ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

Μεταπτυχιακό Πρόγραμμα Σπουδών

Επιστήμη της Τεχνολογίας και Πληροφορικής Υπολογιστών

Τίτλος Διπλωματικής Εργασίας

Μετατροπή αυτοματισμού ενσωματωμένου συστήματος σε Web Server
και πλήρης απομακρυσμένος έλεγχος του, μέσω WiFi,
με την χρήση μικροελεγκτή σειράς ATmega, για βιομηχανική και προσωπική χρήση

Συγγραφέας

Γεωργιάδης Κοσμάς – Απόστολος
ΑΜ: 19057

Επιβλέπων Καθηγητής

Βογιατζής Ιωάννης

Αθήνα, Ιούλιος 2021

Διπλωματική Εργασία

Μετατροπή αυτοματισμού ενσωματωμένου συστήματος
σε Web Server και πλήρης απομακρυσμένος έλεγχος του,
μέσω WiFi, με την χρήση μικροελεγκτή σειράς ATmega,
για βιομηχανική και προσωπική χρήση.

Γεωργιάδης Κοσμάς Απόστολος
ΑΜ: 19057

Η μεταπτυχιακή διπλωματική εργασία εξετάστηκε επιτυχώς από την κάτωθι Εξεταστική Επιτροπή:

ΨΗΦΙΑΚΗ ΥΠΟΓΡΑΦΗ	
Panagiotis Giannakopoulos	Digitally signed by Panagiotis Giannakopoulos Date: 2021.09.08 15:21:55 +03'00'
Stavros Fatouros	Digitally signed by Stavros Fatouros Date: 2021.09.27 08:35:05 +03'00'
Ioannis Voyiatzis	Digitally signed by Ioannis Voyiatzis Date: 2021.09.24 10:37:35 +03'00'

Ημερομηνία Εξέτασης: 28/7/2021

Αθήνα, Ιούλιος 2021

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο κάτωθι υπογεγραμμένος **Γεωργιάδης Κοσμάς Απόστολος** του **Γεωργίου**, με αριθμό μητρώου **19057** φοιτητής του Προγράμματος Μεταπτυχιακών Σπουδών «**Επιστήμη και Τεχνολογία της Πληροφορικής και των Υπολογιστών» του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών**» της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

«Είμαι συγγραφέας αυτής της μεταπτυχιακής εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος.

Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

Ο Δηλών

Γεωργιάδης Κοσμάς – Απόστολος

Αθήνα, Ιούλιος 2021

Περίληψη

Σκοπός της διπλωματικής εργασίας αυτής είναι η σχεδίαση και η υλοποίηση ενός ενσωματωμένου συστήματος, με την δυνατότητα πλήρη απομακρυσμένου ελέγχου του, μέσω WiFi. Στις μέρες μας, κυριαρχεί η εποχή του Internet of Things (IoT), που σημαίνει ότι όλο και περισσότερες συσκευές έχουν την δυνατότητα να συνδέονται στο Διαδίκτυο και να ελέγχονται απομακρυσμένα, δίνοντας έτσι στον χρήστη την δυνατότητα να μην χρειάζεται να είναι παρόν για τον έλεγχο του συστήματος. Με την εξέλιξη της τεχνολογίας, μπορεί οποιοσδήποτε να ασχοληθεί με αυτήν την τεχνολογία και να υλοποιήσει ένα δικό του σύστημα προσφέροντας έτσι, καινοτομία στον τομέα αυτόν. Η διπλωματική εργασία δίνει έμφαση στην υλοποίηση ενός συστήματος χωρίς χρήση λογισμικού ή κώδικα τρίτων. Αυτό, γιατί άλλος ένας στόχος της εργασίας αυτής είναι και η καινοτομία. Όπως αναφέραμε πριν, επειδή οποιοσδήποτε μπορεί να δημιουργήσει δικά του συστήματα, με χρήση πάντα υλικού και λογισμικού τρίτων, έχει δημιουργηθεί ένας κορεσμός όσον αφορά την πρωτοτυπία και ποικιλία, παρότι υπάρχουν πολλά υλοποιημένα συστήματα. Γι' αυτό λοιπόν, επιλέξαμε να σχεδιάσουμε το σύστημα μας όσο πιο ανεξάρτητο γίνεται, από υλικό και λογισμικό τρίτων. Συγκεκριμένα, η εργασία αυτή στοχεύει στην μετατροπή ενός απλού ενσωματωμένου συστήματος, σε Web Server που θα μπορεί να δέχεται και να επεξεργάζεται HTTP αιτήματα από πελάτες. Οι πελάτες θα έχουν την δυνατότητα να έχουν πλήρης έλεγχο του συστήματος μέσω ιστοσελίδας που θα βρίσκεται στον μικροελεγκτή του αυτοματισμού. Ο έλεγχος μπορεί να είναι είτε προβολή μετρήσεων από αισθητήρια όργανα συνδεδεμένα στο ίδιο το σύστημα, ή μία απλή ενεργοποίηση/απενεργοποίηση μίας λυχνίας LED. Για την υλοποίηση του υλικού κομματιού της εργασίας μπορεί να χρησιμοποιηθεί οποιαδήποτε πλακέτα με κάποιες κατάλληλες προδιαγραφές. Τέτοιου είδους πλακέτες υπάρχουν σε αφθονία στο Διαδίκτυο, οπότε ο χρήστης είναι ελεύθερος να επιλέξει μέσα από μία μεγάλη ποικιλία. Για το κομμάτι του λογισμικού, δεν γίνεται απολύτως καμία χρήση κώδικα ή βιβλιοθηκών τρίτων. Βεβαίως δεν αποθαρρύνουμε την χρήση τους, αλλά όπως είπαμε σκοπός μας είναι η σχεδίαση ενός πρωτότυπου συστήματος.

Λέξεις κλειδιά: ενσωματωμένο, σύστημα, απομακρυσμένος, έλεγχος, μικροελεγκτής, web, server, wifi, atmega, esp

Αθήνα, Ιούλιος 2021

Abstract

The purpose of this dissertation is the design and implementation of an integrated system, with complete remote control over it, via WiFi. Nowadays, the Internet of Things (IoT) era prevails, which means that more and more devices have the ability to connect to the Web and be controlled remotely, thus allowing the user to no longer need to be present, to control the system. With the evolution of technology, one can learn this technology and implement their own system, thus offering innovation in this field. The dissertation emphasizes the implementation of a system without the use of third party software or code. This is because another goal of this dissertation is innovation. As mentioned before, because anyone can create their own systems, using third-party hardware and software, a saturation in terms of originality and variety has been created, even though there is an abundance of similar implemented systems. Thus, we chose to design our system as independently as possible, from third party hardware and software. Specifically, this work aims to convert a simple embedded system into a Web Server that can accept and process HTTP requests from clients. Clients will be able to have full control of the system through the website located in the system's microcontroller. The type of control can be either a display of measurements from sensory instruments connected to the system itself, or a simple activation / deactivation of an LED lamp. For the implementation of the hardware board, any board can be used with some suitable specifications. Such boards are in abundance on the Internet, so the user is free to choose from a wide variety. For the software part, there is absolutely no use of third-party code or libraries. Of course, we do not discourage their use, but as we said, our purpose is to design an original system.

Keywords: embedded, system, remote, control, microcontroller, web, server, wifi, atmega, esp

Αθήνα, Ιούλιος 2021

Αθήνα, Ιούλιος 2021

Περιεχόμενα

1. Πρόλογος

Σκοπός αυτής της διπλωματικής εργασίας είναι η σχεδίαση και η δημιουργία ενός συστήματος που θα κάνει χρήση του μικροελεγκτή **ATmega32**, σε συνδυασμό με το **Wi-Fi Module ESP8266**, ώστε να μπορεί να γίνει εφικτή η απομακρυσμένη επικοινωνία ενός **client (πελάτη)** με τον ίδιο τον μικροελεγκτή.

Η απομακρυσμένη επικοινωνία θα γίνεται είτε μέσω **LAN** (τοπικού δικτύου) είτε μέσω του **Διαδικτύου** (Internet). Την διαδικασία επικοινωνίας θα την αναλαμβάνει ένας **Web Server** (ιστοσελίδα) που θα βρίσκεται εγκατεστημένος στον ίδιο τον μικροελεγκτή. Αυτή η ενσωμάτωση του Web Server με τον μικροελεγκτή είναι και ο κύριος στόχος αυτής της διπλωματικής εργασίας.

Η χρήση του μικροελεγκτή ATmega32, γίνεται μέσω μίας πειραματικής πλακέτας που έχει σχεδιαστεί για εκπαιδευτικούς σκοπούς και πειραματισμούς από την **G&K Electronics**. Φυσικά, εννοείται ότι μπορεί να γίνει χρήση οποιασδήποτε πλακέτας, αρκεί να υποστηρίζει κάποιες βασικές προδιαγραφές. Περισσότερα γι' αυτό, θα αναφερθούμε στο κατάλληλο κεφάλαιο.

Η επικοινωνία **Πελάτη – ATmega32**, στα πλαίσια αυτής της διπλωματικής εργασίας θα είναι περισσότερο από μία απλή αποστολή αιτήματος – απάντησης. Ο πελάτης θα έχει την δυνατότητα να χειρίζεται σχεδόν πλήρως την κατάσταση του μικροελεγκτή. Συνοπτικά, η διαχείριση περιλαμβάνει:

- **a)** την πλήρη κατάσταση Εισόδου-Εξόδου (I/O) των ακροδεκτών του μικροελεγκτή,
- **b)** την συλλογή δεδομένων από αισθητήρια όργανα,
- **c)** την δυνατότητα επανεκκίνησης του μικροελεγκτή,
- **d)** την λήψη/εκκαθάριση δεδομένων που βρίσκονται στην μνήμη EEPROM του μικροελεγκτή και τέλος,
- **e)** την δυνατότητα επανεκκίνησης του ESP Wi-Fi module.

Με λίγα λόγια, σκοπός αυτής της διπλωματικής εργασίας μας είναι **η ενσωμάτωση και η μετατροπή ενός αυτοματισμού ενσωματωμένου συστήματος (Embedded System) σε έναν Διαδικτυακού Διακομιστή (Web Server)**.

Φυσικά, όπως θα δείτε παρακάτω, περιοριζόμαστε σε θέματα μνήμης και ταχύτητας λόγω της φύσεως των ίδιων των μικροελεγκτών, αλλά το τελικό αποτέλεσμα αρκεί ώστε να μπορεί να γίνει χρήση σε προσωπικό αλλά και σε βιομηχανικό επίπεδο.

Αυτή η διπλωματική εργασία στοχεύει, κυρίως, σε φοιτητές Τμημάτων Πληροφορικής, προγραμματιστές, ηλεκτρονικούς, αλλά και σε όσους ασχολούνται και θέλουν να μάθουν περισσότερα για τον κόσμο των **Μικροελεγκτών** και **Αυτοματισμών**.

2. Εισαγωγή στους Μικροελεγκτές

2.1 Τι είναι ένας μικροελεγκτής;

Ένας **μικροελεγκτής** είναι ένας μικρός επεξεργαστής, σαν αυτόν που έχει ο Η/Υ μας, που λειτουργεί με ελάχιστο αριθμό ηλεκτρονικών εξαρτημάτων.

Διαθέτει επεξεργαστή, μνήμη, διάφορα περιφερειακά κυκλώματα, καθώς επίσης και θύρες εισόδου/εξόδου **(I/O)** για την επικοινωνία με εξωτερικές συσκευές. Θα μπορούσε να παρομοιαστεί με έναν μικροϋπολογιστή.

Όπως ακριβώς ένας ηλεκτρονικός υπολογιστής έχει επεξεργαστή, μνήμη, περιφερειακές συσκευές και εκτελεί προγράμματα, έτσι κι ένας μικροελεγκτής διαθέτει τα παραπάνω χαρακτηριστικά και μάλιστα σε ένα μόνο ολοκληρωμένο κύκλωμα (**chip**).

Ο μικροελεγκτής είναι ο «**εγκέφαλος**» του συστήματος. Οι μετρητές, χρονιστές, αισθητήρες και τα λοιπά όργανα μέτρησης, είναι τα εργαλεία που χρειάζεται ο «εγκέφαλος» για να «**αντιληφθεί**» το περιβάλλον γύρω του και να **επιδράσει** σε αυτό.

Ο μικροελεγκτής όμως, χρειάζεται κάτι να του «πει» **τί** θα κάνει και **πότε** να το κάνει. Εδώ μπαίνουμε στο δεύτερο και κυριότερο κομμάτι των μικροελεγκτών.

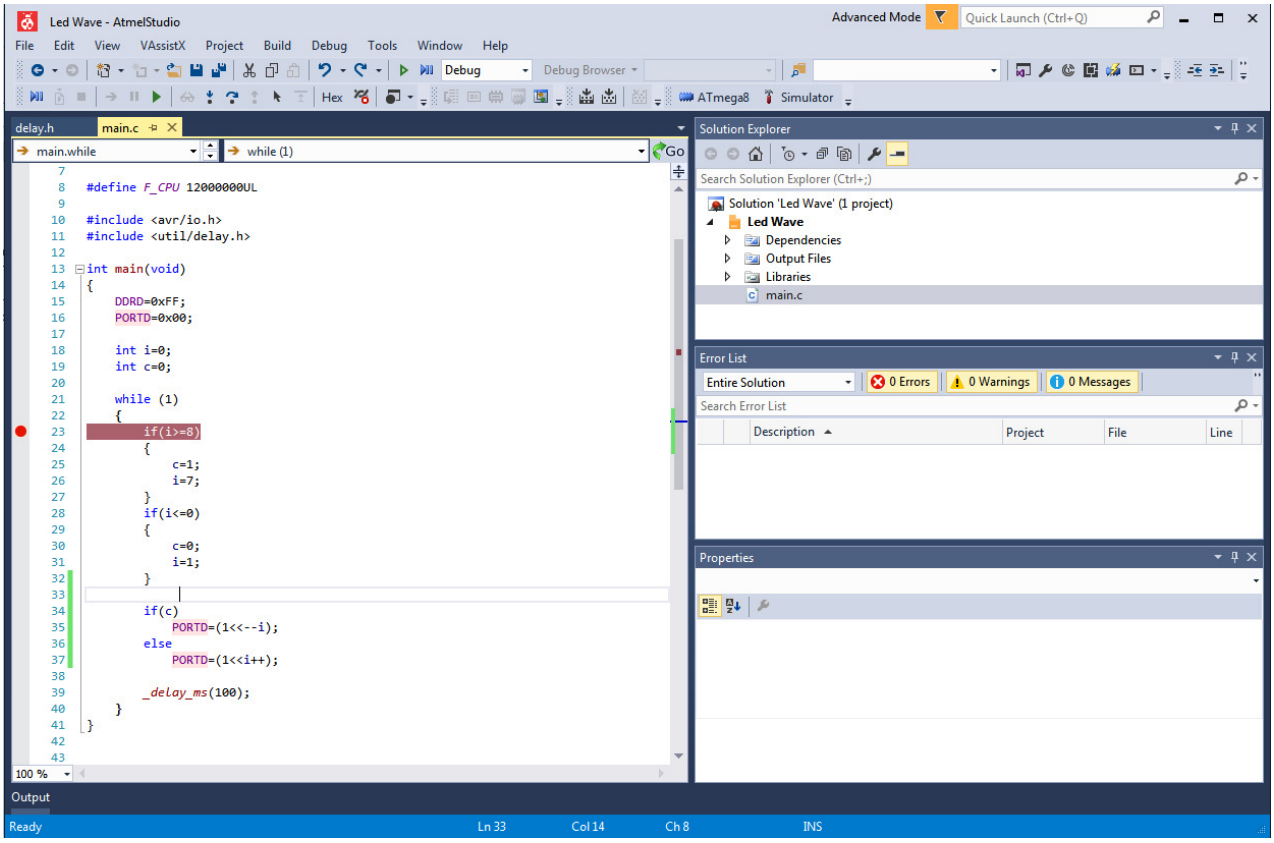
2.2 Προγραμματισμός μικροελεγκτών

Ένα πρόγραμμα, όπως γνωρίζουμε, είναι ένα πεπερασμένο σύνολο εντολών που εκτελείται με προκαθορισμένη σειρά σε ένα υπολογιστικό σύστημα. Έτσι λοιπόν, ένας μικροελεγκτής για να μπορέσει να εκτελέσει όλες τις ενέργειες και τις λειτουργίες που θέλουμε, πρέπει να τον προγραμματίσουμε, δηλαδή να του φορτώσουμε ένα πρόγραμμα, με κατάλληλα διαμορφωμένες εντολές.

Όταν πρωτοεμφανίστηκαν οι μικροελεγκτές, κύρια γλώσσα προγραμματισμού τους ήταν η **Assembly**. Στα μέσα της δεκαετίας του 80’ όμως, η **C**, προτιμήθηκε πιο πολύ από την Assembly επειδή ήταν πιο εύκολη, πιο κατανοητή αλλά και εξίσου «ισχυρή». Μερικές άλλες γλώσσες προγραμματισμού μικροελεγκτών είναι οι: **microC** και η **BASCOM** (ειδική έκδοση της BASIC). Στις μέρες μας όμως, από όλες τις παραπάνω, η **C** είναι η προτιμότερη και η πιο διάσημη για προγραμματισμό μικροελεγκτών.

Για τη δημιουργία ενός προγράμματος για μικροελεγκτές, είναι απαραίτητος ένας συντάκτης (editor) για την συγγραφή του κώδικα και ένας «**Μεταγλωττιστής**» (compiler) όπου θα μετατρέψει τον κώδικα του προγράμματος, σε εκτελέσιμο κώδικα **(Γλώσσα Μηχανής)** για τον μικροελεγκτή.

Η εταιρεία **Atmel,** ή **Microchip** όπως ονομάζεται τώρα, δημιουργός της σειράς μικροελεγκτών **ATmega**, έχει αναπτύξει την δικιά της πλατφόρμα για προγραμματισμό μικροελεγκτών. Η πλατφόρμα ονομάζεται **Atmel Studio** (παλαιότερα **AVR Studio**), όπου εδώ και μερικά χρόνια, βασίζεται στην διάσημη πλατφόρμα προγραμματισμού της **Microsoft**, την **Visual Studio**. Παρακάτω παρατίθεται μια γενική εικόνα της πλατφόρμας.

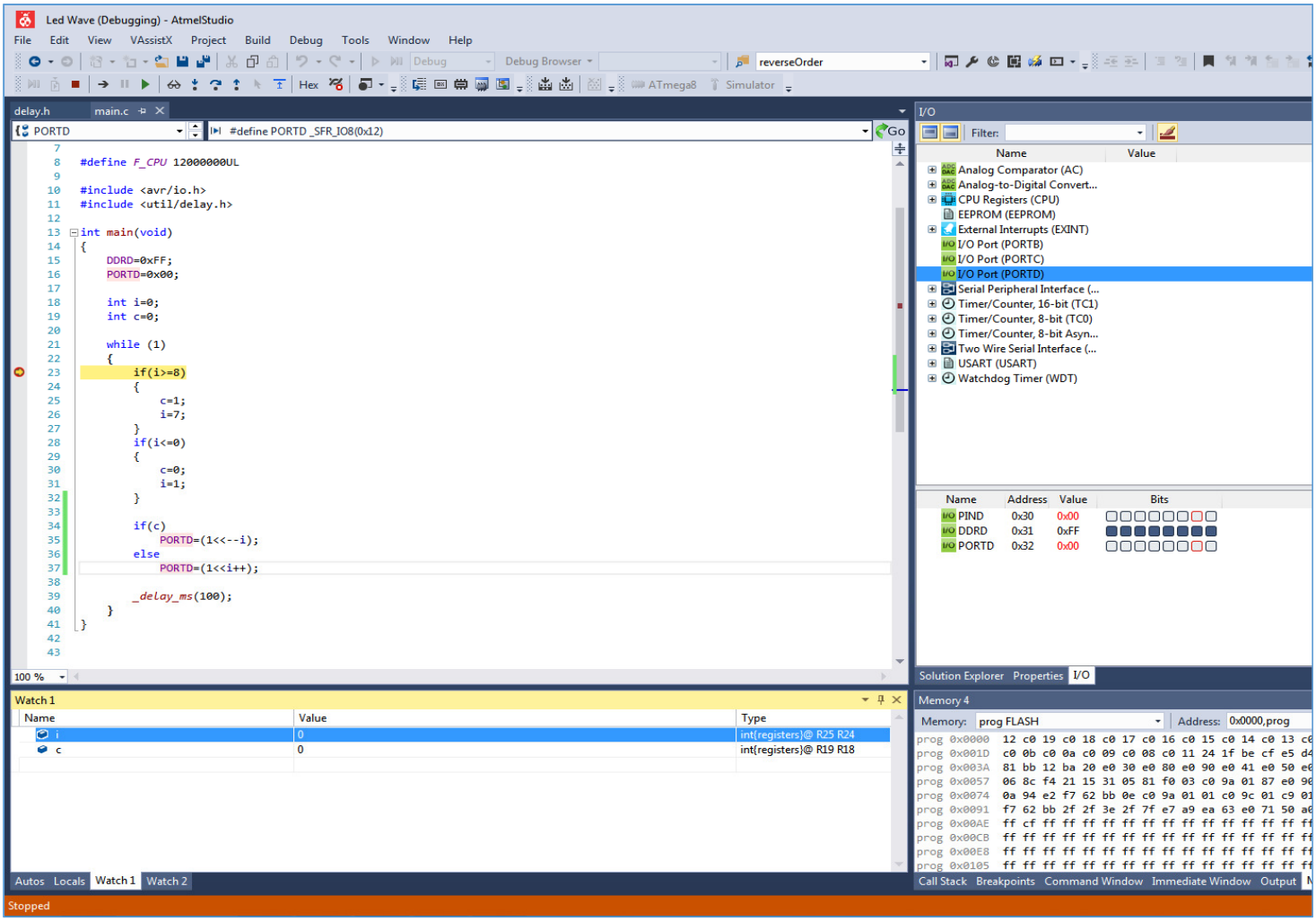


Εικόνα 2.2.1. Πλατφόρμα Atmel Studio.

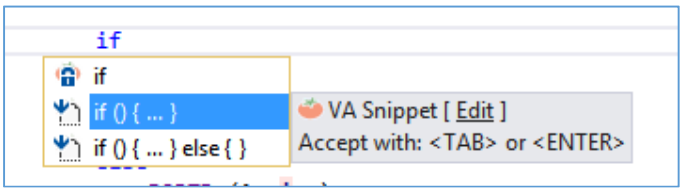
Οποιαδήποτε εμπειρία χρήσης της πλατφόρμας Visual Studio, σίγουρα θα ωφελήσει τον χρήστη που θα ασχοληθεί με την πλατφόρμα Atmel Studio**.** Η ένωση με το Visual Studio φέρνει πολλά πλεονεκτήματα στην πλατφόρμα. Μερικά από αυτά είναι:

1. Καλύτερη διαχείριση και οργάνωση πολλαπλών **Project** ταυτόχρονα.
2. Καλύτερη αποσφαλμάτωση κώδικα **(debugging)**. Γίνεται προσομοίωση εκτέλεσης του προγράμματος και ταυτόχρονα η προβολή του περιεχομένου του μικροεπεξεργαστή (Flash, EEPROM κλπ.).
3. Βοήθεια συγγραφής κώδικα (αυτόματη συμπλήρωση εντολών, εμφάνιση ειδοποιήσεων και σφαλμάτων κατά την συγγραφή κλπ.).
4. Εισαγωγή **custom** προσθέτων (**extensions**), από βιβλιοθήκες κώδικα έως και εργαλεία αποσφαλμάτωσης.

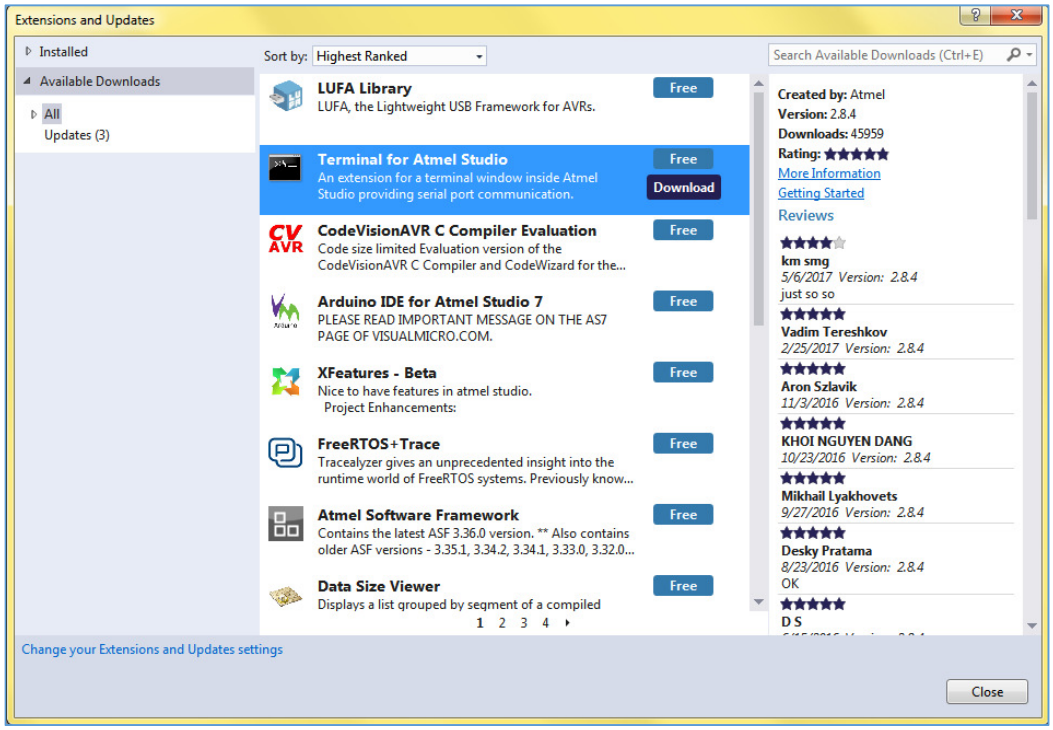
5. Πλήρης προσαρμογή **(customization)** περιβάλλοντος από τον χρήστη. Αλλαγή θέσεων παραθύρων, χρώματα, γραμματοσειρές, συντομεύσεις πληκτρολογίου, μεγέθυνση/σμίκρυνση παραθύρου και πολλά άλλα.



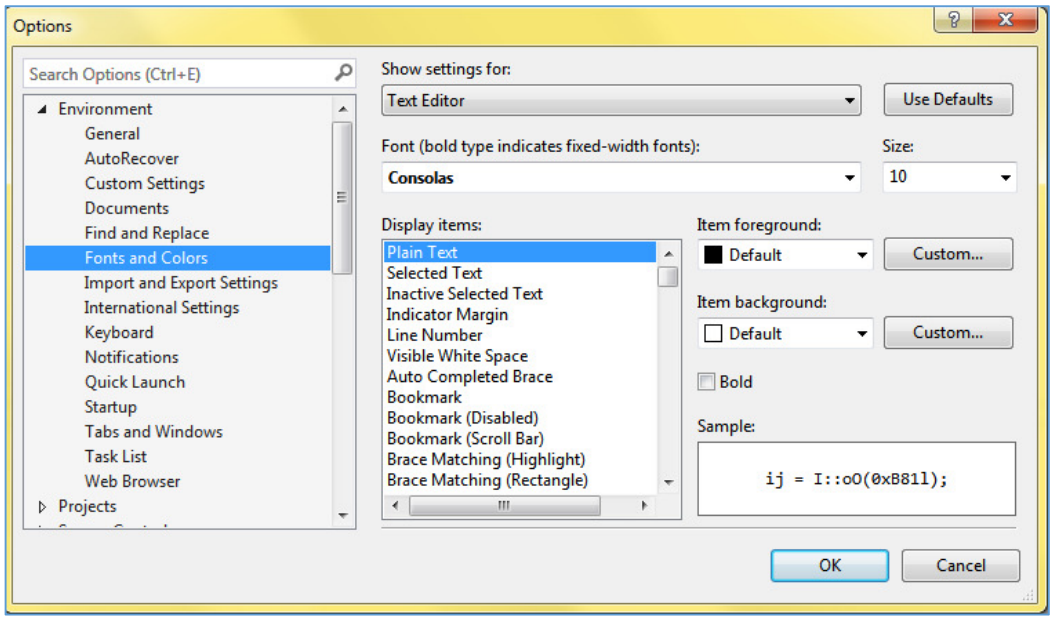
Εικόνα 2.2.2. Debugging στην πλατφόρμα Atmel Studio.



Εικόνα 2.2.3. Αυτόματη συμπλήρωση κώδικα στην πλατφόρμα Atmel Studio.



Εικόνα 2.2.4. Εισαγωγή προσθέτων στην πλατφόρμα Atmel Studio.



Εικόνα 2.2.5. Προσαρμογή πλατφόρμας Atmel Studio.

Η πλατφόρμα Atmel Studio προσφέρει όλα τα κατάλληλα εργαλεία για την δημιουργία ενός προγράμματος σε γλώσσα **C, C++** και **Assembly**, μέχρι και την μετάφραση του σε γλώσσα μηχανής και τέλος, τον προγραμματισμό του μικροελεγκτή (φόρτωση του προγράμματος).

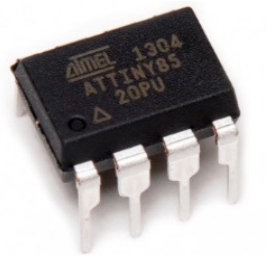
2.3 Λίγα λόγια για την Atmel/Microchip

Η εταιρεία **Atmel** είναι μία από τίς πρώτες εταιρείες που έκανε μαζική παραγωγή μικροελεγκτών όχι μόνος για βιομηχανίες αλλά και για το κοινό, είτε είναι φοιτητές, ηλεκτρονικοί, προγραμματιστές και γενικά για όλους όσους ήθελαν να ασχοληθούν περισσότερο με αυτόν τον τομέα.

Σήμερα συνεχίζει να πρωτοπορεί, δημιουργώντας καλύτερους, μικρότερους, πιο φιλικούς, ως προς την κατανάλωση ενέργειας, μικροελεγκτές, όπου αρχίζουν και χρησιμοποιούνται όλο και πιο πολύ σε αυτοματισμούς όλων των ειδών.

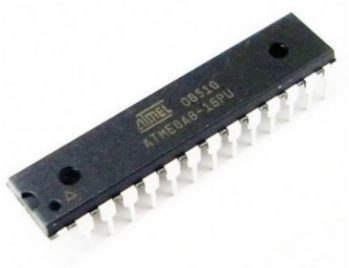
Μερικοί διάσημοι μικροελεγκτές της **Atmel** είναι οι εξής:

1. **ATtiny25/45/85**. Οι μικροελεγκτές της σειράς **ATtiny** είναι πολύ μικροί τόσο σε μέγεθος όσο και κόστος. Ένα επιπλέον χαρακτηριστικό είναι ότι καταναλώνουν ελάχιστη ενέργεια γι' αυτό και προτιμούνται απο βιομηχανίες, όπου ο χώρος και η κατανάλωση ενέργειας είναι υψίστης σημασίας.



Εικόνα 2.3.1. Μικροελεγκτής σειράς ATTINY.

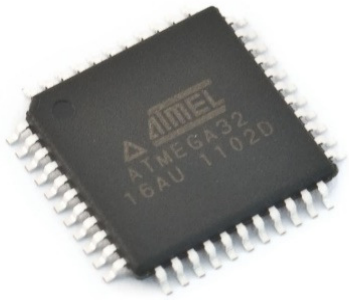
2. **ATmega8/48/88/128/328**. Οι μικροελεγκτές της σειράς **ATmega** είναι ιδανικοί για ερασιτέχνες και για εκπαιδευτικούς σκοπούς, γιατί προσφέρουν πολλές δυνατότητες με πολύ μικρό κόστος. Επίσης, στο διαδίκτυο υπάρχουν εκατοντάδες παραδείγματα κώδικα για αυτήν την σειρά μικροελεγκτών, κάνοντας τους ιδανικούς για φοιτητές και για όσους ασχολούνται για πρώτη φορά.
3. **ATmega32/64/644**. Αυτοί οι συγκεκριμένοι **ATmega** μικροελεγκτές



Εικόνα 2.3.2. Μικροελεγκτής σειράς ATMEGA.

προσφέρουν σχεδόν τα ίδια με τους προηγούμενους αλλά με την διαφορά ότι έχουν περισσότερους ακροδέκτες για ελεύθερη χρήση. Άρα όταν ο χρήστης

περιορίζεται λόγω μη διαθέσιμων ακροδεκτών και δεν θέλει να αλλάξει την σειρά ATmega**,** οι μικροελεγκτές αυτοί είναι φτιαγμένοι γι’ αυτόν το σκοπό.

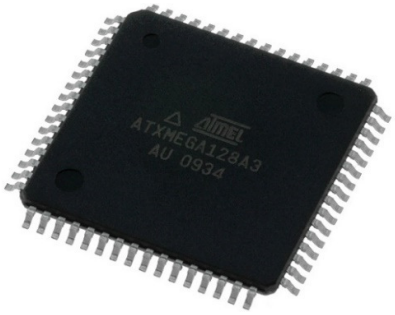


Εικόνα 2.3.3. Μικροελεγκτής ATMEGA32.

4. **XMEGA.** Οι μικροελεγκτές της σειράς **XMEGA** είναι από τους πιο πρόσφατους και προσφέρουν μια πληθώρα εργαλείων και βελτιώσεων από τις προηγούμενες σειρές μικροελεγκτών. Είναι ενσωματωμένοι με όλα τα κατάλληλα πρωτόκολλα για την σύνδεση με το διαδίκτυο μέσω θύρας ETHERNET αλλά και για σύνδεση με USB.

Παραδείγματα χρήσης μικροελεγκτών **Atmel**: σε εγκεφάλους αυτοκινήτων, αεροπλάνων, σε βιομηχανικές εγκαταστάσεις για έλεγχο της παραγωγής και εκατοντάδες άλλους τομείς που μπορεί να μην γνωρίζουμε καν.

Επίσης, εδώ και μερικά χρόνια, άρχισαν να χρησιμοποιούνται και στα κινητά καθώς είναι πολύ ισχυροί σε υπολογιστική ισχύ, μικροί σε μέγεθος αλλά και σε κατανάλωση ενέργειας, δεν υπερθερμαίνονται εύκολα και το κόστος είναι πολύ χαμηλό σε σύγκριση με τους επεξεργαστές άλλων γνωστών κατασκευαστών **(AMD, Intel)**.



Εικόνα 2.3.4. Μικροελεγκτής σειράς XMEGA.

3. Αρχιτεκτονικές σχεδίασης επεξεργαστών

Υπάρχουν δύο αρχιτεκτονικές για την σχεδίαση επεξεργαστών. Η αρχιτεκτονική **CISC** και η αρχιτεκτονική **RISC**.

Η αρχιτεκτονική **CISC (Complex Instruction Set Computing/Υπολογισμός Σύνθετου Συνόλου Εντολών)**, σχεδιάστηκε για να μειώσει την κατανάλωση μνήμης από τα προγράμματα. Παρέχει ένα «πλούσιο» ρεπερτόριο σύνθετων εντολών, άρα και περισσότερους υπολογισμούς, αλλά οι εντολές εκτελούνται με πιο αργό ρυθμό σε σύγκριση με την αρχιτεκτονική RISC.

Παραδείγματα χρήσης αρχιτεκτονικής CISC στους πρώτους μικροεπεξεργαστές:

1. **IBM 370/168**
2. **VAX 11/780**
3. **Intel 80486**

Η αρχιτεκτονική **RISC (Reduced Instruction Set Computing/Υπολογισμός Περιορισμένου Συνόλου Εντολών)**, είναι σχεδιασμένη για να εκτελεί απλές εντολές αλλά σε γρήγορο ρυθμό. Αυτή η αρχιτεκτονική χρησιμοποιείται από όλους τους μικροελεγκτές της Atmel.

Παραδείγματα χρήσης αρχιτεκτονικής RISC:

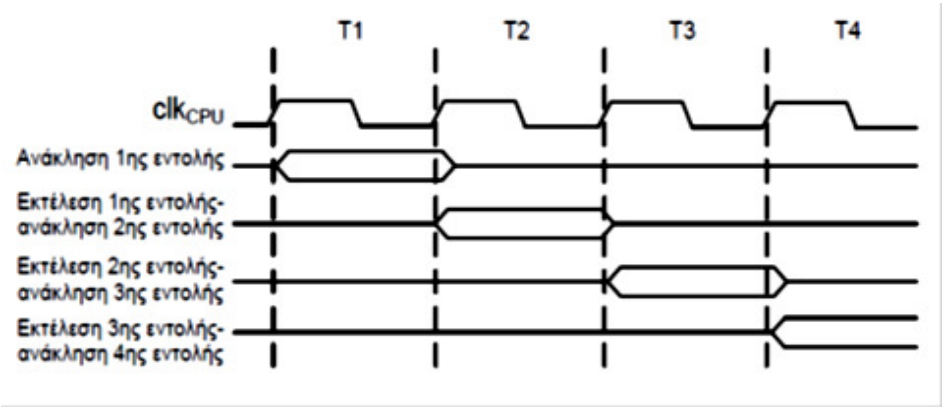
1. **Apple iPod**
2. **Nintendo DS**

Ένα άλλο μοναδικό χαρακτηριστικό της αρχιτεκτονικής RISC, είναι η **Διοχέτευση Εντολών** (**Instruction Pipeline**). Καθώς οι εντολές ενός προγράμματος συνήθως εκτελούνται με την σειρά που είναι τοποθετημένες στην μνήμη, η ΚΜΕ (CPU) του μικροελεγκτή διαθέτει έναν μηχανισμό συνεχής διοχέτευσης εντολών, που λειτουργεί σε δυο στάδια κατά των εξής τρόπο:

1. **Εκτέλεση της εντολής του προγράμματος.**
2. **Ανάκληση της επόμενης εντολής κατά την διάρκεια εκτέλεσης της προηγούμενης.**

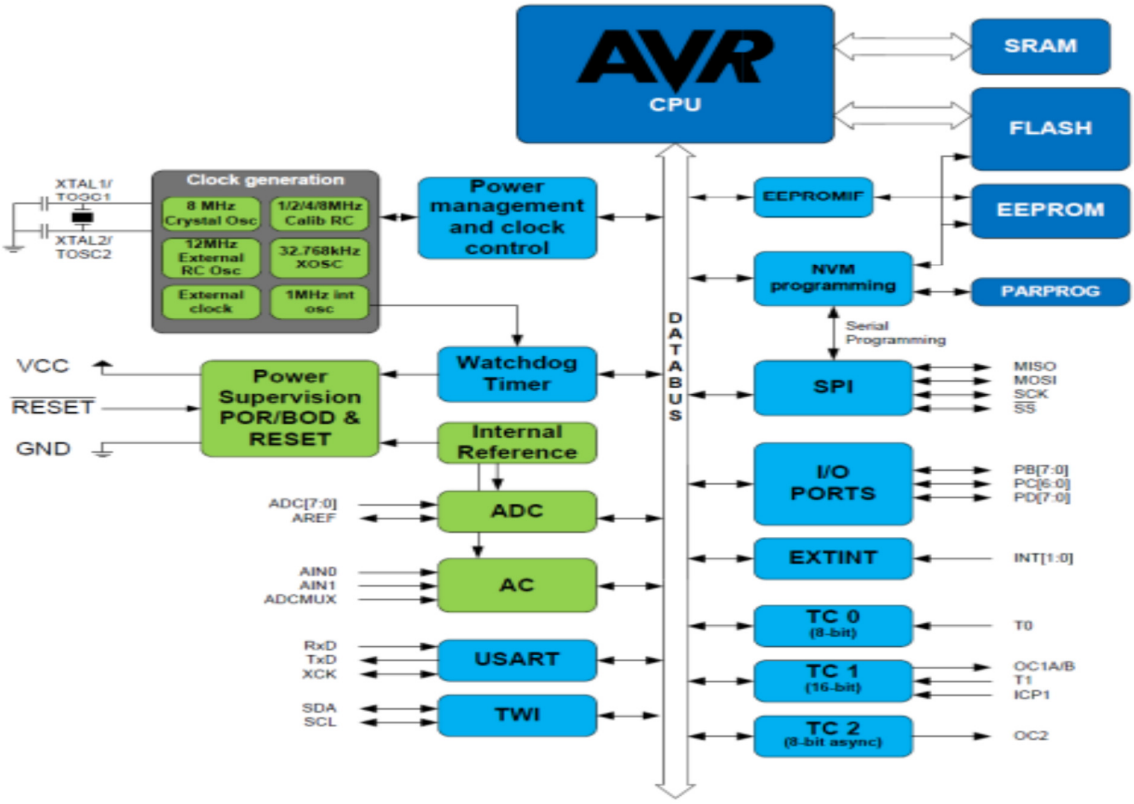
Συμφώνα με τον μηχανισμό αυτό, η ανάκληση και η αποκωδικοποίηση των εντολών, καθώς και η εκτέλεση τους είναι διαδικασίες οι οποίες γίνονται ***ταυτόχρονα***.

Στο παρακάτω σχήμα παρατηρούμε την λειτουργία της συνεχής διοχέτευσης εντολών.



Εικόνα 3.1. Ταυτόχρονη ανάκληση και εκτέλεση εντολών.

Στο παρακάτω σχήμα παρατηρούμε το σχεδιάγραμμα της λειτουργίας αρχιτεκτονικής RISC στους μικροελεγκτές Atmel AVR.



Εικόνα 3.2. Σχεδιάγραμμα της λειτουργίας αρχιτεκτονικής RISC στους μικροελεγκτές Atmel AVR.

Πριν ξεκινήσουμε να περιγράφουμε την πειραματική πλακέτα που θα χρησιμοποιήσουμε στα πλαίσια αυτής της διπλωματικής εργασίας, ας δούμε περιληπτικά κάποιους βασικούς όρους που ίσως μας είναι άγνωστοι.

4. Περιγραφή βασικών όρων

4.1 Τί είναι ένα LED;

Η επίσημη ορολογία είναι η εξής: **LED (Light Emitting Diode/Δίοδος Εκπομπής Φωτός)** αποκαλείται ένας ημιαγωγός ο οποίος εκπέμπει φωτεινή ακτινοβολία στενού φάσματος όταν του παρέχεται μία ηλεκτρική τάση (ρεύμα) κατά τη φορά ορθής πόλωσης (forward-biased).

Με λίγα λόγια είναι ένα μικρό λαμπάκι. Αλλά τι το σημαντικό έχει αυτό το λαμπάκι και το χρησιμοποιούν σε κυκλώματα με μικροελεγκτές και επεξεργαστές;

Μία δίοδος LED μπορεί να φωτίσει με πολύ μικρή τάση ρεύματος (1.8V έως 2.5V), τάσεις πολύ μικρές που χρησιμοποιούνται πολύ συχνά από μικροελεγκτές σαν κύρια τροφοδοσία ρεύματος. Παρόλο που η τάση ρεύματος είναι μικρή, η φωτεινότητα είναι πολύ ισχυρή. Επίσης, το κόστος κατασκευής τους είναι μικρό ειδικά όταν πρόκειται για μαζική παραγωγή.

Εικόνα 4.1.1. Δίοδοι LED σε διαφορετικά χρώματα.

4.2 Τί είναι ένας εξωτερικός κρύσταλλος συχνότητας;

Ένας **Εξωτερικός Κρύσταλλος Συχνότητας** ή αλλιώς **Εξωτερικός Ταλαντωτής**, είναι ένας κρύσταλλος χαλαζία και είναι από τις πιο δημοφιλής επιλογές για τα κυκλώματα χρονισμού. Αυτό οφείλεται στο ότι οι κρύσταλλοι χαλαζία είναι ευρύτατα διαδεδομένοι λόγω του χαμηλού κόστους, λειτουργούν χωρίς προβλήματα, έχουν μεγάλη ακρίβεια και είναι εύκολη στη χρήση. Απαιτούν μόνο δυο εξωτερικούς πυκνωτές για την εκκίνηση και διατήρηση των ταλαντώσεων.

Με λίγα λόγια, ο εξωτερικός κρύσταλλος συχνότητας συγχρονίζει τον μικροελεγκτή, δηλαδή καθορίζει την ταχύτητα που «τρέχει» ο μικροελεγκτής.

Εικόνα 4.2.1. Εξωτερικός ταλαντωτής κρυστάλλου.

4.3 Τί είναι ένα ποτενσιόμετρο;

Ένα **ποτενσιόμετρο** είναι ένας μεταβλητός αντιστάτης που χρησιμοποιείται συνήθως ως διαιρέτης τάσης, για τη μεταβολή δηλαδή του ηλεκτρικού δυναμικού (τάση). Μπορεί να λειτουργήσει και ως ρεοστάτης αν χρησιμοποιηθούν μόνο οι δύο ακροδέκτες, ο μεσαίος και ένας από τους δύο άλλους ακροδέκτες του.

Με λίγα λόγια, ένα ποτενσιόμετρο είναι μια μεταβλητή αντίσταση. Την αυξάνουμε αν θέλουμε μικρότερη τάση, την μειώνουμε αν θέλουμε μεγαλύτερη τάση. Χρησιμοποιείται συνήθως για να ρυθμίζουμε την φωτεινότητα σε οθόνες **LCD (Liquid Crystal Display/Οθόνη Υγρών Κρυστάλλων)**.

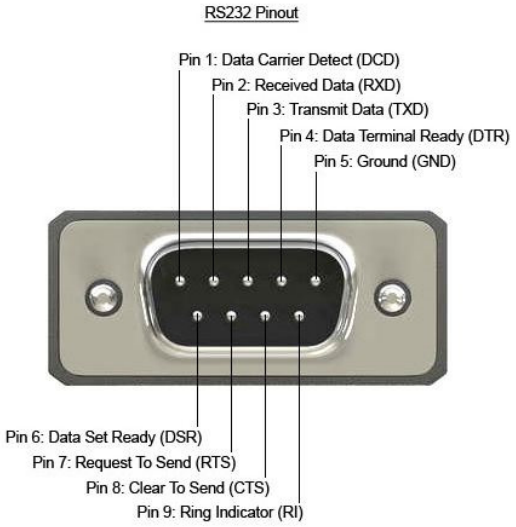
Εικόνα 4.3.1. Ένα ποτενσιόμετρο.

4.4 Τί είναι μία θύρα RS-232;

Μία θύρα **RS-232** είναι θύρα σειριακής επικοινωνίας μεταξύ δύο συσκευών. Είναι από τα πλέον στάνταρ συστατικά ενός μικροελεγκτή και χρησιμοποιείται για την επικοινωνία του ελεγκτή με διάφορα εξωτερικά κυκλώματα.

Η λειτουργία της θύρας βασίζεται στην τεχνική μορφή της σειριακής μετάδοσης των δεδομένων, σε διάφορες ταχύτητες, δηλαδή λαμβάνει δεδομένα από την είσοδο της τα όποια «ολισθαίνει» προς την έξοδο κατά ένα δυαδικό ψηφίο (bit) την φορά, σχηματίζοντας με οχτώ (8) τέτοια δυαδικά ψηφία (bits), μια λέξη που αντιστοιχεί σε ένα (byte).

Οι σειριακές θύρες υπάρχουν σε δυο τύπους: την **ασύγχρονη** σειριακή μετάδοση και την **σύγχρονη** σειριακή μετάδοση. Η λειτουργία μιας σύγχρονης σειριακής θύρας χρειάζεται και ένα παλμό σήματος συγχρονισμού (clock), ενώ αντίθετα σε μια ασύγχρονη σειριακή θύρα δεν απαιτείται η ύπαρξη τέτοιου παλμού συγχρονισμού, διότι οι πληροφορίες χρονισμού και συγχρονισμού είναι ενσωματωμένες μέσα στο σύνολο των δεδομένων που μεταδίδονται σειριακά. Επίσης, μπορεί να γίνει και χρήση πρόσθετων δυαδικών ψηφίων (bits), τα οποία σηματοδοτούν την έναρξη ή παύση, μιας συγκεκριμένης μετάδοσης δεδομένων (start bit και stop bit), αντίστοιχα.



Εικόνα 4.4.1. Μία αρσενική θύρα σειριακής επικοινωνίας, με την περιγραφή των ακροδεκτών της.

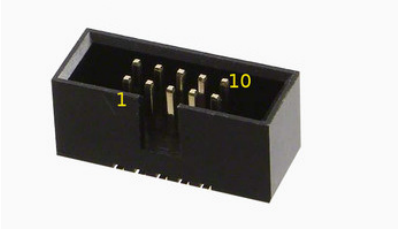
4.5 Τί είναι μία θύρα ISP/SPI;

Μια άλλη μορφή επικοινωνίας μεταξύ των μικροελεγκτών και των διαφόρων περιφερειακών διατάξεων είναι **SPI (Serial Peripheral Interface/Μονάδα Σειριακής Επικοινωνίας)** ή αλλιώς **ISP (In-System Programming)**, η οποία είναι ενσωματωμένη, στην πλειοψηφία των σύγχρονων μικροελεγκτών και φυσικά σε όλους τους μικροελεγκτές AVR Atmel.

Όπως καταλαβαίνουμε η θύρα SPI χρησιμοποιείται κυρίως για την επικοινωνία των μικροελεγκτών με διάφορες περιφερειακές διατάξεις που υποστηρίζουν αυτό το πρωτόκολλο επικοινωνίας, π.χ.: μία SD CARD, μία μονάδα Μνήμης, που έχουν ενσωματωμένη την θύρα σειριακής επικοινωνίας **SPI** και μπορούν εύκολα και γρήγορα να συνδεθούν με ένα μικροελεγκτή, ή άλλη διάταξη που έχει ενσωματωμένη θύρα **SPI**.

Το πρότυπο SPI λειτουργεί ως σύγχρονη μορφή επικοινωνίας και επιτρέπει υψηλή ταχύτητα μεταφοράς δεδομένων μεταξύ Μικροελεγκτών και περιφερειακών συσκευών ή μεταξύ πολλών μικροελεγκτών.

Μια διάταξη θύρας SPI είναι ένας «συγχρονισμένος» δίαυλος δεδομένων που σημαίνει ότι χρησιμοποιεί άλλες γραμμές για τα δεδομένα και άλλη για το **ρολόι συγχρονισμού (SCK)**. Με αυτόν τον τρόπο διατηρούνται και η δυο πλευρές, σε απόλυτο συγχρονισμό.



Εικόνα 4.5.1. Μία θύρα σειριακής επικοινωνίας SPI/ISP, με αρίθμηση των ακροδεκτών της.

4.6 Τί είναι μία «Πόρτα» του μικροελεγκτή;

Μία **Πόρτα (Port)**, είναι μια σειρά από **ακροδέκτες (PINS)** του μικροελεγκτή. Για ευκολία, κάθε μικροελεγκτής Atmel, χωρίζει τους ακροδέκτες του σε ομάδες π.χ.: Α, Β, Γ, κλπ. Μόνο που αντί για ομάδες τις ονομάζει πόρτες: **Port A**, **Port B** κλπ.

Ένας ακροδέκτης της πόρτας Α, ονομάζεται **PAx**, όπου **x** ο αριθμός του ακροδέκτη.
Αν η **Πόρτα Α** έχει έξι (6) ακροδέκτες, τότε οι ακροδέκτες παίρνουν την ονομασία κατά σειρά ως εξής: **PA0, PA1, PA2, PA3, PA4, PA5**. Παρατηρείστε ότι ξεκινάμε την σειρά από το μηδέν (0) και όχι από το ένα (1), όπως θα περίμενε κανείς.

Εικόνα 4.6.1 Ακροδέκτες που συνδέονται στην πόρτα C ενός μικροελεγκτή ATmega8

4.7 Τί είναι μία μονάδα Wi-Fi;

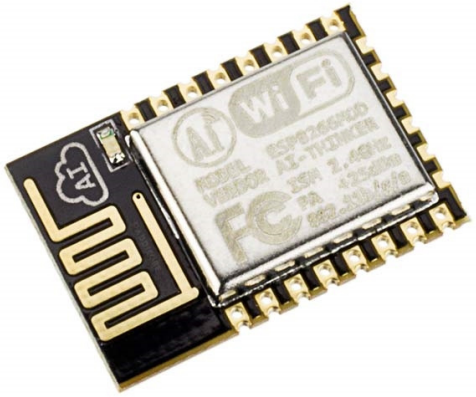
Μία **μονάδα Wi-Fi (Wi-Fi Module)**, είναι μία μονάδα που επιτρέπει την ασύρματη λήψη και μετάδοση δεδομένων μέσω Wi-Fi. Τα πρωτόκολλα TCP/UDP είναι ενσωματωμένα στην μονάδα.

Η μονάδα έχει ξεχωριστούς ακροδέκτες για την **λήψη (Receive)** και **μετάδοση (Transmit)**, δεδομένων. Συνήθως, αυτές οι μονάδες παρέχουν ένα «τερματικό» (terminal) για την ρύθμιση και χρήση της μονάδας. Η επικοινωνία με το τερματικό είναι σειριακή.

Οι μονάδες Wi-Fi μπορούν να συνδεθούν ασύρματα σε ένα υπάρχον δίκτυο ως πελάτης **(station)** ή να δράσουν σαν ένα αυτόνομο δίκτυο **(Access Point)** από μόνες τους. Η ταυτόχρονη λειτουργία ως αυτόνομου δικτύου και πελάτη είναι επίσης εφικτή.

Η πιο διάσημη μονάδα Wi-Fi είναι η **ESP Wi-Fi Module**, της κινέζικης εταιρείας **Espressif**. Από την ίδια εταιρεία υπάρχουν διάφορα μοντέλα π.χ.: **ESP12**, **ESP32, ESP8266** κα. που βασίζονται στην ίδια λογική αλλά έχουν διαφορετικά χαρακτηριστικά όπως μέγεθος μνήμης FLASH, αριθμό ακροδεκτών κλπ.

<u>Στα πλαίσια αυτής της διπλωματικής εργασίας, θα γίνει η χρήση της μονάδας ESP8266.</u>



Εικόνα 4.7.1. Μία μονάδα Wi-Fi ESP8266.

5. Περιγραφή της πειραματικής πλακέτας από την G&K Electronics

Η πειραματική πλακέτα που θα χρησιμοποιήσουμε είναι σχεδιασμένη και συναρμολογημένη από την **G&K Electronics,** και είναι χρησιμοποιείται για σύνδεση αισθητήριων οργάνων και προβολή δεδομένων μέσω LED οθονών (7 Segment Display). Η πλακέτα διαθέτει επίσης και Ρελέ (Relay) για δοκιμές με εξωτερικές ηλεκτρικές συσκευές.

Η πλακέτα χρησιμοποιεί τον μικροελεγκτή **ATmega32L**, της **Atmel**.

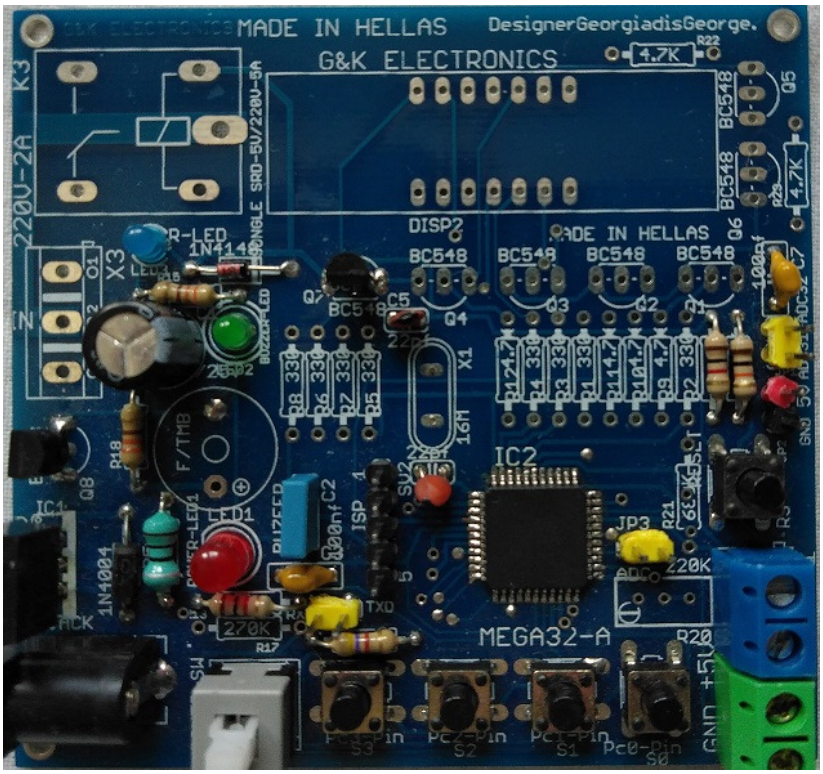
Φυσικά, μπορεί να υποστηρίξει οποιονδήποτε άλλο μικροελεγκτή της σειράς **ATmega** με 44 Pin (ATmega16/164P/324P/644P κλπ.).

