POLITECNICO DI TORINO

Master Degree Course in Electronic Engineering

Master Degree Thesis

Trace&Follow: Modular Power Monitoring Device Based on Open Source Software

Supervisor

Prof. Giovanna TURVANI

Candidate

Federico PECORARA

ACADEMIC YEAR 2022/2023

Summary

Over the past two years, the cost of electrical energy has consistently increased due to the war declared in February 2022 between Russia and Ukraine, and the resulting complex global geopolitical situation. Consequently, companies have felt the need to raise the price of their products as a countermeasure to the escalation of production costs. They require a more precise approach to determine how these cost increases impact the manufacturing of each specific item. This not only helps them better justify price increases on their products, but also provides a transparent explanation to their customers. The device developed in this thesis, named Trace&Follow, focuses on measuring power consumption. It is a modular solution that can be installed on various industrial machines, including welding machines, presses, laser cutting machines and many others. Trace&Follow consists of a baseboard and different add-on boards that expand its functionalities. Data can be sent to a cloud platform over different protocols: Wi-Fi, Ethernet, LoRaWAN, and 4G. The first two are integrated into the baseboard, while an expansion board is required for the others. Once the collected data reach the cloud platform, they are filtered, sorted, and stored into an internal database. They can be retrieved at a later time by an application and displayed to the user. Additionally, commands can be sent to the platform, which will then forward them to the device.

The Trace&Follow board feature an ESP32 microcontroller from Espressif Systems. It is built upon The Xtensa® dual core 32-bit LX6 microprocessor, a highly flexible and energy-efficient microprocessor architecture developed by Cadence Design Systems. The project's software components rely entirely on open-source code. Espressif offers its development tools, SDKs, and microcontroller core library as open source. Furthermore, Espressif microcontrollers are supported within the Arduino ecosystem, which encompasses a vast range of open-source libraries that can be utilized free-of-cost. ThingsBoard, an open-source IoT platform, is employed as the cloud platform. This platform serves for data collection, processing, visualization, and device management. When using LoRaWAN connectivity, an intermediary software is required to convert physical frames into a format that can be received by the cloud. This software, known as ChirpStack, is fully open source.

Firmware running on the device strongly relies on FreeRTOS real-time operating system and it is coded using Arduino programming language, a variant of the C++ language that widely uses the concept of classes and objects to achieve better code organization.

This work begins with an overview of the company where this project has been carried out and a general introduction to the whole project in chapter 1.
Chapter 2 introduces the previously developed device and explain the motivation behind the needs of a new platform development.
Chapter 3 outlines the device's hardware, from the prototype to the final PCB.
Chapter 4 focuses on how firmware running on the board is organized. A brief overview of software running on the cloud is presented.
Chapter 5 shows an installation of the device onto a welding machine.
Chapter 6 explores potential future developments and board costs.

Acknowledgements

"To all those who made this work possible, starting with my supervisor Prof. Turvani and the entire IDT organization for welcoming and allowing me to develop this project. To my family and particularly my sister Camilla for helping me whenever I needed it. To Carola and all my friends for their unwavering support throughout this long journey. And last but not least, to my colleagues, without whom I could not have made it to the end."

Table of Contents

List of Tables

List of Figures

Abbreviations

3P3W 3-Phase-3-Wire

3P3W 3-Phase-4-Wire

AC Alternating Current

ADC Analog-to-Digital Converter

BLE Bluetooth Low Energy

CAN Controller Area Network

CT Current Transformers

DAC Digital-to-Analog Converter

DC Direct Current

I/O Inputs/Outputs

I2C Inter-Integrated Circuit

IC Integrated Circuit

IDC Insulation-Displacement Connector

IoT Internet Of Things

JSON JavaScript Object Notation

LED Light-Emitting Diode

LDO Low-Dropout

LoRaWAN Long Range Wide Area Network

MAC Media Access Control

Mbps Megabits per second

MQTT Message Queuing Telemetry Transport

NC Normally Closed

NO Normally Open

OTA Over-the-air

PCB Printed Circuit Board

PCIe Peripheral Component Interconnect Express

SPI Serial Peripheral Interface

UART Universal Asynchronous Receiver-Transmitter

MCU Microcontroller

Chapter 1

Introduction

1.1 The company

IDT Solution was founded in 2016 with an innovative aim: to introduce "open source" platforms to the industrial automation sector, a relatively unexplored domain at that time. The company began by utilizing Arduino boards, which were well-recognized within the hobbyist community, and subsequently designed the necessary electronics to adapt them for the industrial environment. IDT's main challenge has always been to convince companies that, with the proper support structures, it is possible to employ these platforms in an industrial context delivering comparable results to those of more established proprietary platforms but at significantly lower costs. A challenge that has been widely overcome: IDT today has operational machinery in several Italian and European companies that have faith in their idea. Since its establishment, the company has experienced continuous growth, affording them the ability to fully design their machinery in-house. This includes mechanical, electrical, software, and hardware design, without depending

on third party companies. Their primary advantage is their group of motivated and youthful team members, allowing IDT to stay competitive and develop innovative solutions to meet the needs of their customers. Following increasing demand for custom hardware to build machinery that can meet customer requirements, our R&D department was established in 2020. Initially, the department focused on internal projects, but soon received project requests from external customers. In 2022, the division became a self-contained business unit known as "RedSmart", capable of creating tailor-made embedded solutions according to clients' needs. The project is developed in-house and may involve a range of steps depending on the project's specifications including:

1. PCB Designs
2. Firmware and application development
3. Creation of cloud infrastructure
4. Development of mobile applications and web apps

Examples of RedSmart's developed projects include:

- An **IoT-enabled food truck**, which utilizes a board mounted on the truck to collect signals from various sensors and sends them to a cloud platform using 4G connectivity. Data collected can be visualized through a mobile application and commands can be sent to the board.
- A **smart luggage handle** that integrates sensors such as an accelerometer and inclinometer. Capable of identifying any impacts the suitcase may have sustained, using Bluetooth low energy connectivity, the device transfers this data to a mobile application.

- A **defibrillator trainer**, a device that emulates the functionalities of a real defibrillator used during training. It receives commands through an IR (Infrared radiation) receiver.

1.2 Project overview

The project aims to develop a device capable of measuring the power consumption of industrial-grade machinery and sending the acquired data to a cloud platform. The design follows a modular approach, which includes a baseboard containing a microcontroller that already embeds a Wi-Fi module, Ethernet, power measurement, and I/O circuitry. The baseboard can be extended with expansion boards:

- LoRaWAN expansion board
- 4G expansion Board
- Relay expansion board
- CAN expansion board
- RS485 expansion board
- SD expansion board

Based on the applications in which this system will be installed, the appropriate board will be mounted to meet the demands of each individual project.

In terms of connectivity, there are four available options:

- Wi-Fi (built-in)
- Ethernet (built-in)
- LoRaWAN (exp-board)
- 4G (exp-board)

Once data reach the cloud platform, it is stored in a database and displayed via a web application or mobile app. It can also monitor various types of data. Digital inputs can be configured to count pulses or connect buttons to specific functions. Additionally, it can receive different data from expansion boards, such as the CAN board connected through an expansion IDC connector present on the baseboard. Using an external infrastructure consisting of multiple Bluetooth low energy gateways, it is possible to determine the location of machinery on which the device is mounted within various sections of the production plant. Each piece of collected information is standardized for easy interpretation by the cloud platform and user applications.

Chapter 2

Trace&Follow first version

A previous version of this project was already available, and the board can be seen in Figure 2.1. It was developed using an Arduino board mounted on a specifically designed carrier board. This version was customized for a specific machine, specifically an industrial welder and it features LoRaWAN connectivity provided by the Arduino board through a LoRaWAN modem.

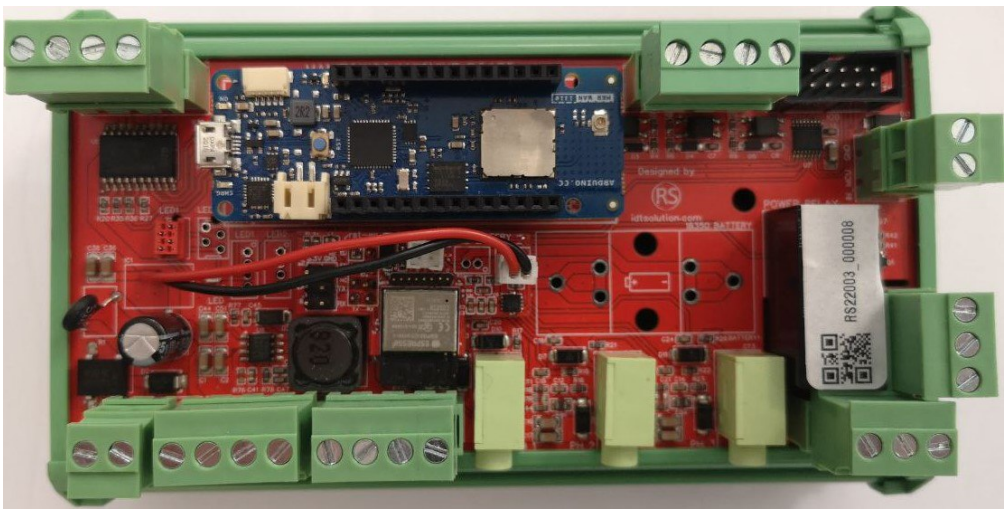

Figure 2.1: T&F board first version

Even though it was working well on that specific machine and in that specific environment, it presents some problems when considering the expansion of the product to a wider range of machines:

- LoRaWAN connectivity is relatively slow. If we would face a situation requiring a higher volume of data, this protocol could become a bottleneck. In addition, Moreover, it necessitates a more intricate and costly infrastructure in comparison to other types of connectivity.
- Power consumption evaluation method was calibrated specifically for that type of machine and and its accuracy beyond that range was limited. To calibrate the device for a new machine type, modifications to certain firmware parameters would be required. This process is time-consuming and, above all, not scalable for larger numbers.
- Some of the microcontroller I/O pins are designated for specific functions that may not be necessary for other machines, reducing available I/O for other features that may be required instead.
- Availability of the Arduino board has decreased since the project began, creating a high risk of it becoming out of stock, which is not feasible in the long term.

Due to the aforementioned reasons, a more flexible version was necessary. This work will describe the various components of the updated version and their interactions with each other.

Chapter 3

Hardware

3.1 Prototypes

Before reaching its final state, every electronic board must undergo several prototypes in which the singles ICs are tested, the functionality of the circuits is analyzed, and the communication between the different components is verified. The development of the Trace&Follow board started on a breadboard with a commercial ESP32 development kit connected to individual components to test their functionality and determine the optimal circuitry to surround them.

After completing the initial stage, a non-modular prototype has been created to test the most critical components together using a preliminary firmware. Once all the required tests had been passed, the modular PCB development process began.

3.2 Final PCB

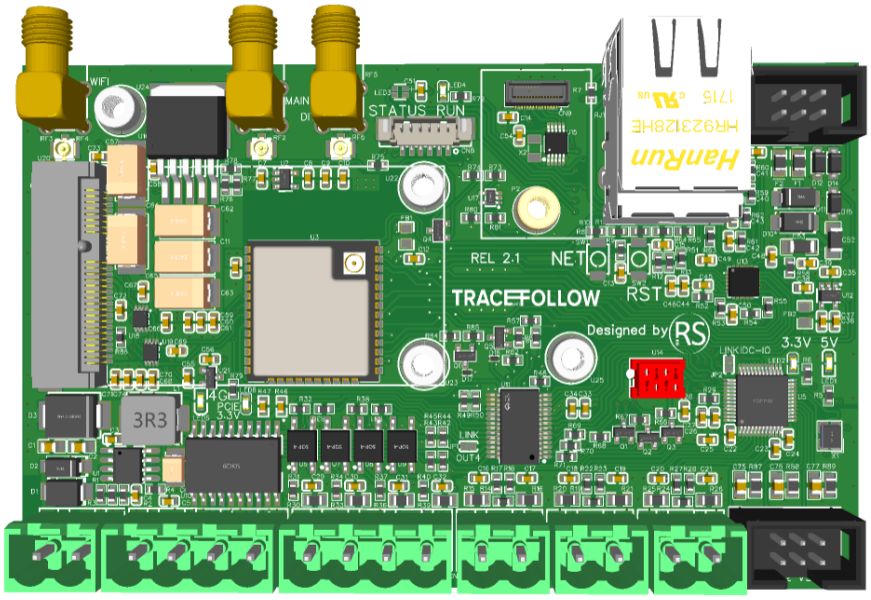

Figure 3.1: Trace&Follow board

The Final PCB, showed in Figure 3.1, incorporates Wi-Fi and Ethernet connectivity and is designed to maximize expandability. Two connectors serve this purpose. A first Mini PCIe form connector allows for expanded board connectivity options, such as LoRaWAN or 4G PCIe boards, while a second IDC connector allows to connect expansion board to support multiple protocols, such as CAN-BUS or RS485. Additionally, it has four digital inputs and outputs. The system state can be communicated to the user through an RGB LED. If the board is enclosed, an external LED can be connected via a connector. Power consumption can be measured using an Atmel energy meter IC. Furthermore, by connecting an SD expansion board, it is possible to save logs on a micro-SD card. The board can be programmed using a USB-Serial converter through a dedicated port.

In Figure 3.2 a block diagram of the main board is reported.

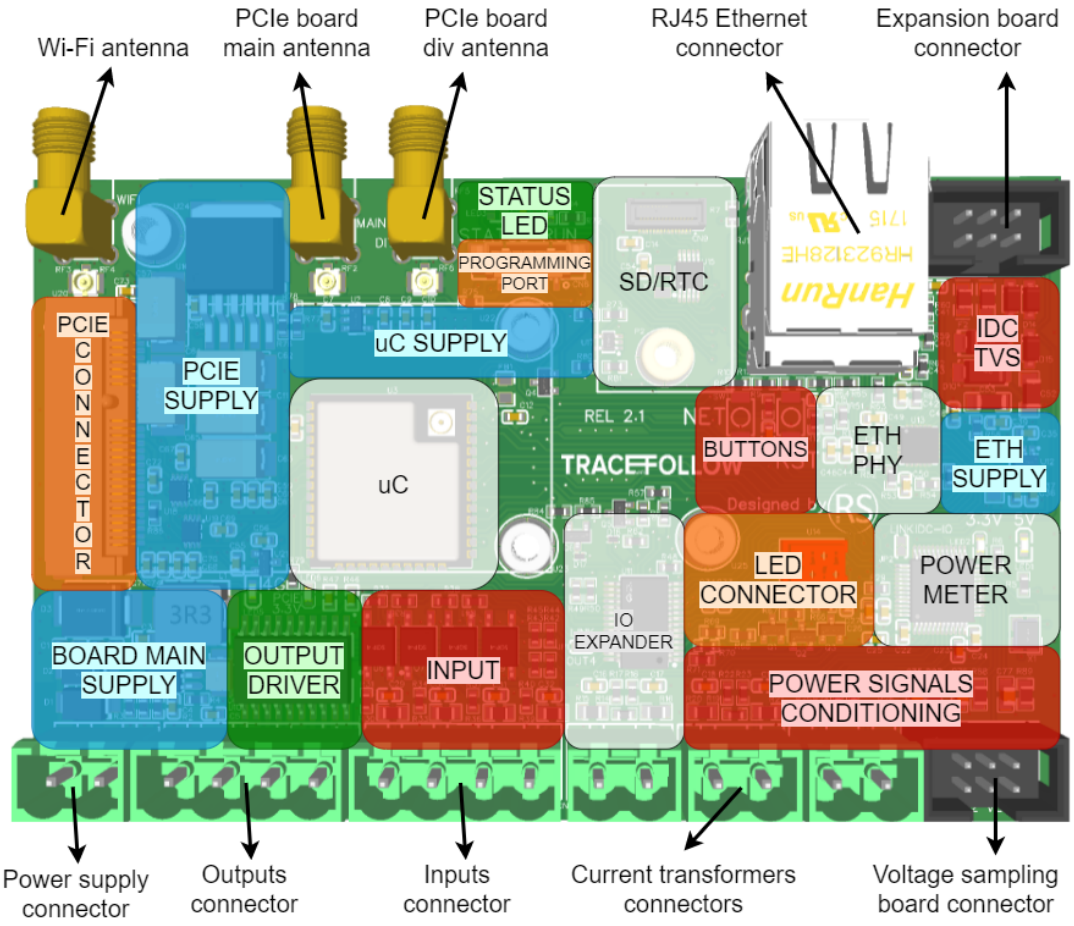

Figure 3.2: T&F board block diagram

3.2.1 Stackup and ground connections

The final printed circuit board is a four-layered PCB with components mounted solely on the top layer to minimize automatic assembly costs. The top and bottom layers contain signal traces while the first inner layer serves as a power plane to distribute supply voltage across the board. The last layer contains both digital and analog ground planes, with a specifically allocated analog ground plane for the analog section of the power meter chip, which internally establishes the connection

between the two planes, thus eliminating the need for external bridges. To ensure more efficient return current distribution and reduce disturbances, connections between the ground pins of components and the ground plane are achieved through multiple vias.

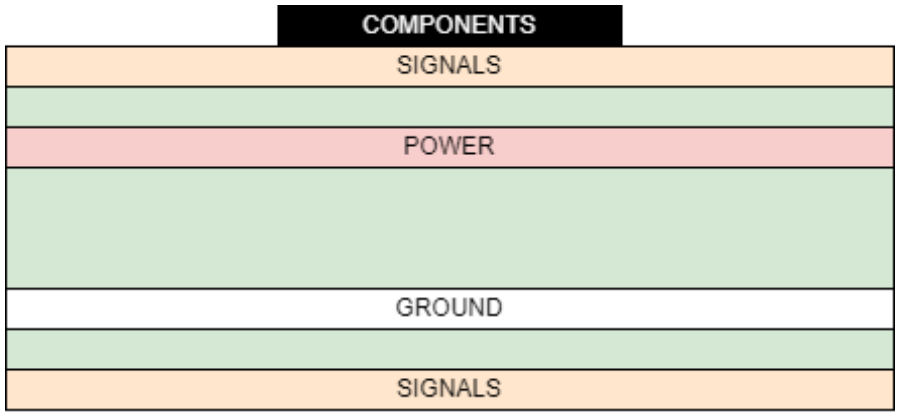

Figure 3.3: Stackup

Looking at Figure 3.4 it is possible to see how ground plane is organized.

As previously explained, the digital and analog planes are not directly connected. The analog section covers the area where the analog signals from the current and voltage sensors are received. The digital section covers most of the board with the exception of two areas. One of them is located near the power supply connector. A TVS (Transient Voltage Suppressor) is located after it to protect the board's components against overvoltages. The ground plane is intentionally not linked directly to the external ground to maximize the protection provided by this component. The second opening has been designed to accommodate the ESP32 package with an SMT (Surface-mount technology) antenna integrated.

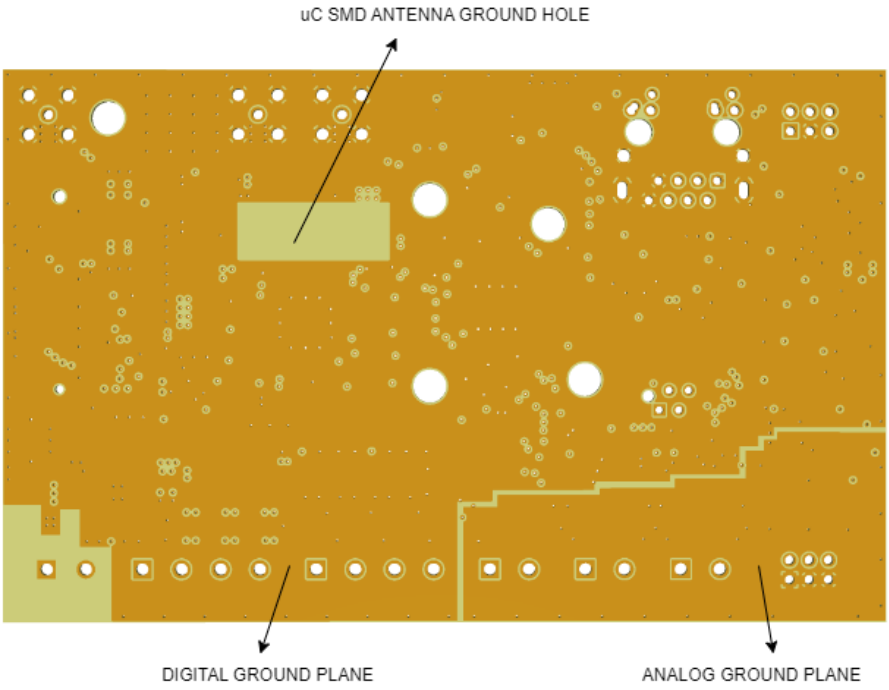

Figure 3.4: Ground plane

3.2.2 Power supply

In the figure, the power distribution tree is presented. The primary buck converter accepts DC voltage within the specified range of 9-38 V. Exceeding this range may cause damage to the converter or result in incorrect regulation of the output voltage. The converter outputs 5V, which supplies three LDOs and can provide up to 3A of continuous current and up to 3.5A of peak current. The regulated 5V supply three different linear regulators:

- **PCIe 3.3V regulator:** The PCIe 3.3V regulator supplies power to the PCIe expansion board if connected. The device must provide at least 2.5A of continuous current and 3A of peak current as these are the requirements when using the 4G module. This regulator can be turned off if no PCIe expansion

module is connected

- **ETHERNET 3.3V regulator:** A dedicated Ethernet power supply module has been added to avoid overloading the main 3.3V regulator. This module is particularly important as it allows for the physical deactivation of the Ethernet interface when it is not in use.

- **BOARD 3.3V regulator:** It supplies everything on the board except for the Ethernet module. It is capable of providing a continuous current of up to 500 mA and a peak current of 700 mA.

