ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

«Συσκευή IoT γενικού σκοπού»

Του φοιτητή
Παπαδόπουλου Κωνσταντίνου
Αρ. Μητρώου: 52213Μ

Επιβλέπων
Ονοματεπώνυμο: Γιακουμής Άγγελος
Βαθμίδα: Καθηγητής

Ημερομηνία 14/02/2026

Τίτλος Δ.Ε. Συσκευή IoT γενικού σκοπού
Κωδικός Δ.Ε. 24240
Ονοματεπώνυμο φοιτητή Κωνσταντίνος Παπαδόπουλος
Ονοματεπώνυμο εισηγητή Άγγελος Γιακουμής
Ημερομηνία ανάληψης Δ.Ε. 01/10/2024
Ημερομηνία περάτωσης Δ.Ε. 14/02/2026

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Η παρούσα διπλωματική εργασία πραγματεύεται τον σχεδιασμό και την υλοποίηση ενός ολοκληρωμένου συστήματος Internet of Things (IoT) γενικού σκοπού, με στόχο να γεφυρώσει την ανάγκη για ευελιξία ανάμεσα σε διαφορετικά σενάρια εγκατάστασης και χρήσης. Στην πράξη, πολλές λύσεις IoT είναι «ειδικού σκοπού», δηλαδή σχεδιάζονται για μία συγκεκριμένη εφαρμογή, με αποτέλεσμα η προσαρμογή τους σε νέα απαίτηση να συνεπάγεται αλλαγές σε υλικό, λογισμικό ή και στη διαδικτυακή πλατφόρμα.

Στο πλαίσιο της εργασίας αναπτύχθηκε μία συσκευή βασισμένη σε μικροελεγκτή STM32, με πληθώρα εισόδων/εξόδων, δυνατότητα μέτρησης ηλεκτρικών μεγεθών δικτύου, υποστήριξη επεκτασιμότητας μέσω CAN και επικοινωνία μέσω WiFi ή LTE. Παράλληλα, υλοποιήθηκε backend σε Python για συλλογή/αποθήκευση δεδομένων και web πλατφόρμα για εποπτεία, ιστορικά και παραμετροποίηση. Κεντρική σχεδιαστική επιλογή αποτελεί ο μηχανισμός κανόνων (rules), ώστε μέρος της λογικής αυτοματισμού να εκτελείται τοπικά στη συσκευή, ενισχύοντας την αυτονομία και την αξιοπιστία σε περιπτώσεις αστάθειας δικτύου.

Η υλοποίηση που παρουσιάζεται δεν στοχεύει μόνο στην επίδειξη επιμέρους τεχνολογιών, αλλά σε μία end-to-end προσέγγιση (hardware–firmware–επικοινωνία–backend–web), όπου η παραμετροποίηση και το abstraction επιτρέπουν την επαναχρησιμοποίηση της ίδιας πλατφόρμας σε διαφορετικές εφαρμογές.

Περίληψη

Η διπλωματική εργασία παρουσιάζει τον σχεδιασμό και την υλοποίηση μιας IoT συσκευής γενικού σκοπού, η οποία μπορεί να χρησιμοποιηθεί σε διαφορετικά σενάρια εποπτείας και ελέγχου χωρίς ανασχεδίαση του υλικού ή επαναπρογραμματισμό για κάθε εφαρμογή. Η προτεινόμενη λύση αποτελείται από τέσσερα βασικά επίπεδα: τη συσκευή (hardware + firmware), το επίπεδο επικοινωνίας, το backend σύστημα και τη διαδικτυακή πλατφόρμα χρήστη.

Σε επίπεδο υλικού, αναπτύχθηκε πλακέτα βασισμένη σε μικροελεγκτή STM32, με υποστήριξη αναλογικών και ψηφιακών εισόδων/εξόδων, PWM, ρελέ, CAN bus και δυνατότητα μέτρησης ηλεκτρικών μεγεθών εναλλασσόμενου δικτύου. Η συνδεσιμότητα υλοποιείται με δύο εναλλακτικά κανάλια, WiFi και LTE, ώστε το σύστημα να μπορεί να λειτουργεί τόσο σε τοπικά δίκτυα όσο και σε απομακρυσμένες εγκαταστάσεις. Η ανταλλαγή δεδομένων και εντολών βασίζεται στο πρωτόκολλο MQTT, αξιοποιώντας τη λογική publish/subscribe.

Το firmware υλοποιείται με modular αρχιτεκτονική, όπου τα υποσυστήματα (μετρήσεις, είσοδοι/έξοδοι, επικοινωνίες κ.λπ.) οργανώνονται ως ανεξάρτητα components. Ιδιαίτερη έμφαση δίνεται στον μηχανισμό παραμετροποίησης και στον rules engine, που επιτρέπουν την αποστολή ρυθμίσεων και κανόνων από την πλατφόρμα προς τη συσκευή και την τοπική εκτέλεσή τους (edge processing), μειώνοντας την εξάρτηση από το backend και βελτιώνοντας την απόκριση σε αυτοματισμούς.

Το backend αναπτύχθηκε σε Python και λειτουργεί ως κόμβος συλλογής τηλεμετρίας μέσω MQTT, κανονικοποίησης και αποθήκευσης σε MySQL/MariaDB, καθώς και ως μηχανισμός downlink για αποστολή ρυθμίσεων/κανόνων προς τη συσκευή. Πάνω από το backend υλοποιήθηκε web εφαρμογή που παρέχει dashboard πραγματικού χρόνου, ιστορικά δεδομένα, διαχείριση κανόνων και ρυθμίσεις λογαριασμού/συσκευής. Τέλος, παρουσιάζεται παράδειγμα λειτουργίας που επαληθεύει την end-to-end ροή: παραμετροποίηση, αποστολή κανόνων και τοπική εκτέλεσή τους στη συσκευή.

«General purpose IoT device»

«Konstantinos Papadopoulos»

Abstract

This thesis presents the design and implementation of a general-purpose Internet of Things (IoT) device intended to be reused across multiple monitoring and control scenarios without redesigning the hardware or rewriting the core firmware for each new application. The proposed solution follows an end-to-end architecture that includes the embedded device (hardware and firmware), the communication layer, a backend service, and a web-based user platform.

On the hardware side, a custom PCB based on an STM32 microcontroller was developed, supporting a wide set of I/O capabilities such as analog and digital inputs/outputs, PWM, relay control, CAN bus expansion, and AC electrical measurements. Connectivity is provided through two alternative channels—WiFi and LTE—allowing operation in local networks as well as remote deployments. Data and commands are exchanged using the MQTT publish/subscribe model.

The firmware adopts a modular component-based structure, separating responsibilities such as sensing, I/O handling, and communications. A key feature of the system is remote parameterization combined with an embedded rules engine, enabling automation logic to be delivered from the platform and executed locally on the device (edge processing). This approach improves autonomy and responsiveness, especially under unstable or unavailable network conditions.

The backend, implemented in Python, subscribes to MQTT telemetry, normalizes incoming payloads, stores historical data in a MySQL/MariaDB database, and performs downlink operations by sending configuration updates and automation rules back to the device. A web application built on top of the backend provides real-time dashboards, historical measurements, rule management, and device/account configuration. Finally, a demonstration scenario validates the full workflow—from configuration and rule creation to device-side execution and real-time visualization.

Ευχαριστίες

Περιεχόμενα

Ευρετήριο Εικόνων

Συντομογραφίες

API	Application Programming Interface
APN	Access Point Name
ARM	Advanced RISC Machines
CAN / CANbus	Controller Area Network
DB	Database (Βάση Δεδομένων)
DMA	Direct Memory Access
ESP	Espressif SoC/Module (WiFi module)
GPIO	General Purpose Input/Output
HTTP	HyperText Transfer Protocol
IDE	Integrated Development Environment
IoT	Internet of Things
JSON	JavaScript Object Notation
LAN	Local Area Network
LTE	Long Term Evolution (4G)
MCU	Microcontroller Unit (Μικροελεγκτής)
MQTT	Message Queuing Telemetry Transport
PWM	Pulse Width Modulation
QoS	Quality of Service (στο MQTT)
REST	Representational State Transfer
RTC	Real-Time Clock
SQL	Structured Query Language
SWD	Serial Wire Debug
UI	User Interface
WAN	Wide Area Network
WiFi	Wireless Fidelity

1. Εισαγωγή

1.1 Αντικείμενο και σκοπός της διπλωματικής

Το αντικείμενο της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η υλοποίηση μίας συσκευής IoT γενικού σκοπού, η οποία μπορεί να χρησιμοποιηθεί σε πλήθος εφαρμογών απομακρυσμένου ελέγχου και εποπτείας, χωρίς να απαιτείται η τροποποίηση του υλικού(hardware) ή λογισμικού(software) για κάθε διαφορετικό σενάριο χρήσης. Η εργασία εντάσσεται στο πεδίο των Εφαρμοσμένων Ηλεκτρονικών και των Ενσωματωμένων Συστημάτων (Embedded Systems), εστιάζοντας στην πρακτική σχεδίαση, υλοποίηση και αξιολόγηση ενός ολοκληρωμένου συστήματος IoT.

Σήμερα, οι περισσότερες διαθέσιμες εμπορικές συσκευές IoT έχουν σχεδιαστεί με στόχο την εξυπηρέτηση συγκεκριμένων εφαρμογών, όπως για παράδειγμα την μέτρηση κατανάλωσης ενέργειας, τον έλεγχο φωτισμού ή την παρακολούθηση περιβαλλοντικών παραμέτρων(θερμοκρασία, υγρασία). Η προσέγγιση αυτή οδηγεί σε συστήματα με περιορισμένη ευελιξία, καθώς η επαναχρησιμοποίησή τους σε διαφορετικές εφαρμογές απαιτεί συχνά αλλαγές στο hardware, στον κώδικα ή και στη διαδικτυακή πλατφόρμα διαχείρισης. Ως αποτέλεσμα, η ανάπτυξη και η συντήρηση τέτοιων λύσεων γίνεται πιο πολύπλοκη και έχει μεγάλο κόστος.

Σκοπός της διπλωματικής εργασίας είναι η ανάπτυξη μίας παραμετροποιήσιμης IoT πλατφόρμας, η οποία αποσυνδέει τη λειτουργικότητα της εφαρμογής από τη φυσική υλοποίηση του υλικού. Η πλακέτα IoT που σχεδιάστηκε για την διπλωματική εργασία βασίζεται σε μικροελεγκτή 32-bit ARM και υποστηρίζει πληθώρα εισόδων και εξόδων, όπως αναλογικές και ψηφιακές εισόδους, μετρήσεις υψηλών και χαμηλών τάσεων, ρεύματος και θερμοκρασίας, καθώς και λειτουργίες ελέγχου μέσω ψηφιακών εξόδων, PWM και ρελέ. Παράλληλα, παρέχεται δυνατότητα επικοινωνίας μέσω WiFi και δικτύου κινητής τηλεφωνίας (LTE), καθώς και υποστήριξη πλακετών επέκτασης (add-on modules) μέσω CANbus για την ενίσχυση της λειτουργικότητας του συστήματος.

Ένας βασικός στόχος της παρούσας εργασίας είναι η ανάπτυξη μηχανισμών παραμετροποίησης, ώστε η λειτουργία της συσκευής να μπορεί να παραμετροποιείται από τον χρήστη μέσω της διαδικτυακής πλατφόρμας. Ο χρήστης έχει τη δυνατότητα να ορίζει τον τρόπο με τον οποίο επεξεργάζονται και παρουσιάζονται οι μετρήσεις, καθώς και να δημιουργεί κανόνες λειτουργίας βασισμένους σε συνθήκες. Οι κανόνες αυτοί εκτελούνται απευθείας στη συσκευή, επιτρέποντας την αυτόνομη λειτουργία της. Με τον τρόπο αυτό, το σύστημα παραμένει ευέλικτο και επεκτάσιμο, χωρίς να απαιτείται επαναπρογραμματισμός του firmware για κάθε νέα εφαρμογή.

Τέλος, η εργασία στοχεύει στη δημιουργία ενός πλήρους end-to-end συστήματος, το οποίο περιλαμβάνει το hardware, το software, το σύστημα επικοινωνίας μέσω MQTT, το backend που είναι υπεύθυνο για την αποθήκευση δεδομένων και ιστοσελίδα. Μέσα από την πρακτική υλοποίηση και επίδειξη της λειτουργίας του συστήματος, αναδεικνύονται τα πλεονεκτήματα μιας IoT αρχιτεκτονικής γενικού σκοπού και η δυνατότητα εφαρμογής της σε διαφορετικά πραγματικά σενάρια.

1.2 Πρόβλημα: IoT συσκευές ειδικού σκοπού και περιορισμοί τους

Η ανάπτυξη των συστημάτων Internet of Things τα τελευταία χρόνια έχει οδηγήσει στην ευρεία διάθεση συσκευών απομακρυσμένης μέτρησης, ελέγχου και εποπτείας, οι οποίες χρησιμοποιούνται σε τομείς όπως η βιομηχανία, η ενέργεια, η γεωργία και τα «έξυπνα» σπίτια. Παρά την πληθώρα των διαθέσιμων λύσεων, η πλειονότητα των εμπορικών IoT συσκευών έχει σχεδιαστεί με γνώμονα την εξυπηρέτηση συγκεκριμένων και προκαθορισμένων εφαρμογών.

Οι συσκευές αυτές χαρακτηρίζονται ως ειδικού σκοπού, καθώς το υλικό, το λογισμικό και η διαδικτυακή πλατφόρμα διαχείρισής τους είναι στενά συνδεδεμένα με το εκάστοτε σενάριο χρήσης. Για παράδειγμα, μία συσκευή σχεδιασμένη για τη μέτρηση κατανάλωσης ηλεκτρικής ενέργειας δεν μπορεί εύκολα να χρησιμοποιηθεί για την παρακολούθηση περιβαλλοντικών παραμέτρων ή για τον έλεγχο εξόδων όπως ρελέ, βαλβίδες, κτλπ, χωρίς να γίνουν σημαντικές τροποποιήσεις. Αντίστοιχα, μία συσκευή που έχει αναπτυχθεί για εφαρμογές αυτοματισμού συχνά δεν υποστηρίζει επαρκώς τη συλλογή και επεξεργασία μετρήσεων υψηλής ακρίβειας.

Ένας από τους βασικούς περιορισμούς των συσκευών ειδικού σκοπού είναι η έλλειψη ευελιξίας. Η αλλαγή ή επέκταση της λειτουργικότητας απαιτεί συνήθως επανασχεδίαση του hardware, τροποποίηση του firmware ή και αλλαγές στη διαδικτυακή πλατφόρμα. Αυτό έχει ως αποτέλεσμα αυξημένο κόστος ανάπτυξης, μεγαλύτερο χρόνο υλοποίησης και δυσκολία στη συντήρηση των συστημάτων, ιδιαίτερα σε περιπτώσεις όπου απαιτείται η υποστήριξη πολλαπλών εφαρμογών ή χρηστών.

Επιπλέον, οι περισσότερες υπάρχουσες λύσεις υιοθετούν μία κλειστή αρχιτεκτονική, όπου ο χρήστης έχει περιορισμένες δυνατότητες παραμετροποίησης. Συνήθως, οι διαθέσιμες επιλογές περιορίζονται στην απλή προβολή μετρήσεων και στη ρύθμιση βασικών παραμέτρων, χωρίς τη δυνατότητα ορισμού σύνθετης λογικής λειτουργίας, όπως είναι η δημιουργία κανόνων οι οποίοι ορίζουν τις εξόδους του συστήματος με βάση τις εισόδους. Ως αποτέλεσμα, η προσαρμογή του συστήματος στις πραγματικές ανάγκες της εφαρμογής καθίσταται δύσκολη ή ανέφικτη.

Ένα ακόμη σημαντικό μειονέκτημα αφορά την ισχυρή εξάρτηση από την κεντρική πλατφόρμα. Σε πολλές περιπτώσεις, η λογική ελέγχου και οι αποφάσεις εκτελούνται αποκλειστικά στο backend, γεγονός που αυξάνει την καθυστέρηση απόκρισης(latency) και καθιστά το σύστημα ευάλωτο σε διακοπές της επικοινωνίας. Η απουσία μηχανισμών τοπικής λήψης αποφάσεων από την ίδια την συσκευή περιορίζει την αξιοπιστία και την αυτονομία των εφαρμογών, ιδιαίτερα σε περιβάλλοντα με ασταθή επικοινωνία.

Συνοψίζοντας, το βασικό πρόβλημα που εντοπίζεται στις υφιστάμενες IoT λύσεις ειδικού σκοπού είναι ο περιορισμένος βαθμός παραμετροποίησης και επεκτασιμότητας, καθώς και η εξαρτημένη σχέση που υπάρχει μεταξύ υλικού, λογισμικού και εφαρμογής. Τα παραπάνω καθιστούν αναγκαία την ανάπτυξη μιας πιο γενικής και ευέλικτης προσέγγισης, η οποία θα επιτρέπει την επαναχρησιμοποίηση της ίδιας υποδομής σε διαφορετικά σενάρια, μειώνοντας το κόστος και την πολυπλοκότητα ανάπτυξης και συντήρησης.

1.3 Στόχοι – Απαιτήσεις συστήματος

Με βάση τα προβλήματα και τους περιορισμούς που αναλύθηκαν στην Ενότητα 1.2, η παρούσα εργασία στοχεύει στη σχεδίαση ενός IoT συστήματος που παραμένει γενικού σκοπού, δηλαδή μπορεί να

προσαρμόζεται σε διαφορετικά σενάρια εγκατάστασης χωρίς αλλαγές στο βασικό υλικό ή επαναπρογραμματισμό για κάθε νέα χρήση. Για να είναι σαφές τι σημαίνει αυτό στην πράξη, οι στόχοι και οι απαιτήσεις οργανώνονται σε: (α) λειτουργικούς στόχους, (β) μη λειτουργικούς στόχους και (γ) περιορισμούς/παραδοχές.

1.3.1 Λειτουργικοί στόχοι

Οι λειτουργικοί στόχοι περιγράφουν τις βασικές δυνατότητες που οφείλει να παρέχει το σύστημα σε επίπεδο συσκευής, επικοινωνίας και πλατφόρμας:

- **Συλλογή δεδομένων από πολλαπλές πηγές**
  Η συσκευή πρέπει να υποστηρίζει συλλογή Αυτό περιλαμβάνει ψηφιακές εισόδους (καταστάσεις), αναλογικές μετρήσεις (τιμές συνεχούς μεταβολής) και μετρήσεις από εξωτερικά υποσυστήματα μέσω διαύλων επικοινωνίας
- **Έλεγχος ενεργοποιητών και εξόδων**
  Το σύστημα πρέπει να επιτρέπει ενεργοποίηση/απενεργοποίηση εξόδων και έλεγχο φορτίων, ώστε να μην περιορίζεται στην παρακολούθηση αλλά να υποστηρίζει και εφαρμογές ελέγχου. Ενδεικτικά, περιλαμβάνονται ψηφιακές έξοδοι, PWM σήματα και έλεγχος ρελέ
- **Αμφίδρομη επικοινωνία μέσω MQTT**
  Η ανταλλαγή δεδομένων πρέπει να υλοποιείται με αμφίδρομο τρόπο:
- **Uplink:** αποστολή τηλεμετρίας (μετρήσεις/καταστάσεις) από τη συσκευή προς το κεντρικό σύστημα
- **Downlink:** λήψη εντολών, ρυθμίσεων και κανόνων από την πλατφόρμα προς τη συσκευή
  Η επικοινωνία βασίζεται στο μοντέλο publish/subscribe, ώστε να είναι εύκολη η δρομολόγηση μεταξύ πολλών συσκευών και χρηστών
- **Υποστήριξη εναλλακτικών καναλιών δικτύου (WiFi και LTE)**
  Η συσκευή πρέπει να μπορεί να λειτουργεί τόσο σε εγκαταστάσεις με τοπικό δίκτυο όσο και σε απομακρυσμένα σημεία χωρίς υποδομή, μέσω LTE. Η ύπαρξη δύο καναλιών καλύπτει διαφορετικές πραγματικές συνθήκες εγκατάστασης και αυξάνει τη δυνατότητα αξιοποίησης της ίδιας συσκευής σε περισσότερα σενάρια
- Διαδικτυακή πλατφόρμα εποπτείας και διαχείρισης
  Απαιτείται web περιβάλλον που να επιτρέπει:
- προβολή τρέχουσας κατάστασης και μετρήσεων
- πρόσβαση σε ιστορικά δεδομένα
- ορισμό/επεξεργασία ρυθμίσεων και κανόνων λειτουργίας
- αποστολή των αλλαγών προς τη συσκευή με ελεγχόμενο τρόπο.
- **Τοπική εκτέλεση κανόνων στη συσκευή**
  Οι κανόνες (π.χ. “αν μια είσοδος είναι ενεργή για Χ χρόνο, ενεργοποίησε μια έξοδο”) πρέπει να εκτελούνται τοπικά στη συσκευή, ώστε το σύστημα να διατηρεί λειτουργικότητα ακόμη και όταν η σύνδεση με το backend δεν είναι διαθέσιμη ή παρουσιάζει καθυστερήσεις.

1.3.2 Μη λειτουργικοί στόχοι

Οι μη λειτουργικοί στόχοι αφορούν την ποιότητα, την επεκτασιμότητα και τη συντηρησιμότητα της λύσης:

- **Παραμετροποίηση χωρίς επαναπρογραμματισμό**
  Η αλλαγή συμπεριφοράς της συσκευής πρέπει να γίνεται μέσω ρυθμίσεων και κανόνων που μεταφέρονται απομακρυσμένα, χωρίς να απαιτείται αλλαγή firmware για κάθε νέα εγκατάσταση.

- **Αφαίρεση (abstraction) μεταξύ hardware και λογικής εφαρμογής**
  Η πλατφόρμα και οι κανόνες πρέπει να "βλέπουν" λογικές οντότητες (π.χ. IN1, OUT1, Vrms) και όχι φυσικές λεπτομέρειες υλοποίησης (pins/καναλιών), ώστε το σύστημα να παραμένει επαναχρησιμοποιήσιμο και ευκολότερα επεκτάσιμο.
- **Επεκτασιμότητα υποσυστημάτων**
  Το σύστημα πρέπει να επιδέχεται επέκταση με πρόσθετα modules (π.χ. μέσω διαύλου όπως CAN), ώστε να υποστηρίζονται νέες λειτουργίες χωρίς αλλαγή της βασικής πλακέτας.
- **Αξιοπιστία επικοινωνίας και ανθεκτικότητα σε αποσυνδέσεις**
  Ιδιαίτερα σε LTE σενάρια, το σύστημα πρέπει να συνεχίζει να λειτουργεί με ασφαλή τρόπο όταν η επικοινωνία διακόπτεται προσωρινά. Αυτό αφορά τόσο τη διατήρηση βασικών λειτουργιών όσο και την ορθή επανασύνδεση/συγχρονισμό όταν το δίκτυο αποκατασταθεί.
- **Υποστήριξη πολλαπλών χρηστών και συσκευών**
  Η αρχιτεκτονική πρέπει να επιτρέπει την ταυτόχρονη λειτουργία πολλών χρηστών/συσκευών, με σαφή απομόνωση δεδομένων και ρυθμίσεων ανά λογαριασμό.
- **Εύκολη συντήρηση και μελλοντική εξέλιξη**
  Σε επίπεδο σχεδίασης, επιδιώκεται δομή που επιτρέπει προσθήκες/αλλαγές χωρίς "σπάσιμο" του συστήματος: καθαρά διαχωρισμένα υποσυστήματα, σταθερά interfaces και δυνατότητα αναβάθμισης επιμέρους τμημάτων.

1.3.3 Περιορισμοί και παραδοχές

Κατά τον σχεδιασμό ελήφθησαν υπόψη συγκεκριμένοι περιορισμοί, ώστε οι επιλογές να παραμένουν ρεαλιστικές ως προς πολυπλοκότητα και κόστος υλοποίησης:

- **Εύρος εφαρμογών**
  Η συσκευή στοχεύει σε εφαρμογές εποπτείας/αυτοματισμού χαμηλής έως μεσαίας πολυπλοκότητας και όχι σε συστήματα αυστηρά πιστοποιημένης ασφάλειας ή υψηλής ακρίβειας μετρητικού επιπέδου.
- **Αρχιτεκτονική firmware χωρίς RTOS**
  Δεν χρησιμοποιείται λειτουργικό πραγματικού χρόνου, ώστε να διατηρηθεί η υλοποίηση απλή και προβλέψιμη ως προς τους πόρους και τη συντήρηση, ενώ οι απαιτήσεις χρόνου εξυπηρετούνται με δομημένο scheduling/interrupts.
- **Ασφάλεια επικοινωνίας**
  Η πλήρης υλοποίηση ισχυρών μηχανισμών κρυπτογράφησης/πιστοποίησης (π.χ. TLS end-to-end) δεν αποτελεί κύριο αντικείμενο της εργασίας, αλλά αναγνωρίζεται ως δυνατότητα μελλοντικής επέκτασης, ειδικά για παραγωγικά περιβάλλοντα.

1.4 Περιγραφή προτεινόμενης λύσης

Για την αντιμετώπιση των προβλημάτων που παρουσιάστηκαν στις προηγούμενες ενότητες, στην παρούσα διπλωματική εργασία προτείνεται και υλοποιείται μία ολοκληρωμένη IoT πλατφόρμα γενικού σκοπού, η οποία συνδυάζει hardware, firmware(STM32), σύστημα επικοινωνίας, backend υποδομή με χρήση python και website μέσω του οποίο του οποίο θα μπορεί να χρήστης να αλληλοεπιδρά με την συσκευή IoT. Η προτεινόμενη λύση σχεδιάστηκε με γνώμονα την ευελιξία, την παραμετροποίηση και την επεκτασιμότητα, ώστε να μπορεί να προσαρμοστεί σε διαφορετικά σενάρια εφαρμογής χωρίς αλλαγές στο βασικό σύστημα.

Στο επίπεδο του υλικού, αναπτύχθηκε μία κεντρική πλακέτα με μικροελεγκτή 32-bit ARM, η οποία ενσωματώνει πληθώρα διεπαφών εισόδου και εξόδου. Η πλακέτα υποστηρίζει αναλογικές και ψηφιακές εισόδους, μετρήσεις υψηλών και χαμηλών τάσεων, ρεύματος και θερμοκρασίας, καθώς και λειτουργίες

ελέγχου μέσω ψηφιακών εξόδων, PWM και ρελέ. Παράλληλα, παρέχεται δυνατότητα σύνδεσης πλακετών επέκτασης μέσω CANbus. Οι πλακέτες επικοινωνίας WiFi και LTE/4G κουμπώνουν στην κύρια πλακέτα ως add-on modules, ώστε να μπορούν να αφαιρεθούν σε περιπτώσεις που δεν απαιτούνται από την εφαρμογή, όπως για παράδειγμα μέτρηση ρεύματος μίας συσκευής σπιτιού όπου το δίκτυο WiFi είναι διαθέσιμο.

Σε επίπεδο firmware, υλοποιήθηκε μία αρχιτεκτονική βασισμένη σε ανεξάρτητα components, όπου κάθε υποσύστημα είναι ανεξάρτητο κομμάτι (π.χ. ψηφιακές είσοδοι/έξοδοι, αναλογικές μετρήσεις, επικοινωνία, έλεγχος PWM). Η προσέγγιση αυτή επιτρέπει την ευκολότερη συντήρηση του κώδικα, καθώς και την επέκταση της λειτουργικότητας χωρίς σημαντικές αλλαγές στον υπάρχοντα σχεδιασμό.

Η επικοινωνία της συσκευής με το κεντρικό σύστημα πραγματοποιείται μέσω του πρωτοκόλλου MQTT, ακολουθώντας το μοντέλο publish/subscribe. Οι μετρήσεις και οι καταστάσεις της συσκευής δημοσιεύονται σε προκαθορισμένα topics, ενώ οι εντολές και οι κανόνες λειτουργίας αποστέλλονται στη συσκευή μέσω αντίστοιχων topics. Η χρήση topic-based αρχιτεκτονικής επιτρέπει τον διαχωρισμό δεδομένων ανά χρήστη και συσκευή, υποστηρίζοντας την ταυτόχρονη λειτουργία πολλαπλών εγκαταστάσεων.

Στο επίπεδο του backend, αναπτύχθηκε κώδικας σε γλώσσα Python, η οποία λειτουργεί ως ενδιάμεσος κόμβος μεταξύ των συσκευών IoT και της βάσης δεδομένων. Η υπηρεσία αυτή λαμβάνει τα δεδομένα μέσω MQTT, τα επεξεργάζεται και τα αποθηκεύει σε βάση δεδομένων SQL, διατηρώντας ιστορικό μετρήσεων και ρυθμίσεων για κάθε χρήστη. Πάνω από το backend, υλοποιήθηκε διαδικτυακή εφαρμογή, μέσω της οποίας ο χρήστης μπορεί να παρακολουθεί τις μετρήσεις, να παραμετροποιεί τη συμπεριφορά της συσκευής και να ορίζει κανόνες λειτουργίας.

Η συνολική αρχιτεκτονική της προτεινόμενης λύσης ακολουθεί μία προσέγγιση από άκρο σε άκρο (end-to-end), όπου η ίδια συσκευή μπορεί να χρησιμοποιηθεί σε διαφορετικές εφαρμογές απλώς με αλλαγή παραμέτρων. Η αλλαγή παραμέτρων είναι εφικτή μέσω της σειριακής θύρας που υπάρχει στην συσκευή IoT. Μέσα από την πρακτική υλοποίηση και επίδειξη του συστήματος, αναδεικνύεται η δυνατότητα δημιουργίας μιας ευέλικτης και επεκτάσιμης IoT πλατφόρμας, κατάλληλης για εφαρμογές απομακρυσμένης εποπτείας, ελέγχου και αυτοματισμού.

1.5 Συνεισφορά και καινοτομία της εργασίας

Η παρούσα διπλωματική εργασία συνεισφέρει στον τομέα των ενσωματωμένων IoT συστημάτων μέσω της σχεδίασης και υλοποίησης μίας ολοκληρωμένης πλατφόρμας γενικού σκοπού, η οποία διαφοροποιείται από τις υπάρχουσες λύσεις ειδικού σκοπού τόσο σε επίπεδο αρχιτεκτονικής όσο και σε επίπεδο λειτουργικότητας. Η κύρια καινοτομία της εργασίας έγκειται στη συνδυαστική προσέγγιση hardware, firmware και διαδικτυακής πλατφόρμας, με έμφαση στην παραμετροποίηση και την επαναχρησιμοποίηση.

Συγκεκριμένα, οι βασικές συνεισφορές της εργασίας συνοψίζονται ως εξής:

- Σχεδιάστηκε και υλοποιήθηκε μία IoT συσκευή γενικού σκοπού, βασισμένη σε μικροελεγκτή 32-bit ARM, η οποία υποστηρίζει πληθώρα εισόδων και εξόδων και μπορεί να χρησιμοποιηθεί σε διαφορετικά σενάρια εφαρμογής χωρίς τροποποίηση του βασικού υλικού.

- Αναπτύχθηκε modular αρχιτεκτονική υλικού, με δυνατότητα σύνδεσης πλακετών επέκτασης και add-on μονάδων επικοινωνίας (WiFi και LTE), επιτρέποντας την προσαρμογή της συσκευής σε διαφορετικές απαιτήσεις εγκατάστασης.
- Υλοποιήθηκε firmware βασισμένο σε ανεξάρτητα components, τα οποία αντιστοιχούν στα επιμέρους υποσυστήματα του υλικού, διευκολύνοντας τη συντήρηση και τη μελλοντική επέκταση της λειτουργικότητας.
- Εισήχθη μηχανισμός παραμετροποίησης μετρήσεων, μέσω του οποίου οι ακατέργαστες τιμές των αισθητήρων μετατρέπονται σε φυσικές μονάδες με χρήση μαθηματικών μετασχηματισμών που ορίζονται από τον χρήστη, χωρίς αλλαγή στο firmware.
- Υλοποιήθηκε μηχανισμός κανόνων στο firmware, επιτρέποντας την τοπική εκτέλεση κανόνων λειτουργίας βασισμένων σε συνθήκες, μειώνοντας την εξάρτηση από το κεντρικό σύστημα και βελτιώνοντας τον χρόνο απόκρισης.
- Αναπτύχθηκε πλήρης πλατφόρμα backend και διαδικτυακή εφαρμογή, με δυνατότητα διαχείρισης πολλαπλών χρηστών και συσκευών μέσω topic-based αρχιτεκτονικής στο MQTT, εξασφαλίζοντας διαχωρισμό δεδομένων και ρυθμίσεων.
- Παρουσιάστηκε και υλοποιήθηκε πλακέτα επέκτασης ψηφιακών εισόδων/εξόδων και ρελέ, επιβεβαιώνοντας στην πράξη την επεκτασιμότητα της προτεινόμενης αρχιτεκτονικής.

Η συνολική προσέγγιση της εργασίας εστιάζει στη γεφύρωση του κενού μεταξύ εξειδικευμένων IoT λύσεων και γενικών, επεκτάσιμων συστημάτων, προσφέροντας μία πρακτική και εφαρμόσιμη αρχιτεκτονική που μπορεί να αποτελέσει βάση για μελλοντικές επεκτάσεις και εμπορικές εφαρμογές.

1.6 Δομή της διπλωματικής

Η παρούσα διπλωματική εργασία οργανώνεται σε εννέα κεφάλαια, τα οποία καλύπτουν σταδιακά το θεωρητικό υπόβαθρο, τη σχεδίαση, την υλοποίηση και την αξιολόγηση της προτεινόμενης IoT πλατφόρμας.

Κεφάλαιο 1 – Εισαγωγή: παρουσιάζεται το αντικείμενο της εργασίας, το πρόβλημα που αντιμετωπίζεται, οι στόχοι και οι απαιτήσεις του συστήματος, καθώς και η συνολική προσέγγιση.

Κεφάλαιο 2 – Θεωρητικό υπόβαθρο και τεχνολογίες: καλύπτονται οι βασικές έννοιες των ενσωματωμένων συστημάτων και του IoT, η αρχιτεκτονική μικροελεγκτών ARM Cortex-M, τα κύρια περιφερειακά που αξιοποιούνται σε embedded εφαρμογές, καθώς και οι τεχνολογίες επικοινωνίας (LAN/WAN) και το πρωτόκολλο MQTT. Επιπλέον, παρουσιάζονται οι γενικές αρχές backend και διαδικτυακών πλατφορμών εποπτείας και ελέγχου.

Κεφάλαιο 3 – Αρχιτεκτονική συστήματο: περιγράφεται η συνολική αρχιτεκτονική (device–communication–backend–web), οι ροές uplink/downlink και η απομόνωση ανά χρήστη/συσκευή. Στη συνέχεια αναλύεται το hardware της συσκευής (τροφοδοσία, είσοδοι/έξοδοι, υποσυστήματα μέτρησης και επικοινωνίας), καθώς και οι βασικές επιλογές σχεδίασης που επηρεάζουν τη λειτουργία και την επεκτασιμότητα.

Κεφάλαιο 4 – Παραμετροποίηση και abstraction του συστήματος: παρουσιάζεται η μεθοδολογία με την οποία επιτυγχάνεται "γενικού σκοπού" λειτουργία, δηλαδή ο διαχωρισμός φυσικών σημάτων από λογικές οντότητες, η αντιστοίχιση μετρήσεων/εισόδων/εξόδων σε αναγνωριστικά (IDs) και η περιγραφή

κανόνων αυτοματισμού με δομημένο τρόπο, ώστε η συσκευή να προσαρμόζεται σε διαφορετικές εφαρμογές χωρίς αλλαγές στον βασικό κώδικα.

Κεφάλαιο 5 – Υλοποίηση Firmware της συσκευή IoT: αναλύεται η αρχιτεκτονική του λογισμικού της συσκευής, οι μηχανισμοί συλλογής μετρήσεων, ο χειρισμός εισόδων/εξόδων, η τοπική εκτέλεση κανόνων και η διαχείριση επικοινωνίας. Περιγράφονται επίσης οι κρίσιμες επιλογές για αξιοπιστία (buffers, χρονισμός, αποθήκευση ρυθμίσεων) και ο τρόπος με τον οποίο το firmware αλληλεπιδρά με τα μηνύματα που στέλνει/λαμβάνει μέσω MQTT.

Κεφάλαιο 6 – Backend και βάση δεδομένων: παρουσιάζεται το backend σε Python, ο μηχανισμός λήψης τηλεμετρίας μέσω MQTT, η διαδικασία parsing/κανονικοποίησης των δεδομένων, καθώς και η αποθήκευση σε βάση δεδομένων. Επιπλέον, αναλύεται ο μηχανισμός downlink για αποστολή ρυθμίσεων και κανόνων προς τη συσκευή και η υποστήριξη real-time ενημέρωσης της πλατφόρμας.

Κεφάλαιο 7 – Web πλατφόρμα και διεπαφή χρήστη: περιγράφεται η δομή της web εφαρμογής, η αυθεντικοποίηση/πολυχρηστικότητα, και οι βασικές σελίδες λειτουργίας (dashboard, ιστορικά δεδομένα, κανόνες, ρυθμίσεις). Αναλύεται επίσης η ανταλλαγή δεδομένων μέσω REST endpoints και real-time μηχανισμών, καθώς και ο τρόπος με τον οποίο ενέργειες του χρήστη μετατρέπονται σε εντολές προς τη συσκευή.

Κεφάλαιο 8 – Παράδειγμα λειτουργίας: παρουσιάζεται ενδεικτικά ένα σενάριο λειτουργίας της πλατφόρμας με βάση ένα σενάριο εφαρμογής. Αναλύεται η παραμετροποίηση και παρατίθενται μετρήσεις που τεκμηριώνουν την ορθή λειτουργία.

Κεφάλαιο 9 – Προβλήματα και αντιμετώπιση τους: σε αυτό το κεφάλαιο παρουσιάζονται τα προβλήματα που παρουσιάστηκαν και ο τρόπος με τον οποίο αντιμετωπίστηκαν

Κεφάλαιο 10 – Συμπεράσματα και προτάσεις βελτίωσης: συνοψίζονται τα συμπεράσματα, αξιολογείται ο βαθμός επίτευξης των στόχων και προτείνονται κατευθύνσεις για βελτίωση ή επέκταση, όπως ενίσχυση ασφάλειας επικοινωνίας, υποστήριξη πρόσθετων τύπων μετρήσεων/εξόδων και περαιτέρω αυτοματοποίηση της παραμετροποίησης.

2. Θεωρητικό υπόβαθρο και τεχνολογίες

Το παρόν κεφάλαιο παρουσιάζει το θεωρητικό υπόβαθρο και τις βασικές τεχνολογίες που αξιοποιήθηκαν για τη σχεδίαση και υλοποίηση της διπλωματικής εργασίας. Η ανάλυση επικεντρώνεται στις αρχές των ενσωματωμένων συστημάτων IoT, στις αρχιτεκτονικές επικοινωνίας, καθώς και στις τεχνολογίες που σχετίζονται με τη συλλογή, τη μετάδοση και την απεικόνιση δεδομένων.

Στόχος του κεφαλαίου δεν είναι η εκτενής θεωρητική ανάλυση, αλλά η παρουσίαση του τεχνολογικού πλαισίου, ώστε να καταστεί σαφές το υπόβαθρο πάνω στο οποίο βασίστηκαν οι σχεδιαστικές επιλογές της διπλωματικής εργασίας.

2.1 Embedded IoT συστήματα: βασικές έννοιες και αρχιτεκτονικές

Τα ενσωματωμένα συστήματα (Embedded Systems) αποτελούν εξειδικευμένα υπολογιστικά συστήματα τα οποία είναι σχεδιασμένα για την εκτέλεση συγκεκριμένων λειτουργιών, συνήθως ενσωματωμένα σε ευρύτερα ηλεκτρονικά ή ηλεκτρομηχανολογικά συστήματα. Τα ενσωματωμένα συστήματα χαρακτηρίζονται από περιορισμένους πόρους σε επεξεργαστική ισχύ, μνήμη και κατανάλωση ενέργειας, ενώ συχνά λειτουργούν σε πραγματικό ή σχεδόν πραγματικό χρόνο.

Η έννοια του Internet of Things επεκτείνει τον ρόλο των ενσωματωμένων συστημάτων, προσθέτοντας δυνατότητες δικτύωσης και απομακρυσμένης επικοινωνίας. Ένα ενσωματωμένο IoT σύστημα συνδυάζει αισθητήρες, ενεργοποιητές (actuators), μικροελεγκτή και διεπαφές επικοινωνίας, επιτρέποντας τη συλλογή δεδομένων από το φυσικό περιβάλλον, την επεξεργασία τους και την αποστολή τους σε απομακρυσμένα συστήματα(server).

2.1.1 Βασικά δομικά στοιχεία ενός IoT embedded συστήματος

Ένα τυπικό ενσωματωμένο IoT σύστημα αποτελείται από τα ακόλουθα βασικά δομικά στοιχεία:

- **Αισθητήρες (Sensors):** Συσκευές που μετατρέπουν φυσικά μεγέθη, όπως τάση, ρεύμα, θερμοκρασία ή πίεση, σε ηλεκτρικά σήματα κατάλληλα για επεξεργασία.
- **Μικροελεγκτής (MCU):** Ο πυρήνας του συστήματος, υπεύθυνο για τη συλλογή δεδομένων, την εκτέλεση αλγορίθμων και τον έλεγχο των εξόδων
- **Ενεργοποιητές (Actuators):** Στοιχεία που επιτρέπουν στο σύστημα να επηρεάζει το φυσικό περιβάλλον, όπως ρελέ, ψηφιακές έξοδοι ή PWM drivers που οδηγούν Transistor/Mosfet.
- **Διεπαφές επικοινωνίας:** Τρόποι δικτύωσης, όπως WiFi, Ethernet, Lora ή GSM/LTE, μέσω των οποίων το σύστημα επικοινωνεί με απομακρυσμένους συστήματα-servers.
- **Τροφοδοσία:** Η παροχή ενέργειας, το οποίο συχνά πρέπει να υποστηρίζει πολλαπλά επίπεδα τάσης και να εξασφαλίζει την σταθερή λειτουργία του συστήματος.

2.1.2 Αρχιτεκτονικές IoT συστημάτων

Οι αρχιτεκτονικές IoT συστημάτων μπορούν γενικά να διαχωριστούν σε τρία βασικά επίπεδα:

- **Επίπεδο συσκευής (Device Layer):** Περιλαμβάνει τις συσκευές που συλλέγουν δεδομένα και εκτελούν τοπικές λειτουργίες.

- **Επίπεδο δικτύου και επικοινωνίας (Communication Layer):** Υπεύθυνο για τη μετάδοση δεδομένων μεταξύ συσκευών και κεντρικών συστημάτων, χρησιμοποιώντας πρωτόκολλα όπως MQTT ή HTTP.
- **Επίπεδο εφαρμογής (Application Layer):** Περιλαμβάνει τα backend συστήματα, τις βάσεις δεδομένων και τις διαδικτυακές εφαρμογές που παρουσιάζουν και επεξεργάζονται τα δεδομένα.

Στις περισσότερες σύγχρονες IoT εφαρμογές, η λογική του συστήματος υλοποιείται κυρίως στο επίπεδο εφαρμογής, με τις συσκευές να λειτουργούν ως απλοί συλλέκτες δεδομένων. Ωστόσο, η προσέγγιση αυτή παρουσιάζει περιορισμούς σε θέματα καθυστέρησης απόκρισης και αξιοπιστίας, ιδιαίτερα σε περιπτώσεις όπου η επικοινωνία μεταξύ IoT συσκευής και server δεν είναι πάντα εφικτή ή σταθερή(π.χ. πρόβλημα στον πάροχο του δικτύου).

2.1.3 Edge processing και τοπική λήψη αποφάσεων

Μία εναλλακτική προσέγγιση, η οποία υιοθετείται και στην παρούσα εργασία, είναι η μεταφορά μέρους της λογικής ελέγχου στο επίπεδο της συσκευής (edge processing). Με αυτήν την αρχιτεκτονική, το ενσωματομένο σύστημα δεν περιορίζεται στη συλλογή δεδομένων, αλλά εκτελεί τοπικά κανόνες και αποφάσεις, μειώνοντας την εξάρτηση από το κεντρικό σύστημα.

Η τοπική λήψη αποφάσεων βελτιώνει τον χρόνο απόκρισης και επιτρέπει την αυτόνομη-ασφαλή λειτουργία της συσκευής, ακόμη και σε περιπτώσεις προσωρινής απώλειας σύνδεσης. Η προσέγγιση αυτή είναι ιδιαίτερα σημαντική για εφαρμογές αυτοματισμού και ελέγχου, όπου απαιτείται άμεση αντίδραση σε μεταβολές των μετρούμενων μεγεθών.

2.2 Μικροελεγκτές ARM Cortex-M και επιλογή STM32

Οι μικροελεγκτές αρχιτεκτονικής ARM Cortex-M αποτελούν μία από τις πιο διαδεδομένες πλατφόρμες για εφαρμογές ενσωματωμένων συστημάτων και IoT, λόγω της ισορροπίας που προσφέρουν μεταξύ υπολογιστικής ισχύος, κατανάλωσης ενέργειας και διαθεσιμότητας περιφερειακών. Η οικογένεια Cortex-M έχει σχεδιαστεί ειδικά για embedded εφαρμογές και περιλαμβάνει επεξεργαστικούς πυρήνες όπως οι Cortex-M0, M3, M4 και M7, οι οποίοι καλύπτουν διαφορετικά επίπεδα απόδοσης και πολυπλοκότητας.

Οι μικροελεγκτές Cortex-M υποστηρίζουν αρχιτεκτονική 32-bit, γεγονός που επιτρέπει την αποδοτική επεξεργασία δεδομένων, την υλοποίηση πιο σύνθετων αλγορίθμων και τη διαχείριση πολλαπλών περιφερειακών. Παράλληλα, προσφέρουν χαμηλή κατανάλωση ισχύος, καθιστώντας τους κατάλληλους για εφαρμογές που λειτουργούν συνεχώς ή σε απομακρυσμένες εγκαταστάσεις.

2.2.1 Χαρακτηριστικά αρχιτεκτονικής ARM Cortex-M

Βασικά χαρακτηριστικά των μικροελεγκτών ARM Cortex-M περιλαμβάνουν:

- Αρχιτεκτονική 32-bit με pipeline σχεδιασμό για γρήγορη εκτέλεση εντολών
- Υποστήριξη μεγάλου αριθμού περιφερειακών, όπως GPIO, ADC, timers, PWM και σειριακών τρόπων επικοινωνίας (UART, SPI, I²C, CAN)
- Μηχανισμούς interrupts (NVIC), οι οποίοι επιτρέπουν άμεση απόκριση σε εξωτερικά γεγονότα
- Υποστήριξη λειτουργιών χαμηλής κατανάλωσης (sleep και low-power modes)

- Ευρεία υποστήριξη από εργαλεία ανάπτυξης(STM32CubeMX), βιβλιοθήκες(HAL/LL) και κοινότητα(forum)

Τα χαρακτηριστικά αυτά καθιστούν τους Cortex-M ιδιαίτερα κατάλληλους για συστήματα IoT, όπου απαιτείται ταυτόχρονη διαχείριση αισθητήρων, επικοινωνιών και διαχείριση των εξόδων του συστήματος(π.χ. ενεργοποίηση βαλβιδών).

2.2.2 Επιλογή οικογένειας STM32

Στην διπλωματική επιλέχθηκε μικροελεγκτής της οικογένειας STM32 της εταιρείας STMicroelectronics. Η επιλογή αυτή βασίστηκε σε μία σειρά τεχνικών και πρακτικών κριτηρίων, τα οποία συνδέονται άμεσα με τις απαιτήσεις της εφαρμογής.

