Plant Monitoring System

Embedded Platforms and Communications for IoT

Alejandro Botas Bárcena
Estela Mora Barba

MIoT 2025-2026

Contents

List of Figures

List of Tables

List of Listings

Acronyms

ADC Analog-to-Digital Converter

ARM Advanced RISC Machine

COM PC Communication

CPU Central Processing Unit

DAC Digital-to-Analog Converter

DMA Direct Memory Access

GND Ground

GPIO General Purpose Input-Output

GPS Global Positioning System

I2C Inter-Integrated Circuit

IoT Internet of Things

LED Light Emitting Diode

LoRa Long Range

LPWAN Low-Power Wide-Area Network

MCU Microcontroller Unit

NMEA National Marine Electronics Association

OS Operating System

PWM Pulse Width Modulation

RGB Red, Green and Blue

RH Relative Humidity

RTOS Real-Time Operating System

RX Reception

SCL Serial Clock Line

SDA Serial Data

SI International System of Units

SPI Serial Peripheral Interface

SRAM Static Random-Access Memory

TX Transmission

UART Universal Asynchronous Receiver-Transmitter

USB Universal Serial Bus

UTC Coordinated Universal Time

Todo list

1 Overview and Introduction

1.1 Document Overview

This document defines the technical specifications, development requirements, hardware architecture, software components, and operational modes of the Internet of Things (IoT)-based Plant Monitoring System to be implemented during the course *Embedded Platforms and Communications for IoT*. The objective of this specification is to establish a clear and comprehensive reference for the design, implementation, verification, and assessment of the final embedded system.

1.2 Project Introduction

The purpose of the final project is to design and implement a fully functional IoT platform capable of monitoring the environmental conditions and physiological state of a plant throughout its lifecycle. Such monitoring is essential for applications such as greenhouse automation, precision agriculture, plant health diagnostics, and traceability in plant transportation and storage.

The system must continuously acquire, process, and report multiple physical variables, including temperature, relative humidity, ambient light intensity, soil moisture, and colour characteristics of a plant leaf. Additional inertial data are collected through an accelerometer to detect events such as impacts, falls, tilting, or abnormal movements. The global position of the plant is obtained via a Global Positioning System (GPS) module, ensuring timestamped logging of all monitored parameters.

This project integrates both hardware and software development activities. Students must interface several digital and analog sensors, configure low-level peripherals (Inter-Integrated Circuit (I2C), Universal Asynchronous Receiver-Transmitter (UART), Analog-to-Digital Converter (ADC), General Purpose Input-Output (GPIO), Pulse Width Modulation (PWM)), and implement multitasking using Zephyr Operating System (OS).

The final embedded application must operate in distinct modes, manage periodic measurements, compute statistical parameters, handle event-based notifications, and provide visual feedback via a Red, Green and Blue (RGB) Light Emitting Diode (LED).

1.3 Summary of the Work Done

This section provides a consolidated overview of the work completed throughout the development of the IoT-based Plant Monitoring System. The tasks performed encompass the full engineering workflow, including requirement analysis, hardware integration, software design, system implementation, testing, and verification.

1.3.1 Requirements Analysis

The project began with an in-depth study of the provided technical specifications and sensor documentation. All functional, hardware, and timing requirements were reviewed to establish a clear design baseline. This included understanding the sensing ranges, communication interfaces, and operating constraints imposed by the STM32WL55JC microcontroller and Zephyr OS.

1.3.2 Hardware Integration

The hardware development stage consisted of identifying, wiring, and validating all sensor interfaces:

- STM32WL55JC microcontroller as the central processing unit.
- Integration of the Si7021 temperature and humidity sensor using the I2C bus.
- Connection and calibration of the HW5P-1 phototransistor for ambient light measurement.
- Analog acquisition and scaling of the SEN-13322 soil moisture probe.
- Digital configuration of the TCS34725 colour sensor over I2C.
- Setup of the MMA8451Q accelerometer for multi-axis measurements.
- Interfacing and configuring the Adafruit GPS module via UART.

- Implementation of a RGB LED driver using PWM emulation for status indication.

1.3.3 Software Development

Software implementation was carried out using Zephyr OS, structured with a multitasking architecture. Key activities included:

- Configuration of device tree overlays and config options for all peripherals.
- Development of sensor drivers and low-level routines for ADC, I2C, UART, and GPIO.
- Design of individual threads for periodic sampling, data processing, GPS acquisition, and mode management.
- Implementation of Test Mode, Normal Mode, and optional Advanced Mode according to system requirements.
- Integration of statistical processing to compute hourly mean, minimum, and maximum values.
- Implementation of colour-based alert mechanisms based on out-of-range sensor values.

Zephyr's logging and shell utilities were used extensively for debugging and validation.

1.3.4 System Testing and Validation

The complete system was evaluated across all operational modes:

- Verification of measurement accuracy and stability under Test Mode.
- Long-term monitoring and statistical computation under Normal Mode.
- Correct operation of the mode-switching mechanism through the push button.
- Validation of RGB LED behaviour for both colour detection and alert signalling.
- GPS time synchronization and conversion to local time for timestamp generation.
- Stress testing of the application to identify stack usage limits and race conditions.

All mandatory functionalities were confirmed to meet the specifications, with optional enhancements explored where possible.

1.3.5 Final Deliverables

The completed work includes:

- A functional embedded system integrating all sensors and the STM32WL55JC Microcontroller Unit (MCU).
- Clean, documented source code developed under Zephyr OS.
- A complete technical report detailing the system design, implementation, and results.
- Final project documentation and code submitted according to course requirements.

Overall, the work conducted demonstrates a full-cycle embedded systems development process, covering hardware, software, real-time processing, testing, and documentation.

2 Specifications

2.1 Specifications required

2.1.1 Hardware

The IoT system is based on the STM32WL55JC microcontroller. Several sensors are required to monitor plant-related environmental and physical parameters. Table 1 summarizes the recommended hardware components, electrical interfaces, and approximate costs.

Table 1: Summary of Suggested Hardware for the IoT System

Parameter	Sensor / Module	Interface
MCU Board	STM32WL55JC	–
Status Indicator	RGB LED	Digital
Ambient Light	HW5P-1 Phototransistor	Analog
Soil Moisture	SEN-13322	Analog
Temperature / Humidity	Si7021	I2C
Leaf Colour	TCS34725	I2C
Accelerometer	MMA8451Q	I2C
Global Location	Adafruit GPS	UART

2.1.2 Software

The software stack required for the development of the IoT system is summarized in Table 2.

Table 2: Software Requirements

Software Tool	Description
Zephyr OS	Real-Time Operating System (RTOS) for Advanced RISC Machine (ARM) Cortex-M devices, used for all system tasks and drivers.
Visual Studio Code	Primary development environment with C/C++, CMake, and Cortex-Debug extensions.
Git / TortoiseGit / GitHub	Version control for project source code.
TeraTerm / PuTTY	Serial terminal emulator for debugging and mode output.

