Implementation of an embedded platform for plant monitoring IoT system

Project documentation

Diego Aceituno Seoane

Embedded Platforms and Communications for IoT
Fall 2024

Contents

List of Figures

List of Tables

GLOSARY

CPU Central Processing Unit 19

FSM Finite State Machine 13

I2C Inter-Integrated Circuit 1, 4, 7, 8, 9, 11, 14, 16, 23

IoT Internet of Things 4

ISR Interrupt Service Routine 19, 20

MCU MicroController Unit 11, 22

NMEA National Marine Electronics Association 10, 17

ODR Output Data Rate 8

TAOS Texas Advanced Optoelectronic Solutions 8

UART Universal Asynchronous Receiver-Transmitter 4, 7, 10, 11, 17

1 INTRODUCTION

This document presents the document of the final project developed for the "Embedded platforms and communications for IoT (Internet of Things)".

The final project consist on the design, integration, implementation and validation of a IoT system to control biological data and other parameters like the GPS location of plants that have unique characteristics such as bonsais. This will allow farmers and producers to detect early problems in crops, problems in the soil or in the transportation of the plants.

This system is centered around a embedded solution based on `ARM Cortex M0+`, with low power consumption and all the capabilities needed to cover the requirements. Also, several sensors are used and integrated in the design to obtain all the previous parameters mentioned. Different connections such as `I2C (Inter-Integrated Circuit)` and `UART (Universal Asynchronous Receiver-Transmitter)` are used to communicate these elements with the `CPU`. With the usage of `Mbed-OS`, a thread-based software solution is used to achieve all the functionalities.

Finally, the complete solution is presented in Figure 1. In the center, the microcontroller connects to two protoboards:

- The left one includes a moisture sensor, a phototransistor and a RGB Led, powered by 3.3V.
- The Right one includes the accelerometer, the color sensor, the temperature and humidity sensor and the GPS, powered by 5V.

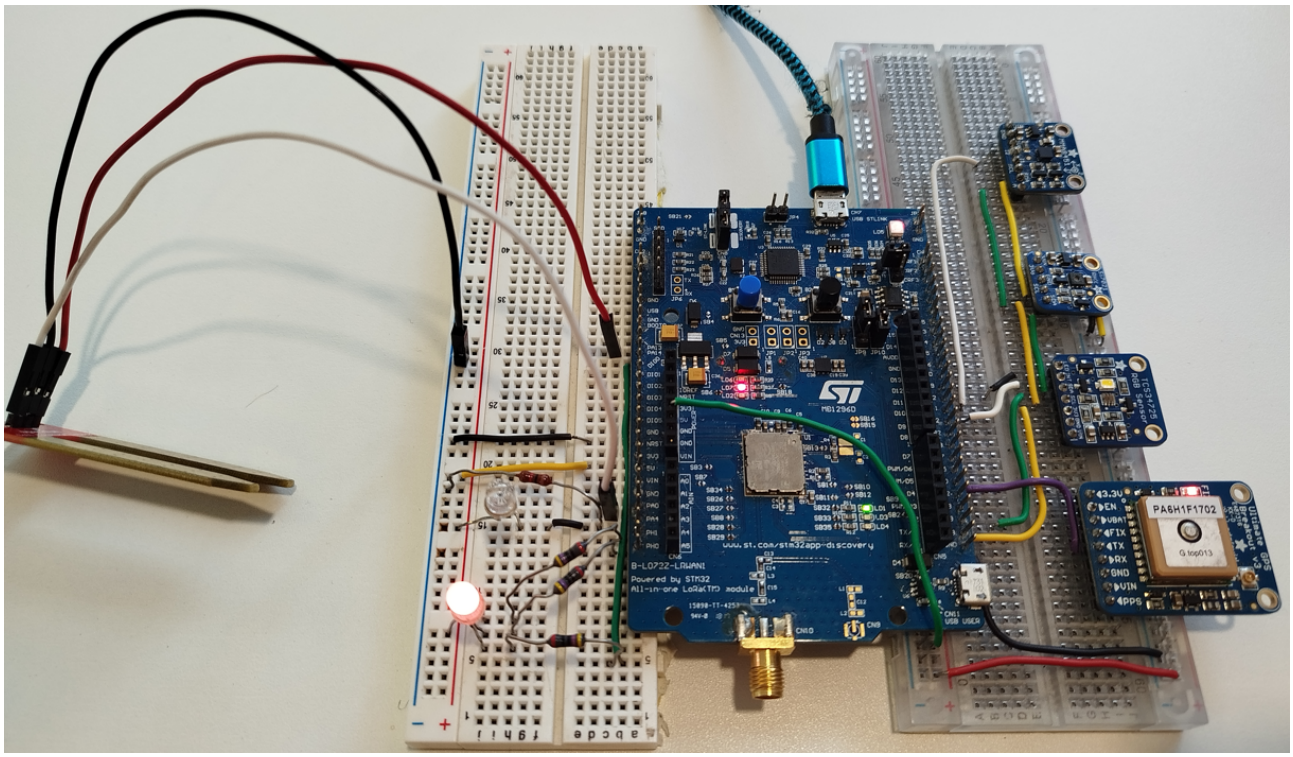


Figure 1: Final solution developed

2 REQUIREMENTS ANALYSIS

2.1 Specifications required and implemented

This section presents in the next tables the list of requirements needed to implement, as well as the implementation status for each specification.

Req. ID	Requirement description	Implemented
SR1	The temperature in the range of -10ºC to 50ºC. Accuracy to one tenth of a degree.	Y
SR2	The relative humidity in the range of 25%HR to 75%HR. Accuracy to one tenth of a percent.	Y
SR3	Ambient light in %, corresponding 0% to total darkness and 100% to maximum light. Accuracy to one tenth of a percent.	Y
SR4	Soil moisture in %, corresponding 0% to total dryness and 100% to maximum moisture. Accuracy to one tenth of a percent.	Y
SR5	Colour of one leaf of the plant. The four associated parameters are clear, red, green, and blue values.	Y
SR6	The global location of the plant should be registered. The GPS module also offers the current time (only time, the date is optional), that will be used to timestamp all the measurements taken by the system.	Y
SR7	The acceleration of the plant. At least the three axes (X, Y and Z) values should be monitored.	Y
GR1	The system must be robust and stable.	Y
GR2	Task partitioning and threads management should be stablished according to the requirements.	Y
GR3	The system starts in TEST MODE and changes from one operating mode to the next one (TEST – NORMAL ADVANCED - TEST...) by pressing the blue button B1 on the B-L072Z-LRWAN1 board in a circular way.	Y

Table 1: General requirements implementation status

Req. ID	Requirement description	Implemented
TM1	Check connections and sensor management.	Y
TM2	All of the required variables should be monitored every 2 seconds.	Y
TM3	The system sends every 2 seconds all the measured values to the computer (using the USB virtual COM port of the B-L072Z-LRWAN1 board).	Y
TM4	The RGB LED should be coloured in the dominant colour detected by the colour sensor.	Y
TM5	In this mode, the LED1 of the B-L072Z-LRWAN1 board should be ON.	Y