Οι μικροελεγκτές STM32 προσφέρουν:

- Μεγάλη ποικιλία σειρών και μοντέλων, καλύπτοντας διαφορετικά επίπεδα απόδοσης και κόστους
- Πλούσιο σύνολο ενσωματωμένων περιφερειακών, απαραίτητων για την υποστήριξη πολλαπλών εισόδων και εξόδων
- Υποστήριξη προηγμένων λειτουργιών, όπως DMA για αποδοτική μεταφορά δεδομένων και timers για ακριβή έλεγχο PWM
- Μεγάλη υποστήριξη μέσω εργαλείων ανάπτυξης, όπως το STM32CubeMX και οι βιβλιοθήκες HAL/LL
- Ευρεία βιομηχανική χρήση και μακροχρόνια διαθεσιμότητα
- Κοινή αρχιτεκτονική μεταξύ των μικροελεγκτών, πράγμα που καθιστά εύκολη την αντικατάσταση ενός μοντέλου με ένα άλλο σε περίπτωση που χρειαστεί.

Η χρήση των εργαλείων STM32CubeMX διευκολύνει την αρχικοποίηση των περιφερειακών του μικροελεγκτή, μειώνοντας τον χρόνο ανάπτυξης και τα σφάλματα υλοποίησης.

2.2.3 Καταλληλότητα για γενικού σκοπού IoT συσκευή

Οι μικροελεγκτές της οικογένειας STM32 προσφέρουν χαρακτηριστικά που τους καθιστούν κατάλληλους για γενικού σκοπού IoT εφαρμογές, όπου απαιτείται συνδυασμός συλλογής δεδομένων, ελέγχου σημάτων και διασύνδεσης με εξωτερικές μονάδες. Ενδεικτικά:

- **Πολλαπλές διεπαφές εισόδου/εξόδου:** ψηφιακές γραμμές, αναλογικοί μετατροπείς (ADC) και χρονιστές (timers) επιτρέπουν την άμεση σύνδεση με αισθητήρες και κυκλώματα διεπαφής
- **Υποστήριξη επικοινωνίας με εξωτερικά υποσυστήματα:** η ύπαρξη πολλαπλών σειριακών διεπαφών παρέχει ευελιξία για σύνδεση με μονάδες επικοινωνίας ή άλλα περιφερειακά
- **Αποτελεσματική διαχείριση πραγματικού χρόνου:** μηχανισμοί interrupts και timers διευκολύνουν την αξιόπιστη απόκριση σε γεγονότα και την εκτέλεση περιοδικών εργασιών
- **Υποστήριξη DMA:** μειώνουν τον φόρτο του CPU σε περιπτώσεις συνεχούς ροής δεδομένων ή παράλληλων λειτουργιών
- **Επαρκής μνήμη προγράμματος και δεδομένων:** επιτρέπει την υλοποίηση πιο σύνθετων λειτουργιών χωρίς λειτουργικούς περιορισμούς σε βιβλιοθήκες, πρωτόκολλα και λογική εφαρμογής.

Με βάση τα παραπάνω, η επιλογή STM32 επιτρέπει την ανάπτυξη μιας πλατφόρμας που μπορεί να προσαρμοστεί σε διαφορετικά σενάρια χρήσης, καθώς ο μικροελεγκτής λειτουργεί ως κοινό υπόβαθρο για μετρήσεις, έλεγχο εξόδων και επικοινωνία. Η συγκεκριμένη αξιοποίηση των περιφερειακών και η αντιστοίχισή τους με τις λειτουργίες της συσκευής παρουσιάζονται αναλυτικά στα κεφάλαια σχεδίασης και υλοποίησης.

2.3 Περιφερειακά μικροελεγκτή και ρόλος τους σε embedded/IoT συστήματα

Οι μικροελεγκτές σύγχρονων ενσωματωμένων συστημάτων δεν αποτελούνται μόνο από τον επεξεργαστικό πυρήνα, αλλά ενσωματώνουν ένα πλήθος περιφερειακών (peripherals) που επιτρέπουν τη διασύνδεση με το φυσικό περιβάλλον και με εξωτερικά υποσυστήματα. Σε IoT εφαρμογές, τα περιφερειακά αυτά είναι καθοριστικά, καθώς υποστηρίζουν λειτουργίες όπως συλλογή μετρήσεων, παραγωγή σημάτων ελέγχου, ακριβή χρονισμό, ανταλλαγή δεδομένων και αξιόπιστη απόκριση σε εξωτερικά γεγονότα. Στις επόμενες υποενότητες παρουσιάζονται συνοπτικά τα βασικότερα περιφερειακά που συναντώνται σε μικροελεγκτές ARM Cortex-M και ο ρόλος τους σε τυπικές εφαρμογές.

2.3.1 GPIO (General Purpose Input/Output)

Τα GPIO αποτελούν τις βασικές ψηφιακές γραμμές εισόδου/εξόδου ενός μικροελεγκτή. Μπορούν να αρχικοποιηθούν ως είσοδοι (ανάγνωση λογικών καταστάσεων), έξοδοι (οδήγηση σημάτων) ή και ως εναλλακτικές λειτουργίες (alternate functions) για σύνδεση με άλλα περιφερειακά (π.χ. UART, SPI, timers). Συνήθεις δυνατότητες είναι οι εσωτερικές αντιστάσεις pull-up/pull-down, η επιλογή ταχύτητας οδήγησης και η παραγωγή interrupts σε αλλαγές στάθμης, στοιχεία που βοηθούν στη διασύνδεση με διακόπτες, αισθητήρες ή ψηφιακά σήματα.

2.3.2 ADC (Analog-to-Digital Converter)

Ο ADC μετατρέπει αναλογικές τάσεις σε ψηφιακές τιμές, επιτρέποντας τη μέτρηση μεγεθών όπως τάση αισθητήρων, ρεύματα(μέσω κατάλληλης μετατροπής), ή άλλες αναλογικές παραμέτρους. Η αξιοπιστία των μετρήσεων εξαρτάται από την ανάλυση (bits), τον ρυθμό δειγματοληψίας, την αναφορά τάσης (reference) και τη σωστή προσαρμογή σήματος (signal conditioning). Σε πρακτικές εφαρμογές χρησιμοποιούνται τεχνικές όπως oversampling, μέσος όρος ή ψηφιακό φιλτράρισμα, ώστε να μειωθεί η επίδραση θορύβου και να βελτιωθεί η σταθερότητα των μετρήσεων.

2.3.3 Timers/Counters και χρονισμός

Οι timers/counters παρέχουν βασικές λειτουργίες χρονισμού και μέτρησης χρόνου. Μπορούν να χρησιμοποιηθούν για:

- παραγωγή περιοδικών interrupts (π.χ. κάθε 1 ms),
- μέτρηση χρονικών διαστημάτων,
- καταμέτρηση παλμών,
- και υποστήριξη εξειδικευμένων λειτουργιών όπως PWM και input capture.

Σε IoT συστήματα, οι timers είναι κρίσιμοι για σταθερούς ρυθμούς δειγματοληψίας, για scheduling εργασιών και για την ακριβή χρονική οργάνωση της εφαρμογής χωρίς να καταναλώνεται επεξεργαστική ισχύς από την CPU.

2.3.4 PWM (Pulse Width Modulation)

Το PWM είναι τεχνική παραγωγής ψηφιακού σήματος με ρυθμιζόμενο duty cycle, ώστε να προσομοιάζει αναλογικό έλεγχο ισχύος. Παράγεται συνήθως από timers και χρησιμοποιείται για έλεγχο φωτισμού (dimming), ταχύτητας κινητήρων, σερβομηχανισμών ή άλλων φορτίων μέσω κατάλληλης βαθμίδας οδήγησης. Τα βασικά χαρακτηριστικά του PWM είναι η συχνότητα και ο κύκλος λειτουργίας (duty cycle), τα οποία καθορίζουν την αποτελεσματική μέση ισχύ που παραδίδεται στο φορτίο.

2.3.5 Interrupts και NVIC (διαχείριση γεγονότων)

Τα interrupts επιτρέπουν στον μικροελεγκτή να ανταποκρίνεται άμεσα σε γεγονότα (π.χ. αλλαγή εισόδου, ολοκλήρωση μεταφοράς δεδομένων, λήξη timer) χωρίς συνεχή polling. Ο ελεγκτής διακοπών NVIC (Nested Vectored Interrupt Controller) οργανώνει τις διακοπές με προτεραιότητες, επιτρέποντας την ταυτόχρονη εξυπηρέτηση πολλών πηγών γεγονότων. Η σωστή χρήση interrupts βελτιώνει την απόδοση και την προβλεψιμότητα σε εφαρμογές πραγματικού χρόνου, ενώ μειώνει και την κατανάλωση, αφού επιτρέπει στη CPU να βρίσκεται σε χαμηλή ισχύ όταν δεν υπάρχουν γεγονότα προς επεξεργασία.

2.3.6 DMA (Direct Memory Access)

Το DMA επιτρέπει τη μεταφορά δεδομένων μεταξύ μνήμης και περιφερειακών χωρίς να εκτελείται για κάθε byte αντίστοιχος κώδικας από την CPU. Αυτό μειώνει τον φόρτο του επεξεργαστή και επιτρέπει υψηλότερους ρυθμούς μεταφοράς με μικρότερη καθυστέρηση. Χρησιμοποιείται συχνά σε δειγματοληψίες ADC, σε λήψη/μετάδοση UART/USART, καθώς και σε μεταφορές SPI, ειδικά όταν απαιτείται συνεχής ροή δεδομένων ή ταυτόχρονη εκτέλεση άλλων εργασιών.

2.3.7 UART/USART (σειριακή επικοινωνία)

Η UART/USART είναι από τις πιο διαδεδομένες διεπαφές σειριακής επικοινωνίας σε embedded συστήματα. Χρησιμοποιείται για επικοινωνία με υπολογιστή (debugging), με εξωτερικές μονάδες (π.χ. modems, GNSS, αισθητήρες) ή με άλλα μικροελεγκτικά συστήματα. Τα βασικά στοιχεία ρύθμισης περιλαμβάνουν baud rate, αριθμό data bits, parity και stop bits. Σε εφαρμογές με αυξημένες απαιτήσεις αξιοπιστίας, χρησιμοποιούνται κυκλώματα ελέγχου ροής (flow control) και μηχανισμοί ανίχνευσης σφαλμάτων.

2.3.8 I²C (Inter-Integrated Circuit)

Το I²C είναι δίαυλος δύο γραμμών (SCL/SDA) που επιτρέπει σύνδεση πολλών συσκευών στο ίδιο bus μέσω διευθυνσιοδότησης. Χρησιμοποιείται ευρέως για αισθητήρες, EEPROMs, RTCs και ολοκληρωμένα ελέγχου. Πλεονεκτεί όταν απαιτείται απλή καλωδίωση και μέτριοι ρυθμοί μετάδοσης, ενώ η ύπαρξη τυποποιημένων registers σε πολλές συσκευές κάνει την υλοποίησή του σχετικά άμεση.

2.3.9 CAN (Controller Area Network)

Το CAN είναι βιομηχανικός δίαυλος επικοινωνίας σχεδιασμένος για ανθεκτικότητα σε θόρυβο και αξιόπιστη πολυ-κομβική λειτουργία. Υποστηρίζει μηχανισμούς προτεραιότητας μηνυμάτων(arbitration), ανίχνευση σφαλμάτων και διαφορική μετάδοση (CANH/CANL), στοιχεία που συμβάλλουν σε σταθερή

επικοινωνία σε απαιτητικά περιβάλλοντα. Χρησιμοποιείται σε εφαρμογές όπου πολλοί κόμβοι ανταλλάσσουν μηνύματα μικρού/μεσαίου μεγέθους με υψηλή αξιοπιστία.[1]

Τα παραπάνω περιφερειακά αποτελούν τα βασικά εργαλεία με τα οποία ένας μικροελεγκτής αλληλεπιδρά με το φυσικό περιβάλλον και με εξωτερικά υποσυστήματα. Στα επόμενα κεφάλαια παρουσιάζεται η αξιοποίησή τους στη σχεδίαση και υλοποίηση της προτεινόμενης IoT συσκευής.

2.4 Επικοινωνίες σε IoT συστήματα

Η επικοινωνία αποτελεί θεμελιώδες στοιχείο κάθε IoT συστήματος, καθώς επιτρέπει τη μεταφορά τηλεμετρίας από τις συσκευές προς κεντρικά συστήματα (backend) και, αντίστροφα, τη μεταφορά εντολών/ρυθμίσεων από την πλατφόρμα προς τη συσκευή. Η επιλογή της τεχνολογίας πρόσβασης στο δίκτυο (π.χ. WiFi ή LTE) και του application-layer πρωτοκόλλου (π.χ. MQTT ή HTTP) επηρεάζει άμεσα την αξιοπιστία, την καθυστέρηση απόκρισης, το λειτουργικό κόστος και τη συνολική ευελιξία της λύσης.

2.4.1 Απαιτήσεις επικοινωνίας σε IoT και κριτήρια επιλογής

Σε αντίθεση με κλασικές δικτυακές εφαρμογές, οι IoT συσκευές συχνά λειτουργούν σε περιβάλλοντα με ασταθή συνδεσιμότητα, περιορισμένους πόρους και ανάγκη συνεχούς λειτουργίας. Τα βασικά κριτήρια που λαμβάνονται υπόψη κατά τον σχεδιασμό της επικοινωνίας είναι:

- **Κάλυψη και εμβέλεια:** αν υπάρχει διαθέσιμη υποδομή τοπικού δικτύου ή απαιτείται δημόσια κάλυψη
- **Καθυστέρηση (latency):** επηρεάζει την "αίσθηση" real-time και τη συμπεριφορά σε αυτοματισμούς/εντολές
- **Ρυθμός μετάδοσης και όγκος δεδομένων:** τηλεμετρία χαμηλού ρυθμού έχει διαφορετικές απαιτήσεις από ροές υψηλής συχνότητας
- **Αξιοπιστία/διαθεσιμότητα:** αποσυνδέσεις, παρεμβολές, αλλαγές κυψέλης σε κινητή, συμφόρηση
- **Κατανάλωση και κόστος:** ιδιαίτερα σε λύσεις που βασίζονται σε δίκτυο κινητής (SIM/data plan)
- **Ασφάλεια και έλεγχος πρόσβασης:** ταυτοποίηση, έλεγχος δικαιωμάτων, κρυπτογράφηση (όπου απαιτείται).

Με βάση τα παραπάνω, σε πολλές πλατφόρμες υιοθετείται "διπλή" δυνατότητα επικοινωνίας: μία λύση για εγκαταστάσεις με διαθέσιμο LAN και μία για απομακρυσμένα σημεία χωρίς τοπική υποδομή.

2.4.2 Επικοινωνία σε τοπικό δίκτυο (LAN): WiFi/Ethernet

Η επικοινωνία μέσω LAN είναι κατάλληλη όταν η συσκευή εγκαθίσταται σε χώρο με διαθέσιμη δικτυακή υποδομή (οικιακή ή βιομηχανική). Σε αυτό το σενάριο, τεχνολογίες όπως WiFi (ή εναλλακτικά Ethernet) παρέχουν:

- ικανοποιητικούς ρυθμούς μετάδοσης
- χαμηλό latency σε σχέση με δίκτυα κινητής
- και συνήθως μηδενικό επιπλέον λειτουργικό κόστος (σε επίπεδο δεδομένων).

Ωστόσο, η αξιοπιστία εξαρτάται από την ποιότητα της εγκατάστασης (σήμα, παρεμβολές 2.4 GHz, φόρτος δικτύου), ενώ σε βιομηχανικούς χώρους μπορεί να απαιτηθούν πρακτικές όπως σωστή τοποθέτηση access points και επιτήρηση της σύνδεσης.

2.4.3 Επικοινωνία σε δημόσιο δίκτυο (WAN): LTE/4G

Η χρήση LTE/4G είναι κρίσιμη σε εγκαταστάσεις όπου δεν υπάρχει διαθέσιμο LAN (π.χ. απομακρυσμένες υποδομές, κινητές εφαρμογές, αγροτικά περιβάλλοντα). Τα δίκτυα κινητής προσφέρουν ευρεία κάλυψη, αλλά παρουσιάζουν ιδιαιτερότητες:

- μεγαλύτερη και πιο μεταβλητή καθυστέρηση
- πιθανές προσωρινές απώλειες λόγω κάλυψης ή αλλαγών κυψέλης
- και πρακτικά θέματα όπως NAT/προσβασιμότητα (συνήθως η συσκευή δεν είναι άμεσα προσπελάσιμη από το internet).

Επομένως, σε LTE σενάρια είναι σημαντική η ύπαρξη μηχανισμών επανασύνδεσης και ανθεκτικού application-layer πρωτοκόλλου που "αντέχει" σε αστάθεια.

2.4.4 Πρωτόκολλα εφαρμογής για IoT: MQTT έναντι HTTP

Σε επίπεδο εφαρμογής, δύο συνηθισμένες προσεγγίσεις είναι:

- **HTTP/REST:** απλό και καθολικά υποστηριζόμενο, αλλά συχνά βασίζεται σε request/response μοντέλο και οδηγεί είτε σε polling είτε σε πιο "βαριές" υλοποιήσεις για real-time.
- **MQTT:** σχεδιασμένο για IoT, με μικρό overhead και μοντέλο publish/subscribe, όπου ο broker αναλαμβάνει τη δρομολόγηση μηνυμάτων.

Στο MQTT, οι συσκευές **δημοσιεύουν (publish)** δεδομένα σε topics και **εγγράφονται (subscribe)** σε topics εντολών. Αυτό αποσυνδέει λειτουργικά τον αποστολέα από τον παραλήπτη (decoupling): η συσκευή δεν χρειάζεται να γνωρίζει ποιος "ακούει" τα δεδομένα της, ενώ το backend μπορεί να λαμβάνει δεδομένα από πολλές συσκευές με ομοιόμορφο τρόπο.

2.4.5 Αξιοπιστία επικοινωνίας και διαχείριση συνδέσεων

Η αξιοπιστία στην πράξη δεν είναι μόνο "να συνδεθεί", αλλά να συνεχίσει να λειτουργεί όταν το δίκτυο διακόπτεται. Σε MQTT-based αρχιτεκτονικές, βασικές έννοιες αξιοπιστίας είναι:

- **Keep-alive και επιτήρηση σύνδεσης:** περιοδικά μηνύματα/σήματα που επιβεβαιώνουν ότι η σύνδεση παραμένει ενεργή
- **Reconnection strategy:** επανασύνδεση με back-off (σταδιακή αύξηση χρόνου αναμονής), ώστε να αποφεύγεται υπερφόρτωση
- QoS (Quality of Service):
- QoS 0: "best effort" (χωρίς επιβεβαίωση)
- QoS 1: "at least once" (με πιθανές διπλοεγγραφές)
- QoS 2: "exactly once" (με μεγαλύτερο overhead).
- **Retained/Last-Will:** τεχνικές που βοηθούν ώστε μια συσκευή ή το backend να μπορεί να ανακτήσει την τελευταία γνωστή κατάσταση ή να δηλώνεται αστοχία/αποσύνδεση.

Σε συστήματα που έχουν και uplink (τηλεμετρία) και downlink (εντολές), συνήθως απαιτείται διαφορετική "φιλοσοφία": η τηλεμετρία συχνά μπορεί να είναι χαμηλότερου QoS (επειδή έρχεται ξανά), ενώ οι εντολές χρειάζονται αυξημένη πιθανότητα παραλαβής και σαφή χειρισμό retries/επιβεβαιώσεων.

2.4.6 Topics και μορφή μηνυμάτων: ονοματοδοσία, payload, χρονικές σημάνσεις

Η σωστή μοντελοποίηση των topics και του payload καθορίζει την επεκτασιμότητα:

- **Namespaces σε topics:** πρακτική για διαχωρισμό ανά χρήστη/συσκευή και αποφυγή ανάμειξης δεδομένων
- **Payload σε JSON:** συνηθισμένη επιλογή λόγω αναγνωσιμότητας και εύκολου parsing σε backend/web
- **Χρονικές σημάνσεις:** συχνά είναι χρήσιμη η διάκριση μεταξύ "χρόνου μέτρησης" (device time/ts) και "χρόνου παραλαβής" (server receive time), ιδιαίτερα σε δίκτυα με καθυστερήσεις όπως LTE
- **Versioning/συμβατότητα:** όταν εξελίσσεται το format, ένα πεδίο έκδοσης ή σταθερή δομή βοηθά να μη "σπάει" η πλατφόρμα.

Με την παραπάνω λογική, οι τεχνολογίες πρόσβασης (WiFi/LTE) και το MQTT πρωτόκολλο συνεργάζονται ώστε να επιτυγχάνεται αμφίδρομη επικοινωνία: μεταφορά τηλεμετρίας προς το backend και μεταφορά εντολών/ρυθμίσεων προς τη συσκευή. Η συγκεκριμένη υλοποίηση των topics, των μηνυμάτων και των μηχανισμών parsing/αποθήκευσης αναλύεται στα κεφάλαια υλοποίησης του backend και της πλατφόρμας.[4]

2.5 Backend και αποθήκευση δεδομένων

Σε ένα σύγχρονο IoT σύστημα, το backend αποτελεί το κεντρικό επίπεδο που γεφυρώνει τις συσκευές IoT με την εφαρμογή χρήστη. Ο βασικός του ρόλος δεν είναι μόνο να παραλαμβάνει τηλεμετρία, αλλά να την οργανώνει σε μια προκαθορισμένη μορφή, να την αποθηκεύει και να δίνει τις πληροφορίες στη διαδικτυακή πλατφόρμα όποτε ζητηθούν. Παράλληλα, το backend λειτουργεί και ως κανάλι ελέγχου, καθώς δρομολογεί ρυθμίσεις και εντολές προς τις συσκευές, επιτρέποντας παραμετροποίηση χωρίς φυσική πρόσβαση ή επαναπρογραμματισμό.

Λειτουργικά, μπορούν να διακριθούν δύο βασικές ροές: (α) η ροή τηλεμετρίας (uplink), όπου οι μετρήσεις φτάνουν στο backend και αποθηκεύονται με την κατάλληλη χρονική σήμανση, και (β) η ροή ελέγχου (downlink), όπου οι αλλαγές που προέρχονται από τον χρήστη μετατρέπονται σε δομημένες εντολές και αποστέλλονται στη συσκευή. Ο διαχωρισμός αυτός είναι χρήσιμος, επειδή οι δύο ροές έχουν διαφορετικές απαιτήσεις: η τηλεμετρία δίνει έμφαση στον όγκο/ρυθμό δεδομένων και στην αποδοτική αποθήκευση, ενώ ο έλεγχος δίνει έμφαση στην αξιοπιστία παραλαβής και στη σωστή εφαρμογή ρυθμίσεων.

Για την αποθήκευση, οι μετρήσεις αντιμετωπίζονται ως δεδομένα ως προς τον χρόνο (time-series), όπου κρίσιμα στοιχεία είναι τα timestamps, η δυνατότητα αναζήτησης ανά χρονικό διάστημα και η συσχέτιση των δεδομένων με χρήστες/συσκευές. Επιπλέον, η πλατφόρμα χρειάζεται μηχανισμό εξυπηρέτησης δεδομένων τόσο για ιστορική ανάλυση (π.χ. queries/γραφικές παραστάσεις) όσο και για προβολή σε σχεδόν πραγματικό χρόνο, όπου συχνά αξιοποιούνται τεχνικές push ενημέρωσης (π.χ. WebSocket) ώστε να αποφεύγεται η συνεχής ανανέωση της σελίδας.

Στο Κεφάλαιο 6 παρουσιάζεται η υλοποίηση του backend της πλατφόρμας που έχει δημιουργηθεί για τη διπλωματική εργασία, με έμφαση στη ροή MQTT μηνυμάτων, στη διαδικασία parsing /κανονικοποίησης, στη μοντελοποίηση της βάσης δεδομένων και στον μηχανισμό αποστολής κανόνων και ρυθμίσεων προς τη συσκευή.

2.6 Διαδικτυακές πλατφόρμες εποπτείας και ελέγχου

Οι διαδικτυακές πλατφόρμες αποτελούν το ανώτερο επίπεδο (application layer) ενός IoT συστήματος και λειτουργούν ως το κύριο σημείο αλληλεπίδρασης του χρήστη με τις συσκευές IoT. Ο ρόλος τους δεν περιορίζεται στην απλή προβολή μετρήσεων αλλά περιλαμβάνει την εποπτεία της κατάστασης, την αναζήτηση ιστορικών δεδομένων, καθώς και την απομακρυσμένη παραμετροποίηση και τον έλεγχο. Με αυτόν τον τρόπο μια πλατφόρμα μετατρέπει ακατέργαστα δεδομένα (telemetry) σε πληροφορία χρήσιμη για λήψη αποφάσεων και προσφέρει μηχανισμούς διαχείρισης της συμπεριφοράς των συσκευών χωρίς φυσική πρόσβαση στον χώρο που βρίσκεται η συσκευή.

Σε μία τυπική αρχιτεκτονική, η διαδικτυακή πλατφόρμα δεν επικοινωνεί απευθείας με τις συσκευές. Αντίθετα, συνεργάζεται με ένα ενδιάμεσο επίπεδο (backend), το οποίο αναλαμβάνει τη συλλογή δεδομένων, την αποθήκευση, και τη δρομολόγηση εντολών και ρυθμίσεων. Ο διαχωρισμός αυτός διευκολύνει την κλιμάκωση, την ασφάλεια και τη συντήρηση του συστήματος, καθώς το frontend εστιάζει στην οπτικοποίηση και στην εμπειρία χρήστη, ενώ το backend εστιάζει στη συνέπεια δεδομένων και στη διαχείριση επικοινωνίας.

2.6.1 Βασικές λειτουργίες πλατφόρμας

Μία σύγχρονη IoT πλατφόρμα καλύπτει συνήθως τρεις βασικές κατηγορίες λειτουργιών:

- **Εποπτεία:** άμεση εικόνα της τρέχουσας κατάστασης μέσω τελευταίων μετρήσεων και ενδείξεων λειτουργίας (π.χ. on/off καταστάσεις, κατάσταση συσκευής)
- **Ανάλυση:** πρόσβαση σε ιστορικά δεδομένα για εντοπισμό τάσεων, συμβάντων ή αποκλίσεων, με δυνατότητα επιλογής χρονικού διαστήματος
- **Έλεγχος και παραμετροποίηση:** ορισμός παραμέτρων, αποστολή εντολών και διαχείριση αυτοματισμών που επηρεάζουν τη λειτουργία των συσκευών.

2.6.2 Ροές δεδομένων και ροές ελέγχου

Από τη σκοπιά του συστήματος, είναι χρήσιμο να διαχωρίζεται η λειτουργία της πλατφόρμας σε δύο ροές:

- **Ροή δεδομένων (data-plane):** αφορά τις μετρήσεις που συλλέγονται από τις συσκευές και προβάλλονται σε πραγματικό ή σχεδόν πραγματικό χρόνο, ενώ παράλληλα αποθηκεύονται ώστε να είναι διαθέσιμες για ιστορική αναδρομή
- **Ροή ελέγχου (control-plane):** αφορά ενέργειες που ξεκινούν από τον χρήστη (π.χ. αλλαγή ρυθμίσεων, ενεργοποίηση λειτουργίας, ορισμός κανόνων) και μετατρέπονται σε εντολές που προωθούνται προς τις συσκευές μέσω του backend.

Ο παραπάνω διαχωρισμός είναι πρακτικός, επειδή οι απαιτήσεις διαφέρουν: στο data-plane προέχει η συχνότητα και η οργάνωση δεδομένων, ενώ στο control-plane προέχει η ορθότητα εφαρμογής αλλαγών και η σαφής επιβεβαίωση της κατάστασης μετά από μία ενέργεια.

2.6.3 Προβολή και ανάλυση μετρήσεων

Η παρουσίαση δεδομένων σε μορφή που διευκολύνει την κατανόηση είναι καθοριστική για τη χρησιμότητα μιας πλατφόρμας. Συνήθεις πρακτικές περιλαμβάνουν:

- **Προβολή τελευταίας τιμής (latest state):** γρήγορη απεικόνιση των πιο πρόσφατων μετρήσεων/καταστάσεων

- **Ιστορική απεικόνιση:** γραφήματα ή πίνακες με δυνατότητα επιλογής χρονικού παραθύρου (ώρες/ημέρες), ώστε να είναι εμφανείς τάσεις και μεταβολές
- **Φιλτράρισμα και ομαδοποίηση:** επιλογή υποσυνόλων μετρήσεων (ανά συσκευή, κατηγορία ή τύπο), ώστε να μειώνεται η πληροφοριακή "υπερφόρτωση
- **Dashboards:** συγκεντρωτικές οθόνες με widgets (ενδείξεις, διακόπτες, μικρά γραφήματα), οι οποίες επιτρέπουν γρήγορη εποπτεία χωρίς μετάβαση σε πολλές σελίδες.

2.6.4 Παραμετροποίηση, εντολές και αυτοματισμοί

Πέρα από την απεικόνιση, οι πλατφόρμες υποστηρίζουν συχνά απομακρυσμένη διαχείριση της λειτουργίας των συσκευών, όπως:

- αλλαγές παραμέτρων λειτουργίας
- χειροκίνητες εντολές ελέγχου (όπου απαιτείται άμεση παρέμβαση)
- κανόνες αυτοματισμού

Σε αυτή την κατηγορία λειτουργιών, είναι σημαντικό η διεπαφή να παρέχει καθαρή εικόνα για το τι αλλάζει, να μειώνει την πιθανότητα λανθασμένης ενέργειας και να υποστηρίζει μηχανισμούς επιβεβαίωσης/ανατροφοδότησης, ιδιαίτερα όταν η εντολή επηρεάζει κρίσιμες λειτουργίες.

2.6.5 Ενημέρωση σε πραγματικό χρόνο: polling έναντι push

Για την ανανέωση δεδομένων στον browser χρησιμοποιούνται δύο βασικές προσεγγίσεις:

- **Polling:** περιοδική ανάκτηση νέων τιμών από το backend, με απλή υλοποίηση αλλά αυξημένο αριθμό αιτήσεων και πιθανή καθυστέρηση ίση με το polling interval
- **Push ενημέρωση (π.χ. WebSocket):** το backend προωθεί νέα δεδομένα όταν αυτά καταφθάνουν, προσφέροντας πιο "ζωντανή" απεικόνιση και αποφυγή συνεχών ανανεώσεων.

Η επιλογή εξαρτάται από τη συχνότητα μετρήσεων, τον αριθμό χρηστών και το πόσο σημαντική είναι η αίσθηση πραγματικού χρόνου.

2.6.6 Ασφάλεια και πολυχρηστικότητα

Σε περιβάλλοντα με πολλούς χρήστες ή πολλαπλές συσκευές, η πλατφόρμα πρέπει να εξασφαλίζει:

- αυθεντικοποίηση (λογαριασμοί χρηστών, συνεδρίες)
- εξουσιοδότηση (δικαιώματα ανά χρήστη/ρόλο)
- απομόνωση δεδομένων (κάθε χρήστης να βλέπει μόνο τις δικές του συσκευές και δεδομένα).

Η ασφάλεια δεν αφορά μόνο την προστασία δεδομένων, αλλά και την αποφυγή μη εξουσιοδοτημένων ενεργειών ελέγχου.

3. Αρχιτεκτονική συστήματος

3.1 Συνολική αρχιτεκτονική συστήματος και ροή δεδομένων

Στην διπλωματική εργασία έχει ακολουθηθεί μία τυπική αρχιτεκτονική IoT πολλαπλών επιπέδων (layered architecture), όπου διαχωρίζονται σαφώς οι ρόλοι της συσκευής IoT, της επικοινωνίας, του backend και της διαδικτυακής εφαρμογής. Ο διαχωρισμός αυτός διευκολύνει την επεκτασιμότητα και τη συντήρηση: η συσκευή επικεντρώνεται στη συλλογή/έλεγχο, το επίπεδο επικοινωνίας στη μεταφορά μηνυμάτων, το backend στην οργάνωση και αποθήκευση δεδομένων, και η διαδικτυακή πλατφόρμα στην εποπτεία και αλληλεπίδραση με τον χρήστη.

Σε υψηλό επίπεδο, διακρίνονται δύο βασικές ροές λειτουργίας:

- Ροή τηλεμετρίας (uplink / data-plane): μεταφορά μετρήσεων από τη συσκευή προς το backend και τελικά προς την πλατφόρμα χρήστη
- Ροή ελέγχου (downlink / control-plane): μεταφορά ρυθμίσεων και κανόνων από τον χρήστη και το backend προς τη συσκευή.

Στις επόμενες ενότητες παρουσιάζονται τα επίπεδα του συστήματος και οι μεταξύ τους διεπαφές, χωρίς τις λεπτομέρειες υλοποίησης, οι οποίες αναλύονται στα αντίστοιχα κεφάλαια.

3.1.1 Device layer (Επίπεδο συσκευής)

Το επίπεδο συσκευής περιλαμβάνει το ενσωματωμένο σύστημα που αλληλεπιδρά με το φυσικό περιβάλλον. Οι κύριες λειτουργίες του είναι:

- **Συλλογή μετρήσεων και καταστάσεων εισόδων** (αναλογικές και ψηφιακές), καθώς και από τα υπόλοιπα υποσυστήματα/αισθητήρες
- **Τοπικός έλεγχος εξόδων** (π.χ. ενεργοποίηση/απενεργοποίηση σημάτων) με βάση την τρέχουσα κατάσταση
- **Βασική λογική λειτουργίας και χρονισμός**, ώστε να εκτελούνται περιοδικές εργασίες (δειγματοληψία, ενημέρωση κατάστασης, αποστολή τηλεμετρίας)
- **Υποστήριξη κανόνων/παραμέτρων λειτουργίας**, ώστε το σύστημα να μπορεί να προσαρμόζεται χωρίς επαναπρογραμματισμό (η λογική κανόνων και ο μηχανισμός εφαρμογής τους αναλύονται στο κεφάλαιο firmware).

Το device layer είναι υπεύθυνο να “πακετάρει” τα δεδομένα σε μορφή κατάλληλη για μεταφορά και να εκτελεί τις εντολές/ρυθμίσεις που λαμβάνει από την πλατφόρμα μέσω του backend.

3.1.2 Communication layer (Επίπεδο επικοινωνίας)

Το επίπεδο επικοινωνίας αφορά τον τρόπο με τον οποίο η συσκευή αποκτά πρόσβαση στο δίκτυο και ανταλλάσσει μηνύματα με το backend. Υποστηρίζονται δύο εναλλακτικά σενάρια πρόσβασης:

- **Τοπική πρόσβαση (LAN):** κατάλληλη όταν υπάρχει διαθέσιμη δικτυακή υποδομή
- **Πρόσβαση μέσω δημόσιου δικτύου (WAN):** κατάλληλη για απομακρυσμένες εγκαταστάσεις χωρίς τοπικό δίκτυο.

Πάνω από το επίπεδο πρόσβασης(LAN ή WAN), χρησιμοποιείται μοντέλο ανταλλαγής μηνυμάτων όπου:

- η συσκευή δημοσιεύει (publish) τηλεμετρία
- λαμβάνει (subscribe) εντολές/ρυθμίσεις.

Οι λεπτομέρειες των πρωτοκόλλων, των μηχανισμών αξιοπιστίας και των παραμέτρων επικοινωνίας καλύπτονται στο Κεφάλαιο 2 (θεωρητικό υπόβαθρο) και στην υλοποίηση στο Κεφάλαιο firmware.

3.1.3 Backend layer (Επίπεδο backend)

Το backend αποτελεί το ενδιάμεσο επίπεδο που γεφυρώνει τις συσκευές με τη διαδικτυακή εφαρμογή, αναλαμβάνοντας την οργανωμένη διαχείριση δεδομένων και εντολών. Σε αρχιτεκτονικό επίπεδο, οι βασικές ευθύνες του backend είναι:

- **Λήψη μηνυμάτων τηλεμετρίας** από τις συσκευές και αντιστοίχισή τους στο σωστό χρήστη/συσκευή
- **Επεξεργασία/κανονικοποίηση** των δεδομένων ώστε να αποθηκεύονται με συγκεκριμένη μορφή
- **Αποθήκευση ιστορικών μετρήσεων** στη βάση δεδομένων και δυνατότητα ανάκτησης ανά χρονικό διάστημα
- **Δρομολόγηση εντολών και ρυθμίσεων** προς τη συσκευή (downlink), όταν ο χρήστης πραγματοποιεί αλλαγές από τη διαδικτυακή πλατφόρμα
- **Παροχή διεπαφής προς το web επίπεδο**, ώστε η εφαρμογή να αντλεί ιστορικά δεδομένα και να λαμβάνει ενημερώσεις "τρέχουσας κατάστασης".

Η αναλυτική υλοποίηση του backend (ροή μηνυμάτων, parsing, μοντελοποίηση βάσης και μηχανισμός αποστολής κανόνων/ρυθμίσεων) παρουσιάζεται στο Κεφάλαιο 6.

3.1.4 Application layer (Επίπεδο διαδικτυακής εφαρμογής)

Το επίπεδο εφαρμογής είναι το περιβάλλον αλληλεπίδρασης με τον χρήστη (web πλατφόρμα). Σε υψηλό επίπεδο, οι βασικές λειτουργίες του είναι:

- **Εποπτεία σε πραγματικό ή σχεδόν πραγματικό χρόνο**, με προβολή των τελευταίων μετρήσεων/καταστάσεων
- **Ιστορική ανάλυση**, με αναζήτηση μετρήσεων σε χρονικά διαστήματα και παρουσίαση σε γραφήματα/αναφορές
- **Παραμετροποίηση συστήματος**, μέσω ρυθμίσεων που επηρεάζουν τη λειτουργία της συσκευής
- **Ορισμός αυτοματισμών (κανόνων)**, ώστε να εκτελούνται ενέργειες όταν ικανοποιούνται συνθήκες.

Οι λεπτομέρειες υλοποίησης της διαδικτυακής εφαρμογής (σελίδες, ροές χρήστη, dashboard, endpoints και real-time ενημέρωση) αναλύονται στο Κεφάλαιο 7.

3.1.5 Ροή τηλεμετρίας (Uplink / Data-plane)

Στη ροή τηλεμετρίας, η συσκευή συλλέγει μετρήσεις/καταστάσεις και τις αποστέλλει περιοδικά προς το backend. Το backend επεξεργάζεται τα δεδομένα, τα αποθηκεύει ως ιστορικό (με χρονική σήμανση) και τα καθιστά διαθέσιμα στη διαδικτυακή πλατφόρμα, τόσο για άμεση εποπτεία όσο και για αναδρομική ανάλυση.

Η βασική αρχή είναι ότι η τηλεμετρία αντιμετωπίζεται ως χρονική ακολουθία δεδομένων, ώστε να μπορούν να γίνουν συγκρίσεις, γραφήματα και εξαγωγή συμπερασμάτων σε βάθος χρόνου.

3.1.6 Ροή ελέγχου (Downlink / Control-plane)

Στη ροή ελέγχου, ο χρήστης πραγματοποιεί αλλαγές από την πλατφόρμα (π.χ. ρυθμίσεις λειτουργίας ή κανόνες αυτοματισμού). Οι αλλαγές αυτές περνούν από το backend, το οποίο τις οργανώνει σε δομημένες εντολές και τις προωθεί προς τη συσκευή. Η συσκευή λαμβάνει τις εντολές, ενημερώνει τις αντίστοιχες παραμέτρους και εφαρμόζει τη νέα λογική κατά τη λειτουργία της.

Η ροή αυτή είναι κρίσιμη διότι μετατρέπει την πλατφόρμα από "παθητική παρακολούθηση" σε εργαλείο απομακρυσμένης διαχείρισης και αυτοματισμού.

3.1.7 Οργάνωση επικοινωνίας και απομόνωση ανά χρήστη/συσκευή

Σε IoT συστήματα με περισσότερες από μία συσκευές ή περισσότερους από έναν χρήστες, απαιτείται σαφής απομόνωση δεδομένων. Μία συνήθης πρακτική είναι η οργάνωση των ροών επικοινωνίας με διακριτούς "χώρους ονοματοδοσίας" (namespaces), ώστε:

- η τηλεμετρία να αντιστοιχίζεται σε συγκεκριμένο λογαριασμό/συσκευή
- οι εντολές να δρομολογούνται μόνο προς τον σωστό παραλήπτη.

Με αυτόν τον τρόπο αποφεύγεται η ανάμειξη δεδομένων και διευκολύνεται η επεκτασιμότητα του συστήματος.

3.1.8 Σύνοψη

Η συνολική αρχιτεκτονική βασίζεται σε διακριτά επίπεδα με σαφείς ρόλους και δύο κύριες ροές λειτουργίας (uplink τηλεμετρίας και downlink ελέγχου). Το μοντέλο αυτό επιτρέπει την ομαλή συνεργασία συσκευής, επικοινωνίας, backend και διαδικτυακής εφαρμογής, ενώ οι λεπτομέρειες υλοποίησης αναλύονται στα αντίστοιχα κεφάλαια: firmware (Κεφ. 5), backend (Κεφ. 6) και web πλατφόρμα (Κεφ. 7).

3.2 Αρχιτεκτονική Υλικού (Hardware Architecture)

Η πλακέτα IoT σχεδιάστηκε με στόχο τη ευρεία χρήση, την αξιοπιστία και τη δυνατότητα χρήσης σε βιομηχανικά και απαιτητικά περιβάλλοντα. Ο σχεδιασμός της πλακέτας έγινε με το Eagle της Autodesk. Η πλακέτα υλοποιεί διακριτά υποσυστήματα, καθένα από τα οποία εξυπηρετεί συγκεκριμένη λειτουργία, επιτρέποντας τη σαφή οργάνωση του κυκλώματος και τη μελλοντική επεκτασιμότητα.

Ο διαχωρισμός του υλικού σε λειτουργικά blocks επιτρέπει:

- την ευκολότερη κατανόηση και ανάλυση του κυκλώματος
- την απομόνωση κρίσιμων λειτουργιών
- και την απλοποίηση της συντήρησης και της επέκτασης του συστήματος.

Στα επόμενα υποκεφάλαια παρουσιάζεται αναλυτικά το κάθε υποσυστήματα της πλακέτας, ξεκινώντας από το υποσύστημα τροφοδοσίας, το οποίο αποτελεί θεμέλιο για τη σωστή και αξιόπιστη λειτουργία όλων των υπόλοιπων κυκλωμάτων.

3.2.1 Τροφοδοσία

Το κύκλωμα τροφοδοσίας της IoT συσκευής έχει σχεδιαστεί ώστε να υποστηρίζει αξιόπιστη λειτουργία σε περιβάλλοντα βιομηχανικού τύπου, όπου είναι συνηθισμένη η χρήση τροφοδοσίας 24VDC. Η επιλογή αυτή καθιστά τη συσκευή συμβατή με πληθώρα εγκαταστάσεων αυτοματισμού και ελέγχου.

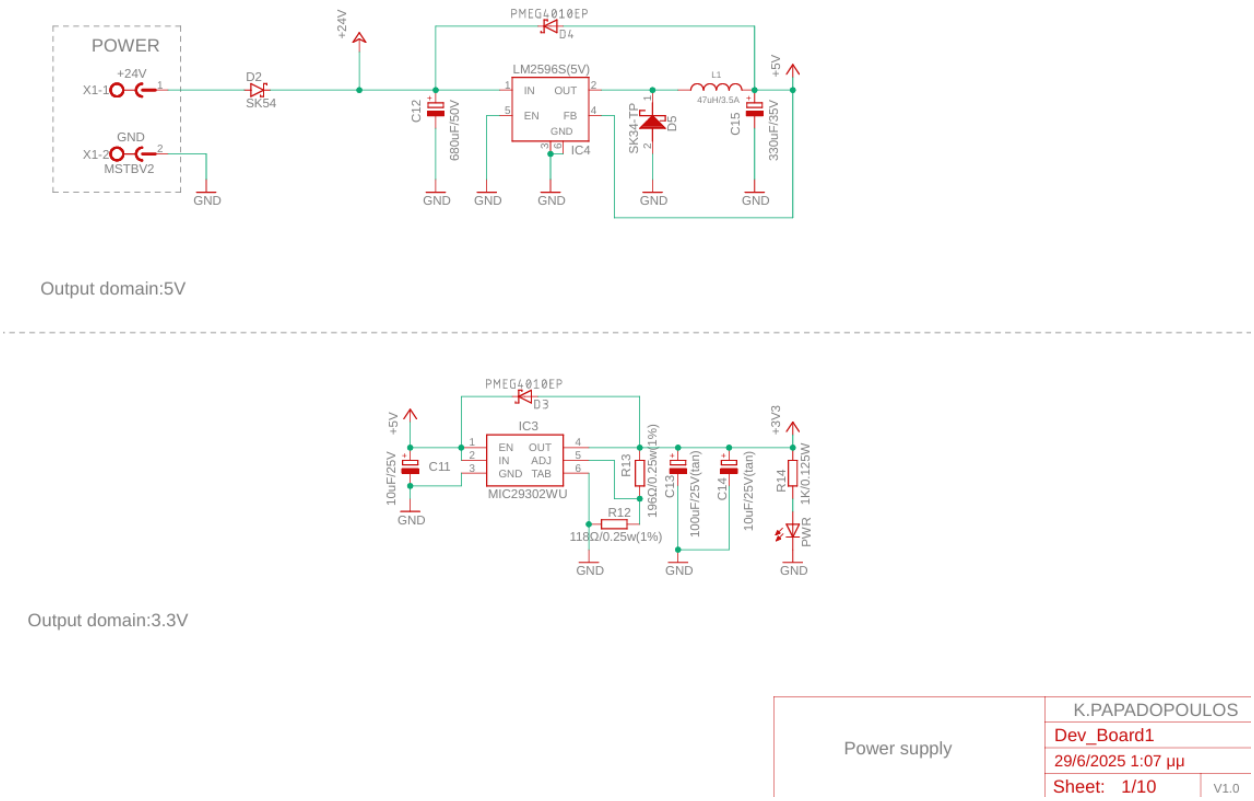
Η τροφοδοσία των +24V συνδέεται στην κλέμα εισόδου X1. Στην συνέχεια έχει τοποθετηθεί μία δίοδος Schottky(SK54), η οποία λειτουργεί ως προστασία από ανάστροφη πολικότητα. Χρησιμοποιήθηκε Schottky καθώς έχει μικρότερη πτώση τάσης σε σχέση με τις κοινές διόδους, άρα και μικρότερες απώλειες ισχύος.

Στην συνέχεια έχει τοποθετηθεί ένας ηλεκτρολυτικός πυκνωτής 680μF/50V ο οποίος:

- λειτουργεί ως bulk capacitor, δηλαδή μειώνει τις στιγμιαίες βυθίσεις τάσης που μπορεί να προκληθούν από τον regulator
- βοηθά στην μείωση του θορύβου/κυματισμού στην γραμμή +24V, ειδικά όταν η τροφοδοσία έρχεται μέσω καλωδίων όπου η επαγωγή αυτών μπορεί να δημιουργήσει αιχμές(spikes) όταν αλλάζει γρήγορα το ρεύμα

Έπειτα έχει τοποθετηθεί ο μετατροπέας των 24V σε 5V. Έχει επιλεχθεί ο LM2596 της Texas [1], οποίος είναι step-down(buck) switching regulator καθώς:

- παρέχει ρεύμα έως και 3A
- αντέχει στην είσοδο του έως 40V
- απαιτεί μικρό αριθμό εξαρτημάτων



Εικόνα 1 Τροφοδοτικό

Η γραμμή των 5V χρησιμοποιείται, μεταξύ άλλων, για την τροφοδοσία του LTE modem. Η επιλογή αυτή βασίζεται στις απαιτήσεις κατανάλωσης του modem, το οποίο κατά τη διάρκεια μετάδοσης μπορεί να εμφανίσει μέγιστη τιμή στιγμιαίου ρεύματος που φτάνει έως και τα 650 mA, σύμφωνα με τα στοιχεία του κατασκευαστή. Η ύπαρξη επαρκούς περιθωρίου ρεύματος στο υποσύστημα τροφοδοσίας εξασφαλίζει τη σταθερή λειτουργία της επικοινωνίας και αποτρέπει ανεπιθύμητες επανεκκινήσεις ή άλλου είδους σφάλματα.