Παρακάτω παρατίθεται μία λίστα με τα χαρακτηριστικά της πλακέτας:

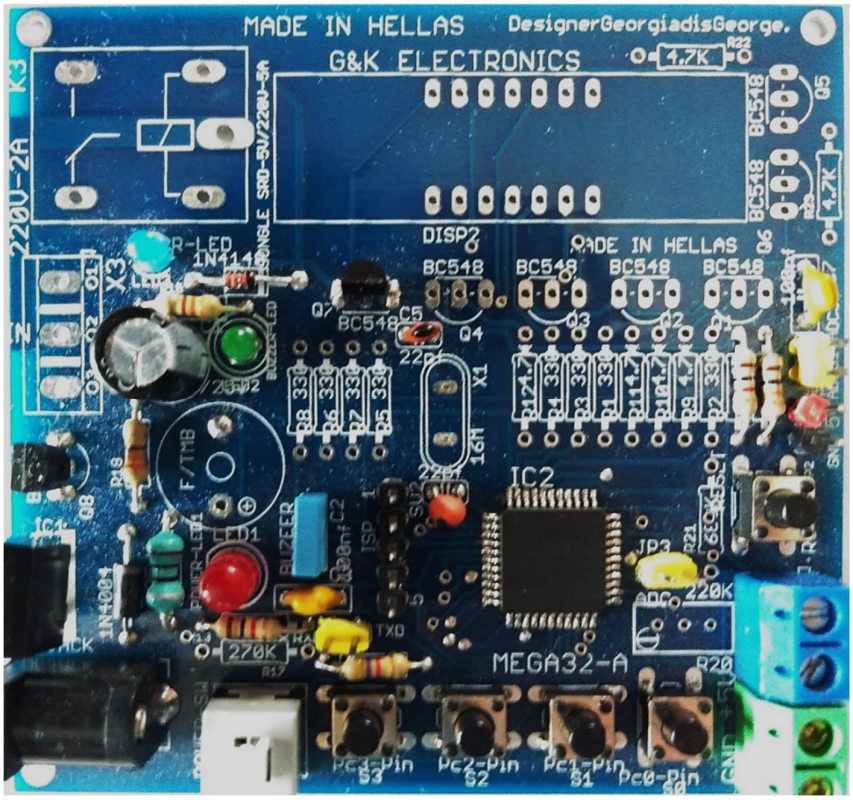
1. Είσοδος τάσης : 6 - 12V DC / 6 - 9V AC
2. Παροχή εξόδου (output) τάσης 3.3V ή 5V, για περιφερειακή χρήση σε άλλες μονάδες.
3. Ενδεικτικό LED λειτουργίας.
4. Διακόπτης ισχύος On/Off.
5. Πλήκτρο επαναφοράς της μονάδας (Reset Button).
6. Τέσσερα (4) πλήκτρα (Button) για ελεύθερη χρήση.
7. Βάση για εξωτερικό κρύσταλλο συχνότητας.
8. Ποτενσιόμετρο για ρύθμιση της τάσης εξόδου.
9. Υποδοχή ακροδεκτών (PIN) για τις λειτουργείες RXD (Receive) και TXD (Transmit), για δυνατότητα σειριακής επικοινωνίας χωρίς την χρήση της θύρας RS232.
10. Πιεζοηλεκτρικός βομβητής (Buzzer) για ελεύθερη χρήση.
11. Ρελέ (Relay) SRD - 05VDC για ελεύθερη χρήση.
12. Υποδοχή (ISP) για τον προγραμματισμό του μικροελεγκτή.
13. Υποδοχή για αισθητήρα θερμοκρασίας NTC.
14. Υποδοχή τεσσάρων (4) ακροδεκτών ADC του μικροελεγκτή για λοιπά αισθητήρια όργανα.
15. Έξι (6) μικρές οθόνες LED (7 Segment Display).

Επίσης, λόγω, της ξεχωριστής υποδοχής **TXD** και **RXD**, είναι εφικτή και η σύνδεση με περιφερειακό **Wi-Fi** Module. Παρακάτω παρατίθεται φωτογραφία της πλακέτας.

Στα πλαίσια της διπλωματικής εργασίας αυτής, έχουμε αποσυνδέσει υλικά τα οποία δεν θα χρησιμοποιήσουμε ώστε να υπάρχει όσο λιγότερος θόρυβος γίνεται.



Εικόνα 5.1. Πειραματική πλακέτα G&K Electronics.



Εικόνα 5.2. Πειραματική πλακέτα G&K Electronics.

5.1 Γιατί όχι Arduino;

Δεν επιλέξαμε Arduino για συγκεκριμένους λόγους.

Πρώτον, η υλοποίηση του συστήματος μας δεν κάνει χρήση εξωτερικών βιβλιοθηκών. Το Arduino παρέχει έτοιμες βιβλιοθήκες όχι μόνο για τον ίδιο τον μικροελεγκτή αλλά και για εξωτερικές συσκευές που μπορεί να συνδέονται με αυτό, όπως ένα ESP Wi-Fi Module. Δεν αποθαρρύνουμε την χρήση έτοιμων βιβλιοθηκών και πλατφόρμων, αλλά υπάρχουν ήδη αμέτρητα παραδείγματα στο Διαδίκτυο που κάνουν χρήση αυτών, για μικρές Web εφαρμογές. Αντιθέτως όμως, η μη χρήση υπαρχόντων βιβλιοθηκών κάνει την εφαρμογή μας ***μοναδική*** γιατί δεν υπάρχει σχεδόν καμία παρόμοια εφαρμογή στο Διαδίκτυο που να μην χρησιμοποιεί εξωτερικές βιβλιοθήκες και γενικά έτοιμο κώδικα.

Δεύτερον, ένας από τους πολλούς στόχους της διπλωματικής εργασίας αυτής, είναι και η ***καινοτομία***. Ο έτοιμος κώδικας και βιβλιοθήκες που είναι σχεδιασμένες για συγκεκριμένα προϊόντα όπως το Arduino, δεν μας αφήνουν πολλά περιθώρια για καινοτομίες.

Τρίτον, το σύστημα μας πρέπει να είναι ***ανεξάρτητο***. Ο μόνος περιορισμός που πρέπει να έχουμε είναι το είδος των μικροελεγκτών που χρησιμοποιούμε. Το σύστημα μας προορίζεται για μικροελεγκτές Atmel/Microchip αλλά υπάρχουν πολλές έτοιμες πλακέτες και έτοιμα κιτ όπως το Arduino, Adafruit κ.α. που παρόλο που χρησιμοποιούν μικροελεγκτές Atmel, έχουν διαφορετικές βιβλιοθήκες και πλατφόρμες για την ανάπτυξη εφαρμογών. Το σύστημα μας όμως, πρέπει να μην κάνει διακρίσεις και να είναι «ευέλικτο» ως προς τις διάφορες πλακέτες και πλατφόρμες, γι' αυτό και επιλέχθηκε να γίνει σε πειραματική πλακέτα που δεν έχει κάποια συγκεκριμένη χρήση. ***Ασχολούμαστε αποκλειστικά με το πρωτογενές επίπεδο, δηλαδή με τον μικροελεγκτή, ανεξαρτήτως της πλακέτας όπου βρίσκεται τοποθετημένος.***

Και τελευταίος, αλλά εξίσου σημαντικός, λόγος, είναι η ***γνώση***. Χωρίς εξωτερικές βιβλιοθήκες και έτοιμο κώδικα μπορούμε να σχεδιάσουμε το δικό μας σύστημα ανακαλύπτοντας νέες έννοιες όχι μόνο για τους μικροελεγκτές και τους αυτοματισμούς, αλλά και για τον ίδιο τον προγραμματισμό. Βρίσκοντας και διορθώνοντας τα λάθη (debugging), μόνοι μας, αποκτούμε περισσότερη γνώση και εμπειρία. Δεν είναι λάθος να ασχοληθεί κάποιος με έτοιμο υλικό, αλλά είναι απαραίτητο εφόσον χρησιμοποιεί αυτό το υλικό, να γνωρίζει πως και γιατί λειτουργεί.

Είναι προφανές ότι τα έτοιμα κιτ όπως το Arduino, δίνουν περισσότερο έμφαση στον προγραμματισμό του μικροελεγκτή και λιγότερο στο ίδιο τον μικροελεγκτή και τα περιφερειακά του. Από την μία πλευρά, αυτό είναι καλό γιατί ο προγραμματιστής μπορεί να αφιερώνει τον χρόνο του στο πρόγραμμα και να μην ασχολείται με τα υλικά/ηλεκτρικά θέματα της πλακέτας και του μικροελεγκτή.

Από την άλλη πλευρά όμως, όσον αφορά τους αυτοματισμούς και τα ενσωματωμένα συστήματα, ο προγραμματιστής θα πρέπει να γνωρίζει τουλάχιστον τις βασικές αρχές που διέπουν έναν μικροελεγκτή και τα περιφερειακά του.

Όχι μόνον γιατί γενικά είναι καλό να γνωρίζει, αλλά επειδή μπορεί να βελτιστοποιήσει και να χρησιμοποιήσει, άγνωστους γι' αυτόν μέχρι τότε, πόρους του μικροελεγκτή, για να σχεδιάσει και προγραμματίσει το σύστημα του όσο καλύτερα και αποτελεσματικά γίνεται. Γι' αυτό λοιπόν, καλό είναι κάποιος αφού ασχοληθεί με ένα έτοιμο κιτ, να προσπαθήσει να καταλάβει και να αναζητήσει πως και γιατί δουλεύουν, αυτά που δουλεύουν.

6. Προετοιμασία πλακέτας

Πριν ξεκινήσουμε να εξηγούμε τί θα κάνουμε, ας αναφέρουμε πρώτα τα απαραίτητα υλικά που θα χρειαστούμε.

6.1 Απαραίτητα υλικά

Καταρχάς, χρειαζόμαστε μία πλακέτα με μικροελεγκτή **ATmega**, που να διαθέτει τουλάχιστον 32KB μνήμη FLASH και 2KB μνήμη SRAM όπως οι: ATmega32, ATmega328, ATmega644 κλπ. Για την εφαρμογή μας καλό θα ήταν να χρησιμοποιήσουμε μία αναπτυξιακή/πειραματική πλακέτα που να μην έχει συγκεκριμένο σκοπό και να είναι πιο πολύ για εκπαιδευτικούς σκοπούς

Επίσης, θα χρειαστούμε και ένα **Wi-Fi Module** για να μπορέσουμε να έχουμε ασύρματη επικοινωνία. Υπάρχουν πολλών ειδών Wi-Fi Module, αλλά εμείς επιλέξαμε να χρησιμοποιήσουμε το **ESP8266 Wi-Fi Module** της **Espressif**, όπως αναφέραμε προηγουμένως. Η αγορά έγινε ηλεκτρονικά και η παραγγελία ήρθε από το εξωτερικό. Είναι διαθέσιμοι και στην Ελλάδα (π.χ. μέσω του www.skroutz.gr). Το κόστος του module κυμαίνεται στα **4 έως 6 € (ευρώ).**

Το επόμενο που θα χρειαστούμε είναι ένας αισθητήρας. Στα πλαίσια της διπλωματικής εργασίας αυτής, επιλέχθηκε αισθητήρας θερμοκρασίας. Συγκεκριμένα ένας **NTC Thermistor**. Ο χρήστης μπορεί να βάλει οποιοδήποτε είδος αισθητήρα επιθυμεί.

Ένα **θερμίστορ** είναι ένα είδος αντίστασης, η τιμή της οποίας επηρεάζεται από τη θερμοκρασία, πολύ περισσότερο απ' όσο στις συνηθισμένες αντιστάσεις (ωμικές αντιστάσεις). Η λέξη θερμίστορ είναι συνένωση των Αγγλικών λέξεων **thermal** και **resistor**.

Τα θερμίστορ χρησιμοποιούνται ευρέως για τον περιορισμό της απότομης αύξησης των ηλεκτρικών τάσεων (ρευμάτων), σαν αισθητήρες θερμοκρασίας (**Αρνητικού Συντελεστή Θερμοκρασίας - Negative Temperature Coefficient ή NTC**), σαν αυτο-επαναφερόμενες ασφάλειες και σαν αυτο-ρυθμιζόμενα στοιχεία θέρμανσης (**Θετικού Συντελεστή Θερμοκρασίας - Positive Temperature Coefficient ή PTC**).

Τα θερμίστορ χωρίζονται σε δύο (2) κατηγορίες:

1. Τα **NTC**, στα οποία η αντίσταση μειώνεται καθώς αυξάνεται η θερμοκρασία. Συνήθως συνδέονται «παράλληλα» στα κυκλώματα, όποτε μέσω αυτών διακλαδίζεται ένα μέρος του ρεύματος.
2. Τα **PTC**, στα οποία η αντίσταση αυξάνεται καθώς αυξάνεται η θερμοκρασία, Συνήθως συνδέονται σε σειρά στα κυκλώματα, σαν αυτο-επαναφερόμενες ασφάλειες.

Η τιμή τους κυμαίνεται μεταξύ **1 έως 2 € (ευρώ).** Συνήθως, πωλούνται σε κομμάτια των πέντε (5) και δέκα (10) τεμαχίων, με τιμές ~**5 και** ~**10 € (ευρώ)** αντίστοιχα. Είναι διαθέσιμοι στο εξωτερικό αλλά και στην Ελλάδα.

Το θερμίστορ που θα χρησιμοποιήσουμε είναι ειδικό για ψυγεία και καταψύκτες γιατί ενώνεται με καλώδιο, ώστε να μην χρειάζεται η πλακέτα να βρίσκεται μέσα στο ψυγείο, για να γίνει η μέτρηση. Δεν θα το χρησιμοποιήσουμε σε ψυγείο, απλώς το καλώδιο μας δίνει περισσότερη ευελιξία.

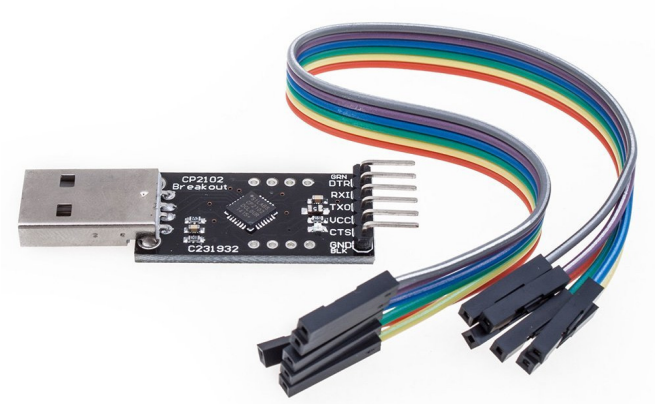
Παρατίθεται φωτογραφία ενός NTC Thermistor με καλώδιο.

Εικόνα 6.1.1 NTC Thermistor

Το επόμενο στην λίστα που θα χρειαστούμε είναι ένα module **USB to TTL Converter**. Δηλαδή έναν μετατροπέα σημάτων USB σε σειριακά. Είναι διαθέσιμα στο εξωτερικό και στην Ελλάδα.

Η τιμή τους κυμαίνεται από **2 έως 5 € (ευρώ)**.

Θα πρέπει να δοθεί προσοχή, πριν την αγορά, να γίνει έλεγχος αν το προϊόν διαθέτει μαζί και καλώδιο για την σύνδεση του με τον υπολογιστή. Επίσης, ένα άλλο εξίσου σημαντικό, είναι ο μετατροπέας να έχει **τάση εξόδου 3.3V**. Είναι απαραίτητο γιατί το ESP8266 απαιτεί τάση εισόδου 3.3V. Κάποιοι μετατροπείς δεν χρησιμοποιούν πλακέτα αλλά μόνο ένα καλώδιο. Απλώς τα εξαρτήματα για την μετατροπή βρίσκονται πάνω στο καλώδιο, στην μία άκρη του.



6.1.2. Μετατροπέας USB to TTL με καλώδια ακροδεκτών.

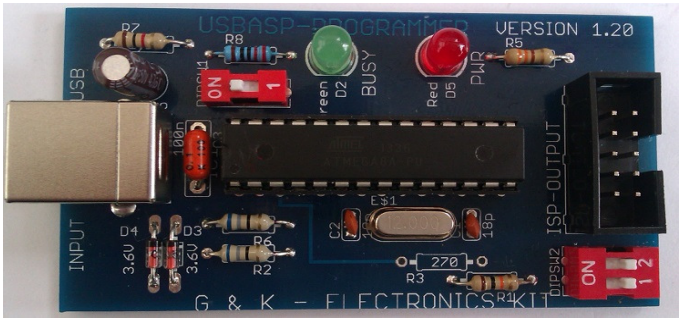
Το επόμενο που χρειαζόμαστε είναι έναν φορτιστή για να τροφοδοτούμε ρεύμα στην πλακέτα μας. Η τιμή τους κυμαίνεται από **5 έως 10 € (ευρώ)**. Είναι διαθέσιμοι στο εξωτερικό αλλά και στην Ελλάδα.

6.1.3. Υποδειγματικός φορτιστής.

Τέλος, χρειαζόμαστε και έναν **Programmer** για μικροελεγκτές Atmel, με τον οποίο θα φορτώσουμε το πρόγραμμα στην πλακέτα μας. Η μία άκρη του **Programmer** συνδέεται στον υπολογιστή μέσω USB και η άλλη μέσω **ISP** στην πλακέτα μας. Ύστερα, μέσω ειδικού λογισμικού, δίνουμε εντολή στον **Programmer** να φορτώσει το πρόγραμμα που θέλουμε, στον μικροελεγκτή της πλακέτας μας.

Μία αναζήτηση για «Atmel Programmer» στο Διαδίκτυο θα εμφανίσει πολλά αποτελέσματα.

Η τιμή τους κυμαίνεται από **5 έως 15 € (ευρώ)**. Είναι διαθέσιμοι στο εξωτερικό αλλά και στην Ελλάδα. Για ευκολία, επιλέξαμε να χρησιμοποιήσουμε **Programmer** από τον κατασκευαστή της πλακέτας μας, **G&K Electronics**.



6.1.4. Programmer της G&K Electronics.

Φυσικά, θα χρειαστούμε και μερικά καλώδια για τις συνδέσεις ακροδεκτών. Επειδή όπως είπαμε χρησιμοποιούμε πειραματική/αναπτυξιακή πλακέτα, μας προσφέρονται καλώδια ακροδεκτών μαζί με την ίδια την πλακέτα. Ο χρήστης θα πρέπει πριν την αγορά να σιγουρευτεί ότι η πλακέτα συνοδεύεται από καλώδια ακροδεκτών ή αλλιώς να τα αγοράσει ξεχωριστά.

Αυτά λοιπόν, όσον αφορά το υλικό κομμάτι της εργασίας μας.

Θα χρειαστούμε όμως και κάποια προγράμματα στον υπολογιστή **(software)**, εκτός από την πλατφόρμα **Atmel Studio**.

Για τον προγραμματισμό της πλακέτας θα χρειαστούμε ένα ειδικό λογισμικό. Υπάρχουν πολλά λογισμικά για προγραμματισμό πλακέτας με μικροελεγκτή **Atmel**, τα περισσότερα απ' αυτά είναι δωρεάν. Μερικά απ' αυτά είναι:

1. **eXtreme Burner - AVR**
2. **AVRDUDE**
3. **Khazama AVR Programmer**

Όλα είναι εξίσου καλά, οι μόνες διαφορές είναι στο γραφικό περιβάλλον. Παρ' όλα αυτά επιλέξαμε το **eXtreme Burner – AVR** με την σκέψη ότι είναι το πιο απλό στην χρήση.

Το πρόγραμμα αυτό είναι δωρεάν λογισμικό και μπορούμε να το κατεβάσουμε από την ιστοσελίδα:

http://extremeelectronics.co.in/avr-tutorials/gui-software-for-usbasp-based-usb-avr-programmers/

Επόμενο στην λίστα είναι ένα ύπερ-τερματικό (hyper-terminal), για την επικοινωνία του **υπολογιστή** με το **Wi-Fi Module**, μέσω του μετατροπέα USB σε TTL.

Υπάρχουν πολλά δωρεάν υπερ-τερματικά στο διαδίκτυο, μερικά από αυτά είναι:

1. **Putty**
2. **TeraTerm**

Το πιο διαδεδομένο από αυτά είναι το **Putty**. Οδηγίες εγκατάστασης και χρήσης του Putty, θα αναφερθούν σε επόμενο κεφάλαιο.

Τέλος, θα χρειαστούμε **drivers** για την επικοινωνία υπολογιστή με τον **Programmer**. Συγκεκριμένα χρειαζόμαστε τους **libusb** drivers.

Πρόκειται για δωρεάν λογισμικό και υπάρχουν παντού στο Διαδίκτυο. Παραθέτουμε έναν από τους πολλούς συνδέσμους που υπάρχουν για την λήψη του:

https://github.com/libusb/libusb/releases/download/v1.0.22/libusb-1.0.22.7z

6.2 Ανακεφαλαίωση - Λίστα απαραιτήτων

Αφού τελειώσαμε και με το υλικό κομμάτι και με το κομμάτι του υπολογιστή, ας κάνουμε μία σύντομη λίστα με όσα προαναφέρθηκαν.

1. Υλικό **(Hardware)**
    a. Πλακέτα με μικροελεγκτή ATmega (32KB Flash, 2KB SRAM).
    b. Wi-Fi Module ESP8266.
    c. NTC Thermistor ή οποιοσδήποτε άλλος αισθητήρας.
    d. Μετατροπέας USB σε TTL.
    e. Programmer.
2. Λογισμικό **(Software)**
    a. Atmel Studio.
    b. eXtreme Burner – AVR.
    c. Ύπερ-τερματικό Putty (ή όποιο άλλο πρόγραμμα της αρεσκείας μας).

6.3 Σύνδεση και ρύθμιση περιφερειακών συσκευών

6.3.1 Σύνδεση φορτιστή

Πρώτα, θα συνδέσουμε τον φορτιστή με την πλακέτα.

Η συγκεκριμένη πλακέτα που χρησιμοποιούμε τροφοδοτείται με τάση **3.3V**. Ορισμένες πλακέτες διαθέτουν **διακόπτη/jumper** για την αλλαγή από 5V σε 3.3V και το αντίστροφο. Ο χρήστης θα πρέπει να πράξει ανάλογα με την πλακέτα που διαθέτει. Όπως και να έχει, ο μικροελεγκτής θα πρέπει να τροφοδοτείται με 3.3V.

Πριν συνδέσουμε τον φορτιστή, ελέγχουμε αν ο διακόπτης εκκίνησης είναι πατημένος ή όχι. Θα πρέπει να μην είναι πατημένος, γιατί θέλουμε πρώτα να συνδέσουμε όλα τα εξαρτήματα, πριν τροφοδοτήσουμε με τάση την πλακέτα.

6.3.2 Σύνδεση θερμίστορ

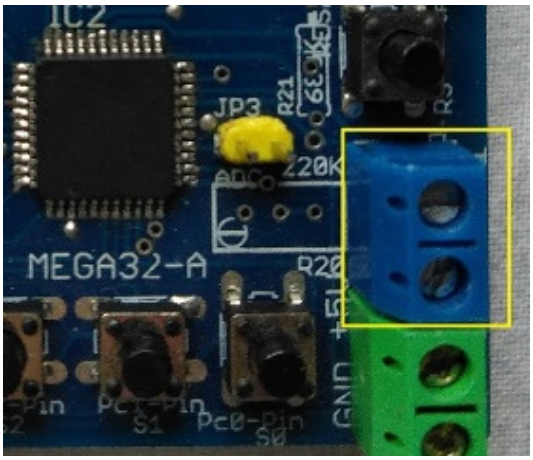
Το θερμίστορ έχει δύο (2) υποδοχές.

Μία είναι για την τάση εισόδου (VCC) και η άλλη για την έξοδο της πληροφορίας (ADC) της θερμοκρασίας που θέλουμε.

Η υποδοχή για την τάση εισόδου μπορεί να συνδεθεί σε μία από τις υποδοχές ρεύματος (**VCC**) που διαθέτει η πλακέτα. Σχεδόν όλες η πειραματικές/αναπτυξιακές πλακέτες διαθέτουν ακροδέκτες για εξωτερική υποδοχή ρεύματος (VCC & GND).

Την υποδοχή για την πληροφορία της θερμοκρασίας πρέπει να την συνδέσουμε σε έναν ακροδέκτη **ADC (Analog - Digital Converter**, **Μετατροπέας Αναλογικού σε Ψηφιακό**). Οι μικροελεγκτές Atmel διαθέτουν πολλούς τέτοιους ακροδέκτες για την χρήση αισθητήριων οργάνων. Συνήθως, αναγράφονται πάνω στις πλακέτες οι ονομασίες ADC0, ADC1 κλπ., δίπλα στους αντίστοιχους ακροδέκτες. Αλλιώς, θα πρέπει ο χρήστης να αναζητήσει στο σχεδιάγραμμα του μικροελεγκτή, σε ποιες πόρτες υπάρχουν ακροδέκτες ADC.

Στους περισσότερους μικροελεγκτές Atmel, η **Πόρτα C** περιέχει ακροδέκτες ADC, οπότε καλό θα ήταν ο χρήστης να τους αναζητήσει εκεί πρώτα. Στην πλακέτα που χρησιμοποιούμε εμείς, έχουμε ειδική θέση για τον NTC θερμίστορ.



Εικόνα 6.3.2.1. Υποδοχή ρεύματος (VCC & ADC) για τον NTC θερμίστορ.

6.3.3 Σύνδεση Wi-Fi Module με υπολογιστή μέσω μετατροπέα USB σε TTL

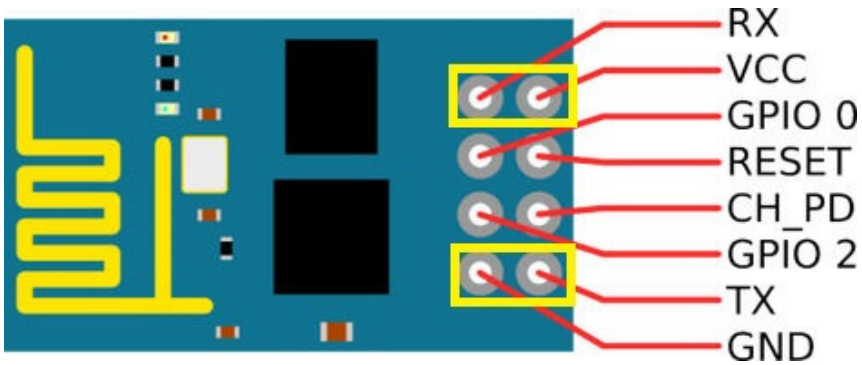
Το επόμενο βήμα είναι να συνδέσουμε το **Wi-Fi Module** με τον υπολογιστή για να το ρυθμίσουμε. Γι' αυτό, θα χρειαστούμε τον μετατροπέα **USB σε TTL** που αναφέραμε πιο πάνω.

Τα καλώδια που πρέπει να συνδέσουμε είναι τέσσερα (4):

1. **VCC** (Τάση)
2. **GND** (Γείωση)
3. **RX** (Receive - Λήψη)
4. **TX** (Transmit - Μετάδοση)

Πρώτα, εντοπίζουμε τους αντίστοιχους ακροδέκτες στο Wi-Fi Module.

Παρακάτω παρατίθεται σχέδιο με τις ονομασίες των ακροδεκτών ενός **ESP8266 Wi-Fi Module** και σημειωμένοι σε κίτρινα πλαίσια είναι οι ακροδέκτες που χρειαζόμαστε**.**



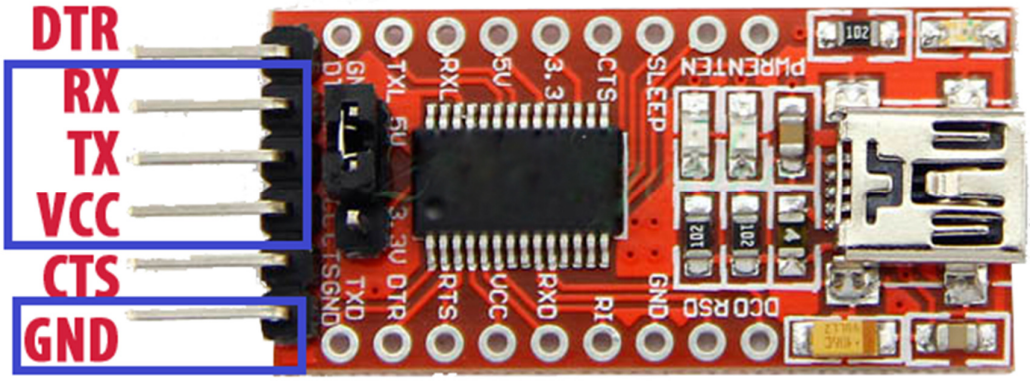
Εικόνα 6.3.3.1. Σχέδιο με τις ονομασίες των ακροδεκτών ενός ESP8266 Wi-Fi Module.

Σημειώνεται ότι το παραπάνω σχέδιο αντιπροσωπεύει μία από τις εκδοχές ενός **ESP8266**. Υπάρχουν εκδοχές που τοποθετούν τους ακροδέκτες σε διαφορετικά σημεία ανάλογα με τον κατασκευαστή και τις ανάγκες του αγοραστή.

Σε οποιαδήποτε περίπτωση όμως, οι εκδοχές για το **ESP8266** δεν ξεπερνούν τις τρείς (3). Μια αναζήτηση στο διαδίκτυο εμφανίζει δεκάδες φωτογραφίες και σχέδια απ' όλες τις εκδοχές, με διακριτές τις ονομασίες των ακροδεκτών.

Αφού εντοπίσαμε τους απαραίτητους ακροδέκτες στο Wi-Fi Module, τώρα πρέπει να εντοπίσουμε τους αντίστοιχους στον **μετατροπέα USB σε TTL**.

Παρακάτω παρατίθεται φωτογραφία με τις ονομασίες των ακροδεκτών ενός **μετατροπέα USB σε TTL**. Σημειωμένοι σε μπλε πλαίσια είναι οι αντίστοιχοι ακροδέκτες που θα χρειαστούμε.



Εικόνα 6.3.3.2. Σχέδιο ενός μετατροπέα USB σε TTL, με τις ονομασίες των ακροδεκτών του.

Τώρα πρέπει να συνδέσουμε τους ακροδέκτες του μετατροπέα με τους ακροδέκτες του **Wi-Fi Module**.

Χρησιμοποιώντας καλώδια ακροδεκτών, κάνουμε την σύνδεση ως εξής:

1. **GND** ----> **GND**
2. **VCC** ----> **VCC**
3. **RX_1** ----> **TX_2**
4. **TX_1** ----> **RX_2**

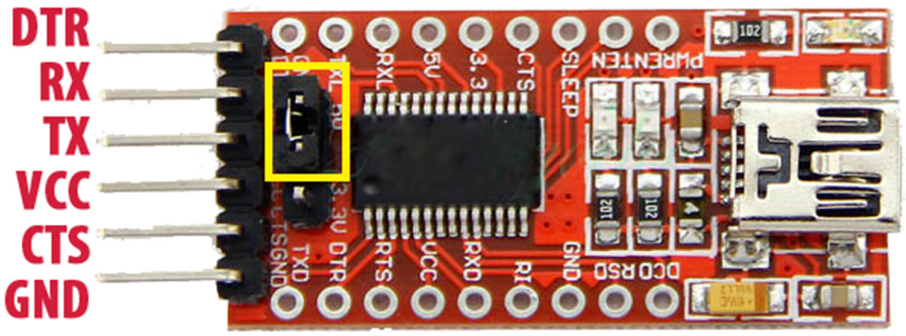
Παρατηρήστε ότι η μετάδοση του ενός (**TX_1**) συνδέεται με την λήψη του άλλου (**RX_2**) και η λήψη του ενός (**RX_1**) με την μετάδοση του άλλου (**TX_2**), όπου είναι και αυτό που θέλουμε.

Μερικές φορές μπορεί κάποιος να μπερδευτεί και να συνδέσει την λήψη με την λήψη (**RX_1 -> RX_2**) ή την μετάδοση με την μετάδοση (**TX_1 -> TX_2**) λόγω της ίδιας ονομασίας.

Το σκεπτικό είναι να συνδέονται στα αντίθετα τους (**TX_1 -> RX_2 και TX_2 -> RX_1**).

Επίσης, ο μετατροπέας πρέπει να έχει έξοδο τάσης **3.3V** ώστε να μπορεί να επικοινωνήσει με το Wi-Fi module. Συνήθως, υπάρχει μια πολύ μικρή πλαστική θήκη, το λεγόμενο **jumper**, που ανάλογα με την σύνδεση της καθορίζει και την ανάλογη τάση εξόδου (**VCC**).

Θα πρέπει να βρούμε πάνω στον μετατροπέα την θήκη αυτήν και να την αλλάξουμε θέση αν χρειάζεται, για να έχουμε τάση εξόδου **3.3V**. Εκεί που θα είναι συνδεδεμένη η θήκη, θα αναφέρονται σε ποιους ακροδέκτες είναι τα **5V** και τα **3.3V** αντίστοιχα.



Εικόνα 6.3.3.3. Θήκη (Jumper) για την ρύθμιση της τάσης εξόδου στα 3.3V, ενός μετατροπέα USB σε TTL.

Τώρα μένει να συνδέσουμε τον μετατροπέα με τον υπολογιστή. Γι’ αυτό, χρειαζόμαστε ένα **καλώδιο USB**. Συνήθως, μαζί με την συσκευασία του μετατροπέα υπάρχει και καλώδιο. Αν όχι, τότε θα πρέπει ο χρήστης να αγοράσει ένα, αν δεν έχει.

Καλώδια USB υπάρχουν σε όλα σχεδόν τα πολυκαταστήματα (π.χ. **Public**, **Γερμανός** κλπ.) και κοστίζουν το πολύ **3 έως 4 € (ευρώ).**

Μόλις συνδέσουμε το καλώδιο USB στον μετατροπέα με τον υπολογιστή, το λειτουργικό θα προσπαθήσει να εγκαταστήσει τους **οδηγούς** (drivers) από μόνο του.

Κατά 99% των περιπτώσεων, η εγκατάσταση είναι επιτυχής. Αν για κάποιο λόγο δεν γίνει η εγκατάσταση των οδηγών λόγω προβλήματος, θα πρέπει ο χρήστης να αναζητήσει στο **Διαδίκτυο** τους οδηγούς για τον συγκεκριμένο μετατροπέα που διαθέτει.

6.3.4 Ρύθμιση ESP Wi-Fi Module

Για να ρυθμίσουμε το module, χρειαζόμαστε ένα υπερ-τερματικό (hyper-terminal) όπως αναφέρθηκε παραπάνω.

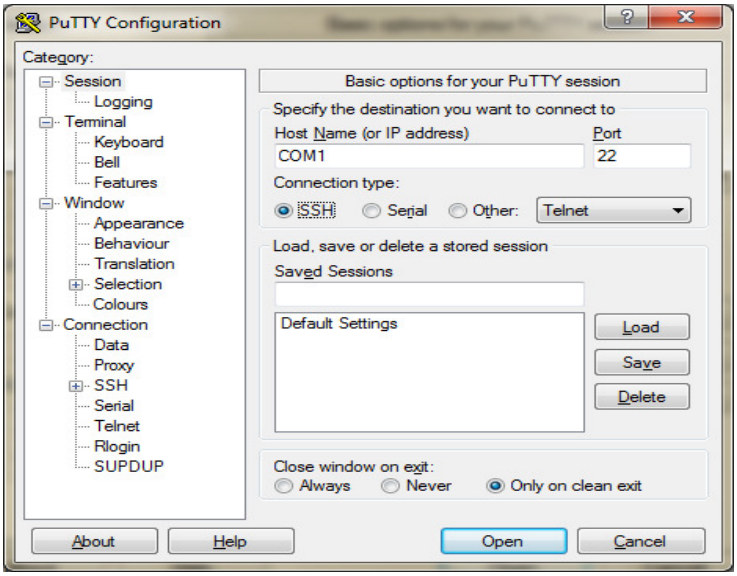
Για τυπικούς λόγους όμως, θα αναφερθούν οδηγίες για την εγκατάσταση και την χρήση ενός δωρεάν υπερ-τερματικού, του **Putty**. Υπενθυμίζουμε ότι οποιοδήποτε άλλο υπερ-τερματικό θα φέρει τα ίδια αποτελέσματα.

Ακολουθώντας τον παρακάτω σύνδεσμο θα οδηγηθούμε σε ιστοσελίδα με δυνατότητα λήψης του **Putty** για **64-bit** αλλά και για **32-bit** λειτουργικά συστήματα.

Η εγκατάσταση είναι απλή και διαρκεί μόνο λίγα λεπτά.

https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

Με την εκτέλεση του Putty, εμφανίζεται η παρακάτω οθόνη:



Εικόνα 6.3.4.1. Οθόνη ρυθμίσεων

Επιλέγουμε Σειριακό **(Serial)** τύπο σύνδεσης **(Connection Type)**. Στην συνέχεια θα πρέπει να εισάγουμε την σειριακή πόρτα (**Serial line**) και την ταχύτητα (**Speed**).

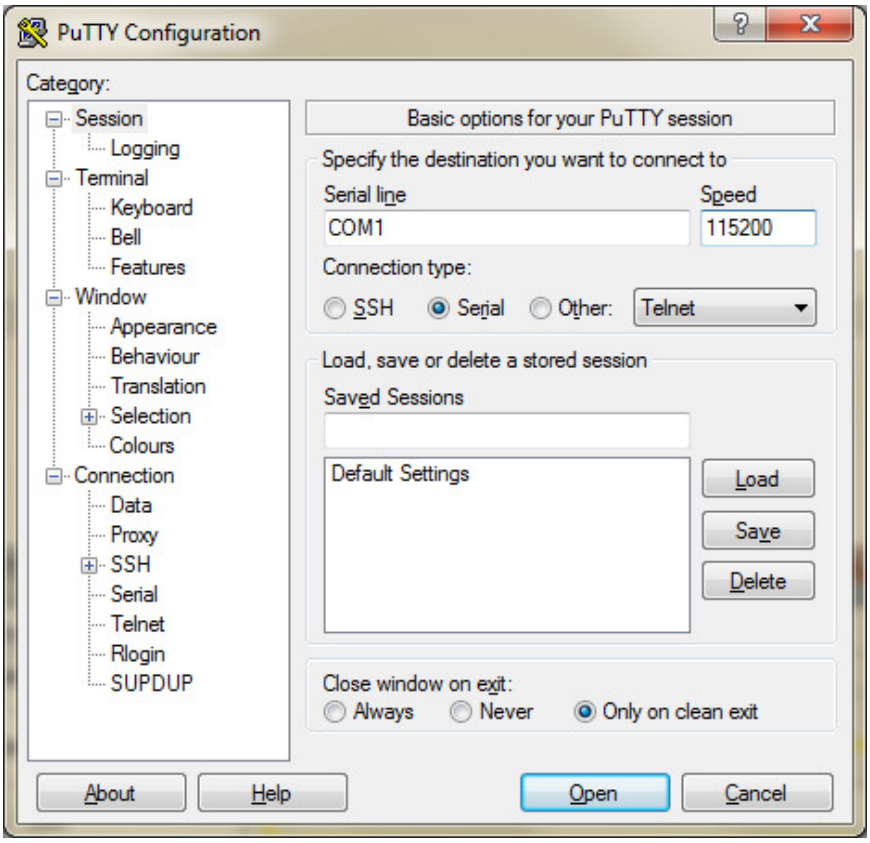
Η σειριακή πόρτα είναι η θύρα που είναι συνδεδεμένος ο μετατροπέας. Πρέπει να ξέρουμε σε ποια είναι συνδεδεμένη για κάνουμε την σύνδεση. Οι σειριακές πόρτες συμβολίζονται με την λέξη **COM** και μετά ακολουθεί ο αριθμός (π.χ. **COM1**, **COM2**).

Τέλος, η ταχύτητα είναι ο ρυθμός (**Baud Rate**) με τον οποίο θα στέλνουμε τα δεδομένα και μετριέται σε bits ανά δευτερόλεπτο. Υπάρχουν κάποιες ταχύτητες που χρησιμοποιούνται συχνά π.χ. 9600, 19200, 57600, 115200 κλπ.

Η προκαθορισμένη (default) ταχύτητα του module είναι 115200 bps.

Επειδή όμως είναι πολύ μεγάλη για τις ανάγκες μας, θα χρησιμοποιήσουμε την ταχύτητα των **9600 bps**, αλλάζοντας την αργότερα μέσω κατάλληλης εντολής. Μέχρι

να το κάνουμε αυτό όμως, θα χρησιμοποιήσουμε αυτήν που έχει προκαθορισμένη το module, δηλαδή **115200 bps**.



Εικόνα 6.3.4.2. Ρύθμιση COM πόρτας και Baud Rate

Συμπληρώνουμε τα κατάλληλα πεδία (**Serial line** και **Speed**) με τις ανάλογες τιμές.

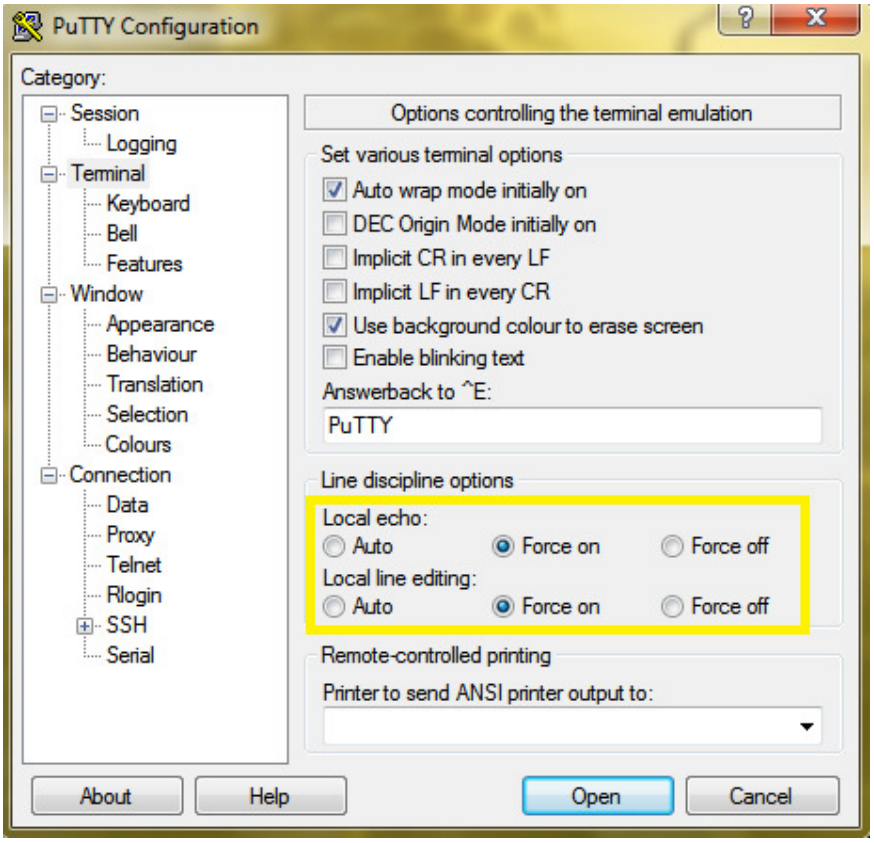
Πριν ξεκινήσουμε την σύνδεση, χρειαζόμαστε να ρυθμίσουμε μερικά πράματα ακόμα. Όπως κοιτάμε την κύρια οθόνη, πηγαίνουμε στην επιλογή **Terminal** στο δέντρο επιλογών στα αριστερά μας. Εκεί υπάρχουν διάφορες ρυθμίσεις που αφορούν το τεχνικό κομμάτι.

Η επιλογή «**Local echo**» είναι απλώς για να εμφανίζονται στο τερματικό αυτά που γράφουμε. Για να την ενεργοποιήσουμε επιλέγουμε «**Force on**».

Η τελευταία επιλογή «**Local line editing**» είναι απλώς για να μπορούμε να τροποποιούμε αυτά που θα στείλουμε, όπως σε έναν κειμενογράφο. Για την αποστολή τους, απλώς πατάμε **Enter**. Για να την ενεργοποιήσουμε επιλέγουμε «**Force on**».

Για να γυρίσουμε στην κεντρική οθόνη, πατάμε την επιλογή **Session**, στο δέντρο επιλογών στα αριστερά.

Παρατίθεται φωτογραφία με την οθόνη επιλογών. Σημειωμένες σε κίτρινα πλαίσια είναι οι επιλογές που μας ενδιαφέρουν.



Εικόνα 6.3.4.3. Ρυθμίσεις Terminal υπερ-τερματικού PuTTy.

Τώρα είμαστε έτοιμοι να κάνουμε την σύνδεση. Επιλέγουμε **Άνοιγμα** (**Open**), για να αρχίσουμε την σύνδεση. Θα εμφανιστεί ένα κενό τερματικό, με έναν κέρσορα, όπως στην φωτογραφία παρακάτω.

Από εδώ, μπορούμε να γράφουμε και να στέλνουμε εντολές στο **ESP**.

Οι εντολές που θα στέλνουμε πρέπει να είναι με κεφαλαία. Αυτό είναι απαίτηση του ESP.

Επίσης, το ESP χρειάζεται, στο τέλος των εντολών, να υπάρχουν οι χαρακτήρες **CR (Carriage Return - Επιστροφή Φορέα)** και **LF (Line Feed – Τροφοδοσία Γραμμής)**. Γι' αυτό θα πρέπει αφού πατήσουμε Enter για να στείλουμε την εντολή, να πατήσουμε τον συνδυασμό Ctrl + J όπου εισάγει τον χαρακτήρα LF και μετά να ξαναπατήσουμε Enter για να τον στείλουμε.

Άρα η σειρά των βημάτων είναι η εξής:

1. Πληκτρολογούμε την εντολή.
2. Πατάμε **Enter**. (Αποστέλλεται ο χαρακτήρας **CR** αυτόματα)

3. Πατάμε τον συνδυασμό **Ctrl + J**. (Εισαγωγή του χαρακτήρα **LF**)
4. Πατάμε **Enter**. (Αποστέλλεται ο χαρακτήρας **LF**).

Παρακάτω, παρουσιάζεται πίνακας με τις εντολές που θα χρησιμοποιήσουμε και το τί κάνει η κάθε μία καθώς και τι επιστρέφει.

1. *Όσες εντολές είναι <u>υπογραμμισμένες</u>, σημαίνει ότι δέχονται και παραμέτρους αλλά δεν γίνεται αναφορά επειδή δεν μας είναι απαραίτητο.*
2. *Εντολές που τελειώνουν με «**DEF**» αποθηκεύουν τις τιμές τους στην μνήμη FLASH. Άρα, άμα γίνει επανεκκίνηση του Module, οι τιμές που δώσαμε θα παραμείνουν.*

6.3.4.4. Αρχική κατάσταση υπερ-τερματικού PuTTy.

Πίνακας 6.3.4.1. Περιγραφή εντολών ESP που θα χρησιμοποιηθούν.

Εντολή	Επιστρέφει	Περιγραφή
AT	**OK**	Υπάρχει μόνο για να ελέγχουμε αν η σύνδεση είναι ενεργή/επιτυχής.
<u>AT+CWLAP</u>	**+CWLAP:<ecn>,<ssid>,<rssi>,<mac>, <ch>** … **OK**	Εμφανίζει λίστα με τα διαθέσιμα δίκτυα Wi-Fi μαζί με διάφορες πληροφορίες (SSID, συχνότητα, MAC, κανάλι κλπ).
<u>AT+CWJAP_DEF=<ssid>,<pwd></u>	**OK** *(Για επιτυχής σύνδεση)* ή **+CWJAP_DEF:<error code>** **ERROR** *(Για αποτυχημένη σύνδεση)*	Συνδέεται στο δίκτυο με όνομα <**ssid**> και κωδικό <**pwd**>. Το δίκτυο αποθηκεύεται στην μνήμη του ESP8266, οπότε στο μέλλον θα μπαίνει αυτόματα. Αν η σύνδεση αποτύχει εμφάνιζει κωδικό σφάλματος <**error code**>.

		Τα στοιχεία αποθηκεύονται στην μνήμη FLASH.
ATE<parameter>	**OK**	Ρυθμίζει την κατάσταση της «ηχούς». Όταν στέλνουμε εντολές στο ESP, αυτό τις επιστρέφει πίσω μαζί με την απάντηση. **<parameter> = 0 : Απενεργοποίηση** **<parameter> = 1 : Ενεργοποίηση**
AT+CIFSR	**+CIFSR:APIP, <SoftAP IP address>** **+CIFSR:APMAC, <SoftAP MAC address>** **+CIFSR:STAIP, <Station IP address>** **+CIFSR:STAMAC, <Station MAC address>** **OK**	Εμφανίζει τις IP και MAC διευθύνσεις του module. **<SoftAP IP address>** = Server IP **<Station IP address>** = Client IP **<SoftAP MAC address>** = Server MAC **<Station MAC address>** = Client MAC
AT+CIPMUX=<mode>	**OK**	Καθορίζει αν το module θα δέχεται μόνο μία ή πολλαπλές συνδέσεις. **<mode> = 0 : Μία σύνδεση** **<mode> = 1 : Πολλαπλές συνδέσεις** Σε κάθε σύνδεση που δημιουργείται αναθέτεται ένα **μοναδικό αναγνωριστικό (link ID).**
AT+CIPSERVER=<mode> [,<port>]	**OK**	Δίνει εντολή στο module να ξεκινήσει/σταματήσει έναν **TCP Server**, στην συγκεκριμένη θύρα **<port>**. **<mode> = 0 : Διακοπή server** **<mode> = 1 : Έναρξη server** Αν δεν δοθεί παράμετρος για την θύρα, επιλέγεται η **θύρα 333**, αυτόματα. Για να γίνει έναρξη ενός server, πρέπει το module να έχει ρυθμιστεί για πολλαπλές συνδέσεις από πρίν. **(AT+CIPMUX=1)**
AT+CWMODE_DEF=<mode>	**OK**	Ρυθμίζει το module να δουλεύει σαν client ή server ή και τα δύο ταυτόχρονα. **<mode> = 1 : Station (Client) mode** **<mode> = 2 : SoftAP (Server) mode** **<mode> = 3 : SoftAP + Station mode** *Η ρύθμιση αποθηκεύεται στην μνήμη FLASH.*
AT+CIPSEND=<link ID>,<length>	**SEND OK** *(Για επιτυχής αποστολή)*	Ξεκινάει μία αποστολή δεδομένων στην σύνδεση με αναγνωριστικό **<link ID>**, μεγέθους **<length>** bytes. Μετά την

	ERROR *(Για αποτυχημένη αποστολή)*	εκτέλεση της εντολής, για να ξεκινήσει η σύνδεση, πρέπει να σταλεί ο χαρακτήρας **« > »** (δεξί βελάκι).
AT+CIPSTA_DEF=<ip>	**OK**	Θέτει μια σταθερή **client IP (STAIP)** διεύθυνση **<ip>** για το module. *Η διεύθυνση αποθηκεύεται στην μνήμη FLASH.*
AT+CIPCLOSE=<link ID>	**OK**	Τερματίζει την αποστολή δεδομένων στην σύνδεση με αναγνωριστικό **<link ID>**.
AT+UART_DEF=<baudrate>, <databits>,<stopbits>, <parity>,<flow control>	**OK**	Ρυθμίζει την σειριακή σύνδεση του module. Από τις παραμέτρους, εμάς μας ενδιαφέρει μόνο ή ταχύτητα **<baudrate>**. *Τα στοιχεία αποθηκεύονται στην μνήμη FLASH.*
AT+CIPSNTPCFG=<enable> [,<timezone>]	**OK**	Ρυθμίζει την κατάσταση και ζώνη ώρας του SNTP πρωτοκόλλου.
AT+CIPSNTPTIME?	**+CIPSNTPTIME:<time>** **OK**	Λαμβάνει την τιμή της τρέχουσας ώρας.

Για περισσότερες πληροφορίες για όλες τις εντολές που μπορεί να δεχτεί το **ESP8266**, μπορείτε να «κατεβάσετε» το πλήρες σετ εντολών στον παρακάτω σύνδεσμο:

http://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

Συνεχίζοντας με την ρύθμιση του module, αφού τώρα έχουμε συνδεθεί και μπορούμε να επικοινωνήσουμε με το module, θα συνδεθούμε στο τοπικό δίκτυο μας.

Πριν συνδεθούμε όμως, πρέπει να γνωρίζουμε το **κλειδί ασφαλείας Wi-Fi** του δικτύου και αν δεν είμαστε διαχειριστές του δικτύου, να ενημερώσουμε τους διαχειριστές για τις επικείμενες πράξεις μας.

Ύστερα, θα πρέπει να θέσουμε μια **σταθερή IP διεύθυνση** για το ESP. Αυτό, γιατί ο router του δικτύου μας αναθέτει IP διευθύνσεις στις συσκευές που είναι συνδεδεμένες με αυτό, με διάφορα κριτήρια. Αυτό συνεπάγεται στο ότι κάθε συσκευή, κάθε μία ή δύο μέρες, μπορεί να έχει διαφορετική **IP** απ' ότι είχε τις προηγούμενες μέρες.

Οι παραπάνω ρυθμίσεις θα γίνουν από το σύστημα μας κατά την εκκίνηση, άρα δεν θα ασχοληθούμε, προς το παρόν, με αυτές. Η μόνη ρύθμιση που πρέπει να γίνει χειροκίνητα μέσω υπερ-τερματικού είναι η **ταχύτητα (baud rate)** της σειριακής σύνδεσης, για να μπορεί να επικοινωνεί η πλακέτα με το module.

6.3.4.5. Ρύθμιση σειριακής ταχύτητας (Baud Rate).

Η διαθέσιμες ταχύτητες είναι προκαθορισμένες από ένα σύνολο τιμών. Δεν μπορούμε να επιλέξουμε μια τυχαία τιμή.

7. Ιδέα και σχέδιο

Όπως αναφέρει και ο τίτλος της διπλωματικής εργασίας αυτής, σκοπός μας είναι ο σχεδιασμός και η υλοποίηση ενός συστήματος που θα μας δίνει την δυνατότητα να συλλέγουμε και να καταγράφουμε δεδομένα από την πλακέτα μας και να έχουμε τον πλήρη έλεγχο της, απομακρυσμένα, μέσω του Διαδικτύου.

Τί είδους δεδομένα όμως;

Όσο περισσότερα μπορούμε. Είτε είναι μετρήσεις αισθητήρων, καταστάσεις PIN, Relay, περιεχόμενα μνήμης EEPROM κ.α.

Πως θα γίνει απομακρυσμένα μέσω Διαδικτύου;

Καταρχήν, θα χρειαστούμε έναν **server** στο τοπικό (LAN) δίκτυο με το **Wi-Fi Module ESP8266**. Για λόγους απλότητας, από ‘δω και πέρα θα το αναφέρουμε ως **ESP**.

Θα μπορούσαμε να χρησιμοποιήσουμε έναν υπολογιστή στο τοπικό δίκτυο μας για τον ρόλο του server, αλλά αυτό, προφανώς, απαιτεί έναν υπολογιστή. Για αυτοματισμούς ενσωματωμένων συστημάτων, οι απαιτήσεις δεν είναι τόσο υψηλές για να χρειαστούν επιπλέον υλικά. Άρα, μπορούμε να χρησιμοποιήσουμε τον ίδιο τον μικροελεγκτή του συστήματος μας, ως **server**.

Υπάρχει αρκετός χώρος στον μικροελεγκτή για να γίνει web server;

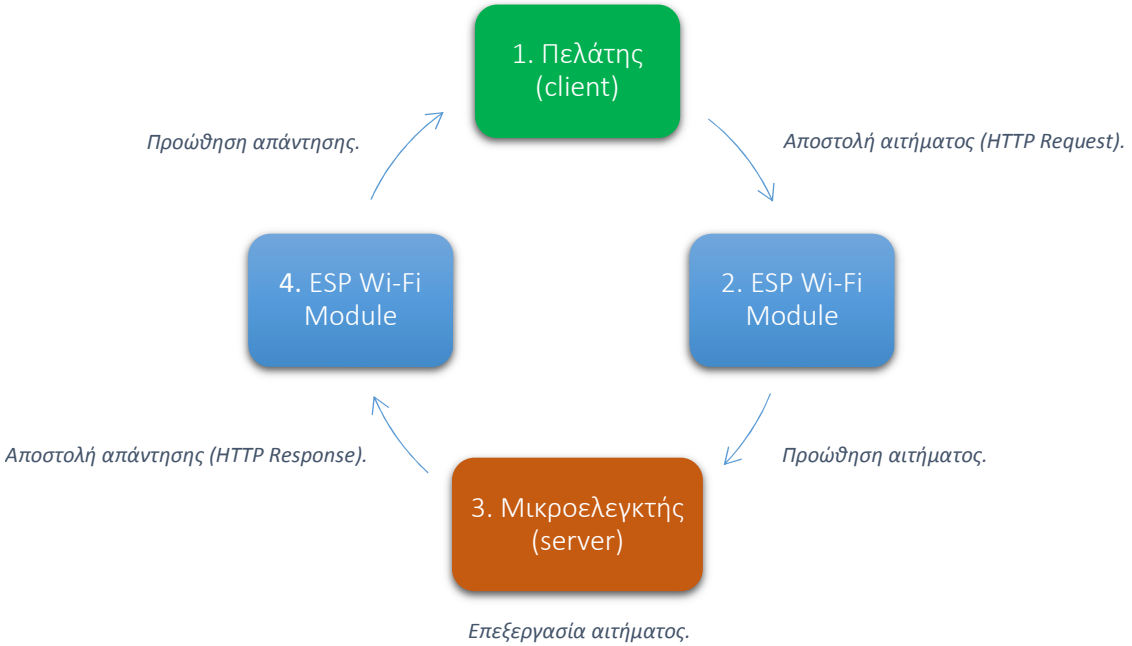
Ναι, υπάρχει αρκετός χώρος. Τον τρόπο με τον οποίο το επιτυγχάνουμε αυτό, θα τον αναλύσουμε στα επόμενα κεφάλαια, μαζί με τον πηγαίο κώδικα.

Ο **server** θα είναι ο δίαυλος μέσω του οποίου θα μπορούν οι απομακρυσμένοι χρήστες να ελέγχουν τον μικροελεγκτή.

Αυτό σημαίνει ότι αν ένας πελάτης **(client)** θελήσει να επικοινωνήσει με τον μικροελεγκτή**,** το αίτημα του (**HTTP request**) θα περάσει πρώτα από το **ESP**, το οποίο με την σειρά του, θα το προωθήσει στον μικροελεγκτή (**server**). Ο server θα αναλύσει το αίτημα και θα απαντήσει, στέλνοντας την απάντηση (**HTTP response**) πίσω στο ESP, όπου αυτό με την σειρά του, θα την προωθήσει πίσω στον πελάτη.