Table 2: Test mode requirements implementation status

Req. ID	Requirement description	Implemented
NM1	All the required variables should be monitored with a cadence of 30s.	Y
NM2	The system sends every 30 seconds all the measured values to the computer (using the USB virtual COM port of the B-L072Z-LRWAN1 board).	Y
NM3	The system calculates the mean, maximum and minimum values of temperature, relative humidity, ambient light and soil moisture every hour. These values are sent to the computer when calculated.	Y
NM4	The system calculates the dominant colour of the leave every hour. This means to calculate which colour has appeared as dominant more times during the last hour. This value is sent to the computer when calculated.	Y
NM5	The system calculates the maximum and minimum values of the three axes (X, y and Z) of the accelerometer every hour. These values are sent to the computer when calculated.	Y
NM6	The global location of the plant (coordinates) is sent to the computer every 30 seconds. This should include the GPS time (UTC) converted to local time.	Y
NM7	Limits for every measured variable (temperature, humidity, ambient light, soil moisture, colour and acceleration) should be fixed. If the current values of the measured parameters are outside the limits, the RGB LED should indicate this situation using a different colour for every parameter.	Y
NM8	In this mode, the LED2 of the B-L072Z-LRWAN1 board should be ON.	Y

Table 3: Normal mode requirements implementation status

2.2 Extra specifications implemented

Req. ID	Requirement description	Implemented
E1	The date is also extracted from the GPS data.	Y
E2	Compensation for the spectral response of the colour sensor is introduced.	Y
E3	Automatic generation of the documentation with a Doxygen file and automatic publishing in a github pages.	Y

Table 4: Extra requirements implemented

3 HARDWARE DESIGN

For this section and phase of the project, the next steps were done:

- Analyzing the hardware elements.
- Analyze the connections needed.
- Select the pin connections for every element.
- Mount the system.

This approach led to a faster design of the solution.

3.1 Hardware elements

For the development of the project, all the hardware elements were identified and studied to obtain information of all the possible utilities that the system could use.

3.1.1 Main Controller

For this project, the `B-L072Z-LRWAN1`[1] in Figure 2 was used.

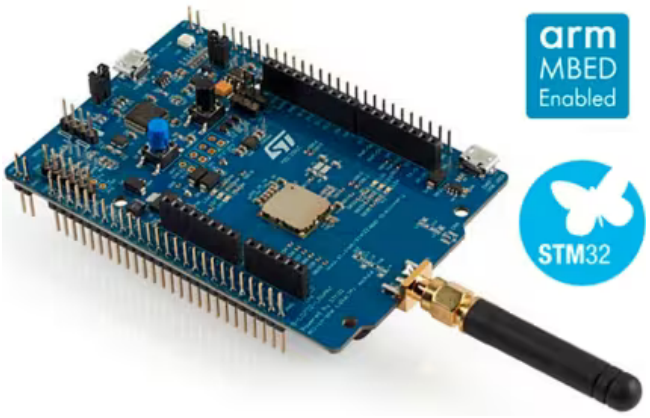


Figure 2: B-L072Z-LRWAN1 LoRa discovery kit

It's a development board by STM, integrating a `STM32L072CZ`[2] arm M0+ processor, that can work at a max clock speed of `32 MHz`.

The board has pins to connect directly to an Arduino UNO.

Some examples of the different capabilities and peripherals of the board are:

- 192KB of flash memory.
- 20KB of RAM.
- USER and RESET button.
- I2C and UART connections.
- 16 bit timers along with LowPower timers.
- Low Power modes.
- A SX1276 transceiver that will be used to implement LoraWan communications on the next subject.

3.1.2 Accelerometer

The accelerometer for this project is the Adafruit breakout board[3] for the MMA8451Q[4] seen in Figure 3.

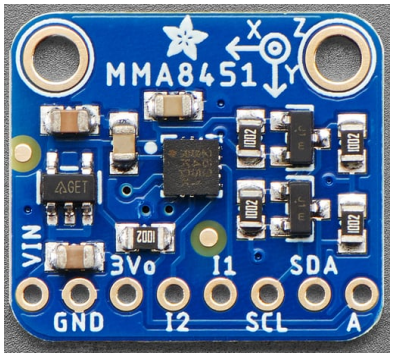


Figure 3: MMA8451Q by Adafruit

The MMA8451Q chip by Freescale if the most precise of it's own family, with a triple-axis 14 bit accelerometer. The sensor communicates with the I2C interface, capable of `400 kHz`, but as will be seen in chapters, the sensor expects a repeated-start communication, this would be a later source for problems.

The chip uses 3.3V, but the Adafruit regulator allows for 5V power supply.

The sensor allows for the configuration of several parameters such as:

- Scale of the measurement.
- ODR (Output Data Rate) as low as `1.56 Hz`.
- Power modes.
- Interruptions in two pins from situations such as: orientation changes, motion detection, freefall, pulse detection, etc.

All this information was taken from the dataset[4].

3.1.3 Color sensor

For the analysis of the plant leaves, the TCS34725[5] in Figure 4 is used.

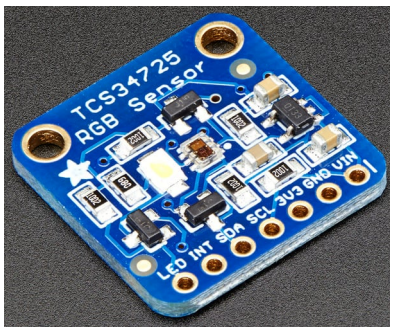


Figure 4: TCS34725 RGB Color sensor by Adafruit

This chip by TAOS (Texas Advanced Optoelectronic Solutions) in the 5V compatible breakout board by Adafruit consist on a 3 by 4 photodiode array connected to several integrators that analyze different

parts of the visible spectrum. It includes an IR blocking filter integrated in the chip that allows for color sensing closer to the vision of humans. Finally, it includes a LED that is driven by the correspondent pin.

It provides readings for clear, red, green and blue values in a 16 bit format. The user can configure things such as:

- Integration time.
- Wait time between readings.
- Interruptions for the clear reading, both in low and high thresholds.
- Analog gain.

The communication is done with an I2C interface, capable of `400 kHz`.

All this information was taken from the dataset[5].

3.1.4 Temperature and Humidity sensor

The Adafruit Si7021A20[6] in Figure 5 is used.

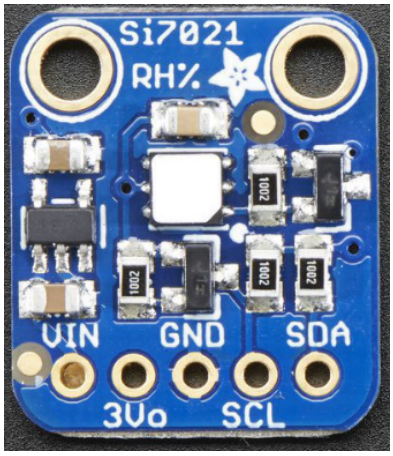


Figure 5: Temperature and Humidity sensor by Adafruit

The chip by Silicon Labs communicates with a I2C interface readings of relative humidity and temperature on-demand. It also allows for the activation of a heating element to remove moisture in the sensor.

All the information was taken from the dataset[6].

3.1.5 Positioning module

The Ultimate GPS v3 by Adafruit[7] in Figure 6 is used.

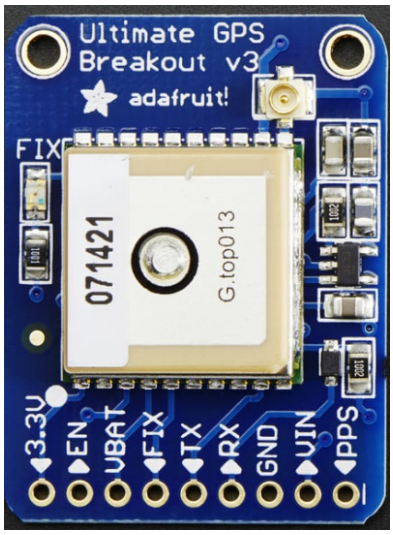


Figure 6: Ultimate GPS v3 by Adafruit

This module communicates all the GPS information by an UART connection. For this, the sensor sends periodic data to the UART with the NMEA (National Marine Electronics Association) data format.