Για την τροφοδοσία των λογικών κυκλωμάτων και του μικροελεγκτή χρησιμοποιείται σταθεροποιητής χαμηλής πτώσης τάσης (LDO) MIC29302WU [2], ο οποίος παράγει τάση 3.3V από τη γραμμή των 5V. Η σύνδεση του LDO στα 5V, παρότι θα μπορούσε να συνδεθεί απ' ευθείας στα 24V έχει τα εξής πλεονεκτήματα:

- καλύτερη συνολική απόδοση καθώς δεν χρειάζεται τόση ισχύς
- λειτουργεί ως post-regulator, μειώνοντας μέρος του θορύβου, που προκύπτει από τον switching regulator, πριν τροφοδοτηθούν τα επόμενα κυκλώματα όπως ο επεξεργαστής, ψηφιακά/αναλογικά κυκλώματα, CANbus transceiver, κτλπ

Είναι σημαντικό να προστεθεί πως αν και το datasheet της microchip αναφέρει πως στην έξοδο του LDO χρειάζεται μόνο ένας πυκνωτής τανταλίου 10μF, έχει προστεθεί ένας εξτρά πυκνωτής 100μF για να ενισχυθεί παραπάνω η σταθερότητα του συστήματος.

Τέλος έχει προστεθεί ένα Led, δίνοντας την οπτική επιβεβαίωση ότι η τροφοδοσία των 3.3V(άρα και οι προηγούμενες βαλβίδες 5V και 24V) είναι ενεργή.

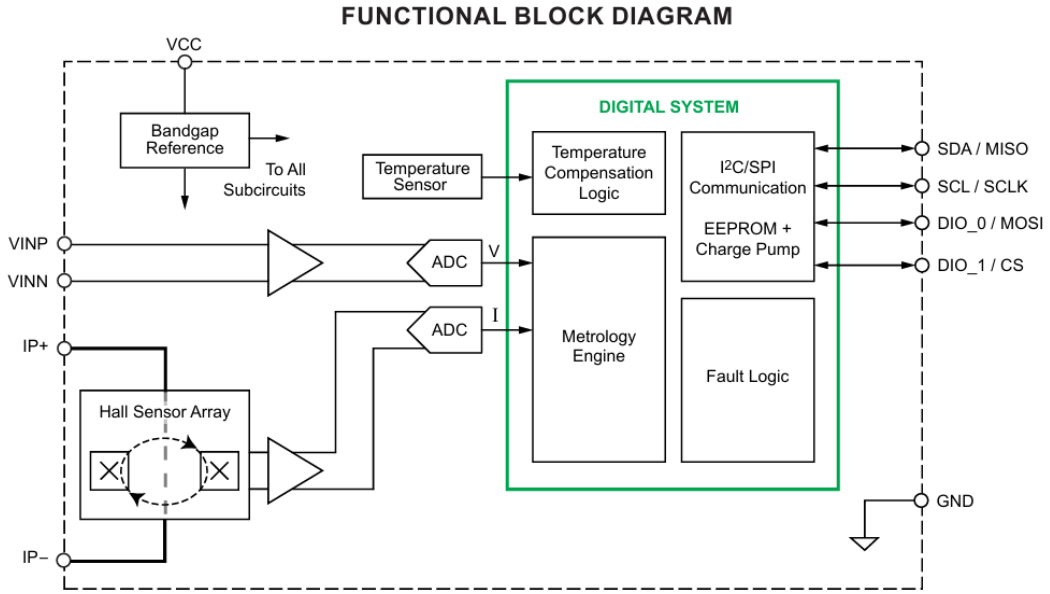
3.2.2 Μετρήσεις εναλλασσόμενου δικτύου (AC Measurements)

Στο υποσύστημα μετρήσεων δικτύου χρησιμοποιείται το ολοκληρωμένο ACS37800KMACTR-030B3-I2C(IC5), το οποίο ενσωματώνει αισθητήριο ρεύματος τύπου Hall και “metrology engine” για υπολογισμό ηλεκτρικών μεγεθών (π.χ. RMS τάσης/ρεύματος και ισχύος). Έχει επιλεχθεί το μοντέλο που αντιστοιχεί σε τροφοδοσία 3.3V, έχει εύρος μέτρησης ρεύματος ±30 A και επικοινωνεί με τον μικροελεγκτή μέσω I²C.

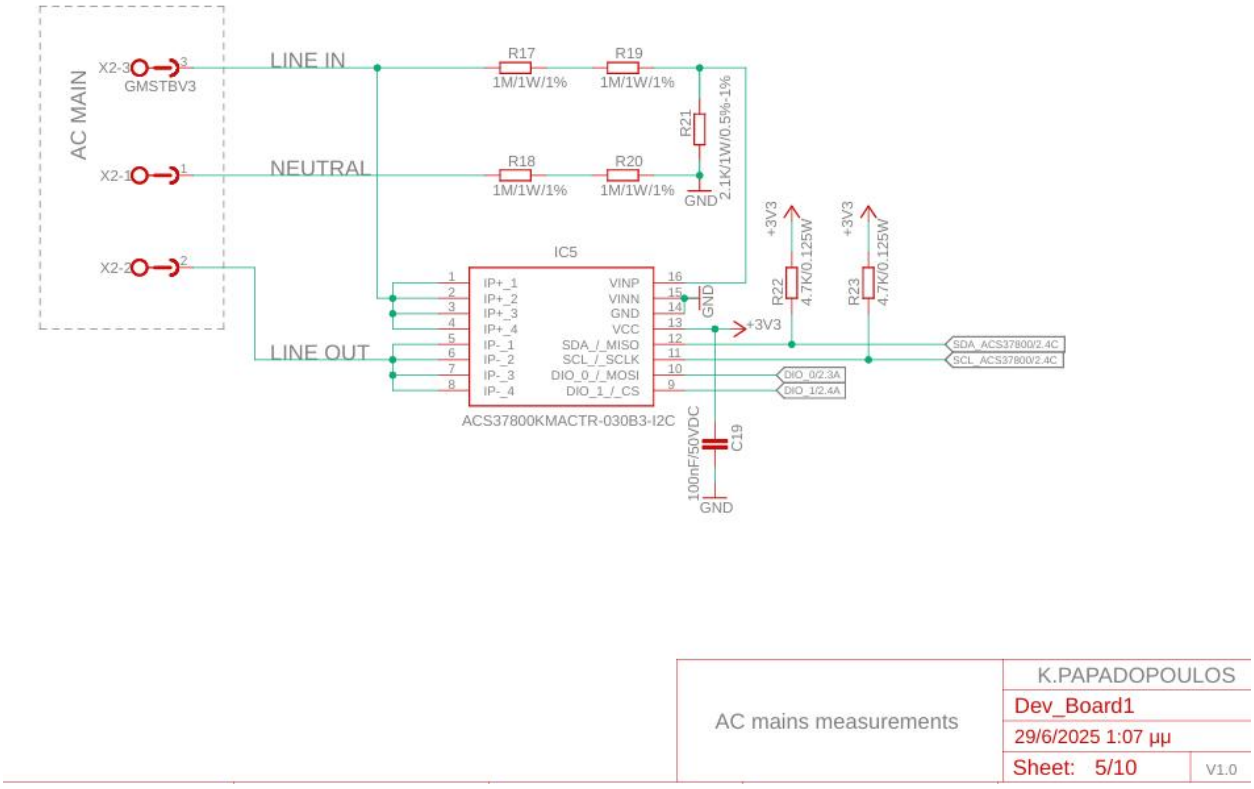
Το ACS37800 παρέχει τη δυνατότητα μέτρησης και υπολογισμού πολλαπλών ηλεκτρικών μεγεθών. Συγκεκριμένα, στη συσκευή μετρούνται:

- η ενεργός τιμή τάσης (Voltage RMS)
- η ενεργός τιμή ρεύματος (Current RMS)
- η ενεργός ισχύς (Active Power)
- η στιγμιαία τιμή τάσης
- και η στιγμιαία τιμή ρεύματος.

Η επεξεργασία των μεγεθών αυτών πραγματοποιείται εσωτερικά στο ολοκληρωμένο κύκλωμα, μειώνοντας σημαντικά το υπολογιστικό φορτίο του μικροελεγκτή και βελτιώνοντας την ακρίβεια των μετρήσεων. Επιπλέον, το ACS37800 υποστηρίζει προγραμματιζόμενο averaging σε επιλεγμένες εξόδους (τάση, ρεύμα ή ισχύ), επιτρέποντας την προσαρμογή της απόκρισης του συστήματος ανάλογα με τις απαιτήσεις της εφαρμογής.



Εικόνα 2 Μπλοκ διάγραμμα ACS37800



Εικόνα 3 Κύκλωμα ACS37800

Πιο συγκεκριμένα, η μέτρηση ρεύματος υλοποιείται “σε σειρά” με τη φάση: το LINE IN οδηγείται στα άκρα IP+ (pins 1–4) και το LINE OUT λαμβάνεται από τα IP− (pins 5–8), αξιοποιώντας τη σχεδιασμένη διαδρομή ισχύος του ολοκληρωμένου (οι ακροδέκτες IP+ και IP− είναι αντίστοιχα εσωτερικά κοινά ανά τετράδα, ώστε να διαχειρίζονται υψηλά ρεύματα). Παράλληλα, η μέτρηση τάσης γίνεται μέσω διαιρέτη αντιστάσεων: οι R17 και R19 (1 MΩ + 1 MΩ) λειτουργούν ως “RISO” (αντιστάσεις απομόνωσης/περιορισμού) και η R21 (2.1 kΩ) ως RSENSE, ώστε η τάση του δικτύου να μετασχηματίζεται σε σήμα εντός του επιτρεπτού εύρους της εισόδου τάσης του ACS37800. Το datasheet ορίζει ότι η διαφορική είσοδος τάσης έχει εύρος ±250 mV με VINN συνδεδεμένο στο GND, ενώ προτείνει αντίστοιχη τοπολογία με ίσες αντιστάσεις 1 MΩ για το τμήμα “RISO” και επιλογή του RSENSE με βάση το μέγιστο αναμενόμενο VLINE.

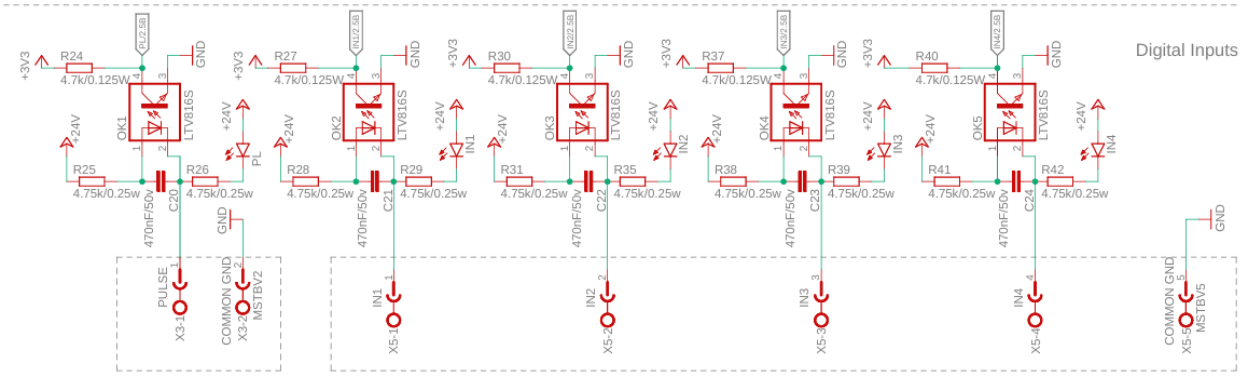
Μεταξύ της VCC και GND του ολοκληρωμένου έχει τοποθετηθεί bypass πυκνωτής 100nF, όπως προτείνεται στο datasheet. Η επικοινωνία μεταξύ του ACS37800 και του μικροελεγκτή πραγματοποιείται μέσω διαύλου I²C και στις γραμμές SDA/SCL έχουν τοποθετηθεί αντιστάσεις pull-up 4.7kΩ(R22, R23), που βρίσκονται εντός του προτεινόμενου εύρους του κατασκευαστή(2.4kΩ έως 10kΩ). Επίσης η χρήση διαύλου I²C μειώνει την ευαισθησία σε ηλεκτρικό θόρυβο και απλοποιεί τη σχεδίαση του υποσυστήματος.

Πέρα από τις μετρήσεις, το ACS37800 διαθέτει και ενσωματωμένες λειτουργίες προστασίας, οι οποίες μπορούν να αξιοποιηθούν για την ασφαλή λειτουργία του συστήματος. Μεταξύ αυτών περιλαμβάνονται:

- προγραμματιζόμενο όριο overvoltage και undervoltage RMS
- προγραμματιζόμενο όριο overcurrent fault
- έξοδος zero-crossing για τάση ή ρεύμα.

3.2.3 Ψηφιακές είσοδοι (Digital Inputs)

Το υποσύστημα ψηφιακών εισόδων σχεδιάστηκε για την αξιόπιστη ανίχνευση καταστάσεων από εξωτερικές επαφές, όπως διακόπτες, ρελέ ή αισθητήρες τύπου "ξηρής επαφής" (dry contact). Οι είσοδοι συνδέονται στην κλέμα X5, όπου υπάρχει κοινή γείωση και ξεχωριστό κανάλι για κάθε είσοδο (IN1–IN4), καθώς και μία επιπλέον είσοδος ανίχνευσης παλμών.



Εικόνα 4 Κύκλωμα ψηφιακών εισόδων

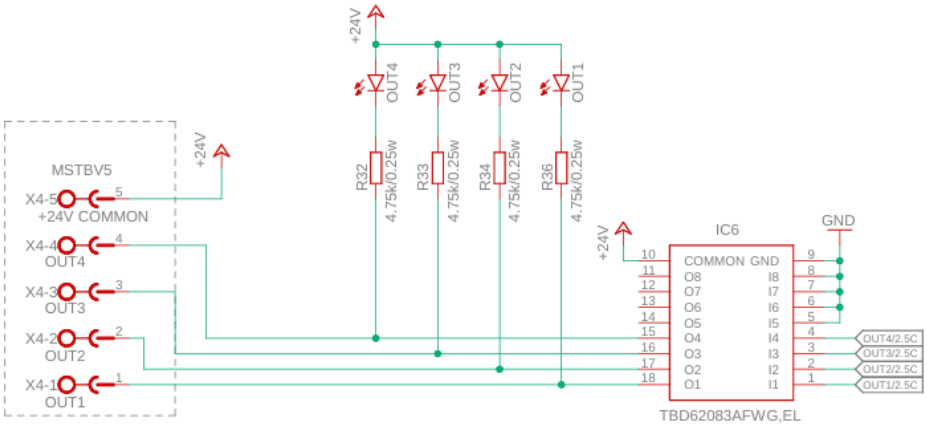
Για την προστασία του μικροελεγκτή και την καλύτερη αντοχή σε θόρυβο πεδίου, κάθε κανάλι υλοποιείται με οπτοζεύκτη LTV816S. Η ενεργοποίηση γίνεται από την πλευρά των 24V, μέσω αντιστάσεων περιορισμού ρεύματος (4.75 kΩ) και ενός απλού φίλτρου RC με πυκνωτή 470 nF, το οποίο μειώνει ψευδοενεργοποιήσεις από ηλεκτρικές παρεμβολές ή αναπηδήσεις επαφών (contact bounce).

Στην πλευρά προς τον μικροελεγκτή, το σήμα κάθε εισόδου οδηγείται σε pin του MCU και υπάρχει αντίσταση pull-up 4.7 kΩ προς +3.3V. Έτσι, όταν η είσοδος είναι ανενεργή, ο MCU "βλέπει" λογικό '1', ενώ όταν ενεργοποιηθεί ο οπτοζεύκτης, η γραμμή τραβιέται στη γείωση και διαβάζεται λογικό '0' (active-low).

Το υποσύστημα ψηφιακών εισόδων περιλαμβάνει επίσης ξεχωριστή είσοδο παλμών (pulse input), η οποία μπορεί να χρησιμοποιηθεί για εφαρμογές μέτρησης παλμών ή συχνότητας, όπως για παράδειγμα επαγωγικούς διακόπτες ή μετρητές παλμών. Η είσοδος αυτή αξιοποιείται σε συνδυασμό με τις δυνατότητες input capture των timers του μικροελεγκτή, επιτρέποντας τον υπολογισμό συχνότητας και duty cycle, όπως αναλύεται στα επόμενα κεφάλαια.

3.2.4 Ψηφιακές έξοδοι (Digital Outputs)

Οι ψηφιακές έξοδοι υλοποιούνται με τη χρήση του ολοκληρωμένου κυκλώματος TBD62083AFWG(IC6) της Toshiba, το οποίο χρησιμοποιείται για την οδήγηση φορτίων 24 V από σήματα λογικής του μικροελεγκτή.



Εικόνα 5 Κύκλωμα ψηφιακών εξόδων

Η σύνδεση των εξωτερικών φορτίων πραγματοποιείται μέσω της κλέμας X4, όπου παρέχεται κοινή γραμμή +24VDC και ξεχωριστές εξόδους για κάθε κανάλι. Με τον τρόπο αυτό, ο χρήστης μπορεί να συνδέσει διαφορετικά φορτία, χρησιμοποιώντας την ίδια τροφοδοσία, χωρίς την ανάγκη επιπλέον εξωτερικών κυκλωμάτων.

Στο κύκλωμα κάθε εξόδου έχουν προβλεφθεί ενδεικτικές λυχνίες LED, οι οποίες παρέχουν οπτική ένδειξη της κατάστασης της εξόδου. Η ύπαρξη ενδείξεων αυτών διευκολύνει τον έλεγχο και τη διάγνωση της λειτουργίας της συσκευής κατά την εγκατάσταση και τη συντήρηση.

Το ολοκληρωμένο TBD62083 επιλέχθηκε λόγω της ικανότητάς του να οδηγεί πολλαπλά κανάλια με επαρκή ρεύματα για τυπικά βιομηχανικά φορτία χαμηλής ισχύος, καθώς και λόγω της ενσωματωμένης προστασίας έναντι υπερτάσεων που προκαλούνται από επαγωγικά φορτία. Η επιλογή αυτή απλοποιεί τη σχεδίαση, μειώνει τον αριθμό εξωτερικών εξαρτημάτων και αυξάνει τη συνολική αξιοπιστία του υποσυστήματος.

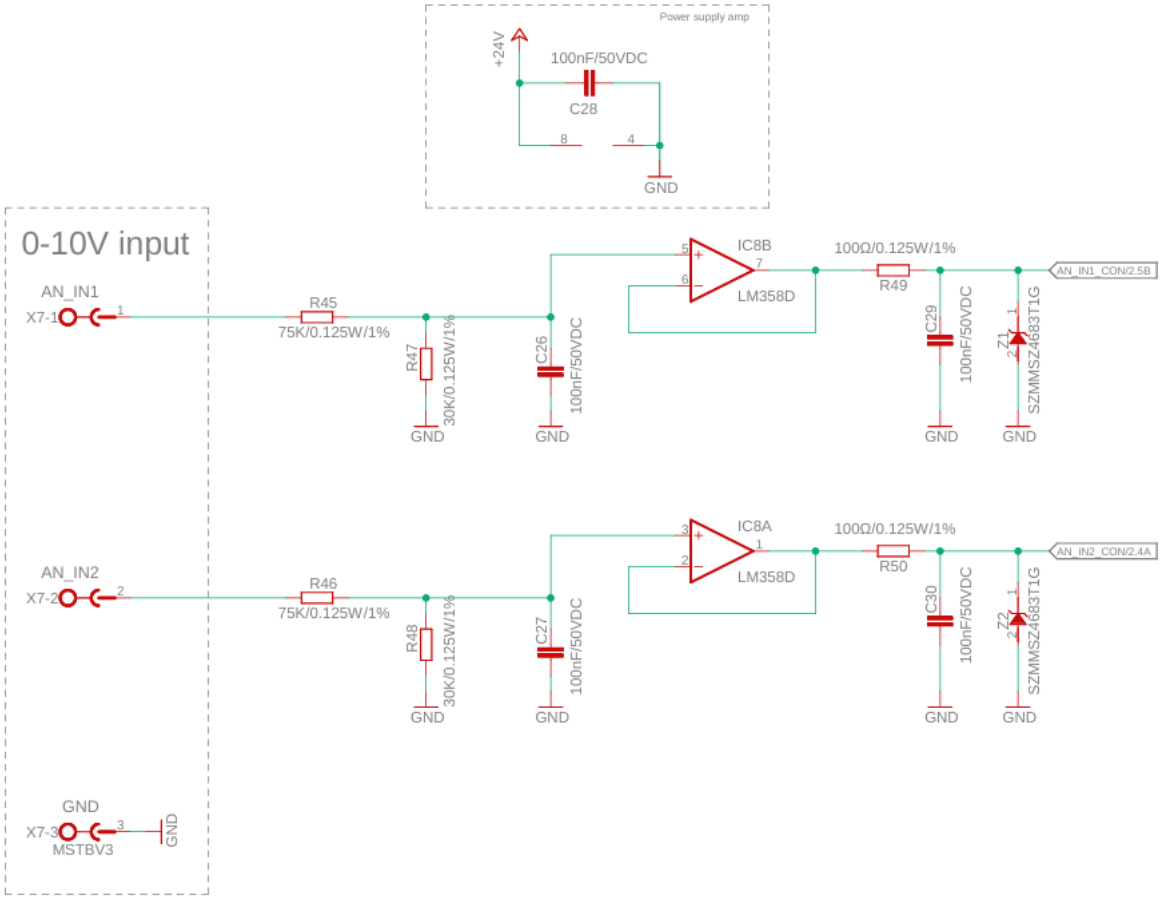
Συνολικά, το υποσύστημα ψηφιακών εξόδων παρέχει:

- αξιόπιστο έλεγχο φορτίων 24VDC
- προστασία έναντι επαγωγικών φαινομένων
- 500mA ανα κανάλι εξόδου

3.2.5 Αναλογικές είσοδοι (0–10V)

Το υποσύστημα αναλογικών εισόδων της προτεινόμενης IoT συσκευής έχει σχεδιαστεί για τη μέτρηση αναλογικών σημάτων εύρους 0–10V, τα οποία είναι ιδιαίτερα διαδεδομένα σε βιομηχανικούς αισθητήρες

και συστήματα αυτοματισμού. Τέτοιου είδους σήματα χρησιμοποιούνται συχνά για τη μέτρηση φυσικών μεγεθών όπως στάθμη, πίεση, θερμοκρασία ή θέση.



Εικόνα 6 Κύκλωμα αναλογικών εισόδων

Πιο συγκεκριμένα κάθε είσοδος (AN_IN1, AN_IN2) περνά πρώτα από έναν διαιρέτη τάσης με R=75 kΩ σε σειρά και R=30 kΩ προς γείωση (R45–R47 για AN_IN1 και R46–R48 για AN_IN2). Ο διαιρέτης μειώνει την τάση κατά λόγο:

$$V = V_{in} \cdot \frac{30K}{75K + 30K} \approx 0.286 \cdot V_{in}$$

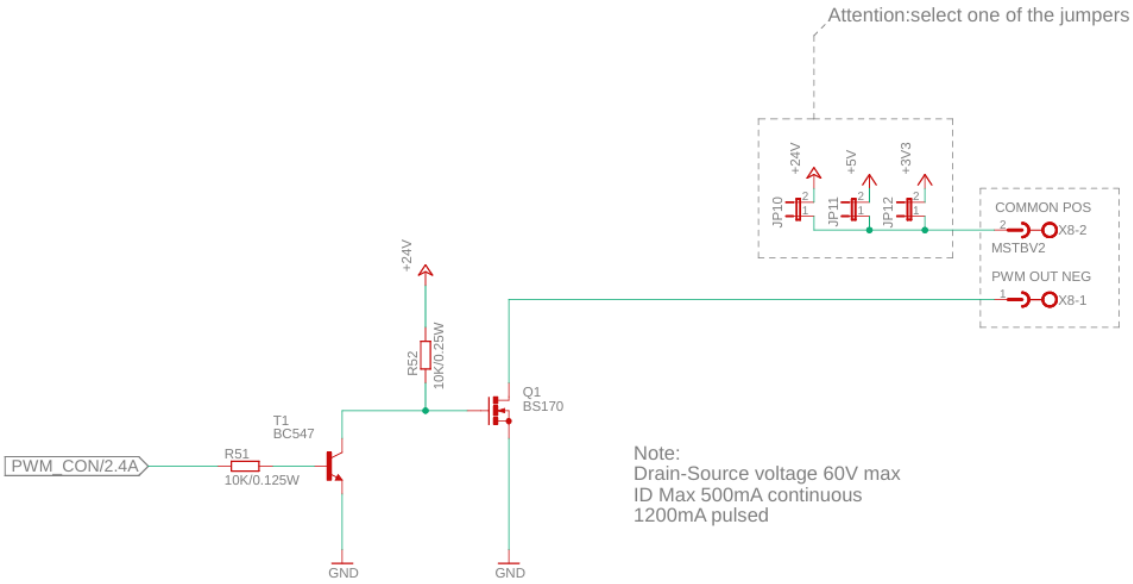
Άρα για είσοδο 10 V, προκύπτουν περίπου 2.86 V, τιμή που είναι κατάλληλη για είσοδο ADC 3.3 V χωρίς να φτάνει η τιμή στο μέγιστο της μέτρησης. Ακόμα έχει τοποθετηθεί πυκνωτής 100nF ως προς GND (C26 για AN_IN1 και C27 για AN_IN2), που λειτουργεί ως ήπιο χαμηλοπερατό φίλτρο, ώστε να μειώνονται γρήγορα spikes/παρεμβολές που μπορεί να έρχονται από καλώδια ή “θορυβώδη” αισθητήρια.

Το σήμα οδηγείται έπειτα στον τελεστικό LM358D που χρησιμοποιείται ως ακόλουθος τάσης (voltage follower), δηλαδή ως μη αναστρέφων buffer με κέρδος 1, αφού η έξοδος επιστρέφει στο αναστρέφον άκρο (−) και το σήμα εφαρμόζεται στο μη αναστρέφον (+). Με αυτή τη διάταξη, η τάση που προκύπτει από τον διαιρέτη 0–10 V αντιγράφεται πρακτικά στην έξοδο χωρίς πρόσθετη ενίσχυση, ενώ

ταυτόχρονα επιτυγχάνεται απομόνωση της εισόδου από το ADC και σταθερότερη οδήγηση του επόμενου σταδίου.

3.2.6 Έξοδος PWM

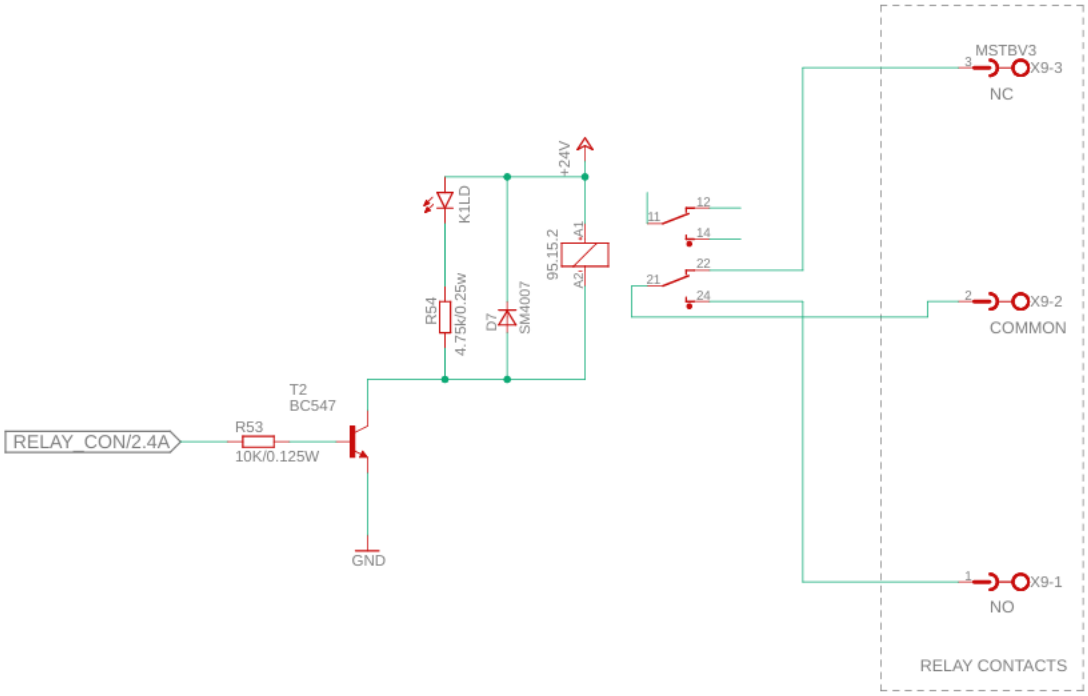
Η έξοδος PWM παράγεται από τον μικροελεγκτή μέσω timer και οδηγείται αρχικά σε ένα ενδιάμεσο στάδιο με NPN τρανζίστορ BC547, το οποίο χρησιμοποιείται ως pre-driver ώστε να απομονώνει την έξοδο του μικροελεγκτή και να εξασφαλίζει σωστή. Το τελικό στάδιο υλοποιείται με MOSFET N-channel BS170 σε διάταξη low-side switching, έτσι ώστε η αρνητική πλευρά του φορτίου (PWM OUT NEG) να συνδέεται παλμικά προς γείωση, επιτρέποντας ρύθμιση της ισχύος μέσω μεταβολής του duty cycle. Στο gate του MOSFET υπάρχει αντίσταση pull-up προς +24 V (R52), που διατηρεί καθορισμένη κατάσταση οδήγησης όταν το ενδιάμεσο στάδιο δεν άγει. Ένα βασικό πλεονέκτημα του υποσυστήματος είναι ότι ο θετικός πόλος του φορτίου (COM POS) δεν είναι σταθερός αλλά επιλέγεται μέσω jumpers, επιτρέποντας τροφοδοσία του φορτίου από +24 V, +5 V ή +3.3 V, ανάλογα με την εφαρμογή. Τέλος, η έξοδος διατίθεται σε δύο ακροδέκτες (COM POS και PWM OUT NEG), ώστε η σύνδεση εξωτερικών φορτίων να είναι απλή και κατανοητή χωρίς πρόσθετα βοηθητικά κυκλώματα.



Εικόνα 7 Κύκλωμα PWM

3.2.7 Ρελέ (Relay Output)

Το κύκλωμα έχει σχεδιαστεί για τον έλεγχο εξωτερικών φορτίων με ηλεκτρική απομόνωση μεταξύ του κυκλώματος ελέγχου και του κυκλώματος ισχύος. Η χρήση ρελέ δίνει ευελιξία, καθώς επιτρέπει τη διακοπή ή σύνδεση φορτίων διαφορετικού τύπου (AC ή DC), πάντοτε εντός των ορίων των χαρακτηριστικών του ίδιου του ρελέ.



Εικόνα 8 Κύκλωμα ρελέ

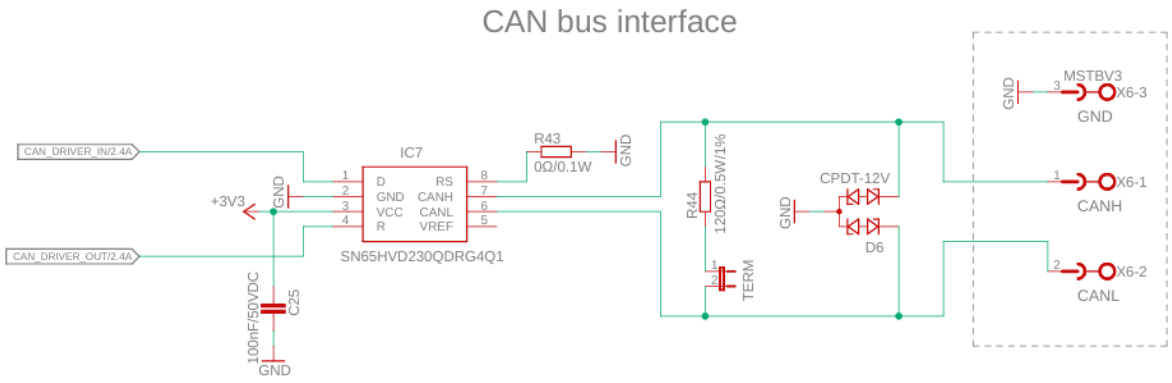
Το πηνίο του ρελέ τροφοδοτείται από +24VDC και η ενεργοποίησή του γίνεται μέσω NPN τρανζίστορ BC547 (T2) σε διάταξη low-side switching. Το σήμα ελέγχου από τον μικροελεγκτή (RELAY_CON) οδηγεί τη βάση του τρανζίστορ μέσω αντίστασης R53 = 10 kΩ, ώστε ο μικροελεγκτής να μην επιβαρύνεται από το ρεύμα του πηνίου και να επιτυγχάνεται ασφαλής οδήγηση.

Για την προστασία του κυκλώματος οδήγησης κατά την απενεργοποίηση του ρελέ, έχει τοποθετηθεί δίοδος flyback SM4007 (D7) παράλληλα με το πηνίο, η οποία αποσβένει την επαγωγική υπέρταση που δημιουργείται όταν καταρρέει το μαγνητικό πεδίο. Επιπλέον, υπάρχει ενδεικτικό LED (KLD1) με αντίσταση R54 = 4.75 kΩ, ώστε η κατάσταση ενεργοποίησης του ρελέ να είναι άμεσα ορατή κατά τον έλεγχο και τη συντήρηση.

Οι επαφές του ρελέ διατίθενται στον χρήστη μέσω της κλέμας X9 και περιλαμβάνουν COM, NO (Normally Open) και NC (Normally Closed). Με αυτόν τον τρόπο ο χρήστης μπορεί να επιλέξει τον επιθυμητό τρόπο λειτουργίας (κανονικά ανοικτό ή κανονικά κλειστό) χωρίς τροποποίηση του υλικού, ενισχύοντας τον γενικού σκοπού χαρακτήρα της συσκευής.

3.2.8 CANbus

Η διεπαφή CAN bus υλοποιείται με τον transceiver SN65HVD230QDRG4Q1 (IC7)[5], ο οποίος γεφυρώνει το επίπεδο λογικής 3.3 V του μικροελεγκτή με το διαφορικό φυσικό επίπεδο του CAN (γραμμές CANH/CANL). Η επικοινωνία με τον μικροελεγκτή γίνεται μέσω των γραμμών CAN_DRIVER_IN και CAN_DRIVER_OUT, ενώ στην πλευρά του διαύλου οι γραμμές CANH και CANL οδηγούνται στην κλέμα X6 μαζί με τη γείωση (GND), ώστε να είναι δυνατή η σύνδεση εξωτερικών κόμβων CAN.



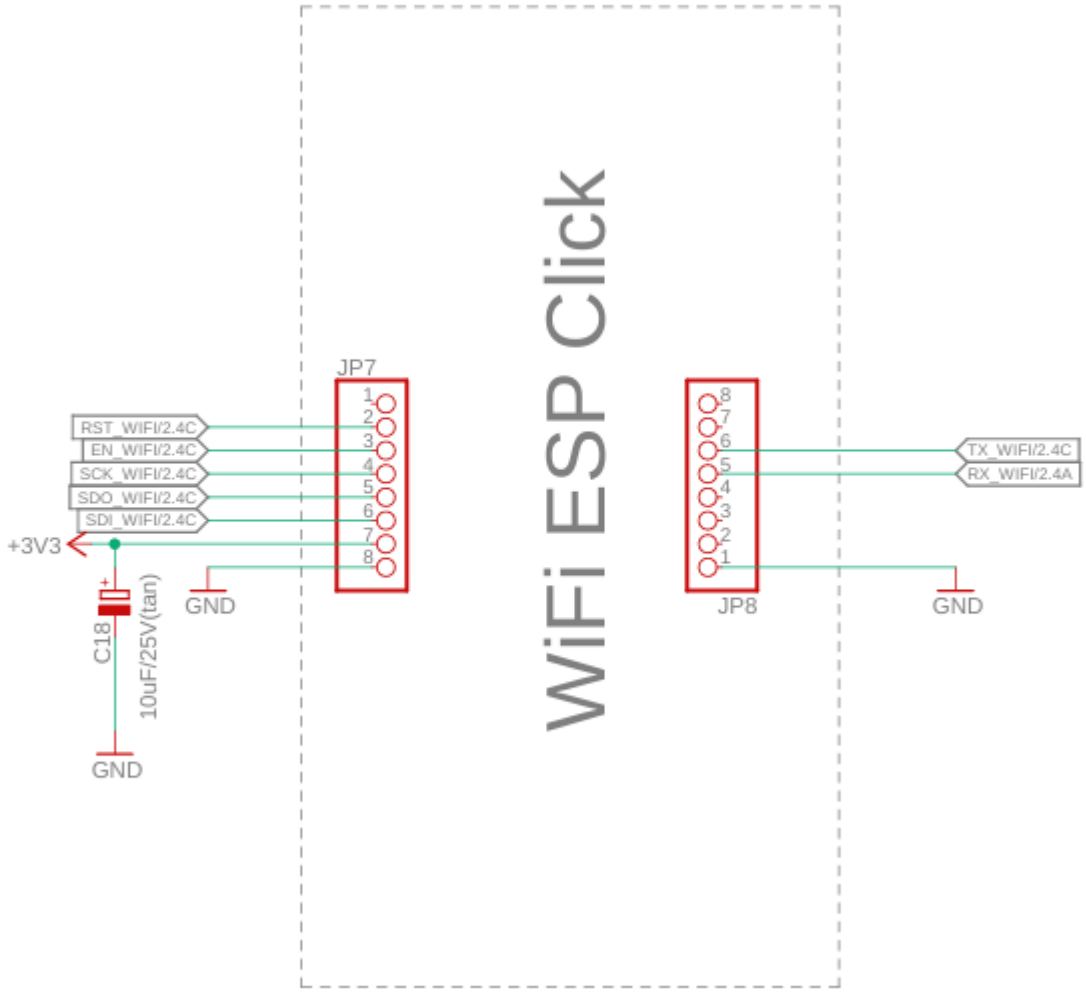
Εικόνα 9 Κύκλωμα CANbus

Για τη σωστή προσαρμογή του διαύλου προβλέπεται τερματισμός 120 Ω (R44) μεταξύ CANH και CANL[3], ο οποίος ενεργοποιείται προαιρετικά μέσω του jumper TERM. Έτσι η πλακέτα μπορεί να λειτουργήσει είτε ως ενδιάμεσος κόμβος είτε ως τελικός κόμβος, ανάλογα με τη θέση της στο δίκτυο.

Σε επίπεδο αξιοπιστίας/ανθεκτικότητας, έχουν ενσωματωθεί προστασίες γραμμής με TVS array (D6, CPDT-12V) προς γείωση, για περιορισμό παροδικών υπερτάσεων/ESD που είναι συχνές σε βιομηχανικές καλωδιώσεις. Παράλληλα, υπάρχει τοπική αποσύζευξη τροφοδοσίας του transceiver με C25 = 100 nF, ενώ το pin ελέγχου κλίσης/λειτουργίας έχει δεθεί στη γείωση μέσω R43 = 0 Ω, σύμφωνα με την επιλεγμένη διαμόρφωση λειτουργίας του transceiver.

3.2.9 WiFi

Για την ασύρματη δικτύωση της συσκευής αξιοποιείται πρόσθετη μονάδα τύπου WiFi ESP Click της MikroElektronika, η οποία ενσωματώνει το ESP-WROOM-02 (SoC ESP8266EX) και παρέχει Wi-Fi στα 2.4 GHz με υποστήριξη 802.11 b/g/n. Η επιλογή add-on μονάδας μειώνει την πολυπλοκότητα της RF σχεδίασης στο κύριο PCB και επιτρέπει αντικατάσταση/αναβάθμιση του υποσυστήματος επικοινωνίας χωρίς ανασχεδίαση της κύριας πλακέτας.



Εικόνα 10 Κύκλωμα WiFi

Η διασύνδεση της μονάδας με τον μικροελεγκτή γίνεται κυρίως μέσω UART, με τα σήματα TX_WIFI και RX_WIFI να καταλήγουν στο pin header JP8 (μαζί με GND). Παράλληλα, μέσω του JP7 έχουν προβλεφθεί γραμμές ελέγχου όπως RST_WIFI και EN_WIFI, καθώς και πρόσθετες γραμμές (SCK/SDO/SDI) που μπορούν να αξιοποιηθούν για βοηθητικές λειτουργίες/παραμετροποίηση από το firmware.

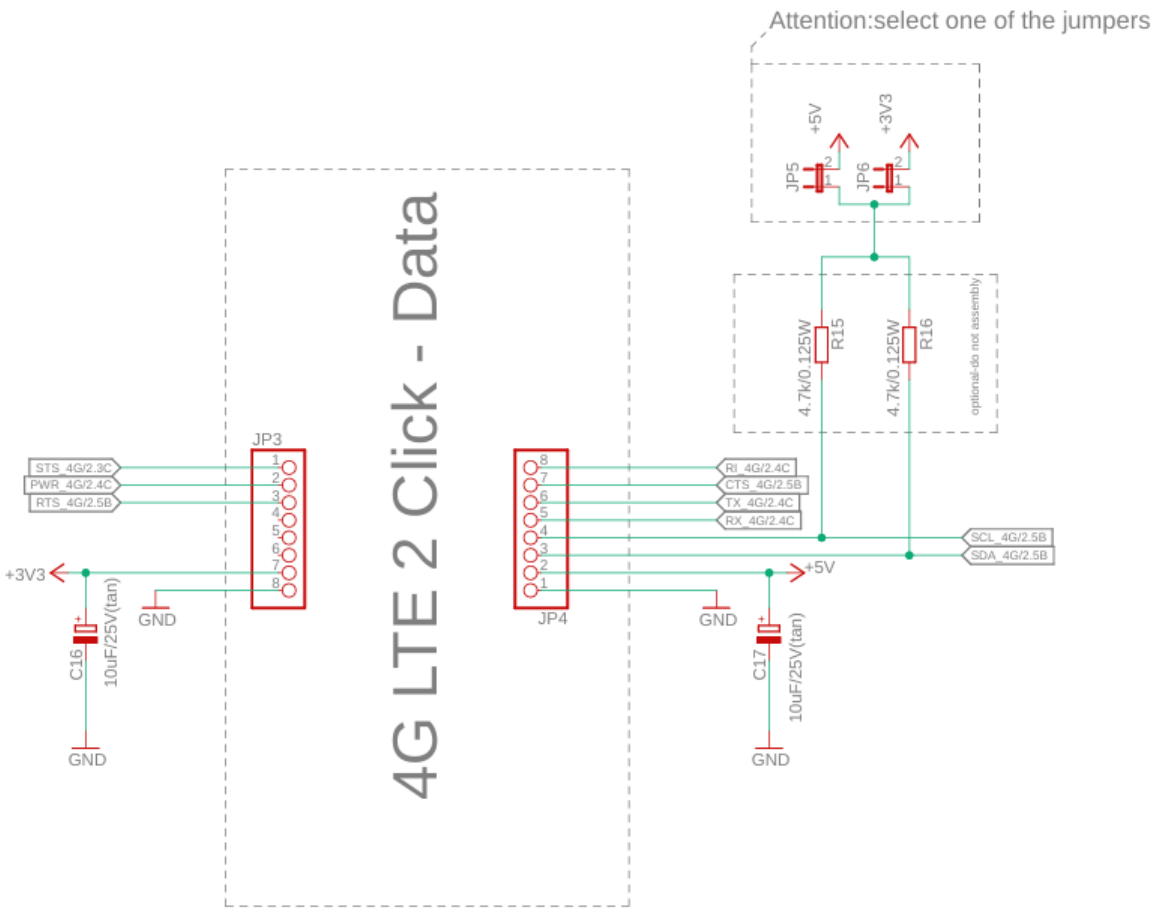
Η τροφοδοσία της μονάδας πραγματοποιείται από τη γραμμή των 3.3V, που είναι εντός των προδιαγραφών λειτουργίας του ESP-WROOM-02 (3.0–3.6 V)[6]. Επειδή κατά τη μετάδοση Wi-Fi εμφανίζονται αυξημένες στιγμιαίες απαιτήσεις ρεύματος, στο κύκλωμα έχει πυκνωτής χωρητικότητας 10μF (C18) κοντά στο pin header τροφοδοσίας, ώστε να μειώνονται βυθίσεις τάσης και να βελτιώνεται η σταθερότητα της λειτουργίας.

Σε επίπεδο δυνατοτήτων δικτύου, το ESP-WROOM-02 υποστηρίζει λειτουργία Station, SoftAP ή συνδυασμό αυτών, καθώς και βασικά πρωτόκολλα δικτύωσης όπως IPv4, TCP/UDP, HTTP και FTP, διευκολύνοντας την ενσωμάτωση σε IoT εφαρμογές με σχετικά "ελαφρύ" κύριο firmware στον μικροελεγκτή.

Εικόνα 11 Mikroelektronika WiFi ESP Click

3.2.10 LTE Modem (4G LTE Cat-1)

Για την παροχή συνδεσιμότητας μέσω δικτύου κινητής τηλεφωνίας έχει προβλεφθεί pin header ώστε να συνδεθεί η πρόσθετη μονάδας 4G LTE 2 Click – Data της Mikroelektronika[7], η οποία βασίζεται στο u-blox LARA-R6001D. Η οικογένεια LARA-R6 υποστηρίζει LTE Cat 1 με δυνατότητα λειτουργίας σε πολλά φάσματα/περιοχές, ενώ η παραλλαγή "D" είναι data-only (χωρίς υποστήριξη voice/audio), κάτι που ταιριάζει στον χαρακτήρα IoT της συσκευής.[8]



Εικόνα 12 Κύκλωμα 4G-LTE

Η επικοινωνία του μικροελεγκτή με το modem υλοποιείται με σειριακή διασύνδεση UART, ώστε ο μικροελεγκτής να στέλνει AT εντολές και να ανταλλάσσει δεδομένα. Ακόμα έχουν προβλεφθεί επιπλέον γραμμές ελέγχου/κατάστασης (RTS/CTS για flow control, RI, καθώς και PWR_4G και STS_4G) ώστε η λειτουργία να είναι πιο αξιόπιστη σε πραγματικές συνθήκες (π.χ. υψηλή κίνηση δεδομένων ή ανάγκη επανεκκίνησης/επιτήρησης της μονάδας).

Σε επίπεδο τροφοδοσίας, η πλακέτα παρέχει τις γραμμές +3.3 V και +5 V, με decoupling πυκνωτές τανταλίου των 10 μF (C16 και C17). Η επιλογή αυτή έγινε επειδή τα LTE modems παρουσιάζουν έντονα μεταβαλλόμενο προφίλ κατανάλωσης κατά τη σύνδεση/μετάδοση, με ρεύματα που μπορούν να φτάσουν σε εκατοντάδες mA (ενδεικτικά έως ~650 mA σε LTE connected mode, ανάλογα με τις συνθήκες).

