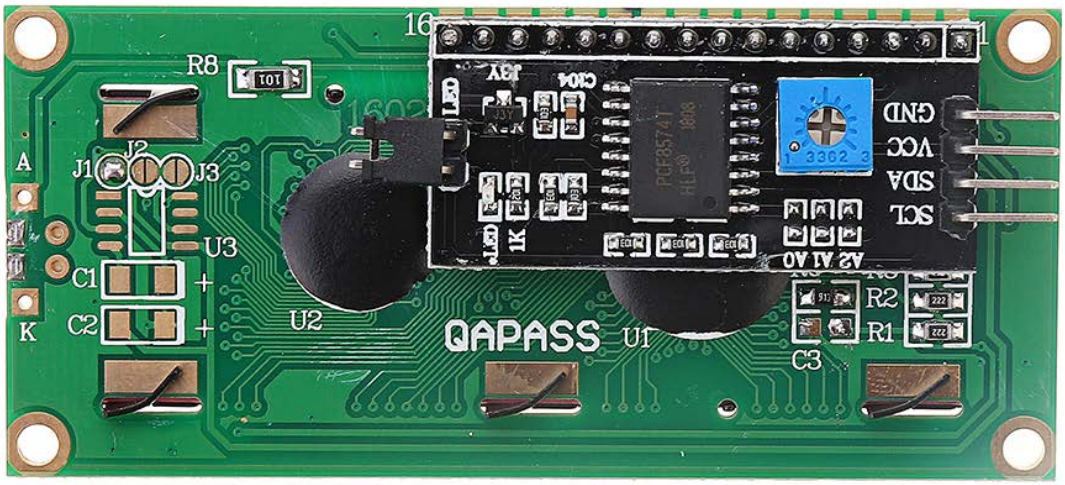
Μια από τις πιο συνηθισμένες συσκευές που συνδέεται σε ένα μικροελεγκτή είναι η οθόνη υγρών κρυστάλλων (LCD). Πιο κοινές είναι οι οθόνες που μπορούν να απεικονίσουν 2 γραμμές των 16 χαρακτήρων (16x2), ή 4 γραμμές των 20 χαρακτήρων (20x4). Τον έλεγχο της οθόνης αναλαμβάνει εξειδικευμένο ολοκληρωμένο κύκλωμα (συνήθως το HD44780), το οποίο συνδέεται στο μικροελεγκτή μέσω 14 ακίδων (παράλληλη διεπαφή).



No	Symbol	Function
1	VSS	Ground
2	VDD	5V +
3	V0	Contrast
4	RS	Register
5	RW	Read/Write
6	E	Enable
7	D0	Data bus
8	D1	Data bus
9	D2	Data bus
10	D3	Data bus
11	D4	Data bus
12	D5	Data bus
13	D6	Data bus
14	D7	Data bus
15	A	Anode (5V+)
16	K	Cathode (GND)

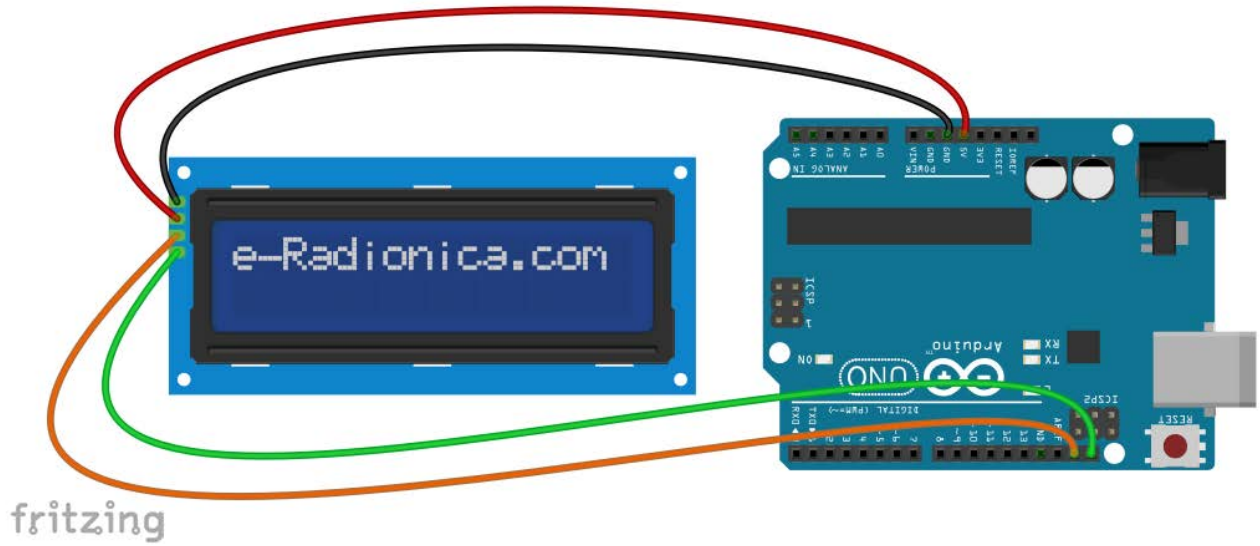
Εικόνα 31: LCD παράλληλης διεπαφής

Με ένα μικροελεγκτή περιορισμένων ακίδων, όπως ο Arduino, η χρήση LCD παράλληλης διεπαφής εύκολα μπορεί να αφήσει τα πρότζεκτ μας χωρίς πόρους. Η λύση στο πρόβλημα αυτό είναι η χρήση μιας οθόνης LCD εφοδιασμένης με ειδικό μετατροπέα της παράλληλης σύνδεσης σε σειριακή για το δίαυλο I²C. Τις δυνατότητες της I²C οθόνης LCD μπορούμε να εκμεταλλευτούμε πλήρως με τη χρήση της βιβλιοθήκης *LiquidCrystal_I2C* του Frank de Brabander, η οποία μπορεί να εγκατασταθεί μέσω του *"Διαχειριστή βιβλιοθήκης"* του Arduino IDE.



Εικόνα 32: Οθόνη LCD με I²C διεπαφή (πίσω όψη)

Μια οθόνη LCD χρησιμοποιεί στο δίαυλο I²C μια διεύθυνση μεταξύ 0x20 ως 0x27. Η ακριβής διεύθυνση της μονάδας που διαθέτετε μπορεί να βρεθεί μέσω του sketch ανίχνευσης συνδεδεμένων συσκευών στο δίαυλο I²C (ex.14.ino), που στην αρχή αυτής της ενότητας συζητήσαμε. Στο επόμενο δοκιμαστικό sketch θεωρούμε πως η χρησιμοποιούμενη οθόνη διαθέτει 2 γραμμές των 16 χαρακτήρων και αναγνωρίζεται στη διεύθυνση 0x27 στο δίαυλο I²C. Αφού λοιπόν συμπεριλάβουμε στο sketch τη σχετική βιβλιοθήκη και δημιουργήσουμε ένα αντικείμενο τύπου *LiquidCrystal_I2C*, είμαστε έτοιμοι να χρησιμοποιήσουμε την οθόνη για την εμφάνιση πληροφοριών και αποτελεσμάτων από το sketch μας. Ακολουθούν η εικόνα της σύνδεσης της οθόνης στον Arduino, και ο κώδικας του sketch:

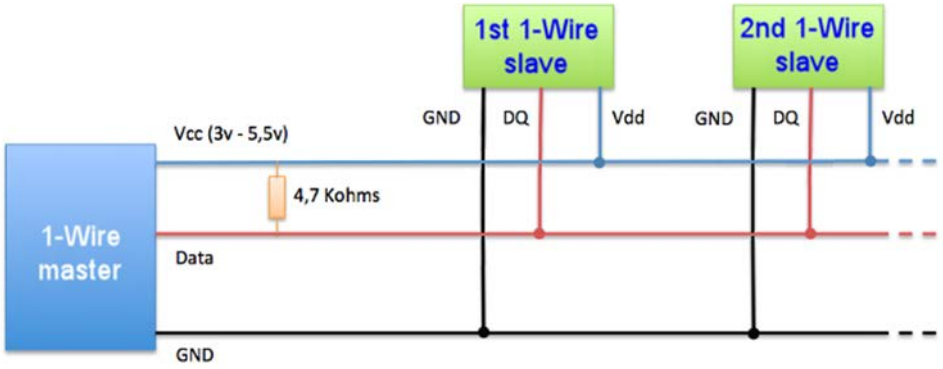


Εικόνα 33: Σύνδεση του LCD στον Arduino

```
/*
        Ex.19 : Δοκιμή της I2C οθόνης υγρών κρυστάλλων
*/

#include <Wire.h>                       // Συμπερίληψη βιβλιοθήκης Wire
#include <LiquidCrystal_I2C.h>          // Συμπερίληψη βιβλιοθήκης   LiquidCrystal_I2C

LiquidCrystal_I2C lcd(0x27,16,2);       // Δημιουργία αντικειμένου LiquidCrystal_I2C
                                        // για οθόνη τύπου 16x2 στην I2C διεύθυνση 0x27

void setup()

{
        lcd.init();                     // Ενεργοποίηση οθόνης

        lcd.backlight();                // Ενεργοποίηση οπίσθιου φωτισμού
        lcd.clear();                    // Εκκαθάριση οθόνης
        lcd.setCursor(2,0);             // Ο δρομέας στο 3ο χαρακτήρα της 1ης γραμμής
        lcd.print("Hello world!");      // Τυπώνει ένα μήνυμα
        lcd.setCursor(1,1);             // Ο δρομέας στο 2ο χαρακτήρα της 2ης γραμμής
        lcd.print("I2C LC Display");    // Τυπώνει ένα μήνυμα
}

void loop()

{
        // Τίποτα δε γίνεται εδώ. Όλα έγιναν στη συνάρτηση setup()
}
```

Χρήση του μονοσύρματου διαύλου 1-Wire

Το σειριακό πρωτόκολλο 1-Wire σχεδιάστηκε από τη Dallas Semiconductors (τώρα Maxim) για τη χαμηλού ρυθμού επικοινωνία (16,3 kbps) μεταξύ των συνδεδεμένων συσκευών, χρησιμοποιώντας μια απλή γραμμή για τη μεταφορά και τον έλεγχο της ροής των δεδομένων. Βεβαίως απαιτούνται και οι κατάλληλες γραμμές τροφοδοσίας (Vcc και GND), ενώ διαθέτει και τη λεγόμενη λειτουργία παρασιτικής τροφοδοσίας, στην οποία η γραμμή μεταφοράς δεδομένων χρησιμοποιείται και για την τροφοδοσία των συνδεδεμένων στο δίαυλο συσκευών, περιορίζοντας έτσι τον αριθμό αγωγών του διαύλου σε δύο: γραμμή μεταφοράς και γείωση. Και για τους δύο τύπους τροφοδοσίας μια pull up αντίσταση των 4,7 kΩ πρέπει να συνδεθεί στο δίαυλο 1-Wire (Εικόνα 34). Μια 1-Wire κύρια (master) συσκευή ξεκινάει και ελέγχει την επικοινωνία με μία ή περισσότερες υποτελείς (slave) συσκευές στο δίαυλο [9].

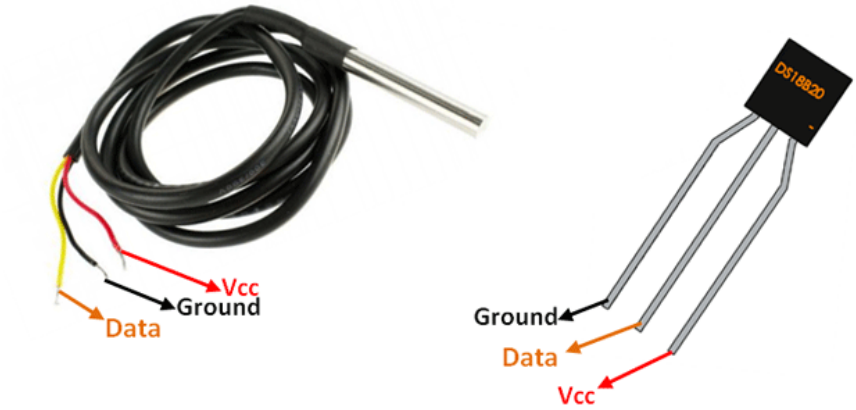


Εικόνα 34: Ο δίαυλος 1-Wire

Κάθε 1-Wire slave συσκευή διαθέτει μια 64-bit μνήμη ROM, στην οποία έχουν εργοστασιακά εγγραφεί:

- Ένας 8-bit "οικογενειακός" κωδικός που χρησιμοποιείται για την αναγνώριση του τύπου της συσκευής.
- Ένας 48-bit αναγνωριστικός αριθμός, που χρησιμεύει ως η διεύθυνση της συσκευής στο δίαυλο 1-Wire.
- Ένας 8-bit κωδικός CRC που χρησιμοποιείται για την επιβεβαίωση της ακεραιότητας των δεδομένων της μνήμης.

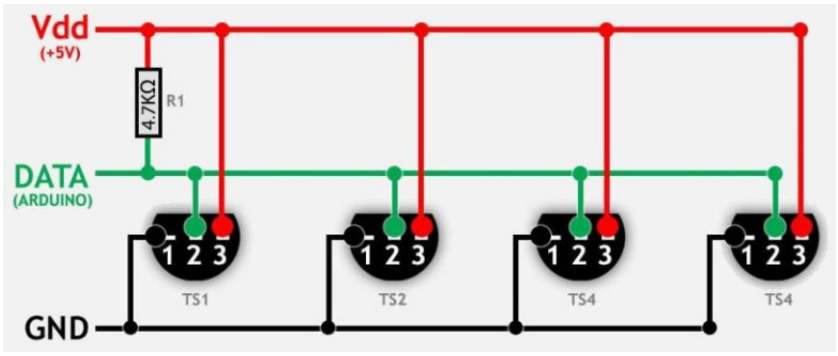
Για να εκμεταλλευτούμε στον Arduino τις δυνατότητες του διαύλου 1-Wire θα χρησιμοποιήσουμε τη βιβλιοθήκη *OpenWire* του Paul Stoffregen. Όμως αντίθετα από το δίαυλο I^2C, που η σχετική βιβλιοθήκη εγκαθίσταται εξ ορισμού με το λογισμικό του Arduino, η βιβλιοθήκη *OpenWire* για την υλοποίηση του πρωτοκόλλου 1-Wire απαιτεί τη χειροκίνητη εγκατάστασή της, η οποία μπορεί να πραγματοποιηθεί στο Arduino IDE μέσω του μενού *"Εργαλεία/Διαχείριση βιβλιοθηκών..."* ή του *"Σχέδιο\Συμπερίληψη βιβλιοθήκης\Διαχείριση βιβλιοθηκών…"*.



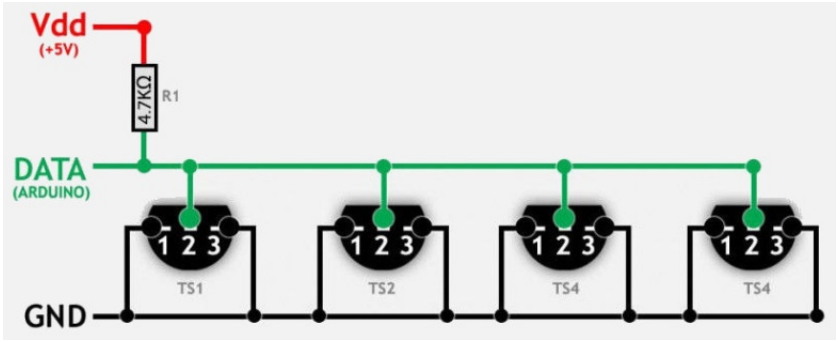
Εικόνα 35: Ο αισθητήρας DS18B20 (αδιάβροχη μορφή και μορφή ολοκληρωμένου 3 ακιδων)

Η πλέον συνηθισμένη συσκευή που μπορεί να συνδεθεί στο δίαυλο 1-Wire είναι το ψηφιακό θερμόμετρο DS18B20 που κατασκευάζεται από τη Dallas Semiconductors. Μετράει θερμοκρασίες από -55°C μέχρι 125°C με ακρίβεια ±0,5°C και διακριτική ικανότητα από 9bit μέχρι 12bit. Διατίθεται με τη μορφή ενός μικρού ολοκληρωμένου κυκλώματος τριών ακίδων, ενώ ως εμπορικό προϊόν υπάρχει και με την αδιάβροχη μορφή του (Εικόνα 35).

Οι αισθητήρες DS18B20 μπορούν να συνδεθούν στον Arduino είτε σε κανονικό (σύνδεση με τρεις αγωγούς) είτε σε παρασιτικό τρόπο (σύνδεση με δύο αγωγούς) λειτουργίας, όπως φαίνεται στα επόμενα σχήματα:

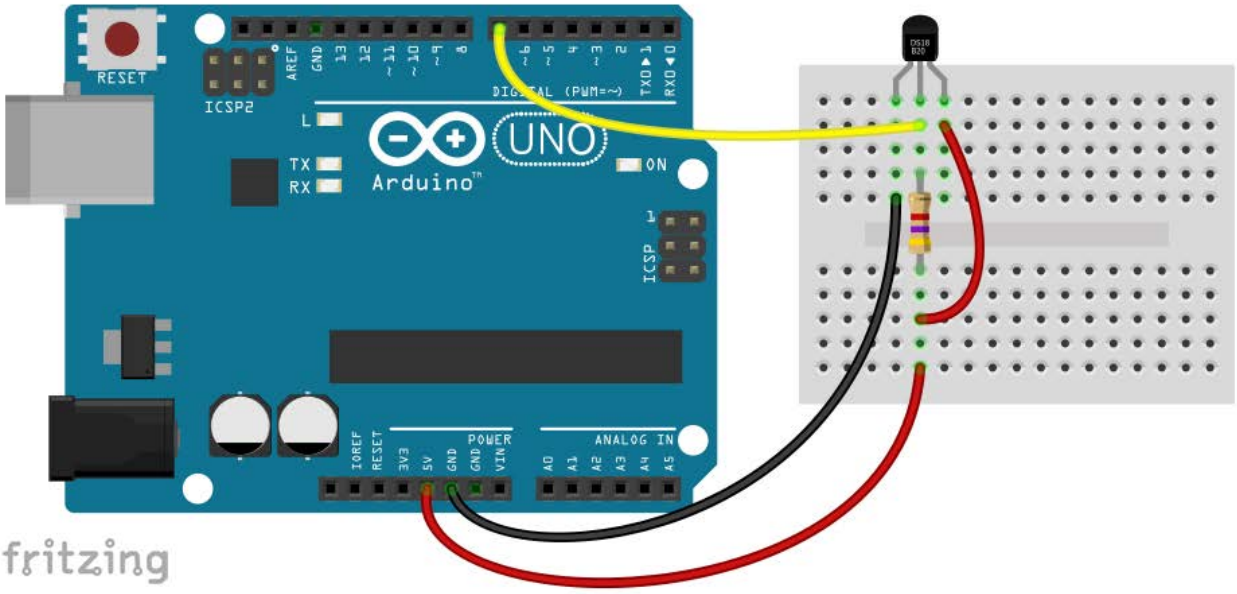


Εικόνα 36: Αισθητήρες DS18B20 σε κανονικό τρόπο λειτουργίας

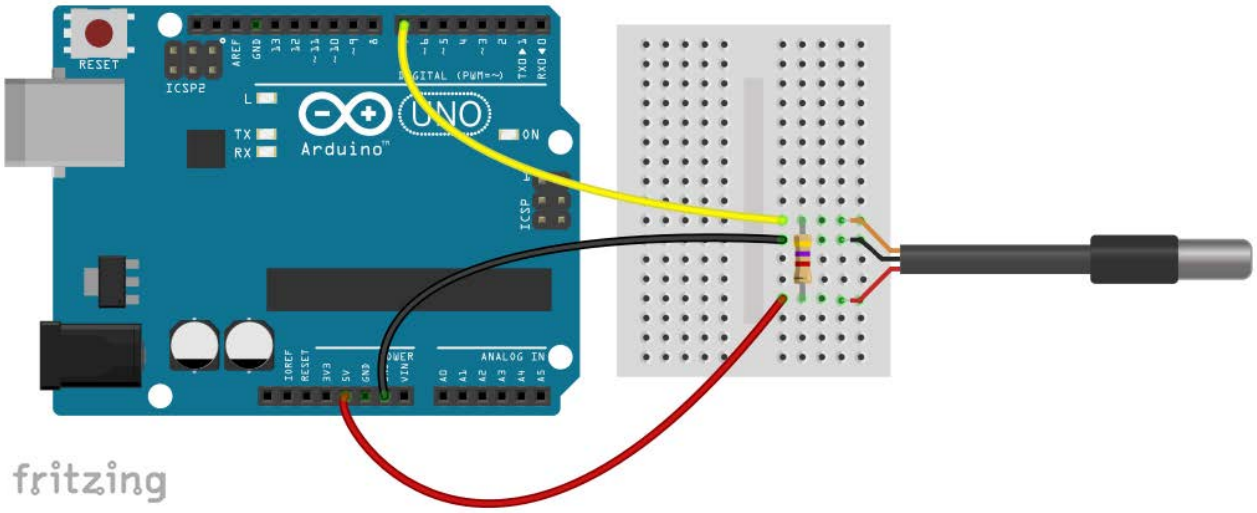


Εικόνα 37: Αισθητήρες DS18B20 σε παρασιτικό τρόπο λειτουργίας

Ο αισθητήρας μπορεί να συνδεθεί (κανονικός τρόπος λειτουργίας) στον Arduino όπως φαίνεται και στο επόμενο σχήμα:



Εικόνα 38: Σύνδεση του αισθητήρα DS18B20 στον Arduino



Εικόνα 39: Σύνδεση της αδιάβροχης εκδοχής του DS18B20 στον Arduino

Για να χρησιμοποιήσουμε τον αισθητήρα DS18B20 στα sketch μας, εκτός από τη βιβλιοθήκη *OpenWire* για την υλοποίηση του διαύλου 1-Wire, θα χρειαστούμε επιπλέον και τη βιβλιοθήκη *Dallas Temperature* του Miles Burton, η οποία υποστηρίζει τους αισθητήρες της συγκεκριμένης οικογένειας. Και πάλι η εγκατάσταση της βιβλιοθήκης γίνεται από το Arduino IDE μέσω του "*Διαχειριστή βιβλιοθήκης*".