This module achieves a level of sensitivity up to `-165 dBm` with an update rate up to `10 Hz`.

It includes a PPS output signal for timing applications, and functionalities such as EASY positioning for orbit prediction or AlwaysLocate function to change power consumption in reaction to the needs.

Sadly, the messages and the communication interface to configure this parameters is propietary, and as such, it won't be used in the project.

All the information was taken from the dataset[8].

3.1.6 Other elements

The project will also use:

- **SEN-13322 Analog Soil Moisture Sensor**[9]: This sensor by SparkFun allows to read the moisture level between the two pads.
- **RGB Led**: A common anode RGB led.
- **Phototransistor**[10]: This element allows for the reading of the brightness in the environment.

3.2 Block Diagram

To design the solution, the diagram in Figure 7 identifies the main blocks and hardware interfaces needed. The left side and the GPS include 5V powered elements, and the right side contains 3.3V elements.

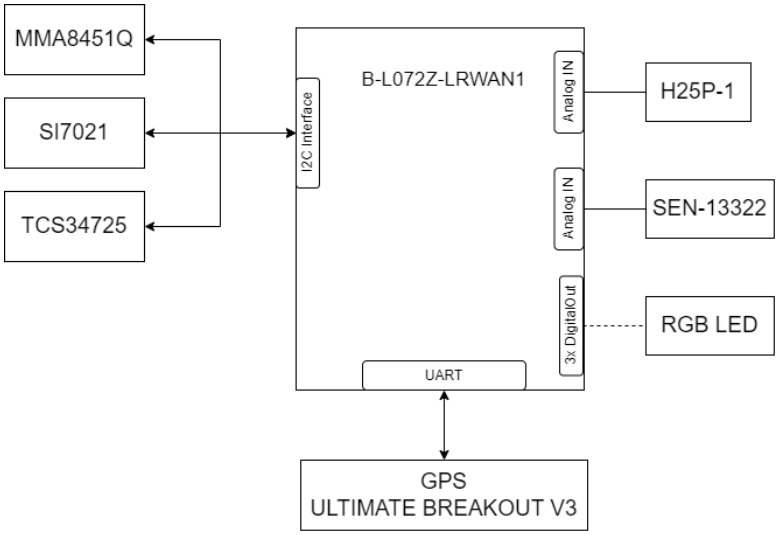
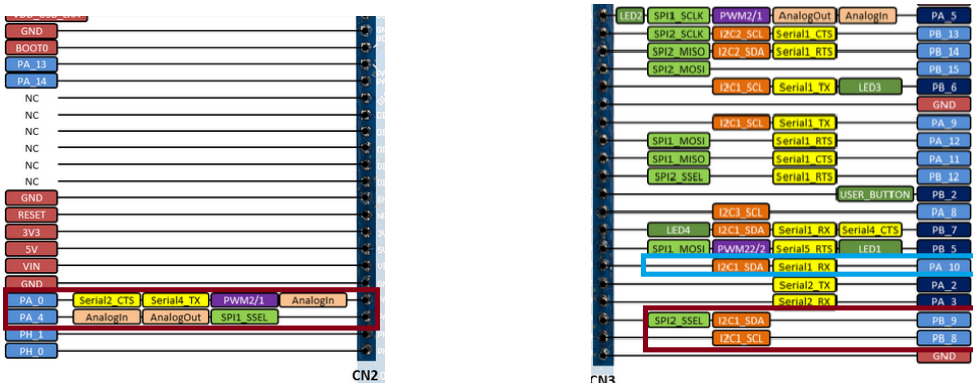


Figure 7: Block identification for the hardware connections

Next, the selection of the connections to the L072CZ were done in the next order: I2C, UART, analog connections and finally DigitalOut connections. It is important to note that, as we needed to indicate the working mode with the board leds, we can't use the pins connected directly to those elements.

- For the I2C, the MCU (MicroController Unit) offers 3 options, I2C1,I2C2 and I2C3, being the second one the worst in terms of capabilities.For this system and this use case, I2C1 was selected because, as seen in Figure 8, its the interface with less conflicts.
- For the Serial interface to communicate with the GPS, the options are `USART1`, `USART2`, `USART4` and `USART5`, with the last two having less capabilities. In this case, the L072CZ really only offers the first two, but the `USART2` is used as a virtual COM port with the ST-Link. The selection was the `USART1`, also, as can be seen in Figure 8, it has the least conflicts of all.
- For the analog sensors, the `PA_0` and `PA_4` pins were selected, as we only need two Analog IN connections.



(a) Analog selection

(b) I2C and UART selection

Figure 8: Communication interfaces selection

At last, the final DigitalOut connections were defined. The final connections for every element are defined in the next tables.

Sensor	**Sensor Pin**	**L072CZ connector**	**L072CZ Pin name**
H25P-1	SIG	CN2_24	PA_4
SEN-13322	SIG	CN2_23	PA_0
RGB Led	RED	CN2_25	PH_1
RGB Led	GREEN	CN2_26	PH_0
RGB Led	BLUE	CN2_10	PA_14
	GND	CN2_17	GND
	VCC	CN2_19	+3.3V

Table 5: Connections for the 3.3V elements

Sensor	**Sensor Pin**	**L072CZ connector**	**L072CZ Pin name**
TCS34725	LED	CN3_13	PA_9
TCS34725	INT	CN3_15	PA_11
GPS	SerialTX	CN3_26	PA_10
MMA8451Q	I2	CN3_14	PA_12
	SCL	CN3_25	PB_8
	SDA	CN3_24	PB_9
	GND	CN3_26	GND
	VCC	CN2_20	+5V

Table 6: Connections for the 5V elements

4 SOFTWARE

4.1 Description of the implementation

The solution is implemented with a FSM (Finite State Machine) with three states, starting from a TEST mode and switching to the next with the USER button of the L072CZ. A common factor of these states is that in all of them, all the data from the sensors need to be obtained.

To solve the need for asynchronous data obtention from the sensors, the solution is based on threads and queues. There are three threads:

- Main thread with the FSM.
- I2C thread.
- GPS thread.

These thread are signalized to wake up when new measurements are needed by the FSM. When all the information is collected, the information goes to the main thread through a Mbed-OS queue. When all the necessary information is obtained, the main thread sends the new data to the computer and waits a set amount of time to start the process again.

4.2 Module, threads and communications design

To make the design modular, each hardware element is implemented in a single module, with a `.cpp` and a `.h` file.

Considering the asynchronous nature of some of the sensors, the i2c sensors were implemented in an independent thread and the GPS in another thread. In the , the final module design as well as the threads implemented can be seen in Figure 9.