Τέλος, έχουν προβλεφθεί και οι γραμμές I²C (SCL_4G, SDA_4G), με αντιστάσεις pull-up R15/R16 = 4.7 kΩ, όπου η τάση των pull-ups επιλέγεται μέσω των jumpers JP5/JP6 (επιλογή μεταξύ 3.3 V και 5 V). Με αυτόν τον τρόπο μπορεί να προσαρμόζεται ευκολότερα σε ενδεχόμενες αλλαγές του συστήματος, όπως είναι η επιλογή άλλου modem.

Συνοπτικά, το block 4G/LTE παρέχει:

- εναλλακτικό κανάλι επικοινωνίας μέσω κινητής (LTE Cat 1, data-only)

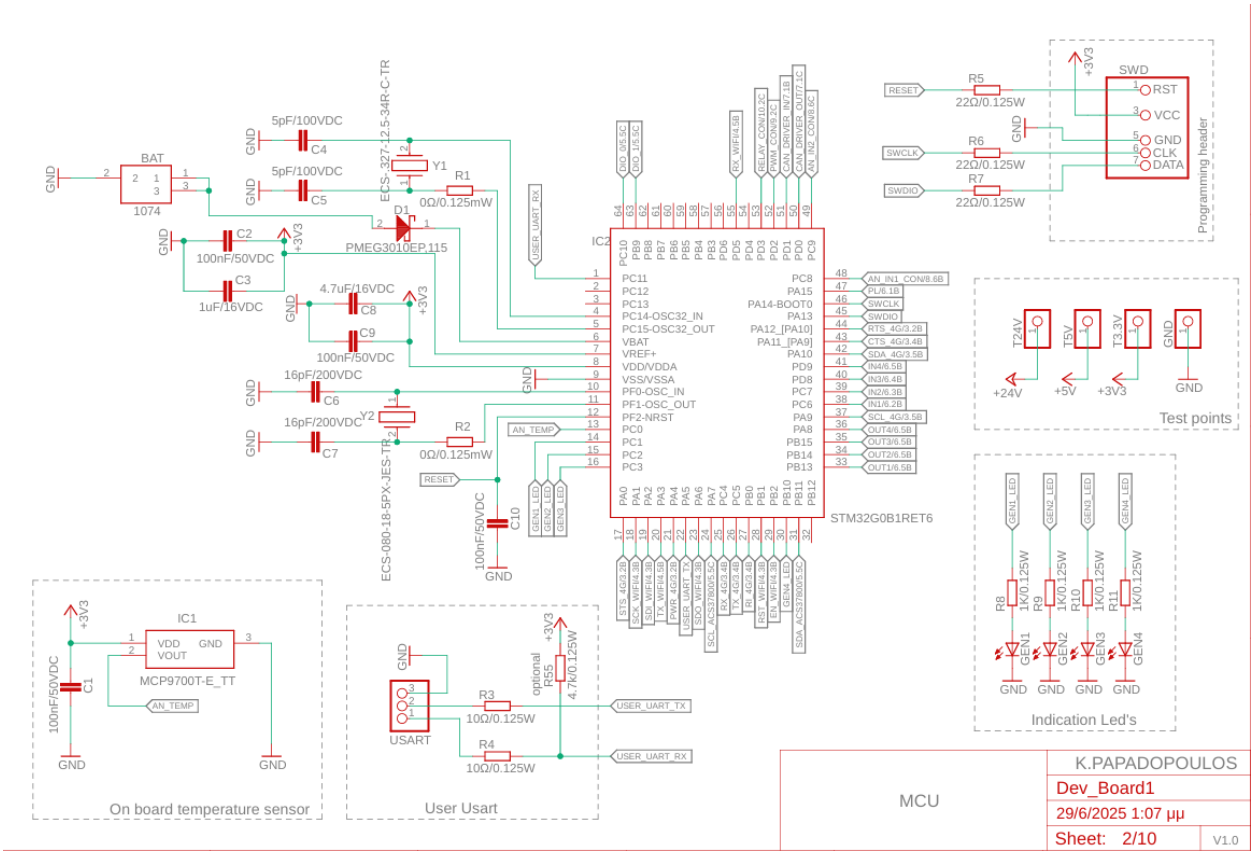
- UART interface με flow control και σήματα επιτήρησης/ελέγχου
- τροφοδοσία και τοπική αποσύζευξη για καλύτερη συμπεριφορά σε φορτία/αιχμές
- I²C γραμμές με επιλέξιμα pull-ups (3.3 V ή 5 V)

Εικόνα 13 Mikroelektronika 4G LTE 2 Click - Data

3.2.11 Μικροελεγκτής STM32 (STM32G0B1RET6)

Ο κεντρικός ελεγκτής της πλακέτας είναι ο STM32G0B1RET6 της STMicroelectronics, βασισμένος σε πυρήνα Arm Cortex-M0+ με μέγιστη συχνότητα λειτουργίας 64 MHz. Η συγκεκριμένη οικογένεια ενσωματώνει μνήμη προγράμματος έως 512 KB Flash και 144 KB SRAM, επαρκείς για firmware που συνδυάζει επικοινωνίες, αποθήκευση παραμέτρων και διαχείριση μετρήσεων.

Στο επίπεδο υλοποίησης της πλακέτας, ο χρονισμός του συστήματος υποστηρίζεται από δύο εξωτερικούς κρυστάλλους: έναν 8MHz για το κύριο ρολόι (Y2 με τα αντίστοιχα C6/C7) και έναν χαμηλής συχνότητας 32.768 kHz (Y1 με C4/C5) για μελλοντική χρήση στο real time clock(RTC). Επίσης, έχει προβλεφθεί τροφοδοσία του backup domain μέσω pin VBAT (υποδοχή BAT), ώστε να μπορεί να διατηρείται λειτουργία RTC και δεδομένα του backup domain σε περίπτωση απώλειας της κύριας τροφοδοσίας. Η επιλογή έχει γίνει σύμφωνα με το “AN2867 Application note” της ST microelectronics[9].



Εικόνα 14 Κύκλωμα μικροελεγκτή STM32

Η τροφοδοσία του STM32 γίνεται από +3.3V, ενώ έχουν προστεθεί και οι αντίστοιχοι decoupling πυκνωτές πολύ κοντά στον μικροελεγκτή, πράγμα το οποίο αναφέρεται και στο datasheet. Για ανάπτυξη και debugging έχει ενσωματωθεί header SWD (SWCLK/SWDIO/NRST) με σειριακές αντιστάσεις 22Ω (R5–R7), προστατεύουν τον μικροελεγκτή αλλά και τον programmer ST-Link V2. Ακόμα, έχει προστεθεί η User UART, σειριακή θύρα για αποσφαλμάτωση και την αρχικοποίηση παραμέτρων του συστήματος. Τέλος υπάρχουν ενδεικτικά LED GEN1–GEN4, καθώς και test points για τις βασικές τροφοδοσίες (+24V, +5V, +3V3, GND).

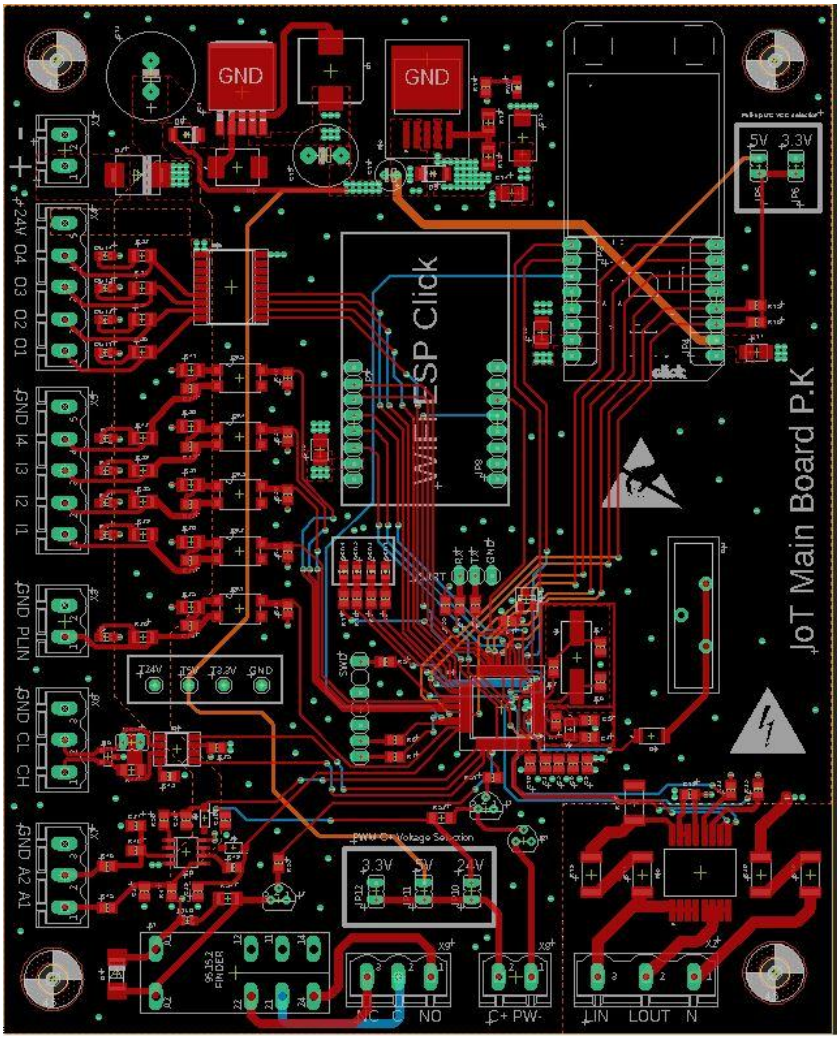
Σε επίπεδο δυνατοτήτων, ο STM32G0B1 παρέχει πλούσιες διεπαφές (6 USART, 3 I²C, 3 SPI, 2 FDCAN), αναλογικά περιφερειακά (ADC/DAC) και πληθώρα timers, καλύπτοντας τις ανάγκες διασύνδεσης της πλακέτας με τα επιμέρους υποσυστήματα (WiFi/LTE, CAN, μετρήσεις, είσοδοι/έξοδοι). Επίσης σε σύγκριση με τις σειρές F απαιτεί λιγότερα εξαρτήματα(π.χ. decoupling πυκνωτές), απλουστεύοντας έτσι το κύκλωμα.

3.2.12 PCB

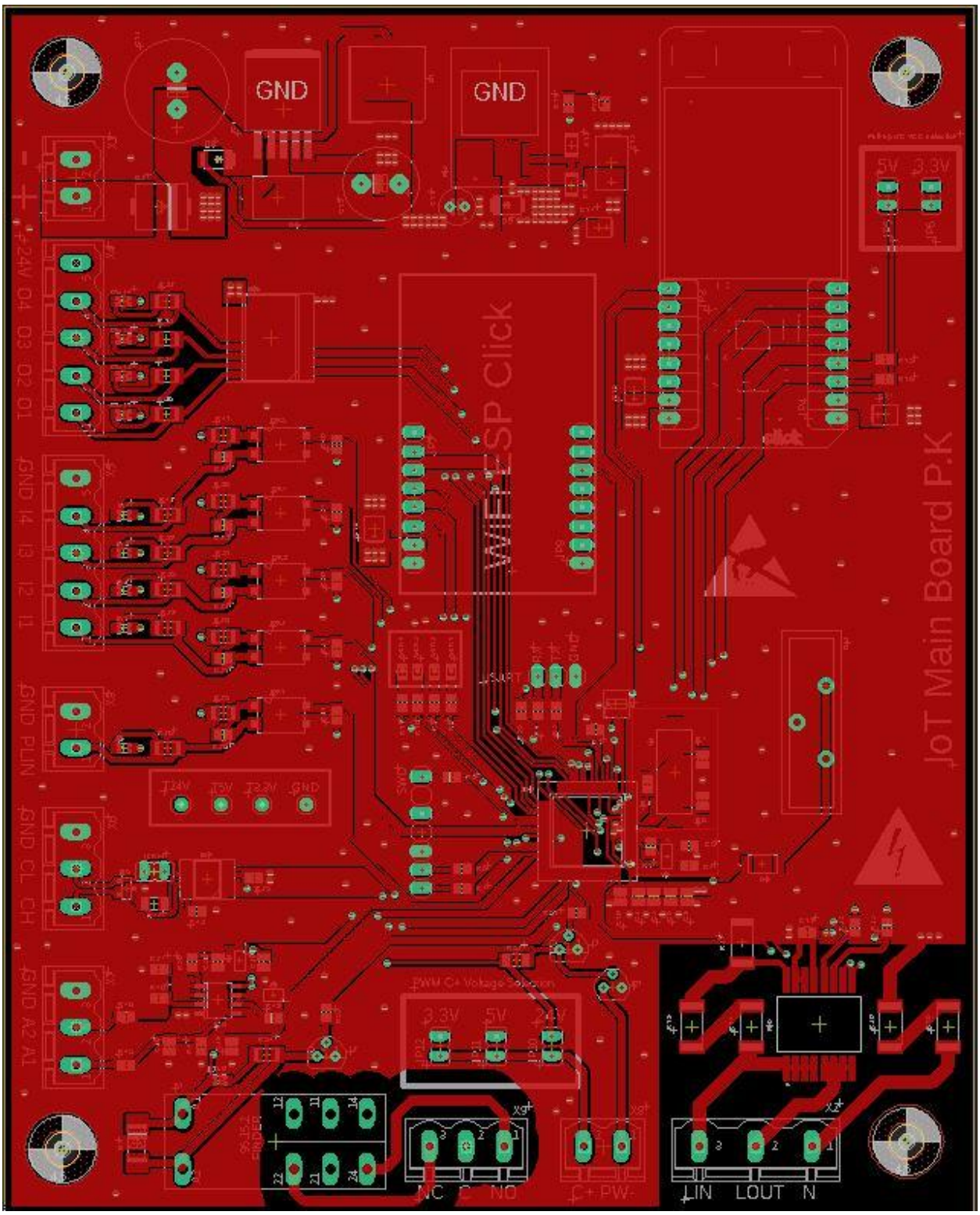
Λεπτομέρειες πλακέτας

- Διαστάσεις: 131.5 mm* 164 mm
- Πάχος 1.6 mm
- Layers: 4
- Υλικό: FR-4
- Χρώμα: Πράσινο
- IPC Class 2 Standard
- Via Covering: Plugged
- Min via hole size/diameter: 0.3mm/(0.4/0.45mm)

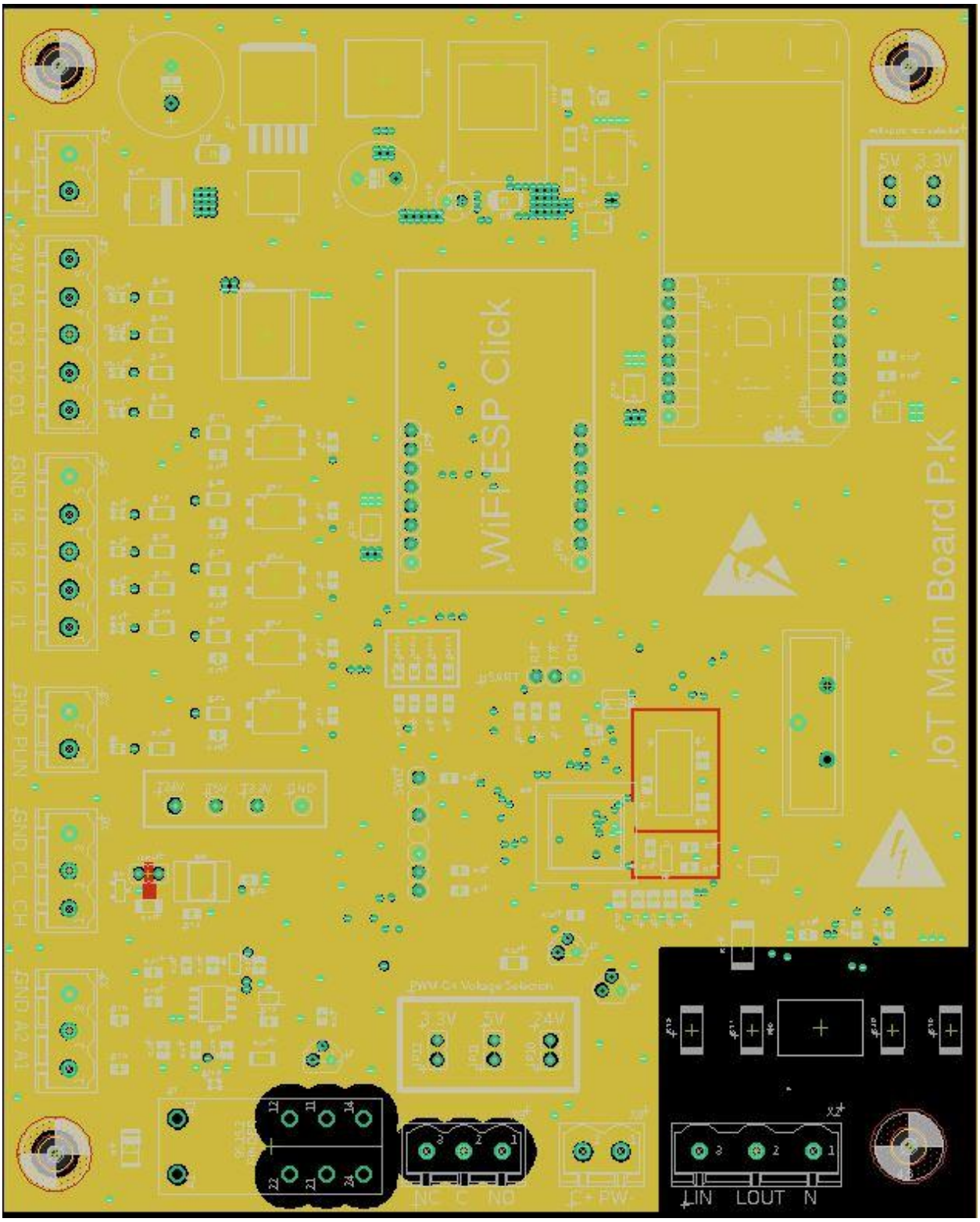
Παρακάτω παρουσιάζονται οι εικόνες από το τυπωμένο της πλακέτας, τα τέσσερα στρώματα(layer), πληροφορίες για το πάχος των layers και φωτογραφία της πλακέτας. Το αρχείο του project με το σχηματικό και το τυπωμένο βρίσκεται στα παραδοτέα αρχεία της διπλωματικής.



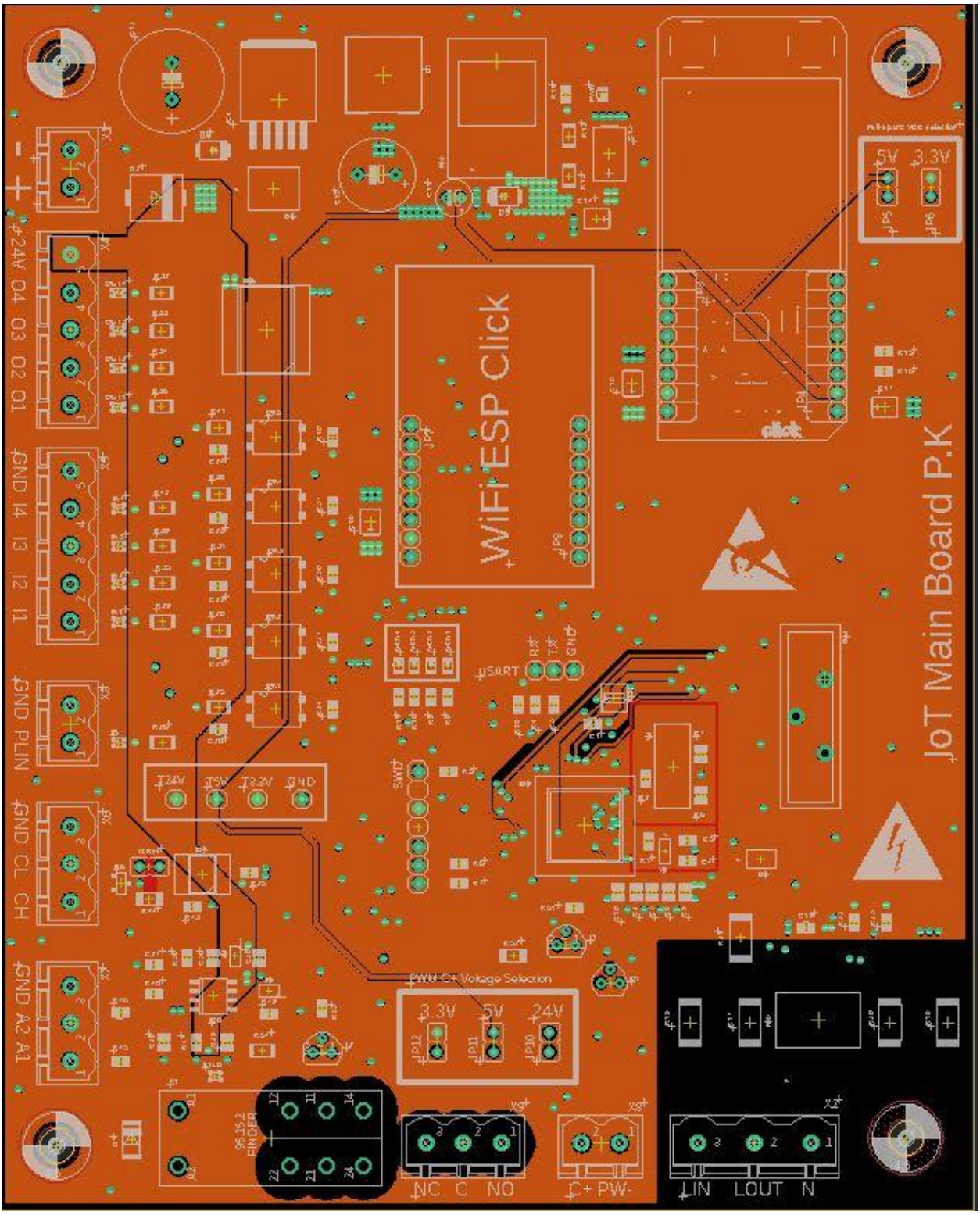
Εικόνα 15 Τυπωμένο πλακέτας



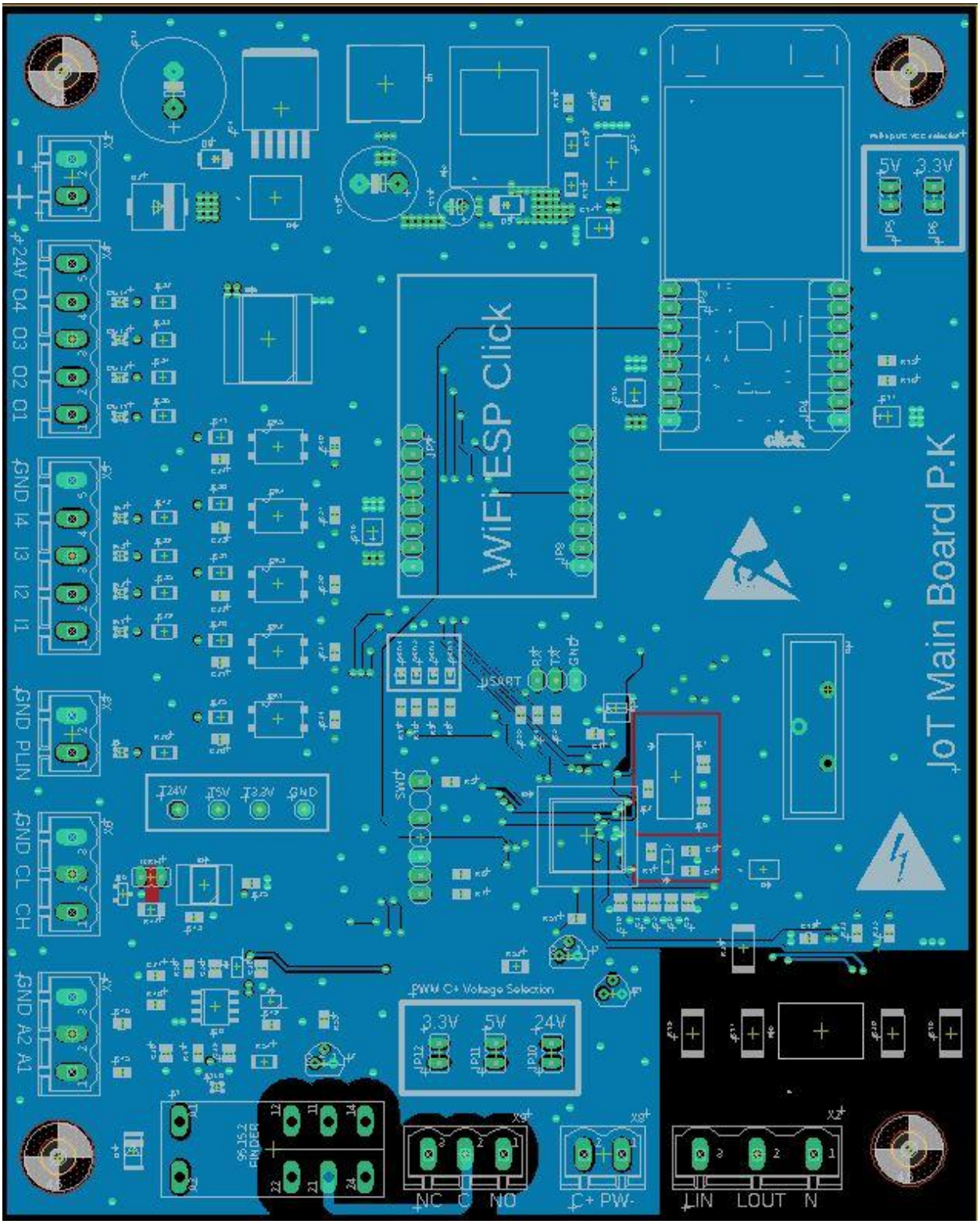
Εικόνα 16 Top Layer (GND Plane)



Εικόνα 17 2nd Layer (GND Plane)

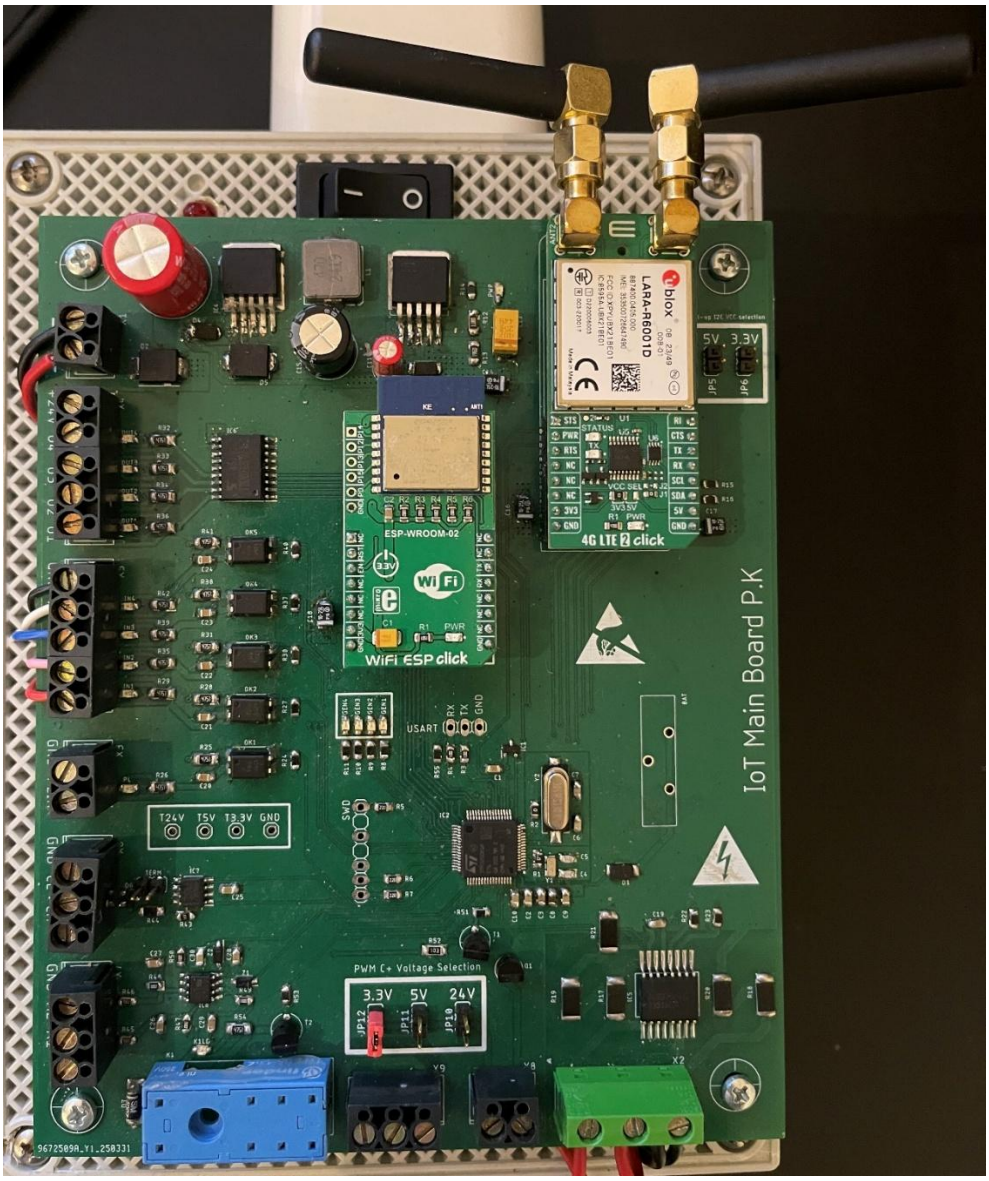


Εικόνα 18 3rd layer (3.3V Plane)



Εικόνα 19 Bottom layer (GND Plane)

Εικόνα 20 Πάχος layer



Εικόνα 21 Πραγματική φωτογραφία της πλακέτας

3.2.13 Κατασκευή και συναρμολόγηση πλακέτας

Διαδικασία συναρμολόγησης

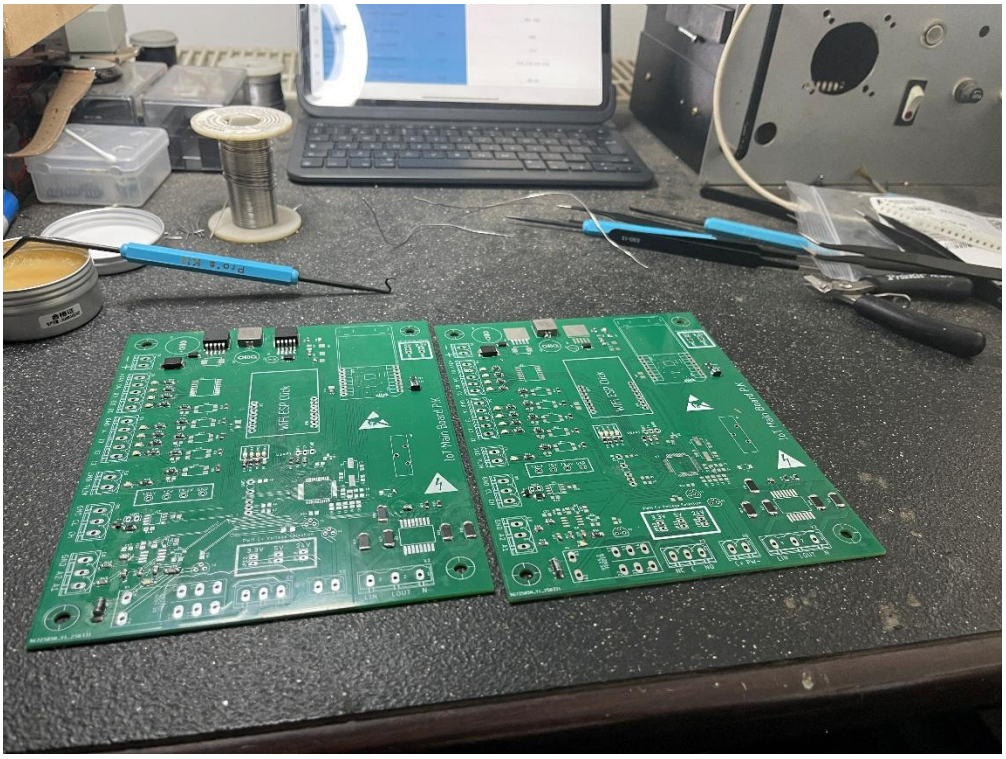
- Όλες οι κολλήσεις (SMD και THT) έγιναν με κολλητήρι
- Στις περισσότερες κολλήσεις, δεδομένου ότι είναι SMD, χρησιμοποιήθηκε λεπτή μύτη στο κολλητήρι
- Η θερμοκρασία στο κολλητήρι ρυθμίστηκε στους 350-400 βαθμούς
- Σε αρκετά σημεία χρησιμοποιήθηκε flux και η περιοχή καθαρίστηκε με οινόπνευμα αμέσως μετά την κόλληση
- Οι κολλήσεις έγιναν τμηματικά ανά block ξεκινώντας με το κύκλωμα τροφοδοσίας, ενώ τελευταία κολλήθηκαν ο επεξεργαστής και οι κλέμες πλακέτας
- Ο στόχος ήταν να κολληθούν δύο πλακέτες ώστε η $2^{η}$ να χρησιμοποιηθεί σαν extension board με CANbus ή ως εφεδρική αν υπάρξει κάποιο πρόβλημα στην $1^{η}$. Έτσι οι κολλήσεις γινόντουσαν για δύο πλακέτας, δηλαδή γινόταν η κόλληση ενός υλικού στην $1^{η}$ πλακέτα και έπειτα στην $2^{η}$. Μετά η διαδικασία συνεχιζόταν για το επομένο υλικό.

Οπτικός έλεγχος και έλεγχοι μετά την κόλληση

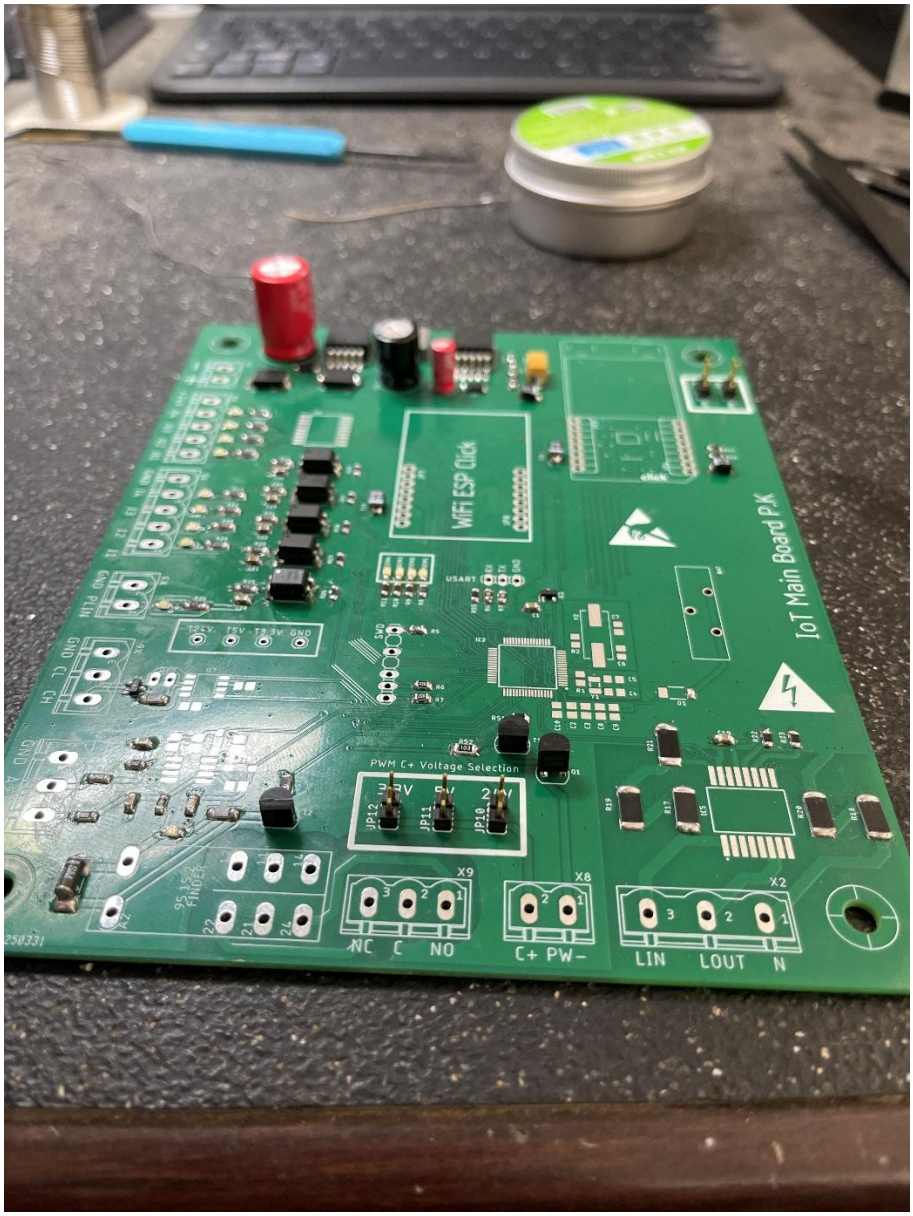
Μετά την ολοκλήρωση των κολλήσεων, η πλακέτα ελέχθηκε οπτικά για ψυχρές κολλήσεις και έπειτα με πολύμετρο για βραχυκυκλώματα στα βασικά σημεία, όπως μεταξύ των 24V, 5V, 3.3V και ως προς GND.

Πρώτη τροφοδότηση και αρχικός έλεγχος λειτουργίας

Η πρώτη τροφοδοσία έγινε χρησιμοποιώντας τροφοδοτικό πάγκου με περιορισμό ρεύματος και χωρίς να έχουν συνδεθεί στην πλακέτα οι έξτρα πλακέτες WiFi και LTE. Αφού τροφοδοτήθηκε η πλακέτα ελέγχθηκαν όλες οι τροφοδοσίες 24V, 5V, 3.3V και έγινε η πρώτη δοκιμή προγραμματισμού του επεξεργαστή. Αμέσως μετά δημιουργήθηκε ένας δοκιμαστικός κώδικας ώστε να ελεγχθούν οι ψηφιακές είσοδοι, έξοδοι, User Uart, Leds.



Εικόνα 22 Πρώτη ημέρα κολλήσεων



Εικόνα 23 Επόμενες ημέρες κολλήσεων

3.2.14 Κοστολόγιο πλακέτας

Λίστα εξαρτημάτων

Παρακάτω παρουσιάζεται η λίστα υλικών με το αντίστοιχο κόστος από την mouser, απ' όπου έγινε και η παραγγελία. Η λίστα υλικών (BOM), η οποία περιλαμβάνει περισσότερες λεπτομέρειες καθώς και link από το site, βρίσκεται στα παραδοτέα αρχεία της διπλωματικής.

Τμχ	Εξάρτημα	Ονομασία στην πλακέτα	Τιμή
6	TSW-102-07-G-S	JP5, JP6, JP10, JP11, JP12, TERM	0.86 €
15	LTST-C171EKT	GEN1, GEN2, GEN3, GEN4, PWR, IN1, IN2, IN3, IN4, K1LD, OUT1, OUT2, OUT3, OUT4, PL	2.28 €
4	RS1-08-G	JP3, JP4, JP7, JP8	1.67 €
3	RC0805FR-100RL	R1, R2, R43	1.43 €
11	08055C104JAT2A	C1, C2, C9, C10, C19, C25, C26, C27, C28, C29, C30	0.10 €
1	T491X107K025AT	C13	2.79 €
2	RC0805FR-07100RL	R49, R50	0.19 €
1	1074	BAT	0.55 €
2	CRCW080510K0JNEC	R51, R53	0.19 €
1	WR12X103 JTL	R52	0.11 €
1	Aluminium Electrolytic Capacitor	C11	0.13 €
4	293D106X9025B2TE3	C14, C16, C17, C18	2.24 €
2	RC0805JR-0710RP	R3, R4	0.19 €
1	CR1206-FX-1180ELF	R12	0.10 €
1	RCS1206120RFKEA	R44	0.21 €
2	08052U160FAT2A	C6, C7	1.05 €
1	RC1206FR-07196RL	R13	0.10 €
5	CRCW08051K00JNEC	R8, R9, R10, R11, R14	0.48 €
4	CRCW25121M00FKEG	R17, R18, R19, R20	1.03 €
1	C0805C105J4RACTU	C3	0.29 €
1	CRCW25122K05FKEG	R21	0.30 €
3	RC0805JR-0722RL	R5, R6, R7	0.29 €
2	RC0805FR-0730KL	R47, R48	0.19 €
1	UVR1V331MPD1TD	C15	0.36 €
15	AC1206FR-074K75L	R25, R26, R28, R29, R31, R32, R33, R34, R35, R36, R38, R39, R41, R42, R54	1.43 €
10	CRCW08054K70JNEB	R15, R16, R24, R27, R30, R37, R40, R55, R22, R23	0.95 €
1	C0805C475J4RACAUTO	C8	0.49 €
5	C0805C474J5RECTU	C20, C21, C22, C23, C24	1.29 €
1	SRP1038AA-470M	L1	1.09 €
2	885012007101	C4, C5	0.19 €
1	Aluminium Electrolytic Capacitor	C12	1.02 €
2	CR0805-FX-7502ELF	R45, R46	0.19 €
1	95.15.2	K1	1.83 €

1	ACS37800KMACTR-030B3-I2C	IC5	4.84 €
2	BC547B	T1, T2	0.21 €
1	BS170	Q1	0.28 €
1	CPDT-12V	D6	0.38 €
1	ECS-.327-12.5-34R-C-TR	Y1	0.58 €
1	ECS-080-18-5PX-JES-TR	Y2	0.30 €
1	11730.1	X2	0.00 €
1	LM2596S-5.0/NOPB	IC4	6.37 €
1	LM358D	IC8	2.95 €
5	LTV-816S	OK1, OK2, OK3, OK4, OK5	1.57 €
1	MCP9700T-E/TT	IC1	0.38 €
1	MIC29302WU	IC3	2.75 €
3	PCB Terminal Block	X1, X3, X8	5.40 €
3	PCB Terminal Block	X6, X7, X9	0.00 €
2	PCB Terminal Block	X4, X5	0.00 €
1	PMEG3010EP,115	D1	0.34 €
2	PMEG4010EP-QX	D3, D4	0.68 €
1	SK34-TP	D5	0.31 €
1	SK54L-TP	D2	0.51 €
1	SM4007	D7	0.16 €
1	SN65HVD230QDRG4Q1	IC7	2.75 €
1	STM32G0B1RET6	IC2	6.48 €
2	SZMMSZ4683T1G	Z1, Z2	0.44 €
1	TBD62083AFWG,EL	IC6	1.70 €
1	WiFi ESP Click	-	17.00 €
1	4G LTE 2 Click – Data	-	148.00 €
Σύνολο			**229.96 €**

Κόστος πλακέτας

Η παραγγελία των πλακετών (5τμχ) έγινε από την JLCPCB με τελικό κόστος 35€ που αντιστοιχεί σε 7€/πλακέτα.

4. Παραμετροποίηση και abstraction του συστήματος

4.1 Έννοια παραμετροποίησης σε IoT συστήματα

Ένα βασικό χαρακτηριστικό των σύγχρονων IoT συστημάτων είναι η δυνατότητα παραμετροποίησης της λειτουργίας τους χωρίς τροποποίηση του hardware ή επαναπρογραμματισμό του firmware. Στην παρούσα εργασία, η παραμετροποίηση υλοποιείται σε επίπεδο εφαρμογής και συσκευής, επιτρέποντας στο ίδιο σύστημα να προσαρμόζεται σε διαφορετικά σενάρια χρήσης.

Η παραμετροποίηση αφορά τόσο την παρουσίαση και επεξεργασία των δεδομένων στον χρήστη, όσο και τη λειτουργική συμπεριφορά της συσκευής (π.χ. χρονισμοί και κανόνες αυτοματισμού).

4.2 Παραμετροποίηση μετρήσεων και παρουσίασης δεδομένων

Οι μετρήσεις που αποστέλλονται από τη συσκευή αποθηκεύονται στη βάση δεδομένων σε ακατέργαστη μορφή (raw values). Η μετατροπή και επεξεργασία των δεδομένων γίνεται στο επίπεδο του frontend, δίνοντας ευελιξία στην απεικόνιση χωρίς αλλοίωση των αρχικών μετρήσεων.

Στο dashboard, ο χρήστης μπορεί να επιλέξει:

- ποιες μετρήσεις θα εμφανίζονται
- τον τρόπο απεικόνισης (widgets)
- και απλές μαθηματικές μετατροπές των τιμών (π.χ. γραμμικές μετατροπές).

Η διαμόρφωση του dashboard αποθηκεύεται στη βάση δεδομένων ανά χρήστη, ώστε η ίδια διάταξη και ρυθμίσεις να φορτώνονται αυτόματα σε κάθε νέα σύνδεση. Με τον τρόπο αυτό, επιτυγχάνεται η εξατομίκευση της πλατφόρμας.

4.3 Παραμετροποίηση συσκευής και επικοινωνίας

Η παραμετροποίηση της συσκευής πραγματοποιείται μέσω της διαδικτυακής πλατφόρμας και αποστέλλεται στη συσκευή με τη χρήση MQTT. Οι βασικές παράμετροι που υποστηρίζονται είναι:

- το χρονικό διάστημα αποστολής των μετρήσεων
- τα στοιχεία σύνδεσης του WiFi (SSID και password)
- γενικές ρυθμίσεις όσον αφορά την λειτουργία.

Οι παράμετροι αποθηκεύονται τόσο στο backend όσο και στη flash μνήμη της συσκευής, ώστε να διατηρούνται μετά από επανεκκίνηση ή διακοπή τροφοδοσίας.

Η επιλογή του τρόπου επικοινωνίας (WiFi μόνο, LTE μόνο ή WiFi με fallback σε LTE) γίνεται κατά την αρχική παραμετροποίηση της συσκευής μέσω της σειριακής θύρας (user UART). Η επιλογή αυτή δεν είναι προσβάσιμη στον τελικό χρήστη της πλατφόρμας, καθώς απαιτεί τεχνική γνώση και αφορά τη βασική ρύθμιση του firmware.

4.4 Rules engine και αυτοματισμοί

Το σύστημα υποστηρίζει μηχανισμό κανόνων (rules engine), ο οποίος επιτρέπει την υλοποίηση απλών αυτοματισμών απευθείας στη συσκευή. Κάθε κανόνας αποτελείται από:

- μία πηγή (ψηφιακή είσοδος, αναλογική μέτρηση, τάση)
- μία συνθήκη
- GT: μεγαλύτερο από (Greater than)
- LT: μικρότερο από (Lower than)
- EQ: ίσο με(equal)
- ACTIVE_FOR: ενεργό για κάποιο χρόνο
- INACTIVE_FOR: ανενεργό για κάποιο χρόνο
- μία τιμή κατωφλίου ή χρονική διάρκεια
- και μία ενέργεια (ενεργοποίηση ψηφιακής εξόδου, ρελέ ή PWM).