Υπενθυμίζουμε ότι τον χειρισμό των αιτημάτων (HTTP requests), την αναλαμβάνει ο μικροελεγκτής και όχι το ESP. Το ESP είναι ο δίαυλος με το Διαδίκτυο και απλώς προωθεί τα αιτήματα στον μικροελεγκτή μας.

Παρατίθεται διάγραμμα με την σειρά των βημάτων που γίνεται η απομακρυσμένη επικοινωνία.



8. Εισαγωγή και Ανάλυση στον Πηγαίο Κώδικα

Πριν αναφερθούμε στον πηγαίο κώδικα μας, θα εξηγήσουμε τα χαρακτηριστικά και τις δυνατότητες του ESP.

Το ESP περιέχει και αυτό έναν μικροελεγκτή με ενσωματωμένο το TCP/IP πρωτόκολλο για να μπορεί να διαχειρίζεται HTTP αιτήματα και να πράττει ανάλογα. Ουσιαστικά μπορεί να λειτουργεί και σαν πελάτης (station) συνδεδεμένος σε ένα δίκτυο, αλλά και σαν ένα αυτόνομο δίκτυο Wi-Fi (Access Point).

Η **Espressif**, κατασκευαστής του ESP, παρέχει δικιά της πλατφόρμα για τον προγραμματισμό του ESP. Δηλαδή ο χρήστης μπορεί να φτιάξει δικά του συστήματα βασισμένα στο ESP χωρίς την χρήση επιπλέον υλικού (π.χ. εξωτερική πλακέτα με δικό της μικροελεγκτή).

Αν γίνεται αυτό όμως, τότε γιατί εμείς χρησιμοποιούμε εξωτερική πλακέτα;

Πρώτον, το ESP παρέχει λιγότερους ακροδέκτες διαθέσιμους για τον χρήστη. Π.χ. Το ESP8266 παρέχει μόνο **έναν (1)** ακροδέκτη ADC (Analog – Digital Converter). Αυτό περιορίζει αρκετά το ESP, όσον αφορά εφαρμογές που χρησιμοποιούν αισθητήρια όργανα. Όσον αφορά ακροδέκτες γενικής χρήσης I/O, το ESP προσφέρει μόνον **έντεκα (11)**, λιγότερους απ' ότι προσφέρει ένας μικροελεγκτής ATmega. Άρα, αν θέλαμε να συνδέσουμε μία οθόνη LCD ή LED (7 Segment), θα είχαμε πρόβλημα.

Δεύτερον, το ESP, από μόνο του, είναι μία μονάδα που προορίζεται για επαγγελματικές ή βιομηχανικές εφαρμογές και δεν είναι κατάλληλη για έναν αρχάριο, όσον αφορά τα ενσωματωμένα συστήματα (embedded systems). Ο χρήστης θα πρέπει να κατέχει ήδη αρκετή εμπειρία τόσο ως προγραμματιστής αλλά και όσο ως χρήστης ενσωματωμένων συστημάτων. Γι' αυτό λοιπόν, συνήθως χρησιμοποιείται παράλληλα με μία άλλη πλακέτα έχει τον ρόλο του «εγκεφάλου» του συστήματος. Βεβαίως, εννοείται ότι ένας έμπειρος χρήστης μπορεί να αναπτύξει μικρές εφαρμογές στο ίδιο το ESP χωρίς την χρήση επιπλέον πλακέτας, εκτός αν θελήσει να επεκτείνει την εφαρμογή του.

Είναι σημαντικό να αναφέρουμε ότι υπάρχουν πολλά project στο Διαδίκτυο που χρησιμοποιούν Wi-Fi Module σε συνδυασμό με μικροελεγκτές για ασύρματη μετάδοση δεδομένων. Τα περισσότερα από αυτά χρησιμοποιούν το **Arduino**, όπου είναι το πιο δημοφιλές. Εκτός από αυτό όμως, χρησιμοποιούν και έτοιμες **βιβλιοθήκες** (libraries) που παρέχουν έτοιμες συναρτήσεις για την επικοινωνία της πλακέτας με το ESP.

Στο δικό μας σύστημα δεν χρησιμοποιούμε καμία εξωτερική βιβλιοθήκη για την επικοινωνία με το ESP και για οτιδήποτε άλλο.

Ο λόγος που γίνεται αυτό δεν είναι ότι έχουμε κάποια ιδιαίτερη προκατάληψη για τις βιβλιοθήκες αλλά επειδή χωρίς την βοήθεια τους, γίνεται πιο κατανοητή η διαδικασία με την οποία επικοινωνεί ο μικροελεγκτής με το ESP. Αυτό έχει ως αποτέλεσμα, να είναι πιο εύκολο για τον προγραμματιστή να κάνει **αποσφαλμάτωση** (debugging), γιατί ο κώδικας θα 'ναι δικός του και θα γνωρίζει καλύτερα πως δουλεύει και ποιο είναι το σκεπτικό πίσω από κάθε γραμμή κώδικα.

Τέλος, εφόσον μπορέσει και φτιάξει τις δικές του συναρτήσεις, ο προγραμματιστής έχει την δυνατότητα, αν θελήσει, να φτιάξει τις δικές του βιβλιοθήκες!

Η χρήση βιβλιοθηκών για το ESP θα έκανε την διπλωματική εργασία αυτή σαν άλλο ένα από τα πολλά παραδείγματα και project που υπάρχουν στο Διαδίκτυο.

Για την συγγραφή του κώδικα χρησιμοποιούμε την πλατφόρμα **Atmel Studio** όπου έγινε αναφορά και επεξήγηση σε προηγούμενο κεφάλαιο. Η γλώσσα προγραμματισμού είναι η **C**, γι' αυτό και οποιαδήποτε εμπειρία με προγραμματισμό σε C/C++ είναι σίγουρα ωφέλιμη.

Γενικότερα, καθώς η εργασία αυτή στοχεύει σε κοινό που είναι ήδη εξοικειωμένο, ακόμα και σε στοιχειώδης επίπεδο, με τον προγραμματισμό, ο αναγνώστης έχει σίγουρα μια ιδέα για τους τεχνικούς όρους προγραμματισμού π.χ.: μεταβλητές, σταθερές, συναρτήσεις, loops κλπ.

Γι' αυτό, πιστεύουμε πως δεν χρειάζεται επεξήγηση για τους απλούς τεχνικούς όρους.

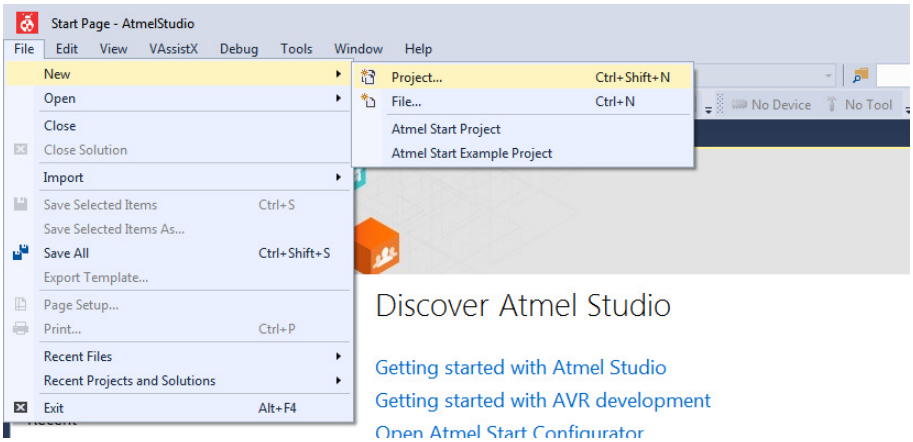
Επειδή όμως προγραμματίζουμε μικροελεγκτές, υπάρχουν κάποιοι όροι που δεν είναι γνωστοί ακόμα και σε έμπειρους προγραμματιστές. Γι' αυτούς θα γίνει κατάλληλη επεξήγηση.

Η ανάλυση του κώδικα θα γίνει με την σειρά που τρέχει το πρόγραμμα. Θα εξηγούμε πως δουλεύει το πρόγραμμα σαν να τρέχει εκείνη την στιγμή. Σε κάθε εντολή του κώδικα υπάρχουν σχόλια που εξηγούν τον ρόλο της εντολής. Σε κάποια σημεία θα γίνει περαιτέρω επεξήγηση για το τί θέλουμε να επιτύχουμε σε εκείνο το σημείο.

Για αρχή θα δώσουμε οδηγίες για την δημιουργία ενός νέου project στο **Atmel Studio** και θα πούμε λίγα λόγια για την δομή ενός προγράμματος για μικροελεγκτές.

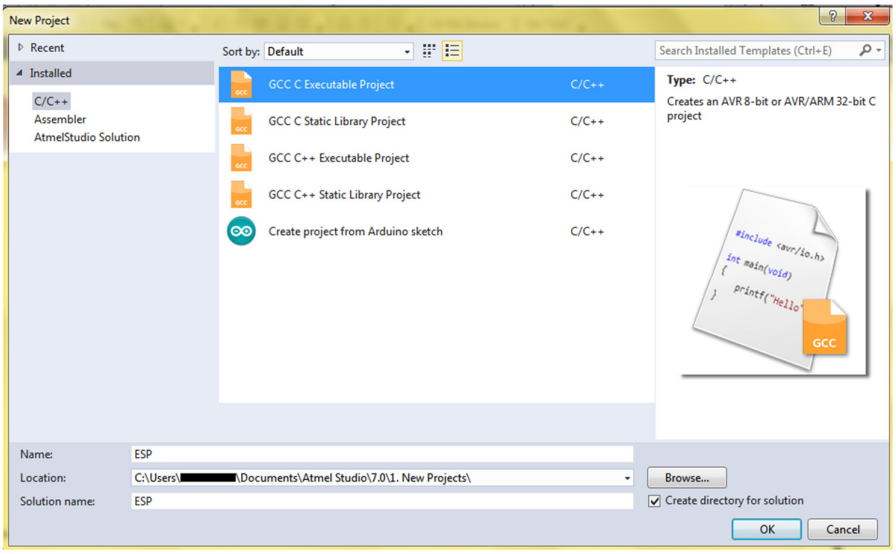
8.1 Δημιουργία Atmel Studio Project

Πρώτα δημιουργούμε ένα νέο project στο Atmel Studio.



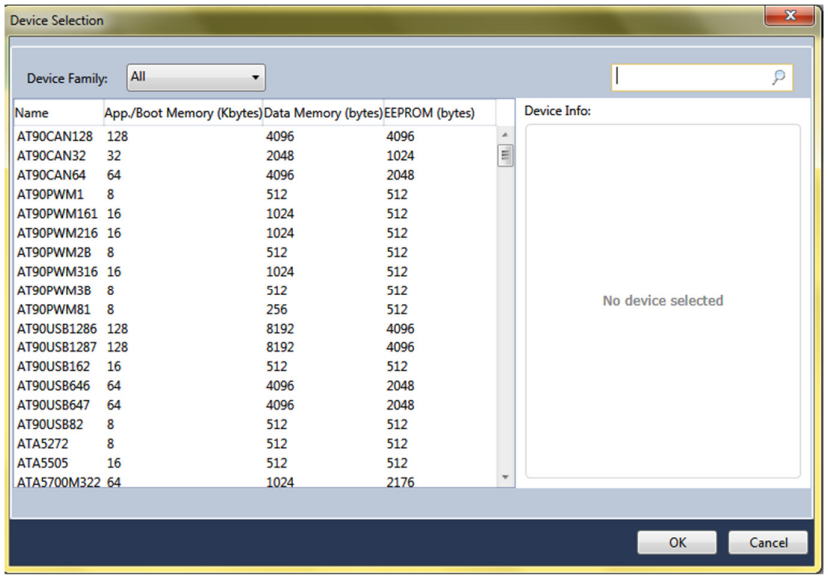
Εικόνα 8.1.1. Δημιουργία νέου project.

Επιλέγουμε τον τύπο του project και βάζουμε όνομα της επιλογής μας.



Εικόνα 8.1.2. Επιλογή τύπου και ονομασία project.

Στη συνέχεια, θα εμφανιστεί μία λίστα με τους υποστηριζόμενους μικροελεγκτές, για να επιλέξουμε ποιον μικροελεγκτή (**target**) στοχεύει το project. Ο χρήστης επιλέγει τον μικροελεγκτή που διαθέτει η πλακέτα του. Στην εργασία αυτή χρησιμοποιούμε τον **ATmega32**. Κάνουμε μια αναζήτηση με το όνομα του μικροελεγκτή και τον επιλέγουμε.



Εικόνα 8.1.3. Αναζήτηση και επιλογή target μικροελεγκτή.

Αφού ολοκληρώσουμε τα παραπάνω βήματα, θα εμφανιστεί ο editor με έναν έτοιμο κώδικα.
Όπως σε κάθε νέο project για **C**, υπάρχει πάντα η συνάρτηση **main** όπου εκτελείται πρώτη κατά την εκτέλεση ενός προγράμματος **C.**

```
#include <avr/io.h>


int main(void)
{
    /* Replace with your application code */
    while (1)
    {
    }
}
```

Όμως, θα παρατηρήσουμε ότι υπάρχει ένας περίεργος βρόγχο **while**. Συγκεκριμένα, είναι ένας ατέρμονος βρόγχος (**infinite loop**), δηλαδή το πρόγραμμα δεν θα βγει ποτέ από τον βρόγχο αυτόν και η συνάρτηση δεν θα τελειώσει ποτέ.

Γιατί όμως έχουμε αυτόν τον περίεργο βρόγχο στο πρόγραμμα μας;

Η απάντηση είναι απλή.

Ο λόγος που χρησιμοποιούμε τον ατέρμονο βρόγχο είναι διότι τα προγράμματα των μικροελεγκτών και γενικά των αυτοματισμών, πρέπει να λειτουργούν (συνήθως) **συνέχεια**.

Φανταστείτε τον θερμοστάτη του σπιτιού σας να άνοιγε για **ένα (1)** δευτερόλεπτο, να έλεγχε την θερμοκρασία και να έκλεινε. Για να ελέγξει την θερμοκρασία πάλι, θα έπρεπε να τον κλείσουμε και να τον ανοίξουμε πάλι.

Γι’ αυτό λοιπόν, σε αυτόν τον βρόγχο βάζουμε τις συναρτήσεις που χρειαζόμαστε για να κάνουμε την δουλειά μας. Σχεδόν σε όλα τα προγράμματα για αυτοματισμούς υπάρχει ένας βρόγχος. Όχι σε όλα όμως.

Πριν φτάσουμε όμως στον βρόγχο, ας γράψουμε τον απαραίτητο κώδικα για την αρχικοποίηση του προγράμματος μας.

Πρώτη μας ενέργεια είναι ο ορισμός της συχνότητας του μικροελεγκτή. Στα χαρακτηριστικά της πλακέτας του χρήστη θα αναγράφεται αν ο μικροελεγκτής κάνει χρήση **εσωτερικού ταλαντωτή (Internal RC Oscillator)** ή **εξωτερικού κρυστάλλου (External Crystal)** για τον συγχρονισμό του μικροελεγκτή και ποια η συχνότητα του. Ουσιαστικά αυτή η συχνότητα είναι η ταχύτητα με την οποία ο μικροελεγκτής εκτελεί εντολές. Όσο μεγαλύτερη τόσο πιο γρήγορος και ο μικροελεγκτής, αλλά αυτό όμως δεν είναι πάντα και το καλύτερο.

Ειδικότερα, οι μικροελεγκτές σειράς ATmega έχουν ο καθένας και ένα μέγιστο όριο συχνότητας. Ο **ATmega32** έχει απόλυτο μέγιστο όριο τα **20 MHz,** με προτεινόμενο μέγιστο όριο τα **16 MHz**. Επειδή όμως το ESP χρειάζεται να τροφοδοτείται με **3.3V**, άρα και ο μικροελεγκτής θα πρέπει να κάνει το ίδιο. Συνήθως, οι μικροελεγκτές τροφοδοτούνται με **5V** το λιγότερο, αλλά επειδή όπως είδαμε, μερικές φορές υπάρχει η ανάγκη να γίνει η χρήση των 3.3V, υπάρχουν μοντέλα μικροελεγκτών που σχεδιάστηκαν για να μπορούν να τροφοδοτούνται με χαμηλή τάση όπως τα 3.3V. Για τους μικροελεγκτές σειράς ATmega, υπάρχει η κατάληξη **“L”** στην ονομασία τους, π.χ.: ATmega8L, ATmega32L κλπ. Γι’ αυτό στην εργασία αυτή χρησιμοποιούμε τον **ATmega32L** για να μπορούμε να τροφοδοτούμε αυτόν, άρα και το ESP, με τάση 3.3V.

Η χρήση χαμηλής τάσης όμως, έχει ένα μειονέκτημα. Ναι μεν, μπορούμε να χρησιμοποιούμε τον μικροελεγκτή και οτιδήποτε περιφερειακά με 3.3V, αλλά μειώνεται η μέγιστη συχνότητα συγχρονισμού του μικροελεγκτή, δηλαδή η ταχύτητα του. Για τον ATmega32, η μέγιστή προτεινόμενη συχνότητα μειώνεται από 16 MHz που αναφέραμε πιο πριν, στα 8 MHz. Δηλαδή, η ταχύτητα μπορεί να μειωθεί στο μισό!

Εμείς θα χρησιμοποιήσουμε τον μικροελεγκτή σε συχνότητα **4 MHz**, που δεν είναι κοντά στο όριο, για ευνόητους λόγους, αλλά ούτε είναι πολύ χαμηλή.

```
 //16MHz - 5.0V
 //8MHz - 3.3V
#define F_CPU 4000000UL     //Συχνότητα (Hz) του μικροελεγκτή.
```

4 MHz = 4.000.000 Hz. Το “**UL**” λέει στον compiler ότι πρόκειται για **Unsigned Long** αριθμό. Δεν είναι απαραίτητο, αλλά το βάζουμε για να είμαστε σίγουροι, όπου αυτή είναι και η καλή πρακτική για προγραμματισμό σε C.

Η σταθερά **F_CPU** είναι μία προκαθορισμένη σταθερά που ορίζεται από τις εσωτερικές βιβλιοθήκες του Atmel Studio και αναφέρεται στην συχνότητα του μικροελεγκτή για τον οποίο προορίζεται το πρόγραμμα. Ορίζοντας την ξανά, στον πρόγραμμα μας, αγνοείται η προκαθορισμένη τιμή που είχε και χρησιμοποιείται η νέα συχνότητα που θέλουμε.

Αν δεν οριστεί από εμάς, το πρόγραμμα χρησιμοποιεί την προκαθορισμένη (default) τιμή των **1 MHz**. Επίσης, πρέπει να την ορίσουμε πριν φορτώσουμε την βιβλιοθήκη **<util/delay.h>**, γιατί ο compiler θα χρησιμοποιήσει τον ορισμό της εσωτερικής αυτής βιβλιοθήκης.

Η συχνότητα έχει σημαντικό ρόλο και στις λειτουργίες των ρολογιών/μετρητών (**timers/counters**) του μικροελεγκτή. Οι **Timers** είναι κυκλώματα που τρέχουν ανεξάρτητα από την ροή του προγράμματος. Δηλαδή, μπορούμε με την χρήση τους, να εκτελούμε «ταυτόχρονα» συναρτήσεις/λειτουργίες με το κυρίως πρόγραμμα (η συνάρτηση **main()**).

Στην πραγματικότητα δεν είναι ταυτόχρονα, απλώς το κυρίως πρόγραμμα σταματάει την ροή του για να εκτελέσει τον κώδικα του timer και μετά επιστρέφει στην κανονική ροή του. Αυτή η διαδικασία ονομάζεται **interrupt** (διακοπή). Επειδή όλη η διαδικασία γίνεται σχεδόν ακαριαία, η ροή του προγράμματος δεν επηρεάζεται.

Οι **Timers** είναι χρήσιμοι όταν θέλουμε να μετρήσουμε χρόνους σε δευτερόλεπτα, λεπτά κλπ. Π.χ. μπορούμε να φτιάξουμε ένα ψηφιακό ρολόι.
Περαιτέρω ανάλυση για το τί είναι οι **Timers**, πως δουλεύουν κλπ., γίνεται σε επόμενο κεφάλαιο.

Συνεχίζουμε με κάποιες βασικές βιβλιοθήκες και χρήσιμες σταθερές.

8.2 Φόρτωση βιβλιοθηκών και ορισμός σταθερών

```
#define F_CPU 4000000UL

#include <avr/io.h>         //Βιβλιοθήκη Εισόδου/Εξόδου του μικροελεγκτή.
#include <avr/interrupt.h>  //Βιβλιοθήκη για συναρτήσεις των ρολογιών.
#include <avr/pgmspace.h>   //Βιβλιοθήκη για συναρτήσεις μνήμης FLASH.
#include <avr/eeprom.h>     //Βιβλιοθήκη για συναρτήσεις μνήμης EEPROM.
#include <avr/wdt.h>        //Βιβλιοθήκη για συναρτήσεις του watchdog timer.

#include <util/delay.h>     //Βιβλιοθήκη για συναρτήσεις καθυστερήσεων.
#include <util/atomic.h>    //Βιβλιοθήκη για «ατομικές» συναρτήσεις.
#include <stdio.h>          //Βιβλιοθήκη για συναρτήσεις εκχώρησης τιμών.
#include <stdlib.h>         //Βιβλιοθήκη για συναρτήσεις μετατροπών.
#include <string.h>         //Βιβλιοθήκη για συναρτήσεις συμβολοσειρών.
#include <ctype.h>          //Βιβλιοθήκη για συναρτήσεις ελέγχου χαρακτήρων.
#include <math.h>           //Βιβλιοθήκη για μαθηματικές συναρτήσεις.

#include "server/settings.h" //Βοηθητική βιβλιοθήκη με δικές μας σταθερές.
#include "helper.h"          //Βοηθητική βιβλιοθήκη με δικές μας συναρτήσεις.
#include "events.h"          //Βοηθητική βιβλιοθήκη με δικές μας συμβολοσειρές για καταγραφή γεγονότων (logging).
```

Στην συνέχεια ορίζουμε την ταχύτητα για την σειριακή μετάδοση. Επιλέξαμε την ταχύτητα των **9600 bps** γιατί είναι η μεγαλύτερη ταχύτητα με την οποία μπορούμε να έχουμε την πιο αξιόπιστη μετάδοση δεδομένων, με το μικρότερο ποσοστό σφάλματος.

```
#define BAUD_RATE       9600 //Η καλύτερη ταχύτητα. Πιο υψηλή από αυτήν και χάνουμε δεδομένα.
```

Για κάθε τιμή ταχύτητας υπάρχει και το ανάλογο **σφάλμα (error)**, λόγω της συχνότητας συγχρονισμού του μικροελεγκτή. Δηλαδή, για συχνότητα συγχρονισμού **4 MHz** σε ATmega32, με σειριακό ρυθμό μετάδοσης **9600 bps**, έχουμε σφάλμα ποσοστού **0.2%**.

Παρατίθεται πίνακας με τα αντίστοιχα ποσοστά σφάλματος για συχνότητα συγχρονισμού 4 MHz για ATmega32.

Πίνακας 8.2.1. Αντιστοιχίες ρυθμού μετάδοσης με ποσοστό σφάλματος σε συχνότητα συγχρονισμού 4 MHz, για ATmega32.

Baud Rate (bps)	f_{osc} = 4.0000MHz			
	U2X = 0		U2X = 1	
	UBRR	Error	UBRR	Error
2400	103	0.2%	207	0.2%
4800	51	0.2%	103	0.2%
9600	25	0.2%	51	0.2%
14.4k	16	2.1%	34	-0.8%
19.2k	12	0.2%	25	0.2%
28.8k	8	-3.5%	16	2.1%
38.4k	6	-7.0%	12	0.2%
57.6k	3	8.5%	8	-3.5%
76.8k	2	8.5%	6	-7.0%
115.2k	1	8.5%	3	8.5%
230.4k	0	8.5%	1	8.5%
250k	0	0.0%	1	0.0%
0.5M	–	–	0	0.0%
1M	–	–	–	–
Max (1)	250Kbps		0.5Mbps	

Όπως θα παρατηρήσετε όμως, η ταχύτητα των 9600 bps δεν είναι η μεγαλύτερη με το μικρότερο σφάλμα. Θα μπορούσαμε να χρησιμοποιήσουμε την ταχύτητα των 19200 bps που κι αυτή προσφέρει ποσοστό σφάλματος 0.2%. Ακόμα καλύτερα, θα μπορούσαμε να χρησιμοποιήσουμε την ταχύτητα των 500.000 = 0.5M bps, όπου έχει μηδενικό ποσοστό σφάλματος. Γιατί τότε χρησιμοποιούμε την ταχύτητα των 9600 bps;

Όπως θα δούμε παρακάτω, επειδή κάθε φορά που λαμβάνουμε έναν χαρακτήρα σειριακά, κάνουμε κάποιους ελέγχους, υπάρχει η πιθανότητα ο επόμενος χαρακτήρας να σταλεί, πριν τελειώσουμε εμείς τον έλεγχο του προηγούμενου και άρα να μην τον λάβουμε. Όσο πιο μεγάλη η ταχύτητα σειριακής μετάδοσης, τόσο πιο πολλούς χαρακτήρες θα «χάσουμε». Άρα πρέπει να βρούμε την κατάλληλη ταχύτητα που να προσφέρει το **μικρότερο σφάλμα**, όσον αφορά τον συγχρονισμό του μικροελεγκτή, αλλά και να είναι **αρκετά αργή ώστε να προλαβαίνουμε** να λαμβάνουμε όλους τους χαρακτήρες.

Ακόμα και κάνοντας τον κώδικα μας, κατά την εκτέλεση των ελέγχων που αναφέραμε, όσο πιο αποδοτικό γίνεται για να μειώσουμε τον χρόνο εκτέλεσης του, πάλι δεν μπορούμε να χρησιμοποιήσουμε μεγαλύτερη από 9600 bps ταχύτητα.

Η μόνη λύση είναι να αυξήσουμε την ταχύτητα του μικροελεγκτή, δηλαδή την συχνότητα συγχρονισμού του που αναλύσαμε στο προηγούμενο κεφάλαιο. Όπως είδαμε όμως, δεν μπορούμε για λόγους που αναφέραμε.
Άρα, ο χρήστης/προγραμματιστής αρχίζει και καταλαβαίνει ότι υπάρχουν πολλές έννοιες και λειτουργίες που πρέπει να έχει υπόψιν του, για την ανάπτυξη εφαρμογών σε ενσωματωμένα συστήματα. Μία μικρή αναγκαστική παραχώρηση στο σύστημα μας (χρήση των 3.3V), βλέπουμε τι επίδραση μπορεί να έχει για το υπόλοιπο της εφαρμογής μας.

Στην συνέχεια αναφέρονται οι σταθερές για την μετατροπή της αναλογικής τιμής της θερμοκρασίας, από τον NTC θερμίστορ, σε ψηφιακή. Για να γίνει αυτό χρησιμοποιούμε την εξίσωση των **Steinhart-Hart** :

$$1/T = A + B * ln(R/Rt) + C * ln(R/Rt)^2 + D * ln(R/Rt)^3$$

Ο συγκεκριμένος θερμίστορ χρησιμοποιεί τρείς (3) συντελεστές: **A**, **B** και **C.**

$$1/T = A + B * ln(R/Rt) + C * ln(R/Rt)^2$$

Στα χαρακτηριστικά του θερμίστορ αναφέρονται κάποιες σταθερές αναλογίες **Αντίστασης – Θερμοκρασίας**. Δηλαδή σε **X** Ohm έχουμε **Y** Kelvin βαθμούς.

Άρα μπορούμε να λύσουμε την εξίσωση ως προς τις σταθερές **A,B,C.**

Στο Διαδίκτυο υπάρχουν πολλές ιστοσελίδες που προσφέρουν τις λύσεις των σταθερών **A, B** και **C**, για δεδομένες τιμές **Θερμοκρασίας (T) – Αντίστασης (R)**.

```
//Σταθερές για την μετατροπή της θερμοκρασίας.

#define REF_VOLT        3.3                 //Η τάση αναφοράς.

//Συντελεστές για την εξίσωση Steinhart-Hart.
#define a               0.000946018574683
#define b               0.000242434141321
#define c               0.000000224104934229

#define R0              10000.0             //Αντίσταση Ohm στους 25 βαθμούς Κελσίου.
```

Παρακάτω γίνεται αναφορά σε σταθερές για την μετατροπή αναλογικού σε ψηφιακό (ADC). Περαιτέρω ανάλυση θα γίνει αργότερα.

```
//Σταθερές για την μετατροπή αναλογικού σε ψηφιακό (ADC).
/*
    μέγεθος_buffer = 4^ν, όπου ν = αριθμός επιπλέον επιθυμητών bits για την μετατροπή.

    Θέλουμε 2 επιπλέον bits, άρα n = 2.
    10 + 2 = 12 bits σύνολο.

    Δες: Application Note AVR121, για περισσότερες πληροφορίες.
*/
#define BUFFERSIZE      16                  //4^2 = 16
#define ADC_PIN         4                   //Ακροδέκτης ADC για τον θερμίστορ.
```

Στην συνέχεια ορίζουμε κάποιες **δομές (structs)** για το πρόγραμμα μας και σταθερές για την κατάσταση μίας εντολής που στέλνουμε.

```
//Struct για την ημ/νία και ώρα.
typedef struct TIME_STRUCT
{
    volatile uint8_t sec;
    volatile uint8_t min;
    volatile uint8_t hour;

    volatile uint8_t day;
    volatile uint8_t mon;
    volatile uint16_t year;

} TIME_STRUCT;

//Struct για τις HTTP συνδέσεις.
typedef struct HTTP_CONNECTION
{
    char *request[ESP_CONNECTION_MAX_REQUESTS]; //Πίνακας με τα αιτήματα της σύνδεσης.
    volatile uint8_t total_reqs;                //Σύνολο αιτημάτων.
    volatile uint8_t closing;                   //Σημαία για την εκκίνηση της διαδικασίας τερματισμού σύνδεσης.
    volatile uint8_t closed;                    //Σημαία για την κατάσταση της σύνδεσης.

} HTTP_CONNECTION;

/*
    Struct για JSON αντικείμενα με 8-bit τιμές.

    Περιέχει έναν μέγιστο αριθμό MAX_JSON_PROP_NUM ιδιοτήτων (properties), καθεμία με ένα όνομα με μέγιστο αριθμό χαρακτήρων
    MAX_JSON_PROP_LEN (συμπεριλαμβανομένου και του τερματικού χαρακτήρα - null char).
    Κάθε ιδιότητα περιέχει έναν πίνακα με μέγιστο αριθμό 8-bit τιμών MAX_JSON_ARRAY_LEN.

    Κάθε τιμή των πινάκων αντιστοιχεί σε μία σημαία "non-empty" όπου σημαίνει αν είναι κενή ή όχι.
*/
typedef struct JSON_STRUCT8
{
    char props[MAX_JSON_PROP_NUM][MAX_JSON_PROP_LEN];       //Πίνακες με τα ονόματα των ιδιοτήτων.
    uint8_t values[MAX_JSON_PROP_NUM][MAX_JSON_ARRAY_LEN];  //Πίνακες με τις 8-bit τιμές των ιδιοτήτων.
    uint8_t used[MAX_JSON_PROP_NUM][MAX_JSON_ARRAY_LEN];    //Πίνακες σημαίων "non-empty" για τις 8-bit τιμές των ιδιοτήτων.
    uint8_t count;                                          //Συνολικός αριθμός ιδιοτήτων.

} JSON_STRUCT8;

//Struct για την σειριακή μετάδοση.
typedef struct USART_BUFFER
{
//Enums για την κατάσταση μίας εντολής.
typedef enum COMMAND_STATUS
{
    CMD_ERROR = -1,
    CMD_NONE,
    CMD_TIMED,
    CMD_SUCCESS,
    CMD_OVLOW

} COMMAND_STATUS;
```

8.3 Ορισμοί Συναρτήσεων και Επεξήγηση Όρων

Στη συνέχεια ορίζουμε τις συναρτήσεις που θα χρησιμοποιήσουμε. Παρατηρείστε ότι ορίζουμε όλες τις συναρτήσεις με τον όρο **static**. Αυτός ο όρος χρησιμοποιείται για συναρτήσεις που δεν θα κληθούν εκτός του αρχείου του οποίου ορίζονται. Αυτό έχει ως αποτέλεσμα να μειώσει το μέγεθος του προγράμματος μας, όπου είναι κι αυτό που επιθυμούμε. Από αυτές θα αναλύσουμε τις πιο σημαντικές.

```
//*****************************************************************************************************************************************************
static void init();
static void init_ADC();
static void init_timer();
static char init_esp();
static void wait_esp();
//*****************************************************************************************************************************************************
static void init_json_struct8(JSON_STRUCT8 *json_struct, uint8_t keep_props);
//*****************************************************************************************************************************************************
static void eep_clear();
static void eep_store_string(const char *str);
//*****************************************************************************************************************************************************
static void reset_timer1();
static void reset_timer2();
//*****************************************************************************************************************************************************
static void USART_init(uint32_t ubrr_value);
static void USART_write_char(char data, uint8_t slow);
static void USART_write_string(char *str, uint8_t append, uint8_t slow);
//*****************************************************************************************************************************************************
static void USART_BUFFER_update(uint16_t cmd_start, USART_BUFFER *buf, uint16_t cmd_end, char *res);
static void send_command(char *command, uint8_t timeout_sec, char *response_str, uint8_t appendOnCmd, uint8_t appendOnRes, uint8_t slow);
//*****************************************************************************************************************************************************
static void listen();
static void find_request(USART_BUFFER *buf, uint16_t offset);
//*****************************************************************************************************************************************************
static void handle_request(char *request, uint8_t id, uint8_t req_index);
//*****************************************************************************************************************************************************
static void BUILD_HTTP_HEADER_200(char *str, uint16_t length, CONTENT_TYPE type);
static void SEND_HTTP_HEADER_204(uint8_t id);
static void SEND_HTTP_HEADER_400(uint8_t id);
static void SEND_HTTP_HEADER_507(uint8_t id);
static void SEND_HTTP_FILE(const char file[], uint8_t id, CONTENT_TYPE type, uint8_t isLog);
static void SEND_INDEX_HTML(const char *file[], uint8_t id);
//*****************************************************************************************************************************************************
static void RST();
static void CWJAP_DEF(const char ssid[], const char key[]);
static void CIPSEND(char *data, uint8_t id, uint8_t first, uint8_t no_ans);
static void CIPSEND_VAR(char *data, uint8_t id, uint8_t first, uint16_t length, uint8_t no_ans);
static void CIPSNTPCFG(uint8_t enabled, char timezone);
static void CIPSNTPTIME();
static void CIPCLOSE(uint8_t id);
//*****************************************************************************************************************************************************
static void get_full_IO(uint8_t id);
static void update_IO(char *req, uint8_t id);
static void toggle_IO(char *req, uint8_t id);
//*****************************************************************************************************************************************************
static void admin_connect(char *req, uint8_t id);
static void download_log(char *req, uint8_t id);
static void clear_log(char *req, uint8_t id);
static void reset_esp(char *req, uint8_t id);
static void reset_chip(char *req, uint8_t id);
//*****************************************************************************************************************************************************
static char *decodeURIComponent(const char *str);
static void json_to_io(const char *encoded_json, char *data_buf);
static void json_to_struct8(const char *encoded_json, JSON_STRUCT8 *json_struct, uint8_t get_prop_names);
//*****************************************************************************************************************************************************
static void float_to_string(float value, char *str);
static void send_temperature(uint8_t id);
static float calculate_temperature();
static uint16_t start_ADC_reading();
static uint16_t read_adc(uint8_t ch);
//*****************************************************************************************************************************************************
static void remove_conn(uint8_t id);
static void start_closing(uint16_t close_delay, uint8_t id);
static void timed_close();
//*****************************************************************************************************************************************************
static void format_datetime(char *date_str);
static void get_local_time(char *final_date);
static void log_event(char *ev, const char ev_P[], uint8_t logTime);
```

Παρακάτω ορίζουμε κάποιες **global** μεταβλητές που θα χρησιμοποιούμε. Ο όρος **volatile** όπου θα δείτε, σημαίνει ότι το πρόγραμμα θα πρέπει να διαβάζει την τιμή της μεταβλητής από την κύρια μνήμη και όχι από την προσωρινά αποθηκευμένη (**cached**) τιμή της.

Αυτό, γιατί η τιμή της μεταβλητής μπορεί να αλλάξει σε κάποια ασύγχρονη συνάρτηση π.χ. από μία **interrupt** ενός **timer**. Άρα, επειδή το πρόγραμμα δεν μπορεί να γνωρίζει αν κάτι θα αλλάξει την τιμή της, πρέπει να την ξαναδιαβάσουμε από την κύρια μνήμη, γιατί η **cached** τιμή της μπορεί να είναι μια παλιά τιμή.

Οι τύποι των μεταβλητών μπορεί να διαφέρουν στην ονομασία αλλά είναι ίδιοι με την **C**. Π.χ.:

***uint8_t** = **unsigned char** = μη-προσημασμένος ακέραιος των 8-bit*
***uint16_t** = **unsigned short int** = μη-προσημασμένος ακέραιος των 16-bit*

```
//Σημαία εκκίνησης αναζήτησης HTTP αιτημάτων στον buffer.
volatile uint8_t search = 0;

//Μεταβλητή τύπου TIME_STRUCT για την αποθήκευση ημ/νίας και ώρας. Το ESP θα την αρχικοποιήσει και θα την συγχρονίζει.
TIME_STRUCT cur_time;
volatile uint8_t sync_time = 0; //Σημαία συγχρονισμού ώρας.

//Μεταβλητές για τον Timer1.
volatile uint16_t milliseconds_1 = 0;
volatile uint8_t seconds_1 = 0;
volatile uint8_t timeout_1 = 0;

//Μεταβλητές για τον Timer2.
volatile uint16_t milliseconds_2 = 0;
volatile uint8_t seconds_2 = 0;
volatile uint8_t timeout_2 = 0;
volatile uint8_t delay = 0;

char in_buf[MAX_RESPONSE_STRING_SIZE];          //Buffer γενικής χρήσης.
char *cmd_buf;                                  //Δείκτης (pointer) για γενική χρήση.
COMMAND_STATUS cmd_status;                      //Κατάσταση τελευταίας εντολής που στάλθηκε.
uint8_t timeout_tries = 0;                      //Πόσες επαναλήψεις κάναμε;

//Οι buffers για την αποθήκευση των δεδομένων από την σειριακή σύνδεση.
char usart_in[MAX_RESPONSE_STRING_SIZE] = {0};
char usart_in_extra[MAX_EXTRA_BUFFER_SIZE] = {0};

//Μεταβλητές τύπου USART_BUFFER για την διαχείριση της σειριακής μετάδοσης.
USART_BUFFER main_buf;
USART_BUFFER extra_buf;                                         //Βοηθητικός buffer για την λήψη ακέραιων αιτημάτων όταν γεμίζει ο κύριος buffer.

HTTP_CONNECTION connections[ESP_MAX_CONNECTIONS];               //Πίνακας τύπου HTTP_CONNECTION για τις εισερχόμενες HTTP συνδέσεις.
volatile uint8_t conn_count = 0;                                //Μετρητής συνδέσεων.
volatile uint16_t conn_timeout_list[ESP_MAX_CONNECTIONS][2] = {{0}};    //Μετρητές για την διαδικασία τερματισμού συνδέσεων.
                                                                //[X][0] = ορισμός αριθμού milliseconds πριν τον τερματισμό.
                                                                //[X][1] = milliseconds που παρήλθαν.

volatile uint8_t busy = 0;                                      //Σημαία "busy" για χρήση κατά την λήψη αιτημάτων.

char EEMEM buffer[1000];                                        //Buffer για την αποθήκευση και καταγραφή γεγονότων (logger).
volatile uint16_t buf_pos = 0;                                  //Επόμενη ελεύθερη θέση στον logger.
volatile uint8_t logger_used = 0;                               //Σημαία για την αποφυγή ταυτόχρονης καταγραφής πολλαπλών γεγονότων.
```

8.4 Ανάλυση Συνάρτησης Main()

```
void main()
{
    log_event(0, EVT_BOOTING, 0); //Καταγραφή γεγονότος εκκίνησης.
    wdt_disable(); //Απενεργοποίηση του watchdog timer.

    uint32_t UBR = F_CPU/BAUD_RATE - 16; //UBR Value Formula => F_CPU / (BAUD_RATE * 16) - 1
    UBR /= 16;

    //Κλήση συναρτήσεων αρχικοποίησης.
    init();
    init_ADC();
    init_timer();
    USART_init(UBR);

    //Αναμονή για αρχικοποίηση του ESP.
    wait_esp();

    //Αν η αρχικοποίηση αποτύχει, σταμάτα το πρόγραμμα.
    if(init_esp() != 1)
        return;

    log_event(0, EVT_SERVER_UP, 1); //Καταγραφή γεγονότος εκκίνησης server και συγχρονισμός ρολογιού.
    reset_timer1(); //Εκκίνηση timer1.

    while(1)
    {
        //Κλήση κύριας συνάρτησης αναμονής και διαχείρισης συνδέσεων.
        listen();
    }
}
```

Η ροή του προγράμματος είναι η εξής:

1. Αρχικοποίηση περιφερειακών μικροελεγκτή **(timers, USART, IO Ports, ADC)**.
2. Αναμονή για αρχικοποίηση **ESP**.
3. Αρχικοποίηση **ESP**.
4. Συγχρονισμός και εκκίνηση ρολογιού **(Timer1)**.
5. Είσοδος σε ατέρμων βρόγχο.
6. Αναμονή για **HTTP** συνδέσεις.
7. Διαχείριση **HTTP** συνδέσεων.
8. Επιστροφή στο **Βήμα 6**.

Όπως παρατηρείτε, το πρόγραμμα μας δεν έχει τέλος. Θα αναμένει και θα απαντάει σε HTTP συνδέσεις μέχρι να αποσυνδέσουμε την πλακέτα από το ρεύμα.

Στην συνέχεια θα αναφερθούμε στο καθένα βήμα ξεχωριστά και στις συναρτήσεις που χρησιμοποιούμε, μία προς μία, αναλύοντας τις όσο πρέπει.

8.5 Αρχικοποίηση I/O Ports

Στην πλακέτα που χρησιμοποιούμε για το σύστημα μας, έχουμε διαθέσιμα δύο LED για δοκιμές. Ο χρήστης στην δικιά του πλακέτα μπορεί να μην έχει κανένα ή διαφορετικά περιφερειακά. Σε κάθε περίπτωση, αν θέλουμε να τα ελέγχουμε θα πρέπει να ορίσουμε τους ακροδέκτες στους οποίους είναι συνδεδεμένα, σε λειτουργία Εξόδου (Output).

Γενικά, οι ακροδέκτες ενός μικροελεγκτή Atmel μπορούν να είναι είτε σε λειτουργία Εισόδου (Input), είτε σε λειτουργία Εξόδου (Output).

Στην λειτουργία Εισόδου, ο ακροδέκτης χρησιμοποιείται για να λαμβάνει σήματα (είσοδος). Άρα, μπορούμε να λαμβάνουμε σήματα από εξωτερικά κυκλώματα π.χ. μια γεννήτρια παλμών και να πράττουμε ανάλογα. Αντιθέτως, στην λειτουργία Εξόδου, ο ακροδέκτης χρησιμοποιείται για να στέλνει σήματα, (έξοδος). Άρα, μπορούμε να στέλνουμε σήματα για να ανοίξουμε ένα Ρελέ, ή να ανάψουμε ένα LED.

Αυτή η λειτουργία ρυθμίζεται στον καταχωρητή DDR (Data Direction Register). Για κάθε πόρτα, υπάρχει και ο αντίστοιχος καταχωρητής DDR π.χ. DDRA, DDRB, DDRC κλπ.

Για να αλλάξουμε την λογική κατάσταση των ακροδεκτών χρησιμοποιούμε τον καταχωρητή PORT. Όπως και προηγουμένως, για κάθε πόρτα, υπάρχει και ο αντίστοιχος καταχωρητής PORT π.χ. PORTA, PORTB, PORTC κλπ.

Για να διαβάσουμε την λογική κατάσταση των ακροδεκτών χρησιμοποιούμε τον καταχωρητή PIN. Όπως και προηγουμένως, για κάθε πόρτα, υπάρχει και ο αντίστοιχος καταχωρητής PIN π.χ. PINA, PINB, PINC κλπ.

```
void init()
{
    DDRC = 0x30; //Αρχικοποίηση των LED ως Έξοδο.
    PORTC = 0; //Απενεργοποίηση και των δύο.

    uint8_t i, j;

    //Μηδενισμός πίνακα συνδέσεων.
    for(i = 0; i < ESP_MAX_CONNECTIONS; i++)
    {
        for(j = 0; j < ESP_CONNECTION_MAX_REQUESTS; j++)
            connections[i].request[j] = 0;

        connections[i].total_reqs = 0;
        connections[i].closing = 0;
        connections[i].closed = 0;
    }
    return;
}
```

Στην παραπάνω συνάρτηση, όπως βλέπετε, αρχικοποιούμε τους ακροδέκτες με τα περιφερειακά μας σε λειτουργία Εξόδου και στην συνέχεια τα απενεργοποιούμε. Έπειτα, αρχικοποιούμε/μηδενίζουμε τον πίνακα με τις HTTP συνδέσεις, ώστε να μην περιέχουν «σκουπίδια» (garbage, τυχαίες τιμές).

8.6 Αρχικοποίηση Analog-Digital Converter (ADC)

Πρώτα, αρχικοποιούμε τον **Analog-Digital Converter (ADC),** όπου μέσω αυτού θα διαβάζουμε την θερμοκρασία από τον θερμίστορ.

```
void init_ADC()
{
    ADMUX |= (1 << REFS0); //Τάση αναφοράς = VCC
    ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0); //Prescaler = 128
}
```

Για την αρχικοποίηση του ADC, καλό είναι ο χρήστης να συμβουλευτεί το εγχειρίδιο χρήσης για τον μικροελεγκτή που διαθέτει.

Ο καταχωρητής **ADMUX** χρησιμοποιείται για να ορίσουμε την πηγή της τάσης αναφοράς του (VREF). Υπάρχουν δύο περιπτώσεις:

1. Με εξωτερική παροχή ρεύματος.
2. Με την υπάρχουσα παροχή ρεύματος **(VCC)**.

Εμείς χρησιμοποιούμε την υπάρχουσα τάση **(VCC)**.

Θα παρατηρήσετε κάποιους περίεργους τελεστές όπως αυτόν **"<<"**. Αυτός είναι ο δυαδικός τελεστής για την αριστερή μετατόπιση **(Left Shift)**.

*Η πράξη **X << N**, μετατοπίζει τον αριθμό **X** κατά **N** bit αριστερά.*

Έστω *X = 00000001 :*

X << 1 => ***X = 0b00000010***
X << 2 => ***X = 0b00000100*** *….*

Η ίδια λογική ισχύει και για την δεξιά μετατόπιση "**>>**".

Ο επόμενος τελεστής που παρατηρούμε είναι ο λογικός τελεστής **OR (|)**.
Δεν χρειάζεται περαιτέρω εξήγηση.

Σε συνδυασμό με τον τελεστή εκχώρησης "**=**" μπορούμε να εξοικονομήσουμε χώρο.
Z |= (X << 1) *ισοδυναμεί με* ***Z = Z | (X << 1)***

Η ίδια λογική ισχύει και με τους αριθμητικούς τελεστές, π.χ.:
X += 1 *ισοδυναμεί με* ***X = X + 1***

Ουσιαστικά, με αυτούς τους τελεστές μπορούμε να ενεργοποιούμε συγκεκριμένα **bits** των καταχωρητών. Οι διάφοροι καταχωρητές που θα χρησιμοποιούμε είναι ήδη φορτωμένοι, μέσω των εσωτερικών βιβλιοθηκών που αναφέραμε προηγουμένως.

Θα αναφέρουμε ένα παράδειγμα με τον καταχωρητή **ADMUX**, για τον **ATmega32**.
Ο καταχωρητής **ADMUX** αποτελείται ως εξής:

Πίνακας 8.6.1. Καταχωρητής ADMUX.

	7 bit	**6 bit**	**5 bit**	**4 bit**	**3 bit**	**2 bit**	**1 bit**	**0 bit**
ADMUX	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0

Σημείωση: η αρίθμηση των bit ξεκινάει από το 0 (πρώτο bit).

Ενεργοποιώντας το **6ο bit (REFS0**), ο **ADC** θα χρησιμοποιεί ως τάση αναφοράς την υπάρχουσα τάση **VCC.**

Για να μην μπερδευόμαστε με **αριθμούς bit**, η Atmel μας προσφέρει μέσω των βιβλιοθηκών της, έτοιμες σταθερές με ονόματα που είναι συναφή των καταχωρητών τους. Ουσιαστικά είναι σταθερές για αριθμούς.

Αν κοιτάξουμε μέσα στις βιβλιοθήκες την σταθερά **REFS0**, θα δούμε αυτό:

```
#define REFS0 6
```

Άρα για να το ενεργοποιήσουμε:

```
ADMUX |= (1 << REFS0)
```

1. *Μετακίνησε τον αριθμό **1** κατά **REFS0 (6)** bit αριστερά.*
2. *Με το αποτέλεσμα, εκτέλεσε το λογικό **OR** με τον καταχωρητή **ADMUX**.*
3. *Αποθήκευσε το αποτέλεσμα στον καταχωρητή **ADMUX**.*

1. *(1 << REFS0) ===> (0b00000001 << 6) ===> 0b01000000*
2. *ADMUX | 0b00100000*
3. *ADMUX = ADMUX | 0b01000000 ===> 0bX1XXXXXX*

Στο αποτέλεσμα μας, μετράει να έχουμε ενεργοποιήσει το **6ο bit**. Τα υπόλοιπα δεν μας ενδιαφέρουν σε τι κατάσταση είναι. Η ίδια λογική ενεργοποίησης bit χρησιμοποιείται σε όλο τον κώδικα του συστήματος μας.

Ο καταχωρητής **ADCSRA** χρησιμοποιείται για να ενεργοποιήσουμε τον **ADC** (**ADEN**) και να του ορίσουμε με ποια συχνότητα θα κάνει μετρήσεις (**ADPS2**, **ADPS1**, **ADPS0**).

Πίνακας 8.6.2. Καταχωρητής ADCSRA.

	7 bit	**6 bit**	**5 bit**	**4 bit**	**3 bit**	**2 bit**	**1 bit**	**0 bit**
ADCSRA	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0

Το έργο αυτό αναλαμβάνει ο **prescaler**, όπου δέχεται συγκεκριμένες τιμές: **2, 4, 8, 16, 32, 64,** και **128**. Αυτοί οι αριθμοί στην ουσία είναι διαιρέτες. Με την τιμή που θα του ορίσουμε, διαιρεί την συχνότητα του μικροελεγκτή μας **(F_CPU)**. Το αποτέλεσμα είναι η συχνότητα των μετρήσεων.

Στη συνέχεια, παρουσιάζεται πίνακας με τους συνδυασμούς των **ADPS** bit με τις αντίστοιχες τιμές τους και ένα παράδειγμα εφαρμογής.

Πίνακας 8.6.3. Επιλογές ADC Prescaler.

ADPS2	ADPS1	ADPS0	Συντελεστής Διαίρεσης
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

F_CPU = 12 MHz, Prescaler = 64
ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) ***=>*** *ADCSRA = 0b10000110*

Άρα συχνότητα μετρήσεων = F_CPU / 64 ===> 12000000 / 64 = ***187,5 KHz***

8.7 Αρχικοποίηση Timer

```
void init_timer()
{
    double sec = 0.001;                 //Ο χρόνος που επιθυμούμε να μετράμε ανά interrupt (1ms).

    //Timer 2.

    //Τον χρησιμοποιούμε για να μετράμε την καθυστέρηση μεταξύ εντολής και απάντησης από το ESP.
    TCCR2 |= (1 << WGM21);              //Mode 2, CTC στον OCR2.
    TIMSK |= (1 << OCIE2);              //Ενεργοποίηση Clear-on-Compare interrupt.
    OCR2 = (F_CPU / 8.0) * sec - 1;     //Υπολογισμός τιμής σύγκρισης.

    //Timer 1.

    //Τον χρησιμοποιούμε ως ρολόι.
    TCCR1B |= (1 << WGM12);             //Mode 4, CTC στον OCR1A.
    TIMSK |= (1 << OCIE1A);             //Ενεργοποίηση Clear-on-Compare interrupt.
    OCR1A = (F_CPU / 8.0) * sec - 1;    //Υπολογισμός τιμής σύγκρισης.

    //Αρχικοποίηση struct ημ/νίας και ώρας.
    cur_time.sec = 0;
    cur_time.min = 0;
    cur_time.hour = 0;
    cur_time.day = 1;
    cur_time.mon = 1;
    cur_time.year = 1970;

    //Ενεργοποίηση ISR interrupt.
    sei();
}
```

Στον **ATmega32**, υπάρχουν τρείς (3) **Timer** που μπορούμε να χρησιμοποιήσουμε. Αυτοί είναι οι:

1. **Timer0** (8-bit)
2. **Timer1** (16-bit)
3. **Timer2** (8-bit)

Στο σύστημα μας κάνουμε χρήση των **Timer1** και **Timer2**. Στην συνέχεια αναφέρουμε με λίγα λόγια πως λειτουργεί ένας **Timer**.

Όλα ξεκινάνε με τον εσωτερικό ρολόι **(Clock Source)** του μικροελεγκτή, δηλαδή την **συχνότητα συγχρονισμού** που ορίσαμε και αναλύσαμε προηγουμένως. Το ρολόι στέλνει παλμούς στον **prescaler** του Timer μας, ο οποίος διαιρεί τους παλμούς με την τιμή prescaler που έχουμε ορίσει μέσω των **CS bits**.

Παρακάτω, παρουσιάζονται οι πίνακες με τις αντίστοιχες ρυθμίσεις prescaler, για τους **Timer1** και **Timer2** αντίστοιχα.

Πίνακας 8.7.1. Ρύθμιση prescaler για τον Timer1.

CS12	CS11	CS10	Περιγραφή
0	0	0	Timer/Counter1 Απενεργοποιημένος
0	0	1	Clock / 1 : Χωρίς prescaling
0	1	0	Clock / 8
0	1	1	Clock / 64
1	0	0	Clock / 256
1	0	1	Clock / 1024
1	1	0	Εξωτερικός παλμός στο T1 pin. Ρολόι στην Πτώση.
1	1	1	Εξωτερικός παλμός στο T1 pin. Ρολόι στην Άνοδο.

Πίνακας 8.7.2. Ρύθμιση prescaler για τον Timer2.

CS22	CS21	CS20	Περιγραφή
0	0	0	Timer/Counter2 Απενεργοποιημένος
0	0	1	Clock / 1 : Χωρίς prescaling
0	1	0	Clock / 8
0	1	1	Clock / 32
1	0	0	Clock / 64
1	0	1	Clock / 128
1	1	0	Clock / 256
1	1	1	Clock / 1024

Το αποτέλεσμα στέλνεται στο εσωτερικό κύκλωμα του Timer, όπου εκεί αυξάνει κατά ένα (1) τον καταχωρητή **TCNTn** (όπου **n** ο αριθμός του Timer που χρησιμοποιούμε). Όταν ο καταχωρητής φτάσει στην μέγιστη τιμή του (**255** για 8-bit Timers και **65535** για 16-bit Timers), μηδενίζεται αυτόματα και στέλνει ένα σήμα **TOVn** (**timer overflow**). Με το σήμα αυτό μπορούμε να εκτελέσουμε μια **Ρουτίνα Διακοπής (Interrupt Routine)**. Αυτή είναι η **Κανονική Λειτουργία** του **Timer**.
Οι **Timers** μπορούν να δουλέψουν σε δύο (2) **Λειτουργίες** (**Mode**):

1. **Normal Mode** (Κανονική Λειτουργία)
2. **CTC Mode** (Clear Timer on Compare / Μηδενισμός Timer σε σύγκριση)

Το πρόβλημα με την **Κανονική Λειτουργία**, είναι ότι είναι πολύ δύσκολο στην χρήση συγκεκριμένων τιμών χρόνου. Γι' αυτό και χρησιμοποιείται, συνήθως, όταν δεν μας ενδιαφέρει ένας συγκεκριμένος χρόνος π.χ. σε 1 ms ή σε 2 ms, αρκεί να γίνεται στον ίδιο χρόνο συνέχεια.

Στο **CTC Mode**, όταν ο καταχωρητής **TCNTn** αυξάνεται κατά ένα (1), η τιμή του συγκρίνεται με την τιμή ενός άλλου καταχωρητή, του **OCRn**. Αν οι τιμές είναι ίδιες, τότε μηδενίζεται ο **TCNTn** και στέλνει το σήμα **TOVn (timer overflow)**.

Το ερώτημα είναι **ποια τιμή** θα βάλουμε στον καταχωρητή **OCRn**, για να εκτελείται μία διακοπή, στο χρόνο που επιθυμούμε;

Δυστυχώς, επειδή ο Timer δεν επεξεργάζεται τον χρόνο σε ώρες, λεπτά ή δευτερόλεπτα, χρησιμοποιούμε λίγα μαθηματικά και τον prescaler για να λύσουμε το πρόβλημα.

*OCRn = [(F_CPU / τιμή_prescaler) * επιθυμητός_χρόνος_σε_δευτερόλεπτα] - 1*

Η τιμή **OCRn** πρέπει να είναι ακέραιος αριθμός. Αν καταλήξουμε σε δεκαδική τιμή, τότε σημαίνει ότι θα έχουμε μια ανακρίβεια στον Timer μας. Επίσης, μην ξεχνάμε ότι οι τιμές που μπορούμε να βάλουμε είναι από **0 έως 255 (8-bit Timers)** και από **0 έως 65535 (16-bit Timers)**.

Π.χ.: θέλουμε, με την χρήση του Timer1**,** να εκτελούμε μία Διακοπή για να αναβοσβήνουμε ένα LED, κάθε **1 δευτερόλεπτο**.