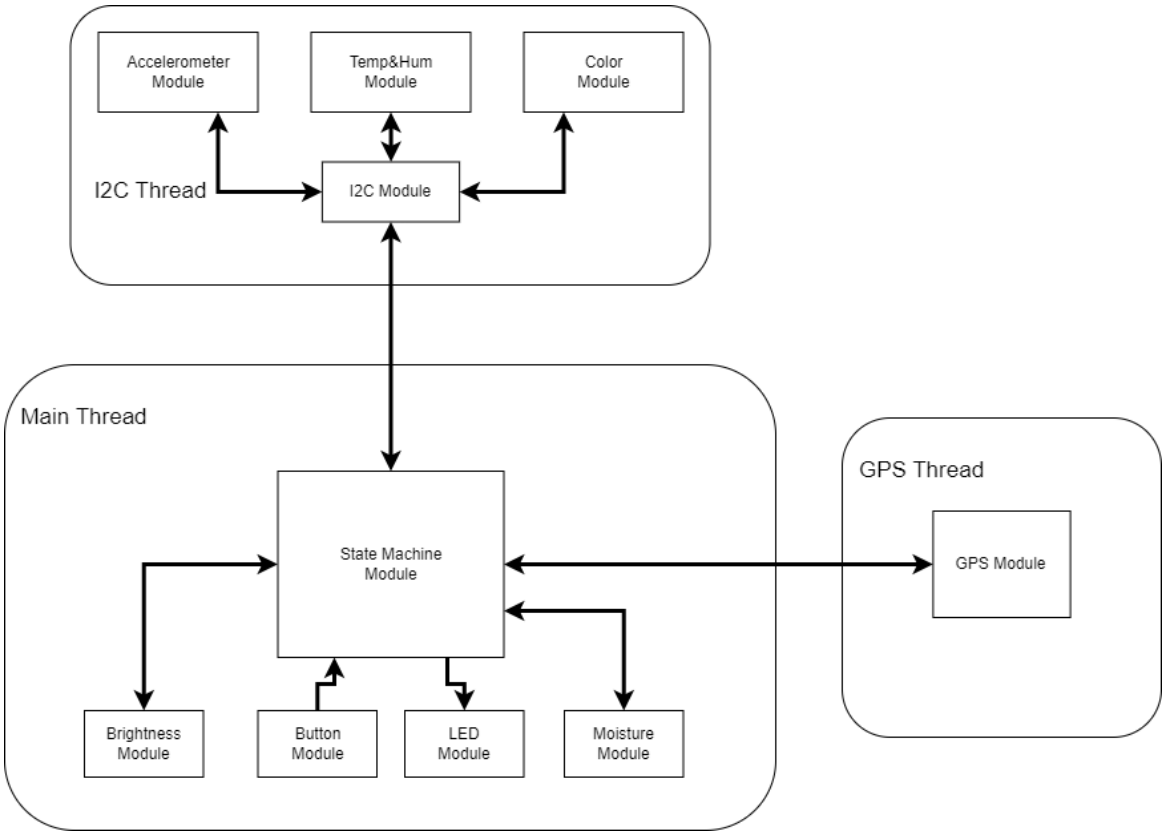


Figure 9: Module diagram of the software and threads

For the communications from the control to the I2C and the GPS, there was only need to indicate to the I2C and GPS threads to launch all the necessary measurements on demand, so in this case signals were used. For the data flowing from the sensors to the control module, as signals in a processor are limited(in our case 32 bit register for signals), a queue was used.

A queue is a thread safe mechanism of low level that implements a FIFO buffer with a in endpoint and an out endpoint. Any process can insert new data into the queue with a blocking call and the same can be done to extract the data. Finally, the data inside the queue is managed by the queue itself(at least in the implementation in Mbed-OS).

To reduce the memory usage, only one queue was defined for the whole system. With this done, extra logic was needed to differentiate messages from different sensors, so a message structure was defined like a pseudo-protocol. This can be seen in the appendix.

4.3 State machine module

At the start of the system, the main calls for the initialization of all the modules. This is important because the creation of the I2C and GPS threads take some time. When this is done, the main initializes the state machine and enters a loop to do a cycle of the state machine.

The state machine includes two timeouts that are configured for each state, depending on the needs.

- The first timeout is configured as the time needed between sending the information to the computer.
- For the second timeout, the configuration is to make it generate an interruption `400 ms` earlier. When this happens, the controller sends signals for every thread. This signal enforces the reading of new measurements.

When the signals are sent, the main thread is sent to sleep `400 ms`, at that time, it wakes up and analyzes all the data that has been sent to the queue. This time was selected as a time to let the other threads collect all the necessary data.

When all the messages are collected, the data is processed depending on the state of the system:

- TEST mode: as the requirements specify, the data is formatted and sent to the computer as in Figure 10. Depending on the highest color component, the RGB led changes accordingly.

```
MODE: TEST

        SOIL MOISTURE: 0.2%
        LIGHT: 62.1%
        GPS: #Sats: 3 Lat(UTC): 40.231712 N Long(UTC): 3.359932 W Altitude: 97 M GPS time: 16:5:31 CET  Day 1, Month 12, Year 2024
        COLOR SENSOR: Clear: 4989 Red: 1851 Green: 1873 Blue: 2078 -- Dominant color: blue
        ACCELEROMETERS: X_axis: 4.6 m/s², Y_axis: 0.2 m/s², Z_axis: 8.5 m/s²
        TEMP/HUM: Temperature:  16.7 °C,        Relative Humidity: 71.5%
```

Figure 10: Information sent to the computer in the TEST mode

- NORMAL mode: as the requirements specify, the time the messages are read, maximum values and mean values are calculated, as well as situations like extreme moisture.

  All the limitations are in the Table 7.

Parameter	**High limit**	**Low limit**	**Led color**
Temperature	35	10	red
Humidity	75	25	blue
Brightness	80	10	Yellow
Moisture	85	5	Purple
Color	Blue		White

Table 7: Limits defined for the parameters in the NORMAL mode

  All of this information is sent to the computer, and when one hour has passed, the maximum and mean values are also sent. The format is presented in Figure 11.

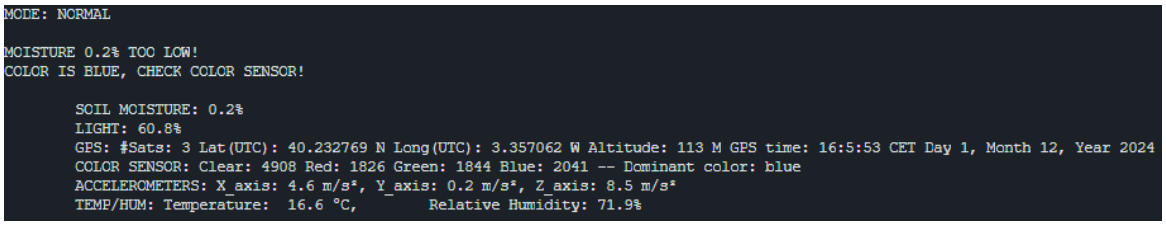


Figure 11: Information sent to the computer in the NORMAL mode

- ADVANCED mode: this mode is presented and described in it's own chapter.

Finally, the USER button allows for switching modes in a circular way. When this button is pressed:

1. The timeouts are re-configured.
2. The led of the board that is on changes to indicate the new state.
3. The RGB led is turned off.
4. The thread waits `400 ms` for any new messages from the other threads, in order to empty the queue for the new state.

As a final note, as this state machine is working under the main thread of Mbed-OS, no changes were done to the stack or any other property of this thread.

4.4 I2C module

The I2C module controls the measurement of the accelerometer, temperature and humidity and color sensors.

As there are three different sensors with different requirements, at the initialization of the I2C thread, all the sensors are started sequentially, using a global I2C object. It must be noted that as all the sensors support it, the I2C is used with a frequency of `400 kHz`.

For the initialization, the next steps are done:

- **Initialization of the accelerometer**: first the accelerometer is reseated, when this finish, the dynamic configuration is set to 8G, this is because the use case doesn't need that much high sensibility.
- **Initialization of the color sensor**: for the color sensor, the dataset specifies a different approach with a command register. In this case, this sensor is configured to remove any wait state, this is done because the sensor has a limit of wait time less that 30 seconds, so the readings are done on demand, this is by turning on and off the sensor. Lastly, the sensor is configured with an integration time of `154 ms`.

When all the configuration is done, the thread enters a loop where it waits for an `I2C_SIGNAL` from the state machine. Once this happens, the flag is cleared and the next steps happen:

1. Reading of the accelerometer data, conversion of the data and queue acceleration message sent.
2. Powering the color sensor, waiting for the integration time and reading of the data to send the color message. Power off the sensor.
3. Force a reading of humidity and temperature, converting values as told in the dataset[6].