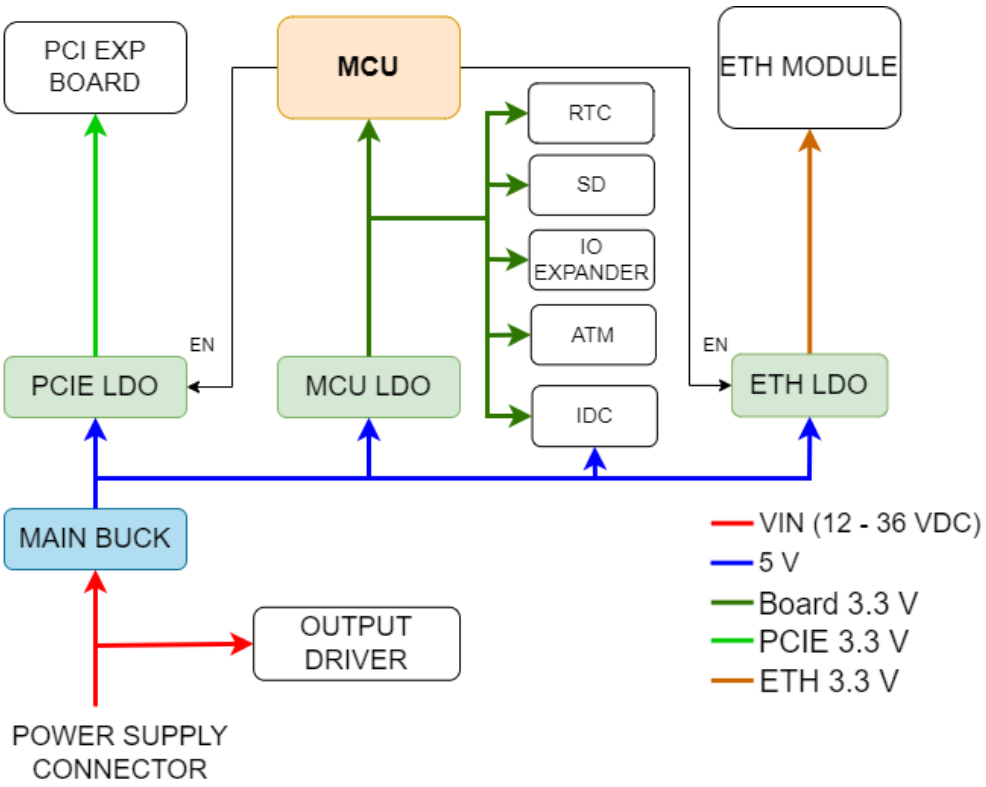

Figure 3.5: Power distribution within the board

3.2.3 Microcontroller and component interface

The board uses the ESP32 WROOM microcontroller, a dual-core microcontroller capable of speeds up to 240 MHz. The selection of this microcontroller was based on its excellent cost-to-performance ratio, open-source libraries availability and on its high availability in the market, which is crucial, especially during the current period. The ESP32 microcontroller is available as a standalone device or as part of a module with integrated antenna matching circuitry and a QUAD-SPI flash, which is itself available with two packages:

- With an on-board PCB antenna directly embedded into the module.
- With a pigtail connector for an external antenna.

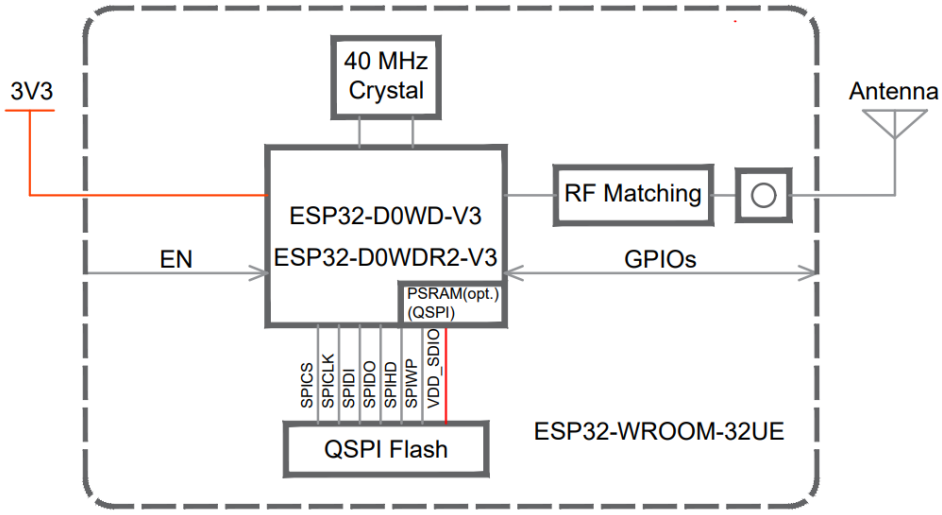

Figure 3.6: ESP32 module overview [1]

The preferred setup involves connecting the module to an external antenna. However, the PCB is also compatible with the alternative package in case of a shortage of the primary option. Among all the available versions of this microcontroller (C3,

S2, S3, H. . .), the WROOM was selected because it already includes the Ethernet MAC interface. Here a summary of ESP32 main feature taken from datasheet is reported [1].

- 448 KB ROM (Read-only memory)
- 520 KB SRAM (Static random access memory)
- 16 KB SRAM in RTC (real-time clock)
- Wi-Fi 802.11b/g/n up to 150 Mbps
- Bluetooth 4.2
- Peripherals: SD card, UART, SPI, I2C, PWM (Pulse width modulation), IR, ADC, DAC
- Operating conditions: -40 $\sim$ 85 °C

Figure 3.7: Available ESP32 WROOM package [2, 3]

The microcontroller communicates with other components on the board via the SPI bus. As shown in the diagram below, the bus lines (MISO, MOSI, CLK) are shared among all the components, but each chip has its dedicated chip select. Only one chip select can be active at any given time. The ESP32's SPI peripheral has

the potential to reach speeds of up to 10 Mbps, but due to constraints from other chips, the maximum achievable bit rate is limited to 8 Mbps.

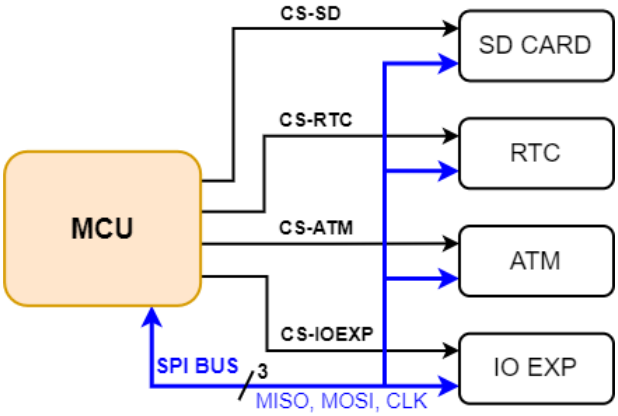

Figure 3.8: SPI distribution

3.2.4 Input/output

The board includes four inputs and four outputs. Due to the restricted number of pins on the microcontroller, the MCP23S17 IC, an 16-pin I/O expander has been added to the board. This chip replicates the microcontroller's ports, enabling the direction to be set and both pull-up and pull-down resistors to be enabled or disabled for each pin. It is split into two 8-pin ports, each with a dedicated interrupt line that is used to signal changes in the state of pins configured as inputs. The four outputs are connected to four pins on the I/O expander, with an output driver in between. It is a BTS724G IC able to provide up to 3.3A of current per channel and an overall total of 7.3A across all four channels. The high-level voltage of these pins matches the supply voltage, as the driver is powered directly by it.

The four digital inputs are also connected to the IO exp. Each input is opto-coupled to obtain a galvanic isolation between the outer world and the board.

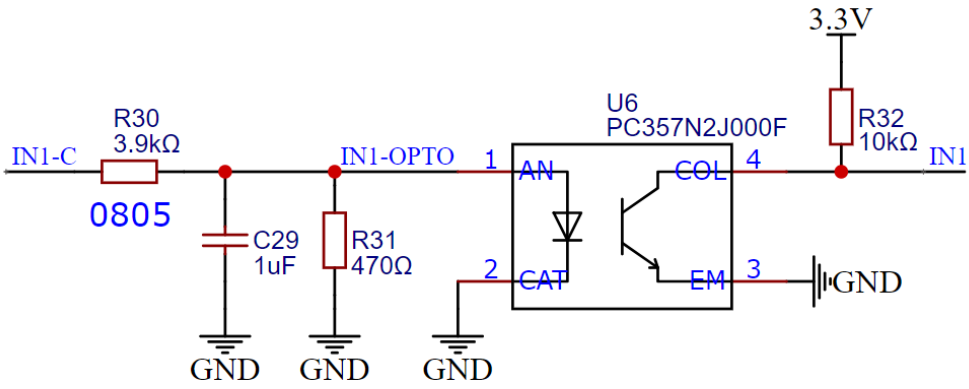

Figure 3.9: Input circuitry

This circuit is designed to scale the input voltage to 3.3 volts, which is the accepted level by the IO expander IC. At first, the input voltage is reduced by a voltage divider before reaching the opto-coupler, which, together with the pull-up resistor on its output, guarantees a pin voltage of 3.3V. The drawback of this circuit is that it introduces logic inversion. It translates a high input voltage into a low voltage read on the IO exp port. This consideration must be taken into account in firmware development.

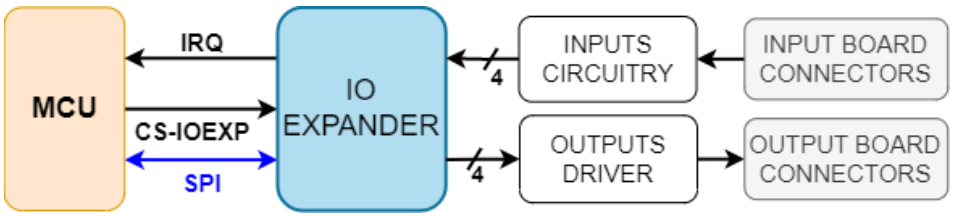

Figure 3.10: Input/output

3.2.5 Connectivity

The board is specifically designed to allow multiple connectivity based on specific application requirements:

- **Wi-Fi:** The ESP32 includes native Wi-Fi connectivity, which can be utilized in two modes, both of which are employed in this project. Specifically, the device can operate in STATION mode, enabling it to connect to pre-existing Wi-Fi networks with internet access, and establish connections to send data to the cloud platform.

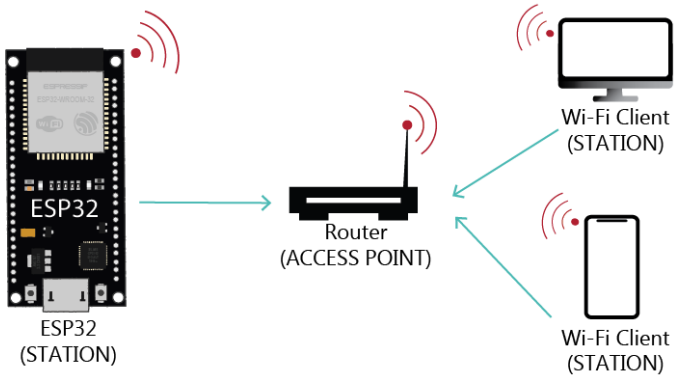

Figure 3.11: Wi-Fi STATION mode [4]

Another mode that can be utilized is the ACCESS POINT mode, where the microcontroller generates its own Wi-Fi network and allows configuration of SSID (Service set identifier), password, type of encryption and other connection parameters. This mode is helpful because when used in this mode, it is possible to host a web server on it, managed directly by the microcontroller. In this project, the web server has been used to host a web page to perform OTA updates. In a future release, the board will also feature a web page that allows for configuring certain parameters without the need for manual editing of the

configuration file.

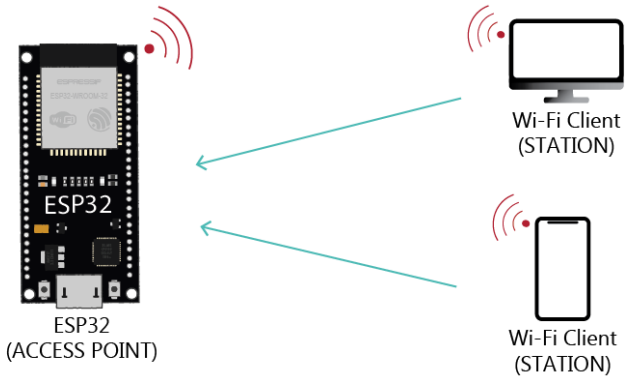

Figure 3.12: Wi-Fi AP mode [5]

- **Ethernet:** The WROOM variant of ESP32 features the Ethernet Medium Access Control (MAC) layer, which interfaces directly with the physical channel of the Ethernet connection. However, a transceiver, specifically the LAN8720 IC, is still required. Disabling the Ethernet interface is achievable by deactivating the LDO providing power to the transceiver. In accordance with Ethernet specifications, a transformer is required between the transceiver and the RJ45 connector. The Trace&Follow board mounts a connector that incorporates this transformer, which allows for space optimization on the board.

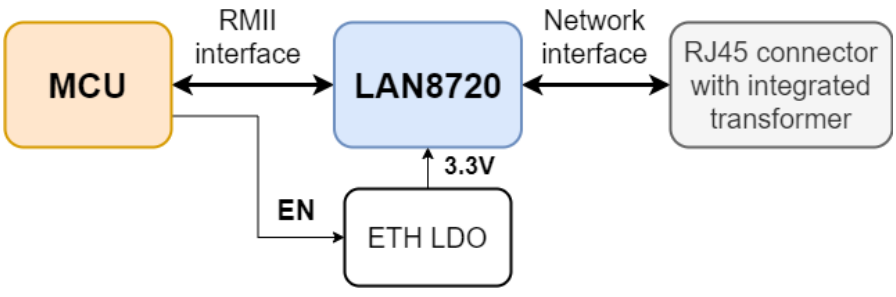

Figure 3.13: Ethernet connection

- **LoRaWAN:** The microcontroller cannot support LoRaWAN connectivity by itself. Therefore, a PCIe expansion board was designed, with a LoRaWAN modem installed, which only requires a UART and reset signal to communicate with the microcontroller. Since this protocol does not allow for direct internet connectivity, an intermediate gateway is necessary. Once the data reaches the gateway, it is distributed to the selected network server, which in this specific case is one part of a software called "Chirpstack" running on an AWS (Amazon Web Services) virtual machine. Additional information will be provided in chapter 4.3 ChirpStack for LoRa.

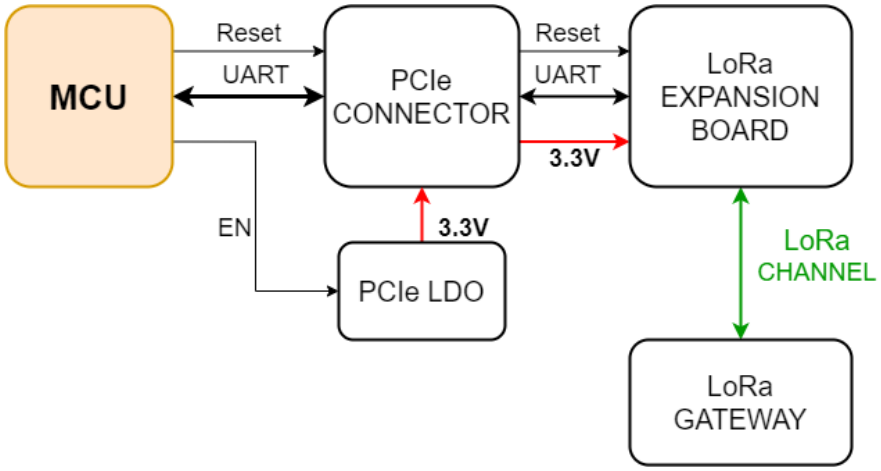

Figure 3.14: LoRaWAN board connection

- **4G:** As with LoRaWAN, 4G connectivity is not supported by the baseboard and requires a PCIe expansion board. Chapter 3.3 Expansion boards contains additional information on this board.

3.2.6 Power measurement

The Trace&Follow product is mainly used for the monitoring of the power consumption. This device can monitor power consumption of individual industrial machinery on which it is mounted. To achieve this, the product integrates the ATM90E32 from Atmel, a poly-phase energy metering IC with outstanding ±0.1% accuracy for active energy and ±0.2% accuracy for reactive energy. It is versatile and can work with both 3P4W and 3P3W systems. It features six autonomous 2nd order sigma-delta analog-to-digital converters used to sample waveforms from three current channels and three voltage channels. Additionally, it integrates a digital signal processor (DSP) to execute calculations related to various kinds of energy [6]. A block diagram of the integrated circuit is presented in Figure 3.15.

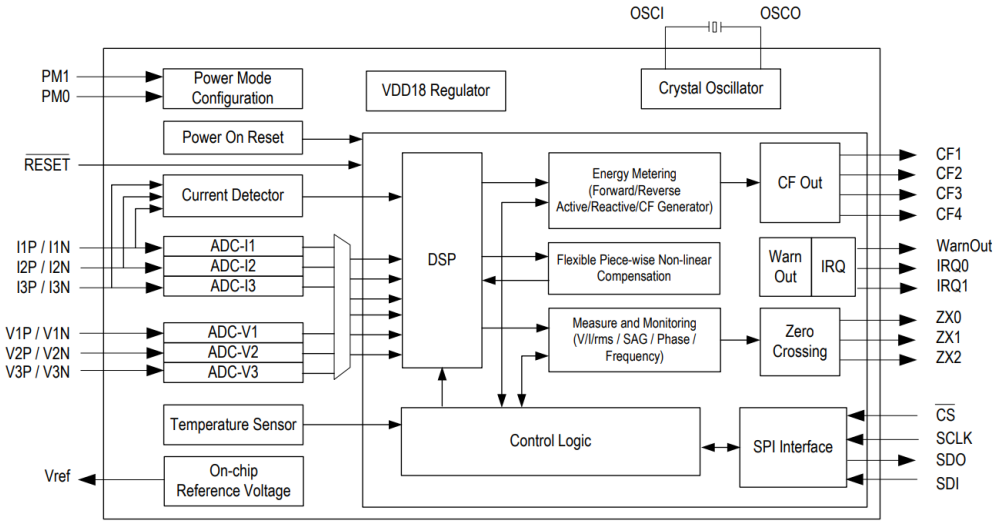

Figure 3.15: ATM90E32 energy meter [6]

Signals on current channels are obtained using three current transformers which function as current sensors. A current transformer is a device that, when placed around a cable, uses electromagnetic induction to produce a current output that

is proportional to the current flowing through the cable. By inserting a resistor between the two terminals of the secondary winding, it's possible to convert the current signal into a voltage signal. Since this device relies on electromagnetic induction, it is limited to use with AC signals. Its output is a sinusoidal signal that matches the frequency of the current flowing through the cable. However, due to power consumption variations, this signal is not perfectly sinusoidal, as its amplitude may vary. It is also possible that some high-frequency noise is collected along the cable and to mitigate its effects, a low-pass filter has been inserted before CT signal is fed into the IC. CT conditioning circuit can be seen in Figure 3.16

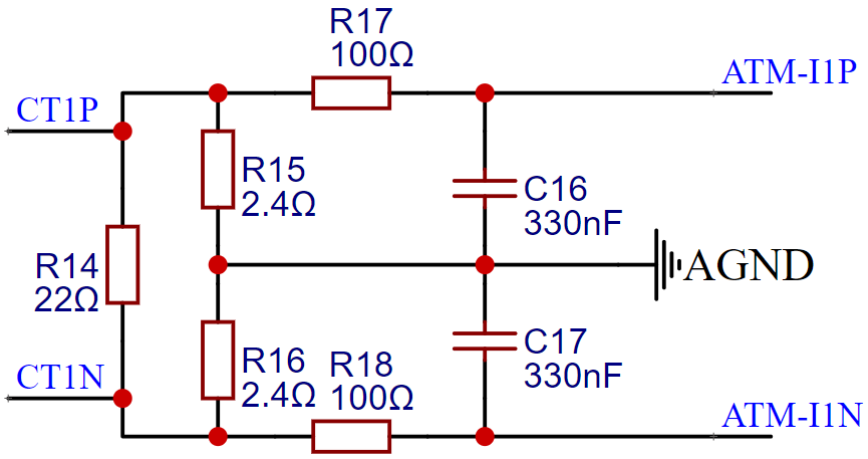

Figure 3.16: CT input filter circuit

The first three resistors convert a current signal to a voltage signal, while the other four components create the filter. The cutoff frequency is about 4.8 kHz. The simulation shows the filter in action, in Figure 3.17 the red trace represents the unfiltered signal and the green trace represents the signal received by the IC.

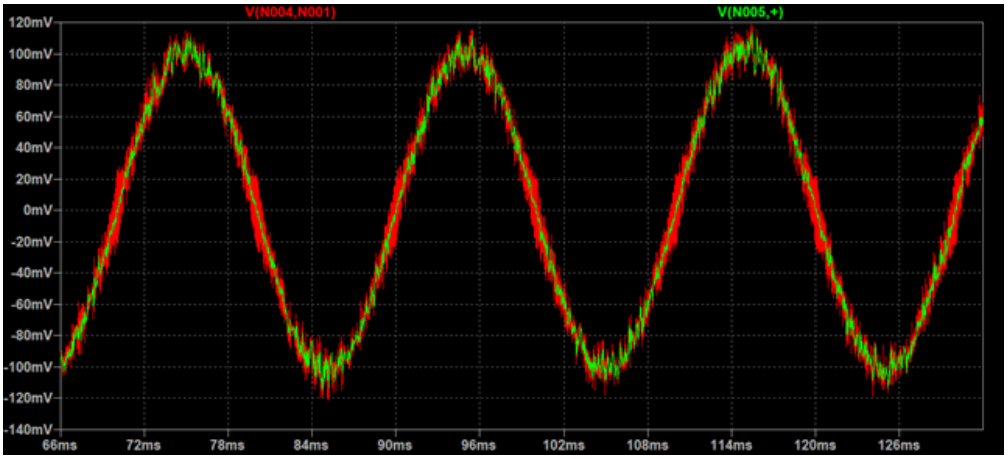

Figure 3.17: Filter simulation

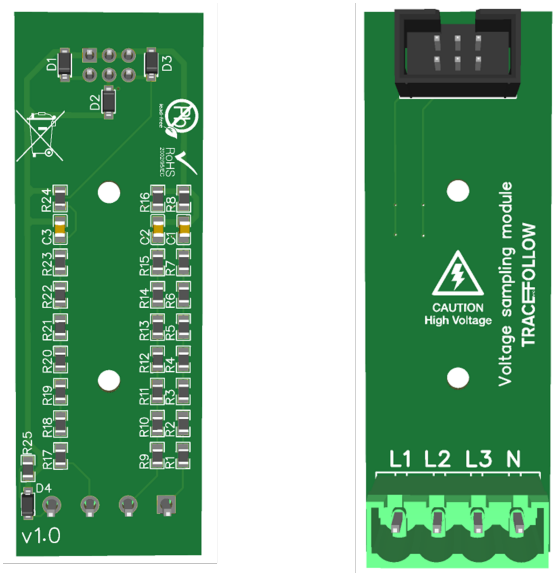

Figure 3.18: Voltage sampling board

After sampling current signals, voltage signals must also be sampled. The voltage to be sampled can be 230V AC or 400V AC, depending on the power supply, as current is measured on the grid cable. This can be achieved through the use of a voltage divider circuit. To prevent damage to the board in case of malfunction, a separate board was developed to isolate the high voltage section. The scaled

signals are then transmitted to the main board via a flat cable. A picture of a 3D model of the voltage sampling module is reported in Figure 3.18

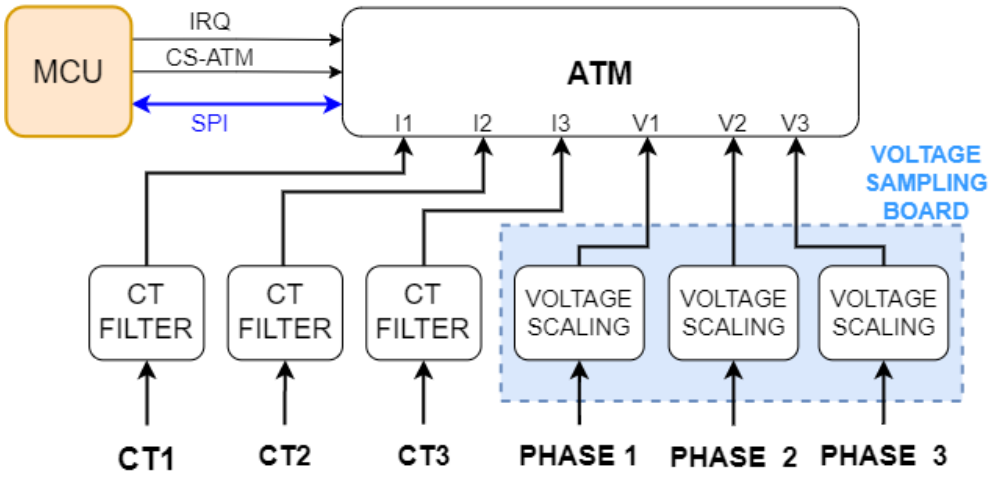

Figure 3.19: ATM connection schematic

The machinery's supply configuration determines the type of connection configuration required. In a 3-phase, 4-wire system, which includes three phases and a neutral connector, all three channels must be used to achieve accurate measurements. However, if there is no neutral connector, as in a 3-phase, 3-wire system, the Aron method can be used.

