UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
FACULDADE DE ENGENHARIA DA COMPUTAÇÃO E
TELECOMUNICAÇÕES

Pedro Henrique Fernandes Rendeiro

DEVELOPMENT OF AN IOT SYSTEM FOR LEAKAGE CURRENT MONITORING IN HIGH VOLTAGE INSULATORS

UFPA / ITEC / FCT
Campus Universitário do Guamá
Belém-Pará-Brasil
2023

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
FACULDADE DE ENGENHARIA DA COMPUTAÇÃO E
TELECOMUNICAÇÕES

Pedro Henrique Fernandes Rendeiro

DEVELOPMENT OF AN IOT SYSTEM FOR LEAKAGE CURRENT MONITORING IN HIGH VOLTAGE INSULATORS

Trabalho de Conclusão de Curso apresentado à Faculdade de Engenharia da Computação e Telecomunicações da Universidade Federal do Pará, como requisito para a obtenção do Grau de Bacharel em Engenharia da Computação.

UFPA / ITEC / FCT
Campus Universitário do Guamá
Belém-Pará-Brasil
2023

In memory of Claudio Rendeiro.

Acknowledgments

Many people insist that everyone should be the protagonist of their own story. I believe that could not be more wrong. So I thank God, who gave meaning to my short story within the great story of his – and who allowed for a bit of engineering in the middle of all of that. In addition, it is impossible to ignore the significance of so many people up to this point.

I thank my parents – Claudio *(in memoriam)* and Adriana – for always supporting me. I would never have made it this far without your support. I also thank my siblings – Sophia *(in memoriam)*, Ana Clara, Rebeca and Kalleb – for all the memories we have gathered. You will always be part of who I am. Likewise, I thank "Os Todos" – grandparents, uncles, aunts, cousins and niece – for caring despite the distance. I could never ask for a better family.

I thank my classmates for sharing the struggles and achievements. In particular, I must mention Danilo Nicioka, for the discussions and collaborations that have enriched my studies from basic education (2007) up to college (2023). You are the only one who has not yet realized the gigantic potential you have. Furthermore, I thank the members of CAECOMP for reminding me that it is better to serve than to be served. Working with you is a privilege.

I thank everyone at LASSE for making it such a great place to work. LASSE is indeed a family. In particular, I thank Jamile Leite, Mercedes Diniz, Lucas Damasceno, Marcos Davi and Flávio Nunes for the technical contributions that brought this project into being. This achievement also belongs to you.

I thank every professor who has contributed to my education. Most of all, my advisor Leonardo Ramalho, not only for giving me an opportunity, but also for teaching me how to make the most of it. I look forward to continuing this partnership in the future.

I thank INESC P&D Brasil and Eneva S.A. for funding this work.

Pedro Rendeiro

October 29th, 2023

Tomou, pois, o Senhor Deus ao homem e o colocou no jardim do Éden para o cultivar e o guardar.

Gênesis 2:15

List of Acronyms

3GPP 3rd Generation Partnership Project

6LoWPAN IPv6 Over Low-Power Wireless Personal Area Network

AC alternating current

ADC analog-to-digital converter

AmpOp operational amplifier

AP access point

API application programming interface

CNN convolutional neural network

COTS commercial off the shelf

DAQ data acquisition

D-BPSK Differential Binary Phase-Shift Keying

DC direct current

DFU device firmware update

DHCP Dynamic Host Configuration Protocol

DL downlink

DSSS direct-sequence spread spectrum

eSIM embedded-SIM

FSM finite state machine

GPIO general purpose input/output

HTTP Hypertext Transfer Protocol

HV high voltage

IC integrated circuit

IEEE Institute of Electrical and Electronics Engineers

IETF Internet Engineering Task Force

IoT internet of things

IR infrared

ISM industrial, scientific and medical

ISR interrupt service routine

ITU International Telecommunication Union

ITU-T ITU Telecommunication Standardization Sector

LAN local area network

LDO low dropout

LoRa Long Range

LoRaWAN Long Range Wide Area Network

LOS line-of-sight

NIST National Institute of Standards and Technology

NLOS non-line-of-sight

LPWAN low power wide area network

MAC medium access control

MCU microcontroller

ML machine learning

NB-IoT Narrowband IoT

NTP Network Time Protocol

OS operating system

OTA over-the-air

PCB printed circuit board

RFC Request for Comments

RSSI received signal strength indicator

RTC real time clock

SDR software-defined radio

SLOC source lines of code

SMD surface-mount device

SPI Serial Peripheral Interface

UMSCF Microprocessed Unit for Leakage Current Sensors

UART Universal Asynchronous Receiver/Transmitter

UL uplink

UML Unified Modeling Language

UNB ultra-narrowband

UV ultraviolet

WLAN wireless local area network

List of Figures

List of Tables

Contents

Abstract

High voltage (HV) insulators are widely adopted to provide mechanical support and electrical insulation in the electricity sector. However, they suffer with the pollution that accumulates on them over time. Thermal plants often face an extra challenge when they settle in coastal areas. In a sense, such regions are advantageous to the companies due to the availability of water for cooling; however, the inherent salinity of the location negatively impacts the performance of the HV insulators. With this in mind, this work proposes a solution based on Internet of Things (IoT) to optimize the washing cycles period of this equipment, so the companies avoid both unnecessary costs with too frequent washings and even higher costs with monetary fines due to failures on the grid. The proposed system consists of multiple IoT devices to capture leakage current signals across different HV insulators on the substation of a thermal unit. This work details the development process of each part of the system, including the design and implementation of the hardware, embedded software, wireless network, server application, as well as a web platform for remote management of the devices. Results indicate that the final solution is useful to digitize the information from the current sensors and reliably transmit it over the network. Such information may be used – along with other features – as a decision variable to help determine to optimum moment to wash the HV insulators, thus renewing its lifespan.

***Keywords* —** Internet of Things, Data Acquisition, Leakage Current, High Voltage Insulators.

Resumo

Os isoladores de alta tensão (AT) são amplamente adotados para fornecer suporte mecânico e isolamento elétrico no setor de energia elétrica. No entanto, eles sofrem com a poluição que se acumula sobre eles ao longo do tempo. As usinas termoelétricas comumente enfrentam um desafio extra ao escolherem áreas costeiras para se instalarem. De certa forma, tais regiões são vantajosas devido à disponibilidade de água para resfriamento; entretanto, a salinidade inerente ao local impacta negativamente o desempenho dos isoladores de AT. Com isso em mente, este trabalho propõe uma solução baseada em Internet das Coisas (IoT) para otimizar o período dos ciclos de lavagem desses equipamentos, de forma que as empresas evitem tanto custos desnecessários com lavagens muito frequentes quanto custos ainda maiores com multas devido a falhas na rede. O sistema proposto consiste em múltiplos dispositivos IoT para capturar sinais de corrente de fuga em diferentes isoladores de AT na subestação de uma unidade termoelétrica. Este trabalho detalha o processo de desenvolvimento de cada parte do sistema, incluindo o projeto e a implementação de hardware, software embarcado, rede sem fio, aplicação servidor, bem como uma plataforma web de gerenciamento remoto dos dispositivos. Os resultados indicam que a solução final é útil para digitalizar a informação dos sensores de corrente e transmiti-la de forma confiável através da rede. Esta informação pode ser utilizada, em conjunto com outras características, como variável de decisão para ajudar a determinar o momento ótimo para lavar os isoladores de AT, renovando assim o seu tempo de vida útil.

Palavras-chave — Internet das Coisas, Aquisição de Dados, Corrente de Fuga, Isoladores de Alta Tensão.

Chapter 1

Introduction

The problem of salinity impacts on high voltage (HV) insulators in thermal power plants located close to the sea has a significant track record. With the advance of technology and the growth in demand for electricity, many of these plants were built in coastal areas due to the availability of water for cooling [1]. However, proximity to the sea exposes equipment, such as insulators, to corrosion and degradation caused by the salinity of the environment. Saltwater contains ions that can corrode insulating materials, compromising their efficiency and safety. These impacts have led to increased maintenance and replacement costs for insulators, as well as interruptions in the power supply [2].

Faced with this challenge, the industry has sought solutions, such as the development of more corrosion-resistant materials and protection techniques, in order to minimize the effects of salinity and ensure efficient operation of thermoelectric plants [3]. However, with this type of solution, even if the effects are minimized, the accumulation of salts would continue to occur in the long term and could cause power grid failures at some point that is difficult to determine.

Another type of solution involves the concept of the internet of things (IoT) and the use of sensors to capture signals from the environment in order to infer the current state of the insulators [4]. In this way, it is possible to optimize the washing cycles period of these equipments, thus avoiding unnecessary recurring maintenance costs in one hand, as well as monetary fines due to failures in the power distribution. That is the approach used by the current work.

This work was carried out as part of the R&D project called Management of Salinity Impacts on Insulation (GImpSI – *"Gestão dos Impactos da Salinidade em Isolamentos"*), developed in partnership with and INESC P&D Brasil, Eneva S.A., as well as other higher education institutions – under the framework of the R&D Program of the Brazilian Electricity Regulatory

Agency (ANEEL), code PD-11278-0001-2021.

Before diving into specific parts of the system, it is important to understand its context and the overall goal. The main product of the project is a system to monitor the impacts of high salinity at the substation of a thermal unit named "UTE Porto de Sergipe I" in the city of Barra dos Coqueiros, Sergipe, Brazil. Nevertheless, the idea is that it could be expanded to be used in other electrical substations. This system is currently under development by several teams in different areas of activity, in the following list.

Areas of Activity:

1. An IoT solution with sensors strategically installed on some of the substation's HV insulators to monitor leakage currents.
2. The use of commercial off the shelf (COTS) equipment such as a weather station, infrared (IR) camera and ultraviolet (UV) detection camera for capturing meteorological and image data.
3. A server capable of concentrating and processing the information collected by the other components of the system.

The first step of the Area of Activity 1 was to develop a leakage current sensor, which is document in [5, 6], where current transformers are used for that purpose. The next step within the first Area is the object of the current work, and it is thoroughly discussed throughout the text. The Areas of Activity 2 and 3 encompasses various works. One of those works is [7], where it is proposed a method for monitoring the operational condition of zinc oxide (ZnO) lightning rods based on IR measurements of the equipment using convolutional neural networks (CNNs) and computer vision techniques.

1.1 Scope, Goals and Contributions

With the overall goal of optimizing the washing cycles period in mind, this particular work focuses on the Area of Activity 1, which is crucial for the correct functioning of the decision-making system that will be running on the server. The captured data must be reliable and suited to be used – along with other sources of information – as a feature into a machine learning (ML) model, which is out of the scope of the current work.

It is also out of the scope of this project to prove the any correlation between leakage current in HV insulators and their contamination levels. It is previously assumed that such correlation exists, based on works such as [8].

Moreover, the HV insulators are usually placed in dangerous and hard-to-reach locations, making the remote management of the electronic device developed essential. This involves not only being able to send over-the-air (OTA) device firmware updates (DFUs), but also the ability to modify application parameters in real time, obtain information about all devices and perform remote operations on each of them individually. Therefore, the specific goals of this work can be summarized as follows:

- Design a hardware capable of digitizing signals from the current sensors and storing the digitized data locally until it is transmitted through the network;
- Set up a wireless network capable of covering the area of the target substation and provide enough bandwidth for the data transmission;
- Deploy a local server capable of storing the data collected by the sensors;
- Develop a web platform capable of remotely monitoring and managing the IoT devices.

Different hardware and software components have been developed and integrated as part of an IoT solution in order to meet all the requirements. It must only be emphasized that the first specific goal – hardware for data acquisition – is still under tests and improvements in partnership with researchers from another institution, and it is not the primary focus of the current work.

1.2 Outline

This work is organized as follows. Chapter 2 reviews the fundamental concepts to understand the system developed, concerning IoT architectures, interfacing and application protocols, as well as wireless technologies used for IoT. Then, Chapter 3 describes the different layers of the developed system. Next, Chapter 4 presents and discusses results obtained from different parts of the project. Finally, Chapter 5 concludes this work and points out open issues that could be improved on a second version of the system.

1.3 Outcomes

The following list summarizes the papers produced as part of this project efforts.

Conference Paper:

1. **Pedro Rendeiro**, Jamile Leite, Marcos Silva, Lucas Silva, George Sales and Leonardo Ramalho. *Connectivity Evaluation of ESP32 in Outdoor Scenarios*. Computer on the Beach 2023 (COTB 2023).

Extended Abstract:

1. **Pedro Rendeiro**, Jamile Leite, Marcos Silva, Lucas Silva, George Sales and Leonardo Ramalho. *Performance Evaluation of ESP32 in Outdoor Links*. XL Brazilian Symposium on Telecommunications and Signal Processing (SBrT 2022).

Chapter 2

Theoretical Foundations

The development of an IoT system touches in a range of different areas at the same time, each of them with its own set of theoretical foundations. The aim of this chapter is to present some of the main concepts and terminologies used in this work. First, a brief overview is given regarding IoT definitions (Section 2.1), hardware interfacing protocols (Section 2.2) and network application protocols (Section 2.3). Then, a review of the main wireless technologies used in IoT systems is presented (Section 2.4).

2.1 Internet of Things (IoT)

It is a challenge to find a single definition that satisfies the whole IoT landscape, since there is no universal consensus on the definition of IoT in academic or technical literature [9]. After reviewing more than 120 different definitions across 216 papers by 2021, the authors from [9] indicate that the theoretical concept has not evolved in the same pace as the technology itself, and that organizations such as National Institute of Standards and Technology (NIST) and Institute of Electrical and Electronics Engineers (IEEE) still need to work towards a common definition.

Although there is still a lot of debate regarding this theme, it is important to adopt a definition. In 2012, the ITU Telecommunication Standardization Sector (ITU-T) published a recommendation entitled “Overview of the Internet of Things”, where they define IoT as “a global infrastructure for the information society, enabling advanced services by interconnecting (physical and virtual) things based on existing and evolving interoperable information and communication technologies” [10].

In the same document, they describe a “thing” as “an object of the physical world (physical things) or the information world (virtual things), which is capable of being identified and integrated into communication networks”, and a “device” as “a piece of equipment with the mandatory capabilities of communication and the optional capabilities of sensing, actuation, data capture, data storage and data processing”.

These are the definitions adopted in this work, since they come from an internationally reputable source.

2.1.1 IoT Architecture Model

Still in [10], there is a reference model to describe IoT applications, which can be seen in Fig. 2.1. It consists of four layers, as well as management and security capabilities, spread across all these layers, which will be described in a bottom-up order.

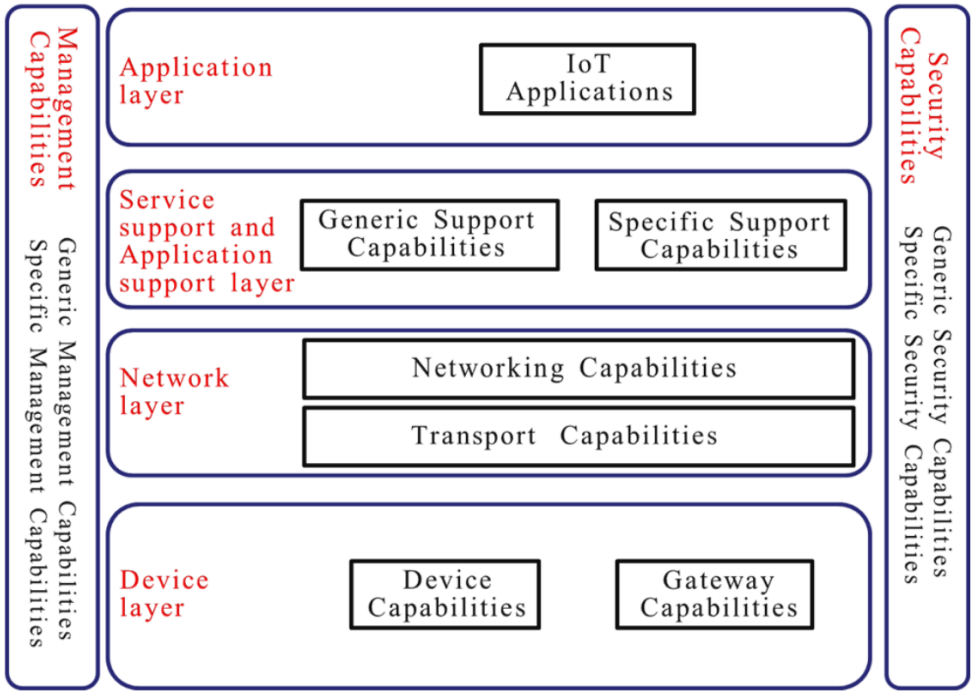


Figure 2.1: IoT reference model from ITU-T.

Source: [10]

Although device layer can be split into device and gateway capabilities, the system designed for this work operates in a single local area network (LAN), there is no need for a gateway. Therefore, only the device capabilities are the listed here, which includes gathering and uploading data from the environment, as well as receiving commands from the network. The

hardware protocols described in Section 2.2 are used in the device layer of this work, which is described in Section 3.1.

Network layer is simpler. It consists of connectivity and transport capabilities, such as access, authentication, transport and mobility management. Wireless technologies like Wi-Fi (IEEE 802.11 b/g/n), Long Range Wide Area Network (LoRaWAN), and cellular networks can be used to in this layer. This along with other of the most common wireless protocols used for IoT are reviewed in Section 2.4. Such review is considered and further discussed in Section 3.2 in order to choose the most suitable technology for this work.

Service support and application support layer – hereinafter called support layer for simplicity – can be grouped into generic and specific capabilities. The first group includes basic functions such as data processing and storage, which are common to various IoT applications. The second group of capabilities can be built on top of the first one, and consists of features that consider the requirements of specific applications. Hypertext Transfer Protocol (HTTP) – described in Section 2.3 – is used to provide communication between the IoT devices and an HTTP server in this work, as detailed in Section 3.3.

The ITU is laconic when describing the application layer, stating only that "application layer contains IoT applications", presumably referring to the high level service provided to the end-user. It is in this layer that the context where the system is applied makes the most difference. In the case of this project, this is the optimization of the washing cycle period of HV insulators, and there must be algorithms designed to do that, as well as adequate end-user interfaces. Since this specific work focuses on the data acquisition part of the IoT solution, there is no application layer to be described.

Management capabilities can include device management, such as diagnostics and firmware updates; LAN topology management; and specific features coupled with application-specific requirements. Finally, security capabilities include mainly confidentiality, integrity, authentication. Information regarding management and security capabilities is spread across Sections 3.1, 3.2 and 3.3.

2.2 Interfacing Protocols

The choice of the interfacing protocols is a fundamental step in a hardware design. In this section, an overview of the two main protocols used in this project, i.e. Serial Peripheral

Interface (SPI) and Universal Asynchronous Receiver/Transmitter (UART).

2.2.1 Serial Peripheral Interface

Introduced by Motorola, SPI is one of the simplest and most used hardware interfacing technologies to communicate between integrated circuits (ICs), specially common on chips like sensors and nonvolatile memory controled by microcontrollers (MCUs) [11]. It works as a "master-slave" protocol and uses only $3 + n$ wires, where n is the number of slaves connected to the master. These wires are: one clock, two data lines and a chip select (Fig. 2.2). They are called SCLK, MOSI (master out, slave in), MISO (master in, slave out) and SS′ (slave select). MOSI, MISO and SS are commonly called SDI (serial data in), SDO (serial data out) and CS (chip select) by the slave ICs.

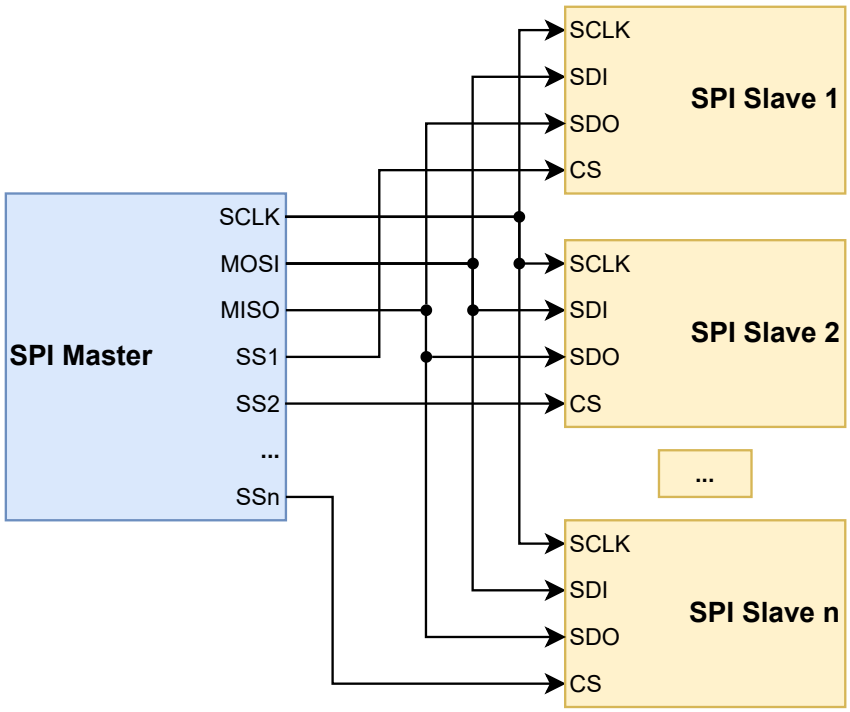


Figure 2.2: SPI protocol model.

This protocol is categorized as full-duplex, since data is transmitted simultaneously in both directions using MOSI and MISO lines. Moreover, it is possible to switch between multiple slave devices using the multiple CS wires and inserting low logic level in the line that should be active and HIGH level in all the others.

Since SPI does not conform to a well-defined standard, it is normally necessary to carefully read the datasheet specifications for each chip [11]. However, many development frameworks already implement application programming interfaces (APIs) to interact with specific groups of devices – such as micro SD cards – so the programmer does not have to deal with all the low level complexities.

2.2.2 Universal Asynchronous Receiver/Transmitter

UART is a hardware serial communication protocol to exchange data between two devices. It is one of the earliest serial protocols, and basically every serial port is UART-based. Although other protocols – such as SPI – have been replacing UART, it is still suitable for lower-speed applications due to its low-cost and easy implementation [12].

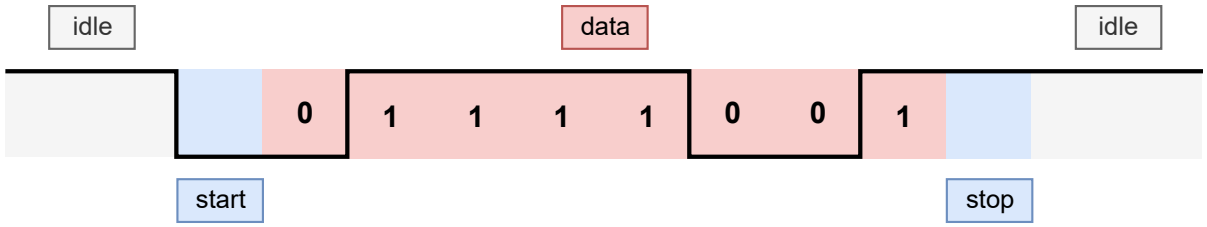
It uses only two wires between the devices, which are normally called RX-TX and TX-RX, and are used to enable full-duplex communication between them. Since the devices do not share a common clock signal – that is the meaning of being an asynchronous protocol – both ends must transmit at the same known speed (baud rate). Typical baud rates are 9600, 115200 and 921600 bps.

In addition, they must use the same frame structure, consisting of start and stop bits, data bits and an optional parity bit. Basically, when no data is being transmitted, the line remains in high logic level. Then the start bit marks the beginning of a transmission by causing a falling edge. Next the data bits are transmitted. Although the number of data bits can vary from 5 to 9, it is very common to use a byte.

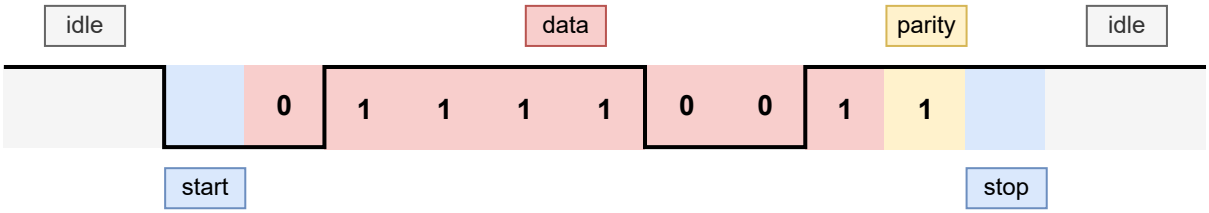
After that, if no parity bit is used, there is the stop bit, marking the end of the transmission by setting a high logic level. It is possible to use two stop bits in order to give more time for the receiver to prepare for the next transmission cycle, but that is seldom used in practice.

Optionally, a parity bit can be used between the data and the stop bits to detect transmission error. It works by counting the data bits whose value is 1, and setting the value of the parity bit to 0 or 1 depending on the type of parity used. The total count of occurrences of 1s (including the parity bit) must be odd if odd parity is used, and even if even parity is used. However, for being extremely simple, this detection method has the intrinsic limitation of being unable to identify errors when an even number of bits are flipped. Hence, the parity bit is seldom used.

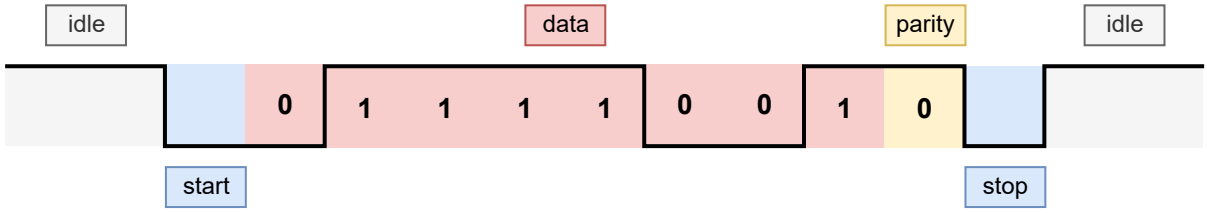
Fig. 2.3 shows the structure of an 8-bit UART frame with a single stop bit and no parity bits **(a)**, even parity bit **(b)** and odd parity bit **(c)**. A pretty common configuration for UART



(a) No parity bit.



(b) Even parity bit.



(c) Odd parity bit.

Figure 2.3: Possible formats of an 8-bit UART frame with a single stop bit.

transmission is 9600 bps baud rate, 8 data bits, no parity bits and 1 stop bit, and it is often shortened to “96008N1”.

2.3 Hypertext Transfer Protocol

It is important that the layers from the ITU model are not mistaken with the TCP/IP layers. For instance, the HTTP is a protocol from the application layer in TCP/IP stack. However, it can be used to exchange information between the IoT nodes and a web platform for management (device and support layers from [10]).

Actually, HTTP is the main protocol used for data exchange in the network design for this work, so it is worth to mention its main characteristics. The latest official document defining HTTP – as November 2023 – is the Request for Comments (RFC) 9110 [13], which is as reference hereinafter.

First, it is important to understand some basic terminology used in this protocol, such

as *resources*, *representations*, *clients* and *servers*. The target of an HTTP request is called a "resource" of undefined nature – it could be a document, an image or a binary file for example. A "representation", in turn, is information intended to "reflect a past, current or desired state of a given resource". It consists necessarily of metadata and optional unbounded stream of data. The stream of data is necessary is case the representation is an executable file, for example.

Furthermore, this protocol follows the "client-server" architecture. That means a "client" program sends HTTP requests to a "server" program, which is responsible for accepting these requests and serving the client with HTTP responses.

The nature of the HTTP requests depends on the method used to send them. There are 8 methods defined in RFC 9110; however, only 2 of those are of interest to this project: GET and POST. The GET method "requests the transfer of a current selected representation for the target resource". The POST method "requests that the target resource process the representation enclosed in the request". The most natural use cases to these methods is to download and upload files, respectively, but it is not limited to that.

Finally, it is important to know the main classes of status codes in HTTP. These are three-digit integer codes that describe the result of a given request. All valid codes are within the range of 100 to 599, and the first digit indicates to which class a code belong. The meaning and the corresponding first digit of each class is resumed in Table 2.1. The last two digits vary depending on the specific cases. Some of these are pre-defined by the RFC, but there is space to define application specific codes as well.

Table 2.1: Classes of the HTTP status codes.

Format	Class	Meaning
1xx	Informational	The request was received, continuing process.
2xx	Successful	The request was successfully received, understood, and accepted.
3xx	Redirection	Further action needs to be taken in order to complete the request.
4xx	Client Error	The request contains bad syntax or cannot be fulfilled.
5xx	Server Error	The server failed to fulfill an apparently valid request.

2.4 Review of Wireless Communication Technologies Used in IoT Networks

IoT-based sensing systems use different types of wireless communication technology, depending on the project requirements. Some of the most common protocols used in IoT networks are LoRaWAN, Sigfox, Narrowband IoT (NB-IoT), Wi-Fi (IEEE 802.11 b/g/n), IPv6 Over Low-Power Wireless Personal Area Network (6LoWPAN) and ZigBee [14, 15, 16, 17].

The following subsections describe each of these technologies, highlighting their characteristics, such as typical range, data rate, energy consumption, payload size and implementation cost.

2.4.1 LoRaWAN

LoRaWAN networks are a type of low power wide area network (LPWAN) and have been specially designed for transmitting small amounts of data over long distances with low energy consumption. Use cases include precision agriculture, smart cities, remote environmental monitoring, infrastructure monitoring and smart grids. All these applications require the measurement of variables a few times a day, such as temperature, humidity and energy consumption. Particularly, LoRaWAN is operated over the Long Range (LoRa) physical layer, which transmits data in unlicensed or industrial, scientific and medical (ISM) frequency bands, such as 868 MHz in Europe, 915 MHz in America and 433 MHz in Asia [18].

LoRa modulation uses spread spectrum techniques and has several advantages such as long range signals, good immunity to interference and the existence of low power modes. It is estimated that LoRa devices can have around 10 years of battery life and can implement point-to-point wireless communication over a distance of several kilometers. However, in order to achieve these characteristics of low energy consumption and long range, LoRa transmissions operate with low data rates ranging from 0.3 to 50kbps [19].

LoRaWAN networks are organized in a star-based topology, in which sensor nodes send their data on the uplink (UL) using LoRa, which can be received by one or more gateways in a single wireless hop. The gateways, in turn, forward the LoRa data to a LoRaWAN network server via a traditional IP network, such as cellular, Ethernet (IEEE 802.3), satellite or Wi-Fi. In this case, much of the complexity is placed on the network server, which is responsible, for example, for eliminating any duplicate data received by more than one gateway [18]. Com-

munication is bidirectional, although UL communication from the sensor nodes to the network server is strongly favored [19].

Data packets contain a payload that can vary in size, with a maximum of 243 bytes [14]. Although LoRaWAN's performance is mainly determined by its medium access control (MAC) layer, duty cycle regulations in the ISM bands emerge as a key limiting factor. The duty cycle in LoRaWAN can vary between 0.1% and 10% depending on the network server and the regulations at the wireless network installation site, which considerably limits the system's availability for sending periodic data.

The cost of a wireless network can be broken down into several factors, such as the cost of the end device, the cost of implementing the network infrastructure and the cost of allocating the spectrum. As the spectrum used in LoRaWAN is unlicensed, the cost of implementing such a network would only be the network infrastructure, with gateways costing more than €100 EUR, and the end devices, which are estimated at between €3 EUR and €5 EUR [14].

Finally, it is worth noting that the LoRaWAN network is open and can be implemented using open software or software developed by those responsible for the network. A technology similar to LoRaWAN in terms of range, data rate and consumption is the Sigfox network, which is described in Section 2.4.2.

2.4.2 Sigfox

The Sigfox network was developed with a software-based connectivity solution, where all the computing and networking complexity is managed in the cloud, rather than on the devices. This drastically reduces energy consumption and device costs.

Sigfox uses Differential Binary Phase-Shift Keying (D-BPSK) modulation for which the message has a fixed bandwidth of 100 Hz and is sent at a rate of 100 bps (for Europe) or 600 bps (for the USA), within an unlicensed frequency spectrum below 1 GHz, for the 868 MHz European region and 915 MHz for the US region [20]. This modulation technique is part of the ultra-narrowband (UNB) modulation type, which requires low power consumption to ensure connections between the nodes and the base station.

