Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Τα πρωτόκολλα επικοινωνίας I2C και SPI και η υλοποίησή τους σε σύστημα τηλεχειρισμού, με τη χρήση της ηλεκτρονικής πλατφόρμας προτυποποίησης Arduino.** **The I2C and SPI communication protocols and their implementation in a remote control system, based on the Arduino electronics prototyping platform.**
Ονοματεπώνυμο Φοιτητή	**Θεμιστοκλής Δημητρακόπουλος**
Πατρώνυμο	**Κωνσταντίνος**
Αριθμός Μητρώου	**ΜΠΠΛ/ 12011**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**

Ημερομηνία Παράδοσης **Μάρτιος 2016**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα	Όνομα Επώνυμο Βαθμίδα

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον υπεύθυνο καθηγητή κ. Χρήστο Δουληγέρη, διότι με την παρούσα εργασία μού δόθηκε η ευκαιρία να συνδυάσω δύο επιστήμες μαζί, αυτή των ηλεκτρονικών και της πληροφορικής. Η επικοινωνία του προσωπικού υπολογιστή με κάποια ηλεκτρονική συσκευή "ζωντανεύει" τη συσκευή, ενώ παράλληλα επεκτείνει τις δυνατότητες του υπολογιστή και πέραν από αυτές της οθόνης. Θα ήθελα επιπλέον να ευχαριστήσω τον αγαπητό μου φίλο Ευάγγελο Ορφανίδη που είχε την ιδέα της συγκεκριμένης εφαρμογής τηλεχειρισμού, τον αγαπητό μου φίλο Νικόλαο Κινόπουλο για τις ιδιαίτερες γνώσεις του στην ανάπτυξη λογισμικού, καθώς και την αγαπητή μου σύζυγο Χριστινούλα, την ευρύτερη οικογένειά μου και τους φίλους μου για τη συμπαράστασή τους και την ηθική τους υποστήριξη.

Περίληψη

Σκοπός της παρούσας μεταπτυχιακής διατριβής είναι η παρουσίαση και η υλοποίηση των δύο πρωτοκόλλων σειριακής επικοινωνίας I²C και SPI, τα οποία χρησιμοποιούνται κυρίως σε ενσωματωμένα συστήματα για τη διακίνηση πληροφορίας ανάμεσα σε κάποιον μικροεπεξεργαστή και τις περιφερειακές του συσκευές.

Για να διαπιστωθούν τα οφέλη των δύο αυτών πρωτοκόλλων, καθώς και οι ιδιαιτερότητές τους κατά την υλοποίησή τους σε πραγματικά συστήματα, χρησιμοποιήθηκε η ηλεκτρονική πλατφόρμα προτυποποίησης Arduino, με σχεδιασμό ηλεκτρονικού κυκλώματος και ανάπτυξη λογισμικού. Η εφαρμογή είχε ως σκοπό τον τηλεχειρισμό 128 ακροδεκτών ψηφιακής εισόδου από ηλεκτρονικό πίνακα ελέγχου και οθόνη, ή από προσωπικό υπολογιστή μέσω δικτύου Ethernet. Κατασκευάστηκε ένα πρόγραμμα (sketch) για τον προγραμματισμό του μικροεπεξεργαστή του Arduino, έτσι ώστε να ελέγχονται οι ακροδέκτες από τον πίνακα χειρισμού, ενώ στα πλαίσια της παρουσίασης της υλοποίησης δημιουργήθηκε πρόγραμμα για το χειρισμό των ακροδεκτών από τον προσωπικό υπολογιστή.

Οι 128 ακροδέκτες ψηφιακής εισόδου αποκτήθηκαν με τη χρήση οκτώ ολοκληρωμένων κυκλωμάτων MCP23017. Για την επικοινωνία των MCP23017 με τον μικροεπεξεργαστή του Arduino χρησιμοποιήθηκε ο δίαυλος I²C.

Η σύνδεση του Arduino με το δίκτυο Ethernet πραγματοποιήθηκε με τη χρήση της πλακέτας επέκτασης Arduino Ethernet Shield. Για την επικοινωνία της πλακέτας επέκτασης με τον μικροεπεξεργαστή του Arduino χρησιμοποιήθηκε ο δίαυλος SPI.

Η υλοποίηση των πρωτοκόλλων I²C και SPI στην παρούσα εφαρμογή τηλεχειρισμού, προσέφερε αξιόπιστη επικοινωνία του μικροεπεξεργαστή του Arduino με τις συγκεκριμένες περιφερειακές του συσκευές. Επιπλέον, αναδείχθηκαν τα ιδιαίτερα πλεονεκτήματα του κάθε πρωτοκόλλου, η ευκολία στη συνδεσμολογία του διαύλου I²C και η ταχύτητα στη διακίνηση της πληροφορίας μέσω του SPI.

Abstract

The goal of this master thesis is to introduce and implement the I²C and SPI serial communication protocols, which are utilized mainly in embedded systems for data transfer between a microprocessor and its peripheral devices.

In order to investigate the benefits of these two protocols as well their special requirements, during their implementation in real systems, an Arduino prototyping platform including electronics circuit designing and software development was deployed. This project's purpose was to control remotely 128 digital input pins from the control panel, or through an Ethernet network by a personal computer. A program (sketch) was developed for the Arduino's microprocessor programming, so that the input pins be able to be controlled from the control panel, while another program was created for the project's presentation needs, which enables the user to control the input pins from a personal computer.

The 128 digital input pins were acquired by using eight integrated circuits MCP23017. In order these IC s to communicate with Arduino's microprocessor, the I²C bus was utilized.

The connection of Arduino with Ethernet network realized by using the Arduino Ethernet Shield extension board. In order this extension board to communicate with Arduino's microprocessor, the SPI bus was utilized.

The implementation of I²C and SPI protocols in this remote control application, contributed to reliable communications between Arduino's microprocessor and these peripheral devices. Moreover, the special advantages of each protocol were brought out, the simplicity when connecting on I²C bus and the high data bit rates through SPI.

Πίνακας περιεχομένων

Λίστα Εικόνων

Λίστα Πινάκων

Εισαγωγή

Στην καθημερινότητά μας χρησιμοποιούμε μια πληθώρα ηλεκτρικών συσκευών. Αυτές μπορεί να είναι οι γνωστές μας οικιακές συσκευές, οι προσωπικοί υπολογιστές, μικροσυσκευές, συσκευές για εξειδικευμένες εργασίες κ.α. Το κοινό τους χαρακτηριστικό είναι ότι ενσωματώνουν κάποιον μικροεπεξεργαστή που επικοινωνεί με μία ή με περισσότερες περιφερειακές του συσκευές, έναν αισθητήρα εικόνας, μία κάρτα δικτύου, μνήμες.

Στην παρούσα εργασία εξετάζονται δύο τρόποι σειριακής επικοινωνίας του μικροεπεξεργαστή με τις περιφερειακές του συσκευές, ο δίαυλος I^2C και ο δίαυλος SPI. Για να διαπιστωθούν τα πλεονεκτήματα αλλά και οι περιορισμοί που προκύπτουν κατά τη χρήση των συγκεκριμένων διαύλων, αναπτύχθηκε εφαρμογή τηλεχειρισμού βασισμένη στην ηλεκτρονική πλατφόρμα προτυποποίησης Arduino.

Στα πρώτα δύο κεφάλαια παρουσιάζονται τα δύο πρωτόκολλα επικοινωνίας, η λειτουργία τους, οι βασικές και συνήθεις δυνατότητές τους και τα ιδιαίτερα χαρακτηριστικά τους. Στη συνέχεια, στο τρίτο κεφάλαιο γίνεται μεταξύ τους σύγκριση, έτσι ώστε να προσδιοριστούν τα κριτήρια με τα οποία θα πρέπει να γίνεται η επιλογή τους κατά το σχεδιασμό ηλεκτρονικών εφαρμογών.

Στο τέταρτο κεφάλαιο παρουσιάζεται συνοπτικά η πλατφόρμα προτυποποίησης Arduino. Γίνεται αναφορά στις ανάγκες που δημιούργησαν το Arduino, τα χαρακτηριστικά των πλακετών Arduino που βρίσκονται σήμερα διαθέσιμα στην αγορά, καθώς και στο περιβάλλον του λογισμικού ανάπτυξης Arduino που χρησιμοποιείται για τον προγραμματισμό του μικροεπεξεργαστή των πλακετών.

Στο πέμπτο κεφάλαιο περιγράφεται το υλικό που χρησιμοποιήθηκε στην παρούσα εφαρμογή τηλεχειρισμού. Γίνεται περιγραφή της πλακέτας Arduino Duemilanove, της πλακέτας επέκτασης Arduino Ethernet Shield, του ολοκληρωμένου κυκλώματος επέκτασης ακροδεκτών εισόδου/εξόδου MCP23017, της οθόνης υγρών κρυστάλλων 20 στηλών επί 4 γραμμών, καθώς και του απομονωτή/ενισχυτή P82B96 σημάτων I^2C που μπορεί να ενταχθεί προαιρετικά στην εφαρμογή μας με σκοπό την επέκταση των καλωδίων του διαύλου I^2C. Επίσης παρουσιάζεται σχηματικά η συνδεσμολογία του υλικού της εφαρμογής.

Στο έκτο κεφάλαιο γίνεται επίδειξη της λειτουργίας της εφαρμογής. Αρχικά παρουσιάζεται περιγραφικά και με στιγμιότυπα η λειτουργία του προγράμματος του Arduino, ενώ στη συνέχεια γίνεται ανάλογη παρουσίαση του προγράμματος C# που εκτελείται από τον προσωπικό υπολογιστή. Για να διευκολυνθεί η κατανόηση του προγράμματος (sketch) του Arduino, καθώς και του προγράμματος σε C#, επεξηγούνται κάποια σημεία τους που κρίνονται σημαντικά και ίσως δυσνόητα.

Στο τέλος ακολουθούν τα συμπεράσματα που προέκυψαν μετά την ολοκλήρωση της εργασίας, όπως και προτάσεις για την κατάλληλη επιλογή και υλοποίηση των συγκεκριμένων πρωτοκόλλων.

ΚΕΦΑΛΑΙΟ 1 - Το πρωτόκολλο επικοινωνίας I2C

1.1 Εισαγωγή

Στις αρχές της δεκαετίας του 80' η Phillips Semiconductors (σήμερα NXP Semiconductors) παρουσίασε έναν πρωτοποριακό δίαυλο σειριακής επικοινωνίας, τον I²C-bus, inter-integrated circuit bus. Παρότι δεν προσέφερε γρήγορους ρυθμούς μετάδοσης δεδομένων, η ευελιξία και η απλότητα στη χρήση του τον κατέστησε για πάνω από τριάντα χρόνια βασική επιλογή των σχεδιαστών ηλεκτρονικών συσκευών [1].

Ο δίαυλος I²C υλοποιείται σήμερα από περισσότερα από 1000 διαφορετικά ολοκληρωμένα κυκλώματα ICs και από περισσότερες από 50 εταιρίες. Επίσης ο I²C χρησιμοποιείται σε ποικίλες αρχιτεκτονικές ελέγχου όπως είναι τα System Management Bus (SMBus), Power Management Bus (PMBus), Intelligent Platform Management Interface (IPMI), Display Data Channel (DDC) και Advanced Telecom Computing Architecture (ATCA) [2].

Ο σκοπός της Phillips Semiconductors που αρχικά ικανοποιούσε ο I²C ήταν να περιορίσει το κόστος παραγωγής των ηλεκτρονικών προϊόντων ευρείας κατανάλωσης (-CE- Consumer Electronics). Έτσι, οι πρώτες εφαρμογές του ήταν ο έλεγχος του φωτισμού και του ήχου σε τηλεοράσεις και ραδιόφωνα. Σήμερα, ο I²C συναντάται σχεδόν σε όλα τα προϊόντα CE, όπως σε κινητά και συμβατικά τηλέφωνα, υπολογιστές, ATMs, αλλά και σε μια πληθώρα άλλα προϊόντα και συστήματα, όπως σε βιομηχανικά, στρατιωτικά και σε προϊόντα και συστήματα για την ψυχαγωγία και την υγεία.

Σχεδόν όλοι οι βασικοί κατασκευαστές ημιαγωγών και ηλεκτρονικών συστημάτων υποστηρίζουν το δίαυλο I²C. Τέτοιες συσκευές είναι οι συσκευές εισόδου και εξόδου, οι μνήμες, οι αισθητήρες, τα ρολόγια πραγματικού χρόνου, οι οθόνες κ.α. [1]

1.2 Τι είναι ο δίαυλος I²C

Κατά την κατασκευή των καταναλωτικών ηλεκτρονικών συσκευών CE εφαρμόζονται τρόποι σχεδίασης οι οποίοι είναι κοινοί στα περισσότερα προϊόντα. Έτσι στις ηλεκτρονικές συσκευές συνήθως συναντάμε:

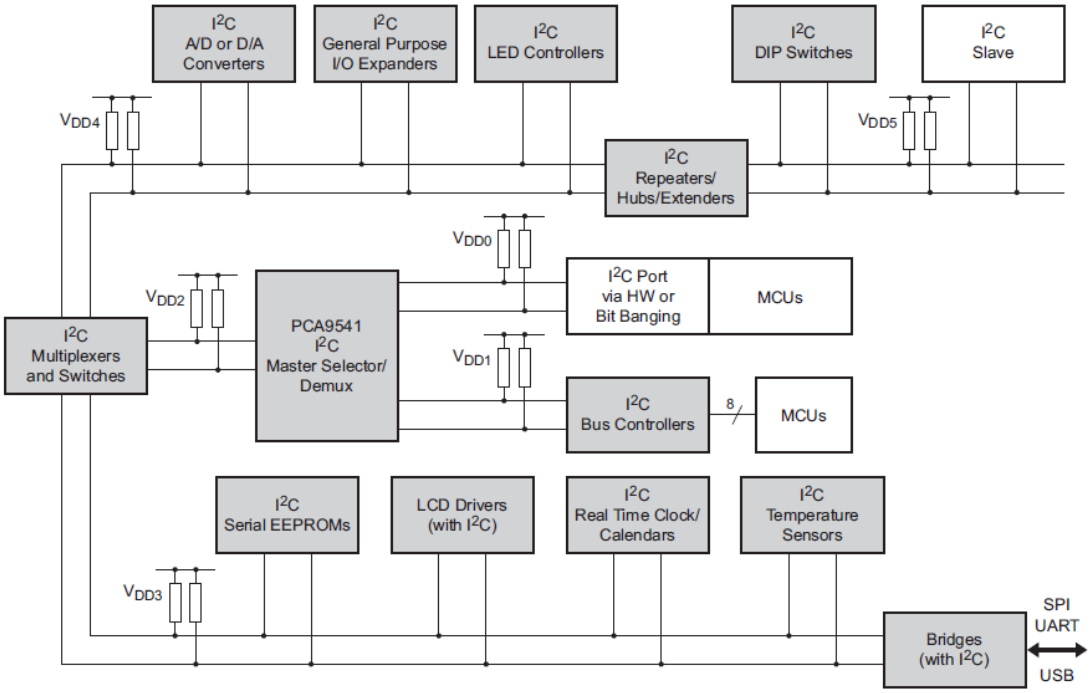
- Κάποιον μικροεπεξεργαστή για τον ευφυή έλεγχο της συσκευής
- Κυκλώματα γενικής χρήσης, όπως οδηγούς οθονών LCD και LED, ακροδέκτες εισόδου-εξόδου, μνήμες RAM και EEPROM, ρολόγια πραγματικού χρόνου και μετατροπείς από ψηφιακά σε αναλογικά (digital to analog D/A) και αντίστροφους A/D
- Κυκλώματα ανάλογα με την εφαρμογή, όπως κυκλώματα επεξεργασίας σήματος και ψηφιακού συντονισμού και διάφορους αισθητήρες όπως αισθητήρες θερμοκρασίας [2]

Παρουσιάζεται επομένως κατά τη σχεδίαση των ηλεκτρονικών συσκευών μία κοινή ανάγκη: να επικοινωνήσουν μεταξύ τους τα διάφορα ψηφιακά εξαρτήματα, τα οποία απαιτούνται για τις λειτουργίες μιας ηλεκτρονικής συσκευής, με την χρησιμοποίηση όσο το δυνατόν λιγότερων επιπλέον εξαρτημάτων και γενικότερα με τον πιο αποδοτικό τρόπο [1].

Τη λύση σε αυτό το πρόβλημα την έδωσε ο δίαυλος I²C, που λόγω των χαρακτηριστικών του άντεξε στο χρόνο και εξακολουθεί να βρίσκει εφαρμογή στα σύγχρονα ηλεκτρονικά συστήματα. Κάποια από τα βασικά χρήσιμα χαρακτηριστικά του είναι τα ακόλουθα:

- Απαιτούνται μόνο δύο γραμμές επικοινωνίας: μία γραμμή σειριακών δεδομένων (SDA) και μία γραμμή σειριακού ρολογιού (SCL).

- Κάθε εξάρτημα που συνδέεται πάνω στο δίαυλο I²C παίρνει μοναδική διεύθυνση και τα εξαρτήματα μεταξύ τους σχετίζονται με απλές σχέσεις master-slave.
- Ο δίαυλος διαθέτει αλγόριθμο ανίχνευσης σύγκρουσης και διαιτησίας για την αποφυγή της αλλοίωσης των δεδομένων όταν δύο ή περισσότεροι masters ξεκινήσουν ταυτόχρονα τη μετάδοση δεδομένων.
- Φίλτρα που βρίσκονται εσωτερικά των εξαρτημάτων απορρίπτουν τυχόν αιχμές στη γραμμή δεδομένων του διαύλου για την διαφύλαξη της ακεραιότητας των δεδομένων.
- Ο δίαυλος υποστηρίζει 8-bit σειριακή αμφίδρομη μεταφορά δεδομένων με ρυθμούς έως 100 kbit/s στο Standard mode, μέχρι 400 kbit/s στο Fast mode, μέχρι 1 Mbit/s στο Fast mode Plus και μέχρι 3,4 Mbit/s στο High-speed mode. Στο Ultra-Fast mode η μετάδοση δεδομένων είναι μονόδρομη και φτάνει μέχρι τα 5 Mbit/s [2].



Εικόνα 1. Παράδειγμα εφαρμογών του διαύλου I²C [2]

1.3 Περιγραφή του πρωτοκόλλου I²C

1.3.1 Αμφίδρομη και μονόδρομη μεταφορά δεδομένων

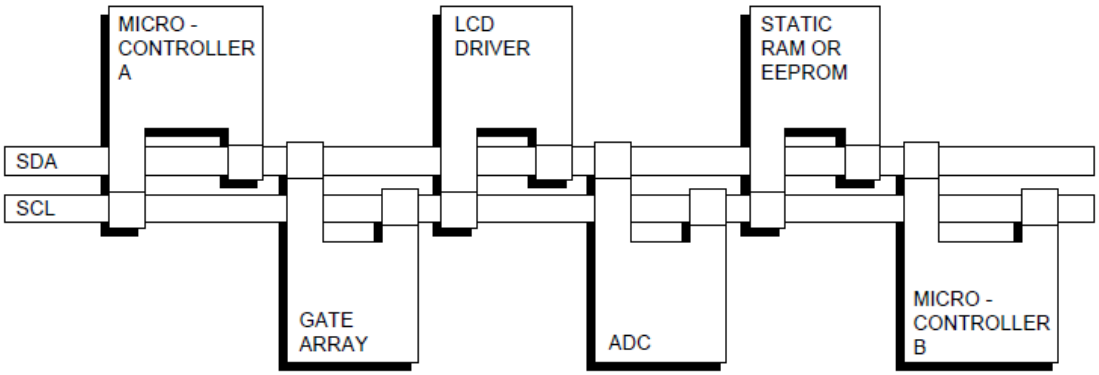
Το πρωτόκολλο I²C ουσιαστικά υποστηρίζει δύο τρόπους λειτουργίας: την αμφίδρομη μεταφορά δεδομένων και τη μονόδρομη. Η λειτουργία όμως σε μονόδρομη μετάδοση δεδομένων αν και επιτρέπει ταχύτητες μετάδοσης μέχρι 5 MHz, βρίσκει εφαρμογή μόνο σε συσκευές που δεν απαιτείται ανατροφοδότηση για τη λειτουργία τους, όπως για παράδειγμα σε κυκλώματα

οδήγησης LED. Γι αυτό το λόγο κρίνεται σκόπιμη στην παρούσα εργασία να γίνει αναφορά στο πρωτόκολλο I²C όταν αυτό υποστηρίζει αμφίδρομη μετάδοση δεδομένων, η οποία συναντάται συχνότερα στις συσκευές του καταναλωτή αλλά και παρέχει όλες τις δυνατότητες του πρωτόκολλου στον ερασιτέχνη και στον επαγγελματία σχεδιαστή ηλεκτρονικών κυκλωμάτων. Το πρωτόκολλο I²C υποστηρίζει αμφίδρομη επικοινωνία στα Standard mode, Fast mode και Fast mode Plus.

1.3.2 Η λειτουργία σε Standard mode, Fast mode και Fast mode Plus

1.3.2.1 Περιγραφή

Οι συσκευές που συνδέονται πάνω στο δίαυλο χρησιμοποιούν δύο γραμμές. Τη γραμμή μεταφοράς σειριακών δεδομένων (SDA: serial data) και τη γραμμή σειριακού ρολογιού για το συγχρονισμό (SCL: serial clock). Κάθε συσκευή συνδεδεμένη στο δίαυλο αναγνωρίζεται από μία μοναδική διεύθυνση. Μία συσκευή κατά την αμφίδρομη λειτουργία μπορεί να είναι πομπός (transmitter) ή δέκτης (receiver) ανάλογα με το εάν στέλνει δεδομένα ή λαμβάνει. Επίσης μπορεί να εκκινεί μία μεταφορά δεδομένων, οπότε λειτουργεί ως “αφέντης” (master) ή να λειτουργεί ως “σκλάβος” (slave) όταν αναφέρεται στη διεύθυνσή της μία συσκευή master. Μία συσκευή μπορεί να είναι τη μία φορά master ενώ την άλλη φορά slave. Έτσι, μία συσκευή μπορεί να είναι master transmitter ή master receiver, αλλά και slave transmitter ή slave receiver [2].



Εικόνα 2. Παράδειγμα σύνδεσης συσκευών σε δίαυλο I²C [2]

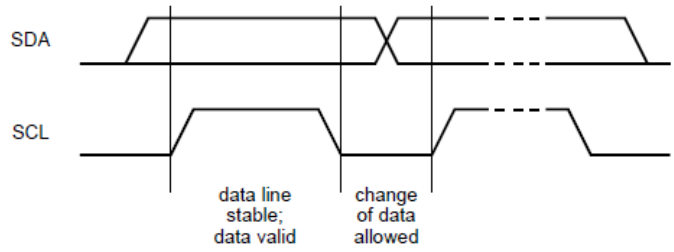
Το πρωτόκολλο I²C υποστηρίζει περισσότερες από μία master συνδεδεμένες πάνω σε ένα δίαυλο. Γι’ αυτό, είναι απαραίτητο να υπάρχει ένας αλγόριθμος που να επιτρέπει σε μία master να καταλαβαίνει πότε είναι ελεύθερος ο δίαυλος για να ξεκινήσει μία μεταφορά δεδομένων, ενώ σε περίπτωση που θέλουν να ξεκινήσουν ταυτόχρονα δύο συσκευές master, να παραχωρείται προτεραιότητα σε μία από τις δύο ώστε να αποφεύγεται η σύγκρουση.

Τον παλμό συγχρονισμού SCL τον παράγει κάθε φορά η master που εκκινεί μία μεταφορά δεδομένων. Επίσης, τα δεδομένα SDA παράγονται αρχικά από τη master έτσι ώστε να προσδιοριστούν οι συσκευές slave τις οποίες αφορά η συγκεκριμένη μετάδοση δεδομένων. Οι παλμοί των δεδομένων στη συνέχεια έχουν ροή από τη master προς τη slave και αντίστροφα σύμφωνα με τη λειτουργία του πρωτοκόλλου I²C και τον τρόπο λειτουργίας της master, εάν είναι δηλαδή master-transmitter ή master-receiver [2].

1.3.2.2 Τα σήματα SCL και SDA

Η κανονική κατάσταση και των δύο γραμμών είναι η κατάσταση HIGH δηλαδή το λογικό “1” και σε αυτή την κατάσταση βρίσκονται οι γραμμές όταν είναι ελεύθερες. Όταν εκπέμπει παλμό κάποια συσκευή στην έξοδο SCL ή SDA τότε ο συγκεκριμένος ακροδέκτης πηγαίνει σε LOW δηλαδή σε λογικό “0”. Η κατάσταση HIGH αντιστοιχεί σε τάση 70% της τάσης τροφοδοσίας, ενώ η γραμμή είναι σε κατάσταση LOW όταν η τάση βρίσκεται κάτω από το 30% της τάσης τροφοδοσίας.

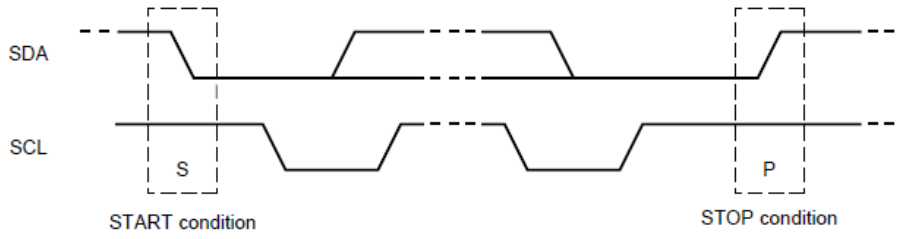
Η αλλαγή κατάστασης στη γραμμή SDA από HIGH σε LOW ή αντίστροφα είναι έγκυρη, μόνο όταν η κατάσταση στη γραμμή SCL είναι LOW. Όταν η SCL είναι σε κατάσταση HIGH, τότε η κατάσταση της SDA πρέπει να παραμένει σταθερή, δηλαδή είτε συνεχώς HIGH είτε συνεχώς LOW [3]. Για κάθε παλμό δεδομένων SDA που μεταδίδεται παράγεται ένας παλμός συγχρονισμού SCL.



Εικόνα 3. Ο συγχρονισμός της γραμμής SDA με την SCL [2]

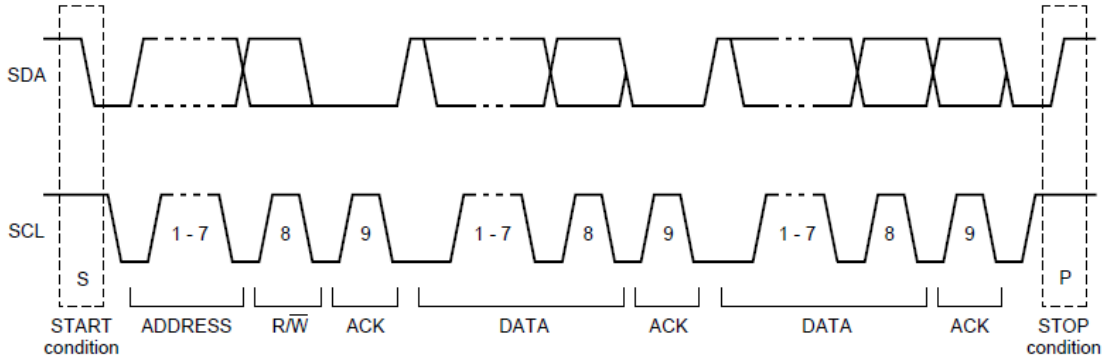
1.3.2.3 Η μετάδοση δεδομένων

Κάθε μετάδοση δεδομένων ξεκινά με μία κατάσταση START (S) και ολοκληρώνεται με μία κατάσταση STOP (P). Και οι δύο καταστάσεις παράγονται από τη συσκευή master. Η κατάσταση START πραγματοποιείται με μετάβαση της γραμμής SDA από HIGH σε LOW όταν η γραμμή SCL είναι σε κατάσταση HIGH, ενώ η κατάσταση STOP με μετάβαση της SDA από LOW σε HIGH με τη γραμμή SCL σε HIGH. Ο δίαυλος παραμένει κατειλημμένος μετά από μία κατάσταση START και μέχρι να υπάρξει μία κατάσταση STOP.



Εικόνα 4. Οι καταστάσεις START και STOP [2]

Μετά από μία κατάσταση START η συσκευή master εκπέμπει τη διεύθυνση της συσκευής slave στην οποία απευθύνεται. Η διεύθυνση αυτή έχει μήκος 7 bit και μετά ακολουθεί το bit R/$\overline{W}$ το οποίο είναι το bit κατεύθυνσης. Με το bit “0” η συσκευή master ζητά να λάβει δεδομένα από τη slave ενώ με το “1” η συσκευή master στέλνει δεδομένα στη slave. Το πρώτο bit του byte που εκπέμπεται είναι το Most Significant Bit (MSB).

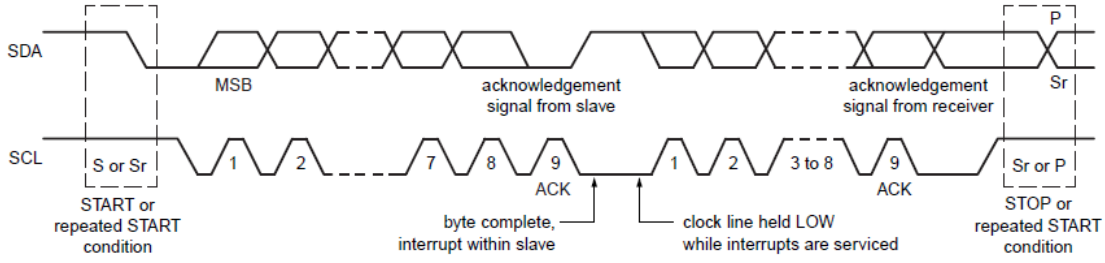


Εικόνα 5. Μία πλήρης μετάδοση δεδομένων [2]

Μετά από την εκπομπή του πρώτου byte αναμένεται το bit επιβεβαίωσης (ACK) από τη συσκευή slave προς τη συσκευή master. Έτσι κατά τη διάρκεια του 9ου παλμού συγχρονισμού, η συσκευή master ελευθερώνει τη γραμμή SDA η οποία ανέρχεται στην κατάσταση HIGH, με σκοπό να απαντήσει η slave με ACK πηγαίνοντας τη γραμμή σε κατάσταση LOW, ή με NACK εάν η γραμμή παραμείνει σε κατάσταση HIGH.

Στη συνέχεια, ακολουθούν τα υπόλοιπα byte αυτής της συνδιαλλαγής της συσκευής master με τη συσκευή slave, ακολουθούμενα πάντα από το bit επιβεβαίωσης ACK με κατεύθυνση από τη συσκευή που λαμβάνει προς τη συσκευή που εκπέμπει. Τα bit των δεδομένων είναι 8 [4].

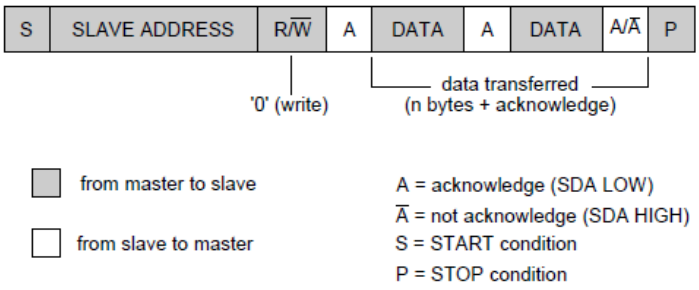
Η συνδιαλλαγή θα τερματιστεί μετά από την κατάσταση STOP που θα παραχθεί από τη master, ή θα συνεχιστεί με κάποια άλλη συσκευή slave, ή με την ίδια με άλλη κατεύθυνση δεδομένων με την παραγωγή από τη master μίας κατάστασης repeated START (Sr). Η κατάσταση repeated START είναι απολύτως ίδια με τη START [2].



Εικόνα 6. Μετάδοση δεδομένων με την επιλογή του repeated START [2]

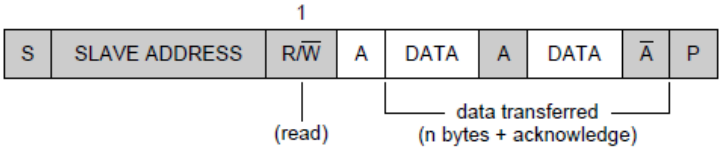
Η μετάδοση των δεδομένων μπορεί να έχει τις παρακάτω μορφές:

- Όταν μία συσκευή master-transmitter εκπέμπει προς μία slave-receiver. Τότε, η κατεύθυνση της μετάδοσης παραμένει από τη master προς τη slave και η slave ενημερώνει τη master για κάθε byte που έλαβε με το bit ACK.



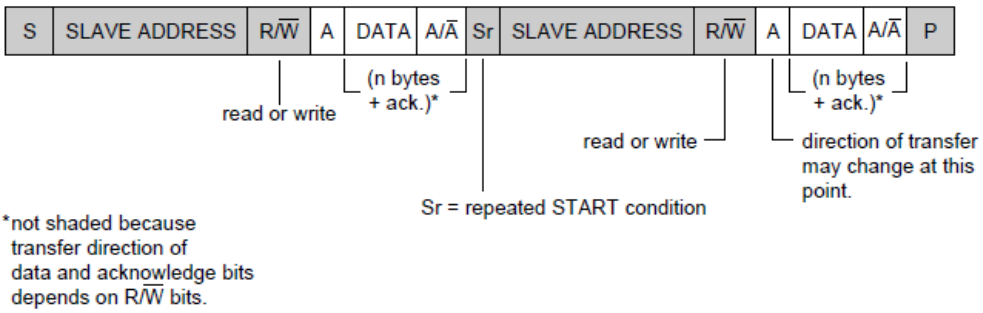
Εικόνα 7. Μία συσκευή master εκπέμπει προς μια slave [2]

- Όταν μία συσκευή master-transmitter αλλάζει σε master-receiver αμέσως μετά το πρώτο byte και την αποδοχή του πρώτου bit ACK. Τότε, η slave-receiver αλλάζει σε slave-transmitter. Η συσκευή master στέλνει τότε bit ACK για κάθε byte που λαμβάνει από τη slave. Η κατάσταση STOP παράγεται όμως πάλι από τη συσκευή master η οποία στέλνει bit NACK $\overline{A}$ ακριβώς πριν την κατάσταση STOP.



Εικόνα 8. Η master διαβάζει αμέσως μετά το πρώτο byte [2]

- Όταν μία συσκευή master παραμένει στο δίαυλο με χρήση repeated START. Τότε, η κατεύθυνση του bit ACK και των δεδομένων αλλάζει κατεύθυνση σύμφωνα με το bit R/$\overline{W}$. Εάν η master-receiver πρέπει να στείλει κατάσταση repeated START, τότε πρέπει πριν να στείλει bit NACK $\overline{A}$.



Εικόνα 9. Μετάδοση με δυνατότητα αλλαγής κατεύθυνσης λόγω του Sr [2]

1.3.2.4 Το Bit επιβεβαίωσης (Acknowledgement Bit ACK)

Η συσκευή που λαμβάνει την πληροφορία πρέπει πάντα να ενημερώνει ότι την έλαβε επιτυχώς εκπέμποντας το bit επιβεβαίωσης ACK.

Όταν μία συσκευή master-transmitter λαμβάνει bit επιβεβαίωσης ACK τότε μπορεί:

- Να μεταδώσει ένα νέο byte δεδομένων
- Να τερματίσει τη μετάδοση με την παραγωγή της κατάστασης STOP
- Να εκκινήσει μια νέα συνδιαλλαγή χωρίς να ελευθερώσει το δίαυλο με την παραγωγή της κατάστασης repeated START

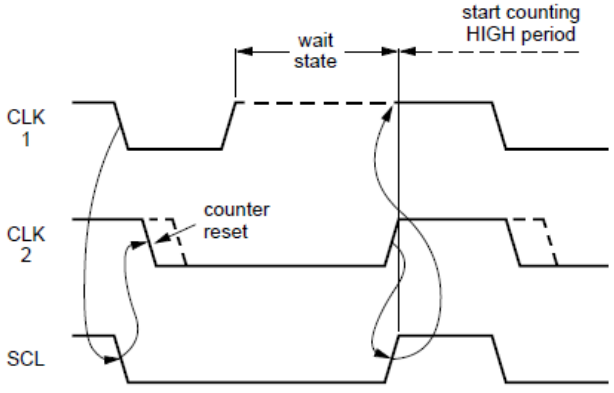
Όμως υπάρχουν και περιπτώσεις που δεν εκπέμπεται ACK αλλά Not ACK (NACK). Αυτό μπορεί να συμβεί:

- όταν η συσκευή που λαμβάνει δεν μπορεί να καταλάβει τα δεδομένα που της αποστέλλονται, ή να λάβει περισσότερα δεδομένα
- όταν δεν υπάρχει slave-receiver που να αντιστοιχεί στην εκπεμπόμενη διεύθυνση, ή η slave-receiver εκτελεί μία λειτουργία πραγματικού χρόνου και δεν μπορεί να επικοινωνήσει με τη master
- όταν μία συσκευή master-receiver θέλει να τερματίσει τη συνδιαλλαγή, οπότε πριν από STOP ή repeated START εκπέμπει NACK $\overline{A}$ [2].

1.3.2.5 Συγχρονισμός ρολογιού και διαιτησία

Όταν υπάρχουν στο δίαυλο δύο ή περισσότερες συσκευές master, τότε πρέπει να υπάρχει κάποια μέθοδος που να καθορίζει ποια master θα αναλαμβάνει τον έλεγχο του διαύλου. Αυτό πραγματοποιείται μέσω του συγχρονισμού του ρολογιού και της διαδικασίας της διαιτησίας. Επειδή κάθε master παράγει το δικό της ρολόι, απαραίτητο είναι να προηγηθεί πρώτα ο συγχρονισμός του ρολογιού.

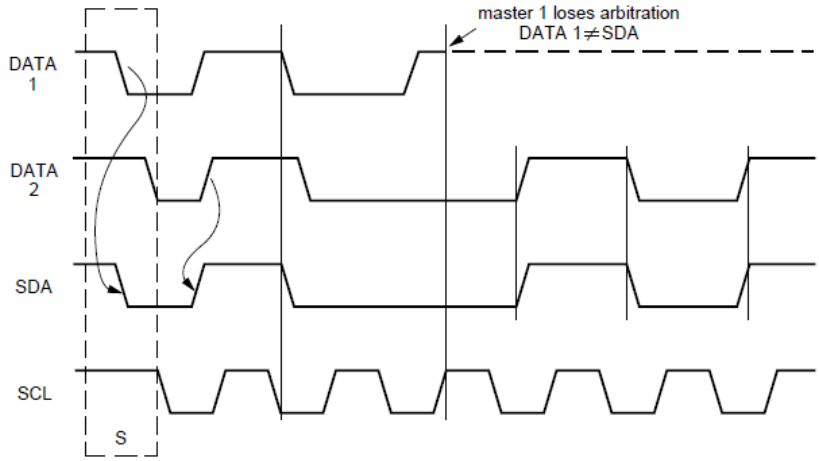
Όλες οι διεπαφές των συσκευών I^2C με τη γραμμή SCL πραγματοποιούνται μέσω συνδέσεων wired-AND. Έτσι, εάν στο ρολόι μιας master συμβεί μετάβαση από κατάσταση HIGH σε κατάσταση LOW, τότε η γραμμή SCL πηγαίνει στη LOW και οι υπόλοιπες συσκευές master που συνδέονται στο δίαυλο ξεκινούν νέα απαρίθμηση της κατάστασης LOW στο ρολόι τους. Η γραμμή SCL παραμένει στη LOW μέχρι να επιστρέψει το ρολόι σε HIGH. Εάν όμως το ρολόι κάποιας άλλης master είναι σε κατάσταση LOW, τότε η γραμμή SCL θα παραμείνει σε LOW μέχρι αυτής το ρολόι μεταβεί σε κατάσταση HIGH. Οι masters με μικρότερες περιόδους στην κατάσταση LOW εισάγουν καθυστέρηση αναμονής HIGH (wait-state), ώστε να παραμείνουν σε κατάσταση HIGH για περισσότερο χρόνο πριν να μεταβούν πάλι σε κατάσταση LOW. Η πρώτη συσκευή master που θα ολοκληρώσει την κατάσταση HIGH στο ρολόι της θα τραβήξει σε κατάσταση LOW τη γραμμή SCL [2].



Εικόνα 10. Ο συγχρονισμός του ρολογιού [2]

Με αυτό τον τρόπο το παραγόμενο ρολόι στη γραμμή SCL έχει περίοδο LOW που καθορίζεται από τη master με τη μεγαλύτερη περίοδο LOW στο ρολόι της, και περίοδο HIGH από τη master με τη μικρότερη περίοδο HIGH.

Μία συσκευή master ξεκινά τη μετάδοση δεδομένων μόνο όταν ο δίαυλος είναι ελεύθερος. Τότε αρχικά η συσκευή master δημιουργεί μία κατάσταση START. Κατά τη διάρκεια όμως του παλμού START υπάρχει περίπτωση να εκπέμψει παλμό START και μία δεύτερη συσκευή master, εάν προηγουμένως είχε ανιχνεύσει και αυτή ότι ο δίαυλος είναι ελεύθερος. Τότε πρέπει να ξεκινήσει η διαδικασία της διαιτησίας (arbitration), διότι τη μετάδοση μόνο μία master θα πρέπει να την ολοκληρώσει, ενώ η δεύτερη master θα πρέπει να διακόψει τη μετάδοσή της.



Εικόνα 11. Η διαδικασία της διαιτησίας με δύο master [2]

Κάθε συσκευή master ελέγχει εάν η κατάσταση στη γραμμή SDA συμφωνεί με κάθε bit των δεδομένων που στέλνει. Στην περίπτωση, επομένως, που δύο συσκευές master στέλνουν ταυτόχρονα δεδομένα τα οποία είναι ίδια μεταξύ τους, υπάρχει συμφωνία μεταξύ της κατάστασης στη γραμμή SDA και των δεδομένων που στέλνουν κι έτσι οι δύο master θεωρούν ότι η μετάδοση των δεδομένων γίνεται με επιτυχία και συνεχίζουν τη μετάδοση. Όταν όμως κάποια συσκευή master διαπιστώσει ότι ενώ έστειλε HIGH η κατάσταση στην SDA είναι LOW, τότε σταματά να στέλνει δεδομένα καθώς αντιλαμβάνεται ότι έχασε στη διαιτησία. Τα δεδομένα που δεν εστάλησαν σωστά θα σταλούν ξανά όταν ελευθερωθεί ο δίαυλος.

