



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Design of a Nurse Calling System with Real Time Indoor Locations Capabilities

Document:

Report

Author:

Enric Botella Pastor

Director /Co-director:

Javier Freire Venegas

Degree:

Bachelor in Industrial Technology Engineering

Examination session:

Extension, 2023

BACHELOR FINAL THESIS

Abstract

The main driver of this project has been to design a prove of concept of a device that allows live voice communication between patients and medical staff and the capability to locate in real time patients in an enclosed environment.

The author of this project had an initial constraint it was supplied by the project director supplied for this project a DWM1001-DEV Module Development Board which provides accurate positioning thanks to its wireless Real Time Location System. After a study of the problem, a selection of components, both hardware and software were selected.

The sound system is composed of a I2S Microphone SPH0645 for real time audio capturing, an I2S Amplifier Breakout board MAX98357A and a generic 8ohms speaker.

The patient interface component is a button used to trigger communication.

For the development of the software the Espressif IoT Development Framework was used, it provides APIs for the user to program the ESP32. The ESP-IDF was installed on VSCode IDE. For debugging the system, we used a J-Link PRO with the Eclipse IDE.

The ESP32 communicates with a python-based server using a Wi-Fi network, the communication is based on the UDP protocol.

The result is a prototype that showcase that a final product based on this system is feasible, it presents great autonomy and excellent real time communication features.

Further lines of work are described, and the presented system is flexible enough to integrate them.

El motor principal d'aquest projecte ha sigut dissenyar un dispositiu que permet la comunicació per veu en directe entre pacient i personal sanitari amb la capacitat de localitzar en temps real els pacients en un entorn tancat.

Per l'elaboració del dispositiu, el director del projecte ens proporciona un mòdul DWM1001-DEV sense fils que estableix un posicionament precís gràcies a l'ús de la tecnologia Ultra Wide Band.

Un cop estudiat el problema, es van seleccionar una sèrie de components tant de hardware com de software. El sistema de so està format per un micròfon I2S SPH0645 per la captura d'àudio en temps real, una targeta I2S Amplifier Breakout MAX98357A i un altaveu genèric de 8 ohms. La interfície pacient- dispositiu es dona a través d'un polsador, una vegada activat, el pacient pot comunicar-se amb el personal de l'hospital.

Pel desenvolupament del software es va fer servir Espressif IoT Development Framework, que proporciona APIs per tal que l'usuari programi l'ESP32. L'ESP-IDF s'ha programat amb l'IDE VSCode; per debuggejar el sistema es va utilitzar un J-Link PRO con el IDE Eclipse.

L'ESP32 es comunica amb un servidor basat en Python fent ús d'una xarxa Wi-Fi, on la comunicació es basa en el protocol UDP.

Com ha resultat tenim un prototip que demostra que un producte final basat en aquest sistema és factible, presenta gran autonomia i excel·lents prestacions de comunicació en temps real.

Es descriuen altres línies de treball, i el sistema presentat és prou flexible per a integrar-les.



Table of contents

ABSTRACT	I
TABLE OF CONTENTS	III
LIST OF TABLES	V
LIST OF FIGURES.....	VI
LIST OF ABBREVIATIONS / GLOSSARY	IX
1. INTRODUCTION.....	1
1.1 OBJECT	1
1.2 SCOPE.....	2
1.3 REQUIREMENTS	3
1.4 RATIONALE	4
2 BACKGROUND AND/OR REVIEW OF THE STATE OF THE ART	5
2.1 REGULATIONS.....	5
2.2 STANDARDS	5
2.3 EXISTING TECHNOLOGY.....	6
2.3.1 <i>Legacy solutions</i>	6
2.3.2 <i>New state of the art solutions</i>	7
3 METHODOLOGY.....	10
3.1 SYSTEM ARCHITECTURE	10
3.2 SUBSYSTEM ARCHITECTURE.....	11
3.2.1 <i>RTLS Subsystem</i>	11
3.2.1.1 Choosing the right Interface	12
3.2.2 <i>Communication system</i>	13
3.2.2.1 Choosing a Physical/Data-Link layer Network Protocol	13
3.2.2.2 Integrated or separate Wi-Fi Module	13
3.2.3 <i>Sound System</i>	14
3.2.3.1 Analog Audio Processing	14
3.2.4 <i>Patient interface</i>	16
3.2.5 <i>Power Supply</i>	16
3.2.6 <i>Master Mcu</i>	17
4 ELECTRONIC COMPONENTS	18
4.1 MASTER MCU	18
4.2 POWER SUPPLY.....	20
4.3 AUDIO SYSTEM.....	21
4.3.1 <i>Recording system</i>	21
4.3.2 <i>Playback System</i>	22
4.4 DWM1001-DEV	24
4.5 PATIENT INTERFACE	24
5 TOOLS.....	25
5.1 SOLDERING AND ELECTRONICS TESTING.....	25
5.2 PROGRAMMING THE ESP32	26
5.2.1 <i>Hardware</i>	26
5.2.2 <i>Software</i>	26
5.2.3 <i>Installation</i>	27
5.2.4 <i>Start a project</i>	27
5.3 JTAG DEBUGGING THE ESP32	28
5.3.1 <i>Hardware</i>	28

5.3.2	<i>Software</i>	28
5.3.3	<i>Set-up</i>	29
5.3.4	<i>Start debugging</i>	30
5.4	WIRESHARK	31
6	CONSIDERATION AND DECISION REGARDING ALTERNATIVE SOLUTIONS	32
7	DISCUSSION OF THE SOLUTION	34
7.1	CONFIGURING THE RTLS	34
7.1.1	<i>Network</i>	34
7.1.2	<i>Scalability</i>	35
7.2	PROGRAMMING THE ESP32	36
7.2.1	<i>RTLS data fetch</i>	38
7.2.1.1	Component Architecture	39
7.2.2	<i>Interface component</i>	42
7.2.2.1	Component Architecture	43
7.2.3	<i>Wi-Fi Component</i>	44
7.2.3.1	Component Architecture	45
7.2.4	<i>Audio pipeline</i>	49
7.2.4.1	Recording pipeline	49
7.2.4.2	Playback pipeline	52
7.2.5	<i>UDP component</i>	54
7.2.5.1	Component architecture	55
7.3	SERVER	60
7.3.1	<i>Server Architecture</i>	61
8	BUDGET SUMMARY	65
9	ANALYSIS AND ASSESSMENT OF ENVIRONMENTAL AND SOCIAL IMPLICATIONS	67
10	CONCLUSIONS	68
11	REFERENCES	70
	ANNEX C BUDGET	80

List of tables

Table 1. Technical Characteristics of MEM Mics (Source: Enric, Botella-Pastor)

Table 2. Technical Characteristic of different RTLS solutions (Source: Gladysz, B., and K. Santarek. "An Approach to Rtls Selection." DEStech Transactions on Engineering and Technology Research, no. icpr, Mar. 2018, <https://doi.org/10.12783/dtetr/icpr2017/17576>.)

Table 3: dwm_pos_get request (Source: Decawave Ltd. DWM1001 Firmware API Guide. 2019, p. 43, <https://www.qorvo.com/products/d/da007975> . Accessed 12 Nov. 2022.)

Table 4: dwm_pos_get response (Source: Decawave Ltd. DWM1001 Firmware API Guide. 2019, p. 43, <https://www.qorvo.com/products/d/da007975> . Accessed 12 Nov. 2022.)

Table 5: Budget summary (Source: Enric, Botella-Pastor)

Table 6: Budget (Source: Enric, Botella-Pastor)

Table 7: Component providers (Source: Enric, Botella-Pastor)

Table 8: Data Sheets (Source: Enric, Botella-Pastor)

List of figures

The titles and numbers of all figures included, in the order in which they appear in the text.

[OBJ]

Figure 1. Stainless Steel Call Units by Carecom (Source: Carecom. Stainless Steel Call Units. 6 Sep. 2022, https://carecom-solutions.com/en_ccs-nurse-call-products/en_ccs-nurse-call-call-units-detail/en_ccs-nurse-call-call-units-stainless-steel-call-units.html)

Figure 2. Nurse Control Station (Source: Carecom. Nurse Control Station - The Staff Base Console. 6 Sep. 2022, https://carecom-solutions.com/en_ccs-nurse-call-products/en_about-our-products-in-detail/en_ccs-nurse-call-display-units-nurse-control-station.html.)

Figure 3. IP-DECT (Source: COBS. IP-DECT 400. 7 Sept. 2022, https://cobs.se/images/cobs_images/blockschema/block-diagram_04.png.)

Figure 4. Alarm with speech and positioning (Source: COBS. Alarm with Speech and Positioning. 7 Sept. 2022, https://cobs.se/images/cobs_images/blockschema/block-diagram_03.png.)

Figure 5. Stanley Aeroescout Tag (Source: Stanley Health Care. Stanley Aeroescout Tag. 7 Nov. 2022, https://www.stanleyhealthcare.com/sites/stanleyhealthcare.com/files/styles/hero_slider/public/media/2018-11/aeroscout-t2s-tag.png?itok=EmguLojs.)

Figure 6. System Architecture Components (Source: Enric, Botella-Pastor)

Figure 7. DWM1001-DEV (Source: QORVO. DWM1001-DEV Ultra-Wideband (UWB) Transceiver Development Board. 14 Sept. 2022, <https://www.qorvo.com/products/i/da007939>.)

Figure 8. Interface of RTLS and MCU components (Source: Enric, Botella-Pastor)

Figure 9. Analog vs Digital Audio Processing Architecture (Source: Enric, Botella-Pastor)

Figure 10. Analog to digital conversion (Source: Alzartech. Variable Frequency Adc Clock. 26 Sept. 2022, <https://www.alzartech.com/LowCal/Resources/static/images/alzartech/variablefrequencyadc/en/7374-alazarTech-variable%20frequency-ADC-clock-01.jpg>.)

Figure 11. Audio System Architecture (Source: Enric, Botella-Pastor)

Figure 12. Interface System Architecture (Source: Enric, Botella-Pastor)

Figure 13. System Device Architecture Sketch (Source: Enric, Botella-Pastor)

Figure 14. ESP-32 Dev Kit C V4 (Source: AzDelivery. ESP-32 Dev Kit C V4. 2 Oct. 2022, https://cdn.shopify.com/s/files/1/1509/1638/products/v4main_130x.jpg?v=1605879848.)

Figure 15. ESP-32 Dev Kit C V4 Pinout (Source: AzDelivery. ESP-32 Dev Kit C V4 Pinout. 2 Oct. 2022, https://cdn.shopify.com/s/files/1/1509/1638/files/ESP32_DevKit_C_V4_Pinout.pdf?v=1615364529.)

Figure 16. I2S Standard Philips Format (Source: Espressif. Inter-IC Sound (I2S). Standard Mode. 4 Oct. 2022, <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2s.html#standard-mode> .)

Figure 17. I2S Data Format SPH0645 (Source: Knowles. SPH0645LM4H-B Rev B Datasheet. 4 Oct. 2022, <https://cdn-shop.adafruit.com/product-files/3421/i2S+Datasheet.PDF>.)

Figure 18. I2S DAC MAX98357A/B (Source: Adafruit. Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A. 5 Oct. 2022, <https://cdn-shop.adafruit.com/970x728/3006-00.jpg>.)

Figure 19. Weewoday Speakers (Source: Weewoday. 6 Piezas 2W 8 Ohmios Altavoces Pequeños. 5 Oct. 2022, <https://www.amazon.es/dp/B09MRK24PP>.)

Figure 20. DWM1001-DEV Technical Data (Source: Qorvo. Product Brief: Dwm1001-Dev. 7 Oct. 2022, <https://www.qorvo.com/products/d/da007951>.)

Figure 21. Push button installed on the system (Source: Enric, Botella-Pastor)

Figure 22. Work In Process. Soldering (Source: Enric, Botella-Pastor)

Figure 23. Code Structure (Source: Enric, Botella-Pastor)

Figure 24. ESP32 Jlink Connections (Source: Visual Micro. Debug Help ESP32 Jlink Connections. <https://www.visualmicro.com/pics/Debug-Help-ESP32-Jlink-Connections.png> . Accessed 14 Nov. 2022.)

Figure 25. Eclipse IDE Debug Configuration (Source: Enric, Botella-Pastor)

Figure 26. Eclipse IDE Control panel (Source: Enric, Botella-Pastor)

Figure 27. Eclipse IDE Debug session (Source: Enric, Botella-Pastor)

Figure 28. Audio packets from ESP32 (Source: Enric, Botella-Pastor)

Figure 29. Tags choice of anchors (Source: Qorvo. DWM1001 System Overview and Performance. Figure 9. Tags Choice of Anchors for TWR. 2018, <https://www.qorvo.com/products/d/da007974>. Accessed 13 Nov. 2022.)

Figure 30: Figure 30: Scaling the DRTLS showing anchor's seat numbers (Source: Qorvo. DWM1001 System Overview and Performance. Figure 10. Scaling the DRTLS showing anchor's seat numbers. 2018, <https://www.qorvo.com/products/d/da007974>. Accessed 13 Nov. 2022.)

Figure 31: System Finite State Machine (Source: Enric, Botella-Pastor)

Figure 32: Entry point. Appmain. (Source: Enric, Botella-Pastor)

Figure 33: Transition to FSM_COMM state Figure 33: Transition to FSM_COMM state from received control packet (Source: Enric, Botella-Pastor)

Figure 33: Transition to FSM_COMM state from GPIO interrupt (Source: Enric, Botella-Pastor)

Figure 34: DWM1001 UART workflow (Source: Qorvo. DWM1001 S DWM1001 FIRMWARE APPLICATION PROGRAMMING INTERFACE (API) GUIDE. Figure 6. DWM1001 UART workflow. 2019, <https://www.qorvo.com/products/d/da007975> . Accessed 20 Nov. 2022.)

Figure 35: SWC_UART Workflow (Source: Enric, Botella-Pastor)

Figure 36: FreeRtos Uart Task (Source: Enric, Botella-Pastor)

Figure 37,38: uart_init() function (Source: Enric, Botella-Pastor)

Figure 39,40: read_write() function (Source: Enric, Botella-Pastor)

Figure 41: SWC_BUTTON Workflow (Source: Enric, Botella-Pastor)

Figure 42: Internal Pull-up resistor configuration (Source: Enric, Botella-Pastor)

Figure 43: init_gpio() function (Source: Enric, Botella-Pastor)

Figure 44: gpio_isr_handler() function (Source: Enric, Botella-Pastor)

Figure 45: SWC_WIFI Workflow (Source: Enric, Botella-Pastor)

Figure 46: wifi_comp_init() function (Source: Enric, Botella-Pastor)

Figure 47: wifi_init_sta() 1 (Source: Enric, Botella-Pastor)

Figure 48: wifi_init_sta() 2 (Source: Enric, Botella-Pastor)

Figure 49: event_handler () function (Source: Enric, Botella-Pastor)

Figure 50: Recording pipeline (Source: Enric, Botella-Pastor)

Figure 51: i2s_rx_init() function (Source: Enric, Botella-Pastor)

Figure 52: MSB adjustment for MIC (Source: Enric, Botella-Pastor)

Figure 53: DMA read call (Source: Enric, Botella-Pastor)

Figure 54,55: sample_32bit_to_16bit() (Source: Enric, Botella-Pastor)

Figure 56,57: i2s_tx_init() (Source: Enric, Botella-Pastor)

Figure 58: recvfrom() function (Source: Enric, Botella-Pastor)

Figure 59: Write audio to DMA (Source: Enric, Botella-Pastor)

Figure 60: SWC_UDP workflow (Source: Enric, Botella-Pastor)

Figure 61: `udp_client_task()` function (Source: Enric, Botella-Pastor)

Figure 62: UDP Socket Creation (Source: Enric, Botella-Pastor)

Figure 63: Communication Workflow (Source: Enric, Botella-Pastor)

Figure 64: Implementation of packet payload building (Source: Enric, Botella-Pastor)

Figure 65: Packet Sending and Error Check (Source: Enric, Botella-Pastor)

Figure 66: Packet Reception and Error Check (Source: Enric, Botella-Pastor)

Figure 67: Treatment of Received Payloads (Source: Enric, Botella-Pastor)

Figure 68: Server Workflow (Source: Enric, Botella-Pastor)

Figure 69: Initialization of Audio Streams (Source: Enric, Botella-Pastor)

Figure 70: Creation of UDP Server Socket (Source: Enric, Botella-Pastor)

Figure 71: Treatment of Received Packets and Creation of Response Payload (Source: Enric, Botella-Pastor)

Figure 72: Implementation of mock control signals (Source: Enric, Botella-Pastor)

Figure 73: Packet Sending (Source: Enric, Botella-Pastor)

Figure 74: SDG Symbol (Source: U.N. SDG Symbol. 2022,

<https://www.un.org/sustainabledevelopment/wp-content/uploads/2018/03/global-goals.png>.

Accessed 5 Jan. 2023.)

Figure 75: Goal 3 (Source: U.N. Goal 3: Ensure Healthy Lives and Promote Well-Being for All at All Ages. Banner. <https://www.un.org/sustainabledevelopment/health/>. Accessed 6 Jan. 2023.)

Figure 76: Goal 8 (Source: U.N. Goal 8: Promote inclusive and sustainable economic growth, employment and decent work for all. Banner. <https://www.un.org/sustainabledevelopment/economic-growth/>. Accessed 6 Jan. 2023.)

Figure 77: Goal 9 (Source: U.N. Goal 9: Build resilient infrastructure, promote sustainable industrialization and foster innovation <https://www.un.org/sustainabledevelopment/infrastructure-industrialization/>. Accessed 6 Jan. 2023.)

Figure 78: Screenshot Step 3 Flashing Firmware (Source: Enric, Botella-Pastor)

Figure 79: Screenshot Step 5 Flashing Firmware (Source: Enric, Botella-Pastor)

Figure 80: Step 2 Configuring DWM1001-DEV (Source: Enric, Botella-Pastor)

Figure 81: Step 4 Configuring DWM1001-DEV (Source: Enric, Botella-Pastor)

Figure 82: Step 5 Configuring DWM1001-DEV (Source: Enric, Botella-Pastor)

Figure 83: Step 6 Configuring DWM1001-DEV 1 (Source: Enric, Botella-Pastor)

Figure 84: Step 6 Configuring DWM1001-DEV 2 (Source: Enric, Botella-Pastor)

Figure 85: Step 7 Configuring DWM1001-DEV 1 (Source: Enric, Botella-Pastor)

Figure 86: Step 7 Configuring DWM1001-DEV 2 (Source: Enric, Botella-Pastor)

Figure 87: Step 8 Configuring DWM1001-DEV 1 (Source: Enric, Botella-Pastor)

Figure 88: Step 8 Configuring DWM1001-DEV 2 (Source: Enric, Botella-Pastor)

Figure 89: Step 9 Configuring DWM1001-DEV 1 (Source: Enric, Botella-Pastor)

Figure 90: Step 9 Configuring DWM1001-DEV 2 (Source: Enric, Botella-Pastor)

Figure 91: Initial Gantt Chrono

Figure 92: Initial Gantt Chart and Diagram

List of abbreviations / Glossary

Abbreviation	Meaning
RTLS	Real-time location system
Wi-Fi	Wireless Fidelity
ICU	Intensive Care Unit
FDA	Foods and Drugs Administration
EMA	European Medicines agency
AEMPS	Agencia Española de Medicamentos y Productos Sanitarios
AC	Alternative Current
MDR	Medical devices Regulation
CE	Conformité Européene
IP	Internet Protocol
Wi-Fi AP or AP	Wi-Fi Access Point
RFID	Radio-frequency identification
CMS	Content Managament System
API	Application Programming Interface
UDP	Unified Datagram Protocol
PCB	Printed Circuit Board
MCU	Micro Controller Unit
WEB UI	WEB User Interface
μC	Micro Controller Unit
BLE	Bluetooth low energy
UWB	Ultra-Wide Band
UART	Universal asynchronous receiver-transmitter
SPI	Serial Peripheral Interface
MOSI	Master Out Slave In

MISO	Master In Slave Out
SCLK	Serial Clock
CS/SS	Chip/Slave Select
OSI	Open Systems Interconnection
ADC	Analog to Digital Converter
DAC	Digital to Analog Converter
I2S	Inter-IC Sound
SD	Serial Data
SCK	Continuous serial clock
WS	Word Select
GPIO	General Purpose Input/Output
PWM	Pulse-width modulation
FCC	Federal Communications Commision
IDE	Integrated development environment
I/O	Input/Output
MEMS	Microelectromechanical systems
ASIC	Application-specific integrated circuit
ECM	Electret condenser microphone
MCLK	Master clock
JTAG	Join test action group
OCD	Open On-Chip Debugger
GPS	Global positioning system
RTOS	Real-time operating system
STA	Station
TCP/IP	Transmission Control Protocol / Internet Protocol
CPU	Central Processing Unit
DMA	Direct Memory Access
IPv4	Internet Protocol Version 4



1. Introduction

1.1 Object

The goal of this project is to design and develop a prototype portable device for real time voice communications and location of patients for use in a medical environment.

Hospitals often get emergency units overcrowded and their communications devices located on hospital beds are fixed, patients and hospital staff lack an efficient way of communicating when they cannot use traditional non portable nurse call systems.

This leads to miscommunication with patients posing a risk for the patient's health and difficulties to locate them.

The device will allow real time voice communication in any place of the hospital and location of patients for a fast and efficient response.

1.2 Scope

This project has three main phases. The first phase research and analysis of the problem, the second phase focus on designing a solution to the problem and the third phase implements that design.

The first phases main work packages are:

- Study of the problem and requirements.
- Study of regulations and standards applied.
- Study of legacy solutions and state of the art solutions to similar problems.
- Study of project related to Audio or RTLS.

The Design of a solution deliverables are:

- Identification of Main Components of the System.
- Identification of Main interfaces between Components.
- Identification of key elements for each Component.
- Proposal of a System Architecture.
- Selection between Hardware/Software alternatives to build or prototype.

The implementation of the design will focus on the delivery of a prototype which will demonstrate if the project is feasible and further development is justified. The prototype at least must deliver real time voice communication and in time positioning. From the knowledge acquired proposals for further development or modifications of the presented prototype will be made.

1.3 Requirements

Requirements for this project have been drafted in close contact with the Tutors and nurse Dr. Adelina, who I thank a lot for her input as a healthcare worker.

The project requirements are the following:

- Location of device user with a 1m radius precision.
- Device should be portable, light weight and rechargeable.
- Device should have an autonomy of ≥ 24 h.
- Device should be able to locate patients in an enclosed space.
- Update position of the patient when he prompts a button and make it available for
- Update position of the patient at least every three minutes.
- The patient should only interact with one button, and it should be easy to access and prompt.
- Once the patient prompts the button, he can talk using a microphone attached to the device.
- Health care workers can contact back the patient via voice.
- The patient should be able to hear back an answer from health care workers.
- Health care workers should be able to know information about the patient who is behind the device. Name, what unit is he/she in (ex. Traumatology), last updated position.
- Creation of interface for health care workers to access information about patients and contact with them if patient press the button.
- The system requires a micro controller with Wi-Fi to link the microphone, speakers and a server.
- The device should use MDEK1001 Development Kit as Realtime location system (RTLS) as we currently own a kit, and the accuracy is in the range of centimetres.

1.4 Rationale

This Project wants to solve a problem in the health care sector. ICU units in most hospitals have a limited number of beds that cannot handle the influx of patients at peak hours or special events. If this scenario is reach patients lay on the alleyways of the hospital in mobile beds or chairs.

Normal communication channels in the hospital are immediately severed. When hospitals run at capacity the communication channels if a patient asks for help consist of a button patient can press.

From a panel hospital staff can spot which room has asked for assistance and then reach out the alleyway where red light above the room signals that the patient inside need assistance.

This method of communication works fine but it is not portable or scalable. Several dysfunctions in a hospital occur because of lacking a communication system that is portable or scalable. Patients are forced to scream if they want to get the attention of the staff.

This method clearly has drawbacks, it is not assured that someone will hear the patient when heard it is not clear where the screams might come from. It is also tiresome for the patient and other patients that hear the screams and might add stress to them which is decremental to health.

Applying embedded systems technology will allow us to solve this issue while allowing us to add new functionalities that current systems lack. RTLS allows us to retrieve locations of a device within an enclosed or open environment with great precision while being portable.

Allowing the location of patients not only inside of the hospital but also in the surrounding of the hospital. The possibility of adding in time communication will allow the patient and healthcare staff to share import information.

The use of an interactive interface with extra information about patients and their location is also a great addition as it allows to get basic information of who is asking for help. This kind of device could not only be used on a hospital but elderly residence or other facilities with people that requires assistance.

One example would be in an elderly residence, people with dementia could be tracked if the patients leave the building.