F_CPU = 12MHz
*Prescaler = 256 (Χρησιμοποιούμε κατάλληλη τιμή ώστε **OCR1 <= 65535**)*

*OCR1 = [(12000000 / 256) * 1] – 1 = 46875 – 1 = **46874***

Στο δικός μας σύστημα επιθυμούμε **1ms** περίοδο και στους δύο Timer. Άρα, έχουμε **F_CPU = 4MHz** και **prescaler = 8**, για να έχουμε ακέραιο αποτέλεσμα.

*OCRn = [(4000000 / 8) * 0.001] – 1 = 500 – 1 = **499***

Τέλος, για να ενεργοποιήσουμε τις **Ρουτίνες Διακοπής (Interrupt Routines)**, πρέπει να καλέσουμε την συνάρτηση **sei()**. Όπως παρατηρήσατε, δεν βάζουμε τιμή **prescaler** στους **Timers** μας ακόμα, γιατί θα τους ενεργοποιούσε. Η ενεργοποίηση τους θα γίνει αργότερα.

8.8 Αρχικοποίηση USART (Σειριακή Επικοινωνία)

```
void USART_init(uint32_t ubrr_value)
{
    //Ορισμός Baud Rate (ταχύτητα).
    //Αν η τιμή είναι μεγαλύτερη από 255, σπάμε τα bytes σε δύο μέρη.
    UBRRH = (unsigned char)(ubrr_value >> 8);
    UBRRL = (unsigned char)ubrr_value;

    //Ενεργοποίηση Δέκτη (Receiver), Μεταδότη (Transmitter) και Διακοπή Ρουτίνας Δέκτη (Receiver Interrupt).
    UCSRB = (1 << RXEN) | (1 << TXEN) | (1 << RXCIE);

    //Ορισμός λοιπών ρυθμίσεων.
    /*
        >> Χωρίς Parity bits
        >> No Parity
        >> 1 StopBit
        >> Μέγεθος πληροφορίας 8 bit
    */
    UCSRC = (1 << URSEL) | (3 << UCSZ0);

    //Απενεργοποίηση Διακοπών.
    cli();

    //Αρχικοποίηση struct σειριακής μετάδοσης.
    main_buf.buffer = usart_in;
    extra_buf.buffer = usart_in_extra;

    main_buf.index = 0;
    main_buf.buf_size = sizeof(main_buf.buffer);
    main_buf.last_cmd_start = 0;
    main_buf.last_cmd_end = 0;
    main_buf.last_req_end = 0;
    main_buf.reset = 0;

    extra_buf.index = 0;
    extra_buf.buf_size = sizeof(extra_buf.buffer);
    extra_buf.last_cmd_start = 0;
    extra_buf.last_cmd_end = 0;
    extra_buf.last_req_end = 0;
    extra_buf.reset = 0;

    //Ενεργοποίηση Διακοπών.
    sei();

   return;
}
```

Τώρα θα ρυθμίσουμε την **Σειριακή Επικοινωνία**. Για να ξεκινήσουμε, θα πρέπει να ορίσουμε στον μικροελεγκτή με τι ρυθμό μετάδοσης/ταχύτητα θα επικοινωνεί **(BAUD RATE)**.

Ο μικροελεγκτής χρησιμοποιεί ένα ρολόι ειδικά σχεδιασμένο για την σειριακή επικοινωνία, το λεγόμενο **UART Baud Rate Generator (UBRG)**. Για να επιτύχει τον ρυθμό μετάδοσης που επιθυμούμε, χρησιμοποιεί τις εξισώσεις στον πίνακα που ακολουθεί.

Πίνακας 8.7.3. Εξισώσεις για υπολογισμό Baud Rate και τιμής UBRR.

Είδος Λειτουργίας	Εξίσωση για υπολογισμό Baud Rate	Εξίσωση για υπολογισμό τιμής UBRR
Asynchronous Normal Mode (U2X = 0)	$BAUD = \frac{f_{osc}}{\mathbf{16}(\boldsymbol{UBRR} + \mathbf{1})}$	$\boldsymbol{UBRR} = \frac{f_{osc}}{16\,BAUD} - 1$
Asynchronous Double Speed Mode (U2X = 1)	$BAUD = \frac{f_{osc}}{\mathbf{8}(\boldsymbol{UBRR} + \mathbf{1})}$	$\boldsymbol{UBRR} = \frac{f_{osc}}{8\,BAUD} - 1$
Synchronous Master Mode	$BAUD = \frac{f_{osc}}{\mathbf{2}(\boldsymbol{UBRR} + \mathbf{1})}$	$\boldsymbol{UBRR} = \frac{f_{osc}}{2\,BAUD} - 1$

Θα παρατηρήσετε τον καταχωρητή **UBRR (UART Baud Rate Register)**. Ο καταχωρητής αυτός αποτελείται από άλλους δύο (2) 8-bit καταχωρητές, τους **UBRRH (UBRR High)** και **UBRRL (UBRR Low).**

Με βάση την τιμή του 16-bit(2 * 8-bit) καταχωρητή **UBBR**, το ρολόι **UBRG** πετυχαίνει τον ρυθμό μετάδοσης που επιθυμούμε, χρησιμοποιώντας τις εξισώσεις του **Πίνακα 8.7.3**. Επειδή αυτή η τιμή μπορεί να είναι μεγαλύτερη από 255, χρησιμοποιούμε τους καταχωρητές **UBRRH** και **UBRRL** για να την αποθηκεύσουμε.

Την τιμή του **UBRR** πρέπει να την ορίσουμε εμείς, γι' αυτό πρέπει να λύσουμε μόνοι μας την κατάλληλη εξίσωση. Άρα, εφόσον χρησιμοποιούμε **Ασύγχρονη Κανονική Λειτουργία (Asynchronous Normal Mode)**, έχουμε:

*UBRR = F_CPU / (16 * BAUD_RATE) – 1*

Στο σύστημα μας, θα χρησιμοποιούμε **Baud Rate = 9600 bps** (bits per second).

F_CPU = 4 MHz
Baud Rate = 9600 bps

*UBRR = 4000000 / (16 * 9600) – 1 = 25,0416666666666667 = 25 (ακέραια τιμή)*

Πρέπει να επιλέξουμε κατάλληλες τιμές Baud Rate ώστε η τιμή UBRR να βγαίνει όσο ακέραια γίνεται. Αν βγει δεκαδική τιμή, σημαίνει ότι το ρολόι μας δεν θα είναι ακριβές. Γι' αυτό, όπως είχαμε εξηγήσει προηγουμένως, θα έχουμε σφάλμα **0.2%** (βλέπε ***Πίνακας 8.2.1***).

Αφού ρυθμίσαμε τον ρυθμό μετάδοσης, ας πούμε λίγα λόγια για το **Είδος Λειτουργίας (Operation Mode)**. Υπάρχουν τρία (3) κύρια είδη λειτουργίας:

1. **Ασύγχρονη Κανονική Λειτουργία** (Asynchronous Normal Mode)
2. **Ασύγχρονη Διπλής Ταχύτητας Λειτουργία** (Asynchronous Double Speed Mode)
3. **Σύγχρονη Λειτουργία** (Synchronous Master Mode)

Τις διαφορές μεταξύ Σύγχρονης και Ασύγχρονης Λειτουργίας, τις αναφέραμε στο Κεφάλαιο 4.4.

Με λίγα λόγια η Ασύγχρονη Λειτουργία δεν χρησιμοποιεί εξωτερικό ρολόι για συγχρονισμό, σε αντίθεση με την Σύγχρονη Λειτουργία. Εμείς θα χρησιμοποιήσουμε την **Ασύγχρονη Κανονική Λειτουργία**.

Η **Ασύγχρονη Διπλής Ταχύτητας Λειτουργία**, είναι μία παραλλαγή της Ασύγχρονης Κανονικής Λειτουργίας που προσφέρει ο ίδιος ο μικροελεγκτής, για τις ανάγκες του χρήστη. Η μόνη διαφορά όπως λέει και το όνομα, είναι ότι χρησιμοποιεί την διπλή ταχύτητα μειώνοντας τον εσωτερικό διαιρέτη από 16 σε 8, που έχει ως αποτέλεσμα να μειώνεται η δειγματοληψία, άρα και η ακρίβεια στην λήψη δεδομένων. Για περισσότερες πληροφορίες, ο χρήστης μπορεί να ανατρέξει στο εγχειρίδιο του ATmega32.

Οι επόμενες ρυθμίσεις στον κώδικα είναι για τον έλεγχο σφαλμάτων κατά την μετάδοση **(Parity Bits)**, τον αριθμό των bits που σημάνουν την λήξη μετάδοσης **(Stop Bits)** και το μέγεθος της πληροφορίας **(Data Bits)**.

Τέλος, αρχικοποιούμε τις δομές (structs) που θα χρησιμοποιήσουμε για να αποθηκεύουμε και χειριζόμαστε τα δεδομένα που λαμβάνουμε, μέσω της σειριακής σύνδεσης.

8.9 Αρχικοποίηση ESP (Wi-Fi Module)

```
void wait_esp()
{
    delay = 5; //Μέγιστος χρόνος που επιθυμούμε να περιμένουμε.
    reset_timer2(); //Αρχικοποίηση και ενεργοποίηση Timer2.

    char *start;

    //Περίμενε μέχρι να τελειώσει ο χρόνος ή να συνδεθεί το ESP στο τοπικό δίκτυο.
    while(!timeout_2)
    {
        //Αναζήτηση σήματος σύνδεσης στους buffer.
        if((start = strstr_P(main_buf.buffer + main_buf.last_cmd_end, AT_CWJAP_DEF_OK)) ||
           (start = strstr_P(extra_buf.buffer + extra_buf.last_cmd_end, AT_CWJAP_DEF_OK)))
        {
            start[strlen_P(AT_CWJAP_DEF_OK) - 1] = 0;
            break;
        }
    }
    delay = 0;
    timeout_2 = 0;

    return;
}
```

```
char init_esp()
{
    log_event(0, EVT_INIT_ESP, 0); //Καταγραφή γεγονότος αρχικοποίησης ESP.

    strcpy_P(in_buf, ATE0);
    send_command(in_buf, 1, ESP_DEFAULT_ANSWER, 1, 1, 0); //Απενεργοποίηση ηχούς.

    strcpy_P(in_buf, AT_CWMODE_DEF_FILLED);
    send_command(in_buf, 2, ESP_DEFAULT_ANSWER, 1, 1, 0); //Λειτουργία του ESP ως Server (Access Point) και Client (Station) ταυτόχρονα.

    strcpy_P(in_buf, AT_CIFSR);
    send_command(in_buf, 2, ESP_DEFAULT_ANSWER, 1, 1, 0); //Εμφάνιση IP διευθύνσεων του ESP.

    //Έλεγχος IP διεύθυνσης.
    if(cmd_status == CMD_SUCCESS)
    {
        char ip[16] = {0};

        strcpy_P(ip, ESP_IP);
        sprintf_P(in_buf, AT_CIFSR_STATION_IP_FORMAT, ip);

        //Αν είναι διαφορετική από αυτήν που θέλουμε, διόρθωσε την.
        if(!strstr(main_buf.buffer + main_buf.last_cmd_start, in_buf) &&
           !strstr(extra_buf.buffer + extra_buf.last_cmd_start, in_buf))
        {
            //Σύνδεση στο τοπικό δίκτυο μας.
            CWJAP_DEF(WIFI_STATION, WIFI_KEY);
            _delay_ms(1000);

            //Αν η σύνδεση απέτυχε, επέστρεψε 0.
            if(cmd_status != CMD_SUCCESS)
                return 0;
        }
    }
    else
        return 0;

    strcpy_P(in_buf, AT_CIPMUX_FILLED);
    send_command(in_buf, 2, ESP_DEFAULT_ANSWER, 1, 1, 0); //Ρύθμιση για πολλαπλές συνδέσεις ταυτόχρονα.

    //todo: catch "no change" answer.
    strcpy_P(in_buf, AT_CIPSERVER_FILLED);
    send_command(in_buf, 2, ESP_DEFAULT_ANSWER, 1, 1, 0); //Ενεργοποίηση Server στην πόρτα 80.

    //Ενεργοποίηση SNTP και ορισμός ζώνης ώρας UTC + 2.
    CIPSNTPCFG(1, 2);
    _delay_ms(4000); //Απαραίτητη καθυστέρηση. Ο απαραίτητος χρόνος καθυστέρησης δεν είναι πάντα ο ίδιος.

    sync_time = 1; //Συγχρονισμός ρολογιού στην επόμενη καταγραφή γεγονότος.
    search = 1; //Ξεκίνα να ελέγχεις για HTTP συνδέσεις.

    return 1;
}
```

Αφού ρυθμίσαμε την **Σειριακή Επικοινωνία**, είμαστε έτοιμοι να ρυθμίσουμε το **ESP**.

Οι βοηθητικές συναρτήσεις που χρησιμοποιούμε για να στείλουμε δεδομένα αναλύονται αργότερα. Προς το παρόν, γίνεται ανάλυση στην ροή της συνάρτησης.

Πριν, όμως καλέσουμε την συνάρτηση για να αρχικοποιήσουμε το ESP, καλούμε πρώτα την συνάρτηση **"wait_esp()".** Αυτή η συνάρτηση είναι βοηθητική, με την έννοια ότι αντί να περιμένουμε πλήρως ένα ορισμένο χρονικό διάστημα πριν αρχικοποιήσουμε το ESP, εφόσον το ίδιο το ESP στέλνει σήμα όταν συνδεθεί στο τοπικό δίκτυο, μπορούμε απλώς να περιμένουμε για το σήμα αυτό. Έτσι, μειώνουμε σημαντικά τον χρόνο μεταξύ της εκκίνησης του συστήματος και την λήψη και διαχείριση HTTP συνδέσεων.

Μόλις λάβουμε το κατάλληλο σήμα ή τελειώσει ο μέγιστος χρόνος καθυστέρησης, καλούμε την συνάρτηση για την αρχικοποίηση του ESP.

Πρώτα, απενεργοποιούμε την **ηχώ (echo)** του ESP, που είχαμε αναφέρει σε προηγούμενο κεφάλαιο. Στη συνέχεια, ρυθμίζουμε το **Είδος Λειτουργίας (mode)** του ESP, ώστε να δουλεύει σαν **Server (Access Point)** και **Client (Station)** ταυτόχρονα.

Ύστερα, ελέγχουμε αν η διεύθυνση του ESP, ως Client, είναι αυτή που θέλουμε. **Η IP αυτή ορίζεται στην βοηθητική βιβλιοθήκη "settings.h" που παρουσιάζεται στο τέλος της ανάλυσης του συστήματος μας.** Αν δεν είναι αυτή που θέλουμε, την ορίζουμε, επιτόπου, στον ESP και συνεχίζουμε. Επόμενο, είναι να ρυθμίσουμε τον server να δέχεται πολλαπλές συνδέσεις. Τέλος, γίνεται ενεργοποίηση του server στην **πόρτα 80.**

Οι επόμενες ρυθμίσεις είναι για την ενεργοποίηση και ρύθμιση της ώρας στον ESP. Αν ο χρήστης στο δικό του σύστημα δεν επιθυμεί την χρήση ώρας, μπορεί να παραλείψει αυτήν την ρύθμιση. **(Η έκδοση λογισμικού (firmware) του ESP, που διαθέτει ο χρήστης, πρέπει να είναι v2.1.0, ή μεγαλύτερη, ώστε να υποστηρίζει τις εντολές για SNTP πρωτόκολλο).**

Τελευταία εντολή πριν επιστρέψουμε στο κυρίως πρόγραμμα, είναι η ενεργοποίηση της σημαίας **"search"**. Η χρήση αυτής της σημαίας γίνεται γιατί δεν θέλουμε, την ώρα που ρυθμίζουμε το ESP, να δεχόμαστε HTTP συνδέσεις.

8.10 Διαχείριση HTTP συνδέσεων και ανάλυση βοηθητικών συναρτήσεων

Αφού τελειώσαμε με όλες τις αρχικοποιήσεις, συνεχίζουμε στην ατέρμων λούπα του προγράμματος.

```
while(1)
{
    //Κλήση κύριας συνάρτησης αναμονής και διαχείρισης συνδέσεων.
    listen();
}
```

Στην συνάρτηση **listen()**, το πρόγραμμα μας εισέρχεται σε έναν νέο ατέρμων βρόγχο. Κατά την διάρκεια του βρόγχου αυτού, το πρόγραμμα μας, κάνει διάφορους ελέγχους. Ένας από αυτούς, είναι ο έλεγχος για νέες HTTP συνδέσεις. Κάθε πελάτης (client) που συνδέεται στο ESP, αποκτά ένα μοναδικό **αναγνωριστικό (ID)**.

Ο μέγιστος αριθμός πελατών που μπορούν να είναι ταυτόχρονα συνδεδεμένοι στο ESP είναι πέντε (5).

Το εύρος τιμών των ID είναι: [0, 4]

Όπως καταλαβαίνετε, το ESP δεν μπορεί να έχει τον ρόλο ενός Web Server, όπως αυτών που χρησιμοποιούμε καθημερινά. Οι χρήση του είναι περιορισμένη αλλά αυτό δεν σημαίνει ότι δεν έχει και καθόλου. Το σύστημα μας δεν προορίζεται για δημόσια χρήση. Για προσωπικές αλλά και βιομηχανικές εφαρμογές, πέντε (5) συνδέσεις είναι υπερ-αρκετές.

Συνεχίζοντας την ανάλυση μας, αφού λάβει το HTTP αίτημα (HTTP request) του πελάτη, το ESP εκπέμπει σειριακά το ίδιο το αίτημα. Εμείς, λαμβάνουμε αυτό το αίτημα και αφού κάνουμε κάποιους ελέγχους, το αποθηκεύουμε μαζί και με το αναγνωριστικό της σύνδεσης. *Αν ένας πελάτης κάνει περισσότερα από ένα αιτήματα, το ESP θα λάβει με το ίδιο αναγνωριστικό πάντα, αφού πρόκειται για τον ίδιο πελάτη.*

Αφού λάβουμε το αίτημα, το αναλύουμε και αν είναι έγκυρο, στέλνουμε πίσω την κατάλληλη απάντηση (HTTP response) και τέλος κλείνουμε την σύνδεση.

```
void listen()
{
    uint8_t i, j;
    uint8_t total_reqs; //Σύνολο αιτημάτων για μία συγκεκριμένη σύνδεση.

    do
    {
        //Αν πρέπει να συγχρονίσουμε το ρολόι και δεν έχουμε καμία ανοικτή σύνδεση, συγχρόνισε το και κατέγραψε το γεγονός.
        if(sync_time && !conn_count)
            log_event(0, EVT_CLOCK_SYNC, 1);

        //Έλεγχος και διαχείριση HTTP συνδέσεων.
        for(i = 0; i < ESP_MAX_CONNECTIONS; i++)
        {
            total_reqs = connections[i].total_reqs;

            //Όσο έχουμε νέα αιτήματα, διαχειρίσου τα.
            while(total_reqs)
            {
                for(j = 0; j < ESP_CONNECTION_MAX_REQUESTS; j++)
                {
                    if(connections[i].request[j])
                        handle_request(connections[i].request[j], i, j);

                    //Ανανέωση μετρητή σε περίπτωση που ήρθε νέο αίτημα κατα την επεξεργασία ενός άλλου.
                    total_reqs = connections[i].total_reqs;
                }
            }
        }

        //Τερματισμός συνδέσεων που δεν τερματίστηκαν άμεσα.
        timed_close();
    }
    while(1);

    return;
}
```

Η ροής της συνάρτησης είναι η εξής:

1. Συγχρονισμός ρολογιού, αν χρειάζεται.
2. Έλεγχος αριθμού αιτημάτων συνδέσεων.
3. Διαχείριση αιτημάτων, αν υπάρχουν.
4. Τερματισμός συνδέσεων που έκλεισαν άμεσα, αν χρειάζεται.
5. Επιστροφή στο **Βήμα 1**.

Την διαχείριση κάθε αιτήματος την λαμβάνει η συνάρτηση **handle_request()**. Τον τερματισμό συνδέσεων που δεν έκλεισαν άμεσα, την λαμβάνει η συνάρτηση **timed_close()**. Ανάλυση τους, θα γίνει αργότερα. Προς το παρόν, ας αναλύσουμε την διαδικασία λήψης και ελέγχου των αιτημάτων μέσω της σειριακής σύνδεσης.

Πως ξεχωρίζουμε ένα HTTP Request από απλά δεδομένα;

Αν κάνουμε μερικές δοκιμές με το ESP, θα δούμε ότι ένα **HTTP Request** έχει την παρακάτω μορφή:

+IPD,0,399:GET /?TEMP=1 HTTP/1.1
Host: 192.168.1.48
Connection: keep-alive
*Accept: text/plain, */*; q=0.01*
Origin: http://192.168.1.2
User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36
DNT: 1
Content-Type: text/plain
Referer: http://192.168.1.2
Accept-Encoding: gzip, deflate
Accept-Language: el,en;q=0.9

Οι χρήσιμες πληροφορίες για εμάς, βρίσκονται στην πρώτη γραμμή:

+IPD,0,399:GET /?TEMP=1 HTTP/1.1

Η σύνταξη είναι ως εξής:

+IPD, <link ID>, <len> : <request>

link ID = αναγνωριστικό σύνδεσης
len = μέγεθος αιτήματος σε bytes
request = αίτημα

Το αναγνωριστικό της σύνδεσης **(ID)** είναι η πρώτη χρήσιμη πληροφορία. Στο παράδειγμα μας, το **ID = 0**.

Στην δεύτερη γραμμή αναφέρεται η **client IP διεύθυνση** του ESP.

Host: 192.168.1.48

Άρα μπορούμε να θεωρήσουμε ότι όταν μέσα στα δεδομένα που λαμβάνουμε, υπάρχει η φράση **“Host:”**, τότε λάβαμε ένα αίτημα. Η σωστή αναζήτηση θα ήταν για την συμβολοσειρά **“\r\n\r\n”**, δηλαδή δύο (2) κενές γραμμές, όπου είναι το επίσημο τέλος ενός HTTP αιτήματος. Επειδή όμως εμείς χρειαζόμαστε μόνο την πρώτη γραμμή και θέλουμε να εξοικονομούμε χρόνο, επιλέγουμε να “κόψουμε” το αίτημα εκεί που πιστεύουμε ότι είναι αρκετό.

Άρα κάνουμε αναζήτηση στα εισερχόμενα δεδομένα για την φράση «**Host:**». Αν βρεθεί, τότε βρήκαμε μία νέα σύνδεση.

Έτσι λοιπόν, μπορούμε να ξεχωρίζουμε τις **HTTP συνδέσεις** από απλά ή άχρηστα δεδομένα. Στην συνέχεια θα αναλύσουμε την ρουτίνα διακοπής **ISR(USART_RXC_vect)**, με την οποία διαβάζουμε τα σειριακά δεδομένα.

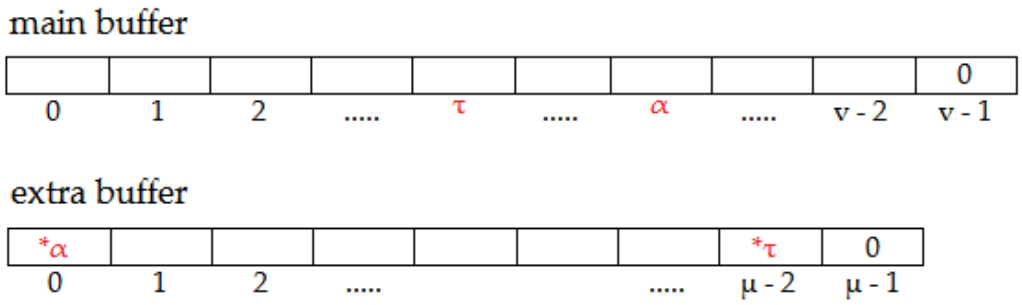
```
ISR(USART_RXC_vect)
{
    uint16_t temp = main_buf.index;

    //Λήψη δεδομένου/χαρακτήρα από τον καταχωρητή UDR.
    main_buf.buffer[main_buf.index++] = UDR;

    //Αντιγραφή χαρακτήρα στον βοηθητικό buffer, όταν πρέπει.
    if(temp >= EXTRA_BUFFER_START || (main_buf.reset && temp <= EXTRA_BUFFER_END))
    {
        extra_buf.buffer[extra_buf.index++] = main_buf.buffer[temp]; //Αντιγραφή.

        //Αναζήτηση αιτημάτων στον βοηθητικό buffer.
        if(search)
            find_request(&extra_buf, extra_buf.last_req_end);

        //Αν φτάσαμε στο όριο, επανεκκίνησε τον βοηθητικό buffer.
        if(extra_buf.index >= MAX_EXTRA_BUFFER_SIZE - 1)
        {
            extra_buf.index = 0;
            extra_buf.last_cmd_end = 0;
            extra_buf.last_req_end = 0;
            extra_buf.reset = 1;
            main_buf.reset = 0;
        }
        extra_buf.buffer[extra_buf.index] = 0; //Μηδενισμός επόμενου byte.
    }

    //Αναζήτηση αιτημάτων στον κύριο buffer.
    if(search)
        find_request(&main_buf, main_buf.last_cmd_end);

    //Αν φτάσαμε στο όριο, επανεκκίνησε τον κύριο buffer.
    if(main_buf.index >= MAX_RESPONSE_STRING_SIZE - 1)
    {
        main_buf.index = 0;
        main_buf.last_cmd_end = 0;
        main_buf.last_req_end = 0;
        main_buf.reset = 1;
        extra_buf.reset = 0;
    }

    //Μηδενισμός επόμενου byte.
    main_buf.buffer[main_buf.index] = 0;
}
```

Η ρουτίνα διακοπής **ISR(USART_RXC_vect)**, εκτελείται μόνο όταν ο **Δέκτης (Receiver)** του μικροελεγκτή λαμβάνει δεδομένα.

Τα δεδομένα που λαμβάνουμε τα αποθηκεύουμε σε **δύο (2) buffer**. Επιλέξαμε δύο γιατί σε περίπτωση που ένα αίτημα έρθει κοντά στο τέλος του buffer, θα κοπεί στα δύο (το μισό στο τέλος και το άλλο μισό στην αρχή του buffer) κι έτσι θα είναι αρκετά δύσκολο, από τεχνικής απόψεως, να χειριστούμε το αίτημα, γιατί δεν θα είναι ενιαίο.

Γι' αυτό χρησιμοποιούμε έναν βοηθητικό buffer που θα είναι περιέχει τα δεδομένα του κύριου buffer από το τέλος και την αρχή του, ενιαία.

Παραθέτουμε μία βοηθητική αναπαράσταση με τα περιεχόμενα των buffer.



Εικόνα 8.10.1. Αναπαράσταση κύριου και βοηθητικού buffer.

Όπως παρατηρείτε, ο βοηθητικός buffer περιέχει δεδομένα του κυρίου buffer ξεκινώντας λίγο πριν από το τέλος του **(α)** και τελειώνοντας με λίγο μετά την αρχή του **(τ)**.

Έτσι, καταφέρνουμε να επεξεργαζόμαστε τα «κομμένα» δεδομένα ως **συμβολοσειρές (strings)**, γι' αυτό το λόγο κιόλας, ο τελευταίος χαρακτήρας και στους δύο buffer είναι ο **τερματικός χαρακτήρας μηδέν (0)**.

Οι τιμές **α** και **τ** αντιστοιχούν στο πρόγραμμα μας, στις σταθερές **EXTRA_BUFFER_START** και **EXTRA_BUFFER_END**. Οι σταθερές αυτές, υπολογίζονται με βάση το μέγιστο μέγεθος που έχουμε ορίσει για τα HTTP αιτήματα **(MAX_URL_SIZE)**.

*Υπενθυμίζουμε ότι οι περισσότερες σταθερές του προγράμματος μας, βρίσκονται στην βοηθητική βιβλιοθήκη **«settings.h»**.*

Σημείωση: ο χρήστης πρέπει να δώσει ιδιαίτερη προσοχή στα μεγέθη των buffer και γενικότερα σε οτιδήποτε είδους buffer, ώστε να υπάρχει πάντα αρκετός χώρος στην SRAM για την διαχείριση και αποθήκευση των HTTP αιτημάτων.

Όπως αναφέραμε και προηγουμένως, δεν θα αναζητάμε το επίσημο τέλος ενός HTTP αιτήματος, αλλά μόνο την πρώτη γραμμή του η οποία έχει και τις χρήσιμες πληροφορίες.

Στο πρόγραμμα μας, η αρχή και το τέλος ενός HTTP αιτήματος υποδηλώνονται από τις συμβολοσειρές **"+IPD" και "Host:"**, αντίστοιχα. Αν βρούμε αυτές τις συμβολοσειρές, τότε μπορούμε να υποθέσουμε ότι λαμβάνουμε ένα HTTP αίτημα. Αφού το λάβουμε, επεξεργαζόμαστε και αποθηκεύουμε τις απαραίτητες πληροφορίες. Αυτές είναι το **ID** της σύνδεσης και το **URL** του αιτήματος.

Παραθέτουμε ένα παράδειγμα ενός HTTP αιτήματος για να γίνει κατανοητή η διαδικασία που αναφέραμε.

+IPD,0,399:GET /?TEMP=1 HTTP/1.1
Host: ~~192.168.1.48~~

~~Connection: keep-alive~~
*~~Accept: text/plain, */*; q=0.01~~*
~~Origin: http://192.168.1.2~~
~~User-Agent: Mozilla/5.0 (Windows NT 6.1; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/74.0.3729.169 Safari/537.36~~
~~DNT: 1~~
~~Content-Type: text/plain~~
~~Referer: http://192.168.1.2~~
~~Accept-Encoding: gzip, deflate~~
~~Accept-Language: el,en;q=0.9~~

Όπως αναφέραμε, δεν κρατάμε όλο το αίτημα παρά μόνο τις χρήσιμες πληροφορίες του.

+IPD,0,399:GET /?TEMP=1 HTTP/1.1
Host:

Ύστερα, αποθηκεύουμε το **ID** της σύνδεσης (“0”) και το **URL** του αιτήματος (“/?TEMP=1”).

Χρησιμοποιώντας την συνάρτηση **find_request()**, αναζητάμε στους buffers μας συγκεκριμένες συμβολοσειρές που να υποδηλώνουν την ύπαρξη ενός HTTP αιτήματος. Μαζί με την συνάρτηση, δίνουμε ως παραμέτρους τον **buffer** στον οποίο θα αναζητήσουμε και την συγκεκριμένη **θέση (offset)** του buffer, από την οποία θα ξεκινήσουμε την αναζήτηση. Το **offset** χρησιμοποιείται γιατί αν ξεκινήσουμε από την αρχή του buffer, υπάρχει μεγάλη πιθανότητα, πριν βρούμε το αίτημα, να συναντήσουμε έναν τερματικό χαρακτήρα, τελειώνοντας έτσι την αναζήτηση πριν βρεθεί το αίτημα. Γι’ αυτό κάνοντας την χρήση των βοηθητικών μεταβλητών της struct **USART_BUFFER** και συγκεκριμένα την μεταβλητή της **last_cmd_end,** ξεκινάμε την αναζήτηση από την τελευταία εντολή ή από το τελευταίο αίτημα που επεξεργαστήκαμε. Κάθε φορά που λαμβάνουμε ένα νέο αίτημα ή στέλνουμε μία εντολή στο ESP, ανανεώνουμε τις βοηθητικές μεταβλητές των buffer ώστε να αποφεύγουμε το παραπάνω πρόβλημα.

Ο χρήστης πρέπει να δώσει ιδιαίτερη προσοχή στην υλοποίηση αυτού του μέρους στο σύστημα του, καθώς είναι πολύ εύκολο να «χάσουμε» αιτήματα λόγω μη σωστής ρύθμισης των βοηθητικών μεταβλητών.

```
void find_request(USART_BUFFER *buf, uint16_t offset)
{
    char *data, *head, *start, *end, *host;

    //Υπολογισμός νέας θέσης με βάση το offset.
    data = buf->buffer + offset;

    //Αναζητάμε το τέλος του αιτήματος.
    host = strstr_P(data, HTTP_REQUEST_HOST);

    //Αν βρέθηκε, συνέχισε.
    if(host)
    {
        //Αναζητάμε την αρχή του αιτήματος.
        head = strstr_P(data, TRANSMISSION_SIGNAL);

        //Αν βρέθηκε, συνέχισε.
        if(head)
        {
            busy = 1; //Μην στέλνεις εντολές στο ESP όσο επεξεργαζόμαστε αιτήματα.

            //Εισάγουμε τερματικό χαρακτήρα στο αίτημα ώστε να μην το ξαναβρούμε.
            *host = 0;

            //Ενημέρωση βοηθητικών μεταβλητών.
            buf->last_req_end = (host - buf->buffer) + 1;
            buf->last_cmd_start = buf->last_req_end;
            buf->last_cmd_end = buf->last_req_end;

            //Υπολογισμός θέσης τέλους αιτήματος στον buffer.
            uint16_t host_start = buf->last_req_end - 1;

            //Αναζήτηση αιτήματος και στον άλλο buffer ωστέ να εισάγουμε και εκεί τον τερματικό χαρακτήρα.

            /*
                Ανάλογα σε ποιό μισό κομμάτι του buffer βρισκόμαστε,
                βρίσκουμε και την αντίστοιχη θέση του αιτήματος στον άλλο buffer.
            */
            if(buf == &main_buf)
            {
                if(host_start >= EXTRA_BUFFER_START)
                {
                    extra_buf.buffer[host_start - EXTRA_BUFFER_START] = 0;
                    extra_buf.last_req_end = (host_start - EXTRA_BUFFER_START) + 1;
                }
                else if(host_start <= HOST_MAX_EXTRA_START)
                {
                    extra_buf.buffer[MAX_EXTRA_BUFFER_SIZE / 2 + host_start] = 0;
                    extra_buf.last_req_end = host_start + 1;
                }
            }
            else if(buf == &extra_buf)
            {
                if(host_start >= MAX_EXTRA_BUFFER_SIZE / 2)
                {
                    main_buf.last_req_end = (host_start - MAX_EXTRA_BUFFER_SIZE / 2) + 1;
                    main_buf.last_cmd_start = main_buf.last_req_end;
                    main_buf.last_cmd_end = main_buf.last_req_end;
                    main_buf.buffer[main_buf.last_req_end - 1] = 0;
                }
                else
                {
                    main_buf.last_req_end = (EXTRA_BUFFER_START + host_start) + 1;
                    main_buf.buffer[main_buf.last_req_end - 1] = 0;
```

```
        //Έλεγχος ορίου.
        if(main_buf.last_req_end >= MAX_RESPONSE_STRING_SIZE - 1)
            main_buf.last_req_end = 0;

        main_buf.last_cmd_start = main_buf.last_req_end;
        main_buf.last_cmd_end = main_buf.last_req_end;
    }
}

uint8_t size;           //Μέγεθος αιτήματος.
char log_str[32] = {0}; //Συμβολοσειρά για καταγραφή γεγονότων.

//Αναζήτηση συμβολοσειράς "GET" για έλεγχο έγκυρου αιτήματος.
start = strstr_P(head, HTTP_GET_REQUEST);

//Αναζήτηση συμβολοσειράς " HTTP" για να βρούμε το τέλος του URL.
end = strstr_P(start, HTTP_SUBSTRING);

uint8_t id = *(head + 5) - '0'; //Μετατροπή ID σε αριθμό.

//Επεξεργαζόμαστε αιτήματα που δεν ανήκουν σε συνδέσεις που βρίσκονται στο στάδιο τερματισμού.
//Τα αιτήματα αυτά θα σταλούν ξανά από το ESP, αφού τερματιστεί η σύνδεση.
if(!connections[id].closing)
{
    //Αν το αίτημα είναι έγκυρο, συνέχισε.
    if(start && end)
    {
        start += strlen_P(HTTP_GET_REQUEST) - 1; //Η αρχή του URL ξεκινάει από το "/".
        size = end - start; //Υπολογισμός μεγέθους URL.

        /*
            Αν το αίτημα είναι η κύρια ιστοσελίδα (index), τότε αύξησε το μέγεθος κατά ένα (1),
            ώστε να αποθηκεύσουμε και τον κενό χαρακτήρα " ".
        */
        if(start[1] == ' ')
            size++;

        //Αν το μέγεθος του URL είναι έγκυρο, συνέχισε.
        if(size <= MAX_URL_SIZE)
        {
            //Αν η σύνδεση δεν μπορεί να δεχτεί άλλα αιτήματα, επέστρεψε πίσω.
            if(connections[id].total_reqs >= ESP_CONNECTION_MAX_REQUESTS)
            {
                sprintf_P(log_str, EVT_OVFLOW_REQ, id);
                log_event(log_str, 0, 1);
                busy = 0;

                return;
            }

            //Μηδενισμός σημαίας τερματισμού.
            connections[id].closed = 0;

            //Ενημέρωση μετρητή αιτημάτων για την σύνδεση.
            uint8_t new = connections[id].total_reqs++;

            //Κάνε χώρο για το αίτημα.
            connections[id].request[new] = calloc(size + 1, sizeof(char));

            //ΑΝ ΔΕΝ ΥΠΑΡΧΕΙ ΑΡΚΕΤΟΣ ΧΩΡΟΣ ΣΤΗΝ SRAM, ΚΛΕΙΣΕ ΤΗΝ ΣΥΝΔΕΣΗ ΚΑΙ ΕΠΕΣΤΡΕΨΕ.
            if(!connections[id].request[new])
            {
```

```
                    log_event(0, SRAM_OVERFLOW, 1);
                    busy = 0;

                    SEND_HTTP_HEADER_507(id); //Αποστολή κατάλληλης απάντησης.
                    CIPCLOSE(id);

                    return;
                }

                //Είναι καινούργια σύνδεση; Αν ναι, τότε αύξησε τον μετρητή συνδέσεων.
                if(connections[id].total_reqs == 1)
                {
                    conn_count++;
                    sprintf_P(log_str, EVT_NEW_CONN, id, new);
                }
                else
                {
                    /*
                        Αλλιώς, η σύνδεση υπάρχει ήδη, μηδένισε τους χρονομετρητές τερματισμού,
                        σε περίπτωση που είχαν ξεκινίσει, ακυρώνοντας έτσι τον τερματισμό της.
                    */
                    conn_timeout_list[id][0] = 0;
                    conn_timeout_list[id][1] = 0;

                    sprintf_P(log_str, EVT_EXIST_CONN, id, new);
                }

                //Αντιγραφή URL αιτήματος.
                strncpy(connections[id].request[new], start, size);

                //Καταγραφή γεγονότος, χωρίς ημ/νία.
                eep_store_string(log_str);
            }
            else
            {
                //Το αίτημα είναι πολύ μεγάλο, τερμάτισε την σύνδεση.
                sprintf_P(log_str, TOO_BIG_REQ, id);
                eep_store_string(log_str);

                CIPCLOSE(id);
            }
        }
        else
        {
            //Το αίτημα δεν είναι έγκυρο, τερμάτισε την σύνδεση.
            sprintf_P(log_str, MALFORMED_REQ, id);
            eep_store_string(log_str);

            CIPCLOSE(id);
        }
    }
}

busy = 0;

return;
}
```

Ουσιαστικά, η συνάρτηση αναζητά στον buffer τις συμβολοσειρές που υποδηλώνουν την ύπαρξη ενός HTTP αιτήματος και αν το βρει, αφού κάνει κάποιος ελέγχους εγκυρότητας, το αποθηκεύει, καταγράφει το γεγονός και επιστρέφει στην ρουτίνα διακοπής.

Η συνάρτηση αυτή πέρασε από πολλά στάδια πριν καταλήξει στην τελική μορφή που παρουσιάζεται εδώ. Ειδικότερα για το σύστημα μας, η συνάρτηση έχει γίνει όσο πιο αποδοτική και γίνεται. Ο χρήστης στο δικό του σύστημα δεν χρειάζεται να έχει την ίδια συνάρτηση με το δικό μας, καθώς οι απαιτήσεις και οι περιορισμοί μπορεί να διαφέρουν.

Στην συνέχεια, θα παρουσιάσουμε και θα αναλύσουμε την συνάρτηση **handle_request()**.

```
void handle_request(char *request, uint8_t id, uint8_t req_index)
{
    uint8_t i;
    memset(in_buf, 0, MAX_RESPONSE_STRING_SIZE); //Μηδενισμός γενικού buffer.

    //Αν λάβαμε αίτημα, συνέχισε.
    if(request)
    {
        //Αναζήτηση αιτήματος στην λίστα με τα προκαθορισμένα αιτήματα μας.
        for(i = 0; i < REQ_LAST; i++)
        {
            //Αν βρέθηκε αίτημα, κατέγραψε το γεγονός και βγές από τον βρόγχο.
            if(strcasestr_P(request, GET_REQUESTS[i]))
            {
                strcpy_P(in_buf + 100, GET_REQUESTS[i]);

                sprintf_P(in_buf, PROCESSING_CONN, id, req_index, in_buf + 100);
                log_event(in_buf, 0, 1);
                break;
            }
        }

        if(i >= REQ_LAST)
        {
            sprintf_P(in_buf, UNKNOWN_REQ, id, req_index);
            log_event(in_buf, 0, 1);
        }

        //Στείλε την κατάλληλη απάντηση ανάλογα το αίτημα.
        switch(i)
        {
            //Αίτημα κύριας ιστοσελίδας index.
            case REQ_INDEX_HTML:
                SEND_INDEX_HTML(INDEX_HTML_CODE, id);
                break;
            //Αίτημα για javascript αρχείο.
            case REQ_CUSTOM_JS:
                SEND_HTTP_FILE(JS_CUSTOM_CODE, id, TEXT_JAVASCRIPT, 0);
```

```
            break;
        //Αίτημα για stylesheet (css) αρχείο.
        case REQ_CSS:
            SEND_HTTP_FILE(STYLE_CSS_CODE, id, TEXT_CSS, 0);
            break;
        //Αίτημα για την κατάσταση όλων των ακροδεκτών του μικροελεγκτή.
        case REQ_FULL_IO:
            get_full_IO(id);
            break;
        //Αίτημα για τιμή θερμοκρασίας.
        case REQ_VAL_TEMP:
            send_temperature(id);
            break;
        //Αίτημα για πλήρης ρύθμιση ακροδεκτών του μικροελεγκτή.
        case REQ_SET_FULL_IO:
            update_IO(request, id);
            break;
        //Αίτημα για ρύθμιση ενός ακροδέκτη του μικροελεγκτή, μόνο.
        case REQ_TOGGLE_IO_PIN:
            toggle_IO(request, id);
            break;
        //Αίτημα για είσοδο διαχειριστή (admin).
        case REQ_ADMIN_CONNECT:
            admin_connect(request, id);
            break;
        //Αίτημα για λήψη αρχείου καταγραφής (log).
        case REQ_DOWNLOAD_LOG:
            download_log(request, id);
            break;
        //Αίτημα για εκκαθάριση αρχείου καταγραφής (log).
        case REQ_CLEAR_LOG:
            clear_log(request, id);
            break;
        //Αίτημα για επανεκκίνηση του ESP.
        case REQ_RESET_ESP:
            reset_esp(request, id);
            break;
        //Αίτημα για επανεκκίνηση του μικροελεγκτή.
        case REQ_RESET_CHIP:
            reset_chip(request, id);
            break;
        default: //Μη έγκυρα αιτήματα.
            SEND_HTTP_HEADER_204(id); //Απάντηση "HTTP 204 No Content".
            break;
    }

    //Αν η σύνδεση δεν έχει τερματιστεί λόγω προβλήματος, συνέχισε.
    if(connections[id].total_reqs && connections[id].request[req_index])
    {
        //Ενημέρωση μετρητή αιτημάτων.
        ATOMIC_BLOCK(ATOMIC_FORCEON)
        {
            connections[id].total_reqs--;

            //Ελευθερώνουμε τον χώρο του αιτήματος.
            free(connections[id].request[req_index]);
            connections[id].request[req_index] = 0; //Μηδενισμός pointer.
        }
    }
```

```
        //Αν αυτό ήταν το τελευταίο αίτημα, τερμάτισε την σύνδεση.
        if(connections[id].total_reqs == 0 && !connections[id].closed)
        {
            //Περίμενε δύο (2) δευτερόλεπτα πριν τερματίσουμε σύνδεση κύριας ιστοσελίδας.
            if(i == REQ_INDEX_HTML)
                start_closing(2000, id);
            else //Αλλιώς, άμεσος τερματισμός.
                CIPCLOSE(id);
        }
    }
    else
    {
        sprintf_P(in_buf, NO_REQ, id, req_index);
        log_event(in_buf, 0, 1);
    }

    return;
}
```

Η συνάρτηση αυτή ελέγχει αν το αίτημα που έλαβε ανήκει σε ένα από τα προκαθορισμένα αιτήματα που έχουμε ορίσει εμείς. Αν βρέθηκε αντιστοιχία τότε στέλνουμε την κατάλληλη απάντηση. Αν όχι, στέλνουμε την απάντηση "**HTTP 504 No Content"** που ουσιαστικά είναι μια κενή απάντηση αλλά όχι μη έγκυρη, δηλαδή ο browser θα το θεωρήσει ως επιτυχημένη σύνδεση και όντως είναι, γιατί δεν υπήρξε πρόβλημα στην επικοινωνία με τον server.

Όπως παρατηρείται, όταν θέλουμε να καθυστερήσουμε τον τερματισμό μίας σύνδεσης, καλούμε την συνάρτηση **start_closing()**. Αυτή η συνάρτηση, τοποθετεί την καθυστέρηση που επιθυμούμε, σε ms, στον πίνακα καθυστερήσεων συνδέσεων **conn_timeout_list** και την χρονομέτρηση της, την αναλαμβάνει ο **Timer1**.

```
void start_closing(uint16_t close_delay, uint8_t id)
{
    if(close_delay && id < ESP_MAX_CONNECTIONS)
        conn_timeout_list[id][0] = close_delay;

    return;
}
```

Ύστερα, στην συνάρτηση listen() που αναλύσαμε προηγουμένως, καλείται η συνάρτηση **timed_close()**. Σε αυτήν την συνάρτηση ελέγχουμε αν η καθυστέρηση έληξε. Αν έληξε, τερματίζουμε την σύνδεση.

```
void timed_close()
{
    uint8_t k;

    for(k = 0; k < ESP_MAX_CONNECTIONS; k++)
    {
        //Αν το χρονόμετρο της σύνδεσης τελείωσε, κλείσε την σύνδεση.
        if(conn_timeout_list[k][0] && conn_timeout_list[k][1] >= conn_timeout_list[k][0])
            CIPCLOSE(k);
    }

    return;
}
```

Ο τερματισμός της σύνδεσης, είτε γίνει άμεσα ή καθυστερημένα, γίνεται μέσω της συνάρτησης **CIPCLOSE()**. Η συνάρτηση αυτή στέλνει την εντολή AT+CIPCLOSE με την κατάλληλη παράμετρο ως ID, στο ESP για να τερματίσει την αντίστοιχη σύνδεση.

```
void CIPCLOSE(uint8_t id)
{
    //Αυτή η εντολή τερματίζει την σύνδεση με συγκεκριμένο ID.

    //Αν η σύνδεση είναι ήδη κλειστή.
    if(connections[id].closed)
        return;

    //Ενεργοποίηση σημαίας.
    connections[id].closing = 1;

    //Αντιγραφή εντολής από την FLASH και αποστολή της.
    sprintf_P(in_buf, AT_CIPCLOSE_FORMAT, id);
    send_command(in_buf, 1, ESP_DEFAULT_ANSWER, 1, 1, 0);

    //Αφαίρεση σύνδεσης από την λίστα.
    remove_conn(id);

    //Απενεργοποίηση σημαίας.
    connections[id].closing = 0;

    //Καταγραφή γεγονότος.
    sprintf_P(in_buf, EVT_CLOSED_CONN, id);
    log_event(in_buf, 0, 1);

    return;
}
```

Για να μηδενίσουμε και να ελευθερώσουμε τον χώρο που καταλαμβάνει η σύνδεση στους πίνακες μας, χρησιμοποιούμε την συνάρτηση **remove_conn()**.

```
void remove_conn(uint8_t id)
{
    connections[id].closed = 1;

    ATOMIC_BLOCK(ATOMIC_FORCEON)
    {
        if(conn_count)
            conn_count--;

        connections[id].total_reqs = 0;
    }
    conn_timeout_list[id][0] = 0;
    conn_timeout_list[id][1] = 0;

    uint8_t i;

    //Διαγραφή αιτημάτων σύνδεσης.
    for(i = 0; i < ESP_CONNECTION_MAX_REQUESTS; i++)
    {
        if(connections[id].request[i])
        {
            free(connections[id].request[i]);
            connections[id].request[i] = 0;
        }
    }

    return;
}
```

Στην συνέχεια θα παρουσιάσουμε και θα αναλύσουμε μία από τις συναρτήσεις για αποστολή αρχείων ως απάντηση σε HTTP αίτημα, οπου καλύπτει και υπόλοιπες συναρτήσεις ταυτόχρονα. Θα θεωρήσουμε ότι λάβαμε αίτημα για την κύρια ιστοσελίδα **(index)** του server μας. Άρα, όταν το πρόγραμμα εισέλθει στον βρόγχο switch στην συνάρτηση **handle_request()**, θα εκτελέσει το κομμάτι κώδικα που ικανοποιεί την κατάλληλη συνθήκη:

```
//Στείλε την κατάλληλη απάντηση ανάλογα το αίτημα.
switch(i)
{
    //Αίτημα κύριας ιστοσελίδας index.
    case REQ_INDEX_HTML:
        SEND_INDEX_HTML(INDEX_HTML_CODE, id);
        break;
```

Άρα, θα καλέσει την συνάρτηση **SEND_INDEX_HTML()** με παραμέτρους την ίδια την ιστοσελίδα που θα στείλουμε και το ID της σύνδεσης. Η ιστοσελίδα καθώς και τα υπόλοιπα αρχεία και γενικότερα τα στατικά δεδομένα, αποθηκεύονται στην μνήμη **FLASH** γιατί δεν χωράνε στην SRAM. Περισσότερα γι' αυτό θα αναφερθούμε σε επόμενο κεφάλαιο.

Πρέπει να αναφέρουμε όμως, ότι η ιστοσελίδα αποθηκεύεται σε κομμάτια. Αυτό γιατί χρειαζόμαστε να τοποθετήσουμε την **IP** του ESP στο κατάλληλο **HTML πεδίο**. Για να γίνει αυτό, θα πρέπει να ξέρουμε ποιο κομμάτι είναι αυτό που περιέχει το πεδίο αυτό.
Γι' αυτό το λόγο, σπάμε την σελίδα που περιέχει το πεδίο, σε κομμάτια που μπορούμε να αναγνωρίσουμε. Στην συνάρτηση που θα αναλύσουμε παρακάτω, θα παρατηρήσετε ότι στέλνουμε τα κομμάτια ένα-ένα. Ο αριθμός των κομματιών ορίζεται στην σταθερά **INDEX_HTML_CODE_NUM_PARTS**.

```
void SEND_INDEX_HTML(const char *file[], uint8_t id)
{
    //Ορισμός και αρχικοποίηση βοηθητικών μεταβλητών.
    char *file_buf = in_buf;
    uint8_t start = 0, no_ans;
    uint16_t i, j, p_len, avail_space, send_len = 0, len = 0;

    //Υπολογισμός μεγέθους αρχείου.
    for(i = 0; i < INDEX_HTML_CODE_NUM_PARTS; i++)
        len += strlen_P(file[i]);

    len += strlen_P(ESP_IP) - 2; //Προσθέτουμε και το μέγεθος της IP μείον του συμβόλου "%s".

    memset(file_buf, 0, MAX_RESPONSE_STRING_SIZE); //Μηδενισμός γενικού buffer.

    //Χτίσιμο HTTP HEADER απάντησης.
    BUILD_HTTP_HEADER_200(file_buf, len, TEXT_HTML);
    p_len = strlen(file_buf); //Υπολογισμός μεγέθους απάντησης μέχρι στιγμής.

    //Ελεύθερος χώρος μέχρι στιγμής.
    avail_space = MAX_FILE_BUF_LENGTH - p_len - 1;

    //Αν τα δεδομένα δεν χωράνε, θα πρέπει να τα στείλουμε σε "πακέτα".
    if(len > avail_space)
    {
        char *body = file_buf + p_len; //Που ξεκινάει ο ελεύθερος χώρος;

        //Έτσι να στέλνεις "πακέτα".
        for(i = 0; i < INDEX_HTML_CODE_NUM_PARTS; i++)
        {
            p_len = strlen_P(file[i]);
            len = p_len;
```

```
j = 0;
send_len = 0;
start = 1;

//Αποστολή πακέτου.
while(j < p_len)
{
    //Αν στέλνουμε το πακέτο με το HTML στοιχείο για την IP, τότε πράττουμε ανάλογα.
    if(file[i] == INDEX_HTML_IP_FIELD)
    {
        char *ip = calloc(strlen_P(ESP_IP), sizeof(char));
        strcpy_P(ip, ESP_IP);
        sprintf_P(body, file[i], ip); //Εισαγωγή IP στον HTML στοιχείο.

        free(ip);

        p_len = strlen(body); //Νέο μέγεθος πακέτου.
        len = p_len;
    }
    else
    {
        strncpy_P(body, file[i] + j, avail_space); //Εισαγωγή δεδομένων πακέτου.
        body[avail_space] = 0;
    }

    //Αποστολή HTTP HEADER μόνο μία φορά.
    if(i == 0 && j == 0)
    {
        CIPSEND(file_buf, id, 1, 0);

        j += avail_space;
        send_len = j;
        len -= j;

        //Τώρα μπορούμε να χρησιμοποιούμε όλο τον buffer.
        body = file_buf;
        avail space = MAX FILE BUF LENGTH - 1;

        //Μικρή καθυστέρηση γιατί τώρα ο πελάτης στέλνει τα αιτήματα για τα επιπλέον αρχεία js css...
        _delay_ms(10);
    }
    else
    {
        j += avail_space; //Ενημέρωση μετρητή.

        //Είναι τα δεδομένα που θα στείλουμε τα τελευταία για την παρούσα ακολουθία CIPSEND;
        no_ans = j < p_len ? 1 : 0;

        //Αν αυτό είναι το πρώτο πακέτο μετά το HTTP HEADER ή υπερβήκαμε το μέγιστο μέγεθος αποστολής δεδομένων στο ESP,
        //ξεκίνα νέα ακολουθία αποστολής για το ESP.
        if(start)
        {
            //Αποστολή δεδομένων με μία εντολή CIPSEND.
            if(len < ESP_MAX_DATA_LENGTH)
            {
                CIPSEND_VAR(body, id, 1, len, no_ans);
                len -= avail_space;
            }
            else
            {
                //Αποστολή δεδομένων με τουλάχιστον δύο (2) εντολές CIPSEND.
                CIPSEND_VAR(body, id, 1, ESP_MAX_DATA_LENGTH, no_ans);
                send_len += avail_space;
            }

            start = 0;
        }
        else //Έχουμε ήδη ξεκινίσει ακολουθία CIPSEND.
        {
            CIPSEND(body, id, 0, no_ans); //Αποστολή δεδομένων.

            if(len >= ESP_MAX_DATA_LENGTH)
            {
                send_len += avail_space; //Ενημέρωση μετρητή.

                //Αν τα επόμενα δεδομένα υπερβαίνουν το όριο αποστολής δεδομένων στο ESP, ξεκίνα νέα ακολουθία CIPSEND.
                if(send_len + avail_space > ESP_MAX_DATA_LENGTH)
                {
```

```
                        //Πόσα μπορούμε να στείλουμε πριν ξεκινήσουμε την νέα ακολουθία;
                        avail_space = ESP_MAX_DATA_LENGTH - send_len;
                        len -= ESP_MAX_DATA_LENGTH; //Ενημέρωση μετρητή συνολικού εναπομείναντου αριθμού δεδομένων.

                        strncpy_P(body, file[i] + j, avail_space); //Εισαγωγή υπόλοιπων δεδομένων.
                        body[avail_space] = 0;

                        //Στέλνουμε τα τελευταία δεδομένα πριν την νέα ακολουθία.
                        CIPSEND(body, id, 0, 0);

                        start = 1; //Ενεργοποίηση σημαίας για να ξεκινήσουμε νέα ακολουθία.

                        //Ενημέρωση μετρητών.
                        j += avail_space;
                        avail_space = MAX_FILE_BUF_LENGTH - 1;
                        send_len = 0;
                    }
                }
                else
                    len -= avail_space;
            }
        }
    }
}
else
{
    //Αποστολή δεδομένων με μία ακολουθία.
    for(i = 0; i < INDEX_HTML_CODE_NUM_PARTS; i++)
    {
        //Αν στέλνουμε το πακέτο με το HTML στοιχείο για την IP, τότε πράττουμε ανάλογα.
        if(file[i] == INDEX_HTML_IP_FIELD)
            sprintf_P(file_buf + strlen(file_buf), file[i], ESP_IP);
        else
            strcat_P(file_buf, file[i]);
    }
    CIPSEND(file_buf, id, 1, 0);
}

return;
}
```

Η ροή της συνάρτησης είναι η εξής:

1. Υπολογισμός μεγέθους αρχείου.
2. Δημιουργία **HTTP Header** και εισαγωγή παραμέτρου "**Content-Length**", εφόσον στέλνουμε αρχείο.
3. Έλεγχος αποστολής σε ένα ή περισσότερα «πακέτα».
4. Αποστολή πακέτων με την χρήση της ESP εντολής, **AT+CIPSEND.** Επειδή η εντολή αυτή χωρίζεται σε δύο μέρη, ένα για τον ορισμό μεγέθους αποστολής και ένα για την αποστολή των δεδομένων, στέλνουμε πάντα μία εντολή AT+CIPSEND και ύστερα στέλνουμε τα δεδομένα συνεχόμενα, μέχρι να τελειώσουμε. Επειδή η εντολή αυτή έχει μέγιστο όριο αποστολής **2048 bytes,** ίσως χρειαστεί να ξαναστείλουμε επιπλέον εντολές AT+CIPSEND**,** μέχρι να αποστείλουμε όλο το αρχείο. Για την αποστολή του πρώτου πακέτου όμως, χρησιμοποιούμε μία ολόκληρη ακολουθία **CIPSEND**. Αυτό γιατί ο πελάτης (client), πρέπει να γνωρίζει όσο το δυνατόν πιο γρήγορα, τί και πόσα δεδομένα θα λάβει από εμάς (server). Αυτό το μαθαίνει μέσω του **HTTP Header**, γι' αυτό και χρησιμοποιούμε ξεχωριστή ακολουθία για το πακέτο που το εμπεριέχει.
5. Αποστολή υπόλοιπων πακέτων.