The advantages of using D-BPSK modulation are that it provides highly efficient access to the middle of the spectrum and is easy to implement. A low bit rate allows the use of low-cost components in the transceiver part. In these networks, a star-based topology is used, where the end devices are connected directly to a base station. The network architecture is thus simplified

and the centralization and sending of information to the Internet is done within a well-defined frame at the level of a physical device (e.g. Base Station or Gateway). Sigfox communication technology was created to connect large tracts of land, in the order of tens of kilometers in the countryside and a few kilometers in urban areas. The gateway modules can be implemented with software-defined radio (SDR), and the network is usually deployed in partnership with mobile operators.

Initially, Sigfox maintained only UL transmission, but then moved on to bidirectional communication. The downlink (DL) transmission only takes place after an UL transmission. The number of messages on the UL is limited to 140 messages per day. The maximum payload length for each UL message is 12 bytes. However, the number of messages on the DL is limited to 4 messages per day, so all messages on the UL may not be recognized. The maximum payload length for each DL message is 8 bytes [14].

Finally, the network costs can also be divided into three: the cost of the end-device, the cost of the network infrastructure and the cost paid for the Sigfox private network. The cost of the Sigfox device can be as low as €2 EUR and the cost of the base station can exceed €1000 EUR [14]. Unlike LoRaWAN networks, Sigfox charges fees for using the network [21], after installing a base station with local partners. At the time of writing, the "UTE Porto de Sergipe I" area was not covered by any Sigfox networks, as shown on the Sigfox network coverage page[1].

2.4.3 NB-IoT

The NB-IoT is a network technology that uses the infrastructure of mobile networks and provides long range in exchange for a reasonable data rate and moderate energy consumption. The NB-IoT specifications were defined by the 3rd Generation Partnership Project (3GPP), initially released in 2016 and updated in 2017 and 2018 through Releases 13, 14 and 15 respectively. Unlike other technologies, this one has already emerged within the IoT context, in response to market demands [22].

One of the main attractions of NB-IoT is its great range, which can be as great as 1km in urban areas or 10km in rural areas [14], which – combined with the possibility of using existing mobile network infrastructure – allows the development of products with almost global connectivity. In order to facilitate the mobility of a product around the globe, the use of embedded-SIM (eSIMs) instead of conventional physical cards has become widespread. In Brazil, the

[1] Available at: https://www.sigfox.com/en/coverage.

operators Claro and Tim support NB-IoT, and the two together offer this technology in 4809 cities in Brazil by September 2023[2]. Another advantage of this network is the maximum payload size of 1600 bytes [23].

Although it is a promising technology for many IoT applications, NB-IoT has some characteristics that need to be considered, such as its maintenance cost, because although the modules can be as cheap as $2 USD [24], it is necessary to maintain a data plan with a telephone operator for connectivity to be provided. Another concern is energy efficiency. Typically, NB-IoT devices consume less energy than devices that connect to traditional mobile networks; even so, they consume more energy than technologies such as ZigBee or LoRaWAN [24]. Furthermore, the low transfer rate (180 and 200kbps for UL and DL respectively) and latency of up to 10s for UL may make it unfeasible to use NB-IoT in critical system projects or those involving medium or high data traffic [23].

2.4.4 Wi-Fi (IEEE 802.11)

The IEEE 802.11 standard, popularly known as Wi-Fi, is a set of physical and link layer protocols widely used in the context of wireless local area networks (WLANs) operating in unlicensed 2.4GHz or 5GHz bands [25]. The first version of this technology was launched in 1997 and due to its success in the market and versatility, many derivative specifications have been produced over time [26]. Some Wi-Fi specifications widely used in IoT devices are IEEE 802.11b, IEEE 802.11g and IEEE 802.11n, which have maximum data rates of 11, 54 and 600 Mbps, respectively [27], with typical ranges of 140m [25], but which can be considerably longer depending on the installation conditions, antennas, power and environment [28].

Among the main positive points of this communication technology, some are worthy of note. The first is the high availability of devices and its low cost, which makes it easy and quick to deploy and test a Wi-Fi network with different equipment. In addition, it provides high data rates at a reasonable range, which can exceed the values mentioned above depending on the transmitter and receiver used. A final notable feature is the payload size, which can be up to 2312 bytes [29].

Although Wi-Fi has many attractive features, its high energy consumption is a concern in IoT projects. However, there are several modules available on the market today that make

[2]According to a report by Teleco Portal, available at: https://www.teleco.com.br/lpwa_cobertura.asp.

use of techniques to optimize the energy efficiency of this technology [15]. Furthermore, consumption in mW is not the only metric to keep in mind when assessing the energy efficiency of a network technology. It is possible to complement the analysis from the point of view of mJ/Mb. From this point of view, in applications that require high data rates, IEEE 802.11 is capable of outperforming ZigBee by more than 20 times in terms of energy efficiency [29].

2.4.5 6LoWPAN

6LoWPAN is a set of standards defined by Internet Engineering Task Force (IETF), which creates and maintains all major Internet standards and network architecture work. The 6LoWPAN standards enable the efficient use of IPv6 in low-power, low-rate wireless networks in simple embedded devices by means of an adaptation layer and the optimization of related protocols. This technology allows IPv6 to be implemented on top of the IEEE 802.15.4 MAC and physical layer protocols, so that together they can implement low-power networks capable of carrying IP packets from small devices, including sensors, controllers and other IoT devices.

This technology has a typical range of up to 200 meters, with the maximum data rate varying depending on the system's operating frequency: 250kbps at 2.4GHz, 40kbps at 915MHz and 20kbps at 868MHz [30]. According to a survey carried out by researchers, the range of current consumed is 20 to 35mA for data reception and 2 to 25mA for data transmission [16]. The size of the payload can vary based on the compression that is done on the packet header: without compression, the payload can be 53 bytes and with compression, the payload size can be 108 bytes [31].

The 6LoWPAN physical layer, implemented through IEEE 802.15.4, operates on unlicensed frequencies. Therefore, the cost of implementation would be split between the end nodes and the equipment to build the network infrastructure. In this case, there is a need to build the border router that implements the bridge between the 6LoWPAN network and other types of networks. The end devices can be built with microcontrollers that have an IEEE 802.15.4 interface, and such devices can cost around $5 USD. On the other hand, the border routers can be implemented in various ways, for example by combining hardware that has IEEE 802.15.4 support with a single board computer, such as the BeagleBone Black [32] or Raspberry Pi [33], which can be found at costs of $40 USD.

2.4.6 ZigBee

ZigBee networks also use the IEEE 802.15.4 protocol as the MAC and physical layer. As such, the network also operates on unlicensed frequencies. The network, security and application layers of this technology are defined by the ZigBee Alliance. The power required to implement ZigBee in embedded systems is generally very small, in most cases using 1mW or less [17]. But it still provides a range of up to 150 meters outdoors, which is achieved by a technique called direct-sequence spread spectrum (DSSS).

In addition, it is possible to implement a mesh network with ZigBee nodes, which would allow the network's range to be increased. The payload for a single packet can be up to 68 bytes, and when the application needs to use more data, ZigBee fragments the message into several packets [34]. The cost of implementation is generally considered to be less than $0.1 USD / foot (0.3048 meters) [35]. It is worth noting that, in this case, a gateway is also required to take the data from the ZigBee network and forward it to external networks and vice versa.

Chapter 3

System Overview

As stated in the Introduction, the goal of this work is to develop an IoT system capable of acquiring leakage current information from HV insulators in order to optimize the washing cycles period. This system comprises multiple IoT devices that must be able to periodically digitize data from current sensors and send it to a local server through a wireless network. The devices must also be able to save the data in a micro SD card in case of a network failure, and transmit them once the wireless connection is reestablished. Finally, there must also be an IoT web platform capable of remotely managing the multiple nodes, which includes OTA DFUs, as well as the possibility to troubleshoot issues in specific devices without impacting the other IoT devices in the network.

Fig. 3.1 shows how such model can be applied to the current work. Device layer is basically comprised of the current sensor, the signal conditioning module, the communication module and the micro SD card shared by both modules. The network layer, in turn, includes a Wi-Fi (IEEE 802.11 b/g/n) network infrastructure, containing access points (APs) to serve the remote nodes. At the other end, there is the local HTTP server, representing the whole service/application support layer.

In this context, this chapter provides an overview of the whole IoT system developed to attend these requirements. The IoT architecture reference model from ITU-T – explained in Section 2.1.1 – is used here. Device, network and support layers are presented in Sections 3.1, 3.2 and 3.3. Results for all of these parts of the system are presented in Chapter 4.

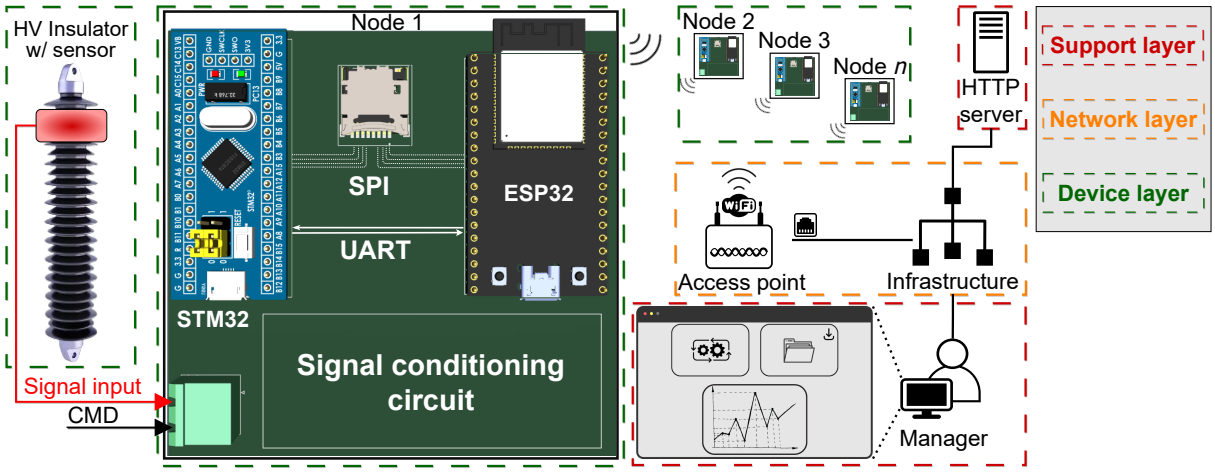


Figure 3.1: Overview of the IoT system.

3.1 Device Layer: The bridge between analog and digital domains

This section gives detailed explanation on the device layer developed in this project. The focus is on the device capabilities mentioned in Section 2.1.1, such as gathering and uploading data from the environment, as well as receiving commands from the network. The device under consideration is called Microprocessed Unit for Leakage Current Sensors (UMSCF – *"Unidade Microprocessada para Sensores de Corrente de Fuga"*), whose main function is to digitize current leakage signals and make sure that the sampled data gets sent to a local platform.

First, Section 3.1.1 describes the hardware design, going through the schematics to present the several parts of the circuit. Next, Sections 3.1.3 and 3.1.4 detail the specification of the embedded software (firmware) developed for the signal conditioning and communication modules of the UMSCF, respectively. Furthermore, Appendix A presents an operation guide of the hardware assembled on the printed circuit boards (PCBs), focusing on listing the components, giving warnings and instructions on how to mount it, as well as describing its operation.

3.1.1 Hardware Design

3.1.1.1 Big Picture of the Hardware

This section provides information about the IoT device hardware, which is composed of three main parts, as shown on Fig. 3.2: i) the current sensor that captures the current that passes over the HV insulator; ii) a data acquisition (DAQ) PCB responsible for conditioning the

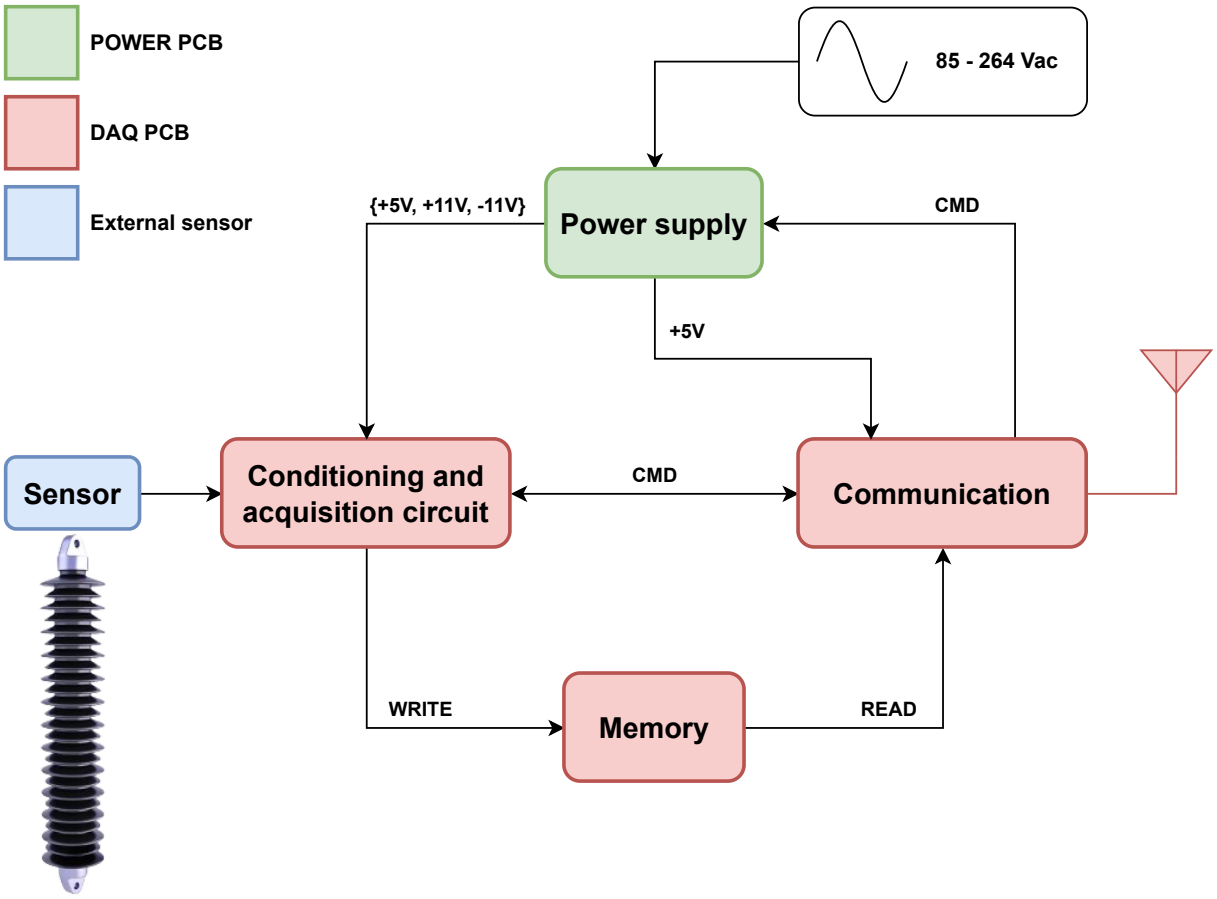


Figure 3.2: Big picture of the hardware design.

signal from the sensor, digitizing the signal, storing it in a local memory and providing wireless communication capabilities; iii) a power supply PCB, responsible for creating the required direct current (DC) voltages for the DAQ PCB, from an alternating current (AC) supply.

In the following sections, both PCBs are described in detail. The conditioning circuit is also described briefly. Nevertheless, it is out of the scope of this work to discuss the conditioning circuit in depth, as well as the sensor itself.

3.1.1.2 Data Acquisition PCB

The schematic of the electronic data acquisition project is divided in parts this Section, but can be found entirely in Appendix B. The DAQ PCB is composed of three main modules, as shown on Fig. 3.3: the conditioning and acquisition circuit, the micro SD card and the communication module.

The signal conditioning and the communication modules use the SPI protocol to write data into micro SD card and to read from it. The communication module selects which mod-

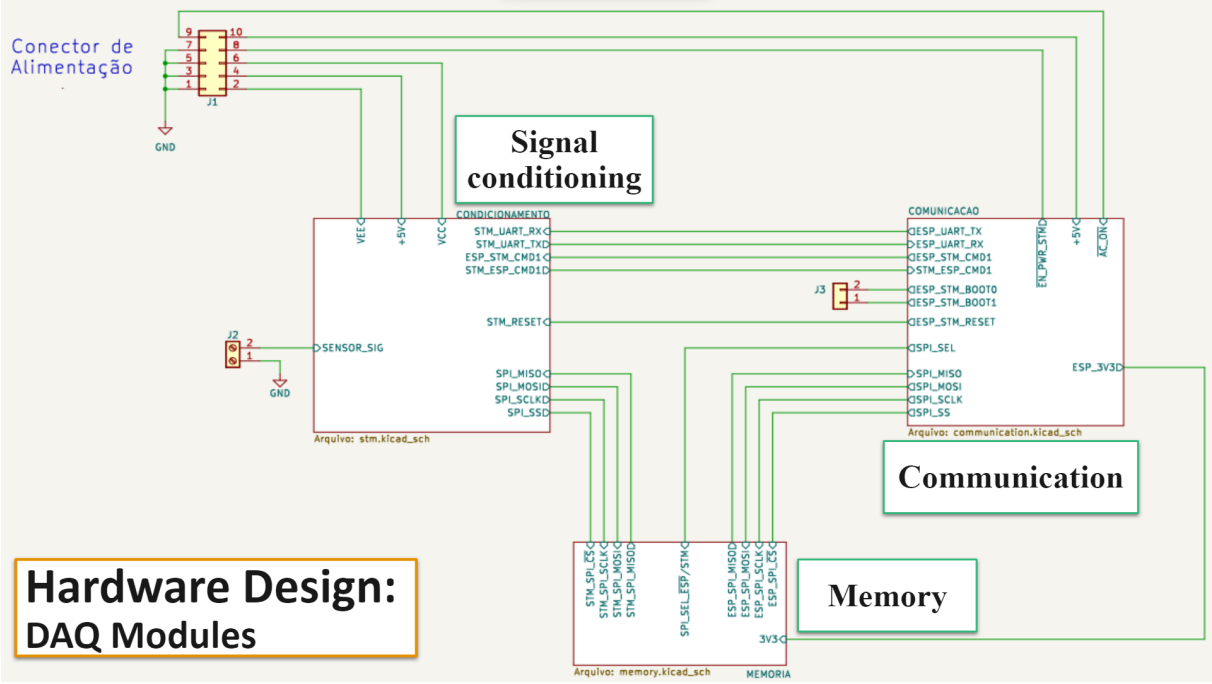


Figure 3.3: Schematic of the PCB with the hierarchical modules of the DAQ PCB.

ule can access the memory through the `SPI_SEL` signal. In addition, there are several UART and general purpose input/output (GPIO) signals between the communication and conditioning modules, so that the former can configure the latter. The sensor signal is provided to the conditioning module through connector `J2`, and the data captured by this module is stored directly in the memory, which can then be read by the communication module and sent to the remote server via an Wi-Fi (IEEE 802.11 b/g/n) wireless network of the project.

All the electrical power for this PCB is provided by connector `J1`, which also has the signals `EN_PWR_STM` and `AC_ON`, where the first one is supplied by the communication module to control the power supply of the conditioning and digitization part, and the second one comes from the power board to indicate if AC voltage is available. Both signals are active low. The conditioning, communication, and memory modules are detailed below.

At first, it must be emphasized that this module is still under tests and improvements in partnership with researchers from another institution, and that it is not the focus of this project.

it must be The conditioning module – presented in Fig. 3.4 – is composed of a series of signal conditioning circuits and a microcontroller responsible for frequency adjustment, amplification, filtering, level adjustment, and analog-to-digital conversion of the leakage current signal.

This circuit has a protection circuit at the input, followed by an amplifier, frequency re-

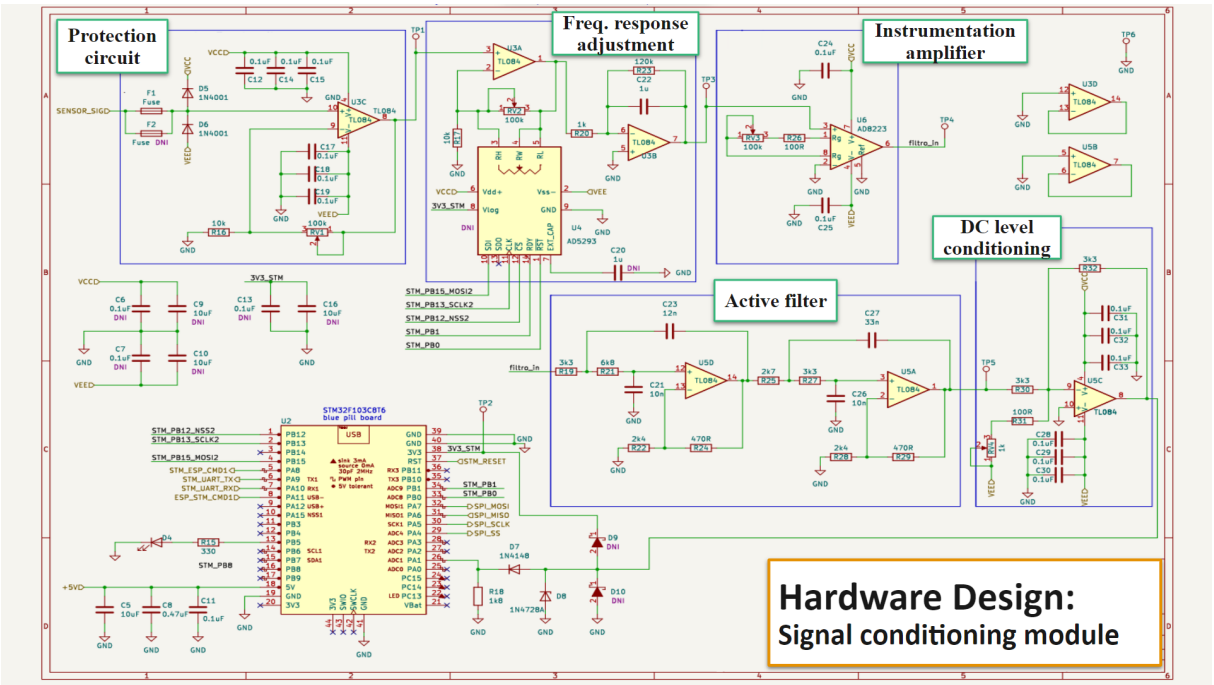


Figure 3.4: Schematic of the signal conditioning module.

sponse adjustment circuit and gain provided by an instrumentation amplifier. The amplified signal is then filtered, and the voltage level is adjusted to be within the dynamic range of the analog-to-digital converter (ADC) of the STM32F103C8T6 MCU. This MCU then digitizes the signal and stores it on the micro SD card through the SPI interface. The `U4` IC is a digital potentiometer that can replace the analog potentiometer `RV2`. The operational amplifiers (AmpOps) `U2D` and `U3B` are components present in the AmpOps packages, but were not used in the project, so they were connected as voltage followers with an input and output in the linear region of the device to avoid noise and high power consumption of the integrated circuit.

The schematic of the communication module is shown in Fig. 3.5, which consists of the ESP32 DevKit, some LEDs, and connections to the signals already presented. It is responsible for orchestrating the data acquisition system of the UMSCF.

Finally, Fig. 3.6 shows the schematic of the memory module, which is responsible for writing/reading the collected signal as well as any persistent data that should be stored even when power is off. It is composed of a multiplexer and a micro SD card connector. The multiplexer is controlled by the ESP32 MCU using the `SPI_SEL` signal, and it determines whether the STM32 MCU or the ESP32 MCU will have access to the micro SD card via SPI interface, in order to avoid conflicts.

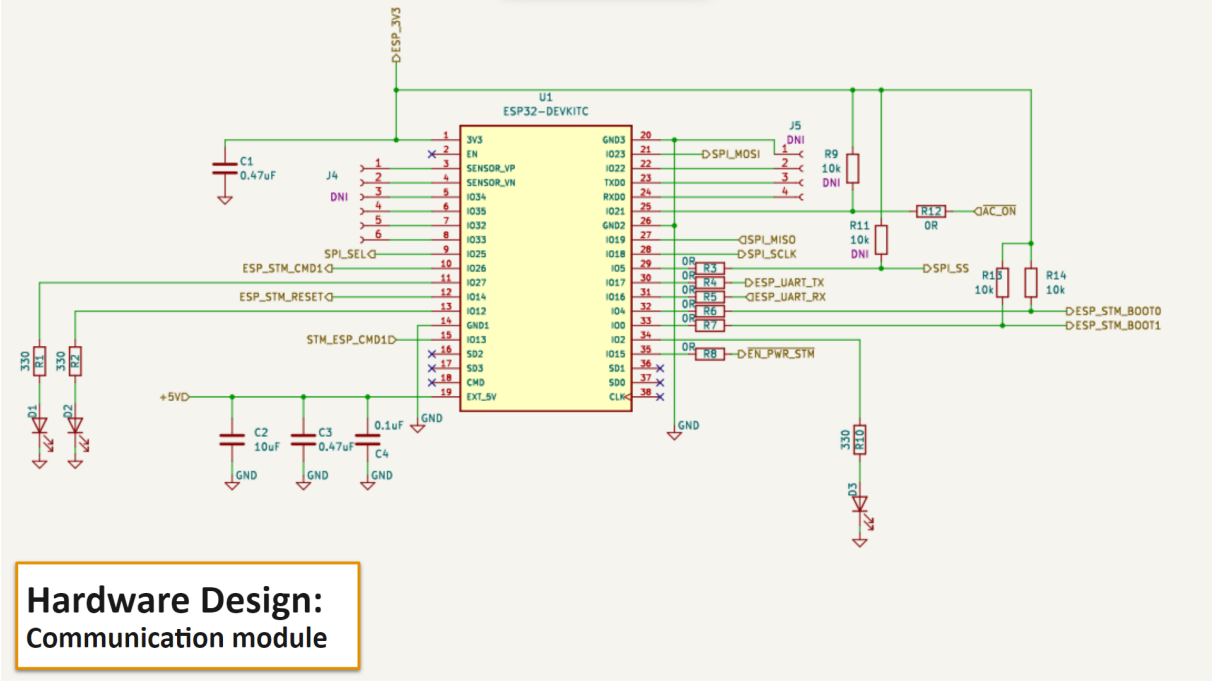


Figure 3.5: Schematic of the communication module.

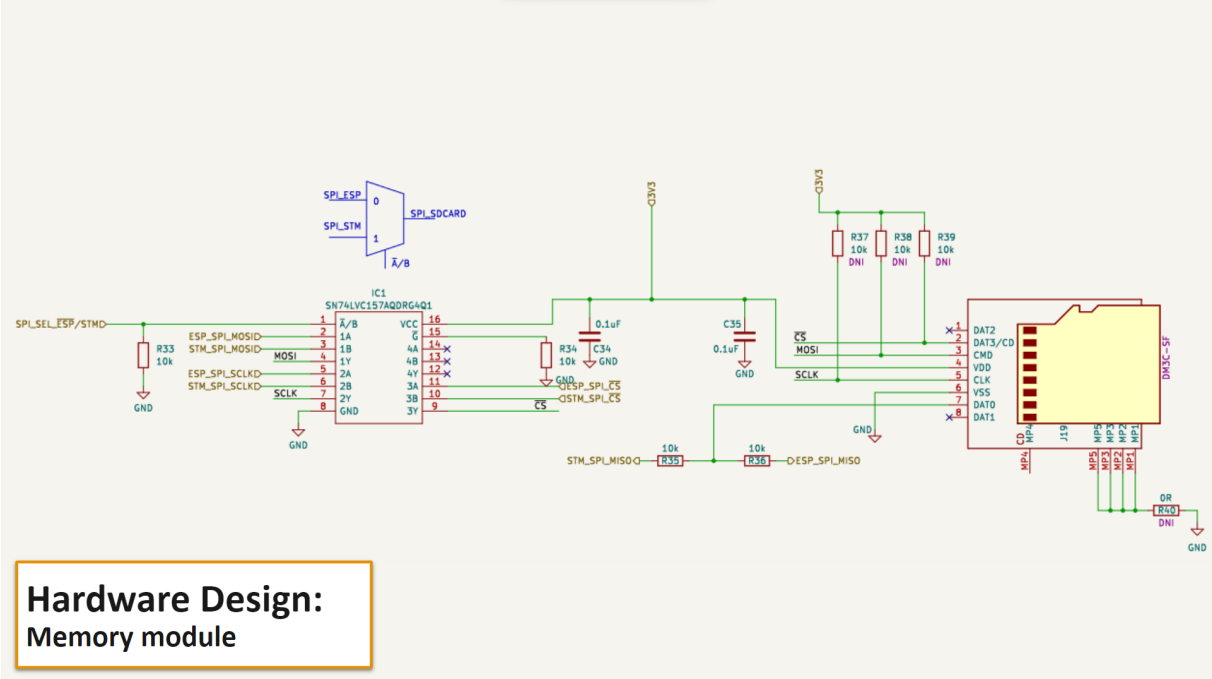


Figure 3.6: Schematic of the memory module.

3.1.1.3 Power Supply PCB

The schematic of the power supply PCB project can be found in its entirety in Appendix C. Its requirements include receiving an AC voltage and converting it into DC signals with the

current required by the DAQ PCB. With the help of the block diagram in Fig. 3.7, it is possible to visualize at a high level the operation of this project. Each block represents an IC responsible for a different stage of the power supply.

The power supply is divided into two stages. The first stage is composed of an AC/DC module, two regulators, and supercapacitors. This circuit converts an AC voltage between 84 and 264V to +12V and -12V (Fig. 3.8) and charges a set of supercapacitors to provide autonomy to the system in case of a power grid failure. The AC voltage is inputted into the board through connector `J1`, and the fuse `F1` along with varistor `RV1` provide protection to the system. The IC `PS1 (RAC20-12DK)` is responsible for performing the AC/DC conversion.

Due to the requirements of the available supercapacitor models for purchase, it is necessary to reduce the voltage to a level supported by these components; in this case, +8.3V and -8.3V. This reduction is achieved by regulators `U1 (LM317)` and `U2 (LM337L)`, positioned between the AC/DC module and the supercapacitors. The positive part has two supercapacitors in parallel, totaling 180F, and can be seen in Fig. 3.9. The negative part is analogous to this, but with only one of 15F, as the energy consumption related to the negative voltage of the circuit is considerably lower. Connectors are placed in parallel with the supercapacitors to enable the connection of other types of supercapacitors if necessary.

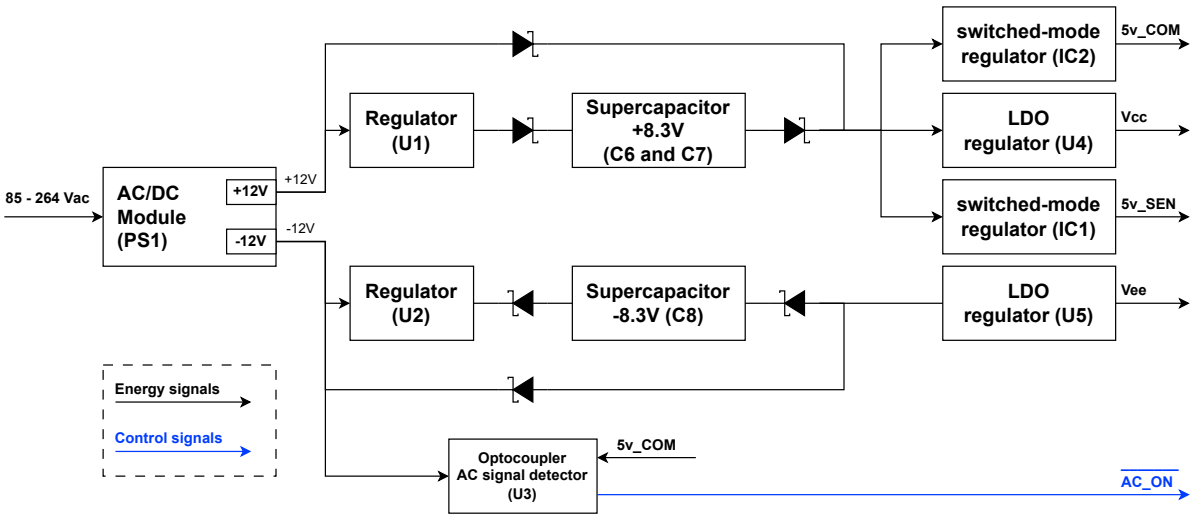


Figure 3.7: Block diagram of the power supply board.