Οι κανόνες ορίζονται μέσω της διαδικτυακής πλατφόρμας, αποθηκεύονται στη βάση δεδομένων και αποστέλλονται στη συσκευή μέσω MQTT. Η εκτέλεση των κανόνων γίνεται αποκλειστικά στη συσκευή, εξασφαλίζοντας αυτόνομη λειτουργία ακόμη και σε περίπτωση απώλειας επικοινωνίας με τον server.

Μετά τη λήψη νέων κανόνων, αυτοί αποθηκεύονται στη μνήμη flash της συσκευής και φορτώνονται στη RAM κατά την εκκίνηση, όπου γίνεται και η αρχικοποίηση του συστήματος.

4.5 Abstraction μεταξύ hardware και χρήστη

Σημαντικό στοιχείο της υλοποίησης είναι το επίπεδο abstraction μεταξύ του φυσικού hardware και του χρήστη. Ο χρήστης δεν αλληλεπιδρά με φυσικά pins ή εσωτερικές ονομασίες του μικροελεγκτή, αλλά με λογικές ονομασίες όπως:

- IN1, IN2, IN3, IN4, OUT1, OUT2, OUT3, OUT4
- Voltage, Current, Power
- PWM Frequency, PWM Duty Cycle.

Η αντιστοίχιση μεταξύ λογικών ονομάτων και φυσικών πόρων υλοποιείται τόσο στο backend όσο και στο firmware, επιτρέποντας κοινή ονομασία για όλες τις συσκευές. Με τον τρόπο αυτό, η ίδια διαδικτυακή πλατφόρμα μπορεί να υποστηρίξει πολλαπλές συσκευές χωρίς τροποποίηση του κώδικα.

4.6 Υποστήριξη πολλαπλών χρηστών-συσκευών

Κάθε συσκευή συσχετίζεται με συγκεκριμένο χρήστη μέσω μοναδικού αναγνωριστικού και ξεχωριστών MQTT topics. Τα δεδομένα που λαμβάνονται αποθηκεύονται σε ξεχωριστό λογαριασμό στη βάση δεδομένων, διασφαλίζοντας απομόνωση και ασφάλεια μεταξύ χρηστών. Αυτό γίνεται χρησιμοποιώντας τον όνομα της συσκευής. Για παράδειγμα αν η συσκευή έχει καταχωρηθεί στην βάση ως demo_user_1 τότε:

- Topic για publish μέσω WiFi: GenIoT/demo_user_1/pub
- Topic για subscribe μέσω WiFi: GenIoT/demo_user_1/sub
- Topic για publish μέσω LTE: GenIoT_LTE/demo_user_1/pub
- Topic για subscribe μέσω LTE: GenIoT_LTE/demo_user_1/sub

Η αρχιτεκτονική αυτή επιτρέπει την ταυτόχρονη υποστήριξη πολλαπλών χρηστών-συσκευών, χρησιμοποιώντας κοινό firmware, backend και πλατφόρμα.

4.7 Συμπεράσματα κεφαλαίου

Η προσέγγιση της παραμετροποίησης και του abstraction που υλοποιήθηκε επιτρέπει στο σύστημα να λειτουργεί ως IoT πλατφόρμα γενικού σκοπού. Μέσω απλών ρυθμίσεων και κανόνων, η ίδια συσκευή μπορεί να προσαρμοστεί σε διαφορετικές εφαρμογές, χωρίς αλλαγές στο υλικό ή στον βασικό κώδικα του firmware.

5. Υλοποίηση Firmware της συσκευή IoT

5.1 Εισαγωγή και φιλοσοφία σχεδίασης

Το firmware έχει σχεδιαστεί με στόχο τη σαφή απομόνωση λειτουργιών. Η αρχιτεκτονική βασίζεται σε ανεξάρτητα components, καθένα από τα οποία αναλαμβάνει μία συγκεκριμένη λειτουργία του συστήματος, ενώ η συνολική ροή ελέγχεται από ένα κεντρικό component το οποίο ονομάζεται System.

Η προσέγγιση αυτή επιτρέπει:

- εύκολη συντήρηση και επέκταση του κώδικα
- επαναχρησιμοποίηση των components
- καθαρό διαχωρισμό μεταξύ του κώδικα επικοινωνίας με τα περιφερειακά(basic software), επόμενου επιπέδου που είναι το hardware abstraction layer(HAL) και τον κώδικα που αφορά την εφαρμογή και είναι τα διάφορα components(application software)

Το firmware υλοποιήθηκε σε γλώσσα C και βασίζεται στη βιβλιοθήκη HAL της STMicroelectronics, με το αρχικό configuration των περιφερειακών να έχει παραχθεί μέσω STM32CubeMX. Ο κώδικας έχει γραφεί και γίνει compile με το Keil μVision V.53.1.0.

5.2 System layer και βασική ροή εκτέλεσης

Στον πυρήνα της αρχιτεκτονικής βρίσκεται το System component, το οποίο λειτουργεί ως κεντρικός συντονιστής όλων των υπολοίπων components.

Κατά την εκκίνηση της συσκευής, καλείται η συνάρτηση System_Init()από τη main(). Η συνάρτηση αυτή είναι υπεύθυνη για την αρχικοποίηση όλων των επιμέρους components του συστήματος, όπως:

- επικοινωνίες (WiFi, LTE)
- αισθητήρες και μετρήσεις
- ανάγνωση κανόνων και διάφορων δεδομένων από την flash

Μετά την ολοκλήρωση της αρχικοποίησης, το σύστημα εισέρχεται στον κύριο βρόχο εκτέλεσης (while(1)), όπου καλείται επαναλαμβανόμενα η συνάρτηση System_MainTask().

Η System_MainTask() καλεί κυκλικά τις main task συναρτήσεις κάθε component σε προκαθορισμένη σειρά. Η προσέγγιση αυτή αποφεύγει τη χρήση λειτουργικού συστήματος (RTOS), μειώνοντας την πολυπλοκότητα, αφού επαρκεί για τις απαιτήσεις της εφαρμογής.

5.3 Διαχείριση interrupts και buffers

Η διαχείριση των ασύγχρονων γεγονότων (κυρίως επικοινωνίας) βασίζεται σε interrupts, τα οποία χρησιμοποιούνται αποκλειστικά για:

- τη λήψη δεδομένων από UART
- ESP32
- LTE module
- User Uart
- την τοποθέτηση των δεδομένων σε ring buffers.

Η επεξεργασία των δεδομένων δεν πραγματοποιείται μέσα στα interrupts, αλλά μεταφέρεται στο επίπεδο των components και εκτελείται όταν αυτά καλούνται από τη System_MainTask(). Για τον σκοπό αυτό

χρησιμοποιείται κοινός μηχανισμός RingBuffer, ο οποίος επιτρέπει ασφαλή και αποδοτική μεταφορά δεδομένων από το interrupt context στο application context.

Η προσέγγιση αυτή εξασφαλίζει:

- μικρό και προβλέψιμο χρόνο εκτέλεσης της κάθε interrupt
- αποφυγή race conditions
- μηδενική καθυστέρηση, καθώς δεν πραγματοποιείται καμία χρονοβόρα ενέργεια όσο είναι ενεργή η κάθε interrupt.

5.4 Component-based αρχιτεκτονική firmware

Κάθε λειτουργία της συσκευής υλοποιείται ως ανεξάρτητο component, με δικό του αρχείο πηγαίου κώδικα(.c) και αντίστοιχο header(.h). Αυτά τα component είναι τα εξής:

- **WiFi:** διαχείριση της ασύρματης σύνδεσης WiFi
- **MQTT:** υλοποίηση MQTT επικοινωνίας μέσω WiFi
- **LTE:** διαχείριση LTE modem και MQTT επικοινωνίας μέσω του δικτύου κινητής τηλεφωνίας
- **ACS37800:** ανάγνωση και επεξεργασία μετρήσεων AC τάσης, ρεύματος και ισχύος
- **DIO:** διαχείριση ψηφιακών εισόδων και εξόδων
- **PWM:** ρύθμιση και μέτρηση συχνότητας και duty cycle
- **CANbus:** επικοινωνία με πλακέτα επέκτασης μέσω CAN
- **Extra:** διαχείριση επιπλέον λειτουργιών, όπως το onboard relay
- **User UART:** διαχείριση επικοινωνίας με υπολογιστή για αποσφαλμάτωση (debug) καθώς και αρχική παραμετροποίηση
- **Flash:** αποθήκευση και ανάκτηση δεδομένων από τη flash μνήμη
- **Parser:** ανάλυση δεδομένων που λαμβάνονται μέσω MQTT και δημιουργία πακέτων προς απόστολή
- **Rules:** υλοποίηση του μηχανισμού κανόνων

Όλα τα components ακολουθούν κοινό μοτίβο, με συναρτήσεις αρχικοποίησης και κυκλικής εκτέλεσης, διευκολύνοντας την ενοποίηση στο system layer.

5.5 IoMap – Abstraction layer μεταξύ rules και hardware

Κεντρικό ρόλο στην αρχιτεκτονική του firmware παίζει το IoMap component, το οποίο λειτουργεί ως το ενδιάμεσο επίπεδο (abstraction layer) μεταξύ των κανόνων (rules) και του φυσικού hardware.

Το IoMap υλοποιεί δύο βασικές λειτουργίες:

- **IoMap_GetInputValue():** αντιστοιχίζει μία λογική είσοδο (π.χ. Voltage RMS) με την αντίστοιχη μέτρηση από το κατάλληλο component
- **IoMap_SetOutput():** αντιστοιχίζει μία λογική ενέργεια (π.χ. ενεργοποίηση ρελέ) με την αντίστοιχη συνάρτηση άλλου component

Η χρήση switch-case δομών επιτρέπει σαφή και ελεγχόμενη αντιστοίχιση, χωρίς το Rules component να έχει γνώση για το ποιο hardware ή ποιο component υλοποιεί κάθε λειτουργία. Με τον τρόπο αυτό επιτυγχάνεται πλήρης αποσύνδεση της λογικής αυτοματισμού από το hardware.

5.6 Μηχανισμός κανόνων και τοπική εκτέλεση αυτοματισμών

Οι κανόνες ορίζονται από τον χρήστη μέσω της διαδικτυακής πλατφόρμας και αποστέλλονται στη συσκευή μέσω MQTT. Μετά τη λήψη τους:

- φορτώνονται στη RAM
- αποθηκεύονται στη flash μνήμη
- και εκτελούνται τοπικά στη συσκευή.

Κάθε κανόνας περιλαμβάνει μία συνθήκη (π.χ. GT, LT, ACTIVE_FOR) και μία ενέργεια. Κατά την εκτέλεση, το Rules component:

1. ζητά την τρέχουσα τιμή εισόδου μέσω του IoMap
2. αξιολογεί τη συνθήκη
3. και, αν πληρούνται τα κριτήρια, ενεργοποιεί την αντίστοιχη ενέργεια μέσω του IoMap.

Η τοπική εκτέλεση των κανόνων επιτρέπει τη λειτουργία της συσκευής ακόμη και χωρίς ενεργή σύνδεση με το backend, αυξάνοντας την αξιοπιστία του συστήματος.

5.7 Επικοινωνία και ροή δεδομένων

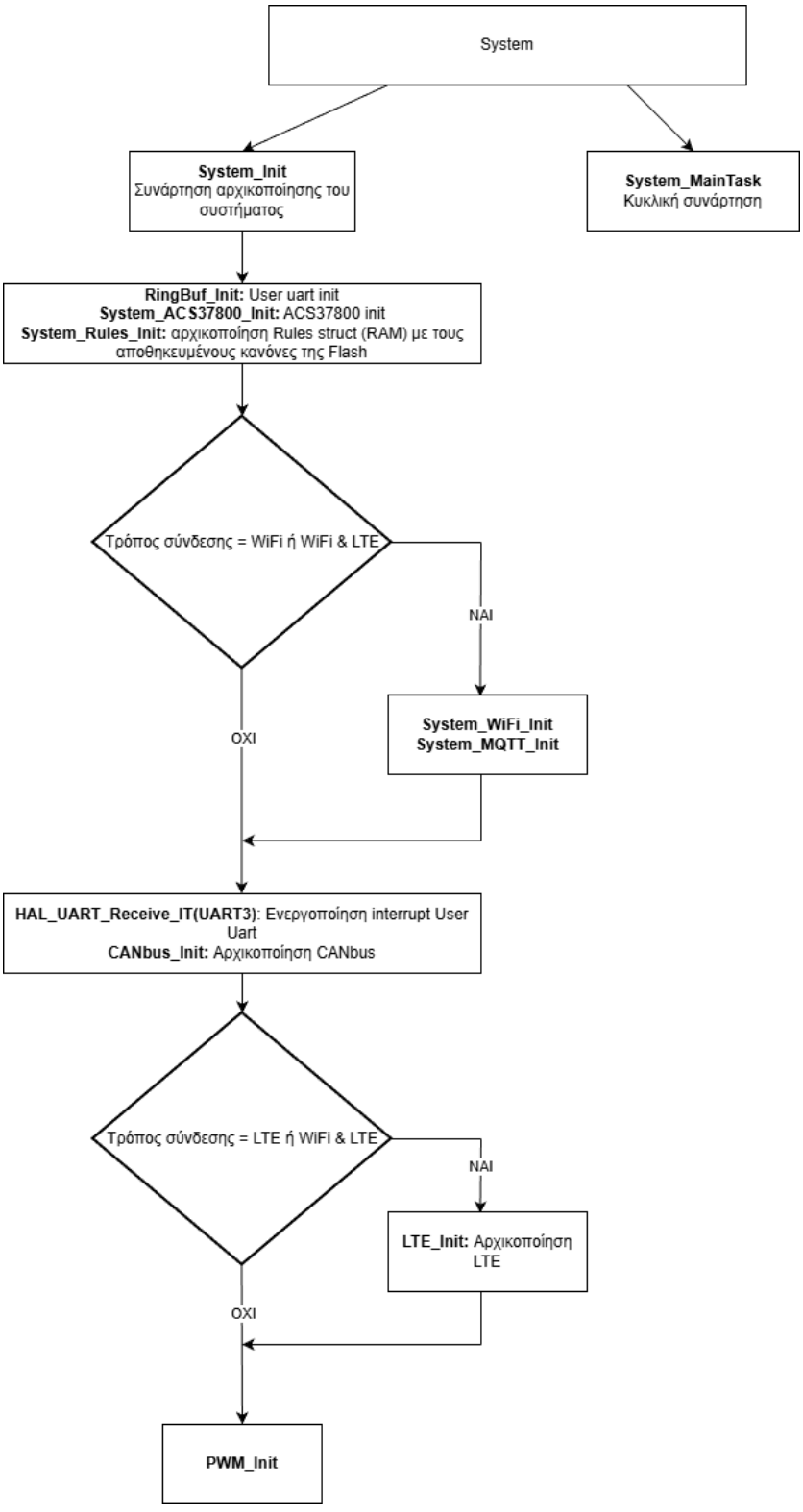
Η συσκευή υποστηρίζει δύο ανεξάρτητους τρόπους επικοινωνίας:

- WiFi + MQTT, όπου το MQTT component χρησιμοποιείται αποκλειστικά για WiFi επικοινωνία
- LTE, όπου το MQTT ενσωματώνεται στο LTE component.

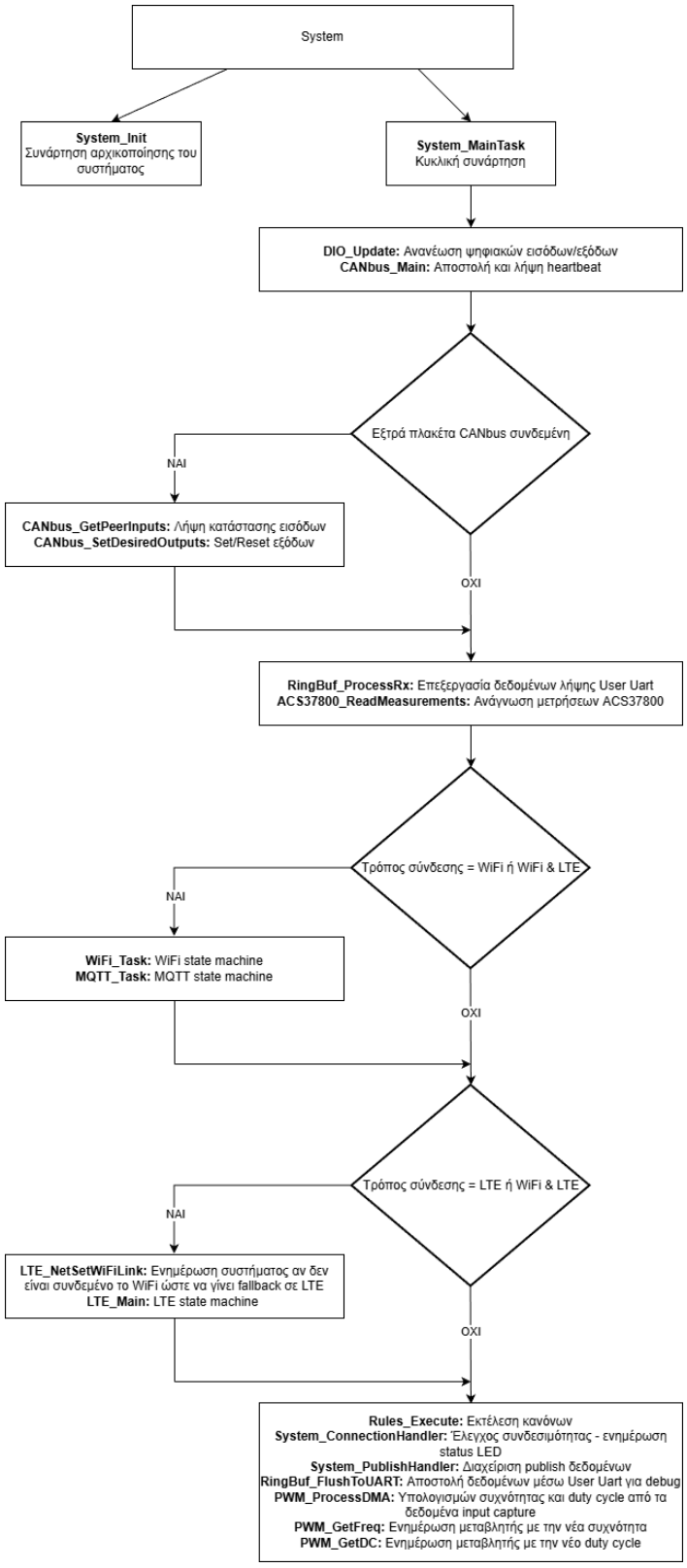
Η επιλογή του τρόπου επικοινωνίας γίνεται κατά την αρχική παραμετροποίηση μέσω User UART, ενώ σε λειτουργία WiFi + LTE εφαρμόζεται μηχανισμός fallback, όπου το LTE χρησιμοποιείται σε περίπτωση απώλειας WiFi σύνδεσης.

Τα δεδομένα μετρήσεων και κατάστασης αποστέλλονται περιοδικά, με χρονισμό που μπορεί να παραμετροποιηθεί και αποθηκεύεται στη flash.

5.8 Block διαγράμματα



Εικόνα 24 Μπλοκ διάγραμμα αρχικοποίησης συστήματος



Εικόνα 25 Μπλοκ διάγραμμα κυκλικών διεργασιών

5.9 Συμπεράσματα κεφαλαίου

Η αρχιτεκτονική του firmware υλοποιεί μία σαφή, modular και επεκτάσιμη προσέγγιση, η οποία επιτρέπει την υποστήριξη πολλαπλών λειτουργιών και εφαρμογών χωρίς αλλαγές στον βασικό κώδικα. Τα δυνατά σημεία είναι ο συνδυασμός component-based σχεδίασης, abstraction μέσω IoMap και τοπικής εκτέλεσης κανόνων.

6. Backend και βάση δεδομένων

Το backend της πλατφόρμας λειτουργεί ως ενδιάμεσος κρίκος ανάμεσα στη συσκευή IoT και στο web περιβάλλον του χρήστη. Σε κανονική λειτουργία, η συσκευή συλλέγει μετρήσεις (π.χ. ψηφιακές εισόδους, αναλογικές τιμές, Vrms/Irms/ισχύ, θερμοκρασία) και τις αποστέλλει περιοδικά μέσω MQTT στο αντίστοιχο topic δημοσίευσης (uplink). Ο MQTT broker προωθεί το μήνυμα στον Python server, ο οποίος αποκωδικοποιεί το payload, το μετατρέπει σε εσωτερική μορφή δεδομένων και το αποθηκεύει στη βάση (ιστορικό μετρήσεων). Ταυτόχρονα, οι τελευταίες τιμές προωθούνται άμεσα στο dashboard μέσω WebSocket, ώστε ο χρήστης να βλέπει real-time κατάσταση χωρίς ανανέωση σελίδας. Από την πλευρά του χρήστη, όταν οριστούν ρυθμίσεις ή κανόνες αυτοματισμού (π.χ. “αν DI1 είναι ενεργή >2 s, ενεργοποίησε έξοδο”), το backend τις καταγράφει στη βάση και δημιουργεί ένα αντίστοιχο μήνυμα εντολής (downlink) σε μορφή JSON. Η εντολή δημοσιεύεται στο topic συνδρομής της συσκευής (sub), η συσκευή την παραλαμβάνει, αποθηκεύει τις νέες παραμέτρους/κανόνες και εφαρμόζει την ενέργεια όταν ικανοποιούνται οι συνθήκες. Η ίδια ροή υποστηρίζεται τόσο μέσω τοπικού δικτύου WiFi όσο και μέσω LTE, χρησιμοποιώντας διαφορετικό broker αλλά κοινή λογική μηνυμάτων.

Στις επόμενες ενότητες αναλύεται λεπτομερώς η υλοποίηση των παραπάνω στα επίπεδα MQTT, parsing, αποθήκευσης και αποστολής εντολών.

6.1 Ρόλος και αρχιτεκτονική του backend

Το backend της πλατφόρμας υλοποιείται ως μία υπηρεσία σε Python που λειτουργεί ταυτόχρονα ως:

1. Web εφαρμογή για την εποπτεία/παραμετροποίηση (σελίδες + HTTP API)
2. Κόμβος συλλογής και δρομολόγησης δεδομένων IoT μέσω MQTT, με δυνατότητα:
    - λήψης μετρήσεων από τη συσκευή
    - ανάλυσης (parsing) και κανονικοποίησης (normalization) των μηνυμάτων
    - αποθήκευσης σε βάση MySQL/MariaDB [10][11]
    - αποστολής ρυθμίσεων/κανόνων πίσω στη συσκευή (downlink).

Η σχεδίαση είναι event-driven: τα δεδομένα “έρχονται” ασύγχρονα στο backend (MQTT callbacks), και το backend αντιδρά άμεσα ενημερώνοντας:

- τη βάση δεδομένων (ιστορικό)
- και το dashboard σε πραγματικό χρόνο (μέσω WebSocket/SocketIO).

Η επιλογή αυτή είναι ιδιαίτερα κατάλληλη για IoT, όπου ο ρυθμός αποστολής δεδομένων είναι συνεχής και δεν ταιριάζει σε μοντέλο “polling” από τον browser.

Για λόγους σαφήνειας, η επικοινωνία διαχωρίζεται σε δύο κατηγορίες ροών. Η πρώτη είναι η ροή δεδομένων (data-plane), η οποία αφορά αποκλειστικά την αποστολή τηλεμετρίας από τη συσκευή προς το backend (uplink), την ανάλυση του payload, την αποθήκευση στη βάση και την προώθηση των τιμών στο dashboard. Η δεύτερη είναι η ροή ελέγχου (control-plane), η οποία αφορά τις εντολές που παράγει η πλατφόρμα (ρυθμίσεις συσκευής και κανόνες αυτοματισμού) και αποστέλλονται προς τη συσκευή μέσω MQTT (downlink). Ο διαχωρισμός αυτός βοηθά στη μελέτη του συστήματος, επειδή οι δύο ροές έχουν διαφορετικές απαιτήσεις: το data-plane δίνει έμφαση στη συχνότητα/όγκο μετρήσεων και στην αποθήκευση, ενώ το control-plane δίνει έμφαση στην αξιοπιστία παραλαβής και στη σωστή εφαρμογή των κανόνων στη συσκευή.

- Data-plane (uplink): telemetry → parsing → DB → WebSocket
- Control-plane (downlink): rules/config → DB → MQTT publish → device apply

6.2 Βιβλιοθήκες και λογισμικό υπόβαθρο

Το backend χρησιμοποιεί στοχευμένα λίγες βιβλιοθήκες, ώστε να παραμένει απλό και ευανάγνωστο:

- **Flask:** web framework για σελίδες και REST endpoints (login, config, rules, measurements)
- **Flask-SocketIO:** σε συνδυασμό με eventlet: προσφέρει WebSocket επικοινωνία για real-time ενημέρωση του dashboard (push δεδομένων στον χρήστη χωρίς refresh)
- **paho-mqtt:** MQTT client που αναλαμβάνει τη σύνδεση στον broker, το subscribe στα topics και την εκτέλεση callbacks όταν καταφθάνει νέο μήνυμα
- **mysql-connector-python:** σύνδεση και εκτέλεση queries σε MySQL/MariaDB
- **bcrypt:** ασφαλής έλεγχος κωδικών πρόσβασης μέσω password hashing.

Στην πράξη, το Flask εξυπηρετεί το UI, ενώ το paho-mqtt "τρέχει στο παρασκήνιο" και τροφοδοτεί το σύστημα με τηλεμετρία.

6.3 Επικοινωνία με MQTT: δύο κανάλια (WiFi / LTE)

Η πλατφόρμα υποστηρίζει δύο τρόπους επικοινωνίας με την ίδια λογική topics και payload, αλλά διαφορετικό broker:

1. **WiFi / τοπικό δίκτυο (LAN):** broker μέσα στο ίδιο δίκτυο με τη συσκευή και τον server
2. **LTE / δημόσιο δίκτυο:** χρήση του public broker test.mosquitto.org.

6.3.1 MQTT clients

Στον κώδικα, δημιουργούνται δύο ανεξάρτητοι MQTT clients (WiFi client και LTE client). Κάθε client:

- συνδέεται στον αντίστοιχο broker
- κάνει subscribe στα topics που περιέχουν τα uplink δεδομένα
- δρομολογεί τα μηνύματα στην ίδια κοινή συνάρτηση επεξεργασίας.

Αυτή η επιλογή απλοποιεί τον σχεδιασμό: η "επιχειρησιακή λογική" (parsing, αποθήκευση, ενημέρωση dashboard) παραμένει ίδια, ενώ αλλάζει μόνο η πηγή του μηνύματος (WiFi broker ή LTE broker).

6.3.2 Δομή topics και διαχωρισμός χρηστών

Για να υποστηρίζονται πολλοί χρήστες/συσκευές χωρίς ανάμειξη δεδομένων, χρησιμοποιείται namespace στο topic:

- **Uplink WiFi:** GenIoT/<username>/pub
- **Downlink WiFi:** GenIoT/<username>/sub
- **Uplink LTE:** GenIoT_LTE/<username>/pub
- **Downlink LTE:** GenIoT_LTE/<username>/sub

Με αυτόν τον τρόπο:

- Το backend μπορεί να διαχωρίσει από ποιον χρήστη προήλθε το μήνυμα χωρίς να βασιστεί στο payload

- Η συσκευή κάνει subscribe μόνο στο δικό της <username>/sub, άρα λαμβάνει μόνο τις δικές της εντολές.

6.3.3 Running mode: threads + MQTT loop

Οι MQTT clients εκτελούνται σε ξεχωριστά daemon threads, μέσα στα οποία καλείται loop_forever(). Έτσι:

- ο web server (Flask/SocketIO) συνεχίζει να λειτουργεί κανονικά
- ενώ οι callbacks του MQTT εκτελούνται παράλληλα κάθε φορά που έρχεται μήνυμα.

6.4 Μορφή δεδομένων και parsing

6.4.1 Κύριο format: JSON telemetry packet

Η συσκευή (ιδίως σε WiFi mode) στέλνει δομημένα μήνυμα τύπου JSON. Το backend ξεκινά πάντα προσπαθώντας να κάνει json.loads() στο πακέτο. Αν το αποτέλεσμα είναι αντικείμενο (dictionary) και το πεδίο type είναι "telemetry", τότε θεωρεί ότι πρόκειται για κανονικό πακέτο τηλεμετρίας.

Η δομή του πακέτου είναι “ομαδοποιημένη” σε επιμέρους ενότητες, π.χ.:

- acs για μετρήσεις ενέργειας/ρεύματος/τάσης (Vrms, Irms, P, Vinst, Iinst)
- ψηφιακές εισόδους (din)
- ψηφιακές εισόδους από πλακέτα επέκτασης (ext_din)
- PWM in (pwm)
- αναλογικές εισόδους (ai)
- θερμοκρασία (temp)

Η επιλογή ομαδοποίησης κάνει το payload πιο καθαρό: κάποιος βλέπει άμεσα ότι π.χ. τα Vrms/Irms/P είναι στο ίδιο “υποσύστημα”.

```
C:\Programs\mosquitto>mosquitto_sub -h test.mosquitto.org -t "GenIoT_LTE/demo_user_1/pub" -v
GenIoT_LTE/demo_user_1/pub {type:telemetry,device:GenIoT-STM32,ts:120366,acs:{Vrms:0.00,Irms:0.000,P:0.00,Vinst:0.00,Iinst:0.000
},din:[1,1,1,0],ext_din:[0,0,0,0],ai:{a1mV:5000,a2mV:3000},pwm:{f:1000.0,d:50.0},temp:50.0}
```

Εικόνα 26 Πακέτο τηλεμετρίας μέσω LTE

6.4.2 LTE – Ιδιαιτερότητα στο format

Στο LTE μπορεί να εμφανιστούν πακέτα που μοιάζουν με JSON αλλά δεν είναι αυστηρό (π.χ. χωρίς εισαγωγικά στα keys). Αν το πρώτο json.loads() αποτύχει, το backend εφαρμόζει έναν δεύτερο μηχανισμό: μετατροπή JS-like object literal σε κανονικό JSON.

Η λογική είναι:

- εντοπίζονται κλειδιά της μορφής key: και μετατρέπονται σε "key":
- εντοπίζονται “bareword” τιμές που είναι strings και μετατρέπονται σε "value"
- αριθμοί/boolean παραμένουν ως έχουν

Έτσι, το backend είναι πιο ανθεκτικό σε μικρές αποκλίσεις του LTE stack ή σε διαφορετικές εκδόσεις firmware.

6.4.3 Fallback parsing: “simple id/value”

Αν το μήνυμα δεν αναγνωριστεί ως structured telemetry, υπάρχει fallback parser για απλούστερες μορφές (π.χ. id=10 value=230.1 ts=... ή μικρό JSON με id/value). Αυτό χρησιμοποιείται:

- για δοκιμές
- για συμβατότητα με πιθανές αλλαγές στο firmware – προηγούμενες εκδόσεις
- για debugging κατά τη φάση ανάπτυξης.

Αν ούτε αυτό πετύχει, το backend κρατά log “Unparsed payload…”, κάτι που βοηθά στη διάγνωση προβλημάτων (π.χ. όταν αλλάζει format στο firmware).

6.5 Πακέτο τηλεμετρίας - εσωτερικά IDs - real-time ενημέρωση

6.5.1 Μοντέλο “signals” με IDs

Για να μπορεί το UI και ο μηχανισμός κανόνων να δουλεύουν με σταθερό τρόπο, ο backend αντιστοιχίζει τις τιμές σε αριθμητικά IDs σημάτων (signal IDs). Παραδείγματα:

- DI1..DI4 → IDs 0..3
- Ext DI1..DI4 → IDs 4..7
- Vrms/Irms/P → IDs 10..12
- PWM In (freq/duty) → IDs 30..31
- Vinst/Iinst → IDs 40..41
- Temperature → ID 50

Αυτό το σχήμα έχει πρακτικό όφελος:

- Οι κανόνες μπορούν να αναφέρονται σε “source_id” αντί για strings
- το dashboard μπορεί να ζητά “δώσε μου το ID=10 (Vrms)” ανεξάρτητα από το πώς ονομάζεται στο payload.

6.5.2 Cache “latest values” και SocketIO push

Το backend διατηρεί ένα in-memory λεξικό latest με την τελευταία γνωστή τιμή ανά signal ID. Κάθε φορά που έρχεται μία νέα τηλεμετρία:

1. ενημερώνεται το latest[sid]
2. εκπέμπεται SocketIO event telemetry προς το dashboard.

Αυτή η προσέγγιση ξεχωρίζει real-time προβολή από ιστορική αποθήκευση:

- Το UI παίρνει άμεσα την τελευταία τιμή (push)
- Η βάση κρατά το πλήρες ιστορικό για γραφήματα και αναδρομή.

6.5.3 Απομόνωση ανά χρήστη με “rooms”

Για να μη δει χρήστης δεδομένα άλλου χρήστη, το SocketIO χρησιμοποιεί rooms:

- Όταν συνδεθεί ο browser (και ο χρήστης είναι logged-in), μπαίνει στο room user:<account_id>
- Όταν έρθει MQTT μήνυμα, το backend βρίσκει το account_id από το <username> του topic και κάνει emit μόνο στο αντίστοιχο room.

Αυτό είναι σημαντικό γιατί ακόμη κι αν ο server λαμβάνει δεδομένα πολλών χρηστών ταυτόχρονα, το UI του κάθε χρήστη λαμβάνει μόνο ό,τι του ανήκει.

6.6 Αποθήκευση σε MySQL/MariaDB

6.6.1 Σχεσιακή βάση δεδομένων

Η επιλογή MySQL/MariaDB (σχεσιακή) εξυπηρετεί δύο βασικές ανάγκες:

1. Ακεραιότητα και συσχέτιση δεδομένων (accounts → measurements/rules)
2. Αποδοτικά queries για χρονικές σειρές (π.χ. "όλες οι μετρήσεις του τελευταίου 24ώρου").

Επιπλέον, ο σχεδιασμός είναι "κοντά" στη λογική της εφαρμογής: κάθε χρήστης έχει ρυθμίσεις, κανόνες και ιστορικό μετρήσεων.

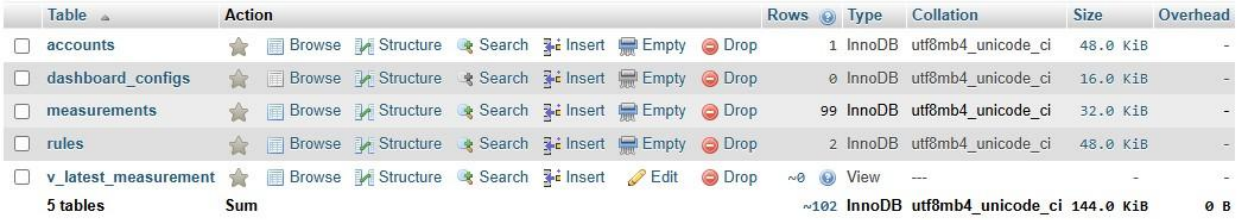


Table	Action	Rows	Type	Collation	Size	Overhead
accounts	Browse Structure Search Insert Empty Drop	1	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
dashboard_configs	Browse Structure Search Insert Empty Drop	0	InnoDB	utf8mb4_unicode_ci	16.0 KiB	-
measurements	Browse Structure Search Insert Empty Drop	99	InnoDB	utf8mb4_unicode_ci	32.0 KiB	-
rules	Browse Structure Search Insert Empty Drop	2	InnoDB	utf8mb4_unicode_ci	48.0 KiB	-
v_latest_measurement	Browse Structure Search Insert Edit Drop	~0	View	---	-	-
5 tables	Sum	~102	InnoDB	utf8mb4_unicode_ci	144.0 KiB	0 B

Εικόνα 27 Βάση δεδομένων (XAMPP)

6.6.2 Accounts: χρήστες + ρυθμίσεις

Ο πίνακας accounts περιέχει:

- στοιχεία login (username + password_hash)
- ρυθμίσεις όπως WiFi SSID/password, email, διάστημα ενημέρωσης
- και παραμέτρους PWM (frequency, duty cycle).

Κρίσιμο σημείο: ο κωδικός δεν αποθηκεύεται ποτέ σε plain text. Γίνεται αποθήκευση hash (bcrypt). Κατά το login, το backend συγκρίνει το password που πληκτρολογεί ο χρήστης με το hash μέσω bcrypt.checkpw().

	account_id	username	password_hash	wifi_ssid	wifi_password	email	update_interval_seconds	pwm_frequency_hz	pwm_duty_cycle	created_at	updated_at
Edit Copy Delete	1	demo_user_1	12345	MyWiFi	MyWiFiPass	demo@example.com	15	1000	50	2026-01-08 20:43:14	2026-01-08 20:43:29
Edit Copy Delete	2	demo_user_2	123123	demo_user_2_SSID	demo_user_2_Pass	demo_user_2@demo.com	30	1000	50	2026-01-24 11:40:00	2026-01-24 11:40:00

Εικόνα 28 Βάση δεδομένων χρηστών

6.6.3 Measurements: "wide table" ανά χρονική σήμανση

Οι μετρήσεις αποθηκεύονται σε πίνακα measurements ως "wide table", δηλαδή μία γραμμή ανά χρονική στιγμή, με πολλές στήλες:

- di1..di4, ext_di1..ext_di4
- pwm_frequency_hz, pwm_duty_cycle
- an1_voltage, an2_voltage
- vrms, irms, power_w, v_instant, i_instant, temperature_c
- timestamps: ts (της μέτρησης) και received_at (χρόνος που το πήρε ο server).

Πλεονέκτημα αυτής της επιλογής:

- Για γραφήματα, είναι απλό: ένα query φέρνει όλες τις στήλες για το χρονικό διάστημα
- Το indexing μπορεί να γίνει αποτελεσματικά σε (account_id, ts).

	measurement_id	account_id	ts	di1	di2	di3	di4	ext_di1	ext_di2	ext_di3	ext_di4	pwm_frequency_hz	pwm_duty_cycle	an1_voltage	an2_voltage	vrms	irms	power_w	v_instant	i_instant	temperature_c	received_at
Edit Copy Delete	1	1	2026-01-08 20:43:15.005707	1	0	1	0	0	0	0	0	1000	45.5	1.234	2.345	230.1	0.85	195.6	228.9	0.83	36.2	2026-01-08 20:43:15.005707
Edit Copy Delete	2	1	2026-01-09 10:37:25.504336	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:37:25.505437
Edit Copy Delete	3	1	2026-01-09 10:37:35.368925	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:37:35.367989
Edit Copy Delete	4	1	2026-01-09 10:37:45.443778	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:37:45.444847
Edit Copy Delete	5	1	2026-01-09 10:37:55.388710	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:37:55.389775
Edit Copy Delete	6	1	2026-01-09 10:38:05.390238	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:38:05.391310
Edit Copy Delete	7	1	2026-01-09 10:38:15.299733	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:38:15.300951
Edit Copy Delete	8	1	2026-01-09 10:38:25.308411	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:38:25.309520
Edit Copy Delete	9	1	2026-01-09 10:38:35.317249	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:38:35.318426
Edit Copy Delete	10	1	2026-01-09 10:38:45.409626	1	1	1	1	0	0	0	0	1000	50	5	3	0	0	0	0	0	50	2026-01-09 09:38:45.410707

Εικόνα 29 Βάση δεδομένων μετρήσεων-τηλεμετρία

6.6.4 Χρονική σήμανση: ts vs received_at

Το backend διατηρεί δύο χρόνους:

- ts: ο χρόνος του πακέτου (όταν υπάρχει αξιόπιστος από τη συσκευή)
- received_at: πότε γράφτηκε στη βάση (server time).

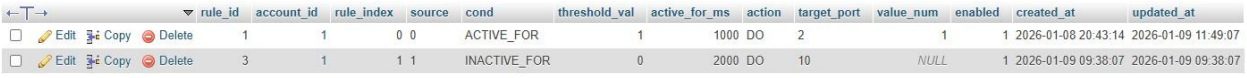
Αυτό είναι χρήσιμο σε περιπτώσεις LTE καθυστέρησης/απώλειας αν κάποιο πακέτο "καθυστερήσει", μπορείς να δεις πότε πραγματικά μετρήθηκε και πότε έφτασε.

6.6.5 Τελευταία μέτρηση

Υπάρχει και view v_latest_measurement, το οποίο είναι πρακτικό για widgets που θέλουν την πιο πρόσφατη εγγραφή χωρίς να κάνουν περίπλοκα queries. Έτσι το frontend μπορεί εύκολα να αντλεί "τρέχουσα κατάσταση".

6.6.6 Rules: αποθήκευση κανόνων αυτοματισμού

Ο πίνακας rules χρησιμοποιείται για την αποθήκευση των κανόνων αυτοματισμού ανά χρήστη/συσκευή. Η επιλογή να αποθηκεύονται οι κανόνες στη βάση (και όχι μόνο προσωρινά στη μνήμη του backend) εξυπηρετεί δύο στόχους: (α) οι κανόνες να παραμένουν διαθέσιμοι μετά από restart του server ή του browser και (β) η πλατφόρμα να μπορεί να ανακτήσει την "τελευταία γνωστή" διαμόρφωση κανόνων οποιαδήποτε στιγμή.



	rule_id	account_id	rule_index	source	cond	threshold_val	active_for_ms	action	target_port	value_num	enabled	created_at	updated_at
Edit Copy Delete	1	1	0	0	ACTIVE_FOR	1	1000	DO	2	1	1	2026-01-08 20:43:14	2026-01-09 11:49:07
Edit Copy Delete	3	1	1	1	INACTIVE_FOR	0	2000	DO	10	NULL	1	2026-01-09 09:38:07	2026-01-09 09:38:07

Εικόνα 30 Βάση δεδομένων για τους κανόνες

Συσχέτιση ανά χρήστη

Κάθε εγγραφή κανόνα συνδέεται με λογαριασμό μέσω του account_id, με foreign key προς τον πίνακα accounts. Με αυτόν τον τρόπο:

- οι κανόνες είναι απομονωμένοι ανά χρήστη
- σε διαγραφή λογαριασμού μπορεί να διαγραφεί αυτόματα και το σύνολο των κανόνων του (cascade).

Σταθερή αντιστοίχιση θέσεων (rule_index 0..15)

Οι κανόνες αποθηκεύονται με σταθερό index (rule_index), το οποίο λειτουργεί ως θέση του κανόνα. Το rule_index περιορίζεται στο εύρος 0..15, ώστε κάθε χρήστης να έχει μέχρι 16 κανόνες. Η προσέγγιση έχει γίνει γιατί και το firmware που χρησιμοποιούν σταθερού μεγέθους πίνακα/δομές, άρα η αντιστοίχιση από DB → συσκευή γίνεται με προβλέψιμο τρόπο.

Για να διασφαλιστεί η μοναδικότητα των κανόνων ανά χρήστη, χρησιμοποιείται unique constraint στην βάση δεδομένων στο ζεύγος account_id και rule_index.

Πεδία κανόνα και αντιστοίχιση σε λογική "if–then"

Κάθε κανόνας αποθηκεύεται με πεδία που αντιστοιχούν απευθείας στην έννοια "συνθήκη → ενέργεια":

- **source:** η πηγή/μέγεθος που παρακολουθείται (π.χ. ψηφιακή είσοδος, αναλογική τιμή, μετρούμενο μέγεθος)
- **cond:** ο τύπος συνθήκης (π.χ. "μεγαλύτερο από", "μικρότερο από", "ενεργό για Χ χρόνο")
- **threshold_val:** τιμή κατωφλίου/σύγκρισης
- **active_for_ms:** χρόνος ενεργοποίησης σε ms (χρησιμοποιείται σε συνθήκες τύπου "ACTIVE_FOR" / "INACTIVE_FOR")
- **action:** τύπος ενέργειας (π.χ. ενεργοποίηση εξόδου/ρελέ, αλλαγή PWM)
- **target_port:** ποια έξοδος/στόχος επηρεάζεται από την ενέργεια
- **value_num:** αριθμητική παράμετρος ενέργειας (π.χ. 0/1 για on/off, duty cycle για PWM κ.λπ.)
- **enabled:** ενεργοποίηση/απενεργοποίηση κανόνα χωρίς διαγραφή του.

Υπάρχουν επίσης timestamps created_at και updated_at, ώστε να είναι δυνατή η παρακολούθηση αλλαγών στο χρόνο και να υπάρχει σαφές "πότε ενημερώθηκε" ένας κανόνας.

Χρήση upsert αντί για απλό insert

Σε επίπεδο λειτουργίας, η web πλατφόρμα επιτρέπει επεξεργασία του πλήρους σετ κανόνων. Για τον λόγο αυτό, στο backend οι ενημερώσεις γίνονται με λογική upsert (εισαγωγή ή ενημέρωση στην ίδια εντολή), χρησιμοποιώντας ως κλειδί το (account_id, rule_index). Έτσι:

- όταν ο χρήστης αλλάζει έναν κανόνα σε συγκεκριμένη θέση, γίνεται update στην ίδια θέση
- όταν ο χρήστης προσθέτει νέο κανόνα, γίνεται insert
- και όταν αφαιρεί κανόνα, το αντίστοιχο slot μπορεί να διαγραφεί ή να απενεργοποιηθεί (enabled=0), ανάλογα με την επιθυμητή πολιτική.

Ρόλος του πίνακα rules στη ροή downlink

Μετά από κάθε αλλαγή του χρήστη:

- ενημερώνεται ο πίνακας rules
- το backend ανακτά τους κανόνες του χρήστη ταξινομημένους ανά rule_index
- τους μετατρέπει σε δομημένη μορφή εντολής
- τους αποστέλλει στη συσκευή μέσω MQTT (π.χ. ως πακέτο τύπου set_rules), ώστε η συσκευή να συγχρονιστεί με την τελευταία κατάσταση της βάσης.

Στην Ενότητα 6.8 παρουσιάζεται αναλυτικά πώς οι κανόνες πακετάρονται σε μήνυμα και αποστέλλονται μέσω MQTT προς τη συσκευή.

6.7 Downlink: αποστολή ρυθμίσεων και κανόνων στη συσκευή

Η πλατφόρμα δεν είναι μόνο "παθητική" (λήψη μετρήσεων), αλλά και "ενεργή": ο χρήστης μπορεί να αλλάξει ρυθμίσεις ή να ορίσει κανόνες, και το backend να τα στείλει στη συσκευή.

6.7.1 Μηχανισμός εντολών (commands) σε JSON

Οι εντολές προς τη συσκευή στέλνονται ως JSON με βασικό πεδίο cmd. Ενδεικτικά:

- {"cmd":"set_wifi", "ssid":"...", "pass":"..."}
- {"cmd":"set_interval", "seconds": 15}
- {"cmd":"set_pwm", "pwm_frequency_hz":1000, "pwm_duty_cycle":50, ...}

Στο PWM command, στέλνονται και "εναλλακτικά" ονόματα πεδίων (π.χ. freq_hz, duty) ώστε να υπάρχει συμβατότητα με διαφορετικές εκδόσεις parser στο firmware. Αυτό είναι πρακτικό όταν εξελίσσεται το πρωτόκολλο και δεν θες να "σπάσουν" παλιές εκδόσεις.

6.7.2 Dual publish: αποστολή σε WiFi και LTE ταυτόχρονα

Για να αυξηθεί η πιθανότητα να φτάσει η εντολή, το backend κάνει publish:

- στο WiFi downlink topic και
- στο LTE downlink topic

για τον ίδιο χρήστη.

Αυτό σημαίνει ότι ακόμη κι αν η συσκευή εκείνη τη στιγμή "ακούει" μόνο τον έναν broker (π.χ. βρίσκεται σε LTE mode), η εντολή μπορεί να παραληφθεί.