Με την ίδια λογική, λειτουργεί και η συνάρτηση **SEND_HTTP_FILE()**, που χρησιμοποιείται για την αποστολή αρχείων π.χ. JavaScript (.js), stylesheet (.css) κλπ.

```
void SEND_HTTP_FILE(const char file[], uint8_t id, CONTENT_TYPE type, uint8_t isLog)
{
    //Ορισμός και αρχικοποίηση βοηθητικών μεταβλητών.
    char *file_buf = in_buf;
    uint8_t start = 0, no_ans;
    uint16_t i = 0, header_len, avail_space, send_len = 0;
    uint16_t p_len = isLog ? 1000 : strlen_P(file), len = p_len; //Στέλνουμε log ή αρχείο;

    memset(file_buf, 0, MAX_RESPONSE_STRING_SIZE); //Μηδενισμός γενικού buffer.

    //Χτίσιμο HTTP HEADER απάντησης.
    BUILD_HTTP_HEADER_200(file_buf, p_len, type);
    header_len = strlen(file_buf);

    //Ελεύθερος χώρος μέχρι στιγμής.
    avail_space = MAX_FILE_BUF_LENGTH - header_len - 1;

    //Αν τα δεδομένα δεν χωράνε, θα πρέπει να τα στείλουμε σε "πακέτα".
    if(p_len > avail_space)
    {
        char *body = file_buf + header_len; //Που ξεκινάει ο ελεύθερος χώρος;

        //Αποστολή πακέτου.
        while(i < p_len)
        {
            //Αν η σύνδεση είναι κλειστή, μην κάνεις τίποτα.
            if(connections[id].closed)
                return;

            //Στέλνουμε log;
            if(isLog)
            {
                //Μην γεμίζεις όλο τον buffer αν δεν χρειάζεται.
                if(len < avail_space)
                    avail_space = len;

                //Αντιγραφή δεδομένων (log) από την EEPROM.
                eeprom_busy_wait();
                eeprom_read_block(body, buffer + i, avail_space);

                body[avail_space] = 0;
            }
            else
                strncpy_P(body, file + i, avail_space); //Εισαγωγή δεδομένων πακέτου.

            //Αποστολή HTTP HEADER μόνο μία φορά.
            if(i == 0)
            {
                CIPSEND(file_buf, id, 1, 0);

                i += avail_space;
                len -= i;

                //Τώρα μπορούμε να χρησιμοποιούμε όλο τον buffer.
                body = file_buf;
                avail_space = MAX_FILE_BUF_LENGTH - 1;
                start = 1;
            }
            else
            {
                i += avail_space; //Ενημέρωση μετρητή.

                //Είναι τα δεδομένα που θα στείλουμε τα τελευταία για την παρούσα ακολουθία CIPSEND;
                no_ans = i < p_len ? 1 : 0;
```

```
            //Αν αυτό είναι το πρώτο πακέτο μετά το HTTP HEADER ή υπερβήκαμε το μέγιστο μέγεθος αποστολής δεδομένων στο ESP,
            //ξεκίνα νέα ακολουθία αποστολής για το ESP.
            if(start)
            {
                //Αποστολή δεδομένων με μία εντολή CIPSEND.
                if(len < ESP_MAX_DATA_LENGTH)
                {
                    CIPSEND_VAR(body, id, 1, len, no_ans);
                    len -= avail_space;
                }
                else
                {
                    //Αποστολή δεδομένων με τουλάχιστον δύο (2) εντολές CIPSEND.
                    CIPSEND_VAR(body, id, 1, ESP_MAX_DATA_LENGTH, no_ans);
                    send_len += avail_space;
                }

                start = 0;
            }
            else //Έχουμε ήδη ξεκινίσει ακολουθία CIPSEND.
            {
                CIPSEND(body, id, 0, no_ans); //Αποστολή δεδομένων.

                if(len >= ESP_MAX_DATA_LENGTH)
                {
                    send_len += avail_space; //Ενημέρωση μετρητή.

                    //Αν τα επόμενα δεδομένα υπερβαίνουν το όριο αποστολής δεδομένων στο ESP, ξεκίνα νέα ακολουθία CIPSEND.
                    if(send_len + avail_space > ESP_MAX_DATA_LENGTH)
                    {
                        avail_space = ESP_MAX_DATA_LENGTH - send_len; //Πόσα μπορούμε να στείλουμε πριν ξεκινήσουμε την νέα ακολουθία;
                        len -= ESP_MAX_DATA_LENGTH; //Ενημέρωση μετρητή συνολικού εναπομείναντου αριθμού δεδομένων.

                        //Στέλνουμε log;
                        if(isLog)
                        {
                            //Αντιγραφή δεδομένων από EEPROM.
                            eeprom_busy_wait();
                            eeprom_read_block(body, buffer + i, avail_space);
                        }
                        else
                            strncpy_P(body, file + i, avail_space); //Εισαγωγή υπόλοιπων δεδομένων.

                        body[avail_space] = 0;

                        //Στέλνουμε τα τελευταία δεδομένα πριν την νέα ακολουθία.
                        CIPSEND(body, id, 0, 0);

                        start = 1; //Ενεργοποίηση σημαίας για να ξεκινήσουμε νέα ακολουθία.

                        //Ενημέρωση μετρητών.
                        i += avail_space;
                        avail_space = MAX_FILE_BUF_LENGTH - 1;
                        send_len = 0;
                    }
                }
                else
                    len -= avail_space;
            }
        }
    }
}
else
{
    //Αποστολή δεδομένων με μία ακολουθία.
    if(isLog)
    {
        //Αντιγραφή δεδομένων από EEPROM.
        eeprom_busy_wait();
        eeprom_read_block(file_buf, buffer, p_len);
    }
    else
        strcat_P(file_buf, file);

    CIPSEND(file_buf, id, 1, 0);
}

return;
}
```

Όταν δεν στέλνουμε αρχεία, για να απαντήσουμε σε ένα **HTTP GET Request**, θα μπορούσαμε απλώς να στείλουμε ως απάντηση την σκέτη την πληροφορία που ζητάει ο χρήστης. Επειδή όμως οι **Browsers (Google Chrome, Internet Explorer κ.α.)** μερικές φορές απορρίπτουν HTTP απαντήσεις που φαίνονται ύποπτες, υπάρχει μεγάλη πιθανότητα να μην γίνει εφικτή η επικοινωνία με το ESP.

Για να κάνουμε «φιλική» την απάντηση μας προς στους Browsers**,** φτιάχνουμε μια κανονική HTTP απάντηση, όπως αυτήν που στέλνουμε και για αρχεία, αλλά με την διαφορά ότι δεν βάζουμε την παράμετρο "**Content-Length**". Έτσι, λέμε στους Browsers ότι τα δεδομένα που ακολουθούν είναι απάντηση σε HTTP Request και ότι είναι ασφαλή, ανεξαρτήτως πηγής.

Ανάλογα με το είδος των δεδομένων και την τοποθεσία του server ίσως χρειαστούν επιπλέον HTTP παράμετροι. Προς το παρόν, το σύστημα μας χρησιμοποιεί τις βασικές παραμέτρους για να γίνει δεκτή η απάντηση.

Το χτίσιμο του HTTP Header αναλαμβάνει η συνάρτηση **BUILD_HTTP_HEADER_200()**. Η συνάρτηση αυτή δέχεται την ως παραμέτρους την συμβολοσειρά στην οποία θα αποθηκεύσει το header, το μήκος των δεδομένων αν πρόκειται για αρχεία και το είδος των δεδομένων. Το είδος των δεδομένων ορίζεται από την λίστα **enum CONTENT_TYPE**.

```
//Λίστα enum με τα είδη δεδομένων σε μία HTTP απάντηση.
enum CONTENT_TYPE
{
    TYPE_DEFAULT, //Αντιστοιχεί στο TEXT_PLAIN.
    TEXT_PLAIN,
    TEXT_HTML,
    TEXT_JAVASCRIPT,
    TEXT_CSS,
    APP_JSON,
    APP_OCTET,
    TYPE_LAST //Σύνολο τύπων.
};
typedef enum CONTENT_TYPE CONTENT_TYPE;
```

Υπάρχουν κι άλλα είδη δεδομένων αλλά για το σύστημα μας δεν χρειάζονται περισσότεροι. Ο χρήσης, αν επιθυμεί, μπορεί να προσθέσει καινούργιους σε αυτήν την λίστα.

```
void BUILD_HTTP_HEADER_200(char *str, uint16_t length, CONTENT_TYPE type)
{
    //Χτίσε το HTTP HEADER 200 με τις κατάλληλες παραμέτρους.

    //CORS header για cross-domain requests.
    strcpy_P(str, HTTP_STATUS_CODE_200); //Εισαγωγή HTTP OK.
    strcat_P(str, HTTP_HEADER_ALLOW_ORIGIN); //Επιτρέπουμε cross-over domain requests.

    //Εισαγωγή παραμέτρου μήκους, όταν χρειάζεται.
    if(length)
    {
```

```
        char *temp = calloc(strlen_P(HTTP_CONTENT_LENGTH_FORMAT) + 5, sizeof(char));

        sprintf_P(temp, HTTP_CONTENT_LENGTH_FORMAT, length);
        strcat(str, temp);

        free(temp);
    }

    //Εισαγωγή παραμέτρου τύπου δεδομένων, όταν χρειάζεται.
    if(type)
    {
        strcat_P(str, HTTP_CONTENT_TYPE_FORMAT);

        if(type < TYPE_LAST)
            strcat_P(str, CONTENT[type - 1]);
        else
            strcat_P(str, CONTENT[TYPE_DEFAULT]); //Προκαθορισμένος τύπος.
    }

    strcat(str, HTTP_DEFAULT_APPENDER); //Τέλος HTTP HEADER.

    return;
}
```

Ο χρήστης θα πρέπει να δώσει ιδιαίτερη προσοχή κατά το χτίσιμο του header, καθώς έστω και ένας χαρακτήρας να λείπει, το αίτημα μπορεί να απορριφθεί. Όπως παρατηρείτε, η απάντηση μας είναι θετική δηλαδή, το αίτημα ελήφθη με επιτυχία **(HTTP 200 SEND OK)**.

Όταν όμως παρουσιαστεί κάποιο πρόβλημα με το αίτημα, πρέπει να στείλουμε την κατάλληλη απάντηση. Για τον χειρισμό τέτοιων απαντήσεων, έχουμε υλοποιήσει τις συναρτήσεις **SEND_HTTP_HEADER_204(), SEND_HTTP_HEADER_400()** και **SEND_HTTP_HEADER_507**, όπου καλύπτουν τα περισσότερα είδη σφαλμάτων.

```
void SEND_HTTP_HEADER_204(uint8_t id)
{
    //Χτίσε HTTP HEADER 204 με τις κατάλληλες παραμέτρους.
    strcpy_P(in_buf, HTTP_STATUS_CODE_204);
    strcat_P(in_buf, HTTP_HEADER_ALLOW_ORIGIN);
    strcat(in_buf, HTTP_DEFAULT_APPENDER);

    //Αποστολή σκέτου HEADER.
    CIPSEND(in_buf, id, 1, 0);
    return;
}
void SEND_HTTP_HEADER_400(uint8_t id)
{
    //Χτίσε HTTP HEADER 400 με τις κατάλληλες παραμέτρους.
    strcpy_P(in_buf, HTTP_STATUS_CODE_400);
    strcat_P(in_buf, HTTP_HEADER_ALLOW_ORIGIN);
    strcat(in_buf, HTTP_DEFAULT_APPENDER);

    //Αποστολή σκέτου HEADER.
    CIPSEND(in_buf, id, 1, 0);
    return;
}
void SEND_HTTP_HEADER_507(uint8_t id)
{
    //Χτίσε HTTP HEADER 507 με τις κατάλληλες παραμέτρους.
    strcpy_P(in_buf, HTTP_STATUS_CODE_507);
    strcat_P(in_buf, HTTP_HEADER_ALLOW_ORIGIN);
    strcat(in_buf, HTTP_DEFAULT_APPENDER);

    //Αποστολή σκέτου HEADER.
    CIPSEND(in_buf, id, 1, 0);
    return;
}
```

8.11 Σύγκριση μνήμης FLASH και SRAM

Όπως αναφέραμε προηγουμένως, οι ιστοσελίδες και τα αρχεία του web server μας, είναι αποθηκευμένα στην μνήμη **FLASH,** δηλαδή την μνήμη στην οποία αποθηκεύεται το πρόγραμμα μας.

Ο ATmega32 διαθέτει **32KB Flash** και **2KB SRAM**. Όπως καταλαβαίνουμε, 2KB δεν φτάνουν για να αποθηκεύσουμε αρχεία όπως μία ιστοσελίδα, ένα JavaScript αρχείο κλπ.

Για να αντιμετωπιστεί αυτό το πρόβλημα, οι μικροελεγκτές της Atmel έχουν την δυνατότητα να αποθηκεύουν στατικά δεδομένα στην μνήμη FLASH τους, αφού φορτώσουμε την εσωτερική βιβλιοθήκη **<avr/pgmspace.h>**. Τοποθετώντας την εντολή «**PROGMEM»** δίπλα από το ορισμό μίας μεταβλητής, ο compiler θα τοποθετήσει τα δεδομένα της μεταβλητής αυτής, στην μνήμη FLASH:

```
static const char TRANSMISSION_SIGNAL[] PROGMEM          = "+IPD";
```

Η χρήση της μνήμης **FLASH** για αποθήκευση δεδομένων έχει δύο μικρά μειονεκτήματα:

1. Ο **χρόνος προσπέλασης** δεδομένων στην μνήμη FLASH, είναι **μεγαλύτερος** από αυτόν που είναι για δεδομένα στην SRAM. Η διαφορά δεν είναι μεγάλη αλλά πρέπει να την λάβει υπόψιν του ο χρήστης, κατά την υλοποίηση του συστήματος του.

2. Για να κάνουμε την χρήση εσωτερικών συναρτήσεων π.χ. **strlen()**, για μεταβλητές που περιέχουν δεδομένα στην FLASH, πρέπει να καλέσουμε τις αντίστοιχες συναρτήσεις που μπορούν να επεξεργαστούν δεδομένα στην FLASH π.χ. **strlen_P()**, όπου η κατάληξη **P** συμβολίζει ότι πρόκειται για συνάρτηση που αφορά δεδομένα στον **Program Space** (χώρος προγράμματος).

   *Η βιβλιοθήκη **<avr/pgmspace.h>** παρέχει τέτοιου είδους συναρτήσεις, για τις περισσότερες κανονικές συναρτήσεις συμβολοσειρών, όπου είναι και οι πιο συχνές στην χρήση.*

8.12 Αποστολή σειριακών δεδομένων, εντολών ESP και ανάλυση βοηθητικών συναρτήσεων

Στην συνέχεια θα αναλύσουμε εν συντομία τις συναρτήσεις **USART_write_char()**, **USART_write_string ()** που αφορούν την αποστολή σειριακών δεδομένων και την συνάρτηση **send_command()**, μέσω της οποίας στέλνουμε εντολές στο ESP.

```
void USART_write_char(char data, uint8_t slow)
{
    //Περίμενε όσο είναι γεμάτος ο buffer του Μεταδότη (Transmitter) και έχουμε ακόμα χρόνο αναμονής.
    while(!(UCSRA & (1 << UDRE)) && !timeout_2)
    {
        //Άδειος βρόχγος.
    }

    //Τώρα, γράψε τα δεδομένα.
    UDR = data;

    if(slow) //Θέλουμε καθυστέρηση;
        _delay_us(ESP_TRANSMIT_DELAY_US);
}
```

Για να στείλουμε σειριακά δεδομένα, δεν έχουμε τίποτε άλλο να κάνουμε παρά να ελέγξουμε αν ο καταχωρητής **UDR** περιέχει δεδομένα **(full)** και αν δεν έχει, γράφουμε τα δικά μας δεδομένα σε αυτόν.

Η μόνη αλλαγή που κάνουμε, είναι η εισαγωγή της παραμέτρου **slow**. Χρησιμοποιούμε αυτήν την παράμετρο όταν θέλουμε να έχουμε μια καθυστέρηση, μεταξύ της αποστολής σειριακών δεδομένων.

Στα αρχικά στάδια του συστήματος γινόταν η χρήση καθυστέρησης κατά την αποστολή δεδομένων. Αργότερα όμως, σταματήσαμε να την χρησιμοποιούμε γιατί δεν υπήρχε λόγος για καθυστέρηση. Παρόλα αυτά όμως, αποφασίσαμε να την κρατήσουμε στο πρόγραμμα μας για πιθανές μελλοντικές χρήσεις.

Η συνάρτηση **USART_write_string()**, ουσιαστικά «σπάει» μία συμβολοσειρά (string) της επιλογής μας, στους χαρακτήρες που το αποτελούν και τους στέλνει σειριακά έναν-έναν με την χρήση της συνάρτησης **USART_write_char()**, που αναλύσαμε πριν.

```
void USART_write_string(char *str, uint8_t append, uint8_t slow)
{
    uint16_t len = strlen(str); //Μέγεθος string.
    char *head = str, *last = head + len - 1;

    while(busy){} //Περίμενε μέχρι να τελειώσουμε την επεξεργασία νέων HTTP αιτημάτων, αν υπάρχουν.

    //Όσο δεν είμαστε στο τέλος του string και έχουμε ακόμα χρόνο αναμονής, στείλε τους χαρακτήρες.
    while(*head && !timeout_2)
    {
        /*
            Όσον αφορά την εντολη CIPSEND με την χρήση καθυστέρησης:

            Όταν στέλνουμε το τελευταίο byte των δεδομένων στο ESP, αυτό θα μας στείλει πίσω απάντηση SEND OK.
            Αν όμως έχουμε καθυστέρηση, δεν θα «πιάσουμε» την απάντηση.
            Οπότε, για τον τελευταίο χαρακτήρα μόνο, απενεργοποιούμε την καθυστέρηση.
        */
        if(head == last)
        {
            //Εισαγωγή appender «\r\n», αν το ζητάμε.
            if(append)
            {
                USART_write_char(*head, slow); //Γράψε τον τελευταίο χαρακτήρα πριν τον appender.
                head = ESP_DEFAULT_APPENDER;

                //Στείλε τον appender με την ίδια λογική.
                while(*head)
                {
                    if(*(head + 1) == 0)
                        USART_write_char(*head++, 0);
                    else
                        USART_write_char(*head++, slow);
                }
                break;
            }
            else
                USART_write_char(*head++, 0); //Στείλε χωρίς delay.
        }
        else
            USART_write_char(*head++, slow); //Στείλε με delay, αν πρέπει.
    }
}
```

Η συνάρτηση **send_command()**, διαχειρίζεται την αποστολή εντολών στο ESP.
Ως παραμέτρους δίνουμε:

1. την εντολή **(command)**,
2. τον χρόνο αναμονής **(timeout_sec)**,
3. την αναμενόμενη απάντηση του ESP **(response_str)**,
4. αν θέλουμε appender στην εντολή **(appendOnCmd)**,
5. αν θέλουμε appender στην απάντηση **(appendOnRes)** και τέλος,
6. αν θα έχουμε καθυστέρηση **(slow)** στην αποστολή δεδομένων.

Για να μετράμε τον χρόνο αναμονής, χρησιμοποιούμε τον **Timer2** και την μεταβλητή **timeout**. Σε περίπτωση που το ESP, αργήσει να απαντήσει, θεωρούμε ότι έχουμε timeout και προσπαθούμε πάλι για τρεις (3) φορές.

```
void send_command(char *command, uint8_t timeout_sec, char *response_str, uint8_t appendOnCmd, uint8_t appendOnRes, uint8_t slow)
{
    uint16_t i = strlen(response_str);
    char cmd = 0;   //Στέλνουμε εντολή ή απλά δεδομένα;
    char *res;      //String αναμενόμενης απάντησης.
    char *buf_res;  //String πραγματικής απάντησης στον buffer.

    cmd_status = CMD_NONE; //Αρχικοποιήση κατάστασης εντολής.

    //Αντιγραφή αναμενόμενης απάντησης, εφόσον είναι έγκυρη και μικρότερη από 16 χαρακτήρες.
    if(i && i < 16)
    {
        //Εισαγωγή appender στην απάντηση, αν το ζητάμε και χωράει.
        if(appendOnRes && (i + strlen(ESP_DEFAULT_APPENDER)) < 16)
        {
            i += strlen(ESP_DEFAULT_APPENDER);
            res = calloc(i + 1, sizeof(char));

            strcpy(res, response_str);
            strcat(res, ESP_DEFAULT_APPENDER);

            //Προς το παρόν θεωρούμε ότι όταν βάζουμε appender, στέλνουμε εντολή και όχι απλά δεδομένα.
            cmd = 1;
        }
        else
            res = response_str;
    }
    else if(i) //Η απάντηση είναι μεγαλύτερη από το όριο μας, επέστρεψε.
    {
        cmd_status = CMD_OVLOW;
        return;
    }

    //Βοηθητικοί δείκτες θέσεως για τους buffer.
    volatile uint16_t temp;
    volatile uint16_t temp_extra;

    //Πριν διαβάσεις τις τιμές, σιγουρέψου ότι δεν τις χρησιμοποιεί κανείς (π.χ. ο ISR).
    ATOMIC_BLOCK(ATOMIC_FORCEON)
    {
        temp = main_buf.index;
        temp_extra = extra_buf.index;
    }

    //Αρχικοποίηση χρόνου αναμονής και εκκίνηση Timer2.
    delay = timeout_sec;
    reset_timer2();
    //Αποστολή εντολής.
    USART_write_string(command, appendOnCmd, slow);

    //Αν δεν έχουμε string απάντησης, επέστρεψε με επιτυχία.
    if(!i && !timeout_2)
    {
        delay = 0;
        timeout_2 = 0;
        cmd_status = CMD_SUCCESS;

        return;
    }
    else if(timeout_2) //Αν είχαμε timeout, επέστρεψε με timeout.
    {
        delay = 0;
        timeout_2 = 0;
        cmd_status = CMD_TIMED;

        return;
    }

    char *error = 0;
    i = 0;

    //Λήψη απάντησης.
```

```
    do
    {
        //Μηδενισμός/επαναρχικοποίηση των βοηθητικών δεικτών αν γεμίσει ο buffer ή λάβαμε αίτημα, πριν λάβουμε την απάντηση.
        if(temp > main_buf.index)
            temp = 0;
        else if(temp < main_buf.last_cmd_end)
            temp = main_buf.last_cmd_end;
        else if(temp_extra > extra_buf.index)
            temp_extra = 0;
        else if(temp_extra < extra_buf.last_cmd_end)
            temp_extra = extra_buf.last_cmd_end;

        //Αναζήτηση αναμενόμενης απάντησης ή μήνυμα λάθους "error" και στους δύο buffer.
        if(((buf_res = strstr(main_buf.buffer + temp, res)) && (i = 1)) ||
           ((buf_res = strstr(extra_buf.buffer + temp_extra, res)) && (i = 2)) ||
            (cmd && (((error = strstr(main_buf.buffer + temp, AT_CMD_ERROR)) && (i = 1)) ||
            ((error = strstr(extra_buf.buffer + temp_extra, AT_CMD_ERROR)) && (i = 2)))))
        {
            //Αν βρήκαμε μήνυμα λάθους, τότε ενημέρωσε τους δείκτες που θα δώσουμε ως παραμέτρους.
            if(error)
            {
                buf_res = error;
                res = AT_CMD_ERROR;
                cmd_status = CMD_ERROR;
            }

            //Ενημέρωση βοηθητικών μεταβλητών των buffer.
            if(i == 1)
                USART_BUFFER_update(temp, &main_buf, buf_res - main_buf.buffer, res);
            else if(i == 2)
                USART_BUFFER_update(temp_extra, &extra_buf, buf_res - extra_buf.buffer, res);

            //Έξοδος από τον βρόγχο.
            break;
        }

    } while(!timeout_2); //Περίμενε για απάντηση όσο έχουμε χρόνο ακόμα.

    delay = 0; //Σταμάτα το χρονόμετρο.

    //Ελευθερώνουμε τον χώρο, αν στείλαμε εντολή.
    if(cmd)
        free(res);

    //Αν είχαμε timeout, προσπάθησε ξανά.
    if(timeout_2)
    {
        //Προσπάθησε μέχρι 3 φορές.
        if(timeout_tries++ < 3)
            send_command(command, timeout_sec, response_str, appendOnCmd, appendOnRes, slow);
        else //Αλλιώς, είχαμε timeout και τις 3 φορές.
        {
            timeout_2 = 0;
            timeout_tries = 0;
            cmd_status = CMD_TIMED;
        }
    }
    else if(cmd_status == CMD_ERROR) //Αν βρέθηκε μήνυμα λάθους, κατέγραψε την εντολή που απέτυχε.
    {
        eep_store_string("error");
        eep_store_string(command);
    }
    else //Επιτυχία.
        cmd_status = CMD_SUCCESS;

    return;
}
```

Η συνάρτηση αυτή είναι μεγάλη γιατί πρέπει να κάνουμε πολλούς ελέγχους πριν, κατά την αναμονή απάντησης αλλά και μετά. Η ροή της είναι η εξής:

1. Έλεγχος αναμενόμενης απάντησης και εισαγωγή appender, αν πρέπει.
2. Αρχικοποίηση και εκκίνηση Timer2 (χρονόμετρο).
3. Αποστολή εντολής/δεδομένων και επιστροφή αν δεν υπάρχει απάντηση.
4. Αναζήτηση και αναμονή απάντησης ή μηνύματος λάθος, όσο δεν τελείωσε ο χρόνος αναμονής.
5. Αν βρέθηκε η αναμενόμενη απάντηση, ενημέρωση βοηθητικών μεταβλητών των buffer, έξοδος από τον βρόγχο και επιστροφή με επιτυχία.
6. Αν είχαμε timeout προσπάθησε πάλι **(Βήμα 1).**
7. Αν είχαμε timeout 3 φορές, έξοδος από τον βρόγχο, και επιστροφή με αποτυχία.
8. Αν βρέθηκε μήνυμα λάθους έξοδος από τον βρόγχο, και επιστροφή με αποτυχία.

Για την αρχικοποίηση των βοηθητικών δεικτών θέσεως για τους buffer, χρησιμοποιούμε την συνάρτηση **ATOMIC_BLOCK**. Η συνάρτηση αυτή απενεργοποιεί όλες τις διακοπές (interrupt) του μικροελεγκτή, μέχρι να τελειώσει να εκτελεί των κώδικα της.

Αυτό είναι χρήσιμο όταν θέλουμε να προσπελάσουμε μεταβλητές μεγαλύτερες των 8-bit, που χρησιμοποιούνται από ρουτίνες διακοπής ISR και υπάρχει πιθανότητα την ώρα που την διαβάζουμε να εκτελεστεί μία τέτοια ρουτίνα διακοπής. Αν η ρουτίνα αλλάξει τα περιεχόμενα της μεταβλητής, τότε όταν επιστρέψουμε στο κυρίως πρόγραμμα, αφού θα έχουμε διαβάσει την μισή μεταβλητή, το άλλο μισό θα είναι καινούργιο και δεν θα αντιστοιχεί με το παλιό.

Παραθέτουμε ένα παράδειγμα για να γίνει καλύτερα κατανοητό.

1. Διάβασε μία μεταβλητή των 16-bit.
    a. Προσπέλαση και αντιγραφή του πρώτου μισού 8-bit της μεταβλητής.
    b. Το πρόγραμμα σταματάει και εκτελεί μία ρουτίνα διακοπής (αλλαγή περιεχομένων μεταβλητής).
    c. Επιστροφή στο κυρίως πρόγραμμα.
    d. Προσπέλαση και αντιγραφή ΔΙΑΦΟΡΕΤΙΚΟΥ δεύτερου μισού 8-bit της μεταβλητής.

Την συνάρτηση **ATOMIC_BLOCK** την χρησιμοποιούμε σε διάφορα σημεία του προγράμματος μας, όταν διαβάζουμε μεταβλητές που χρησιμοποιούνται από ISR και υπάρχει πιθανότητα να αλλάξουν.

Η συνάρτηση **USART_BUFFER_update()** χρησιμοποιείται για να ενημερώνουμε τις βοηθητικές μεταβλητές των buffer. Κάθε φορά που στέλνουμε εντολές/δεδομένα ή λαμβάνουμε αιτήματα μέσω της σειριακής σύνδεσης, ενημερώνουμε και τις βοηθητικές μεταβλητές στους buffer.

```
        }

        main_buf.buffer[main_buf.last_cmd_end - 1] = 0;
        if(del)
            main_buf.buffer[main_buf.last_cmd_end - 2] = 0;
    }
    else
    {
        //Αν ο άλλος buffer δεν έκανε επανεκκίνηση, ενημέρωσε τις βοηθητικές μεταβλητές του.
        if(!main_buf.reset)
        {
            ATOMIC_BLOCK(ATOMIC_FORCEON)
            {
                main_buf.last_cmd_end = EXTRA_BUFFER_START + cmd_end;
                main_buf.last_req_end = main_buf.last_cmd_end;
            }
        }
                                                                                απάντηση αργότερα.
        main_buf.buffer[EXTRA_BUFFER_START + cmd_end - 1] = 0;
        if(del)                                                                 ιου έρχεται μετά τον χαρακτήρα '>'.
            main_buf.buffer[EXTRA_BUFFER_START + cmd_end - 2] = 0;              /η απάντηση του ESP σε εντολές.
    }
}

return;
}

 //Ενημέρωση βοηθητικών μεταβλητών και εισαγωγή τερματικού χαρακτήρα στον άλλο buffer.
 if(buf == &main_buf)
 {
     if(cmd_end > EXTRA_BUFFER_START)
     {
         ATOMIC_BLOCK(ATOMIC_FORCEON)
         {
             extra_buf.last_cmd_end = cmd_end - EXTRA_BUFFER_START;
             extra_buf.last_req_end = extra_buf.last_cmd_end;
         }
         extra_buf.buffer[extra_buf.last_cmd_end - 1] = 0;
         if(del)
             extra_buf.buffer[extra_buf.last_cmd_end - 2] = 0;
     }
     else if(cmd_end <= EXTRA_BUFFER_END + 1)
     {
         //Αν ο άλλος buffer δεν έκανε επανεκκίνηση, ενημέρωσε τις βοηθητικές μεταβλητές του.
         if(!extra_buf.reset)
         {
             ATOMIC_BLOCK(ATOMIC_FORCEON)
             {
                 extra_buf.last_cmd_end = MAX_EXTRA_BUFFER_SIZE / 2 + cmd_end;
                 extra_buf.last_req_end = extra_buf.last_cmd_end;
             }

             extra_buf.buffer[MAX_EXTRA_BUFFER_SIZE / 2 + cmd_end - 1] = 0;
             if(del)
                 extra_buf.buffer[MAX_EXTRA_BUFFER_SIZE / 2 + cmd_end - 2] = 0;
         }
     }
 }
 else if(buf == &extra_buf)
 {
     if(cmd_end > MAX_EXTRA_BUFFER_SIZE / 2)
     {
         ATOMIC_BLOCK(ATOMIC_FORCEON)
         {
             main_buf.last_cmd_end = cmd_end - MAX_EXTRA_BUFFER_SIZE / 2;
             main_buf.last_req_end = main_buf.last_cmd_end;
```

Αν π.χ. στείλουμε μία εντολή στο ESP, πρέπει να κρατήσουμε την θέση από την οποία ξεκινάει και τελειώνει η απάντηση της. Στα αιτήματα κρατάμε μόνο την θέση στην οποία τελειώνουν. Στο σύστημα μας όμως, λόγω του βοηθητικού buffer που χρησιμοποιούμε, πρέπει να ενημερώνουμε και αυτόν γιατί κάνουμε αναζητήσεις και

σε αυτόν. Γι’ αυτό κάθε φορά που ενημερώνουμε έναν buffer, ουσιαστικά ενημερώνουμε και τον άλλον.

Όπως παρατηρείτε, πρέπει να κάνουμε πολλούς ελέγχους για να ενημερώσουμε τους και τους δύο (2) buffer, όπως επίσης και για την εύρεση της θέσης για την εισαγωγή του τερματικού χαρακτήρα. Πρέπει να είμαστε προσεκτικοί, καθώς η συνάρτηση αυτή έχει καθοριστικό ρόλο στην αναζήτηση και εύρεση των απαντήσεων ή αιτημάτων.

Πρέπει να λάβουμε υπόψιν όλα τα πιθανά σενάρια που μπορούν να προκύψουν, καθώς και τα περίεργα **edge cases**, που μπορούν να αποβούν μοιραία κατά την εκτέλεση του προγράμματος μας. Υπενθυμίζουμε ότι αυτή είναι μία από τις πολλές υλοποιήσεις που μπορούν να γίνουν για τέτοιου είδους συστήματα. Ο χρήστης πρέπει να σκεφτεί ο ίδιος αν θα πρέπει να σχεδιάσει εξολοκλήρου δικό του σύστημα ή να βασιστεί σε ένα ήδη υπάρχον.

Στο επόμενο κεφάλαιο θα αναλύσουμε τις συναρτήσεις και **ISR** ρουτίνες για τους **Timer** και θα εξηγήσουμε πως λειτουργούν.

8.13 Timers/Counters και ανάλυση βοηθητικών συναρτήσεων

Η συναρτήσεις **reset_timer1()** και **reset_timer2()**, επαναφέρουν τις μεταβλητές που χρησιμοποιούμε στις αρχικές τιμές τους και ενεργοποιούν τους **Timer1** και **Timer2** αντίστοιχα, με **prescaler = 8**.

```
void reset_timer1()
{
    milliseconds_1 = 0;
    TCNT1 = 0; //Μηδενισμός μετρητή.

    TCCR1B |= (1 << CS11); //Βάλε prescaler = 8 και ξεκίνα τον Timer.
}
void reset_timer2()
{
    timeout_2 = 0;
    milliseconds_2 = 0;
    seconds_2 = 0;
    TCNT2 = 0; //Μηδενισμός μετρητή.

    TCCR2 |= (1 << CS21); //Βάλε prescaler = 8 και ξεκίνα τον Timer.
}
```

Όταν αναλύαμε τις λειτουργίες των **Timer**, είχαμε αναφέρει ότι όταν ο καταχωρητής **TCNT** φτάσει και έχει ίδια τιμή με τον καταχωρητή **OCR**, στέλνεται ένα σήμα **TOV (timer overflow)**.

Αυτό το σήμα, χρησιμοποιείται για ένα εκτελεστεί μία **Διακοπή (Interrupt)**. Αυτή η διακοπή είναι η συνάρτηση **ISR (Interrupt Service Routine)**.

Αυτή η συνάρτηση/ρουτίνα εκτελείται κάθε φορά που έχουμε ένα **TOV** σήμα. Η ρουτίνα αυτή δεν θα ενεργοποιούταν αν δεν είχαμε καλέσει την συνάρτηση **sei()**, που ενεργοποιεί τις **Διακοπές (Interrupts)**.

Ο κώδικας στις ρουτίνες ISR δεν πρέπει να είναι αρκετά μεγάλος γιατί υπάρχει πιθανότητα να μην προλάβει να τελειώσει η ρουτίνα, πριν γίνει η επόμενη διακοπή. Παρόλα αυτά, σε αυτήν την περίπτωση ο μικροελεγκτής «θυμάται» ότι δεν πρόλαβε να τελειώσει η προηγούμενη εκτέλεση της ISR και όταν τελικά τελειώσει, θα εκτελέσει κατευθείαν την ρουτίνα, παρόλο που άργησε.

Με λίγα λόγια, ο μικροελεγκτής μας «σώζει», αλλά αυτό ισχύει μόνο όταν είχαμε μία διακοπή πριν τελειώσει η προηγούμενη. Αν γίνουν περισσότερες από μία διακοπές, χωρίς να έχει τελειώσει η πρώτη, τότε η ρουτίνα δεν θα εκτελεστεί για τις υπόλοιπες.

Συνιστάται ο χρήστης να υλοποιήσει έτσι τον κώδικα ώστε να μην γίνεται καμία διακοπή πριν τελειώσει η προηγούμενη, καθώς αυτό μπορεί να επηρεάσει αρνητικά την λειτουργία του προγράμματος μας και γενικά την αποσφαλμάτωση του (debugging).

```
ISR(TIMER1_COMPA_vect)
{
    uint8_t k;

    milliseconds_1++;

    for(k = 0; k < ESP_MAX_CONNECTIONS; k++)
    {
        //Αν η σύνδεση πρέπει να κλείσει σε λίγο, ξεκίνα το χρονόμετρο.
        if(conn_timeout_list[k][0])
        {
            conn_timeout_list[k][1]++; //Ενημέρωση χρονομέτρου.

            //Αν η καθυστέρηση τελείωσε, κλείσε την σύνδεση.
            if(conn_timeout_list[k][1] >= conn_timeout_list[k][0])
            {
                conn_timeout_list[k][1] = conn_timeout_list[k][0]; //Μην πας παραπάνω.

                //Αν κατά την καθυστέρηση ήρθε νέο αίτημα για την σύνδεση, μηδένισε τους μετρητές,
                //ακυρώνοντας έτσι τον τερματισμό της.
                if(connections[k].total_reqs > 0)
                {
                    conn_timeout_list[k][0] = 0;
                    conn_timeout_list[k][1] = 0;
                    continue;
                }
            }
        }
    }

    //Ενημέρωση ρολογιού.
    if(milliseconds_1 >= 1000)
    {
        milliseconds_1 = 0;

        //Μέτρα χρόνο μόνο όταν δεν συγχρονίζουμε το ρολόι.
        if(!sync_time)
        {
            cur_time.sec++;

            if(cur_time.sec > 59)
            {
                cur_time.sec = 0;
                cur_time.min++;

                if(cur_time.min > 59)
                {
                    cur_time.min = 0;
                    cur_time.hour++;

                    if(cur_time.hour > 23)
                    {
                        cur_time.hour = 0;
                        cur_time.day++;

                        //Συγχρονισμός ρολογιού κάθε μία μέρα.
                        sync_time = 1;
                        //Έλεγχος για τον Φεβρουάριο.
                        if(cur_time.mon == 2)
                        {
                            uint8_t is_leap = (((cur_time.year % 4 == 0) && (cur_time.year % 100!= 0))
                                                || (cur_time.year % 400 == 0)) ? 1 : 0;

                            if((cur_time.day == 29 && !is_leap) || cur_time.day == 30)
                            {
                                cur_time.day = 0;
                                cur_time.mon++;
                            }
                        }
                        else if(cur_time.day == 31)
                        {
                            if(cur_time.mon == 3 || cur_time.mon == 5 ||
                            cur_time.mon == 8 || cur_time.mon == 10)
                            {
                                cur_time.day = 0;
                                cur_time.mon++;
                            }
                        }
```

```
                    else if(cur_time.day >= 32)
                    {
                        cur_time.day = 0;
                        cur_time.mon++;
                    }

                    if(cur_time.mon > 12)
                    {
                        cur_time.mon = 0;
                        cur_time.year++;
                    }
                }
            }
        }
    }
}
```

Η παράμετρος **TIMER1_COMPA_vect (Timer/Counter1 Compare Match A)** υποδηλώνει ότι η ρουτίνα αυτή αφορά τον Timer1 και ότι χρησιμοποιείται **Σύγκριση (Compare)** για να σταλεί το **TOV** σήμα.

Θα μπορούσαμε να έχουμε και δεύτερη ρουτίνα διακοπής για τον ίδιο Timer, με τη χρήση της παραμέτρου **TIMER1_COMPB_vect**, αφού όμως αρχικοποιούσαμε κατάλληλα τον **Timer1**, κάπως έτσι:

```
ISR(TIMER1_COMPB_vect)
{
        ……
}
```

Ο **Timer2**, στον ATmega32, έχει μόνο μία **Διακοπή** για **Σύγκριση (Compare)**. Χρησιμοποιούμε αυτόν τον Timer για να μετράμε τον χρόνο αναμονής, για να λάβουμε μία απάντηση σε μία εντολή που στείλαμε στο ESP. Αν ο χρόνος τελειώσει πριν λάβουμε την απάντηση, τότε έχουμε **timeout**.

```
ISR(TIMER2_COMP_vect)
{
    milliseconds_2++;

    //Απενεργοποίηση Timer όταν δεν υπάρχει χρόνος αναμονής.
    if(!delay)
        TCCR2 &= ~(1 << CS22);

    if(milliseconds_2 >= 1000)
    {
        milliseconds_2 = 0;
        seconds_2++;

        //Αν ο χρόνος αναμονής τελείωσε, σήμανε timeout.
        if(delay && seconds_2 >= delay)
        {
            delay = 0;
            timeout_2 = 1;
        }
    }
}
```

Στον **Timer1** υλοποιούμε το ρολόι και το χρονόμετρο για τον τερματισμό των συνδέσεων. Στον **Timer2** υλοποιούμε το χρονόμετρο για τον χρόνο αναμονής απάντησης για εντολές του ESP.

8.14 Ανάλυση της βασικής μορφής συναρτήσεων ESP εντολών και της συνάρτησης CIPSEND()

Ουσιαστικά, όταν θέλουμε να στείλουμε μία εντολή, καλούμε και την αντίστοιχη συνάρτηση που έχουμε υλοποιήσει. Για κάποιες απλές εντολές δεν έχουμε υλοποιήσει συνάρτηση, αλλά η λογική εξακολουθεί να είναι η ίδια.

Ας πάρουμε την συνάρτηση για την εντολή **AT+RST**:

```
void RST()
{
    //Αυτή η εντολή κάνει επανεκκίνηση (soft reset) τον ESP.

    //Αντιγραφή εντολής από την FLASH και αποστολή της.
    strcpy_P(in_buf, AT_RST);
    send_command(in_buf, 5, ESP_DEFAULT_ANSWER, 1, 1, 0);

    return;
}
```

Όπως παρατηρείτε, πρώτα αντιγράφουμε σε έναν γενικό buffer την εντολή, με την χρήση της κατάλληλης συνάρτησης για επεξεργασία δεδομένων στην FLASH. Ύστερα στέλνουμε την εντολή καλώντας την **send_command()** με τις κατάλληλες παραμέτρους (εντολή, χρόνος αναμονής, αναμενόμενη απάντηση, appender στην εντολή, appender στην απάντηση, καθυστέρηση κατά την αποστολή).

Η ίδια λογική ισχύει και για τις υπόλοιπες συναρτήσεις που στέλνουν εντολές. Η μόνη που διαφέρει είναι η συνάρτηση για την εντολή **AT+CIPSEND**. Επειδή, όπως αναφέραμε και πιο πριν, η εντολή αυτή έχει δύο (2) στάδια, πρέπει να κάνουμε διάφορους ελέγχους. Παρόλα αυτά, η λογική παραμένει ίδια.

Επίσης, για την εντολή **AT+CIPSEND** έχουμε υλοποιήσει δύο συναρτήσεις: **CIPSEND()** και **CIPSEND_VAR()**. Η συνάρτηση CIPSEND_VAR() διαφέρει από την CIPSEND() μόνο στο γεγονός ότι για το μέγεθος των δεδομένων που θα στείλουμε, δεν μετράει το μέγεθος του buffer των δεδομένων, αλλά παίρνει μία σταθερή τιμή που του ορίζουμε εμείς μέσω της ειδικής παραμέτρου της συνάρτησης.

Αυτό γιατί όταν στέλνουμε αρχεία σε μία σύνδεση, όπως είπαμε, το μέγεθος αρχείων ξεπερνάει το όριο των buffer μας, άρα δεν μπορούμε να βασιστούμε στο μέγεθος του buffer μας. Γι' αυτό στέλνουμε το συνολικό μέγεθος ως παράμετρο στην συνάρτηση.

*Π.χ. CIPSEND_VAR(data, id, 1, **LENGTH**, 0)*

```
void CIPSEND(char *data, uint8_t id, uint8_t first, uint8_t no_ans)
{
    //Αυτή η εντολή στέλνει δεδομένα σε σύνδεση με συγκεκριμένο ID.

    //Αν η σύνδεση είναι κλειστή, μην κάνεις τίποτα.
    if(connections[id].closed)
        return;

    //Αν στέλνουμε δεδομένα σε πακέτα, τότε στέλνουμε μία φορα την εντολή.
    if(first)
    {
        //Αρχικοποίηση buffer εντολής.
        uint8_t cmd_size = strlen_P(AT_CIPSEND_FORMAT) + 4 + 2; //4 είναι ο μέγιστος αριθμός ψηφίων του μεγέθους δεδομένων και
                                                                //2 για το ID και τον τερματικό χαρακτήρα.
        cmd_buf = calloc(cmd_size, sizeof(char));

        sprintf_P(cmd_buf, AT_CIPSEND_FORMAT, id, strlen(data)); //Αντιγραφή εντολής και εισαγωγή παραμέτρων.
        send_command(cmd_buf, 2, ">", 1, 0, 0); //Αποστολή εντολής.

        //Αναγκαστική καθυστέρηση τουλάχιστον 100ms, σύμφωνα με οδηγίες τρίτων, για να προλαβαίνει το ESP.
        _delay_ms(100);
    }
    else //Αλλιώς στείλε μόνο δεδομένα.
    {
        //Χρειαζόμαστε την απάντηση μόνο στο τελευταίο πακέτο δεδομένων.
        if(!no_ans)
            cmd_buf = calloc(strlen_P(AT_CIPSEND_SEND_OK) + 1, sizeof(char));

        cmd_status = CMD_SUCCESS; //Δεν στείλαμε εντολή, άρα θεωρούμε ότι είχαμε επιτυχία για να περάσουμε τον έλεγχο μετά.
    }

    //Αν στείλαμε εντολή και είχαμε επιτυχία τότε στείλε τα δεδομένα.
    if(cmd_status == CMD_SUCCESS)
    {
        //Αν δεν περιμένουμε απάντηση (δεν είναι το τελευταίο πακέτο), στείλε τα δεδομένα κατευθείαν.
        if(no_ans)
            USART_write_string(data, 0, 0);
        else //Αλλιώς στείλε το τελευταίο πακέτο δεδομένων και περίμενε απάντηση.
        {
            sprintf_P(cmd_buf, AT_CIPSEND_SEND_OK);
            send_command(data, 5, cmd_buf, 0, 1, 0);
        }
    }

    //Αν δεν είχαμε επιτυχία, κλείσε την σύνδεση.
    if(cmd_status != CMD_SUCCESS)
        CIPCLOSE(id);

    //Ελευθέρωσε τον χώρο, όταν πρέπει.
    if(first || !no_ans)
        free(cmd_buf);

    return;
}
```

Βλέπετε λοιπόν ότι δεν είναι μία απλή αντιγραφή και αποστολή της εντολής. Πρέπει να περάσουμε από πολλά στάδια και ελέγχους πριν τελειώσουμε την αποστολή των δεδομένων.

Στο επόμενο κεφάλαιο θα αναλύσουμε τις διαδικασίες επεξεργασίας των HTTP αιτημάτων και θα αναλύσουμε τις αντίστοιχες συναρτήσεις.

8.15 Επεξεργασία HTTP αιτημάτων, απαντήσεων και ανάλυση βοηθητικών συναρτήσεων

Στο σύστημα μας έχουμε σχεδιάσει και υλοποιήσει εννέα (9) αιτήματα, στα οποία μπορούμε να απαντήσουμε επιτυχώς. Παραθέτουμε τον πίνακα με τα αιτήματα και τις αντίστοιχες συναρτήσεις που αναλαμβάνουν την απάντηση τους.

Πίνακας 8.15.1. Αντιστοίχιση αιτημάτων και συναρτήσεων απάντησης τους.

Αίτημα	**Συνάρτηση**
/?all=1	get_full_IO
/?upd_io={"ddr":["XX","XX","XX","XX"],"port":["XX","XX","XX","XX"]}	update_IO
/?toggle=a1 ή /?toggle=a1&state=0	toggle_IO
/?temp=1	send_temperature
/?admin=1	admin_connect
/?dlog=1	download_log
/?clog=1	clear_log
/?reset_esp=1	reset_esp
/?reset_chip=1	reset_chip

Η συνάρτηση **get_full_IO()** καλείται για να απαντήσουμε στο HTTP αίτημα που ζητάμε την πλήρη κατάσταση όλων των I/O Πορτών του μικροελεγκτή.

GET /?all=1

Όταν λέμε την πλήρη κατάσταση, εννοούμε τις τιμές των καταχωρητών **DDR**, **PORT** και **PIN** για κάθε πόρτα. Ο καλύτερος τρόπος για να στείλουμε τέτοιου είδους δεδομένα είναι να μετατρέψουμε σε μία **JSON (JavaScript Object Notation)** συμβολοσειρά. Η JSON συμβολοσειρές δεν είναι τίποτε άλλο παρά συμβολοσειρές με συγκεκριμένη μορφή για μεταφορά δεδομένων.

Π.χ. μία JSON συμβολοσειρά για τα στοιχεία ενός πελάτη:

```
{
  "Όνομα": "John",
  "Επώνυμο": "Smith",
  "Τηλέφωνα": [
        "1234567890",
        "0987654321"
    ]
}
```

Με τον ίδιο τρόπο ακριβώς, στέλνουμε και τις δικές μας απαντήσεις, για όλα τα αιτήματα που αφορούν καταστάσεις για I/O Ports. Για κάποια αιτήματα έχουμε προκαθορισμένες τις μορφές των συμβολοσειρών στην μνήμη FLASH ως σταθερές.

Για άλλα αιτήματα όμως, πρέπει να «χτίζουμε» επιτόπου την JSON συμβολοσειρά, γιατί δεν γίνεται αλλιώς.

<u>Για την κωδικοποίηση και αποκωδικοποίηση JSON συμβολοσειρών, υλοποιήσαμε δικές μας συναρτήσεις και σε κανένα στάδιο δεν γίνεται χρήση εξωτερικών βιβλιοθηκών.</u>

Η ροή της συνάρτησης είναι εξής:

1. Χτίσιμο **HTTP HEADER 200** (επιτυχής απάντηση).
2. Αρχικοποίηση JSON συμβολοσειράς που θα στείλουμε ως απάντηση.
3. Εισαγωγή όλων των καταστάσεων **(DDR, PORT, PIN)** για όλες τις **Πόρτες** του μικροελεγκτή, στην JSON συμβολοσειρά.
4. Αποστολή JSON συμβολοσειράς ως απάντηση.

Υπενθυμίζουμε ότι σε κάθε συνάρτηση, έχουν τοποθετηθεί σχόλια για την καλύτερη κατανόηση της ροής της.

```
void get_full_IO(uint8_t id)
{
    //Αυτό το αίτημα επιστρέφει την πλήρη κατάσταση όλων των Port του μικροελεγκτή.
    char *data_buf = in_buf;

    //Χτίσιμο HTTP HEADER απάντησης.
    memset(data_buf, 0, MAX_RESPONSE_STRING_SIZE);
    BUILD_HTTP_HEADER_200(data_buf, 0, APP_JSON);

    //Καλό είναι να μην καταναλώσουμε νεό χώρο γιατί τα JSON strings είναι μεγάλα και ίσως προκαλέσουμε SRAM overflow.
    char *json = data_buf + strlen(data_buf);

    //get /?all=1

    //Εισαγωγή κατάλληλων δεδομένων ως JSON συμβολοσειρά.
    if(CHIP == 32)
        sprintf_P(json, FULL_IO_JSON, DDRA, DDRB, DDRC, DDRD,
                                      PORTA, PORTB, PORTC, PORTD,
                                      PINA, PINB, PINC, PIND);
    else
        sprintf_P(json, FULL_IO_JSON, DDRB, DDRC, DDRD,
                                      PORTB, PORTC, PORTD,
                                      PINB, PINC, PIND);

    //Αποστολή JSON.
    CIPSEND(data_buf, id, 1, 0);

    return;
}
```

Σε αυτήν την συνάρτηση, έχουμε προκαθορισμένη την μορφή της JSON συμβολοσειράς, στην σταθερά **FULL_IO_JSON**, η οποία είναι εξής:

```
static const char FULL_IO_JSON[] PROGMEM = "{"
                                               "\"ddr\":[\"%X\",\"%X\",\"%X\",\"%X\"],"
                                               "\"port\":[\"%X\",\"%X\",\"%X\",\"%X\"],"
                                               "\"pin\":[\"%X\",\"%X\",\"%X\",\"%X\"]"
                                           "}";
```

Άρα η τελική JSON συμβολοσειρά θα μοιάζει κάπως έτσι:

{"ddr":["23","A1","CC","25"],"port":["B","A","E1","67"],"pin":["FF","86","45","F"]}

Η επόμενη συνάρτηση που θα αναλύσουμε είναι η **update_IO()**. Αυτή η συνάρτηση καλείται για να απαντήσουμε σε HTTP αίτημα που ενημερώνουμε καταστάσεις, μίας ή πολλών, I/O Πορτών του μικροελεγκτή.

GET /?upd_io={"ddr":["XX","XX","XX","XX"],"port":["XX","XX","XX","XX"]}

Αυτό το αίτημα είναι χρήσιμο όταν θέλουμε να ενημερώσουμε καταστάσεις πολλαπλών Πορτών του μικροελεγκτή, ταυτόχρονα. Συγκεκριμένα, οι καταστάσεις που μπορούμε να ελέγξουμε είναι οι **DDR** και **PORT**. Ως απάντηση λαμβάνουμε τις νέες καταστάσεις αντίστοιχα, συμπεριλαμβανομένης όμως και της κατάστασης **PIN**.

Οι θέσεις των τιμών σε κάθε ιδιότητα (property) στην JSON συμβολοσειρά, αντιστοιχούν σε Πόρτες του μικροελεγκτή. Δηλαδή, η πρώτη θέση αφορά την **Πόρτα A**, η δεύτερη στην **Πόρτα B** κοκ. Γι' αυτό, όταν μία Πόρτα δεν αλλάζει καθόλου, για να κρατήσουμε την σειρά ανέπαφη, τοποθετούμε μία κενή συμβολοσειρά **“”**.

```
void update_IO(char *req, uint8_t id)
{
    //Αυτό το αίτημα ενημερώνει την κατάσταση μίας ή πολλών Port του μικροελεγκτή.

    char *data_buf = in_buf;
    char *param;

    //get /?upd_io={"ddr":["XX","XX","XX","XX"],"port":["XX","XX","XX","XX"]}

    //Βρες την θέση της παραμέτρου "upd_io".
    param = strcasestr_P(req, GET_SET_FULL_IO_PARAM);

    //Αν δεν βρέθηκε η παράμετρος, στείλε απάντηση HTTP 400.
    if(!param[strlen_P(GET_SET_FULL_IO_PARAM)])
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Χτίσιμο HTTP HEADER απάντησης.
    memset(data_buf, 0 , MAX_RESPONSE_STRING_SIZE);
    BUILD_HTTP_HEADER_200(data_buf, 0, TEXT_PLAIN);

    //Επεξεργασία JSON string και ενημέρωση IO μικροελεγκτή.
    json_to_io(param + strlen_P(GET_SET_FULL_IO_PARAM), data_buf);

    CIPSEND(data_buf, id, 1, 0); //Στείλε την νέα κατάσταση ως απάντηση.

    return;
}
```

Η ροή της συνάρτησης είναι εξής:

1. Έλεγχος παραμέτρου και αποστολή κατάλληλης απάντησης αν δεν βρέθηκε.
2. Χτίσιμο **HTTP HEADER 200** (επιτυχής απάντηση).
3. Αρχικοποίηση JSON συμβολοσειράς που θα στείλουμε ως απάντηση.
4. Επεξεργασία JSON συμβολοσειράς και ενημέρωση των καταστάσεων των I/O Πορτών του μικροελεγκτή.
5. Αποστολή νέων καταστάσεων **(DDR, PORT, PIN)** με JSON συμβολοσειρά, ως απάντηση.

Την επεξεργασία της JSON συμβολοσειράς, την αναλαμβάνει η συνάρτηση **json_to_io()**. Αυτή η συνάρτηση αποκωδικοποιεί μία JSON συμβολοσειρά που περιέχει καταστάσεις Πορτών, σε δομή **JSON_STRUCT8** και ύστερα ενημερώνει τις Πόρτες με τις νέες καταστάσεις. Ταυτόχρονα, «χτίζει» και την JSON συμβολοσειρά που θα στείλουμε ως απάντηση, προσθέτοντας σε αυτήν τις νέες καταστάσεις που άλλαξαν (DDR, PORT), αλλά και της κατάστασης PIN για όλες τις Πόρτες, ανεξαρτήτως μεταβολής τους.