The second stage of the power supply is responsible for regulating the voltage to the levels required by the data acquisition electronics. It consists of two switched-mode regulators `(TPS562207DRLR)` and two linear regulators of the low dropout (LDO) type. The LDO regulators will supply power to the signal conditioning circuits, providing lower noise voltage at their

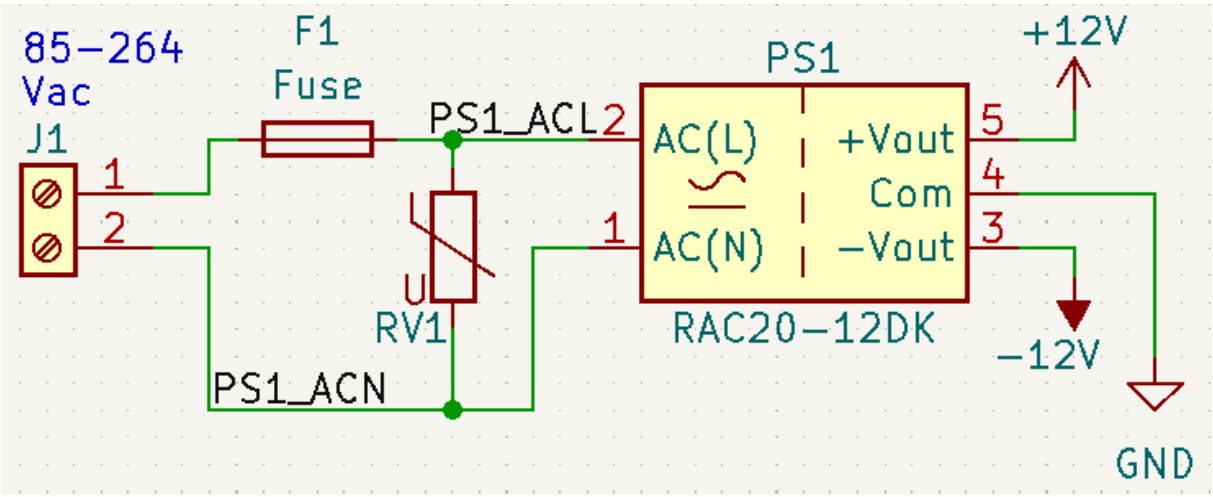


Figure 3.8: First stage of the power supply, converting AC into DC voltages.

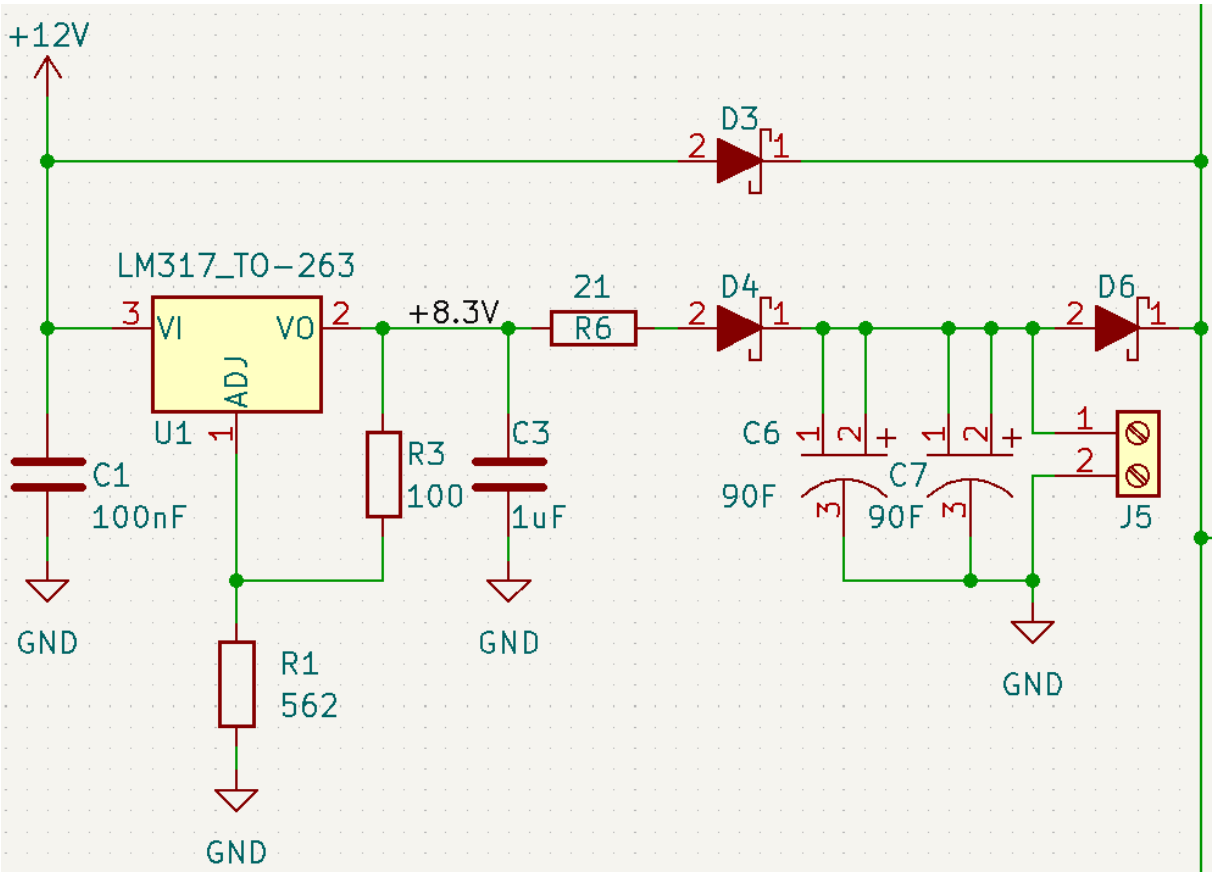


Figure 3.9: Supercapacitors and charging circuit for the positive supply.

output (when compared to the switched-mode regulators) and also being able to operate with low voltage differences between the regulated output and the regulator's input. This last characteristic allows the circuit to operate even if the regulator's input is supplied by supercapacitors that have their voltage reduced over time.

One of the switched-mode regulators (IC2) is used to provide +5V to the MCU responsible for the communication system (Fig. 3.10). For signal conditioning and digitization, the voltages are supplied: Vcc by the LDO U4 (ADP7142ARDZ), Vee by the LDO U5 (ADP7182AUJZ), and +5V by the switched-mode regulator (IC1), as shown in Figs. 3.11 and 3.12. The reason for having two independent power sources for the two MCUs is to avoid noise between the two powered parts.

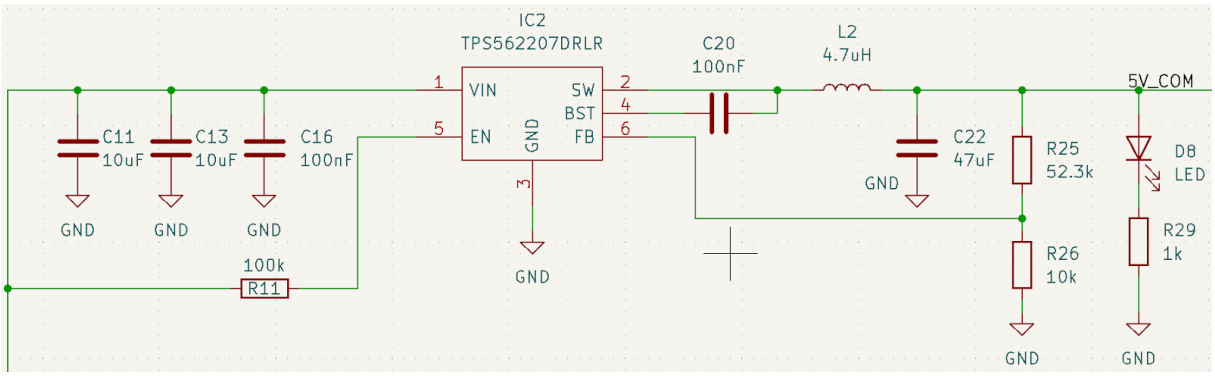


Figure 3.10: Project of the power supply that converts 12V into 5V for the communication system (5V_COM).

Finally, the optocoupler U3 (SFH617A-1X007T) is used as an AC signal detector, generating a control signal that indicates whether the system is being powered by the electrical grid or by the supercapacitors (Fig. 3.13). In the first case, AC_ON is kept at a low level. Otherwise, the signal is brought to a high level. This allows the MCUs to adjust their power-saving policy according to the available power source.

All regulators have LEDs on their outputs to visually indicate the operation of each regulated voltage. Additionally, the output voltages of all regulators are configured with resistors, which allows for changing their output voltages if necessary. Finally, the voltages and control signals of the power board will be connected to the data acquisition PCB through connector J4, as shown in Fig. 3.14. In the connector, all the voltages used by the signal acquisition board are present (Vcc, Vee, and +5V for signal conditioning, and +5V for communication) along with two control signals: AC_ON, which informs the signal acquisition board whether AC voltage is

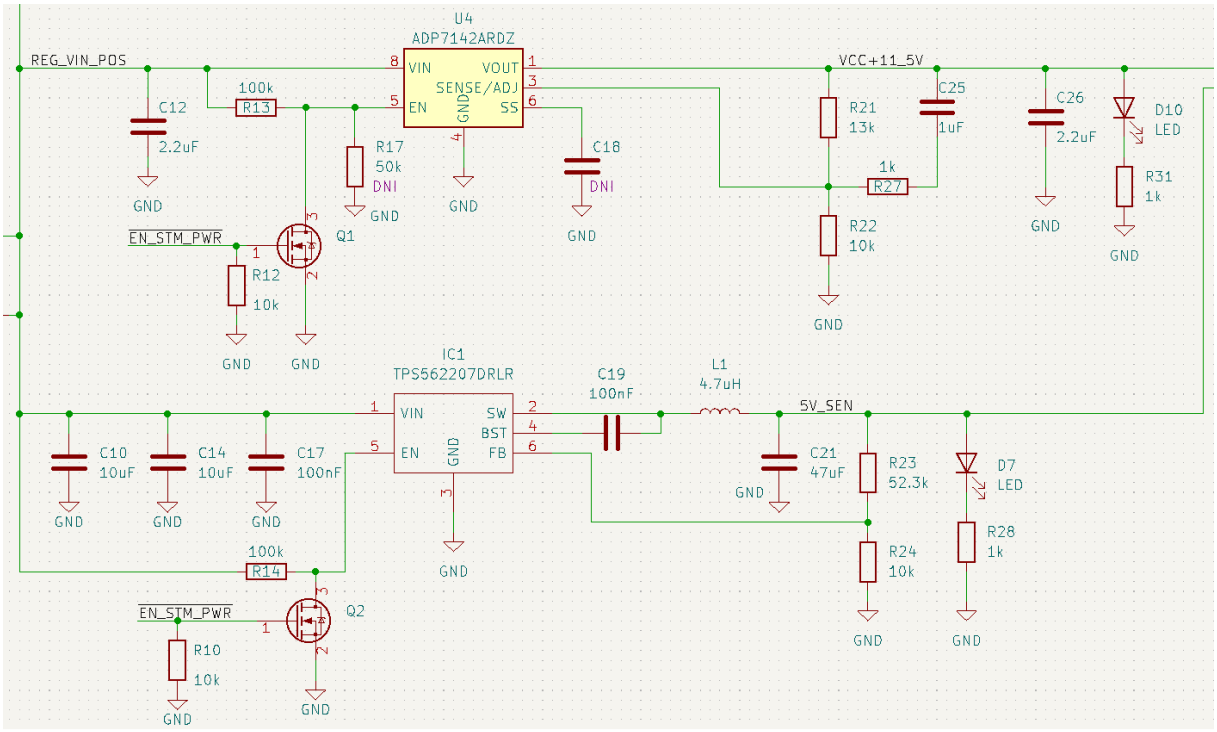


Figure 3.11: Regulators that generate the supply voltages Vcc and 5V for the electronic circuit responsible for signal conditioning and digitization.

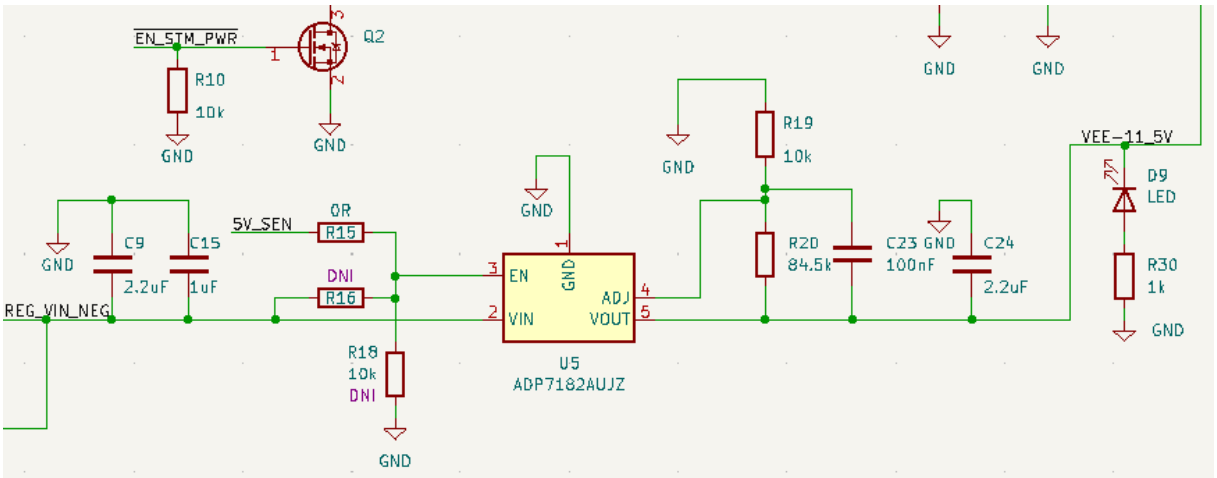


Figure 3.12: Regulator that generates the supply voltage Vee for the electronic circuit responsible for signal conditioning and digitization.

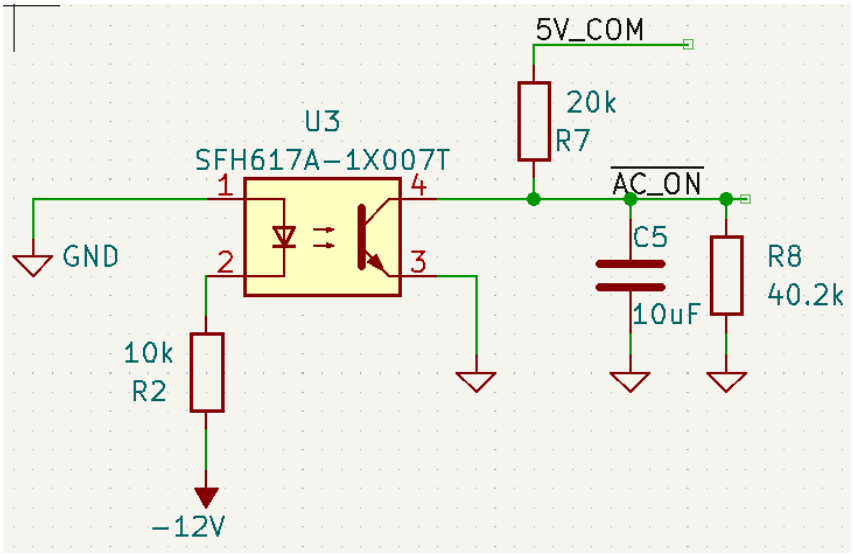


Figure 3.13: Optocoupler that provides an AC_ON signal to indicate whether there is voltage in the AC line or not.

available or not, and the `EN_STM_PWR` signal that comes from the data acquisition board and allows for turning off all power to the signal conditioning part (Vcc, Vee, and +5V).

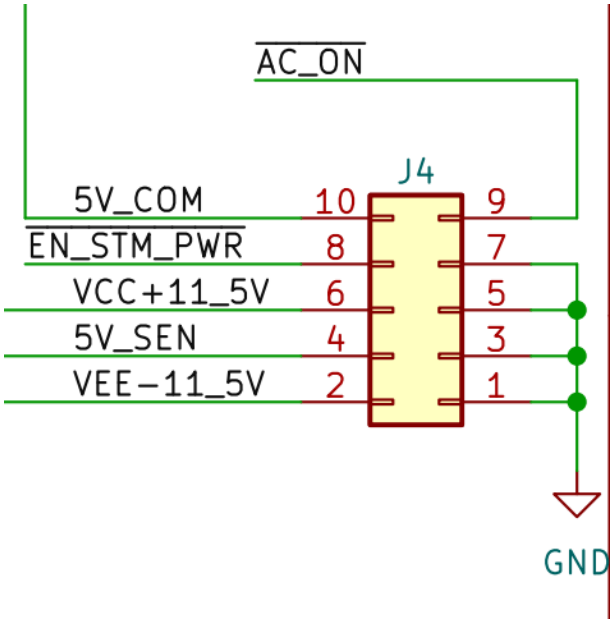
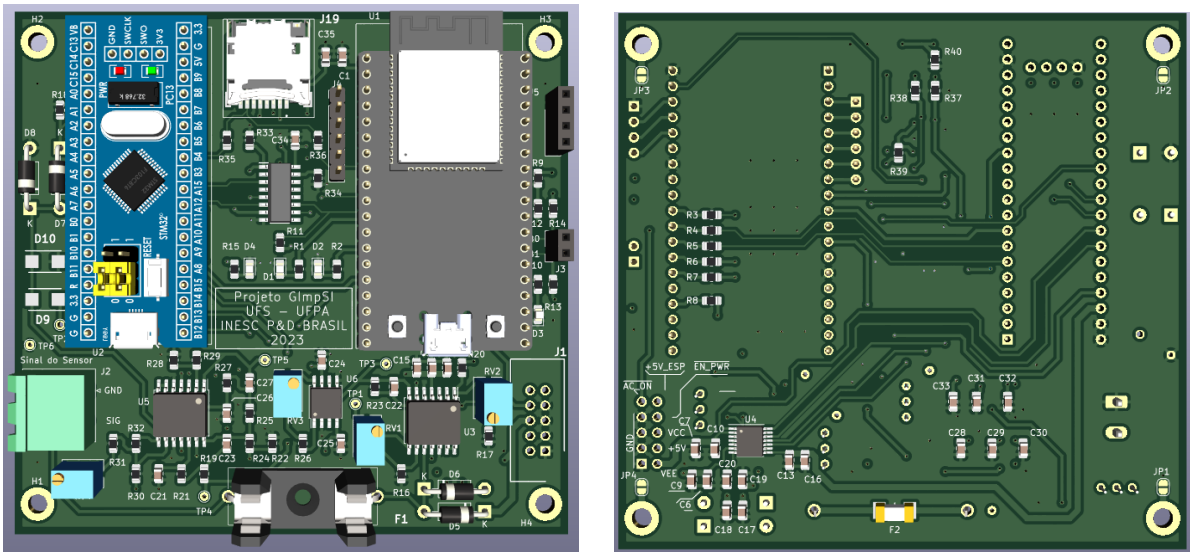


Figure 3.14: Connector to supply the voltages and control signals from the power board.

3.1.2 UMSCF Assembled Hardware

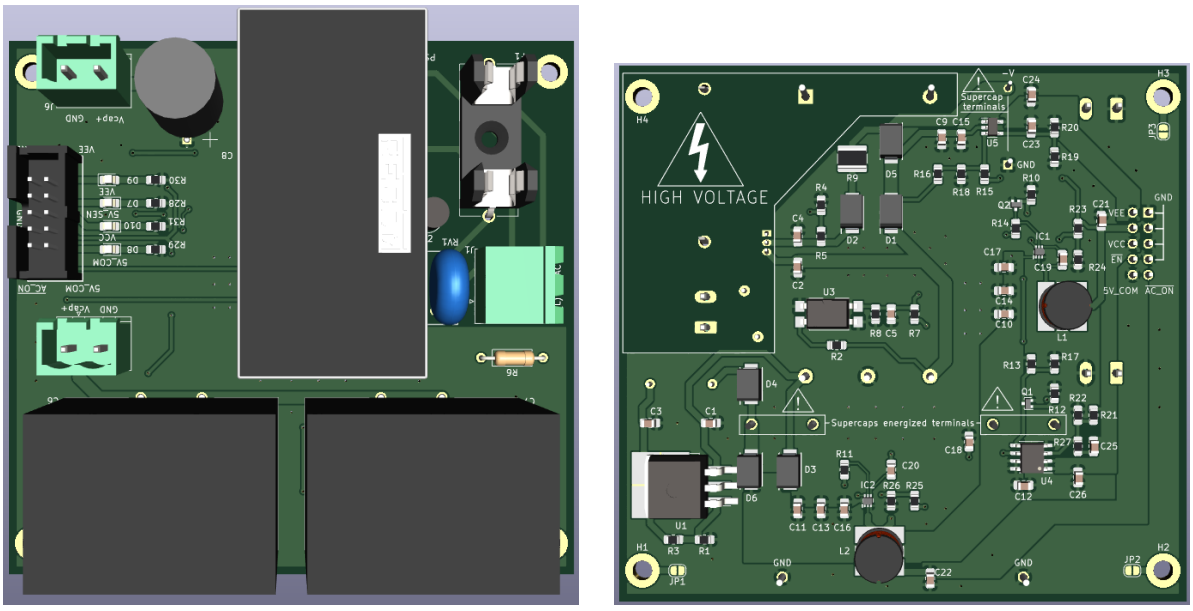
The complete UMSCF system was developed based on the design presented in Section 3.1.1. The resulting 3D models of DAQ and POWER PCBs can be seen in Figs. 3.15

and 3.16, respectively. They were generated using KiCAD[1] softawre. Fig. 3.17, in turn, shows the complete UMSCF assembled hardware.



(a) Top layer. **(b)** Bottom layer.

Figure 3.15: 3D model of the DAQ PCB.



(a) Top layer **(b)** Bottom layer

Figure 3.16: 3D model of the POWER PCB.

[1]Available at: https://www.kicad.org/.

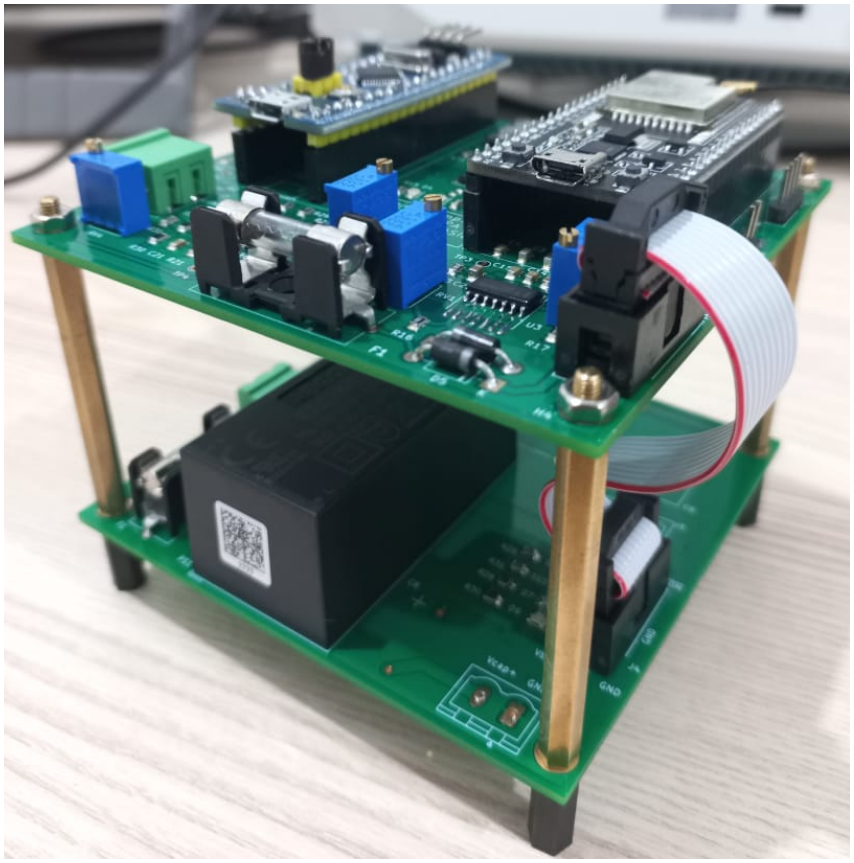


Figure 3.17: 3D Models and Assembled Hardware.

3.1.3 Embedded Software for the Signal Conditioning Module

The signal conditioning module – controlled by the STM32 MCU – has the following responsibility: taking samples from the current sensor and storing them into the micro SD card, shared by both STM32 and ESP32 MCUs. This procedure can be modeled by the following flowchart diagram shown in Fig. 3.18. When the module is powered up, it simply runs initial configurations (setup).

Next, it waits for the "begin sampling" message from the communication module to start a new sampling window. Such message follows the format represented in Fig. 3.19. The characters marked with asterisks *("*")* are used to separate the different parts of the message. Each of these parts is considered in the paragraphs below.

The regular expression "$\backslash d\{1,6\}$" after the "s" represents the number of samples, which can be up to 6 digits from 0 to 9, while "$\backslash d\{1,6\}$" after the "p" represents the sampling period in microseconds and can be represented with up to 4 digits from 0 to 9. In addition, the binary file must be stored in the *"samples"* folder of the micro SD card and its name must

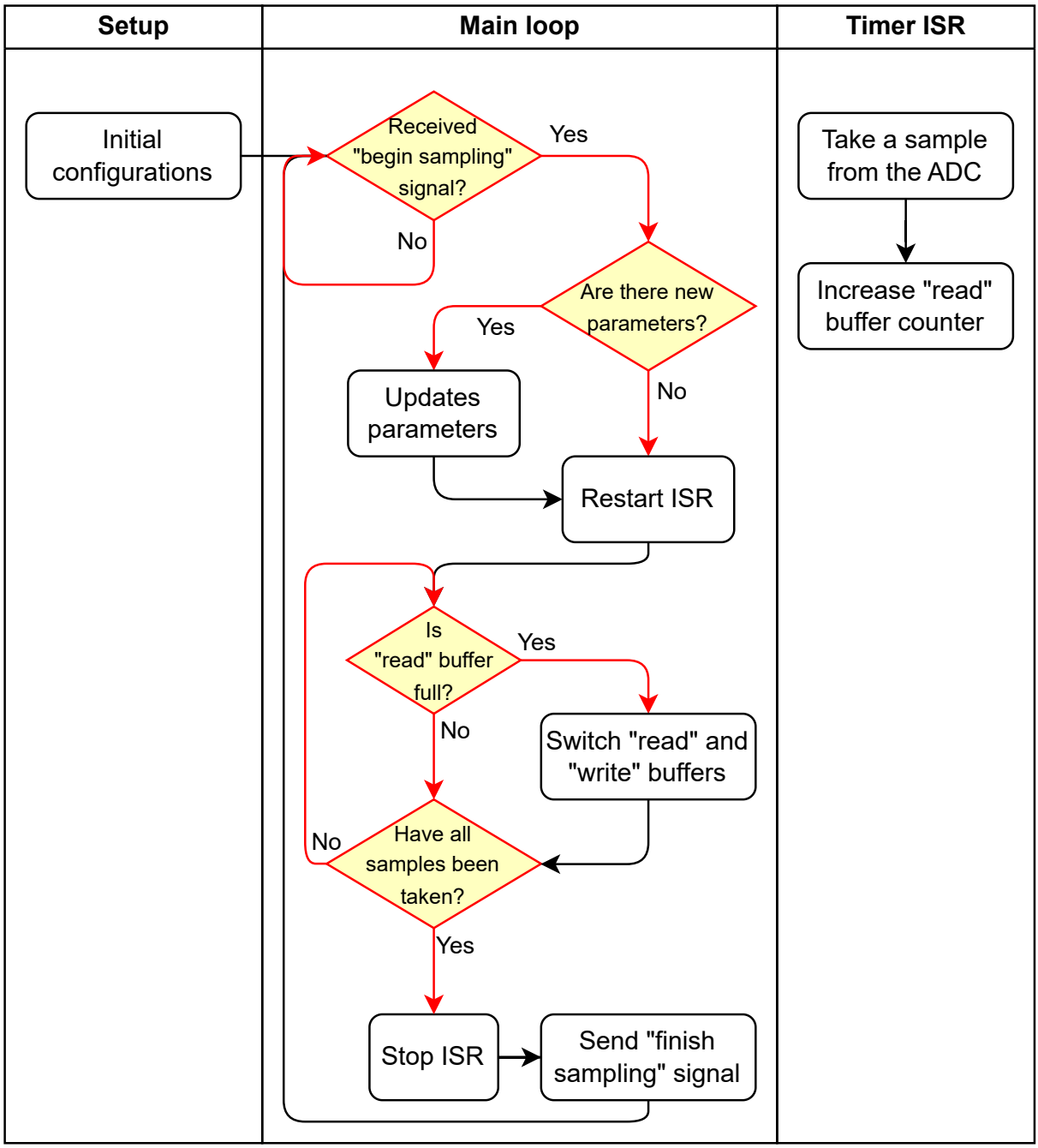


Figure 3.18: Flowchart representing the sampling process.

match the timestamp provided by the real time clock (RTC). The hash symbol *("#")* is used to signal the end of the message to the signal conditioning module. For example, the message "s100000p125-/samples/20230422_131600-01.bin#" informs the signal conditioning module that 100k samples must be captured using a sampling period of 125µs and the captured samples must be stored in the file "/samples/20230422_131600-01.bin", with the start of this sampling taking place on 22/04/2023 at 13:16:00.

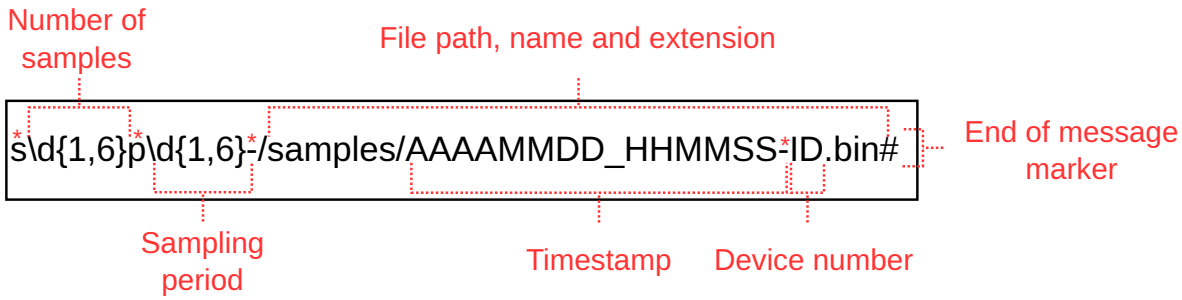


Figure 3.19: Format of the message that indicates the start of each sampling window.

If the parameters received (“p” and “s”) are different from the ones used for the previous sampling windows MCU, then such parameters are updated. Otherwise, the previous values are used.

After that, “ping pong buffers” are used to continuously read samples from the ADC and write them to the micro SD card. This technique, also known as double buffering, is used to efficiently transfer data between two entities, such as a peripheral and a processor, without introducing data corruption or contention issues. Such scheme is suitable for embedded systems due to its low hardware performance requirements [36]. Here a timer interrupt service routine (ISR) is used to ensure smooth and uninterrupted data transfer. It works as described below.

First, two buffers *(Ping and Pong)* of same size (2 bytes) are allocated in memory. This size is enough, since the ADC used by STM32 has a 12-bit resolution. The system starts by defining which of the buffers will be filled first (e.g., Buffer Ping). Then, a timer is configured to trigger an interrupt at a specific interval (the sampling period). When the timer interrupt occurs, the ISR is executed, which is responsible for the data transfer. Inside the ISR, the system checks which buffer is currently being used for data transfer. If Buffer Ping gets completely filled, then it switches to Buffer Pong, and vice versa. The process repeats at the rate defined by the timer. The key advantage here is that while one buffer is being filled, the other is being emptied, i.e. its content is being written to the micro SD card. This allows for continuous, uninterrupted data transfer.

When the total number of requested samples (“s”) is taken, the signal conditioning module sends a command to the communication module informing the sampling window has finished. Then, it simply waits for the next message from the communication module to start a new sampling window.

3.1.4 Embedded Software for the Communication Module

The communication module's firmware is way more complex than the one described in Section 3.1.3. Thus, in order to facilitate visual understanding of the system, several Unified Modeling Language (UML) diagrams were built for describing it. Each of them is meant to help the reader answering fundamental questions about the system's structure and behavior, as indicated by the titles of the subsections below.