2.1.3 System Requirements

The IoT system must measure environmental and physical parameters and operate in three modes: Test, Normal, and Advanced (optional). Table 3–Table 4 present the system requirements.

check resolutions

Table 3: Sensing Requirements

ID	Requirement	Specification
SR1	Temperature	Range: -10°C to 50°C. Resolution: 0.1°C
SR2	Relative Humidity	Range: 25%–75% RH. Resolution: 0.1%
SR3	Ambient Light	0–100%. Resolution: 0.1%
SR4	Soil Moisture	0–100%. Resolution: 0.1%
SR5	Leaf Colour	Clear, red, green, blue values
SR6	GPS Location	Coordinates + Coordinated Universal Time (UTC) time
SR7	Acceleration	X, Y, Z axes. Formatted output
GR1	Robustness	System must be stable and robust
GR2	Thread Management	Tasks must be partitioned using multitasking

Table 4: Operating Modes Requirements

Mode	ID	Description
Test Mode	TM1	Verify sensor connections and operation.
	TM2	Sampling period: 2 seconds.
	TM3	Send all measurements every 2 seconds via Universal Serial Bus (USB) virtual PC Communication (COM).
	TM4	RGB LED indicates dominant leaf colour.
	TM5	Blue LED (LED1) must remain ON.
Normal Mode	NM1	Sampling period: 30 seconds.
	NM2	Send all measurements every 30 seconds.
	NM3	Compute hourly mean, max, min for temperature, humidity, light, moisture.
	NM4	Compute hourly dominant colour (frequency-based).
	NM5	Compute hourly max and min accelerometer values.
	NM6	Send GPS location + local time every 30 seconds.
	NM7	Color-coded RGB alert when limits exceeded.
	NM8	Green LED (LED2) must remain ON.
Advanced Mode (Optional)	AM1	Requirements provided during validation stage.
	AM2	Red LED (LED3) must remain ON.

2.2 Additional specifications implemented

In addition to the mandatory requirements defined in the project specifications, several extended functionalities and robustness mechanisms were implemented to improve system reliability, diagnostic capability, maintainability, and user feedback. These additional specifications are summarized in Table 5.

Table 5: Additional Specifications Implemented

ID	Additional Requirement	Description
AS1	**Sensor configuration**	All sensors are configured during system initialization, including adjustable internal parameters such as the colour sensor gain and integration time.
AS2	**Fail-safe initialization**	If any sensor fails to initialize correctly, the entire application halts execution and enters a safe error state.
AS3	**Sensor reconnection**	If a sensor becomes disconnected, a fault condition is reported. If the sensor is later reconnected, the system automatically reinitializes it and resumes normal measurement without requiring a reset.
AS4	**Alarm sequencing**	When operating in NORMAL mode, all active alarms are cycled through sequentially using the RGB LED, with a display cadence of **0.5 seconds per alarm**.
AS5	**Advanced Mode behaviour**	In ADVANCED mode, the system extends the behaviour of the NORMAL mode by reproducing on the RGB LED the exact colour intensity measured by the TCS34725 sensor. Since the existing LED pins cannot be modified, PWM operation is emulated in software to match LED brightness with the RGB sensor readings proportionally. All remaining system behaviour matches NORMAL mode.
AS6	**Stack usage measurement**	The system includes automatic instrumentation to evaluate thread stack consumption at runtime.
AS7	**Verification tools**	External analysis tools were used to inspect the system.
AS8	**Documentation**	All modules, drivers, data-processing routines, and operating modes are fully documented.

3 Hardware Analysis

acronimos, vcc, vin?

3.1 Block diagram

The block diagram shown in Figure 1 provides an overview of the complete hardware architecture. It illustrates how the STM32WL55JC microcontroller interacts with the different sensors and output devices integrated into the system. Each peripheral is connected through the appropriate interface, such as analog inputs, I2C buses, UART communication lines, and GPIO pins, allowing the microcontroller to gather environmental data, process it, and generate feedback.

This diagram serves as a high-level representation of the system's structure, highlighting the flow of information between components and the role of the microcontroller as the central control unit.

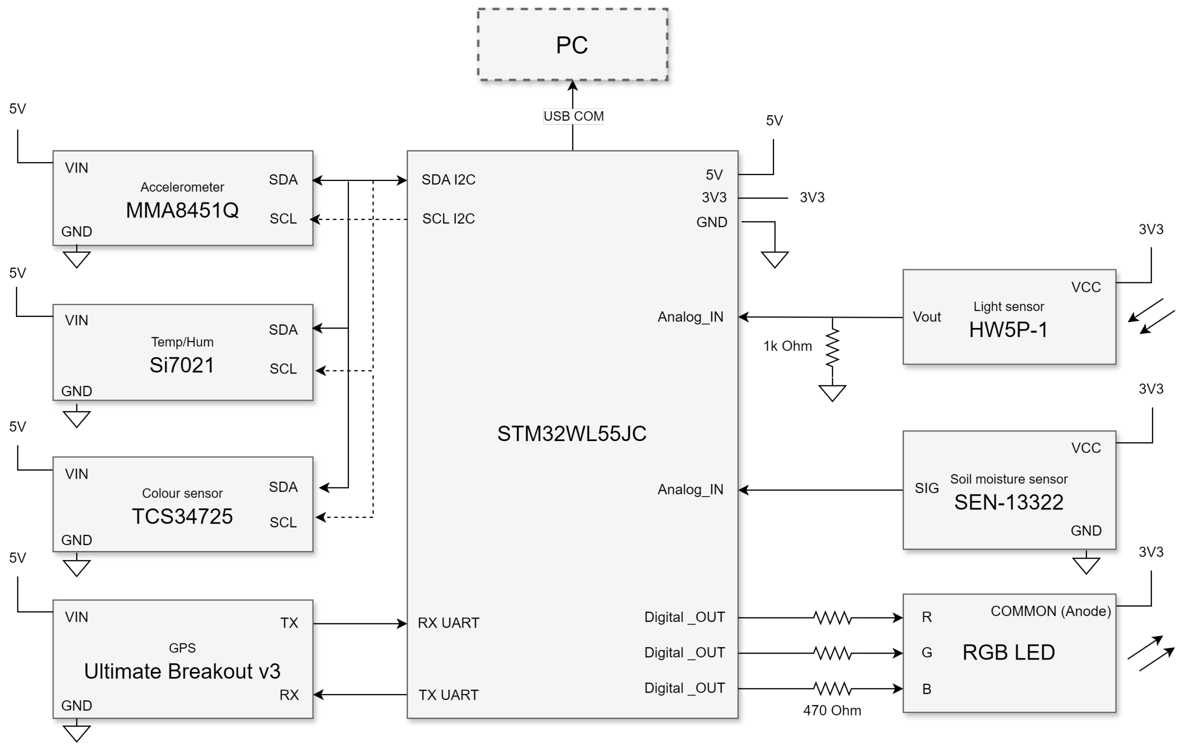


Figure 1: Block Diagram of the Hardware System

The microcontroller employs several of its internal peripherals to interface with the different sensors and modules in the system. One of the available ADC channels is used to read the analog outputs of the soil moisture sensor and the ambient light phototransistor. The I2C2 bus is shared by the temperature and humidity sensor (Si7021), the colour sensor (TCS34725), and the accelerometer (MMA8451Q). A UART interface is dedicated to the GPS module, enabling continuous reception of positioning data.

