ΔΙΕΘΝΕΣ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΤΗΣ ΕΛΛΑΔΟΣ

ΠΟΛΥΤΕΧΝΙΚΗ ΣΧΟΛΗ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

«ΑΝΑΠΤΥΞΗ ΘΥΡΟΤΗΛΕΦΩΝΟΥ ΜΕ ΔΙΑΣΥΝΔΕΣΗ ΣΕ ΚΙΝΗΤΟ ΤΗΛΕΦΩΝΟ»

Των φοιτητών
ΟΡΕΣΤΗ ΜΑΛΑ
Αρ. Μητρώου: 515079
ΑΝΑΝΙΑΔΗ ΘΕΜΙΣΤΟΚΛΗ
Αρ. Μητρώου: 515007

Επιβλέπων
Ονοματεπώνυμο
ΚΙΟΣΚΕΡΙΔΗΣ ΙΟΡΔΑΝΗΣ
Βαθμίδα Καθηγητής

Ημερομηνία 29/05/2025

Τίτλος Π.Ε. ΑΝΑΠΤΥΞΗ ΘΥΡΟΤΗΛΕΦΩΝΟΥ ΜΕ ΔΙΑΣΥΝΔΕΣΗ ΣΕ ΚΙΝΗΤΟ ΤΗΛΕΦΩΝΟ

Κωδικός Π.Ε **23359**

Ονοματεπώνυμο φοιτητών: ΟΡΕΣΤΗΣ ΜΑΛΑ, ΑΝΑΝΙΑΔΗΣ ΘΕΜΙΣΤΟΚΛΗΣ

Ονοματεπώνυμο εισηγητή: ΚΙΟΣΚΕΡΙΔΗΣ ΙΟΡΔΑΝΗΣ

Ημερομηνία ανάληψης Π.Ε 03-04-2024

Ημερομηνία περάτωσης Π.Ε 29-05-2025

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 Πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η παρούσα πτυχιακή εργασία είναι αφιερωμένη στους γονείς μας που με την στήριξη τους καταφέραμε να φέρουμε εις πέρας τις σπουδές μας.

Στους φίλους και συναδέλφους μας που στάθηκαν δίπλα μας στην πορεία μας όλα αυτά τα χρόνια.

Και τέλος στους εαυτούς μας, για την δύναμη και την υπομονή που έχουμε καταβάλει καθόλη την διάρκεια των σπουδών μας.

«Αφιέρωση»

Πρόλογος

Η παρούσα πτυχιακή εργασία επικεντρώνεται στο σχεδιασμό και την υλοποίηση ενός συστήματος θυροτηλεφώνου με ασύρματη διασύνδεση σε κινητό τηλέφωνο.

Με την τεχνολογία να εξελίσσεται ραγδαία και πιο συγκεκριμένα στο κομμάτι της ασύρματης επικοινωνίας καθώς και στο κομμάτι των "έξυπνων" εφαρμογών αποφασίσαμε να δημιουργήσουμε ένα θυροτηλέφωνο με ασύρματη διασύνδεση σε κινητό τηλέφωνο με σκοπό να παρέχουμε στον χρήστη την δυνατότητα να ελέγχει την είσοδο της κατοικίας του από απόσταση.

Για την υλοποίηση της συσκευής βασιστήκαμε στον μικροελεγκτή ESP32 λόγω το ότι προσφέρει ενσωματωμένη συνδεσιμότητα WI-FI καθώς και στην πλατφόρμα MIT App Invertor μέσω της οποίας αναπτύχθηκε μια εφαρμογή για Android κινητές συσκευές η οποία δίνει την δυνατότητα στον εκάστοτε χρήστη να λαμβάνει ειδοποιήσεις και να επικοινωνεί σε πραγματικό χρόνο με τον επισκέπτη.

Η εργασία αυτή συνδυάζει τόσο θεωρητικές όσο και πρακτικές γνώσεις για την ανάπτυξη ενός σύγχρονου συστήματος θυροτηλεφώνου αξιοποιώντας της δυνατότητες της σύγχρονης τεχνολογιας.

Περίληψη

Η παρούσα εργασία έχει ως αντικείμενο τον σχεδιασμό και την υλοποίηση ενός ασύρματου συστήματος θυροτηλεφώνου το οποίο δίνει την δυνατότητα στον χρήστη να ελέγχει απομακρυσμένα την είσοδο της κατοικίας του με του κινητού του τηλεφώνου.

Η εργασία ξεκινά με μια ιστορική αναδρομή της εξέλιξης των συστημάτων θυροτηλεφώνων, παρουσιάζοντας τη μετάβαση από τις αρχικές, ενσύρματες λύσεις σε πιο σύγχρονες, ασύρματες και "έξυπνες" εφαρμογές. Στη συνέχεια, ακολουθεί αναλυτική παρουσίαση των τεχνολογιών και των εργαλείων που χρησιμοποιήθηκαν, τόσο σε επίπεδο υλικού όσο και λογισμικού. Ιδιαίτερη έμφαση δίνεται στον μικροελεγκτή ESP32, καθώς και στην ανάπτυξη της εφαρμογής Android μέσω της πλατφόρμας MIT App Inventor. Περιγράφεται λεπτομερώς η διαδικασία σχεδιασμού του κυκλώματος, η σύνδεση των απαραίτητων περιφερειακών (μικρόφωνο, ηχείο, κουμπί κλήσης κ.λπ.), καθώς και ο τρόπος με τον οποίο επιτυγχάνεται η επικοινωνία με την κινητή συσκευή. Τέλος κλείνουμε με μια αξιολόγηση του συστήματος, αναλύονται τα αποτελέσματα και προτείνονται πιθανές βελτιώσεις.

Η συνολική δομή της εργασίας στοχεύει στο να προσφέρει μια πλήρη εικόνα, από τη θεωρητική προσέγγιση μέχρι την πρακτική υλοποίηση.

« DEVELOPMENT OF AN INTERCOM WITH INTERFACE TO A MOBILE PHONE »

«ORESTI MALA

ANANIADIS THEMISTOKLIS»

Abstract

The present thesis focuses on the design and implementation of a wireless door intercom system, which enables the user to remotely control the entrance of their residence through a mobile device.

The study begins with a historical overview of the evolution of door intercom systems, highlighting the transition from traditional wired solutions to modern wireless and "smart" applications. It then presents a detailed analysis of the technologies and tools used in the development process, covering both hardware and software components. Particular emphasis is placed on the ESP32 microcontroller, which provides built-in Wi-Fi connectivity, as well as the MIT App Inventor platform used for developing the Android mobile application. The thesis describes the circuit design, the integration of essential peripherals (such as microphone, speaker, and call button), and the communication flow between the device and the mobile application. Finally, the system is evaluated based on its functionality and performance, the results are discussed, and potential improvements are proposed for future development.

Overall, this project aims to provide a complete view from the theoretical background to the practical implementation.

Ευχαριστίες

Πρώτα απ όλα θα θέλαμε να ευχαριστήσουμε τον επιβλέποντα καθηγητή μας κ. Ιορδάνη Κιοσκερίδη για την στήριξη καθώς και την επιστημονική καθοδήγηση που μας προσέφερε καθόλη την διάρκεια η οποία ήταν πολύτιμη για την ολοκλήρωση του έργου.

Επιπλέον τις θερμές μας ευχαριστίες στις οικογένειες μας για την αγάπη και την υποστήριξη που μας παρείχαν όλα αυτά τα χρόνια.

Περιεχόμενα

Κατάλογος Εικόνων

Κεφάλαιο 1

Κεφάλαιο 2

Κεφάλαιο 3

Κεφάλαιο 4

Κατάλογος Πινάκων

Κεφάλαιο 1ο: Θυροτηλέφωνα

1.1 Ορισμός

Το θυροτηλέφωνο (intercom συντομία του intercommunication device), είναι ένα σύστημα επικοινωνίας που χρησιμοποιείται συνήθως σε κτίρια όπως, πολυκατοικίες, μονοκατοικίες ή ακόμη και μικρή ομάδα κτιρίων. Επιτρέπει την αμφίδρομη επικοινωνία ανάμεσα σε άτομα που βρίσκονται σε διαφορετικούς χώρους, χωρίς αναγκαστικά να χρησιμοποιεί κάποιο τηλεφωνικό δίκτυο. Επίσης η συσκευή έχει την δυνατότητα να κλειδώσει και να ξεκλειδώσει μια πόρτα.

1.2 Ιστορική Αναδρομή

Τον δρόμο χάραξε η εφεύρεση του Graham Bell το 1876, το τηλέφωνο. Πριν όμως απο αυτές τις εφευρέσεις, ο τρόπος για να τραβήξεις την προσοχή κάποιου ήταν με την μορφή κάποιου «σινιάλου». Για παράδειγμα, η επικοινωνία σε πλούσιες έπαυλες από το σαλόνι στο κατάλυμα των υπηρετών γινόταν από ένα σύστημα τροχαλιών συνδεδεμένες σε καμπάνες που θα ηχούσαν όταν τις τραβάς. Έτσι όταν η καμπάνα του αντίστοιχου υπηρέτη ηχούσε, αυτός ήξερε πως πρέπει να εμφανιστεί. Καθώς ο ηλεκτρισμός και οι ηλεκτρικές συσκευές γίνονταν πιο συνηθισμένα, το προηγούμενο σύστημα τροχαλιών αντικαταστάθηκε από συστήματα με καλώδια τα οποία συνέδεαν ένα μπουτόν με κάποιου είδους βομβητή που θα ηχούσε αντίστοιχα. Μέχρι αυτό το σημείο όμως ο υπάλληλος απλά ειδοποιούνταν πως κάποιος χρειαζόταν τις υπηρεσίες του, οπότε έπρεπε να περπατήσει από την μια περιοχή στην άλλη για να ενημερωθεί για το τι κάποιος χρειαζόταν.

Εικόνα 1-1:Σύστημα κλήσης με τροχαλία.

Πιο σημαντικό όμως, για να αυξηθεί η αποδοτικότητα , ήταν να μπορεί να υπάρχει επικοινωνία ανάμεσα σε δυο δωμάτια που βρίσκονταν μακριά. Έτσι γεννήθηκε η ιδέα του «Σωλήνα φωνής». Πρόκειται για δύο κώνους σε κάθε μεριά, ενωμένους από μία σωλήνα από την οποία ταξιδεύει ο ήχος. Στις αρχές του 18ου αιώνα ο Jean-Baptiste Biot πειραματιζόταν με το πώς ταξιδεύει ο ήχος σε μακριές σωλήνες, χρησιμοποιώντας αυτές του Παρισίου, δηλαδή τις αποχετεύσεις. Το πόρισμα που έβγαλε, ήταν ότι όσο οριοθετείς τον ήχο σε σωλήνα με μικρή διάμετρο τότε οι λέξεις μπορεί να γίνουν κατανοητές σε απόσταση μέχρι και 1000 περίπου μέτρα σε σχέση με ανοιχτό χώρο. Η πρώτη πραγματική αναφορά όμως στον σωλήνα αυτό έγινε το 1849 σε ένα άρθρο στο περιοδικό «Scientific American» που περιέγραφε έναν «ακουστικό τηλέγραφο» ο οποίος θα επιτρέπει στους ανθρώπους να επικοινωνούν μεταξύ τους σε απόσταση έως και 60 μίλια απο μια σωλήνα φτιαγμένη από gutta percha. Αυτό όμως αποδείχθηκε κάπως φιλόδοξο, παρόλα αυτά το άρθρο έδειξε πόσο χρήσιμο θα ήταν ένα τέτοιο εργαλείο σε διάφορα κτίρια όπως βιομηχανίες,δημόσια κτίρια κ.λ.π. Στην συνέχεια ο Antonio Meucci, Ιταλός μετανάστης στην Νέα Υόρκη ανέπτυξε περαιτέρω το «σωλήνα φωνής» όπως επίσης εγκατέστησε ένα τέτοιο σύστημα στο ίδιο του το σπίτι. Επίσης, η εφεύρεση αυτή χρησιμοποιήθηκε και σε πρώιμα μοντέλα αεροσκαφών όπου η επικοινωνία μεταξύ πιλότου και δεύτερου πιλότου ήταν δύσκολη λόγω του κινητήρα. Για αυτό τον λόγο, εγκαταστάθηκε ένας τέτοιος πλαστικός σωλήνας που συνδεόταν με ακουστικά φτιαγμένα από ύφασμα και καουτσούκ στην άλλη άκρη. Έτσι, ο πιλότος μιλούσε στο χωνί και δεύτερος πιλότος άκουγε μέσω αυτών των ακουστικών.

Εικόνα 1-2: «Σωλήνας φωνής» σε αποβατικό σκάφος του ναυτικού των Ηνωμένων πολιτειών. Φωτογραφία απο το 2005.

Επιστρέφουμε όμως πάλι στο τηλέφωνο, το οποίο ήταν η εφεύρεση που έπρεπε να προηγηθεί για να έχουμε το θυροτηλέφωνο. Το σύστημα θυροτηλεφώνου βασισμένο σε τηλέφωνο πατένταρε πρώτη η εταιρεία “Kellogg Switchboard and Supply Company” το 1894. Αυτό λειτουργούσε όταν ο επισκέπτης τραβούσε ενα χερούλι στην είσοδο της κατοικίας κλείνοντας έτσι ένα κύκλωμα και επιτρέποντας το ρεύμα να περάσει από το καλώδιο στον δέκτη που βρισκόταν μέσα στο διαμέρισμα του κατοίκου. Ο δέκτης ήταν ένα μπάζερ που απαρτιζόταν από δύο καμπάνες και ένα μικρο κομμάτι μέταλλο στην μέση που χτυπούσε όταν το διαπερνούσε ρεύμα. Όμως αυτό δεν επέτρεπε ακόμα την επικοινωνία μεταξύ επισκέπτη και κατοίκου. Το 1920 με 1930 η “Kellogg Switchboard and Supply Company” πρόσθεσε τηλεφωνικά κυκλώματα με ακουστικό και στόμιο.

Εικόνα 1-3: Θυροτηλέφωνο την δεκαετία του 20 στην έπαυλη Πίτοκ στις Ηνωμένες πολιτείες.

Εώς το 1950 τα περισσότερα κτίρια της Νέας Υόρκης και του λονδίνου διέθεταν τέτοια συστήματα με εμφανείς κιόλας βελτιώσεις όπως κουμπιά στην είσοδο και την επιλογή στον κάτοικο να ανοίγει την κεντρική είσοδο εξ αποστάσεως. Το 1947, με την εφεύρεση του τρανζίστορ, μια συσκευή που ενισχύει ή λειτουργεί σαν διακόπτης χωρίς να κινείται, ανέπτυξαν περαιτέρω το θυροτηλέφωνο. Όσο περνούσαν τα χρόνια η κατασκευή του γινόταν μικρότερη με καλύτερα συστήματα ήχου (ηχεία/μικρόφωνα) . Τις δεκαετίες 1990 με 2000 προστέθηκαν σε αυτό λειτουργίες όπως βίντεο και ασύρματη σύνδεση. Τα τελευταία 20 χρόνια το θυροτηλέφωνο έχει μεταμορφωθεί σε ψηφιακή συσκευή με το αναλογικό σύστημα ήχου να δίνει την θέση του σε IP συστήματα, με την χρήση πρωτοκόλλων δικτύου και εξ αποστάσεως πρόσβαση μέσω Smartphones.

Εικόνα 1-4 Εξωτερικό πάνελ σε πολυκατοικία στην δεκαετία του 1950.

1.3 Τύποι Συστημάτων Θυροτηλεφώνου

Τα θυροτηλέφωνα έρχονται σε διάφορους τύπους, ο καθένας κατάλληλος για τα διάφορα περιβάλλοντα, τεχνολογίες και επικοινωνιακές ανάγκες. Τα συστήματα αυτά εξελίχθηκαν από απλές αναλογικές συσκευές που μετέφεραν μόνο ήχο, σε σύνθετες, που είναι διασυνδεμένες σε πολλές πλατφόρμες δικτυακά. Αυτοί οι τύποι μπορούν κατηγοριοποιηθούν ως εξής.

1.3.1 Καλωδιωμένα Συστήματα Θυροτηλεφώνου

Ίσως ο πιο διαδεδομένος και παραδοσιακός τύπος σε οικιακό, εμπορικό και βιομηχανικό σκηνικό. Αυτά τα συστήματα εκμεταλλεύονται ειδική καλωδίωση για να διευκολύνουν την μετάδοση ήχου ίσως και βίντεο ανάμεσα σε επικοινωνιακά τερματικά, συνήθως ανάμεσα σε κάποιο πάνελ εισόδου και ενός ή περισσότερων δεκτών. Η ειδική αυτή καλωδίωση μπορεί να είναι :

1. Ζεύγος συνεστραμμένων καλωδίων(πχ. Cat5e/Cat6), χρησιμοποιούνται συχνά σε σύγχρονες εγκαταστάσεις, ειδικά σε ψηφιακά ή σε συστήματα που βασίζονται σε δίκτυο, λόγω της ιδιότητάς τους να χειρίζονται υψηλότερα εύρη ζώνης και διάφορους τύπους σημάτων.
2. Ομοαξονικά καλώδια, χρησιμοποιήθηκαν σε συστήματα θυροτηλεφώνου με βίντεο για την μεταφορά αναλογικών σημάτων βίντεο.
3. Πολύκλωνα καλώδια τα οποία χρησιμοποιούνται σε αναλογικά συστήματα για να μεταφέρουν ισχύ,ήχο και σήματα ελέγχου.

Ο λόγος που προτιμούνται τα καλωδιωμένα συστήματα είναι πως προσφέρουν προστασία από ηλεκτρομαγνητικές παρεμβολές, απώλεια σήματος και ανεξουσιοδότητη πρόσβαση. Παρόλα αυτά έρχονται και με κάποιους περιορισμούς όπως η εγκατάσταση τους η οποία πρέπει να προγραμματιστεί κατά το χτίσιμο του κτιρίου αυξάνοντας το κόστος, ειδικά σε περίπτωση που η εγκατάσταση γίνει μεταγενέστερα. Επίσης όταν η εγκατάσταση ολοκληρωθεί τα καλώδια δεν μπορούν να μεταφερθούν και η επέκταση του συστήματος καθίσταται δύσκολη.

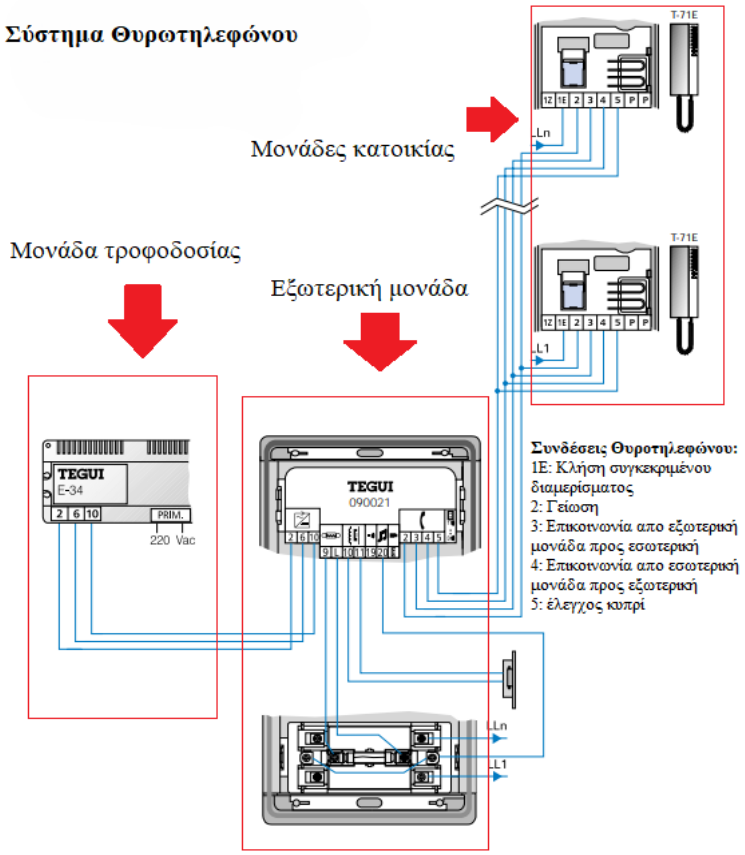
Σε αυτή την κατηγορία συστημάτων ανήκουν τα συστήματα δύο καλωδίων, συστήματα τεσσάρων καλωδίων και συστήματα τεσσάρων+N (4+N) όπου τα τελευταία είναι και τα πιο διαδεδομένα στην ελλάδα.

- Το σύστημα δύο καλωδίων είναι ένα επικοινωνιακό σύστημα όπου τροφοδοσία και σήματα ήχου μεταδίδονται σε δύο ίδιους αγωγούς (συνήθως σε ένα ζευγάρι στριμμένων καλωδίων,όπως τα καλώδια ηχείων). Αυτά τα συστήματα είναι συνήθως αναλογικά, απλά και ιδανικά για μετάδοση ενός μηνύματος από ένα σταθμό σε πολλούς (π.χ. σχολεία,εργοστάσια ή μικρά κτίρια). Η κατασκευή και εγκατάσταση του είναι πολύ εύκολη και με σχετικά χαμηλό κόστος χωρίς να απαιτεί πολύπλοκη υποδομή. Επίσης τα συστήματα αυτά μπορούν να λειτουργήσουν και με μερική ζημιά. Παρόλα αυτά όμως αυτά υποστηρίζουν broadcast only (μονόπλευρη μετάδοση) ή Push-to-talk (πίεσε για να μιλήσεις), σε μεγάλα καλώδια μειώνεται η ποιότητα του ήχου και είναι δύσκολα στην διαχείριση όταν εμπλέκονται πολλοί χρήστες.

- Στην περίπτωση των συστημάτων τεσσάρων καλωδίων, δυο καλώδια χρησιμοποιούνται για την μετάδοση ήχου και τα άλλα δύο για την λήψη, εξού και η ονομασία. Αυτός ο διαχωρισμός επιτρέπει την αμφίδρομη επικοινωνία σε σχέση με το σύστημα δύο καλωδίων. Διαθέτει πολύ καλή ποιότητα ήχου και μπορεί να ενσωματωθεί σε πολύπλοκα δίκτυα επικοινωνιών πράγμα που το κάνει ευέλικτο. Στην αρνητική πλευρά, το κόστος είναι υψηλότερο,απαιτεί περισσότερη υποδομή,η τροφοδοσία είναι ξεχωριστή και η εγκατάσταση δεν είναι τόσο απλή.

- Τα συστήματα 4+N όπως αναφέρθηκε και προηγουμένως είναι και τα πιο διαδεδομένα στην Ελλάδα. Εφαρμόζεται σε αναλογικά θυροτηλέφωνα. Το «4+N» αναφέρεται στα 4 βασικά καλώδια που μοιράζονται όλα τα διαμερίσματα και στα «N» καλώδια που αντιστοιχούν στο κάθε διαμέρισμα ξεχωριστά (N=αριθμός διαμερισμάτων). Τυπικά τα 4 αυτά καλώδια εξυπηρετούν τις παρακάτω λειτουργίες αντίστοιχα :

  1. Γραμμή ηχείου που μεταφέρει τα σήματα ήχου από το εξωτερικό πάνελ (επισκέπτης) στον εσωτερικό πάνελ-τηλέφωνο.
  2. Γραμμή μικροφώνου που μεταφέρει τον ήχο από το διαμέρισμα στο πάνελ εισόδου.
  3. Γραμμή τροφοδοσίας, η οποία τροφοδοτεί το σύστημα ρεύμα.
  4. Γραμμή ξεκλειδώματος πόρτας η οποία ενεργοποιεί την ηλεκτρική κλειδαριά επιτρέποντας έτσι στον κάτοικο να ανοίξει την πόρτα απομακρυσμένα.



Εικόνα 1-5 Σύστημα θυροτηλεφώνου 4+N.

1.3.2 Ασύρματα Συστήματα Θυροτηλεφώνου

Τα ασύρματα συστήματα θυροτηλεφώνου, αντιπροσωπεύουν την μοντέρνα εξέλιξη του κλασικού, αναλογικού. Προσφέρουν ευελιξία και μειώνουν την ανάγκη για μεγάλες υποδομές. Τα συστήματα αυτά εκμεταλλεύονται ασύρματα επικοινωνιακά πρωτόκολλα για να μεταδώσουν ήχο ή και βίντεο, μεταξύ ενός κεντρικού πίνακα (συνήθως στην είσοδο) και άλλων τερματικών σταθμών (οθόνη στην οικία, κινητά κλπ.).

Αυτά, χρησιμοποιούν διάφορες τεχνολογίες βάση της σχεδίασης τους και της εφαρμογής τους. Οι πιο συνηθισμένες είναι:

- Ραδιοσυχνότητες: Αυτά τα συστήματα συνήθως λειτουργούν σε μη αδειοδοτημένες ζώνες συχνοτήτων (π.χ. 433MHz ή 900 MHz) προσφέροντας επικοινωνία από σημείο σε σημείο ή σε πολλά σημεία. Παρότι δεν χρησιμοποιούνται τόσο σε σενάρια εισόδου-διαμερίσματος (χρησιμοποιούνται πιο συχνά για επικοινωνία εσωτερική) είναι σημαντικό να αναφερθούν.

- Wi-Fi (IEEE 802.11): Ίσως η πιο διαδεδομένη τεχνολογία ασύρματης επικοινωνίας σε μικρή εμβέλεια, είναι τα τελευταία χρόνια το Wi-Fi. Καθώς, κατά την πλειοψηφία κάθε σπίτι πλέον διαθέτει Wi-Fi, τα συστήματα αυτά αξιοποιούν αυτά τα υπάρχοντα ασύρματα δίκτυα για να επιτρέπουν την σε πραγματικό χρόνο μετάδοση βίντεο, ήχου και απομακρυσμένη πρόσβαση από κινητό τηλέφωνο.

- Bluetooth: Λόγω της μικρής εμβέλειας του οι εφαρμογές για κεντρική είσοδο με διαμέρισμα δεν είναι βιώσιμες. Χρησιμοποιείται όμως σε περιπτώσεις επικοινωνίας μεταξύ μικρών και κοντινών δωματίων.

- Δίκτυα κινητής τηλεφωνίας: Ορισμένα προηγμένα συστήματα ενσωματώνουν κάρτες SIM και χρησιμοποιούν δίκτυα δεδομένων για να προσφέρουν απομακρυσμένη επικοινωνία ακόμη και σε τεράστιες αποστάσεις.

Ενώ τα ασύρματα συστήματα θυροτηλεφώνων προσφέρουν ευκολία εγκατάστασης, φορητότητα σε περίπτωση αλλαγής της θέσης τους και απομακρυσμένη πρόσβαση, υπόκεινται σε περιορισμούς όπως:

1. Παρεμβολές σήματος που δημιουργούν φυσικά εμπόδια όπως κτήρια (τσιμεντένιοι τοίχοι, μεταλλικές κατασκευές) ή άλλες ηλεκτρονικές συσκευές που λειτουργούν σε κοντινές συχνότητες.
2. Προβλήματα ασφαλείας, σε περίπτωση που δεν εφαρμοστούν τα κατάλληλα πρωτόκολλα ασφαλείας.
3. Εξάρτηση στην ποιότητα του εκάστοτε δικτύου. Σε περίπτωση κακής ποιότητας μπορεί να δημιουργηθεί lag(καθυστέρηση, χαμηλή ποιότητα ήχου και βίντεο ακόμη και διακοπές.

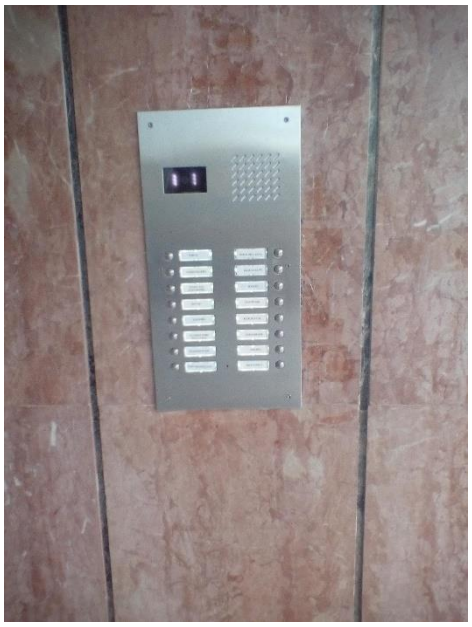
Βάση της μελέτης που διεξήγαμε, θεωρούμε πως το ιδανικότερο πρωτόκολλο ασύρματης επικοινωνίας είναι το Wi-Fi το οποίο και ενσωματώσαμε στην κατασκευή μας.

Εικόνα 1-6:Ασύρματο Θυροτηλέφωνο EP7 της εταιρείας Ezviz.