To finalize, the thread goes back to wait for another signal from the state machine.

4.5 GPS module

The GPS module implements the GPS thread and focuses on obtaining and parsing the data from the GPS chip.

The GPS is constantly sending data to the serial connection in a baud rate of `9600`. To read this data, when the thread wakes from a `GPS_SIGNAL`, the thread is constantly reading the characters obtained in the UART and adding them to a buffer. If the next character is a new line character, the buffer is sent to the parse_gps_sentence function.

For this parsing, as the system only needs the location, time and data, the parsing is only done to specific sentences of the NMEA data format in Figure 12.

Message	Description
GGA	Time, position and fix type data
GLL	Latitude, longitude, UTC time of position fix and status
GSA	GPS receiver operating mode, satellites used in the position solution, and DOP values
GSV	Number of GPS satellites in view satellite ID numbers, elevation, azimuth, & SNR values
MSS	Signal-to-noise ratio, signal strength, frequency, and bit rate from a radio-beacon receiver
RMC	Time, date, position, course and speed data
VTG	Course and speed information relative to the ground
ZDA	PPS timing message (synchronized to PPS)
150	OK to send message
151	GPS Data and Extended Ephemeris Mask
152	Extended Ephemeris Integrity
154	Extended Ephemeris ACK

Figure 12: NMEA possible sentences[11]

In this use case, the GPS thread only searches for:

- **GGA**: contains the information about the actual time, latitude, longitude, altitude and number of satellites detected.
- **RMC**: contains information about the actual time, the actual date and other data that the system doesn't use.

To parse the sentences, the function `strncmp` was used to analyze the sentence type and `sscanf` to parse all the information like a pseudo-regex.

When both sentences are found, the message for the queue is built and inserted, finally, the GPS thread goes back to wait for a new signal from the state machine.

4.6 Code size

For the code size of the project, the information resulting from the compilation in Mbed Studio is presented in the next figures. It must be noted that no changes to the default compilation profiles were done.

Module	.text	.data	.bss
[lib]\c_p.l	16093(+714)	16(+0)	348(+0)
[lib]\fz_ps.l	3986(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	520(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	26(+0)	0(+0)	0(+0)
mbed-os\cmsis	11585(+2277)	168(+0)	6993(+128)
mbed-os\connectivity	831(+583)	0(+0)	0(+0)
mbed-os\drivers	5792(+818)	0(+0)	84(+0)
mbed-os\features	18(+0)	0(+0)	0(+0)
mbed-os\hal	2386(+712)	8(+0)	114(+0)
mbed-os\platform	7111(+2305)	64(+0)	420(-36)
mbed-os\rtos	1450(+566)	0(+0)	0(+0)
mbed-os\targets	20148(+1443)	8(-4)	1369(-19)
modules\accelerometer_sensor	580(+72)	0(+0)	36(+0)
modules\brightness_sensor	68(+8)	0(+0)	104(+0)
modules\button	104(+20)	0(+0)	84(+0)
modules\color_sensor	328(+80)	0(+0)	66(-2)
modules\gps_sensor	755(+80)	7(-1)	1156(-3)
modules\i2c	140(+24)	0(+0)	188(+0)
modules\led	164(+28)	0(+0)	84(+0)
modules\moisture_sensor	136(+40)	0(+0)	132(+0)
modules\state_machine	5856(+1416)	31(+0)	902(-6)
modules\temp_hum_sensor	260(+36)	0(+0)	31(-1)
Subtotals	78413(+11222)	302(-5)	13135(+61)

Total Static RAM memory (data + bss): 13437(+56) bytes
Total Flash memory (text + data): 78715(+11217) bytes

(a) Debug compilation

Module	.text	.data	.bss
[lib]\c_p.l	15429(-664)	16(+0)	348(+0)
[lib]\fz_ps.l	3986(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	520(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	26(+0)	0(+0)	0(+0)
mbed-os\cmsis	10633(-952)	168(+0)	6865(-128)
mbed-os\connectivity	234(-597)	0(+0)	0(+0)
mbed-os\drivers	5349(-443)	0(+0)	84(+0)
mbed-os\features	0(-18)	0(+0)	0(+0)
mbed-os\hal	2242(-144)	8(+0)	114(+0)
mbed-os\platform	6412(-699)	64(+0)	420(+0)
mbed-os\rtos	1479(+29)	0(+0)	0(+0)
mbed-os\targets	19506(-642)	8(+0)	1381(+12)
modules\accelerometer_sensor	520(-60)	0(+0)	36(+0)
modules\brightness_sensor	60(-8)	0(+0)	104(+0)
modules\button	88(-16)	0(+0)	84(+0)
modules\color_sensor	300(-28)	0(+0)	66(+0)
modules\gps_sensor	739(-16)	7(+0)	1156(+0)
modules\i2c	124(-16)	0(+0)	188(+0)
modules\led	184(+20)	0(+0)	84(+0)
modules\moisture_sensor	104(-32)	0(+0)	132(+0)
modules\state_machine	5132(-724)	31(+0)	902(+0)
modules\temp_hum_sensor	244(-16)	0(+0)	31(+0)
Subtotals	73387(-5026)	302(+0)	13019(-116)

Total Static RAM memory (data + bss): 13321(-116) bytes
Total Flash memory (text + data): 73689(-5026) bytes

(b) Develop compilation

Module	.text	.data	.bss
[lib]\c_p.l	15379(+0)	16(+0)	348(+0)
[lib]\fz_ps.l	3986(+0)	0(+0)	0(+0)
[lib]\libcppabi_p.l	44(+0)	0(+0)	0(+0)
[lib]\m_ps.l	520(+0)	0(+0)	0(+0)
anon$$obj.o	32(+0)	0(+0)	1024(+0)
main.o	26(+0)	0(+0)	0(+0)
mbed-os\cmsis	9308(+0)	168(+0)	6865(+0)
mbed-os\connectivity	248(+0)	0(+0)	0(+0)
mbed-os\drivers	4974(+0)	0(+0)	84(+0)
mbed-os\features	18(+0)	0(+0)	0(+0)
mbed-os\hal	1674(+0)	8(+0)	114(+0)
mbed-os\platform	4806(+0)	64(+0)	456(+0)
mbed-os\rtos	884(+0)	0(+0)	0(+0)
mbed-os\targets	18705(+0)	12(+0)	1388(+0)
modules\accelerometer_sensor	508(+0)	0(+0)	36(+0)
modules\brightness_sensor	60(+0)	0(+0)	104(+0)
modules\button	84(+0)	0(+0)	84(+0)
modules\color_sensor	248(+0)	0(+0)	68(+0)
modules\gps_sensor	675(+0)	8(+0)	1159(+0)
modules\i2c	116(+0)	0(+0)	188(+0)
modules\led	136(+0)	0(+0)	84(+0)
modules\moisture_sensor	96(+0)	0(+0)	132(+0)
modules\state_machine	4440(+0)	31(+0)	908(+0)
modules\temp_hum_sensor	224(+0)	0(+0)	32(+0)
Subtotals	67191(+0)	307(+0)	13074(+0)

Total Static RAM memory (data + bss): 13381(+0) bytes
Total Flash memory (text + data): 67498(+0) bytes

(c) Release compilation

Figure 13: Code size and module size for every compilation profile

The most heavy modules in terms of size were:

1. The state machine with 476 lines of code.
2. The GPS with 86 lines of code, but with a parsing of a string, which needs lot of resources.