Το επόμενο sketch αναζητά συσκευές στο δίαυλο 1-Wire και, όταν βρει μια, τυπώνει τη διεύθυνσή της στη σειριακή κονσόλα:

```
/*
        Ex.20 : Αναζήτηση συσκευών στο δίαυλο 1-Wire
*/

#include <OneWire.h>
#define W1PIN          7        // Η ψηφιακή ακίδα στην οποία συνδέεται ο αγωγός δεδομένων

/*
*/
OneWire owbus(W1PIN);     // Δημιουργία αντικειμένου για τον έλεγχο του διαύλου 1-Wire

void setup(void)
{
        Serial.begin(9600);     // Ενεργοποίηση σειριακής επικοινωνίας
        searchBus();            // Αναζήτηση για 1-Wire συσκευές στο δίαυλο
}

void searchBus(void)            // Αναζήτηση της επόμενης συσκευής στο δίαυλο
{
        byte addr[8];           // Πίνακας 8 bytes για αποθήκευση της διεύθυνσης της συσκευής
        byte nDevices = 0;      // Αριθμός των συσκευών στο δίαυλο

        Serial.println("Searching for 1-Wire devices...\n\r");
        while(owbus.search(addr))               // Βρόχος αναζήτησης συσκευών στο δίαυλο
        {
                Serial.print("Address = ");
```

```
            for(byte i = 0; i < 7; i++)       // Τυπώνει τα 7 bytes της διεύθυνσης της συσκευής
            {
                  if (addr[i] < 16)
                  {
                        Serial.print('0');
                  }
                  Serial.print(addr[i], HEX);
                  Serial.print(":");
            }
            Serial.println(addr[7], HEX);          // Τυπώνει και το τελευταίο byte της διεύθυνσης
            nDevices++;                            // Αυξάνει τον αριθμό των συσκευών κατά 1
      }

      owbus.reset_search();                        // Προετοιμασία για την επόμενη αναζήτηση

      if (nDevices > 0)       // Τυπώνει το τελικό μήνυμα ανάλογα αν βρέθηκαν ή όχι συσκευές
      {
            Serial.println();
            Serial.print(nDevices);
            Serial.println(" device(s) found.");
      }
      else
      {
            Serial.println("No devices found.");
      }
}

void loop(void)
{
      // Τίποτα δε γίνεται εδώ. Όλα έγιναν στη συνάρτηση setup()
}
```

Για να χρησιμοποιήσουμε τον αισθητήρα για μέτρηση της θερμοκρασίας, απαιτείται κατ' αρχάς να συμπεριλάβουμε στο sketch και τη βιβλιοθήκη *Dallas Temperature*, με τη δήλωση:

```
#include <DallasTemperature.h>
```

Έπειτα πρέπει να ορίσουμε ένα αντικείμενο τύπου *DallasTemperature* για να αποκτήσουμε πρόσβαση στον κώδικα που αφορά τη διαχείριση του αισθητήρα DS18B20. Για τη δημιουργία του αντικειμένου *DallasTemperature* απαιτείται ως παράμετρος η διεύθυνση ενός αντικειμένου τύπου *OneWire*, που πρέπει να οριστεί νωρίτερα για τη δημιουργία και τον έλεγχο του διαύλου 1-Wire:

```
#define W1BUS 7
OneWire owbus(W1BUS);
DallasTemperature devices(&owbus);
```

Στη συνάρτηση *setup()* του sketch αρχικοποιείται ο δίαυλος 1-Wire και αναγνωρίζονται οι συνδεδεμένες συσκευές μέσω της εντολής:

```
devices.begin()
```

Για τη μέτρηση της θερμοκρασίας πρέπει να στείλουμε μια γενική αίτηση στο δίαυλο για επιστροφή των θερμοκρασιών από όλους τους συνδεδεμένους αισθητήρες με την εντολή:

```
devices.requestTemperatures();
```

Στη συνέχεια η ανάγνωση των επιστρεφομένων τιμών θερμοκρασίας μπορεί να γίνει με δύο τρόπους:

1. Αν είναι γνωστή η διεύθυνση του αισθητήρα, π.χ. ως πίνακας τύπου *DeviceAddress*:

   DeviceAddress sensor1 = { 0x28, 0xFF 0x0B, 0xB0, 0x67, 0x14, 0x03, 0x43 };

   τότε μπορούμε άμεσα να καλέσουμε τη συνάρτηση *getTempC()* της βιβλιοθήκης *Dallas Temperature*, ως εξής:

   float tempC = devices.getTempC(sensor1);

2. Αν δεν είναι γνωστή η διεύθυνση του αισθητήρα, μπορούμε να διαβάσουμε τη θερμοκρασία από τον πρώτο για παράδειγμα αισθητήρα, που αναγνωρίστηκε στο δίαυλο, με την εντολή:

   devices.getTempCByIndex(0);

   και ανάλογα για του υπόλοιπους (αν υπάρχουν) αισθητήρες στο δίαυλο.

Τα επόμενα δύο sketches υλοποιούν τους δύο αυτούς τρόπους που περιγράψαμε:

```
/*
      Ex.21 : Πρώτο παράδειγμα χρήσης αισθητήρα DS18B20
*/

#include <OneWire.h>
#include <DallasTemperature.h>

#define W1BUS 7                       // Η γραμμή μεταφοράς δεδομένων του διαύλου I2C
                                      // συνδέεται στην ακίδα 7 του Arduino

OneWire owbus(W1BUS);                 // Ορισμός αντικειμένου διαχείρισης διαύλου 1-Wire
DallasTemperature devices(&owbus);    // Ορισμός αντικειμένου DallasTemperature

// Ορισμός της διεύθυνσης του συνδεδεμένου αισθητήρα.
// Πρέπει να αντικατασταθεί με τη διεύθυνση του δικού σας αισθητήρα
DeviceAddress sensors1 = { 0x28, 0xFF, 0x0B, 0xB0, 0x67, 0x14, 0x03, 0x43 };

void setup(void)
{
      Serial.begin(9600);             // Ενεργοποίηση σειριακής επικοινωνίας
      devices.begin();                // Ενεργοποίηση διαύλου 1-Wire
                                      // και αναγνώριση των συνδεδεμένων συσκευών
      devices.setResolution(sensors1, 10);  // Ορισμός διακριτικής ικανότητας αισθητήρα
}

void loop(void)
{
      devices.requestTemperatures();        // Γενική αίτηση για λήψη μετρήσεων

      float tempC = devices.getTempC(sensors1);   // Ανάγνωση θερμοκρασίας

      Serial.print("Temperature: ");        // Τυπώνει τη θερμοκρασία
      Serial.print(tempC);
      Serial.println("°C");
      delay(1000);                          // Αναμονή 1 sec
}
```

```
/*
        Ex.22 : Δεύτερο παράδειγμα χρήσης αισθητήρα DS18B20
*/

#include <OneWire.h>
#include <DallasTemperature.h>

#define W1BUS 7                         // Η γραμμή μεταφοράς δεδομένων συνδέεται
                                        // στην ακίδα 7 του Arduino

OneWire owbus(W1BUS);                   // Ορισμός αντικειμένου διαχείρισης διαύλου 1-Wire
DallasTemperature devices(&owbus);      // Ορισμός αντικειμένου DallasTemperature

void setup(void)
{
        Serial.begin(9600);                     // Ενεργοποίηση σειριακής επικοινωνίας

        // Ενεργοποίηση διαύλου 1-Wire και αναγνώριση των συνδεδεμένων συσκευών
        devices.begin();
 }

void loop(void)
{
        devices.requestTemperatures();          // Γενική αίτηση για λήψη μετρήσεων

        float tempC = devices. getTempCByIndex(0);      // Ανάγνωση θερμοκρασίας από τον
                                                        // πρώτο αισθητήρα του διαύλου
        Serial.print("Temperature: ");                  // Τυπώνει τη θερμοκρασία
        Serial.print(tempC);
        Serial.println("°C");
        delay(1000);                                    // Αναμονή 1 sec
}
```

Χρήση του διαύλου SPI

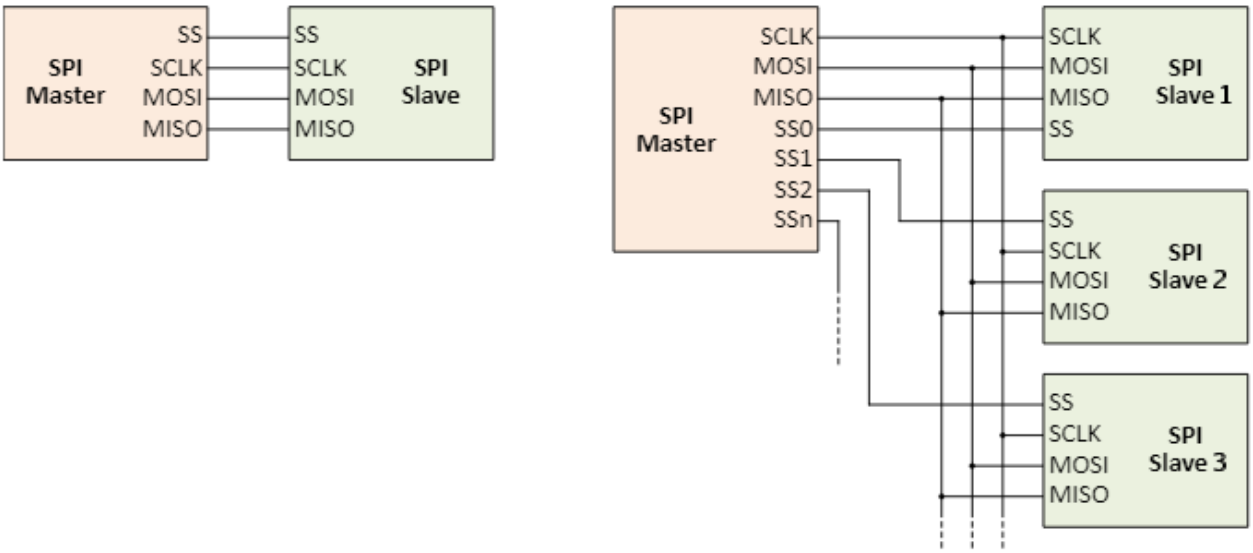
Το Serial Peripheral Interface (SPI) είναι ένα σειριακό πρωτόκολλο που χρησιμοποιείται συνήθως για την επικοινωνία (αποστολή και λήψη δεδομένων) μεταξύ μικροελεγκτών και περιφερειακών συσκευών, όπως αισθητήρες, κάρτες SD, κ.ά.. Πρόκειται για ένα γρήγορο (εξ ορισμού ρυθμός επικοινωνίας στον Arduino Uno 4 MHz), εύκολο και απλό στην υλοποίηση πρωτόκολλο, που υποστηρίζει την επικοινωνία μεταξύ μιας κύριας (*master*) και πολλών υποτελών (*slaves*) συσκευών. Το πρωτόκολλο υποστηρίζει σύγχρονη επικοινωνία, δηλ. επικοινωνία ελεγχόμενη από κατάλληλα χρονισμένους παλμούς για τον τέλειο συγχρονισμό master - slave συσκευών.

Τυπικά για την υλοποίηση του πρωτοκόλλου SPI, και εκτός από τους αγωγούς τροφοδοσίας, απαιτούνται τρεις γραμμές (αγωγοί) κοινές για όλες τις συσκευές που πρόκειται να συνδεθούν στο δίαυλο [10]:

- MISO (Master In Slave Out): Η γραμμή μεταφοράς δεδομένων από τις υποτελείς συσκευές προς την κύρια.
- MOSI (Master Out Slave In): Η γραμμή μεταφοράς δεδομένων από την κύρια προς τις υποτελείς συσκευές.
- SCK ή SCLK (Serial Clock): Τα σήματα στη γραμμή αυτή συγχρονίζουν τη μεταφορά των δεδομένων, και δημιουργούνται από την κύρια συσκευή του διαύλου.

Επιπλέον για κάθε υποτελή συσκευή στο δίαυλο απαιτείται και μια ξεχωριστή γραμμή:

- SS (Slave Select) ή CS (Chip Select), μέσω της οποίας η κύρια συσκευή ενεργοποιεί ή απενεργοποιεί την υποτελή. Χαμηλή στάθμη στην ακίδα CS μιας υποτελούς συσκευής οδηγεί σε ενεργοποίηση της επικοινωνίας της με την κύρια συσκευή και αντίστροφα.



Εικόνα 40: Σύνδεση στο δίαυλο SPI

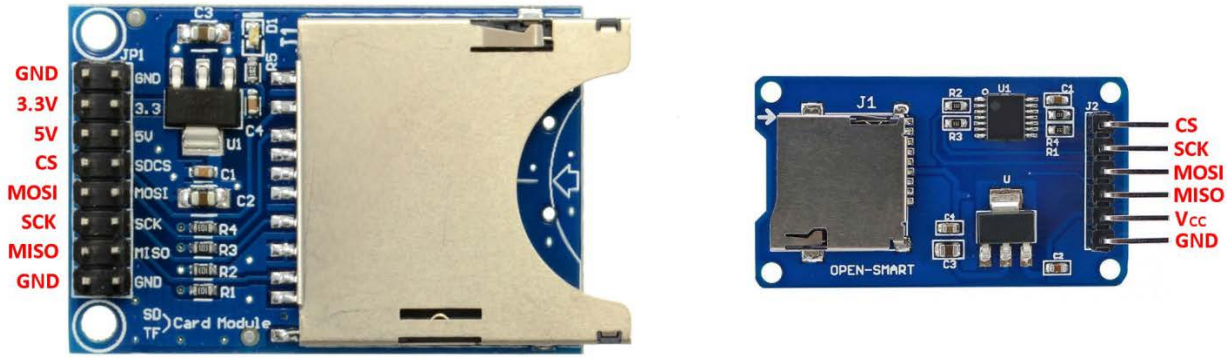
Στον Arduino Uno οι γραμμές του διαύλου SPI διανέμονται ως εξής [4]:

- MOSI στην ψηφιακή ακίδα 11
- MISO στην ψηφιακή ακίδα 12
- SCK στην ψηφιακή ακίδα 13
- SS (ή CS) στην ψηφιακή ακίδα 10

Η βιβλιοθήκη *SPI*, που επιτρέπει την επικοινωνία του Arduino με SPI συσκευές, εγκαθίσταται εξ ορισμού με την εγκατάσταση του Arduino IDE. Όπως ήδη αναφέραμε χαμηλή στάθμη στην ακίδα 10 (CS) του Arduino Uno θα τον ενεργοποιούσε ως υποτελή. Αυτό θα καθιστούσε τη βιβλιοθή-

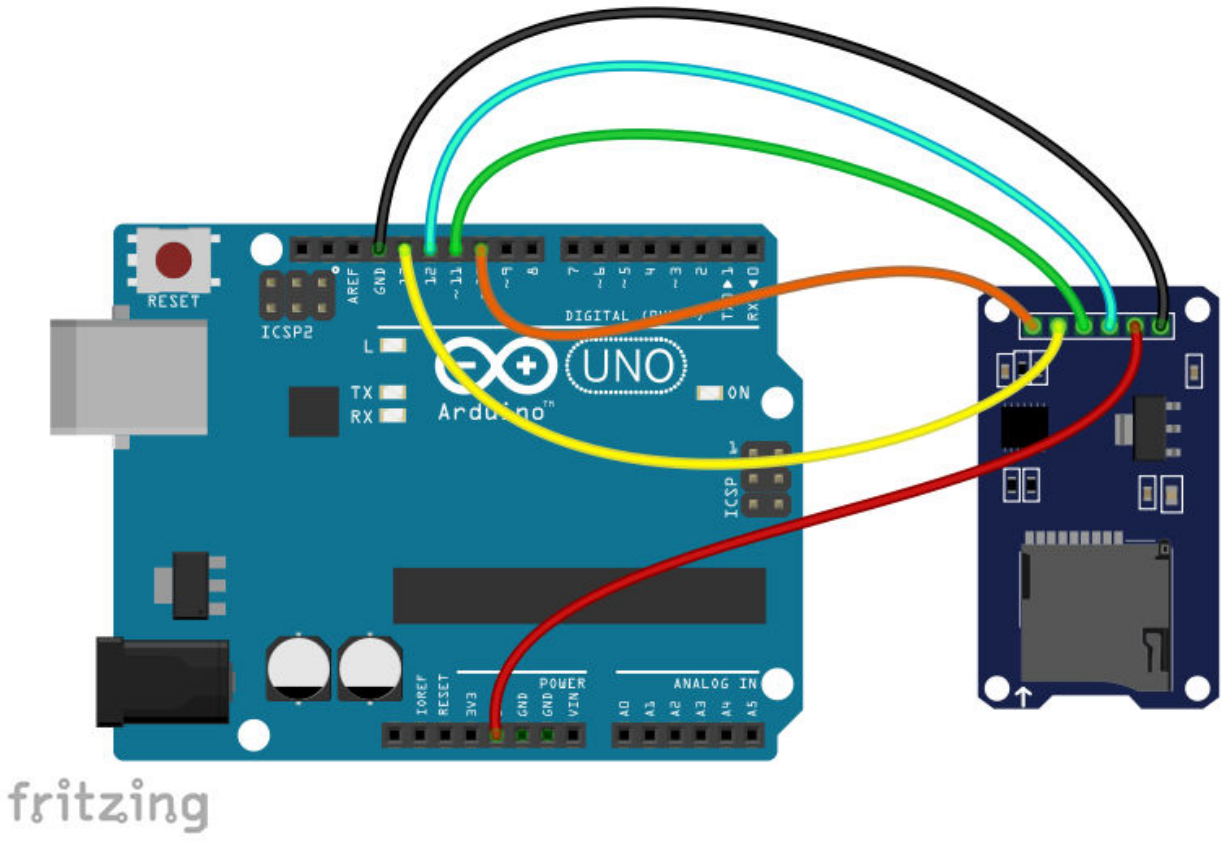
κη *SPI* αδρανή, αφού υποστηρίζει μόνο λειτουργία του Arduino ως κύρια (master) συσκευή. Για το λόγο αυτό η ψηφιακή ακίδα 10 του Arduino πρέπει πάντα, όταν χρησιμοποιούμε τη βιβλιοθήκη *SPI*, να χαρακτηρίζεται ως έξοδος (OUTPUT). Υπάρχει βέβαια η δυνατότητα να χρησιμοποιήσουμε και οποιαδήποτε άλλη ακίδα για την ενεργοποίηση μιας υποτελούς συσκευής.

Εξ ορισμού επίσης εγκαθίσταται και η βιβλιοθήκη *SD*, η οποία με χρήση του πρωτοκόλλου SPI, χρησιμοποιείται για την επικοινωνία του Arduino με μια μονάδα κάρτας μνήμης SD ή micro SD. Η βιβλιοθήκη *SD* αποτελεί ένα ανωτέρου επιπέδου κέλυφος για τη βιβλιοθήκη *SDFat*, η οποία παρέχει πρόσβαση ανάγνωσης/εγγραφής σε συστήματα αρχείων FAT16/FAT32 για κάρτες μνήμης SD/SDHC. Η κάρτα που θα χρησιμοποιήσουμε θα πρέπει προηγουμένως να έχει διαμορφωθεί, κατά προτίμηση για σύστημα αρχείων FAT16. Η διαμόρφωση της κάρτας SD μπορεί να γίνει για παράδειγμα με ένα ηλεκτρονικό υπολογιστή που είναι εφοδιασμένος με συσκευή ανάγνωσης/εγγραφής καρτών SD.