6.7.3 Κανόνες (rules): αποθήκευση και μετατροπή σε STM32 format

Οι κανόνες αποθηκεύονται στον πίνακα rules με "slots" rule_index (0..15). Αυτό ταιριάζει με embedded λογική: στο firmware είναι πιο εύκολο να έχεις ένα fixed array 16 δομών παρά μία δυναμική λίστα.

Όταν ο χρήστης αποθηκεύσει κανόνες:

1. γίνεται ενημέρωση στη βάση (insert/update ανά rule_index)
2. το backend "ξαναδιαβάζει" τους κανόνες και τους στέλνει στη συσκευή σε σταθερό format

Παράδειγμα δομής που παράγεται από το backend:

- idx: θέση κανόνα (0..15)
- source_id: ID πηγής (π.χ. DI1)
- condition: τύπος συνθήκης (GT/LT/EQ/ACTIVE_FOR/INACTIVE_FOR)
- threshold: αριθμητικό όριο (όταν υπάρχει)
- duration_ms: χρόνος για ACTIVE_FOR/INACTIVE_FOR
- action: τύπος ενέργειας και στόχος (π.χ. relay/output/PWM)

Στη συνέχεια, το backend στέλνει {"cmd":"set_rules","rules":[...]} στο downlink topic.

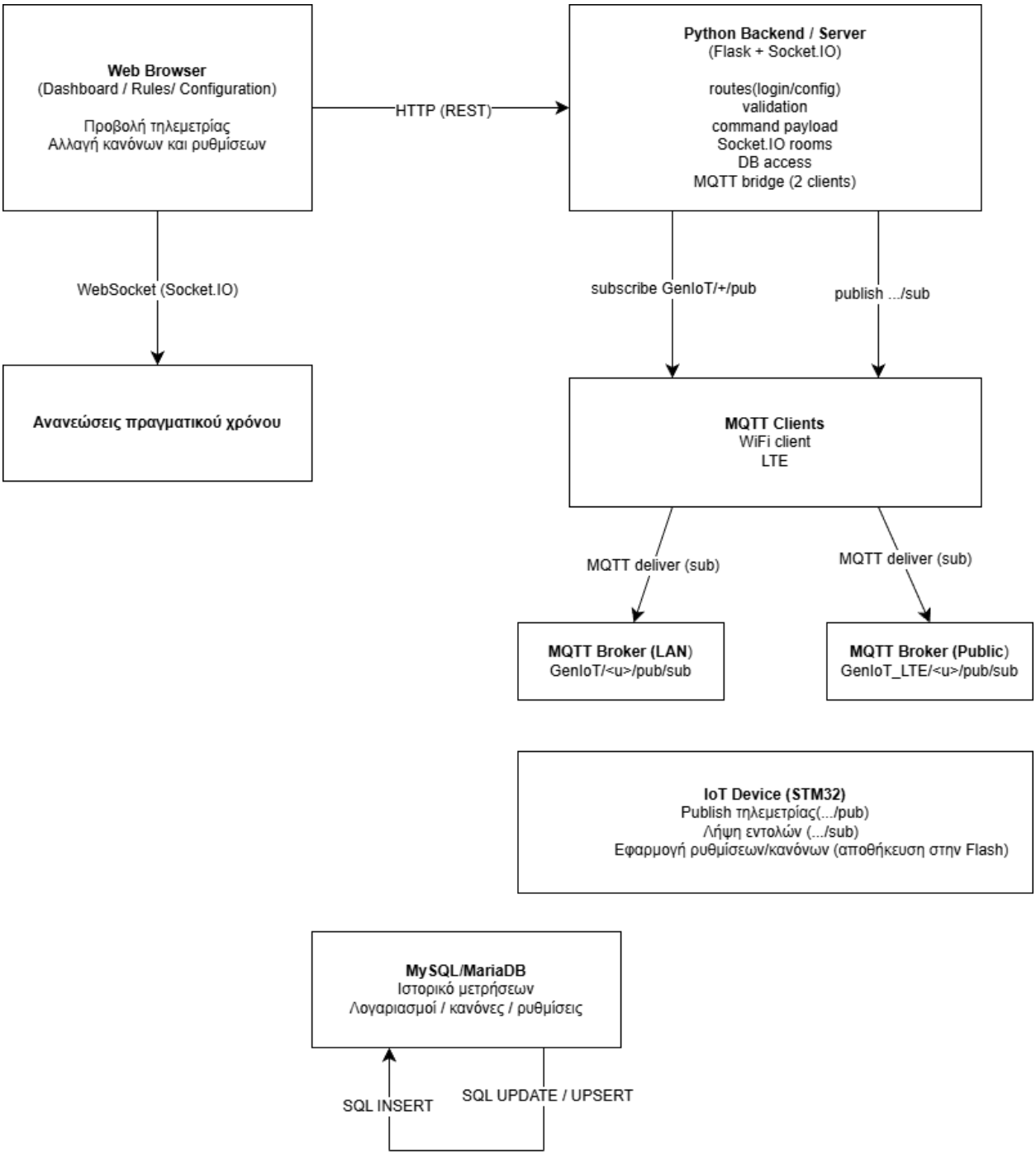
6.7.4 Backend και ανακατασκευή κανόνων

Το backend ανακατασκευάζει τους κανόνες αντί να στέλνει απευθείας ό,τι έστειλε το UI γιατί:

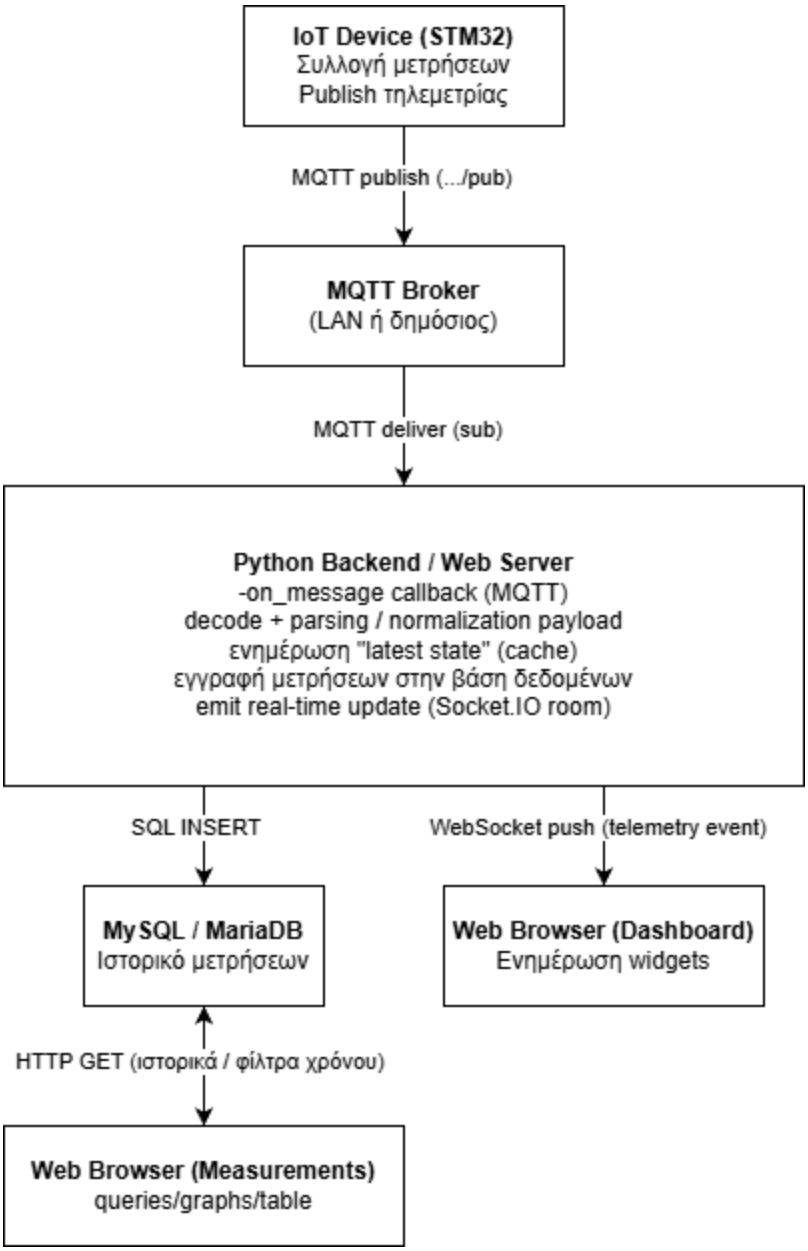
- Το UI μπορεί να στείλει τιμές με διαφορετικές μορφές (π.χ. “DI1”, “Digital input 1”, “id=0”)
- Το backend αναλαμβάνει να τις ενοποιήσει και να παράγει ένα format που είναι ακριβώς αυτό που περιμένει ο parser της συσκευής.

Έτσι, ο κανόνας εφαρμόζεται με συνεπή τρόπο, ανεξάρτητα από το πώς τον εξέφρασε ο χρήστης στο interface.

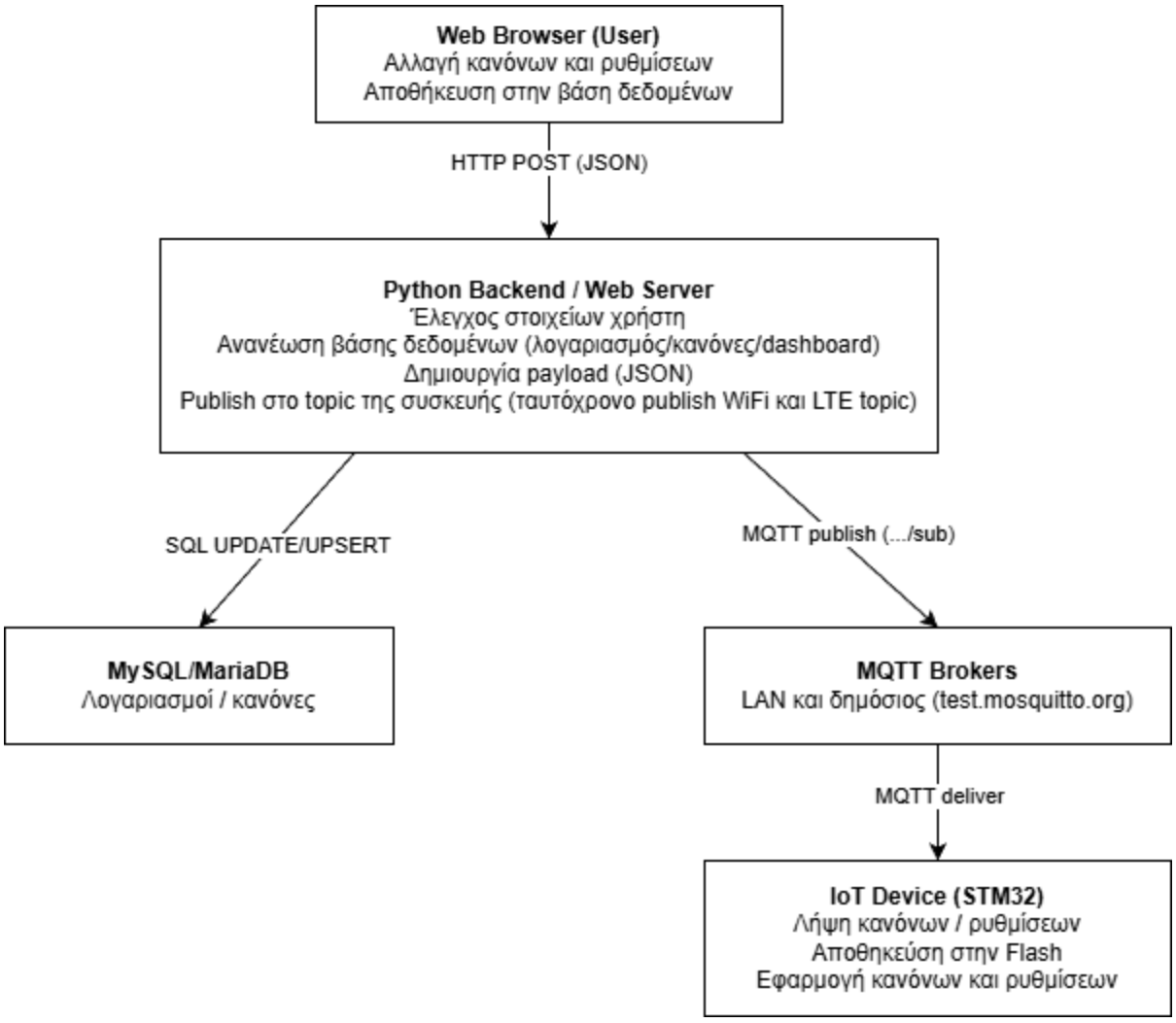
6.8 Block διαγράμματα



Εικόνα 31 Συνολικό block διάγραμμα (Backend – Brokers – Device)



Εικόνα 32 Ροή τηλεμετρίας (uplink _ data-plane)



Εικόνα 33 Ροή ελέγχου (downlink _ control-plane)

6.9 Σημεία αξιοπιστίας και πρακτικές επιλογές υλοποίησης

1. **Ανθεκτικότητα σε αποκλίσεις format (LTE):** ο μηχανισμός normalization μειώνει τα "χαμένα" πακέτα λόγω parsing
2. **Διαχωρισμός real-time vs ιστορικού:** latest cache + DB insert επιτρέπουν γρήγορο UI και πλήρες ιστορικό
3. **Απομόνωση ανά χρήστη:** topic namespaces + room-based SocketIO εξασφαλίζουν ότι δεν αναμειγνύονται δεδομένα
4. **Dual publish:** αυξάνει πιθανότητα παραλαβής εντολών όταν δεν είναι γνωστό αν η συσκευή είναι σε WiFi ή LTE mode
5. **Σχεσιακή βάση και indexes:** το (account_id, ts) επιταχύνει queries σε χρονικές σειρές και κάνει εύκολη τη δημιουργία γραφημάτων.

7. Web πλατφόρμα και διεπαφή χρήστη

Η web πλατφόρμα αποτελεί το επίπεδο της διεπαφής χρήστη (user interface) του συστήματος και έχει ως στόχο να επιτρέπει εποπτεία και έλεγχο της IoT συσκευής με τρόπο κατανοητό, χωρίς να απαιτείται γνώση του firmware ή του hardware. Σε πρακτικό επίπεδο, ο χρήστης αλληλεπιδρά με την πλατφόρμα για τρεις βασικές εργασίες:

1. προβολή της τρέχουσας κατάστασης μέσω dashboard
2. πρόσβαση σε ιστορικά δεδομένα με δυνατότητα φιλτραρίσματος χρόνου
3. αποστολή παραμέτρων/κανόνων λειτουργίας προς τη συσκευή.

Η σχεδίαση ακολουθεί την αρχή "μία σελίδα – μία λειτουργία", ώστε η πλοήγηση να είναι ξεκάθαρη: Dashboard για εποπτεία, Measurements για ιστορικά, Rules για κανόνες αυτοματισμού και Configuration για ρυθμίσεις λογαριασμού/συσκευής. Παράλληλα, η πλατφόρμα υποστηρίζει πολυχρηστικότητα, με απομόνωση δεδομένων και ρυθμίσεων ανά λογαριασμό

7.1 Στόχος και θέση της πλατφόρμας στην αρχιτεκτονική

Η web πλατφόρμα δεν επικοινωνεί απευθείας με τη συσκευή. Βασίζεται σε ένα backend (server σε Python) που λειτουργεί ως ενδιάμεσο:

- από τη μία πλευρά παραλαμβάνει τηλεμετρία και την αποθηκεύει στη βάση
- από την άλλη δέχεται ενέργειες του χρήστη (π.χ. αλλαγή κανόνων) και παράγει εντολές προς τη συσκευή.

Έτσι, ο browser παραμένει "ελαφρύς" (εμφάνιση και χειρισμός UI), ενώ οι υπόλοιπες λειτουργίες (αποθήκευση, validation, δρομολόγηση εντολών) υλοποιούνται στον server.

7.2 Τεχνολογική στοίβα και δομή εφαρμογής (server/client)

7.2.1 Server layer

Η εφαρμογή υλοποιείται σε Python με χρήση του Flask ως web framework. Ο Flask server εξυπηρετεί:

- σελίδες (HTML) μέσω templates
- REST endpoints (JSON responses) για ανάκτηση/αποθήκευση δεδομένων
- μηχανισμό real-time ενημέρωσης μέσω Flask-SocketIO όπου χρειάζεται.

Η διαχείριση σύνδεσης χρήστη γίνεται με sessions (cookie-based) και βασίζεται στο SECRET_KEY του Flask. Μετά από επιτυχή login, αποθηκεύεται στο session το account_id και το username, ώστε κάθε επόμενη κλήση να γνωρίζει σε ποιον λογαριασμό αντιστοιχεί.

7.2.2 Client layer

Το frontend αποτελείται από:

- **Jinja2 templates** (π.χ. base.html, dashboard.html, measurements.html, rules.html, config.html) ώστε όλες οι σελίδες να μοιράζονται κοινή εμφάνιση (navigation bar, layout)
- **JavaScript** για δυναμικές λειτουργίες: φόρτωση δεδομένων μέσω fetch, αποθήκευση επιλογών του χρήστη, επικοινωνία με API endpoints και περιοδική ανανέωση (polling)
- **CSS** για βασική αισθητική/διάταξη.

Η ύπαρξη κοινού base.html λειτουργεί ως "σκελετός" σελίδας. Όλες οι επιμέρους σελίδες κληρονομούν την ίδια μπάρα πλοήγησης και το ίδιο περιβάλλον, μειώνοντας την επανάληψη και διευκολύνοντας τη συντήρηση

7.2.3 Διασύνδεση με βάση δεδομένων

Για μόνιμη αποθήκευση χρησιμοποιείται MySQL/MariaDB, με σύνδεση μέσω mysql-connector-python. Η βάση οργανώνει:

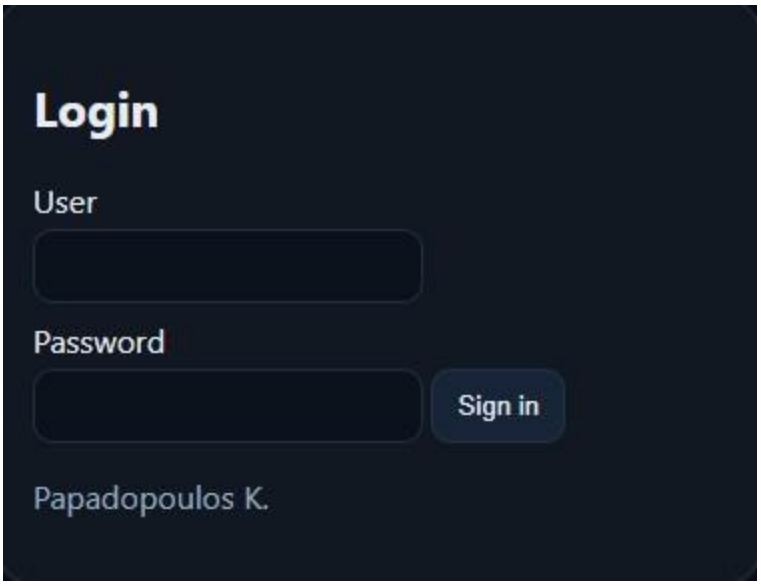
- λογαριασμούς χρηστών και ρυθμίσεις (accounts)
- ιστορικές μετρήσεις (measurements)
- κανόνες αυτοματισμού (rules)
- αποθήκευση dashboard διαμόρφωσης ανά χρήστη σε μορφή JSON (dashboard_configs).

Αυτή η δομή επιτρέπει να αποθηκεύονται τόσο "χρονικά" δεδομένα (μετρήσεις), όσο και "config" δεδομένα (κανόνες, UI ρυθμίσεις), με σαφή συσχέτιση στο account_id.

7.3 Αυθεντικοποίηση και απομόνωση δεδομένων ανά χρήστη

7.3.1 Login flow και sessions

Η πρόσβαση στην πλατφόρμα ξεκινά από τη σελίδα login. Με την υποβολή credentials, ο server αναζητά τον χρήστη στον πίνακα accounts και, αν η επαλήθευση είναι επιτυχής, δημιουργεί session με τα βασικά αναγνωριστικά. Από εκεί και πέρα, οι σελίδες της πλατφόρμας κάνουν έλεγχο "require_login", ώστε να αποκλείεται πρόσβαση χωρίς ενεργό session.



Εικόνα 34 Σελίδα Login

7.3.2 Αποθήκευση κωδικών (bcrypt) και πολιτική πρόσβασης

Οι κωδικοί πρόσβασης αποθηκεύονται σε hashed μορφή με bcrypt (όπου υπάρχει bcrypt hash). Κατά το login γίνεται έλεγχος με bcrypt.checkpw. Στη σελίδα configuration υπάρχει δυνατότητα αλλαγής password: αν ο χρήστης δώσει νέο password, αποθηκεύεται ως bcrypt hash στη βάση.

Το username διατηρείται σταθερό, επειδή χρησιμοποιείται και ως αναγνωριστικό στο επίπεδο επικοινωνίας (π.χ. topics ανά χρήστη). Με αυτόν τον περιορισμό διασφαλίζεται ότι η δρομολόγηση δεδομένων παραμένει σταθερή, ο χρήστης βλέπει μόνο τα δεδομένα που ανήκουν στον λογαριασμό του και οι εντολές που στέλνει αντιστοιχίζονται στη "δική του" συσκευή/κανάλι.

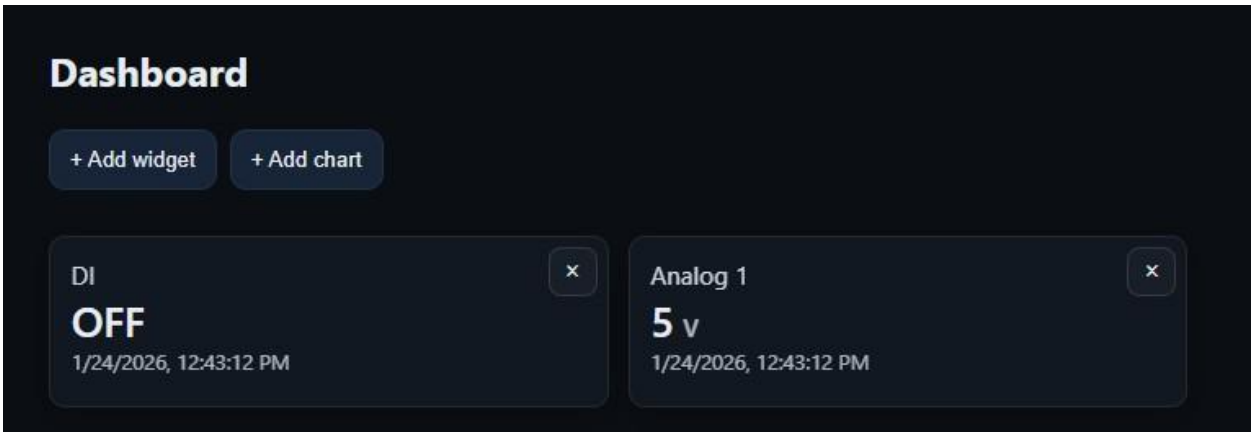
7.4 Σελίδα Dashboard: εποπτεία και widgets

7.4.1 Έννοια widgets και signal IDs

Το dashboard υλοποιεί την έννοια ενός δυναμικού "πίνακα οργάνων" όπου ο χρήστης επιλέγει τι θέλει να βλέπει. Κάθε widget αντιστοιχεί σε ένα signal id (π.χ. Vrms, Irms, θερμοκρασία, ψηφιακές είσοδοι κ.λπ.). Στο UI ο χρήστης επιλέγει:

- τίτλο widget
- σήμα (signal id)
- προαιρετικά μονάδα μέτρησης(π.χ. mV)
- και προαιρετικά έναν απλό μετασχηματισμό τιμής (formula), ώστε να αλλάζει μόνο η απεικόνιση και όχι τα αποθηκευμένα δεδομένα.

Σημαντικό στοιχείο είναι ότι οι ψηφιακές είσοδοι αντιμετωπίζονται ως καταστάσεις (ON/OFF), ενώ τα αναλογικά/μετρούμενα μεγέθη εμφανίζονται ως αριθμητικές τιμές.



Εικόνα 35 Σελίδα Dashboard

7.4.2 Αποθήκευση διαμόρφωσης dashboard στη βάση (JSON)

Η διαμόρφωση του dashboard (δηλαδή ποια widgets υπάρχουν και πώς είναι στημένα) αποθηκεύεται στον πίνακα dashboard_configs ως JSON, συσχετισμένη με το account_id. Η πλατφόρμα παρέχει REST endpoints:

- GET /api/dashboard για ανάκτηση του config
- POST /api/dashboard για αποθήκευση/ενημέρωση.

Η χρήση JSON εδώ είναι πρακτική γιατί επιτρέπει ευέλικτη αποθήκευση "λίστας widgets" χωρίς να απαιτείται ξεχωριστός πίνακας ανά πεδίο. Έτσι, το UI μπορεί να επεκταθεί (π.χ. νέα πεδία widget) χωρίς αλλαγές στη δομή της βάσης.

7.4.3 Τρέχουσα κατάσταση και ενημέρωση δεδομένων

Το dashboard εμφανίζει την πιο πρόσφατη διαθέσιμη κατάσταση/μέτρηση για κάθε σήμα. Στο σύστημα χρησιμοποιούνται δύο συμπληρωματικοί μηχανισμοί ενημέρωσης:

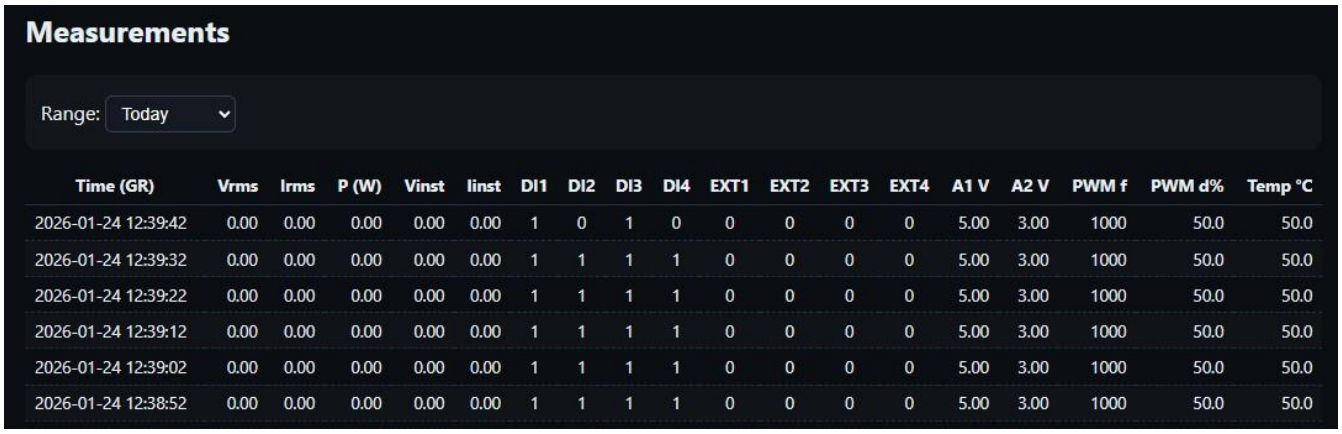
- **pull (HTTP polling)** για άντληση της τελευταίας μέτρησης/σειράς από REST endpoint, όταν ο στόχος είναι να ενημερωθεί το UI με βάση τα δεδομένα της βάσης
- **push (Socket.IO)** όταν επιδιώκεται real-time προώθηση νέων τιμών προς τον browser.

Για το real-time, εφαρμόζεται επιπλέον απομόνωση με "rooms" ανά χρήστη: κάθε websocket σύνδεση εντάσσεται σε room της μορφής user:<account_id>, ώστε οι ενημερώσεις να στέλνονται μόνο στον σωστό λογαριασμό

7.5 Σελίδα Measurements: ιστορικά και φίλτρα χρόνου

7.5.1 Προβολή πίνακα (server-rendered)

Η σελίδα Measurements εμφανίζει πίνακα ιστορικών μετρήσεων με στόχο την αναλυτική απεικόνιση. Ο χρήστης μπορεί να επιλέξει χρονικό εύρος (π.χ. σήμερα, χθες, εβδομάδα ή συγκεκριμένη περίοδο). Ο server εκτελεί ερώτημα στη βάση, φιλτράροντας με βάση το account_id και το χρονικό διάστημα, και επιστρέφει αποτελέσματα με ταξινόμηση από τα νεότερα προς τα παλαιότερα και όριο πλήθους εγγραφών για να παραμένει γρήγορη η σελίδα.



Time (GR)	Vrms	Irms	P (W)	Vinst	Iinst	DI1	DI2	DI3	DI4	EXT1	EXT2	EXT3	EXT4	A1 V	A2 V	PWM f	PWM d%	Temp °C
2026-01-24 12:39:42	0.00	0.00	0.00	0.00	0.00	1	0	1	0	0	0	0	0	5.00	3.00	1000	50.0	50.0
2026-01-24 12:39:32	0.00	0.00	0.00	0.00	0.00	1	1	1	1	0	0	0	0	5.00	3.00	1000	50.0	50.0
2026-01-24 12:39:22	0.00	0.00	0.00	0.00	0.00	1	1	1	1	0	0	0	0	5.00	3.00	1000	50.0	50.0
2026-01-24 12:39:12	0.00	0.00	0.00	0.00	0.00	1	1	1	1	0	0	0	0	5.00	3.00	1000	50.0	50.0
2026-01-24 12:39:02	0.00	0.00	0.00	0.00	0.00	1	1	1	1	0	0	0	0	5.00	3.00	1000	50.0	50.0
2026-01-24 12:38:52	0.00	0.00	0.00	0.00	0.00	1	1	1	1	0	0	0	0	5.00	3.00	1000	50.0	50.0

Εικόνα 36 Σελίδα προβολής μετρήσεων

7.5.2 JSON API για δεδομένα γραφημάτων

Παράλληλα, υπάρχει endpoint GET /api/measurements που επιστρέφει JSON μορφή των μετρήσεων, κατάλληλη για χρήση από JavaScript (π.χ. για γραφήματα ή εξαγωγή). Το API επιστρέφει για κάθε εγγραφή:

- timestamp
- ψηφιακές εισόδους ως λίστα
- αναλογικές τάσεις
- μεγέθη ισχύος (Vrms/Irms/P κ.λπ.),
- θερμοκρασία.

Η ύπαρξη ξεχωριστού JSON endpoint είναι σημαντική γιατί αποσυνδέει το UI από τον τρόπο που προβάλλεται το HTML. Οι ίδιες μετρήσεις μπορούν να χρησιμοποιηθούν σε διαφορετικές απεικονίσεις χωρίς να αλλάζει ο server.

7.5.3 Auto refresh όταν έρθει νέα μέτρηση

Για να ανανεώνεται η σελίδα όταν υπάρχει νεότερη εγγραφή, χρησιμοποιείται έλεγχος μέσω endpoint GET /api/measurements/latest_ts. Ο browser ζητά περιοδικά (ανά λίγα δευτερόλεπτα) το πιο πρόσφατο received_at της βάσης για τον συγκεκριμένο χρήστη. Αν αλλάξει, γίνεται ανανεώση της σελίδας, ώστε ο χρήστης να βλέπει άμεσα τις νεότερες τιμές χωρίς να πατά χειροκίνητα ανανεώση.

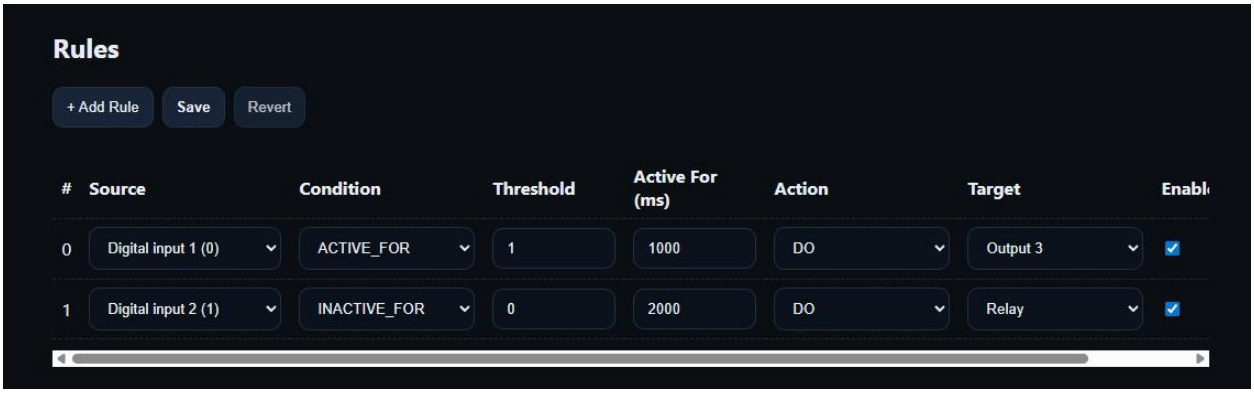
7.6 Σελίδα Rules: ορισμός κανόνων και αποστολή στη συσκευή

7.6.1 Μοντέλο κανόνα (condition/action)

Η διεπαφή Rules επιτρέπει ορισμό κανόνων τύπου “αν–τότε” με πεδία:

- πηγή (source) / σήμα που παρακολουθείται,
- συνθήκη (π.χ. GT, LT, EQ, ACTIVE_FOR, INACTIVE_FOR)
- threshold και διάρκεια (όπου χρειάζεται)
- ενέργεια (π.χ. DO, RELAY, PWM) και στόχος (target port).

Σκοπός του UI είναι ο χρήστης να ορίζει κανόνες με δομημένο τρόπο, περιορίζοντας λάθη (π.χ. μη επιτρεπτές τιμές condition/action). Οι κανονές έχουν αναλυθεί στο κεφάλαιο 4.4.



Εικόνα 37 Σελίδα κανόνων

7.6.2 Validation, αποθήκευση και όριο πλήθους κανόνων

Στο server υπάρχει endpoint GET /api/rules για ανάκτηση κανόνων και POST /api/rules για αποθήκευση. Κατά την αποθήκευση, ο server:

- ελέγχει ότι οι συνθήκες και οι ενέργειες ανήκουν σε επιτρεπτό σύνολο
- μετατρέπει αριθμητικά πεδία σε σωστό τύπο
- εφαρμόζει “σταθερές θέσεις” κανόνων με rule_index (π.χ. 0..15), ώστε το σύστημα να έχει προβλέψιμο mapping και να μην ξεπερνά τα 16 slots
- ενημερώνει/εισάγει εγγραφές με λογική upsert (ON DUPLICATE KEY UPDATE)
- αφαιρεί κανόνες που δεν υπάρχουν πλέον στο νέο σετ.

Με αυτόν τον τρόπο η βάση διατηρεί πάντα ένα “καθαρό” σύνολο ενεργών κανόνων ανά χρήστη.

7.6.3 Μετατροπή στο format του firmware και publish προς συσκευή

Μετά την αποθήκευση, το backend δεν κρατά τους κανόνες μόνο στη βάση, αλλά τους μετατρέπει σε συγκεκριμένο format για να μπορεί να διαβαστεί από το firmware και τους αποστέλλει μέσω MQTT. Η αποστολή γίνεται ως ένα ενιαίο πακέτο που περιλαμβάνει ολόκληρη τη λίστα κανόνων, ώστε η συσκευή να συγχρονίζεται πλήρως με την τελευταία κατάσταση της βάσης.

Επιπλέον, η δρομολόγηση προς τη συσκευή ακολουθεί per-user ονοματοδοσία, ώστε οι κανόνες ενός χρήστη να σταλούν μόνο στην συγκεκριμένη συσκευή.

7.7 Σελίδα Configuration: ρυθμίσεις και downlink παραμετροποίηση

7.7.1 Ρυθμίσεις λογαριασμού στη βάση

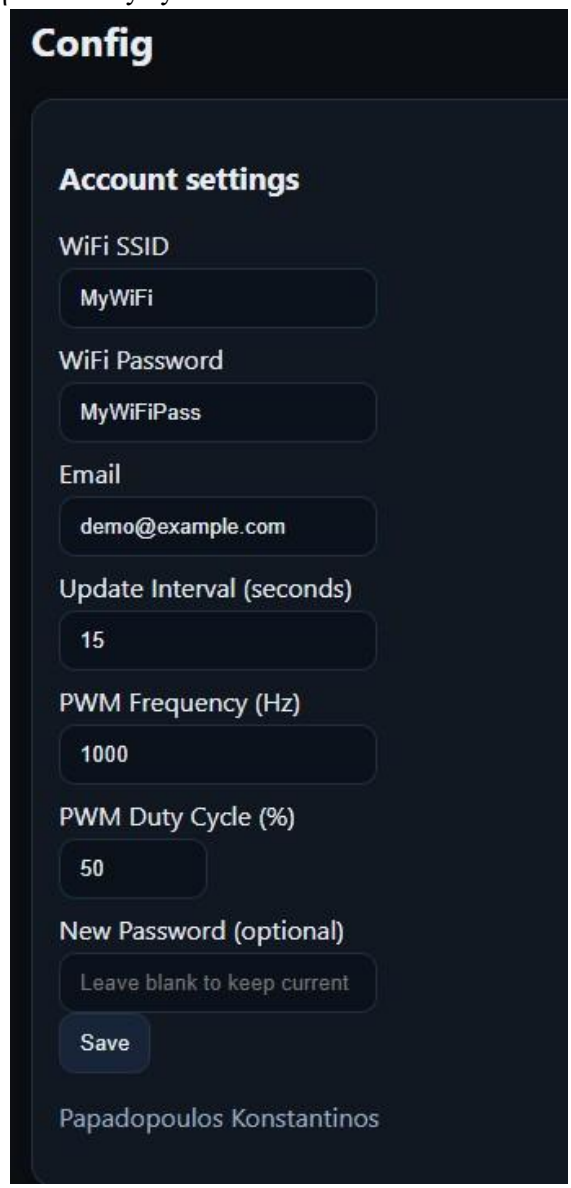
Η σελίδα Configuration λειτουργεί ως κεντρικό σημείο αλλαγής των παραμέτρων λειτουργίας. Σε επίπεδο βάσης, οι ρυθμίσεις αποθηκεύονται στον πίνακα accounts (π.χ. WiFi SSID/password, email, update

interval, PWM frequency και duty cycle). Όταν ο χρήστης πατήσει “Save”, γίνεται POST JSON προς τον server, ο οποίος ελέγχει βασικούς περιορισμούς (π.χ. duty 0–100%, συχνότητα > 0) πριν γράψει στη βάση.

7.7.2 Παραγωγή και αποστολή εντολών προς τη συσκευή

Μετά την επιτυχή αποθήκευση, το backend δημιουργεί εντολές ελέγχου σε μορφή JSON και τις αποστέλλει προς τη συσκευή μέσω MQTT (downlink). Ενδεικτικά χρησιμοποιούνται:

- set_wifi με πεδία ssid και pass
- set_interval με ms
- set_pwm με συχνότητα και duty cycle.



Εικόνα 38 Σελίδα ρυθμίσεων

Σχεδιαστικά, αυτή η επιλογή επιτρέπει στον χρήστη να αλλάζει κρίσιμες παραμέτρους λειτουργίας της συσκευής απομακρυσμένα, χωρίς να απαιτείται επαναπρογραμματισμός ή φυσική πρόσβαση.

7.8 Συμπεράσματα κεφαλαίου

Η διαδικτυακή πλατφόρμα συγκεντρώνει τις βασικές λειτουργίες εποπτείας και ελέγχου σε ενιαία διεπαφή: real-time/τρέχουσα κατάσταση μέσω dashboard, ιστορικά δεδομένα μέσω σελίδας Measurements και απομακρυσμένη παραμετροποίηση μέσω Rules και Configuration. Η αρχιτεκτονική με σαφή διάκριση server/client, η αποθήκευση στη βάση με συσχέτιση ανά χρήστη και η χρήση structured endpoints επιτρέπουν κλιμάκωση, συντήρηση και προσθήκη νέων λειτουργιών χωρίς ανασχεδιασμό της πλατφόρμας.

8. Παράδειγμα λειτουργίας

8.1 Σενάριο εφαρμογής

Για την επίδειξη της λειτουργίας της πλατφόρμας επιλέγεται ένα σενάριο που συνδυάζει:

- παραμετροποίηση της συσκευής από την ιστοσελίδα
- δημιουργία νέων κανόνων
- προβολή μετρήσεων στην ιστοσελίδα
- δημιουργία widget με τροποποίηση της μέτρησης πριν την προβολή.
- χρήση δικτύου LTE με κάρτα SIM της Vodafone

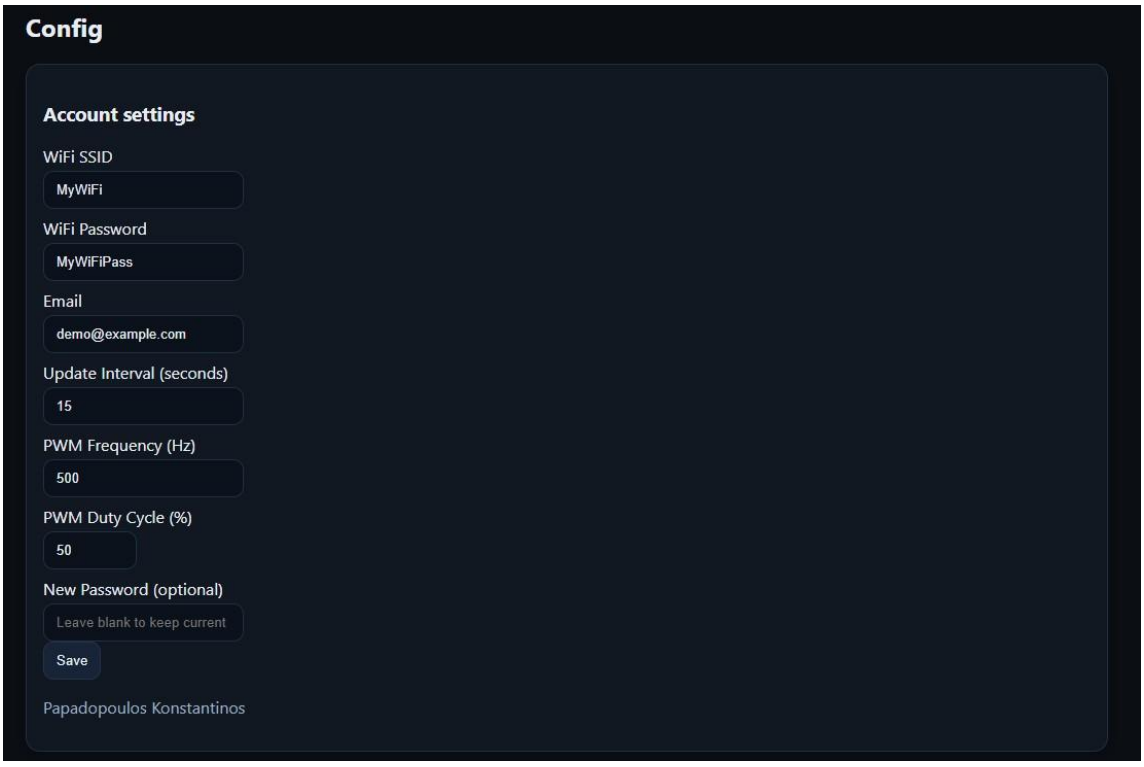
Ο κανόνες που θα δημιουργηθούν στην πλατφόρμα, θα αποσταλούν στην συσκευή και θα εφαρμόζονται από αυτήν είναι:

1. Αν η ψηφιακή είσοδο 2 είναι ενεργή για πάνω από 2 δευτερόλεπτα τότε να ενεργοποιηθεί η ψηφιακής έξοδος 2
2. Αν το Irms είναι μεγαλύτερο από 1A τότε να ενεργοποιηθεί η έξοδος pwm της οποίας έχει ρυθμιστεί πρώτα η συχνότητα και το duty cycle στα 500Hz και 50% duty cycle.

Η έξοδος του pwm θα συνδεθεί στην είσοδο (input capture) με σκοπό να φανεί και στις μετρήσεις.

8.2 Παραμετροποίηση συσκευής

Αρχικά θέτουμε στην σελίδα Config θέτουμε την συχνότητα στα 500Hz και το duty cycle στο 50%. Αμέσως μετά πατάμε save ώστε οι νέες τιμές να σταλούν στην συσκευή και να σωθούν στην βάση δεδομένων.



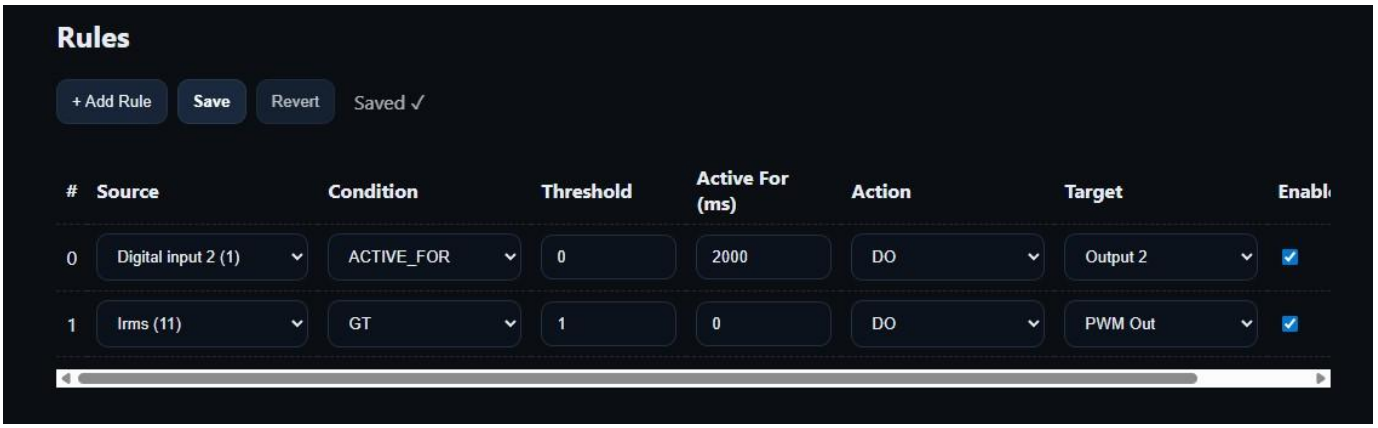
Εικόνα 39 Αλλαγή συχνότητα και duty cycle

Οι ρυθμίσεις αποστέλλονται στην συσκευή IoT και αποθηκεύονται στην Flash.

```
C:\Programs\mosquitto>mosquitto_sub -h test.mosquitto.org -t "GenIoT_LTE/demo_user_1/sub" -v
GenIoT_LTE/demo_user_1/sub {"cmd": "set_wifi", "ssid": "MyWiFi", "pass": "MyWiFiPass"}
GenIoT_LTE/demo_user_1/sub {"cmd": "set_interval", "ms": 15000}
GenIoT_LTE/demo_user_1/sub {"cmd": "set_pwm", "pwm_frequency_hz": 500, "pwm_duty_cycle": 50, "freq_hz": 500, "duty": 50}
```

Εικόνα 40 Αποστολή ρυθμίσεων μέσω LTE

Έπειτα δημιουργούμε τους δύο κανόνες όπως αυτοί αναφέρθηκαν στο προηγουμένως. Αφού τελειώσουμε με την δημιουργία των κανόνων πατάμε save ώστε αυτοί να αποσταλούν στην συσκευή IoT, να αποθηκευτούν στην βάση δεδομένων μέσω του backend.