Στην περίπτωση που μία συσκευή master λειτουργεί και ως slave, θα πρέπει όταν χάσει στη διαιτησία να γυρίσει αμέσως σε λειτουργία slave, μήπως η συσκευή master που κέρδισε στη διαιτησία και συνεχίζει να μεταδίδει, στέλνει δεδομένα που αφορούν τη διεύθυνσή της [2].

1.3.2.6 Επιμήκυνση ρολογιού

Το πρωτόκολλο I^2C διευκολύνει τη συνδιαλλαγή των συσκευών που συνδέονται στο δίαυλο με τη δυνατότητα της επιμήκυνσης του ρολογιού (clock stretching). Αυτή είναι μία επιπλέον δυνατότητα που δεν περιλαμβάνεται σε όλες τις συσκευές slave και με την οποία η συσκευή slave μπορεί να προκαλέσει παύση σε μία συνδιαλλαγή με την κράτηση της γραμμής SCL σε κατάσταση LOW.

Όταν για παράδειγμα μία συσκευή receiver χρειάζεται περισσότερο χρόνο για να αποθηκεύσει το byte που έλαβε και να προετοιμαστεί για τη λήψη του επόμενου byte, μπορεί να κρατήσει τη γραμμή SCL σε κατάσταση LOW μετά τη λήψη του byte και την αποστολή του bit ACK. Έτσι, η

συσκευή transmitter αναγκάζεται να εισέλθει στη wait state μέχρι η συσκευή receiver να καταστεί έτοιμη για τη λήψη του επόμενου byte.

Επιμήκυνση του ρολογιού μπορεί να συμβεί και μετά από κάθε bit ρολογιού. Για παράδειγμα, ένας επεξεργαστής χρησιμοποιεί την επιμήκυνση του ρολογιού και επεκτείνει την περίοδο της κατάστασης LOW, έτσι ώστε να μειώσει τη συχνότητα του ρολογιού του διαύλου για να συμβαδίζει με το δικό του ρολόι [2].

1.3.2.7 Εκχώρηση διευθύνσεων

1.3.2.7.1 Διεύθυνση γενικής κλήσης

Σε ένα δίαυλο I²C μπορεί να υπάρξουν 128 διαφορετικές διευθύνσεις, οι οποίες προκύπτουν από το συνδυασμό των 7 bits που αποτελούν τη διεύθυνση slave. Από αυτές, οι 8 είναι δεσμευμένες για κάποιο σκοπό, αλλά μπορεί να χρησιμοποιηθούν από τον σχεδιαστή εάν δεν απαιτούνται οι συγκεκριμένες λειτουργίες.

X = don't care; 1 = HIGH; 0 = LOW.

Slave address	R/W bit	Description
0000 000	0	general call address[1]
0000 000	1	START byte[2]
0000 001	X	CBUS address[3]
0000 010	X	reserved for different bus format[4]
0000 011	X	reserved for future purposes
0000 1XX	X	Hs-mode master code
1111 1XX	1	device ID
1111 0XX	X	10-bit slave addressing

[1] The general call address is used for several functions including software reset.

[2] No device is allowed to acknowledge at the reception of the START byte.

[3] The CBUS address has been reserved to enable the inter-mixing of CBUS compatible and I²C-bus compatible devices in the same system. I²C-bus compatible devices are not allowed to respond on reception of this address.

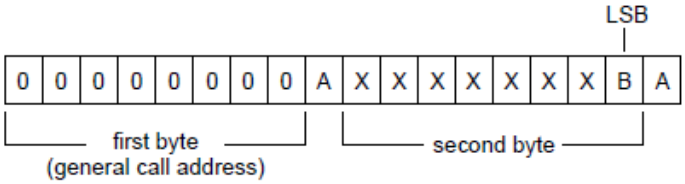
[4] The address reserved for a different bus format is included to enable I²C and other protocols to be mixed. Only I²C-bus compatible devices that can work with such formats and protocols are allowed to respond to this address.

Πίνακας 1. Οι εκχωρημένες διευθύνσεις [2]

Στον πίνακα 1 παρατηρούμε τη διεύθυνση γενικής κλήσης (general call address). Τη διεύθυνση αυτή τη χρησιμοποιεί μια συσκευή master όταν θέλει να απευθυνθεί σε όλες τις συσκευές που είναι συνδεδεμένες στο δίαυλο εκείνη τη στιγμή. Όταν κάποια συσκευή περιμένει δεδομένα μέσω της διεύθυνσης γενικής κλήσης, τότε αποδέχεται τη διεύθυνση και απαντά στη master με bit επιβεβαίωσης ACK, διαφορετικά την αγνοεί και δεν απαντά. Η πληροφορία της διεύθυνσης γενικής κλήσης μεταδίδεται με το δεύτερο byte. Κάθε slave-receiver που μπορεί να μεταχειριστεί την πληροφορία του δεύτερου byte καθώς και των bytes που τυχόν ακολουθούν απαντά με bit επιβεβαίωσης.

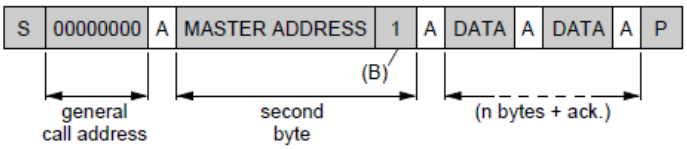
Καθοριστικό ρόλο έχει το λιγότερο σημαντικό bit (LSB) B του δεύτερου byte. Εάν είναι “0”, τότε το δεύτερο byte μπορεί να είναι μόνο 0000 0110, ή 0000 0100, είτε 0000 0000. Στην πρώτη

περίπτωση, οι συσκευές που αποδέχονται τη διεύθυνση γενικής κλήσης κάνουν πρώτα reset και προγραμματίζονται, στη δεύτερη προγραμματίζονται χωρίς reset, ενώ η τελευταία περίπτωση δεν μπορεί να υπάρξει ως δεύτερο byte. Οι άλλοι συνδυασμοί byte δεν έχουν καθοριστεί.



Εικόνα 12. Η διεύθυνση γενικής κλήσης [2]

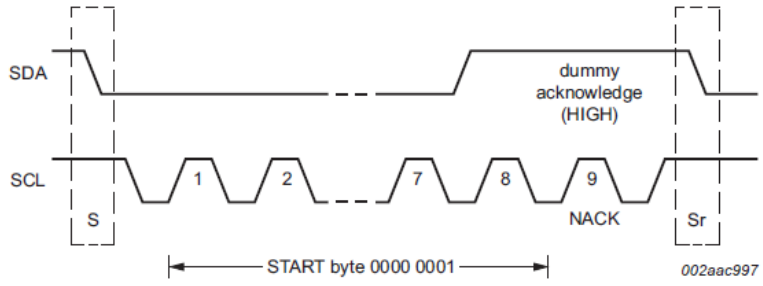
Εάν το LSB B είναι “1” τότε η γενική κλήση είναι “γενική κλήση υλικού” (hardware general call). Αυτή η κλήση προέρχεται από μία hardware master συσκευή, όπως έναν ανιχνευτή πληκτρολογίου, η οποία καθώς δεν γνωρίζει προς τα πού να μεταδώσει την πληροφορία, δημιουργεί μια γενική κλήση υλικού τοποθετώντας στα υπόλοιπα 7 bit του δεύτερου byte τη δική της διεύθυνση. Έτσι, μία έξυπνη συσκευή όπως ένας μικροεπεξεργαστής θα αναγνώσει τη διεύθυνση της hardware master και θα αποδεχθεί την πληροφορία [2].



Εικόνα 13. Μετάδοση από συσκευή hardware master [2]

1.3.2.7.2 Το START byte

Η διεύθυνση slave μπορεί επίσης να παίξει το ρόλο του start byte σε μία νέα μετάδοση. Αυτό είναι απαραίτητο στην περίπτωση που η συσκευή slave, για παράδειγμα ένας επεξεργαστής, δεν διαθέτει υλικό διασύνδεσης I^2C και συνεπώς είναι υποχρεωμένη να δειγματοληπτεί το δίαυλο με λογισμικό, ώστε να ανιχνεύσει την πληροφορία που την αφορά. Και αυτό διότι ο μικροεπεξεργαστής είναι αναγκασμένος να δειγματοληπτεί το δίαυλο με υψηλή συχνότητα, γεγονός που καθυστερεί τη λειτουργία του. Με την ύπαρξη όμως του start byte η δειγματοληψία του διαύλου αρκείται σε χαμηλότερη συχνότητα, καθώς αυτή η διαδικασία απαιτεί περισσότερο χρόνο.



Εικόνα 14. Διαδικασία έναρξης μετάδοσης με START byte [2]

Μετά τη μετάδοση της κατάστασης START (S), η συσκευή master μεταδίδει το start byte (0000 0001). Όταν ανιχνευτούν τα 7 μηδενικά του start byte από το μικροεπεξεργαστή slave, τότε αυξάνει τη συχνότητα που δειγματοληπτεί το δίαυλο, με σκοπό να ανιχνεύσει την κατάσταση repeated START (Sr) την οποία θα χρησιμοποιήσει για το συγχρονισμό [2].

1.3.2.7.3 Ταυτότητα συσκευής

Στον πίνακα επίσης βλέπουμε το πεδίο Device ID. Αυτό είναι προαιρετικό και περιγράφεται από λέξη read only των 3 bytes. Παρέχει πληροφορίες, οι οποίες είναι η επωνυμία του κατασκευαστή σε 12 bits (π.χ. NXP), ο κωδικός του εξαρτήματος σε 9 bits (π.χ. PCA9698) και η σειρά αναθεώρησης σε 3 bits (π.χ. RevX).

1.3.2.7.4 Διεύθυνση CBUS

Η διεύθυνση slave 0000 001 είναι δεσμευμένη για να μπορούν να συνυπάρξουν σε ένα σύστημα συσκευές συμβατές με δίαυλο I^2C μαζί με συσκευές συμβατές με δίαυλο CBUS.

1.3.2.7.5 Διευθύνσεις των 10 bit

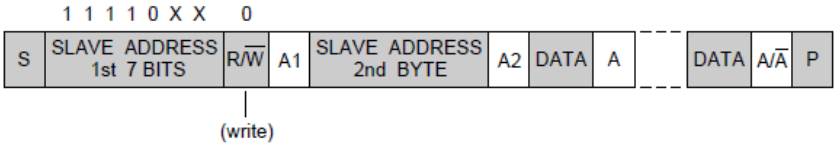
Η διεύθυνση slave 1111 0XX είναι δεσμευμένη για να πραγματοποιούνται αναφορές σε διευθύνσεις των 10-bit (10-bit addressing).

1.3.2.8 Αναφορές σε διευθύνσεις των 10 bit (10-bit addressing)

Στο δίαυλο I^2C μπορεί να συνδεθούν συσκευές με διευθύνσεις των 7 bits αλλά και των 10 bits. Διευθύνσεις των 10 bits χρησιμοποιούνται ώστε να διευρυνθεί το πλήθος των πιθανών διευθύνσεων, αλλά όμως η χρήση τους δεν είναι τόσο διαδεδομένη.

Τα 10 bits προκύπτουν από τα 2 bits του πρώτου byte 1111 0XX και τα 8 bits XXXX XXXX του δεύτερου byte που ακολουθεί. Τα 2 bits (XX) του πρώτου byte αποτελούν τα πιο σημαντικά bits MSB (Most Significant Bits).

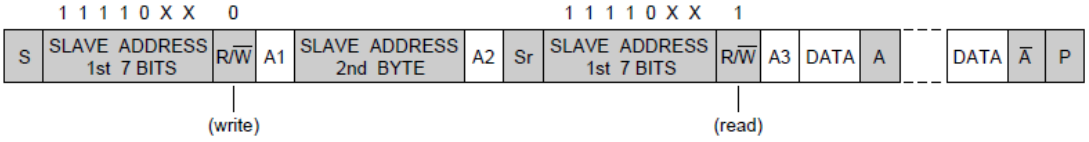
Σε μια συνδιαλλαγή master-transmitter με slave-receiver, μετά την κατάσταση START (S), μεταδίδεται το πρώτο byte της διεύθυνσης slave. Οι συσκευές που βρίσκονται στο δίαυλο συγκρίνουν τα bits (1111 0XX) με τη δική τους διεύθυνση και ελέγχουν εάν το bit R/$\overline{W}$ είναι “0”. Όσες συσκευές συμφωνούν με αυτή την πληροφορία εκπέμπουν bit επιβεβαίωσης A1. Στη συνέχεια η master μεταδίδει το δεύτερο byte της διεύθυνσης slave (XXXX XXXX). Με αυτή τη διεύθυνση θα βρεθεί να συμφωνεί μία μόνο συσκευή, η οποία θα εκπέμψει το bit επιβεβαίωσης A2. Η συνδιαλλαγή master-slave θα συνεχιστεί μέχρι να υπάρξει κατάσταση STOP (P) ή repeated START (Sr) [2].



Εικόνα 15. Συνδιαλλαγή master-transmitter με slave-receiver με διεύθυνση των 10 bit [2]

Στην περίπτωση που έχουμε μία συνδιαλλαγή master-receiver με slave-transmitter ισχύουν τα ίδια όπως και πριν μέχρι και το bit επιβεβαίωσης A2. Για να δηλώσει η συσκευή master ότι θέλει να ζητήσει δεδομένα από τη slave και να γίνει δηλαδή receiver, εκπέμπει μετά τη λήψη του bit A2, κατάσταση repeated START (Sr) και ξανά πάλι τα πρώτα 7 bits της slave address ακολουθούμενη από bit ανάγνωσης R/$\overline{W}$ ίσο με “1”. Η συσκευή slave συγκρίνει πάλι τα 7 bits με τη δική της διεύθυνση εάν συμφωνούν, και καταλαβαίνει από το bit ανάγνωσης ότι πρέπει να στείλει δεδομένα προς τη master. Στη συνέχεια εκπέμπει το bit επιβεβαίωσης A3 και τα υπόλοιπα

δεδομένα. Η συνδιαλλαγή θα συνεχιστεί μέχρι να υπάρξει κατάσταση STOP (P) ή repeated START (Sr).



Εικόνα 16. Συνδιαλλαγή master-receiver με slave-transmitter με διεύθυνση των 10 bit [2]

1.4 Χρήσεις του πρωτοκόλλου I²C σε αρχιτεκτονικές συστημάτων

Το πρωτόκολλο I²C μπορεί να χρησιμοποιηθεί σε αρχιτεκτονικές συστημάτων όπως έχει, ή βελτιωμένο με περισσότερες δυνατότητες, ή να είναι συμβατό και να συνυπάρχει με άλλα πρωτόκολλα επικοινωνίας.

Έτσι, σε ένα δίαυλο I²C μπορεί να συνδεθούν συσκευές receiver που είναι συμβατές με το πρωτόκολλο CBUS. Το πρωτόκολλο CBUS είναι διαφορετικό από το I²C, γι' αυτό οι συσκευές που είναι συμβατές με I²C δεν πρέπει να ανταποκρίνονται σε μηνύματα CBUS. Όταν επομένως πρόκειται οι master-transmitters να επικοινωνήσουν με συσκευές CBUS, εκπέμπεται στο δίαυλο η δεσμευμένη διεύθυνση (0000 001X) την οποία δεν δέχονται οι συσκευές I²C αλλά οι συμβατές με CBUS. Στη συνέχεια ελευθερώνεται μία επιπλέον γραμμή, η DLEN μέσω της οποίας εκπέμπονται τα μηνύματα CBUS. Μετά την εκπομπή κατάστασης STOP όλες οι συσκευές είναι έτοιμες να λάβουν ξανά.

Το πρωτόκολλο I²C μπορεί επίσης να χρησιμοποιηθεί σε έναν δίαυλο SMBus (System Management Bus). Ο SMBus χρησιμοποιεί το υλικό του I²C καθώς και το υλικό διευθυνσιοδότησης του I²C. Όμως, επιπλέον έχει προστεθεί σε αυτό ένα πρωτόκολλο επίλυσης διευθύνσεων με σκοπό η κατανομή των διευθύνσεων να γίνεται δυναμικά. Αυτό έχει το πλεονέκτημα καινούριες συσκευές να προστίθενται στο δίαυλο ενώ βρίσκεται σε λειτουργία, χωρίς επανεκκίνηση του συστήματος. Με αυτό τον τρόπο, κάθε νέα συσκευή που συνδέεται στο δίαυλο αναγνωρίζεται αυτόματα και αποκτά μία μοναδική διεύθυνση και η διασύνδεση καθίσταται plug and play.

Μία επιπλέον βελτίωση που εισήγαγε το πρωτόκολλο SMBus είναι η δυνατότητα time-out. Σύμφωνα με αυτό το χαρακτηριστικό, οι συσκευές επαναφέρονται (reset) όταν μία επικοινωνία διαρκεί πολύ χρόνο. Αυτή η καθυστέρηση δεν μπορεί να ξεπερνά τα 35 ms. Γι αυτό το λόγο, το ρολόι του SMBus έχει κατώτερη συχνότητα τα 10 KHz, ενώ το I²C δεν έχει περιορισμό και φτάνει μέχρι τη συχνότητα 0 KHz. Μεταξύ επίσης των δύο πρωτοκόλλων υπάρχει μία μικρή διαφορά στις λογικές στάθμες των LOW και HIGH. Δύο εκδόσεις υπάρχουν του SMBus, η 1.0 (Low power) και η 2.0 (High power), με διαφορές στα ηλεκτρικά χαρακτηριστικά τους.

Στο πρωτόκολλο SMBus 1.1 βασίζεται το πρωτόκολλο PMBus (Power Management Bus). Ένας εξυπηρετητής συστήματος (system host) χρησιμοποιεί το PMBus, για να επικοινωνήσει και να ελέγξει τους πολλαπλούς μετατροπείς ισχύος που μπορεί να υπάρχουν σε ένα σύστημα.

Μία αρχιτεκτονική η οποία βασίζεται στο πρωτόκολλο I²C είναι η IPMI (Intelligent Platform Management Interface). Είναι εκείνο το υλικό και λογισμικό που αναλαμβάνει την ευφυή διαχείριση του υλικού ενός συστήματος, έτσι ώστε να προβλέψει τυχόν προβλήματα στη λειτουργία του και να βοηθήσει στην επίλυσή τους. Επίσης, μπορεί να παρέχει λειτουργίες διαχείρισης, όπως αυτόματους συναγερμούς, αυτόματη επανεκκίνηση του συστήματος ή κλείσιμό του, απομακρυσμένο έλεγχο της ενεργοποίησής του, αλλά και εποπτεία κάποιων παραμέτρων του συστήματος, όπως θερμοκρασίες, τάσεις, ανεμιστήρες κ.λπ.

Αξίζει να γίνει επίσης αναφορά στο Display Data Channel (DDC), το οποίο έχει σκοπό την ψηφιακή επικοινωνία μεταξύ μίας συσκευής απεικόνισης και του προσαρμογέα γραφικών. Η συσκευή απεικόνισης ενημερώνει μέσω του DDC τον προσαρμογέα σχετικά με την ταυτότητά της, τους τρόπους λειτουργίας της και της δυνατότητές της, ενώ μπορεί να γίνει ο έλεγχος κάποιων παραμέτρων της συσκευής, όπως της φωτεινότητας και της αντίθεσης. Το DDC υπάρχει σε αρκετές εκδόσεις. Η πιο κοινή είναι η DDC2 και η οποία είναι βασισμένη στο πρωτόκολλο I^2C [2].

ΚΕΦΑΛΑΙΟ 2 - Το πρωτόκολλο επικοινωνίας SPI

2.1 Περιγραφή του SPI

2.1.1 Εισαγωγή

Το πρωτόκολλο SPI (Serial Peripheral Interface) αναπτύχθηκε από την εταιρία Motorola, μετέπειτα Freescale Semiconductor, τωρινή NXP, στη δεκαετία του 70'. Πρωτοεμφανίστηκε με την παρουσίαση του επεξεργαστή 68000 της Motorola το 1979 ως ένας από τους διαύλους επικοινωνίας του επεξεργαστή με τις περιφερειακές του συσκευές. Πολύ σύντομα το πρωτόκολλο αυτό καθιερώθηκε από πολλές κατασκευάστριες εταιρίες και είναι επίσης γνωστό ως "four-wire" σειριακός δίαυλος [3], [5], [6], [7].

Χρησιμοποιείται κυρίως σε επίπεδο πλακέτας ή σε πολύ κοντινές αποστάσεις για την επικοινωνία μεταξύ μίας συσκευής master με πολλές συσκευές slave.Τέτοιες συσκευές slave μπορεί να είναι:

- Μνήμες Flash, SRAM, EEPROM
- Μετατροπείς DAC και ADC
- LCD s
- Διάφοροι αισθητήρες: θερμοκρασίας, ατμοσφαιρικής πίεσης, επιτάχυνσης κ.λπ. [7],[8],[9]

Για το δίαυλο χρησιμοποιούνται τέσσερα καλώδια και υποστηρίζονται ταχύτητες σύγχρονης επικοινωνίας τουλάχιστον 10Mbps. Μπορεί να χρησιμοποιηθεί για επικοινωνίες εκτός πλακέτας αλλά με μειωμένες ταχύτητες μετάδοσης, χάνοντας έτσι το πλεονέκτημα της υψηλής ταχύτητας στην επικοινωνία [10].

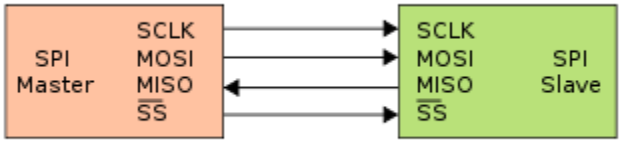
Σπάνια επίσης είναι η χρήση του διαύλου από πέραν της μίας master συσκευής, παρόλο που το πρωτόκολλο επιτρέπει επικοινωνίες multi-master. Το σύνηθες είναι να συνδέονται στο δίαυλο ένας επεξεργαστής με τις περιφερειακές συσκευές του. Η μετάδοση των δεδομένων μπορεί να είναι πλήρως αμφίδρομη 4-wire full duplex, ή εναλλάξ αμφίδρομη 3-wire half duplex [9].

2.1.2 Τα σήματα του πρωτοκόλλου SPI

Το πρωτόκολλο στη βασική του μορφή χρησιμοποιεί δύο σήματα ελέγχου (SS και SCLK) και δύο σήματα δεδομένων (MOSI και MISO):

- Το σήμα Slave Select (SS), που χρησιμοποιείται για την επιλογή της ενεργού συσκευής slave με την οποία θα ανταλλάξει δεδομένα η συσκευή master. Το σήμα SS από τη master καταλήγει στον ακροδέκτη SS ή CS (Chip Select) της συσκευής slave.
- Το σήμα Serial Clock (SCLK ή SCK), το οποίο παράγεται από τη συσκευή master για το συγχρονισμό της μεταφοράς δεδομένων μεταξύ των συσκευών master και slave.
- Το σήμα Master Output Slave Input (MOSI), που είναι το σήμα που μεταφέρει τα δεδομένα από τη master στη slave. Το σήμα MOSI από τη master καταλήγει στον ακροδέκτη MOSI ή SDI (Serial Data Input) της slave. Ο ακροδέκτης MOSI όταν η συσκευή λειτουργεί ως master μεταδίδει δεδομένα, ενώ όταν λειτουργεί ως slave δέχεται δεδομένα.
- Το σήμα Master Input Slave Output (MISO), που είναι το σήμα που μεταφέρει τα δεδομένα από τη slave στη master. Το σήμα από τον ακροδέκτη MISO ή SDO (Serial Data Output) της slave καταλήγει στον ακροδέκτη MISO της master. Ο ακροδέκτης MISO όταν η συσκευή λειτουργεί ως master δέχεται δεδομένα, ενώ όταν λειτουργεί ως slave μεταδίδει δεδομένα [10], [11].

Η επικοινωνία γίνεται κάθε φορά μεταξύ μίας συσκευής master με μία συσκευή slave, ανεξάρτητα του αριθμού των συσκευών master και slave που μπορεί να συνδέονται στο δίαυλο SPI. Η έναρξη της επικοινωνίας γίνεται από τη συσκευή master με την παραγωγή του σήματος SCLK και την επιλογή της επιθυμητής συσκευής slave, οδηγώντας τη γραμμή SS (ή CS) της συγκεκριμένης slave σε κατάσταση LOW.



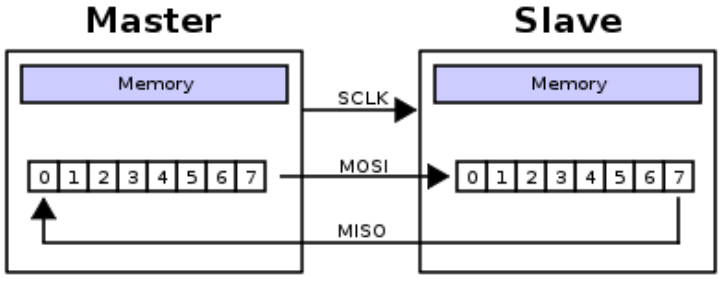
Εικόνα 17. Σύνδεση SPI μίας συσκευής master με μία slave [6]

Δεν είναι απαραίτητη η χρήση και των τεσσάρων σημάτων σε μια επικοινωνία. Αν για παράδειγμα υπάρχει μόνο μία συσκευή master και μία συσκευή slave, τότε η γραμμή SS δεν χρησιμοποιείται και είναι αρκετό να γειωθεί μόνιμα ο ακροδέκτης SS της slave (κατάσταση LOW). Επίσης, αν η slave είναι ένας ελεγκτής LCD (Liquid Crystal Display), τότε δεν χρησιμοποιείται ο ακροδέκτης MISO (ή SDO) της slave. Αν η slave είναι ένας μετατροπέας ADC (Analog to Digital Converter), τότε δεν χρησιμοποιείται ο ακροδέκτης MOSI (ή SDI) της slave [10].

2.1.3 Περιγραφή μιας master-slave επικοινωνίας

Η συσκευή master ρυθμίζει τη συχνότητα του σειριακού ρολογιού, τυπικά μερικά MHz, ώστε να είναι μικρότερη ή το πολύ ίση με τη μέγιστη συχνότητα που υποστηρίζει η συσκευή slave με την οποία πρόκειται να επικοινωνήσει. Στη συνέχεια επιλέγει την επιθυμητή συσκευή slave οδηγώντας τον ακροδέκτη SS της slave σε κατάσταση LOW. Οι υπόλοιπες συσκευές slave που ίσως βρίσκονται στο δίαυλο και οι οποίες δεν έχουν επιλεγεί με τον ανωτέρω τρόπο, απορρίπτουν τα σήματα του ρολογιού και των δεδομένων. Όλες οι συσκευές που συνδέονται στο δίαυλο θα πρέπει να διαθέτουν εξόδους τριών καταστάσεων (tri-state outputs: κατάσταση LOW, κατάσταση HIGH και κατάσταση υψηλής εμπέδησης), έτσι ώστε όταν οι συσκευές είναι ανενεργές, οι ακροδέκτες τους να παραμένουν σε κατάσταση υψηλής εμπέδησης και να μην επηρεάζουν τις αντίστοιχες γραμμές του διαύλου [6], [10].

Σε κάθε κύκλο του ρολογιού πραγματοποιείται μια πλήρης αμφίδρομη επικοινωνία. Η συσκευή master μεταδίδει ένα bit πληροφορίας από τη γραμμή MOSI. Ύστερα, η slave λαμβάνει το bit από την ίδια γραμμή MOSI και μεταδίδει ένα bit προς τη master από τη γραμμή MISO. Τέλος, η master λαμβάνει το bit από την ίδια γραμμή MISO.



Εικόνα 18. Οι δύο καταχωρητές ολίσθησης της master και της slave σχηματίζουν μία ενιαία κυκλική ενδιάμεση μνήμη (buffer) [6]

Για τη μετάδοση των δεδομένων χρησιμοποιούνται ένας καταχωρητής ολίσθησης στη συσκευή master και ένας καταχωρητής ολίσθησης (shift register) στη συσκευή slave για ιδίου μεγέθους λέξη, συνήθως 8 bit. Στην έξοδο κάθε καταχωρητή ολισθαίνει συνήθως το πιο σημαντικό ψηφίο (MSB), ενώ στην είσοδο έρχεται ένα νέο λιγότερο σημαντικό ψηφίο (LSB). Για παράδειγμα, το MSB του καταχωρητή της master ολισθαίνει στο LSB του καταχωρητή της slave και το MSB της slave ολισθαίνει στο LSB της master. Οι δύο καταχωρητές μοιάζουν να είναι συνδεσμολογημένοι σε μια νοητή τοπολογία δακτυλίου. Με αυτό τον τρόπο οι δύο καταχωρητές ανταλλάσσουν τα δεδομένα τους και πραγματοποιούνται τόσοι κύκλοι ρολογιού, έως ότου ολοκληρωθεί η ανταλλαγή όλων των δεδομένων και η συσκευή master να διακόψει την επικοινωνία με τη slave [6].

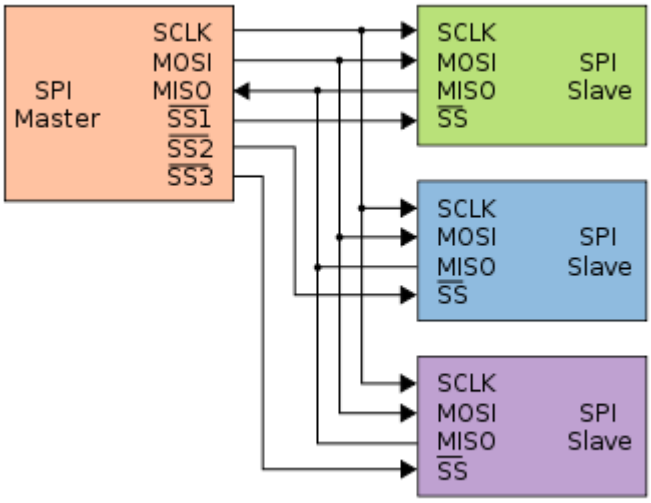
2.1.4 Διατάξεις συνδεσμολογιών

Μία συσκευή master, για παράδειγμα ένας μικροεπεξεργαστής, μπορεί να συνδεθεί με τις συσκευές slave με τις οποίες θέλει να επικοινωνήσει, με δύο τρόπους:

- Οι συσκευές slave σε ανεξάρτητη σύνδεση (independent), ή αλλιώς σε παράλληλη σύνδεση
- Οι συσκευές slave σε αλυσιδωτή σύνδεση (daisy chain), ή αλλιώς σε διαδοχική (cascaded) σύνδεση

2.1.4.1 Οι συσκευές slave σε ανεξάρτητη σύνδεση

Αυτή η συνδεσμολογία είναι η πιο συνηθισμένη για την επικοινωνία μίας συσκευής master με πολλές συσκευές slave. Η επιλογή της επιθυμητής συσκευής slave γίνεται από τη master μέσω της αντίστοιχης κάθε φορά γραμμής SS (ή CS), για κάθε δηλαδή συσκευή slave υπάρχει ξεχωριστή γραμμή SS που ξεκινά από τη master. Όλες οι γραμμές MOSI είναι μεταξύ τους συνδεμένες, όπως επίσης είναι μεταξύ τους συνδεμένες οι γραμμές MISO και SCLK αντίστοιχα [10].



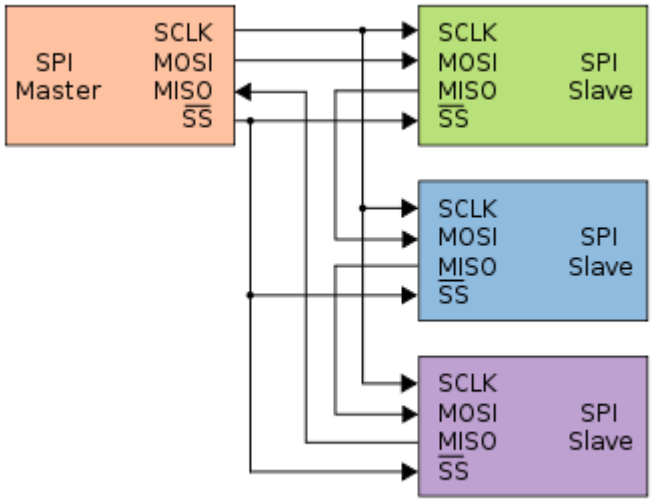
Εικόνα 19. Σύνδεση μίας συσκευής master με τρεις ανεξάρτητες slave [6]

2.1.4.2 Οι συσκευές slave σε διαδοχική σύνδεση

Στη διαδοχική σύνδεση η πρώτη συσκευή slave παίρνει είσοδο δεδομένων από την έξοδο της master, ενώ η έξοδός της συνδέεται στην είσοδο της επόμενης συσκευής slave, η οποία με τη

σειρά της κατευθύνει την έξοδό της στην είσοδο της επόμενης από αυτήν slave. Η έξοδος της τελευταίας slave συνδέεται στην είσοδο της master. Η έξοδος καθεμίας συσκευής slave κατά τη διάρκεια της δεύτερης ομάδας παλμών συγχρονισμού, στέλνει ένα ακριβές αντίγραφο των δεδομένων, τα οποία λήφθηκαν κατά τη διάρκεια της πρώτης ομάδας παλμών συγχρονισμού. Με αυτό τον τρόπο σχηματίζεται ένας εκτεταμένος καταχωρητής ολίσθησης και οι συσκευές slave που συνδέονται στο δίαυλο φαίνονται σαν μία.

Όλες οι γραμμές επιλογής slave (SS) συνδέονται μεταξύ τους και ανάλογα, όλες οι γραμμές παλμών συγχρονισμού (SCLK) [6], [10], [11].



Εικόνα 20. Οι συσκευές slave σε διαδοχική σύνδεση [6]

2.1.5 Πολικότητα και φάση των παλμών συγχρονισμού

Εκτός από τη συχνότητα των παλμών συγχρονισμού, η συσκευή master για να επικοινωνήσει επιτυχώς με μία συσκευή slave θα πρέπει να ρυθμίσει κατάλληλα την πολικότητα και τη φάση των παλμών συγχρονισμού, έτσι ώστε να ταιριάζει με τις αντίστοιχες παραμέτρους με τις οποίες λειτουργεί η συγκεκριμένη συσκευή slave. Η παράμετρος που αντιπροσωπεύει την πολικότητα του ρολογιού είναι η CPOL, ενώ η παράμετρος που αντιστοιχεί στην φάση του ρολογιού είναι η CPHA.

Η παράμετρος CPOL παίρνει τις τιμές “0” ή “1” όπως και η CPHA τις τιμές “0” και “1”. Το σύνολο των συνδυασμών αυτών των τιμών μας δίνει τους τέσσερις δυνατούς τρόπους λειτουργίας του ρολογιού συγχρονισμού [10].

SPI Mode	CPOL	CPHA
0	0	0
1	0	1
2	1	0
3	1	1

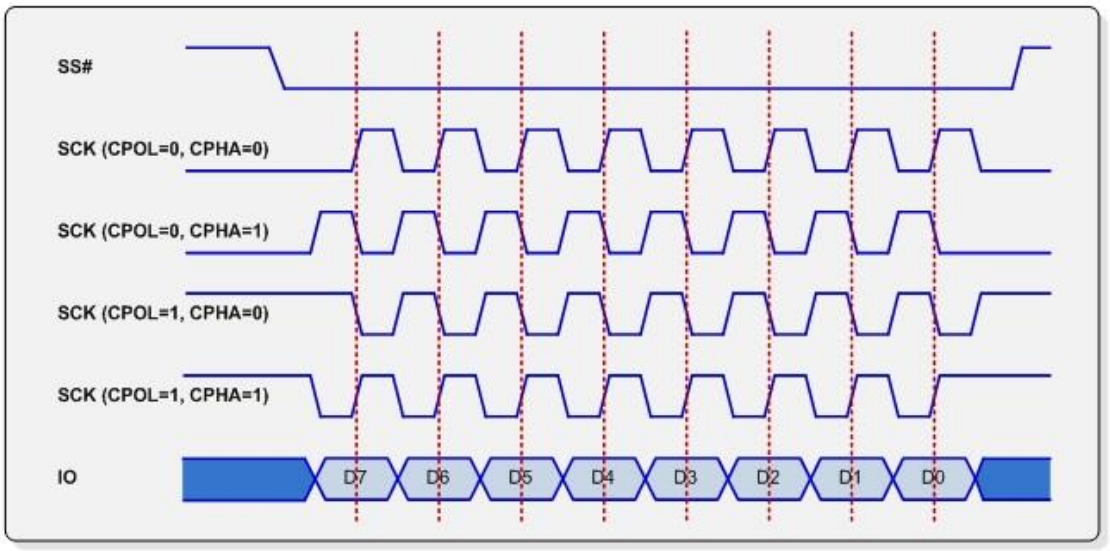
Πίνακας 2. Οι τέσσερις τρόποι λειτουργίας του ρολογιού συγχρονισμού [10]

Όταν η CPOL=0 τότε η βάση του ρολογιού είναι το “0” και η ενεργή κατάσταση είναι το “1”. Τότε:

- Αν η CPHA=0, η πληροφορία στην είσοδο λαμβάνεται κατά την ανερχόμενη ακμή του παλμού ρολογιού, από το “0” στο “1”, και η πληροφορία τοποθετείται στην έξοδο κατά την κατερχόμενη ακμή, δηλαδή όταν ο παλμός μεταβαίνει από το “1” στο “0”.
- Αν η CPHA=1, η πληροφορία στην είσοδο λαμβάνεται κατά την κατερχόμενη ακμή του παλμού ρολογιού και η πληροφορία τοποθετείται στην έξοδο κατά την ανερχόμενη ακμή.

Όταν η CPOL=1 τότε η βάση του ρολογιού είναι το “1” και η ενεργή κατάσταση είναι το “0”. Τότε:

- Αν η CPHA=0, η πληροφορία στην είσοδο λαμβάνεται κατά την κατερχόμενη ακμή του παλμού ρολογιού, και η πληροφορία τοποθετείται στην έξοδο κατά την ανερχόμενη ακμή.
- Αν η CPHA=1, η πληροφορία στην είσοδο λαμβάνεται κατά την ανερχόμενη ακμή του παλμού ρολογιού και η πληροφορία τοποθετείται στην έξοδο κατά την κατερχόμενη ακμή.



Εικόνα 21. Διάγραμμα χρονισμού του διαύλου SPI [9]

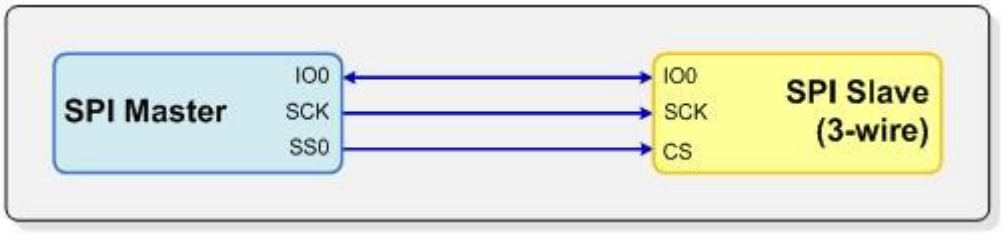
Συνοπτικά, ανεξάρτητα της CPOL, όταν η CPHA=0 τότε η πληροφορία στην είσοδο λαμβάνεται στην ακμή του πρώτου ρολογιού και η πληροφορία τοποθετείται στην έξοδο στη μετάβαση από την ενεργή στην αδρανή κατάσταση. Όταν η CPHA=1, τότε η πληροφορία στην είσοδο λαμβάνεται στην ακμή του δεύτερου ρολογιού και η πληροφορία τοποθετείται στην έξοδο στη μετάβαση από την αδρανή στην ενεργή κατάσταση.

Η πληροφορία τοποθετείται στην έξοδο πάντα κατά την αντίθετη ακμή ρολογιού από εκείνη κατά την οποία έγινε η λήψη της πληροφορίας στην είσοδο [6].

2.1.6 Εναλλακτικές διαμορφώσεις του διαύλου SPI

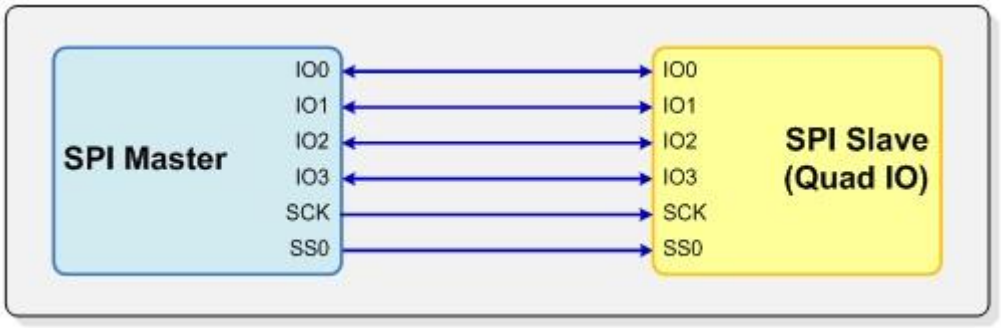
Εκτός της βασικής διαμόρφωσης του SPI με τις τέσσερις γραμμές, υπάρχει η διαμόρφωση με τις τρεις γραμμές για μειωμένο αριθμό γραμμών, ή με πολλαπλές γραμμές εισόδου-εξόδου (IO) για αυξημένες δυνατότητες στη διακίνηση των δεδομένων.

Στη διαμόρφωση με τις τρεις γραμμές, οι δύο γραμμές MOSI και MISO συνδυάζονται σε μία γραμμή εισόδου-εξόδου IO, η οποία επιτρέπει επικοινωνία εναλλάξ διπλής κατεύθυνσης (half-duplex). Το πλεονέκτημα είναι ότι απαιτείται μειωμένος αριθμός συνδέσεων, αλλά ταυτόχρονα περιορίζεται ο όγκος των δεδομένων που μπορούν να διακινηθούν μέσω του διαύλου κι έτσι αυτή η διαμόρφωση χαρακτηρίζεται από τη χαμηλή αποδοτικότητά της.