1.4 Τρόπος Λειτουργίας και Δομή Κλασσικού Θυροτηλεφώνου

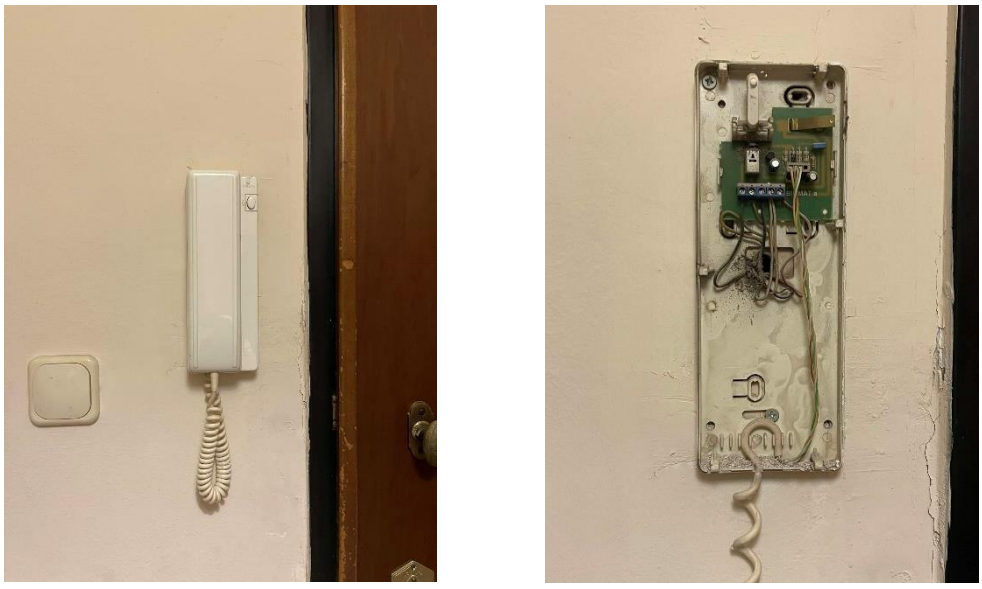
Καθώς υπάρχουν αμέτρητες συσκεύες θυροτηλεφώνων, με τεράστιες διαφορές ανάμεσα τους ως προς τις λειτουργίες τους, τον τρόπο σύνδεσής τους κ.α, θα αναφερθούμε στον τρόπο λειτουργίας και την δομή των πιο διαδεδομένων συσκευών στην Ελλάδα. Οι συσκευές αυτές αποτελλούνται από τα εξής βασικά στοιχεία:

1. Εξωτερική μονάδα: Αυτή τοποθετείται στην είσοδο του κτιρίου σε κουτί εντοιχισμού το οποίο συνήθως κατασκευάζεται από πλαστικό. Διαθέτει τα κουμπιά κλήσης που αντιστοιχούν στο κάθε διαμέρισμα με το όνομα δίπλα και σε κάποιες περιπτώσεις είναι και τα δύο μαζί. Επίσης περιέχει ένα μικρόφωνο και ηχείο για την αμφίδρομη επικοινωνία των δύο μονάδων τα οποία προστατεύονται από τις καιρικές συνθήκες συνήθως με κάποια ειδική μεμβράνη. Όλα τα προηγούμενα μαζί με τα ηλεκτρονικά κυκλώματα που διαθέτει για την ενίσχυση ήχου του μικροφώνου και του ηχείου και την διαχείριση της επικοινωνίας προστατεύονται από ένα πλαίσιο ή κάλυμμα, συνήθως κατασκευασμένο από αλουμίνιο, ανοξείδωτο ατσάλι ή πλαστικό υψηλής αντοχής, το οποίο τα προστατεύει από καιρικές συνθήκες, βανδαλισμούς και φθορές.



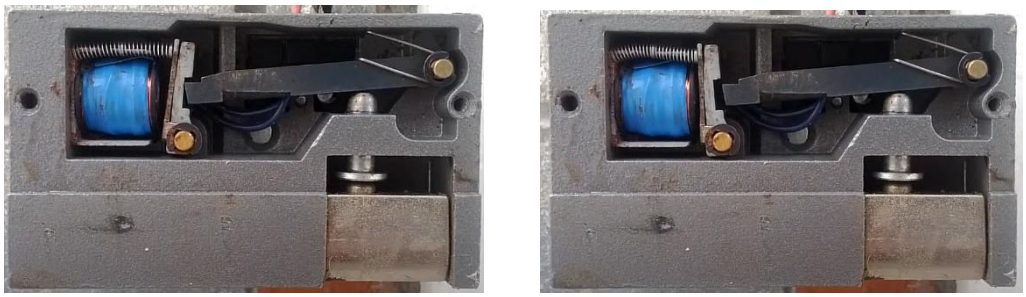
Εικόνα 1-7: Εξωτερική μονάδα θυροτηλεφώνου.

2. Εσωτερική Μονάδα(Θυροτηλέφωνο): Το πρώτο μέρος της μονάδας αυτής, είναι το ακουστικό (handset). Μέσα σε αυτό βρίσκονται το μικρόφωνο και το ηχείο. Το μικρόφωνο συνήθως είναι πυκνωτικό και το ακουστικό τύπου μεμβράνης. Το δεύτερο μέρος αυτής της μονάδας είναι η βάση. Σε αυτήν, ενσωματώνεται ένα ηλεκτρονικό κύκλωμα για τη σύνδεση καλωδίων και την διαχείριση των λειτουργιών. Στο ίδιο κύκλωμα βρίσκεται το buzzer (Βομβητής) το οποίο ενεργοποιείται όταν πατηθεί το κουμπί του αντίστοιχου διαμερίσματος και συνήθως είναι πιεζοηλεκτρικό. Πάνω στην βάση, βρίσκεται επίσης το κουμπί ανοίγματος πόρτας που όταν πατηθεί κλείνει το κύκλωμα που ανοίγει την πόρτα.



Εικόνα 1-8: Εσωτερική μονάδα θυροτηλεφώνου.

3. Ηλεκτρομαγνητική κλειδαριά (Κυπρί): Τοποθετημένη στην κάσα της πόρτας επιτρέπει το απομακρυσμένο άνοιγμα της πόρτας με το πάτημα του κουμπιού στην εσωτερική μονάδα. Με το πάτημα λοιπόν του κουμπιού, εφαρμόζεται μια τάση (συνήθως 12-24 V AC/DC) ενεργοποιόντας ενα πηνίο. Το πηνίο έλκει έναν ηλεκτρομαγνητικό μοχλό ή κλείστρο, επιτρέποντας έτσι το άνοιγμα της πόρτας με ένα απλό σπρώξιμο. Στην Ελλάδα εφαρμόζεται κατα κύριο λόγω τάση 12V AC και για αυτό ακούμε κατά το άνοιγμα της εισόδου έναν έντονο βόμβο σε αντίθεση με το 12V DC που ακούγεται ένας χαμηλότονος ήχος κλικ. Υπάρχουν δύο βασικές κατηγορίες τέτοιων κλειδαριών: Η Fail-secure και η Fail-safe. Η πρώτη, είναι κλειδωμένη όταν δεν της εφαρμόζεται τάση και ανοίγει μόνο με ρεύμα ενώ η δεύτερη το ακριβώς αντίθετο, είναι ανοιχτή όταν δεν έχει τάση και κλειδώνει μόνο με ρεύμα. Η Fail-secure τύποι χρησιμοποιούνται κατα κόρον στην Ελλάδα διότι προσφέρει ασφάλεια ακόμα και σε διακοπή ρεύματος.



Εικόνα 1-9: Κυπρί σε κατάσταση κλειστού (αριστερά) και ανοιχτού (δεξιά).

4. Τροφοδοτικό: Παρέχει την απαιτούμενη τάση για την λειτουργία όλου του συστήματος. Δέχεται σαν είσοδο 230V AC και είτε μετατρέπει σε DC ή μετασχηματίζει την τάση σε χαμηλότερη AC ή και τα δυο ανάλογα με τις ανάγκες του κάθε συστήματος. Κατά κύριο λόγο χρησιμοποιούνται τροφοδοτικά που προσφέρουν DC τάση για την εσωτερική και εξωτερική μονάδα και AC για το κυπρί. Συνήθως τοποθετούνται στον ηλεκτρολογικό πίνακα της πολυκατοικίας ή στο υπόγειο. Ταυτόχρονα, είτε ενσωματωμένα είτε εκτός συνδέεται ένα ρελέ για την προστασία του κυκλώματος.

Εικόνα 1-10:Ενδεικτικό τροφοδοτικό συστήματος θυροτηλεφώνου.

Ανακεφαλαιώνοντας, το κλασσικό σύστημα θυροτηλεφώνου λειτουργεί ως εξής. Ο επισκέπτης από την εξωτερική μονάδα πατάει το κουμπί που αντιστοιχεί στο διαμέρισμα που τον ενδιαφέρει και ηχεί από το buzzer του τηλεφώνου το κουδούνι. Ο κάτοικος σηκώνει το ακουστικό (headset) στην εσωτερική μονάδα και μπορεί να ακούσει την φωνή του επισκέπτη ενώ ταυτόχρονα μιλάει μέσω του μικροφώνου. Ο ήχος μεταδίδεται μέσω των καλωδίων αμφίδρομα οπότε το ίδιο ισχύει και για τον επισκέπτη. Αν ο κάτοικος θέλει να επιτρέψει την είσοδο στον επισκέπτη, το μόνο που έχει να κάνει είναι να πατήσει το κουμπί πάνω στην βάση το οποίο ενεργοποιεί την ηλεκτρομαγνητική κλειδαριά (κυπρί) στην είσοδο η οποία στιγμιαία επιτρέπει το άνοιγμα της πόρτας για τον επισκέπτη.

1.5 Τρόπος Λειτουργίας και Δομή Μοντέρνου Θυροτηλεφώνου

Τα μοντέρνα θυροτηλέφωνα συνδυάζουν όλες τις λειτουργίες ενός κλασσικού θυροτηλεφώνου με νέες τεχνολογίες που τα αναβαθμίζουν σε προηγμένες μορφές επικοινωνίας και ελέγχου πρόσβασης. Η δομή τους παραμένει σχεδόν ίδια με τα κλασσικά, δηλαδή μια μονάδα στην είσοδο,μια στο διαμέρισμα και το κυπρί της πόρτας. Στην εξωτερική μονάδα, το μικρόφωνο, το ηχείο και το κουμπί κλήσης είναι αυτά που παραμένουν. Όμως σε αυτή συνήθως προστίθενται, μια κάμερα για να παρέχει εικόνα του επισκέπτη και ακόμα και να καταγράφει την κεντρική είσοδο. Επίσης κατά βάση εγκαθίστανται πληκτρολόγιο για την είσοδο με PIN ή RFID reader που προσφέρει πρόσβαση με κάρτα. Στην εσωτερική μονάδα εκτός των κλασσικών λειτουργιών, υπάρχει οθόνη η οποία εμφανίζει την εικόνα που καταγράφει η εξωτερική κάμερα. Η επικοινωνία συνήθως πραγματοποιείται μέσω Wi-Fi προσφέροντας ευκολότερη εγκατάσταση ή καλώδιο χρησιμοποιώντας έτσι,την υφιστάμενη καλωδίωση των κτηρίων. Πιο συγκεκριμένο παράδειγμα οι συσκευές HikVision DS-KH6320-WTE1 (Εσωτερική μονάδα) και η HikVision DS-KD8003-IME1 (Εξωτερική μονάδα). Πιο συγκεκριμένα η Εσωτερική μονάδα (HikVision DS-KH6320-WTE1) είναι μια οθόνη αφής 7 ιντσών με ανάλυση 1024x600 απο την οποία μπορεί να ρυθμιστεί όλο το σύστημα. Υποστηρίζει SIP και VOIP επικοινωνία. Ταυτόχρονα, μπορεί να συνδεθεί μέσω Wi-Fi ή ενσύρματα και διαθέτει εφαρμογή ελέγχου για την παρακολούθηση σε πραγματικό χρόνο της κάμερας και την λήψη ειδοποιήσεων. Η εξωτερική μονάδα (HikVision DS-KD8003-IME1) διαθέτει μια κάμερα 2MP και όλες τις άλλες λειτουργίες ενός κλασσικού θυροτηλεφώνου.

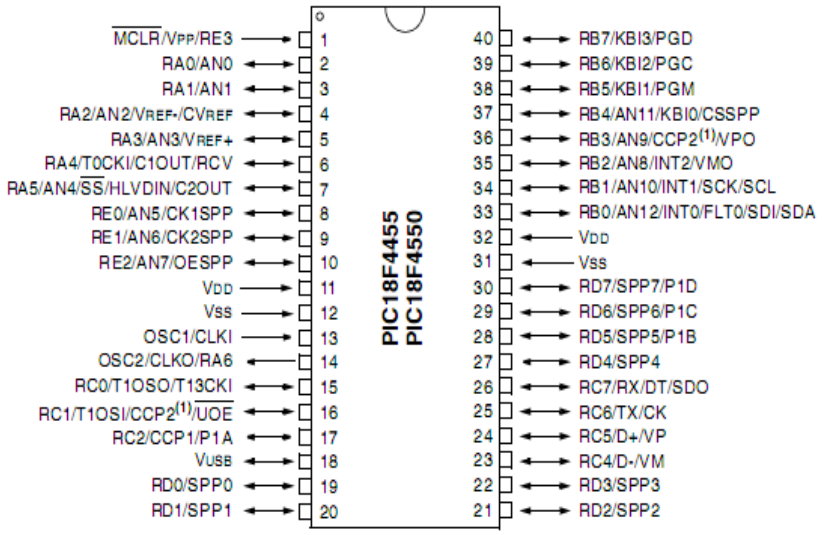
Κεφάλαιο 2ο: Μικροελεγκτές

2.1 Εισαγωγή

2.1.1 Ορισμός Μικροελεγκτή

Ο μικροελεγκτής (microcontroller) είναι ένα προγραμματιζόμενο ολοκληρωμένο κύκλωμα το οποίο ενσωματώνει σε ένα και μόνο chip τον επεξεργαστή (CPU), μονάδες μνήμης (RAM και ROM ή Flash), καθώς και περιφερειακά κυκλώματα εισόδου/εξόδου (I/O).

Η κύρια λειτουργία του είναι η αυτόνομη εκτέλεση εντολών, οι οποίες είναι αποθηκευμένες στη μνήμη προγράμματος, με στόχο τον έλεγχο εξωτερικών ηλεκτρονικών συστημάτων ή συσκευών.



Εικόνα 2-1: Μικροελεγκτής PIC18F4455/PIC18F4550.

Βασικά Χαρακτηριστικά είναι:

- Ενσωμάτωση πολλαπλών λειτουργιών σε ένα chip: CPU, μνήμη, I/O και περιφερειακά κυκλώματα βρίσκονται ενσωματωμένα σε ένα και μόνο ολοκληρωμένο κύκλωμα.
- Προγραμματιζόμενη λειτουργία: Ο μικροελεγκτής εκτελεί προγράμματα γραμμένα σε γλώσσες όπως C, C++ ή Assembly.
- Αλληλεπίδραση με τον φυσικό κόσμο: Μέσω των θυρών εισόδου/εξόδου, μπορεί να συνδεθεί με αισθητήρες, κινητήρες, LEDs, κουμπιά και άλλες συσκευές.

- Αυτόνομη λειτουργία: Δεν απαιτείται εξωτερικό λειτουργικό σύστημα· η εκτέλεση ξεκινά μόλις τροφοδοτηθεί με ρεύμα.

2.1.2 Συγκριτική Αναφορά με Μικροϋπολογιστές

Ο μικροελεγκτής μπορεί να θεωρηθεί ως μια εξειδικευμένη μορφή μικροϋπολογιστή, με τη διαφορά ότι είναι σχεδιασμένος για ειδικό σκοπό και όχι για γενική χρήση. Σε αντίθεση με έναν μικροϋπολογιστή, ο οποίος εκτελεί διαφορετικά προγράμματα και συνδέεται με πλήθος εξωτερικών συσκευών, ο μικροελεγκτής είναι βιομηχανικά προσανατολισμένος σε συγκεκριμένες, επαναλαμβανόμενες λειτουργίες.

2.1.3 Εφαρμογές Μικροελεγκτών

Οι μικροελεγκτές χρησιμοποιούνται σε πλήθος τεχνολογικών και βιομηχανικών εφαρμογών, εξαιτίας της ικανότητάς τους να επιτελούν στοχευμένες, επαναλαμβανόμενες και αυτόνομες λειτουργίες με ακρίβεια και χαμηλό ενεργειακό κόστος.

2.1.4 Κύριοι Τομείς Εφαρμογής

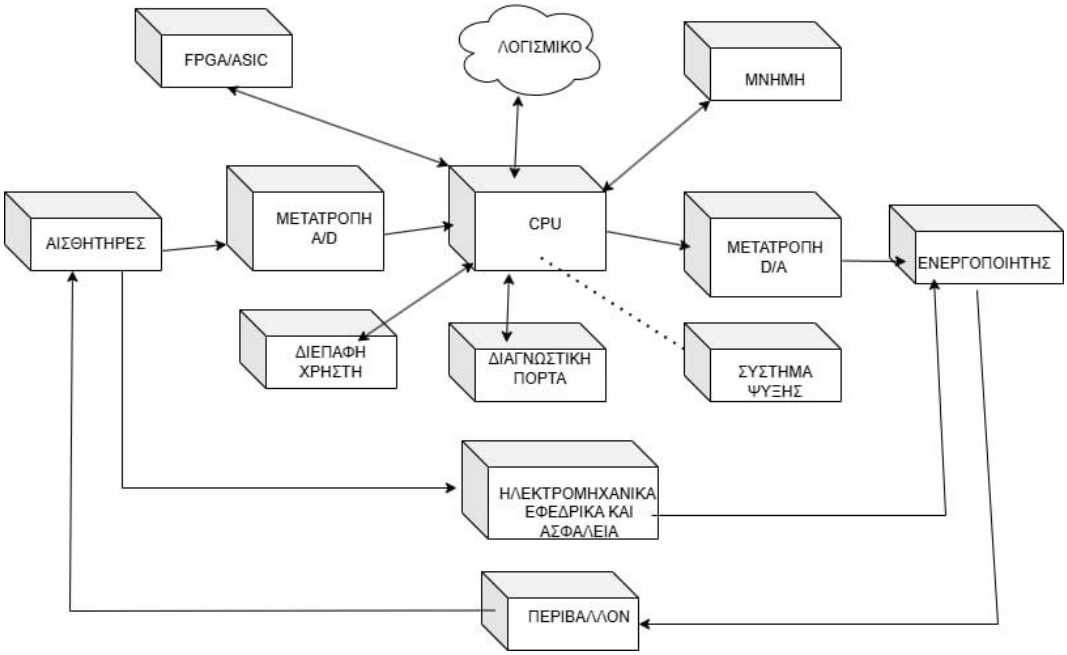
Οι σημαντικότεροι τομείς όπου αξιοποιούνται μικροελεγκτές περιλαμβάνουν:

- Συστήματα αυτοματισμού: Χρησιμοποιούνται για τον έλεγχο βιομηχανικών διεργασιών, γραμμών παραγωγής και αυτοματοποιημένων μηχανών.
- Τηλεπικοινωνιακά συστήματα: Συμμετέχουν στη διαχείριση πρωτοκόλλων επικοινωνίας, στη μετάδοση δεδομένων και στον έλεγχο συσκευών δικτύωσης.
- Ηλεκτρονικές συσκευές: Από τηλεοράσεις και χειριστήρια έως φορητές συσκευές και οικιακά gadgets.
- Ηλεκτρικές οικιακές συσκευές: Ρυθμίζουν και ελέγχουν λειτουργίες σε πλυντήρια, φούρνους μικροκυμάτων, ψυγεία κ.λπ.
- Συστήματα τηλεματικής: Εφαρμόζονται σε οχήματα για την επεξεργασία γεωγραφικών δεδομένων, πλοήγηση και τηλεπικοινωνιακές υπηρεσίες.
- Συστήματα συλλογής δεδομένων (Data Acquisition Systems): Καταγράφουν φυσικά μεγέθη μέσω αισθητήρων και μεταφέρουν τα δεδομένα για επεξεργασία.
- Ηλεκτρονικά ισχύος: Ρυθμίζουν την ενέργεια σε κυκλώματα μετατροπής ισχύος, όπως φορτιστές και μετατροπείς τάσης.

- Συστήματα διασύνδεσης (Interfaces): Ενσωματώνονται ως γέφυρες επικοινωνίας μεταξύ διαφορετικών κυκλωμάτων ή ψηφιακών συσκευών.
- Δίκτυα επικοινωνίας: Συμμετέχουν στον έλεγχο και την αποστολή δεδομένων σε ενσύρματα και ασύρματα δίκτυα.

2.1.5 Ενσωματωμένα Συστήματα (Embedded Systems)

Οι μικροελεγκτές αποτελούν βασικό συστατικό των ενσωματωμένων συστημάτων (embedded systems), τα οποία είναι ειδικά σχεδιασμένα για να εκτελούν συγκεκριμένες λειτουργίες. Σε αυτά περιλαμβάνονται συστήματα που βασίζονται σε μικροεπεξεργαστές, FPGA ή DSP, ανάλογα με την πολυπλοκότητα και τις απαιτήσεις της εφαρμογής.



Εικόνα 2-2:Διάγραμμα ροής τυπικού ενσωματωμένου συστήματος.

2.1.6 Πλεονεκτήματα και Μειονεκτήματα των Γλωσσών Χαμηλού Επιπέδου στον Προγραμματισμό Μικροελεγκτών

Οι γλώσσες χαμηλού επιπέδου, όπως η συμβολική γλώσσα (assembly), χρησιμοποιούνται ευρέως στον προγραμματισμό μικροελεγκτών, ιδιαίτερα σε εφαρμογές που απαιτούν ακριβή χρονισμό και άμεσο έλεγχο του υλικού. Ωστόσο, η επιλογή αυτής της κατηγορίας γλωσσών συνοδεύεται τόσο από πλεονεκτήματα όσο και από μειονεκτήματα.

Πλεονεκτήματα:

- Απόλυτος έλεγχος υλικού: Ο χρήστης μπορεί να διαχειριστεί με ακρίβεια κάθε στοιχείο του μικροελεγκτή, γεγονός που επιτρέπει τον βελτιστοποιημένο έλεγχο των λειτουργιών του συστήματος.
- Ακριβής διαχείριση χρονισμού: Είναι δυνατή η ρύθμιση χρονισμών με μεγάλη ακρίβεια, απαραίτητο σε εφαρμογές πραγματικού χρόνου.
- Χαμηλό κόστος εργαλείων: Οι assemblers διατίθενται συνήθως δωρεάν από τους κατασκευαστές μικροελεγκτών, καθιστώντας την ανάπτυξη οικονομικά προσιτή.

Μειονεκτήματα:

- Αυξημένη πολυπλοκότητα μάθησης: Η εκμάθηση της assembly απαιτεί εξοικείωση με την αρχιτεκτονική κάθε μικροελεγκτή, γεγονός που την καθιστά δυσκολότερη για αρχάριους προγραμματιστές.
- Μειωμένη αναγνωσιμότητα κώδικα: Τα προγράμματα σε συμβολική γλώσσα είναι δύσκολα στην ανάγνωση και κατανόηση, γεγονός που δυσχεραίνει τη συντήρηση και την τροποποίηση τους με την πάροδο του χρόνου.
- Περιορισμένη δυνατότητα συνεργασίας: Η συνεργασία πολλών προγραμματιστών στο ίδιο έργο γίνεται πιο δύσκολη, λόγω της έλλειψης ευανάγνωστης και δομημένης σύνταξης.

2.1.7 Πλεονεκτήματα και Μειονεκτήματα των Γλωσσών Υψηλού Επιπέδου στον Προγραμματισμό Μικροελεγκτών

Οι γλώσσες υψηλού επιπέδου, όπως οι C, C++, και Python, χρησιμοποιούνται συχνά για τον προγραμματισμό μικροελεγκτών, κυρίως λόγω της μεγαλύτερης ευχρηστίας και του αυξημένου επιπέδου αφαίρεσης που προσφέρουν. Παρόλα αυτά, η χρήση αυτών των γλωσσών συνοδεύεται από πλεονεκτήματα και περιορισμούς που πρέπει να ληφθούν υπόψη κατά την ανάπτυξη ενσωματωμένων συστημάτων.

Πλεονεκτήματα:

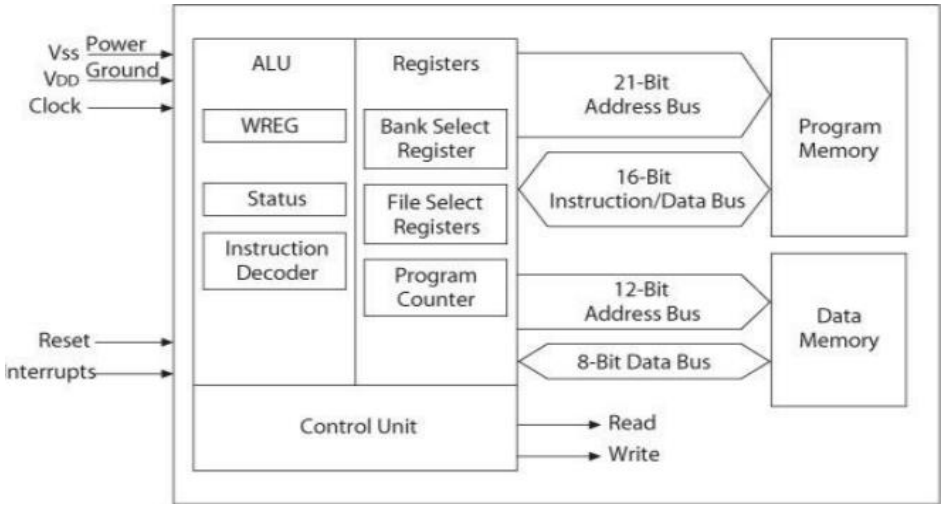
- Ευκολότερη ανάπτυξη σύνθετων προγραμμάτων: Οι γλώσσες υψηλού επιπέδου επιτρέπουν την εύκολη ανάπτυξη μεγάλων και πολύπλοκων εφαρμογών με λιγότερη γραφή κώδικα, καθιστώντας την ανάπτυξη ταχύτερη και πιο προσβάσιμη.
- Εύκολη συνεργασία πολλών προγραμματιστών: Η δομημένη σύνταξη και η υψηλότερη αφαίρεση του κώδικα διευκολύνουν τη συνεργασία πολλών προγραμματιστών σε ένα έργο, επιτρέποντας την ομαλή κατανομή εργασιών.

Μειονεκτήματα:

- Δυσκολία συγγραφής κώδικα για εφαρμογές με κρίσιμους χρονισμούς: Σε εφαρμογές που απαιτούν ακριβή χρονισμό και αντιμετώπιση πραγματικού χρόνου, ο προγραμματισμός με γλώσσες υψηλού επιπέδου μπορεί να είναι πιο δύσκολος, καθώς οι αλγόριθμοι που παράγονται δεν είναι τόσο «άμεσοι» όσο με γλώσσες χαμηλού επιπέδου.
- Κόστος του compiler: Αν και ορισμένοι compilers για γλώσσες υψηλού επιπέδου είναι δωρεάν, άλλοι μπορεί να απαιτούν την αγορά, προσθέτοντας ένα επιπλέον κόστος στη διαδικασία ανάπτυξης.
- Μειωμένη βελτιστοποίηση του κώδικα: Στους παλαιότερους compilers, ο παραγόμενος κώδικας μηχανής δεν ήταν πάντοτε βελτιστοποιημένος, με αποτέλεσμα να απαιτούνταν μικροελεγκτές με μεγαλύτερη μνήμη. Ωστόσο, οι σύγχρονοι compilers διαθέτουν εξελιγμένα εργαλεία βελτιστοποίησης (optimization), τα οποία έχουν βελτιώσει σημαντικά την αποδοτικότητα και την αποτελεσματικότητα του παραγόμενου κώδικα.

2.1.8 Βασική Λειτουργική Διαδικασία Μικροελεγκτή

1. Προγραμματισμός: Το πρόγραμμα γράφεται σε μια γλώσσα προγραμματισμού και στη συνέχεια φορτώνεται στη μνήμη ROM του μικροελεγκτή. Η μνήμη ROM αποθηκεύει το πρόγραμμα μόνιμα, επιτρέποντας στον μικροελεγκτή να εκτελεί τις προκαθορισμένες εντολές.



Εικόνα 2-3:Διάγραμμα αρχιτεκτονικής ενός μικροελεγκτή.

2. Εκτέλεση: Η CPU του μικροελεγκτή διαβάζει τις εντολές του προγράμματος από τη μνήμη ROM και εκτελεί συγκεκριμένες ενέργειες ή υπολογισμούς. Για παράδειγμα, μπορεί να ενεργοποιήσει ένα φως ή να μετρήσει τη θερμοκρασία, αναλόγως με τη λειτουργία του προγράμματος.

3. Αλληλεπίδραση: Χρησιμοποιώντας περιφερειακά εισόδου/εξόδου (I/O), ο μικροελεγκτής επικοινωνεί με το εξωτερικό περιβάλλον. Αλληλεπιδρά με αισθητήρες για να συλλέξει δεδομένα και στέλνει σήματα στους ενεργοποιητές για να εκτελέσει ενέργειες, όπως η ενεργοποίηση ενός κινητήρα ή η αλλαγή της κατάστασης μιας συσκευής.

2.1.9 Τύποι Μικροελεγκτών

Υπάρχουν διάφοροι τύποι μικροελεγκτών, οι οποίοι διαφέρουν ανάλογα με τις ανάγκες των εφαρμογών και την πολυπλοκότητα των εργασιών που πρέπει να εκτελούν.

1. Μικροελεγκτές 8 bit: Αυτοί οι μικροελεγκτές είναι οι πιο απλοί και συνήθως χρησιμοποιούνται σε απλές εφαρμογές, όπως τηλεχειριστήρια, ηλεκτρονικά παιχνίδια και μικρές συσκευές. Διαθέτουν επεξεργαστή που χειρίζεται 8 bit δεδομένων κάθε φορά, κάτι που τους καθιστά οικονομικούς και αποτελεσματικούς για εργασίες που δεν απαιτούν μεγάλη υπολογιστική ισχύ.