Aron method

The Aron method is a technique for measuring power consumption in a 3-phase, 3-wire system that only requires 2 channels of a wattmeter. For every three-phase device, power consumption can be written in this way [7]:

$$P = \vec{E_1} \cdot \vec{I_1} + \vec{E_2} \cdot \vec{I_2} + \vec{E_3} \cdot \vec{I_3}$$

In this kind of systems, since they do not have the neutral conductor, is it true that:

$$\vec{I_1} + \vec{I_2} + \vec{I_3} = 0$$

We can write one of the current as:

$$\vec{I_2} = -\vec{I_1} - \vec{I_3}$$

If we came back to substitute in the initial formula, we end up to:

$$P = \vec{E_1} \cdot \vec{I_1} + \vec{E_2} \cdot (-\vec{I_1} - \vec{I_3}) + \vec{E_3} \cdot \vec{I_3} = (\vec{E_1} - \vec{E_2}) \cdot \vec{I_1} + (\vec{E_3} - \vec{E_2}) \cdot \vec{I_3}$$

$$P = \vec{V_{12}} \cdot \vec{I_1} + \vec{V_{32}} \cdot \vec{I_3}$$

$$P = W_a + W_b$$

That shows that the total power consumption can be evaluated only using two channels of the wattmeter, saving one current sensor for each machine.

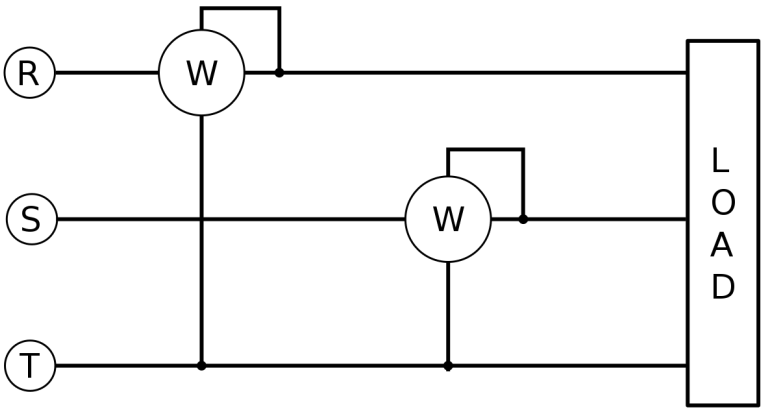

Figure 3.20: Measurement in 3P3W systems using Aron method [7]

3.3 Expansion boards

Since this system was designed following a modular approach, multiple expansion boards were developed to extend the basic functionality of the main board. These boards can be connected to the main board using either the PCIe connector or the IDC expansion connector with a flat cable. The PCIe board complies with the PCIe mini card standard with dimensions of 30 x 50.95 mm. Within this project there are two PCIe board:

- **LoRaWAN board:** This board allows the use of LoRa connectivity. It was created as a component of the project and features a LoRa modem connected to the microcontroller via a UART interface. The board contains both pigtail and SMA (SubMiniature version A) antenna connectors.

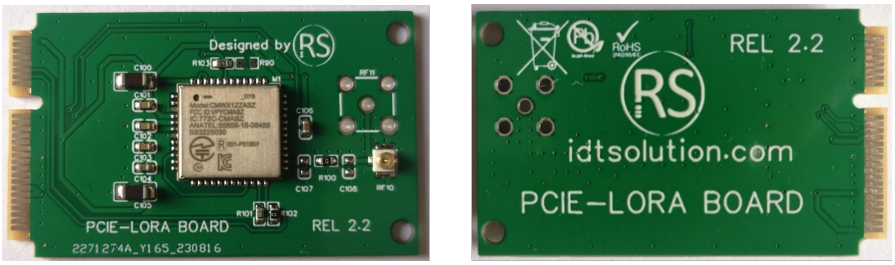

Figure 3.21: LoRa expansion board

- **4G board:** This device enables the use of 4G connectivity. The board has not been designed specifically for this project. It arrives with modems pre-installed, available with various models. Among these, two modems have been identified for integration into Trace&Follow.

When utilizing 4G modems, an issue arises as a result of variations in the logic levels of signals between different modems. Although both modems communicate

via UART interface, some use 3.3V while others use 1.8V. As a solution, a system with two level shifters, each including an enable signal, has been created. As a result, driving the enable signals correctly allows for only one logic level to be active at any given time. Additionally, the two enable signals drive the enable signal of the LDO, which supplies power to the entire PCIe board. This method saves a microcontroller pin. It also works with the Lora board, which operates at the 3.3V level and only requires the activation of the corresponding level shifter.

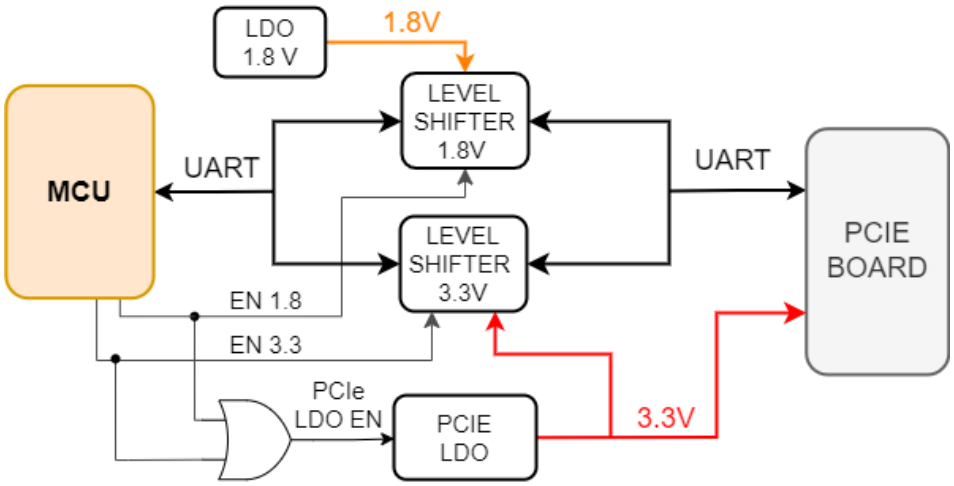

Figure 3.22: PCIe signals interface

The main board is equipped with two IDC connectors. As explained earlier, one port is dedicated to connecting the voltage sample module while the other is reserved for expansion boards. The boards do not comply with a specific format, but they can be commercially available modules or ones designed specifically for needed purpose.

Already available ones are:

- **Relay board:** Some machinery might need a relay to control some signals or to cut off some power. As discussed in section 5.2 Cutoff, the board was first employed in the implementation of the cutoff system. It utilizes just one digital output from the primary board. The second IDC connector allows easy connection of extra expansion boards.

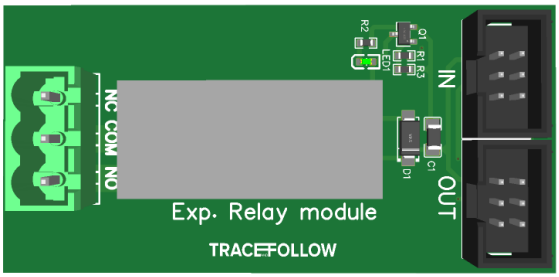

Figure 3.23: Relay expansion board

- **CAN board:** This board is available on the market and includes an Atmel microcontroller and a CAN transceiver. It can be programmed to receive CAN frames from a CAN-BUS line, analyze them, and transmit the obtained data over UART. The example provided in section 5.3 Wire monitoring illustrates how it was utilized to retrieve specific data from a welding machine.

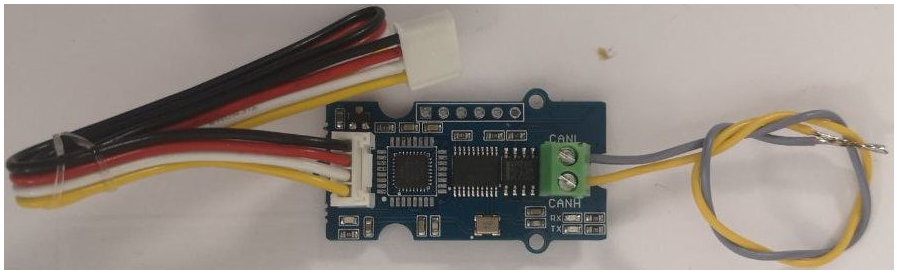

Figure 3.24: CAN expansion board

- **RS485 board:** At the time of writing, this board is still in the development phase. Its functionality includes virtualizing I/O through an RS485 bus, with the implementation of a MODBUS-RTU protocol.

Other expansion boards:

- **SD board:** In case some log files need to be saved for monitoring the proper functioning of the entire system and easily identifying problems if they occur, this board can be installed. It has its own connector on the main board.

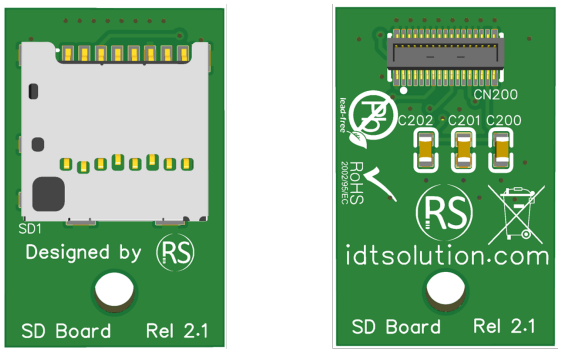

Figure 3.25: SD expansion board

In Figure 3.26, a summary of the available boards with their respective communication protocols is provided.

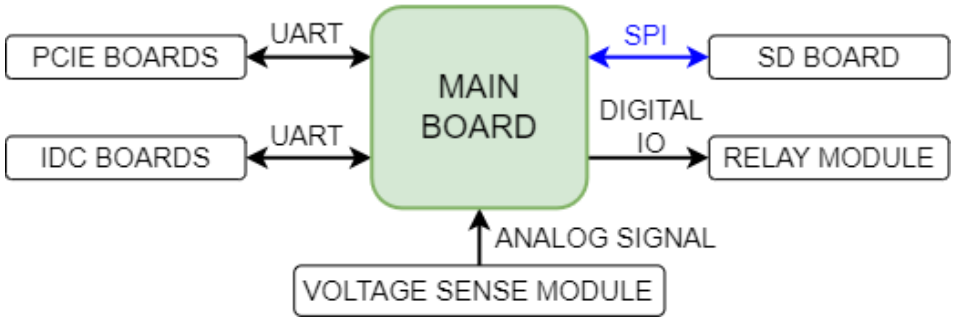

Figure 3.26: Interface with expansion boards

Chapter 4

Software

In this chapter the software part developed for the Trace&Follow project will be examined. This includes analyzing the firmware for the main board, providing an overview of the cloud platform with its rule chain for correctly interpreting data from the device, and examining the program necessary for receiving LoRaWAN frames and forwarding them to the cloud platform.

4.1 Firmware

The firmware was developed using open-source software components. Espressif system provides all the libraries for their microcontrollers and development tools as open source. Additionally, Espressif microcontrollers can be integrated into the Arduino ecosystem, enabling the use of all developed libraries present within it. Using this programming approach has clear advantages, including decreased development time, but also potential drawbacks. In the Arduino ecosystem, anyone is able to add libraries, which can be a problem because when using some libraries, there is no tearing they have been fully tested. Hence, it is a good practice to verify and test everything to ensure that everything works properly. Firmware is entirely based on FreeRTOS, which is already embedded within Espressif framework and provide specific functions to interact more effectively with the ESP32 microcontroller hardware.

4.1.1 Overview

The board's primary function is to collect data from the machine to which it is attached and send it to a cloud platform. The collected data are categorized as either periodic or asynchronous. The first category includes all data that needs to be monitored over time, which are:

- Power consumption value
- Pulses on digital inputs if configured in this way
- Data coming from expansions board (CAN or RS485)
- Position if this feature is enabled

The second category consists of data that can change at any moment, and it's more important to know their current state than to track their changes over time. It includes:

- Current inputs and outputs digital value
- Function associated to inputs if configured in this manner
- Configuration parameters that can be modified at run-time

To correctly track periodic data over time, it is very important that no data is lost due to lack of connectivity or transmission errors, while in the case of asynchronous data, an error during transmission can be handled on the next attempt without loss of data. To address this potential problem, periodic data is inserted into a structure with associated a timestamp at configurable intervals. This structure is called a "packet" and, when one of them is ready is saved into a buffer. Once the board is ready to send data to the cloud, it sends all existing packets into the buffer (up to the maximum payload size that can be sent). If the sending operation is successful, the packets are removed from the buffer. Otherwise, they will remain untouched and will be attempted for transmission in the next opportunity.

To acquire and transmit these data, the application relies on the freeRTOS operating system. Within a freeRTOS application, tasks are executed based on their priority under the control of the scheduler. They interact with each other using special features provided by the operating system such as queues, mutexes, semaphores and more. The Arduino framework automatically creates a task that execute the main function that starts at boot time. Therefore, any code added to the main function is executed within this task. The main task is responsible for two functions. The first one involves retrieving the configuration file from flash

memory. This file contains all the necessary parameters for the correct operation of the board in JSON format. It is possible to load this file onto the ESP32 without having to reprogram the entire firmware. Beside this it handles OTA updates. The board can be updated using Wi-Fi connectivity instead of a cable, as explained in section 3.2.5 Connectivity. This feature is based on an open-source library found on GitHub[8], and it allows for the uploading of either a new firmware version or a new configuration file only.

After completing the board configuration steps, the main function generates additional tasks to handle system functionalities, as summarized in Figure 4.1.

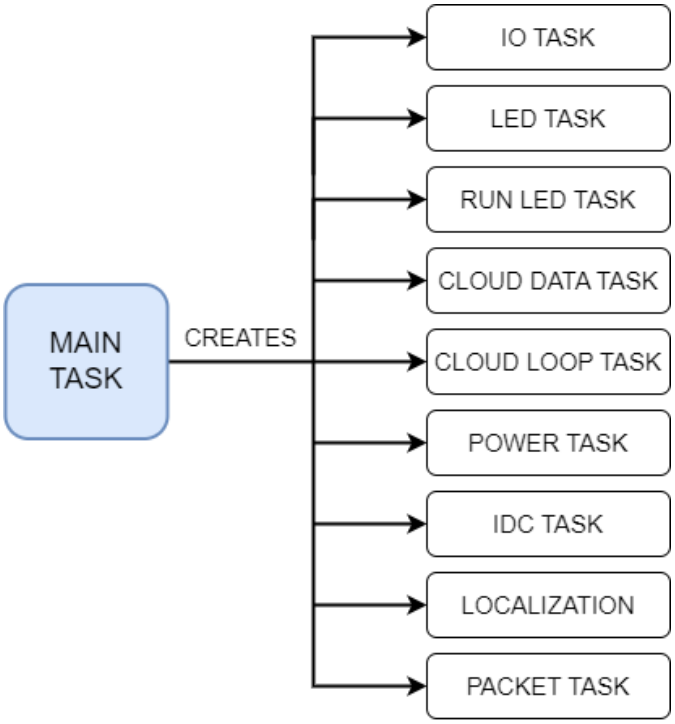

Figure 4.1: Task creation

- **IO TASK:** Manage I/O, that can be configured into different modes.

- **LED and RUN LED TASKS:** Manage status LEDs according to current state of the board.

- **POWER TASK:** Measure power consumption interacting with ATM IC.

- **IDC TASK:** Manage data coming from IDC connected expansion board if present.

- **LOCALIZATION TASK:** Manage localization using Bluetooth if enabled.

- **PACKET TASK:** Retrieve data from different task and prepare them to be inserted into packet.

- **CLOUD DATA TASK:** Create packet that will be sent to cloud.

- **CLOUD LOOP TASK:** Send data to cloud.

After all these tasks have been created, they start executing their functions, interact one to each other to realize the complete program running on the microcontroller. They mainly interact using queues though which they also exchange data. An overview of the whole program is provided in Figure 4.2

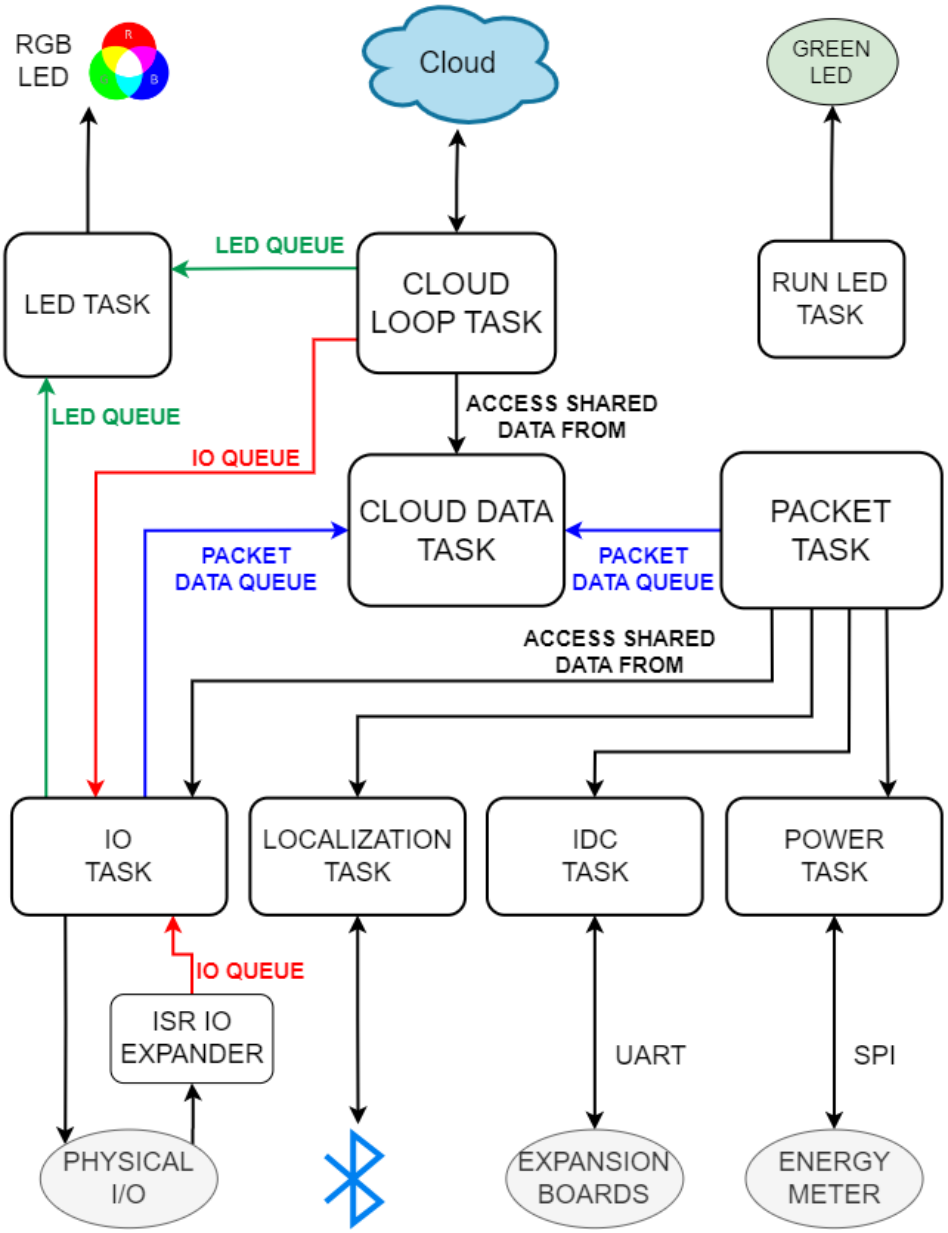

Figure 4.2: Task interactions

4.1.2 IO Task

This task manages the physical input/output. Initially, the IO expander IC is configured to set each pin's direction and enable interrupts. Afterwards, the I/O configuration is retrieved from the flash memory configuration file. Each input is associated with a configuration structure, which is outlined below.

```
typedef struct
{
    uint8_t num;          // input number
    bool    enabled;      // if =0 operation are not allowed
    uint8_t mod;          // 1->NORMAL, 2->COUNTER, 3->FUNCTION
    bool    polarity;     // if =1 logic is inverted
    pin_t   pin;          // physical MCU pin
    bool    isVirtual;    // if =1 not correspond to a physical pin

} input_config_t;
```

- *num:* Number associated with that input.
- *enabled:* Each pin can be enabled on disabled. If operations are performed on a disabled input an error will be reported.
- *mod:* Each input can be configured into three different modes:
  - **Normal:** A generic input pin where only its current state is important.
  - **Counter:** In this mode the firmware counts the number of pulses received.
  - **Function:** Using a CAN or RS485 expansion board makes it possible to process data from them as virtual inputs. An example of this operation

is demonstrated in cap(). By utilizing this field in the configuration structure, it's possible to determine whether the input is physical or virtual.

- *polarity:* Configures whether logic level 1 corresponds to a high or low physical level.
- *pin:* Physical pin associated with that input.
- *isVirtual:* Using CAN or RS485 expansion board is possible to treat data coming from them as virtualized inputs. An example of this operation is shown in chapter 5.3 Wire monitoring. By using this field in the configuration structure, is possible to determine whether it is a physical or virtual input.

Similarly, each output has its own configuration structure.

```
typedef struct
{
    uint8_t num;     // output number
    bool polarity;   // if =1 logic is inverted
    pin_t pin;       // physical MCU pin
    bool isVirtual; // if =1 not correspond to a physical pin

} output_config_t;
```

The meanings of the fields are the same as input ones. I/O data are kept in memory using two classes: Input and Output classes.

```
class Input
{
public:
    // public methods
private:
    input_config_t _config; //config structure
    uint64_t       _count;  //needed if mod=COUNTER
    bool           _functionActive;//needed if mod=FUNCTION
    TickType_t     _pressTime; //time at which a press is detected
    bool           _state; //logical state
    uint32_t       _longPressTime; //long press timeout
    TimerHandle_t  _longPressTimer_handle; //timer handle
    Mcp23S17 *     _ioExp; //pointer to io exp object
    mutex_t        _dataMutex; // mutex to protect access to data
};

class Output
{
public:
    // public methods
private:
    output_config_t _config;            //config structure
    bool            _state;             //pin state
    TimerHandle_t   _pulseTimer_handle; //timer handle
    Mcp23S17 *      _ioExp;             //pointer to io exp object
};
```

Here, only data structure is presented, while methods are omitted. Each class holds a pointer to the IO expander object to enable communication and interacts with respective configuration structures. To ensure the protection of all data during access, a mutex is present because some of them can be accessed by multiple tasks. IO expander is also managed through a class.

```
class Mcp23S17
{
public:
    // public methods
private:
    uint8_t      _error;          //last detected error
    uint8_t      _select;         //chip select
    uint32_t     _SPIspeed;       //spi bus speed
    spiObj_t *   _spi;            // spi pointer object
    SPISettings  _spi_settings;   // spi config struct

    mutex_t      _mutex = {
                     .handle = NULL,
                     .name = "ioexp mutex"
                 };               // mutex

};
```

Since SPI is necessary for the use of this IC, the class holds a pointer to a SPI object to interact with physical hardware. As this object can be accessed by multiple tasks, it is protected by a mutex to prevent simultaneous access.