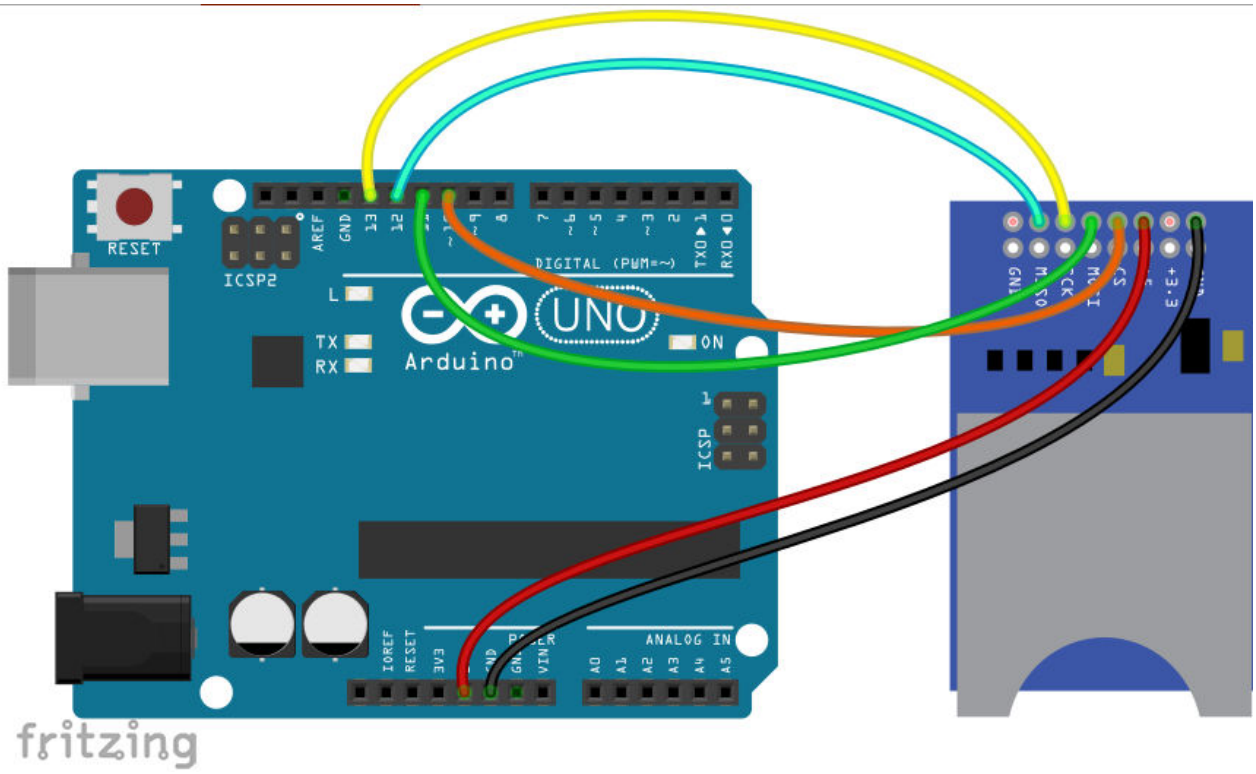


Εικόνα 41 : Αριστερά η μονάδα SD Card και δεξιά η μονάδα micro SD Card

Οι συνδέσεις που απαιτούνται για την επικοινωνία της μονάδας micro SD ή SD Card με τον Arduino φαίνονται στα επόμενα δύο σχήματα:



Εικόνα 42: Σύνδεση μονάδας κάρτας micro SD στον Arduino



Εικόνα 43 : Σύνδεση της μονάδας SD Card στον Arduino

Ένα πρώτο δοκιμαστικό sketch με τη μονάδα κάρτας SD ή micro SD, που απλά ελέγχει αν υπάρχει κάρτα SD στη μονάδα, έχει ως εξής:

```
/*
        Ex.23 : Πρώτο δοκιμαστικό sketch με τη μονάδα κάρτας SD ή micro SD
*/

#include <SD.h>                     // Συμπερίληψη βιβλιοθήκης SD
#include <SPI.h>                    // Συμπερίληψη βιβλιοθήκης SPI

void setup()
{
        Serial.begin(9600);         // Ενεργοποίηση σειριακής επικοινωνίας
        Serial.println("Initializing SD card...");
        Serial.println();

        pinMode(SS, OUTPUT);        // Σύμφωνα με την τεκμηρίωση η ακίδα 10 (SS) πρέπει
                                    // να οριστεί ως έξοδος, ακόμη κι αν δε χρησιμοποιείται

        if (SD.begin())             // Έλεγχος αν υπάρχει SD κάρτα που μπορεί να αρχικοποιηθεί
        {
                Serial.println("SD initiliazed.");
        }
        else
        {
                Serial.println("SD did not initialize.");
        }
}
```

```
void loop()
{
        // Τίποτα δε γίνεται εδώ. Όλα έγιναν στη συνάρτηση setup()
}
```

Η συνάρτηση *SD.begin()* προετοιμάζει για χρήση τόσο το δίαυλο SPI όσο και την κάρτα SD, και επιστρέφει λογικό 1 (*true*) αν η προετοιμασία ήταν επιτυχής, και λογικό 0 (*false*) σε περίπτωση αποτυχίας. Στην περίπτωση που χρησιμοποιείται άλλη ψηφιακή ακίδα του Arduino (π.χ. η 4) ως ακίδα επιλογής (CS) μιας συσκευής SPI, αυτό πρέπει να δηλωθεί ως εξής:

```
SD.begin(4);
```

Παρατήρηση : *Η μονάδα κάρτας SD μέσω της αντίστοιχης ακίδας που διαθέτει μπορεί να τροφοδοτηθεί είτε από τάση 5V, είτε από 3,3V.*

Για να ανοίξουμε ένα αρχείο κειμένου στην κάρτα SD για εγγραφή, πρέπει πρώτα να δημιουργήσουμε ένα αντικείμενο κλάσης File, ως εξής:

```
File myfile;
```

και στη συνέχεια να δημιουργήσουμε το αρχείο στην κάρτα SD, ως εξής:

```
myFile = SD.open("datafile.txt", FILE_WRITE);
```

Η πρώτη παράμετρος ("datafile.txt") είναι το όνομα του αρχείου, ενώ με τη δεύτερη παράμετρο (FILE_WRITE) δηλώνουμε ότι έχουμε τις δυνατότητες ανάγνωσης και εγγραφής στο αρχείο. Υπάρχει επιπλέον και η δυνατότητα (με την παράμετρο FILE_READ) να ανοιχθεί το αρχείο μόνο για ανάγνωση. Στην περίπτωση που το αρχείο υπάρχει ήδη στην κάρτα, τότε με την παραπάνω εντολή το αρχείο ανοίγει για εγγραφή, αντί να δημιουργηθεί ένα νέο.

Την ύπαρξη ενός αρχείου (π.χ. με το όνομα datafile.txt) στην κάρτα SD μπορούμε να την ελέγξουμε με την εντολή:

```
SD.exists("datafile.txt");
```

Όταν όλες οι ενέργειες εγγραφής/ανάγνωσης στο αρχείο ολοκληρωθούν, πρέπει να το κλείσουμε με την εντολή:

```
SD.close("datafile.txt");
```

Για την εγγραφή δεδομένων στο αρχείο (myfile) που έχουμε ήδη ανοίξει για εγγραφή, μπορούμε να χρησιμοποιήσουμε τις εντολές:

- `myfile.write(data);` *// data : char ή byte ή string*
- `myfile.write(buf, len);` *// buf : array of char, len : αριθμός χαρακτήρων για εγγραφή*
- `myfile.print(data);` *// data : char, byte, string ή αριθμός*
- `myfile.println(data);` *// data : char, byte, string ή αριθμός*

Για την ανάγνωση δεδομένων από το αρχείο που έχουμε ήδη ανοίξει, μπορούμε να χρησιμοποιήσουμε τις συναρτήσεις:

- `data = myfile.read();` *// data : byte ή char*
- `line = myfile.readStringUntil(terminator)` *// line : string, terminator : char*

Η συνάρτηση *readStringUntil()* προέρχεται από την κλάση *Stream*, και -στην περίπτωσή μας- επιστρέφει το αλφαριθμητικό που διαβάζει από το αρχείο μέχρι να ανιχνευθεί ο χαρακτήρας σήμανσης τέλους (terminator). Συνηθισμένος χαρακτήρας σήμανσης τέλους ο χαρακτήρας newline, που δηλώνεται ως **'\n'**.

Αν θέλουμε να ελέγξουμε αν υπάρχουν και άλλοι χαρακτήρες διαθέσιμοι για διάβασμα από κάποιο αρχείο που έχουμε ήδη ανοίξει, χρησιμοποιούμε τη συνάρτηση *available()*. Για παράδειγμα η εντολή:

```
int count = myfile.available();
```

αποθηκεύει στην ακέραια μεταβλητή *count* τον αριθμό των χαρακτήρων στο αρχείο *myfile* που είναι διαθέσιμοι για διάβασμα.

Στο επόμενο sketch, αφού δημιουργήσουμε ένα αρχείο κειμένου στην κάρτα SD, θα εγγράψουμε μερικά δεδομένα σ' αυτό, και τελικά θα διαβάσουμε τα δεδομένα αυτά και θα τα τυπώσουμε στη σειριακή κονσόλα:

```
/*
        Ex.24 : Δημιουργία αρχείου στη μονάδα κάρτας SD
*/
#include <SD.h>                     // Συμπερίληψη βιβλιοθήκης SD
#include <SPI.h>                    // Συμπερίληψη βιβλιοθήκης SPI

char myfile_name[] = "datafile.txt";  // Το όνομα του αρχείου

void setup()
{
        Serial.begin(9600);         // Ενεργοποίηση σειριακής επικοινωνίας
        pinMode(SS, OUTPUT);        // Η ακίδα 10 (SS) πρέπει να οριστεί ως έξοδος
        Serial.print("Initializing SD card...");
        if (SD.begin())             // Έλεγχος αν υπάρχει SD κάρτα που μπορεί να αρχικοποιηθεί
        {
                Serial.println("SD initiliazed.");
                writeToFile(myfile_name);       // Δημιουργία αρχείου και εγγραφή δεδομένων
                Serial.println();
                readFromFile(myfile_name);      // Ανάγνωση δεδομένων από το αρχείο
                Serial.println("All done!!!");
        }
        else
        {
                Serial.println("SD did not initialize.");
        }
}

void loop()
{
        // Τίποτα δε γίνεται εδώ. Όλα έγιναν στη συνάρτηση setup()
}

// Δημιουργία αρχείου στην κάρτα SD, εγγραφή δεδομένων και αποθήκευση
void writeToFile(char * filename)
{
        File myfile;                    // Δημιουργία αντικειμένου τύπου File
        if (SD.exists(filename))        // Αν ήδη υπάρχει το αρχείο στην κάρτα
```

```
        {
                SD.remove(filename);          // διάγραψέ το
                Serial.println("Existing file removed !");
        }
        // Δημιουργία αρχείου για εγγραφή
        Serial.println("Creating a new file…");
        myfile = SD.open(filename, FILE_WRITE);
        if (myfile)                    // Αν το αρχείο δημιουργήθηκε επιτυχώς
        {
                Serial.println("Writing some data to file…");
                for (int i = 1; i < 11; i++)       // Εγγραφή δεδομένων στο αρχείο
                {
                        myfile.print("This is Data Line number ");
                        myfile.println(i);

                        // Εκτύπωση στη σειριακή κονσόλα
                        Serial.print("Writing : ");
                        Serial.print("This is Data Line number ");
                        Serial.println(i);
                }
        }
        myfile.close();                // Κλείσιμο του αρχείου
}

// Άνοιγμα αρχείου για διάβασμα δεδομένων
void readFromFile(char * filename)
{
        File myfile;                           // Δημιουργία αντικειμένου τύπου File
        String line = "";                      // Για την ανάγνωση δεδομένων από το αρχείο
        // Άνοιγμα αρχείου για ανάγνωση
        myfile = SD.open(filename, FILE_READ);
        Serial.println("Reading from file...");
        while (myfile.available() != 0)        // Όσο υπάρχουν δεδομένα διαθέσιμα για διάβασμα
        {
                line = myfile.readStringUntil('\n');    // διάβασέ τα
                Serial.println(line);                   // και τύπωσέ τα στη σειριακή κονσόλα
        }
        myfile.close();                        // Κλείσιμο του αρχείου
        Serial.println();
}
```

Διακοπές

Οι διακοπές είναι σήματα που διακόπτουν την τρέχουσα δραστηριότητα του μικροελεγκτή. Μπορούν να ενεργοποιηθούν είτε με βάση κάποιο εξωτερικό γεγονός (την αλλαγή για παράδειγμα της κατάστασης σε μια ψηφιακή είσοδο), είτε από κάποιο εσωτερικό γεγονός (π.χ. ένα σήμα από κάποιο χρονιστή). Με την ενεργοποίηση της διακοπής ο επεξεργαστής αφού αποθηκεύσει την κατάσταση στην οποία βρισκόταν τη στιγμή της ενεργοποίησης της διακοπής, διακόπτει την τρέχουσα δραστηριότητά του, εκτελεί μια ρουτίνα (ρουτίνα - συνάρτηση εξυπηρέτησης διακοπής: ISR) συσχετισμένη με τη συγκεκριμένη διακοπή, και αφού ολοκληρώσει την εκτέλεσή της επιστρέφει στην κανονική ροή του προγράμματος.

Το βασικό πλεονέκτημα από τη χρήση διακοπών είναι η ταχύτατη ανταπόκριση του συστήματος σε εξωτερικά ή εσωτερικά σήματα. Εξ ορισμού καμιά άλλη διακοπή δεν ενεργοποιείται όταν ήδη εκτελείται μια ρουτίνα διακοπής, συνεπώς απαιτείται οι ρουτίνες διακοπής να είναι όσο το δυνατό λιγότερο χρονοβόρες. Επίσης υπάρχουν και κάποιοι περιορισμοί σε σχέση με τις ρουτίνες διακοπής, όπως για παράδειγμα ότι δε δέχονται παραμέτρους και δεν επιστρέφουν καμία τιμή, ενώ κάθε μεταβλητή που η τιμή της μεταβάλλεται από τη ρουτίνα πρέπει να χαρακτηριστεί ως **πτητική** (volatile). Ο χαρακτηρισμός αυτός αντικατοπτρίζει το γεγονός ότι η μεταβλητή αυτή μπορεί να αλλάξει από εξωτερικούς παράγοντες που δε βρίσκονται στον απόλυτο έλεγχο του προγράμματος, και αποτρέπει τον μεταγλωττιστή (compiler) να αφαιρέσει την αναφορά στη μεταβλητή, αν τυχόν δε χρησιμοποιείται άμεσα στις συναρτήσεις *setup()* και *loop()* του προγράμματος.

Οι διακοπές μπορούν να απενεργοποιηθούν με τις εντολές *noIterrupts()* ή *cli()*, και αντίστοιχα να ενεργοποιηθούν με τις εντολές *interrupts()* ή *sei()*. Εξ' ορισμού είναι ενεργοποιημένες. Όταν ο μικροελεγκτής αρχίζει την εκτέλεση μιας ρουτίνας εξυπηρέτησης διακοπής (ISR) οι διακοπές απενεργοποιούνται, και εκ νέου ενεργοποιούνται κατά την έξοδο από την ISR.

α. Εξωτερικές διακοπές

Ο Arduino Uno διαθέτει δύο εξωτερικές διακοπές: INT0 και INT1, συσχετισμένες με τις ψηφιακές ακίδες 2 και 3 αντίστοιχα [4]. Δηλαδή κάποιο εξωτερικό σήμα στην ακίδα 2 ενεργοποιεί τη διακοπή INT0, και αντίστοιχα εξωτερικό σήμα στην ακίδα 3 ενεργοποιεί τη διακοπή INT1. Υποστηρίζονται τέσσερεις διαφορετικοί τρόποι ενεργοποίησης των διακοπών:

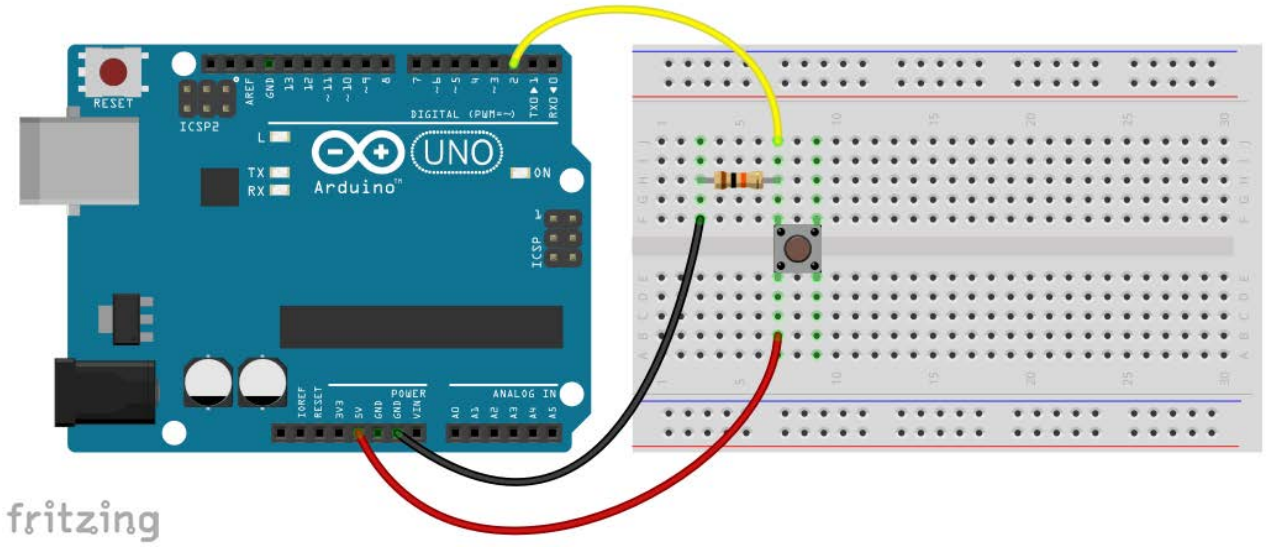
- LOW : Ενεργοποίηση οποτεδήποτε η συσχετισμένη ακίδα είναι σε χαμηλή στάθμη.
- CHANGE : Ενεργοποίηση οποτεδήποτε η συσχετισμένη ακίδα αλλάζει κατάσταση.
- RISING : Ενεργοποίηση οποτεδήποτε η συσχετισμένη ακίδα αλλάζει κατάσταση από χαμηλή σε υψηλή στάθμη.
- FALLING : Ενεργοποίηση οποτεδήποτε η συσχετισμένη ακίδα αλλάζει κατάσταση από υψηλή σε χαμηλή στάθμη.

Στον Arduino η διαδικασία ενεργοποίησης/απενεργοποίησης μιας εξωτερικής διακοπής έχει απλοποιηθεί με τις εντολές:

- **attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)**: Καθορίζει τη ρουτίνα που θα εκτελείται όταν η αντίστοιχη διακοπή ενεργοποιηθεί. Δέχεται τρεις παραμέτρους ως εξής:
    - *pin* : η ακίδα (2 ή 3 στον Arduino Uno) ενεργοποίησης της διακοπής.
    - *ISR* : Η ρουτίνα (συνάρτηση) που καλείται όταν ενεργοποιηθεί η διακοπή.
    - *mode* : ο τρόπος ενεργοποίησης της διακοπής (LOW, CHANGE, RISING, FALLING).
- **detachInterrupt(digitalPinToInterrupt(pin))**: Απενεργοποιεί τη διακοπή που έχει συσχετιστεί με την ακίδα *pin*.

Ένα παράδειγμα χρήσης εξωτερικής διακοπής

Ένας πιεστικός διακόπτης (push button) συνδέεται στον Arduino μέσω μιας pull down αντίστασης των 10kΩ, όπως φαίνεται στο επόμενο σχήμα:



Εικόνα 44: Σύνδεση διακόπτη στην ψηφιακή είσοδο 2 του Arduino

Χρησιμοποιώντας την εξωτερική διακοπή INT0 -συσχετισμένη με την ψηφιακή είσοδο 2 στην οποία συνδέεται ο ένας πόλος του διακόπτη- το πρόγραμμά μας ανιχνεύει τις αλλαγές κατάστασης του διακόπτη και ανάλογα ενεργοποιεί ή απενεργοποιεί το LED που από κατασκευής είναι συνδεδεμένο στην ψηφιακή ακίδα 13 του Arduino.

```
/*
        Ex.25 : Παράδειγμα χρήσης εξωτερικής διακοπής
*/

#define LEDPIN 13
#define INTERRUPT_PIN 2

volatile int state = LOW;                   // Η πτητική (volatile)μεταβλητή
                                            // για καταχώρηση της κατάστασης του διακόπτη

void setup()
{
        pinMode(LEDPIN, OUTPUT);            // Καθορίζουμε την ψηφιακή ακίδα 13 ως έξοδο

        // Ενεργοποίηση διακοπής στην ακίδα 2 , έλεγχος οποιασδήποτε αλλαγής κατάστασης
        attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), stateChange, CHANGE);
}

void stateChange()                  // Η ρουτίνα της διακοπής αλλάζει την κατάσταση του LED
{
        state = !state;                     // Μεταβολή της τιμής της μεταβλητής state
                                            // από false σε true και αντίστροφα
        digitalWrite(LEDPIN, state);        // Αλλαγή κατάστασης LED
}
```

```
void loop()
{
        // Δε χρειάζεται καμιά ενέργεια. Όλα γίνονται μέσω της ρουτίνας διακοπής.
}
```

Παρατήρηση : *Όταν πιέζουμε το κουμπί ενός διακόπτη ενδέχεται να μη "κλείσει" άμεσα ο διακόπτης. Μπορεί οι δύο επαφές του διακόπτη να ακουμπήσουν με τη μία πλευρά τους, μετά με τις άλλες πλευρές τους και τελικά να αποκατασταθεί πλήρως η επαφή τους. Αυτή η αποκατάσταση και διακοπή της επαφής ονομάζεται "αναπήδηση", και μπορεί να δημιουργήσει σημαντικά προβλήματα στα sketches μας. Καθώς στο διαδίκτυο υπάρχουν διάφορες λύσεις στο πρόβλημα της "αναπήδησης" (debouncing) δε θα ασχοληθούμε περισσότερο με το θέμα.*

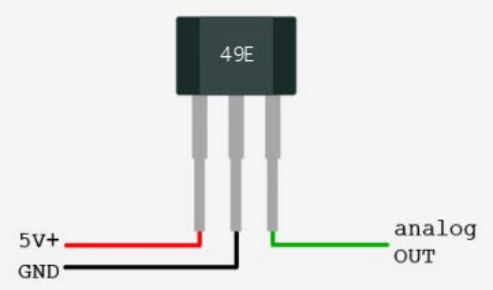
β. Διακοπές χρονιστή (Timer Interrupts)

Οι διακοπές χρονιστή επιτρέπουν την εκτέλεση μιας συγκεκριμένης εργασίας σε επακριβώς καθορισμένα χρονικά διαστήματα, ανεξάρτητα από οτιδήποτε άλλο συμβαίνει στον κώδικά μας. Καθώς η διαδικασία ενεργοποίησης μιας διακοπής χρονιστή είναι μάλλον πολύπλοκη για τον αρχάριο χρήστη, θα δείξουμε πως μπορούμε να εκμεταλλευτούμε αυτή τη δυνατότητα του Arduino με τη χρήση της εξειδικευμένης βιβλιοθήκης ***FlexiTimer2***, που κάνει χρήση του χρονιστή #2 (Timer2) του μικροελεγκτή, και μπορεί να εγκατασταθεί μέσω του *"Διαχειριστή βιβλιοθήκης"*, κατά τα γνωστά. Η βιβλιοθήκη *FlexiTimer2* έχει τρεις συναρτήσεις μέσω των οποίων υλοποιούνται οι διακοπές (Interrupts) για το χρονιστη #2:

- **set(interval, T2ISR)**: Θέτει το χρονικό διάστημα (*interval* σε ms) μεταξύ δύο διαδοχικών ενεργοποιήσεων της διακοπής, καθώς και τη ρουτίνα (*T2ISR*) που θα εκτελείται κατά την ενεργοποίηση της διακοπής.
- **start()**: Ενεργοποίηση των διακοπών του χρονιστή #2.
- **stop()**: Απενεργοποίηση των διακοπών.