5 ADVANCED MODE IMPLEMENTATION

The next advanced mode was proposed by the teachers:

Extend the functionality of the original design to enter the system into low-power mode whenever possible. To do this, replicate the NORMAL functionality in ADVANCED mode, where you should put the system to sleep whenever possible, switching to interrupt management for as many processes as possible. You should also power the ground humidity sensor via a DigitalOut signal. Finally, minimum and maximum clear ranges shall be defined for the RGB sensor. If a clear value is out of range, and interruption must happen and the system will return automatically to TEST mode.

5.1 Basics

First, the NORMAL mode was replicated into the advanced mode, with the same timings. Next the ground humidity VCC pin was connected to `PA_13` and the reading configured to first set high the pin and lastly, set to low.

As all the limits were defined in the NORMAL mode, there was no need to define the limits in the advanced mode.

5.2 Implementation of the Low Power mode

The L072CZ board allows for the next main low-power modes:

- Sleep mode: the CPU (Central Processing Unit) is stopped, and all the peripherals continue to operate and can wake up the CPU.
- Low-power run mode: the internal oscillator is set to a low clock speed and the internal regulator in low power mode. The number of enabled peripherals is limited.
- Low-power sleep mode: combination of the two above.
- Stop mode: the lowest consumption while retaining the RAM and register contents.
- Standby mode: the lowest consumption, but when entering this mode, all the state is lost.

Considering the above, all of the modes were valid, but standby.

To implement this, the slides provided by the teachers were used, starting with a program to test the mode. With the test done, the low power mode selected was Low-Power Sleep mode as the Stop mode wasn't able to go back to the code, only could execute the ISR (Interrupt Service Routine).

To implement the Low-Power sleep mode, when the device enters the advanced mode, the next configuration in Figure 14 performed:

```c
/**
* Inline(every call pastes the same code) function to control the sleep of the MCU
*/
inline void deep_sleep_stop(){
    if(sleep_ready){
        /*Prepare the entering to sleep mode*/
        FLASH->ACR &= ~(FLASH_ACR_SLEEP_PD);
        PWR->CR |= PWR_CR_LPSDSR_Msk;
        SCB->SCR &= ~(SCB_SCR_SLEEPDEEP_Msk);
        SCB->SCR |= SCB_SCR_SLEEPONEXIT_Msk;
    }
    __disable_irq();
    __WFI();
    __enable_irq();
}
```

Figure 14: Low power mode function of the system

- First, the Flash is configured to power on in sleep mode.
- Second, the voltage regulator is put in a low-power state during sleep mode.
- Third, the low-power mode is configured for the sleep mode.
- Finally, the system will re-enter sleep mode when coming from an ISR.

5.3 Interruption management

As explained in the next chapter, the color sensor wasn't able to generate interruptions.

To compensate for this, an orientation detection was implemented with the accelerometer. To make this, the accelerometer initialization was changed:

- The interruptions on orientations were activated and mapped to PIN2 of the accelerometer sensor.
- As we are working with orientation changes, if we don't work around fast changing orientations, we will have problems. To solve this, the internal debounce counter of the accelerometer was put to `100ms`.
- Finally, we clean a first interruption that happens at start of the accelerometer. This is done automatically by reading one specific register.

Finally, Figure 15 presents the end result of data sent to the computer. Also, when some orientation change is detected, it is also sent to the computer.

```
MODE: ADVANCED

MOISTURE 0.2% TOO LOW!
COLOR IS BLUE, CHECK COLOR SENSOR!

        SOIL MOISTURE: 0.2%
        LIGHT: 60.1%
        GPS: #Sats: 5 Lat(UTC): 40.233879 N Long(UTC): 3.354633 W Altitude: 118 M GPS time: 16:6:26 CET Day 1, Month 12, Year 2024
        COLOR SENSOR: Clear: 4814 Red: 1802 Green: 1811 Blue: 2000 -- Dominant color: blue
        ACCELEROMETERS: X_axis: 4.5 m/s², Y_axis: 0.2 m/s², Z_axis: 8.5 m/s²
        TEMP/HUM: Temperature:  16.5 °C,        Relative Humidity: 72.3%
```

Figure 15: Advanced information sent to the computer

6 PROBLEMS DETECTED AND SOLUTIONS IMPLEMENTED

This chapter describes problems that appeared during the development of the project as well as solutions implemented.

6.1 Debug mode

When trying to work with the debug mode of Mbed Studio, and in an early version of the project, the debugger wasn't launching. The error can be seen in the next figure.

```
    File "c:\Users\Diego\AppData\Local\Mbed Studio\mbed-studio-tools\python\lib\site-packages\pyocd\probe\
      return self._read_mem(addr, size, Commands.JTAG_READMEM_32BIT, self.MAXIMUM_TRANSFER_SIZE, apsel)
    File "c:\Users\Diego\AppData\Local\Mbed Studio\mbed-studio-tools\python\lib\site-packages\pyocd\probe\
      raise self._ERROR_CLASSES[status](error_message)
pyocd.core.exceptions.TransferError: STLink error (22): DP error
"0014566:ERROR:gdbserver:Unhandled exception in handle_message: STLink error (22): DP error"
Remote failure reply: E01
```

Figure 16: Debug error

As the project involved multiple modules, communication interfaces and threads. Some time was spent understanding the error and getting the debugger to work.

After research and trial and error. The solution was found. The L072CZ connects the `PA_13` and `PA_14` pins directly to the SW debugger of the ST Link as presented in the Figure 17.

Table 17. Alternate functions port A

Port		AF0	AF1	AF2	AF3	AF4
		SPI1/SPI2/I2S2/ USART1/2/ LPUART1/USB/ LPTIM1/TSC/ TIM2/21/22/ EVENTOUT/ SYS_AF	SPI1/SPI2/I2S2/ I2C1/TIM2/21	SPI1/SPI2/I2S2/ LPUART1/ USART5/USB/L PTIM1/TIM2/3/E VENTOUT/ SYS_AF	I2C1/TSC/ EVENTOUT	I2C1/USART1/2 /LPUART1/ TIM3/22/ EVENTOUT
Port A	PA0	-	-	TIM2_CH1	TSC_G1_IO1	USART2_CTS
	PA1	EVENTOUT		TIM2_CH2	TSC_G1_IO2	USART2_RTS/ USART2_DE
	PA2	TIM21_CH1		TIM2_CH3	TSC_G1_IO3	USART2_TX
	PA3	TIM21_CH2		TIM2_CH4	TSC_G1_IO4	USART2_RX
	PA4	SPI1_NSS	-	-	TSC_G2_IO1	USART2_CK
	PA5	SPI1_SCK	-	TIM2_ETR	TSC_G2_IO2	
	PA6	SPI1_MISO		TIM3_CH1	TSC_G2_IO3	LPUART1_CTS
	PA7	SPI1_MOSI		TIM3_CH2	TSC_G2_IO4	-
	PA8	MCO		USB_CRS_ SYNC	EVENTOUT	USART1_CK
	PA9	MCO		-	TSC_G4_IO1	USART1_TX
	PA10	-		-	TSC_G4_IO2	USART1_RX
	PA11	SPI1_MISO	-	EVENTOUT	TSC_G4_IO3	USART1_CTS
	PA12	SPI1_MOSI	-	EVENTOUT	TSC_G4_IO4	USART1_RTS/ USART1_DE
	PA13	SWDIO	-	USB_NOE	-	-
	PA14	SWCLK	-	-	-	USART2_TX
	PA15	SPI1_NSS		TIM2_ETR	EVENTOUT	USART2_RX

Figure 17: Pins used by the SW debugger

When the RGB Led was started, the alternate function of the pin is overwritten, and the connection to the debugger is lost. To solve this, a macro was defined in the system, that can be commented and allow the RGB led module to work with all the colors.