2. Μικροελεγκτές 16 bit: Οι μικροελεγκτές 16 bit προσφέρουν μεγαλύτερη υπολογιστική ισχύ και ικανότητα επεξεργασίας σε σχέση με τους 8 bit. Αυτοί χρησιμοποιούνται σε πιο σύνθετες εφαρμογές, όπως συστήματα ήχου, έλεγχος συσκευών και βιομηχανικοί αυτοματισμοί. Ο επεξεργαστής τους χειρίζεται 16 bit δεδομένων κάθε φορά, επιτρέποντας μεγαλύτερη ακρίβεια και ταχύτητα εκτέλεσης των εντολών.
3. Μικροελεγκτές 32 bit: Αυτοί είναι οι πιο προηγμένοι και ισχυροί μικροελεγκτές, κατάλληλοι για απαιτητικές εφαρμογές που χρειάζονται εντατική επεξεργασία δεδομένων. Χρησιμοποιούνται σε συστήματα αυτοκινήτων, συσκευές IoT (Internet of Things) και άλλες τεχνολογίες που απαιτούν υψηλή απόδοση. Οι 32 bit μικροελεγκτές μπορούν να επεξεργαστούν 32 bit δεδομένων ταυτόχρονα, προσφέροντας σημαντική αύξηση στην απόδοση και τη δυνατότητα multitasking.
4. Ειδικοί Μικροελεγκτές Εφαρμογής: Αυτοί οι μικροελεγκτές είναι σχεδιασμένοι για συγκεκριμένες εφαρμογές, με ενσωματωμένες λειτουργίες που τους καθιστούν ιδανικούς για εργασίες όπως ο έλεγχος οθονών αφής ή η λειτουργία ιατρικών συσκευών. Οι ειδικοί μικροελεγκτές προσφέρουν βελτιστοποιημένες δυνατότητες για συγκεκριμένες ανάγκες, προσφέροντας μικρότερη κατανάλωση ενέργειας και καλύτερη απόδοση σε εξειδικευμένα περιβάλλοντα.

2.2 Μικροελεγκτής ESP32

Ο ESP32 είναι μια οικογένεια μικροελεγκτών χαμηλού κόστους και ενεργειακής αποδοτικότητας που ενσωματώνει τόσο δυνατότητες Wi-Fi όσο και Bluetooth, καθιστώντας τον ιδανικό για εφαρμογές ασύρματης επικοινωνίας. Ο ESP32 χρησιμοποιείται κυρίως σε συσκευές Internet of Things (IoT), έξυπνα οικιακά προϊόντα και άλλα ενσωματωμένα συστήματα.

Εικόνα 2-4:Μικροελεγκτής ESP32.

2.2.1 Κύρια Χαρακτηριστικά του ESP32

1. Επεξεργαστικές Επιλογές Dual-Core και Single-Core: Ο ESP32 προσφέρει διάφορες διαμορφώσεις επεξεργαστή:

    - Tensilica Xtensa LX6: Διατίθεται σε εκδόσεις dual-core και single-core.
    - Tensilica Xtensa LX7: Dual-core επεξεργαστής, προσφέροντας μεγαλύτερη απόδοση και αποδοτικότητα.
    - RISC-V:Επιλογή single-core επεξεργαστή.

2. Ενσωματωμένη Ασύρματη Επικοινωνία: Ο ESP32 υποστηρίζει τόσο Wi-Fi όσο και Bluetooth, επιτρέποντας την ασύρματη επικοινωνία σε δίκτυα, κάτι που είναι απαραίτητο για εφαρμογές IoT.

3. Ενσωματωμένα Στοιχεία Ασύρματης Επικοινωνίας: Περιλαμβάνει διάφορα σημαντικά στοιχεία για την ασύρματη επικοινωνία, όπως:

    - Διακόπτες Κεραιών
    - RF Balun (Radio Frequency Balun)
    - Ενισχυτές Ισχύος

- Δέκτες Χαμηλού Θορύβου
- Φίλτρα
- Μονάδες Διαχείρισης Ικανότητας

4. Ενεργειακή Απόδοση: Ένα από τα κύρια πλεονεκτήματα του ESP32 είναι η ενεργειακή του αποδοτικότητα, κάνοντάς τον ιδανικό για συσκευές με μπαταρία και εφαρμογές IoT που απαιτούν μεγάλη διάρκεια λειτουργίας,

5. Διαδικασία Κατασκευής: Ο ESP32 κατασκευάζεται από την TSMC (Taiwan Semiconductor Manufacturing Company) με τη διαδικασία 40 nm, η οποία επιτρέπει καλύτερη απόδοση και ταυτόχρονα μειώνει την κατανάλωση ενέργειας.

6. Σετ Ανάπτυξης και GPIO Pins: Ο ESP32 παρέχεται συνήθως ως μέρος σετ ανάπτυξης, που περιλαμβάνουν μια ποικιλία GPIO (γενικής χρήσης είσοδοι/έξοδοι) και συνδέσεις. Αυτές οι πλακέτες ανάπτυξης είναι ιδανικές για πρωτοτυποποίηση, με διάφορα μοντέλα που προσφέρουν διαφορετικές διαμορφώσεις για εξειδικευμένες ανάγκες.

7. Διάδοχος του ESP8266: Ο ESP32 αποτελεί εξέλιξη του ESP8266, προσφέροντας καλύτερη απόδοση, επιπλέον δυνατότητες (όπως Bluetooth) και συνολικά καλύτερη απόδοση για σύγχρονες εφαρμογές ασύρματης επικοινωνίας.

2.2.2 Εφαρμογές ESP 32

1. Internet of Things (IoT): Έξυπνες συσκευές όπως θερμοστάτες, συστήματα αυτοματισμού σπιτιού και φορητές συσκευές.

2. Ασύρματη Επικοινωνία: Συσκευές που απαιτούν τόσο Wi-Fi όσο και Bluetooth για τη μεταφορά δεδομένων.

3. Ενσωματωμένα Συστήματα: Συστήματα χαμηλής κατανάλωσης ενέργειας που απαιτούν ασύρματη επικοινωνία για συλλογή δεδομένων, απομακρυσμένο έλεγχο ή παρακολούθηση.

Συμπέρασμα:

Ο ESP32 προσφέρει μια ισχυρή πλατφόρμα για την ανάπτυξη έργων IoT και ασύρματης επικοινωνίας. Η συνδυασμένη δυνατότητα επεξεργασίας διπλού πυρήνα, η ενεργειακή αποδοτικότητα, οι ενσωματωμένες ασύρματες δυνατότητες και οι ποικίλες επιλογές ανάπτυξης τον καθιστούν εξαιρετικά δημοφιλή επιλογή για μηχανικούς και προγραμματιστές που εργάζονται σε ενσωματωμένα συστήματα.

2.2.3 Προγραμματισμός του ESP32

Ο ESP32 υποστηρίζει πληθώρα γλωσσών προγραμματισμού, πλαισίων εργασίας (frameworks), και περιβαλλόντων ανάπτυξης που επιτρέπουν στους προγραμματιστές να δημιουργήσουν εφαρμογές με διάφορους τρόπους. Εδώ είναι μια επισκόπηση των πιο δημοφιλών εργαλείων και τεχνολογιών για τον προγραμματισμό του ESP32:

1. ESP-IDF (Espressif IoT Development Framework): Το επίσημο πλαίσιο ανάπτυξης της Espressif για τα SoCs (System on Chip) ESP32, ESP32-S, ESP32-C και ESP32-H. Παρέχει πλήρη υποστήριξη για την ανάπτυξη εφαρμογών IoT σε χαμηλό επίπεδο, συμπεριλαμβανομένων των λειτουργικών συστημάτων, του χειρισμού της επικοινωνίας Wi-Fi και Bluetooth, και των δυνατοτήτων διαχείρισης ενέργειας.

2. Arduino-ESP32: Η υποστήριξη του ESP32 για την πλατφόρμα Arduino, η οποία επιτρέπει τη χρήση της γνωστής και δημοφιλούς γλώσσας προγραμματισμού του Arduino για την ανάπτυξη εφαρμογών στο ESP32. Υποστηρίζει πολλές δυνατότητες και βιβλιοθήκες για εύκολη ανάπτυξη εφαρμογών IoT.

3. Espruino: Μια υλοποίηση JavaScript για μικροελεγκτές, που προσομοιώνει το Node.js και επιτρέπει την ανάπτυξη εφαρμογών IoT με JavaScript για τον ESP32. Είναι ιδανικό για όσους είναι εξοικειωμένοι με το περιβάλλον Node.js.

4. MicroPython (και CircuitPython): Μια ελαφριά υλοποίηση της Python 3 για μικροελεγκτές. Το MicroPython επιτρέπει στους προγραμματιστές να χρησιμοποιούν τον δημοφιλή και εύκολο στη χρήση γλωσσικό περιβάλλον Python για να προγραμματίσουν τον ESP32. Το CircuitPython είναι μια παραλλαγή του MicroPython που απευθύνεται σε εκπαιδευτικά περιβάλλοντα.

5. Mongoose OS: Ένα λειτουργικό σύστημα για συνδεδεμένα προϊόντα σε μικροελεγκτές, το οποίο υποστηρίζει προγραμματισμό με JavaScript ή C. Συνιστάται από την Espressif, το AWS IoT και το Google Cloud IoT. Είναι ιδανικό για εφαρμογές IoT που απαιτούν σύνδεση στο διαδίκτυο.

6. mruby για τον ESP32: Μια ελαφριά υλοποίηση της γλώσσας προγραμματισμού Ruby, που προορίζεται για χρήση σε μικροελεγκτές όπως ο ESP32.

7. Nim για τον ESP32: Η γλώσσα προγραμματισμού Nim μπορεί να χρησιμοποιηθεί για την ανάπτυξη εφαρμογών στον ESP32, με χαρακτηριστικά όπως την υψηλή απόδοση και την ευκολία στην ανάπτυξη.

8. NodeMCU: Firmware βασισμένο στο Lua, το οποίο επιτρέπει την ανάπτυξη εφαρμογών IoT και τη χρήση του ESP32 σε πλατφόρμες που υποστηρίζουν Lua.

9. Rust: Η γλώσσα Rust προσφέρει υψηλή απόδοση και ασφάλεια μνήμης, καθιστώντας την μια καλή επιλογή για ανάπτυξη εφαρμογών υψηλής απόδοσης και αξιόπιστων συστημάτων στο ESP32.

10. Swift: Η γλώσσα προγραμματισμού Swift, που είναι γνωστή για την ανάπτυξη εφαρμογών iOS, υποστηρίζεται και στον ESP32 για τη δημιουργία εφαρμογών με εξαιρετική απόδοση.

11. Visual Studio Code με το ESP-IDF Extension: Το Visual Studio Code (VS Code) είναι ένα δημοφιλές IDE που μπορεί να χρησιμοποιηθεί με το ESP-IDF Extension για να αναπτυχθούν εφαρμογές στο ESP32, με πλήρη υποστήριξη για την πλατφόρμα του Espressif και εύκολες δυνατότητες debugging.

12. Zerynth: Μια πλατφόρμα που επιτρέπει τη χρήση Python για ανάπτυξη εφαρμογών IoT και μικροελεγκτών, περιλαμβανομένου του ESP32. Εστιάζει στην ευχρηστία και τη σύνδεση συσκευών στο διαδίκτυο.

13. Matlab Simulink: Ένα εργαλείο για τη μοντελοποίηση, τον προγραμματισμό και την ανάπτυξη εφαρμογών για τον ESP32. Χρησιμοποιείται συνήθως για εφαρμογές σε περιοχές όπως η αυτοματοποίηση, η μηχανική και η ανάλυση δεδομένων.

Η ποικιλία εργαλείων και γλωσσών προγραμματισμού που υποστηρίζονται από τον ESP32 καθιστά τον μικροελεγκτή εξαιρετικά ευέλικτο και κατάλληλο για διαφορετικές εφαρμογές, από IoT και έξυπνες συσκευές έως πιο εξειδικευμένες χρήσεις που απαιτούν υψηλή απόδοση ή προγραμματισμό σε γλώσσες όπως Python, JavaScript, Rust και άλλες.

2.2.4 Χρήση και Εφαρμογές του ESP32

Ο ESP32 χρησιμοποιείται εκτεταμένα σε εμπορικές, βιομηχανικές και ακαδημαϊκές εφαρμογές λόγω της χαμηλής του τιμής, της υψηλής απόδοσης και των δυνατοτήτων του για συνδεσιμότητα μέσω Wi-Fi και Bluetooth. Εδώ είναι μερικές από τις σημαντικότερες χρήσεις του:

Χρήση σε εμπορικές συσκευές:

1. IoT LED βραχιόλι της Alibaba: Το 2017, η Alibaba χρησιμοποίησε ESP32 για τα IoT LED βραχιόλια που φορούσαν οι συμμετέχοντες στη συνάντηση της ετήσιας εκδήλωσης. Κάθε βραχιόλι λειτουργούσε ως "pixel", λαμβάνοντας εντολές για συγχρονισμένο έλεγχο LED φωτισμού, δημιουργώντας μια "ζωντανή και ασύρματη" οθόνη.

2. DingTalk M1: Ένα βιομετρικό σύστημα παρακολούθησης παρουσιών για εργαζομένους, που χρησιμοποιεί τον ESP32 για τη συλλογή και επεξεργασία δεδομένων.

3. mruby για τον ESP32: Μια ελαφριά υλοποίηση της γλώσσας προγραμματισμού Ruby, που προορίζεται για χρήση σε μικροελεγκτές όπως ο ESP32.

4. Hard Kernel's Odroid Go: Ένα σύστημα παιχνιδιών χειρός βασισμένο σε ESP32, που δημιουργήθηκε για να γιορτάσει την 10η επέτειο της Odroid.

5. Playdate: Μια φορητή κονσόλα βιντεοπαιχνιδιών που αναπτύχθηκε από την Panic Inc. και την Teenage Engineering, χρησιμοποιώντας τον ESP32 για λειτουργίες και συνδεσιμότητα.

6. Octopus Energy Mini: Μια συσκευή παρακολούθησης ενέργειας σε πραγματικό χρόνο βασισμένη στο ESP32-C6.

7. Mysa Smart Thermostats: Έξυπνοι θερμοστάτες που βασίζονται στο ESP32-WROOM, που προσφέρουν απομακρυσμένο έλεγχο και αυτοματισμούς θέρμανσης.

Χρήση σε βιομηχανικές συσκευές:

1. Moduino X σειρά της TECHBASE: Τα βιομηχανικά υπολογιστικά μοντέλα X1 και X2 βασίζονται στον ESP32-WROVER / ESP32-WROVER-B και χρησιμοποιούνται για αυτοματοποίηση και παρακολούθηση, υποστηρίζοντας ψηφιακές εισόδους/εξόδους, αναλογικές εισόδους και διάφορες διεπαφές δικτύου.

2. NORVI IIOT Βιομηχανικές Συσκευές: Βιομηχανικές συσκευές αυτοματισμού και παρακολούθησης που βασίζονται στον ESP32-WROVER / ESP32-WROVER-B, προσφέροντας δυνατότητες LoRa και Nb-IoT ως επεκτάσιμα modules για βιομηχανική χρήση.

Ακαδημαϊκή χρήση:

1. Ακαδημαϊκή έρευνα και εκπαιδευτικά έργα: Ο ESP32 χρησιμοποιείται σε εκπαιδευτικά περιβάλλοντα και ερευνητικά έργα για την ανάπτυξη καινοτόμων συστημάτων. Για παράδειγμα, χρησιμοποιείται για την ανάπτυξη ενός συστήματος έξυπνου σπιτιού που παρακολουθεί και ελέγχει τη φόρτιση ηλεκτρικών οχημάτων, λαμβάνοντας υπόψη την κατανάλωση ρεύματος άλλων συσκευών και την ισχύ που έχει συμβληθεί από το ηλεκτρικό δίκτυο.

2. DIY Έργα: Στην ακαδημαϊκή κοινότητα και στους λάτρεις των έργων DIY (Do-It-Yourself), ο ESP32 χρησιμοποιείται για τη δημιουργία φθηνών drones και άλλων ηλεκτρονικών κατασκευών.

Η ευελιξία του ESP32, η οικονομία του και οι δυνατότητες του για συνδεσιμότητα καθιστούν τον μικροελεγκτή ιδανικό για ένα ευρύ φάσμα εφαρμογών, τόσο σε εμπορικές όσο και σε βιομηχανικές και

ακαδημαϊκές χρήσεις. Από το IoT και την αυτοματοποίηση μέχρι την ακαδημαϊκή έρευνα και τα DIY έργα, ο ESP32 προσφέρει λύσεις για πολλές διαφορετικές ανάγκες.

2.3 IoT (Internet of Things)

Το IoT (Internet of Things), ή αλλιώς Διαδίκτυο των Πραγμάτων, αναφέρεται σε ένα σύνολο συσκευών που συνδέονται μεταξύ τους και με το διαδίκτυο, επιτρέποντας την απομακρυσμένη παρακολούθηση, έλεγχο και ανάλυση δεδομένων. Αυτές οι συσκευές μπορεί να είναι αισθητήρες, οικιακές συσκευές, ιατρικά όργανα, βιομηχανικά μηχανήματα ή ακόμα και οχήματα.

Η βασική ιδέα του IoT είναι ότι μέσω της σύνδεσης και της επικοινωνίας, οι συσκευές μπορούν να λαμβάνουν δεδομένα για το περιβάλλον τους και να παίρνουν αποφάσεις ή να ενημερώνουν τον χρήστη.

Οι κύριες τεχνολογίες που χρησιμοποιούνται στο IoT περιλαμβάνουν:

- Αισθητήρες: Μετρούν φυσικά μεγέθη όπως θερμοκρασία, υγρασία, πίεση, φως, κίνηση κ.ά.
- Μικροελεγκτές: Μονάδες ελέγχου που συλλέγουν και επεξεργάζονται δεδομένα από τους αισθητήρες.
- Δίκτυα επικοινωνίας: Wi-Fi, Bluetooth, Zigbee, LoRa, NB-IoT, Ethernet.
- Cloud υπηρεσίες: Φιλοξενούν τα δεδομένα και παρέχουν εργαλεία για ανάλυση, ειδοποιήσεις ή αυτοματισμούς.

Ένα παράδειγμα που θα μπορούσαμε να δώσουμε θα ήταν για ένα "έξυπνο σπίτι" όπου ένας αισθητήρας θερμοκρασίας καταγράφει τη θερμοκρασία του δωματίου και αποστέλλει τα δεδομένα μέσω Wi-Fi σε μια cloud πλατφόρμα. Εκεί, ο χρήστης μπορεί να δει την ένδειξη από το κινητό του, και αν η θερμοκρασία είναι χαμηλή, το σύστημα μπορεί να ενεργοποιήσει αυτόματα τη θέρμανση.

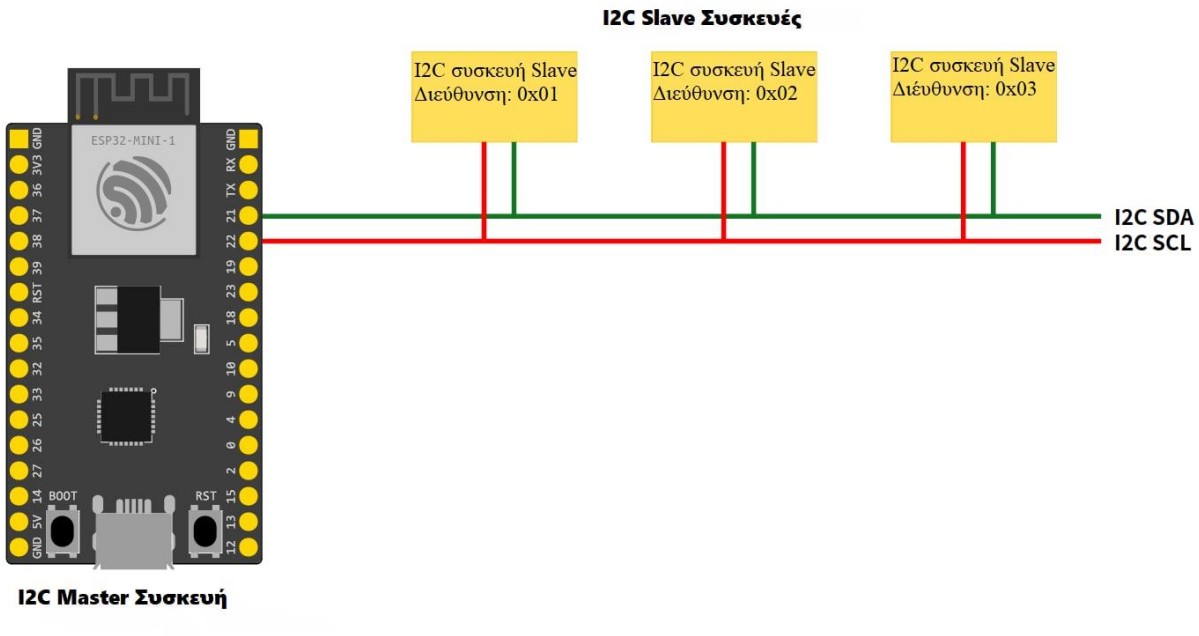
2.4 I2C (Inter-Integrated Circuit)

Το I2C (Inter-Integrated Circuit) είναι ένα πρωτόκολλο σειριακής επικοινωνίας που χρησιμοποιείται για τη μεταφορά δεδομένων μεταξύ ολοκληρωμένων κυκλωμάτων μέσα σε μια πλακέτα.

Το I2C επινοήθηκε από την εταιρεία Philips τη δεκαετία του 1980 και έχει υιοθετηθεί ευρέως λόγω της απλότητας και της αποδοτικότητάς του.

Ένα από τα βασικά χαρακτηριστικά του είναι ότι χρησιμοποιεί δύο καλώδια για την επικοινωνία των ολοκληρωμένων. Την γραμμή δεδομένων SDA (Serial Data) και την γραμμή χρονισμού SCL (Serial Clock). Επίσης μπορεί υποστηρίξει πολλαπλές συσκευές στο ίδιο δίαυλο, με μια ιεραρχία master-slave.

Κάθε slave συσκευή έχει μια μοναδική διεύθυνση (I2C address) ενώ ο master (συνήθως ένας μικροελεγκτής όπως το Arduino ή το ESP32) ελέγχει την επικοινωνία. Το I2C χρησιμοποιείται κυρίως για να διασυνδέει αισθητήρες, οθόνες LCD, EEPROM, ρολόγια πραγματικού χρόνου (RTC), κ.ά.



Εικόνα 2-5:Επικοινωνία I2C με Slave Συσκευές

2.5 Σχέση I2C με IoT

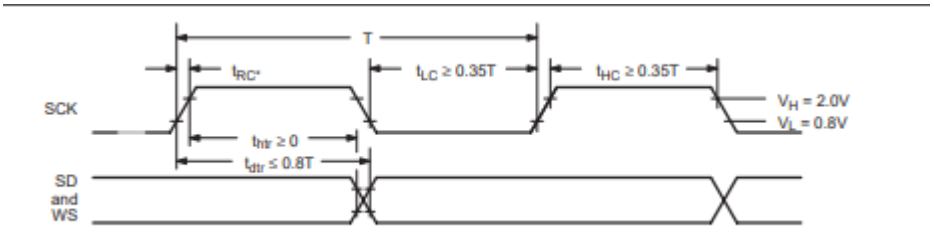
Το IoT βασίζεται έντονα στην απόκτηση και μετάδοση δεδομένων από το φυσικό περιβάλλον προς το δίκτυο. Οι αισθητήρες που συλλέγουν αυτά τα δεδομένα συνδέονται συχνά με μικροελεγκτές (όπως το ESP32, Arduino, Raspberry Pi κ.ά.) μέσω I2C λόγω της απλότητάς του και της δυνατότητας πολυπλοκότητας σε μικρό χώρο.

Ένας τυπικός κύκλος λειτουργίας περιλαμβάνει τον αισθητήρα (π.χ. ένας BME280 για θερμοκρασία, υγρασία και πίεση) ο οποίος στέλνει τα δεδομένα του στον μικροελεγκτή μέσω I2C. Τότε ο μικροελεγκτής επεξεργάζεται τα δεδομένα και τα αποστέλλει μέσω Wi-Fi, Bluetooth ή άλλης τεχνολογίας σε ένα cloud σύστημα ή application. Τέλος ο χρήστης ή το σύστημα κάνει χρήση των δεδομένων για προβολή, ειδοποιήσεις, ή ενεργοποίηση άλλων συσκευών (π.χ. ενεργοποίηση ανεμιστήρα).

Αυτός ο τρόπος επικοινωνίας είναι ιδανικός για IoT εφαρμογές επειδή το I2C επιτρέπει τη σύνδεση πολλών αισθητήρων σε λίγες γραμμές, μειώνοντας το κόστος και την πολυπλοκότητα του hardware.

2.6 I2S (inter-integrated circuit sound)

Το I2S είναι ένας σειριακός δίαυλος που έχει σχεδιαστεί για την σύνδεση και για την μεταφορά ψηφιακού ήχου ολοκληρωμένων εξαρτημάτων ήχου μεταξύ τους, μέσα σε μια ηλεκτρονική συσκευή. Τέτοια εξαρτήματα μπορεί να είναι DACs (Μετατροπέας ψηφιακού σε αναλογικό), ADCs (Μετατροπέας αναλογικού σε ψηφιακό), ψηφιακά μικρόφωνα, ενισχυτές και μικροελεγκτές. Τα δεδομένα ψηφιακού ήχου μεταφέρονται σε δυο κανάλια (stereo) και με την μέθοδο της παλμοκωδικής διαμόρφωσης. Ένας δίαυλος I2S αποτελείται απο τρείς βασικές γραμμές (σήματα). Το Bit Clock δηλαδή το ρολόι χρονισμού (BCLK), το οποίο καθορίζει τον χρονισμό των bit, . Το Word Select (WS/LRCLK/LRC) που επιλέγει το κανάλι ήχου την κάθε στιγμή. Όταν το WS=0 τότε μεταδίδεται το αριστερό κανάλι και όταν το WS=1 μεταδίδεται στο δεξί κανάλι.Τέλος το Serial Data (SD) μεταφέρει τα δεδομένα bit ήχου σειριακά ξεκινώντας απο το πιο σημαντικό bit (MSB). Λειτουργεί όμοια με το I2C στο κομμάτι ιεραρχίας master-slave (πομπού-δέκτη).

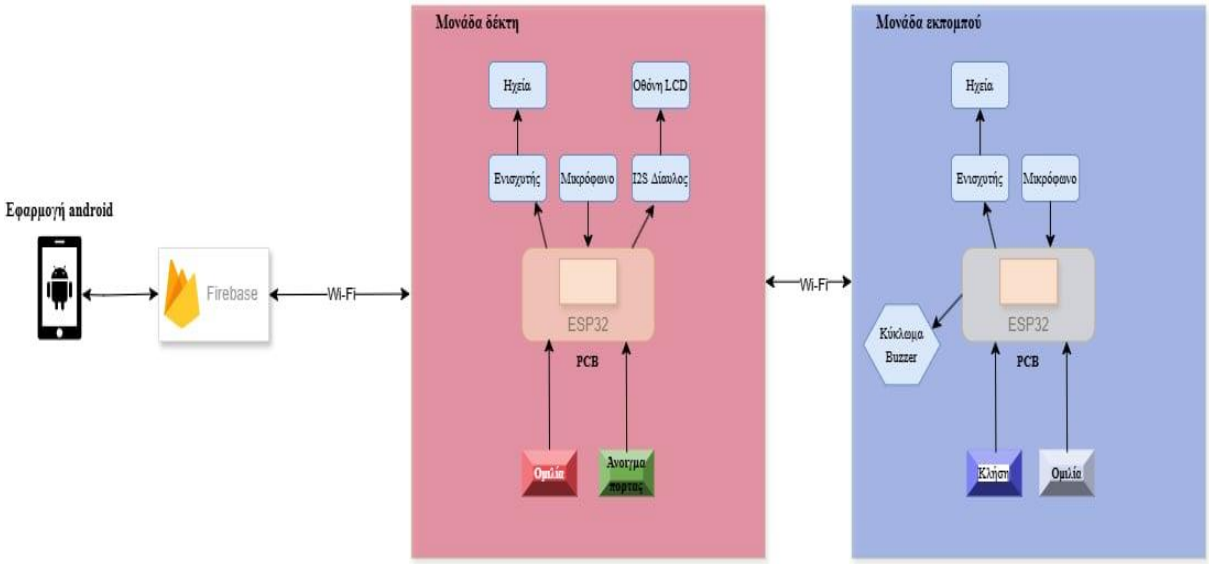


Εικόνα 2-6:Χρονισμός I2S ενός πομπού.

Κεφάλαιο 3ο: Αρχιτεκτονική συστήματος

3.1 Σχεδιασμός Θυροτηλεφώνου

Για τις ανάγκες του συστήματος χρειάστηκε να δημιουργηθούν δύο μονάδες. Η πρώτη που θα βρίσκεται στην είσοδο της πολυκατοικίας και θα έχει τον ρόλο του πομπού (transmitter) για το λόγω ότι αυτή η μονάδα ξεκινάει την επικοινωνία πρώτη. Η δεύτερη θα βρίσκεται εντός του διαμερίσματος και θα έχει τον ρόλο του δέκτη (receiver) επειδή διαθέτει πρόσβαση στο router της κάθε οικίας και κατά συνέπεια σταθερή πρόσβαση στο διαδίκτυο. Όπως κάθε κλασσικό θυροτηλέφωνο, απαιτεί κάποια περιφερειακά όπως μικρόφωνο, ακουστικό/ηχείο, κώδωνα(buzzer) έτσι και οι μονάδες αυτές περιέχουν τα παραπάνω, με την διαφορά ότι δεν υπάρχει φυσική επικοινωνία με καλώδιο λόγω των πλακετών ESP32. Επίσης στον δέκτη εμπεριέχεται μια οθόνη LCD.



Εικόνα 3.1 Διάγραμμα δομής συστήματος.