Εικόνα 22. Διαμόρφωση με τρεις γραμμές και μία slave [9]

Οι διαμορφώσεις με τις πολλαπλές γραμμές εισόδου-εξόδου προσθέτουν επιπλέον γραμμές για τη διακίνηση των δεδομένων, κερδίζοντας έτσι αυτό το πλεονέκτημα που διαθέτουν οι παράλληλες συνδέσεις, αλλά διατηρώντας ταυτόχρονα χαμηλό των αριθμό συνδέσεων. Οι γραμμές εισόδου-εξόδου IO είναι half-duplex όπως στη διαμόρφωση των τριών γραμμών. Όταν χρησιμοποιούνται διπλές γραμμές IO, η δυνατότητα διακίνησης δεδομένων είναι συγκρίσιμη με αυτή της βασικής διαμόρφωσης των τεσσάρων γραμμών, ενώ όταν χρησιμοποιούνται τετραπλές γραμμές IO, ο όγκος των δεδομένων είναι διπλάσιος από αυτόν της βασικής διαμόρφωσης [6], [9].

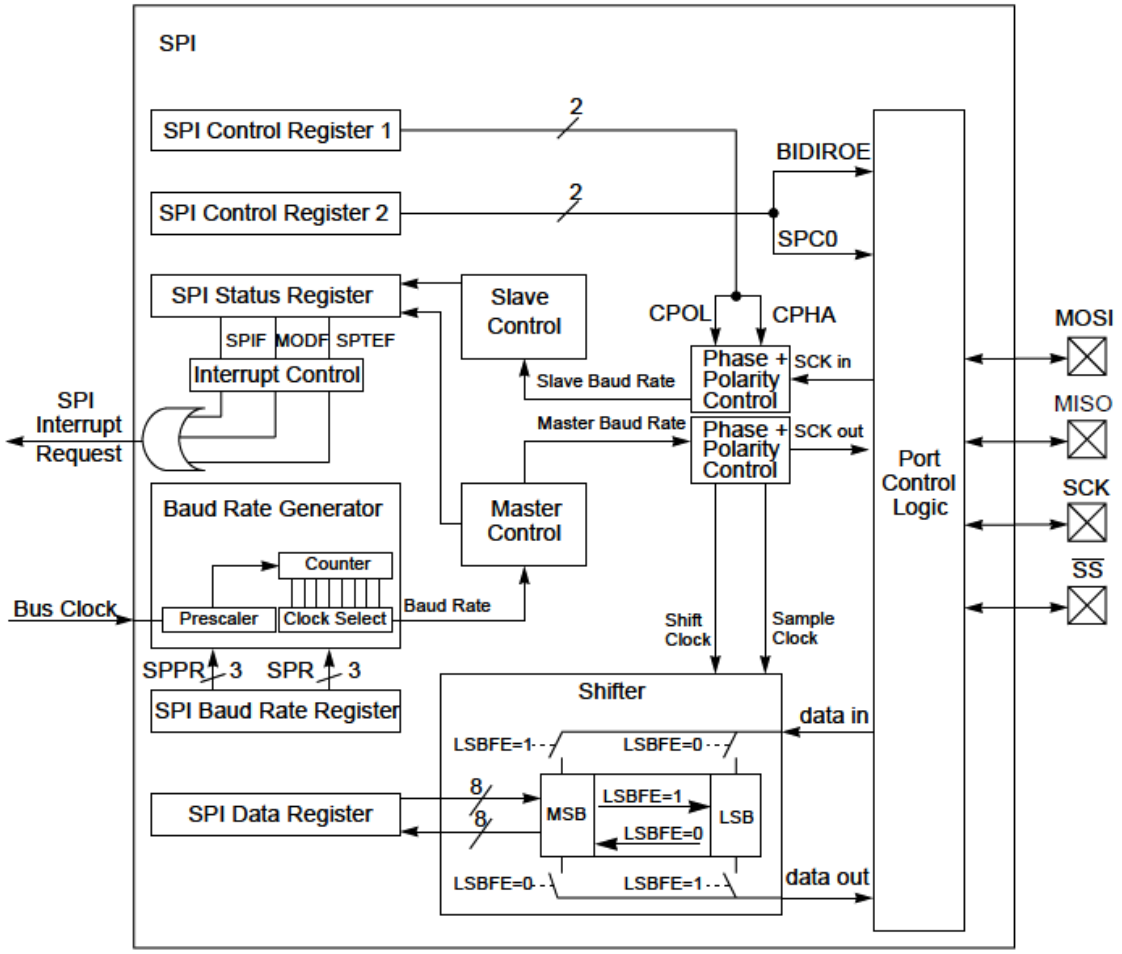


Εικόνα 23. Διαμόρφωση με 4 γραμμές I/O και μία slave [9]

2.2 Η αρχιτεκτονική του SPI

2.2.1 Γενική περιγραφή

Στο ακόλουθο σχήμα παρουσιάζεται η βασική δομή της αρχιτεκτονικής SPI. Τα βασικά μέρη απαρτίζουν οι καταχωρητές κατάστασης, ελέγχου και δεδομένων (status, control και data registers), η γεννήτρια ρυθμού μετάδοσης (baud rate generator), τα κυκλώματα ολίσθησης (shifter logic), τα κυκλώματα ελέγχου θύρας (port control logic) και τα κυκλώματα ελέγχου master/slave (master/slave control logic).



Εικόνα 24. Το δομικό διάγραμμα του υλικού SPI [12]

Οι καταχωρητές ελέγχου 1 & 2 περιέχουν τις παραμέτρους που καθορίζουν τη λειτουργία του υλικού SPI. Ο καταχωρητής κατάστασης περιέχει τις σημαίες (flags) που δείχνουν κάποιες συγκεκριμένες καταστάσεις στις οποίες περιέρχεται το συγκεκριμένο υλικό SPI. Οι σημαίες συνθέτουν τα αιτήματα διακοπών (interrupt requests) που δύναται να αποσταλούν προς μία CPU. Ο καταχωρητής δεδομένων αποτελεί καταχωρητή για τα δεδομένα εισόδου αλλά και για τα δεδομένα εξόδου. Ο καταχωρητής ρυθμού μετάδοσης (Baud Rate Register) περιέχει τα bits τα οποία θα προσδιορίσουν το διαιρέτη με τον οποίο θα διαιρεθεί το ρολόι συγχρονισμού του διαύλου (bus clock) και θα αποτελέσει το ρυθμό μετάδοσης των δεδομένων (SCLK).

Το υλικό SPI μπορεί να βρεθεί σε τρεις καταστάσεις λειτουργίας, να εκτελεί (Run), να περιμένει (Wait) και να είναι αδρανές (Stop). Η βασική κατάσταση λειτουργίας είναι η κατάσταση Run.

Η κατάσταση λειτουργίας Wait ελέγχεται από το bit SPISWAI που περιέχεται στο καταχωρητή 2 (SPICR2). Όταν το bit είναι “0” τότε βρίσκεται σε λειτουργία Run. Όταν το bit γίνει “1”, τότε η λειτουργία Wait ενεργοποιείται και το υλικό SPI οδηγείται σε κατάσταση χαμηλής κατανάλωσης ενέργειας και η παραγωγή του ρολογιού συγχρονισμού σταματά.

Εάν η συσκευή SPI που λειτουργεί σε Wait είναι συσκευή master, τότε κάθε μετάδοση δεδομένων που βρίσκεται σε εξέλιξη διακόπτεται, ενώ συνεχίζεται πάλι όταν επιστρέψει σε λειτουργία Run. Εάν η συσκευή SPI είναι συσκευή slave, τότε η συσκευή SPI συνεχίζει να λειτουργεί έτσι όπως λειτουργούσε πριν τεθεί σε κατάσταση Wait. Δεν αλλάζει καταστάσεις σε σημαίες (flags), ούτε μπορεί να αντιγραφούν δεδομένα στον καταχωρητή δεδομένων κατά τη λήψη, με αποτέλεσμα τα δεδομένα στην είσοδο της slave να χάνονται όταν εισέρχεται σε λειτουργία Wait. Εάν η συσκευή slave μεταδίδει δεδομένα στην έξοδο, τότε εξακολουθεί να στέλνει το ίδιο byte του καταχωρητή δεδομένων SPIDR που έστελνε όταν μετέβη σε λειτουργία Wait από Run, ή εάν μετέδιδε το τελευταίο byte που έλαβε από τη master, τότε θα συνέχιζε τη μετάδοση καθενός από τα προηγούμενα bytes που είχε λάβει από τη master. Η συσκευή slave σε λειτουργία Wait εξακολουθεί να βρίσκεται σε συγχρονισμό με τη master.

Η λειτουργία Stop έχει τα ίδια χαρακτηριστικά με τη λειτουργία Wait. Η διαφορά είναι ότι αυτή η λειτουργία δεν ενεργοποιείται από το bit SPISWAI, αλλά όταν το ρολόι συγχρονισμού της συσκευής απενεργοποιηθεί, εάν δηλαδή κρατηθεί σε κατάσταση HIGH ή LOW [12].

2.2.2 Οι καταχωρητές του SPI

Κάθε καταχωρητής έχει τη διεύθυνσή του, η οποία προκύπτει από το άθροισμα της διεύθυνσης βάσης με τη μετατόπιση διεύθυνσης που αντιστοιχεί σε κάθε καταχωρητή. Η διεύθυνση βάσης είναι η διεύθυνση του SoC (System-on-a-chip) στο οποίο βρίσκεται ο καταχωρητής. Στον πίνακα 3 φαίνεται ο χάρτης μνήμης και οι ονομασίες των καταχωρητών SPI. Οι αναγνώσεις από κατειλημμένα bits επιστρέφουν μηδέν, ενώ οι εγγραφές σε κατειλημμένα bits δεν έχουν καμία επίδραση [12].

Address	Use	Access
$___0	SPI Control Register 1 (SPICR1)	Read / Write
$___1	SPI Control Register 2 (SPICR2)	Read / Write [1]
$___2	SPI Baud Rate Register (SPIBR)	Read / Write [1]
$___3	SPI Status Register (SPISR)	Read [2]
$___4	Reserved	— [2 3]
$___5	SPI Data Register (SPIDR)	Read / Write
$___6	Reserved	— [2 3]
$___7	Reserved	— [2 3]

Πίνακας 3. Ο χάρτης μνήμης των καταχωρητών [12]

2.2.2.1 SPI Control Register 1 (SPICR1)

Στον καταχωρητή αυτό εγγράφονται οι τιμές παραμέτρων που αφορούν τη λειτουργία του υλικού SPI.

Register Address: $___0

	Bit 7	6	5	4	3	2	1	Bit 0
R W	SPIE	SPE	SPTIE	MSTR	CPOL	CPHA	SSOE	LSBFE
Reset:	0	0	0	0	0	1	0	0

Εικόνα 25. Ο καταχωρητής ελέγχου 1 (SPICR1) [12]

Αναλυτικά τα bits του καταχωρητή είναι:

- Το SPI Interrupt Enable bit – SPIE
  Ενεργοποιεί τις αιτήσεις διακοπών εάν οι σημαίες των καταστάσεων SPIF ή MODF είναι ενεργές. Οι διακοπές ενεργοποιούνται όταν το bit είναι “1”.
- Το SPI System Enable bit – SPE
  Ενεργοποιεί το σύστημα SPI και αντιστοιχεί τους ακροδέκτες των θυρών SPI στις λειτουργίες του SPI συστήματος. Το σύστημα είναι ενεργοποιημένο όταν το bit είναι “1”, ενώ όταν είναι “0” το σύστημα μεταβαίνει σε κατάσταση αδρανείας, χαμηλής κατανάλωσης ενέργειας, και τα bits κατάστασης του καταχωρητή κατάστασης SPISR επαναφέρονται στην προκαθορισμένη τους κατάσταση.
- Το SPI Transmit Interrupt Enable bit – SPTIE
  Όταν το bit είναι “1” και η σημαία SPTEF είναι ενεργή, τότε ενεργοποιούνται οι αιτήσεις διακοπών.
- Το SPI Master/Slave Mode Select bit – MSTR
  Όταν το bit είναι “1” τότε το SPI λειτουργεί ως master, ενώ με “0” λειτουργεί ως slave.
- Το SPI Clock Polarity bit – CPOL
  Με αυτό το bit επιλέγεται να είναι ανεστραμμένοι ή όχι οι παλμοί συγχρονισμού SCLK του SPI. Όταν το bit είναι “1”, τότε το ρολόι συγχρονισμού SCLK έχει ενεργή κατάσταση τη LOW και ανενεργή τη HIGH. Όταν το bit είναι “0” συμβαίνει το αντίστροφο.
- Το SPI Clock Phase bit – CPHA
  Με αυτό το bit επιλέγεται η φάση του ρολογιού SCLK. Όταν το bit είναι “0”, τότε η δειγματοληψία των δεδομένων πραγματοποιείται σε περιττές ακμές (1,3,5,..), ενώ όταν το bit είναι “1” η δειγματοληψία των δεδομένων πραγματοποιείται σε άρτιες ακμές (2,4,6,..).
- Το Slave Select Output Enable bit – SSOE
  Ο ακροδέκτης $\overline{SS}$ λειτουργεί ως έξοδος μόνο σε λειτουργία Master. Με την προϋπόθεση ότι η λειτουργία MODFEN είναι ενεργή, το bit SSOE καθορίζει εάν το $\overline{SS}$ θα αποτελεί είσοδο ή έξοδο στη συσκευή master, όπως δείχνει ο πίνακας 4.

MODFEN	SSOE	Master Mode	Slave Mode
0	0	$\overline{SS}$ not used by SPI	$\overline{SS}$ input
0	1	$\overline{SS}$ not used by SPI	$\overline{SS}$ input
1	0	$\overline{SS}$ input with MODF feature	$\overline{SS}$ input
1	1	$\overline{SS}$ is slave select output	$\overline{SS}$ input

Πίνακας 4. Επιλογή του ακροδέκτη $\overline{SS}$ για είσοδο ή έξοδο [12]

- Το LSB-First Enable bit – LSBFE
  Στον καταχωρητή δεδομένων το MSB είναι πάντα το bit 7. Το bit LSBFE δεν αλλάζει τη θέση των MSB και LSB στον καταχωρητή δεδομένων, αλλά το ποιο από τα δύο θα μεταφερθεί αρχικά. Έτσι, όταν το LSBFE είναι “1” μεταφέρεται πρώτα το LSB, ενώ όταν είναι “0” μεταφέρεται πρώτα το MSB [12].

2.2.2.2 SPI Control Register 2 (SPICR2)

Όπως και στον καταχωρητή ελέγχου 1, στον καταχωρητή αυτό εγγράφονται οι τιμές παραμέτρων που αφορούν τη λειτουργία του υλικού SPI. Από τα 8 bit του καταχωρητή, τα 4 είναι αυτά που χρησιμοποιούνται.

Register Address: $___1

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	0	0	MODFEN	BIDIROE	0	SPISWAI	SPC0
W								
Reset:	0	0	0	0	0	0	0	0

= Reserved

Εικόνα 26. Ο καταχωρητής ελέγχου 2 (SPICR2) [12]

Αναλυτικά τα bits του καταχωρητή που χρησιμοποιούνται είναι:

- Το Mode Fault Enable bit – MODFEN
  Όταν το bit είναι “1”, τότε επιτρέπει την ανίχνευση σφάλματος MODF στον ακροδέκτη $\overline{SS}$ τη στιγμή που λειτουργεί ως είσοδος σε λειτουργία Master. Ο τρόπος λειτουργίας του ακροδέκτη $\overline{SS}$ επιλέγεται σε συνδυασμό με το bit SSOE του καταχωρητή SPICR1.
- Το Output enable in the Bidirectional mode of operation bit – BIDIROE
  Με την προϋπόθεση ότι το SPI βρίσκεται σε εναλλάξ αμφίδρομη λειτουργία (το bit SPC0 είναι ενεργό), το bit BIDIROE ελέγχει την ενδιάμεση μνήμη (buffer) των ακροδεκτών MOSI και MISO. Σε λειτουργία master το bit BIDIROE ελέγχει την ενδιάμεση μνήμη της MOSI, ενώ σε λειτουργία slave ελέγχει την ενδιάμεση μνήμη της MISO. Ο έλεγχος της ενδιάμεσης μνήμης ενεργοποιείται όταν το bit BIDIROE είναι “1” και απενεργοποιείται με “0”.
- Το SPI Stop in Wait Mode bit – SPISWAI
  Το bit αυτό χρησιμοποιείται για την μείωση στην κατανάλωση ενέργειας όταν το SPI βρίσκεται σε κατάσταση Wait. Έτσι, όταν το bit είναι “1” σταματά η παραγωγή του ρολογιού, ενώ όταν είναι “0” η παραγωγή του ρολογιού στην κατάσταση Wait συνεχίζεται κανονικά.
- Το Serial Pin Control bit 0 – SPC0
  Με αυτό το bit επιτρέπονται εναλλάξ αμφίδρομες διαμορφώσεις των ακροδεκτών, όπως φαίνεται στον πίνακα 5 [12].

Pin Mode	SPC0	BIDIROE	MISO	MOSI
Master Mode of Operation				
Normal	0	X	Master In	Master Out
Bidirectional	1	0	MISO not used by SPI	Master In
		1		Master I/O
Slave Mode of Operation				
Normal	0	X	Slave Out	SlaveIn
Bidirectional	1	0	Slave In	MOSI not used by SPI
		1	Slave I/O	

Πίνακας 5. Οι διαμορφώσεις των ακροδεκτών σε εναλλάξ αμφίδρομη επικοινωνία [12]

2.2.2.3 SPI Baud Rate Register (SPIBR)

Τα bits του καταχωρητή ρυθμού μετάδοσης δεδομένων προσδιορίζουν τον διαιρέτη με τον οποίο θα διαιρεθεί το ρολόι συγχρονισμού του διαύλου (bus clock).

Register Address: $___2

	Bit 7	6	5	4	3	2	1	Bit 0
R	0	SPPR2	SPPR1	SPPR0	0	SPR2	SPR1	SPR0
W								
Reset:	0	0	0	0	0	0	0	0

= Reserved

Εικόνα 27. Ο καταχωρητής ρυθμού μετάδοσης δεδομένων (SPIBR) [12]

Περιέχει δύο ομάδες bits. Τα bits SPPR2-SPPR0, που είναι τα bits προεπιλογής του ρυθμού μετάδοσης και τα bits SPR2-SPR0, που είναι τα bits επιλογής του ρυθμού μετάδοσης. Ο διαιρέτης του ρολογιού συγχρονισμού προκύπτει από την ακόλουθη εξίσωση:

$$BaudRateDivisor = (SPPR+1) \cdot 2^{(SPR+1)}$$

ενώ ο ρυθμός μετάδοσης δεδομένων προκύπτει από την εξίσωση:

$$Baud\ Rate = BusClock / BaudRateDivisor$$

Παραδείγματα επιλογής του ρυθμού μετάδοσης δεδομένων παρουσιάζονται στον πίνακα 6 [12].

SPPR2	SPPR1	SPPR0	SPR2	SPR1	SPR0	BaudRate Divisor	Baud Rate
0	0	0	0	0	0	2	12.5 MHz
0	0	0	0	0	1	4	6.25 MHz
0	0	0	0	1	0	8	3.125 MHz
0	0	0	0	1	1	16	1.5625 MHz
0	0	0	1	0	0	32	781.25 kHz
0	0	0	1	0	1	64	390.63 kHz
0	0	0	1	1	0	128	195.31 kHz
0	0	0	1	1	1	256	97.66 kHz
0	0	1	0	0	0	4	6.25 MHz
0	0	1	0	0	1	8	3.125 MHz
0	0	1	0	1	0	16	1.5625 MHz
0	0	1	0	1	1	32	781.25 kHz
0	0	1	1	0	0	64	390.63 kHz
0	0	1	1	0	1	128	195.31 kHz
0	0	1	1	1	0	256	97.66 kHz
0	0	1	1	1	1	512	48.83 kHz
0	1	0	0	0	0	6	4.16667 MHz
0	1	0	0	0	1	12	2.08333 MHz
0	1	0	0	1	0	24	1.04167 MHz
0	1	0	0	1	1	48	520.83 kHz
0	1	0	1	0	0	96	260.42 kHz
0	1	0	1	0	1	192	130.21 kHz
0	1	0	1	1	0	384	65.10 kHz
0	1	0	1	1	1	768	32.55 kHz
0	1	1	0	0	0	8	3.125 MHz
0	1	1	0	0	1	16	1.5625 MHz
0	1	1	0	1	0	32	781.25 kHz
0	1	1	0	1	1	64	390.63 kHz
0	1	1	1	0	0	128	195.31 kHz
0	1	1	1	0	1	256	97.66 kHz
0	1	1	1	1	0	512	48.83 kHz
0	1	1	1	1	1	1024	24.41 kHz

Πίνακας 6. Παράδειγμα επιλογής ρυθμού μετάδοσης δεδομένων [12]

2.2.2.4 SPI Status Register (SPISR)

Ο καταχωρητής κατάστασης περιέχει τις τιμές των σημαιών (flags), οι οποίες ενεργοποιούνται όταν το σύστημα SPI έχει επέλθει σε συγκεκριμένες καταστάσεις. Οι σημαίες μπορούν να χρησιμοποιηθούν ως αιτήματα διακοπών (interrupt requests) προς κάποια CPU.

Register Address: $___3

	Bit 7	6	5	4	3	2	1	Bit 0
R	SPIF	0	SPTEF	MODF	0	0	0	0
W								
Reset:	0	0	1	0	0	0	0	0

= Reserved

Εικόνα 28. Ο καταχωρητής κατάστασης (SPISR) [12]

Τρία είναι τα bits που καταχωρούν τις τιμές των αντίστοιχων σημαιών:

- Το SPIF Interrupt Flag bit – SPIF
  Το bit αυτό γίνεται “1” όταν ένα byte που λήφθηκε έχει μεταφερθεί στον καταχωρητή δεδομένων, ενώ γίνεται πάλι “0” όταν υπάρξει πρόσβαση ανάγνωσης στον καταχωρητή δεδομένων.
- Το SPI Transmit Empty Interrupt Flag bit – SPTEF
  Όταν ο καταχωρητής δεδομένων είναι κενός, τότε αυτό το bit έχει την τιμή “1”. Για να γίνει “0” πρέπει να γίνει ανάγνωση του SPTEF=“1” με ταυτόχρονη εγγραφή δεδομένων στον καταχωρητή δεδομένων SPIDR. Κάθε εγγραφή στον καταχωρητή δεδομένων αγνοείται εάν δεν αναγνωστεί το SPTEF=“1”.
- Το Mode Fault Flag bit – MODF
  Όταν το SPI λειτουργεί ως master ο ακροδέκτης $\overline{SS}$ λειτουργεί ως είσοδος, με σκοπό την ανίχνευση εκπομπής από κάποια άλλη συσκευή master. Όταν επιχειρήσει να μεταδώσει κάποια άλλη master, τότε ο ακροδέκτης $\overline{SS}$ γίνεται LOW και το MODF γίνεται “1” επισημαίνοντας σφάλμα. Προϋπόθεση αποτελεί ότι το bit MODFEN του καταχωρητή SPICR2 είναι ενεργό [12].

2.2.2.5 SPI Data Register (SPIDR)

Register Address: $___5

	Bit 7	6	5	4	3	2	1	Bit 0
R W	Bit 7	6	5	4	3	2	2	Bit 0
Reset:	0	0	0	0	0	0	0	0

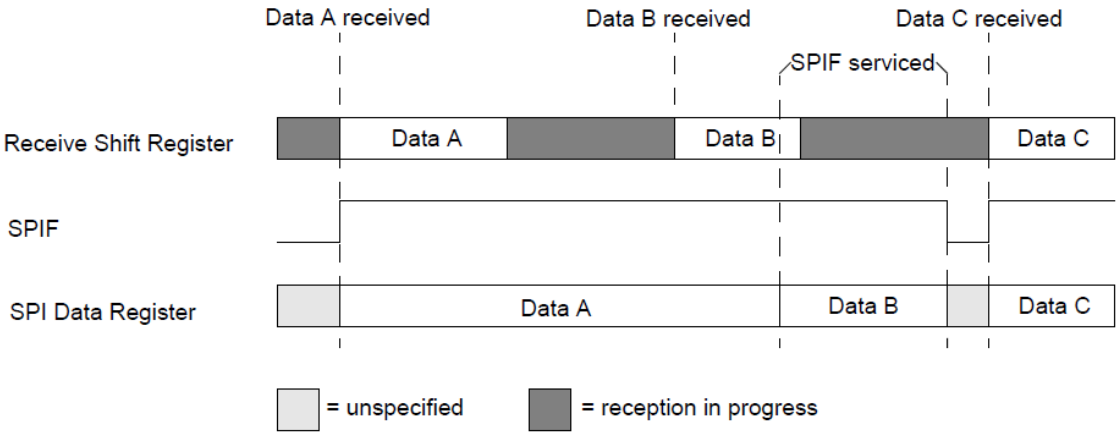
Εικόνα 29. Ο καταχωρητής δεδομένων (SPIDR) [12]

Ο καταχωρητής δεδομένων είναι καταχωρητής εισόδου και εξόδου σε μία συσκευή SPI. Τα δεδομένα που εισέρχονται τοποθετούνται στη σειρά για να μεταδοθούν από την έξοδο. Μετά τη μετάδοση ενός byte ξεκινά η μετάδοση του επόμενου στη σειρά byte. Μετά τη μετάδοση ενός byte ο καταχωρητής αδειάζει, το SPTEF γίνεται “1” και ο καταχωρητής είναι έτοιμος να δεχθεί από τον καταχωρητή ολίσθησης (shift register) το επόμενο byte.

Όταν το SPIF είναι “0” και ο καταχωρητής δεδομένων ολοκληρώσει την εγγραφή ενός νέου byte, τότε το SPIF γίνεται “1”. Το SPIF εξυπηρετείται όταν γίνει ανάγνωση του καταχωρητή κατάστασης SPISR και υπάρξει πρόσβαση για ανάγνωση στον καταχωρητή δεδομένων. Τότε το SPIF γίνεται “0”.

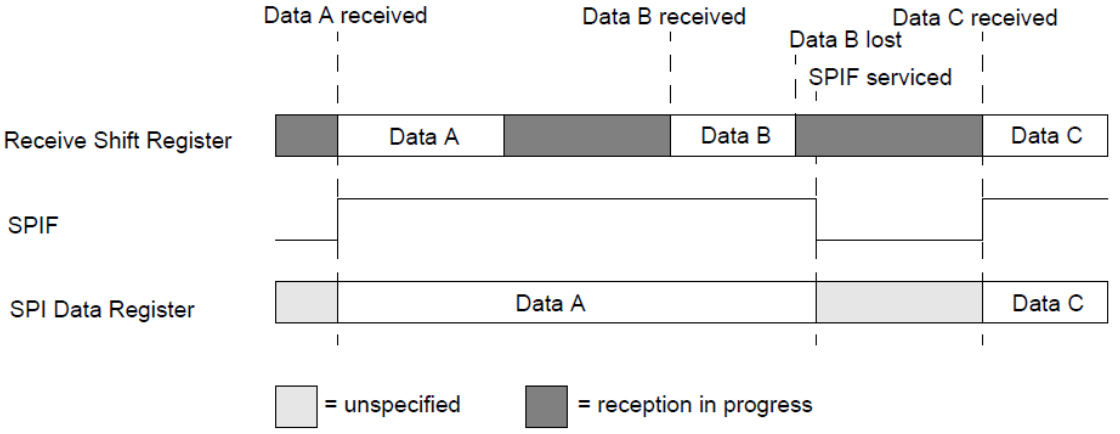
Εάν το SPIF δεν έχει εξυπηρετηθεί και ο shift register έχει λάβει δεύτερο byte, τότε το δεύτερο byte παραμένει στο shift register μέχρι να εξυπηρετηθεί το SPIF και να ξεκινήσει η μετάδοση του byte του καταχωρητή δεδομένων.

Εάν πρόκειται να ληφθεί ένα τρίτο byte, με το SPIF να είναι “1”, δηλαδή να μην έχει εξυπηρετηθεί, τότε υπάρχει κίνδυνος το δεύτερο byte να χαθεί. Εάν το SPIF εξυπηρετηθεί πριν ξεκινήσει η λήψη του τρίτου byte, τότε το δεύτερο byte θα έχει προλάβει να μεταφερθεί στον καταχωρητή δεδομένων, ενώ το πρώτο θα έχει ήδη μεταδοθεί. Το τρίτο byte θα έχει βρει άδειο τον καταχωρητή ολίσθησης [12].



Εικόνα 30. Λήψη δεδομένων με έγκαιρη εξυπηρέτηση του SPIF [11]

Εάν όμως δεν έχει προλάβει να εξυπηρετηθεί το SPIF, τότε η λήψη του τρίτου byte θα ξεκινήσει όταν το δεύτερο byte βρίσκεται ακόμα στον καταχωρητή ολίσθησης. Τότε το δεύτερο byte θα χαθεί.



Εικόνα 31. Λήψη δεδομένων με αργοπορημένη εξυπηρέτηση του SPIF [12]

2.3 Περιγραφή λειτουργίας του SPI

2.3.1 Η λειτουργία Master

Η λειτουργία Master επιλέγεται με την ενεργοποίηση του bit MSTR. Για να ξεκινήσει η μετάδοση πρώτα γράφεται το byte στον καταχωρητή δεδομένων και μεταφέρεται στον καταχωρητή ολίσθησης εάν αυτός είναι κενός. Από εκεί το byte ολισθαίνει για να μεταδοθεί από τον ακροδέκτη MOSI προς τη συσκευή slave σύμφωνα με τον ρυθμό μετάδοσης του SCLK.

Ο ακροδέκτης $\overline{SS}$ λειτουργεί ως έξοδος επιλογής της συσκευής slave όταν τα bits MODFEN και SSOE είναι “1”. Κατά τη μετάδοση ο $\overline{SS}$ πηγαίνει σε κατάσταση LOW, διαφορετικά είναι σε κατάσταση HIGH. Όταν το bit MODFEN είναι “1” και το SSOE είναι “0”, τότε ο ακροδέκτης $\overline{SS}$ λειτουργεί ως είσοδος ανίχνευσης σφάλματος. Εάν προσπαθήσει να μεταδώσει και μία άλλη συσκευή master, τότε η είσοδος $\overline{SS}$ θα γίνει LOW και θα διαγνωσθεί σφάλμα. Τότε, το SPI μεταβαίνει αμέσως σε λειτουργία slave, διακόπτεται κάθε μετάδοση που ίσως να ήταν προηγουμένως σε εξέλιξη, το bit MSTR γίνεται “0” και ενεργοποιείται η σημαία MODF στον καταχωρητή κατάστασης SPISR [11].

2.3.2 Η λειτουργία Slave

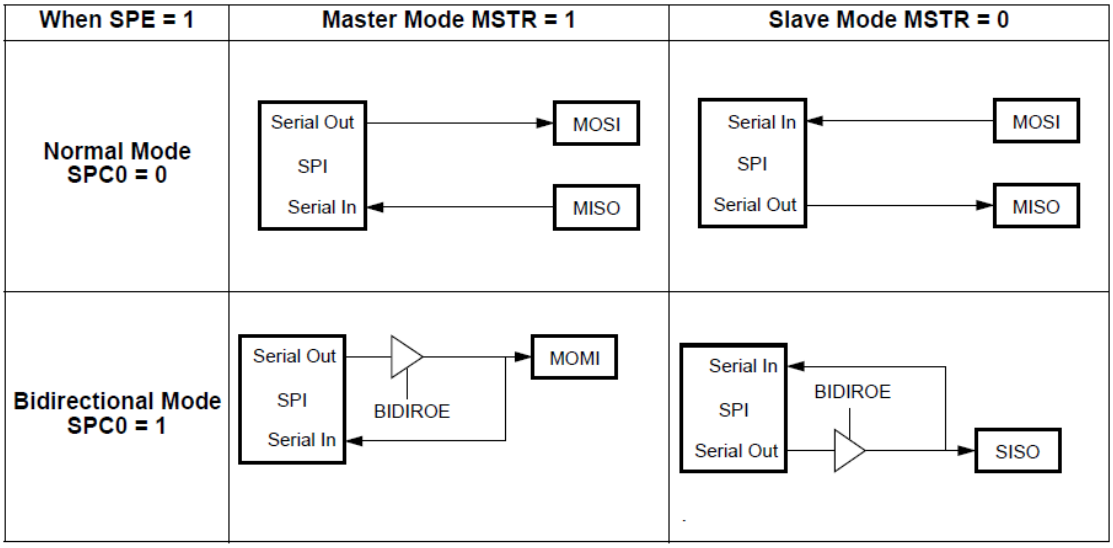
Μία συσκευή master μπορεί να λαμβάνει δεδομένα μόνο από μία συσκευή slave, αλλά μπορεί να στέλνει δεδομένα σε πολλές. Στη λειτουργία slave ο ακροδέκτης $\overline{SS}$ αποτελεί την είσοδο επιλογής της συσκευής slave. Όταν ξεκινήσει μία μετάδοση, ο ακροδέκτης $\overline{SS}$ της συσκευής slave που επιλέγεται πρέπει να γίνει LOW και να παραμείνει έτσι μέχρι να ολοκληρωθεί.

Εάν ο ακροδέκτης $\overline{SS}$ είναι HIGH, τότε η συγκεκριμένη συσκευή slave δεν έχει επιλεγεί, ο ακροδέκτης εξόδου παραμένει σε υψηλή εμπέδηση, η είσοδος SCLK αγνοείται και δεν πραγματοποιείται καμία εσωτερική ολίσθηση στον καταχωρητή ολίσθησης. Όταν ο $\overline{SS}$ γίνει HIGH, τότε το πρώτο bit που βρίσκεται στον καταχωρητή δεδομένων οδηγείται προς την έξοδο [12].

2.3.3 Η λειτουργία σε εναλλάξ αμφίδρομη επικοινωνία

Στην εναλλάξ αμφίδρομη επικοινωνία (Bidirectional Mode), οι δύο γραμμές MOSI και MISO συνδυάζονται σε μία γραμμή εισόδου-εξόδου IO, η οποία επιτρέπει επικοινωνία εναλλακτικά προς τις δύο κατευθύνσεις (half-duplex).

Το bidirectional mode επιλέγεται από το bit SPC0 του καταχωρητή SPICR2 όταν γίνει “1”. Τότε, ο ακροδέκτης MOSI γίνεται για τη συσκευή master είσοδος και ταυτόχρονα έξοδος (I/O) για τα σειριακά δεδομένα, και παίρνει το όνομα MOMI (Master Output, Master Input). Παρόμοια, για τη συσκευή slave, ο ακροδέκτης MISO γίνεται SISO (Serial Input, Serial Output). Ο ακροδέκτης MISO στη master και ο MOSI στη slave δεν χρησιμοποιούνται στο bidirectional mode.



Εικόνα 32. Το SPI σε κανονική λειτουργία και σε εναλλάξ αμφίδρομη επικοινωνία [12]

Η φορά των ακροδεκτών MOMI και SISO διαμορφώνεται σύμφωνα με το bit BIDIROE του καταχωρητή SPICR2, έτσι ώστε να μπορούν να λειτουργούν άλλοτε ως είσοδοι και άλλοτε ως έξοδοι των συσκευών master και slave στο bidirectional mode [12].

ΚΕΦΑΛΑΙΟ 3 - Σύγκριση των δύο πρωτοκόλλων επικοινωνίας

Τα δύο πρωτόκολλα I²C και SPI αναπτύχθηκαν με σκοπό την επικοινωνία των μικροεπεξεργαστών με τις περιφερειακές τους συσκευές και κυρίως σε ενσωματωμένα συστήματα (embedded systems), τα οποία εντοπίζονται σε οικιακές συσκευές, φωτογραφικές μηχανές, κινητά τηλέφωνα κ.α. Αφορούν κυρίως επικοινωνίες που περιορίζονται στην περιοχή του τυπωμένου κυκλώματος, καθότι και τα δύο πρωτόκολλα λόγω της απλότητάς τους είναι ευάλωτα σε αλλοιώσεις της πληροφορίας που διακινούν.

Αναφορικά με την τοπολογία των ηλεκτρονικών κυκλωμάτων και τα εξαρτήματα που απαιτούνται για την πραγματοποίηση επικοινωνίας με τα πρωτόκολλα I²C και SPI, μπορούμε να παρατηρήσουμε τα εξής:

- Ο δίαυλος I²C απαιτεί μόνο 2 καλώδια ανεξάρτητα από τον αριθμό των συσκευών που συνδέονται στο δίαυλο. Για το δίαυλο SPI απαιτούνται τουλάχιστον 3 καλώδια – λειτουργία σε bidirectional mode, ή διάταξη daisy chain [11].
- Ο δίαυλος I²C μπορεί να χρησιμοποιηθεί και σε συνδέσεις εκτός τυπωμένου κυκλώματος, σε κοντινές αποστάσεις, λόγω του κάπως πιο ανθεκτικού πρωτοκόλλου απέναντι στις αλλοιώσεις της πληροφορίας.
- Λόγω επίσης της ιδιότητας του I²C να επιλέγεται η συσκευή slave μέσω διεύθυνσης αλλά και να επιτρέπονται περισσότερες από μία συσκευές master (ιδιότητα multi-master), δίνεται η δυνατότητα να προστίθεται ή να αφαιρείται μία συσκευή από το δίαυλο χωρίς να χρειάζεται παρέμβαση στην τοπολογία του ηλεκτρονικού κυκλώματος (hot swapping). Αυτή η δυνατότητα αποκλείεται τελείως από το δίαυλο SPI.
- Ο δίαυλος SPI χρησιμοποιεί εξαρτήματα που καταναλώνουν λιγότερη ενέργεια λόγω του ότι το πρωτόκολλο είναι πιο απλό και απαιτείται λιγότερη επεξεργασία των σημάτων που μεταδίδονται.

Αναφορικά με την ευελιξία κατά τη σχεδίαση διατάξεων με δίαυλο I²C και με δίαυλο SPI, και τις δυνατότητες των πρωτοκόλλων μπορούμε να παρατηρήσουμε τα εξής:

- Ο δίαυλος SPI υποστηρίζει πλήρως αμφίδρομες επικοινωνίες (full duplex), ενώ ο I²C εναλλάξ αμφίδρομες (half duplex). Στην πραγματικότητα όμως τις περισσότερες φορές οι επικοινωνίες είναι εναλλάξ αμφίδρομες. Εάν σε ένα δίαυλο SPI μία συσκευή master θέλει να διαβάσει από μία συσκευή slave, τότε στέλνει εικονικά bytes (dummy bytes) για να ενεργοποιεί το ρολόι έτσι ώστε να στέλνει δεδομένα η slave. Παρόμοια, όταν η συσκευή master μεταδίδει δεδομένα, τότε σπανίως η slave αναφέρει πίσω προς τη master [13].
- Ο δίαυλος I²C παρέχει ταχύτητες μέχρι 3,4Mbps στο High Speed Mode, ενώ ο SPI δεν έχει περιορισμό. Τυπικά ξεπερνά τα 10 Mbps και μπορεί να φτάσει τα 100 Mbps [14].
- Στο δίαυλο I²C υπάρχει διασφάλιση των δεδομένων με το bit επιβεβαίωσης, ενώ στον SPI δεν υπάρχει αυτή η δυνατότητα.
- Στο δίαυλο SPI η συσκευή slave δεν μπορεί να μειώσει το ρυθμό μετάδοσης των δεδομένων, ενώ στο δίαυλο I²C υπάρχει αυτή η δυνατότητα.
- Το πρωτόκολλο I²C έχει τη δυνατότητα της διαιτησίας όταν θέλουν να επικοινωνήσουν περισσότερες από μία συσκευές master, ενώ επίσης μία συσκευή master μπορεί να γίνει slave και αντίστροφα. Έτσι, ο δίαυλος I²C έχει τη δυνατότητα να συνδέονται σε αυτόν περισσότερες της μίας συσκευής master (δυνατότητα multi-master), ενώ ο δίαυλος SPI επιτρέπει μόνο μία master [15].

- Λόγω της δυνατότητας της διαιτησίας μεταξύ των master και της επιλογής της συσκευής slave μέσω διεύθυνσης, το πρωτόκολλο I²C παρέχει τη δυνατότητα να συνδέεται ή να αποσυνδέεται μία συσκευή I²C χωρίς να απαιτείται τροποποίηση στην τοπολογία του ηλεκτρονικού κυκλώματος (hot swapping).

Συμπερασματικά, ο δίαυλος I²C προσφέρεται για εφαρμογές, στις οποίες:

- Απαιτούνται λιγότερες καλωδιώσεις
- Οι συσκευές επικοινωνούν σε χαμηλές ταχύτητες
- Θέλουμε να συνδέουμε και να αποσυνδέουμε συσκευές χωρίς να παρεμβαίνουμε στο ηλεκτρονικό κύκλωμα
- Υπάρχουν περισσότερες από μία συσκευές master
- Θέλουμε να επικοινωνήσουν μεταξύ τους συσκευές που δεν βρίσκονται στο ίδιο ή σε κοντινά τυπωμένα κυκλώματα

Το δίαυλο SPI θα τον προτιμήσουμε για εφαρμογές στις οποίες:

- Θέλουμε αμφίδρομη επικοινωνία (full duplex)
- Θέλουμε να έχουμε όσο το δυνατό λιγότερη κατανάλωση στην ενέργεια που παρέχουμε
- Χρειαζόμαστε ταχύτερες μεταδόσεις των δεδομένων μεταξύ των συσκευών ή όταν υπάρχει μεγάλος όγκος δεδομένων που πρέπει να μεταδίδονται

ΚΕΦΑΛΑΙΟ 4 - Συνοπτική παρουσίαση της ηλεκτρονικής πλατφόρμας προτυποποίησης Arduino

4.1 Εισαγωγικά στοιχεία

Η πλατφόρμα προτυποποίησης ανοικτού κώδικα Arduino έχει ως σκοπό να παρέχει υλικό και λογισμικό για την κατασκευή πρωτοτύπων, προς τον καθένα που θέλει να δημιουργήσει διαδραστικά αντικείμενα ή περιβάλλοντα και να πειραματιστεί με ηλεκτρονικές κατασκευές που απαιτούν υπολογιστική ισχύ και αυτοματισμό. Δεν απαιτείται να έχει κάποιος εξειδικευμένες γνώσεις στα ηλεκτρονικά κυκλώματα και στον προγραμματισμό. Ακολουθώντας απλά τα βήματα των οδηγιών στις συσκευασίες για αρχαρίους (starter kits), οποιοσδήποτε μπορεί να ενταχθεί στον κόσμο του Arduino και να αποκτά σταδιακά ικανότητες και την ευελιξία για να αυτοσχεδιάζει και να πραγματοποιεί τις ιδέες του. Απευθύνεται στο μαθητή, το φοιτητή, τον καθηγητή, τον ερευνητή, αλλά και στον καλλιτέχνη και το χομπίστα, που δεν θέλει να εντρυφήσει σε έννοιες και πολύπλοκους κανόνες σχεδίασης πάνω στα ηλεκτρονικά κυκλώματα και σε γλώσσες προγραμματισμού [16], [17].