2 Background and/or review of the state of the art

Nursing call systems, also known as patient call systems or emergency call systems are considered medical devices. They are subject to Regulations, guidelines and standards have been created to assure the correct functioning of the device.

2.1 Regulations

Medical devices due to their critical function must be approved by the regulatory body of each jurisdiction before being marketed. The biggest agencies are the FDA, with oversight over the USA, the EMA, which is used as reference by the member states of the European Union to set regulations. Achieving approval from both agencies, which are seen as global standards, help access third markets.

In Spain the governing body is the AEMPS which as part of the EU, uses the EMA legal framework as reference but approves the use of medical devices for its jurisdiction.

The FDA classifies devices powered by an AC- or battery-powered intended for medical purposes that is used by a patient to operate an environmental control function, such as a telephone or to sound an alarm for assistance as powered environmental control system which are Class II (special controls) medical devices. Class II medical devices are of moderate risk. The specific requirements are under Title 21 of the Code of Federal Regulations, subchapter H. [1][2]

The EMA classifies medical devices according to the Medical Devices Regulation.M

According to Rule 10 Active devices intended for diagnosis and monitoring are classified as class IIa, an active device is a device used to supply information for monitoring conditions, states of health which our nursing call system falls in. [3]

We will focus on the general obligations by the EMA, as is the main regulatory body in Europe and the AEMPS uses its legal framework to make decisions and other national bodies.

The device must:

- Meet the general safety and performance requirements specified in the Annex I of the MDR, which requires a premarket clinical investigation.
- After complying with the MDR must get a CE marking.
- A Unique Device Identifier number and be registered in the electronic system, according with MDR Article 29.
- Be subject to reporting requirements under the medical device vigilance system.
- According to Article 52 of the MDR, a class IIa device must have a quality management system, be subject to an audit and have a surveillance assessment of it. [4]

2.2 Standards

Standards are useful to ensure quality and reliability of medical devices. Compliance with regulatory requirements for the approval of the product as standards have requirements and guidelines to ensure the safety and effectiveness.

The following standards are useful to ensure the quality and compliance of the product.

ISO 14981:2019: Medical devices – Application of risk management to medical devices

This document specifies terminology, principles and a process for risk management of medical devices, including software as a medical device and in vitro diagnostic medical devices. [5]

ISO 80001-1:2021: Application of risk management for IT-networks incorporating medical devices — Part 1: Safety, effectiveness and security in the implementation and use of connected medical devices or connected health software [6]

IEC/TR 80001-2-1:2012 to IEC/TR 80001-2-9:2017 Technical Reports which provide guidelines on how to apply ISO 80001-1:2021. [7]

ISO 13485:2016 is a standard that specifies requirements for a quality management system for organizations that manufacture or provide medical devices and related services. It covers the entire life cycle of medical devices, including design, production, and distribution. The standard helps demonstrate regulatory requirements for medical devices. [8]

Another useful standard from the German normalisation body is the DIN VDE 0834: Call systems in hospitals, care homes and similar institutions.

It specifies the standards for equipment, planning, erection and operation and Environmental conditions and electromagnetic compatibility.

All other relevant ISO can be found in ANNEX B

2.3 Existing technology

2.3.1 Legacy solutions

Old nurse communication systems, are phone call systems over IP, made up of a Control Station ranging from a simple telephone with a Display that can show the origin of the call and allow bidirectional communication to more complex integrated software suites that allow to position patients in a floor plan.

Modern interfaces for the patient allow it to request different services, from emergency assistance to other functions, such as switching off/on the light.

One of such systems is made by Schrack Seconet Care Communication GmbH. They sell a Control Station named “Multi-Functional CT Touch Displays”, which integrates a telephone terminal which can be used to do calls to groups, room listening and control of door entrances. [9] [10]

The system works along their call panels.



Figure 1. Stainless Steel Call Units by Carecom Figure 2. Nurse Control Station

The disadvantage of these systems is that they require a physical installation of a telephone cable from the Control Station to the room where the Call Panel is located. If such installation is not available, the cost to escalate or set-up the system is expensive, as we are required to perform renovations. Another consequence of this physical constrain is that the patient or nurse is not allowed to interact with the system unless close to the interfaces. Which could lead to the patient not being able to call for help or the nurse missing precious time in critical situations.

If the system allows positioning it shows a fixed position as the device is not portable which limits this functionality.

2.3.2 New state of the art solutions

Novel solutions have moved from a traditional wired system to wireless solutions, which solve the main disadvantage of legacy systems. The scalability now is not depended on the ability to have sockets where you can plug your Call units. For example, the IP-DECT 400 COBS TALK system, which is made up of phones and a necklace wireless device, which allow nurse and patients to communicate over voice and call for assistance. For achieving this Wi-Fi technology is used as physical and data link layer for small distances, Wi-Fi AP are positioned around the facility, each AP is linked using an ethernet physical layer to achieve communication over large distances. A COBS Message server CMS, receive all data and processes it, handling requests from all the devices. [11]

It also has a display that allows nurses to know the state of each patient.

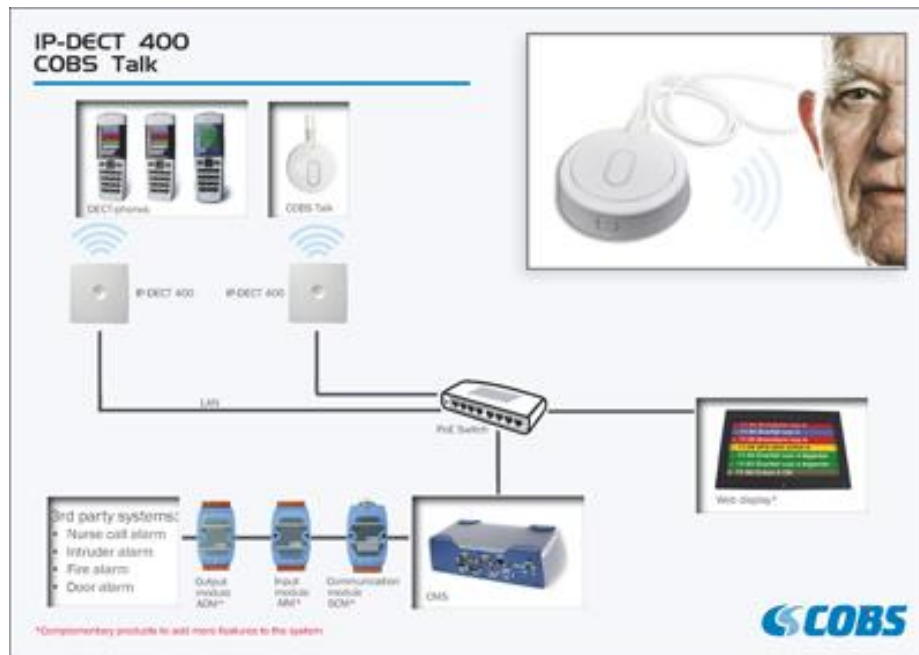


Figure 3. IP-DECT 400

As a RTLS solution they offer another product which can be integrated with the system, with the use of a RFID signal emitter and a portable receiver. When the signal is captured by the receiver the identification signal is sent to the CMS which takes further action. [12]

One downside of this combination is the need of using two products to achieve real time location and voice communication, increasing the complexity of the system. The manufacturer specifies that the range of each emitter is up to 1-4m which can be short ranged. [12]

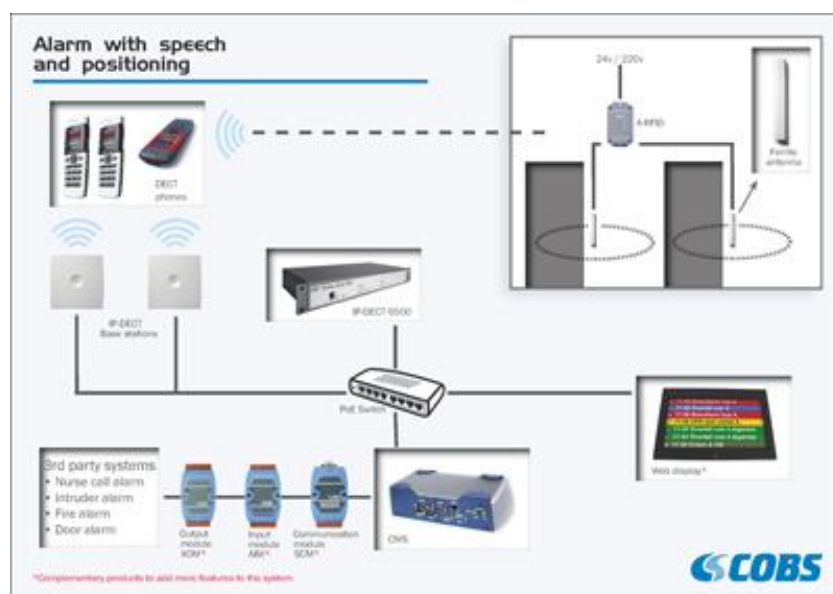


Figure 4. Alarm with speech and positioning

Another company is Stanley Healthcare, their Aeroscout line of RTLS products use ultrasound technology. The system consists of a device named by Stanley Healthcare as exciter which emits signals that are detected by tags wore by the patients. Then the tags emit their position to a server using Wi-Fi technology. [13]



Figure 5. Stanley Aeroscout Tag

The tags used by the patient also allow to do emergency calls to nurses.

All the information and devices can be managed from their proprietary software, MobileView , a platform that integrates and unifies various sources of visibility data, including RTLS, Passive RFID, and sensor data, to provide real-time location, status, and condition information for assets. It has an open API and out-of-the-box adapters for integration with third-party systems and offers a range of applications including Asset Management, Environmental Monitoring, and Infant Protection. MobileView also has tag management capabilities, including over-the-air updates and reports and alerts on tag battery levels, tag assignment history, and other information. [14]

3 Methodology

In this chapter we discuss the process of system design from the understanding of stakeholder's requirements to the definition of the components needed to create the device and their interfaces.

3.1 System Architecture

To design the system, we must first understand the stakeholders' requirements. For this project, we interviewed professor and nurse Dr. Adelina which carefully explained us the requirements she found most important. First, the device should be portable, wireless, battery powered with at least 24h battery life and the interface machine-patient should be easy to use. It must allow bi-directional patient-staff communication and real-time positioning.

Once we understand the requirements, we search for projects that have used RTLS technology or audio, both professional and amateur.

Amateur projects related to real time audio streaming and RTLS with the most common Development Boards available were studied, such as ESP32, STM32, and Arduino boards.

Notable mentions:

Chris Greening ESP32 Walkie-Talkie:

Chris Greening made an amazing Walkie-Talkie using the ESP32 mcu. The system consists of an ESP32 with an I2S microphone and an I2S Amplifier, data captured is streamed over UDP broadcast or ESP-NOW. [\[15\]](#)

Localino: Open-Source Indoor Positioning System (Arduino + Decawave)

A project posted in Instructables, a webpage where creators share how they created their projects but now has grown to full sized marketable company. They use a DWM1000, ESP8266-07 with a custom-made PCB. The system allows to track the position DWM1000 modules. [\[16\]](#)

This helped us identify the different components that our system would require.

- Sound System, broken down into two subsystems, the recording system and the playback system.
- A Communication system, to transmit and receive data from/to other devices.
- A RTLS system, which would locate the device in the space.
- A physical interface for the patient.
- A master Mcu which would coordinate all the components of the system.
- A power supply system with a battery incorporated that will feed our tags.
- A server that will handle alarms, position and audio being stream from devices and will route it to its final user.
- A web UI for the medical staff to interact with the system.

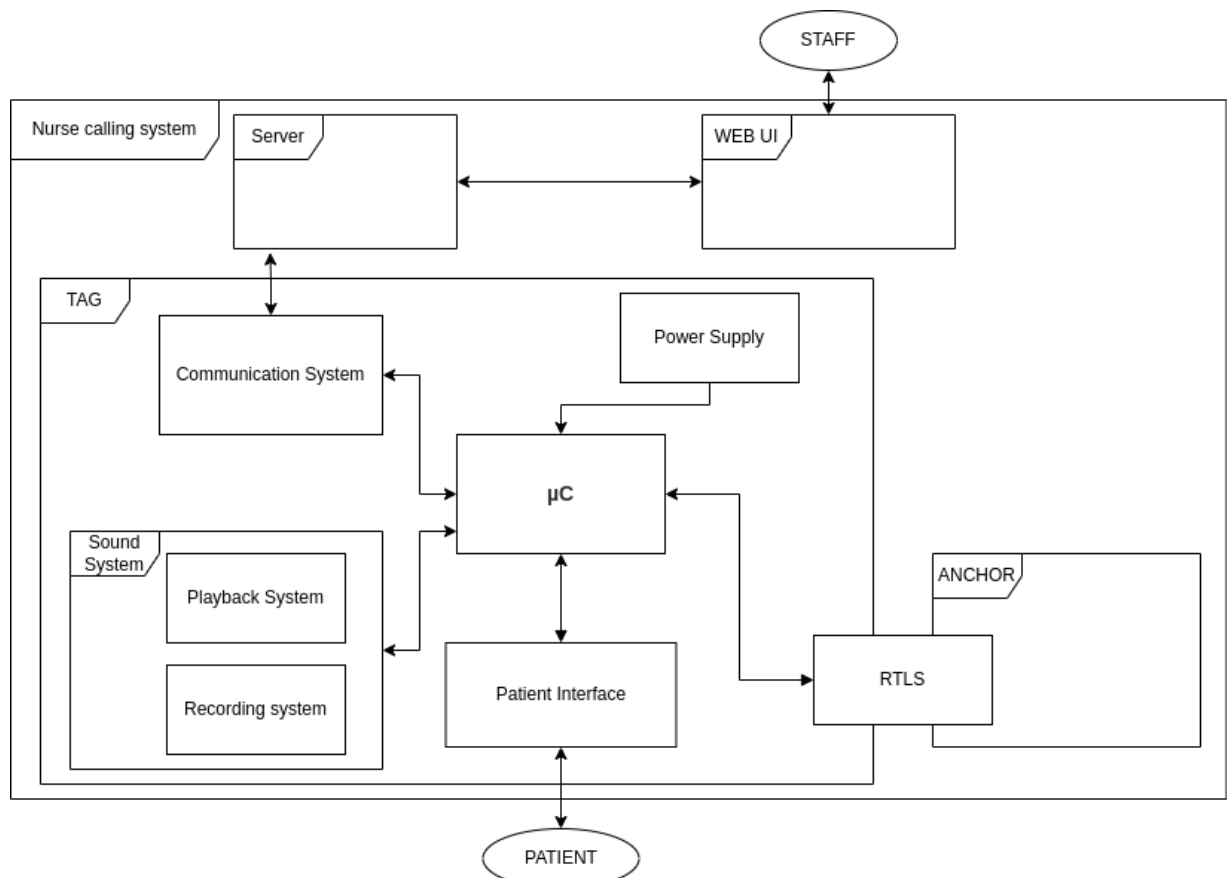


Figure 6. System Architecture Components

Once we have broken our needs on different components, we should start defining them based on our interfaces and needs.

Both the development of a server and web UI are out of the scope of this project.

3.2 Subsystem Architecture

The main constrain in this architecture is the use of the module DWM1001-DEV for RTLS. Its interface will be defined. It will be the first subsystem to be studied and it will define the interface between it and other subsystems.

3.2.1 RTLS Subsystem

The DWM1001-DEV is a development board for the DWM1001 module. It is a hardware platform that includes a DWM1001 module and other components, such as a microcontroller, connectors, and headers.

The DWM1001-DEV board provides access to the various features of the DWM1001 module, including the BLE and UWB radios, and allows developers to interact with the

module using a variety of interfaces, such as USB, UART, and SPI for configuration and control. [17]

The subsystem works independently of the system, and we only need to fetch the position of the DWM1001-DEV board using its interfaces.



Figure 7. DWM1001-DEV

3.2.1.1 Choosing the right Interface

The USB this interface will be used to flash the device or debug it, for the purpose of communicating with the Master Mcu is discarded.

UART (Universal Asynchronous Receiver/Transmitter) and SPI (Inter-Integrated Circuit) are two common communication protocols that are used to transfer data between electronic devices.

UART uses two wires, TX, and RX (plus VDD and GND) while the SPI uses 4 wires MOSI, MISO, SCLK and SS/CS (plus VDD and GND).

UART allows communication between two devices while SPI allows for multiple devices connected to the same bus.

UART is asynchronous each device must be configured to the same bus speed to understand the data being sent. In SPI, the master shares a clock which is used to synchronize devices, which allows for more speed as the UART protocols uses start/stop and parity bits to ensure correct data transmission. [18] [19]

The data to be sent from the Master Mcu to the DWM1001-DEV to request its position has a size of two bytes and the response size is 18 bytes. [20]

Considering the low data intensity of the interface, the number of wires that the SPI protocols use in comparison to UART, 4 vs 2, which might be needed by another component of the system or for future components. **UART is the most suitable option.**

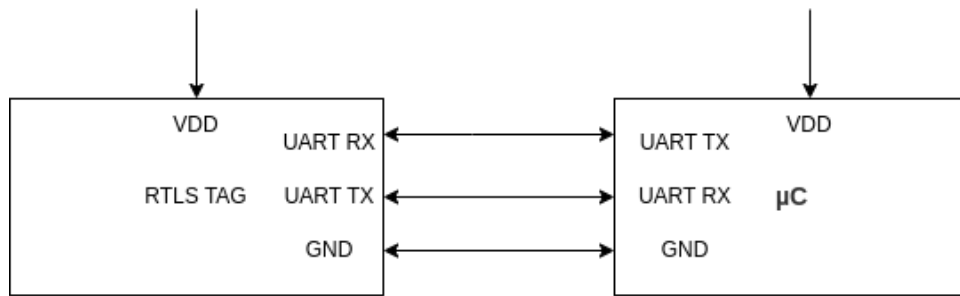


Figure 8. Interface of RTLS and MCU components

3.2.2 Communication system

The main function of the Communication system will be to transmit audio data, position data and control data from the MCU to the server.

We should be able to transmit the data to an access point in the same building that could be located tens of metres away and using wireless methods.

3.2.2.1 Choosing a Physical/Data-Link layer Network Protocol

We cannot use methods that require cables, such as Ethernet.

The most used protocols for establishing and maintaining connections between devices on a wireless network are Bluetooth and Wi-Fi.

Most hospitals have a private Wi-Fi network used to transmit medical information from all their connected devices.

We also require a server. Servers connect to networks using the IP protocol, the most common data-link layers under the IP network protocol are ethernet and Wi-Fi. [21]

Also, not all computers come with Bluetooth. [22]

Most consumer products that use Bluetooth are classified as class 2. Its range would be up to 10 m while using 2.4Ghz Wi-Fi the range is up-to 45m.

Wi-Fi Bandwidth is about 60 times larger than Bluetooth, 54 Mbps vs 3 Mbps. [23]

The logical choice would be to use the already existing hospital Wi-Fi infrastructure to reduce costs and integration problems as not all devices might have Bluetooth interfaces and most servers use the IP protocol, which underlying layers are either ethernet or Wi-Fi.

3.2.2.2 Integrated or separate Wi-Fi Module

We can buy a separated Wi-Fi module that should then interface with the Mcu with high data bandwidth protocols such as SPI or I2C or buy an integrated Mcu with Wi-Fi modules.

For this project the integrated solution has been chosen, a built-in solution has lower risk. If separate module had to be bought, the integration of hardware and software would be time consuming, have higher technical risk and cost than an off the shelf solution.

3.2.3 Sound System

The Sound System is divided into two subsystem which do not work together but are treated as subsystems of the same because of their function.

The Recoding system will capture audio and send it to the Mcu through an interface.

The Playback system will read audio being sent through an interface from the Mcu and play it back using speakers.

3.2.3.1 Analog Audio Processing

The only way to send Audio over Wi-Fi will be using Digital Audio, as we are sending bits of data. That means that either the Sound System will carry the duty to convert Analog Audio to Digital Audio or it will be the Mcu.

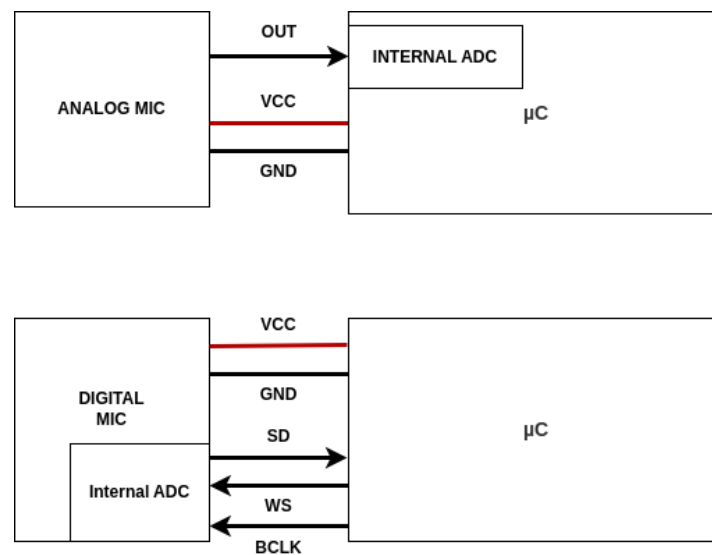


Figure 9. Analog vs Digital Audio Processing Architecture

When Audio is captured by a microphone, sound waves are converted into Voltage, producing a continuous voltage signal over time with varying amplitude. To convert it into Digital ADC converters are used, they take sample of the amplitude of the Analog signal at a certain rate (44.1 Khz for CD-quality audio) then at each sampling point, the amplitude is measured and converted into a digital representation, such as 16,24 or 32-bit binary number.

The number of bits is called audio bit depth. It is the number of values it can represent. If we increase the audio bit depth, we can capture more changes on the voltage value. [24]

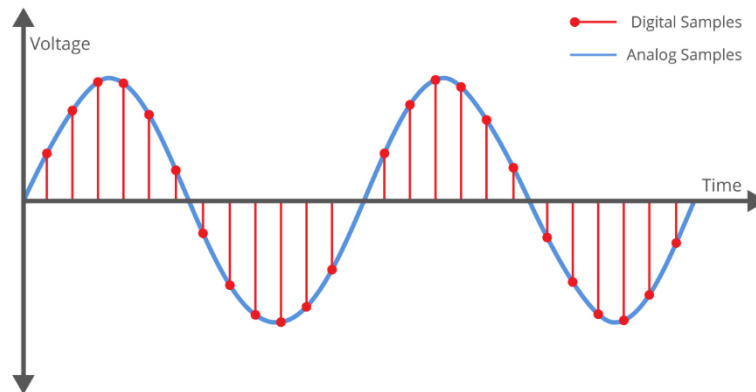


Figure 10. Analog to digital conversion

If we want to convert back digital audio to Analog audio we use a DAC, a digital-to-analog converter receives a digital input which is a series of bit binary numbers which represents the amplitude of an analog signal. [24]

Some Mcu have DAC and ADC who carry this duty, but the bit width of the ADC goes from 10 to 12 bits for the most known DEV boards (ESP32, STM32 and Arduino). The bit width of the DAC's goes from 6 to 12 bits.

Sometimes Microphones have an internal ADC that convert analog to digital, they can range from 8 to 32-bit width. [25][26][27]

Using peripherals for those tasks relieves the Mcu processing power to do other tasks while also getting better quality audio, as they have more bit width.

For this reason, the preferred solution has been to process analog audio in the Audio system before sending to the Mcu. The preferred interface used in the world of audio is I2S. Peripherals that output digital audio or read digital audio will use it and most Development boards like ESP32 and STM32 have I2S interfaces.

I2S interfaces consist of three lines, Serial Data (SD) line which carries the actual audio data, Clock (SCK) line which provide the timing reference for the data transfer, Word Select (WS) line which indicates the start of a new audio data sample. Our Mcu must handle two I2S interfaces.

The Recording subsystem will have a microphone that outputs digital audio in an I2S format with at least 16-bit width resolution, which is then sent to the Mcu.

The Playback subsystem will have an I2S DAC converter and for better audio it could have a filter and amplifier but might not be needed for a prove of concept and a speaker.