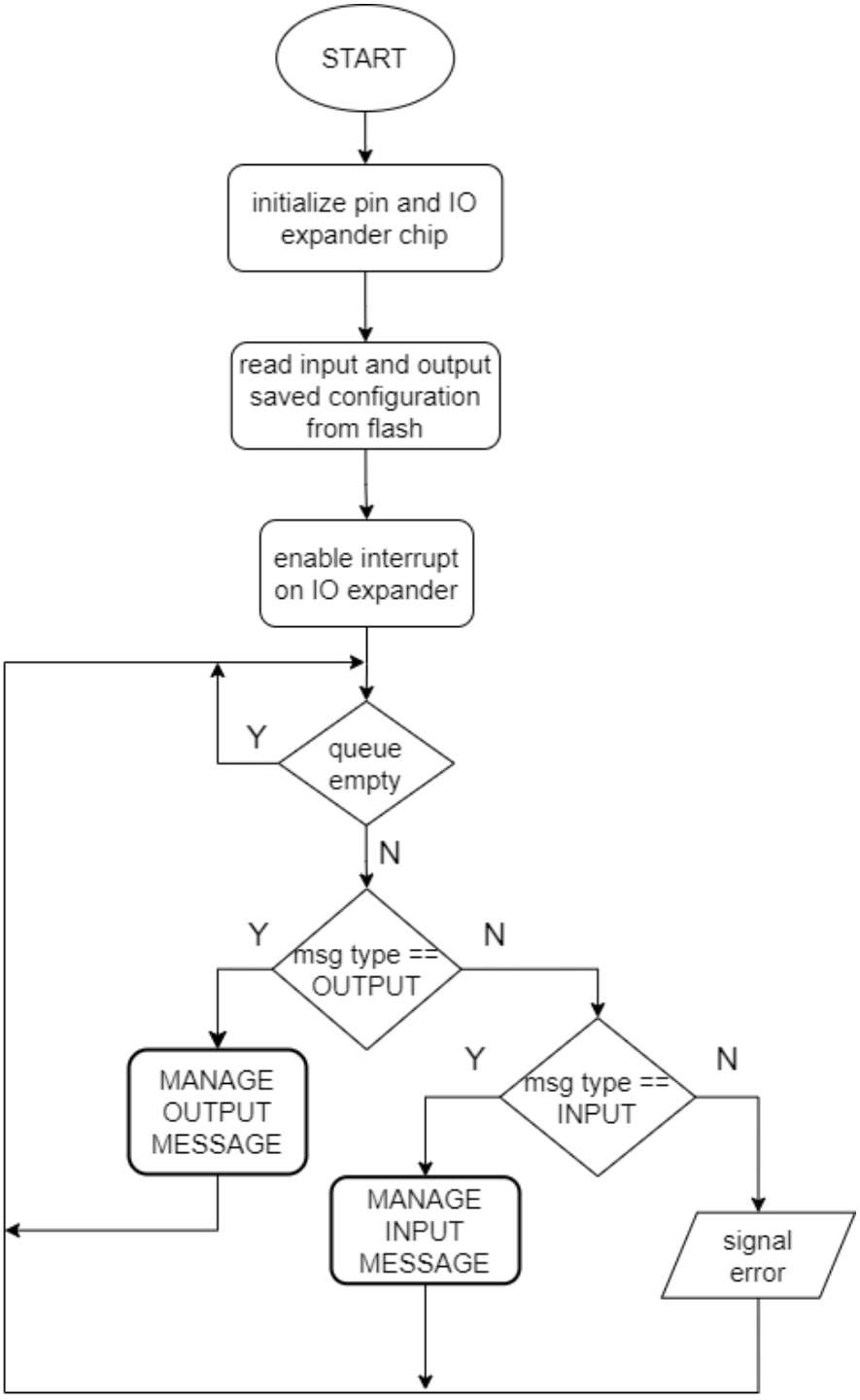

Figure 4.3: IO task function flow chart

After the configuration steps are completed, the task will wait until the IO queue is no longer empty. The structure of an IO message is the following:

```
typedef struct
{
    //0-> input, 1-> output
    bool        type;
    //if type=input unused, if type=output->tell the output number
    uint8_t     num;
    //if type=input unused, if type=output->tell the output new value
    bool        val;
    //if type=input unused, if type=output->tell the output pulse
    time
    uint32_t    time;

} IoMsg;
```

Messages in the IO queue can originate from two sources: the IO expander ISR, which is triggered by a change in input pin state, and the Cloud loop task, which arises from receiving an output command from cloud to board.

When dealing with an output message, two scenarios are possible: either the message time is set to 0, which means that the output must be set for an unlimited duration or time is non-zero, indicating that the output must remain at the required level for a designated time before returning to its previous state. To regulate the output state modification, a FreeRTOS software timer (pulse timer) is used.

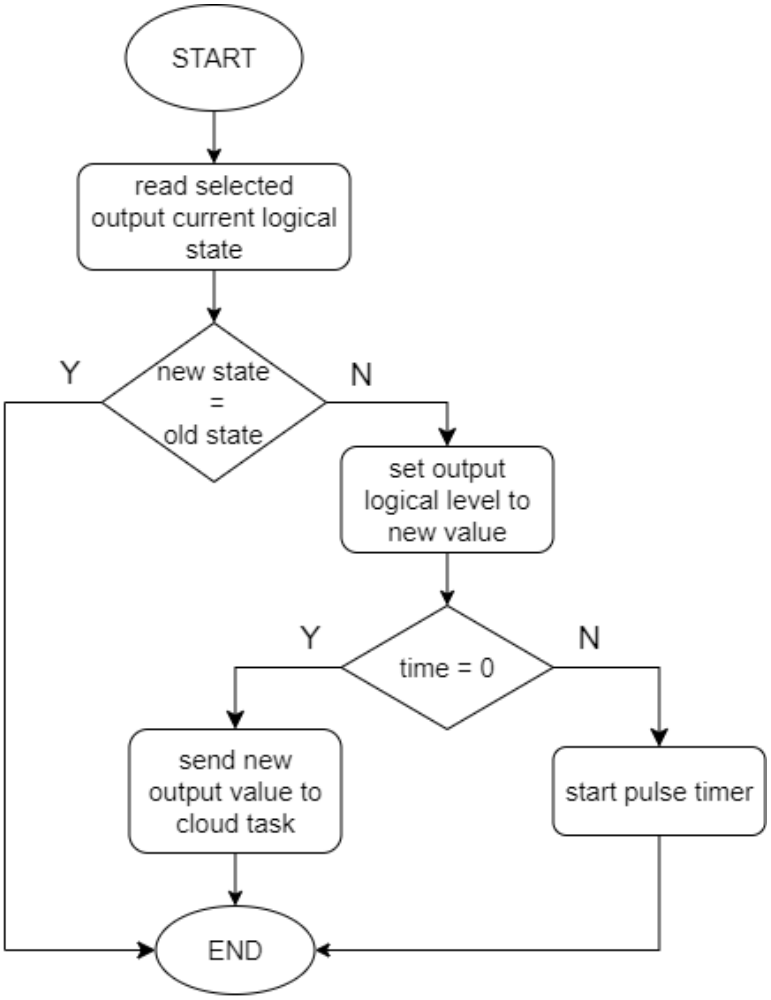

Figure 4.4: Manage output message flow chart

In the case of an input message, the logic becomes more complex. Firstly, it is necessary to determine which pin has changed its state, which can be done by inspecting the interrupt flag register of the IO expander. Then, by reading the port register, the new state of the pin can be obtained. Once this information has been retrieved, a series of actions must be performed depending on the configured mode.

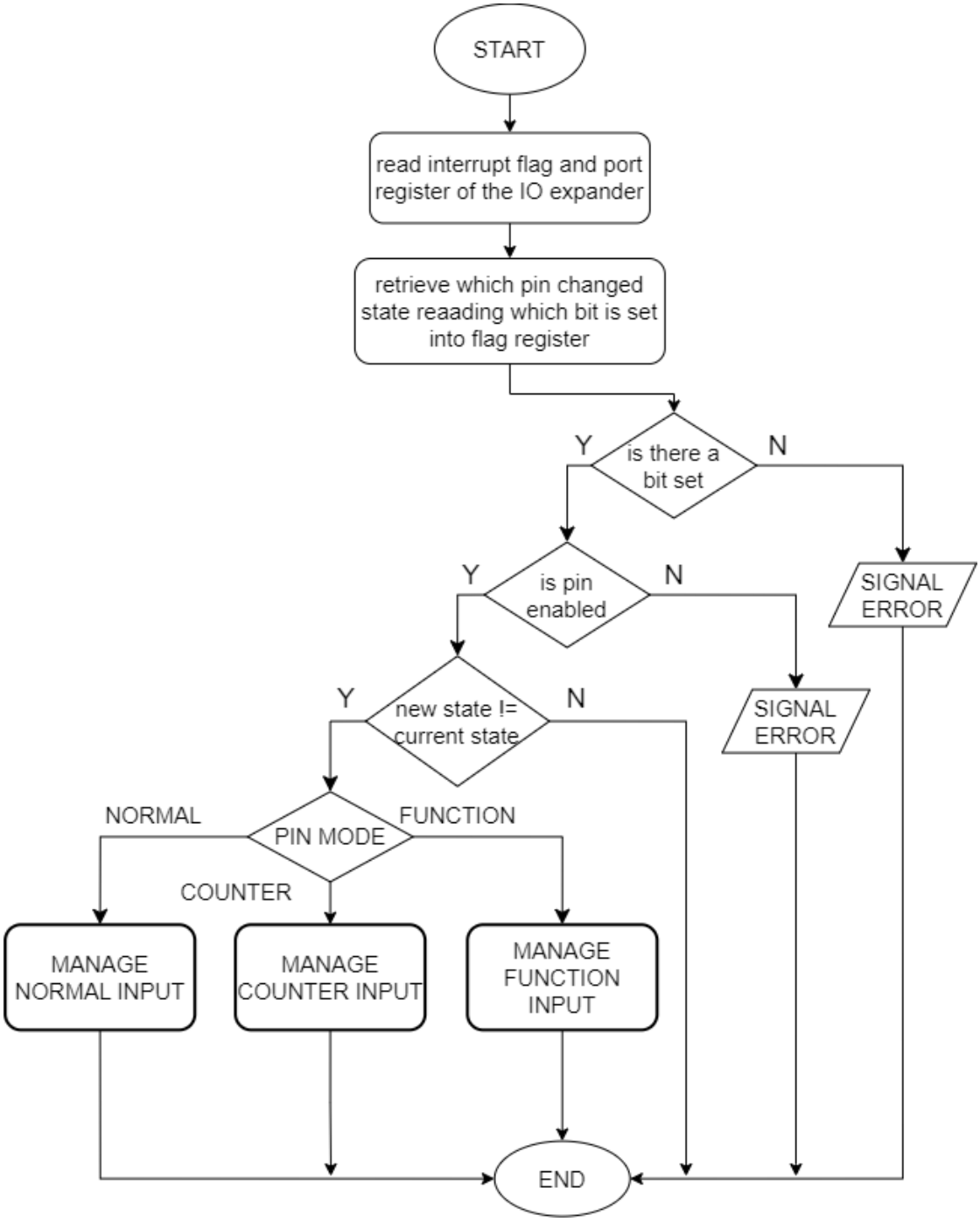

Figure 4.5: Manage input message flow chart

If normal mode is selected, the only required operation is to send the updated value to the cloud task for transmission to the cloud platform.

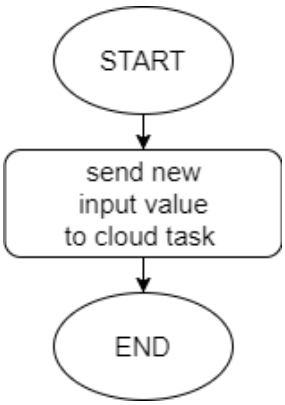

Figure 4.6: Manage normal input flow chart

When the counter mode is set, the counter simply increments on the rising or falling edge, depending on the polarity selected.

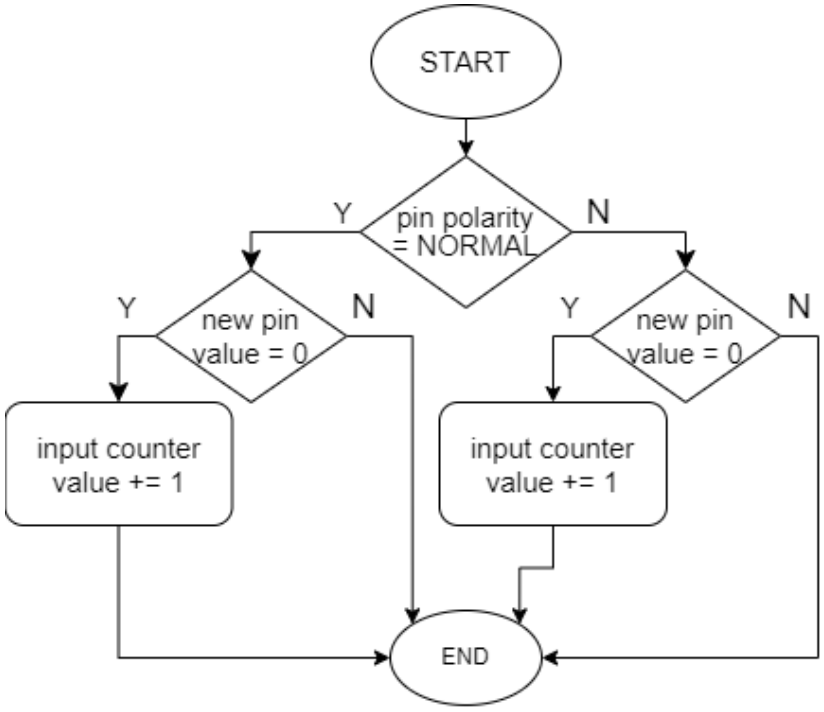

Figure 4.7: Manage counter input flow chart

Finally, when the input mode is set to function, the first thing to do is to check what the current state of the function is. If it is not currently active and a button press (logical zero) occurs, it indicates that the function needs to be activated. However, if the function was already active, a button press may signal the start of a long press, which is the action required to deactivate it. In this situation, a software timer is started with a three-second timeout. If a button release is detected, the timer is stopped if it was running. If no button release occurs within three seconds, the timer's callback is triggered leading to function's deactivation.

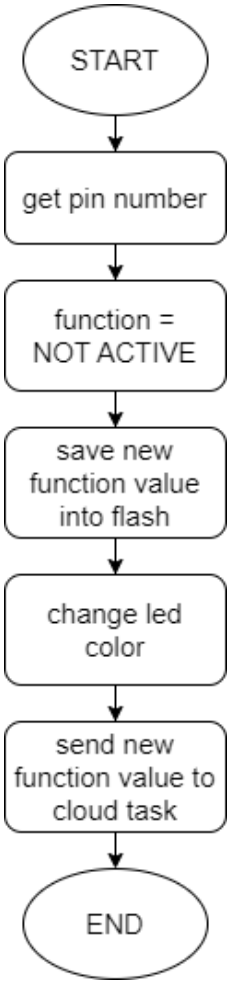

Figure 4.8: Timer callback flow chart

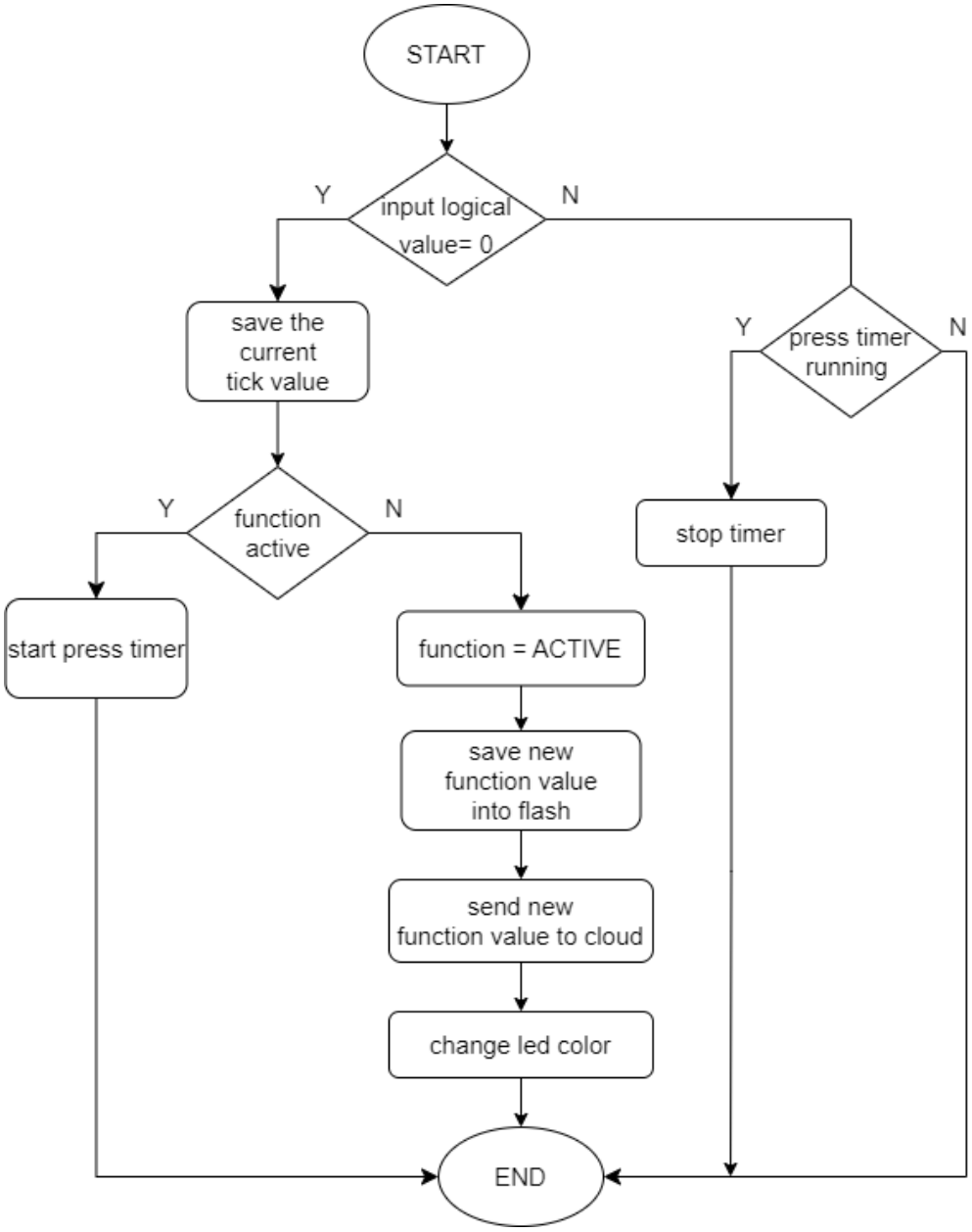

Figure 4.9: Manage function input flow chart

4.1.3 Leds Tasks

There are two LEDs on the board, each assigned to a distinct task. The first LED is a simple green light that blinks at 1 Hz. The task sleeps for 500ms before toggling the LED state. The second one is a RGB LED, which is useful to signal the system state to the user. The task receives the color required to set the LED via a queue.

```
typedef struct
{
    uint8_t  color; //requested color
    uint32_t time;  //pulse duration, if =0, led set without timeout

} led_queue_msg_t;
```

This is the message format that it receives in the queue:

- **color:** The required color.
- **time:** It is possible to change the LED state for un unlimited period by setting this time to 0, or set a timeout after which the LED returns to its previous color. This timeout is implemented using software timer with an elapsed time equal to the specified timeout. Once it expires, its callback is triggered, and it reset LED color. While a pulse is executing, no message from the queue are accepted.

The task's function waits that some messages are received in the queue. Once messages are present, it changes the LED color accordingly and, if necessary, starts the timer. The function then waits again for a new message after the pulse has concluded.

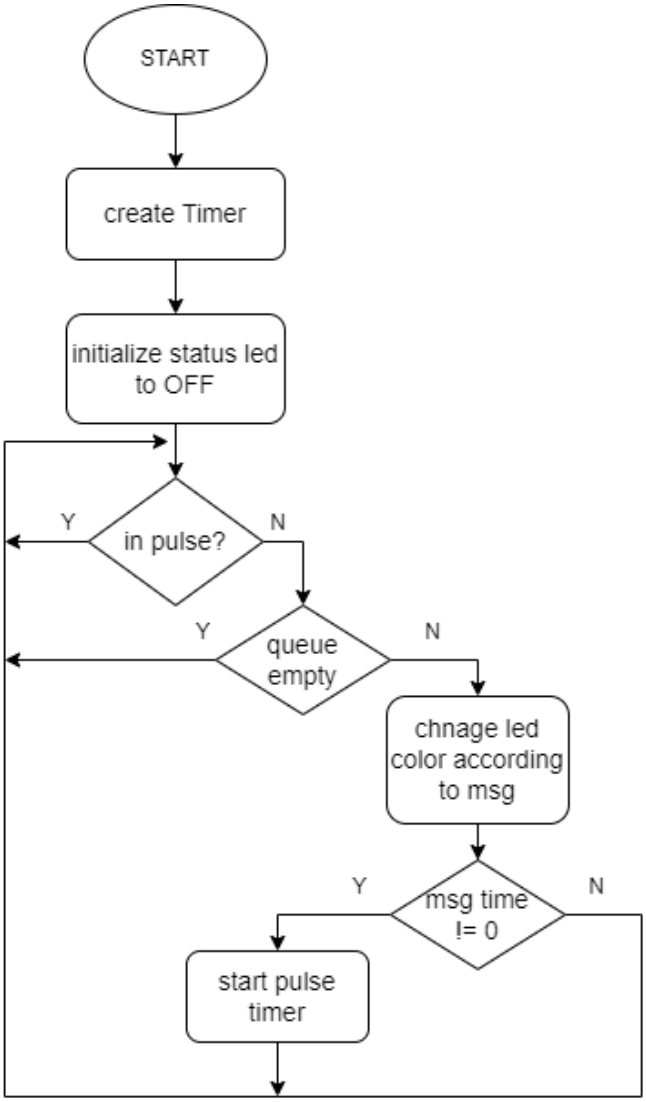

Figure 4.10: LED task flow chart

4.1.4 Power task

This task is responsible for acquiring power measurements. The majority of calculations occur within the energy meter IC, thus it is only necessary to periodically read a few registers to obtain the total and instantaneous consumption. Similarly to inputs and outputs, the necessary variables for the management of this chip are stored in memory by utilizing a class. The class includes configuration structures that hold calibration values for the ATM90E32 IC, a mutex to protect data access, variables for the cutoff function, and the object used for direct interaction with the IC registers.

```
1  class Power
2  {
3  public:
4      // public methods
5  private:
6      power_measurement_config_t   _config; //config struct
7      power_data_t                 _powerData; //power values
8      mutex_t                      _powerDataMutex; //data mutex
9      cut_off_config_t             _cutOffConfig; //cutoff config struct
10     ATM90E32                     _atm; //energy meter object
11     TimerHandle_t                _cutOffTimer_handle; //timer handle
       for cutoff feature
12     bool                         _idle; // tells if power consumption
       is above or below the threashold
13
14
15 };
```

Data structure needed for management of this IC is similar to the one used for IO exp.

```
class ATM90E32
{
public:
    // public methods
private:
    int          _cs;            //SPI chip select
    spiObj_t *   _spi;           //pointer to SPI object
    SPISettings  _spiSettings;   //config struct for SPI bus
    uint32_t     _spiSpeed;      //SPI bus speed
};
```

Even in this case, an SPI object is needed to interact with the hardware. However, a mutex was not implemented as this object is only accessed by this task. It is also important to note that the ESP32 SPI library already contains a mutex that prevents multiple SPI transactions from occurring simultaneously.