Παράδειγμα διακοπής χρονιστή

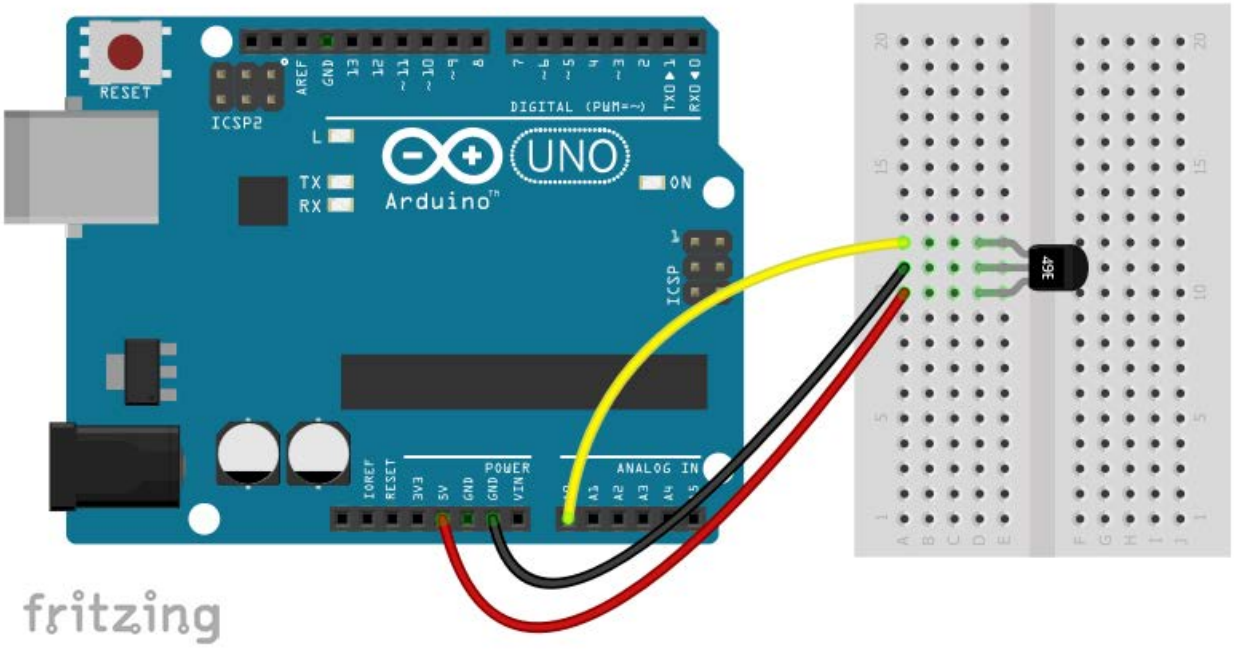
Ας δούμε πως όλα αυτά μπορούν να υλοποιηθούν στην περίπτωση ενός απλού συστήματος μέτρησης μαγνητικού πεδίου. Θα χρησιμοποιήσουμε ένα αισθητήρα μαγνητικού πεδίου (SS49E της Honeywell) για τη μέτρηση του πεδίου ενός μαγνήτη. Η λειτουργία του αισθητήρα, που διατίθεται με τη μορφή ολοκληρωμένου κυκλώματος τριών ακίδων, βασίζεται στο φαινόμενο Hall.



Εικόνα 45: Ο αναλογικός αισθητήρας μαγνητικού πεδίου

Ο κατασκευαστής του αισθητήρα δίνει τα ακόλουθα στοιχεία [11]: Απουσία μαγνητικού πεδίου η τάση στην έξοδο (ακίδα analog OUT) είναι ίση με το μισό της τάσης τροφοδοσίας (τυπικά 2,5 V θεωρώντας την τάση τροφοδοσίας 5V). Η παρουσία νότιου πόλου κάθετα στην επιφάνεια του αισθητήρα αυξάνει την τάση στην έξοδο από την τάση ηρεμίας μέχρι περίπου τα 4V με τυπικό ρυθμό 1,4 mV/Gauss. Αντίστοιχα η παρουσία βόρειου πόλου προκαλεί αναλογική μείωση (μέχρι την τιμή 1V) της τάσης εξόδου του αισθητήρα. Για τη μετατροπή της τιμής που παίρνουμε από τον αναλογικο-ψηφιακό μετατροπέα του Arduino σε τιμή μαγνητικού πεδίου (σε Gauss), πρέπει να σκεφτούμε πως: τα 1024 αναλογικά βήματα του ADC αντιστοιχούν σε τάση 5000 mV και 1 Gauss αντιστοιχεί σε 1,4 mV. Οπότε κάθε αναλογικό βήμα του ADC αντιστοιχεί σε 3,4877 Gauss.

Η σύνδεσή του αισθητήρα στον Arduino είναι εξαιρετικά απλή όπως φαίνεται και από το επόμενο σχήμα.



Εικόνα 46: Σύνδεση αισθητήρα μαγνητικού πεδίου στον Arduino

Θέλουμε να παίρνουμε μετρήσεις του μαγνητικού πεδίου σε συγκεκριμένα χρονικά διαστήματα, ας πούμε 2 μετρήσεις κάθε δευτερόλεπτο, δηλ. να καθορίσουμε ένα σταθερό ρυθμό δειγματοληψίας 2 Hz. Παρότι κάτι τέτοιο θα μπορούσε να επιτευχθεί με χρήση της κατάλληλης καθυστέρησης μεταξύ των διαδοχικών μετρήσεων, αυτή η λύση δεν είναι η ενδεδειγμένη αφού με τη συνάρτηση *delay()* ο μικροελεγκτής σταματά κάθε δραστηριότητα (εκτός των διακοπών) για το καθορισμένο χρονικό διάστημα, δηλ. έχουμε απώλεια σημαντικού χρόνου που θα μπορούσε να χρησιμοποιηθεί από το πρόγραμμά μας για άλλες εργασίες. Η χρήση διακοπών χρονιστή είναι η λύση στο πρόβλημά μας. Το sketch που ακολουθεί δύο φορές κάθε δευτερόλεπτο επιστρέφει θετικές τιμές για την ένταση του μαγνητικού πεδίου όταν νότιος μαγνητικός πόλος προσεγγίζει την πρισματική επιφάνεια του αισθητήρα, και αρνητικές τιμές όταν αντίστοιχα προσεγγίζει βόρειος μαγνητικός πόλος:

```
/*
        Ex.26 : Παράδειγμα διακοπής που ενεργοποιείται από το χρονιστή #2
*/

#include "FlexiTimer2.h"                    // Συμπερίληψη βιβλιοθήκης FlexiTimer2

// Η αναλογική είσοδος στην οποία συνδέεται ο αισθητήρας μαγνητικού πεδίου
#define HALLSENS_PIN A0

// Η πειραματική τιμή που επιστρέφει ο A/D μετατροπέας με τον αισθητήρα εκτός μαγνητικού πεδίου.
#define ADCZERO 505L

// Συντελεστής μετατροπής της τιμής του A/D μετατροπέα σε Gauss
#define ADCVTOGAUSS 3.4877

// Πτητική μεταβλητή που όταν γίνεται true δηλώνει ότι είναι η κατάλληλη στιγμή
// για αποστολή των δεδομένων στη σειριακή κονσόλα
volatile boolean intEnabled = false;
```

```
// Η ρουτίνα διαχείρισης της διακοπής του χρονιστή #2.
// Όταν εκτελείται απλά θέτει την τιμή της μεταβλητής intEnabled σε true
void runADC()
{
        intEnabled = true;
}

void setup()
{
        Serial.begin(115200);                   //Ενεργοποίηση σειριακής επικοινωνίας
        FlexiTimer2::set(500, runADC);          // Η ρουτίνα runADC() θα εκτελείται κάθε 500 ms
                                                // δηλ. ο ρυθμός δειγματοληψίας είναι 2 Hz.
}

void loop()
{
        Serial.print("Send a key to start...");
        while (Serial.available() == 0);        // Αναμονή μέχρι τη σειριακή λήψη κάποιου χαρακτήρα
        FlexiTimer2::start();                   // Ενεργοποίηση της διακοπής του χρονιστή #2
        do                                      // Βρόχος do … while
        {
                float gauss = (analogRead(HALLSENS_PIN) - ADCZERO) * ADCVTOGAUSS;
                if (intEnabled)
                {
                        Serial.print(gauss, 2);
                        Serial.println(" Gauss ");
                        intEnabled = false;
                }
        }
        while (Serial.read() != 'S');           // Επανάληψη μέχρι να ληφθεί ο χαρακτήρας 'S'
        FlexiTimer2::stop();                    // Απενεργοποίηση της διακοπής χρονιστή
}
```

Στη συνάρτηση *setup()* ενεργοποιείται αρχικά η σειριακή επικοινωνία με ρυθμό 115200bps, και μετά καθορίζεται η συνάρτηση που θα εκτελείται καθώς και ο ρυθμός με τον οποίο θα εκτελείται κάθε φορά που ενεργοποιείται η διακοπή του χρονιστή.

Στη συνάρτηση *loop()* αρχικά υπάρχει ένας βρόχος αναμονής μέχρι να ληφθεί σειριακά κάποιος χαρακτήρας. Όταν ο χαρακτήρας ληφθεί, ενεργοποιείται η διακοπή του χρονιστή #2 και ο κώδικας μπαίνει σε ένα *do … while* βρόχο μέχρι να ληφθεί σειριακά ο χαρακτήρας 'S'. Στο βρόχο αυτό γίνεται διαρκώς μετατροπή του αναλογικού σήματος από τον αισθητήρα σε ψηφιακό, και εν συνεχεία μετατροπή του σε μονάδες έντασης μαγνητικού πεδίου. Όταν η πτητική μεταβλητή *intEnabled* γίνει true (δηλ. κάθε 0,5 s ή 500 ms) το αποτέλεσμα των μετατροπών τυπώνεται στη σειριακή κονσόλα, και η μεταβλητή *intEnabled* γίνεται false. Με την έξοδο από το βρόχο η διακοπή του χρονιστή #2 απενεργοποιείται, και ξεκινάει από την αρχή η εκτέλεση του κώδικα της συνάρτησης *loop()*.

Η τιμή (ADCZERO) που επιστρέφει ο αισθητήρας μαγνητικού πεδίου όταν βρεθεί εκτός μα-

γνητικού πεδίου (αλλά στην πραγματικότητα εντός του ασθενούς γήινου μαγνητικού πεδίου), μπορεί να προσδιοριστεί με το επόμενο sketch:

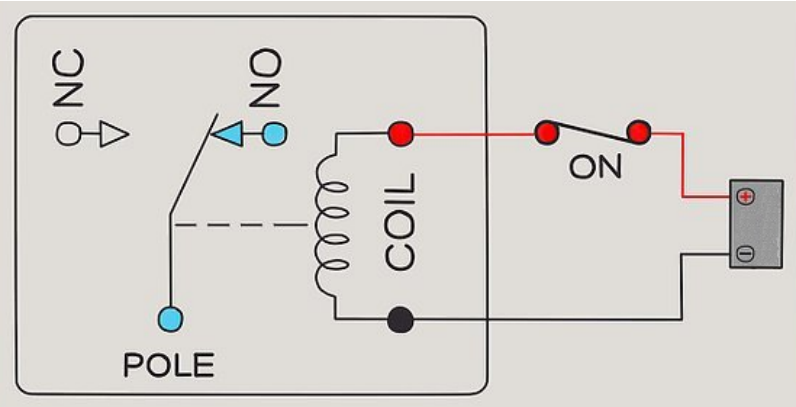
```
/*
        Ex.27 : Ρύθμιση για αισθητήρα μαγνητικού πεδίου SS49E
*/

#define HALLSENS_PIN A0

void setup()
{
        Serial.begin(9600);
}

void loop()
{
        Serial.print ("ADC Value: ");
        Serial.println(analogRead(HALLSENS_PIN));
        delay(1000);
}
```

Παρατήρηση : *Στη θέση του αισθητήρα SS49E μπορούν να χρησιμοποιηθούν και άλλοι αναλογικοί αισθητήρες Hall, με διαφορετικούς όμως συντελεστές μετατροπής της τιμής που επιστρέφει ο αναλογικο-ψηφιακός μετατροπέας σε Gauss, όπως: ο A1302 με συντελεστή μετατροπής 3,756 Gauss/βήμα ή ο A1301 με συντελεστή μετατροπής 1,953 Gauss/βήμα.*

Πηγαίνοντας λίγο μακρύτερα...

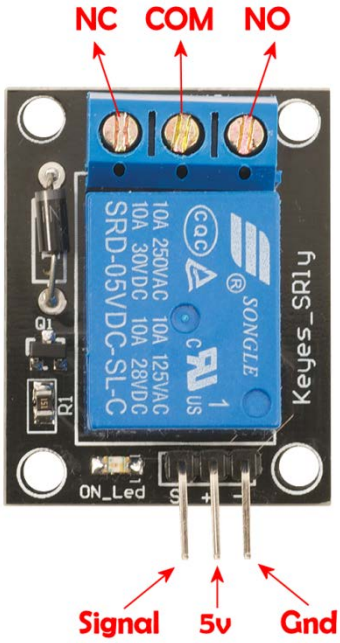
Ο ηλεκτρονόμος ή ρελέ (relay)

Πρόκειται για μια ηλεκτρομηχανική διάταξη που λειτουργεί ως διακόπτης ελεγχόμενος από τάση.



Εικόνα 47: Ο ηλεκτρονόμος ή ρελέ (relay)

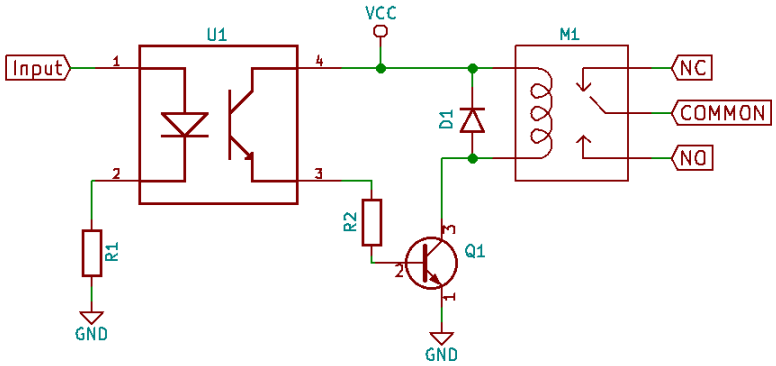
Στην απλούστερη μορφή του πρόκειται για ένα διακόπτη (αποτελούμενο από τις επαφές POLE, NC, NO) ενεργοποιούμενο με τη βοήθεια ενός ηλεκτρομαγνήτη. Όταν ο ηλεκτρομαγνήτης δε διαρρέεται από ρεύμα είναι βραχυκυκλωμένες οι επαφές POLE (ή COMMON) και NC (Normal Closed) του διακόπτη, ενώ όταν ο ηλεκτρομαγνήτης διαρρέεται από ρεύμα διακόπτεται η επαφή POLE - NC, και βραχυκυκλώνονται οι επαφές POLE και NO (Normal Opened). Ουσιαστικά με το ρελέ η ακίδα POLE μπορεί να συνδέεται είτε στην ακίδα NC (όταν το ρελέ είναι απενεργοποιημένο), είτε στην ακίδα NO (όταν το ρελέ είναι ενεργοποιημένο). Αφήνοντας την επαφή NC χωρίς σύνδεση, μπορούμε να συνδέσουμε το ρελέ ως απλό (on-off) διακόπτη σε οποιοδήποτε ηλεκτρικό κύκλωμα, που μπορεί μάλιστα να λειτουργήσει σε μια ευρεία περιοχή τάσεων (μέχρι 250 V AC) και ρευμάτων (μέχρι και 10 A).



Εικόνα 48: Η μονάδα ρελέ

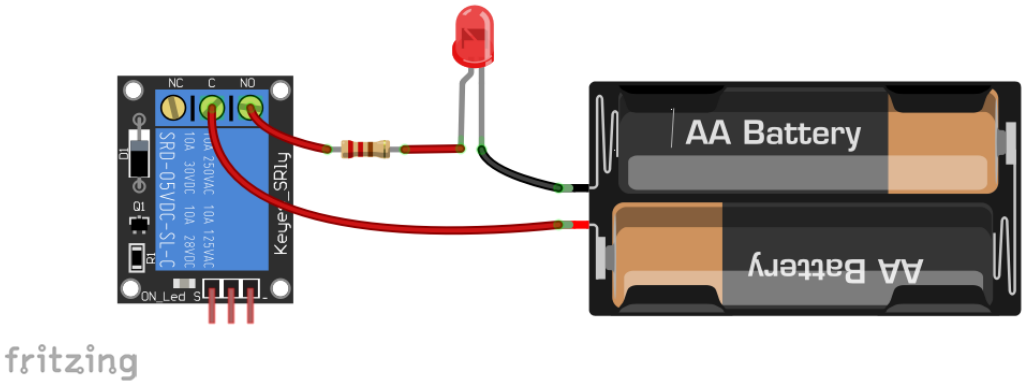
Στη μονάδα (module) ρελέ που πρόκειται να χρησιμοποιήσουμε η ενεργοποίηση του ηλεκτρομαγνήτη γίνεται μέσω μιας ακίδας του (Signal ή Input), που συνδέεται σε μια ψηφιακή έξοδο του

Arduino. Λογικό 1 στην ακίδα Signal (ή Input) ενεργοποιεί τον ηλεκτρομαγνήτη, και λογικό μηδέν τον απενεργοποιεί. Στην εικόνα 49 φαίνεται το κυκλωματικό διάγραμμα της μονάδας του ρελέ:



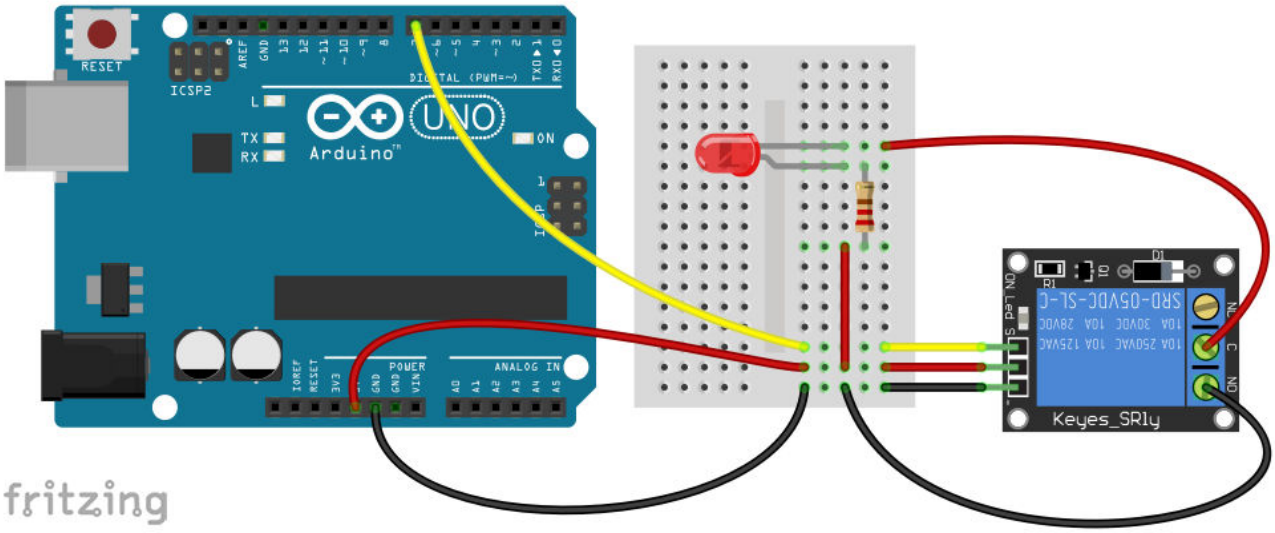
Εικόνα 49: Το κυκλωματικό διάγραμμα της μονάδας ρελέ

Η σύνδεση του ρελέ ως διακόπτη σε ένα σύστημα που περιλαμβάνει σε σειρά μια αντίσταση (τιμής από 220-1000 Ω), ένα LED, και ένα σύστημα μπαταριών συνολικής τάσης 3V (ή μια πλακέ μπαταρία των 4,5V) φαίνεται στην Εικόνα 50. Ενεργοποίηση του ρελέ έχει ως αποτέλεσμα το άναμμα του LED, ενώ με την απενεργοποίηση του ρελέ το LED σβήνει.