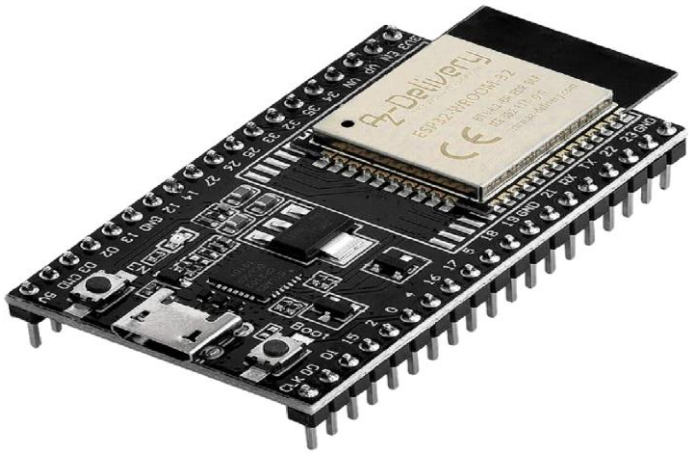
3.2 Περιφερειακά που Χρησιμοποιήθηκαν

Τα υλικά που χρησιμοποιήθηκαν ήταν τα εξής:

- 2 x ESP32
- 16x2 LCD οθόνη
- LCD display I2C interface module
- Buzzer
- 2 Πλαστικές θήκες
- 4 x Μπουτόν με LED (άσπρο, μπλέ, πράσινο, κόκκινο)
- 2 x Ενισχυτής max98357a
- 2 x Μικρόφωνα MEMS (INMP441)
- 2 Ηχεία
- DIY Πλακέτα
- 8 x Αντιστάσεις 330 Ohm
- 2 x Αντιστάσεις 470 Ohm
- 2 x Αντιστάσεις 1K Ohm
- 2 x Αντιστάσεις 4.7 KOhm
- 2 x Αντιστάσεις 47 KOhm
- 4 x Αντιστάσεις 10 KOhm
- 2 x DC JACK
- 2 x RGB LED
- 2 x LED κόκκινου χρώματος
- 2 x Τρανζίστορ BC817-16
- 2 x Schottky Δίοδοι

3.2.1 ESP 32

Αρχικά εφόσον η βάση της πτυχιακής εργασίας είναι η ασύρματη επικοινωνία, επιλέξαμε μια πλακέτα ανάπτυξης που ενσωματώνει WI-FI. Για τον λόγο αυτό επιλέχθηκε το ESP 32 Development Board το οποίο έχει ενσωματωμένο WI-FI διευκολύνοντας έτσι την πρόσβαση στο διαδίκτυο. Επίσης κρίθηκε κατάλληλο για να εξυπηρετήσει τις ανάγκες μας διότι εμπεριέχει διαύλους I2C και I2S οι οποίοι ήταν απαραίτητοι για την διασύνδεση της οθόνης και του ηχείου/μικροφώνου αντίστοιχα. Επιπλέον διαθέτει τον απαραίτητο αριθμό ψηφιακών εισόδων και εξόδων (GPIOs) για την ενσωμάτωση των υπόλοιπων περιφερειακών (led, buzzer etc.). Ταυτόχρονα το χαμηλό κόστος όπως και η χαμηλή κατανάλωση σε ρεύμα το καθιστά ιδανικό. Αξίζει να σημειωθεί ότι το ESP32 υποστηρίζει γλώσσες προγραμματισμού C/C++ και είναι πλήρως συμβατό με εργαλεία ανάπτυξης όπως το Visual Studio όπου θα αναλύσουμε στην συνέχεια.



Εικόνα 3-2:ESP32 DEVKITC πλακέτα ανάπτυξης.

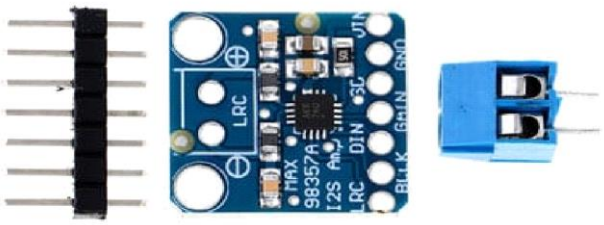
Επεξεργαστής	**Dual-core Tensilica Xtensa LX6 χρονισμένο στα 240Mhz**
Μνήμη RAM	**520Kb SRAM (16Kb cache)**
Wi-fi	**802.11 b/g/n (2.4 GHz)**
Bluetooth	**V4.2 BR/EDR και BLE**
I/O Pins	**34 GPIOs**
Αναλογικές είσοδοι (ADCs)	**16 (12-bit)**
DACs	**2 (8-bit)**

Μνήμη ROM	**448Kb**
Τροφοδοσία	**5V μέσω USB, 3.3V onboard LDO**
PWM	**28 ακίδες ικανές για pwm**
SPI / UART / I2C / I2S	**4 pins/ 3 pins / 2 pins / 2 pins**
Διαστάσεις (μήκος,πλάτος)	**48,26 mm - 27,94mm**
Καταναλώσεις ρεύματος	**240 mA μέγιστη μA**

Πίνακας 3-1:Πίνακας χαρακτηριστικών ESP32

3.2.2 Ήχος

Όσον αφορά τον ήχο επιλέχθηκαν δύο ηχεία των 80 Ohm/5W. Παρόλα αυτά ήταν απαραίτητη η ενίσχυση του ήχου κάτι το οποίο επιτεύχθηκε με την προσθήκη ενός ολοκληρωμένου κυκλώματος ενισχυτή του MAX98357A. Η επιλογή του συγκεκριμένου ενισχυτή έγινε διότι είναι σχεδιασμένος να δέχεται απευθείας είσοδο I2S χωρίς την ανάγκη μετατροπής από ψηφιακό σε αναλογικό διότι αποκωδικοποιεί από μόνος του τα ψηφιακά σήματα σε αναλογικά. Σημαντικοί λόγοι ήταν επίσης η υψηλή αποδοτικότητα αφού είναι ενισχυτής κλάσης D σε συνδυασμό με το ότι δεν χρειάζεται φίλτρα στην έξοδο όπως και το ότι προσφέρει πέντε διαφορετικές ρυθμίσεις κέρδους (3dB, 6dB, 9dB, 12dB και 15dB) και όλα αυτά σε πολύ μικρό μέγεθος με λίγους ακροδέκτες.



Εικόνα 3-3:MAX98357A.

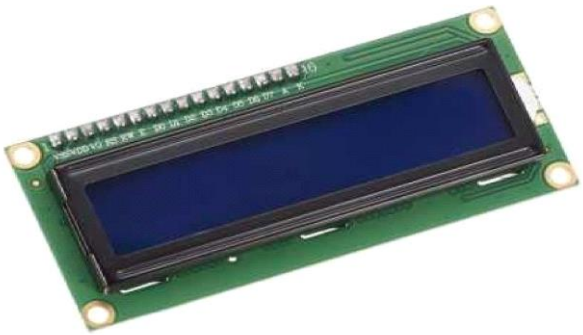
3.2.3 Μικρόφωνο INMP441

Για την καταγραφή ήχου επιλέχθηκε το μικρόφωνο INMP441. Το INMP441 είναι ένα μικροηλεκτρονικό μικρόφωνο (MEMS), δηλαδή χρησιμοποιεί μικρής κλίμακας μηχανικά μέρη για μετατροπή ακουστικών σημάτων σε ψηφιακά. Χρησιμοποιεί I2S πρωτόκολλο που το καθιστά συμβατό με τον ESP 32 προσφέροντας ταυτόχρονα καλή ποιότητα ήχου με χαμηλό θόρυβο και υψηλή ευαισθησία καθώς και χαμηλή κατανάλωση σε συνδυασμό με το μικρό μέγεθος του.

Εικόνα 3-4:Μικρόφωνο INMP441.

3.2.4 LCD Οθόνη

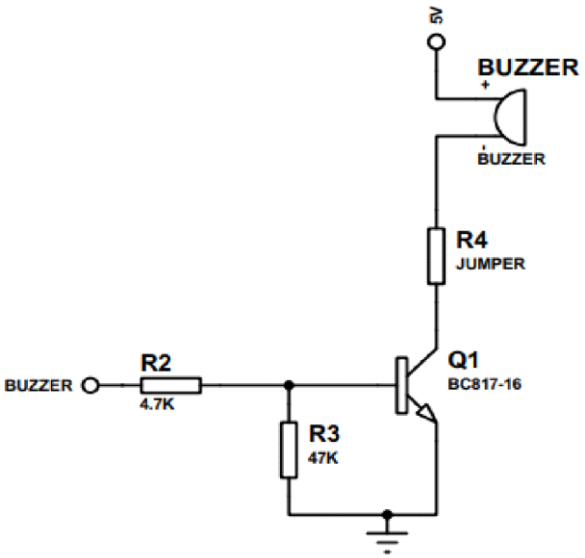
Η οθόνη που βρίσκεται στην μονάδα του δέκτη είναι μια οθόνη υγρού κρυστάλλου η οποία μπορεί να επιδείξει δύο γραμμές, με κάθε γραμμή να περιέχει δεκαέξι στήλες δηλαδή 16 διαφορετικούς χαρακτήρες. Παρέχει έναν απλό τρόπο για την απεικόνιση των μηνυμάτων για τον χρήστη και επίσης μας ενημερώνει για την ώρα και ημερομηνία. Επειδή η οθόνη δεν διαθέτει I2C δίαυλο και τα 16 pin της καταλαμβάνουν πολλά GPIOs προστέθηκε ένα κύκλωμα διασύνδεσης I2C το οποίο απαιτεί δύο καλώδια.



Εικόνα 3-5:LCD ΟΘΟΝΗ.

3.2.5 Buzzer

Το buzzer υπάρχει και στα δύο κυκλώματα αλλά χρησιμοποιείται μόνο στην μονάδα του πομπού. Ο ρόλος του ειναι να προσομοιώνει το κυπρί. Για την λειτουργία του buzzer δημιουργήθηκε ένα απλό κύκλωμα σε συνδυασμό με το τρανζίστορ BC817-16 όπου παίζει τον ρόλο του διακόπτη. Όταν δεχτεί λογικό 1/3.3V από τον ESP32 ο οποίος είναι συνδεδεμένο στην βάση του τρανζίστορ τότε πολώνεται η βάση του τρανζίστορ και οδηγείται στον κόρο. Έπειτα ο συλλέκτης «τραβά» προς την γείωση άρα το αρνητικό άκρο του μπάζερ συνδέεται με την γείωση, το διαπερνά ρεύμα και ηχεί. Όταν το σήμα από τον μικροελεγκτή βρίσκεται στην λογική κατάσταση 0/0V τότε το τρανζίστορ βρίσκεται στην αποκοπή οπότε δεν άγει άρα το buzzer δεν διαπερνάται από ρεύμα.



Εικόνα:3-6:Κύκλωμα buzzer.

3.2.6 Μπουτόν

Στις μονάδες υπάρχουν 4 μπουτόν με ενσωματωμένο led, δύο για κάθε μονάδα. Συγκεκριμένα στην μονάδα του πομπού υπάρχει ένα μπλε και ένα άσπρο μπουτόν ενώ στην μονάδα του δέκτη υπάρχει ένα πράσινο και ένα κόκκινο μπουτόν. Ο ρόλος του μπλε μπουτόν είναι για να καλεί τον δέκτη. Κάθε φορά που ο επισκέπτης πατάει το μπλε μπουτόν ενεργοποιείται ο μηχανισμός κλήσης προς την εσωτερική μονάδα και ανάβει το κόκκινο led. Όταν ο επισκέπτης θέλει να μιλήσει με τον δέκτη τότε πατάει το άσπρο μπουτόν και τότε ανάβει το μπλε led το οποίο είναι ένδειξη ότι η επικοινωνία είναι ενεργή. Όσων αφορά τον δέκτη το πράσινο μπουτόν λειτουργεί για να ανοίγει η πόρτα στον πομπό ανάβοντας ταυτόχρονα το κόκκινο led του δέκτη και το άσπρο led του πομπού, ενώ το κόκκινο μπουτόν λειτουργεί για την επικοινωνία με τον πομπό και ανάβει το πράσινο led.

Εικόνα 3-7:Μπροστινή όψη της θήκης πομπού (αριστερά) και δέκτη (δεξιά) μαζί με αντίστοιχα μπουτόν.

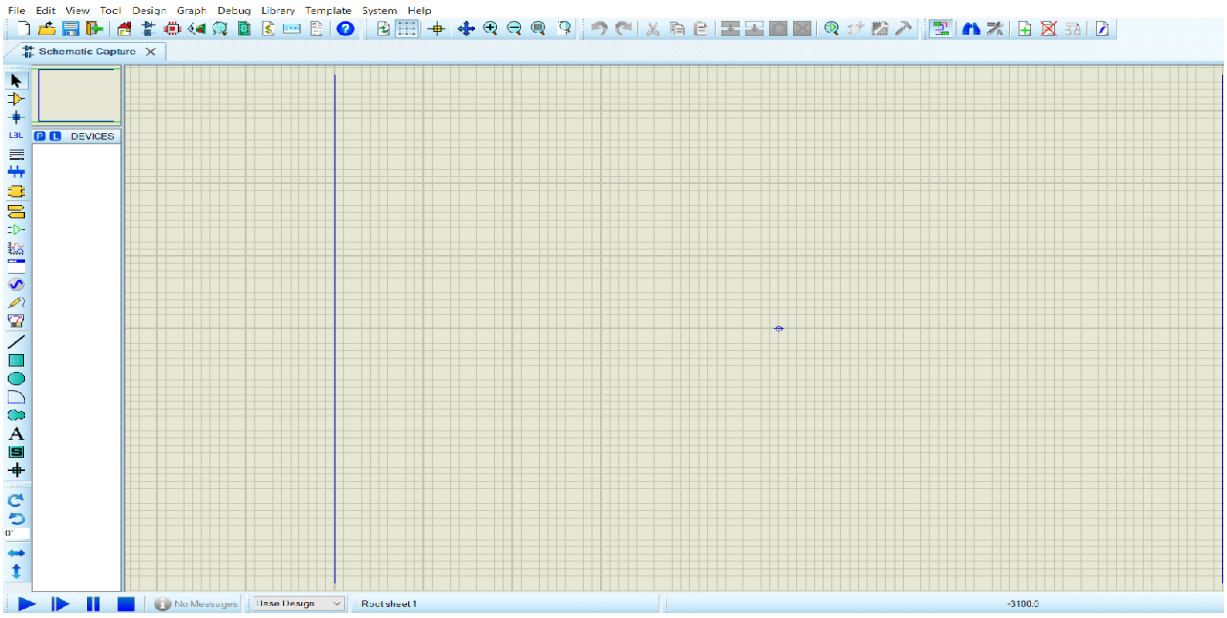
3.3 Ανάπτυξη Συστήματος

Αρχικά πριν την υλοποίηση και σύνδεση κάποιου συστήματος κρίνεται απαραίτητος ο σχεδιασμός και η προσομοίωση του σε ένα περιβάλλον σχεδίασης ηλεκτρονικών κυκλωμάτων. Αυτό γίνεται για τον έλεγχο της σχεδίασης πριν και κατά την κατασκευή, την επιδιόρθωση και βελτιστοποίηση του εκάστοτε συστήματος, την πρόληψη-ανάλυση βλαβών, για πολλαπλές δοκιμές και ανάπτυξη PCB (Printed Circuit Board). Για την εξυπηρέτηση αυτού του σκοπού επιλέχθηκε το περιβάλλον PROTEUS 8.

3.3.1 Proteus

Το PROTEUS είναι λογισμικό σχεδίασης και προσομοίωσης που χρησιμοποιείται ευρέως για την ανάλυση και την δημιουργία ηλεκτρονικών κυκλωμάτων. Κάποιες από τις δυνατότητες του λογισμικού που μας οδήγησαν στην συγκεκριμένη επιλογή ήταν το εύχρηστο περιβάλλον του, η προσομοίωση και το debugging όπου κατα την διάρκεια της προσομοίωσης ο χρήστης μπορεί να ελέγχει τάσεις και ρεύματα και την κατάσταση των λογικών θυρών σε πραγματικό χρόνο. Βοηθάει στην μείωση του

κόστους της κατασκευής καθώς μετά από δοκιμές επιτυγχάνουμε να βρούμε την καλύτερη δυνατή λύση με όσο το δυνατόν λιγότερους πόρους.



Εικόνα 3-8:Πρόγραμμα σχεδίασης PROTEUS 8.

Αρχικά για να γίνει πιο κατανοητό το πως λειτουργεί το συγκεκριμένο περιβάλλον θα το χωρίσουμε σε τρία μερη:

1. Schematic Capture: Εκεί όπου γίνεται ο βασικός σχεδιασμός του κυκλώματος και αποτελεί ένα είδος καμβά.

2. Στήλη εργαλείων: Στα αριστερά της οθόνης απεικονίζονται κάποια απο τα βασικά εργαλεία του προγράμματος για τον σχεδιασμό. Όπως για παράδειγμα η τοποθέτηση παθητικών και ενεργών εξαρτημάτων, ειδικών συσκευών π.χ μικροελεγκτές και καλωδίων.

3. Γραμμή εργαλείων: Επάνω από το Schematic Capture βρίσκεται η γραμμή εργαλείων στην οποία εμπεριέχονται βασικές λειτουργίες του προγράμματος όπως το PCB layout, 3D Visualiser, κόστος υλικών κ.α.

3.3.2 Συνδεσμολογία Συστήματος

Η ανάλυση της συνδεσμολογίας του συστήματος γίνεται με γνώμονα ένα γενικό κύκλωμα το οποίο είναι πανομοιότυπο και για τις δύο μονάδες με μικρές διαφορές. Η ανάπτυξη του συστήματος ξεκινάει με την κεντρική μονάδα που δεν είναι άλλη από τον ESP 32 Development Board. Αυτός διαθέτει 38 pins. Αρχικά η τροφοδοσία που απαιτεί η πλακέτα ESP είναι 5V και εφαρμόζεται στο pin19. Η δίοδος MBRS1100T3 τύπου Schottky χρησιμοποιείται για προστασία από ανάστροφη πολικότητα.

- Μικρόφωνο: Το μικρόφωνο διαθέτει 6 pins. Το VDD το οποίο είναι η τροφοδοσία που απαιτεί το μικρόφωνο συνδέεται στον ακροδέκτη των 3.3V του ESP. Η γείωση συνδέεται με τον ακροδέκτη GND3 του ESP. Το SD (Serial Data) είναι το κανάλι δεδομένων και συνδέεται στον IO23. Τα δεδομένα ήχου που λαμβάνονται από το μικρόφωνο αποστέλλονται σειριακά μέσω αυτής της γραμμής. Το SCK (Serial Clock) είναι το ρολόι συγχρονισμού των δεδομένων και συνδέεται στο IO26 του ESP. Αυτό καθορίζει το ρυθμό με τον οποίο μεταφέρονται τα bits στον SD. Το ESP συγχρονίζεται με τον ρυθμό του ρολογιού για να γνωρίζει πότε να διαβάζει τα bits από την γραμμή δεδομένων. Ο ακροδέκτης L/R ορίζει μόνιμα σε ποιο κανάλι θα στέλνει τα δεδομένα το μικρόφωνο. Στην περίπτωση που αυτό συνδεθεί στα 0V (λογικό 0) δηλαδή στην γείωση στέλνει τα δεδομένα στο αριστερό κανάλι ενώ όταν συνδέεται στα 3.3V (λογικό 1) τότε στέλνει τα δεδομένα στο δεξί κανάλι. Η επιλογή αυτή δίνεται σε περίπτωση που η εφαρμογή διαθέτει δύο μικρόφωνα και απαιτεί στέρεο ήχο. Όσον αφορά την δική μας εφαρμογή η ακίδα αυτή καταλήγει στην γείωση αφού χρησιμοποιούμαι μόνο ένα μικρόφωνο και θα στείλουμε δεδομένα μόνο στο αριστερό κανάλι. Τέλος η ακίδα WS που συνδέεται στην IO25 του ESP 32 είναι ένα χρονισμένο σήμα που χρησιμοποιείται για την μετάδοση δεδομένων στην I2C διασύνδεση. Με βάση τον ρυθμό του SCK το WS εναλλάσεται συνεχώς μεταξύ HIGH και LOW. Όταν το WS είναι λογικό 0 τα δεδομένα ανήκουν στο αριστερό κανάλι ενώ όταν είναι λογικό 1 τα δεδομένα ανήκουν στο δεξί κανάλι.

- Ενισχυτής και ηχείο: Το άκρο Vin του ενισχυτή συνδέεται στην τάση των 5V του τροφοδοτικού για μέγιστη απόδοση παρόλο που λειτουργεί με τάσεις απο 2.7V-5.5V. Το άκρο GND οδηγείται σε κοινή γείωση. Το άκρο BCLK που συνδέεται στην IO19 είναι το ρολόι για συγχρονισμό των δεδομένων I2S, ουσιαστικά καθορίζει τον ρυθμό μεταφοράς των bits στην γραμμή δεδομένων. Έπειτα το άκρο LRC (Left/Right Clock) συνδέεται στην IO18 όπου καθορίζει πότε τα δεδομένα αφορούν το αριστερό η το δεξιό κανάλι. Το DIN (Data In) είναι η είσοδος ψηφιακού σήματος ήχου και συνδέεται στο IO5 του ESP32. Τα δεδομένα που αποστέλονται μέσω I2S εισάγονται από αυτή την ακίδα. Ο ακροδέκτης τέσσερα δηλαδή ο GAIN ανάλογα με το που και το πως θα συνδεθεί παρέχει το αντίστοιχο κέρδος. Επιλέχθηκε ο σύνδεση στην γείωση έτσι ώτσε το κέρδος να είναι στα 12dB, το οποίο είναι αρκετά για το ηχείο που συνδέθηκε στους αντίστοιχους ακροδέκτες της πλακέτας.

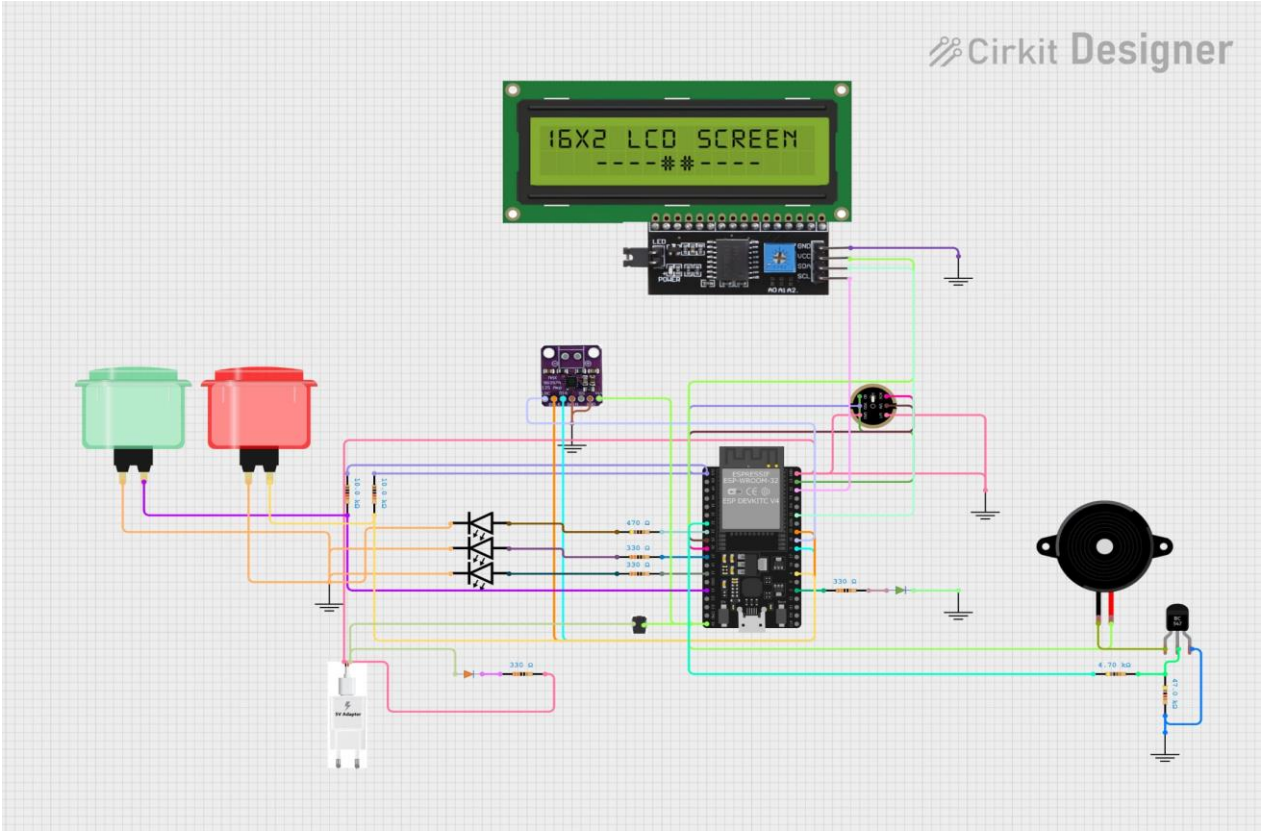
Gain	Gain Pin Configuration
15dB	100K pull down to GND
12dB	GND
9dB	Floating
6dB	VIN
3dB	100K pull up to VIN

Πίνακας 3-2: Συνδεσμολογίες ακροδέκτη "GAIN" και αντίστοιχο κέρδος.

- Μπουτόν: Και στις δύο μονάδες, η σύνδεση των μπουτόν είναι πανομοιότυπη κσι γίνεται με την χρήση pull-up αντιστάσεων. Αυτό σημαίνει ότι οι είσοδοι IO13 και IO4 των μικροελεγκτών όσο είναι ανοιχτός ο διακόπτης, δηλαδή το μπουτόν δεν είναι πατημένο βρίσκονται σε κατάσταση λογικού 1 και κατά συνέπεια τα LED παραμένουν σβηστά διότι δεν τα διαπερνά ρεύμα καθώς είναι ενσωματομένα στα μπουτόν. Μόλις ο χρήστης πατήσει το μπουτόν τότε κλείνει ο διακόπτης. Αυτό σημαίνει ο η είσοδος στον ESP 32 παίρνει λογική τιμή 0 διότι το άλλο άκρο του μπουτόν είναι συνδεδεμένο στην γείωση και ανάβει το LED. Τα LED είναι σε σειρά συνδεδεμένα με αντιστάσεις των 330 Ohm στις ακίδες IO12 και IO2 του μικροελεγκτή για προστασία.

- LED μονάδας WI-FI: Στο εξωτερικό πλαίσιο της κάθε μονάδας παρατηρούνται από δύο led σε κάθε μια. Ο λόγος ύπαρξης τους είναι για να μας ενημερώνει για την κατάσταση συνδεσιμότητας στο WI-FI. Όσο το σύστημα είναι συνδεδεμένο στο WI-FI είναι αναμένη η πράσινη ένδειξη led, ενώ όταν είναι αναμμένη η κόκκινη ένδειξη μας ενημερώνει ότι η σύνδεση είναι σε κατάσταση αναμονής. Επίσης όταν τα led αναβοσβήνουν σε κόκκινο χρώμα σημαίνει ότι υπάρχει ειδοποίηση. Τα led οδηγούνται από αντιστάσεις των 470 Ohm και 330 Ohm αντίστοιχα.

- LCD Οθόνη Δέκτη (Receiver): Η οθόνη συνδέεται επάνω σε διεπαφή I2C για την μείωση των καλωδιακών ενώσεων. Η διεπαφή αυτή έχει 4 pins (Vcc, Gnd, SDA, SCL). Το Gnd συνδέεται στην γείωση του συστήματος ενώ η ακίδα τροφοδοσίας Vcc συνδέεται με το τροφοδοτικό των 5V. Το SDA είναι συνδεδεμένο στην IO21 του μικροελεγκτή από όπου μεταφέρονται τα δεδομένα και το SCL στην IO22 η οποία λειτουργεί σαν ρολόι και συγχρονίζει την μεταφορά δεδομένων.



Εικόνα:3-9:Συνδεσμολογία στο www.cirkitstudio.com

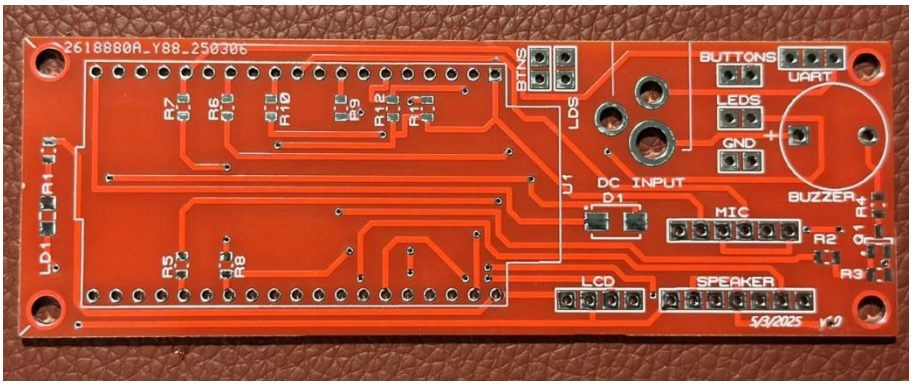
- Buzzer: Το κύκλωμα του buzzer του οποίου την λειτουργία εξηγήσαμε παραπάνω συνδέεται με το υπόλοιπο σύστημα ως εξής. Η τάση στην βάση του τρανζίστορ καθορίζεται από τον διαιρέτη τάσης των αντιστάσεων R2 και R3 και από την τάση (3.3V) που εφαρμόζεται από την IO32 του μικροελεγκτή. Η τάση στην βάση του τρανζίστορ μετά τον διαιρέτη τάσης είναι 3V και κατά συνέπεια το ρεύμα στην βάση είναι 0.064mA. Αφού το ρεύμα συλλέκτη εξαρτάται από το κέρδος (β) το οποίο είναι 100 άρα το ρεύμα συλλέκτη είναι 6.4mA το οποίο είναι αρκετό για να δημιουργήσει έναν ήχο προσομείωσης.