Calibration is the most crucial aspect to ensure correct operation of the ATM IC. It requires a reference power meter to correctly set IC calibration registers and obtain a precise measurement of power consumption. During this operation, both gain and offset registers must be configured for each voltage and current channel. This calibration will remain valid until a different type of CT is utilized. If this situation occurs, the procedure must be repeated.

The ATM IC has two important types of registers used in this project:

- **Total energy register:** It accumulates energy consumption across the three phases in Watt-hours (Wh). This register is cleared upon reading and is used

to evaluate total consumption, from which the value sent to the cloud platform is derived. It is not read periodically and then accumulated, but it is only read when its value is needed by another task to build a packet.

- **Instantaneous power register:** provides the instantaneous power consumption, which is useful for the cutoff function to determine whether the machine is in idle state (below the power threshold) or in running state (above the threshold).

Cutoff is a feature that can be enabled in the configuration file. It allows the system to turn off the machinery it is connected to using the relay expansion board and some additional hardware that depends on the type of machine it is being installed on (an example is provided in section 5.2Cutoff). It is possible to configure it in two ways:

- **Automatic cutoff:** The board automatically switches off the machine if the instantaneous power consumption falls below a configurable threshold for a configurable amount of time.

- **Manual cutoff:** Users can send a command to the board from the user application to switch off the machine.

This task manages the automatic cutoff when it is enabled. It checks the current power consumption at fixed, configurable intervals. If the power consumption falls below the threshold, a software timer is reset and started, while if it goes above the timer is stopped. The board activates the relay and switches off the machine if the timer is not halted before the callback is triggered. The initialization of this timer occurs when the task function starts.

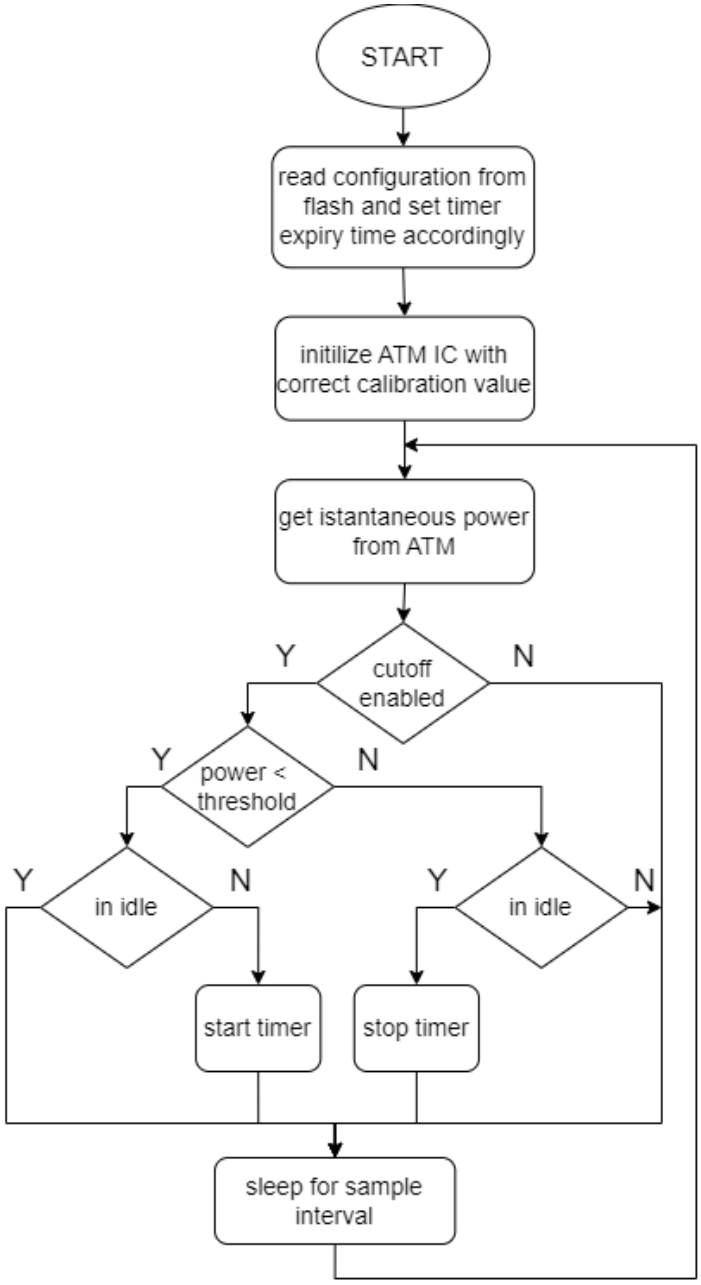

Figure 4.11: Power task function flow chart

4.1.5 IDC task

IDC connector has been designed to Extend main board functionalities using expansion boards. This connector brings out the RX and TX signals of a UART, which means that expansion boards have to use this serial connection to exchange data with the main board. Currently, only the CAN expansion board is supported, since the RS485 expansion board is still under development. The board processes the data from the CAN expansion board as a virtualized input. The standard message consists of a start byte, two data bytes and one byte for the CRC, evaluated with the CRC8 algorithm [9].

Frame			
Starting char	Byte 0	Byte 1	Byte 2
%	0xyy	0xyy	CRC

Table 4.1: CAN expansion board frame

From the configuration file is possible to configure this virtual input in normal, counter or function mode just like physical ones. In case of normal and function mode, in the serial frames the data bytes contains only 0 or 1 values, indicating whether the input/function is active or not. In counter mode, instead, data bytes are interpreted as a count, that the board accumulates. By using this strategy, it's possible to manage a wide variety of data since, at the board level, they are treated as simple counters. They will acquire meaning when they reach the cloud platform, where their units of measurement will be assigned. For example, considering two machines in which the board has to measure wire consumption. This information is directly provided by the machine on a CAN-BUS line, received by the CAN expansion board and sent to the main board as described above. The two machines

uses different unit of measurements, centimeters for the first, inches for the latter. The main board will acquire the data in the same way and send them to the cloud platform as dimensionless count. When this data will reach the platform, the correct meaning is assigned and showed to the user with the correct unit of measurement. Obviously, this require an additional configuration step on the cloud platform, which is easier than configuring it on the board. The task function configures UART peripheral at the beginning and then, when some data are available, checks if starting byte and CRC are correct and then increment the counter inside the virtual input data structure or save the new state.

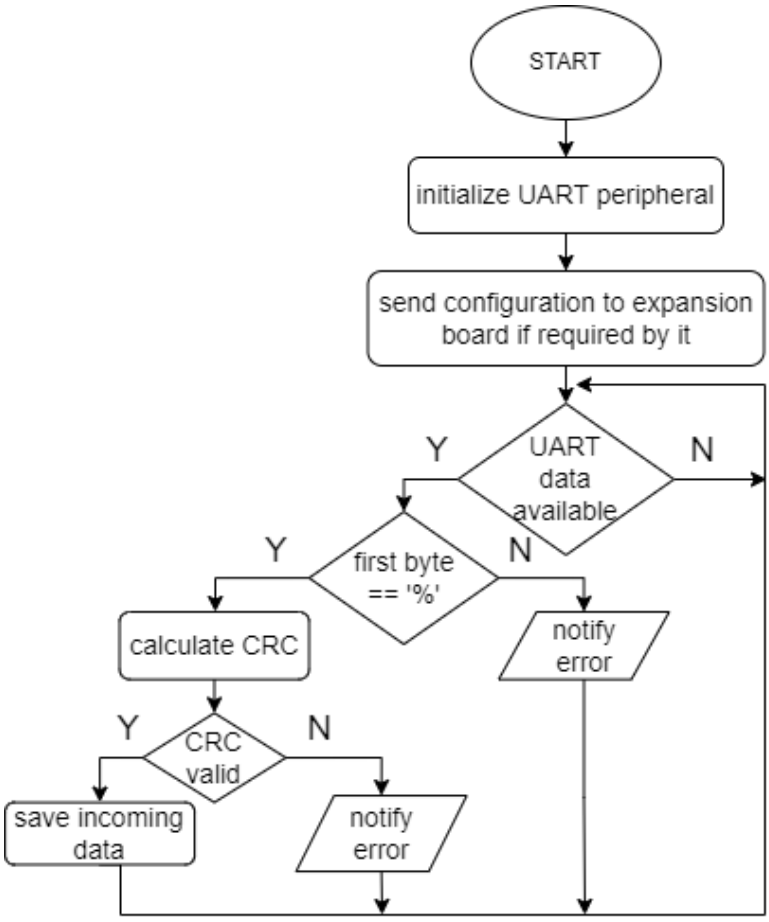

Figure 4.12: IDC task function flow chart

4.1.6 Localization task

ESP32 microcontrollers natively embed Bluetooth connectivity, which can coexist with Wi-Fi. However, they share the same antenna. Consequently, Bluetooth cannot receive or transmit data while Wi-Fi is actively receiving or transmitting data, and vice versa. ESP32 uses the time-division multiplexing method to handle the reception and transmission of packets.

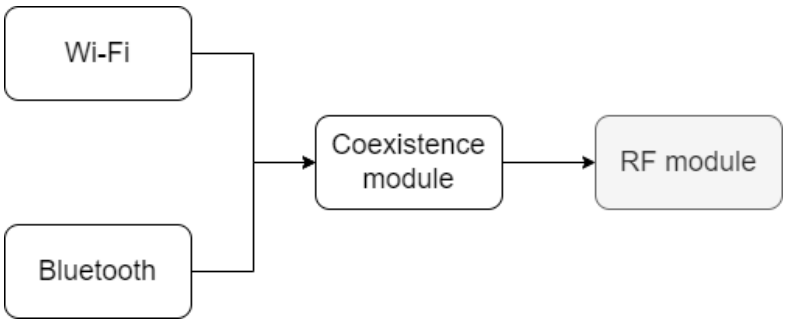

Figure 4.13: ESP32 RF coexistence [10]

The possibility of using also BLE has led to the development of a method for locating machines within the production plant. This is particularly useful for smaller machinery that can be moved. To enable this localization capability, an external infrastructure must be installed inside the plant. It consists of several BLE gateway that can be detected by the Trace&Follow board during a BLE scanning operation. The localization algorithm is based on RSSI (Received Signal Strength Indicator) received by the main board when it encounters a new gateway. Higher RSSI indicates a closer gateway while lower RSSI means a farther gateway. However, it's important to note that objects between the board and a gateway can drop the RSSI value. This is generally not a significant issue since most plants have wide, open spaces. During system installation, careful analysis of floor plans is

conducted to determine the optimal gateway placements. Furthermore, the required precision is in the order of meters, so even if the board occasionally considers a slightly more distant gateway as closer, it does not represent a major problem.

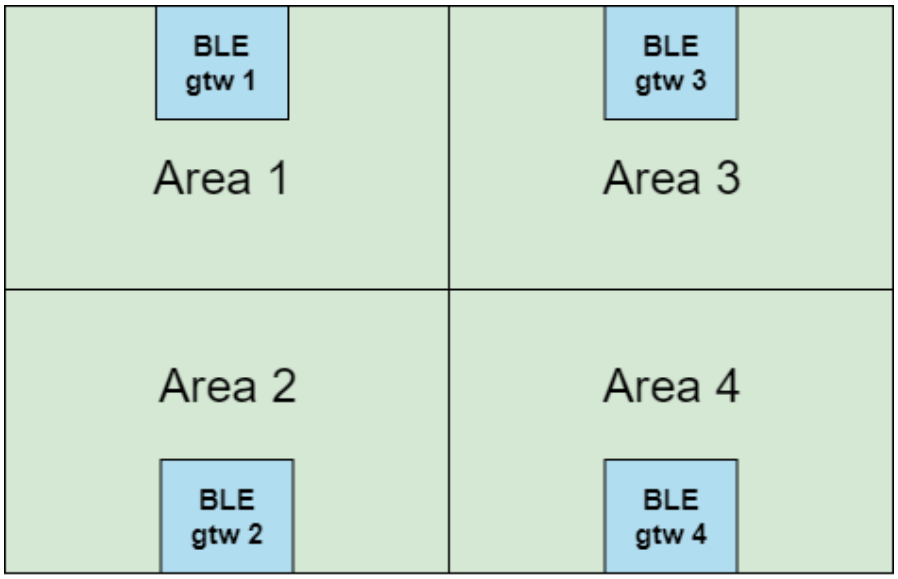

Figure 4.14: Localization areas

To locate gateways, the main board performs a scan operation. The discovered devices are stored in a vector and analyzed to determine whether a device is a gateway or not. This sorting is based on Bluetooth device names, which for gateways starts with a standard identifier. When the scan ends, the board analyzes gateways RSSI value to determine the closest one, which is saved. If it differs from the closest gateway found during the previous scan, a flag is set to notify the packet task that it needs to be sent to the cloud.

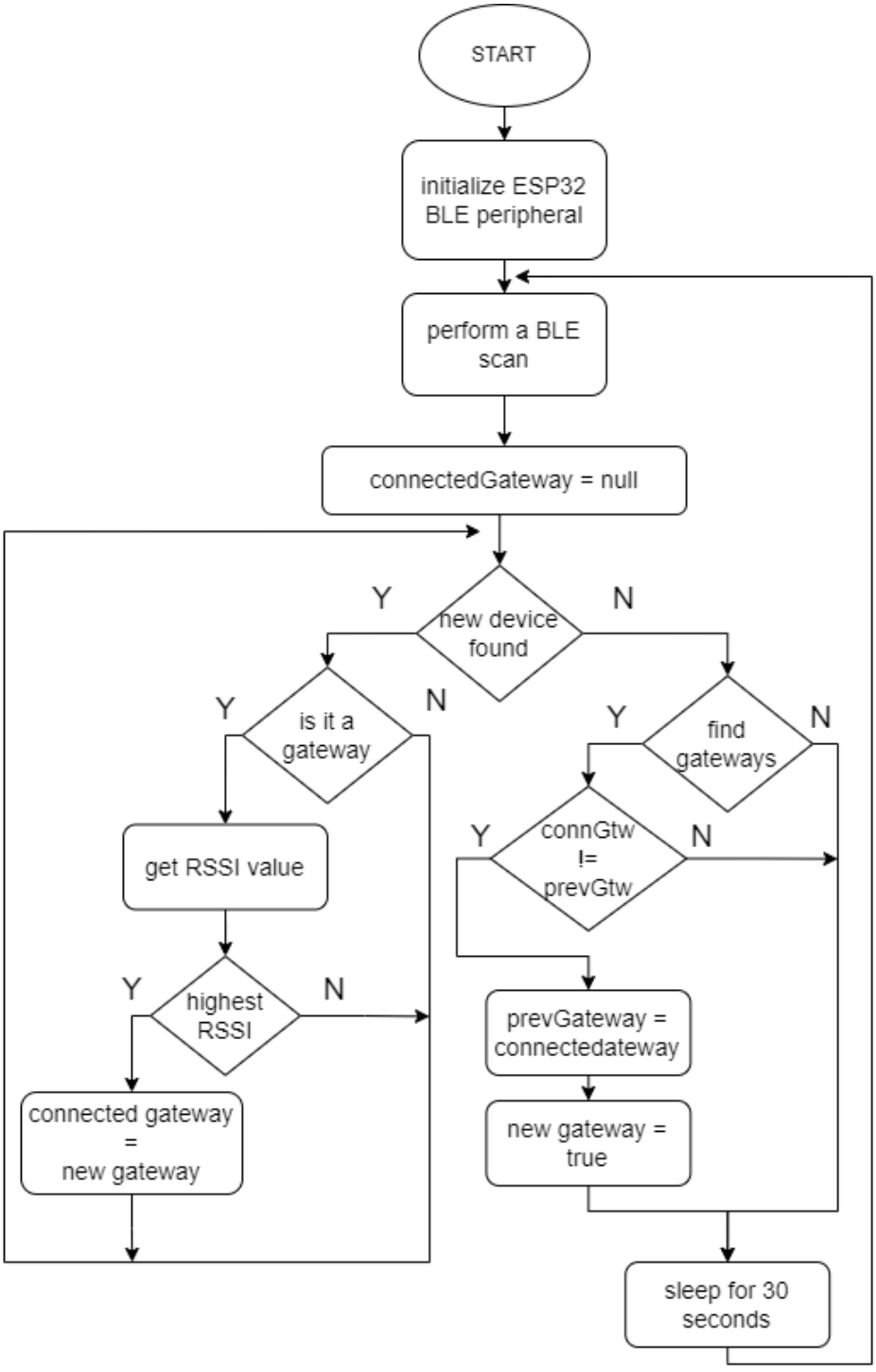

Figure 4.15: Localization algorithm flow chart

4.1.7 Packet task

While previous analyzed tasks are responsible for data acquisition, this task's role is to organize and send them to the first of the cloud tasks. Once again, a queue is employed to transfer data from this task to the cloud task. The most important step the task perform is the assignment of the timestamp to each data. This step is particularly important when using LoRa connectivity because it doesn't automatically provide the current date and time. To solve this issue, timestamp associate to each value is relative to a starting point, which coincides with the moment the connection to the cloud is established To keep track of the current time, we rely on FreeRTOS ticks. A tick represents the smallest unit of time that the operating system can measure. For example, if the cloud connection is established at tick = 1000, and a data packet is created at tick 5000, the timestamp associated with that packet would be 5000-1000 = 4000. When this packet reaches the cloud platform relative TS will be translated into the absolute one. The required values are extracted from data structures managed by other tasks. To ensure the integrity of the retrieved data, each of these structures is protected by a mutex.

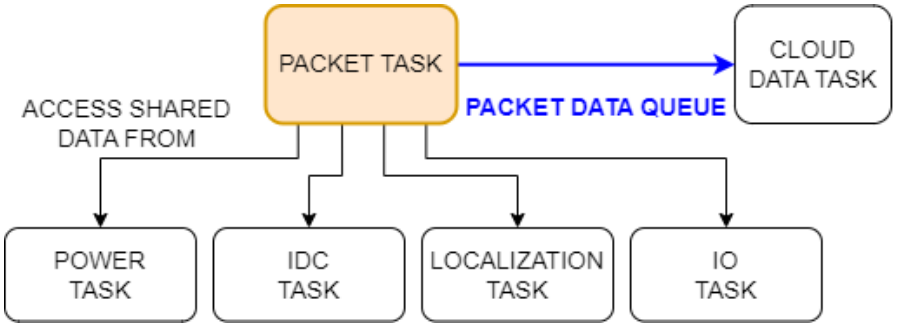

Figure 4.16: Packet task interactions

After each data is read, and its timestamp has been assigned, they are sent, using

packet data queue, to cloud data task, which stores them in the appropriate buffer. The message sent via this queue is provided below.

```
typedef struct
{
    uint32_t    timeStamp;  //timestamp of the acquire data
    uint64_t    uintData;   //acquired data
    uint8_t     type;       //flag that tells data meaning
    uint8_t     num;        //required if there are more instance of
   the same object
} packet_queue_msg_t;
```

- **timestamp:** timestamp associated with the transmitted data.
- **uintData:** the data transmitted.
- **type:** an integer value representing the type of data present into the message.
- **num:** used only in case there are multiple entity of the same type, such as inputs or outputs.

Time interval between consecutive sample is configurable in the configuration file and is called “Packet time”. The task waits for this time before entering in ready state, and after it is scheduled and executed for one loop cycle, it comes back to blocked state. To reduce the number of transmissions, all periodic data is sampled at the same time and then inserted into the same packet, which means they have the same timestamp. Since data are sent individually, an "end of packet" message is sent after the data messages to simplify packet creation and reception by the cloud task. The end-of-packet message doesn’t carry any information but serves as

an indicator for the cloud task to recognize when a packet is complete. Packets are not directly created within this task and sent already grouped. This approach allows us to manage asynchronous data that also needs to reach the cloud task, and it enables the use of a single queue for handling all types of data flows.

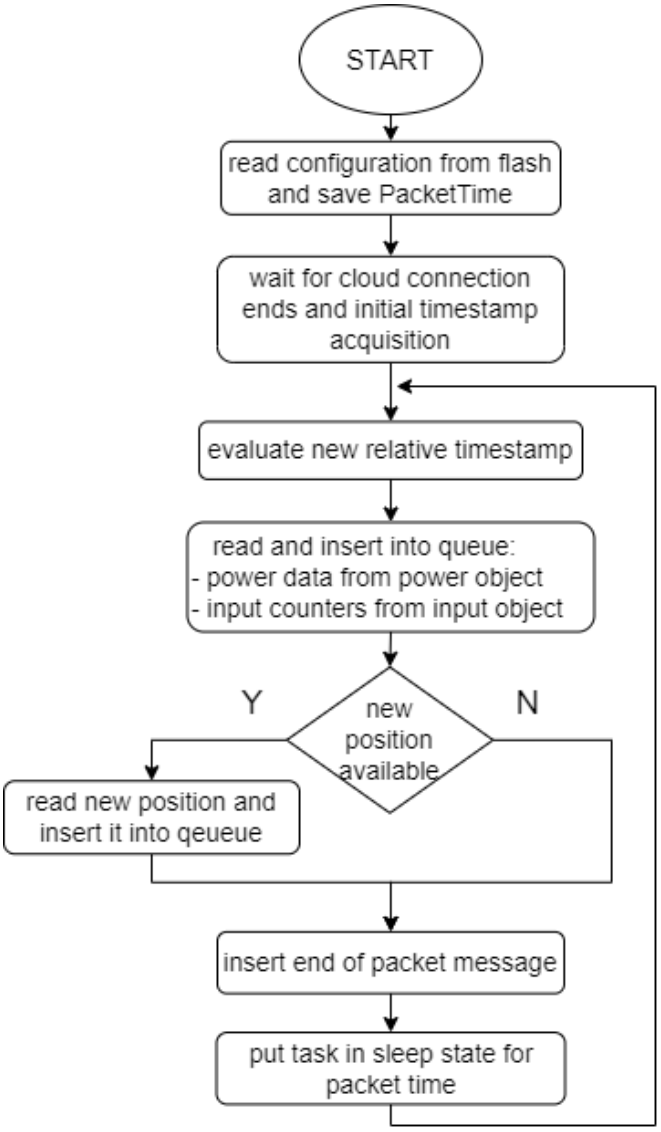

Figure 4.17: Packet task function flow chart

4.1.8 Cloud task

There are two tasks that manage cloud connection and data sending.