In addition, three GPIO pins are configured as digital outputs to drive the RGB LED through current-limiting resistors. The system also uses the 3.3V and 5V power rails provided by the board, as well as the ground reference shared by all components. Together, these resources form a compact and energy-efficient hardware configuration that leverages the STM32WL55JC's ADCs, GPIOs, communication peripherals, and power distribution capabilities.

3.2 Interfaces of the system

Table 6 details all electrical interfaces used in the system. Each sensor or module is mapped to the corresponding STM32WL55JC pins, specifying power connections, communication buses, and signal types. The design integrates a mix of digital and analog interfaces, including I2C for multi-sensor communication, UART for GPS data, and ADC channels for analog measurements such as soil moisture and ambient light. In addition, several GPIO pins are used for driving the RGB LED.

comparar con las diapos

Table 6: System Connections

Parameter	Sensor	Pin Description	Sensor PIN name	STM32WL55JC Connector	STM32WL55JC PIN name	STM32WL55JC Function
LED RGB	RGB LED + Resistors (470 Ohm)	Common (Anode)	C	CN6 3V3	3V3	3.3V
		Red + 470 Ohm	R	CN5 PA_6	D12	R
		Green + 470 Ohm	G	CN5 PA_7	D11	G
		Blue + 470 Ohm	B	CN5 PA_9	D9	B
Ambient Light	HW5P-1 Phototransistor + Resistor (1k Ohm)	VCC 3.3V	VCC	CN6 3V3	3V3	3.3V
		Vout	Vout	CN8 PB_1	ADC 1/5	Analog input
		Ground	Ground (GND)	CN6 GND	GND	Ground
Soil Moisture	SEN-13322	VCC 3.3V	VCC	CN6 3V3	3V3	3.3V
		Vout	SIG	CN8 PB_13	ADC 1/0	Analog input
		Ground	GND	CN6 GND	GND	Ground
Temperature and Humidity	Si7021	VCC 5V	VIN	CN6 5V	5V	5V
		Ground	GND	CN6 GND	GND	Ground
		I2C Serial Clock Line (SCL)	SCL	CN5 PA_12	I2C2_SCL	I2C SCL
		I2C Serial Data (SDA)	SDA	CN5 PA_11	I2C2_SDA	I2C SDA
		Output 3.3V	3Vo	–	–	–
Leaf Colour	TCS34725	VIN 5V	VIN	CN6 5V	5V	5V
		Ground	GND	CN6 GND	GND	Ground
		I2C SCL	SCL	CN5 PA_12	I2C2_SCL	I2C SCL
		I2C SDA	SDA	CN5 PA_11	I2C2_SDA	I2C SDA
		Output 3.3V	3V3	–	–	–
		Interrupt out	INT	–	–	–
		LED on/off	LED	–	–	–
Accelerometer	MMA8451Q	VIN 5V	VIN	CN6 5V	5V	5V
		Ground	GND	CN6 GND	GND	Ground
		I2C SCL	SCL	CN5 PA_12	I2C2_SCL	I2C SCL
		I2C SDA	SDA	CN5 PA_11	I2C2_SDA	I2C SDA
		Output 3.3V reg	3Vo	–	–	–
		Inertial Interrupt 1, Output pin	I1	–	–	–
		Inertial Interrupt 2, Output pin	I2	–	–	–
		I2C least significant bit of the device I2C address	A	–	–	–
GPS	Adafruit Ultimate GPS Breakout v3	VIN 5V	VIN	CN6 5V	5V	5V
		Ground	GND	CN6 GND	GND	Ground
		Serial Transmission (TX)	TX	CN9 PB_7	UART1_Reception (RX)	UART RX
		Serial RX	RX	CN9 PB_6	UART1_TX	UART TX
		Output 3.3V reg	3.3V	–	–	–
		Enable	EN	–	–	–
		Fix output	FIX	–	–	–
		Vbackup (battery)	VBAT	–	–	–
		Pulse Per Second output	PPS	–	–	–

3.3 Communication Interfaces used in the system

The system relies on several hardware communication interfaces that allow the STM32WL55JC microcontroller to exchange data efficiently with the different sensors and modules. Each interface is selected based on the nature of the signal (analog or digital), the required data rate, and the number of devices connected.

3.3.1 Analog-to-Digital Converter (ADC)

The STM32WL55JC includes a 12-bit ADC capable of converting analog voltages into digital values. This interface is used for sensors that provide an output voltage proportional to a physical quantity, such as:

- HW5P-1 phototransistor (ambient light)
- SEN-13322 soil moisture sensor

The ADC samples the voltage at the input pin and converts it into a numerical value between 0 and 4095 (12 bits), enabling the microcontroller to process continuous physical signals using digital logic.

3.3.2 I2C Bus

The I2C (Inter-Integrated Circuit) bus is a two-wire digital communication interface consisting of:

- SCL: clock line
- SDA: data line

Multiple sensors can share the same bus because each device has a unique address. In this system, I2C2 is used, and three devices share it:

- Si7021 temperature and humidity sensor
- TCS34725 colour sensor
- MMA8451Q accelerometer

I2C allows simple wiring, energy-efficient transmission, and reliable short-distance communication, making it ideal for embedded sensor networks.

3.3.3 UART Interface

The Universal Asynchronous Receiver/Transmitter (UART) is a serial communication interface used for asynchronous data transfer. It uses two lines:

- TX: microcontroller transmits data
- RX: microcontroller receives data

The Adafruit Ultimate GPS Breakout v3 communicates via a dedicated UART port, continuously streaming National Marine Electronics Association (NMEA) sentences that include position, altitude, speed, and time. UART is preferred here because it supports continuous high-latency streams and long-format messages without requiring a synchronized clock signal.

3.3.4 GPIO Digital Pins

General-Purpose Input/Output (GPIO) pins are used for simple digital control or sensing. In this project, several GPIOs are configured as outputs to drive the RGB LED. Each colour channel (red, green, and blue) is controlled by switching the corresponding GPIO pin on or off.

GPIOs allow:

- Driving LEDs or actuators
- Reading simple digital sensors
- Triggering interrupts

Their flexibility and direct control make them suitable for simple digital signals.

3.3.5 Power Interfaces

The system also uses fixed-voltage power rails:

- **3.3V**: used by analog sensors and logic inputs (e.g., phototransistor, soil sensor)
- **5V**: used by some breakout boards that include internal regulators (e.g., Si7021, TCS34725, MMA8451Q, GPS)
- **GND**: common electrical reference shared by all modules

A shared ground is essential for stable communication because all signal voltages must be referenced to the same electrical level. These interfaces together form an efficient and compact architecture that ensures reliable data acquisition and control across all hardware modules.

3.4 Hardware devices

3.4.1 STM32WL55JC microcontroller

The STM32WL55JC[1] is an ultra-low-power microcontroller that integrates both a processing unit and a long-range sub-GHz radio in a single chip. It combines an ARM Cortex-M4 core for the main application and an ARM Cortex-M0+ core for security and background tasks, providing efficient performance with very low energy consumption. The device includes 256KB of Flash, 64KB of Static Random-Access Memory (SRAM), and a wide set of protection features to ensure firmware integrity.