Εικόνα 50: Το ρελέ ως διακόπτης

Για την ενεργοποίηση του ρελέ μπορούμε να χρησιμοποιήσουμε τον Arduino, συνδέοντας τις ακίδες (+ ή 5v) και (- ή Gnd) της μονάδας του ρελέ με τις αντίστοιχες ακίδες του Arduino, και την ακίδα (S ή Signal ή Input) της μονάδας ρελέ σε μια ψηφιακή ακίδα (π.χ. την 7) του Arduino. Για την απλοποίηση του κυκλώματος δε θα χρησιμοποιήσουμε εξωτερική μπαταρία για την τροφοδοσία του LED, αλλά θα το τροφοδοτήσουμε από τις ακίδες 5V και GND της πλακέτας του Arduino.



Εικόνα 51: Σύνδεση της μονάδας ρελέ στον Arduino

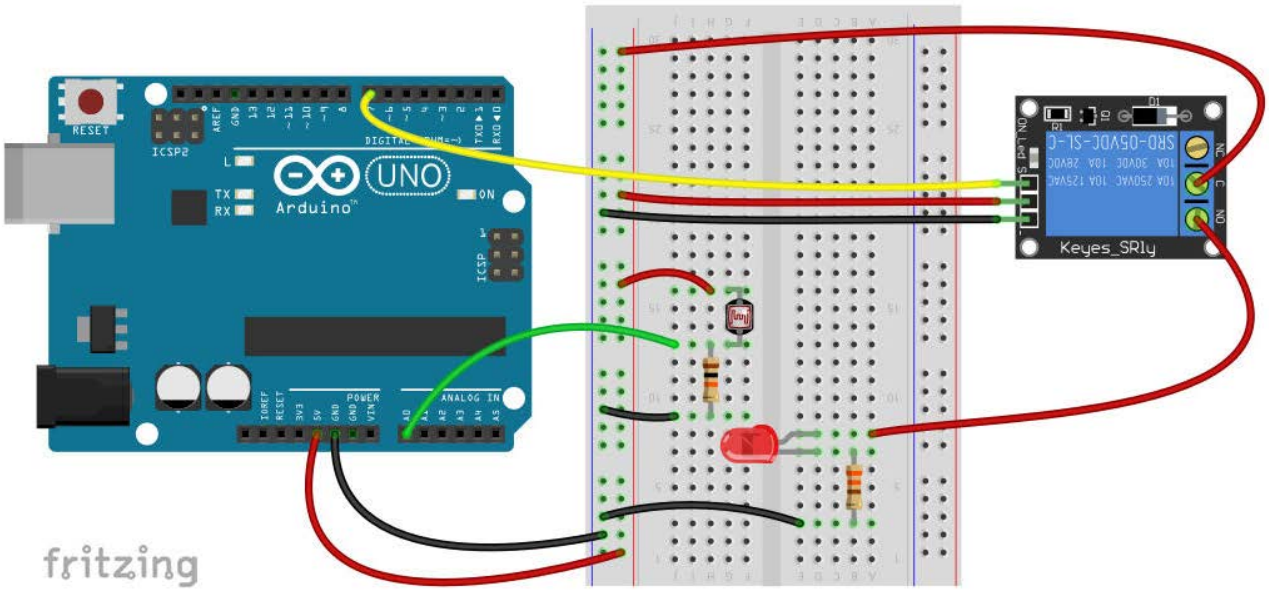
Το σχετικό sketch για τη δοκιμή του ρελέ μπορεί να έχει τη μορφή:

```
/*
        Ex.28 : Αναβοσβήνουμε ένα LED με τη βοήθεια του ρελέ
*/

#define RLPIN 7

void setup ()
{
        pinMode (RLPIN, OUTPUT);        // Η ακίδα RLPIN καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
        digitalWrite (RLPIN, HIGH);     // Ενεργοποίηση το relay
        delay (1000);                   // Αναμονή 1sec (1000 milliseconds)
        digitalWrite (RLPIN, LOW);      // Απενεργοποίηση του relay
        delay (1000);                   // Αναμονή 1sec
}
```

Κάποιος παρατηρητικός θα έχει ήδη καταλάβει πως το προηγούμενο sketch λειτουργικά είναι ακριβώς ίδιο με το δεύτερο παράδειγμα του βιβλίου (σελ. 16), με το οποίο αναβοσβήναμε ένα LED που είχε συνδεθεί σε κάποια ψηφιακή ακίδα (π.χ την ακίδα 7) του Arduino. Μπορούμε, δηλαδή, να πούμε πως η παρούσα εκδοχή κάνει κακή χρήση των πόρων του Arduino, αφού το ίδιο αποτέλεσμα μπορεί να επιτευχθεί με πολύ απλούστερο τρόπο, και συνεπώς μόνο διδακτική αξία μπορεί να έχει για την επίδειξη της λειτουργίας του ρελέ.

Ένας απλός αυτοματισμός με ηλεκτρονόμο

Συνδυάζοντας το κύκλωμα του ρελέ για το αναβόσβημα του LED με μια φωτοαντίσταση, εύκολα δημιουργούμε το κύκλωμα ενός απλού αυτοματισμού, με τον οποίο το ρελέ ενεργοποιείται ή απενεργοποιείται (και συνεπώς ανάβει ή σβήνει το LED) ανάλογα με την ένταση του φωτός που προσπίπτει στη φωτοαντίσταση. Οι συνδέσεις με τον Arduino φαίνονται στο ακόλουθο σχήμα:



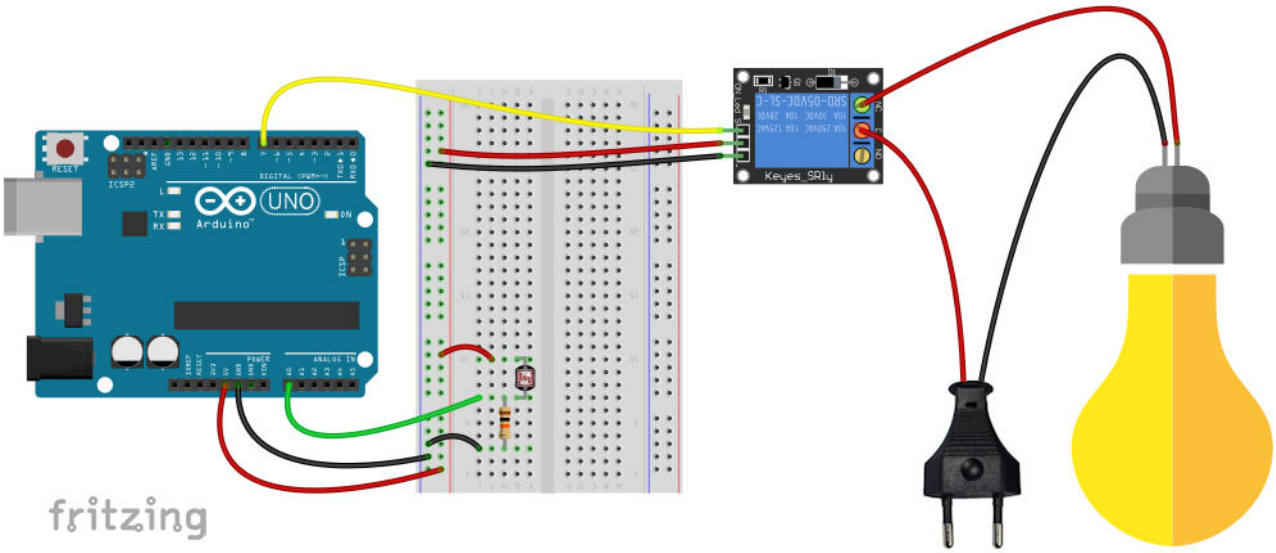
Εικόνα 52: Σύνδεση ρελέ, LED και φωτοαντίστασης στον Arduino

Το σχετικό sketch έχει τη μορφή:

```
/*
        Ex.29 : Απλός αυτοματισμός
*/

void setup ()
{
        pinMode (7, OUTPUT);            // Η ακίδα 7 καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
        int value = analogRead(A0);     // "Διαβάζουμε" την ένταση του φωτός
        delay(500);                     // Μικρή αναμονή πριν τη λήψη απόφασης
        if (value < 400)                // Αν είναι σκοτάδι (η τιμή 400 προέκυψε από δοκιμές)
        {
                digitalWrite (7, HIGH); // Ενεργοποίηση το relay και άναμμα του led
        }
        else                            // Αλλιώς
        {
                digitalWrite (7, LOW);  // Απενεργοποίηση του relay και σβήσιμο του led
        }
}
```

Στο βρόχο *loop()*, πρώτα "διαβάζουμε" από την αναλογική είσοδο A0 την τρέχουσα τιμή σχετικά με τη φωτεινότητα, και μετά από 0,5 sec ανάλογα με την τιμή της ενεργοποιούμε ή απενεργοποιούμε το ρελέ, ανάβοντας ή σβήνοντας αντίστοιχα το LED. Η τιμή από την αναλογική είσοδο A0 στην οποία θα γίνεται η αλλαγή κατάστασης του ρελέ (400 στην περίπτωση του sketch) πρέπει να προσδιοριστεί πειραματικά (μικρότερες τιμές σημαίνουν μικρότερη φωτεινότητα).

Αντί για LED που αναβοσβήνει ανάλογα με την ένταση του φωτισμού στην φωτοαντίσταση, μπορούμε να χρησιμοποιήσουμε μια λάμπα των 220V, λαμβάνοντας όλες τις απαραίτητες προφυλάξεις, αφού η τάση των 220V είναι επικίνδυνη για την ανθρώπινη ζωή.



Εικόνα 53: Κύκλωμα απλού αυτοματισμού

Θα κάνουμε και κάποιες βελτιώσεις στον κώδικα, αφού: Αν έχει πέσει σκοτάδι και η λάμπα έχει

ήδη ανάψει (δηλ. έχει ενεργοποιηθεί το ρελέ), δε χρειάζεται να ξανα-ενεργοποιηθεί το ρελέ για όσο χρόνο είναι ακόμη σκοτάδι. Αντίστοιχα δεν υπάρχει λόγος να απενεργοποιήσουμε ένα ήδη απενεργοποιημένο ρελέ όσο είναι ακόμη ημέρα. Στην αντίθετη περίπτωση ο Arduino εκτελεί πολύ μεγάλο αριθμό περιττών ενεργειών. Για να λύσουμε αυτό το πρόβλημα, απαιτείται:

1. Να γνωρίζουμε κάθε στιγμή την κατάσταση της λάμπας. Αυτό το επιτυγχάνουμε με τον ορισμό μιας *global* λογικής μεταβλητής (τύπου *bool*), που της δώσαμε το όνομα *lampState*, και παίρνει μόνο δύο τιμές: *false* αν η λάμπα είναι σβηστή, και *true* αν είναι αναμμένη. Αρχικά λοιπόν, που η λάμπα είναι σβηστή, η μεταβλητή *lampState* έχει τιμή *false*, ενώ της δίνεται η τιμή *true* (lampState = true) όταν το ρελέ ενεργοποιηθεί και ανάψει η λάμπα, και πάλι τη τιμή *false* (lampState = false) όταν το ρελέ απενεργοποιηθεί και σβήσει η λάμπα.
2. Να κάνουμε ορισμένους επιπλέον ελέγχους. Το ρελέ πρέπει να ενεργοποιείται (και να ανάβει η λάμπα) μόνο αν έχει πέσει σκοτάδι και η λάμπα είναι σβηστή. Αυτός ο έλεγχος προγραμματιστικά υλοποιείται μέσω της εντολής λήψης απόφασης:

   if ((value < 400) && (lampState == false))

   Η εντολή περιλαμβάνει δύο επιμέρους συνθήκες συνδεδεμένες μεταξύ τους με τον λογικό τελεστή **and** (**&&**), που σημαίνει ότι για να εκτελεστεί το αντίστοιχο μπλοκ εντολών (για το άναμμα της λάμπας) πρέπει **και** οι δύο συνθήκες να είναι αληθείς (true). Η πρώτη συνθήκη (value < 400) είναι η συνθήκη που όταν είναι αληθής σημαίνει ότι έχει αρχίσει να πέφτει το σκοτάδι, και η δεύτερη (lampState == false) όταν είναι αληθής σημαίνει πως η λάμπα είναι σβηστή. Όταν η συνολική συνθήκη του *if* είναι ψευδής (false) εκτελείται το μπλοκ εντολών που ακολουθεί μετά το *else*, και το οποίο περιλαμβάνει ένα δεύτερο *if* με επίσης δύο επιμέρους συνθήκες συνδεδεμένες με λογικό **and**. Η πρώτη συνθήκη (value >= 400) είναι αληθής από όταν αρχίζει να ξημερώνει και μετά, και η δεύτερη (lampState == true) όταν η λάμπα είναι ήδη αναμμένη. Συνεπώς όταν και οι δύο συνθήκες είναι αληθείς (true) εκτελείται το μπλοκ εντολών με το οποίο σβήνει η λάμπα. Σε όλες τις άλλες περιπτώσεις (π.χ. όταν έχει αρχίσει να σκοτεινιάζει αλλά η λάμπα έχει ήδη ανάψει ή όταν έχει ξημερώσει αλλά η λάμπα είναι σβηστή) ο Arduino δεν εκτελεί κάποια συγκεκριμένη ενέργεια, αλλά επαναλαμβάνει τον κύκλο εντολών της συνάρτησης *loop()*. Περισσότερα για τους λογικούς τελεστές στο Παράρτημα Β.

Το σχετικό sketch διαμορφώνεται ως εξής:

```
/*
      Ex.30 : Αυτόματο άναμμα και σβήσιμο λάμπας
*/

bool lampState = false;                  // Για να γνωρίζουμε την κατάσταση της λάμπας

void setup ()
{
      pinMode (7, OUTPUT);               // Η ακίδα 7 καθορίζεται ως ψηφιακή έξοδος
}

void loop ()
{
      int value = analogRead(A0);        // "Διαβάζουμε" την ένταση του φωτός

      // Αν έχει πέσει σκοτάδι και η λάμπα είναι σβηστή
      if ((value < 400) && (lampState == false))
      {
            delay(500);                  // Μικρή αναμονή πριν την ενεργοποίηση
            digitalWrite (7, HIGH);      // Ενεργοποίηση το relay και άναμμα της λάμπας
```

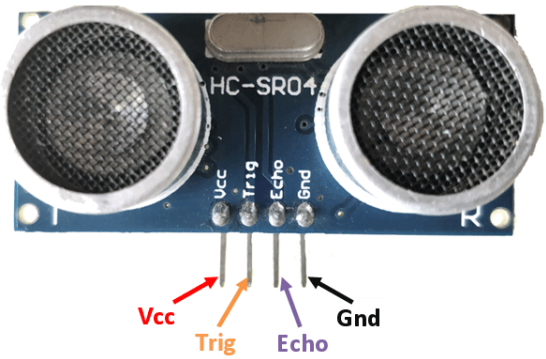
```
            lampState = true;               // Η λάμπα είναι αναμμένη
        }
        else if ((value >= 400) && (lampState == true))             // Αλλιώς
        {
            delay(500);                     // Μικρή αναμονή πριν την απενεργοποίηση
            digitalWrite (7, LOW);          // Απενεργοποίηση του relay και σβήσιμο της λάμπας
            lampState = false;              // Η λάμπα είναι σβηστή
        }
}
```

Για την ασφαλή και ορθή λειτουργία της διάταξης πρέπει να λάβουμε ιδιαίτερες προφυλάξεις, όπως:

1. Το ρελέ να τοποθετηθεί μέσα σε πλαστικό κουτί, ώστε να αποφευχθεί η κατά λάθος επαφή με τους ακροδέκτες που φέρουν την υψηλή τάση των 220V.
2. Η φωτοαντίσταση πρέπει να τοποθετηθεί σε κατάλληλη θέση, ώστε να "αισθάνεται" τον περιβάλλοντα φωτισμό, αλλά να μην επηρεάζεται από τη φωτοβολία της λάμπας που ανάβει όταν ενεργοποιείται το ρελέ.

Ο αισθητήρας υπερήχων HC-SR04

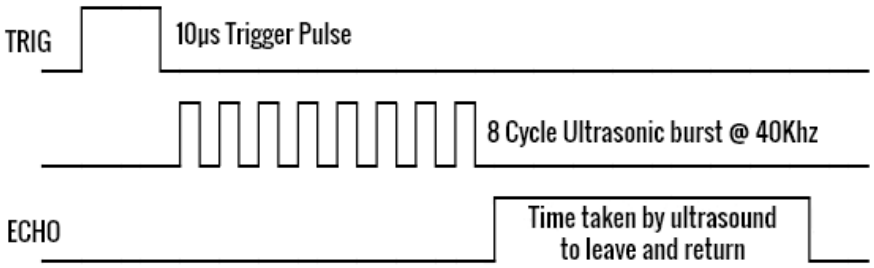
Ο αισθητήρας HC-SR04 ενσωματώνει σε μία μονάδα ένα πομπό και ένα δέκτη υπερήχων (συχνότητας 40 kHz), καθώς και τα κατάλληλα κυκλώματα ελέγχου.



Εικόνα 54: Ο αισθητήρας υπερήχων HC-SR04

Η μονάδα του αισθητήρα διαθέτει τέσσερεις ακίδες για την τροφοδοσία και την επικοινωνία με τον Arduino:

- VCC που συνδέεται στην ακίδα 5V του Arduino,
- GND που συνδέεται στην αντίστοιχη ακίδα του Arduino,
- Trig (Trigger) και Echo (Receive) που συνδέονται σε δύο διαφορετικές ψηφιακές ακίδες στον Arduino.



Εικόνα 55: Κύκλος λειτουργίας του αισθητήρα HC-SR04

Η λειτουργία του αισθητήρα HC-SR04 συνοψίζεται ως εξής:

1. Ο αισθητήρας ξεκινάει ένα κύκλο μέτρησης αν η ακίδα του Trig τεθεί σε υψηλή λογική στάθμη για χρόνο τουλάχιστον 10 μs.
2. Ο πομπός αυτόματα εκπέμπει 8 μικρού εύρους παλμούς υπερήχων συχνότητας 40 kHz, και αμέσως μετά θέτει την ακίδα Echo σε υψηλή λογική στάθμη. Όταν αυτοί οι παλμοί προσπέσουν σε κάποιο αντικείμενο που βρίσκεται στη γειτονιά του αισθητήρα ανακλώνται και επιστρέφουν πίσω.
3. Η ακίδα Echo παραμένει σε υψηλή λογική στάθμη μέχρι να ανιχνευθεί το ανακλώμενο σήμα από τον δέκτη του αισθητήρα.

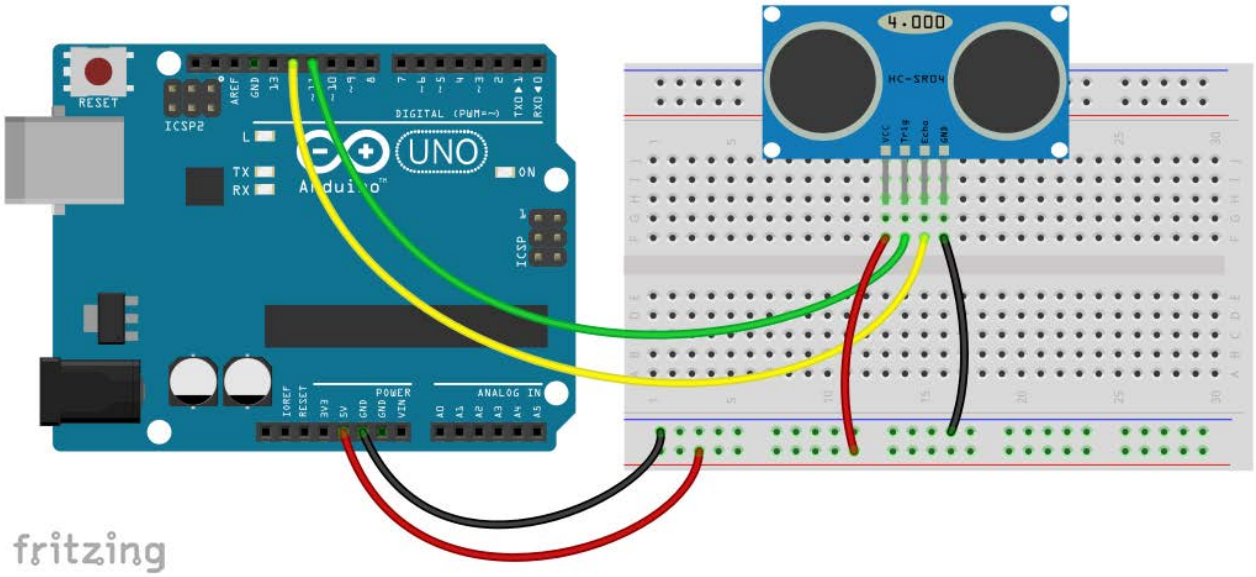
Με τη βοήθεια του Arduino μπορούμε κατ' αρχάς να υπολογίσουμε το χρονικό διάστημα που μεσολαβεί ανάμεσα στην εκπομπή και τη λήψη του σήματος. Αν γνωρίζουμε την απόσταση ανάμεσα στον αισθητήρα και στην ανακλαστική επιφάνεια μπορούμε να υπολογίσουμε την ταχύτητα διάδοσης του ήχου, ή αντίθετα θεωρώντας γνωστή την ταχύτητα διάδοσης του ήχου (343 m/s στους 20 °C στον ξηρό αέρα) μπορούμε να υπολογίσουμε την απόσταση της ανακλαστικής επιφάνειας από τον αισθητήρα. Οι υπολογισμοί γίνονται με βάση την απλή σχέση:

$$(\text{απόσταση}) = (\text{ταχύτητα}) \times (\text{χρόνος})$$

Στη σχέση αυτή ο χρόνος είναι ίσος με το μισό αυτού που υπολογίζουμε με τον Arduino, αφού ο

χρόνος ανάμεσα στην εκπομπή και τη λήψη του σήματος αντιστοιχεί στο χρόνο που απαιτείται για να πάει το σήμα από τον αισθητήρα στην ανακλαστική επιφάνεια και να επιστρέψει. Ο αισθητήρας παρέχει λειτουργίες μέτρησης από 2 cm μέχρι 400 cm, ενώ η ακρίβεια των μετρήσεων μπορεί να φτάσει μέχρι τα 3 mm.