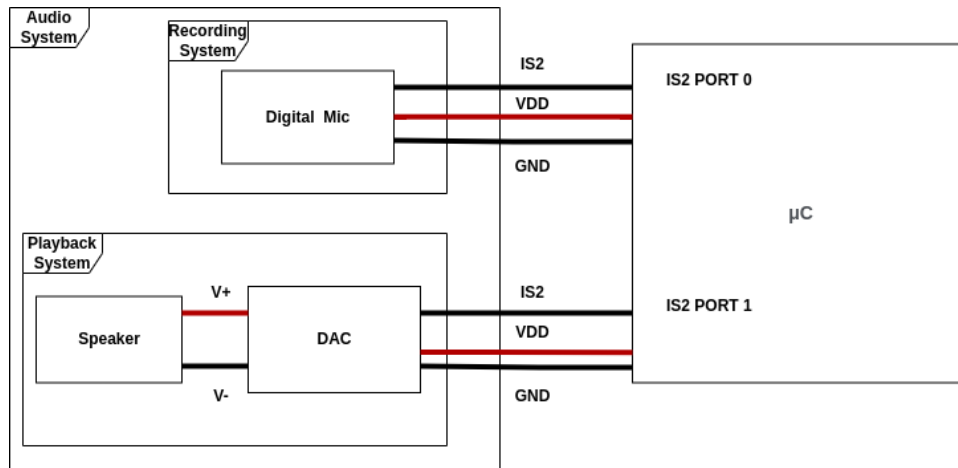


Figure 11. Audio System Architecture

3.2.4 Patient interface

Solutions range from the traditional push buttons used on nurse calling systems to date to more innovative ways like speech recognition which would be useful on a lot of cases.

For the scope of this project the most innovative ways have been discarded. They require a lot of work to make them correctly functional and would be a great continuation of this project.

As per the standard DIN VDE 0834 for Nurse Call Systems the call button must be red with a clear pictogram and must be illuminated when dark.

For this purpose, we will need a push button and a red led. To get the state of the push button we will need a GPIO interface and to control the led a GPIO PWM interface.

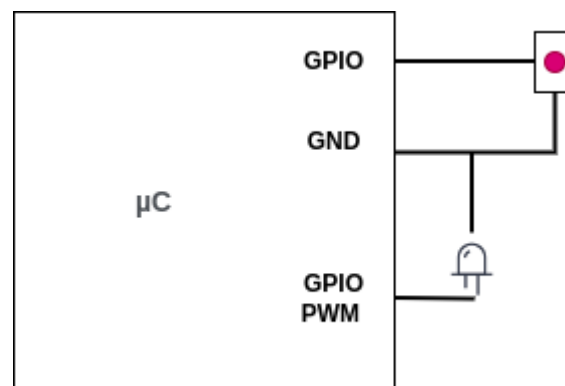


Figure 12. Interface System Architecture

3.2.5 Power Supply

The DWM1001-DEV Kit has a voltage level between 3.6V to 5.0V while most Mcu have a 3.3V or 5V pins to power the Mcu or an USB type B port.

The power pins on Mcu can also be used to power peripherals.

It is important to know the available power resource the Mcu have before choosing a strategy for the power supply.

3.2.6 Master Mcu

Since we know all the peripherals that will be connected to the Master Mcu we can set the requirements:

- Two I2S ports. Two I2S ports mean 6 lines and 6 dedicated pins in the Mcu.
- Wi-fi Module integrated.
- 2 GPIO for the Patient Interface.
- One UART port for communicating with the DWM1001-DEV board.
- Pins for debugging with J-link.
- Extra pins for future features.
- TensorFlow supported platform for adding speech recognition capabilities.
- 5V/3.3V power pins.

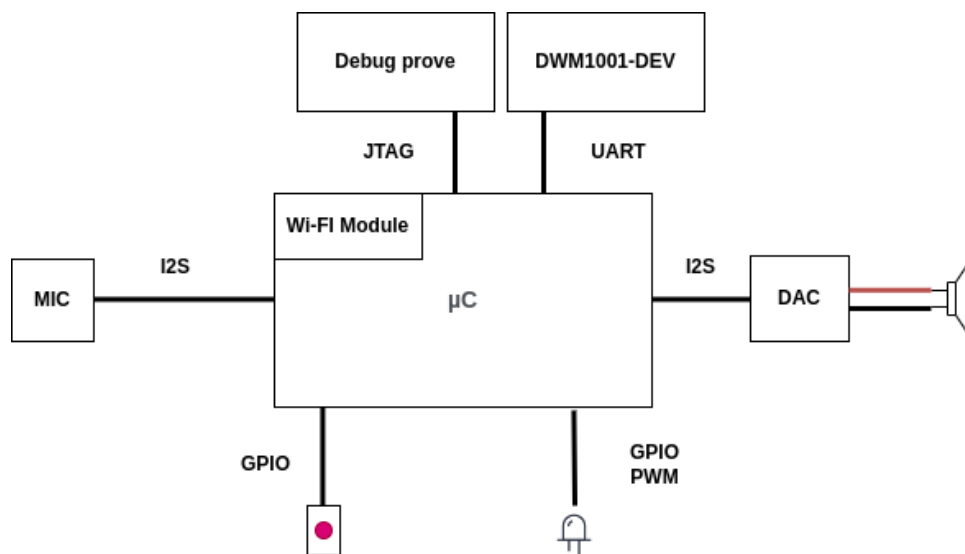


Figure 13 System Device Architecture Sketch

4 Electronic components

Based on the components defined on the system architecture we find the appropriate solutions offered in the market.

4.1 Master Mcu

To select the Master Mcu we focused on the most restrictive constrain, the Mcu must have support to run TensorFlow.

The supported Development boards are: [28]

- [Arduino Nano 33 BLE Sense](#)
- [SparkFun Edge](#)
- [STM32F746 Discovery kit](#)
- [Adafruit EdgeBadge](#)
- [Adafruit TensorFlow Lite for Microcontrollers Kit](#)
- [Adafruit Circuit Playground Bluefruit](#)
- [Espressif ESP32-DevKitC](#)
- [Espressif ESP-EYE](#)
- [Wio Terminal: ATSAMD51](#)
- [Himax WE-I Plus EVB Endpoint AI Development Board](#)
- [Synopsys DesignWare ARC EM Software Development Platform](#)
- [Sony Spresense](#)

Only these 4 boards have a Wi-Fi module.

- Expressif ESP32-DevKitC
- Expressif ESP-EYE
- Wio Terminal: ATSAMD51
- Synopsys DesignWare ARC EM Software Development Platform

Expressif ESP-EYE has been discarded because it does not have enough Pins for GPIO, the Wio Terminal: ATSAMD51 comes with an encapsulation and a screen device which we do not need so it has been discarded and the Synopsys DesignWare ARC EM Software Development Platform is a powerful platform, but it is sold at 950 euros per unit.[29]

The logical choice is one of the Expressif ESP32-DevKitC boards. We have selected the ESP32 Series because it matches our requirements, and it is an affordable Mcu with a powerful API, called ESP-IDF and are CE and FCC certified which is important in order to get our medical device market approval. [30]

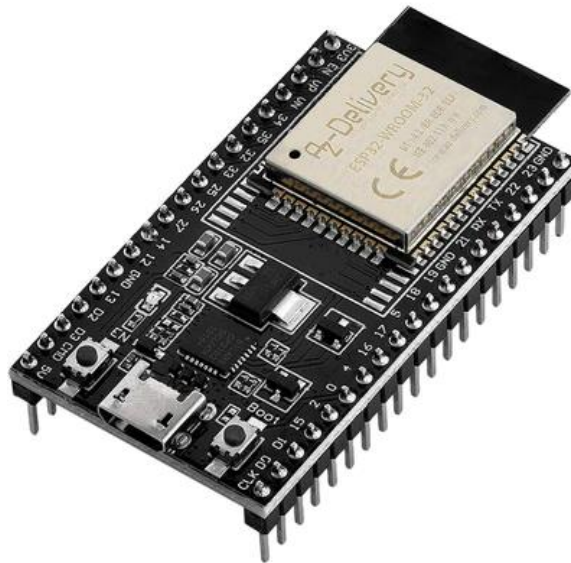


Figure 14.ESP-32 Dev Kit C V4

Some of the major features of the ESP32 DevkitC include:

- Wi-Fi: The device has built-in Wi-Fi, which can be used to connect to the Internet or other devices wirelessly.
- Bluetooth: The ESP32 DevkitC has dual-mode Bluetooth, which allows it to support both classic Bluetooth and Bluetooth Low Energy (BLE).
- Security: The device has a built-in hardware cryptographic engine for secure communication.
- Low-power operation: The ESP32 DevkitC is designed to be energy efficient, with a deep sleep current of less than 5 μ A.
- Development support: The device is supported by a range of development tools, including the Arduino Integrated Development Environment (IDE) and the Espressif IoT Development Framework (ESP-IDF).
- Input/output (I/O) capabilities: The ESP32 DevkitC has a wide range of I/O capabilities, including:
 - Digital I/O pins: 36 total, of which 4 can be used as pulse-width modulation (PWM) outputs
 - Analog input pins: 12 total, with a resolution of up to 12 bits
 - Interfaces: The device has I2C, I2S, and SPI interfaces, as well as a UART interface for serial communication.
- 5V and 3.3V pins.
- Micro USB type B.

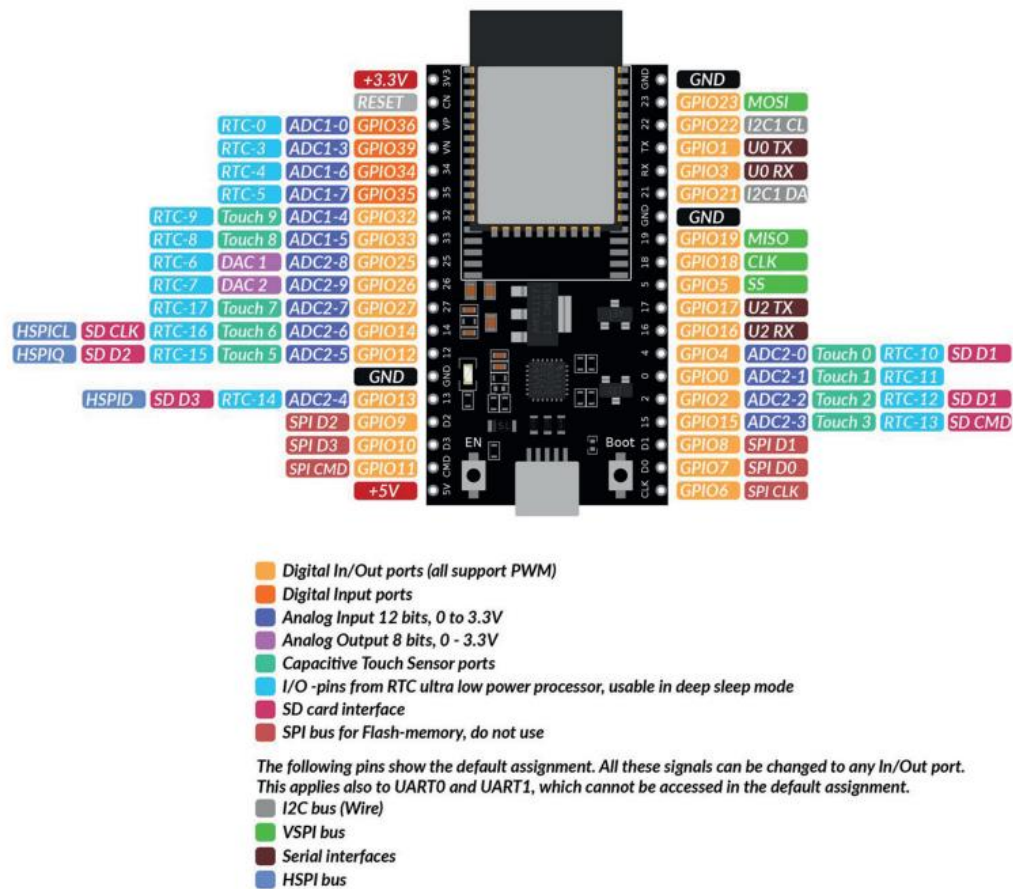


Figure 15. ESP-32 Dev Kit C V4 Pinout

4.2 Power Supply

The ESP32 can be powered in three ways [31]:

- Using a 3.3V pin but the voltage must be regulated with an external circuit as it does not have an in-built regulator.
- Using a 5V pin with an in-built voltage regulator AMS1117-3.3 that converts it into 3.3V.
- Using a Micro USB type B.

For this project we will power the ESP32 using an external battery with an USB output connected to the ESP32. Freeing the 3.3V and 5V pins we are able to power all the peripherals attached to the ESP32 including the DWM1001-DEV module with the 5V pin.

For fast prototyping we have used an old power bank. The main characteristics are [32]:

- 5V supply.
- 5000mAh capacity Li-Po battery.
- 2x USB 2.4A output ports
- 1x Micro USB input port.

4.3 Audio System

4.3.1 Recording system

A MEMS microphone (Microelectromechanical Systems microphone) is a type of microphone that uses a tiny mechanical structure to detect sound waves and convert them into an electrical signal. MEMS microphones are small and lightweight, and they are often used in portable devices such as smartphones and laptops. They have an ASIC (Application-specific integrated circuit) which includes a pre-amplifier and/or an ADC. They have several advantages over other types of microphones, including high sensitivity, low power consumption, and good resistance to noise and distortion. [33]

<https://www.cuidevices.com/blog/comparing-mems-and-electret-condenser-microphones>

An Electret condenser microphone (ECM) is a type of microphone that uses a thin, electrically charged diaphragm to detect sound waves and convert them into an electrical signal. The diaphragm is positioned close to a fixed metal plate, and the electrical charge on the diaphragm changes as it moves in response to sound waves. This change in electrical charge is then amplified and converted into an audio signal. They are known for their high sensitivity and ability to accurately capture a wide range of frequencies. They require an external power source.[34]

For this project we have selected a MEMS microphone as it is smaller, lighter, has lower power consumption, digital output and does not need an external power source.

We need a MEMS microphone with digital output (I2S) breakout board, a breakout board is a PCB that has been designed to connect the pins of the integrated circuit to external devices.

The available breakout boards are:

Table 1. Technical Characteristics of MEM Mics

	ICS-43434 [37]	INMP441 [36]	SPH0645LM4H-B [35]
Sensitivity	-26dBFS	-26dBFS	-26dBFS
SNR	65dBA	61dBA	65dBA
Current at 1.8V	490 μ A	1.4 mA	600 μ A
AOP	120 dB SPL	120 dB SPL	120 dB SPL
Sample Rate	6.25kHz to 51.6kHz	7.8 kHz to 50 kHz	32kHz to 64kHz
Data Precision	24 bit	24 bit	18 bit

Both ICS-43434 and INMP441 are I2S Standard Philips Format, Data signal have one bit shift compared to WS signal and the duty of WS signal is 50%.

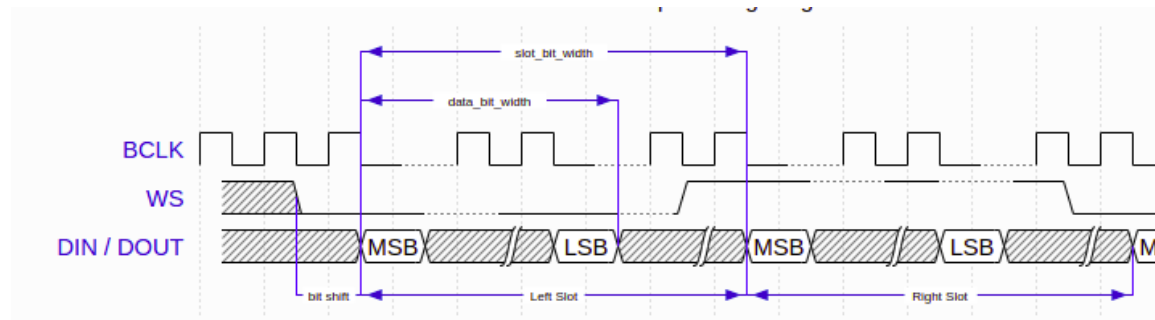


Figure 16. I2S Standard Philips Format

While the SPH0645LM4H-B has an I2S Data Format with MSB delayed 1 BCLK cycle after WS changes.

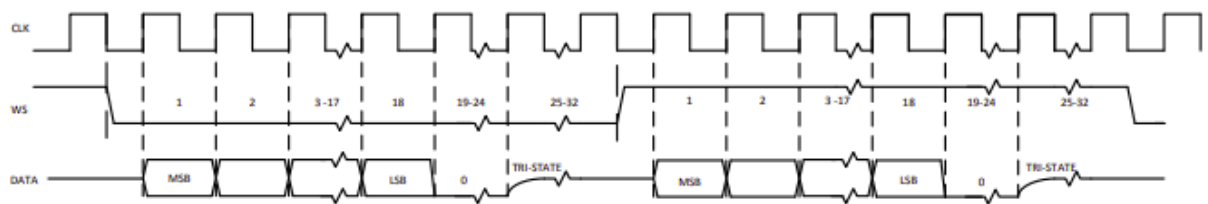


Figure 17. I2S Data Format SPH0645

The best choice overall is ICS-43434. It has the lowest consumption and the best sound quality. ICS-43434 had to be shipped from overseas with long lead times so it was discarded.

The INMP441 Data format is supported by ESP-IDF and has higher data precision than SPH and samples are cheaper. Decision to buy four samples was made but after testing them they were defective.

A compromise with SPH0645LM4H-B was made, two samples were bought from trusted seller "ADAFRUIT" despite Data format not supported off the shelf by ESP-IDF API. It could be modified to support it.

4.3.2 Playback

System

We need a I2S DAC breakout board that takes I2S audio and outputs analog audio, the only available option is the MAX98357A/B. The MAX98357A/B breakout board works with 4/8 ohms speakers. Major features are: [\[38\]](#)

- 16/24/32-bit I2S data input supported.
- Sample Rate of 8kHz to 96 kHz
- Voltage 2.5V to 5.5V
- 2.4mA Consumption
- No MCLK required
- 92% power efficiency

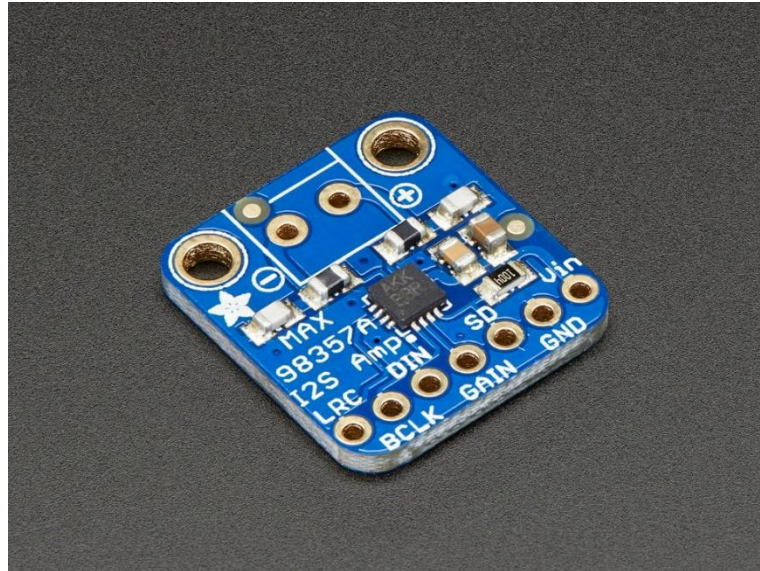


Figure 18. I2S DAC MAX98357A/B

The output of the DAC is sent to an 8-ohm speaker which will convert the analog signal into sound.

We have selected some standard 8-ohm 2W speakers from Weewoday. [\[39\]](#)



Figure 19. Weewoday Speakers

4.4 DWM1001-DEV

The DWM1001-DEV module can be configured as anchor or tag to build a RTLS.

Characteristics [40]:

- Supports UWB & Bluetooth®
- Access to all DWM1001 GPIOs and interfaces via on board headers
- Software APIs for application customization
- Module APIs for configuration and interfacing over SPI, UART, BLE Power Sources.
- Power Source from PIN (5V), USB(5V) or battery (3.6V to 5.5V).
- Current level 500mA.

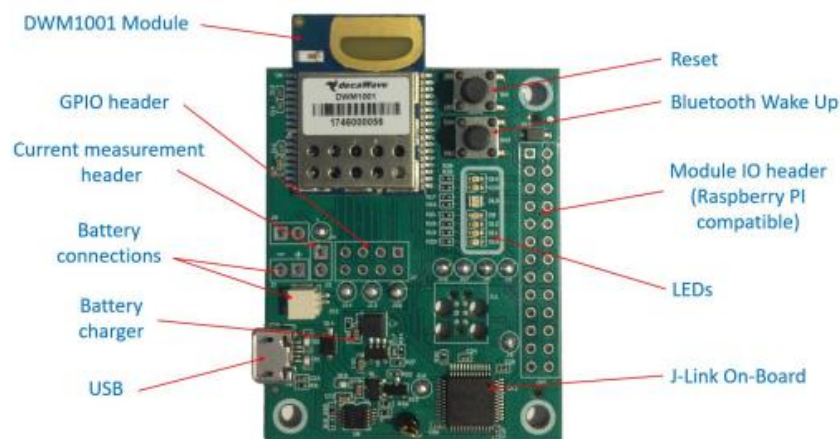


Figure 20. DWM1001-DEV Technical Data

4.5 Patient interface

For the patient interface we have bought standard push buttons from Gerbiled, which include a red cap.

The led was not bought as the budget was exhausted and it was the least critical component of the prototype. For future work it will be included.

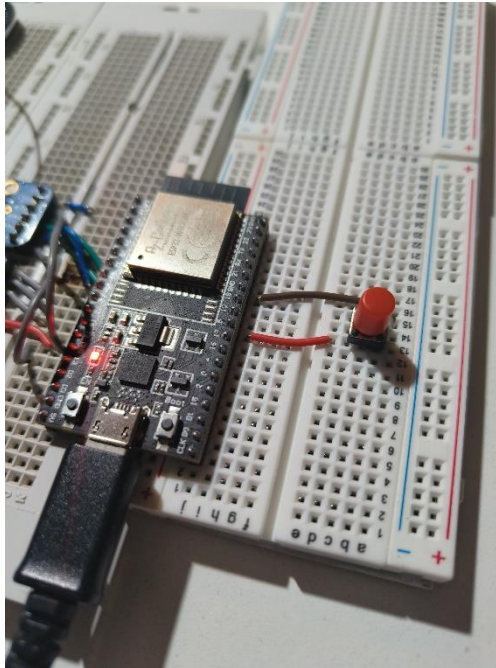


Figure 21. Push button installed on the system

5 Tools

5.1 Soldering and electronics testing

Breakout boards come without pins header soldered to them. For this reason, we need a soldering iron.

A solder Iron is used to heat and melt alloy of tin to join pins header to the breakout board.

For this project, a cheap solder iron was bought from amazon. For more advanced projects a solder iron which temperature sensor is recommended.

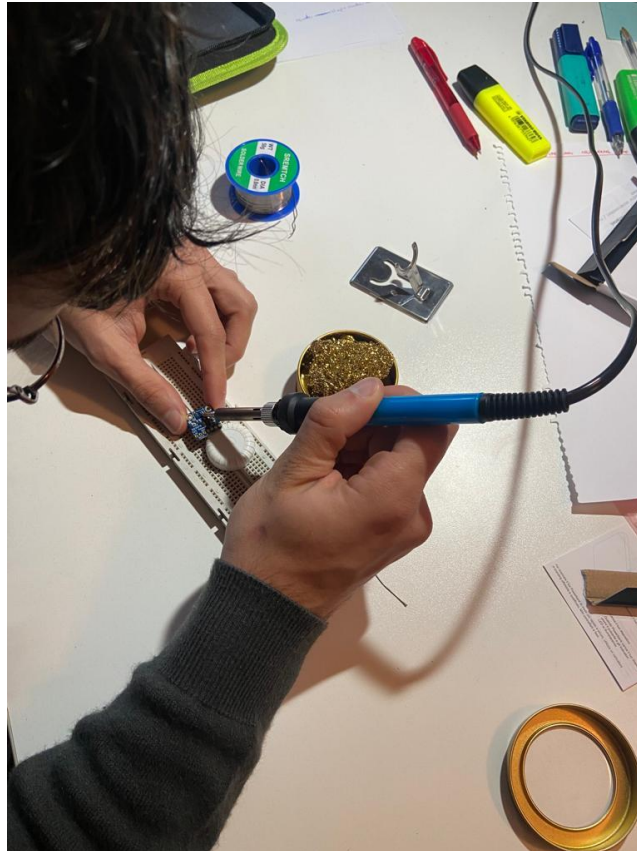


Figure 22. Work In Process. Soldering

Once we have soldered the items, we connect the devices to power and use a multimeter to test for short circuits.

5.2 Programming the Esp32

To write code in VSCode we will use the ESP-IDF framework, it allows us to create firmware for their products thanks to the API calls. The framework is written in C. We will also write in C the firmware.

5.2.1 Hardware

- ESP32-DevkitC
- USB cable. USB A/ micro-USB B.
- Computer running Linux but Windows or macOS are available.

5.2.2 Software

- VSCode IDE
- VSCode Expressif IDF Extension.