After that, the current Section also describes the partition table created for the ESP32 MCU, as well as the OTA DFU process, and the rollback mechanism that prevents the device from becoming unusable due to a problematic DFU. Finally, this Section is concluded with information about tools, libraries and patterns used during the firmware development.

3.1.4.1 Deployment Diagram: Where is it running?

The purpose of the deployment diagram is to model the physical aspects of a distributed system, which is generally used to model the static deployment view of a system (hardware topology). In addition, the diagram models the links between the different items in the system. This visualization is important for understanding the environment in which the embedded software is inserted or, in other words, understanding the big picture.

Fig. 3.20 shows the system's deployment diagram. It must be noticed that, although this image presents the perspective of a single device, there will be multiple hardware running simultaneously on the same network.

In this diagram, the STM32 MCU, used in the signal conditioning module, captures data from the current sensor and writes the information to an external micro SD memory. In turn, the ESP32 MCU, used in the communication module, reads the data stored in the micro SD and communicates with a local server to send the read data and to receive firmware updates, new sampling parameters or commands – such as restarting the device or formatting the memory card.

Such communication takes place via the HTTP protocol using an IEEE 802.11 b/g/n (Wi-Fi) wireless network. The two MCUs are also connected to coordinate the sensor's sampling process, control access to the micro SD card and so that the communication module's MCU can update the firmware of the signal conditioning module's MCU.

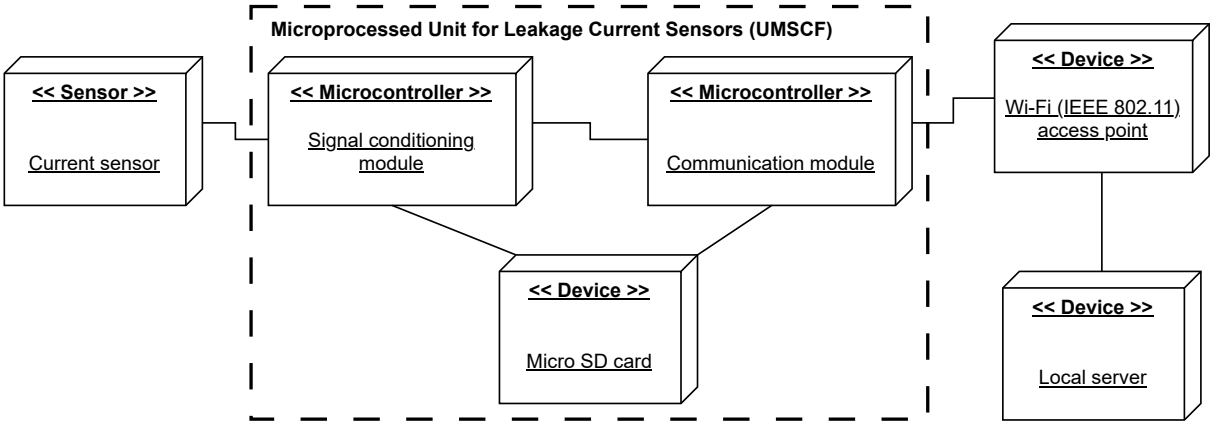


Figure 3.20: UML deployment diagram.

3.1.4.2 Use Case Diagram: What can it do?

Use case diagrams make it possible to model system functions at a high level, as well as the actors who interact with these functions. Use cases are represented by ellipses and describe a function performed by the system. Actors can be users or external components that interact (actively or passively) with the system; they are illustrated by "stick man" icons.

This type of diagram makes it possible to describe different forms of relationship between use cases, as well as between actors and use cases. In Fig. 3.21, the "extend" and "include" association diagrams were used. The association is represented by a solid line between the actor and the use case and indicates the actor's participation in a given use case. "Extend" is represented by a dashed arrow with the label «Extend» specifying that a use case extends the behavior of another use case, describing an additional action. Finally, "Include" is represented by a dashed arrow labeled «Include» and specifies that a use case includes the functionality of another use case, thereby representing a kind of reuse.

Fig. 3.21 shows the use case diagram of the communication module, describing its behavior and its interactions with three external actors, the "Signal conditioning module", the "Local server" and the "micro SD card". These actors have been positioned to the right of the diagram to indicate that their participation in the use cases is passive rather than active, i.e. in all these use cases, the communication module initiates the actions and the external actors react to the actions initiated.

At the top of the image is the *"Trigger new sampling window"* use case. It indicates that the communication module can trigger the start of the sampling window by the signal conditioning module. In order for this to happen, the communication module needs to construct the

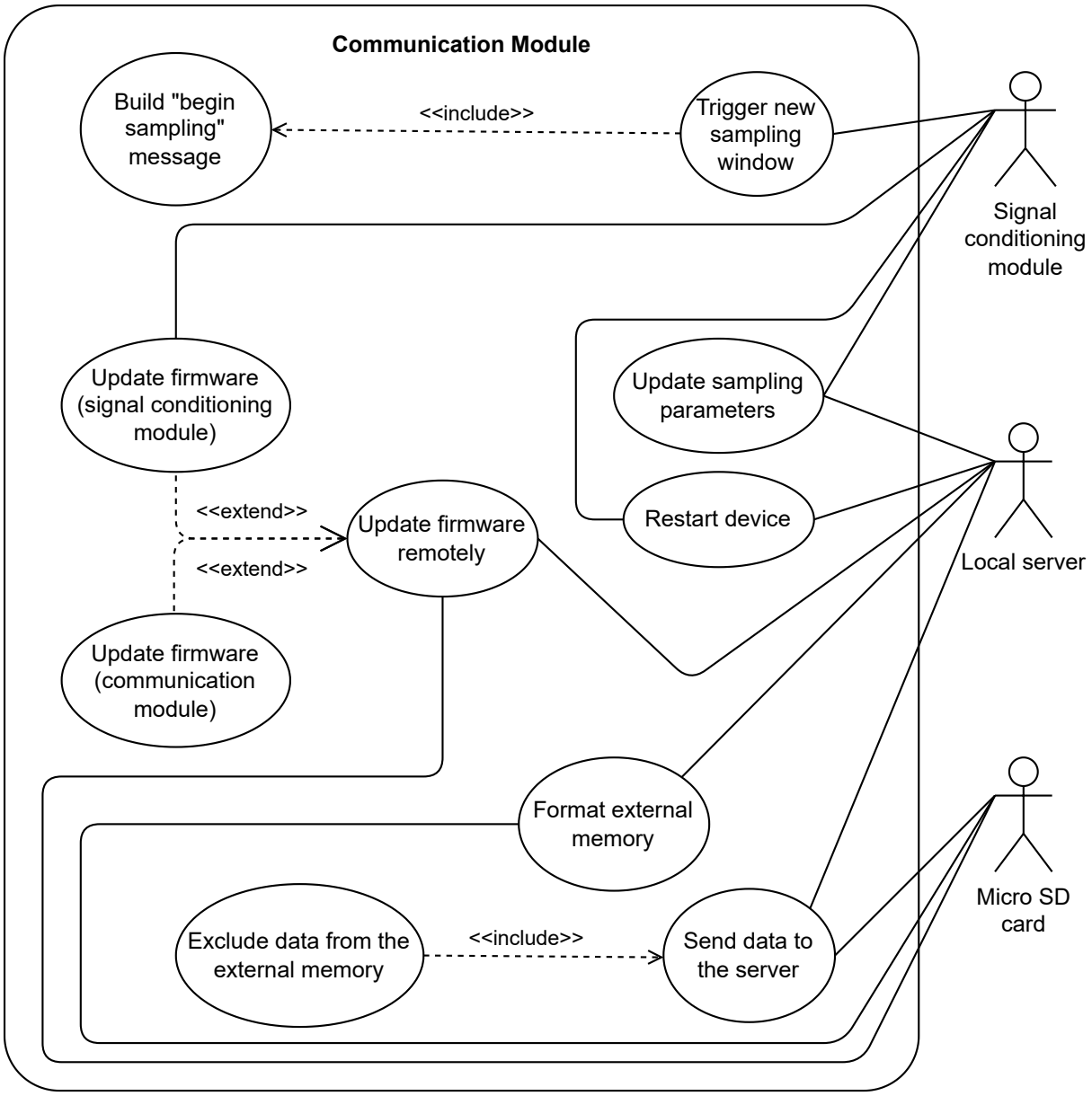


Figure 3.21: UML use case diagram for the communication module.

sampling start message which carries the timestamp information and two parameters: sampling period and number of samples to be captured in the window. The format and construction of this message are detailed in Fig. 3.19.

There is also the *"Update sampling parameters"* use case, which involves the local server and the signal conditioning module. On this occasion, the server sends new sampling parameters to the communication module, which in turn transmits them to the signal conditioning module from the next sampling start message.

Both *"Restart device"* and *"Format micro SD card"* use cases include receiving a remote command and act based on that. In the former case, it simply restarts itself, and consequently

restarts the signal conditioning module – since this is done by default at the setup routine of the communication module. In the later case, a procedure to format the micro SD card file system is called.

The *"Update firmware remotely"* use case involves at least two an up to all the three actors at the same time. In this process, the communication module checks whether firmware updates are available on the server for itself or for the signal conditioning module. As a result, the use case is subdivided into two others, depending on the target of the update. In both scenarios, the binary update file is downloaded over the wireless network and stored in the UMSCF's micro SD card. The communication module then updates its own firmware or the firmware of the signal conditioning module.

Finally, at the bottom of Fig. 3.21, there is the *"Send data to the server"* use case. This action involves both the micro SD card and the local server, since the samples are stored in files on the micro SD card, which are read and sent to the server by the communication module. The files successfully received by the server are then deleted from the micro SD card.

3.1.4.3 Sequence Diagram: When does each thing happen?

Sequence diagrams, also known as interaction diagrams, are responsible for illustrating the interactions between the various components of the system following a sequential order, i.e. a temporal order in which the interactions take place. A sequence diagram shows, as parallel vertical lines (called "lifelines"), different processes or objects that are active simultaneously and, as horizontal arrows, the messages exchanged between them, in the order in which they occur.

Fig. 3.22 shows the sequence diagram drawn up to illustrate the interactions between the signal conditioning module, the communication module, the local server and the micro SD card shared between the two modules. Initially, the communication module sends a signal to the signal conditioning module to start sampling the current sensor. This signal is a text message via the UART[2] interface informing the timestamp and parameters to be used in the sampling window. The collected data is then stored by the signal conditioning module as files in micro SD card and, after all the samples have been stored, the signal conditioning module informs the communication module that sampling has finished, via a falling edge on the digital pin connected between STM32 and ESP32. In this way, the communication module is aware that

[2]The 9600 8N1 configuration was used (9600 bps, 8 data bits, no parity bit and 1 stop bit).

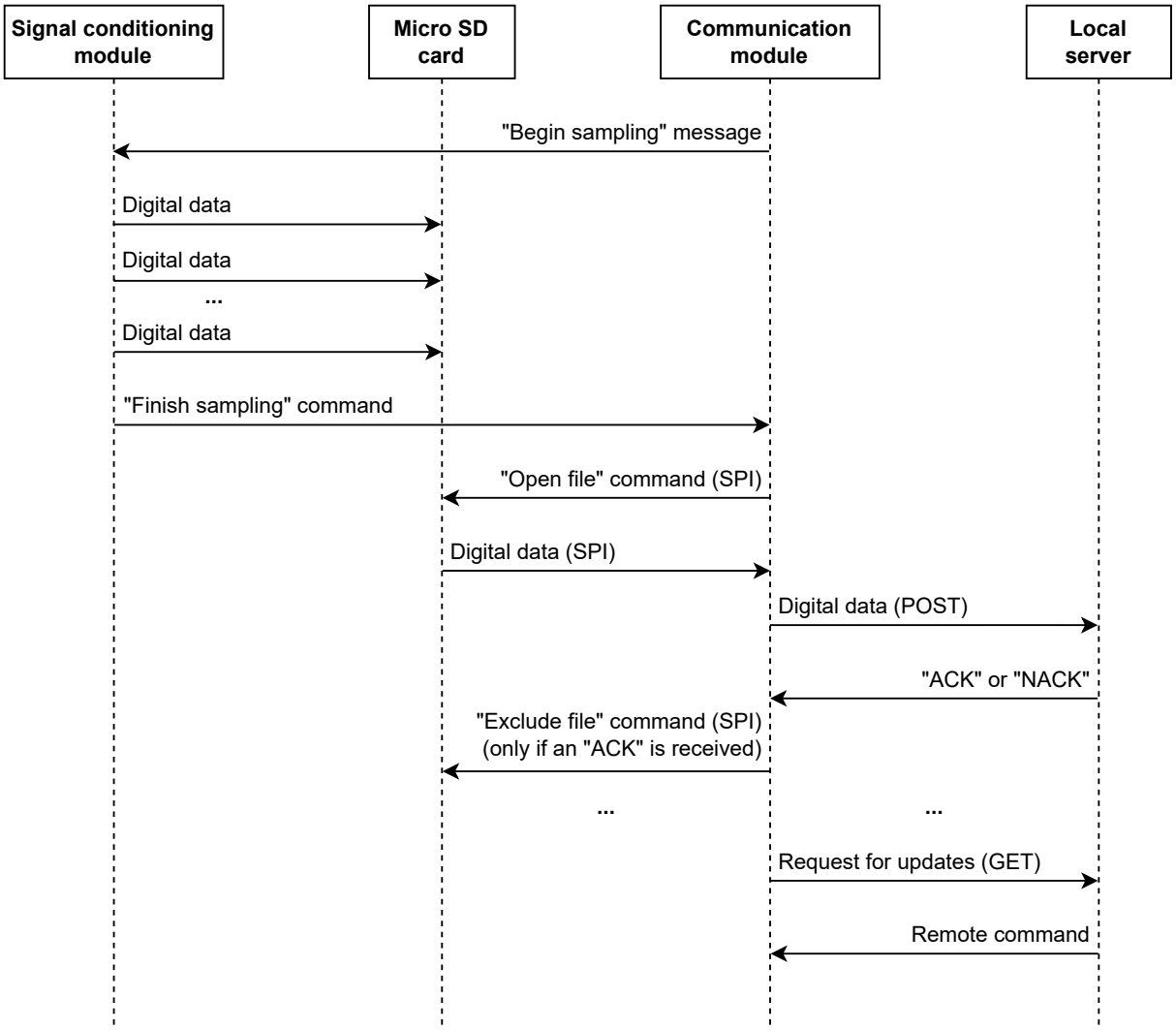


Figure 3.22: UML sequence diagram representing the interaction between the main actors of the system.

all the requested data is ready to be consulted on the memory card.

Next, the communication module requests a stored file from the micro SD card in order to send it to the local server via an HTTP POST request. The server, in turn, returns a response of receipt (code 200 or 204) or non-receipt (another code). If code 200 or 204 is confirmed, the file is deleted from the micro SD. This process is repeated until all the data stored in the micro SD card has been transmitted to the local server. When the communication module detects that there is no more data in the micro SD card to be transmitted, it processes any commands received from the local server.

Finally, the communication module enters a sleep state (not shown in the Fig. 3.22), waiting for the next sampling window to start, when it again sends a message asking the signal conditioning module to start sampling the current data on the isolator and the whole procedure

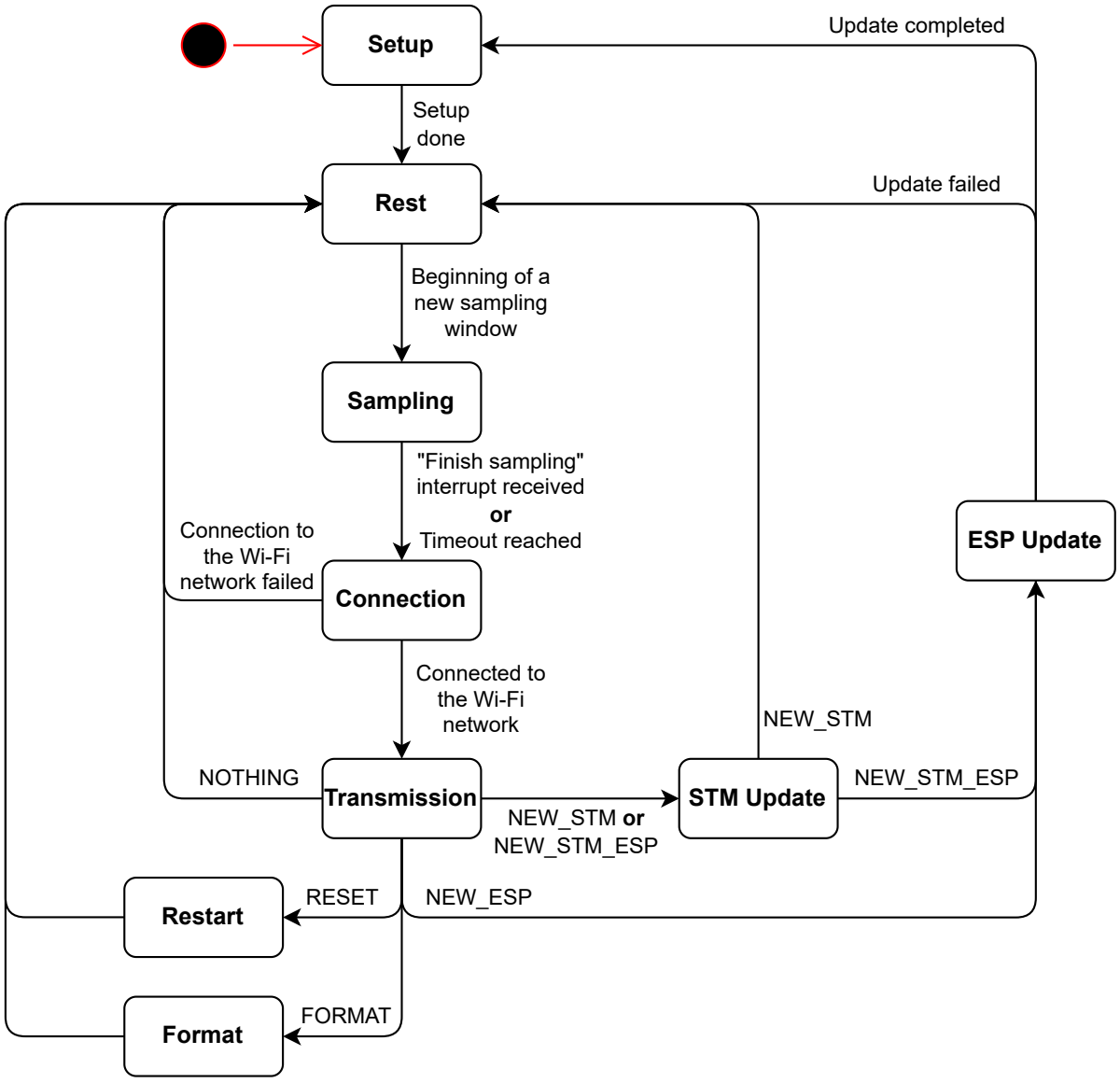


Figure 3.23: UML state machine diagram for the communication module.

is repeated in the sequence described above. The entire sequence of events shown in Fig. 3.22 is repeated every minute, with the number of samples and the sampling period carried out by the signal conditioning module being sent by the communication module along with the timestamp.

3.1.4.4 State Machine Diagram: Which are the possible stages?

The operation of the UMSCF communication module is better detailed using the state machine diagram shown in Fig. 3.23. In this diagram, the boxes with rounded corners represent the different states of the system's firmware, while the arrows represent the events that cause the transitions between one state and another.

In normal operation, the system should start in state *Setup* and immediately proceed to state *Rest*, then go to state *Sampling* and repeat the cycle from *Sampling* to *Rest* indefinitely. Some special events can cause deviations in this cycle, as described in the paragraphs below.

When the system starts, initial configurations are performed in *Setup*, including: configuring the UART communication pins and command GPIO pins between the modules; recording the compile date; validating the current firmware; and updating the RTC.

In *Sampling* state, the "begin sampling" command to start sampling in the signal conditioning module is executed. The *ESP_STM_CMD* pin is set to high logic level to trigger the start of sampling. It should be noted that the parameters are stored in the communication module's non-volatile memory, so that even if the firmware is updated, the last parameters set are always retrieved. These values are incorporated into the message sent by the UART, according to the sampling start message format shown in Fig. 3.19. Upon receiving that message, the signal conditioning module is expected not only digitize the signal, but also store the digitized data in the micro SD card.

Before leaving the state *Sampling*, the MCU checks whether the sampling has actually been completed successfully or not. To do this, it waits for a confirmation signal from a GPIO interrupt (falling edge on the *STM_ESP_CMD* pin). A timeout of $t_{samplestatus}$ (microseconds) is used to stop waiting for the confirmation signal in case it is not received in time. This value $(t_{samplestatus})$ is proportional and greater than the theoretical time required to capture all the samples ($T \times N$, where T and N are the values that defined as sampling period and number of samples, respectively). If the confirmation signal is received before the maximum waiting time is reached, sampling has been successfully completed.

The next state is *Connection*, where the communication module attempts to establish a connection with the Wi-Fi network. If the connection to the AP fails, the program flow is diverted from state *Connection* straight to state *Rest*. This means some the last file of digitized samples will remain stored in micro SD card. All pending data will only be transmitted the next time the system enters *Transmission* state.

Following the execution flow, once the communication module has established the connection, the sampled data stored in micro SD card (state *Sampling*) can finally be sent to the local server in state *Transmission*. At this point, the micro SD card is accessed via SPI in the and the *"samples"* directory is used to collect the files to be sent one at a time via HTTP POST requests.

In addition, at the end of each data transmission to the server, the system checks if it is time to update the internal RTC, which must be carried out periodically in order to keep the timestamp synchronized with a more accurate clock. If the running time reaches a time defined in the code *(UPDATE_TIME_FROM_NTP_SERVER_MS)*, the communication module establishes a connection with an Network Time Protocol (NTP) server to adjust the RTC. For this step to be successful, such server must be running on the LAN – since the devices are not connected to the internet.

Furthermore, as a response to the POST request, the device receives one among six possible status codes from the server, which are shown in Table 3.1. All these codes starts with "2" to indicate a successful request, satisfying the HTTP standard classes. The following two digits are arbitrary values, whose only constraint is to be different from those already defined in RFC 9110.

If there is no action to be taken, the server returns code 230, which in the code is named with the *NOTHING* macro. In this case, the state machine goes to state *Rest*, where the ESP32 just waits for the next sampling window.

The next three 3 possible responses are related to DFU. If a firmware update is available, these three scenarios are possible: new firmware only for the signal conditioning module *(NEW_STM)*, only for the communication module *(NEW_ESP)* or for both *(NEW_STM_ESP)*. Priority is given to updating the signal conditioning module, so if the result of this check is *NEW_STM* or *NEW_STM_ESP*, the system enters state *STM Update*, in which the new MCU STM32 binary file is downloaded and saved. If the result is *NEW_STM_ESP*, the flow continues from state *STM Update* to state *ESP Update*, where the download and write process is carried out again, this time to the ESP32 MCU. In the scenario where the result is *NEW_ESP*, the transition is made directly from state *Transmission* to *ESP Update*.

On the other hand, the last 2 values are related to remote operations. If the device receives the code 240 *(RESET)*, it goes to state *Reset*, where it restarts itself. If code 241 *(FORMAT)* is received, the device goes to state *Format*, where the procedure to format the micro SD card gets executed.

Accompanying the information on the availability of firmware updates, the server always sends the communication module the number of samples and the sampling period that should be used from the next sampling window. These parameters are stored in a text file on the server.

If the communication module is successfully updated, the system returns to state *Setup*. If

Table 3.1: HTTP status codes from the local server.

Macro	Value
NOTHING	230
NEW_STM	231
NEW_ESP	232
NEW_STM_ESP	233
RESET	240
FORMAT	241

only the signal conditioning module is updated, or if either of the two possible updates fails, the flow goes to state *Rest* and waits for the next sampling window, when the DFU shall be retried.

In the event of connection or update failures, as well as in the best cases where the execution flow has successfully proceeded from state *Transmission*, state *Rest* is reached. From there, the connection to the network is disabled and the device waits the beginning of the next sampling window. This duration is may vary for each iteration of the system, and is given by the equation

$$t_{rest} = min(0, \Delta_t - t_i)$$

where: t_i represents the time taken from the beginning of the last sampling window until the current instant, and t_{rest} indicates how long the system should wait before the next sampling window. The only fixed value is Δ_t, which is the time between sampling windows. All these values are in seconds. If $\Delta_t - t_i$ is a non-positive number, them the system immediately begins the next sampling window.

3.1.4.5 OTA DFU Updates

UMSCFs are designed to be installed in hard-to-reach places. For this reason, the possibility of OTA DFUs is an important feature for this work. For this reason, the activity diagram in Fig. 3.24 was developed to represent the firmware update process, taking into account the actions carried out by the communication module and the project's local server which will store the update files.

The activity diagram functions as a flowchart that takes into account the interaction between different actors. In this case, in Fig. 3.24, the lower box represents the actions taken by the communication module, while the upper box shows the actions performed by the local server during the remote firmware update operation.

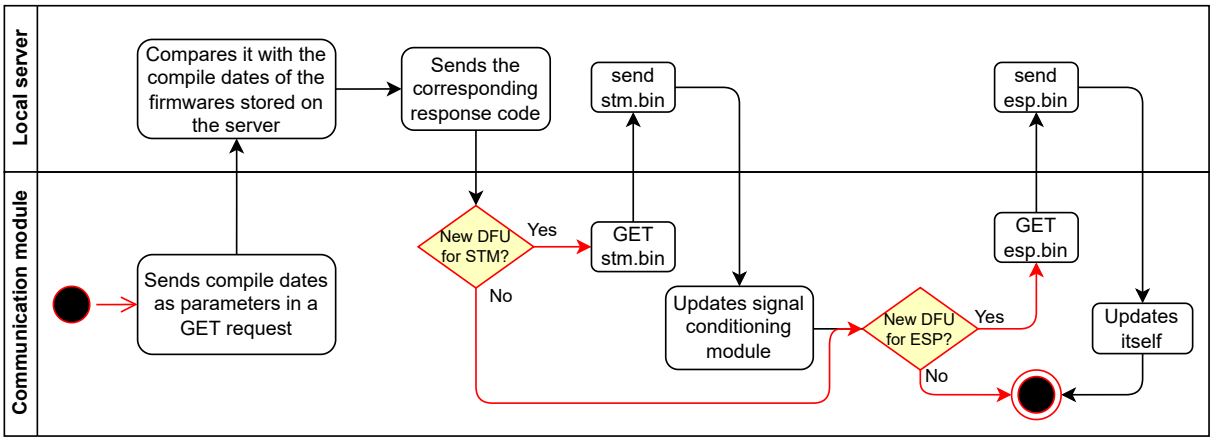
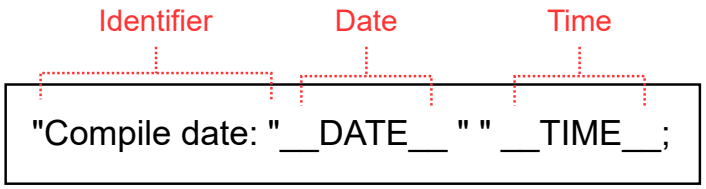
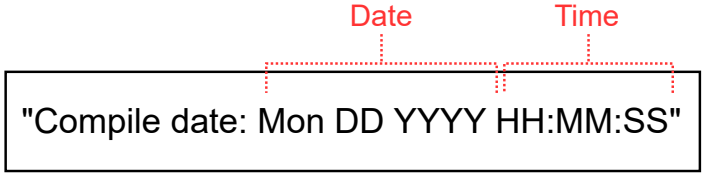


Figure 3.24: UML activity diagram representing OTA DFU process.

First, a comparison is made between the compile dates of the files available on the server and those of the applications that are currently running, the format of which is illustrated in Fig. 3.25, where **(a)** is the upper part shows the command in C/C++ language to capture the compile date and **(b)** shows the format of the string generated by the command, where "Mon" is the three-letter abbreviation of the month, *DD* is the day, *YYYY* is the year, *HH* is the hour, *MM* is the minute and *SS* is the second. For example, a code compiled at 15:05:30 on 22/04/2023 would generate a string given by "Compile date: Apr 22 2023 15:05:30". This routine for checking for new firmware on the server is part of the *Transmission* state shown in Fig. 3.23. In other words, this check for updates is carried out continuously after data is transmitted to the server.



(a) C/C++ code to generate a string with the compilation time.



(b) String generated with the compilation time.

Figure 3.25: Compile date format.

The updates themselves take place depending on the result of this comparison. If the compile date of the file available on the server is more recent, it is understood that there is a new update available. This check is carried out independently for the signal conditioning module and the communication module. It is supposed that the most recent versions of both MCUs are always stored as binary files on a special directory of the server, and that it is possible to read their compile dates from these files. If there is no firmware file stored on the server, it is assumed that the devices have the most recent versions of it; hence, no update is needed in that case.

In addition to what is described in the state machine diagram, updating the communication module involves two special features: integrity and firmware validation tests.

Firstly, the integrity test checks whether the file has been corrupted in the transmission process. This is possible because the binary file received from the server includes the checksum of the new firmware. With this information, the system verifies the checksum received and calculates the checksum of the new firmware received. In this way, it is possible to check whether any data has changed during transmission and storage on the micro SD card and, in the event of corrupted data, the new firmware will not be used on the device.

If the integrity test is successful, the new image is written to one of the application partitions and the device is rebooted to run the new program. However, there may be critical flaws in this new program that cannot be detected by the integrity test. For this reason, when entering the *Setup* state, a validation test is run to check that the device is still able to connect to the wireless network. This check is only carried out the first time the new firmware is run and, if successful, the application is marked as valid. Otherwise, the system is rebooted and reverts to a previous version. The mechanism that enables the validation test is described in more detail in Section 3.1.4.6.

In summary, if the update process as a whole is successful, the activity is finished and the system moves on to the next state – as explained in Section 3.1.4.4. If there is an error in the program integrity or validation test, the update fails, which is followed by a reboot of the communication module and execution of the previous firmware.

3.1.4.6 Firmware Rollback

As detailed in Section 3.1.4.5, the criterion that the UMSCF uses to check whether updates are available is based on comparing the device's compile date with the one stored on the server.

Naturally, a rollback would result in a return not only to an older application, but also to an older compile date, so that the system would enter an infinite loop of updates and rollbacks. To avoid this problem, it is necessary for the running program to have access to some information related to what happened in the previous application. In order to understand how the rollback is implemented, it is necessary to know about the ESP32's flash memory partitions.

The ESP32's Flash memory – the MCU chosen for the UMSCF communication module – has 4MB and is divided into several partitions. The Partition Table is used to manage these divisions. In this MCU, there are two main types of partition, each of which has a set of subtypes. The *app* type is used for application partitions and is subdivided into *factory* (basic program, which cannot be overwritten with OTA updates), *ota_0* and *ota_1* (programs received via OTA update). The other type is *data*, used for various categories of data, among which the most important in the context of this work are: *ota* (works as a pointer, indicating which partition should be initialized at the next boot), *nvs* (non-volatile storage, to store variables whose value remains stored even if the device is restarted) and *coredump* (stores critical error logs such as *crash* and *panic*).

In order to increase the robustness of the device and the efficiency of memory use, a customized partition table has been created for the UMSCF, shown in Table 3.2. Its main characteristics are: 20KB of persistent storage *(nvs)*; 8KB for operating OTA updates *(otadata)*, 0.9375 for the factory app, 1.4375MB for *app0* and 1.4375MB for *app1*. The latter two are responsible for the main application. In addition, this memory partition does not include FAT or SPIFFS file systems, since the external micro SD card meets this need.

In Table 3.2, each row corresponds to a partition and each column to an attribute. The "#Name" column assigns an arbitrary name to each partition. "Type" and "SubType" refer to the types and subtypes explained above. "Offset" indicates the start address of the partition in flash memory in hexadecimal. In turn, "Size" specifies the size in hexadecimal to be occupied by the partition.

Two OTA application partitions are needed because of the way remote updates are handled on the ESP32. If the *factory* program is running, and a remote update is received, the new application is stored in *app0 (ota_0)*. The device then reboots, running the *app0* program. If another remote update is received, a third partition will be needed to store it, since the *factory* cannot be overwritten remotely and *app0* cannot overwrite itself while it is running. Thus, the new application is stored in *app1 (ota_1)* and the device reboots into this partition. When

Table 3.2: Partition table created for the communication module.