Η ονομασία Arduino προήλθε από το βασιλιά Arduin της Ιβρέα, κωμόπολη στα περίχωρα του Τορίνο. Ο συνιδρυτής της πλατφόρμας Arduino, Massimo Banzi, σύχναζε σε ένα μπαρ της Ιβρέα, το “Bar di Re Arduino”, του οποίο το όνομα έδωσε στην πλακέτα του για να το τιμήσει.

Το έτος 2002, ο Massimo Banzi, αρχιτέκτονας λογισμικού (software architect), ως καθηγητής στο Ινστιτούτο Διαδραστικής Σχεδίασης της Ιβρέα (IDII: Interaction Design Institute Ivrea), είχε σκοπό την προώθηση πρωτοποριακών μεθόδων στη διαδραστική σχεδίαση, δηλαδή στον φυσικό προγραμματισμό (physical computing). Ως εργαλείο τότε υπήρχε ο μικροεπεξεργαστής Stamp με γλώσσα προγραμματισμού την Basic, η οποία πλατφόρμα είχε τα εξής δύο βασικά μειονεκτήματα: χαμηλή επεξεργαστική ισχύ και υψηλό κόστος αγοράς για τους μαθητές του.

Εν τω μεταξύ, στο MIT είχε αναπτυχθεί η φιλική προς το σχεδιαστή γλώσσα προγραμματισμού Processing για τη δημιουργία γραφικών, κινητών εικόνων (animation) και διαδραστικών περιβάλλοντων. Η ιδέα ήταν να χρησιμοποιηθεί το περιβάλλον ανάπτυξης της Processing για την κωδικοποίηση των εντολών κάποιου επεξεργαστή. Η λύση προέκυψε από έναν μαθητή του Massimo Banzi, τον Hernando Barragan, ο οποίος δημιούργησε την πλατφόρμα προτυποποίησης Wiring, η οποία περιλάμβανε ένα φιλικό προς το χρήστη περιβάλλον ανάπτυξης λογισμικού και μία ηλεκτρονική πλακέτα με μικροεπεξεργαστή. Τελικά, το 2005 η ομάδα του Banzi κατασκεύασε την πρώτη πλατφόρμα Arduino και από τότε μέχρι και σήμερα έχει κατασκευαστεί μία πληθώρα μοντέλων και παραλλαγών της [18], [19], [20], [21].

Η πλατφόρμα Arduino αποτελείται από τα εξής συστατικά: το υλικό (hardware), το λογισμικό (software) και την κοινότητα Arduino (Arduino community), τα ιδιαίτερα χαρακτηριστικά των οποίων δίνουν εκείνα τα πλεονεκτήματα που κατέστησαν την πλατφόρμα Arduino ως την πλέον δημοφιλέστερη πλατφόρμα για την ανάπτυξη πρωτοτύπων. Έτσι, η πλατφόρμα Arduino προσφέρει:

- Χαμηλού κόστους υλικά – Η πλακέτα Arduino κοστίζει λιγότερο από αντίστοιχες άλλες πλακέτες ανάπτυξης πρωτοτύπων. Στο ηλεκτρονικό εμπόριο μπορεί κάποιος να αγοράσει πλακέτα Arduino με κόστος κάτω των 2 ευρώ.
- Λογισμικό ανάπτυξης ανεξαρτήτου πλατφόρμας (cross-platform) – Το λογισμικό ανάπτυξης Arduino (IDE: Integrated Development Environment) υποστηρίζεται από τα λειτουργικά συστήματα Windows, Macintosh OSX και Linux, ενώ οι περισσότεροι μικροεπεξεργαστές περιορίζονται στα Windows
- Απλό περιβάλλον προγραμματισμού – Το λογισμικό ανάπτυξης IDE του Arduino είναι βασισμένο στο περιβάλλον προγραμματισμού Processing το οποίο είναι απλό και

εύχρηστο, ειδικά κατασκευασμένο για αρχάριους χρήστες που δεν είναι εξοικειωμένοι με προηγμένες έννοιες προγραμματισμού, αλλά παρέχοντας ταυτόχρονα δυνατότητες στους προχωρημένους χρήστες

- Λογισμικό ανοικτού κώδικα (open source) και επεκτάσιμο – Το λογισμικό προγραμματισμού είναι ανοικτού κώδικα και οι προχωρημένοι χρήστες μπορούν να το επεκτείνουν και να το βελτιώσουν. Για περισσότερη ευελιξία μπορούν να χρησιμοποιηθούν βιβλιοθήκες σε κώδικα C++, καθώς επίσης να χρησιμοποιηθεί η γλώσσα προγραμματισμού AVR C στην οποία είναι βασισμένη η γλώσσα του Arduino και αξιοποιεί σε μεγαλύτερο βαθμό τις δυνατότητες του επεξεργαστή AVR
- Υλικό ανοικτού κώδικα – Τα σχέδια των πλακετών Arduino είναι δημοσιευμένα με την άδεια της Creative Commons, η οποία επιτρέπει τη δημόσια χρήση τους με σκοπό την προώθηση της γνώσης και της δημιουργικότητας. Έτσι ο οποιοσδήποτε μπορεί να χρησιμοποιήσει τα σχέδια των πλακετών, να τα τροποποιήσει, να τα βελτιώσει και να κατασκευάσει τη δικιά του πλακέτα [16], [22].
- Την κοινότητα Arduino – Το λογισμικό ανοικτού κώδικα και το υλικό με δημόσια άδεια του Arduino έδωσαν το πλεονέκτημα της ελεύθερης διακίνησης της γνώσης και των ιδεών, έτσι ώστε όσοι ενασχολούνται με το Arduino να αποτελέσουν μία κοινότητα μέσα στην οποία να μπορούν να βοηθηθούν αλλά και να συνεισφέρουν με νέες γνώσεις, βελτιώσεις και επιτεύγματα.

4.2 Το υλικό (Hardware)

Το υλικό της πλατφόρμας προτυποποίησης Arduino είναι μία μητρική πλακέτα, η καρδιά της οποίας είναι κάποιος μικροεπεξεργαστής. Μέχρι το 2015 χρησιμοποιούσε τους μικροεπεξεργαστές AVR 8-, 16-, ή 32-bit της Atmel. Από το 2015 χρησιμοποιούνται επιπλέον επεξεργαστές και άλλων κατασκευαστριών εταιριών. Η πλακέτα παρέχει ακροδέκτες για ψηφιακές και αναλογικές εισόδους και εξόδους, όπως και εξόδους διαμόρφωσης εύρους παλμών PWM (Pulse Width Modulation). Στη μητρική πλακέτα μπορούν να συνδεθούν πλακέτες επέκτασης, shields, οι οποίες παρέχουν περεταίρω δυνατότητες στο Arduino, όπως δυνατότητες ασύρματης επικοινωνίας Wi-Fi και Bluetooth, προσθήκη αισθητήρων επιτάχυνσης και γυροσκοπίου, δυνατότητες κίνησης με κινητήρες (motors) κ.α.

Για τον προγραμματισμό του ο μικροεπεξεργαστής επικοινωνεί σειριακά με έναν προσωπικό υπολογιστή. Δεν απαιτείται ειδική συσκευή προγραμματισμού, διότι στη μνήμη flash του μικροεπεξεργαστή υπάρχει ήδη υλικολογισμικό (firmware) που εκτελείται κατά την εκκίνησή του, το bootloader, το οποίο διευκολύνει τη φόρτωση νέων προγραμμάτων στη μνήμη flash κατευθείαν από έναν προσωπικό υπολογιστή. Στην πλακέτα Arduino μπορεί να υπάρχουν μετατροπείς στάθμης σειριακών σημάτων μεταξύ σημάτων RS232 και TTL (Transistor-Transistor Logic), ή μετατροπείς σημάτων USB σε RS232 με τη χρήση συνήθως του ολοκληρωμένου κυκλώματος (chip) της εταιρίας FDTI, το FT232. Σε ένα πρόσφατο μοντέλο Arduino, το UNO, αντί του FT232 υπάρχει ένα επιπλέον AVR chip, το οποίο περιέχει firmware μετατροπής σημάτων USB σε RS232, καθώς και ακροδέκτες προγραμματισμού ICSP (In-Circuit Serial Programming) [23].

Μία πλακέτα Arduino διαθέτει τρεις μνήμες: τη μνήμη Flash, τη στατική μνήμη τυχαίας προσπέλασης SRAM (Static Random-Access Memory) και τη μνήμη EEPROM (Electrically Erasable Programmable Read Only Memory). Τα περιεχόμενα των μνημών Flash και EEPROM διατηρούνται με την απώλεια του ηλεκτρικού ρεύματος, ενώ ό,τι υπάρχει στην SRAM σβήνεται όταν σταματήσει η τροφοδοσία με ρεύμα. Η μνήμη Flash είναι πιο αργή και προσπελαύνονται τμήματα δεδομένων (blocks of Bytes), ενώ η EEPROM είναι πιο γρήγορη και προσπελαύνονται Bytes δεδομένων. Στη μνήμη Flash βρίσκεται ο bootloader και γράφεται το πρόγραμμα του

χρήστη. Στη μνήμη SRAM το πρόγραμμα όταν εκτελείται γράφει και διαβάζει τις τιμές των μεταβλητών που χρησιμοποιεί. Στην EEPROM ο προγραμματιστής αποθηκεύει πληροφορίες οι οποίες θα χρησιμοποιούνται από το πρόγραμμα για μεγάλο διάστημα και δεν θα χάνονται όταν διακόπτεται η παροχή του ρεύματος [24], [25].

Τις αυθεντικές πλακέτες Arduino αρχικά τις κατασκεύαζε η Ιταλική εταιρία Smart Projects, και εν συνεχεία και οι Αμερικάνικες εταιρίες SparkFun Electronics, Adafruit Industries και Gravitech Electronic Experimental Solutions . Με την ανάπτυξη νέων πλακετών αντικαθίστανται κάποιες και αποσύρονται από την παραγωγή. Αυτή τη στιγμή κυκλοφορούν 15 μοντέλα πλακετών Arduino: [23], [26]

- UNO: Αποτελεί τη δημοφιλέστερη πλακέτα Arduino. Είναι βασισμένη στον μικροεπεξεργαστή ATmega328P, έχει 14 ψηφιακές εισόδους/εξόδους (από τις οποίες 6 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 6 αναλογικές εισόδους, 16MHz κρυσταλλικό ταλαντωτή, σύνδεση USB, είσοδο τροφοδοσίας ρεύματος, ακροδέκτες ICSP, και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 32kB από τα οποία τα 0,5kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 2kB και μνήμη EEPROM 1kB. Υποστηρίζει επικοινωνία SPI και I^2C με περιφερειακές του συσκευές.
- 101: Σχεδιάστηκε σε συνεργασία με την Intel. Χρησιμοποιεί το μικροεπεξεργαστή Inter Curie, ο οποίος έχει δύο πυρήνες, έναν x86 (Quark) και έναν 32-bit αρχιτεκτονικής ARC (Argonaut Risc Core) με ρολόι χρονισμού στα 32MHz. Επιπρόσθετα του Arduino UNO, έχει δυνατότητες Bluetooth και αισθητήρα επιτάχυνσης/γυροσκόπιο 6 αξόνων. Διαφέρει επίσης από το UNO στο ότι έχει 4 εξόδους PWM, μνήμη Flash 196kB και μνήμη SRAM 24kB.
- PRO: Έχει σχεδιαστεί για μόνιμη εγκατάσταση σε κατασκευές. Βασίζεται στον ATmega168 ή στον ATmega328. Σε σχέση με το UNO, έχει είσοδο για σύνδεση μπαταρίας, και ακροδέκτες για να συνδεθεί ο μετατροπέας της FDTI από USB σε RS232. Στην έκδοση με τον ATmega168 έχει μνήμη Flash 16kB, μνήμη SRAM 1kB και μνήμη EEPROM 512Bytes.
- MICRO: Βασίζεται στο μικροεπεξεργαστή ATmega32U4 και αναπτύχθηκε με συνεργασία της Adafruit. Διαθέτει 20 ψηφιακές εισόδους/εξόδους (από τις οποίες 7 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM και 12 ως αναλογικές είσοδοι), 16MHz κρυσταλλικό ταλαντωτή, σύνδεση micro USB, ακροδέκτες ICSP, και κουμπί επανεκκίνησης. Ο συγκεκριμένος επεξεργαστής έχει ενσωματωμένη δυνατότητα επικοινωνίας USB κι έτσι δεν είναι απαραίτητος ο μετατροπέας της FDTI. Επίσης έχει μνήμη Flash 32kB από τα οποία τα 4kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 2,5kB και μνήμη EEPROM 1kB.
- PRO MINI: Σχεδιάστηκε και κατασκευάζεται από τη SparkFun Electronics και προορίζεται για μόνιμη εγκατάσταση. Είναι βασισμένη στον μικροεπεξεργαστή ATmega328P, όπως και το UNO. Απουσιάζει η είσοδος για τροφοδοσία ρεύματος, όπως και ο μετατροπέας της FDTI.
- NANO: Σχεδιάστηκε και κατασκευάζεται από την εταιρία Gravitech. Όπως το PRO MINI, αλλά υπάρχει και έκδοση με τον ATmega168. Επιπλέον διαθέτει το μετατροπέα της FDTI και σύνδεση micro USB.
- MEGA 2560: Βασίζεται στο μικροεπεξεργαστή ATmega2560, έχει 54 ψηφιακές εισόδους/εξόδους (από τις οποίες 15 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 16 αναλογικές εισόδους, 16MHz κρυσταλλικό ταλαντωτή, σύνδεση USB, είσοδο

τροφοδοσίας ρεύματος, ακροδέκτες ICSP, και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 256kB από τα οποία τα 8kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 8 kB και μνήμη EEPROM 4kB.

- ZERO: Βασίζεται στο μικροεπεξεργαστή της Atmel, ATSAMD21G18, 32-Bit ARM Cortex M0+. Διαθέτει 20 ψηφιακές εισόδους/εξόδους (από τις οποίες 10 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 48MHz ρολόι χρονισμού, 2 συνδέσεις USB, η μία για προγραμματισμό, είσοδο τροφοδοσίας ρεύματος και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 256kb, μνήμη SRAM 32kB και μνήμη EEPROM 16kB με εξομοίωση.
- DUE: Είναι η πρώτη πλακέτα που βασίζεται σε 32-bit ARM επεξεργαστή. Διαθέτει τον επεξεργαστή της Atmel SAM3X8E ARM Cortex-M3. Έχει 54 ψηφιακές εισόδους/εξόδους (από τις οποίες 12 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 12 αναλογικές εισόδους, 84MHz ρολόι της CPU, 2 συνδέσεις USB και δύο σειριακές, είσοδο τροφοδοσίας ρεύματος και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 512kB διαθέσιμη ολόκληρη για τους χρήστες, μνήμη SRAM 96kB. Υποστηρίζει επικοινωνία SPI και δύο διαύλους I2C με περιφερειακές του συσκευές.
- YUN: Βασίζεται στο μικροεπεξεργαστή της Atmel ATmega32u4 όπως και το MICRO. Όμως διαθέτει και τον μικροεπεξεργαστή Atheros AR9331, ο οποίος προσθέτει σύνδεση Ethernet, ασύρματη σύνδεση WiFi, θέση για κάρτα SD, και σύνδεση USB για επικοινωνία με περιφερειακές συσκευές. Οι συνδέσεις Ethernet και WiFi λειτουργούν μόνο σε περιβάλλον Linux.
- GEMMA: Ανήκει στις πλακέτες που αναπτύχθηκαν με σκοπό να μπορούν να ραφτούν και να φορεθούν (wearable) και κατασκευάζεται από την Adafruit. Βασίζεται στο μικροεπεξεργαστή ATtiny85 της Atmel. Έχει 3 ψηφιακές εισόδους/εξόδους (από τις οποίες 2 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM και μία ως αναλογική είσοδος), 8MHz ταλαντωτή, σύνδεση micro USB και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 8kB από τα οποία τα 2,75kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 512Bytes και μνήμη EEPROM 512Bytes. Υποστηρίζει επικοινωνία I2C με περιφερειακές του συσκευές.
- LILYPAD MAIN BOARD: Βασίζεται στον μικροεπεξεργαστή ATmega168V (που είναι η έκδοση χαμηλής ενέργειας του ATmega168) ή στον ATmega368V. Ο σχεδιασμός και η ανάπτυξη έγινε από την Leah Buechley και την εταιρία SparkFun. Έχει 14 ψηφιακές εισόδους/εξόδους (από τις οποίες 6 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 6 αναλογικές εισόδους, 8MHz ρολόι, ακροδέκτες ICSP, και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 16kB από τα οποία τα 2kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 1kB και μνήμη EEPROM 512Bytes.
- LILYPAD SIMPLE: Βασίζεται στον μικροεπεξεργαστή ATmega328. Έχει 9 ψηφιακές εισόδους/εξόδους (από τις οποίες 5 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM), 4 αναλογικές εισόδους, 8MHz ρολόι και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 32kB από τα οποία τα 2kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 2kB και μνήμη EEPROM 1kB.
- LILYPAD SIMPLE SNAP: Είναι παρόμοιο με το LILYPAD SIMPLE με τη διαφορά ότι αντί να έχει περιμετρικά τρύπες για να περάσει κλωστή και να ραφτεί, διαθέτει μεταλλικά κουμπιά. Επίσης υπάρχει ενσωματωμένη μπαταρία λιθίου πολυμερών.

- LILYPAD USB: Βασίζεται στον μικροεπεξεργαστή ATmega32u4. Έχει 9 ψηφιακές εισόδους/εξόδους (από τις οποίες 4 μπορούν να χρησιμοποιηθούν ως έξοδοι PWM και 4 ως αναλογικές είσοδοι), 8MHz ρολόι, σύνδεση micro USB και κουμπί επανεκκίνησης. Επίσης έχει μνήμη Flash 32kB από τα οποία τα 4kB χρησιμοποιούνται από το bootloader, μνήμη SRAM 2,5kB και μνήμη EEPROM 1kB.

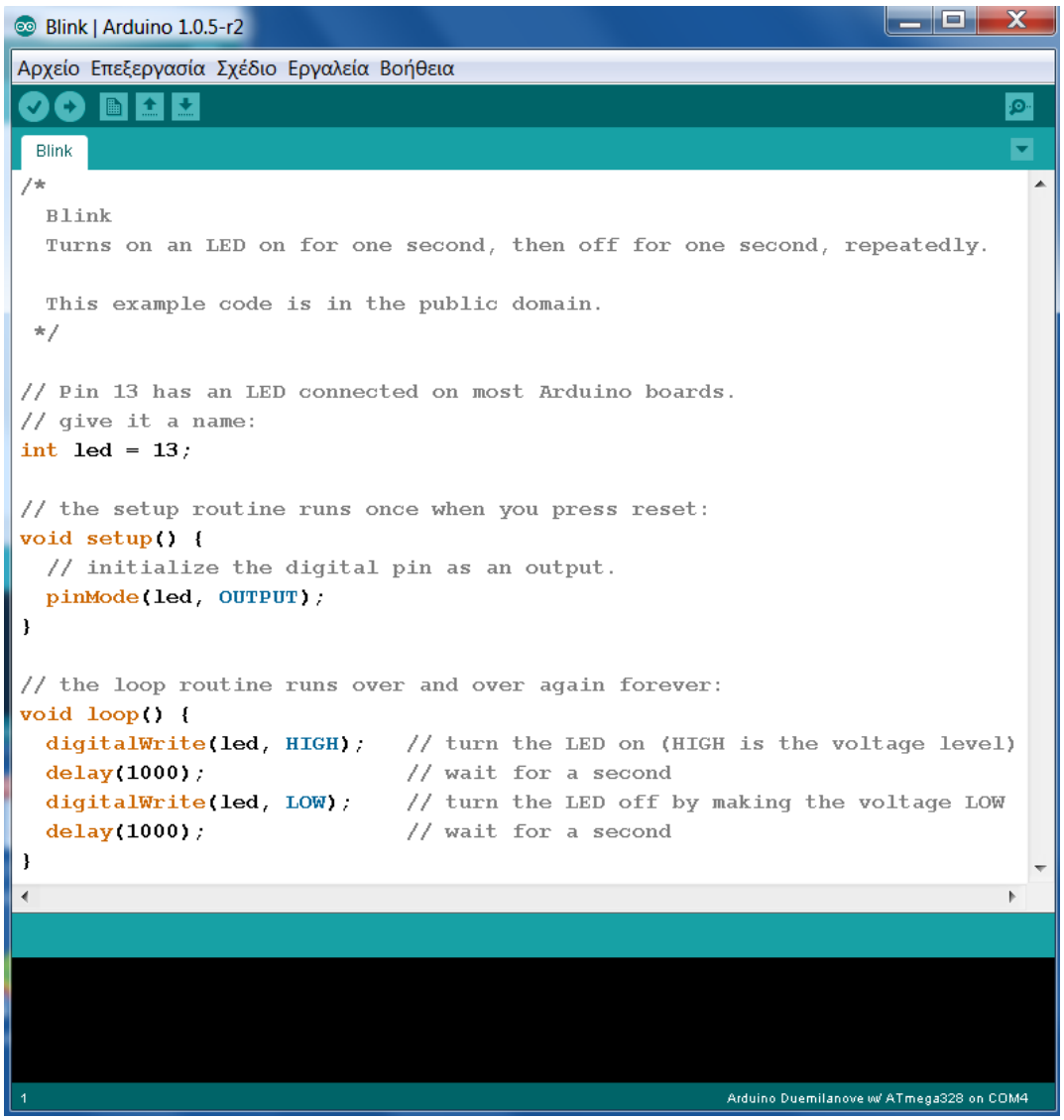
4.3 Το λογισμικό προγραμματισμού και το περιβάλλον ανάπτυξης (IDE)

Ο προγραμματισμός του Arduino μπορεί να γίνει με οποιαδήποτε γλώσσα προγραμματισμού αρκεί ο κώδικας να μετατραπεί σε γλώσσα μηχανής με το μεταγλωττιστή (compiler). Η Atmel διαθέτει το περιβάλλον ανάπτυξης Atmel Studio για τον προγραμματισμό των μικροεπεξεργαστών AVR.

Η πλατφόρμα Arduino παρέχει το δικό της ολοκληρωμένο περιβάλλον ανάπτυξης IDE (Integrated Development Environment), το οποίο είναι γραμμένο σε γλώσσα Java και είναι βασισμένο στο IDE της γλώσσας προγραμματισμού Processing. Η γλώσσα προγραμματισμού που χρησιμοποιείται στο Arduino IDE είναι βασισμένη στη γλώσσα προγραμματισμού της πλατφόρμας Wiring, η οποία είναι μια παραλλαγή της γλώσσας C για προγραμματισμό μικροεπεξεργαστών. Το Arduino IDE υποστηρίζει όλες τις βασικές δομές της γλώσσας C και κάποια χαρακτηριστικά της C++, αλλά όμως χρησιμοποιεί ειδικούς κανόνες στην οργάνωση του κώδικα. Επιπρόσθετα διαθέτει κάποιες ειδικές εντολές, συναρτήσεις και σταθερές ειδικά κατασκευασμένες για την πλακέτα Arduino, βιβλιοθήκες για τη διασύνδεσή της με συγκεκριμένους τύπους υλικού (hardware), καθώς και τη βιβλιοθήκη AVR Libc για την αξιοποίηση περαιτέρω των δυνατοτήτων των επεξεργαστών AVR.

Το πρόγραμμα που δημιουργεί ο χρήστης στο Arduino IDE ονομάζεται sketch. Το Arduino IDE έχει σκοπό να διευκολύνει το χρήστη στην κατασκευή του sketch και στην επιτυχή μεταφόρτωσή του στο μικροεπεξεργαστή του Arduino. Διαθέτει συντακτική χρωματική σήμανση, παραδείγματα έτοιμων sketch προς άμεση εφαρμογή, βιβλιοθήκες για σύνδεση της πλακέτας Arduino με κάποιες συγκεκριμένες συσκευές και παράθυρο για την εποπτεία της σειριακής θύρας USB και την αποστολή αλφαριθμητικών χαρακτήρων από τον υπολογιστή προς το Arduino. Μετά τη δημιουργία του sketch, το Arduino IDE δίνει τη δυνατότητα της μεταγλώττισης του sketch μέσω του ενσωματωμένου μεταγλωττιστή του, και στη συνέχεια να μεταφορτωθεί στον επεξεργαστή του Arduino.

Ένα sketch αποτελείται από δύο βασικές συναρτήσεις: τη *setup()* και τη *loop()*. Η συνάρτηση *setup()* εκτελείται μία φορά κατά την εκκίνηση του προγράμματος και έχει σκοπό την αρχικοποίηση κάποιων παραμέτρων του προγράμματος. Η συνάρτηση *loop()* εκτελείται κυκλικά διαρκώς μέχρι να τεθεί εκτός λειτουργίας η πλακέτα Arduino και περιέχει το βασικό κορμό του προγράμματος [17], [23].



Εικόνα 33. Το περιβάλλον ανάπτυξης Arduino IDE με ανοιγμένο sketch το παράδειγμα blink

ΚΕΦΑΛΑΙΟ 5 - Το υλικό της εφαρμογής τηλεχειρισμού

Σε αυτή την εφαρμογή χρησιμοποιήθηκε ως πλακέτα Arduino το Arduino Duemilanove, το οποίο αποσύρθηκε από την κυκλοφορία με την εμφάνιση του Arduino UNO. Η βασική διαφορά τους είναι στο chip μετατροπής της θύρας USB σε σειριακή. Το UNO γι αυτό το σκοπό χρησιμοποιεί το ATMega8U2, ενώ το Duemilanove χρησιμοποιεί το FT232 της FDTI.

Για την επέκταση των ακροδεκτών εισόδου/εξόδου της πλακέτας Arduino χρησιμοποιήθηκαν 8 ολοκληρωμένα κυκλώματα MCP23017 της Microchip. Το καθένα από αυτά προσφέρει 16 ψηφιακές εισόδους/εξόδους, οπότε τα 8 έδωσαν επιπρόσθετα άλλους 128 ακροδέκτες εισόδου/εξόδου.

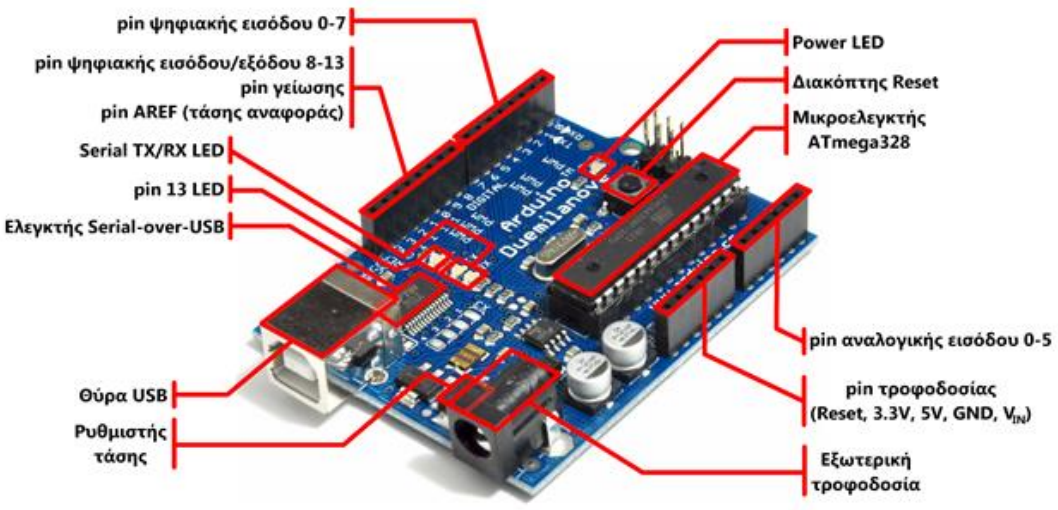
Για την ενημέρωση του χρήστη για την κατάσταση των ακροδεκτών των οκτώ MCP23017 χρησιμοποιήθηκε μία οθόνη υγρών κρυστάλλων (LCD) απεικόνισης αλφαριθμητικών χαρακτήρων είκοσι στηλών και τεσσάρων γραμμών, η J204A (20x4).

Χρησιμοποιήθηκε επίσης η πλακέτα επέκτασης Arduino Ethernet Shield για να συνδεθεί το Arduino με δίκτυο Ethernet μέσω καλωδίου RJ45, με σκοπό την απεικόνιση της κατάστασης και τον έλεγχο των ακροδεκτών εισόδου/εξόδου των MCP23017 από τον υπολογιστή.

Στη συνέχεια περιγράφονται συνοπτικά τα βασικά χαρακτηριστικά των υλικών που χρησιμοποιήθηκαν στην εφαρμογή.

5.1 Το Arduino Duemilanove

Η ονομασία του προήλθε από τη λέξη “2009” που είναι το έτος της κυκλοφορίας του. Βασίζεται στον μικροεπεξεργαστή ATmega328 με ρολόι χρονισμού στα 16MHz, ο οποίος διαθέτει 2kB μνήμη SRAM, 1kB μνήμη EEPROM και 32kB μνήμη Flash. Ο μικροεπεξεργαστής υποστηρίζει επίσης σειριακή επικοινωνία και με ένα μετατροπέα από USB σε σειριακή της FDTI, το FT232, η πλακέτα Duemilanove του παρέχει τη δυνατότητα να συνδεθεί με έναν προσωπικό υπολογιστή. Το υλικολογισμικό (firmware) των 2kB που υπάρχει ήδη γραμμένο στη μνήμη Flash και ονομάζεται bootloader, εκτελείται κατά την εκκίνηση και προσφέρει τη δυνατότητα στο μικροεπεξεργαστή να προγραμματιστεί από έναν προσωπικό υπολογιστή μέσω της USB χωρίς τη χρήση συσκευής προγραμματισμού μικροεπεξεργαστών.



Εικόνα 34. Η πλακέτα Arduino Duemilanove [17]

Η πλακέτα μπορεί να τροφοδοτηθεί με ρεύμα από τον υπολογιστή μέσω της θύρας USB, είτε από την υποδοχή ρεύματος με εξωτερική τροφοδοσία μεταξύ 7 και 12V. Στην περίπτωση που δεν εξυπηρετεί η υποδοχή ρεύματος, η πλακέτα μπορεί να τροφοδοτηθεί από τον ακροδέκτη Vin της θηλυκής ακιδοσειράς τροφοδοσίας από εξωτερική τροφοδοσία. Εάν η πλακέτα τροφοδοτείται από την υποδοχή ρεύματος, τότε από τον ακροδέκτη Vin μπορούμε να έχουμε πρόσβαση κατευθείαν σε αυτή την τροφοδοσία. Σε αυτή την ακιδοσειρά υπάρχει επίσης ο ακροδέκτης Reset, ο οποίος όταν γειωθεί θα γίνει επανεκκίνηση του Arduino, οι ακροδέκτες 3,3V και 5V από τους οποίους μπορούμε να τροφοδοτήσουμε τα εξαρτήματά μας με τις αντίστοιχες τάσεις, όπως επίσης και δύο ακροδέκτες GND για γείωση.

Διαθέτει 14 ακροδέκτες, 0 έως 13, οι οποίοι μπορούν να προγραμματιστούν ως ψηφιακές είσοδοι ή έξοδοι:

- Οι ακροδέκτες 0 και 1 αποτελούν το σήμα λήψης RX και το σήμα εκπομπής TX αντίστοιχα, όταν επικοινωνεί ο μικροεπεξεργαστής σειριακά με τον υπολογιστή μέσω του μετατροπέα σήματος USB σε σήμα TTL.
- Οι ακροδέκτες 2 και 3 μπορούν να ενεργοποιήσουν διακοπές (interrupts) στο μικροεπεξεργαστή όταν συμβαίνουν ανερχόμενες ή κατερχόμενες ακμές τάσης στις επαφές τους και μπορούν να χρησιμοποιηθούν για παράδειγμα όταν θέλουμε να ανιχνευθούν γρήγορες μεταβολές όπως είναι η διακοπή κάποιας ακτίνας laser.
- Οι ακροδέκτες 3, 5, 6, 9, 10 και 11 μπορούν να προγραμματιστούν ως έξοδοι PWM των 8-bit. Το εύρος των παλμών μπορεί να μεταβληθεί μεταξύ 2^8=256 καταστάσεων και να ρυθμίσουν έτσι τη φωτεινότητα σε ένα λαμπάκι led ή τις στροφές ενός βηματικού κινητήρα.
- Οι ακροδέκτες 10, 11, 12, 13 μπορούν να χρησιμοποιηθούν για τη σύνδεση του μικροεπεξεργαστή με κάποια συσκευή μέσω του διαύλου SPI και μεταφέρουν τα τέσσερα σήματα SS, MOSI, MISO και SCK αντίστοιχα.
- Ο ακροδέκτης 13 είναι συνδεδεμένος με ένα LED και όταν τεθεί σε κατάσταση HIGH το LED ανάβει.

Διαθέτει επίσης 6 ακροδέκτες, A0 έως A5, ως αναλογικές εισόδους με ανάλυση 10 bit. Είναι προκαθορισμένες να μετρούν τάσεις μέχρι 5V με διαβάθμιση 1024 επιπέδων. Η ανώτατη τάση των 5V μπορεί να αλλάξει τροφοδοτώντας το pin AREF με οποιαδήποτε τάση μεταξύ 2 και 5V.

Οι αναλογικές είσοδοι A4 και A5 μπορούν να χρησιμοποιηθούν για τη σύνδεση του μικροεπεξεργαστή με συσκευές μέσω του διαύλου I^2C και μεταφέρουν τα δύο σήματα SDA και SCL αντίστοιχα [17], [27].

5.2 Η πλακέτα επέκτασης Arduino Ethernet Shield

Η Arduino Ethernet Shield βασίζεται στο ολοκληρωμένο κύκλωμα Wiznet W5100 για σύνδεση στο Ethernet. Με την πλακέτα Arduino επικοινωνεί μέσω του διαύλου SPI. Για τη σύνδεση με το Arduino διαθέτει από την κάτω πλευρά της πλακέτας αρσενικές ακιδοσειρές οι οποίες κουμπώνουν επάνω στις θηλυκές ακιδοσειρές του Arduino. Η σύνδεση με SPI γίνεται μέσω της ακιδοσειράς ICSP και αντιστοιχεί στους ακροδέκτες 10, 11, 12 και 13 των ψηφιακών εισόδων/εξόδων [28].

Το Wiznet W5100 λειτουργεί ως συσκευή SPI Slave και υποστηρίζει τον τρόπο λειτουργίας 0 (Mode 0). Επίσης χρησιμοποιεί δύο κωδικούς λειτουργίας (op-codes), έναν για ανάγνωση και έναν για εγγραφή και λειτουργεί με μονάδες ροής δεδομένων 32-bit (unit of 32-bit stream). Η μία μονάδα αποτελείται από 1 byte του op-code, 2 bytes διεύθυνση και 1 byte δεδομένων. Πρώτα

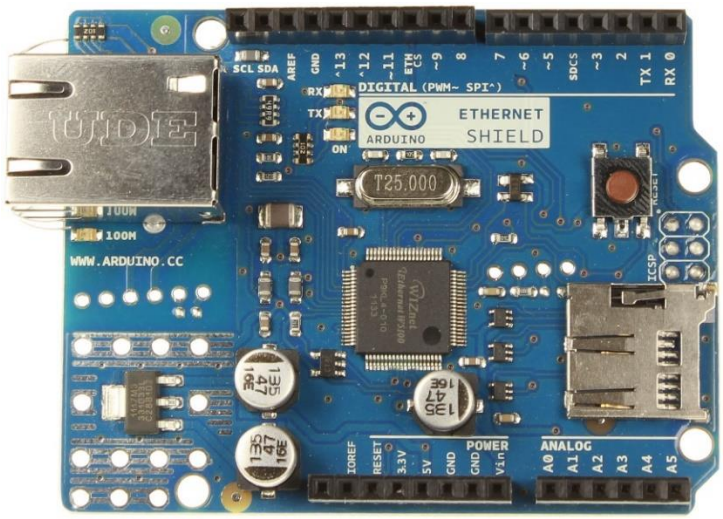
μεταδίδεται το πιο σημαντικό ψηφίο (MSB) του op-code και στο τέλος το λιγότερο σημαντικό ψηφίο (LSB) των δεδομένων [29].

Command	OP-Code Field		Address Field	Data Field
Write operation	0xF0	1111 0000	2 bytes	1 byte
Read operation	0x0F	0000 1111	2 bytes	1 byte

Εικόνα 35. Η μορφή μίας μονάδας ροής δεδομένων 32-bit [29]

Μέσω του διαύλου SPI επικοινωνεί επίσης και η κάρτα μνήμης SD που μπορεί να τοποθετηθεί στην ειδική θήκη που υπάρχει πάνω στην κάρτα. Κάθε φορά όμως μπορεί να επικοινωνήσει μία μόνο συσκευή με το μικροεπεξεργαστή του Arduino μέσω του διαύλου SPI. Όταν επικοινωνεί το chip W5100 τότε δεν μπορεί να επικοινωνήσει η κάρτα SD και αντίστροφα. Ο ακροδέκτης 10 χρησιμοποιείται για την επιλογή του W5100, ενώ ο ακροδέκτης 4 για την επιλογή της κάρτας SD. Όταν μία από τις δύο συσκευές δεν χρησιμοποιείται καθόλου από την εφαρμογή μας, τότε καλό είναι να απενεργοποιείται τελείως. Εάν για παράδειγμα δεν χρησιμοποιείται η συσκευή SD, τότε πρέπει να προγραμματιστεί ο ακροδέκτης 4 ως έξοδος και να τεθεί σε κατάσταση HIGH. Παρόμοια πρέπει να πραγματοποιηθεί για το W5100 και για τον ακροδέκτη 10.

Η Ethernet Shield παρέχει διεύθυνση IP και στα δύο πρωτόκολλα TCP και UDP. Για να μπορεί η πλακέτα να ενταχθεί σε μία εφαρμογή αρκεί να γίνει χρήση της βιβλιοθήκης Ethernet και των ειδικών εντολών της στο πρόγραμμα sketch του χρήστη.



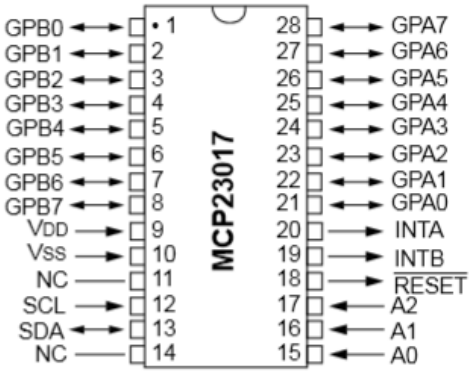
Εικόνα 36. Η πλακέτα επέκτασης Ethernet Shield [28]

Στην πλακέτα Ethernet Shield μπορούμε επιπλέον να εντοπίσουμε την υποδοχή RJ45 για τη σύνδεση Ethernet, το κουμπί Reset για επανεκκίνηση της Ethernet Shield αλλά και του Arduino, όπως επίσης και κάποια ενδεικτικά LED. Το LED Link δηλώνει την παρουσία σύνδεσης με δίκτυο και αναβοσβήνει κατά την εκπομπή ή τη λήψη δεδομένων. Το LED FULLD δείχνει ότι η σύνδεση με το δίκτυο είναι πλήρως αμφίδρομη (Full Duplex). Το LED 100M επισημαίνει την παρουσία σύνδεσης στα 100Mb/s. Τα LED RX και TX αναβοσβήνουν κατά τη λήψη ή εκπομπή δεδομένων αντίστοιχα και το LED COLL δηλώνει ότι ανιχνεύθηκαν συγκρούσεις στο δίκτυο [28].

5.3 Το ολοκληρωμένο κύκλωμα επέκτασης εισόδων/εξόδων MCP23017

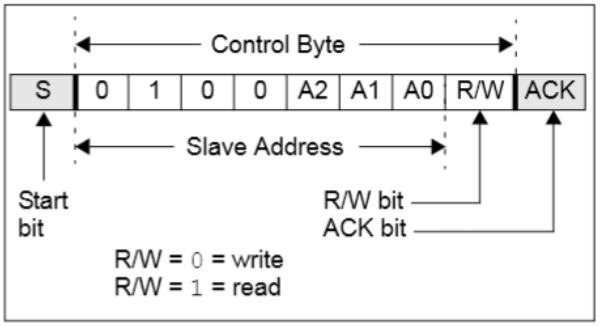
Η χρήση του είναι απαραίτητη όταν οι ακροδέκτες ψηφιακών εισόδων/εξόδων που μας παρέχει η πλακέτα Arduino δεν επαρκούν για τις ανάγκες της εφαρμογής μας. Το MC23017 μας παρέχει επιπλέον 16 ακροδέκτες και συνδέεται στο Arduino μέσω του διαύλου I²C.