Εικόνα 41 Εισαγωγή κανόνων στην ιστοσελίδα

Οι κανόνες αποστέλλονται στην συσκευή IoT και αποθηκεύονται στην Flash.

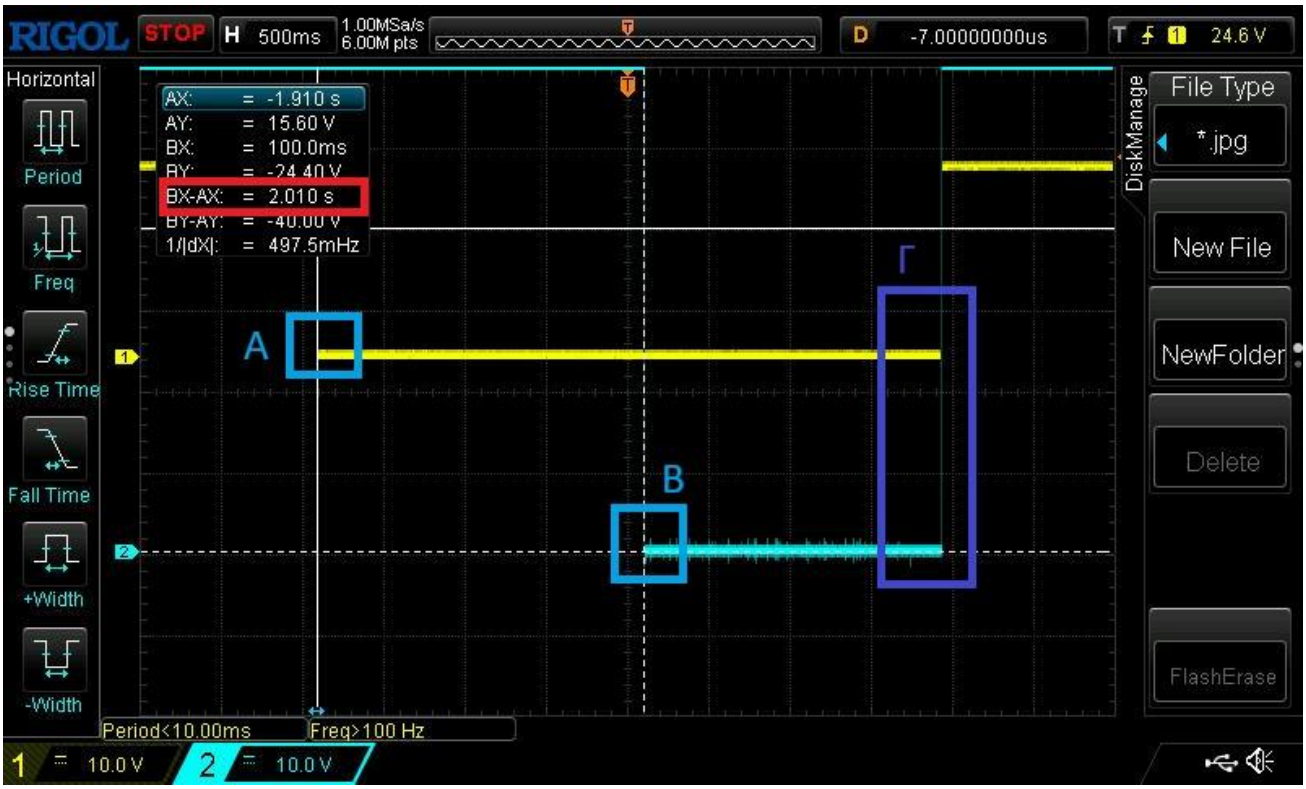
```
GenIoT_LTE/demo_user_1/sub {"cmd": "set_rules", "rules": [{"idx": 0, "source_id": 1, "condition": "ACTIVE_FOR", "thresho
ld": 0.0, "duration_ms": 2000, "action": {"type": "DO", "target_id": 1}}, {"idx": 1, "source_id": 11, "condition": "GT",
 "threshold": 1.0, "duration_ms": 0, "action": {"type": "DO", "target_id": 30}}]}
```

Εικόνα 42 Αποστολή κανόνων μέσω LTE

8.3 Εκτέλεση κανόνων

Κανόνας 1

Αρχικά ενεργοποιούμε την είσοδο δύο (IN2) και μετρώντας με τον παλμογράφο βλέπουμε πως η έξοδος δύο (O2) ενεργοποιείται μετά από δύο δευτερόλεπτα όπως επιλέχθηκε και στον κανόνα.



Εικόνα 43 Κανόνας 1: Ενεργοποίηση O2 2sec μετά την ενεργοποίηση της IN2

Πιο συγκεκριμένα παρατηρούμε πως ενεργοποιούμε την είσοδο 2 (A) και έπειτα από 2.010s ενεργοποιείται η έξοδος 2 (B). Απενεργοποιώντας την είσοδο, η έξοδος επανέρχεται χωρίς καθυστέρηση (Γ).

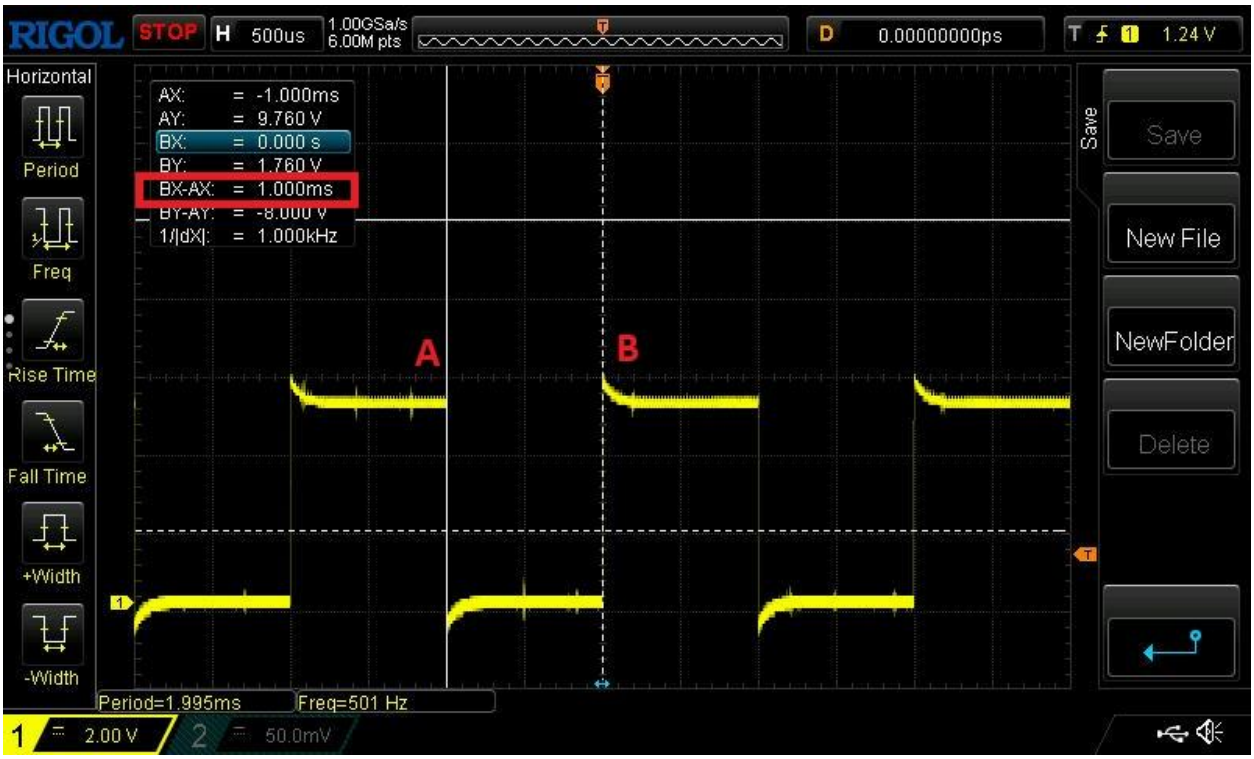
Κανόνας 2

Για να μετρήσουμε πάνω από 1A ρεύμα ως φορτίο χρησιμοποιήθηκε πιστολάκι μαλλιών στην 1η κλίμακα (ζεστό και αέρας στο 1). Έπειτα βλέπουμε πως ενεργοποιήθηκε το pwm με συχνότητα 500Hz και 50% duty cycle. Παραπάνω παρατηρούμε πως το ρεύμα που μετρήθηκε είναι 3.61A, ενώ η είσοδος PWM που έχει βραχυκυκλωθεί με την έξοδο ώστε να δούμε την μέτρηση και στην σελίδα μας δείχνει την συχνότητα και το duty cycle που έχει ρυθμιστεί.

Time (GR)	Vrms	Irms	P (W)	Vinst	Iinst	DI1	DI2	DI3	DI4	EXT1	EXT2	EXT3	EXT4	A1 V	A2 V	PWM f	PWM d%	Temp °C
2026-01-28 22:06:46	224.89	3.61	574.50	117.10	-0.00	1	1	1	1	0	0	0	0	5.00	3.00	500	52.0	50.0

Εικόνα 44 Μέτρηση ρεύματος και PWM In (input Capture)

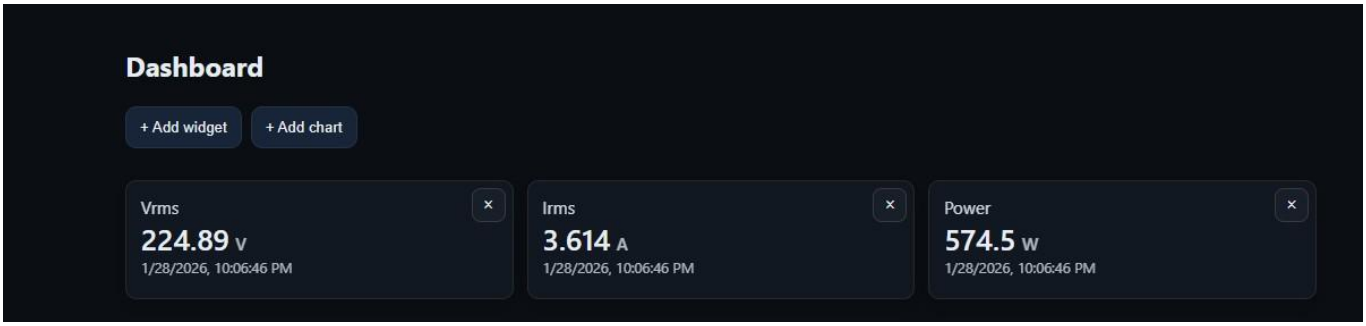
Ακόμα παρακάτω βλέπουμε και την μέτρηση με παλμογράφο στην έξοδο του PWM, αφότου υπάρχει φορτίο (πιστολάκι) στο δίκτυο το οποίο οδηγεί στην ενεργοποίηση του 2ου κανόνα.



Εικόνα 45 Μέτρηση PWM out με παλμογράφο

Παρατηρούμε πως η μέτρηση ξεκινά από την πίπτουσα παρυφή (falling edge) και αυτό γιατί η έξοδος υλοποιείται ως low-side N-MOSFET, όπου το φορτίο τροφοδοτείται από την κοινή θετική τάση (COMMON POS) και η διακοπή/διαμόρφωση PWM γίνεται στην αρνητική πλευρά μέσω MOSFET προς γείωση (GND).

Τέλος βλέπουμε την απεικόνιση από το dashboard την στιγμή της μέτρησης.



Εικόνα 46 Dashboard (live μετρήσεις)

9. Προβλήματα και αντιμετώπιση τους

9.1 Πρόβλημα βραχυκυκλώματος στην τροφοδοσία και αντικατάσταση μικροελεγκτή

Κατά τη φάση κατασκευής και συναρμολόγησης της πλακέτας, οι κολλήσεις πραγματοποιήθηκαν χειροκίνητα με κολλητήρι, χωρίς χρήση αυτοματοποιημένου εξοπλισμού. Ως αποτέλεσμα, προέκυψε βραχυκύκλωμα στο κύκλωμα τροφοδοσίας, το οποίο δεν εμφανίστηκε κατά τους αρχικούς ελέγχους. Μετά από κάποιες ώρες λειτουργίας του συστήματος, το σφάλμα οδήγησε σε αστοχία του μικροελεγκτή και τελικά σε καταστροφή του.

Η αιτία εντοπίστηκε στο στάδιο παραγωγής των 5V, και συγκεκριμένα στον LM2596S (5V), όπου η χειροκίνητη κόλληση και ο μη σωστός καθαρισμός του flux δημιούργησε γεφύρωση/βραχυκύκλωμα. Για την αποκατάσταση του προβλήματος, πραγματοποιήθηκε αποκόλληση του κατεστραμμένου μικροελεγκτή και αντικατάστασή του με νέο, ακολουθούμενη από επανέλεγχο της τροφοδοσίας πριν την επόμενη εκκίνηση.

Αυτό το περιστατικό ανέδειξε τη σημασία του λεπτομερούς ελέγχου (οπτικού και ηλεκτρικού) στο κύκλωμα τροφοδοσίας μετά από χειροκίνητες κολλήσεις, ιδιαίτερα σε σημεία υψηλής πυκνότητας ή/και ισχύος, όπως οι μετατροπείς DC–DC.

9.2 Ασυμβατότητα αντιστοίχισης ADC pins -αλλαγή μικροελεγκτή (STM32F - STM32G)

Ένα δεύτερο πρόβλημα προέκυψε κατά τη μετάβαση του σχεδιασμού από μικροελεγκτή της σειράς STM32F σε μικροελεγκτή της σειράς STM32G. Στον αρχικό σχεδιασμό, θεωρήθηκε ότι τα περισσότερα GPIO μπορούν να χρησιμοποιηθούν ως είσοδοι ADC μέσω εναλλακτικών λειτουργιών (alternate functions), όπως συνέβαινε στην προηγούμενη επιλογή μικροελεγκτή. Ωστόσο, μετά την αλλαγή σε STM32G, δεν εντοπίστηκε εγκαίρως ότι οι είσοδοι ADC υποστηρίζονται από συγκεκριμένο υποσύνολο ακροδεκτών (κυρίως από PAx).

Αυτό είχε ως αποτέλεσμα οι αναλογικές είσοδοι (ADC) να έχουν δρομολογηθεί σε ακροδέκτες PCx, οι οποίοι στη συγκεκριμένη σειρά δεν υποστηρίζουν ADC. Κατά συνέπεια, οι μετρήσεις ADC δεν μπορούσαν να λειτουργήσουν όπως είχε σχεδιαστεί σε επίπεδο PCB.

Η λύση που εφαρμόστηκε ήταν πρακτική και δεν απαίτησε επανασχεδίαση της πλακέτας: πραγματοποιήθηκαν εξωτερικές γεφυρώσεις (jumper wires) ώστε τα σήματα των αναλογικών εισόδων να οδηγηθούν σε διαθέσιμα PAx pins που δεν χρησιμοποιούνται. Συγκεκριμένα αξιοποιήθηκαν τα pins SCK_WIFI (PA1), SDI_WIFI (PA2) και SDO_WIFI (PA6), τα οποία είχαν προβλεφθεί αρχικά για επικοινωνία SPI με το WiFi module. Στην τελική υλοποίηση, το WiFi επικοινωνεί με τον μικροελεγκτή μέσω UART, οπότε τα συγκεκριμένα pins παρέμειναν ανενεργά και μπορούσαν να επαναχρησιμοποιηθούν για τις αναλογικές εισόδους.

Το συγκεκριμένο περιστατικό ανέδειξε τη σημασία της επαλήθευσης pin mapping (ιδιαίτερα των ADC-capable pins) μετά από αλλαγή οικογένειας μικροελεγκτή, ακόμη και όταν η αρχιτεκτονική παραμένει συμβατή σε επίπεδο εργαλείων και περιφερειακών.

10. Συμπεράσματα και προτάσεις βελτίωσης

10.1 Συμπεράσματα

Στο πλαίσιο της παρούσας διπλωματικής εργασίας αναπτύχθηκε ένα ολοκληρωμένο σύστημα IoT, το οποίο συνδυάζει σχεδίαση υλικού, υλοποίηση firmware, backend σύστημα και διαδικτυακή πλατφόρμα. Στόχος της εργασίας ήταν η δημιουργία μίας ευέλικτης πλατφόρμας απομακρυσμένου ελέγχου και εποπτείας, ικανής να προσαρμόζεται σε διαφορετικές εφαρμογές χωρίς αλλαγές στο βασικό hardware.

Η συσκευή υλοποιήθηκε με βάση μικροελεγκτή 32-bit ARM και υποστηρίζει πληθώρα εισόδων και εξόδων, όπως αναλογικές και ψηφιακές εισόδους, μετρήσεις AC τάσης και ρεύματος, PWM έξοδο, ρελέ και επικοινωνία μέσω CAN bus. Παράλληλα, ενσωματώνει ασύρματη επικοινωνία τόσο μέσω WiFi όσο και μέσω LTE, επιτρέποντας τη λειτουργία της σε διαφορετικά περιβάλλοντα και σενάρια εγκατάστασης.

Ιδιαίτερη έμφαση δόθηκε στη δομή του firmware, το οποίο σχεδιάστηκε με modular αρχιτεκτονική βασισμένη σε ανεξάρτητα components και κεντρικό system layer. Η χρήση abstraction μέσω του IoMap και η τοπική εκτέλεση κανόνων αυτοματισμού καθιστούν το σύστημα επεκτάσιμο και ανεξάρτητο από συγκεκριμένες εφαρμογές. Με τον τρόπο αυτό, η ίδια συσκευή μπορεί να χρησιμοποιηθεί σε διαφορετικά σενάρια, όπως ενεργειακή παρακολούθηση, αυτοματισμοί κτιρίων ή απομακρυσμένος έλεγχος βιομηχανικών διεργασιών.

Το backend σύστημα και η web πλατφόρμα συμπληρώνουν τη λειτουργικότητα της συσκευής, παρέχοντας στον χρήστη ενιαία διεπαφή για την παρακολούθηση δεδομένων, τη διαμόρφωση κανόνων και την παραμετροποίηση της λειτουργίας. Η αρχιτεκτονική πολλαπλών χρηστών και συσκευών επιτρέπει την κλιμάκωση του συστήματος και τη χρήση κοινής υποδομής.

Συνολικά, η εργασία αποδεικνύει ότι είναι εφικτή η ανάπτυξη μίας IoT συσκευής γενικού σκοπού με έμφαση στην ευελιξία, την επεκτασιμότητα και την αυτονομία λειτουργίας, αξιοποιώντας σύγχρονες τεχνολογίες ενσωματωμένων συστημάτων και διαδικτυακών εφαρμογών.

10.2 Μελλοντικές επεκτάσεις

Παρότι το σύστημα που αναπτύχθηκε καλύπτει ένα ευρύ φάσμα λειτουργιών, υπάρχουν αρκετές δυνατότητες μελλοντικής επέκτασης και βελτίωσης.

Μία βελτίωση που μπορεί να γίνει είναι η ενίσχυση της ασφάλειας επικοινωνίας, με τη χρήση πιστοποιητικών TLS και ιδιωτικών MQTT brokers. Επιπλέον, θα μπορούσε να υλοποιηθεί μηχανισμός απομακρυσμένης αναβάθμισης firmware (OTA), επιτρέποντας την εύκολη συντήρηση και αναβάθμιση των συσκευών στο πεδίο.

Στο κομμάτι της συνδεσιμότητας η σύνδεση WiFi μπορεί να παρέχει πρόσβαση στο internet, ώστε η συσκευή να συνδέεται μέσω ιδιωτικού MQTT broker, όπως συμβαίνει και με το LTE. Η σύνδεση WiFi στην παρούσα φάση είναι τοπική και ο server πρέπει να βρίσκεται στο ίδιο δίκτυο με την συσκευή. Επίσης στο κομμάτι της συνδεσιμότητας μία ακόμα βελτίωση μπορεί να είναι η δυνατότητα παραμετροποίησης του PIN και του APN ώστε να μπορούν να καλυφθούν όλα τα δίκτυα κινητής τηλεφωνίας.

Σε επίπεδο hardware, η ανάπτυξη επιπλέον πλακετών επέκτασης θα μπορούσε να αυξήσει περαιτέρω τη λειτουργικότητα της συσκευής, προσθέτοντας νέους τύπους εισόδων και εξόδων. Παράλληλα, θα μπορούσε να διερευνηθεί η βελτιστοποίηση της κατανάλωσης ενέργειας για εφαρμογές που απαιτούν λειτουργία με μπαταρία.

Όσον αφορά το backend η πιο σημαντική βελτίωση που μπορεί να γίνει είναι η υποστήριξη πάνω από μία συσκευή για κάθε λογαριασμό χρήση, δεδομένου πως ο κάθε χρήστης μπορεί να έχει στην κατοχή του πάνω από μία συσκευές.

Στο επίπεδο της web πλατφόρμας, μελλοντικές επεκτάσεις θα μπορούσαν να περιλαμβάνουν πιο προηγμένα εργαλεία ανάλυσης δεδομένων, ειδοποιήσεις σε πραγματικό χρόνο (notifications) και υποστήριξη περισσότερων τύπων οπτικοποίησης. Επιπλέον, η ενσωμάτωση αλγορίθμων ανάλυσης ή πρόβλεψης θα μπορούσε να προσδώσει επιπλέον αξία στο σύστημα.

Μία ακόμα πολύ σημαντική βελτίωση είναι η κρυπτογράφηση των δεδομένων της μνήμης Flash, καθώς οι κωδικοί της κάρτας SIM, αλλά και του WiFi αποθηκεύονται σε αυτήν και μπορούν να ανακτηθούν με ένα απλό διάβασμα της μνήμης.

Τέλος η δημιουργία μίας εφαρμογής στον υπολογιστή και κινητό ώστε ο χρήστης να μπορεί να παραμετροποιήσει την συσκευή χωρίς να χρειάζεται η πληκτρολόγηση των εντολών UART αποτελεί μία ακόμα βελτίωση. Προσφέρει ακόμα και στον χρήστη την δυνατότητα μερικής παραμετροποίησης κυρίως στην περίπτωση που δεν έχει πρόσβαση στο internet (πλατφόρμα) και χρειάζεται να αλλάξει κάποια ρύθμιση όπως τον κωδικό του WiFi.

Οι παραπάνω επεκτάσεις καταδεικνύουν ότι η πλατφόρμα που αναπτύχθηκε μπορεί να αποτελέσει βάση για περαιτέρω έρευνα και εξέλιξη στον τομέα των IoT συστημάτων και των ενσωματωμένων εφαρμογών.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Texas Instruments. (n.d.). LM2596 SIMPLE SWITCHER® power converter 150-kHz 3-A step-down voltage regulator [Datasheet]. Retrieved February 14, 2026, from https://www.ti.com/lit/ds/symlink/lm2596.pdf

[2] Microchip Technology. (2014, October 3). MIC29302A: 3A fast-response LDO regulator [Datasheet]. Retrieved February 14, 2026, from https://ww1.microchip.com/downloads/en/DeviceDoc/MIC29302A.pdf

[3] International Organization for Standardization. (2015). ISO 11898-1:2015: Road vehicles — Controller area network (CAN) — Part 1: Data link layer and physical coding sub-layer. Retrieved February 14, 2026, https://www.iso.org/standard/63648.html

[4] OASIS. (2019, March 7). MQTT Version 5.0 (OASIS Standard) [PDF]. Retrieved February 14, 2026, https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.pdf

[5] Texas Instruments. (n.d.). SN65HVD230Q-Q1 automotive catalog 3.3-V CAN transceiver (Product page). Retrieved February 14, 2026, https://www.ti.com/product/SN65HVD230Q-Q1

[6] Espressif Systems. (2016). ESP-WROOM-02 datasheet. Retrieved February 14, 2026, from https://www.espressif.com/sites/default/files/documentation/0c-esp-wroom-02_datasheet_en.pdf

[7] MikroElektronika. (n.d.). 4G LTE 2 Click – Data (Product page). Retrieved February 14, 2026, https://www.mikroe.com/4g-lte-2-click-data

[8] u-blox. (2022, August 18). LARA-R6 series: Single or multi-mode LTE Cat 1 modules with Secure Cloud—Data sheet (Document No. UBX-21004391, Rev. R09).

[9] STMicroelectronics. (2025, January). AN2867 application note: Guidelines for oscillator design on STM8AF/AL/S and STM32 MCUs/MPUs (AN2867). Retrieved February 14, 2026, https://www.st.com/resource/en/application_note/an2867-guidelines-for-oscillator-design-on-stm8afals-and-stm32-mcusmpus-stmicroelectronics.pdf

[10] Oracle. (n.d.). MySQL Documentation. Retrieved February 14, 2026, https://dev.mysql.com/doc/

[11] MariaDB. (n.d.). MariaDB Documentation. Retrieved February 14, 2026, https://mariadb.com/docs

[12] Eclipse Foundation. (n.d.). Eclipse Mosquitto documentation. Retrieved February 14, 2026, https://mosquitto.org/documentation/

[13] Eclipse Foundation. (n.d.). test.mosquitto.org — Public MQTT test broker. Retrieved February 14, 2026, https://test.mosquitto.org/

[14] Python Software Foundation. (n.d.). Python 3.13 documentation. Retrieved February 14, 2026, https://docs.python.org/3/

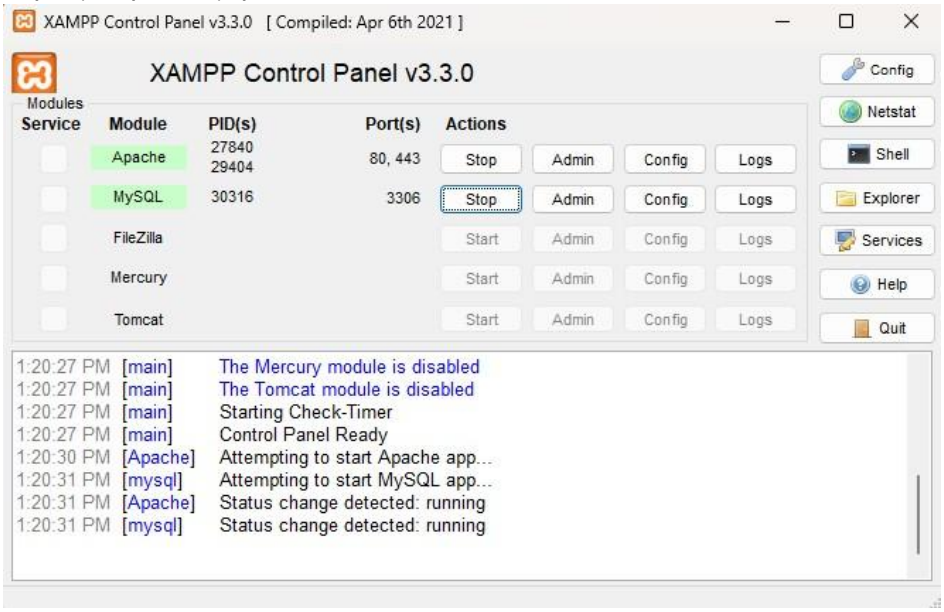
ΠΑΡΑΡΤΗΜΑ Α : Εργαλεία ανάπτυξης και υποστηρικτικό λογισμικό

Στο παρόν παράρτημα παρουσιάζονται συνοπτικά τα βασικά εργαλεία λογισμικού και υλικού που χρησιμοποιήθηκαν για την υλοποίηση, δοκιμή και αποσφαλμάτωση (debug) του συστήματος. Η επιλογή των εργαλείων έγινε με κριτήριο τη σταθερότητα, την ευκολία ενσωμάτωσης στη ροή ανάπτυξης και την υποστήριξη των τεχνολογιών που αξιοποιούνται (STM32, UART, MQTT, Python backend, βάση δεδομένων).

Α.1 XAMPP

Το XAMPP χρησιμοποιήθηκε ως τοπικό περιβάλλον ανάπτυξης (local development stack) για υπηρεσίες που σχετίζονται με την αποθήκευση και διαχείριση δεδομένων. Παρέχει έτοιμη εγκατάσταση και παραμετροποίηση ενός τυπικού “server stack”, διευκολύνοντας:

- τη λειτουργία της βάσης δεδομένων τοπικά (development/testing)
- τη γρήγορη δημιουργία/διαχείριση σχημάτων (schemas) και πινάκων
- την επαλήθευση της ροής δεδομένων από το backend προς τη βάση, χωρίς να απαιτείται απομακρυσμένη υποδομή.



Εικόνα 47 XAMPP Control Panel

Με αυτόν τον τρόπο, η ανάπτυξη και ο έλεγχος των λειτουργιών αποθήκευσης πραγματοποιήθηκαν σε ελεγχόμενο περιβάλλον, επιτρέποντας ταχύτερο έλεγχο σφαλμάτων και αλλαγών.

Α.2 Mosquitto MQTT Broker (τοπικό messaging & δοκιμές)

Ο Mosquitto χρησιμοποιήθηκε ως MQTT broker για την τοπική ανταλλαγή μηνυμάτων (LAN σενάριο) και για την ενδιάμεση δρομολόγηση/συγκέντρωση δεδομένων προς το backend. Η χρήση τοπικού broker ήταν χρήσιμη για:

- πλήρη έλεγχο της ροής MQTT (publish/subscribe) στο εργαστηριακό περιβάλλον.
- επαναληψιμότητα δοκιμών (σταθερές ρυθμίσεις, γνωστό latency, εύκολη παρακολούθηση logs).
- μεγαλύτερη ανεξαρτησία από εξωτερικές υπηρεσίες κατά το development.

A.2.1 Εκκίνηση Mosquitto με batch αρχείο (Windows)

Για την εκκίνηση του broker με συγκεκριμένο αρχείο ρυθμίσεων (mosquitto.conf)[12] δημιουργήθηκε ένα .bat αρχείο ώστε να μην πληκτρολογούνται κάθε φορά οι εντολές στο command prompt. Το αρχείο είναι όπως φαίνεται παρακάτω:

```
@echo off

rem go to mosquitto folder and run with your config

pushd "C:\Programs\mosquitto"

mosquitto.exe -c "C:\Development\Projects\Development IoT\Mosquitto\mosquitto.conf" -v

popd
```

Εικόνα 48 Moquitto - Εκκίνηση broker

A.2.2 Χρήση δημόσιου broker για LTE δοκιμές (test.mosquitto.org)

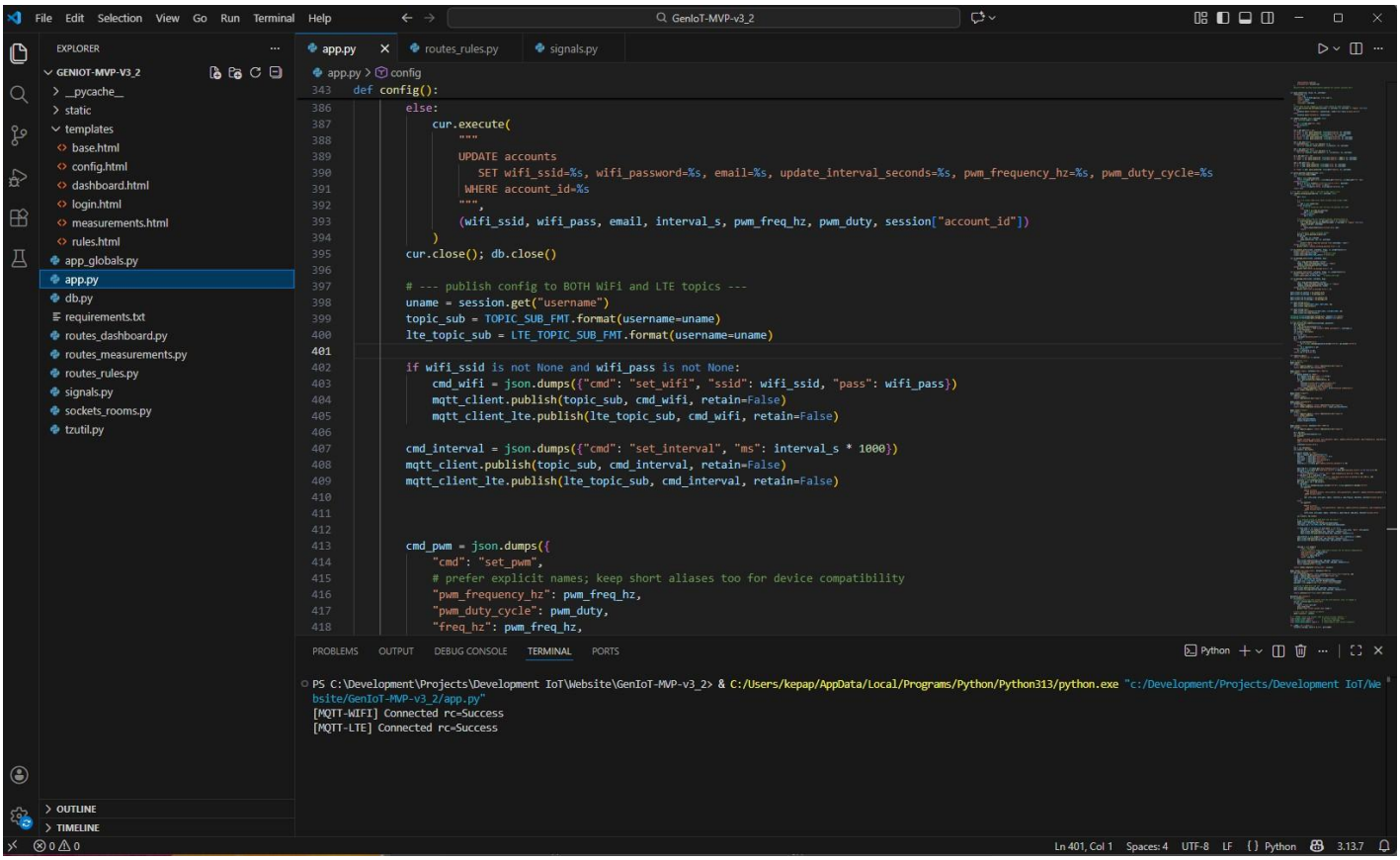
Για δοκιμές μέσω LTE, αξιοποιήθηκε και δημόσιος MQTT broker (π.χ. test.mosquitto.org)[13] ως σημείο δοκιμαστικής δημοσίευσης/λήψης μηνυμάτων. Παρότι δεν αποτελεί “εργαλείο” που εγκαθίσταται τοπικά, λειτουργεί ως εξωτερική υπηρεσία δοκιμών. Στο τελικό workflow, όπου απαιτείται, τα μηνύματα μπορούν να συλλεχθούν/αναδρομολογηθούν προς το backend μέσω της τοπικής υποδομής (π.χ. με local broker/clients), ώστε η αποθήκευση και η οπτικοποίηση να παραμένουν ενιαίες.

A.3 Visual Studio Code (ανάπτυξη και εκτέλεση backend -frontend)

Το Visual Studio Code (VS Code) χρησιμοποιήθηκε ως βασικό περιβάλλον ανάπτυξης τόσο για το backend όσο και για το frontend της εφαρμογής. Συγκεκριμένα:

- Χρησιμοποιήθηκε για επεξεργασία κώδικα Python και εκτέλεση της εφαρμογής (π.χ. app.py) ώστε να εκκινεί το backend
- Χρησιμοποιήθηκε για την ανάπτυξη του frontend (αρχεία HTML/CSS/JavaScript και templates όπου απαιτείται), επιτρέποντας γρήγορες αλλαγές και έλεγχο της διεπαφής χρήστη

- Διευκόλυνε τη διαχείριση του έργου (δομή φακέλων, extensions, ενσωματωμένο τερματικό) και την παρακολούθηση logs κατά τις δοκιμές
- Υποστήριξε τη διαδικασία debugging/ελέγχου κατά την ανάπτυξη, τόσο για λειτουργίες του server όσο και για συμπεριφορές της διεπαφής.



Εικόνα 49 Visual Studio Code - Εκτέλεση backend

A.4 Python (runtime περιβάλλον)

Για την εκτέλεση του backend χρησιμοποιήθηκε η έκδοση Python 3.13.7 [14]. Η εγκατάσταση της συγκεκριμένης έκδοσης εξασφάλισε συμβατότητα και σταθερό runtime περιβάλλον για την εφαρμογή (εκτέλεση services, βιβλιοθήκες, MQTT client λειτουργίες, διασύνδεση με βάση δεδομένων κ.λπ.).

A.5 HTerm (UART δοκιμές, αποσφαλμάτωση και παραμετροποίηση συσκευής)

Το HTerm χρησιμοποιήθηκε ως εργαλείο σειριακής επικοινωνίας (serial terminal) για:

- **Debug μέσω UART:** έλεγχος μηνυμάτων κατά την εκτέλεση (printf-style logs ή status messages)
- **Παραμετροποίηση μέσω User UART:** αποστολή παραμέτρων/εντολών προς τη συσκευή για ρύθμιση λειτουργίας, δοκιμές ή έλεγχο επικοινωνίας.

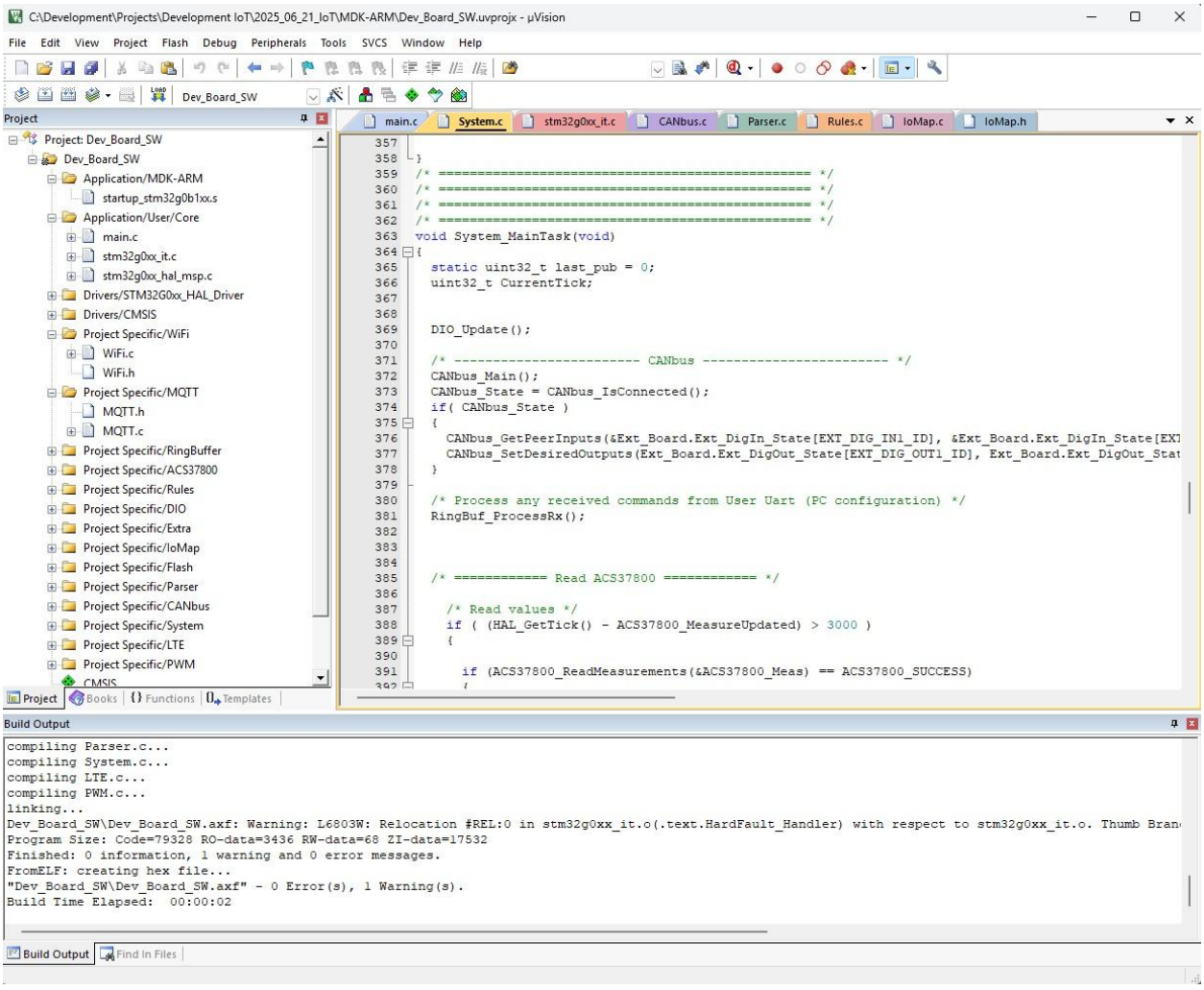
Η χρήση ενός αξιόπιστου serial terminal επέτρεψε γρήγορη επαλήθευση της επικοινωνίας σε επίπεδο χαρακτήρων/πλαισίων (frames) και διευκόλυνε την απομόνωση σφαλμάτων σε firmware ή συνδεσμολογία.

A.6 Keil µVision (ανάπτυξη firmware)

Το Keil µVision χρησιμοποιήθηκε ως IDE για το firmware του STM32, καλύπτοντας:

- Μεταγλώττιση (build) του κώδικα
- Διαχείριση έργου και ρυθμίσεων compiler/linker
- Υποστήριξη debugging σε συνδυασμό με hardware programmer/debugger.

Αποτελεί κλασική επιλογή σε STM32/ARM Cortex έργα, ειδικά όταν απαιτείται σταθερό περιβάλλον ανάπτυξης και εύκολη πρόσβαση σε debug εργαλεία.



Εικόνα 50 Keil uVision

A.7 ST-Link V2 (προγραμματισμός και debugging μέσω SWD)

Ο ST-Link V2 χρησιμοποιήθηκε για:

- προγραμματισμό (flashing) του μικροελεγκτή.
- debugging μέσω διεπαφής SWD (Serial Wire Debug).

Με αυτόν τον τρόπο κατέστη δυνατή η επιβεβαίωση της σωστής εκκίνησης του firmware, ο έλεγχος μεταβλητών/ροής εκτέλεσης και η διάγνωση προβλημάτων που δεν είναι εύκολο να εντοπιστούν μόνο μέσω UART logs.