#Name	Type	SubType	Offset	Size	
				Hex	Bytes
nvs	data	nvs	0x9000	0x5000	20 KB
otadata	data	ota	0xE000	0x2000	8 KB
factory	app	factory	0x10000	0xF0000	0.9375 MB
app0	app	ota_0	0x100000	0x170000	1.4375 MB
app1	app	ota_1	0x270000	0x170000	1.4375 MB
coredump	data	coredump	0x3E0000	0x10000	64 KB

the next update is received, the *app0* partition is overwritten and the device reboots into this partition. This process continues indefinitely, so that app0 and *app1* are switched between, while the factory partition remains unchanged.

With this in mind, it is possible to visualize how the rollback process works. To this end, two variables stored in the nvs partition are used: *compile-date* and *use-real-date*. The activity diagram in Fig. 3.26 shows a scenario that provides a better understanding of the rollback mechanism. In this scenario, the running application is in the *app1* partition. If the validation test is unsuccessful, *compile-date* receives the build date of the running firmware, *use-real-date* is reset, the application is marked as invalid and the system is rebooted into the *app0* partition. At this point, the *use-real-date* flag is checked. If it is active (value 1), the build date of the new firmware is taken as a reference. If it is cleared (value 0), the previous program's build date is taken as the reference. If, once again, the validation test is unsuccessful, the steps described above are repeated and the system is rolled back again, this time to the factory partition.

Again, the *use-real-date* variable is tested. As the system is never rolled back when it is in the *factory* partition, all that can be done is wait for a new update. However, due to the mechanism implemented, the update will only occur if the file available on the server has been compiled at a later time, not only for the running program, but also for those that have been rolled back. This prevents the problem of the infinite update and rollback loop until then.

3.1.4.7 Tools, Libraries and Patterns

The embedded software for the UMSCF communication module was developed using version 2.0.6 of the Arduino-ESP32 toolchain[3]. This toolchain provides high-level functions

[3]Available at: https://github.com/espressif/arduino-esp32.

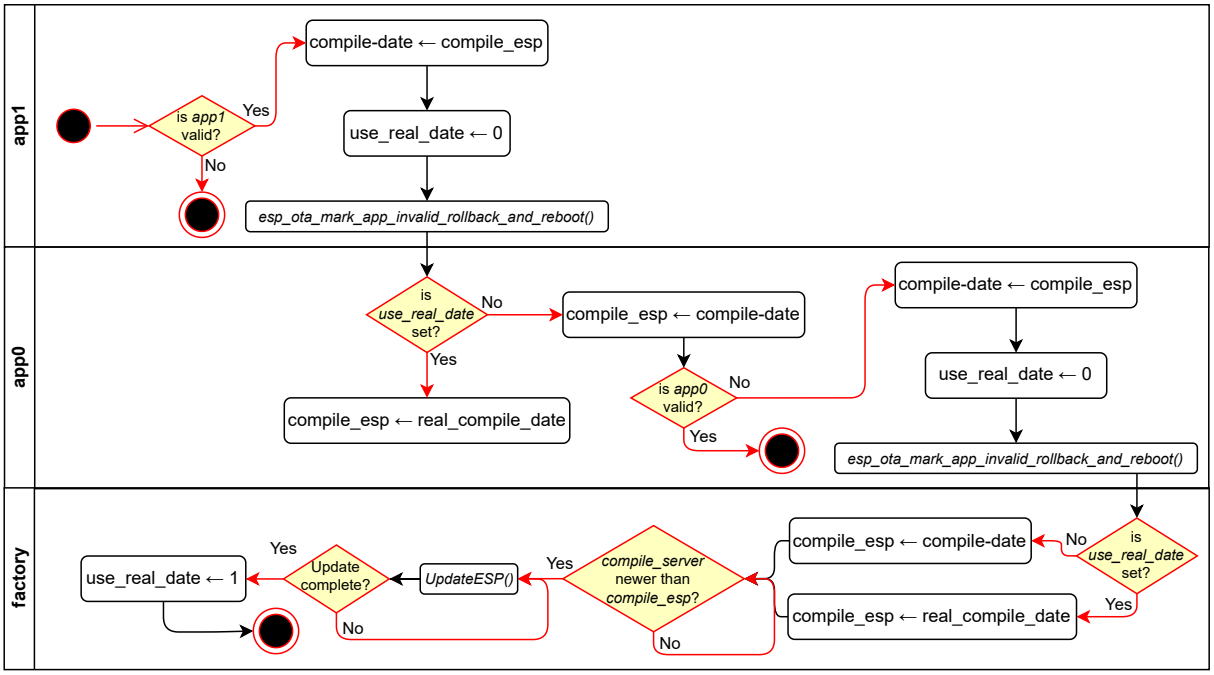


Figure 3.26: Activity diagram illustrating the firmware rollback process.

to handle wireless network access, interfacing protocols, input and output mechanisms, among others. In addition, various libraries were built or adapted to deal with the device's different functionalities, which are listed in Table 3.3.

Table 3.3: C/C++ libraries created or adapted for the communication module's firmware.

Title	Goal
config	General settings
esp_to_stm	Interface between both MCUs
http_gimpsi	HTTP requests
sd_gimpsi	Interaction with the micro SD card
stm_flash_sd	DFU for the signal conditioning module (STM32)
stm_pro_mode	DFU for the signal conditioning module (STM32)
timestamp_gimpsi	RTC and NTP related procedures

For the system to work properly, the wireless network credentials and the hostname of the local server that will be used must be entered in the *config.h* file. The network name and password must be entered, respectively, after the `#define WIFI_SSID` and `#define WIFI_PASS` directives.

Another important issue is related to the *ESP_LOGx()* instructions for writing to the serial, which are present throughout the source code. In these instructions, 'x' corresponds to the

category of information to be printed *(V - Verbose; D - Debug; I - Info; W - Warning; E - Error)*. When compiling a new program, it is possible to specify changing the sensitivity level (to print only warnings and errors, for example) without having to modify the code itself. It is also possible to ignore all categories, resulting in a more optimized application.

Finally, the following naming convention has been adopted for all libraries developed or adapted by the team, as well as for the main file. Functions follow the standard pascal case (PascalCase); volatile memory variables follow the snake case (snake_case); non-volatile memory variables, the kebab case (kebab-case); and macros follow the upper case (UPPER_CASE). This makes it easier to distinguish between the different types of entity, speeding up the development process.

Taking into account all the embedded software developed for the communication signal conditioning modules, more than 70 functions have been built across 17 files, which together have 1903 physical source lines of code (SLOC) [37], excluding comments and blank spaces.

3.2 Network Layer: From the remote devices to the local platform

Connectivity is another key aspect of any IoT solution, and it is handled by the network layer. In this context, a major question is about what communication technology should be used. In this chapter, first a technical review of the most common communication technologies is provided in order to choose the most suitable for this project.

As a first step, it is mandatory to have established what requirements the network is supposed to attend. The system under consideration in this work is used to collect the waveform of current signals flowing through insulators scattered around the substation of “UTE Porto de Sergipe I”, and correlate current levels with the salinity deposited in them or anticipate possible faults that leakage currents may indicate. The considerations below are relevant to determine what technology to choose.

Main requirements:

1. The captured samples must be sent to a central unit via a wireless network.
2. The wireless network must be able to cover the area where the insulators are installed

in the “UTE Porto de Sergipe I”, which is approximately 150 square meters in line-of-sight (LOS) or near LOS conditions.

3.2.1 Choice of Wireless Technology to Be Used

The wireless network technologies discussed in Section 2.4 have different characteristics, as summarized in Table 3.4. Hence, the most suitable protocol for a given application will be defined based on the project requirements.

Table 3.4: Comparison between different communication protocols.

	LoRaWAN	**Sigfox**	**NB-IoT**	**Wi-Fi**	**6LoWPAN**	**ZigBee**
Typical range	$\leq$ 15km	10 – 40km	1 – 10km	140m	200m	150m
Energy consumption	low	low	medium	high	low	low
Max. data rate	50kbps	0.6kbps	200kbps	600Mbps	250kbps	250kbps
Payload (bytes)	243	8	1600	2312	108	68
Device cost	low	low	medium	low	low	low
AP/gateway cost	medium	medium	high	medium	low	low
Requires data plan?	no	yes	yes	no	no	no

All the considered options fulfill Requirement 1, since they are all wireless communication technologies. However, other aspects must still be discussed.

For example, LoRaWAN, Sigfox and NB-IoT technologies have a relatively high typical range when compared to the other technologies; however, they have very limited data transfer rates. In this case, these technologies are more advantageous to use when large areas need to be covered and the volume of data is small.

The maximum rate achieved by NB-IoT is considerably higher than LoRaWAN and Sigfox, but its use is linked to the contracting of a data plan, as well as Sigfox networks. Such approach comes with the advantage of eliminating infrastructure and maintenance costs, as they are offered by third parties. On the other hand, the recurring cost of the data plan, specially when considering multiple devices, may be a limiting factor.

Fig. 3.27 shows the Wi-Fi coverage simulation of a mesh network based on the Google Earth image of the “UTE Porto de Sergipe I” with one AP positioned in the control room and another positioned in the near to a central power socket. In this case it is assumed that each AP has a range of 150m.

Figure 3.27: Mesh network coverage simulation in the substation of "UTE Porto de Sergipe I".
Source: Google Earth.

For this project, it was found that Wi-Fi satisfies the network implementation requirements. Some advantages of Wi-Fi for the project are: low implementation cost, sufficient range – for Requirement 2 – relatively large payload and easiness of network expansion and maintenance. In addition, the range of the network can be easily extended using a mesh network with multiple APs, which is tested in Section 4.3.

3.3 Support Layer: Data storage and device management

As described in Section 1.1, the scenario under investigation is composed of n IoT nodes connected to a local server over a Wi-Fi (IEEE 802.11) network infrastructure. Once the devices are deployed over the wireless network infrastructure, the Support Layer is responsible for providing a platform with computational services to the remote nodes. For the current project, it consists of data storage and metadata processing – generic support capabilities – as well as time services (NTP), and remote operations – specific support capabilities.

At the beginning of this chapter, instructions are given on how to install and configure the prerequisites for the application that will be running on the local server (Section 3.3.1). Then,

the basic usage of the platform is described, including images from the front-end, and code used for validation logic on the back-end (Section 3.3.2).

3.3.1 Local Server Configurations

To enable the operation of UMSCFs, there must be a properly configured server on the same network as the devices are connected to. On this server, the NTP – to provide date and time for the devices – and HTTP – to receive samples from the UMSCFs and interact remotely with the devices – protocols must be enabled. The server's hostname must also be configured so that it is accessible on the network by the name expected by the UMSCF.

The installation steps presented in this Section assume a Linux operating system (OS) environments. The steps indicated for the Linux environment work for any Debian-based distribution (including Ubuntu and Raspbian). Throughout development, operational tests were conducted extensively on a Raspberry Pi 3b+ using the Raspbian 11 Linux distribution.

Throughout this Section, various bash commands are indicated which should be executed. To indicate the beginning of each command, the "$" symbol is used, which should not be copied with the rest of the text.

3.3.1.1 Prerequisites

First, the *systemd* tool must be installed in order to check the current hostname and change it to the correct one, according to what is expected by the UMSCFs, as indicated in Listing 3.1 – every time *{hostname}* is used in this chapter, it must be replaced for the real hostname of the server. After that, the system must be rebooted for the changes to take effect.

Listing 3.1: Bash commands to change server's hostname

```
1  $ sudo apt install systemd
2  $ hostnamectl status
3  $ hostnamectl set-hostname {hostname}
```

Next, the local NTP service must be enabled by adding the text from Listing 3.2 into the `/etc/ntp.conf` configurations file. After that, the NTP service must be installed – if not yes installed – and restarted, as indicated in Listing 3.3.

Listing 3.2: Code to enable the local NTP server

```
# Local NTP server
server 127.127.1.0
fudge 127.127.1.0 stratum 10
```

Listing 3.3: Bash commands to restart the NTP server

```
$ sudo apt-get install ntp
$ sudo systemctl restart ntp
```

Finally, the web application that interacts with the device is based on the Python programming language and uses the socket 8081. Therefore, it is also necessary to ensure that Python 3.8 or later installed and that the socket 8081 is available.

3.3.2 Implementation and Basic Usage

Once the devices are deployed into the network and local server is configured, the files with the data collected by each UMSCF will be sent to the server and stored in the `./uploads` folder where the Python application is running. The name of each file is composed of a timestamp and the device ID, using the format indicated in Section 3.1.4.5.

It is possible to access the captured files using one of the following methods: (1) accessing the server's uploads folder locally; (2) remotely via SSH; or (3) via the web browser, accessing the following link from a computer on the same network as the server (or from the server itself): `http://{hostname}.local:8081/upload/`.

3.3.2.1 Device Monitoring

In order to quickly monitor the remote devices, the Python application provides useful logs, like the one presented in Fig. 3.28. Part **(a)** shows what happens when the server receives POST request in the `upload` endpoint, to receive sample files from the UMSCF 02. First, an "[Upload]" indication followed by the filename is printed, and a status code 204 is sent back to the device. By the left of these three lines, the device's IP address and the time *(DD/Mon/YYYY HH:MM:SS)* is also displayed.

On the other hand, part **(b)** shows what happens when UMSCF 02 is running the *Main App* and makes a GET request to the `verify` endpoint, to check if there are any updates available. If

```
RECEIVE SAMPLES - UMSCF 02
200.239.93.106 - - [25/Oct/2023 13:59:13] [Uploaded] "20231025_135900-02.bin"
200.239.93.106 - - [25/Oct/2023 13:59:13] "POST /upload HTTP/1.1" 204 -
200.239.93.106 - - [25/Oct/2023 13:59:13] "POST /upload HTTP/1.1" 200 -
```

(a) Logs from the `verify` endpoint.

```
VERIFY UPDATES - UMSCF 02 (Main App)
ESP compile date on server: 'Oct 17 2023 10:33:43'
STM compile date on server: 'Sep 18 2023 11:51:20'
ESP compile date on device: 'Oct 17 2023 10:33:43'
STM compile date on device: 'Sep 18 2023 11:51:20'
Sample period (T): 125 us
Number of samples: 80000
```

(b) Logs from the `upload` endpoint.

Figure 3.28: Log messages from the HTTP server.

a problematic new application was sent via OTA DFUs causing the device to be rolled back to the *Factory App* – as explained in Section 3.1.4.6 – this would be indicated in the log message, and the user would know that a new DFU is necessary.

These messages can help on the identification of problems on the devices. An indication that a particular node with ID n has a problem is if there are only log messages on the `verify` endpoint, but not on the `upload` for this particular node n. This can occur, for instance, due to a failure in the device's power supply during a write operation to micro SD card, resulting in corruption of its file system. In this case, a remote operation can be triggered to format the SD card, as described in Section 3.3.2.4.

3.3.2.2 Sampling Parameters Update Webpage

Furthermore, one of the features of the IoT system developed with the UMSCF is that the sampling parameters can be changed at runtime. These parameters are the number of samples and the sampling period – consequently the sampling rate. The most intuitive and recommended way is using the web browser, at the subpage `http://{hostname}.local:8081/parameters.html` (Fig. 3.29).

When this is done, the new parameters are submitted to a validation routine on the back-end, shown in Listing 3.4. Based on the multiplication both parameters, this routine verifies if the resulting sampling window time is shorter than half the time between two consecutive sampling windows. If it is, then the parameters are considered valid and sent to the remote

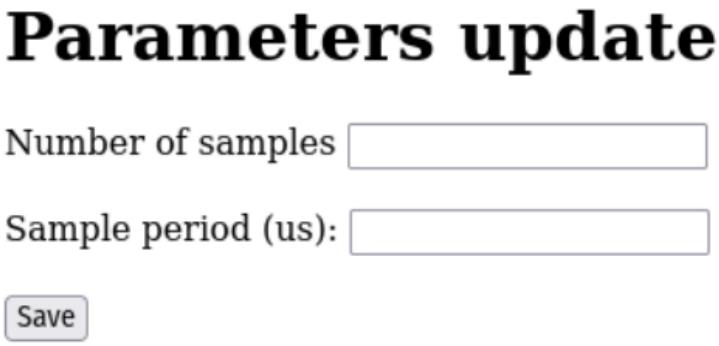


Figure 3.29: Webpage for changing the sampling parameters.

nodes. Otherwise, the user will receive an error message back in the browser asking for valid parameters. This is important to guarantee that the UMSCFs will have enough time to pass through all the states indicated in Section 3.1.4.4 without violating the next window.

Listing 3.4: Routine for validating the sampling parameters received by the server.

```
1    def validate_parameters(number_of_samples, sampling_period):
2        time_between_windows = 60
3        new_window = 1e-6*(number_of_samples*sampling_period)
4        if new_window > 0.5*time_between_windows:
5            return False
6        return True
```

3.3.2.3 OTA DFU Webpage

In addition, it is possible to remotely send an OTA DFU to all the UMSCFs in the network. To do this, the subpage `http://{hostname}.local:8081/firmware.html` must be accessed (Fig. 3.30). In this page, there are fields to attach the compiled binary file and to indicate the module for which this update is meant for. If it is for the signal conditioning module, "STM32" should be selected; in the case the communication module is the target, "ESP32" should be selected.

Similarly to the case with new parameters, when a new firmware is uploaded to the server, it first goes through a validation process. A simplified version of such validation procedure is shown in Listing 3.5.

As explained in Section 3.1.4.5, the DFU logic for the UMSCFs to know when a new firmware is available depends on the compile date information. Therefore, the server must

Figure 3.30: Webpage for uploading new device firmwares.

check if the binary firmware file received from the browser contains this information. If it does, then the upload can be accepted and made available to the remote nodes; otherwise, rejected and not excluded. That is what the code snippet from the aforementioned listing is doing.

Listing 3.5: Routine for validating the new firmware received by the server.

```
1   def receive_firmware(handler):
2       [...]
3       with open(destination, 'wb') as f:
4           data = field.file.read()
5           index = data.find(b'Compile date: ')
6           if index != -1:
7               f.write(data)
8           else:
9               return (http.HTTPStatus.BAD_REQUEST, 'The given
                    file does not have a Compile Date')
10      return (http.HTTPStatus.OK, 'New firmware received
            successfully')
```

3.3.2.4 Remote Operations Webpage

Finally, the following operations can be carried out remotely: forcing DFUs[4]; rebooting the system; and formatting the micro SD card. In this way, any faults can be repaired remotely. There is an order of priority between the operations. The highest priority is formatting the card,

[4]The forced DFUs may be useful mainly for development and debugging purposes, since it should happen automatically for all nodes every time a new firmware is uploaded to the server via the `/firmware.html` subpage.

followed by rebooting the system and finally updating the device. This means that if multiple requests are sent to the same device in a short period of time, only the one with the highest priority will be considered.

To do that, the subpage `http://{hostname}.local:8081/remote_operations.html` must be accessed (Fig. 3.31). Note that, unlike the process to upload a new firmware or new sampling parameters, the Remote Operations page allows the user to send commands to a specific device, informing its ID. In this case, the desired operation and target device – possibly all of them – must be selected.

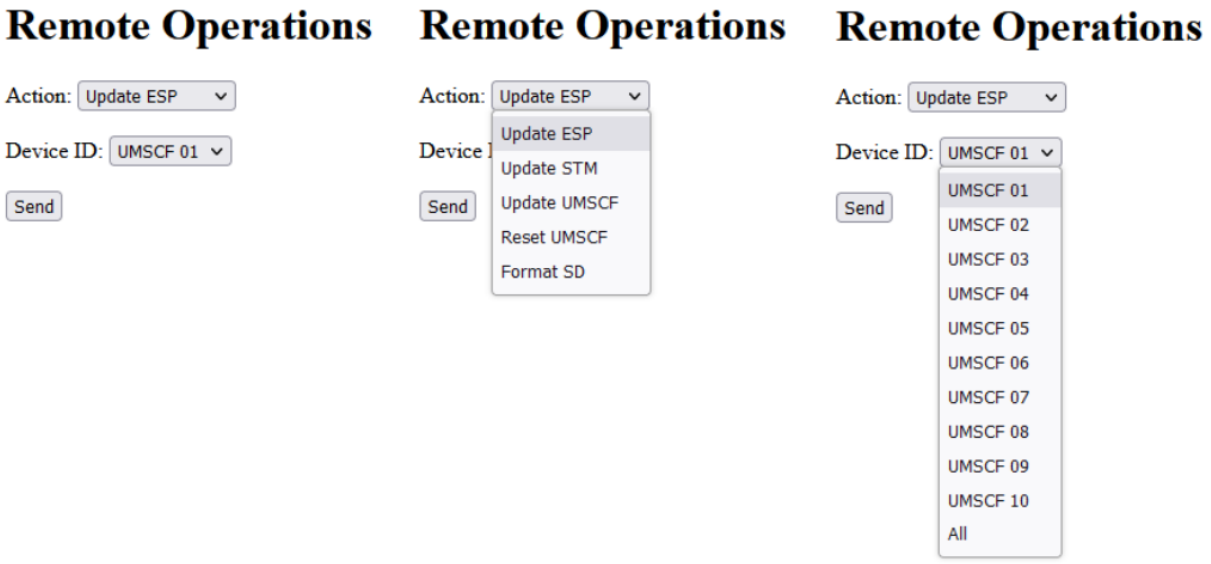


Figure 3.31: Webpage for triggering remote operations over the UMSCFs.

This flexibility is desirable in applications where IoT devices are spread across different areas of monitoring or controlling critical operations, which is the case of the current project. In case one of the devices fails, it is useful to troubleshoot it individually, without the need to interrupt the operation of the others, in order to minimize downtime-related losses.

3.3.2.5 Use Case Example

In order to highlight the system manages remote operations, Fig. 3.32 shows the UML sequence diagram of the interaction among the components of the system in different OTA update scenarios. The main actors are the data acquisition MCU (STM32), micro SD card, communication module (ESP32) and HTTP server (manipulated by an operator). The following scenarios are considered: (1) the firmware running on both MCUs are outdated; (2) the file

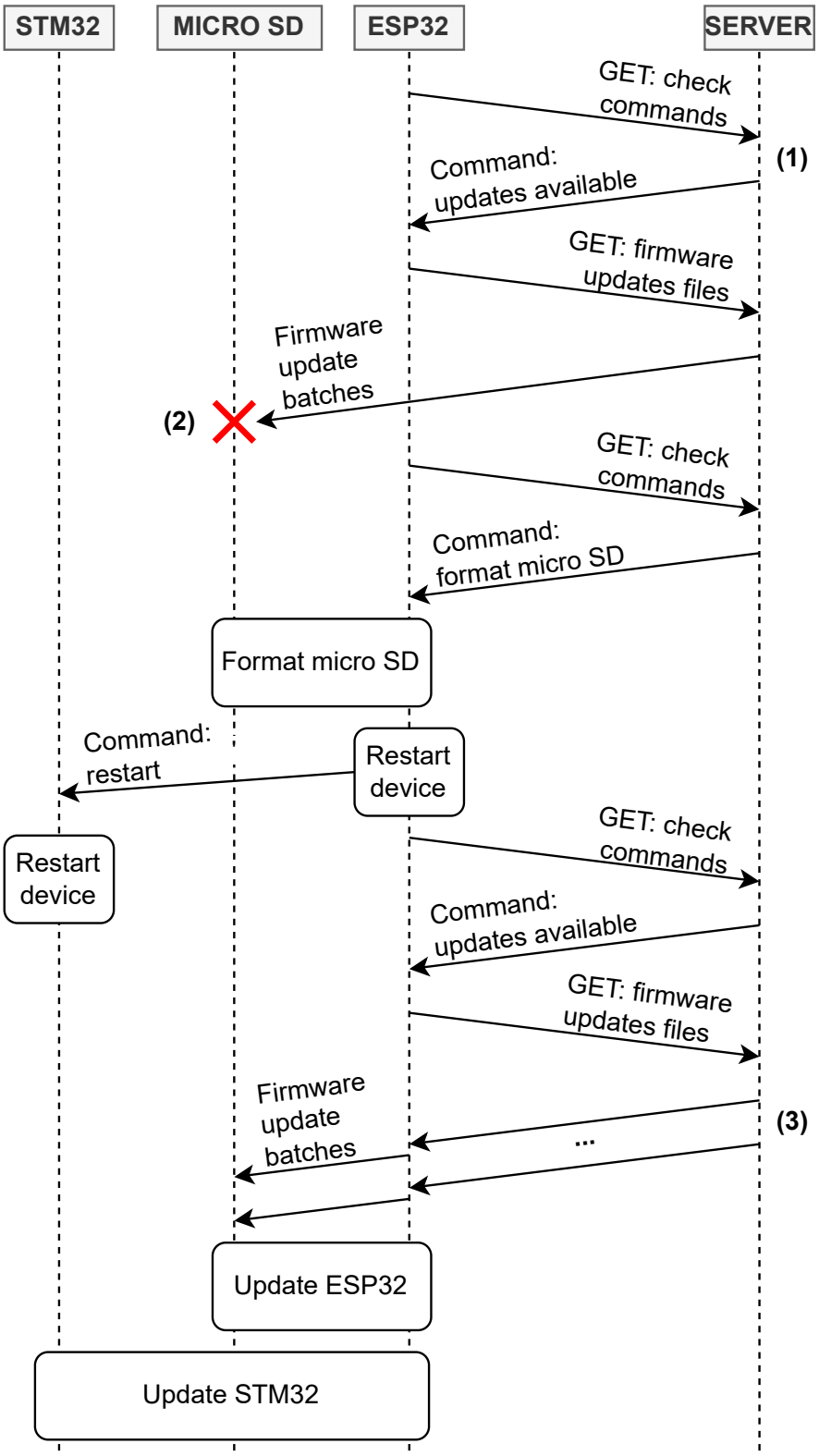


Figure 3.32: UML sequence diagram highlighting the use of the implemented remote operations in different scenarios.

system of the micro SD card has been corrupted; (3) there is not enough space available to store the updates on internal RAM of the communication module.

Reading it from top to bottom, Fig. 3.32 presents the flow of data (arrows) and actions (rectangles) for the given scenario. First, ESP32 tries to download firmware update files, but fails because it cannot write data into the micro SD card. Then, a server operator, noticing a certain device is not working, triggers the command to format the device's micro SD card. That being done, the system is restarted and ESP32 can now perform firmware updates correctly.

Chapter 4

Results

This chapter presents results obtained from experiments conducted with the system that was described in Chapter 3. Section 4.1 presents the results of the tests performed to evaluate integrity of a single data file captured by the DAQ board, in order to validate acquisition mechanism. Then, Section 4.2 brings tests regarding the integrity of multiple data files from multiple IoT nodes when the devices are kept active for many days. Finally, Section 4.3 presents results from an experimental network topology deployed to validate the Wi-Fi (IEEE 802.11 b/g/n) network designed for the project.

4.1 Signal Integrity

In order to evaluate if the DAQ platform operation, an experimental setup was designed, consisting of an MFG-4225 signal generator, a TBS1102C oscilloscope and the UMSCF, as shown in Fig. 4.1. In addition, an HTTP server implemented on a Raspberry Pi was used to receive data from the UMSCF. It should be noted that the POWER PCB worked properly, generating the DC voltages expected by the DAQ platform, so that it was possible to insert a known signal into the DAQ board and view it on the server, as described below.

Two experiments were made to evaluate the signal integrity. First, in Section 4.1.1, the generated signal emulates a leakage current signal, then it is connected to the current sensor input of the DAQ board and passes through all the conditioning stages described in Section 3.1.1.2. However, it is important to stress that the final version of the signal conditioning circuit is still under development, and out of the scope of the current work – as mentioned in Section 3.1.1.1 – so the results may change in the future.

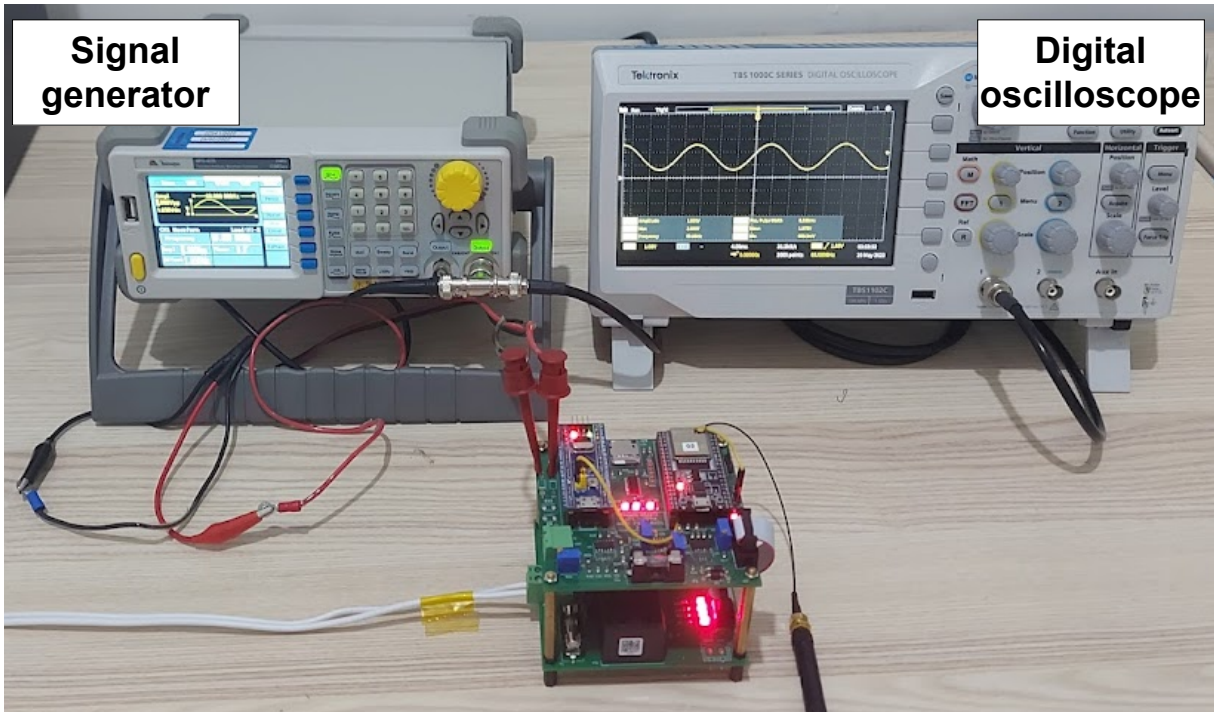


Figure 4.1: Workbench set up for evaluating the DAQ platform.

In order to evaluate the signal acquisition ignoring the conditioning stage, a higher amplitude signal is generated with a DC level previously applied. Then, the conditioning circuit is bypassed, and this signal is directly sampled by the ADC of the STM32 MCU. These results are presented in Section 4.1.2.

4.1.1 Signal Integrity Using the Conditioning Circuit

In this first test, the signal generator was configured to create a 60Hz sinusoidal wave with 100mVpp. Its output was connected simultaneously to an oscilloscope and to the signal input of the UMSCF. The signal conditioning module is responsible for amplifying, filtering and adding a DC level to the input signal in a way it can be correctly digitized by the ADC of the STM32 MCU.

The UMSCF was configured to capture 10 seconds of signal using a sampling frequency of 8kHz. Part of one of the captured signals is shown in Fig. 4.2, from the file "20230816_153900-04.bin", indicating that this file was captured with the timestamp 20230816_153900 by UMSCF 04. In other words, the UMSCF that has the identification 04 captured the data on Aug 16, 2023 at 15:39:00. More details about this capture are described below.

Fig. 4.2 shows two overlapping signals: the signal captured by the UMSCF in blue and

an ideal sine wave given by $x[n] = Asin(\omega n + \theta) + x_{DC}$, where A, ω, and x_{DC} were estimated from the digitized signal. For both signals, the vertical axis on the left shows the ADC's digital output, with the possible values ranging from 0 to 4095, since the ADC integrated with the STM32 MCU has a 12-bit resolution.