- **Cloud data task:** this task is responsible for receiving data from other tasks using a queue and prepare them to be sent to cloud.
- **Cloud loop task:** this task is responsible for managing the connection to cloud and send data when sending condition are satisfied. These conditions differ based on the used protocol (Wi-Fi, Ethernet, LoRa, 4G)

Cloud data task organize single received data into packets or save them if they belong to the asynchronous type. To better organize data, packets are managed using a class, which is presented below.

```
class Packet
{
public:
    // ... public methods
private:
    uint32_t            _timestamp; //packet timestamp
    int32_t             _power; // power value
    uint64_t            _position; // position if available
    uint8_t             _inSize = 0; //counter array size
    input_cnt_element_t _inputCounters[10]; //array of counters for
    input with mod=COUNTER
    uint8_t             _serializedPacketDimension; //dimension of
    the packet when serialized for LoRa interface
    bool                _ready; //if=1 pck is ready to be processed
};
```

The cloud global structure is kept in memory instantiating a class object. The class structure is here reported.

```
class Cloud
{
public:
    // ... public methods
private:
    //pointer to cloud interface base class
    CloudInterface *                _cloudInterface;
    //buffer for ready packets
    std::vector<Packet>             _packetBuffer;
    // maximum allowed buffer elements
    uint16_t                        _maxQueueElements;
    //buffer for asynchronous data
    std::vector<packet_queue_msg_t> _asynchDataBuffer;
    //mutex to protect data access
    mutex_t                         _dataMutex = {
                                        .handle = NULL,
                                        .name = "cloud data mutex"
                                    };
};
```

It includes:

- A pointer to the physical cloud structure used to interact with different modules based on the required connectivity. This concept will be explained later in this chapter

- Packet and asynchronous data buffers implemented using vector objects provided by the C++ libraries
- Mutex for protecting data access

The task saves data into packet object and, when a "end of packet" message is received, sets the ready flag, allowing it to be saved into packet buffer. Asynchronous data coming from queue are directly saved into asynchronous buffer.

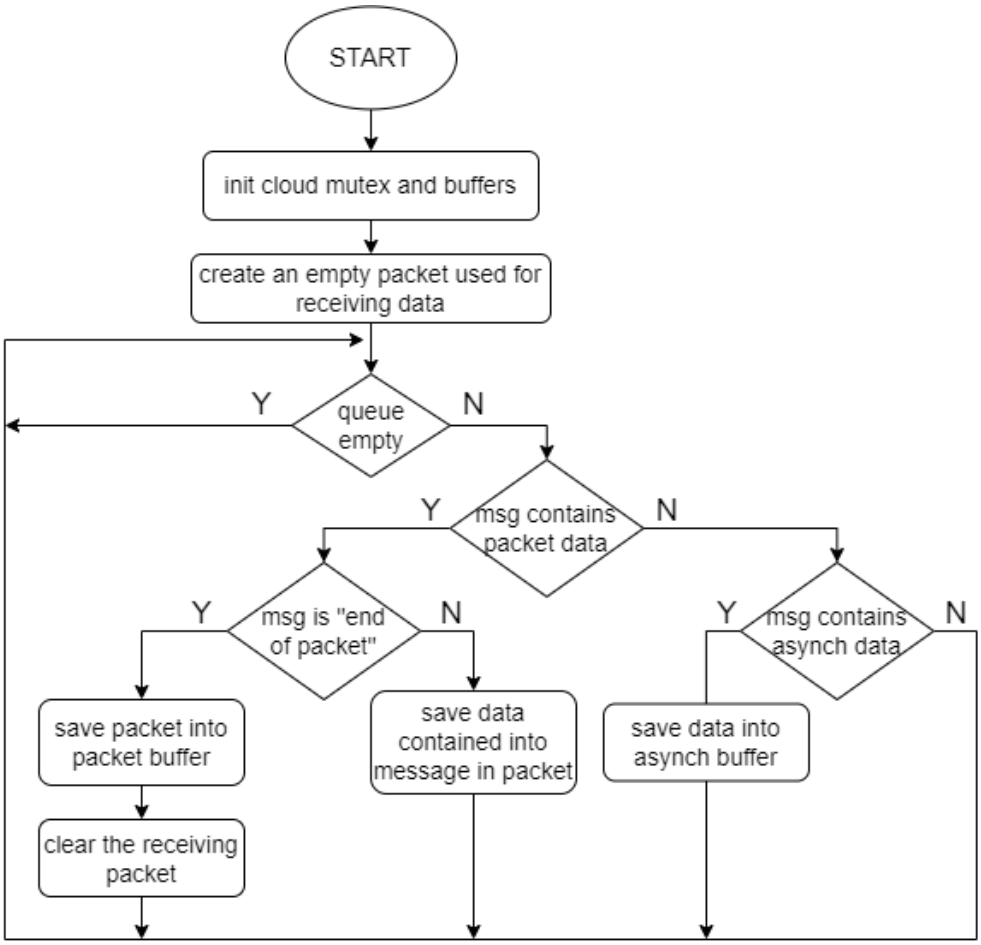

Figure 4.18: Cloud data task function flow chart

Once the data are correctly placed into buffers, cloud loop task is responsible for sending them according to the configured connectivity interface. Since we don't know at compile time which interface needs to be used, we need to keep the code generic. Each interface has its own data structures, its own methods to interact with cloud. These interfaces are kept in memory using classes and object. One option we have is to instantiate all the four objects, one for each interface, and then select the active one at the beginning. However, this approach would lead to complex and messy code because we'd have to call dedicated functions for each operation, and it would consume significant RAM to store unused variables for unused interfaces. To address this issue, we can use C++ concepts of inheritance and pure virtual classes. As shown earlier, in the cloud class, there is only one pointer to a generic cloud interface:

```
1            CloudInterface * _cloudInterface;
```

This class has been defined as a pure virtual class. This means that it is not possible to instantiate it, but it can be used as a template for derived classes.

```
1 class CloudInterface
2 {
3 public:
4     CloudInterface(){}  //constructor
5     ~CloudInterface(){} //deconstructor
6
7     virtual bool Init() = 0;     //init method
8     // other virtual methods
```

```
protected:
    int           _state = 0; //connection state
    TickType_t    _lastSend;  //timestamp of last data sending
    uint32_t      _transmissionInterval; //time between transmission
    mutex_t       _mutex = {
                  .handle = NULL,
                  .name = "cloud mutex"
              };  //data mutex
private:

};
```

Looking at the *Init()* function (other ones have been omitted), it is defined as a 'virtual' followed by '= 0'. This notation means that this function must be redefined in each of the derived classes. Failure to do so will result in a compiler error. From this class, individual interface classes are derived. For example, the LoRa interface class is provided below. As it is possible to see, the *Init()* function is defined without the 'virtual' and the '= 0' qualifier, as this is not a pure virtual class.

```
class CloudInterfaceLora : public CloudInterface
{

public:
    CloudInterfaceLora();    //constructor
    ~CloudInterfaceLora();   //deconstructor

    bool Init();             //init method
    // other public methods
```

```
private:
    // private methods related to LoRa management

    LoraModule       _loraModem; //lora modem object
    lora_config_t    _config;    //config struct
    bool             _isPrevError = false; //have errors occurred yet?
    int              _consecutiveSendingError = 0; //errors number
    receivedData_t   _receivedData = {
                         .buf =  {'\0'},
                         .size = 0
                     }; //struct in which received data is saved
    TickType_t       _baseTimestamp = 0; //initial timestamp
    bool             _waitingForDownlink = false; //if=1 the reception
    window is open
};
```

Since this class is derived from the *"CloudInterface"* virtual class, it is possible to assign a pointer of type *"CloudInterfaceLora"* to a pointer of type *"CloudInterface"*. This enables the calling of functions declared as pure virtual in the base class, which have been redefined in the derived class. This is the reason for which in the cloud class there is only a pointer to base class. This is why there is only a pointer to the base class in the cloud class. The derived interfaces can have their own variables to perform operations; for example, the LoRa class contains the *"LoraModule"* object used to interact with the physical modem IC, which other interface classes do not have. However, they follow a similar structure. The selection of the interface occurs at the beginning of the cloud loop function. The interface that needs to be used is saved into the configuration file Once this information is retrieved, the associated object is dynamically allocated as follows.

```
1     _cloudInterface = new CloudInterfaceLora;
```

From this point on, every function called using this pointer is the one defined inside LoRa interface. Let's take the *Init()* method as example. It is called inside the cloud loop task function, where the interface is initialized. If the code would be:

```
1     _cloudInterface = new CloudInterfaceLora;
2     _cloudInterface->Init();
```

This means that the UART peripheral is initialized, since the LoRa expansion board needs to be mounted on the PCIe connector, that the PCIe LDO must be enabled to supply power to the board and finally the LoRa modem is started. If, instead, the code would have been:

```
1     _cloudInterface = new CloudInterfaceWiFi;
2     _cloudInterface->Init();
```

only the internal Wi-Fi module is initialized, and the previously described operation are not executed.

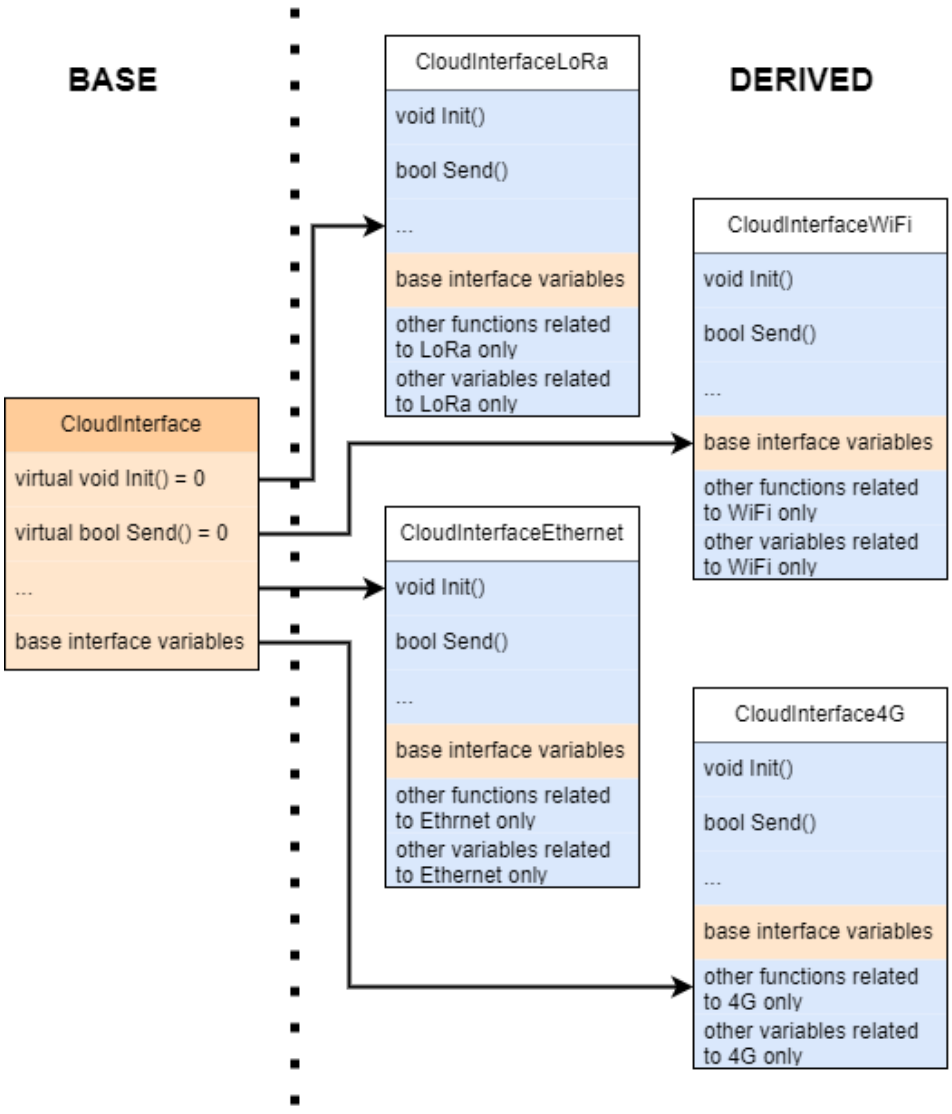

Figure 4.19: Connectivity interface classes

Cloud loop task performs three important steps. It starts initializing the correct interface and sending the starting message to cloud. Then every loop cycle it checks if the connection with cloud is still active and, if all the sending condition are satisfied tries to send data saved into buffers.

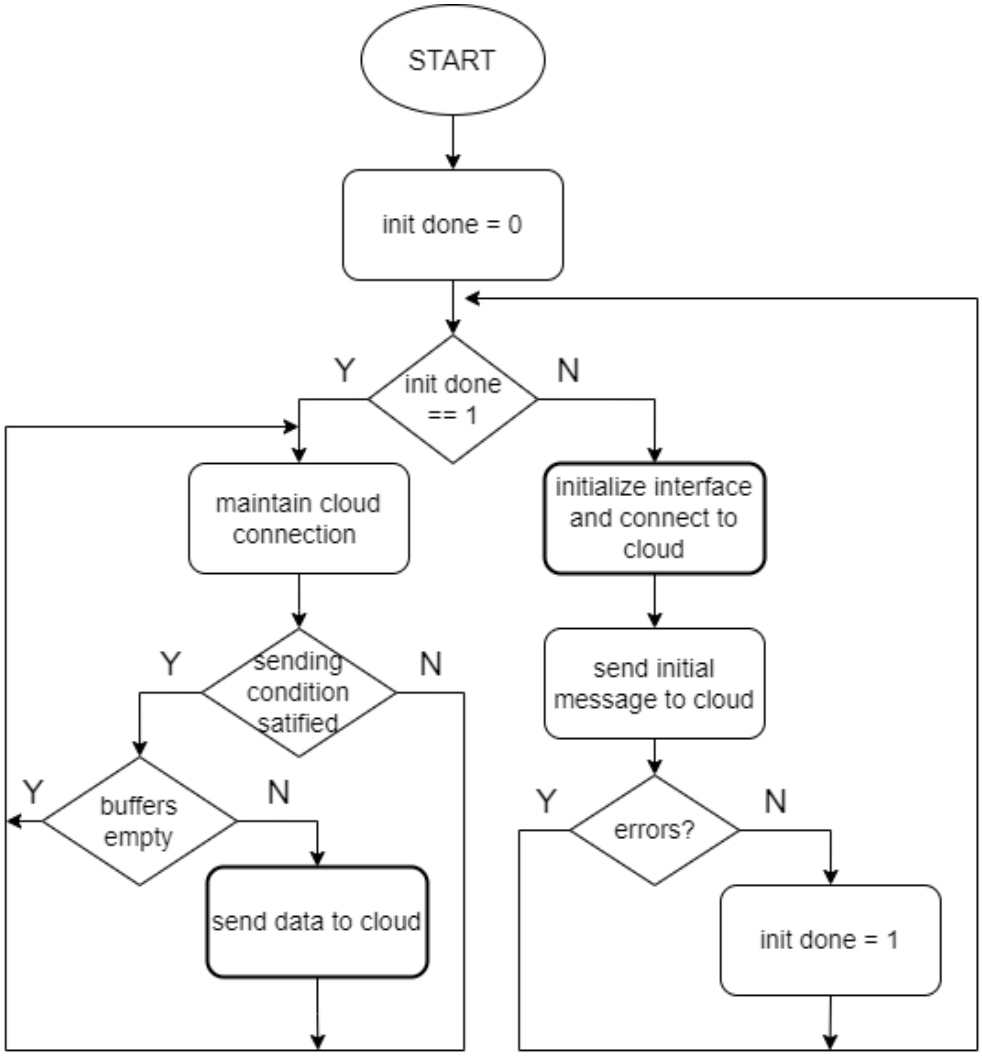

Figure 4.20: Cloud loop task function flow chart

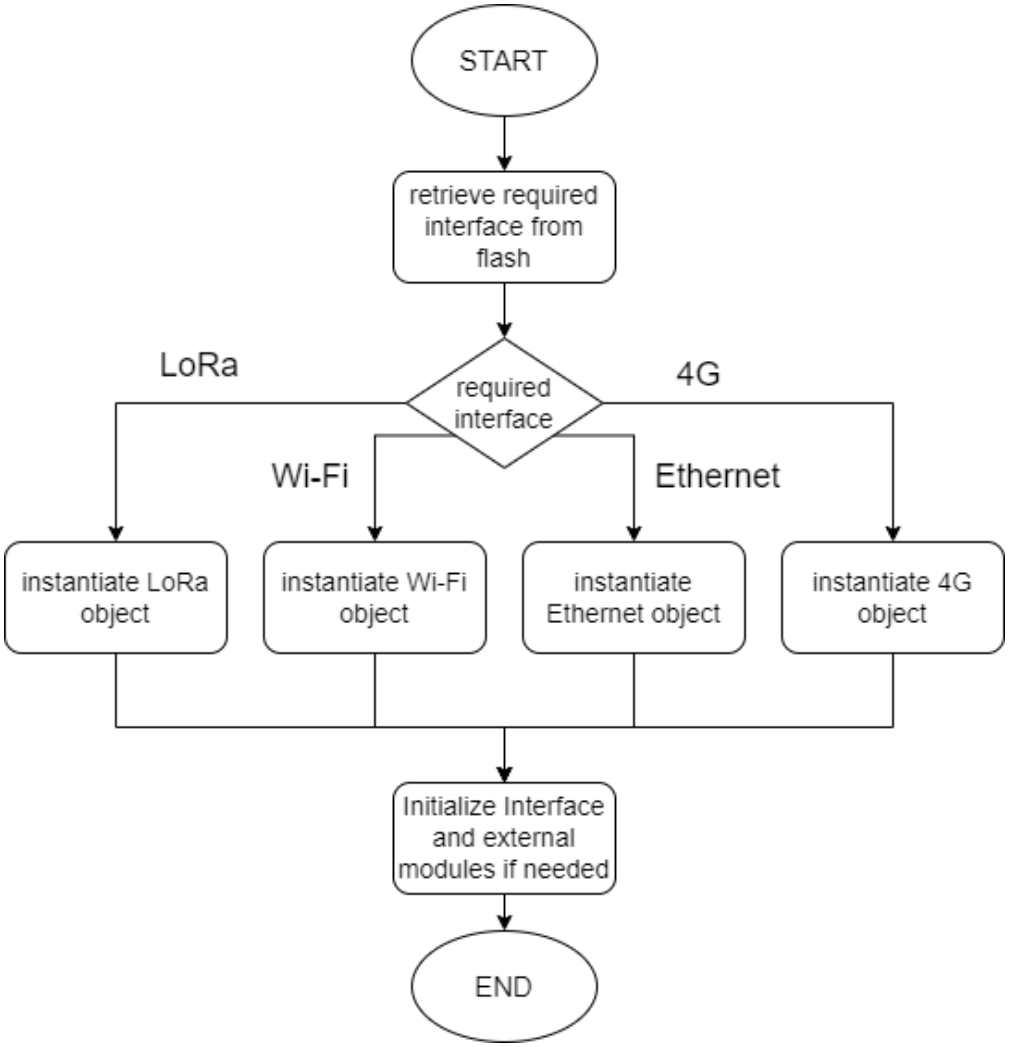

Figure 4.21: Cloud init function flow chart

During initialization phase, the most important step is the interface object instantiation, that is executed as previously described. Once it is correctly configured, the board sends a first message to cloud containing:

- Firmware and hardware versions
- Inputs, outputs and cutoff configurations
- Packet creation and cloud parameters

This message is very important because allow the cloud platform to synchronize its initial timestamp with the board timestamp. Consequently, the whole program does not start until this sending operation succeeds, otherwise the received data will be associated with a wrong timestamp.

Once the initial message has been sent correctly, the loop starts. At each cycle, the connection state to the platform is checked, and if some problem arises, a reconnection attempt is made. The exact procedure depends on the interface currently used. After that, if all the sending condition are satisfied the system tries to send data to cloud. Conditions includes correct cloud connection and minimum sending interval elapsed. From configuration file, indeed, it's possible to set a minimum interval between two consecutive transmissions. This is necessary when using LoRa connectivity because, because of the physical requirements derived from the used LoRa connection class, which impose a minimum time between transmissions.

The sending operation and data format depend on the type of interface being used, which falls into two groups. When using Wi-Fi, Ethernet, and 4G interface, the board is directly connected to the cloud. Data is sent to the platform using the MQTT protocol, and it is formatted using a JSON document. As the device utilizing LoRa connectivity type is not directly linked to the cloud platform (with the gateway and network server acting as intermediaries), and due to the limited data transmission capacity over the physical channel, the transmitted payload requires compression into a byte array instead of JSON document, which size is much bigger. Consequently, this array will be interpreted by the code running on the network server and translated into a JSON document formatted as in other interfaces. However, the process for constructing the payload remains the same:

asynchronous data is added first, followed by packets. Prior to adding any data, the payload dimension is checked to ensure it does not exceed the maximum allowed payload size. In case any errors occur during transmission, the board will notify the user using a color sequence on the RGB LED.

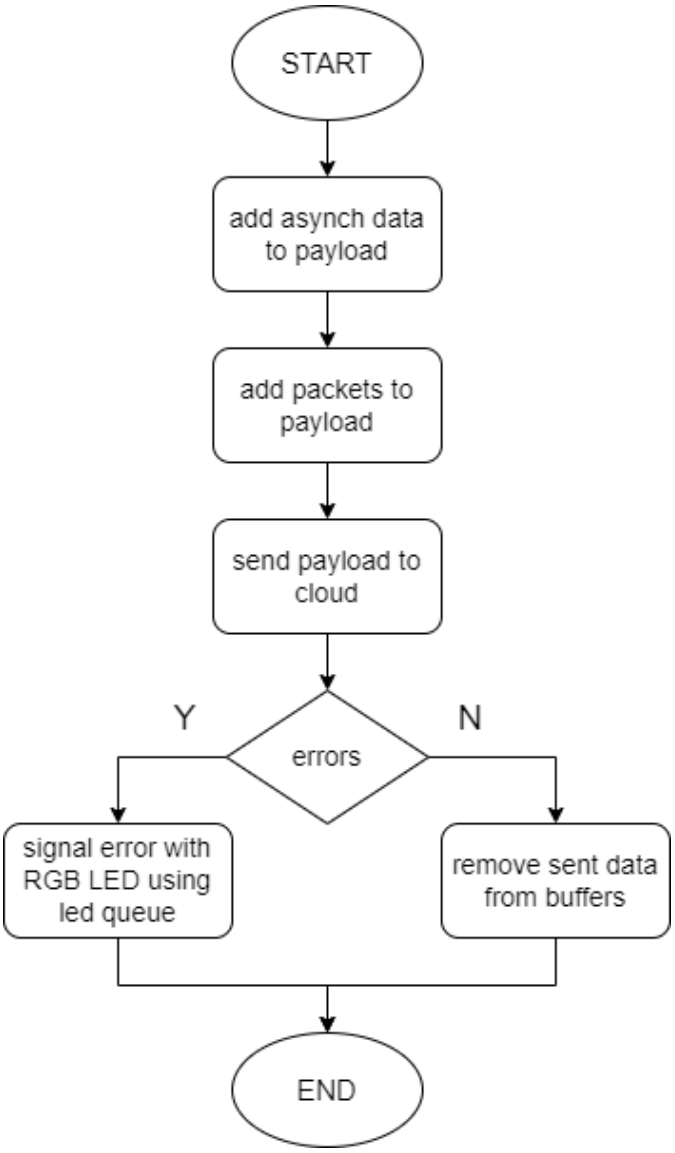

Figure 4.22: Send data function flow chart

4.2 Cloud platform

Once data have been sent from the board, they are collected by a cloud platform, in our case it is called ThingsBoard. ThingsBoard is an open-source IoT platform for data collection, processing, visualization, and device management [11]. it supports multiple protocols to for receiving data, but this project only uses MQTT.