Η δομή (struct) **JSON_STRUCT8** είναι μία βοηθητική δομή για να αποθηκεύουμε τιμές και ονόματα καταστάσεων για I/O Πόρτες του Μικροελεγκτή. Παραθέτουμε τον ορισμό της, μαζί με σχόλια για περαιτέρω εξήγηση.

```
/*
    Struct για JSON αντικείμενα με 8-bit τιμές.

    Περιέχει έναν μέγιστο αριθμό MAX_JSON_PROP_NUM ιδιοτήτων (properties), καθεμία με ένα όνομα με μέγιστο αριθμό χαρακτήρων
    MAX_JSON_PROP_LEN (συμπεριλαμβανομένου και του τερματικού χαρακτήρα - null char).
    Κάθε ιδιότητα περιέχει έναν πίνακα με μέγιστο αριθμό 8-bit τιμών MAX_JSON_ARRAY_LEN.

    Κάθε τιμή των πινάκων αντιστοιχεί σε μία σημαία "non-empty" όπου σημαίνει αν είναι κενή ή όχι.
*/
typedef struct JSON_STRUCT8
{
    char props[MAX_JSON_PROP_NUM][MAX_JSON_PROP_LEN];       //Πίνακες με τα ονόματα των ιδιοτήτων.
    uint8_t values[MAX_JSON_PROP_NUM][MAX_JSON_ARRAY_LEN]; //Πίνακες με τις 8-bit τιμές των ιδιοτήτων.
    uint8_t used[MAX_JSON_PROP_NUM][MAX_JSON_ARRAY_LEN];    //Πίνακες σημαίων "non-empty" για τις 8-bit τιμές των ιδιοτήτων.
    uint8_t count;                                          //Συνολικός αριθμός ιδιοτήτων.

} JSON_STRUCT8;
```

Με λίγα λόγια, η struct **JSON_STRUCT8** αποθηκεύει τιμές για οποιαδήποτε κατάσταση/καταχωρητή **(DDR, PORT, PIN)** I/O Πόρτας του μικροελεγκτή. Οι τιμές που αποθηκεύει είναι μεγέθους 8-bit, εφόσον αφορούν τιμές 8-bit καταχωρητών. Αν κάποια Πόρτα δεν αλλάζει καθόλου, δηλαδή έχει τιμή την κενή συμβολοσειρά **“”**, τότε ενεργοποιούμε και την αντίστοιχη σημαία στον πίνακα **used**.

```
void json_to_io(const char *encoded_json, char *data_buf)
{
    /*
        Αυτή η συνάρτηση αποκωδικοποιεί JSON συμβολοσειρά που περιέχει καταστάσεις Πορτών
        και ενημερώνει τις αντίστοιχες Πόρτες του μικροελεγκτή.
    */

    JSON_STRUCT8 json_struct;

    init_json_struct8(&json_struct, 0); //Αρχικοποίηση struct.
    json_to_struct8(encoded_json, &json_struct, 1); //Μετατροπή JSON string σε κατάλληλη struct.

    uint8_t i, j;

    //Αν βρέθηκαν τουλάχιστον δύο ιδιότητες, συνέχισε.
    if(json_struct.count >= 2)
    {
        //Είναι τα ονόματα των ιδιοτήτων αυτά που θέλουμε;
        if(!strcasecmp_P(json_struct.props[0], IO_DDR) && !strcasecmp_P(json_struct.props[1], IO_PORT))
        {
            char *json = data_buf + strlen(data_buf); //JSON συμβολοσειρά που θα στείλουμε ως απάντηση.

            //Έκεινα να φτιάχνεις το string.
            json[0] = '{';

            //Εισαγωγή δύο (2) ιδιοτήτων (DDR, PORT).
            for(i = 0; i < 2; i++)
            {
                sprintf_P(json + strlen(json), JSON_PROP_ARRAY, json_struct.props[i]); //Εισαγωγή JSON ιδιότητας.

                for(j = 0; j < 4; j++)
                {
                    //Εισαγωγή κόμματος, εκτός όταν είμαστε στην τελευταία τιμή.
                    if(j < 4 - 1)
                        strcat(json, ",");

                    //Αν η τιμή δεν είναι άδεια, πάρτην.
                    if(json_struct.used[i][j])
                    {
                        uint8_t io_num = j;
                        uint8_t *value = json_struct.values[i] + j;

                        //Αν δεν υποστηρίζουμε την Πόρτα Α, ξεκίνα από την επόμενη.
                        if(CHIP != 32)
                            io_num++;

                        //Λήψη κατάστασης DDR ή PORT.
                        switch(io_num)
                        {
                            case 0:
                                if(i == 0)
                                {
                                    DDRA = *value;
                                    *value = DDRA;
                                }
                                else if(i == 1)
                                {
                                    PORTA = *value;
                                    *value = PORTA;
                                }
                                break;

                            case 1:
                                if(i == 0)
                                {
                                    DDRB = *value;
                                    *value = DDRB;
                                }
                                else if(i == 1)
                                {
                                    PORTB = *value;
                                    *value = PORTB;
                                }
                                break;
```

```
                    case 2:
                        if(i == 0)
                        {
                            DDRC = *value;
                            *value = DDRC;
                        }
                        else if(i == 1)
                        {
                            PORTC = *value;
                            *value = PORTC;
                        }
                        break;

                    case 3:
                        if(i == 0)
                        {
                            DDRD = *value;
                            *value = DDRD;
                        }
                        else if(i == 1)
                        {
                            PORTD = *value;
                            *value = PORTD;
                        }
                        break;

                    default:
                        break;
                }

                char num[3]; //8-bit τιμές μόνο.
                sprintf(num, "%X", *value); //Μετατροπή τιμής σε string.

                //Εισαγωγή τιμής στο JSON string.
                sprintf_P(json + strlen(json), JSON_PROP_VAL, num);
            }
            else //Αλλιώς, βάλε κενό string.
                strcat(json, "\"\"");
        }

        strcat(json, "]"); //Τέλος πίνακα τιμών ιδιότητας.

        //Εισαγωγή κόμματος, εκτός όταν είμαστε στην τελευταία ιδιότητα.
        if(i < 2 - 1)
            strcat(json, ",");
    }

    //Θα προσθέσουμε όμως και την κατάσταση PIN.
    strcat(json, ",");

    //Εισαγωγή κατάστασης PIN για όλες τις Πόρτες, ανεξαρτήτως αν τις αλλάξαμε ή οχι.
    if(CHIP == 32)
        sprintf_P(json + strlen(json), GET_PIN_JSON, PINA, PINB, PINC, PIND);
    else
        sprintf_P(json + strlen(json), GET_PIN_JSON, PINB, PINC, PIND);

    json[strlen(json)] = '}'; //Τέλος JSON string.
  }
}

return;
}
```

Την μετατροπή της JSON συμβολοσειράς σε δομή **JSON_STRUCT8** την αναλαμβάνει η συνάρτηση **json_to_struct8()**.

Ουσιαστικά, αυτή η συνάρτηση επεξεργάζεται την συμβολοσειρά χαρακτήρα-χαρακτήρα και ελέγχει αν η σύνταξη της είναι σωστή. Στο σύστημα μας, αυτή η συνάρτηση μπορεί να δεχτεί JSON συμβολοσειρές με ιδιότητες που περιέχουν πίνακες τιμών, αφού ασχολούμαστε με καταστάσεις I/O Πορτών.

Ο χρήστης στο δικό του σύστημα μπορεί να προσθέσει επιπλέον λειτουργία στην συνάρτηση ώστε να αποδέχεται μία JSON συμβολοσειρά με όλους τους διάφορους συνδυασμούς τιμών και ιδιοτήτων.

Προς το παρόν, στο σύστημα μας, υλοποιούμε την βασική JSON σύνταξη, όπου μας είναι αρκετή. Υπενθυμίζουμε ότι και το μέγεθος του κώδικα μας, πρέπει να είναι μικρό για να έχουμε αρκετό χώρο και στην μνήμη FLASH. Άρα, βλέπουμε πάλι τι είδους περιορισμούς έχουμε να αντιμετωπίσουμε σε αυτοματισμούς ενσωματωμένων συστημάτων.

Επίσης, για την αρχικοποίηση της **JSON_STRUCT8** χρησιμοποιούμε την συνάρτηση **init_json_struct8()**. Δεν χρειάζεται περαιτέρω ανάλυση της, καθώς τα σχόλια στον κώδικα είναι αρκετά.

```
void init_json_struct8(JSON_STRUCT8 *json_struct, uint8_t keep_props)
{
    uint8_t i, j;

    for(i = 0; i < MAX_JSON_PROP_NUM; i++)
    {
        //Διέγραψε τα ονόματα, αν πρέπει.
        if(!keep_props)
        {
            for(j = 0; j < MAX_JSON_PROP_LEN; j++)
                json_struct->props[i][j] = 0;
        }

        //Μηδενισμός υπόλοιπων μεταβλητών.
        for(j = 0; j < MAX_JSON_ARRAY_LEN; j++)
        {
            json_struct->values[i][j] = 0;
            json_struct->used[i][j] = 0;
        }
    }
    json_struct->count = 0;

    return;
}
```

Για την **json_to_struct8()**, δεν θα κάνουμε ανάλυση γιατί πρόκειται για καθαρά επεξεργασία συμβολοσειράς και τα σχόλια είναι αρκετά για την κατανόηση της.

```
void json_to_struct8(const char *encoded_json, JSON_STRUCT8 *json_struct, uint8_t get_prop_names)
{
    //Αυτή η συνάρτηση μετατρέπει μία JSON συμβολοσειρά σε δομή JSON_STRUCT8.

    uint8_t i, p = 0, is_used;
    char brace;
    const char *prop, *end, *array, *array_end;
    char *json = encoded_json;

    json = decodeURIComponent(encoded_json);

    if(json && json[0] == '{')
    {
        prop = json + 1;

        while(1)
        {
            if(p >= MAX_JSON_PROP_NUM)
                break; //Υπερχείλιση αριθμού ιδιοτήτων.

            if(prop[0] == '"')
            {
                i = 0;
                end = strchr(prop + 1, '"');

                if(end)
                {
                    uint8_t len = end - prop; //Μέγεθος ονόματος ιδιότητας.

                    if(len <= MAX_JSON_PROP_LEN)
                    {
                        //Θέλουμε να αντιγράψουμε τα ονόματα ιδιοτήτων;
                        if(get_prop_names)
                            strncpy(json_struct->props[p], prop + 1, len - 1);

                        if(prop[len] && prop[len + 1] == ':')
                        {
                            brace = prop[len + 2];

                            if(brace == '[')
                            {
                                array = prop + len + 3;

                                if(array[0] && array[0] != ']')
                                {
                                    array_end = strchr(array, ']'); //Βρες το τέλος του πίνακα τιμών.

                                    //Επεξεργασία τιμών ιδιότητας.
                                    while(1)
                                    {
                                        is_used = 1;

                                        if(i >= MAX_JSON_ARRAY_LEN)
                                            break; //Υπερχείλιση πίνακα τιμών.

                                        //Είναι συμβολοσειρά η τιμή;
                                        if(array[0] == '"')
                                        {
                                            //Είναι κενή συμβολοσειρά;
                                            if(array[1] == '"')
                                            {
                                                //Επόμενη τιμή ή τέλος πίνακα.
                                                array++;

                                                json_struct->used[p][i] = 0; //Η τιμή δεν χρησιμοποιείται.
                                                is_used = 0;
                                            }
                                            else if(array[1]) //Μη κενή συμβολοσειρά.
                                            {
                                                end = strchr(array + 1, '"'); //Βρες το τέλος της.
```

```
                                    //Αν δεν βρέθηκε τέλος, προχώρα στην επόμενη ιδιότητα.
                                    if(!end)
                                        break;
                                }
                            }
                            else if(!isdigit(array[0])) //Αν δεν είναι ούτε αριθμός, προχώρα στην επόμενη ιδιότητα.
                                break;

                            end = strchr(array + 1, ','); //Που τελειώνει η τιμή;

                            //Αν δεν βρέθηκε τέλος, έλεγξε τον πίνακα.
                            if(!end)
                            {
                                end = strchr(array + 1, ']');

                                //Αν δεν είναι έγκυρος πίνακας, προχώρα στην επόμενη ιδιότητα.
                                if(!end)
                                    break;
                            }
                            else if(end > array_end) //Αν ήταν η τελευταία τιμή, βγες από τον πίνακα.
                                end = array_end;

                            //Αν η τιμή χρησιμοποιείται, συνέχισε.
                            if(is_used)
                            {
                                //Μετατροπή δεκαεξαδικού string σε αριθμό.
                                int num = strtol(array + 1, NULL, 16);

                                //Εισαγωγή αριθμού στην struct.
                                json_struct->values[p][i] = num;
                            }
                            json_struct->used[p][i] = is_used; //Ενημέρωση σημαίας.

                            //Επόμενη τιμή.
                            array = end + 1;
                            i++;

                            //Φτάσαμε στο τέλος του πίνακα;
                            if(end[0] == ']')
                            {
                                end++;
                                break;
                            }
                        }
                    }
                    else
                        end = array;

                    p++;

                    if(end[0] == ',')
                        prop = end + 1; //Επόμενη ιδιότητα.
                    else if(end[0] == '}') //Τέλος JSON;
                        break;
                    else //Μη έγκυρο JSON.
                        break;
                }
                else
                    break; //Χρησιμοποιούμε μόνο πίνακες ώς τιμές.
            }
            else
                break; //Μη έγκυρη ιδιότητα.
        }
        else
            break; //Υπερχείλιση ονόματος ιδιότητας.
    }
    else
        break; //Μη έγκυρο όνομα ιδιότητας.
}
else
    break; //Μη έγκυρο JSON.
}
}

json_struct->count = p; //Ενημέρωση μετρητή ιδιοτήτων.

//Ελευθέρωση χώρου, αν πρέπει.
if(json)
    free(json);

return;
}
```

Θα παρατηρήσετε όμως ότι καλούμε την συνάρτηση **decodeURIComponent()**. Αυτή η συνάρτηση αποκωδικοποιεί συμβολοσειρά με κωδικοποιημένους **URI** παραμέτρους.

Ουσιαστικά, όταν στέλνουμε **HTTP GET** requests, συνήθως, οι τιμές των παραμέτρων πρέπει να είναι κωδικοποιημένες για να μην υπάρξουν τυχόν σφάλματα, κατά την επεξεργασία τους από τον server. Π.χ. αν στην τιμή υπάρχει ο χαρακτήρας **'&'**, ο server μπορεί να θεωρήσει ότι πρόκειται για την επόμενη παράμετρο ενώ στην πραγματικότητα να είναι μία απλή συμβολοσειρά.

Γι' αυτό, κωδικοποιούμε όλες τις τιμές, με βάση την κωδικοποίηση **UTF-8**.
Π.χ. η παράμετρος **"/?test=1 & 2"** θα κωδικοποιηθεί σε **"/?test =1%20%26%202"**

Όπως παρατηρείτε, οι χαρακτήρες κενού και '&' μετατράπηκαν σε ομάδες με δεκαεξαδικούς αριθμούς. Οι αριθμοί αυτοί αντιστοιχούν στην UTF-8 κωδικοποίηση των χαρακτήρων.

Στην δικιά μας συνάρτηση, η κωδικοποίηση γίνεται μόνο για τους πρώτους 256 **ASCII** χαρακτήρες, όπου περιέχεται το αγγλικό αλφάβητο, οι χαρακτήρες αριθμών και τα βασικά σύμβολα (!, @, # κλπ.).

Όπως και στις προηγούμενες συναρτήσεις, δεν θα γίνει περαιτέρω ανάλυση της, καθώς είναι καθαρά τεχνικής φύσεως και τα σχόλια είναι αρκετά για την κατανόηση της.

```
char *decodeURIComponent(const char *str)
{
    //Αυτή η συνάρτηση αποκωδικοποιεί συμβολοσειρά με κωδικοποιημένους URI παραμέτρους.

    char *new_string = 0;

    if(str)
    {
        int len = strlen(str);

        if(len && len < MAX_URL_SIZE)
        {
            char *temp, *group = strchr(str, '%'); //Βρες κωδικοποιημένη ομάδα.
            uint8_t group_num = 1; //Μετρητής αριθμού ομάδων.

            if(group)
            {
                temp = group;

                //Βρες πόσες ομάδες υπάρχουν.
                while((temp = strchr(temp + 1, '%')))
                    group_num++;

                uint8_t encoded_char, new_len = 0;
                char *prev_group = str, encoded_value[3] = {0};

                //Κατανάλωση χώρου για το αποκωδικοποιημένο string.
                new_string = calloc(len - group_num * 2, sizeof(char));
```

```
            //Επεξεργασία συμβολοσειράς.
            do
            {
                //Αποκωδικοποίηση ομάδας.
                if(strlen(group + 1) > 1)
                {
                    strncpy(encoded_value, group + 1, 2); //Αντιγραφή μόνο δύο (2) δεκαεξαδικών χαρακτήρων.
                    encoded_char = strtol(encoded_value, NULL, 16); //Μετατροπή δεκαεξαδικού string σε αριθμό.

                    //Αντιγραφή προηγούμενων χαρακτήρων.
                    strncpy(new_string + new_len, prev_group, group - prev_group);
                    new_len += group - prev_group;

                    //Αποθήκευση αποκωδικοποιημένου χαρακτήρα.
                    if(encoded_char)
                        new_string[new_len++] = encoded_char;

                    //Επόμενη ομάδα.
                    prev_group =  group + 3;
                    group = strchr(prev_group, '%');
                }
                else
                    break;

            } while(group);

            //Αντιγραφή υπόλοιπης συμβολοσειράς.
            if(prev_group)
                strncpy(new_string + new_len, prev_group, strlen(prev_group));
        }
    }
}

//Επιστροφή αποκωδικοποιημένου string.
//Μην ξεχάσετε να ελευθερώσετε τον χώρο μετά.
return new_string;
}
```

Όπως καταλάβατε, για να μετατρέψουμε μία JSON συμβολοσειρά σε μορφή που να μπορούμε να επεξεργαστούμε, περάσαμε από τέσσερις (4) συναρτήσεις. Θα μπορούσαμε να μειώσουμε τον αριθμό στις δύο (2) συναρτήσεις αλλά επιλέξαμε αυτή την μορφή γιατί δίνει στον χρήστη την δυνατότητα προσαρμογής των συναρτήσεων στις ανάγκες του.

Καθεμία απ' τις συναρτήσεις για την επεξεργασία JSON συμβολοσειράς, έχει σχεδιαστεί και υλοποιηθεί με τέτοιο τρόπο ώστε να μπορεί να προσαρμοστεί εύκολα στις ανάγκες του χρήστη. Βεβαίως, ανάλογα με το σύστημα, ο χρήστης μπορεί να επιθυμήσει να υλοποιήσει τις δικές του συναρτήσεις, χρησιμοποιώντας ως σημείο αναφοράς τις συναρτήσεις του συστήματος μας, όπου αυτή είναι και η προτεινόμενη λύση.

Επόμενη συνάρτηση που θα αναλύσουμε είναι η **toggle_IO()**. Ουσιαστικά, αυτή η συνάρτηση μεταβάλλει την λογική κατάσταση (0/1) ενός ακροδέκτη (PIN) του μικροελεγκτή.

Η συνάρτηση επιστρέφει ως απάντηση τις καταστάσεις PORT και PIN της πόρτας του ακροδέκτη που επιλέξαμε, ως JSON συμβολοσειρά.

```
void toggle_IO(char *req, uint8_t id)
{
    //Αυτό το αίτημα μεταβάλλει την λογική κατάσταση (0/1) ενός ακροδέκτη του μικροελεγκτή.

    char *data_buf = in_buf;

    char *io, *state;
    uint8_t io_num[2] = {0}, pin_num = 0;

    //get /?toggle=a1
    //get /?toggle=a1&state=0

    //Βρες παράμετρο ακροδέκτη.
    io = strchr(req, '=') + 1;

    //Έλεγξε και βρες πόρτα και αριθμό ακροδέκτη.
    if(isalpha(io[0]) && isdigit(io[1])) //Π.χ. A1
    {
        io_num[0] = toupper(io[0]); //Αποθήκευση πόρτας.
        pin_num = io[1] - '0'; //Μετατροπή χαρακτήρα αριθμού ακροδέκτη σε αριθμό.
    }
    else
        io = 0;

    if(io)
    {
        //Αναζήτηση και έλεγχος παραμέτρου νέας κατάστασης.
        if((state = strcasestr_P(io, TOGGLE_IO_PIN_REQ_PARAM_STATE)) && isdigit(state[7]))
            *state = state[7] - '0'; //Μετατροπή χαρακτήρα σε αριθμό.
    }

    //Αν κάποια από τις παραμέτρους δεν βρέθηκε ή δεν είχε έγκυρη τιμή, στείλε απάντηση HTTP 400.
    if(!io || !pin_num || io_num[0] > 'D' || (CHIP != 32 && io_num[0] == 'A'))
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Μεταβολή κατάστασης ακροδέκτη.
    switch(io_num[0])
    {
        case 'A':
            if(!state)
                PORTA ^= (1 << pin_num);
            else if(*state == 1)
                PORTA |= (1 << pin_num);
            else if(*state == 0)
                PORTA &= ~(1 << pin_num);

            io_num[0] = PORTA;
            io_num[1] = PINA;
            break;

        case 'B':
            if(!state)
                PORTB ^= (1 << pin_num);
            else if(*state == 1)
                PORTB |= (1 << pin_num);
            else if(*state == 0)
                PORTB &= ~(1 << pin_num);

            io_num[0] = PORTB;
            io_num[1] = PINB;
            break;

        case 'C':
            if(!state)
                PORTC ^= (1 << pin_num);
            else if(*state == 1)
                PORTC |= (1 << pin_num);
            else if(*state == 0)
                PORTC &= ~(1 << pin_num);

            io_num[0] = PORTC;
            io_num[1] = PINC;
            break;
```

```
        case 'D':
            if(!state)
                PORTD ^= (1 << pin_num);
            else if(*state == 1)
                PORTD |= (1 << pin_num);
            else if(*state == 0)
                PORTD &= ~(1 << pin_num);

            io_num[0] = PORTD;
            io_num[1] = PIND;
            break;

        default:
        break;
    }

    //Χτίσιμο HTTP HEADER απάντησης.
    memset(data_buf, 0, MAX_RESPONSE_STRING_SIZE);
    BUILD_HTTP_HEADER_200(data_buf, 0, APP_JSON);

    char port[5] = {0}, pin[4] = {0};
    strcpy_P(port, IO_PORT);
    strcpy_P(pin, IO_PIN);

    //Επιστροφή JSON string με καταστάσεις PORT and PIN.
    sprintf_P(data_buf + strlen(data_buf), JSON_TOGGLE_IO, port, io_num[0], pin, io_num[1]);

    CIPSEND(data_buf, id, 1, 0); //Αποστολή απάντησης.

    return;
}
```

Οι επόμενες συναρτήσεις που θα αναλύσουμε αφορούν λειτουργίες του διαχειριστή (admin) του συστήματος. Αποφασίσαμε να τις χωρίσουμε από τον χρήστη για να δείξουμε τις πιθανές εφαρμογές ενός τέτοιου συστήματος. Οι λειτουργίες αυτές, στο σύστημα μας, γίνονται με χρήση **Desktop εφαρμογής Windows**, που αναπτύχθηκε αποκλειστικά για την διπλωματική εργασία αυτή. Υπενθυμίζουμε ότι και η εφαρμογή αυτή μπορεί να παραλειφθεί και οι λειτουργίες τις να ενσωματωθούν στην ιστοσελίδα του server.

Οι συναρτήσεις αυτές είναι οι **admin_connect()**, **download_log()**, **clear_log()**, **reset_esp()** και **reset_chip()**.

Η συνάρτηση **admin_connect()** υπάρχει μόνο και μόνο για μελλοντικές επεκτάσεις του συστήματος. Το μόνο που κάνει είναι να απαντά στο αίτημα με επιτυχία. Μπορεί να παραληφθεί στην τωρινή κατάσταση του συστήματος μας.

```
void admin_connect(char *req, uint8_t id)
{
    //Αυτό το αίτημα χρησιμοποιείται να την είσοδο διαχείριστών (admin).
    char *data_buf = in_buf;
    char *admin;

    //get /?admin=1

    //Βρες παράμετρο "admin".
    admin = strcasestr_P(req, ADMIN_REQ);
```

```
    //Έλεγχος παραμέτρου.
    if(admin)
    {
        if(admin[8] != '1')
            admin = 0;
    }

    //Αν δεν βρέθηκε η παράμετρος ή δεν είναι έγκυρη, στείλε απάντηση HTTP 400.
    if(!admin)
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Αποστολή απάντησης HTTP 200.
    BUILD_HTTP_HEADER_200(data_buf, 0, TEXT_PLAIN);
    CIPSEND(data_buf, id, 1, 0);

    return;
}
```

Οι συναρτήσεις **download_log()** και **clear_log()**, αφορούν το αρχείο καταγραφής γεγονότων **(log)** του συστήματος μας. Όπως αναφέρουν και τα ονόματα τους, η **download_log()** μας επιτρέπει να κατεβάσουμε το log αρχείο και η **clear_log()** μηδενίζει/καθαρίζει το log αρχείο.

```
void download_log(char *req, uint8_t id)
{
    //Αυτό το αίτημα στέλνει το αρχείο καταγραφής (log).
    char *download;

    //get /?dlog=1

    //Βρες παράμετρο "dlog".
    download = strcasestr_P(req, DOWNLOAD_LOG_REQ);

    //Έλεγχος παραμέτρου.
    if(download)
    {
        if(download[7] != '1')
            download = 0;
    }

    //Αν δεν βρέθηκε η παράμετρος ή δεν είναι έγκυρη, στείλε απάντηση HTTP 400.
    if(!download)
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    SEND_HTTP_FILE(0, id, APP_OCTET, 1); //Αποστολή αρχείου log.

    return;
}
```

```
void clear_log(char *req, uint8_t id)
{
    //Αυτό το αίτημα "καθαρίζει" το αρχείο καταγραφής.

    char *data_buf = in_buf;
    char *clear;

    //get /?clog=1

    //Βρες παράμετρο "clog".
    clear = strcasestr_P(req, CLEAR_LOG_REQ);

    //Έλεγχος παραμέτρου.
    if(clear)
    {
        if(clear[7] != '1')
            clear = 0;
    }

    //Αν δεν βρέθηκε η παράμετρος ή δεν είναι έγκυρη, στείλε απάντηση HTTP 400.
    if(!clear)
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Καθαρισμός μνήμης EEPROM.
    eep_clear();

    //Αποστολή απάντησης HTTP 200.
    BUILD_HTTP_HEADER_200(data_buf, 0, TEXT_PLAIN);
    CIPSEND(data_buf, id, 1, 0);

    //Καταγραφή γεγονότος.
    log_event(0, EVT_LOG_CLEAR , 1);

    return;
}
```

Οι συναρτήσεις **reset_esp()** και **reset_chip()**, κάνουν επανεκκίνηση το ESP και τον μικροελεγκτή, αντίστοιχα. Για την επανεκκίνηση του ESP στέλνουμε την εντολή AT+RST.

Για την επανεκκίνηση του μικροελεγκτή χρησιμοποιούμε τον **Watchdog Timer (WDT)**. Ουσιαστικά, ο **WDT** χρησιμοποιείται για να κάνουμε επανεκκίνηση τον μικροελεγκτή όταν το πρόγραμμα έχει «κολλήσει» σε κάποιο σημείο ή γενικά δεν εκτελέστηκε σωστά.

Όταν ενεργοποιούμε τον WDT, πρέπει μέσα σε ένα συγκεκριμένο χρονικό διάστημα, που έχουμε επιλέξει, να μηδενίσουμε τον μετρητή του WDT πριν λήξει το χρονικό όριο. Αν το χρονικό όριο λήξει, ο WDT εκτελεί επανεκκίνηση στον μικροελεγκτή, θεωρώντας ότι το πρόγραμμα μας κόλλησε. Τα χρονικά όρια που μπορούμε να επιλέξουμε διαφέρουν ανάλογα με τον μικροελεγκτή.

Για μικροελεγκτές σειράς ATmega, επειδή έχουμε τάση τροφοδοσίας 3.3V, τα χρονικά όρια έχουν ένα μικρό σφάλμα.

```
void reset_esp(char *req, uint8_t id)
{
    //Αυτό το αίτημα κάνει επανεκκίνηση το ESP.

    char *data_buf = in_buf;
    char *reset;

    //get /?reset_esp=1

    //Βρες παράμετρο "reset_esp".
    reset = strcasestr_P(req, RESET_ESP_REQ);

    //Έλεγχος παραμέτρου.
    if(reset)
    {
        if(reset[12] != '1')
            reset = 0;
    }

    //Αν δεν βρέθηκε η παράμετρος ή δεν είναι έγκυρη, στείλε απάντηση HTTP 400.
    if(!reset)
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Αποστολή απάντησης HTTP 200.
    BUILD_HTTP_HEADER_200(data_buf, 0, TEXT_PLAIN);
    CIPSEND(data_buf, id, 1, 0);

    //Τερματισμός σύνδεσης πριν την επανεκκίνηση.
    CIPCLOSE(id);

    //Καταγραφή γεγονότος.
    log_event(0, EVT_RESET_ESP, 1);

    //Επανεκκίνηση ESP.
    RST();
    wait_esp();

    //Ελευθέρωσε τον χώρο που καταλαμβάνουν οι συνδέσεις.
    uint8_t i;
    for(i = 0; i < ESP_MAX_CONNECTIONS; i++)
        remove_conn(i);

    //Αρχικοποίηση ESP πάλι.
    init_esp();

    return;
}
```

```
void reset_chip(char *req, uint8_t id)
{
    //Αυτό το αίτημα κάνει επανεκκίνηση τον μικροελεγκτή.

    char *data_buf = in_buf;
    char *reset;

    //get /?reset_chip=1

    //Βρες παράμετρο "reset_chip".
    reset = strcasestr_P(req, RESET_CHIP_REQ);

    //Έλεγχος παραμέτρου.
    if(reset)
    {
        if(reset[13] != '1')
        reset = 0;
    }

    //Αν δεν βρέθηκε η παράμετρος ή δεν είναι έγκυρη, στείλε απάντηση HTTP 400.
    if(!reset)
    {
        SEND_HTTP_HEADER_400(id);
        return;
    }

    //Αποστολή απάντησης HTTP 200.
    BUILD_HTTP_HEADER_200(data_buf, 0, TEXT_PLAIN);
    CIPSEND(data_buf, id, 1, 0);

    //Τερματισμός σύνδεσης πριν την επανεκκίνηση.
    CIPCLOSE(id);

    //Ενεργοποίηση Watchdog Timer για 15ms.
    //Εφόσον δεν κάνουμε reset τον WDT, αυτός θα επανεκκινήσει τον μικροελεγκτή.
    wdt_enable(WDTO_15MS);

    while(1){} //Περίμενε μέχρι να γίνει επανεκκίνηση.
}
```

Η συνάρτηση **wdt_enable()** και η σταθερά **WDTO_15MS** είναι μέρος της εσωτερικής βιβλιοθήκης **<avr/wdt.h>.**

Παρατηρείστε ότι πριν εκτελέσουμε την επανεκκίνηση, τερματίζουμε την σύνδεση επιτυχώς γιατί δεν μπορούμε να απαντήσουμε στην σύνδεση μετά την επανεκκίνηση.

Η τελευταία συνάρτηση που απομένει από τα αιτήματα είναι η **send_temperature()**. Αυτή η συνάρτηση στέλνει ως απάντηση την τιμή θερμοκρασίας που λαμβάνουμε από τον θερμίστορ. Πρώτα όμως θα πρέπει να μετατρέψουμε την ADC τιμή που λαμβάνουμε, στην αντίστοιχη τιμή θερμοκρασίας σε βαθμούς Κελσίου.

Γι’ αυτό, πρώτα θα εξηγήσουμε πως λαμβάνουμε την ADC τιμή και πως μπορούμε να την φιλτράρουμε για να απαλείψουμε τον θόρυβο και ταυτόχρονα να έχουμε μεγαλύτερη ακρίβεια.

8.15.1 Φιλτράρισμα και λήψη ADC τιμής

Η διαδικασία ξεκινάει με την συνάρτηση **start_ADC_reading()**, οπού κάνει το φιλτράρισμα της ADC τιμής.

Ουσιαστικά, η διαδικασία είναι η εξής:

1. Διαβάζουμε X ADC τιμές από τον θερμίστορ, όπου **X** αριθμός δύναμης του 4.
2. Τις προσθέτουμε όλες μαζί και κάνουμε **Δεξιά Μετακίνηση (Right Shift)** κατά Y bits, το αποτέλεσμα, όπου Y ο αριθμός των επιπλέον bit ακρίβειας που επιθυμούμε.

Με αυτήν την διαδικασία επιτυγχάνουμε την ακρίβεια της τιμής που επιθυμούμε. Ο μικροελεγκτής **ATmega32** προσφέρει ακρίβεια **δέκα (10) bits** *(μέγιστη τιμή = 2^10 = 1024)* στην μετατροπή αναλογικού σε ψηφιακό (ADC). Στο σύστημα μας επιθυμούμε να έχουμε επιπλέον ακρίβεια δύο (2) bits, άρα σύνολο **δώδεκα (12)** bits.

Παραθέτουμε τον ορισμό του αριθμού των τιμών **(BUFFERSIZE)** που θα διαβάζουμε για την μετατροπή της τιμής του θερμίστορ σε κατάλληλη τιμή θερμοκρασίας, μαζί με τα σχόλια τα οποία που εξηγούν τις επιλογές μας.

```
//Σταθερές για την μετατροπή αναλογικού σε ψηφιακό (ADC).
/*
    μέγεθος_buffer = 4^ν, όπου ν = αριθμός επιπλέον επιθυμητών bits για την μετατροπή.

    Θέλουμε 2 επιπλέον bits, άρα ν = 2.
    10 + 2 = 12 bits σύνολο.

    Δες: Application Note AVR121, για περισσότερες πληροφορίες.
*/
#define BUFFERSIZE      16                  //4^2 = 16
#define ADC_PIN         4                   //Ακροδέκτης ADC για τον θερμίστορ.
```

Με λίγα λόγια, όσο περισσότερα bits διαθέτουμε τόσο μεγαλύτερο είναι και το εύρος πιθανών τιμών άρα και η ακρίβεια.

*Π.χ. αν διαθέτουμε ακρίβεια 10 bits και τάση αναφοράς 5V, η **αναλογική τιμή 2.5V** θα αντιστοιχεί στην **ψηφιακή τιμή 512**, καθώς:*

5 / 2.5 = 1024 / 512

Αν είχαμε όμως αναλογική τιμή **2.556V**, η αντίστοιχη ψηφιακή τιμή της θα ήταν πάλι **512.** Άρα δεν θα μπορούσαμε να διακρίνουμε μικρές μεταβολές στην τάση της τάξεως των 0.001V, γιατί πολύ απλά δεν διαθέτουμε αρκετό εύρος τιμών για να τις αντιστοιχίσουμε.

Γι' αυτό, όσο περισσότερα bits διαθέτουμε, τόσο μεγαλύτερο είναι και το εύρος τιμών, άρα μεγαλώνει και η ακρίβεια των τιμών μας.

Δεν θα αναφερθούμε περισσότερο, γιατί το η θεωρία πίσω από το **Φιλτράρισμα ADC τιμών και η Αύξηση της Ακρίβειας του**, είναι αρκετά πολύπλοκη. Γι' αυτούς που επιθυμούν να μάθουν περισσότερα, παραθέτουμε στις **Πηγές** το εγχειρίδιο **Application Note AVR121: Enhancing ADC resolution by oversampling**[5].

```
uint16_t start_ADC_reading()
{
    /*
        Δες εγχειρίδιο: Application Note AVR121: Enhancing ADC resolution by oversampling

        Με λίγα λογια:
        Διαβάζουμε πολλές τιμές από τον ADC.
        Αφού έχουμε διαβάσει έναν συγκεκριμένο αριθμό τιμών, τις προσθέτουμε όλες μαζί και κάνουμε Δεξιά Μετακίνηση (Right Shift)
        κατά δύο (2) bits το αποτέλεσμα. Με αυτήν την κίνηση επιτυγχάνουμε την ακρίβεια που επιθυμούμε.
    */

    uint8_t i; //Μετρητής τιμών ADC.
    uint16_t sum = 0; //Αριθμητικό σύνολο τιμών.

    //Διάβασε BUFFERSIZE τιμές από τον ADC και πρόσθεσε τις μαζί.
    for(i = 0; i < BUFFERSIZE; i++)
        sum += read_adc(ADC_PIN);

    return (sum >> 2); //Δεξια Μετακίνηση κατά 2 bits.
}
```

Για να διαβάσουμε μία τιμή από ένα **ADC PIN** καλούμε την συνάρτηση **read_adc()**. Αυτή η συνάρτηση είναι επιλέγει τον ακροδέκτη ADC **(ch)** από τον οποίο θα λάβουμε την τιμή και τέλος επιστρέφει την τιμή αυτή. Η διαδικασία λήψης ADC τιμής είναι μία διαδικασία που μοιάζει μεταξύ των μικροελεγκτών ίδιας σειράς. Η μόνη διαφορά μπορεί να είναι στις ονομασίες των καταχωρητών και στον αριθμό των διαθέσιμων ADC ακροδεκτών.

```
uint16_t read_adc(uint8_t ch)
{
    //Σε ποιο PIN θέλουμε να κάνουμε ADC;
    ch &= 7;
    ADMUX = (ADMUX & 0xF8) | ch;

    _delay_us(10);

    //Ξεκίνα την μετατροπή.
    ADCSRA |= (1 << ADSC);

    //Περίμενε μέχρι να τελειώσει.
    while(!(ADCSRA & (1 << ADIF)));

    //Απενεργοποίηση ADIF bit, γράφοντας ένα (1).
    ADCSRA |= (1 << ADIF);

    return ADC;
}
```

Για την εκκίνηση της διαδικασίας αυτής χρησιμοποιείται η συνάρτηση **calculate_temperature()**.

```
float calculate_temperature()
{
    double val;

    //Διάβασε την φιλτραρισμένη ADC τιμή και υπολόγισε την τάση.
    val = (start_ADC_reading() * REF_VOLT) / 4095.0;

    //Υπολόγισε την αντίσταση.
    val = (R0 * REF_VOLT) / val - R0;

    //Υπολογισμός θερμοκρασίας με την εξίσωση Steinhart-Hart.

    val = log(val);

    //T = 1 / ( a + b * ln(res) + c * ln(res) ^ 3 )
    val = 1.0 / (a + b * val + c * (val * val * val));

    val -= 273.15; //Μετατροπή Kelvin σε βαθμούς Κελσίου.

    //Θέλουμε δεκαδική ακρίβεια ενός ψηφίου.
    val *= 10.0;
    val = (int16_t)val / 10.0;

    return (float)val;
}
```

Έχουμε ήδη αναφέρει τον τρόπο με τον οποίο θα μετατρέψουμε την φιλτραρισμένη ADC τιμή σε βαθμούς Κελσίου (^oC), με την χρήση της εξίσωσης **Steinhart-Hart**.

Θυμηθείτε ότι έχουμε συνδέσει τους δύο ακροδέκτες του θερμίστορ σε **ADC PIN** και σε **PIN παροχής τάσεως (VCC)**. Το θερμίστορ μας στέλνει την τάση του και εμείς πρέπει να μετατρέψουμε την τιμή αυτή, σε βαθμούς Κελσίου. Είχαμε αναφέρει ότι ο θερμίστορ είναι τύπου **NTC (Negative Temperature Coefficient)**, που σημαίνει ότι η αντίσταση του μειώνεται όσο αυξάνεται η θερμοκρασία.

Πρώτα διαβάζουμε την φιλτραρισμένη τιμή από τον **ADC**, καλώντας την συνάρτηση **start_ADC_reading()**, όπου αναλύσαμε προηγουμένως. Στη συνέχεια, μετατρέπουμε την τιμή σε **Volts** και ύστερα υπολογίζουμε την **αντίσταση**.

Χρησιμοποιώντας την εξίσωση **Steinhart-Hart** υπολογίζουμε την θερμοκρασία. Μετά κάνουμε μία απλή μετατροπή από βαθμούς **Kelvin** σε βαθμούς **Κελσίου**. Τέλος, κάνοντας κάνουμε τις απαραίτητες πράξεις, για έχουμε την ακρίβεια που επιθυμούμε.

Και τέλος η συνάρτηση **send_temperature()** η οποία στέλνει την θερμοκρασία ως απάντηση.

```
void send_temperature(uint8_t id)
{
    //Αυτό το αίτημα στέλνει την τιμή θερμοκρασίας που λαμβάνουμε από τον θερμίστορ.

    char *data = in_buf;
    char *temp_buf = calloc(8, sizeof(char));

    //Χτίσιμο HTTP HEADER απάντησης.
    memset(data, 0, MAX_RESPONSE_STRING_SIZE);
    BUILD_HTTP_HEADER_200(data, 0, TYPE_DEFAULT);

    float_to_string(calculate_temperature(), temp_buf); //Μετατροπή θερμοκρασίας σε string.
    strcat(data, temp_buf); //Εισαγωγή τιμής θερμοκρασίας στην απάντηση.

    CIPSEND(data, id, 1, 0); //Αποστολή απάντησης.
    free(temp_buf);

    return;
}
```

Η συνάρτηση **float_to_string()** είναι μια απλή μετατροπή δεκαδικού αριθμού σε συμβολοσειρά.

```
void float_to_string(float value, char *str)
{
    int16_t int_val = value;

    //Μετατροπή δεκαδικής τιμής σε συμβολοσειρά σπάζοντας την σε κομμάτια (ακέραιο + δεκαδικό).
    //Κρατάμε μέχρι ένα δεδαδικο ψηφίο.
    sprintf(str, "%d.%d", int_val, (uint8_t)(fabs(value - int_val) * 10.0));

    return;
}
```

8.16 Καταγραφή γεγονότων (log) και συγχρονισμός ρολογιού

Για την καταγραφή των γεγονότων και το συγχρονισμού του ρολογιού μας, χρησιμοποιούμε τις συναρτήσεις **log_event()**, **get_local_time()** και **format_datetime()**.

Η διαδικασία ξεκινάει με την συνάρτηση **log_event()**, οπού δημιουργεί μια συμβολοσειρά με την τωρινή ώρα και το μήνυμα που δώσαμε ως παράμετρο. Αφού ενώσει την ώρα με το μήνυμα, αποθηκεύει την συμβολοσειρά στην μνήμη **EEPROM**, όπου την χρησιμοποιούμε για να αποθηκεύουμε τα γεγονότα μας και ουσιαστικά είναι το αρχείο καταγραφής μας (log).

Η μνήμη **EEPROM** είναι ένας ειδικός τύπος μνήμης καθώς έχει την δυνατότητα να μην χάνει τα δεδομένα της ακόμα και όταν ο μικροελεγκτής δεν τροφοδοτείται με ρεύμα. Με λίγα λόγια, είναι σαν ένας σκληρός δίσκος, αλλά με **εξαιρετικά μικρή χωρητικότητα**.

Ο **ATmega32** παρέχει μνήμη EEPROM με χωρητικότητα **1KB** (1024 bytes). Όπως καταλαβαίνετε, το αρχείο καταγραφής θα είναι αρκετά μικρό και θα πρέπει να αντικαθιστούμε προηγούμενα γεγονότα, για να αποθηκεύσουμε νέα, όταν γεμίσει η μνήμη.

```
void log_event(char *ev, const char ev_P[], uint8_t logTime)
{
    uint8_t len;

    //Υπολογισμός μεγέθους γεγονότος, χρησιμοποιώντας την κατάλληλη συνάρτηση.
    if(ev)
        len = strlen(ev);
    else if(ev_P)
        len = strlen_P(ev_P);
    else
        len = 0;

    if(len)
    {
        char *final_log;
        char time_stamp[25] = {0}; //Ακριβές μέγεθος.

        //Αν καταγράφουμε με ώρα, πάρτην.
        if(logTime)
            get_local_time(time_stamp);

        //Καταγραφή ώρας.
        final_log = calloc(strlen(time_stamp) + len + 1, sizeof(char));
        strcpy(final_log, time_stamp);

        //Ενωση μηνύματος με την ώρα.
        if(ev)
            strcat(final_log, ev);
        else
            strcat_P(final_log, ev_P);

        //Αποθήκευση γεγονότος στην μνήμη EEPROM.
        eep_store_string(final_log);

        //Ελευθέρωσε τον χώρο.
        free(final_log);
    }

    return;
}
```

Η ροή της συνάρτησης είναι η εξής:

1. Υπολογισμός μεγέθους γεγονότος.
2. Λήψη τωρινής ημ/νίας και ώρας, αν χρειάζεται.
3. Ένωση συμβολοσειρών ώρας και γεγονότος.
4. Αποθήκευση τελικού γεγονότος στην μνήμη EEPROM.

Η συνάρτηση **get_local_time()** χρησιμοποιείται για λάβουμε την τωρινή ημ/νία και ώρα, ως συμβολοσειρά. Επίσης, σε αυτήν την συνάρτηση γίνεται και ο συγχρονισμός του ρολογιού, όταν χρειάζεται.

```
void get_local_time(char *final_date)
{
    uint8_t i;
    char *start;

    //Συγχρονισμός ρολογιού, αν πρέπει και μπορούμε.
    if(sync_time && !conn_count)
    {
        CIPSNTPTIME(); //Λήψη ώρας από το ESP.

        //Αναζήτηση ώρας στους buffer.
        if(((start = strstr_P(main_buf.buffer + main_buf.last_cmd_start, AT_CIPSNTPTIME_TIME)) && (i = 1)) ||
           ((start = strstr_P(extra_buf.buffer + extra_buf.last_cmd_start, AT_CIPSNTPTIME_TIME)) && (i = 2)))
        {
            //Αν δεν βρεθεί το τέλος της γραμμής, τότε σημαίνει ότι είναι σπασμένο στη μέση. Βρες το υπόλοιπο.
            if(!strstr(start, ESP_DEFAULT_APPENDER))
            {
                if(i == 1)
                    start = strstr_P(extra_buf.buffer + extra_buf.last_cmd_start, AT_CIPSNTPTIME_TIME);
                else
                    start = strstr_P(main_buf.buffer + main_buf.last_cmd_start, AT_CIPSNTPTIME_TIME);
            }

            start += strlen_P(AT_CIPSNTPTIME_TIME) + 4; //Παράλειψη μέρας.
            strncpy(final_date, start, 20); //Αντιγραφή γραμμής.
        }
        else //Αλλιώς, είχαμε αποτυχία.
            strcpy_P(final_date, TIMESTAMP_ERROR);
    }

    //Ενημέρωση ρολογιού και αλλαγή μορφής.
    format_datetime(final_date);
    sync_time = 0;

    return;
}
```

Για να ενημερώσουμε το ρολόι μας με την νέα ημ/νία και ώρα και να αλλάξουμε την μορφή της συμβολοσειράς, καλούμε την συνάρτηση **format_datetime()**.

```
void format_datetime(char *date_str)
{
    //Αν συγχρονίσαμε το ρολόι, ενημέρωση την struct.
    if(sync_time)
    {
        //Αν ο συγχρονισμός δεν ήταν επιτυχής, επέστρεψε.
        if(date_str[0] == '[')
            return;

        //Παράδειγμα τελικής μορφής  = "Aug 04 14:48:05 2016"

        char mon[4] = {0};

        //Διάβασε την ώρα.
        sscanf_P(date_str, SNTP_FORMAT, mon, &cur_time.day, &cur_time.hour, &cur_time.min, &cur_time.sec, &cur_time.year);
        cur_time.mon = month_to_num(mon);
    }

    //Αποθήκευση τελικού string.
    sprintf_P(date_str, LOG_TIMESTAMP_FORMAT, cur_time.day, cur_time.mon, cur_time.year, cur_time.hour, cur_time.min, cur_time.sec);

    return;
}
```

Η μορφή της ημ/νίας και ώρας ορίζεται στην σταθερά **LOG_TIMESTAMP_FORMAT**, που έχουμε ορίσει ως εξής:

```
static const char LOG_TIMESTAMP_FORMAT[] PROGMEM            = "[%02d/%02d/%02d - %02d:%02d:%02d] ";
```

Η συνάρτηση **month_to_num** ορίζεται στην βοηθητική βιβλιοθήκη **«helper.h»** και ο κώδικας της βρίσκεται στο αντίστοιχο **«helper.c»** αρχείο. Δέχεται ως παράμετρο μία συμβολοσειρά με έναν μήνα και επιστρέφει τον αριθμό που αντιστοιχεί στον μήνα αυτό (Ιαν = 1, Φεβ = 2, …. Δεκ = 12).

```
#ifndef HELPER_H_
#define HELPER_H_

#include <avr/io.h>

uint8_t month_to_num(char *month);
#endif /* HELPER_H_ */
```

Κώδικας συνάρτησης:

```
 #include "helper.h"

//Βοηθητική συνάρτηση για μετατροπή μήνα στον αντίστοιχο αριθμό.
uint8_t month_to_num(char *month)
{
    switch(month[0])
    {
        case 'J':
            if(month[1] == 'a')
                return 1;                   //Jan
            else if(month[2] == 'n')
                return 6;                   //Jun
            else
                return 7;                   //Jul
        case 'F':
            return 2;                       //Feb
        case 'M':
            if(month[2] == 'r')
                return 3;                   //Mar
            else
                return 5;                   //May
        case 'A':
            if(month[1] == 'p')
                return 4;                   //Apr
            else
                return 8;                   //Aug
        case 'S':
            return 9;                       //Sep
        case 'O':
            return 10;                      //Oct
        case 'N':
            return 11;                      //Nov
        case 'D':
            return 12;                      //Dec
        default:
            return 1;                       //Default είναι Jan.
    }
}
```

Οι συναρτήσεις για την αποθήκευση και εκκαθάριση της EEPROM είναι οι **eep_store_string()** και **eep_clear()**. Η ενημέρωση της EEPROM είναι μια αργή διαδικασία καθώς είναι ένα από τα μειονεκτήματα της. Γι' αυτό, πρέπει να προσέχουμε όταν καταγράφουμε γεγονότα κατά την εκτέλεση ρουτίνων ISR, να μην αποθηκεύουμε μεγάλα μηνύματα.

```
void eep_clear()
{
    uint16_t i;

    for(i = 0; i < 1000; i++)
    {
        eeprom_busy_wait();
        eeprom_update_byte(buffer + i, 0xFF);
    }
    buf_pos = 0;

    return;
}
void eep_store_string(const char *str)
{
    uint16_t len = strlen(str);
    uint16_t temp;

    if(!str || !len)
        return;

    //Επανεκκίνηση θέσης.
    if(buf_pos + len > 1000)
        buf_pos = 0;

    temp = buf_pos;

    //Επόμενη θέση.
    buf_pos += len;

    //Αποθήκευση string στην μνήμη EEPROM.
    eeprom_busy_wait();
    eeprom_update_block(str, buffer + temp, len);

    return;
}
```

8.17 Λοιπές εσωτερικές βιβλιοθήκες

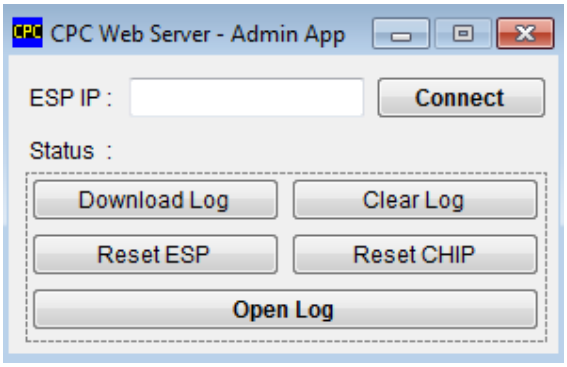
Παραθέτουμε στα Παραρτήματα τις βοηθητικές βιβλιοθήκες μας **settings.h** και **events.h**, που περιέχουν τις διάφορες σταθερές που χρησιμοποιούμε στο πρόγραμμα μας, καθώς και τα αρχεία ιστοσελίδας του server μας: **index_html.h**, **custom_js.h** και **style_css.h**. Τα αρχεία αυτά περιέχουν ως συμβολοσειρές τον **minified** κώδικα των αντίστοιχων αρχείων (.html, .js, .css), ώστε να καταλαμβάνουν τον μικρότερο δυνατό χώρο στην FLASH.

Παρουσίαση όλων των κανονικών αρχείων του server μας θα γίνει σε επόμενο κεφάλαιο.

8.18 Κώδικας Windows εφαρμογής για λειτουργίες διαχειριστή (admin)

```
Imports System.IO
Imports System.Net
Imports System.Text.RegularExpressions
Public Class MainForm

    Dim successText As String = ""
    Dim log_path As String = IO.Directory.GetParent(Application.ExecutablePath).FullName

    Private Sub btnConnect_Click(sender As Object, e As EventArgs) Handles btnConnect.Click

        Dim webClient As New WebClient
        Dim ip As String = txtIP.Text

        lblRes.Visible = True

        Dim regexIP As Regex = New Regex("^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$")

        'Σύνδεση με ESP.
        If regexIP.Match(ip).Success Then

            status("Connecting...", 1)
            actionPanel.Enabled = False
            My.Settings.esp_ip = ip
            My.Settings.Save()

            successText = "Connected."
            executeRequest(webClient, "/?admin=1", "")
        Else
            status("Invalid IP address.", 0)
        End If

    End Sub

    Private Sub btnDLog_Click(sender As Object, e As EventArgs) Handles btnDLog.Click

        Dim webClient As New WebClient

        successText = "Download complete."
        executeRequest(webClient, "/?dlog=1", "file")

    End Sub

    Private Sub btnCLog_Click(sender As Object, e As EventArgs) Handles btnCLog.Click

        Dim webClient As New WebClient

        successText = "Log cleared."
        executeRequest(webClient, "/?clog=1", "")

    End Sub

    Private Sub btnResetESP_Click(sender As Object, e As EventArgs) Handles btnResetESP.Click

        Dim webClient As New WebClient

        successText = "ESP successfully reset."
        executeRequest(webClient, "/?reset_esp=1", "")
        actionPanel.Enabled = False

    End Sub

    Private Sub btnResetChip_Click(sender As Object, e As EventArgs) Handles btnResetChip.Click

        Dim webClient As New WebClient

        successText = "CHIP successfully reset."
        executeRequest(webClient, "/?reset_chip=1", "")
        actionPanel.Enabled = False

    End Sub

    Private Sub btnOpenLog_Click(sender As Object, e As EventArgs) Handles btnOpenLog.Click

        'Άνοιγμα log αρχείου.
        If System.IO.File.Exists(log_path + "\cpc_log.txt") = True Then
            Shell("Notepad.exe " + log_path + "\cpc_log.txt", vbNormalFocus)
        Else
            MsgBox("File does not exist! Please download the log again.")
            btnOpenLog.Enabled = False
        End If

    End Sub
```

```
Sub executeRequest(ByVal webClient As WebClient, ByVal param As String, ByVal type As String)

    Dim uriString As String = "http://" + My.Settings.esp_ip + param

    lblRes.ResetText()

    'Εκτέλεση GET request.
    Try
        If type = String.Empty Then

            RemoveHandler webClient.DownloadStringCompleted, AddressOf requestComplete
            AddHandler webClient.DownloadStringCompleted, AddressOf requestComplete
            webClient.DownloadStringAsync(New Uri(uriString))

        ElseIf type = "file" Then

            RemoveHandler webClient.DownloadFileCompleted, AddressOf downloadComplete
            AddHandler webClient.DownloadFileCompleted, AddressOf downloadComplete
            webClient.DownloadFileAsync(New Uri(uriString), "cpc_log.txt")

            lblRes.Text = "Downloading..."

        End If


    Catch ex As Exception
        status("Cannot initiate connection.", 0)
        Exit Sub
    End Try

End Sub
```



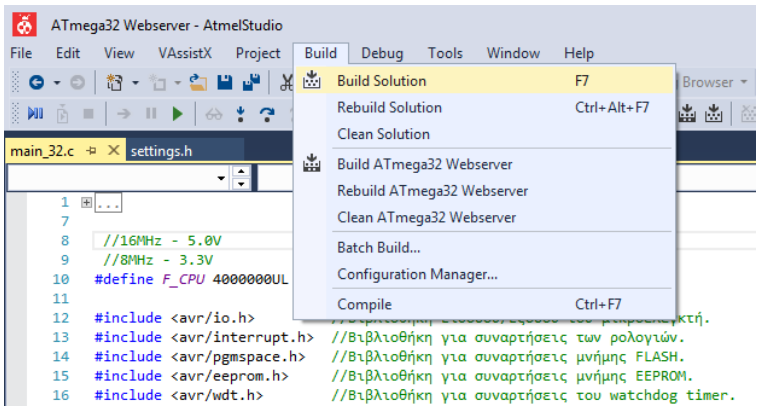
Εικόνα 8.18.1. Φόρμα σχεδίασης αρχικής οθόνης εφαρμογής.

Επιστρέφοντας στο Atmel Studio, αφού το πρόγραμμα μας είναι έτοιμο, επόμενο βήμα μας είναι να το κάνουμε **compile** και να χτίσουμε το αρχείο **HEX** το οποίο θα φορτώσουμε στον μικροελεγκτή.

Στο επόμενο κεφάλαιο θα αναφερθούμε στην όλη διαδικασία δημιουργίας του **HEX** αρχείου και την φόρτωση του, στον μικροελεγκτή.

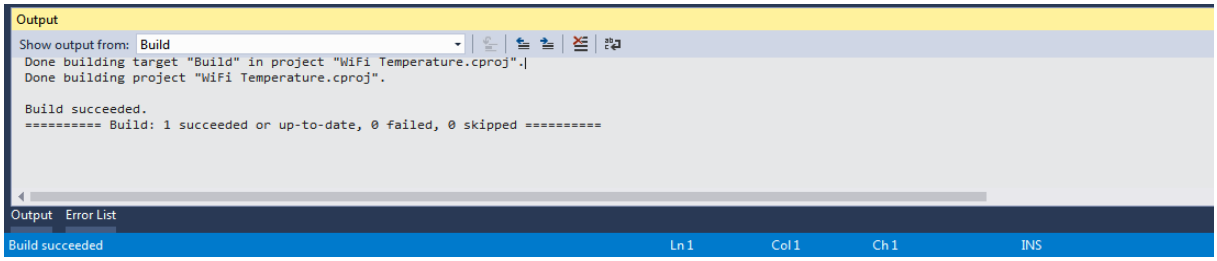
9. Προγραμματισμός Μικροελεγκτή

Έχοντας ανοιχτό το **Atmel Studio**, πηγαίνουμε στην καρτέλα «**Build**» και επιλέγουμε την επιλογή «**Build Solution**».