Οι ακροδέκτες εισόδου/εξόδου αντιστοιχούν σε δύο θύρες των 8-bit η καθεμία, η PORTA και η PORTB. Στην PORTA ανήκουν οι ακροδέκτες 21 (GPA0) έως 28 (GPA7), ενώ στην PORTB οι ακροδέκτες 1 (GPB0) έως 8 (GPB7). Κάθε ακροδέκτης μπορεί να προγραμματιστεί ως είσοδος ή έξοδος από τα bit των καταχωρητών IODIRA και IODIRB. Οι γραμμές SCL και SDA του διαύλου I²C συνδέονται στους ακροδέκτες 12 και 13 αντίστοιχα. Στον ακροδέκτη 9 (V_{DD}) συνδέεται η τροφοδοσία ρεύματος που μπορεί να κυμαίνεται από 2,7V έως 5,5V. Στον ακροδέκτη (V_{SS}) συνδέεται η γείωση. Από τον ακροδέκτη 18, όταν τεθεί στα 0V, μπορεί να πραγματοποιηθεί επανεκκίνηση του MC23017, δηλαδή επαναφορά των καταχωρητών στην προκαθορισμένη αρχική τους κατάσταση [30].

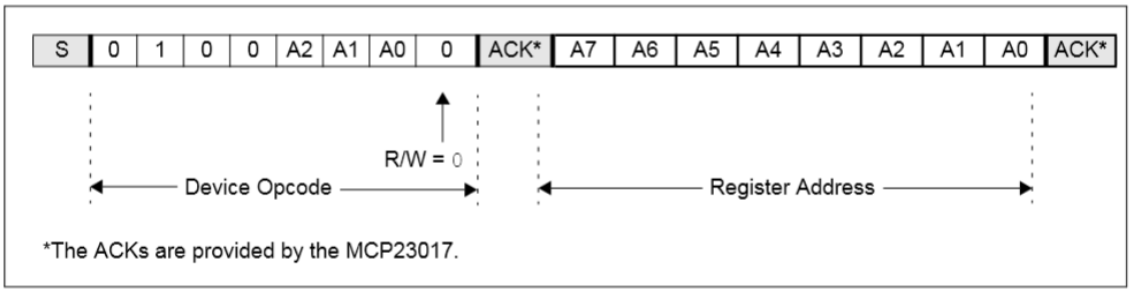


Εικόνα 37. Οι ακροδέκτες του MC23017 [30]

Η διευθυνσιοδότηση του MCP23017 μπορεί να γίνει από τους ακροδέκτες 15 (A0), 16 (A1) και 17 (A2). Ο συνδυασμός των τριών ακροδεκτών μπορεί να μας δώσει μέχρι 8 διευθύνσεις. Για παράδειγμα, όταν το A0 είναι στα 5V (λογικό “1”) και τα A1 και A2 είναι στα 0V (λογικό “0”), τότε το ολοκληρωμένο παίρνει τη διεύθυνση 001. Τα υπόλοιπα 4 bits από τα συνολικά 7 bits της διεύθυνσης slave του I²C πρωτοκόλλου είναι σταθερά 0100. Έτσι τα 7 bit διεύθυνσης έχουν τη μορφή 0100(A2)(A1)(A0).



Εικόνα 38. Το byte ελέγχου του πρωτοκόλλου I²C [30]



Εικόνα 39. Η επιλογή συγκεκριμένου MCP23017 και καταχωρητή του [30]

Οι ακροδέκτες 20 (INTA) και 19 (INTB) είναι έξοδοι για αιτήματα διακοπών (interrupts), οι οποίες είτε ενεργοποιούνται ανεξάρτητα η καθεμία από τις θύρες PORTA και PORTB αντίστοιχα, είτε ενεργοποιούνται και οι δύο μαζί με την αλλαγή της κατάστασης σε οποιαδήποτε εκ των δύο θυρών. Η ενεργοποίηση διακοπής μπορεί να προγραμματιστεί να συμβαίνει όταν υπάρχει αλλαγή κατάστασης από την ήδη υπάρχουσα σε κάποιον ακροδέκτη, ή όταν αυτή η αλλαγή στην κατάσταση διαφέρει από μία προκαθορισμένη τιμή [30].

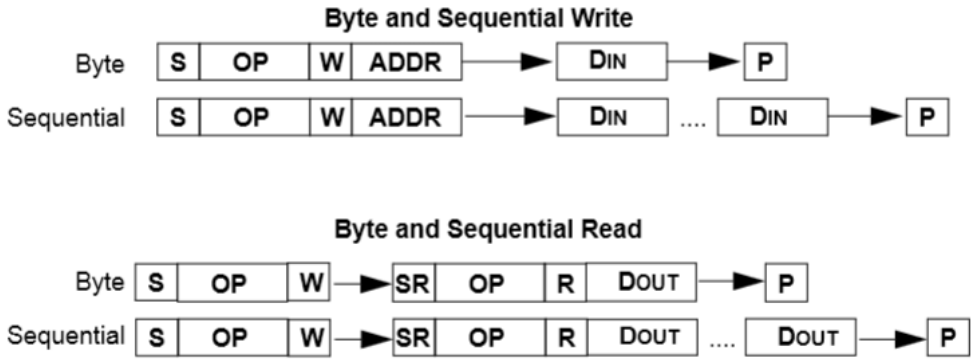
Address IOCON.BANK = 1	Address IOCON.BANK = 0	Access to:
00h	00h	IODIRA
10h	01h	IODIRB
01h	02h	IPOLA
11h	03h	IPOLB
02h	04h	GPINTENA
12h	05h	GPINTENB
03h	06h	DEFVALA
13h	07h	DEFVALB
04h	08h	INTCONA
14h	09h	INTCONB
05h	0Ah	IOCON
15h	0Bh	IOCON
06h	0Ch	GPPUA
16h	0Dh	GPPUB
07h	0Eh	INTFA
17h	0Fh	INTFB
08h	10h	INTCAPA
18h	11h	INTCAPB
09h	12h	GPIOA
19h	13h	GPIOB
0Ah	14h	OLATA
1Ah	15h	OLATB

Πίνακας 7. Η διευθύνσεις των καταχωρητών σε λειτουργία 8-bit και σε 16-bit [30]

Τους 8-bit καταχωρητές PORTA και PORTB υπάρχει η δυνατότητα να τους διαχειριζόμαστε ως ένα 16-bit καταχωρητή με τη ενεργοποίηση της 16-bit λειτουργίας του MC23017. Αυτή η λειτουργία ενεργοποιείται από το bit BANK του καταχωρητή IOCON όταν τεθεί στο “0”. Η λειτουργία αυτή ανήκει στις προκαθορισμένες επιλογές του MC23017 και είναι ενεργή με την εκκίνησή του. Με την αλλαγή της λειτουργίας από 8-bit σε 16-bit αλλάζουν επίσης οι διευθύνσεις των καταχωρητών του MC23017 και οι καταχωρητές της μίας θύρας ζευγαρώνονται (paired) με τους αντίστοιχους καταχωρητές της άλλης θύρας.

Υπάρχουν τρεις τρόποι λειτουργίας του MC23017:

- Το Bit BANK=”0”. Το MC23017 μεταπίπτει σε λειτουργία 16-bit και παράλληλα ο δείκτης διεύθυνσης εναλλάσσεται μεταξύ των διευθύνσεων του ζεύγους καταχωρητών στο οποίο τυγχάνει να βρίσκεται. Για παράδειγμα, εάν ο δείκτης βρίσκεται στη διεύθυνση 12h (GPIOA) ή στην 13h (GPIOB), ο δείκτης θα εναλλάσσεται μεταξύ του GPIOA και του GPIOB.
- Το Bit BANK=”1” και το Bit SEQOP=”1”. Το MC23017 βρίσκεται σε λειτουργία 8-bit και επίσης η διαδοχική λειτουργία (Sequential Mode) έχει απενεργοποιηθεί, βρίσκεται δηλαδή στη λειτουργία Byte (Byte Mode). Σε αυτή τη λειτουργία ο δείκτης διεύθυνσης δεν αυξάνει αυτόματα κατά τη μεταφορά δεδομένων. Έτσι δίνεται η δυνατότητα της συνεχούς πρόσβασης στην ίδια διεύθυνση, όταν για παράδειγμα θέλουμε να παρακολουθούμε συνεχώς έναν καταχωρητή εισόδου για αλλαγές ή να γράφουμε συνεχώς στις ίδιες εξόδους.
- Το Bit BANK=”1” και το Bit SEQOP=”0”. Το MC23017 βρίσκεται σε λειτουργία 8-bit και είναι επίσης ενεργή η διαδοχική λειτουργία (Sequential Mode). Σε αυτή τη λειτουργία ο δείκτης διεύθυνσης αυξάνει αυτόματα μετά από κάθε byte κατά τη μεταφορά δεδομένων. Μετά αφού φτάσει στον τελευταίο καταχωρητή, μεταπηδά πάλι στη διεύθυνση 00h [30].



Εικόνα 40. Μετάδοση δεδομένων στις λειτουργίες Byte και Sequential [30]

Στον παρακάτω πίνακα 8 παρατίθενται οι καταχωρητές ελέγχου του MCP23017 σε λειτουργία 16-bit (IOCON.BANK=0), καθώς και οι προκαθορισμένες αρχικές τους τιμές κατά την εκκίνηση ή την επανεκκίνηση του ολοκληρωμένου. Στη λειτουργία 16-bit, ανά δύο οι καταχωρητές που έχουν αντίστοιχους ρόλους στις δύο θύρες A και B, έχουν διαδοχικές διευθύνσεις, αποτελούν ζεύγη και αντιμετωπίζονται ως ένας 16-bit καταχωρητής. Συνοπτικά μπορούν να αναφερθούν για τους καταχωρητές τα εξής:

- Οι καταχωρητές IODIRA και IODIRB. Καθένα από τα 8 bit κάθε καταχωρητή αφορά και έναν ακροδέκτη του ολοκληρωμένου. Όταν ένα bit είναι “1”, τότε ο αντίστοιχος ακροδέκτης ορίζεται ως είσοδος, ενώ με “0” ορίζεται ως έξοδος.
- Οι IPOLA και IPOLB. Όταν ένα bit είναι “1”, τότε το αντίστοιχο bit στον καταχωρητή θύρας GPIOA ή GPIOB δείχνει την κατάσταση του αντίστοιχου ακροδέκτη με την αντίθετη πολικότητα (polarity). Όταν ένα bit είναι “0”, τότε η κατάσταση στον αντίστοιχο ακροδέκτη εμφανίζεται στο αντίστοιχο bit του GPIOA ή GPIOB με την πραγματική πολικότητα.
- Οι GPINTENA και GPINTENB. Ελέγχουν τη δυνατότητα να παράγονται αιτήματα διακοπών (interrupts) κατά την αλλαγή κατάστασης καθενός από τους ακροδέκτες εισόδου/εξόδου (Interrupt-on-change). Η δυνατότητα ενεργοποιείται σε έναν ακροδέκτη εισόδου/εξόδου όταν το bit του GPINTENA ή του GPINTENB που αναφέρεται στον ακροδέκτη αυτόν, γίνει “1”. Διαφορετικά με “0” η δυνατότητα αυτή για το συγκεκριμένο ακροδέκτη είναι ανενεργή.
- Οι INTCONA και INTCONB. Κάθε bit των καταχωρητών καθορίζει με ποιον τρόπο η τιμή του αντίστοιχου ακροδέκτη συγκρίνεται για να παραχθεί αίτημα διακοπής όταν είναι ενεργή η δυνατότητα Interrupt-on-change (αντίστοιχο bit GPINTENA ή GPINTENB ίσο με “1”). Εάν ένα bit στον INTCONA ή INTCONB είναι “0”, τότε η κατάσταση στον αντίστοιχο ακροδέκτη εισόδου/εξόδου συγκρίνεται με την προηγούμενη. Εάν το bit είναι “1”, τότε η κατάσταση στον αντίστοιχο ακροδέκτη συγκρίνεται με την τιμή που έχει τοποθετηθεί στο αντίστοιχο bit του καταχωρητή DEFVALA ή DEFVALB.
- Οι DEFVALA και DEFVALB. Περιέχουν τις προκαθορισμένες τιμές με τις οποίες θα συγκριθούν οι καταστάσεις των αντίστοιχων ακροδεκτών, όταν είναι ενεργή η δυνατότητα Interrupt-on-change για τους συγκεκριμένους ακροδέκτες και τα αντίστοιχα bit στο καταχωρητή INTCONA ή INTCONB είναι “1”.
- Οι IOCONA και IOCONB. Περιέχουν bits αρμόδια για τη διαμόρφωση της λειτουργίας του ολοκληρωμένου. Ιδιαίτερη σημασία έχει το bit BANK που όταν είναι “0” το ολοκληρωμένο τίθεται σε λειτουργία 8-bit, ενώ όταν είναι “1” τοποθετείται σε λειτουργία 16-bit. Επίσης με το bit SEQOP επιλέγεται ή όχι η διαδοχική λειτουργία
- Οι GPPUA και GPPUB. Με τα bit “1” ή “0” ενεργοποιείται ή απενεργοποιείται η εσωτερική pull-up αντίσταση 10kΩ των αντίστοιχων ακροδεκτών όταν λειτουργούν ως είσοδοι.
- Οι INTFA και INTFB. Οι καταχωρητές αυτοί είναι μόνο για ανάγνωση. Με “1” σε κάποιο bit δηλώνει ότι ο αντίστοιχος ακροδέκτης προκάλεσε αίτημα για διακοπή (interrupt)
- Οι INTCAPA και INTCAPB. Είναι μόνο για ανάγνωση. Αποθηκεύουν την τιμή του καταχωρητή θύρας GPIOA ή GPIOB τη στιγμή που συνέβη το interrupt. Ο καταχωρητής παραμένει ως έχει μέχρι να σβήσει το interrupt με ανάγνωση του INTCAP ή GPIO
- Οι GPIOA και GPIOB. Περιέχουν τις τιμές των ακροδεκτών, “0” ή “1”
- Οι OLATA και OLATB. Είναι οι καταχωρητές για τους ακροδέκτες όταν είναι προγραμματισμένοι ως έξοδοι. Με την ανάγνωση αυτών των καταχωρητών γίνεται ανάγνωση των OLAT και όχι των θυρών. Όμως με την εγγραφή σε αυτούς τους καταχωρητές τοποθετείται η κατάσταση στους αντίστοιχους ακροδέκτες εξόδου [30].

Register Name	Address (hex)	bit 7	bit 6	bit 5	bit 4	bit 3	bit 2	bit 1	bit 0	POR/RST value
IODIRA	00	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IODIRB	01	IO7	IO6	IO5	IO4	IO3	IO2	IO1	IO0	1111 1111
IPOLA	02	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
IPOLB	03	IP7	IP6	IP5	IP4	IP3	IP2	IP1	IP0	0000 0000
GPINTENA	04	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
GPINTENB	05	GPINT7	GPINT6	GPINT5	GPINT4	GPINT3	GPINT2	GPINT1	GPINT0	0000 0000
DEFVALA	06	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
DEFVALB	07	DEF7	DEF6	DEF5	DEF4	DEF3	DEF2	DEF1	DEF0	0000 0000
INTCONA	08	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
INTCONB	09	IOC7	IOC6	IOC5	IOC4	IOC3	IOC2	IOC1	IOC0	0000 0000
IOCON	0A	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
IOCON	0B	BANK	MIRROR	SEQOP	DISSLW	HAEN	ODR	INTPOL	—	0000 0000
GPPUA	0C	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
GPPUB	0D	PU7	PU6	PU5	PU4	PU3	PU2	PU1	PU0	0000 0000
INTFA	0E	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTFB	0F	INT7	INT6	INT5	INT4	INT3	INT2	INT1	INTO	0000 0000
INTCAPA	10	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
INTCAPB	11	ICP7	ICP6	ICP5	ICP4	ICP3	ICP2	ICP1	ICP0	0000 0000
GPIOA	12	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
GPIOB	13	GP7	GP6	GP5	GP4	GP3	GP2	GP1	GP0	0000 0000
OLATA	14	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000
OLATB	15	OL7	OL6	OL5	OL4	OL3	OL2	OL1	OL0	0000 0000

Πίνακας 8. Οι καταχωρητές ελέγχου σε λειτουργία 16-bit (IOCON.BANK=0) [30]

5.4 Η οθόνη υγρών κρυστάλλων 20x4 χαρακτήρων

Για τις ανάγκες τις εφαρμογής χρησιμοποιήθηκε η οθόνη υγρών κρυστάλλων 20 στηλών και 4 γραμμών J204A, συμβατή με την οθόνη HD44780 της Hitachi. Για τη λειτουργία της διαθέτει 16 ακροδέκτες:

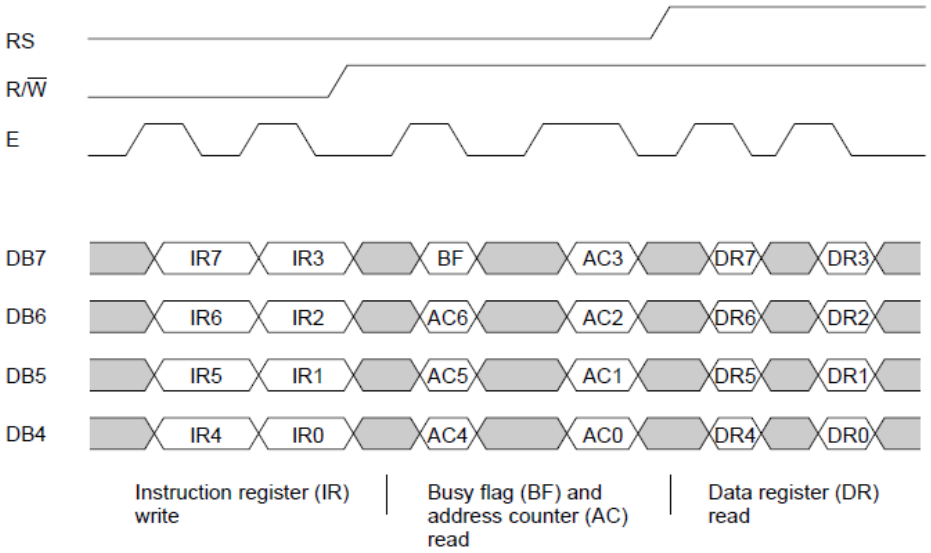
No	Symbol	Function	No	Symbol	Function
1	VSS	GND, 0V	9	D2	Data
2	VDD	+5V	10	D3	Data
3	V0	LCD bias	11	D4	Data
4	RS	Register select	12	D5	Data
5	RW	Read/Write select	13	D6	Data
6	E	Read/Write enable	14	D7	Data
7	D0	Data	15	BLA	Backlighting positive
8	D1	Data	16	BLK	Backlighting negative

Πίνακας 9. Οι ακροδέκτες της οθόνης υγρών κρυστάλλων [31], [32]

Οι ακροδέκτες VSS και VDD υπάρχουν για την παροχή ρεύματος στην οθόνη υγρών κρυστάλλων, ενώ οι BLA και BLK για το φωτισμό της οθόνης μέσω της ενσωματωμένης διόδου LED. Από τον ακροδέκτη V0 με διαιρέτη τάσης ρυθμίζεται η πόλωση των υγρών κρυστάλλων, δηλαδή εάν θα εμφανίζονται έντονα ή αχνά τα γράμματα στην οθόνη [31].

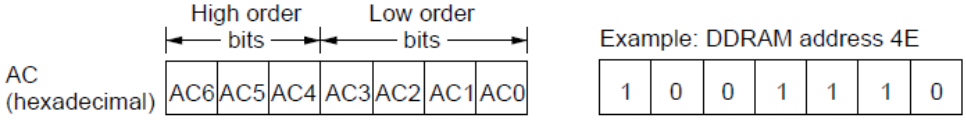
Το ολοκληρωμένο κύκλωμα οδήγησης της LCD διαθέτει δύο καταχωρητές των 8-bit, τον καταχωρητή εντολών IR (Instruction register) και τον καταχωρητή δεδομένων DR (Data register). Από τον ακροδέκτη RS επιλέγεται ο καταχωρητής με τον οποίο θέλει να επικοινωνήσει ο μικροεπεξεργαστής του Arduino. Η επιλογή της ανάγνωσης ή της εγγραφής στον καταχωρητή γίνεται από τον ακροδέκτη RW και η ενεργοποίηση της ανάγνωσης ή της εγγραφής γίνεται από τον ακροδέκτη E [32].

Οι ακροδέκτες από D0 έως D7 είναι για την αμφίδρομη μετάδοση των δεδομένων. Μπορούν να χρησιμοποιηθούν είτε οι 4, είτε και οι 8 γραμμές δεδομένων. Όταν χρησιμοποιείται μετάδοση 4-bit χρησιμοποιούνται οι ακροδέκτες D4 έως D7 και οι D0 έως D3 απενεργοποιούνται. Η μετάδοση 4-bit ολοκληρώνεται όταν αποσταλούν δύο φορές 4 bit δεδομένων [32].



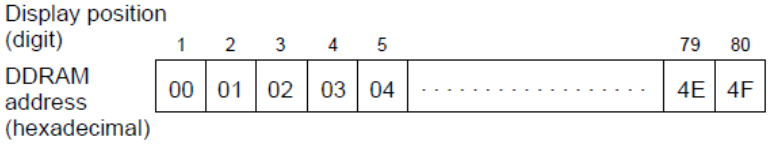
Εικόνα 41. Μετάδοση δεδομένων 4-bit [32]

Η πληροφορία που εμφανίζεται στην οθόνη αποθηκεύεται στη μνήμη DDRAM (Display Data Ram) με χωρητικότητα 80 χαρακτήρων και μέγεθος χαρακτήρα 8bit. Τη θέση κάθε χαρακτήρα στη DDRAM τη δείχνει ο μετρητής διεύθυνσης AC (Address Counter).



Εικόνα 42. Παράδειγμα διεύθυνσης της DDRAM [32]

Εάν υποθέσουμε ότι η οθόνη ήταν μίας γραμμής και 80 χαρακτήρων, ο κάθε χαρακτήρας στην οθόνη θα αντιστοιχούσε στην αμέσως επόμενη θέση από αυτή του προηγούμενου χαρακτήρα στη μνήμη DDRAM [32].

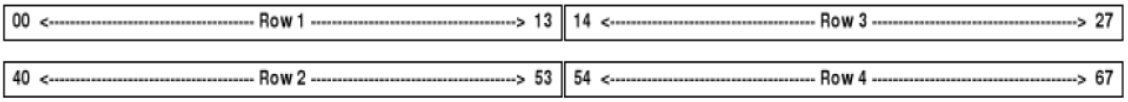


Εικόνα 43. Οθόνη μίας γραμμής [32]

Όμως, σε μία οθόνη 20x4 τα πράγματα δεν συμβαίνουν ακριβώς έτσι. Στη συνέχεια της πρώτης γραμμής βρίσκεται η τρίτη γραμμή και όχι η δεύτερη όπως θα περιμέναμε. Μετά το τέλος της τρίτης γραμμής, διεύθυνση 27h, ο μετρητής διεύθυνσης δείχνει αμέσως τη 40h που είναι η αρχή της δεύτερης γραμμής και στο τέλος ακολουθεί η τέταρτη γραμμή [33].

00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F	10	11	12	13
40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F	50	51	52	53
14	15	16	17	18	19	1A	1B	1C	1D	1E	1F	20	21	22	23	24	25	26	27
54	55	56	57	58	59	5A	5B	5C	5D	5E	5F	60	61	62	63	64	65	66	67

Εικόνα 44. Οι διευθύνσεις των χαρακτήρων σε οθόνη 20x4 [33]

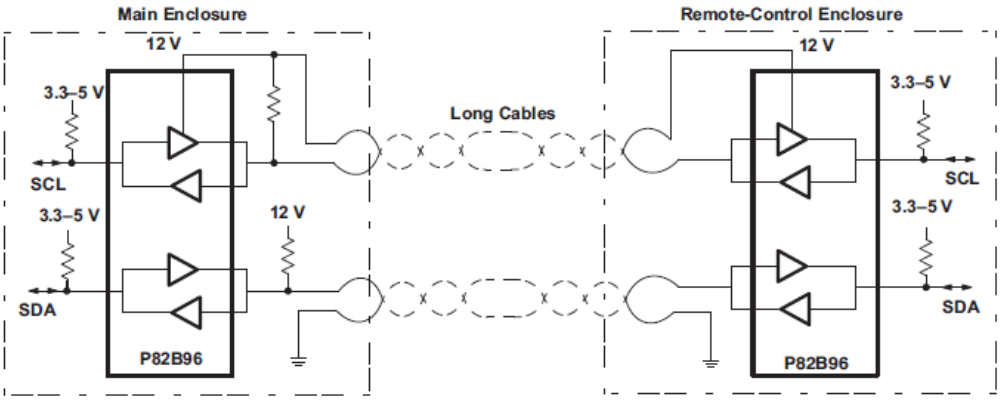


Εικόνα 45. Η χαρτογράφηση των γραμμών στη μνήμη [33]

Ο τρόπος επομένως, που χαρτογραφούνται οι χαρακτήρες της οθόνης στη μνήμη πρέπει να λαμβάνεται υπόψιν κατά τον προγραμματισμό του οποιοδήποτε μικροεπεξεργαστή που πρόκειται να συνδεθεί σε μία οθόνη 20x4.

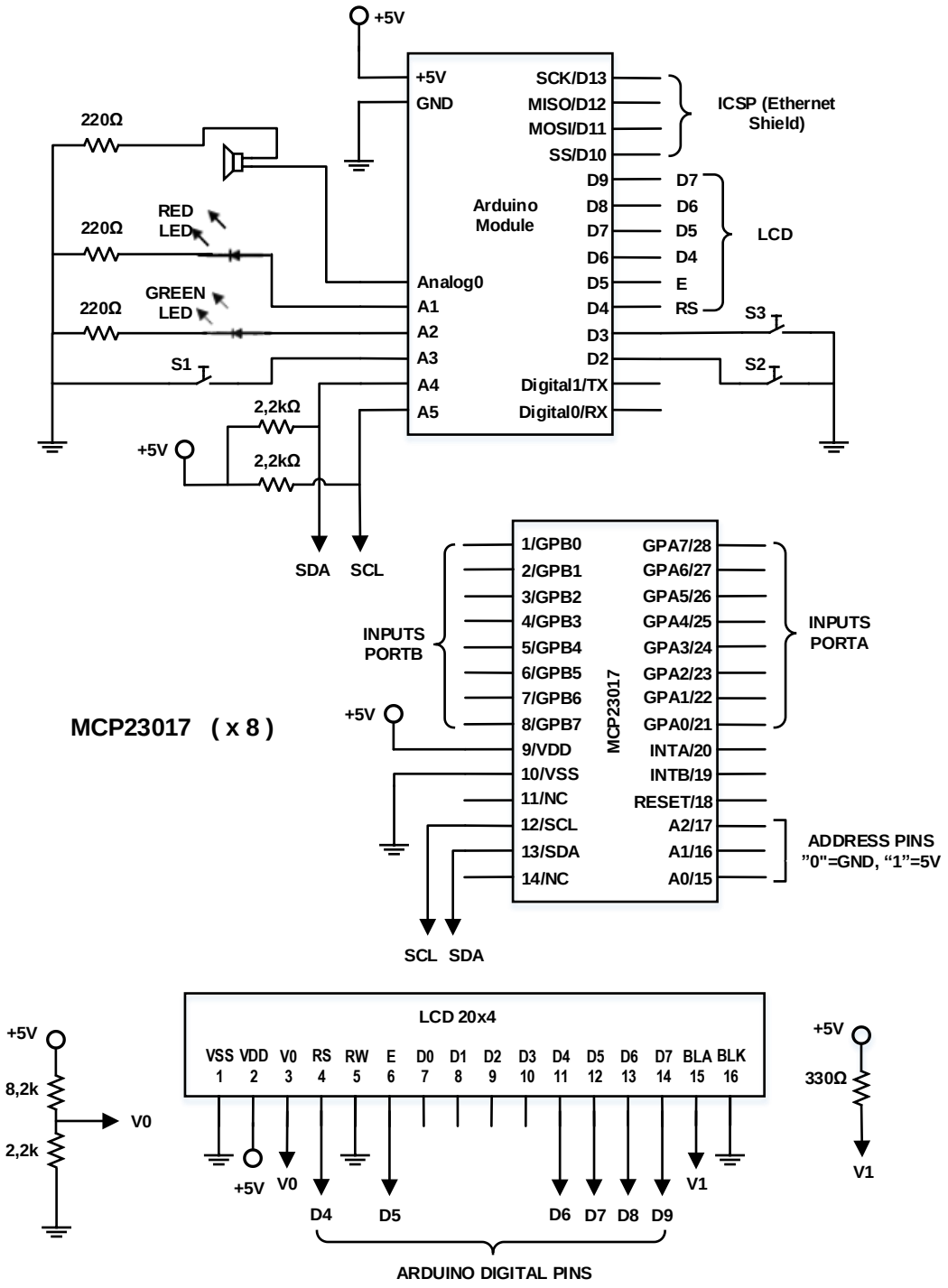
5.5 Ο αμφίδρομος απομονωτής/ενισχυτής P82B96 σημάτων I²C

Προαιρετικά μπορεί να χρησιμοποιηθεί ζεύγος του ολοκληρωμένου κυκλώματος P82B96 για να γίνει επέκταση των καλωδίων μεταφοράς σημάτων I²C. Με αυτόν τον τρόπο το μήκος ενός καλωδίου συνεστραμμένων ζευγών μπορεί να ξεπεράσει τα 20 μέτρα σε λειτουργία Fast mode, με ρυθμό μετάδοσης 400kHz [34].



Εικόνα 46. Ο απομονωτής (buffer) επέκτασης γραμμών μεταφοράς σημάτων I²C [34]

5.6 Η συνδεσμολογία του υλικού της εφαρμογής



Εικόνα 47. Η συνδεσμολογία του υλικού της εφαρμογής

Για την συνδεσμολογία των εξαρτημάτων μπορούν να γίνουν κάποιες επισημάνσεις. Στις γραμμές SCL και SDA συνδέονται οι αντίστοιχοι ακροδέκτες του Arduino καθώς και των 8 ολοκληρωμένων κυκλωμάτων MCP23017. Στις γραμμές SCL και SDA παρατηρούμε επίσης ότι υπάρχουν δύο αντιστάσεις Pull-up, τιμής 2,2kΩ η καθεμία, οι οποίες έχουν σκοπό να θέτουν όλους τους αντίστοιχους ακροδέκτες σε κατάσταση “HIGH” όταν δεν υπάρχει κάποιο άλλο σήμα που να εφαρμόζεται στα άκρα τους. Αυτές οι αντιστάσεις είναι απαραίτητες έτσι ώστε να αποφεύγονται οι καταστάσεις αβεβαιότητας (“0” ή “1”) όταν δεν εφαρμόζεται σήμα στους ακροδέκτες.

Η αντίσταση pull-up δεν πρέπει να έχει πολύ μικρή τιμή γιατί αυξάνει το ρεύμα που καταναλώνεται σε αυτή, αλλά ούτε πολύ μεγάλη τιμή γιατί εισάγεται καθυστέρηση στη μεταβολή της τάσης στον ακροδέκτη, με αποτέλεσμα σε μεγαλύτερες συχνότητες να παραμορφώνεται το σήμα [36].

Ανάλογα με τη συχνότητα Freq των παλμών συγχρονισμού, υπάρχουν δύο μαθηματικές σχέσεις με τις οποίες μπορούμε να υπολογίσουμε τη μικρότερη και τη μεγαλύτερη τιμή μιας αντίστασης pull-up:

- Για Freq < 100kHz: $R_{min} = (V_{CC} - 0,4V) / 3mA$ και $R_{max} = 1000ns / C_b$
- Για Freq > 100kHz: $R_{min} = (V_{CC} - 0,4V) / 3mA$ και $R_{max} = 300ns / C_b$
  όπου V_{CC} είναι η τάση τροφοδοσίας και C_b η συνολική χωρητικότητα του διαύλου [35], [37].

Με χωρητικότητα στους ακροδέκτες I²C του Arduino στα 20pF, στους ακροδέκτες του MC23017 αντίστοιχα 20pF, και με συχνότητα συγχρονισμού στα 100kHz, παίρνουμε ελάχιστη τιμή της αντίστασης pull-up R_{min} = 1,5kΩ και R_{max} = 5kΩ.

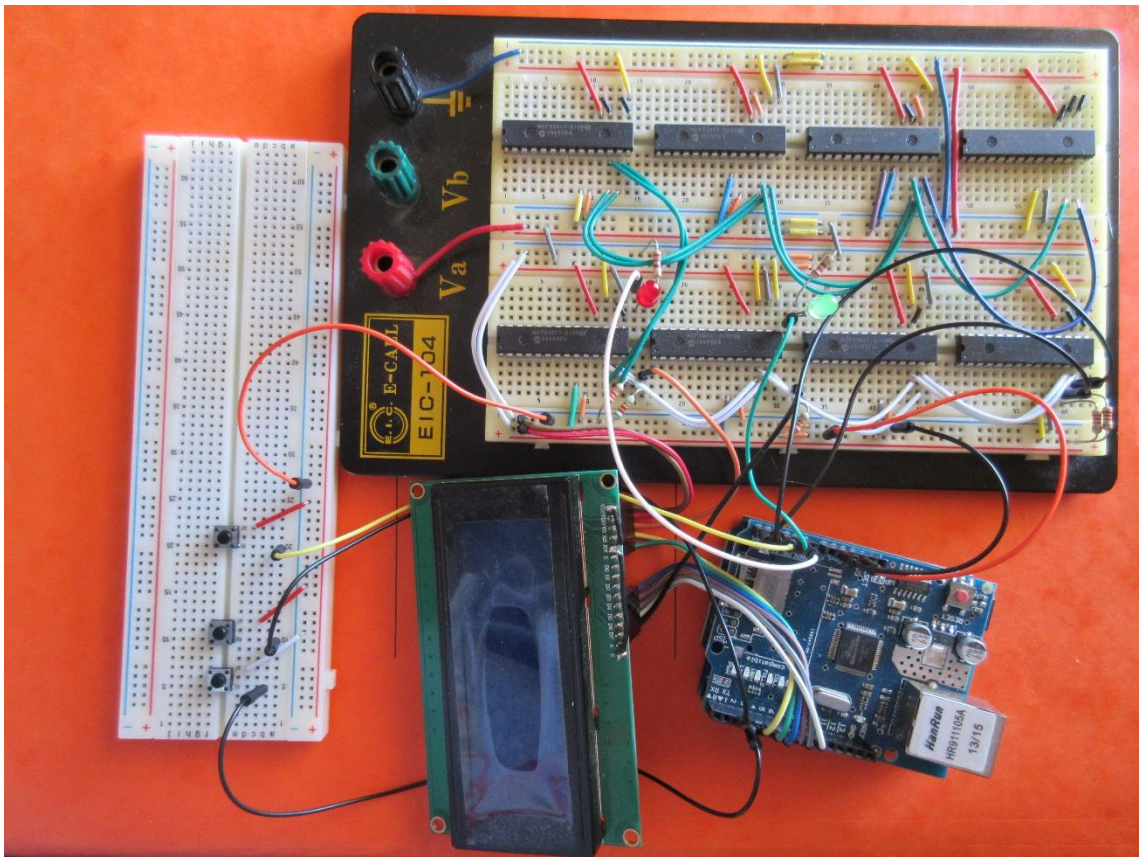
Οι ακροδέκτες D10 έως D13 του Arduino αφορούν το δίαυλο SPI, μέσω του οποίου επικοινωνεί το Arduino με την κάρτα δικτύου Ethernet Shield. Στους ακροδέκτες αυτούς δεν κάνουμε κάποιες συνδέσεις γιατί η σύνδεση SPI πραγματοποιείται μέσω της ακιδοσειράς ICSP στην οποία κουμπώνει η Ethernet Shield.

Για την απεικόνιση των αλφαριθμητικών χαρακτήρων στην οθόνη LCD, χρησιμοποιούμε 4-bit μετάδοση δεδομένων και όχι 8-bit, δηλαδή τους ακροδέκτες D4 έως D7 της οθόνης, έτσι ώστε να περιορίσουμε τον αριθμό των συνδέσεων. Έτσι, οι ακροδέκτες RS, E, D4, .., D7 της οθόνης συνδέονται στους ακροδέκτες D4 έως D9 του Arduino.

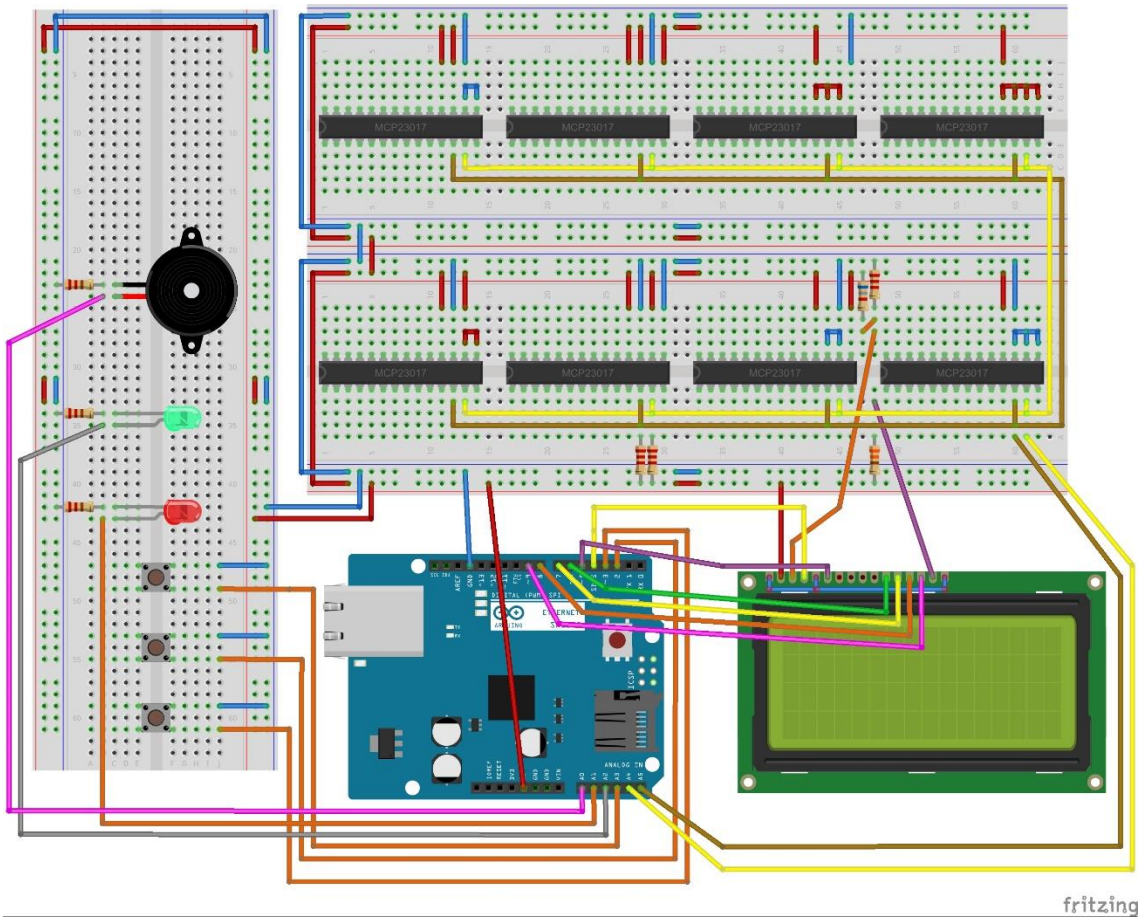
Μέσω των ακροδεκτών τους A0 (low bit), A1 και A2, τα οκτώ MCP23017 παίρνουν τις διευθύνσεις τους, με χαμηλότερη τη 000 και τελευταία την 111. Το bit “0” αντιστοιχεί σε γείωση του ακροδέκτη, ενώ το bit “1” σε σύνδεση του ακροδέκτη στα 5V. Τα οκτώ MCP23017 συνολικά μας δίνουν 8x16, δηλαδή 128 ψηφιακές εισόδους/εξόδους.

Στον ακροδέκτη A0 του Arduino συνδέεται ένας βομβητής (buzzer), στον ακροδέκτη A1 ένα κόκκινο ενδεικτικό LED και στον A2 ένα πράσινο. Στον ακροδέκτη A3 συνδέεται ένας μεταγωγός S1 (push button) και παρόμοια, στον ακροδέκτη D2 ο μεταγωγός S2 και στον D3 ο μεταγωγός S3, με αρχική τους θέση την ανοικτή επαφή.

Στην εικόνα 48 φαίνονται τα εξαρτήματα σε συνδεσμολογία. Μπορούμε να εντοπίσουμε τα οκτώ ολοκληρωμένα MCP23017, την πλακέτα Ethernet Shield με το Arduino Duemilanove συνδεμένο από την κάτω πλευρά της, την οθόνη LCD και τα υπόλοιπα εξαρτήματα, όπως push buttons, ενδεικτικά LED και καλώδια σύνδεσης. Στην εικόνα 49 παρουσιάζεται η συνδεσμολογία όπως πραγματοποιήθηκε με χρήση του λογισμικού fritzing [38].



Εικόνα 48. Φωτογραφία της συνδεσμολογίας του υλικού της εφαρμογής



Εικόνα 49. Συνδεσμολογία υλικού με χρήση του λογισμικού σχεδίασης fritzing

ΚΕΦΑΛΑΙΟ 6 - Το λογισμικό της εφαρμογής τηλεχειρισμού

6.1 Το πρόγραμμα (sketch) της πλατφόρμας Arduino

6.1.1 Περιγραφή και στιγμιότυπα της λειτουργίας

Το πρόγραμμα που γράφτηκε για το Arduino έχει σκοπό την απεικόνιση των συμβάντων σε μία οθόνη υγρών κρυστάλλων (LCD), το χειρισμό της οθόνης (αλλαγή σελίδας σε προηγούμενη και επόμενη), το χειρισμό των συμβάντων (αποδοχή και καθαρισμό των συμβάντων) και την ενεργοποίηση συναγερμών. Ως συμβάν ορίζουμε το κλείσιμο με τη γείωση ενός ακροδέκτη εισόδου των MCP23017. Τα οκτώ MCP23017 παρέχουν 128 εισόδους.