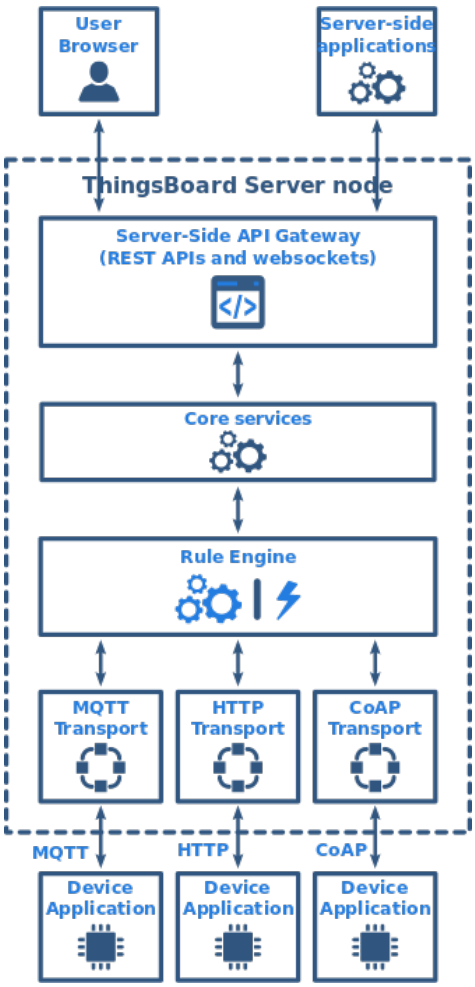

Figure 4.23: ThingsBoard architecture overview [12]

The most crucial element of the system is the Rule Engine, which enables the filtering of incoming messages using various node types. The script nodes allow for the

insertion of JavaScript code to accomplish complex operations. The Trace&Follow rule, visible in Figure 4.23, is an example of this.

Figure 4.24: Trace&Follow board rule chain

All received messages are sorted according to their typology. For instance, when the starting message is received, the software separates all message fields. It then saves them to the database and records the current UNIX timestamp, representing the time as the number of seconds that have passed since January 1, 1970 at 00:00:00 UTC (Coordinated Universal Time). This timestamp will be utilized to

convert the relative timestamp associated with the message coming from the board to an absolute one. For instance, if the starting message arrives on 01/09/23 at 11:00:00, the corresponding UNIX timestamp would be 1693566000. The board will save this value as the base timestamp. Upon receiving a message from the board containing a packet with a relative timestamp of 1230, this value is added to the base timestamp to obtain 1693567230, corresponding to 01/09/23 at 11:20:30. This newly calculated timestamp is then associated with the data contained within the packet. ThingsBoard provides APIs (Application Programming Interface) for data retrieval and display in user applications after the values have been stored in the internal database. It is also possible for user applications to send commands to the cloud platform, which then forwards them to the device using a specific MQTT topic. On the board, these commands are managed as RPCs (Remote Procedure Calls). RPC is a request-response protocol, where the client initiates the request by sending a message to a known remote server to execute a specified procedure with supplied parameters. The remote server responds to the client, and the application can proceed with its process. In this scenario, the Trace&Follow board functions as the server, while the cloud platform functions as the client.

4.3 ChirpStack for LoRa

When using Ethernet, Wi-Fi or 4G connectivity to interact with the cloud, the connection to the cloud is direct, meaning there is no other software or physical component between the board and the platform. They exchange data using the MQTT protocol. This is not possible when using LoRa connectivity instead, as it does not provide direct internet access. LoRa frames are then received by a gateway equipped with an internet connection, which forwards these frames to a network server that can handle them. This project uses ChirpStack. ChirpStack is an open-source LoRaWAN Network Server which can be used to setup LoRaWAN networks. ChirpStack provides a web-interface for the management of gateways, devices, and tenants as well to setup data integrations with the major cloud providers, databases and services commonly used for handling device data[13].

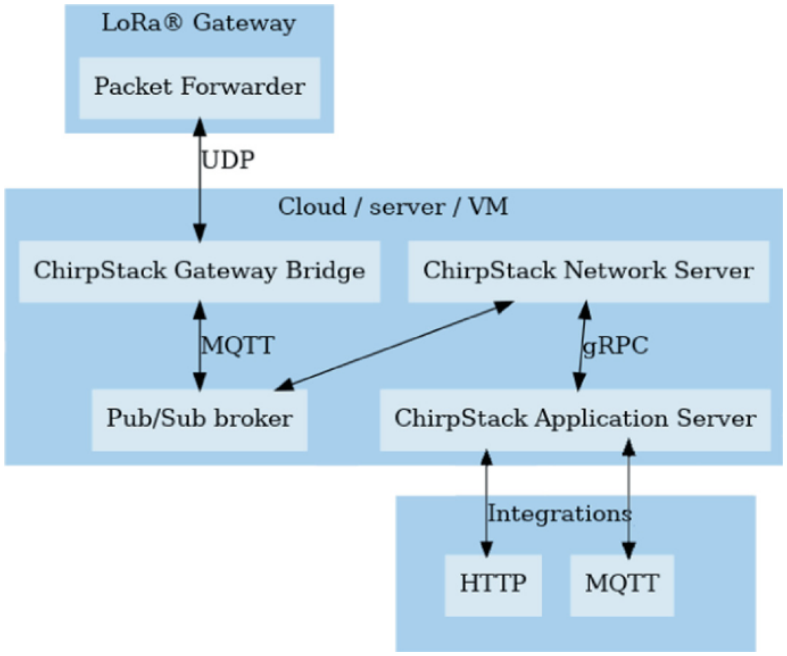

Figure 4.25: ChirpStack architecture [14]

Once the frame reaches the Application Server, it is analyzed by a JavaScript code running on it. The code is designed to convert the byte array within LoRa frames into a JSON document that is formatted similarly to the documents sent by the board. This eliminates the need to construct a customized rule chain on ThingsBoard, while still allowing the use of the rule chain shown in Figure 4.24, despite of the T&F board's type of connectivity. The whole LoRaWAN architecture in reported in Figure 4.26

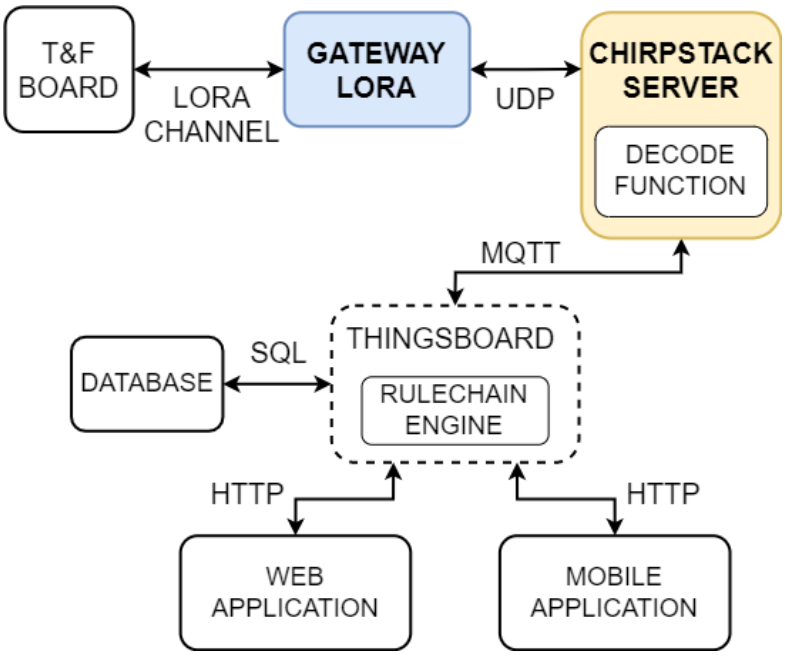

Figure 4.26: LoRaWAN architecture

Chapter 5

First Trace&Follow installation

5.1 Overview

After completing the development phase, we proceeded to install the system on the first machine which happened to be a manual welder. It is important to note that the Trace&Follow board cannot be mounted on its own, but instead requires other components. To ensure proper installation, we have included it within a steel box that contains all the necessary elements. The board utilizes LoRaWAN connectivity as the welder may be moved, making Ethernet connection unsuitable.

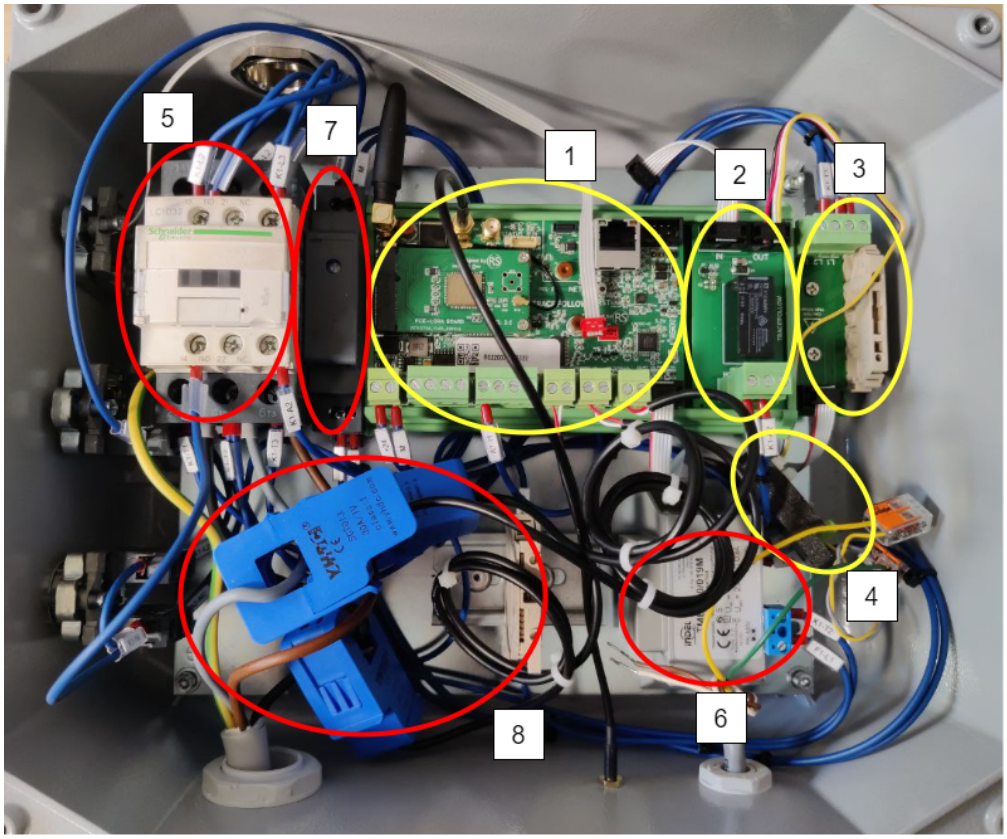

Figure 5.1: Complete system inside its steel box

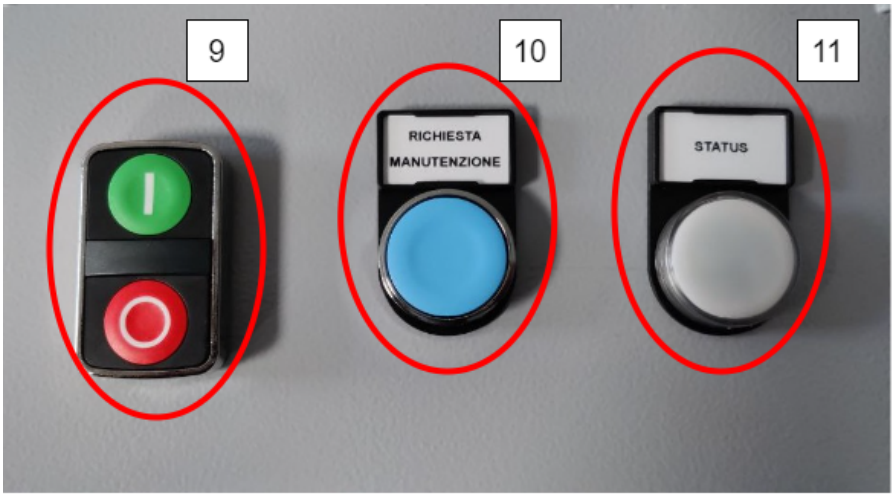

Figure 5.2: Top view of the steel box

1. Trace&Follow board with LoRa expansion board mounted on PCIe connector
2. Relay expansion board used to implement cutoff feature
3. Voltage sampling board
4. CAN expansion board enclosed in a 3D printed case
5. Contactor needed for cutoff feature
6. Transformer 400V AC – 230V AC
7. Transformer 400V AC – 230V AC
8. CT for current measurement
9. Switch on/off button
10. User button
11. Status led

5.2 Cutoff

The system includes a cutoff feature to turn off the machine either automatically or manually via a command from the user application.
Two of the machine's three-phase supplies are utilized for this feature. The start/stop button is located on the top of the steel box to switch the machine on and off. At the start, to power on the machine, simply press the green button which closes its NO contact. Since both the board relay and red button are connected to NC contact, phase 1 reaches the coil of the contactor. Another phase on the

opposite side of the coil energizes it, resulting in the closure of the NA contacts of the contactor. Since Phase 1 is currently present on the coil, it remains energized and its normally open contacts remain closed. Pressing the stop button (red) or activating the relay causes the NC contacts to open, stopping Phase 1, which will no longer be present on the coil. This means that the normally open (NA) contact of the contactor returns to an open state, causing the machine to switch off.

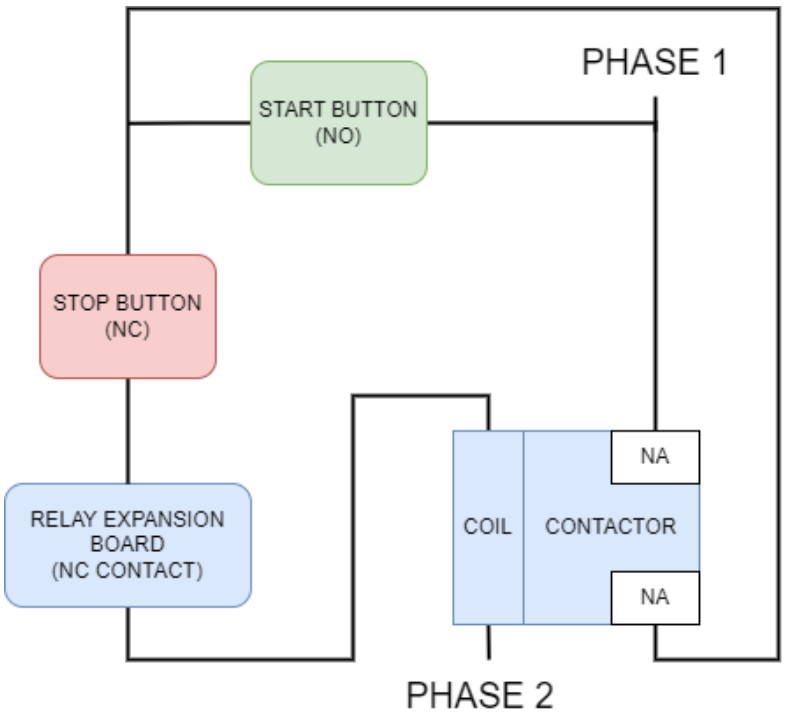

Figure 5.3: Cutoff element electrical connection

5.3 Wire monitoring

On this welding machine, customers wanted to know how much welding wire they were using, in addition to monitoring the power consumption. To address this, the machine's internal CAN-BUS line, which carries data related to wire speed, is read and converted into a length value with the help of the CAN expansion board. After conversion, the value is transmitted to the main board, where it is treated as a virtual input configured as a counter and sent to the cloud. In this manner, it is feasible to derive a curve similar to that of the power consumption, which can be used both to better estimate the cost of each piece, and to compare it with a reference in order to identify a potentially faulty piece. In Figure 5.4 the acquired power consumption and wire plots are reported.

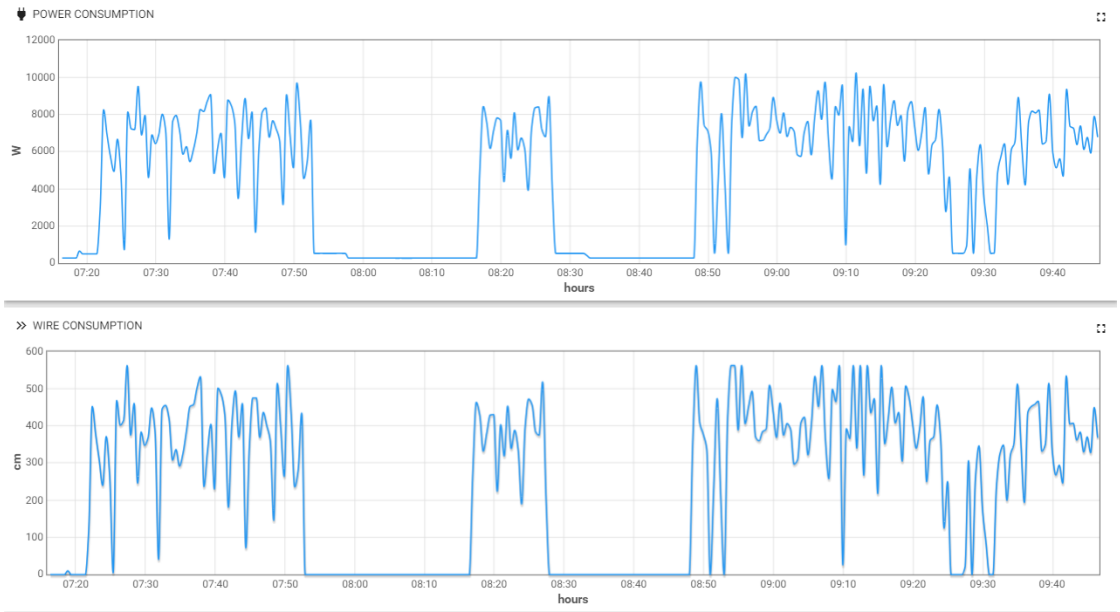

Figure 5.4: Waveform for power and wire consumption

5.4 Localization

Other than maintenance, the customer requested a way to determine the approximate location of each machine within the plant. Thus, the localization feature has been enabled, particularly helpful for the maintenance department to locate faulty welders. Several BLE gateways were installed throughout the production plant, dividing it into zones. When the Trace&Follow board finds a gateway, it sends its identifier to the cloud, which associates the welder with the zone identified by the gateway. The application display will then show this information, as demonstrated in the screenshot provided in Figure 5.5.

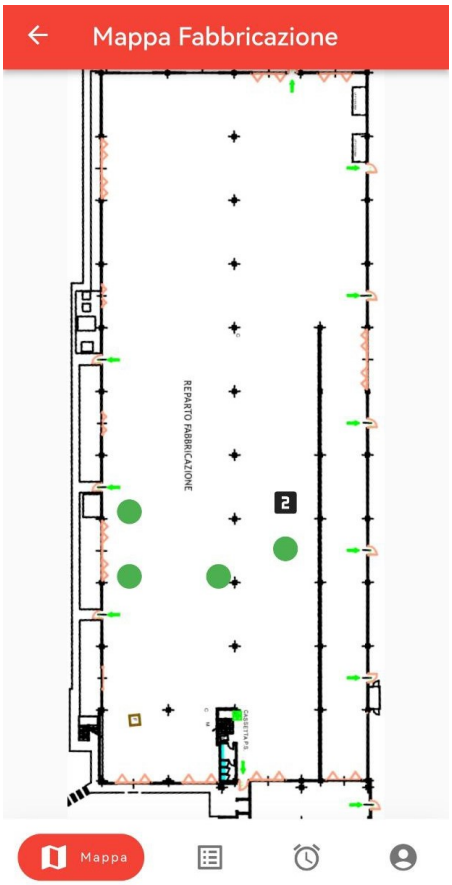

Figure 5.5: Screenshot showing localization feature

5.5 Maintenance

The customer requested a means of knowing when maintenance is required for the machine. To comply with their request, a board input has been configured in function mode to connect a button. By pressing the button, the operator can signal that the machine is not working properly and requires maintenance. When the button is pressed, the associated function is activated and the new state is sent to the cloud. Then, the user application displays that this particular machine requires maintenance, and the maintenance department can be notified of the request. After the issue has been solved, the function can be reset by either holding the button for three seconds or resetting it directly from the application. When the function is enabled, the status LED which is typically green under normal circumstances, changes to yellow allowing the operator and maintenance team to easily identify if the machine requires service.

5.6 Data visualization

Using a web or mobile application, it is possible to visualize the collected data. The mobile app organizes machines by plant and department. For each department a list of welders is present and indicates whether they are on or not. Each machine has a page showing the power and wire consumption for the selected period. Users have the ability to reset the maintenance flag, set the cutoff timeout, and turn it off.

Figure 5.6: Screenshots showing mobile app pages

The web application is designed only for data visualization, so it is not possible to set parameters on the devices. The mobile app displays a summary for all welders, distinguishing between robotic and manual ones, and presenting the total power consumption over the chosen period for the selected type of machine, in addition to the data already visualized in the mobile app.