Εικόνα 9.1. Επιλογή Build Solution.

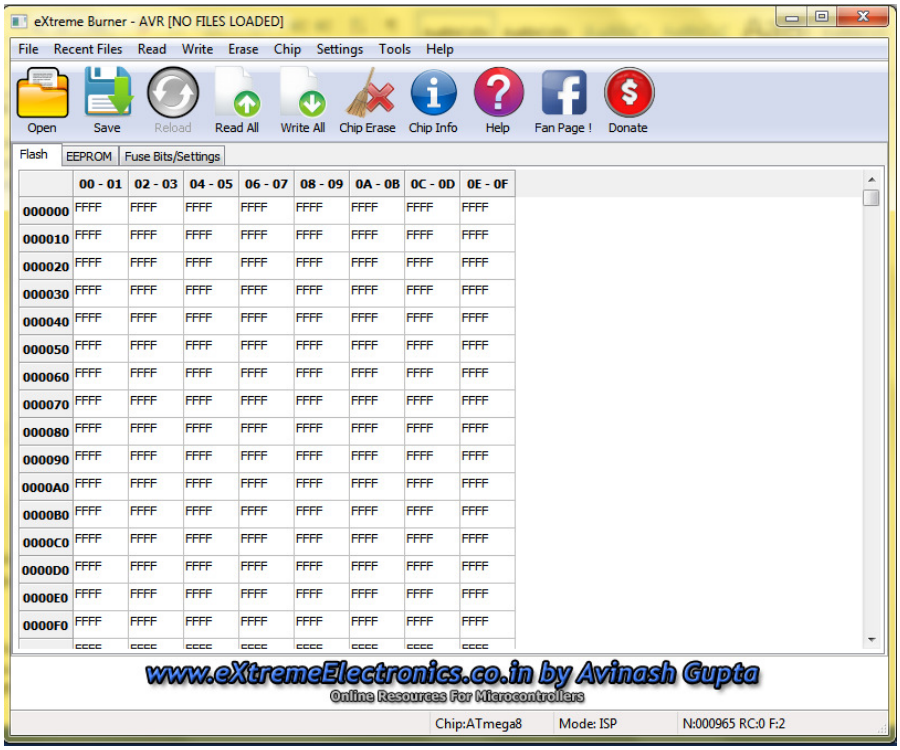
Στην συνέχεια, το **Atmel Studio** θα κάνει **compile** τον κώδικα μας και θα τον ελέγξει για συντακτικά λάθη. Αν δεν υπάρχουν λάθη, θα εμφανιστεί η παρακάτω οθόνη.



Εικόνα 9.2. Οθόνη επιτυχούς Build.

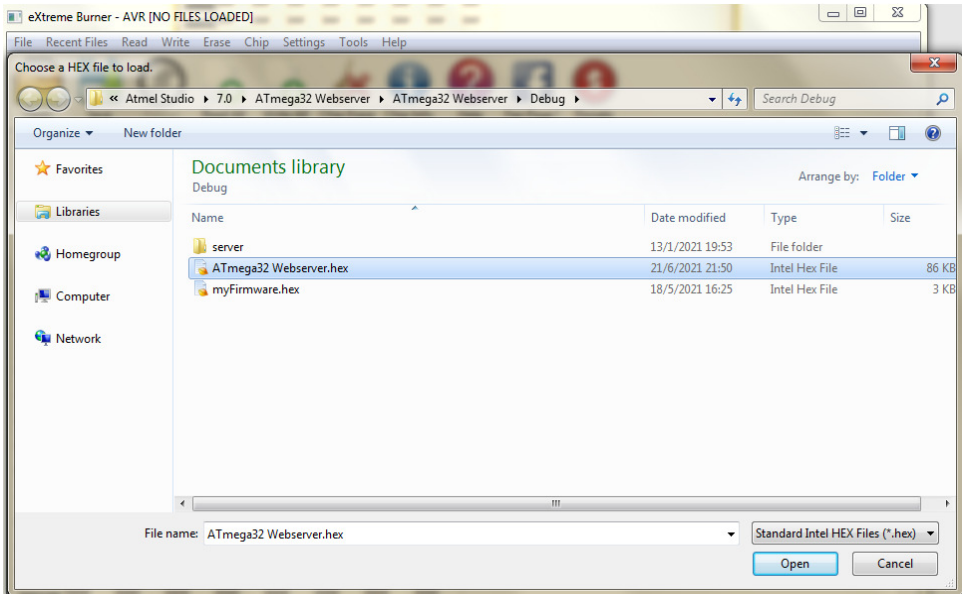
Αφού κάναμε **Build** το **Project** με επιτυχία, σημαίνει ότι έχουμε χτίσει και το αρχείο **HEX**. Το μόνο που μένει είναι να το φορτώσουμε στον μικροελεγκτή. Για να το κάνουμε αυτό θα χρησιμοποιήσουμε το πρόγραμμα **eXtreme Burner – AVR**, που έχουμε αναφέρει σε προηγούμενο κεφάλαιο.

Αφού το έχουμε κατεβάσει και εγκαταστήσει, το ανοίγουμε. Θα εμφανιστεί η παρακάτω οθόνη:



Εικόνα 9.3. Πρόγραμμα eXtreme Burner.

Στη συνέχεια πατάμε την επιλογή «**Open**». Θα μας εμφανιστεί η παρακάτω οθόνη:



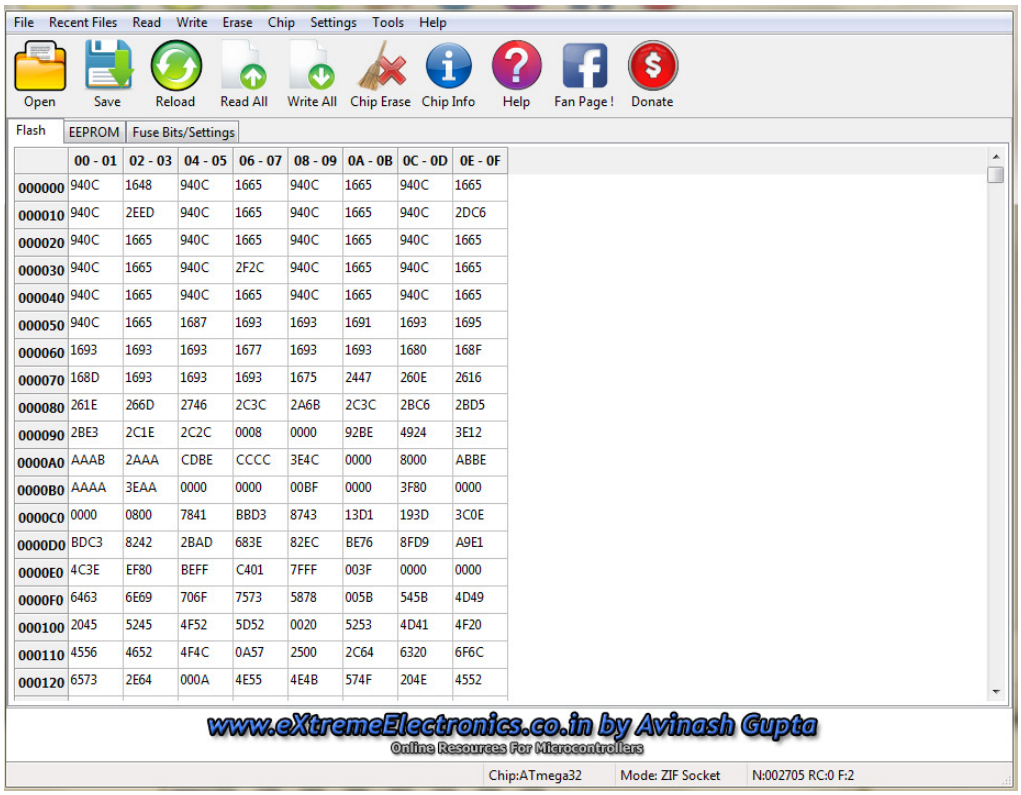
Εικόνα 9.4. Επιλογή HEX αρχείου.

Θα πρέπει να βρούμε το αρχείο **HEX** που έχουμε δημιουργήσει. Συνήθως, το αρχείο βρίσκεται στο φάκελο που βρίσκεται και το **project** μας:

C:\Users\<Όνομα_Χρήστη>\Documents\Atmel Studio\7.0\<Όνομα_Project>\<Όνομα_Project>\Debug

Το αρχείο θα ονομάζεται: <όνομα_project>.hex

Το επιλέγουμε και πατάμε «**Open**». Αφού το φορτώσουμε θα παρατηρήσουμε ότι οι τιμές των κελιών που ήταν «**FFFF»** έχουν διάφορες τιμές, κάπως έτσι:



	00 - 01	02 - 03	04 - 05	06 - 07	08 - 09	0A - 0B	0C - 0D	0E - 0F
000000	940C	1648	940C	1665	940C	1665	940C	1665
000010	940C	2EED	940C	1665	940C	1665	940C	2DC6
000020	940C	1665	940C	1665	940C	1665	940C	1665
000030	940C	1665	940C	2F2C	940C	1665	940C	1665
000040	940C	1665	940C	1665	940C	1665	940C	1665
000050	940C	1665	1687	1693	1693	1691	1693	1695
000060	1693	1693	1693	1677	1693	1693	1680	168F
000070	168D	1693	1693	1693	1675	2447	260E	2616
000080	261E	266D	2746	2C3C	2A6B	2C3C	2BC6	2BD5
000090	2BE3	2C1E	2C2C	0008	0000	92BE	4924	3E12
0000A0	AAAB	2AAA	CDBE	CCCC	3E4C	0000	8000	ABBE
0000B0	AAAA	3EAA	0000	0000	00BF	0000	3F80	0000
0000C0	0000	0800	7841	BBD3	8743	13D1	193D	3C0E
0000D0	BDC3	8242	2BAD	683E	82EC	BE76	8FD9	A9E1
0000E0	4C3E	EF80	BEFF	C401	7FFF	003F	0000	0000
0000F0	6463	6E69	706F	7573	5878	005B	545B	4D49
000100	2045	5245	4F52	5D52	0020	5253	4D41	4F20
000110	4556	4652	4F4C	0A57	2500	2C64	6320	6F6C
000120	6573	2E64	000A	4E55	4E4B	574F	204E	4552

Εικόνα 9.5. Φορτωμένο .hex αρχείο

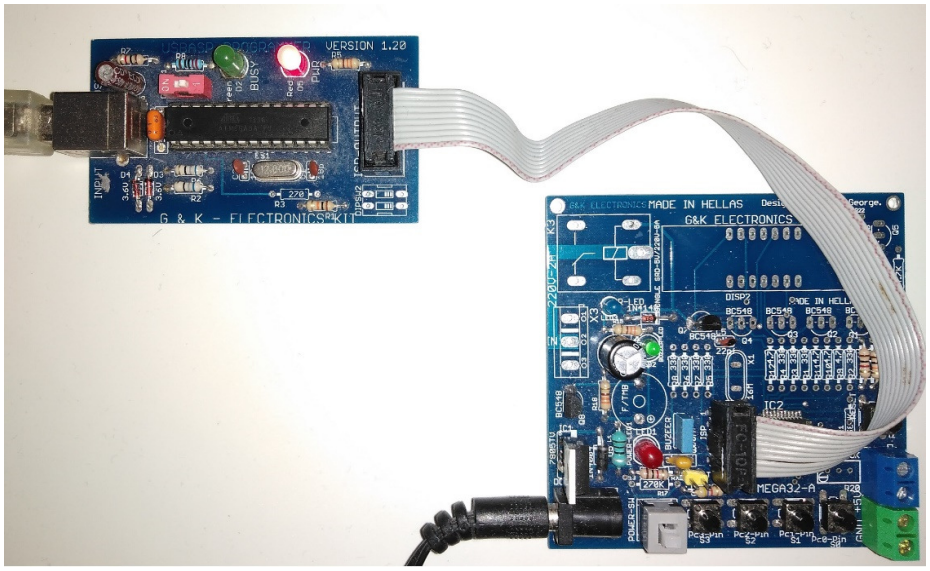
Είμαστε έτοιμοι να προγραμματίσουμε τον μικροελεγκτή μας. Πρώτα, συνδέουμε τον **Programmer** με τον υπολογιστή μας. Στην συνέχεια, συνδέουμε την πειραματική πλακέτα μας με τον **Programmer**:

Συνδέουμε τον φορτιστή με την πλακέτα μας και την ανοίγουμε πατώντας τον διακόπτη εκκίνησης (αν υπάρχει).

Συνήθως, μόλις συνδέσουμε τον φορτιστή ή γενικά μόλις η πλακέτα ενεργοποιηθεί, θα ανάψει ένα **ενδεικτικό LED λειτουργίας**.

Στη συνέχεια πρέπει να συνδέσουμε το **ESP** με την πλακέτα. Από το ESP, μας ενδιαφέρουν τέσσερα (4) **PIN**.

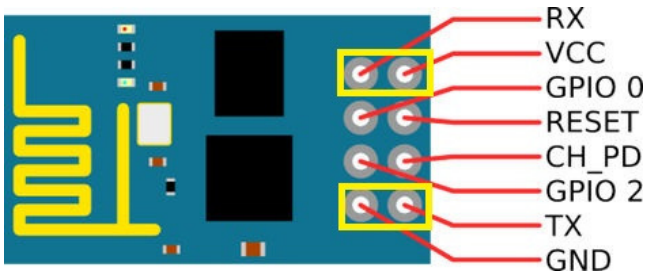
Όπως είχαμε αναφέρει, υπάρχουν πολλών ειδών **ESP Module**. Παραθέτουμε εικόνες για τις δύο πιο γνωστές εκδόσεις ESP Module:



Εικόνα 9.6. Σύνδεση Programmer και πλακέτας μέσω ISP καλωδίου.

Χρησιμοποιώντας καλώδια, κάνουμε την σύνδεση μεταξύ **Πλακέτας** και **ESP**:

1. **GND** ----> **GND**
2. **VCC** ----> **VCC**
3. **RX_1** ----> **TX_2**
4. **TX_1** ----> **RX_2**

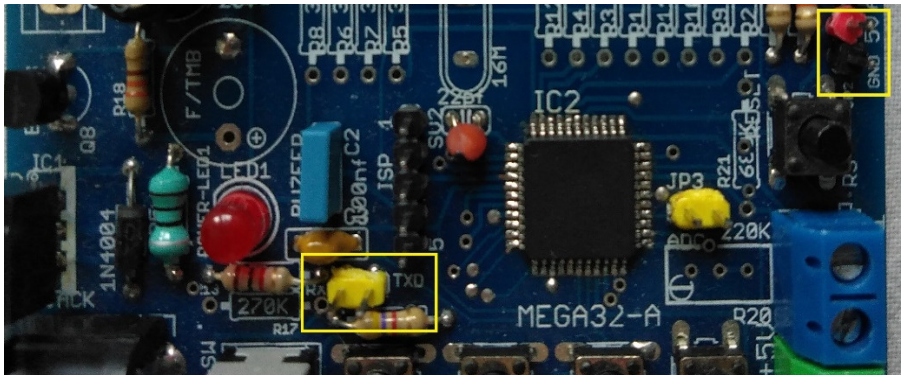


Εικόνα 9.7. Με κίτρινο πλαίσιο, είναι σημειωμένα τα PIN που μας ενδιαφέρουν.

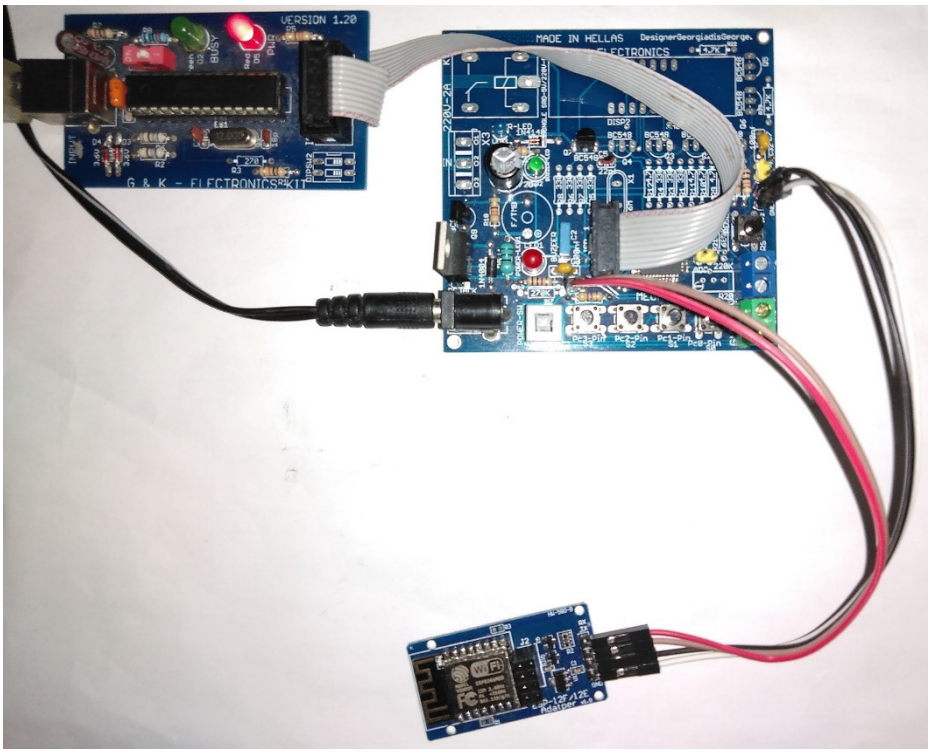
Στο τέλος, τα υλικά που πρέπει να έχουμε συνδέσει με την πλακέτα μας, είναι:

1. **Programmer**
2. **ESP Module**
3. **NTC Thermistor**

Εικόνα 9.8. Με κίτρινο πλαίσιο, είναι σημειωμένα τα PIN που μας ενδιαφέρουν.



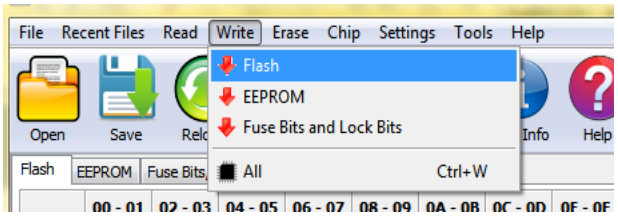
Εικόνα 9.9. Με κίτρινο πλαίσιο, είναι σημειωμένα τα αντίστοιχα PIN που μας ενδιαφέρουν.
Σημείωση: οι ακροδέκτες αυτοί αφορούν την δικιά μας πλακέτα. Ο χρήστης θα πρέπει να αναζητήσει στην δικιά του πλακέτα τους αντίστοιχους ακροδέκτες.



Εικόνα 9.10. Η τελική συνδεσμολογία μεταξύ πλακέτας και ESP.

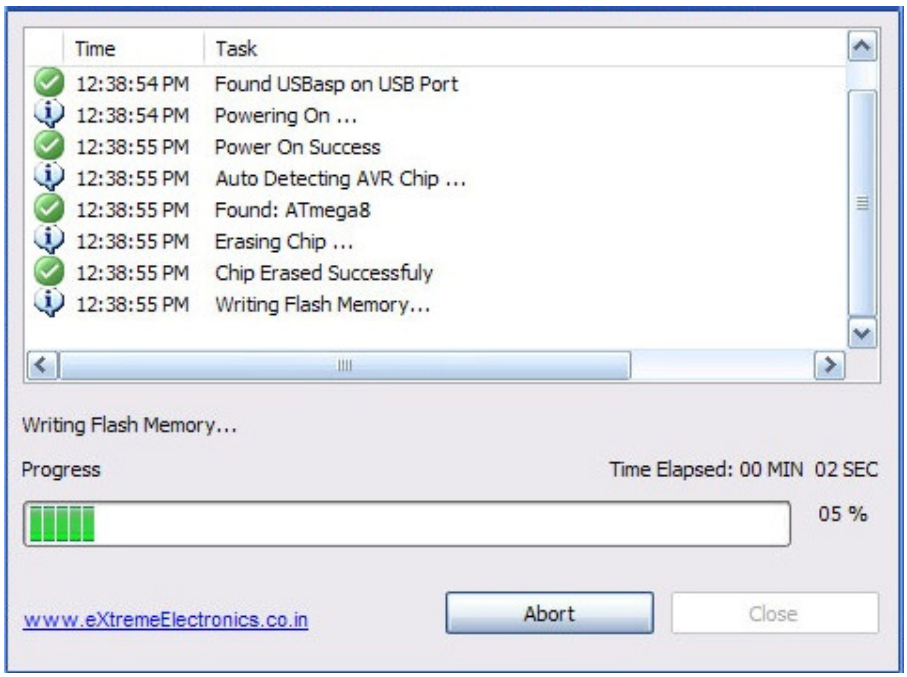
Είμαστε έτοιμοι να προγραμματίσουμε τον μικροελεγκτή μας. Επιστρέφοντας, στο **eXtreme Burner – AVR**, πηγαίνουμε στο μενού «**Chip**» και επιλέγουμε τον μικροελεγκτή που χρησιμοποιούμε, δηλαδή τον **ATmega8**.

Στη συνέχεια, πηγαίνουμε στο μενού «**Write**» και πατάμε «**Flash**».



Εικόνα 9.12. Επιλογή εκκίνησης προγραμματισμού μικροελεγκτή.

Μόλις το πατήσουμε, το πρόγραμμα θα φορτώσει το αρχείο **HEX** στην μνήμη **FLASH** του μικροελεγκτή μας.



Εικόνα 9.13. Προγραμματισμός μικροελεγκτή.

Αφού το φορτώσουμε επιτυχώς, το πρόγραμμα μας είναι σε αναμονή για εξωτερικές **HTTP** συνδέσεις. Στο επόμενο κεφάλαιο θα παρουσιάσουμε τον κώδικα για τα αρχεία ιστοσελίδας του server μας.

10. Αρχεία ιστοσελίδας (html, js, css)

Το αρχείο «**style.css**» χειρίζεται την μορφή των HTML στοιχείων μίας ιστοσελίδας (χρώματα, μεγέθη, γραμματοσειρές, σχετικές θέσεις κλπ.). Ο χρήστης, βεβαίως, καλείται να δημιουργήσει το δικό του css αρχείο.

```
#status{
    font-family: Segoe UI;
    font-size: 18px;
}
#status.error{
    color: red;
}
#status.success{
    color: green;
}
.tab {
  overflow: hidden;
  border: 1px solid #ccc;
  background-color: #97e6f1;
}
.tab button {
  background-color: inherit;
  float: left;
  border: 1px solid #999999;
  outline: none;
  cursor: pointer;
  padding: 14px 16px;
  transition: 0.3s;
  font-size: 17px;
}
.tab button:hover {
  background-color: #d3ffd3;
}
.tab button.active {
  background-color: #ffed44;
}
.tabcontent {
  display: none;
  padding: 20px 12px;
  border: 2px solid #a2a2a2;
  border-top: none;
  background-color: #fff48b;
}
.row {
  display: inline;
}
table, th, td {
  border: 1px solid black;
  border-collapse: collapse;
  text-align: center;
}
th, td {
  padding: 15px;
}
td.toggle {
  padding: 0px;
}
td input[type="text"] {
  text-align: center;
  width: 35%;
}
td.toggle button {
```

```
  width: 20px;
  height: 20px;
  border-radius: 50%;
}
td.toggle button:last-child {
    background-color: blue;
    border-radius: 0;
    float: right;
    margin-right: 10px;
}
td.noalign {
    text-align: none;
}
#tempGrad {
  padding: 0 10px;
  height: 322px;
  margin-left: 20px;
  background-image: linear-gradient(to top, #0000ff, #0043ff, #009fe1, #54cbdd, #00d691,
                    #0dd600, #bfd600, #f3e615, #ffff00, #ff8100, #ff3b00, #ff0000);
}
#tempArrow {
  width: 0;
  height: 0;
  border-top: 10px solid transparent;
  border-bottom: 10px solid transparent;
  border-right:10px solid black;
  float: right;
}
```

Το αρχείο «**custom.js**» χειρίζεται την λειτουργία των HTML στοιχείων μίας ιστοσελίδας (onclick συναρτήσεις, εμφάνιση/κρύψιμο στοιχείων κλπ.). Ο χρήστης, βεβαίως, καλείται να δημιουργήσει το δικό του js αρχείο.

```
window.addEventListener('load', function ()
{
    var isLocal = 1;
    var tries = 3;
    var i, j, r;
    var colourCycle = ["red", "green", "blue", "yellow", "white"];
    var currentColour = 0;
    var status = document.getElementById("status");

    //IO tab.
    var io_tab = document.getElementById("IO_tab");
    var rows = io_tab.getElementsByClassName("portRow");
    var inputs = io_tab.getElementsByTagName("input");

    var io = {
    //           a,  b,  c,  d
        ddr:    ["", "", "", ""],
        port:   ["", "", "", ""],
        pin:    ["", "", "", ""]
    };

    //Βρες τα Toggle buttons.
    var toggleColumns = io_tab.getElementsByClassName("toggle");
    for(i = 0; i < toggleColumns.length; i++)
    {
        var buttons = toggleColumns[i].getElementsByTagName("button");

        for(j = 0; j < buttons.length; j++)
        {
            //Toggle colour buttons.
            if(j == buttons.length - 1)
            {
```

```
            buttons[j].style.backgroundColor = colourCycle[currentColour];
            buttons[j].addEventListener("click", function(){
                toggleColour(this);
            });
        }
        else //Toggle PIN buttons.
        {
            buttons[j].addEventListener("click", function(){
                toggleIO(this);
            });
        }
    }
}

//Βρες τα reset/update  buttons.
document.getElementById("resd").addEventListener('click', function() {
        resetIO("ddr");
});
document.getElementById("resp").addEventListener('click', function() {
        resetIO("port");
});
document.getElementById("upd").addEventListener('click', function() {
        updateIO(this);
});

//Temperature tab.
var gradient = document.getElementById("tempGrad"); //Βρες το gradient button.
var arrow = document.getElementById("tempArrow");

//Πρέπει να εισάγουμε τιμές στις ιδιότητες χειροκίνητα για να υπάρχουν αργότερα.
gradient.style.height = "322px";
gradient.style.borderWidth = "2px";
arrow.style.borderTopWidth = "10px";

var gradTrueHeight = parseInt(gradient.style.height) - parseInt(gradient.style.borderWidth) * 2; //Αφαίρεση πλάτους border από το ύψος.
var minTemp = -50;
var maxTemp = 110;
var tempVal = document.getElementById("tempVal");

//Μέγιστη και ελάχιστη τιμή θερμοκρασίας.
var infoMaxTemp = document.getElementById("maxTemp");
var infoMinTemp = document.getElementById("minTemp");
var infoMaxTempVal = minTemp;
var infoMinTempVal = maxTemp;

//*****************************************************************************************
var createClickHandler = function(arg1, arg2) {
  return function(){ openTab(arg1, arg2); };
}

//Βάλε onclick συναρτήσεις στα tab buttons.
var tablinks = document.getElementsByClassName("tablinks");
for (i = 0; i < tablinks.length; i++)
    tablinks[i].onclick = createClickHandler(tablinks[i], tablinks[i].dataset.tab);

//*****************************************************************************************
var xhr; //Αντικείμενο για να κάνουμε HTTP αιτήματα.
var timer; //Timer για επαναλαμβανόμενα αιτήματα.

var resetIO = function(type)
{
    for(r = 0; r < 4; r++)
    {
        //Βρές τα inputs στην σειρά.
        var inputs = rows[r].getElementsByTagName('input');

        switch(type)
        {
            case "ddr":
                inputs[0].value = io[type][r];
                break;
            case "port":
                inputs[1].value = io[type][r];
                break;
            default:
        }
    }
};
```

```
var clearIO = function()
{
    for(i = 0; i < inputs.length; i++)
    {
        if(inputs[i].type == 'text')
            inputs[i].value = "";
    }

    //Εκκαθάριση IO τιμών.
    Object.keys(io).forEach(k => io[k].forEach((v, i) => io[k][i] = ""));
};

var toggleIO = function(btn)
{
    btn.disabled = true; //Απενεργοποίηση button μέχρι να τελειώσει το αίτημα.

    xhr = new XMLHttpRequest();
    sendRequest(xhr, "toggle", btn);
};

var updateIO = function(btn)
{
    btn.disabled = true; //Απενεργοποίηση button μέχρι να τελειώσει το αίτημα.
    var changed = false;

    var new_io = {
                    ddr:    ["", "", "", ""],
                    port:   ["", "", "", ""]
                 };

    for(r = 0; r < rows.length; r++)
    {
        inputs = rows[r].getElementsByTagName("input");

        for(i = 0; i < inputs.length; i++)
        {
            if(inputs[i].type == 'checkbox' && inputs[i].checked)
            {
                for(j = 0; j < inputs.length; j++)
                {
                    if(inputs[j].type == 'text')
                    {
                        if(j == 0 && inputs[j].value != io.ddr[r])
                        {
                            new_io.ddr[r] = inputs[j].value;
                            changed = true;
                        }
                        else if(j == 1 && inputs[j].value != io.port[r])
                        {
                            new_io.port[r] = inputs[j].value;
                            changed = true;
                        }
                    }
                }
            }
        }
    }
    //Αποστολή αιτήματος αν έχει αλλάξει τουλάχιστον μία τιμή.
    if(changed)
    {
        var json_io = JSON.stringify(new_io);

        xhr = new XMLHttpRequest();
        sendRequest(xhr, "io_upd", [json_io, btn]);
    }
    else
        btn.disabled = false;
};

var updatePIN = function(io_status, port_num)
{
    //Ενημέρωση κατάστασης PIN (toggle).
    var buttons = rows[port_num].getElementsByTagName('button');

    for(i = 0; i < buttons.length - 1; i++)
    {
        var colour;

        if(parseInt(io_status.pin[port_num], 16) & (1 << i))
            colour = colourCycle[currentColour];
        else
            colour = "darkgrey";

        buttons[buttons.length - 2 - i].style.backgroundColor = colour;
    }
};
```

```
var setToggle = function(data, btn)
{
    var port_num = btn.id.charCodeAt(0);

    if(port_num < 'a' || port_num > 'd')
        failGET("", "Invalid port."); //Μη έγκυρη Πόρτα.
    else
    {
        port_num -= 'a'.charCodeAt(0); //Βρες την θέση αριθμού πόρτας.

        //Ενημέρωση κατάστασεων PORT και PIN.
        io.port[port_num] = data.port[0];
        io.pin[port_num] = data.pin[0];

        console.log(port_num);
        var text_inputs = rows[port_num].querySelectorAll('input[type=text]');

        text_inputs[1].value = io.port[port_num].toUpperCase();
        text_inputs[2].value = io.pin[port_num].toUpperCase();

        updatePIN(io, port_num);
    }
    btn.disabled = false;
};

var getIO = function()
{
    xhr = new XMLHttpRequest();
    sendRequest(xhr, "io", 0);
};

var getTemp = function()
{
    xhr = new XMLHttpRequest();
    sendRequest(xhr, "temp", 0);
};

var setTemp = function(data)
{
    var temp = minTemp;

    //Μη έγκυρη τιμή.
    if(data == "")
    {
        tempVal.value = "NaN";
    }
    else
    {
        temp = parseFloat(data); //Μετατροπή τιμής σε float.

        if(temp) //Αν είναι έγκυρη.
        {
            tempVal.value = data; //Εμφάνισε την τιμή.

            //Έλεγχος τιμών.
            if(temp > infoMaxTempVal)
                infoMaxTempVal = temp;
            if(temp < infoMinTempVal)
                infoMinTempVal = temp;

            //Ενημέρωση τιμών.
            infoMaxTemp.value = infoMaxTempVal;
            infoMinTemp.value = infoMinTempVal;
        }
        else
        {
            tempVal.value = "Error"; //Μη-έγκυρα δεδομένα.
            temp = minTemp;
        }
    }

    //Μετακίνηση δείκτη ανάλογα με την θερμοκρασία.

    /*
        Εξίσωση για αντιστοίχιση τιμών θερμοκρασίας με το ύψους του δείκτη.

        [0, gradHeight] --> [minTemp, maxTemp]

        f(x) = height - (x + |minTemp|) * height / (maxTemp + |minTemp|), όπου 'x' η θερμοκρασία και 'f(x)' το αντίστοιχο ύψος σε pixel.
        Έχουμε υπόψιν και τα πλάτη των border του δείκτη και της μπάρας.
    */
    arrow.style.setProperty("margin-top", "calc(" + gradTrueHeight + "px - " +
                                "((" + String(temp) + "px + " + Math.abs(minTemp) + "px)*" + gradTrueHeight + "/" + (maxTemp + Math.abs(minTemp)) +
                                 ") - " +
                                String(parseInt(arrow.style.borderTopWidth) + parseInt(gradient.style.borderWidth)) + "px)");
};
```

```
var setIO = function(data, btn)
{
    if(data == "")
        clearIO();
    else
    {
        var new_io = data; //Δεν χρειάζεται μετατροπή.

        if(new_io) //Αν είναι έγκυρο.
        {
            for(r = 0; r < 4; r++)
            {
                //Βρες τα inputs στην σειρά.
                var inputs = rows[r].getElementsByTagName('input');

                for(i = 0; i < inputs.length; i++)
                {
                    if(inputs[i].type == 'text')
                    {
                        switch(i)
                        {
                            case 0:
                                if(new_io.ddr[r] != "" && new_io.ddr[r] != io.ddr[r])
                                    inputs[i].value = new_io.ddr[r].toUpperCase();
                                    io.ddr[r] = inputs[i].value;
                                break;
                            case 1:
                                if(new_io.port[r] != "" && new_io.port[r] != io.port[r])
                                    inputs[i].value = new_io.port[r].toUpperCase();
                                    io.port[r] = inputs[i].value;
                                break;
                            case 2:
                                if(new_io.pin[r] != "" && new_io.pin[r] != io.pin[r])
                                    inputs[i].value = new_io.pin[r].toUpperCase();
                                    io.pin[r] = inputs[i].value;
                                break;
                            default:
                        }
                    }

                    //Ενημέρωση κατάστασης PIN (toggle).
                    updatePIN(new_io, r);
                }
            }
            else
                clearIO();
        }

        if(btn)
            btn.disabled = false;
};

var sendRequest = function(xhr, request, param)
{
    var ip;
    //Είναι το ESP στο τοπικό δίκτυο μας;
    if(isLocal == 1)
        ip = document.querySelector("#ip").value; //Λήψη IP του ESP.
    else
        ip = window.location.hostname + ":1088"; //Εισαγωγή πόρτας, αν είναι εξωτερική IP.

    var url = "http://" + ip; //Χτίσιμο URL.

    xhr.onreadystatechange = function()
    {
        //Αν το αίτημα ήταν επιτυχές.
        if(this.readyState == 4 && this.status == 200)
        {
            isLocal = 1;
            tries = 3; //Αρχικοποίηση αριθμού επαναλήψεων.

            successGET(xhr.response, request, param);

            if(request == "temp")
                timer = setTimeout(function() { sendRequest(xhr, request, param);} , 1000); //Επανάληψη κάθε 1 sec.
        }
```

```
    else if(this.status == 404) //Αποτυχής σύνδεση.
    {
        isLocal = 0;

        failGET(xhr.response, "ESP not found in LAN, checking remote network...");
        sendRequest(xhr, request, param); //Δοκίμασε απομακρυσμένη σύνδεση.
    }
    else if(this.status == 204)
    {
        failGET("", "Unknown request.");
    }
};

xhr.ontimeout = function()
{
    tries--;

    if(tries == 0)
    {
        if(isLocal == 1)
        {
            isLocal = 0;
            tries = 3; //Αρχικοποίηση αριθμού επαναλήψεων.

            failGET("", "ESP not found in LAN, checking remote network...");
            sendRequest(xhr, request, param); //Δοκίμασε απομακρυσμένη σύνδεση.
        }
        else
        {
            failGET("", "Connection timed out. Please try again.");

            if(param && param.disabled)
                param.disabled = false;
        }
    }
    else //Try again.
        sendRequest(xhr, request, param);
};

xhr.onerror = function()
{
    if(isLocal)
    {
        isLocal = 0;
        tries = 3; //Αρχικοποίηση αριθμού επαναλήψεων.

        failGET("", "ESP not found in LAN, checking remote network...");
        sendRequest(xhr, request, param); //Δοκίμασε απομακρυσμένη σύνδεση.
    }
    else
    {
        failGET(xhr.response, "");

        if(param && param.disabled)
            param.disabled = false;
    }
};
var content = "text/plain";
var type = "text";

switch(request)
{
    case "io":
        url += "?ALL=1";
        content = "application/json";
        type = "json";
        break;

    case "io_upd":
        url += "?io_upd=" + encodeURIComponent(param[0]);
        content = "application/json";
        type = "json";
        break;

    case "toggle":
        url += "?toggle=" + encodeURIComponent(param.id);
        type = "json";
        break;

    case "temp":
        url += "?TEMP=1";
        break;
    default:
}
```

```
    xhr.open("GET", url);
    xhr.setRequestHeader("Content-Type", content);
    xhr.responseType = type;
    xhr.timeout = isLocal ? 5000 : 10000;
    xhr.send();
};

var successGET = function(data, request, param)
{
    status.classList.remove("error");
    status.classList.add("success");
    status.innerHTML = "Successfully connected to ESP Server.";

    switch(request)
    {
        case "io":
            setIO(data);
            break;

        case "temp":
            setTemp(data);
            break;

        case "io_upd":
            setIO(data, param[1]);
            break;

        case "toggle":
            setToggle(data, param);
            break;

        default:
    }
};

var failGET = function(error, msg = "")
{
    var activeTab = document.getElementsByClassName("tablinks active");

    if(activeTab.length == 1)
    {
        switch(activeTab[0].dataset.tab)
        {
            case "IO_tab":
                setIO("");
                break;
            case "TEMP_tab":
                setTemp("");
                break;
            default:
        }
    }

    status.classList.remove("success");
    status.classList.add("error");

    if(msg)
        status.innerHTML = msg;
    else
        status.innerHTML = "Failed to connect to ESP server: " + error;
};

var toggleColour = function(btn)
{
    if(currentColour < 4)
        currentColour++;
    else
        currentColour = 0;

    var row = btn.parentNode.parentNode;

    //Βρες την parent row.
    for(r = 0; r < rows.length; r++)
    {
        if(rows[r] === row)
            break;
    }
    if(r >= rows.length)
        return;

    btn.style.backgroundColor = colourCycle[currentColour];

    var buttons = btn.parentNode.getElementsByTagName("button");
    for(i = 0; i < buttons.length - 1; i++)
    {
        if(parseInt(io.pin[r], 16) & (1 << i))
            buttons[buttons.length - 2 - i].style.backgroundColor = colourCycle[currentColour];
    }
};
```

```
    var openTab = function(elem, name)
    {
        var i, tabcontent, tablinks;
        tabcontent = document.getElementsByClassName("tabcontent");

        for (i = 0; i < tabcontent.length; i++)
            tabcontent[i].style.display = "none";

        tablinks = document.getElementsByClassName("tablinks");
        for (i = 0; i < tablinks.length; i++)
        {
            tablinks[i].className = tablinks[i].className.replace(" active", "");
            tablinks[i].disabled = false;
        }

        document.getElementById(name).style.display = "block";
        elem.className += " active";

        //Σταμάτα όλα τα επαλαμβανόμενα αιτήματα.
        if(timer)
        {
            clearTimeout(timer);
            timer = 0;
        }

        //Κάλεσε την αντίστοιχη συνάρτηση.
        switch(name)
        {
            case "IO_tab":
                elem.disabled = true;
                getIO();
                break;
            case "TEMP_tab":
                elem.disabled = true;
                getTemp();
                break;
            default:
        }
    };

    //Ξεκίνα.
    openTab(tablinks[0], "IO_tab");

});
```

Η αρχική ροή του **script** μας, είναι ως εξής:

1. Λήψη **LAN IP** του **ESP.**
2. Εκτέλεσε **LAN** σύνδεση με το **ESP**.
3. Αν ήταν επιτυχής η σύνδεση εκτέλεση αίτημα **GET /ALL=1**.
4. Αν δεν ήταν επιτυχής η **LAN** σύνδεση, δοκίμασε **απομακρυσμένη** σύνδεση.
5. Αν δεν ήταν επιτυχής ούτε η **απομακρυσμένη** σύνδεση, εμφάνισε κατάλληλο μήνυμα και σταμάτα.

Με λίγα λόγια, ψάχνουμε να βρούμε το **ESP**, είτε στο τοπικό δίκτυο **LAN** είτε **απομακρυσμένα**. Αν δεν το βρούμε πουθενά, σταματάμε. Αν το βρούμε, εκτελούμε το κατάλληλο **HTTP GET Request** για την πλήρη κατάσταση των I/O Πορτών του μικροελεγκτή.

Ένα **HTTP GET Request**, ουσιαστικά είναι ένα αίτημα προς έναν **web server** (ιστοσελίδα) με σκοπό την λήψη κάποιας πληροφορίας. Σε αυτό το αίτημα μπορούμε να δώσουμε και παραμέτρους ώστε ο **web server** να γνωρίζει ακριβώς τί ζητάμε.

Πατώντας στο κουμπί/tab **«Temperature»,** βλέπουμε την τιμή θερμοκρασίας του συνδεδεμένου θερμίστορ στην πλακέτα μας.

Στην συνέχεια, παρουσιάζουμε το αρχείο **«index.html»**, όπου είναι το κύριο αρχείο της ιστοσελίδας μας. Σε αυτό το αρχείο, υπάρχει και το κρυμμένο (hidden) **HTML στοιχείο**, όπου θα μπει η **LAN IP** του **ESP**. Από αυτό το στοιχείο, το script μας «**custom.js**» θα λαμβάνει, με την σειρά του, την IP.

```
<html>
    <head>
        <link rel="icon" href="data:,">
        <script type="text/javascript" src="custom.js"></script>
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <input type="hidden" id="ip" value="">
    <body bgcolor="LightGray">
        <hr>
        <font face="Segoe UI" size="+2" color="firebrick"><b>ATmega32 Web Server using ESP8266</b></font>
        <font face="Segoe UI" size="+2" color="darkviolet"><b>  -  Κοσμάς Γεωργιάδης</b></font>
        <hr>
        <hr>
        <div id="status"></div>
        <hr>
        <br>
        <br>

        <div class="tab">
          <button class="tablinks" data-tab="IO_tab">I/O</button>
          <button class="tablinks" data-tab="TEMP_tab">Temperature</button>
        </div>

        <div id="IO_tab" class="tabcontent">

          <table style="width:100%">
            <tr>
              <th style="width: 5%;">#</th>
              <th style="width: 15%;">DDR (HEX)</th>
              <th style="width: 15%;">PORT (HEX)</th>
              <th style="width: 50%;">PIN (TOGGLE)</th>
              <th style="width: 15%;">VALUE (HEX)</th>
              <th>SET</th>
            </tr>
            <tr id="PORTA" class="portRow">
              <td>A</td>
              <td><input type="text" maxlength="2"></td>
              <td><input type="text" maxlength="2"></td>
              <td class="toggle" >
                <button id="a7"></button>
                <button id="a6"></button>
                <button id="a5"></button>
                <button id="a4"></button>
                <button id="a3"></button>
                <button id="a2"></button>
                <button id="a1"></button>
                <button id="a0"></button>
                <button id="acol"></button>
              </td>
              <td><input type="text" maxlength="2" disabled></td>
              <td><input type="checkbox"></td>
            </tr>
            <tr id="PORTB" class="portRow">
              <td>B</td>
```

```
      <td><input type="text" maxlength="2"></td>
      <td><input type="text" maxlength="2"></td>
      <td class="toggle" >
        <button id="b7"></button>
        <button id="b6"></button>
        <button id="b5"></button>
        <button id="b4"></button>
        <button id="b3"></button>
        <button id="b2"></button>
        <button id="b1"></button>
        <button id="b0"></button>
        <button id="bcol"></button>
      </td>
      <td><input type="text" maxlength="2" disabled></td>
      <td><input type="checkbox"></td>
    </tr>
    <tr id="PORTC" class="portRow">
      <td>C</td>
      <td><input type="text" maxlength="2"></td>
      <td><input type="text" maxlength="2"></td>
      <td class="toggle" >
        <button id="c7"></button>
        <button id="c6"></button>
        <button id="c5"></button>
        <button id="c4"></button>
        <button id="c3"></button>
        <button id="c2"></button>
        <button id="c1"></button>
        <button id="c0"></button>
        <button id="ccol"></button>
      </td>
      <td><input type="text" maxlength="2" disabled></td>
      <td><input type="checkbox"></td>
    </tr>
    <tr id="PORTD" class="portRow">
      <td>D</td>
      <td><input type="text" maxlength="2"></td>
      <td><input type="text" maxlength="2"></td>
      <td class="toggle" >
        <button id="d7"></button>
        <button id="d6"></button>
        <button id="d5"></button>
        <button id="d4"></button>
        <button id="d3"></button>
        <button id="d2"></button>
        <button id="d1"></button>
        <button id="d0"></button>
        <button id="dcol"></button>
      </td>
      <td><input type="text" maxlength="2" disabled></td>
      <td><input type="checkbox"></td>
    </tr>
    <tr>
      <td></td>
      <td><button id="resd">RESET</button></td>
      <td><button id="resp">RESET</button></td>
      <td></td>
      <td></td>
      <td><button id="upd">UPDATE</button></td>
    </tr>
  </table>
</div>
```

```
        <div id="TEMP_tab" class="tabcontent">
          <table style="width:40%">
            <tr>
              <th style="width: 5%;">Info</th>
              <th style="width: 30%;">Value (Celcius)</th>
              <th style="width: 5%;">Monitor</th>
            </tr>
            <tr>
               <td>Max</td>
               <td><input id="maxTemp" type="text" disabled></td>
               <td rowspan=3 class="noalign">
                   <input id="tempGrad" type="button"/>
                   <div id="tempArrow"></div>
               </td>
            </tr>
            <tr>
               <td>Current</td>
               <td><input id="tempVal" type="text" disabled></td>
            </tr>
            <tr>
               <td>Min</td>
               <td><input id="minTemp" type="text" disabled></td></td>
            </tr>
            </table>
        </div>
    </body>
</html>
```

Όπως παρατηρείτε η ιστοσελίδα μας είναι αρκετά λιτή. Ο χρήστης μπορεί να την φτιάξει όπως θέλει, αλλά θα πρέπει να έχει υπόψιν του ότι όσο μικρότερο το μέγεθος της, τόσο πιο γρήγορη θα είναι και η σύνδεση, αλλά και περισσότερο ελεύθερο χώρο θα έχουμε στην μνήμη FLASH.

Για να μπορούμε να κάνουμε εξωτερικές συνδέσεις όμως, πρέπει να πούμε στον **router** μας, να επιτρέπει τις εξωτερικές συνδέσεις στο **ESP**, χρησιμοποιώντας ξεχωριστή **HTTP Πόρτα (port)**.

Πρώτα θα πρέπει να μπούμε στις ρυθμίσεις του **router** μας.

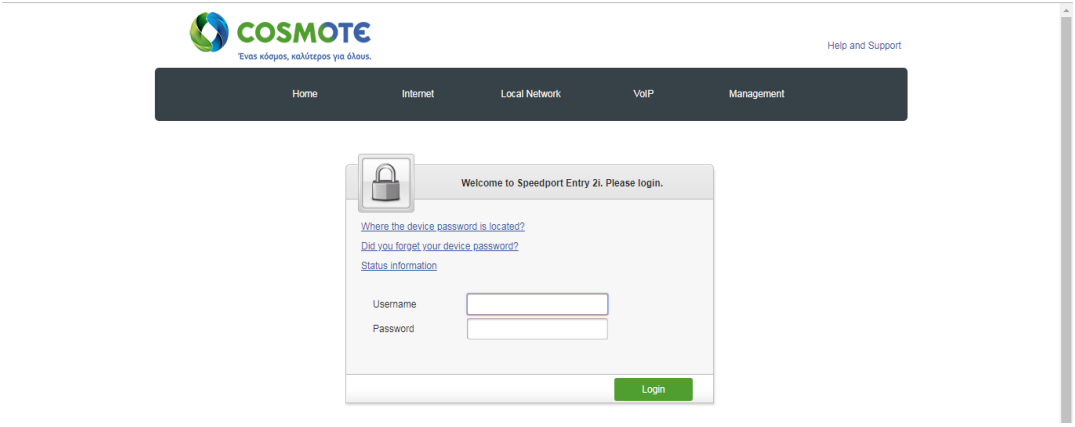
Ανοίγουμε μία νέα καρτέλα στον **Browser** της επιλογής μας και πληκτρολογούμε την διεύθυνση «**192.168.1.1**». Συνήθως, οι ρυθμίσεις του **router** βρίσκονται σε αυτήν την διεύθυνση.

Αφού πληκτρολογήσουμε τους κωδικούς πρόσβασης, θα πρέπει να βρούμε το μενού που να αφορά την **Ασφάλεια**. Σε αυτήν το μενού θα υπάρχει ένα υπό-μενού που θα αφορά τις **Πόρτες**. Συνήθως έχουν την ονομασία **Port Forwarding**.

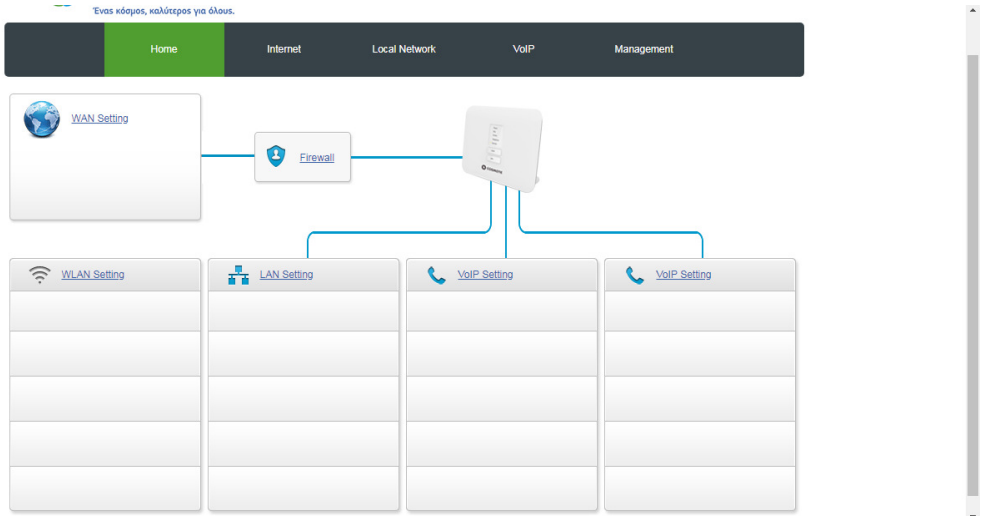
Στην συνέχεια θα πρέπει να δημιουργήσουμε μία νέα εγγραφή, που θα αφορά το **ESP**. Θα χρειαστούμε την **LAN IP** του ESP (όπου έχουμε ορίσει στο αρχείο “settings.h”) και την **Πόρτα** της επιλογής μας. Η πόρτα που θα επιλέξουμε δεν πρέπει να χρησιμοποιείται από άλλο πρόγραμμα ή συσκευή του δικτύου μας. Εμείς, επιλέγουμε την **Πόρτα 1088** όπως έχουμε ορίσει και στο αρχείο **«custom.js»**.

Τέλος, αποθηκεύουμε την εγγραφή μας και είμαστε έτοιμοι.

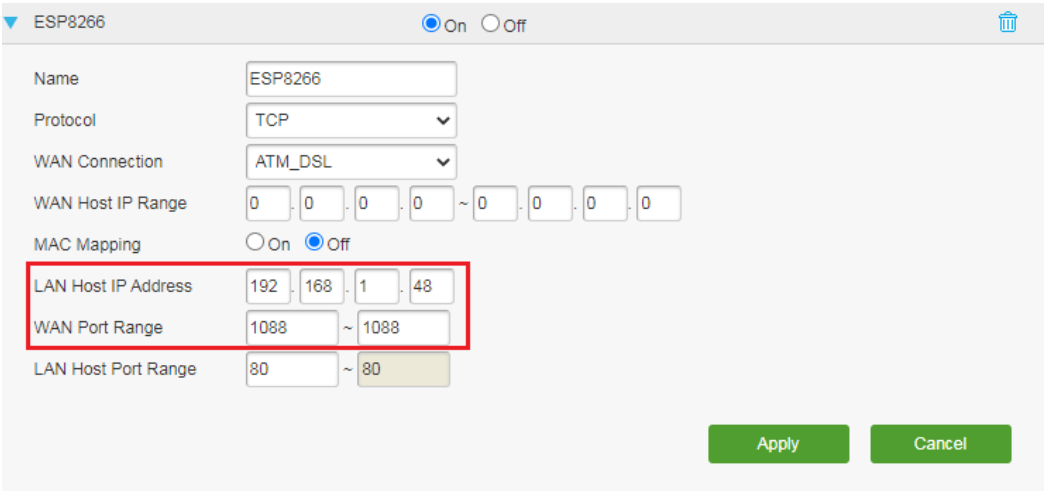
Παρακάτω παραθέτουμε φωτογραφίες με την όλη διαδικασία. Σημειώνουμε ότι διαφορετικοί **routers**, έχουν διαφορετικές ιστοσελίδες. Αν αντιμετωπίζετε προβλήματα, επικοινωνήστε με τον διαχειριστή του δικτύου σας.



Εικόνα 10.1. Φόρμα σύνδεσης με το Router του τοπικού μας δικτύου.



Εικόνα 10.2. Κεντρική οθόνη ρυθμίσεων του Router.



Εικόνα 10.3. Δημιουργία εγγραφής Port Forwarding για τον ESP. Με κόκκινο είναι σημειωμένα τα στοιχεία που χρειαζόμαστε. Τα υπόλοιπα συμπληρώνονται αυτόματα.

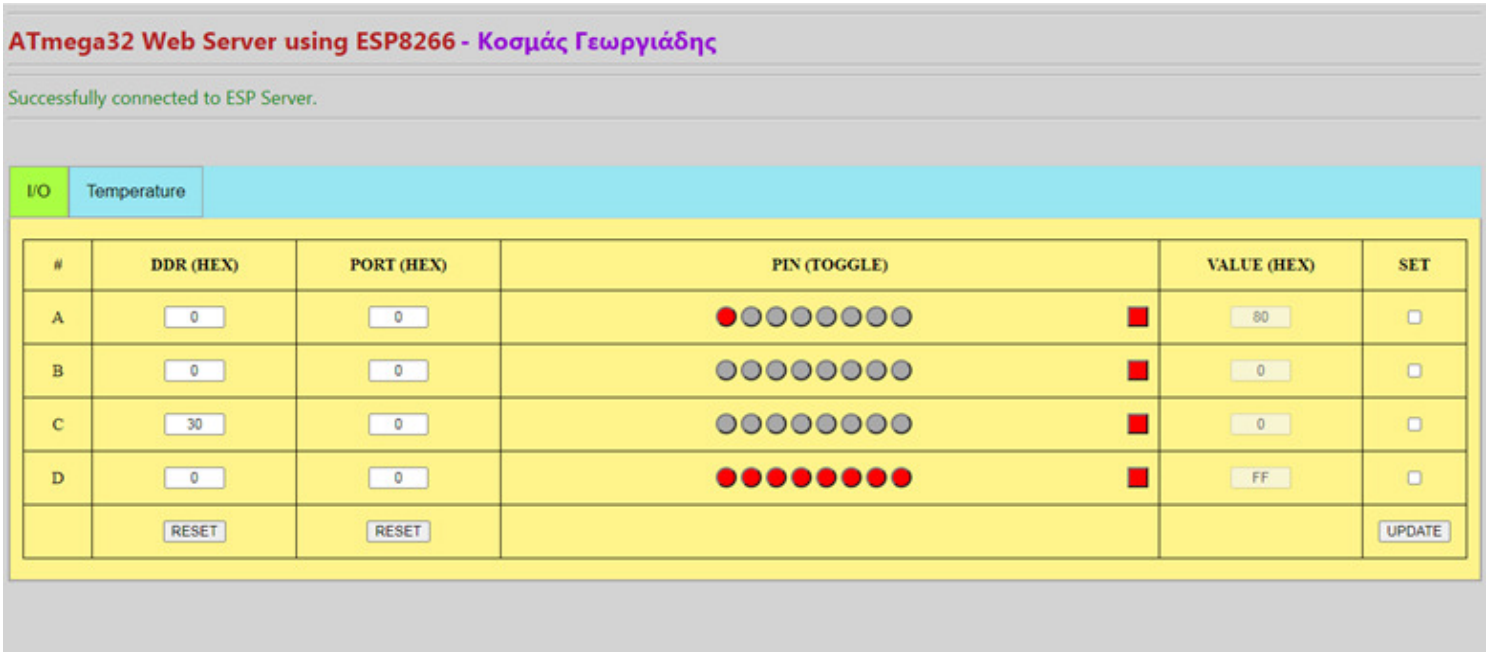
Στο επόμενο κεφάλαιο, θα δοκιμάσουμε να επικοινωνήσουμε με το **ESP** τοπικά και απομακρυσμένα.

11. LAN και Απομακρυσμένη Σύνδεση με το ESP

Από οποιαδήποτε συσκευή στο **LAN** δίκτυο που είναι συνδεδεμένο και το ESP, ανοίγουμε μία νέα καρτέλα σε έναν Browser, πληκτρολογούμε την LAN IP διεύθυνση του **ESP/server** και πατάμε Enter.

Π.χ.: **192.168.1.48**

Αν τα έχουμε κάνει όλα σωστά, θα εμφανιστεί η παρακάτω οθόνη (με κάποια καθυστέρηση):



Εικόνα 11.1 Αρχική οθόνη ιστοσελίδας.

Όπως βλέπουμε καταφέραμε και συνδεθήκαμε με τον **Server** και κατ' επέκταση με το **ESP**. Η κύρια οθόνη αποτελείται από όλες τις καταστάσεις όλων των I/O Πορτών του μικροελεγκτή.