3.3.3 Σχεδιασμός PCB Πλακέτας

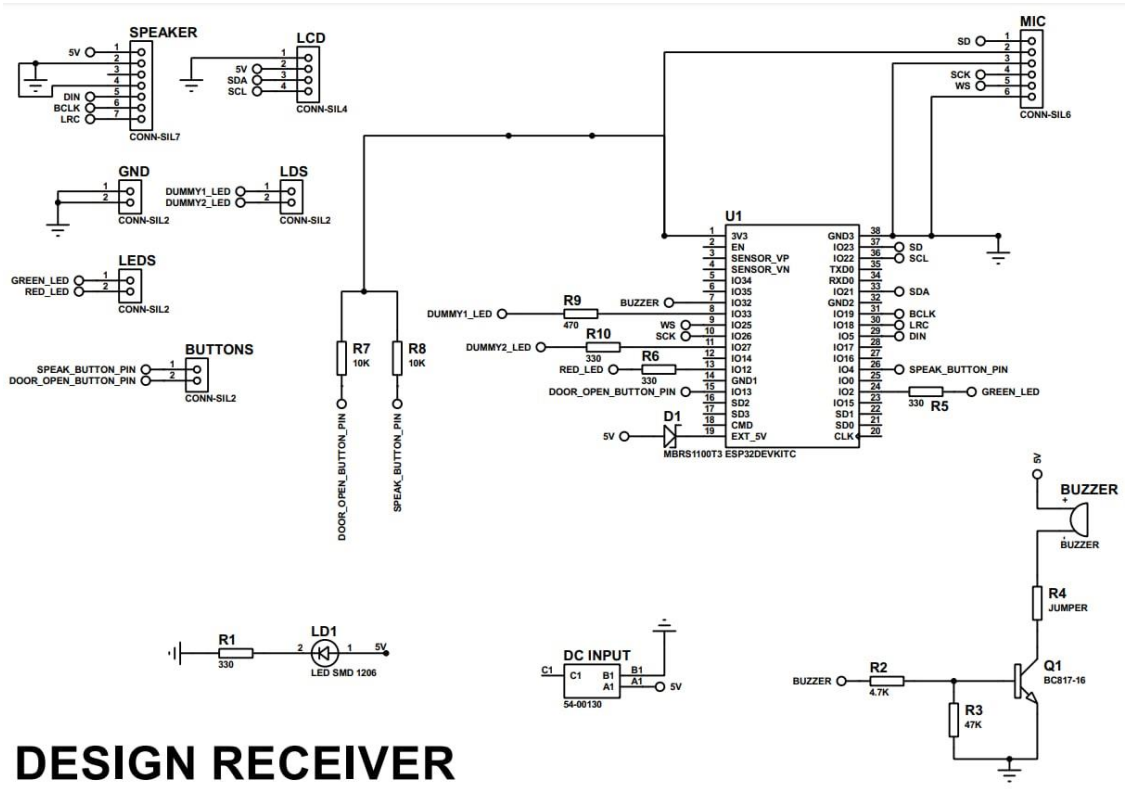
Για την καλύτερη διαχείρηση του συστήματος αλλά και την μείωση του μεγέθους της κάθε μονάδας κρίθηκε απαραίτητη η σχεδίαση και η κατασκευή δύο πλακετών μια για κάθε μονάδα καθώς κατά την διάρκεια των δοκιμών στο ράστερ παρατηρήθηκε ότι τα καλώδια τα οποία απαιτεί στο σύστημα ήταν πάρα πολλά. Η πλατέτες είναι πανομοιότυπες και για τα δύο συστήματα και εμπεριέχει όλες τις συνδεσμολογίες που αναφέρθηκαν προηγουμένως. Η κάθε πλακέτα ενσωματώνει υποδοχές ακίδων για το κάθε περιφεριακό και είναι οι εξής:

- 2x19 για την πλακέτα ανάπτυξης ESP32.
- 4x2 για γειώσεις,led και κουμπιά.
- 1x7 για τον ενισχυτή του ηχείου.
- 1x4 για την διεπαφή της οθόνης lcd.
- 1x6 για το μικρόφωνο.

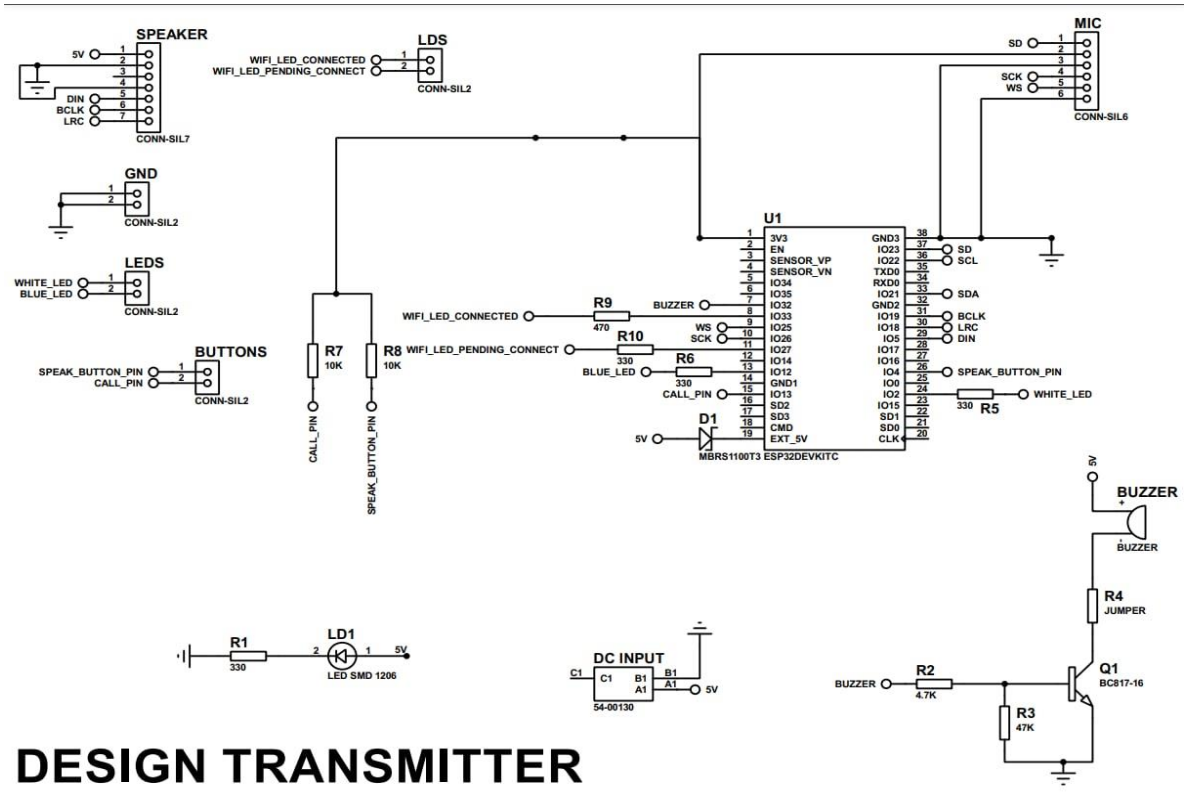
Επίσης, έχει κολληθεί μια υποδοχή (dc jack) συνεχούς ρεύματος για την τροφοδοσία της πλακέτας. Έχει προστεθεί ένα Led ένδειξης λειτουργίας και τι κύκλωμα buzzer τυπώθηκε και αυτό. Τέλος όλες οι αντιστάσεις και η δίοδος ενσωματώθηκαν και αυτές στην πλακέτα.



Εικόνα 3-10:Η πλακέτα PCB του συστήματος.



Εικόνα 3-11:Κύκλωμα PCB του δέκτη στο λογισμικό σχεδιασμού και προσομειώσεων PROTEUS.



Εικόνα 3-12: Κύκλωμα PCB του πομπού στο λογισμικό σχεδιασμού και προσομειώσεων PROTEUS.

Κεφάλαιο 4ο: Λογισμικό, Εφαρμογή και Επικοινωνία

Στο παρακάτω κεφάλαιο θα παρουσιαστεί η συνολική λειτουργία του συστήματος θυροτηλεφώνου που αναπτύχθηκε στα πλαίσια της πτυχιακής εργασίας. Η υλοποίηση του συστήματος απαρτίζεται από τρία βασικά μέρη. Τον προγραμματισμό των μονάδων θυροτηλεφώνου, την πλατφόρμα επικοινωνίας που χρησιμοποιήθηκε και την εφαρμογή Android. Στην συνέχεια θα αναλύσουμε το πως αλληλοεπιδρούν τα επιμέρους κομμάτια και θα εξηγήσουμε την ροή των δεδομένων από την στιγμή που κάποιος χτυπάει το κουδούνι, μέχρι την επικοινωνία με την εφαρμογή. Παράλληλα θα παρουσιαστεί ο κώδικας που αναπτύχθηκε για κάθε ενότητα, ώστε να γίνει κατανοητός ο τρόπος με τον οποίο επιτυγχάνεται αυτή η αλληλεπίδραση.

4.1 Προγραμματισμός ESP 32

4.1.1 Visual Studio: Integrated Development Environment (IDE)

Το προγραμματιστικό περιβάλλον το οποίο επιλέξαμε να δημιουργήσουμε την εφαρμογή μας είναι Visual Studio, το οποίο αποτελεί ένα ολοκληρωμένο περιβάλλον ανάπτυξης λογισμικού (IDE) που δημιουργήθηκε από τη Microsoft. Χρησιμοποιείται για την ανάπτυξη υπολογιστικών προγραμμάτων, όπως ιστοσελίδες, διαδικτυακές εφαρμογές, διαδικτυακές υπηρεσίες και εφαρμογές για κινητά. Το Visual Studio υποστηρίζει πλατφόρμες ανάπτυξης της Microsoft, όπως το Windows API, Windows Forms, Windows Presentation Foundation (WPF), Microsoft Store και Microsoft Silverlight. Μπορεί να παράγει τόσο εγγενή κώδικα (native code) όσο και διαχειριζόμενο κώδικα (managed code).

Το Visual Studio υποστηρίζει πολλές γλώσσες προγραμματισμού, όπως C#, C++, Visual Basic, Python, JavaScript, F#, HTML, XAML και άλλες. Αυτή η πολυγλωσσική υποστήριξη το καθιστά ιδανικό για την ανάπτυξη λογισμικού σε διάφορες πλατφόρμες και εφαρμογές. Επίσης το Visual Studio περιλαμβάνει ισχυρά εργαλεία εντοπισμού και επίλυσης σφαλμάτων. Οι προγραμματιστές μπορούν να εντοπίσουν προβλήματα στον κώδικα τους μέσω ενός βήμα προς βήμα ελέγχου, που βοηθά στην ταχεία αποδιοργάνωση σφαλμάτων και στην εύρεση λύσεων. Επιπλέον παρέχει υποστήριξη για εργαλεία ελέγχου εκδόσεων όπως το Git και το GitHub, διευκολύνοντας την συνεργασία μεταξύ των μελών μιας ομάδας ανάπτυξης και την παρακολούθηση αλλαγών στον κώδικα σε διάφορες εκδόσεις. Ακόμα παρέχει δυναμική αυτόματη συμπλήρωση και προβλέψεις κώδικα. Αυτή η δυνατότητα βοηθά τους προγραμματιστές να γράφουν κώδικα πιο γρήγορα και με λιγότερα λάθη, καθώς το εργαλείο τους προτείνει πιθανές εντολές ή παραμέτρους με βάση τον κώδικα που γράφουν.

Το Visual Studio αποτελεί ένα ισχυρό εργαλείο για προγραμματιστές, καθώς προσφέρει πολλές δυνατότητες που καλύπτουν τις ανάγκες ανάπτυξης λογισμικού σε διάφορες πλατφόρμες. Η ευελιξία, η επεκτασιμότητα και η ενσωμάτωση με σύγχρονα εργαλεία και υπηρεσίες καθιστούν το Visual Studio μια αξιόπιστη επιλογή για επαγγελματίες και φοιτητές στον τομέα της πληροφορικής.

4.1.2 Ανάλυση Βοηθητικών Αρχειών ESP 32

- GPIO.h: Αρχίκα στο αρχείο GPIO.h ορίζονται οι ακίδες που χρησιμοποιούνται από τον μικροελεγκτή με βάση την συνδεσμολογία που πραγματοποιήθηκε για την επικοινωνία με μικρόφωνο, ηχείο, leds και κουμπιά. Ο ορισμός των ακίδων είναι πανομοιότυπος και για τις δύο μονάδες με την διαφορά ότι αλλάζει η ονομασία που έχουμε ορίσει για τα leds, το κουμπί κλήσης (πομπός), το κουμπί ανοίγματος της πόρτας (δέκτης) και επίσης στον πομπό ορίζεται και η ακίδα του buzzer.

- INIT_GPIO.h: Έπειτα στο αρχείο INIT_GPIO.h εισάγεται η βιβλιοθήκη «Arduino.h» γιατί επιτρέπει την αλληλεπίδραση με τα GPIO pins μέσω των συναρτήσεων «pinMode(), digitalWrite(), digitalRead ()» και επίσης περιέχει εντολές χρονισμού όπως «delay(ms), millis()». Μετέπειτα αρχικοποιούμε τις ακίδες σαν εισόδους και εξόδους και καθορίζεται η αρχική λογική τους κατάσταση. Τα RGB led που αντιστοιχούν στις καταστάσεις του Wi-Fi (connected-pending) είναι σε ανάστροφη λογική λόγω κοινής ανόδου. Το κόκκινο (pending connection) led αρχικοποιείται ως low άρα είναι αναμμένο ενώ το πράσινο (connected) είναι high άρα σβηστό. Όσων αφορά τον πομπό έχουμε ορίσει ως εισόδους τις ακίδες του μπουτόν της κλήσης (CALL_PIN) και του μπουτόν ομιλίας (SPEAK_BUTTON_PIN) και ως εξόδους τα WHITE_LED_PIN (άνοιγμα πόρτας), BLUE_LED_PIN (ομιλία προς δέκτη), WIFI_LED_CONNECTED, WIFI_LED_PENDING_CONNECT και το BUZZER_PIN(κυπρί). Στον δέκτη σαν είσοδοι ορίζονται οι ακίδες ομιλίας (SPEAK_BUTTON_PIN) και το κουμπί ξεκλειδώματος της πόρτας εισόδου (DOOR_OPEN_BUTTON_PIN) ενώ ως έξοδοι ορίζονται τα RED_LED_PIN (κλήση από πομπό και άνοιγμα πόρτας), GREEN_LED_PIN (ομιλίας), WIFI_LED_CONNECTED και το WIFI_LED_PENDING_CONNECT. Τα led ομιλίας και πόρτας του δέκτη αρχικοποιούνται σε κατάσταση λογικού 0 (LOW) και μπαίνουν σε λογική κατάσταση 1(HIGH) μόνο όταν ο χρήστης τα πατήσει. Το ίδιο ισχύει και για τα led ομιλίας και κλήσης του πομπού.

- VARIABLES.h: Στο αρχείο αυτό περιέχονται οι βασικές μεταβλητές για το WI-FI την UDP επικοινωνία και μόνο στο αρχείο του δέκτη προστίθεται το πρωτόκολλο δικτυακού χρόνου (NTP-Network Time Protocol) το οποίο χρησιμοποιείται για τον συγχρονισμό ρολογιού υπολογιστικών συστημάτων. Επίσης ρυθμίζεται η χειμερινή ώρα Ελλάδος (UTC +2) και η προσθήκη μιας ώρας κατά την θερινή περίοδο. Οι βασικές μεταβλητές του WI-FI είναι το SSID (όνομα δικτύου) και ο κωδικός του ρούτερ. Για την UDP επικοινωνία ρυθμίζονται τα Ports απο τα οποία η κάθε μονάδα αποστέλλει και δέχεται δεδομένα ήχου, π.χ. Ο πομπός δέχεται ήχο απο την Port 1234 και στέλνει στην Port 9999 ενώ αντίστοιχα ο δέκτης δέχεται ήχο απο την Port 9999 και στέλνει απο την Port 1234. Τα Lock_port είναι και στις δύο μονάδες ίδια και είναι 5678 και τα Call_Port 4324. Η επιλογή αυτών των Ports έγινε με σκεπτικό της αποφυγής των Well-Known Ports κάποιων Registered Ports, αλλά κατά τα άλλα τυχαία.

- SCREEN.h: Στο SCREEN.h εισάγεται η βιβλιοθήκη «LiquidCrystal_I2C» για να επιτραπεί η επικοινωνία μέσω του πρωτοκόλλου I2C. Επίσης περιέχονται οι αρχικοποιήσεις της οθόνης lcd και κάποια κείμενα που εμφανίζονται στην αρχή (Booting). Αρχικά με την συνάρτηση "void texts()" εμφανίζεται το μύνημα «International Hellenic University». Καθώς όμως το μύνημα αυτό είναι πολύ μεγάλο, διαχωρίστηκε σε δύο κομμάτια. Το πρώτο μέρος είναι η λέξη «International», έπειτα το μύνημα «Hellenic University» μετατοπίζεται απο τα δεξιά προς τα αριστερά μέχρι να εξαφανιστεί εντελώς από την οθόνη. Στην συνέχεια το επόμενο μήνυμα που εμφανίζεται είναι το «Thesis» και το «Master Degree!» με το πρώτο να εμφανίζεται στην πρώτη γραμμή και να ξεκινάει από την πέμπτη στήλη ενώ το δεύτερο να εμφανίζεται στην δεύτερη γραμμή και πρώτη στήλη. Αφού καθαριστεί η οθόνη έπειτα εμφανίζονται τα ονόματα «Mala Orestis» στην πρώτη γραμμή/δεύτερη στήλη και το «Ananiadis Themis» στην δεύτερη γραμμή/πρώτη στήλη. Τέλος εμφανίζεται τα μηνύματα «Sindos, 2025» στην πρώτη γραμμή/δεύτερη στήλη, το «Fw Ver. 1.0.» στην δεύτερη γραμμή/πρώτη στήλη και αφού καθαριστεί το «System Init…» στην πρώτη γραμμή/πρώτη στήλη.

- SOUND.h: Αρχικά εισάγονται οι βιβλιοθήκες «driver/i2s.h, math.h (μόνο στον receiver)» με την πρώτη να περιέχει όλες τις συναρτήσεις και τις παραμέτρους για τον έλεγχο του πρωτοκόλλου I2S στο ESP32 και την δεύτερη να περιέχει μαθηματικές συναρτήσεις όπως η «sin()» που χρησιμοποιείται για την δημιουργία ημιτονοειδών κυμάτων για την παραγωγή ήχου στην μονάδα του δέκτη. Στην συνέχεια ορίζεται ο ρυθμός δειγματοληψίας, τα bits ανά δείγμα, η επιλογή καναλιού mono και το μέγεθος του buffer εισόδου/εξόδου στα 256 bytes για τον λόγο ότι ο ρυθμός που συλλέγει το μικρόφωνο τα δείγματα δεν είναι ίδιος απαραίτητα με τον ρυθμό που τα επεξεργάζεται ο επεξεργαστής και εάν δεν υπάρχει buffer μπορεί κάποια δεδομένα να χαθούν για αυτό αποθηκεύονται μέχρι να επεξεργαστούν. Έπειτα ορίζονται οι θύρες I2S, το μέγεθος των δεδομένων που θα επεξεργάζεται κάθε φορά ο buffer και μόνο στο αρχείο του receiver η συχνότητα του ήχου (440 Hz) που θα παραχθεί στην κλήση. Στην συνέχεια με την συνάρτηση void i2s_mic_install() αρχικοποιείται και εγκαθίσταται ο οδηγός (driver) για τα μικρόφωνα. Πιο αναλυτικά ορίζεται η λειτουργία του I2S σε master mode και σε receive mode. Αυτό σημαίνει πως το ESP παράγει τα ρολόγια (BCLK, WS) και διαβάζει τα δεδομένα από το μικρόφωνο. Καθορίζονται τα «sample_rate και bits_per_sample» όπως και η χρήση του αριστερού καναλιού, ακολουθώντας τη βασική αρχικοποίηση του ESP με I2S. Ορίζεται η χρήση του πρωτοκοκόλλου I2S, τα interrupts εκχωρούνται στο επίπεδο 1 το χαμηλότερο επίπεδο διακοπών. Επιλέγονται τέσσερα DMA Buffers των οποίων το μήκος ορίζεται από τις προηγούμενες ρυθμίσεις με αποτέλεσμα μικρότερο χρόνο καθυστέρησης. Δεν γίνεται χρήση του APLL καθώς δεν χρειάζεται ακριβής συγχρονισμός ρολογιού. Τέλος γίνεται εγκατάσταση του driver στην προκαθορισμένη πόρτα., Στην συνέχεια με την συνάρτηση «i2s_mic_setpin()» διαμορφώνονται οι ακίδες για I2S επικοινωνία με το μικρόφωνο. Όσων αφορά το ηχείο στην συνάρτηση «i2s_speaker_install()» που ευθύνεται για την παραμετροποίηση της διεπαφής I2S για μετάδοση ήχου ακολουθούνται οι ίδιες ρυθμίσεις με το μικρόφωνο με την διαφοραά ότι ο ESP μπαίνει σε λειτουργία μετάδοσης. Όπως στο μικρόφωνο έτσι και στο ηχείο υπάρχει η συνάρτηση «i2s_speaker_setpin()» η οποία διαμορφώνει τις ακίδες για I2S επικοινωνία. Για την δημιουργία ήχου κλήσης στον δέκτη δημιουργήθηκε η συνάρτηση «playSineWave()».

Στην αρχή ορίζονται οι παράμετροι:

1. Διάρκειας (2000ms).
2. Ρυθμός δειγματοληψίας (16kHz).
3. Συχνότητα ήχου (440Hz, νότα ΛΑ).
4. Πλάτος (32767, μέγιστο πλάτος για 16-bit audio).

Υπολογίζεται ο αριθμός δειγμάτων με τον εξής τύπο:

$$Αριθμός\ δειγμάτων = \frac{Διάρκεια \times Ρυθμό\ δειγματοληψίας}{1000}$$

Το οποίο ισούται με 32000 δείγματα.

Στη συνέχεια στην for παράγεται το ημιτονοειδές κύμα:

$$Δείγμα = \ Πλάτος\ \times Hμ(2 \times \pi \times f \times \frac{i}{Ρυθμό\ Δειγματοληψίας})$$

Το αποτέλεσμα αυτό μετατρέπεται σε ακέραιο 16 bit. Στην συνέχεια μεταφέρεται το δείγμα στην έξοδο I2S. Τέλος με την ολοκλήρωση της λούπας για την σίγαση του ηχείου δημιουργήθηκε η συνάρτηση «silence_speaker()». Αυτή δημιουργεί ένα buffer γεμάτο μηδενικά, δηλαδή 16 bit δείγματα με την τιμή 0 και στέλνεται στην I2S διεπαφή σταματώντας τον ήχο. Εμφανίζει επίσης στην σειριακή θύρα «Sine wave stopped.».

Αφού περιγράφηκαν οι αρχικοποιήσεις και οι βασικές ρυθμίσεις των μονάδων, παρακάτω αναλύονται οι κύριοι (main) κώδικες των μονάδων καθώς ο τρόπος πραγραμματισμού τους.

Όσων αφορά των δέκτη ορίστηκαν 4 αντικείμενα επικοινωνίας UDP χρησιμοποιώντας την κλάση WI-FI UDP. Το ένα διαχειρίζεται την λήψη ήχου, το άλλο την μετάδοση και τα άλλα δυο το έλεγχο της κληδαριάς και την το κουμπί κλήσης της εξωτερικής μονάδας.

4.1.3 Ανάλυση Βασικού Κώδικα Πομπού

Αρχικά συμπεριλαμβάνονται τα βοηθητικά αρχεία που αναλύθηκαν παραπάνω και δημιουργούνται τα αντικείμενα “udp_receive_audio”, “udp_transmit_audio”, “udp_lock” και “udp_call_button” που βασίζονται στην κλάση WIFIUDP η οποία παρέχεται από την βιβλιοθήκη WiFiUdp. Έπειτα γίνεται αρχικοποίηση του συστήματος με την “void setup()”. Δηλαδή στην συνάρτηση αυτή πρώτα ξεκινάει η

σειριακή επικοινωνία με τον υπολογιστή για debugging και επλέγεται baud rate 15200 και καλείται η συνάρτηση “Init_Pins();” για την αρχικοποίηση των ακίδων. Στην συνέχεια ξεκινάει η προσπάθεια σύνδεσης στο WI-FI με το SSID και το κωδικό που έχει δηλωθεί χειροκίνητα στο Variables.h. και αυτό επαναλαμβάνεται μέχρι να γίνει επιτυχής σύνδεση στο διαδύκτιο. Κατά την διάρκεια της προσπάθειας σύνδεσης στο WI-FI παραμένει αναμένο το κόκκινο led όπως έχει αρχικοποιηθεί και αλλάζει σε πράσινο μόνο όταν επιτευχθεί η σύνδεση.

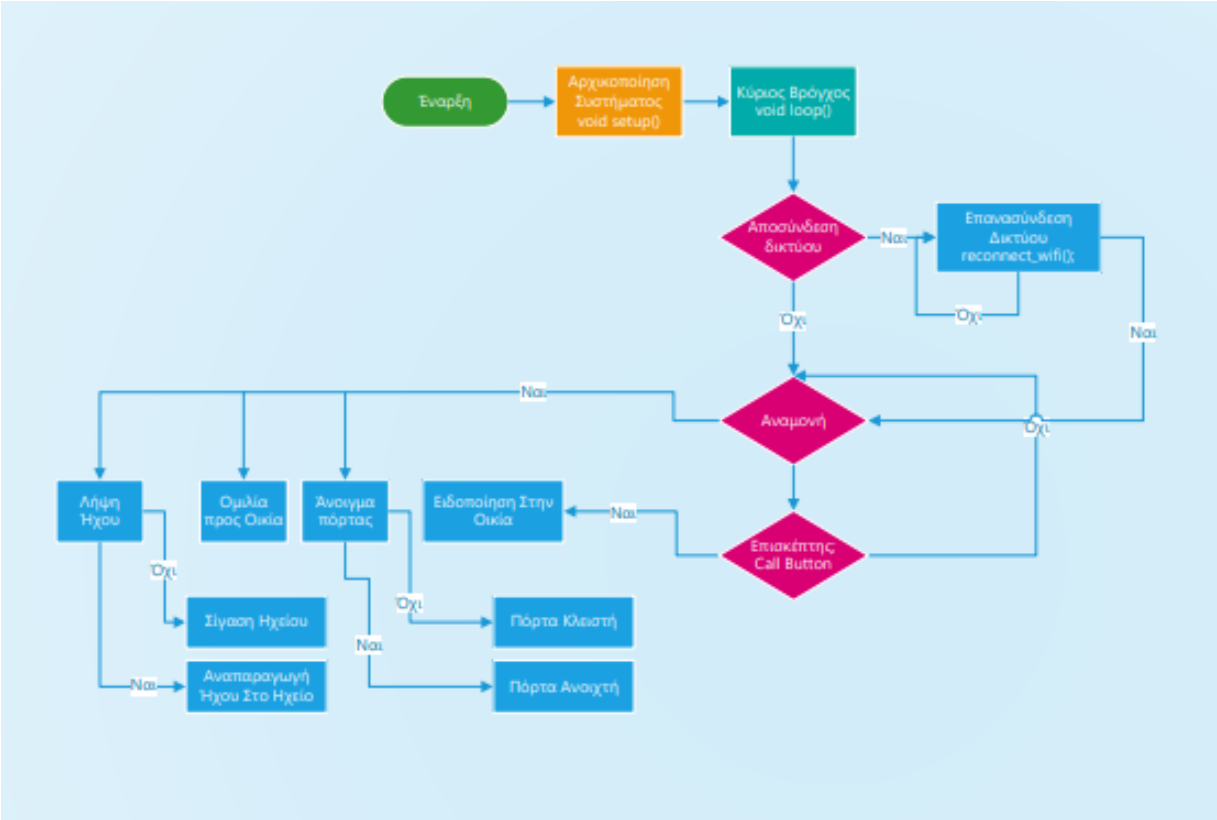
Έπειτα διαμορφώνεται η σύνδεση με το Firebase όπου ορίζεται το API_KEY και το URL του Database και ξεκίναει η σύνδεση με το Firebase. Σε περίπτωση που χαθεί η σύνδεση με το Firebase το σύστημα προσπαθεί αυτόματα να ξανασυνδεθεί. Στην συνέχεια γίνεται ανώνυμη σύνδεση και καλούνται οι συναρτήσεις “Set_IP” και “Get_IP”. Ξεκινάει η επικοινωνία μέσω των UDP Ports, καλούνται οι συναρτήσεις των ρυθμίσεων μικροφώνου, ηχείου και γίνεται έναρξη των I2S θύρων.

Στον κύριο βρόγχο του συστήματος “void loop()” αρχικά πραγματοποιείται έλεγχος σύνδεσης δικτύου. Σε περίπτωση που χαθεί η σύνδεση με το δίκτυο καλείται η συνάρτηση “reconnect_wifi();” η οποία πραγματοποιεί προσπάθεια επανασύνδεσης που επαναλαμβάνεται μέχρι να συνδεθεί. Μετά από αυτόν το έλεγχο το σύστημα βρίσκεται σε κατάσταση αναμονής. Σε αυτήν την φάση εάν πατηθεί το κουμπί κλήσης (Call Button) τότε στέλνεται μια ειδοποίηση σε μορφή μελωδίας ότι υπάρχει επισκέπτης με την συνάρτηση “sendMelodySignal()” η οποία με την σειρά της αποστέλει μέσω UDP τα δεδομένα της μεταβλητής “call_signal” που έχουν την τιμή δύο. Επίσης σε αυτή την κατάσταση ο πομπός μπορεί να ομιλήσει προς την οικία, να πραγματοποιήσει λήψη ήχου και να να δεχθεί σήμα για να ανοίξει την πόρτα.