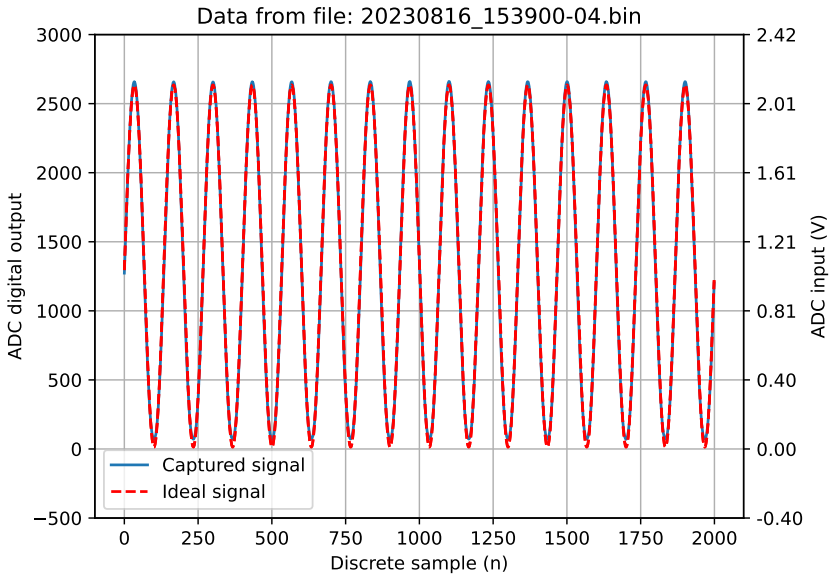


Figure 4.2: First 2000 samples of the captured and estimated signals using the conditioning circuit.

On the other hand, the vertical axis on the right shows what would be the ADC's analog input in volts for the acquired signal. The complete signal in the file has 80000 samples – 10 seconds considering the sampling frequency that was configured – but only 2000 of them are shown in the graph so that the behavior of the signals could be better visualized.

In order to demonstrate that no samples were lost during the stages of digitizing, storing and sending the signal via the wireless interface, the digitized signal was subtracted from the estimated ideal signal, which has no noise. Thus, when there is no loss of samples, the difference is due to some small error in estimating the phase of the signal or due to random noise that is present in the analog input, which has a low amplitude in this experiment. On the other hand, if any samples were lost, the difference would have a larger amplitude. Below there are two figures to illustrate the operations of sampling and transmission of the signal carried out successfully by the UMSCF, i.e. without any loss of samples.

Fig. 4.3 shows the difference between the sampled and the ideal signal for the same time window as in Fig. 4.2, i.e. for 2000 samples. Fig. 4.4 shows the difference signal **(a)** for all the 80000 samples captured, as well as the histogram of this difference **(b)**. Most of the differences

ranges from -40 to +40, which corresponds to approximately 10% of the maximum possible error. The remaining differences come mainly from noise in the digitized signal or small errors in estimating the phase of the ideal estimated signal.

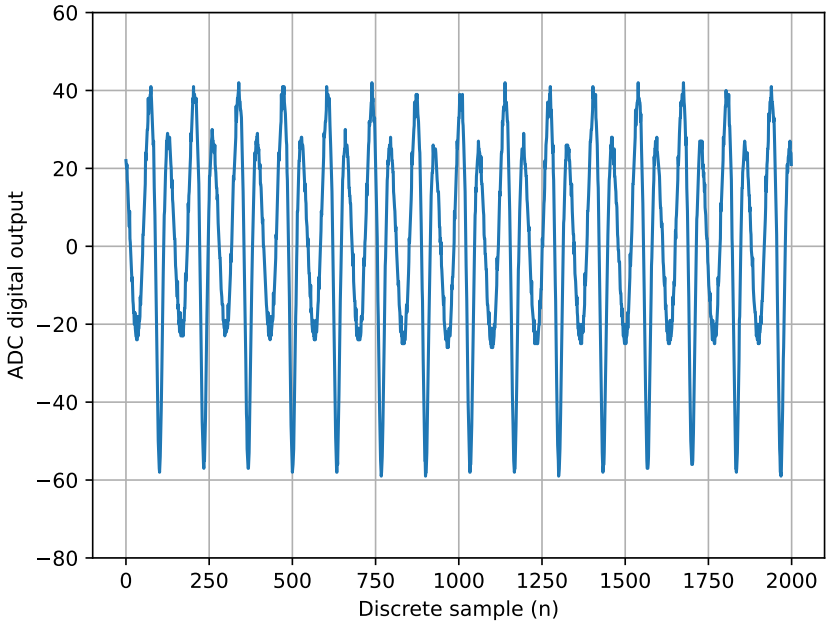
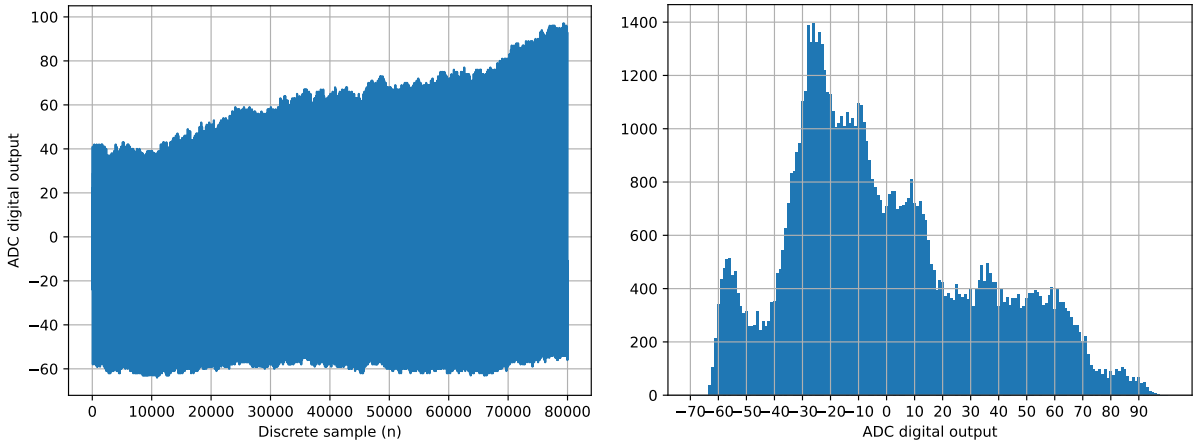


Figure 4.3: Difference between the captured and ideal sign waves for the first 2000 samples using the conditioning circuit.



(a) Difference signal.

(b) Histogram of the difference.

Figure 4.4: Difference between real and ideal sine waves for all samples using the conditioning circuit.

Lastly, Fig. 4.5 shows a simulation of the effect of losing a single sample. In this case, it shows the effect of subtracting the sampled signal from the estimated signal, with the 1000-th sample of the real captured signal being lost. In other words, it shows the effect of the UMSCF

failing to sample or losing the 1000-th sample and considering sample $n = 1001$ as the sample $n = 1000$, sample $n = 1002$ as sample $n = 1001$, and so on.

It is worth noting that the signal shown in Fig. 4.5 is just a simulation so that the reader is aware of the expected effect if a sample were lost. In that sense, it can be verified that, in Fig. 4.3, the unwanted effect is not reproduced. There are only occasional samples in which the difference between the estimated signal and the captured signal has a high amplitude ($\geq |50|$), but this is due to noise in the analog input.

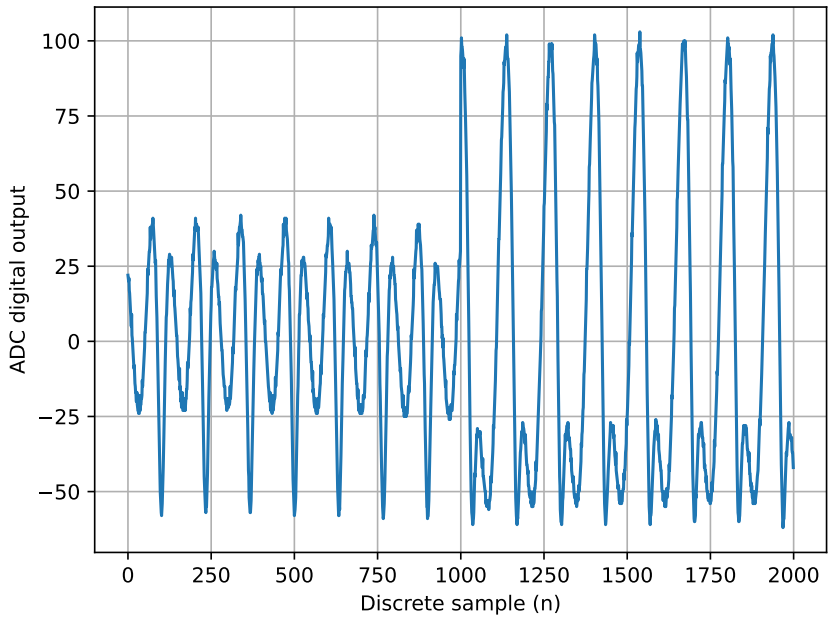


Figure 4.5: Sample loss simulation.

4.1.2 Signal Integrity Without Using the Conditioning Circuit

In this second test, the frequency of the generated wave is the same (60Hz), but the amplitude was increased by a factor of 20, up to 2Vpp. This time, the output of the signal generator was connected directly to the ADC of the STM32; therefore, a 1.65V offset was applied in order to preserve the MCU. Once again, 10 seconds of signal were sampling using a 8kHz sampling frequency.

Figs. 4.6, 4.7 and 4.8 are analogous to Figs. 4.2, 4.3 and 4.4. However, now the file under analysis is “20230519_171106-02.bin”, which was acquired when the conditioning circuit was bypassed.

As expected, the results are much better, with the vast majority of the differences ranging

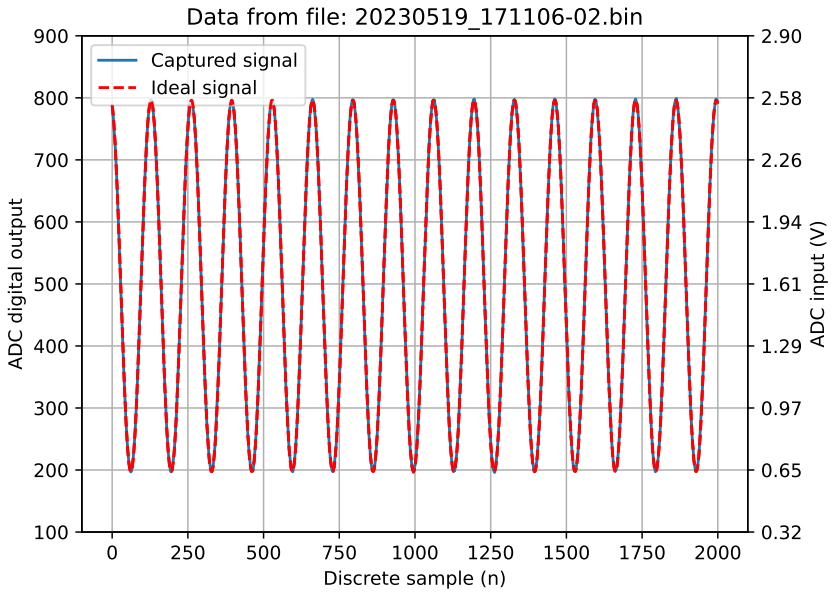


Figure 4.6: First 2000 samples of the captured and estimated signals bypassing the conditioning circuit.

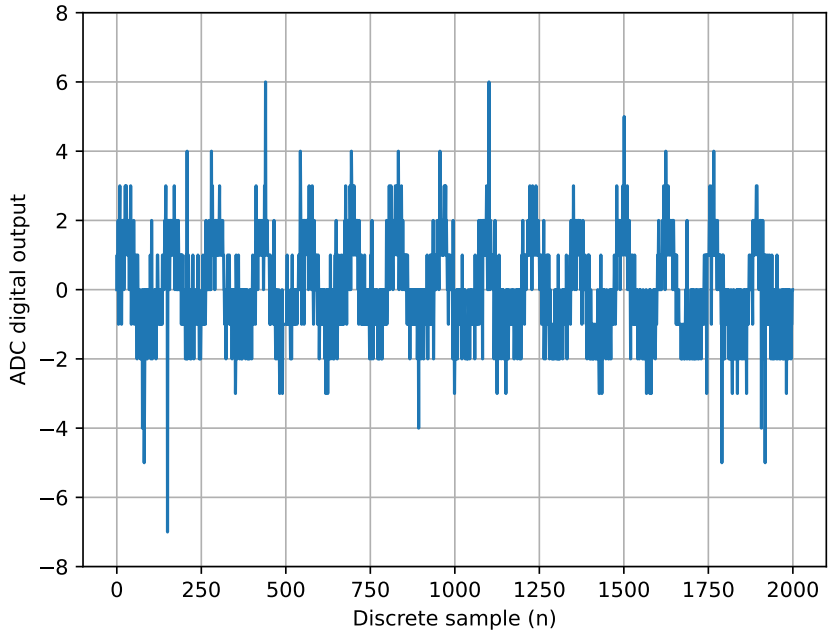
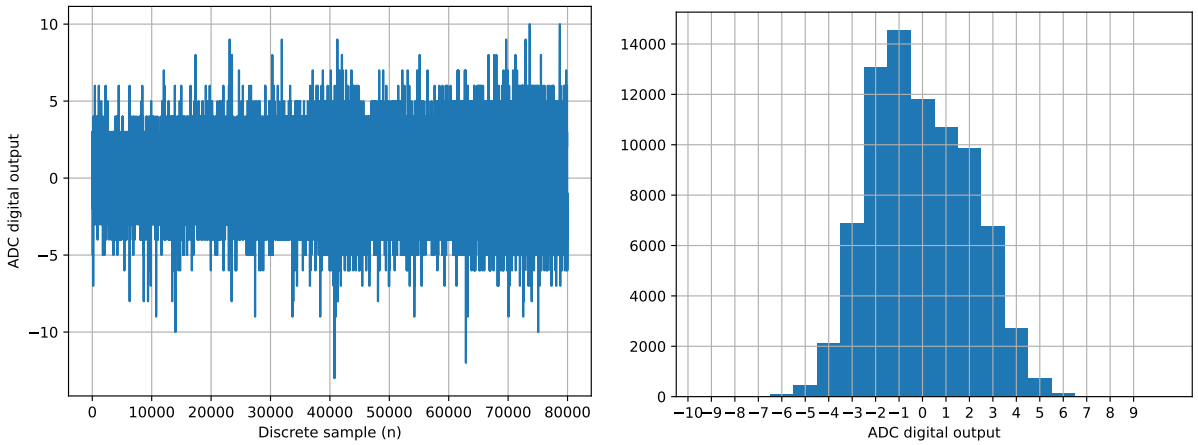


Figure 4.7: Difference between the captured and ideal sign waves for the first 2000 samples bypassing the conditioning circuit.



(a) Difference signal.

(b) Histogram of the difference.

Figure 4.8: Difference between real and ideal sine waves for all samples bypassing the conditioning circuit.

from -4 to +4, that is, only 1% of the maximum possible error. It is clear that the unwanted effect shown in Fig. 4.5 is not present at all.

4.2 Integrity Over Time

Since the multiple IoT devices must work continuously and simultaneously, it is important to have a way to check if these requirements are being attended. It is possible to do it at by observing the timestamps and the metadata of the data files over time, as shown by the Sections 4.2.1 and 4.2.2 respectively. For this experiment, two UMSCFs were kept active for 72 hours continuously (t_h), from 21st to 23rd October 2023. Both devices were configured to collect 80000 samples every 30 seconds.

4.2.1 Timestamp Analysis

First of all, the total number of expected files (N_{total}) can be calculated as

$$N_{total} = d \times f_{rate} \times t_h$$

where d is the number of active devices and f_{rate} is the number of files per hour per device; respectively 2 and 30 in this case. Therefore, it is expected N_{total} to have 17280 files in the server. It is also expected the difference between two consecutive files – i.e the forward difference –

to always be 30 seconds. Since the timestamps are used as filenames for all the data files, this information can easily be extracted in the server.

Indeed, the results are very close to that. The histogram in Fig. 4.9 indicates that the server received all N_{total} (17280) files, with 17168 and 112 occurrences of 30-second and 29-second forward differences, respectively.

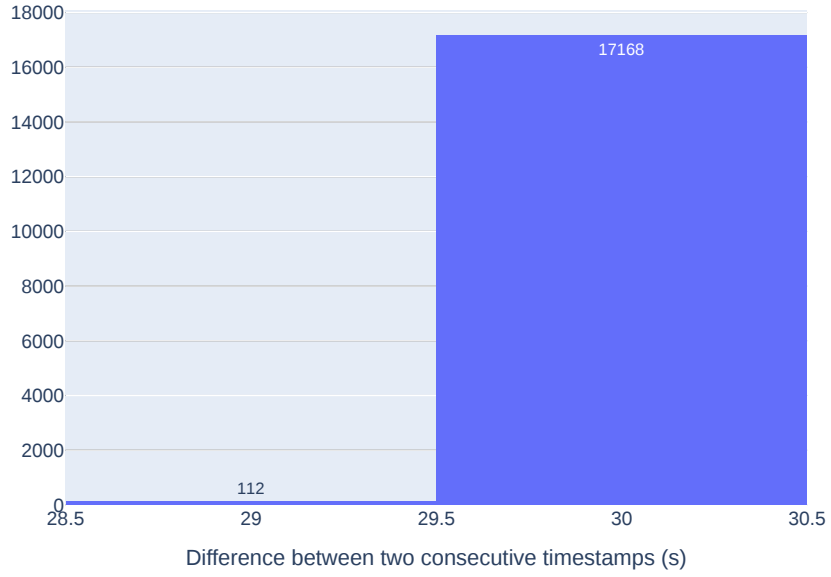


Figure 4.9: Histogram of the forward differences of the timestamps over 72 hours.

The scatter plot from Fig. 4.10 shows this same result from another perspective. It reveals that the files where the errors occurred are not localized in a short timespan. On the contrary, they are spread across the 72 hours domain. This indicates the error is likely to be caused by a small imprecision of the mechanism used to control the timing of sampling windows.

It is important to stress two things. Firstly, that the vast majority of the files are clearly as expected, and the maximum error (1 second), and can be ignored with no harm for the purposes of this work. Secondly, that each timestamp is obtained by the communication module via the local NTP server, and indicates the instant immediately before the beginning of a new sampling window. Hence, it is a reliable metric to check if the f_{rate} is being achieved, even if the wireless network fails at some point.

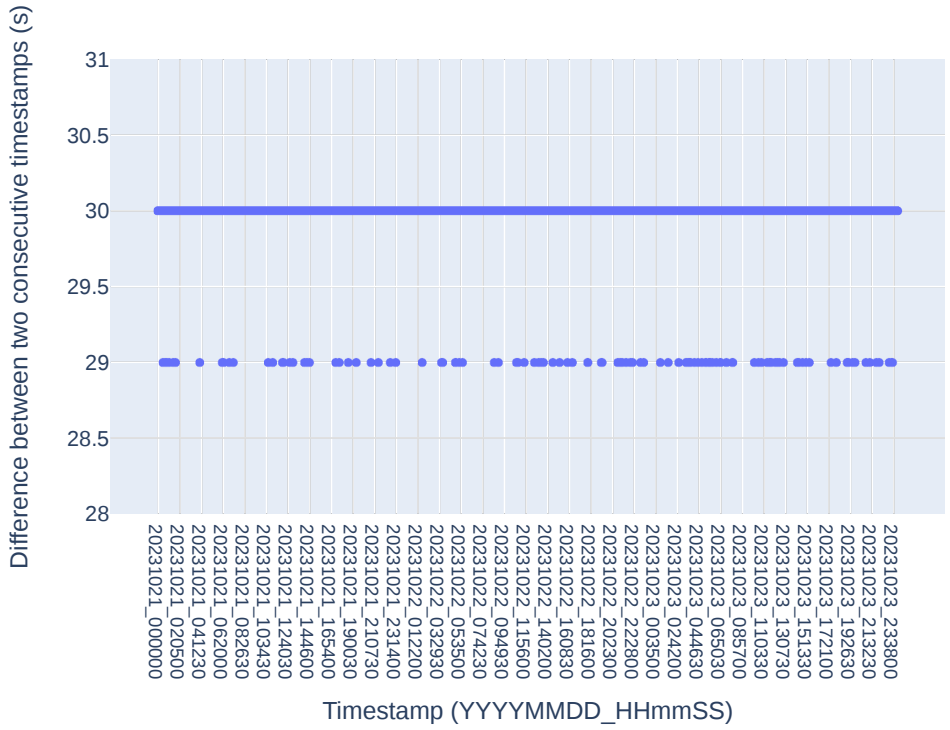


Figure 4.10: Scatter plot of the forward differences of the timestamps over 72 hours.

4.2.2 File Metadata Analysis

The next question to be answered is if all these files were received in their entirety. That is, if all the samples were actually collected or if there was sample loss. Due to the big number of them, it is not viable to observe individual plots for each sample file in the same way made in Section 4.1.

With this scalability issue in mind, metadata information is inserted in the beginning of the files as shown in Fig. 4.11. This image presents the file format of the TCP/IP application layer payload defined for the current work. Such payload carries three different information about the file, which are: *"Sampling period* (T)*"*, occupying 2 bytes; *"Number of samples (N)"*, 4 bytes; and the actual *"Leakage current data"*, whose variable length is calculated as $2 \times N$.

Even though the ADC used has a 12-bit resolution, it is convenient to store each sample of the leakage current data in two bytes, because the minimum allocable resource of the API utilized to interact with the micro SD card is a byte. Consequently, the expected size of a whole sample file in bytes can be calculated as $f_{size} = 2 + 4 + 2 \times N$. Once $N = 80000$ is being used, then f_{size} is equal to 160006.

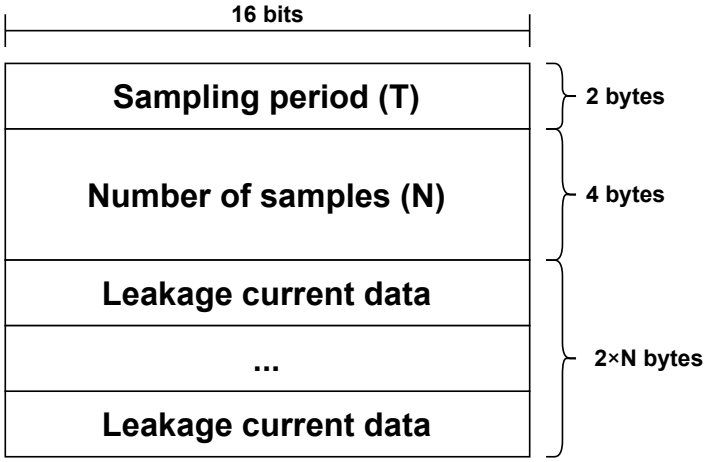


Figure 4.11: File format of the application payload.

The scatter plot from Fig. 4.12 shows the file sizes in bytes for the same data collection analyzed in Section 4.2.1. This time, all the considered files were received with the exact amount of files expected.

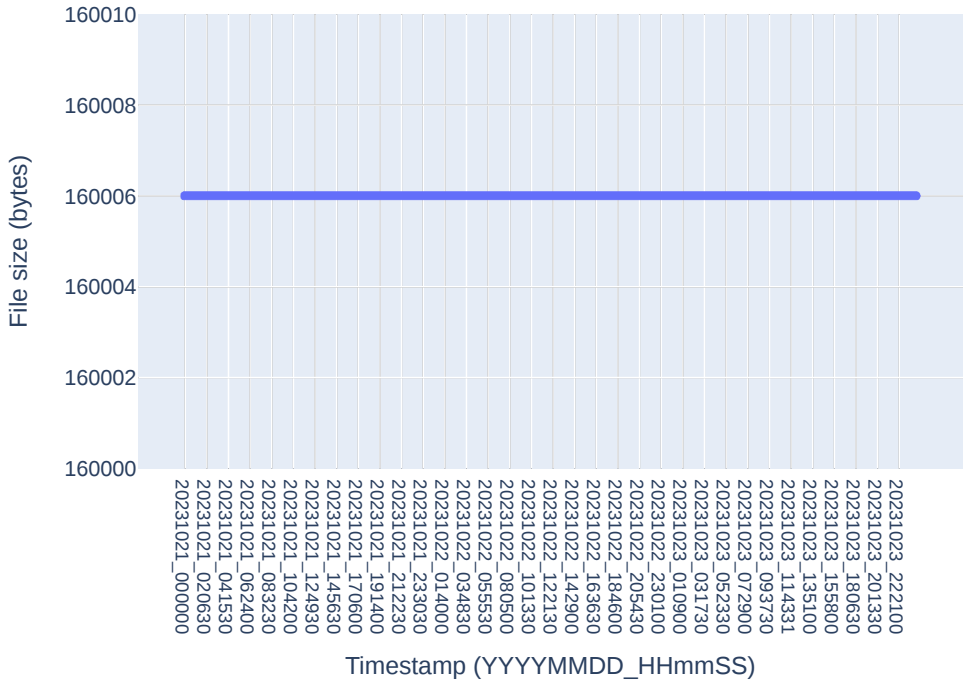


Figure 4.12: Scatter plot of the file sizes (in bytes) over 72 hours.

4.3 Wireless Network Coverage

In this section, we present the tests performed to evaluate the coverage of the wireless network designed for the project using COTS APs. At first, the tests were conducted in a controlled environment (Section 4.3.2), then also in-field (Section 4.3.3).

To enable these experiments, a mobile prototype was built based on the ESP32, which is the MCU used to control the communication module of the UMSCF, as described in Chapter 3.1. This prototype was developed in conjunction with a wireless network infrastructure and a client-server application running on the TCP/IP protocol stack in order to create a network infrastructure for the devices in this project.

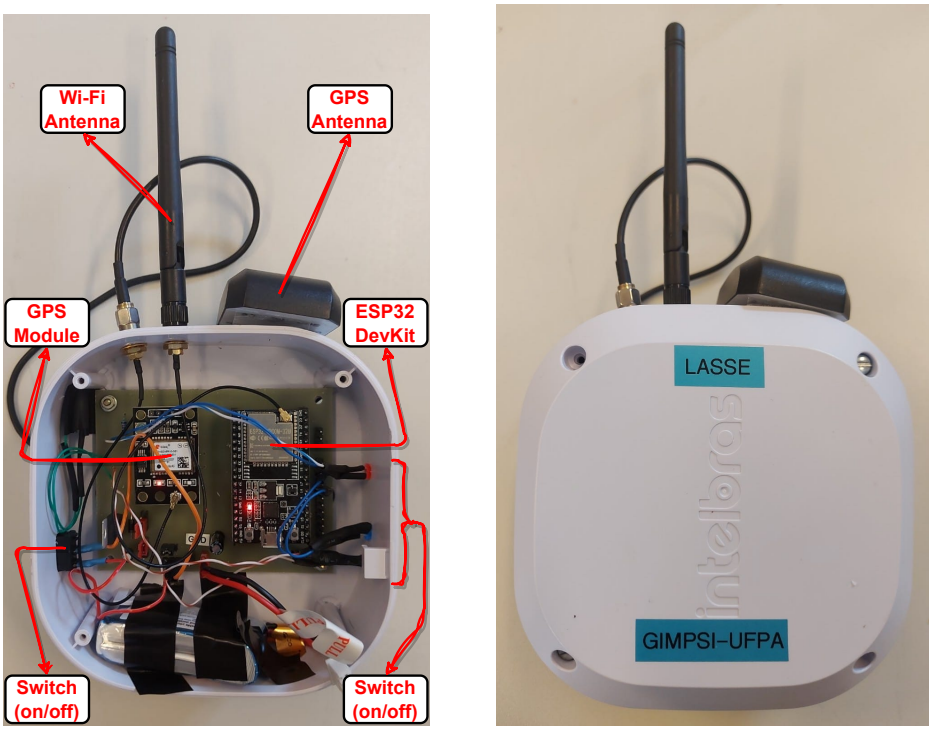
4.3.1 Hardware and Software Developed for the Network Validation Experiment

The hardware prototype for wireless communication consists of a rechargeable battery, a voltage regulator, a Gy-neo6mv2 GPS module and an ESP32 DevKit. Both the GPS and the MCU used external antennas, the latter using an omnidirectional Wi-Fi antenna (3dBi gain). Fig. 4.13 shows the hardware prototype mounted in the portable enclosure with and without the front cover – **(a)** and **(b)** respectively.

The firmware developed for ESP32 (client) and software for the local server were designed as finite state machines (FSMs) synchronized with each other in order to capture the received signal strength indicator (RSSI) and application data rate. Fig. 4.14 describes the expected sequence of client and server states defined to implement the expected behavior.

Note that Bluetooth and Position states exist only on the client side, while state Store is present only on the server side. The other states were positioned symmetrically in relation to the vertical axis of the image in order to highlight the synchronization between the client and server.

In state 0, the client waits until it receives a message via Bluetooth and, only then, tries to establish a connection with the server (state 1). Once connected, they both enter state 2, on which the MCU starts sending packets to the server during a pre-defined amount of time and stores the total data, in bytes, that was sent. Next (state 3), it starts receiving packets from the server for the same period of time and stores how many bytes were received.



(a) Without the front cover. **(b)** With the front cover.

Figure 4.13: Mobile enclosure used for the network validation experiment.

Once this part is finished, the client captures its coordinates from the GPS module through UART interface (state 4). At this point, in state 5, the RSSI value and timestamp in datetime format *(yyyy-mm-ddThh-mm-ss)* are obtained from the network. The timestamp is provided via NTP. Based on the number of bytes successfully sent and received, the device calculates the application UL and DL transfer rates in Mbps.

Still in state 5, all the above information is put into a JSON message and sent to the server: timestamp, latitude, longitude, RSSI, UL data rate, and DL data rate. Then, the server forwards the data to a database (state 6).

All stored data can be viewed via a web page, enabling a more detailed analysis of RSSI values and respective data rates on a map.

Lastly, to set up the Wi-Fi network, the TP-Link's EAP225-Outdoor [38] was used. This AP is designed for outdoor environments with an IP-65 enclosure, resistant to rain and other weather conditions. It has dual-band Wi-Fi, operating on 2.4GHz and 5GHz. In addition, it can be used to create a mesh network with multiple APs of the same model to extend the coverage area. The manufacturer's website states that field tests with the EAP225 provide coverage of at

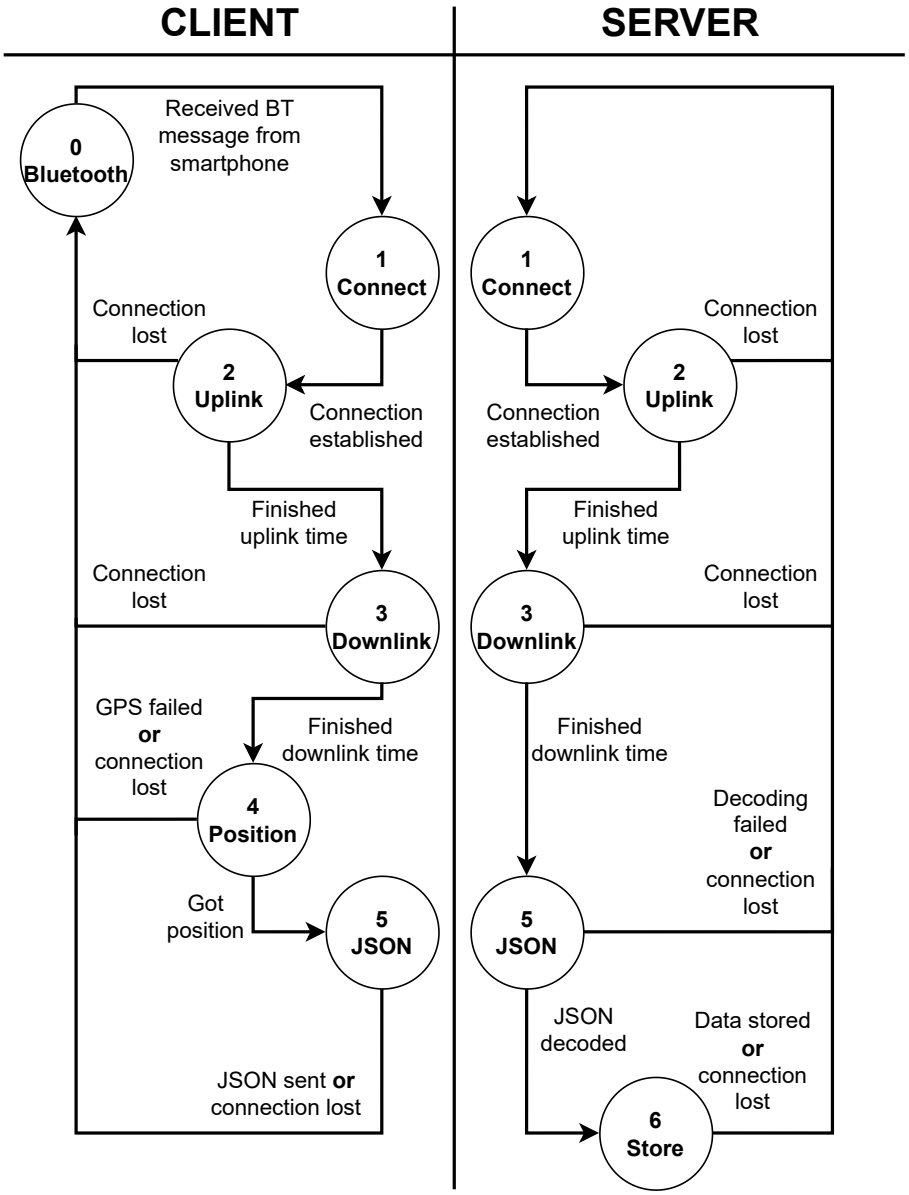


Figure 4.14: FSMs defined for the network validation experiment.

least 200 meters operating at 2.4GHz[1]. In all tests, it was configured to transmit at maximum power, which is 23dBm.