Για τις ανάγκες των δοκιμών μας θα συνδέσουμε τις ακίδες Trig και Echo του αισθητήρα στις ψηφιακές ακίδες 11 και 12 του Arduino αντίστοιχα, όπως φαίνεται στην επόμενη εικόνα.



Εικόνα 56: Σύνδεση του αισθητήρα υπερήχων στον Arduino

Θα χρησιμοποιήσουμε τον αισθητήρα υπερήχων για να προσδιορίσουμε την ταχύτητα του ήχου στον αέρα, και στις συνθήκες που επικρατούν κατά τη διάρκεια του πειράματος. Για το σκοπό αυτό θα τοποθετήσουμε μια ανακλαστική επιφάνεια μπροστά από τον αισθητήρα και σε κάποια απόσταση (50 – 100 cm), την οποία θα μετρήσουμε με ακρίβεια με τη βοήθεια μετροταινίας. Προσοχή πρέπει να δοθεί, ώστε να μην υπάρχουν άλλα αντικείμενα στη γειτονιά του αισθητήρα προς αποφυγή ανεπιθύμητων ανακλάσεων από αυτά, καθώς ο αισθητήρας παρουσιάζει ένα εύρος πεδίου περί τις 15°.

Όμως, για να εκμεταλλευτούμε τις δυνατότητες του αισθητήρα στον Arduino θα πρέπει κατ' αρχάς να εγκαταστήσουμε την κατάλληλη βιβλιοθήκη λογισμικού. Με τη βοήθεια του *"Διαχειριστή βιβλιοθήκης"* (που εκτελείται μέσω του μενού *"Εργαλεία\Διαχείριση βιβλιοθηκών"* ή του *"Σχέδιο\Συμπερίληψη βιβλιοθήκης\Διαχείριση βιβλιοθηκών"*) αναζητούμε και εγκαθιστούμε τη βιβλιοθήκη *"NewPing"*, που περιλαμβάνει όλο τον απαραίτητο κώδικα για τη διαχείριση του αισθητήρα. Για παράδειγμα:

- Η συνάρτηση *ping()* επιστρέφει το χρονικό διάστημα από την εκπομπή μέχρι τη λήψη του ανακλώμενου σήματος.
- Η συνάρτηση *ping_cm()* επιστρέφει την απόσταση (σε cm) ανάμεσα στον αισθητήρα και την ανακλαστική επιφάνεια.

Αφού δημιουργήσουμε ένα νέο sketch, πρέπει να δηλώσουμε την συμπερίληψη του κώδικα από τη βιβλιοθήκη. Αυτό μπορεί να γίνει είτε αυτόματα (μέσω του μενού *"Σχέδιο\Συμπερίληψη βιβλιοθήκης"*), είτε χειροκίνητα, εισάγοντας αμέσως μετά τα αρχικά σχόλια τη σχετική δήλωση συμπερίληψης:

```
#include <NewPing.h>
```

Στη συνέχεια πρέπει να δημιουργήσουμε ένα (προγραμματιστικό) αντικείμενο κλάσης *NewPing*,

μέσω του οποίου θα αποκτήσουμε πρόσβαση στον κώδικα και τα δεδομένα της βιβλιοθήκης, ενώ για τη ρύθμιση των αρχικών παραμέτρων του αντικειμένου απαιτείται να δηλώσουμε τις ψηφιακές ακίδες του Arduino που χρησιμοποιούνται για τη σύνδεση του αισθητήρα:

```
#define TRIGGER_PIN  11   // Ακίδα του Arduino που συνδέεται στην ακίδα Trigger του αισθητήρα
#define ECHO_PIN     12   // Ακίδα του Arduino που συνδέεται στην ακίδα Echo του αισθητήρα

NewPing us_sensor(TRIGGER_PIN, ECHO_PIN); // Δημιουργία-ρύθμιση αντικειμένου NewPing
```

Το sketch τελικά αποκτά τη μορφή:

```
/*
      Ex.31 : Παράδειγμα χρήσης του αισθητήρα υπερήχων
*/

#include <NewPing.h>          // Συμπερίληψη βιβλιοθήκης NewPing

#define TRIGGER_PIN 11    // Ακίδα του Arduino που συνδέεται στην ακίδα Trigger του αισθητήρα
#define ECHO_PIN    12    // Ακίδα του Arduino που συνδέεται στην ακίδα Echo του αισθητήρα

NewPing us_sensor(TRIGGER_PIN, ECHO_PIN); // Δημιουργία-ρύθμιση αντικειμένου NewPing

void setup()
{
      Serial.begin(9600);     // Ενεργοποίηση σειριακής επικοινωνίας στα 9600 bps
}

void loop()
{
      Serial.print("Time: ");
      Serial.print(us_sensor.ping());          // χρόνος σε μs
      Serial.println(" us");

      delay(1000);                             // Αναμονή  1s
}
```

Για να προσδιορίσουμε την ταχύτητα του ήχου με βάση τις τιμές χρόνου που επιστρέφει ο Arduino, μπορούμε να εργαστούμε ως εξής:

1. Υπολογίζουμε το μέσο όρο από πέντε τουλάχιστον διαδοχικές μετρήσεις που επιστρέφει στη σειριακή κονσόλα ο Arduino, και διαιρούμε αυτή την τιμή διά 2, αφού όπως έχουμε ήδη πει ο χρόνος που επιστρέφει ο Arduino είναι αυτός που απαιτείται για να πάει ο ήχος (υπέρηχος για να είμαστε ακριβείς) από τον αισθητήρα στην ανακλαστική επιφάνεια και να επιστρέψει.
2. Με το χρόνο που υπολογίσαμε στο προηγούμενο βήμα, και τη μετρημένη απόσταση ανάμεσα στον αισθητήρα και την ανακλαστική επιφάνεια, υπολογίζουμε την ταχύτητα του ήχου με την εξίσωση:

$$ταχύτητα = \frac{απόσταση}{χρόνος}$$

Βέβαια ο αισθητήρας μπορεί να χρησιμοποιηθεί και για τον υπολογισμό της απόστασης ανάμεσα

στον αισθητήρα και κάποιο αντικείμενο. Το σχετικό sketch, μπορεί να έχει τη μορφή:

```
/*
        Ex.32 : Μέτρηση απόστασης με τον αισθητήρα υπερήχων
*/

#include <NewPing.h>          // Συμπερίληψη βιβλιοθήκης NewPing

#define TRIGGER_PIN 11    // Ακίδα του Arduino συνδεδεμένη στην ακίδα Trigger του αισθητήρα
#define ECHO_PIN      12    // Ακίδα του Arduino συνδεδεμένη στην ακίδα Echo του αισθητήρα

NewPing us_sensor(TRIGGER_PIN, ECHO_PIN); // Δημιουργία-ρύθμιση αντικειμένου NewPing

void setup() {
        Serial.begin(9600);     // Ενεργοποίηση σειριακής επικοινωνίας στα 9600 bps
}

void loop() {
        Serial.print("Distance: ");
        Serial.print(us_sensor.ping_cm());     // Απόσταση σε cm
        Serial.println(" cm");

        delay(1000);                                  // Αναμονή  1s
}
```

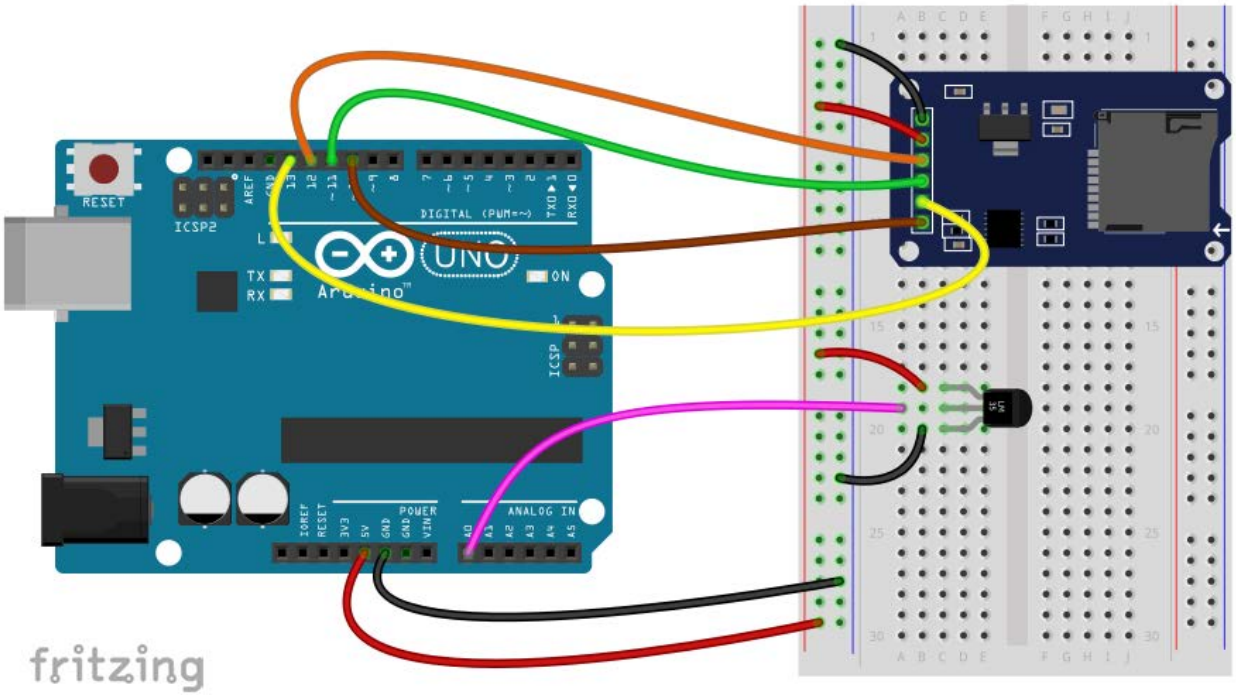
Ένας σημαντικός παράγοντας σφαλμάτων κατά τη μέτρηση αποστάσεων με τον αισθητήρα υπερήχων είναι το γεγονός πως η ταχύτητα του ήχου στον αέρα δεν είναι σταθερή, αλλά εξαρτάται και από τη θερμοκρασία, σύμφωνα με την εξίσωση [12]:

$$\upsilon_{\eta\chi.} = 20,05\sqrt{\theta + 273,15} \quad m/s \ (\theta \ \sigma\varepsilon \ ^\circ C)$$

Η μέτρηση της θερμοκρασίας με κατάλληλο αισθητήρα (π.χ. τον αναλογικού τύπου LM35), μπορεί να οδηγήσει σε ακριβέστερους υπολογισμούς για την ταχύτητα του ήχου, και συνεπώς σε ακριβέστερες τιμές για τις αποστάσεις που μετράμε.

Ένα απλό σύστημα καταγραφής δεδομένων (Data Logger)

Θα χρησιμοποιήσουμε το αναλογικού τύπου θερμόμετρο LM35, καθώς και μία μονάδα κάρτας SD για να φτιάξουμε ένα απλό σύστημα συλλογής και καταγραφής σε αρχείο δεδομένων της θερμοκρασίας περιβάλλοντος. Οι συνδέσεις στον Arduino φαίνονται στην ακόλουθη εικόνα:



Εικόνα 57: Απλό σύστημα συλλογής και καταγραφής δεδομένων

Το σχετικό sketch μπορεί να έχει τη μορφή:

```
/*
        Ex.33 : Data Logger
*/

#include <SD.h>                     // Συμπερίληψη βιβλιοθήκης SD
#include <SPI.h>                    // Συμπερίληψη βιβλιοθήκης SPI

File myFile;                        // Δημιουργία αντικειμένου τύπου File
int startTime = 0;                  // Ο χρόνος έναρξης της εκτέλεσης του κώδικα
char filename[] = "datafile.txt";   // Το όνομα του αρχείου

void setup()
{
        Serial.begin(9600);
        pinMode(SS, OUTPUT);        // Η ακίδα 10 (SS) πρέπει να οριστεί ως έξοδος
        if (SD.begin())             // Αν υπάρχει SD κάρτα που μπορεί να αρχικοποιηθεί
        {
                if (SD.exists(filename))            // Αν ήδη υπάρχει το αρχείο στην κάρτα
                {
                        SD.remove(filename);        // διάγραψέ το
                }
                myFile = SD.open(filename, FILE_WRITE);   // Δημιουργία αρχείου
```

```
            if (myFile)                         // Αν το αρχείο δημιουργήθηκε επιτυχώς
            {
                myFile.println("Time (s)\tTemperature(°C) ");   // Εγγραφή επικεφαλίδας

                Serial.println("Data Logger started…\n");
                Serial.println("Send a key to stop…\n");

                // Η μεταβλητή startTime παίρνει τιμή ίση με το χρόνο (σε msecs )
                // που πέρασε από τη στιγμή που άρχισε η εκτέλεση του sketch
                startTime = millis();
            }
        }
        else                    // Αν η κάρτα SD δεν υπάρχει ή δε μπορεί να αρχικοποιηθεί
        {
            Serial.println("No SD Card found!!!");
            while(1);       // Ατέρμων βρόχος
        }
}

void loop()
{
        long tmval = (millis() - startTime)/1000;      //  Η χρονική στιγμή λήψης της μέτρησης

        //Λήψη της θερμοκρασίας από τον αισθητήρα LM35
        float val = ((5.0 * analogRead(A0)) / 1024) * 100.0;

        // Με το επόμενο μπλοκ εντολών τα αποτελέσματα (χρόνος και θερμοκρασία)
        // εγγράφονται σε μια γραμμή στο αρχείο στην κάρτα SD
        myFile.print(tmval);            // Εγγράφει τη χρονική στιγμή
        myFile.print("\t");             // Εγγράφει το χαρακτήρα ελέγχου tab
        myFile.println(val);            // Εγγράφει τη θερμοκρασία

        if (Serial.available() > 0)     // Αν έχει αποσταλεί ένας χαρακτήρας σειριακά
        {
            myFile.close();             // Κλείσιμο αρχείου
            readFromFile(filename);     // Εκτύπωση των δεδομένων από το αρχείο
            while(1);                   // Ατέρμων βρόχος
        }
        delay(1000);                    // Αναμονή 1sec για τη λήψη της επόμενης μέτρησης
}

void readFromFile(char * filename)          // Άνοιγμα αρχείου για διάβασμα δεδομένων
{
        File myfile;            // Δημιουργία αντικειμένου τύπου File
        String line = "";       // Για την αποθήκευση δεδομένων από το αρχείο

        // Άνοιγμα αρχείου για ανάγνωση
        myfile = SD.open(filename, FILE_READ);
        Serial.println("Reading from file...");
        while (myfile.available() != 0)             // Αν υπάρχουν διαθέσιμα δεδομένα
        {
```

```
            line = myfile.readStringUntil('\n');    // διάβασέ τα
            Serial.println(line);                   // και τύπωσέ τα στη σειριακή κονσόλα
        }
        myfile.close();                 // Κλείσιμο του αρχείου
        Serial.println();
}
```

Σύμφωνα με το φυλλάδιο δεδομένων του αισθητήρα LM35, η τάση στην ακίδα εξόδου του είναι ευθέως ανάλογη της θερμοκρασίας [13], ισχύει δηλαδή:

$$V_{out} = \lambda \cdot \theta$$

όπου V_{out} η τάση στην έξοδο σε mV, θ η θερμοκρασία σε °C, και λ η σταθερά αναλογίας με τιμή 10 mV/°C. Συνεπώς για να υπολογίσουμε τη θερμοκρασία πρέπει:

- Να μετατρέψουμε την τιμή που επιστρέφει ο μετατροπέας αναλογικού σε ψηφιακό του Arduino σε τάση (mV), ως εξής: $Voltage = \left(\frac{5.0}{1024}\right) \times \left(analog\,\mathrm{Re}\,ad(A0)\right) \times 1000.0$.
- Στη συνέχεια να διαιρέσουμε την τάση εξόδου με τη σταθερά αναλογίας λ, ώστε να υπολογίσουμε τη θερμοκρασία: $\theta(^{\circ}C) = \frac{Voltage}{\lambda} = \left(\frac{5.0}{1024}\right) \times \left(analog\,\mathrm{Re}\,ad(A0)\right) \times 100.0$.

Έτσι εξηγείται η εντολή: *float val = ((5.0 * analogRead(A0)) / 1024) * 100.0* που χρησιμοποιήσαμε στο sketch για τον υπολογισμό της τιμής της θερμοκρασίας. Επιπλέον γίνεται φανερό και ένα μειονέκτημα του αισθητήρα LM35: Επειδή η τάση στην έξοδό του δε μπορεί να πάρει αρνητικές τιμές, δε μπορεί να χρησιμοποιηθεί για μέτρηση θερμοκρασιών κάτω του μηδενός. Αν είναι επιθυμητή η μέτρηση τέτοιων θερμοκρασιών πρέπει να χρησιμοποιηθεί είτε ένας ψηφιακός αισθητήρας π.χ. ο DS18B20 για το δίαυλο 1-Wire, είτε κάποιος πιο ευέλικτος αναλογικός αισθητήρας, π.χ. ο TMP36.

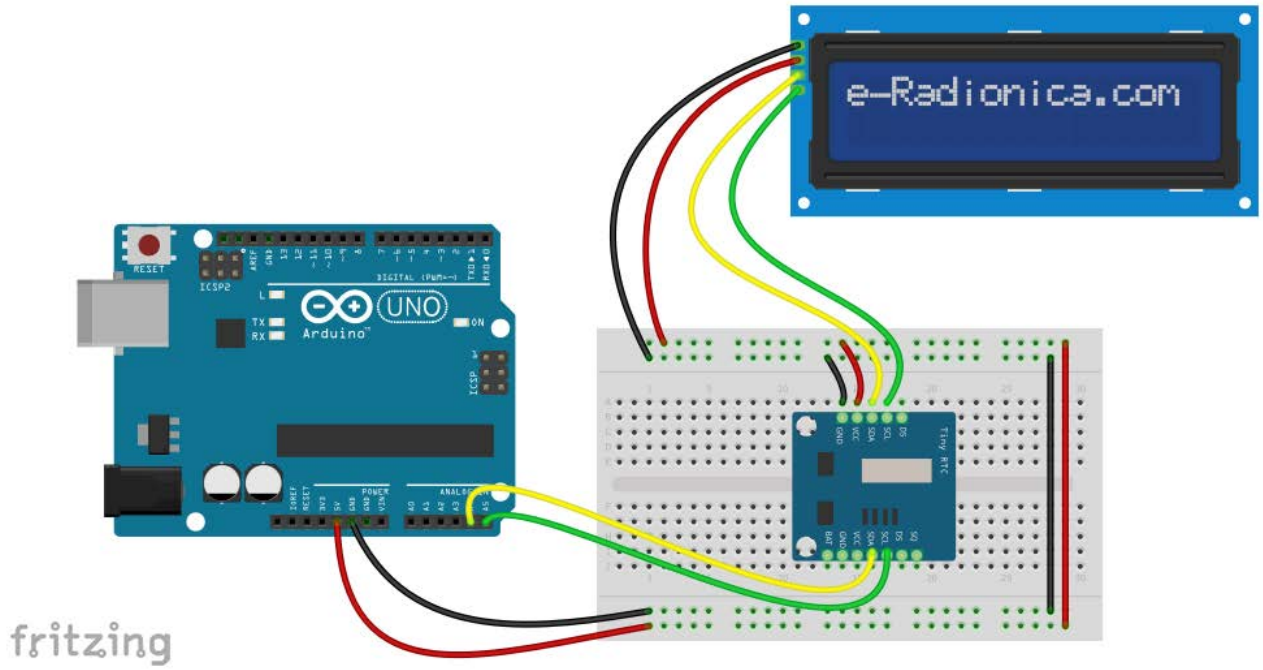
Παρατήρηση : *Η συνάρτηση* **millis()** *επιστρέφει το χρόνο (σε msecs) που πέρασε από τότε που άρχισε η εκτέλεση του κώδικα στον Arduino. Η τιμή του ίδιου χρόνου σε μικροδευτερόλεπτα (μs) υπολογίζεται -με θεωρητική ακρίβεια 4μs- μέσω της συνάρτησης* **micros()** *[4].*

Ένα ρολόι πραγματικού χρόνου με οθόνη υγρών κρυστάλλων

Θα χρησιμοποιήσουμε τη μονάδα Tiny RTC σε συνδυασμό με μια οθόνη υγρών κρυστάλλων για σύνδεση μέσω του διαύλου I^2C. Για το πρότζεκτ αυτό θα θεωρήσουμε πως:

- η μονάδα Tiny RTC αναγνωρίζεται στη διεύθυνση 0x68 στο δίαυλο I^2C, και
- η μονάδα LCD στη διεύθυνση 0x27.