Όσων αφορά την ομιλία προς την οικία διαβάζεται η κατάσταση του “SPEAK_BUTTON” δηλαδή του μπουτόν ομιλίας. Όταν αυτό πατηθεί τότε ανάβει το μπλέ led ως ένδειξη ορθής λειτουργίας και πραγματοποιείται το ¨διάβασμα¨ των I2S δεδομένων και δημιουργείται ένα UDP πακέτο στο οποίο γράφονται τα ηχητικά δεδομένα και στην συνέχεια αποστέλονται στον δέκτη στην συγκεκριμένη IP και πόρτα.

Επειδή ο τρόπος ομιλίας και για τις δύο μονάδες είναι ίδιος δηλαδή τα δεδομένα αποστέλονται με την μορφή UDP πακέτων, για να γίνει η λήψη ήχου γίνεται έλεγχος εάν υπάρχει κάποιο εισερχόμενο πακέτο στην καθορισμένη πόρτα “Receive_Audio_Port”, το διαβάζει και το αναπαράγει μέσω I2S στο ηχείο. Σε άλλη περίπρωση που δεν θα υπάρχει κάποιο εισερχόμενο πακέτο τότε καλείται η συνάρτηση “silence_speaker”.

Τέλος για το άνοιγμα της πόρτας αποστέλεται με τον ίδιο τρόπο από τον δέκτη το περιεχόμενο της μεταβλητής (signal = 1), διαβάζεται και μεταφέρεται στην μεταβλητή “led_signal” . Εάν ο αριθμός των bits είναι μεγαλύτερος του μηδενός και η μεταβλητή “led_signal” ισούται με ένα τότε το λευκό led ανάβει και ταυτόχρονα ηχεί το buzzer. Αυτό συμβαίνει για πέντε δευτερόλεπτα και μετά σβήνουν.



Εικόνα 4-1:Διάγραμμα ροής κώδικα πομπού.

4.1.4 Ανάλυση Βασικού Κώδικα Δέκτη

Όπως και στον κώδικα του πομπού έτσι και στου δέκτη αρχικά συμπεριλαμβάνονται οι κατάλληλες βιβλιοθήκες και τα απαραίτητα βοηθητικά αρχεία. Επίσης δημιουργούνται τα ίδια αντικείμενα με τον πομπό για την μεταφορά δεδομένων σε μορφή UDP πακέτων. Στην συνέχεια δημιουργήθηκε μια συνάρτηση Callback που καλείται αυτόματα κάθε φορά που υπάρχει αλλαγή στα δεδομένα στου stream (ροής) της βάσης δεδομένων του Firebase. Ουσιαστικά γίνεται λήψη διαδρομών δηλαδή, την βασική διαδρομή που παρακολουθούμε και το συγκεκριμένο στοιχείο που άλλαξε π.χ. /Door_System /Door. Πραγματοποιείται ανάγνωση των δεδομένων και σε περίπτωση που τα δεδομένα είναι τύπου JSON τα μετατρέπει σε αντικείμενο FirebaseJSON διαφορετικά τα αντιμετωπίζει ως απλό κείμενο. Επίσης γίνεται εκτύπωση της διαδρομής του stream, του συμβάντος, τον τύπο των δεδομένων, τον τύπο του συμβάντος και πόσα bytes περιέχει το τελευταίο stream στην σειριακή θύρα για έλεγχο. Τέλος ορίζεται μια σημαία η οποιά ελέγχεται παρακάτω στην “void loop()”.

Για τον έλεγχο αποσύνδεσης ή timeout ορίστηκε η συνάρτηση “streamTimeoutCallback”. Αυτή ελέγχει εάν η μεταβλητή timeout είναι αληθής και εκτυπώνη στην σειριακή ένα μήνυμα. Τέλος ελέγχειεάν η σύνδεση είναι ενεργή και εάν δεν είναι εκτυπώνεται πάλι στην σειριακή τον κωδικό σφάλματος και την αιτία.

Για τις αρχικοποιήσεις του συστήματος αρχίκα ξεκινάει η σειριακή επικοινωνία και στην συνέχεια καλούνται οι συναρτήσεις που έχουν δημιουργηθεί στα βοηθητικά αρχεία για την αρχικοποίηση των ακίδων, της lcd (void text). Άρα πρώτα εμφανίζονται τα κείμενο που έχουν αναφερθεί στο Screen.h. Παρακάτω ξεκινάει η σύνδεση με το WI-FI όπως και στον πομπό και εμφανίζεται στην οθόνη "Connecting to WI-FI..." όπως και στην σειριακή θύρα. Στην επιτυχία σύνδεσης το κόκκινο led που ήταν που την αρχή αναμένο σβήνει και ανάβει το πράσινο. Ταυτόχρονα με αυτή την αλλαγή η οθόνη εμφανίζει το κείμενο "Connected!" και το "Please wait…".

Παρακάτω ξεκινάει η σύνδεση με το Firebase και καλούνται οι συναρτήσεις "Set_IP" και "Get_IP". Γίνεται έλεγχος του stream και αν αυτό αποτύχει ενημερωνόμαστε στην σειριακή θύρα. Ξεκινάει η επικοινωνία μέσω των UDP ports, εμφανίζεται η ώρα και η ημερομηνία με την συνάρτηση "displayTime". Αυτή η συνάρτηση παίρνει την ώρα από τον NTP Server, τα διαμορφώνει σε μορφή ΄Ωρα: Λεπτά Ημέρα/Μήνας/Ετος και την εμφανίζει στην πρώτη σειρά της οθόνης. Οι τελευταίες συναρτήσεις που καλούνται της "void setup()" είναι αυτές του μικροφώνου, του ηχείου και η έναρξη των I2S ports.

Μπαίνοντας στην "void loop()" το σύστημα πρώτα ελέγχει εάν έχει αποσυνδεθεί από το WI-FI. Εάν αποσυνδεθεί καλεί την "reconnect_wifi" η οποία με την σειρά της επιχειρεί την επανασύνδεση με το WI-FI και όταν επιτευχθεί αποστέλει τις IP στο Firebase. Καλείται η συνάρτηση "control_indication_led" η οποία είναι υπεύθυνη για τον έλεγχο του κόκκινου led σε περίπτωση ειδοποίησης. Πιο συγκεκριμένα εάν υπάρχει ειδοποίηση τότε η μεταβλητή "currentMillis" παίρνει την τιμή εκείνης της στιγμής της συνάρτησης "millis()". Αυτή η συνάρτηση επιστρέφει τον αριθμό των χιλιοστών του δευτερολέπτου που έχουν περάσει από την έναρξη του προγράμματος. Εάν η διαφορά της "currentMillis" με την "previousMillis" (αρχικοποιημένη με τιμή μηδέν) είναι μεγαλύτερη ή ίση της μεταβλητής "interval" (αρχικοποιημένη στα 200ms) τότε η "previousMillis" παίρνει την τιμή της "currentMillis" έτσι ώστε την επόμενη φορά θα ξέρουμε πότε ξανά άλλαξε το led. Με βάση τις προυποθέσεις αυτές ελέγχεται εάν η κατάσταση του led είναι LOW (αναμμένο) και σε αυτή την περίπτωση γίνεται HIGH (σβηστό) αλλιώς αν είναι HIGH τότε γίνεται LOW. Για την ενημέρωση και τον έλεγχο της ακίδας του κόκκινου pin χρησιμοποιείται η μεταβλητή "ledState". Στην περίπτωση που δεν υπάρχει ειδοποίηση παραμένει σβηστό το κόκκινο led και αντιθέτως ανάβει το πράσινο.

Δημιουργείται ένας πίνακας, για την αποθήκευση δεδομένων ήχου που λαμβάνονται μέσω UDP. Αν υπάρχει πακέτο ήχου και πακέτο από την κλήση μέσω UDP, αποθηκεύεται το μέγεθος τους σε bytes για να αναγνωστεί μετέπειτα και δημιουργείται μια μεταβλητή στην οποία θα αποθηκευτούν τα δεδομένα που θα στείλει από το κουμπί κλήσης ο πομπός. Απαραίτητη προϋπόθεση για να συνεχίσει ο αλγόριθμος είναι η σύνδεση με το Firebase και το signup να είναι επιτυχή. Όταν ισχύουν τα παραπάνω τότε γίνεται και έλεγχος για το αν έχει δημιουργηθεί κάποια νέα πληροφορία που πρέπει να ανέβει στο Firebase και δημιουργεί το πλήρες URL για το path στην βάση (π.χ. αν αλλάξει η τιμή της μεταβλητής People τότε το URL θα είναι της μορφής "Door_System/People"). Κατόπιν στέλνεται το "bucketData" στο αντίστοιχο path. Αξίζει να σημειωθεί ότι μηδενίζεται το flag του "uploadBucket" έτσι ώστε μην επαναληφθεί η αποστολή του ίδιου πράγματος. Στην συνέχεια ακολουθεί έλεγχος εάν υπάρχει πακέτο ήχου (δηλαδή αν έχει μέγεθος μεγαλύτερο από το 0) και εφόσον ισχύει διαβάζει τα δεδομένα και τα αποθηκεύει στην μετάβλητη που δημιουργήσαμε για τον ήχο (audio_buffer[BUFFER_SIZE]). Εάν διαβάστηκαν όλα τα δεδομένα τότε γράφει τα bytes στην I2S πόρτα του ηχείου στον πομπό. Αλλιώς αν δεν στέλνονται ηχητικά δεδομένα ελέγχει αν υπάρχει πακέτο μελωδίας. Πιο αναλυτικά, αν η μεταβλητή "melodyPacketSize" στην οποία αποθηκεύονται τα δεδομένα από το κουμπί κλήσης είναι μεγαλύτερη του μηδενός τότε ενημερώνεται το σύστημα ότι η παίζει η μελωδία, διαβάζονται τα δεδομένα (τιμή και

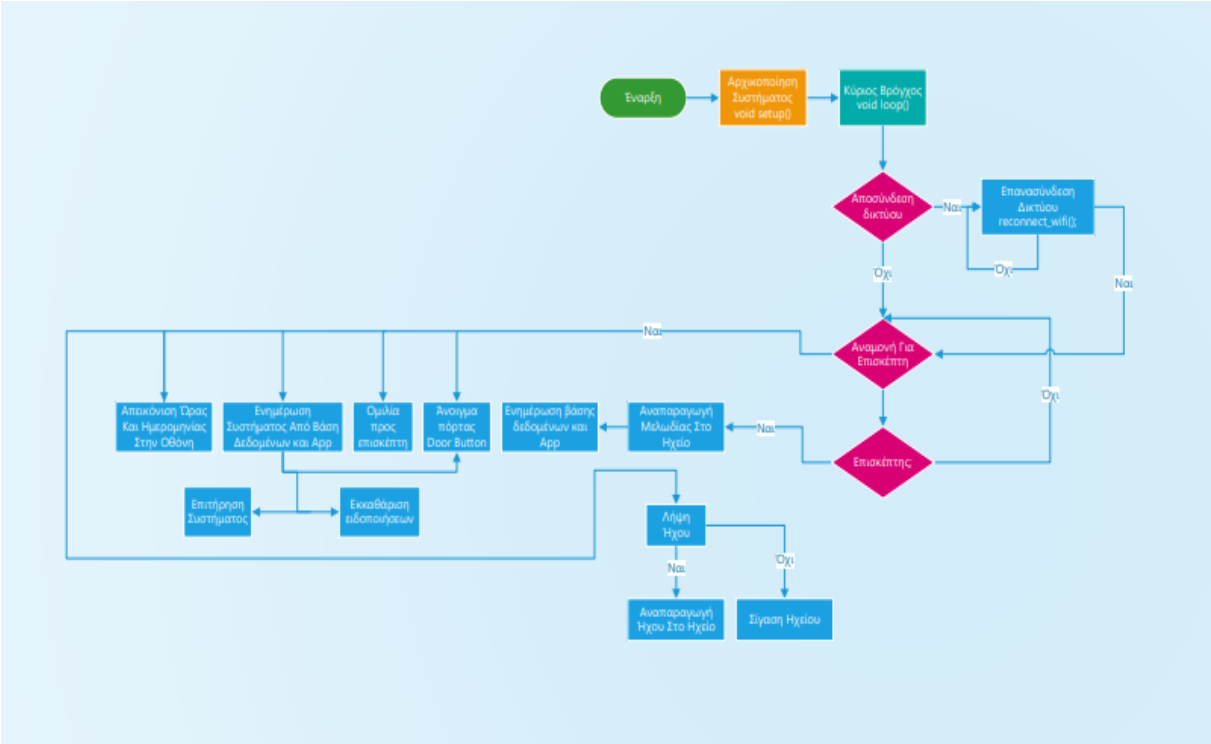
μέγεθος) από το κουμπί κλήσης και αποθηκεύονται σε μια μεταβλητή (len). Εάν η τιμή που επιστρέφει το κουμπί είναι μεγαλύτερο του μηδενός και η μεταβλητή “melody_signal” ισούται με δύο (== 2) και δεν παίζει ήδη μελωδία, τότε ενημερώνεται το Firebase ότι υπάρχει επισκέπτης μέσω της συνάρτησης “people()”, μηδενίζεται το σήμα της μελωδίας, ανάβει το κόκκινο led και εκτυπώνεται στην οθόνη το κείμενο “Visitor Speaking”. Ταυτόχρονα καλείται η συνάρτηση “playSineWave()” και οι “no_people()”, “set_notificatoin()” για να δηλωθεί ότι δεν υπάρχει νέος επισκέπτης και για να σταλθεί ειδοποίηση.

Μέχρι στιγμής στην “void_loop()” αναφερόμαστε σε συνθήκες παρόλαυτα το σύστημα όσο δεν συμβαίνει κάποιο γεγονός, η μεταβλητή “isMelodyPlaying” είναι “false” και μαζί με αυτήν καλείται η συνάρτηση “silence_speaker()” έτσι ώστε να μην υπάρχει θόρυβος από το ηχείο και εκτυπώνει στην οθόνη το κείμενο “Waiting…”.

Σε περίπτωση που πατηθεί το Speak Button και δεν μιλάμε ήδη, τότε εκτυπώνεται στην οθόνη το κείμενο “You Can Speak With Visitor”, ανάβει το πράσινο led, διαβάζονται τα δεδομένα από το μικρόφωνο και αποστέλονται προς τον πομπό, αλλιώς επιστρέφει στην κατάσταση αναμονής.

Στην συνέχεια γίνεται έλεγχος εάν έχουν αλλάξει τα δεδομένα των μεταβλητών από την εφαρμογή Android. Αν η αλλαγή αφορά το άνοιγμα της πόρτας και το “stream_data” ισούται με ένα τότε καλείται η συνάρτηση “void door()”. Αλλιώς αν η αλλαγή αφορά την ειδοποίηση και η τιμή του “stream_data” ισούται με μηδέν καλείται η συνάρτηση “reset_notification()” η οποία σβήνει τις ειδοποιήσεις στον δέκτη. Στην αντίθετη περίπτωση, δηλαδή αν το “stream_data” ισούται με ένα τότε καλείται η “set_notifications()” η οποία ενεργοποιεί την ειδοποίηση στον δέκτη. Μετά καθαρίζεται η μεταβλητή “stream_data” για την επόμενη αλλαγή.

Τέλος αν πατηθεί το κουμπί ανοίγματος πόρτας (Door Open) τότε καλείται και πάλι η συνάρτηση “void door()”. Σε αυτή την συνάρτηση εμπεριέχεται η συνάρτηση “Open_The_Door()” η οποία στέλνει το σήμα στον πομπό για να ανοίξει η πόρτα, όπως επίσης ενημερώνεται και η μεταβλητή στο Firebase. Ταυτόχρονα εμφανίζεται στην οθόνη το κείμενο “Door Open! Access Granted”, ανάβει το κόκκινο Led και μετά απο πέντε δευτερόλεπτα καθαρίζει η οθόνη και ενημερώνεται το Firebase για το κλείσιμο της πόρτας και για την επαναφορά των ειδοποιήσεων μέσω των συναρτήσεων “close_door()” και “reset_notification” αντίστοιχα.



Εικόνα 4-2:Διάγραμμα ροής κώδικα δέκτη.

4.2 Επικοινωνία Μονάδων-Εφαρμογής (Firebase)

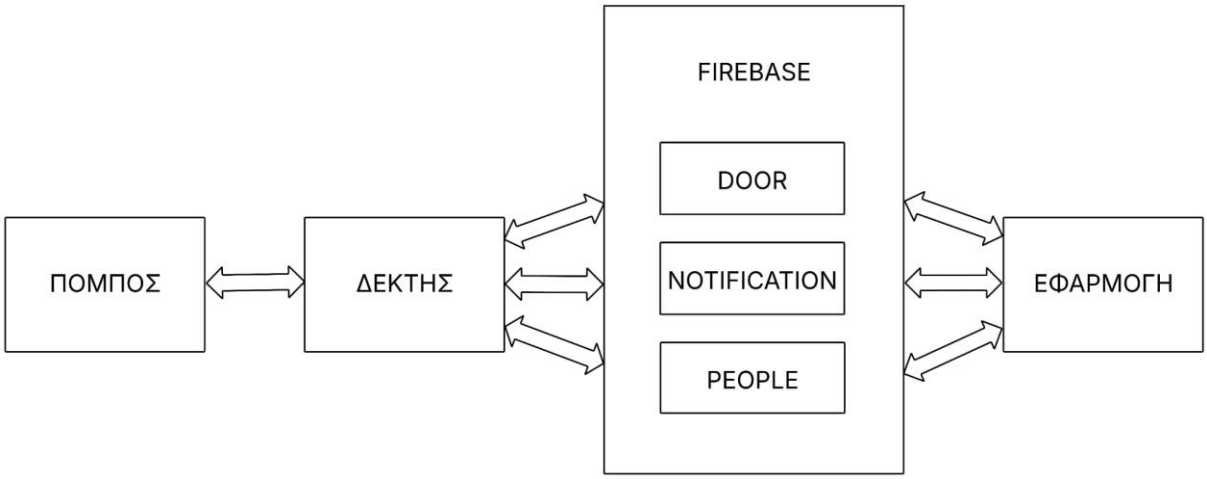
Το Firebase είναι μια πλατφόρμα ανάπτυξης εφαρμογών για κινητά και web αγορασμένη απο την Google το 2014. Αποτελεί απόγονο της εφαρμογής Envolve δημιουργημένη από τους Andrew Lee και James Tamplin. Η εφαρμογή Envolve ήταν εφαρμογή chat, παρόλα αυτά οι χρήστες την χρησιμοποιούσαν για την μεταφορά δεδομένων εφαρμογών. Έτσι ο Tamplin και Lee αποφάσισαν να δημιουργήσουν μια βάση δεδομένων πραγματικού χρόνου. Το Firebase παρέχει εργαλεία και υπηρεσίες που βασίζονται στο cloud, τα οποία διευκολύνουν την δημιουργία, βελτίωση και ανάπτυξη εφαρμογών.

Μερικά απο αυτά τα εργαλεία είναι:

- Βάση δεδομένων πραγματικού χρόνου: Βάση δεδομένων NoSQL που επιτρέπει την αποθήκευση και τον συγχρονισμό των δεδομένων μεταξύ των χρηστών σε πραγματικό χρόνο.
- Firestore: Αποτελεί μια αναβαθμισμένη έκδοση του Realtime Database προσφέροντας προηγμένο querying και καλύτερη επεκτασιμότητα.
- Αυθεντικοποίηση: Ευκολόχρηστα εργαλεία για την διαχείρηση της επικύρωσης στοιχείων με e-mail και κωδικό, μέσω Google, Facebook, X κ.α.
- Cloud χώρο αποθήκευσης: Αποθηκεύει και προσφέρει περιεχόμενο όπως φωτογραφίες, video και αρχεία με ασφάλεια.
- Cloud Functions: Είναι μια υπηρεσία στην οποία ο προγραμματιστής δεν χρειάζεται να διαχειρίζεται τον δίκο του server.

- Hosting: Ασφαλής και γρήγορη φιλοξενία για διαδυκτυακές εφαρμογές και μικρουπηρεσίες.
- Analytics: Το Google Analytics χρησιμοποιείται για την παρακολούθηση της συμπεριφοράς των χρηστών και την απόδοση της εφαρμογής.
- Push Notifications: Διαχειρίζεται ειδοποιήσεις και μηνύματα μεταξύ χρηστών σε διάφορες πλατφόρμες.

Πρακτικά σηκώνει το δύσκολο φορτίο μιας εφαρμογής και αυτό με τα περισσότερα εργαλεία να είναι δωρεάν. Στην παρούσα εργασία χρησιμοποιείται σαν δίαυλος επικοινωνίας μεταξύ της μονάδας του δέκτη και της εφαρμογής. Πιο συγκεκριμένα δημιουργήθηκε ένα Database πραγματικού χρόνου που περιέχει τρεις μεταβλητές. Τις Door, Notification και People. Η μεταβλητή Door διαχειρίζεται το άνοιγμα της πόρτας, δηλαδή όταν ο χρήστης ανοίξει την πόρτα μέσω της εφαρμογής,αυτή παίρνει λογική τιμή 1. Έπειτα ο δέκτης διαβάζει την αλλαγή κατάστασης από το Database και ενημερώνει τον πομπό για να ανοίξει την πόρτα. Κάθε φορά που κάποιος επισκέπτης χτυπάει το κουδούνι, ο πομπός ενημερώνει τον δέκτη και αυτός με σειρά του αλλάζει την κατάσταση του Notification και του People του Database σε λογικό 1. Έπειτα η μεταβλητή People γινεται κατευθείαν λογικό 0 ενώ η εφαρμογή διαβάζει την αλλαγή κατάστασης και την εμφανίζει στο UI της εφαρμογής. Μόλις ο χρήστης ¨διαβάσει¨ την ειδοποίηση τότε η μεταβλητή Notification παίρνει ξανά λογική τιμή 0.



Εικόνα 4-3 Διάγραμμα ροής επικοινωνίας συστήματος..

Για την αρχικοποίηση και την ρύθμιση της σύνδεσης μεταξύ Firebase και ESP δημιουργήθηκε ένα αρχείο με όνομα FIREBASE.h. Αρχικά συμπεριλαμβάνεται η βιβλιοθήκη «Firebase_ESP_Client.h» όπου εμπεριέχει βασικές λειτουργίες για την επικοινωνία του μικροελεγκτή με το Firebase. Μαζί με την βιβλιοθήκη έρχονται τα βοηθητικά αρχεία «TokenHelper.h και RTDBHelper.h» όπου αυτά βοηθούν στην αυθεντικοποίηση και στον χειρισμό λειτουργιών βάσης δεδομένων πραγματικού χρόνου. Έπειτα καθορίζεται το μοναδικό για την εργασία API_KEY και DATABASE_URL. Το API_KEY αυθεντικοποιεί ετήματα σχετικά με την Firebase εργασία και επιτρέπει στον μικροελεγκτή να έχει πρόσβαση με ασφάλεια στις υπηρεσίες του Firebase. Το DATABASE_URL είναι μια διεύθυνση που κατευθήνει απευθείας στην βάση δεδομένων πραγματικού χρόνου.

Έπειτα δημιουργούνται έξι μεταβλητές τύπου string οι οποίες είναι πολύ σημαντικές για την επικοινωνία του συστήματος με το Firebase:

1. door_intercom_system: Πρόκειται για μια αναγνωριστική μεταβλητή που χρησιμοποιείται για να ορίζει ένα μοναδικό όνομα στο Database του Firebase. Ουσιαστικά αποτελεί την ¨ρίζα¨ ή τονν κύριο κόμβο κάτω από τον οποίο αποθηκεύονται όλα τα δεδομένα που αφορούν το σύστημα θυροτηλεφώνου.
2. stream_path: Αυτή η μεταβλητή μας δείχνει ποιό μέρος της βάσης δεδομένων παρακολουθούμε για αλλαγές.
3. event_path: Αυτή η μεταβλητή μας δείχνει σε ποιό σημείο ακριβώς έγινε η αλλαγή μέσα στο Database.
4. stream_data: Η μεταβλητή αυτή αποθηκεύει τα δεδομένα που έρχονται από το Firebase μέσω της διαδικασίας streaming. Ουσιαστικά περιέχει την τιμή που άλλαξε στο συγκεκριμένο path.
5. bucketData: Το bucketData χρησιμοποιείται για την αποθήκευση πληροφοριών σχετικά με το Firebase Storage Bucket. Στην περίπτωση μας αποθηκεύονται οι λογικές καταστάσεις των μεταβλητών μας.
6. bucketPath: Η μεταβλητή bucketPath αποθηκεύει το μονοπάτι των μεταβλητών μας στο Firebase Storage.

Έπειτα πραγματοποιούνται κάποιες βασικές αρχικοποιήσεις του Firebase. Αυτές οι αρχικοποιήσεις χρειάζονται για να μπορούμε να στείλουμε η να διαβάσουμε αρχεία από το Firebase, να γνωρίζουμε εάν έχουν έρθει νέα δεδομένα απο το Firebase, εάν έχει γίνει επιτυχώς η σύνδεση όπως και να παρακολουθούμε αλλαγές σε πραγματικό χρόνο. Επίσης αρχικοποιούνται οι λειτουργίες για γενικές λειτουργίες ανάγνωσης δεδομένων από και προς Firebase όπως και η ρυθμίσεις του project του Firebase.

Στην συνέχεια δημιουργείται μια συνάρτηση η οποία στέλνει έναν αριθμό σε ένα συγκεκριμένο URL μέσα στην βάση δεδομένων. Σε αυτήν γίνεται έλεγχος εάν το Firebase είναι έτοιμο και σε περίπτωση που το Firebase δεν είναι έτοιμο η αποτύχει η σύνδεση τότε εμφανίζεται το μήνυμα "Write Failed: Firebase not ready OR signup not OK" στην σειριακή θύρα. Έπειτα προστίθεται η ¨ρίζα¨ μπροστά από το URL και ελέγχεται στην σειριακή θύρα. Μετά προστίθεται το URL στην αρχική διαδρομή ¨door_intercom_system¨ για να δημιουργηθεί η τελική διαδρομή (path) αποθήκευσης στην βάση. Αν η λειτουργία είναι επιτυχής, εμφανίζονται σχετικές πληροφορίες για την επιτυχία. Αν αποτύχει εμφανίζεται ο λόγος αποτυχίας.

Υπάρχουν εφτά συναρτήσεις για να ανεβάσουν τις αλλαγές μεταβλητών στο Firebase. Αυτές είναι οι εξής. Η people() η οποία στέλνει λογικό 1 στο μονοπάτι people του Firebase δηλαδή ότι υπάρχει επισκέπτης, η no_people() η οποία στέλνει λογικό 0 όταν δεν υπάρχει επισκέπτης. Η set_notification() στέλνει 1 στο μονοπάτι του Notification όταν υπάρχει ειδοποίηση και η reset_notification η οποία στέλνει λογικό 0 και απενεργοποιεί το notification και τέλος το ¨open_door()/close_door() που ρυθμίζει την μεταβλητή της πόρτας.

Τέλος για την λήψη και την αποστολή της IP από συσκευή σε συσκευή δημιουργήθηκαν δύο συναρτήσης, η “Set_IP” και “Get_IP”. Η “Set_IP” και στις δύο συσκευές (πομπός/δέκτης) αποστέλει την IP τους στο Firebase. Από την άλλη η “Get_IP” διαβάζει την IP από το Firebase και μετατρέπει την συμβολοσειρά σε τέσσερις ακέραιους αριθμούς (της μορφής x.x.x.x) . Είτε σε περίπτωση επιτυχίας ή αποτυχίας εκτυπώνεται αντίστοιχο μύνημα στην σειριακή θύρα.

4.3 MIT App Inventor-Εφαρμογή Android

Το MIT App Inventor είναι μια δωρεάν online πλατφόρμα στην οποία δίνεται η δυνατότητα δημιουργίας εφαρμογών Android και IOS (κατά κύριο λόγο Android) χωρίς την ανάγκη γνώσεων προγραμματισμού. Η γλώσσα που χρησιμοποιείται είναι υψηλού επιπέδου και βασισμένη σε μπλοκ κώδικα κάτι που την κάνει μια οπτική γλώσσα προγραμματισμού. Αρχικά η πλατφόρμα δημιουργήθηκε απο την Google και πλέον συντηρείται απο το πανεπιστήμιο του MIT(Massachusetts Institute of Technology). Χρησιμοποείται κατά κόρων στην εκπαίδευση και στην ανάπτυξη απλών εφαρμογών. Χωρίζεται σε δυο μέρη, το designer και το block editor. Στο designer δημιουργείται η διεπαφή χρήστη(User Interface) με τα στοιχεία (κουμπιά, ετικέτες, εικόνες, κ.α.) να είναι drag and drop. Στο block editor δημιουργείται η λογική της εφαρμογής και αντί να γράφεται κώδικας συνδέονται block εντολών, μεταβλητών κ.α. μεταξύ τους.