5.2.3 Installation

- Download and Install VSCode IDE from <https://code.visualstudio.com/> .
- Install VSCode Espressif IDF extension <https://github.com/espressif/vscode-esp-idf-extension/blob/master/docs/tutorial/install.md> .

5.2.4 Start a project

For first use it is recommended to read the tutorials on VSCode ESP-IDF extension github.

[VSCode ESP-IDF basic use](#)

The firmware will be divided in components. A component is a self-contained piece of code that provides a specific functionality. Components can be added to an ESP-IDF project by including them in the project's component.mk file or by using the `idf_component_register()` function in the project's CMakeLists.txt file, so they can be compiled and linked. [42]

Components help structure the project and make them more readable and easier to understand.

We have divided components based on their functionality, this allows for easier maintenance of the code and portability as it can be reused in other projects.

The structure of our project is:

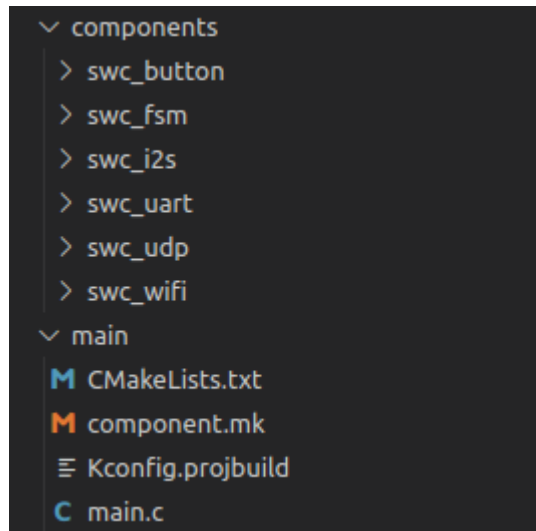


Figure 23. Code Structure

- Swc_button implements logic for managing the button interface.
- Swc_fsm is used to create the global variable that holds the state of the system.
- Swc_i2s configures the i2s peripherals used for audio
- Swc_uart configures the uart for communication with the DWM1001-C module and communicates with it.
- Swc_udp implements the communication logic with the server. It sends and receive data from it.
- Swc_wifi configures the access to the Wi-Fi network used by the ESP32.

5.3 JTAG Debugging the ESP32

Some of the key features of JTAG debugging include the ability to read and write memory, set breakpoints, and perform single stepping (executing one instruction at a time). This can be particularly useful for identifying and fixing problems in a hardware design.

5.3.1 Hardware

- JTAG Adapter Hardware (J-Link Pro from Segger)
- ESP32

5.3.2 Software

- Eclipse
- OCD

5.3.3 Set-up

- Set up ESP IDF tools after installation of ESP IDF VSCode Extension using the following commands from the Linux Terminal. This will install OpenOCD.

```
cd ~/esp/esp-idf
```

```
./install.sh esp32
```

- Install the J-link Software and Documentation pack from their website.

<https://www.segger.com/downloads/jlink/>

- Connect the ESP32 pins to the J-link Pro:

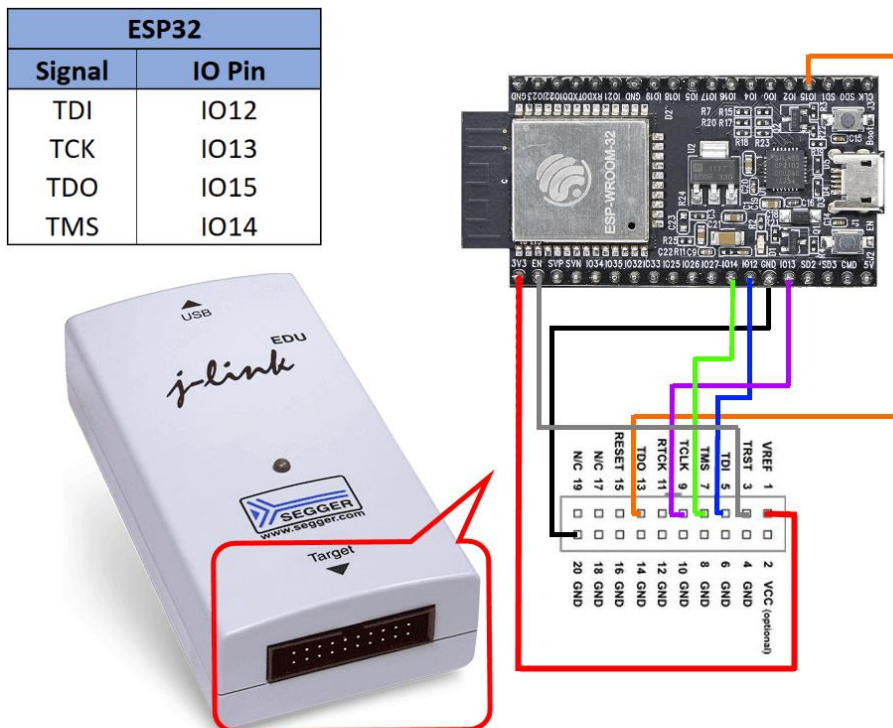


Figure 24. ESP32 Jlink Connections

- Modify the jlink.cfg file located at /home/hen/.espressif/tools/openocd-esp32/share/openocd/scripts/interface and add the jlink adapter speed:

```
adapter speed: 20000 kHz
```

- Install Eclipse IDE for C/C++ Developers 2022-09 (2021-06 to 2022-09 should work fine) from here. [Eclipse IDE for C/C++ Developers Download](#)
- Install the IDF Plugin using the existing ESP-IDF configuration. Install the Embedded C/C++ OpenOCD Debugging extension for debugging the ESP32. More instructions at: [ESP-IDF Eclipse plugin](#)
- Once installed go to [Create a new debug configuration](#). Follow the instructions. For this project, the debug tab should be configured as:

OpenOCD Setup

☒ Start OpenOCD locally

Executable path:

Actual executable:
(to change it use the [global](#) or [workspace](#) preferences pages or the [project](#) properties page)

GDB port:

Telnet port:

Tcl port:

Flash voltage:

Target:

Board:

Config options:

☒ Allocate console for OpenOCD ☐ Allocate console for the telnet connection

GDB Client Setup

☒ Start GDB session

Actual Executable:

Other options:

Commands:

Figure 25. Eclipse IDE Debug Configuration

5.3.4 Start debugging

To start debugging, with eclipse IDE open, select the configuration on the scrollable menu on the right. In this case `udp_client`. And the debug mode in the left scrollable menu. Then click on bug symbol to start debugging.



Figure 26. Eclipse IDE Control panel

Now we can set breakpoints in different parts of the code. In this case we have a breakpoint in the switch used to set the UDP packets depending on the state of the fsm.

We can also watch variable to track their value at any moment of the program execution.

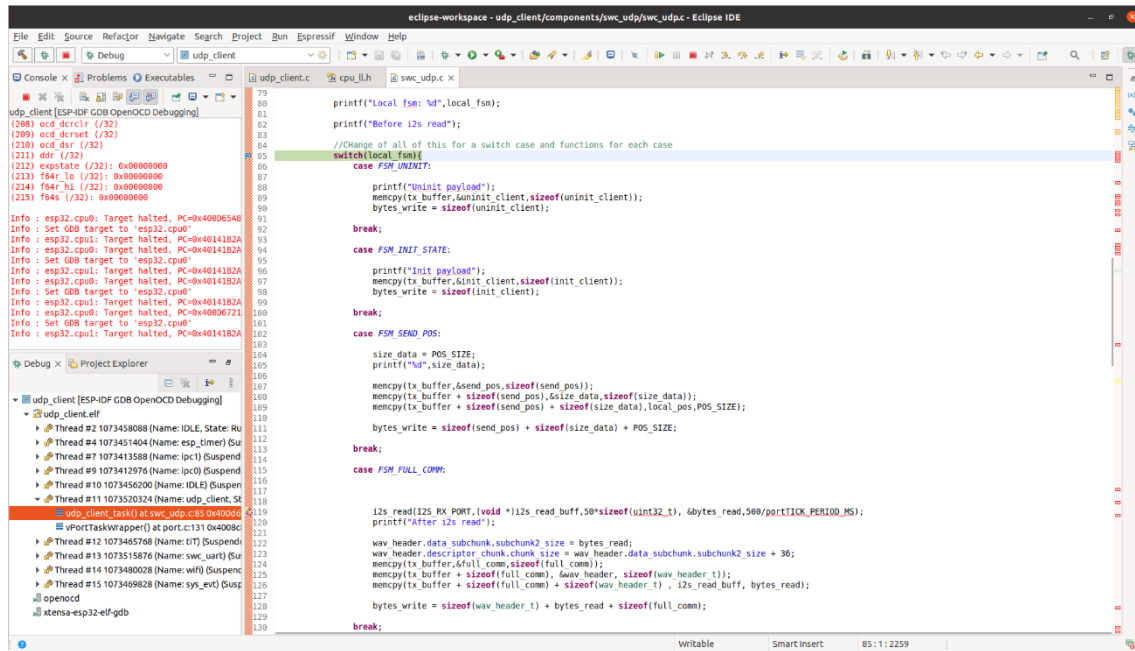


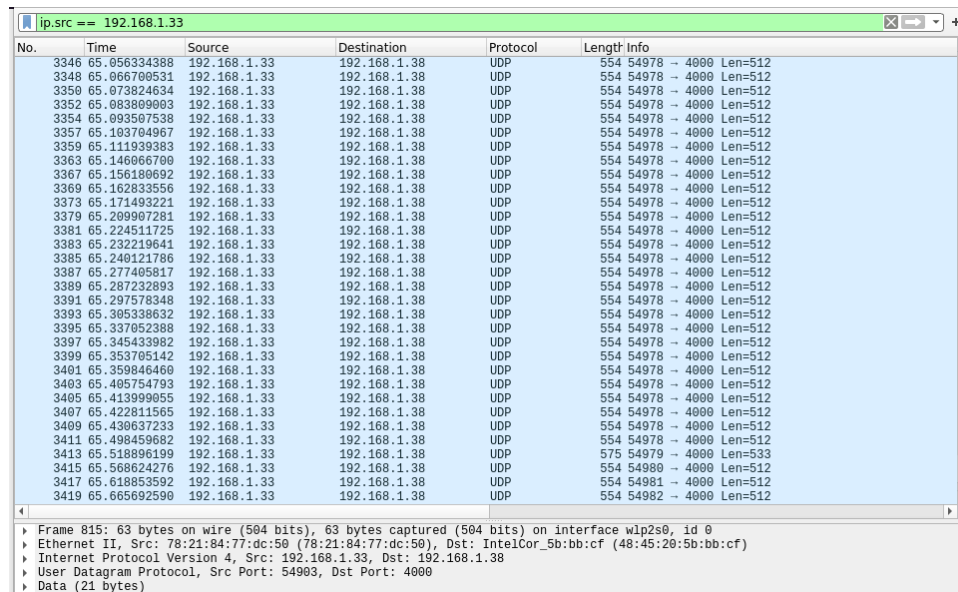
Figure 27. Eclipse IDE Debug session

5.4 Wireshark

Wireshark® is a network protocol analyser. It lets you capture and interactively browse the traffic running on a computer network.

Wireshark allows users to see all traffic being transmitted over a network, including data frames, packets, and segments. It can decode and dissect various protocols, providing a detailed view of the contents of each packet. Wireshark also has a range of features for filtering and searching for specific packets, as well as the ability to save and export packet data. [43]

It was used to analyse the UDP packets between the Mcu and our server. Check for endianness, correct construction of packets and monitoring the timestamp of the sent packets to check if the program flow was correct.



No.	Time	Source	Destination	Protocol	Length	Info
3346	65.056334388	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3348	65.066700531	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3350	65.073824634	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3352	65.083809903	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3354	65.093507538	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3357	65.103704967	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3359	65.111939383	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3363	65.146066700	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3367	65.156180692	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3369	65.162833556	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3373	65.171493221	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3379	65.209907281	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3381	65.224511725	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3383	65.232219641	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3385	65.240121786	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3387	65.277405817	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3389	65.287232893	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3391	65.297578348	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3393	65.305338632	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3395	65.337052388	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3397	65.345433982	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3399	65.353705142	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3401	65.359846460	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3403	65.405754793	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3405	65.413999055	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3407	65.422811565	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3409	65.430637233	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3411	65.490459682	192.168.1.33	192.168.1.38	UDP	554	54978 → 4000 Len=512
3413	65.518896199	192.168.1.33	192.168.1.38	UDP	575	54979 → 4000 Len=533
3415	65.568624276	192.168.1.33	192.168.1.38	UDP	554	54980 → 4000 Len=512
3417	65.618853592	192.168.1.33	192.168.1.38	UDP	554	54981 → 4000 Len=512
3419	65.665692590	192.168.1.33	192.168.1.38	UDP	554	54982 → 4000 Len=512

Frame 815: 63 bytes on wire (504 bits), 63 bytes captured (504 bits) on interface wlp2s0, id 0
 Ethernet II, Src: 78:21:84:77:dc:50 (78:21:84:77:dc:50), Dst: IntelCor_5b:bb:cf (48:45:20:5b:bb:cf)
 Internet Protocol Version 4, Src: 192.168.1.33, Dst: 192.168.1.38
 User Datagram Protocol, Src Port: 54903, Dst Port: 4000
 Data (21 bytes)

Figure 28. Audio packets from ESP32

6 Consideration and decision regarding alternative solutions

The RTLS technology used in this project was one of the constrains but is it the best technology available for our project.

Some examples of RTLS solutions:

- Radio-frequency identification (RFID) systems: Use of radio frequency technology to transmit and receive data from RFID tags or transponders attached to the objects being tracked. Ultra-high frequency (UHF) RFID radio signals are used for RTLS. [44]
- GPS (Global Positioning System): This system uses a network of satellites to determine the location of a device equipped with a GPS receiver.
- Bluetooth Low Energy (BLE) beacon systems: Bluetooth technology to transmit location data from BLE beacons, which can be placed at fixed locations within a facility or on mobile assets.
- Ultrasonic systems: Uses High-frequency sound waves to determine the location of an object or person.
- Wi-Fi-based systems: These use Wi-Fi network infrastructure to determine the location of a device equipped with a Wi-Fi module.
- Infrared systems: It uses infrared technology to determine the location of an object or person based on the reflection of infrared light.
- Visual light communication (VLC) systems: Using visible light technology to transmit location data from VLC beacons.
- Ultra-wideband (UWB) technology: Uses wireless communication to transmit location data from UWB tags or transponders attached to the objects being tracked. UWB-based RTLS systems can provide location data with a high degree of accuracy, making them suitable for use in precision applications. UWB-based RTLS systems can operate indoors and do not require a clear line of sight, making them suitable for use in environments where GPS or RFID systems may not work. UWB-based

RTLS systems are also relatively low power, which makes them suitable for use in battery-powered devices. [45]

Table 2. Technical Characteristic of different RTLS solutions

Parameter	US	UHF	BT	LF	UWB	Wi-Fi	IR	GPS
Coverage area of singular locating node [m]	30	15	10	2	30	50	0.5	Outdoor
Accuracy [m]	0.3*	0.2* 2-7	3	0.2* 5	0.15	2	0.5*	2
Room level usefulness [%]	100	50	50	40	60	50	80	0
RF interference potential	No	H	M/H	M	No	H	L	L
Bit rate	L	M	M	L/M	VH	H	L/M	H
Complexity	L	L	M	M	VH	H	L	H
Initial cost	L	M	M	M/H	H	H	M	H
Security and privacy	VH	M	M	L/M	M/H	M	L	M
Health concern	VL	M	M	L/M	M/H	M	M	M

*Legend: VL – very low; L – low; M – medium; H – high, VH – very high, * for the chokepoint detection use case [45]

Based on the result of studies. Solutions with less than 1m accuracy, more than 10 metre coverage which do not cause RF interferences, important on a medical environment were a lot of devices use RF for operating are US and UWB.

The main advantage of US over UWB is that it has 100% room level usefulness while UWB is 60%. It is a suitable alternative to UWB.

7 Discussion of the solution

The System can be divided into a Realtime Location System, the patient device and the server and its user interface. We will discuss each component on its own and how it has been managed to accomplish the goals.

7.1 Configuring the RTLS

The RTLS system with DWM1001-DEV consists of modules configured as anchors or tags. The tag collects information about anchors by listening to anchor messages. It will calculate the distance to the closest anchor in each quadrant until it leaves the quadrant when it will again search for the closest anchors. [46]

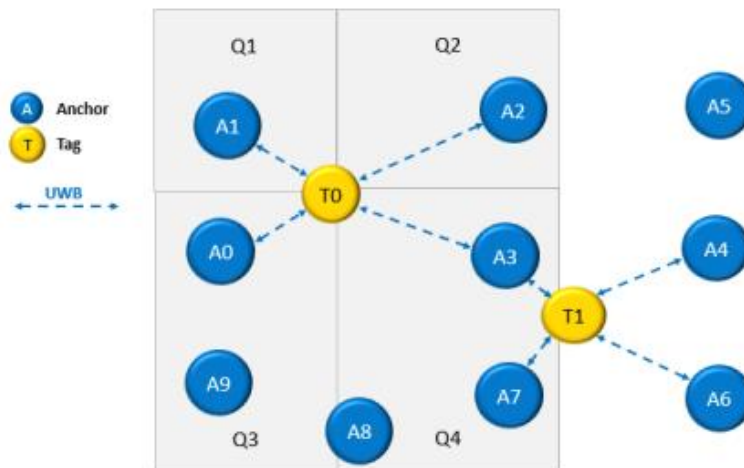


Figure 29. Tags choice of anchors

7.1.1 Network

To achieve optimal performance in a wireless network using anchors, it is recommended to space them 20 to 25 meters apart in a square grid configuration, which is equivalent to a diagonal distance of 28 to 35 meters between opposite anchors. However, in cases where there is no direct line of sight between anchors, it may be necessary to reduce the distance between them to ensure satisfactory performance.

It is important to conduct testing in the specific implementation environment to determine the best spacing for the anchors. This will allow you to determine the optimal distance for your set of conditions and ensure that the network performs at its best.

7.1.2 Scalability

The system expansion for anchors in the network infrastructure consists of assigning each anchor a seat number between 0 and 29, ensuring that no anchor can hear two anchors with the same seat number, synchronizing all nodes with the superframe timing of the initiator, and limiting the maximum clock level to 127.

It can support up to 30 anchors, but if a 31st anchor is added, it must reuse a seat number if the other anchors are spaced far enough apart to prevent any anchor from hearing two anchors with the same seat number. [46]

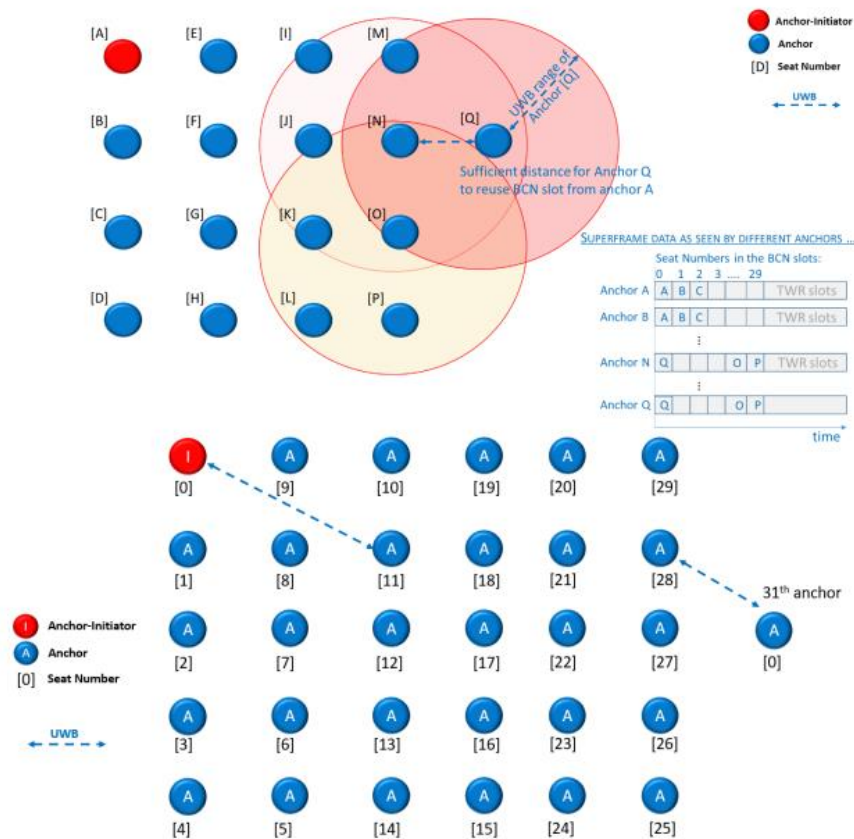


Figure 30: Scaling the DRTLS showing anchor's seat numbers

The system expansion for tags allows for a maximum of 150 Hz system capacity, with tags ranging to up to 4 anchors and being assigned a specific Time Waveform Ranging (TWR) slot. If all TWR slots are already allocated, a new tag will not be able to obtain a slot. [46]

Multiple networks in the same area to share the same air space and not interfere with each other by overlapping messages.

In ANNEX A Configuration of Module DWM1001 as an anchor or tag go is explained.

7.2 Programming the ESP32

All the code discussed here is located at my GitHub: <https://github.com/solubleCopper>.

The finite state machine of our system has three states, UNINIT, INIT and FULL COMM.

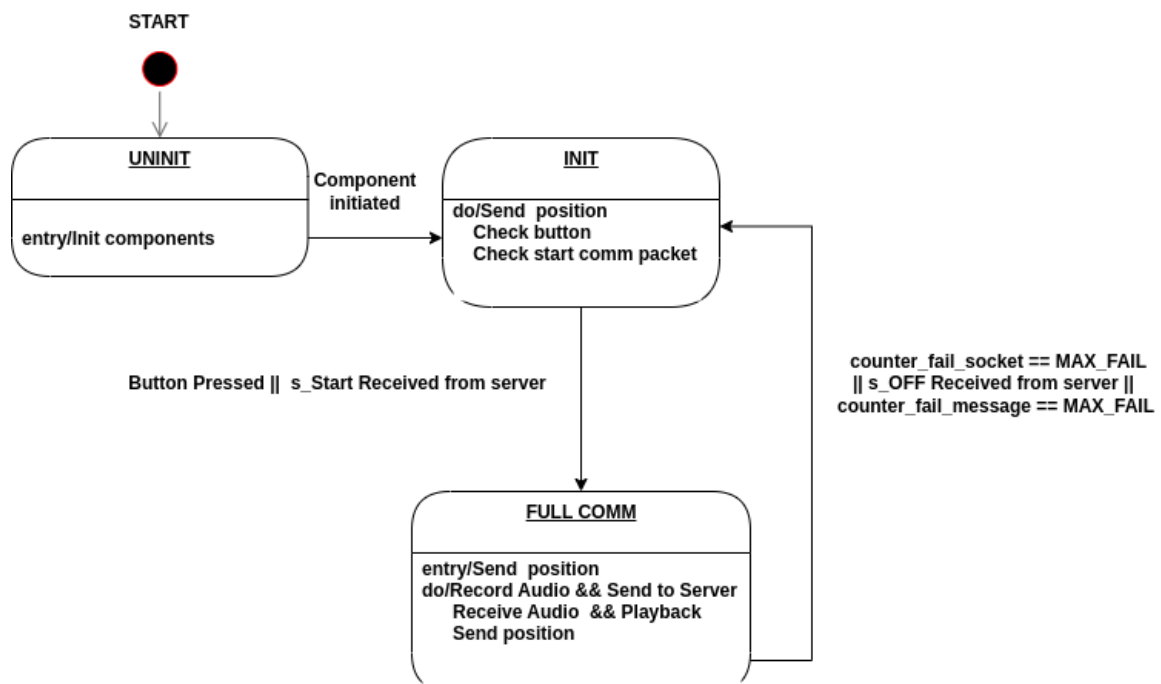


Figure 31: System Finite State Machine

In the INIT state peripherals are initiated, Wi-Fi, I2S, UART, GPIO and component task such as swc_uart and udp_client.

The system transition to INIT state occurs when components have been initiated in the entry point of the system. The function app_main() in the main.c file is the entry point.

```
void app_main(void)
{
    fsm_state = FSM_UNINIT;

    wifi_comp_init();
    i2s_mic_init();

    init_gpio();
    xTaskCreate(vTask_uart, "swc_uart", 4096, NULL, 5, NULL);
    xTaskCreate(udp_client_task, "udp_client", 8192, NULL, 5, NULL);

    fsm_state = FSM_INIT;
}
```

Figure 32: Entry point. App_main.