Επίσης δίνεται η δυνατότητα της σύνδεσης του Arduino σε δίκτυο Ethernet, έτσι ώστε να δέχεται εντολές μέσω της γραμμής διευθύνσεων και να απαντά σε γλώσσα XML. Μέσω του περιηγητή μπορούμε να δούμε τις ρυθμίσεις δικτύου της κάρτας Ethernet Shield και να τις αλλάξουμε, να στείλουμε ερωτήματα και να λάβουμε απαντήσεις για την κατάσταση των συμβάντων και των συναγερμών και να κάνουμε αποδοχή ή καθαρισμό των συμβάντων.

Με την εκκίνηση του Arduino εμφανίζονται οι ακόλουθες ενδείξεις (βλέπε εικόνα 50) στην οθόνη υγρών κρυστάλλων.

Εικόνα 50. Αρχική απεικόνιση οθόνης LCD

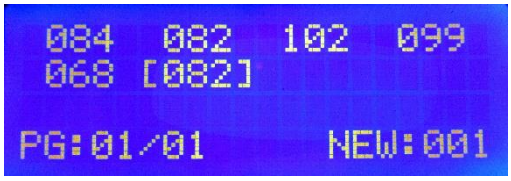
Οι ενδείξεις αυτές ενημερώνουν ότι δεν υπάρχουν συμβάντα, “No events”, ότι εμφανίζεται η πρώτη σελίδα, “PG:01/01” και ότι δεν υπάρχουν νέα συμβάντα, “NEW:000”. Εάν κάποιος ακροδέκτης εισόδου από τα MCP23017 κλείσει κύκλωμα με τη γείωση, τότε ο αριθμός του θα εμφανιστεί ανάμεσα σε αγκύλες στην οθόνη. Ταυτόχρονα θα αναβοσβήνει το κόκκινο ενδεικτικό LED και θα ηχεί η σειρήνα. Στην εικόνα 51 εμφανίζονται 5 νέα συμβάντα και οι αντίστοιχοι αριθμοί τους.

Εικόνα 51. Οθόνη με 5 νέα συμβάντα

Εάν πιέσουμε το κουμπί της αποδοχής, τότε γινόμαστε ενήμεροι των συμβάντων και οι αριθμοί παύουν να εμφανίζονται ανάμεσα σε αγκύλες. Η ένδειξη “NEW:” μηδενίζει. Ταυτόχρονα το κόκκινο LED ανάβει σταθερά και η σειρήνα παύει να ηχεί. Η εικόνα που παρουσιάζει η οθόνη φαίνεται στην εικόνα 52.

Εικόνα 52. Η εικόνα της οθόνης μετά από την αποδοχή των συμβάντων

Το νέο συμβάν θα εμφανιστεί ανάμεσα σε αγκύλες, η ένδειξη “NEW:” θα αυξηθεί κατά ένα συμβάν, το κόκκινο LED θα αναβοσβήνει και η σειρήνα θα ηχεί (βλέπε εικόνα 53).



Εικόνα 53. Νέο συμβάν μετά από την αποδοχή των προηγούμενων

Η κάθε σελίδα της οθόνης μπορεί να εμφανίσει μέχρι 12 συμβάντα. Τα περισσότερα συμβάντα εμφανίζονται σε επιπλέον σελίδες (βλέπε εικόνα 54).

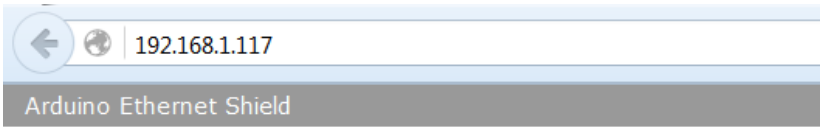
Εικόνα 54. Τα συμβάντα εμφανίζονται σε επιπλέον της μίας σελίδες

Με τα κουμπιά πίσω και μπρος μπορούμε να κινηθούμε μεταξύ των σελίδων (βλέπε εικόνα 55).

Εικόνα 55. Μετακίνηση μεταξύ σελίδων της οθόνης

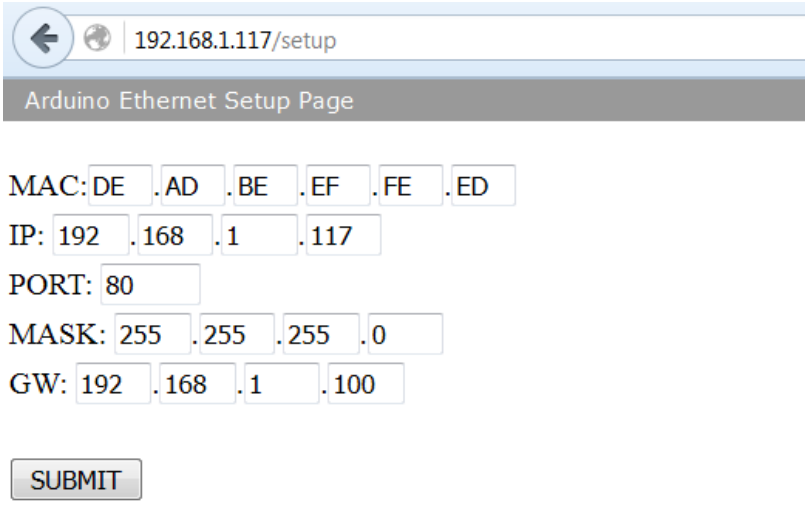
Με την προϋπόθεση ότι έχει γίνει αποδοχή όλων των συμβάντων, μπορούμε να προχωρήσουμε σε καθαρισμό όλων των συμβάντων και να επανέλθει η αρχική εικόνα της οθόνης. Έτσι, εάν πιέσουμε συνεχώς επί τρία δευτερόλεπτα το κουμπί της αποδοχής, για όσο χρόνο πιέζουμε θα ανάβει το πράσινο LED και μετά τα τρία δευτερόλεπτα τα συμβάντα θα σβήσουν.

Το Arduino μπορεί να συνδεθεί σε ένα δίκτυο Ethernet μέσω της Ethernet Shield. Από τον περιηγητή ιστοσελίδων μπορούμε να δώσουμε τη διεύθυνση 192.168.1.117 και θα δούμε να εμφανίζεται η εικόνα 56. Με αυτόν τον τρόπο ενημερωνόμαστε ότι βρισκόμαστε στην αρχική σελίδα της εφαρμογής μας.



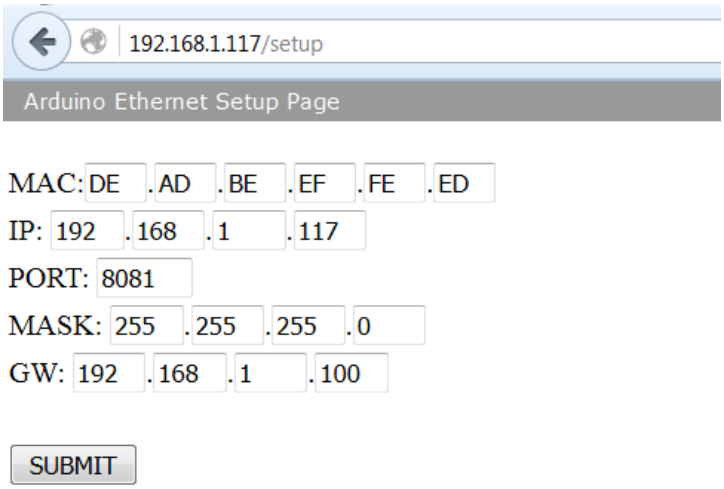
Εικόνα 56. Η αρχική σελίδα της Ethernet Shield

Εάν πληκτρολογήσουμε 192.168.1.117/setup, τότε εμφανίζεται η σελίδα παραμέτρων δικτύου της Ethernet Shield.



Εικόνα 57. Η σελίδα των παραμέτρων δικτύου της Ethernet Shield

Από αυτή τη σελίδα μπορούμε να μεταβάλουμε όλες τις παραμέτρους. Για παράδειγμα αλλάζουμε την παράμετρο “PORT” από “80” σε “8081” και υποβάλουμε την αλλαγή με “SUBMIT” (βλέπε εικόνα 58).

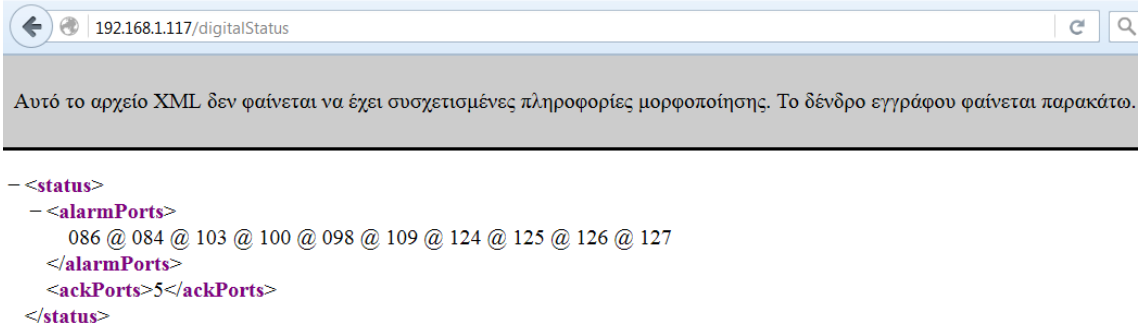


Εικόνα 58. Η αλλαγή της παραμέτρου “PORT”

Για να ελέγξουμε ότι η αλλαγή ήταν επιτυχής, πληκτρολογούμε 192.168.1.117:8081.

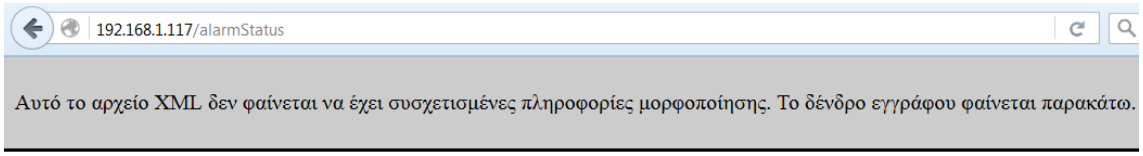
Εικόνα 59. Επιτυχής αλλαγή της παραμέτρου “PORT”

Μπορούμε να ζητήσουμε τα συμβάντα και των αριθμό των συμβάντων που έχουν γνωστοποιηθεί με την εντολή digitalStatus. Η απάντηση είναι σε γλώσσα XML. Στην εικόνα 60 τα συνολικά συμβάντα είναι 10 από τα οποία τα 5 έχουν γίνει γνωστά.



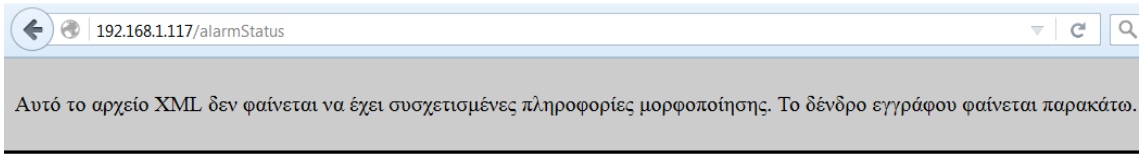
Εικόνα 60. Απάντηση με XML στο ερώτημα “digitalStatus”

Για να ζητήσουμε την κατάσταση των συναγερμών γράφουμε την εντολή alarmStatus. Για την κατάσταση συμβάντων της εικόνας 60 παίρνουμε την απάντηση “Buzzer”, γιατί υπάρχουν νέα συμβάντα στα οποία δεν έχει γίνει αποδοχή με αποτέλεσμα να αναβοσβήνει το κόκκινο LED και να ηχεί η σειρήνα (βλέπε εικόνα 61).



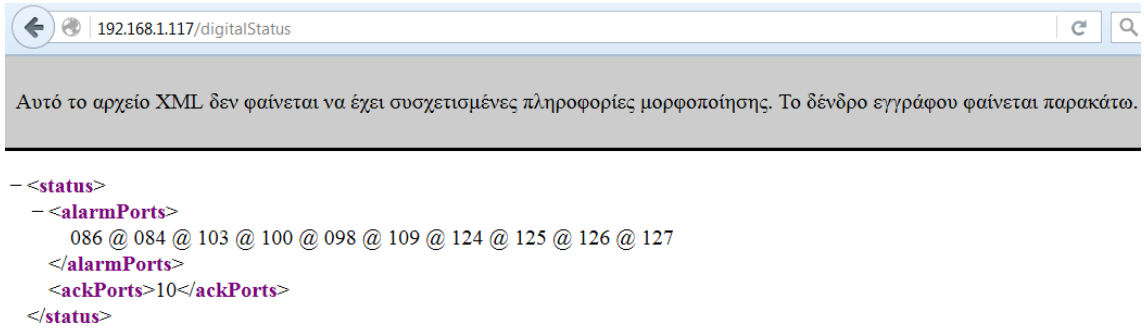
Εικόνα 61. Απάντηση “Buzzer” σε ερώτημα “alarmStatus”

Εάν γίνει η αποδοχή των συμβάντων από το κουμπί αποδοχής, τότε η απάντηση στο ερώτημα alarmStatus θα είναι “ackEvents”, όπως φαίνεται στην εικόνα 62.



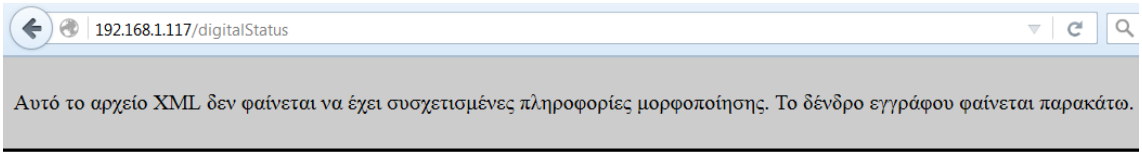
Εικόνα 62. Απάντηση “ackEvents” σε ερώτημα “alarmStatus”

Εάν ζητήσουμε πάλι την κατάσταση των συμβάντων, θα δούμε ότι και τα 10 συμβάντα έχουν γνωστοποιηθεί (βλέπε εικόνα 63).



Εικόνα 63. Απάντηση σε “digitalStatus” όταν όλα τα συμβάντα είναι γνωστά

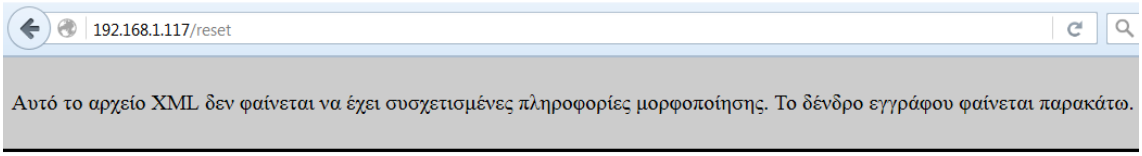
Στην εικόνα 64 εντοπίζουμε ένα νέο συμβάν, τον ακροδέκτη 003.




```
− <status>
  − <alarmPorts>
      086 @ 084 @ 103 @ 100 @ 098 @ 109 @ 124 @ 125 @ 126 @ 127 @ 003
    </alarmPorts>
    <ackPorts>10</ackPorts>
  </status>
```

Εικόνα 64. Νέο συμβάν

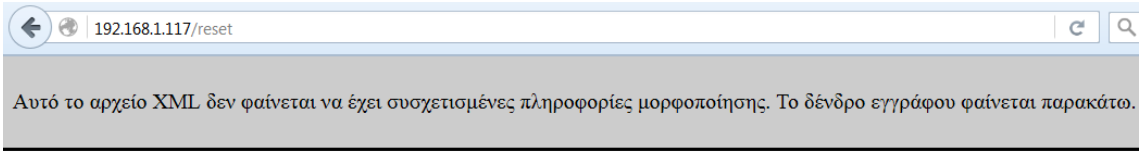
Με την εντολή “reset” γίνεται καθαρισμός των γεγονότων με την προϋπόθεση ότι έχει γίνει η αποδοχή τους. Εάν ζητήσουμε “reset” ενώ υπάρχουν ακόμα νέα γεγονότα, τότε η απάντηση είναι “FALSE”.




```
− <userReset>
    <success> FALSE </success>
  </userReset>
```

Εικόνα 65. Απάντηση “FALSE” σε ερώτημα “reset”

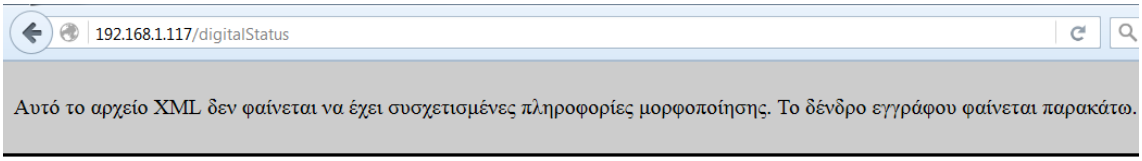
Εάν γίνει αποδοχή όλων των γεγονότων, τότε η εντολή “reset” θα εκτελεσθεί και θα δοθεί απάντηση “TRUE”. Όλα τα συμβάντα τότε θα σβηστούν.




```
− <userReset>
    <success> TRUE </success>
  </userReset>
```

Εικόνα 66. Απάντηση “TRUE” σε ερώτημα “reset”

Η απάντηση στο ερώτημα “digitalStatus” όταν δεν υπάρχουν συμβάντα είναι “noEvents”



Εικόνα 67. Απάντηση “noEvents” σε ερώτημα “digitalStatus”

6.1.2 Επεξήγηση βασικών σημείων του προγράμματος

Παρατίθεται ο κώδικας της βασικής συνάρτησης loop(), ο οποίος εκτελείται κυκλικά συνεχώς για όσο διάστημα τροφοδοτείται η πλακέτα Arduino με ρεύμα και στη συνέχεια ακολουθεί συνοπτική επεξήγησή του [39] – [45].

```
void loop()
{
  EthernetServer server(port);
  EthernetClient client = server.available();
  if (client)
  {
    WaitForRequest(client);
    ParseReceivedRequest();
    PerformRequestedCommands(client);
    client.stop();
  }
  ProcessChip(0);
  ProcessChip(1);
  ProcessChip(2);
  ProcessChip(3);
  ProcessChip(4);
  ProcessChip(5);
  ProcessChip(6);
  ProcessChip(7);
  PrintTable();
  PrintMove();
  AckReset();
  Alarms();
}
```

Στην πρώτη γραμμή της `loop()` δημιουργείται ένας Ethernet server (εξυπηρετητής) που “ακούει” τη συγκεκριμένη θύρα. Στη δεύτερη γραμμή δημιουργείται ένας client (πελάτης) που είναι συνδεμένος στο server στη συγκεκριμένη θύρα και έχει δεδομένα για να στείλει.

Με την προϋπόθεση ότι ο client έχει δεδομένα να στείλει, δηλαδή η συνθήκη if είναι αληθής, εκτελούνται οι συναρτήσεις που περικλείονται από το if. Πρώτα εκτελείται η `WaitForRequest()` η οποία έχει ως σκοπό της την αποθήκευση των χαρακτήρων της γραμμής

διευθύνσεων σε μία μεταβλητή. Μετά, το περιεχόμενο της μεταβλητής κόβεται σε τμήματα από την `ParseReceivedRequest()` με σκοπό να εντοπιστεί η εντολή που δόθηκε από τη γραμμή διευθύνσεων, και να αποθηκευτεί σε μία μεταβλητή. Η εντολή βρίσκεται στη γραμμή διευθύνσεων μετά τους αριθμούς της διεύθυνσης και της θύρας, αφού προηγηθεί ο χαρακτήρας της καθέτου (slash) "\".

Η συνάρτηση `PerformRequestedCommands()` εκτελεί την εντολή που ζητήθηκε μέσω της γραμμής διευθύνσεων, καλώντας τις συναρτήσεις `RemoteDigitalStatus()`, `UserReset()`, `UserAck()`, `UserAlarm()`, `AddressSetup()`, ή `AddressConfirm()`, ανάλογα με το εάν η εντολή που δόθηκε από τη γραμμή διευθύνσεων ήταν: "digitalStatus", "reset", "acknowledge", "alarmStatus", "setup", ή "".

Με τη συνάρτηση `client.stop()` αποσυνδέεται ο client από τον server. Ακολουθεί η συνάρτηση `ProcessChip()`, η οποία εκτελείται για το καθένα από τα οκτώ ολοκληρωμένα MCP23017. Η συνάρτηση αυτή έχει σκοπό να εντοπίσει της ενεργές θύρες A και B των ολοκληρωμένων και πιο συγκεκριμένα, να εντοπίσει ποιος ή ποιοι ακροδέκτες έχουν κλείσει με τη γείωση. Οι ακροδέκτες που ενεργοποιήθηκαν αποθηκεύονται με αύξουσα σειρά σε πίνακα.

Με την `PrintTable()` εμφανίζονται οι αριθμοί των ενεργών ακροδεκτών στην οθόνη. Θεωρούμε κάθε ενεργό ακροδέκτη ως ένα συμβάν. Εάν ο χρήστης δεν έχει λάβει γνώση των συμβάντων, οι αριθμοί εμφανίζονται μέσα σε αγκύλες. Εάν έχει γίνει αποδοχή των συμβάντων, τότε οι αριθμοί εμφανίζονται χωρίς αγκύλες. Οι συμβολοσειρές (strings) "001" έως "128" και "[001]" έως "[128]" είναι τύπου PROGMEM και αποθηκεύονται στη μνήμη Flash για να μην καταλαμβάνουν χώρο στην SRAM. Επίσης τύπου PROGMEM είναι και οι πίνακες δεικτών προς strings στους οποίους περιέχονται τα ανωτέρω strings.

Η `PrintMove()` είναι υπεύθυνη για την εμφάνιση των συμβάντων στην οθόνη όταν μετακινούμε την οθόνη ανά σελίδα μπροστά ή πίσω.

Η `AckReset()` ελέγχει το πάτημα του διακόπτη αποδοχής των συμβάντων. Έτσι, εάν πατηθεί ο διακόπτης θα γίνει αποδοχή των συμβάντων και η PrintMove(), όταν έρθει η σειρά της να εκτελεσθεί, θα εμφανίσει τους αριθμούς των ακροδεκτών χωρίς αγκύλες. Εάν ξαναπατηθεί ο διακόπτης θα ανάψει το πράσινο ενδεικτικό LED για όσο χρόνο πιέζεται ο διακόπτης. Εάν ο διακόπτης πιέζεται για πάνω από τρία δευτερόλεπτα, τότε ο πίνακας των συμβάντων θα σβήσει και το πράσινο ενδεικτικό LED σβήνει.

Η `Alarms()` είναι υπεύθυνη για τη λειτουργία του κόκκινου ενδεικτικού LED και της σειρήνας (buzzer). Όταν συμβεί νέο συμβάν, τότε το κόκκινο LED θα αναβοσβήνει και η σειρήνα θα ηχήσει. Όταν γίνει αποδοχή των συμβάντων το κόκκινο LED θα ανάβει σταθερά και θα σταματήσει να ηχεί η σειρήνα. Όταν ο πίνακας συμβάντων σβήσει, τότε θα σβήσει και το κόκκινο LED.

Οι συναρτήσεις που εκτελούνται όταν αυτό ζητηθεί από τον client μέσω της γραμμής διευθύνσεων, όπως αναφέρθηκε είναι οι ακόλουθες: `RemoteDigitalStatus()`, `UserReset()`, `UserAck()`, `UserAlarm()`, `AddressSetup()`, και `AddressConfirm()`. Η `RemoteDigitalStatus()` απαντά με XML και στέλνει τους αριθμούς των ακροδεκτών που έχουν ενεργοποιηθεί (συμβάντα). Επίσης στέλνει το συνολικό αριθμό των συμβάντων στα οποία έχει γίνει αποδοχή. Η `UserReset()` εκτελεί επανεκκίνηση της εφαρμογής, δηλαδή καθαρισμό των συμβάντων και της οθόνης LCD. Εάν πραγματοποιηθεί η επανεκκίνηση, τότε απαντά με XML τη λέξη "TRUE", διαφορετικά στέλνει "FALSE". Η `UserAck()` εκτελεί την αποδοχή των συμβάντων. Εάν η αποδοχή πραγματοποιηθεί, τότε στέλνει απάντηση XML προς τον client τη λέξη "TRUE", διαφορετικά απαντά "FALSE". Η `UserAlarm()` απαντά με τη λέξη "Buzzer" εάν έχει ενεργοποιηθεί η σειρήνα και με τη λέξη "ackEvents" εάν έχει γίνει αποδοχή των συμβάντων. Η `AddressConfirm()` ενημερώνει με HTML και JavaScript με το κείμενο "Arduino Ethernet Shield" ότι εντοπίσαμε από την εφαρμογή περιήγησης τη διεύθυνση IP και τη θύρα του Arduino Ethernet Shield.

Στα επόμενα αναφερόμαστε στη συνάρτηση `AddressSetup()` πιο αναλυτικά. Αρχικά εμφανίζεται στην οθόνη του περιηγητή ιστοσελίδων η πληροφορία HTML και JavaScript που έστειλε το Arduino στον client με ονομασία “Arduino Ethernet Setup Page”. Η πληροφορία αυτή αφορά τις τρέχουσες ρυθμίσεις δικτύου της Ethernet Shield που είναι η διεύθυνση MAC, η διεύθυνση IP, η μάσκα υποδικτύου MASK, η διεύθυνση πύλης GATEWAY, και η θύρα PORT. Η πληροφορία HTML και JavaScript είναι γραμμένη στη μνήμη Flash για να μη δεσμεύουμε το χώρο της μνήμης SRAM, η οποία έχει περιορισμένη χωρητικότητα, όπως επίσης και για το λόγο ότι η συγκεκριμένη πληροφορία δεν μεταβάλλεται κατά τη λειτουργία της εφαρμογής. Οι παράμετροι που αποθηκεύονται στη Flash είναι strings τύπου PROGMEM και πίνακες δεικτών προς strings τύπου PROGMEM. Μέσα από τη σελίδα αυτή ο χρήστης έχει τη δυνατότητα να αλλάξει τις τιμές MAC, IP, MASK, GATEWAY και PORT, των οποίων οι αρχικές τιμές ορίζονται στην αρχή του sketch ως μεταβλητές τύπου byte. Οι καινούργιες τιμές αυτών των μεταβλητών υποβάλλονται από τη σελίδα αυτή πατώντας το κουμπί “SUBMIT” και οι οποίες τιμές γράφονται μέσω της `EEPROM.write()` της βιβλιοθήκης EEPROM.h στη μνήμη EEPROM του Arduino. Χρησιμοποιούμε τη μνήμη EEPROM για αποθήκευση μεταβλητών, οι οποίες θέλουμε να παραμένουν μετά τη διακοπή της παροχής ρεύματος στο Arduino. Μετά το “SUBMIT” θα χρησιμοποιούνται οι νέες τιμές των παραμέτρων αυτών για τη σύνδεση στο δίκτυο του Arduino και οι οποίες θα εξακολουθούν να ισχύουν μετά από επανεκκίνηση ή διακοπή της τροφοδοσίας του Arduino μετά από εκτέλεση της `ShieldSetup()`.

Παρατίθεται ο κώδικας της βασικής συνάρτησης setup(), η οποία εκτελείται μία μόνο φορά κατά την εκκίνηση του Arduino και έχει σκοπό την αρχικοποίηση των ακροδεκτών του Arduino, ορισμένων μεταβλητών, τη χρήση των βιβλιοθηκών και γενικότερα ότι σχετίζεται με την αρχική κατάσταση του Arduino.

```
void setup()
{
  ShieldSetup(); //Setup the Ethernet shield
  EthernetServer server(port);
  server.begin();
  Serial.begin(9600);
  pinMode(outputAlarmBuzzer, OUTPUT);
  pinMode(outputAlarmLed, OUTPUT);
  pinMode(outputResetLed, OUTPUT);
  pinMode(inputAckReset, INPUT_PULLUP);
  pinMode(inputPrev, INPUT_PULLUP);
  pinMode(inputNext, INPUT_PULLUP);
  lcd.begin(20, 4);  // set up the LCD's number of columns and rows
  LCD_Start();
  Wire.begin();
  MCPmode(0);
  MCPmode(1);
  MCPmode(2);
  MCPmode(3);
  MCPmode(4);
  MCPmode(5);
  MCPmode(6);
  MCPmode(7);
}
```

Σε αυτή τη συνάρτηση μπορούμε να επισημάνουμε την αρχικοποίηση της Ethernet Shield με τη συνάρτηση `ShieldSetup()`, και την εκκίνηση του server. Επίσης ορίζονται οι ακροδέκτες του

Arduino ως είσοδοι ή έξοδοι, γίνεται εκκίνηση της οθόνης LCD με την `lcd.begin` και τη συνάρτηση `LCD_Start()` και ενεργοποιείται η σύνδεση I²C μέσω της βιβλιοθήκης `WIRE.h` και της `Wire.begin()`.

Παρατηρούμε ότι καλείται η συνάρτηση `MCPmode()` για καθένα από τα 8 ολοκληρωμένα MCP23017.

```
void MCPmode(byte opcode)
{
  byte controlbyte  = B0100000 | opcode;
  MCPstore (controlbyte,iodira,0xffff);            // all input
  MCPstore (controlbyte,gppua,0xffff);             // all pullups on
}
```

Με τη συνάρτηση MCPmode() γίνεται αρχικοποίηση των MCP23017. Για παράδειγμα, το δεύτερο κατά σειρά ολοκληρωμένο έχει τη διεύθυνση 001 (opcode), αλλά για να γίνει η επικοινωνία μαζί του μέσω του I²C παίρνει τη διεύθυνση 0100001. Ύστερα μέσω της `MCPstore()` γράφονται οι καταχωρητές IODIRA και GPPUA με το byte FFFF, δηλαδή με 11111111 στο δυαδικό. Αυτό σημαίνει, ότι ορίζονται και οι 8 ακροδέκτες της θύρας A ως είσοδοι, ενώ επίσης ορίζεται η αρχική τους κατάσταση το "1" με τη σύνδεσή τους εσωτερικά σε αντιστάσεις pullup. Επειδή όμως το MCP23017 βρίσκεται σε λειτουργία 16-bit από προεπιλογή, το ίδιο με τη θύρα A γίνεται και στη θύρα B, δηλαδή γράφονται και οι καταχωρητές IODIRB και GPPUB, διότι οι αντίστοιχοι καταχωρητές 8-bit της θύρας A και θύρας B ανά δύο αντιμετωπίζονται ως ένας καταχωρητής 16-bit. Ο κώδικας της `MCPstore()` είναι ο ακόλουθος:

```
void MCPstore (byte i2caddress, byte i2cregister, word i2cdata)    //
Store 16 bits in specified register of MCP23017
{
  byte controlbyte  = B0100000 | i2caddress;    // note that bit 0 not
used b/c 7-bit opcode for Wire library
  Wire.beginTransmission(controlbyte);
  Wire.write(i2cregister);
  Wire.write(loByte(i2cdata));
  Wire.write(hiByte(i2cdata));
  Wire.endTransmission();
}
```

Παρατηρούμε, ότι στο συγκεκριμένο καταχωρητή που θέλουμε να γράψουμε, επειδή το ολοκληρωμένο βρίσκεται σε λειτουργία 16-bit, γράφεται πρώτα το LOW Byte που αφορά τον καταχωρητή της θύρας A και ύστερα γράφεται το HIGH Byte στον αντίστοιχο καταχωρητή της θύρας B.

Παραθέτουμε τον κώδικα της `MCPread()`, η οποία εξυπηρετεί την ανάγνωση των καταστάσεων των θυρών των MCP23017:

```
word MCPread (byte i2caddress, byte i2cregister)           // Read 16
bits from MCP23017
{
  byte controlbyte  = B0100000 | i2caddress;               // note
that bit 0 not used b/c 7-bit opcode for Wire library
  word tempdata;
  Wire.beginTransmission(controlbyte);
```

```
  Wire.write(i2cregister);
  Wire.endTransmission();
  Wire.requestFrom(controlbyte,(byte)2);                    //
otherwise function gets one byte and one int
  tempdata = Wire.read();                          // get lower 8 bytes
  tempdata |= (Wire.read() << 8);                  // get upper 8 bytes
  return tempdata;
}
```

Μπορούμε να δούμε τη διαδικασία ανάγνωσης μέσω πρωτοκόλλου I²C. Αρχικά, ξεκινά η μετάδοση προς τη διεύθυνση που αντιστοιχεί στο ζητούμενο MCP23017. Στη συνέχεια, δηλώνεται ο καταχωρητής από τον οποίο θα ζητηθεί η πληροφορία και ύστερα ακολουθεί η ανάγνωση δύο bytes από τον καταχωρητή. Η `MCPread()` θα κληθεί από την `ProcessChip()` για την ανάγνωση των καταχωρητών GPIO (ζεύγος A και B) κατά των έλεγχο της κατάστασης των ακροδεκτών εισόδου των MCP23017.

Ακολουθεί τμήμα του κώδικα της `ProcessChip()`:

```
byte ProcessChip(byte chip)
{
  word mcpdata = MCPread(chip,gpioa);
  byte inputsA = ~loByte(mcpdata);
  byte inputsB= ~hiByte(mcpdata);
  if (inputsA>0 || inputsB>0){
    if (inputsA>0){
      if (inputsA & B00000001)
      {
        byte port=chip*16;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
```

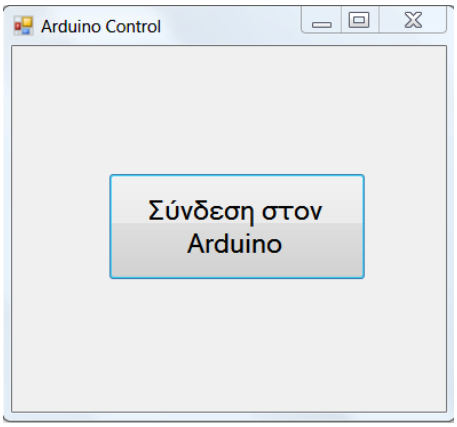
Αρχικά διαβάζεται ο 16-bit καταχωρητής GPIO του συγκεκριμένου MCP23017. Αποθηκεύονται σε αντίστοιχες μεταβλητές το low byte, που αντιστοιχεί στο περιεχόμενο του GPIOA και το high byte, που αντιστοιχεί στο περιεχόμενο του GPIOB. Παρατηρούμε ότι χρησιμοποιείται ο τελεστής “~” έτσι ώστε να αποθηκεύονται τα bytes με τα αντίθετα ψηφία τους, δηλαδή τα “0” γίνονται “1” και αντίστροφα. Αυτό γιατί η αρχική κατάσταση των ακροδεκτών είναι η “1” και όχι η “0”. Με την προϋπόθεση ότι υπάρχει αλλαγή στον καταχωρητή A (πρώτη και δεύτερη συνθήκη `if`), συγκρίνεται χωριστά το καθένα από τα 8 bit για μεταβολή από “0” σε “1”. Το τρίτο `if` δηλαδή, επαναλαμβάνεται 8 φορές, από “00000001” έως “10000000”. Το ίδιο πραγματοποιείται και για τον καταχωρητή B με την προϋπόθεση ότι έχει υπάρξει αλλαγή στα περιεχόμενα του καταχωρητή.

6.2 Το λογισμικό του απομακρυσμένου χρήστη

6.2.1 Περιγραφή και στιγμιότυπα της λειτουργίας

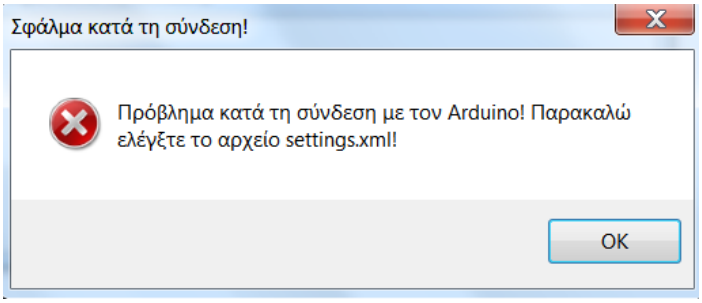
Για τις ανάγκες παρουσίασης της εφαρμογής κατασκευάστηκε πρόγραμμα σε C# με τη βοήθεια του Visual Studio Ultimate 2012. Η βασική του λειτουργία είναι η δημιουργία client για την αποστολή ερωτημάτων μέσω της γραμμής διευθύνσεων και τη λήψη απαντήσεων μέσω κειμένου XML.

Αρχικά εμφανίζεται παράθυρο για να επιλέξουμε να συνδεθούμε με το server του Arduino



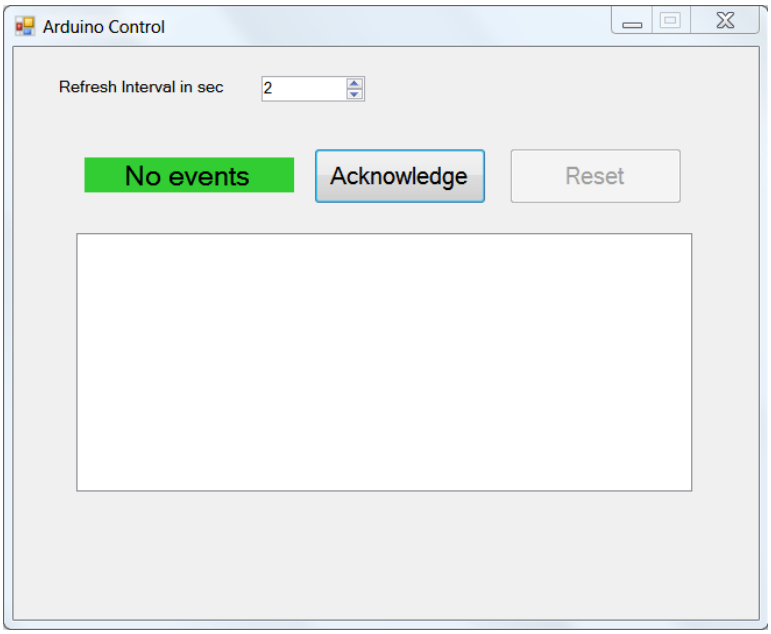
Εικόνα 68. Αρχικό παράθυρο για σύνδεση με το server του Arduino

Στην περίπτωση που δεν μπορεί να γίνει η σύνδεση εμφανίζεται παράθυρο με μήνυμα σφάλματος.



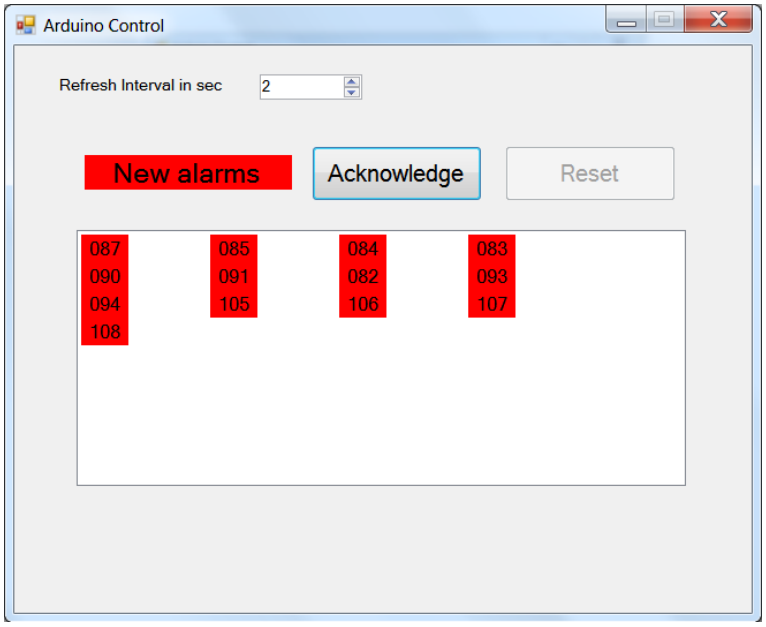
Εικόνα 69. Μήνυμα σφάλματος κατά τη σύνδεση με το server

Όταν επιτευχθεί η σύνδεση εμφανίζεται το βασικό παράθυρο της εφαρμογής του χρήστη. Από τη στιγμή που δεν υπάρχουν συμβάντα εμφανίζεται η ένδειξη σε πράσινο φόντο “No events”. Επίσης υπάρχει το κουμπί “Acknowledge” για την αποδοχή των νέων συμβάντων, καθώς και το κουμπί “Reset” για τη διαγραφή των συμβάντων, το οποίο είναι ανενεργό. Επιπλέον, μπορούμε να διακρίνουμε το πεδίο ρύθμισης του ρυθμού ανανέωσης της πληροφορίας, η οποία εμφανίζεται στο αντίστοιχο πλαίσιο.



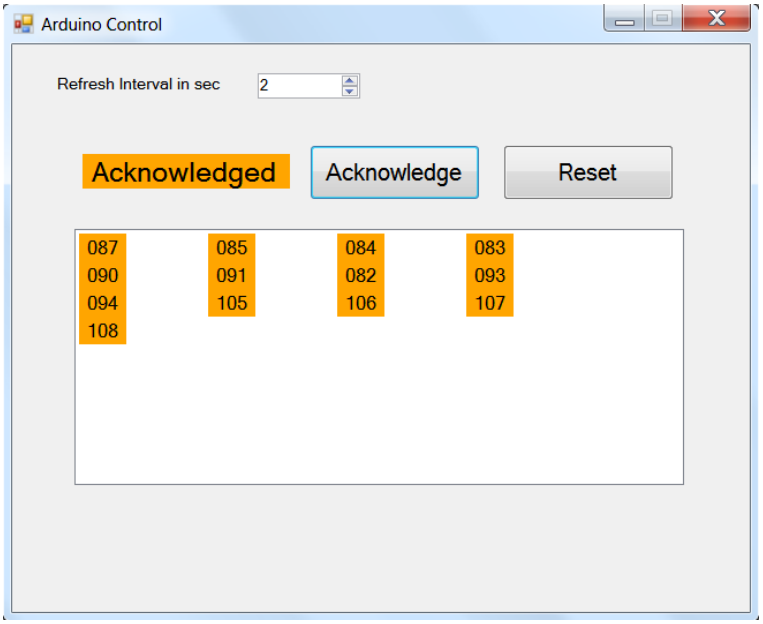
Εικόνα 70. Το βασικό παράθυρο της εφαρμογής όταν δεν υπάρχουν συμβάντα

Στην περίπτωση που υπάρξουν νέα συμβάντα, οι αριθμοί τους εμφανίζονται με κόκκινο χρώμα στο αντίστοιχο πλαίσιο, ενώ εμφανίζεται σε κόκκινο φόντο η ένδειξη “New alarms”.