Its built-in radio supports several Low-Power Wide-Area Network (LPWAN) modulations, including Long Range (LoRa), enabling long-distance communication. The microcontroller also offers a rich collection of peripherals—such as 12-bit ADC/Digital-to-Analog Converter (DAC), multiple timers, Direct Memory Access (DMA) controllers, and interfaces like UART, I2C, and Serial Peripheral Interface (SPI), making it highly adaptable to sensor-based and low-power embedded applications.

3.4.2 RGB LED and 470 Ohm resistors

The system includes a common-anode RGB LED used to provide visual feedback during operation. This type of LED shares a single positive terminal connected to the 3.3,V rail, while each color channel (red, green, and blue) is controlled individually through the microcontroller. The STM32WL55JC drives the three channels using pins PA_6, PA_7, and PA_9, which can be toggled to generate different brightness levels and color combinations.

Each LED channel is connected in series with a 470 Ohm resistor to ensure proper current limiting and protect both the LED and the microcontroller outputs. This simple circuit allows the system to display a wide range of colors, enabling intuitive status indication, such as alerts or measurement feedback.

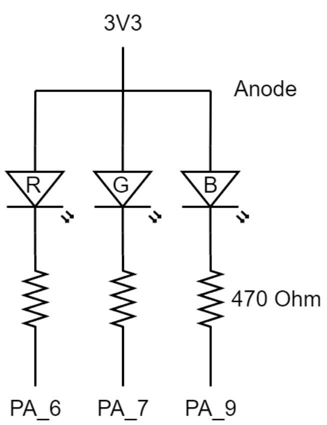


Figure 2: RGB LED circuit diagram

3.4.3 HW5P-1 Phototransistor and 1k Ohm resistor

The system uses a HW5P-1 phototransistor [2] to measure ambient light intensity. The phototransistor is connected in a simple voltage-divider configuration with a 1k Ohm resistor, converting the light-dependent current into a measurable voltage at the junction between the two components. This analog voltage is fed directly into the STM32WL55JC's ADC1/5 channel, allowing the microcontroller to quantify the light level.

By sampling the ADC input, the system can monitor changes in illumination and use this information for environmental sensing or automatic control tasks. The 3.3V supply powers the phototransistor, while a shared ground ensures proper reference for the ADC measurements.

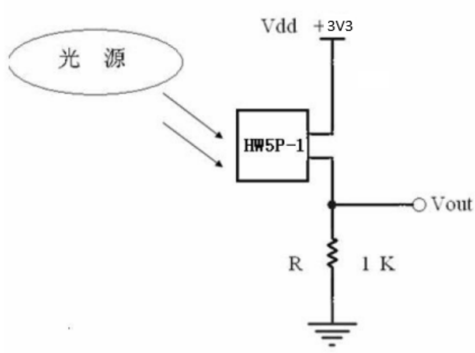


Figure 3: Phototransistor circuit diagram

To express the ambient light as a percentage, the ADC reading is first converted to a voltage using the reference voltage V_{ref} of 3.3V: comprobar formulas

$$V_{\text{phototransistor}} = \frac{\text{ADC\_value}}{\text{ADC\_max}} \cdot V_{\text{ref}}$$

Then, this voltage is normalized to a percentage of the maximum measurable light:

$$\text{Light\%} = \frac{V_{\text{phototransistor}}}{V_{\text{ref}}} \cdot 100 = \frac{\text{ADC\_value}}{\text{ADC\_max}} \cdot 100$$

This approach ensures that the ambient light intensity is represented in a standardized form from 0% (dark) to 100% (maximum brightness measurable by the sensor).

The measured light percentage is used by the system to evaluate illumination conditions in the surrounding environment.

3.4.4 SEN-13322 Soil Moisture Sensor

The system uses a SEN-13322 soil moisture sensor[3] to monitor the water content of the soil. This sensor outputs an analog voltage that varies proportionally with the soil's moisture level. The voltage is measured by the STM32WL55JC using ADC1/0 channel, allowing the microcontroller to quantify the moisture.

The sensor is powered by the 3.3V supply from the board, with a common ground shared with the microcontroller to ensure accurate ADC measurements.

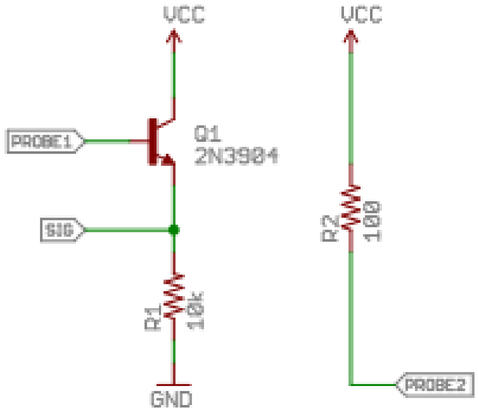


Figure 4: Soil moisture sensor circuit diagram

A direct reading of the ADC value can be converted into a soil moisture percentage using a similar normalization method as for the phototransistor: comprobar formulas

$$\text{Moisture\%} = \frac{\text{ADC\_value}}{\text{ADC\_max}} \times 100$$

This provides a simple and effective way to represent soil moisture from 0% (completely dry) to 100% (fully saturated).

By continuously monitoring this value, the system can perform environmental sensing, trigger alerts, or control irrigation mechanisms in an automated manner.

3.4.5 Si7021 Temperature and Humidity Sensor

The system integrates a Si7021 digital temperature and humidity sensor[4] to monitor environmental conditions. This sensor communicates with the STM32WL55JC microcontroller via the I2C2 bus, using pins PA_12 (SCL) and PA_11 (SDA). The sensor is powered by the 5V supply from the board, while a shared ground ensures reliable communication and stable operation.

The Si7021 provides fully digital readings for both temperature and relative humidity, eliminating the need for additional signal conditioning or ADC conversion.

Relative Humidity (RH) is the amount of water vapor present in the air expressed as a percentage of the maximum humidity the air can hold at a given temperature. Mathematically, it is expressed as:

$$\text{RH (\%)} = \frac{P_{\text{humidity}}}{P_{\text{max\_humidity}}(T)} \times 100$$

The microcontroller can query the sensor at regular intervals to obtain accurate temperature in degrees Celsius and relative humidity in percentage. These measurements can then be used for environmental monitoring, data logging, or as input for control algorithms, such as adjusting irrigation based on humidity levels.

3.4.6 TCS34725 Colour Sensor

The system uses a TCS34725 digital colour sensor[5] to measure the color and brightness of objects or ambient light. This sensor communicates with the STM32WL55JC microcontroller via the I2C2 bus, using pins PA_12 (SCL) and PA_11 (SDA). The sensor is powered by the 5V supply from the board, with a shared ground for proper signal reference.

The TCS34725 integrates an array of photodiodes with color-specific filters (red, green, and blue) to detect the intensity of each primary color. Additionally, it includes a clear photodiode that measures the total light intensity without any color filtering. This *clear* channel allows the microcontroller to compensate for variations in ambient light and normalize the color measurements, improving accuracy under different lighting conditions.