In the INIT state the position message is sent to the server. The system also checks if the button has been pressed by the patient or the server has sent a start communication packet.

The system transitions to COMM state if the patient presses the button or the server has sent a start communication packet. The udp_client_task monitors connections with the server if start com message is received it transitions the system to com state.

```
if (local_fsm == FSM_INIT && rx_buffer[0] == 0x03 && rx_buffer[1] == 0x02)
{
    local_fsm = FSM_COMM;
    counter_fail_socket = 0;
    counter_fail_msg = 0;
}
```

Figure 33: Transition to FSM_COMM state from received control packet.

The swc_button component monitors the button state, a GPIO interrupt handler is triggered when the button is pressed, the handler transitions the system to com state.

```
static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    if(fsm_state == FSM_INIT){
        fsm_state = FSM_COMM;
    }
}
```

Figure 34: Transition to FSM_COMM from GPIO interrupt

In the COM state the position is sent each minute to the server and the audio pipeline is activated. Packets with audio are sent from the nurse calling system to the server and in response audio packets are sent back and then handled to the i2s component.

The system transitions to INIT state when the socket error counter for receiving and sending messages has reached the maximum, when the counter for incorrect incoming messages has reached the maximum or the server request the client to close communications.

These functionalities are managed from the swc_udp which manages the communications with the server.

7.2.1 RTLS data fetch

The component `swc_uart` configures the UART port of the ESP32 to interface the DWM1001-DEV UART port.

The module interacts with DWM API via its UART interface at baud rate 115200. Figure 34 shows the workflow of the DWM1001 UART interface. In the UART Generic mode communication, the ESP32 device is acting as the initiator, while the DWM1001 module is the responder. [47]

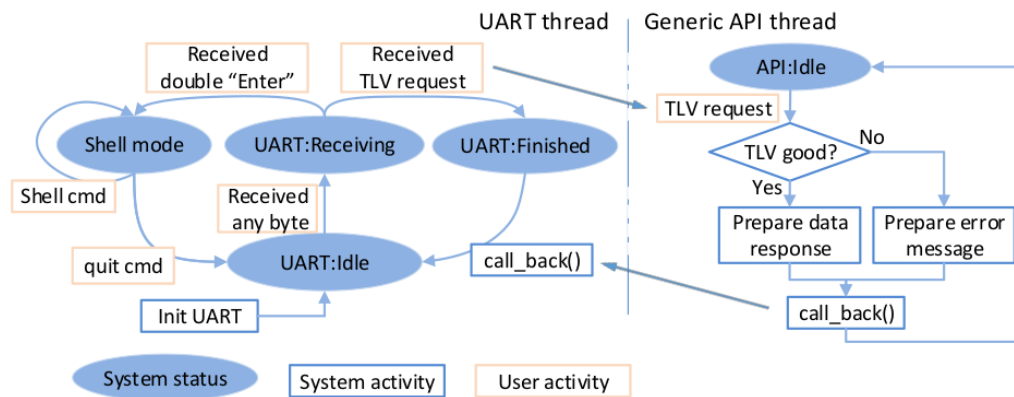


Figure 34: DWM1001 UART workflow

The API TLV `dwm_pos_get` request is used to obtain the position of the node. The request and response are defined as:

Table 3: `dwm_pos_get` request

TLV request	
Type	Length
0x02	0x00

Example response:

Table 4: `dwm_pos_get` response

TLV response							
Type	Length	Value	Type	Length	Value		
		err_code			Position		
					32 bit value in little endian is x coordinate in millimeters	32 bit value in little endian is y coordinate in millimeters	32 bit value in little endian is z coordinate in millimeters
					8 bit value is quality factor in percents (0-100)		
0x40	0x01	0x00	0x41	0x0D	0x79 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0xfb 0x00 0x00 0x00 0x64		

We will have to send two bytes, 0x02 and 0x00 to the UART interface of the DWM and parse an 18byte response from the DWM.

7.2.1.1 Component Architecture

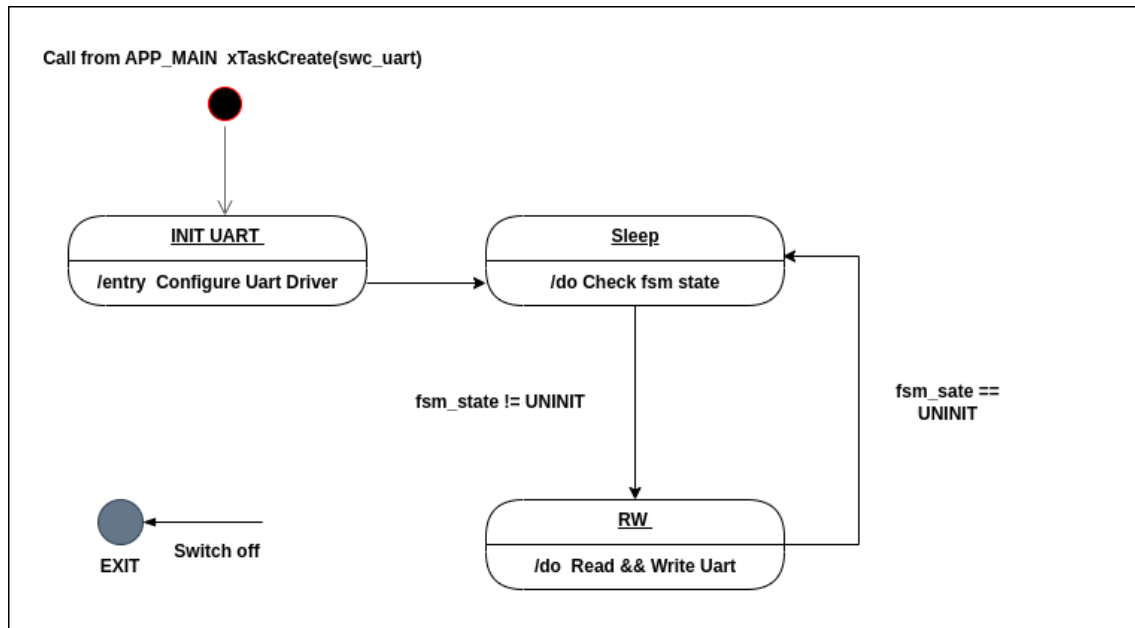


Figure 35: SWC_UART Workflow

To implement this functionality, we will use the esp-idf driver/uart.h API, FreeRTOS libraries FreeRTOS.h and task.h.

The driver/uart.h is a series of functions in the ESP-IDF that allow the configuration and use of the UART peripheral, transmit and receive data over UART serial connection, set baud rates and other communication parameters. [48]

FreeRTOS is a real-time operating system (RTOS) for ESP-IDF, provides a set of functions and tools for creating multi-threaded applications on the ESP32, allowing users to prioritize and schedule tasks. The freertos/freertos.h and freertos/task.h provide the basic functionalities of the RTOS. [49]

The flow of the component is implemented using Tasks and delays from FreeRTOS.

At entry the UART driver is initiated. Then each 15 ms we check for the state of the system. If the state is different from UNINIT we will call the function read_write() which reads and writes to the UART, parses the result and updates the global variable that holds the position.

```
void vTask_uart(void *pvParameters)
{
    uint8_t local_fsm = fsm_state;

    uart_init();

    while(1)
    {
        local_fsm = fsm_state;

        if(local_fsm != FSM_UNINIT){
            read_write();
        }
        else{
        }

        vTaskDelay(15000/ portTICK_PERIOD_MS);
    }

    vTaskDelete(NULL);
}
```

Figure 36: FreeRtoos Uart Task

The `uart_init()` function, configures the UART driver, sets buffers and queues for the received data and initiates de driver.

```
void uart_init(void)
{
    data = (uint8_t*) malloc(BUF_SIZE);           //Allocate memory for data incoming from DEV.
    pos_raw = (uint8_t*) calloc(POS_SIZE,sizeof(uint8_t)); //Definition of global
    | | | | | | | | | | | | | | | | | | | | | | //variable pos_raw. With 18 bytes of the position data

    uart_config_t uart_config = { //Structure with uart driver parameters
        .baud_rate = BAUD_RATE,      //115200 bauds
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE,
    };

    // Set UART log level
    esp_log_level_set(TAG_UART, ESP_LOG_INFO);

    ESP_LOGI(TAG_UART, "Start UART configuration.");

    // Install UART driver using an event queue here,
    //which will create an event when data is read and a buffer to store received data.
    ESP_ERROR_CHECK(uart_driver_install(UART_NUM, BUF_SIZE, 0, 10, &uart_queue, 0));
    vTaskDelay(20);

    // Configure UART parameters
    ESP_ERROR_CHECK(uart_param_config(UART_NUM, &uart_config));

    ESP_LOGI(TAG_UART, "UART set pins, mode and install driver.");

    // Set UART pins as per Config settings
    ESP_ERROR_CHECK(uart_set_pin(UART_NUM, 17, 16, UART_PIN_NO_CHANGE,UART_PIN_NO_CHANGE ));
}
```

Figure 37,38: *uart_init()* function

The read_write() function, triggered every 15ms, writes to UART the request position payload and waits for an event in the UART driver. If data is received it checks if it is the expected payload and then copies it to pos_raw global variable.

```

void read_write(void)
{
    uart_event_t event;
    uint8_t pos_req[2] = {0x02,0x00}; // request position payload

    // Send request position payload to uart
    if (uart_write_bytes(UART_NUM, (const void *) &pos_req, sizeof(pos_req)) != sizeof(pos_req)) {
        ESP_LOGE(TAG_UART, "Data not sent");
    }
    else if(xQueueReceive( uart_queue, (void *) &event, portMAX_DELAY)){ // Wait for data from uart
        bzero(data, BUF_SIZE); //Clear data buffer
        switch(event.type)
        {
            case UART_DATA: // Data received

                ESP_LOGI(TAG_UART,"event size = %d \n", (int) event.size );
                uart_read_bytes(UART_NUM, data ,event.size, portMAX_DELAY); // Read data received

                // Check if response equals to expected response payload.
                if((int) event.size == 18 && (uint8_t) data[2] == 0 && (uint8_t) data[0] == 64 && (uint8_t) data[4] == 13 )
                {
                    memcpy(pos_raw,data,event.size); // copy data to global variable pos_raw
                }
                default:
                    break;
        }
    }
    else{
        ESP_LOGI(TAG_UART,"No data received \n");
    }
}

```

Figure 39,40: read_write() function

7.2.2 Interface component

The interface component is programmed in the swc_button component of the code. The button is connected to an ESP32 pin and GND. When pressed an interrupt in ESP32 occurs, the state of the system is changed and the communication with the server starts.

To program this component, we will use the GPIO API from ESP-IDF and its libraries “driver/gpio.h” and “esp_intr_alloc.h”. The “driver/gpio.h” allows you to configure the direction and state of the GPIO pins, as well as read and write data to them and the “esp_intr_alloc.h” allows us to allocate an interrupt handler that will change the state value. [50]

7.2.2.1 Component Architecture

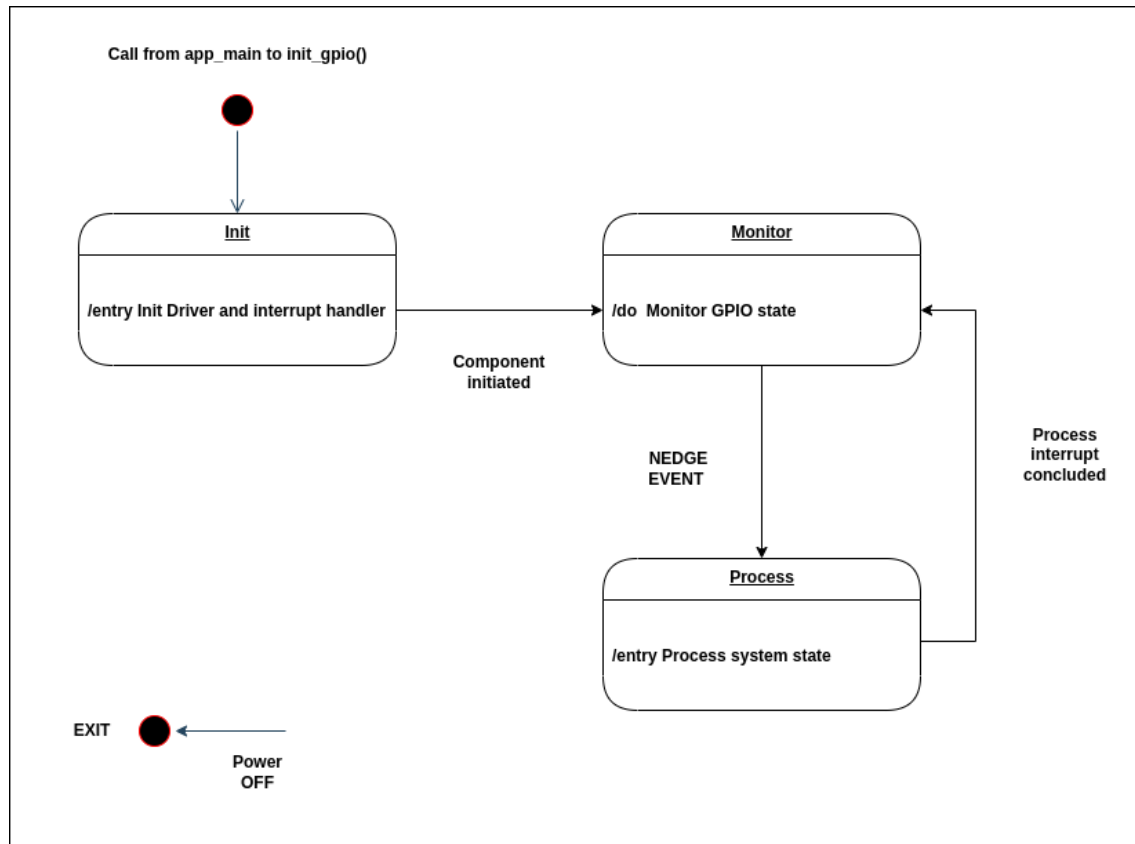


Figure 41: SWC_BUTTON Workflow

The function `init_gpio()` initializes the component. The button is configured as internal pull-up resistor. The resistor is used to avoid a shortage between the V_{dd} and GND when the switch is on, which will cause overheating and may end up damaging the electronics.

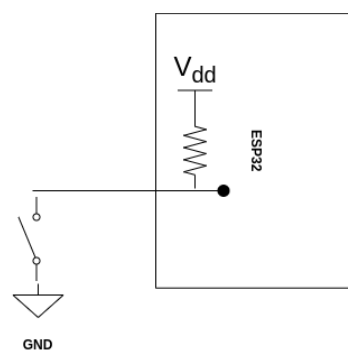


Figure 42: Internal Pull-up resistor configuration

The default state of the GPIO pin is HIGH. When the switch is on it falls to LOW, the interrupt should be triggered on an edge fall.

```

//init gpio
void init_gpio (void)
{
    //Configuration of the pin
    gpio_config_t io_conf = {

        //bit mask of the pins, use GPIO5 here
        .pin_bit_mask = 1ULL<< 5,
        //Sets input mode
        .mode = GPIO_MODE_INPUT,
        //Enables pullup mode
        .pull_up_en = GPIO_PULLUP_ENABLE,
        //Disable pulldown mode
        .pull_down_en = GPIO_PULLDOWN_DISABLE,
        //Sets interrupt when edge goes from high to low
        .intr_type = GPIO_INTR_NEGEDGE,
    };

    //configure gpio driver with the given settings
    ESP_ERROR_CHECK(gpio_config(&io_conf));

    //install gpio isr service
    ESP_ERROR_CHECK(gpio_install_isr_service(ESP_INTR_FLAG_DEFAULT));

    //hook isr handler for specific gpio pin
    ESP_ERROR_CHECK(gpio_isr_handler_add(GPIO_NUM_5, gpio_isr_handler, NULL));
}

```

Figure 43: *init_gpio()* function

Once the interrupt service has been installed, the component monitors the state of the GPIO continuously and raises and interrupts when it detects a negative edge. The interrupt then calls back the handler function `gpio_isr_handler()`.

The handler function checks the state of the system, if the state is `FSM_INIT` it transitions the state to `FSM_COMM`.

```

static void IRAM_ATTR gpio_isr_handler(void* arg)
{
    if(fsm_state == FSM_INIT){
        fsm_state = FSM_COMM;
    }
}

```

Figure 44: *gpio_isr_handler()* function

7.2.3 Wi-Fi Component

Wi-Fi provides the physical medium for our data to be linked between devices. It transmits and receive data over a wireless network and manages how devices interact with it. In the OSI model its layers are physical layer and data-link layer.

In the Wi-Fi protocol there are defined different types of nodes. For this project Access Points (AP) and Stations (STA) are relevant. An AP connects wireless devices to the ethernet network providing internet access and provides devices on the Wi-Fi network the ability to communicate. The STA is a device connected to the Wi-Fi through an AP device.

Our device will be configured as a STA. The ESP32 has Wi-Fi capabilities and the ESP-IDF library “esp_wifi.h” provides support for configuring and monitoring ESP32 Wi-Fi network. [51]

The swc_wifi configures and initialises the Wi-Fi Station. For this purpose, we have modified the ESP-IDF example code on how to set-up a Wi-Fi STA. [52]

7.2.3.1 Component Architecture

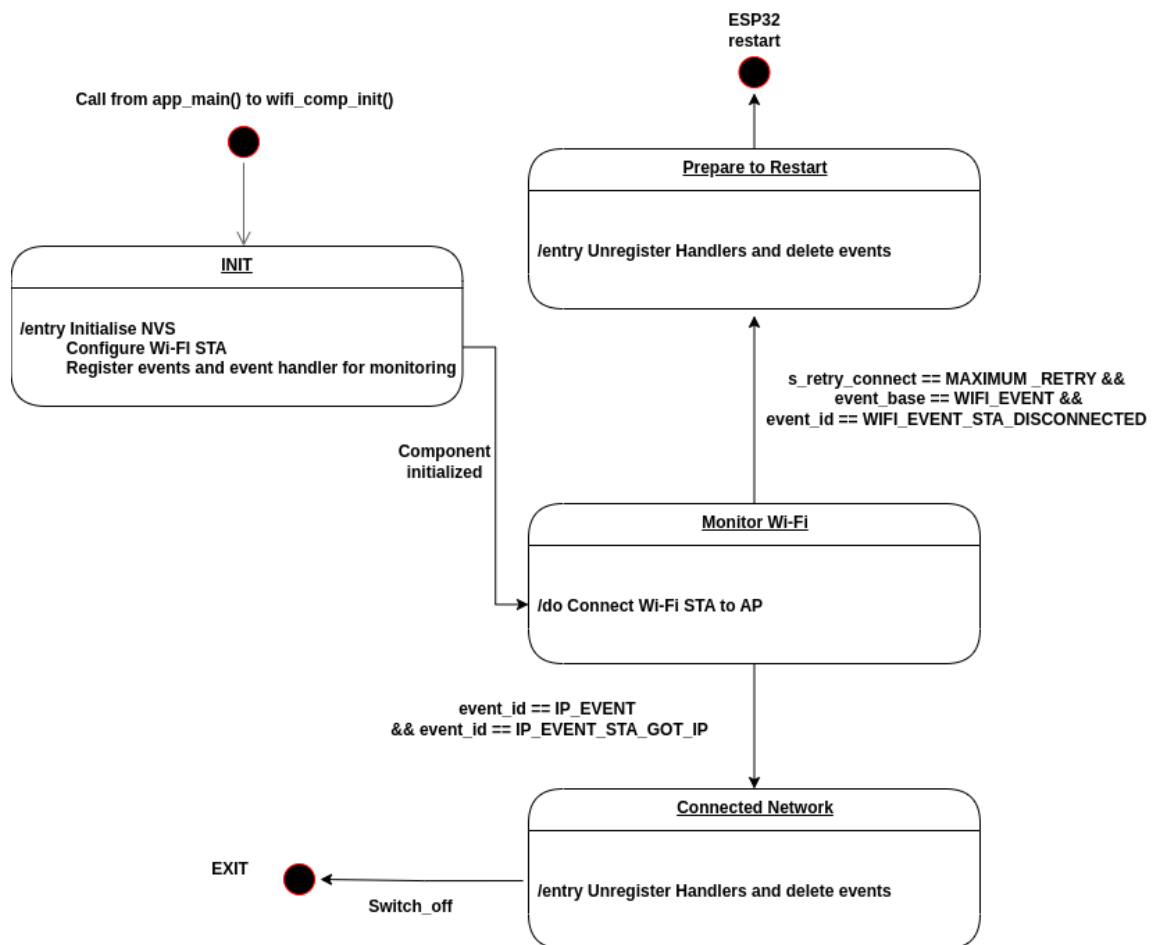


Figure 45: SWC_WIFI Workflow

On start of the system the `wifi_comp_init()` is called from the `app_main()` entry point.

The function `wifi_comp_init()`, initiates the Non-volatile Storage (NVS), used for storing configuration data and calls the function that configures the Wi-Fi as a Station, `wifi_init_sta()`.

```
void wifi_comp_init(){  
    //Initialize Non-Volatile Storage to store key-value pairs. Needed for some func like esp_wifi_init to store/read info.  
  
    esp_err_t ret = nvs_flash_init();  
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret == ESP_ERR_NVS_NEW_VERSION_FOUND) {  
        ESP_ERROR_CHECK(nvs_flash_erase());  
        ret = nvs_flash_init();  
    }  
    ESP_ERROR_CHECK(ret);  
  
    ESP_LOGI(TAG, "ESP_WIFI_MODE_STA");  
    wifi_init_sta(); //Function that configures  
}
```

Figure 46: *wifi_comp_init()* function

The `wifi_init_sta()` initializes the Wi-Fi STA, the TCP/IP stack use for Network and Transport protocols which would route the incoming packets from the Wi-Fi or from upper layers and creates an event handler and a group wait bits, to monitor the Wi-Fi start-up.

```
void wifi_init_sta(void)
{
    //Creates wait bit event group for WiFi events.
    s_wifi_event_group = xEventGroupCreate();

    // Initialize the underlying TCP/IP stack
    ESP_ERROR_CHECK(esp_netif_init());

    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));

    esp_event_handler_instance_t instance_any_id;
    esp_event_handler_instance_t instance_got_ip;

    //Handler to parse WIFI events.
    ESP_ERROR_CHECK(esp_event_handler_instance_register(WIFI_EVENT,
                                                         ESP_EVENT_ANY_ID,
                                                         &event_handler,
                                                         NULL,
                                                         &instance_any_id));

    //Handler to parse Internet Protocol events.
    ESP_ERROR_CHECK(esp_event_handler_instance_register(IP_EVENT,
                                                         IP_EVENT_STA_GOT_IP,
                                                         &event_handler,
                                                         NULL,
                                                         &instance_got_ip));

    wifi_config_t wifi_config = { //Configures the WiFi station.
        .sta = {
            .ssid = WIFI_SSID,
            .password = WIFI_PASS,
            /* Setting a password implies station will connect to all security
            modes including WEP/WPA.* However these modes are deprecated
            and not advisable to be used. Incase your Access point* doesn't support WPA2,
            these mode can be enabled by commenting below line */
            .threshold.authmode = WIFI_AUTH_WPA2_PSK,
        },
    };

    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA) );
    ESP_ERROR_CHECK(esp_wifi_set_config(WIFI_IF_STA, &wifi_config) );
    ESP_ERROR_CHECK(esp_wifi_start() );
}
```

Figure 47: wifi_init_sta() 1

The function will wait until the group wait bits has been cleared, if the WIFI_CONNECTED_BIT or the WIFI_FAIL_BIT is set to 1, this means that the connection has been successful, or the STA could not connect to the AP. We will unregister the event handler and event for both scenarios and restart the ESP32 if it fails to connect to the AP.

```

ESP_LOGI(TAG, "wifi_init_sta finished.");

/* Waiting until either the connection is established (WIFI_CONNECTED_BIT)
or connection failed for the maximum* number of re-tries (WIFI_FAIL_BIT).
The bits are set by event_handler() (see above) */
EventBits_t bits = xEventGroupWaitBits(s_wifi_event_group,
    WIFI_CONNECTED_BIT | WIFI_FAIL_BIT,
    pdFALSE,
    pdFALSE,
    portMAX_DELAY);

/* xEventGroupWaitBits() returns the bits before the call returned,
hence we can test which event actually happened. */
if (bits & WIFI_CONNECTED_BIT) {
    ESP_LOGI(TAG, "connected to ap SSID:%s",
        WIFI_SSID);
} else if (bits & WIFI_FAIL_BIT) {
    ESP_LOGI(TAG, "Failed to connect to SSID:%s , Restarting ESP32",
        WIFI_SSID);
    esp_restart(); // The ESP32 is restarted to avoid unexpected behavior.
} else {
    ESP_LOGE(TAG, "UNEXPECTED EVENT");
}

/* The event will not be processed after unregister */
ESP_ERROR_CHECK(esp_event_handler_instance_unregister(IP_EVENT, IP_EVENT_STA_GOT_IP, instance_got_ip));
ESP_ERROR_CHECK(esp_event_handler_instance_unregister(WIFI_EVENT, ESP_EVENT_ANY_ID, instance_any_id));
vEventGroupDelete(s_wifi_event_group);
}

```

Figure 48: *wifi_init_sta()* 2

The registered event_handler monitors the WIFI and IP events, if it fails to connect to the AP, it will retry it until the maximum allowed tries is reached and set the WIFI_FAIL_BIT. If the IP event received is IP_EVENT_STA_GOT_IP means that the AP assigned us an IP, so we are connected to the network, and sets the WIFI_CONNECTED_BIT.

```

static void event_handler(void* arg, esp_event_base_t event_base,
    int32_t event_id, void* event_data)
{
    // Checks if the station has started.
    if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_START) {
        esp_wifi_connect();

        // Checks if the station is disconnected from AP.
    } else if (event_base == WIFI_EVENT && event_id == WIFI_EVENT_STA_DISCONNECTED) {
        if (s_retry_connect_num < MAXIMUM_RETRY) {
            esp_wifi_connect();
            s_retry_connect_num++;
            ESP_LOGI(TAG, "retry to connect to the AP");
        } else {
            xEventGroupSetBits(s_wifi_event_group, WIFI_FAIL_BIT);
        }
        ESP_LOGI(TAG, "connect to the AP fail");

        // Checks if an IP has been assigned to STA by AP.
    } else if (event_base == IP_EVENT && event_id == IP_EVENT_STA_GOT_IP) {
        ip_event_got_ip_t* event = (ip_event_got_ip_t*) event_data;
        ESP_LOGI(TAG, "got ip:" IPSTR, IP2STR(&event->ip_info.ip));
        s_retry_connect_num = 0;
        xEventGroupSetBits(s_wifi_event_group, WIFI_CONNECTED_BIT);
    }
}

```

Figure 49: *event_handler()* function

7.2.4 Audio pipeline

We have two audio pipelines. The playback pipeline and the recording pipeline. The main Software components involved are the `swc_i2s` and the `swc_udp`.