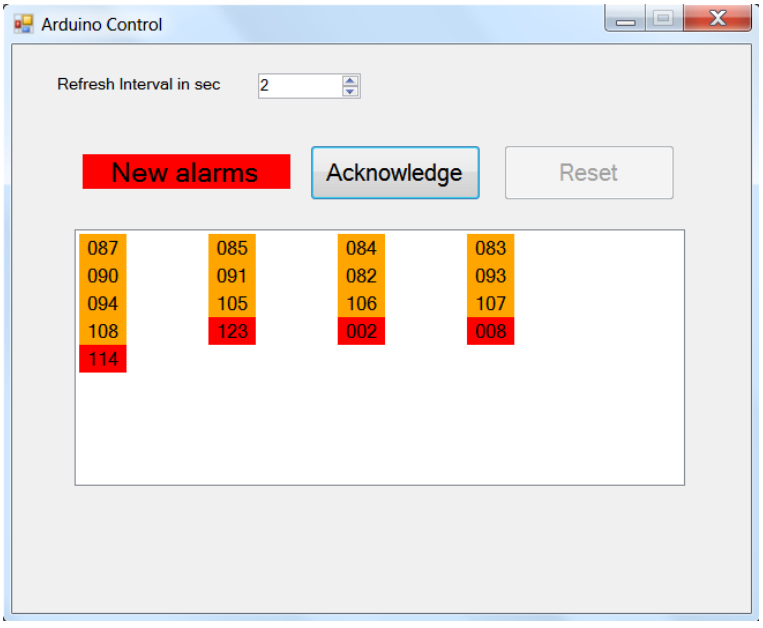
Εικόνα 71. Το παράθυρο της εφαρμογής με νέα συμβάντα

Όταν πατήσουμε το πλήκτρο “Acknowledge”, τότε οι αριθμοί των συμβάντων εμφανίζονται σε πορτοκαλί φόντο, εμφανίζεται η ένδειξη “Acknowledged” κι επίσης το πλήκτρο “Reset” έχει ενεργοποιηθεί.



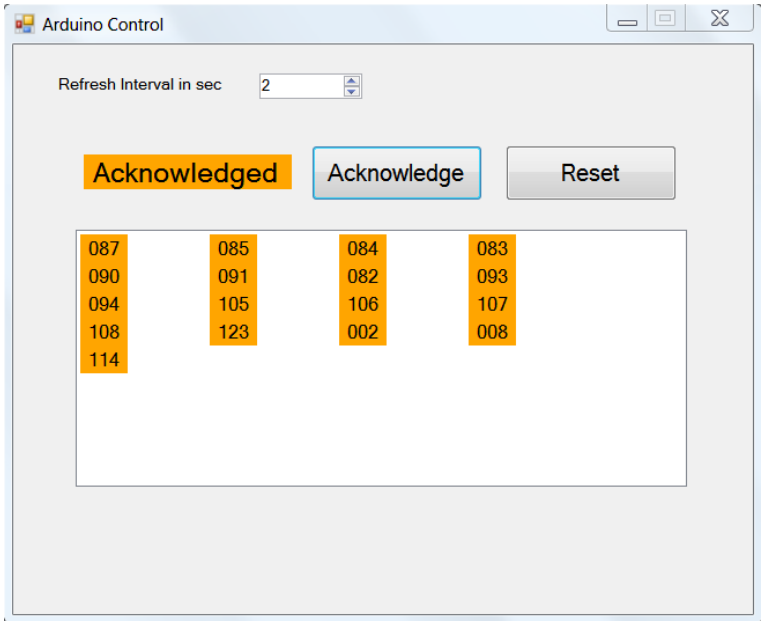
Εικόνα 72. Αποδοχή των συμβάντων

Σε περίπτωση νέων συμβάντων, οι καινούριοι αριθμοί εμφανίζονται σε κόκκινο φόντο και αναγράφεται η ένδειξη “New alarms”.



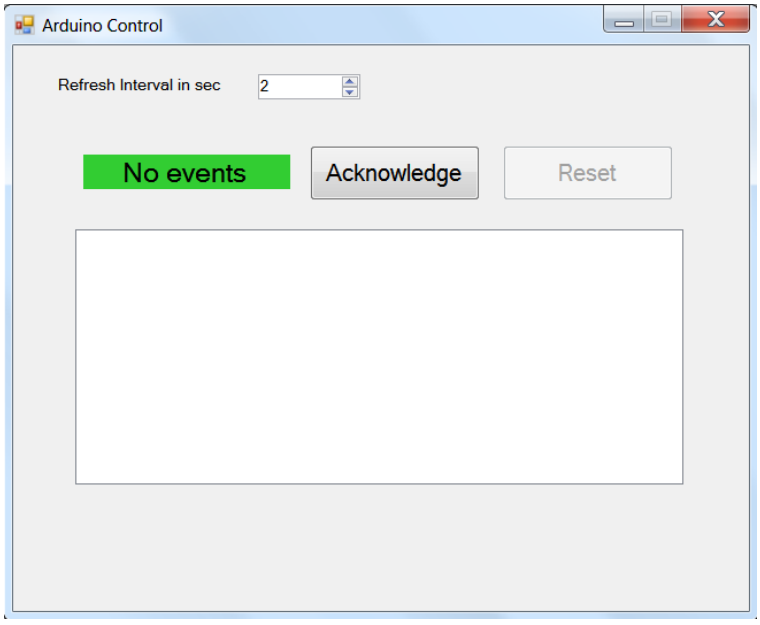
Εικόνα 73. Εμφάνιση νέων συμβάντων

Για να γίνει καθαρισμός των συμβάντων πρέπει πρώτα να πραγματοποιηθεί η αποδοχή τους, έτσι ώστε να ενεργοποιηθεί η δυνατότητα “Reset”.



Εικόνα 74. Αποδοχή των συμβάντων και ενεργοποίηση του “Reset”

Με το πάτημα του “Reset” τα συμβάντα διαγράφονται.



Εικόνα 75. Διαγραφή των συμβάντων μετά από “Reset”

6.2.2 Επεξήγηση βασικών σημείων του προγράμματος

Η βασική συνάρτηση του προγράμματος είναι η `refreshStatus()`, η οποία έχει ως σκοπό να διαβάζει τις απαντήσεις XML από το server του Arduino και να τις τοποθετεί στο παράθυρο της εφαρμογής. Οι απαντήσεις αυτές αφορούν τους αριθμούς και την κατάσταση των συμβάντων, καθώς και την κατάσταση των συναγερμών. Οι αριθμοί των συμβάντων τοποθετούνται σε αντικείμενο λίστας `ListViewItem()` και ανάλογα με το εάν έχει γίνει ή όχι η αποδοχή τους εμφανίζονται με πορτοκαλί ή κόκκινο χρώμα. Στο παράθυρο της φόρμας υπάρχει επίσης ένα `label` το οποίο εμφανίζει διαφορετικές ενδείξεις ανάλογα με την κατάσταση των συμβάντων και των συναγερμών. Εάν υπάρχουν νέα γεγονότα εμφανίζεται η κόκκινη ένδειξη “New alarms”, εάν έχει γίνει αποδοχή τους η πορτοκαλί ένδειξη “Acknowledged” και εάν δεν υπάρχουν γεγονότα η πράσινη “No events”. Εάν έχει γίνει αποδοχή των συμβάντων ενεργοποιείται το `button2` για να μπορεί να πραγματοποιηθεί η διαγραφή των συμβάντων.

Από τη `refreshStatus()` αρχικά καλείται η `readArduinoStatus()`, η οποία έχει ως σκοπό την ανάγνωση της κατάστασης των συναγερμών και την ανάγνωση των συμβάντων. Πρώτα καλεί την `alarmStatus()`, η οποία επιστρέφει μία μεταβλητή τύπου `String` με την κατάσταση των συναγερμών. Για να το πετύχει αυτό, στέλνει ερώτημα μέσω URL στο server του Arduino και επεξεργάζεται την απάντηση XML. Ύστερα καλεί την `getPortNamesList()`, η οποία με τον ίδιο τρόπο τοποθετεί σε πίνακα `String` τους αριθμούς των συμβάντων, καθώς και σε μία μεταβλητή τύπου `Int` το συνολικό αριθμό των συμβάντων που έχουν γνωστοποιηθεί.

Όταν πατηθεί το κουμπί “Acknowledge” καλείται η συνάρτηση `acknowledge()` και η `refreshStatus()` και παρόμοια όταν πατηθεί το κουμπί “Reset” καλείται η `reset()` και η `refreshStatus()`. Οι συναρτήσεις `acknowledge()` και `reset()` στέλνουν εντολή μέσω URL στο Arduino, έτσι ώστε να πραγματοποιήσει τις λειτουργίες της αποδοχής και διαγραφής των συμβάντων αντίστοιχα. Η `refreshStatus()` καλείται για να εμφανιστεί στο παράθυρο της εφαρμογής η νέα κατάσταση συμβάντων και συναγερμών που προέκυψε μετά την πραγματοποίηση των ανωτέρω ενεργειών.

ΚΕΦΑΛΑΙΟ 7 - Συμπεράσματα και Μελλοντικές Κατευθύνσεις

Στα πλαίσια της διατριβής αυτής παρουσιάστηκαν δύο πρωτόκολλα σειριακής επικοινωνίας, το I^2C (Inter-Integrated Circuit bus) και το SPI (Serial Peripheral Interface), τα οποία χρησιμοποιούνται για την επικοινωνία ενός μικροεπεξεργαστή με τις περιφερειακές του συσκευές. Αντικειμενικός σκοπός ήταν πρώτα να μελετηθεί θεωρητικά η λειτουργία τους και στη συνέχεια να εφαρμοστούν σε ηλεκτρονικό σύστημα για τη μετάδοση της πληροφορίας.

Η εφαρμογή πραγματοποιήθηκε με τη χρησιμοποίηση της ηλεκτρονικής πλατφόρμας προτυποποίησης Arduino Duemilanove, το ολοκληρωμένο κύκλωμα επέκτασης εισόδων/εξόδων MCP23017 και την κάρτα δικτύου Arduino Ethernet Shield. Το Arduino Duemilanove παρείχε στην εφαρμογή μας το μικροεπεξεργαστή ATmega328, ενώ το MCP23017 και η Arduino Ethernet Shield αποτέλεσαν τις περιφερειακές συσκευές του μικροεπεξεργαστή με τις οποίες έπρεπε να επικοινωνήσει μέσω των διαύλων I^2C και SPI αντίστοιχα.

Τα οκτώ ολοκληρωμένα MCP23017 έδωσαν τη δυνατότητα να δοκιμαστεί πλήρως η διαδικασία διευθυνσιοδότησης στο δίαυλο I^2C με κατάλληλο προγραμματισμό του Arduino και οι συνολικά 128 ακροδέκτες τους αντιστοιχήθηκαν επιτυχώς σε 128 ψηφιακές εισόδους για τις ανάγκες της εφαρμογής μας. Για την απεικόνιση των καταστάσεων των εισόδων χρησιμοποιήθηκε οθόνη υγρών κρυστάλλων 20 στηλών και 4 γραμμών. Ως συχνότητα των παλμών συγχρονισμού του διαύλου χρησιμοποιήθηκε η προεπιλεγμένη συχνότητα του Arduino για το δίαυλο, τα 100kHz.

Τα αποτελέσματα χρήσης του διαύλου I^2C ήταν ικανοποιητικά, καθώς οι καταστάσεις των ακροδεκτών απεικονίστηκαν αποτελεσματικά στην οθόνη. Επίσης η συχνότητα των παλμών συγχρονισμού του διαύλου ήταν ικανοποιητική, καθώς η μεταβολή στην κατάσταση κάθε ακροδέκτη απεικονιζόταν άμεσα στην οθόνη. Οι συνδέσεις του διαύλου με τα δύο καλώδια έκαναν απλή τη συνδεσμολόγηση του ηλεκτρονικού κυκλώματος και προσέφεραν σταθερή και αξιόπιστη μεταφορά της πληροφορίας.

Με τη χρήση της Arduino Ethernet Shield δοκιμάστηκε η δυνατότητα επικοινωνίας με το μικροεπεξεργαστή μέσω του διαύλου SPI. Μετά από κατάλληλο προγραμματισμό του Arduino, η Ethernet Shield αξιοποιήθηκε και παρείχε χειρισμό και εποπτεία των 128 ακροδεκτών μέσω δικτύου Ethernet. Με ερωτήματα από τη γραμμή διευθύνσεων ενός περιηγητή ιστοσελίδων και απαντήσεις σε μορφή XML από το server του Arduino πραγματοποιήθηκε επιτυχώς ο τηλεχειρισμός των ακροδεκτών της εφαρμογής μας. Επιπρόσθετα, κατασκευάστηκε πρόγραμμα σε γλώσσα C# για την καλύτερη παρουσίαση της δυνατότητας τηλεχειρισμού των ακροδεκτών από προσωπικό υπολογιστή μέσω δικτύου Ethernet.

Ο δίαυλος I^2C μας έδωσε το σημαντικό πλεονέκτημα της ευκολίας στη συνδεσμολογία, ενώ για εφαρμογές που δεν απαιτούνται υψηλοί ρυθμοί μετάδοσης δεδομένων αξιολογείται ως η καλύτερη πρόταση εκ των δύο σειριακής επικοινωνίας. Με την προαιρετική χρήση ζεύγους απομονωτών/ενισχυτών I^2C, τα δύο καλώδια που απαιτούνται μπορούν να επεκταθούν τουλάχιστον στα 20 μέτρα για ταχύτητες μετάδοσης μέχρι 400kHz. Όταν όμως απαιτούνται ταχύτερες μεταφορές δεδομένων, όπως στην περίπτωση σύνδεσης μιας κάρτας δικτύου, προτείνεται ο δίαυλος SPI, με την προϋπόθεση ότι οι συσκευές που συνδέονται εντοπίζονται στο χώρο του τυπωμένου κυκλώματος ή σε κοντινές μεταξύ τους αποστάσεις.

Η παρούσα εφαρμογή τηλεχειρισμού χρησιμοποιεί υλικά, τα οποία δεν αποτελούν την καλύτερη πρόταση για μία μόνιμη εγκατάσταση και εξυπηρετούν κυρίως το σκοπό της ανάπτυξης εφαρμογών και της επίδειξης. Η πλακέτα Arduino Duemilanove, όπως και η πλακέτα επέκτασης Arduino Ethernet Shield, έχουν αρκετά μεγάλες διαστάσεις και καταναλώνουν αρκετή ενέργεια για να χρησιμοποιηθούν σε κάποια μόνιμη εγκατάσταση. Αντί αυτών θα μπορούσε να χρησιμοποιηθεί η πλακέτα Arduino Pro Mini που έχει τα ίδια χαρακτηριστικά με την Duemilanove, αλλά όμως είναι αρκετά μικρότερη σε μέγεθος, όπως και η πλακέτα επέκτασης ENC28J60 Mini Ethernet Module σε αντικατάσταση της Ethernet Shield [46], [47].

Για την επικοινωνία του Arduino με τον προσωπικό υπολογιστή χρησιμοποιήθηκε το μοντέλο web server με web client. Το βασικό μειονέκτημα της επικοινωνίας αυτής είναι το ότι πρέπει να ερωτηθεί ο web server για να αποστείλει τα νέα δεδομένα προς τον client. Το πρόβλημα αυτό επιλύεται με τη χρήση του πρωτοκόλλου SNMP (Simple Network Management Protocol), τη δημιουργία ενός SNMP agent (πράκτορα) στην πλατφόρμα Arduino και την αποστολή ασύγχρονων ειδοποιήσεων, SNMP traps, προς τον υπολογιστή που έχει το ρόλο του διαχειριστή δικτύου (-NMS- Network Management System) κάθε φορά που παρουσιάζεται κάποιο γεγονός [48], [49].

Εξέλιξη των πρωτοκόλλων I^2C και SPI θα αποτελέσει το πρωτόκολλο I^3C το οποίο αναπτύχθηκε από τον οργανισμό MIPI Alliance INC. Σε αυτόν τον οργανισμό συμμετέχουν 275 εταιρίες, κάποιες από τις οποίες είναι οι AMD, Audience, Broadcom, Cadence, Intel, InvenSense, Lattice, MediaTek, Mentor, Graphics, Nvidia, NXP, STMicroelectronics, Synopsys, Qualcomm, QuickLogic και ZMDI. Ο δίαυλος I^3C θα χρησιμοποιεί και αυτός δύο καλώδια, θα είναι συμβατός με τον I^2C, αλλά όμως θα έχει δυνατότητες στη διακίνηση των δεδομένων συγκρίσιμες με τον SPI. Οι προδιαγραφές του έχουν σκοπό να προβλέψουν τις νέες αρχιτεκτονικές στους αισθητήρες που θα καλύψουν τις μελλοντικές ανάγκες σε κινητά συστήματα, σε συστήματα που σχετίζονται με κινητά, ή σε ενσωματωμένα συστήματα. Η υλοποίηση του I^3C σε κινητές συσκευές θα έχει την ονομασία SenceWire και επιπλέον θα έχει σκοπό τη διασύνδεση των κινητών συσκευών με μια σειρά από αισθητήρες άμεσα ή έμμεσα [50].

Βιβλιογραφία

[1] P. Corcoran, *Two Wires and 30 Years, A tribute and introductory tutorial to the I²C two-wire bus*, IEEE Consumer Electronics Magazine, July 2013.

[2] NXP, *I²C-bus specification and user manual*, Rev. 6, April 2014.
Διαθέσιμο από: http://www.nxp.com/documents/user_manual/UM10204.pdf

[3] Byte Paradigm, *Introduction to I²C and SPI protocols*.
Διαθέσιμο από: http://www.byteparadigm.com/applications/introduction-to-i2c-and-spi-protocols.

[4] best-microcontroller-projects.com, *An I2C Tutorial*.
Διαθέσιμο από: http://www.best-microcontroller-projects.com/i2c-tutorial.html

[5] Wikipedia – The Free Encyclopedia, *Motorola 68000*.
Διαθέσιμο από: https://en.wikipedia.org/wiki/Motorola_68000

[6] Wikipedia – The Free Encyclopedia, *Serial Peripheral Interface Bus*.
Διαθέσιμο από: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface_Bus

[7] mct.net, SPI – Serial Peripheral Interface.
Διαθέσιμο από: http://www.mct.net/faq/spi.html#manufacturer

[8] best-microcontroller-projects.com, *SPI Interface Tutorial*.
Διαθέσιμο από: http://www.best-microcontroller-projects.com/spi-interface.html

[9] Corelis – An EWA Company, *SPI Interface*.
Διαθέσιμο από: http://www.corelis.com/education/SPI_Tutorial.htm

[10] EE Herald, *Online course on Embedded Systems – SPI Bus interface*.
Διαθέσιμο από: http://www.eeherald.com/section/design-guide/esmod12.html

[11] Maxim-IC application note 3947, *Daisy-Chaining SPI Devices*.
Διαθέσιμο από: https://www.maximintegrated.com/en/app-notes/index.mvp/id/3947

[12] Freescale Semiconductor Inc., *SPI Block Guide V04.01*.
Διαθέσιμο από: http://read.pudn.com/downloads87/ebook/336282/S12SPIV4.pdf

[13] BitWizard.nl, *SPI versus I2C protocols*.
Διαθέσιμο από: http://www.bitwizard.nl/wiki/index.php/SPI_versus_I2C_protocols

[14] How2ans, *I2C vs SPI Communication Protocol Basic*.
Διαθέσιμο από: http://www.how2ans.in/2015/03/i2c-vs-spi-synchronous-serial-protocol.html

[15] Quora, *What are the pros and cons of I2C versus SPI interface?*
Διαθέσιμο από: https://www.quora.com/What-are-the-pros-and-cons-of-I2C-versus-SPI-interface

[16] Arduino.cc, *What is Arduino*.
Διαθέσιμο από: https://www.arduino.cc/en/Guide/Introduction

[17] DeltaHacker, *Εισαγωγή στο Arduino – Το απόλυτο geek toy.*
Διαθέσιμο από: https://deltahacker.gr/arduino-intro/

[18] IEEE Spectrum, *The Making of Arduino*.
Διαθέσιμο από: http://spectrum.ieee.org/geek-life/hands-on/the-making-of-arduino/0

[19] Circuits Today, *Story and History of Development of Arduino*.
Διαθέσιμο από: http://www.circuitstoday.com/story-and-history-of-development-of-arduino

[20] Processing. Διαθέσιμο από: https://processing.org/

[21] Wiring. Διαθέσιμο από: http://wiring.org.co/

[22] Creative Commons Corporation. Διαθέσιμο από: https://creativecommons.org/about/

[23] Wikipedia – The Free Encyclopedia, *Arduino*.
Διαθέσιμο από: https://en.wikipedia.org/wiki/Arduino

[24] DifferenceBetween.net, Difference Between EEPROM and Flash.
Διαθέσιμο από: http://www.differencebetween.net/technology/hardware-technology/difference-between-eeprom-and-flash/

[25] Arduino.cc, *Memory*.
Διαθέσιμο από: https://www.arduino.cc/en/Tutorial/Memory

[26] Arduino.cc, *Arduino Boards*.
Διαθέσιμο από: https://www.arduino.cc/en/Main/Products

[27] Arduino.cc, *Arduino Duemilanove*.
Διαθέσιμο από: https://www.arduino.cc/en/Main/ArduinoBoardDuemilanove

[28] Arduino.cc, *Arduino Ethernet Shield*.
Διαθέσιμο από: https://www.arduino.cc/en/Main/ArduinoEthernetShield

[29] WIZnet Co., *W5100 Datasheet Version 1.2.6*.
Διαθέσιμο από: http://www.wiznet.co.kr/wp-content/uploads/wiznethome/Chip/W5100/Document/W5100_Datasheet_v1.2.6.pdf

[30] Microchip Technology Inc., *MCP23017/MCP23S17*.
Διαθέσιμο από: http://ww1.microchip.com/downloads/en/DeviceDoc/21952b.pdf

[31] Chipster.ru, J204A instructions.
Διαθέσιμο από: http://chipster.ru/upload/files/datasheet/2/7/2/272_file.pdf

[32] Hitachi, *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller/Driver)*.
Διαθέσιμο από: https://www.sparkfun.com/datasheets/LCD/HD44780.pdf

[33] Arduino-info Wiki, *LCD Addressing*.
Διαθέσιμο από: https://arduino-info.wikispaces.com/file/view/LCD+Addressing.pdf

[34] Texas Instruments, *P82B96 I2C Compatible Dual Bidirectional Bus Buffer*.
Διαθέσιμο από: http://www.ti.com/lit/ds/symlink/p82b96.pdf

[35] Electronics StackExchange, *Electrical Engineering: Is there a correct resistance value for I2C pull-up resistors?*
Διαθέσιμο από: http://electronics.stackexchange.com/questions/1849/is-there-a-correct-resistance-value-for-i2c-pull-up-resistors

[36] DSS Circuits, *Effects of Varying I2C Pull-Up Resistors*.
Διαθέσιμο από: http://dsscircuits.com/articles/86-articles/47-effects-of-varying-i2c-pull-up-resistors

[37] Texas Instruments, *I2C Bus Pullup Resistor Calculation*, Application Report SLVA689 – February 2015.
Διαθέσιμο από: http://www.ti.com.cn/cn/lit/an/slva689/slva689.pdf

[38] Fritzing. Διαθέσιμο από: http://fritzing.org/home/

[39] Arduino forum, *Implementation of an eeprom integer read / write*.
Διαθέσιμο από: http://forum.arduino.cc/index.php/topic,37470.0.html

[40] Arduino forum, *Arduino Data Web Server*.
Διαθέσιμο από: http://forum.arduino.cc/index.php/topic,6595.0.html#0
[41] Arduino forum, *Network settings web page FORM using EEPROM to save submit*.
Διαθέσιμο από: http://forum.arduino.cc/index.php/topic,55044.0.html
[42] JO3RI, *Embedded Ethernet Setup Webpage using EEPROM*.
Διαθέσιμο από: http://www.jo3ri.be/arduino/projects/network-settings-web-page-form-using-eeprom-to-save-submit
[43] Arduino playground, *PROGMEM*.
Διαθέσιμο από: http://playground.arduino.cc/Main/PROGMEM
[44] Arduino forum, *Remote I/O Expander I2C MCP23017*.
Διαθέσιμο από: http://forum.arduino.cc/index.php/topic,13227.0.html
[45] Arduino.cc, *LiquidCrystal Library*.
Διαθέσιμο από: https://www.arduino.cc/en/Reference/LiquidCrystal
[46] Arduino.cc, *Arduino Pro Mini*.
Διαθέσιμο από: https://www.arduino.cc/en/Main/ArduinoBoardProMini
[47] ELEC Freacs, *ENC28J60 Mini Ethernet Module (3.3V/5V)*.
Διαθέσιμο από: http://www.elecfreaks.com/wiki/index.php?title=ENC28J60_Mini_Ethernet_Module_%283.3V/5V%29
[48] CISCO, *Understanding Simple Network Management Protocol (SNMP) Traps*.
Διαθέσιμο από: http://www.cisco.com/c/en/us/support/docs/ip/simple-network-management-protocol-snmp/7244-snmp-trap.html
[49] Google Code, agentuino.
Διαθέσιμο από: https://code.google.com/archive/p/agentuino/
[50] EEtimes, *I3C sensor interface set to superset I2C, SPI*.
Διαθέσιμο από: http://www.electronics-eetimes.com/news/i3c-sensor-interface-set-superset-i2c-spi

Παράρτημα: Ο κώδικας της εφαρμογής

Παρατίθεται ολόκληρος ο κώδικας της εφαρμογής. Αρχικά δίνεται ο κώδικας του προγράμματος (sketch) του Arduino και στη συνέχεια ο κώδικας του προγράμματος C#.