The sensor provides digital output values for each channel (red, green, blue, and clear), which can be read directly by the microcontroller. To calculate the relative intensity of each color, the readings can be normalized against the clear channel:

$$\text{Color\_ratio} = \frac{\text{Color\_value}}{\text{Clear\_value}}$$

This ratio provides a normalized measurement of the color composition independent of the overall light intensity.

The microcontroller can use this information for environmental monitoring, assessing leaf color for plant health, or other applications requiring color detection.

3.4.7 MMA8451Q Accelerometer

The system includes an MMA8451Q 3-axis digital accelerometer[6] to measure linear acceleration along three orthogonal axes (X, Y, and Z). The sensor communicates with the STM32WL55JC microcontroller via the I2C2 bus using pins PA_12 (SCL) and PA_11 (SDA), and is powered by the 5V supply from the board with a common ground reference.

The MMA8451Q outputs digital values corresponding to the acceleration experienced along each axis, which includes static acceleration due to gravity. By convention, the Z-axis measures the acceleration in the vertical direction, while X and Y correspond to horizontal directions.

The raw digital readings from the sensor can be converted into acceleration in units of gravitational acceleration (g), using the sensor's sensitivity parameter (S_{range}), which depends on the configured full-scale range (e.g., ±2g, ±4g, or ±8g):

revisar rangos del acelerometro

$$a_{\text{axis}}\,[g] = \frac{\text{Raw\_value}}{2^{12-1}} \cdot S_{\text{range}}$$

Raw_value is the 12-bit signed output from the accelerometer, and $2^{12-1} = 2048$ accounts for the 12-bit resolution with signed values. To convert the acceleration into International System of Units (SI) units (m/s^2), the following relation is used:

$$a_{\text{axis}}\,[\text{m/s}^2] = a_{\text{axis}}\,[g] \cdot g_0$$

where $g_0 = 9.80665\ \text{m/s}^2$ is the standard acceleration due to gravity. This conversion allows the microcontroller to quantify acceleration in physical units, providing meaningful data for motion detection, tilt sensing, or vibration monitoring.

By continuously reading the accelerometer, the system can track orientation changes, detect movement events, and combine the data with other sensors for environmental and behavioral monitoring applications.

3.4.8 Adafruit Ultimate GPS Breakout v3

The system integrates an Adafruit Ultimate GPS Breakout v3 module[7] to obtain accurate geolocation and time information. The GPS communicates with the STM32WL55JC microcontroller via a dedicated UART interface, using pins PB_6 (TX) and PB_7 (RX). The module is powered by the 5V supply from the board, with a shared ground reference.

The GPS module outputs position and time data in the standard NMEA sentence format. Key information includes:

- **Latitude and Longitude:** Provided in degrees and minutes (DDMM.MMMM for latitude and DDDMM.MMMM for longitude) along with a directional indicator: 'N' or 'S' for latitude, and 'E' or 'W' for longitude.

Latitude is referenced to the **Equator** (0°), which divides the Earth into the Northern and Southern hemispheres. Thus, 'N' indicates a position north of the Equator, while 'S' indicates a position south of it.

Longitude is referenced to the **Prime Meridian** (0°), also known as the **Greenwich Meridian**, which separates the Eastern and Western hemispheres. Positions east of Greenwich are marked with 'E', and those to the west with 'W'.

These values can be converted to decimal degrees using:

$$\text{Decimal Degrees} = \text{Degrees} + \frac{\text{Minutes}}{60}$$

For coordinates in the Southern or Western hemispheres, the decimal degrees are taken as negative.

Decimal degrees are used because they provide a **continuous numerical representation** of geographic coordinates, which simplifies mathematical operations such as distance calculations, interpolation, mapping transformations, and data storage. This format is easier for microcontrollers and software libraries to process compared to the degrees-minutes format used in raw NMEA sentences.

- **Altitude:** Measured in meters above mean sea level.
- **UTC Time and Date:** Provided as Coordinated Universal Time (hhmmss.ss for time and ddmmyy for date). UTC is referenced to the **Prime Meridian in Greenwich**, meaning it is the global baseline from which all time zones are defined.

  The system is located in Spain (peninsular), where local time is typically:

$$\text{Local Time} = \text{UTC} + 1\,\text{hour}$$

By parsing the NMEA sentences, the microcontroller can extract and store accurate position coordinates, altitude, and UTC-based timestamps. Continuous reception ensures that the system always has up-to-date location and timing information for real-time applications, enabling georeferenced sensor measurements and time-stamped environmental monitoring.

4 Software Organization

4.1 Description of the Implementation

4.2 Modules

4.2.1

4.3 Threads

4.3.1 Main Thread

4.3.2 Sensors Thread

4.3.3 GPS Thread

4.4 Global Behaviour

4.5 Thread Stack and CPU Usage Analysis

Zephyr provides runtime diagnostics that allow monitoring of the stack usage and CPU load of each thread in the system. The output shown in the image presents detailed information for all active threads, including their stack consumption, remaining free stack space, and the total number of CPU cycles executed since startup.

```
Thread analyze:
 gps_thread          : STACK: unused 768 usage 256 / 1024 (25 %); CPU: 0 %
                     : Total CPU cycles used: 56978
 sensors_thread      : STACK: unused 592 usage 432 / 1024 (42 %); CPU: 0 %
                     : Total CPU cycles used: 698566
 thread_analyzer     : STACK: unused 560 usage 464 / 1024 (45 %); CPU: 0 %
                     : Total CPU cycles used: 119419
 sysworkq            : STACK: unused 808 usage 216 / 1024 (21 %); CPU: 0 %
                     : Total CPU cycles used: 1012
 logging             : STACK: unused 164 usage 604 / 768 (78 %); CPU: 1 %
                     : Total CPU cycles used: 48822096
 idle                : STACK: unused 192 usage 64 / 256 (25 %); CPU: 98 %
                     : Total CPU cycles used: 2830266161
 main                : STACK: unused 544 usage 480 / 1024 (46 %); CPU: 0 %
                     : Total CPU cycles used: 834931
 ISR0                : STACK: unused 1672 usage 376 / 2048 (18 %)
```

Figure 5: Thread stack and CPU usage

It is important to understand the purpose of each thread displayed:

- **gps_thread:** Thread responsible for configuring, reading and parsing GPS data.
- **sensors_thread:** Thread responsible for sensor readings (accelerometer, colour sensor, etc.).
- **thread_analyzer:** Internal diagnostic thread used to collect and report thread metrics such as stack usage. It runs periodically and only consumes CPU during short analysis windows.
- **sysworkq:** The global Zephyr system workqueue. It is used to run small background tasks that do not need their own dedicated thread. Typical examples include executing callbacks.
- **logging:** Internal Zephyr thread in charge of processing log messages.
- **idle:** Lowest-priority thread that runs whenever no other thread is ready. It accounts for the majority of CPU cycles, which is expected and desirable in a low-power sensor system.
- **main:** The initial thread created at system startup.
- **ISR0 (ISR stack):** Not a regular thread, but the shared stack region used by all interrupt service routines.