The `swc_i2s` initializes and configures both input and output i2s driver and provide functions to write and read data from the i2s ports. To implement these functions, the ESP-IDF library "driver/i2s.h" is included.

The `swc_udp` creates a new socket, the socket provides an interface for our application to access the transport layer functionalities. It creates packets to be sent to the server and handles the incoming packets from the server and populates the data towards other components. The TCP/IP stack libraries "`lwip/err.h`", "`lwip/sockets.h`", "`lwip/sys.h`" and "`lwip/netdb.h`" are used to manage the transport layer. [53]

7.2.4.1 Recording pipeline

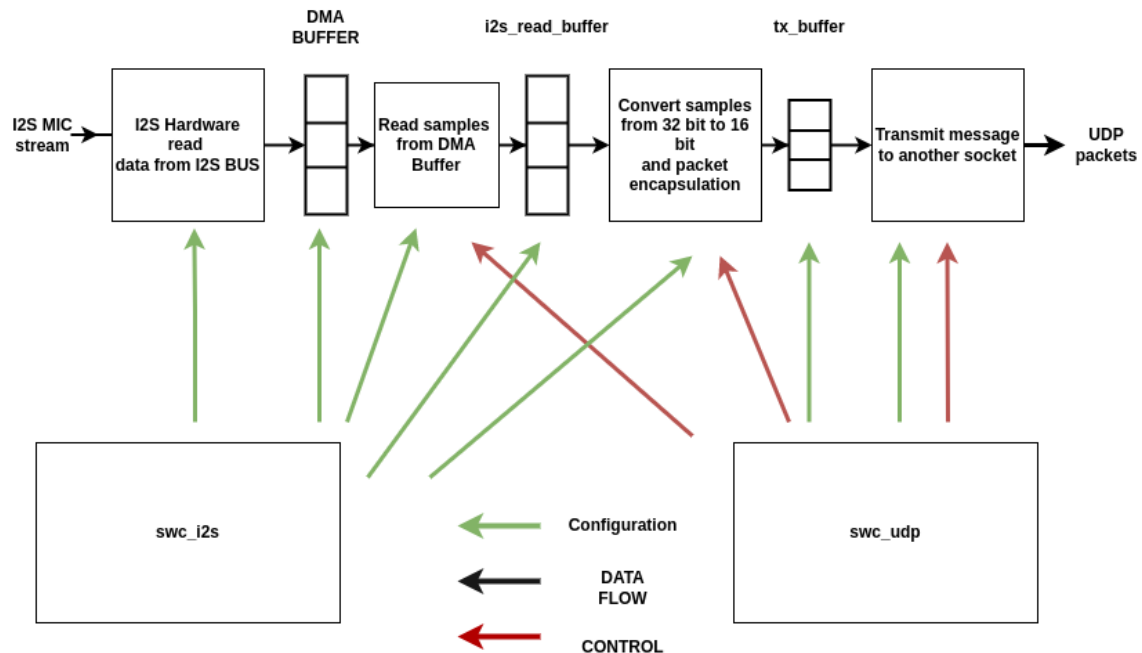


Figure 50: Recording pipeline

The recording pipeline starts from the I2S stream coming from the MIC. Data is read by the I2S RX PORT which is configured by the `swc_i2s`. The function `i2s_rx_init()` which is called on entry of the system configures the i2s driver. The component uses the library "driver/i2s.h" and "soc/i2s_reg.h" to access the peripheral and configure it.

```

void i2s_rx_init(){
    static const i2s_config_t i2s_configuration_rx = {
        // The ESP32 is the MASTER. The BCLOCK and WS are generated by the ESP32
        .mode = I2S_MODE_MASTER | I2S_MODE_RX,
        .sample_rate = I2S_SAMPLE_RATE, // 16Khz
        .bits_per_sample = I2S_BITS_SAMPLE, // 32 bits
        .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT, // L/R is grounded, only left. MONO
        .communication_format = I2S_COMM_FORMAT_STAND_I2S,
        // The default data format is I2S (two's complement), MSB-first.
        // In this format, the MSB of each word is delayed by one SCK cycle from the start of each half-frame.
        //The SPH MIC does not use this format. The MSB is delayed half SCK cycle
        .tx_desc_auto_clear = false,
        .dma_buf_count = 4, //buffers not free until DMA has written and also cpu has read them once dma wrote
        .dma_buf_len = 1024, // Max length available so if server takes a lot to answer we accumulate data.
        .use_apll = false,
        .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1 // Interrupt level 1, default 0
    };

    static const i2s_pin_config_t pin_configuration_rx = {
        .mck_io_num = I2S_PIN_NO_CHANGE,
        .bck_io_num = I2S_BCLK_RX,
        .ws_io_num = I2S_WS_RX,
        .data_out_num = I2S_PIN_NO_CHANGE,
        .data_in_num = I2S_DATA_IN
    };

    if (ESP_OK != i2s_driver_install(I2S_RX_PORT, &i2s_configuration_rx, 0, NULL)) {
        printf("ERROR : i2s_driver_install");
    }

    //FIX for MIC. MSB bit comes after one bclock after WS not one bit afert WS.
    REG_SET_BIT(I2S_TIMING_REG(I2S_RX_PORT), BIT(9));
    REG_SET_BIT(I2S_CONF_REG(I2S_RX_PORT), I2S_RX_MSB_SHIFT);

    if (ESP_OK != i2s_set_pin(I2S_RX_PORT, &pin_configuration_rx)) {
        printf("ERROR : i2s_set_pin");
    }

    i2s_zero_dma_buffer(I2S_RX_PORT);
}

```

Figure 51: *i2s_rx_init()* function

The main parameters of the I2S bus are set. The sample rate is set by the master (ESP32) it will share its BCLOCK with the MIC which will use it to get the correct sample rate. We have set 16Khz as the sample rate as it the standard format for Wideband Audio used in VoIP, which provides good quality audio for voice applications. [54]

Channel format is only left as we only have one MIC and the communication I2S_COMM_FORMAT_STAND_I2S but is later modified to set that MSB bit comes after half BCLOCK not one with the REG_SET_BIT () calls.

```

//FIX for MIC. MSB bit comes after one bclock after WS not one bit afert WS.
REG_SET_BIT(I2S_TIMING_REG(I2S_RX_PORT), BIT(9));
REG_SET_BIT(I2S_CONF_REG(I2S_RX_PORT), I2S_RX_MSB_SHIFT);

```

Figure 52: MSB adjustment for MIC

The pins for the i2s peripheral are also set.

The DMA buffers are also set. Up to 4 buffers of 1024 samples are set to store data. DMA buffers is accessed by the peripheral directly and stores data independently of the CPU, freeing resources. 1024 sample are chosen to interrupt the CPU the less.

The DMA buffer is accessed by a call from the `udp_client_task` from the `swc_udp` when we are in the COMM state. The component calls the `i2s_read()` function from the "driver/i2s.h". Passes a pointer to the `i2s_read_buffer` where it will store the data on the DMA, the size of the data it wants to read, it is set to 255 samples after that it collects garbage, a pointer to a `size_t` param to store the number of bytes read and the ticks to wait. The function will wait until the information is received.

```

case FSM_COMM:

    counter_send_pos +=1;

    if (counter_send_pos < POS_TIMEOUT)
    {
        // 255 Chunk max for 16k before collecting garbage
        i2s_read(I2S_RX_PORT,(void *)i2s_read_buff,
        |      CHUNK_SEND*sizeof(int32_t), &bytes_read, portMAX_DELAY);

        bytes_write = sample_32bit_to_16bit(i2s_read_buff,bytes_read,
        |      |      |      |      |      |      |      |      tx_buffer,SAMPLES_TO_SEND,0);
    }
    else{

        counter_send_pos = 0;
        bytes_write = packet_pos(tx_buffer,local_pos,1);
        i2s_read(I2S_RX_PORT,(void *)i2s_read_buff,
        |      CHUNK_SEND*sizeof(int32_t), &aux_byte_write, portMAX_DELAY);
        bytes_write += sample_32bit_to_16bit(i2s_read_buff,aux_byte_write,
        |      |      |      |      |      |      |      |      tx_buffer,SAMPLES_TO_SEND,1);
    }

break;

```

Figure 53: DMA read call

Once the information is received, we call the `sample_32bit_to_16bit` function from the `swc_i2s` component. This function will resample the data, as the data received from the mic is 18-bit width as the 18-24 bits are set to 0 and 24 to 32 are tri-stated, to use less bandwidth. The data will be copied to the `tx_buffer` and a header that identifies audio packets from our application (0x01,0x03) will be added.

```

size_t sample_32bit_to_16bit(int32_t* i2s_read_buff,
                             size_t bytes_read,
                             char * buffer, size_t max_size,
                             uint8_t type)
{
    uint16_t sample_size = bytes_read/4;
    uint16_t aux_i = 0;
    if(sample_size > 1024)
    {
        sample_size = 1024;
    }

    bytes_read = bytes_read/2 + 2;
    if( type == 0){
        buffer[0] = 0x01;
        buffer[1] = 0x03;
        aux_i = 2;
    }
    else if (type == 1){
        buffer[21] = 0x01;
        buffer[22] = 0x03;
        aux_i = 23;
    }
    else
    {
    }

    for(int i = 0; i < sample_size; i++)
    {
        i2s_read_buff[i] = (i2s_read_buff[i] & 0xFFFFFFFF)>>16;

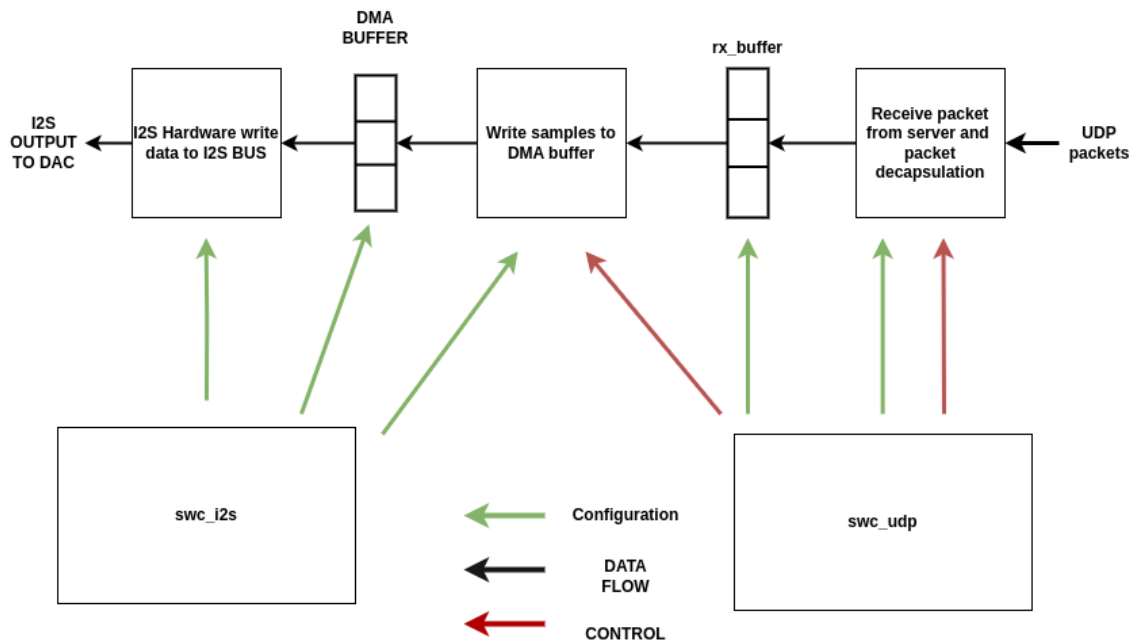
        buffer[i*2 + aux_i] = (i2s_read_buff[i] & 0xFF);
        buffer[i*2 + 1 + aux_i] = (i2s_read_buff[i] & 0xFFFF)>>8;
    }
    return bytes_read;
}

```

Figure 54,55: sample_32bit_to_16bit()

After copying the data to the tx_buffer the udp_client_task will send the data to socket using the sendto() function. Which will encapsulate the data in a udp packet and send to the set address and port in the function.

7.2.4.2 Playback pipeline



As in the recording pipeline, we set the configuration of the I2S driver and pins from the `swc_i2s`. On entry the function `i2s_tx_init()` is called. We will configure the sample rate as 16Khz, 16 bits per sample, only one channel left and `I2S_COMM_FORMAT_STAND_I2S` as the audio received from the server is in this format. The DMA buffer length is reduced to decrease the latency.

```
void i2s_tx_init(){
    static const i2s_config_t i2s_configuration_tx = {
        .mode = I2S_MODE_MASTER | I2S_MODE_TX,
        .sample_rate = I2S_SAMPLE_RATE_TX, // 16Khz
        .bits_per_sample = I2S_BITS_SAMPLE_TX, // 16 bits
        .channel_format = I2S_CHANNEL_FMT_ONLY_LEFT, // L/R is grounded, only left
        .communication_format = I2S_COMM_FORMAT_STAND_I2S,
        // The information received is in this format
        .dma_buf_count = 4, //buffers not free until DMA has written
        //and also cpu has read them once dma wrote
        .dma_buf_len = 256, // Max length available so if server
        //takes a lot to answer we accumulate data.
        .use_apll = false,
        .tx_desc_auto_clear = true, // See if tx descriptor needs to be auto cleared:
        // This will avoid any kind of noise that may
        //get introduced due to transmission
        // of previous data from tx descriptor on I2S line.
        .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1, // Interrupt level 1, default 0
        .fixed_mclk = 0
    };
};
```

```

static const i2s_pin_config_t pin_configuration_tx = {
    .mck_io_num = I2S_PIN_NO_CHANGE,
    .bck_io_num = I2S_BCLK_TX,
    .ws_io_num = I2S_WS_TX,
    .data_out_num = I2S_DATA_OUT,
    .data_in_num = I2S_PIN_NO_CHANGE
};

if (ESP_OK != i2s_driver_install(I2S_TX_PORT, &i2s_configuration_tx, 0, NULL)) {
    printf("ERROR : i2s_driver_install");
}
if (ESP_OK != i2s_set_pin(I2S_TX_PORT, &pin_configuration_tx)) {
    printf("ERROR : i2s_set_pin");
}

i2s_zero_dma_buffer(I2S_TX_PORT);
}

```

Figure 56,57: i2s_tx_init()

The `recvfrom()` function called in the `udp_client_task` from the `swc_udp` waits for a packet from the server before the timeout expires. When received, the function populates the `rx_buffer` with the message contained in the udp packet after decapsulating it.

```

int len = recvfrom(sock, rx_buffer, sizeof(rx_buffer) - 1, 0,
    (struct sockaddr *)&source_addr, &socklen);

```

Figure 58: `recvfrom()` function

If in `FSM_COMM` state and our packet header is (0x01,0x03), the data received is written to the DMA calling the function `i2s_write()` passing the `rx_buffer`.

```

else if(local_fsm == FSM_COMM && rx_buffer[0] == 0x01
    && rx_buffer[1] == 0x03)
{
    size_t bytes_to_write = 0;
    //Copy data to DMA Bufer
    i2s_write(I2S_TX_PORT, rx_buffer + 2, len-2, (size_t*) &bytes_to_write, 20);
}

```

Figure 59: Write audio to DMA

The I2S driver will read from the DMA and send it through the I2S bus.

7.2.5 UDP component

This component allows the application to communicate with the server. It is where the application interacts with the transport layer used to transport the packets to the destination. UDP known as Unified Datagram Protocol has been chosen and not TCP known as Transmission Control Protocol because is connectionless and does not require both end points to synchronise, it does not perform heavy checks on received data and doesn't reorder packets in a chronological order like TCP. The UDP header is also smaller than TCPs. For those reasons UDP is lighter and faster than TCP and the preferred transport protocol for streaming. [55][56]

7.2.5.1 Component architecture

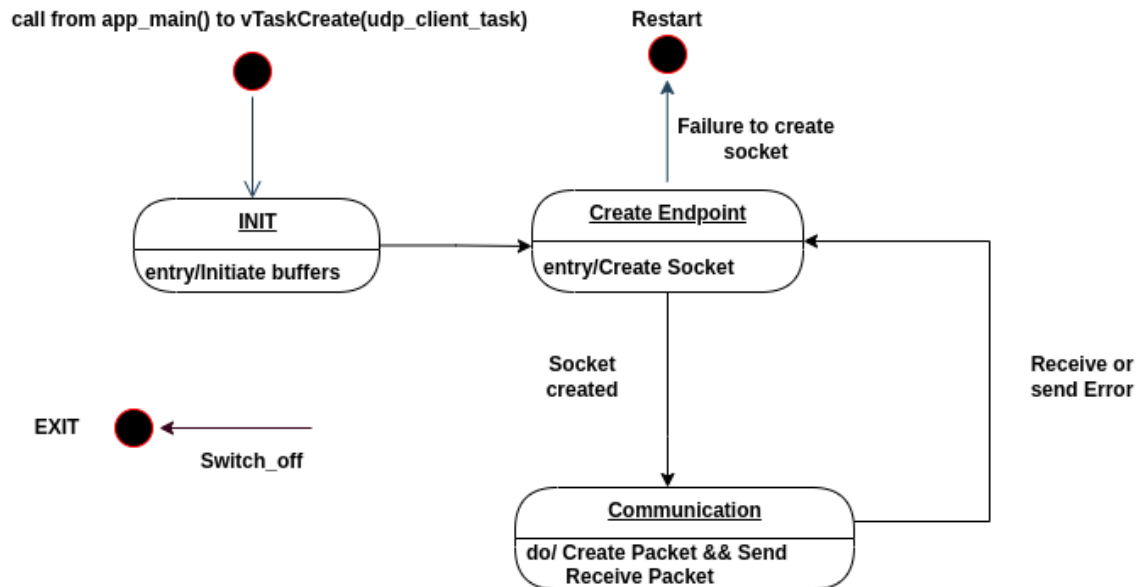


Figure 60: SWC_UDP workflow

On init buffers for receive and transmit payloads are allocated and other needed variables are defined.

```

void udp_client_task(void *pvParameters)
{
    char tx_buffer[1024];
    char rx_buffer[1032];
    char host_ip[] = HOST_IP_ADDR;
    int addr_family = 0;
    //Memory to allocate position of DWM
    uint8_t* local_pos = malloc(sizeof(uint8_t)*POS_SIZE);
  
```

Figure 61: udp_client_task() function

Then the Socket is created, sockets provide end to end communication between two endpoints. The Socket represents an endpoint. We set the socket domain to IPv4, the SOCK_DGRAM which creates an UDP socket and the protocol to UDP. We also set a timeout on the receiving packets of 50 MS. The timeout is set to avoid blocking the process and packet loss. It has been adjusted monitoring Wi-Fi Network with Wireshark.

If we fail to create the socket, a restart is called to avoid unexpected behaviour.

```

struct sockaddr_in dest_addr;
dest_addr.sin_addr.s_addr = inet_addr(HOST_IP_ADDR); //Set Host IP
dest_addr.sin_family = AF_INET; // SET IPV4
dest_addr.sin_port = htons(PORT); //Set Port
addr_family = AF_INET; // SET IPV4

//Create socket with IPv4 address format, UDP type socket and protocol UDP.
int sock = socket(addr_family, SOCK_DGRAM, IPPROTO_UDP);
if (sock < 0) {
    ESP_LOGE(TAG, "Unable to create socket: errno %d", errno);
    esp_restart(); // The ESP32 is restarted to avoid unexpected behavior.
}
// Set timeout. As udp is connectionless, we need to set timeout for recvfrom.
struct timeval timeout;
timeout.tv_sec = 0;
timeout.tv_usec = 50000;
setsockopt (sock, SOL_SOCKET, SO_RCVTIMEO, &timeout, sizeof timeout);

ESP_LOGI(TAG, "Socket created, sending to %s:%d", HOST_IP_ADDR, PORT);

```

Figure 62: UDP Socket Creation

If the endpoint is created successfully, the task starts managing communications. It performs operations depending on the state of the system.

The following diagram explains how the communications work once the socket has been created.