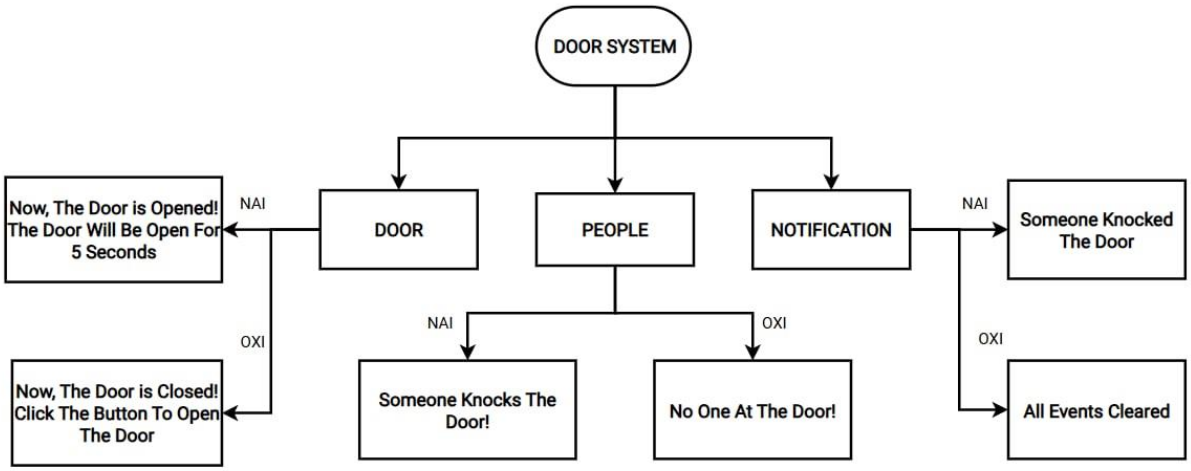
Το Designer χωρίζεται σε πέντε κυρίως τμήματα. Την παλέτα, που περιέχει όλα τα διαθέσιμα στοιχεία οργανωμένα σε κατηγορίες όπως User Interface (κουμπιά, εικόνες κ.α.), Layout (οριζόντιες η κατακόρυφες διατάξεις για οργάνωση της οθόνης), Media (ήχος, βίντεο, κάμερα), Drawing and Animation (καμβάς για σχέδιο, σχήματα, κινούμενα σχέδια), Sensors (επιτάχυνση, τοποθεσία GPS, πυξίδα), Social (αποστολή SMS, e-mail), Connectivity (Bluetooth, web e.t.c) και το Experimental/Extensions (πειραματικά η εξωτερικά στοιχεία).

Το δεύτερο μέρος αποτελέι την οθόνη σχεδιασμού στην οποία γίνονται drag n drop στοιχεία από την παλέτα με σκοπό να δημιουργηθεί η οθόνη της εφαρμογής.

Το τρίτο μέρος είναι τα Components. Εκεί φαίνονται όλα τα στοιχεία που έχουν προστεθεί στην οθόνη. Πρακτικά είναι μια λίστα του τι υπάρχει στην εφαρμογή.

Τέταρτο μέρος αποτελούν τα Properties. Δηλαδή εκεί μπορούν να αλλάξουν οι ιδιότητες ενός στοιχείου δηλαδή όνομα, μέγεθος, χρώμα κ.α.

Πέμπτο και τελευταίο μέρος είναι το Media. Εκεί φαίνονται εικόκες, ήχοι και αρχεία που έχουν ανέβει στο Project. Επόμενο βήμα μετά την εξερεύνηση του Designer είναι η ανάλυση του Block Editor το οποίο χωρίζεται σε τρία μέρη στα Blocks, Media και Viewer. Το Media και το Viewer έχουν ακριβώς την ίδια χρήση με το Designer. Τα Blocks αποτελούν τον κώδικα στην μορφή χρωματιστών block που κουμπώνουν μεταξύ τους σαν παζλ. Το κάθε χρώμα αντιπροσωπεύει και κάποια συγκεκριμένη λειτουργία.



Εικόνα 4-4:Διάγραμμα ροής κώδικα Android.

Αρχικά για τις ανάγκες της εφαρμογής δημιουργήθηκε πρώτα το UI (User Interface). Η αρχική οθόνη της εφαρμογής είναι και μόνη που διαθέτει. Στην κορυφή αυτής μέσα σε ένα οριζόντιο δοχείο διάταξης (τα οποία δοχεία χρησιμοποιούνται για την τοποθέτηση άλλων στοιχείων όπως κουμπιά, ετικέτες, εικόνες κ.α.) τοποθετήθηκε κείμενο ¨Door Intercom System Based On ESP32¨. Παρακάτω, προστίθεται ένα ακόμη οριζόντιο δοχείο στο οποίο περιέχεται το κουμπί "Clear Events". Η λειτουργία αυτού του κουμπιού είναι να στέλνει λογικό 0 στην μεταβλητή "Notifications" του Firebase. Στο ίδιο δοχείο δίπλα απο το κουμπί υπάρχουν μια ετικέτα κειμένου και μια φωτογραφία. Η ετικέτα αλλάζει απο "All events cleared" σε "Someone Knocked The Door" όταν ενημερωθεί η μεταβλητή του Firebase "Notifications" σε λογικό 1 απο το δέκτη του θυροτηλεφώνου. Ταυτόχρονα η φωτογραφία ενημερώνεται μαζί με τις αλλαγές όπως με την ετικέτα, δηλαδή όταν η μεταβλητή του "Notifications" αλλάζει σε 1 τότε εμφανίζεται ενα μπλέ καμπανάκι και όταν πατάμε το κουμπί "Clear events" της εφαρμογής το καμπανάκι αλλάζει σε ενα «τικ». Στη συνέχεια σε ένα κάθετο δοχείο, μεγαλύτερο απο τα προηγούμενα φαίνεται μια ετικέτα κειμένου στην οποία αναγράφεται "No One At The Door!" και μια εικόνα που δείχνει πως δεν υπάρχει επισκέπτης, όταν η μεταβλητή απο το Firebase "people" είναι 0. Όταν υπάρχει επισκέπτης στην πόρτα η μεταβλητή "people" στο Firebase γίνεται 1 και κατά συνέπεια αλλάζει η τιμή στην εφαρμογή με αποτέλεσμα το κείμενο να αλλάξει σε "Someone Knocks The Door!" και η εικόνα να εμφανίζει ανθρώπους. Όσον αφορά το άνογμα της πόρτας έχουν δημιουργηθεί 2 δοχεία. Το ένα περιέχει μια ετικέτα κειμένου που αναγράφει "Now, The Door Is Closed" και μία φωτογραφία με ένα κόκκινο led. Το άλλο δοχείο που αφορά το άνοιγμα της πόρτας περιέχει το κείμενο " Click The Button To Open The Door" και το κουμπί (το οποίο περιέχει την φωτογραφία μιας πόρτας) που ανοίγει την πόρτα. Όταν πατηθεί το κουμπί τότε το led αλλάζει χρώμα σε πράσινο και το κείμενο του γίνεται "Now , The Door Is Opened" και το κείμενο του άλλου δοχείου "The Door Will Be Open For 5 Seconds". Τέλος στο κάτω μέρος της εφαρμογής υπάρχουν τα στοιχεία μας και η έκδοση της εφαρμογής μας.

	Έλεγχος Ροής	Μπλοκ για if, if else, for, while, when, wait κ.λ.π.
	Λογική	Μπλοκ για λογικές συνθήκες: true/false, =, and, or, not.
	Μαθηματικά	Μπλοκ για αριθμούς και πράξεις: +, -, *, /, mod, sqrt, random.
	Κείμενο	Μπλοκ για συμβολοσειρές (text/strings): "Hello", join,
	Λίστες	Μπλοκ για λίστες τιμών: create list, add item, select item
	Λεξικά	Μπλοκ για key:value ζευγάρια
	Χρώματα	Μπλοκ για επιλογή χρώματος π.χ. make color, red κ.λ.π
	Μεταβλητές	Μπλοκ για αποθήκευση και αλλαγή δεδομένων. initialize global, set, get.
	Συναρτήσεις	Μπλόκ απο υπορουτίνες

Πίνακας 4-1:Πίνακας λειτουργιών MIT APP INVENTOR.

Έπειτα μετά την ολοκλήρωση του UI επόμενο βήμα είναι ο προγραμματισμός αυτού. Στο βασικό κομμάτι του κώδικα το οποίο δέχεται στοιχεία απο το Database του Firebase, αρχίκα πραγματοποιείται έλεγχος αν έχει αλλάξει κάποια μεταβλητή. Εφόσον διαπιστωθεί κάποια αλλαγή τότα πρώτα ελέγχεται αν το tag που άλλαξε είναι αυτό που μας ενδιαφέρει, δηλαδή το ¨Door System¨. Οπότε ουσιαστικά ελέγχει εάν άλλαξε κάτι και το που άλλαξε κάτι στην βάση δεδομένων. Στην συνέχεια γίνεται η μετατροπή του JSON σε λεξικό (dictionary) και επίσης γίνεται αναζήτηση για όλους τους χαρακτήρες «\"» και τους αντικαθιστά με κανονικά «"» ώστε το κείμενο να γίνει έγκυρο JSON.

Μετέπειτα πραγματοποιεί αναζήτηση σε ποια ακριβώς από τις τρεις μεταβλητές (Door, Notification, People) έχει γίνει αλλαγή κατάστασης και ανάλογα με το ποια μεταβλητή αλλάζει ακολουθείται μια σειρά από εντολές. Στην περίπτωση που η μεταβλητή Door γίνει λογικό 1 τότε γίνοται οι αλλαγές που έχουν αναφερθεί στο UI . Στην περίπτωση αλλαγής της μεταβλητής People γίνονται οι προαναφερθείσες αλλαγές του UI με την διαφορά οτι στέλνεται μια δόνηση στο κινητό του χρήστη. Τέλος, το ίδιο ισχύει και για την μεταβλητή Notification.

Πέρα από τον βασικό κομμάτι που δέχεται στοιχεία υπάρχουν και δύο συναρτήσεις η οποίες στέλνουν δεδομένα προς το Firebase. Η μια απο τις δύο είναι υπεύθυνη για το κουμπί ανοίγματος της πόρτας και στέλνει την τιμή ένα στην αντίστοιχη μεταβλητή του Firebase έτσι ώστε να ενημερωθεί τελευταίος ο

πομπός και να ανοίξει την πόρτα. Η δεύτερη συνάρτηση ευθύνεται για τον καθαρισμό των ειδοποιήσεων και όταν πατηθεί το κουμπί (Clear Events) στέλνει την τιμή μηδέν στο Firebase.

Εικόνα 4-5:UI (User interface) Android εφαρμογής.

Συμπεράσματα και Βελτιώσεις

Η παρούσα πτυχιακή εργασία είχε ως στόχο την ανάπτυξη ενός έξυπνου θυροτηλεφώνου, το οποίο αξιοποιεί σύγχρονες τεχνολογίες όπως το ESP32, επικοινωνία μέσω Wi-Fi και εφαρμογή για διασύνδεση σε κινητό τηλέφωνο. Μέσα από τον σχεδιασμό και την υλοποίηση του συστήματος, έγινε κατανοητό πως ακόμη και με περιορισμένους πόρους, είναι εφικτή η δημιουργία λειτουργικών και προσιτών λύσεων για καθημερινές ανάγκες όπως η επικοινωνία με επισκέπτες σε μια κατοικία.

Το σύστημα επιτυγχάνει βασικές λειτουργίες όπως ειδοποίηση μέσω κουμπιού, αποστολή ήχου από δύο μονάδες, και διαχείριση βασικών λειτουργιών μέσω κινητού τηλεφώνου. Η χρήση του ESP32 αποδείχθηκε ιδανική για εφαρμογές αυτού του τύπου λόγω της υποστήριξης Wi-Fi και της χαμηλής κατανάλωσης ενέργειας.

Ωστόσο, υπάρχουν αρκετά σημεία που θα μπορούσαν να βελτιωθούν στο μέλλον. Συγκεκριμένα:

- Διαχείριση πολλαπλών χρηστών ή μονάδων: Η παρούσα υλοποίηση αφορά ένα απλό σενάριο. Στο μέλλον θα μπορούσε να επεκταθεί ώστε να υποστηρίζει περισσότερους χρήστες ή σύνδεση με περισσότερες εξωτερικές μονάδες.
- Επικοινωνία ήχου από και προς το κινητό τηλέφωνο: Καθώς το σύστημα αυτό δεν προσφέρει ομιλία από την συσκευή κινητού τηλεφώνου μια τέτοια βελτίωση θα ήταν πολύ χρήσιμη. Αυτή θα μπορούσε να επιτευχθεί με την χρήση της τεχνολογίας WebRTC η οποία διευκολύνει την επικοινωνία “peer to peer”.
- Προσθήκη κάμερας: Μια τέτοια προσθήκη θα εξυπηρετούσε προβλήματα ασφαλείας καθώς θα υπήρχε οπτική επαφή με την κεντρική είσοδο.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Paul Black Totally Tubular, “A History of the Development of Intercom Monday,” June 26, 2017.

[2] Johanna Gruber, “The History of the Apartment Intercom System,” April 4, 2025, Los Angeles.

[3] Camilla Ashdown, “Echoes of Innovation: The Evolution of IP Intercoms,” 06 October 2023, **www.2n.com**.

[4] “What is Intercom? Unveiling the Power of Advanced Communication Systems,” 14 May 2023, **bas-ip.com**.

[5] Michael Hines, “Everything Guide for Intercom and Telephone Entry Systems (2025),” March 11, 2025, swiftlane.com.

[6] **https://www.friendsofdalnavert.ca/blog/2019/1/10/the-speaking-tube**

[7] Jennifer Ouellette, “Can You Hear Me Now? Sound Technology of the 19th Century,” July 23, 2012.

[8] **https://www.urmet.com/en-us/Professional/Products/Video-door-phone/Video-door-phone-systems/4-n-Wire-system**

[9] BPT YC200 Intercom Handset Data Sheet.

[10] Forward Lumbwe, “Intercom systems — Purpose, Types and how do they work?,” 29 January 2020.

[11] Τεχνικό εγχειρίδιο συμβατικού θυροτηλεφώνου TEGUI 4+N.

[12] https://en.wikipedia.org/wiki/Visual_Studio#References

[13] https://en.wikipedia.org/wiki/ESP32#Reception_and_use

[14] https://www.redeweb.com/el/παρόν/τι-είναι-ένας-μικροελεγκτής/

[15] I2S Protocol : Working, Differences & Its Applications **https://www.elprocus.com/i2s-protocol/**.

[16] ESP 32 DEVKIT **https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html#power-supply-options.**

[17] ESP32 Series Datasheet Version 4.9.

[18] INMP 441 **https://components101.com/modules/inmp441-mems-omnidirectional-microphone**.

[19] MAX98357A **https://www.adafruit.com/product/3006**.

ΠΑΡΑΡΤΗΜΑ Α: ΚΩΔΙΚΑΣ ΠΟΜΠΟΎ

```
#include "SOUND.h"
#include "VARIABLES.h"
#include "INIT_GPIO.h"
#include "FIREBASE.h"
#include <WiFi.h>
#include <WiFiUdp.h>
#include <Firebase_ESP_Client.h>

WiFiUDP udp_receive_audio;  // UDP for speaker
WiFiUDP udp_transmit_audio; // UDP for microphone
WiFiUDP udp_lock;           // UDP for lock led
WiFiUDP udp_call_button;    // UDP for receiver calling
String ipStr;

void setup()
{
  Serial.begin(115200);
  Init_Pins();

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.println("Connecting to WiFi...");
  }

   digitalWrite(WIFI_LED_PENDING_CONNECT, HIGH); // RED LED OFF Reverse logic due to common anode rgb led
   digitalWrite(WIFI_LED_CONNECTED, LOW);          // GREEN LED ON Reverse logic due to common anode rgb led
  Serial.print("Connected to WiFi network with IP Address: ");
```

```
Serial.println(WiFi.localIP());

// Firebase config
config.api_key = API_KEY;
config.database_url = DATABASE_URL;

Firebase.begin(&config, &auth);
Firebase.reconnectWiFi(true);

// Anonymous sign in
if (Firebase.signUp(&config, &auth, "", ""))
{
  Serial.println("Anonymous sign up success");
}
else
{
  Serial.printf("Sign up failed: %s\n", config.signer.signupError.message.c_str());
}

ipStr = WiFi.localIP().toString(); // load Wifi IP in ipStr variable to upload on firebase
Set_IP(ipStr);                  // Set transmitter IP on firebase
Get_IP();                       // Get receiver IP from firebase

// begin Udp communication ports
udp_receive_audio.begin(Receive_Audio_Port);
udp_transmit_audio.begin(Transmit_Audio_Port);
udp_lock.begin(Lock_Port);
udp_call_button.begin(Call_Port);

delay(1000);

// setup for speaker and microphone
i2s_mic_install();
```

```
  i2s_mic_setpin();
  i2s_speaker_install();
  i2s_speaker_setpin();

  // start I2S ports
  i2s_start(I2S_MIC_PORT);
  i2s_start(I2S_SPEAKER_PORT);
}

void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    reconnect_wifi();
    ipStr = WiFi.localIP().toString(); // load Wifi IP in ipStr variable to upload on firebase
    Set_IP(ipStr);                    // Set transmitter IP on firebase
    Get_IP();                         // Get receiver IP from firebase
  }

  // scan for sound from receiver to send on speaker
  uint8_t audio_buffer[BUFFER_SIZE];
  int audioPacketSize = udp_receive_audio.parsePacket();
  if (audioPacketSize > 0)
  {
    int len = udp_receive_audio.read(audio_buffer, sizeof(audio_buffer));
    if (len > 0) // send sound from receiver microphone to speaker
    {
      size_t bytes_written;
      i2s_write(I2S_SPEAKER_PORT, audio_buffer, len, &bytes_written, portMAX_DELAY);
      Serial.printf("Received %d bytes of audio\n", len);
    }
  }
  else // silence
```

```
{
  silence_speaker();
}

// receive signal from receiver to open the door
uint8_t led_signal;
int ledPacketSize = udp_lock.parsePacket();
if (ledPacketSize > 0)
{
  int len = udp_lock.read(&led_signal, sizeof(led_signal));
  if (len > 0 && led_signal == 1)
  {
    digitalWrite(WHITE_LED_PIN, HIGH); // open the door
    digitalWrite(BUZZER_PIN, HIGH);    // buzzer on
    delay(5000);                       // delay 5 seconds
    digitalWrite(WHITE_LED_PIN, LOW);  // close the door
    digitalWrite(BUZZER_PIN, LOW);     // buzzer off
  }
}

// we can call the receiver while press the SPEAK_BUTTON_PIN
  if (!digitalRead(CALL_PIN))
  {
    delay(200); // Debounce delay
    sendMelodySignal();
  }

  // we can speak while press the SPEAK_BUTTON_PIN
  if (!digitalRead(SPEAK_BUTTON_PIN))
  {
    if (!isSending)
    {
      isSending = true;
```

```
      digitalWrite(BLUE_LED_PIN, HIGH);
    }

   uint8_t mic_buffer[BUFFER_SIZE];
   size_t bytes_read;
   i2s_read(I2S_MIC_PORT, mic_buffer, sizeof(mic_buffer), &bytes_read, portMAX_DELAY);
   udp_transmit_audio.beginPacket(receiverIP, Transmit_Audio_Port); // send to receiver udp port
   udp_transmit_audio.write(mic_buffer, bytes_read);              // write the data
   udp_transmit_audio.endPacket();                                // end of communication
   Serial.printf("Sent %d bytes of audio\n", bytes_read);
  }
  else if (isSending)
  {
   isSending = false;
   digitalWrite(BLUE_LED_PIN, LOW);
  }

  vTaskDelay(1); // Δώσε χρόνο σε άλλα tasks
}

void sendMelodySignal()
{
  uint8_t call_signal = 2;                          // Το σήμα για το CALL στον δέκτη
  udp_call_button.beginPacket(receiverIP, Call_Port);       // send to receiver udp port
  udp_call_button.write(&call_signal, sizeof(call_signal)); // write the data
  udp_call_button.endPacket();                          // end of communication
  Serial.println("Melody signal sent to receiver");
  delay(500);
  call_signal = 10; // WRONG VALUE FOR 1 TIME MELODY PLAYING
}

void reconnect_wifi()
{
```

```
  WiFi.disconnect();
  WiFi.begin(ssid, password);
   digitalWrite(WIFI_LED_PENDING_CONNECT, HIGH); // RED LED OFF Reverse logic due to common anode rgb led
   digitalWrite(WIFI_LED_CONNECTED, HIGH);        // GREEN LED OFF Reverse logic due to common anode rgb led

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
     digitalWrite(WIFI_LED_PENDING_CONNECT, LOW); // RED LED ON Reverse logic due to common anode rgb led
  }
}
```

GPIO.h

```
// Pins I2S for microphone
#define I2S_MIC_CLK              26
#define I2S_MIC_WS               25
#define I2S_MIC_DATA             23

// Pins I2S for speaker
#define I2S_SPK_CLK              19
#define I2S_SPK_WS               18
#define I2S_SPK_DATA              5

// Leds
#define WHITE_LED_PIN             2  // led for door opening
#define BLUE_LED_PIN             12 // led when speaks

// button for speaking
```

```
#define SPEAK_BUTTON_PIN          4

// Button for door opening
#define CALL_PIN                  13

// Led for Wi-Fi status connection
#define WIFI_LED_CONNECTED        33 // wi-fi connected
#define WIFI_LED_PENDING_CONNECT  27 // wifi not connected

#define BUZZER_PIN 32
```

INIT_GPIO.h

```
#include "GPIO.h"
#include "Arduino.h"

void Init_Pins()
{
   // configuration pins as input or output and set the first status
   pinMode(CALL_PIN, INPUT);
   pinMode(WHITE_LED_PIN, OUTPUT);
   pinMode(SPEAK_BUTTON_PIN, INPUT);
   pinMode(BLUE_LED_PIN, OUTPUT);
   pinMode(WIFI_LED_CONNECTED, OUTPUT);
   pinMode(WIFI_LED_PENDING_CONNECT, OUTPUT);
   pinMode(BUZZER_PIN, OUTPUT);
   digitalWrite(BUZZER_PIN, LOW);
   digitalWrite(WHITE_LED_PIN, LOW);
   digitalWrite(BLUE_LED_PIN, LOW);
    digitalWrite(WIFI_LED_CONNECTED, HIGH);       // GREEN LED OFF Reverse logic due to common anode rgb led
    digitalWrite(WIFI_LED_PENDING_CONNECT, LOW); // RED LED ON Reverse logic due to common anode rgb led
}
```

SOUND.h

```
#include <driver/i2s.h>
#include "GPIO.h"

#define I2S_SAMPLE_RATE                 16000
#define I2S_SAMPLE_BITS I2S_BITS_PER_SAMPLE_16BIT
#define I2S_CHANNELS                    1
#define BUFFER_SIZE                     256 // Μικρότερο buffer

#define I2S_MIC_PORT I2S_NUM_0
#define I2S_SPEAKER_PORT I2S_NUM_1      // Χρησιμοποιούμε το δεύτερο I2S για το ηχείο

#define BUFFER_LEN                      64  // default was 64

void i2s_mic_install()
{
  const i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
      .sample_rate = I2S_SAMPLE_RATE,
      .bits_per_sample = I2S_SAMPLE_BITS,
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format = I2S_COMM_FORMAT_I2S,
      .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
      .dma_buf_count = 4,       // Μειωμένο dma buffer count
      .dma_buf_len = BUFFER_LEN, // Μειωμένο dma buffer length
      .use_apll = false};
  i2s_driver_install(I2S_MIC_PORT, &i2s_config, 0, NULL);
}

void i2s_mic_setpin()
{
```

```
  const i2s_pin_config_t pin_config = {
      .bck_io_num = I2S_MIC_CLK,
      .ws_io_num = I2S_MIC_WS,
      .data_out_num = I2S_PIN_NO_CHANGE,
      .data_in_num = I2S_MIC_DATA};
  i2s_set_pin(I2S_MIC_PORT, &pin_config);
}

void i2s_speaker_install()
{
  const i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX),
      .sample_rate = I2S_SAMPLE_RATE,
      .bits_per_sample = I2S_SAMPLE_BITS,
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format = I2S_COMM_FORMAT_I2S,
      .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
      .dma_buf_count = 4,       // Μειωμένο dma buffer count
      .dma_buf_len = BUFFER_LEN, // Μειωμένο dma buffer length
      .use_apll = false};
  i2s_driver_install(I2S_SPEAKER_PORT, &i2s_config, 0, NULL);
}

void i2s_speaker_setpin()
{
  const i2s_pin_config_t pin_config = {
      .bck_io_num = I2S_SPK_CLK,
      .ws_io_num = I2S_SPK_WS,
      .data_out_num = I2S_SPK_DATA,
      .data_in_num = I2S_PIN_NO_CHANGE};
  i2s_set_pin(I2S_SPEAKER_PORT, &pin_config);
}
```

```
void silence_speaker()
{
  // set the BUFFER_SIZE with {0} to silence the speaker
  int16_t silence[BUFFER_SIZE] = {0};
  size_t bytes_written;
  i2s_write(I2S_SPEAKER_PORT, silence, sizeof(silence), &bytes_written, portMAX_DELAY);

  // Serial.println("Sine wave stopped.");
}
```

VARIABLES.h

```
// Wi-Fi credentials

const char *ssid = "HUAWEI Mate 20 Pro";
const char *password = "12345678";

// const char *ssid = "";
// const char *password = "";

// Udp ports for communication with transmitter
const int Receive_Audio_Port = 1234;
const int Transmit_Audio_Port = 9999;
const int Lock_Port = 5678;
const int Call_Port = 4321;
bool isSending = false;
```

FIREBASE.h

```
#include <Firebase_ESP_Client.h>
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>

/* 2. Define the API Key */
```

```
#define API_KEY "AIzaSyBYhzZYozt_o4pFjc1BYm72N6vqLUSWonU"
//------------------------------------------------------------------------
/* 3. Define the RTDB URL */
#define DATABASE_URL "intercom-system-ea9a7-default-rtdb.firebaseio.com"
//------------------------------------------------------------------------

FirebaseAuth auth;
FirebaseConfig config;
FirebaseData firebaseData;

IPAddress receiverIP; // Global or local variable as needed

void Set_IP(String ipStr)
{
   if (Firebase.RTDB.setString(&firebaseData, "/devices/esp_sender/ip", ipStr))
   {
      Serial.println("IP address uploaded to Firebase successfully.");
   }
   else
   {
      Serial.print("Failed to upload IP: ");
      Serial.println(firebaseData.errorReason());
   }
}

void Get_IP()
{
   // Διάβασε την IP του sender
   if (Firebase.RTDB.getString(&firebaseData, "/devices/esp_receiver/ip"))
   {
      String ip = firebaseData.stringData();
      Serial.print("Read IP from Firebase: ");
      Serial.println(ip);
```

```
      int ip1, ip2, ip3, ip4;
      if (sscanf(ip.c_str(), "%d.%d.%d.%d", &ip1, &ip2, &ip3, &ip4) == 4)
      {
        receiverIP = IPAddress(ip1, ip2, ip3, ip4);
        Serial.print("Parsed IPAddress: ");
        Serial.println(receiverIP);
      }
      else
      {
        Serial.println("Invalid IP format received from Firebase.");
        // receiverIP = IPAddress(192, 168, 1, 100); // fallback IP
      }
    }
    else
    {
      Serial.print("Failed to read IP: ");
      Serial.println(firebaseData.errorReason());
      // receiverIP = IPAddress(192, 168, 1, 100); // fallback IP
    }
}
```