For each thread, the following metrics are displayed:

- **STACK:** Reports the unused stack space, the amount of stack used, and the total allocated stack size. For example, for `gps_thread`:

  unused 768 B, used 256 B, total 1024 B

  This corresponds to a stack usage of 25%, indicating that the assigned memory is sufficient and no overflow risk is present.

  As a general guideline, **stack usage below 60% is considered safe** in Zephyr, as it leaves enough headroom for context switching, interrupts, and occasional peak loads.

- **CPU:** Shows the percentage of CPU time consumed by each thread. Most application threads such as `gps_thread` or `sensors_thread` show 0% CPU usage because they predominantly sleep while waiting for periodic timers or I/O events.

- **Total CPU cycles used:** Indicates the cumulative processor cycles consumed by each thread since boot.

  Threads like `idle` present extremely large values, which is expected since the idle thread runs whenever no other thread is ready to execute. A high idle count is a positive indicator of energy efficiency.

The stack usage analysis helps ensure that no thread is close to exhausting its allocated stack memory.

This is obtained thanks to the following configuration options enabled in `prj_nucleo_wl55jc.conf`, which allow Zephyr to track stack usage, assign human-readable thread names, and automatically generate periodic thread analysis reports:

Listing 1: Thread stack and CPU usage report - prj_nucleo_wl55jc.conf

```
CONFIG_INIT_STACKS=y
CONFIG_THREAD_STACK_INFO=y
CONFIG_THREAD_ANALYZER=y
CONFIG_THREAD_ANALYZER_AUTO=y
CONFIG_THREAD_NAME=y
```

Overall, the reported values confirm that:

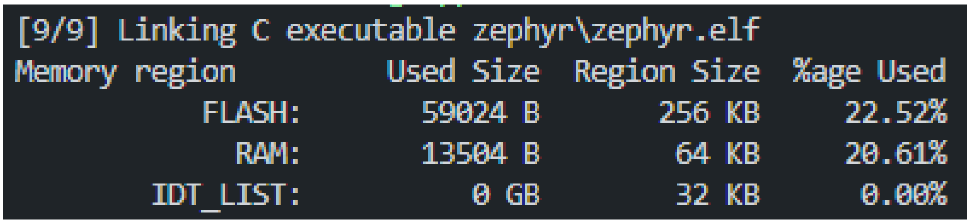
- All thread stacks remain within safe usage ranges, with most below the recommended 60% threshold.
- CPU usage distribution behaves as expected for a sensor-driven, event-based embedded application.
- The idle thread dominates CPU cycles, indicating efficient low-power execution and minimal background processing overhead.

4.6 Compilation and Flashing Output Analysis

During the compilation process, Zephyr generates a memory usage summary that indicates how much Flash and RAM the final application occupies. As shown in Figure 6, after linking the executable `zephyr.elf`, the memory report provides the following information:

- **FLASH:** 59.024B used out of 256KB (approximately 22.5%).
- **RAM:** 13.504B used out of 64KB (approximately 20.6%).

This confirms that the firmware comfortably fits within the memory limits of the STM32WL55 microcontroller, leaving sufficient headroom for future improvements or additional functionality.



```
[9/9] Linking C executable zephyr\zephyr.elf
Memory region         Used Size  Region Size  %age Used
           FLASH:       59024 B       256 KB     22.52%
             RAM:       13504 B        64 KB     20.61%
        IDT_LIST:          0 GB        32 KB      0.00%
```

Figure 6: Compilation memory usage report

4.7 Flashing the Firmware onto the STM32WL55

The Figure 7 corresponds to the flashing process performed using `STM32CubeProgrammer`, which communicates with the NUCLEO-WL55JC board via the onboard ST-LINK debugger. The tool successfully identifies the target device, displaying key details such as:

- **Device:** STM32WLxx.
- **Flash Size:** 256 KB.
- **Core:** ARM Cortex-M4.
- **Supply Voltage:** 3.28 V.
- **Connection Mode:** Under Reset.

After loading the generated `zephyr.hex` file (57.64 KB), the programmer performs the following steps:

1. Erases the internal Flash sectors (0 to 28).
2. Programs the firmware at address `0x08000000`.
3. Verifies the integrity of the written data.
4. Starts the application.

The final message, *"Application is running, Please Hold on..."*, indicates that the microcontroller has successfully been programmed and is now executing the uploaded Zephyr firmware.

```
-------------------------------------------------------------------
                  STM32CubeProgrammer v2.20.0
-------------------------------------------------------------------

ST-LINK SN  : 003E00124741500120383733
ST-LINK FW  : V3J7M3
Board       : NUCLEO-WL55JC
Voltage     : 3.28V
SWD freq    : 8000 KHz
Connect mode: Under Reset
Reset mode  : Hardware reset
Device ID   : 0x497
Revision ID : Rev Y
Device name : STM32WLxx
Flash size  : 256 KBytes
Device type : MCU
Device CPU  : Cortex-M4
BL Version  : 0xC4



Opening and parsing file: zephyr.hex


Memory Programming ...
  File          : zephyr.hex
  Size          : 57.64 KB
  Address       : 0x08000000


Erasing memory corresponding to segment 0:
Erasing internal memory sectors [0 28]
Download in Progress:
██████████████████████████████████████████████████ 100%

File download complete
Time elapsed during download operation: 00:00:01.654

RUNNING Program ...
  Address:      : 0x8000000
Application is running, Please Hold on...
Start operation achieved successfully
```

Figure 7: Flashing process using STM32CubeProgrammer

4.8 Code Documentation

The project documentation is generated automatically through a continuous integration workflow implemented using a GitHub Action (subsection A.3). This workflow executes the Doxygen engine, which extracts structured information directly from the annotated comments within the source code. By following Doxygen's documentation conventions, each module, function, and data structure is described where it is implemented, ensuring that the documentation remains consistent with the evolving codebase.

Whenever new commits are pushed to the repository, the GitHub Action is triggered, automatically regenerating the documentation and preventing discrepancies between the implementation and its technical description. As part of the same workflow, the generated documentation is automatically deployed to a GitHub Pages site, making it accessible online without requiring manual intervention.

The documentation can be accessed directly through the following link: `https://estelamb.github.io/Embedded_IoT/`.

5 Results

Test implemented to demonstrate the specifications Code (github, suggested), highlight your contributions vs external code Screen captures. Links to videos showing the terminal output.

6 Advanced Specifications Implemented

6.1 Problem Statement

The objective of the *ADVANCED* mode is to extend the behaviour of the baseline system so that the RGB LED reproduces, the colour measured by the RGB sensor. Unlike the *NORMAL* mode, where the LED output follows predefined patterns, the *ADVANCED* mode must directly map the sensor's RGB data to the LED output using the same relative chromatic composition.

Since the LED pin assignment cannot be modified, the implementation must simulate PWM in software in order to control the LED brightness according to the sensor readings. All other functional parameters must remain identical to those of the *NORMAL* mode.