Οι σχετικές συνδέσεις φαίνονται στην επόμενη εικόνα:



Εικόνα 58: Ρολόι πραγματικού χρόνου με οθόνη υγρών κρυστάλλων

Το αντίστοιχο sketch μπορεί να έχει τη μορφή:

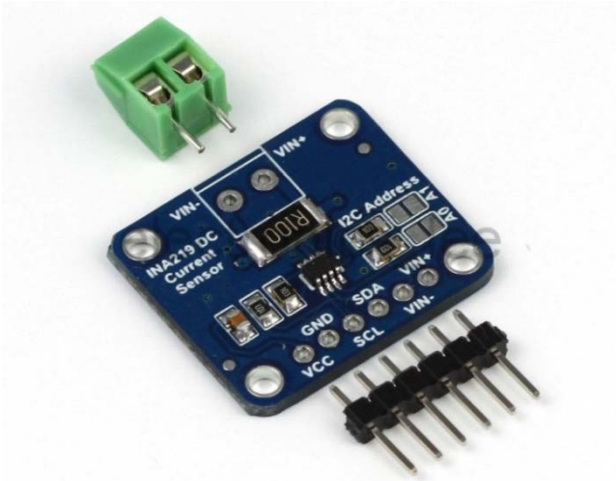
```
/*
        Ex.34 : Ρολόι πραγματικού χρόνου με οθόνη υγρών κρυστάλλων
*/

#include <Wire.h>                       // Συμπερίληψη της βιβλιοθήκης Wire
#include <RTClib.h>                     // Συμπερίληψη της βιβλιοθήκης RTCLib
#include <LiquidCrystal_I2C.h>          // Συμπερίληψη βιβλιοθήκης  LiquidCrystal_I2C

LiquidCrystal_I2C lcd(0x27,16,2);       // Δημιουργία αντικειμένου LiquidCrystal_I2C
RTC_DS1307 rtc;                         // Δημιουργία αντικειμένου τύπου RTC_DS1307

void setup ()
{
        Serial.begin(9600);             // Ενεργοποίηση σειριακής επικοινωνίας
        rtc.begin();                    // Εκκίνηση επικοινωνίας με RTC μέσω I2C
        lcd.init();                     // Ενεργοποίηση οθόνης
        lcd.backlight();                // Ενεργοποίηση οπίσθιου φωτισμού της οθόνης
        lcd.clear();                    // Εκκαθάριση οθόνης
        lcd.print("Real Time Clock");   // Εισαγωγικό μήνυμα
}
```

```
void loop()             // Τυπώνει ημερομηνία και ώρα στην οθόνη υγρών κρυστάλλων
{
        DateTime now = rtc.now();                   // Λήψη τρέχουσας ημερομηνίας και ώρας

        lcd.clear();
        lcd.print("Date: ");
        lcd.print(now.year(), DEC);
        lcd.print('/');
        lcd.print(now.month(), DEC);
        lcd.print('/');
        lcd.print(now.day(), DEC);

        lcd.setCursor(0,1);
        lcd.print("Time: ");
        lcd.print(now.hour(), DEC);
        lcd.print(':');
        lcd.print(now.minute(), DEC);
        lcd.print(':');
        lcd.print(now.second(), DEC);

        delay(1000);                                // Αναμονή 1 sec
}
```

Εννοείται πως πρέπει να έχει προηγηθεί ο συγχρονισμός του RTC με την τρέχουσα ημερομηνία και ώρα, όπως περιγράφηκε στη σχετική παράγραφο (σελ.53) της δεύτερης ενότητας.

Ηλεκτρικές μετρήσεις με τον Arduino και το INA219 - Νόμος του Ohm

Το ολοκληρωμένο INA219 της Texas Instruments μετράει την τάση κατά μήκος μιας αντίστασης 0,1Ω (ανοχής 1%), η οποία παρεμβάλλεται στη διαδρομή του ρεύματος, και από την τάση αυτή προσδιορίζει την ένταση του ρεύματος που διαρρέει την αντίσταση. Στο εμπόριο διατίθενται προσυναρμολογημένες μονάδες που περιέχουν το INA219 μαζί με τα απαραίτητα ηλεκτρονικά εξαρτήματα για τη σύνδεση μέσω του διαύλου I^2C σε μικροϋπολογιστικά συστήματα.



Εικόνα 59: Προσυναρμολογημένη μονάδα INA219

Το ολοκληρωμένο INA219 περιλαμβάνει στην είσοδό του ένα τελεστικό ενισχυτή ακριβείας, που η μέγιστη επιτρεπτή διαφορά δυναμικού στις εισόδους του είναι ±320mV. Με βάση το νόμο του Ohm ($V = I \cdot R$) και με $V = \pm 320mV$ και $R = 100m\Omega$ συμπεραίνουμε πως μπορούν να μετρηθούν ρεύματα στην περιοχή $\pm 3{,}2\ A$.

Το INA219 ενσωματώνει ένα αναλογικο-ψηφιακό μετατροπέα μέγιστης ακρίβειας *12bit* που σημαίνει πως στην περιοχή των $\pm 3{,}2\ A$ έχει διακριτική ικανότητα $0{,}8mA$. Μεταβάλλοντας το κέρδος του τελεστικού ενισχυτή μπορούμε να μεταβάλλουμε την περιοχή μέτρησης αλλά και την αντίστοιχη διακριτική ικανότητα. Για παράδειγμα με το ελάχιστο προκαθορισμένο κέρδος η περιοχή μέτρησης περιορίζεται στα $\pm 400mA$ με διακριτική ικανότητα $0{,}1mA$. Κάθε αναλογικοψηφιακή μετατροπή σε ακρίβεια *12bit* διαρκεί $532\mu s$. Το ολοκληρωμένο μπορεί να προγραμματιστεί για συνεχή τρόπο λειτουργίας, κατά τον οποίο με τη λήξη μιας μετατροπής ξεκινάει αυτόματα μια άλλη μετατροπή, ενώ το αποτέλεσμα διατηρείται στο σχετικό καταχωρητή του ολοκληρωμένου. Ένα επιπλέον σημαντικό χαρακτηριστικό του INA219 είναι η δυνατότητα λήψης 2^N ($N = 1..7$) διαδοχικών μετατροπών και η επιστροφή του μέσου όρου τους, με αποτέλεσμα τη μείωση του ψηφιακού θορύβου, αλλά με σημαντική αύξηση του απαιτούμενου χρόνου για τη μέτρηση [14].

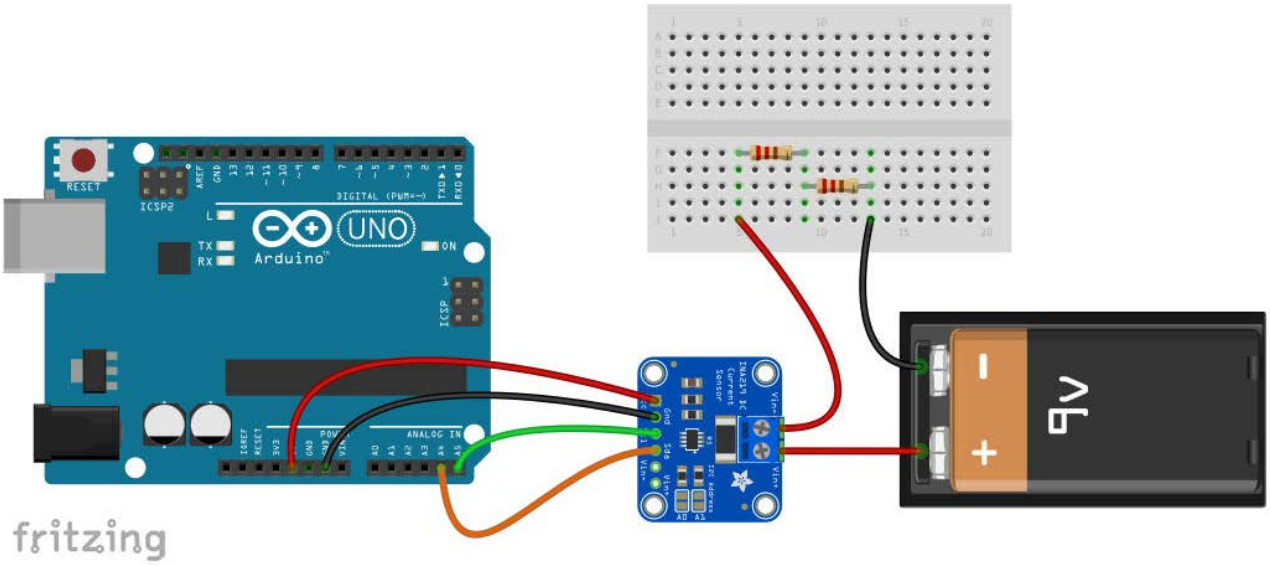
Η επικοινωνία της μονάδας INA219 με τον Arduino υλοποιείται μέσω του δισύρματου διαύλου I^2C, στον οποίο εξ ορισμού αναγνωρίζεται στη δεκαεξαδική διεύθυνση 0x40, κάτι που όμως μπορεί να αλλάξει βραχυκυκλώνοντας τις κατάλληλες επαφές στην πλακέτα της μονάδας INA219.

Υπάρχουν διάφορες βιβλιοθήκες που απλοποιούν τον προγραμματισμό του Arduino για τη μέτρηση της έντασης ρεύματος με τη μονάδα INA219:

- Adafruit INA219 Library
- INA219 Library του John De Cristofaro
- INA219 Library του Korneliusz Jarzebski

Στα επόμενα παραδείγματα χρησιμοποιούμε τη βιβλιοθήκη του Korneliusz Jarzebski, η οποία μπορεί να μεταφορτωθεί από την ιστοσελίδα https://github.com/jarzebski/Arduino-INA219 ως ένα συμπιεσμένο (τύπου zip) αρχείο, και να εγκατασταθεί μέσω της σχετικής εντολής (από το μενού *"Σχέδιο\Συμπερίληψη βιβλιοθήκης\Προσθήκη βιβλιοθήκης ZIP"*) του Arduino IDE.

Στο πρώτο παράδειγμα με χρήση της μονάδας INA219 μετράμε την ένταση του ρεύματος που διαρρέει ένα απλό κύκλωμα, που περιλαμβάνει δύο αντιστάσεις συνδεδεμένες σε σειρά και τροφοδοτούμενες από μπαταρία των 9V.



Εικόνα 60: Σύνδεση μονάδας INA219 στον Arduino για μέτρηση ρεύματος

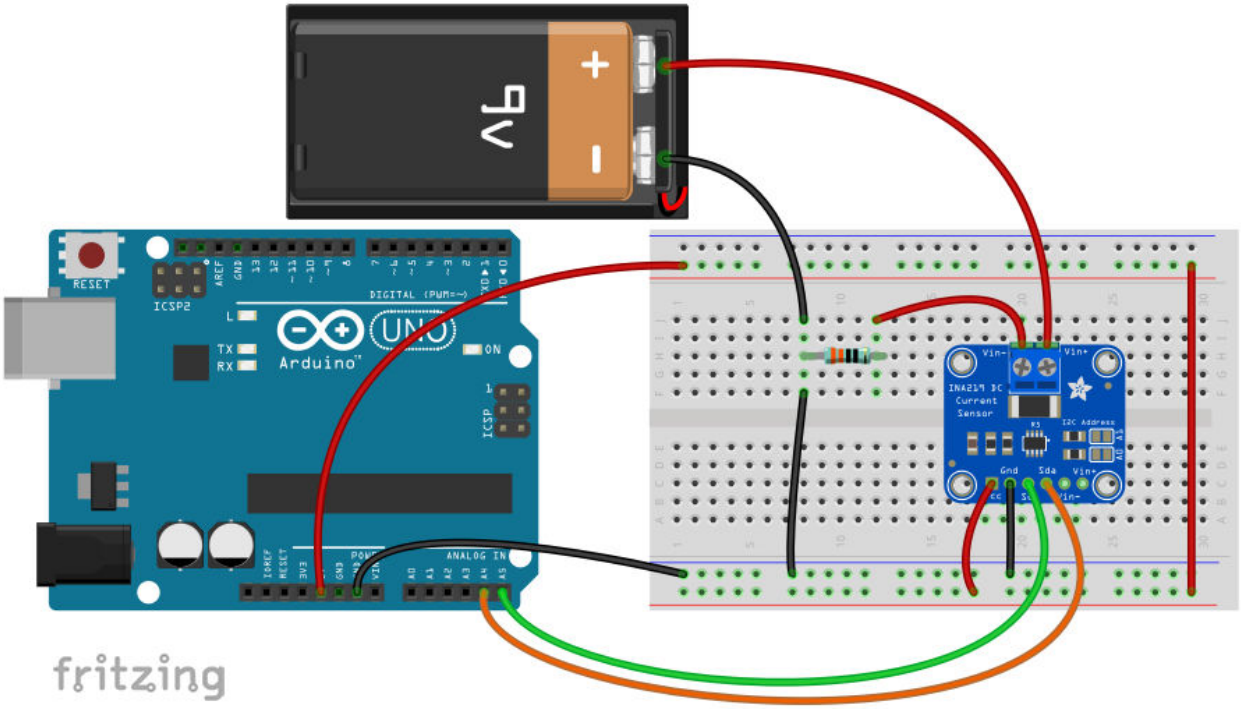
```
/*
        Ex.35 : INA219 Bi-directional Current/Power Monitor
*/

#include <Wire.h>
#include <INA219.h>

INA219 ina219;

void setup()
{
        Serial.begin(9600);
        Serial.println("INA219 Current monitor");
        Serial.println("------------------------------");

        // Default INA219 address is 0x40
        ina219.begin();

        // Configure INA219
        // INA219_RANGE_32V: Max input Voltage = 32V
        // INA219_GAIN_320MV:Max ShuntVoltage = 320mV - Current range  ±3,2 A
        // INA219_BUS_RES_12BIT: ADC precision 12bit
        // INA219_SHUNT_RES_12BIT_1S: 1 measurement =  1 Conversion
        ina219.configure( INA219_RANGE_32V, INA219_GAIN_320MV,
                          INA219_BUS_RES_12BIT, INA219_SHUNT_RES_12BIT_1S);

        // Calibrate INA219: Rshunt = 0.1 ohm, Max excepted current = 2A
        ina219.calibrate(0.1, 2);
}

void loop()
{
        Serial.print("Current (A) : ");
        Serial.println(ina219.readShuntCurrent(), 3);
        delay(1000);
}
```

Μπορούμε να χρησιμοποιήσουμε τη δυνατότητα του INA219 λήψης πολλών διαδοχικών δειγμάτων από αντίστοιχο αριθμό μετατροπών και επιστροφής του μέσου όρου τους, π.χ. με την εντολή:

```
ina219.configure( INA219_RANGE_32V, INA219_GAIN_320MV,
                  INA219_BUS_RES_12BIT, INA219_SHUNT_RES_12BIT_16S);
```

η τιμή της έντασης ρεύματος που λαμβάνουμε είναι ο μέσος όρος 16 διαδοχικών δειγμάτων.

Επιπλέον το INA219 έχει τη δυνατότητα μέτρησης της τάσης μεταξύ του ακροδέκτη του (**V-**) και της γείωσης, καθώς και την τάση στα άκρα της παρεμβαλλόμενης στο κύκλωμα αντίστασης. Το άθροισμα των δύο αυτών τάσεων είναι η συνολική τάση στα άκρα του προς μέτρηση κυκλώματος. Προφανώς γνωρίζοντας την τάση στα άκρα του κυκλώματος και το ρεύμα που το διαρρέει, είμαστε σε θέση να προσδιορίσουμε και την ισχύ που καταναλώνει. Για να εκμεταλλευτούμε τη δυνατότητα αυτή πρέπει η γείωση του Arduino να συνδεθεί στον αρνητικό πόλο της μπαταρίας.



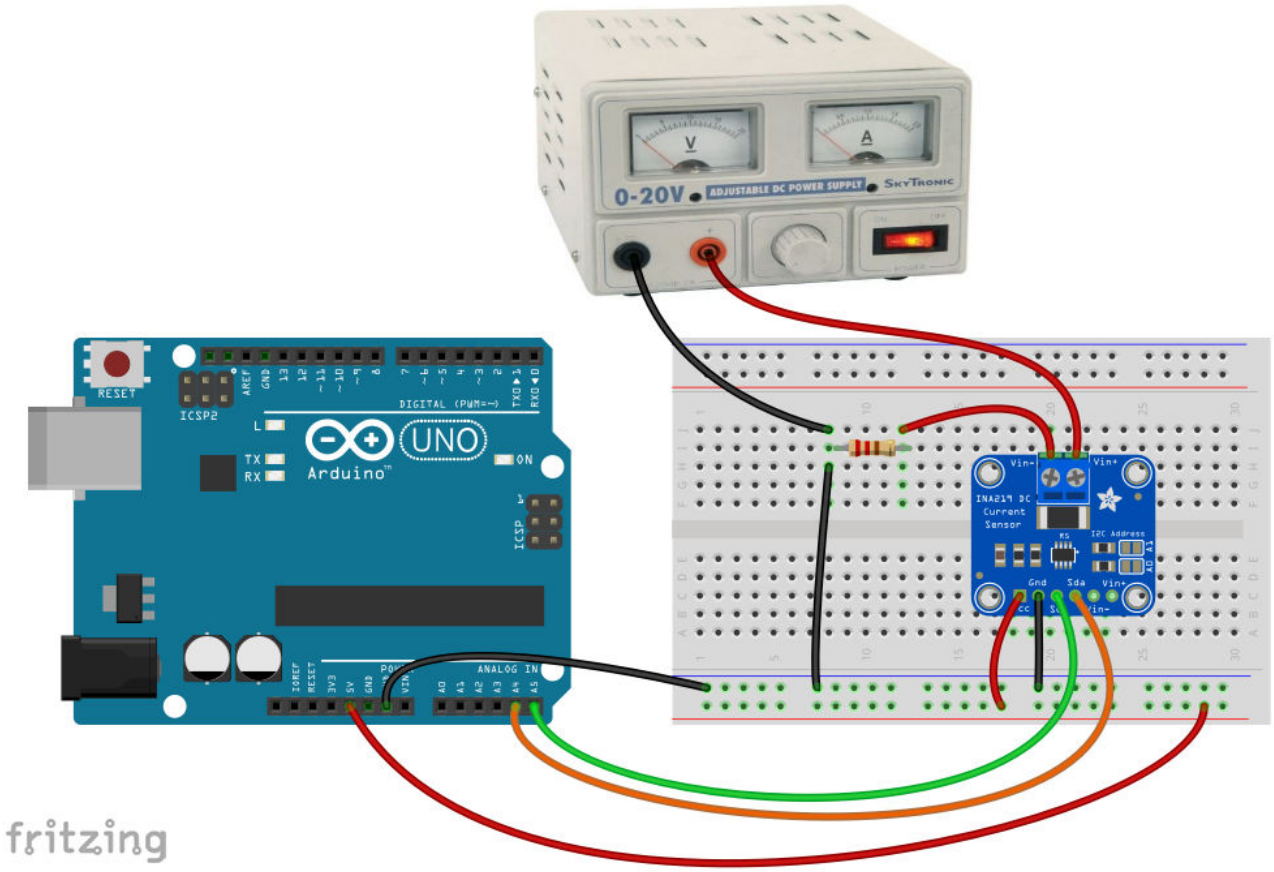
Εικόνα 61: Τροποποιημένο κύκλωμα για μέτρηση και της τάσης διαύλου

Το αντίστοιχο sketch του Arduino παίρνει τη μορφή:

```
/*
        Ex.36 : INA219 Bi-directional Current/Power Monitor
*/

#include <Wire.h>
#include <INA219.h>

INA219 ina219;

void setup()
{
        Serial.begin(9600);
        Serial.println("INA219 Current/Power monitor");
        Serial.println("----------------------------");

        // Default INA219 address is 0x40
        ina219.begin();
```

```
        // Configure INA219
        ina219.configure( INA219_RANGE_32V, INA219_GAIN_320MV,
                          INA219_BUS_RES_12BIT, INA219_SHUNT_RES_12BIT_16S);

        // Calibrate INA219 - Rshunt = 0.1 ohm, Max excepted current = 2A
        ina219.calibrate(0.1, 2);
}

void loop()
{
        Serial.print("Current (A) : ");
        Serial.print(ina219.readShuntCurrent(), 3);
        Serial.print(" Bus Voltage (V) : ");
        Serial.print(ina219.readBusVoltage(), 4);
        Serial.print(" Total Voltage (V) : ");
        Serial.print(ina219.readBusVoltage()+ina219.readShuntVoltage(), 4);
        Serial.print(" Bus Power (W): ");
        Serial.println(ina219.readBusPower(), 4);
        delay(1000);
}
```