4.3.2 Results in a Controlled Environment

Tests were carried out at the Guamá Science and Technology Park with the objective of analyzing the Wi-Fi RSSI versus the distance between the hardware prototype and the EAP225-Outdoor APs comprising a mesh network. This way, it would be possible to verify the maximum

[1]Available at: https://www.tp-link.com/br/support/download/omada-software-controller/.

distances that the prototype could reach without signal loss. Most of the tests were carried out in LOS conditions or with little obstruction between the prototype and the AP. Results can be seen in Fig. 4.15.

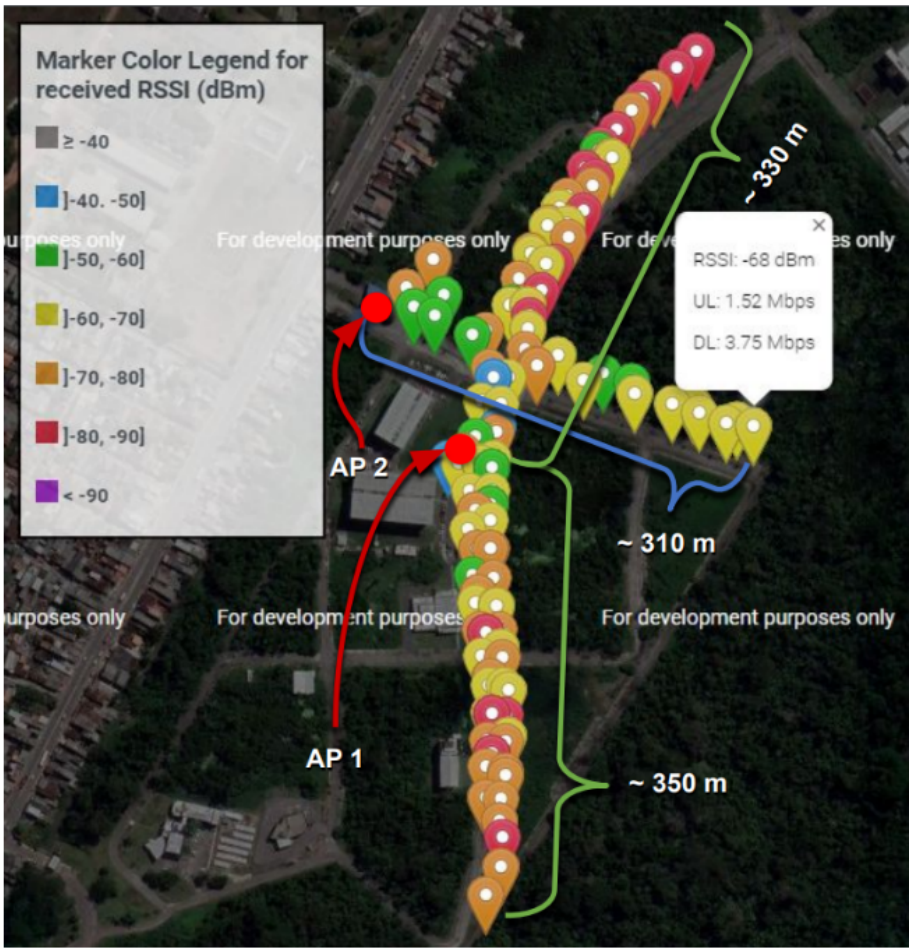


Figure 4.15: Results of the network validation experiment in a controlled environment.

This figure shows the positions of the APs and the RSSI values acquired during the tests. AP1 was connected to the LAN via a wired Ethernet network and AP2 was connected to the same network via the mesh network implemented with AP1. In other words, AP2 was only physically connected to a power cable. The APs were positioned in an open area with positions from which it there was LOS between them.

After deploying the Wi-Fi mesh network in an open area, the mobile prototype was put into operation. It is possible to get an indication of the power levels received by the colors of the markers, as shown in the legend of Fig. 4.15. Power levels above -70 dBm are considered good power levels for establishing a stable Wi-Fi wireless link, but it is still possible to maintain communication when the power level is above -90 dBm.

The prototype was able to communicate with AP1 (AP connected to the LAN via Ethernet cable) at a distance of approximately 350 meters and with AP2 at a distance of approximately 310 meters. It is worth noting that in the tests carried out with AP2, the prototype was in non-line-of-sight (NLOS) condition in relation to AP1 due to the presence of high vegetation obstructing the radio signal. In this way, it was possible to test the APs' mesh network with EAP225-Outdoor and the ESP32 MCU hardware prototype. Overall, the results were satisfactory for the requirements that were established.

4.3.3 Results In-Field

After the idea was tested in a controlled environment, it still had to be validated in-field, i.e. into the substation of "UTE Porto de Sergipe I" (Figs. 4.16 and 4.17). For that purpose, a similar deployment was installed there.

Figure 4.16: Panoramic view of the "UTE Porto de Sergipe I".
Source: ENEVA S.A.

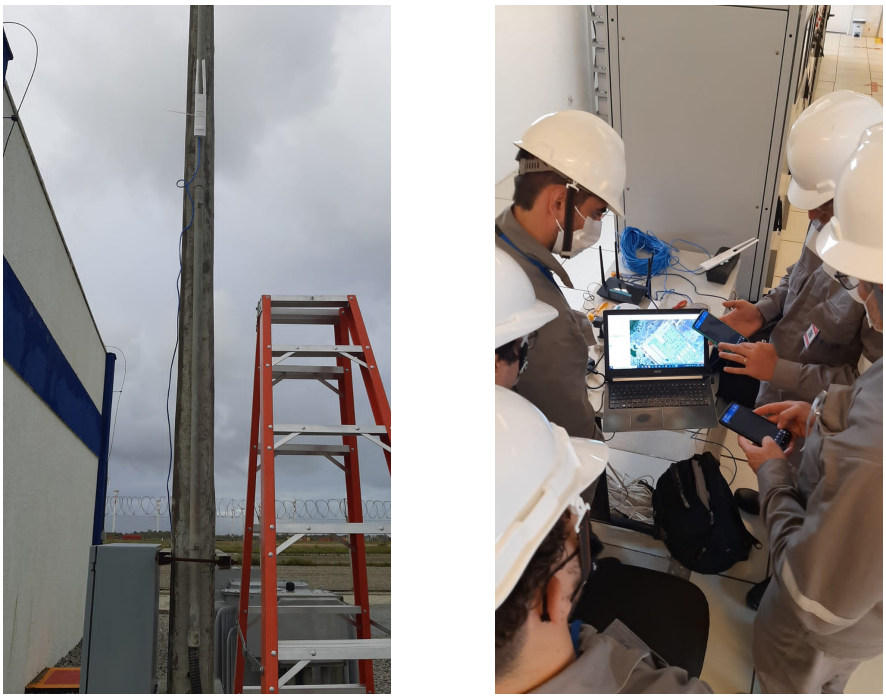
The tests described below were carried out with a single EAP225-Outdoor AP, positioned on the pole next to the control room at a height of approximately 3.5m from the ground, as shown in Fig. 4.18a. Due to the unavailability of an Ethernet AP inside the control room, a local network was set up so that the AP, the hardware prototype and the server could communicate inside the substation. The COTS "Action RF 1200" router from Intelbras was used as a Dynamic

Figure 4.17: Photograph close to the HV insulators of the "UTE Porto de Sergipe I" substation.
Source: ENEVA S.A.

Host Configuration Protocol (DHCP) server to provide IP for the network. The AP and the computer responsible for running the TCP/IP server were connected via Ethernet cable to the Intelbras router. The mobile prototype was connected to the AP via its Wi-Fi network, similarly to what was shown in Section 4.3.2, but with a single AP. Next, together with the team from the Federal University of Sergipe (UFS), measurements were taken along all the spots where the UMSCF is planned to be installed. Fig. 4.18b shows the members of both UFPA and UFS teams setting up the environment to take the measurements.

Fig. 4.19 shows the web application screen with the markers indicating the RSSI value captured along the route inside the substation. There were collected 51 RSSI and data rate data at different positions, with three captures showing a value greater than or equal to -50 dBm, four captures between -50 and -60 dBm, 24 RSSIs between -60 and -70 dBm, 20 captures between -70 and -80 dBm and only one capture between -80 and -90 dBm.

The tests carried out with the mobile prototype proved promising, as it was possible to connect it at distances up to 300 meters with the EAP225-Outdoor APs, as well as connecting the prototype to the mesh network deployed with it. In the tests carried out at "UTE Porto de Sergipe I", it was possible to connect the hardware prototype to the AP at all points in the substation.



(a) Placement of the AP besides the control room.

(b) GImpSI teams setting up the test environment.

Figure 4.18: Photographs taken during the network in-field tests.

Figure 4.19: Results of the network validation experiment in-field.

Chapter 5

Conclusion

This work presented the development of an IoT system for leakage current monitoring in HV insulators. The motivation for it came from the problem of optimizing the maintenance costs with insulators in power plants located near close to the sea, that suffer with corrosion and degradation caused by the salinity of the environment.

Nevertheless, most parts of the developed system – comprising both hardware and software – could be easily adapted to be applied to other common IoT contexts, such as mobility, smart homes/buildings, environment monitoring, medical and agriculture, among others [39].

Based on the results outlined in the text, it is evident that the system not only functions properly, but also effectively responds to unforeseen scenarios by offering potential solutions through the remote monitoring and management system.

5.1 Future Work

A set of improvements could be implemented as future works. For example, the hardware was originally planned to be able to supply the boards with supercapacitors in the event of a power failure. The POWER PCB was designed with this in mind, but this functionality was not thoroughly tested in the current project, and some adjustment is needed on the circuit in order to use it. In addition, part of the signal conditioning circuit is still under tests, and a newer version of that shall be produced soon.

As for the device firmware, energy and network efficiency questions can be further explored. An interesting feature would be to enter low power modes when the device is in *Rest* state, in order to enhance battery life. Another idea would be to exchange information with the

server only after a given number of cycles. Finding the optimum number would be a problem itself, in order to maximize energy and/or network efficiency while guaranteeing that the maximum capacity of the micro SD card is not exceeded, as well as the time between two adjacent sampling windows.

It is also possible to work on improvements for the software of the local server. A more user-friendly interface could be designed for the web platform for both monitoring and management of the IoT devices. Moreover, some actions that now require a human operator – as described in Chapter 3.3 – could be automated, such as restarting a specific remote node or formatting its micro SD card based on the information exchanged between the devices and the server.

Bibliography

[1] E. A. Byers, J. W. Hall, and J. M. Amezaga, "Electricity Generation and Cooling Water Use: UK pathways to 2050", *Global Environmental Change*, vol. 25, pp. 16–30, 2014.

[2] M. M. Hussain, S. Farokhi, S. G. McMeekin, and M. Farzaneh, "The Effects of Salt Contamination Deposition on HV Insulators under Environmental Stresses", in *2015 IEEE 11th International Conference on the Properties and Applications of Dielectric Materials (ICPADM)*, pp. 616–619, 2015.

[3] K. Belhouchet, *Contribution to the Study and Improvement of the Electric Performance of High Voltage Insulators*. Thesis, Université Ferhat ABBAS – Sétif 1, Feb. 2020. Accepted: 2020-02-26T08:32:13.

[4] N. Harid, A. C. Bogias, H. Griffiths, S. Robson, and A. Haddad, "A Wireless System for Monitoring Leakage Current in Electrical Substation Equipment", *IEEE Access*, vol. 4, pp. 2965–2975, 2016.

[5] S. S. Carvalho, "Desenvolvimento de um Sistema de Medição de Corrente de Fuga para Isoladores de Alta Tensão", Electronic Engineering, Universidade Federal de Sergipe, São Cristóvão, 2022.

[6] G. V. S. Oliveira, "Sistema de Medição de Picos de Corrente de Fuga para Isoladores de Alta Tensão", Master's thesis in Electrical Engineering, Universidade Federal de Sergipe, São Cristóvão, 2023.

[7] B. V. S. Araújo, "Monitoramento de Para-raios de ZnO com Uso de Redes Neurais Convolutivas e Processamento de Imagem", Master's thesis in Electrical Engineering, Universidade Federal de Sergipe, São Cristóvão, 2023.

[8] A. A. Salem, R. Abd-Rahman, S. A. Al-Gailani, M. S. Kamarudin, H. Ahmad, and Z. Salam, "The Leakage Current Components as a Diagnostic Tool to Estimate Contamination Level on High Voltage Insulators", *IEEE Access*, vol. 8, pp. 92514–92528, 2020.

[9] K. Sorri, N. Mustafee, and M. Seppänen, "Revisiting IoT Definitions: A framework towards comprehensive use", *Technological Forecasting and Social Change*, vol. 179, p. 121623, 2022.

[10] ITU-T Rec. Y.2060, "Overview of the Internet of Things", 2012.

[11] P. Horowitz and W. Hill, *The Art of Electronics; 3rd ed.* Cambridge: Cambridge University Press, 2015.

[12] Rhodes & Schwarz, "Understanding UART". Available at: https://www.rohde-schwarz.com/cz/products/test-and-measurement/essentials-test-equipment/digital-oscilloscopes/understanding-uart_254524.html.

[13] R. T. Fielding, M. Nottingham, and J. Reschke, "HTTP Semantics", Request for Comments RFC 9110, Internet Engineering Task Force, June 2022. Num Pages: 194.

[14] K. Mekki, E. Bajic, F. Chaxel, and F. Meyer, "Overview of Cellular LPWAN Technologies for IoT Deployment: Sigfox, LoRaWAN, and NB-IoT", in *2018 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 197–202, Mar. 2018.

[15] M. Mahmoud and A. Mohamad, "A Study of Efficient Power Consumption Wireless Communication Techniques/ Modules for Internet of Things (IoT) Applications", *Advances in Internet of Things*, vol. 06, pp. 19–29, Jan. 2016.

[16] I. Alaoui Ismaili, A. Azyat, N. Raissouni, N. Ben Achhab, C. Asaad, and M. Lahraoua, "Comparative Study of ZigBee and 6LoWPAN Protocols: Review", Jan. 2019.

[17] C. M. Ramya, M. Shanmugaraj, and R. Prabakaran, "Study on ZigBee Technology", in *2011 3rd International Conference on Electronics Computer Technology*, vol. 6, pp. 297–301, Apr. 2011.

[18] F. Adelantado, X. Vilajosana, P. Tuset-Peiro, B. Martinez, J. Melia-Segui, and T. Watteyne, "Understanding the Limits of LoRaWAN", *IEEE Communications Magazine*, vol. 55, pp. 34–40, Sept. 2017.

[19] L. Alliance, "A Technical Overview of LoRa and LoRaWAN", 2015.

[20] A. Lavric, A. Petrariu, and P. Valentin, "Long Range SigFox Communication Protocol Scalability Analysis Under Large-Scale, High-Density Conditions", *IEEE Access*, vol. PP, pp. 1–1, Mar. 2019.

[21] A. Ikpehai, B. Adebisi, K. Rabie, K. Anoh, R. Ande, M. Hammoudeh, H. Gacanin, and U. Mbanaso, "Low-Power Wide Area Network Technologies for Internet-of-Things: A Comparative Review", *IEEE Internet of Things Journal*, 2019.

[22] GSMA, "NB-IoT Deployment Guide to Basic Feature set Requirements", 2019.

[23] G. Leenders, G. Callebaut, G. Ottoy, L. van der perre, and L. Strycker, "Multi-RAT for IoT: The Potential in Combining LoRaWAN and NB-IoT", *IEEE Communications Magazine*, vol. 59, pp. 98–104, Dec. 2021.

[24] E. M. Migabo, K. D. Djouani, and A. M. Kurien, "The Narrowband Internet of Things (NB-IoT) Resources Management Performance State of Art, Challenges, and Opportunities", *IEEE Access*, vol. 8, pp. 97658–97675, 2020.

[25] P. Sharma and G. Singh, "Comparison of Wi-Fi IEEE 802.11 Standards Relating to Media Access Control Protocols", *International Journal of Computer Science and Information Security,*, vol. 14, pp. 856–862, Oct. 2016.

[26] G. Hiertz, T. Denteneer, L. Stibor, Y. Zang, X. Costa-Pérez, and B. Walke, "The IEEE 802.11 Universe", *Communications Magazine, IEEE*, vol. 48, pp. 62–70, Feb. 2010.

[27] F. Siddiqui, S. Zeadally, and K. Salah, "Gigabit Wireless Networking with IEEE 802.11ac:Technical Overview and Challenges", *Journal of Networks*, vol. 10, Apr. 2015.

[28] J. Brinkhoff and J. Hornbuckle, "Characterization of WiFi Signal Range for Agricultural WSNs", in *2017 23rd Asia-Pacific Conference on Communications (APCC)*, pp. 1–6, Dec. 2017.

[29] J.-S. Lee, Y.-W. Su, and C.-C. Shen, "A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi", in *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, pp. 46–51, Nov. 2007. ISSN: 1553-572X.

[30] A. Mahmood, N. Javaid, and S. Razzaq, "A Review of Wireless Communications for Smart Grid", *Renewable and Sustainable Energy Reviews*, vol. 41, pp. 248–260, Jan. 2015.

[31] Z. Shelby, "Introduction", in *6LoWPAN: The Wireless Embedded Internet*, p. 244, Wiley Telecom, 2009.

[32] P. K. Kamma, C. R. Palla, U. R. Nelakuditi, and R. S. Yarrabothu, "Design and Implementation of 6LoWPAN Border Router", *2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN)*, pp. 1–5, July 2016. ISBN: 9781467389754.

[33] G. A. Naidu, J. Kumar, V. Garudachedu, and P. R. Ramesh, "6LoWPAN Border Router Implementation for IoT Devices on RaspberryPi", *SSRN Electronic Journal*, 2018.

[34] S. Labs, "AN1138: Zigbee Mesh Network Performance", 2021.

[35] D. Gislason, *Zigbee Wireless Networking*. AbeBooks, 2008.

[36] H. Hu, Y. Liu, Y. Xu, and W. Chen, "A graphical display method for low performance embedded systems based on hybrid programming and double buffering technology", in *2022 International Conference on Cloud Computing, Big Data Applications and Software Engineering (CBASE)*, pp. 14–17, 2022.

[37] V. Nguyen, S. Deeds-Rubin, T. Tan, and B. W. Boehm, "A SLOC Counting Standard", 2007.

[38] TP-Link, "Omada EAP - Business Wi-Fi Series", 2023.

[39] A. Khanna and S. Kaur, "Internet of Things (IoT), Applications and Challenges: A comprehensive review", *Wireless Personal Communications*, vol. 114, pp. 1687–1762, Sept. 2020.

Appendices

Appendix A

UMSCF Operation Guide

A.1 Components

Fig. A.1 shows the corresponding locations of each component highlighted in Table A.1. This table lists the main removable board components that are essential for the operation of the UMSCF. If they need to be replaced, it is recommended to use the models described there.

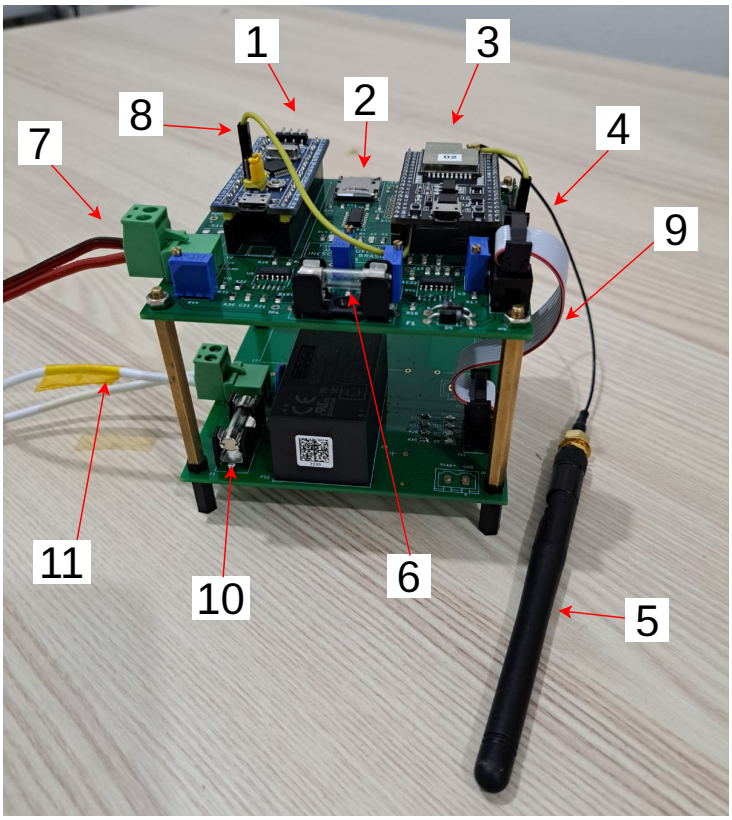


Figure A.1: Identification of the removable components of the UMSCF.

Table A.1: Removable components of the UMSCF.

#	Board	Component	Model
1	DAQ	STM23 DevKit	Bluepill STM32F103C8T6
2	DAQ	Micro SD card	32 GB Class 10 micro SD
3	DAQ	ESP32 DevKit	ESP32-WROOM-32UE
4	DAQ	Antenna cable	Pigtail to RP-SMA
5	DAQ	Antenna	2.4GHz 50Ω antenna
6	DAQ	Fuse	*Fast action 30mA 250V
7	DAQ	Input signal cable	**Born 5.08mm female 2 vias 90º
8	DAQ	Jumper	Female-to-female
9	DAQ/POWER	Internal supply cable	Flat Ribbon (10 vias 1.27mm)
10	POWER	Fuse	Glass fuse 500mA 250V
11	POWER	External supply cable	**Born 5.08mm female 2 vias 90º

* On the underside of the DAQ PCB it is possible to solder an surface-mount device (SMD) fuse of faster-action than the traditional glass fuse.

** There are several compatible options; for instance, the 691352710002 model from Wurth Elektronik.

A.2 Warnings

A few warnings should be taken under consideration to guarantee the safety for those operating the UMSCF.

Warnings:

- Before connecting the board to the power supply, please check if all components are correctly positioned on the board;
- Be careful when touching the board when it is connected to the power supply due to the high voltage area located on the underside of the POWER board, as delimited in the “High Voltage” area shown in the top right-hand corner of Fig. A.2;
- If any of the board’s removable components need to be replaced, follow the recommendations in Table A.1, in which the model is referenced.

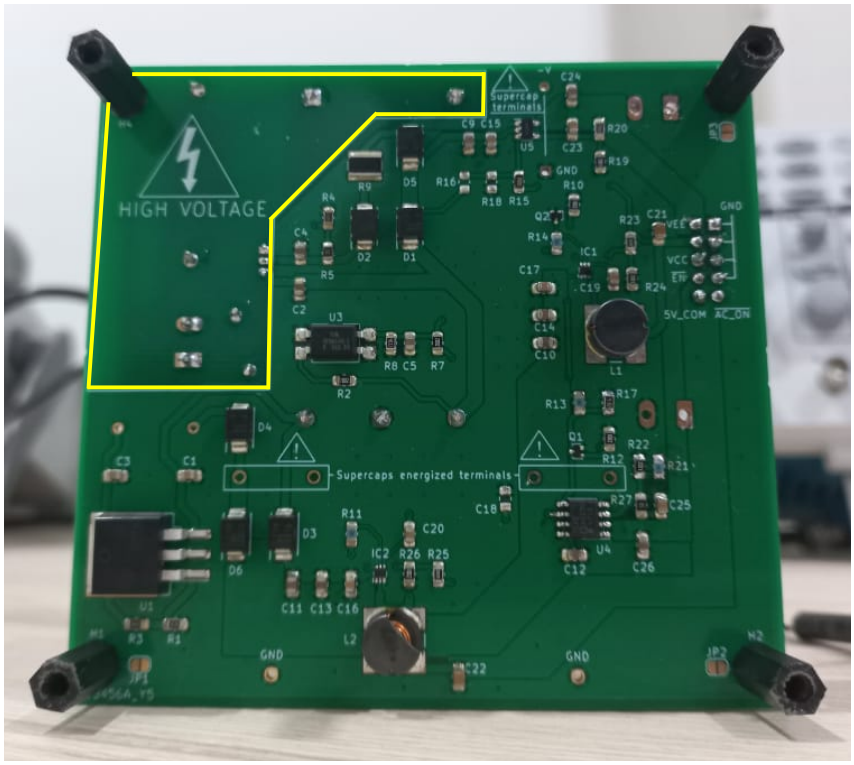


Figure A.2: "HIGH VOLTAGE" region of the POWER PCB.

A.3 Mounting

The step-by-step assembly of the equipment is described below.

The steps for installing the micro SD card must first be carried out by suspending the top part of the card module. The card must then be inserted into the part of the module that has been suspended. Finally, the upper part of the module, with the card in place, must be inserted into the lower part of the module, as shown in Fig. A.3.

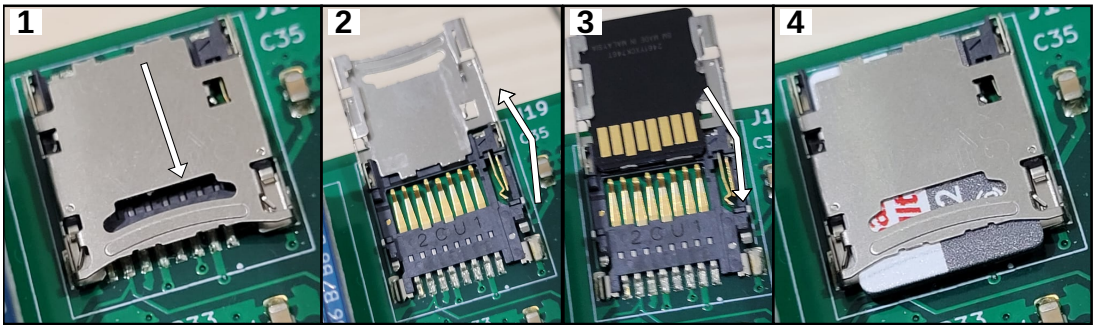


Figure A.3: Steps for inserting the micro SD card into the UMSCF.

The external antenna must first be attached to the pigtail cable via the SMA connectors. Then connect the other end of the pigtail cable to the MHF/U.FL connector on the ESP32 MCU,

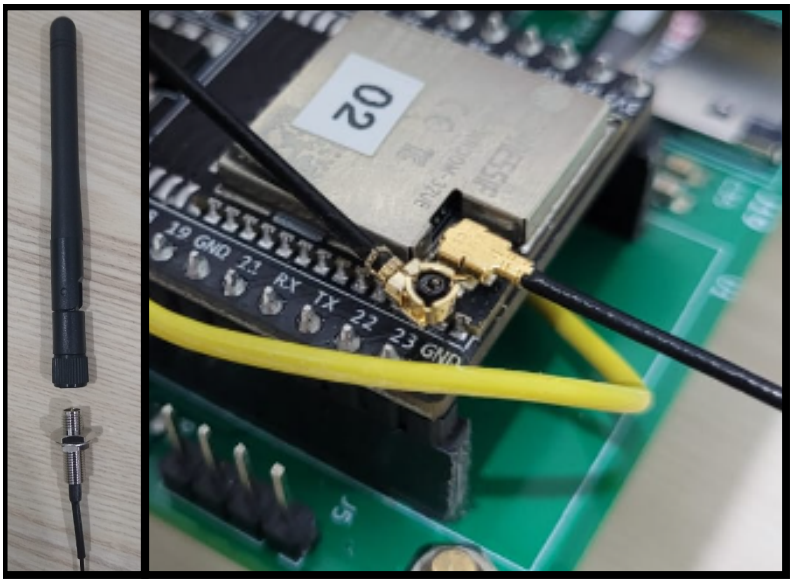


Figure A.4: Part of the UMSCF where the antenna cable should be plugged.

as shown in Fig. A.4.

Once the removable components have been placed, the Bluepill board (STM32) boot pins must be set up. Connect BOOT1 to GND and BOOT0 to the J3 connector next to the ESP32 MCU, as shown in Fig. A.5.

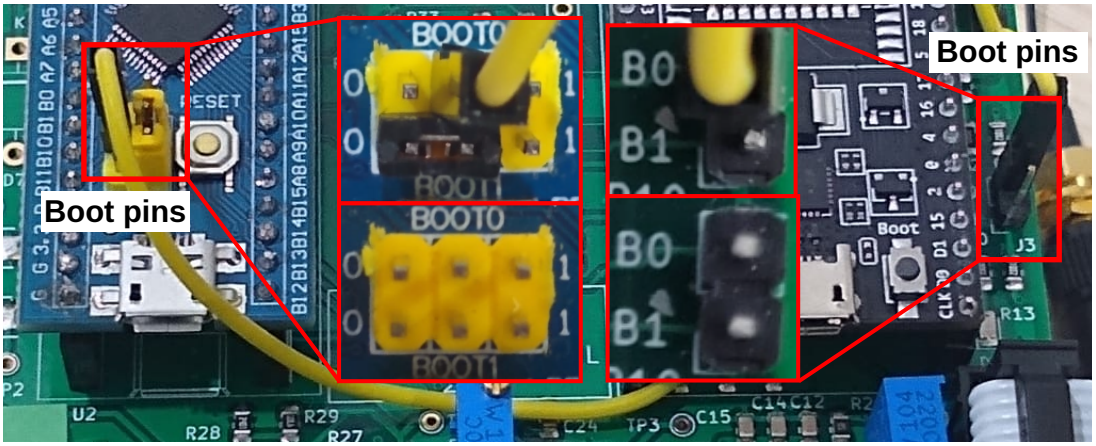


Figure A.5: Connection of BOOT0 and BOOT1 pins of the STM32.

The STM32 MCU Bluepill board has two BOOT configuration pins (BOOT0 and BOOT1), which are used to configure the device's BOOT mode. For example, BOOT0 is used to switch between the microcontroller's execution and programming modes. As shown in Fig. A.5, the two pins are located on the top of the STM32 MCU board (Bluepill board) with three contacts for each. As the STM32 is programmed by the other ESP32 MCU, the Bluepill's BOOT0 must

be connected to the B0 signal that is controlled by the ESP32. In this way, the ESP32 can control the STM32's operating mode.

Then, connect the power cable to the connectors indicated (see Fig. A.6). The lower PCB generates the DC voltages for the data acquisition and communication board from an AC supply (110 - 240V 50/60Hz).

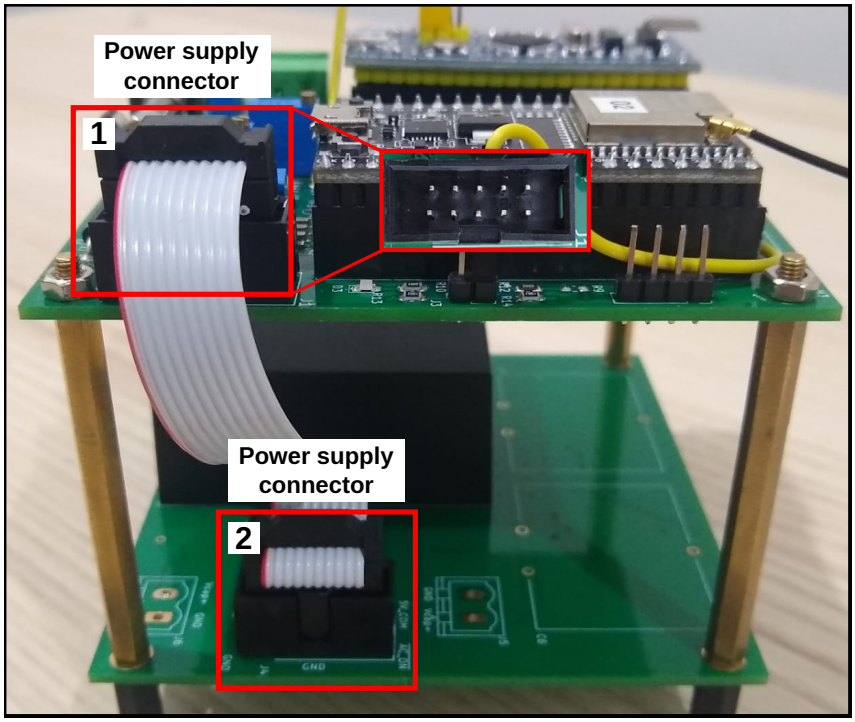


Figure A.6: Power supply link from one PCB (POWER) to the other (DAQ).