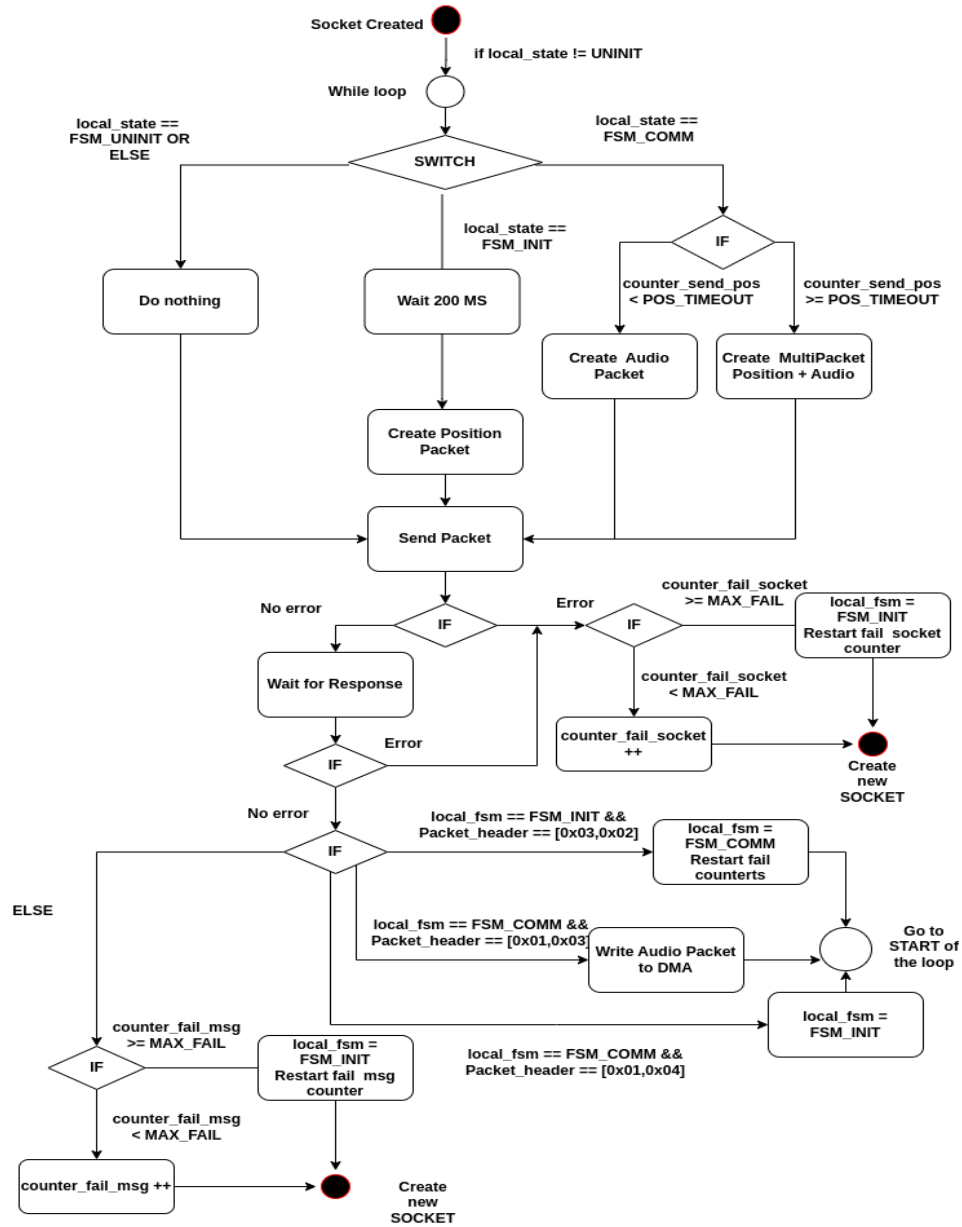


Figure 63: Communication Workflow

The code that implements that logic is as it follows:

Here packets payloads are built depending on the system state.

```
switch(local_fsm){
    case FSM_UNINIT:
        //NO COMMUNICATION WITH SERVER
        break;
    case FSM_INIT:
        vTaskDelay(200/portTICK_PERIOD_MS);
        // BUILD THE PACKET
        bytes_write = packet_pos(tx_buffer,local_pos,0);
        break;
    case FSM_COMM:
        counter_send_pos +=1;
        if (counter_send_pos < POS_TIMEOUT)
        {
            // 255 Chunk max for 16k before collecting garbage
            i2s_read(I2S_RX_PORT,(void *)i2s_read_buff,
                CHUNK_SEND*sizeof(int32_t), &bytes_read, portMAX_DELAY);
            bytes_write = sample_32bit_to_16bit(i2s_read_buff,bytes_read,
                tx_buffer,SAMPLES_TO_SEND,0);
        }
        else{
            counter_send_pos = 0;
            bytes_write = packet_pos(tx_buffer,local_pos,1);
            i2s_read(I2S_RX_PORT,(void *)i2s_read_buff,
                CHUNK_SEND*sizeof(int32_t), &aux_byte_write, portMAX_DELAY);
            bytes_write += sample_32bit_to_16bit(i2s_read_buff,aux_byte_write,
                tx_buffer,SAMPLES_TO_SEND,1);
        }
        break;
    default:
        break;
}
```

Figure 64: Implementation of packet payload building

Once we have our payload, we use the function `sendto()` to send the data on the socket with UDP encapsulation. After that we will do error check.

```
int err = sendto(sock,(void *) tx_buffer, bytes_write, 0,
    (struct sockaddr *)&dest_addr, sizeof(dest_addr));
if (err < 0) {
    ESP_LOGE(TAG, "Error occurred during sending: errno %d", errno);
    if(counter_fail_socket >= MAX_FAIL){
        fsm_state = FSM_INIT;
        counter_fail_socket = 0;
    }
    counter_fail_socket += 1;
    break;
}
//ESP_LOGI(TAG, "Message sent");
```

Figure 65: Packet Sending and Error Check

If it has been successful, we will wait for a response until the rx timeout expires. On response we will do error check.

```
int len = recvfrom(sock, rx_buffer, sizeof(rx_buffer) - 1, 0,
    (struct sockaddr *)&source_addr, &socklen);

// Error occurred during receiving
if (len < 0) {
    //ESP_LOGE(TAG, "recvfrom failed: errno %d", errno);

    if(counter_fail_socket >= MAX_FAIL){
        local_fsm = FSM_INIT;
        counter_fail_socket = 0;
    }

    counter_fail_socket += 1;

    break;
}
```

Figure 66: Packet Reception and Error Check

If no error, we will parse the data. If it is an audio packet, we will send it to the DMA or if it is the server requesting us to start communicating, we will change the state to FSM_COMM. If the packet is not recognized, we will increase the error counter.

```
else {
    //ESP_LOGI(TAG, "Received %d bytes from %s:", len, host_ip); only for debug
    // first byte 0x03-> request to change state
    if (local_fsm == FSM_INIT && rx_buffer[0] == 0x03 && rx_buffer[1] == 0x02)
    {
        local_fsm = FSM_COMM;
        counter_fail_socket = 0;
        counter_fail_msg = 0;
    }
    else if(local_fsm == FSM_COMM && rx_buffer[0] == 0x01 && rx_buffer[1] == 0x03)
    {
        size_t bytes_to_write = 0;
        //Copy data to DMA Bufer
        i2s_write(I2S_TX_PORT, rx_buffer + 2, len-2, (size_t*) &bytes_to_write, 20);
    }
    else if(local_fsm == FSM_COMM && rx_buffer[0] == 0x01 && rx_buffer[1] == 0x04)
    {
        local_fsm = FSM_INIT; // Change state to init after stop comm from server
    }
    else
    {
        if(counter_fail_msg >= MAX_FAIL)
        {
            local_fsm = FSM_INIT;
            counter_fail_msg = 0;
        }
        counter_fail_msg += 1;
    }
}

fsm_state = local_fsm;
```

Figure 67: Treatment of Received Payloads

7.3 Server

The server must handle incoming communications and route them to correct users. It also should support a graphical user interface for the staff to interact with all the patient devices in the network.

In short it should receive audio packets and position packets from patient devices and audio from the staff access point to the network and route them.

For this project a basic server has been written in python. It creates a UDP server using sockets and the PyAudio library creates an input and output stream of audio, to playback or record audio from the server. It also stores the position sent by the patient device. [\[57\]](#)

The server is not capable of receiving input from user, is quite slow and might struggle to handle multiple connections, as it is written in python. Creating a fully functional server would require a TFG of its own.

7.3.1 Server Architecture

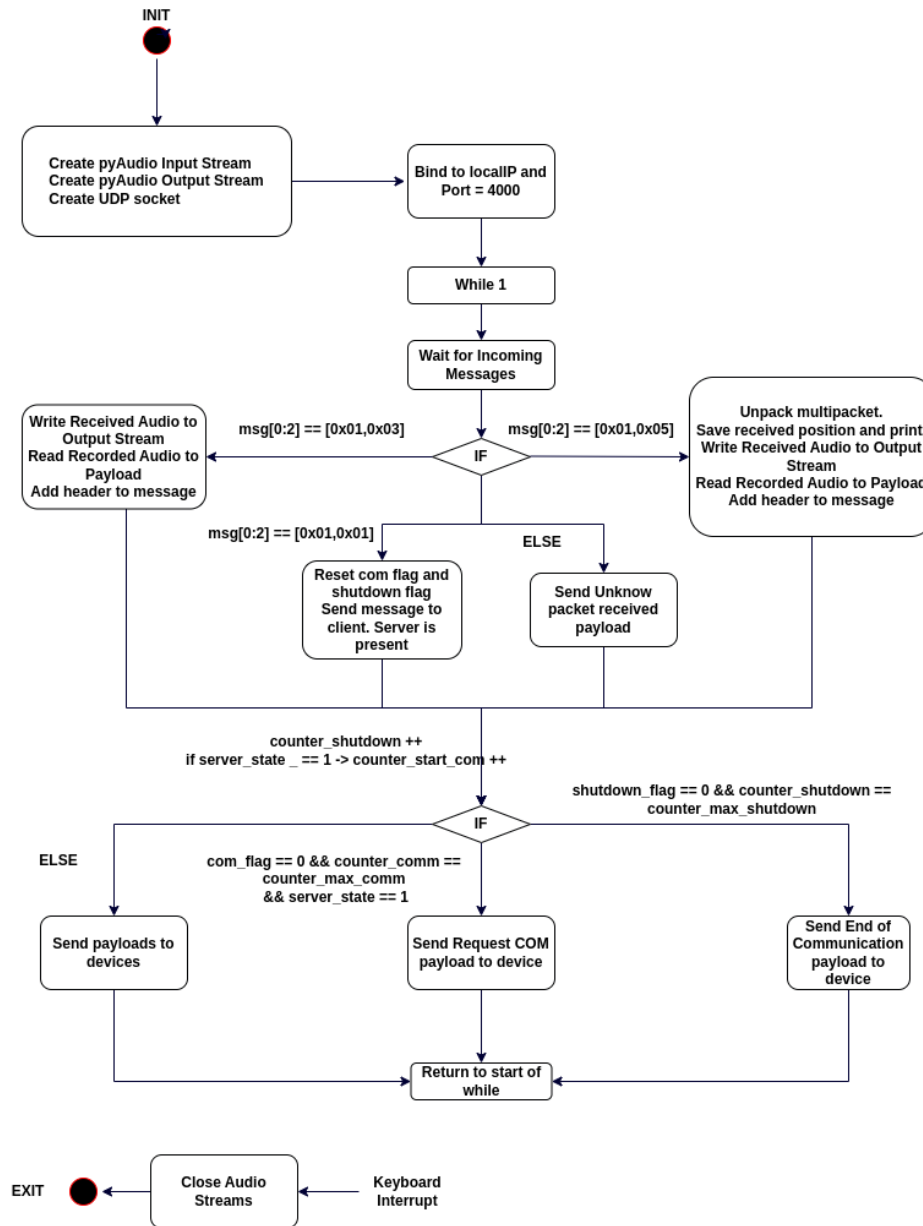


Figure 68: Server Workflow

On start we initiate the Audio streams; the Audio format is 16 bit per sample, 16Khz sample rate and one channel. The frames per chunk are optimised to reduce latency as the server struggles to read and write packets without missing UDP packets.

The keyboard interrupt call-back function is also defined.

```

# Set up the pyaudio recording

CHUNK = 16 #frames per buffer in outgoing stream to udp.
        | | #Small as possible to avoid delays
FORMAT = pyaudio.paInt16 # 16bits per sample pcm
CHANNELS = 1 # Mono audio
RATE = 16000 #16Khz
input_p = pyaudio.PyAudio() #Create the pyaudio object
output_p = pyaudio.PyAudio() #Create the pyaudio object
CHUNK2 = 6000 # Frames per buffer to store data received from esp32

#Stream that reads from pc mic and writes to udp
input_stream = input_p.open(format=FORMAT,
        | | | | | channels=CHANNELS,
        | | | | | rate=RATE,
        | | | | | input=True,
        | | | | | frames_per_buffer=CHUNK)

# Stream that reads from udp incoming packets and writes to speakers
output_stream = output_p.open(format=pyaudio.paInt16,
        | | | | | channels=CHANNELS,
        | | | | | rate=RATE,
        | | | | | output=True,
        | | | | | frames_per_buffer=CHUNK2)

#This is the handler for the keyboard interrupt. It will shutdown
# the server and close the streams
def sigint_handler(signal, frame):
    print("Keyboard Interrupt")
    output_stream.stop_stream()
    output_stream.close()
    input_stream.stop_stream()
    input_stream.close()
    input_p.terminate()
    output_p.terminate()
    sys.exit(0)

signal.signal(signal.SIGINT, sigint_handler) #Set the interrupt handler

```

Figure 69: Initialization of Audio Streams

Once configured we start the socket, and we bind it to localIP and localPort. Those values must be the same as HostIP and HostPort in ESP32, if not packets will not reach its destination.

```

# Create a datagram socket
UDPServerSocket = socket.socket(family=socket.AF_INET, type=socket.SOCK_DGRAM)

# Bind to address and ip
UDPServerSocket.bind((localIP, localPort))

```

Figure 70: Creation of UDP Server Socket

We start the communication loop. In this loop we wait for incoming message and depending on the header, the first two bytes of the messages, we write different payloads.

```
# Listen for incoming datagrams
while(True):
    #Waits for incoming messages
    message,address = UDPServerSocket.recvfrom(1024)

    if(message[:2] == b'\x01\x03'):
        #Writes audio to speakers
        output_stream.write(message[2:])

        #Writes 230 frames of audio from the PC mic
        # to the payload of udp stream
        bytesToSend = b'\x01\x03' + input_stream.read(230)

        server_state = 3

    elif(message[:2] == b'\x01\x05'):
        #write the audio to the speakers
        output_stream.write(message[2:])
        #Saves position
        position = message[2:21]
        #Writes 230 frames of audio to the payload of udp stream
        bytesToSend = b'\x01\x03' + input_stream.read(230)

        #print("0x",position)

    elif(message[:2] == b'\x01\x01'): #means that the client is present
        com_flag = 0
        shutdown_flag = 0
        position = message[2:] #get the position of the client
        bytesToSend = b'\x01\x01' #send message server is present
        server_state = 1 #server is waiting to start full comm

    else:
        bytesToSend = b'\x01\x00' #Unknown packet received
```

Figure 71: Treatment of Received Packets and Creation of Response Payload

As the server does not have a User Interface for the patient staff where they can start communication and stop communication orders. We have added two timers that will simulate this function so we can test the logic in the ESP32 works.

```

if(shutdown_flag == 0):
    counter_shutdown += 1

    if(counter_shutdown == counter_max_shutdown):
        shutdown_flag = 1
        counter_shutdown = 0

if(shutdown_flag == 1):
    bytesToSend = b'\x01\x04'

if(com_flag == 0 and server_state == 1):
    counter_start_comm += 1

    if(counter_start_comm == counter_max_pos):
        com_flag = 1
        counter_start_comm = 0

if(com_flag == 1 and server_state == 1):
    bytesToSend = b'\x03\x02'

```

Figure 72: Implementation of mock control signals

We send the data to the client and start the loop again.

```
UDPServerSocket.sendto(bytesToSend, address)
```

Figure 73: Packet Sending

8 Budget summary

Table 5: Budget summary

Budget				
HARDWARE BOM				
Name	Concept	QTY	Unit of price	Total Price
AZDelivery ESP32 DEVKITC V4	MCU	1	unit	12,00
Gebildet Tact Push Button	Push Button	1	unit	0,08
BOJAK Solderless Flexible Cable	Cables	15	unit	0,30
AdaFruit I2S 3W Class D Amplifier Breakout	I2S DAC Breakout Board	1	unit	16,27
ARISTON Breadboard 710 tie points	Bread Board	2	unit	10,00
BOJACK Breadboard 400 tie points	Bread Board	2	unit	6,00
EUARY Power Bank 5000mAh	Power bank	1	unit	14,00
Weewooday 2W 8 Ohm Speaker	Speakers	1	unit	1,44
Adafruit SPH0645LM4H	I2S MEMS MIC	1	unit	18,13
TECNOIOT INMP441	I2S MEMS MIC	2	unit	14,20
DWM1001-DEV Board	RTLS Board	1	unit	21,00
Segger Jlink Pro	JTAG debug probe	180	day	50,00
SREMTCH Soldering Iron Kit	Soldering Iron Kit	180	day	0,80
LOMVUM Digital Multimeter	Multimeter	180	day	1,70
Subtotal				165,92
Professional Fees				

Industrial Engineer Fees	Development of the project	600	hour	24000
Subtotal				24000
Total				24165,92

9 Analysis and assessment of environmental and social implications



Figure 74: SDG Symbol

The UN sets seventeen Sustainable Development Goals for 2030. It is a call to promote prosperity while building economic growth and address a range of social needs including education, health, social protection, and job opportunities, while tackling climate change and environmental protection. [58]



Figure 75: Goal 3

Goal number three aims to ensure healthy lives and promoting well-being at all ages. This project ensures a good communication between patients and hospital staff, which can provide lifesaving. It also has applications on elderly care homes where the system can be used to track residents with Alzheimer or other degenerative diseases. [59]



Figure 76: Goal 8

Goal number 8 aims to create higher levels of economic productivity through technological upgrading and innovation, through high value added and labour-intensive sectors. The creation of a business unit that manufacture RTLS nurse calling system address this goal. [60]



Figure 77: Goal 9

Goal number nine aims to enhance the scientific, technological capabilities of industrial sectors, encouraging innovation. The project creates new technological capabilities for this country with the manufacturing RTLS Nursing Call Systems. [61]

10 Conclusions

The nurse calling system designed accomplishes most of the goals we wanted to achieve.

The device successfully integrates the DWM1001-DEV module which provides excellent location capabilities in the cm range in an indoor environment.

The device is wireless does not need any wired external interface and the weight of all prototype components is about 250gr creating a portable and light weight device.

The device successfully integrates an independent power source with enough power to have an independent autonomy of more than 24h. It also provides a common USB micro-B interface for easy recharge of the device.

The device has a friendly user interface, with one button we can start communicating with the staff. It can also be remotely activated and only the staff can cut the communication.

The device software is readable and well commented, the division into components dives code based on its functionality which makes the code easier to maintain and scale.

We have created a protocol to share data with the server. Depends on the state of the device it will send nothing, its position or both the position and audio captured. The device in return will read messages coming from the server, control messages like request for starting or stopping communication or audio data.

The captured position is refreshed and shared with the server in less than 3 minutes as it was required.

We have been able to successfully create two parallel audio pipelines for recording and playback of audio in the device with good quality, the parallel audio pipeline work in real time.

The created server can send control commands to the server, it can start an audio transmission or stop it. Depending on the state of the system, it sends PC mic audio to the device and playbacks the received audio with good quality. On receiving position data, it stores the data and prints it to the terminal.

We have not been able to offer the staff a user interface where it can see the data and control the system. The work required to do it was underestimated and the implementation of this functionality must be delayed but is feasible.

The designed software for the ESP32 and the hardware architecture have accomplished the requirements expected for a prototype but still has room for improvement.

Points of improvement have been identified:

- Integrate all components into a PCB. This will reduce weight and volume. Audio quality will benefit from it as it very sensible to EMC interference which will be reduced.
- Use of filters to improve audio quality.
- Integration of a low consumption MEMS MIC, e.g: ICS-43434.
- Implementation of data compression, one minute of 16-bit mono PCM audio are 1.92 Mbytes, data compression will reduce network overload.
- Design of external case for the final product.
- Implementation of voice commands using Machine Learning such as TensorFlow to be used as patient interface and leds.
- Implementation of low consume operating mode of DWM1001-DEV, which will increase the device autonomy.
- Implementation of device Identification.

The designed server is enough to demonstrate that the system is feasible but for a final production stage it needs a lot of improvement. The help of a backend and frontend software engineer would be needed.

The system trials should be performed in an hospital environment to find the appropriate RTLS tags and anchor layout.

11 References

- 1 FDA. "CFR." Code of Federal Regulations Title 21 Chapter I Subchapter H – Medical Devices, <https://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/cfrsearch.cfm?fr=890.3725>. Accessed 4 Sept. 2022.
- 2 FDA. "Federal Register" Code of Federal Regulations Title 21. Subchapter-H, <https://www.ecfr.gov/current/title-21/chapter-I/subchapter-H> . Accessed 4 Sept. 2023.
- 3 EP/CoE. REGULATION (EU) 2017/745 OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL of 5 April 2017 on medical devices, amending Directive 2001/83/EC, Regulation (EC) No 178/2002 and Regulation (EC) No 1223/2009 and repealing Council Directives 90/385/EEC and 93/42/EEC. 2017, <https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:02017R0745-20200424&from=EN#tocId3> . Accessed 5 Sept. 2022.
- 4 MDCG. *Guidance on Classification of Medical Devices*. Oct. 2021, https://health.ec.europa.eu/system/files/2021-10/mdcg_2021-24_en_0.pdf. Accessed 7 Sept. 2022.
- 5 "ISO 14971:2019." ISO, <https://www.iso.org/standard/72704.html>. Accessed 10 Sept. 2022.
- 6 "IEC 80001-1:2021." ISO, <https://www.iso.org/standard/72026.html>. Accessed 10 Sept. 2022.
- 7 "IEC 80001-1:2021." ISO, <https://www.iso.org/standard/72026.html>. Accessed 10 Sept. 2022.
- 8 "ISO 13485:2016." ISO, <https://www.iso.org/standard/59752.html>. Accessed 10 Sept. 2022.
- 9 SCHRACK SECONET. *Call Units for Nurse Call Systems*. https://carecom-solutions.com/en_ccs-nurse-call-products/en_ccs-nurse-call-call-units.html. Accessed 16 Sept. 2022.
- 10 SCHRACK SECONET. *Nurse Control Station*. https://carecom-solutions.com/en_ccs-nurse-call-products/en_about-our-products-in-detail/en_ccs-nurse-call-display-units-nurse-control-station.html . Accessed 16 Sept. 2022.
- 11 COBS. *IP-DECT 400 COBS Talk*. <https://cobs-ws.com/en/products/wireless-communication/item/ip-dect-400-cobs-talk> . Accessed 17 Sept. 2022.
- 12 COBS. *Alarms with Indoor Location*. <https://cobs-ws.com/en/products/wireless-communication/item/larm-med-tal-och-positionering>. Accessed 17 Sept. 2022.
- 13 Stanley Health Care. *MobileView®5 Most Advanced Software Platform for Unified Asset and People Visibility Management*. 2022,

https://www.stanleyhealthcare.com/sites/stanleyhealthcare.com/files/2022-10/MobileView%20Data%20Sheet_09-2022.pdf . Accessed 18 Sept. 2022.

14 Stanley Health Care. *MobileView®5 Most Advanced Software Platform for Unified Asset and People Visibility Management*. 2022,

https://www.stanleyhealthcare.com/sites/stanleyhealthcare.com/files/2022-10/MobileView%20Data%20Sheet_09-2022.pdf . Accessed 18 Sept. 2022.

15 Greening, Chris. "ESP32 Walkie-Talkie." Hackster.io, 5 Apr. 2021, <https://www.hackster.io/chris-greening/esp32-walkie-talkie-f8d0dd>. Accessed 21 Sept. 2022.

16 Localino. "Localino: Open Source Indoor Positioning System (Arduino + Decawave)." Instructables, 1 Dec. 2016, <https://www.instructables.com/Localino-Open-Source-Indoor-Location-System-Arduino/>. Accessed 21 Sept. 2022.

17 Qorvo. "DWM1001-DEV - Qorvo." Qorvo - RF Solutions for Mobile, Infrastructure and Defense, <https://www.qorvo.com/products/p/DWM1001-DEV>. Accessed 14 Sept. 2022.

18 Campbell, Scott. "Basics of UART Communication." Circuit Basics, 13 Feb. 2016, <https://www.circuitbasics.com/basics-uart-communication/>. Accessed 18 Sept. 2022.

19 Campbell, Scott. "Basics of the SPI Communication Protocol." Circuit Basics, 13 Feb. 2016, <https://www.circuitbasics.com/basics-of-the-spi-communication-protocol>. Accessed 18 Sept. 2022.

20 Ltd., Decawave/Qorvo. DWM1001 Firmware API Guide. Page 43. 2019, <https://www.qorvo.com/products/d/da007975> . Accessed 22 Sept. 2022.

21 Computer Hope. What Is a Server? <https://www.computerhope.com/jargon/s/server.htm> . Accessed 22 Sept. 2022.

22 Knerl, Linsey. "Does My Computer Have Bluetooth?" HP, 18 Oct. 2019, <https://www.hp.com/us-en/shop/tech-takes/does-my-computer-have-bluetooth>. Accessed 22 Sept. 2022.

23 Tanaza. "5 Things You Should Know about Wi-Fi Speed and Distance Covered". 17 Mar. 2017, <https://www.tanaza.com/classichotspot/blog/5-things-know-wi-fi-speed-distance-covered/>. Accessed 22 Sept. 2022.

24 Texas Instruments, Incorporated. Principles of Data Acquisition and Conversion (Rev. A). 2015, <https://www.ti.com/lit/an/sbaa051a/sbaa051a.pdf>. Accessed 26. Sept. 2022.

25. STMICROELECTRONICS. *Datasheet - STM32F205xx STM32F207xx*. 2020, <https://www.st.com/resource/en/datasheet/stm32f207vg.pdf>. Accessed 26 Sept. 2022.

26. Arduino. *Produt Reference Manual – Arduino UNO R3 2023*.

<https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf>. Accessed 26 Sept. 2022

27. Espressif. *ESP32-DevKitC V4 Getting Started Guide - ESP32 - — ESP-IDF Programming Guide Latest Documentation*. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html#get-started-esp32-devkitc-board-front>. Accessed 26 Sept. 2022.

28 TensorFlow. *TensorFlow Lite for Microcontrollers*. <https://www.tensorflow.org/lite/microcontrollers>. Accessed 30 Sept. 2022.

29 Trenz Electronic GmbH. “DesignWare ARC EM Software Development Platform.” Trenz Electronic GmbH Online Shop (EN), <https://shop.trenz-electronic.de/en/TEC0089-02-D2C-1-D-DesignWare-ARC-EM-Software-Development-Platform?c=539>. Accessed 2 Oct. 2022.