6.2 Problems with the color sensor

The color sensor was configured to implement the requirements of the advanced mode in regards to the interruptions generation. The configuration was properly validated with a logical analyzer.

When everything was configured, no observable change in the logical level of the interrupt pin was seen.

To see the problem in a modular way, an external implementation[12] of an interface to the sensor was used. When trying to enable the interruptions, the code returned a 0, indicating that a problem had occurred.

6.3 Problems with the accelerometer

When working with the accelerometer, the data obtained wasn't being read properly. When inspected with the logical analyzer, the data that was being sent back to the MCU was fixed. This can be seen in the next figure.

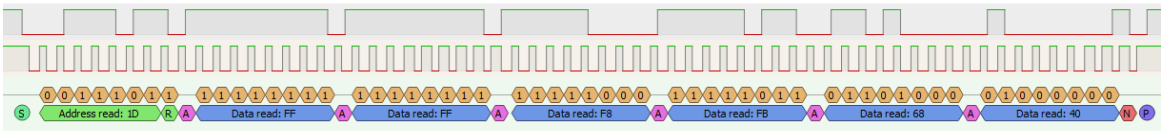


Figure 18: Accelerometer fixed data without repeated start

After some research and, as vaguely indicated by the dataset[4], the sensor expects repeated start between write and read operations.

6.4 Fault problem

When implementing and testing the GPS, and in the final week, some HardFaults were seen in the project. After talking with the teachers of the subject, the main suspect of this problem were the change done in the default stack size of the Mbed-OS threads. In this case, one of the main possible conflict point is the GPS thread.

As the GPS thread performs parsing on strings, the amount of memory needed by the thread is more than expected, to solve this, more stack size was assign to the GPS thread.

7 TESTING AND VALIDATION

7.1 Incremental testing

To maintain coherence of the whole system, the project was implemented with a GIT workflow, this meant that each functionality was implemented in a separated branch and tested until working as the requirements specified.

When the functionality was implemented and tested, it was merged to the project.

7.2 Sensor testing

For the sensors, some public interfaces were used to test the correct functionality of the element:

- Accelerometer: a public repository[13] with a class an a interface was used to the the correct reading of the acceleration values.
- Color sensor: as mentioned in the problems, a public project[12] was used to test the sensor and the interruption capabilities.

7.3 I2C validation

To validate all the initialization of the sensors, the LHT00SU1 logical analyzer in Figure 19 was used.

Figure 19: Logical analyzer used in the project

7.4 Power mode validation

To analyze the correct implementation of the power mode, the current going to the micro-usb connection of the L072CZ was characterized. The Figure 20 shows the current in the test mode and the Figure 21 shows the current in the advanced mode when the system is in sleep mode.

Both pictures are done when the new state is waiting for the timeout. The only difference is that in the test mode picture, the led is turned on, so the current difference between them is less.

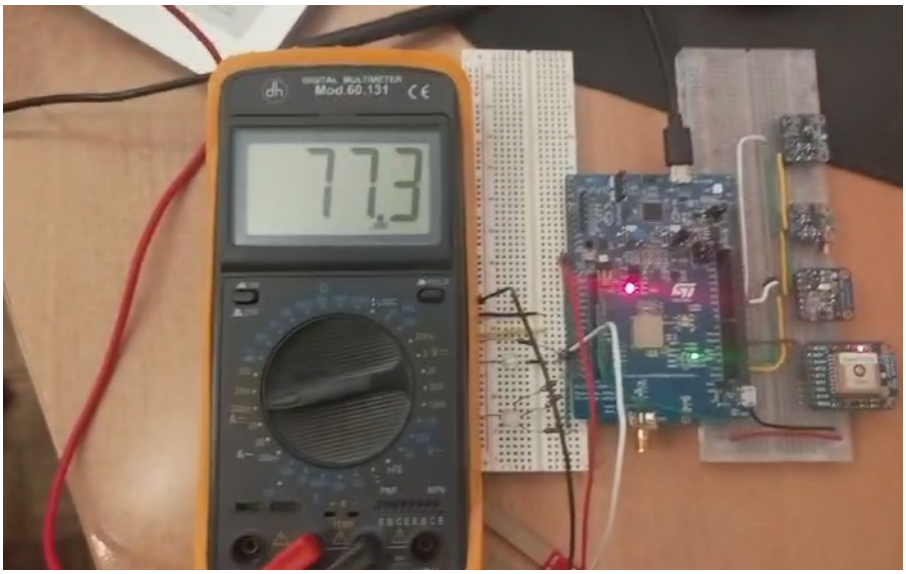


Figure 20: Current in test mode

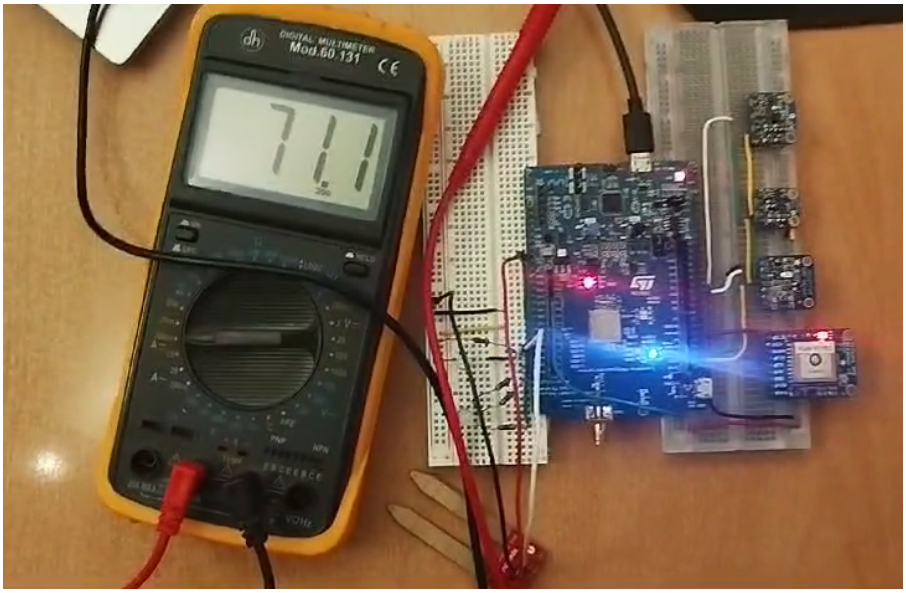


Figure 21: Current in the advanced mode

8 DOCUMENTATION

As this project integrates different kinds of elements and software modules and all the design would be used for next subjects, it was decided that some of the efforts were going to be allocated on generating good documentation of the code.

To approach this, doxygen[14] was used. Doxygen defines both the standard and an application to generate documentation in a Javadoc style in both HTML and LaTex documents.

All the documentation in the code was defined following the doxygen specifications. Lastly, a Doxyfile configuration file is generated considering the needs of the documentation. This created the documentation files but two problems were observed:

1. The default HTML documentation feels old, so a open style for Doxygen called "doxygen-awesome-css"[15] was used. This led to a much clean documentation as in the Figure 22.