To protect the on-board electronics against overcurrents, fuses must be used, as shown in Fig. A.7. On the POWER board (bottom board), a 500mA fuse is used to protect the power supply circuits. On the DAQ board (top board), a 30mA glass fuse is used or an SMD fuse that can be soldered to the underside of the data acquisition board.

Next, connect the output of the leakage current sensor to the green connector on the DAQ board (see item 1 in Fig. A.8).

Finally, plug the power cable into the connector located on the POWER board (see item 2 in Fig. A.8) and into the mains socket, making sure that the mains voltage is within the range 110 to 240 Vac.

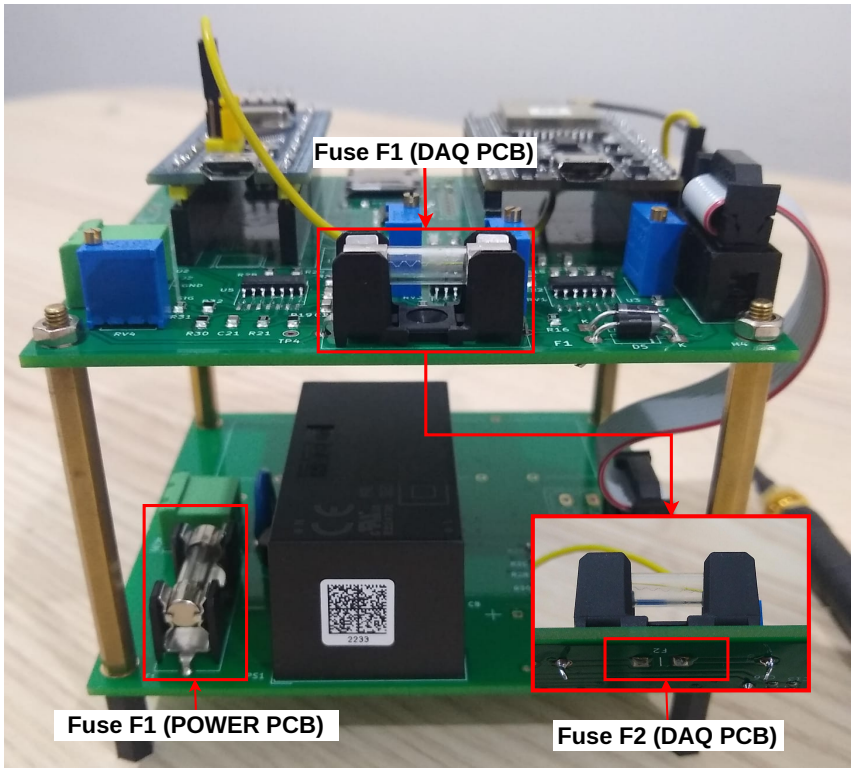


Figure A.7: Fuse arrangement.

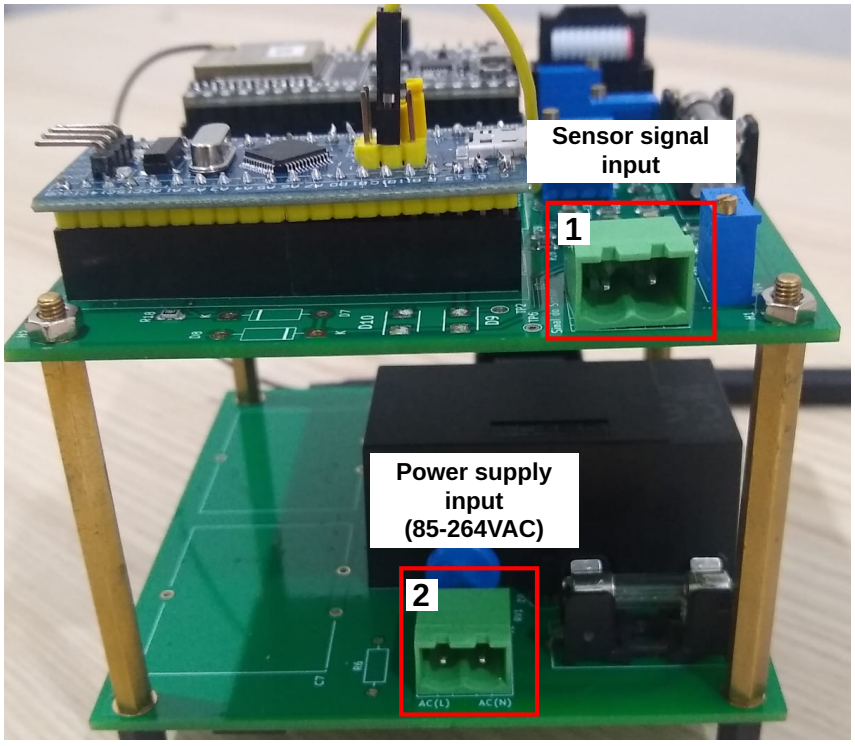


Figure A.8: External input connectors: (1) Leakage current sensor signal input; (2) Power supply input.

A.4 Operation

Once connected, the UMSCF will perform its functions autonomously, without the need for any additional configurations. Periodically, the device captures samples from the current sensor via the ADC of the signal conditioning module, saves it on the memory card, connects to an HTTP server via a Wi-Fi wireless interface and sends all the captured data, as well as searching for new firmware updates for the two MCUs present in the device. If the Wi-Fi connection or communication with the server fails, the data is saved on the local memory card until the next communication opportunity is successfully executed.

For the purposes of testing and debugging the hardware and firmware, the device has connectors that allow serial communication. The J5 connector allows connection to the ESP32's serial output, as well as a GPIO pin for testing. Fig. A.9 shows how to connect it to a USB-Serial module that allows you to monitor the device's log messages.

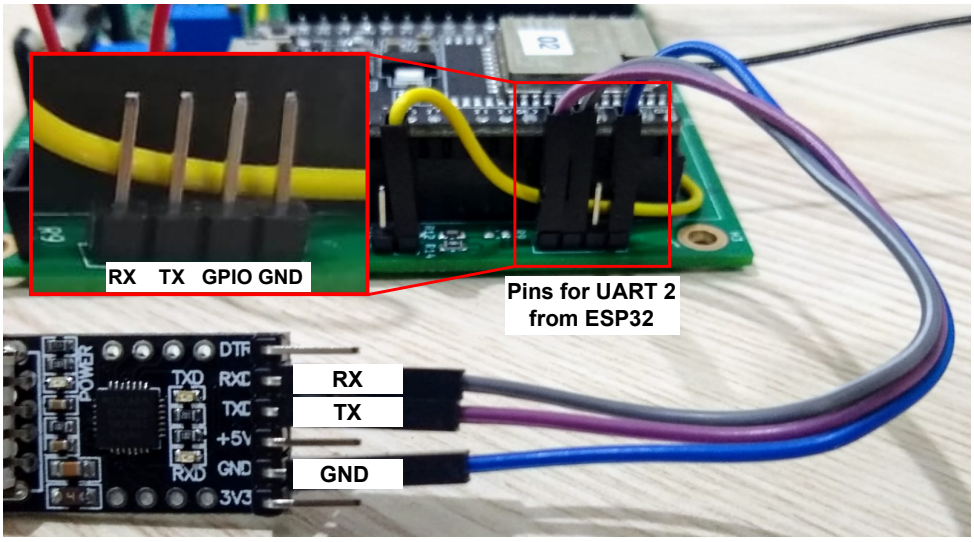


Figure A.9: Serial output for the ESP32 MCU.

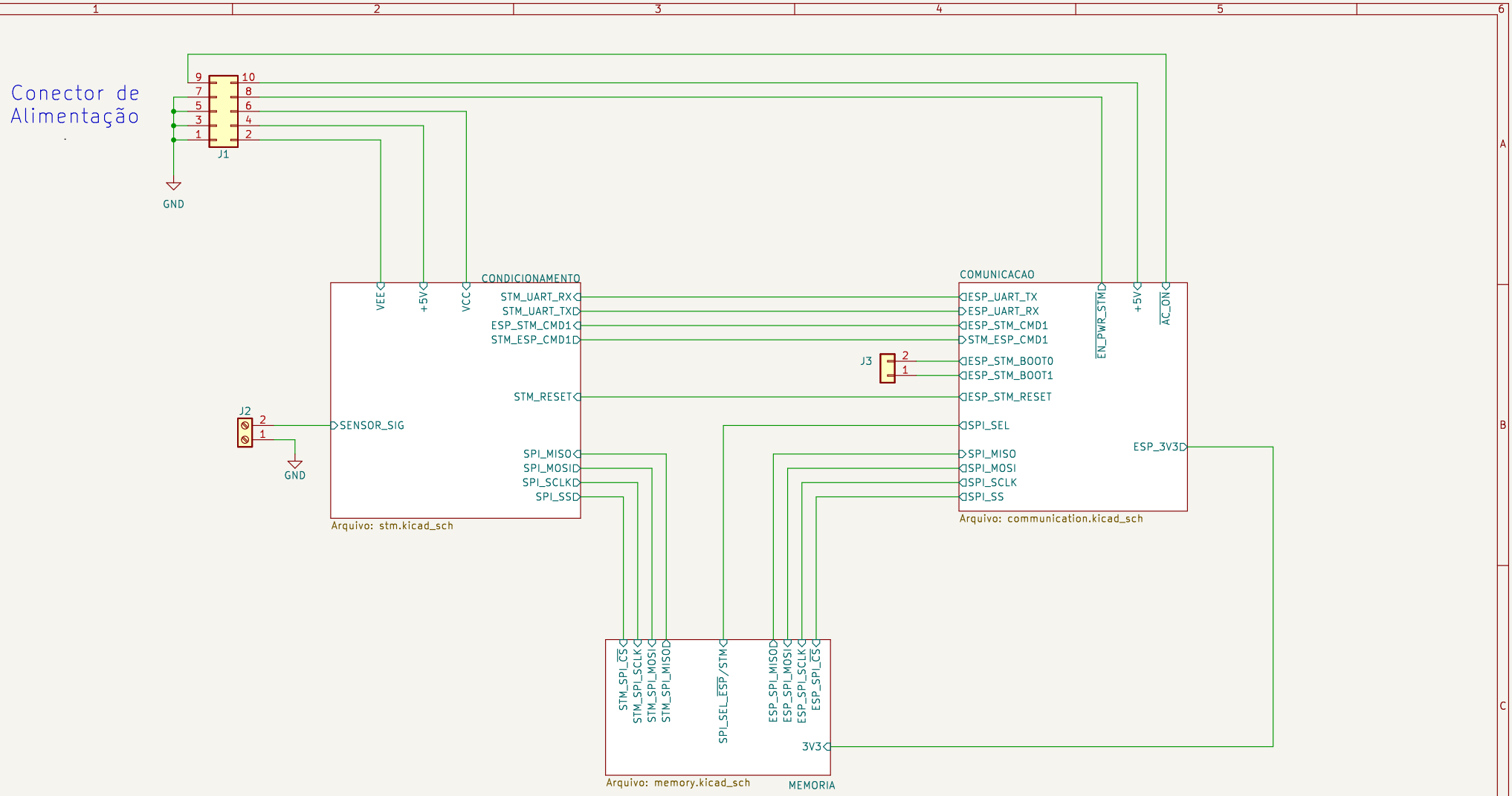
Finally, DAQ board has three LEDs connected to the ESP32 MCU (D1, D2 and D3) and one LED connected to the STM32 MCU (D4). The state of the LEDs is linked to some state of the firmware's finite state machine, as described in Table A.2, where the values 0 and 1 indicate the LED is off or on respectively. When the MCUs are switched off, all the LEDs remain off. The states referenced in this table are explained in Section 3.1.4.4.

Table A.2: List of signal LED states in each of the UMSCF possible state.

	LEDs connected to ESP			**LEDs connected to STM**
UMSCF State	**D1**	**D2**	**D3**	**D4**
OFF	0	0	0	0
SETUP	0	0	1	Oscillates at 1Hz
SAMPLE	0	1	0	Oscillates at 5Hz
CONNECT	0	1	1	Oscillates at 1Hz
TRANSMIT	1	0	0	Oscillates at 1Hz
UPDATE_FIRMWARE_STM	1	0	1	Undefined
UPDATE_FIRMWARE_ESP	1	1	0	Oscillates at 1Hz
REST	1	1	1	Oscillates at 1Hz

Appendix B

Project of the Data Acquisition PCB



Unidade microprocessada para sensores de corrente de fuga (UMSCF)

UFPA-UFS-CELSE

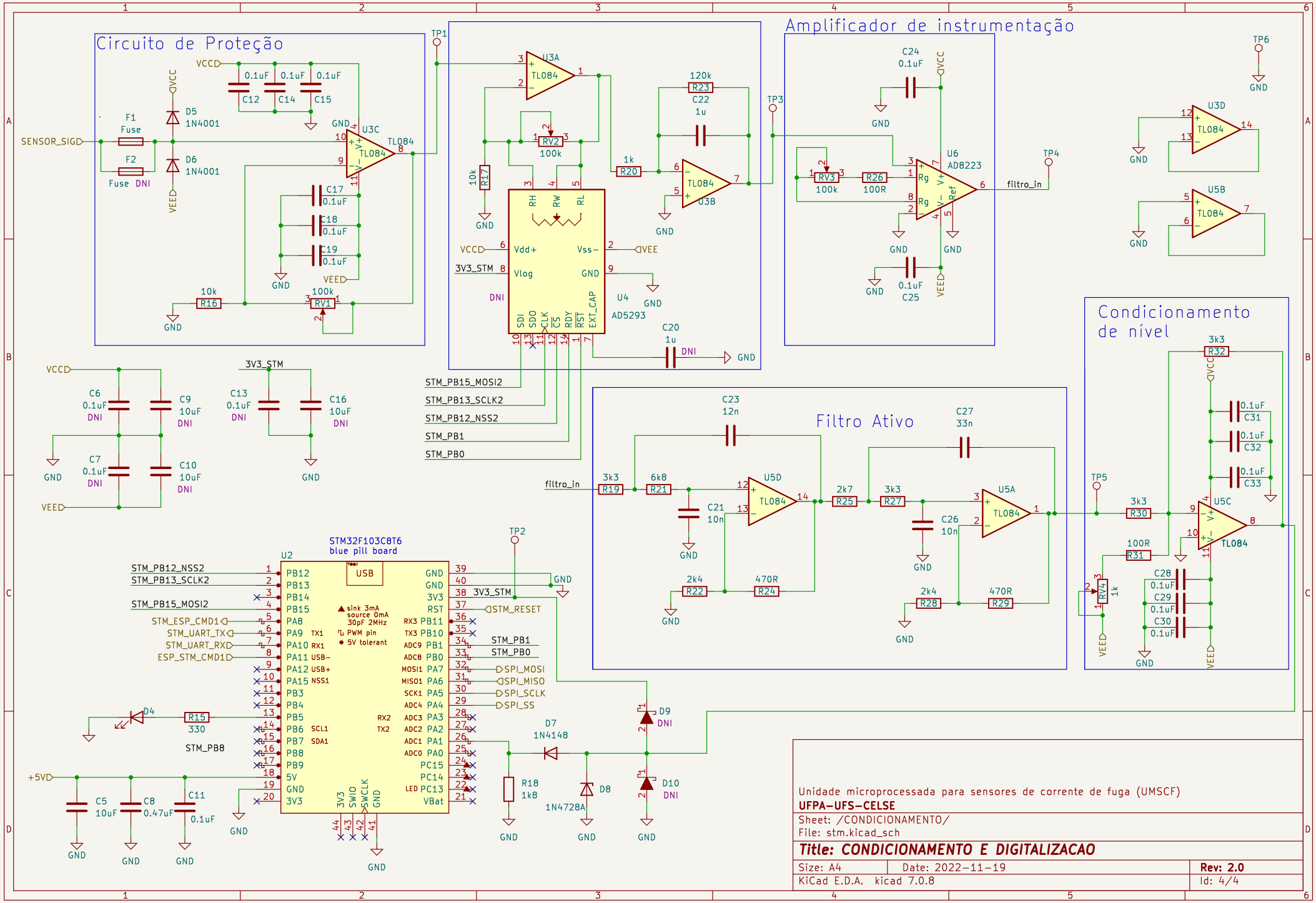
Sheet: /

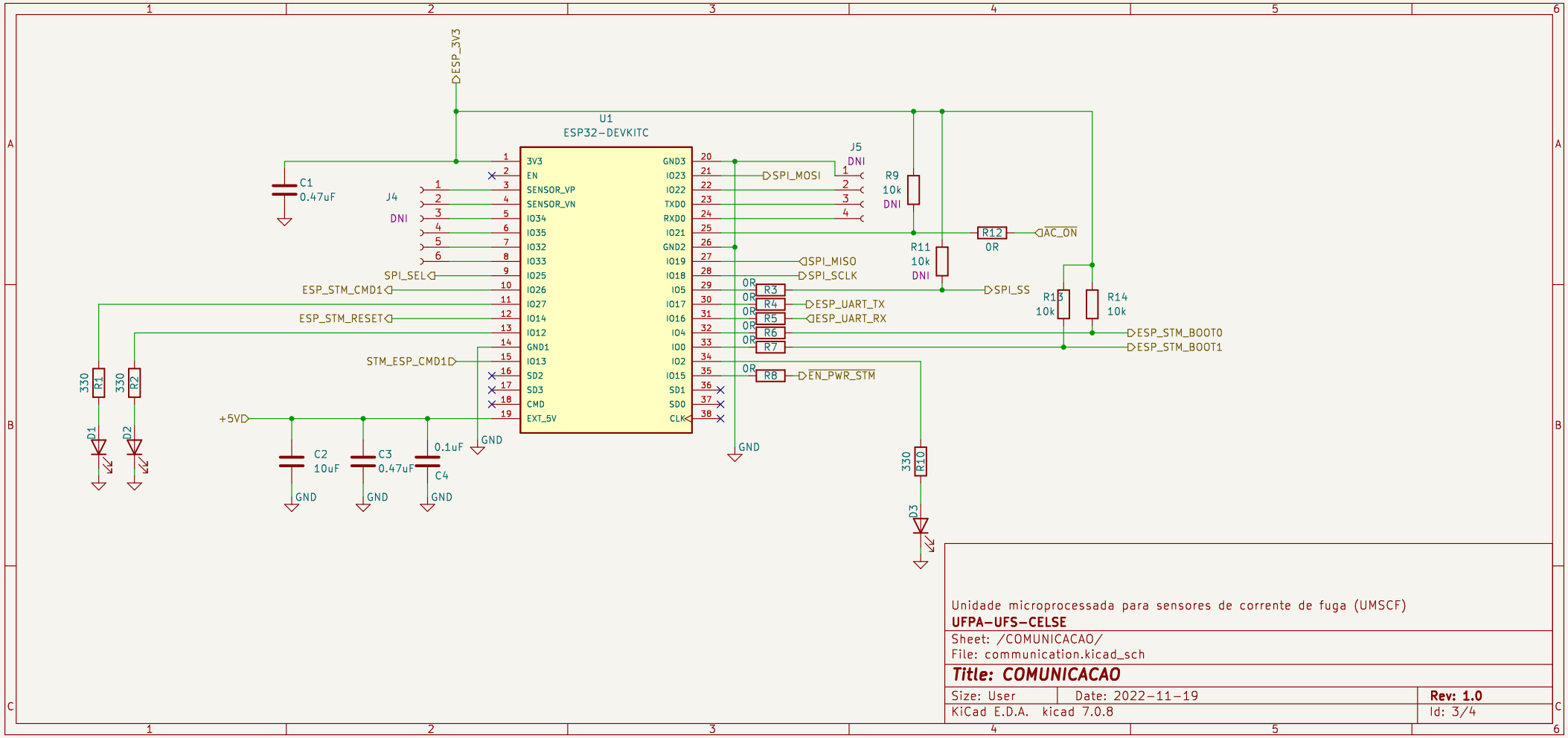
File: pcb-gimpsi.kicad_sch

Title: DIAGRAMA DE BLOCOS

Size: A4	Date: 2022-11-19	**Rev: 1.0**
KiCad E.D.A. kicad 7.0.8		Id: 1/4

Ajuste da Resposta em Frequência





Unidade microprocessada para sensores de corrente de fuga (UMSCF)

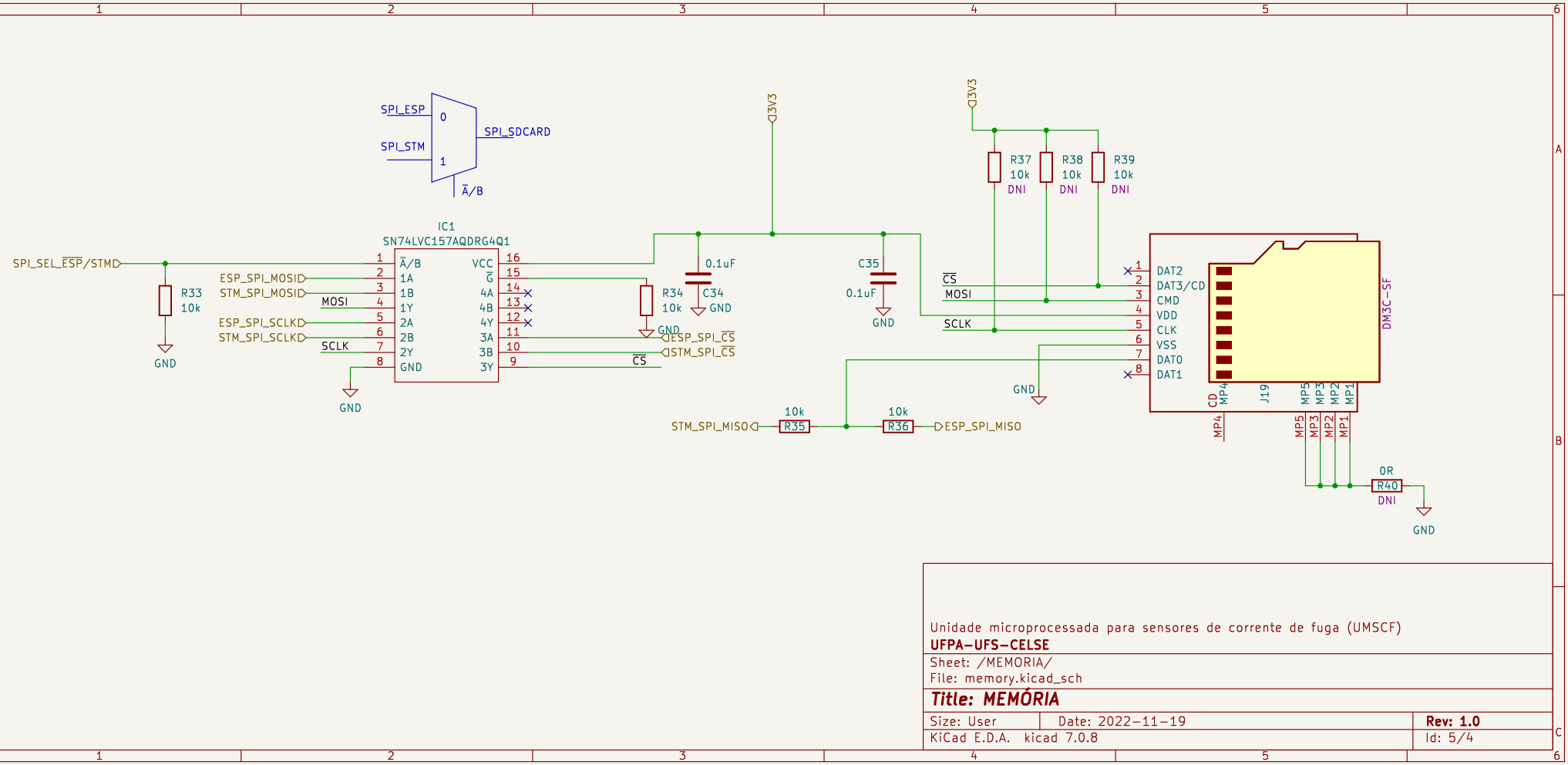
UFPA-UFS-CELSE

Sheet: /COMUNICACAO/

File: communication.kicad_sch

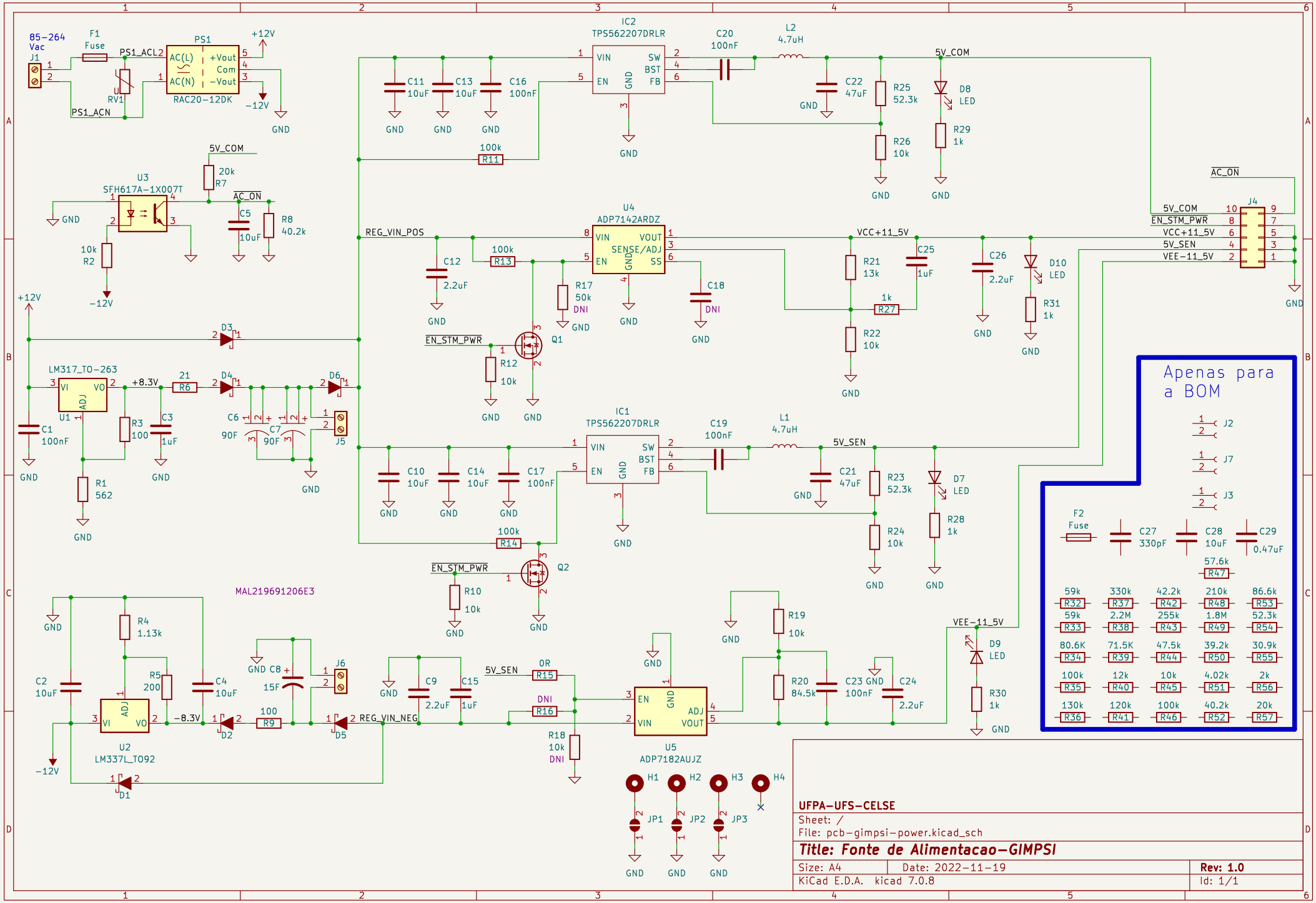
Title: COMUNICACAO

Size: User	Date: 2022-11-19	**Rev: 1.0**
KiCad E.D.A. kicad 7.0.8		Id: 3/4


SPI_ESP
SPI_STM
0
1
SPI_SDCARD
A/B
IC1
SN74LVC157AQDRG4Q1
SPI_SEL_ESP/STM
R33
10k
GND
ESP_SPI_MOSI
STM_SPI_MOSI
MOSI
ESP_SPI_SCLK
STM_SPI_SCLK
SCLK
3V3
0.1uF
R34
10k
C34
ESP_SPI_CS
STM_SPI_CS
CS
C35
R37
10k
DNI
R38
10k
DNI
R39
10k
DNI
DAT2
DAT3/CD
CMD
VDD
CLK
VSS
DAT0
DAT1
DM3C-SF
J19
STM_SPI_MISO
R35
R36
ESP_SPI_MISO
R40
0R
Unidade microprocessada para sensores de corrente de fuga (UMSCF)
UFPA–UFS–CELSE
Sheet: /MEMORIA/
File: memory.kicad_sch
Title: MEMÓRIA
Size: User
Date: 2022–11–19
Rev: 1.0
KiCad E.D.A. kicad 7.0.8
Id: 5/4

Appendix C

Project of the Power Supply PCB


85-264 Vac
J1
F1 Fuse
PS1_ACL
PS1_ACN
PS1
AC(L)
AC(N)
+Vout
Com
-Vout
RAC20-12DK
RV1
+12V
-12V
GND
IC2
TPS562207DRLR
VIN
EN
SW
BST
FB
C11 10uF
C13 10uF
C16 100nF
C20 100nF
L2 4.7uH
C22 47uF
R25 52.3k
R26 10k
D8 LED
R29 1k
R11 100k
5V_COM
AC_ON
U3
SFH617A-1X007T
20k R7
C5 10uF
R8 40.2k
10k R2
U4
ADP7142ARDZ
VIN
VOUT
SENSE/ADJ
EN
GND
SS
REG_VIN_POS
C12 2.2uF
R13 100k
R17 50k DNI
C18 DNI
VCC+11_5V
R21 13k
C25 1uF
R27 1k
R22 10k
C26 2.2uF
D10 LED
R31 1k
J4
5V_COM
EN_STM_PWR
VCC+11_5V
5V_SEN
VEE-11_5V
Q1
R12 10k
D3
LM317_TO-263
U1
+8.3V
R6 21
D4
D6
C1 100nF
R3 100
C3 1uF
R1 562
C6 90F
C7 90F
J5
IC1
TPS562207DRLR
C10 10uF
C14 10uF
C17 100nF
C19 100nF
L1 4.7uH
C21 47uF
R23 52.3k
R24 10k
D7 LED
R28 1k
5V_SEN
R14 100k
Q2
R10 10k
MAL219691206E3
R4 1.13k
C2 10uF
R5 200
C4 10uF
C8 15F
J6
U2
LM337L_TO92
-8.3V
R9 100
D1
D2
D5
REG_VIN_NEG
C9 2.2uF
C15 1uF
5V_SEN
0R
R15
DNI
R16
R18 10k DNI
U5
ADP7182AUJZ
EN
GND
ADJ
VIN
VOUT
R19 10k
R20 84.5k
C23 100nF
C24 2.2uF
VEE-11_5V
D9 LED
R30 1k
H1
H2
H3
H4
JP1
JP2
JP3
Apenas para a BOM
J2
J7
J3
F2 Fuse
C27 330pF
C28 10uF
C29 0.47uF
R47 57.6k
R32 59k
R37 330k
R42 42.2k
R48 210k
R53 86.6k
R33 59k
R38 2.2M
R43 255k
R49 1.8M
R54 52.3k
R34 80.6K
R39 71.5K
R44 47.5k
R50 39.2k
R55 30.9k
R35 100k
R40 12k
R45 10k
R51 4.02k
R56 2k
R36 130k
R41 120k
R46 100k
R52 40.2k
R57 20k
UFPA-UFS-CELSE
Sheet: /
File: pcb-gimpsi-power.kicad_sch
Title: Fonte de Alimentacao-GIMPSI
Size: A4
Date: 2022-11-19
Rev: 1.0
KiCad E.D.A. kicad 7.0.8
Id: 1/1