Με τη βοήθεια του αισθητήρα INA219, εύκολα μπορούμε να σχεδιάσουμε ένα πείραμα για την επιβεβαίωση του νόμου του Ohm. Το σχετικό κύκλωμα φαίνεται στην επόμενη εικόνα:



Εικόνα 62: Συνδεσμολογία για το πείραμα του νόμου του Ohm

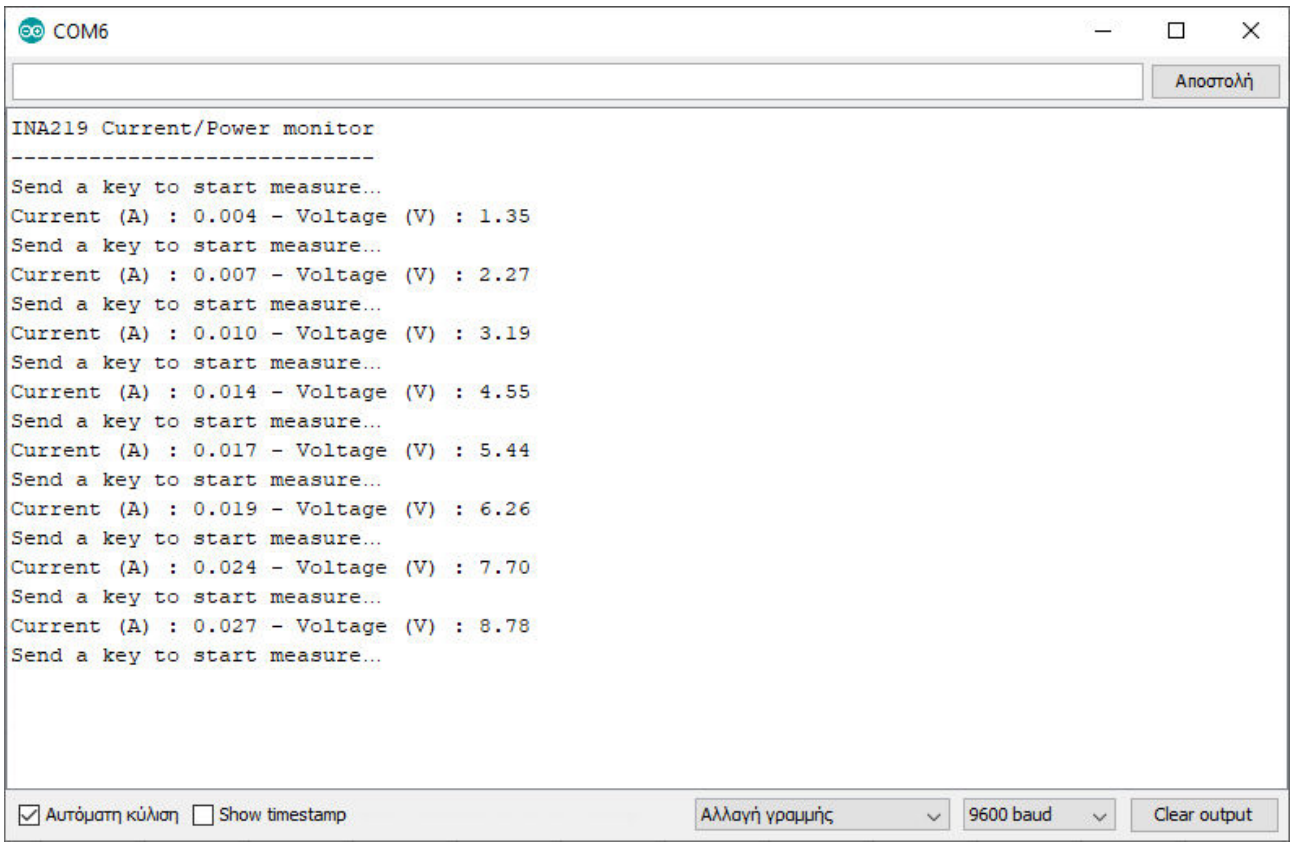
Χρησιμοποιήσαμε μια αντίσταση ονομαστικής τιμής 330Ω και με τη βοήθεια του ρυθμιζόμενου σταθεροποιημένου τροφοδοτικού περιορίσαμε την τάση στα άκρα της μέχρι τα 9V, ώστε η μέγιστη

τιμή της έντασης του ρεύματος στο κύκλωμα να μη ξεπερνάει τα 30mA περίπου.

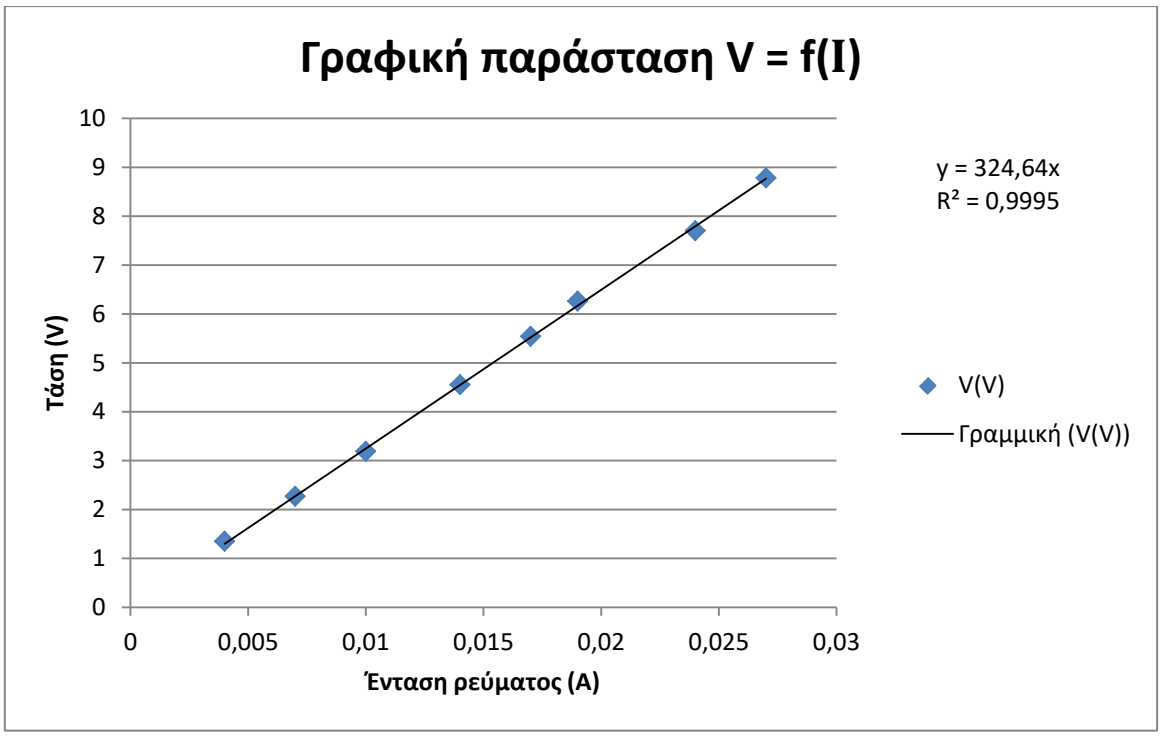
Το σχετικό sketch για τον Arduino μπορεί να έχει τη μορφή:

```
/*
        Ex.37 : Επιβεβαίωση του Νόμου του Ohm με το INA219
*/

#include <Wire.h>
#include <INA219.h>

INA219 ina219;

void setup()
{
        Serial.begin(9600);
        Serial.println("INA219 Current/Power monitor");
        Serial.println("-----------------------------");

        // Default INA219 address is 0x40
        ina219.begin();

        // Configure INA219
        ina219.configure( INA219_RANGE_32V,
                          INA219_GAIN_320MV,
                          INA219_BUS_RES_12BIT,
                          INA219_SHUNT_RES_12BIT_16S);

        // Calibrate INA219 - Rshunt = 0.1 ohm, Max excepted current = 2A
        ina219.calibrate(0.1, 2);
}

void loop()
{
        Serial.println("Send a key to start measure…");
        while (Serial.available() == 0);

        Serial.print("Current (A) : ");
        Serial.print(ina219.readShuntCurrent(), 3);
        Serial.print(" - Voltage (V) : ");
        Serial.println(ina219.readBusVoltage(), 2);

        // Clear Serial Buffer
        while (Serial.available() != 0)
        {
                int c = Serial.read();
        }

}
```

Το λογισμικό με τη συνάρτηση *readShuntCurrent()* επιστρέφει την τιμή *I* του ρεύματος που διαρρέει την αντίσταση, και με τη συνάρτηση *readBusVoltage()* την τάση *V* στα άκρα της (Εικόνα 63) . Η επιβεβαίωση του νόμου του Ohm μπορεί να γίνει π.χ. μέσω της γραφικής παράστασης *V = f(I)*, που μπορούμε να σχεδιάσουμε με τη βοήθεια του λογισμικού Excel (Εικόνα 64).



Εικόνα 63 : Πειραματικά δεδομένα για την επιβεβαίωση του Νόμου του Ohm



Εικόνα 64 : Η γραφική παράσταση V = f (I)

Από την κλίση της γραφικής παράστασης *V = f(I)*, μπορούμε να υπολογίσουμε την πειραματική τιμή της αντίστασης που χρησιμοποιήσαμε: *R = 325Ω*.

Παράρτημα Α: Τύποι δεδομένων

Η γλώσσα προγραμματισμού του Arduino υποστηρίζει τους εξής τύπους για την αποθήκευση δεδομένων [4]:

int	Στον Arduino Uno (και σε όλα τα μοντέλα που βασίζονται σε μικροελεγκτές ATmega) χρησιμοποιείται για την αποθήκευση ακέραιων αριθμών 16 bits (2 bytes) από -32.768 μέχρι 32.767. Στον Arduino Due και στα μοντέλα που βασίζονται σε επεξεργαστές SAMD χρησιμοποιείται για την αποθήκευση ακέραιων αριθμών 32 bits (4 bytes) από -2.147.483.648 μέχρι 2.147.483.647.
short	Για την αποθήκευση ακέραιων αριθμών 16 bits (2 bytes) από -32.768 μέχρι 32.767.
long	Για ακέραιες μεταβλητές των 32 bits (ή 4 bytes) από -2.147.483.648 μέχρι 2.147.483.647.
word	Για ακέραιους αριθμούς 16 bits χωρίς πρόσημο (από 0 μέχρι 65.535).
float	Για δεκαδικούς αριθμούς από 3.4028235E+38 μέχρι -3.4028235E+38. Αποθηκεύονται σαν 32 bits (4 bytes) πληροφορίας.
double	Στον Arduino Uno (και σε όλα τα μοντέλα που βασίζονται σε μικροελεγκτές ATmega) δεν έχει καμία διαφορά με τον τύπο float. Στον Arduino Due και στα μοντέλα που βασίζονται σε επεξεργαστές SAMD χρησιμοποιείται για την αποθήκευση δεκαδικών αριθμών ακρίβειας 64 bits.
byte	Για την αποθήκευση ακέραιων αριθμών ακρίβειας 8 bits από 0 μέχρι 255.
char	Καταλαμβάνει 1 byte μνήμης και αποθηκεύει την τιμή ενός χαρακτήρα. Τα γράμματα των χαρακτήρων γράφονται σε μοναδικά εισαγωγικά, π.χ: 'A'.
bool	Καταλαμβάνει 1 byte μνήμης και αποθηκεύει μόνο δύο τιμές: **true** ή **false**.
array	Συλλογή μεταβλητών που είναι προσβάσιμες με έναν αριθμό ευρετηρίου (πίνακας).
string	Συλλογή αλφαριθμητικών χαρακτήρων (συμβολοσειρά). Μπορεί να δηλωθεί με τον τύπο δεδομένων **String**, ή ως ένας πίνακας χαρακτήρων τερματισμένος με το χαρακτήρα null (ASCII code 0).
unsigned int	Στον Arduino Uno (και σε όλα τα μοντέλα που βασίζονται σε μικροελεγκτές ATmega) χρησιμοποιείται για την αποθήκευση ακέραιων αριθμών 16 bits (2 bytes) χωρίς πρόσημο από 0 μέχρι 65.535. Στον Arduino Due και στα μοντέλα που βασίζονται σε επεξεργαστές SAMD χρησιμοποιείται για την αποθήκευση ακέραιων αριθμών 32 bits (4 bytes) από 0 μέχρι 4.294.967.295.
unsigned long	Χρησιμοποιείται για την αποθήκευση ακέραιων αριθμών 32 bits (4 bytes) από 0 μέχρι 4.294.967.295.
unsigned char	Ίδιος με τον τύπο byte.

Παράρτημα Β: Τελεστές

Ένας τελεστής είναι ένα σύμβολο (ή συνδυασμός συμβόλων) που πληροφορεί τον μεταγλωττιστή να πραγματοποιήσει συγκεκριμένη μαθηματική ή λογική πράξη. Οι τελεστές της γλώσσας C++ μπορεί να είναι:

- Αριθμητικοί
- Λογικοί
- Σύγκρισης
- Επιπέδου bit
- Αντικατάστασης

Στους πίνακες που ακολουθούν αναφέρονται οι τελεστές που χρησιμοποιούνται στον προγραμματισμό του Arduino καθώς και η σύντομη περιγραφή τους [15, 16, 4].

Αριθμητικοί Τελεστές

Όνομα	Σύμβολο	Παράδειγμα	Περιγραφή
Ανάθεση	=	A = 5	Αποθηκεύει την τιμή 5 στη μεταβλητή A
Πρόσθεση	+	Y = A + B	Πρόσθεση των αριθμών A και B
Αφαίρεση	-	Y = A – B	Αφαίρεση του B από τον A
Πολλαπλασιασμός	*	Y = A * B	Πολλαπλασιασμός των αριθμών A, B
Διαίρεση	/	Y = B / A	Διαίρεση B διά A. Όταν και οι δύο αριθμοί είναι ακέραιοι, το αποτέλεσμα είναι επίσης ακέραιος, ενώ αν ένας από τους δύο είναι κινητής υποδιαστολής, εφαρμόζεται διαίρεση πραγματικών αριθμών.
Υπόλοιπο	%	Y = B % A	Υπόλοιπο της διαίρεσης B / A. Εφαρμόζεται μόνο σε ακέραιους αριθμούς.

Λογικοί τελεστές

Όνομα	Σύμβολο	Παράδειγμα	Περιγραφή
AND	&&	(A && B)	Λογική σύζευξη (AND) Το αποτέλεσμα είναι true, **μόνο** αν και οι δύο τελεστέοι (A, B) είναι true.
OR	||	(A || B)	Λογικό διάζευξη (OR) Το αποτέλεσμα είναι true, αν **ένας τουλάχιστον** από τους τελεστέους είναι true.
NOT	!	!(A)	Λογική άρνηση (NOT) Αν ο τελεστέος είναι true η λογική του άρνηση είναι false και το αντίστροφο.

Τελεστές σύγκρισης

Όνομα	Σύμβολο	Παράδειγμα	Περιγραφή
Ίσο με	==	(A == B)	True αν η μεταβλητή A είναι ίση με τη B
Όχι ίσο με	!=	(A != B)	True αν η μεταβλητή A δεν είναι ίση με τη B
Μικρότερο από	<	(A < B)	True αν η μεταβλητή A είναι μικρότερη από τη B
Μεγαλύτερο από	>	(A > B)	True αν η μεταβλητή A είναι μεγαλύτερη από τη B
Μικρότερο από ή ίσο με	<=	(A <= B)	True αν η μεταβλητή A είναι μικρότερη από ή ίση με τη B
Μεγαλύτερο από ή ίσο με	>=	(A >= B)	True αν η μεταβλητή A είναι μεγαλύτερη από ή ίση με τη B

Τελεστές επιπέδου bit

Όνομα	Σύμβολο	Παράδειγμα	Περιγραφή
and	&	(A & B)	Στο αποτέλεσμα, ένα bit είναι 1, μόνο αν τα αντίστοιχα bits στους τελεστέους είναι και τα δύο 1.
or	|	(A | B)	Στο αποτέλεσμα, ένα bit είναι 1, αν τουλάχιστον ένα από τα αντίστοιχα bits στους τελεστέους είναι 1.
xor	^	(A ^ B)	Στο αποτέλεσμα, ένα bit είναι 1, αν ακριβώς ένα (όχι και τα δύο) από τα αντίστοιχα bits στους τελεστέους είναι 1.
not	~	(~A	Αντιστρέφει τα bits του τελεστέου, δηλαδή αν είναι 0 τα κάνει 1 και αν είναι 1 τα κάνει 0.
shift left	<<	A << 2	Μετατοπίζει τα bits του αριστερού τελεστέου προς τα αριστερά. Ο αριθμός των θέσεων μετατόπισης καθορίζεται από το δεξιό τελεστέο (2 στην περίπτωση του παραδείγματος). Οι κενές θέσεις που προκύπτουν στα δεξιά γεμίζουν με μηδενικά.
shift right	>>	A >> 2	Μετατοπίζει τα bits του αριστερού τελεστέου προς τα δεξιά. Ο αριθμός των θέσεων μετατόπισης καθορίζεται από το δεξιό τελεστέο (2 στην περίπτωση του παραδείγματος). Οι κενές θέσεις που προκύπτουν στα αριστερά γεμίζουν με μηδενικά.

Τελεστές αντικατάστασης

Όνομα	Σύμβολο	Παράδειγμα	Περιγραφή
Αύξηση	++	A++	Ισοδύναμο με A = A + 1
Μείωση	--	A--	Ισοδύναμο με A = A - 1
Πρόσθεση	+=	B += A	Ισοδύναμο με B = B + A
Αφαίρεση	-=	B -= A	Ισοδύναμο με B = B - A
Πολλαπλασιασμός	*=	B *= A	Ισοδύναμο με B = B * A
Διαίρεση	/=	B /= A	Ισοδύναμο με B = B / A
Υπόλοιπο	%=	B %= A	Ισοδύναμο με B = B % A
OR επιπέδου bit	|=	A |= 2	Ισοδύναμο με A = A | 2
AND επιπέδου bit	&=	A &= 2	Ισοδύναμο με A = A & 2

Βιβλιογραφία

[1] H. Barragán, «The Untold History of Arduino,» [Ηλεκτρονικό]. Available: https://arduinohistory.github.io. [Πρόσβαση Δεκέμβριος 2018].

[2] «Arduino,» [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Arduino. [Πρόσβαση Δεκέμβριος 2018].

[3] «ATmega328P Datasheet,» [Ηλεκτρονικό]. Available: https://engineering.purdue.edu/ME588/SpecSheets/atmega328p.pdf. [Πρόσβαση Δεκέμβριος 2018].

[4] «Arduino Reference,» [Ηλεκτρονικό]. Available: https://www.arduino.cc/reference/en/.

[5] «What does this while(true) loop do in Arduino or any other language?,» [Ηλεκτρονικό]. Available: https://stackoverflow.com/questions/48524857/what-does-this-whiletrue-loop-do-in-arduino-or-any-other-language. [Πρόσβαση Δεκέμβριος 2018].

[6] «Photoresistor,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Photoresistor. [Πρόσβαση Δεκέμβριος 2018].

[7] «I²C,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/I²C. [Πρόσβαση Δεκέμβριος 2018].

[8] «TinyRTC I2C Module,» Mantech Electronics, [Ηλεκτρονικό]. Available: http://www.mantech.co.za/Datasheets/Products/MD0095-180521A.pdf. [Πρόσβαση Δεκεμβριος 2018].

[9] «1-Wire,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/1-Wire. [Πρόσβαση Δεκέμβριος 2018].

[10] «Serial Peripheral Interfac,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface. [Πρόσβαση Δεκέμβριος 2018].

[11] Honeywell, «SS39ET/SS49E/SS59ET Series Linear Hall-effect Sensor ICs,» [Ηλεκτρονικό]. Available: https://sensing.honeywell.com/index.php?ci_id=50359. [Πρόσβαση Φεβρουάριος 2019].

[12] «Speed of Sound,» Wikipedia, [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Speed_of_sound. [Πρόσβαση Δεκέμβριος 2018].

[13] «LM35 Precision Centigrade Temperature Sensors,» Texas Instruments, [Ηλεκτρονικό]. Available: http://www.ti.com/lit/ds/symlink/lm35.pdf. [Πρόσβαση Δεκέμβριος 2018].

[14] «Adafruit INA219 Current Sensor Breakout,» [Ηλεκτρονικό]. Available: https://cdn-learn.adafruit.com/downloads/pdf/adafruit-ina219-current-sensor-breakout.pdf. [Πρόσβαση Δεκέμβριος 2018].

[15] «Arduino Operators,» [Ηλεκτρονικό]. Available: https://www.tutorialspoint.com/arduino/arduino_operators.htm. [Πρόσβαση Δεκέμβριος 2018].

[16] «Εισαγωγή στη γλώσσα C & Εφαρμογές,» [Ηλεκτρονικό]. Available: http://www.dga.gr/guest/pluginfile.php/5773/mod_resource/content/4/03%20-%20Operations%2C%20Operators%20and%20Constants.pdf. [Πρόσβαση Δεκέμβριος 2018].

ISBN : 978-618-83502-3-6