Οι στήλες **«DDR (HEX)»** και **«PORT (HEX)»** αντιστοιχούν στις καταστάσεις DDR και PORT των Πορτών, σε δεκαεξαδική μορφή (hexadecimal).

Η στήλες **«PIN (TOGGLE)»** και **«VALUE (HEX)»** αντιπροσωπεύουν την κατάσταση PIN των Πορτών σε μορφή κουμπιών (button) και σε δεκαεξαδική, αντίστοιχα. Στην στήλη **«PIN (TOGGLE)»** κάθε κουμπί αντιπροσωπεύει και έναν ακροδέκτη της αντίστοιχης πόρτας. Τα κουμπιά αυτά, υποδεικνύουν την λογική κατάσταση των ακροδεκτών (0 ή 1), ανάλογα αν είναι χρωματισμένα ή όχι.

Ουσιαστικά, μπορούμε να τα θεωρήσουμε ως μικρά LED συνδεδεμένα σε κάθε ακροδέκτη. Αν ο ακροδέκτης είναι σε κατάσταση **Λογικού ένα (1)** τότε το κουμπί/LED θα είναι χρωματισμένο/αναμμένο. Αν όχι, τότε θα είναι μη-χρωματισμένο/σβησμένο (γκρι χρώμα).

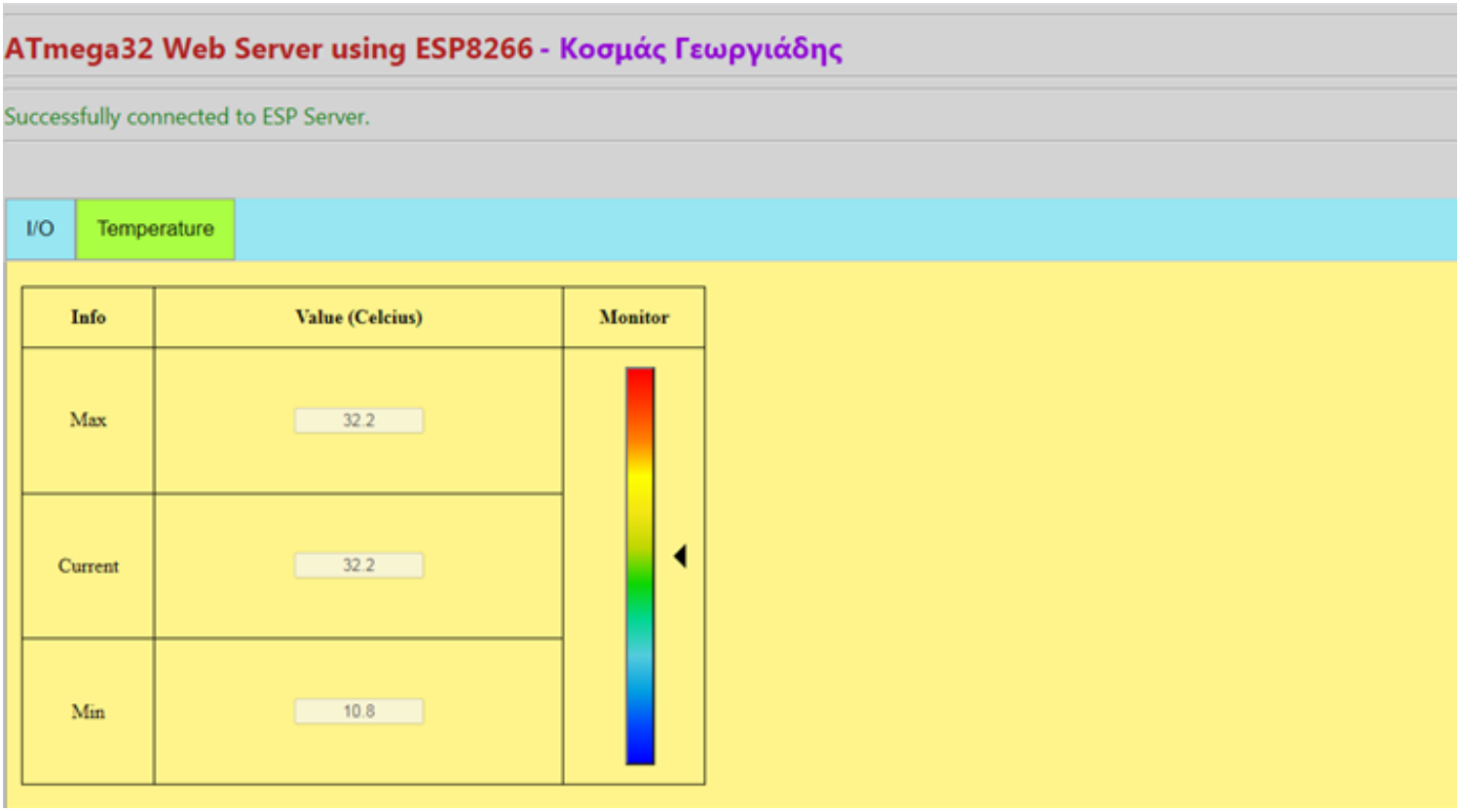
Πατώντας σε οποιαδήποτε κουμπί/LED, μπορούμε να μεταβάλλουμε (toggle) την κατάσταση του ακροδέκτη.

Αν είναι σε Λογικό ένα (1), θα γίνει Λογικό μηδέν (0) και το αντίστροφο. Έτσι μπορεί ο χρήστης εύκολα, να ανοιγοκλείνει συσκευές που θα μπορούσαν να ήταν συνδεδεμένες σε κάποιον ακροδέκτη, απομακρυσμένα.

Αν θέλουμε να αλλάξουμε τις καταστάσεις DDR και PORT, τοποθετούμε τις κατάλληλες τιμές στα αντίστοιχα πεδία τιμών και «τσεκάρουμε» το κουτί, που αντιστοιχεί στην πόρτα θα κάνουμε την αλλαγή, στην στήλη «SET». Ύστερα, πατάμε το κουμπί «UPDATE» για να στείλουμε το αίτημα με τις νέες καταστάσεις.

Τα κουμπιά **«RESET»** επαναφέρουν τις αρχικές τιμές για τις αντίστοιχες καταστάσεις (DDR/PORT).

Για να ελέγξουμε την τιμή της θερμοκρασίας του θερμίστορ μας, πατάμε στο κουμπί/tab **«Temperature»**, εμφανίζοντας την παρακάτω οθόνη:



Εικόνα 11.2 Οθόνη "Temperature" ιστοσελίδας.

Οι γραμμές **«Max»**, **«Current»** και **«Min»** αντιπροσωπεύουν την μέγιστη, τωρινή και ελάχιστη τιμή θερμοκρασίας που έχουμε λάβει. Η στήλη **«Monitor»** περιέχει την μπάρα θερμοκρασίας και τον δείκτη που δείχνει στο κατάλληλο χρώμα (μπλε – χαμηλή θερμοκρασία, κόκκινο – υψηλή θερμοκρασία).

Για να συνδεθούμε εκτός τοπικού δικτύου, πρέπει να γνωρίζουμε την **εξωτερική IP** του **server** μας. Χρησιμοποιώντας τις πολυάριθμες ιστοσελίδες στο Διαδίκτυο μπορούμε να την βρούμε εύκολα. Πληκτρολογώντας την παρακάτω διεύθυνση από μία συσκευή που βρίσκεται στο ίδιο δίκτυο με το ESP, βρίσκουμε την εξωτερική IP του δικτύου μας.

https://whatsmyip.com/

Πληκτρολογούμε την **IP** σε μία νέα καρτέλα, αλλά αυτή τη φορά προσθέτουμε και την **πόρτα** που έχουμε ορίσει για τον **server**.

Π.χ.: **46.12.212.70:1088**

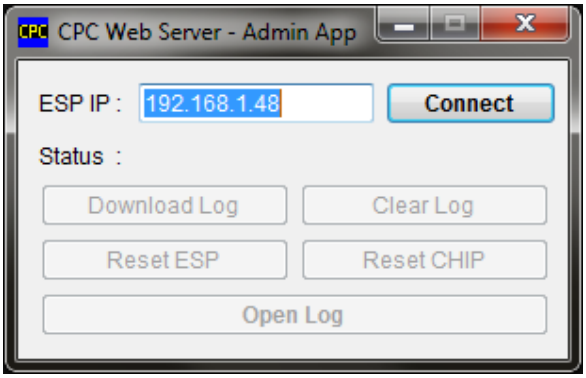
Στην αρχή θα μας εμφανίσει μήνυμα ότι δεν βρέθηκε το **ESP** στο **LAN** δίκτυο και ότι θα προσπαθήσει να γίνει απομακρυσμένη σύνδεση.

Εικόνα 11.3 Οθόνη αποτυχημένης σύνδεσης στο LAN.

Μετά από μερικά δευτερόλεπτα θα μας εμφανιστεί η αρχική οθόνη με τις τιμές των καταστάσεων των I/O Πορτών του μικροελεγκτή.

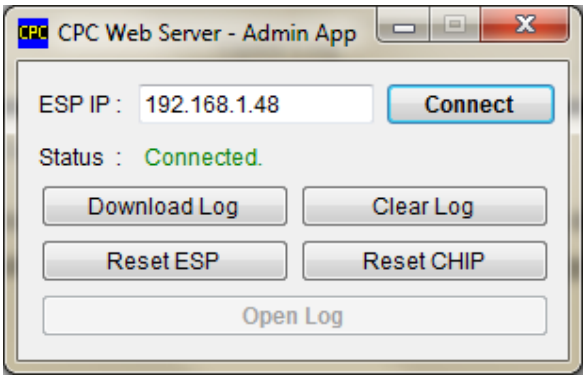
Στην συνέχεια, θα δοκιμάσουμε την εφαρμογή windows για τις λειτουργίες διαχειριστή (admin).

Η windows εφαρμογή μας αναπτύχθηκε σε γλώσσα **VB. NET** στην πλατφόρμα **Visual Studio 2019**. Εκτελώντας την εφαρμογή, θα εμφανιστεί η παρακάτω οθόνη:



Εικόνα 11.4 Αρχική οθόνη windows εφαρμογής.

Πληκτρολογούμε την **LAN IP** του server/ESP που έχουμε ορίσει, στο αρχείο **«settings.h»** και πατώντας Enter ή το κουμπί **«Connect»**, στέλνουμε το HTTP αίτημα **/?admin=1**.
Αν η σύνδεση έγινε επιτυχώς, θα εμφανιστεί η παρακάτω οθόνη:

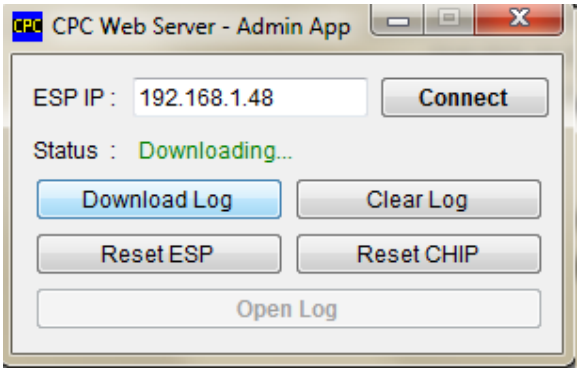


Εικόνα 11.5 Οθόνη επιτυχημένης σύνδεσης με το ESP.

Τώρα είμαστε έτοιμοι να εκτελέσουμε, οποιαδήποτε λειτουργία από τις τέσσερις (4) που έχουμε υλοποιήσει στο σύστημα μας.

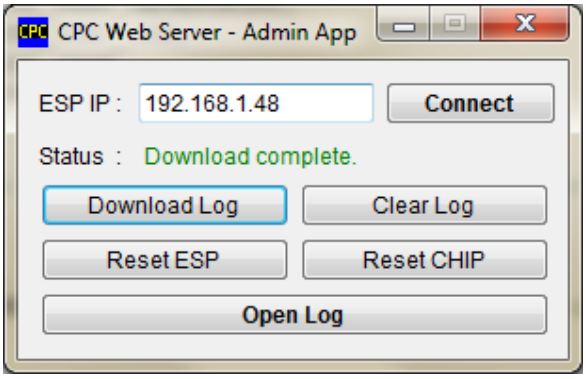
1. **Download Log** (Λήψη αρχείου καταγραφής σε μορφή .txt).
2. **Clear Log** (Εκκαθάριση αρχείου καταγραφής / μνήμης EEPROM).
3. **Reset ESP** (Επανεκκίνηση ESP).
4. **Reset CHIP** (Επανεκκίνηση μικροελεγκτή πλακέτας).

Ας δοκιμάσουμε την λήψη του αρχείου καταγραφής (1). Το αρχείο θα αποθηκευτεί στον φάκελο όπου βρίσκεται και η εφαρμογή μας **(.exe)**.



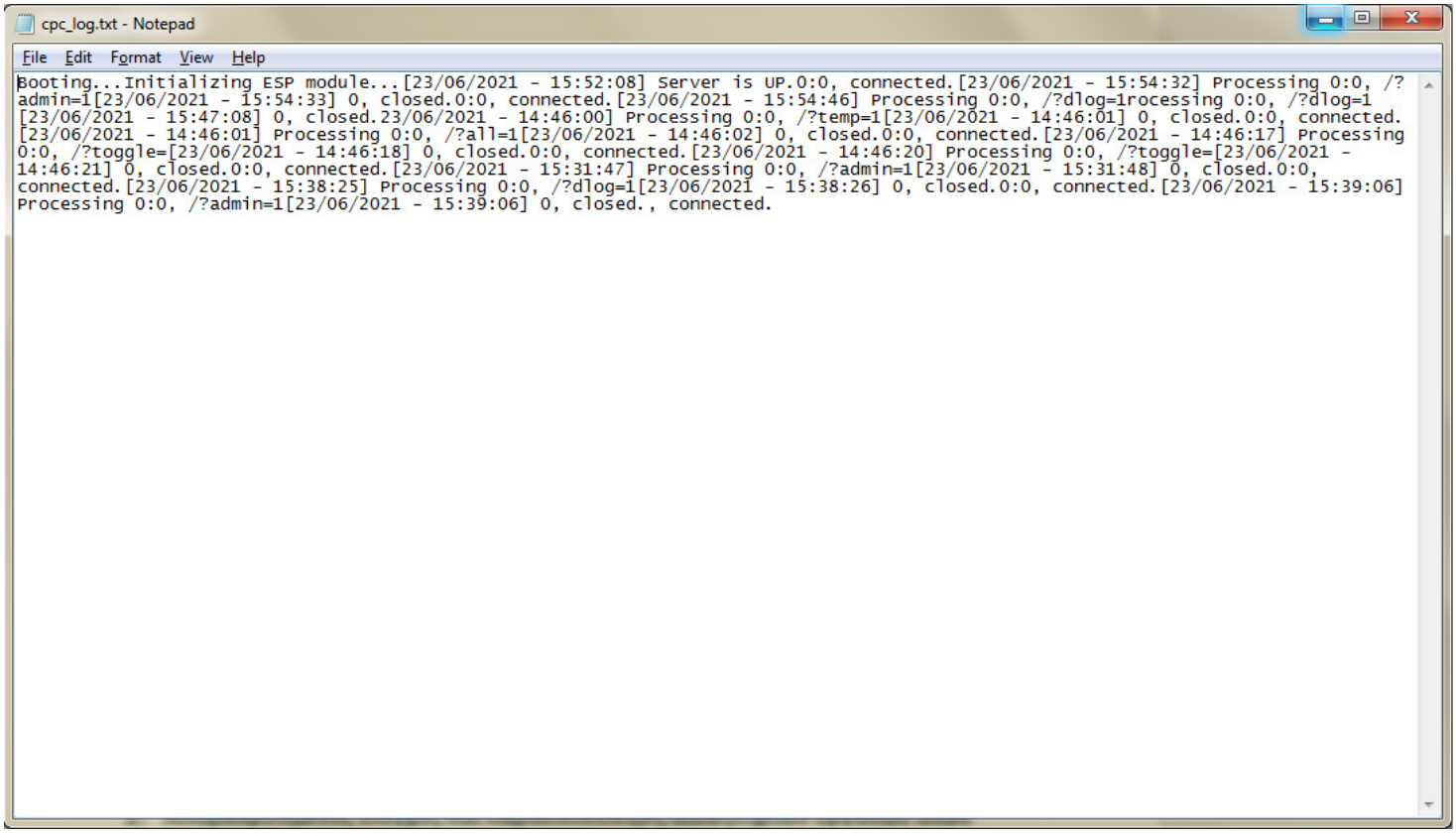
Εικόνα 11.6 Λήψη αρχείου καταγραφής.

Αφού ολοκληρωθεί η λήψη, το κουμπί **«Open Log»** θα ενεργοποιηθεί:



Εικόνα 11.7 Επιτυχής λήψη αρχείου καταγραφής.

Πατώντας στο κουμπί **«Open Log»**, ανοίγουμε αυτόματα το αρχείο καταγραφής με την εφαρμογή **Notepad.exe** (**Σημειωματάριο)**. Επειδή οι γραμμές χωρίζονται μόνο με χαρακτήρες **Line Feed (\n)**, το Σημειωματάριο αδυνατεί να τις επεξεργαστεί ως γραμμές γι' αυτό και μας εμφανίζει τα δεδομένα κολλητά.



Εικόνα 11.8 Αρχείο καταγραφής ανοιγμένο με Σημειωματάριο.

Χρησιμοποιώντας άλλη εφαρμογή όπως το **Notepad++**, μπορούμε να έχουμε καλύτερα αποτελέσματα:

```
Booting...
Initializing ESP module...
[23/06/2021 - 15:52:08] Server is UP.
0:0, connected.
[23/06/2021 - 15:54:32] Processing 0:0, /?admin=1
[23/06/2021 - 15:54:33] 0, closed.
0:0, connected.
[23/06/2021 - 15:54:46] Processing 0:0, /?dlog=1
rocessing 0:0, /?dlog=1
[23/06/2021 - 15:47:08] 0, closed.
23/06/2021 - 14:46:00] Processing 0:0, /?temp=1
[23/06/2021 - 14:46:01] 0, closed.
0:0, connected.
[23/06/2021 - 14:46:01] Processing 0:0, /?all=1
[23/06/2021 - 14:46:02] 0, closed.
0:0, connected.
[23/06/2021 - 14:46:17] Processing 0:0, /?toggle=
[23/06/2021 - 14:46:18] 0, closed.
0:0, connected.
[23/06/2021 - 14:46:20] Processing 0:0, /?toggle=
[23/06/2021 - 14:46:21] 0, closed.
0:0, connected.
[23/06/2021 - 15:31:47] Processing 0:0, /?admin=1
[23/06/2021 - 15:31:48] 0, closed.
0:0, connected.
[23/06/2021 - 15:38:25] Processing 0:0, /?dlog=1
[23/06/2021 - 15:38:26] 0, closed.
0:0, connected.
[23/06/2021 - 15:39:06] Processing 0:0, /?admin=1
[23/06/2021 - 15:39:06] 0, closed.
, connected.
```

Εικόνα 11.9 Αρχείο καταγραφής ανοιγμένο με Notepad++.

Όπως βλέπετε, τα αποτελέσματα είναι σαφώς καλύτερα μιας και έχουμε μία καλύτερη εικόνα για το τί γεγονότα καταγράψαμε. Μερικές φορές όμως, επειδή όταν γεμίζει η μνήμη EEPROM, αντικαθιστούμε παλιά γεγονότα με νέα, μπορούμε να δούμε «σπασμένα» ή μη κατανοητά γεγονότα, λόγω της αντικατάστασης που γίνεται ανά χαρακτήρα. Αυτός είναι ένας περιορισμός που αναγκαζόμαστε να δεχτούμε λόγω της φύσεως τόσο του συστήματος μας, όσο και της μνήμης EEPROM.

Με αυτό, τελειώσαμε την παρουσίαση της windows εφαρμογής. Αφού το σύστημα μας δουλεύει με επιτυχία, μπορούμε να μιλήσουμε για τις αμέτρητες πιθανές χρήσεις ενός τέτοιου συστήματος. Μερικές από αυτές είναι:

1. **Απομακρυσμένη πρόσβαση σε οικιακές συσκευές, γκαραζόπορτες, ηλεκτρικές βάνες, θερμοσίφωνες και οτιδήποτε συνδέεται σε ηλεκτρικό πίνακα.**
2. **Απομακρυσμένος έλεγχος και παρακολούθηση αισθητήριων οργάνων όλων των ειδών (θερμοκρασίας, υγρασίας, πίεσης κλπ.). Π.χ.: για θερμοκήπια, για ψυγεία.**

Αυτές είναι μόνο μερικές από τις πολλές χρήσεις. Οι χρήσεις του φτάνουν όσο φτάνει η φαντασία του επιστήμονα.

12. Επίλογος – Ιδέες για βελτίωση του συστήματος

Όπως είδαμε, χρησιμοποιήσαμε τον μικροελεγκτή **ATmega32**. Αν χρησιμοποιούσαμε έναν άλλον με περισσότερη μνήμη **FLASH, SRAM** και **EEPROM** (π.χ. **ATmega644**), τι θα κερδίζαμε με τον περίσσιο χώρο;

Θα μπορούσαμε, να υλοποιήσουμε μεγαλύτερη εφαρμογή, επεκτείνοντας το ρεπερτόριο εντολών ESP ώστε να διαχειριζόμαστε το ίδιο το ESP απομακρυσμένα. Οι ιστοσελίδες μας θα ήταν μεγαλύτερες και αυτό σημαίνει περισσότερες λειτουργίες/εφαρμογές.

Μια άλλη βελτίωση θα ήταν η προσθήκη ασφάλειας **SSL** κατά την αποστολή αιτημάτων. Προς το παρόν, το σύστημα μας δεν διαθέτει καμία ασφάλεια, εκτός από την ασφάλεια που παρέχει το ίδιο το δίκτυο που βρίσκεται συνδεδεμένο το ESP. Χρησιμοποιώντας SSL μπορούμε να προστατεύσουμε πλήρως το σύστημα μας από κακόβουλα λογισμικά και ανεπιθύμητους χρήστες. Το ESP παρέχει σύστημα για υλοποίηση SSL αλλά απαιτεί ιδιαίτερη γνώση του ESP και προγραμματισμό του ίδιου του ESP.

Όπως αναφέραμε, σκοπός της διπλωματικής εργασίας μας είναι η ανάδειξη των δυνατοτήτων ενός τέτοιου συστήματος, αλλά ταυτόχρονα, να εμπλουτιστεί ο χρήστης με την απαραίτητη γνώση ώστε να μπορεί να φτιάξει το δικό του σύστημα, χωρίς εξωτερική βοήθεια. Ως αποτέλεσμα τούτου, είναι και η **καινοτομία**. Η καινοτομία φέρνει με την σειρά της, **ανάπτυξη**.

13. Παράρτημα Α’

Στο παράρτημα αυτό παρατίθεται ο κώδικας του αρχείου **“settings.h”**.

```
#ifndef SETTINGS_H_
#define SETTINGS_H_

#include <avr/sfr_defs.h>
#include <avr/pgmspace.h>
#include "index_html.h"
#include "style_css.h"
#include "custom_js.h"

static const char ESP_DEFAULT_ANSWER[]                  = "OK";
static const char ESP_DEFAULT_APPENDER[]                = "\r\n";
static const char HTTP_DEFAULT_APPENDER[]               = "\r\n";
static const char AT_CMD_ERROR[]                        = "ERROR";

//Μεγέθη buffer.
#define MAX_RESPONSE_STRING_SIZE    412
#define MAX_FILE_BUF_LENGTH         412
#define MAX_URL_SIZE                140

//Έλεγχος για είδος μικροελεγκτή.
#if defined (__AVR_ATmega32__)
    #define CHIP                        32
#else
    #define CHIP                        328
    #define DDRA                        DDRB
    #define PORTA                       PORTB
    #define PINA                        PINB
#endif

//Λίστα σταθερών για το ESP.
#define ESP_TRANSMIT_DELAY_US           10
#define ESP_MAX_DATA_LENGTH             2048
#define ESP_MAX_CONNECTIONS             4
#define ESP_CONNECTION_MAX_REQUESTS     4

#define MAX_EXTRA_BUFFER_SIZE           2 * (MAX_URL_SIZE + 32)                                         //Μέγεθος βοηθητικού buffer. (πολλαπλάσιο του 2)
#define EXTRA_BUFFER_START              (uint16_t)((MAX_RESPONSE_STRING_SIZE - 1) - MAX_EXTRA_BUFFER_SIZE / 2)  //Η πρώτη θέση από την οποία αντιγράφουμε στον
βοηθητικό buffer.
                                                                                                        //ΣΗΜΕΙΩΣΗ: το casting είναι απαραίτητο για να
                                                                                                        αποφύγουμε την μετατροπή του σε float.
#define EXTRA_BUFFER_END                MAX_EXTRA_BUFFER_SIZE / 2 - 2                                   //Η τελευταία θέση από την οποία αντιγράφουμε στον
βοηθητικό buffer.
#define HOST_MAX_EXTRA_START            EXTRA_BUFFER_END - strlen_P(HTTP_REQUEST_HOST)                  //Μέγιστη θέση του string "Host:" που μπορεί να χωρέσει
στον βοηθητικό buffer.

#define MAX_JSON_PROP_NUM               5
#define MAX_JSON_PROP_LEN               10
#define MAX_JSON_ARRAY_LEN              10
//Ο χρήστης πρέπει να τα αλλάξει και να βάλει τα δικά του στοιχεία δικτύου.
static const char WIFI_STATION[] PROGMEM                = "TP-LINK";        //SSID του WiFI δικτύου που επιθυμούμε να συνδεθούμε.
static const char WIFI_KEY[] PROGMEM                    = "XXXXXXXXXXXXXX"; //Κλειδί ασφαλείας για το δίκτυο.
static const char ESP_IP[] PROGMEM                      = "192.168.1.48";   //Στατική IP για το ESP.

//Λίστα με προκαθορισμένους παραμέτρους για τον HTTP Header.
static const char HTTP_STATUS_CODE_200[] PROGMEM        = "HTTP/1.1 200 OK\r\n";
static const char HTTP_STATUS_CODE_204[] PROGMEM        = "HTTP/1.1 204 No Content\r\n";
static const char HTTP_STATUS_CODE_400[] PROGMEM        = "HTTP/1.1 400 Bad Request\r\n";
static const char HTTP_STATUS_CODE_507[] PROGMEM        = "HTTP/1.1 507 Insufficient Storage\r\n";
static const char HTTP_HEADER_ALLOW_ORIGIN[] PROGMEM    = "Access-Control-Allow-Origin: *\r\n";
static const char HTTP_CONTENT_LENGTH_FORMAT[] PROGMEM  = "Content-Length: %d\r\n";
static const char HTTP_CONTENT_TYPE_FORMAT[] PROGMEM    = "Content-Type: ";
static const char HTTP_REQUEST_HEADER_END[] PROGMEM     = "\r\n\r\n";

//Λίστα με προκαθορισμένες συμβολοσειρές για την αναγνώρηση ενός HTTP αιτήματος.
static const char TRANSMISSION_SIGNAL[] PROGMEM         = "+IPD";
static const char HTTP_GET_REQUEST[] PROGMEM            = "GET /";
static const char HTTP_SUBSTRING[] PROGMEM              = " HTTP";
static const char HTTP_REQUEST_HOST[] PROGMEM           = "Host:";

//Λίστα με προκαθορισμένες πλήρες/ρυθμιζόμενες ESP εντολές.
static const char AT_TEST[] PROGMEM                     = "AT";
static const char AT_RST[] PROGMEM                      = "AT+RST";
static const char ATE0[] PROGMEM                        = "ATE0";
static const char AT_CIFSR[] PROGMEM                    = "AT+CIFSR";
static const char AT_CIFSR_STATION_IP_FORMAT[] PROGMEM  = "STAIP,\"%s\"";
static const char AT_CWMODE_DEF_FILLED[] PROGMEM        = "AT+CWMODE_DEF=3";
static const char AT_CWJAP_DEF_FORMAT[] PROGMEM         = "AT+CWJAP_DEF=\"%s\",\"%s\"";
static const char AT_CWJAP_DEF_OK[] PROGMEM             = "WIFI GOT IP";
static const char AT_CWLAP[] PROGMEM                    = "AT+CWLAP";
static const char AT_CIPSTA_DEF_FORMAT[] PROGMEM        = "AT+CIPSTA_DEF=\"%s\"";
static const char AT_CIPMUX_FILLED[] PROGMEM            = "AT+CIPMUX=1";
static const char AT_CIPSERVER_FILLED[] PROGMEM         = "AT+CIPSERVER=1,80";
static const char AT_CIPSEND_FORMAT[] PROGMEM           = "AT+CIPSEND=%d,%d";
static const char AT_CIPSTATUS[] PROGMEM                = "AT+CIPSTATUS";
static const char AT_CIPCLOSE_FORMAT[] PROGMEM          = "AT+CIPCLOSE=%d";
static const char CONNECTION_CLOSED[] PROGMEM           = ",CLOSED";
static const char AT_CIPSNTPCFG_FORMAT[] PROGMEM        = "AT+CIPSNTPCFG=%d,%d";
static const char AT_CIPSNTPCFG_OFF[] PROGMEM           = "AT+CIPSNTPCFG=0";
static const char AT_CIPSNTPTIME[] PROGMEM              = "AT+CIPSNTPTIME?";
static const char AT_CIPSNTPTIME_TIME[] PROGMEM         = "+CIPSNTPTIME:";
static const char AT_CIPSEND_RECV_FORMAT[] PROGMEM      = "Recv %d bytes";
static const char AT_CIPSEND_SEND_OK[] PROGMEM          = "SEND OK";
static const char AT_NO_CHANGE[] PROGMEM                = "no change";
```

```
//Λίστα enum με τα είδη δεδομένων σε μία HTTP απάντηση.
enum CONTENT_TYPE
{
    TYPE_DEFAULT, //Αντιστοιχεί στο TEXT_PLAIN.
    TEXT_PLAIN,
    TEXT_HTML,
    TEXT_JAVASCRIPT,
    TEXT_CSS,
    APP_JSON,
    APP_OCTET,
    TYPE_LAST //Σύνολο τύπων.
};
typedef enum CONTENT_TYPE CONTENT_TYPE;

//Λίστα με προκαθορισμένες συμβολοσειρές HTTP Content-Type.
static const char HTTP_CONTENT_TYPE_TEXT_PLAIN[] PROGMEM        = "text/plain; charset=utf-8\r\n";
static const char HTTP_CONTENT_TYPE_TEXT_HTML[] PROGMEM         = "text/html; charset=utf-8\r\n";
static const char HTTP_CONTENT_TYPE_TEXT_JAVASCRIPT[] PROGMEM   = "text/javascript; charset=utf-8\r\n";
static const char HTTP_CONTENT_TYPE_TEXT_CSS[] PROGMEM          = "text/css\r\n";
static const char HTTP_CONTENT_TYPE_JSON[] PROGMEM              = "application/json\r\n";
static const char HTTP_CONTENT_TYPE_OCTET[] PROGMEM             = "application/octet-stream\r\n";

PGM_P CONTENT[TYPE_LAST - 1] =
{
    HTTP_CONTENT_TYPE_TEXT_PLAIN,
    HTTP_CONTENT_TYPE_TEXT_HTML,
    HTTP_CONTENT_TYPE_TEXT_JAVASCRIPT,
    HTTP_CONTENT_TYPE_TEXT_CSS,
    HTTP_CONTENT_TYPE_JSON,
    HTTP_CONTENT_TYPE_OCTET
};

//Λίστα enum για να ορίσουμε την σειρά των αιτημάτων στον πίνακα "GET_REQUESTS".
enum REQUEST
{
    REQ_INDEX_HTML,
    REQ_IP_PAGE = REQ_INDEX_HTML,   //Το HTML στοιχείο για την IP βρίσκεται στην κύρια ιστοσελίδα (index.html).
    REQ_CUSTOM_JS,
    REQ_CSS,
    REQ_FULL_IO,
    REQ_VAL,                        //Βοηθητική τιμή για να ξεχωρίζουμε τα αρχεία από τις απλές τιμές.
    REQ_VAL_TEMP = REQ_VAL,
    REQ_SET_FULL_IO,
    REQ_TOGGLE_IO_PIN,
    REQ_ADMIN_CONNECT,
    REQ_DOWNLOAD_LOG,
    REQ_CLEAR_LOG,
    REQ_RESET_ESP,
    REQ_RESET_CHIP,
    REQ_LAST
};

//Λίστα με προκαθορισμένες συμβολοσειρές HTTP αιτημάτων.
static const char INDEX_REQ[] PROGMEM                       = "/ ";             //index.html
static const char CUSTOM_JS_REQ[] PROGMEM                   = "/custom.js";     //Αρχείο JavaScript για την ιστοσελίδα μας.
static const char STYLE_REQ[] PROGMEM                        = "/style.css";     //Αρχείο stylesheet για την ιστοσελίδα μας.
static const char FULL_IO_REQ[] PROGMEM                      = "/?all=1";        //Ληψη πλήρης I/O κατάστασης.
static const char TEMP_REQ[] PROGMEM                         = "/?temp=1";       //Ληψη τιμής θερμοκρασίας.
static const char GET_SET_FULL_IO_REQ[] PROGMEM              = "/?io_upd=";      //Ενημέρωση I/O.
static const char GET_SET_FULL_IO_PARAM[] PROGMEM            = "?io_upd=";
static const char TOGGLE_IO_PIN_REQ[] PROGMEM                = "/?toggle=";      //Μεταβολή ενός ακροδέκτη I/O.
static const char TOGGLE_IO_PIN_REQ_PARAM_STATE[] PROGMEM    = "&state=";
```

```
//JSON συμβολοσειρές.
#if defined (__AVR_ATmega32__)                                  //Ο ATmega32 έχει μία έχτρα πόρτα.
    static const char FULL_IO_JSON[] PROGMEM                = "{"
                                                                    "\"ddr\":[\"%X\",\"%X\",\"%X\",\"%X\"],"
                                                                    "\"port\":[\"%X\",\"%X\",\"%X\",\"%X\"],"
                                                                    "\"pin\":[\"%X\",\"%X\",\"%X\",\"%X\"]"
                                                              "}";
    static const char UPDATE_IO_JSON[] PROGMEM              = "{"
                                                                    "\"ddr\":[\"%X\",\"%X\",\"%X\",\"%X\"],"
                                                                    "\"port\":[\"%X\",\"%X\",\"%X\",\"%X\"]"
                                                              "}";
    static const char GET_PIN_JSON[] PROGMEM                =       "\"pin\":[\"%X\",\"%X\",\"%X\",\"%X\"]";
#else
    static const char FULL_IO_JSON[] PROGMEM                = "{"
                                                                    "\"ddr\":[\"%X\",\"%X\",\"%X\"],"
                                                                    "\"port\":[\"%X\",\"%X\",\"%X\"]"
    static const char UPDATE_IO_JSON[] PROGMEM              = "{"
                                                                    "\"ddr\":[\"%X\",\"%X\",\"%X\"],"
                                                                    "\"port\":[\"%X\",\"%X\",\"%X\"]"
                                                              "}";
    static const char GET_PIN_JSON[] PROGMEM                =       "\"pin\":[\"%X\",\"%X\",\"%X\"]";
#endif

//Λίστα με αιτήματα διαχειριστή (admin).
static const char ADMIN_REQ[] PROGMEM                       = "/?admin=1";      //Σύνδεση διαχειριστή (admin).
static const char DOWNLOAD_LOG_REQ[] PROGMEM                = "/?dlog=1";       //Λήψη log αρχείου.
static const char CLEAR_LOG_REQ[] PROGMEM                   = "/?clog=1";       //Εκκαθάριση log αρχείου.
static const char RESET_ESP_REQ[] PROGMEM                   = "/?reset_esp=1";  //Επανεκκίνηση του ESP.
static const char RESET_CHIP_REQ[] PROGMEM                  = "/?reset_chip=1"; //Επανεκκίνηση του chip.

//Λίστα με βοηθητικές συμβολοσειρές.
static const char IO_DDR[] PROGMEM                          = "ddr";
static const char IO_PORT[] PROGMEM                         = "port";
static const char IO_PIN[] PROGMEM                          = "pin";

//Λίστα με βοηθητικές JSON συμβολοσειρές.
static const char JSON_PROP_ARRAY[] PROGMEM                 = "\"%s\":[";
static const char JSON_PROP_VAL[] PROGMEM                   = "\"%s\"";
static const char JSON_TOGGLE_IO[] PROGMEM                  = "{"
                                                                    "\"%s\":[\"%X\"],"
                                                                    "\"%s\":[\"%X\"]"
                                                              "}";

//Πίνακας HTTP αιτημάτων. Αυτή ο πίνακας πρέπει να αντιστοιχεί με την enum λίστα "REQUEST".
PGM_P GET_REQUESTS[REQ_LAST] =
{
    INDEX_REQ,
    CUSTOM_JS_REQ,
    STYLE_REQ,
    FULL_IO_REQ,
    TEMP_REQ,
    GET_SET_FULL_IO_REQ,
    GET_SET_IO_REQ,
    TOGGLE_IO_PIN_REQ,
    ADMIN_REQ,
    DOWNLOAD_LOG_REQ,
    CLEAR_LOG_REQ,
    RESET_ESP_REQ,
    RESET_CHIP_REQ
};

//Συμβολοσειρές μορφών ημ/νίας και ώρας.
static const char SNTP_FORMAT[] PROGMEM                     = "%s %hhu %hhu:%hhu:%hhu %d";
static const char LOG_TIMESTAMP_FORMAT[] PROGMEM            = "[%02d/%02d/%02d - %02d:%02d:%02d] ";

#endif /* SETTINGS_H_ */
```

14. Παράρτημα Β’

Στο παράρτημα αυτό παρατίθεται ο κώδικας του αρχείου **“events.h”**.

```
#ifndef EVENTS_H_
#define EVENTS_H_

#include <avr/pgmspace.h>

//Λίστα με προκαθορισμένες συμβολοσειρές γεγονότων.
static const char EVT_BOOTING[] PROGMEM             = "Booting...\n";
static const char EVT_INIT_ESP[] PROGMEM            = "Initializing ESP module...\n";
static const char EVT_RESET_ESP[] PROGMEM           = "Resetting ESP module...\n";
static const char EVT_SERVER_UP[] PROGMEM           = "Server is UP.\n";
static const char EVT_CLOCK_SYNC[] PROGMEM          = "Clock sync...\n";
static const char EVT_LOG_CLEAR[] PROGMEM           = "Log cleared.\n";

static const char EVT_NEW_CONN[] PROGMEM            = "%d:%d, connected.\n";
static const char EVT_EXIST_CONN[] PROGMEM          = "%d:%d: %s, extra.\n";
static const char EVT_OVFLOW_CONN[] PROGMEM         = "%d, connection overflow.\n";
static const char EVT_OVFLOW_REQ[] PROGMEM          = "%d, request overflow.\n";

static const char PROCESSING_CONN[] PROGMEM         = "Processing %d:%d, %s\n";

static const char MALFORMED_REQ[] PROGMEM           = "MALFORMED REQ: %d\n";
static const char TOO_BIG_REQ[] PROGMEM             = "TOO BIG %d\n";
static const char UNKNOWN_REQ[] PROGMEM             = "UNKNOWN REQ: %d:%d\n";
static const char NO_REQ[] PROGMEM                  = "NO REQ: %d:%d\n";

static const char EVT_CLOSED_CONN[] PROGMEM         = "%d, closed.\n";
static const char EVT_CLIENT_CLOSED_CONN[] PROGMEM  = "%d, closed (client).\n";
static const char EVT_WAIT_CLOSE[] PROGMEM          = "%d, wait close.\n";
static const char EVT_ABORT_CLOSE[] PROGMEM         = "%d, no close.\n";

static const char SRAM_OVERFLOW[] PROGMEM           = "SRAM OVERFLOW\n";
static const char JSON_FAIL[] PROGMEM               = "JSON FAIL\n";
static const char TIMESTAMP_ERROR[] PROGMEM         = "[TIME ERROR] ";

#endif /* EVENTS_H_ */
```

15. Παράρτημα Γ’

Στο παράρτημα αυτό παρατίθεται ο κώδικας του αρχείου **“index_html.h”**.

```
#ifndef INDEX_HTML_H_
#define INDEX_HTML_H_

#include <avr/pgmspace.h>
#include "settings.h"

//Αριθμός κομματιών.
#define INDEX_HTML_CODE_NUM_PARTS   3

static const char INDEX_HTML_HEAD[] PROGMEM = "<link href=data:, rel=icon><script src=custom.js></script><link href=style.css rel=stylesheet>";
static const char INDEX_HTML_IP_FIELD[] PROGMEM = "<input type=hidden id=ip value=%s>";
static const char INDEX_HTML_BODY[] PROGMEM = "<body bgcolor=LightGray><hr><font color=firebrick face=\"Segoe UI\"size=+2><b>ATmega32 Web Server"
                                              "using ESP8266</b></font> <font color=darkviolet face=\"Segoe UI\"size=+2><b>- Κοσμάς Γεωργιάδης</"
                                              "b></font><hr><hr><div id=status></div><hr><br><br><div class=tab><button class=tablinks data-tab="
                                              "IO_tab>I/O</button> <button class=tablinks data-tab=TEMP_tab>Temperature</button></div><div id=IO"
                                              "_tab class=tabcontent><table style=width:100%><tr><th style=width:5%>#<th style=width:15%>DDR (HE"
                                              "X)<th style=width:15%>PORT (HEX)<th style=width:50%>PIN (TOGGLE)<th style=width:15%>VALUE (HEX)<t"
                                              "h>SET<tr class=portRow id=PORTA><td>A<td><input type=text maxlength=2><td><input type=text maxlen"
                                              "gth=2><td class=toggle><button id=a7></button> <button id=a6></button> <button id=a5></button> <b"
                                              "utton id=a4></button> <button id=a3></button> <button id=a2></button> <button id=a1></button> <bu"
                                              "tton id=a0></button> <button id=acol></button><td><input type=text maxlength=2 disabled><td><inpu"
                                              "t type=checkbox><tr class=portRow id=PORTB><td>B<td><input type=text maxlength=2><td><input type="
                                              "text maxlength=2><td class=toggle><button id=b7></button> <button id=b6></button> <button id=b5><"
                                              "/button> <button id=b4></button> <button id=b3></button> <button id=b2></button> <button id=b1></"
                                              "button> <button id=b0></button> <button id=bcol></button><td><input type=text maxlength=2 disable"
                                              "d><td><input type=checkbox><tr class=portRow id=PORTC><td>C<td><input type=text maxlength=2><td><"
                                              "input type=text maxlength=2><td class=toggle><button id=c7></button> <button id=c6></button> <but"
                                              "ton id=c5></button> <button id=c4></button> <button id=c3></button> <button id=c2></button> <butt"
                                              "on id=c1></button> <button id=c0></button> <button id=ccol></button><td><input type=text maxlengt"
                                              "h=2 disabled><td><input type=checkbox><tr class=portRow id=PORTD><td>D<td><input type=text maxlen"
                                              "gth=2><td><input type=text maxlength=2><td class=toggle><button id=d7></button> <button id=d6></b"
                                              "utton> <button id=d5></button> <button id=d4></button> <button id=d3></button> <button id=d2></bu"
                                              "tton> <button id=d1></button> <button id=d0></button> <button id=dcol></button><td><input type=te"
                                              "xt maxlength=2 disabled><td><input type=checkbox><tr><td><td><button id=resd>RESET</button><td><b"
                                              "utton id=resp>RESET</button><td><td><td><button id=upd>UPDATE</button></table></div><div id=TEMP_"
                                              "tab class=tabcontent><table style=width:40%><tr><th style=width:5%>Info<th style=width:30%>Value "
                                              "(Celcius)<th style=width:5%>Monitor<tr><td>Max<td><input type=text disabled id=maxTemp><td class="
                                              "noalign rowspan=3><input type=button id=tempGrad><div id=tempArrow></div><tr><td>Current<td><inpu"
                                              "t type=text disabled id=tempVal><tr><td>Min<td><input type=text disabled id=minTemp></table></div>";

//Πίνακας με τα κομμάτια.
PGM_P INDEX_HTML_CODE[INDEX_HTML_CODE_NUM_PARTS] =
{
    INDEX_HTML_HEAD,
    INDEX_HTML_IP_FIELD,
    INDEX_HTML_BODY
};

#endif /* INDEX_HTML_H_ */
```

16. Παράρτημα Δ΄

Στο παράρτημα αυτό παρατίθεται ο κώδικας του αρχείου **“custom_js.h”**.

```
#ifndef CUSTOM_JS_H_
#define CUSTOM_JS_H_

#include <avr/pgmspace.h>

static const char JS_CUSTOM_CODE[] PROGMEM = "window.addEventListener(\"load\",function(){for(var o,l=1,i=3,s=[\"red\",\"green\","
"\"blue\",\"yellow\",\"white\"],r=0,d=document.getElementById(\"status\"),e=document"
".getElementById(\"IO_tab\"),c=e.getElementsByClassName(\"portRow\"),p=e.getElements"
"ByTagName(\"input\"),u={ddr:[\"\",\"\",\"\",\"\",\"\"],port:[\"\",\"\",\"\",\"\",\"\"],pin:[\""
"\",\"\",\"\",\"\",\"\"]},t=e.getElementsByClassName(\"toggle\"),m=0;m<t.length;m++)for(va"
"r n=t[m].getElementsByTagName(\"button\"),g=0;g<n.length;g++)g==n.length-1?(n[g].sty"
"le.backgroundColor=s[r],n[g].addEventListener(\"click\",function(){R(this)})):n[g]."
"addEventListener(\"click\",function(){M(this)});document.getElementById(\"resd\").a"
"ddEventListener(\"click\",function(){S(\"ddr\")}),document.getElementById(\"resp\")"
".addEventListener(\"click\",function(){S(\"port\")}),document.getElementById(\"upd\""
").addEventListener(\"click\",function(){P(this)});var a=document.getElementById(\"te"
"mpGrad\"),y=document.getElementById(\"tempArrow\");a.style.height=\"322px\",a.style."
"borderWidth=\"2px\",y.style.borderTopWidth=\"10px\";var h,v,b=parseInt(a.style.heigh"
"t)-2*parseInt(a.style.borderWidth),f=-50,E=document.getElementById(\"tempVal\"),k=do"
"cument.getElementById(\"maxTemp\"),N=document.getElementById(\"minTemp\"),B=f,I=110,"
"w=document.getElementsByClassName(\"tablinks\");for(m=0;m<w.length;m++)w[m].onclick="
"function(e,t){return function(){U(e,t)}}(w[m],w[m].dataset.tab);function C(){for(m=0"
";m<p.length;m++)\"text\"==p[m].type&&(p[m].value=\"\");Object.keys(u).forEach(n=>u[n"
"].forEach((e,t)=>u[n][t]=\"\"))}function L(e,t){var n=c[t].getElementsByTagName(\"bu"
"tton\");for(m=0;m<n.length-1;m++){var a=parseInt(e.pin[t],16)&1<<m?s[r]:\"darkgrey\""
";n[n.length-2-m].style.backgroundColor=a}}function T(e){var t=f;\"\"==e?E.value=\"N"
"aN\":(t=parseFloat(e))?(E.value=e,B<t&&(B=t),t<I&&(I=t),k.value=B,N.value=I):(E.val"
"ue=\"Error\",t=f),y.style.setProperty(\"margin-top\",\"calc(\"+b+\"px - ((\"+String"
"(t)+\"px + \"+Math.abs(f)+\"px)*\"+b+\"/\"+(110+Math.abs(f))+\") - \"+String(parseI"
"nt(y.style.borderTopWidth)+parseInt(a.style.borderWidth))+\"px)\")}function x(e,t){"
"if(\"\"==e)C();else{var n=e;if(n)for(o=0;o<4;o++){var a=c[o].getElementsByTagName(\""
"input\");for(m=0;m<a.length;m++)if(\"text\"==a[m].type)switch(m){case 0:\"\"!=n.ddr["
"o]&&n.ddr[o]!=u.ddr[o]&&(a[m].value=n.ddr[o].toUpperCase()),u.ddr[o]=a[m].value;bre"
"ak;case 1:\"\"!=n.port[o]&&n.port[o]!=u.port[o]&&(a[m].value=n.port[o].toUpperCase()"
"),u.port[o]=a[m].value;break;case 2:\"\"!=n.pin[o]&&n.pin[o]!=u.pin[o]&&(a[m].value"
"=n.pin[o].toUpperCase()),u.pin[o]=a[m].value}L(n,o)}else C()}t&&(t.disabled=!1)}var "
"S=function(e){for(o=0;o<4;o++){var t=c[o].getElementsByTagName(\"input\");switch(e)"
"{case\"ddr\":t[0].value=u[e][o];break;case\"port\":t[1].value=u[e][o]}}},M=function"
"(e){e.disabled=!0,h=new XMLHttpRequest,_(h,\"toggle\",e)},P=function(e){var t,n=!(e"
".disabled=!0),a={ddr:[\"\",\"\",\"\",\"\",\"\"],port:[\"\",\"\",\"\",\"\",\"\"]};for(o=0;o<c."
"length;o++)for(p=c[o].getElementsByTagName(\"input\"),m=0;m<p.length;m++)if(\"check"
"box\"==p[m].type&&p[m].checked)for(g=0;g<p.length;g++)\"text\"==p[g].type&&(0==g&&p"
"[g].value!=u.ddr[o]?(a.ddr[o]=p[g].value,n=!0):1==g&&p[g].value!=u.port[o]&&(a.port"
"[o]=p[g].value,n=!0));n?(t=JSON.stringify(a),h=new XMLHttpRequest,_(h,\"io_upd\",[t,"
"e])):e.disabled=!1},_=function(e,t,n){var a=1==l?document.querySelector(\"#ip\").va"
"lue:window.location.hostname+\":1088\",o=\"http://\"+a;e.onreadystatechange=functio"
"n(){4==this.readyState&&200==this.status?(l=1,i=3,q(e.response,t,n),\"temp\"==t&&(v"
"=setTimeout(function(){_(e,t,n)},1e3))):404==this.status?(l=0,A(e.response,\"ESP not"
" found in LAN, checking remote network...\"),_(e,t,n)):204==this.status&&A(\"\",\"U"
"nknown request.\")},e.ontimeout=function(){0==--i?1==l?(l=0,i=3,A(\"\",\"ESP not fo"
"und in LAN, checking remote network...\"),_(e,t,n)):(A(\"\",\"Connection timed out."
" Please try again.\"),n&&n.disabled&&(n.disabled=!1)):_(e,t,n)},e.onerror=function("
"){l?(l=0,i=3,A(\"\",\"ESP not found in LAN, checking remote network...\"),_(e,t,n))"
":(A(e.response,\"\"),n&&n.disabled&&(n.disabled=!1))};var s=\"text/plain\",r=\"text"
"\";switch(t){case\"io\":o+=\"?ALL=1\",s=\"application/json\",r=\"json\";break;case\""
"io upd\":o+=\"?io upd=\"+encodeURIComponent(n[0]),s=\"application/json\",r=\"json\""
```

```
";break;case\"toggle\":o+=\"?toggle=\"+encodeURIComponent(n.id),r=\"json\";break;cas"
"e\"temp\":o+=\"?TEMP=1\"}e.open(\"GET\",o),e.setRequestHeader(\"Content-Type\",s),e"
".responseType=r,e.timeout=l?5e3:1e4,e.send()},q=function(e,t,n){switch(d.classList.r"
"emove(\"error\"),d.classList.add(\"success\"),d.innerHTML=\"Successfully connected t"
"o ESP Server.\",t){case\"io\":x(e);break;case\"temp\":T(e);break;case\"io_upd\":x(e,"
"n[1]);break;case\"toggle\":a=e,(s=(o=n).id.charCodeAt(0))<\"a\"||\"d\"<s?A(\"\",\"I"
"nvalid port.\"):(s-=\"a\".charCodeAt(0),u.port[s]=a.port[0],u.pin[s]=a.pin[0],conso"
"le.log(s),(a=c[s].querySelectorAll(\"input[type=text]\"))[1].value=u.port[s].toUppe"
"rCase(),a[2].value=u.pin[s].toUpperCase(),L(u,s)),o.disabled=!1}var a,o,s},A=functio"
"n(e,t=\"\"){var n=document.getElementsByClassName(\"tablinks active\");if(1==n.lengt"
"h)switch(n[0].dataset.tab){case\"IO_tab\":x(\"\");break;case\"TEMP_tab\":T(\"\")}d.c"
"lassList.remove(\"success\"),d.classList.add(\"error\"),d.innerHTML=t||\"Failed to "
"connect to ESP server: \"+e},R=function(e){r<4?r++:r=0;var t=e.parentNode.parentNode"
";for(o=0;o<c.length&&c[o]!==t;o++);if(!(o>=c.length)){e.style.backgroundColor=s[r];"
"var n=e.parentNode.getElementsByTagName(\"button\");for(m=0;m<n.length-1;m++)parseIn"
"t(u.pin[o],16)&1<<m&&(n[n.length-2-m].style.backgroundColor=s[r])}},U=function(e,t){"
"for(var n,a=document.getElementsByClassName(\"tabcontent\"),o=0;o<a.length;o++)a[o]"
".style.display=\"none\";for(n=document.getElementsByClassName(\"tablinks\"),o=0;o<n"
".length;o++)n[o].className=n[o].className.replace(\" active\",\"\"),n[o].disabled=!"
"1;switch(document.getElementById(t).style.display=\"block\",e.className+=\" active\""
",v&&(clearTimeout(v),v=0),t){case\"IO_tab\":e.disabled=!0,h=new XMLHttpRequest,_(h,"
"\"io\",0);break;case\"TEMP_tab\":e.disabled=!0,h=new XMLHttpRequest,_(h,\"temp\",0)"
"}};U(w[0],\"IO_tab\")});";

#endif /* CUSTOM_JS_H_ */
```

17. Παράρτημα Ε’

Στο παράρτημα αυτό παρατίθεται ο κώδικας του αρχείου **“style_css.h”**.

```
#ifndef STYLE_CSS_H_
#define STYLE_CSS_H_

#include <avr/pgmspace.h>

static const char STYLE_CSS_CODE[] PROGMEM = "#status{font-family:Segoe UI;font-size:18px}#status.error{color:red}#status.success{color:green}."
                                             "tab{overflow:hidden;border:1px solid #ccc;background-color:#97e6f1}.tab button{background-color:"
                                             "inherit;float:left;border:1px solid #999;outline:0;cursor:pointer;padding:14px 16px;transition:."
                                             "3s;font-size:17px;color:black;}.tab button:hover{background-color:#d3ffd3}.tab button.active{bac"
                                             "kground-color:#aaff44}.tabcontent{display:none;padding:20px 12px;border:2px solid #a2a2a2;border"
                                             "-top:none;background-color:#fff48b}.row{display:inline}table,td,th{border:1px solid #000;border-"
                                             "collapse:collapse;text-align:center}td,th{padding:15px}td.toggle{padding:0}td input[type=text]{t"
                                             "ext-align:center;width:35%}td.toggle button{width:20px;height:20px;border-radius:50%}td.toggle b"
                                             "utton:last-child{background-color:#00f;border-radius:0;float:right;margin-right:10px}td.noalign{"
                                             "text-align:none;}#tempGrad{padding:0 10px;height:322px;margin-left:20px;background-image:linear-"
                                             "gradient(to top,#00f,#0043ff,#009fe1,#54cbdd,#00d691,#0dd600,#bfd600,#f3e615,#ff0,#ff8100,#ff3b0"
                                             "0,red)}#tempArrow{width:0;height:0;border-top:10px solid transparent;border-bottom:10px solid tr"
                                             "ansparent;border-right:10px solid #000;float:right}";

#endif /* STYLE_CSS_H_ */
```

Βιβλιογραφία

1. Difference between RISC and CISC architecture of processor, http://totalecer.blogspot.gr/2016/06/compare-risc-and-cisc-architecture-of.html

2. What is a Thermistor?, https://www.ei-sensor.com/what-is-a-thermistor/

3. LED Basics, US Department of Energy, https://www.energy.gov/eere/ssl/led-basics

4. AVR 8/16 Bit Timers/Counters – Tutorial #11, T.K. HAREENDRAN, https://www.electroschematics.com/avr-8-16-bit-timers-counters/

5. AVR121: Enhancing ADC resolution by oversampling, Atmel /Microchip, 2005 http://ww1.microchip.com/downloads/en/AppNotes/doc8003.pdf

6. 8-bit Atmel with 8KBytes In-System Programmable Flash, Atmel /Microchip, Rev.2486AA–AVR–02/2013, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2486-8-bit-AVR-microcontroller-ATmega8_L_datasheet.pdf

7. ESP8266 AT Instruction Set, Espressif Systems, Version 3.0.3, 2020, https://www.espressif.com/sites/default/files/documentation/4a-esp8266_at_instruction_set_en.pdf

8. C Programming and the ATmega16 Microcontroller, Jeffrey J. Richardson, https://web.ics.purdue.edu/~jricha14/Serial_Stuff/UART_information.htm

9. Atmel 8-bit AVR Microcontroller with 2/4/8K Bytes In-System Programmable Flash, Atmel/Microchip, Rev. 2586Q–AVR–08/2013, https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2586-AVR-8-bit-Microcontroller-ATtiny25-ATtiny45-ATtiny85_Datasheet.pdf

10. 8-bit Microcontroller with 32KBytes In-System Programmable Flash, Atmel/Microchip, 2503Q–AVR–02/11, http://ww1.microchip.com/downloads/en/devicedoc/doc2503.pdf

11. AVR XMEGA 8/16-bit High Performance Low Power Flash Microcontrollers, Atmel/Microchip, Rev.: 7925C-AVR-10/08/10M, 2008, http://ww1.microchip.com/downloads/en/DeviceDoc/doc7925.pdf

12. What is a potentiometer?, http://www.resistorguide.com/potentiometer/

13. RS232 Data Interface, a Tutorial on Data Interface and cables, ARC Electronics, https://arcelect.com/rs232.htm