Ο κώδικας του προγράμματος του Arduino

```
/*
 Author: Themistoklis Dimitrakopoulos May 2014
 128 digital inputs over I2C & port expander IC
 Liquid Crystal 20x4 shows events on inputs
 */

/*
 References:
 [1] Implementation of an eeprom integer read / write,
http://forum.arduino.cc/index.php/topic,37470.0.html
 [2] Arduino Data Web Server,
http://forum.arduino.cc/index.php/topic,6595.0.html#0
 [3] Network settings web page FORM using EEPROM to save submit,
http://forum.arduino.cc/index.php/topic,55044.0.html
 [4] Embedded Ethernet Setup Webpage using EEPROM,
http://www.jo3ri.be/arduino/projects/network-settings-web-page-form-
using-eeprom-to-save-submit
 [5] Arduino Playground, PROGMEM,
http://playground.arduino.cc/Main/PROGMEM
 [6] Arduino forum, Remote I/O Expander I2C MCP23017,
http://forum.arduino.cc/index.php/topic,13227.0.html
 [7] Arduino.cc, LiquidCrystal Library,
https://www.arduino.cc/en/Reference/LiquidCrystal
 */

#include <LiquidCrystal.h>          // Liquid Crystal library
#include <Wire.h>                   // I2C library
#include <avr/pgmspace.h>           // Flash Memory library
#include <stdio.h>                  // for sprintf()
#include <SPI.h>                    // SPI library
#include <Ethernet.h>               // library for Ethernet Shield
#include <string.h>                 // for strings
#include <TextFinder.h>             // used for ethernet setup only
#include <EEPROM.h>                 // used for ethernet setup only

//seting up the EthernetShield
//change the defaults to match your own network
byte mac[] = {
  0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED};
byte ip[] = {
  192,168,1,117};
byte subnet[] = {
  255,255,255,0};
byte gateway[] = {
  192,168,1,1};
```

```
byte dnsserver[] = {
  192,168,1,1};
int port = 80;

// -- used for ethernet setup --
// This is our buffer through which we will will "flow" our HTML code.
// It has to be as big as the longest character chain +1 including the
"
char buffer[100];

// HTML code chopped up. The HTML code has been typed in an editor and
then
// the characters have been counted and divided by 8.
// HTML can be chopped on every place, but not on the \" parts.
// We use \" instead of simple " within the HTML, otherwise it will
not work.

prog_char htmlx0[] PROGMEM = "<html><title>Arduino Ethernet Setup
Page</title><body marginwidth=\"0\" marginheight=\"0\" ";
prog_char htmlx1[] PROGMEM = "leftmargin=\"0\" style=\"margin: 0;
padding: 0;\"><table bgcolor=\"#999999\" border";
prog_char htmlx2[] PROGMEM = "=\"0\" width=\"100%\" cellpadding=\"1\"
style=\"font-family:Verdana;color:#fff";
prog_char htmlx3[] PROGMEM = "fff;font-size:12px;\"><tr><td> 
Arduino Ethernet Setup Page</td></tr></table><br>";
PROGMEM const char *string_table0[] = {
  htmlx0, htmlx1, htmlx2, htmlx3};

prog_char htmla0[] PROGMEM = "<script>function hex2num (s_hex)
{eval(\"var n_num=0X\" + s_hex);return n_num;}";
prog_char htmla1[] PROGMEM = "</script><table><form><input
type=\"hidden\" name=\"SBM\" value=\"1\"><tr><td>MAC:";
prog_char htmla2[] PROGMEM = "<input id=\"T1\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT1\" value=\"";
prog_char htmla3[] PROGMEM = "\">.<input id=\"T3\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT2\" value=\"";
prog_char htmla4[] PROGMEM = "\">.<input id=\"T5\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT3\" value=\"";
prog_char htmla5[] PROGMEM = "\">.<input id=\"T7\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT4\" value=\"";
prog_char htmla6[] PROGMEM = "\">.<input id=\"T9\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT5\" value=\"";
prog_char htmla7[] PROGMEM = "\">.<input id=\"T11\" type=\"text\"
size=\"2\" maxlength=\"2\" name=\"DT6\" value=\"";
PROGMEM const char *string_table1[] = {
  htmla0, htmla1, htmla2, htmla3, htmla4, htmla5, htmla6, htmla7};

prog_char htmlb0[] PROGMEM = "\"><input id=\"T2\" type=\"hidden\"
name=\"DT1\"><input id=\"T4\" type=\"hidden\" name=\"DT2";
prog_char htmlb1[] PROGMEM = "\"><input id=\"T6\" type=\"hidden\"
name=\"DT3\"><input id=\"T8\" type=\"hidden\" name=\"DT4";
```

```
prog_char htmlb2[] PROGMEM = "\"><input id=\"T10\" type=\"hidden\"
name=\"DT5\"><input id=\"T12\" type=\"hidden\" name=\"D";
prog_char htmlb3[] PROGMEM = "T6\"></td></tr><tr><td>IP: <input
type=\"text\" size=\"3\" maxlength=\"3\" name=\"DT7\" value=\"";
prog_char htmlb4[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT8\" value=\"";
prog_char htmlb5[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT9\" value=\"";
prog_char htmlb6[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT10\" value=\"";
prog_char htmlb7[] PROGMEM = "\"></td></tr><tr><td>PORT: <input
type=\"text\" size=\"5\" maxlength=\"5\" name=\"DT19\" value=\"";
PROGMEM const char *string_table2[] = {
  htmlb0, htmlb1, htmlb2, htmlb3, htmlb4, htmlb5, htmlb6, htmlb7};

prog_char htmlc0[] PROGMEM = "\"></td></tr><tr><td>MASK: <input
type=\"text\" size=\"3\" maxlength=\"3\" name=\"DT11\" value=\"";
prog_char htmlc1[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT12\" value=\"";
prog_char htmlc2[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT13\" value=\"";
prog_char htmlc3[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT14\" value=\"";
PROGMEM const char *string_table3[] = {
  htmlc0, htmlc1, htmlc2, htmlc3};

prog_char htmld0[] PROGMEM = "\"></td></tr><tr><td>GW: <input
type=\"text\" size=\"3\" maxlength=\"3\" name=\"DT15\" value=\"";
prog_char htmld1[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT16\" value=\"";
prog_char htmld2[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT17\" value=\"";
prog_char htmld3[] PROGMEM = "\">.<input type=\"text\" size=\"3\"
maxlength=\"3\" name=\"DT18\" value=\"";
prog_char htmld4[] PROGMEM =
"\"></td></tr><tr><td><br></td></tr><tr><td><input
id=\"button1\"type=\"submit\" value=\"SUBMIT\" ";
prog_char htmld5[] PROGMEM =
"></td></tr></form></table></body></html>";
PROGMEM const char *string_table4[] = {
  htmld0, htmld1, htmld2, htmld3, htmld4, htmld5};

prog_char htmle0[] PROGMEM =
"Onclick=\"document.getElementById('T2').value ";
prog_char htmle1[] PROGMEM = "=
hex2num(document.getElementById('T1').value);";
prog_char htmle2[] PROGMEM = "document.getElementById('T4').value =
hex2num(document.getElementById('T3').value);";
prog_char htmle3[] PROGMEM = "document.getElementById('T6').value =
hex2num(document.getElementById('T5').value);";
prog_char htmle4[] PROGMEM = "document.getElementById('T8').value =
hex2num(document.getElementById('T7').value);";
```

```
prog_char htmle5[] PROGMEM = "document.getElementById('T10').value =
hex2num(document.getElementById('T9').value);";
prog_char htmle6[] PROGMEM = "document.getElementById('T12').value =
hex2num(document.getElementById('T11').value);\"";
PROGMEM const char *string_table5[] = {
  htmle0, htmle1, htmle2, htmle3, htmle4, htmle5, htmle6};

prog_char htmlz0[] PROGMEM = "<html><title>Arduino Ethernet
Shield</title><body marginwidth=\"0\" marginheight=\"0\" ";
prog_char htmlz1[] PROGMEM = "leftmargin=\"0\" style=\"margin: 0;
padding: 0;\"><table bgcolor=\"#999999\" border";
prog_char htmlz2[] PROGMEM = "=\"0\" width=\"100%\" cellpadding=\"1\"
style=\"font-family:Verdana;color:#fff";
prog_char htmlz3[] PROGMEM = "fff;font-size:12px;\"><tr><td> 
Arduino Ethernet Shield</td></tr></table><br></body></html>";
PROGMEM const char *string_table6[] = {
  htmlz0, htmlz1, htmlz2, htmlz3};

const byte ID = 0x92; //used to identify if valid data in EEPROM, the
"know" bit,
// if this is written in EEPROM the sketch has ran before
// We use this, so that the very first time we'll run this sketch it
will use
// the values written above.
// defining which EEPROM address we are using for what data

// -- end of ethernet setup --

// -- Parameters --
char cmd[15];    // For commands and parameters.
char param1[15];
char param2[15];
#define bufferMax 128
int bufferSize;
char buffer2[bufferMax];
byte ledPin=0;
byte digPinState[9];

// construction of each string to flash memory (show new event)
prog_char string_0[] PROGMEM = "[001]";
prog_char string_1[] PROGMEM = "[002]";
prog_char string_2[] PROGMEM = "[003]";
prog_char string_3[] PROGMEM = "[004]";
prog_char string_4[] PROGMEM = "[005]";
prog_char string_5[] PROGMEM = "[006]";
prog_char string_6[] PROGMEM = "[007]";
prog_char string_7[] PROGMEM = "[008]";
prog_char string_8[] PROGMEM = "[009]";
prog_char string_9[] PROGMEM = "[010]";
prog_char string_10[] PROGMEM = "[011]";
prog_char string_11[] PROGMEM = "[012]";
prog_char string_12[] PROGMEM = "[013]";
```

```
prog_char string_13[] PROGMEM = "[014]";
prog_char string_14[] PROGMEM = "[015]";
prog_char string_15[] PROGMEM = "[016]";
prog_char string_16[] PROGMEM = "[017]";
prog_char string_17[] PROGMEM = "[018]";
prog_char string_18[] PROGMEM = "[019]";
prog_char string_19[] PROGMEM = "[020]";
prog_char string_20[] PROGMEM = "[021]";
prog_char string_21[] PROGMEM = "[022]";
prog_char string_22[] PROGMEM = "[023]";
prog_char string_23[] PROGMEM = "[024]";
prog_char string_24[] PROGMEM = "[025]";
prog_char string_25[] PROGMEM = "[026]";
prog_char string_26[] PROGMEM = "[027]";
prog_char string_27[] PROGMEM = "[028]";
prog_char string_28[] PROGMEM = "[029]";
prog_char string_29[] PROGMEM = "[030]";
prog_char string_30[] PROGMEM = "[031]";
prog_char string_31[] PROGMEM = "[032]";
prog_char string_32[] PROGMEM = "[033]";
prog_char string_33[] PROGMEM = "[034]";
prog_char string_34[] PROGMEM = "[035]";
prog_char string_35[] PROGMEM = "[036]";
prog_char string_36[] PROGMEM = "[037]";
prog_char string_37[] PROGMEM = "[038]";
prog_char string_38[] PROGMEM = "[039]";
prog_char string_39[] PROGMEM = "[040]";
prog_char string_40[] PROGMEM = "[041]";
prog_char string_41[] PROGMEM = "[042]";
prog_char string_42[] PROGMEM = "[043]";
prog_char string_43[] PROGMEM = "[044]";
prog_char string_44[] PROGMEM = "[045]";
prog_char string_45[] PROGMEM = "[046]";
prog_char string_46[] PROGMEM = "[047]";
prog_char string_47[] PROGMEM = "[048]";
prog_char string_48[] PROGMEM = "[049]";
prog_char string_49[] PROGMEM = "[050]";
prog_char string_50[] PROGMEM = "[051]";
prog_char string_51[] PROGMEM = "[052]";
prog_char string_52[] PROGMEM = "[053]";
prog_char string_53[] PROGMEM = "[054]";
prog_char string_54[] PROGMEM = "[055]";
prog_char string_55[] PROGMEM = "[056]";
prog_char string_56[] PROGMEM = "[057]";
prog_char string_57[] PROGMEM = "[058]";
prog_char string_58[] PROGMEM = "[059]";
prog_char string_59[] PROGMEM = "[060]";
prog_char string_60[] PROGMEM = "[061]";
prog_char string_61[] PROGMEM = "[062]";
prog_char string_62[] PROGMEM = "[063]";
prog_char string_63[] PROGMEM = "[064]";
prog_char string_64[] PROGMEM = "[065]";
```

```
prog_char string_65[] PROGMEM = "[066]";
prog_char string_66[] PROGMEM = "[067]";
prog_char string_67[] PROGMEM = "[068]";
prog_char string_68[] PROGMEM = "[069]";
prog_char string_69[] PROGMEM = "[070]";
prog_char string_70[] PROGMEM = "[071]";
prog_char string_71[] PROGMEM = "[072]";
prog_char string_72[] PROGMEM = "[073]";
prog_char string_73[] PROGMEM = "[074]";
prog_char string_74[] PROGMEM = "[075]";
prog_char string_75[] PROGMEM = "[076]";
prog_char string_76[] PROGMEM = "[077]";
prog_char string_77[] PROGMEM = "[078]";
prog_char string_78[] PROGMEM = "[079]";
prog_char string_79[] PROGMEM = "[080]";
prog_char string_80[] PROGMEM = "[081]";
prog_char string_81[] PROGMEM = "[082]";
prog_char string_82[] PROGMEM = "[083]";
prog_char string_83[] PROGMEM = "[084]";
prog_char string_84[] PROGMEM = "[085]";
prog_char string_85[] PROGMEM = "[086]";
prog_char string_86[] PROGMEM = "[087]";
prog_char string_87[] PROGMEM = "[088]";
prog_char string_88[] PROGMEM = "[089]";
prog_char string_89[] PROGMEM = "[090]";
prog_char string_90[] PROGMEM = "[091]";
prog_char string_91[] PROGMEM = "[092]";
prog_char string_92[] PROGMEM = "[093]";
prog_char string_93[] PROGMEM = "[094]";
prog_char string_94[] PROGMEM = "[095]";
prog_char string_95[] PROGMEM = "[096]";
prog_char string_96[] PROGMEM = "[097]";
prog_char string_97[] PROGMEM = "[098]";
prog_char string_98[] PROGMEM = "[099]";
prog_char string_99[] PROGMEM = "[100]";
prog_char string_100[] PROGMEM = "[101]";
prog_char string_101[] PROGMEM = "[102]";
prog_char string_102[] PROGMEM = "[103]";
prog_char string_103[] PROGMEM = "[104]";
prog_char string_104[] PROGMEM = "[105]";
prog_char string_105[] PROGMEM = "[106]";
prog_char string_106[] PROGMEM = "[107]";
prog_char string_107[] PROGMEM = "[108]";
prog_char string_108[] PROGMEM = "[109]";
prog_char string_109[] PROGMEM = "[110]";
prog_char string_110[] PROGMEM = "[111]";
prog_char string_111[] PROGMEM = "[112]";
prog_char string_112[] PROGMEM = "[113]";
prog_char string_113[] PROGMEM = "[114]";
prog_char string_114[] PROGMEM = "[115]";
prog_char string_115[] PROGMEM = "[116]";
prog_char string_116[] PROGMEM = "[117]";
```

```
prog_char string_117[] PROGMEM = "[118]";
prog_char string_118[] PROGMEM = "[119]";
prog_char string_119[] PROGMEM = "[120]";
prog_char string_120[] PROGMEM = "[121]";
prog_char string_121[] PROGMEM = "[122]";
prog_char string_122[] PROGMEM = "[123]";
prog_char string_123[] PROGMEM = "[124]";
prog_char string_124[] PROGMEM = "[125]";
prog_char string_125[] PROGMEM = "[126]";
prog_char string_126[] PROGMEM = "[127]";
prog_char string_127[] PROGMEM = "[128]";

// construction of table of strings to flash memory (show new event)
PROGMEM const char *myRooms[] =
{
  string_0, string_1, string_2, string_3, string_4, string_5,
string_6, string_7, string_8, string_9,
  string_10, string_11, string_12, string_13, string_14, string_15,
string_16, string_17, string_18, string_19,
  string_20, string_21, string_22, string_23, string_24, string_25,
string_26, string_27, string_28, string_29,
  string_30, string_31, string_32, string_33, string_34, string_35,
string_36, string_37, string_38, string_39,
  string_40, string_41, string_42, string_43, string_44, string_45,
string_46, string_47, string_48, string_49,
  string_50, string_51, string_52, string_53, string_54, string_55,
string_56, string_57, string_58, string_59,
  string_60, string_61, string_62, string_63, string_64, string_65,
string_66, string_67, string_68, string_69,
  string_70, string_71, string_72, string_73, string_74, string_75,
string_76, string_77, string_78, string_79,
  string_80, string_81, string_82, string_83, string_84, string_85,
string_86, string_87, string_88, string_89,
  string_90, string_81, string_92, string_93, string_94, string_95,
string_96, string_97, string_98, string_99,
  string_100, string_101, string_102, string_103, string_104,
string_105, string_106, string_107, string_108, string_109,
  string_110, string_111, string_112, string_113, string_114,
string_115, string_116, string_117, string_118, string_119,
  string_120, string_121, string_122, string_123, string_124,
string_125, string_126, string_127
};

// same as above (show acknowledged event)
prog_char Ackstring_0[] PROGMEM = " 001 ";
prog_char Ackstring_1[] PROGMEM = " 002 ";
prog_char Ackstring_2[] PROGMEM = " 003 ";
prog_char Ackstring_3[] PROGMEM = " 004 ";
prog_char Ackstring_4[] PROGMEM = " 005 ";
prog_char Ackstring_5[] PROGMEM = " 006 ";
prog_char Ackstring_6[] PROGMEM = " 007 ";
prog_char Ackstring_7[] PROGMEM = " 008 ";
```

```
prog_char Ackstring_8[] PROGMEM = " 009 ";
prog_char Ackstring_9[] PROGMEM = " 010 ";
prog_char Ackstring_10[] PROGMEM = " 011 ";
prog_char Ackstring_11[] PROGMEM = " 012 ";
prog_char Ackstring_12[] PROGMEM = " 013 ";
prog_char Ackstring_13[] PROGMEM = " 014 ";
prog_char Ackstring_14[] PROGMEM = " 015 ";
prog_char Ackstring_15[] PROGMEM = " 016 ";
prog_char Ackstring_16[] PROGMEM = " 017 ";
prog_char Ackstring_17[] PROGMEM = " 018 ";
prog_char Ackstring_18[] PROGMEM = " 019 ";
prog_char Ackstring_19[] PROGMEM = " 020 ";
prog_char Ackstring_20[] PROGMEM = " 021 ";
prog_char Ackstring_21[] PROGMEM = " 022 ";
prog_char Ackstring_22[] PROGMEM = " 023 ";
prog_char Ackstring_23[] PROGMEM = " 024 ";
prog_char Ackstring_24[] PROGMEM = " 025 ";
prog_char Ackstring_25[] PROGMEM = " 026 ";
prog_char Ackstring_26[] PROGMEM = " 027 ";
prog_char Ackstring_27[] PROGMEM = " 028 ";
prog_char Ackstring_28[] PROGMEM = " 029 ";
prog_char Ackstring_29[] PROGMEM = " 030 ";
prog_char Ackstring_30[] PROGMEM = " 031 ";
prog_char Ackstring_31[] PROGMEM = " 032 ";
prog_char Ackstring_32[] PROGMEM = " 033 ";
prog_char Ackstring_33[] PROGMEM = " 034 ";
prog_char Ackstring_34[] PROGMEM = " 035 ";
prog_char Ackstring_35[] PROGMEM = " 036 ";
prog_char Ackstring_36[] PROGMEM = " 037 ";
prog_char Ackstring_37[] PROGMEM = " 038 ";
prog_char Ackstring_38[] PROGMEM = " 039 ";
prog_char Ackstring_39[] PROGMEM = " 040 ";
prog_char Ackstring_40[] PROGMEM = " 041 ";
prog_char Ackstring_41[] PROGMEM = " 042 ";
prog_char Ackstring_42[] PROGMEM = " 043 ";
prog_char Ackstring_43[] PROGMEM = " 044 ";
prog_char Ackstring_44[] PROGMEM = " 045 ";
prog_char Ackstring_45[] PROGMEM = " 046 ";
prog_char Ackstring_46[] PROGMEM = " 047 ";
prog_char Ackstring_47[] PROGMEM = " 048 ";
prog_char Ackstring_48[] PROGMEM = " 049 ";
prog_char Ackstring_49[] PROGMEM = " 050 ";
prog_char Ackstring_50[] PROGMEM = " 051 ";
prog_char Ackstring_51[] PROGMEM = " 052 ";
prog_char Ackstring_52[] PROGMEM = " 053 ";
prog_char Ackstring_53[] PROGMEM = " 054 ";
prog_char Ackstring_54[] PROGMEM = " 055 ";
prog_char Ackstring_55[] PROGMEM = " 056 ";
prog_char Ackstring_56[] PROGMEM = " 057 ";
prog_char Ackstring_57[] PROGMEM = " 058 ";
prog_char Ackstring_58[] PROGMEM = " 059 ";
prog_char Ackstring_59[] PROGMEM = " 060 ";
```

```
prog_char Ackstring_60[] PROGMEM = " 061 ";
prog_char Ackstring_61[] PROGMEM = " 062 ";
prog_char Ackstring_62[] PROGMEM = " 063 ";
prog_char Ackstring_63[] PROGMEM = " 064 ";
prog_char Ackstring_64[] PROGMEM = " 065 ";
prog_char Ackstring_65[] PROGMEM = " 066 ";
prog_char Ackstring_66[] PROGMEM = " 067 ";
prog_char Ackstring_67[] PROGMEM = " 068 ";
prog_char Ackstring_68[] PROGMEM = " 069 ";
prog_char Ackstring_69[] PROGMEM = " 070 ";
prog_char Ackstring_70[] PROGMEM = " 071 ";
prog_char Ackstring_71[] PROGMEM = " 072 ";
prog_char Ackstring_72[] PROGMEM = " 073 ";
prog_char Ackstring_73[] PROGMEM = " 074 ";
prog_char Ackstring_74[] PROGMEM = " 075 ";
prog_char Ackstring_75[] PROGMEM = " 076 ";
prog_char Ackstring_76[] PROGMEM = " 077 ";
prog_char Ackstring_77[] PROGMEM = " 078 ";
prog_char Ackstring_78[] PROGMEM = " 079 ";
prog_char Ackstring_79[] PROGMEM = " 080 ";
prog_char Ackstring_80[] PROGMEM = " 081 ";
prog_char Ackstring_81[] PROGMEM = " 082 ";
prog_char Ackstring_82[] PROGMEM = " 083 ";
prog_char Ackstring_83[] PROGMEM = " 084 ";
prog_char Ackstring_84[] PROGMEM = " 085 ";
prog_char Ackstring_85[] PROGMEM = " 086 ";
prog_char Ackstring_86[] PROGMEM = " 087 ";
prog_char Ackstring_87[] PROGMEM = " 088 ";
prog_char Ackstring_88[] PROGMEM = " 089 ";
prog_char Ackstring_89[] PROGMEM = " 090 ";
prog_char Ackstring_90[] PROGMEM = " 091 ";
prog_char Ackstring_91[] PROGMEM = " 092 ";
prog_char Ackstring_92[] PROGMEM = " 093 ";
prog_char Ackstring_93[] PROGMEM = " 094 ";
prog_char Ackstring_94[] PROGMEM = " 095 ";
prog_char Ackstring_95[] PROGMEM = " 096 ";
prog_char Ackstring_96[] PROGMEM = " 097 ";
prog_char Ackstring_97[] PROGMEM = " 098 ";
prog_char Ackstring_98[] PROGMEM = " 099 ";
prog_char Ackstring_99[] PROGMEM = " 100 ";
prog_char Ackstring_100[] PROGMEM = " 101 ";
prog_char Ackstring_101[] PROGMEM = " 102 ";
prog_char Ackstring_102[] PROGMEM = " 103 ";
prog_char Ackstring_103[] PROGMEM = " 104 ";
prog_char Ackstring_104[] PROGMEM = " 105 ";
prog_char Ackstring_105[] PROGMEM = " 106 ";
prog_char Ackstring_106[] PROGMEM = " 107 ";
prog_char Ackstring_107[] PROGMEM = " 108 ";
prog_char Ackstring_108[] PROGMEM = " 109 ";
prog_char Ackstring_109[] PROGMEM = " 110 ";
prog_char Ackstring_110[] PROGMEM = " 111 ";
prog_char Ackstring_111[] PROGMEM = " 112 ";
```

```
prog_char Ackstring_112[] PROGMEM = " 113 ";
prog_char Ackstring_113[] PROGMEM = " 114 ";
prog_char Ackstring_114[] PROGMEM = " 115 ";
prog_char Ackstring_115[] PROGMEM = " 116 ";
prog_char Ackstring_116[] PROGMEM = " 117 ";
prog_char Ackstring_117[] PROGMEM = " 118 ";
prog_char Ackstring_118[] PROGMEM = " 119 ";
prog_char Ackstring_119[] PROGMEM = " 120 ";
prog_char Ackstring_120[] PROGMEM = " 121 ";
prog_char Ackstring_121[] PROGMEM = " 122 ";
prog_char Ackstring_122[] PROGMEM = " 123 ";
prog_char Ackstring_123[] PROGMEM = " 124 ";
prog_char Ackstring_124[] PROGMEM = " 125 ";
prog_char Ackstring_125[] PROGMEM = " 126 ";
prog_char Ackstring_126[] PROGMEM = " 127 ";
prog_char Ackstring_127[] PROGMEM = " 128 ";

PROGMEM const char *myRoomsAck[] =
{
  Ackstring_0, Ackstring_1, Ackstring_2, Ackstring_3, Ackstring_4,
Ackstring_5, Ackstring_6, Ackstring_7, Ackstring_8, Ackstring_9,
  Ackstring_10, Ackstring_11, Ackstring_12, Ackstring_13,
Ackstring_14, Ackstring_15, Ackstring_16, Ackstring_17, Ackstring_18,
Ackstring_19,
  Ackstring_20, Ackstring_21, Ackstring_22, Ackstring_23,
Ackstring_24, Ackstring_25, Ackstring_26, Ackstring_27, Ackstring_28,
Ackstring_29,
  Ackstring_30, Ackstring_31, Ackstring_32, Ackstring_33,
Ackstring_34, Ackstring_35, Ackstring_36, Ackstring_37, Ackstring_38,
Ackstring_39,
  Ackstring_40, Ackstring_41, Ackstring_42, Ackstring_43,
Ackstring_44, Ackstring_45, Ackstring_46, Ackstring_47, Ackstring_48,
Ackstring_49,
  Ackstring_50, Ackstring_51, Ackstring_52, Ackstring_53,
Ackstring_54, Ackstring_55, Ackstring_56, Ackstring_57, Ackstring_58,
Ackstring_59,
  Ackstring_60, Ackstring_61, Ackstring_62, Ackstring_63,
Ackstring_64, Ackstring_65, Ackstring_66, Ackstring_67, Ackstring_68,
Ackstring_69,
  Ackstring_70, Ackstring_71, Ackstring_72, Ackstring_73,
Ackstring_74, Ackstring_75, Ackstring_76, Ackstring_77, Ackstring_78,
Ackstring_79,
  Ackstring_80, Ackstring_81, Ackstring_82, Ackstring_83,
Ackstring_84, Ackstring_85, Ackstring_86, Ackstring_87, Ackstring_88,
Ackstring_89,
  Ackstring_90, Ackstring_81, Ackstring_92, Ackstring_93,
Ackstring_94, Ackstring_95, Ackstring_96, Ackstring_97, Ackstring_98,
Ackstring_99,
  Ackstring_100, Ackstring_101, Ackstring_102, Ackstring_103,
Ackstring_104, Ackstring_105, Ackstring_106, Ackstring_107,
Ackstring_108, Ackstring_109,
```

```
  Ackstring_110, Ackstring_111, Ackstring_112, Ackstring_113,
Ackstring_114, Ackstring_115, Ackstring_116, Ackstring_117,
Ackstring_118, Ackstring_119,
  Ackstring_120, Ackstring_121, Ackstring_122, Ackstring_123,
Ackstring_124, Ackstring_125, Ackstring_126, Ackstring_127
};

LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // pins for LCD

//redefine highByte and lowByte to return byte, not int
#define loByte (byte)lowByte
#define hiByte (byte)highByte
#define iodira 0x00              // iodira register address
#define gpioa 0x12               // gpioa register address
#define gppua 0x0C               // gppua register address
#define wait 100                 // time space 1 sec between two
sequential page changes
#define resetTime 3000           // time 3 sec for keeping pressing
reset button to reset
#define ledflash 500             // time 0.5 sec red led flashing
#define inputNext 3              // push button Next
#define inputPrev 2              // push button Previous
#define inputAckReset 17         // push button Acknowledge
#define outputResetLed 16        // green led indicating reset process
#define outputAlarmLed 15        // red led alarm
#define outputAlarmBuzzer 14     // buzzer alarm

byte inputsA, inputsB;     // 8 bit number inputs A or inputs B of IC
byte active[128];          // if event appears on port, port is active
byte change=0;             // changes to 1 if event happens or
acknowledge button pressed
byte buttonChangeReset=0;        // changes to 1 if reset is possible
byte page=0;                     // page variable
byte pageMax=0;                  // max page
byte totalPage=0;                // max page + 1 (appears on display)
byte currentPage=0;              // page + 1 (appears on display)
byte printAck=0;                 // changes to 1 if acknowledge
happens & serves screen refreshing
byte stateAck=0;                // changes to 1 if acknowledge happens
byte stateReset=0;              // changes to 1 if reset happens
byte buzState=0;                // changes to 1 if buzzer activated
byte ledState=0;                // changes to 1 if led activated
byte table[128]; // each port go active is stored in a table ascending
byte events=0;   // events++ on every port go active. Counts events.
byte numAck=0;   // number of events that got acknowledged
byte newEvents=0;               // number of new events
byte k=0;
byte j=0;
byte buttonPrev=1;
byte buttonNext=1;
byte buttonAckReset=1;
byte resetTrue=0;
```

```
byte ackTrue=0;
byte ackEvents=0;
char text[3];
char tempStr[6];
unsigned long movePreviousMillis=0;
unsigned long resetPreviousMillis=0;
unsigned long resetCurrentMillis=0;
unsigned long ledPreviousMillis=0;

void ShieldSetup()
{
  int idcheck = EEPROM.read(0);
  if (idcheck != ID){
    //ifcheck id is not the value as const byte ID,
    //it means this sketch has NOT been used to setup the shield
before
    //just use the values written in the beginning of the sketch
  }
  if (idcheck == ID){
    //if id is the same value as const byte ID,
    //it means this sketch has been used to setup the shield.
    //So we will read the values out of EERPOM ans use them
    //to setup the shield.
    for (int i = 0; i < 6; i++){
      mac[i] = EEPROM.read(i+1);
    }
    for (int i = 0; i < 4; i++){
      ip[i] = EEPROM.read(i+7);
    }
    for (int i = 0; i < 4; i++){
      subnet[i] = EEPROM.read(i+11);
    }
    for (int i = 0; i < 4; i++){
      gateway[i] = EEPROM.read(i+15);
    }
    port = EEPROMReadInt(19);
  }
  Ethernet.begin(mac, ip, dnsserver, gateway, subnet);
}

void LCD_Start()
{
  lcd.clear(); // clear screen
  lcd.setCursor(0,0);  // set the cursor to column 0, line 0
  lcd.print("No events");
  lcd.setCursor(0, 3);
  lcd.print("PG:01/01");
  lcd.setCursor(13, 3);
  lcd.print("NEW:000");
  for (byte i=0; i<128;i++)
  {
    table[i]=0;
```

```
    active[i]=0;
  }
  pageMax=0;
  page=0;
  events=0;
  j=0;
  numAck=0;
  ackEvents=0;
  resetCurrentMillis=millis();
  resetPreviousMillis=millis();
  digitalWrite(outputAlarmLed, LOW);
  digitalWrite(outputResetLed, LOW);
}

void MCPstore (byte i2caddress, byte i2cregister,  word i2cdata)    //
Store 16 bits in specified register of MCP23017
{
  byte controlbyte  = B0100000 | i2caddress;               // note
that bit 0 not used b/c 7-bit opcode for Wire library
  Wire.beginTransmission(controlbyte);
  Wire.write(i2cregister);
  Wire.write(loByte(i2cdata));
  Wire.write(hiByte(i2cdata));
  Wire.endTransmission();
}

void MCPmode(byte opcode)
{
  byte controlbyte  = B0100000 | opcode;
  MCPstore (controlbyte,iodira,0xffff);         // all input
  MCPstore (controlbyte,gppua,0xffff);          // all pullups on
}

word MCPread (byte i2caddress, byte i2cregister)           // Read 16
bits from MCP23017
{
  byte controlbyte  = B0100000 | i2caddress;               // note
that bit 0 not used b/c 7-bit opcode for Wire library
  word tempdata;
  Wire.beginTransmission(controlbyte);
  Wire.write(i2cregister);
  Wire.endTransmission();
  Wire.requestFrom(controlbyte,(byte)2);                   //
otherwise function gets one byte and one int
  tempdata = Wire.read();                          // get lower 8 bytes
  tempdata |= (Wire.read() << 8                    // get upper 8 bytes
  return tempdata;
}

void PrintTable()
{
  if (change==1)
```

```
  {
    change=0;
    byte x=0;
    byte y=0;
    page=pageMax;
    if (events % 12==1 && events!=1 && printAck==0)
    {
      page++;
      lcd.clear();
    }
    printAck=0;
    pageMax=page;
    k=12*page;
    totalPage=page+1;
    newEvents=events-numAck;
    if (currentPage>0 && currentPage<totalPage)
    {
      currentPage=totalPage;
      lcd.clear();
    }
    if (events==1)
    {
      lcd.setCursor(0, 0);
      lcd.print("          ");
    }
    lcd.setCursor(0, 3);
    sprintf(text,"PG:%02d/%02d     NEW:%03d", totalPage, totalPage,
      newEvents);
    lcd.print(text);
    for (byte i=k;i<events;i++)
    {
      x=5*(i % 4);
      lcd.setCursor(x,y);
      byte m=table[i];
      if (i<numAck)
        strcpy_P(tempStr, (char*)pgm_read_word(&(myRoomsAck[m])));
      else
        strcpy_P(tempStr, (char*)pgm_read_word(&(myRooms[m])));
      lcd.print(tempStr);
      if (x==15) y++;
    }
  }
}

void PrintMove()
{
  unsigned long moveCurrentMillis = millis();
  if (moveCurrentMillis - movePreviousMillis > wait)
  {
    buttonPrev = digitalRead(inputPrev);
    buttonNext = digitalRead(inputNext);
```

```
    if ((buttonPrev==LOW && buttonNext==HIGH & page>0) ||
      (buttonNext==LOW && buttonPrev==HIGH & page<pageMax))
    {
      if (buttonPrev==LOW)
      {
        buttonPrev=1;
        page--;
      }
      if (buttonNext==LOW)
      {
        buttonNext=1;
        page++;
      }
      byte x=0;
      byte y=0;
      k=12*page;
      currentPage=page+1;
      byte mb=min(k+12,events);
      if (page==pageMax) lcd.clear(); // clear screen
      lcd.setCursor(0, 3);
      sprintf(text,"PG:%02d/%02d     NEW:%03d", currentPage,
       totalPage, newEvents);
      lcd.print(text);
      for (byte i=k;i<mb;i++)
      {
        x=5*(i % 4);
        lcd.setCursor(x,y);
        byte m=table[i];
        if (i<numAck)
          strcpy_P(tempStr, (char*)pgm_read_word(&(myRoomsAck[m])));
        else
          strcpy_P(tempStr, (char*)pgm_read_word(&(myRooms[m])));
        lcd.print(tempStr);
        if (x==15) y++;
      }
      movePreviousMillis = moveCurrentMillis;
    }
  }
}

void AckReset()
{
  buttonAckReset = digitalRead(inputAckReset);
  if (buttonAckReset==HIGH && stateAck==1) stateAck=0;
  if (buttonAckReset==LOW && newEvents>0)
  {
    digitalWrite(outputResetLed, LOW);
    if (buttonChangeReset==0 && stateAck==0)
    {
      buttonAckReset=1;
      numAck=events;
      printAck=1;
```

```
      change=1;
      stateAck=1;
      lcd.clear();
      PrintTable();
    }
  }
  if (buttonAckReset==HIGH && buttonChangeReset==1)
  {
    resetCurrentMillis=millis();
    resetPreviousMillis=millis();
    buttonChangeReset=0;
    stateReset=0;
    digitalWrite(outputResetLed, LOW);
  }
  if (buttonAckReset==LOW && events>0 && newEvents==0 && stateAck==0
    && stateReset==0)
  {
    resetCurrentMillis = millis();
    if (buttonChangeReset==0)
    {
      resetPreviousMillis=resetCurrentMillis;
      buttonChangeReset=1;
      digitalWrite(outputResetLed, HIGH);
    }
    if (resetCurrentMillis - resetPreviousMillis > resetTime)
    {
      stateReset=1;
      LCD_Start();
    }
  }
}

void Alarms()
{
  if (newEvents>0 && buzState==0)
  {
    buzState=1;
    digitalWrite(outputAlarmBuzzer, HIGH);
  }
  if (newEvents==0 & buzState==1)
  {
    digitalWrite(outputAlarmBuzzer, LOW);
    digitalWrite(outputAlarmLed, HIGH);
    buzState=0;
    ackEvents=1;
  }
  if (newEvents>0)
  {
    unsigned long ledCurrentMillis = millis();
    if (ledCurrentMillis - ledPreviousMillis > ledflash)
    {
      ledPreviousMillis = ledCurrentMillis;
```

```
      if (ledState == LOW)
        ledState=HIGH;
      else
        ledState=LOW;
      digitalWrite (outputAlarmLed, ledState);
    }
  }
}

byte ProcessChip(byte chip)
{
  word mcpdata = MCPread(chip,gpioa);
  byte inputsA = ~loByte(mcpdata);
  byte inputsB= ~hiByte(mcpdata);
  if (inputsA>0 || inputsB>0){
    if (inputsA>0){
      if (inputsA & B00000001)
      {
        byte port=chip*16;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B00000010)
      {
        byte port=chip*16+1;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B00000100)
      {
        byte port=chip*16+2;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
```

```
      if (inputsA & B00001000)
      {
        byte port=chip*16+3;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B00010000)
      {
        byte port=chip*16+4;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B00100000)
      {
        byte port=chip*16+5;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B01000000)
      {
        byte port=chip*16+6;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsA & B10000000)
      {
        byte port=chip*16+7;
        if (active[port]!=1)
```

```
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
    }
    if (inputsB>0){
      if (inputsB & B00000001)
      {
        byte port=chip*16+8;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsB & B00000010)
      {
        byte port=chip*16+9;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsB & B00000100)
      {
        byte port=chip*16+10;
        if (active[port]!=1)
        {
          table[j]=port;
          j++;
          events++;
          change=1;
          active[port]=1;
        }
      }
      if (inputsB & B00001000)
      {
        byte port=chip*16+11;
        if (active[port]!=1)
        {
          table[j]=port;
```

```
        j++;
        events++;
        change=1;
        active[port]=1;
      }
    }
    if (inputsB & B00010000)
    {
      byte port=chip*16+12;
      if (active[port]!=1)
      {
        table[j]=port;
        j++;
        events++;
        change=1;
        active[port]=1;
      }
    }
    if (inputsB & B00100000)
    {
      byte port=chip*16+13;
      if (active[port]!=1)
      {
        table[j]=port;
        j++;
        events++;
        change=1;
        active[port]=1;
      }
    }
    if (inputsB & B01000000)
    {
      byte port=chip*16+14;
      if (active[port]!=1)
      {
        table[j]=port;
        j++;
        events++;
        change=1;
        active[port]=1;
      }
    }
    if (inputsB & B10000000)
    {
      byte port=chip*16+15;
      if (active[port]!=1)
      {
        table[j]=port;
        j++;
        events++;
        change=1;
        active[port]=1;
```

```
        }
      }
    }
  }
}

void setup()
{
  ShieldSetup(); //Setup the Ethernet shield
  EthernetServer server(port);
  server.begin();
  Serial.begin(9600);
  pinMode(outputAlarmBuzzer, OUTPUT);
  pinMode(outputAlarmLed, OUTPUT);
  pinMode(outputResetLed, OUTPUT);
  pinMode(inputAckReset, INPUT_PULLUP);
  pinMode(inputPrev, INPUT_PULLUP);
  pinMode(inputNext, INPUT_PULLUP);
  lcd.begin(20, 4);  // set up the LCD's number of columns and rows
  LCD_Start();
  Wire.begin();
  MCPmode(0);
  MCPmode(1);
  MCPmode(2);
  MCPmode(3);
  MCPmode(4);
  MCPmode(5);
  MCPmode(6);
  MCPmode(7);
}

void WaitForRequest(EthernetClient client) // Sets buffer2[] and
bufferSize
{
  bufferSize = 0;
  while (client.connected()) {
    if (client.available()) {
      char c = client.read();
      if (c == '\n'){
        break;
      }
      else
        if (bufferSize < bufferMax)
          buffer2[bufferSize++] = c;
        else
          break;
    }
  }
}

void ParseReceivedRequest()
{
```

```
  //Received buffer contains "GET /cmd/param1/param2 HTTP/1.1".  Break
it up.
  char* slash1;
  char* slash2;
  char* space1;
  char* space2;
  char* qmark;

  slash1 = strstr(buffer2, "/") + 1; // Look for first slash
  slash2 = strstr(slash1, "/") + 1; // second slash
  space1 = strstr(slash1, " ") + 1; // space after second slash (in
case there is no second slash)
  qmark = strstr(buffer2, "?") + 1;
  space2 = strstr(slash2, " ") + 1; // space after second slash (in
case there is no third slash)

  if (slash2 > space1) slash2=space1;
  if (qmark > " ") slash2=qmark;

  // strncpy does not automatically add terminating zero, but strncat
does! So start with blank string and concatenate.
  cmd[0] = 0;
  strncat(cmd, slash1, slash2-slash1-1);
}

void PerformRequestedCommands(EthernetClient client)
{
  if ( strcmp(cmd,"digitalStatus") == 0 ) RemoteDigitalStatus(client);
  else
    if ( strcmp(cmd,"reset") == 0 ) UserReset(client);
    else
      if ( strcmp(cmd,"acknowledge") == 0 ) UserAck(client);
      else
        if ( strcmp(cmd,"alarmStatus") == 0 ) UserAlarm(client);
  if ( strcmp(cmd,"setup") == 0 ) AddressSetup(client);
  else
    if ( strcmp(cmd,"") == 0 ) AddressConfirm(client);
}

void UserReset(EthernetClient client)
{
  if (events>0 && newEvents==0 && stateAck==0 && stateReset==0)
  {
    resetTrue=1;
    LCD_Start();
  }
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/xml");
  client.println();
  client.println("<?xml version = \"1.0\" ?>");
  client.println("<userReset>");
  client.println("<success>");
```

```
  if (resetTrue==1)
  {
    resetTrue=0;
    client.println("TRUE");
  }
  else
    client.println("FALSE");
  client.println("</success>");
  client.println("</userReset>");
}

void UserAck(EthernetClient client)
{
  if (newEvents>0)
  {
    if (buttonChangeReset==0 && stateAck==0)
    {
      ackTrue=1;
      buttonAckReset=1;
      numAck=events;
      printAck=1;
      change=1;
      stateAck=1;
      lcd.clear();
      PrintTable();
    }
  }
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/xml");
  client.println();
  client.println("<?xml version = \"1.0\" ?>");
  client.println("<userAck>");
  client.println("<success>");
  if (ackTrue==1)
  {
    ackTrue=0;
    client.println("TRUE");
  }
  else
    client.println("FALSE");
  client.println("</success>");
  client.println("</userAck>");
}

void UserAlarm(EthernetClient client)
{
  // send the XML file with alarm statuses
client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/xml");
  client.println();
  client.println("<?xml version = \"1.0\" ?>");
  client.print("<userAlarm>");
```

```
  if (buzState==1)
  {
    client.print("Buzzer");
  }
  else
    if (ackEvents==1)
      client.print("ackEvents");
    else
      client.print("noAlarms");
  client.println("</userAlarm>");
}

void RemoteDigitalStatus(EthernetClient client)
{
  // send the XML file with digital input statuses
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/xml");
  client.println();
  client.println("<?xml version = \"1.0\" ?>");
  client.println("<status>");
  if (events > 0)
  {
    client.print("<alarmPorts>");
    char tempRoom[6];
    for (byte i=0;i<events;i++)
    {
      byte m=table[i];
      if(i>0){
        client.print('@');
      }
      strcpy_P(tempRoom, (char*)pgm_read_word(&(myRoomsAck[m])));
      client.print(tempRoom);
    }
    client.println("</alarmPorts>");
    client.print("<ackPorts>");
    client.print(numAck, DEC);
    client.println("</ackPorts>");
  }
  else
    client.println("noEvents");
  client.println("</status>");
}

void AddressSetup(EthernetClient client)
{
  TextFinder  finder(client );
  // if you find the word "SBM" continue looking for more
  // if you don't find that word, stop looking and go further
  // it means the SUBMIT button hasn't been pressed and nothing has
  // been submitted. Just go to the place where the setup page is
  // been build and show it in the client's browser.
  if (finder.findUntil("SBM", "\n\r")){
```

```
    byte SET = finder.getValue();
    // Now while you are looking for the letters "DT", you'll have to
remember
    // every number behind "DT" and put them in "val" and while doing
so, check
    // for the according values and put those in mac, ip, subnet and
gateway.
    while(finder.findUntil("DT", "\n\r")){
      int val = finder.getValue();
      // if val from "DT" is between 1 and 6 the according value must
be a MAC value.
      if(val >= 1 && val <= 6) {
        mac[val - 1] = finder.getValue();
      }
      // if val from "DT" is between 7 and 10 the according value must
be a IP value.
      if(val >= 7 && val <= 10) {
        ip[val - 7] = finder.getValue();
      }
      // if val from "DT" is between 11 and 14 the according value
must be a MASK value.
      if(val >= 11 && val <= 14) {
        subnet[val - 11] = finder.getValue();
      }
      // if val from "DT" is between 15 and 18 the according value
must be a GW value.
      if(val >= 15 && val <= 18) {
        gateway[val - 15] = finder.getValue();
      }
      // if val from "DT" is 19 the according value must be a PORT
value.
      if(val == 19) {
        port = finder.getValue();
      }
    }
    // Now that we got all the data, we can save it to EEPROM
    for (int i = 0 ; i < 6; i++){
      EEPROM.write(i + 1,mac[i]);
    }
    for (int i = 0 ; i < 4; i++){
      EEPROM.write(i + 7, ip[i]);
    }
    for (int i = 0 ; i < 4; i++){
      EEPROM.write(i + 11, subnet[i]);
    }
    for (int i = 0 ; i < 4; i++){
      EEPROM.write(i + 15, gateway[i]);
    }
    EEPROMWriteInt(19, port);

    // set ID to the known bit, so when you reset the Arduino will use
the EEPROM values
```

```
    EEPROM.write(0, 0x92);
    // if al the data has been written to EEPROM we should reset the
arduino
    // for now you'll have to use the hardware reset button
  }

  // and from this point on, we can start building our setup page
  // and show it in the client's browser.
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println();
  for (int i = 0; i < 4; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table0[i])));
    client.print( buffer );
  }
  for (int i = 0; i < 3; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[i])));
    client.print( buffer );
  }
  client.print(mac[0],HEX);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[3])));
  client.print( buffer );
  client.print(mac[1],HEX);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[4])));
  client.print( buffer );
  client.print(mac[2],HEX);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[5])));
  client.print( buffer );
  client.print(mac[3],HEX);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[6])));
  client.print( buffer );
  client.print(mac[4],HEX);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table1[7])));
  client.print( buffer );
  client.print(mac[5],HEX);
  for (int i = 0; i < 4; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table2[i])));
    client.print( buffer );
  }
  client.print(ip[0],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table2[4])));
  client.print( buffer );
  client.print(ip[1],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table2[5])));
  client.print( buffer );
  client.print(ip[2],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table2[6])));
  client.print( buffer );
  client.print(ip[3],DEC);
```

```
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table2[7])));
  client.print( buffer );
  client.print(port,DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table3[0])));
  client.print( buffer );
  client.print(subnet[0],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table3[1])));
  client.print( buffer );
  client.print(subnet[1],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table3[2])));
  client.print( buffer );
  client.print(subnet[2],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table3[3])));
  client.print( buffer );
  client.print(subnet[3],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[0])));
  client.print( buffer );
  client.print(gateway[0],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[1])));
  client.print( buffer );
  client.print(gateway[1],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[2])));
  client.print( buffer );
  client.print(gateway[2],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[3])));
  client.print( buffer );
  client.print(gateway[3],DEC);
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[4])));
  client.print( buffer );
  for (int i = 0; i < 7; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table5[i])));
    client.print( buffer );
  }
  strcpy_P(buffer, (char*)pgm_read_word(&(string_table4[5])));
  client.print( buffer );
}

void AddressConfirm(EthernetClient client)
{
  client.println("HTTP/1.1 200 OK");
  client.println("Content-Type: text/html");
  client.println();
  // put your own html from here on
  for (int i = 0; i < 4; i++)
  {
    strcpy_P(buffer, (char*)pgm_read_word(&(string_table6[i])));
    client.print( buffer );
  }
}
```

```
//This function will write a 2 byte integer to the eeprom at the
specified address and address + 1
void EEPROMWriteInt(int p_address, int p_value)
{
  byte lowByte = ((p_value >> 0) & 0xFF);
  byte highByte = ((p_value >> 8) & 0xFF);

  EEPROM.write(p_address, lowByte);
  EEPROM.write(p_address + 1, highByte);
}

//This function will read a 2 byte integer from the eeprom at the
specified address and address + 1
unsigned int EEPROMReadInt(int p_address)
{
  byte lowByte = EEPROM.read(p_address);
  byte highByte = EEPROM.read(p_address + 1);

  return ((lowByte << 0) & 0xFF) + ((highByte << 8) & 0xFF00);
}

void loop()
{
  EthernetServer server(port);
  EthernetClient client = server.available();
  if (client)
  {
    WaitForRequest(client);
    ParseReceivedRequest();
    PerformRequestedCommands(client);
    client.stop();
  }
  ProcessChip(0);
  ProcessChip(1);
  ProcessChip(2);
  ProcessChip(3);
  ProcessChip(4);
  ProcessChip(5);
  ProcessChip(6);
  ProcessChip(7);
  PrintTable();
  PrintMove();
  AckReset();
  Alarms();
}
```

Ο κώδικας του προγράμματος C#

A) Form1 class

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Globalization;
using System.Linq;
using System.Net;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;

namespace I2C_io_control
{
    public partial class Form1 : Form
    {
        private String alarmState;
        private List<string> portNames = new List<string>();
        private int ackPorts = 0;

        public Form1()
        {
            InitializeComponent();
            timer1.Enabled = false;
            refreshStatus();
        }

        private void refreshStatus()
        {
            readArduinoStatus();
            //populate list
            listView1.Items.Clear();
            foreach (string elem in portNames)
            {
                listView1.Items.Add(new ListViewItem(elem));
            }
            for (int i = 0; i<listView1.Items.Count; i++)
            {
                if (i < ackPorts)
                {
                    listView1.Items[i].BackColor = Color.Orange;
                }
                else
                {
                    listView1.Items[i].BackColor = Color.Red;
                }
            }

            if (alarmState=="Buzzer")
            {
```

```
                alarmsLabel.Text = "New alarms";
                alarmsLabel.BackColor = Color.Red;
            }
            else if (alarmState == "ackEvents")
            {
                alarmsLabel.Text = "Acknowledged";
                alarmsLabel.BackColor = Color.Orange;
            }
            else
            {
                alarmsLabel.Text = "No events";
                alarmsLabel.BackColor = Color.LimeGreen;
            }
            if (alarmState == "ackEvents")
            {
                button2.Enabled = true;
            }
            else
            {
                button2.Enabled = false;
            }
        }

        private void readArduinoStatus()
        {
           alarmState =  alarmStatus();
           getPortNamesList();
        }

        private String alarmStatus()
        {
            String url = "http://" +
              ValuesFromStorage.getInstance().getArduinoIP();
            url += ":" + ValuesFromStorage.getInstance().getArduinoPort();
            url += "/alarmStatus/";
            // Retrieve XML document
            XmlTextReader reader = new XmlTextReader(url);
            // Skip non-significant whitespace
            reader.WhitespaceHandling = WhitespaceHandling.Significant;
            // Read nodes one at a time
            string elementName="";
            string elementValue="";
            try
            {
                while (reader.Read())
                {
                    switch (reader.NodeType)
                    {
                        case XmlNodeType.Element:
                            if (reader.Name == "userAlarm")
                            {
                                elementName = reader.Name;
                            }
```

```
                        else
                        {
                            elementName = "";
                        }
                        break;
                    case XmlNodeType.Text:
                        if (elementName == "userAlarm")
                        {
                            elementValue = reader.Value;
                        }
                        break;
                }
                if (elementValue.Length > 0)
                {
                    break;
                }
            }
            timer1.Enabled = true;
        }
        catch (WebException e)
        {
          MessageBox.Show("Πρόβλημα κατά τη σύνδεση με τον Arduino! Παρακαλώ
           ελέγξτε το αρχείο settings.xml!",
          "Σφάλμα κατά τη σύνδεση!",
          MessageBoxButtons.OK,
          MessageBoxIcon.Error,
          MessageBoxDefaultButton.Button1);
          Application.Exit();
        }
        return elementValue;
    }

    private void getPortNamesList()
    {
        String url = "http://" +
          ValuesFromStorage.getInstance().getArduinoIP();
        url += ":" + ValuesFromStorage.getInstance().getArduinoPort();
        url += "/digitalStatus/";
        // Retrieve XML document
        XmlTextReader reader = new XmlTextReader(url);
        // Skip non-significant whitespace
        reader.WhitespaceHandling = WhitespaceHandling.Significant;
        // Read nodes one at a time
        string elementName = "";
        string elementValue = "";
        portNames.Clear();
        try
        {
            while (reader.Read())
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
```

```
                        if (reader.Name == "alarmPorts" || reader.Name ==
                          "ackPorts" )
                        {
                            elementName = reader.Name;
                        }
                        else
                        {
                            elementName = "";
                        }
                        break;
                    case XmlNodeType.Text:
                        if (elementName == "alarmPorts" )
                        {
                            elementValue = reader.Value;
                            if (elementValue.Length > 0)
                            {
                                String[] temp = elementValue.Split('@');
                                for (int i = 0; i < temp.Length; i++)
                                {
                                    portNames.Add(temp[i]);
                                }
                            }
                        }
                        if (elementName == "ackPorts")
                        {
                            elementValue = reader.Value;
                            ackPorts = 0;
                            ackPorts = Int16.Parse(elementValue);
                        }
                        break;
                }
            }
        }
        catch (WebException e)
        {
            MessageBox.Show("Πρόβλημα κατά τη σύνδεση με τον Arduino!
              Παρακαλώ ελέγξτε το αρχείο settings.xml!",
           "Σφάλμα κατά τη σύνδεση!",
           MessageBoxButtons.OK,
           MessageBoxIcon.Error,
           MessageBoxDefaultButton.Button1);
            Application.Exit();
        }
    }

    private void acknowledge()
    {
        String url = "http://" +
          ValuesFromStorage.getInstance().getArduinoIP();
        url += ":" + ValuesFromStorage.getInstance().getArduinoPort();
        url += "/acknowledge/";
        WebRequest webRequest = WebRequest.Create(url);
        webRequest.Proxy = null;
```

```
            using (var response = (HttpWebResponse)webRequest.GetResponse())
            {
            }
        }

        private void reset()
        {
            String url = "http://" +
              ValuesFromStorage.getInstance().getArduinoIP();
            url += ":" + ValuesFromStorage.getInstance().getArduinoPort();
            url += "/reset/";
            WebRequest webRequest = WebRequest.Create(url);
            webRequest.Proxy = null;
            using (var response = (HttpWebResponse)webRequest.GetResponse())
            {
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            acknowledge();
            refreshStatus();
        }

        private void timer1_Tick(object sender, EventArgs e)
        {
            refreshStatus();
        }

        private void refreshInterval_ValueChanged(object sender, EventArgs e)
        {
            int value = (int)refreshInterval.Value;
            timer1.Interval = value  * 1000;
        }

        private void button2_Click(object sender, EventArgs e)
        {
            reset();
            refreshStatus();
        }
    }
}
```

B) Initial class

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace I2C_io_control
{
    public partial class initial : Form
    {
        public initial()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            Form1 form = new Form1();
            form.ShowDialog();
        }
    }
}
```

Γ) ValuesFromStorage Class

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;

namespace I2C_io_control
{
    class ValuesFromStorage
    {
        private string arduinoIP, arduinoPort;
        private static ValuesFromStorage instance;

        public static ValuesFromStorage getInstance(){
            if (instance == null)
            {
                instance = new ValuesFromStorage();
                instance.readFromStorage();
            }
            return instance;
        }

        private void readFromStorage()
        {
            arduinoIP = readPropertyFromXML("arduino_ip");
            arduinoPort = readPropertyFromXML("arduino_port");
        }

        public string getArduinoIP()
        {
            return arduinoIP;
        }
        public string getArduinoPort()
```

```
        {
            return arduinoPort;
        }

        private String readPropertyFromXML(String propertyName)
        {
            String url = "./settings.xml";
            // Retrieve XML document
            XmlTextReader reader = new XmlTextReader(url);
            // Skip non-significant whitespace
            reader.WhitespaceHandling = WhitespaceHandling.Significant;
            // Read nodes one at a time
            string elementName="";
            string elementValue="";
            while (reader.Read())
            {
                switch (reader.NodeType)
                {
                    case XmlNodeType.Element:
                        if(reader.Name == propertyName){
                            elementName = reader.Name;
                        }
                        else{
                            elementName = "";
                        }
                        break;
                    case XmlNodeType.Text:
                        if (elementName == propertyName)
                        {
                            elementValue = reader.Value;
                        }
                        break;
                }
                if (elementValue.Length>0)
                {
                    break;
                }
            }
            return elementValue;
        }
    }
}
```