6.2 Implementation

To reproduce the detected colour, the raw RGB values obtained from the sensor are normalised with respect to the clear channel. This ensures that the emitted light preserves the chromatic ratios of the measured colour independently of the absolute illumination level.

A software-based PWM mechanism is implemented to generate duty cycles proportional to the normalised RGB components. The PWM period is discretised into fixed-size steps, and during each step the LED channels are switched on or off based on whether the current time index is below the computed duty threshold. This emulates real PWM behaviour while keeping the original hardware configuration unchanged.

The following code fragment corresponds to the execution flow in *ADVANCED* mode, after acquiring and displaying the colour measurements:

Listing 2: Software-based PWM implementation for ADVANCED mode

```c
#define PWM_STEP  1                      /**< PWM step in milliseconds. */
#define PWM_PERIOD 15                    /**< PWM period in milliseconds. */
#define PWM_STEPS (PWM_PERIOD / PWM_STEP) /**< Number of PWM steps per period. */

/* In ADVANCED mode */
if (main_data.c <= 0.0f) {
    printk("[WARN] - Color clear channel == 0\n");
    r_norm = g_norm = b_norm = 0.0f;
} else {
    r_norm = (main_data.r / main_data.c) * 100.0f;
    g_norm = (main_data.g / main_data.c) * 100.0f;
    b_norm = (main_data.b / main_data.c) * 100.0f;
}

printk("NORMALIZED COLOR VALUES: R: %.2f%%, G: %.2f%%, B: %.2f%%\n\n",
       (double)r_norm, (double)g_norm, (double)b_norm);

r_duty = (int)(r_norm * PWM_STEPS / 100);
g_duty = (int)(g_norm * PWM_STEPS / 100);
b_duty = (int)(b_norm * PWM_STEPS / 100);

keep_running = true;
while (keep_running) {
    for (int t = 0; t < PWM_PERIOD; t += PWM_STEP) {
        r_value = (t < r_duty) ? 1 : 0;
        g_value = (t < g_duty) ? 1 : 0;
        b_value = (t < b_duty) ? 1 : 0;

        rgb_led_pwm_step(&rgb_leds, r_value, g_value, b_value);
        k_sleep(K_MSEC(PWM_STEP));

        if (k_sem_take(&main_sem, K_NO_WAIT) == 0) {
            keep_running = false;
            break;
```

```
        }
    }
}
```

During development, several timing constraints were encountered when attempting to reduce the PWM step below the millisecond scale. The Zephyr scheduler, together with the hardware capabilities of the target board, prevented reliable delays shorter than one millisecond when using `k_sleep()`.

Alternative approaches were evaluated, including the use of `k_busy_wait()` to achieve microsecond-level blocking delays. However, the board exhibited unstable behaviour and noticeable performance degradation when executing busy-wait loops within the real-time colour reproduction cycle. Consequently, the software PWM design was constrained to millisecond-resolution timing.

6.2.1 Macro definitions and timing parameters

Listing 3: Macros for *ADVANCED* mode

```
#define PWM_STEP  1                        /**< PWM step in milliseconds. */
#define PWM_PERIOD 15                      /**< PWM period in milliseconds. */
#define PWM_STEPS (PWM_PERIOD / PWM_STEP) /**< Number of PWM steps per period. */
```

These macros define the temporal discretisation used by the software PWM:

- `PWM_STEP` specifies the length of a single PWM step in milliseconds. Each step is one atomic time-slot in which the LED channels are either ON or OFF.
- `PWM_PERIOD` is the total duration of one PWM cycle in milliseconds. The perceived brightness is determined by the fraction of this period during which a channel is ON.
- `PWM_STEPS` is the number of discrete steps per PWM period (computed as `PWM_PERIOD / PWM_STEP`). It is used to convert percentage-based normalised colour values into integer duty counts.

This configuration intentionally uses millisecond resolution due to the underlying RTOS scheduler and board timing limitations discussed previously.

6.2.2 Sensor clear-channel safety check and normalisation

Listing 4: Color normalisation based on clear channel

```
if (main_data.c <= 0.0f) {
    printk("[WARN] - Color clear channel == 0\n");
    r_norm = g_norm = b_norm = 0.0f;
} else {
    r_norm = (main_data.r / main_data.c) * 100.0f;
    g_norm = (main_data.g / main_data.c) * 100.0f;
    b_norm = (main_data.b / main_data.c) * 100.0f;
}
```

Explanation:

- `main_data.c` denotes the clear (ambient) channel returned by the colour sensor. A zero or negative value indicates an invalid or saturated measurement.
- The **safety check** prevents division by zero by forcing the normalised channels to 0% and emitting a warning via `printk` if the clear channel is non-positive.
- When valid, each raw channel (`r`, `g`, `b`) is normalised by the clear channel and converted to percentage units by multiplying by 100. This preserves the chromatic ratios while removing absolute intensity dependence.

6.2.3 Diagnostic logging

Listing 5: Diagnostic logging of normalised color values

```
printk("NORMALIZED COLOR VALUES: R: %.2f%%, G: %.2f%%, B: %.2f%%\n\n",
       (double)r_norm, (double)g_norm, (double)b_norm);
```

Explanation:

- This log statement prints the computed normalised percentages for each channel. It serves as a diagnostic aid during development and in-field debugging to verify that sensor readings and normalisation behave as expected.
- Casting to `double` is used to satisfy the `printk` format specifier and to ensure consistent formatting across platforms.

6.2.4 Duty-cycle computation

Listing 6: Computation of duty counts from normalised percentages

```
r_duty = (int)(r_norm * PWM_STEPS / 100);
g_duty = (int)(g_norm * PWM_STEPS / 100);
b_duty = (int)(b_norm * PWM_STEPS / 100);
```

Explanation:

- The normalised percentages are converted into integer *duty counts* that range from 0 to `PWM_STEPS`.
- Example: with `PWM_STEPS = 15`, a normalised red value of 50% results in `r_duty = 7` (approximately half of the period steps ON).
- Integer truncation is acceptable here because the PWM resolution is limited by `PWM_STEPS`; if higher fidelity is required, increase `PWM_STEPS` by reducing `PWM_STEP`, subject to the timing limits of the platform.

6.2.5 Main PWM loop and per-step evaluation

Listing 7: Main software PWM loop

```
keep_running = true;
while (keep_running) {
    for (int t = 0; t < PWM_PERIOD; t += PWM_STEP) {
        r_value = (t < r_duty) ? 1 : 0;
        g_value = (t < g_duty) ? 1 : 0;
        b_value = (t < b_duty) ? 1 : 0;

        rgb_led_pwm_step(&rgb_leds, r_value, g_value, b_value);
        k_sleep(K_MSEC(PWM_STEP));

        if (k_sem_take(&main_sem, K_NO_WAIT) == 0) {
            keep_running = false;
            break;
        }
    }
}
```

Step-by-step explanation:

1. `keep_running` controls the outer loop that maintains continuous PWM operation until an external condition requests termination.
2. The inner `for` loop iterates over the PWM period in increments of `PWM_STEP`. The loop variable `t` effectively indexes the current step within the period.