ΠΑΡΑΡΤΗΜΑ Β : Συνοπτική παρουσίαση πηγαίου κώδικα

B.1 STM32

B.1.1 main.c

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * Copyright (c) 2025 STMicroelectronics.
  * All rights reserved.
  *
  * This software is licensed under terms that can be found in the LICENSE file
  * in the root directory of this software component.
  * If no LICENSE file comes with this software, it is provided AS-IS.
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"


/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "System.h"
#include "Flash.h"
/* USER CODE END Includes */
```

```
/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
FDCAN_HandleTypeDef hfdcan1;

I2C_HandleTypeDef hi2c1;
I2C_HandleTypeDef hi2c2;

RTC_HandleTypeDef hrtc;

SPI_HandleTypeDef hspi1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
DMA_HandleTypeDef hdma_tim2_ch1;
DMA_HandleTypeDef hdma_tim2_ch2;
```

```
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
UART_HandleTypeDef huart3;
DMA_HandleTypeDef hdma_usart1_tx;
DMA_HandleTypeDef hdma_usart2_tx;

WWDG_HandleTypeDef hwwdg;

/* USER CODE BEGIN PV */

uint32_t RunCnt;
uint8_t Testing_Condition = 0;

extern Flash_UserData_t g_UserData;

/* ---------------------------------------------------- */
/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_FDCAN1_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2C2_Init(void);
static void MX_RTC_Init(void);
static void MX_SPI1_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_USART3_UART_Init(void);
```

```
static void MX_WWDG_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM2_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{

  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
```

```
/* USER CODE END Init */

/* Configure the system clock */
SystemClock_Config();

/* USER CODE BEGIN SysInit */

/* USER CODE END SysInit */

/* Initialize all configured peripherals */
MX_GPIO_Init();
MX_DMA_Init();
MX_FDCAN1_Init();
MX_I2C1_Init();
MX_I2C2_Init();
MX_RTC_Init();
MX_SPI1_Init();
MX_USART1_UART_Init();
MX_USART2_UART_Init();
MX_USART3_UART_Init();
MX_WWDG_Init();
MX_TIM1_Init();
MX_TIM2_Init();
/* USER CODE BEGIN 2 */

        System_Init();

        HAL_Delay(100);

/* USER CODE END 2 */
```

```
  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

                    RunCnt++;

                    HAL_WWDG_Refresh(&hwwdg);

                    System_MainTask();

            }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
```

```
  HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1);

  /** Configure LSE Drive Capability
  */
  HAL_PWR_EnableBkUpAccess();
  __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType                                        =
RCC_OSCILLATORTYPE_HSE|RCC_OSCILLATORTYPE_LSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.LSEState = RCC_LSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = RCC_PLLM_DIV1;
  RCC_OscInitStruct.PLL.PLLN = 16;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
  RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1;
```

```
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief FDCAN1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_FDCAN1_Init(void)
{

  /* USER CODE BEGIN FDCAN1_Init 0 */

  /* USER CODE END FDCAN1_Init 0 */

  /* USER CODE BEGIN FDCAN1_Init 1 */

  /* USER CODE END FDCAN1_Init 1 */
  hfdcan1.Instance = FDCAN1;
  hfdcan1.Init.ClockDivider = FDCAN_CLOCK_DIV1;
  hfdcan1.Init.FrameFormat = FDCAN_FRAME_CLASSIC;
  hfdcan1.Init.Mode = FDCAN_MODE_NORMAL;
  hfdcan1.Init.AutoRetransmission = ENABLE;
```

```
  hfdcan1.Init.TransmitPause = DISABLE;
  hfdcan1.Init.ProtocolException = DISABLE;
  hfdcan1.Init.NominalPrescaler = 16;
  hfdcan1.Init.NominalSyncJumpWidth = 2;
  hfdcan1.Init.NominalTimeSeg1 = 13;
  hfdcan1.Init.NominalTimeSeg2 = 2;
  hfdcan1.Init.DataPrescaler = 1;
  hfdcan1.Init.DataSyncJumpWidth = 1;
  hfdcan1.Init.DataTimeSeg1 = 1;
  hfdcan1.Init.DataTimeSeg2 = 1;
  hfdcan1.Init.StdFiltersNbr = 2;
  hfdcan1.Init.ExtFiltersNbr = 0;
  hfdcan1.Init.TxFifoQueueMode = FDCAN_TX_FIFO_OPERATION;
  if (HAL_FDCAN_Init(&hfdcan1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN FDCAN1_Init 2 */

  /* USER CODE END FDCAN1_Init 2 */

}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{
```

```
/* USER CODE BEGIN I2C1_Init 0 */

/* USER CODE END I2C1_Init 0 */

/* USER CODE BEGIN I2C1_Init 1 */

/* USER CODE END I2C1_Init 1 */
hi2c1.Instance = I2C1;
hi2c1.Init.Timing = 0x10B17DB5;
hi2c1.Init.OwnAddress1 = 0;
hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2 = 0;
hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
if (HAL_I2C_Init(&hi2c1) != HAL_OK)
{
  Error_Handler();
}

/** Configure Analogue filter
*/
if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
{
  Error_Handler();
}

/** Configure Digital filter
```

```
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */

  /* USER CODE END I2C1_Init 2 */

}

/**
  * @brief I2C2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C2_Init(void)
{

  /* USER CODE BEGIN I2C2_Init 0 */

  /* USER CODE END I2C2_Init 0 */

  /* USER CODE BEGIN I2C2_Init 1 */

  /* USER CODE END I2C2_Init 1 */
  hi2c2.Instance = I2C2;
  hi2c2.Init.Timing = 0x10B17DB5;
  hi2c2.Init.OwnAddress1 = 0;
  hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
```

```
  hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c2.Init.OwnAddress2 = 0;
  hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
  hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c2) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Analogue filter
  */
  if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
  {
    Error_Handler();
  }

  /** Configure Digital filter
  */
  if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2C2_Init 2 */

  /* USER CODE END I2C2_Init 2 */

}

/**
```

```
  * @brief RTC Initialization Function
  * @param None
  * @retval None
  */
static void MX_RTC_Init(void)
{

  /* USER CODE BEGIN RTC_Init 0 */

  /* USER CODE END RTC_Init 0 */

  RTC_TimeTypeDef sTime = {0};
  RTC_DateTypeDef sDate = {0};

  /* USER CODE BEGIN RTC_Init 1 */

  /* USER CODE END RTC_Init 1 */

  /** Initialize RTC Only
  */
  hrtc.Instance = RTC;
  hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
  hrtc.Init.AsynchPrediv = 127;
  hrtc.Init.SynchPrediv = 255;
  hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
  hrtc.Init.OutPutRemap = RTC_OUTPUT_REMAP_NONE;
  hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
  hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
  hrtc.Init.OutPutPullUp = RTC_OUTPUT_PULLUP_NONE;
  if (HAL_RTC_Init(&hrtc) != HAL_OK)
```

```
{
  Error_Handler();
}

/* USER CODE BEGIN Check_RTC_BKUP */

/* USER CODE END Check_RTC_BKUP */

/** Initialize RTC and set the Time and Date
*/
sTime.Hours = 0;
sTime.Minutes = 0;
sTime.Seconds = 0;
sTime.SubSeconds = 0;
sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
sTime.StoreOperation = RTC_STOREOPERATION_RESET;
if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BIN) != HAL_OK)
{
  Error_Handler();
}
sDate.WeekDay = RTC_WEEKDAY_MONDAY;
sDate.Month = RTC_MONTH_JANUARY;
sDate.Date = 1;
sDate.Year = 0;

if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BIN) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN RTC_Init 2 */
```

```
  /* USER CODE END RTC_Init 2 */

}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */

  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */

  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_4BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
```

```
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 7;
  hspi1.Init.CRCLength = SPI_CRC_LENGTH_DATASIZE;
  hspi1.Init.NSSPMode = SPI_NSS_PULSE_ENABLE;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */

  /* USER CODE END SPI1_Init 2 */

}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
```

```
TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

/* USER CODE BEGIN TIM1_Init 1 */

/* USER CODE END TIM1_Init 1 */
htim1.Instance = TIM1;
htim1.Init.Prescaler = 63;
htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
htim1.Init.Period = 65535;
htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim1.Init.RepetitionCounter = 0;
htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
{
```

```
  Error_Handler();
}
sConfigOC.OCMode = TIM_OCMODE_PWM1;
sConfigOC.Pulse = 500;
sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
}
sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
sBreakDeadTimeConfig.DeadTime = 0;
sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
sBreakDeadTimeConfig.BreakFilter = 0;
sBreakDeadTimeConfig.BreakAFMode = TIM_BREAK_AFMODE_INPUT;
sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
sBreakDeadTimeConfig.Break2Filter = 0;
sBreakDeadTimeConfig.Break2AFMode = TIM_BREAK_AFMODE_INPUT;
sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
{
  Error_Handler();
}
```

```
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_IC_InitTypeDef sConfigIC = {0};

  /* USER CODE BEGIN TIM2_Init 1 */

  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 63;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 4294967295;
```

```
htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
{
  Error_Handler();
}
if (HAL_TIM_IC_Init(&htim2) != HAL_OK)
{
  Error_Handler();
}
sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
{
  Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
sConfigIC.ICFilter = 0;
if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
{
  Error_Handler();
}
sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
```

```
  sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
  if (HAL_TIM_IC_ConfigChannel(&htim2, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief USART1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART1_UART_Init(void)
{

  /* USER CODE BEGIN USART1_Init 0 */

  /* USER CODE END USART1_Init 0 */

  /* USER CODE BEGIN USART1_Init 1 */

  /* USER CODE END USART1_Init 1 */
  huart1.Instance = USART1;
  huart1.Init.BaudRate = 115200;
  huart1.Init.WordLength = UART_WORDLENGTH_8B;
  huart1.Init.StopBits = UART_STOPBITS_1;
```

```
  huart1.Init.Parity = UART_PARITY_NONE;
  huart1.Init.Mode = UART_MODE_TX_RX;
  huart1.Init.HwFlowCtl = UART_HWCONTROL_RTS_CTS;
  huart1.Init.OverSampling = UART_OVERSAMPLING_16;
  huart1.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart1.Init.ClockPrescaler = UART_PRESCALER_DIV1;
  huart1.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetTxFifoThreshold(&huart1, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetRxFifoThreshold(&huart1, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_DisableFifoMode(&huart1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART1_Init 2 */

  /* USER CODE END USART1_Init 2 */

}

/**
```

```
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.Init.ClockPrescaler = UART_PRESCALER_DIV1;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetTxFifoThreshold(&huart2, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  if (HAL_UARTEx_SetRxFifoThreshold(&huart2, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_UARTEx_DisableFifoMode(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * @brief USART3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART3_UART_Init(void)
{

  /* USER CODE BEGIN USART3_Init 0 */

  /* USER CODE END USART3_Init 0 */

  /* USER CODE BEGIN USART3_Init 1 */
```

```
/* USER CODE END USART3_Init 1 */
huart3.Instance = USART3;
huart3.Init.BaudRate = 115200;
huart3.Init.WordLength = UART_WORDLENGTH_8B;
huart3.Init.StopBits = UART_STOPBITS_1;
huart3.Init.Parity = UART_PARITY_NONE;
huart3.Init.Mode = UART_MODE_TX_RX;
huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
huart3.Init.OverSampling = UART_OVERSAMPLING_16;
huart3.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
huart3.Init.ClockPrescaler = UART_PRESCALER_DIV1;
huart3.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
if (HAL_UART_Init(&huart3) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_SetTxFifoThreshold(&huart3, UART_TXFIFO_THRESHOLD_1_8) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_SetRxFifoThreshold(&huart3, UART_RXFIFO_THRESHOLD_1_8) != HAL_OK)
{
  Error_Handler();
}
if (HAL_UARTEx_DisableFifoMode(&huart3) != HAL_OK)
{
  Error_Handler();
}
/* USER CODE BEGIN USART3_Init 2 */
```

```
  /* USER CODE END USART3_Init 2 */

}

/**
  * @brief WWDG Initialization Function
  * @param None
  * @retval None
  */
static void MX_WWDG_Init(void)
{

  /* USER CODE BEGIN WWDG_Init 0 */

  /* USER CODE END WWDG_Init 0 */

  /* USER CODE BEGIN WWDG_Init 1 */

  /* USER CODE END WWDG_Init 1 */
  hwwdg.Instance = WWDG;
  hwwdg.Init.Prescaler = WWDG_PRESCALER_1;
  hwwdg.Init.Window = 64;
  hwwdg.Init.Counter = 64;
  hwwdg.Init.EWIMode = WWDG_EWI_DISABLE;
  if (HAL_WWDG_Init(&hwwdg) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN WWDG_Init 2 */
```

```
  /* USER CODE END WWDG_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Channel1_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel1_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel1_IRQn);
  /* DMA1_Channel2_3_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel2_3_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel2_3_IRQn);
  /* DMA1_Ch4_7_DMA2_Ch1_5_DMAMUX1_OVR_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Ch4_7_DMA2_Ch1_5_DMAMUX1_OVR_IRQn, 1, 0);
  HAL_NVIC_EnableIRQ(DMA1_Ch4_7_DMA2_Ch1_5_DMAMUX1_OVR_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
```

```
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* USER CODE BEGIN MX_GPIO_Init_1 */

  /* USER CODE END MX_GPIO_Init_1 */

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC,                    GEN1_LED_Pin|GEN2_LED_Pin|GEN3_LED_Pin,
GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, PWR_4G_Pin|OUT4_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, RST_WIFI_Pin|EN_WIFI_Pin|GEN4_LED_Pin|OUT1_Pin
                    |OUT2_Pin|OUT3_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(RELAY_CON_GPIO_Port, RELAY_CON_Pin, GPIO_PIN_RESET);
```

```
/*Configure GPIO pins : AN_TEMP_Pin AN_IN1_CON_Pin AN_IN2_CON_Pin */
GPIO_InitStruct.Pin = AN_TEMP_Pin|AN_IN1_CON_Pin|AN_IN2_CON_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_ANALOG;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : GEN1_LED_Pin GEN2_LED_Pin GEN3_LED_Pin */
GPIO_InitStruct.Pin = GEN1_LED_Pin|GEN2_LED_Pin|GEN3_LED_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pin : STS_4G_Pin */
GPIO_InitStruct.Pin = STS_4G_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
HAL_GPIO_Init(STS_4G_GPIO_Port, &GPIO_InitStruct);

/*Configure GPIO pins : PWR_4G_Pin OUT4_Pin */
GPIO_InitStruct.Pin = PWR_4G_Pin|OUT4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

/*Configure GPIO pins : RI_4G_Pin DIO_1_Pin */
GPIO_InitStruct.Pin = RI_4G_Pin|DIO_1_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_PULLUP;
```

```
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : RST_WIFI_Pin EN_WIFI_Pin GEN4_LED_Pin OUT1_Pin
                         OUT2_Pin OUT3_Pin */
GPIO_InitStruct.Pin = RST_WIFI_Pin|EN_WIFI_Pin|GEN4_LED_Pin|OUT1_Pin
                      |OUT2_Pin|OUT3_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

/*Configure GPIO pins : IN1_Pin IN2_Pin */
GPIO_InitStruct.Pin = IN1_Pin|IN2_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

/*Configure GPIO pins : IN3_Pin IN4_Pin */
GPIO_InitStruct.Pin = IN3_Pin|IN4_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
GPIO_InitStruct.Pull = GPIO_NOPULL;
HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

/*Configure GPIO pin : RELAY_CON_Pin */
GPIO_InitStruct.Pin = RELAY_CON_Pin;
GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
GPIO_InitStruct.Pull = GPIO_NOPULL;
GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
HAL_GPIO_Init(RELAY_CON_GPIO_Port, &GPIO_InitStruct);
```

```
  /*Configure GPIO pin : DIO_0_Pin */
  GPIO_InitStruct.Pin = DIO_0_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_PULLUP;
  HAL_GPIO_Init(DIO_0_GPIO_Port, &GPIO_InitStruct);

  /* USER CODE BEGIN MX_GPIO_Init_2 */

  /* USER CODE END MX_GPIO_Init_2 */
}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  __disable_irq();
  while (1)
  {
  }
  /* USER CODE END Error_Handler_Debug */
}
#ifdef USE_FULL_ASSERT
```

```
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */
```

B.1.2 System.c

```
#include "main.h"
#include "stdio.h"
#include "string.h"
#include "System.h"
#include "ACS37800.h"
#include "Flash.h"
#include "Parser.h"
#include "WiFi.h"
#include "MQTT.h"
#include "RingBuffer.h"
#include "CANbus.h"
#include "DIO.h"
#include "Extra.h"
```

```
#include "LTE.h"
#include "PWM.h"



#define SYSTEM_MQTT_HEARTBEAT_INTERVAL              5000



/* ============================================================ */
/* ================== External variables ====================== */
/* ============================================================ */
extern I2C_HandleTypeDef hi2c2;
extern UART_HandleTypeDef huart1;
extern UART_HandleTypeDef huart2;
extern UART_HandleTypeDef huart3;



/* ============================================================ */
/* ================== Global variables ====================== */
/* ============================================================ */
/* Config loaded from flash into RAM */
Flash_Config_t G_Config;
Rule_t IoTRules[FLASH_MAX_RULES];
uint8_t WiFiRxData[1] = {0};
uint8_t UserUartRxData[1] = {0};
bool CANbus_State;



/* Led state array */
uint8_t Led_State[3] = {0, 0, 0};
```

```
/* Debug User Uart */
char UserUart_MsgBuffer[100];

/* External board */
EXT_BOARD_t Ext_Board;

/* ACS37800 */
uint32_t ACS37800_MeasureUpdated = 0;
/* Measurement struct */
ACS37800_Measurements_t ACS37800_Meas;

/* WiFi variables */
WiFi_Status_t WiFi_ConStat;
WiFi_Error_t WiFi_Err;

/* LTE Variables */
uint8_t System_LTE_Status;
uint8_t LteRxData[1];

/* MQTT Variables */
MQTT_Config_t mqttCfg;
MQTT_Status_t MQTT_ConStat;
static uint8_t mqttStarted = 0;
/* Publish */
uint8_t MQTT_PublishStatus;
uint32_t MQTT_MeasUpdatedTime = 0;
uint32_t MQTT_HearbeatUpdatedTime = 0;
uint32_t MQTT_MeasUpdateInterval = 10000;
float Parser_DummyAn1 = 5000;
```

```
float Parser_DummyAn2 = 3000;
float Parser_DummyTemperature = 50;
/* Local buffer for received messages */
static char RxMsg[800];
static int RxMsgLen = 0;

/* PWM test */
uint8_t PWM_Test = 0U;
uint32_t PWM_FreqSet = 1000U;
uint8_t PWM_DcSet = 50U;
uint32_t PWM_FreqRead = 0U;
uint8_t PWM_DcRead = 0U;

HAL_StatusTypeDef st;




extern Flash_UserData_t g_UserData;

extern void LTE_PumpRx(void);
/* ================================================= */
/* ==================== Local API ==================== */
/* ================================================= */

static void System_ACS37800_Init(void)
{
/* ------------------- ACS37800 init ------------------- */

/* Initialize the ACS37800 sensor */
        if (ACS37800_Init(&hi2c2, ACS37800_DEFAULT_I2C_ADDRESS) == ACS37800_SUCCESS)
```

```
        {
                __NOP();
        }
        else
        {
                __NOP();
        }

/* Configure measured resistor values */
        ACS37800_SetSenseRes(2040.0f);       /* Measured RSENSE in ohms */
        ACS37800_SetDividerRes(2000000.0f);  /* Voltage divider total (RISO1+RISO2) */
        ACS37800_SetCurrentRange(30.0f);     /* 30A version */

}
/* ================================================ */
/* ================================================ */
static void System_Rules_Init(void)
{
        /* Load rules from flash  */
        if (Flash_LoadConfig(&G_Config) == 0)
        {
                /* Copy rules into working array */
                for (uint8_t i = 0; i < G_Config.RuleCount; i++)
                {
                        IoTRules[i] = G_Config.Rules[i];
                }

                Rules_Init(IoTRules, G_Config.RuleCount);
        }
        else
```

```
        {
                /* No config in flash. Start with 0 rules */
                G_Config.RuleCount = 0;
                Rules_Init(IoTRules, 0);
        }


        /*  Tell the parser where the rules live and how many slots exist */
        Parser_Init(IoTRules, FLASH_MAX_RULES);

}
/* ================================================ */
/* ================================================ */
static void System_WiFi_Init(void)
{

        HAL_UART_Receive_IT(&huart2, WiFiRxData, 1);
        WiFi_Init(&huart2);

        /* WiFi configuration */
        WiFi_Config_t WiFiConfig;
        WiFiConfig.SSID = "COSMOTE-774632";          // replace with your SSID
        WiFiConfig.Password = "dpbg569skee456nk4gbm"; // replace with your password
        WiFiConfig.TimeoutMs = 5000;       // 5s timeout per step
        WiFiConfig.MaxRetries = 5;         // max 3 reconnect attempts
        WiFi_Configure(&WiFiConfig);

}
/* ================================================ */
/* ================================================ */
```

```
static void System_MQTT_Init(void)
{

	/* === MQTT Config === */

	mqttCfg.ClientID     = "GenIoT-STM32";
	mqttCfg.BrokerHost   = MQTT_DEFAULT_BROKER_HOST;
	mqttCfg.BrokerPort   = MQTT_DEFAULT_BROKER_PORT;
	mqttCfg.TopicPub     = MQTT_DEFAULT_TOPIC_PUB;
	mqttCfg.TopicSub     = MQTT_DEFAULT_TOPIC_SUB;
	mqttCfg.KeepAliveSec = 30;
	mqttCfg.MaxRetries   = 3;

}
/* ================================================ */
/* ================================================ */
static void System_ConnectionHandler(void)
{
	/* WiFi */
	WiFi_ConStat = WiFi_GetStatus();
	if (WiFi_ConStat == WIFI_CONNECTED)
	{
			/* Set Led when WiFi is connected */
			Led_State[2] = GPIO_PIN_SET;
	}
	else if (WiFi_ConStat == WIFI_ERROR)
	{
			/* Reset Led when WiFi is connected */
			Led_State[2] = GPIO_PIN_RESET;
			WiFi_Err = WiFi_GetLastError();
```

```
	}
	HAL_GPIO_WritePin(GEN4_LED_GPIO_Port, GEN4_LED_Pin, Led_State[2]);


	/* Start MQTT only after WiFi is connected + some extra delay */
	if (!mqttStarted && ( ((g_UserData.ConnMode == CON_MODE_WIFI) || (g_UserData.ConnMode == CON_MODE_WIFI_LTE)) && (WiFi_ConStat == WIFI_CONNECTED) )
	)
	{
			static uint32_t wifiConnectTick = 0;


			if (wifiConnectTick == 0)
			{
					wifiConnectTick = HAL_GetTick(); /* mark the moment WiFi first went connected */
			}
			else if ((HAL_GetTick() - wifiConnectTick) > 5000) /* wait 5 seconds */
			{
					MQTT_Init(&mqttCfg);
					mqttStarted = 1;
			}
	}


}
/* ================================================ */
/* ================================================ */
static void System_PublishHandler(void)
{
	static uint8_t sent = 0;
	uint8_t i, Loc_DigInState[DIG_IN_MAX_NUMBER], Loc_ExtDigInState[EXT_DIG_IN_MAX_NUMBER];
	uint8_t ConnectionPub;
```

```
if(MQTT_ConStat == MQTT_CONNECTED)
{
	ConnectionPub = 1U;
}
else if(System_LTE_Status)
{
	ConnectionPub = 2U;
}
else
{
	ConnectionPub = 0U;
}



/* Check if connected to broker */
MQTT_ConStat = MQTT_GetStatus();
if ((MQTT_ConStat == MQTT_CONNECTED) || System_LTE_Status)
{
	HAL_GPIO_WritePin(GEN3_LED_GPIO_Port, GEN3_LED_Pin, GPIO_PIN_SET);
		/* Example: publish once */

		if (!sent)
		{
			if(MQTT_ConStat == MQTT_CONNECTED)
			{
				MQTT_Publish("Hello from STM32 WiFi!");
				sent = 1;
			}
			else if(System_LTE_Status)
```

```
                {
                        LTE_MQTT_Publish("Hello from STM32 LTE!");
                        sent = 1;
                }
                MQTT_MeasUpdatedTime = HAL_GetTick();
        }

    /* ------------------------------------------------------------------------ */
    /* -------------------- P U B L I S H   M E A S U R E M E N T S -------------------- */
    /* ------------------------------------------------------------------------ */
    if ( (HAL_GetTick() - MQTT_MeasUpdatedTime) > MQTT_MeasUpdateInterval )
    {
                HAL_GPIO_WritePin(GEN2_LED_GPIO_Port,  GEN2_LED_Pin,
GPIO_PIN_SET);
                for(i=0;i<DIG_IN_MAX_NUMBER;i++)
                {
                        Loc_DigInState[i] = DigIn_Get(i);
                        Loc_ExtDigInState[i] = Ext_Board.Ext_DigIn_State[i];
                }



                MQTT_PublishStatus  =  Parser_PublishTelemetry(ConnectionPub,
ACS37800_Meas.VoltageRMS,

                                                ACS37800_Meas.CurrentRMS,

                                                ACS37800_Meas.ActivePower,

                                                ACS37800_Meas.InstantVoltage,

                                                ACS37800_Meas.InstantCurrent,
```

```
                                                    Loc_DigInState,          Loc_ExtDigInState,
Parser_DummyAn1/*Analog  1*/,  Parser_DummyAn2/*Analog  2*/,  PWM_FreqRead/*Hz*/,
PWM_DcRead/*Duty Cycle*/, Parser_DummyTemperature/*Temperature*/);

                                RingBuf_FlushToUART(&huart3, 300);
                                MQTT_MeasUpdatedTime = HAL_GetTick();
                                MQTT_HearbeatUpdatedTime = HAL_GetTick();

                                HAL_GPIO_WritePin(GEN2_LED_GPIO_Port,      GEN2_LED_Pin,
GPIO_PIN_RESET);

                        }
                /* ------------------------------------------------------------------------- */
                /* ------------------------ Check for received messages -------------------------- */
                /* ------------------------------------------------------------------------- */
                        RxMsgLen = MQTT_Receive(RxMsg, sizeof(RxMsg));
                        if ( RxMsgLen> 0)
                        {
                                char Loc_Resp[320];
                                char Loc_Log[300];
                                uint8_t Loc_ParserResp;

                                Loc_ParserResp  =  Parser_ProcessMessage(RxMsg,  Loc_Resp,
sizeof(Loc_Resp));
                                /* Log in Uart */
                                snprintf(Loc_Log, sizeof(Loc_Log), "\r\nSTM32 Website Rec: %s\r\n",
RxMsg);
                                RingBuf_PutBuf((uint8_t *)Loc_Log, sizeof(Loc_Log));
                                RingBuf_FlushToUART(&huart3, sizeof(Loc_Log));
                                RingBuf_FlushToUART(&huart3, sizeof(Loc_Log));
```

```
                                /* Echo back to broker */
                                MQTT_Publish(Loc_Resp);
                        }
                }
                else
                {
                        HAL_GPIO_WritePin(GEN3_LED_GPIO_Port, GEN3_LED_Pin, GPIO_PIN_RESET);

                }









}

/* ================================================ */
/* ================== Global API ================== */
/* ================================================ */

void System_Init(void)
{
        RingBuf_Init();
        System_ACS37800_Init();
        System_Rules_Init();
        if( (g_UserData.ConnMode == CON_MODE_WIFI) || (g_UserData.ConnMode ==
CON_MODE_WIFI_LTE) )
        {
                System_WiFi_Init();
                System_MQTT_Init();
```

```
        }

        /* Enable interrupts for User Uart */
        HAL_UART_Receive_IT(&huart3, UserUartRxData, 1);



        CANbus_Init(CANBUS_NODE_ID_1, CANBUS_ROLE_MASTER);

        /* power up the LARA-R6 and start the state machine */
        if( (g_UserData.ConnMode == CON_MODE_LTE) || (g_UserData.ConnMode ==
CON_MODE_WIFI_LTE) )
        {
                LTE_Init();
                HAL_GPIO_WritePin(PWR_4G_GPIO_Port, PWR_4G_Pin, GPIO_PIN_SET);
  HAL_Delay(100);
  HAL_GPIO_WritePin(PWR_4G_GPIO_Port, PWR_4G_Pin, GPIO_PIN_RESET);
                HAL_Delay(100);
                HAL_UART_Receive_IT(&huart1, LteRxData, 1);
        }

        PWM_Init();



}
/* ================================================ */
/* ================================================ */
/* ================================================ */
/* ================================================ */
void System_MainTask(void)
{
```

```
        static uint32_t last_pub = 0;
        uint32_t CurrentTick;



        DIO_Update();


        /* ------------------------ CANbus ------------------------ */
        CANbus_Main();
        CANbus_State = CANbus_IsConnected();
        if( CANbus_State )
        {
                CANbus_GetPeerInputs(&Ext_Board.Ext_DigIn_State[EXT_DIG_IN1_ID],
&Ext_Board.Ext_DigIn_State[EXT_DIG_IN2_ID],   &Ext_Board.Ext_DigIn_State[EXT_DIG_IN3_ID],
&Ext_Board.Ext_DigIn_State[EXT_DIG_IN4_ID]);
                CANbus_SetDesiredOutputs(Ext_Board.Ext_DigOut_State[EXT_DIG_OUT1_ID],
Ext_Board.Ext_DigOut_State[EXT_DIG_OUT2_ID],
Ext_Board.Ext_DigOut_State[EXT_DIG_OUT3_ID],
Ext_Board.Ext_DigOut_State[EXT_DIG_OUT4_ID], Ext_Board.Ext_Relay_State);
        }


        /* Process any received commands from User Uart (PC configuration) */
 RingBuf_ProcessRx();




        /* ============= Read ACS37800 ============= */


                /* Read values */
                if ( (HAL_GetTick() - ACS37800_MeasureUpdated) > 3000 )
                {
```

```
                if (ACS37800_ReadMeasurements(&ACS37800_Meas) == ACS37800_SUCCESS)
                {
                    sprintf(UserUart_MsgBuffer,"============================\n");
                    RingBuf_PutBuf((uint8_t*)UserUart_MsgBuffer, strlen(UserUart_MsgBuffer));

                    sprintf(UserUart_MsgBuffer,
                                        "Vrms = %.2f V, Irms = %.3f A, P = %.2f W, "
                                        "Vinst = %.2f V, Iinst = %.3f A\r\n",
                                        ACS37800_Meas.VoltageRMS, ACS37800_Meas.CurrentRMS, ACS37800_Meas.ActivePower, ACS37800_Meas.InstantVoltage, ACS37800_Meas.InstantCurrent);
                    RingBuf_PutBuf((uint8_t*)UserUart_MsgBuffer, strlen(UserUart_MsgBuffer));

                    sprintf(UserUart_MsgBuffer,"============================\n");
                    RingBuf_PutBuf((uint8_t*)UserUart_MsgBuffer, strlen(UserUart_MsgBuffer));
                }
                else
                {
                    /* Increase a counter to set values to zero or indicate an error */
                    sprintf(UserUart_MsgBuffer,"R E A D  A C S 3 7 8 0 0 -> E R R O R \n");
                    RingBuf_PutBuf((uint8_t*)UserUart_MsgBuffer, strlen(UserUart_MsgBuffer));
                }

            ACS37800_MeasureUpdated = HAL_GetTick();

            }
```

```
		/* Run WiFi and MQTT state machines */
		if( (g_UserData.ConnMode == CON_MODE_WIFI) || (g_UserData.ConnMode == CON_MODE_WIFI_LTE) )
		{
			WiFi_Task();
			MQTT_Task();
		}

		WiFi_ConStat = WiFi_GetStatus();

		/* Run LTE state machine */
		if( (g_UserData.ConnMode == CON_MODE_LTE) || (g_UserData.ConnMode == CON_MODE_WIFI_LTE))
		{
			LTE_NetSetWiFiLink((WiFi_ConStat == WIFI_CONNECTED));
			LTE_Main();

			System_LTE_Status = LTE_IsReady();
		}

		/* Run Rules */
		CurrentTick = HAL_GetTick();
		Rules_Execute(CurrentTick);

		/* Connection handler */
		System_ConnectionHandler();

		/* Publish handler */
		System_PublishHandler();
```

```
        /* Periodically flush up to 20 bytes of debug logs */
        RingBuf_FlushToUART(&huart3, 20);



        if(PWM_IsNewSampleReady() != 0U)
                {
                PWM_ClearNewSampleFlag();

                PWM_ProcessDMA();

                PWM_FreqRead= PWM_GetFreq();
                PWM_DcRead = PWM_GetDC();

                }


} /* End of System_MainTask */



/**
 * @brief  Main:HAL_UART_RxCpltCallback function.
 * @details Auto-generated header. See implementation for specifics.
 * @param  huart  (in) parameter.
 * @return Return status or void depending on function.
 */
void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
```

```
{

	if(huart->Instance == USART2)
	{
		WiFi_ReceiveIT(WiFiRxData[0]);
		/* Push received byte into debug buffer */
		//RingBuf_Put(WiFiRxData[0]); Deactivate to avoid hard fault

		HAL_UART_Receive_IT(huart, WiFiRxData, 1);
	}
	else if(huart->Instance == USART3)
	{
		RingBuf_RxPut(UserUartRxData[0]);
		HAL_UART_Receive_IT(huart, UserUartRxData, 1);

	}
	else if(huart->Instance == USART1)
	{
			LTE_ReceiveIT(LteRxData[0]);
			HAL_UART_Receive_IT(huart, LteRxData, 1);

	}
}

/**
  * @brief Tx Transfer completed callback.
  * @param huart UART handle.
  * @retval None
  */
void HAL_UART_TxCpltCallback(UART_HandleTypeDef *huart)
```

```
{
  if(huart->Instance == USART2)
  {
    WiFi_TxCpltCallback();
  }
        else if(huart->Instance == USART1)
        {
                LTE_TxCpltIT();
        }
}

/**
  * @brief Callback hook to be called from HAL_TIM_IC_CaptureCallback().
  *
  * @param htim Pointer to TIM handle from HAL.
  */
void PWM_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
{

        PWM_TIM_IC_CaptureCallback(htim);
}



void HAL_UART_ErrorCallback(UART_HandleTypeDef *huart)
{
        if (huart->Instance == USART1)
        {
                uint32_t err = huart->ErrorCode;

                /* Abort current RX to reset huart->RxState */
```

```
    (void)HAL_UART_AbortReceive_IT(huart);

                __HAL_UART_CLEAR_FLAG(huart, UART_CLEAR_FEF | UART_CLEAR_NEF |
UART_CLEAR_OREF);

                /* Reset HAL error state so it doesn't stay "stuck" */
                huart->ErrorCode = HAL_UART_ERROR_NONE;

                /* Re-arm RX for the next byte */
                st = HAL_UART_Receive_IT(huart, LteRxData, 1);

        }
}
```

B.2 Backend (Python)

```
# app.py (GenIoT-MVP v4, patched with LTE MQTT)
import json, re, threading, time
from datetime import datetime, timedelta, timezone
from flask import jsonify, redirect, render_template, request, session, url_for
from flask_socketio import emit, join_room
import paho.mqtt.client as mqtt
import bcrypt

from app_globals import (
    app, socketio,
    mqtt_client, mqtt_client_lte,
    MQTT_HOST, MQTT_PORT,
    LTE_MQTT_HOST, LTE_MQTT_PORT,
    TOPIC_PUB, TOPIC_PUB_LEGACY,
    TOPIC_SUB_FMT,
```

```
    LTE_TOPIC_PUB, LTE_TOPIC_SUB_FMT
)
from db import get_db
from signals import SOURCES, TARGETS, ID_TO_NAME

# ===== App & MQTT config =====
app.config['SECRET_KEY'] = "dev-secret-change-me"

CLIENT_ID  = "GenIoT-WebBridge"
LTE_CLIENT_ID = "GenIoT-WebBridge-LTE"

latest = {}  # id -> {id,name,value,ts,username}

# ===== Helpers =====
def js_like_to_json(s: str) -> str:
    """
    Convert a JS-like object literal:
      {type:telemetry,device:GenIoT-STM32,...}
    into valid JSON:
      {"type":"telemetry","device":"GenIoT-STM32",...}
    """
    # 1) Quote keys: {type:  -> {"type":
    s = re.sub(r'([{,])(\s*)([A-Za-z_][A-Za-z0-9_]*)\s*:',
               r'\1"\3":', s)

    # 2) Quote bareword string values (telemetry, GenIoT-STM32, etc.)
    def _val_repl(m):
        prefix = m.group(1)   # ':"' or ': '
        val    = m.group(2)
```

```
        # Already JSON-ish / numeric? leave it
        if (re.fullmatch(r'-?\d+(\.\d+)?', val) or
            val in ("true", "false", "null") or
            val.startswith(("{", "[", ""))):
            return prefix + val

        # Otherwise treat as string
        return prefix + "" + val + ""

    s = re.sub(r'(:\s*)([^,\]}]+)', _val_repl, s)
    return s

def topic_username(topic: str):
    """Extract <username> from 'GenIoT/<username>/pub'."""
    parts = topic.split('/')
    if len(parts) >= 3 and parts[0] == "GenIoT" and parts[2] == "pub":
        return parts[1]
    return None

def topic_username_lte(topic: str):
    """Extract <username> from 'GenIoT_LTE/<username>/pub'."""
    parts = topic.split('/')
    if len(parts) >= 3 and parts[0] == "GenIoT_LTE" and parts[2] == "pub":
        return parts[1]
    return None

def get_account_by_username(username: str):
    db = get_db()
    cur = db.cursor(dictionary=True)
    cur.execute("SELECT * FROM accounts WHERE username=%s", (username,))
```

```
    row = cur.fetchone()
    cur.close(); db.close()
    return row

def store_measurement(account_id: int, pkt: dict):
    ts = pkt.get("ts")
    ts_dt = datetime.now(timezone.utc) + timedelta(hours=3)
    try:
        if isinstance(ts, (int, float)) and ts > 10_000_000:
            ts_dt = datetime.utcfromtimestamp(float(ts) / 1000.0)
    except Exception:
        pass

    din = pkt.get("din", [])
    di1 = int(din[0]) if isinstance(din, list) and len(din) >= 1 else int(pkt.get("di1", 0))
    di2 = int(din[1]) if isinstance(din, list) and len(din) >= 2 else int(pkt.get("di2", 0))
    di3 = int(din[2]) if isinstance(din, list) and len(din) >= 3 else int(pkt.get("di3", 0))
    di4 = int(din[3]) if isinstance(din, list) and len(din) >= 4 else int(pkt.get("di4", 0))

    ext = pkt.get("ext_din", [])
    ext_di1 = int(ext[0]) if isinstance(ext, list) and len(ext) >= 1 else int(pkt.get("ext_di1", 0))
    ext_di2 = int(ext[1]) if isinstance(ext, list) and len(ext) >= 2 else int(pkt.get("ext_di2", 0))
    ext_di3 = int(ext[2]) if isinstance(ext, list) and len(ext) >= 3 else int(pkt.get("ext_di3", 0))
    ext_di4 = int(ext[3]) if isinstance(ext, list) and len(ext) >= 4 else int(pkt.get("ext_di4", 0))

    acs = pkt.get("acs", {})
    vrms = acs.get("Vrms"); irms = acs.get("Irms"); power = acs.get("P")
    vinst = acs.get("Vinst"); iinst = acs.get("Iinst")

    ai = pkt.get("ai", {})
```

```
a1mV = ai.get("a1mV"); a2mV = ai.get("a2mV")
an1_voltage = (float(a1mV)/1000.0) if a1mV is not None else None
an2_voltage = (float(a2mV)/1000.0) if a2mV is not None else None

pwm = pkt.get("pwm", {})
pwm_f = pwm.get("f"); pwm_d = pwm.get("d")
temperature = pkt.get("temp")

db = get_db()
cur = db.cursor()
cur.execute("""
    INSERT INTO measurements (
      account_id, ts,
      di1, di2, di3, di4,
      ext_di1, ext_di2, ext_di3, ext_di4,
      pwm_frequency_hz, pwm_duty_cycle,
      an1_voltage, an2_voltage,
      vrms, irms, power_w, v_instant, i_instant, temperature_c)
    VALUES (%s,%s,
            %s,%s,%s,%s,
            %s,%s,%s,%s,
            %s,%s,
            %s,%s,
            %s,%s,%s,%s,%s,%s)
""", (account_id, ts_dt,
      di1, di2, di3, di4,
      ext_di1, ext_di2, ext_di3, ext_di4,
      pwm_f, pwm_d,
      an1_voltage, an2_voltage,
      vrms, irms, power, vinst, iinst, temperature))
```

```
    cur.close(); db.close()

    # Tell all connected browsers that measurements changed
    from app_globals import socketio
    socketio.emit(
        "measurements_updated",
        {"account_id": account_id}
    )
    #print(f"[WS] emitted measurements_updated for account {account_id}")



def push_latest(sid, value, ts, username):
    latest[sid] = {
        "id": sid,
        "name": ID_TO_NAME.get(sid, f"ID {sid}"),
        "value": value,
        "ts": int(ts),
        "username": username
    }
    # Emit ONLY to the logged-in user's room (based on topic username)
    acc = get_account_by_username(username) if username and username != "legacy" else None
    if acc:
        socketio.emit("telemetry", latest[sid], room=f"user:{acc['account_id']}")
    else:
        socketio.emit("telemetry", latest[sid])

def ingest_rich(pkt: dict, username: str):
    ts = int(time.time() * 1000)
    try:
        ts = int(pkt.get("ts", ts))
```

```
    except Exception:
        pass

    acs = pkt.get("acs", {})
    if "Vrms" in acs: push_latest(10, float(acs["Vrms"]), ts, username)
    if "Irms" in acs: push_latest(11, float(acs["Irms"]), ts, username)
    if "P" in acs: push_latest(12, float(acs["P"]), ts, username)
    if "Vinst" in acs: push_latest(40, float(acs["Vinst"]), ts, username)
    if "Iinst" in acs: push_latest(41, float(acs["Iinst"]), ts, username)

    din = pkt.get("din")
    if isinstance(din, list) and len(din) >= 4:
        for i in range(4): push_latest(i, int(din[i]), ts, username)

    ext = pkt.get("ext_din")
    if isinstance(ext, list) and len(ext) >= 4:
        for i in range(4): push_latest(4 + i, int(ext[i]), ts, username)

    ai = pkt.get("ai", {})
    if "a1mV" in ai: push_latest(20, float(ai["a1mV"]) / 1000.0, ts, username)
    if "a2mV" in ai: push_latest(21, float(ai["a2mV"]) / 1000.0, ts, username)

    pwm = pkt.get("pwm", {})
    if "f" in pwm: push_latest(30, float(pwm["f"]), ts, username)
    if "d" in pwm: push_latest(31, float(pwm["d"]), ts, username)

    if "temp" in pkt: push_latest(50, float(pkt["temp"]), ts, username)

def parse_payload_simple(payload: str):
    ts = int(time.time()*1000)
```

```
    try:
        data = json.loads(payload)
        return int(data.get("id")), float(data.get("value")), int(data.get("ts", ts))
    except Exception:
        pairs = dict(re.findall(r"(\w+)\s*=\s*([\w.\-]+)", payload))
        if "id" in pairs and "value" in pairs:
            return int(pairs["id"]), float(pairs["value"]), ts
    return None

# ===== MQTT callbacks (WiFi + LTE share same logic) =====
def _handle_incoming_payload(raw: str, username: str):
    try:
        pkt = None

        # 1) Try normal JSON first (WiFi already sends proper JSON)
        try:
            pkt = json.loads(raw)
        except Exception:
            # 2) Try to normalize your JS-like LTE payload into JSON
            try:
                fixed = js_like_to_json(raw)
                pkt = json.loads(fixed)
            except Exception:
                pkt = None

        # If this looks like our telemetry packet, process/store it
        if isinstance(pkt, dict) and pkt.get("type") == "telemetry":
            acc = get_account_by_username(username) if username != "legacy" else None
            ingest_rich(pkt, username)
            if acc:
```

```
            store_measurement(acc["account_id"], pkt)
        return

    # 3) Fallback: simple id/value format
    parsed = parse_payload_simple(raw)
    if parsed:
        sid, val, ts = parsed
        push_latest(sid, val, ts, username)
    else:
        print(f"[MQTT] Unparsed payload from {username}: {raw}")
  except Exception as e:
    print("[MQTT] _handle_incoming_payload error:", e)

def on_connect_wifi(client, userdata, flags, rc, properties=None):
  print(f"[MQTT-WIFI] Connected rc={rc}")
  client.subscribe(TOPIC_PUB)        # GenIoT/+/pub
  client.subscribe(TOPIC_PUB_LEGACY) # GenIoT/pub

def on_message_wifi(client, userdata, msg):
  try:
    raw = msg.payload.decode().strip()
    uname = topic_username(msg.topic) or "legacy"
    _handle_incoming_payload(raw, uname)
  except Exception as e:
    print("[MQTT-WIFI] on_message error:", e)

def on_connect_lte(client, userdata, flags, rc, properties=None):
  print(f"[MQTT-LTE] Connected rc={rc}")
  client.subscribe(LTE_TOPIC_PUB)    # GenIoT_LTE/+/pub
```

```
def on_message_lte(client, userdata, msg):
    try:
        raw = msg.payload.decode().strip()
        uname = topic_username_lte(msg.topic) or "legacy"
        _handle_incoming_payload(raw, uname)
    except Exception as e:
        print("[MQTT-LTE] on_message error:", e)

mqtt_client.on_connect = on_connect_wifi
mqtt_client.on_message = on_message_wifi

mqtt_client_lte.on_connect = on_connect_lte
mqtt_client_lte.on_message = on_message_lte

def mqtt_thread_wifi():
    mqtt_client.connect(MQTT_HOST, MQTT_PORT, 30)
    mqtt_client.loop_forever()

def mqtt_thread_lte():
    mqtt_client_lte.connect(LTE_MQTT_HOST, LTE_MQTT_PORT, 30)
    mqtt_client_lte.loop_forever()

threading.Thread(target=mqtt_thread_wifi, daemon=True).start()
threading.Thread(target=mqtt_thread_lte, daemon=True).start()

# ===== Auth (MySQL) =====
def get_account_by_credentials(username, password):
    db = get_db()
    cur = db.cursor(dictionary=True)
    cur.execute("SELECT * FROM accounts WHERE username=%s", (username,))
```

```
    row = cur.fetchone()
    cur.close(); db.close()
    if not row:
        return None
    ph = row.get("password_hash") or ""
    ok = False
    try:
        if ph.startswith("$2"):
            ok = bcrypt.checkpw(password.encode("utf-8"), ph.encode("utf-8"))
        else:
            ok = (password == ph)
    except Exception:
        ok = (password == ph)
    return row if ok else None

def require_login():
    return "account_id" in session

# ===== Routes =====
@app.route("/")
def index():
    if not require_login(): return redirect(url_for("login"))
    return redirect(url_for("dashboard"))

@app.route("/login", methods=["GET","POST"])
def login():
    if request.method == "POST":
        u = request.form.get("user","").strip()
        p = request.form.get("pass","")
        row = get_account_by_credentials(u, p)
```

```
        if row:
            session["account_id"] = row["account_id"]
            session["username"] = row["username"]
            return redirect(url_for("dashboard"))
        return render_template("login.html", error="Invalid credentials")
    return render_template("login.html")

@app.route("/logout")
def logout():
    session.clear()
    return redirect(url_for("login"))

@app.route("/dashboard")
def dashboard():
    if not require_login(): return redirect(url_for("login"))
    return render_template("dashboard.html", input_sources=SOURCES)

@app.route("/rules")
def rules():
    if not require_login(): return redirect(url_for("login"))
    return render_template(
        "rules.html",
        input_sources=SOURCES,
        output_targets=TARGETS
    )

@app.route("/config", methods=["GET","POST"])
def config():
    if not require_login(): return redirect(url_for("login"))
```

```
    db = get_db()
    cur = db.cursor(dictionary=True)
    cur.execute(
        """
        SELECT username, wifi_ssid, wifi_password, email, update_interval_seconds, pwm_frequency_hz,
pwm_duty_cycle
        FROM accounts WHERE account_id=%s
        """,
        (session["account_id"],)
    )
    acc = cur.fetchone()
    cur.close(); db.close()

    if request.method == "POST":
        data = request.get_json(force=True)
        new_pass   = data.get("password") or None
        wifi_ssid  = data.get("wifi_ssid")
        wifi_pass  = data.get("wifi_password")
        email      = data.get("email")
        interval_s = int(data.get("update_interval_seconds") or 30)



        pwm_freq_hz = int(data.get("pwm_frequency_hz") or 1000)
        pwm_duty = float(data.get("pwm_duty_cycle") if data.get("pwm_duty_cycle") is not None else 50)
        if pwm_freq_hz <= 0:
            return jsonify({"ok": False, "err": "pwm_frequency_hz must be > 0"}), 400
        if pwm_duty < 0 or pwm_duty > 100:
            return jsonify({"ok": False, "err": "pwm_duty_cycle must be between 0 and 100"}), 400
        # store duty as integer percent for consistency
        pwm_duty = int(round(pwm_duty))
```

```
    db = get_db(); cur = db.cursor()
    if new_pass:
        ph = bcrypt.hashpw(new_pass.encode("utf-8"), bcrypt.gensalt()).decode("utf-8")
        cur.execute(
            """
            UPDATE accounts
              SET password_hash=%s, wifi_ssid=%s, wifi_password=%s, email=%s, update_interval_seconds=%s, pwm_frequency_hz=%s, pwm_duty_cycle=%s
             WHERE account_id=%s
            """,
            (ph, wifi_ssid, wifi_pass, email, interval_s, pwm_freq_hz, pwm_duty, session["account_id"])
        )
    else:
        cur.execute(
            """
            UPDATE accounts
              SET wifi_ssid=%s, wifi_password=%s, email=%s, update_interval_seconds=%s, pwm_frequency_hz=%s, pwm_duty_cycle=%s
             WHERE account_id=%s
            """,
            (wifi_ssid, wifi_pass, email, interval_s, pwm_freq_hz, pwm_duty, session["account_id"])
        )
    cur.close(); db.close()

    # --- publish config to BOTH WiFi and LTE topics ---
    uname = session.get("username")
    topic_sub = TOPIC_SUB_FMT.format(username=uname)
    lte_topic_sub = LTE_TOPIC_SUB_FMT.format(username=uname)

    if wifi_ssid is not None and wifi_pass is not None:
        cmd_wifi = json.dumps({"cmd": "set_wifi", "ssid": wifi_ssid, "pass": wifi_pass})
```

```
            mqtt_client.publish(topic_sub, cmd_wifi, retain=False)
            mqtt_client_lte.publish(lte_topic_sub, cmd_wifi, retain=False)

        cmd_interval = json.dumps({"cmd": "set_interval", "ms": interval_s * 1000})
        mqtt_client.publish(topic_sub, cmd_interval, retain=False)
        mqtt_client_lte.publish(lte_topic_sub, cmd_interval, retain=False)




        cmd_pwm = json.dumps({
            "cmd": "set_pwm",
            # prefer explicit names; keep short aliases too for device compatibility
            "pwm_frequency_hz": pwm_freq_hz,
            "pwm_duty_cycle": pwm_duty,
            "freq_hz": pwm_freq_hz,
            "duty": pwm_duty
        })
        mqtt_client.publish(topic_sub, cmd_pwm, retain=False)
        mqtt_client_lte.publish(lte_topic_sub, cmd_pwm, retain=False)
        return jsonify({"ok": True})

    return render_template("config.html", acc=acc)

@app.route("/api/send_rules", methods=["POST"])
def api_send_rules():
    if not require_login(): return jsonify({"ok":False,"err":"auth"}), 401
    rules = request.get_json(force=True).get("rules",[])
    uname = session.get("username")
    topic_sub = TOPIC_SUB_FMT.format(username=uname)
    lte_topic_sub = LTE_TOPIC_SUB_FMT.format(username=uname)
```

```
    payload = json.dumps({"cmd":"set_rules","rules":rules})

    # publish to BOTH WiFi and LTE
    mqtt_client.publish(topic_sub, payload, retain=False)
    mqtt_client_lte.publish(lte_topic_sub, payload, retain=False)

    return jsonify({"ok":True,"sent":len(rules)})

@socketio.on("connect")
def ws_connect():
    # Automatically put this socket into the user-specific room, if logged in
    acc_id = session.get("account_id")
    if acc_id:
        room = f"user:{acc_id}"
        join_room(room)
        print(f"[WS] client joined room {room}")

    # Still send the snapshot as before
    emit("snapshot", latest)

# --- IMPORT routes and sockets LAST to avoid circular imports ---
from sockets_rooms import *          # per-user Socket.IO rooms
from routes_rules import *           # /api/rules GET/POST
from routes_measurements import *    # /measurements with custom from–until

if __name__ == "__main__":
    socketio.run(app, host="0.0.0.0", port=5000)
```