30. Espressif. *Certificates*. <https://www.espressif.com/en/support/documents/certificates>. Accessed 2 Oct. 2022.

31. Espressif. *ESP32 DevKitC V4 Schematics*. 06 Dec. 2017, https://dl.espressif.com/dl/schematics/esp32_devkitc_v4-sch.pdf. Accessed 2 Oct. 2022.

32. Amazon. “EUARY PowerBank 5000mAh - Cargador Portátil Ultra Compacto Con Salida 2A.” *Batería Externa de Bolsillo Para Batería de Huawei, Samsung, Xiaomi y Otros (Negro)*: Amazon.Es: Electrónica, <https://www.amazon.es/gp/product/B082PMBWRZ>. Accessed 2 Oct. 2022.

33. STM. “MEMS Microphones.” STMicroelectronics, <https://www.st.com/en/mems-and-sensors/mems-microphones.html>. Accessed 4 Oct. 2022.

34. Cuidevices. “Comparing MEMS and Electret Condenser (ECM) Microphones.” CUI Devices, 8 Jan. 2019, <https://www.cuidevices.com/blog/comparing-mems-and-electret-condenser-microphones>. Accessed 4 Oct. 2022.

35. KNOWLES. “SPH0645LM4H-B I2S Output Digital Microphone.” 17 July 2015, <https://cdn-shop.adafruit.com/product-files/3421/i2S+Datasheet.PDF>. Accessed 4 Oct. 2022.

36. IvenSense. *INMP441 Omnidirectional Microphone with Bottom Port and I2 S Digital Output*. 2014, <https://invensense.tdk.com/wp-content/uploads/2015/02/INMP441.pdf>. Accessed 4 Oct. 2022.

37. IvenSense. *ICS-43434 Multi-Mode Microphone with I2S Digital Output*. 2014, <https://invensense.tdk.com/wp-content/uploads/2016/02/DS-000069-ICS-43434-v1.2.pdf>. Accessed 4 Oct. 2022.

38. Adafruit. *Adafruit I2S 3W Class D Amplifier Breakout - MAX98357A*. <https://www.adafruit.com/product/3006>. Accessed 4 Oct. 2022.

39. Amazon. *Altavoz Weewooday 6 Piezas 2W 8 Ohmios*.
<https://www.amazon.es/dp/B09MRK24PP>. Accessed 4 Oct. 2022.
40. DecaWave. *DW1000 Datasheet*. 2017, <https://www.qorvo.com/products/d/da007952>. Accessed 7 Oct. 2022.
41. Amazon. *Gebildet Tact Push Button Interruptor*.
<https://www.amazon.es/dp/B082DBBPGC?th=1>. Accessed 7 Oct. 2022.
42. Espressif. *Build System - ESP32 - ESP-IDF Programming Guide Latest Documentation*. <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/build-system.html>. Accessed 23 Oct. 2023.
43. Wireshark. *Wireshark.Frequently Asked Questions*.
https://www.wireshark.org/faq.html#_what_is_wireshark. Accessed 14 Nov. 2022.
44. Terso. "What Is RFID? HF vs. UHF Explained." Terso, 20 Sept. 2019,
<https://www.tersosolutions.com/what-is-rfid/>. Accessed 18 Nov. 2022.
45. Gladysz, B., and K. Santarek. "AN APPROACH TO RTLS SELECTION." *DEStech Transactions on Engineering and Technology Research*, no. icpr, Mar. 2018,
<https://doi.org/10.12783/dtetr/icpr2017/17576>.
46. Qorvo. *DWM1001 System Overview and Performance*. 2018,
<https://www.qorvo.com/products/d/da007974>. Accessed 13 Nov. 2022.
47. Qorvo. *DWM1001 Firmware API Guide*. 2019,
<https://www.qorvo.com/products/d/da007975>. Accessed 20 Nov. 2022.
48. Espressif. *Universal Asynchronous Receiver/Transmitter (UART) - ESP32 - ESP-IDF Programming Guide v4.4.3 Documentation*. <https://docs.espressif.com/projects/esp-idf/en/v4.4.3/esp32/api-reference/peripherals/uart.html>. Accessed 20 Oct. 2022.
49. Espressif. *FreeRTOS - ESP32 - ESP-IDF Programming Guide v4.4.3 Documentation*. <https://docs.espressif.com/projects/esp-idf/en/v4.4.3/esp32/api-reference/system/freertos.html>. Accessed 20 Nov. 2022.
50. Espressif. *GPIO & RTC GPIO - ESP32 - ESP-IDF Programming Guide v4.4.3 Documentation*. <https://docs.espressif.com/projects/esp-idf/en/v4.4.3/esp32/api-reference/peripherals/gpio.html>. Accessed 26 Nov. 2022.
51. Espressif. *Wi-Fi - ESP32 - ESP-IDF Programming Guide Latest Documentation*. https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/network/esp_wifi.html. Accessed 30 Nov. 2022.

52. Espressif. "Esp-Idf/Station_example_main.c at v4.4.1 · Espressif/Esp-Idf." *Github*, https://github.com/espressif/esp-idf/blob/v4.4.1/examples/wifi/getting_started/station/main/station_example_main.c. Accessed 20 Nov. 2022.
53. Eggert, L., et al. *RFC 8085: UDP Usage Guidelines*. <https://www.rfc-editor.org/rfc/rfc8085.html>. Accessed 22 Nov. 2022.
54. Cox, Richard, et al. "ITU-T Coders for Wideband, Superwideband, and Fullband Speech Communication [Series Editorial." *IEEE Communications Magazine*, vol. 47, no. 10, Oct. 2009, pp. 106–09, <https://doi.org/10.1109/mcom.2009.5273816>.
55. Postel, J. *RFC 768: User Datagram Protocol*. <https://www.rfc-editor.org/rfc/rfc768>. Accessed 24 Nov. 2022.
56. Eddy, Wesley M. *RFC 9293: Transmission Control Protocol (TCP)*. 2022, <https://www.rfc-editor.org/rfc/rfc9293.html>. Accessed 24 Nov. 2022.
57. Pharm, Hubert. *PyAudio: Cross-Platform Audio I/O for Python, with PortAudio*. 2006, <https://people.csail.mit.edu/hubert/pyaudio/>. Accessed 31 Nov. 2022.
58. United Nations. "Home." *United Nations Sustainable Development*, 12 Mar. 2018, <https://www.un.org/sustainabledevelopment/>. Accessed 31 Dec. 2022.
59. United Nations. "Health." *United Nations Sustainable Development*, 7 Jan. 2015, <https://www.un.org/sustainabledevelopment/health/>. Accessed 31 Dec. 2022.
60. United Nations. "Economic Growth." *United Nations Sustainable Development*, 7 Jan. 2015, <https://www.un.org/sustainabledevelopment/economic-growth/>. Accessed 31 Dec. 2022.
61. United Nations. "Infrastructure and Industrialization." *United Nations Sustainable Development*, 7 Jan. 2015, <https://www.un.org/sustainabledevelopment/infrastructure-industrialization/>. Accessed 9 Jan. 2023.
62. ZVEI. *Call Systems According to DIN VDE 0834*. 2017, https://www.zvei.org/fileadmin/user_upload/Presse_und_Medien/Publikationen/2017/Juli/Rufanlagen_nach_DIN_VDE_0834/Rufanlagen_Call_Systems_DIN_VDE_0834_Online.pdf. Accessed 8 Sept. 2022.

ANNEX A CONFIGURE DWM1001-DEV AS TAGS OR ANCHORS

Hardware toolchain

- DWM1001-C
- PC with Ubuntu.

Software toolchain

- Segger J-Flash Lite available at <https://www.segger.com/downloads/embedded-studio>
- Minicom.

Flash last firmware

1. Download DWM1001C Software and Documentation Pack at <https://www.qorvo.com/products/d/da008479>.
2. Open J-Flash Lite.
3. Choose nrf52832_XXAA as Device and SWD as interface, use default speed 1000. Click "OK".

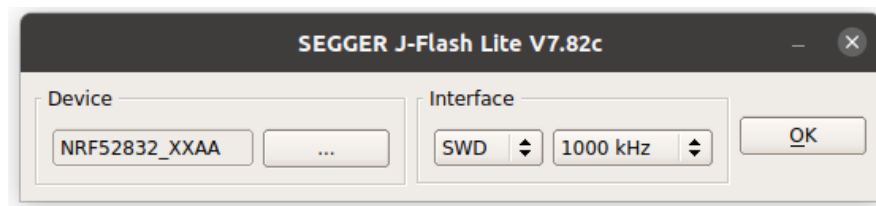


Figure 78: Screenshot Step 3 Flashing Firmware

4. Click "Erase Chip" to do a full chip erase.
5. In Data File, click and browse to the hex file provided in the DWM1001C Software and Documentation Pack (\$ExtractionPath/DWM1001/Factory_Firmware_Image/DWM1001_PANS_R2.1.hex) to flash, click "Program Device".

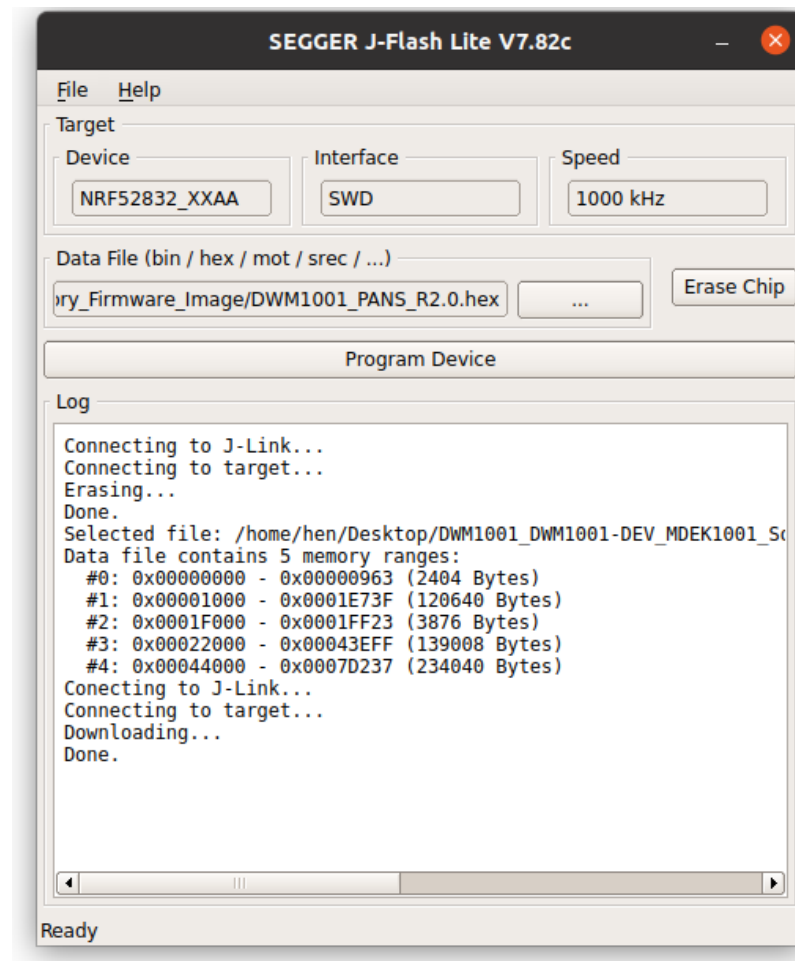


Figure 79: Screenshot Step 5 Flashing Firmware

The LEDs on the boards should be active once the flash update completes.

3.3 Guides to flash the DWM1001 with factory image Source

Set DWM1001-DEV as Anchor or TAG

FIRMWARE API GUIDE 6 SHELL COMMANDS page 97

1. Download minicom.
 - Open terminal. Do `Sudo apt-get install minicom`.
2. Configure the minicom terminal. Do `minicom -s`

```
hen@hen-GL552VW:~$ minicom -s
```

Figure 80: Step 2 Configuring DWM1001-DEV

3. Go to Serial setup and check if Bps/Par/Bits is 115200 8N1. If not Change click E and change.
4. Do `dmseg` and check which device name has DWM1001 module.

```
hen@hen-GL552VW:~$ dmesg
```

Figure 81: Step 4 Configuring DWM1001-DEV

5. Do `sudo minicom -D {device_name}`

```
hen@hen-GL552VW:~$ sudo minicom -D /dev/ttyACM0
```

Figure 82: Step 5 Configuring DWM1001-DEV

6. When this message pops up do double enter to join shell mode, a message from the vendor will print.

```
Welcome to minicom 2.7.1

OPTIONS: I18n
Compiled on Dec 23 2019, 02:06:26.
Port /dev/ttyACM0, 21:36:17

Press CTRL-A Z for help on special keys
```

Figure 83: Step 6 Configuring DWM1001-DEV 1

```
dwm>
dwm>

DWM1001 TWR Real Time Location System

Copyright : 2016-2019 LEAPS and Decawave
License : Please visit https://decawave.com/dwm1001_license
Compiled : Mar 27 2019 03:35:59

Help : ? or help
```

Figure 84: Step 6 Configuring DWM1001-DEV 2

7. Configure the first node as anchor-initiator. **There should only be one node set as anchor-initiator.** Type `nmi` in the console. Then re-enter shell mode doing double enter, after entering the command `nmg` it should output "ani (act,real)".

```
dwm> nmi
```

Figure 85: Step 7 Configuring DWM1001-DEV 1

```
dwm> nmg
mode: ani (act,real)
```

Figure 86: Step 7 Configuring DWM1001-DEV 2

8. To configure a node as an anchor, enter the shell and the command `nma`. Re-enter the shell. Enter the command `nmg`, the output should be "an (act,-)".

```
dwm> nma
```

Figure 87: Step 8 Configuring DWM1001-DEV 1

```
dwm> nmg
mode: an (act,-)
```

Figure 88: Step 8 Configuring DWM1001-DEV 2

9. To configure a node as a tag, enter the shell and the command `nmt`. Re-enter the shell. Enter the command `nmg`, the output should be "tn (act,twr,np,le)".

```
dwm> nmt
```

Figure 89: Step 9 Configuring DWM1001-DEV 1

```
dwm> nmg  
mode: tn (act,twr,np,le)
```

Figure 90: Step 9 Configuring DWM1001-DEV 2

ANNEX B OTHER RELEVANT STANDARDS

DIN EN ISO 11197 is a standard that specifies requirements for medical supply units.

IEC 60601-1 specifies general requirements for the basic safety and essential performance of medical electrical equipment.

IEC 60601-1-8 is a supplementary standard to DIN EN 60601-1 that specifies general requirements for the safety of medical electrical equipment, specifically related to alarm systems.

IEC 61000-6-1 sets requirements for the immunity of residential, commercial, and light-industrial environments to electromagnetic interference.

IEC 61000-6-3 specifies requirements for the emission of electromagnetic interference from residential, commercial, and light-industrial environments.

DIN VDE 0100-710 establishes requirements for low-voltage electrical installations in medical locations.

DIN VDE 0100-410 specifies requirements for the protection of low-voltage electrical installations for safety, specifically related to protection against electric shock.

EN 50468 is a standard that specifies resistibility requirements for equipment with telecommunications ports to overvoltages and overcurrents caused by lightning.

EN 62368 sets requirements for audio/video, information, and communication technology equipment. It has been replaced by DIN EN 62368.

EN 50134 establishes requirements for alarm systems in domestic situations. [\[62\]](#)

ANNEX C BUDGET

Table 6: Budget

Budget						
HARDWARE BOM						
Name	Concept	QTY	Unit of Measure	Rate €	Unit of price	Total Price
AZDelivery ESP32 DEVKITC V4	MCU	1	unit	12,000	unit	12,00
Gebildet Tact Push Button	Push Button	1	unit	0,08	unit	0,08
BOJAK Solderless Flexible Cable	Cables	15	unit	0,02	unit	0,30
AdaFruit I2S 3W Class D Amplifier Breakout	I2S DAC Breakout Board	1	unit	16,27	unit	16,27
ARISTON Breadboard 710 tie points	Bread Board	2	unit	5,0	unit	10,00
BOJACK Breadboard 400 tie points	Bread Board	2	unit	14,00	unit	6,00
EUARY Power Bank 5000mAh	Power bank	1	unit	14,00	unit	14,00
Weewooday 2W 8 Ohm Speaker	Speakers	1	unit	1,44	unit	1,44
Adafruit SPH0645LM4H	I2S MEMS MIC	1	unit	18,13	unit	18,13
TECNOIOT INMP441	I2S MEMS MIC	2	unit	7,10	unit	14,20
DWM1001-DEV Board	RTLS Board	1	unit	21	unit	21,00
Segger Jlink Pro	JTAG debug probe	180	day	0,2750	day	50,00
SREMTCH Soldering Iron Kit	Soldering Iron Kit	180	day	0,005	day	0,80

LOMVUM Digital Multimeter	Multimeter	180	day	0,0094	day	1,70
Subtotal						165,92
Professional Fees						
Industrial Engineer Fees	Development of the project	600	hour	40	hour	24000
Subtotal						24000
Total						24165,92

Table 7: Component providers

<i>Component</i>	<i>Provider</i>
AZDelivery ESP32 DEVKITC V4	AZDelivery Amazon Store
Gebildet Tact Push Button	Gebildet Amazon Store
BOJAK Solderless Flexible Cable	BOJACK Amazon Store
AdaFruit I2S 3W Class D Amplifier Breakout	Paradisetrone Amazon Store
ARISTON Breadboard 710 tie points	Out of Production
BOJACK Breadboard 400 tie points	BOJACK Amazon Store
EUARY Power Bank 5000mAh	EUARY Amazon Store
Weewooday 2W 8 Ohm Speaker	Weewooday Amazon Store
Adafruit SPH0645LM4H	Paradisetrone Amazon Store
TECNOIOT INMP441	TECNOIOT Amazon Store
DWM1001-DEV Board	Mouser
Segger Jlink Pro	SEGGER
SREMTCH Soldering Iron Kit	SREMTCH Amazon Store
LOMVUM Digital Multimeter	LOMVUM Amazon Store

ANNEX D HARDWARE COMPONENT DATA SHEET

Table 8: Data Sheets

Component	Data Sheet
AZDelivery ESP32 DEVKIT C V4	AZ_DELIVERY_ESP32_DvKit_C_DATA_SHEET
Adafruit I2S 3W Clase D Amplificador Breakout - MAX98357A	MAX98357A/B DATASHEET
Adafruit SPH0645LM4H	SPH0645LM4H DATASHEET
DWM1001-DEV	DWM1001-DEV DATASHEET

ANNEX E GANTT DIAGRAM

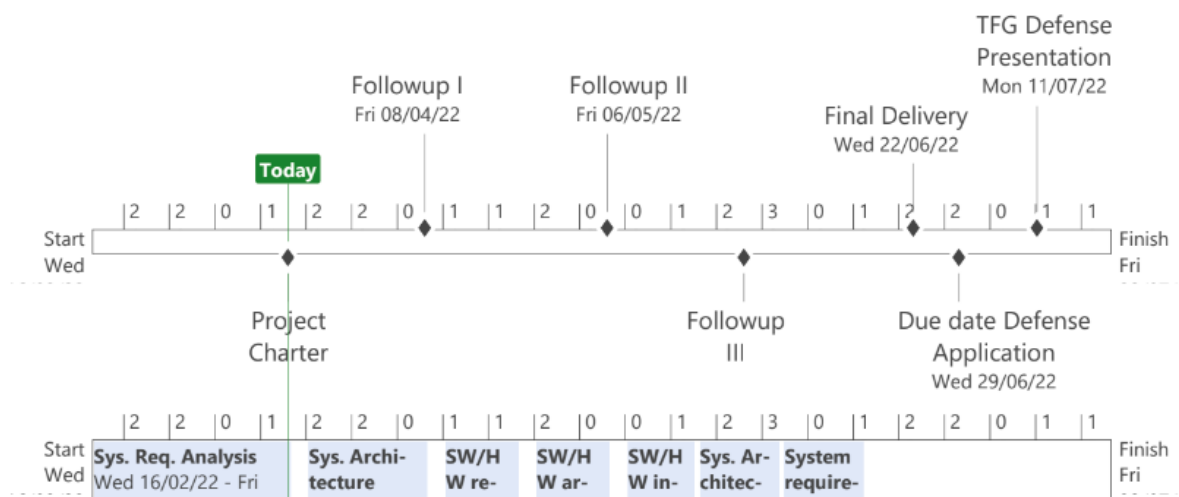


Figure 91: Initial Gantt Chrono

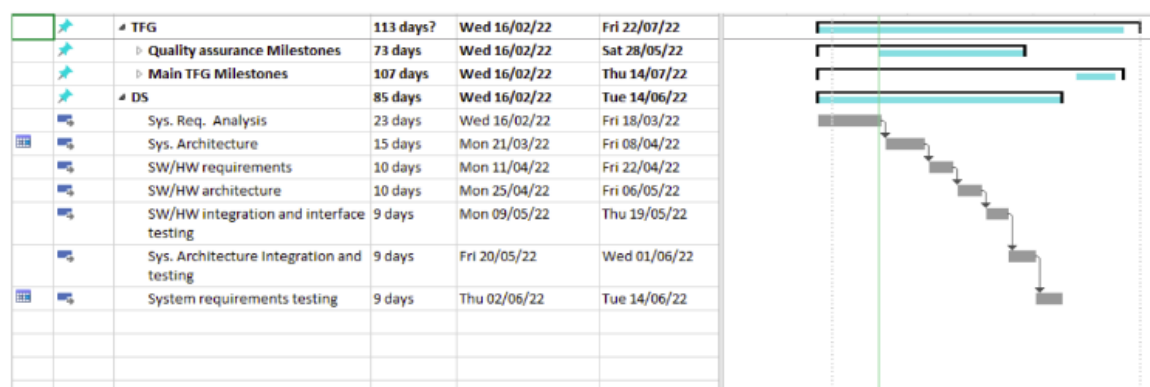


Figure 92: Initial Gantt Chart and Diagram

The initial planned duration of the project was one semester, but the project has been delayed one semester. The reasons where:

The project requirements discussed with the tutor on the start of the project cannot be feasible reached by one student on its own in one semester.

My knowledge of C and the ESP-IDF framework was not good enough. This delayed the project as I had to start an online course to learn.

As we know there has been a global microchip crisis, delivery times for electronic components are longer than expected and the reliability of them is worse. We bought MEMS mic to an Amazon Store with good reviews but once the product arrived and was tested it did not work. Microphones from another store had to be bought but their delivery times where long. This delayed the project one month and a half as we were incapable to test the code and we could not work in other components in parallel.

There have not been follow-up deliverables as the tutor did not request them.

ANNEX E QUALITY CHECKLIST

Quality checklist

Student's name Enric Botella Pastor.

Final thesis title: Design of a Nurse Calling System with Real Time Indoor Location Capabilities.

Formal features – Format

A1 – Cover formats	<input checked="" type="checkbox"/>
A2 – Table of contents	<input checked="" type="checkbox"/>
A3 – Summary of tables and figures	<input checked="" type="checkbox"/>
A4 – Spelling/units	<input checked="" type="checkbox"/>
A5 – Tables/figures	<input checked="" type="checkbox"/>
A6 – Document formats	<input checked="" type="checkbox"/>
A7 – Thesis length	<input checked="" type="checkbox"/>
A8 – References	<input checked="" type="checkbox"/>
A9 – List of documents	<input checked="" type="checkbox"/>

Formal features – Content

B1 – Presentation of the problem	<input checked="" type="checkbox"/>
B2 – Background and literature review	<input checked="" type="checkbox"/>
B3 – Presentation and justification of solutions	<input checked="" type="checkbox"/>
B4 – Compliance with the scope and specifications	<input checked="" type="checkbox"/>
B5 – Economic, environmental and safety features	<input checked="" type="checkbox"/>
B6 – Timing	<input checked="" type="checkbox"/>
B7 – Conclusions and recommendations	<input checked="" type="checkbox"/>

Assessment criteria

Difficulty of the work	<input checked="" type="checkbox"/> Very high	<input type="checkbox"/> High	<input type="checkbox"/> Medium	<input type="checkbox"/> Low
Accomplishment of objectives	<input type="checkbox"/> Complete	<input checked="" type="checkbox"/> High	<input type="checkbox"/> Partial	
Support and monitoring by the professor	<input type="checkbox"/> Very high	<input type="checkbox"/> High	<input checked="" type="checkbox"/> Medium	<input type="checkbox"/> Low
Number of hours spent on the thesis	<input checked="" type="checkbox"/> More than 600	<input type="checkbox"/> More than 400	<input type="checkbox"/> More than 300	<input type="checkbox"/> Less than 300

Other considerations:



Student's signature

Enric Botella Pastor