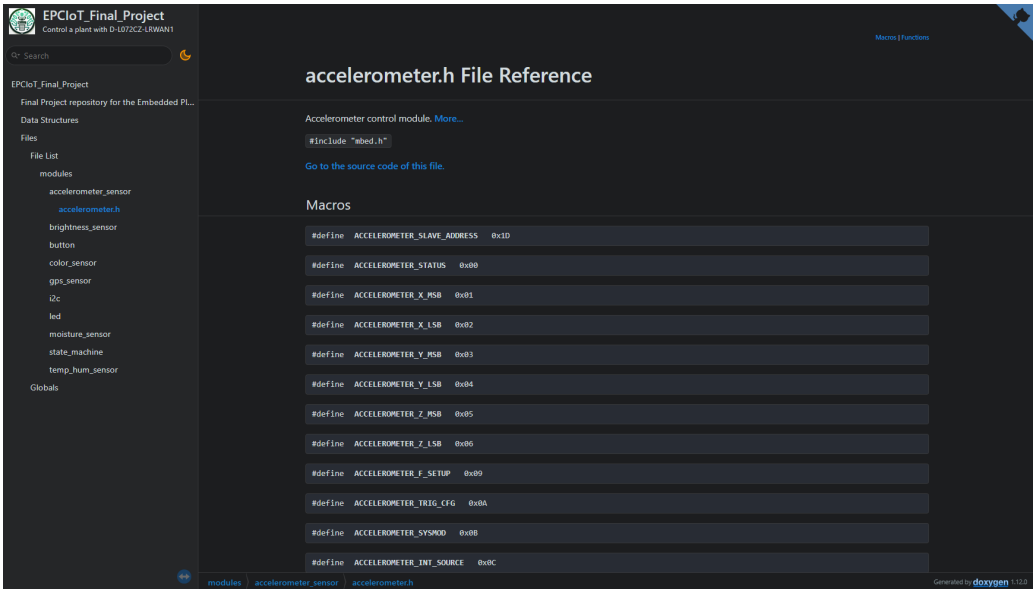


Figure 22: Final documentation with the modified Doxygen

2. As more functionalities were being added, the documentation needed to be re-generated. To solve this, a GitHub workflow was configured to auto-generate and auto-publish the generated HTML with an action in a branch of the repository. With GitHub pages, the automatic deployment was configured.
   The final result is available here: `https://ryvenkappa.github.io/EPCIoT_Final_Project/`

9 RESULTS

The project was successfully implemented following the requirements specified. Also, the advanced mode was implemented and validated, following the requirements told by the teachers of the subject.

Also, for this subject, Doxygen was learnt and implemented to generate the documentation of the project. This was combined with GitHub actions.

The final public repository is available at: `https://github.com/RyvenKappa/EPCIoT_Final_Project`.

10 REFERENCES

[1] *DISCO-L072CZ-LRWAN1 — Mbed.* [Online]. Available: https://os.mbed.com/platforms/ST-Discovery-LRWAN1/ (visited on 11/28/2024).

[2] *STM32L072CZ - Ultra-low-power Arm Cortex-M0+ MCU with 192-Kbytes of Flash memory, 32 MHz CPU, USB - STMicroelectronics.* [Online]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32l072cz.html (visited on 11/28/2024).

[3] L. Ada, *Downloads — Adafruit MMA8451 Accelerometer Breakout — Adafruit Learning System.* [Online]. Available: https://learn.adafruit.com/adafruit-mma8451-accelerometer-breakout/downloads (visited on 11/28/2024).

[4] *MMA8451Q-1.* [Online]. Available: https://cdn-shop.adafruit.com/datasheets/MMA8451Q-1.pdf (visited on 11/28/2024).

[5] *TCS34725.* [Online]. Available: https://cdn-shop.adafruit.com/datasheets/TCS34725.pdf (visited on 10/14/2024).

[6] *Support_Documents_TechnicalDocs_Si7021-A20.* [Online]. Available: https://cdn-learn.adafruit.com/assets/assets/000/035/931/original/Support_Documents_TechnicalDocs_Si7021-A20.pdf (visited on 10/14/2024).

[7] L. Ada, *Downloads — Adafruit Ultimate GPS — Adafruit Learning System.* [Online]. Available: https://learn.adafruit.com/adafruit-ultimate-gps/downloads (visited on 10/14/2024).

[8] *GlobalTop-FGPMMOPA6H-Datasheet-V0A.* [Online]. Available: https://cdn-shop.adafruit.com/datasheets/GlobalTop-FGPMMOPA6H-Datasheet-V0A.pdf (visited on 11/28/2024).

[9] *SparkFun Soil Moisture Sensor - SEN-13322 - SparkFun Electronics.* [Online]. Available: https://www.sparkfun.com/products/13322 (visited on 10/14/2024).

[10] *HW5P-1_2015__1_.* [Online]. Available: https://cdn-shop.adafruit.com/product-files/2831/HW5P-1_2015__1_.pdf (visited on 11/28/2024).

[11] SiRF Technology, "NMEA Reference Manual", [Online]. Available: https://www.sparkfun.com/datasheets/GPS/NMEA%20Reference%20Manual-Rev2.1-Dec07.pdf.

[12] *TCS3472_I2C - Class which provides functions to control a TAO... — Mbed.* [Online]. Available: https://os.mbed.com/users/karlmaxwell67/code/TCS3472_I2C/ (visited on 11/28/2024).

[13] *MMA8451 - A driver for the MMA8451 3-axis Accelerometer. — Mbed.* [Online]. Available: https://os.mbed.com/users/wilminc/code/MMA8451/ (visited on 11/28/2024).

[14] *Doxygen: Getting started.* [Online]. Available: https://www.doxygen.nl/manual/starting.html (visited on 11/30/2024).

[15] jothepro, *Jothepro/doxygen-awesome-css*, Nov. 2024. [Online]. Available: https://github.com/jothepro/doxygen-awesome-css (visited on 11/30/2024).

11 APPENDIX: Control message structures

```
#ifndef CONTROL_H
#define CONTROL_H
#include "mbed.h"
#include <cstdint>

/**
* @file control.h
*
* @brief Control thread definition module.
*
* @author Diego Aceituno Seoane
*
* Module that defines the required elements to communicate the threads
    and relevant information for signals.
*
*/

#define QUEUE_SIZE 10

#define I2C_SIGNAL 1U
#define GPS_SIGNAL 2U

/**
* @brief Enumeration for the entering msg to the main thread.
*/
typedef enum{
    ACCELEROMETER,
    GPS,
    TEMP_HUM,
    COLOR
}msg_type;


/**
* @brief Structure for the payload of the accelerometer msg.
*/
typedef struct{
    float x_acc;
    float y_acc;
    float z_acc;
}accelerometer_msg_t;

/**
* @brief Structure for the payload of the gps msg
*/
typedef struct{
    uint8_t sats;
    uint8_t time_h;
    uint8_t time_m;
    uint8_t time_s;
```

```
50      uint8_t time_day;
51      uint8_t time_month;
52      uint16_t time_year;
53      uint16_t altitude;
54      char altitude_c;
55      char lat_n;
56      char lng_w;
57      float lat;
58      float lng;
59  }gps_msg_t;
60
61  /**
62  * @brief Structure for the payload of the temperature and humidity msg
        .
63  */
64  typedef struct{
65      float temp;
66      float hum;
67  }temp_hum_msg_t;
68
69  /**
70  * @brief Structure for the payload of the color sensor msg.
71  */
72  typedef struct{
73      uint16_t clear;
74      uint16_t red;
75      uint16_t green;
76      uint16_t blue;
77  }color_msg_t;
78
79  /**
80  * @brief Structure for the msg entering the main thread.
81  */
82  typedef struct{
83      msg_type type;
84      union{
85          accelerometer_msg_t accelerometer_msg;
86          gps_msg_t           gps_msg;
87          temp_hum_msg_t      temp_hum_msg;
88          color_msg_t         color_msg;
89      };
90  }ctrl_msg;
91
92
93  extern Queue<ctrl_msg, QUEUE_SIZE> ctrl_in_queue;
94
95  #endif
```

Listing 1: Control.h