ΠΑΡΑΡΤΗΜΑ Β: ΚΩΔΙΚΑΣ ΔΈΚΤΗ

```
#include "SOUND.h"
#include "VARIABLES.h"
#include "INIT_GPIO.h"
#include <WiFi.h>
#include <WiFiUdp.h>
#include <Wire.h>
#include "time.h"
#include "FIREBASE.h"
#include "SCREEN.h"

WiFiUDP udp_receive_audio;  // UDP for speaker
WiFiUDP udp_transmit_audio; // UDP for microphone
WiFiUDP udp_lock;           // UDP for lock led
WiFiUDP udp_call_button;    // UDP for visitor
String ipStr;

void streamCallback(FirebaseStream data)
{
  stream_path = data.streamPath().c_str();
  event_path = data.dataPath().c_str();

  if (String(data.dataType().c_str()) == "json")
  {
    jsonData = data.to<FirebaseJson>();
  }
  else
  {
    stream_data = data.stringData();
  }
  Serial.printf("stream path, %s\nevent path, %s\ndata type, %s\nevent type, %s\n\n",
                stream_path,
```

```
            event_path,
            data.dataType().c_str(),
            data.eventType().c_str());
  printResult(data); // see addons/RTDBHelper.h
  Serial.println();


  Serial.printf("Received stream payload size: %d (Max. %d)\n\n",
            data.payloadLength(), data.maxPayloadLength());

  dataChanged = true;
}

void streamTimeoutCallback(bool timeout)
{
  if (timeout)
    Serial.println("stream timed out, resuming...\n");

  if (!stream.httpConnected())
    Serial.printf("error code: %d, reason: %s\n\n", stream.httpCode(),
              stream.errorReason().c_str());
}

void setup()
{
  Serial.begin(115200);
  Init_Pins();
  lcd.init();
  lcd.backlight();
  texts(); // Print text on the screen

  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
```

```
{
  lcd.setCursor(3, 0);
  lcd.print("Connecting"); // Print text on the screen
  lcd.setCursor(3, 1);
  lcd.print("to WiFi..."); // Print text on the screen
  Serial.println("Connecting to WiFi...");
  delay(500);
}

 digitalWrite(WIFI_LED_PENDING_CONNECT, HIGH); // RED LED OFF Reverse logic due to common anode rgb led
 digitalWrite(WIFI_LED_CONNECTED, LOW);        // GREEN LED ON Reverse logic due to common anode rgb led
 lcd.clear();                          // Clear the screen
 lcd.setCursor(3, 0);
 lcd.print("Connected!"); // Print text on the screen
 lcd.setCursor(1, 1);
 lcd.print("Please Wait..."); // Print text on the screen
 Serial.print("Connected to WiFi network with IP Address: ");
 Serial.println(WiFi.localIP());
 Serial.println();
 //------------------------------------------------------------------------
 Serial.printf("Firebase Client v%s\n\n", FIREBASE_CLIENT_VERSION);
 delay(2000);

 /* Assign the api key (required) */
 config.api_key = API_KEY;

 /* Assign the RTDB URL (required) */
 config.database_url = DATABASE_URL;

 /* Assign the callback function for the long running token generation task */
 config.token_status_callback = tokenStatusCallback; // see addons/TokenHelper.h
```

```
//------------------------------------------------------------------------
/*Or Sign up */
if (Firebase.signUp(&config, &auth, "", ""))
{
  Serial.println("Firebase signUp ok");
  signupOK = true;
}
else
{
  Serial.printf("%s\n", config.signer.signupError.message.c_str());
}

Firebase.begin(&config, &auth);
Firebase.reconnectWiFi(true);

ipStr = WiFi.localIP().toString(); // load Wifi IP in ipStr variable to upload on firebase
Set_IP(ipStr);                  // Set receiver IP on firebase
Get_IP();                       // Get transmitter IP from firebase

if (!Firebase.RTDB.beginStream(&stream, door_intercom_system))
  Serial.printf("sream begin error, %s\n\n", stream.errorReason().c_str());
//------------------------------------------------------------------------
Firebase.RTDB.setStreamCallback(&stream, streamCallback, streamTimeoutCallback);

lcd.clear(); // Clear the screen

// begin Udp communication ports
udp_receive_audio.begin(Receive_Audio_Port);
udp_transmit_audio.begin(Transmit_Audio_Port);
udp_lock.begin(Lock_Port);
udp_call_button.begin(Call_Port);

configTime(gmtOffset_sec, daylightOffset_sec, ntpServer);
```

```
  delay(1000);
  displayTime(); // display time on screen

  // setup for speaker and microphone
  i2s_mic_install();
  i2s_mic_setpin();
  i2s_speaker_install();
  i2s_speaker_setpin();

  // start I2S ports
  i2s_start(I2S_MIC_PORT);
  i2s_start(I2S_SPEAKER_PORT);
  reset_notification();
}

void loop()
{
  if (WiFi.status() != WL_CONNECTED)
  {
    reconnect_wifi();
    ipStr = WiFi.localIP().toString(); // load Wifi IP in ipStr variable to upload on firebase
    Set_IP(ipStr);                  // Set receiver IP on firebase
    Get_IP();                       // Get transmitter IP from firebase
  }

  control_indication_led(); // set the rgb led

  uint8_t audio_buffer[BUFFER_SIZE];
  int audioPacketSize = udp_receive_audio.parsePacket();
  uint8_t melody_signal;
  int melodyPacketSize = udp_call_button.parsePacket();

  if (Firebase.ready() && signupOK)
```

```
 {
  if (uploadBucket == true)
  {
   uploadBucket = false;
   String URL = door_intercom_system + "/" + bucketPath;
   Serial.println(URL);
    Serial.printf("Set String... %s\n\n", Firebase.RTDB.setString(&fbdo, URL, bucketData) ? "ok" : fbdo.errorReason().c_str());
  }
  bucketPath = "";
  bucketData = "";
 }
 // scan for sound from transmitter to send on speaker
 if (audioPacketSize > 0)
 {
  int len = udp_receive_audio.read(audio_buffer, sizeof(audio_buffer));
  if (len > 0) // send sound from receiver microphone to speaker
  {
   size_t bytes_written;
   i2s_write(I2S_SPEAKER_PORT, audio_buffer, len, &bytes_written, portMAX_DELAY);
   Serial.printf("Received %d bytes of audio\n", len);
  }
 }
 else if (!isSending)
 {
  if (melodyPacketSize > 0)
  {
   isMelodyPlaying = true; // Ενημέρωσε ότι η μελωδία παίζει
   int len = udp_call_button.read(&melody_signal, sizeof(melody_signal));
   if (len > 0 && melody_signal == 2 && isMelodyPlaying)
   { // Το σήμα για την μελωδία είναι 2 και δεν παίζει ήδη
    // delay(200);
    people();
```

```
      melody_signal = 0;
      digitalWrite(RED_LED_PIN, HIGH);
      lcd.clear(); // Clear the screen
      lcd.setCursor(0, 0);
      lcd.print("Visitor Speaking"); // Print text on the screen

      // uncomment this line on finishing
      playSineWave(); // Παίξε την ημιτονοειδή κυματομορφή για 3 δευτερόλεπτα
      Serial.println("Playing sine wave...");
      no_people();
      set_notification();
    }
  }
  // the system waits for visitor
  isMelodyPlaying = false;
  silence_speaker();
  digitalWrite(RED_LED_PIN, LOW);
  displayTime(); // display time on screen
  lcd.setCursor(0, 0);
  lcd.print(" "); // Print text on the screen
  lcd.setCursor(0, 1);
  lcd.print("Waiting...    "); // Print text on the screen
}

// we can speak while press the SPEAK_BUTTON_PIN
if (!digitalRead(SPEAK_BUTTON_PIN))
{
  if (!isSending)
  {
    isSending = true;
    lcd.clear(); // Clear the screen
    lcd.setCursor(1, 0);
    lcd.print("*You Can Speak"); // Print text on the screen
```

```
   lcd.setCursor(2, 1);
   lcd.print("With Visitor"); // Print text on the screen
   digitalWrite(GREEN_LED_PIN, HIGH);
  }

  // send the sound from receiver microphone to transmitter speaker
  uint8_t mic_buffer[BUFFER_SIZE];
  size_t bytes_read;
  i2s_read(I2S_MIC_PORT, mic_buffer, sizeof(mic_buffer), &bytes_read, portMAX_DELAY);
  udp_transmit_audio.beginPacket(transmitterIP, Transmit_Audio_Port); // send to transmitter udp port
  udp_transmit_audio.write(mic_buffer, bytes_read);                   // write the data
  udp_transmit_audio.endPacket();                                     // end of communication
  Serial.printf("Sent %d bytes of audio\n", bytes_read);
}

// the system waits for visitor
else if (isSending)
{
  isSending = false;
  lcd.clear();   // Clear the screen
  displayTime(); // display time on screen
  lcd.setCursor(0, 0);
  lcd.print(" "); // Print text on the screen
  lcd.setCursor(0, 1);
  lcd.print("Waiting...    "); // Print text on the screen
  digitalWrite(GREEN_LED_PIN, LOW);
}

// check the firebase variables if data changed from mobile app
if (dataChanged)
{
  dataChanged = false;
  stream_data.replace("\\\"", "");
```

```
  if (event_path == "/Door")
  {
    if (stream_data == "1")
    {
      door(); // open the door
    }
  }
  else if (event_path == "/Notification")
  {
    if (stream_data == "0")
    {
      reset_notification();
    }
    else if (stream_data == "1")
    {
      set_notification();
    }
  }

  else if (event_path == "/")
  {
    ;
  }
  stream_data = "";
 }

 // we can open the for if press the DOOR_OPEN_BUTTON_PIN
 if (!digitalRead(DOOR_OPEN_BUTTON_PIN))
 {
  Serial.println("Door button pressed! Sending signal...");
  door(); // open the door
 }
}
```

```
void door()
{
  Open_The_Door(); // send to transmitter to open the door
  lcd.clear();     // Clear the screen
  lcd.setCursor(3, 0);
  lcd.print("Door Open!"); // Print text on the screen
  lcd.setCursor(1, 1);
  lcd.print("Access Granted"); // Print text on the screen
  digitalWrite(RED_LED_PIN, HIGH);
  delay(5000);
  lcd.clear(); // Clear the screen
  close_door();
  reset_notification();
}

void Open_The_Door()
{
  uint8_t signal = 1;
  udp_lock.beginPacket(transmitterIP, Lock_Port); // send to transmitter udp port
  udp_lock.write(&signal, sizeof(signal));        // write the data
  udp_lock.endPacket();                           // end of communication
  Serial.println("Signal sent to transmitter (LED)");
  open_door(); // set firebase variable
}

void displayTime()
{
  struct tm timeinfo;
  if (!getLocalTime(&timeinfo))
  {
    Serial.println("Failed to obtain time");
    return;
```

```
  }
  char timeString[16];
  strftime(timeString, sizeof(timeString), "%H:%M %d/%m/%y", &timeinfo);
  lcd.setCursor(1, 0);
  lcd.print(timeString); // Print text on the screen
  lcd.setCursor(15, 0);
  lcd.print(" "); // Print text on the screen
}

void control_indication_led()
{
  if (notification)
  {
    unsigned long currentMillis = millis();

    if (currentMillis - previousMillis >= interval)
    {
      previousMillis = currentMillis;

      if (ledState == LOW)
      {
        ledState = HIGH;
      }
      else
      {
        ledState = LOW;
      }

      digitalWrite(WIFI_LED_PENDING_CONNECT, ledState); // RED LED BLINK
      digitalWrite(WIFI_LED_CONNECTED, HIGH);          // GREEN LED OFF Reverse logic due to common anode rgb led
    }
  }
```

```
  else
  {
    digitalWrite(WIFI_LED_PENDING_CONNECT, HIGH); // RED LED OFF Reverse logic due to common anode rgb led
     digitalWrite(WIFI_LED_CONNECTED, LOW);          // GREEN LED ON Reverse logic due to common anode rgb led
  }
}

void reconnect_wifi()
{
  WiFi.disconnect();
  WiFi.begin(ssid, password);
   digitalWrite(WIFI_LED_PENDING_CONNECT, HIGH); // RED LED OFF Reverse logic due to common anode rgb led
   digitalWrite(WIFI_LED_CONNECTED, HIGH);         // GREEN LED OFF Reverse logic due to common anode rgb led

  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
     digitalWrite(WIFI_LED_PENDING_CONNECT, LOW); // RED LED ON Reverse logic due to common anode rgb led
  }
}
```

GPIO.h

```
// Pins I2S for microphone
#define I2S_MIC_CLK              26
#define I2S_MIC_WS               25
#define I2S_MIC_DATA             23

// Pins I2S for speaker
```

```
#define I2S_SPK_CLK               19
#define I2S_SPK_WS                18
#define I2S_SPK_DATA               5

// Leds
#define RED_LED_PIN               2  // led for door opening
#define GREEN_LED_PIN             12 // led when speaks

// button for speaking
#define SPEAK_BUTTON_PIN          4

// Button for door opening
#define DOOR_OPEN_BUTTON_PIN      13

// Led for Wi-Fi status connection
#define WIFI_LED_CONNECTED        33 // wi-fi connected
#define WIFI_LED_PENDING_CONNECT  27 // wifi not connected
```

INIT_GPIO.h

```
#include "GPIO.h"
#include "Arduino.h"

void Init_Pins()
{
   // configuration pins as input or output and set the first status
   pinMode(RED_LED_PIN, OUTPUT);
   pinMode(SPEAK_LED_PIN, OUTPUT);
   pinMode(SPEAK_BUTTON_PIN, INPUT);
   pinMode(DOOR_OPEN_BUTTON_PIN, INPUT);
   pinMode(WIFI_LED_CONNECTED, OUTPUT);
   pinMode(WIFI_LED_PENDING_CONNECT, OUTPUT);
   digitalWrite(RED_LED_PIN, LOW);
```

```
  digitalWrite(GREEN_LED_PIN, LOW);
   digitalWrite(WIFI_LED_CONNECTED, HIGH);       // GREEN LED OFF Reverse logic due to common anode rgb led
   digitalWrite(WIFI_LED_PENDING_CONNECT, LOW); // RED LED ON Reverse logic due to common anode rgb led
}
```

SCREEN.h

```
#include <LiquidCrystal_I2C.h>

LiquidCrystal_I2C lcd(0x27, 16, 2);

String Hellenic_University_message = "Hellenic University"';

int scrollPosition = 0;

#define CLEAR_DELAY 200

void texts()
{
  // 1st screen
  lcd.setCursor(1, 0);        // Set cursors on the screen
  lcd.print("'International"); // Print text on the screen

  // scroll Hellenic University from right to left
  int Message_Length = Hellenic_University_message.length();

  for (scrollPosition = 16; scrollPosition > 0; scrollPosition--)
  {
    lcd.setCursor(0, 1);
    lcd.print("               "); // clean line
    lcd.setCursor(scrollPosition - 1, 1);
```

```
      lcd.print(Hellenic_University_message.substring(0, 16 - (scrollPosition - 1))); // Print text on the screen
      delay(300);
    }

    // Αφού "μπει" όλο το μήνυμα, συνεχίζουμε κανονικό scrolling
    for (int i = 0; i <= Message_Length - 16; i++)
    {
      lcd.setCursor(0, 1);
      lcd.print(Hellenic_University_message.substring(i, i + 16));
      delay(300);
    }

    delay(1000);

    // 2nd screen
    for (int i = 0; i < 3; i++)
    {
      lcd.clear(); // Clear the screen
      delay(CLEAR_DELAY);
      lcd.setCursor(5, 0);        // Set cursors on the screen
      lcd.print("Thesis");        // Print text on the screen
      lcd.setCursor(1, 1);        // Set cursors on the screen
      lcd.print("Master Degree!"); // Print text on the screen
      delay(1000);
    }

    delay(2000);
    lcd.clear(); // Clear the screen
    delay(CLEAR_DELAY);

    // 3rd screen
    lcd.setCursor(2, 0);          // Set cursors on the screen
```

```
    lcd.print("Mala Orestis");     // Print text on the screen
    lcd.setCursor(0, 1);           // Set cursors on the screen
    lcd.print("Ananiadis Themis"); // Print text on the screen
    delay(3000);
    lcd.clear(); // Clear the screen
    delay(CLEAR_DELAY);

    // 4th screen
    lcd.setCursor(2, 0);       // Set cursors on the screen
    lcd.print("Sindos, 2025"); // Print text on the screen
    lcd.setCursor(2, 1);       // Set cursors on the screen
    lcd.print("Fw Ver. 1.0."); // Print text on the screen
    delay(3000);
    lcd.clear(); // Clear the screen
    delay(CLEAR_DELAY);
    lcd.setCursor(0, 0);         // Set cursors on the screen
    lcd.print("System Init..."); // Print text on the screen
    delay(3000);
    lcd.clear(); // Clear the screen
}
```

SOUND.h

```
#include <driver/i2s.h>
#include "GPIO.h"
#include <math.h>                              // Για τη συνάρτηση sin()

#define I2S_SAMPLE_RATE                        16000
#define I2S_SAMPLE_BITS I2S_BITS_PER_SAMPLE_16BIT
#define I2S_CHANNELS                           1
#define BUFFER_SIZE                            256 // Μικρότερο buffer

#define I2S_MIC_PORT I2S_NUM_0
```

```
#define I2S_SPEAKER_PORT I2S_NUM_1                    // Χρησιμοποιούμε το δεύτερο I2S για το ηχείο

#define BUFFER_LEN                          64  // default was 64

bool isMelodyPlaying = false; // Κατάσταση αναπαραγωγής της μελωδίας

void i2s_mic_install()
{
  const i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
      .sample_rate = I2S_SAMPLE_RATE,
      .bits_per_sample = I2S_SAMPLE_BITS,
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format = I2S_COMM_FORMAT_I2S,
      .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
      .dma_buf_count = 4,       // Μειωμένο dma buffer count
      .dma_buf_len = BUFFER_LEN, // Μειωμένο dma buffer length
      .use_apll = false};
  i2s_driver_install(I2S_MIC_PORT, &i2s_config, 0, NULL);
}

void i2s_mic_setpin()
{
  const i2s_pin_config_t pin_config = {
      .bck_io_num = I2S_MIC_CLK,
      .ws_io_num = I2S_MIC_WS,
      .data_out_num = I2S_PIN_NO_CHANGE,
      .data_in_num = I2S_MIC_DATA};
  i2s_set_pin(I2S_MIC_PORT, &pin_config);
}

void i2s_speaker_install()
```

```
{
  const i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_TX),
      .sample_rate = I2S_SAMPLE_RATE,
      .bits_per_sample = I2S_SAMPLE_BITS,
      .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT,
      .communication_format = I2S_COMM_FORMAT_I2S,
      .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
      .dma_buf_count = 4,       // Μειωμένο dma buffer count
      .dma_buf_len = BUFFER_LEN, // Μειωμένο dma buffer length
      .use_apll = false};
  i2s_driver_install(I2S_SPEAKER_PORT, &i2s_config, 0, NULL);
}

void i2s_speaker_setpin()
{
  const i2s_pin_config_t pin_config = {
      .bck_io_num = I2S_SPK_CLK,
      .ws_io_num = I2S_SPK_WS,
      .data_out_num = I2S_SPK_DATA,
      .data_in_num = I2S_PIN_NO_CHANGE};
  i2s_set_pin(I2S_SPEAKER_PORT, &pin_config);
}

void playSineWave()
{
  const uint32_t duration_ms = 2000;  // 2 δευτερόλεπτα (2000 ms)
  const uint32_t sample_rate = 16000; // 16 kHz δειγματοληψία
  const float frequency = 440.0f;     // 440 Hz (η συχνότητα του "λά")
  const float amplitude = 32767.0f;   // Μέγιστο πλάτος για 16-bit audio

  // Υπολογίζουμε τον αριθμό των δειγμάτων που αντιστοιχούν σε 2 δευτερόλεπτα
  const size_t num_samples = (duration_ms * sample_rate) / 1000;
```

```
  // Δημιουργούμε ένα buffer με 1 δείγμα (αλλά θα το επαναλάβουμε)
  int16_t sample;
  size_t bytes_written;

  // Παράγουμε και στέλνουμε τα δείγματα ένα-ένα
  for (size_t i = 0; i < num_samples; i++)
  {
    // Υπολογίζουμε την τιμή του sine wave για το τρέχον δείγμα
    sample = (int16_t)(amplitude * sin(2 * M_PI * frequency * i / sample_rate));

    // Στέλνουμε το δείγμα μέσω I2S
    i2s_write(I2S_SPEAKER_PORT, &sample, sizeof(sample), &bytes_written, portMAX_DELAY);
  }

  isMelodyPlaying = false; // Ενημερώνουμε ότι ο ήχος τελείωσε
}

void silence_speaker()
{
  // set the BUFFER_SIZE with {0} to silence the speaker
  int16_t silence[BUFFER_SIZE] = {0};
  size_t bytes_written;
  i2s_write(I2S_SPEAKER_PORT, silence, sizeof(silence), &bytes_written, portMAX_DELAY);

  Serial.println("Sine wave stopped.");
}
```

VARIABLES.h

```
// Wi-Fi credentials
const char *ssid = " HUAWEI Mate 20 Pro ";
```

```
const char *password = "12345678";

// const char *ssid = "";
// const char *password = "";

// Udp ports for communication with transmitter
const int Receive_Audio_Port = 9999;
const int Transmit_Audio_Port = 1234;
const int Lock_Port = 5678;
const int Call_Port = 4321;

bool isSending = false;

// NTP configuration
const char *ntpServer = "pool.ntp.org";
const long gmtOffset_sec = 7200;     // UTC+2 για Ελλάδα (Χειμερινή ώρα)
const int daylightOffset_sec = 3600; // Προσθήκη 1 ώρας όταν έχουμε θερινή ώρα (seconds)

unsigned long previousMillis = 0;  // will store last time LED was updated

// constants won't change:
const long interval = 200;  // interval at which to blink (milliseconds)

int ledState = LOW;  // ledState used to set the LED
```

FIREBASE.h

```
#include <Firebase_ESP_Client.h>
#include <addons/TokenHelper.h>
#include <addons/RTDBHelper.h>

/* 2. Define the API Key */
#define API_KEY "AIzaSyBYhzZYozt_o4pFjc1BYm72N6vqLUSWonU"
```

```
//------------------------------------------------------------------------
/* 3. Define the RTDB URL */
#define DATABASE_URL "intercom-system-ea9a7-default-rtdb.firebaseio.com"
//------------------------------------------------------------------------
String door_intercom_system = "Door_System";

String stream_path = "";
String event_path = "";
String stream_data = "";
String bucketData = "";
String bucketPath = "";

FirebaseJson jsonData;

volatile bool dataChanged = false;
bool signupOK = false;
bool uploadBucket = false;
bool notification = false;

FirebaseData stream;
FirebaseData fbdo;
FirebaseAuth auth;
FirebaseConfig config;
FirebaseData firebaseData;

IPAddress transmitterIP; // Global or local variable as needed

void FirebaseWrite(String URL, int data)
{
   if (!Firebase.ready() && !signupOK)
   {
      Serial.println("Write Failed: Firebase not ready OR signup not OK");
      return;
```

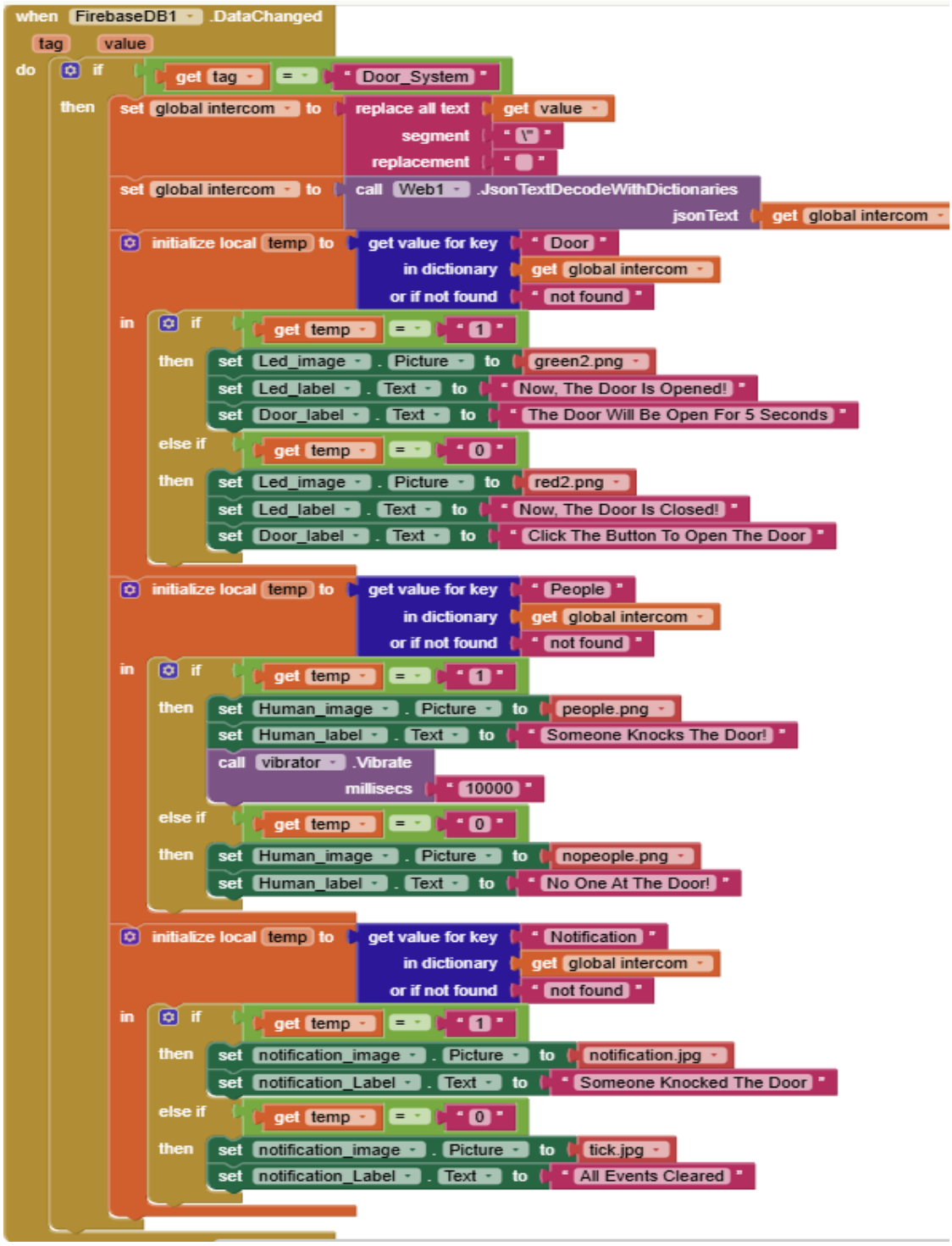
```
    }

    URL = door_intercom_system + "/" + URL;
    Serial.println(URL);
    Serial.println(String(data));

    if (Firebase.RTDB.setString(&fbdo, URL, String(data)))
    {
       Serial.println("PASSED");
       Serial.println("PATH: " + fbdo.dataPath());
       Serial.println("TYPE: " + fbdo.dataType());
    }
    else
    {
       Serial.println("FAILED");
       Serial.println("REASON: " + fbdo.errorReason());
    }
}
void people()
{
    uploadBucket = true;
    bucketData = "1";
    bucketPath = "People";
    delay(100); // μικρή καθυστέρηση debounce
    //  αποστολή τιμής "1" στο Firebase
    FirebaseWrite("People", 1);
}
void no_people()
{
    uploadBucket = true;
    bucketData = "0";
    bucketPath = "People";
    delay(100); // μικρή καθυστέρηση debounce
```

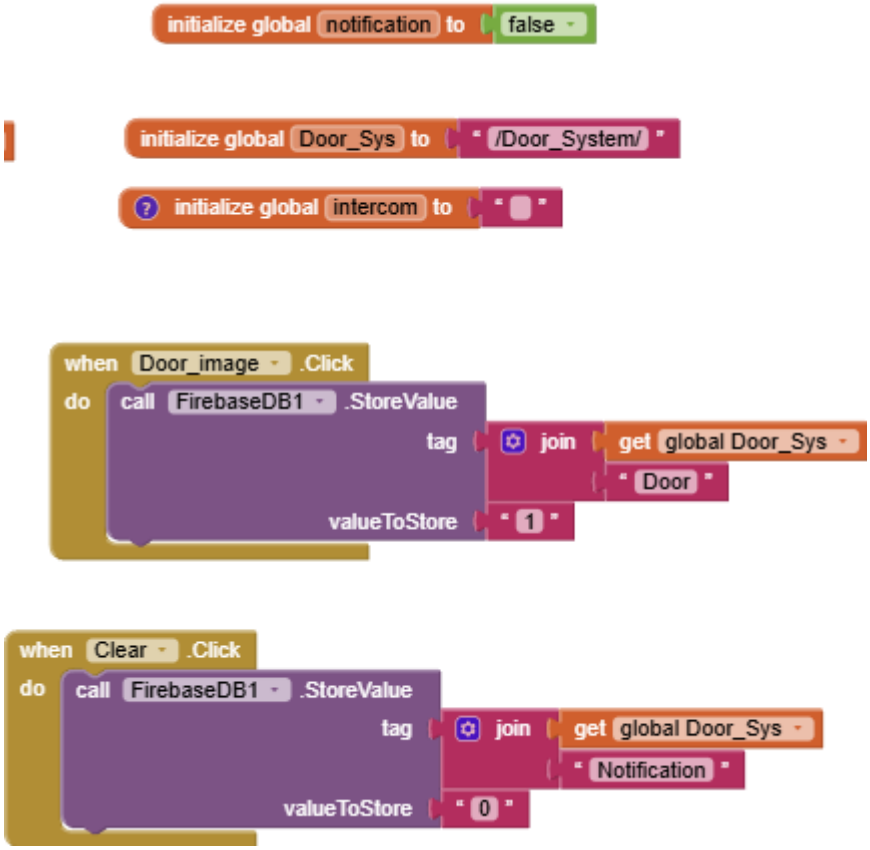
```
    //  αποστολή τιμής "0" στο Firebase
    FirebaseWrite("People", 0);
}

void set_notification()
{
    uploadBucket = true;
    bucketData = "1";
    bucketPath = "Notification";

    delay(100); // μικρή καθυστέρηση debounce
    // αποστολή τιμής "1" στο Firebase
    FirebaseWrite("Notification", 1);
    notification = true;
}
void reset_notification()
{
    uploadBucket = true;
    bucketData = "0";
    bucketPath = "Notification";
    delay(100); // μικρή καθυστέρηση debounce
    // αποστολή τιμής "1" στο Firebase
    FirebaseWrite("Notification", 0);
    notification = false;
}
void open_door()
{
    uploadBucket = true;
    bucketData = "1";
    bucketPath = "Door";
    delay(100); // μικρή καθυστέρηση debounce
    // αποστολή τιμής "1" στο Firebase
    FirebaseWrite("Door", 1);
```

```
}
void close_door()
{
   uploadBucket = true;
   bucketData = "0";
   bucketPath = "Door";
   delay(100); // μικρή καθυστέρηση debounce
   // αποστολή τιμής "0" στο Firebase
   FirebaseWrite("Door", 0);
}
void Set_IP(String ipStr)
{
   if (Firebase.RTDB.setString(&firebaseData, "/devices/esp_receiver/ip", ipStr))
   {
      Serial.println("IP address uploaded to Firebase successfully.");
   }
   else
   {
      Serial.print("Failed to upload IP: ");
      Serial.println(firebaseData.errorReason());
   }
}

void Get_IP()
{
   // Διάβασε την IP του sender
   if (Firebase.RTDB.getString(&firebaseData, "/devices/esp_sender/ip"))
   {
      String ip = firebaseData.stringData();
      Serial.print("Read IP from Firebase: ");
      Serial.println(ip);
      int ip1, ip2, ip3, ip4;
      if (sscanf(ip.c_str(), "%d.%d.%d.%d", &ip1, &ip2, &ip3, &ip4) == 4)
```

```
        {
            transmitterIP = IPAddress(ip1, ip2, ip3, ip4);
            Serial.print("Parsed IPAddress: ");
            Serial.println(transmitterIP);
        }
        else
        {
            Serial.println("Invalid IP format received from Firebase.");
            // transmitterIP = IPAddress(192, 168, 1, 100); // fallback IP
        }
    }
    else
    {
        Serial.print("Failed to read IP: ");
        Serial.println(firebaseData.errorReason());
        // transmitterIP = IPAddress(192, 168, 1, 100); // fallback IP
    }
}
```

ΠΑΡΆΡΤΗΜΑ Γ: ΚΩΔΙΚΑΣ ANDROID ΕΦΑΡΜΟΓΗΣ




initialize global notification to false
initialize global Door_Sys to /Door_System/
initialize global intercom to
when Door_image .Click
do call FirebaseDB1 .StoreValue
tag join get global Door_Sys
Door
valueToStore 1
when Clear .Click
do call FirebaseDB1 .StoreValue
tag join get global Door_Sys
Notification
valueToStore 0