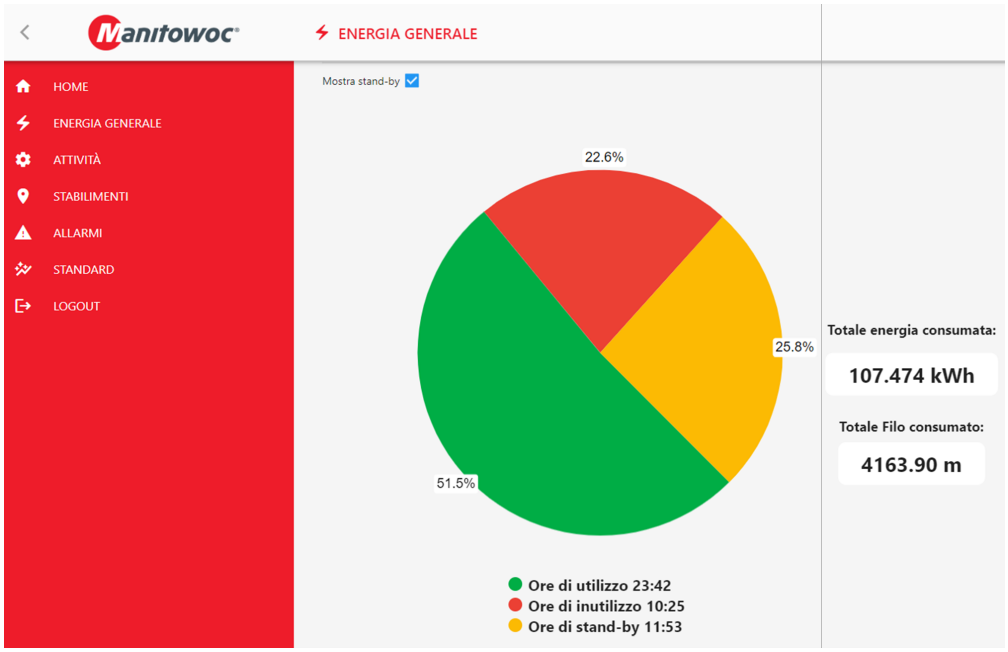

Figure 5.7: Web app total consumption page

MACCHINA	Ore di utilizzo	Ore di stand-by	Ore di inutilizzo	Energia [kWh]	Filo [m]
Machine RS62 R1	04:24	03:10	03:56	21.057	979.96
Machine RS66 R4	05:01	03:08	03:21	23.047	1166.66
Machine RS67 R2	07:04	04:25	00:01	25.314	0.00
Machine RS63 R3	07:13	01:10	03:07	38.055	2017.28

Figure 5.8: Web app machine overview page

5.7 Installed system

The positioning of the metal box, which contains the entire system, depends on the model of the soldering machine on which it is to be installed. In this case, since it was equipped with a cart, the box was anchored to it. All components within the box are mountable on a DIN rail. Therefore, the steel box is no longer necessary if the system is installed in an electrical cabinet.

Figure 5.9: Welder equipped with Trace&Follow box

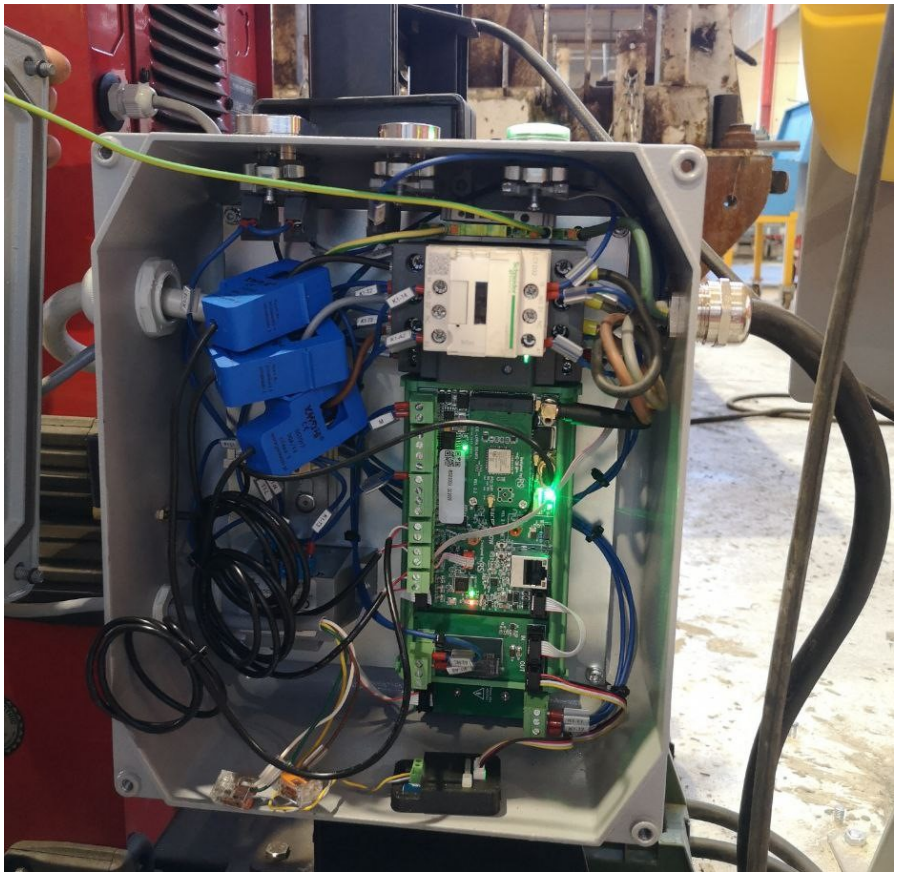

Figure 5.10: Content of Trace&Follow box

Chapter 6

Conclusions and future works

The initial version of the Trace&Follow board was limited to a specific machine working in a specific environment. The improved version created during this thesis resolves this limitation by being adaptable to a wider range of machines. Although the board has been specifically designed for industrial environments, it can also be used in civil environments, for example to monitor the consumption of an office or a house.

Thanks to its versatile inputs and outputs, this device can be employed not only for power consumption monitoring, but also for other types of data collection. Its flexible connectivity allows it to function in nearly any environment, whether it be a modern factory with Internet connectivity or an older plant, by selecting the most appropriate hardware configuration. Following a modular approach allows meeting customer needs while keeping costs low as only necessary modules

are installed. Additionally, this feature facilitates extending board functionalities, meeting the demand for custom implementations by simply developing an expansion board instead of designing an entirely new system. The board creates its own Wi-Fi network, enabling updates without requiring a programmer. This allows for placement in difficult-to-reach areas, as for example a locked electrical cabinet, while still receiving updates. The next evolutionary step in this area will be the implementation of updating through the cloud platform, so that boards can be updated remotely.

At the time of writing, eight boards are already installed and operational in a production plant near Turin. An additional three boards are scheduled for installation in the coming months. Furthermore, several companies have expressed interest in testing this system in their own production facilities.

6.1 Economical aspects

Aside from technical considerations, commercial factors also play a crucial role in product development. In order to demonstrate the advantages of this system it has been compared to an existing one based on a Siemens PLC, declined in two configurations: the first with basic and cheaper components and the seconds with more powerful hardware.

The primary advantage of Trace&Follow lies in its modular approach. It allows for the installation of only the necessary modules, which helps to keep the costs low. The production costs of electronic boards are dependent on the number of devices manufactured. As the number of units increases, the cost of both components and PCBs decreases. This results in a reduction in the price at which the company sells the device. It is important to note that the Trace&Follow board can be utilized in two different modes. This board can function as a component within the Trace&Follow product in combination with the cloud platform, web and mobile applications, or it is possible to specify, through the configuration file, a different platform to which the board will send the acquired data using the MQTT protocol. This results in lower prices since there are no software licenses included.

It should be noted that the listed values indicate the sale price of the board, not the production costs to obtain a better comparison. All prices are vat excluded. The tables below outline the components of the commercial PLC-based systems used for comparison. The selected PLCs have similar I/O configuration of Trace&Follow board. The setups are based on two types of power meters, both equipped with an RS-485 bus to communicate with the PLC. In addition, 4G and CAN modules have been inserted to compare the possible expansion of the Trace&Follow board.

For the base configuration the components are:

PLC Siemens LOGO! (8 inputs, 4 outputs)[15]	180 €
RS PRO power meter with RS485 bus[16]	194 €
Siemens 6NH3112-3BA00-6XX1 RS485 expansion module[17]	348 €
TOT	**722 €**

For the advanced configuration the components are:

PLC Siemens SIMATIC S7-1200 (6 inputs, 4 outputs)[18]	241 €
Finder power meter 7M[19]	337 €
Siemens 6NH3112-3BA00-6XX1 RS485 expansion module[20]	266 €
TOT	**844 €**

Expansion modules that can be added to the setup:

Siretta QUARTZ-COMPACT Router 4G[21]	329 €
Ixxat CM CANopen (compatible only with S7-1200 PLC)[22]	521 €

Table 6.1 and Table 6.2 show the comparison of these prices with those of the Trace&Follow board. A 10% discount was assumed for the commercial system with 20 boards. The last column shows the savings compared to the PLC-based system. Reported values represent the unitary cost across various configurations.

	Trace&Follow	PLC-based	saving
1 board with Ethernet connectivity	640 €	722 €	11.4%
20 boards with Ethernet connectivity	511 €	650 €	21.4%
1 board with 4G connectivity	820 €	1051 €	22%
20 boards with 4G connectivity	655 €	946 €	30.8%

Table 6.1: Prices comparison between T&F board and base PLC setup

	Trace&Follow	PLC-based	saving
1 board with Ethernet connectivity	640 €	844 €	24.2%
20 boards with Ethernet connectivity	511 €	760 €	32.8%
1 board with 4G connectivity	820 €	1694 €	42.1%
20 boards with 4G connectivity	655 €	1524 €	48.6%

Table 6.2: Prices comparison between T&F board and advanced PLC setup

As evidenced, the Trace&Follow board offers a significant price advantage. The ability to be integrated into a complete platform is another strength of this product. A customer purchasing this system does not have to develop anything, neither at the PLC program level nor at the data visualization application level. But it could also be appealing to system integrators who require a multifunctional power monitoring platform that can be embedded into their solutions.

6.2 Future improvements

The device developed during this project is not the final product, but rather a starting point for the development of a more complete platform. Its modular design allows for significant expansion, with the ability to collect data from additional devices once the RS485 expansion board is completed, resulting in a more flexible platform.

In addition to the hardware development, the software aspect is the most interesting. First, in this next step, the ESP32 will function as a web server, much like it did in the OTA update. Through a web page, the device will be configured without having to manually edit and upload the configuration file. The file will be automatically edited whenever any parameters are changed.

The next step in improving the Trace&Follow product will be to implement artificial intelligence algorithms on the cloud platform. Based on energy consumption curves, the platform will be able to automatically recognize the manufacturer's piece. By comparing these curves with a reference one, it will be able to tell if the produced piece meets the standards set by the company. This feature will be useful for both discarding potentially faulty pieces and analyzing whether the manufacturing process can be optimized.

Appendix A

LoRaWAN

When discussing LoRa and LoRaWAN networks, it is crucial to distinguish between the two. LoRa refers only to the lower physical layer, while LoRaWAN is a network protocols designed to build out all of the upper network layers. LoRa is a low-power, wide area network (LPWAN) RF modulation technology that enables long-range communication. Its range can reach up to 3 miles (5 kilometers) in urban areas and 10 miles (15 kilometers) or more in rural areas (line of sight). Main drawback is the low data rate achievable compared with other wireless technologies. Based on the region where it is deployed, it uses a different frequency band. In Europe for instance, the ISM (Industrial, Scientific and Medical) band is used. [23, 24]

Region	**Frequency (MHz)**
Europe	863-870, 433.05-434.7
Australia	915-928
Canada	902-928
China	470-510, 779-787
US	902-928

Table A.1: LoRa regional parameters [25]

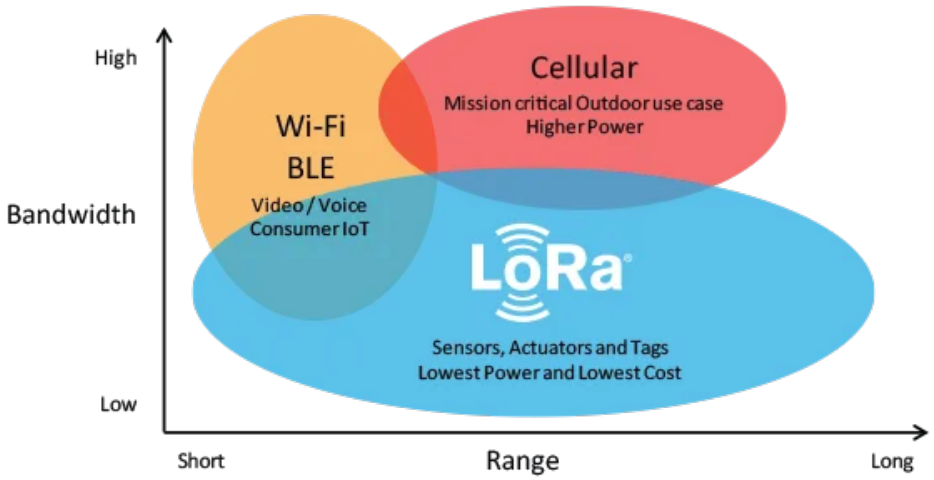

Figure A.1: Bandwidth vs range in wireless communications [26]

LoRaWAN is a Media Access Control (MAC) layer protocol built on top of LoRa modulation. It is a software layer which defines how devices use the LoRa hardware, for example when they transmit, and the format of messages [26] Messages from a device to the network are called uplink messages. Messages received by a device are called downlink messages.

The LoRaWAN specification defines three device classes, which define how the devices communicate with the network [27]:

- Class A: A class A device has the ability to transmit an uplink message at any given moment. Following the uplink transmission's completion, the device initiates two brief receive windows to receive downlink messages from the network. It the lowest power mode but has the highest latency.
- Class B: Class B devices extend Class A capabilities by opening receive windows, called ping slots, periodically, to receive downlink messages.
- Class C devices extend Class A capabilities by keeping the receive windows open unless transmitting an uplink.

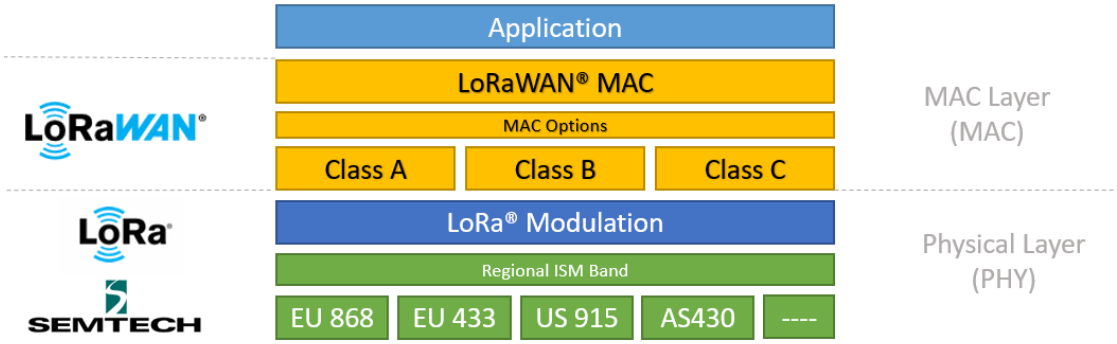

Figure A.2: LoRaWAN layers [28]

LoRaWAN specifies three security keys: NwkSKey, AppSKey, and AppKey. During network activation, both an application session key (AppSKey) and a network session key (NwkSKey) are generated. The network shares the NwkSKey, while the AppSKey is kept private. Only the device and the application have knowledge of the application key (AppKey). Dynamically activated devices (OTAA) utilize the Application Key (AppKey) to generate the two session keys in the activation process [29].

Appendix B

FreeRTOS

Developed in partnership with the world's leading chip companies, FreeRTOS is a market-leading real-time operating system (RTOS) for microcontrollers and small microprocessors. Distributed freely under the MIT open source license, FreeRTOS includes a kernel and a growing set of IoT libraries suitable for use across all industry sectors [30].

FreeRTOS is a layered architecture featuring multiple customizable modules that can be adapted to the needs of the system. This real-time operating system is meant to run on small embedded systems and can be ported to a variety of microcontrollers and processors. The architecture of FreeRTOS is based on a kernel that provides multitasking and real-time scheduling capabilities. The kernel manages task execution, which are the fundamental components of a FreeRTOS system. Each task runs as a dedicated thread of execution, independent of other tasks. The kernel provides the necessary mechanisms to create, delete, manage and synchronize the execution of tasks. FreeRTOS supports various inter-task communication mechanisms, including queues, semaphores, and mutexes. These

mechanisms enable tasks to communicate and synchronize their execution within a real-time system [31].

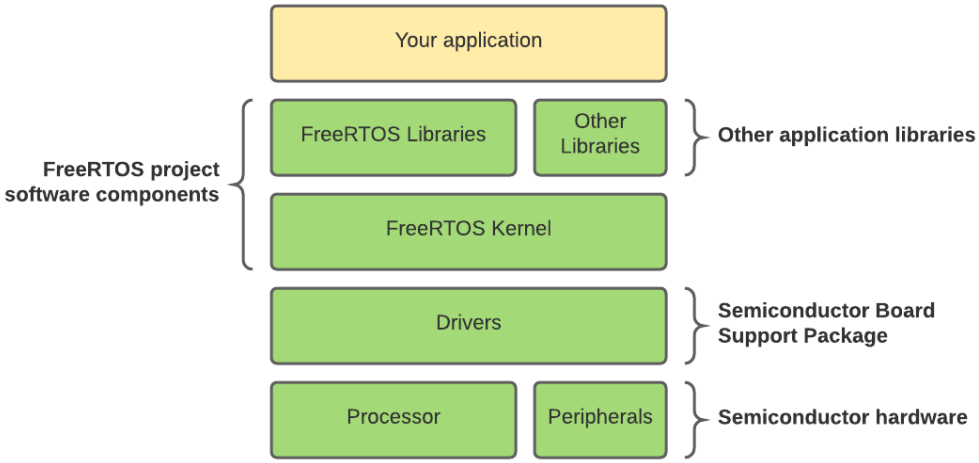

Figure B.1: FreeRTOS layers structure [32]

The original FreeRTOS is a small and efficient Real Time Operating System supported on many single-core MCUs and SoCs. However, to support numerous dual core ESP targets (such as the ESP32, ESP32-S3 and ESP32-P4), ESP-IDF provides a dual core SMP (Symmetric Multiprocessing) capable implementation of FreeRTOS. In general, an SMP system [33] is a computing architecture where two or more identical CPUs (cores) are connected to a single shared main memory and controlled by a single operating system. The main advantages of an SMP system compared to single core or Asymmetric Multiprocessing systems are that the presence of multiple CPUs allows for multiple hardware threads, thus increases overall processing throughput. Furthermore, having symmetric memory means that threads can switch cores during execution. This in general can lead to better CPU utilization.

Bibliography

[1] *ESP32 WROOM 32E datasheet*. 1.6. Espressif Systems. Jan. 2023 (cit. on pp. 13, 14).

[2] *ESP32-WROOM-32U-N4*. Accessed: 2023-09-3. URL: https://www.digikey.it/en/products/detail/espressif-systems/ESP32-WROOM-32U-N4/9381719 (cit. on p. 14).

[3] *ESP32-WROOM-32D*. Accessed: 2023-09-3. URL: https://www.soselectronic.com/it/products/espressif/esp32-wroom-32d-esp32-wroom-32d-n4-291230 (cit. on p. 14).

[4] *Wi-Fi Station*. Accessed: 2023-09-7. URL: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/ (cit. on p. 17).

[5] *Access Point*. Accessed: 2023-09-7. URL: https://randomnerdtutorials.com/esp32-useful-wi-fi-functions-arduino/ (cit. on p. 18).

[6] *Enhanced Poly-Phase High-Performance Wide-Span Energy Metering IC*. 2nd ed. Atmel Corporation. Dec. 2015 (cit. on p. 20).

[7] *Inserzione Aron*. Accessed: 2023-09-5. URL: https://it.wikipedia.org/wiki/Inserzione_Aron (cit. on pp. 23, 24).

[8] *ElegantOTA library Github repository*. URL: https://github.com/ayushsharma82/ElegantOTA (cit. on p. 32).

[9] *CRC8 Simple Algorithm for C*. Accessed: 2023-08-2. URL: https://devcoons.com/crc8/ (cit. on p. 52).

[10] *RF Coexistence*. Accessed: 2023-08-12. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/coexist.html#:~:text=ESP32%20has%20only%20one%202.4,to%20receive%20and%20transmit%20packets (cit. on p. 54).

[11] *ThingsBoard Open-source IoT Platform*. Accessed: 2023-09-8. URL: https://thingsboard.io/ (cit. on p. 72).

[12] *ThingsBoard Architecture*. Accessed: 2023-09-8. URL: https://thingsboard.io/docs/reference/architecture/ (cit. on p. 72).

[13] *ChirpStack, open-source LoRaWAN® Network Server*. Accessed: 2023-09-8. URL: https://www.chirpstack.io/ (cit. on p. 75).

[14] Bruno Mendes, Shani du Plessis, Dário Passos, and Noélia Correia. «Framework for the Integration of Transmission Optimization Components into LoRaWAN Stack». In: *Communication and Intelligent Systems*. Ed. by Harish Sharma, Vivek Shrivastava, Kusum Kumari Bharti, and Lipo Wang. Singapore: Springer Nature Singapore, 2022, pp. 421–431. ISBN: 978-981-19-2130-8 (cit. on p. 75).

[15] *PLC Siemens LOGO*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/plc/2097104 (cit. on p. 92).

[16] *RS PRO power meter*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/misuratori-di-energia/2369297 (cit. on p. 92).

[17] *Siemens LOGO! CIM*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/plc/2368553 (cit. on p. 92).

[18] *PLC Siemens SIMATIC S7-1200*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/plc/8624455?gb=s (cit. on p. 92).

[19] *Finder power meter 7M*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/misuratori-di-energia/2216573 (cit. on p. 92).

[20] *Siemens 6NH3112-3BA00-6XX1 RS485 expansion module*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/schede-di-espansione-e-adattatori/2553948?gb=s (cit. on p. 92).

[21] *Siretta QUARTZ-COMPACT Router 4G*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/router/1853070?gb=s (cit. on p. 92).

[22] *Ixxat CM CANopen*. Accessed: 2023-10-10. URL: https://it.rs-online.com/web/p/accessori-per-plc/2262730 (cit. on p. 92).

[23] *LoRa technology: what you need to know*. Accessed: 2023-09-15. URL: https://edalab.it/en/lora-iot/ (cit. on p. 95).

[24] *What is LoRaWAN® Specification*. Accessed: 2023-09-15. URL: https://lora-alliance.org/about-lorawan/ (cit. on p. 95).

[25] *LoRa*. Accessed: 2023-09-16. URL: https://lora.readthedocs.io/en/latest/ (cit. on p. 96).

[26] *What are LoRa and LoRaWAN?* Accessed: 2023-09-15. URL: https://www.thethingsnetwork.org/docs/lorawan/what-is-lorawan/ (cit. on p. 96).

[27] *Device Classes*. Accessed: 2023-09-16. URL: https://www.thethingsnetwork.org/docs/lorawan/classes/ (cit. on p. 96).

[28] *What are LoRa® and LoRaWAN®?* Accessed: 2023-09-15. URL: https://lora-developers.semtech.com/documentation/tech-papers-and-guides/lora-and-lorawan/ (cit. on p. 97).

[29] *Security*. Accessed: 2023-09-15. URL: https://www.thethingsnetwork.org/docs/lorawan/security/ (cit. on p. 97).

[30] *FreeRTOS™ Real-time operating system for microcontrollers*. Accessed: 2023-09-18. URL: https://www.freertos.org/index.html (cit. on p. 99).

[31] *FreeRTOS: Introduction to FreeRTOS*. Accessed: 2023-09-18. URL: https://piembsystech.com/free-rtos/ (cit. on p. 100).

[32] Richard Elberger. *Why SESIP™ Certification for FreeRTOS Matters*. Accessed: 2023-09-18. Mar. 20121. URL: https://www.freertos.org/2021/03/why-sesip-certification-for-freertos-matters.html (cit. on p. 100).

[33] *FreeRTOS (ESP-IDF)*. Accessed: 2023-09-18. URL: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos_idf.html (cit. on p. 100).