3. For each step, the boolean channel values (`r_value`, `g_value`, `b_value`) are computed by comparing the current step index `t` with the corresponding duty count. If the index is strictly less than the duty count, the channel is considered ON for that step.
4. `rgb_led_pwm_step(...)` is the hardware abstraction that applies the computed ON/OFF values to the LED pins. It is expected to be non-blocking and to update GPIO (or LED driver) outputs accordingly.
5. `k_sleep(K_MSEC(PWM_STEP))` yields the CPU for the duration of one step. This implements the time base for the PWM emulation while allowing other RTOS threads to run.
6. After sleeping, the loop checks `k_sem_take(&main_sem, K_NO_WAIT)` to determine whether a semaphore has been signalled. If the semaphore is available, the loop exits gracefully. The semaphore is triggered either when the user presses the button or when the 30-second timeout associated with each *ADVANCED* mode cycle elapses.

6.3 Result

capturas

The software-based PWM strategy successfully reproduces the colour detected by the RGB sensor. The LED output maintains the same chromatic proportions observed in the ambient light, while ensuring smooth transitions and stable operation. The system preserves full compatibility with the existing hardware configuration and maintains identical behaviour to the *NORMAL* mode regarding synchronisation, timing, and user interaction.

7 Conclusions and Future Works

8 Bibliography

[1] "STM32WL55JC — Product - STMicroelectronics," Accessed: Dec. 9, 2025. [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32wl55jc.html.

[2] A. Industries. "Photo Transistor Light Sensor," Accessed: Nov. 24, 2025. [Online]. Available: https://www.adafruit.com/product/2831.

[3] "SparkFun Soil Moisture Sensor - SparkFun Electronics," Accessed: Nov. 24, 2025. [Online]. Available: https://www.sparkfun.com/sparkfun-soil-moisture-sensor.html.

[4] A. Industries. "Adafruit Si7021 Temperature & Humidity Sensor Breakout Board," Accessed: Nov. 24, 2025. [Online]. Available: https://www.adafruit.com/product/3251.

[5] A. Industries. "RGB Color Sensor with IR filter and White LED - TCS34725," Accessed: Nov. 24, 2025. [Online]. Available: https://www.adafruit.com/product/1334.

[6] A. Industries. "Adafruit Triple-Axis Accelerometer - ±2/4/8g @ 14-bit - MMA8451," Accessed: Nov. 24, 2025. [Online]. Available: https://www.adafruit.com/product/2019.

[7] A. Industries. "Adafruit Ultimate GPS Breakout - 66 channel w/10 Hz updates," Accessed: Nov. 24, 2025. [Online]. Available: https://www.adafruit.com/product/746.

[8] "Curso: Embedded platforms and communications for iot — MOODLE UPM - OFICIALES 25-26," Accessed: Nov. 24, 2025. [Online]. Available: https://moodle.upm.es/titulaciones/oficiales/course/view.php?id=3568.

[9] "Zephyr API Documentation: Introduction," Accessed: Nov. 24, 2025. [Online]. Available: https://docs.zephyrproject.org/latest/doxygen/html/index.html.

Appendix A - GitHub

A.1 GitHub Repository - Source Code

The complete source code of the Plant Monitoring System project is hosted on GitHub and can be accessed via the following link: `https://github.com/Estelamb/Embedded_IoT`.

A.2 GitHub Pages - Documentation

The project documentation can be accessed through the following link: `https://estelamb.github.io/Embedded_IoT/`.

A.3 GitHub Action - Workflow

Listing 8: GitHub Action workflow

```
name: Cppcheck, Zephyr Build, Valgrind and Doxygen

on:
  push:
    branches: [ main ]
  workflow_dispatch:

env:
  ZEPHYR_BASE: ${{ github.workspace }}/zephyrproject/zephyr
  PROJECT_DIR: plant_monitoring_system

jobs:
  cppcheck:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Install cppcheck
        run: sudo apt-get update && sudo apt-get install -y cppcheck

      - name: Run cppcheck
        run: |
          if [ -d "plant_monitoring_system" ]; then
            cd plant_monitoring_system
          fi
          cppcheck --enable=all --inconclusive --std=c++17 -I include src

  build:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Install system dependencies
        run: |
          sudo apt-get update
          sudo apt-get install -y \
            python3-dev \
            python3-pip \
            python3-venv \
            cmake \
            ninja-build \
            gcc \
            g++ \
            valgrind \
            device-tree-compiler
```

```
    - name: Install West and Zephyr dependencies
      run: |
        pip3 install west
        pip3 install -r
            https://raw.githubusercontent.com/zephyrproject-rtos/zephyr/main/scripts/requirements.txt

    - name: Set up Zephyr environment
      run: |
        west init zephyrproject
        cd zephyrproject
        west update
        west zephyr-export

    - name: List available boards to verify native_sim
      run: |
        cd zephyrproject/zephyr
        west boards | grep native

    - name: Build project for native_sim (Valgrind compatible)
      run: |
        cd $PROJECT_DIR
        source $ZEPHYR_BASE/zephyr-env.sh
        west build -b native_sim --build-dir build_valgrind --pristine always

    - name: Find the generated executable
      run: |
        cd $PROJECT_DIR/build_valgrind/zephyr
        find . -name "zephyr*" -type f -executable
        ls -la

    - name: Upload executable for Valgrind
      uses: actions/upload-artifact@v4
      with:
        name: zephyr-native-executable
        path: ${{ env.PROJECT_DIR }}/build_valgrind/zephyr/zephyr.exe
        retention-days: 7

valgrind:
  runs-on: ubuntu-latest
  needs: build
  steps:
    - name: Install Valgrind
      run: sudo apt-get update && sudo apt-get install -y valgrind

    - name: Download native executable artifact
      uses: actions/download-artifact@v4
      with:
        name: zephyr-native-executable
        path: valgrind_test/

    - name: Make executable
      run: chmod +x valgrind_test/zephyr.exe

    - name: Run Valgrind memory analysis
      run: |
        echo "=== Running Valgrind memory check ==="
        timeout 30s valgrind --leak-check=full --track-origins=yes --show-leak-kinds=all
            valgrind_test/zephyr.exe || echo "Valgrind completed or was terminated"

    - name: Run basic Valgrind check
      run: |
        echo "=== Running basic Valgrind ==="
```

```
        timeout 30s valgrind valgrind_test/zephyr.exe || echo "Valgrind completed or was
            terminated"

  doxygen:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout code
        uses: actions/checkout@v4

      - name: Install Doxygen
        run: sudo apt-get update && sudo apt-get install -y doxygen graphviz

      - name: Generate documentation for plant_monitoring_system
        run: |
          cd plant_monitoring_system
          doxygen Doxyfile

      - name: Create docs directory if it doesn't exist
        run: |
          cd plant_monitoring_system
          mkdir -p docs/html

      - name: Verify HTML documentation output
        run: |
          ls -la ./plant_monitoring_system/docs/
          ls -la ./plant_monitoring_system/docs/html

      - name: Deploy to GitHub Pages
        uses: peaceiris/actions-gh-pages@v3
        with:
          github_token: ${{ secrets.GITHUB_TOKEN }}
          publish_dir: ./plant_monitoring_system/docs/html
```